อัลกอริทึม การจัดกลุ่มงานและลำดับการทำงาน สำหรับสถาปัตยกรรมแบบกระจายหน่วยความจำ โดยใช้ DIRECTED ACYCLIC TASK GRAPHS

TASK CLUSTERING AND SCHEDULING ALGORITHMS FOR DISTRIBUTED MEMORY ARCHITECTURE BY USING DIRECTED ACYCLIC TASK GRAPHS

ศักดิ์ชัย คันธะขจร

SAKCHAI KANTAKAJORN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2543

ISBN 974-622-912-5

อัลกอริทึม การจัดกลุ่มงานและลำดับการทำงาน สำหรับสถาปัตยกรรมแบบกระจายหน่วยความจำ โดยใช้ DIRECTED ACYCLIC TASK GRAPHS

TASK CLUSTERING AND SCHEDULING ALGORITHMS FOR DISTRIBUTED MEMORY ARCHITECTURE BY USING DIRECTED ACYCLIC TASK GRAPHS

ศักดิ์ชัย คันธะขจร

SAKCHAI KANTAKAJORN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2543

ISBN 974-622-912-5

TASK CLUSTERING AND SCHEDULING ALGORITHMS FOR DISTRIBUTED MEMORY ARCHITECTURE BY USING DIRECTED ACYCLIC TASK GRAPHS

SAKCHAI KANTAKAJORN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2000

ISBN 974-622-912-5

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ อัลกอริทึม การจัดกลุ่มงานและลำดับการทำงาน สำหรับสถาปัตยกรรมแบบกระจายหน่วยความจำโดยใช้ DIRECTED ACYCLIC TASK GRAPHS
TASK CLUSTERING AND SCHEDULING ALGORITHMS FOR DISTRIBUTED MEMORY ARCHITECTURE BY USING DIRECTED ACYCLIC TASK GRAPHS

ชื่อนักศึกษา นายศักดิ์ชัย คันธะขจร

รหัสประจำตัว 40061063

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมไฟฟ้า

อาจารย์ผู้ควบคุมวิทยานิพนธ์ ผศ.บรรจง ปียธำรง

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ดร.วิศิษฎ์	หิรัญกิตติ	
รศ.ดร.ชม	กิ้มปาน	
ดร.ชุติเมษฎ์	ศรีนิลทา	
ผศ.ดร.บุญธีร์	เครือตราชู	
ผศ.บรรจง	ปียธำรง	

วัน/เดือน/ปี ที่สอบ 4 กรกฎาคม 2543 **เวลา** 12.00-13.00 น.

สถานที่สอบ ณ ห้องสอบวิทยานิพนธ์ คณะวิศวกรรมศาสตร์ ตึก 12 ชั้น ชั้น 4 ห้อง (E12-403)

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.มนัส สังวรศิลป์)

คณบดีบัณฑิตวิทยาลัย

วันที่ 26 **เดือน** กรกฎาคม **พ.ศ.** 2543

หัวข้อวิทยานิพนธ์	อัลกอริทึม การจัดกลุ่มงานและลำดับการทำงาน สำหรับสถาปัตยกรรมแบบกระจายหน่วยความจำ โดยใช้ Directed Acyclic Task Graphs
นักศึกษา	นาย ศักดิ์ชัย คันธะขจร
รหัสประจำตัว	40061063
ปริญญา	วิศวกรรมศาสตร์มหาบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2543
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ผศ. บรรจง ปิยธำรง

บทคัดย่อ

งานวิทยานิพนธ์นี้มีเป้าหมายเพื่อแสดงอัลกอริทึม ในการจัดการเกี่ยวกับกลุ่มงานของโปรแกรมแบบขนาน บนระบบเครือข่ายและลำดับการทำงานของกลุ่มงานโดยการแบ่งโหนด โดยจะใช้ DAG(Directed Acyclic Task Graphs)แสดงการทำงานของโปรแกรมแบบขนานที่มีลักษณะการกระจายหน่วยความจำ โดยอัลกอริทึมมีชื่อว่า ATCS อัลกอริทึม(Adaptive Task Clustering and Scheduling Algorithms) ATCS อัลกอริทึมได้แสดงวิธีการจัดกลุ่มงาน โดยกลุ่มงานจะกระจายอยู่บนระบบเครือข่าย และแต่ละกลุ่มงานจะทำงานอยู่บนหนึ่งตัวประมวลผล เพื่อลดปัญหาเกี่ยวกับโอเวอร์เฮดของค่าใช้จ่ายการติดต่อสื่อสาร(Communication Cost)ระหว่างตัวประมวลผลให้น้อยลง เพราะบนระบบเครือข่ายที่มีโอเวอร์เฮดมากๆจะทำให้ระบบของงานทำงานได้ช้าลง โดยเฉพาะงานของโปรแกรมแบบขนานบนระบบเครือข่ายที่มีลักษณะเม็ดเนื้องานแบบละเอียด(a Fine Grain Tasks) เม็ดเนื้องานแบบละเอียดจะมีโอเวอร์เฮดเพราะมีค่าใช้จ่ายในการติดต่อสื่อสารระหว่างตัวประมวลผลมาก ATCS อัลกอริทึมจะพยายามลดโอเวอร์เฮดที่เกิดขึ้นให้น้อยลง โดยจะทำการแปลงลักษณะเม็ดเนื้องานแบบละเอียดให้เป็นลักษณะเม็ดเนื้องานแบบหยาบ(a Coarse Grain Taks) ทำให้จำนวนตัวประมวลผลที่ใช้จากเดิมลดน้อยลง วิทยานิพนธ์นี้จะใช้แบร็คแทรกกิ้งอัลกอริทึม(Back-Tracking Algorithm) ในการจัดกลุ่มงานจาก DAG ที่ถูกแปลงให้อยู่ในรูปแบบของต้นไม้กลับตัวชี้ โดย ATCS อัลกอริทึมจะมีค่าความสลับซับซ้อนของเวลาในการทำงานของอัลกอริทึมเท่ากับ $O(|V^2|)$ เมื่อ V คือจำนวนของโหนดที่ปรากฏอยู่บน DAG

Thesis Title	Task Clustering and Scheduling Algorithms for Distributed Memory Architecture by using Directed Acyclic Task Graphs
Student	Mr. Sakchai Kantakajorn
Student ID.	40061063
Degree	Master of Engineering in Electrical Engineering
Programe	Electrical Engineering
Year	2000
Thesis Advisor	Assistant Prof. Banjong Piyatamrong

ABSTRACT

This thesis presents a task-clustering algorithm named Adaptive Task Clustering and Scheduling Algorithms (ATCS). This algorithm represents tasks of a parallel program as nodes and scheduling them with Directed Acyclic Task Graph or DAG. The parallel program focused in this thesis is that has a feature of distributed memory architecture. Adaptive Task Clustering and Scheduling Algorithms implements the task-clustering method by partitioning nodes on a DAG. Each group or cluster within a DAG is placed on only one processor. The algorithm partitions the DAG as several clusters to reduce an overhead communication cost between processors. Because the network has a lot of overhead communication cost that increases total execution time or makespan of the tasks, especially the parallel program that is a fine grain task on network. The fine grain task has a high communication cost because of communication time between processors. ATCS algorithm tries to reduce an overhead communication cost by transforming a fine grain task to a coarse grain task. So it reduces the number of processors and the communication cost consequently. ATCS uses a backtracking algorithm to transform a DAG to an inverse pointer binary-tree graph or an In-tree, and then group the tasks into clusters. ATCS has a time complexity of algorithm equals to $O(|V^2|)$, where V is a number of nodes on a DAG.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้อย่างดี ด้วยคำแนะนำและคำปรึกษาเกี่ยวกับวิธีการทำงานของตัวประมวลผลแบบขนาน และวิธีการประมวลผลของงานในระบบแบบกระจายหน่วยความจำ จาก ผศ. บรรจง ปิยธำรง ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ผู้วิจัยรู้สึกซาบซึ้งในความอนุเคราะห์จากท่านและขอกราบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ ผศ. ดร. บุญธีร์ เครือตราชู ภาควิชาวิศวกรรมคอมพิวเตอร์ ที่ช่วยเหลือแก้ไขและให้คำแนะนำในบางจุดที่ผู้วิจัยบกพร่องบางอย่าง ซึ่งมีส่วนช่วยทำให้ผู้วิจัยเข้าใจในปัญหานั้นได้ดียิ่งขึ้น และทำให้วิทยานิพนธ์ฉบับสมบูรณ์มากยิ่งขึ้น

สุดท้ายนี้ขอขอบพระคุณ รศ. ดร. กิตติ ไพฑูรย์วัฒนกิจ ภาควิชาวิศวกรรมอิเลคทรอนิคส์ ที่ให้คำแนะนำ เป็นที่ปรึกษา และกำลังใจ ในระหว่างที่ศึกษาอยู่ตลอดมา จนจบการศึกษา

คุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

ศักดิ์ชัย คันธะขจร

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

สารบัญตาราง

นิยามคำศัพท์เฉพาะ

Directed Acyclic Task Graphs / DAG	หมายถึง กราฟงานที่ไม่มีทิศทางเป็นลูป (Loop)
Edge	หมายถึง เส้นทางการเชื่อมต่อระหว่าง โหนด(Node) , ตัวประมวลผล, หรืองาน ที่อยู่บน DAG
Directed Edge	หมายถึง Edge ที่มีทิศทาง
Task Graph	หมายถึง รูปแบบที่ใช้แสดงลำดับการทำงานของงานต่างๆที่อยู่บนโปรแกรมแบบขนาน และลำดับก่อนหลังมีความสัมพันธ์ต่อลำดับการทำงานของงาน
Grain	หมายถึง ค่าอัตราส่วนระหว่างค่าใช้จ่ายที่ใช้ในการประมวลผลของงาน ต่อค่าใช้จ่ายในการติดต่อสื่อสารระหว่างงานหรือตัวประมวลผล
Task Granularity	หมายถึง ค่าของเม็ดเนื้องานในระดับโกลบอล (Global)
Task Grain	หมายถึง ค่าของเม็ดเนื้องานในระดับโลคัล (Local)
Task Duplicate	หมายถึง ชุดของกลุ่มงานที่มีงานซ้ำซ้อนปรากฏขึ้นในระหว่างชุดของกลุ่มงานมากกว่าหนึ่งกลุ่มงาน
Non Task Duplicate	หมายถึง ชุดของกลุ่มงานที่ไม่มีงานซ้ำซ้อนเกิดขึ้นในระหว่างชุดของกลุ่มงานเหล่านั้น
Task Scheduling	หมายถึง การจัดลำดับการทำงานของงานต่างๆที่อยู่บนกราฟงานหรือ กลุ่มงาน
Task Clustering	หมายถึง กลุ่มงานที่ประกอบไปด้วยงานมากกว่าหนึ่งงาน
Distributed Memory Architecture	หมายถึง สถาปัตยกรรมคอมพิวเตอร์ที่มีหน่วยความจำอาศัยอยู่บนเครื่องคอมพิวเตอร์เครื่องๆนั้น โดยมิได้แบ่งให้เครื่องคอมพิวเตอร์เครื่องอื่นเข้ามาร่วมใช้
Parallel Processing	หมายถึง การบวนการในการประมวลผลงานพร้อมๆกันมากกว่าหนึ่งงาน
Partitioning	หมายถึง การแบ่งงานต่างๆที่อยู่บนกราฟงานหรือ DAG ออกเป็นกลุ่มๆ

Makespan	หมายถึง เวลาที่ใช้ในการประมวลผลของงานต่างๆที่อยู่ในระบบงานจนเสร็จสิ้น หรือเส้นทางวิกฤต(Critical Path)

บทที่ 1
บทนำ

1.1 ความเป็นมาของปัญหา

ลำดับการทำงานของโปรแกรมแบบขนาน(Parallel Program) ได้ถูกจัดให้เป็นสาขาวิชาหนึ่งที่น่าสนใจ โดยเฉพาะผลลัพธ์ที่ได้จากการจัดลำดับงาน(Schedule Task) หรือลำดับงาน มีผลเกี่ยวเนื่องมาจากการจัดลำดับของงาน(Task)หรือกระบวนการ(Process) ต่างๆที่อยู่บนโปรแกรมแบบขนาน ปัญหาของการจัดลำดับการทำงานได้เกิดขึ้น เมื่อมีวิธีให้เลือกทำต่อลำดับของจำนวนงานต่างๆที่สามารถถูกกระทำได้ และสามารถที่จะกำหนดงานเหล่านั้นเพื่อให้เครื่องคอมพิวเตอร์ทำการประมวลผล(Processing)กับกระบวนการ ได้อย่างมีประสิทธิภาพ ปัญหาของการจัดลำดับของงานอาจจะพบได้จาก การวางแผนการผลิต ระบบการให้บริการแถวคอยฝากถอนที่เคาน์เตอร์ของธนาคาร และงานต่างๆที่อยู่บนโปรแกรมที่จะถูกประมวลผลด้วยบนระบบคอมพิวเตอร์แบบขนานหรือแบบกระจาย (Parallel or Distributed Computer System) ในยุคของการคำนวณแบบขนานและแบบกระจาย ปัญหาของการจัดลำดับการทำงานของงานที่อยู่บนโปรแกรมแบบขนานทำให้ได้รับพิจารณาจากนักวิจัย เมื่อโปรแกรมหนึ่งโปรแกรมสามารถพิจารณาเหมือนกับกลุ่มของงานที่จะทำงานในลักษณะเป็นแบบเรียงตามลำดับหรือแบบขนาน เป้าหมายของการจัดลำดับการทำงานของโปรแกรมแบบขนาน เพื่อให้งานถูกทำการประมวลผลและลำดับของงานที่จะถูกประมวลผลได้อย่างเหมาะสม (Optimize) การจัดลำดับการทำงานที่เหมาะสมจะพิจารณาทั้งการจัดแบ่งงานและลำดับของการประมวลผลของงานแต่ละงาน ถ้ากลุ่มงานของโปรแกรมไม่มีความสัมพันธ์ของลำดับก่อนหลัง(Precedance Relation) ในการประมวลผลของกลุ่มงานเหล่านั้น ปัญหาของการจัดลำดับการทำงานของกลุ่มงานประเภทนี้ถูกเรียกว่าปัญหาของการจัดแบ่งงาน (Task Allocation) ปัญหาของการจัดลำดับการทำงานเป็นปัญหาที่มีความท้าทายโดยเฉพาะอย่างยิ่งในการคำนวณแบบขนานและแบบกระจาย และปัญหาของการจัดลำดับการทำงานจัดให้เป็นปัญหาแบบ NP-complete ทั้งนี้เนื่องมาจากข้อจำกัดหลายๆด้านในการจัดการกับปัญหา นักวิจัยได้กำหนดขอบเขตรูปแบบของการศึกษาของปัญหาโดยการบังคับอยู่ในรูปแบบของกราฟงาน (Task Graph) ซึ่งเป็นการแสดงของโปรแกรมแบบขนานหรือรูปแบบของโมเดลของระบบคอมพิวเตอร์ ปัญหาการจัดลำดับการทำงานสามารถถูกแก้ไขปัญหาได้ด้วยอัลกอริทึมต่างๆ ฮิวลิสติก (Heuristics) อัลกอริทึมถูกจัดให้เป็นวิธีการแก้ไขปัญหาการจัดลำดับการทำงานของโปรแกรมแบบขนานวิธีหนึ่งได้อย่างมีประสิทธิภาพ แต่ฮิวลิสติกอัลกอริทึมไม่สามารถทำงานแก้ไขปัญหาการจัดลำดับการทำงานได้ทุกกรณี บางกรณี

ฮิวลิสติกอัลกอริทึมสามารถจัดการลำดับการทำงานของโปรแกรมแบบขนานได้อย่างมีประสิทธิภาพ แต่บางกรณีฮิวลิสติกอัลกอริทึม(Heuristics Algorithm) ไม่สามารถทำงานได้อย่างมีประสิทธิภาพ ฮิวลิสติกอัลกอริทึมไม่สามารถรับรองผลได้ว่าจะสามารถทำการแก้ไขปัญหาแล้วให้ค่าอย่างเหมาะสม แต่ฮิวลิสติกอัลกอริทึมพยายามที่จะแก้ไขปัญหาให้ค่าที่ใกล้เคียงกับค่าที่เหมาะสมมากที่สุด

1.2 วัตถุประสงค์ในการวิจัย

ในการทำงานของโปรแกรมแบบขนาน อย่างเช่น การพยากรณ์อากาศ, การคำนวณหาค่าทางคณิตศาสตร์ เป็นต้น มักต้องการเครื่องคอมพิวเตอร์ที่มีความเร็วสูงในการประมวลผล เครื่องคอมพิวเตอร์ที่มีความเร็วสูงในการประมวลนั้นโดยส่วนใหญ่แล้วจะเป็นเครื่องคอมพิวเตอร์ที่มีตัวประมวลผล(Processor) มากกว่าหนึ่งตัว หรือเรียกว่าตัวประมวลผลแบบขนาน (Parallel Processor) เครื่องคอมพิวเตอร์ประเภทนี้จะมีราคา(Price)และค่าบำรุงรักษา(Maintenance)ที่สูงมาก ทำให้บางหน่วยงานไม่สามารถซื้อเครื่องคอมพิวเตอร์ประเภทนี้มาใช้เพื่อการทำงานได้ ในปัจจุบันนี้ได้มีการพัฒนาระบบเครือข่าย(Local Area Network) คอมพิวเตอร์ที่สามารถทำงานบนโปรแกรมแบบขนานได้ โดยการนำเอาเครื่องพีซีหลายๆเครื่องมาต่อเข้าด้วยกันเป็นระบบเครือข่ายหรือเรียกว่าคลัสเตอร์ (Cluster) ดังนั้นในหนึ่งคลัสเตอร์จะต้องประกอบไปด้วยตัวประมวลผลหลายๆตัว เปรียบเสมือนเครื่องคอมพิวเตอร์หนึ่งเครื่องที่มีตัวประมวลผลหลายๆตัว ซึ่งในหนึ่งคลัสเตอร์เราสามารถทำการประมวลผลโปรแกรมแบบขนานได้อย่างมีประสิทธิภาพ แต่ในการประมวลผลของโปรแกรมแบบขนานได้อย่างมีประสิทธิภาพนั้น จะต้องมีจำนวนของตัวประมวลผลเท่ากับจำนวนของงานในโปรแกรมแบบขนานนั้น ซึ่งเป็นการสิ้นเปลืองค่าใช้จ่ายในการที่จะต้องหาเครื่องคอมพิวเตอร์มาใช้ให้พอดีเท่ากับจำนวนของงานบนโปรแกรมแบบขนานนั้น ดังนั้นในการที่จะนำเอาเครื่องคอมพิวเตอร์ที่มีอยู่มาใช้ให้เกิดประโยชน์สูงสุด วิธีการแก้ไขปัญหาโดยการจัดงานต่างๆที่อยู่บนโปรแกรมแบบขนานออกเป็นกลุ่มๆ หนึ่งกลุ่มงานก็จะทำงานอยู่บนเครื่องคอมพิวเตอร์หนึ่งเครื่องหรือหนึ่งตัวประมวลผล แต่ในการจัดกลุ่มงานของโปรแกรมแบบขนานนั้นจะต้องอาศัยอัลกอริทึมเข้ามาช่วยในการจัดกลุ่มงานของโปรแกรมแบบขนานนั้น เพื่อให้เกิดกลุ่มงานที่สามารถทำงานอยู่บนคลัสเตอร์ได้อย่างมีประสิทธิภาพ อย่างเช่นจะต้องใช้เวลาในการประมวลผลของโปรแกรมแบบขนานที่อยู่บนคลัสเตอร์นั้นให้สั้นที่สุด เป็นต้น

วิทยานิพนธ์ฉบับนี้มีวัตถุประสงค์เพื่อที่จะสร้างอัลกอริทึมที่ชื่อว่า เอทีซีเอส อัลกอริทึม(ATCS Algorithm)เพื่อใช้ในการจัดกลุ่มงานและกำหนดลำดับการทำงาน ให้กับงานต่างๆที่อยู่บน DAG(Directed Acyclic Task Graphs) หรือโปรแกรมแบบขนาน โดยกลุ่มงานที่ได้จาก เอทีซีเอส อัลกอริทึมจะมีลักษณะที่มีงานซ้ำซ้อนเกิดขึ้นในระหว่างกลุ่มงาน

1.3 ขอบเขตการวิจัย

ขอบเขตการวิจัยของวิทยานิพนธ์ฉบับนี้ โดยมีเป้าหมายหลักอยู่ที่การใช้เอทีซีเอสอัลกอริทึมในการจัดกลุ่มงานและกำหนดลำดับการทำงานให้กับโปรแกรมแบบขนานหรือ DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด(A Fine Grain Tasks)เป็นหลัก โดยจะทำการจำลอง(Simulate) โปรแกรมแบบขนานหรือ DAG ในลักษณะดังกล่าวขึ้นมา ดังนั้นในการทำงานหาผลลัพธ์ต่างๆของวิทยานิพนธ์ฉบับนี้ จะได้มาจากการจำลองแทนการทำงานที่อยู่บนระบบเครือข่าย(Network)จริงๆ ดังนั้นวิทยานิพนธ์ฉบับนี้จะทำการสร้างโปรแกรมขึ้นมา เพื่อใช้ในการทดลองหาผลลัพธ์จากการทำงานของเอทีซีเอสอัลกอริทึม โดยโปรแกรมที่สร้างขึ้นมามีชื่อว่า ATCS (ซอสโค้ดของ โปรแกรม ATCS ปรากฏอยู่ใน ภาคผนวก ก.) โปรแกรม ATCS จะทำการสร้าง DAG หรือโปรแกรมแบบขนานขึ้นมา ในการจำลองสร้าง DAG และหาผลลัพธ์ที่ได้มาจากการสร้างกลุ่มงานและกำหนดลำดับการทำงานให้กับโปรแกรมแบบขนาน จะทำการจำลองอยู่บนเครื่องไมโครคอมพิวเตอร์ Pentium II 233 Mhz ด้วยการใช้ Microsoft Visual C++ version 6.0 ในการสร้างโปรแกรม ATCS

โดยทั่วๆไปแล้ววิธีการในการจัดกลุ่มงานและกำหนดลำดับการทำงานให้กับโปรแกรมแบบขนานจะถูกแบ่งออกเป็นสองประเภทหลักๆ ด้วยกันคือ แบบ Deterministic(Static) และแบบ Non-Deterministic (รายละเอียดจะถูกกล่าวอยู่ในบทที่ 2 ต่อไป) วิธีการของเอทีซีเอสอัลกอริทึมถูกจัดอยู่ในแบบ Deterministic และหลักการทำงานของเอทีซีเอสอัลกอริทึมเป็นแบบฮิวริสติกอัลกอริทึม โดยอาศัยความสัมพันธ์ของลำดับก่อนและหลังระหว่างงานต่างๆที่อยู่บน DAG เป็นหลักในการทำงาน

ในการจำลองสร้าง DAG หรือโปรแกรมแบบขนานขึ้นมา ลักษณะของ DAG มีสองลักษณะด้วยกัน โดย DAG แบบแรกเป็น DAG ที่ได้มาจากการป้อนข้อมูลของผู้ใช้(Users) ผู้ใช้จะต้องใส่ข้อมูลต่างๆที่เกี่ยวกับ DAG ที่ต้องการจะหาผลลัพธ์ เช่น ค่าใช้จ่ายในการติดต่อสื่อสารระหว่างตัวประมวลผล(Communication Cost) ค่าใช้จ่ายในการประมวลผลที่ตัวประมวลผลต่างๆ(Execution Cost) ระดับ(Level)ของโหนดต่างๆที่อยู่บน DAG เป็นต้น และ DAG แบบที่สองเป็น DAG ที่ได้มาจากการสุ่ม(Random)สร้างของโปรแกรม โดยผู้ใช้จะต้องใส่ข้อมูลต่างๆที่คล้ายกับแบบแรก(รายละเอียดต่างๆ จะกล่าวอยู่ในบทที่ 5)

ในการจัดกลุ่มงานและลำดับการทำงานของโปรแกรมแบบขนาน ซึ่งถูกแสดงอยู่ในรูปของกราฟงานหรือ DAG โดยโปรแกรมแบบขนานนั้นจะถูกประมวลผลอยู่บนระบบคอมพิวเตอร์ที่มีลักษณะสถาปัตยกรรมแบบกระจายหน่วยความจำ โดยเอทีซีเอสอัลกอริทึมจะให้กลุ่มงานและกำหนดลำดับการทำงานของกลุ่มงานเหล่านั้น โดยอาศัยข้อมูลต่างๆที่อยู่บน DAG นั้น เช่น ค่าใช้จ่ายที่ใช้ในการประมวลผลของงานๆนั้น ค่าใช้จ่ายที่ใช้ในการติดต่อสื่อสารระหว่างงาน(Tasks)หรือ

ตัวประมวลผล(Processors) เป็นเครื่องมือใช้ในการสร้างกลุ่มงานและลำดับการทำงานของกลุ่มงานเหล่านั้น

เอทีซีเอสอัลกอริทึมใช้แบล็คแทรกกิ้งอัลกอริทึม(Back-Tracking Algorithm)เข้ามาช่วยประยุกต์ใช้ในการทำงาน เพื่อให้ได้ชุดของกลุ่มงานที่เหมาะสมและช่วยลดเวลาที่ใช้ในการประมวลของโปรแกรมแบบขนานให้สั้นที่สุด โดยลักษณะของกลุ่มงานที่ได้จากเอทีซีเอสอัลกอริทึมจะมีลักษณะที่มีงานซ้ำซ้อนเกิดขึ้นในระหว่างกลุ่มงาน ขณะเดียวกันเอทีซีเอสอัลกอริทึมยังช่วยลดช่วงเวลาที่หยุดนิ่ง (Idle Time) บนตัวประมวลผลให้ลดน้อยลงด้วย โดยวิธีการของเอทีซีเอสอัลกอริทึมจะอยู่ภายใต้ข้อกำหนดที่ว่าเครื่องคอมพิวเตอร์ทั้งหมดที่อยู่ในระบบเครือข่ายจะต้องมีประสิทธิความเร็วในการประมวลผลเท่ากันหมด เช่น ในระบบเครือข่ายที่ตั้งขึ้นเพื่อใช้ในการประมวลผลโปรแกรมแบบขนาน เครื่องคอมพิวเตอร์ทุกเครื่องจะมีตัวประมวลผลแบบ Pentium II 200 Mhz เหมือนกันหมดทั้งระบบเครือข่าย หรือความเร็ว(Speed)ของตัวประมวลผลจะต้องเท่ากันทั้งหมด เป็นต้น

รายละเอียดของวิทยานิพนธ์ฉบับนี้จึงแยกออกเป็นส่วนๆ คือ จะกล่าวถึงความรู้พื้นฐานเกี่ยวกับลำดับการทำงานของโปรแกรมแบบขนาน ประเภทของลำดับการทำงาน การจัดแบ่งประเภทของลำดับการทำงาน รูปแบบของลำดับการทำงาน อัลกอริทึมที่ใช้ในการจัดกลุ่มงานและลำดับการทำงานของโปรแกรมแบบขนานในแบบต่างๆกัน และขั้นตอนการทำงานของเอทีซีเอสอัลกอริทึม ผลลัพธ์ที่ได้จากเอทีซีเอสอัลกอริทึม การเปรียบเทียบผลลัพธ์ที่ได้จากเอทีซีเอสอัลกอริทึมกับอัลกอริทึมของงานวิจัยอื่นๆ และสรุปผลการทดลองที่ได้จากเอทีซีเอสอัลกอริทึม

สำหรับวิทยานิพนธ์ฉบับนี้จะใช้คำว่ากระบวนการ(Process) และคำว่างาน(Task) ในความหมายเดียวกัน ขึ้นอยู่กับโอกาสในการใช้ภายในประโยคที่เหมาะสม

บทที่ 2
ความรู้พื้นฐานเกี่ยวกับ ลำดับการทำงานของโปรแกรมแบบขนาน

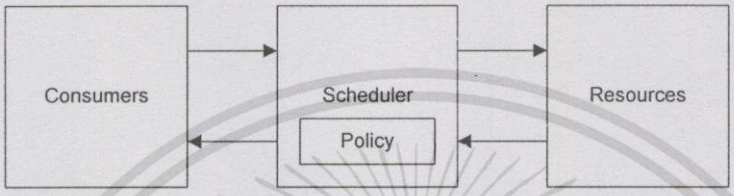
2.1 ปัญหาของลำดับการทำงาน

ปัญหาของการจัดลำดับการทำงานได้ถูกอธิบายในวิธีที่แตกต่างกันออกไปตามสาขาวิชาต่างๆ เช่น ปัญหาของการเรียงลำดับงานในการควบคุมการผลิตมีผลอย่างมากต่อขั้นตอนในการผลิตสินค้า โดยส่วนใหญ่ในขบวนการผลิตจะเกี่ยวข้องกับขบวนการแปรรูปจากวัตถุดิบให้ออกมาเป็นตัวสินค้า ซึ่งปัญหาในการพิจารณาลำดับขบวนการขั้นตอนการทำงานมักจะเกี่ยวข้องกับทางด้านเศรษฐศาสตร์ ปัญหาที่พบและถูกอ้างถึงคือการเรียงลำดับขั้นตอนการทำงาน หลายปีที่ผ่านมามีอยู่หลายๆวิธีด้วยกันที่ถูกนำมาใช้ในการแก้ไขปัญหา อย่างเช่น Complete Enumeration, Heuristics Rules, Integer Programming, และ Sampling Methods เป็นต้น วิธีการ Complete Enumeration จัดได้ว่าเป็นวิธีการหนึ่งที่ดีใช้ช่วยการแก้ไขปัญหาได้ แต่ในทางปฏิบัติแล้วไม่สามารถช่วยแก้ไขปัญหาได้ ในทางปฏิบัติมักจะใช้ Heuristics Rules เข้ามาช่วยในการแก้ไขปัญหาได้เป็นส่วนใหญ่

โดยทั่วๆไปแล้วปัญหาของลำดับงานหรือการทำงาน มักจะสมมติว่าประกอบไปด้วยจำนวนของทรัพยากร (Resource) และกลุ่มของผู้บริโภค (Consumer) โดยมีการกำหนดนโยบายที่แน่นอนในการบริการการใช้ทรัพยากรเหล่านั้นต่อกลุ่มผู้บริโภค โดยพื้นฐานของธรรมชาติด้วยข้อจำกัดอยู่บนกลุ่มของผู้บริโภคและจำนวนของทรัพยากรที่มีอยู่ ปัญหาคือต้องการหานโยบายที่มีสมรรถนะ (Performance) เพื่อควบคุมการใช้ทรัพยากรเหล่านั้นอย่างเหมาะสมที่สุด โดยวัดจากสมรรถนะของความยาวของการจัดลำดับ (Schedule Length) ในระบบของการจัดลำดับ (Schedule System) จะประกอบไปด้วยกลุ่มของผู้บริโภค จำนวนของทรัพยากร และนโยบายของการจัดลำดับ (Schedule Policy) ดังแสดงอยู่ในรูปที่ 2.1 งานในโปรแกรม งานในระบบการผลิต และลูกค้าของธนาคาร เป็นตัวอย่างของผู้ใช้บริการ ขบวนการในการทำงาน เครื่องจักรในโรงงาน และพนักงานเคาน์เตอร์ที่ทำหน้ารับฝากเงินของธนาคาร ก็เป็นตัวอย่างหนึ่งของการใช้ทรัพยากร และFirst-come-first-served เป็นตัวอย่างหนึ่งในนโยบายของการจัดลำดับ สมรรถนะของนโยบายของการจัดลำดับขึ้นอยู่กับปัจจัยหลายๆด้าน First-come-first-served อาจจะเหมาะสมสำหรับระบบธนาคาร ในขณะเดียวกัน First-come-first-served อาจจะไม่เหมาะสำหรับขบวนขั้นตอนในการผลิตของโรงงาน

สมรรถนะและประสิทธิภาพ (Efficiency) คือสองลักษณะที่ใช้ในการประเมินระบบการจัดลำดับงาน อย่างเช่นถ้าเราต้องการประเมินระบบการจัดลำดับงานอยู่บนพื้นฐานของความต้องการที่จะสร้างระบบการจัดลำดับงานและประสิทธิภาพของนโยบาย หรืออาจจะกล่าวได้ว่าเรา

ต้องการหาวิธีการที่จะสร้างระบบการจัดลำดับงานหรือลำดับการทำงานที่ดีได้อย่างไรและประสิทธิภาพของผู้จัดลำดับงาน (Scheduler) เอง ซึ่งการสร้างระบบการจัดลำดับงานถูกกำหนดอยู่บนพื้นฐานของการกำหนดมาตรฐานหรือเกณฑ์ที่พยายามให้ได้ผลลัพธ์ที่เหมาะสมที่สุด ตัวอย่างเช่นเราต้องการหาค่าของเวลาที่เสร็จสิ้นสมบูรณ์ของโปรแกรมที่เหมาะสมที่สุด ถ้าเวลาที่เสร็จสิ้นสมบูรณ์ใช้ไปน้อยแสดงว่าการสร้างระบบการจัดลำดับนั้นดี ในสาขาวิชาคอมพิวเตอร์นั้นนโยบายหรือผู้จัดลำดับสามารถถูกประเมินอยู่บนพื้นฐานของค่าความสลับซับซ้อนของเวลาของการจัดลำดับงาน



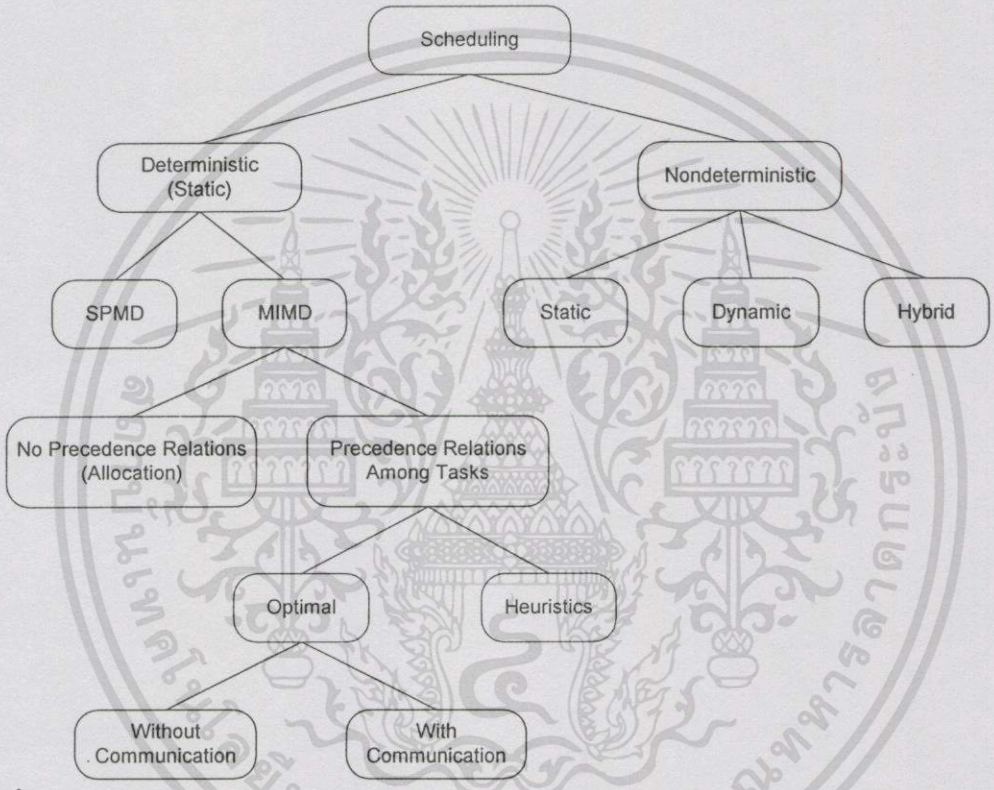
รูปที่ 2.1 แสดงระบบการจัดลำดับการทำงาน

สำหรับวิทยานิพนธ์ฉบับนี้เราจะเกี่ยวข้องกับการจัดลำดับงานหรือลำดับการทำงานของโปรแกรมที่อยู่บนระบบแบบขนานและแบบกระจาย งานของโปรแกรมคือผู้บริโภคซึ่งจะถูกแสดงในรูปของ DAG หรือกราฟงาน ในขณะเดียวกันส่วนประกอบในขบวนการทำงานคือทรัพยากรซึ่งถูกแสดงในรูปของค่าใช้จ่ายที่ใช้ในการประมวลผลของงานที่ตัวประมวลผลและค่าใช้จ่ายที่ใช้ในการติดต่อสื่อสารระหว่างงานหรือตัวประมวลผล ในการจัดลำดับงานจะใช้ทามมิ่งไดอะแกรม (Timing Diagram) ที่เรียกว่าแกทชาร์ท (Gantt Chart) ในการแสดงการแบ่งงานของโปรแกรมแบบขนานลงบนแต่ละตัวประมวลผลของเครื่องเป้าหมาย (Target Machines) แต่ละเครื่อง รายการของงานทั้งหมดที่ถูกกำหนดแบ่งลงบนตัวประมวลผล โดยเรียงลำดับตามเวลาที่ใช้ในการประมวลผลรวมทั้งเวลาเริ่มต้นและเวลาที่สิ้นสุดของงานแต่ละงาน คำว่างาน หรือโหนดบนกราฟงานจะถูกพิจารณาเสมือนกับผู้บริโภค ส่วนตัวประมวลผลจะถูกพิจารณาเหมือนกับทรัพยากรที่ถูกใช้ในการประมวลผล

2.2 ประเภทของการจัดลำดับงาน

เทคนิคที่ใช้ในการจัดลำดับงานโดยอาศัยข้อมูลที่อยู่บนโปรแกรมนั้น สามารถถูกแบ่งออกเป็นสองประเภทหลักๆด้วยกัน[1]คือ แบบเป็นไปตามทฤษฎี (Deterministic) และแบบไม่เป็นไปตามทฤษฎี (Non Deterministic) วิธีการจัดลำดับงานแบบเป็นไปตามทฤษฎีจะอาศัยข้อมูลทั้งหมดที่เกี่ยวกับงานมาใช้ในการจัดลำดับงานและข้อมูลอื่นๆที่มีความสัมพันธ์กับงานๆนั้น โดยข้อมูลทั้งหมดเหล่านี้จะต้องถูกรู้ก่อนที่จะเริ่มทำการประมวลผล วิธีการจัดลำดับงานแบบไม่เป็นไป

ตามทฤษฎีจะอาศัยโทโปโลจี (Topology) ของกราฟงานที่ใช้แสดงโปรแกรม ค่าใช้จ่ายในการประมวลผล และค่าใช้จ่ายที่ใช้ในการติดต่อสื่อสารระหว่างงานหรือตัวประมวลผล ซึ่งข้อมูลเหล่านี้ไม่จำเป็นที่จะถูกรู้ก่อนที่จะเริ่มทำการประมวลผล การเกิดทางเลือก (Branches) ของเงื่อนไขและการเกิดลูป (Loop) ของโปรแกรมอาจจะก่อให้เกิดวิธีการจัดลำดับงานแบบไม่เป็นไปตามทฤษฎี สำหรับตัวอย่างเช่นทิศทางของทางเลือกอาจจะยังไม่รู้จนกระทั่งโปรแกรมนั้นได้เริ่มถูกประมวลผลไปแล้วกลางทางจึงจะทราบ หรือแม้แต่การเกิดลูปของโปรแกรมยังไม่รู้ได้จนกว่าโปรแกรมได้ถูกเริ่มต้นทำการประมวลผล เป็นต้น



รูปที่ 2.2 แสดงการจัดแบ่งประเภทของการจัดลำดับการทำงาน

รูปที่ 2.2 แสดงการจัดแบ่งประเภทของการจัดลำดับการทำงาน จากรูปแสดงให้เห็นว่าวิธีการจัดลำดับแบบไม่เป็นไปตามทฤษฎีนั้น สามารถทำได้ด้วยกันสามวิธีคือ แบบสเตติก (Static), แบบไดนามิก (Dynamic), และแบบไฮบริด (Hybrid) เวลาที่ใช้ในการตัดสินใจการจัดลำดับ (Scheduling Decision) ถูกใช้เป็นตัวบ่งบอกของความแตกต่างของวิธีการทั้งสาม การจัดลำดับแบบสเตติกโดยอาศัยข้อมูลบนกราฟงานของโปรแกรมจะถูกใช้ในการประมาณก่อนที่จะมีการประมวลผล สำหรับตัวอย่างเช่นความเป็นไปได้ที่การประมวลผลของทางเลือกของเงื่อนไข ภายใต้ทางเลือกของคำสั่งเงื่อนไขหนึ่งๆ เราสามารถนำมาใช้ในการคาดคะเนได้ว่าทางเลือกถูกใช้ไปใน

เวลาส่วนใหญ่ของการประมวลผล การจัดลำดับแบบไดนามิกที่อยู่ภายใต้ระบบตัวประมวลผลแบบขนาน จะพยายามจัดลำดับการทำงานในระหว่างการทำงาน ดังนั้นการตัดสินใจของการจัดลำดับถูกทำให้เกิดขึ้นในขณะที่กำลังทำการประมวลผลอยู่ โดยปกติแล้ววิธีการนี้จะถูกจัดการด้วยวิธีการของโหลดบารานซิ่งฮิวริสติก (Load Balancing Heuristic) เมื่อผู้จัดลำดับมีเพียงข้อมูลโลคัลเท่านั้นเกี่ยวกับโปรแกรมแบบขนานในขณะนั้น ข้อเสียของการจัดลำดับแบบไดนามิกคือโอเวอร์เฮดที่เกิดจากการพิจารณาการจัดลำดับในขณะที่โปรแกรมกำลังทำการประมวลผลอยู่ การจัดลำดับแบบไฮบริดเป็นการผสมผสานระหว่างวิธีการของสเตติกและไดนามิก โดยกระบวนการประมวลผลบางขั้นตอนก่อนที่จะเริ่มมีการประมวลผลจริง ได้ถูกนำมาจัดลำดับเป็นแบบสเตติก ผลลัพธ์ที่ได้จากกระบวนการในขั้นตอนนี้จะถูกใช้เป็นข้อมูลพื้นฐานในขั้นตอนของการจัดลำดับแบบไดนามิกต่อไปในภายหน้า

ลำจำกัดความสำหรับการจัดลำดับแบบเป็นไปตามทฤษฎี ถูกใช้ในความหมายของการจัดลำดับแบบสเตติก โดยที่ข้อมูลต่างๆบนกราฟงานเช่นค่าใช้จ่ายในการประมวลผลงานที่ตัวประมวลผล และค่าใช้จ่ายของการติดต่อสื่อสารระหว่างตัวประมวลผลหรืองาน ข้อมูลเหล่านี้เป็นข้อมูลเบื้องต้นก่อนที่โปรแกรมจะเริ่มทำการประมวลผล ในกระบวนการทำงานของการจัดลำดับงานเป็นแบบสเตติก งานแต่ละงานบนกราฟงานได้ถูกกำหนดอย่างแน่นอนให้ทำงานที่ตัวประมวลผลเฉพาะ และแต่ละครั้งที่งานถูกให้ทำงานสำหรับการประมวลผล งานๆนั้นจะถูกกำหนดต่อตัวประมวลผล เมื่อตัวประมวลผลทั้งหมดถูกให้ทำการประมวลผลที่โปรแกรมเดียวกันแต่ใช้ข้อมูลที่ต่างกันพร้อมกัน เราเรียกการทำงานอย่างขนานแบบนี้ว่าเอสพีเอ็มดี (Single Program Multiple Data/SPMD หรือ Data Parallel) ภายใต้รูปแบบของเอสพีเอ็มดีการจัดลำดับแบบสเตติกสามารถได้รับประโยชน์จากกราฟงาน แต่ถ้าระบบตัวประมวลผลใช้ในการประมวลผลด้วยงานที่แตกต่างกันและชุดข้อมูลที่ใช้ในการประมวลต่างกันด้วย เราจะเรียกการทำงานขนานแบบนี้ว่าเอ็มไอเอ็มดี (Multiple Instruction Multiple Data/MIMD) การทำงานแบบเอ็มไอเอ็มดีเป็นรูปแบบที่ใช้กันเป็นส่วนใหญ่ในการทำงานแบบขนาน

ถ้าในระหว่างงานไม่มีความสัมพันธ์ของลำดับก่อนหลังของการทำงาน ปัญหาการจัดลำดับของงานประเภทนี้เราเรียกว่าปัญหาของการจัดแบ่งงาน โดยปัญหาของการจัดแบ่งงานจะเกี่ยวข้องกับระบบคอมพิวเตอร์แบบกระจาย ระบบคอมพิวเตอร์แบบกระจายประกอบไปด้วยส่วนต่างๆของกระบวนการ (Processing Element) ของงานที่มีการเชื่อมต่อกันเป็นระบบเครือข่ายและการกระจายงานของโปรแกรมจะเกี่ยวข้องกับงานที่มีการติดต่อสื่อสาร (Communication Task) เทคนิคการแบ่งงานโดยทำการกำหนดงานของโปรแกรม (Program Tasks) ต่อส่วนต่างๆของกระบวนการด้วยฟังก์ชันที่มีเป้าหมายแน่นอน (a Certain Objective Function) ที่เหมาะสม ตัวอย่างของฟังก์ชันที่มีเป้าหมายที่เหมาะสมจะต้องใช้เวลาในการประมวลผลที่ตัวประมวลผลและค่าใช้จ่ายในการติดต่อสื่อสารระหว่างงานหรือตัวประมวลผลโดยรวมน้อยที่สุด

เมื่อกราฟงานใช้ถูกแสดงลำดับการทำงานของโปรแกรม ดังนั้นทำให้กราฟงานสามารถแสดงถึงชุดของลำดับของงานส่วนต่างๆ ปัญหาการจัดลำดับงานมีสองประเภทด้วยกัน ปัญหาแบบแรกการจัดลำดับงานสามารถแก้ไขปัญหาได้ด้วยวิธีการที่เหมาะสม ปัญหาแบบที่สองการจัดลำดับงานไม่สามารถแก้ไขปัญหาได้ด้วยวิธีการที่เหมาะสม ปัญหาที่เกิดขึ้นในแบบหลังเราเรียกว่าปัญหาแบบดื้อด้าน (Intractable Problem or Computational Intractable) ปัญหาดังกล่าวได้ถูกจัดให้เป็น NP-complete ปัญหาของการจัดลำดับงานถ้าหากไม่มีการพิจารณาค่าใช้จ่ายที่เกิดจากการติดต่อสื่อสารระหว่างงานหรือตัวประมวลผลแล้ววิธีการแก้ไขที่เหมาะสมสำหรับปัญหาประเภทนี้เราสามารถทำได้ด้วยสามวิธีการ วิธีที่ 1) ให้พิจารณากราฟงานเหมือนกับต้นไม้ (Tree) วิธีที่ 2) ให้พิจารณากราฟงานเหมือนกับลำดับช่วง (An Interval Order) และวิธีที่ 3) ใช้ตัวประมวลผลจำนวนสองตัวประมวลผลช่วยในการจัดการกับปัญหาการจัดลำดับงานบนกราฟงาน ทั้งสามวิธีการดังกล่าวงานทั้งหมดบนกราฟงานจะถูกสมมติว่ามีเวลาที่ใช้ในการประมวลผลที่เหมือนกัน แต่ถ้าหากมีการพิจารณาค่าใช้จ่ายในการติดต่อสื่อสารระหว่างตัวประมวลผล ดังนั้นวิธีการแก้ไขปัญหาที่เหมาะสมสามารถทำได้ด้วยข้อจำกัด ตัวอย่างเช่นถ้ากราฟงานถูกพิจารณาเหมือนกับลำดับช่วงให้ทำการพิจารณาค่าใช้จ่ายในการติดต่อสื่อสารระหว่างงานหรือตัวประมวลผลเช่นเดียวกับค่าใช้จ่ายในการประมวลผลของงาน ทำให้เราสามารถได้ลำดับการทำงานที่เหมาะสมได้เช่นกัน

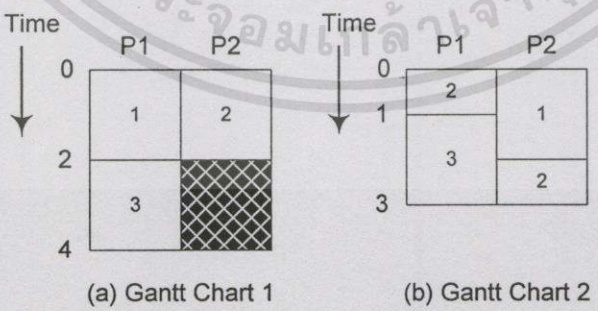
ถ้าปัญหาการจัดลำดับงานสามารถแก้ไขได้ด้วยโพลีโนเมียลทาม (Polynomial time) เราอาจจะกล่าวได้ว่าวิธีการดังกล่าวเป็นวิธีการแก้ไขปัญหาจริง (Exact Solution) ที่สามารถแก้ไขปัญหาดังกล่าวได้ แต่ถ้าหากในเวลาจริง (real time)เราไม่สามารถหาวิธีการแก้ไขปัญหาจริงได้ เราสามารถที่จะใช้ฮิวลิสติกเพื่อให้ได้ผลลัพธ์ที่ใกล้เคียงกับวิธีการที่เหมาะสม (an approximation of the optimal solution) บ่อยครั้งที่เราจะใช้วิธีการสังเกตการณ์หรือฮิวลิสติกโดยการใช้พารามิเตอร์ต่างๆที่มีผลต่อระบบในทางอ้อมเพื่อให้ได้ผลลัพธ์ที่ใกล้เคียงกับวิธีการที่เหมาะสม ตัวอย่างเช่นในระหว่างงานที่มีค่าใช้จ่ายในการติดต่อสื่อสารระหว่างกันมากๆ เราสามารถที่จะนำเอางานทั้งสองนั้นมารวมกันเพื่อให้ทำงานที่ตัวประมวลผลเดียว ผลจากการใช้ฮิวลิสติกดังกล่าวทำให้เราสามารถลดเวลาที่ใช้ในการติดต่อสื่อสารระหว่างตัวประมวลผลได้ และยังเป็นการลดเวลาที่ใช้ในการประมวลของระบบลง

2.3 การจัดแบ่งประเภทของลำดับการทำงาน

การจัดแบ่งประเภทของการจัดลำดับการทำงานดังแสดงไว้ในรูปที่2.2 เป็นการจัดแบ่งอย่างคราวๆ นอกเหนือไปจากนี้ยังมีลักษณะอื่นอีกที่แตกต่างกันออกไปในระบบการจัดลำดับการทำงานที่กล่าวมา ในส่วนนี้จะพิจารณาการจัดแบ่งประเภทของลำดับการทำงานอื่นๆที่นอกเหนือไปจากรูปที่2.2 ได้แสดงไว้

2.3.1 ระบบที่มีแอฟฟริเคชั่นเดียวกับระบบที่มีมากกว่าหนึ่งแอฟฟริเคชั่น การจัดแบ่งประเภทลำดับการทำงานโดยวิธีการนี้จะพิจารณาถึงสิ่งที่มีตัวตน (Entities) ที่สามารถจะถูกกำหนดลำดับการทำงานได้ ถ้าสิ่งที่มีตัวตนนี้คืองาน โดยที่งานนี้ประกอบกันขึ้นเป็นโปรแกรมแบบขนาน ดังนั้นจะมีอยู่สองกรณีให้ทำการพิจารณา แบบแรกเมื่อมีเพียงแอฟฟริเคชั่นเดียวที่กำหนดให้ทำงานได้ในแต่ละครั้ง ตัวอย่างเช่นมีอยู่หนึ่งแอฟฟริเคชั่นที่ประกอบไปด้วยงานหลายๆงานร่วมทำงานกันและมีการติดต่อสื่อสารระหว่างกัน เมื่อแอฟฟริเคชั่นนี้ถูกกำหนดให้ทำการประมวลอยู่ระบบแบบไฮเปอร์คิว (Hypercube) ดังนั้นเป้าหมายสำคัญเพื่อให้เวลาที่ใช้ในการประมวลผลโดยรวมของระบบน้อยที่สุด แบบที่สองถ้ามีแอฟฟริเคชั่นมากกว่าหนึ่งแอฟฟริเคชั่นและแต่ละแอฟฟริเคชั่นจะทำงานแบบขนาน โดยให้แอฟฟริเคชั่นแบบขนานหลายๆแอฟฟริเคชั่นทำงานพร้อมๆกัน ตัวอย่างเช่น ถ้าแอฟฟริเคชั่นแบบขนานหลายๆแอฟฟริเคชั่นทำงานพร้อมกันในลักษณะสภาวะแวดล้อมแบบขนานโดยแบ่งเวลาร่วมกัน (A time-sharing parallel environment) ซึ่งเหมือนกับการเรียงลำดับที่สมมาตรกัน (Sequent Symmetry) โดยที่วัตถุประสงค์ในที่นี้เพื่อให้เวลาที่ใช้ในการตอบสนอง (Response Time) และเวลาที่เสร็จสิ้นสมบูรณ์ของแต่ละแอฟฟริเคชั่นน้อยที่สุด

2.3.2 นอนพรีเอ็มทีฟ และ พรีเอ็มทีฟ (Non-preemptive and Preemptive) การจัดลำดับการทำงานแบบนอนพรีเอ็มทีฟ หมายถึงงานหนึ่งงานไม่สามารถที่จะให้หยุดกลางคันในระหว่างการประมวลผลได้ เมื่องานๆนั้นการประมวลผลได้เริ่มต้นผ่านไปแล้ว โดยจะต้องให้งานๆนั้นถูกทำการประมวลผลจนเสร็จสิ้นสมบูรณ์ โดยทั่วๆไปแล้วการจัดลำดับการทำงานแบบพรีเอ็มทีฟจะยอมให้งานนั้นๆ สามารถถูกหยุดการทำงานกลางคันในระหว่างการประมวลผลได้ และจะถูกถอดถอนออกจากตัวประมวลผล ซึ่งอยู่ภายใต้ข้อสมมติฐานที่ว่าในที่สุดแล้วงานๆนั้นจะถูกให้ทำการประมวลอีกครั้งเมื่องานๆนั้นถูกเรียกให้ทำงานอีกครั้ง การหยุดกลางคันในระหว่างการประมวลผลของงานในการจัดลำดับการทำงานแบบพรีเอ็มทีฟเป็นการสนับสนุนให้ระบบเกิดโอเวอร์เฮด



รูปที่ 2.3 แสดงแกทชาร์ทสำหรับ (a) การจัดลำดับการทำงานแบบนอนพรีเอ็มทีฟ (b) การจัดลำดับการทำงานแบบพรีเอ็มทีฟ ที่ประกอบไปด้วย 2 ตัวประมวลผล

รูปที่2.3แสดงแกทชาร์ทของการจัดลำดับการทำงานแบบนอนพรีเอ็มทีฟและแบบพรีเอ็มทีฟ โดยมีข้อกำหนดว่ามีกราฟงานที่ประกอบไปด้วยงาน3งานที่เป็นอิสระต่อกัน โดยงานแต่ละงานต้องการใช้เวลาในการประมวลผลเท่ากับ2หน่วย(Units) โดยงานทั้งหมดเหล่านี้จะถูกกำหนดลำดับการทำงานอยู่บนเครื่องเป้าหมาย (Target Machine) ที่มีตัวประมวลผลจำนวน2ตัวประมวลผลที่เหมือนกัน จากรูปแสดงให้เห็นว่าการจัดลำดับการทำงานแบบนอนพรีเอ็มทีฟสำหรับตัวอย่างนี้จะต้องใช้เวลาในการประมวลผลจำนวน4หน่วยเวลา ในขณะที่การจัดลำดับการทำงานแบบพรีเอ็มทีฟใช้เวลาในการประมวลผลจำนวน3หน่วยเวลา

2.3.3 นอนอะเด็ฟทีฟและอะเด็ฟทีฟ (Non-Adaptive and Adaptive) ลำดับการทำงานแบบนอนอะเด็ฟทีฟจะสามารถเปลี่ยนแปลงพฤติกรรมการทำงานได้ โดยลำดับการทำงานจะต้องสอดคล้องต่อการตอบสนอง(feedback)จากระบบ ในทางตรงกันข้ามการจัดลำดับการทำงานแบบอะเด็ฟทีฟ สามารถที่จะตัดสินใจเปลี่ยนแปลงลำดับการทำงานให้สอดคล้องกับพฤติกรรมของระบบก่อนหน้านี้และปัจจุบัน ในการตอบสนองต่อการตัดสินใจก่อนหน้านี้กระทำได้โดยระบบการจัดลำดับการทำงาน (Scheduling System) โดยปกติแล้วการจัดลำดับการทำงานแบบบอะเด็ฟทีฟเป็นการจัดลำดับการทำงานแบบไดนามิก เพราะว่าผู้จัดลำดับการทำงานอาจจะเปลี่ยนแปลงข้อมูลเกี่ยวกับระบบและการตัดสินใจของลำดับการทำงานเมื่อใดก็ได้ (On the fly)

2.4 รูปแบบของลำดับการทำงาน

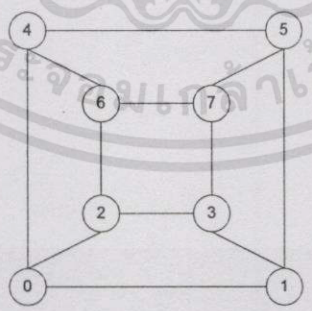
ในส่วนนี้จะอธิบายเกี่ยวกับรูปแบบโดยทั่วๆไปสำหรับการแจกแจงปัญหาของลำดับการทำงาน โดยปัญหาส่วนใหญ่ที่ทำการศึกษามักจะถูกแสดงออกมาในรูปแบบพิเศษของปัญหานั้นๆ และบ่อยครั้งทีเดียวที่การทำงานของโปรแกรมมีลักษณะเป็นแบบลูป ซึ่งลักษณะเช่นนี้เราไม่สามารถแสดงออกมาในรูปแบบของโปรแกรมแบบขนานได้ รูปแบบที่เราจะทำการศึกษาในส่วนนี้เป็นแบบเป็นไปตามทฤษฎี โดยอาศัยข้อมูลทั้งหมดใช้ในการควบคุมการตัดสินใจของลำดับการทำงาน โดยเฉพาะอย่างยิ่งกราฟงานที่ใช้แสดงโปรแกรมแบบขนานและเครื่องเป้าหมายได้ถูกสมมติแล้วว่าเป็นไปได้หรือมีอยู่จริง ก่อนที่โปรแกรมแบบขนานนั้นจะเริ่มทำการประมวลผล

ในกระบวนขั้นตอนแสดงลำดับการทำงานของโปรแกรมแบบขนานจะประกอบไปด้วยส่วนประกอบหลักห้าส่วนด้วยกัน 1) เครื่องเป้าหมาย 2) งานแบบขนาน (Parallel Tasks) 3) ค่าใช้จ่ายในการประมวลผลและการติดต่อสื่อสาร 4) การกำหนดลำดับการทำงาน (Generated Schedule) และ 5) การวัดสมรรถนะ ซึ่งส่วนประกอบเหล่านี้สามารถนำมาใช้ในการประมาณเวลาที่ใช้ในการประมวลผล และค่าความล่าช้าหรือความหน่วงเนื่องมาจากการติดต่อสื่อสาร (Communication Delays) ระหว่างงานหรือตัวประมวลผล

2.4.1 เครื่องเป้าหมาย ได้ถูกสมมติไว้ว่าประกอบไปด้วยส่วนเครื่องคอมพิวเตอร์ที่แตกต่างกันจำนวน m เครื่อง เครื่องคอมพิวเตอร์เหล่านี้ได้ถูกเชื่อมต่อกันเป็นระบบเครือข่ายในรูปแบบอะไรก็ได้ (arbitrary interconnection network) โดยที่เครื่องคอมพิวเตอร์แต่ละเครื่องสามารถที่จะประมวลผลได้เพียงหนึ่งงานต่อการทำงานหนึ่งครั้งเท่านั้น และงานทั้งหมดสามารถถูกประมวลผลอยู่บนเครื่องคอมพิวเตอร์เครื่องใดก็ได้ โดยปกติแล้วลักษณ์ของเป้าหมายสามารถถูกอธิบายเหมือนกับระบบหนึ่งระบบที่ประกอบได้ด้วยสัญญลักษณ์ $(P, [P_{ij}], [S_i], [I_i], [B_i], [R_{ij}])$ เมื่อ

1. $P = \{P_1, ..., P_m\}$ คือเซตหรือชุดของตัวประมวลผลที่อยู่ในรูปแบบของสถาปัตยกรรมแบบขนาน
2. $[P_{ij}]$ คือ โทโพโลยีแบบเมทริก (topology matrix) ที่มีการเชื่อมต่อขนาด m x m
3. S_i เป็นการระบุความเร็วของตัวประมวลผล P_i เมื่อ $1 \leq i \leq m$
4. I_i เป็นการระบุค่าใช้จ่ายเริ่มต้นของการเริ่มส่งข้อมูลบนตัวประมวลผล P_i เมื่อ $1 \leq i \leq m$
5. B_i เป็นการระบุค่าใช้จ่ายเริ่มต้นของการเริ่มประมวลผลงานที่ตัวประมวลผล P_i เมื่อ $1 \leq i \leq m$
6. R_{ij} คืออัตราการส่งข้อมูลบนเครือข่ายระหว่างสองตัวประมวลผล P_i และ P_j ที่เชื่อมติดกัน (Adjacent Processors)

การเชื่อมต่อกันของตัวประมวลผลสามารถถูกแสดงได้โดยการใช้ Undireted Graph ซึ่งถูกเรียกว่า กราฟของเครื่องเป้าหมาย (Target Machine Graph) รูปที่ 2.4 แสดงตัวอย่างของเครื่องเป้าหมายที่ประกอบไปด้วยเซตของตัวประมวลผลจำนวน 8 ตัว(m = 8) ที่มีลักษณะการเชื่อมต่อเป็นแบบไฮเปอร์คิว (Hyper Cube) สามมิติ บางครั้งตัวประมวลผลอาจจะถูกอ้างถึงอย่างง่ายเช่น 1 หมายถึงตัวประมวลผลตัวที่หนึ่ง มากกว่าที่จะถูกแสดงในรูปของ P_1

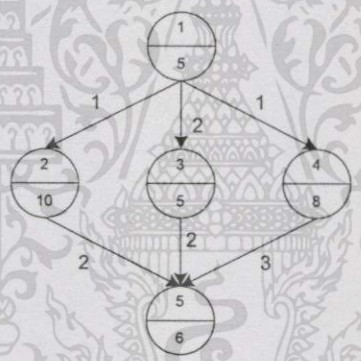


รูปที่ 2.4 แสดงเครื่องเป้าหมาย

2.4.2 งานของโปรแกรมแบบขนาน โปรแกรมแบบขนานมักจะถูกแสดงในรูปแบบของพาร์เชียล ออร์เดอร์เซต (partial order set) $(T, <)$ เมื่อ T คือเซตของกลุ่มงาน การแสดงความ

สัมพันธ์ u < v แสดงถึงการคำนวณของงาน v ขึ้นอยู่กับผลที่ได้จากการคำนวณของงาน u โดยที่งาน u จะต้องถูกคำนวณก่อนงาน v และผลลัพธ์ที่ได้จากการคำนวณของงาน u จะต้องถูกรู้โดยตัวประมวลผลที่กำลังคำนวณงาน v อยู่ ลักษณะของโปรแกรมแบบขนานสามารถถูกกำหนดอยู่ในรูปแบบของ $(T, <, [D_{ij}], [A_{ij}])$ โดยที่

1. $T = \{t_1, t_2, \ldots, t_n\}$ คือเซตของกลุ่มงานที่จะถูกประมวลผล
2. $<$ คือ พาร์เชียล ออร์เดอร์ ที่กำหนดอยู่บน T เมื่อสัญญลักษณ์ดังกล่าวเป็นการระบุถึงข้อบังคับของลำดับก่อนหลังของการทำงาน (operational precedence constraints) ตัวอย่างเช่น $t_i < t_j$ เป็นการแสดงถึงงาน t_i จะต้องเสร็จสิ้นสมบูรณ์เสียก่อน ก่อนที่งาน t_j จะสามารถเริ่มต้นทำการประมวลผลได้
3. $[D_{ij}]$ เป็นการแสดงเมทริกขนาน n x n ของการสื่อสารข้อมูล ถ้า $D_{ij} \geq 0$ คือจำนวนของข้อมูลที่ถูกส่งจากงาน t_i ไปยังงาน t_j, เมื่อ $1 \leq i\, j \leq n$
4. $[A_i]$ คือ เวกเตอร์ (vector) ของจำนวนของการคำนวณที่มีขนาด n ตัวอย่างเช่น $A_i > 0$ คือ จำนวนของคำสั่งที่ต้องการในการประมวลผลงาน t_i เมื่อ $1 \leq i \leq n$



รูปที่ 2.5 แสดงกราฟงานที่ประกอบได้ด้วยงานหรือตัวประมวลผลจำนวน 5 ตัว โดยที่แต่ละตัวประมวลผลแสดงค่าใช้จ่ายในการประมวลผลของงานและค่าใช้จ่ายในการติดต่อสื่อสารระหว่างตัวประมวลผล

พาร์เชียล ออร์เดอร์ ($<$) เป็นการแสดงกราฟงานที่มีทิศทางหรือ directed acyclic task graph หรือเรียกว่ากราฟงาน Directed Edge $< i, j >$ เป็นการแสดงการทิศทางการเชื่อมต่อของงานสองงานกล่าวคือ งาน t_i และ t_j โดยที่งาน t_i จะต้องเสร็จสิ้นสมบูรณ์เสีย ก่อนที่งาน t_j จะสามารถเริ่มต้นทำการประมวลผลได้ รูปที่ 2.5 แสดงตัวอย่างของกราฟงานที่ประกอบไปด้วยโหนด (Node) จำนวน 5 โหนด (n = 5) โดยที่แต่ละโหนดเป็นการแสดงถึงงาน หมายเลขที่บ่รากฏอยู่ส่วนบนของโหนดหมายถึงหมายเลขของโหนดหรืองาน ส่วนหมายเลขที่ปรากฏอยู่ส่วนล่างหมายถึงพารา-มิเตอร์ของ A_i หรือจำนวนของการคำนวณที่ต้องการสำหรับงาน t_i และหมายเลขที่ปรากฏอยู่บน

Directed Edge < i, j> เป็นการแสดงถึงพารามิเตอร์ D_{ij} ตัวอย่างจากรูปที่ 2.5 สามารถแสดงได้ว่า $A_1 = 5$, และ $D_{45} = 3$ เป็นต้น

2.4.3 ค่าใช้จ่ายในการประมวลผลและการติดต่อสื่อสาร การกำหนดรูปแบบของโปรแกรมแบบขนานและอธิบายของเครื่องเป้าหมายที่จะทำการประมวลผลโปรแกรม เวลาที่ใช้ในการประมวลผลงาน และค่าของความหน่วง (Latency Time) เนื่องมาจากการติดต่อสื่อสารระหว่างงานหรือตัวประมวลผล สามารถถูกกำหนดได้ดังนี้

1. T_{ij}: เป็นการแสดงเวลาที่ใช้ในการประมวลผลของงาน i เมื่อถูกกำหนดให้ทำการประมวลผลอยู่บนตัวประมวลผล j จากข้อกำหนดดังกล่าวสามารถแสดงอยู่ในรูปแบบของสมการได้ดังนี้

$$T_{ij} = \frac{A_i}{S_j} + B_j \tag{2.1}$$

2. $C(i_1, i_2, j_1, j_2)$: เป็นการแสดงค่าของความหน่วงหรือค่าความล่าช้า (Delay Time) เนื่องมาจากการติดต่อสื่อสารระหว่างงาน t_1 และ t_2 เมื่องานเหล่านี้ถูกประมวลผลอยู่บนตัวประมวลผล j_1 และ j_2 ตามลำดับ ค่า$C(i_1, i_2, j_1, j_2)$แสดงถึงผลกระทบต่อพารามิเตอร์ของประสิทธิภาพเครื่องเป้าหมาย เมื่อค่าดังกล่าวเป็นแสดงถึงขนาดของข้อมูลที่ถูกส่งออกจากงาน i_1 ไปยัง i_2 และ $C(i_1, i_2, j_1, j_2)$ สามารถคำนวณได้จาก :
สมมติให้ j_1 และ j_2 เป็นตัวประมวลผลที่อยู่ติดกัน แล้วค่าความหน่วงเนื่องมาจากการติดต่อสื่อสารระหว่างตัวประมวลผลของการส่งข้อมูลจากงาน i_1 ที่กำลังประมวลผลอยู่บนตัวประมวลผล j_1 ไปยังงาน i_2 ที่กำลังประมวลผลอยู่บนตัวประมวลผล j_2 ดังนั้นค่า $C(i_1, i_2, j_1, j_2)$ เท่ากับ:

$$C(i_1, i_2, j_1, j_2) = \frac{D_{i_1 i_2}}{R_{j_1 j_2}} + I_{j_1} \tag{2.2}$$

โดยปกติแล้วมักจะมีมากกว่าหนึ่งข้อมูลที่สามารถส่งผ่านจากตัวประมวลผลหนึ่งไปยังอีกหนึ่งตัวประมวลผล โดยการใช้เส้นทางการส่งผ่านข้อมูลเดิม ดังนั้นค่าของ คอนเทนชั่นดีเลย์ (Contention Delay) มักจะถูกนำมาพิจารณาร่วมด้วย ถ้าสมมติว่าเราสามารถประมาณค่าของคอนเทนชั่นดีเลย์ บนเส้นทางการส่งผ่านข้อ

มูลจากตัวประมวลผล j_1 และ j_2 (ถูกแสดงได้ด้วยสัญญลักษณ์ $CD_{j1\,j2}$) จากสมการข้างบนเราสามารถนำมาแสดงใหม่ได้ดังนี้

$$C(i_1, i_2, j_1, j_2) = \frac{D_{i_1 i_2}}{R_{j_1 J_2}} + I_{j_1} + CD_{j_1 j_2} \qquad (2.3)$$

อะไรเกิดขึ้นหากตัวประมวลผลที่ต้นทางและปลายทางไม่ได้อยู่ติดกัน สมมติว่าการส่งผ่านข้อมูลจากตัวประมวลผล j_1 ไปยัง j_2 จะต้องผ่านเส้นทาง $j_1, k_1, k_2, \ldots, k_z, j_2$ ตามลำดับ ดังนั้นจำนวนของตัวประมวลผลที่จะต้องผ่านเท่ากับ $z + 1$ ดังนั้นค่าของ $C(i_1, i_2, j_1, j_2)$ สามารถหาได้ดังนี้

$$C(i_1, i_2, j_1, j_2) = \frac{D_{i_1 i_2}}{R_{j_1 k_1}} + I_{j_1} + CD_{j_1 k_2} + \frac{D_{i_1 i_2}}{R_{k_2 j_2}} + I_{k_2} + CD_{k_2 j_2} +$$

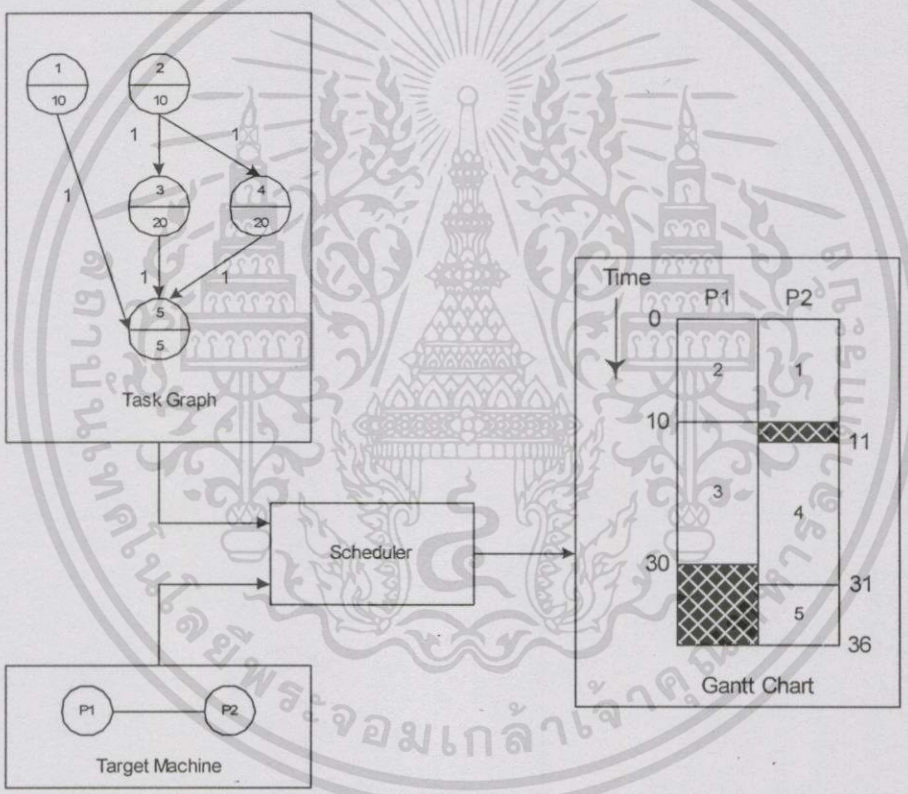
$$\sum_{i-1}^{z-1} \left(\frac{D_{i_1 i_2}}{R_{k_i k_{i+1}}} + I_{k_i} + CD_{k_i k_{i+1}} \right) \qquad (2.4)$$

ในกรณีที่ตัวประมวลผลทั้งหมดที่มีอยู่ในระบบมี I/O ที่เหมือนกันและจำนวนของเวลาที่ใช้ในการเริ่มส่งข้อมูลเท่ากัน และอัตราการส่งผ่านข้อมูลบนเส้นทางการส่งผ่านข้อมูลเท่ากันหมด เราสามารถแสดง $C(i_1, i_2, j_1, j_2)$ ได้ใหม่ดังนี้

$$C(i_1, i_2, j_1, j_2) = \left(\frac{D_{i_1 i_2}}{R} + I \right) * (Z + 1) + CD_{j_1 j_2} \qquad (2.5)$$

2.4.4 ลำดับการทำงาน การจัดลำดับการทำงานสำหรับกราฟงานใดๆ $G = (T, A)$ บนเครื่องเป้าหมายที่ประกอบไปด้วยจำนวนตัวประมวลผล m ตัวประมวลผล คือฟังก์ชัน f ใดๆ โดยที่ฟังก์ชันf กำหนดให้งานแต่ละงานใดๆถูกวางลงบนหนึ่งตัวประมวลผล และมีเพียงเวลาเริ่มต้นเดียว โดยปกติแล้ว $f: T \rightarrow \{1, 2, 3, \ldots, m\} \times [0, \infty)$ ถ้าฟังก์ชัน $f(v) = (i, t)$ เมื่อ $v \in T$ เราสามารถกล่าวได้ว่า งาน v ถูกกำหนดลำดับการทำงานโดยจะถูกประมวลผลบนตัวประมวลผล P_i และกำหนดให้เวลาเริ่มต้นเท่ากับ t สังเกตได้ว่าไม่มีงาน u แสดงว่า $f(v) = f(u)$ แต่ถ้า $v < u$ และกำหนดให้ $f(v) = (i, t_1)$, $f(u) = (j, t_2)$ แล้ว $t_1 < t_2$ ฟังก์ชันกำหนดลำดับการทำงานคือลักษณะที่สามารถกำหนดลำดับการทำงานได้ โดยฟังก์ชันมีการกำหนดความสัมพันธ์ของลำดับก่อนหลังและการติด

ต่อสื่อสารเป็นสิ่งสำคัญ แกรฑชาร์ทสามารถให้ข้อมูลของการจัดลำดับการทำงาน เมื่อเวลาเริ่มต้นและเวลาเสร็จสิ้นหรือสิ้นสุดสำหรับงานใดๆบนกราฟงานสามารถใช้แกรฑชาร์ทแสดงให้เห็นได้โดยง่าย รูปที่2.6แสดงระบบการจัดลำดับการทำงาน เมื่ออินพุทประกอบไปด้วยกราฟงาน และเครื่องเป้าหมาย ขณะที่เอาพุทของการจัดลำดับการทำงานถูกแสดงด้วยรูปแบบของแกรฑชาร์ท จากรูปที่ 2.6 แสดงให้เห็นว่า งาน1เริ่มต้น ณ เวลาเท่ากับ 0 และเสร็จสิ้น ณ เวลาที่ 10 ขณะเดียวกัน งาน4เริ่มต้น ณ เวลาที่ 11 และเสร็จสิ้น ณ เวลาที่ 31 เป็นต้น บริเวณสีทึบระหว่างเวลาที่ 10 และ 11 บนตัวประมวลผล P_2 แสดงถึงค่าความหน่วงอันเนื่องมาจากการติดต่อสื่อสารระหว่างงานหรือตัวประมวลผล จากรูปแสดงให้เห็นว่าเป็นค่าความหน่วงของ งาน2 บนตัวประมวลผล P_1 ที่ติดต่อไปยังงาน4 บนตัวประมวลผล P_2



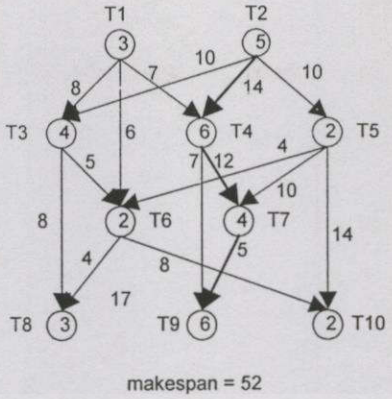
รูปที่ 2.6 แสดงกราฟงานที่ได้จากผู้จัดลำดับการทำงาน เครื่องเป้าหมาย ประกอบกันเป็นอินพุทของระบบการจัดลำดับการทำงาน และเอาพุทเป็นการแสดงแกรฑชาร์ทของลำดับงาน

2.4.5 การวัดสมรรถนะ อัลกอริทึมที่มีประสิทธิภาพเพื่อสำหรับการจัดลำดับการทำงานของงานบนตัวประมวลผลต่างๆได้อย่างเหมาะสม เป็นการอธิบายถึงการวัดสมรรถนะในการแก้ไขปัญหาการจัดลำดับการทำงาน การวัดสมรรถนะของอัลกอริทึมหรือวิธีการแก้ไขปัญหาการจัดลำดับ

การทำงานจะมีอยู่ด้วยกันที่สำคัญๆหลายวิธีการด้วยกัน เช่น ความสมดุลในการกระจายงานลงบนตัวประมวลผลต่างๆ (Load Balancing) และ เวลาที่เสร็จสิ้นสมบูรณ์ของงานที่น้อยที่สุด (minimizing completion time) ในวิทยานิพนธ์เล่มนี้มีเป้าหมายที่สำคัญของการจัดลำดับการทำงาน คือ เวลาโดยรวมที่เสร็จสิ้นที่น้อยที่สุดของงานบนโปรแกรมแบบขนาน ดังนั้นในการวัดสมรรถนะของอัลกอริทึมหรือวิธีการในการจัดลำดับการทำงานของงานบนโปรแกรมแบบขนาน คือ ความยาวของการจัดลำดับการทำงาน หรือเวลาที่เสร็จสิ้นสมบูรณ์ที่มากที่สุด ความยาวของการจัดลำดับการทำงานสามารถอธิบายดังนี้ กำหนดให้กราฟงาน $G = (T, A)$ และการจัดลำดับการทำงานสำหรับกราฟงานบนตัวประมวลผลจำนวน m ตัวประมวลผลสำหรับกราฟงานใดๆ แสดงด้วย f และความยาวของลำดับการทำงานสำหรับ f ของ G คือเวลาที่เสร็จสิ้นที่มากที่สุดสำหรับงานใดๆ ในกราฟงาน G โดยปกติแล้ว ความยาวของ f แสดงด้วย $\text{length}(f) = t_{max}$ เมื่อ $t_{max} = \text{maximum}\{t + T_{ij}\}$ และ $f(i) = (j, t)$ โดยที่ $\forall i \in T, 1 \leq j \leq m$ โดยที่ T_{ij} คือเวลาที่ใช้ในการประมวลผลของงาน i บนตัวประมวลผล j

2.5 ลักษณะของเม็ดเนื้องานบนกราฟงาน

การแสดงขั้นตอนและทิศทางการทำงานของโปรแกรมแบบขนานดังแสดงไว้ในหัวข้อที่แล้วนั้น ยังมีวิธีการแสดงที่นอกเหนือไปจากวิธีการดังกล่าว ที่มักจะนิยามใช้กัน กล่าวคือจะกำหนดให้ Weighted Directed Acyclic Task $G = (V, E, \mu, \lambda)$ หรือ Weight DAG G เมื่อ V คือเซตของโหนดหรืองานบน DAG ส่วน E คือเซตของอาร์ค (Arc) ระหว่างโหนดใน V เมื่อ $v_i \in V$ เมื่อ $i = 1, 2, 3, \ldots, n$ โดยที่ n คือจำนวนสมาชิกใน V และ $e_i \in E$ เมื่อ $i = 1, 2, 3 \ldots, m$ โดยที่ m คือจำนวนของอาร์คหรือจำนวนคู่ของตัวประมวลผลที่มีการติดต่อสื่อสารระหว่างโหนดบน DAG ถ้ากำหนดให้ u และ v เป็นโหนดสมาชิกของ V และกำหนดให้ $(u, v) \in E$ แสดงให้เห็นว่างานของโหนด u จะต้องเสร็จสิ้นสมบูรณ์ก่อน แล้วโหนด u จึงจะเริ่มต้นส่งข้อมูลมายังโหนด v จากนั้นโหนด v จึงจะสามารถเริ่มต้นทำการประมวลผลได้ กำหนดให้ $\mu(v)$ เมื่อ $v \in V$ หมายถึงค่าใช้จ่ายที่ใช้ในการประมวลผลของงานบนโหนด v หรือตัวประมวลผล v และ $\lambda(u, v)$ หมายถึงค่าใช้จ่ายที่ใช้ในการติดต่อสื่อสารระหว่างตัวประมวลผลหรือค่าความหน่วง ที่เกิดขึ้นในการส่งผ่านข้อมูลจากโหนด u ไปยังโหนด v เมื่อ u และ $v \in V$ โดยที่ $\lambda(u, v)$ ไม่เท่ากับศูนย์เมื่อโหน u และ v ถูกกำหนดให้อยู่บนตัวประมวลผลที่ต่างกัน และเท่ากับศูนย์เมื่อโหนด u และ v อยู่บนตัวประมวลผลเดียวกัน



รูปที่ 2.7 แสดง Weighted DAG

รูปที่ 2.7 เป็นการแสดง Weighted DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด (A Fine Grain Task) รายละเอียดของลักษณะเม็ดเนื้องานจะกล่าวถึงในภายหลัง โดยที่แต่ละโหนดบน DAG จะอาศัยอยู่บนตัวประมวลผลที่แยกออกจากกัน นอกเหนือจากนี้ยังแสดงให้เห็นลำดับก่อนและหลังการทำงานของโหนดต่างๆ เช่น งานที่โหนด T5 จะเริ่มต้นทำงานได้ก็ต่อเมื่องานที่โหนด T2 ทำงานเสร็จสิ้นแล้วและมีการส่งผ่านข้อมูลทั้งหมดจากโหนด T2 มายังโหนด T5 หมดสิ้น จากตัวอย่างดังกล่าวแสดงให้เห็นว่าค่าของ μ(T2) เท่ากับ 5 และ μ(T5) เท่ากับ 2 ขณะเดียวกัน λ(T2, T5) เท่ากับ 10 ส่วนค่าของ Makespan ของลำดับการทำงานหรือเวลาที่ใช้ในการประมวลผลของ DAG คือค่าของผลรวมระหว่างโหนดและเส้นทางระหว่างโหนดที่มีค่าที่มากที่สุดบน DAG นั้น ค่าของ Makespan ของลำดับการทำงานสำหรับ DAG ในรูปที่ 2.7 มีค่าเท่ากับ 52

ส่วน DAG ใดๆจะถูกกล่าวว่าจะมีลักษณะของเม็ดเนื้องานแบบละเอียด หรือแบบหยาบ (A Coarse Grain Task) ขึ้นอยู่กับค่าของ Granularity ของ DAG นั้น ถ้าค่าของ Granularity ของ DAG มีค่ามากกว่าหรือเท่ากับ 1 ($g(G) \geq 1$) แสดงว่า DAG นั้นจะมีลักษณะของเม็ดเนื้องานแบบหยาบ ในทางตรงกันข้าม หากค่าของ Granularity ของ DAG มีค่าน้อยกว่า 1 ($g(G) < 1$) แสดงว่า DAG นั้นมีลักษณะของเม็ดเนื้องานแบบละเอียด ความหมายของ Task Grain และ Granularity จะยึดตามนิยามจาก [2] เป็นหลักในการอ้างอิงของวิทยานิพนธ์ฉบับนี้ โดยค่าของ Grain เป็นค่าของอัตราส่วนระหว่างค่าของ μ ต่อ λ ของ v_i เมื่อ $v_i \in V$ ค่าของ Task Grain เป็นค่าของเม็ดเนื้องานในระดับโลคัล ส่วนค่าของ Granularity เป็นค่าของเม็ดเนื้องานในระดับโกลบอล ให้พิจารณารูปที่ 2.8 เป็นการแสดงฟอกเซต (Fork Set) ของโหนดn_x (รูปที่ 2.8(a)) และจอยย์เซต (Joint Set) ของโหนดn_x (รูปที่ 2.8(b)) จากข้อกำหนดสำหรับ DAG ใดๆ G = (V, E, μ, λ)แล้ว ค่า Granularity ของ DAG G ถูกแสดงด้วย g(G) และค่าของ Task Grain ของงานหรือโหนดใดๆถูกแสดงด้วย g_n เมื่อ $n \in V$

Fork Set $F_x = \{n_1, n_2, \ldots, n_m\}$ จะได้

$$g(F_x) = \min_{i=1,2,\ldots,m}(\mu_i) / \max_{i=1,2,\ldots,m}(\lambda_i) \quad (2.6)$$

โดยที่ λ_i จะต้องเป็นค่าของเวลาที่ใช้ในการติดต่อสื่อสารระหว่างโหนดn_x กับโหนดn_i ใดๆเมื่อ $\forall n_i \in F_x$ และ Joint Set $J_x = \{n_1, n_2, \ldots, n_m\}$ จะได้

$$g(J_x) = \min_{i=1,2,\ldots,m}(\mu_i) / \max_{i=1,2,\ldots,m}(\lambda_i) \quad (2.7)$$

โดยที่ λ_i จะต้องเป็นค่าของเวลาที่ใช้ในการติดต่อสื่อสารระหว่างโหนดn_x กับโหนดn_i ใดๆเมื่อ $\forall n_i \in J_x$

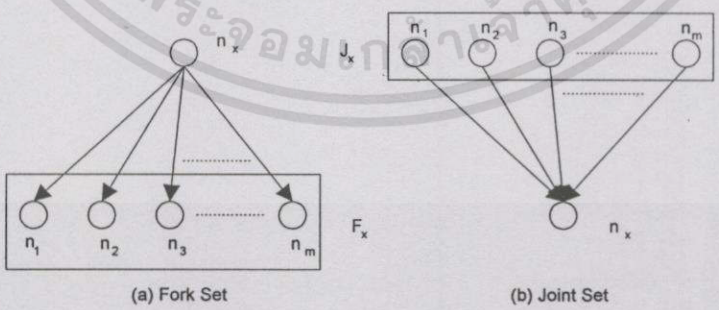
ดังนั้นจากสมการ(2.6)และ(2.7) ค่าของ Task Grain ของโหนดใดๆบน DAG จะมีค่าเท่ากับ

$$g_n = \min\{g(F_x), g(J_x)\} \quad (2.8)$$

เมื่อ n คือโหนดใดๆบน DAG ขณะเดียวกันค่าของ Granularity ของ DAG มีค่าเท่ากับ

$$g(G) = \min_{n=1,2,\ldots,m}\{g_n\} \quad (2.9)$$

และ n คือโหนดใดๆบน DAG โดยที่ $\forall n \in V$ และ m คือจำนวนโหนดบน DAG



รูปที่ 2.8 แสดง ฟอกเซต และ จอยย์เซต สำหรับโหนดn_x ใดๆ

จากรูปที่ 2.7 เราสามารถหาค่าของ Task Grain ของโหนดใดๆบน DAG ได้ เช่นพิจารณาโหนด T6 ที่ประกอบไปด้วยชุดฟอกเซต F_x = {T8, T10} และชุดของจอยย์เซต J_x = {T1, T3, T5} เมื่อ $g(F_x)$ เท่ากับ 2/8 และ $g(J_x)$ เท่ากับ 2/6 ดังนั้นค่าของ Task Grain สำหรับโหนด T6 เท่ากับ min{2/8, 2/6} และค่าของ Granularity ของ DAGในรูปที่ 2.7 เท่ากับ 1/7

บทที่ 3
อัลกอริทึมการจัดลำดับงาน สำหรับโปรแกรมแบบขนาน

บทนี้เป็นการกล่าวถึงการใช้อัลกอริทึมในการจัดลำดับงานสำหรับโปรแกรมแบบขนาน โดยใช้รูปแบบของ Acyclic Directed Task Graph หรือ DAG ในการแสดงความสัมพันธ์ระหว่างงานหรือกระบวนการต่าง ๆ ที่ปรากฏอยู่บนโปรแกรมแบบขนาน ความแตกต่างที่เห็นได้อย่างชัดเจนอย่างระหว่าง โปรแกรมแบบขนานดังที่เคยกล่าวมาบ้างแล้วในบทที่ 2 ส่วนโปรแกรมแบบกระจายจะพบได้จาก[3][4][5][6][7] โปรแกรมแบบกระจายจะใช้รูปแบบของ Undirected Task Graph [6][7][8][9]แสดงความสัมพันธ์ระหว่างงานที่อยู่บนโปรแกรมแบบกระจาย และ Undirected Task Graph จะไม่มีเรื่องของลำดับก่อนและหลังของงานเข้ามาเกี่ยวข้อง และลักษณะการทำงานของงานที่อยู่บนโปรแกรมแบบกระจายจะไม่มีทิศทาง[10][11][12][13] ส่วนโปรแกรมแบบขนานจะใช้รูปแบบของ Acyclic Directed Task Graph โดยความสัมพันธ์ระหว่างงานจะมีลำดับก่อนและหลังเข้ามาเกี่ยวข้อง และลักษณะการทำงานของงานที่อยู่บนโปรแกรมแบบขนานจะมีทิศทาง จะสังเกตได้อีกอย่างหนึ่งเกี่ยวกับคำที่ใช้ในการแก้ปัญหาของโปรแกรมแบบกระจาย และแบบขนานจะต่างกัน ในการแก้ไขปัญหาของโปรแกรมแบบกระจายมักจะใช้คำว่า "การจัดแบ่งงาน"[9] ส่วนการแก้ไขปัญหาของโปรแกรมแบบขนานมักจะใช้คำว่า "การจัดลำดับงาน หรือลำดับการทำงาน"

โดยทั่วไปแล้วค่าของความสลับซับซ้อนของเวลาของอัลกอริทึมเป็นการอ้างถึง ค่าที่ใช้ในกระทำหรือการประมวลผลของงาน และค่าที่ใช้ในกระทำจะเป็นฟังก์ชันของงานด้วย เราจะทำการอ้างถึงค่าความสลับซับซ้อนของอัลกอริทึมการจัดลำดับงานเหมือนกับฟังก์ชันของจำนวนของงานและจำนวนของตัวประมวลผล ความสลับซับซ้อนของเวลาของอัลกอริทึมที่ใช้ในการจัดลำดับงานจะถูกกำหนดด้วยโพลิโนเมียล หรืออาจจะกล่าวได้ว่าความสลับซับซ้อนของเวลาของอัลกอริทึมในการจัดลำดับงานเป็นแบบโพลิโนเมียลทาม ออฟติมัลอัลกอริทึม(Optimal algorithms)ถูกพิจารณาว่าเป็นวิธีการหนึ่งที่มีประสิทธิภาพ ถ้าหากอัลกอริทึมดังกล่าวสามารถที่ทำงานด้วยเวลาในการทำงานเป็นแบบโพลิโนเมียลทาม อัลกอริทึมที่ไม่มีประสิทธิภาพจะมีค่าความสลับซับซ้อนของเวลาของอัลกอริทึมเป็นแบบเอ็กโฟเนนเชียล(Exponentail) ปัญหาของการจัดลำดับงานของโปรแกรมแบบขนานบนระบบที่มีตัวประมวลผลมากกว่าหนึ่งตัว หรือมัลติโพรเซสเซอร์มักจะแสดงอยู่ในรูปของ NP-complete มีอัลกอริทึมการจัดลำดับงานที่มีค่าความสลับซับซ้อนของเวลาเป็นแบบโพลิโนเมียลทามจำนวนน้อยมาก ดังนั้นได้มีการกำหนดข้อบังคับที่เข้มงวดมากบนกราฟงาน ที่แสดงถึงโปรแกรมที่มีรูปแบบของตัวประมวลผลแบบขนาน

อัลกอริทึมต่างๆที่จะแสดงในบทนี้จะมีอยู่ด้วยกันในหลายๆวิธีการ วิธีการหนึ่งที่ใช้ในการจัดลำดับงานที่น่าสนใจคือการจัดกลุ่มงาน อัลกอริทึมที่ใช้ในการจัดกลุ่มงานจะมีด้วยกันหลายวิธีการ รวมทั้งผลงานวิจัยและวิทยานิพนธ์ของกระผมเองด้วย ดังจะแสดงต่อมาในภายหลังของบทนี้

3.1 อัลกอริทึมการจัดลำดับงานที่ดีที่สุด

ในส่วนนี้จะแสดงถึงประสิทธิภาพของอัลกอริทึมการจัดลำดับงานที่เหมาะสม(Optimal Scheduling Algorithms)บางอัน ที่มีค่าความสลับซับซ้อนของเวลาของอัลกอริทึมเป็นแบบโพลิโนเมียลทาม จะแสดงถึงสองวิธีการด้วย โดยที่อัลกอริทึมของทั้งสองวิธีการจะสมมติว่างานทั้งหมดจะมีค่าของเวลาที่ใช้ในการกระทำ หรือการประมวลผลที่เหมือนกันทั้งหมด และการติดต่อสื่อสารระหว่างงานจะไม่ถูกพิจารณา ตัวอย่างเช่นอัลกอริทึมที่กำหนดให้กราฟงานแสดงอยู่ในของต้นไม้(Tree) กำหนดให้การกระทำหรือการประมวลผลของงานทั้งหมดอยู่ในรูปของหนึ่งหน่วยเวลา(one time unit) โดย Hu[14] ได้ทำการแนะนำลิเนียอัลกอริทึม(A Linear algorithm) ลิเนียอัลกอริทึมจะทำการกำหนดหมายเลขระดับงานของต้นไม้เสมือนกับความยาวของเส้นทางจากโหนดไปยังโหนดที่สิ้นสุด โดยหมายเลขดังกล่าวจะเสมือนกับหมายเลขของลำดับความสำคัญ

3.1.1 อัลกอริทึมการจัดลำดับงานของกราฟงานที่มีลักษณะเป็นแบบต้นไม้(Scheduling Tree-Structured Task Graphs Algorithm) อัลกอริทึมนี้ได้รับการพัฒนามาจาก Hu[14](ฮู) หรือจะเรียกอัลกอริทึมนี้ว่า เลเวลอัลกอริทึม(Level Algorithm) อัลกอริทึมนี้สามารถทำการแก้ไขปัญหาการจัดลำดับงานในรูปแบบของเวลาที่มีลักษณะเป็นแบบเส้นตรง(Linear Time) เมื่อกราฟงานถูกแสดงอยู่ในรูปของ อิน-ฟอเรส(in-forest) ตัวอย่างของกราฟงานที่มีลักษณะเป็นแบบอินฟอเรส เมื่องานแต่ละงานจะมี อิมมีเดียท-สับเซสเซอร์(Immediate Successor) อย่างมากที่สุดอยู่หนึ่งตัว หรือกราฟงานถูกแสดงอยู่ในรูปแบบของ เอา-ฟอเรส(out-forest) ตัวอย่างของกราฟงานที่มีลักษณะเป็นแบบเอา-ฟอเรส เมื่องานแต่ละงานจะมี อิมมีเดียท-พรีดีเซสเซอร์(Immediate Predecessor) อย่างมากที่สุดอยู่หนึ่งตัว และงานทั้งหมดที่อยู่บนกราฟงานจะมีค่าของเวลาในการกระทำ หรือการประมวลผลที่เท่ากันหมด ในที่นี้เราจะใช้กราฟงานเป็นแบบอิน-ฟอเรส ที่ประกอบไปด้วยงานจำนวน n งาน ดังแสดงอยู่ในรูปที่ 3.1 โดยอัลกอริทึมที่เราจะพิจารณาในส่วนนี้เป็นแบบลิเนียสำหรับการพิจารณาการจัดลำดับที่มีความยาวที่ไม่มาก โดย ฮู ได้กำหนดความหมายของคำต่างๆที่ใช้ในอัลกอริทึมดังนี้

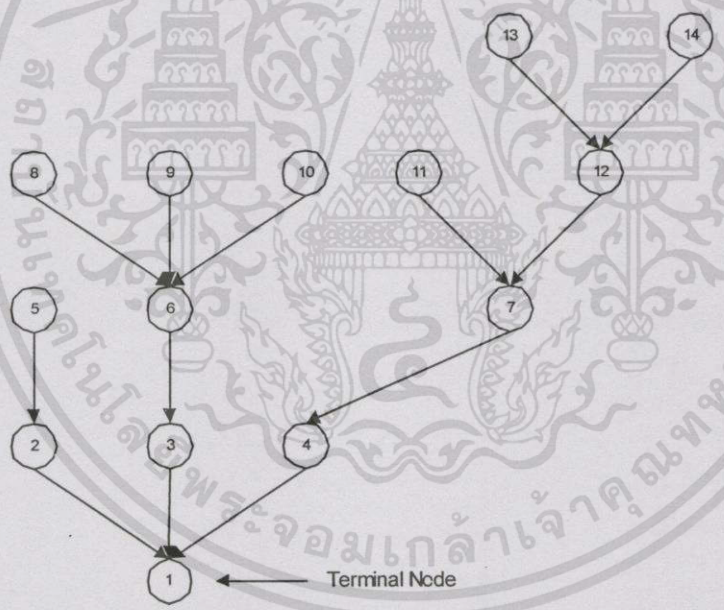
- ระดับงาน(Task Level) กำหนดให้ระดับงานของโหนด x ใดๆที่อยู่บนกราฟงานเท่ากับจำนวนของโหนดที่มากที่สุด ที่อยู่บนเส้นทางจากโหนดx ไปยังโหนดสิ้นสุดหรือ terminal node โดยจะรวมโหนดตัวมันเองด้วย(โหนดx) กล่าวคือจะมีเพียงหนึ่งเส้นทางเท่านั้นที่อยู่บนกราฟงานนั้น ตัวอย่างเช่นงานหมายเลข 2 ที่

ปรากฏอยู่บนกราฟงานรูปที่ 3.1 จะมีค่าของระดับงานเท่ากับ 2 และงานหมายเลข 6 จะมีค่าของระดับงานเท่ากับ 3

- งานที่มีสถานะพร้อม(Ready Tasks) หมายถึงงานที่จะถูกกำหนดให้อยู่ในสถานะที่พร้อมก็ต่อเมื่อ งานดังกล่าวไม่มีงานบรรพบุรุษ(task predecessor)ปรากฏอยู่ หรือเมื่อใดที่งานบรรพบุรุษทั้งหมดของงานดังกล่าว ได้ถูกกระทำหรือประมวลผลไปแล้ว

อัลกอริทึม ฮู

1. กำหนดระดับงานให้กับโหนดต่างๆที่อยู่บนกราฟงาน และค่าดังกล่าวเสมือนกับค่าของลำดับความสำคัญของโหนดแต่ละโหนด
2. เมื่อใดก็ตามที่ตัวประมวลผลพร้อมที่จะรับงานอื่นเข้ามาทำการประมวลผล หรืออยู่ในสถานะหยุดนิ่ง ให้ทำการกำหนดงานที่ยังไม่ได้รับการกระทำหรือประมวลผล ที่มีสถานะพร้อมที่จะได้รับการบริการจากตัวประมวลผล ด้วยงานที่มีค่าของลำดับความสำคัญที่มากที่สุด



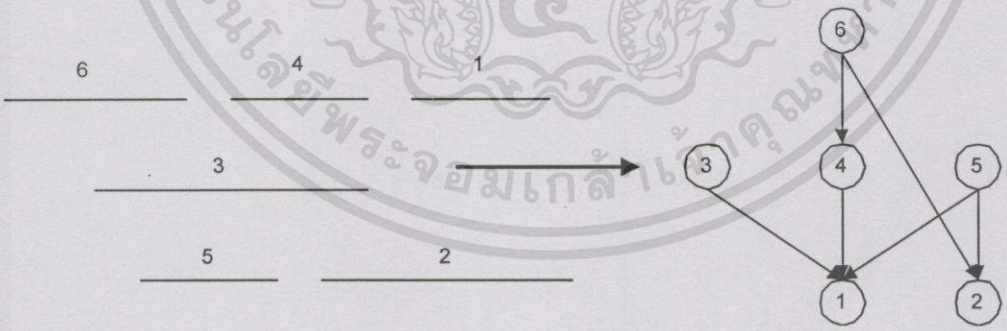
รูปที่ 3.1 กราฟงานที่มีลักษณะเป็นแบบโครงสร้างต้นไม้

อัลกอริทึมของ ฮู สามารถนำไปใช้กับกรณีของกราฟงานเป็นแบบ เอา-ฟอเรส โดยจะต้องมีการเปลี่ยนแปลงแก้ไขอัลกอริทึมข้างบนเล็กน้อย อย่างไรก็ตามอัลกอริทึมข้างบนจะไม่ให้ผลลัพธ์ที่เหมาะสม ในกรณีที่กราฟงานเป็นแบบออฟโฟซซิ่ง-ฟอเรส(opposing forest) โดยที่ออฟโฟซซิ่ง-ฟอเรสคือกราฟงานที่มีลักษณะเป็น disjoint union ของกราฟงานทั้งแบบ อิน-ฟอเรส และ เอา-ฟอ

เรส ลำดับงานหรือลำดับการทำงานสำหรับกราฟงานเป็นแบบออฟโฟซซิ่ง-ฟอเรส ได้รับการพิสูจน์แล้วว่าเป็น NP-complete

ทฤษฎี ฮู อัลกอริทึมของ ฮู สามารถใช้ในการแก้ไขปัญหาการจัดลำดับงาน โดยให้ผลลัพธ์ที่เหมาะสมสำหรับกราฟงาน ที่มีลักษณะเป็นแบบต้นไม้ได้บางแบบ

3.1.2 อัลกอริทึมการจัดลำดับงานโดยใช้ลำดับช่วงของงาน(Scheduling Interval-Ordered Tasks Algorithm) ในส่วนนี้เราจะเกี่ยวข้องกับกรณีพิเศษของระบบงานที่เรียกว่า ลำดับช่วงของงาน(interval ordered tasks) วิธีการนี้ถูกนำเสนอโดย PY[15](พาย) คำว่า interval-ordered tasks ถูกใช้ในการบ่งชี้ถึงกราฟงานที่อธิบายถึงความสัมพันธ์ก่อนหลังระหว่างงานที่อยู่ในระบบ หรือกล่าวได้ว่างานในระบบเป็นแบบลำดับช่วง กราฟงานใดๆที่จะมีลักษณะเป็นแบบ ลำดับช่วง ก็ต่อเมื่อส่วนประกอบที่อยู่ในกราฟงานสามารถถูกแมพ(mapped) ลงสู่ช่วงต่างๆที่อยู่บน real-line และสองส่วนประกอบใดๆที่อยู่บนกราฟงานถูกให้มีความสัมพันธ์ต่อกัน ก็ต่อเมื่อช่วงที่สอดคล้องกันจะต้องไม่เลื่อมล้ำกัน(overlap) โดยปกติแล้วพาร์เชียล-ออร์เดอร์เซต$(V, <)$(ได้เคยกล่าวมาแล้วในบทที่1) คือลำดับช่วงถ้าส่วนประกอบของกราฟงานไม่สามารถถูกmapped ลงสู่ช่วงที่อยู่บน real-line$(R, <)$ ได้ เช่นถ้าหาก $\forall x,y \in V$, โดยที่ $x < y$ ก็ต่อเมื่อหากช่วงที่กำหนดต่อ x มีความสมบูรณ์ก่อน ช่วงที่กำหนดต่อ y หรืออาจกล่าวได้ว่าถ้าหากช่วงที่กำหนดให้กับแต่ละประกอบ v เมื่อ $v \in V$ จะถูกแสดงด้วย a left point L(v) และ a right point R(v) แล้ว $x < y$ ก็ต่อเมื่อ $R(x) < L(y)$ ตัวอย่างของลำดับช่วง และช่วงที่สอดคล้องกับงานต่างๆที่อยู่บนกราฟงานแสดงอยู่ในรูปที่ 3.2



รูปที่ 3.2 กราฟงานที่แสดงลำดับงานของงาน

ลำดับช่วงมีโครงสร้างที่ละเอียด โดยโครงสร้างดังกล่าวจะถูกสร้างโดยอาศัยคุณสมบัติดังต่อไปนี้ ถ้า N(v) เป็นการแสดงชุดของโหนดลูกหลาน(successors)ของงานที่โหนด v แล้วสำหรับ

ลำดับช่วงใดๆ กำหนดโดย P=(V,A) และกำหนดให้ u และ v $\in$V โดยแต่ละ $N(v) \subseteq N(u)$ หรือ $N(u) \subseteq N(v)$ เมื่อ $N(v) = \{u: (v,u) \in A\}$ จากคุณสมบัติเหล่านี้แสดงให้เห็นว่า สำหรับคู่ลำดับช่วงใดๆของงาน u และ v แล้วโหนดลูกหลานของงาน u จะเป็นโหนดลูกหลานของงาน v ด้วย หรือโหนดลูกหลานของงาน v จะเป็นโหนดลูกหลานของงาน u ด้วย เมื่องานแต่ละงานสอดคล้องกับช่วงที่อยู่บน real-line แล้ว $R(u) \leq R(v)$ ส่อเป็นนัยว่าโหนดลูกหลาน x ใดๆที่เป็นของงานโหนด v ก็จะเป็นโหนดลูกหลานของงานโหนด u ด้วยเช่นกัน ในกรณีที่ $v < x$ ส่อเป็นนัยว่า $L(x) > R(v) \geq R(u)$ และส่อเป็นนัยว่า $u < v$ สำหรับโหนดลูกหลาน x ใดๆที่เป็นของ u ด้วยคุณสมบัติเหล่านี้ของลำดับช่วงของงาน ทำให้ความเป็นไปได้ที่จะใช้กรีดี้อัลกอริทึมในการหาการจัดลำดับงานที่เหมาะสม ในกรณีที่ซึ่งเมื่อเวลาที่ใช้ในการกระทำหรือประมวลผลของงานทั้งหมดเหมือนกัน ในบางครั้งถ้ามีงานมากกว่าหนึ่งงานพร้อมที่จะให้ทำการกระทำการหรือประมวลผล อัลกอริทึมดังกล่าวจะเลือกงานที่มีจำนวนของโหนดลูกหลานที่มากที่สุด ซึ่งการเลือกดังกล่าวจะทำให้ได้รับผลลัพธ์ที่ดีจากการแก้ไขปัญหาที่เหมาะสม เพราะว่าชุดของโหนดลูกหลานจะเป็นชุดของโหนดลูกหลานของงานอื่นๆที่อยู่ในสถานะที่พร้อมด้วยเช่นกัน วิธีการดังกล่าวที่แสดงด้วยอัลกอริทึมของ พาย ดังนี้

<u>อัลกอริทึม พาย</u>

1. จำนวนของโหนดลูกหลานของแต่ละโหนดถูกใช้เสมือนเป็นการกำหนดความสำคัญให้กับโหนดๆนั้น
2. เมื่อใดก็ตามที่ตัวประมวลผลพร้อมที่จะให้บริการกับกระบวนการหรืองาน หรือตัวประมวลผลอยู่ในสถานะที่หยุดนิ่ง งานตัวถัดไปที่พร้อมจะได้รับบริการจะต้องเป็นงานที่มีคุณสมบัติยังไม่ได้รับการประมวลผลเลย และจะต้องอยู่ในสถานะที่พร้อมด้วยค่าลำดับความสำคัญที่มากที่สุด

อัลกอริทึมดังกล่าวใช้ในการแก้ไขปัญหาการจัดลำดับหน่วยเวลาของการประมวลผล(The unit execution time)สำหรับลำดับช่วง (V, A) ด้วยค่าความสลับซับซ้อนของเวลาของอัลกอริทึม(Complexity Time)เท่ากับ $O(|A| + |V|)$ ครั้ง เมื่อกำหนดให้จำนวนของโหนดลูกหลานสำหรับงานทั้งหมดที่อยู่บนกราฟสามารถถูกกระทำได้เท่ากับ $O(|A|)$ ครั้ง และในการจัดประเภทของงานซึ่งจะต้องสอดคล้องกับจำนวนของโหนดลูกหลาน สามารถกระทำไปด้วยเวลาเท่ากับ $O(|V|)$ ครั้ง ในการจัดการลำดับการทำงานในขั้นตอนที่ 2 จะใช้เวลาในการกระทำเท่ากับ $O(|V|)$ ครั้ง

3.2 ฮิวริสติกอัลกอริทึมที่ใช้ในการจัดลำดับงาน

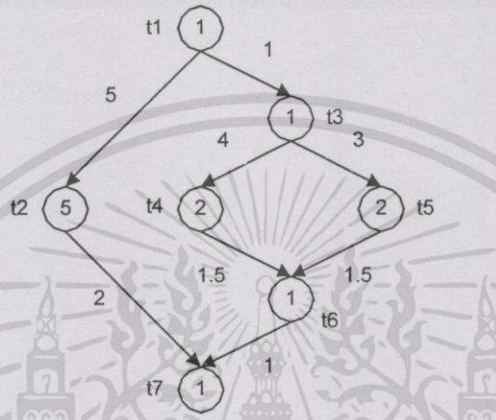
ฮิวริสติกอัลกอริทึมที่ใช้ในการจัดลำดับงาน(Heuristics algorithm for Task Scheduling) เกิดจากปัญหาการจัดลำดับงานบางปัญหาถือได้ว่าอยากต่อการควบคุม หรือจะกล่าวได้ว่าเป็นปัญหาที่ดื้อด้านในการแก้ไขปัญหา โดยปัญหาดังกล่าวได้นำไปสู่วิธีการต่างๆจำนวนมากมาย เพื่อนำมาใช้ในการแก้ไขปัญหาที่เหมาะสมโดยประมาณ เราเรียกวิธีการเหล่านี้ว่าฮิวริสติกอัลกอริทึม ประสิทธิภาพของฮิวริสติกที่ใช้ในการจัดลำดับงานขึ้นอยู่กับพารามิเตอร์ หรือปัจจัยต่างๆที่เกี่ยวข้องกับโปรแกรมแบบขนานและเครื่องเป้าหมาย ฮิวริสติกอัลกอริทึมวิธีการหนึ่งๆสามารถใช้จัดลำดับงานบางรูปแบบของกราฟงานบนเครื่องเป้าหมายได้อย่างเหมาะสม แต่ในขณะเดียววิธีการของฮิวริสติกอัลกอริทึมวิธีการนี้อาจจะไม่ให้ผลลัพธ์ที่เหมาะสมสำหรับรูปแบบของกราฟงานอื่นๆที่อยู่บนเครื่องเป้าหมายอื่นที่แตกต่างออกไป ดังนั้นจำนวนของวิธีการฮิวริสติกได้ถูกนำเสนอขึ้นมามากมาย แต่ละวิธีการอาจจะทำงานอยู่ภายใต้เหตุการณ์ที่แตกต่างกันออกไป

วิธีการของฮิวริสติกมีแบบต่างๆมากมาย แต่เนื่องจากวิทยานิพนธ์ฉบับนี้เป็นวิธีการของฮิวริสติกอัลกอริทึมแบบหนึ่ง ดังนั้นจะนำเสนอวิธีการของฮิวริสติกที่ใช้ในการจัดแบ่งงาน และกำหนดลำดับงาน ของผลงานวิจัยต่างๆที่ผ่านมา การจัดแบ่งงานคือการจัดงานต่างๆที่อยู่บนกราฟงานออกเป็นกลุ่มๆ เนื่องจากการจัดแบ่งงานที่อยู่บนกราฟงานสามารถทำได้อย่างทันทีทันใด เพื่อทำการแก้ไขปัญหาการจัดแบ่งงานของการจัดลำดับงานหรือกระบวนการ โดยงานที่อยู่ภายในกลุ่มเดียวกันจะถูกจัดลำดับให้ทำงานอยู่บนตัวประมวลผลที่เหมือนกัน

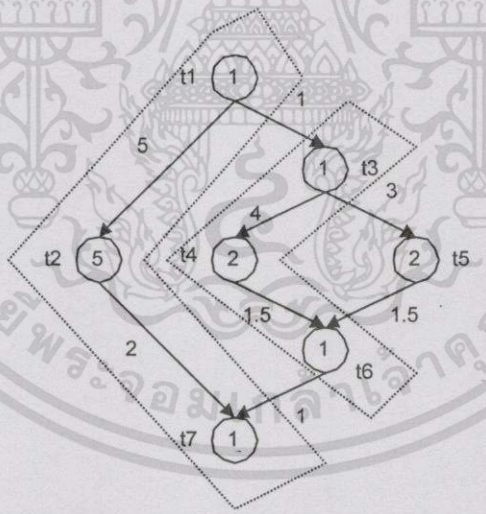
3.2.1 อัลกอริทึมการจัดกลุ่มงาน(Clustering Algorithm) การทำจัดกลุ่มงานสามารถถูกกำหนดเหมือนกับ กระบวนการในการวางโหนดต่างๆของกราฟงานด้วยลาเบลของกลุ่มงานต่างๆ โดยงานทั้งหมดที่อยู่ภายในกลุ่มงานเดียวกันจะต้องถูกประมวลผลที่ตัวประมวลผลตัวเดียวกัน แต่เนื่องจากปัญหาในการจัดกลุ่มงานที่เหมาะสมโดยอาศัยกราฟงานทั่วไปถือได้ว่าเป็นปัญหาแบบ NP-complete ด้วยผลลัพธ์ทำให้เป็นที่คาดหวังได้ว่าวิธีการจัดกลุ่มงานที่เหมาะสมจะนำไปสู่การจัดลำดับงานที่เหมาะสมที่เหมือนกันด้วย และยังจะได้ค่าความสลับซับซ้อนของเวลาของอัลกอริทึมที่เหมือนกัน

โดยปกติแล้ว การจัดกลุ่มงานคือปัญหาของการพิจารณาในการวางงานต่างๆลงบนชุดของกลุ่มงาน(a set of clusters)จำนวน k กลุ่มด้วยกัน$\{C_1, C_2, C_3, \ldots, C_k\}$ โดยมีการกำหนดให้งานต่างๆที่อยู่ในกลุ่มงานเดียวกันจะต้องถูกประมวลผลที่ตัวประมวลผลตัวเดียวกัน ถ้าหากงานสองงานที่อิสระต่อกันถูกวางอยู่บนกลุ่มงานเดียวกัน เราจะเรียกกลุ่มงานดังกล่าวว่ากลุ่มงานแบบนอน-ลิเนีย(nonlinear clustering) ในทางตรงกันข้ามเราจะเรียกว่า กลุ่มงานแบบลิเนีย(linear clustering) ตัวอย่างให้พิจารณาจากรูปที่ 3.3 เป็นการแสดงกราฟงาน รูปที่ 3.4 แสดงกลุ่มงานแบบลิเนีย และรูปที่

3.5 เป็นการแสดงกลุ่มงานแบบนอน-ลิเนีย จากรูปกลุ่มงานที่เป็นแบบลิเนียจะประกอบไปด้วยกลุ่มงานต่างๆจำนวนสามกลุ่มด้วยกัน $\{t_1, t_2, t_7\}$, $\{t_3, t_4, t_6\}$, และ $\{t_5\}$ สำหรับกรณีของกลุ่มงานแบบนอน-ลิเนียจะประกอบไปด้วยกลุ่มงานจำนวนสามกลุ่มเช่นกันดังนี้ $\{t_1, t_2\}$, $\{t_3, t_4, t_5, t_6\}$, และ $\{t_7\}$ จะสังเกตพบว่าในกรณีของกลุ่มงานแบบนอน-ลิเนีย งาน t_4 และ t_5 จะเป็นอิสระต่อกันจะถูกวางลงบนกลุ่มงานที่เหมือนกัน รูปที่ 3.6 เป็นการแสดงแกรฑชาร์ทสำหรับการจัดแบ่งงานและลำดับงานแบบนอน-ลิเนีย



รูปที่ 3.3 กราฟงาน หรือ DAG

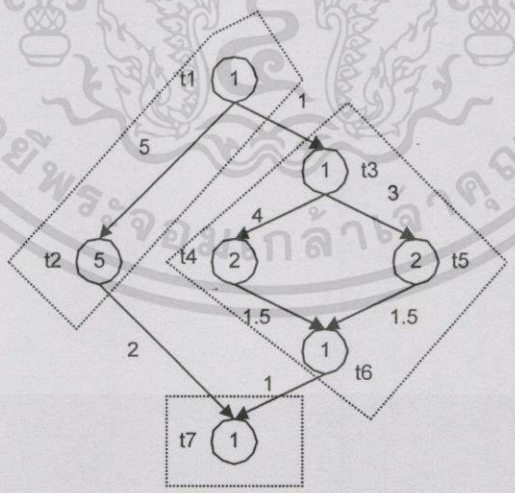


รูปที่ 3.4 กลุ่มงานแบบลิเนีย ที่ได้จากการแบ่งงานบนกราฟงาน

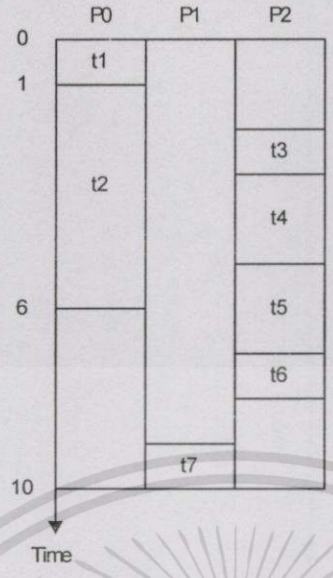
การจัดทำกลุ่มงานสามารถนำไปใช้ในขั้นตอนของการจัดลำดับงานบนสถาปัตยกรรมตัวประมวลแบบขนาน(Parallel Processor Architecture) ด้วยภายใต้ขอบเขตจำนวนของตัวประมวลผลอย่างมีจำกัดจำนวน โดยผลดังกล่าวอาจจะได้การเชื่อมต่อกันระหว่างงานต่างๆที่สมบูรณ์แบบหรืออาจจะไม่สมบูรณ์แบบก็ได้ ในการจัดทำกลุ่มงานสำหรับการจัดลำดับงานเป็นวิธีการที่

ประกอบไปด้วยสองขั้นตอน(Two-phase) ขั้นตอนแรกเป็นขบวนการของการทำจัดกลุ่มงาน ขั้นตอนที่สองเป็นการจัดลำดับกลุ่มงานต่างๆให้สามารถทำงานบนตัวประมวลผลต่างๆที่มีอยู่ เช่นในระบบมีตัวประมวลผลจำนวน m ตัว โดยในขั้นตอนที่สองจะต้องถูกกระทำในลักษณะขั้นตอนการเรียงลำดับอย่างเหมาะสมดังนี้ ขั้นตอนแรกการรวมกลุ่มงาน(Cluster Merging) ถ้าหากจำนวนของกลุ่มงานมากกว่าจำนวนของตัวประมวลผลที่เป็นไปได้ ดังนั้นจะต้องทำการรวมกลุ่มงานเหล่านั้นให้เหลือเพียง m กลุ่มงาน(เพราะในระบบขณะนี้ประกอบไปด้วยตัวประมวลผลจำนวน m ตัว) ขั้นตอนที่สองการทำฟิสิคัลแมปปิ้ง(Physical Mapping) ถ้าหากเครื่องเป้าไม่ได้มีการเชื่อมต่อกันอย่างเต็มรูปแบบหรือทุกๆหนึ่งคู่ของเครื่องเป้าหมายจะต้องมีเส้นทางเชื่อมต่อหากัน การทำแมปปิ้งหรือการวางกลุ่มงานต่างๆลงบนตัวประมวลผลต่างๆ จะถูกพิจารณา อย่างเช่นค่าใช้จ่ายในการติดต่อสื่อสารโดยรวมระหว่างกลุ่มงานจะต้องถูกลดลงให้น้อยที่สุด ขั้นตอนที่สามลำดับในการกระทำหรือการประมวลผลของงาน(Task Execution Ordering) ถ้าหากงานในแต่ละกลุ่มงานถูกกำหนดให้มีความสัมพันธ์โดยพาร์เชียล-ออร์เดอร์ ดังนั้นลำดับของกระทำของงานจะถูกพิจารณาต่อความสัมพันธ์ของลำดับก่อนหลังทั้งหมด

ฮิวลิสติกอัลกอริทึมวิธีหนึ่งมักจะได้รับกล่าวถึงบ่อยๆ คือวิธีการของ Sarkar's Heuristic algorithm[16] ซาร์การ์(Sarkar)ได้เรียกวิธีการแบ่งงานว่าการทำ internalization pre-pass วิธีการของ Sarkar ถูกนำไปใช้ในแก้ไขปัญหาการจัดกลุ่มงานอย่างง่ายๆ วิธีการนี้จะกล่าวต่อมาในภายหลังก่อนที่จะมีการนำเสนออัลกอริทึมสำหรับการจัดกลุ่มงาน จะทำการแสดงคำนิยามต่างๆที่ใช้ในการจัดกลุ่มงานสำหรับการจัดลำดับของโปรแกรมแบบขนานก่อน



รูปที่ 3.5 กลุ่มงานแบบนอน-ลิเนีย ที่ได้จากการแบ่งงานบนกราฟงาน

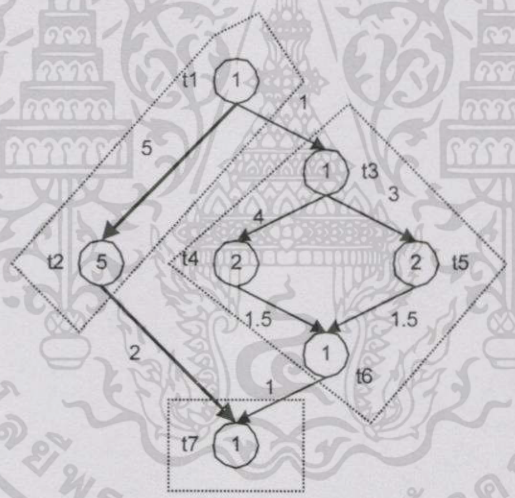


รูปที่ 3.6 แกรฟทชาร์ทสำหรับการจัดแบ่งและลำดับงานแบบนอน-ลิเนีย

3.2.2 นิยามทั่วๆไปที่ใช้ในอัลกอริทึมการจัดกลุ่มงาน อัลกอริทึมที่ใช้ในการจัดกลุ่มงานโดยปกติจะเริ่มต้นจาก การทำอินนิเชียล(Initial)กลุ่มงาน จากนั้นให้ทำการปรับปรุงกลุ่มงาน(Clustering Refinements)เพื่อให้ตรงตามเป้าหมายที่ได้ออกแบบเอาไว้ ฮิวลิสติกอัลกอริทึมการจัดกลุ่มงานจะต้องเป็นแบบนอน-แบล็คแทรกกิ้ง(Non Black Tracking) เพื่อต้องการหลีกเลี่ยงการเกิดความสลับซับซ้อนที่สูง อย่างเช่นเมื่อใดที่กลุ่มงานได้ถูกรวมเข้าด้วยกันในขั้นตอนของการปรับปรุงกลุ่มงาน กลุ่มงานเหล่านั้นไม่สามารถที่จะทำการไม่รวบรวม(Unmerge)ย้อนกลับไปได้ โดยปกติแล้วในขั้นตอนของการปรับปรุงกลุ่มงานเป็นการรวมกลุ่มงานสองกลุ่มงานเข้าด้วยกัน และทำให้อาร์คที่ใช้ในการเชื่อมต่อระหว่างกลุ่มงานเท่ากับศูนย์ หรือเรียกว่าการทำซีโร(Zero) ในการทำซีโรต่อค่าใช้จ่ายในการติดต่อสื่อสารบนอาร์คระหว่างการรวมงานสองกลุ่มเข้าด้วยกันเป็นสิ่งจำเป็นเพราะว่าโหนดเริ่มต้นและโหนดสิ้นสุดจะถูกจัดลำดับให้ทำงานอยู่บนตัวประมวลผลตัวเดียวกัน ทำให้ค่าใช้จ่ายในการติดต่อสื่อสารระหว่างโหนดที่อยู่คนละกลุ่มงานถูกละทิ้งไป

ในขั้นตอนของการปรับปรุงกลุ่มงาน อาร์คที่ใช้ในการเชื่อมต่อระหว่างงานสองกลุ่มงานจะถูกเลือกสำหรับการทำ edge-zeroing โดยปกติเกณฑ์ที่ใช้ในการเลือกอาร์คคือการเลือกอาร์คที่สามารถลดเวลาแบบขนาน(Parallel Time/PT)ของการจัดลำดับงานลงได้ PT ถูกกำหนดจากความยาวของเส้นทางที่ยาวที่สุดในกราฟงานที่แสดงลำดับงานหรือลำดับการทำงาน หรืออาจจะกล่าวได้ว่า PT ของลำดับงานเท่ากับเวลาที่เสร็จสิ้นสมบูรณ์ที่อยู่ภายใต้ข้อสมมติฐานที่ว่าจำนวนของกลุ่มงานจะต้องไม่มากไปกว่าจำนวนของตัวประมวลผล นอกเหนือไปจากปัจจัยหรือพารามิเตอร์ในการเลือกอาร์คในขั้นตอนของการทำการปรับปรุงกลุ่มงาน ยังมีพารามิเตอร์อื่นๆอีกที่ใช้ในการปรับปรุง

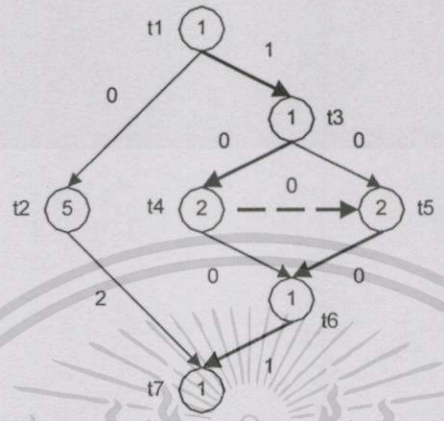
กลุ่มงานเช่น เส้นทางวิกฤต(Critical Path/CP) ของกลุ่มงานในกราฟงาน และการเรียงลำดับที่เด่นชัด(Dominant Sequence/DS)ของลำดับงานในกราฟงาน เส้นทางวิกฤต(PS)เป็นเส้นทางที่ยาวที่สุดในกราฟงาน ส่วนการเรียงลำดับที่เด่นชัด(DS)เป็นเส้นทางที่ยาวที่สุดของการจัดลำดับงานบนกราฟงาน หรือเป็นความยาวของเส้นทางที่เท่ากับเวลาแบบขนานจริงๆสำหรับการจัดลำดับงาน อาจจะกล่าวได้ว่าเส้นทางวิกฤตเป็นพารามิเตอร์ของกราฟงานเท่านั้น ในขณะที่การเรียงลำดับที่เด่นชัดเป็นพารามิเตอร์ของการจัดลำดับงานของกราฟงาน สำหรับตัวอย่างเช่นจากรูปที่ 3.5 แสดงกลุ่มงานที่อยู่บนกราฟงานจะมีเส้นทางวิกฤต CP = $<t_1, t_2, t_7>$ ด้วยความยาวเท่ากับ 5 ในขณะเดียวกันการเรียงลำดับที่เด่นชัด DS ของการจัดลำดับงาน = $<t_1, t_3, t_4, t_5, t_6, t_7>$ ดังปรากฏอยู่ในรูปที่ 3.6 ซึ่งความยาวเท่ากับ 10 คือค่าของเวลาแบบขนานสำหรับการจัดลำดับงาน รูปที่ 3.7 และ 3.8 เป็นการแสดง CP และ DS จากรูปดังกล่าวจะสังเกตได้ว่าในกรณีของกลุ่มงานแบบลิเนีย ค่าของ DS และค่าของ CP ที่อยู่บนกราฟงานจะให้ค่าที่เหมือนกันหมด โดยทั่วๆไปแล้วความยาวของ DS จะต้องมากกว่าหรือเท่ากับ ความยาวของ CP ในกรณีเฉพาะของกลุ่มงานแบบนอน-ลิเนียเท่านั้นที่จะให้ผลลัพธ์ที่ต่างกัน



รูปที่ 3.7 แสดงเส้นทางวิกฤต(CP)ของกลุ่มงานในกราฟงาน

ในตัวอย่างที่จะแสดงอยู่ในแต่ละอัลกอริทึมดังต่อไปนี้ เราจะทำการสมมติว่าความเร็วของตัวประมวลผลทั้งหมดและอัตราการขนถ่ายข้อมูลของแต่ละเส้นทางการขนถ่ายข้อมูลหรือลิ้งค์เท่ากับหนึ่งหน่วย โดยหนึ่งหน่วยนี้หมายถึงหนึ่งหน่วยเวลาของจำนวนการคำนวณงานหรือการประมวลผลของงานที่ตัวประมวลผล และหนึ่งหน่วยเวลาของขนาดการขนถ่ายข้อมูลที่เส้นทางการขนถ่ายข้อมูล วิธีการหรืออัลกอริทึมในการทำ edge-zeroing ก่อให้เกิดขั้นตอนที่สำคัญในกระบวนการทำแบ่งกลุ่มงานสำหรับอัลกอริทึมการจัดกลุ่มงาน ในการทำ edge-zeroing ได้ก็ต่อเมื่อส่วน

ประกอบที่อยู่บนกราฟงานจะต้องมีการเชื่อมต่อหากัน กล่าวคืองานสองงานที่อยู่บนกราฟงานจะต้องมีเส้นทางการขนถ่ายข้อมูลเชื่อมหากัน ในขณะเดียวกันการทำ edge-zeroing จะไม่ทำการรวมงานสองงานเข้าด้วยกัน หากงานสองงานดังกล่าวไม่มีเส้นทางการขนถ่ายข้อมูลต่อกัน



รูปที่ 3.8 แสดงการเรียงลำดับที่ชัดเจน(DS)ของกราฟงาน

ในการทำ edge-zeroing ของการทำจัดแบ่งกลุ่มงานจะก่อให้เกิดกราฟงานที่เริ่มต้น(Initial Task Graph)จำนวน k ที่เรียงลำดับกันที่แยกออกจากกัน จะถูกแสดงด้วย $G_r = (T, <, [D_{ij}]_r, [Ai])$ เมื่อ $0 \leq r \leq k$ กำหนดได้ op_r เป็นการแสดงการดำเนินการ(Operation)ที่เกิดขึ้นจากการแก้ไขเมทริกการติดต่อสื่อสารข้อมูล$[D_{ij}]_{r-1}$ ต่อเมทริก $[D_{ij}]_r$ โดยการทำ edge-zeroing ต่อค่าใช้จ่ายในการติดต่อสื่อสารของอาร์คในเมทริก$[D_{ij}]$ ต่อไปนี้เป็นการกำหนดความหมายของสัญญลักษณ์ต่างๆที่ใช้ฮิวลิสติกอัลกอริทึม

- CH แสดงถึงวิธีการของฮิวลิสติกอัลกอริทึมในการแบ่งงาน โดยการเลือกอาร์คให้เท่ากับศูนย์
- SH แสดงถึงวิธีการของฮิวลิสติกอัลกอริทึมในการจัดลำดับงานที่อยู่บนพื้นฐานวิธีการของ CH
- SG_r แสดงวิธีการจัดลำดับงานที่อยู่บนกราฟงานที่เสร็จสิ้นสมบูรณ์ในขั้นตอนของการดำเนินการของ op_r โดยจะต้องสอดคล้องกับวิธีการของ CH
- CP_r เป็นชุดของโหนดทั้งหมดที่อยู่ในเส้นทางวิกฤตของ G_r
- DS_r เป็นการเรียงลำดับที่ชัดเจนที่เสร็จสิ้นสมบูรณ์ในขั้นตอนของการดำเนินการของ op_r ซึ่งก็คือเส้นทางวิกฤตของ SG_r

- bot_level(t_i, r) คือเส้นทางที่ยาวที่สุดระหว่างโหนด t_i และโหนดล่างสุด(bottom node)ที่อยู่ใน SG_r โดยจะรวมถึงค่าใช้จ่ายในการติดต่อสื่อสารและค่าใช้จ่ายในการประมวลผลในเส้นทาง
- top_level(t_i, r) คือความยาวของเส้นทางที่ยาวที่สุดระหว่างโหนด t_i และโหนดที่อยู่บนสุด(top node)ที่อยู่ใน SG_r จะรวมถึงค่าใช้จ่ายในการติดต่อสื่อสารและค่าใช้จ่ายในการประมวลผลในเส้นทาง แต่จะไม่รวมเวลาในการกระทำของ t_i
- PT_r เป็นการแสดงเวลาแบบขนานที่เสร็จสิ้นสมบูรณ์ในการดำเนินการของ op_r

$$PT_r = \sum_{t_i \in DS_r} A_i + \sum_{t_i, t_j \in DS_r} D_{ij}$$
$$= top\_level(t_i r) + bot\_level(t_i, r), t_i \in DS_r \qquad 3.1$$

ปัญหาในการหากลุ่มงานที่เหมาะสมได้ถูกแสดงอยู่ในรูปของ NP-complete โดยหน้าที่หลักในการแก้ไขปัญหาของวิธีการหรืออัลกอริทึมดังกล่าวคือต้องการใช้เวลาแบบขนานให้น้อยที่สุด แม้ว่าจะไม่จำกัดจำนวนของตัวประมวลผลที่ใช้เพื่อให้เกิดการเชื่อมต่อกันที่สมบูรณ์แบบบนเครื่องเป้าหมาย หรือทุกๆหนึ่งคู่ของเครื่องเป้าหมายจะต้องมีเส้นทางการเชื่อมต่อหากัน วิธีการของฮิวลิสติกอัลกอริทึมแบบต่างๆได้ถูกนำเสนอ เพื่อทำการแก้ไขปัญหาในรูปแบบต่างๆของปัญหาในการจัดกลุ่มงาน โดยในส่วนนี้จะนำเสนอสามวิธีการด้วยกัน ซึ่งแต่ละวิธีการได้อาศัยการทำ edge-zeroing ทั้งสามวิธีการ ได้แก่วิธีการของ Kim and Brown, Sarkar, และ Yang and Gerasoulis โดยวิธีการทั้งสามจะถูกกำหนดอยู่ใต้สมมติฐานที่ว่า เครื่องเป้าหมายจะต้องมีการเชื่อมต่อกันอย่างสมบูรณ์แบบ จำนวนของตัวประมวลผลจะต้องมากกว่าหรือเท่ากับจำนวนของกลุ่มงาน

3.2.3 อัลกอริทึมของคิมแอนบราวน์(Kim and Brown's Algorithm) วิธีการของอัลกอริทึมของคิมแอนบราวน์[17]ถูกจัดให้อยู่ในประเภทของอัลกอริทึมในการจัดกลุ่มงานแบบลิเนียกล่าวคือจะไม่มีกลุ่มงานใดเลยที่จะมีคู่ของงานที่เป็นอิสระต่อกัน รายละเอียดของอัลกอริทึมของคิมแอนบราวถูกแสดงดังต่อไปนี้

<u>อัลกอริทึม คิมแอนบราวน์</u>

1. ในตอนเริ่มต้นกำหนดให้อาร์คทั้งหมดถูกมาร์ค(marked) แสดงถึงอาร์คเหล่านั้นยังมิได้ถูกกระทำ(unexamined) และงานแต่ละงานจะถูกกำหนดให้อยู่ในกลุ่มงานที่แยกออกจากกัน กล่าวคืองานหนึ่งงานจะอาศัยอยู่บนหนึ่งกลุ่มงาน หรือหนึ่งกลุ่มงานจะมีเพียงงานหนึ่งงานเท่านั้นในตอนเริ่มต้น
2. วนลูปกระทำซ้ำๆกัน จนกระทั่งอาร์คทั้งหมดได้รับการอันมาร์ค(unmarked)แล้ว

- กำหนดให้ CP_r เป็นเส้นทางที่ยาวที่สุดที่ประกอบไปด้วยอาร์คที่ยังไม่ได้รับการกระทำใดๆ โดยการใช้ฟังก์ชันของค่าใช้จ่ายเป็นตัวพิจารณา

$$\cos t = \omega_1 * \sum A_i + (1-\omega_1)(\omega_2 \sum D_{ij} + (1-\omega_2) * \sum D_{ij}^{adj}) \qquad 3.2$$

- จัดโหนดต่างๆที่อยู่ในเส้นทางให้อยู่ในรูปของกลุ่มงานและกำหนดให้ค่าใช้จ่ายของอาร์คที่อยู่ในกลุ่มงานนั้นเท่ากับศูนย์
- ทำการอันมาร์คอาร์คทั้งหมดที่อยู่ภายใน CP_r ซึ่งแสดงถึงอาร์คเหล่านั้นได้รับการกระทำแล้ว

พารามิเตอร์ ω_1 และ ω_2 ในสมการที่ 3.2 เป็นปัจจัยที่ใช้ในการทำ normalization ผลรวมที่ได้จากสมการดังกล่าวคือโหนดทั้งหมดที่อยู่ในเส้นทาง และ D_{ij}^{adj} เป็นค่าใช้จ่ายในการติดต่อสื่อสารที่อยู่บนอาร์คระหว่างโหนดที่อยู่ในเส้นทางกับโหนดอื่นๆทั้งหมดที่ใกล้เคียงกัน(adjacent node)แต่อยู่ภายนอกเส้นทาง ถ้ากำหนดให้ $\omega_1 = ½$ และ $\omega_2 = 1$ สมการที่ 3.2จะทำการลดความยาวของเส้นทางวิกฤต ภายใต้ข้อสมมติฐานนี้อัลกอริทึมของคิมแอนบราวมีเป้าหมายเพื่อที่จะทำการหากลุ่มงานที่เป็นแบบลิเนียด้วยค่าเวลาแบบขนานที่น้อยที่สุด ภายหลังจากที่ทำการหากลุ่มงานต่างๆเสร็จเรียบร้อยแล้ว ขั้นตอนที่สองในการกำหนดลำดับงานจะต้องทำการจัดแบ่งกลุ่มงานต่างๆลงบนแต่ละตัวประมวลผล จริงๆแล้วในขั้นตอนนี้ไม่มีความจำเป็น เพราะว่ากลุ่มงานที่ได้รับจากอัลกอริทึมเป็นกลุ่มงานแบบลิเนียอยู่แล้ว ในการวิเคราะห์ความสลับซับซ้อนของเวลาของอัลกอริทึมสามารถพิจารณาได้ดังนี้ ในการหาเส้นทางที่ยาวที่สุดเท่ากับ O(n+e) เมื่อ e เป็นจำนวนของอาร์คและ n เป็นจำนวนของโหนดที่อยู่ในกราฟงาน โดยจำนวนของกลุ่มงานมากที่สุดจะได้รับเท่ากับ n ดังนั้นจำนวนของขั้นตอนในการกระทำจะถูกจำกัดขอบเขตโดย n ดังนั้นโดยสรุปแล้วค่าความสลับซับซ้อนของเวลาของอัลกอริทึมคิมแอนบราวเท่ากับ O(n(n+e))

3.2.4 อัลกอรีทึมของซาการ์(Sarkar's Algorithm) กลุ่มงานที่ได้รับจากวิธีการนี้ไม่เสมอไปที่จะได้รับกลุ่มงานเป็นแบบลิเนีย ดังนั้นอัลกอริทึมของซาการ์[16]ถูกจัดอยู่ในประเภทของอัลกอริทึมการจัดกลุ่มงานแบบนอน-ลิเนีย รายละเอียดของอัลกอริทึมถูกแสดงดังต่อไปนี้

<u>อัลกอริทึม ซาร์การ์</u>

1. เริ่มต้นกำหนดให้อาร์คทั้งหมดถูกมาร์คเอาไว้ ถึงแสดงถึงอาร์คเหล่านั้นยังไม่เคยได้การกระทำ และกำหนดให้งานแต่ละงานถูกวางอยู่บนกลุ่มงานที่แยกออกจากกัน กล่าวคือแต่ละกลุ่มงานจะประกอบไปด้วยงานเพียงหนึ่งงาน

2. ให้ทำการเรียงลำดับอาร์คที่อยู่บนกราฟงานด้วยลำดับจากมากไปหาน้อย โดยค่าใช้จ่ายในการติดต่อสื่อสารเป็นหลักในการเรียงลำดับของอาร์ค
3. กระทำซ้ำๆกัน จนกระทั่งอาร์คทั้งหมดได้รับการอันมาร์คแล้ว
    - ทำการ edge-zeroing ให้กับอาร์คที่ยังไม่รับการอันมาร์คที่สูงที่สุด ในลำดับของอาร์คที่ได้มีการจัดเรียงไว้ในข้อที่ 2 โดยมีข้อกำหนดว่าค่าของ PT จะต้องไม่เพิ่มขึ้น
    - ให้ทำการอันมาร์คกับอาร์คได้รับการกระทำแล้ว
    - เมื่อใดที่กลุ่มงานสองกลุ่มถูกรวมกัน งานที่อยู่ในกลุ่มงานเหล่านั้นจะถูกเรียงลำดับให้สอดคล้องกับ bot_level ที่มากที่สุด

จะสังเกตพบว่าเป้าหมายหลักของการทำจัดกลุ่มงานของอัลกอริทึมของซาร์การ์คือการทำให้เวลาแบบขนานน้อยที่สุด ในขณะเดียวกันวิธีการของซาร์การ์ยังสอดคล้องกับเป้าหมายในการจัดลำดับงานคือการทำให้ค่าใช้จ่ายในการติดต่อสื่อสาร(Communication Cost/CC)น้อยที่สุด โดยไม่ได้ทำให้ค่าของ PT สูงขึ้น การหาค่าของ PT ถูกแสดงดังต่อไปนี้

$$PT = \sum_{t_i \in DS} A_i + \sum_{t_i, t_j \in DS} D_{ij} \leq \sum_{t_i \in DS} A_i + CC \qquad 3.3$$

ในการวิเคราะห์ค่าความสลับซับซ้อนของเวลาของอัลกอริทึม โดยพิจารณาจากในแต่ละขั้นตอน ค่าใช้จ่ายในการคำนวณของระดับต่างๆเท่ากับ O(n+e) ของแต่ละขั้นตอน เนื่องจากขั้นตอนทั้งหมดประกอบไปด้วย e ขั้นตอน ดังนั้นค่าความสลับซับซ้อนของเวลาของอัลกอริทึมเท่ากับ O(e(n+e)) โดยอัลกอริทึมนี้จะต้องถูกกำหนดอยู่ใต้ข้อสมมติฐานที่ว่า จำนวนของตัวประมวลผลจะต้องมากกว่าจำนวนของกลุ่มงาน และจะต้องมีการเชื่อมต่อกันระหว่างตัวประมวลผลทุกๆคู่

3.2.5 โดมิเนนท์-ซีเควั้น คลัสเตอร์ริ่งอัลกอริทึม(The Dominant-Sequenuce Clustering Algorithm/DS) วิธีการนี้ถูกนำเสนอโดย Yang และ Gerasoulis[18]โดยอาศัยฮิวริสติกของการเรียงลำดับที่ชัดเจน(DS) ความคิดหลักๆของวิธีการนี้อยู่ที่การกำหนดการทำ DS ในแต่ละขั้นตอนและการทำ edge-zeroing กับอาร์คทั้งหมดที่อยู่ในลำดับที่ต่อเนื่องกัน การทำ edge-zeroing ของ DS จะทำให้ช่วยลดความยาวที่อยู่ใน DS และยังช่วยลดเวลาแบบขนานลงได้ รายละเอียดอัลกอริทึมของ DS ถูกแสดงดังนี้

<u>อัลกอริทึม Yang and Gerasoulis</u>

1. เริ่มต้นกำหนดงานแต่ละงานถูกวางอยู่ในกลุ่มงานที่แยกจากกัน หรือหนึ่งกลุ่มงานจะมีเพียงงานหนึ่งงานปรากฏอยู่ และทำการมาร์คอาร์คทั้งหมด ซึ่งหมายถึงอาร์คเหล่านั้นยังไม่ได้ถูกกระทำ
2. ในตอนเริ่มต้นกำหนดให้ $r = 0$
3. ทำการคำนวณ DS เริ่มต้น ถูกกำหนดให้เป็น DS_0
4. วนลูปซ้ำๆจนกระทั่ง อาร์คทั้งหมดได้ถูกอันมาร์คหมดแล้ว
    - ทำ edge-zeroing ใน DS_r โดยไม่ได้ทำให้ค่าของ PT เพิ่มขึ้น
    - ทำการอันมาร์ค กับอาร์คที่ได้รับการกระทำแล้ว
    - กำหนดให้ $r = r + 1$
    - ทำการหา DS_r ใหม่

กำหนดให้โหนดหนึ่งโหนด $t_i \in SG_r$ และ $t_i \in DS_r$ ก็ต่อเมื่อเงื่อนไขดังต่อไปนี้เป็นจริง $top\_level(t_i, r) + bot\_level(t_i, r) = PT$ แสดงให้เห็นว่า DS_r ในแต่ละขั้นตอนมีความต้องการคำนวณหา top_level และ bot_level ที่เหมือนๆกัน ซึ่งทำให้เกิดค่าความสลับซับซ้อนของเวลาเท่ากับ O(n+e) ภายในแต่ละขั้นตอน ดังนั้นค่าความสลับซับซ้อนของเวลาของอัลกอริทึมเท่ากับ O(n(n+e))

ต่อมา Yang และ Gerasoulis ได้ทำการพัฒนาอัลกอริทึมใหม่ที่มีชื่อว่า อัลกอริทึมการจัดกลุ่มงานที่มีการเรียงลำดับอย่างชัดเจนหรือ DSC (Dominant Sequence Clustering Algorithm/DSC) โดยวิธีการของ DSC[19][20] ดังกล่าวจะทำการเปลี่ยนแปลงวิธีการหา DS_r ใหม่ เพื่อให้ DSC ไม่ต้องมีการคำนวณหา DS_r ใหม่ในทุกขั้นตอนเหมือนกับวิธีการของ DS (ในแต่ละขั้นตอนการคำนวณหา top_level และ bot_level) หรืออาจจะกล่าวได้ว่าจะมีค่าของ DS_r ที่สามารถนำไปใช้ได้ในทุกขั้นตอนโดยไม่ต้องมีการคำนวณหาค่า DS_r ใหม่ในทุกขั้นตอน ผลดังกล่าวทำให้ค่าความสลับซับซ้อนของเวลาของอัลกอริทึมลดลงจากเดิมมาเป็น O((n+e) log n) รายละเอียดของวิธีการ DSC จะไม่กล่าวไว้ที่นี้เพราะวิธีการของอัลกอริทึมจะคล้ายกับของเดิม

3.3 อัลกอริทึมการแบ่งงานและการรวบรวมเม็ดเนื้องาน

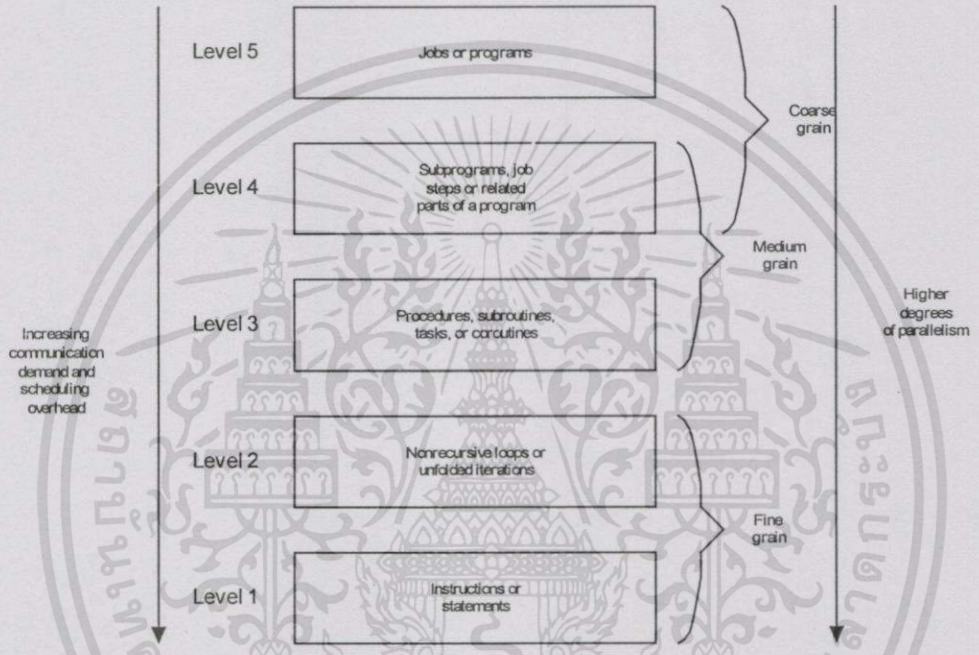
อัลกอริทึมการแบ่งงานและการรวบรวมเม็ดเนื้องาน(Task Partitioning and Grain Packing Algorithm) เป็นวิธีการจัดแบ่งงานและการจัดลำดับการทำงานอีกวิธีการหนึ่งที่น่าสนใจ คำๆหนึ่งที่มักจะพบได้ในอัลกอริทึมนี้คือคำว่า ขนาดเม็ดเนื้องาน(Grain Size)[2][3] และมักจะมีคำถามเสมอว่าขนาดเม็ดเนื้องานคืออะไร ขนาดเม็ดเนื้องานจะขึ้นอยู่กับจำนวนของงานที่ถูกรวบรวมเข้าด้วยกัน จนกลายเป็นกลุ่มของงาน บางเม็ดเนื้องานอาจจะมีขนาดใหญ่และบางเม็ดเนื้องานอาจจะ

มีขนาดเล็ก ทำให้เม็ดเนื้องานอาจจะประกอบไปด้วยงานหนึ่งงานหรือจำนวนงานที่มากกว่าหนึ่งงาน ดังนั้นขนาดของเม็ดเนื้องานอาจจะถูกเพิ่มจำนวนงานให้มากขึ้น หรือทำการดึงงานออกจากเม็ดเนื้องานทำให้จำนวนงานน้อยลง งานหนึ่งงานอาจจะมีขนาดเล็ก เช่นมีเพียงหนึ่งคำสั่ง หรืองานหนึ่งงานอาจจะมีขนาดใหญ่ เช่นงานๆนั้นมีจำนวนคำสั่ง(Instruction)หลายๆคำสั่งประกอบกันขึ้นมาเป็นโมดูลหรือโพรซิเยอร์(Procedure) ดังนั้นเม็ดเนื้องานคือคำสั่งหนึ่งคำสั่ง หรือหลายๆคำสั่งหรือชุดของคำสั่งที่เรียงกันและถูกรวบรวมคำสั่งเหล่านั้นเกิดเป็นโมดูล โดยชุดของคำสั่งเหล่านั้นจะถูกประมวลผลอย่างเป็นลำดับที่ตัวประมวลผลตัวเดียวกัน จะสังเกตได้ว่าหาก ขนาดของเม็ดเนื้องานมีขนาดใหญ่(เม็ดเนื้องานแบบหยาบ)มาก การทำงานแบบขนานจะถูกลดลง เพราะว่างานที่ถูกให้ทำพร้อมๆกันจะถูกรวบรวมไว้เข้าด้วยกันและถูกให้ประมวลผลอย่างเป็นลำดับที่ตัวประมวลผลตัวเดียวกัน แต่ถ้าหากขนาดของเม็ดเนื้องานมีขนาดที่เล็ก(เม็ดเนื้องานแบบละเอียด) จะทำให้เกิดโอเวอร์เฮดที่อยู่ในรูปแบบของ context-switch และเวลาของลำดับการทำงาน และความล่าช้าอันเนื่องมาจากการติดต่อสื่อสาร สิ่งเหล่านี้ก่อให้เกิดโอเวอร์เฮดที่ใช้ไปในการประมวลผล ปัญหาของขนาดเม็ดเนื้องานจะมีความสัมพันธ์กับปัญหาแบบ มิน-แมก เพราะต้องการทำให้เกิดความสมดุลระหว่างความต้องการให้ทำงานแบบขนาน(เม็ดเนื้องานแบบละเอียด) ในขณะเดียวกันต้องการลดโอเวอร์เฮดอันเนื่องมาจากการติดต่อสื่อสาร(เม็ดเนื้องานแบบหยาบ)

ดังที่เราได้เคยกล่าวมาแล้วว่าเม็ดเนื้องานหนึ่งๆอาจจะประกอบไปด้วยงานเพียงหนึ่งงานหรือมากกว่าหนึ่งงานที่ถูกรวบรวมไว้เข้าด้วยกัน และให้เม็ดเนื้องานนั้นถูกทำการประมวลผลอยู่บนตัวประมวลผลหนึ่งตัว ปัญหาในการเลือกจำนวนของงานแล้วทำการรวบรวมเข้าด้วยกัน ให้เป็นเม็ดเนื้องานเราเรียกปัญหาดังกล่าวว่า ปัญหาการรวบรวมเม็ดเนื้องาน(grain packing problem) ในการรวบรวมเม็ดเนื้องานเพื่อทำการลดโอเวอร์เฮดของการติดต่อสื่อสาร ดังนั้นทำให้โปรแกรมแบบขนานสามารถประมวลผลได้เร็วมากขึ้นในขณะเดียวทำให้จำนวนของตัวประมวลผลลดน้อยลง จากปัญหาดังกล่าวได้นำไปสู่ความสัมพันธ์ในการจัดลำดับงาน เพื่อให้ได้การจัดลำดับงานที่เหมาะสมโดยใช้เวลาในการประมวลผลน้อยที่สุด

วิธีการบางวิธีการต้องการให้เกิดความจุที่สมดุล(load balancing) โดยการพยายามที่จะใช้การทำงานแบบขนานทั้งหมดเท่าที่เป็นไปได้ โดยมิได้มีการพิจารณาถึงค่าใช้จ่ายอันเนื่องมาจากการติดต่อสื่อสารที่สูงมาก บางครั้งสิ่งที่เกิดขึ้นได้นำไปสู่การทำงานของโปรแกรมบนตัวประมวลผลแบบขนานกลับให้สมรรถนะที่ต่ำไปกว่าการทำงานของโปรแกรมที่อยู่บนตัวประมวลผลเพียงตัวเดียว ซึ่งสาเหตุที่ทำให้เกิดการลดสมรรถนะ อันเนื่องมาจากขนาดของงานแต่ละงานที่มีลักษณะเป็นแบบขนาน เล็กไปกว่าค่าใช้จ่ายในการติดต่อสื่อสาร ดังนั้นในการทำความจุที่สมดุลจะพยายามทำให้ตัวประมวลผลทั้งหมดได้ทำงานอยู่ตลอดเวลา แต่ในทางตรงกันข้ามไม่ได้พยายามที่จะลดโอเวอร์เฮดอันเนื่องมาจากการประมวลผลให้ลดลง

ดังนั้นจะต้องหาวิธีการเข้ามาทดแทนในสิ่งที่ไม่ต้องการให้เกิดขึ้น อันเนื่องมาจากการทำความจุที่สมดุล โดยจะต้องทำการปรับปรุงอัตราส่วนระหว่างขนาดของงานและเวลาที่ใช้ในการติดต่อสื่อสาร วิธีการดังกล่าวเราสามารถทำได้โดยการเพิ่มขนาดของงานหรือทำการรวบรวมงานที่มีขนาดเล็กๆเข้าด้วยกัน จะได้งานที่มีขนาดใหญ่ขึ้น การทำดังกล่าวทำให้จำนวนของงานที่เกิดขึ้นพร้อมๆกันลดลง เพราะงานทั้งหมดได้ถูกนำมารวมกันให้ทำการประมวลผลในลักษณะเรียงลำดับแทน รูปที่ 3.9 เป็นการแสดงระดับความเป็นแบบขนานของโปรแกรมที่มีขนาดของเม็ดเนื้องานต่างๆกัน[3]



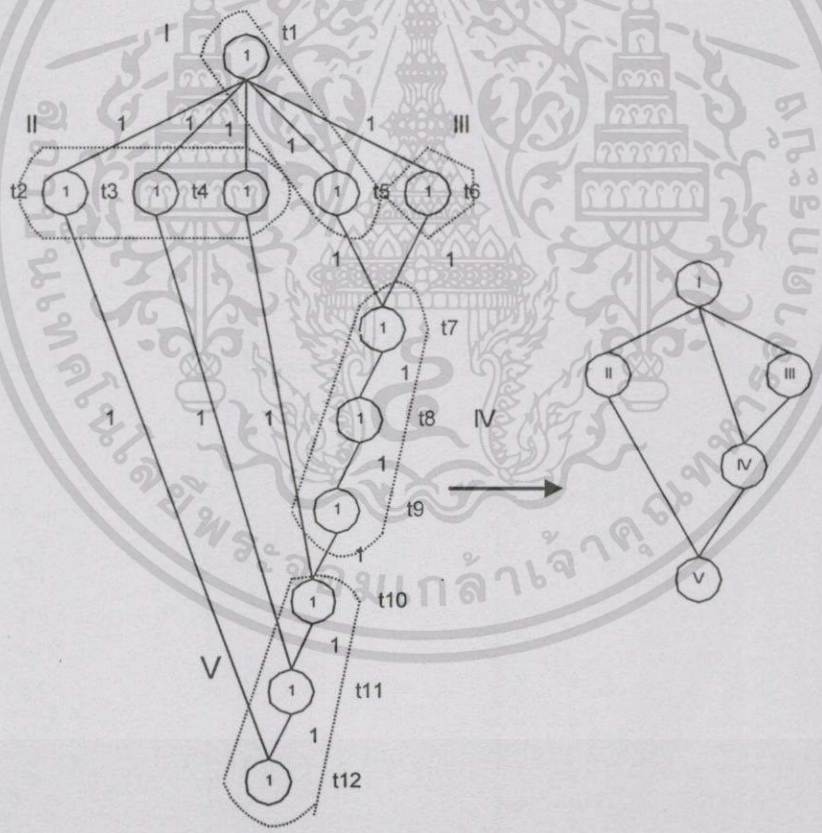
รูปที่ 3.9 แสดงระดับความเป็นแบบขนานของโปรแกรมที่มีขนาดของเม็ดเนื้องานต่างๆกัน

วิธีการหรืออัลกอริทึมที่จะนำเสนอในส่วนนี้จะประกอบไปด้วยสองวิธีการด้วยกัน โดยวิธีการแรกเป็นวิธีการของ Kruatrachue and Lewis[21] โดยวิธีการของ Kruatrachue and Lewis จะใช้การจัดลำดับงานเป็นวิธีการในการรวบรวมเม็ดเนื้องาน วิธีการที่สองเป็นวิธีการของ McCreary and Gill[22] เป็นการสร้างเม็ดเนื้องานจากราฟงาน โดยรวมกลุ่มของโหนดต่างๆลงสู่กลุ่มเดียวกัน

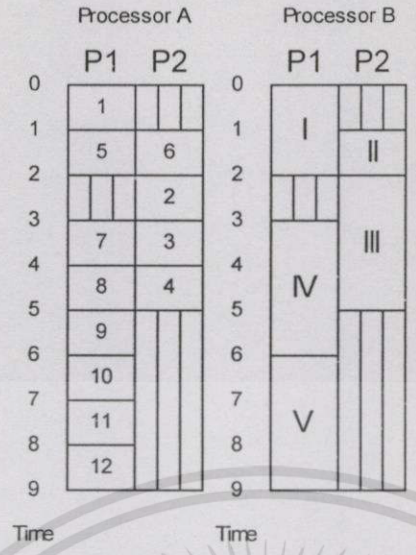
การรวบรวมเม็ดเนื้องานเป็นวิธีการในการเลือกหาขนาดเม็ดเนื้องานที่ดีที่สุดอย่างอัตโนมัติ เพื่อให้สามารถจัดลำดับงานบนระบบเป้าหมายในการประมวลผลแบบขนานได้ แทนที่เราจะทำการกำหนดขนาดของเม็ดเนื้องานที่ดีที่สุดและทำการจัดลำดับงานสำหรับเม็ดเนื้องานเหล่านั้นเสียเอง ในการรวบรวมเม็ดเนื้องานจะเริ่มจากขนาดของเม็ดเนื้องานที่เล็กที่สุด และทำการจัดลำดับงานกับขนาดของเม็ดเนื้องานขนาดเล็กหรือเป็นเม็ดเนื้องานแบบละเอียด จากนั้นทำการเพิ่ม

ขนาดของเม็ดเนื้องานให้ใหญ่ขึ้น เราจะได้รับเม็ดเนื้องานที่ขนาดต่างๆสิ้นสุดลงก็ต่อเมื่อ หลังจากที่เราได้ทำการจัดลำดับงานให้กับเม็ดเนื้องานต่างๆที่เกิดขึ้นก่อนหน้านั้น โดยการนำเอาเม็ดเนื้องานที่ได้มาจัดให้สามารถทำงานแบบขนานได้ โดยจะทำการยอมรับเม็ดเนื้องานดังกล่าว หากผลลัพธ์ที่ได้ไม่ทำให้สมรรถนะของระบบลดลง และผลลัพธ์ที่ได้จากการทำงานแบบขนานที่เกิดขึ้นจะถูกละทิ้ง หากการทำงานดังกล่าวทำให้สมรรถนะของระบบลดลง

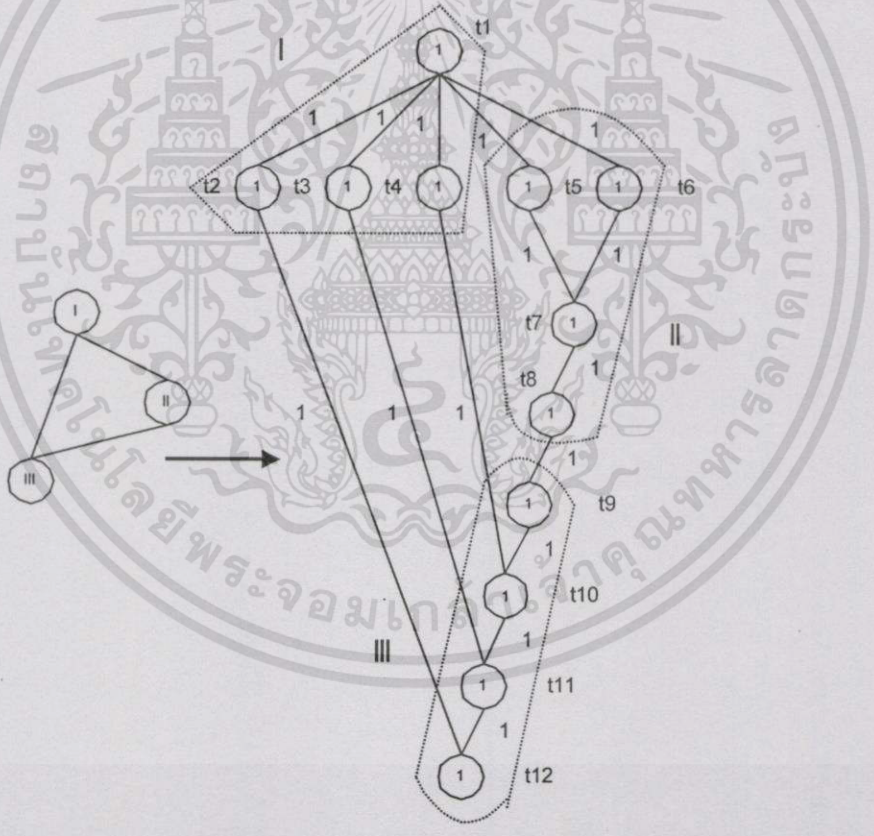
สิ่งสำคัญในการรวบรวมเม็ดเนื้องานโดยผ่านวิธีการจัดลำดับงาน หรือการใช้วิธีการจัดลำดับงานเข้าช่วยดังแสดงอยู่ในรูปที่ 3.10(a) หากเราจะทำการจัดลำดับงานที่อยู่บนกราฟงาน โดยมิได้ทำการรวบรวมเม็ดเนื้องาน ให้ดูจากรูปที่ 3.10(b)ของแกรทชาร์ทA ผลลัพธ์ที่ได้จะใช้เวลาไปทั้งหมด 9 หน่วยเวลา แต่ถ้าหากเราทำการรวบรวมงานเหล่านั้นเข้าด้วยกัน แล้วทำการจัดลำดับงานให้กับงานเหล่านั้นบนตัวประมวลผล P_1 เราจะได้รับเม็ดเนื้องานที่แยกออกจากกัน โดยการใช้การติดต่อสื่อสารเป็นตัวแยก ผลลัพธ์ที่ได้จากการรวบรวมงานเหล่านั้นจะได้กราฟงานที่มีขนาดของเม็ดเนื้องานที่ใหญ่ขึ้น และถูกแสดงด้วยแกรทชาร์ทB สังเกตได้ว่าผลลัพธ์ที่ได้ยังคงเหมือนเดิม



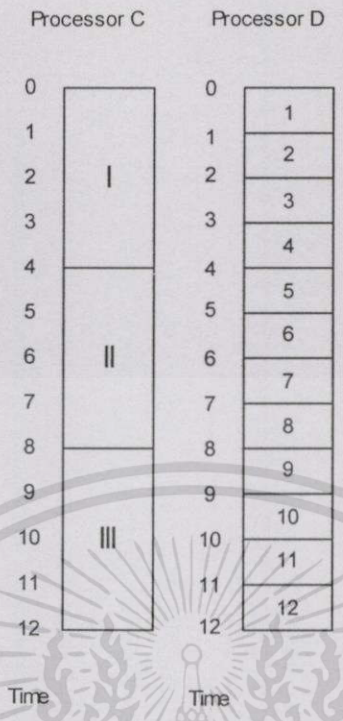
รูปที่ 3.10(a) การจัดลำดับงานโดยผ่านการรวบรวมเม็ดเนื้องาน



รูปที่ 3.10(b) แสดงแกรฟชาร์ทที่ได้จากการรวบรวมเม็ดเนื้องาน



รูปที่ 3.11(a) การรวบรวมเม็ดเนื้องานโดยการจัดลำดับงาน



รูปที่ 3.11(b) แสดงแกรฟชาร์ทที่ได้จากการจัดลำดับงาน

แต่ถ้าหากเราใช้วิธีการรวบรวมเม็ดเนื้องานขนาดเล็กต่างๆที่อยู่บนกราฟงาน ลงบนกลุ่มงานต่างๆ ดังแสดงอยู่ในรูปที่ 3.11(a) ก่อนที่เราจะทำการจัดลำดับงานให้กับงานเหล่านั้น ผลลัพธ์ที่ได้คือกราฟงานที่มีขนาดของเม็ดเนื้องานที่ใหญ่ขึ้น หลังจากนั้นทำการจัดลำดับงานให้กับกลุ่มงานต่างๆเหล่านั้นลงบนแกรฟชาร์ท C และ D ดังแสดงอยู่ในรูปที่ 3.11(b) จากผลลัพธ์ที่ได้เป็นการจำกัดความสามารถในการทำงานแบบขนาน และจากการจัดลำดับงานจะได้ลำดับงานที่ยาวมาก จากผลลัพธ์ที่ได้จากสองวิธีการทำให้มีความจำเป็นที่จะต้องหาวิธีการในการจัดลำดับงานที่ดี เพื่อจะได้รับเม็ดเนื้องานที่เหมาะสม

3.3.1 การจัดกลุ่มงานโดยอาศัยการจัดลำดับงาน วิธีการนี้เป็นฮิวลิสติกอีกวิธีการหนึ่งที่ใช้ในการจัดกลุ่มงาน วิธีการนี้ถูกสร้างโดย Kruatrachue และ Lewis ขั้นตอนการทำงานหลักๆของวิธีการนี้มีดังนี้

อัลกอริทึม Kruatrachue และ Lewis

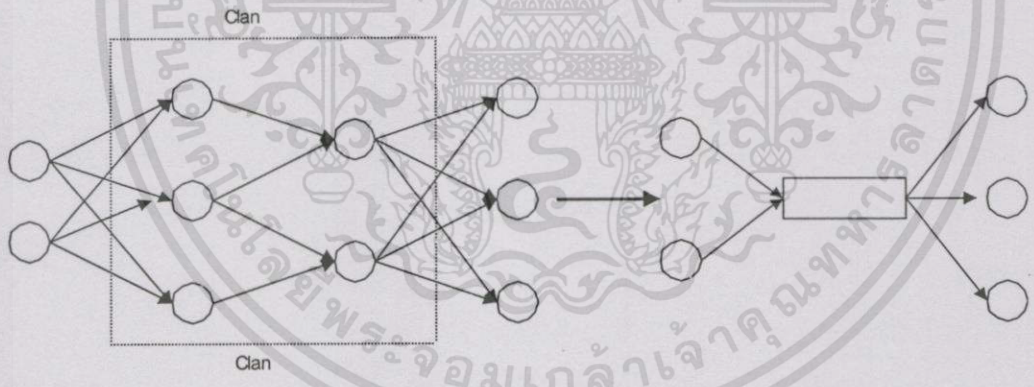
1. ทำการสร้างกราฟงานที่มีลักษณะเป็นแบบเม็ดเนื้องานแบบละเอียด กราฟงานดังกล่าวถูกสร้างมาจากผู้ใช้(user) ดังนั้นความเป็นลักษณะของงานที่เป็นแบบขนานจะถูกแสดงออกมาบนกราฟงานนี้ด้วย โดยกราฟงานที่ได้จะเป็นอิสระจากขนาดของเม็ดเนื้องาน ดังนั้นกราฟงานที่ได้จะเป็นกราฟงานที่มีลักษณะเม็ดเนื้องานแบบละเอียด

2. ให้ทำการคำนวณหาขนาดของงาน ขนาดของงานแต่ละงานจะถูกประมาณจากความยาวของเวลาที่ใช้ในการกระทำของคำสั่งทั้งหมดในงานๆนั้น ขนาดจะถูกวัดในหน่วยของรอบตัวประมวลผล(CPU cycles) ดังนั้นหากในระบบประกอบไปด้วยซีพียูที่มีความเร็วในการประมวลผลที่แตกต่างกัน ทำให้ได้ขนาดของงานแต่ละงานแตกต่างกันออกไป สำหรับแต่ละซีพียูนั้นๆ
3. ให้ทำการคำนวณขนาดของอาร์ค ค่าความล่าช้าในการติดต่อสื่อสารถูกคำนวณได้จากการประมาณเวลาที่ใช้ในการขนถ่ายข้อมูลระหว่างตัวประมวลผล ถ้าแต่ละอาร์คมีอัตราการขนถ่ายข้อมูลที่แตกต่างกัน และระยะทางระหว่างตัวประมวลผลสองตัวในระบบไม่เหมือนกันแล้ว จะทำให้ค่าความล่าช้าในการติดต่อสื่อสารถูกคำนวณได้โดยการเพิ่มสิ่งแตกต่างเหล่านี้เข้าไป
4. การจัดลำดับงานของขนาดเม็ดเนื้องานแบบละเอียด กราฟงานจะถูกจัดลำดับงานอยู่บนระบบที่มีตัวประมวลผลแบบขนาน โดยการใช้ตัวจัดลำดับเม็ดเนื้องานแบบละเอียด เพื่อที่ตัวจัดลำดับเม็ดเนื้องานแบบละเอียดสามารถจะใช้ประโยชน์จากความเป็นแบบขนาน ในขณะเดียวกันช่วยลดการเกิดความล่าช้าในการติดต่อสื่อสาร ขนาดของเม็ดเนื้องานและเวลาที่ใช้ในการประมวลผลของโปรแกรมขึ้นอยู่กับการเลือกตัวจัดลำดับงาน การเลือกตัวกำหนดลำดับงานเป็นสิ่งสำคัญ การเลือกใช้ตัวกำหนดลำดับงานในการทำการรวบรวมเม็ดเนื้องานจะต้องสามารถทำการแก้ไขปัญหา มิน-แมก
5. ในขั้นตอนการทำรวบรวมเม็ดเนื้องาน เราจะใช้แกรฑชาร์ทเข้ามาช่วยในการวิเคราะห์ และเม็ดเนื้องานแบบละเอียดจะถูกรวบรวมเข้าด้วยกัน ทำให้ได้เม็ดเนื้องานที่ใหญ่ขึ้น ในขณะเดียวกันจะเป็นการช่วยลดการเกิดโอเวอร์เฮด โดยปกติแล้วในเม็ดเนื้องานขนาดใหญ่จะมีโอเวอร์เฮดน้อยมาก

จากวิธีการที่กล่าวมาจะพบว่าขั้นตอนที่สำคัญในการทำงานของอัลกอริทึมดังกล่าว คือจะต้องทำการหาตัวกำหนดลำดับงานที่ดี โดยตัวกำหนดลำดับงานนั้นจะต้องสามารถแก้ไขปัญหา มิน-แมก ได้ ตัวอย่างการทำงานของการเลือกใช้ตัวกำหนดลำดับงานสามารถดูได้จาก [1][3][13]

3.3.2 การรวบรวมเม็ดเนื้องานโดยการแบ่งกราฟงานออกเป็นกลุ่ม วิธีการนี้เป็นการจัดลำดับงานที่อาศัยการแบ่งโหนดต่างๆบนกราฟงานออกเป็นเคล็น(Clan) คำว่าเคล็นในที่นี้หมายถึงการรวมกันเป็นปึกแผ่น หรือเป็นกลุ่มก้อนนั้นเอง แต่ในที่นี้จะใช้ว่าคำว่าเคล็นแทนคำว่ากลุ่มก้อนจะได้ตรงตาม[22]ได้กล่าวไว้

ความคิดที่เป็นจุดสำคัญของวิธีการนี้อยู่ที่เทคนิคในการรวบรวมโหนดต่างๆบนกราฟงานจะได้เป็นสับเซตโหนดของกราฟงาน โดย McCreary เรียกสับเซตนี้ว่าเคล็น นอกจากนี้ McCreary ยังได้กล่าวว่าเคล็นจะเหมือนกับเม็ดเนื้องาน เคล็นจะมีคุณสมบัติต่างๆทำให้เป็นสิ่งดึงดูดที่น่าสนใจเช่นเดียวกับเม็ดเนื้องานสำหรับกระบวนการแบบขนาน คุณสมบัติของเคล็นข้อแรก ซอสโหนดทั้งหมดของเคล็นหนึ่งเคล็นจะมีชุดของบรรพบุรุษที่เหมือนกัน ดังนั้นการติดต่อสื่อสารสามารถถูกลดลงได้โดยการรวมเอาโหนดต่างๆที่อยู่ในเคล็นเหล่านั้นรวมลงสู่ตัวประมวลผลตัวเดียวได้ เช่นเดียวกันซิ้งค์โหนดทั้งหมดของเคล็นหนึ่งเคล็นจะมีโหนดลูกหลานเหมือนกันหมด ดังนั้นการติดต่อสื่อสารสามารถถูกรวบรวมได้ ดังแสดงอยู่ในรูปที่ 3.12 นอกจากนี้ต้นไม้แบบกระจาย(Parse Tree) ได้เสนอระบบในการวัดค่าใช้จ่ายที่เกิดจากการรวบรวมโหนดต่างๆเข้าเป็นเคล็น โดยกำหนดให้โหนดรากของต้นไม้เป็นการแสดงถึงการกระทำที่จะเกิดขึ้นทั้งหมด โดยกระทำในลักษณะเรียงตามลำดับงานที่อยู่บนตัวประมวลผลเดียว ในการดำเนินการจะเริ่มจากบนลงล่างของต้นไม้ การกระทำดังกล่าวเป็นการเพิ่มโอกาสที่จะได้ใช้ตัวประมวลผลเพิ่มมากขึ้น ในบางครั้งอาจจะต้องมีการยอมรับโอกาสอันนั้นหรือทำการปฏิเสธที่จะใช้โอกาสอันนั้น อันเนื่องมาจากต้องการพิจารณาในเรื่องของค่าใช้จ่ายและการติดต่อสื่อสารเป็นหลัก การแสดงอัลกอริทึมของ McCreary และ Gill โดยอินพุทของอัลกอริทึมคือ DAG ส่วนเอาพุทที่ได้จากอัลกอริทึมคือเม็ดเนื้องานที่สามารถทำงานอยู่บนตัวประมวลผลแบบขนานได้



รูปที่ 3.12 การรวบรวมโหนดต่างๆเข้าเป็นเคล็น

<u>อัลกอริทึม McCreary and Gill</u>

1. ทำการกระจาย DAG ให้อยู่ในรูปของต้นไม้
2. กำหนดค่าใช้จ่ายในการกระทำหรือประมวลผลให้กับโหนดบัพใบ(leaves nodes)
3. คำนวนหาค่าใช้จ่ายของโหนดภายในทั้งหมด(internal nodes)
4. พิจารณาการบรรจุของงานหรือโหนดของแต่ละเม็ดเนื้องาน ด้วยข้อกำหนดดังนี้

- เมื่อใดที่ค่าใช้จ่ายที่โหนด เท่ากับ ผลรวมของค่าใช้จ่ายของโหนดลูกหลาน จะกำหนดให้โหนดลูกหลานเหล่านั้นจะถูกวางอยู่ที่ตัวประมวลผลตัวเดียวกัน
- เมื่อใดที่ค่าใช้จ่ายที่โหนด น้อยกว่า ผลรวมของค่าใช้จ่ายของโหนดลูกหลาน จะกำหนดให้โหนดลูกหลานเหล่านั้นถูกรวบรวมลงสู่เม็ดเนื้องาน

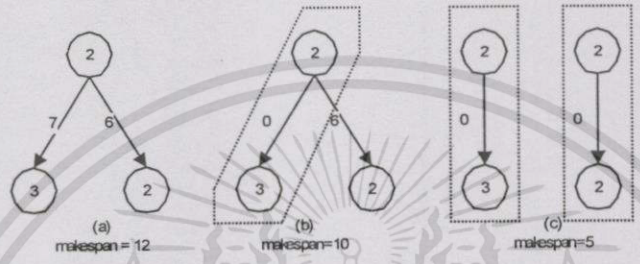
วิธีการนี้เป็นการนำเสนอวิธีการในการรวบรวมเม็ดเนื้องานที่ไม่สลับซับซ้อนมากนัก โดยเป้าหมายหลักวิธีการนี้ เพื่อการทำการเลือกส่วนประกอบต่างๆที่อยู่บนโปรแกรม จากนั้นทำการรวบรวมสิ่งเหล่านั้นให้อยู่ในรูปของเม็ดเนื้องานและกำหนดให้เม็ดเนื้องานอยู่บนตัวประมวลผล เป็นขั้นตอนสุดท้าย วัตถุประสงค์ของอัลกอริทึมจะทำการจัดลำดับงานของโปรแกรมเพื่อให้การประมวลผลได้เร็วขึ้น มากกว่าที่จะใช้จำนวนของทรัพยากรในการประมวลผลให้มากที่สุด

3.4 การประยุกต์ใช้อัลกอริทึมพื้นฐาน ในการจัดกลุ่มงานและลำดับงาน

ในส่วนนี้เป็นการนำเสนอฮิวลิสติกอัลกอริทึมอีกรูปแบบหนึ่ง โดยมีการใช้อัลกอริทึมพื้นฐานเดิมเข้ามาช่วยในการจัดกลุ่มงานและจัดลำดับการทำงาน อัลกอริทึมพื้นฐานได้แก่ กรีดี้อัลกอริทึม แบล็คแทรกกิ้งอัลกอริทึม ดีไวแอนคองเคอร์(Divide-and-Conquer)อัลกอริทึม เป็นต้น ส่วนนี้จะนำเสนอสองอัลกอริทึมด้วยกันคือ อัลกอริทึมของ Michael, Jing-Chiou, และ David[23] ส่วนอัลกอริทึมที่สองเป็นการนำเสนอผลงานวิจัยของกระผมเอง[24] โดยอัลกอริทึมทั้งสองใช้กรีดี้อัลกอริทึมในการจัดกลุ่มงานและลำดับการทำงาน

อัลกอริทึมในการจัดกลุ่มงานโดยส่วนใหญ่แล้ว จะแบ่งวิธีการจัดกลุ่มงานออกเป็นสองประเภทด้วยกัน คือแบบกลุ่มงานที่มีงานซ้ำซ้อนเกิดขึ้น(Task Duplication) และแบบกลุ่มงานที่ไม่มีงานซ้ำซ้อนเกิดขึ้น(without Task Duplication) กลุ่มงานที่มีงานซ้ำซ้อนเกิดขึ้นหมายถึงงานบนกราฟงานสามารถเกิดขึ้นได้บนกลุ่มงานต่างๆได้มากกว่าหนึ่งกลุ่มงาน โดยที่การจัดลำดับงานของกลุ่มงานแต่ละกลุ่มเหล่านี้เป็นอิสระต่อกัน ส่วนกลุ่มงานแบบไม่มีงานซ้ำซ้อนหมายถึงงานแต่ละงานที่อยู่บนกราฟงาน สามารถเกิดขึ้นได้เพียงกลุ่มงานเดียวและกลุ่มงานทั้งหมดจะถูกกำหนดจัดลำดับการทำงาน อัลกอริทึมที่ใช้ในการจัดกลุ่มงานแบบที่ไม่มีงานซ้ำซ้อนเกิดขึ้นจะมีความสลับซับซ้อนมากกว่าอัลกอริทึมการจัดกลุ่มงานแบบที่มีงานซ้ำซ้อนเกิดขึ้น ตัวอย่างของอัลกอริทึมที่ใช้ในการจัดกลุ่มงานแบบที่มีงานซ้ำซ้อนเกิดขึ้นได้แก่[23][24][25] ข้อดีอย่างหนึ่งของอัลกอริทึมการจัดกลุ่มงานแบบมีงานซ้ำซ้อนเกิดขึ้น จะให้ค่าของ makespan สำหรับการกำหนดลำดับงานน้อยกว่า

อัลกอริทึมแบบกลุ่มงานที่ไม่มีงานซ้ำซ้อนเกิดขึ้น รูปที่ 3.13 เป็นการแสดงตัวอย่างของค่า makespan ที่ได้จากอัลกอริทึมทั้งสองแบบ จากรูป 3.13 (b) เป็นการแสดงลักษณะกลุ่มงานแบบไม่มีงานซ้ำซ้อนเกิดขึ้นจะให้ค่าของ makespan มากกว่ารูปที่ 3.13 (c) ที่มีลักษณะกลุ่มงานแบบมีงานซ้ำซ้อนเกิดขึ้น เพราะเหตุว่าค่าใช้จ่ายในการติดต่อสื่อสารระหว่างโหนดที่อยู่บนตัวประมวลผลตัวเดียวกันหรือกลุ่มงานเดียวกันจะเท่ากับศูนย์ ปัญหาการจัดกลุ่มงานทั้งแบบมีงานซ้ำซ้อนและแบบไม่มีงานซ้ำซ้อนเกิดขึ้นจัดได้ว่าเป็นปัญหาแบบ NP-complete



รูปที่ 3.13 แสดงประเภทของกลุ่มงาน (a) แสดง DAG (b) กลุ่มงานแบบไม่มีงานซ้ำซ้อน (c) กลุ่มงานแบบมีงานซ้ำซ้อน

3.4.1 อัลกอริทึมการจัดกลุ่มงานและลำดับงานสำหรับสถาปัตยกรรมแบบขนานที่มีลักษณะกระจายหน่วยความจำ(Task Clustering and Scheduling for Distributed Memory Parallel Architectures/TCS) กรีดี้อัลกอริทึมได้ถูกนำมาใช้ในการจัดกลุ่มงาน โดยลักษณะกลุ่มงานเป็นแบบที่มีงานซ้ำซ้อน หรืองานหนึ่งงานบนกราฟงานที่มีลักษณะเม็ดเนื้องานขนาดเล็กใดๆ(Arbitrary Granularity) ได้ถูกปรากฏอยู่บนกลุ่มงานมากกว่าหนึ่งกลุ่มงานหรือมากกว่าหนึ่งตัวประมวลผล ซึ่งจะทำให้ค่าของ makespan ของลำดับงานของกลุ่มงานที่อยู่บนกราฟงาน ที่ได้จากทีซีเอส(TCS)มีค่าอย่างมากที่สุดเท่ากับสองเท่าที่ได้จากการจัดกลุ่มงานที่เหมาะสม(Optimal Cluster) นอกจากนั้นยังเป็นการพิสูจน์ให้เห็นว่าทีซีเอสได้ทำการปรับปรุงคุณภาพลำดับงาน โดยการทำให้เม็ดเนื้องานที่มีขนาดเล็กบนกราฟงานมีขนาดใหญ่ขึ้น หรืออาจจะกล่าวได้ว่าเวลาที่ใช้ในการกระทำหรือการประมวลผลที่ตัวประมวลผลมากกว่าเวลาที่ใช้ในการติดต่อสื่อสารระหว่างตัวประมวลผล สำหรับตัวอย่างถ้ากราฟงานใดๆที่มีค่าของ Granularity อย่างน้อยที่สุดเท่ากับ ½ แล้วค่าของ makespan ของลำดับการทำงานของกลุ่มงานบนกราฟงานที่ได้จากการใช้ทีซีเอสอัลกอริทึมจะมีค่าอย่างมากที่สุดเท่ากับ(5/3)*(Optimal Cluster) สำหรับกราฟงานใดๆที่ประกอบไปด้วยงานหรือโหนดจำนวน V โหนด และมีอาร์คจำนวนเท่ากับ E แล้ว ทีซีเอสอัลกอริทึมจะมีค่าความสลับซับซ้อนของเวลาเท่ากับO(|V|(|V| lg |V| + |E|))

อัลกอริทึม ทีซีเอส

ความคิดพื้นฐานของอัลกอริทึมนี้จะคล้ายกับ PY อัลกอริทึม[25] กำหนดให้แต่ละโหนด v เป็นสมาชิกของ V ($v \in V$) ขั้นตอนแรกจะเริ่มทำการคำนวณหาค่าเวลาเริ่มต้นที่เร็วที่สุดสำหรับโหนด v (e(v) = เวลาเริ่มต้นของโหนด v) ขั้นตอนนี้ทำได้โดยการหากลุ่มงานสำหรับโหนด v (C(v))หรือกลุ่มงานที่สามารถทำการบรรจุโหนด v ได้ โดย C(v) จะต้องให้ค่าเวลาเริ่มต้นที่เร็วที่สุดเท่าที่เป็นไปได้ และโหนดอื่นๆทั้งหมดใน C(v) ถูกประมวลผลอยู่บนตัวประมวลผลตัวเดียวกัน และโหนดอื่นๆใน V - C(v) ถูกประมวลผลอยู่บนตัวประมวลผลตัวอื่นๆ ดังนั้นการสร้างโดยวิธีการดังกล่าวเราสามารถทำได้โดยการใช้กรีดี้อัลกอริทึม ทำให้กลุ่มงานสามารถขยายใหญ่ขึ้นได้หรือจำนวนสมาชิกภายในกลุ่มงานเพิ่มมากขึ้นได้เพียงหนึ่งโหนดต่อหนึ่งครั้งการทำงาน เมื่อได้ชุดของกลุ่มงานมาต่อจากนั้นจะทำการวางชุดของกลุ่มงานเหล่านั้นลงบนตัวประมวลผลต่างๆที่มีไว้ โดยอัลกอริทึมจะแสดงให้เห็นว่าการวางชุดของกลุ่มงานต่างๆลงบนตัวประมวลผล แล้วจะให้ค่าของ makespan ของลำดับการทำงานของชุดกลุ่มงานใกล้เคียงกับค่า makespan ของลำดับการทำงานที่เหมาะสม

การคำนวณหาค่าเวลาเริ่มต้น(e)ขึ้นอยู่กับลักษณะลำดับของโหนดใน DAG G สำหรับซอสโหนด(source node)แล้วเวลาเริ่มต้นของมันเท่ากับศูนย์ และโหนดอื่นๆค่าเวลาเริ่มต้นจะถูกคำนวณหลังจากที่โหนดบรรพบุรุษ(ancestors)ของมันได้ถูกกำหนดค่าเริ่มต้นเอาไว้แล้ว ในการหาค่าเวลาเริ่มต้นของโหนด v ใดๆจะต้องทำการหาเวลาเริ่มต้นของโหนดบรรพบุรุษของโหนด v ทั้งหมดเสียก่อน ดังนั้นค่าเวลาเริ่มต้นที่ต่ำที่สุดสำหรับโหนด v ใดๆ หรือ e(v) เราสามารถพิจารณาเพียงแค่โหนดบรรพบุรุษต่างๆที่อยู่ในกลุ่มงานที่มี v บรรจุอยู่เท่านั้น (C(v)) ดังนั้นโหนดบรรพบุรุษเหล่านั้นที่อยู่ใน C(v) อาจเป็นเพียงส่วนหนึ่ง(หรือสับเซต)ของโหนดบรรพบุรุษทั้งหมดของ v ที่มีอยู่ใน DAG G ก็ได้ ถ้ากำหนดให้กลุ่มงาน C' ประกอบไปด้วยโหนด w โดยที่โหนด w นั้นไม่ได้เป็นโหนดบรรพบุรุษของโหนด v การดึงเอาโหนด w ออกจากกลุ่มงาน C' ความเป็นไปได้ที่จะทำให้เวลาเริ่มต้นของโหนด v เร็วขึ้น แต่เวลาเริ่มต้นของโหนด w จะไม่ช้าไปกว่าเวลาเริ่มต้นของโหนด v ในกลุ่มงาน C'

กำหนดให้กลุ่มงาน C ประกอบไปด้วยโหนด v และสับเซต(Subset)ของโหนดบรรพบุรุษ $\{u_1, u_2, \ldots, u_k\}$ ของโหนด v เราต้องการหาค่าเวลาเริ่มต้นที่ต่ำที่สุดของโหนด v ในกลุ่มงาน C ดังนั้นถูกแสดงด้วย $e_C(v)$ และโหนดบรรพบุรุษอื่นๆทั้งหมดที่อยู่ในกลุ่มงาน C จะต้องถูกประมวลผลอยู่บนตัวประมวลผลตัวเดียวกัน ในการหาค่าเวลาเริ่มต้นที่ต่ำที่สุดของ v ในกลุ่มงาน C จะประกอบไปด้วย 2 ขั้นตอนด้วยกัน ขั้นตอนที่ 1 จะทำการหาค่าของ makespan ของกลุ่มงานที่อยู่ภายใน C ขณะนั้น และขั้นตอนที่ 2 จะทำการหาค่าอาร์ค c ซึ่งเป็นอาร์คที่ใช้ในการเชื่อมต่อจากโหนดภายนอกเข้ามาโหนดภายในกลุ่มงาน C ทั้งสองขั้นตอนดังกล่าวจะใช้กรีดี้อัลกอริทึมเป็นส่วนสำคัญในการหาคำตอบ

ขั้นตอนที่1) คำถามของปัญหาต้องการทราบว่าลำดับของเวลาเริ่มต้นของโหนด v เท่ากับเท่าใด คำตอบก็คือค่าของ makespan ของลำดับการทำงานที่เหมาะสมสำหรับปัญหาการจัดลำดับด้วยตัวประมวลผลตัวเดียว(one-processor scheduling problem) ด้วยค่าของเวลาเริ่มต้นสำหรับแต่ละโหนดบรรพบุรุษ $\{u_1, u_2, \ldots, u_k\}$ โดยที่ค่าของเวลาเริ่มต้นสำหรับแต่ละโหนดบรรพบุรุษเท่ากับ $e(u_i)$ และ $\mu(u_i)$ เป็นค่าของเวลาที่ใช้ในการประมวลผลของโหนด u_i ตามลำดับ ปัญหาดังกล่าวสามารถทำการแก้ไขปัญหาให้เหมาะสมได้โดยการใช้กรีดี้อัลกอริทึม ซึ่งจะทำการประมวลผลด้วยค่าของเวลาเริ่มต้นของโหนดบรรพบุรุษแต่ละโหนดจากน้อยไปหามาก(nondecreasing order) ดังนั้นค่าของเวลาที่เริ่มต้นที่ต่ำที่สุดของโหนด v ในกลุ่มงาน C เท่ากับ

$$e_C(v) \geq GREEDY\_SCHEDULE\ (C - \{v\}) \qquad 3.4$$

เมื่อ GREEDY_SCHEDULE(C – {v}) จะให้ค่าของ makespan ของลำดับการทำงานด้วยตัวประมวลผลตัวเดียวสำหรับชุดของงานที่ระบุอยู่ในอาร์กิวเมนท์(argument)

ขั้นตอนที่2) ต่อมาทำการพิจารณาชุดของอาร์ค(set of arcs) ที่ใช้ในการเชื่อมต่อโหนดที่อยู่ภายนอกกลุ่มงาน C อย่างเช่นกลุ่มงาน C อาจจะมีอาร์คจากโหนดภายนอกเชื่อมต่อเข้ามาหาโหนดที่อยู่ในกลุ่มงาน C ดังนั้นเราจะทำการกำหนดให้ c(u,w) เป็นค่าของอาร์คจากโหนดภายนอกเชื่อมต่อเข้ามาหาโหนดที่อยู่ภายใน C นี้ โดยที่ $c(u,w) = e(u) + \mu(u) + \lambda(u,w)$ จาก c(u,w) บอกให้ทราบว่าโหนด v นั้นจะไม่สามารถถูกจัดลำดับการทำงานก่อนค่าของ c(u,w) ได้เพราะว่าค่าของ c(u,w)เป็นการบ่งบอกถึงเส้นทางจากโหนด u ซึ่งอยู่ภายนอกกลุ่มงาน C เข้ามาหาโหนด v ที่อยู่ภายในกลุ่ม C ได้ก็ต่อเมื่อจะต้องผ่านโหนด w ก่อน โดยที่โหนด w เป็นโหนดบรรพบุรุษใดๆของโหนด v ที่อยู่ภายในกลุ่ม C และโหนด w ให้โหนด u ใช้เชื่อมต่อเข้ามาหาโหนด v ดังนั้นค่าของเวลาเริ่มต้นที่ต่ำที่สุดของโหนด v ในกลุ่มงาน C เท่ากับ

$$e_C(v) \geq MAX\_C\_VALUE\ (C) \qquad 3.5$$

เมื่อค่าของ MAX_C_VALUE(C) จะให้ค่าของ c(u,w) ที่มากที่สุดในขณะนั้น จาก (3.4) และ (3.5) จะได้ค่าของเวลาเริ่มต้นที่ต่ำที่สุดของโหนด v ที่อยู่ในกลุ่มงาน C

$$e_C(v) \geq \max\{GREEDY\_SCHEDULE(C - \{v\}), MAX\_C\_VALUE(C)\} \qquad 3.6$$

และค่าเวลาเริ่มต้นของโหนด v ใดๆ จะต้องไม่น้อยไปกว่าค่าของเวลาเริ่มต้นที่ต่ำที่สุดของโหนด v ใดๆ ที่อยู่กลุ่มงาน C

$$e(v) \geq \min_{C}\{e_C(v)\} \qquad 3.7$$

จากสมการที่ (3.6) และ (3.7) ทำให้เราสามารถหาค่าของเวลาเริ่มต้นที่ต่ำที่สุดของโหนด v ใดๆ ในกลุ่มงาน C ได้ และจากสมการที่ (3.4) ถึง (3.7) เราได้ใช้กรีดี้อัลกอริทึมทั้งสิ้นช่วยในการหาค่าของเวลาที่ต่ำที่สุดของโหนด v ใดๆ

หลักการทำงานของอัลกอริทึมมีอยู่ว่าในตอนเริ่มแรกกลุ่มงาน C จะเท่ากับเซตว่าง(empty set) จากนั้นจะนำเอาโหนด v เอามาใส่ไว้ในกลุ่มงาน C ดังนั้นโหนด v นี้จึงเป็นโหนดแรกของกลุ่มงาน C นี้ จากนั้นกลุ่มงาน C จึงเริ่มทำการหาโหนดบรรพบุรุษอื่นๆ ที่เป็นของโหนด v ใน DAG G มาใส่เพิ่ม โดยที่โหนดบรรพบุรุษเหล่านั้นของโหนด v ที่จะนำเอามาใส่ไว้ในกลุ่มงาน C ได้ก็ต่อเมื่อ ถ้านำเอาโหนดบรรพบุรุษมาใส่ไว้ในกลุ่มงาน C แล้วจะทำให้ค่าเวลาเริ่มต้นของโหนด v ในกลุ่มงาน C นั้นลดลงได้ แต่ถ้าหากการนำเอาโหนดบรรพบุรุษเหล่านั้นมาใส่ไว้ในกลุ่มงาน C แล้วทำให้เวลาเริ่มต้นของโหนด v ไม่ลดลงเราจะไม่นำเอาโหนดบรรพบุรุษเหล่านั้นเข้ามารวมไว้ในกลุ่มงาน และการขยายขนาดของกลุ่มงาน C จะหยุดลงเมื่อไม่มีโหนดบรรพบุรุษใดแล้วที่ทำให้เวลาเริ่มต้นของโหนด v ในกลุ่มงานนั้นลดลงได้ ลักษณะการขยายขนาดของกลุ่มงาน C ในแต่ละครั้งนั้น จะสามารถทำได้โดยการเพิ่มโหนดเข้ามาในกลุ่มงาน C ได้เพียงโหนดเดียวเท่านั้น

จากสมการที่ (3.6) เราสามารถสรุปได้ว่าในการหาค่าเวลาเริ่มต้นของโหนด v ของกลุ่ม C หรือ $e_C(v)$ จะเกิดขึ้นได้ด้วยสองกรณีด้วยกัน

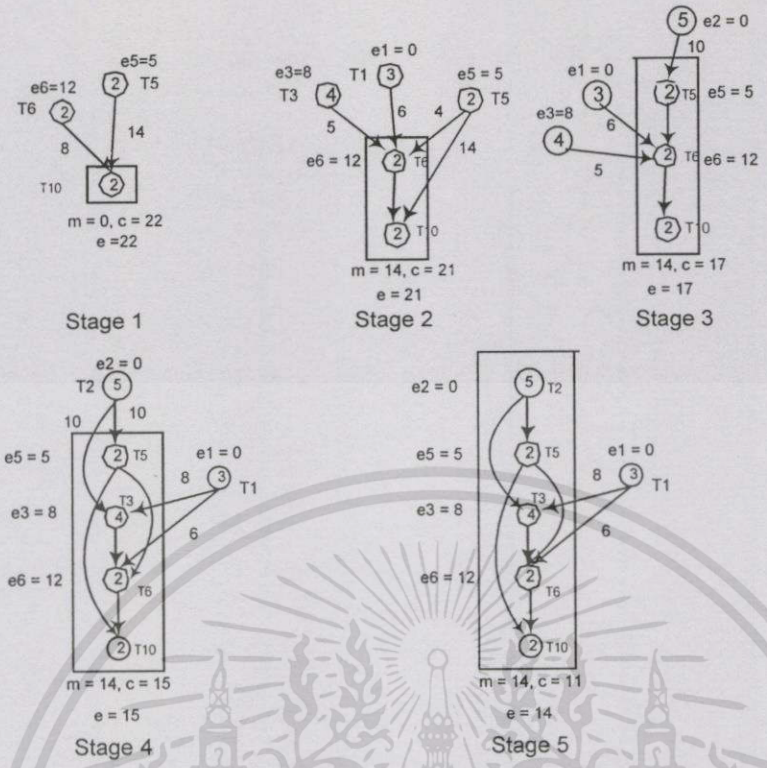
กรณีที่ 1) **MAX_C_VALUE(C) > GREEDY_SCHEDULE(C – {v})** โดยที่ (u,w) เป็นอาร์คที่ใช้ในการเชื่อมต่อจากโหนดภายนอกเข้ามาหาโหนดในกลุ่มงาน C เมื่อ c(u,w) = MAX_C_VALUE(C) ในกรณีทำให้ $e_C(v) \geq c(u,w)$ ผลที่ได้ทำให้ค่าของเวลาเริ่มต้นของโหนด v ในกลุ่มงาน C ลดลง ดังนั้นเราสามารถนำเอาโหนด u ที่อยู่ภายนอกกลุ่มงาน C เข้ามารวมในกลุ่มงาน C ได้ ทำให้กลุ่มงาน C มีขนาดเพิ่มขึ้น

กรณีที่ 2) **GREEDY_SCHEDULE(C – {v}) ≥ MAX_C_VALUE(C)** ในกรณีนี้ทำให้ $e_C(v) \geq$ GREEDY_SCHEDULE(C - {v}) ดังนั้นการเพิ่มโหนด X ใดๆเข้ามาในกลุ่มงาน C แล้วไม่ได้ทำให้เวลาเริ่มต้นของโหนด v ในกลุ่มงาน C ลดลงได้ เพราะว่า GREEDY_SCHEDULE(C $\cup$ {x} – {v}) ≥ GREEDY_SCHEDULE(C – {v}) ผลดังกล่าวทำให้การเพิ่มขนาดของกลุ่มงาน C หยุดลง

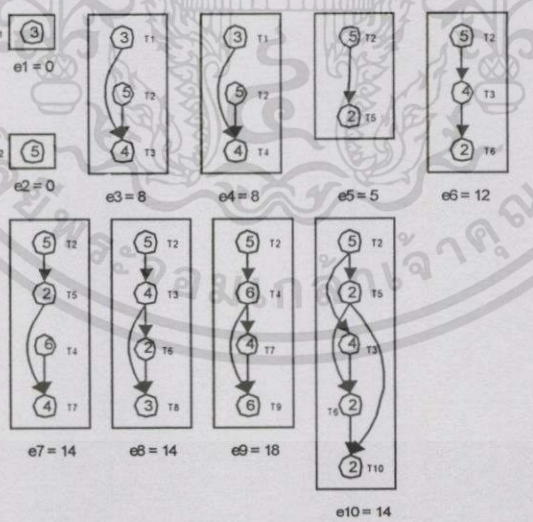
จากสมการที่ (3.6) และ (3.7) ทำให้เราสามารถนำมาสร้างอัลกอริทึมที่สมบูรณ์สำหรับการคำนวณหาค่าเวลาเริ่มต้นที่ต่ำที่สุดของโหนด v (e(v))ใดๆ โดยอัลกอริทึมดังกล่าวชื่อว่า COMPUTE_E_VALUE อัลกอริทึมดังปรากฏอยู่ข้างล่างนี้ รูปที่ 3.14 แสดงตัวอย่างขั้นตอนการ

ทำงานในการสร้างกลุ่มงานสำหรับโหนดT10 และรูปที่ 3.15 เป็นการแสดงผลลัพธ์ที่ได้จากการใช้ COMPUTE_E_VALUE อัลกอริทึม โดยใช้ DAG G รูปที่ 2.7 เป็นโจทย์ของปัญหา

```
Algorithm COMPUTE_E_VALUE( v, G)
Begin
        If v is a source node then return (0);
        C ← {v};
        m ← 0;
        c ← MAX_C_VALUE( C );
        e ← c;
        while m < c do
                let (u,w) be an arc such that u ∉ C, w ∈C, and c = c(u,w);
                C ← C ∪ {u };
                m ← GREEDY_SCHEDULE( C – {v} );
                c ← MAX_C_VALUE( C );
                e ← min{ e, max{m,c} };
        endwhile;
        return(e);
End COMPUTE_E_VALUE
```



รูปที่ 3.14 แสดงขั้นตอนการสร้างกลุ่มงานสำหรับโหนด T10 โดยมีค่าของ e=14 แสดงให้เห็นถึงค่าเวลาเริ่มต้นที่น้อยที่สุดสำหรับโหนด T10



รูปที่ 3.15 แสดงผลลัพธ์ที่ได้จากการใช้อัลกอริทึม COMPUTE_E_VALUE( )

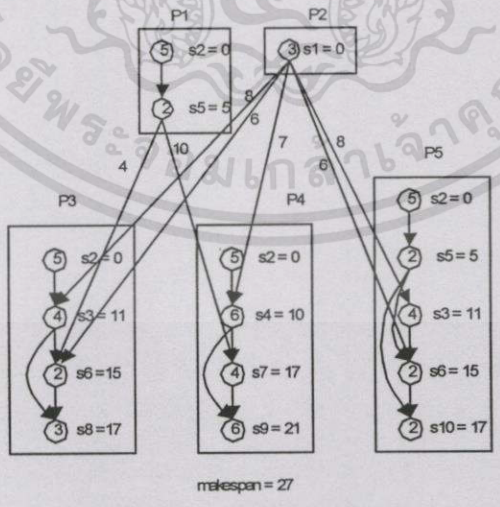
จาก Algorithm COMPUTE_E_VALUE(v, G) นี้นอกเหนือไปจากสามารถหาค่าของเวลาเริ่มต้นที่ต่ำที่สุดของโหนด v ใดๆที่อยู่บนกลุ่มงาน C แล้ว ยังสามารถนำมาดัดแปลงเพื่อนำมาสร้างชุดของกลุ่มงานซึ่งแทนด้วยสัญญลักษณ์ $\Phi(G)$ สำหรับ DAG G ใดๆ โดยการนำเอา C(v) ต่างๆที่ได้จาก Algorithm COMPUTE_E_VALUE(v, G) มาทำการสร้าง $\Phi(G)$ หลักการในการสร้าง $\Phi(G)$ จะต้องทำการเยี่ยม(visit)โหนดต่างๆที่อยู่บน DAG G ในลักษณะกลับทิศทาง(กล่าวคือจะทำการเยี่ยมโหนดในลักษณะจากซิ้งค์โหนดกลับไปหาซอสโหนดหรือตัวชี้กลับทิศ) ในตอนเริ่มต้นเราจะทำการกำหนดให้ $\Phi(G)$ เท่ากับเซตว่าง, $\Phi(G) = \phi$, และซิ้งค์โหนดใน DAG G จะถูกมาร์คเอาไว้ ขั้นตอนถัดไปจะทำตามขั้นตอนดังต่อไปนี้จนกระทั่งโหนดต่างๆใน G ไม่มีมาร์คโหนดเหลืออยู่อีก

(1) ให้ทำการดึงมาร์คโหนด v และทำการเพิ่ม C(v) ใส่เข้าไปใน $\Phi(G)$

(2) จากนั้นทำการอันมาร์คโหนด v ทิ้งและทำการมาร์คโหนด u ทั้งหมดแทน เมื่อโหนด u เหล่านั้นเป็นโหนดบรรพบุรุษของโหนด v เมื่อโหนด u นั้นอยู่บนอาร์คของ (u,w) เมื่อ $u \notin C(v)$ แต่ $w \in C(v)$

ในการจัดลำดับการทำงานของ $\Phi(G)$ ทำได้จากการวางกลุ่มงาน C(v) ต่างๆที่อยู่ใน $\Phi(G)$ ลงบนตัวประมวลผลต่างๆ โดยที่หนึ่งกลุ่มงานต่อหนึ่งตัวประมวลผล และในการประมวลผลของงานต่างๆที่อยู่บนตัวประมวลผลตัวเดียวกันจะเรียงลำดับจากน้อยไปหามากด้วยค่าของเวลาเริ่มต้นของงานๆนั้น

จากการจัดลำดับการทำงานของ $\Phi(G)$ แสดงให้เห็นว่า เราสามารถหาค่าของ makespan ของลำดับการทำงานที่ดีสำหรับ DAG G ใดๆได้ดังนี้



รูปที่ 3.16 แสดงผลการสร้าง $\Phi(G)$ โดยใช้กลุ่มงานต่างๆในรูปที่ 3.15

จากรูปที่ 3.16 แสดงให้เห็นว่ากลุ่มงานต่างๆที่ถูกสร้างจาก Algorithm COMPUTE_E_VALUE(v, G) โดยใช้ DAG G ในรูปที่ 2.7 เป็นปัญหาของโจทย์ เมื่อถูกวางอยู่บนตัวประมวลผลต่างๆกัน ค่าของ makespan ที่ได้สำหรับ $\Phi(G)$ เท่ากับ 27

ค่าความสลับซับซ้อนของเวลาที่ใช้ใน Algorithm COMPUTE_E_VALUE(v, G) ขึ้นอยู่กับฟังก์ชันของ MAX_C_VALUE และ GREEDY_SCHEDULE ใช้ในการคำนวณหาค่าของเวลาเริ่มต้นที่น้อยที่สุด ฟังก์ชัน MAX_C_VALUE จะใช้ Fibonacci-heap[26] ช่วยในการเก็บค่าอาร์ค (u,w) ต่างๆที่ใช้ในเชื่อมต่อโหนด u เข้ามาหาโหนด v ที่อยู่ในกลุ่มงาน C เมื่อ $u \notin C$ และ $w \in C$ ดังนั้นในการสร้างกลุ่มงาน C สำหรับโหนด v ใดๆใน DAG G ที่มีการเรียกใช้ฟังก์ชัน MAX_C_VALUE ทั้งหมดเท่ากับ $O(|V| \lg |V| + E)$ ครั้ง ในขณะเดียวกันฟังก์ชัน GREEDY_SCHEDULE จะใช้ 2-3 tree[27] อัลกอริทึมช่วยในการเก็บโหนดต่างๆที่อยู่ภายในกลุ่มงาน C ในขณะนั้น แต่ละครั้งของการเรียกฟังก์ชัน GREEDY_SCHEDULE เพื่อทำการคำนวณหาค่า makespan ของลำดับการทำงานสำหรับโหนดเหล่านั้นบนหนึ่งตัวประมวลผลเท่ากับ $O(\lg |V|)$ เวลาโดยรวมของการเรียกฟังก์ชัน GREEDY_SCHEUDLE สำหรับโหนดต่างๆที่อยู่ภายในกลุ่มงาน C เท่ากับ $O(|V| \lg |V|)$ ฉะนั้นค่าความสลับซับซ้อนโดยรวมของ MAX_C_VALUE และ GREEDY_SCHEDULE ทั้งหมดที่อยู่ภายใน Algorithm COMPUTE_E_VALUE(v, G) เท่ากับ $O(|V| (|V| \lg |V| + |E|))$ ส่วนค่าความสลับซับซ้อนของเวลาที่ถูกใช้ในการสร้างชุดของกลุ่มงาน $\Phi(G)$ มีค่าเท่ากับ $O(|V| + |E|)$ ดังนั้นค่าความสลับซับซ้อนของเวลาที่ใช้ในการชุดกลุ่มงานโดยรวม(ทั้ง COMPUTE_E_VALUE(v, G) และ $\Phi(G)$)เท่ากับ $O(|V| (|V| \lg |V| + |E|))$

3.4.2 กรีดี้อัลกอริทึมการจัดกลุ่มงานและลำดับงานสำหรับสถาปัตยกรรมตัวประมวลผลแบบกระจายหน่วยความจำ(GTCS: A Greedy Task Clustering and Scheduling Algorithm for Distributed Memory Processor Architecture) เป้าหมายของจีทีซีเอสอัลกอริทึม(GTCS Algorithm)เพื่อใช้ในการจัดแบ่งงานและกำหนดลำดับการทำงาน โดยใช้กรีดี้อัลกอริทึมเข้ามาจัดการ โดยกลุ่มงานที่ได้จากจีทีซีเอสอัลกอริทึมจะเป็นลักษณะแบบกลุ่มงานที่มีงานซ้ำซ้อน ผลลัพธ์ที่ได้จากจีทีซีเอสอัลกอริทึมจะให้ค่าของ makespan ของลำดับการทำงานของกลุ่มงานมีค่าอย่างมากที่สุดเท่ากับสองเท่าที่ได้จากการจัดกลุ่มงานที่เหมาะสม จีทีซีเอสอัลกอริทึมจะมีค่าความสลับซับซ้อนของเวลาของอัลกอริทึมเท่ากับ $O(m(|V| \lg |V| + |E|))$ เมื่อ m คือจำนวนของกลุ่มงานที่ได้จากจีทีซีเอสอัลกอริทึม จีทีซีเอสอัลกอริทึมได้พัฒนามาจาก[23] โดยจีทีซีเอสอัลกอริทึมจะมีกลไกการสร้างกลุ่มงานที่แตกต่างไปจาก[23] ทำให้จีทีซีเอสอัลกอริทึมมีค่าความสลับซับซ้อนของเวลาน้อยกว่า[23] จีทีซีเอสอัลกอริทึมจะทำการสร้างกลุ่มงานเท่าที่ต้องการจะใช้เท่านั้น และการหาค่าเวลาเริ่มต้นของโหนด v ใดๆบน DAG G เร็วกว่า[23]

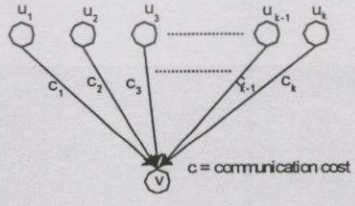
อัลกอริทึม จีทีซีเอส

ในการทำงานของจีทีซีเอสอัลกอริทึม จะใช้กรีดี้อัลกอริทึมเป็นหลักในการทำงาน จีทีซีเอสจะสร้างชุดของกลุ่มงาน, Φ, และทำการจัดลำดับการทำงานให้กับชุดของกลุ่มงานนั้น, $\Phi(G)$, สำหรับกราฟงาน Granularity ของ DAG Gใดๆ ดังที่เคยกล่าวมาแล้วว่าจีทีซีเอสอัลกอริทึมจะให้ค่า makespan ของลำดับการทำงานอย่างมากที่สุด เป็นสองเท่าของลำดับการทำงานของกลุ่มงานที่เหมาะสม และจีทีซีเอสได้พัฒนามาจาก[23] แต่จีทีซีเอสได้แสดงถึงวิธีการใหม่สำหรับการหาค่าเวลาเริ่มต้น, e(v), สำหรับโหนด v ใดๆที่อยู่บน DAG G โดยจะใช้เวลาน้อยกว่า[23]ที่ได้เคยทำได้ เนื่องมาจาก[23]ใช้เวลาค่อนข้างมากสำหรับการหาเวลาเริ่มต้น ทั้งนี้ขึ้นอยู่กับความลึกของ DAG G ด้วย ถ้าหาก DAG G มีความลึกมากเท่าใด วิธีการของ[23]จะใช้เวลาในการหาค่าเวลาเริ่มต้นมากขึ้นเท่านั้น โดยจีทีซีเอสอัลกอริทึมจะใช้ข้อมูลของ อิมมีเดียท-พรีดีเซสเซอร์(Immediate Predecessor) ในการหาค่าเวลาเริ่มต้น, e(v), ของโหนด v ใดๆที่อยู่บน DAG G จะมีข้อกำหนดว่าค่าเวลาเริ่มต้นของซอสโหนดจะค่าเท่ากับศูนย์เสมอ และเราจะไม่ทำการหาค่าเริ่มต้นให้กับซิ้งค์โหนด

วิธีการหาค่าเวลาเริ่มต้นของโหนด vใดๆที่อยู่บน DAG G, (e(v)), จะเริ่มจากการกำหนดให้ ω เป็นสัญลักษณ์แสดงถึงไดนามิกอะเรย์ใช้ในการเก็บซิ้งค์โหนดบน C_i และΨ เป็นสัญลักษณ์แสดงต้นไม้แบบทวิภาค(binary tree)ใช้ในการเก็บชุดของซิ้งค์โหนดบน C_i ที่อยู่บน Φ ดังนั้นซิ้งค์โหนดของ C_i ที่อยู่บน Ψ จะถูกจัดเก็บในรูปของต้นไม้แบบทวิภาค ในการเริ่มต้นส่วนประกอบตัวแรกของ ω จะถูกชี้และเป็นโหนดแรกสำหรับการเริ่มต้นสร้างกลุ่มงาน กำหนดΨ ถูกใช้เพื่อสำหรับการตรวจสอบว่ามีซิ้งค์โหนดเกิดขึ้นซ้ำซ้อนปรากฏอยู่บน Φ หรือไม่ ระหว่างการทำงานของอัลกอริทึม ทีจีทีซีเอสทำการสร้างชุดของกลุ่มงาน ในขณะเดียวกันจีทีซีเอสจะทำการหาซิ้งค์โหนดตัวใหม่ที่อยู่บน DAG G และทำการตรวจสอบว่าซิ้งค์โหนดตัวใหม่นี้ซ้ำกับซิ้งค์โหนดที่มีอยู่ในΨอีกหรือไม่ ดังนั้นหากซิ้งค์โหนดตัวใหม่ดังกล่าวปรากฏอยู่ใน Ψ จะทำให้จำนวนสมาชิกของ Ψ และ ω ไม่เพิ่มขึ้น แต่ถ้าหากซิ้งโหนดตัวใหม่ยังไม่มีอยู่ใน Ψ จำนวนสมาชิกของ Ψ และ ω จะมากขึ้น ในการหาเวลาเริ่มต้นของทุกๆ โหนด v, e(v), ยกเว้นซิ้งค์โหนดที่อยู่บน DAG G เราจะใช้อัลกอริทึมฟังก์ชัน Create_E_Value( ) และในการสร้างชุดของกลุ่มงานเราจะใช้อัลกอริทึมฟังก์ชัน Create_Cluster( ) ในตอนเริ่มต้นเราจะให้ ω และ Ψ เก็บซิ้งค์โหนดที่อยู่บน DAG G โดยเราจะใช้ DAG G ในรูปที่ 2.7 เป็นโจทย์ในการแก้ไขปัญหาเช่นเดียวกับวิธีการของทีซีเอส ดังนั้นในตอนเริ่มต้น Ψ และ ω จะทำการเก็บซิ้งค์โหนดของ DAG G ในรูปที่ 2.7 ได้ดังนี้ ω = {T8, T9, T10} โดยซิ้งค์โหนด T8 จะถูกใช้สำหรับการสร้าง C_i และ Ψ จะประกอบไปด้วยสมาชิกที่เหมือนกับ ω หลังจากที่ทำการสร้างกลุ่มงานที่มีซิ้งค์โหนด T8 เป็นโหนดเริ่มต้นสำหรับ C_i เสร็จสิ้นแล้ว ซิ้งค์โหนด T9 จะเป็นโหนดเริ่มต้นสำหรับการสร้างกลุ่มงาน C_i ชุดถัดไป ในการทำงานจะกระทำในลักษณะนี้จนกระทั่งสมาชิกที่อยู่ใน ω ได้ถูกกระทำหมดแล้ว

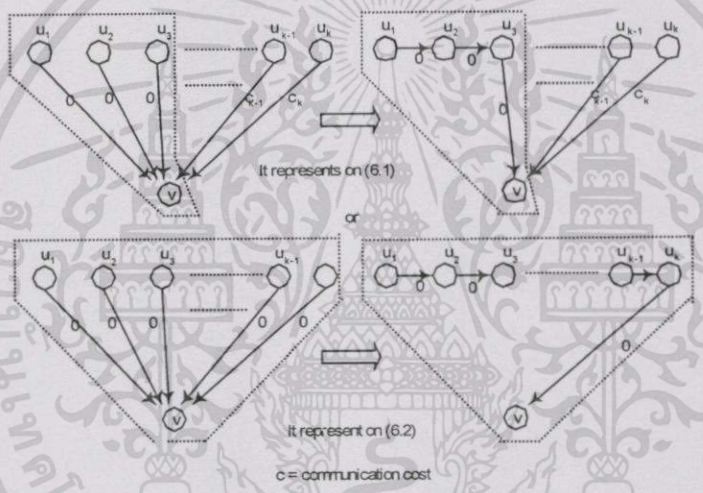
การหาค่าเวลาเริ่มต้นสำหรับโหนด v ใดๆ, e(v), จะพิจารณาจากค่าผลรวมของ ค่าใช้จ่ายในการติดต่อสื่อสาร($\lambda(u_i, v)$) ค่าใช้จ่ายในการประมวลผล($\mu(u_i)$) และเวลาเริ่มต้นของอิมมิเดียท-พรีดีเซสเซอร์ทุกๆโหนดที่เป็นของโหนด v ($S(u_i, P)$ = เป็นเวลาเริ่มต้นที่เป็นไปได้ของงานโหนดอิมมีเดียท-พรีดีเซสเซอร์ u_i ที่ตัวประมวลผล P) หรือแสดงอยู่ในรูปของ $\sum S(u_i, P) + \mu(u_i) + \lambda(u_i, v)$ โดย $1 \leq i \leq n$ เมื่อ n คือจำนวนของอิมมิเดียท-พรีดีเซสเซอร์สำหรับโหนดv ในการหาค่าเวลาเริ่มต้นสำหรับโหนด v ใดๆ, e(v), จะทำการเก็บแต่ละค่าของ $S(u_i, P) + \mu(u_i) + \lambda(u_i, v)$ อยู่ในรูปของต้นไม้แบบทวิภาคสำหรับโหนด u_i ใดๆและในการเข้าถึงค่าเหล่านี้ที่อยู่ในต้นไม้แบบทวิภาคจะเป็นแบบลักษณะ มาก→น้อย หรือ ขวา→ราก→ซ้าย ดังแสดงอยู่ในรูปที่ 3.17

```
ALGORITHM Create_E_Value(v,G)
BEGIN
        If v is a source node return zero;
        C = {v};   m = 0;
        c = first arc(u_i, v) incoming ;where arc(u_i, v) ∈ Ψ
        e(v) = c;
        while( m < c)do
                C = C ∪ {u_i};
                m = GREEDY_SCHEDULE(C – {v});
                c = next arc(u_i, v) incoming ; where u_i ∈ Ψ
                e(v) = min{e(v), max(m,c) };
        endwhile
        return e(v);
END
```



รูปที่ 3.17 แสดงอิมมีเดียท-ดีเซสเซอร์ของโหนด v ใดๆ โดยที่โหนด u_i จะวางอยู่บนตัวประมวลผล P ใดๆ

อัลกอริทึม Create_E_Value( ) สามารถให้ค่าของ e(v) ที่น้อยที่สุดสำหรับโหนด v ใดๆได้ โดยจะเริ่มต้นพิจารณาจากค่าที่สูงที่สุดของ $S(u_i, P) + \mu(u_i) + \lambda(u_i, v)$ ที่เป็นของอิมมีเดียท-พรีดีเซสเซอร์ และทำการพิจารณาค่าที่สูงที่สุดที่เหลืออยู่ของ $S(u_i, P) + \mu(u_i) + \lambda(u_i, v)$ ที่เป็นของอิมมีเดียท-พรีดีเซสเซอร์ถัดไป อัลกอริทึม Create_E_Value( ) จะให้ค่าของ e(v) ที่น้อยที่สุดก็ต่อเมื่อ ค่าของ GREEDY_SCHEDULE( )(วิธีการจะถูกกล่าวต่อมาในภายหลัง) มีค่าน้อยกว่าค่าของ c โดย GREEDY_SCHEDULE ( )จะทำการพิจารณาค่าของค่าใช้จ่ายในการติดต่อสื่อสารให้เท่ากับศูนย์ดังแสดงอยู่ในรูปที่ 3.18 หน้าที่ของฟังก์ชัน GREEDY_SCHEDULE( ) จะทำการจัดลำดับงานของอิมมีเดียท-พรีดีเซสเซอร์, หรือ u_i, ทั้งหมดที่อยู่บน C_i โดยการเรียงลำดับของค่าที่มากขึ้นของ $e(u_i)$ จากรูปที่ 3.18 แสดงให้เห็นว่าโหนด u_i ทั้งหมดที่ปรากฏอยู่ในกรอบจะถูกกระทำโดย GREEDY_SCHEDULE(C – {v}) สำหรับการสร้างลำดับการทำงานให้กับ u_i ทั้งหมด



รูปที่ 3.18 แสดงการหาเวลาเริ่มต้น e(v) สำหรับโหนด v ใดๆ และลำดับการทำงานของโหนด u_i ใดๆจะถูกทำโดย GREEDY_SCHEDULE(C – {v})

จากรูปที่ 3.18 เราสามารถแสดงสมการ การหาค่า e(v) ได้ดังนี้

$$e(v) = \sum_{i=1}^{j-1} (e(u_i) + \mu(u_i)) < (e(u_j) + \mu(u_j) + \lambda(u_j, v)) \qquad 3.8$$

หรือ

$$e(v) = \sum_{i=1}^{k} (e(u_i) + \mu(u_i)); If\, m < the - least - c(u_i, v) \qquad 3.9$$

ดังนั้นในการหาค่าเวลาเริ่มต้น e(v) สำหรับโหนด v ใดๆที่อยู่บน DAG G สามารถหาได้จากสมการที่ 3.8 หรือ 3.9

ขั้นตอนถัดไปเราต้องการสร้างชุดของกลุ่มงาน, Φ, โดยการใช้ e(v) ของโหนด v ต่างๆ ที่ได้เคยสร้างไว้ โดยใช้อัลกอริทึม Create_E_Value( ) ในการสร้างชุดของกลุ่มงาน, Φ,จะเริ่มต้นจากซิ้งค์โหนดที่อยู่ DAG G ก่อน ซึ่งซิ้งค์โหนดดังกล่าวได้ถูกจัดเก็บอยู่ใน ω อยู่แล้ว ในแต่ละครั้งของการสร้าง Φ จะทำให้ขนาดของ ω และ ψ ขยายใหญ่ขึ้น เพราะว่าซิ้งค์โหนดของกลุ่มงานตัวใหม่ได้ถูกเพิ่มเข้าไปใน ψ และ ω หรือ ψ และ ω อาจจะไม่ขยายเพิ่มมากขึ้นถ้าหากซิ้งค์โหนดของกลุ่มงานตัวใหม่ซ้ำซ้อนกับสมาชิกที่มีอยู่ใน ψ ในการสร้าง Φ เราจะใช้อัลกอริทึม Create_Cluster(v, G) ในแต่ละครั้งของการสร้างชุดกลุ่มงาน ทำให้ชุดกลุ่มงาน, Φ, ดังกล่าวได้รับ C_i เพิ่มมากขึ้น โดย C_i ตัวใหม่จะถูกใส่ลงเซตของ Φ ในขณะเดียวกันฟังก์ชัน GET_TASK_CROSS_CLUSTER( ) จะทำการหาซิ้งค์โหนดสำหรับกลุ่มงานตัวใหม่เข้ามา เมื่อฟังก์ชัน GET_TASK_CROSS_CLUSTER( ) สามารถหาซิ้งค์โหนดตัวใหม่ได้ จากนั้นจะทำการแทรกซิ้งค์โหนดตัวใหม่นี้เข้าไปใน ω และ ψ แต่มีข้อแม้ว่าซิ้งค์โหนดตัวใหม่นี้จะต้องไม่ซ้ำซ้อนกับชุดของซิ้งค์โหนดที่มีอยู่ใน ψ อยู่แล้ว ดังรายละเอียดได้เคยกล่าวมาแล้ว

```
ALGORITHM Create_Cluster(v,G)
BEGIN
    If v is a source node then return;
    C = {v};  m = 0;
    c = FIBANACCI_START_TIME(C);  e(v) = c;
    while( m < c) do
        C = C ∪ {u};
        m = GREEDY_SCHEDULE(C-{v});
        //let (u,w) be an arc where u ∉C, w∈C, and
        //c=c(u,w)
        c = FIBANACCI_START_TIME(C);
        e(v) = min{e(v), max(m,c)};
    endwhile
    GET_TASK_CROSS_CLUSTER();
    return e(v);
END
```

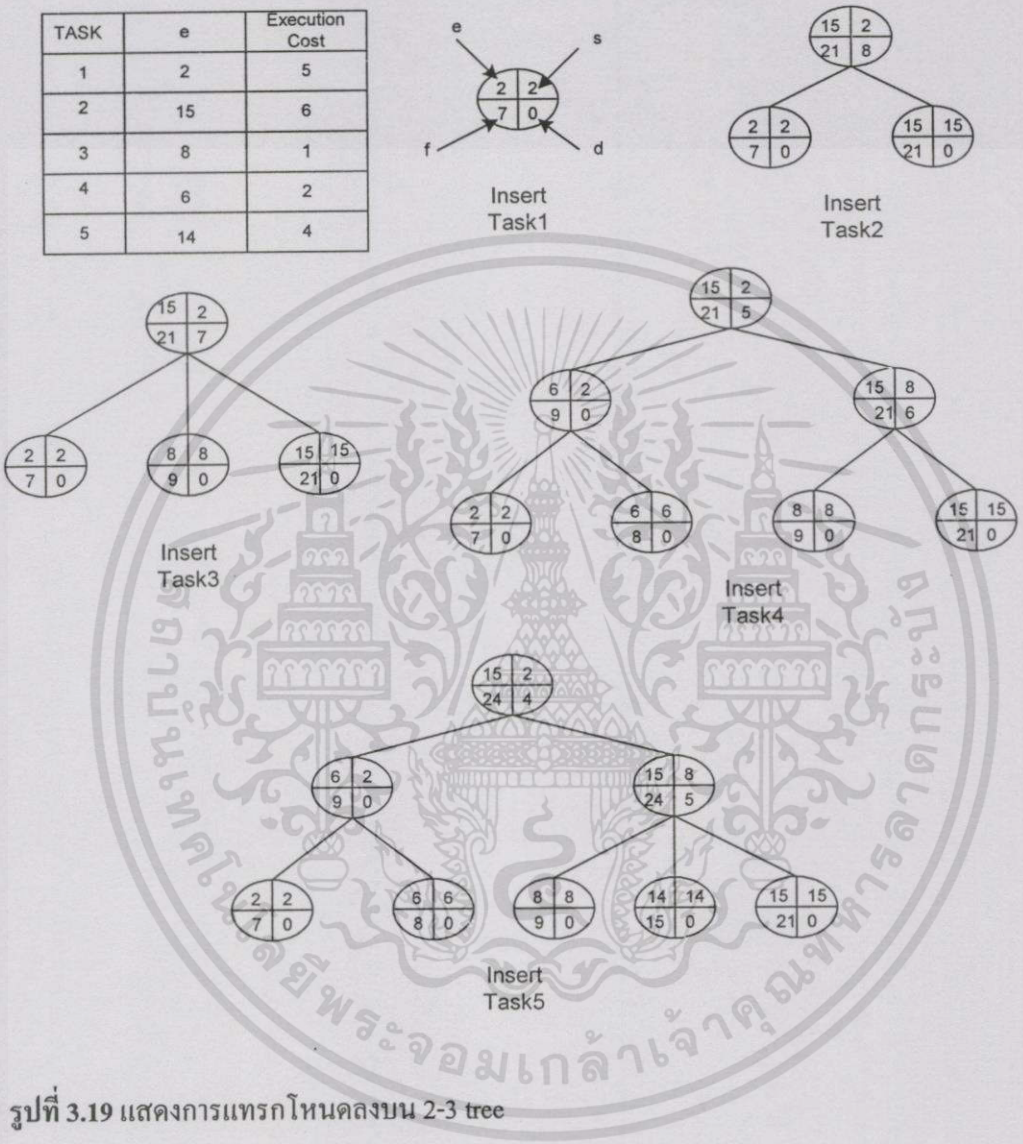
จะสังเกตพบได้ว่าอัลกอริทึม Create_Cluster(v, G) จะคล้ายกับอัลกอริทึม COMPUTE_E_VALUE(v, G) ที่ปรากฏอยู่ในวิธีการของทีซีเอส จะต่างกันตรงที่อัลกอริทึม Create_Cluster(v, G)ได้เพิ่มฟังก์ชัน GET_TASK_CROSS_CLUSTER( ) เข้ามา จากการกระทำดังกล่าวทำให้กลไกการสร้างชุดกลุ่มงานโดยวิธีการของจีทีซีเอส แตกต่างไปจากวิธีการของทีซีเอส นอกจากนี้ยังทำให้ค่าของความสลับซับซ้อนของเวลาของอัลกอริทึมลดลงจากเดิม ดังจะกล่าวต่อมาในภายหลัง

หลักการทำงานของฟังก์ชัน GREEDY_SCHEDULE(C – {v}) โดยใช้ 2-3 tree อัลกอริทึมในการจัดลำดับการทำงาน ให้กับงานของโหนดต่างๆที่เป็นอิมมีเดียท-พรีดีเซสเซอร์ของโหนด vใดๆในกรณีของอัลกอริทึม Create_E_Value() และงานของโหนดต่างๆที่เป็นพรีดีเซสเซอร์ของโหนด v ใดๆในกรณีของอัลกอริทึม Create_Cluster() โดยฟังก์ชัน GREEDY-SCHEDULE(C – {v})ในอัลกอริทึม Create_E_Value() และ Create_Cluster() จะทำงานเหมือนกันทุกอย่าง ความแตกต่างอยู่ตรงที่อัลกอริทึม Create_E_Value()จะพิจารณาเฉพาะโหนดที่เป็นอิมมีเดียท-พรีดีเซสเซอร์ของโหนด v ใดๆ และอัลกอริทึม Create_Cluster()จะพิจารณาทุกๆโหนดที่เป็นพรีดีเซสเซอร์ของโหนด v ใดๆ หลักการทำงานของ 2-3 tree โดยในรายละเอียดจะปรากฏอยู่ใน[27] ฟังก์ชัน GREEDY-SCHEDULE(C-{v}) จะให้ค่า makespan (หรือเท่ากับ m ในอัลกอริทึม Create_E_Value() และ Create_Cluster()) ซึ่งเป็นเวลาที่เสร็จสิ้นการทำงานของโหนดต่างๆที่เป็นอิมมีเดียท-พรีเซสเซอร์ของโหนด v ในอัลกอริทึม Create_E_Value() และโหนดต่างๆที่เป็นพรีดีเซสเซอร์ของโหนด v ในอัลกอริทึม Create_Cluster()(ดังแสดงอยู่ในรูปที่ 3.14 และ 3.22) รูปที่ 3.19 และ 3.20 แสดงขั้นตอนการทำงานของฟังก์ชัน GREEDY_SCHEDULE(C-{v}) ในรูปที่ 3.19 แสดงการแทรกโหนดลงบน 2-3 tree และรูปที่ 3.20 แสดงวิธีการปรับ(update)ค่าของคีย์ที่อยู่บนโหนดเมื่อมีการแทรกโหนดใหม่เข้ามาใน 2-3 tree โหนดแต่ละโหนดที่อยู่บน 2-3 tree จะประกอบไปด้วยค่าของคีย์ต่างๆดังนี้

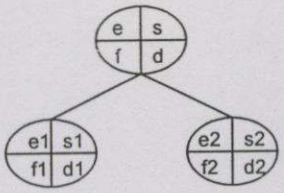
1. e เป็นค่าของคีย์สำหรับค่าของเวลาเริ่มต้นที่มากที่สุดของโหนดหรืองานต่างๆที่อยู่ใน 2-3 tree ในขณะนั้น
2. s เป็นค่าของคีย์สำหรับเวลาเริ่มต้นของงานหรือโหนดที่จะถูกประมวลผลเป็นงานแรกสุด
3. f เป็นค่าของคีย์สำหรับเวลาที่สิ้นสุดหรือเสร็จสิ้นการทำงานของงานหรือโหนดที่จะถูกประมวลผลหลังสุด
4. d เป็นค่าของคีย์สำหรับเวลาที่หยุดนิ่งที่เกิดขึ้น ในการจัดลำดับการทำงานให้กับงานหรือโหนดต่างๆที่อยู่บน 2-3 tree

สังเกตพบได้ว่า 2-3 tree ในฟังก์ชันGREEDY_SCHEDULE() สามารถช่วยให้การจัดลำดับการทำงานของโหนดต่างๆที่เป็นอิมมีเดียท-พรีดีเซสเซอร์ในอัลกอริทึม Create_E_Value()

และพรีดีเซสเซอร์ในอัลกอริทึม Create_Cluster() ได้อย่างมีประสิทธิภาพ และเวลาที่ใช้ในการเข้าถึงโหนดหรืองานต่างๆที่อยู่บน 2-3 tree ในแต่ละครั้งใช้เวลาเพียง O(lg |V|) เท่านั้น รายละเอียดของวิธีการในการแทรกโหนดและการเข้าถึงโหนดต่างๆบน 2-3 tree จะปรากฏอยู่ใน[27]

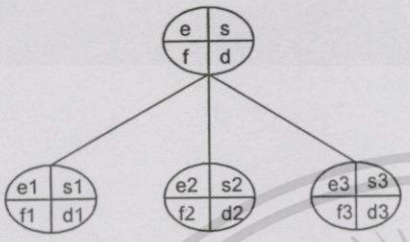


รูปที่ 3.19 แสดงการแทรกโหนดลงบน 2-3 tree



e = e2
s = s1
f = f2 + max{0, f1 - (s2 + d2)}
d = d1 + max{0, (s2 + d2) - f1}

(a) algorithms updating key value with two children



e = e3
s = s1
f' = f2 + max{0, f1 - (s2 + d2)}
d' = d1 + max{0, (s2 + d2) - f1}
f = f3 + max{0, f' - (s3 + d3)}
d = d' + max{0, (s3 + d3) - f'}

(b) algorithms updating key value with three children

รูปที่ 3.20 แสดงอัลกอริทึมในการปรับค่าของคีย์ต่างๆที่อยู่บนโหนด เมื่อมีการแทรกโหนดใหม่เกิดขึ้นลงบน 2-3 tree (a) อัลกอริทึมที่ใช้ในการปรับค่าของคีย์เมื่อมีจำนวนของโหนดลูกหลานจำนวน 2 โหนด (b) อัลกอริทึมที่ใช้ในการปรับค่าของคีย์เมื่อมีจำนวนของโหนดลูกหลานจำนวน 3 โหนด

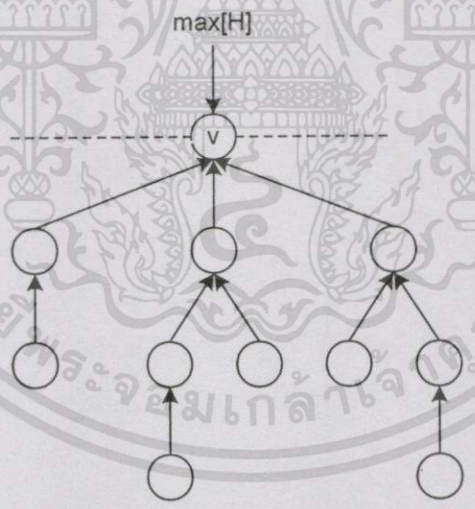
ต่อมาเราจะทำการพิจารณาการทำงานของฟังก์ชัน FIBANACCI_START_TIME() วิธีการที่สำคัญซึ่งเป็นหัวใจของฟังก์ชัน FIBANACCI_START_TIME() คือการนำเอาอัลกอริทึม Fibancci-heap [26] เข้ามาช่วยในการจัดเก็บอาร์ค(u,w) ต่างๆที่ใช้ในการเชื่อมต่อโหนด u เข้ามาหาโหนด v ที่อยู่ในกลุ่มงาน C เมื่อ $u \notin C$ และ $w \in C$ หรือใช้ในการพิจารณาในการนำเอาโหนดบรรพบุรุษต่างๆ นำมาใส่ไว้ในกลุ่มงาน C หลักการทำงานของ FIBANACCI_START_TIME() จะประกอบไปด้วยขั้นตอนการทำงานของ ATTRACK_MAX()[26] และ INCREASE_KEY()[26] โดย ATTRACK_MAX() และ INCREASE_KEY() เป็นฟังก์ชันที่มีกลไกสำคัญในการทำงานของ Fibanacci-heap

ATTRACK_MAX() เป็นฟังก์ชันของกระบวนการในการถอดถ้อนเอาโหนดที่มีค่าของคีย์ที่มากที่สุด ที่อยู่บนฮีป(heap) ออกไปจากฮีป ขั้นตอนในการทำงานของ ATTRACK_MAX() เป็นกระบวนการที่สลับซับซ้อนมากที่สุดในการทำงานของ Fibanacci-heap ค่าของความสลับซับซ้อนในการทำงานของฟังก์ชันนี้เท่ากับ O(lg |V|) amortized time ในการกระทำกับโหนดใดๆ

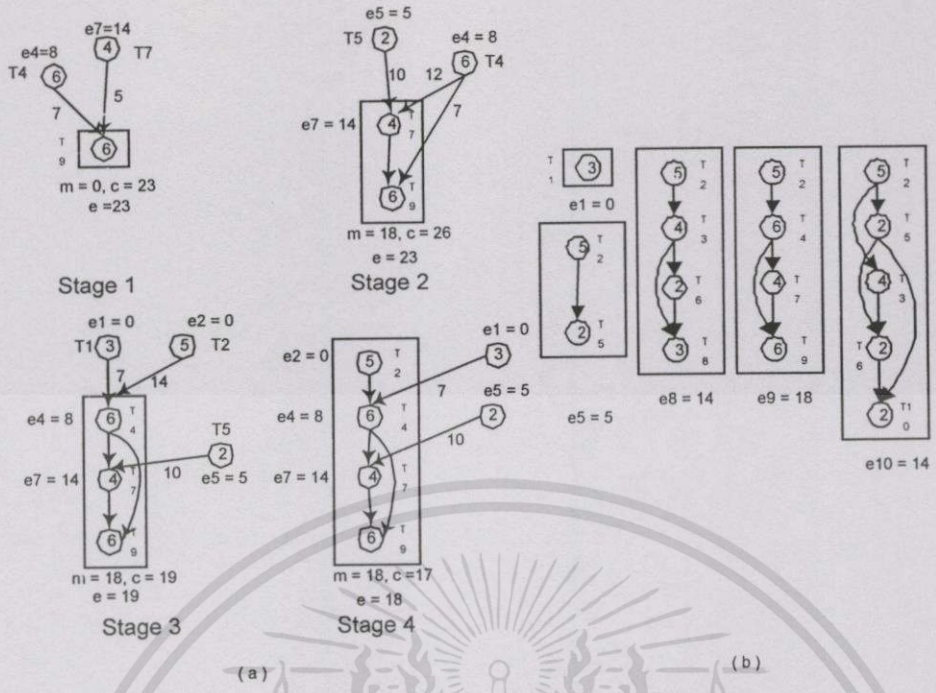
INCREASE_KEY() เป็นฟังก์ชันของกระบวนการในการเพิ่มค่าของคีย์ให้กับโหนดที่อยู่บนฮีป โดยในขั้นตอนนี้จะมีค่าของความสลับซับซ้อนในการทำงานของฟังก์ชันเท่ากับ O(1) amortized time ในการกระทำกับโหนดใดๆ

โดยสรุปแล้วกระบวนการทำงานทั้งหมดที่เกิดขึ้นในขั้นตอนการทำงานของ FIBANACCI_START_TIME() ที่ประกอบไปด้วยการทำงานของฟังก์ชัน ATTRACK_MAX() จะถูกกระทำอย่างมากที่สุดจำนวน V ครั้ง เมื่อ V เป็นจำนวนของโหนดที่อยู่บน DAG และฟังก์ชัน INCREASE_KEY() จะถูกกระทำอย่างมากที่สุดจำนวน E ครั้ง เมื่อ E เป็นจำนวนของอาร์คทั้งหมดที่อยู่บน DAG จากฟังก์ชั้นทั้งสองทำให้เราทราบจำนวนของค่าความสลับซับซ้อนทั้งหมดที่เกิดขึ้นบน FIBANACCI_START_TIME() เท่ากับ O(|V| lg |V| + |E|) amortized time ที่เกิดขึ้นในการสร้างกลุ่มงานแต่ละกลุ่มงาน

ดังนั้นทำให้เราสามารถสรุปได้ว่า อัลกอริทึม Create_Cluster() จะประกอบไปด้วยสองฟังก์ชันที่มีบทบาทสำคัญในการทำงานคือ 1) ฟังก์ชัน FIBANACCI_START_TIME() และ 2) ฟังก์ชัน GREEDY_SCHEDULE() ทำให้อัลกอริทึม Create_Cluster() จะต้องใช้เวลาในการทำงานเท่ากับ O(|V| lg|V| + |E|) + O(|V| lg|V|) สำหรับในการสร้างชุดของกลุ่มงานแต่ละกลุ่มงาน รูปที่ 3.21 แสดงตัวอย่างของ Fibanacci-heap ในจีทีซีเอส อัลกอริทึม



รูปที่ 3.21 แสดงลักษณะโครงสร้างของ Fibanacci-heap ในจีทีซีเอส อัลกอริทึม

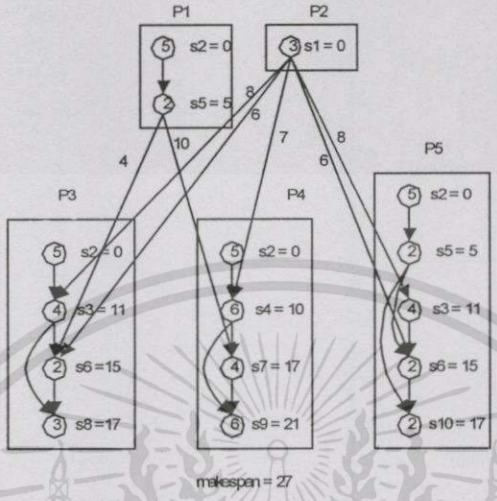


รูปที่ 3.22 (a) แสดงขั้นตอนการสร้างกลุ่มงานสำหรับโหนด T8 (b) แสดง $\Phi(G)$ โดยใช้จีทีซีเอส

หลักการทำงานของอัลกอริทึม Create_Cluster(v, G)จะคล้ายกับอัลกอริทึม Create_E_Value(v, G) ความแตกต่างระหว่างอัลกอริทึมทั้งสองอยู่ตรงที่อัลกอริทึม Create_E_Value(v, G) จะพิจารณาเฉพาะอิมมีเดียท-พรีดีเซสเซอร์เท่านั้นที่ชี้เข้าหาโหนด v อยู่ในขณะนั้น แต่หากอัลกอริทึม Create_Cluster(v, G) จะพิจารณาถึงอิมมีเดียท-พรีดีเซสเซอร์ทั้งหมดที่ชี้เข้าหาโหนดทุกๆตัวที่อยู่ใน C_i ในขณะนั้น ดังนั้นในการสร้าง C_i ใดๆของอัลกอริทึม Create_Cluster(v, G)จะเริ่มจากการแทรกซิ้งค์โหนดของ DAG G ที่อยู่บน ω เป็นโหนดเริ่มต้นของกลุ่มงาน C_i กลุ่มงานหนึ่งๆหรือ C_i ใดๆจะขยายใหญ่ได้มากขึ้นจากการแทรกอิมมีเดียท-พรีดีเซสเซอร์ของโหนดต่างๆที่อยู่ใน C_i จากการทำดังกล่าวของอัลกอริทึม Create_Cluster(v, G) ทำให้ได้ค่าของเวลาเริ่มต้นที่น้อยที่สุดของซิ้งค์โหนด และได้ชุดของกลุ่มงาน C_i สำหรับซิ้งค์โหนดตัวนั้น

สำหรับตัวอย่างการทำงานของอัลกอริทึมทั้งสองจะใช้ DAG Gในรูปที่ 2.7 เป็นโจทย์ของปัญหา ในตอนเริ่มต้นอัลกอริทึม Create_Cluster(v, G)จะใช้ซิ้งค์โหนด T8 เป็นโหนดเริ่มต้นสำหรับการสร้าง C_i เพราะโหนดดังกล่าวเป็นโหนดแรกที่อยู่ใน ω หลังจากที่อัลกอริทึม Create_Cluster(v, G) ทำงานเสร็จสิ้นแล้ว เราจะได้ C_i กลุ่มแรกดังนี้ C_i = {T8, T6, T3, T2} ในขณะเดียวกัน ω และψ จะได้สมาชิกเพิ่มมากขึ้นดังนี้ ω = {T8, T9, T10, T5, T1} และ ψ {T8, T9, T10, T5, T1} ผลดังกล่าวได้มาจากการทำงานของฟังก์ชัน GET_TASK_CROSS_CLUSTER( ) โดยที่โหนด T5 และ T1 เป็นซิ้งค์โหนดตัวใหม่สำหรับการสร้าง C_i กลุ่มถัดไป ส่วนในรายละเอียดขั้นตอนการทำงานของอัลกอริทึม Create_Cluster(v, G) จะเหมือนกับอัลกอริทึม

COMPUTE_E_VALUE(v, G) ที่อยู่ในวิธีการของทีซีเอส ดังนั้นจะไม่ขอกล่าวเพิ่มอีก รูปที่ 3.22 เป็นการแสดงชุดของกลุ่มงาน, Φ, ที่ได้จากอัลกอริทึม Create_E_Value(v, G) และอัลกอริทึม Create_Cluster(v, G)



รูปที่ 3.23 แสดงผลการสร้าง $\Phi(G)$ โดยใช้กลุ่มงานต่างๆในรูปที่ 3.22

สำหรับการจัดลำดับการทำงานให้กับชุดของกลุ่มงาน, Φ, เราจะไม่ใช้อัลกอริทึมใดเลยเข้ามาช่วย ซึ่งต่างไปจากวิธีการของทีซีเอสที่ต้องการเอาอัลกอริทึมเข้ามาช่วยในการจัดลำดับการทำงาน การจัดลำดับการทำงานของจีทีซีเอสเพียงแค่วางกลุ่มงานต่างๆที่ได้จากในขั้นตอนที่แล้ว ลงบนตัวประมวลผลต่างๆ โดยจะวางหนึ่งกลุ่มงานต่อหนึ่งตัวประมวลผล และงานต่างๆที่อยู่ในกลุ่มงานนั้นจะทำงานเรียงตามลำดับที่ตัวประมวลผลตัวเดียวนั้น รูปที่ 3.23 เป็นการแสดงการวางกลุ่มงานต่างๆลงบนตัวประมวลต่างๆ จำนวน 5 ตัว หลังจากที่วางกลุ่มงานต่างๆลงบนตัวประมวลผล จะทำให้เห็นว่าค่าเวลาเริ่มต้นของโหนดต่างๆได้เปลี่ยนแปลงไป ผลดังกล่าวทำให้ได้ค่า makespan ที่ได้จากวิธีการของอัลกอริทึมจีทีซีเอส และสังเกตได้ว่าค่า makespan ของอัลกอริทึมจีทีซีเอสจะเท่ากับค่า makespan ของอัลกอริทึมทีซีเอสด้วย ดังนั้นเราสามารถกล่าวได้ว่าวิธีการของอัลกอริทึมจีทีซีเอสให้ผลลัพธ์ที่ไม่ต่างไปจากวิธีการของอัลกอริทึมทีซีเอสเลย

การวิเคราะห์ค่าความสลับซับซ้อนของอัลกอริทึมจีทีซีเอสจะประกอบไปด้วย ค่าความสลับซับซ้อนของอัลกอริทึม Create_E_Value( ) และอัลกอริทึม Create_Cluster( ) โดยอัลกอริทึม Create_E_Value( ) จะให้ค่าเวลาเริ่มต้น, e(v), สำหรับโหนด v ใดๆ โดยพิจารณาจากอาร์คของอิมมีเดียท-พรีดีเซสเซอร์ของโหนด v นั้นๆ กระบวนการทำงานอัลกอริทึม Create_E_Value( )จะพิจารณาจาก $S(u_i, P) + \mu(u_i) + \lambda(u_i, v)$ ต่างๆของอิมมีเดียท-พรีดีเซสเซอร์ซึ่งถูกจัดเก็บอยู่ในลักษณะของต้นไม้แบบทวิภาค ดังนั้นค่าความสลับซับซ้อนของอัลกอริทึมเท่ากับ O(c lg c) สำหรับ

แต่ละโหนด v_i ใดๆ เมื่อ c คือจำนวนของอาร์คที่เป็นของอิมมีเดียท-พรีดีเซสเซอร์ของโหนด v_i และ $c \in E$ โดยที่ $c << E$ และ $v_i \in V$ ทำให้เราสามารถสรุปได้ว่าค่าความสลับซับซ้อนของเวลาในการหาค่าเวลาเริ่มต้นสำหรับโหนด v ใดๆ โดยที่ยกเว้นซิ้งค์โหนดที่อยู่บน DAG G เท่ากับ O(n(c lg c)) เมื่อ n เป็นจำนวนโหนดที่อยู่บน DAG G โดยยกเว้นซิ้งค์โหนด

สำหรับการวิเคราะห์ค่าความสลับซับซ้อนของอัลกอริทึม Create_Cluster( ) จะเหมือนกับอัลกอริทึม COMPUTE_E_VALUE( ) ที่อยู่ในอัลกอริทึมทีซีเอส ดังนั้นจะไม่ขอกล่าวในที่นี้อย่างละเอียด โดยสรุปแล้วอัลกอริทึม Create_Cluster( ) จะใช้เวลาไปเท่ากับ O(|V| lg |V| + |E|) + O(|V| lg |V|) สำหรับแต่ละ C_i ดังนั้นค่าความสลับซับซ้อนของเวลาของอัลกอริทึมในการสร้างชุดกลุ่มงานเท่ากับ O(m(|V| lg |V| + |E|)) เมื่อ m เป็นจำนวนของกลุ่มงาน C_i โดยที่ V และ E เป็นจำนวนของโหนดและอาร์คที่อยู่บน DAG G และ $1 \leq m < V$

บทที่ 4
การจัดกลุ่มงาน และลำดับการทำงานของเอทีซีเอสอัลกอริทึม

วัตถุประสงค์ของวิทยานิพนธ์นี้เพื่อเสนอวิธีการ การจัดกลุ่มงานและลำดับการทำงานของโปรแกรมแบบขนาน ที่อยู่บนระบบเครือข่าย โดยใช้ DAG เพื่อแสดงทิศทางการทำงาน และลำดับการทำงานที่มีความสัมพันธ์กัน เมื่อโปรแกรมแบบขนานนี้จะต้องถูกกำหนดให้ประมวลผลอยู่บนเครือข่ายที่มีลักษณะของสถาปัตยกรรมแบบกระจายหน่วยความจำ Adaptive Task Clustering and Scheduling Algorithm หรือ ATCS/เอทีซีเอสอัลกอริทึม เป็นการวิธีการในการจัดกลุ่มงานที่กระจายอยู่บนระบบเครือข่ายให้เป็นกลุ่มๆ และแต่ละกลุ่มจะทำงานอยู่บนหนึ่งตัวประมวลผล ทำให้ช่วยลดการเกิดโอเวอร์เฮดของค่าใช้จ่ายในการติดต่อสื่อสารระหว่างตัวประมวลผล เพราะระบบเครือข่ายที่มีโอเวอร์เฮดมากๆจะทำให้โปรแกรมแบบขนานต้องใช้เวลาในการประมวลผลค่อนข้างมาก โดยเฉพาะอย่างยิ่งโปรแกรมแบบขนานที่มีลักษณะเป็นแบบเม็ดเนื้องานแบบละเอียด(Granularity < 1) ดังที่ได้เคยกล่าวมาแล้วในบทที่ 2 โดยเอทีซีเอสอัลกอริทึมจะพยายามลดโอเวอร์เฮดให้น้อยลง โดยทำการแปลงลักษณะเม็ดเนื้องานแบบละเอียดให้เป็นลักษณะเม็ดเนื้องานแบบหยาบ ทำให้จำนวนของตัวประมวลผลลดลงตามด้วย เมื่อจำนวนของตัวประมวลผลลดน้อยลงค่าใช้จ่ายในการติดต่อสื่อสารระหว่างงานหรือตัวประมวลผลลดน้อยลงตามไปด้วย ผลของกลุ่มงานที่ได้จากเอทีซีเอสอัลกอริทึมจะได้ชุดของกลุ่มงานแบบมีงานซ้ำซ้อนเกิดขึ้นในระหว่างกลุ่มงาน ปัญหาการจัดกลุ่มงานจัดได้ว่าเป็นปัญหาแบบ NP-complete แม้ว่าจะไม่มีข้อจำกัดของการกำหนดจำนวนตัวประมวลผลที่จะใช้และยอมให้มีการกำหนดงานที่ซ้ำซ้อนเกิดขึ้นได้ก็ตาม เอทีซีเอสอัลกอริทึมจะใช้แบล็คแทรกกิ้งอัลกอริทึม(BackTracking Algorithm)ในการจัดกลุ่มงานที่อยู่บน DAG หลังจากที่ DAG ได้ถูกแปลงให้อยู่ในรูปของต้นไม้กลับตัวชี้(Inverse Pointer Tree หรือ Intree) และเอทีซีเอสอัลกอริทึมจะมีค่าความสลับซับซ้อนของเวลาของอัลกอริทึมเท่ากับ $O(|V^2|)$ เมื่อ V คือจำนวนของโหนดที่อยู่บน DAG

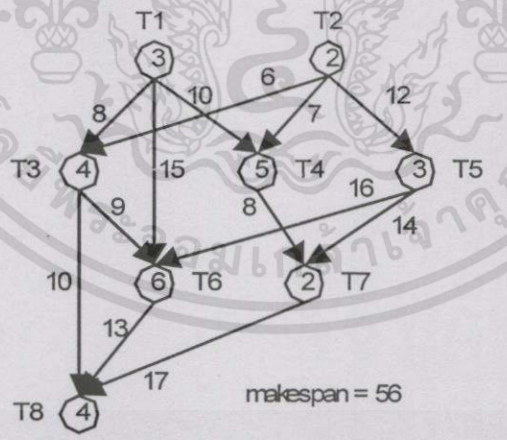
4.1 ข้อกำหนดและขอบเขตของเอทีซีเอส อัลกอริทึม

ในการแสดงขั้นตอนการทำงานของ เอทีซีเอสอัลกอริทึมจะใช้ DAG แสดงถึงทิศทางการทำงานของงานและลำดับความสัมพันธ์ก่อนและหลังของงานของโปรแกรมแบบขนาน โดยโปรแกรมแบบขนานถูกกำหนดรูปแบบด้วย DAG $G = (V, E, \mu, \lambda)$ เมื่อ V คือจำนวนของงานหรือโหนดที่อยู่บน DAG E คือจำนวนของอาร์คที่ปรากฏอยู่บน DAG ส่วน μ เป็นการแสดงค่าใช้

จ่ายในการประมวลผลของงานที่ตัวประมวลผล และ λ เป็นการแสดงค่าใช้จ่ายในการติดต่อสื่อสารระหว่างงานสองงานที่อยู่บนตัวประมวลผลที่ต่างกัน กำหนดให้ $\forall v_i \in V$, $\forall e_i \in E$, $\mu(v_i)$ แสดงถึงค่าใช้จ่ายในการประมวลของงานโหนด v_i ใดๆ และ$\lambda(u_i, v_i)$ เป็นการแสดงถึงค่าใช้จ่ายในการติดต่อสื่อสารระหว่างงานหรือโหนด u_i และ v_i โดยที่งาน u_i จะต้องเกิดขึ้นและเสร็จสิ้นสมบูรณ์ก่อน จากนั้นงาน u_i จะต้องขนถ่ายข้อมูลทั้งหมดมายังงาน v_i งาน v_i จึงจะสามารถเริ่มต้นทำงานได้

DAG ที่ใช้แสดงงานของโปรแกรมแบบขนานจะประกอบไปด้วย ข้อมูลของหน่วยการประมวลผลของงานที่โหนด v_i ใดๆ และหน่วยของการขนถ่ายข้อมูลระหว่างตัวประมวลผล เงื่อนไขที่ใช้ภายใต้การทำงานของเอทีซีเอสอัลกอริทึมมีอยู่ว่า ความเร็วในการประมวลผลของตัวประมวลผล และอัตราความเร็วในการขนถ่ายข้อมูลของลิ้งค์ที่ใช้ในการเชื่อมต่อระหว่างตัวประมวลผล กำหนดให้เท่ากันทั้งหมดในระบบ

ในการทดสอบประสิทธิภาพของเอทีซีเอสอัลกอริทึม จะประกอบไปด้วย DAG ที่มีลักษณะต่างๆกัน ทั้งที่แบบสร้างขึ้นและเป็นแบบที่ได้จากการสุ่ม(Random) ลักษณะของ DAG ที่สร้างขึ้นได้มาจากตัวอย่างในหนังสือ[6][28][30][31]และบทความวิจัยทางวิชาการ[2][32] เช่น DAG ในรูปที่ 2.7 ได้มาจาก[2] เป็นต้น ส่วน DAG ที่ได้จากการสุ่มโดยจะให้โปรแกรมเป็นตัวสร้างขึ้นมาเอง ค่าทุกค่าที่ประกอบกันอยู่บน DAG เช่น ค่าใช้จ่ายในการประมวลผลของงาน ค่าใช้จ่ายในการติดต่อสื่อสารระหว่างงาน จำนวนของโหนดที่อยู่บน DAG จำนวนของอาร์คที่ออกจากโหนดแต่ละโหนด ค่าเหล่านี้ได้มาจากการสุ่มทั้งหมด และลักษณะของ DAG ที่ได้จากการสุ่มในการทดสอบอัลกอริทึมจะเป็นเม็ดเนื้องานแบบละเอียด

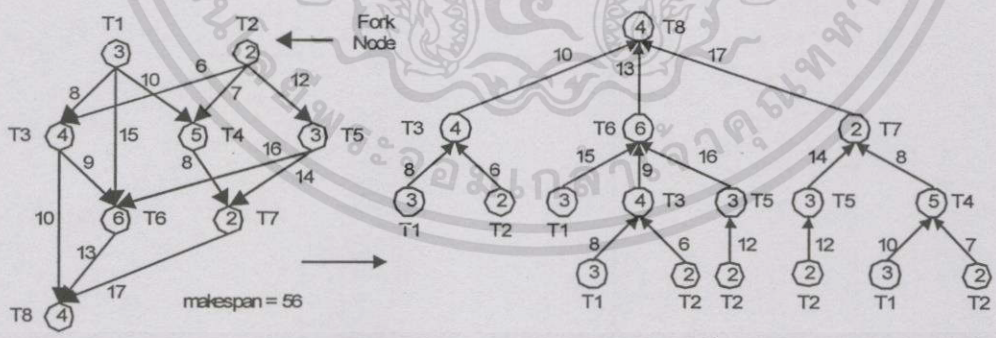


รูปที่ 4.1 แสดง DAG

4.2 ขั้นตอนการทำงานของเอทีซีเอส อัลกอริทึม

ดังที่ได้เคยกล่าวไว้แล้วว่า เอทีซีเอสอัลกอริทึมจะใช้แบล็คแทรกกิ้งอัลกอริทึมเป็นตัวช่วยในการจัดกลุ่มงาน ผลลัพธ์ที่ได้คือชุดของกลุ่มงานมาหนึ่งชุด โดยที่งานเหล่านี้จะปรากฏอยู่บนต้นไม้ที่ได้รับการแปลงมาจาก DAG ขั้นตอนการสร้างชุดของกลุ่มงานและการจัดลำดับการทำงานประกอบไปด้วย 4 ขั้นตอนดังนี้ ขั้นตอนที่หนึ่งทำการแปลง DAG ให้อยู่ในรูปของต้นไม้กลับตัวชี้ ขั้นตอนที่สองทำการแปลงต้นไม้กลับตัวชี้ให้อยู่ในรูปของต้นไม้แบบทวิภาคกลับตัวชี้(Inverse Pointer Binary Tree) ขั้นตอนที่สามจะทำการสร้างชุดของกลุ่มงาน และในขั้นตอนที่สี่จัดลำดับการทำงานให้กับชุดของกลุ่มงานที่ได้จากในขั้นตอนที่สาม ในตัวอย่างการทำงานเราจะใช้ DAG ในรูปที่ 4.1 เป็นตัวอย่าง

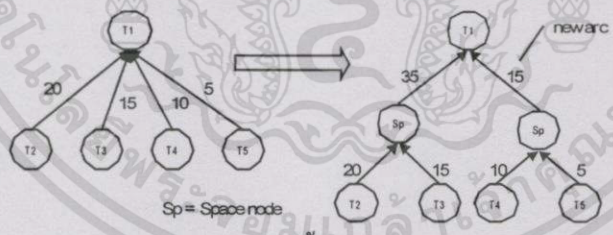
<u>ขั้นตอนที่หนึ่ง</u> การแปลง DAG ให้อยู่ในรูปแบบของต้นไม้กลับตัวชี้ ในการแปลง DAG ให้อยู่ในรูปแบบของต้นไม้กลับตัวชี้ การทำงานภายในขั้นตอนนี้จะเริ่มต้นพิจารณาจากซอสโหนดที่อยู่บน DAG ก่อน จากนั้นทำการพิจารณาโหนดอื่นๆในระดับ(Level)ถัดลงมา โดยระดับของโหนดที่อยู่บน DAG จะเหมือนกับระดับของโหนดที่อยู่บนต้นไม้ ให้ทำการพิจารณาจากทุกๆฟอกโหนดบน DAG นั้น ดังที่ได้เคยกล่าวเอาไว้แล้วว่าฟอกโหนดคือโหนดที่มีอาร์คชี้ออกจากโหนดนั้นๆ รูปที่ 4.2 เป็นตัวอย่างของต้นไม้กลับตัวชี้ที่ได้จากการแปลง DAG ในรูปที่ 4.1 จากรูปจะสังเกตเห็นได้ว่าเกิดงานซ้ำซ้อนเกิดขึ้นปรากฏอยู่บนต้นไม้กลับตัวชี้ เช่น เดิมจาก DAG ในรูปที่ 4.1 มีโหนด T2 เป็นซอสโหนดปรากฏอยู่ และโหนด T2 นี้เป็นฟอกโหนดด้วย ดังนั้นโหนด T2 จะปรากฏเป็นโหนดที่ซ้ำซ้อนอยู่บนต้นไม้



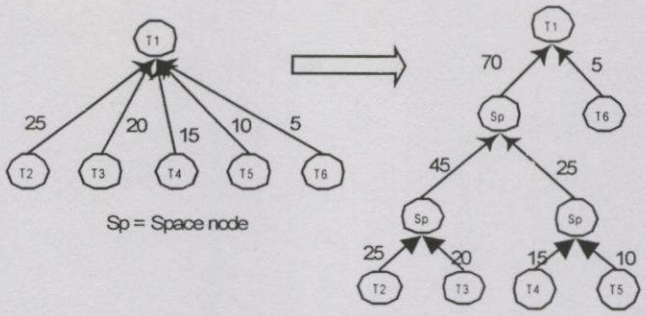
รูปที่ 4.2 ผลจากการแปลง DAG ให้อยู่ในรูปแบบของต้นไม้กลับตัวชี้

<u>ขั้นตอนที่สอง</u> การแปลงต้นไม้กลับตัวชี้ให้อยู่ในรูปของต้นไม้แบบทวิภาคกลับตัวชี้ โดยในขั้นตอนนี้จะทำการแปลงโหนดใดๆที่มีจำนวนของโหนดลูกหลาน(Successor node)มากกว่าสองโหนดขึ้นไป ให้ลดลงเหลือเพียงสองโหนดลูกหลาน โดยที่โหนดเหล่านั้นอาจจะเป็นโหนด

ราก(Root Node) หรือโหนดรากต้นไม้ส่วนย่อย(Sub-tree Root Node) หลักในการแปลงโหนดลูกหลานของโหนดรากหรือโหนดรากของต้นไม้ส่วนย่อยให้เหลือเพียงสองโหนดลูกหลาน โดยการนำเอาสองโหนดลูกหลานสองตัวแรกที่มีค่าของ λ ที่มากที่สุดหรือเส้นทางวิกฤตมารวมกัน ผลที่ได้จะได้รับ λ_{new} ซึ่งเกิดจากการรวมกันของ λ ของสองโหนดลูกหลาน($\lambda_{new} = \lambda_{child} + \lambda_{child}$ เมื่อ λ_{child} เป็นอาร์คของโหนดลูกหลานที่ต้องการจะรวมเข้าด้วยกัน) และทำการรวมกันของสองโหนดลูกหลานถัดไปหากยังมีโหนดลูกหลานเหลืออยู่มากกว่าหนึ่งตัว โดยพิจารณาจากค่าของ λ เช่นกัน ในขณะเดียวกันจะทำการสร้างสเปซโหนด(Space Node) ขึ้นมาเพื่อใช้เป็นโหนดเชื่อมต่อระหว่างโหนดรากหรือโหนดรากของต้นไม้ส่วนย่อย โดยสเปซโหนดนี้มีค่าของ μ เท่ากับศูนย์ จะใช้สัญลักษณ์ Sp เป็นสัญลักษณ์ของสเปซโหนด ดังนั้น $\mu(Sp) = 0$ รูปที่ 4.3 และ 4.4 เป็นตัวอย่างในการพิจารณาการแปลงโหนดใดๆ ที่มีจำนวนของโหนดลูกหลานมากกว่าสองตัวขึ้นไปให้เหลือเพียงสองโหนดลูกหลาน ตัวอย่างในรูปที่ 4.4 เป็นโหนดใดๆที่มีจำนวนของโหนดลูกหลานเท่ากับ 5 ดังนั้นจะต้องทำการแปลงจำนวนของโหนดลูกหลานสำหรับโหนด T1 ให้เหลือเพียงสองโหนดลูกหลาน จะสังเกตพบว่าในขั้นตอนแรกของการแปลงจำนวนของโหนดลูกหลานยังคงมีจำนวนของโหนดลูกหลานมากกว่าสองจำนวน ดังนั้นจะต้องทำการแปลงจำนวนของโหนดลูกหลานสำหรับโหนด T1 ให้เหลือเพียงสองโหนดลูกหลานให้ได้ ดังนั้นจำเป็นที่จะต้องสร้างสเปซโหนดขึ้นมาอีกชั้นหนึ่งเพื่อทำการลดจำนวนของโหนดลูกหลานให้เหลือเพียงสองโหนดลูกหลานให้ได้ รูปที่ 4.5 เป็นผลลัพธ์ที่ได้จากการแปลงต้นไม้กลับตัวชี้ในรูปที่ 4.2 ให้อยู่ในรูปของต้นไม้แบบทวิภาคกลับตัวชี้



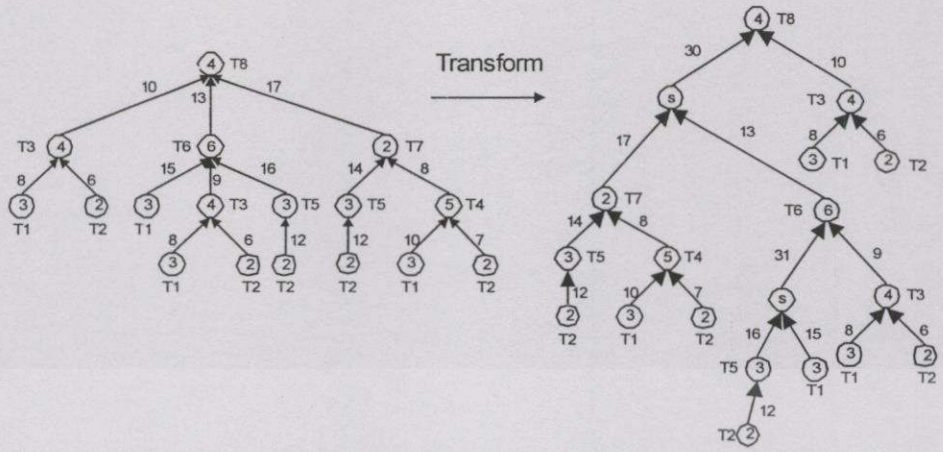
รูปที่ 4.3 แสดงตัวอย่างการแปลงต้นไม้กลับตัวชี้ให้อยู่ในรูปแบบของต้นไม้แบบทวิภาคกลับตัวชี้



รูปที่ 4.4 แสดงตัวอย่างการแปลงต้นไม้กลับตัวชี้ให้อยู่ในรูปแบบของต้นไม้แบบทวิภาคกลับตัวชี้

ในการแปลงต้นไม้กลับตัวชี้ให้อยู่ในรูปแบบของต้นไม้แบบทวิภาคกลับตัวชี้ จะใช้อัลกอริทึม MakeBinaryTree(v) เมื่อพารามิเตอร์ v สำหรับอัลกอริทึม MakeBinaryTree( ) คือราก ของต้นไม้ ดังแสดงต่อไปนี้

```
MakeBinaryTree(v)  // v is root of tree or root of subtree
{  IF( node v is not leaf node)
    {  for( each child of node v)   // u_i = each child of node v
        CCu_i = MakeBinaryTree(u_i);
        // CCu_i is a communication cost between node v to u_i
        While( child of v > 2 ) // if  v has exceed two childs
            (1) to combine each pair of child of node v that has largest CCu_i and
                to make CCsp_i = CCu_i + CCu_(i+1);
            (2) to create space node and link each pair of child of node v
                from (1) to space node then replace root subtree of each pair of
                child node with space node;
            (3) to remove connection between node v with child from (1);
            (4) to link node v to space node from (2) with new connection whose
                communication cost equals to CCsp_i ;
    }
    Return λ that is communication cost between node v and parent of node v
}
```



รูปที่ 4.5 ผลของการแปลง DAG ให้อยู่ในรูปของต้นไม้แบบทวิภาคกลับตัวชี้

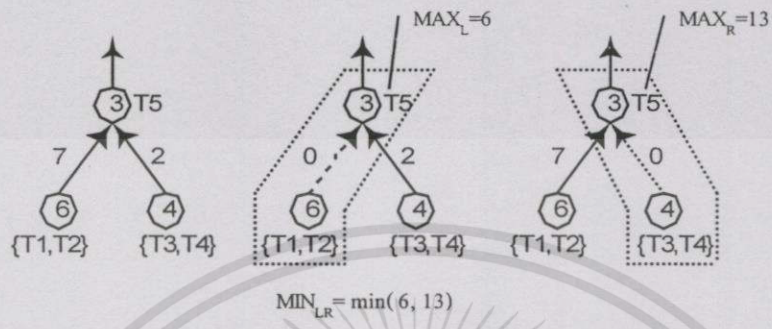
ขั้นตอนที่สาม ทำการสร้างชุดของกลุ่มงาน Φ การทำงานในขั้นตอนนี้เราจะใช้อัลกอริทึม AdaptiveTaskCluster( ) เป็นตัวดำเนินการ การทำงานของอัลกอริทึม AdaptiveTaskCluster( ) จะคล้ายกับอัลกอริทึม MakeBinaryTree( ) หลักการทำงานของอัลกอริทึม AdaptiveTaskCluster( ) จะใช้แบล็คแทรกกิ้งอัลกอริทึมมาประยุกต์ใช้ในการจัดกลุ่มโหนดหรืองานที่อยู่บนต้นไม้แบบทวิภาคกลับตัวชี้ ออกเป็นกลุ่มๆ โดยที่แต่ละกลุ่มจะมีจำนวนของโหนดหรืองานไม่เท่ากัน ลักษณะการทำงานของ AdaptiveTaskCluster( ) จะทำการสร้างกลุ่มงานโดยเริ่มต้นจากซอสโหนดกลับไปหาซิ้งค์โหนด หรือมีลักษณะเป็นแบบจากล่างขึ้นบน(Bottom Up) ในการสร้างกลุ่มงานของ AdaptiveTaskCluster( )จะเริ่มจากการนำเอาซอสโหนดมาใส่ไว้ในกลุ่มงานก่อน จากนั้นจึงค่อยหาโหนดที่มีความสัมพันธ์กับซอสโหนดมาใส่เพิ่ม

AdaptiveTaskCluster( )อัลกอริทึมจะพยายามลดช่วงเวลาที่หยุดนิ่งบนตัวประมวลผล หรืออาจจะกล่าวได้ว่าอัลกอริทึมดังกล่าวพยายามที่จะลดเวลาที่ใช้ในการประมวลผลของงานแบบขนานให้น้อยที่สุด หลักการทำงานจะใช้สมการที่ 4.1 เป็นหลักในการพิจารณาดังนี้

$$MIN_{LR} = \min(MAX_L, MAX_R) \qquad 4.1$$

ค่า MIN_{LR} เป็นค่าที่ใช้ในการหาค่าเวลาเริ่มต้น(e(v))ที่น้อยที่สุดสำหรับโหนดหรืองานตัวใหม่ที่จะถูกนำเข้ามารวมในกลุ่มงาน ซึ่งงานหรือโหนดตัวใหม่อาจจะเป็นโหนดรากหรือโหนดรากของต้นไม้ส่วนย่อย นอกจากนี้ค่าของ MAX_L และ MAX_R คือค่าเวลาที่เสร็จสิ้นการทำงานที่มากที่สุดของต้นไม้ส่วนย่อยด้านซ้ายและด้านขวา หรือกลุ่มของโหนดลูกหลานด้านซ้ายและขวา

รูปที่ 4.6 เป็นการแสดงตัวอย่างในการสร้างกลุ่มงานโดยอาศัยสมการที่ 4.1 เป็นหลักในการทำงาน จากรูปจะเห็นได้ว่าค่าของ MIN_{LR} เป็นค่าเวลาเริ่มต้น(e(v))ที่น้อยที่สุดที่เป็นไปได้สำหรับงานหรือโหนด T5 หลักในการพิจารณาเป็นไปได้ 3 กรณีด้วยกัน



รูปที่ 4.6 ตัวอย่างการสร้างกลุ่มงานสำหรับโหนด v ใดๆ

กรณีแรกหากนำเอางานหรือโหนด T5 เข้าไปรวมกลุ่มงานด้านซ้ายหรือขวา แล้วกลุ่มงานด้านใดด้านหนึ่งสามารถให้เวลาเริ่มต้นที่น้อยที่สุดเท่าที่เป็นไปได้สำหรับงานหรือโหนดตัวใหม่ ก็จะนำเอางานหรือโหนด T5 เข้าไปรวมในกลุ่มงานนั้น จากรูปแสดงให้เห็นว่าหากนำเอางาน T5 เข้าไปรวมในกลุ่มงาน{T1, T2} แล้วค่าเวลาเริ่มต้นเท่าที่เป็นไปได้สำหรับงาน T5 หรือ $MAX_L = 6$ ในขณะเดียวกันหากนำเอางาน T5 เข้าไปรวมในกลุ่มงาน {T3, T4} แล้วค่าเวลาเริ่มต้นเท่าที่เป็นไปได้สำหรับงาน T5 หรือ $MAX_R = 13$ ดังนั้น $MIN_{LR} = \min(MAX_L , MAX_R) = 6$ ซึ่งเป็นค่าเวลาเริ่มต้น(e(T5))ที่น้อยที่สุดเท่าที่เป็นไปได้สำหรับโหนด T5

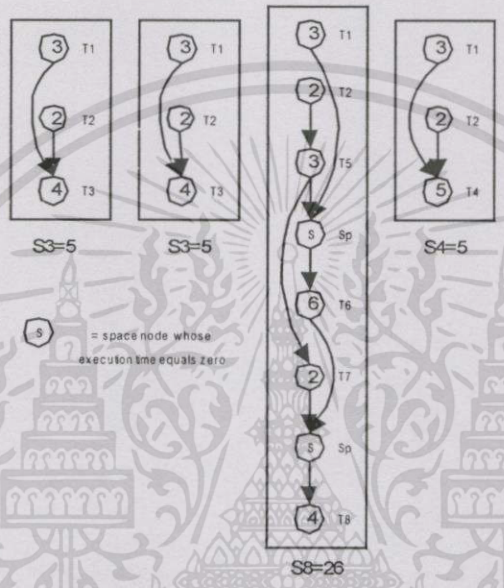
กรณีที่สองหากค่าของ MAX_L และ MAX_R เท่ากันเราจะนำเอางานไปรวมในกลุ่มงานด้านใดด้านหนึ่งก็ได้

กรณีที่สามหากนำเอากลุ่มงานทั้งสองด้านมาทำการรวมกัน หรือยูเนี่ยน(Union)กันแล้วทำให้ค่าเวลาเริ่มต้นสำหรับงานหรือโหนด T5 น้อยกว่าในกรณีแรก เราจะทำการยูเนี่ยนกลุ่มงานทั้งสองเข้าด้วยกันและทำการรวมงาน T5 นี้ด้วย ดังนั้น {T1, T2, T3, T4} = {T1, T2} $\cup$ {T3, T4} จะให้ค่าเวลาเริ่มต้นสำหรับงาน T5 เท่ากับ 10 ซึ่งยังคงมีค่ามากกว่าค่าของ MIN_{LR} ที่เกิดขึ้นในกรณีแรก ดังนั้นกรณีที่สามนี้จึงไม่ได้รับการพิจารณา

จากรูปจะเห็นได้ว่าการหาค่า MIN_{LR} ของโหนดหรืองาน T5 ซึ่งเป็นโหนดตัวใหม่ที่จะถูกนำเข้ามารวมในกลุ่มงาน ในตัวอย่างแสดงให้เห็นว่าโหนด T5 จะถูกนำมาเข้ารวมในกลุ่มงาน {T1, T2}ทำให้เป็นสมาชิกตัวใหม่ของกลุ่มงานดังกล่าว ดังนั้นผลลัพธ์จะได้กลุ่มงานที่ขยายออกเป็น{T1, T2, T5} ในขณะเดียวกันกลุ่มงาน{T3, T4} จะทำงานในลักษณะขนานกับกลุ่มงาน{T1, T2, T5}

จากสมการที่ 4.1 แสดงถึง มิน-แมก ไครทีเรียน(Min-Max criterion) สำหรับการสร้างกลุ่มงานใดๆ ดังที่ได้เคยกล่าวมาแล้วในบทที่ 3 และ 4

อัลกอริทึม AdaptiveTaskCluster( ) ดังแสดงข้างล่างดังต่อไปนี้ ใช้เป็นเครื่องมือในการสร้างชุดของกลุ่มงาน รูปที่ 4.7 เป็นผลลัพธ์ที่ได้จากการใช้ AdaptiveTaskCluster( ) แสดงถึงชุดของกลุ่มงานและได้ชุดของกลุ่มงานแบบที่มีงานซ้ำซ้อนเกิดขึ้น โดยการใช้ต้นไม้แบบทวิภาคกลับตัวชี้ในรูปที่ 4.5 เป็นอินพุทของอัลกอริทึมนี้



รูปที่ 4.7 ชุดของกลุ่มงานที่ได้จากอัลกอริทึม AdaptiveTaskCluster

L = left sub-tree, R = right sub-tree

μ_L = computation cost of left child node of node v

μ_R = computation cost of right child node of node v

μ_v = computation cost of node v

λ_L = communication cost between left child node and node v

λ_R = communication cost between right child node and node v

$Start_v$ = Started time of node v

C_v = Task clustering at node v

CR = Task clustering at right child sub-tree of node v

CL = Task clustering at left child sub-tree of node v

$Compl_v$ = Completion time of node v

AdaptiveTaskCluster(v, λ, start, compl, C)

{ IF(node v is not leaf node)

{ AdaptiveTaskCluster(v_L , λ_L , $Start_L$, $Compl_L$, CL);

AdaptiveTaskCluster(v_R , λ_R , $Start_R$, $Compl_L$, CR);

$MAX_L = max(compl_L + \lambda_L, compl_R)$;

$MAX_R = max(compl_L, compl_R + \lambda_R)$;

$MAX_v = compl_{(CR \cap CL)} + compl_{\sim(CR \cap CL)}$;

$MIN_{LR} = min(MAX_L, MAX_R)$;

IF(node v is space node) //**** case 1

{ $C_v = C_v \cup (CR \cap CL) \cup \sim(CR \cap CL)$;

$Start_v = max(compl_L, compl_R) + \exists(\mu(v))$ when $v \notin$ (either L or R)

but $v \in$ (either L or R) that depends on $max(compl_L, compl_R)$

such as $v \notin L$ but $v \in R$;

} else IF($MIN_{LR} < MAX_v$) // *** case 2

{ $C_v = C_v \cup$ {task clustering left or right child of node v that depends

on MIN_{LR} }

$Start_v = MIN_{LR}$;

$\Phi = \Phi \cup$ {task clustering that depends on condition of MIN_{LR} };

} else IF($MIN_{LR} >= MAX_v$) //*** case 3

{ IF(($CL \cap CR$) = ϕ) // ϕ means empty set *** case 3.1

$C_v = C_v \cup CL \cup CR$;

$Start_v = compl_R + compl_L$;

} else IF (($\forall L_i \in CR$) or $\forall R_i \in CL$)) //when $L_i \in CL$ and

$R_i \in CL$ ***case 3.2

{ $C_v = C_v \cup$ {CR or CL that depends on condition};

$Start_v = compl_{CR}$ or $compl_{CL}$;// depends on condition

} else //*** case 3.3

{ $C_v = C_v \cup$ {task clustering left or right child of node v

that depends on MIN_{LR} };

$Start_v = MIN_{LR}$;

$\Phi = \Phi \cup$ {task clustering that depends on condition of MIN_{LR} };

} $compl_v = Start_v + \mu_v$;

}

}else { $\lambda = \lambda_v$;

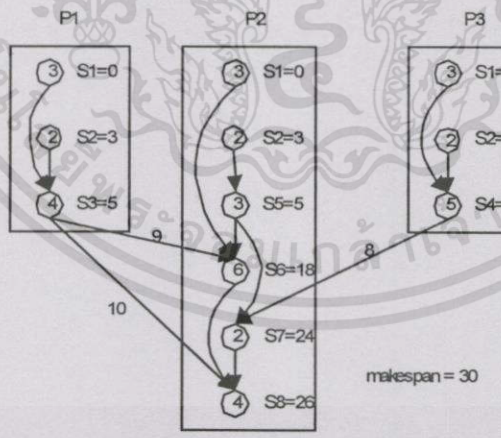
Start = $Start_v$;

Compl = Start + μ_v;

$C = C \cup C_v$; }

} // End

ขั้นตอนที่สี่ จัดลำดับการทำงานให้กับชุดของกลุ่มงาน จากขั้นตอนที่สามเมื่อได้ชุดของกลุ่มงาน Φ จากนั้นจะนำเอาชุดของกลุ่มงานที่ได้มาทำการจัดลำดับการทำงาน โดยกลุ่มงานที่ได้จากขั้นตอนที่สามประกอบได้ด้วยกลุ่มงานแบบที่มีงานซ้ำซ้อนเกิดขึ้นดังนี้ Φ = {{T1, T2, T3}, {T1, T2, T3}, {T1, T2, T5, T6, T7, T8}, {T1, T2, T4}} ซึ่งเป็นชุดของกลุ่มงานในรูปที่ 4.7 ในการจัดลำดับการทำงานให้กับชุดของกลุ่มงานเราจะใช้อัลกอริทึม ScheduleOfTask(C, Φ) ดังแสดงอยู่ข้างล่าง โดยอินพุทของอัลกอริทึม ScheduleOfTask( ) คือชุดของกลุ่มงาน Φ และกลุ่มงาน C ที่มีซิ้งค์โหนดประกอบอยู่ด้วย ดังนั้นกลุ่มงาน C ที่มีซิ้งค์โหนดประกอบอยู่สำหรับชุดของกลุ่มงาน Φ = {{T1, T2, T3}, {T1, T2, T3}, {T1, T2, T5, T6, T7, T8}, {T1, T2, T4}} คือ C = { T1, T2, T5, T6, T7, T8} เป็นอินพุท



รูปที่ 4.8 ผลลัพธ์จากการจัดลำดับงานให้กับชุดของกลุ่มงาน

รูปที่ 4.8 เป็นการแสดงผลลัพธ์จากการจัดลำดับงานให้กับชุดของกลุ่มงานในรูปที่ 4.7 จากรูปแสดงให้เห็นว่าจะได้กลุ่มงานจำนวนสามกลุ่มงาน วางอยู่บนตัวประมวลผลจำนวนสามตัว หรือหนึ่งกลุ่มงานต่อหนึ่งตัวประมวลสำหรับโปรแกรมแบบขนานของ DAG ในรูปที่ 4.1 ดังนั้น Φ(DAG) = {{T1, T2, T3}, {T1, T2, T4}, {T1, T2, T5, T6, T7, T8}} จากรูปที่ 4.8 แสดงให้เห็นว่าเอทีซีเอสอัลกอริทึมได้ทำการลดโอเวอร์เฮดระหว่างงานต่างๆของโปรแกรมแบบขนานได้ นอกเหนือจากนี้ได้แสดงให้เห็นว่าค่าของ makespan ของชุดของกลุ่มงานหลังจากที่ได้จัดลำดับการทำงานให้แล้ว จะมีค่าเท่ากับ 30 ซึ่งน้อยกว่าค่าของ makespan ของ DAG ซึ่งมีค่าเท่ากับ 56 ดังนั้นสรุปได้ว่า เอทีซีเอสอัลกอริทึมสามารถลดโอเวอร์เฮดอันเนื่องมาจากการติดต่อสื่อสาร ช่วยลดช่วงเวลาที่หยุดนิ่งของตัวประมวลผล และในขณะเดียวกันเป็นการเพิ่มขนาดเม็ดเนื้องานที่ตัวประมวลผลให้มากขึ้นหรือค่าของ Granularity $g(G) \geq 1$ ทำให้ลักษณะเม็ดเนื้องานที่ได้เป็นเม็ดเนื้องานแบบหยาบ

ScheduleOfTask(C, Φ) // C is task clustering
{ Φ(DAG) = ϕ; // Φ(DAG) is set of task clustering of a DAG
Φ(DAG) = Φ(DAG) $\cup$ C;
Sink node of C is marked;
while(There are no more marked node) // $c_i \in C$ when i = 1,2,3,...,n
1) to pick marked node;
2) Sink node of C is unmarked;
3) If C has an arc connected to outside cluster ($\lambda_{u,w}$ when w = c_i and
u $\notin$ C but u $\in$ other cluster, w $\in$ C)
Then union cluster of u to Φ(DAG) // u must be sink node of outside clusters C
and All Sink nodes u are marked;
}

4.3 การวิเคราะห์ค่าความสลับซับซ้อนของเวลาของเอทีซีเอสอัลกอริทึม

ในการวิเคราะห์หาค่าความสลับซับซ้อนของเวลาของเอทีซีเอสอัลกอริทึมจะประกอบไปด้วย 4 ขั้นตอนด้วยกันดังนี้

1. การหาค่าความสลับซับซ้อนของเวลาในการแปลง DAG ให้อยู่ในรูปของต้นไม้กลับตัวชี้ ภายในขั้นตอนนี้จะต้องทำการพิจารณาโหนดทุกๆโหนดที่อยู่บน

DAG ดังนั้นค่าความสลับซับซ้อนเชองเวลาในขั้นตอนนี้เท่ากับ O(|V| + |E|) ทั้งนี้เนื่องมาจาก DAG จะถูกจัดเก็บอยู่ในรูปแบบของ โครงสร้างของมัลติลิ้งค์เกต(Multilinked Structure)[27] ของกราฟงาน ดังนั้นในการเข้าถึงงานหรือโหนดต่างๆ เท่ากับ O(V) และอาร์คต่างๆเท่ากับ O(E) ที่อยู่บนกราฟงาน เพื่อนำมาสร้างเป็นต้นไม้กลับตัวชี้

2. ในการแปลงต้นไม้กลับตัวชี้ให้อยู่ในรูปของต้นไม้แบบทวิภาคกลับตัวชี้ เนื่องจากอาร์คต่างๆของโหนดลูกหลานจะถูกจัดเก็บเรียงอย่างเป็นลำดับอยู่ในรูปของต้นไม้แบบทวิภาคอยู่แล้วตั้งแต่เริ่มสร้างต้นไม้กลับตัวชี้ ดังนั้นการเข้าถึงอาร์คของโหนดลูกหลาน(CCu_i)จะเท่า O(lg m) เมื่อ m เป็นจำนวนของโหนดลูกหลานที่ไม่ใช่บัพใบ โดยในที่นี้จะสมมติว่าทุกๆโหนดบนต้นไม้จะประกอบไปด้วยโหนดลูกหลานจำนวน m ตัว ในกระบวนการกระทำซ้ำภายในลูป จะกำหนดให้ c (c << m)เป็นค่าคงที่ในการกระทำซ้ำ ในขั้นตอนที่ 4 เป็นการเชื่อมต่อสเปซโหนดให้เข้ากับโหนด v ใดๆหรือโหนดรากหรือโหนดรากของต้นไม้ส่วนย่อยจะต้องใช้เวลาในการกระทำเท่ากับ O(lg m) ส่วนในขั้นตอนที่ 1, 2, และ 3 ในลูปต้องใช้เวลาเท่ากับ O(1) ดังนั้นในขั้นตอนของการกระทำซ้ำๆเท่ากับ T(c(lg m + 1 + 1 + 1)) = O(lg m) ความเป็นไปได้ที่จำนวนครั้งในการกระทำกับโหนดลูกหลานทั้งหมดจำนวน V ครั้ง ทำให้เวลาที่ต้องใช้ไปเท่ากับ O(V lg m) หากในกรณีที่เลวร้าย m = V ดังนั้นค่าความสลับซับซ้อนของเวลาของอัลกอริทึมในส่วนนี้เท่ากับ O(V lg V)
3. ในการรวมกลุ่มงานสำหรับงานหรือโหนดต่างๆที่อยู่บนต้นไม้แบบทวิภาคกลับตัวชี้จะมีค่าความสลับซับซ้อนของเวลาที่ใช้ในขั้นตอนนี้เท่ากับ $O(|V^2|)$ เพราะเหตุว่าแต่ละโหนด v_i ใดๆที่อยู่บน DAG G เมื่อ $\forall v_i \in V$ โหนดเหล่านี้สามารถที่จะถูกเข้าถึง V -1 ครั้งด้วยกัน และกำหนดให้ S เป็นจำนวนของสเปซโหนดที่เกิดขึ้นบนต้นไม้ ถ้าเราสมมติว่า ขนาดของ S เท่ากับ V ดังนั้นความเป็นไปได้ที่ค่าความสลับซับซ้อนของเวลาที่ใช้ไปเท่ากับ $|V|(|V| - 1 + S) = O(|V^2|)$
4. ค่าความสลับซับซ้อนของเวลาที่ใช้ไปในอัลกอริทึม ScheduleOfTask( )จะเท่ากับ O(|v| + |e|) เมื่อ v คือจำนวนของงานที่อยู่ภายในชุดของกลุ่มงาน และ e เป็นจำนวนของอาร์คที่ใช้ในการเชื่อมต่อไปยังโหนดที่อยู่ภายนอกกลุ่มงาน แต่ยังคงอยู่ภายในชุดของกลุ่มงาน ค่าความสลับซับซ้อนของเวลาของอัลกอริทึมเท่ากับ O(|V| + |E|)

จากทั้ง 4 ขั้นตอนดังนั้นค่าความสลับซับซ้อนของเวลาของเอทีซีเอสอัลกอริทึม เท่ากับ $O(|V| + |E|) + O(V \lg V) + O(|V^2|) + O(|V| + |E|) = O(|V^2|)$

4.4 การวัดประสิทธิภาพของเอทีซีเอสอัลกอริทึม

วิธีการอย่างหนึ่งที่ใช้ในการวัดส่วนที่ดี(goodness)[33]ของค่าใช้จ่ายที่เกิดขึ้นของอัลกอริทึมแบบขนานคือการคำนวณหาประสิทธิภาพของอัลกอริทึม กำหนดให้ $time_1$ เป็นเวลาที่ใช้ไปเร็วที่สุดในการประมวลผลของงาน และ p_1 เป็นจำนวนของตัวประมวลผลที่ต้องใช้ในการทำงานสำหรับอัลกอริทึมแบบขนานที่ถูกเปรียบเทียบ คล้ายกันกำหนดให้ $time_2$ เป็นเวลาที่ใช้เร็วที่สุดในการประมวลผลของงาน และ p_2 เป็นจำนวนของตัวประมวลผลที่ต้องใช้ในการทำงานสำหรับอัลกอริทึมแบบขนานที่ต้องการจะเปรียบเทียบ โดยที่ทั้งสองอัลกอริทึมนี้จะต้องถูกกำหนดให้ทำการแก้ปัญหาเดียวกันหรือโจทย์ข้อเดียวกัน ดังนั้นในการวัดประสิทธิภาพของอัลกอริทึมที่หนึ่ง เปรียบเทียบกับอัลกอริทึมที่สองถูกแสดงด้วย

$$E(p_1, p_2) = \frac{(p_1) * (time_1)}{(p_2) * (time_2)} \qquad 4.2$$

จากสมการ 4.2 จะเห็นได้ว่าจำนวนของตัวประมวลผลที่ใช้ไปในแต่ละวิธีการของอัลกอริทึมค่าดังกล่าวเป็นค่าใช้จ่ายที่เกิดขึ้นในการทำงาน ดังนั้นในการวัดประสิทธิภาพของอัลกอริทึมที่ถูกเปรียบเทียบจะมีค่าอย่างมากที่สุดเท่ากับ 1 เหตุการณ์ที่เกิดขึ้นเท่าที่เป็นได้ มีสามกรณีด้วยกันสำหรับค่าของ $E(p_1, p_2)$ ดังนี้ :

1. ถ้า $E(p_1, p_2) < 1$ แสดงว่าวิธีการของอัลกอริทึมที่ถูกเปรียบเทียบ จะให้ประสิทธิภาพที่สูงกว่าวิธีการของอัลกอริทึมที่ต้องการจะเปรียบเทียบ ทั้งนี้ต้องขึ้นอยู่กับปัจจัยของตัวแปรของสมการด้วย ถ้าหากเวลาสำหรับ $time_1$ เท่ากับ $time_2$ แต่จำนวนของ p_1 น้อยกว่า p_2 แสดงว่าวิธีการของอัลกอริทึมที่ถูกเปรียบเทียบให้ค่าใช้จ่ายที่เหมาะสมมากกว่าวิธีการของอัลกอริทึมที่ต้องการจะเปรียบเทียบ ในขณะเดียวกันหากจำนวนของ p_1 เท่ากับ p_2 แสดงวิธีการของอัลกอริทึมที่ถูกเปรียบเทียบให้เวลาที่เหมาะสมมากกว่าวิธีการของอัลกอริทึมที่ต้องการจะเปรียบเทียบ
2. ถ้า $E(p_1, p_2) = 1$ แสดงว่าวิธีการของอัลกอริทึมทั้งสองให้ประสิทธิภาพที่เท่ากัน แต่ทั้งนี้จะต้องทำการพิจารณาปัจจัยของตัวแปรที่อยู่ในสมการเช่นเดียวกับในข้อแรก กรณีแรกถ้าหาก p_1 เท่ากับ p_2 และ $time_1$ เท่ากับ $time_2$ แสดงว่าวิธีการของ

อัลกอริทึมทั้งสองให้ผลลัพธ์ที่เหมาะสมเช่นเดียวกัน กรณีที่สองหาก p_1 น้อยกว่า p_2 แสดงว่าวิธีการของอัลกอริทึมที่ถูกเปรียบเทียบจะให้ค่าใช้จ่ายของการใช้ตัวประมวลผลที่เหมาะสมมากกว่าวิธีการของอัลกอริทึมที่ต้องการจะเปรียบเทียบ ในขณะเดียววิธีการของอัลกอริทึมที่ต้องการจะเปรียบเทียบกลับให้เวลาที่เหมาะสมกว่าวิธีการของอัลกอริทึมที่ถูกเปรียบเทียบ และในกรณีที่สาม หาก p_1 มากกว่า p_2 แสดงว่าวิธีการของอัลกอริทึมที่ต้องการเปรียบเทียบจะให้ค่าใช้จ่ายของการใช้ตัวประมวลผลที่เหมาะสมมากกว่าวิธีการของอัลกอริทึมที่ถูกเปรียบเทียบ และในขณะเดียวกันวิธีการของอัลกอริทึมที่ถูกเปรียบเทียบจะให้เวลาที่เหมาะสมมากกว่าวิธีการของอัลกอริทึมที่ต้องการเปรียบเทียบ

3. ถ้า $E(p_1, p_2) > 1$ แสดงว่าวิธีการของอัลกอริทึมที่ต้องการเปรียบเทียบให้ประสิทธิภาพมากกว่าวิธีการของอัลกอริทึมที่ถูกเปรียบเทียบ ทั้งนี้จะต้องทำการพิจารณาปัจจัยของแปรด้วย หลักในการพิจารณาเช่นเดียวกันกับในข้อแรก

จากค่าที่เป็นไปได้ของ $E(p_1, p_2)$ ทั้งสามค่าจะพบว่าในกรณีที่ $E(p_1,p_2) = 1$ ทำให้เกิดการตัดสินในการเลือกใช้อัลกอริทึมใดอัลกอริทึมหนึ่งได้หลายกรณีด้วยกัน ทั้งนี้ขึ้นอยู่กับความเหมาะสมหรือขึ้นอยู่กับการใช้ทรัพยากรให้เกิดประโยชน์

4.5 การวัดความเร็วในการทำงานของเอทีซีเอสอัลกอริทึม

เมื่อใดที่มีความต้องการที่จะประเมินอัลกอริทึมการทำงานแบบขนาน บ่อยครั้งที่เรามักจะให้ความสนใจในเรื่องการที่ต้องการจะทราบว่า ผลที่ได้รับจากการกระทำเพื่อทำการแก้ไขปัญหาเดียวกัน ด้วยวิธีการของอัลกอรทึมแบบขนานที่เหนือกว่าวิธีการของอัลกอริทึมแบบเรียงตามลำดับ(Sequential)มากน้อยเท่าใดสามารถทำได้ด้วยวิธีการวัดความเร็ว(Speed Up)[33] การวัดความเร็วเป็นวิธีการหนึ่งที่ใช้วัดความสัมพันธ์ ของประโยชน์ที่ได้รับจากการแก้ไขปัญหาด้วยวิธีการของอัลกอริทึมแบบขนานที่มีต่อการแก้ไขปัญหาเดียวกันด้วยวิธีการของอัลกอริทึมแบบเรียงตามลำดับ การวัดความเร็วจะถูกกำหนดด้วยอัตราส่วนระหว่างเวลาที่ใช้ในการประมวลผลที่อยู่บนตัวประมวลผลตัวเดียวด้วยอัลกอริทึมแบบเรียงตามลำดับ ต่อเวลาที่ใช้ในการประมวลผลที่อยู่บนตัวประมวลผลมากกว่าหนึ่งตัวด้วยอัลกอริทึมแบบขนาน(โดยตัวประมวลผลที่ใช้ในจัดลำดับงานเป็นแบบขนานและแบบเรียงตามลำดับเหล่านี้ จะต้องอยู่ภายใต้ข้อกำหนดที่ว่า ตัวประมวลผลเหล่านี้เหมือนกันทั้งหมด กล่าวคือ ความเร็วในการประมวลผลของตัวประมวลผล) ที่อยู่ภายใต้โจทย์หรือปัญหาเดียวกัน เราจะแสดงสัญลักษณ์ของการวัดความเร็วด้วย $S(1, p)$ กำหนดให้ 1 หมายถึงจำนวน

ตัวประมวลผลหนึ่งตัวที่ใช้ในการจัดลำดับงานด้วยอัลกอริทึมแบบเรียงตามลำดับ และ p เป็นจำนวนของตัวประมวลผลที่ใช้มากกว่าหนึ่งตัวในการจัดลำดับงานด้วยอัลกอริทึมแบบขนาน กำหนดให้ t_1 แสดงถึงเวลาที่ใช้ในประมวลผลสำหรับการแก้ไขปัญหาที่อยู่ภายใต้การทำงานของตัวประมวลผลเดียวหรือการทำงานของอัลกอริทึมในการจัดลำดับงานแบบเรียงตามลำดับ ส่วน t_p แสดงถึงเวลาที่ใช้ในการประมวลผลสำหรับการแก้ไขปัญหาที่อยู่ภายใต้การทำงานของตัวประมวลผลมากกว่าหนึ่งตัวหรือการทำงานของอัลกอริทึมแบบขนาน ดังแสดงอยู่ในสมการ 4.3

$$S(1, p) = \frac{t_1}{t_p} \qquad 4.3$$

สำหรับปัญหาใดๆ จะมีอัลกอริทึมที่ใช้ในการจัดลำดับงานในลักษณะเป็นแบบเรียงตามลำดับมากมายมากกว่าหนึ่งวิธีการหรือมากกว่าหนึ่งอัลกอริทึม ที่สามารถนำมาใช้ในการแก้ไขปัญหาได้ดี แต่ไม่ได้หมายความว่าอัลกอริทึมแบบเรียงตามลำดับทั้งหมดจะสามารถทำการแก้ไขปัญหาได้ดีหรือเหมาะสมเทียบเท่าอัลกอริทึมที่ใช้ในการจัดลำดับงานแบบขนานได้ดีเสมอไป ดังนั้นเราต้องการที่จะวัดความเร็วที่ใช้ในการประมวลผลระหว่างวิธีการในการจัดลำดับงานของอัลกอริทึมแบบขนานที่ทำงานอยู่บนตัวประมวลผลมากกว่าหนึ่งตัว กับวิธีการในการจัดลำดับงานของอัลกอริทึมแบบเรียงตามลำดับที่ทำงานอยู่บนตัวประมวลผลเพียงตัวเดียว ที่อยู่ภายใต้การแก้ไขปัญหาหรือโจทย์ข้อเดียวกัน

จากสมการที่ 4.3 แสดงให้เห็นถึงการเปรียบเทียบระหว่าง t_1เป็นเวลาที่ใช้ในการประมวลผลของอัลกอริทึมแบบเรียงตามลำดับ ซึ่งเป็นกรณีที่เลวร้าย(worse case)ของเวลาที่เร็วที่สุดสำหรับอัลกอริทึมแบบเรียงตามลำดับโดยการใช้ตัวประมวลผลเพียงตัวเดียว และ t_p เป็นเวลาที่ใช้ในการประมวลผลของอัลกอริทึมแบบขนาน ซึ่งเป็นกรณีที่เลวร้ายของเวลาที่เร็วที่สุดสำหรับอัลกอริทึมแบบขนาน โดยใช้จำนวนของตัวประมวลผลจำนวน p ตัว

บทที่ 5

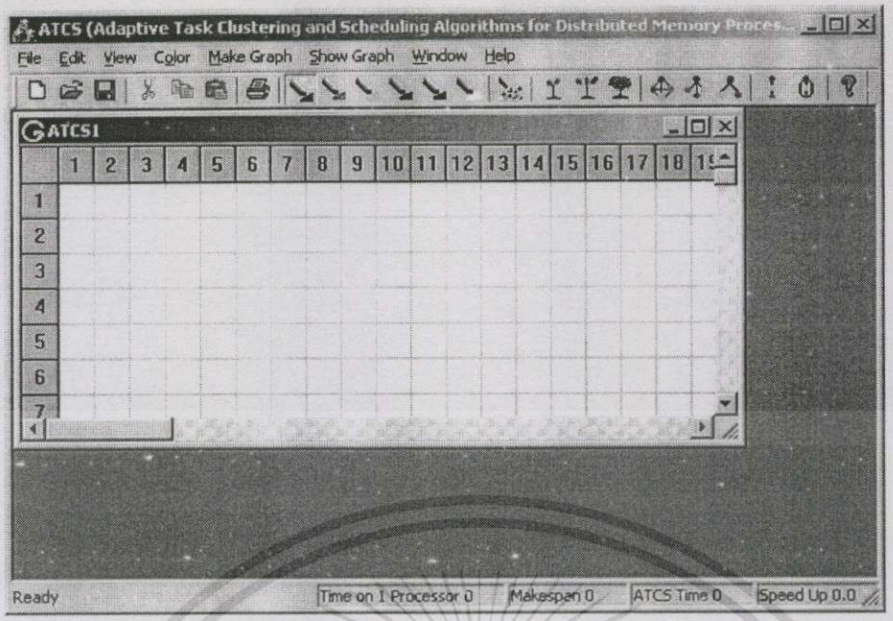
การจำลองและผลการทดลองของเอทีซีเอสอัลกอริทึม

ในบทนี้จะเป็นการแสดงผลการทดลองด้วย DAG ที่มีลักษณะต่างๆกัน โดย DAG ที่ใช้มีทั้งได้มาจากบทความทางวิชาการ[23][32] และได้จากการสุ่มโดยใช้โปรแกรมเป็นตัวสร้างขึ้นมา ดังที่ได้เคยกล่าวมาแล้วว่า DAG ที่ได้จากการสุ่มนั้น ค่าทุกค่าที่อยู่บน DAG ได้มาจากการสุ่มทั้งสิ้น ไม่ว่าจะเป็นค่าใช้จ่ายในการประมวลผลของงานที่ตัวประมวลผล ค่าใช้จ่ายในการติดต่อสื่อสารระหว่างงานหรือตัวประมวลผล จำนวนของโหนดที่อยู่บน DAG จำนวนของอาร์คที่ชี้ออกจากโหนดแต่ละโหนด และจำนวนระดับของ DAG ดังนั้น DAG ที่ได้จากการสุ่มอาจจะเป็น DAG ที่มีลักษณะไม่ได้เป็นจริง หรือลักษณะของ DAG ที่ได้อาจจะเกินความเป็นจริงก็ได้ เหตุผลที่ต้องมีการทดลองกับ DAG ที่ได้จากการสุ่ม เพราะว่าเราต้องการพิสูจน์วิธีการของเอทีซีเอสอัลกอริทึมด้วยการทำเจนเนอริกเคส(Generic case) นอกเหนือไปจาก DAG ที่ได้จากหนังสือและบทความทางวิชาการต่างๆ

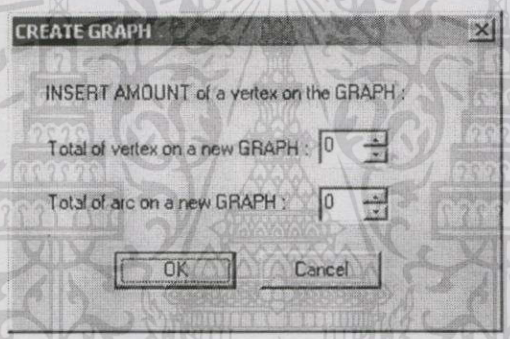
5.1 วิธีการจำลองในการสร้าง DAG

ในการจำลอง DAG แบบต่างๆดังที่ได้กล่าวไว้แล้วว่ามีสองแบบ แบบแรกเป็น DAG ที่สร้างขึ้นมา โดยอาศัยรูปแบบต่างๆจากตัวอย่างในหนังสือและบทความทางวิชาการ ส่วนแบบที่สองเป็นแบบที่ได้จากการสุ่ม โดย DAG ทั้งสองแบบจะใช้โปรแกรมเอทีซีเอส(Program ATCS)ในการสร้างดังแสดงในรูปที่ 5.1 เป็นการแสดงลักษณะโดยทั่วๆไปของโปรแกรมเอทีซีเอส ส่วนซอสโค้ดของโปรแกรมเอทีซีเอส แสดงอยู่ในส่วนของภาคผนวก ก.

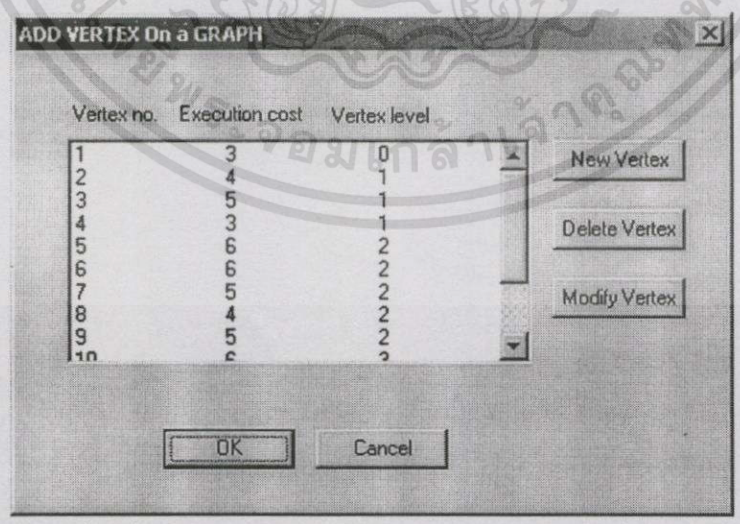
DAG ที่สร้างขึ้นมานั้นผู้ใช้จะต้องทำการป้อนข้อมูลเกี่ยวกับ จำนวนของโหนดและจำนวนของอาร์คลงในไดอะล็อกบ็อก(Dialog Box) “CREATE GRAPH” ดังแสดงอยู่ในรูปที่ 5.2 จากนั้นผู้ใช้จะต้องใส่ค่าใช้จ่ายในการประมวลผลของงานหรือโหนดของแต่ละงาน และระดับของโหนดที่อยู่บน DAG ดังแสดงอยู่ในรูปที่ 5.3 ไดอะล็อกบ็อก “ADD VERTEX On a GRAPH” ในการสร้าง DAG ใดๆจะต้องมีความสัมพันธ์ระหว่างโหนดต้นทางและโหนดปลายทาง และค่าใช้จ่ายในการติดต่อสื่อสารโดยจะใช้ไดอะล็อกบ็อก “JOINT VERTEX” ดังแสดงอยู่ในรูปที่ 5.4 เมื่อผู้ใช้ป้อนข้อมูลต่างๆจนครบแล้ว จากนั้นโปรแกรมเอทีซีเอสจะสร้าง DAG ให้และทำการแสดง DAG ขึ้นมาเมื่อผู้ใช้กดปุ่มทูลบาร์(Tool Bar) “Graphs” ดังที่จะแสดงอยู่ในส่วนที่ 5.5



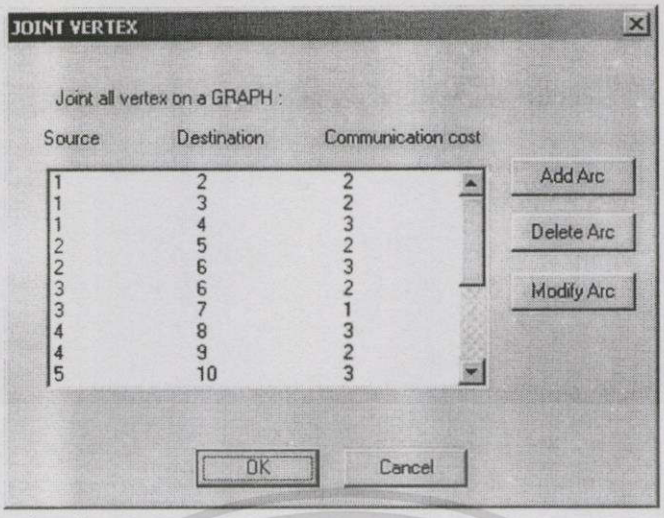
รูปที่ 5.1 ลักษณะโดยทั่วๆไปของโปรแกรมเอทีซีเอส



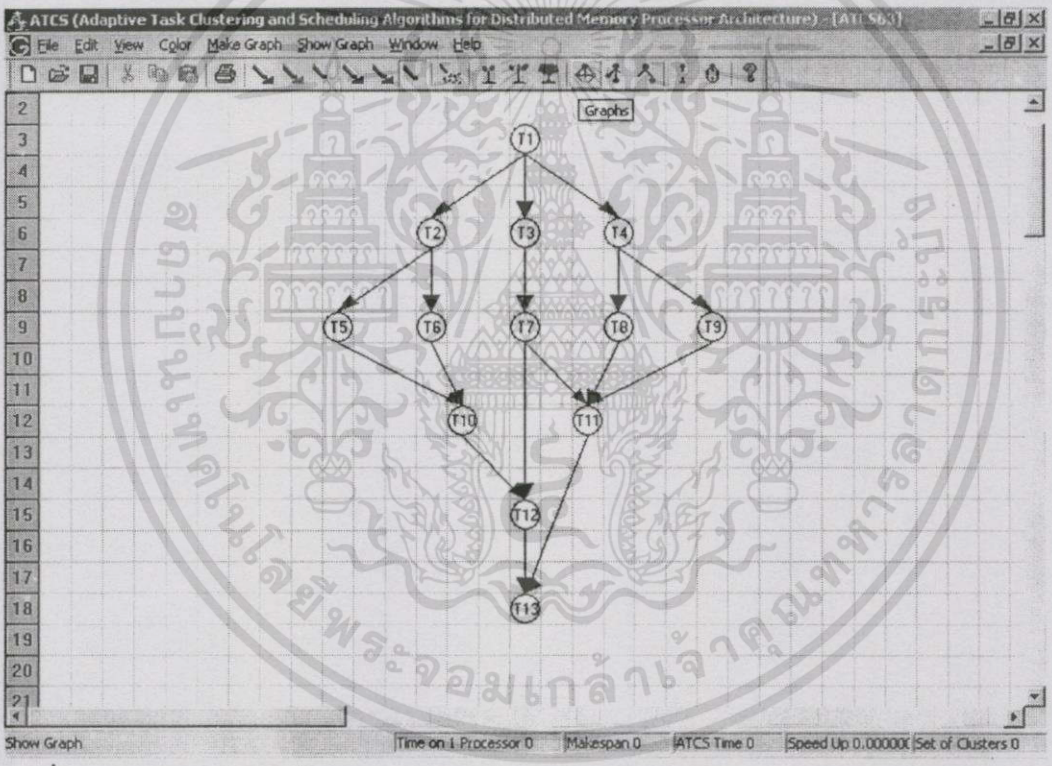
รูปที่ 5.2 แสดงไดอะล็อกบ็อก "CREATE GRAPH"



รูปที่ 5.3 แสดงไดอะล็อกบ็อก "ADD VERTEX On a GRAPH"



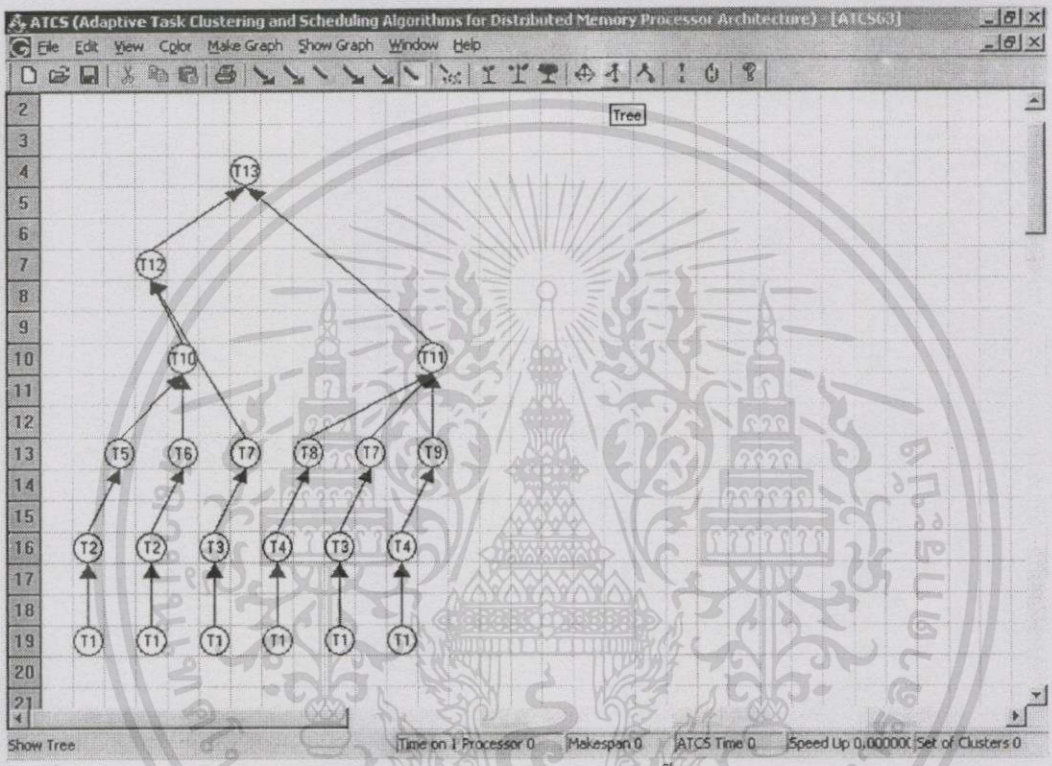
รูปที่ 5.4 แสดงไดอะล็อกบ็อก "JOINT VERTEX"



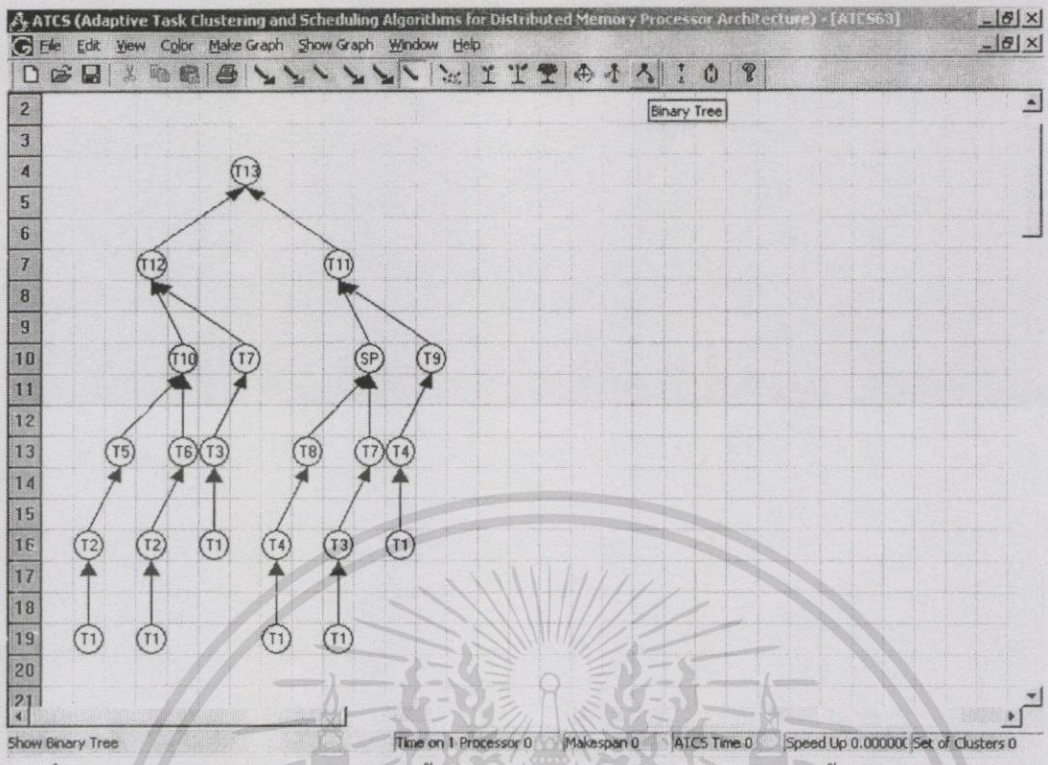
รูปที่ 5.5 ตัวอย่างจากการสร้าง DAG

รูปที่ 5.5 เป็นตัวอย่างของการสร้าง DAG เมื่อผู้ใช้ได้ทำการใส่ข้อมูลต่างๆจนครบแล้ว และทำการกดปุ่มทูลบาร์ "Graphs" ขั้นตอนถัดไปถ้าหากผู้ใช้กดปุ่มทูลบาร์ "Tree" โปรแกรมเอทีซีเอสจะทำการแปลง DAG ให้อยู่ในรูปของต้นไม้กลับตัวชี้(เหมือนกับขั้นตอนการทำงานที่เคยกล่าวเอาไว้ในบทที่แล้ว) ดังแสดงอยู่รูปที่ 5.6 จากนั้นหากกดปุ่มทูลบาร์ "Binary Tree" โปรแกรมเอทีซี

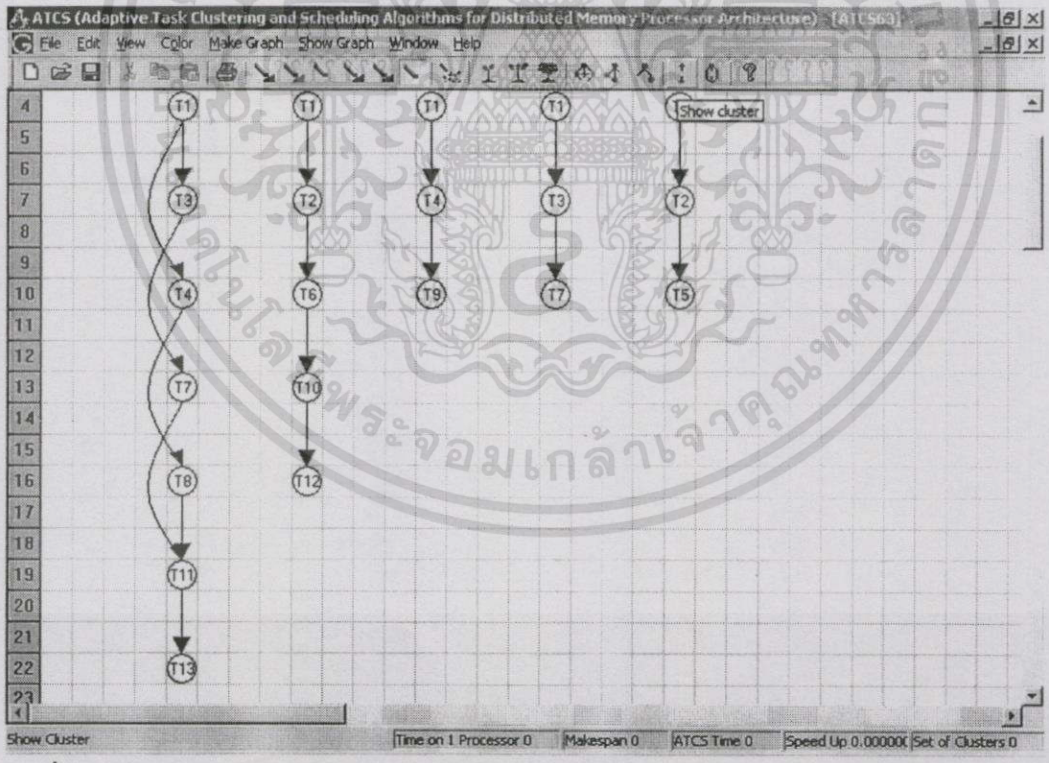
เอสจะทำการแปลงต้นไม้กลับตัวชี้ให้อยู่ในรูปของต้นไม้ทวิภาคแบบกลับตัวชี้ดังแสดงอยู่ในรูปที่ 5.7 รูปที่ 5.8 เป็นการแสดงชุดของกลุ่มงานเมื่อผู้ใช้กดปุ่มทูลบาร์ “Show Cluster” ในส่วนนี้โปรแกรมเอทีซีเอสจะทำการสร้างชุดของกลุ่มงาน



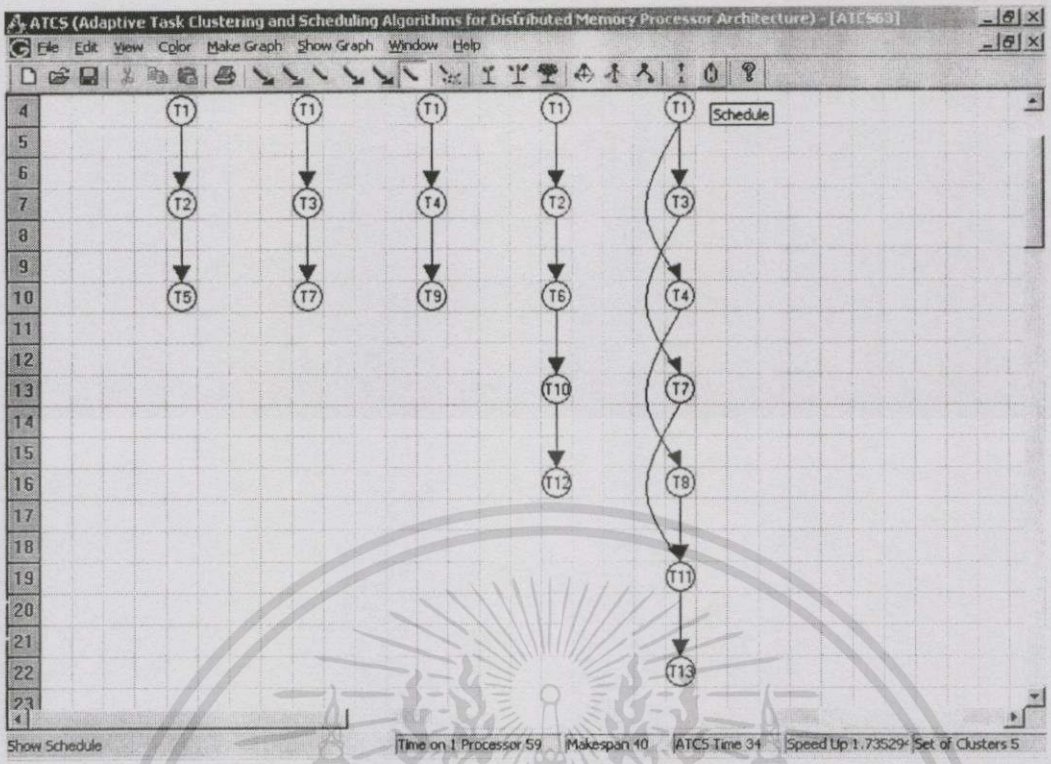
รูปที่ 5.6 แสดงการแปลง DAG ให้อยู่ในรูปของต้นไม้กลับตัวชี้



รูปที่ 5.7 แสดงการแปลงต้นไม้กลับตัวชี้ให้อยู่ในรูปของต้นไม้ทวิภาคกลับตัวชี้



รูปที่ 5.8 แสดงชุดของกลุ่มงาน Φ



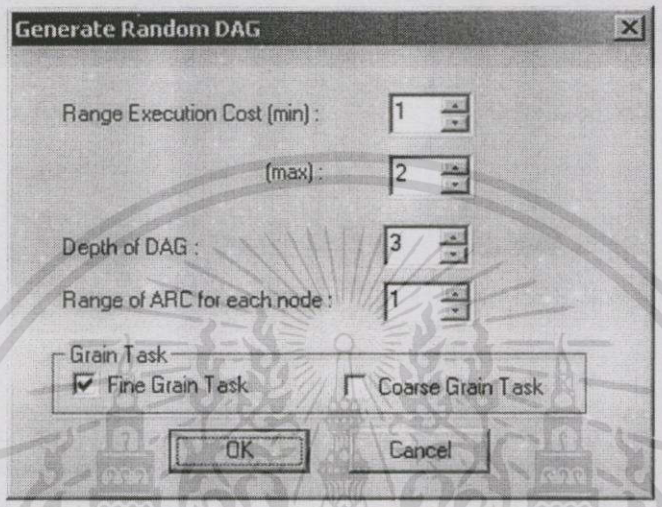
รูปที่ 5.9 แสดงการจัดลำดับการทำงานให้กับชุดของกลุ่มงาน Φ(DAG)

รูปที่ 5.9 เป็นการแสดง Φ(DAG) ที่ได้จากการจัดลำดับการทำงานให้กับชุดของกลุ่มงาน ซึ่งเป็นขั้นตอนสุดท้ายในการทำงานของโปรแกรมเอทีซีเอส เมื่อผู้ใช้กดปุ่มทูลบาร์ "Schedule" นอกเหนือไปจากการแสดง Φ(DAG) บนหน้าต่าง(Window)แล้วที่สเตตัสบาร์(Status Bar)ยังแสดงข้อมูลต่างๆดังนี้

1. ค่า Time on 1 Processor เป็นค่าของเวลาที่ใช้ในการประมวลผลของงานต่างๆ ที่ถูกวางอยู่บนตัวประมวลผลเพียงตัวเดียว โดยการใช้อัลกอริทึมแบบเรียงลำดับเป็นตัวจัดลำดับงานให้
2. ค่า Makespan คือค่าเส้นทางวิกฤตที่อยู่บน DAG หรือเป็นค่าของลำดับการทำงานของโปรแกรมแบบขนานที่ยังมิได้ถูกเปลี่ยนรูปโดยอัลกอริทึมใดๆ
3. ค่า ATCS Time คือค่าของลำดับการทำงานของชุดของกลุ่มงานที่ได้จากเอทีซีเอสอัลกอริทึม ค่าดังกล่าวจะได้รับก็ต่อเมื่อการทำงานของเอทีซีเอสอัลกอริทึมเสร็จสิ้นสมบูรณ์แล้ว หรือค่าของ Makespan ของเอทีซีเอสอัลกอริทึม
4. ค่า Speed Up เป็นค่าของการวัดความเร็ว ดังที่แสดงอยู่ในสมการที่ 5.3 ในส่วนที่ 5.5 โดยค่าของ Speed Up = (Time on 1 Processor)/(ATCS Time)
5. ค่า set of Task-Clusters เป็นการบ่งบอกถึงจำนวนของกลุ่มงานที่อยู่ในชุดของกลุ่มงาน ที่จะนำเอาไปใช้งานจริงในการจัดลำดับการทำงาน หรือเป็นการบ่ง

บอกถึงจำนวนของตัวประมวลผลที่จะต้องใช้ในการประมวลผลให้โปรแกรมแบบขนาน หลังจากที่ใช้เอทีซีเอสอัลกอริทึมในการจัดกลุ่มงานและลำดับการทำงาน

จากค่าต่างๆที่กล่าวมา เราสามารถนำเอาค่าเหล่านี้ไปทำการวัดประสิทธิภาพ(ดังที่ได้เคยกล่าวมาแล้วในบทที่แล้ว)ของเอทีซีเอสอัลกอริทึม ที่มีต่อโปรแกรมแบบขนานนี้ได้



รูปที่ 5.10 แสดงไดอะล็อกบ็อก "Generate Random DAG"

ส่วน DAG ที่ได้จากการสุ่ม โดยโปรแกรมเอทีซีเอสจะพยายามสร้าง DAG ให้ได้ใกล้เคียงกับ DAG ในลักษณะที่เป็นจริงให้มากที่สุด อย่างเช่น โปรแกรมจะไม่สร้างความสัมพันธ์ของโหนดที่ต่างระดับชั้นเกินกว่าสองชั้น(โดยการใช้อาร์คเป็นตัวสร้างความสัมพันธ์ระหว่างโหนด) ขอบเขตของการแกว่งของอาร์คที่ออกจากโหนดแต่ละโหนดจะต้องไม่มากจนเกินไป ความกว้างของ DAG และระดับความลึกของ DAG จะต้องไม่ต่างกว่ากันมากนัก เป็นต้น DAG ที่ได้จากการสุ่มดังแสดงอยู่ในส่วนของผลการทดลอง จะใช้ DAG ที่มีจำนวนของโหนดขนาดต่างๆกัน รูปที่ 5.10 เป็นการแสดงไดอะล็อกบ็อก "Generate Random DAG" เมื่อผู้ใช้กดปุ่มทูลบาร์ "Generate DAG" โปรแกรมเอทีซีเอสจะแสดงไดอะล็อกบ็อกขึ้นมา บนไดอะล็อกบ็อกบอกถึงสิ่งที่ผู้ใช้จะต้องทำการป้อนข้อมูลต่างๆดังนี้

1. Range Execution Cost (min) และ (max) เป็นการให้ระบุขอบเขตของค่าใช้จ่ายในการประมวลผลของงานต่างๆที่อยู่บน DAG ด้วยค่าที่น้อยที่สุด(min) และค่าที่มากที่สุด(max) ค่ากำหนดเริ่มต้นที่น้อยที่สุดโดยปริยาย(Default Min value)เท่ากับ 1 และค่ากำหนดเริ่มต้นที่มากที่สุดโดยปริยาย(Default Max value)เท่ากับ 2

2. Depth of DAG เป็นการระบุความลึกของระดับชั้นของ DAG ค่าที่กำหนดให้มีค่าความลึกโดยปริยาย(Default Depth value)เท่ากับ 3
3. Range of ARC for each node เป็นการระบุจำนวนของอาร์คที่ชี้ออกจากโหนดแต่ละโหนด กำหนดให้ค่าโดยปริยายสำหรับจำนวนของอาร์คเท่ากับ 1 แสดงว่าจำนวนของอาร์คสำหรับแต่ละโหนดที่อยู่บน DAG จะมีค่าอย่างน้อยที่สุดและมากที่สุดเท่ากับ 1 หากระบุค่าเท่ากับ 4 แสดงว่าจำนวนของอาร์คสำหรับแต่ละโหนดที่อยู่บน DAG จะมีค่าอย่างน้อยที่สุดเท่ากับ 1 และมากที่สุดเท่ากับ 4 สำหรับแต่ละโหนดใดๆ
4. Grain Task เป็นการระบุว่าต้องการให้ทำการสร้าง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด(Granularity หรือ $g(G) < 1$) หรือแบบหยาบ(Granularity ≥ 1)

จะสังเกตพบว่าไม่ได้มีการระบุค่าของค่าใช้จ่ายในการติดต่อสื่อสารข้อมูลหรืออาร์คสำหรับ DAG ที่ได้จากการสุ่ม แต่โปรแกรมเอทีซีเอสจะทำการสร้างให้ โดยอาศัยข้อมูลของ Rang Execution Cost และ Grain Task เป็นข้อมูลสำหรับการสุ่มค่าของค่าใช้จ่ายในการติดต่อสื่อสารข้อมูลที่อยู่บน DAG โดยแสดงอยู่ในรูปของค่า Granularity

เนื่องมาจากเป้าหมายของวิทยานิพนธ์เล่มนี้ต้องการแสดงวิธีการในการจัดกลุ่มงานและกำหนดลำดับการทำงานให้กับ DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียดเป็นสิ่งสำคัญ ดังนั้นการทดลองโดยส่วนใหญ่จะกระทำกับ DAG ที่มีลักษณะเป็นแบบเม็ดเนื้องานแบบละเอียด(Granularity < 1) ที่มีค่าของ Granularity ขนาดต่างๆกัน และทราบดีแล้วว่าลักษณะของ DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียดโดยรวมแล้วจะมีค่า $\lambda > \mu$ ดังนั้นในการระบุว่า DAG ใดๆที่มีลักษณะเป็นเม็ดเนื้องานแบบละเอียดและมีค่าของ Granularity ขนาดเท่ากับ 1/2 หมายถึง DAG นั้นจะมีค่าของ λ อย่างมากที่สุดเป็น 2 เท่าของ μ ที่น้อยที่สุด และอย่างน้อยที่สุดจะต้องมากกว่าค่าของ μ ที่มากที่สุด นอกจากนี้ยังได้ทำการทดลองกับ DAG แบบอื่นๆที่อยู่นอกเหนือขอบเขตเป้าหมายของวิทยานิพนธ์ฉบับนี้ กล่าวคือเป็น DAG ที่มีลักษณะเม็ดเนื้องานแบบหยาบ และ DAG ที่มีลักษณะนอกเหนือไปจากนิยามของ[2] เมื่อลักษณะเม็ดเนื้องานแบบหยาบโดยรวมแล้วจะมีค่าของ $\mu > \lambda$ ดังที่ได้เคยอธิบายไว้แล้วในบทที่ 1 โดยอาศัยนิยามของ[2] เป็นหลักในการอธิบาย หากแต่ว่านิยามของ[2] ได้กำหนดไว้ว่าDAG ใดๆที่มีลักษณะเป็นเนื้องานแบบหยาบก็ต่อเมื่อ งานหรือโหนดแต่ละโหนดจะมีค่าใช้จ่ายในการขนส่งหรือรับข้อมูลน้อยมาก เมื่อเทียบกับค่าใช้จ่ายในการประมวลผลของโหนดๆนั้น ดังนั้นDAG ใดๆจะมีลักษณะเม็ดเนื้องานแบบหยาบก็ต่อเมื่อมีความแตกต่างกันระหว่างค่าของ μ และλ ค่อนข้างมากหรืออย่างน้อยที่สุดแล้ว ค่าของ μ จะต้องมีค่าเป็นสองเท่าของ λ ส่วน DAG ลักษณะที่เหลือจะกล่าวต่อมาในภายหลัง

ตัวอย่างของ DAG ที่ได้จากการสุ่มจะแสดงอยู่ในส่วนของ 5.2 ขั้นตอนการทำงานในส่วนที่เหลือจะเหมือนกับการสร้าง DAG กล่าวคือผู้ใช้สามารถทำการกดปุ่มทูลบาร์ "Graphs" ได้เลยเพื่อให้โปรแกรมเอทีซีเอสสร้าง DAG ขึ้นมาให้ จากนั้นทำการแปลง DAG ให้อยู่ในรูปของต้นไม้กลับตัวชี้ในขั้นตอนถัดไปโดยการกดปุ่มทูลบาร์ "Tree" และในขั้นตอนที่เหลือจะเหมือนกับวิธีการที่ได้กล่าวไว้ในส่วนบน

ในการทดลองการทำงานและประสิทธิภาพของเอทีซีเอสอัลกอริทึม จะทำการทดลองด้วย DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และแบบหยาบ ดังที่ได้เคยกล่าวไว้แล้วว่าในการทดลองส่วนใหญ่ของเอทีซีเอสอัลกอริทึม จะทำการทดลองด้วย DAG ที่มีลักษณะเป็นแบบเม็ดเนื้องานแบบละเอียดเป็นส่วนใหญ่ และทำการทดลองกับ DAG ที่มีลักษณะเม็ดเนื้องานแบบหยาบด้วย นอกจากนี้ยังได้ทำการทดลองด้วย DAG ที่มีลักษณะแตกต่างไปจากนิยามของ[2] กล่าวคือเป็น DAG ที่มีลักษณะค่าของ Grain ที่มีความแตกต่างระหว่างค่าของ λ และ μ น้อยมากหรือไม่มีความแตกต่างกันเลย($\mu / \lambda = 1$) สำหรับแต่ละโหนดใดๆที่อยู่บน DAG ดังปรากฏอยู่ในรูปที่ 5.45 [32] และ 5.46-5.47(ได้จากการสุ่มของโปรแกรม) จะทำการแสดงอยู่ในส่วนที่ 5.2.3 เราอาจจะกล่าวได้ว่าลักษณะของ DAG ดังกล่าวเป็นแบบ Grain แบบหยาบที่มีลักษณะแตกต่างไปจากนิยามของ[2]

5.2 ผลการทดลองของเอทีซีเอสอัลกอริทึม

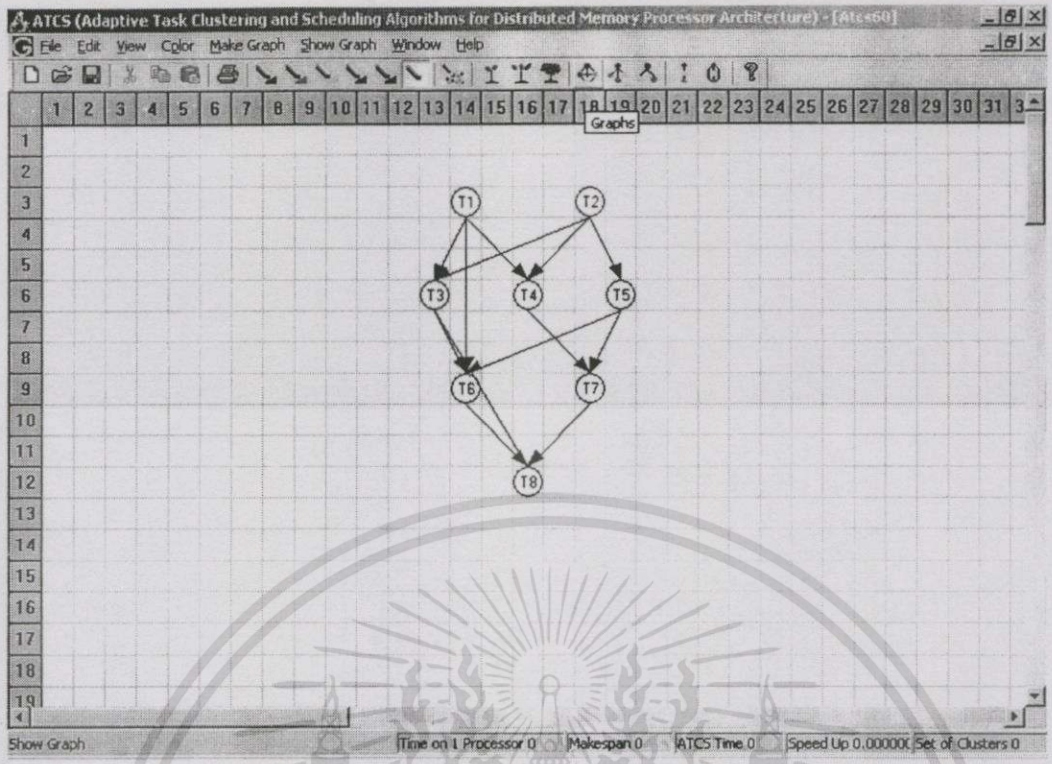
ในการแสดงผลการทดลองของ DAG แต่ละ DAG โดยวิธีการของเอทีซีเอสอัลกอริทึม เราจะไม่ทำการแสดงทุกขั้นตอนการทำงาน เช่น DAG $\rightarrow$ ต้นไม้กลับตัวชี้ $\rightarrow$ ต้นไม้ทวิภาคแบบกลับตัวชี้ $\rightarrow$ การสร้างชุดของกลุ่มงาน $\rightarrow$ การกำหนดลำดับการทำงาน เป็นต้น โดยเราจะแสดง DAG ที่ได้จากการป้อนข้อมูลต่างๆให้กับ DAG เช่น ค่าใช้จ่ายในการประมวลผลของงานต่างๆ จำนวนระดับความลึกของ DAG เป็นต้น และจะทำการแสดงผลลัพธ์ในขั้นตอนสุดท้ายคือ การกำหนดลำดับการทำงาน ซึ่งจะได้ค่าต่างๆ คือ 1)ค่าของ Time on 1 Processors 2)ค่าของ Makespan 3)ค่าของ ATCS Time 4)ค่าของ Speed Up และ 5)ค่าของ set of Task-Clusters หรือจำนวนของตัวประมวลผลที่ต้องใช้ในการประมวลผลชุดของกลุ่มงานที่ได้จากเอทีซีเอสอัลกอริทึม

นอกจากนี้ในกรณีของ DAG ที่ได้จากการสุ่ม ในโปรแกรมเอทีซีเอสได้มีการกำหนดไว้ว่า หากจำนวนของโหนดที่อยู่บน DAG มากไปกว่า 50 โหนด โปรแกรมเอทีซีเอสจะไม่ทำการแสดงผลลัพธ์ของการเปลี่ยนแปลงให้เห็นบนหน้าจอหรือสกรีน(Screen) เนื่องมาจากในการแสดงผลลัพธ์ของการเปลี่ยนแปลงจำเป็นที่จะต้องใช้เนื้อที่บนสกรีนและหน่วยความจำค่อนข้างมาก แต่โปรแกรมเอทีซีเอสยังคงมีการประมวลผลอยู่ในแต่ละขั้นตอนการทำงาน เพียงแต่ไม่ได้แสดงผลลัพธ์ให้เห็นเท่านั้น โดยจะทำการแสดงเฉพาะในขั้นตอนแรกคือการแสดง DAG ในขั้นตอนสุดท้าย

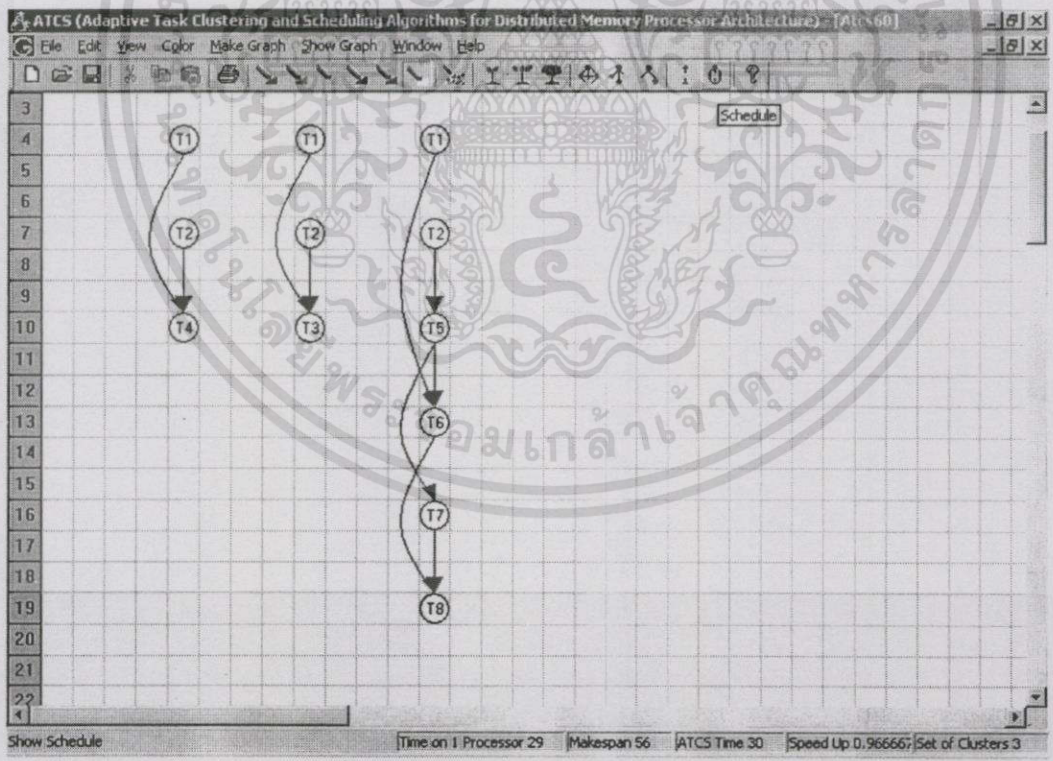
ของการทำงานของโปรแกรมเอทีซีเอส จะให้ 1)ค่าของ Time on 1 Processor 2)ค่าของ Makespan 3)ค่าของ ATCS Time 4)ค่าของ Speed Up และ5)ค่าของ set of Task-Clusters อยู่เช่นเดิม

5.2.1 ผลการทดลองกับ DAG ที่มีลักษณะเป็นเม็ดเนื้องานแบบละเอียด ในส่วนนี้เป็นการแสดงผลการทดลองกับ DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด ด้วยค่าขนาดของ Granularity ที่ต่างๆกัน และขนาดของจำนวนโหนดที่ปรากฏอยู่บน DAG มีขนาดต่างๆกัน โดยผลการทดลองที่ปรากฏอยู่ในรูปที่ 5.11-5.12 เป็น DAG ที่สร้างขึ้นมาและ DAG ที่ได้มาจาก[23]ตามลำดับ และผลการทดลองที่ปรากฏอยู่ในรูปที่ 5.13-5.43 เป็น DAG ที่ได้มาจากการสุ่มโดยใช้โปรแกรมเอทีซีเอส สุ่มสร้างขึ้นมา

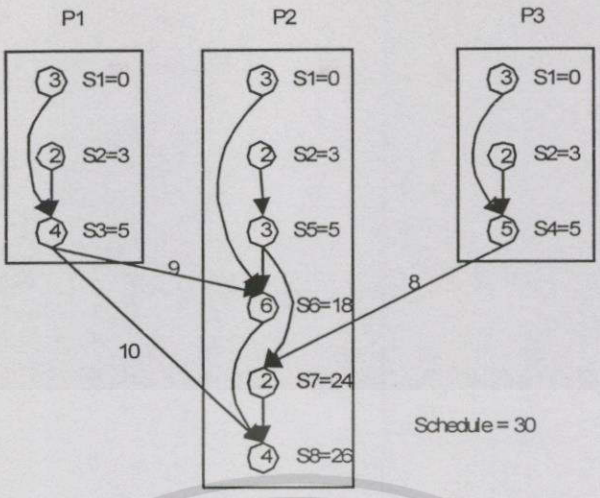
จากผลการทดลองในกรณีของ DAG ที่มีลักษณะเป็นเม็ดเนื้องานแบบละเอียดทั้งหมด 33 การทดลอง ด้วยลักษณะของ DAG ที่หลากหลายรูปแบบ ที่มีจำนวนของโหนดบน DAG จากจำนวน 8 โหนดไปจนถึงจำนวนกว่า 100 โหนด ลักษณะของ DAG ที่มีความหนาแน่นของจำนวนของอาร์คที่ต่างๆกัน และระดับความลึกหรือระดับชั้นของ DAG ที่ต่างๆกัน แสดงผลลัพธ์ออกมาที่แตกต่างกันออกไป จากผลการทดลองทั้งหมด 33 การทดลองแสดงให้เห็นว่าเอทีซีเอสอัลกอริทึมสามารถทำการจัดกลุ่มงานและกำหนดลำดับการทำงานให้กับโปรแกรมแบบขนานได้อย่างมีประสิทธิภาพ ไม่ว่า DAG นั้นจะมีจำนวนขนาดของโหนดที่มาก(มากกว่า 100 โหนด) เอทีซีเอสอัลกอริทึมก็ยังสามารถดำเนินการจัดกลุ่มงานและกำหนดลำดับการทำงานให้กับโปรแกรมแบบขนานได้เช่นกัน



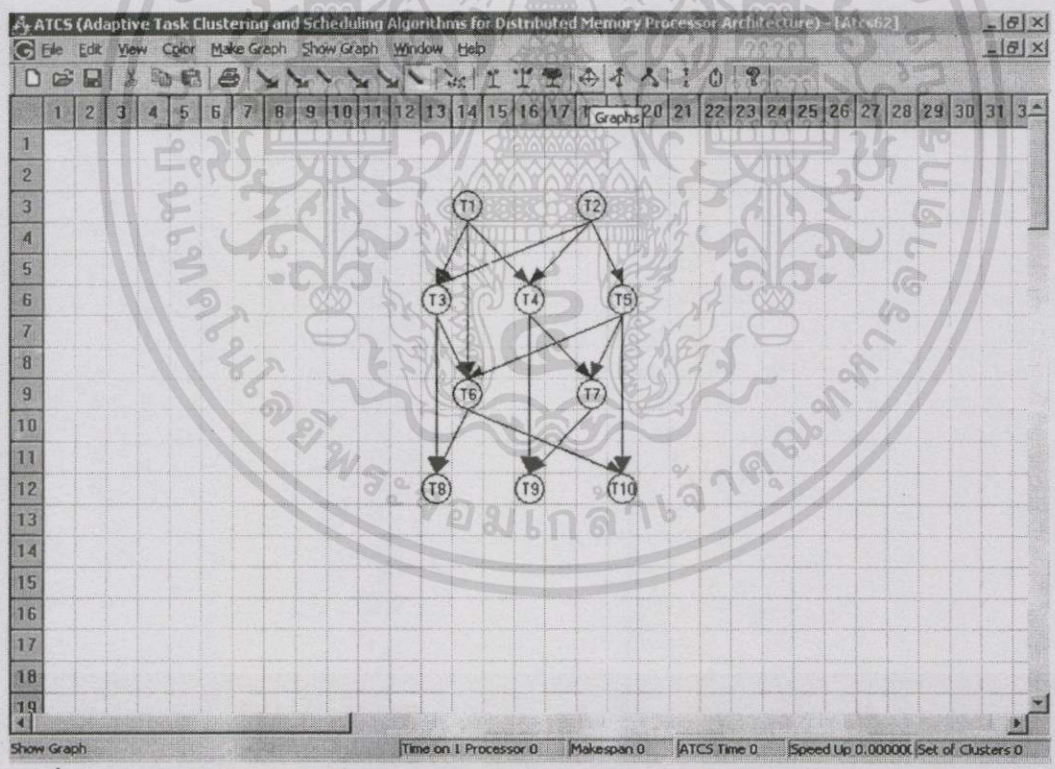
รูปที่ 5.11(a) แสดง DAG เริ่มต้น โดยมีค่าของ Granularity เท่ากับ 1/7



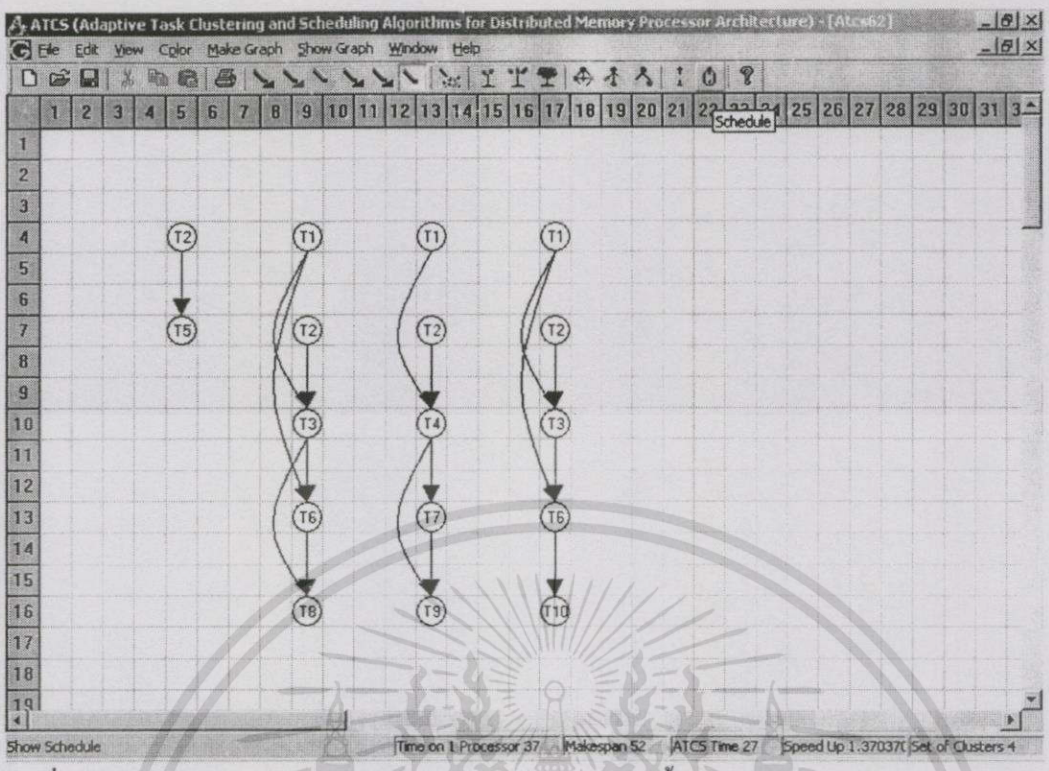
รูปที่ 5.11(b) แสดงผลลัพธ์ในขั้นตอนสุดท้ายของเอทีซีเอสอัลกอริทึมจาก DAG รูปที่ 5.11(a) โดยแสดงค่าของ Speed Up เท่ากับ 0.9666



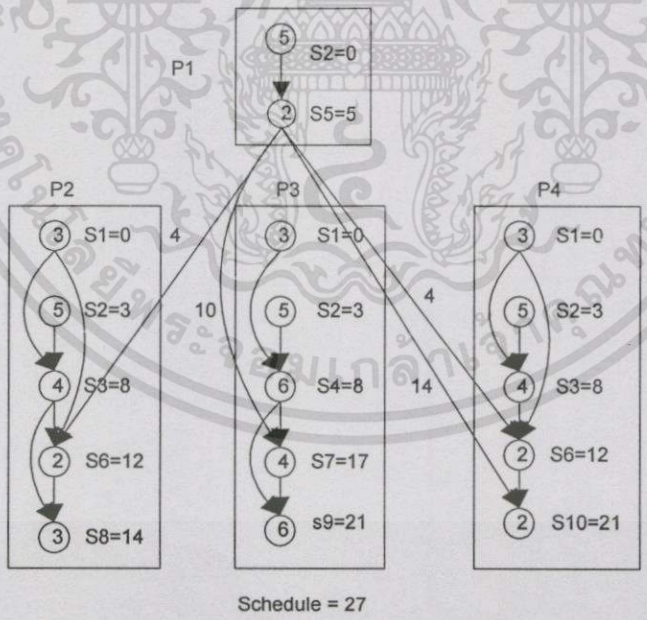
รูปที่ 5.11(c) แสดงการจัดลำดับการทำงานของกลุ่มงานต่างๆ เมื่อนำเอากลุ่มงานเหล่านั้นวางลงบนตัวประมวลผลต่างๆ



รูปที่ 5.12(a) แสดง DAG[23] ที่มีค่าของ Granularity เท่ากับ 1/7



รูปที่ 5.12(b) แสดงให้เห็นถึงค่าของ Speed Up สำหรับ DAG นี้มีค่าเท่ากับ 1.3703



รูปที่ 5.12(c) แสดงการจัดลำดับการทำงานของกลุ่มงานต่างๆ เมื่อนำเอากลุ่มงานเหล่านั้นวางลงบนตัวประมวลผลต่างๆ

ตารางที่ 5.1 แสดงค่าใช้จ่ายในการประมวลผล(μ) ระดับของโหนด และค่าใช้จ่ายในการติดต่อสื่อสาร(λ)ระหว่างโหนดต้นทางไปยังโหนดปลายทางบน DAG รูปที่ 5.11

Node No.	Execution Cost(μ)	Level
T1	3	0
T2	2	0
T3	4	1
T4	5	1
T5	3	1
T6	6	2
T7	2	2
T8	4	3

Source Node	Destination Node	Communication Cost(λ)
T1	T3	8
T1	T4	10
T1	T6	15
T2	T3	6
T2	T4	7
T2	T5	12
T3	T6	9
T3	T8	10
T4	T7	8
T5	T6	16
T5	T7	14
T6	T8	13
T7	T8	17

ตารางที่ 5.2 แสดงค่าใช้จ่ายในการประมวลผล(μ) ระดับของโหนด และค่าใช้จ่ายในการติดต่อสื่อสาร(λ)ระหว่างโหนดต้นทางไปยังโหนดปลายทางบน DAG รูปที่ 5.12

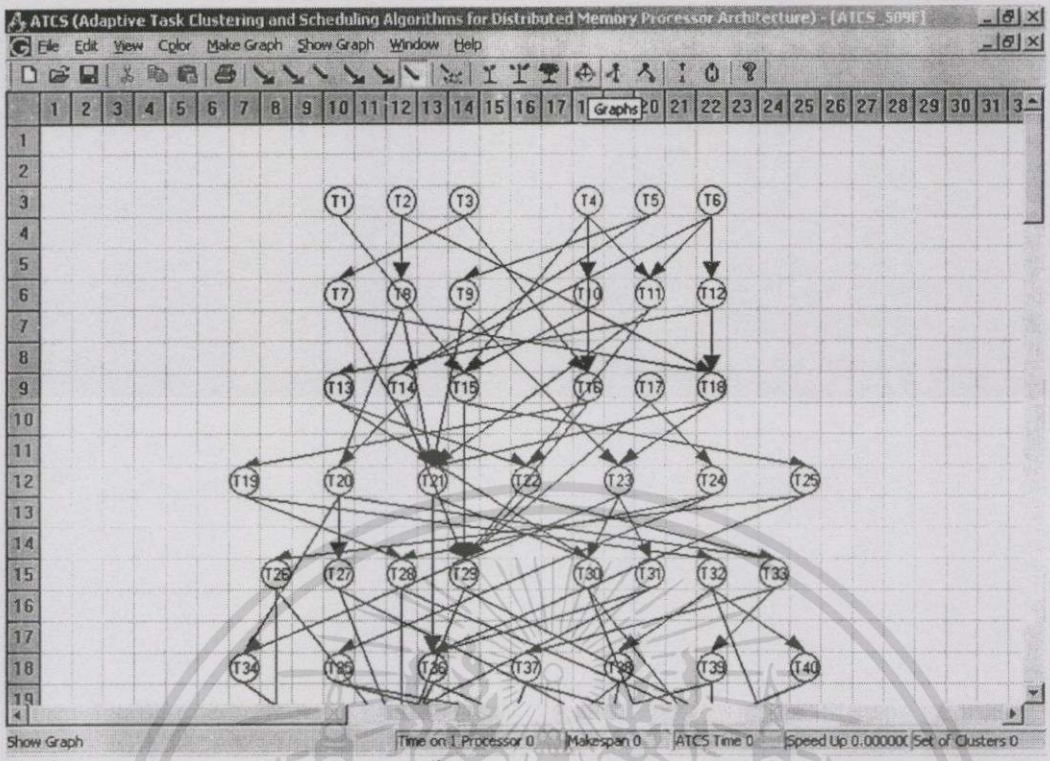
Node No.	Execution Cost(μ)	Level
T1	3	0
T2	5	0
T3	4	1
T4	6	1
T5	2	1
T6	2	2
T7	4	2
T8	3	3
T9	6	3
T10	2	3

Source Node	Destination Node	Communication Cost(λ)
T1	T3	8
T1	T4	7
T1	T6	6
T2	T3	10
T2	T4	14
T2	T5	10
T3	T6	5
T3	T8	8
T4	T7	12
T4	T9	7
T5	T6	4
T5	T7	10
T5	T10	14
T6	T8	4
T6	T10	8
T7	T9	5

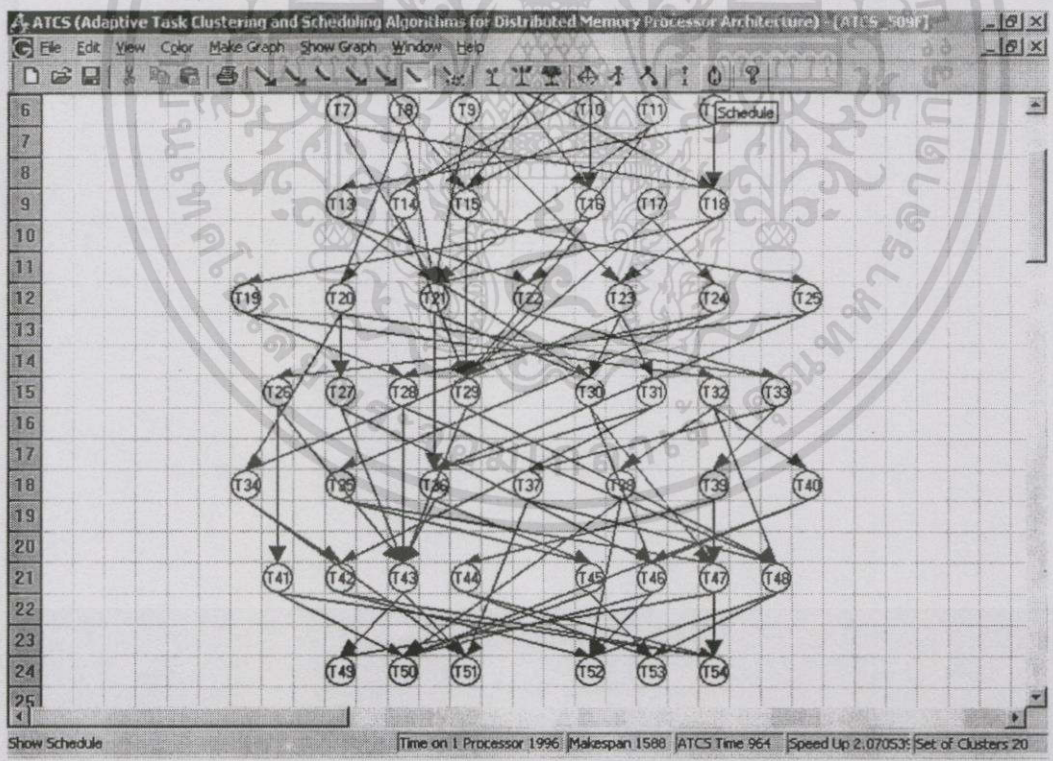
ตารางที่ 5.1 และ 5.2 แสดงค่าใช้จ่ายในการประมวลผลของโหนดแต่ละโหนด ระดับของโหนดที่อยู่บนDAG และค่าใช้จ่ายในการติดต่อสื่อสารจากโหนดต้นทางไปยังโหนดปลายทาง ของ DAG ในรูปที่ 5.11-5.12 ตามลำดับ

รูปที่ 5.13-5.43 เป็นการแสดงถึงความแตกต่างที่ชัดเจนในลักษณะการกระจายของภาระงานที่แตกต่างกันของโหนดต่างๆที่อยู่บน DAG ในลักษณะที่แตกต่างกัน โดยลักษณะของ DAG ที่มีการกระจายภาระงานให้กับโหนดต่างๆได้อย่างใกล้เคียงกัน ผลที่ได้รับจากการใช้เอทีซีเอสอัลกอริทึม ทำให้เวลาที่ใช้ในการประมวลผลลดลงได้อย่างมาก หากเมื่อทำการเปรียบเทียบกับ DAG ที่มีลักษณะการกระจายของภาระงานที่ไม่เท่ากันหรือมีความแตกต่างกันค่อนข้างมาก ทำให้เวลาที่ใช้ในการประมวลผลลดลงค่อนข้างน้อย นอกจากนี้สามารถสังเกตได้ด้วยว่า เวลาที่ใช้ในการประมวลผลของเอทีซีเอสอัลกอริทึม(ATCS Time)ลดลงมากน้อยเพียงใด เมื่อเทียบกับเวลาของ Makespan ไม่ได้ส่งผลให้ค่าของความเร็วหรือค่าของ Speed Up เพิ่มมากขึ้นหรือลดลงเลย

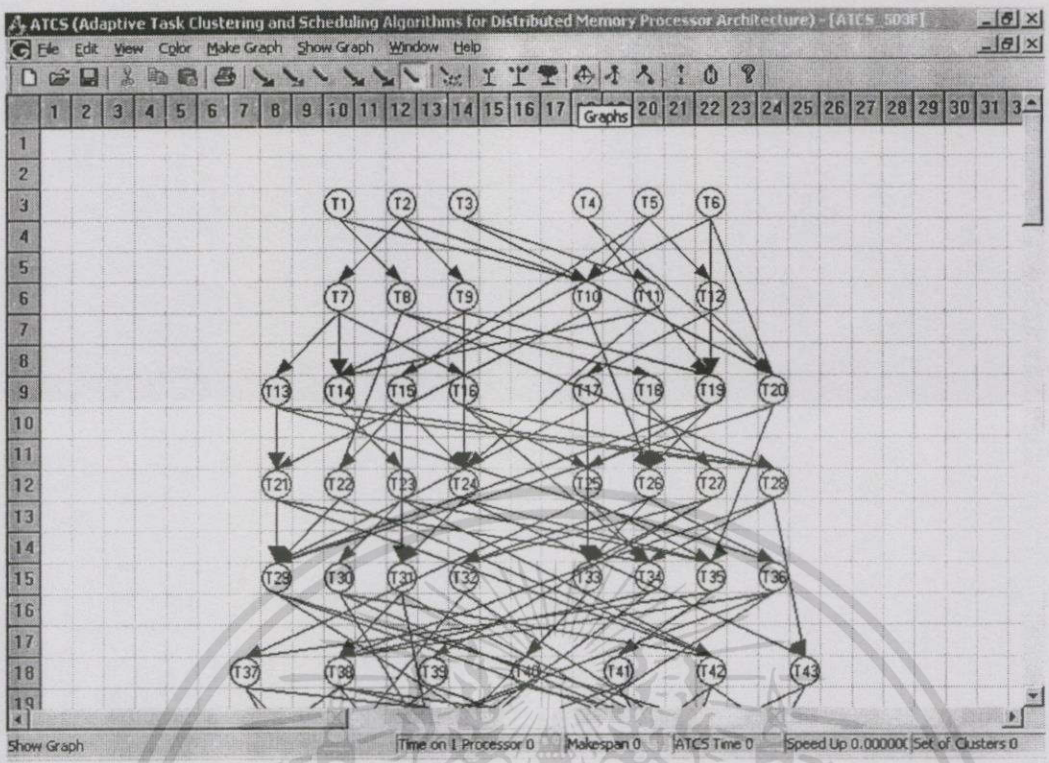
รูปที่ 5.34-5.43 เป็นการแสดง DAG ที่มีขนาดมากกว่า 100 โหนด จากการทดลองการจัดกลุ่มงานและกำหนดลำดับการทำงานโดยเอทีซีเอสอัลกอริทึม แสดงให้เห็นว่าความสามารถของเอทีซีเอสอัลกอริทึมยังคงสามารถลดเวลาการประมวลผลของโปรแกรมแบบขนานลงได้ในระดับที่ค่อนข้างมาก โดยสังเกตได้จากค่าของ Speed Up เมื่อเปรียบเทียบกับ DAG ที่มีขนาดของโหนดน้อยกว่า ความสามารถในการลดเวลาการประมวลผลของโปรแกรมแบบขนานยังคงเพิ่มมากขึ้น หากพิจารณาจากค่าของ Speed Up แล้วอาจจะกล่าวได้ว่าเมื่อขนาดของ DAG ใหญ่ขึ้นความสามารถของเอทีซีเอสอัลกอริทึมไม่ได้ด้อยลงได้



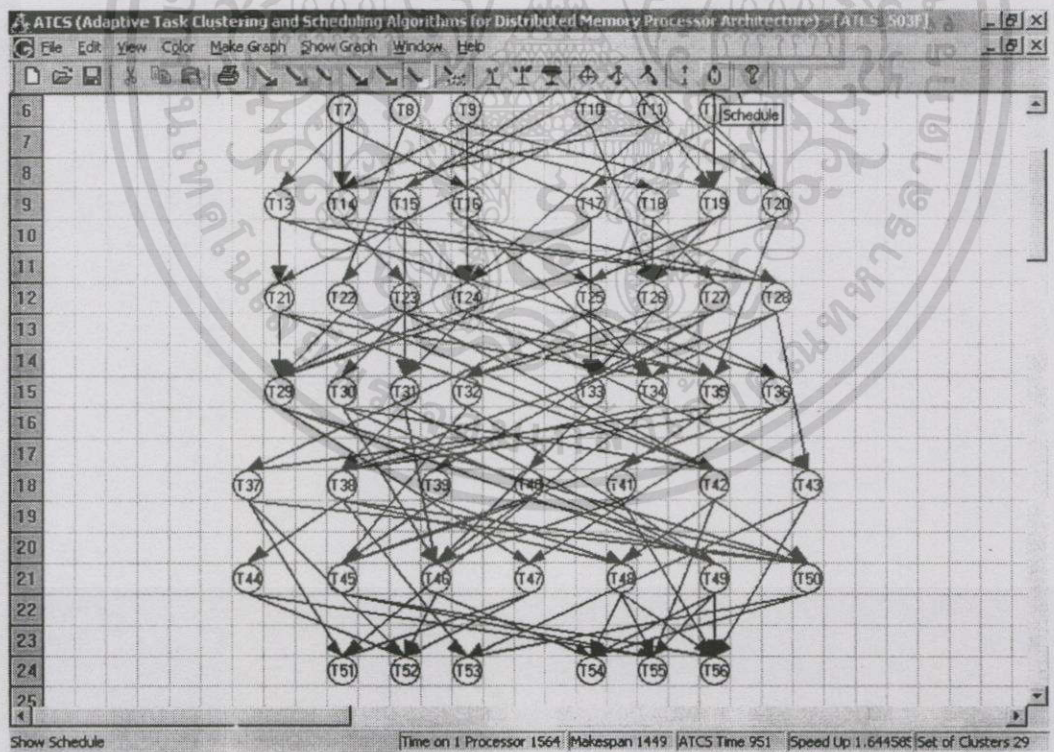
รูปที่ 5.13 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/6



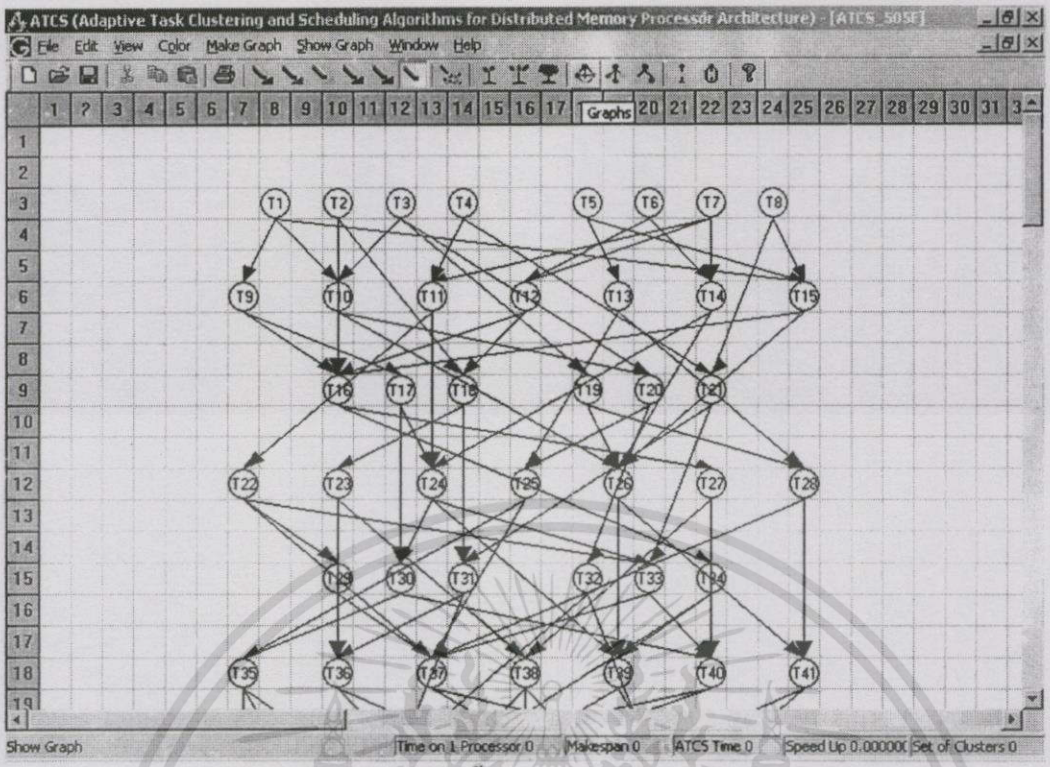
รูปที่ 5.13(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1588-964 = 624 หน่วย และค่าของ Speed Up เท่ากับ 2.0705 โดยกำหนดให้มี $33 \leq \mu \leq 40$



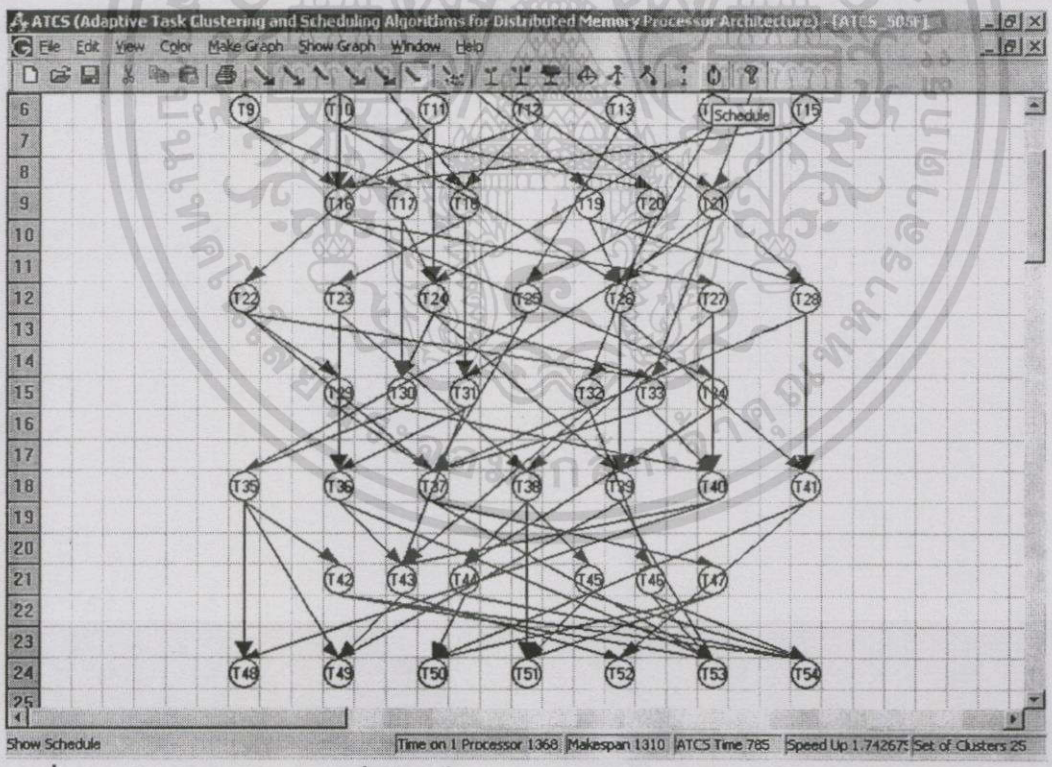
รูปที่ 5.14 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/8



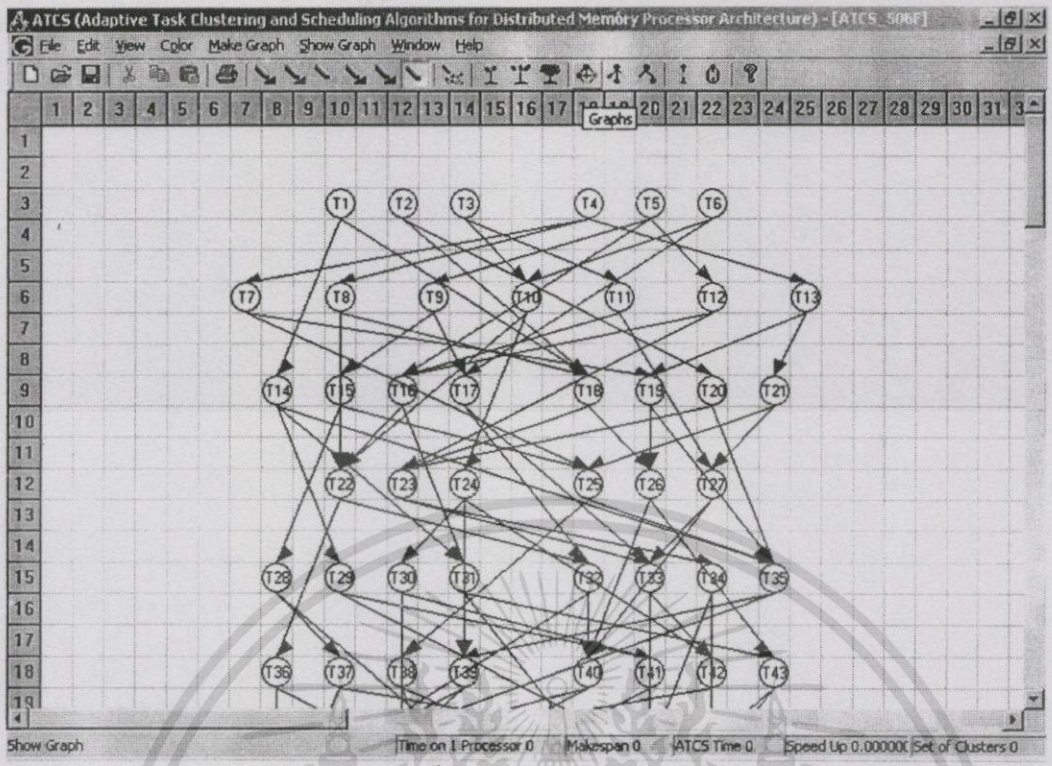
รูปที่ 5.14(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1449-951 = 498 หน่วย และค่าของ Speed Up เท่ากับ 1.6445 โดยกำหนดให้มี $22 \leq \mu \leq 34$



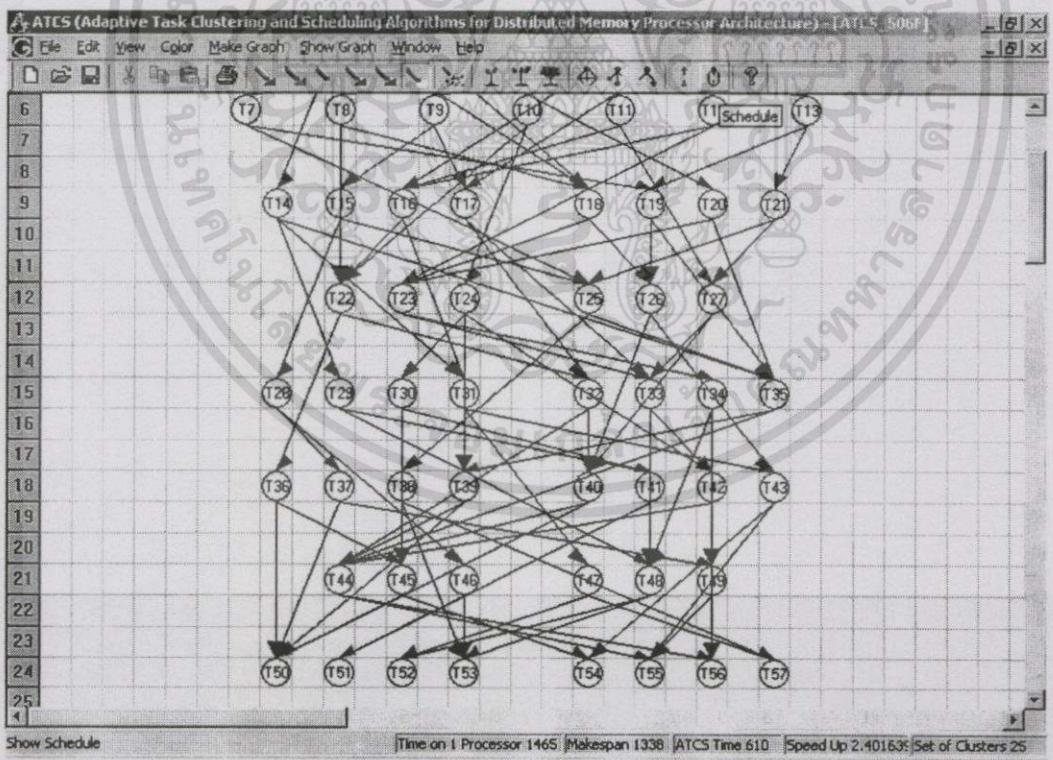
รูปที่ 5.15 แสดง DAG ที่มีลักษณะเป็นเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/7



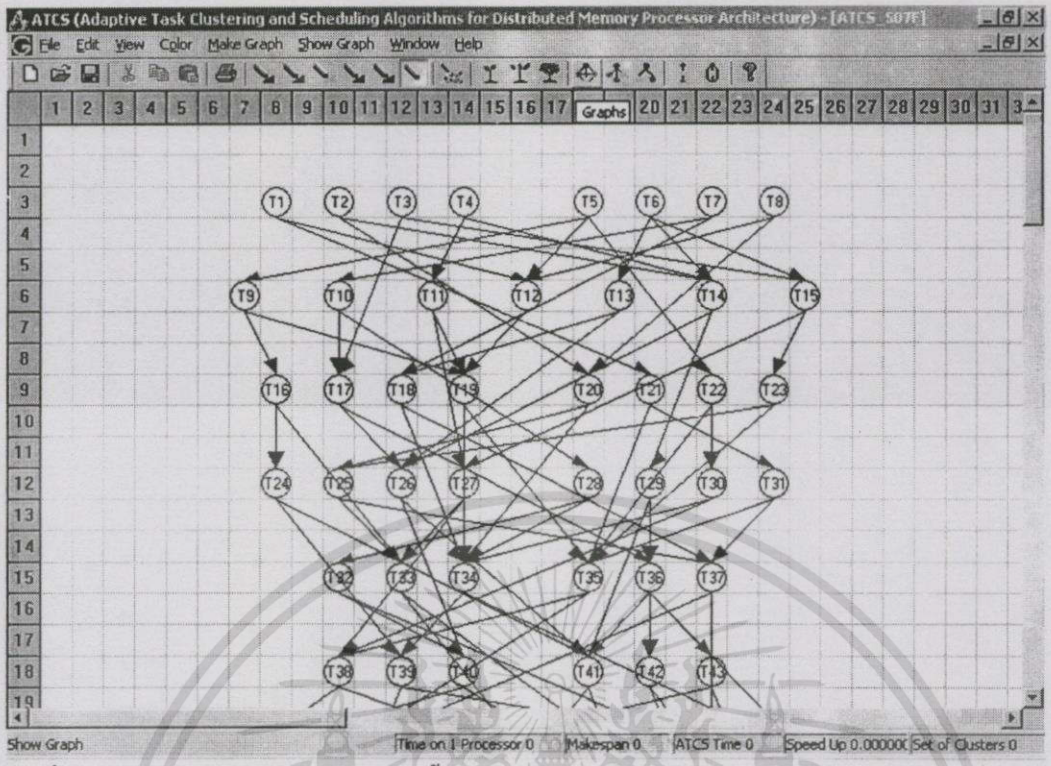
รูปที่ 5.15(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลดเท่ากับ 1310-785 = 525 หน่วย และค่าของ Speed Up เท่ากับ 1.7426 โดยกำหนดให้มี $23 \leq \mu \leq 32$



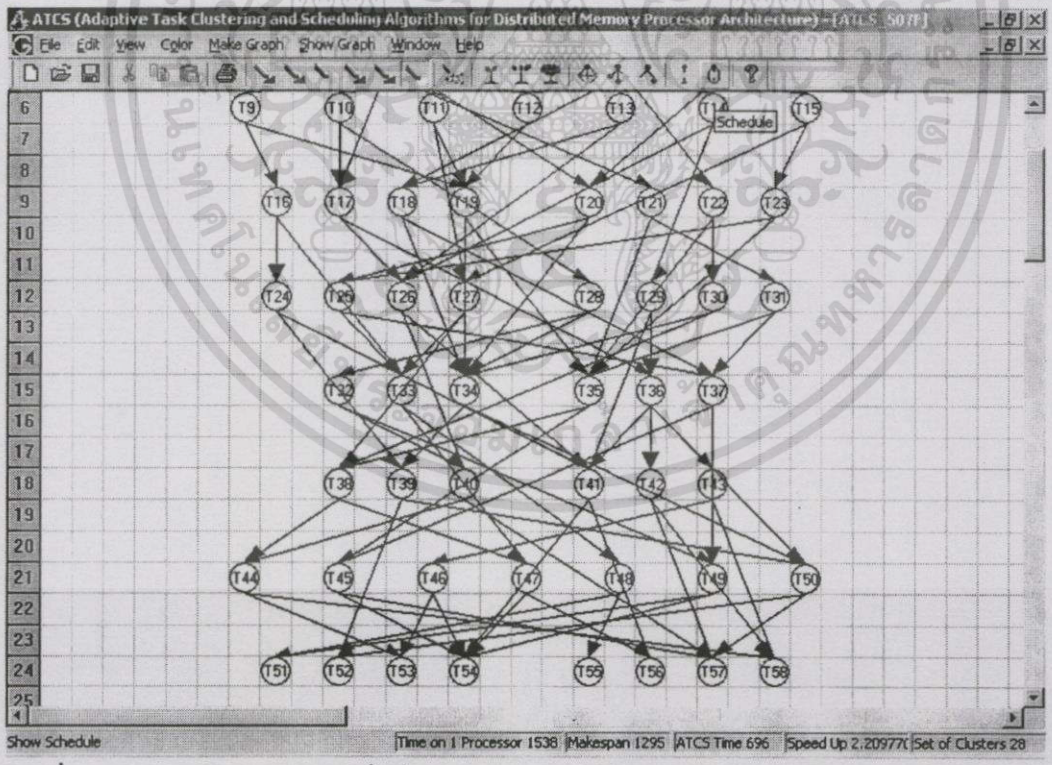
รูปที่ 5.16 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/8



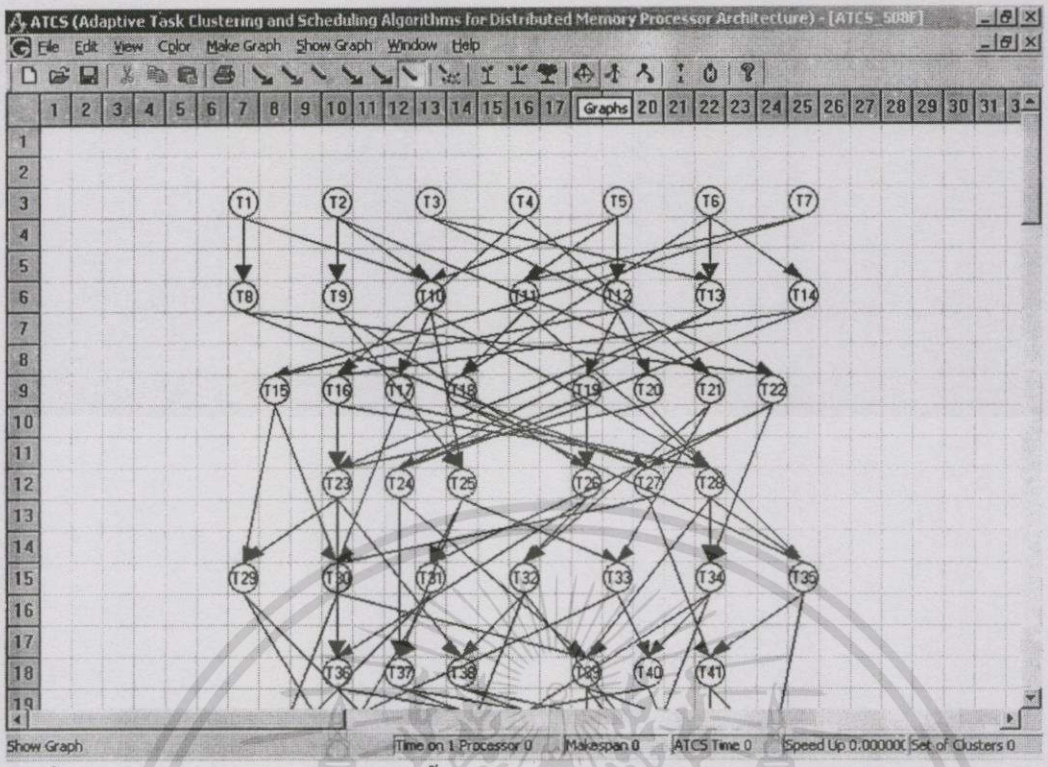
รูปที่ 5.16(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1338 - 610 = 728 หน่วย และค่าของ Speed Up เท่ากับ 2.4016 โดยกำหนดให้มี $22 \leq \mu \leq 32$



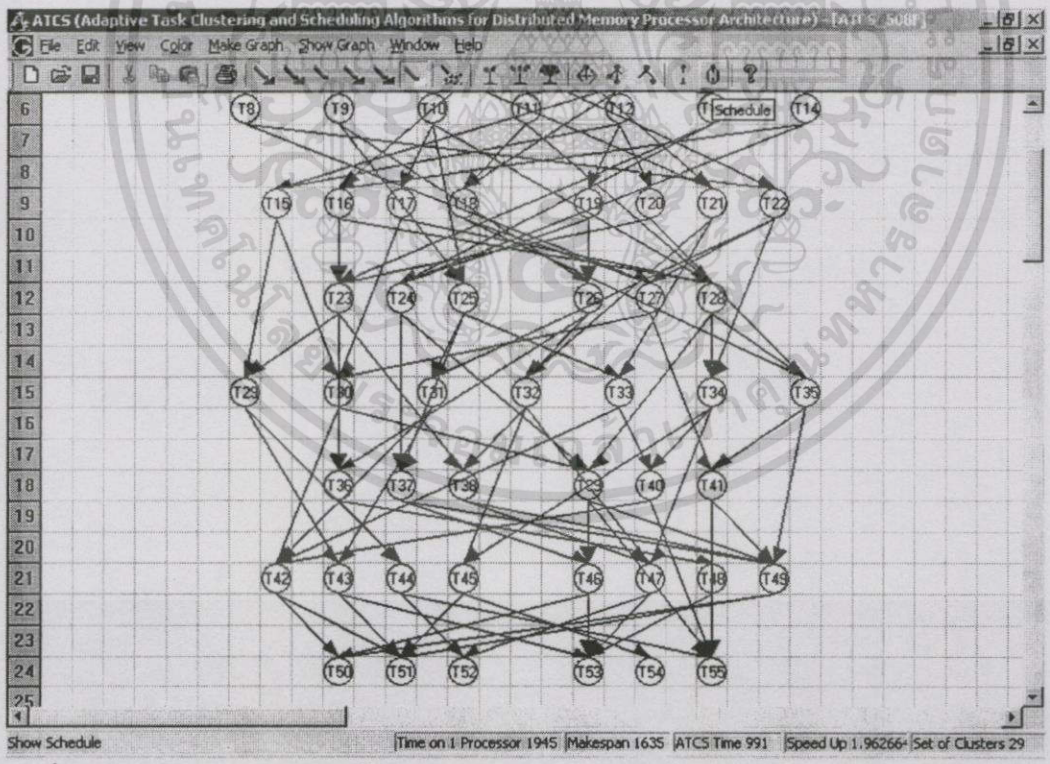
รูปที่ 5.17 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/9



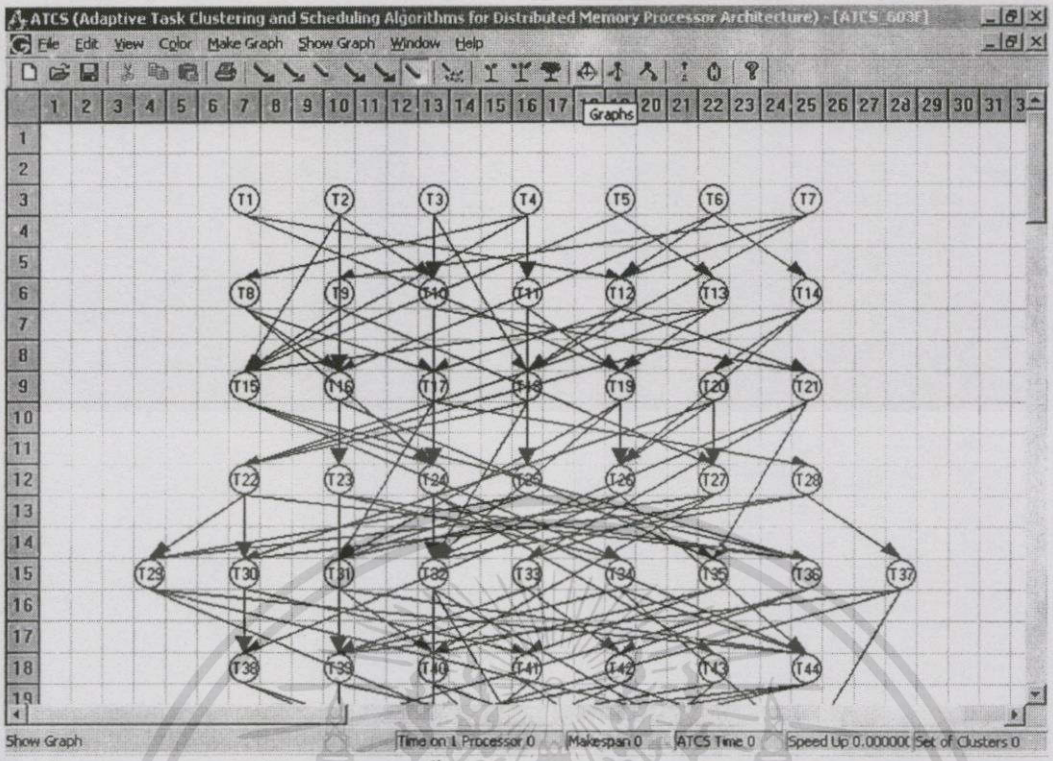
รูปที่ 5.17(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1295- 696 = 599 หน่วย และค่าของ Speed Up เท่ากับ 2.2097 โดยกำหนดให้มี $23 \leq \mu \leq 33$



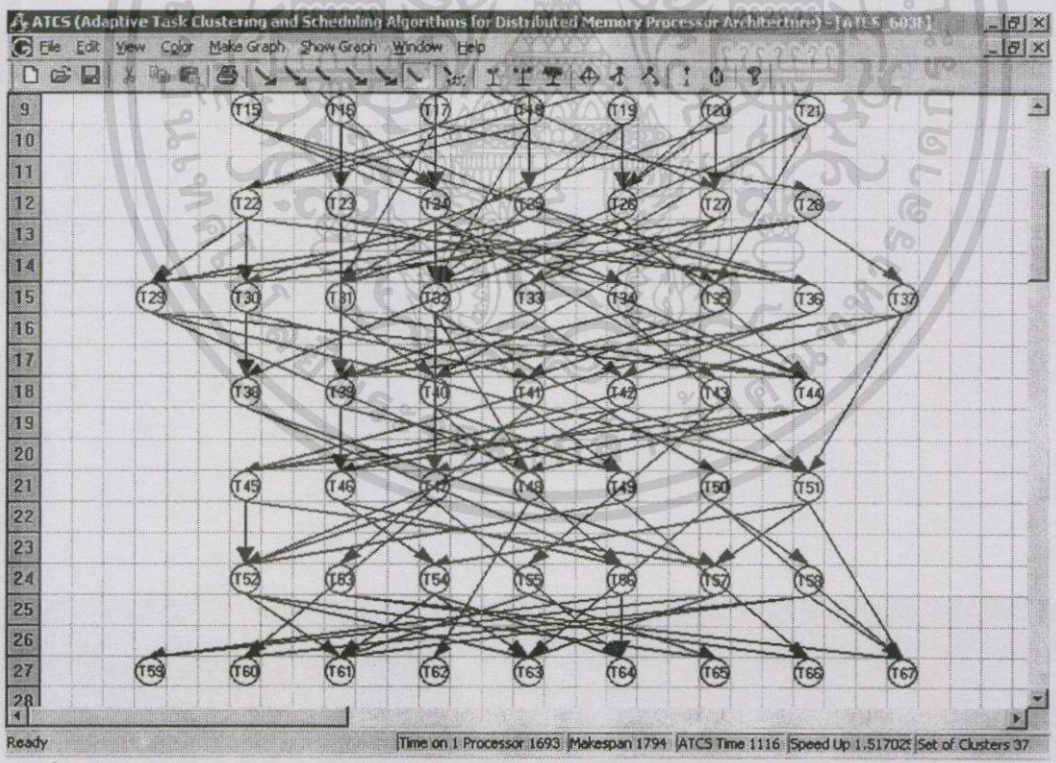
รูปที่ 5.18 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/6



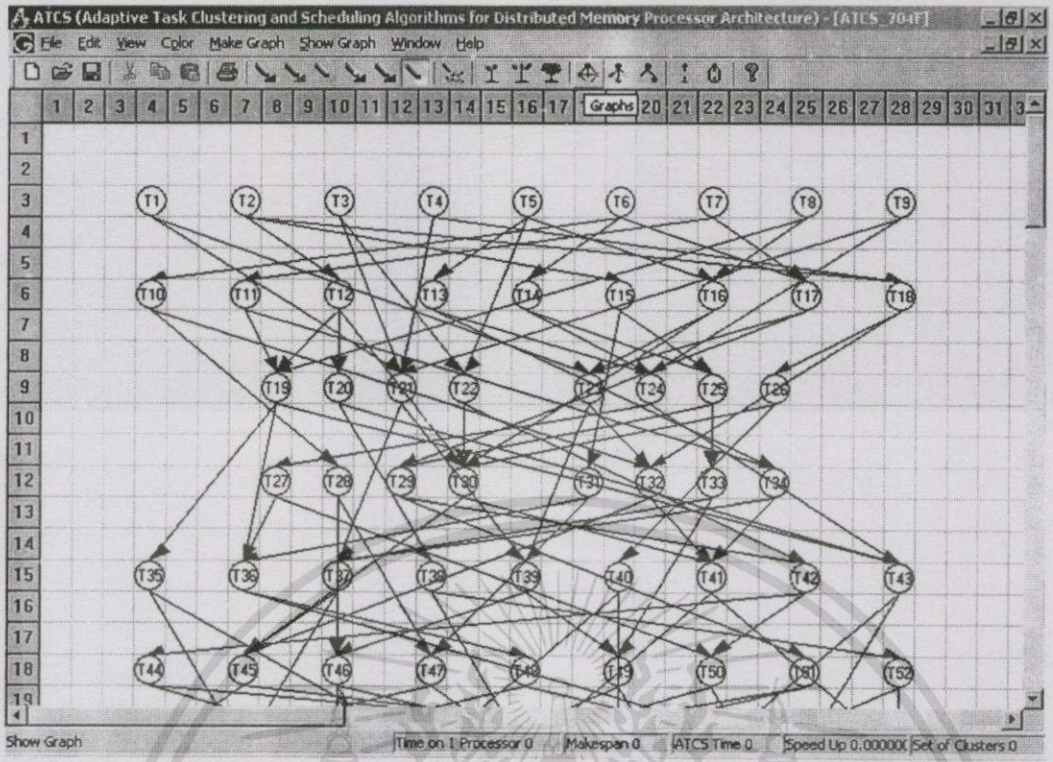
รูปที่ 5.18(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1635-991 = 644 หน่วย และค่าของ Speed Up เท่ากับ 1.9626 โดยกำหนดให้มี $31 \leq \mu \leq 40$



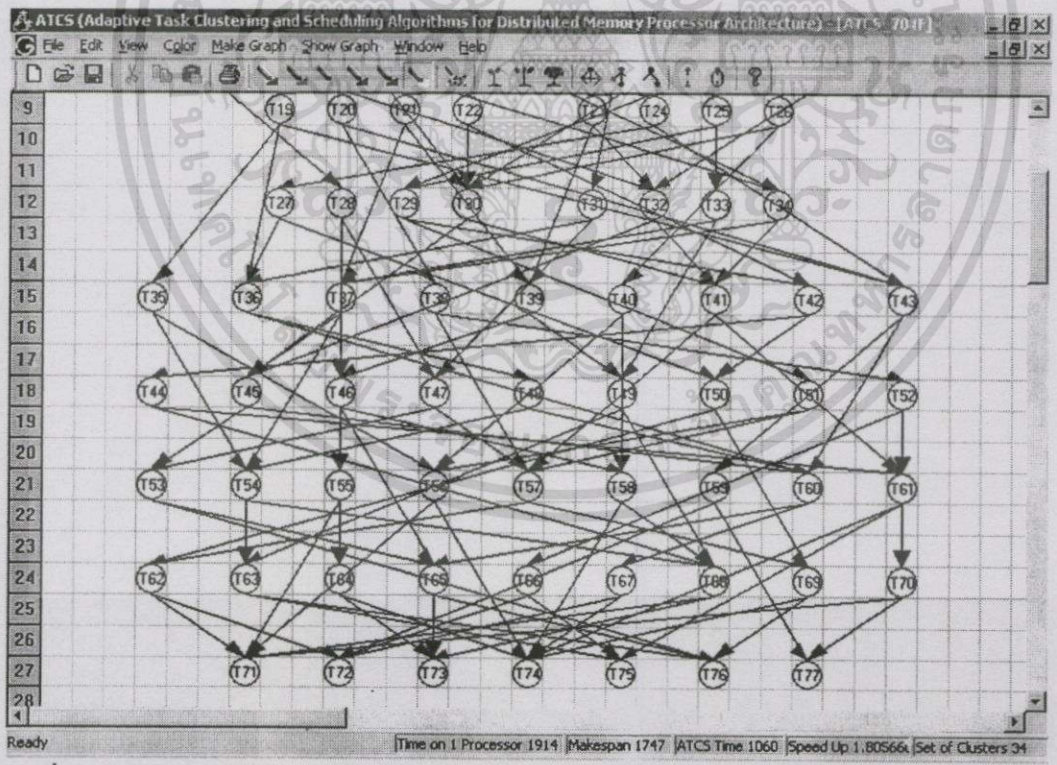
รูปที่ 5.19 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/8



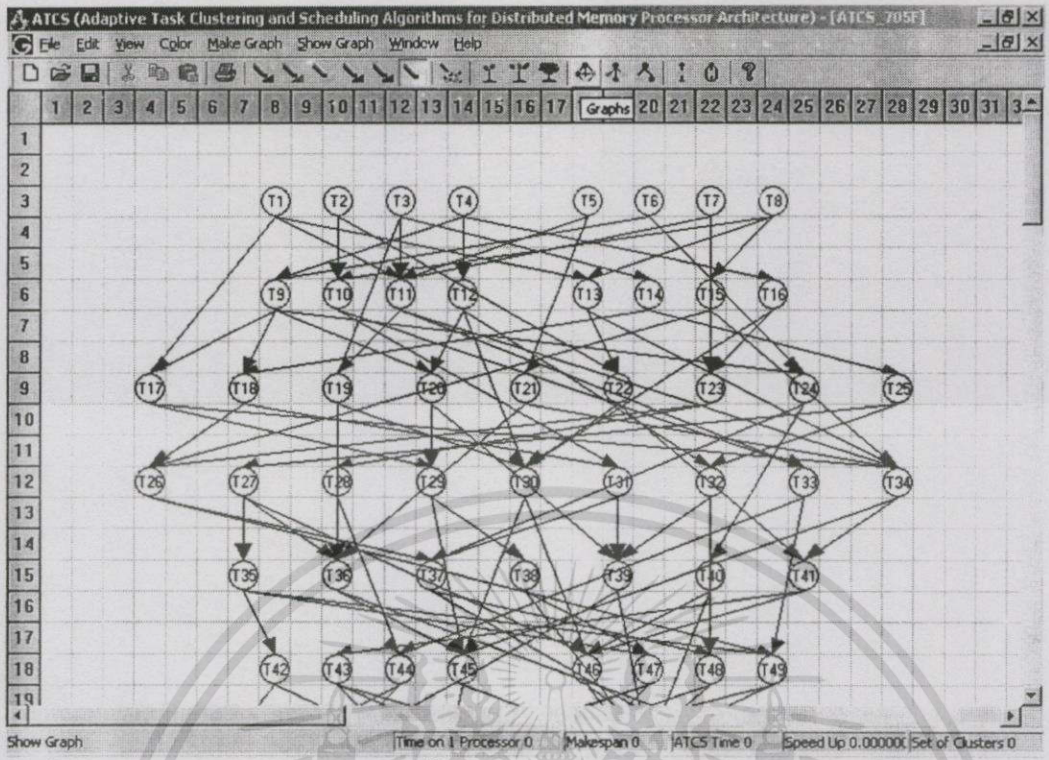
รูปที่ 5.19(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1794-1116 = 678 หน่วย และค่าของ Speed Up เท่ากับ 1.5170 โดยกำหนดให้มี $22 \leq \mu \leq 32$



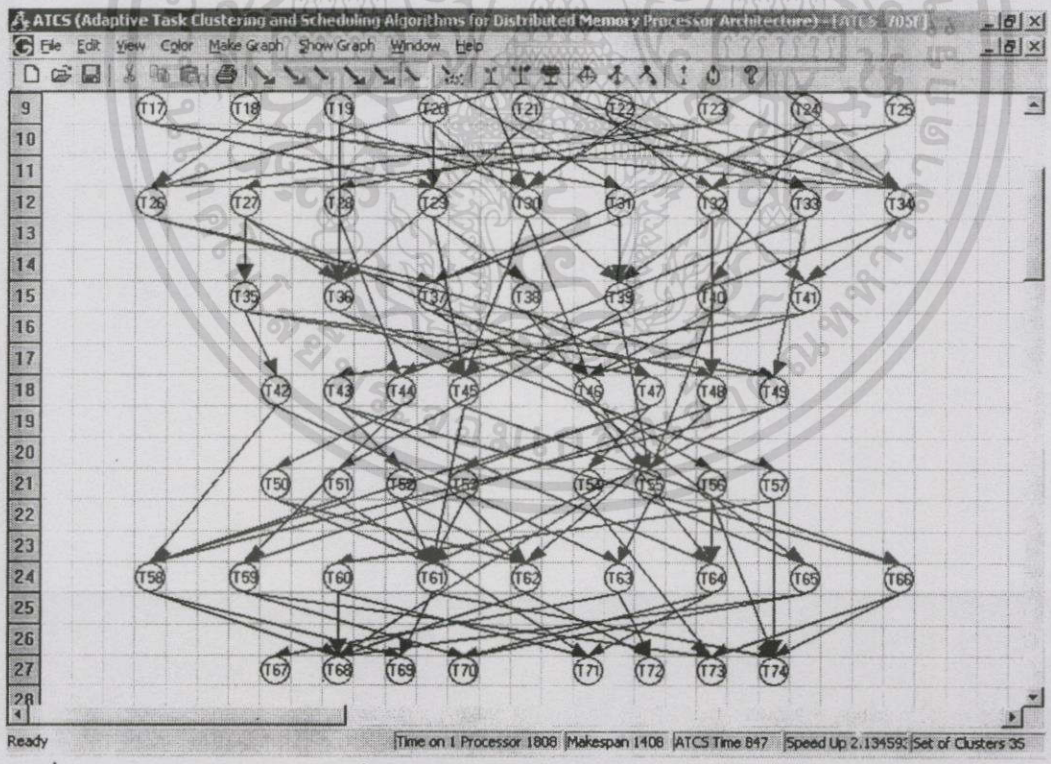
รูปที่ 5.20 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/7



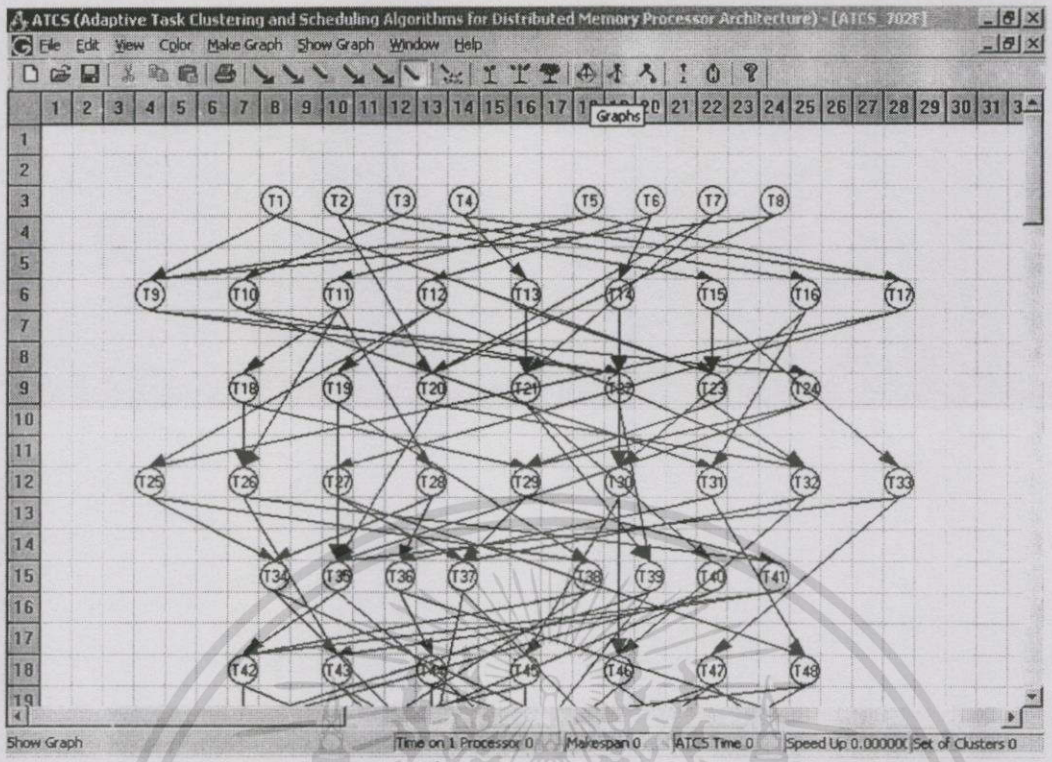
รูปที่ 5.20(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1747 -1060 = 687 หน่วย และค่าของ Speed Up เท่ากับ 1.8056 โดยกำหนดให้มี $20 \leq \mu \leq 30$



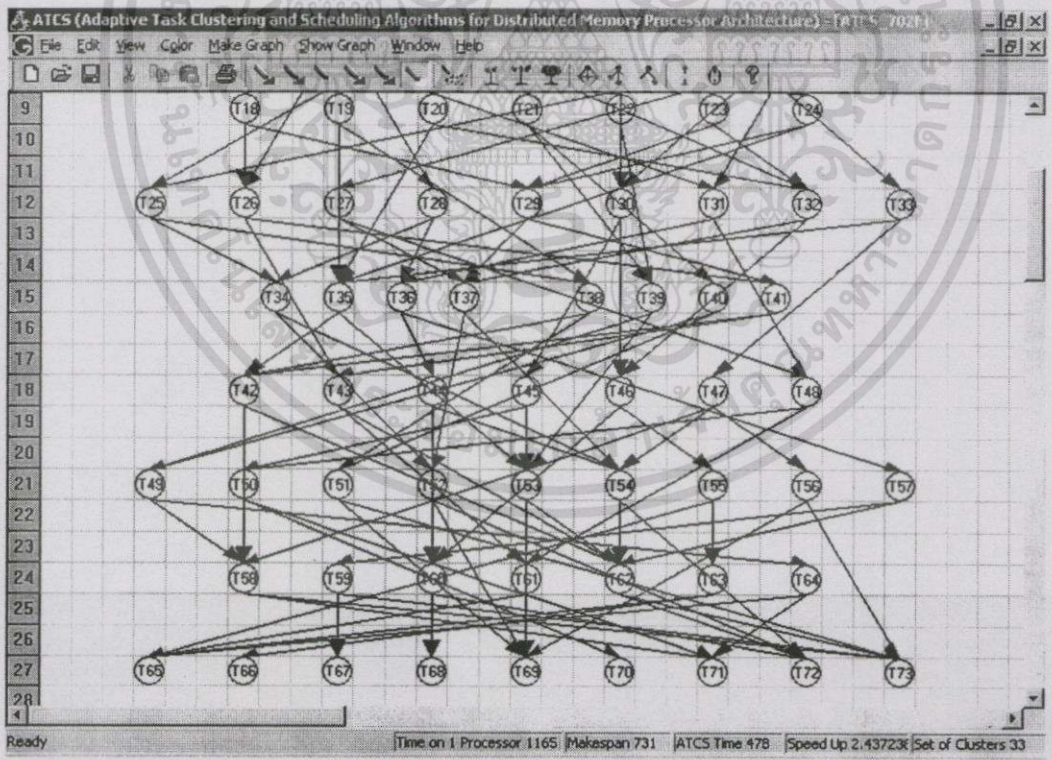
รูปที่ 5.21 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/8



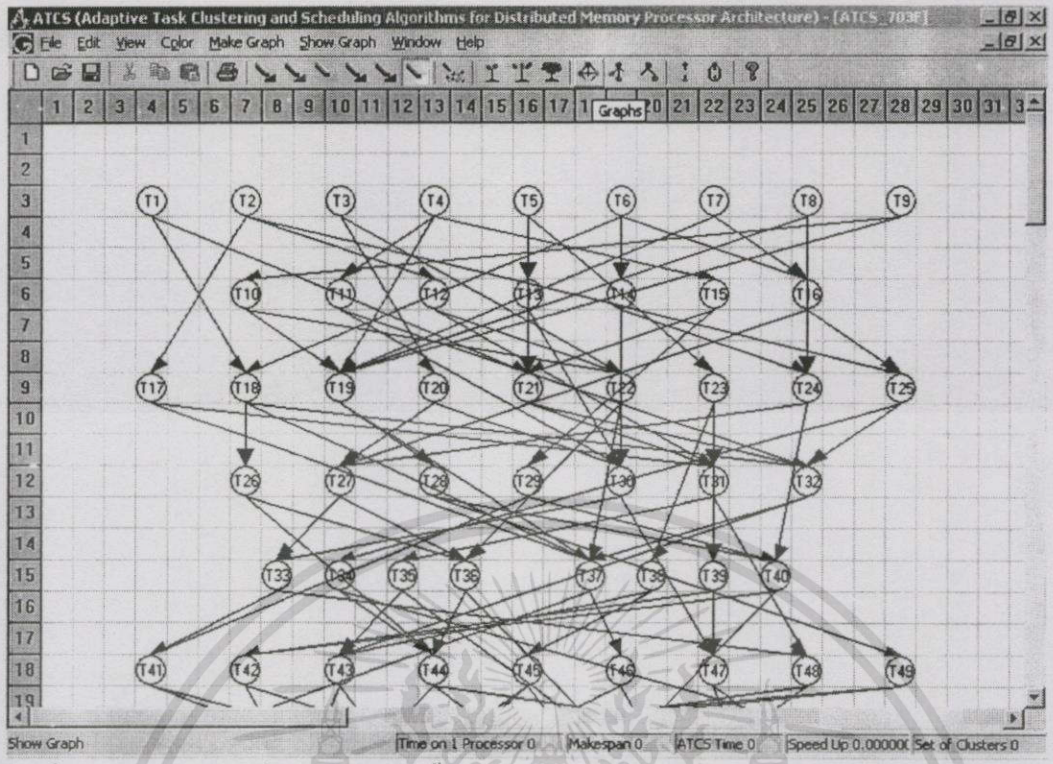
รูปที่ 5.21(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1408-847 = 561 หน่วย และค่าของ Speed Up เท่ากับ 2.1345โดยกำหนดให้มี $21 \leq \mu \leq 30$



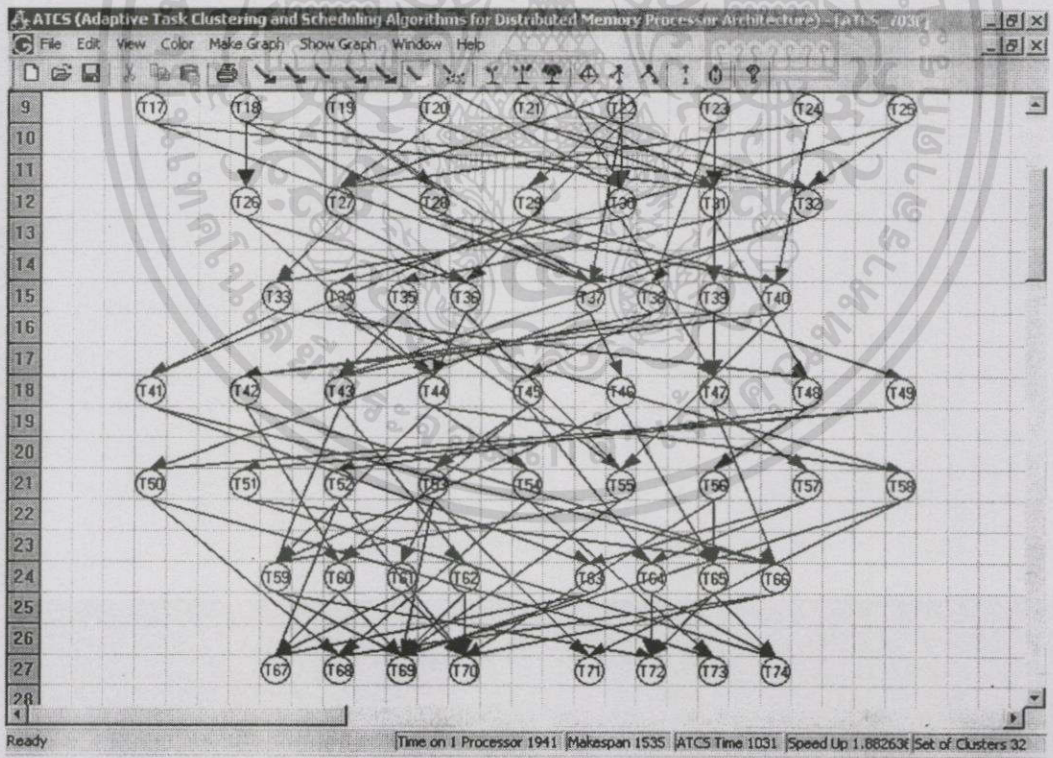
รูปที่ 5.22 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/8



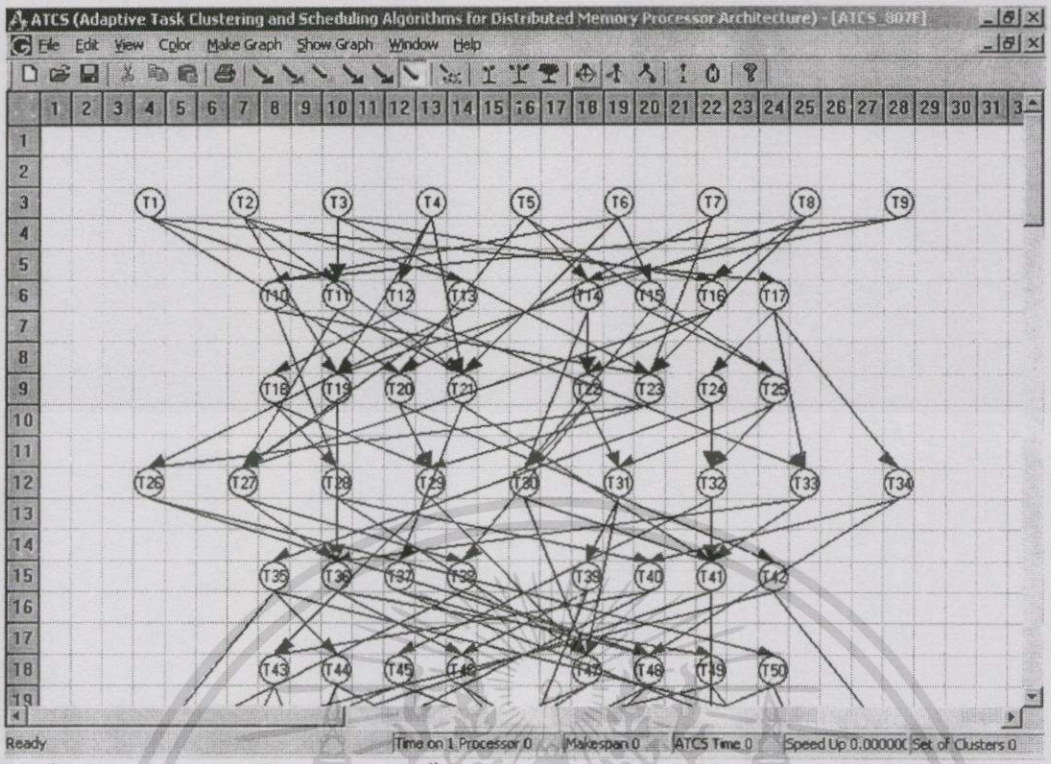
รูปที่ 5.22(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 731-478 = 253 หน่วย และค่าของ Speed Up เท่ากับ 2.4372 โดยกำหนดให้มี $12 \leq \mu \leq 22$



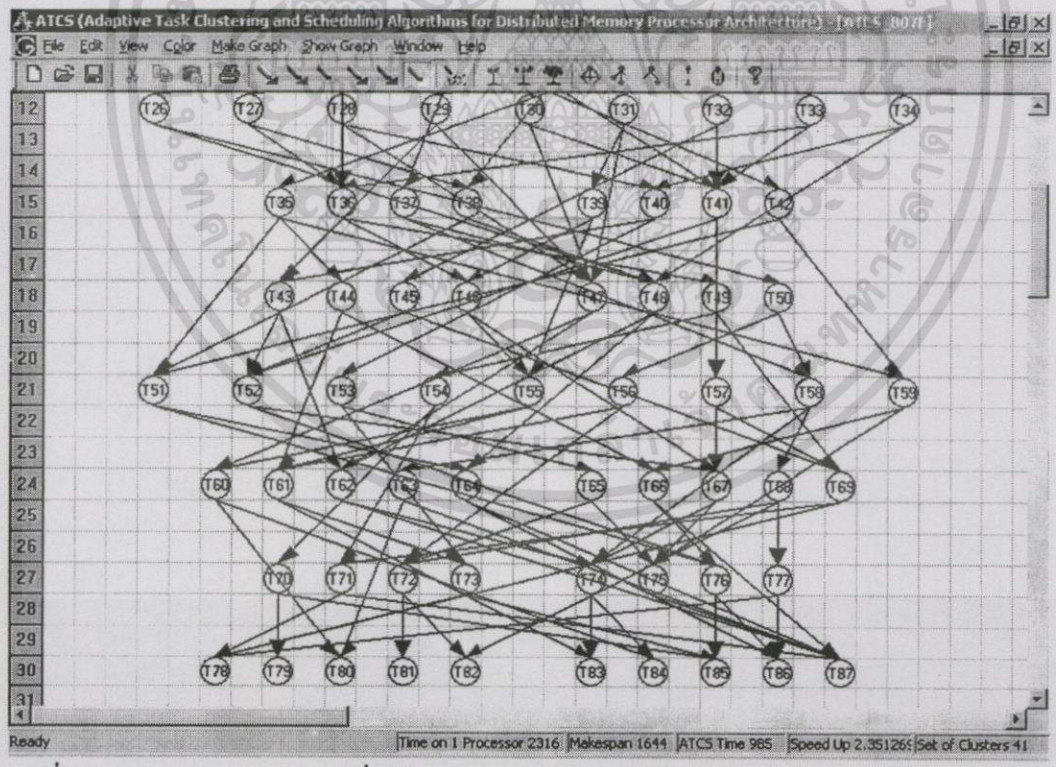
รูปที่ 5.23 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/7



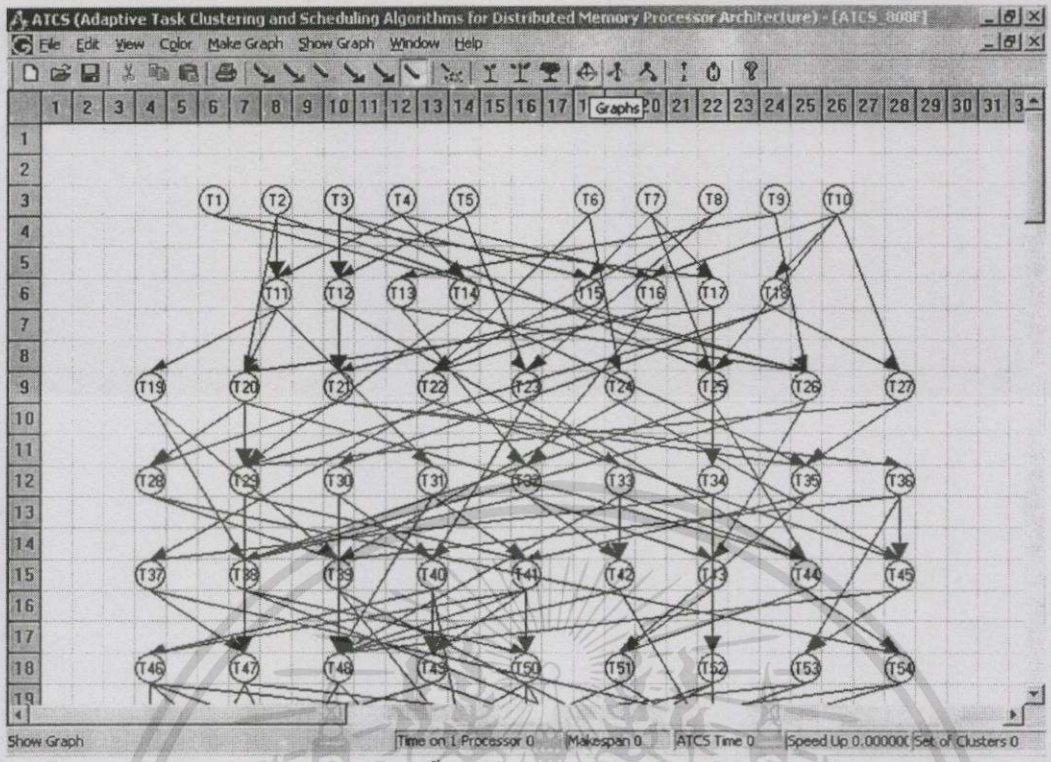
รูปที่ 5.23(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1535 -1031 = 504 หน่วย และค่าของ Speed Up เท่ากับ 1.8826 โดยกำหนดให้มี $22 \leq \mu \leq 34$



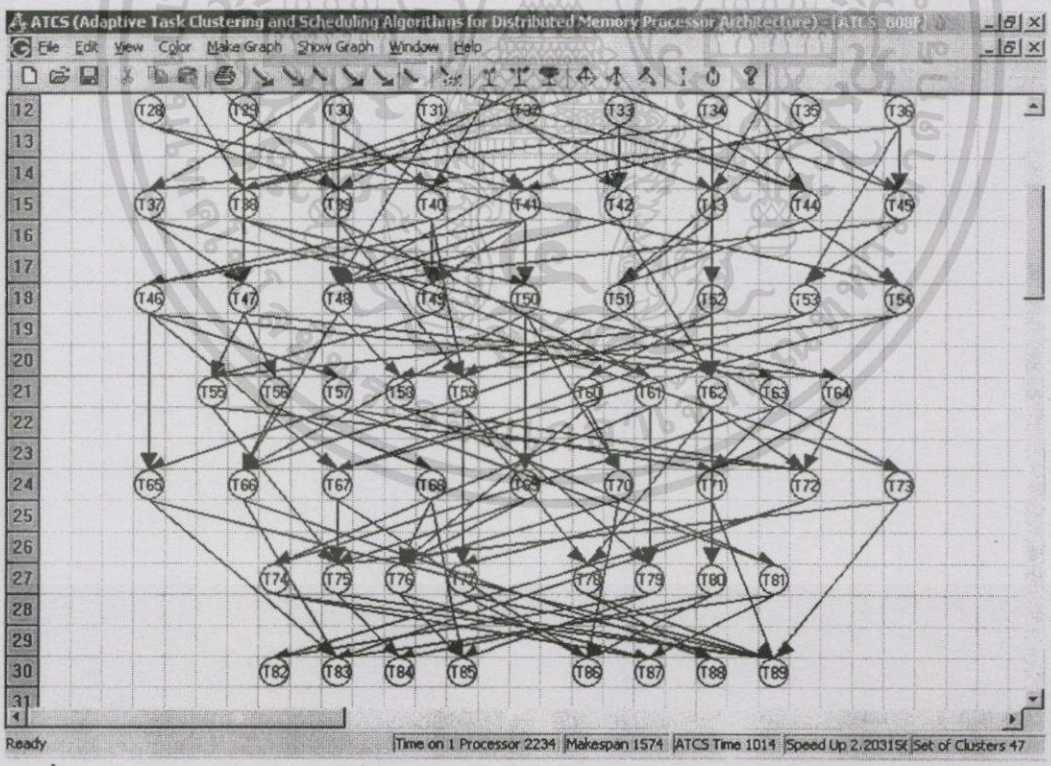
รูปที่ 5.24 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/8



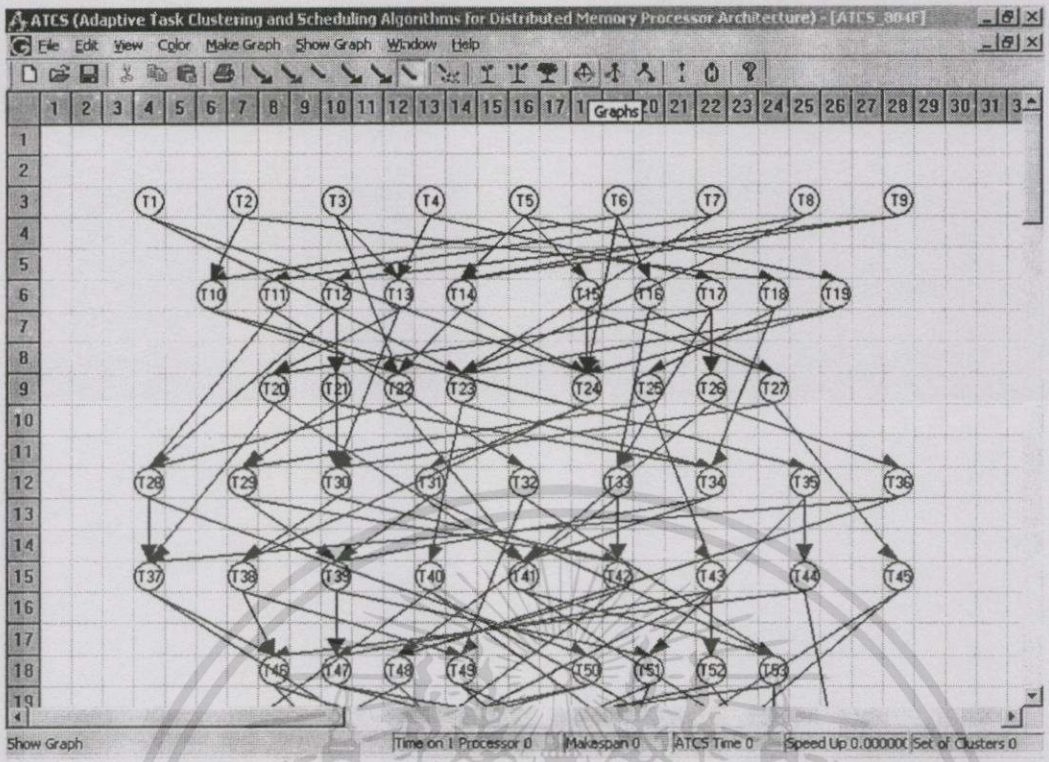
รูปที่ 5.24(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1644-985 = 659 หน่วย และค่าของ Speed Up เท่ากับ 2.3512 โดยกำหนดให้มี $23 \leq \mu \leq 30$



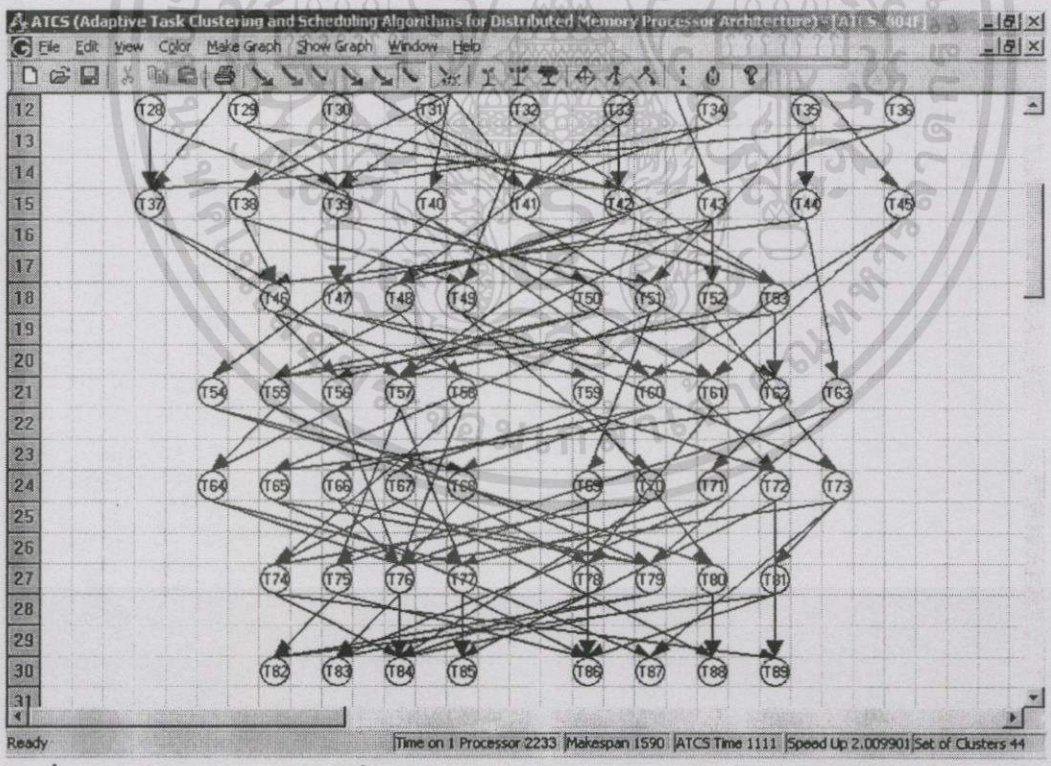
รูปที่ 5.25 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/7



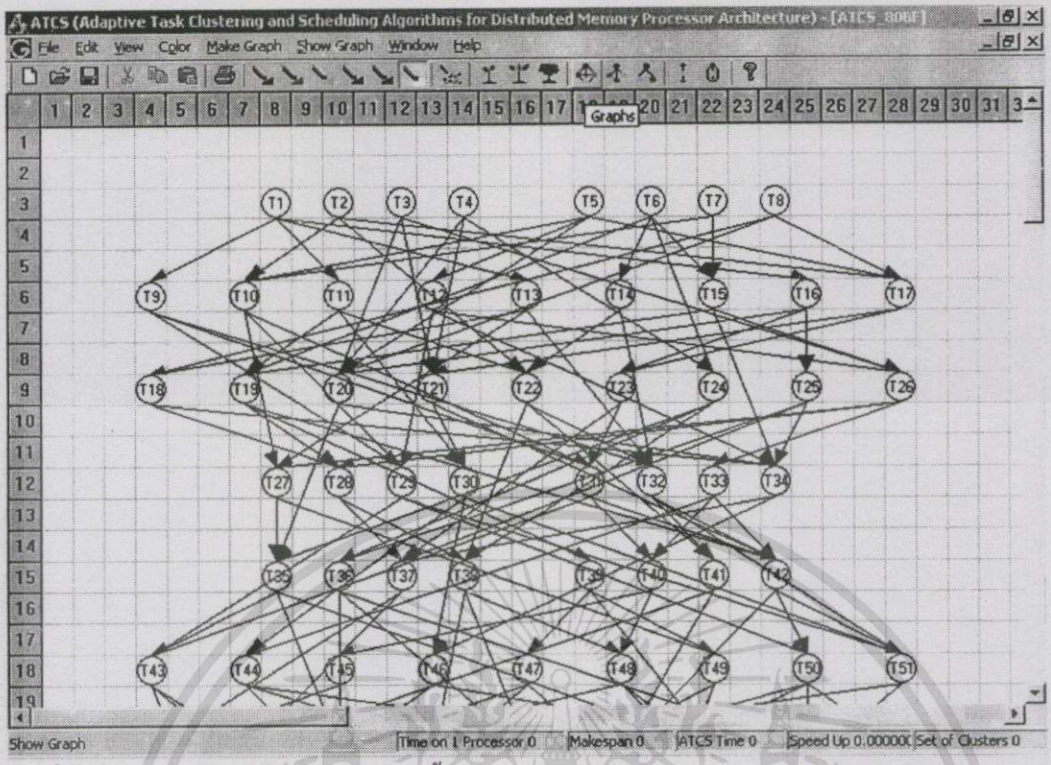
รูปที่ 5.25(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1574-1014 = 560 หน่วย และค่าของ Speed Up เท่ากับ 2.2031 โดยกำหนดให้มี $23 \leq \mu \leq 33$



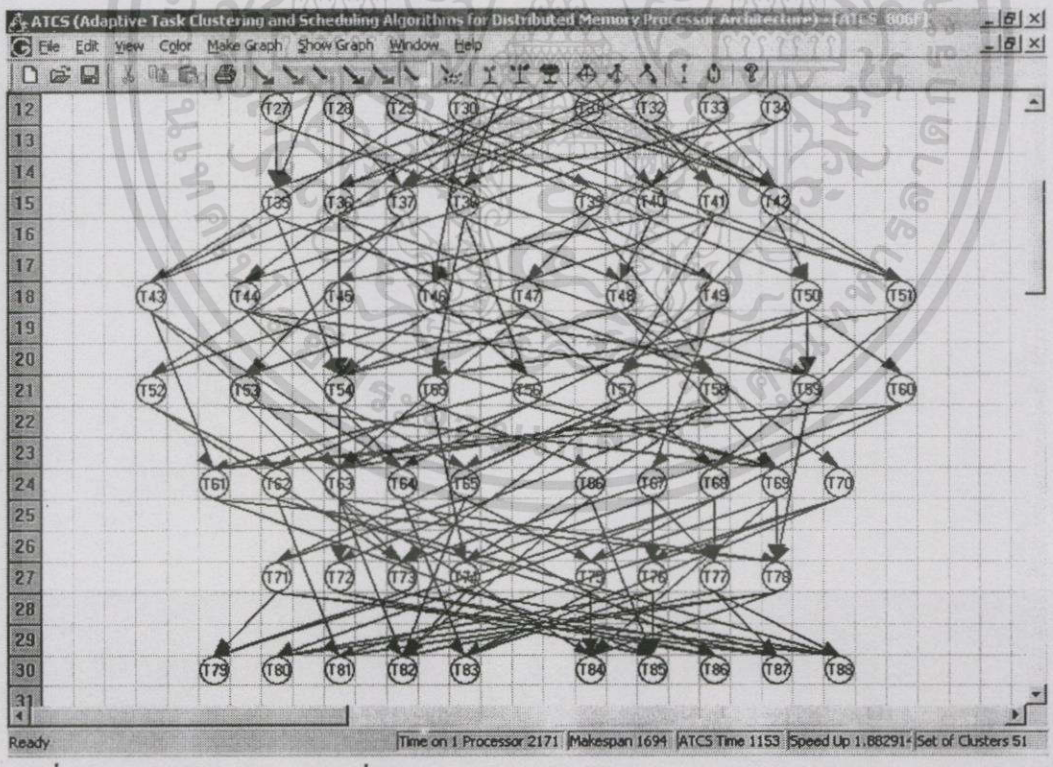
รูปที่ 5.26 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/10



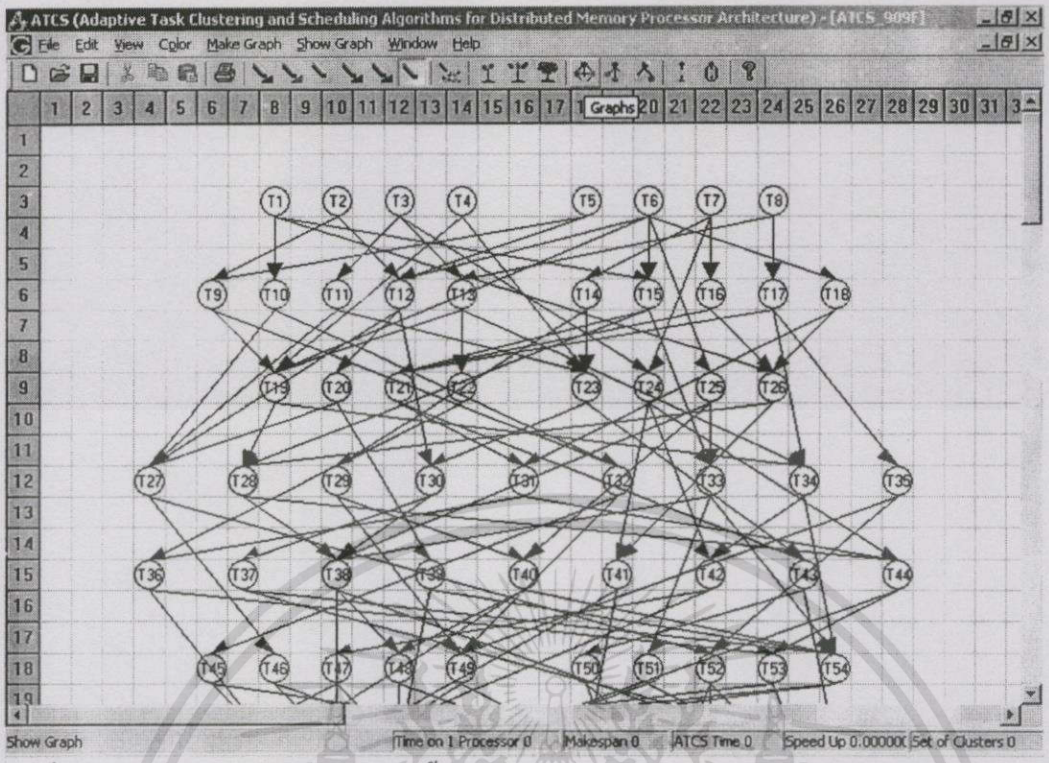
รูปที่ 5.26(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1590-1111 = 479 หน่วย และค่าของ Speed Up เท่ากับ 2.0099 โดยกำหนดให้มี $22 \leq \mu \leq 29$



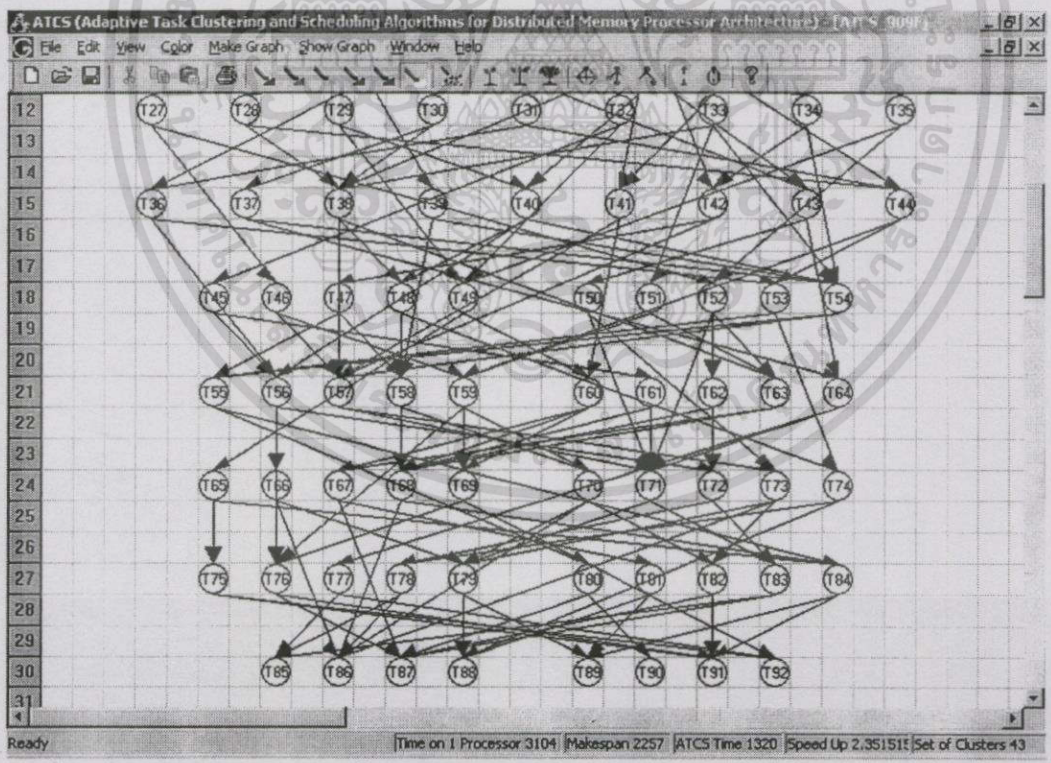
รูปที่ 5.27 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/9



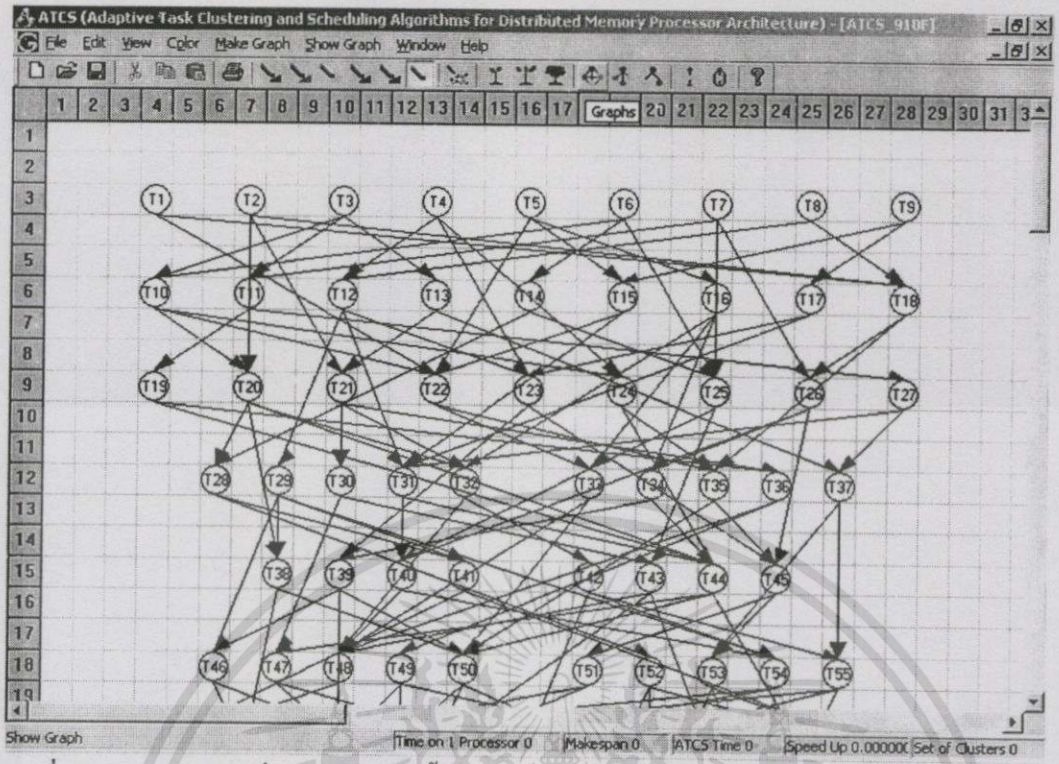
รูปที่ 5.27(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1694-1153 = 541 หน่วย และค่าของ Speed Up เท่ากับ 1.8829 โดยกำหนดให้มี $22 \leq \mu \leq 30$



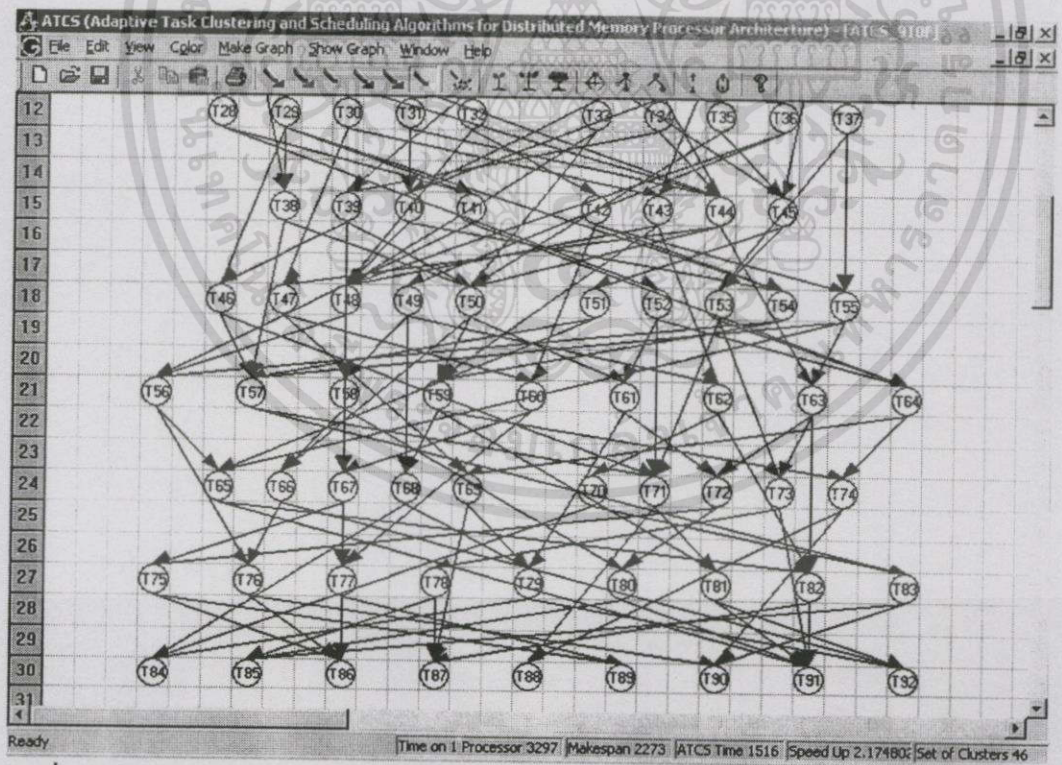
รูปที่ 5.28 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/7



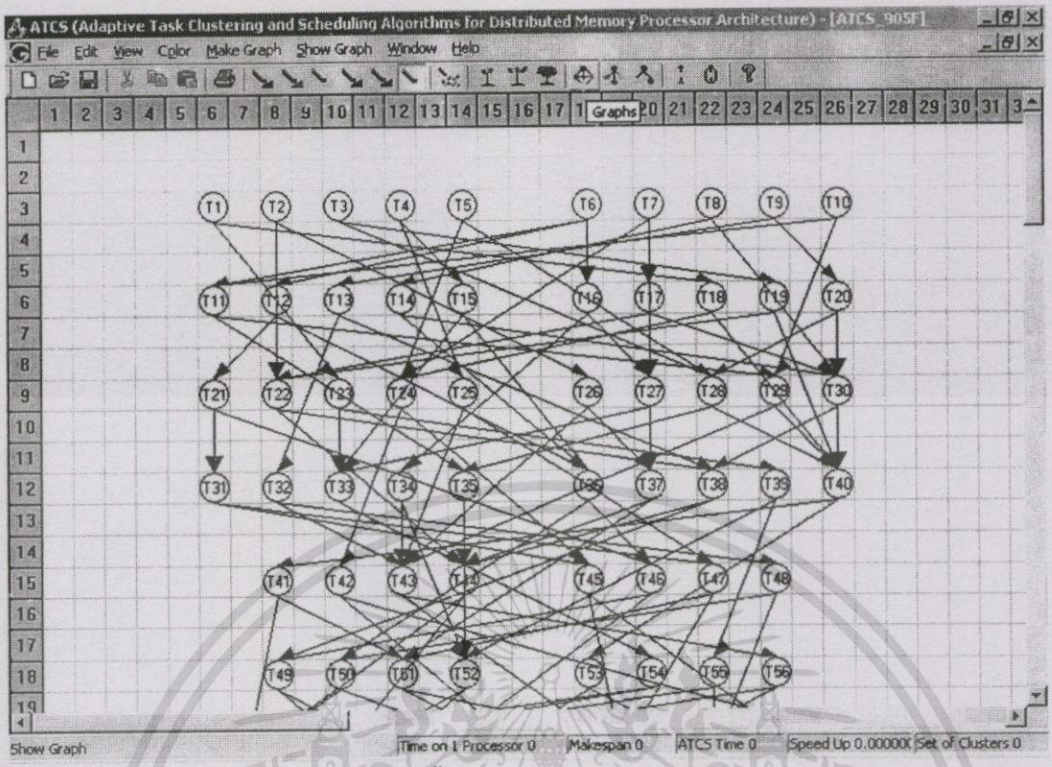
รูปที่ 5.28(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 2257-1320 = 937 หน่วย และค่าของ Speed Up เท่ากับ 2.3515 โดยกำหนดให้มี $31 \leq \mu \leq 39$



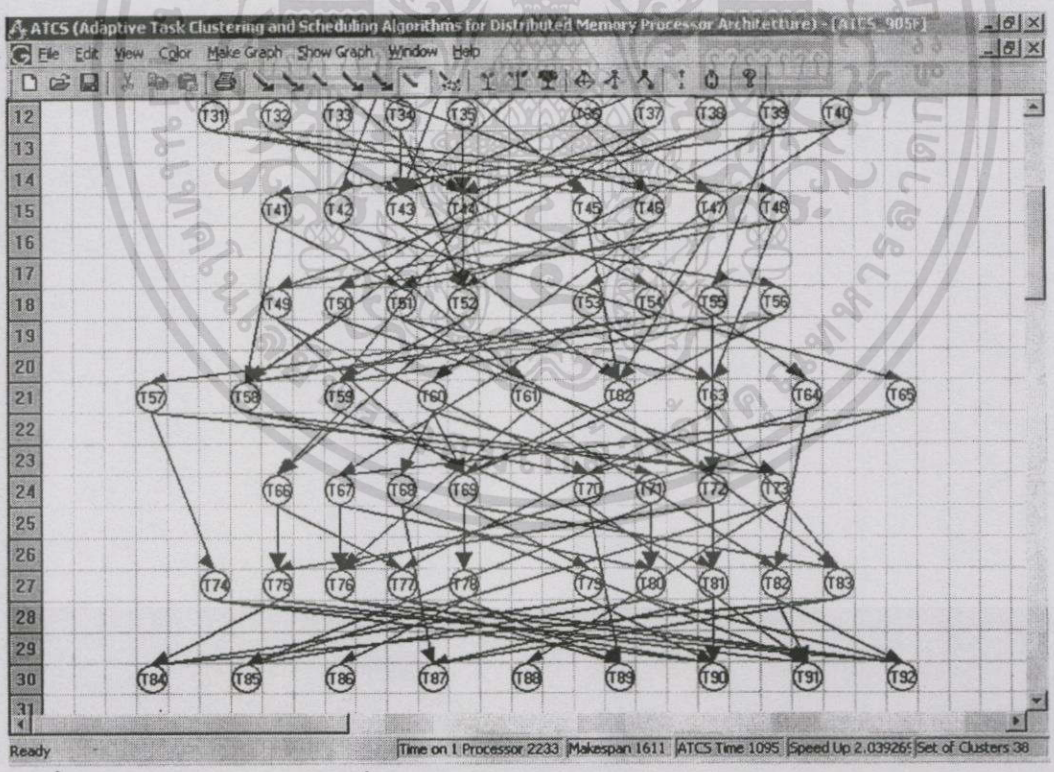
รูปที่ 5.29 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/7



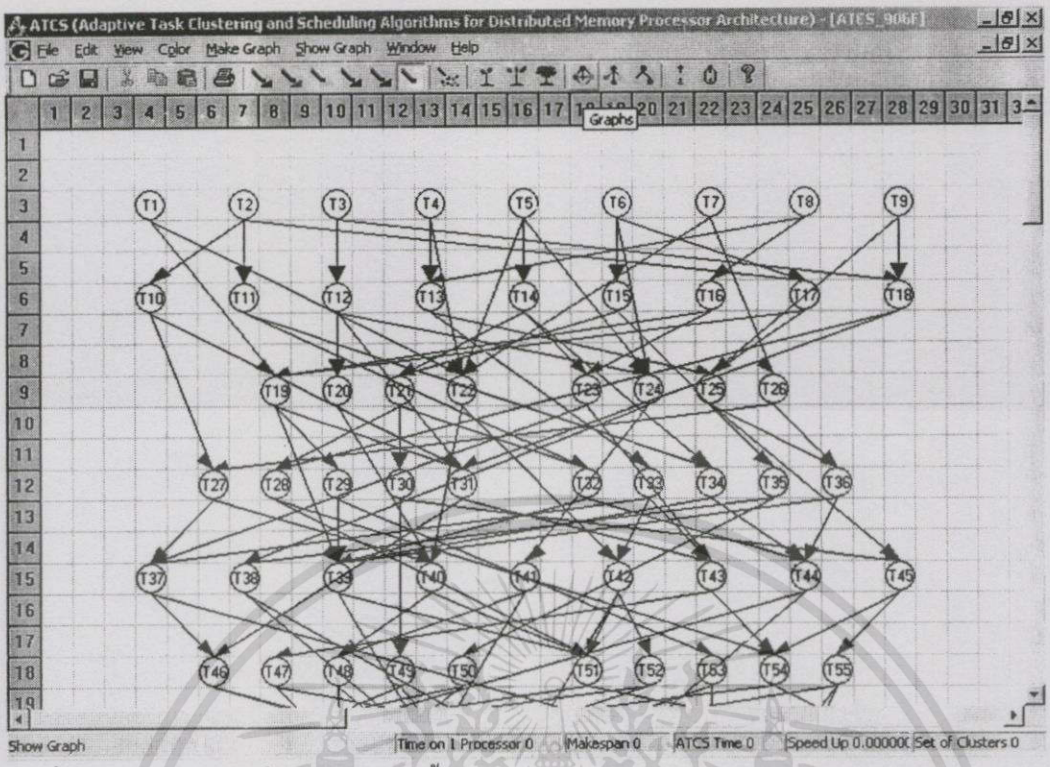
รูปที่ 5.29(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 2273-1516 = 757 หน่วย และค่าของ Speed Up เท่ากับ 2.1748 โดยกำหนดให้มี $32 \leq \mu \leq 40$



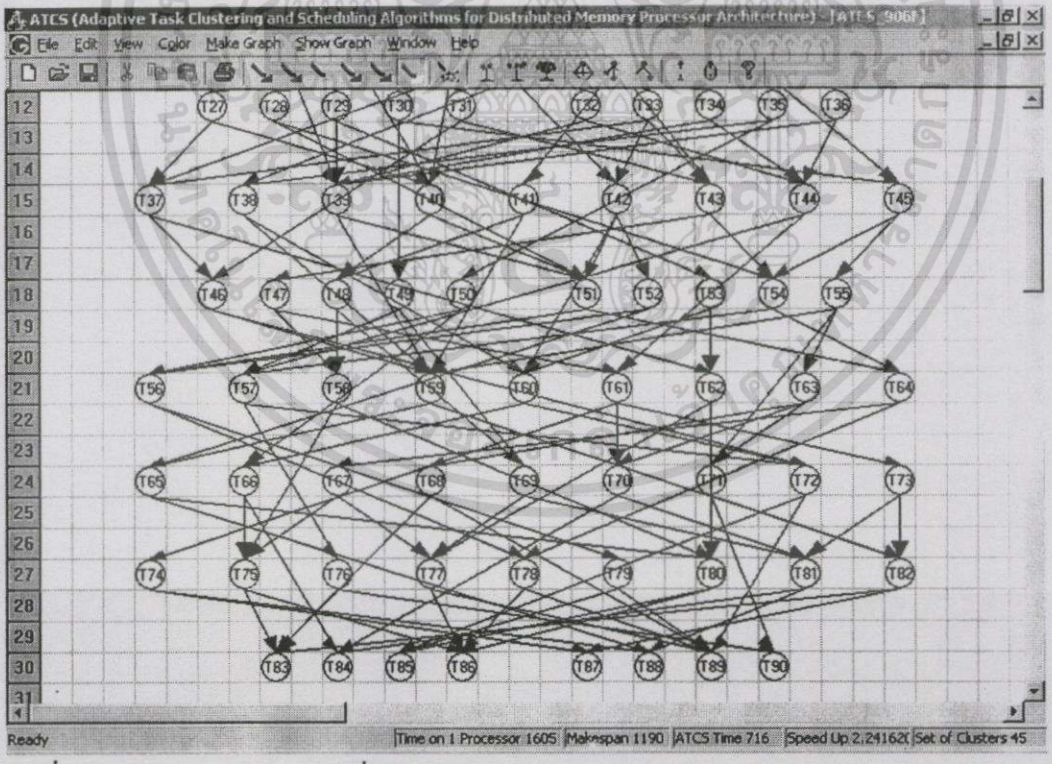
รูปที่ 5.30 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/8



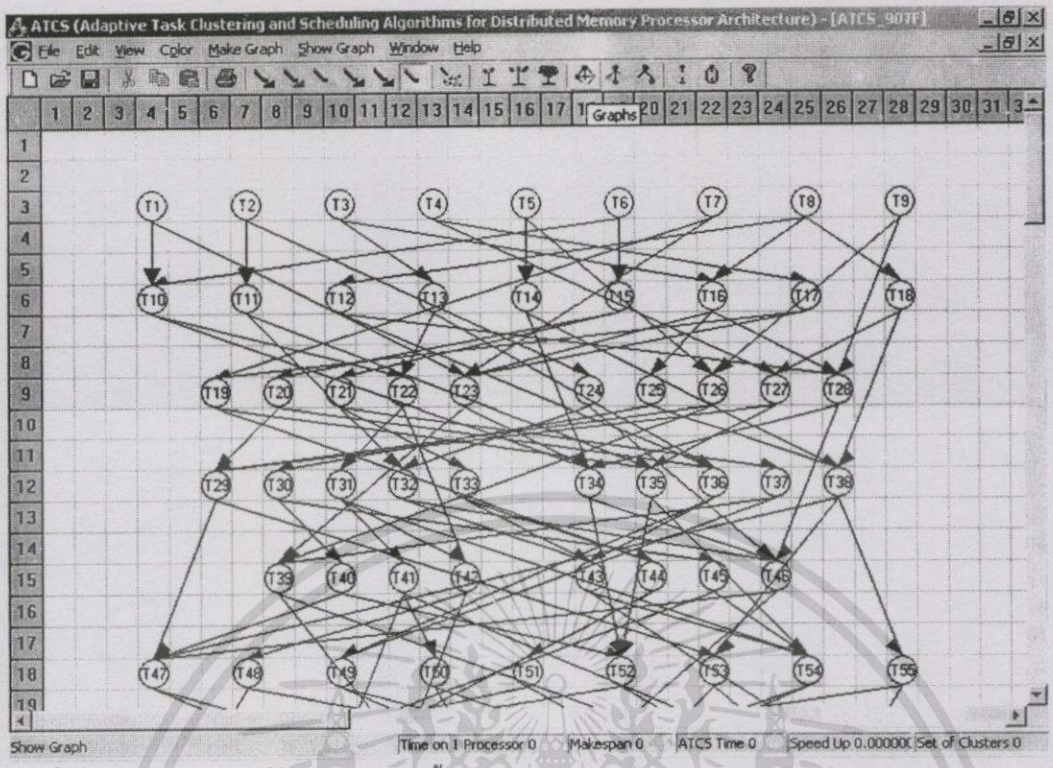
รูปที่ 5.30(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1611-1095 = 516 หน่วย และค่าของ Speed Up เท่ากับ 2.0392 โดยกำหนดให้มี $22 \leq \mu \leq 30$



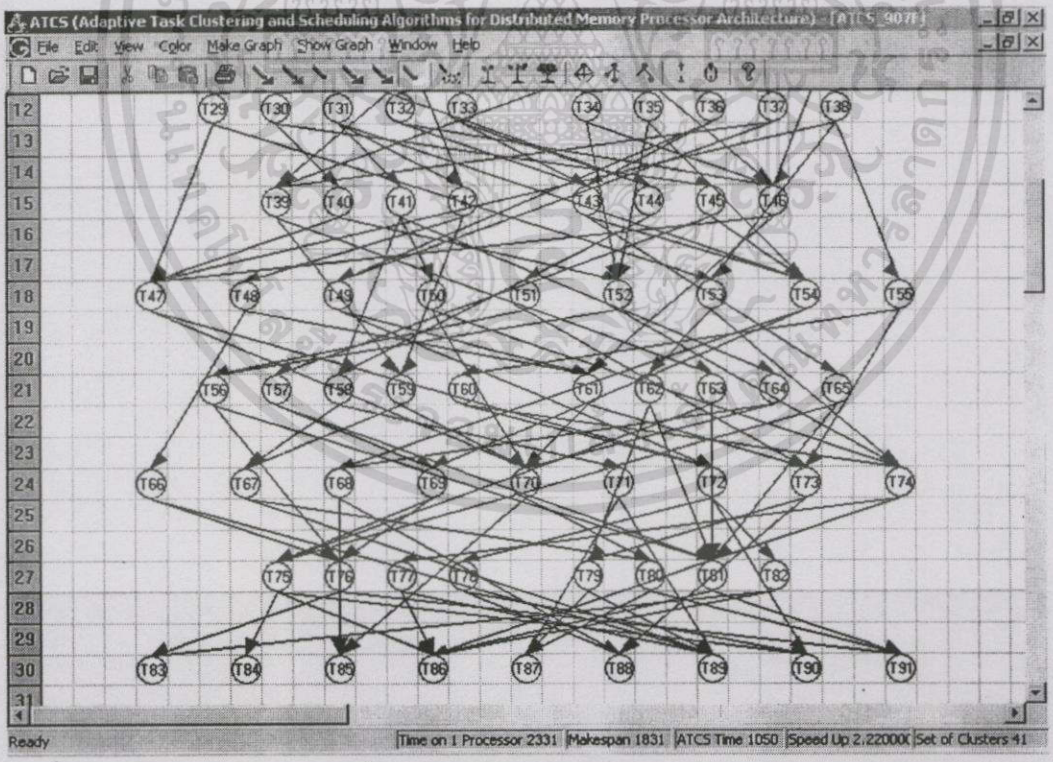
รูปที่ 5.31 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/8



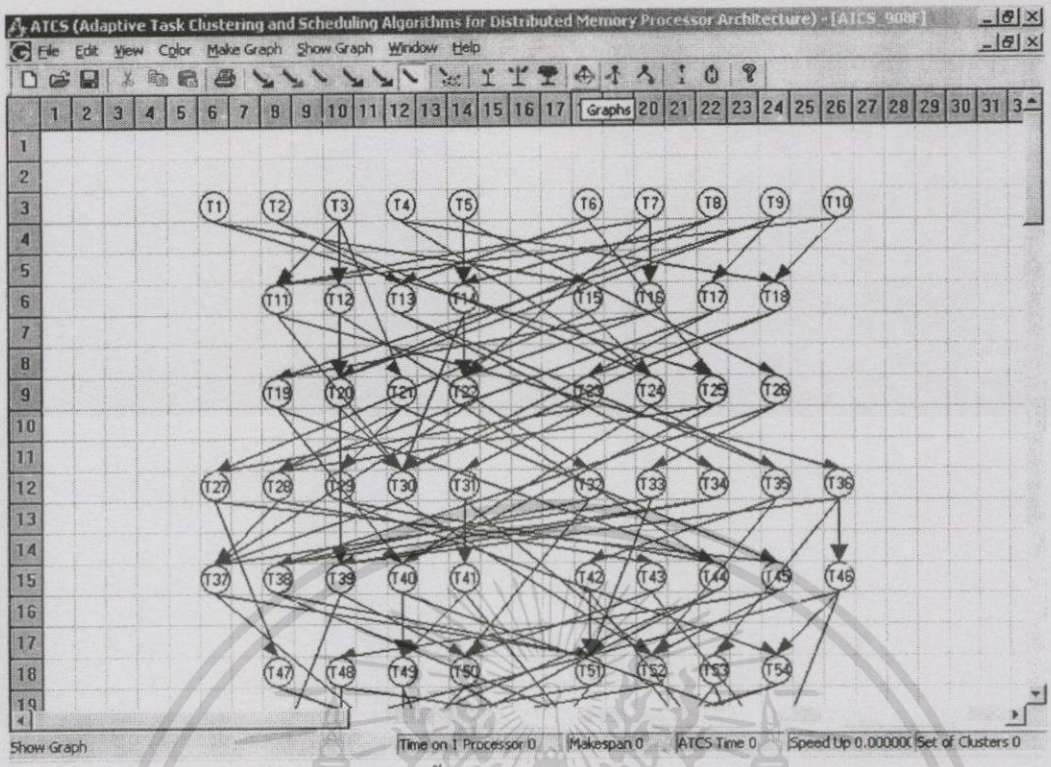
รูปที่ 5.31(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1190-716 = 474 หน่วย และค่าของ Speed Up เท่ากับ 2.2416 โดยกำหนดให้มี $15 \leq \mu \leq 22$



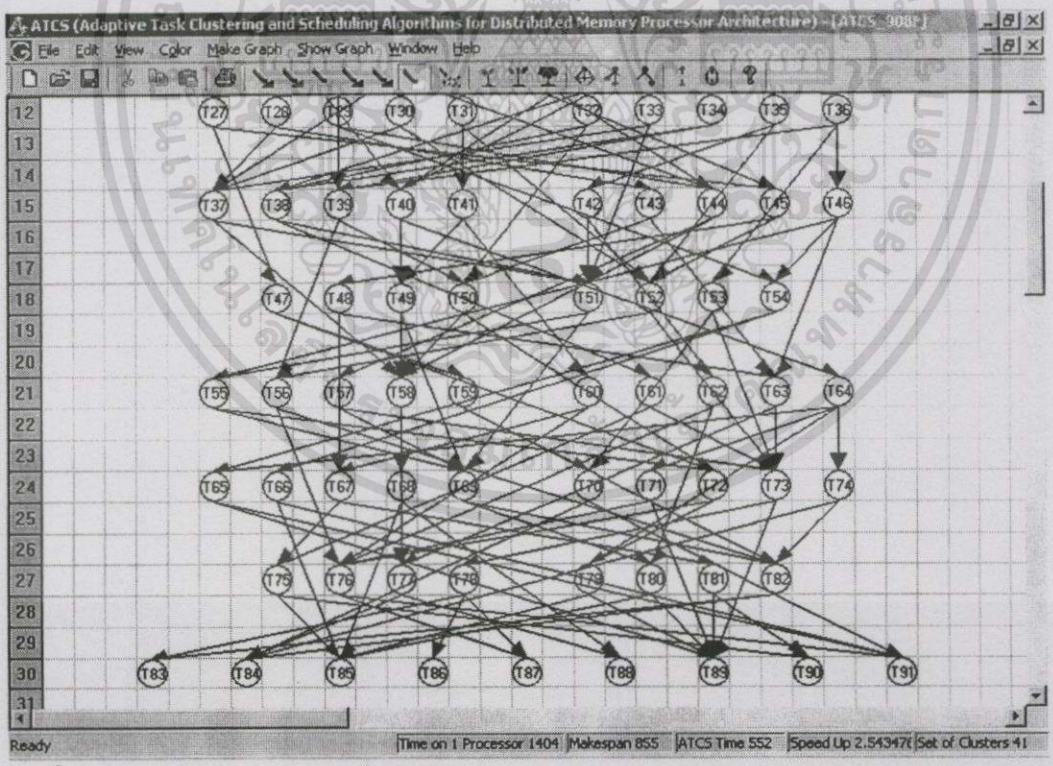
รูปที่ 5.32 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/9



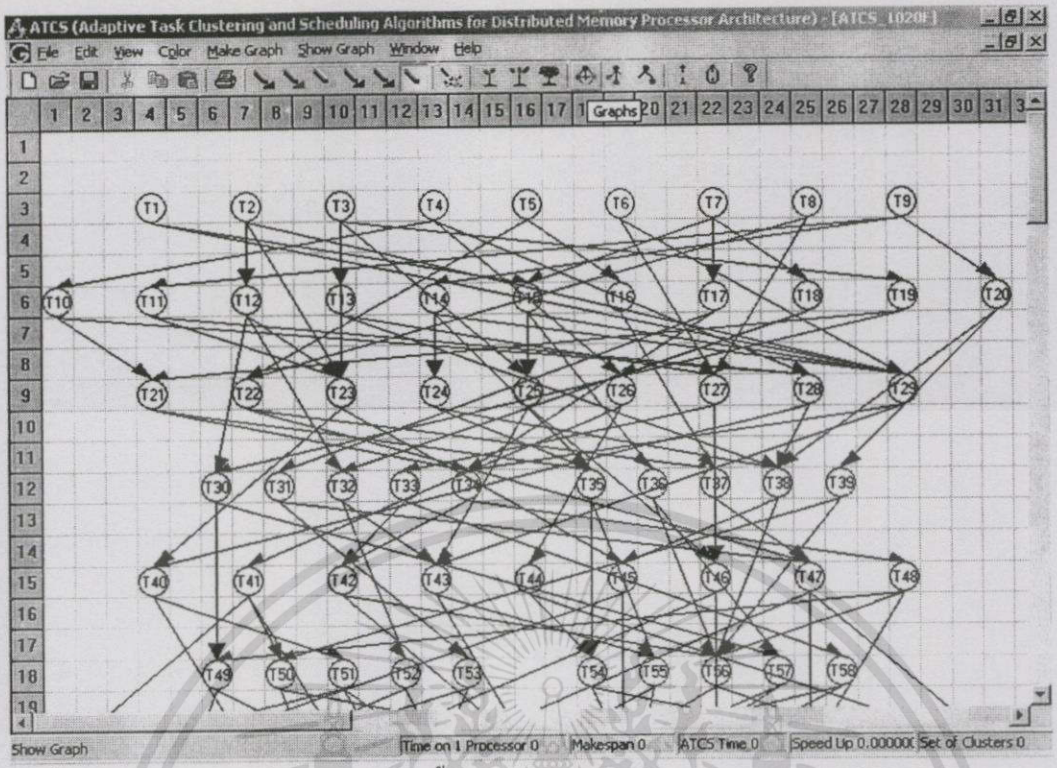
รูปที่ 5.32(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1831-1050 = 781 หน่วย และค่าของ Speed Up เท่ากับ 2.22 โดยกำหนดให้มี $20 \leq \mu \leq 33$



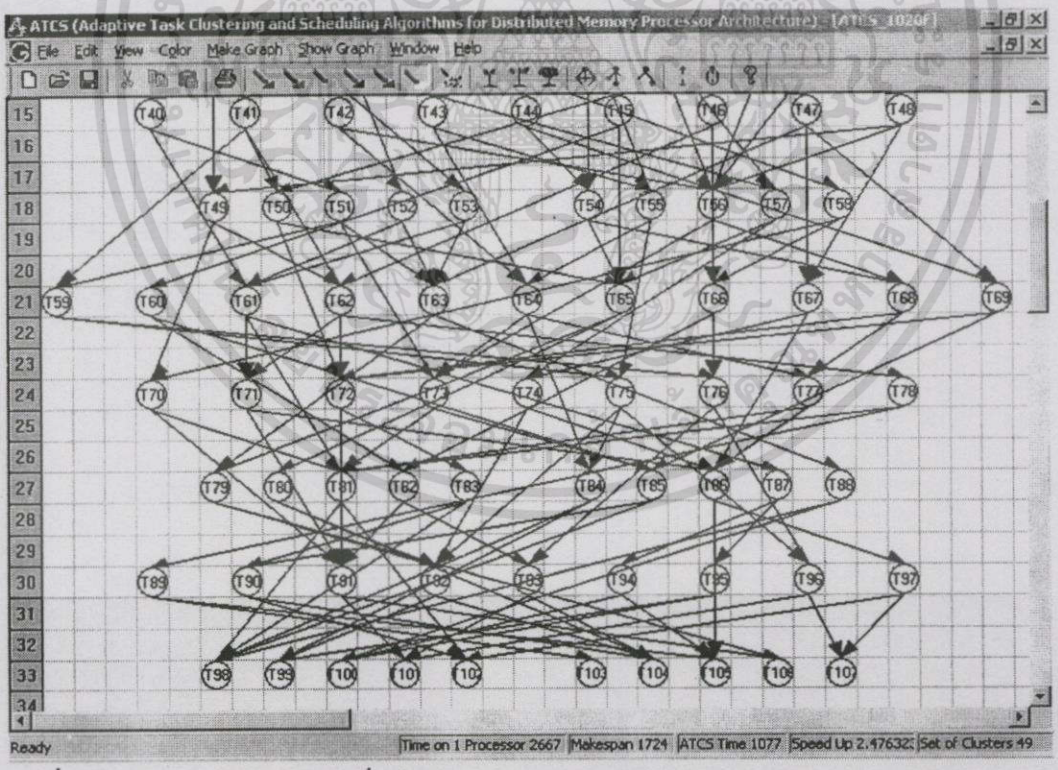
รูปที่ 5.33 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/7



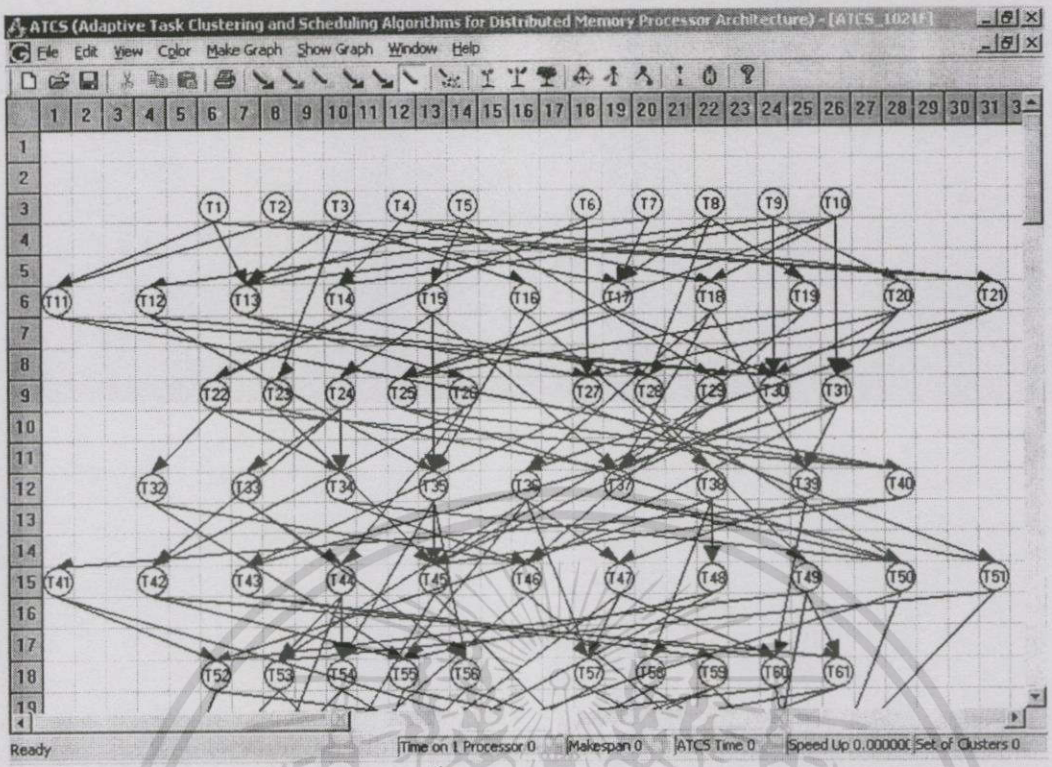
รูปที่ 5.33(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 855-552 = 303 หน่วย และค่าของ Speed Up เท่ากับ 2.5434 โดยกำหนดให้มี $10 \leq \mu \leq 20$



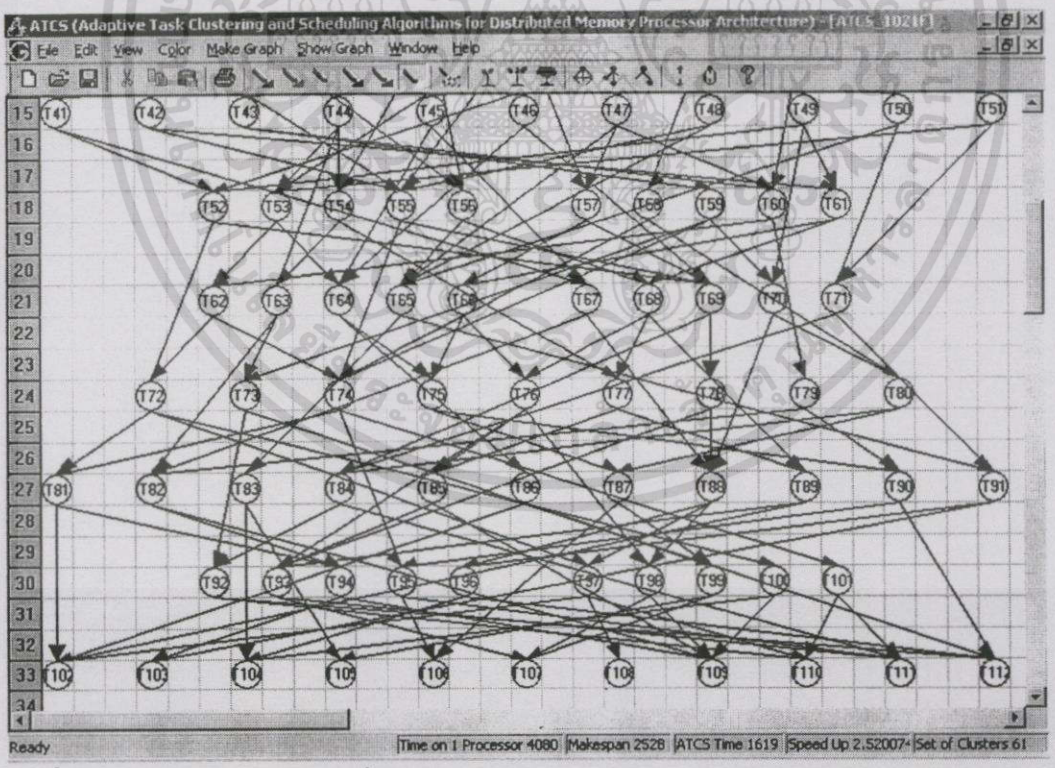
รูปที่ 5.34 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/7



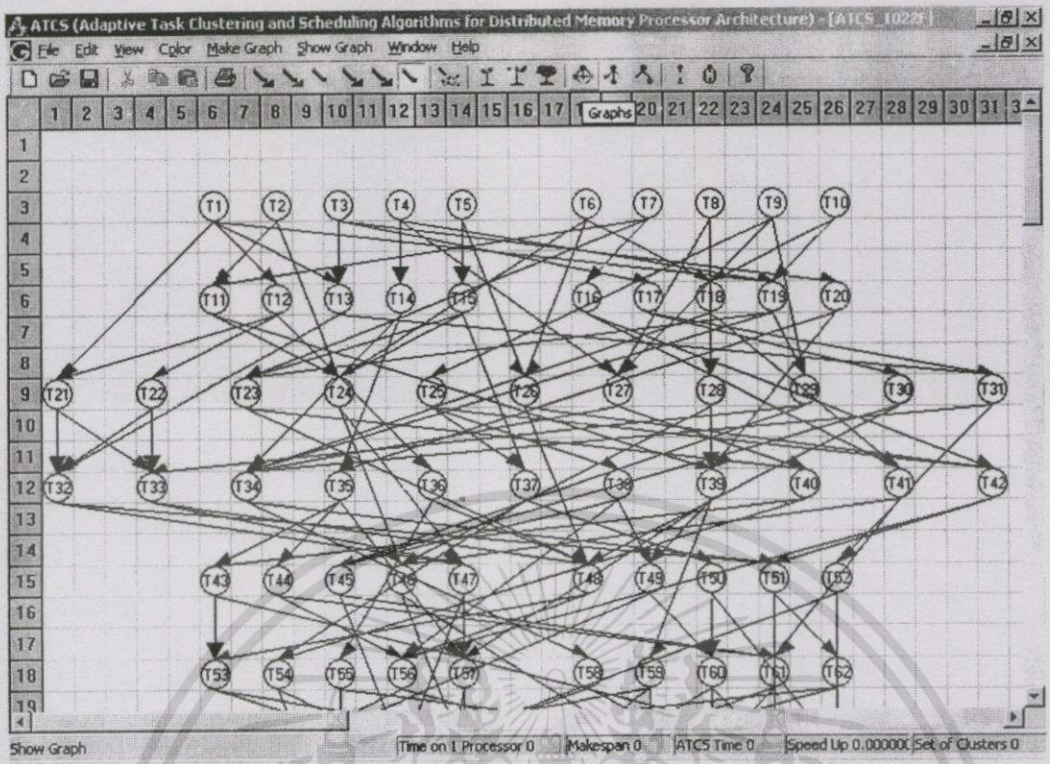
รูปที่ 5.34(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1724-1077 = 647 หน่วย และค่าของ Speed Up เท่ากับ 2.4763 โดยกำหนดให้มี $22 \leq \mu \leq 29$



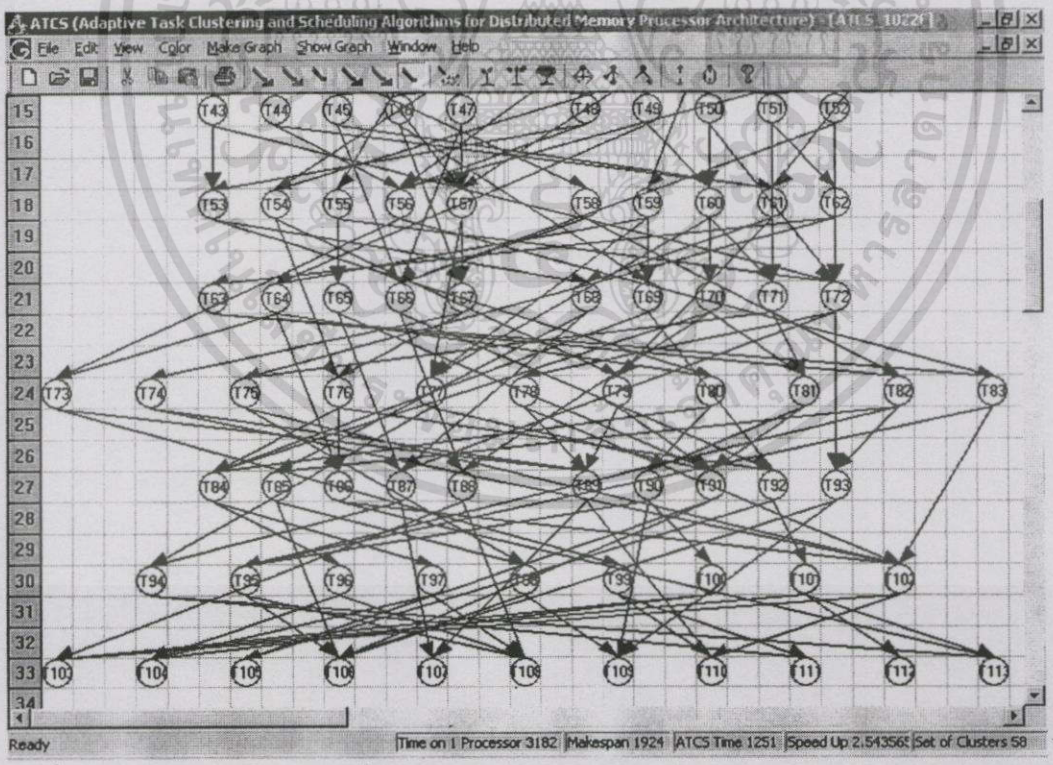
รูปที่ 5.35 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/7



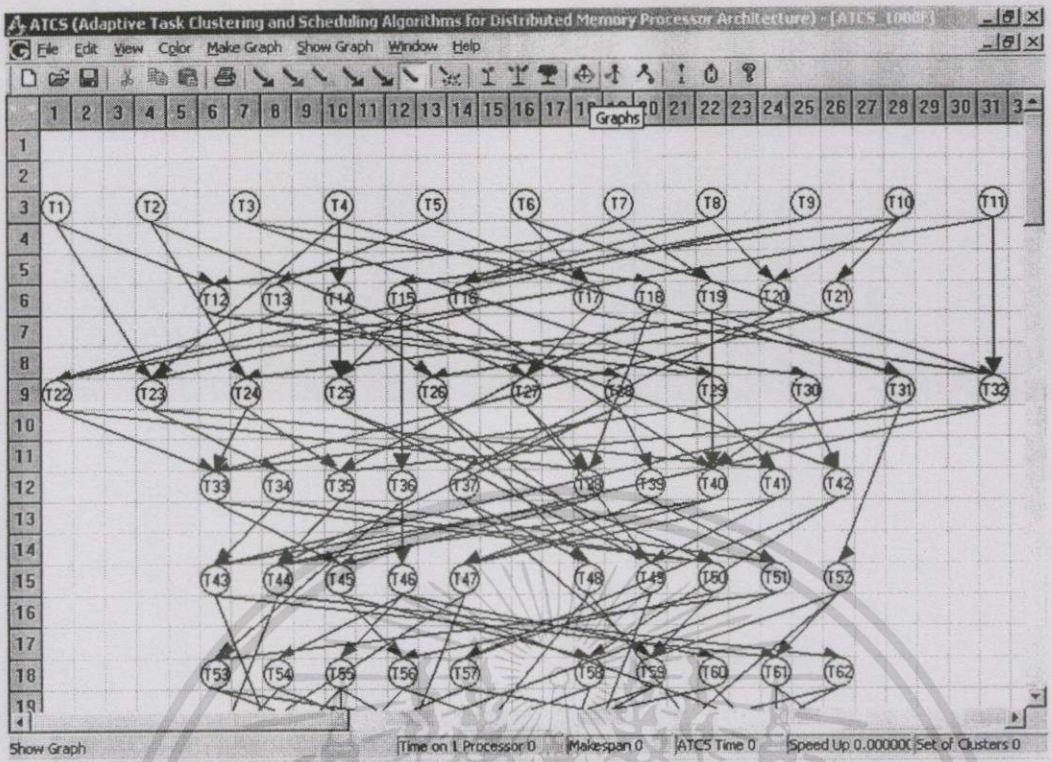
รูปที่ 5.35(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 2528-1619 = 909 หน่วย และค่าของ Speed Up เท่ากับ 2.52 โดยกำหนดให้มี $33 \leq \mu \leq 41$



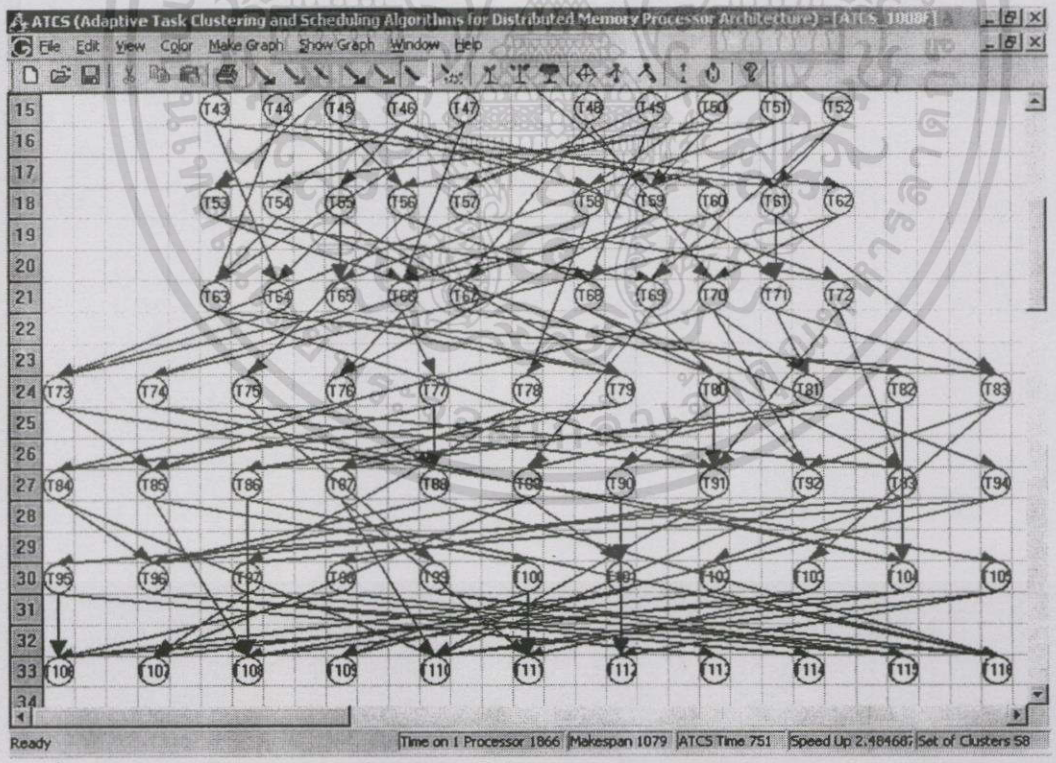
รูปที่ 5.36 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/7



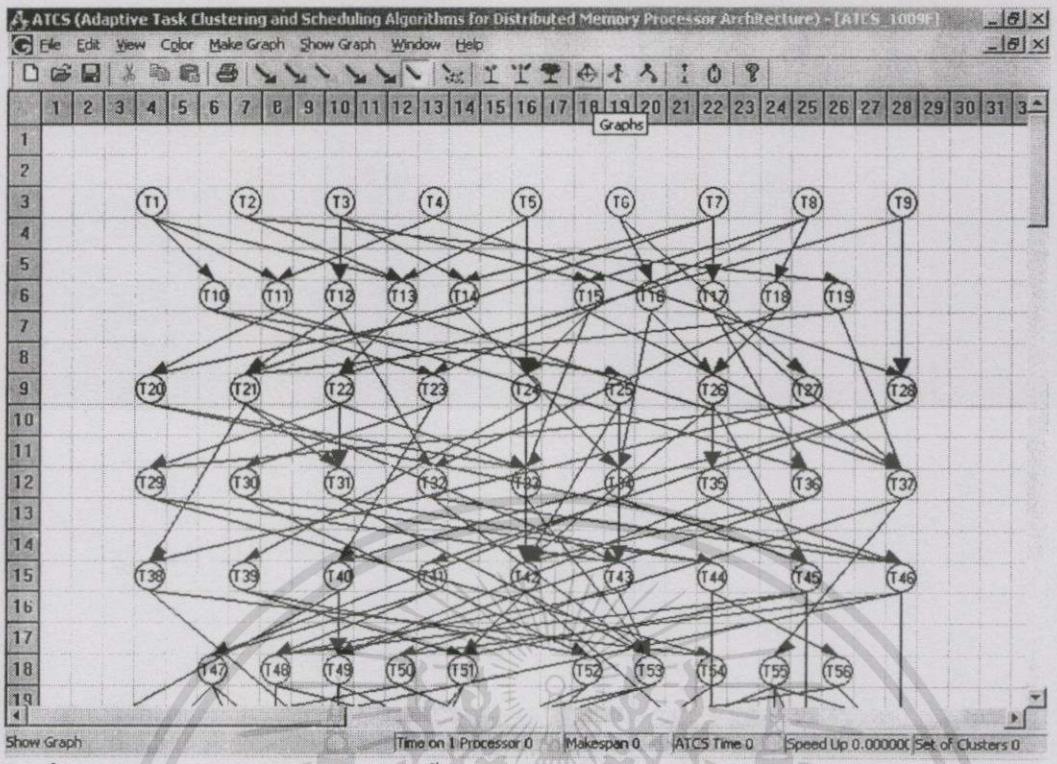
รูปที่ 5.36(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1924-1251 = 673 หน่วย และค่าของ Speed Up เท่ากับ 2.5435 โดยกำหนดให้มี $24 \leq \mu \leq 32$



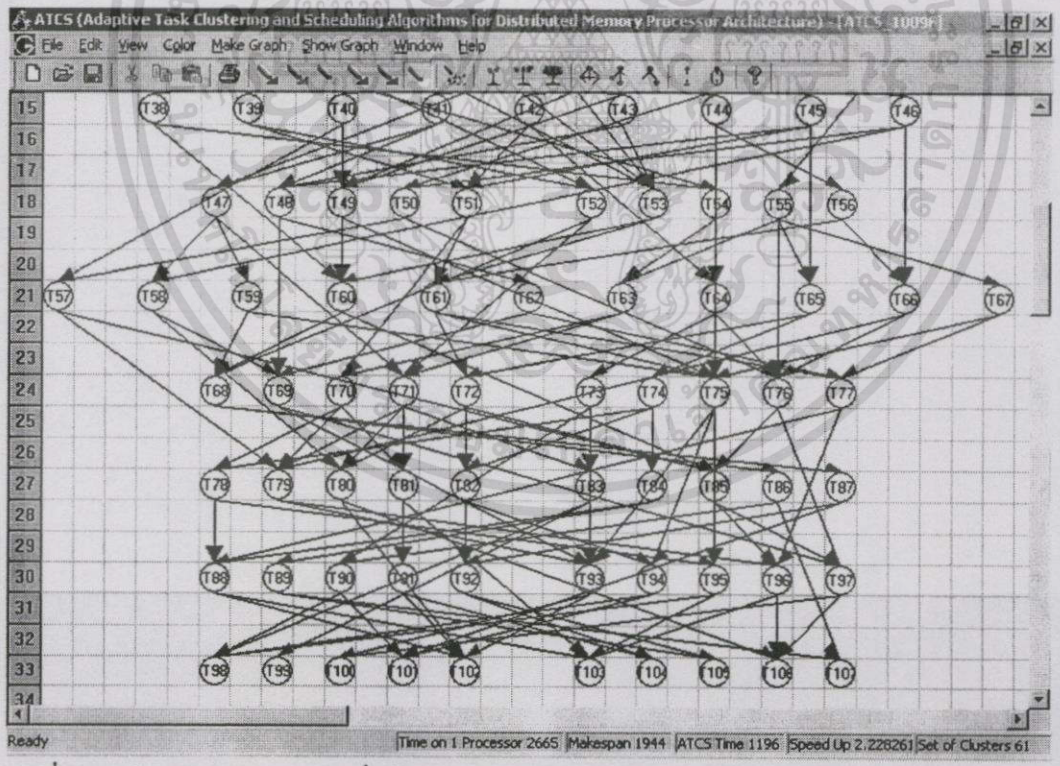
รูปที่ 5.37 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/8



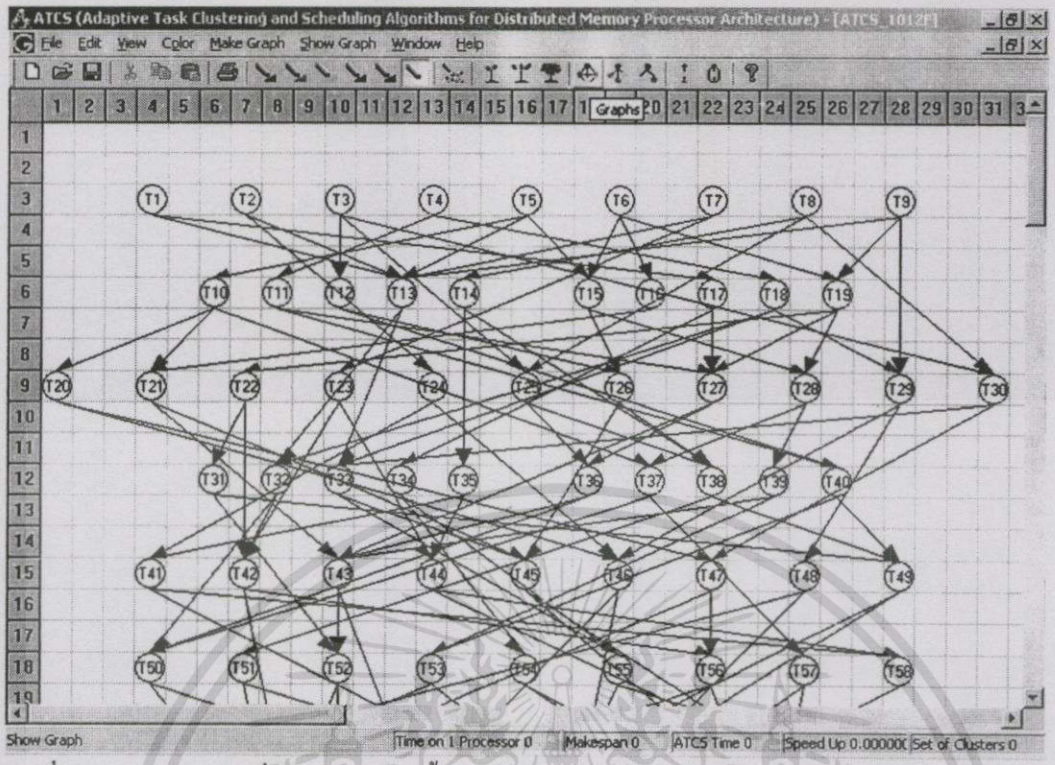
รูปที่ 5.37(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1079-751 = 328 หน่วย และ ค่าของ Speed Up เท่ากับ 2.4846 โดยกำหนดให้มี $12 \leq \mu \leq 20$



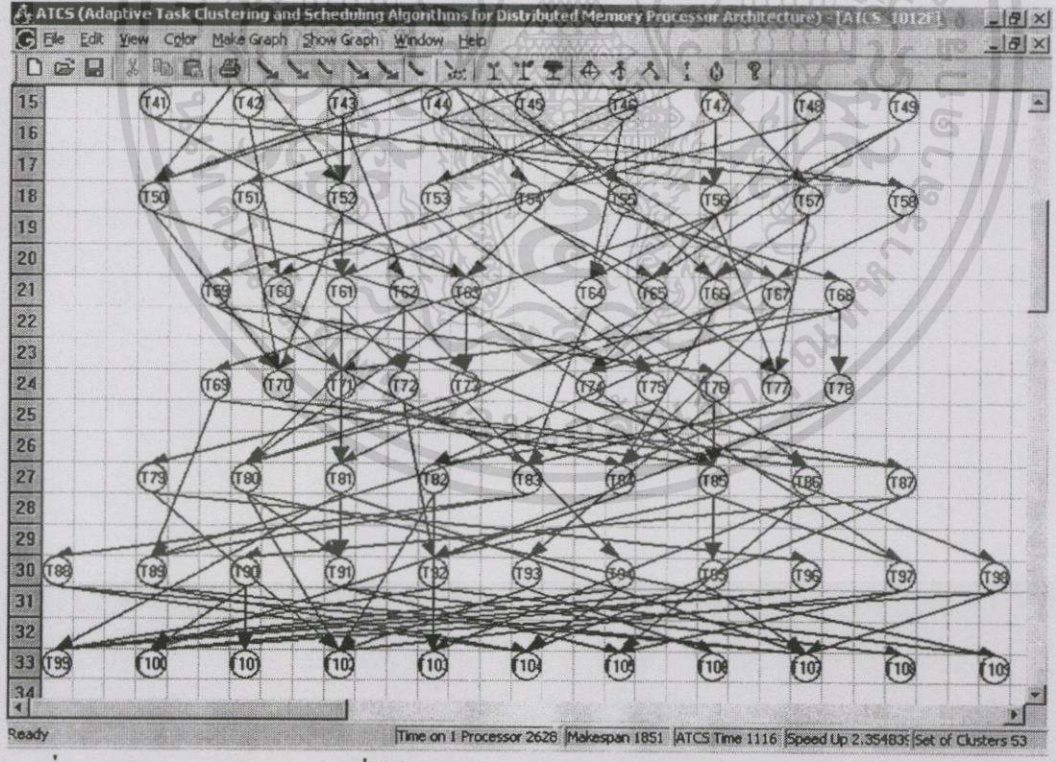
รูปที่ 5.38 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/6



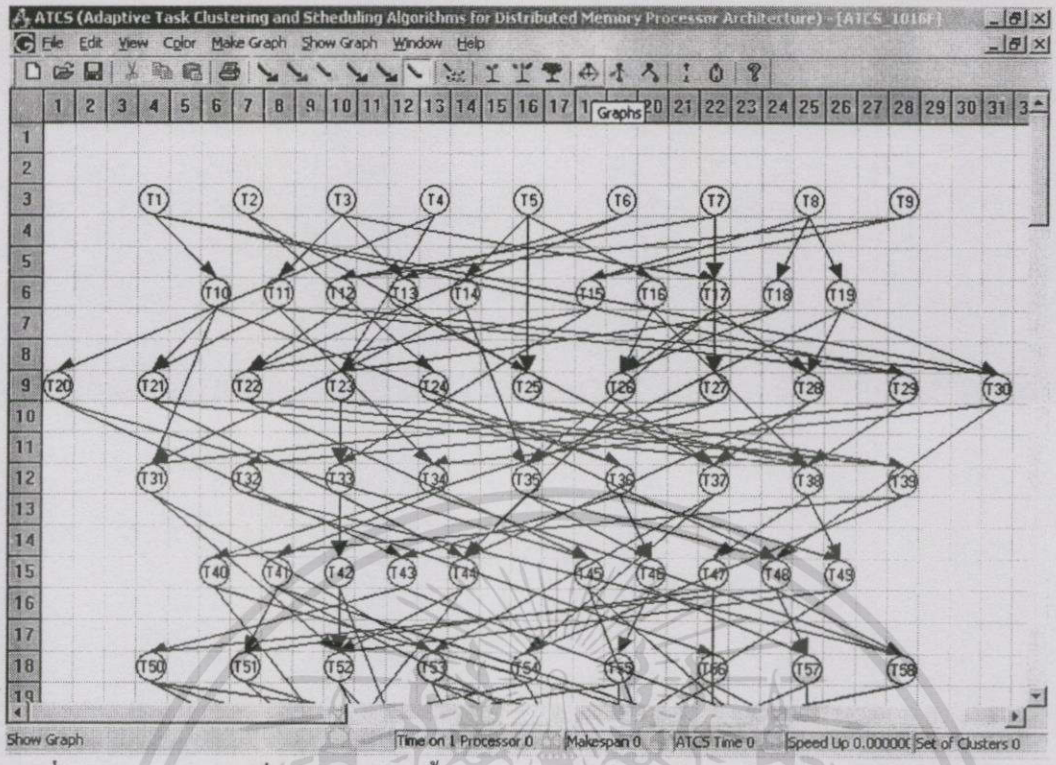
รูปที่ 5.38(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1944-1196 = 748 หน่วย และค่าของ Speed Up เท่ากับ 2.2282 โดยกำหนดให้มี $20 \leq \mu \leq 31$



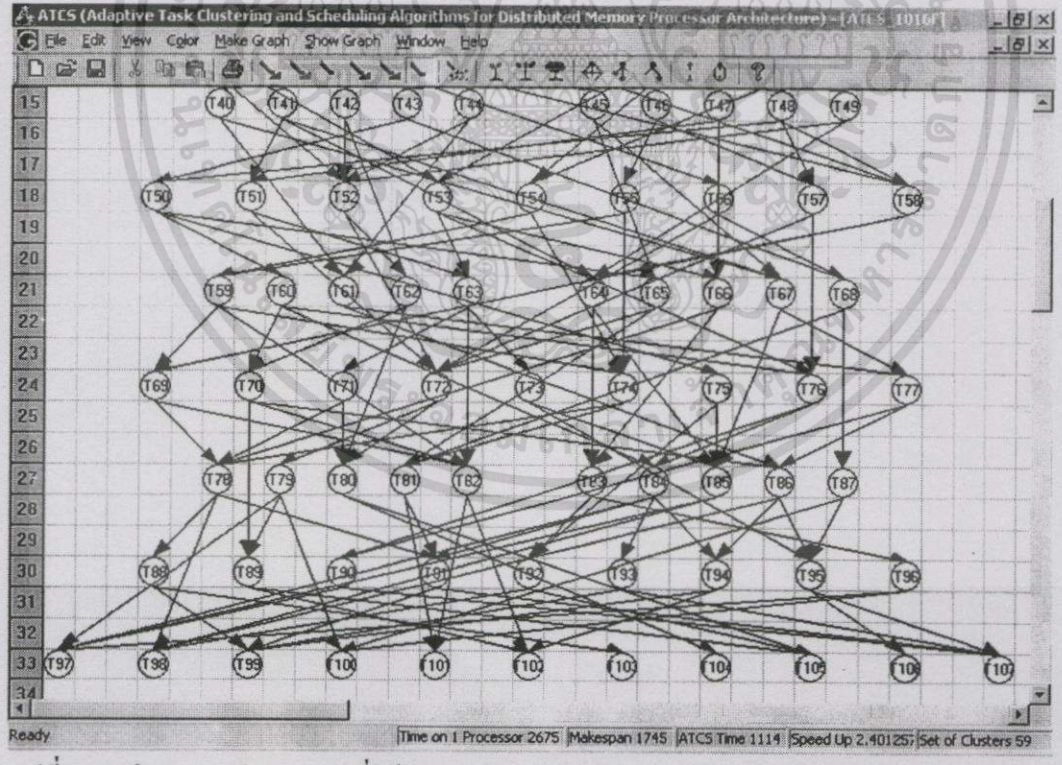
รูปที่ 5.39 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/7



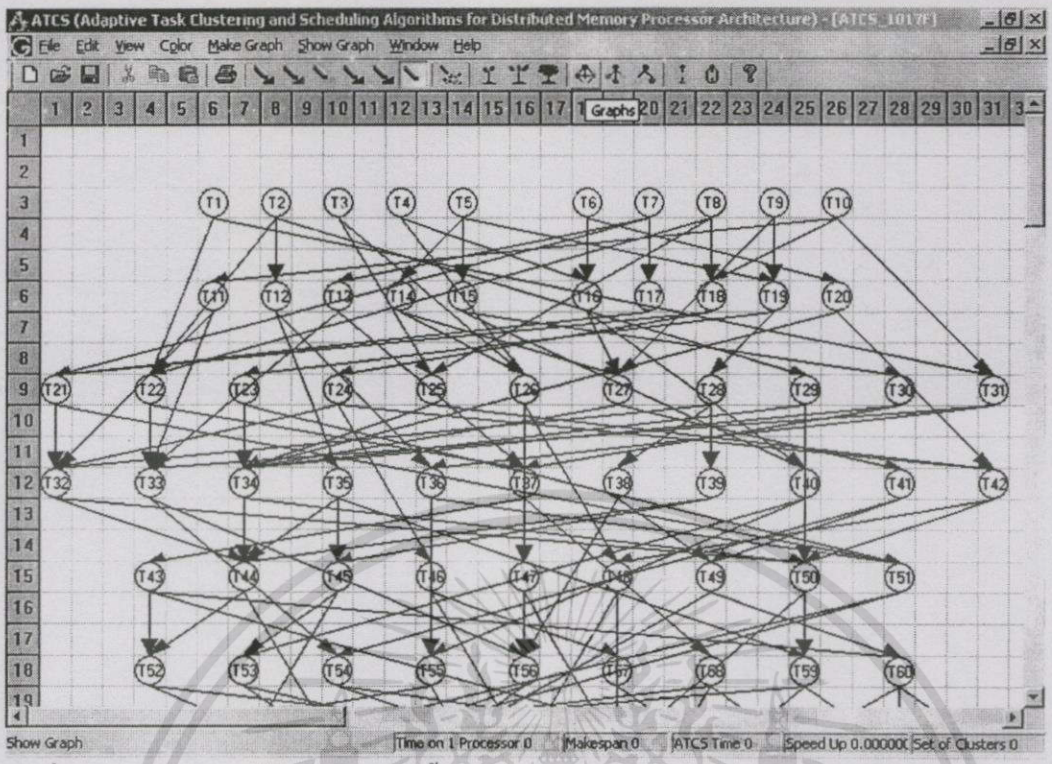
รูปที่ 5.39(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1851-1116 = 735 หน่วย และค่าของ Speed Up เท่ากับ 2.3548 โดยกำหนดให้มี $22 \leq \mu \leq 35$



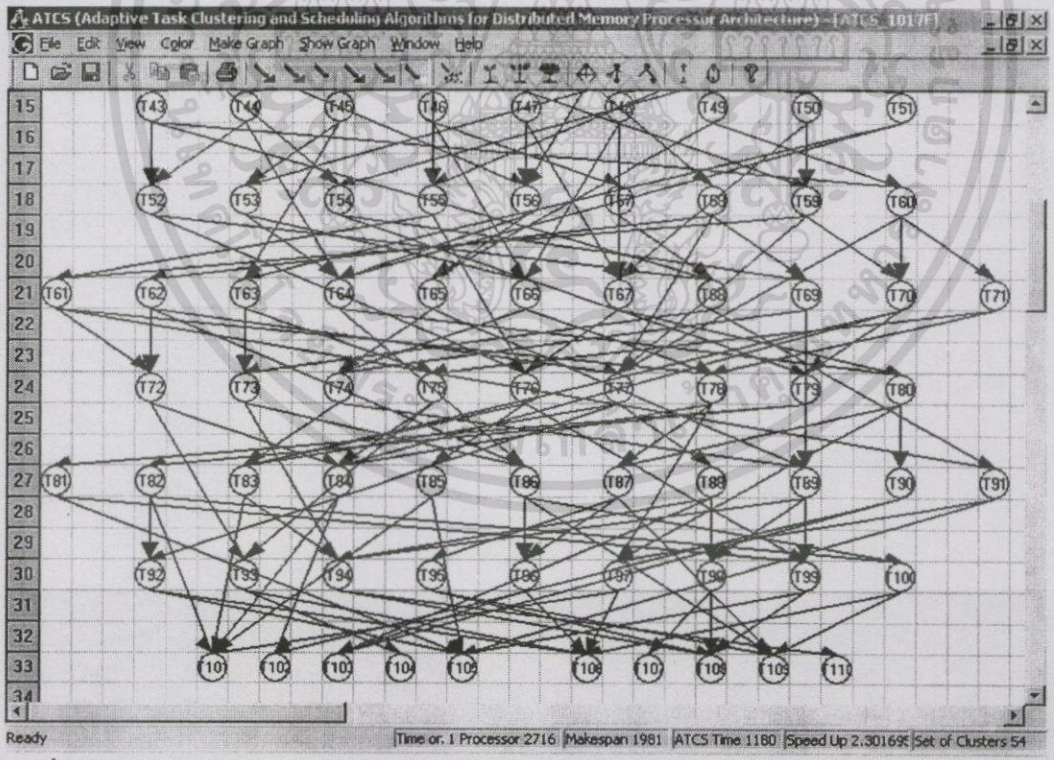
รูปที่ 5.40 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/10



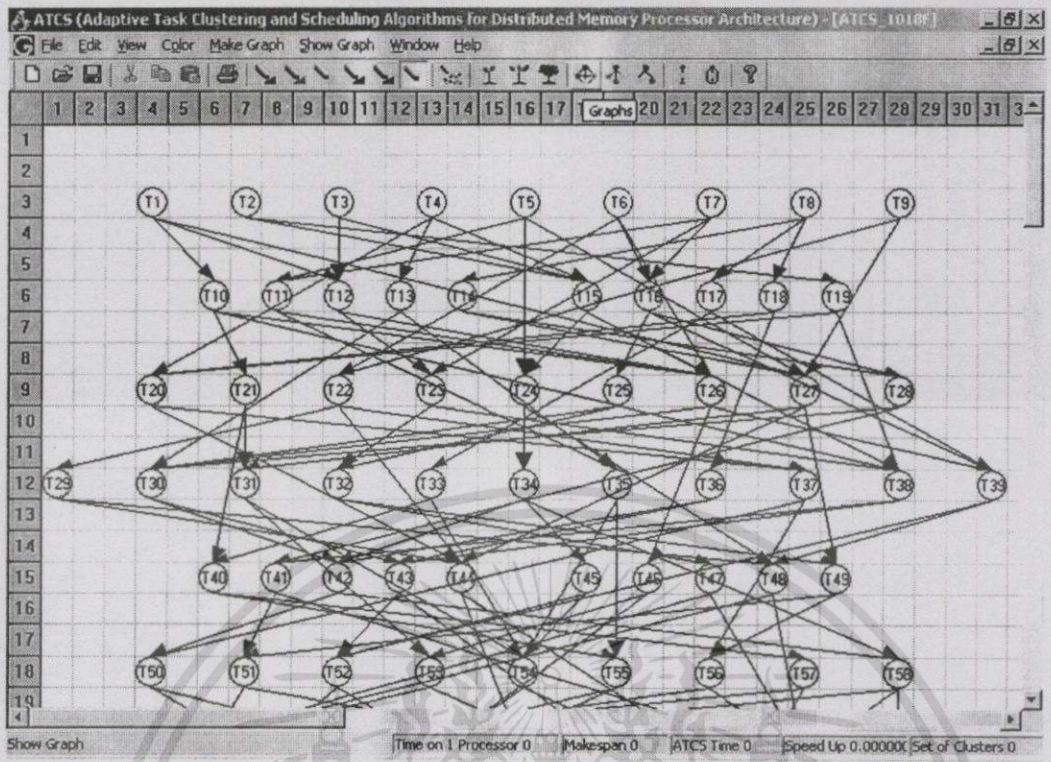
รูปที่ 5.40(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1745-1114 = 631 หน่วย และค่าของ Speed Up เท่ากับ 2.4012 โดยกำหนดให้มี $22 \leq \mu \leq 30$



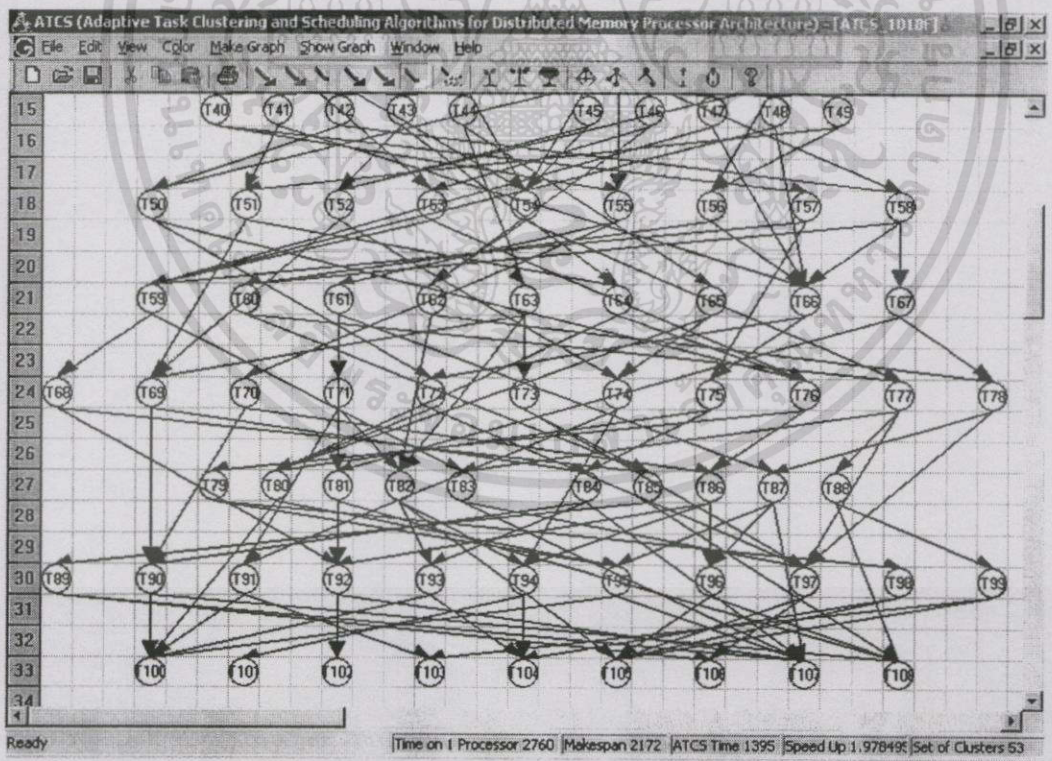
รูปที่ 5.41 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/10



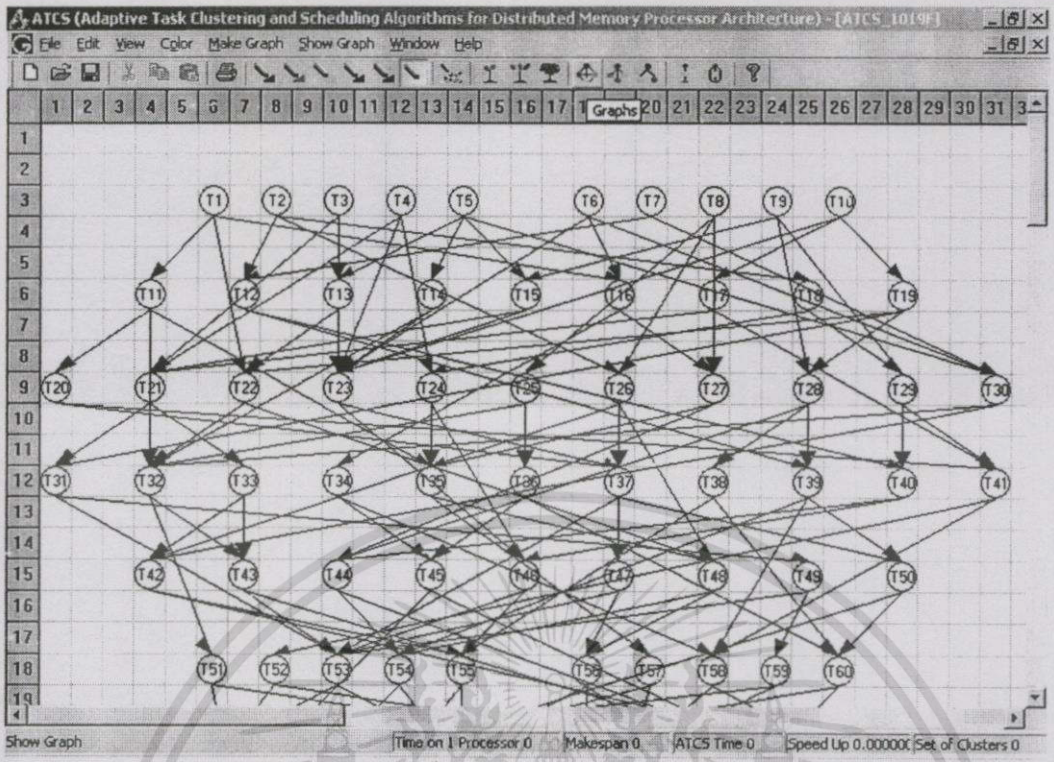
รูปที่ 5.41(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1981-1180 = 801 หน่วย และค่าของ Speed Up เท่ากับ 2.3016 โดยกำหนดให้มี $22 \leq \mu \leq 35$



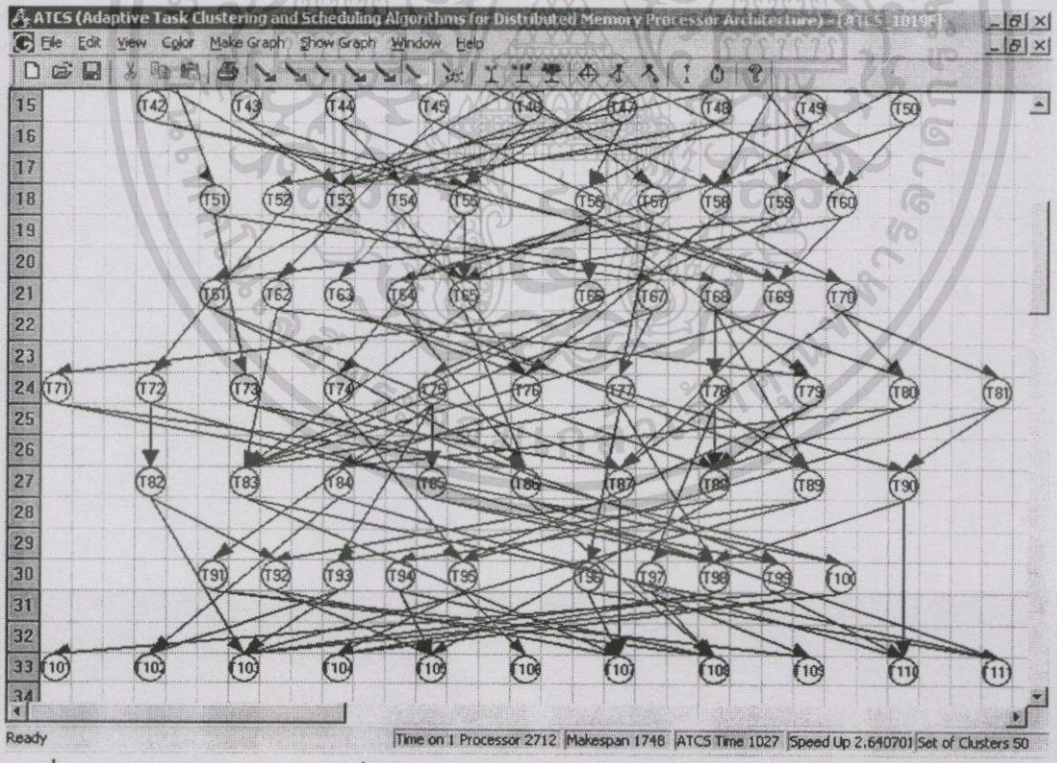
รูปที่ 5.42 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/9



รูปที่ 5.42(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 2172-1395 = 777 หน่วย และค่าของ Speed Up เท่ากับ 1.9784 โดยกำหนดให้มี $20 \leq \mu \leq 30$



รูปที่ 5.43 แสดง DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด และมี Granularity เท่ากับ 1/9



รูปที่ 5.43(ต่อ) แสดงค่าของเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 1748-1027 = 721 หน่วย และ ค่าของ Speed Up เท่ากับ 2.6407 โดยกำหนดให้มี $20 \leq \mu \leq 35$

ตารางที่ 5.3 แสดงเวลาที่ใช้ในการประมวลผลของ DAG ต่างๆ ที่ได้จากการสุ่มของโปรแกรม ดังแสดงอยู่ในรูปที่ 5.13 – 5.43 และค่าของ Speed Up

DAG รูปที่	Range of Execution cost	Time on 1 Processor	Makespan	ATCS Time	Speed Up	A Set of Task-clusters
5.13	$33 \le \mu \le 40$	1996	1588	964	2.0705	20
5.14	$22 \le \mu \le 34$	1564	1449	951	1.6445	29
5.15	$23 \le \mu \le 32$	1368	1310	785	1.7426	25
5.16	$22 \le \mu \le 32$	1465	1338	610	2.4016	25
5.17	$23 \le \mu \le 32$	1538	1295	696	2.2097	28
5.18	$31 \le \mu \le 40$	1945	1635	991	1.9626	29
5.19	$22 \le \mu \le 32$	1693	1794	1116	1.5170	37
5.20	$20 \le \mu \le 30$	1914	1747	1060	1.8056	34
5.21	$21 \le \mu \le 30$	1808	1408	847	2.1345	35
5.22	$12 \le \mu \le 22$	1165	731	478	2.4372	33
5.23	$22 \le \mu \le 34$	1941	1535	1031	1.8826	32
5.24	$23 \le \mu \le 30$	2316	1644	985	2.3512	41
5.25	$23 \le \mu \le 33$	2234	1574	1014	2.2031	47
5.26	$22 \le \mu \le 29$	2233	1590	1111	2.0099	44
5.27	$22 \le \mu \le 30$	2171	1694	1153	1.8829	51
5.28	$31 \le \mu \le 39$	3104	2257	1320	2.3515	43
5.29	$32 \le \mu \le 40$	3297	2273	1516	2.1748	46
5.30	$22 \le \mu \le 30$	2233	1611	1095	2.0392	38
5.31	$15 \le \mu \le 22$	1605	1190	716	2.2416	45
5.32	$20 \le \mu \le 33$	2331	1831	1050	2.2200	41
5.33	$10 \le \mu \le 20$	1404	855	552	2.5434	41
5.34	$22 \le \mu \le 29$	2667	1724	1077	2.4763	49
5.35	$33 \le \mu \le 41$	4080	2528	1619	2.5200	61
5.36	$24 \le \mu \le 32$	3182	1924	1251	2.5435	58
5.37	$12 \le \mu \le 20$	1866	1079	751	2.4846	58
5.38	$20 \le \mu \le 31$	2665	1944	1196	2.2282	61
5.39	$22 \le \mu \le 35$	2628	1851	1116	2.3548	53
5.40	$22 \le \mu \le 30$	2675	1745	1114	2.4012	59
5.41	$22 \le \mu \le 35$	2716	1981	1180	2.3016	54
5.42	$20 \le \mu \le 30$	2760	2172	1395	1.9784	53
5.43	$20 \le \mu \le 35$	2712	1748	1027	2.6407	50

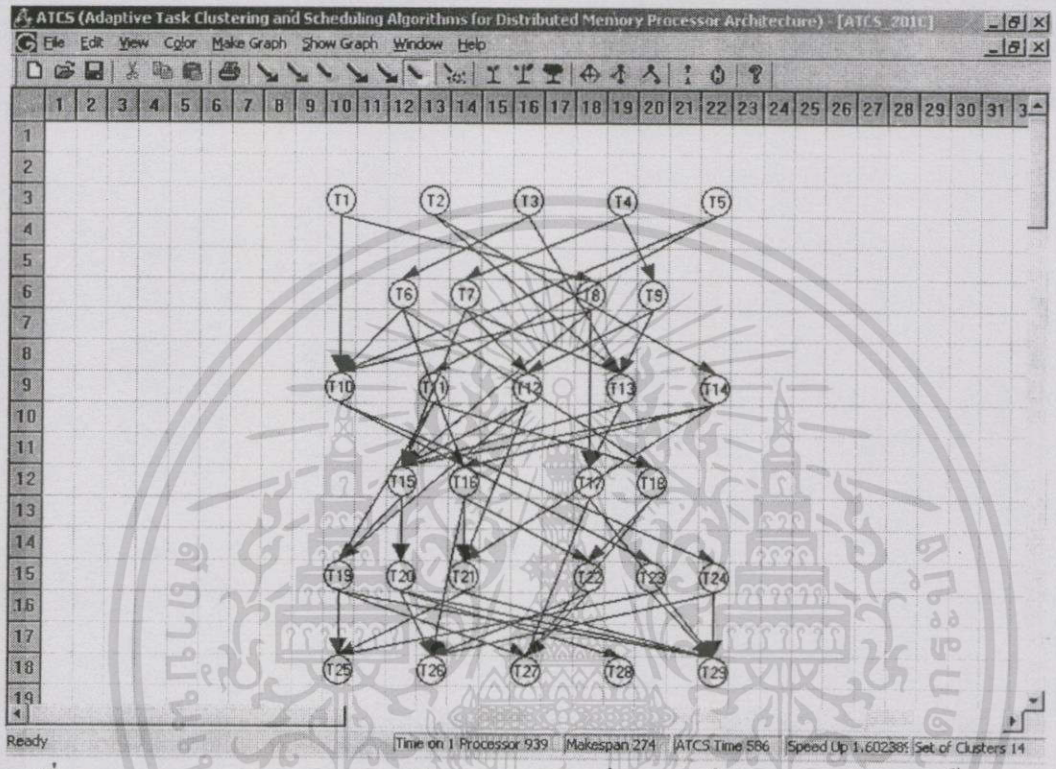
จากผลการทดลองด้วยลักษณะของ DAG ที่ได้จากการสุ่มสร้าง จะพบได้ว่าเอทีซีเอสอัลกอริทึมจะมีความสามารถในการลดเวลาการประมวลผลของโปรแกรมแบบขนานได้มากน้อยแค่ไหน ขึ้นอยู่กับลักษณะของ DAG เป็นสิ่งสำคัญ จากการทดลองจะสังเกตพบได้ว่าขนาดของจำนวนของโหนดที่เพิ่มมากขึ้นอาจจะเป็นปัจจัยสำคัญที่ทำให้ประสิทธิภาพในการลดเวลาในการประมวลผลของโปรแกรมแบบขนานลดน้อยลง เมื่อเทียบกับค่าของ Makespan ดังนั้นปัจจัยสำคัญที่ทำให้ประสิทธิภาพในการลดเวลาในการประมวลผลของโปรแกรมแบบขนานลดน้อยลงได้แก่ลักษณะของ 1)การกระจายภาระงานของโหนดต่างๆโดยสังเกตได้จากจำนวนของอาร์คที่ชี้เข้าหาโหนด 2)จำนวนความหนาแน่นของอาร์คของ DAG 3)ขนาดของ DAG ที่ใหญ่มากขึ้น ในทางตรงกันข้ามค่าของ Speed Up ไม่ได้ลดลงเหมือนกับค่าความแตกต่างระหว่างค่าของ Makespan และค่าของ ATCS Time

จากการทดลองจะสังเกตได้ว่า เมื่อขนาดของ DAG ใหญ่มากขึ้นความสลับซับซ้อนของ DAG ก็เพิ่มมากขึ้นด้วย โดยสังเกตได้จากค่าของ Time on 1 Processor จะสูงกว่าค่าของ Makespan และ ATCS Time ดังนั้นเราสามารถสรุปได้ว่า เมื่อขนาดของ DAG ใหญ่มากขึ้นหรือลักษณะของโปรแกรมแบบขนานมีขนาดที่ใหญ่มากขึ้น ไม่เหมาะแก่การที่จะนำเอางานลักษณะดังกล่าวไปประมวลผลบนเครื่องคอมพิวเตอร์ที่มีตัวประมวลผลเพียงตัวเดียว จะสังเกตพบได้ว่าเมื่อโปรแกรมแบบขนานมีขนาดที่ใหญ่มากขึ้นค่าของ Speed Up ก็จะมากตามไปด้วย และจะสังเกตพบได้ว่าเมื่อ DAG ที่มีโครงสร้างไม่ใหญ่มากนักหรือมีความสลับซับซ้อนไม่มาก ความแตกต่างระหว่างค่าของ Time on 1 Processor, Makespan, และ ATCS Time จะพบว่ามีความแตกต่างกันน้อยมาก DAG บางลักษณะกลับให้ค่าของ Time on 1 Processor น้อยกว่าค่าของ Makespan และ ATCS Time ด้วยซ้ำไป ดังนั้น DAG ลักษณะดังกล่าวเหมาะแก่การนำไปประมวลผลบนเครื่องคอมพิวเตอร์ที่มีตัวประมวลผลเพียงตัวเดียวก็เป็นการเพียงพอแล้ว

สองส่วนถัดไปจากนี้เป็นผลการทดลองจากลักษณะของ DAG ที่อยู่นอกเหนือขอบเขตเป้าหมายของวิทยานิพนธ์ฉบับนี้ ผลการทดลองเป็นเพียงการแสดงผลลัพธ์บางอย่างที่เกิดจากการใช้เอทีซีเอสอัลกอริทึม ซึ่งวิธีการของเอทีซีเอสอัลกอริทึมอาจจะเป็นประโยชน์ก็ได้จากลักษณะของ DAG ดังกล่าว

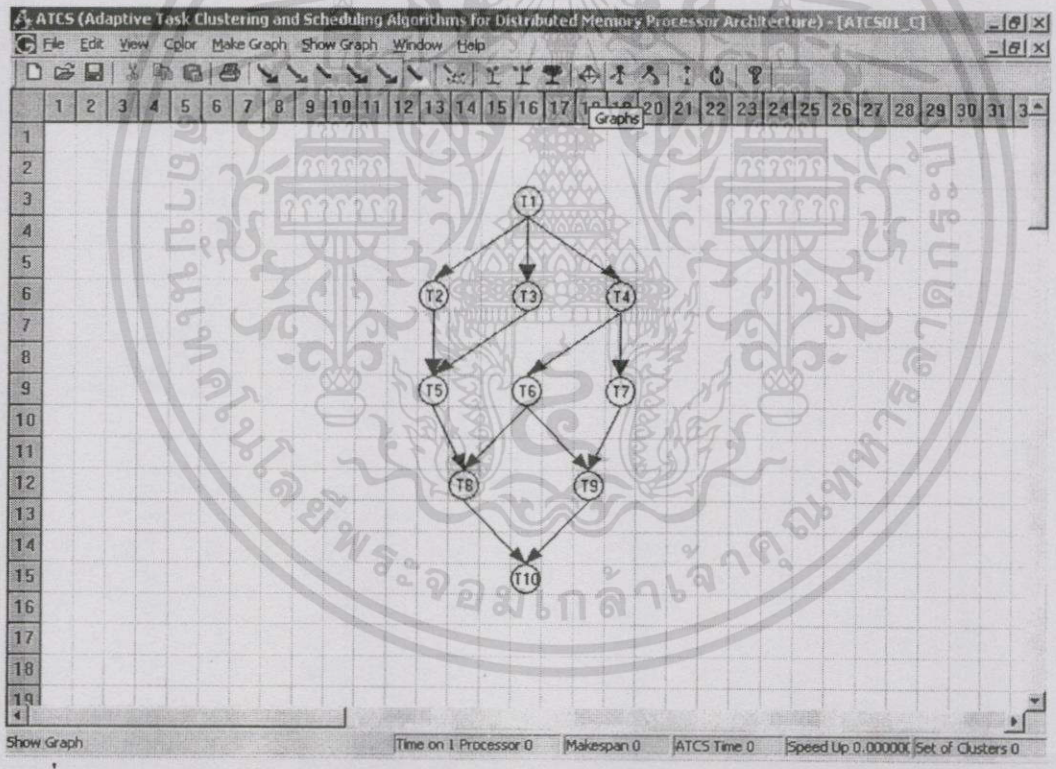
5.2.2 ผลการทดลองกับ DAG ที่มีลักษณะเป็นเม็ดเนื้องานแบบหยาบ ในส่วนนี้เป็นการแสดงผลการทดลองกับ DAG ที่มีลักษณะเม็ดเนื้องานแบบหยาบ หรือ DAG ที่มีลักษณะของงานหรือโหนดใดๆที่มีค่าของ λ น้อยมากเมื่อเทียบกับค่าของ μ ของโหนดนั้นๆ โดยในที่นี้จะทำการแสดงเพียงผลการทดลองเดียว เนื่องมาจากวิธีการของเอทีซีเอสอัลกอริทึมไม่สามารถทำการจัดกลุ่มงานสำหรับ DAG ที่มีลักษณะเม็ดเนื้องานแบบหยาบได้

รูปที่ 5.44 เป็นการแสดงผลการทดลองกับ DAG ที่มีลักษณะเม็ดเนื้องานแบบหยาบ จากผลการทดลองแสดงให้เห็นว่า ค่าของ ATCS Time ที่ได้รับจากการใช้เอทีซีเอสอัลกอริทึมจะมากกว่าค่าของ Makespan หรือเส้นทางวิกฤตของ DAG แสดงให้เห็นว่าวิธีการของเอทีซีเอสอัลกอริทึมไม่สามารถช่วยลดเวลาการทำงานของโปรแกรมแบบขนานลงได้

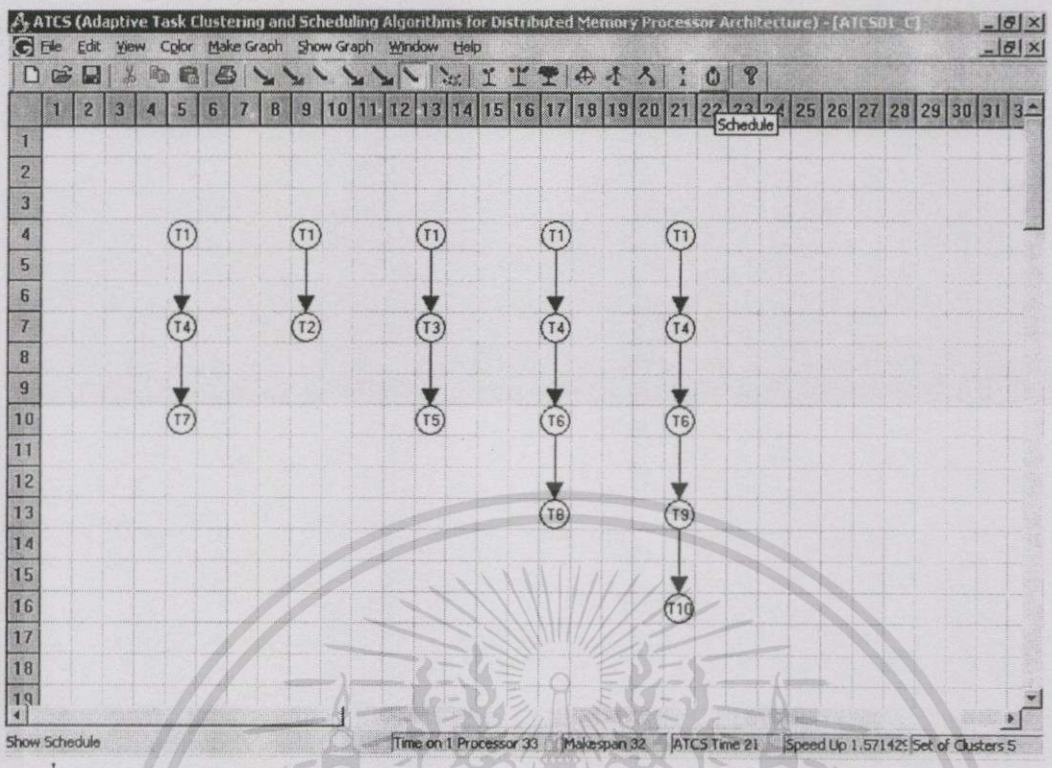


รูปที่ 5.44 แสดงผลการทดลองของเอทีซีเอสอัลกอริทึม เมื่อ DAG ที่ใช้มีลักษณะเป็นเม็ดเนื้องานแบบหยาบ เมื่อกำหนดให้ Granularity เท่ากับ 2

5.2.3 ผลการทดลองกับ DAG ที่มีลักษณะแบบอื่นๆ ในส่วนนี้เป็นการแสดงผลการทดลองกับ DAG ที่มีลักษณะเม็ดเนื้องานที่นอกเหนือไปจากนิยามของ[2] โดย DAG ที่แสดงในส่วนนี้ได้มาจาก[32]และได้จากการสุ่มของโปรแกรมเอทีซีเอส เมื่อลักษณะของ DAG ดังกล่าวยังคงมีค่าของ $\mu \geq \lambda$ (อาจจะกล่าวได้ว่าเป็น DAG ที่มีลักษณะนอกเหนือไปจากนิยามของ[2])โดยผลลัพธ์ที่ได้จากการทดลองกับ DAG ใน[32]แสดงอยู่ในรูปที่ 5.45 และรูปที่ 5.46-5.47 เป็นการแสดงผลลัพธ์ที่ได้จากการทดลองกับ DAG ที่ได้จากการสุ่มของโปรแกรม ทั้งนี้เนื่องมาจากเอทีซีเอสอัลกอริทึมสามารถให้ผลลัพธ์อย่างมีประสิทธิภาพเฉพาะ DAG ที่มีลักษณะดังกล่าวได้ในบางกรณีเท่านั้น กล่าวคือเป็น DAG ที่มีจำนวนของอาร์คไม่มาก จำนวนของโหนดไม่มาก ค่าของความแตกต่างระหว่างค่าของ μ และ λ ค่อนข้างน้อย และระดับความลึกของ DAG ไม่มากนัก รูปที่ 5.48-5.49 เป็นการแสดงผลลัพธ์ที่ได้จากการทดลองกับ DAG ที่ได้จากการสุ่มของโปรแกรม โดย DAG มีลักษณะที่มีจำนวนของอาร์คไม่มากสำหรับโหนดใดๆ และจำนวนของระดับชั้นที่อยู่บน DAG มากกว่า 3 ระดับ



รูปที่ 5.45(a) ลักษณะของ DAG [32]



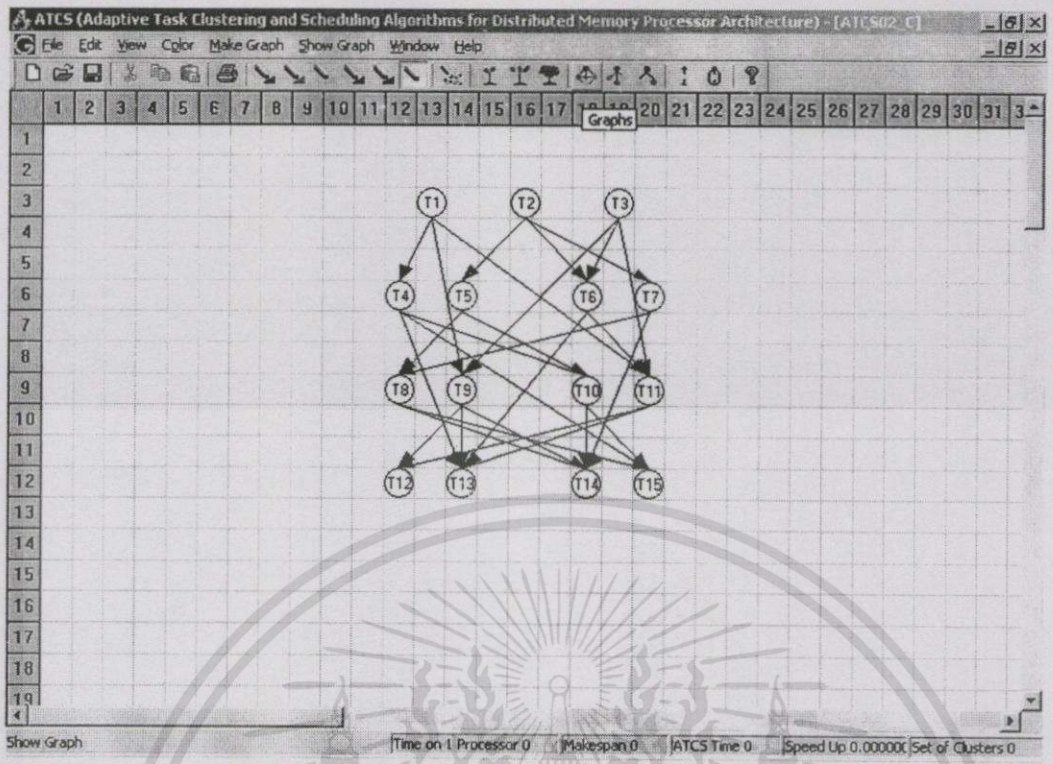
รูปที่ 5.45(b) แสดงผลลัพธ์จากการจัดกลุ่มงานและกำหนดลำดับการทำงานให้กับ DAG ของ[32] โดยใช้เอทีซีเอสอัลกอริทึม และค่าของ Speed Up เท่ากับ 1.5714

ตารางที่ 5.4 แสดงค่าใช้จ่ายในการประมวลผล(μ) ระดับของโหนด และค่าใช้จ่ายในการติดต่อสื่อสาร(λ)ระหว่างโหนดต้นทางไปยังโหนดปลายทางบน DAG รูปที่ 5.45

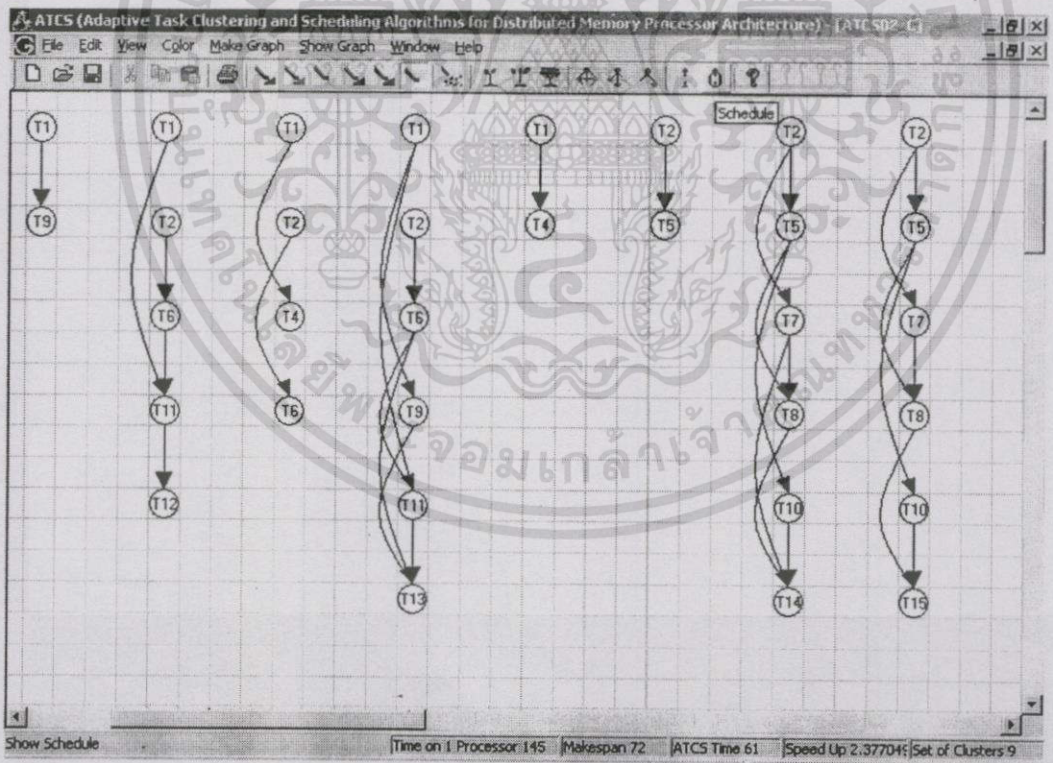
Task No.	μ	Level
T1	3	0
T2	2	1
T3	4	1
T4	3	1
T5	2	2
T6	6	2
T7	2	2
T8	2	3
T9	6	3
T10	3	4

Source Node	Destination Node	λ
1	2	2
1	3	3
1	4	3
2	5	2
3	5	4
4	6	4
4	7	2
5	8	2
6	8	2
6	9	2
7	9	2
8	10	2
9	10	3

รูปที่ 5.45 เป็นการแสดงผลลัพธ์ที่ได้จากการใช้เอทีซีเอสอัลกอริทึม โดยเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 32 – 21 = 11 หน่วยเวลา ในขณะเดียวกันจำนวนของตัวประมวลผลที่ต้องการจะใช้สำหรับการประมวลผลสำหรับกลุ่มงานชุดนี้ต้องการเพียง 5 ตัวประมวลผล ซึ่งเป็นการลดจำนวนของตัวประมวลผลเหลือเพียงครึ่งเดียว รูปที่ 5.46 เป็นการแสดงผลลัพธ์จากการใช้เอทีซีเอสอัลกอริทึม ซึ่งทำให้เวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 72 – 61 = 11 หน่วยเวลา และตัวประมวลผลที่ต้องการจะใช้เหลือเพียง 9 ตัวประมวลผล และรูปที่ 5.47 เป็นการแสดงผลลัพธ์จากการใช้เอทีซีเอสอัลกอริทึม โดยเวลาที่ใช้ในการประมวลผลลดลงเท่ากับ 102 – 86 = 16 หน่วยเวลา รูปที่ 5.48 แสดงให้เห็นว่าวิธีการของเอทีซีเอสอัลกอริทึมไม่สามารถทำงานได้อย่างมีประสิทธิภาพ ในกรณีที่ DAG นั้นมีจำนวนของระดับชั้นมากกว่า 3 ชั้น โดยจะเห็นได้ว่า ค่าของ ATCS Time มีค่ามากกว่าค่าของเส้นทางวิกฤตหรือ Makespan ดังนั้นวิธีการของเอทีซีเอสอัลกอริทึมไม่ได้ช่วยทำให้เวลาที่ใช้ในการประมวลผลของงานของโปรแกรมแบบขนานลดลงได้ รูปที่ 5.49 เป็นผลการทดลองอีกอันหนึ่งที่แสดงให้เห็นว่า วิธีการของเอทีซีเอสอัลกอริทึมมีประสิทธิภาพที่ด้อยลง อันเนื่องมาจากลักษณะของ DAG ที่มีความสลับซับซ้อนเพิ่มมากขึ้นโดยสรุปแล้ววิธีการของเอทีซีเอสอัลกอริทึมไม่สามารถให้ผลลัพธ์ที่ดีได้เสมอไปสำหรับกรณีของ DAG ในส่วนนี้



รูปที่ 5.46(a) DAG ชุดที่1 เมื่อค่าระหว่าง μ และ λ มีความแตกต่างกันไม่มาก

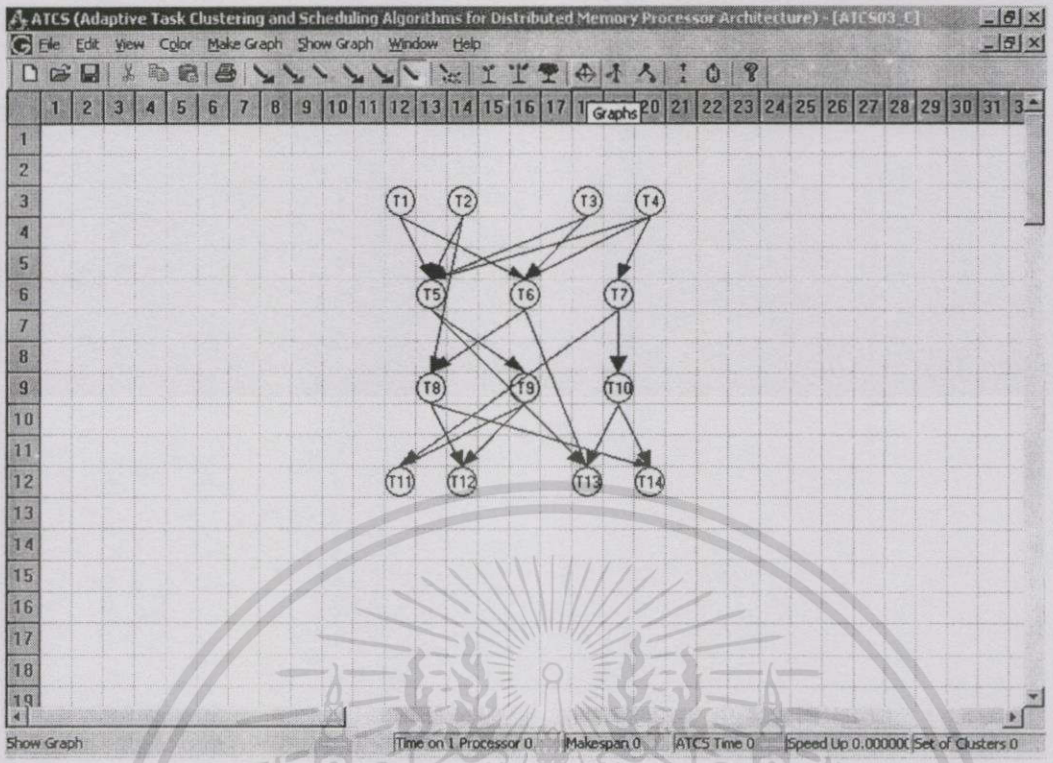


รูปที่ 5.46(b) ผลลัพธ์จากการใช้เอทีซีเอสอัลกอริทึมด้วยDAG ชุดที่1 โดยมีค่าของ Speed Up เท่ากับ 2.377

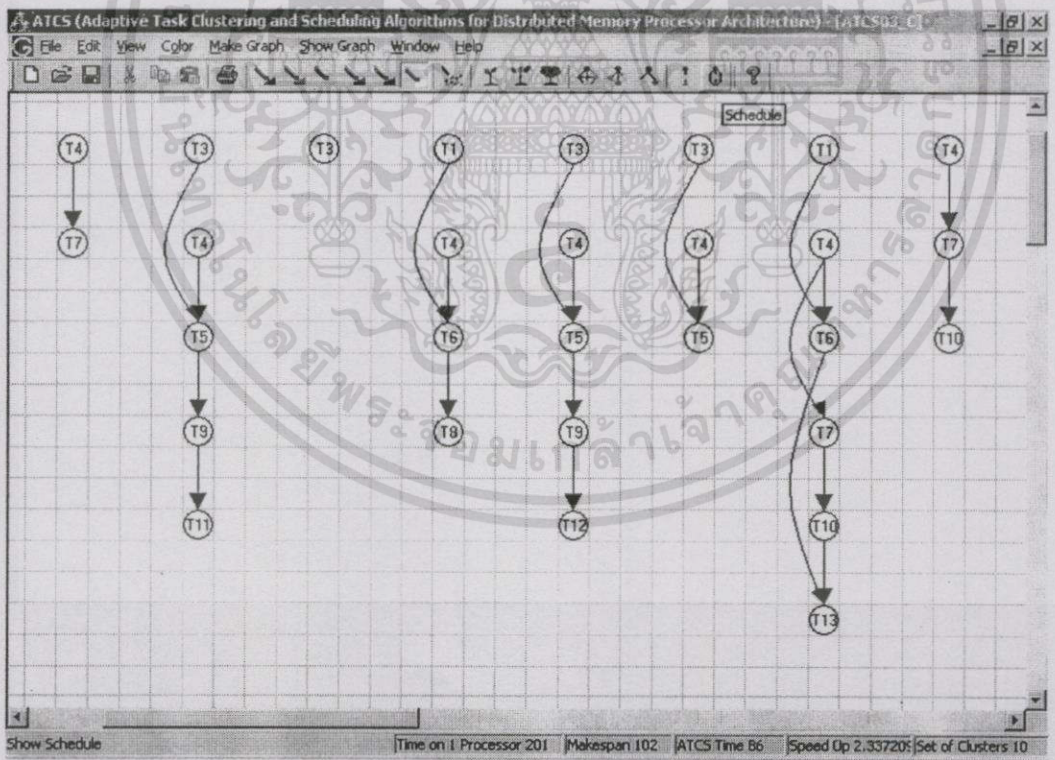
ตารางที่ 5.5 แสดงค่าใช้จ่ายในการประมวลผล(μ) ระดับของโหนด และค่าใช้จ่ายในการติดต่อสื่อสาร(λ)ระหว่างโหนดต้นทางไปยังโหนดปลายทางบน DAG รูปที่ 5.46

Task No.	μ	Level
T1	9	0
T2	11	0
T3	8	0
T4	9	1
T5	8	1
T6	10	1
T7	10	1
T8	10	2
T9	9	2
T10	10	2
T11	10	2
T12	9	3
T13	12	3
T14	8	3
T15	12	3

Source Node	Destination Node	λ
1	9	9
1	11	9
2	7	10
2	5	6
3	11	8
3	9	8
4	15	8
4	13	8
5	10	6
5	8	5
6	13	6
6	11	10
7	8	10
7	14	7
8	15	9
8	14	5
9	14	7
9	13	9
10	15	10
10	14	6
11	13	9
11	12	8
1	4	5
3	6	5
2	6	8
4	10	5
9	12	6



รูปที่ 5.47(a) DAG ชุดที่2 เมื่อค่าระหว่าง μ และ λ มีความแตกต่างกันไม่มาก

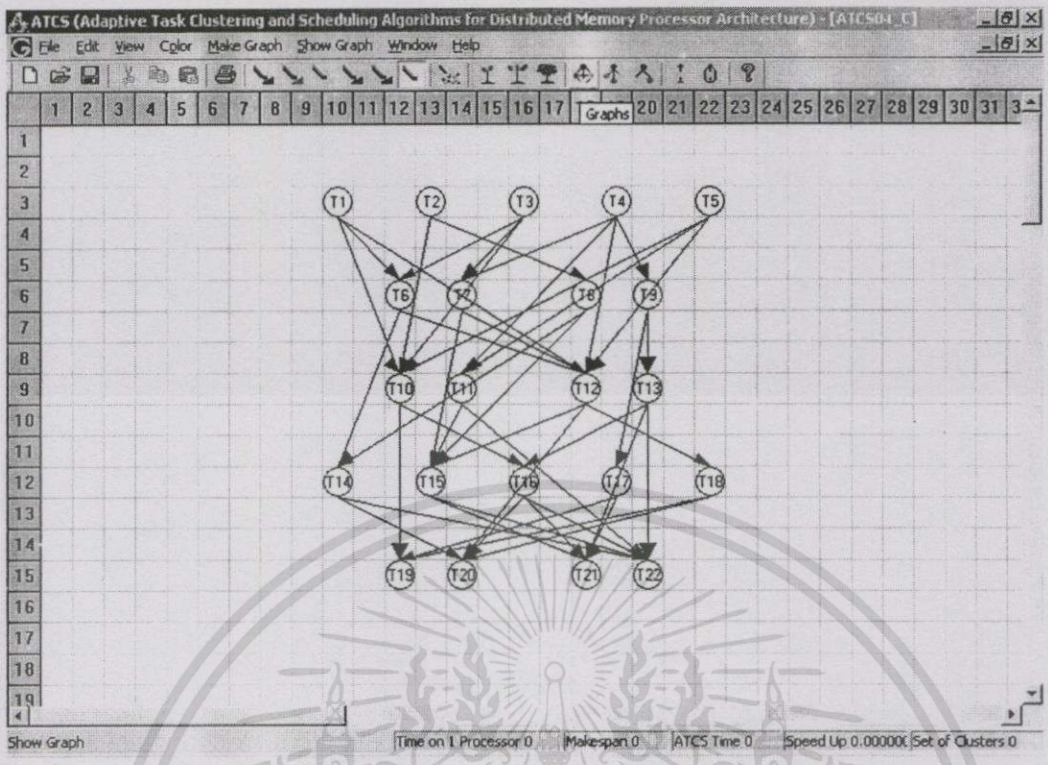


รูปที่ 5.47(b) ผลลัพธ์จากการใช้เอทีซีเอสอัลกอริทึมด้วย DAG ชุดที่2 โดยมีค่าของ Speed Up เท่ากับ 2.3372

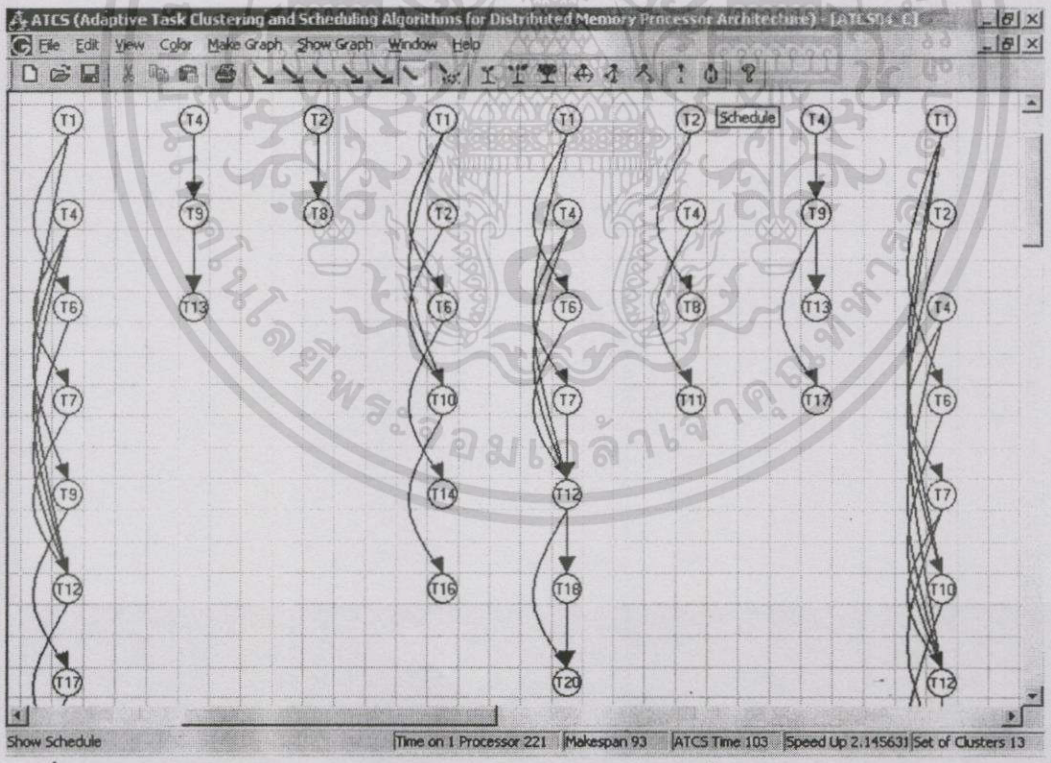
ตารางที่ 5.6 แสดงค่าใช้จ่ายในการประมวลผล(μ) ระดับของโหนด และค่าใช้จ่ายในการติดต่อสื่อสาร(λ)ระหว่างโหนดต้นทางไปยังโหนดปลายทางบน DAG รูปที่ 5.47

Task No.	μ	Level
T1	13	0
T2	16	0
T3	13	0
T4	15	0
T5	15	1
T6	15	1
T7	16	1
T8	14	2
T9	14	2
T10	12	2
T11	16	3
T12	16	3
T13	14	3
T14	12	3

Source Node	Destination Node	λ
1	6	13
1	5	11
2	8	10
2	5	10
3	5	13
3	6	9
4	5	15
4	6	15
5	13	8
5	8	14
6	13	12
6	8	8
7	11	16
7	10	10
8	14	10
8	12	13
9	12	13
9	11	10
10	13	8
10	14	12
4	7	8



รูปที่ 5.48(a) DAG ชุดที่3 เมื่อค่าระหว่าง μ และ λ มีความแตกต่างกันไม่มาก



รูปที่ 5.48(b) ผลลัพธ์จากการใช้เอทีซีเอสอัลกอริทึมด้วย DAG ชุดที่3 โดยเวลาของ ATCS Time มากกว่าเวลาของ Makespan

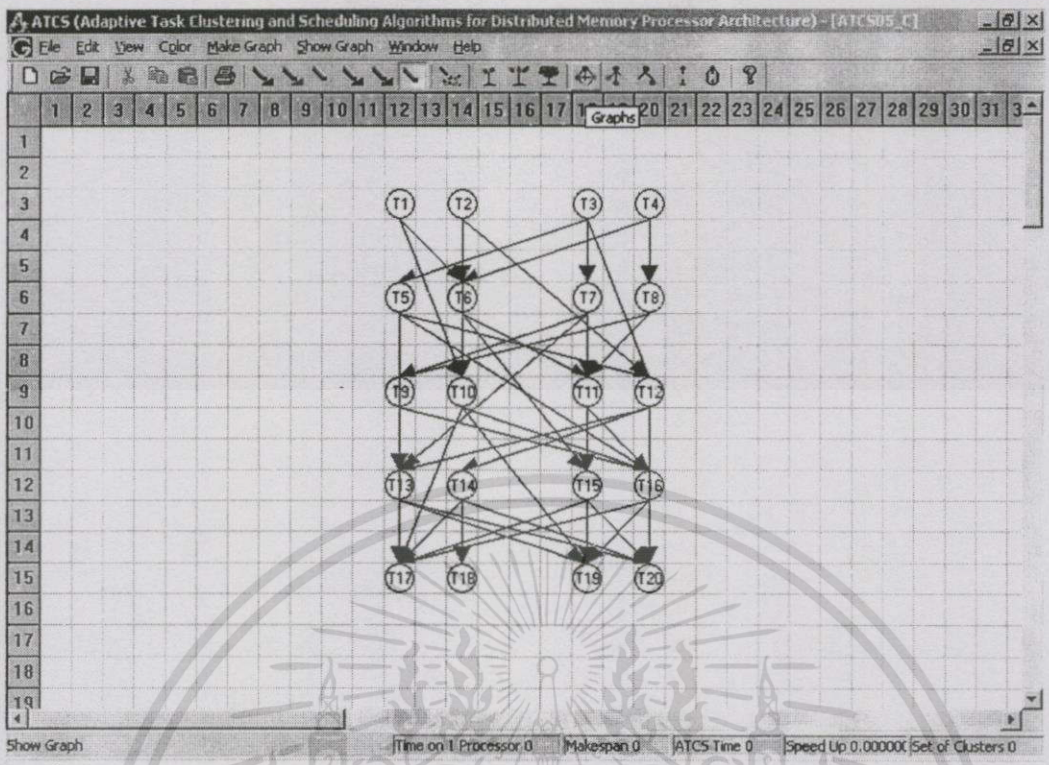
ตารางที่ 5.7 แสดงค่าใช้จ่ายในการประมวลผล(μ) ระดับของโหนด และค่าใช้จ่ายในการติดต่อสื่อสาร(λ)ระหว่างโหนดต้นทางไปยังโหนดปลายทางบน DAG รูปที่ 5.48

Task No.	μ	Level
T1	12	0
T2	10	0
T3	8	0
T4	10	0
T5	8	0
T6	11	1
T7	9	1
T8	12	1
T9	12	1
T10	10	2
T11	11	2
T12	11	2
T13	9	2
T14	8	3
T15	9	3
T16	9	3
T17	8	3
T18	10	3
T19	12	4
T20	10	4
T21	10	4
T22	12	4

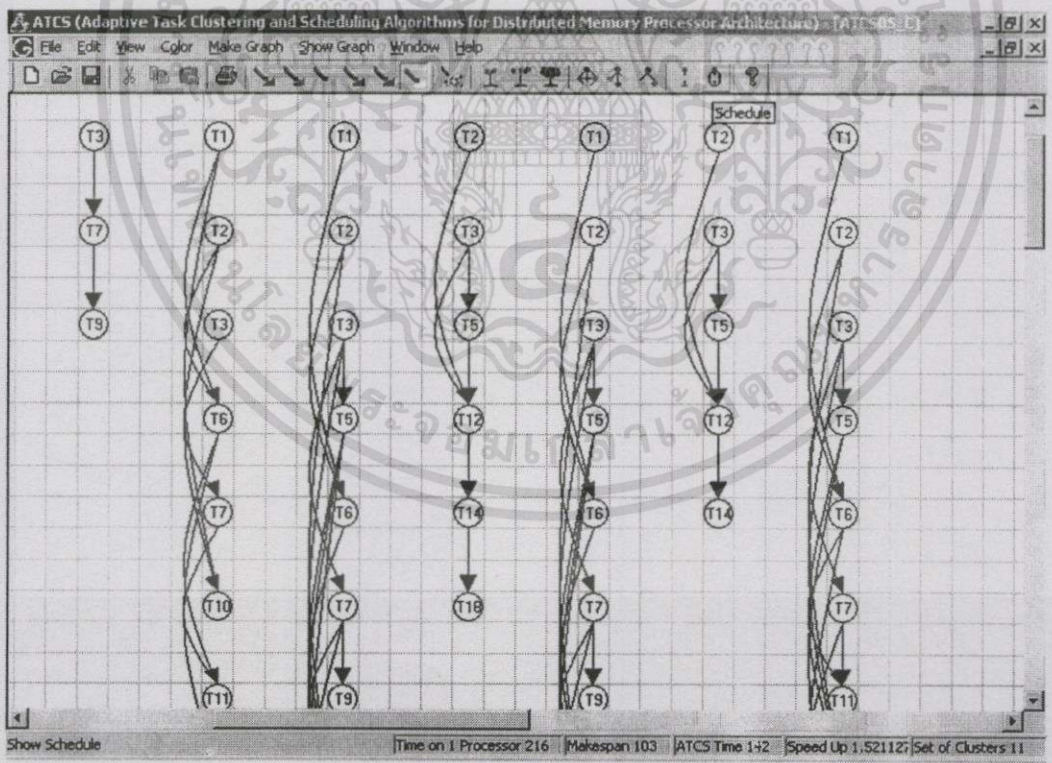
Source Node	Destination Node	λ
1	10	8
1	12	7
1	6	9
2	8	9
2	10	10
3	10	7
3	7	8
3	6	7
4	12	7
4	7	8
4	11	6
4	9	8
5	11	8
5	12	6
5	10	5
6	12	11
6	14	8
7	12	9
7	15	9
8	15	5
8	14	6
9	17	8
9	13	7
10	16	7
10	19	7
11	22	8
11	15	7
12	15	9
12	20	10
12	18	7
13	21	8
13	16	9
13	22	8
14	20	7
14	22	6
15	22	9
15	21	5

ตารางที่ 5.7(ต่อ) แสดงค่าใช้จ่ายในการประมวลผล(μ) ระดับของโหนด และค่าใช้จ่ายในการติดต่อสื่อสาร(λ)ระหว่างโหนดต้นทางไปยังโหนดปลายทางบน DAG รูปที่ 5.48

Source Node	Destination Node	λ
16	21	5
16	20	9
16	22	8
17	21	5
17	19	8
18	19	7
18	20	10



รูปที่ 5.49(a) DAG ชุดที่4 เมื่อค่าระหว่าง μ และ λ มีความแตกต่างกันไม่มาก



รูปที่ 5.49(b) ผลลัพธ์จากการใช้เอทีซีเอสอัลกอริทึมด้วย DAG ชุดที่4 โดยเวลาของ ATCS Time มากกว่าเวลาของ Makespan

ตารางที่ 5.8 แสดงค่าใช้จ่ายในการประมวลผล(μ) ระดับของโหนด และค่าใช้จ่ายในการติดต่อสื่อสาร(λ)ระหว่างโหนดต้นทางไปยังโหนดปลายทางบน DAG รูปที่ 5.49

Task No.	μ	Level
T1	10	0
T2	12	0
T3	13	0
T4	8	0
T5	11	1
T6	10	1
T7	12	1
T8	8	1
T9	11	2
T10	13	2
T11	12	2
T12	9	2
T13	8	3
T14	12	3
T15	11	3
T16	12	3
T17	10	4
T18	11	4
T19	13	4
T20	10	4

Source Node	Destination Node	λ
1	6	10
1	10	8
2	6	9
2	12	8
2	10	9
3	7	9
3	12	5
3	5	6
4	6	8
4	8	6
5	13	7
5	11	12
5	9	8
6	15	6
6	11	8
7	13	7
7	11	12
7	9	8
8	9	8
8	11	5
9	13	6
9	16	1
9	17	6
10	17	10
10	16	7
11	15	6
11	16	12
12	13	5
12	14	5
12	20	6
13	20	7
13	19	6
13	17	6
14	20	6
14	18	12
14	17	9
15	17	10

ตารางที่ 5.8(ต่อ) แสดงค่าใช้จ่ายในการประมวลผล(μ) ระดับของโหนด และค่าใช้จ่ายในการติดต่อสื่อสาร(λ)ระหว่างโหนดต้นทางไปยังโหนดปลายทางบน DAG รูปที่ 5.49

Source Node	Destination Node	λ
15	19	10
15	20	8
16	17	6
16	19	8

5.3 ประสิทธิภาพของเอทีซีเอสอัลกอริทึม

การแสดงประสิทธิภาพในการทำงานของเอทีซีเอสอัลกอริทึม โดยจะทำการเปรียบเทียบผลลัพธ์ที่ได้จากเอทีซีเอสอัลกอริทึมกับอัลกอริทึม[23] และ[32] ตามลำดับ ในการแสดงการเปรียบเทียบจะแสดงผลลัพธ์อยู่ในรูปของแกรฟชาร์ท เพื่อดูเวลาที่หยุดนิ่งที่ปรากฏอยู่บนตัวประมวลผล และการหาค่าของ $E(p_1, p_2)$ ดังที่ได้เคยแสดงเอาไว้ในบทที่ 5 เมื่อกำหนดให้ p_1 เป็นผลลัพธ์ที่ได้จากวิธีการของอัลกอริทึมที่ถูกเปรียบเทียบ และ p_2 เป็นผลลัพธ์ที่ได้จากวิธีการของอัลกอริทึมที่ต้องการจะเปรียบเทียบ ในส่วนนี้จะไม่มีการแสดงประสิทธิภาพของเอทีซีเอสอัลกอริทึมด้วย DAG ที่มีลักษณะเม็ดเนื้องานแบบหยาบเพราะว่า วิธีการของเอทีซีเอสอัลกอริทึมไม่สามารถทำการจัดกลุ่มงานและกำหนดลำดับการทำงานสำหรับ DAG ที่มีลักษณะเม็ดเนื้องานแบบหยาบได้

5.3.1 ประสิทธิภาพของเอทีซีเอสอัลกอริทึมด้วย DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด ในการเปรียบเทียบประสิทธิภาพของเอทีซีเอสอัลกอริทึมด้วย DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียด จะทำการเปรียบเทียบกับอัลกอริทึม[23] เนื่องจากอัลกอริทึม[23] สามารถสร้างกลุ่มงานและกำหนดลำดับการทำงานของโปรแกรมแบบขนานได้อย่างมีประสิทธิภาพอีกวิธีการหนึ่ง ดังนั้นกำหนดให้ p_{ATCS} เป็นการแสดงผลลัพธ์ที่ได้จากวิธีการของเอทีซีเอสอัลกอริทึม และ $p_{[23]}$ เป็นการแสดงผลลัพธ์ที่ได้จากวิธีการของอัลกอริทึม[23](หรือวิธีการของ TCS อัลกอริทึมในบทที่ 3) โดยจะไม่ทำการเปรียบเทียบประสิทธิภาพของเอทีซีเอสอัลกอริทึมกับอัลกอริทึม[32] เพราะว่าอัลกอริทึม[32]ไม่สามารถทำการจัดกลุ่มงานและกำหนดลำดับการทำงานสำหรับโปรแกรมแบบขนานหรือ DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียดได้อย่างมีประสิทธิภาพ

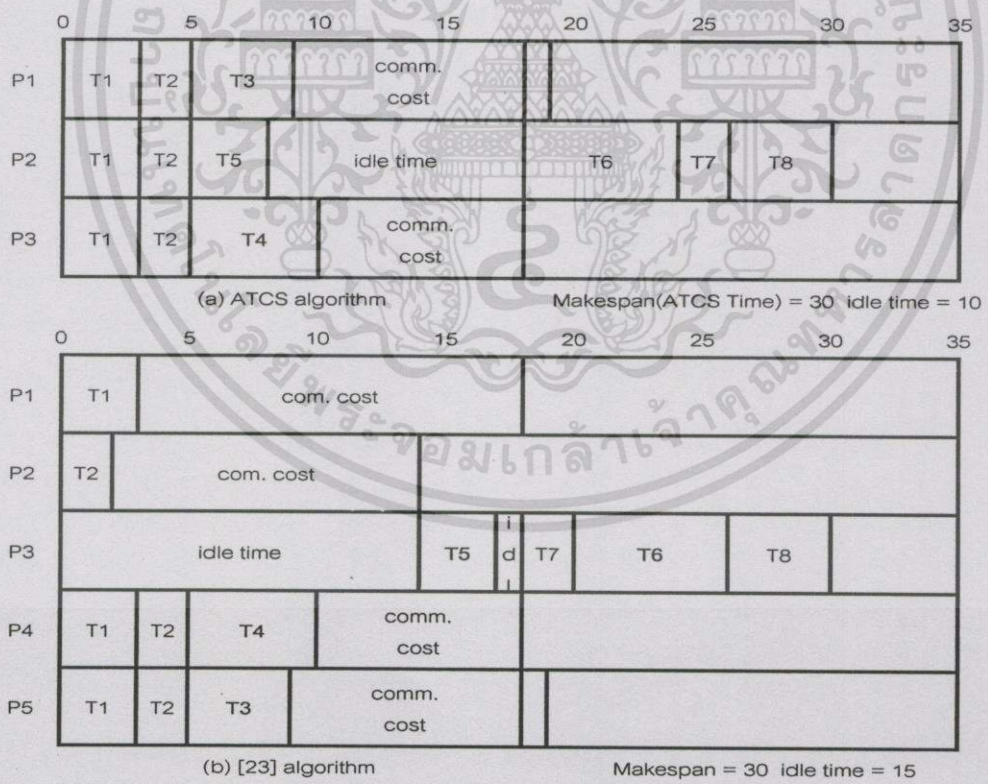
นอกจากนี้จะทำการเปรียบเทียบประสิทธิภาพระหว่างอัลกอริทึมการจัดลำดับงานแบบเรียงตามลำดับ(Sequential algorithm) กับวิธีการของเอทีซีเอสอัลกอริทึม โดยกำหนดให้ p_{seq} เป็นการแสดงผลลัพธ์ที่ได้จากวิธีการของอัลกอริทึมการจัดลำดับงานแบบเรียงตามลำดับ

รูปที่ 5.50 แสดงแกรฟทชาร์ทที่ได้มาจากวิธีการของเอทีซีเอสอัลกอริทึม และอัลกอริทึม[23] ด้วยการใช้ DAG ในรูปที่ 5.11 เป็นอินพุทของอัลกอริทึมทั้งสอง การแสดงของแกรฟทชาร์ทในรูปที่ 5.50 ได้แสดงให้เห็นถึงลำดับการทำงานของงานต่างๆที่อยู่ในกลุ่มงานนั้น ดังปรากฏอยู่บนตัวประมวลผลแต่ละตัว นอกจากนี้ยังแสดงให้เห็นว่าวิธีการของเอทีซีเอสอัลกอริทึมมีจำนวนของเวลาที่หยุดนิ่งน้อยกว่าวิธีการของอัลกอริทึม[23] เอทีซีเอสอัลกอริทึมมีจำนวนของเวลาที่หยุดนิ่งเท่ากับ 10 ในขณะที่วิธีการของอัลกอริทึม[23]มีจำนวนของเวลาที่หยุดนิ่งเท่ากับ 15 แต่พบว่าวิธีการของอัลกอริทึมทั้งสองให้เวลาที่ใช้ในการประมวลผลเท่ากัน หลังจากที่มีการใช้อัลกอริทึมของแต่ละวิธีการเข้าทำการจัดกลุ่มงานและกำหนดลำดับการทำงานให้กับ DAG ต่อไปจะทำการหาค่าของ $E(p_{ATCS}, p_{[23]})$ ของอัลกอริทึมทั้งสองดังนี้

$$E(p_{ATCS}, P_{[23]}) = \frac{p_{ATCS} * (time_{ATCS})}{p_{[23]} * (time_{[23]})}$$

$$= \frac{3*30}{5*30} = \frac{3}{5}$$

จากผลลัพธ์ดังกล่าวแสดงให้เห็นว่า วิธีการของเอทีซีเอสอัลกอริทึมจะให้ประสิทธิภาพของค่าใช้จ่ายในการใช้ตัวประมวลผลที่เหมาะสมมากกว่าวิธีการของอัลกอริทึม[23]



รูปที่ 5.50 แสดงแกรฟทชาร์ทของเอทีซีเอสอัลกอริทึม และอัลกอริทึม[23]

รูปที่ 5.51 แสดงแกรฟชาร์ทของการจัดวางเรียงลำดับงานของอัลกอริทึมการจัดลำดับงานแบบเรียงตามลำดับงาน โดยใช้ DAG ในรูปที่ 5.11 เป็นงานที่จะถูกวางจัดเรียงอยู่บนเครื่องคอมพิวเตอร์ที่มีตัวประมวลผลเพียงตัวเดียว ดังนั้นค่าของ $E(p_{seq}, p_{ATCS})$ เท่ากับ

$$E(p_{seq}, p_{ATCS}) = \frac{p_{seq} * (time_{seq})}{p_{ATCS} * (time_{ATCS})}$$
$$= \frac{1*29}{3*30}$$

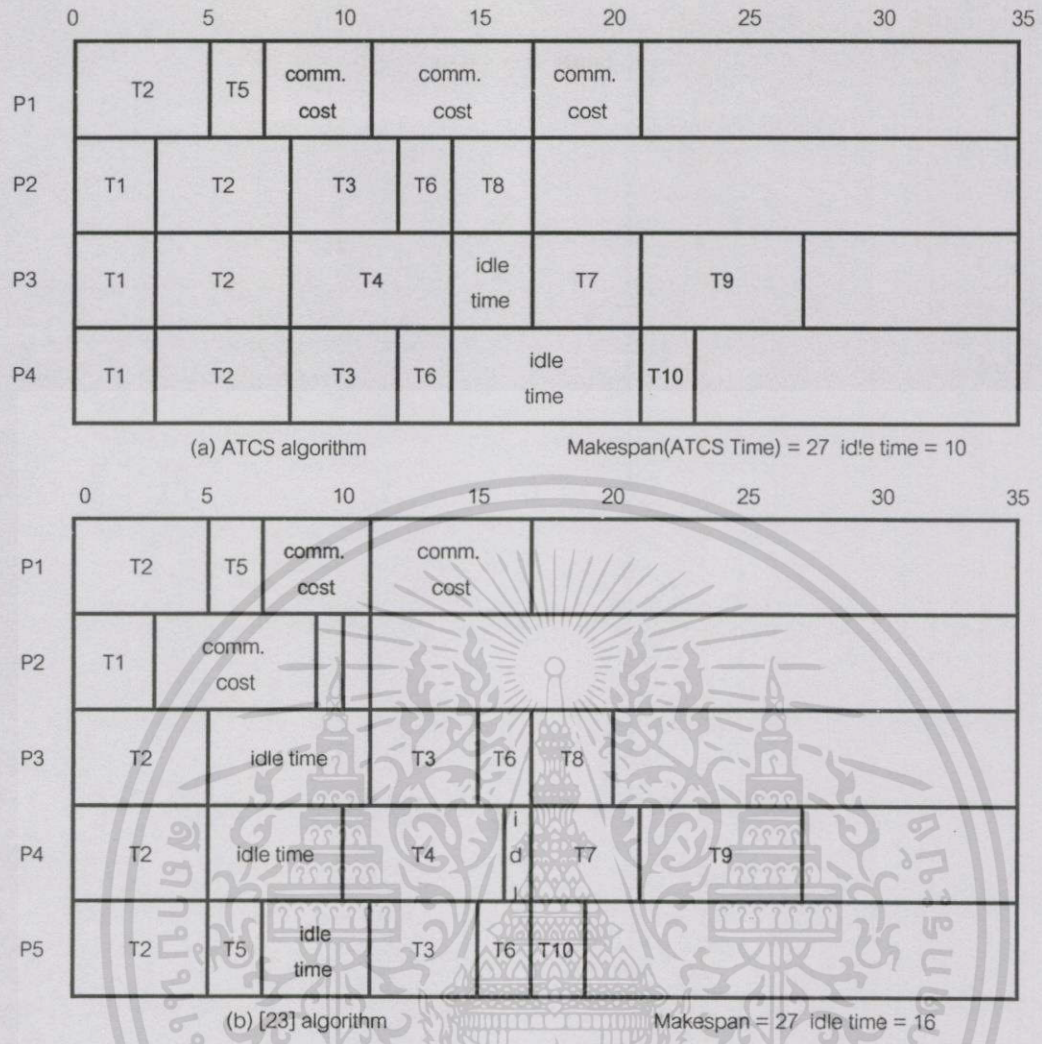
จากผลลัพธ์ของค่า $E(p_{seq}, p_{ATCS})$ แสดงให้เห็นว่า DAG ในรูปที่ 5.11 เหมาะแก่การที่นำเอาไปประมวลผลบนเครื่องคอมพิวเตอร์ที่มีตัวประมวลผลเพียงหนึ่งตัว ซึ่งจะเหมาะกว่าที่นำไปประมวลผลบนเครื่องคอมพิวเตอร์ที่มีตัวประมวลผลมากกว่าหนึ่งตัว

รูปที่ 5.51 แสดงแกรฟชาร์ทของอัลกอริทึมการจัดลำดับงานแบบเรียงตามลำดับสำหรับ DAG ในรูปที่ 5.11

รูปที่ 5.52 แสดงแกรฟชาร์ทที่ได้มาจากวิธีการของเอทีซีเอสอัลกอริทึม และวิธีการของอัลกอริทึม[23] ด้วยการใช้ DAG ในรูปที่ 5.12จากรูปแสดงให้เห็นว่าวิธีการของเอทีซีเอสอัลกอริทึมยังคงมีจำนวนของเวลาที่หยุดนิ่งน้อยกว่าวิธีการของอัลกอริทึม[23] ต่อไปจะทำการวิเคราะห์หาค่าของ $E(p_{ATCS}, p_{[23]})$ ของอัลกอริทึมทั้งสองดังนี้

$$E(p_{ATCS}, p_{[23]}) = \frac{p_{ATCS} * (time_{ATCS})}{p_{[23]} * (time_{[23]})}$$
$$= \frac{4*27}{5*27} = \frac{4}{5}$$

จากผลลัพธ์ดังกล่าวบอกให้ทราบว่า วิธีการของเอทีซีเอสอัลกอริทึมจะให้ประสิทธิภาพของค่าใช้จ่ายในการใช้ตัวประมวลผลที่เหมาะสมกว่าวิธีการของอัลกอริทึม[23]



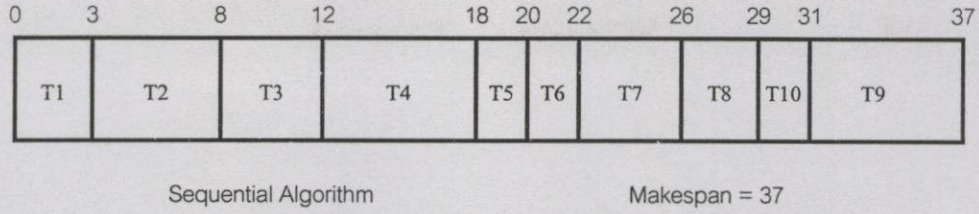
รูปที่ 5.52 แสดงแกรฟทชาร์ฟทของเอทีซีเอสอัลกอริทึม และอัลกอริทึม[23]

รูปที่ 5.53 แสดงแกรฟทชาร์ฟทของการจัดลำดับงานของ DAG ในรูปที่ 5.12 โดยใช้อัลกอริทึมการจัดลำดับงานแบบเรียงตามลำดับ ดังนั้นค่าของ $E(p_{seq}, p_{ATCS})$ มีค่าเท่ากับ

$$E(p_{seq}, p_{ATCS}) = \frac{p_{seq} * (time_{seq})}{p_{ATCS} * (time_{ATCS})}$$

$$= \frac{1*37}{4*27}$$

จากค่าของ $E(p_{seq}, p_{ATCS})$ แสดงให้เห็นว่าประสิทธิภาพในการทำงานของเอทีซีเอสอัลกอริทึม จะด้อยกว่าวิธีการของอัลกอริทึมการจัดลำดับงานแบบเรียงตามลำดับ เนื่องมาจากวิธีการของเอทีซีเอสอัลกอริทึมจะใช้จำนวนของตัวประมวลผลที่มากกว่า



รูปที่ 5.53 แสดงแกรฟทชาร์ทของอัลกอริทึมการจัดลำดับงานแบบเรียงตามลำดับสำหรับ DAG ในรูปที่ 5.12

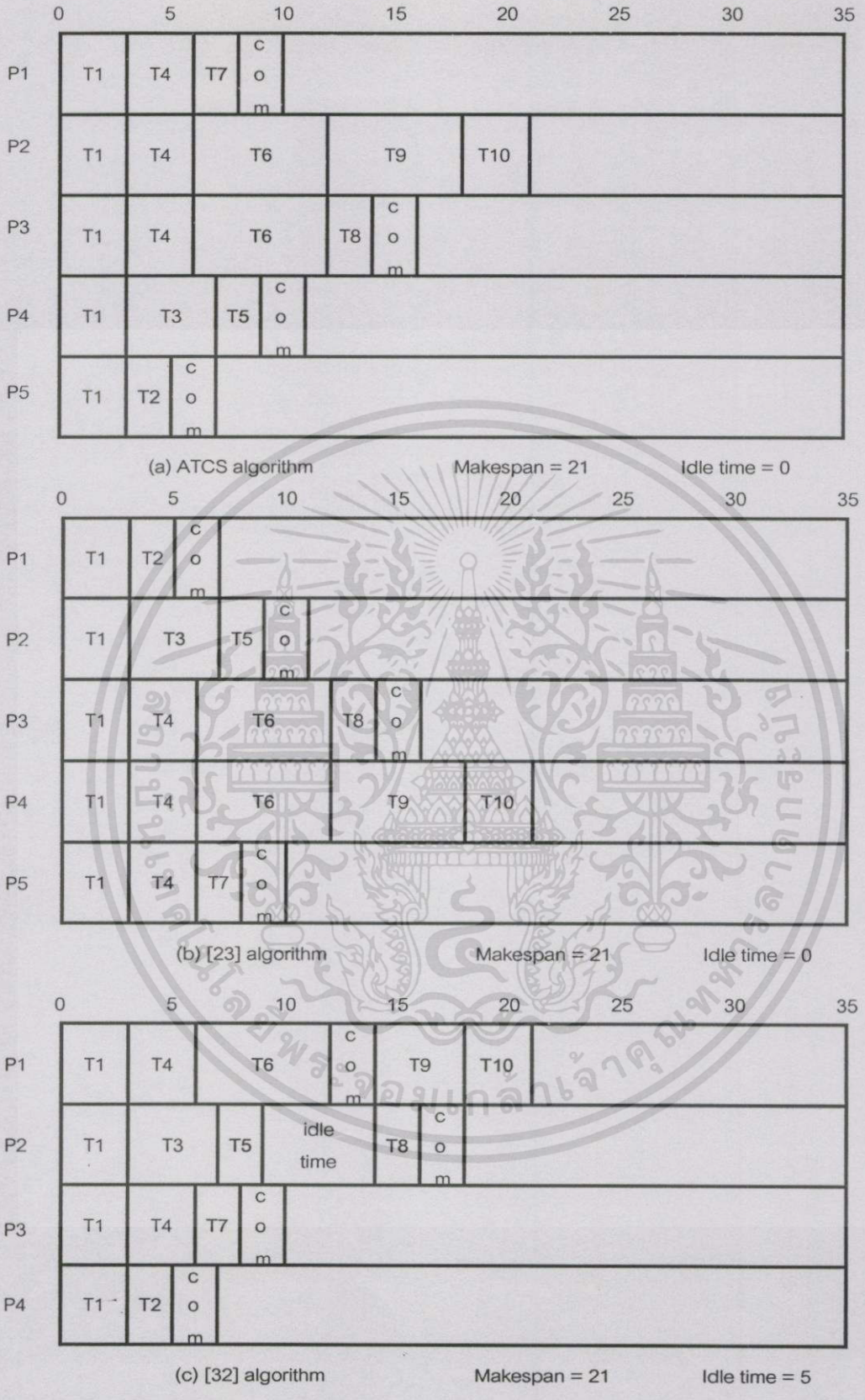
5.3.2 ประสิทธิภาพของเอทีซีเอสอัลกอริทึมด้วย DAG ที่มีลักษณะแบบอื่นๆ การเปรียบเทียบประสิทธิภาพของเอทีซีเอสอัลกอริทึมด้วย DAG ที่มีลักษณะนอกเหนือไปจากนิยามของ[2] โดยจะทำการเปรียบเทียบกับอัลกอริทึม [23] และ[32] ตามลำดับ กำหนดให้ p_1 เป็นการแสดงผลลัพธ์ที่ได้จากวิธีการของเอทีซีเอสอัลกอริทึม และกำหนดให้ p_2 เป็นการแสดงผลลัพธ์ที่ได้จากวิธีการของอัลกอริทึม[23] และ[32]

รูปที่ 5.54 แสดงแกรฟทชาร์ทของเอทีซีเอสอัลกอริทึม อัลกอริทึม[23] และอัลกอริทึม[32] โดยการใช้ DAG ในรูปที่ 5.45 เป็นโจทย์ในการแก้ไขปัญหาของอัลกอริทึมทั้งสามวิธีการดังกล่าว จากแกรฟทชาร์ทแสดงให้เห็นว่า วิธีการของเอทีซีเอสอัลกอริทึมจะให้ผลลัพธ์ที่เหมือนกับวิธีการของอัลกอริทึม[23] หากต้องการทำการเปรียบเทียบประสิทธิภาพของวิธีการทั้งสองโดยการหาค่าของ $E(p_{ATCS}, p_{[23]})$ จะพบว่าจะให้ค่าที่เท่ากัน หากทำการพิจารณาแกรฟทชาร์ทของผลลัพธ์ที่ได้จากวิธีการของอัลกอริทึม[32] จะพบว่าจำนวนของตัวประมวลผลที่ต้องการใช้จะน้อยกว่าวิธีการของเอทีซีเอสอัลกอริทึมและอัลกอริทึม[23] แต่เวลาที่ใช้ในการประมวลผลจะเท่ากันหมดทั้งสามวิธีการ ในขณะเดียววิธีการของอัลกอริทึม[32]จะใช้เวลาที่หยุดนิ่งมากกว่าวิธีการของเอทีซีเอสอัลกอริทึมและอัลกอริทึม[23] ต่อไปทำการพิจารณาหาค่า $E(p_{ATCS}, p_{[32]})$ ของอัลกอริทึมทั้งสองดังนี้

$$E(p_{ATCS}, p_{[32]}) = \frac{p_{ATCS} * (time_{ATCS})}{p_{[32]} * (time_{[32]})}$$

$$= \frac{5*21}{4*21} = \frac{5}{4}$$

จากค่าของผลลัพธ์ดังกล่าวแสดงให้เห็นว่า วิธีการของอัลกอริทึม[32]ให้ประสิทธิภาพในการทำงานสำหรับการจัดกลุ่มงานและกำหนดลำดับการทำงานได้ดีกว่าวิธีการของเอทีซีเอสอัลกอริทึม สำหรับ DAG ของโปรแกรมแบบขนานที่มีลักษณะดังกล่าว โดยวิธีการของอัลกอริทึม[32]จะให้ประสิทธิภาพของค่าใช้จ่ายในการใช้ตัวประมวลผลที่เหมาะสมมากกว่าวิธีการของเอทีซีเอสอัลกอริทึม และค่าของ $E(p_{[23]}, p_{[32]})$ จะให้ผลลัพธ์เช่นเดียวกันกับ $E(p_{ATCS}, p_{[32]})$

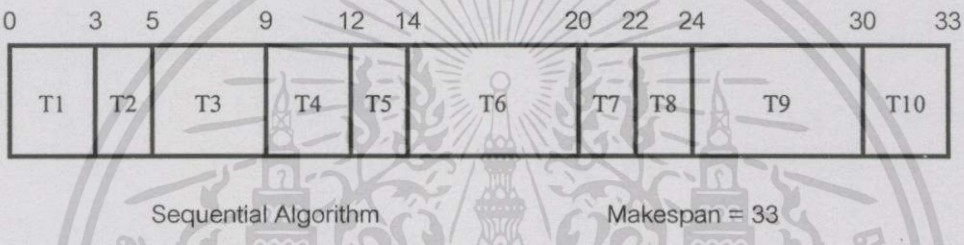


รูปที่ 5.54 แสดงแกรฟชาร์ทของเอทีซีเอสอัลกอริทึม อัลกอริทึม[23] และอัลกอริทึม[32]

นอกจากนี้เราจะทำการเปรียบเทียบประสิทธิภาพในการทำงาน ระหว่างเอทีซีเอสอัลกอริทึมกับอัลกอริทึมในการจัดลำดับงานแบบเรียงตามลำดับ รูปที่ 5.55 เป็นการแสดงแกรฟชาร์ฟของอัลกอริทึมการจัดลำดับงานแบบเรียงตามลำดับ โดยใช้ DAG ในรูปที่ 5.45 เป็นโจทย์ของปัญหา ดังนั้นค่าของ $E(p_{seq}, p_{ATCS})$ เท่ากับ

$$E(p_{seq}, p_{ATCS}) = \frac{p_{seq} * (time_{seq})}{p_{ATCS} * (time_{ATCS})}$$
$$= \frac{1*33}{5*21}$$

จากค่าของ $E(p_{seq}, p_{ATCS})$ เมื่อพิจารณาโดยรวมแล้ว ค่าดังกล่าวแสดงให้เห็นว่าประสิทธิภาพในการทำงานของเอทีซีเอสอัลกอริทึมด้อยกว่าวิธีการของอัลกอริทึมการจัดลำดับงานแบบเรียงตามลำดับ



รูปที่ 5.55 แสดงแกรฟชาร์ฟของอัลกอริทึมการจัดลำดับงานแบบเรียงตามลำดับสำหรับ DAG ในรูปที่ 5.45

5.4 สรุปผลการทดลอง

จากผลการทดลองด้วยลักษณะของ DAG ในหลายๆรูปแบบ ทั้งขนาดของจำนวนของโหนดที่อยู่บน DAG มีขนาดต่างๆกัน แสดงให้เห็นว่าวิธีการของเอทีซีเอสอัลกอริทึมสามารถทำการจัดกลุ่มงานและกำหนดลำดับการทำงานให้กับโปรแกรมแบบขนานสำหรับลักษณะเม็ดเนื้องานแบบละเอียดได้อย่างมีประสิทธิภาพ หากลักษณะของ DAG ดังกล่าวจะต้องมีการกระจายของภาระงานที่เท่าๆกันของโหนดต่างๆที่ปรากฏอยู่บน DAG นั้น โดยสังเกตได้จากค่าของความแตกต่างระหว่าง ค่าของ Makespan และค่าของ ATCS Time จากลักษณะของ DAG บางรูปแบบที่มีการกระจายของภาระงานที่ไม่เท่ากันแสดงเห็นว่าค่าของความแตกต่างระหว่างค่าของ Makspan และค่าของ ATCS Time มีความแตกต่างกันน้อยมาก แสดงให้เห็นว่าประสิทธิภาพของเอทีซีเอสอัลกอริทึมจะลดลงอันเนื่องมาจากลักษณะของ DAG ที่มีการกระจายภาระงานไม่เท่ากัน นอกจากนี้ DAG บางลักษณะที่มีความหนาแน่นของอาร์คมากๆจะทำให้ประสิทธิภาพของเอทีซีเอสอัลกอริทึมจะลดลงได้เช่นเดียวกัน ส่วนลักษณะของ DAG ที่มีการกระจายของภาระงานที่สม่ำเสมอเท่าๆกัน และมี

จำนวนความหนาแน่นของอาร์คที่ไม่มาก ลักษณะดังกล่าวจะทำให้เอทีซีเอสอัลกอริทึมสามารถทำงานได้อย่างมีประสิทธิภาพ

ผลของการทดลองของเอทีซีเอสอัลกอริทึมด้วย DAG ที่มีลักษณะเม็ดเนื้องานแบบละเอียดที่มีจำนวนของโหนดขนาดมากกว่า 100 โหนด ดังแสดงอยู่ในรูปที่ 5.34-5.43 แสดงให้เห็นว่า วิธีการของเอทีซีเอสอัลกอริทึมยังสามารถทำการจัดกลุ่มงานและกำหนดลำดับการทำงานได้เช่นกัน ถ้าหากลักษณะของ DAG ยังคงมีการกระจายภาระงานของโหนดต่างๆได้อย่างเท่าๆกัน จากการทดลองหาผลลัพธ์ประสิทธิภาพของเอทีซีเอสอัลกอริทึมด้วย DAG ที่มีจำนวนมาก 100 โหนด แสดงให้เห็นว่าวิธีการของเอทีซีเอสอัลกอริทึมยังสามารถทำงานได้อยู่ในระดับหนึ่ง ความสามารถของเอทีซีเอสอัลกอริทึมอาจจะลดลง หากจำนวนของระดับชั้นของ DAG มีมากยิ่งขึ้น แม้ว่าลักษณะ DAG ดังกล่าวจะมีการกระจายภาระงานของโหนดต่างๆ ก็ตาม เพราะโหนดต่างๆที่อยู่บน DAG ยังคงมีความสัมพันธ์ต่อกันในลักษณะของลำดับการทำงานของงานต่างๆ

ที่กล่าวมาเป็นการเปรียบเทียบผลลัพธ์ของความแตกต่างระหว่างค่าของ Makespan และค่าของ ATCS Time ค่าดังกล่าวทั้งสองค่าไม่ได้สอดคล้องกับค่าของ Speed Up ที่เพิ่มขึ้นหรือลดลง โดยที่ค่าของ Speed Up เป็นการเปรียบเทียบอัตราส่วนระหว่างค่าของ Time on 1 Processor หรือค่าของเวลาที่ใช้ในการประมวลผลของงานต่างๆที่อยู่บนตัวประมวลผลเพียงตัวเดียว กับค่าของ ATCS Time หรือค่าของเวลาที่ใช้ในการประมวลผลของงานต่างๆที่อยู่บนตัวประมวลผลมากกว่าหนึ่งตัว จากผลของการทดลองจะพบว่า เมื่อลักษณะของ DAG ที่ใหญ่มากขึ้นหรือมีลักษณะโครงสร้างที่มีความสลับซับซ้อนเพิ่มมากขึ้น ค่าของ Time on 1 Processor และ Speed Up กลับมีค่าที่เพิ่มมากขึ้น ดังนั้นเราสามารถบอกได้ว่าเมื่อขนาดของ DAG มีขนาดใหญ่ขึ้นหรือมีความสลับซับซ้อนเพิ่มมากขึ้นไม่ได้ทำให้ค่าของ Speed Up ลดต่ำลง จากค่าของ Speed Up บ่งบอกให้ทราบว่าเมื่องานของโปรแกรมแบบขนานมีโครงที่ใหญ่มากขึ้น การนำเอาอัลกอริทึมในการจัดลำดับงานแบบขนานเข้ามาช่วย ทำให้เวลาที่ใช้ในการประมวลผลลดลงได้อย่างมีประสิทธิภาพ ในทางตรงกันข้าม DAG ที่มีลักษณะโครงสร้างไม่ใหญ่ ไม่เหมาะในการที่จะนำเอาอัลกอริทึมในการจัดลำดับการทำงานแบบขนานมาใช้กับ DAG ที่มีจำนวนของงานไม่มากและมีความสลับซับซ้อนน้อยๆ จากการทดลองของ DAG ที่ได้จากสุ่มจะพบว่า เมื่อ DAG ที่มีขนาดใหญ่หรือมีจำนวนของงานมากๆ ค่า Time on 1 Processor จะมากตามไปด้วย ในขณะเดียวกันค่าของ Time on 1 Processor จะแตกต่างไปจากค่าของ Makespan และค่าของ ATCS Time ค่อนข้างมาก

ในกรณีของ DAG ของโปรแกรมแบบขนานที่มีลักษณะของเม็ดเนื้องานแบบหยาบ เอทีซีเอสอัลกอริทึมไม่สามารถทำการจัดกลุ่มงานและกำหนดลำดับการทำงานได้อย่างมีประสิทธิภาพ หากแต่ว่าวิธีการของเอทีซีเอสอัลกอริทึมสามารถทำงานได้อย่างมีประสิทธิภาพสำหรับ DAG ที่มีลักษณะแบบอื่นๆได้ในบางลักษณะ ดังที่ได้เคยกล่าวไว้แล้ว(ค่าความแตกต่างของ μ และ λ ค่อนข้างน้อย) กล่าวคือเอทีซีเอสอัลกอริทึมไม่สามารถทำการจัดกลุ่มงานและกำหนดลำดับการทำงาน

สำหรับ DAG ที่มีขนาดของจำนวนโหนดมากๆ ระดับความลึกของ DAG ต้องไม่เกินกว่า 3 ระดับชั้น และจำนวนของอาร์คที่ชี้ออกจากโหนดใดๆที่อยู่บน DAG ที่มีลักษณะดังกล่าวจะต้องไม่มากไปกว่า 3 อาร์ค โดยสังเกตได้จากผลการทดลองของ DAG ในรูปที่ 5.45-5.47

5.5 บทวิจารณ์

จากการทดลองการจัดกลุ่มงานและกำหนดลำดับการทำงานของเอทีซีเอสอัลกอริทึม ด้วยลักษณะของ DAG แบบต่างๆกันทั้งขนาดของจำนวนระดับชั้นของ DAG และขนาดของจำนวนของโหนด แสดงให้เห็นว่าประสิทธิภาพของเอทีซีเอสอัลกอริทึมจะลดลงด้วยลักษณะของ DAG ดังนี้(โดยพิจารณาจากค่าของ Makespan และค่า ATCS Time) กรณีแรก DAG มีการกระจายของภาระงานของโหนดต่างๆไม่เท่ากัน กรณีที่สองจำนวนของระดับชั้นของ DAG มีขนาดใหญ่มากๆ และกรณีที่สามวิธีการของเอทีซีเอสอัลกอริทึมจะไม่สามารถทำงานได้อย่างมีประสิทธิภาพ หากลักษณะของ DAG ที่มีลักษณะของเม็ดเนื้องานแบบหยาบ จากลักษณะของ DAG ที่เกิดขึ้นทั้งสามกรณีก่อให้เกิดประสิทธิภาพของวิธีการของเอทีซีเอสอัลกอริทึมลดลง ดังนั้นควรจะมีการปรับปรุงการทำงานของเอทีซีเอสอัลกอริทึมให้สามารถทำการจัดกลุ่มงานและกำหนดลำดับการทำงานได้อย่างมีประสิทธิภาพ หากมีลักษณะ DAG ดังกล่าวเกิดขึ้น โดยเฉพาะอย่างยิ่งลักษณะของ DAG ที่มีขนาดใหญ่มากขึ้น

เอกสารอ้างอิง

[1] Hesham E. et. al. **Task Scheduling in Parallel and Distributed Systems.** Prentice Hall. 1994.

[2] Gerasoulis A., Yang T. "On the granularity and clustering of directed acyclic task graphs." IEEE Trans. Parallel and Distributed Systems, vol.4, no.6, June 1993. pp. 686-701.

[3] Hwang K. **Advanced Computer Architecture.** McGraw-Hill. 1993.

[4] Gary J.N. **Centralized and Distributed Operating Systems.** Prentice Hall. 1992.

[5] Stallings W. **Operating Systems.** Second Edition. Prentice Hall International, Inc. 1995.

[6] Balakrishman V.K. **Network Optimization.** CHAPMAN&HALL. 1995.

[7] Lee D., Lee C. and Kim M. "Optimal Task Assignment in Linear Array Networks." IEEE Trans. On Computer, 1992. pp. 877-880.

[8] Tarjan R. "Data structures and network algorithms." SIAM, 1983.

[9] Stone H. "Multiprocessor scheduling with the aid of network flow algorithms." IEEE Trans. Software Engineer, 1977, pp. 85-93.

[10] Lo V. "Heuristic algorithm for task assignment in distributed systems." IEEE Trans. On Computer, vol.41, no.7, 1988. pp. 1384-1397.

[11] Abraham S., Davidson E. "Task assignment using network flow methods for minimizing communication in n-processors systems." Technical report, Center for Supercomputing Research and Development, University of Illinois, 1986.

[12] Ali H., El-Rewini H. "Task allocation in distributed systems: A split-graph model." Journal of Combinatorial Mathematics and Combinatorial Computing, 14, 1993. pp. 15-32.

[13] Golumbic M. **Algorithm Graph Thoery and Perfect Graphs.** Academic Press. 1980.

[14] Hu T.C. "Paralle sequencing and assembly line problems." Operation Research 9, no.6, 1961. pp. 841-848.

[15] Papadimitriou C., Yannakakis M. "Scheduling interval-ordered tasks." SIAM Journal of Comp

[16] Sarkar V. **Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors.** Cambrige, Mass. : MIT Press. 1989.

[17] Kim S., Browne J. "A general approach to mapping of parallel computation upon multiprocessor architectures." Proceeding of the International Conference on Parallel Processing, 1988, pp. 1-8.

[18] Yang T., Gerasoulis A. "A dominant sequence clustering heuristic algorithm for scheduling DAGs on multiprocessors." Technical Report, Department of Computer Science, Rutgers University, 1999.

[19] Gerasoulis A., Yang T. "A Comparison of clustering heuristics for scheduling DAGs on multiprocessors." Journal of Parallel and Distributed Computing, 1992.

[20] Yang T., Gerasoulis A. "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors." IEEE Trans. Parallel and Distributed Systems, vol.5, no.9, Sept. 1994. pp. 951-967.

[21] Kruatrachue B., Lewis T. "Grain size determination for parallel processing." IEEE Software, Jan. 1988. pp. 23-32.

[22] McCreary C., Gill H. "Automatic determination of grain size for efficient parallel processing." Comm. ACM, Sept. 1989. pp. 1073-1078.

[23] Palis M.A., Liou J.C. and Wei D.S.L. "Task Clustering and Scheduling for Distributed Memory Parallel Architectures." IEEE Trans. Parallel and Distributed Systems, vol.7, no.1, Jan. 1996. pp. 46-55.

[24] Piyatamrong B., Ohara S. and Kantakajorn S. "GTCS : A Greedy Task Clustering and Scheduling Algorithms for Distributed Memory Processor Architecture." Proceedings IEEE The Fourth International Conference/Exhibition on High Performance Computing in ASIA-Pacific Region (IEEE HPC-Asia 2000), May 2000. pp. 310-314 .

[25] Papadimitriou C.H., Yannakakis M. "Towards an architecture-indepent analysis of parallel algorithms." SIAM Journal Computing, vol.19, no.2, April 1990. pp. 322-328.

[26] Fredman M.L., Tarjan R.E. "Fibanacii heaps, and their uses in improved network optimization algorithms." Proc. 25th Annual Symposium on Foundations of Computer Science, Oct. 1984. pp. 338-346.

[27] Aaron M.T. et. al. **Data Structures Using C.** Prentice Hall International, Inc. 1990.

[28] Golumbic M. **Algorithm Graph Thoery and Perfect Graphs.** Academic Press. 1980.

[30] Bernard C. **Graphs and Network.** Oxford University Press. 1979.

[31] Ahuja R.K. et. al. **Network flows theory, algorithms, and applications.** Prentice Hall, Inc. 1993.

[32] Darbha S., Agrawal D.P. "Optimal Scheduling Algorithm for Distributed-Memory Machines." IEEE Trans. Parallel and Distributed Systems, vol.9, no.1, Jan. 1998. pp. 87-95.

[33] Selim G.A. **Parallel Computation models and methods.** Prentice Hall. 1997.

ภาคผนวก

ภาคผนวก ก.
ซอสโค้ดเอทีซีเอสอัลกอริทึม

ส่วนนี้เป็นการแสดงซอสโค้ด(Source Code)ของโปรแกรมเอทีซีเอสหรือเอทีซีเอสอัลกอริทึม โดยโปรแกรมเอทีซีเอสจะเขียนด้วย Visual C++ version 6.0 ในส่วนที่เป็นอัลกอริทึมหลักของเอทีซีเอสอัลกอริทึมจะมีหมายเหตุแสดงไว้อย่างชัดเจน โปรแกรมเอทีซีเอสจะถูกแสดงออกเป็นหลายส่วนด้วยกันดังต่อไปนี้

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//              Constants.h File
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#define MAX_MATRIX 1024

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//              Graph.h File
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#if !defined(Graph_h)
#define Graph_h
const UINT VERSION_NUMBER =1;
const COLORREF BLACK  = RGB(0,0,0);
const COLORREF RED    = RGB(255,0,0);
const COLORREF GREEN  = RGB(0,255,0);
const COLORREF BLUE   = RGB(0,0,255);
const COLORREF YELLOW = RGB(255,255,0);
const COLORREF WHITE  = RGB(255, 255, 255);
const COLORREF SELECT_COLOR = RGB(255, 0, 180);
const SHOW_GRAPH  = 1;
const SHOW_TREE   = 2;
const SHOW_BITREE  = 3;
const SHOW_CLUSTER = 4;
const SHOW_ATCS   = 5;

const SON        = 100;
const NEXT       = 101;
const TREE       = 200;
const BI_TREE    = 201;
const CLUSTER    = 202;
const SET_CLUSTER = 203;
```

```
const SET_OF_NODE_IN_CLUSTER = 1000;
const UNION            = 300;
const INTERSECT        = 301;
const INVERSE_INTERSECT  = 302;

#define PI  3.141592653589793
#define SPACE_NODE  999
////////////////////////////////////////////////
//
class CLinkList
{
public:
        int info;
        CLinkList *pNextNode;

        CLinkList()
        { info = 0; }

        ~CLinkList()   { }
};

class CSingleNode
{
public:
        int m_number;
        int m_cost;
        int m_level;
        int m_started;
        int m_finished;
        BOOL m_Source_Node;
};

////////////////////////////////////////////////
//
class CSingleArc
{
public:
        int m_source_node;
        int m_destination_node;
        int m_commu_cost;
};
////////////////////////////////////////////////
//
```

```
class CElement : public CObject
{
DECLARE_SERIAL(CElement)
// attributes
public:
        CRect    m_EnclosingRect;
protected:
        COLORREF  m_Color;
        int      m_Pen;
// operation
public:
        virtual ~CElement() {}       //Virtual destructor
        virtual void Draw(CDC* pDC, CFont* pFont=0, CElement* pElement=0) {} //Virtual draw operation
        virtual void Serialize(CArchive& ar);
        CRect  GetBoundRect();       //Get the bounding rectangle
protected:
        CElement() {}       // Defaul Constructor
};
////////////////////////////////////////////
// CNode is used as temporary vertex before
// fill it on the graph
class CNode : public CElement
{
DECLARE_SERIAL(CNode)
// attribute
public:
        // properties of node or vertex on a graph
        int m_Execute_cost;  // Execution cost
        int m_Task_no;       // Task number of node
        int m_Vertex_level;  // Node Level on a graph of tree
// operation
public:
        CNode(int Task_no, int Execute_cost,int Vertex_level,
                COLORREF aColor);  // Constructor
        virtual void Serialize(CArchive& ar);
        virtual void Draw(CDC* pDC, CFont* pFont=0, CElement* pElement=0);
        // Destructor
        //~CNode() { }
protected:
        CNode() {};  // Default Constructor
};
////////////////////////////////
// class CArc
```

```
class CArc : public CElement
{
DECLARE_SERIAL(CArc)
// attribute
public:
        int m_Commu_cost;     // Communication cost between node or vertex
        int m_Start_vertex;   // Starting node of m_Comm_cost
        int m_End_vertex;     // Ending node of m_Comm_cost
//operation
public:
        CArc(int Start_node, int End_node, int Commu_cost);
        virtual void Serialize(CArchive& ar);
        virtual void Draw(CDC* pDC, CFont* pFont=0, CElement* pElement=0);
protected:
        CArc() {}; //Default Constructor
};
////////////////////////////////////////////////
class CEdge;     // Forward declaration
////////////////////////////////////////////////
//  class CVertex
class CVertex : public CElement
{
DECLARE_SERIAL(CVertex)
public: // attribute
        int     m_Execute_cost;  // Execution cost on node
        int     m_Task_no;       // Task number
        int     m_Vertex_level;  // level vertex on a Graph or tree
        CRect   m_Rect;
        CEdge   *m_First_edge;   // point to edge
        CVertex *m_Next_vertex;  // point to vertex
        BOOL    m_Marked;
public:  // operation
        CVertex(CRect aRect, COLORREF aColor);
        virtual void Serialize(CArchive& ar);
        virtual void Draw(CDC* pDC, CFont* pFont=0, CElement* pElement=0);
        void SetBoundRect(CRect aRect)
        {
                m_EnclosingRect = aRect;
                m_EnclosingRect.NormalizeRect();
        }
protected:
        CVertex() {}; //Default Constructor
};
```

```
//////////////////////////////////////////////
// class Edge
class CEdge : public CElement
{
DECLARE_SERIAL(CEdge)
public: // attribute
        int     m_Commu_cost;  // Communication cost on node edge
        CVertex *m_End_point;  // point to vertex
        CEdge   *m_Next_edge;  // point to edge
        CPoint   m_StartPoint;
        CPoint   m_EndPoint;
        BOOL     m_STATE_POLY;
        CPoint   m_Arrow[3];
        CPoint   m_POLY1;
        CPoint   m_POLY2;
        CPoint   m_POLY3;
        CPoint   m_ar1;
        CPoint   m_ar2;
        CPoint   m_ar3;
public: // operation
        CEdge(CPoint Start, CPoint End);
        virtual void Serialize(CArchive& ar); //Serialize function for CElement
        virtual void Draw(CDC* pDC, CFont* pFont=0, CElement* pElement=0);
        void SetBoundRect(CRect aRect)
        {
            m_EnclosingRect = aRect;
            m_EnclosingRect.NormalizeRect();
        }
protected:
        CEdge() {}; //Default Constructor
};

//////////////////////////////////////////////
// Acyclic Directed Graph
class CDigraph : public CElement
{
DECLARE_SERIAL(CDigraph)

public:
        // Constructure
        CDigraph(int AmountOfVertex, COLORREF ColorVertex);
        CElement* Make_Graph(CElement* pElement);
```

```
        // Joint Graph and new vertex
        void Joint_Vertex(CElement *pGraph, CElement *pVertex, CElement* pElement);
        // Traverse all vertexs on Graph
        BOOL Traverse_Graph(CVertex *pGraph, CVertex *pVertex);
        // To determines whether a pVertex is adjacent to a pGraph
        BOOL Adjacent(CVertex *pGraph, CVertex *pVertex);
        // Find any vertex on Graph
        CElement* Find_Vertex(CElement* pGraph, int Task_no);
        // Add new vertes on to Graph
        CElement* Add_Vertex(CElement* pGraph, CElement* pElement);
        void Delete_Digraph(CElement* pGraph);
protected:
        CVertex *m_Vertex;
        CEdge   *m_Edge;
        int     m_AmountOfVertex;   //a number of vertex of a graph
        int     m_PenWidth;         //pen size
        int     m_Color;            //set current for vertex
        COLORREF m_Color_vertex;
        COLORREF m_Color_edge;
        CElement* Alloc_Vertex(CRect aRect, COLORREF aColor);
        CElement* Alloc_Edge(int aStart, int aEnd);
        BOOL Free_Vertex(CVertex *pVertex);
        BOOL Free_Edge(CEdge *pEdge);
        // Remove edge from Graph
        void Remove_Edge(CVertex *pGraph, CVertex *pVertex);
public:
        virtual void Serialize(CArchive& ar); //Serialize function for CElement
        virtual void Draw(CDC* pDC, CFont* pFont=0, CElement* pElement=0);
protected:
        CDigraph() {}  //Defualt constructor
};

//////////////////////////////////////////
// class tree-node
class CTreenode : public CElement
{
DECLARE_SERIAL(CTreenode)
public:
        int     m_Execute_cost;  // Execution cost on node
        int     m_Task_no;       // Task number
        int     m_Vertex_level;  // level vertex on a Graph or tree
        CRect   m_Rect;
```

```
        CTreenode* m_father;
        CTreenode* m_son;
        CTreenode* m_next;
        int m_commu_cost;  // communication cost from itself to parent node
                           // communication cost at rooted tree that has
                           // always zero.
        int m_CompleteTime;
        CPoint   m_StartPoint; // for draw edge between vertex of a tree
        CPoint   m_EndPoint;
        CPoint   m_Arrow[3];  // define boundary for a head arrow
        CPoint   m_ar1;
        CPoint   m_ar2;
        CPoint   m_ar3;
protected:
public:
        CTreenode(CRect aRect, COLORREF aColor)
        {
                m_Color  = aColor;
                m_father = NULL;
                m_son    = NULL;
                m_next   = NULL;
                m_CompleteTime = 0;
        }//constructor
        virtual void Serialize(CArchive& ar);
        virtual void Draw(CDC* pDC, CFont* pFont=0, CElement* pElement=0);
        // set bound rectangle for a vertex and an edge
        void SetBoundRect(CRect aRect)
        {
                m_EnclosingRect = aRect;
                m_EnclosingRect.NormalizeRect();
        }
protected:
        CTreenode() {}//Default constructor
};

//////////////////////////////////////////
// class CTree
class CTree : public CElement
{
DECLARE_SERIAL(CTree)
// member variable and function operation
public:
        COLORREF  m_Color_Vertex;
```

```
        COLORREF  m_Color_Edge;
        int m_SizeLevel;
        int m_CurrentLevel;
        int m_PrevLevel;
        CElement* MakeTree(CElement* pElement);
        CElement* FindFather(CElement* aVertexTree, CElement* aFather, CElement* aNode);
        CElement* Addson_SortedCommu(CElement* pTree, CElement* pVertex, CElement* pEdge);
        void Traverse_Intree(CElement* pElement);
        void Assign_Position(CElement* pElement, int TYPE);
        void Assign_Edge(CElement* pElement, int TYPE);
        void MakeBinaryTree(CElement *pElement, CElement *pFather, int SON_NEXT);
        void AssignLevel(CElement *pElement, int aLevel);
        void GetCellRect(int row, int col, LPRECT pRect);
        void Deleted_Tree(CElement* pElement);
// member variable and function operation
public:
        int m_Leaf;
        int m_StartGroup;
        int m_Start;
        int m_Current;
        int m_ChildGroup;
// function operation for virtual function
public:
        CTree(COLORREF aColor)
        {
          m_Color_Vertex = aColor;
          m_Color_Edge   = BLACK;
          m_Leaf = 0;
          m_StartGroup = 1;
          m_Start = 0;
          m_Current = 0;
          m_ChildGroup = 0;

          m_CurrentLevel = 0;
          m_PrevLevel = 0;
          m_SizeLevel = 0;   // level size of tree
        } //constructor

        virtual void Serialize(CArchive& ar);
        //Destructor
        ~CTree()
        { };
```

```
protected:
        CElement* Alloc_Treenode(CRect aRect, COLORREF aColor);
        BOOL    Free_Treenode(CTreenode* aNode);
        void    Create_InverseHeadArrow(UINT max1, UINT max2,
                                        CEdge *pEdge, CRect aRect, CPoint aThirdPoint=0);
        CTree() {} //Defautl constructor
};

class CConnect;
//////////////////////////////////////////
// class CTask derives from CVertex
class CTask : public CVertex
{
DECLARE_SERIAL(CTask)
// public memeber variable
public:
        int m_StartedTime;
        int m_CompleteTime;
        CTypedPtrArray<CPtrArray, BOOL*> m_ArrayTaskCluster;
// protected member function
protected:
//function operation for virtual function
public:
        CTask(CRect aRect, COLORREF aColor);
        virtual void Serialize(CArchive& ar);
//protected operation function
protected:
        CTask() {} //Default constructor
};

//////////////////////////////////////////
//
class CConnect : public CElement
{
DECLARE_SERIAL(CConnect)
//public member variable
public:
        CPoint m_StartPoint, m_EndPoint;
        int   m_CommuCost;
//protected member function
protected:
//function operation for virtual function
```

```
public:
        CConnect(CPoint aStart, CPoint aEnd);
        virtual void Serialize(CArchive& ar);
//protect operation function
protected:
        CConnect() {}  //Default constructor
};

////////////////////////////////////////////////////////////////////////////
class CSetTask : public CObject
{
DECLARE_SERIAL(CSetTask)
public:
        BYTE   *m_SetCluster;
        int m_SizeCluster;
public:
        ~CSetTask()
        {
                if(m_SetCluster)
                { delete [ ] m_SetCluster; }
        }
        CSetTask(int pSize)
        {
                m_SizeCluster = pSize;
                m_SetCluster = new BYTE[pSize];
                memset(m_SetCluster, '\0', pSize);
        }
        virtual void Serialize(CArchive& ar)
        {
                CObject::Serialize(ar);
                if(ar.IsStoring())
                {  ar << m_SizeCluster;  }
                else
                {  ar >> m_SizeCluster;  }
        }
        int GetTotalElement()
        {   return m_SizeCluster;   }
protected:
        CSetTask() {}
};
```

```
//////////////////////////////////////////
// Class for Binary Tree
class CNodeBi
{
public:
        int info;
        int makespan;
        int started_time;
        int finished_time;
        CNodeBi *left;
        CNodeBi *right;
public:
        CNodeBi()
        {
                info = 0;
                started_time = 0;
                finished_time = 0;
                makespan = 0;
        }
        ~CNodeBi()
        { }
protected:
};

//////////////////////////////////////////
// class node cluster
// define attribute of node on cluster
class CCluster : public CDigraph
{
DECLARE_SERIAL(CCluster)
// public memeber variable
public:
        CSingleNode  *m_SetTask;
        CSetTask     *m_CSetTask;
        CTypedPtrList<CObList, CSetTask*> m_ArrayTaskCluster;
// member variable and function operation
public:
        CElement* MakeCluster(CElement *pElement);
        void InitSet(BOOL *pSet_Cluster, int size);
        void Allocate_SetOfTask();
        void TraverseTree(CElement* pElement);
        void AdaptiveTaskCluster(CElement *pElement, int &pCommuCost, int &pStartedTime, int &pCompletedTime,
                                                        BOOL *pSetTaskChild);
```

```
        void ScheduleOfTask();
        int  CompletionTime(BOOL *pSetOfTask);
        void OperationSet(BOOL *pSetLeft, BOOL *pSetRight, BOOL *pSetResult, int pOperation);
        BOOL CheckEmptySet(BOOL *pSetLeft, BOOL *pSetRight);
        BOOL CheckMemberSet(BOOL *pSetSource, BOOL *pSetDestinate);
        void Put_TotalTask(int pTotalTask);
        int  Get_TotalTask();
        // array of a set of task cluster
        BOOL Empty_SetOfTaskCluster();
        void Insert_SetOfTaskCluster(BOOL *pSetOfTask);
        CSetTask* Get_SetOfTaskCluster();
        void Del_SetOfTaskCluster(CSetTask *pSetofTask);
        void Assign_NodePosition(CElement* pElement, int pCurrentRow,  int pCurrentColumn);
        void Assign_ArcPosition(CElement* pElement);
        void Debug_Packet(BYTE* pPacket);
        CNodeBi *Alloc_NodeBi();
        CNodeBi *MakeBiTree(int KeyValue);
        BOOL SetLeftBiTree(CNodeBi *pBiTree, int KeyValue);
        BOOL SetRightBiTree(CNodeBi *pBiTree, int KeyValue);
        BOOL InsertNodeOnBiTree(CNodeBi *pRooted, int KeyValue);
        void TraverseBiTree(CNodeBi *pPoint);
        void DeleteBiTree(CNodeBi *pPoint);
        void GetSinkNode_OnBiTree(CNodeBi *pPoint, CSingleNode *pArrayNode, int aSize);
        CLinkList *MakeLinkList(int pInfo);
        CLinkList *InsertToLinkList(CLinkList *pLinkList, int pInfo);
        void    FreeLinkList(CLinkList *pLinkList);
        CLinkList *GetTailLinkList(CLinkList *pLinkList, int &pInfo);
        void    PushMakeSpan(int *pLinkMakeSpan, int info);
        int     PopMakeSpan(int *pLinkMakeSpan);
protected:
        CVertex     *m_Vertex; //CTask derives from CVertex
        CEdge       *m_Edge;
        CLinkList   *m_LinkList;
        COLORREF    m_Color_Edge;
        int         m_Task_Total;
        CElement*   Add_Task(CElement* pCluster, CElement* pElement);
        CElement*   Alloc_Task(CRect aRect, COLORREF aColor);
        BOOL        Free_Task(CTask *pTask);
        int         Get_ExecuteCost(int pTaskNo);
// function operation for virtual function
public:
        CCluster(COLORREF aColor);  //constructor
        virtual void Serialize(CArchive& ar);
```

```
        virtual ~CCluster()
        { if(m_SetTask)  delete [ ] m_SetTask;  }
protected:
        CCluster() {} //Defautl constructor
};
#endif //!defined(Graph_h)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//              Graph.cpp File
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Implementations of the element classes
#include "stdafx.h"  // standard system header files
#include "ATCS.h"
//#include "ATCSDoc.h"
//#include "ATCSView.h"
#include "Graph.h"
#include "ATCSDoc.h"
#include "ATCSView.h"
#include <math.h>

IMPLEMENT_SERIAL(CElement, CObject, VERSION_NUMBER)
IMPLEMENT_SERIAL(CNode, CElement, VERSION_NUMBER)
IMPLEMENT_SERIAL(CArc,  CElement, VERSION_NUMBER)
IMPLEMENT_SERIAL(CVertex,  CElement, VERSION_NUMBER)
IMPLEMENT_SERIAL(CEdge,  CElement, VERSION_NUMBER)
IMPLEMENT_SERIAL(CDigraph, CElement, VERSION_NUMBER)
IMPLEMENT_SERIAL(CTreenode, CElement, VERSION_NUMBER)
IMPLEMENT_SERIAL(CTree, CElement, VERSION_NUMBER)

IMPLEMENT_SERIAL(CTask, CVertex, VERSION_NUMBER)
IMPLEMENT_SERIAL(CConnect, CElement, VERSION_NUMBER)
IMPLEMENT_SERIAL(CCluster, CDigraph, VERSION_NUMBER)
IMPLEMENT_SERIAL(CSetTask, CObject, VERSION_NUMBER)
////////////////////////////////////////////////////////////////
// Get the bounding rectangle for an element
CRect CElement::GetBoundRect()
{       CRect BoundingRect;  //Object to store bounding rectangle
        BoundingRect = m_EnclosingRect;  //Store the enclosing rectangle
        //Increase the rectangle by the pen width
        int offset = m_Pen == 0 ? 1 : m_Pen;   // Width must be at least 1
        BoundingRect.InflateRect(offset, offset);
        return BoundingRect;
}
```

```
void CElement::Serialize(CArchive &ar)
{
        CObject::Serialize(ar);  // call the base class function
        if(ar.IsStoring())
        {       ar << m_Color //Store the color
                  << m_EnclosingRect //and the enclosing rectangle
                  << m_Pen;  //and the pen width
        }
        else{
                ar >> m_Color
                  >> m_EnclosingRect
                  >> m_Pen;
        }
}
//////////////////////////////////////////////////////
// Constructor CNode
CNode::CNode(int Task_no, int Execute_cost, int Vertex_level, COLORREF aColor)
{       m_Execute_cost = Execute_cost;
        m_Task_no     = Task_no;
        m_Vertex_level = Vertex_level;
        m_Pen = 1;
}
//////////////////////////////////////////////////////
// Serialize for CNode
void CNode::Serialize(CArchive &ar)
{       CElement::Serialize(ar);  // call the base class function
        if(ar.IsStoring())
        {       ar  << m_Task_no
                    << m_Execute_cost
                    << m_Vertex_level;
        }
        else
        {       ar  >> m_Task_no
                    >> m_Execute_cost
                    >> m_Vertex_level;
        }
}
//////////////////////////////////////////////////////
// Draw a circle
void CNode::Draw(CDC* pDC, CFont* pFont, CElement* pElement)
{  }
//////////////////////////////////////////////////////
// Constructor for CArc
```

```
CArc::CArc(int Start_node, int End_node, int Commu_cost)
{       m_Commu_cost = Commu_cost;
        m_Start_vertex = Start_node;
        m_End_vertex = End_node;
}
////////////////////////////////////////////////////////////
// Serailize of CArc
void CArc::Serialize(CArchive &ar)
{       CElement::Serialize(ar);  // call the base class function
        if(ar.IsStoring())
        {       ar << m_Start_vertex
                   << m_End_vertex
                   << m_Commu_cost;
        }
        else
        {       ar >> m_Start_vertex
                   >> m_End_vertex
                   >> m_Commu_cost;
        }
}
////////////////////////////////////////////////////////////
void CArc::Draw(CDC* pDC, CFont* pFont, CElement* pElement)
{       }
////////////////////////////////////////////////////////////
CVertex::CVertex(CRect aRect, COLORREF aColor)
{       //First calculate the radius
        //We use floating point because that is required by
        //the library function (in math.h) for calculating a square root.
        //long Radius = (long) sqrt((double)((End.x-Start.x)*(End.x-Start.x)+(End.y-Start.y)*(End.y-Start.y)) );
        //Now calculate the rectangle enclosing
        //the circle assuming the MM_TEXT mapping mode
        //m_EnclosingRect = CRect(Start.x-Radius, Start.y-Radius, Start.x+Radius, Start.y+Radius);
        m_Rect  = aRect;
        m_Color = aColor;  //Set the color for the circle
        m_Marked = FALSE;
}
////////////////////////////////////////////////////////////
void CVertex::Serialize(CArchive &ar)
{       CElement::Serialize(ar);  // call the base class function
        if(ar.IsStoring())
        {       ar << m_Execute_cost
                   << m_Task_no
                   << m_Vertex_level
```

```
                << m_Rect;
        }
        else
        {       ar >> m_Execute_cost
                >> m_Task_no
                >> m_Vertex_level
                >> m_Rect;
        }
}
//////////////////////////////////////////////////////////////////////
void CVertex::Draw(CDC* pDC, CFont* pFont, CElement* pElement)
{       CBrush brush(m_Color);
        pDC->SelectObject(&brush);
        pDC->Ellipse(&m_Rect);
        pDC->SetBkMode(TRANSPARENT);
        CFont* pOldFont = pDC->SelectObject(pFont);
        CString label;

        if(m_Task_no == SPACE_NODE)
                label.Format(_T("SP"));
        else
                label.Format(_T("T%d"), m_Task_no);

        pDC->DrawText(label, m_Rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
        pDC->SelectObject(pOldFont);
}
//////////////////////////////////////////////////////////////////////////
CEdge::CEdge(CPoint Start, CPoint End)
{
        m_StartPoint = Start;           m_EndPoint = End    m_STATE_POLY = FALSE;
        m_ar1.x = 0;        m_ar1.y = 0;        m_ar2.x = 0;        m_ar2.y = 0;
        m_ar3.x = 0;        m_ar3.y = 0;
        m_POLY1.x =0;       m_POLY1.y =0;       m_POLY2.x =0;       m_POLY2.y =0;
        m_POLY3.x =0;       m_POLY3.y =0;
}
//////////////////////////////////////////////////////////////////////////
void CEdge::Serialize(CArchive &ar)
{
        CElement::Serialize(ar);  // call the base class function
        if(ar.IsStoring())
        {       ar  << m_Commu_cost
                    << m_StartPoint
                    << m_EndPoint
```

```
                << m_STATE_POLY
                << m_POLY1
                << m_POLY2
                << m_POLY3
                << m_ar1
                << m_ar2
                << m_ar3;
        }
        else
        {       ar >> m_Commu_cost
                >> m_StartPoint
                >> m_EndPoint
                >> m_STATE_POLY
                >> m_POLY1
                >> m_POLY2
                >> m_POLY3
                >> m_ar1
                >> m_ar2
                >> m_ar3;
        }
}
//////////////////////////////////////////////////////////////////////
void CEdge::Draw(CDC* pDC, CFont* pFont, CElement* pElement)
{       CPen aPen;
        COLORREF aColor = RGB(0,0,0); // BACK COLOR
        m_Pen = 1;
        if(!aPen.CreatePen(PS_SOLID, m_Pen, aColor))
        {       //Pen creation failed. Abort the program
                AfxMessageBox("Pen Creattion failed drawing a line", MB_OK);
                AfxAbort();
        }
        //draw line
        CPen *pOldPen = pDC->SelectObject(&aPen);
        if(m_STATE_POLY)
        {       //pDC->Arc(m_EnclosingRect, m_StartPoint, m_EndPoint);
                POINT aPoint[4] = {m_POLY1.x, m_POLY1.y, m_POLY2.x, m_POLY2.y,
                                m_POLY2.x, m_POLY2.y+10, m_POLY3.x, m_POLY3.y};
                pDC->PolyBezier(aPoint, 4);
        }
        else
        {       pDC->MoveTo(m_StartPoint);
                pDC->LineTo(m_EndPoint);
        }
```

```
        pDC->SelectObject(pOldPen);
        //draw head arrow
        CBrush brush(aColor);
        CBrush* pOldBrush = pDC->SelectObject(&brush);
        m_Arrow[0].x = m_ar1.x;     m_Arrow[0].y = m_ar1.y;
        m_Arrow[1].x = m_ar2.x;     m_Arrow[1].y = m_ar2.y;
        m_Arrow[2].x = m_ar3.x;     m_Arrow[2].y = m_ar3.y;
        pDC->Polygon(m_Arrow,3);
        pDC->SelectObject(pOldBrush);
}
////////////////////////////////////////////////////////////////////////////
void CDigraph::Serialize(CArchive &ar)
{       CElement::Serialize(ar);  // call the base class function
}
////////////////////////////////////////////////////////////////////////////
void CDigraph::Draw(CDC* pDC, CFont* pFont, CElement* pElement)
{       }
////////////////////////////////////////////////////
//Add definifions for member functions here
CDigraph::CDigraph(int AmountOfVertex, COLORREF ColorVertex)
{       m_AmountOfVertex = AmountOfVertex;
        m_Color_vertex  = ColorVertex;
        //Set black color for edge by default value
        m_Color_edge    = BLACK;
}
////////////////////////////////////////////////////
// To Allocate dynamic memory for a new vertex
CElement* CDigraph::Alloc_Vertex(CRect aRect, COLORREF aColor)
{       return new CVertex(aRect, aColor);      }
////////////////////////////////////////////////////
// To Allocate dynamic memory for a new edage
CElement* CDigraph::Alloc_Edge(int aStart, int aEnd)
{       return new CEdge(aStart, aEnd);         }
////////////////////////////////////////////
// To release dynamic memory allocation for a vertex
BOOL CDigraph::Free_Vertex(CVertex *pVertex)
{       if(pVertex != NULL)
        {       delete pVertex;
                return TRUE;
        }
        else
                return FALSE;
}
```

```
//////////////////////////////////////////////////////////////////
// To release dynamic memory allocation for a edge
BOOL CDigraph::Free_Edge(CEdge *pEdge)
{       if(pEdge != NULL)
        {       delete pEdge;
                return TRUE;
        }
        else
                return FALSE;
}
//////////////////////////////////////////////////////////////////
//  To Allocate dinamic memory for a new Graph
CElement* CDigraph::Make_Graph(CElement* pElement)
{       CElement* pVertex;
        // allocation a new vertex for a Graph
        pVertex = Alloc_Vertex(0, m_Color_vertex);
        // attribute
        ((CVertex*)pVertex)->m_Task_no      = ((CNode*)pElement)->m_Task_no;
        ((CVertex*)pVertex)->m_Execute_cost = ((CNode*)pElement)->m_Execute_cost;
        ((CVertex*)pVertex)->m_Vertex_level = ((CNode*)pElement)->m_Vertex_level;
        ((CVertex*)pVertex)->m_First_edge = NULL;
        ((CVertex*)pVertex)->m_Next_vertex = NULL;
        ((CVertex*)pVertex)->m_Marked = FALSE;
        return pVertex;
}
//////////////////////////////////////////////////////////////////
// It will be joint a Global graph and a new vertex to that graph
void CDigraph::Joint_Vertex(CElement *pGraph, CElement *pVertex, CElement* pElement)
{       CElement *m_Edge1, *m_Edge2;
        m_Edge1 = m_Edge2 = NULL;
        m_Edge1 = ((CVertex*)pGraph)->m_First_edge;
        while( m_Edge1 != NULL && ((CEdge*)m_Edge1)->m_End_point != pVertex)
        {       m_Edge2 = m_Edge1;
                m_Edge1 = ((CEdge*)m_Edge1)->m_Next_edge;
        }// end while()
        if(m_Edge1 != NULL)
        {       // The m_Edge represents an edge from a pGraph to a new pVertex.
                //m_Edge1->m_Commu_cost = Weight;
                ((CEdge*)m_Edge1)->m_Commu_cost = ((CArc*)pElement)->m_Commu_cost;
                return;
        }
        // A Edge from a pGraph to a pVertex does exist.
        m_Edge1 = Alloc_Edge(0,0);
```

```
            //m_Edge1->m_Commu_cost = Weight;
            ((CEdge*)m_Edge1)->m_Commu_cost = (((CArc*)pElement)->m_Commu_cost;
            ((CEdge*)m_Edge1)->m_End_point  = (CVertex*)pVertex;
            ((CEdge*)m_Edge1)->m_Next_edge  = NULL;
            (m_Edge2 == NULL) ? (((CVertex*)pGraph)->m_First_edge = (CEdge*)m_Edge1) :
                                                    (((CEdge*)m_Edge2)->m_Next_edge = (CEdge*)m_Edge1);
}
////////////////////////////////////////////////////////
// It will be remove a pVertex from a pGraph
void CDigraph::Remove_Edge(CVertex *pGraph, CVertex *pVertex)
{           CEdge *pEdge1, *pEdge2;
            pEdge1 = pEdge2 = NULL;
            // to traverse by using an edge or an arc
            pEdge1 = pGraph->m_First_edge;
            while(pEdge1 != NULL && pEdge1->m_End_point != pVertex)
            {       pEdge2 = pEdge1;
                    pEdge1 = pEdge1->m_Next_edge;
            }// end while()
            if(pEdge1 != NULL)
            {       // a pEdge1 points to an edge from a pGraph to a pVertex
                    (pEdge2 == NULL) ? (pGraph->m_First_edge = pEdge1->m_Next_edge) :
                                                    (pEdge2->m_Next_edge = pEdge1->m_Next_edge);
                    Free_Edge(pEdge1);
                    return;
            }// End if
}
////////////////////////////////////////////////////////
// It will be adjacent vertex on a Graph
BOOL CDigraph::Adjacent(CVertex *pGraph, CVertex *pVertex)
{           CEdge *Edge;
            Edge = pGraph->m_First_edge;
            while( Edge != NULL)
            {       if(Edge->m_End_point == pVertex)        return TRUE;
                    else                                    Edge = Edge->m_Next_edge;
            }
            return FALSE;
}
////////////////////////////////////////////////////////
// It will be find any vertex on Graph
CElement *CDigraph::Find_Vertex(CElement *pGraph, int Task_no)
{           CElement* pVertex;
            pVertex = pGraph;
            while(pVertex != NULL)
```

```
        {       if( ((CVertex*)pVertex)->m_Task_no == Task_no)
                                return pVertex;
                else
                                pVertex = ((CVertex*)pVertex)->m_Next_vertex;
        }
        return NULL;
}
//////////////////////////////////////////////////////////
// It will be add a pVertex to a pGraph
CElement *CDigraph::Add_Vertex(CElement *pGraph, CElement* pElement)
{       CElement* pVertex;
        // allocation for a new vertex to graph
        pVertex = Alloc_Vertex(0, m_Color_vertex);
        // set attribute
        ((CVertex*)pVertex)->m_Task_no     = ((CNode*)pElement)->m_Task_no;
        ((CVertex*)pVertex)->m_Execute_cost = ((CNode*)pElement)->m_Execute_cost;
        ((CVertex*)pVertex)->m_Vertex_level = ((CNode*)pElement)->m_Vertex_level;
        ((CVertex*)pVertex)->m_First_edge = NULL;
        ((CVertex*)pVertex)->m_Next_vertex = (CVertex*)pGraph;
        pGraph = pVertex;
        // return a new locate for a Graph
        return  pVertex;
}
//////////////////////////////////////////////////////////
void CDigraph::Delete_Digraph(CElement* pGraph)
{       CElement *pPoint;
        CEdge   *pEdge;
        CEdge   *aEdge;
        while(pGraph)
        {       pPoint = pGraph;
                pEdge = ((CVertex*)pPoint)->m_First_edge;
                while(pEdge)
                {       aEdge = pEdge;      pEdge = pEdge->m_Next_edge;
                        delete aEdge;       aEdge = 0;
                }
                pGraph = ((CVertex*)pGraph)->m_Next_vertex;
                delete pPoint;          pPoint = 0;
        }
}
//////////////////////////////////////////////////////////////////////////////
void CTreenode::Serialize(CArchive &ar)
{       CElement::Serialize(ar);  // call the base class function
        if(ar.IsStoring())
```

```
        {       ar << m_Execute_cost
                << m_Task_no
                << m_Vertex_level
                << m_Rect
                << m_commu_cost
                << m_CompleteTime
                << m_StartPoint
                << m_EndPoint
                << m_ar1
                << m_ar2
                << m_ar3;
        }
        else
        {       ar >> m_Execute_cost
                >> m_Task_no
                >> m_Vertex_level
                >> m_Rect
                >> m_commu_cost
                >> m_CompleteTime
                >> m_StartPoint
                >> m_EndPoint
                >> m_ar1
                >> m_ar2
                >> m_ar3;
        }
}
////////////////////////////////////////////////////////////////////////////
void CTreenode::Draw(CDC* pDC, CFont* pFont, CElement* pElement)
{       CBrush brush(m_Color);
        pDC->SelectObject(&brush);
        pDC->Ellipse(&m_Rect);
        pDC->SetBkMode(TRANSPARENT);
        CFont* pOldFont = pDC->SelectObject(pFont);
        CString label;
        if(m_Task_no == SPACE_NODE)
                label.Format(_T("SP"));
        else
                label.Format(_T("T%d"), m_Task_no);
        pDC->DrawText(label, m_Rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
        pDC->SelectObject(pOldFont);
        // Draw Edge
        CPen aPen;
        COLORREF aColor = RGB(0,0,0); // BACK COLOR
```

```
        m_Pen = 1;
        if(!aPen.CreatePen(PS_SOLID, m_Pen, aColor))
        {       //Pen creation failed, Abort the program
                AfxMessageBox("Pen Creattion failed drawing a line", MB_OK);
                AfxAbort();
        }
        //draw line
        CPen *pOldPen = pDC->SelectObject(&aPen);
        pDC->MoveTo(m_StartPoint);
        pDC->LineTo(m_EndPoint);
        pDC->SelectObject(pOldPen);
}
//////////////////////////////////////////////////////////////////////////
void CTree::Serialize(CArchive &ar)
{       CElement::Serialize(ar);  // call the base class function   }
//////////////////////////////////////////////////////////////////////////
CElement* CTree::MakeTree(CElement* pElement)
{       CElement* pTreenode;
        // allocation a new vertex for a Graph
        pTreenode = Alloc_Treenode(0, m_Color_Vertex);
        // attribute
        ((CTreenode*)pTreenode)->m_Task_no     = ((CVertex*)pElement)->m_Task_no;
        ((CTreenode*)pTreenode)->m_Execute_cost = ((CVertex*)pElement)->m_Execute_cost;
        ((CTreenode*)pTreenode)->m_Vertex_level  =  m_SizeLevel - ((CVertex*)pElement)->m_Vertex_level;
        ((CTreenode*)pTreenode)->m_father = NULL;
        ((CTreenode*)pTreenode)->m_son    = NULL;
        ((CTreenode*)pTreenode)->m_next   = NULL;
        ((CTreenode*)pTreenode)->m_CompleteTime = 0;
        ((CTreenode*)pTreenode)->m_commu_cost = 0; // at rooted tree has a zero value
        return pTreenode;
}
//////////////////////////////////////////////////////////////////////////
CElement* CTree::Alloc_Treenode(CRect aRect, COLORREF aColor)
{       return new CTreenode(aRect, aColor);   }
//////////////////////////////////////////////////////////////////////////
BOOL CTree::Free_Treenode(CTreenode* pNode)
{       if(pNode != NULL)
        {       delete pNode;           return TRUE;          }
        else
                return FALSE;
}
```

```
////////////////////////////////////////////////////////////////////////
CElement* CTree::FindFather(CElement* aVertexTree, CElement* aFather, CElement* aNode)
{
        CElement* xFather=NULL;   CElement* aSon=NULL;        BOOL afound = FALSE;
        if(aVertexTree)
        {       if( (((CTreenode*)aVertexTree)->m_Task_no) ==  (((CVertex*)aFather)->m_Task_no) )
                {       xFather = aVertexTree;          afound = FALSE;
                        aSon = ((CTreenode*)aVertexTree)->m_son;
                        while(aSon)
                        {
                                if( (((CTreenode*)aSon)->m_Task_no) == (((CVertex*)aNode)->m_Task_no) )
                                        afound = TRUE;
                                aSon = ((CTreenode*)aSon)->m_next;
                        }
                        if(!afound)
                        {       //AfxMessageBox("Can add son");
                                return xFather;
                        }
                }
                xFather = FindFather( ((CTreenode*)aVertexTree)->m_son, aFather, aNode);
                if(xFather)                     return xFather;
                xFather = FindFather( ((CTreenode*)aVertexTree)->m_next, aFather, aNode);
                if(xFather)                     return xFather;
        }
        return NULL;
}
////////////////////////////////////////////////////////////////////////
CElement* CTree::Addson_SortedCommu(CElement* pTree, CElement* pVertex, CElement* pEdge)
{       CElement* pElement;          CElement* pSource = pTree;
        CTreenode* tmp=0;           CTreenode* prev=0;
        pElement = Alloc_Treenode(0, m_Color_Vertex);
        ((CTreenode*)pElement)->m_Task_no     = ((CVertex*)pVertex)->m_Task_no;
        ((CTreenode*)pElement)->m_Execute_cost = ((CVertex*)pVertex)->m_Execute_cost;
        ((CTreenode*)pElement)->m_Vertex_level = m_SizeLevel - ((CVertex*)pVertex)->m_Vertex_level;
        ((CTreenode*)pElement)->m_father = (CTreenode*)pTree;
        ((CTreenode*)pElement)->m_son   = NULL;          ((CTreenode*)pElement)->m_next   = NULL;
        ((CTreenode*)pElement)->m_CompleteTime = 0;
        ((CTreenode*)pElement)->m_commu_cost = ((CEdge*)pEdge)->m_Commu_cost;
        tmp = prev = ((CTreenode*)pTree)->m_son;
        while(tmp != NULL)
        {       if( ((CTreenode*)pElement)->m_commu_cost >= tmp->m_commu_cost )
                {
                        ((CTreenode*)pElement)->m_next = tmp;
```

```
                        break;
                }
                else
                {       if(prev != tmp)         prev = prev->m_next;
                        tmp = tmp->m_next;
                }
        }
        if(prev == tmp)
                ((CTreenode*)pTree)->m_son = (CTreenode*)pElement;
        else
                ((CTreenode*)prev)->m_next = (CTreenode*)pElement;
        pTree = pSource;
        return pTree;
}
////////////////////////////////////////////////////////////////////////////
void CTree::Traverse_Intree(CElement* pElement)
{       CString msg;
        if(pElement)
        {
                msg.Format(_T(" Task No = T%d  Vertex Level = %d Commu = %d"),
                                ((CTreenode*)pElement)->m_Task_no, ((CTreenode*)pElement)->m_Vertex_level,
                                ((CTreenode*)pElement)->m_commu_cost);
                AfxMessageBox(msg);
                Traverse_Intree(((CTreenode*)pElement)->m_son);
                Traverse_Intree(((CTreenode*)pElement)->m_next);
        }
}
////////////////////////////////////////////////////////////////////////////
void CTree::Assign_Position(CElement* pElement, int TYPE)
{
        int aRow, aCol;     CRect  aRect = 0 ;     CString msg;
        if(pElement)
        {
                Assign_Position(((CTreenode*)pElement)->m_son, TYPE);
                m_CurrentLevel = ((CTreenode*)pElement)->m_Vertex_level;
                if(m_CurrentLevel > m_PrevLevel)
                {
                        m_PrevLevel = m_CurrentLevel;
                        aRow = (m_CurrentLevel * 2) + (m_CurrentLevel + 3);
                        aCol = (m_Leaf * 2) + 1;          m_Leaf++;          m_Current = aCol;
                        m_Start = aCol;
                        if(m_Leaf == 1)  m_StartGroup = m_Start;
                }
```

```
else if (m_CurrentLevel < m_PrevLevel)
{
        m_PrevLevel = m_CurrentLevel;
        aRow = (m_CurrentLevel * 2) + (m_CurrentLevel + 3);
        if(m_Current > m_Start)
        {       if(m_CurrentLevel == 1)
                {       aCol = m_StartGroup + (m_Current - m_StartGroup)/2;
                        m_StartGroup = m_Current + 2;
                }
                else if(m_CurrentLevel == 0)
                {       aCol = 1 + (m_Current-1)/2;     }
                else
                {       aCol = m_Start + (m_Current - m_Start)/2;   }
        }
        else
        {       if(m_CurrentLevel == 0)
                {       aCol = 1 + (m_Current - 1)/2;
                }
                else if(m_CurrentLevel == 1)
                {       aCol = m_StartGroup + (m_Current - m_StartGroup)/2;
                        m_StartGroup = m_Current + 2;
                }
                else
                {
                        aCol = m_Start + (m_Current - m_Start)/2;
                }
        }
        m_Start = m_Current + 2;
}
else
{
        m_PrevLevel = m_CurrentLevel;   aRow = (m_CurrentLevel * 2) + (m_CurrentLevel + 3);
        aCol = (m_Leaf * 2) + 1;        m_Leaf++;               m_Current = aCol;
}
CATCSView* pView =
        (CATCSView*)((CMDIFrameWnd*)AfxGetMainWnd())->GetActiveFrame()->GetActiveView();
ASSERT_KINDOF(CATCSView, pView);
pView->GetCellRect(aRow, aCol, 0);
aRect = pView->GetDocument()->m_Cell;
((CTreenode*)pElement)->m_Rect = aRect;
((CTreenode*)pElement)->SetBoundRect(aRect);
```

```
        switch(TYPE)
            {       case TREE:
                        pView->GetDocument()->AddObject_Tree(pElement);
                        break;
                    case BI_TREE:
                        pView->GetDocument()->AddObject_BiTree(pElement);
                        break;
            };
            Assign_Position(((CTreenode*)pElement)->m_next, TYPE);
        }
}
////////////////////////////////////////////////////////////////////////////
void CTree::Assign_Edge(CElement* pElement, int TYPE)
{
        CEdge *pEdge;    CElement *aSon1, *aSon2;    CPoint aStartPoint=0;   CPoint aEndPoint=0;
        UINT min1, min2, max1, max2;          CRect aRect = 0;
        CATCSView* pView =
            (CATCSView*)((CMDIFrameWnd*)AfxGetMainWnd())->GetActiveFrame()->GetActiveView();
        ASSERT_KINDOF(CATCSView, pView);
        if(pElement)
        {
            aStartPoint.x = ((CTreenode*)pElement)->m_Rect.right - (pView->m_nCellWidth/2);
            aStartPoint.y = ((CTreenode*)pElement)->m_Rect.bottom;
            aSon1 = ((CTreenode*)pElement)->m_son;  aSon2 = aSon1;
            while(aSon1)
            {       pEdge = new CEdge(0,0);
                    aEndPoint.x = (((CTreenode*)aSon1)->m_Rect.left + (pView->m_nCellWidth/2)) -1;
                    aEndPoint.y = ((CTreenode*)aSon1)->m_Rect.top;
                    pEdge->m_StartPoint = aStartPoint;
                    pEdge->m_EndPoint = aEndPoint;
                    min1 = min2 = max1 = max2 = 0;
                    min1 = min( ((CTreenode*)pElement)->m_Rect.right,
                                ((CTreenode*)pElement)->m_Rect.left );
                    min2 = min( ((CTreenode*)aSon1)->m_Rect.right,
                                ((CTreenode*)aSon1)->m_Rect.left);
                    max1 = max( ((CTreenode*)pElement)->m_Rect.right,
                                ((CTreenode*)pElement)->m_Rect.left);
                    max2 = max( ((CTreenode*)aSon1)->m_Rect.right,
                                ((CTreenode*)aSon1)->m_Rect.left);
                    aRect.left = min(min1, min2);
                    aRect.right = max(max1, max2);
                    aRect.top = ((CTreenode*)pElement)->m_Rect.bottom;
                    aRect.bottom = ((CTreenode*)aSon1)->m_Rect.top;
```

```
                    Create_InverseHeadArrow(max1, max2, pEdge, aRect);
                    pEdge->SetBoundRect(aRect);
                    switch(TYPE)
                    {       case TREE:
                                pView->GetDocument()->AddObject_Tree(pEdge);
                                break;
                            case BI_TREE:
                                pView->GetDocument()->AddObject_BiTree(pEdge);
                                break;
                    }
                    aSon1 = ((CTreenode*)aSon1)->m_next;
                }// while loop
                while(aSon2)
                {       Assign_Edge(aSon2, TYPE);
                        aSon2 = ((CTreenode*)aSon2)->m_next;
                }
        }
}
////////////////////////////////////////////////////////////////////////////
void CTree::Create_InverseHeadArrow(UINT max1, UINT max2, CEdge *pEdge, CRect aRect, CPoint aThirdPoint)
{       double x_axis, y_axis, angle;
        x_axis = y_axis = angle = 0;
        CATCSView* pView =
                (CATCSView*)((CMDIFrameWnd*)AfxGetMainWnd())->GetActiveFrame()->GetActiveView();
        ASSERT_KINDOF(CATCSView, pView);
        if( (aThirdPoint.x != 0) || (aThirdPoint.y != 0) )
        {       // don't anything       }
        y_axis = aRect.bottom - aRect.top;
        if(max1 > max2)
        { // 0<angle<45, 45<angle<90, and angle = 45 degree
                max1 -= (pView->m_nCellWidth/2);        max2 += (pView->m_nCellWidth/2);
                x_axis = max1 - max2;
                angle = atan2(y_axis, x_axis); //find arctan angle radius
                angle = (angle *180)/PI;      //find angle degree
                if( (0<angle) && (angle<20) )  // 0< x <20
                {       pEdge->m_ar1.x = pEdge->m_StartPoint.x;
                        pEdge->m_ar1.y = pEdge->m_StartPoint.y;
                        pEdge->m_ar2.x = pEdge->m_StartPoint.x - ((pView->m_nCellWidth/8)*3);
                        pEdge->m_ar2.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*2) - 1;
                        pEdge->m_ar3.x = pEdge->m_StartPoint.x - ((pView->m_nCellWidth/8)*4);
                        pEdge->m_ar3.y = pEdge->m_StartPoint.y + 1;
                }
                else if( (20<=angle) && (angle<40)) // 20<= x <40
```

```
        {       pEdge->m_ar1.x = pEdge->m_StartPoint.x;
                pEdge->m_ar1.y = pEdge->m_StartPoint.y;
                pEdge->m_ar2.x = pEdge->m_StartPoint.x - ((pView->m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*3)-1;
                pEdge->m_ar3.x = pEdge->m_StartPoint.x - ((pView->m_nCellWidth/8)*4);
                pEdge->m_ar3.y = pEdge->m_StartPoint.y + (pView->m_nCellWidth/8)-1;
        }
        else if( (40<=angle) && (angle<60) ) // 40<= x < 60
        {       pEdge->m_ar1.x = pEdge->m_StartPoint.x;
                pEdge->m_ar1.y = pEdge->m_StartPoint.y;
                pEdge->m_ar2.x = pEdge->m_StartPoint.x - ((pView->m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*4) - 1;
                pEdge->m_ar3.x = pEdge->m_StartPoint.x - ((pView->m_nCellWidth/8)*5);
                pEdge->m_ar3.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*2) - 1;
        }
        else if( (60<=angle) && (angle<80) )  // 60 <= x < 80
        {       pEdge->m_ar1.x = pEdge->m_StartPoint.x;
                pEdge->m_ar1.y = pEdge->m_StartPoint.y;
                pEdge->m_ar2.x = pEdge->m_StartPoint.x - ((pView->m_nCellWidth/8)*2);
                pEdge->m_ar2.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*4) - 1;
                pEdge->m_ar3.x = pEdge->m_StartPoint.x - ((pView->m_nCellWidth/8)*4);
                pEdge->m_ar3.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*2) - 1;
        }
        else  // 80 <= x < 90
        {
                pEdge->m_ar1.x = pEdge->m_StartPoint.x;
                pEdge->m_ar1.y = pEdge->m_StartPoint.y;
                pEdge->m_ar2.x = pEdge->m_StartPoint.x - ((pView->m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*3)-1;
                pEdge->m_ar3.x = pEdge->m_StartPoint.x;
                pEdge->m_ar3.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*4)-1;
        }
}
else if(max1 < max2)
{  // 90<angle<135, 135<angle<180, and angle = 45 degree
        max1 += (pView->m_nCellWidth/2);          max2 -= (pView->m_nCellWidth/2);
        x_axis = max2 - max1;
        angle = atan2(y_axis, x_axis); //find arctan angle degree
        angle = (angle *180)/PI;      //find angle degree
        if( (0<angle) && (angle<20) )  // 0< x <20
        {       pEdge->m_ar1.x = pEdge->m_StartPoint.x;
                pEdge->m_ar1.y = pEdge->m_StartPoint.y;
                pEdge->m_ar2.x = pEdge->m_StartPoint.x + ((pView->m_nCellWidth/8)*3);
```

```
                pEdge->m_ar2.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*2) - 1;
                pEdge->m_ar3.x = pEdge->m_StartPoint.x + ((pView->m_nCellWidth/8)*4);
                pEdge->m_ar3.y = pEdge->m_StartPoint.y + 1;
        }
        else if( (20<=angle) && (angle<40)) // 20<= x <40
        {       pEdge->m_ar1.x = pEdge->m_StartPoint.x;
                pEdge->m_ar1.y = pEdge->m_StartPoint.y + 1;
                pEdge->m_ar2.x = pEdge->m_StartPoint.x + ((pView->m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*3)-1;
                pEdge->m_ar3.x = pEdge->m_StartPoint.x + ((pView->m_nCellWidth/8)*4);
                pEdge->m_ar3.y = pEdge->m_StartPoint.y + (pView->m_nCellWidth/8)-1;
        }
        else if( (40<=angle) && (angle<60) ) // 40<= x < 60
        {       pEdge->m_ar1.x = pEdge->m_StartPoint.x;
                pEdge->m_ar1.y = pEdge->m_StartPoint.y + 1;
                pEdge->m_ar2.x = pEdge->m_StartPoint.x + ((pView->m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*4) - 1;
                pEdge->m_ar3.x = pEdge->m_StartPoint.x + ((pView->m_nCellWidth/8)*5);
                pEdge->m_ar3.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*2) - 1;
        }
        else if( (60<=angle) && (angle<80) )  // 60 <= x < 80
        {       pEdge->m_ar1.x = pEdge->m_StartPoint.x;
                pEdge->m_ar1.y = pEdge->m_StartPoint.y + 1;
                pEdge->m_ar2.x = pEdge->m_StartPoint.x + ((pView->m_nCellWidth/8)*2);
                pEdge->m_ar2.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*4) - 1;
                pEdge->m_ar3.x = pEdge->m_StartPoint.x + ((pView->m_nCellWidth/8)*4);
                pEdge->m_ar3.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*2) - 1;
        }
        else  // 80 <= x < 90
        {       pEdge->m_ar1.x = pEdge->m_StartPoint.x;
                pEdge->m_ar1.y = pEdge->m_StartPoint.y + 1;
                pEdge->m_ar2.x = pEdge->m_StartPoint.x + ((pView->m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*3)-1;
                pEdge->m_ar3.x = pEdge->m_StartPoint.x;
                pEdge->m_ar3.y = pEdge->m_StartPoint.y + ((pView->m_nCellWidth/8)*4)-1;
        }
}
else // max1 == max2
{       angle = 90;  // degree
        pEdge->m_ar1.x = pEdge->m_StartPoint.x;
        pEdge->m_ar1.y = pEdge->m_StartPoint.y + 1;
        pEdge->m_ar2.x = pEdge->m_StartPoint.x + ((pView->m_nCellWidth/8)*2);
        pEdge->m_ar2.y = pEdge->m_StartPoint.y + (pView->m_nCellWidth/2)-1;
```

```
                pEdge->m_ar3.x = pEdge->m_StartPoint.x - ((pView->m_nCellWidth/8)*2);
                pEdge->m_ar3.y = pEdge->m_StartPoint.y + (pView->m_nCellWidth/2)-1;
        }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void CTree::MakeBinaryTree(CElement *pElement, CElement *pFather, int SON_NEXT)
{       CElement *aPoint;
        if(pElement)
        {       aPoint = pElement;
                if( ((CTreenode*)aPoint)->m_next ) // indicate that it has greather child than 2
                {       if( (((CTreenode*)aPoint)->m_next)->m_next != NULL )
                        {       CElement *SP;       SP = Alloc_Treenode(0, m_Color_Vertex);
                                ((CTreenode*)SP)->m_Task_no     = SPACE_NODE;
                                ((CTreenode*)SP)->m_Execute_cost = 0;
                                ((CTreenode*)SP)->m_Vertex_level = ((CTreenode*)aPoint)->m_Vertex_level;
                                ((CTreenode*)SP)->m_father = ((CTreenode*)aPoint)->m_father;
                                ((CTreenode*)SP)->m_son    = (CTreenode*)aPoint;
                                ((CTreenode*)SP)->m_commu_cost = 0;
                                ((CTreenode*)SP)->m_CompleteTime = 0;
                                if(SON_NEXT == SON)
                                {   ((CTreenode*)SP)->m_next = (((CTreenode*)aPoint)->m_next)->m_next;
                                    ((CTreenode*)pFather)->m_son = (CTreenode*)SP;
                                    (((CTreenode*)aPoint)->m_next)->m_next = NULL;
                                }
                                else
                                {       ((CTreenode*)SP)->m_next = NULL;
                                        ((CTreenode*)pFather)->m_next = (CTreenode*)SP;
                                }
                                aPoint = SP;
                                // compute communication cost
                                int son1 , son2;
                                if( ((CTreenode*)aPoint)->m_son )
                                        son1 = (((CTreenode*)aPoint)->m_son)->m_commu_cost;
                                else
                                        son1 = 0;
                                if( (((CTreenode*)aPoint)->m_son)->m_next )
                                        son2 = ((((CTreenode*)aPoint)->m_son)->m_next)->m_commu_cost;
                                else
                                        son2 = 0;
                                ((CTreenode*)aPoint)->m_commu_cost = son1 + son2;
                        }
                        else
                        { // child equal two
```

```
                    if( SON_NEXT == NEXT)
                    {       CElement *SP;
                            SP = Alloc_Treenode(0, m_Color_Vertex);
                            ((CTreenode*)SP)->m_Task_no     = SPACE_NODE;
                            ((CTreenode*)SP)->m_Execute_cost = 0;
                            ((CTreenode*)SP)->m_Vertex_level=
                                    ((CTreenode*)aPoint)->m_Vertex_level;
                            ((CTreenode*)SP)->m_father = ((CTreenode*)aPoint)->m_father;
                            ((CTreenode*)SP)->m_son    = (CTreenode*)aPoint;
                            ((CTreenode*)SP)->m_next   = NULL;
                            ((CTreenode*)SP)->m_commu_cost = 0;
                            ((CTreenode*)SP)->m_CompleteTime = 0;
                            ((CTreenode*)pFather)->m_next = (CTreenode*)SP;
                            aPoint = SP;
                            int son1 , son2;
                            if( ((CTreenode*)aPoint)->m_son )
                                son1 = (((CTreenode*)aPoint)->m_son)->m_commu_cost;
                            else
                                son1 = 0;
                            if( (((CTreenode*)aPoint)->m_son)->m_next )
                              son2 = ((((CTreenode*)aPoint)->m_son)->m_next)->m_commu_cost;
                            else
                              son2 = 0;
                            ((CTreenode*)aPoint)->m_commu_cost = son1 + son2;
                    }
                }
            }// child >=2
            MakeBinaryTree( ((CTreenode*)aPoint)->m_son, aPoint, SON );
            MakeBinaryTree( ((CTreenode*)aPoint)->m_next, aPoint, NEXT);
        }// end if pElement
}
////////////////////////////////////////////////////////////////////////////
void CTree::AssignLevel(CElement *pElement, int aLevel)
{       if(pElement)
        {       ((CTreenode*)pElement)->m_Vertex_level = aLevel;
                AssignLevel( ((CTreenode*)pElement)->m_son, aLevel + 1 );
                AssignLevel( ((CTreenode*)pElement)->m_next, aLevel );
        }
}
////////////////////////////////////////////////////////////////////////////
void CTree::Deleted_Tree(CElement* pElement)
{       if(pElement)
        {       Deleted_Tree(((CTreenode*)pElement)->m_son);
```

```
                Deleted_Tree(((CTreenode*)pElement)->m_next);
                // delete node on a tree
                Free_Treenode((CTreenode*)pElement);
        }
}
////////////////////////////////////////////////////////////////////////////
CTask::CTask(CRect aRect,COLORREF aColor)
{       m_Rect      = aRect;        m_Color     = aColor;
        m_StartedTime  = 0;         m_CompleteTime = 0;
        m_Marked    = FALSE;
}
////////////////////////////////////////////////////////////////////////////
void CTask::Serialize(CArchive &ar)
{       CElement::Serialize(ar);  // call the base class function
        if(ar.IsStoring())
        {       ar << m_Execute_cost
                << m_Task_no
                << m_Vertex_level
                << m_Rect
                << m_Marked
                << m_StartedTime
                << m_CompleteTime;
        }
        else
        {       ar >> m_Execute_cost
                >> m_Task_no
                >> m_Vertex_level
                >> m_Rect
                >> m_Marked
                >> m_StartedTime
                >> m_CompleteTime;
        }
}
////////////////////////////////////////////////////////////////////////////
CConnect::CConnect(CPoint aStart, CPoint aEnd)
{       m_StartPoint = aStart;
        m_EndPoint   = aEnd;
        m_CommuCost = 0;
}
////////////////////////////////////////////////////////////////////////////
void CConnect::Serialize(CArchive &ar)
{       CElement::Serialize(ar);  // call the base class function
        if(ar.IsStoring())
```

```
        {       ar << m_CommuCost
                   << m_StartPoint
                   << m_EndPoint;
        }
        else
        {       ar >> m_CommuCost
                   >> m_StartPoint
                   >> m_EndPoint;
        }
}
////////////////////////////////////////////////////////////////////////////////////////
CCluster::CCluster(COLORREF aColor)
{       m_Color_vertex = aColor;      m_Color_Edge  = BLACK;      m_Task_Total  = 0;
        m_AmountOfVertex = 0;
}
////////////////////////////////////////////////////////////////////////////////////////
void CCluster::Serialize(CArchive &ar)
{       CElement::Serialize(ar);  // call the base class function
}
////////////////////////////////////////////////////////////////////////////////////////
void CCluster::InitSet(BOOL *pSet_Cluster, int size)
{       for(int i=0; i<size; i++)
        {       *(pSet_Cluster + i) = FALSE;    }
}
////////////////////////////////////////////////////////////////////////////////////////
void CCluster::TraverseTree(CElement* pElement)
{       CString msg;
        if(pElement)
        {       msg.Format(_T(" Task No = T%d  Vertex Level = %d Commu = %d"),
                        ((CTreenode*)pElement)->m_Task_no, ((CTreenode*)pElement)->m_Vertex_level,
                        ((CTreenode*)pElement)->m_commu_cost);
                AfxMessageBox(msg);
                TraverseTree(((CTreenode*)pElement)->m_son);  TraverseTree(((CTreenode*)pElement)->m_next);
        }
}
////////////////////////////////////////////////////////////////////////////////////////
void CCluster::Allocate_SetOfTask()
{       m_SetTask = new CSingleNode[m_Task_Total];
        for(int i=0; i< m_Task_Total; i++)
        {       (m_SetTask+i)->m_cost = 0;      (m_SetTask+i)->m_level = 0;     (m_SetTask+i)->m_number = i+1;
        }
}
```

```
////////////////////////////////////////////////////////////////////////////////
CElement* CCluster::Alloc_Task(CRect aRect, COLORREF aColor)
{       return new CTask(aRect, aColor);        }
////////////////////////////////////////////////////////////////////////////////
BOOL CCluster::Free_Task(CTask *pTask)
{       if(pTask != NULL)
        {       delete pTask;           return TRUE;
        }
        else
                return FALSE;
}
////////////////////////////////////////////////////////////////////////////////
CElement* CCluster::MakeCluster(CElement *pElement)
{       CElement* pTask;
        // allocation a new vertex for a Graph
        pTask = Alloc_Task(0, m_Color_vertex);
        // attribute
        ((CTask*)pTask)->m_Task_no      = ((CTreenode*)pElement)->m_Task_no;
        ((CTask*)pTask)->m_Execute_cost = ((CTreenode*)pElement)->m_Execute_cost;
        ((CTask*)pTask)->m_Vertex_level = ((CTreenode*)pElement)->m_Vertex_level;
        ((CTask*)pTask)->m_First_edge = NULL;   ((CTask*)pTask)->m_Next_vertex = NULL;
        ((CTask*)pTask)->m_StartedTime = 0;     ((CTask*)pTask)->m_Marked = FALSE;
        return pTask;
}
////////////////////////////////////////////////////////////////////////////////
void CCluster::OperationSet(BOOL *pSetLeft, BOOL *pSetRight, BOOL *pResult, int pOperation)
{       int i=0;
        switch(pOperation)
        {       case UNION:
                        for(i=0; i<m_Task_Total; i++)
                        {
                                if( (*(pSetLeft+i)== TRUE ) || (*(pSetRight+i) == TRUE) )
                                        *(pResult+i) = TRUE;
                                else
                                        *(pResult+i) = FALSE;
                        }
                        break;

                case INTERSECT:
                        for(i= 0; i< m_Task_Total; i++)
                        {
                                if( (*(pSetLeft+i)== TRUE ) && (*(pSetRight+i) == TRUE))
                                        *(pResult+i) = TRUE;
```

```
                    else
                        *(pResult+i) = FALSE;
                }
                break;

        case INVERSE_INTERSECT:
                for(i=0; i< m_Task_Total; i++)
                {
                    if( (*(pSetLeft+i)== TRUE ) && (*(pSetRight+i) == TRUE))
                        *(pResult+i) = FALSE;
                    else if( (*(pSetLeft+i)== FALSE ) && (*(pSetRight+i) == FALSE))
                        *(pResult+i) = FALSE;
                    else
                        *(pResult+i) = TRUE;
                }
                break;
        }
}
////////////////////////////////////////////////////////////////////////////////
BOOL CCluster::CheckEmptySet(BOOL *pSetLeft, BOOL *pSetRight)
{
        BOOL *SetOfTask;        BOOL result = FALSE;
        SetOfTask = new BOOL[m_Task_Total];
        InitSet(SetOfTask, m_Task_Total);
        OperationSet(pSetLeft, pSetRight, SetOfTask, INTERSECT);
        for(int i=0; i< m_Task_Total; i++)
        {   if( *(SetOfTask+i) == TRUE)
            {   result = FALSE;
                return result;
            }
            else
                result = TRUE;
        }
        delete [ ] SetOfTask;
        return result;
}
////////////////////////////////////////////////////////////////////////////////////////
int CCluster::Get_ExecuteCost(int pTaskNo)
{       for(int i=0; i< m_Task_Total; i++)
        {   if( (m_SetTask+i)->m_number == pTaskNo )
                return (m_SetTask+i)->m_cost;
        }
```

```
        return 0;
}
////////////////////////////////////////////////////////////////////////////////
int CCluster::CompletionTime(BOOL *pSetOfTask)
{       int aTotalExecute = 0;
        for(int i=0; i < m_Task_Total; i++)
        {       if( *(pSetOfTask+i) == TRUE)
                        aTotalExecute += Get_ExecuteCost(i+1);
        }
        return aTotalExecute;
}
////////////////////////////////////////////////////////////////////////////////
void CCluster::AdaptiveTaskCluster(CElement *pElement, int &pCommuCost,  int &pStartedTime, int &pCompletedTime,
                        BOOL *pSetTaskChild)
{
        int  MAX_L, MAX_R, MAX_V; int  MIN_LR;
        int  aLEFT = 0, aRIGHT = 0;
        int  aCommuCost_L=0, aCommuCost_R=0;  int  aStartedTime_L=0, aStartedTime_R=0;
        int  aStartedTime_V=0;  int  aCompletedTime_L=0, aCompletedTime_R=0;
        BOOL *aSetOfTask_L, *aSetOfTask_R;
        BOOL *aSetOfTask_V, *aSetOfTask;
        CString msg;
        aSetOfTask_L = new BOOL[m_Task_Total];  aSetOfTask_R = new BOOL[m_Task_Total];
        aSetOfTask_V = new BOOL[m_Task_Total];  aSetOfTask   = new BOOL[m_Task_Total];

        memset(aSetOfTask_L, '\0', sizeof(BOOL) * (m_Task_Total));
        memset(aSetOfTask_R, '\0', sizeof(BOOL) * (m_Task_Total));
        memset(aSetOfTask_V, '\0', sizeof(BOOL) * (m_Task_Total));
        memset(aSetOfTask, '\0', sizeof(BOOL) * (m_Task_Total));

        InitSet(aSetOfTask_L, m_Task_Total);    InitSet(aSetOfTask_R, m_Task_Total);
        InitSet(aSetOfTask_V, m_Task_Total);    InitSet(aSetOfTask, m_Task_Total);
        if(pElement)
        {       if( !((CTreenode*)pElement)->m_son )
                {       //make cluster
                        pStartedTime = 0;
                        pCommuCost = ((CTreenode*)pElement)->m_commu_cost;
                        pCompletedTime = ((CTreenode*)pElement)->m_Execute_cost + pStartedTime;
                        *(pSetTaskChild+( ((CTreenode*)pElement)->m_Task_no - 1)) = TRUE;
                }
                else
                {       // left child
                        AdaptiveTaskCluster(((CTreenode*)pElement)->m_son, aCommuCost_L,
```

```
                    aStartedTime_L, aCompletedTime_L, aSetOfTask_L);
// right child
AdaptiveTaskCluster((((CTreenode*)pElement)->m_son)->m_next,  aCommuCost_R,
                    aStartedTime_R,  aCompletedTime_R, aSetOfTask_R);
MAX_L = max(aCompletedTime_L + aCommuCost_L, aCompletedTime_R);
MAX_R = max(aCompletedTime_L, aCompletedTime_R + aCommuCost_R);
InitSet(aSetOfTask, m_Task_Total);
OperationSet(aSetOfTask_L,aSetOfTask_R, aSetOfTask, UNION);
MAX_V = CompletionTime(aSetOfTask);
MIN_LR = min(MAX_L, MAX_R);

//for space node that task number equals Space Node
if( ((CTreenode*)pElement)->m_Task_no == SPACE_NODE)
{       InitSet(aSetOfTask, m_Task_Total); //init for current node
        OperationSet(aSetOfTask_L, aSetOfTask_R, aSetOfTask, UNION);
        OperationSet(aSetOfTask, pSetTaskChild, pSetTaskChild, UNION);
        pStartedTime = CompletionTime(pSetTaskChild);
        pCommuCost = ((CTreenode*)pElement)->m_commu_cost;
}
else if(MIN_LR < MAX_V)
{       InitSet(aSetOfTask, m_Task_Total);
        aLEFT = aCompletedTime_L + aCommuCost_L;
        aRIGHT = aCompletedTime_R + aCommuCost_R;
        if(aLEFT > aRIGHT)
        { // combine with left side
          OperationSet(aSetOfTask_L, aSetOfTask, aSetOfTask,UNION);
          if(pSetTaskChild == NULL)
                OperationSet(aSetOfTask, aSetOfTask_V, aSetOfTask_V, UNION);
          else
                OperationSet(aSetOfTask, pSetTaskChild, pSetTaskChild,UNION);
          Insert_SetOfTaskCluster(aSetOfTask_R);
        }
        else // aLEFT < aRIGHT
        { // combine with right side
          OperationSet(aSetOfTask_R, aSetOfTask, aSetOfTask,UNION);
          if(pSetTaskChild == NULL)
                OperationSet(aSetOfTask, aSetOfTask_V, aSetOfTask_V, UNION);
          else
                OperationSet(aSetOfTask, pSetTaskChild, pSetTaskChild,UNION);
          Insert_SetOfTaskCluster(aSetOfTask_L);
        }
```

```
            if(pSetTaskChild == NULL)// insert final cluster
            {  *(aSetOfTask_V+( ((CTreenode*)pElement)->m_Task_no - 1)) = TRUE;
                    //Debug_Packet(aSetOfTask_V);
                    Insert_SetOfTaskCluster(aSetOfTask_V);
            }
            else
                *(pSetTaskChild+( ((CTreenode*)pElement)->m_Task_no - 1)) = TRUE;
            pStartedTime = MIN_LR;
            pCommuCost = ((CTreenode*)pElement)->m_commu_cost;
}
else if(MIN_LR >= MAX_V)
{           // check empty set
            if(CheckEmptySet(aSetOfTask_L, aSetOfTask_R))
            { InitSet(aSetOfTask, m_Task_Total);
              OperationSet(aSetOfTask_L, aSetOfTask_R, aSetOfTask,UNION);
              if( pSetTaskChild == NULL)
              {   OperationSet(aSetOfTask, aSetOfTask_V, aSetOfTask_V, UNION);
                  *(aSetOfTask_V+( ((CTreenode*)pElement)->m_Task_no - 1)) = TRUE;
                  //Debug_Packet(aSetOfTask_V);
                  Insert_SetOfTaskCluster(aSetOfTask_V);
              }
              else
              {   OperationSet(aSetOfTask, pSetTaskChild, pSetTaskChild, UNION);
                  *(pSetTaskChild+( ((((CTreenode*)pElement)->m_Task_no) - 1)) = TRUE;
              }
            //Debug_Packet(pSetTaskChild);
              pStartedTime = aCompletedTime_L + aCompletedTime_R;
              pCommuCost = ((CTreenode*)pElement)->m_commu_cost;
            }
            else if((CheckMemberSet(aSetOfTask_L, aSetOfTask_R)) ||
                    (CheckMemberSet(aSetOfTask_R, aSetOfTask_L)) )
            {       InitSet(aSetOfTask, m_Task_Total);
                    if(CheckMemberSet(aSetOfTask_L, aSetOfTask_R))
                    { if(pSetTaskChild == NULL)
                      { OperationSet(aSetOfTask_R, aSetOfTask_V, aSetOfTask_V,
                                                                  UNION);
                        *(aSetOfTask_V+( ((CTreenode*)pElement)->m_Task_no - 1))
                                                                  = TRUE;

                        //Debug_Packet(aSetOfTask_V);
                        Insert_SetOfTaskCluster(aSetOfTask_V);
                      }
                      else
```

```
        { OperationSet(aSetOfTask_R, pSetTaskChild, pSetTaskChild,
                                                    UNION);
          *(pSetTaskChild+( ((CTreenode*)pElement)->m_Task_no - 1))
                                                        = TRUE;
        }
        pStartedTime = aCompletedTime_R;
    }
    else
    {   if(pSetTaskChild == NULL)
        { OperationSet(aSetOfTask_L, aSetOfTask_V, aSetOfTask_V,
                                                        UNION);
          *(aSetOfTask_V+( ((CTreenode*)pElement)->m_Task_no - 1))
                                                    = TRUE;
          //Debug_Packet(aSetOfTask_V);
          Insert_SetOfTaskCluster(aSetOfTask_V);
        }
        else
        { OperationSet(aSetOfTask_L, pSetTaskChild, pSetTaskChild,
                                    UNION);
          *(pSetTaskChild+( ((CTreenode*)pElement)->m_Task_no - 1))
                                                        = TRUE;
        }
        pStartedTime = aCompletedTime_L;
    }
    pCommuCost = ((CTreenode*)pElement)->m_commu_cost;
}
else
{       InitSet(aSetOfTask, m_Task_Total);
        aLEFT = aCompletedTime_L + aCommuCost_L;
        aRIGHT = aCompletedTime_R + aCommuCost_R;
        if(aLEFT > aRIGHT)
        { // combine with left side
          OperationSet(aSetOfTask_L, aSetOfTask, aSetOfTask,UNION);
          if(pSetTaskChild == NULL)
                OperationSet(aSetOfTask, aSetOfTask_V, aSetOfTask_V,
                                        UNION);
          else
                OperationSet(aSetOfTask, pSetTaskChild, pSetTaskChild,
                                        UNION);
          //Debug_Packet(aSetOfTask_R);
          Insert_SetOfTaskCluster(aSetOfTask_R);
        }
        else // aLEFT < aRIGHT
```

```
                                        {  // combine with right side
                                           OperationSet(aSetOfTask_R, aSetOfTask, aSetOfTask,UNION);
                                           if(pSetTaskChild == NULL)
                                              OperationSet(aSetOfTask, aSetOfTask_V, aSetOfTask_V, UNION);
                                           else
                                             OperationSet(aSetOfTask, pSetTaskChild, pSetTaskChild,UNION);
                                           //Debug_Packet(aSetOfTask_L);
                                           Insert_SetOfTaskCluster(aSetOfTask_L);
                                        }
                                        if(pSetTaskChild == NULL)// insert final cluster
                                        {  *(aSetOfTask_V+( ((CTreenode*)pElement)->m_Task_no - 1))
                                                                                = TRUE;
                                           //Debug_Packet(aSetOfTask_V);
                                           Insert_SetOfTaskCluster(aSetOfTask_V);
                                        }
                                        else
                                           *(pSetTaskChild+( ((CTreenode*)pElement)->m_Task_no - 1))
                                                                                = TRUE;
                                        pStartedTime = MIN_LR;
                                        pCommuCost = ((CTreenode*)pElement)->m_commu_cost;
                                                }
                                             }
                                          pCompletedTime = pStartedTime + ((CTreenode*)pElement)->m_Execute_cost;
                                          }
                                    }
                                    else
                                    {       pCommuCost = 0;
                                            pStartedTime = 0;
                                            pCompletedTime = 0;
                                            InitSet(pSetTaskChild, m_Task_Total);
                                    }
                                    delete [ ] aSetOfTask_L;
                                    delete [ ] aSetOfTask_R;
                                    delete [ ] aSetOfTask_V;
                                    delete [ ] aSetOfTask;
}
//////////////////////////////////////////////////////////////////////////////
void CCluster::Put_TotalTask(int pTotalTask)
{               m_Task_Total = pTotalTask;              }
//////////////////////////////////////////////////////////////////////////////
void CCluster::Insert_SetOfTaskCluster(BOOL *pSetOfTask)
{               m_CSetTask = new CSetTask(m_Task_Total);
```

```
        if(!m_CSetTask)
        {       AfxMessageBox(_T("Connot allocate memory for a set of task cluser"));
                return;
        }
        for(int i=0; i< m_Task_Total; i++)
        {       if( *(pSetOfTask+i) == TRUE)
                        *(m_CSetTask->m_SetCluster + i) = 1;
                else
                        *(m_CSetTask->m_SetCluster + i) = 0;
        }
        m_ArrayTaskCluster.AddTail(m_CSetTask);
}
//////////////////////////////////////////////////////////////////////////////
CSetTask* CCluster::Get_SetOfTaskCluster()
{       POSITION aPos;
        aPos = m_ArrayTaskCluster.GetTailPosition();
        if(aPos)
        {       CSetTask *CObj = m_ArrayTaskCluster.GetPrev(aPos);
                m_ArrayTaskCluster.RemoveTail();
                return CObj;
        }
        else
                return NULL;
}
//////////////////////////////////////////////////////////////////////////////
void CCluster::Del_SetOfTaskCluster(CSetTask *pSetOfTask)
{       if(pSetOfTask)
        {       POSITION aPosition = m_ArrayTaskCluster.Find(pSetOfTask);
                m_ArrayTaskCluster.RemoveAt(aPosition);
        }
}
//////////////////////////////////////////////////////////////////////////////
BOOL CCluster::Empty_SetOfTaskCluster()
{       POSITION aPos;
        aPos = m_ArrayTaskCluster.GetTailPosition();
        if(aPos)
                return FALSE;
        else
                return TRUE;
}
//////////////////////////////////////////////////////////////////////////////
// Every element is a set of pSetDestinate
BOOL CCluster::CheckMemberSet(BOOL *pSetSource, BOOL *pSetDestinate)
```

```
{       BOOL result = FALSE;
        for(int i =0; i<m_Task_Total; i++)
        {       if( *(pSetSource + i) == TRUE)
                {       if( *(pSetDestinate + i) == TRUE )
                                result = TRUE;
                        else
                        {       result = FALSE;
                                return result;
                        }
                }
        }
        return result;
}
////////////////////////////////////////////////////////////////////////////
int CCluster::Get_TotalTask()
{       return m_Task_Total;        }
////////////////////////////////////////////////////////////////////////////
void CCluster::ScheduleOfTask()
{       }
////////////////////////////////////////////////////////////////////////////
void CCluster::Debug_Packet(BYTE *pPacket)
{       BYTE *packet;       int i=0;
        packet = new BYTE[m_Task_Total];
        memset(packet, '\0', m_Task_Total);
        for(i=0; i<m_Task_Total; i++)
        {       if( *(pPacket+i) == 1)
                        *(packet+i) = 1;
                else
                        *(packet+i) = 0;
        }
        CString msg;
        msg.Format(_T("[%d][%d][%d][%d][%d][%d][%d][%d][%d][%d]"),
                *(packet), *(packet+1), *(packet+2), *(packet+3), *(packet+4),
                *(packet+5), *(packet+6), *(packet+7), *(packet+8), *(packet+9));
        AfxMessageBox(msg);         delete [ ] packet;
}
////////////////////////////////////////////////////////////////////////////
CNodeBi* CCluster::Alloc_NodeBi()
{       return new CNodeBi();       }
////////////////////////////////////////////////////////////////////////////
CNodeBi* CCluster::MakeBiTree(int KeyValue)
{       CNodeBi *aElement;
```

```
        aElement = Alloc_NodeBi();   aElement->info = KeyValue;
        aElement->left = NULL;       aElement->right = NULL;
        return aElement;
}
////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CCluster::SetLeftBiTree(CNodeBi *pBiTree, int KeyValue)
{       if(!pBiTree)
                return FALSE;
        if(pBiTree->left != NULL)
                return FALSE;
        else
        {       pBiTree->left = MakeBiTree(KeyValue);
                return TRUE;
        }
}
////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CCluster::SetRightBiTree(CNodeBi *pBiTree, int KeyValue)
{       if(!pBiTree)
                return FALSE;
        if(pBiTree->right != NULL)
                return FALSE;
        else
        {       pBiTree->right = MakeBiTree(KeyValue);          return TRUE;    }
}
////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CCluster::InsertNodeOnBiTree(CNodeBi *pRooted, int KeyValue)
{       CNodeBi *p,*q;
        p = q = 0;
        if(!pRooted)
        {       pRooted = MakeBiTree(KeyValue);         }
        else
        {       p = q = pRooted;
                while((p->info != KeyValue) && (q != NULL))
                {       p = q;
                        if(KeyValue < p->info)
                                q = p->left;
                        else
                                q = p->right;
                } // end whild
                if(KeyValue == p->info)
                        return FALSE;
                else if(KeyValue < p->info)
                        SetLeftBiTree(p, KeyValue);
```

```
                else
                        SetRightBiTree(p, KeyValue);
        }
        return TRUE;
}
////////////////////////////////////////////////////////////////////////////////
void CCluster::DeleteBiTree(CNodeBi *pPoint)
{       if(pPoint)
        {       DeleteBiTree(pPoint->left);     DeleteBiTree(pPoint->right);
                delete pPoint;
        }
}
////////////////////////////////////////////////////////////////////////////////
void CCluster::TraverseBiTree(CNodeBi *pPoint)
{       CString msg;
        if(pPoint)
        {       TraverseBiTree(pPoint->left);
                msg.Format(_T("BiTree Task No [%d]"), pPoint->info);
                AfxMessageBox(msg);
                TraverseBiTree(pPoint->right);
        }
}
////////////////////////////////////////////////////////////////////////////////
void CCluster::GetSinkNode_OnBiTree(CNodeBi *pPoint,CSingleNode *pArrayNode, int aSize)
{       if(pPoint)
        {       GetSinkNode_OnBiTree(pPoint->left, pArrayNode, aSize);
                int i=0;
                while((pArrayNode+i)->m_number != 0)
                        i++;
                (pArrayNode+i)->m_number = pPoint->info;
                (pArrayNode+i)->m_cost  = 0;
                GetSinkNode_OnBiTree(pPoint->right, pArrayNode, aSize);
        }
}
////////////////////////////////////////////////////////////////////////////////
CLinkList *CCluster::MakeLinkList(int pInfo)
{       CLinkList *aNodeList;       aNodeList = new CLinkList();
        aNodeList->info = pInfo;    aNodeList->pNextNode = NULL;
        return aNodeList;
}
////////////////////////////////////////////////////////////////////////////////
CLinkList *CCluster::GetTailLinkList(CLinkList *pLinkList, int &pInfo)
{       CLinkList *pCurr, *pPrev, *pPPrev;
```

```
        pCurr = pPrev = pPPrev = NULL;
        if(!pLinkList)
                return NULL;
        else
        {       pCurr = pPrev = pPPrev = pLinkList;
                while(pCurr)
                {       if(pPPrev != pPrev)
                                pPPrev = pPrev;
                        pPrev = pCurr;
                        pCurr = pCurr->pNextNode;
                }
                pInfo = pPrev->info;
                if(pPPrev != pPrev)
                        pPPrev->pNextNode = NULL;
                delete pPrev;           pPrev = NULL;
                if(pLinkList)
                        return pLinkList;
                else
                        return NULL;
        }
}
////////////////////////////////////////////////////////////////////////////////////////////
CLinkList *CCluster::InsertToLinkList(CLinkList *pLinkList, int pInfo)
{       CLinkList *pCurrent, *pPrev;
        pCurrent = pPrev = NULL;
        if(!pLinkList)
        {       pLinkList = MakeLinkList(pInfo);        }
        else
        {       pCurrent = pPrev =pLinkList;
                while(pCurrent)
                {       pPrev = pCurrent;
                        pCurrent = pCurrent->pNextNode;
                }
                pPrev->pNextNode = MakeLinkList(pInfo);
        }
        return pLinkList;
}
////////////////////////////////////////////////////////////////////////////////////////////
void CCluster::FreeLinkList(CLinkList *pLinkList)
{       CLinkList *pCurr, *pNext;
        pCurr = pNext = 0;
```

```
        pCurr = pNext = pLinkList;
        while(pCurr)
        {       pNext = pCurr->pNextNode;       delete pCurr;
                pCurr = NULL;           pCurr = pNext;
        }
}
////////////////////////////////////////////////////////////////////////////////
void CCluster::PushMakeSpan(int *pLinkMakeSpan, int info)
{       int *pTmpLink;
        pTmpLink = pLinkMakeSpan;
        while(pTmpLink)
        {       if( *pTmpLink == 0)
                {       *pTmpLink = info;
                        break;
                }
                pTmpLink++;
        }
}
////////////////////////////////////////////////////////////////////////////////
int CCluster::PopMakeSpan(int *pLinkMakeSpan)
{       int makespan=0;     int index=0;
        while(pLinkMakeSpan + index)
        {       if(*(pLinkMakeSpan+index) == 0)
                        break;
                index++;
        }
        index--;
        makespan = *(pLinkMakeSpan+index);
        *(pLinkMakeSpan+index) = 0;
        return makespan;
}

////////////////////////////////////////////////////////////////////////////////
//              ATCS.h File
////////////////////////////////////////////////////////////////////////////////
// ATCS.h : main header file for the ATCS application
//
#if !defined(AFX_ATCS_H__DA60A495_30EB_11D3_93C5_00A0C92CFD9E__INCLUDED_)
#define AFX_ATCS_H__DA60A495_30EB_11D3_93C5_00A0C92CFD9E__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
```

```
#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif
#include "resource.h"		// main symbols
/////////////////////////////////////////////////////////////////////////////
// CATCSApp:
// See ATCS.cpp for the implementation of this class
class CATCSApp : public CWinApp
{
public:
	CATCSApp();
// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CATCSApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL
// Implementation
	//{{AFX_MSG(CATCSApp)
	afx_msg void OnAppAbout();
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif // !defined(AFX_ATCS_H__DA60A495_30EB_11D3_93C5_00A0C92CFD9E__INCLUDED_)

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//		ATCS.cpp File
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// ATCS.cpp : Defines the class behaviors for the application.
//
#include "stdafx.h"
#include "ATCS.h"
#include "MainFrm.h"
#include "ChildFrm.h"
#include "Graph.h"
#include "ATCSDoc.h"
#include "ATCSView.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CATCSApp
BEGIN_MESSAGE_MAP(CATCSApp, CWinApp)
        //{{AFX_MSG_MAP(CATCSApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
        // Standard print setup command
        ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////////
// CATCSApp construction
CATCSApp::CATCSApp()
{       // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}
/////////////////////////////////////////////////////////////////////////////
// The one and only CATCSApp object
CATCSApp theApp;
/////////////////////////////////////////////////////////////////////////////
// CATCSApp initialization
BOOL CATCSApp::InitInstance()
{       AfxEnableControlContainer();
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.
#ifdef _AFXDLL
        Enable3dControls();                     // Call this when using MFC in a shared DLL
#else
        Enable3dControlsStatic();       // Call this when linking to MFC statically
#endif
        // Change the registry key under which our settings are stored.
        // TODO: You should modify this string to be something appropriate
```

```
        // such as the name of your company or organization.
        SetRegistryKey(_T("Local AppWizard-Generated Applications"));
        LoadStdProfileSettings();  // Load standard INI file options (including MRU)
        // Register the application's document templates.  Document templates
        //  serve as the connection between documents, frame windows and views.
        CMultiDocTemplate* pDocTemplate;
        pDocTemplate = new CMultiDocTemplate(
                IDR_ATCSTYPE,
                RUNTIME_CLASS(CATCSDoc),
                RUNTIME_CLASS(CChildFrame), // custom MDI child frame
                RUNTIME_CLASS(CATCSView));
        AddDocTemplate(pDocTemplate);

        // create main MDI Frame window
        CMainFrame* pMainFrame = new CMainFrame;
        if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
                return FALSE;
        m_pMainWnd = pMainFrame;

        // Parse command line for standard shell commands, DDE, file open
        CCommandLineInfo cmdInfo;
        ParseCommandLine(cmdInfo);

        // Dispatch commands specified on the command line
        if (!ProcessShellCommand(cmdInfo))
                return FALSE;

        // The main window has been initialized, so show and update it.
        //pMainFrame->ShowWindow(m_nCmdShow);
        pMainFrame->ShowWindow(SW_MAXIMIZE);
        pMainFrame->UpdateWindow();
        return TRUE;
}
//////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{
public:
        CAboutDlg();
// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA
```

```
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL
// Implementation
protected:
        //{{AFX_MSG(CAboutDig)
                // No message handlers
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{       //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}
/////////////////////////////////////////////////////////////////////////////
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{       CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}
/////////////////////////////////////////////////////////////////////////////
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////////
// App command to run the dialog
void CATCSApp::OnAppAbout()
{       CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}
/////////////////////////////////////////////////////////////////////////////
// CATCSApp message handlers


//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//              MainFrm.h File
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// MainFrm.h : interface of the CMainFrame class
//
```

```
#if !defined(AFX_MAINFRM_H__DA60A499_30EB_11D3_93C5_00A0C92CFD9E__INCLUDED_)
#define AFX_MAINFRM_H__DA60A499_30EB_11D3_93C5_00A0C92CFD9E__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMainFrame : public CMDIFrameWnd
{       DECLARE_DYNAMIC(CMainFrame)
public:
        CMainFrame();
// Attributes
public:
// Operations
public:
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL
// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;
// Generated message map functions
protected:
        BOOL CreateStatusBar();
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif // !defined(AFX_MAINFRM_H__DA60A499_30EB_11D3_93C5_00A0C92CFD9E__INCLUDED_)
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//              MainFrm.cpp File
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// MainFrm.cpp : implementation of the CMainFrame class
//
#include "stdafx.h"
#include "ATCS.h"
#include "MainFrm.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CMainFrame
IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        ON_WM_CREATE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction
CMainFrame::CMainFrame()
{       // TODO: add member initialization code here        }
CMainFrame::~CMainFrame()
{       }
/////////////////////////////////////////////////////////////////////////////
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{       if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;
        //m_wndToolBar.Create(this);
        if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP
                | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
                !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
        {       TRACE0("Failed to create toolbar\n");
                return -1;      // fail to create
        }

        if(!CreateStatusBar())
                return -1;
```

```
        // TODO: Delete these three lines if you don't want the toolbar to
        //  be dockable
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);
        return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{       if( !CMDIFrameWnd::PreCreateWindow(cs) )
                return FALSE;
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs
        cs.style &= ~FWS_ADDTOTITLE;
        return TRUE;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CMainFrame::CreateStatusBar()
{       static UINT nIndicators[] =
        {       ID_SEPARATOR,           // status line indicator
                ID_INDICATOR_SINGLE_P,
                ID_INDICATOR_MAKESPAN,
                ID_INDICATOR_SCHEDULE,
                ID_INDICATOR_SPEEDUP
        };
        if(!m_wndStatusBar.Create(this))
                return FALSE;
        m_wndStatusBar.SetIndicators(nIndicators, 6);
        return TRUE;
}
////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics
#ifdef _DEBUG
void CMainFrame::AssertValid() const
{       CMDIFrameWnd::AssertValid();            }
////////////////////////////////////////////////////////////////////
void CMainFrame::Dump(CDumpContext& dc) const
{       CMDIFrameWnd::Dump(dc);  }
#endif // _DEBUG
////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//      ATCSDoc.h File
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// ATCSDoc.h : interface of the CATCSDoc class
//
#if !defined(AFX_ATCSDOC_H__DA60A49D_30EB_11D3_93C5_00A0C92CFD9E__INCLUDED_)
#define AFX_ATCSDOC_H__DA60A49D_30EB_11D3_93C5_00A0C92CFD9E__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
////////////////////////////////////////////////////////////////////////
class CATCSDoc : public CDocument
{
protected: // create from serialization only
        CATCSDoc();
        DECLARE_DYNCREATE(CATCSDoc)
// Attributes
pubiic:
        int     m_TOTAL_VERTEX;  // Amount of task on graph
        int     m_TOTAL_ARC;
        COLORREF m_Color;
        BOOL    m_StatGraph;    // Check To create whether graph complete
        int     m_SizeLevel;
        CRect   m_Cell;
        int     m_StageShown;
        ////////////////////////////
        // The following variable does not fill into serialize
        BOOL    m_StateTREE;
        BOOL    m_StateBITREE;
        BOOL    m_StateCLUSTER;
        BOOL    m_StateSCHEDULE;
        ////////////////////////////////
protected:
        CSize   m_DocSize;                      // set Document size
  int   m_PenWidth;
        CTypedPtrList<CObList, CElement*> m_NodeList;
        CTypedPtrList<CObList, CElement*> m_ArcList;
        CTypedPtrList<CObList, CElement*> m_ObjectList;
        CTypedPtrList<CObList, CElement*> m_ObjectCluster;
        CTypedPtrList<CObList, CElement*> m_aSetOfTaskCluster;
        ////////////////////////////////////////////////////////////////
        // used for stack operation on edge and father
        CTypedPtrList<CObList, CElement*> m_StackFather;
        CTypedPtrList<CObList, CElement*> m_StackEdge;
        CTypedPtrList<CObList, CElement*> m_ArrayTree;
```

```
        CTypedPtrList<CObList, CElement*> m_BinaryTree;
        CTypedPtrList<CObList, CElement*> m_Cluster;
        CTypedPtrList<CObList, CElement*> m_ScheduleCluster;

        // used for a list pointer of several trees
        CTypedPtrList<CObList, CElement*> m_ObjectTree;
        CTypedPtrList<CObList, CElement*> m_ObjectBiTree;
// Operations
public:
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CATCSDoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        //}}AFX_VIRTUAL
        CSize GetDocSize()
        {return m_DocSize;}
        COLORREF GetColor()
        {return m_Color;}

        // for node
        void AddNode(CElement* pNode)
        {   m_NodeList.AddTail(pNode);
            SetModifiedFlag();
        }

        POSITION GetListHeadPosition( )
        {return m_NodeList.GetHeadPosition( ); }

        CElement* GetNext(POSITION& aPos)
        {return m_NodeList.GetNext(aPos); }

        POSITION GetListTailPosition()
        {return m_NodeList.GetTailPosition();}

        CElement *GetPrev(POSITION &aPos)
        {return m_NodeList.GetPrev(aPos); }

        void Clean_Node();

        BOOL EmptyNode()
        {       POSITION aPos;
```

```
        aPos = m_NodeList.GetTailPosition();
        if(aPos)
                return FALSE;
        else
                return TRUE;
}

void DeleteElement(CElement *pElement);  //Delete an element
////////////////////////////////////////
// for arc
void AddArc(CElement* pArc)
{
        m_ArcList.AddTail(pArc);
        SetModifiedFlag();
}
POSITION GetArcListHeadPosition()
{ return m_ArcList.GetHeadPosition(); }

CElement* GetArcNext(POSITION& aPos)
{ return m_ArcList.GetNext(aPos); }

POSITION GetArcListTailPosition()
{ return m_ArcList.GetTailPosition(); }

CElement* GetArcPrev(POSITION &aPos)
{ return m_ArcList.GetPrev(aPos); }

void Clean_Arc();
BOOL EmptyArc()
{       POSITION aPos;
        aPos = m_ArcList.GetTailPosition();
        if(aPos)
                return FALSE;
        else
                return TRUE;
}
void DeleteArc(CElement *pElement); //delete an arc
////////////////////////////////////////////////
// for all object
void AddObject(CElement* pNode)
{       m_ObjectList.AddTail(pNode);
  SetModifiedFlag();
}
```

```
POSITION GetListHeadObjectPosition()
{return m_ObjectList.GetHeadPosition(); }

CElement* GetNextObject(POSITION& aPos)
{return m_ObjectList.GetNext(aPos); }

POSITION GetListTailObjectPosition()
{return m_ObjectList.GetTailPosition();}

CElement *GetPrevObject(POSITION &aPos)
{return m_ObjectList.GetPrev(aPos); }

void DeleteObject(CElement *pElement);  //Delete an element

void InitObject();

BOOL EmptyObject()
{       POSITION aPos;
        aPos = m_ObjectList.GetHeadPosition();
        if(aPos)
                return FALSE;
        else
                return TRUE;
}
//////////////////////////////////////////////////////
// for object cluster
void AddObjCluster(CElement* pNode)
{       m_ObjectCluster.AddTail(pNode);
  SetModifiedFlag();
}

POSITION GetListHeadObjClusterPosition()
{return m_ObjectCluster.GetHeadPosition(); }

CElement* GetNextObjCluster(POSITION& aPos)
{return m_ObjectCluster.GetNext(aPos); }

POSITION GetListTailObjClusterPosition()
{return m_ObjectCluster.GetTailPosition();}

CElement *GetPrevObjCluster(POSITION &aPos)
{return m_ObjectCluster.GetPrev(aPos); }
```

```
void InitObjCluster();

void DeleteObjCluster(CElement *pElement);  //Delete an element

BOOL EmptyObjCluster()
{       POSITION aPos;
        aPos = m_ObjectCluster.GetTailPosition();
        if(aPos)
                return FALSE;
        else
                return TRUE;
}
//////////////////////////////////////////
void Add_aSetOfTaskCluster(CElement* pNode)
{       m_aSetOfTaskCluster.AddTail(pNode);
         SetModifiedFlag();
}

POSITION Get_aSetOfTaskCluster()
{return m_aSetOfTaskCluster.GetHeadPosition(); }

CElement* GetNext_aSetOfTaskCluster(POSITION& aPos)
{return m_aSetOfTaskCluster.GetNext(aPos); }

POSITION GetTail_aSetOfTaskCluster()
{return m_aSetOfTaskCluster.GetTailPosition();}

CElement *GetPrev_aSetOfTaskCluster(POSITION &aPos)
{return m_aSetOfTaskCluster.GetPrev(aPos); }

void Delete_aSetOfTaskCluster(CElement *pElement);

void Init_aSetOfTaskCluster();

BOOL Empty_aSetOfTaskCluster()
{       POSITION aPos;
        aPos = m_aSetOfTaskCluster.GetTailPosition();
        if(aPos)
                return FALSE;
        else
                return TRUE;
}
```

```
////////////////////////////////////////////////////
// for object tree list
void AddObject_Tree(CElement* pNode)
{       m_ObjectTree.AddTail(pNode);
        SetModifiedFlag();
}

POSITION GetHeadObject_TreePosition()
{ return m_ObjectTree.GetHeadPosition(); }

CElement* GetNextObject_Tree(POSITION& aPos)
{  return  m_ObjectTree.GetNext(aPos); }

POSITION GetTailObject_TreePosition()
{ return  m_ObjectTree.GetTailPosition(); }

CElement *GetPrevObject_Tree(POSITION &aPos)
{ return  m_ObjectTree.GetPrev(aPos);  }

void DeleteObject_Tree(CElement *pElement);  //Delete an element
void InitObject_Tree();
BOOL EmptyObject_Tree()
{       POSITION aPos;
        aPos = m_ObjectTree.GetHeadPosition();
        if(aPos)
                return FALSE;
        else
                return TRUE;
}
////////////////////////////////////////////////////
// for object binary tree
void AddObject_BiTree(CElement* pNode)
{       m_ObjectBiTree.AddTail(pNode);
        SetModifiedFlag();
}

POSITION GetHeadObject_BiTreePosition()
{ return m_ObjectBiTree.GetHeadPosition();  }

CElement* GetNextObject_BiTree(POSITION& aPos)
{return m_ObjectBiTree.GetNext(aPos); }

POSITION GetTailObject_BiTreePosition()
```

```
{return m_ObjectBiTree.GetTailPosition();}

CElement *GetPrevObject_BiTrec(POSITION &aPos)
{return m_ObjectBiTree.GetPrev(aPos); }

void DeleteObject_BiTree(CElement *pElement);  //Delete an element

BOOL EmptyObject_BiTree()
{       POSITION aPos;
        aPos = m_ObjectBiTree.GetTailPosition();
        if(aPos)
                return FALSE;
        else
                return TRUE;
}

void InitObject_BiTree();
BOOL Empty_StackEdge();
void Push_StackEdge(CElement* pElement);
CElement* Pop_StackEdge();
void Del_EdgeOnStack(CElement* pElement);

BOOL Empty_StackFather();
void Push_StackFather(CElement* pElement);
CElement* Pop_StackFather();
void Del_FatherOnStack(CElement* pElement);

BOOL Empty_StackArrayTree();
void Push_RootedTree(CElement* pElement);
CElement* Pop_RootedTree();
void Del_RootedTreeOnStack(CElement* pElement);

BOOL Empty_StackBinaryTree();
void Push_RootedBinaryTree(CElement* pElement);
CElement* Pop_RootedBinaryTree();
void Del_RootedBinaryTreeOnStack(CElement *pElement);

BOOL Empty_StackCluster();
void Push_RootedCluster(CElement* pElement);
CElement* Pop_RootedCluster();
void Del_RootedClusterOnStack(CElement* pElement);

POSITION GetHeadCluster()
```

```
        {return m_Cluster.GetHeadPosition(); }

        CElement* GetNextCluster(POSITION& aPos)
        {return m_Cluster.GetNext(aPos); }

        POSITION GetTailCluster()
        {return m_Cluster.GetTailPosition();}

        CElement *GetPrevCluster(POSITION &aPos)
        {return m_Cluster.GetPrev(aPos); }

        // Used for finding makespan
        BOOL Empty_StackSchedule();
        void Push_RootedSchedule(CElement *pElement);
        CElement *Pop_RootedSchedule();
        void Del_RootedSchedule(CElement *pElement);

        POSITION GetHeadSchedule()
        { return m_ScheduleCluster.GetHeadPosition(); }

        CElement *GetNextSchedule(POSITION &aPos)
        { return m_ScheduleCluster.GetNext(aPos); }

        POSITION GetTailSchedule()
        { return m_ScheduleCluster.GetTailPosition(); }

        CElement *GetPrevSchedule(POSITION &aPos)
        { return m_ScheduleCluster.GetPrev(aPos); }

// Implementation
public:
        virtual ~CATCSDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CATCSDoc)
        afx_msg void OnColorBlack();
```

```
        afx_msg void OnUpdateColorBlack(CCmdUI* pCmdUI);
        afx_msg void OnColorBlue();
        afx_msg void OnUpdateColorBlue(CCmdUI* pCmdUI);
        afx_msg void OnColorGreen();
        afx_msg void OnUpdateColorGreen(CCmdUI* pCmdUI);
        afx_msg void OnColorRed();
        afx_msg void OnUpdateColorRed(CCmdUI* pCmdUI);
        afx_msg void OnColorYellow();
        afx_msg void OnUpdateColorYellow(CCmdUI* pCmdUI);
        afx_msg void OnColorWhite();
        afx_msg void OnUpdateColorWhite(CCmdUI* pCmdUI);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_ATCSDOC_H__DA60A49D_30EB_11D3_93C5_00A0C92CFD9E__INCLUDED_)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//              ATCSDoc.cpp File
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// ATCSDoc.cpp : implementation of the CATCSDoc class
//
#include "stdafx.h"
#include "ATCS.h"
#include "Graph.h"
#include "ATCSDoc.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CATCSDoc
IMPLEMENT_DYNCREATE(CATCSDoc, CDocument)
BEGIN_MESSAGE_MAP(CATCSDoc, CDocument)
        //{{AFX_MSG_MAP(CATCSDoc)
        ON_COMMAND(ID_COLOR_BLACK, OnColorBlack)
        ON_UPDATE_COMMAND_UI(ID_COLOR_BLACK, OnUpdateColorBlack)
        ON_COMMAND(ID_COLOR_BLUE, OnColorBlue)
        ON_UPDATE_COMMAND_UI(ID_COLOR_BLUE, OnUpdateColorBlue)
```

```
        ON_COMMAND(ID_COLOR_GREEN, OnColorGreen)
        ON_UPDATE_COMMAND_UI(ID_COLOR_GREEN, OnUpdateColorGreen)
        ON_COMMAND(ID_COLOR_RED, OnColorRed)
        ON_UPDATE_COMMAND_UI(ID_COLOR_RED, OnUpdateColorRed)
        ON_COMMAND(ID_COLOR_YELLOW, OnColorYellow)
        ON_UPDATE_COMMAND_UI(ID_COLOR_YELLOW, OnUpdateColorYellow)
        ON_COMMAND(ID_COLOR_WHITE, OnColorWhite)
        ON_UPDATE_COMMAND_UI(ID_COLOR_WHITE, OnUpdateColorWhite)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////
// CATCSDoc construction/destruction
CATCSDoc::CATCSDoc()
{       // TODO: add one-time construction code here
        m_TOTAL_VERTEX  = 0;
        m_TOTAL_ARC     = 0;
        m_StatGraph     = FALSE;
        m_DocSize       = CSize(3000,3000);
        m_Color         = RED;
        m_SizeLevel     = 0;
        m_Cell = 0;
        m_StageShown = 1;
        m_StateTREE     = FALSE;
        m_StateBITREE   = FALSE;
        m_StateCLUSTER  = FALSE;
        m_StateSCHEDULE = FALSE;
}
CATCSDoc::~CATCSDoc()
{       // for m_NodeList
        //Get the position at the head of the list
        POSITION aPosition;
        aPosition = 0;
        if(!EmptyNode())
        {       aPosition = m_NodeList.GetTailPosition();
                while(aPosition)
                        delete m_NodeList.GetPrev(aPosition);
                m_NodeList.RemoveAll();
        }
        // for m_ArcList
        aPosition = 0;
        if(!EmptyArc())
        {       aPosition = m_ArcList.GetTailPosition();
                while(aPosition)
```

```
                delete m_ArcList.GetPrev(aPosition);
        m_ArcList.RemoveAll();
}
// for m_ObjectList
aPosition = 0;
if(!EmptyObject())
{       aPosition = m_ObjectList.GetHeadPosition();
        while(aPosition)
                delete m_ObjectList.GetNext(aPosition);
        m_ObjectList.RemoveAll();
}
// for m_StackEdge
aPosition = 0;
if(!Empty_StackEdge())
{       aPosition = m_StackEdge.GetTailPosition();
        while(aPosition)
                delete m_StackEdge.GetPrev(aPosition);
        m_StackEdge.RemoveAll();
}
// for m_StackFather
aPosition = 0;
if(!Empty_StackFather())
{       aPosition = m_StackFather.GetTailPosition();
        while(aPosition)
                delete m_StackFather.GetPrev(aPosition);
        m_StackFather.RemoveAll();
}
// for m_ArrayTree
aPosition = 0;
if(!Empty_StackArrayTree())
{       aPosition = m_ArrayTree.GetTailPosition();
        while(aPosition);
                delete m_ArrayTree.GetPrev(aPosition);
        m_ArrayTree.RemoveAll();
}
// for m_BinaryTree
aPosition = 0;
if(!Empty_StackBinaryTree())
{       aPosition = m_BinaryTree.GetTailPosition();
        while(aPosition)
                delete m_BinaryTree.GetPrev(aPosition);

        m_BinaryTree.RemoveAll();
```

```
        }
        // for m_Cluster
        aPosition = 0;
        if(!Empty_StackCluster())
        {       aPosition = m_Cluster.GetTailPosition();
                while(aPosition)
                        delete m_Cluster.GetPrev(aPosition);
                m_Cluster.RemoveAll();
        }
        // for m_ScheduleCluster
        aPosition = 0;
        if(!Empty_StackSchedule())
        {       aPosition = m_ScheduleCluster.GetTailPosition();
                while(aPosition)
                        delete m_ScheduleCluster.GetPrev(aPosition);
                m_ScheduleCluster.RemoveAll();
        }
        // for m_ObjectTree
        if(m_StateTREE)
        {       aPosition = 0;
                if(!EmptyObject_Tree())
                {       aPosition = m_ObjectTree.GetHeadPosition();
                        while(aPosition)
                                delete m_ObjectTree.GetNext(aPosition);
                        m_ObjectTree.RemoveAll();
                }
        }
        ////////////////////////////////////////////
        Init_aSetOfTaskCluster();       InitObjCluster();
        InitObject_Tree();              InitObject_BiTree();            InitObject();
        Clean_Node();       Clean_Arc();
        m_aSetOfTaskCluster.RemoveAll();
        m_ObjectCluster.RemoveAll();
        m_ObjectBiTree.RemoveAll();
        m_ObjectTree.RemoveAll();
        m_ObjectList.RemoveAll();
        m_NodeList.RemoveAll();
        m_ArcList.RemoveAll();
}

//////////////////////////////////////////////////////
BOOL CATCSDoc::OnNewDocument()
{       if (!CDocument::OnNewDocument())
```

```
            return FALSE;
        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)
        return TRUE;
}
/////////////////////////////////////////////////////////////////////////////
// CATCSDoc serialization
void CATCSDoc::Serialize(CArchive& ar)
{       m_NodeList.Serialize(ar);
        m_ArcList.Serialize(ar);
        m_ObjectList.Serialize(ar);
        m_ObjectCluster.Serialize(ar);
        if (ar.IsStoring())
        {       // TODO: add storing code here
                ar << m_TOTAL_VERTEX
                  << m_TOTAL_ARC
                  << m_StatGraph
                  << m_DocSize
                  << m_Color
                  << m_PenWidth;
        }
        else
        {       // TODO: add loading code here
                ar >> m_TOTAL_VERTEX
                  >> m_TOTAL_ARC
                  >> m_StatGraph
                  >> m_DocSize
                  >> m_Color
                  >> m_PenWidth;
                  //>> m_SizeLevel;
        }
}
/////////////////////////////////////////////////////////////////////////////
// CATCSDoc diagnostics
#ifdef _DEBUG
void CATCSDoc::AssertValid() const
{       CDocument::AssertValid();    }
void CATCSDoc::Dump(CDumpContext& dc) const
{       CDocument::Dump(dc);         }
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CATCSDoc commands
```

```
void CATCSDoc::DeleteElement(CElement *pElement)
{       if(pElement)
        {       //If the element pointer is valid
                SetModifiedFlag();
                POSITION aPosition = m_NodeList.Find(pElement);
                m_NodeList.RemoveAt(aPosition);
                delete pElement;
        }
}
////////////////////////////////////////////////////////////////////////
void CATCSDoc::Clean_Node()
{       POSITION aPos=0;        CElement *pObj;     pObj=0;
        while(!EmptyNode())
        {       aPos = m_NodeList.GetTailPosition();
                if(aPos)
                {       pObj = m_NodeList.GetPrev(aPos);
                        m_NodeList.RemoveTail();
                }
        }
}
////////////////////////////////////////////////////////////////////////
// To delete object element list of an arc list
void CATCSDoc::DeleteArc(CElement* pElement)
{       if(pElement)
        {       SetModifiedFlag();
                POSITION aPos = m_ArcList.Find(pElement);
                m_ArcList.RemoveAt(aPos);     delete pElement;
        }
}
////////////////////////////////////////////////////////////////////////
void CATCSDoc::Clean_Arc()
{       POSITION aPos=0;        CElement *pObj;     pObj = 0;
        while(!EmptyArc())
        {       aPos = m_ArcList.GetTailPosition();
                if(aPos)
                {       pObj = m_ArcList.GetPrev(aPos);         m_ArcList.RemoveTail();
                }
        }
}
////////////////////////////////////////
// To delete object element list on object list
void CATCSDoc::InitObject()
{       POSITION aPos=0;        CElement *pObj;     pObj = 0;
```

```
        while(!EmptyObject())
        {       aPos = m_ObjectList.GetTailPosition();
                if(aPos)
                {       pObj = m_ObjectList.GetPrev(aPos);
                        m_ObjectList.RemoveTail();
                }
        }
}
////////////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::DeleteObject(CElement *pElement)
{       if(pElement)
        {       //If the element pointer is valid
                SetModifiedFlag();
                POSITION aPosition = m_ObjectList.Find(pElement);
                m_ObjectList.RemoveAt(aPosition);       delete pElement;
        }
}
////////////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::InitObjCluster()
{       POSITION aPos=0;
        CElement *pObj=0
        while(!EmptyObjCluster() )
        {       aPos = m_ObjectCluster.GetTailPosition();
                if(aPos)
                {       pObj = m_ObjectCluster.GetPrev(aPos);
                        m_ObjectCluster.RemoveTail();
                }
        }
}
////////////////////////////////////////////////////////////////////////////////////////
// To delete Object element on a task cluster
void CATCSDoc::DeleteObjCluster(CElement *pElement)
{       if(pElement)
        {       //If the element pointer is valid
                SetModifiedFlag();
                POSITION aPosition = m_ObjectCluster.Find(pElement);
                m_ObjectCluster.RemoveAt(aPosition);       delete pElement;
        }
}
////////////////////////////////////////////////////////////////////////////////////////
// To delete object element list on Tree
void CATCSDoc::Init_aSetOfTaskCluster()
{       POSITION aPos=0;
```

```
        CElement *pObj=0;
        while(!Empty_aSetOfTaskCluster())
        {       aPos = m_aSetOfTaskCluster.GetTailPosition();
                if(aPos)
                {       pObj =  m_aSetOfTaskCluster.GetPrev(aPos);
                        m_aSetOfTaskCluster.RemoveTail(); //delete all pointer
                }
        }
}
////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Delete_aSetOfTaskCluster(CElement *pElement)
{       if(pElement)
        {       //If the element pointer is valid
                SetModifiedFlag();
                POSITION aPosition = m_aSetOfTaskCluster.Find(pElement);
                m_aSetOfTaskCluster.RemoveAt(aPosition);        delete pElement;
        }
}
////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::InitObject_Tree()
{       POSITION aPos=0;
        CElement *pObj = 0;
        while(!EmptyObject_Tree())
        {       aPos = m_ObjectTree.GetTailPosition();
                if(aPos)
                {       pObj = m_ObjectTree.GetPrev(aPos);
                        m_ObjectTree.RemoveTail();
                }
        }
}
////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::DeleteObject_Tree(CElement* pElement)
{       if(pElement)
        {       //If the element pointer is valid
                SetModifiedFlag();
                POSITION aPosition = m_ObjectTree.Find(pElement);
                m_ObjectTree.RemoveAt(aPosition);       delete pElement;
        }
}
////////////////////////////////////////////////////////////////////////////////
// To delete object element list on Tree
void CATCSDoc::DeleteObject_BiTree(CElement* pElement)
{       if(pElement)
```

```
        {       //If the element pointer is valid
                SetModifiedFlag();
                POSITION aPosition = m_ObjectBiTree.Find(pElement);
                m_ObjectBiTree.RemoveAt(aPosition);     delete pElement;
        }
}
////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::InitObject_BiTree()
{       POSITION aPos=0;
        CElement *pObj = 0;
        while(!EmptyObject_BiTree())
        {       aPos = m_ObjectBiTree.GetTailPosition();
                if(aPos)
                {       pObj = m_ObjectBiTree.GetPrev(aPos);
                        m_ObjectBiTree.RemoveTail();
                }
        }
}
////////////////////////////////////////////////////////////////////////////////
BOOL CATCSDoc::Empty_StackEdge()
{       POSITION aPos;
        aPos = m_StackEdge.GetTailPosition();
        if(aPos)
                return FALSE;
        else
                return TRUE;
}
////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Push_StackEdge(CElement* pElement)
{       m_StackEdge.AddTail(pElement);          }
////////////////////////////////////////////////////////////////////////////////
CElement* CATCSDoc::Pop_StackEdge()
{       POSITION aPos;
        aPos = m_StackEdge.GetTailPosition();
        if(aPos)
        {       //return m_StackEdge.GetPrev(aPos);
                CElement *CObj = m_StackEdge.GetPrev(aPos);
                m_StackEdge.RemoveTail();       return CObj;
        }
        else    return NULL;
}
////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Del_EdgeOnStack(CElement* pElement)
```

```
{       if(pElement)
        {       //If the element pointer is valid
                POSITION aPosition = m_StackEdge.Find(pElement);
                m_StackEdge.RemoveAt(aPosition);
        }
}
////////////////////////////////////////////////////////////////////////////////
BOOL CATCSDoc::Empty_StackFather()
{       POSITION aPos;
        aPos = m_StackFather.GetTailPosition();
        if(aPos)
                return FALSE;
        else
                return TRUE;
}
////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Push_StackFather(CElement* pElement)
{       m_StackFather.AddTail(pElement);        }
////////////////////////////////////////////////////////////////////////////////
CElement* CATCSDoc::Pop_StackFather()
{       POSITION aPos;
        aPos = m_StackFather.GetTailPosition();
        if(aPos)
        {       //return m_StackFather.GetPrev(aPos);
                CElement* CObj = m_StackFather.GetPrev(aPos);
                m_StackFather.RemoveTail();     return CObj;
        }
        else
                return NULL;
}
////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Del_FatherOnStack(CElement* pElement)
{       if(pElement)
        {       //If the element pointer is valid
                POSITION aPosition = m_StackFather.Find(pElement);
                m_StackFather.RemoveAt(aPosition);
        }
}
////////////////////////////////////////////////////////////////////////////////
BOOL CATCSDoc::Empty_StackArrayTree()
{       POSITION aPos;
        aPos = m_ArrayTree.GetTailPosition();
        if(aPos)
```

```
                return FALSE;
        else
                return TRUE;
}
////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Push_RootedTree(CElement* pElement)
{       m_ArrayTree.AddTail(pElement);          }
////////////////////////////////////////////////////////////////////////////
CElement* CATCSDoc::Pop_RootedTree()
{       POSITION aPos;
        aPos = m_ArrayTree.GetTailPosition();
        if(aPos)
        {       CElement *CObj = m_ArrayTree.GetPrev(aPos);
                m_ArrayTree.RemoveTail();
                return CObj;
        }
        else
                return NULL;
}
////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Del_RootedTreeOnStack(CElement* pElement)
{       if(pElement)
        {       //If the element pointer is valid
                POSITION aPosition = m_ArrayTree.Find(pElement);
                m_ArrayTree.RemoveAt(aPosition);
        }
}
////////////////////////////////////////////////////////////////////////////
// Binary Tree
BOOL CATCSDoc::Empty_StackBinaryTree()
{       POSITION aPos;
        aPos = m_BinaryTree.GetTailPosition();
        if(aPos)
                return FALSE;
        else
                return TRUE;
}
////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Push_RootedBinaryTree(CElement* pElement)
{       m_BinaryTree.AddTail(pElement);         }

////////////////////////////////////////////////////////////////////////////
CElement* CATCSDoc::Pop_RootedBinaryTree()
```

```
{       POSITION aPos;    aPos = m_BinaryTree.GetTailPosition();
        if(aPos)
        {       CElement *CObj = m_BinaryTree.GetPrev(aPos);
                m_BinaryTree.RemoveTail();      return CObj;
        }
        else
                return NULL;
}
////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Del_RootedBinaryTreeOnStack(CElement* pElement)
{       if(pElement)
        {       //If the element pointer is valid
                POSITION aPosition = m_BinaryTree.Find(pElement);
                m_BinaryTree.RemoveAt(aPosition);
        }
}
////////////////////////////////////////////////////////////////////////////////
// about Cluster
BOOL CATCSDoc::Empty_StackCluster()
{       POSITION aPos;
        aPos = m_Cluster.GetTailPosition();
        if(aPos)
                return FALSE;
        else
                return TRUE;
}
////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Push_RootedCluster(CElement* pElement)
{       m_Cluster.AddTail(pElement);            }
////////////////////////////////////////////////////////////////////////////////
CElement* CATCSDoc::Pop_RootedCluster()
{       POSITION aPos;
        aPos = m_Cluster.GetTailPosition();
        if(aPos)
        {       //return m_StackEdge.GetPrev(aPos);
                CElement *CObj = m_Cluster.GetPrev(aPos);
                m_Cluster.RemoveTail();         return CObj;
        }
        else
                return NULL;
}
////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Del_RootedClusterOnStack(CElement* pElement)
```

```
{           if(pElement)
            {       //If the element pointer is valid
                    POSITION aPosition = m_Cluster.Find(pElement);
                    m_Cluster.RemoveAt(aPosition);
            }
}
////////////////////////////////////////////////////////////////////////////////////////
// Creating schedule for task cluster
BOOL CATCSDoc::Empty_StackSchedule()
{           POSITION aPos;
            aPos = m_ScheduleCluster.GetTailPosition();
            if(aPos)
                    return FALSE;
            else
                    return TRUE;
}
////////////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Push_RootedSchedule(CElement* pElement)
{           m_ScheduleCluster.AddTail(pElement);    }
////////////////////////////////////////////////////////////////////////////////////////
CElement* CATCSDoc::Pop_RootedSchedule()
{           POSITION aPos;
            aPos = m_ScheduleCluster.GetTailPosition();
            if(aPos)
            {       CElement *CObj = m_ScheduleCluster.GetPrev(aPos);
                    m_ScheduleCluster.RemoveTail();
                    return CObj;
            }
            else
                    return NULL;
}
////////////////////////////////////////////////////////////////////////////////////////
void CATCSDoc::Del_RootedSchedule(CElement* pElement)
{           if(pElement)
            {       //If the element pointer is valid
                    POSITION aPosition = m_ScheduleCluster.Find(pElement);
                    m_ScheduleCluster.RemoveAt(aPosition);
            }
}
////////////////////////////////////////
// Set Black color
void CATCSDoc::OnColorBlack()
{           // TODO: Add your command handler code here
```

```
        m_Color = BLACK;
}
//////////////////////////////////////////////////////////////////////
// Set Check Black color
void CATCSDoc::OnUpdateColorBlack(CCmdUI* pCmdUI)
{       // TODO: Add your command update UI handler code here
        pCmdUI->SetCheck(m_Color == BLACK);
}
//////////////////////////////////////////////////////////////////////
//  Set Blue color
void CATCSDoc::OnColorBlue()
{       // TODO: Add your command handler code here
        m_Color = BLUE;
}
//////////////////////////////////////////////////////////////////////
//  Set Check Blue color
void CATCSDoc::OnUpdateColorBlue(CCmdUI* pCmdUI)
{       // TODO: Add your command update UI handler code here
        pCmdUI->SetCheck( m_Color == BLUE);
}
//////////////////////////////////////////////////////////////////////
//  Set Green color
void CATCSDoc::OnColorGreen()
{       // TODO: Add your command handler code here
        m_Color = GREEN;
}
//////////////////////////////////////////////
// Set check green color
void CATCSDoc::OnUpdateColorGreen(CCmdUI* pCmdUI)
{       // TODO: Add your command update UI handler code here
          pCmdUI->SetCheck(m_Color == GREEN);
}
//////////////////////////////////////////////
// Set Red color
void CATCSDoc::OnColorRed()
{       // TODO: Add your command handler code here
        m_Color = RED;
}
//////////////////////////////////////////////////////////////////////
//  Set check red color
void CATCSDoc::OnUpdateColorRed(CCmdUI* pCmdUI)
{       // TODO: Add your command update UI handler code here
        pCmdUI->SetCheck(m_Color == RED);
```

```
}
////////////////////////////////////////////////////////////////////////////
// Set yellow color
void CATCSDoc::OnColorYellow()
{		// TODO: Add your command handler code here
		m_Color = YELLOW;
}
////////////////////////////////////////////////////////////////////////////
// Set check yellow color
void CATCSDoc::OnUpdateColorYellow(CCmdUI* pCmdUI)
{		// TODO: Add your command update UI handler code here
		pCmdUI->SetCheck(m_Color == YELLOW);
}
////////////////////////////////////////////////////////////////////////////
void CATCSDoc::OnColorWhite()
{		// TODO: Add your command handler code here
		m_Color = WHITE;
}
void CATCSDoc::OnUpdateColorWhite(CCmdUI* pCmdUI)
{		// TODO: Add your command update UI handler code here
		pCmdUI->SetCheck(m_Color == WHITE);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//		ATCSView.h
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// ATCSView.h : interface of the CATCSView class
////////////////////////////////////////////////////////////////////////////
#if !defined(AFX_ATCSVIEW_H__DA60A49F_30EB_11D3_93C5_00A0C92CFD9E__INCLUDED_)
#define AFX_ATCSVIEW_H__DA60A49F_30EB_11D3_93C5_00A0C92CFD9E__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "Constants.h"
class CDigraph;
class CATCSView : public CScrollView
{
protected: // create from serialization only
		CATCSView();
		DECLARE_DYNCREATE(CATCSView)
		void ResetScrollSizes();
```

```
        BOOL InitMatrixDigraph(CElement *pElement);
        void InitArrayRootedTree();
        BOOL InitMatrixDigraph(CElement *pElement, int pMatrix[][MAX_MATRIX]);
// Attributes
public:
        CATCSDoc* GetDocument();
protected:
        int       m_Task_No;
        CElement  *m_pTempNode;
        CElement  *m_pTempArc;
        CDigraph  *m_TempDigraph;
        int       *m_Matrix[MAX_MATRIX];
        CDigraph  *m_Digraph;

        CElement  *m_Original_DiGraph;
        CElement  *m_TmpVertex;
        CElement  *m_Vertex;
        CEdge     *m_Edge;
        int       m_ArrayTask[MAX_MATRIX];
        int       m_MatrixDigraph[MAX_MATRIX][MAX_MATRIX];
        int       m_RootedTree[MAX_MATRIX];

        CTree     *m_Tree;
        CTree     *m_BiTree;
        CElement  *m_VertexTree;
        CCluster  *m_Cluster;

        CElement  **m_ArrayTree;
        CElement  *m_NodeCluster;
public:
        int m_nCurrentCol;          int m_nCurrentRow;          int m_nRibbonWidth;
        int m_nCellHeight;          int m_nCellWidth;
        int m_nMakeSpan;
        int m_nScheduleTime;
        double m_nSpeedUp;
        int m_nFinalCluster;
        int m_nSchedule_Single_P;
// Operations
public:
        void GetCellRect(int row, int col, LPRECT pRect);
        int  Get_MakeSpan(CSingleNode *aArrayNode, int aSize);
        BOOL Check_Source_Node(CSingleNode *pArray_Source, int aSize, CElement *pNode);
        CLinkList *Read_Source_Node(CLinkList *pLinkList, CElement *pNode);
```

```
        BOOL Check_Is_On_Cluster(CElement *pCluster, CLinkList *pNode);
        int Get_Finished_Time_Of_Outside_Cluster(CSingleNode *pSinkNode, int aSize, int pPredecessor);
        int Communication_Time_Of_Node_Outside_Cluster(int pSource, int pDestinate);
        BOOL Insert_Finished_Time_Onto_Cluster(CSingleNode *pArraySinkNode, int aSize, int pSinkNode,
                                                int aFinished_Time);
        int Get_Makespan_DAG();
        BOOL Difference_Level_DAG(int aDestinate_TaskNo, int aCurrentLevel);
        int Get_ExecuteCost(int aTask_No);
        int Get_Random_Communicate(int aRange_MinExecute, int aRange_MaxExecute, int aSource_Execute,
                                    int aDestinate_Execute, BOOL aFine_Grain, BOOL aCoarse_Grain);
        BOOL Check_Duplicate_Node(int *pArrayNode, int aDestinate_Node, int aSize);
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CATCSView)
public:
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
        virtual void OnInitialUpdate(); // called first time after construct
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
        //}}AFX_VIRTUAL
protected:
        CElement* Create_Node(int task_no, int execute_cost, int vertex_level);
        CElement* Create_Arc(int commu_cost, int source_node, int destination_node);
        BOOL   Allocate_Matrix(int size);
        void   Add2Matrix(int task_no, int vertex_level);
// Implementation
public:
        virtual ~CATCSView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
protected:
// Generated message map functions
protected:
        BOOL m_bSmooth;
        void CreateHeadArrow(UINT max1, UINT max2, CEdge *pEdge, CRect aRect, CPoint aThirdPoint=0);
        CFont m_font;
        //{{AFX_MSG(CATCSView)
```

```
        afx_msg void OnCreateGraph();
        afx_msg void OnAddVertex();
        afx_msg void OnJointVertex();
        afx_msg void OnShowGraph();
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnShowTree();
        afx_msg void OnShowBinaryTree();
        afx_msg void OnShowCluster();
        afx_msg void OnShowSchedule();
        afx_msg void OnRandomdag();
        //}}AFX_MSG
        afx_msg void OnUpdateSingleProcessor(CCmdUI* pCmdUI);
        afx_msg void OnUpdateMakespan(CCmdUI* pCmdUI);
        afx_msg void OnUpdateSchedule(CCmdUI* pCmdUI);
        afx_msg void OnUpdateSpeedup(CCmdUI* pCmdUI);
        afx_msg void OnUpdateSetCluster(CCmdUI* pCmdUI);
        DECLARE_MESSAGE_MAP()
};
#ifndef _DEBUG  // debug version in ATCSView.cpp
inline CATCSDoc* CATCSView::GetDocument()
   { return (CATCSDoc*)m_pDocument; }
#endif
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif // !defined(AFX_ATCSVIEW_H__DA60A49F_30EB_11D3_93C5_00A0C92CFD9E__INCLUDED_)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//              ATCSView.cpp File
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// ATCSView.cpp : implementation of the CATCSView class
//
#include "stdafx.h"
#include "ATCS.h"
#include "Graph.h"
#include "ATCSDoc.h"
#include "ATCSView.h"
#include "DlgCreateGraph.h"
#include "DlgAddNode.h"
#include "DlgJointArc.h"
#include "GeneratRandomDAG.h"

#include <stdio.h>
#include <string.h>
```

```
#include <math.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
////////////////////////////////////////////////////////////////////////
// CATCSView
IMPLEMENT_DYNCREATE(CATCSView, CScrollView)
BEGIN_MESSAGE_MAP(CATCSView, CScrollView)
        //{{AFX_MSG_MAP(CATCSView)
        ON_COMMAND(ID_CREATE_GRAPH, OnCreateGraph)
        ON_COMMAND(ID_ADD_VERTEX, OnAddVertex)
        ON_COMMAND(ID_JOINT_VERTEX, OnJointVertex)
        ON_COMMAND(ID_SHOW_GRAPH, OnShowGraph)
        ON_WM_LBUTTONDOWN()
        ON_COMMAND(ID_SHOW_TREE, OnShowTree)
        ON_COMMAND(ID_SHOW_BINARY_TREE, OnShowBinaryTree)
        ON_COMMAND(ID_SHOW_CLUSTER, OnShowCluster)
        ON_COMMAND(ID_SHOW_SCHEDULE, OnShowSchedule)
        ON_COMMAND(ID_RANDOMDAG, OnRandomdag)
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)

        ON_UPDATE_COMMAND_UI(ID_INDICATOR_SINGLE_P, OnUpdateSingleProcessor)
        ON_UPDATE_COMMAND_UI(ID_INDICATOR_MAKESPAN, OnUpdateMakespan)
        ON_UPDATE_COMMAND_UI(ID_INDICATOR_SCHEDULE, OnUpdateSchedule)
        ON_UPDATE_COMMAND_UI(ID_INDICATOR_SPEEDUP,  OnUpdateSpeedup)
        ON_UPDATE_COMMAND_UI(ID_INDICATOR_SETCLUSTER, OnUpdateSetCluster)
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////
// CATCSView construction/destruction
CATCSView::CATCSView()
{       // TODO: add construction code here
        m_pTempNode = 0;
        SetScrollSizes(MM_TEXT, CSize(0,0));
        m_font.CreatePointFont(40, _T("MS Sans Serif"));
        m_Vertex  = 0;  m_Edge = 0;  m_Original_DiGraph = NULL;  m_TmpVertex  = NULL;
        m_Digraph = 0;      m_Tree = 0;             m_BiTree = 0;          m_VertexTree = 0;
        m_Cluster = 0;      m_nMakeSpan = 0;                m_nScheduleTime = 0;
```

```
        m_nSpeedUp = 0;  m_nFinalCluster  = 0;
}
////////////////////////////////////////////////////////////////
CATCSView::~CATCSView()
{       int **m_pTmp;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        m_pTmp = m_Matrix;
        if(m_pTmp)
        {       for(int i=0; i < m_Task_No; i++)    delete *m_pTmp++;   }
        // Delete a Diagraph on OnJointVertex( )
        CElement *pPoint = NULL;    CEdge   *pEdge = NULL;        CEdge   *aEdge = NULL;
        CElement *pGraph = m_Vertex;
        while(pGraph)
        {       pPoint = pGraph;
                pEdge = ((CVertex*)pPoint)->m_First_edge;
                while(pEdge)
                {       aEdge = pEdge;
                        pEdge = pEdge->m_Next_edge;
                        delete aEdge;           aEdge = 0;
                }
                pGraph = ((CVertex*)pGraph)->m_Next_vertex;
                delete pPoint;          pPoint = 0;
        }
        m_Vertex = NULL;
        if(m_Cluster)
        {       CElement *pRoot;
                while(!pDoc->Empty_StackSchedule())
                {       pRoot = NULL;
                        pRoot = pDoc->Pop_RootedSchedule();
                        if(pRoot)
                                m_Cluster->Delete_Digraph(pRoot);
                }
                // Delete BiTree
                delete m_Cluster;       m_Cluster = 0;
        }
        if(m_Digraph)
        {       delete m_Digraph;       m_Digraph = 0;      }
        if(m_Tree)
        {       delete m_Tree;          m_Tree = 0;         }
        if(m_BiTree)
        {       delete m_BiTree;        m_BiTree = 0;       }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CATCSView::PreCreateWindow(CREATESTRUCT& cs)
{       // TODO: Modify the Window class or styles here by modifying  the CREATESTRUCT cs
        return CScrollView::PreCreateWindow(cs);
}

////////////////////////////////////////////////////////////////////////////
// CATCSView drawing
void CATCSView::OnDraw(CDC* pDC)
{       CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        // TODO: add draw code for native data here
        CSize size = GetTotalSize();  //get view's logical width and high from SetScrollSize() that gives total pixel
        CPen pen(PS_SOLID, 0, RGB(192,192,192));
        CPen* pOldPen = pDC->SelectObject(&pen);
        for(int i=0; i<99; i++)
        {       int y=(i*m_nCellHeight) + m_nCellHeight;
                pDC->MoveTo(0, y);
                pDC->LineTo(size.cx, y); // size and y are total pixels or point coordinate in pixels
        }
        for(int j=0; j<99; j++)
        {       int x = (j*m_nCellWidth) + m_nRibbonWidth;
                pDC->MoveTo(x,0);
                pDC->LineTo(x, size.cy); // size and x are total pixels or point coordinate in pixels
        }
        pDC->SelectObject(pOldPen);

        //Draw the bodies of the rows and column headers
        CBrush brush;
        brush.CreateStockObject(LTGRAY_BRUSH);

        CRect rcTop(0, 0, size.cx, m_nCellHeight);          pDC->FillRect(rcTop, &brush);
        CRect rcLeft(0, 0, m_nRibbonWidth, size.cy);        pDC->FillRect(rcLeft, &brush);

        pDC->MoveTo(0, m_nCellHeight);      pDC->LineTo(size.cx, m_nCellHeight);
        pDC->MoveTo(m_nRibbonWidth, 0);     pDC->LineTo(m_nRibbonWidth, size.cy);
        pDC->SetBkMode(TRANSPARENT);

        //Add numbers and button outlines to the row headers
        //
        for(i=0; i<99; i++)
        {
                int y = (i*m_nCellHeight) + m_nCellHeight;
```

```
        pDC->MoveTo(0,y);
        pDC->LineTo(m_nRibbonWidth, y); // y and m_nRibbonWidth are a number of pixels or point coordinate
                                        // in pixels
        CString string;
        string.Format(_T("%d"), i+ 1);

        CRect rect(0,y, m_nRibbonWidth, y+m_nCellHeight);
        pDC->DrawText(string, &rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
        rect.top++;
        pDC->Draw3dRect(rect, RGB(255,255,255), RGB(128,128,128));
}

//Add letters and button outlines to the column headers
for(j=0; j<99; j++)
{       int x = (j * m_nCellWidth) + m_nRibbonWidth;
        pDC->MoveTo(x , 0);
        pDC->LineTo(x , m_nCellHeight);
        CString string;         string.Format(_T("%d"), j+1);
        CRect rect(x,0,x+m_nCellWidth, m_nCellHeight);
        pDC->DrawText(string, &rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
        rect.left++;            pDC->Draw3dRect(rect, RGB(255,255,255), RGB(128, 128, 128));
}
// Draw addres labels into the individuals cells
CRect rect;
pDC->GetClipBox(&rect);// rect from GetClipBox is region needs to be redraw
int nStartRow = max(0, (rect.top - m_nCellHeight)/m_nCellHeight);
int nEndRow   = min(98, (rect.bottom - 1)/m_nCellHeight);
int nStartCol = max(0, (rect.left - m_nRibbonWidth) / m_nCellWidth);
int nEndCol   = min(25, ((rect.right + m_nCellWidth - 1) -  m_nRibbonWidth) / m_nCellWidth);
if(!pDoc->m_StatGraph)
        return;
CElement* pElement=0;       POSITION aPos=0;
switch(pDoc->m_StageShown)
{       case 1:     aPos = pDoc->GetListHeadObjectPosition();
                    while(aPos)
                    {       pElement = pDoc->GetNextObject(aPos);
                            //if the element is visible
                            if(pDC->RectVisible(pElement->GetBoundRect()) )
                                    pElement->Draw(pDC, &m_font);
                    }
                    break;
        case 2:     aPos = pDoc->GetHeadObject_TreePosition();
                    while(aPos)
```

```
                {       pElement = pDoc->GetNextObject_Tree(aPos);
                        //if the element is visible
                        if(pDC->RectVisible(pElement->GetBoundRect()) )
                                pElement->Draw(pDC, &m_font);

                }
                break;
        case 3: aPos = pDoc->GetHeadObject_BiTreePosition();
                while(aPos)
                {       pElement = pDoc->GetNextObject_BiTree(aPos);
                        if(pDC->RectVisible(pElement->GetBoundRect()) )
                                pElement->Draw(pDC, &m_font);
                }
                break;
        case 4: aPos = pDoc->GetListHeadObjClusterPosition();
                while(aPos)
                {       pElement = pDoc->GetNextObjCluster(aPos);
                        if(pDC->RectVisible(pElement->GetBoundRect()) )
                                pElement->Draw(pDC, &m_font);
                }
                break;
        case 5: aPos = pDoc->Get_aSetOfTaskCluster();
                while(aPos)
                {       pElement = pDoc->GetNext_aSetOfTaskCluster(aPos);
                        if(pDC->RectVisible(pElement->GetBoundRect()) )
                                pElement->Draw(pDC, &m_font);
                }
                break;
        default: break;
        }
}
////////////////////////////////////////////////////////////////////////
void CATCSView::OnInitialUpdate()
{       ResetScrollSizes();     CScrollView::OnInitialUpdate();
}
////////////////////////////////////////////////////////////////////////
// CATCSView printing
BOOL CATCSView::OnPreparePrinting(CPrintInfo* pInfo)
{       return DoPreparePrinting(pInfo);
}
////////////////////////////////////////////////////////////////////////
void CATCSView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{       // TODO: add extra initialization before printing
```

```
}
////////////////////////////////////////////////////////////////////////
void CATCSView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{       // TODO: add cleanup after printing
}
////////////////////////////////////////////////////////////////////////
// CATCSView diagnostics
#ifdef _DEBUG
void CATCSView::AssertValid() const
{       CScrollView::AssertValid();
}
void CATCSView::Dump(CDumpContext& dc) const
{       CScrollView::Dump(dc);
}
CATCSDoc* CATCSView::GetDocument() // non-debug version is inline
{       ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CATCSDoc)));
        return (CATCSDoc*)m_pDocument;
}
#endif //_DEBUG
////////////////////////////////////////////////////////////////////////
// CATCSView message handlers
void CATCSView::ResetScrollSizes()
{       m_nCurrentRow = 0;  m_nCurrentCol = 0;   m_bSmooth    = FALSE;
        CClientDC dc(this);
        m_nCellWidth = dc.GetDeviceCaps(LOGPIXELSX)/4;   //a number of pixels on width cell
        m_nCellHeight = dc.GetDeviceCaps(LOGPIXELSY)/4;  // a number of pixels on heigh cell
        m_nRibbonWidth = m_nCellWidth;
        // a number of pixels on Logical width view
        int nWidth = (99 * m_nCellWidth) + m_nRibbonWidth;
        // a number of pixels on Logical width heigh
        int nHeight = m_nCellHeight * 100;
        // CSize (nWidth, nHeight) is to set logical view or view dimension
        SetScrollSizes(MM_TEXT, CSize(nWidth, nHeight));
}
////////////////////////////////////////////////////////////////
void CATCSView::OnCreateGraph()
{       CATCSDoc* pDoc = GetDocument();     ASSERT_VALID(pDoc);
        CDlgCreateGraph aDlg;
        aDlg.m_TOTAL_VERTEX = pDoc->m_TOTAL_VERTEX;
        aDlg.m_TOTAL_ARC    = pDoc->m_TOTAL_ARC;
        if(pDoc->m_StatGraph == FALSE)
        {       if(aDlg.DoModal() == IDOK)
                { // save to Document
```

```
                pDoc->m_TOTAL_VERTEX = aDlg.m_TOTAL_VERTEX;
                pDoc->m_TOTAL_ARC    = aDlg.m_TOTAL_ARC;
                pDoc->m_StatGraph    = TRUE;
                for(int i=0; i< pDoc->m_TOTAL_VERTEX; i++)
                {       m_pTempNode = Create_Node(i+1, 0, 0);
                        pDoc->AddNode(m_pTempNode);
                        m_pTempNode = 0;
                }
                for(int j=0; j< pDoc->m_TOTAL_ARC; j++)
                {       m_pTempArc = Create_Arc(0,0,0);
                        pDoc->AddArc(m_pTempArc);               m_pTempArc = 0;
                }
            }
        }
        else
                AfxMessageBox("You can't create a new GRAPH");
}
////////////////////////////////////////////////////////////////////////////
// Allocate Matrix
BOOL CATCSView::Allocate_Matrix(int size)
{   int **m_pTmp;
    m_pTmp = m_Matrix;
    if(m_pTp)
    {       for(int j=0; j < m_Task_No; j++)
            {       *m_pTmp = new int[size+1];
                    memset(*m_pTmp, 0, sizeof(int)* size );
                    m_pTmp++;
            }
            return  TRUE;
     }
    return FALSE;
}
////////////////////////////////////////////////////////////////////////////
// Create_Node is used for allocate a new node
CElement* CATCSView::Create_Node(int task_no, int execute_cost, int vertext_level)
{       CATCSDoc* pDoc = GetDocument();     ASSERT_VALID(pDoc);
        return new CNode(task_no, execute_cost,vertext_level, pDoc->GetColor());
}
////////////////////////////////////////////////////////////////////////////
CElement* CATCSView::Create_Arc(int source_node, int destination_node, int commu_cost)
{       CATCSDoc* pDoc = GetDocument();     ASSERT_VALID(pDoc);
        return new CArc(source_node, destination_node, commu_cost);
}
```

```
////////////////////////////////////////////////////////////////////////
void CATCSView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{       // TODO: Add your specialized code here and/or call the base class
}
////////////////////////////////////////////////////////////////////////

void CATCSView::Add2Matrix(int task_no, int vertex_level)
{       int **m_pMatrix, index;
        m_pMatrix = m_Matrix + vertex_level;
        if(*m_pMatrix)
        {       // increase index for row matrix
                for(index = 0; (*((*m_pMatrix) + index)) != 0; index++);
                (*((*m_pMatrix) + index)) = task_no;  //fill into matrix
        }
}
////////////////////////////////////////////////////////////////////////
// The OnAddVertex( ) is called when the user click Add Vexter to a DAG
//
void CATCSView::OnAddVertex()
{       // TODO: Add your command handler code here
        int i=0;  int Old_TotalVertex=0;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        CDlgAddNode aDlg;
        CElement *pElement=0;
        // All data of Document that is send to DialogBox
        aDlg.m_TOTAL_VERTEX = pDoc->m_TOTAL_VERTEX;
        Old_TotalVertex     = pDoc->m_TOTAL_VERTEX;  // save old amount vertex
        aDlg.m_StatGraph    = pDoc->m_StatGraph;
        POSITION aPos       = pDoc->GetListHeadPosition();
        while(aPos)
        {       pElement = pDoc->GetNext(aPos);
                aDlg.m_ChuntData[i][0] = ((CNode*)pElement)->m_Task_no;  // fixed sequence no.
                aDlg.m_ChuntData[i][1] = ((CNode*)pElement)->m_Execute_cost;  // vary execution cost
                aDlg.m_ChuntData[i][2] = ((CNode*)pElement)->m_Vertex_level;  // vary vertex level
                i++;
        }
        if(aDlg.DoModal() == IDOK)
        {       pDoc->m_TOTAL_VERTEX = aDlg.m_TOTAL_VERTEX;
                pDoc->m_StatGraph    = aDlg.m_StatGraph;
                i=0; aPos=0;
                if(Old_TotalVertex < pDoc->m_TOTAL_VERTEX)
                {       aPos = pDoc->GetListHeadPosition();
```

```
            while(aPos)  // for old amount
            {  pElement = pDoc->GetNext(aPos);
               ((CNode*)pElement)->m_Task_no     = aDlg.m_ChuntData[i][0];
               ((CNode*)pElement)->m_Execute_cost = aDlg.m_ChuntData[i][1];
               ((CNode*)pElement)->m_Vertex_level = aDlg.m_ChuntData[i][2];
               i++;
            }

            // for new amount
            for(; i< pDoc->m_TOTAL_VERTEX; i++)
            {  m_pTempNode = Create_Node(aDlg.m_ChuntData[i][0], aDlg.m_ChuntData[i][1],
                                                                    aDlg.m_ChuntData[i][2]);
               pDoc->AddNode(m_pTempNode);
               m_pTempNode = 0;
            }
}
else if(Old_TotalVertex > pDoc->m_TOTAL_VERTEX)
{
      for(i=0; i< (Old_TotalVertex - (pDoc->m_TOTAL_VERTEX)); i++)
      {   POSITION aPos = pDoc->GetListTailPosition();
          if(aPos)
          {         pElement = pDoc->GetPrev(aPos);
                    pDoc->DeleteElement(pElement);
          }
      }
      i = 0; aPos = 0;
      aPos = pDoc->GetListHeadPosition();
      while(aPos)
      {   pElement = pDoc->GetNext(aPos);
          ((CNode*)pElement)->m_Task_no     = aDlg.m_ChuntData[i][0];
          ((CNode*)pElement)->m_Execute_cost = aDlg.m_ChuntData[i][1];
          ((CNode*)pElement)->m_Vertex_level = aDlg.m_ChuntData[i][2];
          i++;
      }
}
else // Old_TotalVertex == pDoc->m_TOTAL_VERTEX
{         for(i=0; i< (pDoc->m_TOTAL_VERTEX); i++)
          {         POSITION aPos = pDoc->GetListTailPosition();
                    if(aPos)
                    {         pElement = pDoc->GetPrev(aPos);
                              pDoc->DeleteElement(pElement);
                    }
          }
```

```
                    for(i=0; i< (pDoc->m_TOTAL_VERTEX) : i++)
                    {       m_pTempNode = Create_Node(aDlg.m_ChuntData[i][0], aDlg.m_ChuntData[i][1],
                                                                                    aDlg.m_ChuntData[i][2]);
                            pDoc->AddNode(m_pTempNode);
                            m_pTempNode = 0;
                    }
                }
                return;
        }
}
////////////////////////////////////////////////////////////////////////////////////////////////
// The OnJointVertex( ) is called when the user click Joint Vertex Button, that this
// function gives user enter several commumication cost between nodes on a DAG.
//
void CATCSView::OnJointVertex()
{       // TODO: Add your command handler code here
        int i=0;   int Old_TotalArc=0;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        CDlgJointArc aDlg;
        CElement*  pElement=0;
        aDlg.m_TOTAL_ARC = pDoc->m_TOTAL_ARC;
        Old_TotalArc    = pDoc->m_TOTAL_ARC;
        POSITION aPos   = pDoc->GetArcListHeadPosition();
        while(aPos)
        {       pElement = pDoc->GetArcNext(aPos);
                aDlg.m_ChuntArc[i][0] = ((CArc*)pElement)->m_Start_vertex;  // vary execution cost
                aDlg.m_ChuntArc[i][1] = ((CArc*)pElement)->m_End_vertex;  // vary vertex level
                aDlg.m_ChuntArc[i][2] = ((CArc*)pElement)->m_Commu_cost;  // fixed sequence no.
                i++;
        }
        if(aDlg.DoModal() == IDOK)
        {       i=0; aPos=0;
                pDoc->m_TOTAL_ARC = aDlg.m_TOTAL_ARC;
                if(Old_TotalArc < pDoc->m_TOTAL_ARC)  // Add vertex
                {       aPos = pDoc->GetArcListHeadPosition();
                        while(aPos)  // for old amount
                        {       pElement = pDoc->GetArcNext(aPos);
                                ((CArc*)pElement)->m_Start_vertex  = aDlg.m_ChuntArc[i][0];
                                ((CArc*)pElement)->m_End_vertex    = aDlg.m_ChuntArc[i][1];
                                ((CArc*)pElement)->m_Commu_cost    = aDlg.m_ChuntArc[i][2];
                                i++;
                        }
```

```
                    // for new amount
                    for(; i< pDoc->m_TOTAL_ARC; i++ )
                    {       m_pTempArc = Create_Arc(aDlg.m_ChuntArc[i][0], aDlg.m_ChuntArc[i][1],
                                                                    aDlg.m_ChuntArc[i][2]);
                            pDoc->AddArc(m_pTempArc);
                            m_pTempArc = 0;
                    }
            }
            else if( Old_TotalArc > pDoc->m_TOTAL_ARC) // Delete vertex
            {       for(i=0; i< (Old_TotalArc - (pDoc->m_TOTAL_ARC)); i++)
                    {       POSITION aPos = pDoc->GetArcListTailPosition();
                            if(aPos)
                            {       pElement = pDoc->GetArcPrev(aPos);
                                    pDoc->DeleteArc(pElement);
                            }
                    }
                    i = 0; aPos = 0;        aPos = pDoc->GetArcListHeadPosition();
                    while(aPos)
                    { pElement = pDoc->GetArcNext(aPos);
                      ((CArc*)pElement)->m_Start_vertex  = aDlg.m_ChuntArc[i][0];
                      ((CArc*)pElement)->m_End_vertex    = aDlg.m_ChuntArc[i][1];
                      ((CArc*)pElement)->m_Commu_cost    = aDlg.m_ChuntArc[i][2];
                      i++;
                    }
            }
            else // Old_TotalArc == pDoc->m_TOTAL_ARC
            {       for(i=0; i< (pDoc->m_TOTAL_ARC); i++)
                    {       POSITION aPos = pDoc->GetArcListTailPosition();
                            if(aPos)
                            {       pElement = pDoc->GetArcPrev(aPos);
                                    pDoc->DeleteArc(pElement);
                            }
                    }
            for(i=0; i< (pDoc->m_TOTAL_ARC); i++ )
            {       m_pTempArc = Create_Arc(aDlg.m_ChuntArc[i][0], aDlg.m_ChuntArc[i][1],
                                                                    aDlg.m_ChuntArc[i][2]);
                    pDoc->AddArc(m_pTempArc);
                    m_pTempArc = 0;
            }
    }
    return;
  }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
// The OnShowGraph( ) is called when the user click Show Graph button.  OnShowGraph( )
// is used when user enter all component on a DAG finish.
//
void CATCSView::OnShowGraph()
{       // TODO: Add your command handler code here
        BOOL    nFlageMake = FALSE;
        CElement* pSource;          CElement* pDestinate;       CElement* pElement=0;
        CRect   aRect=0;            CPoint   aPointStart=0;     CPoint   aPointEnd=0;
        int     aMax=0, aLevel=0;   int      aCol = 0, aRow = 0;
        int     aSegment=0, aOffset=0;          int     aIndex=0;
        CString  err_msg;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        int total_vertex    = pDoc->m_TOTAL_VERTEX;
        COLORREF color_vertex = pDoc->m_Color;
        if(!pDoc->m_StatGraph) // don't create graph
        {       AfxMessageBox(_T("You does not create nodes and arcs."));
                return;
        }
        // initialize severals objects
        if(m_Cluster)
        {       CElement *pRoot;
                while(!pDoc->Empty_StackSchedule())
                {       pRoot = NULL;
                        pRoot = pDoc->Pop_RootedSchedule();
                        if(pRoot)
                                m_Cluster->Delete_Digraph(pRoot);
                }
                delete m_Cluster;                   m_Cluster = 0;
        }

        pDoc->m_StateTREE    = TRUE;        pDoc->m_StateBITREE   = FALSE;
        pDoc->m_StateCLUSTER  = FALSE;      pDoc->m_StateSCHEDULE = FALSE;
        // Initialize object
        pDoc->InitObject();
        // delete all nodes and edges on Digraph
        // delete CDigraph
        if( m_Original_DiGraph && m_Digraph)
        {       if(m_Original_DiGraph)
                {       m_Digraph->Delete_Digraph(m_Original_DiGraph);
                        m_Original_DiGraph = NULL;
```

```
        }
        delete m_Digraph;                m_Digraph = 0;
}

CElement *aPoint = NULL;    CEdge    *pEdge_x =  NULL;    CEdge    *aEdge = NULL;
CElement *pGraph = m_TmpVertex;
while(pGraph)
{       aPoint = pGraph;
        pEdge_x = ((CVertex*)aPoint)->m_First_edge;
        while(pEdge_x)
        {       aEdge = pEdge_x;
                pEdge_x = pEdge_x->m_Next_edge;
                delete aEdge;           aEdge = 0;
        }

        pGraph = ((CVertex*)pGraph)->m_Next_vertex;
        delete aPoint;          aPoint = NULL;
}
m_TmpVertex = NULL;

// connection all node on a graph
m_Digraph = new CDigraph(total_vertex, color_vertex);
POSITION aPos = pDoc->GetListTailPosition();
while(aPos)
{       pElement = pDoc->GetPrev(aPos);
        if(!nFlageMake)
        {       nFlageMake = TRUE;
                m_Vertex = m_Digraph->Make_Graph(pElement);
        }
        else
                m_Vertex = m_Digraph->Add_Vertex(m_Vertex, pElement);
}
// Connection arc between all nodes
aPos = pDoc->GetArcListHeadPosition();
while(aPos)
{       pSource = pDestinate =0;
        pElement = pDoc->GetArcNext(aPos);
        pSource   = m_Digraph->Find_Vertex(m_Vertex,((CArc*)pElement)->m_Start_vertex);
        pDestinate = m_Digraph->Find_Vertex(m_Vertex,((CArc*)pElement)->m_End_vertex);
        if((pSource != NULL) && (pDestinate != NULL))
                        m_Digraph->Joint_Vertex(pSource, pDestinate, pElement);
}
// Now, a m_Vertex is final a graph that is connected all
```

```
// Save Original Digraph
m_Original_DiGraph = NULL;              m_Original_DiGraph = m_Vertex;
CEiement* pPoint;               pPoint = m_Vertex;
for(int i=0; i< MAX_MATRIX; i++)
                        m_ArrayTask[i] = 0;
// allocate memory
int **aCell, *aTable[MAX_MATRIX];
aCell = aTable;
for(i=0;i<MAX_MATRIX;i++)
{       *aCell = new int[MAX_MATRIX];
        memset(*aCell, 0, sizeof(int)*MAX_MATRIX);
        aCell++;
}

aCell = aTable;
while(pPoint)
{       m_ArrayTask[((CVertex*)pPoint)->m_Vertex_level] += 1;
        if(((CVertex*)pPoint)->m_Vertex_level > aLevel)
                        aLevel++; //increase level
        if(m_ArrayTask[((CVertex*)pPoint)->m_Vertex_level] > aMax)
                        aMax++;
        aCell = aCell + ((CVertex*)pPoint)->m_Vertex_level;
        for(i=0; *((*aCell)+i) != NULL; i++);
                *((*aCell)+i) = ((CVertex*)pPoint)->m_Task_no;
        aCell = aTable;
        pPoint = ((CVertex*)pPoint)->m_Next_vertex; // next
}

int   j=0;          aCell = aTable;       pPoint = m_Vertex;   aLevel++;
pDoc->m_SizeLevel = aLevel - 1;
for(i=0; i< aLevel;i++)
{       aRow = (i*3) + 2;       aSegment = (m_ArrayTask[i] / 2);    aOffset  = (m_ArrayTask[i] % 2);
        j=0;
        if(aOffset > 0)
        {       for(aIndex=aSegment, j=0 ; aIndex > 0; aIndex--, j++)
                {       aCol = 15 - (aIndex*3);
                        GetCellRect(aRow, aCol, &aRect);
                        // Find vertex on a graph
                        pSource = 0;
                        pSource = m_Digraph->Find_Vertex(m_Vertex, *((*aCell)+j) );
                        ((CVertex*)pSource)->m_Rect = aRect;
                        ((CVertex*)pSource)->SetBoundRect(aRect);
                        pDoc->AddObject(pSource);
```

```
            }
            aCol = 15;
            GetCellRect(aRow, aCol, &aRect);
            pSource = 0;
            pSource = m_Digraph->Find_Vertex(m_Vertex, *((*aCell)+j) ):
            ((CVertex*)pSource)->m_Rect = aRect;
            ((CVertex*)pSource)->SetBoundRect(aRect);
            pDoc->AddObject(pSource);
            j++;
            for(aIndex=1; aIndex < (aSegment+1); aIndex++, j++)
            {       aCol = 15 + (aIndex*3);
                    GetCellRect(aRow, aCol, &aRect);
                    //Find vertex on a graph
                    pSource = 0;
                    pSource = m_Digraph->Find_Vertex(m_Vertex, *((*aCell)+j) );
                    ((CVertex*)pSource)->m_Rect = aRect;
                    ((CVertex*)pSource)->SetBoundRect(aRect);
                    pDoc->AddObject(pSource);
            }
    }
    else // offset equals zero
    {       for(aIndex=aSegment, j=0 ; aIndex > 0; aIndex--, j++)
            {       aCol = 15 - (aIndex*2);
                    GetCellRect(aRow, aCol, &aRect);
                    // Find vertex on a graph
                    pSource = 0;
                    pSource = m_Digraph->Find_Vertex(m_Vertex, *((*aCell)+j) );
                    ((CVertex*)pSource)->m_Rect = aRect;
                    ((CVertex*)pSource)->SetBoundRect(aRect);
                    pDoc->AddObject(pSource);
            }
            for(aIndex=1; aIndex < (aSegment+1); aIndex++, j++)
            {       aCol = 15 + (aIndex*2);
                    GetCellRect(aRow, aCol, &aRect);
                    //Find vertex on a graph
                    pSource = 0;
                    pSource = m_Digraph->Find_Vertex(m_Vertex, *((*aCell)+j) );
                    ((CVertex*)pSource)->m_Rect = aRect;
                    ((CVertex*)pSource)->SetBoundRect(aRect);
                    pDoc->AddObject(pSource);
            }
    }// end if else
    aCell++;
```

```
}// end for
// Calculate start point and end point, then drawing edge on a graph
CElement* pVertex;              CEdge*   pEdge;
UINT  min1, min2, max1, max2;
pVertex = m_Vertex;  // get root a Graph
while(pVertex)
{       pEdge = ((CVertex*)pVertex)->m_First_edge;
        while(pEdge)
        {       aPointStart.x = ((CVertex*)pVertex)->m_Rect.right - (m_nCellWidth/2);
                aPointStart.y = ((CVertex*)pVertex)->m_Rect.bottom;
                aPointEnd.x = ((pEdge->m_End_point)->m_Rect.left + (m_nCellWidth/2)) -1;
                aPointEnd.y = (pEdge->m_End_point)->m_Rect.top;
                pEdge->m_StartPoint = aPointStart;
                pEdge->m_EndPoint   = aPointEnd;

                // calculate enclosing boundary
                min1 = min2 = max1 = max2 =0;
                min1 = min( ((CVertex*)pVertex)->m_Rect.right,
                            ((CVertex*)pVertex)->m_Rect.left );
                min2 = min( (pEdge->m_End_point)->m_Rect.right,
                            (pEdge->m_End_point)->m_Rect.left);
                max1 = max( ((CVertex*)pVertex)->m_Rect.right,
                            ((CVertex*)pVertex)->m_Rect.left);
                max2 = max( (pEdge->m_End_point)->m_Rect.right,
                            (pEdge->m_End_point)->m_Rect.left);

                aRect.left = min(min1, min2);     aRect.right = max(max1, max2);
                aRect.top = ((CVertex*)pVertex)->m_Rect.bottom;
                aRect.bottom = (pEdge->m_End_point)->m_Rect.top;

                // create head arrow of each edge
                CreateHeadArrow(max1, max2, pEdge, aRect);
                pEdge->SetBoundRect(aRect);
                pDoc->AddObject(pEdge);    // added to object list
                pEdge = pEdge->m_Next_edge; // get next edge
        }
        pVertex = ((CVertex*)pVertex)->m_Next_vertex;
}
// clear memory
aCell = aTable;
for(i=0; i<MAX_MATRIX; i++)
{       delete *aCell;          aCell++;   }
// Update windows
```

```
        m_Vertex = NULL;
        pDoc->m_StageShown = SHOW_GRAPH;
        InvalidateRect(0);
}

////////////////////////////////////////////////////////////////////////////////////////////
void CATCSView::GetCellRect(int row, int col, LPRECT pRect)
{       if( pRect == 0)
        {       CATCSDoc *pDoc = GetDocument();
                pDoc->m_Cell.left = m_nRibbonWidth + (col * m_nCellWidth) + 1;
                pDoc->m_Cell.top =  m_nCellHeight + (row * m_nCellHeight) + 1;
                pDoc->m_Cell.right = pDoc->m_Cell.left + m_nCellWidth - 1;
                pDoc->m_Cell.bottom = pDoc->m_Cell.top + m_nCellHeight - 1;
        }
        else
        {       pRect->left = m_nRibbonWidth + (col * m_nCellWidth) + 1;
                pRect->top  = m_nCellHeight + (row * m_nCellHeight) + 1;
                pRect->right = pRect->left + m_nCellWidth - 1;
                pRect->bottom = pRect->top + m_nCellHeight - 1;
        }
}
////////////////////////////////////////////////////////////////////////////////////////////
void CATCSView::CreateHeadArrow(UINT max1, UINT max2, CEdge *pEdge, CRect aRect, CPoint aThirdPoint)
{       double x_axis, y_axis,  angle;
        x_axis = y_axis = angle = 0;
        if( (aThirdPoint.x != 0) || (aThirdPoint.y != 0) )
        {       // don't anything
        }
        y_axis = aRect.bottom - aRect.top;
        if(max1 > max2)
        {  // 0<angle<45, 45<angle<90, and angle = 45 degree
                max1 -= (m_nCellWidth/2);       max2 += (m_nCellWidth/2);
                x_axis = max1 - max2;
                angle = atan2(y_axis, x_axis); //find arctan angle radius
                angle = (angle *180)/PI;      //find angle degree

                if( (0<angle) && (angle<20) )  // 0< x <20
                {       pEdge->m_ar1.x = pEdge->m_EndPoint.x;
                        pEdge->m_ar1.y = pEdge->m_EndPoint.y-1;
                        pEdge->m_ar2.x = pEdge->m_EndPoint.x + ((m_nCellWidth/8)*3);
                        pEdge->m_ar2.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*2) - 1;
                        pEdge->m_ar3.x = pEdge->m_EndPoint.x + ((m_nCellWidth/8)*4);
```

```
                pEdge->m_ar3.y = pEdge->m_EndPoint.y -1;
        }
        else if( (20<=angle) && (angle<40)) // 20<= x <40
        {       pEdge->m_ar1.x = pEdge->m_EndPoint.x;
                pEdge->m_ar1.y = pEdge->m_EndPoint.y-1;
                pEdge->m_ar2.x = pEdge->m_EndPoint.x + ((m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*3)-1;
                pEdge->m_ar3.x = pEdge->m_EndPoint.x + ((m_nCellWidth/8)*4);
                pEdge->m_ar3.y = pEdge->m_EndPoint.y - (m_nCellWidth/8)-1;
        }
        else if( (40<=angle) && (angle<60) ) // 40<= x < 60
        {       pEdge->m_ar1.x = pEdge->m_EndPoint.x;
                pEdge->m_ar1.y = pEdge->m_EndPoint.y-1;
                pEdge->m_ar2.x = pEdge->m_EndPoint.x + ((m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*4) - 1;
                pEdge->m_ar3.x = pEdge->m_EndPoint.x + ((m_nCellWidth/8)*5);
                pEdge->m_ar3.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*2) - 1;
        }
        else if( (60<=angle) && (angle<80) )  // 60 <= x < 80
        {       pEdge->m_ar1.x = pEdge->m_EndPoint.x;
                pEdge->m_ar1.y = pEdge->m_EndPoint.y-1;
                pEdge->m_ar2.x = pEdge->m_EndPoint.x + ((m_nCellWidth/8)*2);
                pEdge->m_ar2.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*4) - 1;
                pEdge->m_ar3.x = pEdge->m_EndPoint.x + ((m_nCellWidth/8)*4);
                pEdge->m_ar3.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*2) - 1;
        }
        else  // 80 <= x < 90
        {       pEdge->m_ar1.x = pEdge->m_EndPoint.x;
                pEdge->m_ar1.y = pEdge->m_EndPoint.y-1;
                pEdge->m_ar2.x = pEdge->m_EndPoint.x + ((m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*3)-1;
                pEdge->m_ar3.x = pEdge->m_EndPoint.x;
                pEdge->m_ar3.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*4)-1;
        }
}
else if(max1 < max2)
{       // 90<angle<135, 135<angle<180, and angle = 45 degree
        max1 += (m_nCellWidth/2);       max2 -= (m_nCellWidth/2);
        x_axis = max2 - max1;
        angle = atan2(y_axis, x_axis); //find arctan angle degree
        angle = (angle *180)/PI;       //find angle degree
        if( (0<angle) && (angle<20) )  // 0< x <20
        {       pEdge->m_ar1.x = pEdge->m_EndPoint.x;
```

```
                pEdge->m_ar1.y = pEdge->m_EndPoint.y-1;
                pEdge->m_ar2.x = pEdge->m_EndPoint.x - ((m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*2) - 1;
                pEdge->m_ar3.x = pEdge->m_EndPoint.x - ((m_nCellWidth/8)*4);
                pEdge->m_ar3.y = pEdge->m_EndPoint.y -1;
        }
        else if( (20<=angle) && (angle<40)) // 20<= x <40
        {       pEdge->m_ar1.x = pEdge->m_EndPoint.x;
                pEdge->m_ar1.y = pEdge->m_EndPoint.y-1;
                pEdge->m_ar2.x = pEdge->m_EndPoint.x - ((m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*3)-1;
                pEdge->m_ar3.x = pEdge->m_EndPoint.x - ((m_nCellWidth/8)*4);
                pEdge->m_ar3.y = pEdge->m_EndPoint.y - (m_nCellWidth/8)-1;
        }
        else if( (40<=angle) && (angle<60) ) // 40<= x < 60
        {       pEdge->m_ar1.x = pEdge->m_EndPoint.x;
                pEdge->m_ar1.y = pEdge->m_EndPoint.y-1;
                pEdge->m_ar2.x = pEdge->m_EndPoint.x - ((m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*4) - 1;
                pEdge->m_ar3.x = pEdge->m_EndPoint.x - ((m_nCellWidth/8)*5);
                pEdge->m_ar3.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*2) - 1;
        }
        else if( (60<=angle) && (angle<80) )  // 60 <= x < 80
        {       pEdge->m_ar1.x = pEdge->m_EndPoint.x;
                pEdge->m_ar1.y = pEdge->m_EndPoint.y-1;
                pEdge->m_ar2.x = pEdge->m_EndPoint.x - ((m_nCellWidth/8)*2);
                pEdge->m_ar2.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*4) - 1;
                pEdge->m_ar3.x = pEdge->m_EndPoint.x - ((m_nCellWidth/8)*4);
                pEdge->m_ar3.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*2) - 1;
        }
        else  // 80 <= x < 90
        {       pEdge->m_ar1.x = pEdge->m_EndPoint.x;
                pEdge->m_ar1.y = pEdge->m_EndPoint.y-1;
                pEdge->m_ar2.x = pEdge->m_EndPoint.x - ((m_nCellWidth/8)*3);
                pEdge->m_ar2.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*3)-1;
                pEdge->m_ar3.x = pEdge->m_EndPoint.x;
                pEdge->m_ar3.y = pEdge->m_EndPoint.y - ((m_nCellWidth/8)*4)-1;
        }
}
else // max1 == max2
{       angle = 90;  // degree
        pEdge->m_ar1.x = pEdge->m_EndPoint.x;
        pEdge->m_ar1.y = pEdge->m_EndPoint.y-1;
```

```
                pEdge->m_ar2.x = pEdge->m_EndPoint.x - ((m_nCellWidth/8)*2);
                pEdge->m_ar2.y = pEdge->m_EndPoint.y - (m_nCellWidth/2)-1;
                pEdge->m_ar3.x = pEdge->m_EndPoint.x + ((m_nCellWidth/8)*2);
                pEdge->m_ar3.y = pEdge->m_EndPoint.y - (m_nCellWidth/2)-1;
        }
}
////////////////////////////////////////////////////////////////////////////////////////////
void CATCSView::OnLButtonDown(UINT nFlags, CPoint point)
{       // TODO: Add your message handler code here and/or call default
        CScrollView::OnLButtonDown(nFlags, point);
        CRect rect;             CPoint pos = point;
        CBrush brush(RGB(0,255,255));           CClientDC dc(this);
        OnPrepareDC(&dc);               dc.DPtoLP(&pos);
        int row = (pos.y - m_nCellHeight) / m_nCellHeight;   int col = (pos.x - m_nRibbonWidth) / m_nCellWidth;
        GetCellRect(row, col, &rect);
}
////////////////////////////////////////////////////////
void CATCSView::InitArrayRootedTree()
{       for(int i=0; i<MAX_MATRIX; i++)
                m_RootedTree[i] = 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////
// The OnShowTree( ) is called when the user click ShowTree button. OnShowTree( ) is used to
// convert from a DAG into an inverse pointer Binary tree.
//
void CATCSView::OnShowTree()
{       // TODO: Add your command handler code here
        CElement*  pSource = NULL;              CElement*  pDestinate = NULL;
        CElement*  pElement = NULL;             CElement*  pVertex = NULL;
        BOOL     aFlageTree = FALSE;
        CDigraph *pDigraph = NULL;
        CString msg;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        int total_vertex     = pDoc->m_TOTAL_VERTEX;
        COLORREF color_vertex = pDoc->m_Color;
        if(!pDoc->m_StatGraph) // don't create graph
        {       AfxMessageBox(_T("You must create nodes and arcs!"));
                return;
        }
        if(!pDoc->m_StateTREE)
        {       AfxMessageBox(_T("You must initialize on show Graph."));
                return;
```

```
}
pDoc->m_StateBITREE = TRUE;          pDoc->m_StateTREE = FALSE;
InitArrayRootedTree();
// connection all node on a graph
pDigraph = new CDigraph(total_vertex, color_vertex);
POSITION aPos = pDoc->GetListHeadPosition();
while(aPos)
{       pElement = pDoc->GetNext(aPos);
        if(!aFlageTree)
        {           aFlageTree = TRUE;
                    pVertex = pDigraph->Make_Graph(pElement);
        }
        else
                    pVertex = pDigraph->Add_Vertex(pVertex, pElement);
        if( ((CVertex*)pVertex)->m_Vertex_level > pDoc->m_SizeLevel)
                    pDoc->m_SizeLevel++;
}
m_Vertex = NULL;
m_TmpVertex = pVertex;
m_Vertex = pVertex;
// Connection arc between all nodes
aPos = pDoc->GetArcListHeadPosition();
while(aPos)
{       pSource = pDestinate =0;
        pElement = pDoc->GetArcNext(aPos);
        pSource   = pDigraph->Find_Vertex(pVertex,((CArc*)pElement)->m_End_vertex);
        pDestinate = pDigraph->Find_Vertex(pVertex,((CArc*)pElement)->m_Start_vertex);
        if((pSource != NULL) && (pDestinate != NULL))
                        pDigraph->Joint_Vertex(pSource, pDestinate, pElement);
}
// Now, a m_Vertex is final a graph that is connected all
//////////////////////////////////////////////////////
// To create tree by traverse diagraph
//
// This algorithms takes to convert a diagraph to a tree
BOOL Flage_Check = FALSE;
CElement  *aFather = NULL;          CElement  *xFather = NULL;
CEdge     *aEdge   = NULL;          CElement  *aPoint = NULL;
CElement  *pPoint = pVertex;
if(m_Tree)
{       delete m_Tree;       m_Tree = 0;
}
// outer loop
```

```
while(pPoint)
{
        aFather = xFather = NULL;       aEdge = NULL;       m_Tree = NULL;
        m_Tree = new CTree(pDoc->m_Color);
        aFlageTree = FALSE;             aPoint = pPoint;
        m_VertexTree = NULL;
        //inner loop
        while(aPoint)
        {       ((CVertex*)aPoint)->m_Marked = TRUE;
                m_Tree->m_SizeLevel = pDoc->m_SizeLevel;
                if(!aFlageTree)
                {   aFlageTree = TRUE;
                    m_VertexTree = m_Tree->MakeTree(aPoint); //create rooted tree and get rooted graph
                }
                else
                {       xFather = m_Tree->FindFather(m_VertexTree, aFather, aPoint);
                        if(xFather)
                                m_Tree->Addson_SortedCommu(xFather, aPoint, aEdge);
                }
                aEdge = ((CVertex*)aPoint)->m_First_edge;
                if(aEdge)
                {       aFather = aPoint;
                        if(aEdge->m_Next_edge)
                        {       pDoc->Push_StackEdge(aEdge);
                                pDoc->Push_StackFather(aFather);
                        }
                        aPoint = aEdge->m_End_point;
                }
                else
                {       if( (pDoc->Empty_StackEdge()) && (pDoc->Empty_StackFather()))
                                aPoint = NULL;
                        else
                        {       aEdge = (CEdge*)pDoc->Pop_StackEdge();
                                aFather = pDoc->Pop_StackFather();
                                aEdge = aEdge->m_Next_edge;
                                if(aEdge->m_Next_edge)
                                {       pDoc->Push_StackEdge(aEdge);
                                        pDoc->Push_StackFather(aFather);
                                }
                                aPoint =  aEdge->m_End_point;
                        }
                }
        }// while inner loop
```

```
        m_RootedTree[(((CVertex*)m_VertexTree)->m_Task_no] = 1;
        pDoc->Push_RootedTree(m_VertexTree);
        Flage_Check = FALSE;
        pPoint = pVertex;

        // this loop checks amount of trees
        while(pPoint)
        {       if( (((CVertex*)pPoint)->m_Marked) == TRUE )
                                pPoint = (((CVertex*)pPoint)->m_Next_vertex;
                else
                {       Flage_Check = TRUE;         break;
                }
        }
        if(Flage_Check == FALSE)
                pPoint = NULL;
}// outer loop
////////////////////////////////
// initialize object tree
pDoc->InitObject_Tree();
// Traverse in tree and delete all trees
while(!pDoc->Empty_StackArrayTree())
{       aPoint = pDoc->Pop_RootedTree();
        // save rooted tree
        pDoc->Push_RootedBinaryTree(aPoint);
        //if Total Vertex exeed 50 don't paint TREE
        if(pDoc->m_TOTAL_VERTEX < 50)
        {       // Re-initialize level on tree
                m_Tree->m_CurrentLevel = 0;   m_Tree->m_PrevLevel = 0;
                // assign position
                m_Tree->Assign_Position(aPoint, TREE);
                // assign edge
                m_Tree->Assign_Edge(aPoint, TREE);
                //delete a tree
                //m_Tree->Deleted_Tree(aPoint);
        }
}
// Delete CDigraph
if(pDigraph)
{       delete pDigraph;        pDigraph = 0;
}
// Update Windows
if(pDoc->m_TOTAL_VERTEX < 50)
{       pDoc->m_StageShown = SHOW_TREE;
```

```
                InvalidateRect(0);
        }
}


////////////////////////////////////////////////////////////////////////////
// The OnShowBinaryTree( ) is called when the user clicks on a BinaryTree button.
// OnShowBinaryTree( ) is used to converse an inverse pointer binary tree into an
// inverse pointer binary tree.
//
void CATCSView::OnShowBinaryTree()
{       // TODO: Add your command handler code here
        CElement *aPoint=NULL;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        if(!pDoc->m_StatGraph) // don't create graph
        {       AfxMessageBox(_T("You must create nodes and arcs!"));
                return;
        }
        if(!pDoc->m_StateBITREE)
        {       AfxMessageBox(_T("You must create tree or reinitalize Digraph again."));
                return;
        }
        if(pDoc->Empty_StackBinaryTree())
        {       AfxMessageBox(_T("You can not create Binary InTree."));
                return;
        }
        pDoc->m_StateCLUSTER = TRUE;
        pDoc->InitObject_BiTree();
        m_Tree->m_Leaf = 0;             m_Tree->m_StartGroup = 1;
        m_Tree->m_Start = 0;            m_Tree->m_Current = 0;
        m_Tree->m_ChildGroup = 0;   m_Tree->m_SizeLevel = 0;   // level size of tree

        while(!pDoc->Empty_StackBinaryTree() )
        {       aPoint = pDoc->Pop_RootedBinaryTree();
                m_Tree->MakeBinaryTree(aPoint, NULL, 0);
                pDoc->Push_RootedCluster(aPoint);
                //if Total Vertex exceed 50 nodes don't paint binarytree
                if(pDoc->m_TOTAL_VERTEX < 50)
                {       m_Tree->AssignLevel(aPoint, 0);
                        // Re-initialize level on tree
                        m_Tree->m_CurrentLevel = 0;    m_Tree->m_PrevLevel = 0;
```

```
                    m_Tree->Assign_Position(aPoint, BI_TREE);
                    m_Tree->Assign_Edge(aPoint, BI_TREE);
                }
            }//outer loop
            //delete m_Tree object
            if(m_Tree)
            {       delete m_Tree;          m_Tree = 0;
            }
            // Update Windows
            // if Total vertex exeec 50 nodes then don't paint binarytree
            if(pDoc->m_TOTAL_VERTEX < 50)
            {       pDoc->m_StageShown = SHOW_BITREE;
                    InvalidateRect(0);
            }
}
////////////////////////////////////////////////////////////////////////////////
// The OnShowCluster( ) is called when the user clicks on a ShowCluster button.
// OnShowCluster( ) is used to parition nodes on an inverse pointer binary tree into
// a several task-clusters.
//
void CATCSView::OnShowCluster()
{       // TODO: Add your command handler code here
        int i=0;
        int aCommuCost, aStartedTime, aCompletedTime;
        int aCurrent_R, aCurrent_C;     int dif_level=0;
        UINT min1, min2, max1, max2;
        CRect aRect;
        CElement *aElement;             CElement *aTask;
        CElement *aPoint;
        CEdge    *aEdge;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        if(!pDoc->m_StatGraph) // don't create graph
        {       AfxMessageBox(_T("You must create nodes and arcs!"));
                return;
        }
        if(!pDoc->m_StateCLUSTER)
        {       AfxMessageBox(_T("You must create Binary Tree Before."));
                return;
        }
        if(pDoc->Empty_StackCluster())
        {       AfxMessageBox(_T("You must create or reinitialize Digraph again."));
                return;
```

```
}
pDoc->m_StateSCHEDULE = TRUE;
CPoint aPointStart = 0;          CPoint aPointEnd = 0;
CString msg;
if(m_Cluster)
{       AfxMessageBox(_T("You must reinitailize Digraph again."));
        pDoc->m_StateCLUSTER = FALSE;
        return;
}
m_Cluster = new CCluster(pDoc->m_Color);//create cluster
if(!m_Cluster)
{       AfxMessageBox(_T("Cannot allocate Task Cluster"));
        return;
}
m_Cluster->Put_TotalTask(pDoc->m_TOTAL_VERTEX);
m_Cluster->Allocate_SetOfTask();
aElement = m_Vertex; //m_Vertex from Digraph
while(aElement)
{       ((m_Cluster->m_SetTask)+i)->m_cost = ((CVertex*)aElement)->m_Execute_cost;
        ((m_Cluster->m_SetTask)+i)->m_number =   ((CVertex*)aElement)->m_Task_no;
        aElement = ((CVertex*)aElement)->m_Next_vertex;
        i++;
}
while(!pDoc->Empty_StackCluster())
{       aElement = pDoc->Pop_RootedCluster();
        m_Cluster->AdaptiveTaskCluster(aElement, aCommuCost, aStartedTime, aCompletedTime, NULL);
}
//****** make Graph task cluster ********
CSetTask *aSetCluster;

if(!m_Original_DiGraph)
{       AfxMessageBox(_T("The original Digraph does fail!"));
        return;
}
int index=0, aLevel=0;
CElement *pElement;
BOOL FlageMakeCluster;
POSITION aPos;
aCurrent_R = aCurrent_C =0;
// Initialize Object Cluster
pDoc->InitObjCluster();

// outer while loop
```

```
while(!m_Cluster->Empty_SetOfTaskCluster())
{       aSetCluster = m_Cluster->Get_SetOfTaskCluster();
        FlageMakeCluster = FALSE;
        index = m_Cluster->Get_TotalTask();
        aPos = pDoc->GetListTailPosition();
        pElement = 0;
        // Combine all vertexs
        while(aPos)
        {       pElement = pDoc->GetPrev(aPos);
                if( *(aSetCluster->m_SetCluster + (index-1)) == 1)
                {       if(!FlageMakeCluster)
                        {       FlageMakeCluster  = TRUE;
                                m_NodeCluster = m_Cluster->Make_Graph(pElement);
                        }
                        else
                                m_NodeCluster = m_Cluster->Add_Vertex(m_NodeCluster, pElement);
                }// end if
                index--;
        }// end Combine all vertexs
        // Assign new level on a task cluster
        aLevel = 0;
        aTask = m_NodeCluster;
        while(aTask)
        {       ((CVertex*)aTask)->m_Vertex_level = aLevel++;
                aTask = ((CVertex*)aTask)->m_Next_vertex;
        }
        // Connection arc between all vertexs
        CElement *pSource, *pDestinate;
        aPos = 0;
        aPos = pDoc->GetArcListHeadPosition();
        while(aPos)
        {       pSource = pDestinate = NULL;
                pElement = pDoc->GetArcNext(aPos);
                pSource   = m_Cluster->Find_Vertex(m_NodeCluster,((CArc*)pElement)->m_Start_vertex);
                pDestinate = m_Cluster->Find_Vertex(m_NodeCluster,((CArc*)pElement)->m_End_vertex);
                if((pSource != NULL) && (pDestinate != NULL))
                        m_Cluster->Joint_Vertex(pSource, pDestinate, pElement);
        }
        //Define Position of task and arc on a task cluster
        if(pDoc->m_TOTAL_VERTEX <50)
        {       aCurrent_R = 3;       aCurrent_C += 4;      aRect = 0;
                aPoint = m_NodeCluster;
                while(aPoint)
```

```
{       GetCellRect(aCurrent_R, aCurrent_C, &aRect);
        ((CVertex*)aPoint)->m_Rect = aRect;
        ((CVertex*)aPoint)->SetBoundRect(aRect);
        pDoc->AddObjCluster(aPoint);  //save object cluster
        aPoint = ((CVertex*)aPoint)->m_Next_vertex;
        aCurrent_R += 3;
}
aPoint = m_NodeCluster;
while(aPoint)
{       aEdge = ((CVertex*)aPoint)->m_First_edge;
        while(aEdge)
        {  aPointStart.x = ((CVertex*)aPoint)->m_Rect.right - (m_nCellWidth/2);
           aPointStart.y = ((CVertex*)aPoint)->m_Rect.bottom;
           aPointEnd.x=((aEdge->m_End_point)->m_Rect.left + (m_nCellWidth/2)) -1;
           aPointEnd.y = (aEdge->m_End_point)->m_Rect.top;
           aEdge->m_StartPoint = aPointStart;
           aEdge->m_EndPoint   = aPointEnd;
           dif_level = (aEdge->m_End_point)->m_Vertex_level –
                              ((CVertex*)aPoint)->m_Vertex_level;
           if(dif_level > 1)
           { // draw crave line
                 aEdge->m_STATE_POLY = TRUE;
                 aEdge->m_POLY1 = aPointStart;
                 aEdge->m_POLY2.x = ((CVertex*)aPoint)->m_Rect.left -
                                          m_nCellWidth;
                 aEdge->m_POLY2.y = aPointEnd.y - ((aPointEnd.y - aPointStart.y)/2);
                 aEdge->m_POLY3 = aPointEnd;
                 aRect.left   = (aEdge->m_End_point)->m_Rect.left + m_nCellWidth;
                 aRect.right  = (((CVertex*)aPoint)->m_Rect.right - (m_nCellWidth/2);
                 aRect.top    = (((CVertex*)aPoint)->m_Rect.bottom;
                 aRect.bottom = (aEdge->m_End_point)->m_Rect.top;
                 max1 = aRect.right;
                 max2 = aRect.left + (m_nCellWidth/2);
                 // CreateHeadArrow
                 CreateHeadArrow(max1, max2, aEdge, aRect);
           }
           else
           { // draw straight line
                 min1 = min2 = max1 = max2 = 0;
                 min1 = min( ((CVertex*)aPoint)->m_Rect.right,
                                          ((CVertex*)aPoint)->m_Rect.left );
                 min2 = min( (aEdge->m_End_point)->m_Rect.right,
                                          (aEdge->m_End_point)->m_Rect.left);
```

```
                                        max1 = max( ((CVertex*)aPoint)->m_Rect.right,
                                                    ((CVertex*)aPoint)->m_Rect.left);
                                        max2 = max( (aEdge->m_End_point)->m_Rect.right,
                                                    (aEdge->m_End_point)->m_Rect.left);
                                        aRect.left = min(min1, min2);
                                        aRect.right = max(max1, max2);
                                        aRect.top = ((CVertex*)aPoint)->m_Rect.bottom;
                                        aRect.bottom = (aEdge->m_End_point)->m_Rect.top;
                                        // create head arrow of each edge
                                        CreateHeadArrow(max1, max2, aEdge, aRect);
                                    }
                                    aEdge->SetBoundRect(aRect);
                                    pDoc->AddObjCluster(aEdge);    // added to object list
                                    aEdge = aEdge->m_Next_edge;
                                }//while aEdge
                                aPoint = ((CVertex*)aPoint)->m_Next_vertex;
                            }//while aPoint
                        }//if pDoc->m_TOTAL_VERTEX < 50
                        //Save Each Task Cluster on to the stack
                        pDoc->Push_RootedCluster(m_NodeCluster);
                }// end outer while loop
                // Update View Windows
                if(pDoc->m_TOTAL_VERTEX < 50)
                {       pDoc->m_StageShown = SHOW_CLUSTER;
                        InvalidateRect(0);
                }
}
////////////////////////////////////////////////////////////////////////////////////////////////
// The OnShowSchedule( ) is called when the user clicks on a ShowSchedule button.
// OnShowSchedule( ) is used to show a set of task-cluster that is used to schedule on
// processors.
//
void CATCSView::OnShowSchedule()
{       // TODO: Add your command handler code here
        int   aCurrent_R, aCurrent_C;          int   dif_level=0;
        int   aTotalCluster;
        CRect  aRect;
        UINT   min1, min2, max1, max2;
        CPoint aPointStart = 0;        CPoint aPointEnd = 0;
        CEdge *aEdge=0;                aTotalCluster = 0;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        if(!pDoc->m_StatGraph) // don't create graph
```

```
{       AfxMessageBox(_T("You must create nodes and arcs!"));
        return;
}
if(!pDoc->m_StateSCHEDULE)
{       AfxMessageBox(_T("You must create Cluster before."));
        return;
}
pDoc->m_StateBITREE  = FALSE;      pDoc->m_StateTREE    = FALSE;
pDoc->m_StateCLUSTER = FALSE;      pDoc->m_StateSCHEDULE = FALSE;

// Initialize object a set of task cluster
pDoc->Init_aSetOfTaskCluster();          aCurrent_R = aCurrent_C = 0;
if(!m_Original_DiGraph)
{       AfxMessageBox(_T("Please intialize Digraph again!"));
        return;
}
if(!InitMatrixDigraph(m_Original_DiGraph))
        AfxMessageBox(_T("Can not create matrix Digraph."));
if(pDoc->Empty_StackCluster())
{       AfxMessageBox(_T("You can not create schedule for this cluster."));
        return;
}
CString msg;
BOOL FlageMake = FALSE;   BOOL FlageSave = FALSE;
CElement *pSink, *pNode;     CElement *pPrev, *aPoint;
CNodeBi  *pRooted=0;

while(!pDoc->Empty_StackCluster())
{       pSink = pNode = pPrev = aPoint = 0;        FlageSave = FALSE;
        pNode = pDoc->Pop_RootedCluster();     pPrev = pNode;
        while(pPrev)
        {       pSink = pPrev;
                pPrev = ((CVertex*)pPrev)->m_Next_vertex;
        }
        // Insert task-cluster onto a set of task-clusters
        // if a task-cluster duplicates on a set of tack-clusters.
        // It don't insert onto a set of task-clusters.
        if(!FlageMake)
        {       FlageMake = TRUE;
                pRooted = m_Cluster->MakeBiTree( ((CVertex*)pSink)->m_Task_no );
                aPoint = pNode;
                pDoc->Push_RootedSchedule(pNode);
                aTotalCluster++;
```

```
            FlageSave = TRUE;
        }
        else
        {       if(m_Cluster->InsertNodeOnBiTree(pRooted, ((CVertex*)pSink)->m_Task_no))
                {       aPoint = pNode;
                        pDoc->Push_RootedSchedule(pNode);
                        aTotalCluster++;
                        FlageSave = TRUE;
                }
        }
        if((FlageSave) && (pDoc->m_TOTAL_VERTEX<50))
        {       // to define positon to paint on the view
                aCurrent_R = 3;         aCurrent_C += 4;
                aRect = 0;
                while(aPoint)
                {       GetCellRect(aCurrent_R, aCurrent_C, &aRect);
                        ((CVertex*)aPoint)->m_Rect = aRect;
                        ((CVertex*)aPoint)->SetBoundRect(aRect);
                        pDoc->Add_aSetOfTaskCluster(aPoint);  //save object cluster
                        aPoint = ((CVertex*)aPoint)->m_Next_vertex;
                        aCurrent_R += 3;
                }
                aPoint = pNode;
                while(aPoint)
                {       aEdge = ((CVertex*)aPoint)->m_First_edge;
                        while(aEdge)
                        {  aPointStart.x = ((CVertex*)aPoint)->m_Rect.right - (m_nCellWidth/2);
                           aPointStart.y = ((CVertex*)aPoint)->m_Rect.bottom;
                           aPointEnd.x = ((aEdge->m_End_point)->m_Rect.left + (m_nCellWidth/2)) -1;
                           aPointEnd.y = (aEdge->m_End_point)->m_Rect.top;
                           aEdge->m_StartPoint = aPointStart;
                           aEdge->m_EndPoint   = aPointEnd;
                           dif_level = (aEdge->m_End_point)->m_Vertex_level -
                                                        ((CVertex*)aPoint)->m_Vertex_level;

                           if(dif_level > 1)
                           { // draw crave line
                             aEdge->m_STATE_POLY = TRUE;
                             aEdge->m_POLY1 = aPointStart;
                             aEdge->m_POLY2.x = ((CVertex*)aPoint)->m_Rect.left -  m_nCellWidth;
                             aEdge->m_POLY2.y = aPointEnd.y - ((aPointEnd.y - aPointStart.y)/2);
                             aEdge->m_POLY3 = aPointEnd;
                             aRect.left   = (aEdge->m_End_point)->m_Rect.left + m_nCellWidth;
```

```
                    aRect.right  = ((CVertex*)aPoint)->m_Rect.right - (m_nCellWidth/2);
                    aRect.top    = ((CVertex*)aPoint)->m_Rect.bottom;
                    aRect.bottom = (aEdge->m_End_point)->m_Rect.top;
                    max1 = aRect.right;
                    max2 = aRect.left + (m_nCellWidth/2);
                    // CreateHeadArrow
                    CreateHeadArrow(max1, max2, aEdge, aRect);
                  }
                  else
                  { // draw straight line
                    min1 = min2 = max1 = max2 = 0;
                    min1 = min( ((CVertex*)aPoint)->m_Rect.right,
                                                ((CVertex*)aPoint)->m_Rect.left );
                    min2 = min( (aEdge->m_End_point)->m_Rect.right,
                                                (aEdge->m_End_point)->m_Rect.left);

                    max1 = max( ((CVertex*)aPoint)->m_Rect.right,
                                                ((CVertex*)aPoint)->m_Rect.left);
                    max2 = max( (aEdge->m_End_point)->m_Rect.right,
                                                (aEdge->m_End_point)->m_Rect.left);
                    aRect.left = min(min1, min2);
                    aRect.right = max(max1, max2);
                    aRect.top = ((CVertex*)aPoint)->m_Rect.bottom;
                    aRect.bottom = (aEdge->m_End_point)->m_Rect.top;
                    // create head arrow of each edge
                    CreateHeadArrow(max1, max2, aEdge, aRect);
                  }
                  aEdge->SetBoundRect(aRect);
                  pDoc->Add_aSetOfTaskCluster(aEdge);    // added to object list
                  aEdge = aEdge->m_Next_edge;
                  }// end while edge
                  aPoint = ((CVertex*)aPoint)->m_Next_vertex;
              }// end while point
        }// if statmente FlageSave

}// end while pop up stack cluster
// Computed Makespan
CSingleNode *aNode= new CSingleNode[aTotalCluster];

for(int i=0; i<aTotalCluster; i++)
{       (aNode+i)->m_number = 0;       (aNode+i)->m_cost = 0;
        (aNode+i)->m_started = 0;      (aNode+i)->m_finished = 0;
}
```

```
        m_Cluster->GetSinkNode_OnBiTree(pRooted, aNode, aTotalCluster);
        m_nScheduleTime  = Get_MakeSpan(aNode, aTotalCluster);
        m_nMakeSpan      = Get_Makespan_DAG();
        m_nSpeedUp       = 0.0;
        m_nSpeedUp       = (double)(m_nSchedule_Single_P)/(double)(m_nScheduleTime);
        m_nFinalCluster  = 0;
        m_nFinalCluster  = aTotalCluster;
        delete [ ] aNode;

        // Update View Windows
        if(pDoc->m_TOTAL_VERTEX < 50)
        {       pDoc->m_StageShown = SHOW_ATCS;
                InvalidateRect(0);
        }
        CElement *pPoint;           CEdge   *pEdge;     aEdge = NULL;
        CElement *pGraph = m_Vertex;
        while(pGraph)
        {       pPoint = pGraph;
                pEdge = ((CVertex*)pPoint)->m_First_edge;
                while(pEdge)
                {       aEdge = pEdge;
                        pEdge = pEdge->m_Next_edge;
                        delete aEdge;       aEdge = 0;
                }
                pGraph = ((CVertex*)pGraph)->m_Next_vertex;
                delete pPoint;      pPoint = NULL;
        }
        m_Vertex = NULL;            m_TmpVertex = NULL;
}
////////////////////////////////////////////////////////////////////////////////
// Computed MakeSpan on a DAG
int CATCSView::Get_MakeSpan(CSingleNode *aArrayNode, int aSize)
{       POSITION aPos, aIndexStack;
        CElement *pNode = NULL;   CElement *pPrev = NULL;
        CElement *pCurrent = NULL;   CElement *pSink = NULL;
        CLinkList *pLinkList = NULL;          CLinkList *pTempLinkList = NULL;
        CElement *pArc = NULL;
        int pStart, pEnd;    int makespan = 0;    int aStarted_Time = 0;
        int aFinished_Time = 0;      int Time_Finished_Outside =0;   int Schedule_Time = 0;
        int j=0;
        CSingleNode  *a_Set_Of_Source;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
```

```
CString msg;
// Get a set of source node on a DAG
// Source node is set to TRUE
// Sink node is set to FALSE
int index;
a_Set_Of_Source = new CSingleNode[pDoc->m_TOTAL_VERTEX];
for(j=0; j< pDoc->m_TOTAL_VERTEX; j++)
{       (a_Set_Of_Source+j)->m_number = j+1;
        (a_Set_Of_Source+j)->m_Source_Node = TRUE;
}
aIndexStack   = pDoc->GetArcListHeadPosition();
while(aIndexStack)
{       pNode = pDoc->GetArcNext(aIndexStack);
        index = ((CArc*)pNode)->m_End_vertex;
        (a_Set_Of_Source + (index-1))->m_Source_Node = FALSE;
}
CElement * aArray_Cluster;              pStart = pEnd = 0;
for(int i=0; i < aSize; i++)
{       aPos = pDoc->GetTailSchedule();
        while(aPos)
        {       pNode = pDoc->GetPrevSchedule(aPos);
                pPrev = pNode;
                pStart = ((CVertex*)pNode)->m_Task_no;
                while(pPrev)
                {       pSink = pPrev;
                        pPrev = ((CVertex*)pPrev)->m_Next_vertex;
                }
                pEnd = ((CVertex*)pSink)->m_Task_no;
                if( ((CVertex*)pSink)->m_Task_no == (aArrayNode+i)->m_number )
                        break;
        }
        /// continue
        aArray_Cluster = pNode;     // a cluster
        pCurrent = pNode;           // a cluster

        aStarted_Time = 0;          aFinished_Time = 0;         Time_Finished_Outside = 0;
        while(pCurrent) //every node in a cluster
        {       if(Check_Source_Node(a_Set_Of_Source, pDoc->m_TOTAL_VERTEX, pCurrent))
                {       aFinished_Time = aStarted_Time + ((CVertex*)pCurrent)->m_Execute_cost;
                        aStarted_Time = aFinished_Time;
                }
                else
                {       //Get a set of predecessor nodes of current node on a cluster
```

```
                pLinkList = Read_Source_Node(pLinkList, pCurrent);
                pTempLinkList = pLinkList;
                while(pTempLinkList)
                { if(Check_Is_On_Cluster(aArray_Cluster, pTempLinkList))
                        aStarted_Time = aStarted_Time:
                  else // is outside on cluster
                        Time_Finished_Outside =
                            Get_Finished_Time_Of_Outside_Cluster(aArrayNode,aSize,
                                                        pTempLinkList->info) +
                            Communication_Time_Of_Node_Outside_Cluster(
                                                    pTempLinkList->info,
                                                    ((CVertex*)pCurrent)->m_Task_no);

                  if(Time_Finished_Outside > aStarted_Time)
                        aStarted_Time = Time_Finished_Outside;
                   // get next a predecedance node
                        pTempLinkList  = pTempLinkList->pNextNode;
                  }// while

                  //define aFinished_Time and aStarted_Time
                  aFinished_Time = aStarted_Time +
                                ((CVertex*)pCurrent)->m_Execute_cost;
                  aStarted_Time = aFinished_Time;
                }// else if
                pCurrent = ((CVertex*)pCurrent)->m_Next_vertex; //get next node in a cluster
                if(pLinkList)
                {       m_Cluster->FreeLinkList(pLinkList);
                        pLinkList = NULL;
                }

                if( aFinished_Time > Schedule_Time)
                        Schedule_Time = aFinished_Time;
          }// while for pCurrent

          Insert_Finished_Time_Onto_Cluster(aArrayNode, aSize, pEnd, aFinished_Time);

    }// for loop
    delete [ ] a_Set_Of_Source;
    return Schedule_Time;
}
//////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CATCSView::Insert_Finished_Time_Onto_Cluster(CSingleNode *aArraySinkNode, int aSize, int pSinkNode,
                                                int aFinished_Time)
```

```
{       for(int index = 0; index < aSize; index++)
        {       if((aArraySinkNode +index)->m_number == pSinkNode)
                {       (aArraySinkNode+index)->m_finished = aFinished_Time;
                        return TRUE;
                }
        }
        return FALSE;
}
////////////////////////////////////////////////////////////////////////////////////////////////
int CATCSView::Get_Finished_Time_Of_Outside_Cluster(CSingleNode *pSinkNode, int aSize, int pPredecessor)
{       for(int index=0; index < aSize; index++)
        {       if( (pSinkNode+index)->m_number == pPredecessor)
                        return ((pSinkNode+index)->m_finished);
        }
        return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////
int CATCSView::Communication_Time_Of_Node_Outside_Cluster(int pSource, int pDestinate)
{       POSITION aPos;
        CElement *pArc = NULL;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        aPos  = pDoc->GetArcListHeadPosition();
        while(aPos)
        {       pArc = pDoc->GetArcNext(aPos);
                if( ( ((CArc*)pArc)->m_Start_vertex  ==  pSource) &&
                        ( ((CArc*)pArc)->m_End_vertex == pDestinate)  )
                        return ((CArc*)pArc)->m_Commu_cost;
        }
        return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CATCSView::Check_Is_On_Cluster(CElement *pCluster, CLinkList *pNode)
{       while(pCluster)
        {       if( pNode->info == ((CVertex*)pCluster)->m_Task_no )
                        return TRUE;
                pCluster = ((CVertex*)pCluster)->m_Next_vertex;
        }
        return FALSE;
}
////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CATCSView::Check_Source_Node(CSingleNode *pArray_Source, int aSize, CElement *pNode)
{       CString msg;
```

```
        for(int i=0; i< aSize; i++)
        {       if( ((pArray_Source+i)->m_number ==  ((CVertex*)pNode)->m_Task_no ) &&
                        ( (pArray_Source+i)->m_Source_Node == TRUE ) )
                        return TRUE;
        }
        return FALSE;
}
////////////////////////////////////////////////////////////////////////////////////////////////////
CLinkList *CATCSView::Read_Source_Node(CLinkList *pLinkList, CElement *pNode)
{       POSITION aPos;
        CElement *pArc = NULL;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        aPos    = pDoc->GetArcListHeadPosition();
        while(aPos)
        {       pArc = pDoc->GetArcNext(aPos);
                if( ((CArc*)pArc)->m_End_vertex  == ((CVertex*)pNode)->m_Task_no )
                        pLinkList = m_Cluster->InsertToLinkList(pLinkList, ((CArc*)pArc)->m_Start_vertex);
        }
        return pLinkList;
}
////////////////////////////////////////////////////////////////////////////////////////////////////
int CATCSView::Get_Makespan_DAG()
{       int MakeSpan, Grand_MakeSpan;
        CString msg;
        CElement *aPoint = NULL;
        CElement *pPoint = NULL;
        CElement *pFather = NULL;
        CEdge *pEdge = NULL;
        int *pLinkMakeSpan;
        // Get Class Document
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        // Allocate and Initialize
        pLinkMakeSpan = new int[100];
        for(int i=0; i< 100; i++)
                pLinkMakeSpan[i] = 0;
        BOOL Flage_Check = FALSE;
        MakeSpan = Grand_MakeSpan = 0;
        pPoint = m_Original_DiGraph;
        // initialize marked for all nodes on a DAG
        aPoint = pPoint;
        while(aPoint)
```

```
{       ((CVertex*)aPoint)->m_Marked = FALSE;
        aPoint = ((CVertex*)aPoint)->m_Next_vertex;
}
while(pPoint)// Outer while loop
{       pFather = NULL;     pEdge  = NULL;     aPoint = pPoint;
        while(aPoint) // inner while loop
        {       ((CVertex*)aPoint)->m_Marked = TRUE;
                // To Compute makespan
                MakeSpan += ((CVertex*)aPoint)->m_Execute_cost;
                pEdge = ((CVertex*)aPoint)->m_First_edge;
                if(pEdge)
                {       pFather = aPoint;
                        if(pEdge->m_Next_edge)
                        {       pDoc->Push_StackEdge(pEdge);
                                pDoc->Push_StackFather(pFather);
                                // push makespan
                                m_Cluster->PushMakeSpan(pLinkMakeSpan, MakeSpan);
                        }
                        // To Compute makespan
                        MakeSpan += pEdge->m_Commu_cost;
                        aPoint = pEdge->m_End_point;
                }
                else
                {
                if(MakeSpan > Grand_MakeSpan)
                                Grand_MakeSpan = MakeSpan;
                if( (pDoc->Empty_StackEdge()) && (pDoc->Empty_StackFather()) )
                                aPoint = NULL;
                        else
                        {       pEdge = (CEdge*)pDoc->Pop_StackEdge();
                                pFather = pDoc->Pop_StackFather();
                                // Pop makespan
                                MakeSpan = m_Cluster->PopMakeSpan(pLinkMakeSpan);
                                pEdge = pEdge->m_Next_edge;
                                if(pEdge->m_Next_edge)
                                {       pDoc->Push_StackEdge(pEdge);
                                        pDoc->Push_StackFather(pFather);
                                        // push makespan
                                        m_Cluster->PushMakeSpan(pLinkMakeSpan, MakeSpan);
                                }
                                // To Compute makespan
                                MakeSpan += pEdge->m_Commu_cost;
                                aPoint = pEdge->m_End_point;
```

```
                        }
                    }
                }// inner while loop
                pPoint = m_Original_DiGraph;            Flage_Check = FALSE;
                while(pPoint)
                {       if( ((CVertex*)pPoint)->m_Marked == TRUE )
                        {       pPoint = ((CVertex*)pPoint)->m_Next_vertex;
                                MakeSpan = 0;
                        }
                        else
                        {       Flage_Check = TRUE;
                                break;
                        }
                }
                if(Flage_Check == FALSE)
                        pPoint = NULL;
            }// Outer while loop

            //******* To find executeion time on a single processor **********
            m_nSchedule_Single_P = 0;
            pPoint = m_Original_DiGraph;
            while(pPoint)
            {
                m_nSchedule_Single_P += ((CVertex*)pPoint)->m_Execute_cost;
                pPoint = ((CVertex*)pPoint)->m_Next_vertex;
            }
            delete [ ] pLinkMakeSpan;
            return Grand_MakeSpan;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CATCSView::InitMatrixDigraph(CElement *pElement, int pMatrix[ ][MAX_MATRIX])
{       CElement *aPoint;  CEdge   *aEdge;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        if(!pElement)
                return FALSE;
        for(int i=0; i<MAX_MATRIX; i++)
                for(int j=0;j<MAX_MATRIX; j++)
                        pMatrix[i][j] =  0;
        int aPos=0;         aPoint = pElement;
        while(aPoint)
        {       aPos = ((CVertex*)aPoint)->m_Task_no;
                aEdge = ((CVertex*)aPoint)->m_First_edge;
```

```
                while(aEdge)
                {       pMatrix[aPos][(((CVertex*)(aEdge->m_End_point))->m_Task_no] = 1;
                        aEdge = aEdge->m_Next_edge;
                }
                aPoint = ((CVertex*)aPoint)->m_Next_vertex;
        }
        return TRUE;
}
////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CATCSView::InitMatrixDigraph(CElement *pElement)
{       CElement *aPoint;
        CEdge    *aEdge;
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        if(!pElement)
                return FALSE;
        for(int i=0; i<MAX_MATRIX; i++)
                for(int j=0;j<MAX_MATRIX; j++)
                        m_MatrixDigraph[i][j] = 0;
        int aPos=0;     aPoint = pElement;
        while(aPoint)
        {       Pos = ((CVertex*)aPoint)->m_Task_no;
                aEdge = ((CVertex*)aPoint)->m_First_edge;
                while(aEdge)
                {       m_MatrixDigraph[aPos][(((CVertex*)(aEdge->m_End_point))->m_Task_no] = 1;
                        aEdge = aEdge->m_Next_edge;
                }
                aPoint = ((CVertex*)aPoint)->m_Next_vertex;
        }
        return TRUE;
}
////////////////////////////////////////////////////////////////////////////////////////////////
//
void CATCSView::OnUpdateSingleProcessor(CCmdUI* pCmdUI)
{
        CString msg;
        msg.Format(_T("Time on 1 Processor %d"), m_nSchedule_Single_P);

        pCmdUI->Enable(TRUE);
        pCmdUI->SetText(msg);
}
```

```
////////////////////////////////////////////////////////////////////////////////
void CATCSView::OnUpdateMakespan(CCmdUI* pCmdUI)
{
        CString string;
        string.Format(_T("Makespan %d"), m_nMakeSpan);

        pCmdUI->Enable(TRUE);
        pCmdUI->SetText(string);
}

////////////////////////////////////////////////////////////////////////////////
//
void CATCSView::OnUpdateSchedule(CCmdUI* pCmdUI)
{
        CString string;
        string.Format(_T("ATCS Time %d"), m_nScheduleTime);

        pCmdUI->Enable(TRUE);
        pCmdUI->SetText(string);
}

////////////////////////////////////////////////////////////////////////////////
//
void CATCSView::OnUpdateSpeedup(CCmdUI* pCmdUI)
{
        CString string;

        string.Format(_T("Speed Up %04f"), m_nSpeedUp);

        pCmdUI->Enable(TRUE);
        pCmdUI->SetText(string);
}

////////////////////////////////////////////////////////////////////////////////
//
void CATCSView::OnUpdateSetCluster(CCmdUI* pCmdUI)
{
        CString msg;
        msg.Format(_T("Set of Clusters %d"), m_nFinalCluster );
        pCmdUI->Enable(TRUE);
        pCmdUI->SetText(msg);
}
```

```
////////////////////////////////////////////////////////////////////////////
void CATCSView::OnRandomdag()
{       CString msg;
        CElement *pElement;
        int aRange_Execute=0;
        int aRange_MaxExecute = 0;
        int aRange_Communicate = 0;
        int aRange_Depth = 0;
        int aRange_Arc  = 0;
        BOOL aFine_Grain = FALSE;
        BOOL aCoarse_Grain = FALSE;
        int random_amount_node;
        int random_execute_node;
        int random_arc_node;
        int random_destinate_node;
        int random_communicate;
        int *array_old_node;
        int *pPoint_x = NULL;
        int amount_node=0;
        int Total_Arc=0;
        // TODO: Add your command handler code here
        CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        //Open Dialog "Generate Random DAG Dialog Box
        CGeneratRandomDAG aDlg;
        // Initialize with minimun value for each item
        aDlg.m_RANGE_EXECUTE = 1;
        aDlg.m_RANGE_MAXEXECUTE = 2;
        aDlg.m_RANGE_DEPTH = 3;
        aDlg.m_RANGE_ARC = 1;
        aDlg.m_FINE_GRAIN  = TRUE;
        aDlg.m_COARSE_GRAIN = FALSE;

        if(pDoc->m_StatGraph == FALSE)
        {       if( aDlg.DoModal() == IDOK)
                {       pDoc->m_StatGraph = TRUE:
                        aRange_Execute    = aDlg.m_RANGE_EXECUTE;
                        aRange_MaxExecute = aDlg.m_RANGE_MAXEXECUTE;
                        aRange_Depth      = aDlg.m_RANGE_DEPTH;
                        aRange_Arc        = aDlg.m_RANGE_ARC;
                        aFine_Grain       = aDlg.m_FINE_GRAIN;
                        aCoarse_Grain     = aDlg.m_COARSE_GRAIN;
```

```
if(aRange_MaxExecute <= aRange_Execute)
{       AfxMessageBox(_T("Max Execute must be exceed Min Execute."));
        return;
}
if( (aFine_Grain == FALSE) && (aCoarse_Grain == FALSE))
{       AfxMessageBox(_T("You must select Fine or Coarse Grain."));
        return;
}
if( (aFine_Grain == TRUE) && (aCoarse_Grain == TRUE))
{       AfxMessageBox(_T("You must select Fine or Coarse Grain."));
        return;
}
if( aRange_Arc >= aRange_Depth)
{       AfxMessageBox(_T("A range Depth must exceed a range Arc."));
        return;
}

int aTaskNo=0;
for(int i=0; i<= (aRange_Depth); i++)
{       random_amount_node = 0; //random amount node on each level
        while( (random_amount_node <= (aRange_Depth - 2 )) ||
                        (random_amount_node>(aRange_Depth + 1)) )
                random_amount_node = rand()%255;
        for(int j=0; j<random_amount_node ; j++)
        {       random_execute_node = 0; //random execute node
                while( (random_execute_node < aRange_Execute) ||
                        (random_execute_node > aRange_MaxExecute) )
                        random_execute_node = rand()%255;
                m_pTempNode = Create_Node(j+aTaskNo+1, random_execute_node, i);
                pDoc->AddNode(m_pTempNode);
                m_pTempNode = 0;
        }
        aTaskNo += random_amount_node;
        amount_node += random_amount_node;
}// for loop
pDoc->m_TOTAL_VERTEX = amount_node;

// generate communicate cost arcs for each node
POSITION aPos = pDoc->GetListHeadPosition();
while(aPos)
{       pElement = pDoc->GetNext(aPos);
        if( ((CNode*)pElement)->m_Vertex_level < aRange_Depth)
```

```
{       random_arc_node = 0;
        while( (random_arc_node < (2)) ||
                (random_arc_node > aRange_Arc) )
                random_arc_node = rand()%255;
        // loop for a number of arcs per each node
        array_old_node = new int[random_arc_node];
        for(int i=0; i<random_arc_node; i++)
                array_old_node[i] = 0;
        for(int k=0; k<random_arc_node; k++)
        {   random_destinate_node =0;
            pPoint_x = 0;
            while( (random_destinate_node < 1) ||
                (random_destinate_node > aTaskNo) ||
                (!Difference_Level_DAG(random_destinate_node,
                        ((CNode*)pElement)->m_Vertex_level)) ||
                (Check_Duplicate_Node(array_old_node,
                        random_destinate_node, random_arc_node)) )
                {
                        random_destinate_node = rand()%255;
                }
                random_communicate=
                        Get_Random_Communicate(aRange_Execute,
                                aRange_MaxExecute,
                                ((CNode*)pElement)->m_Execute_cost,
                                Get_ExecuteCost(random_destinate_node),
                                aFine_Grain, aCoarse_Grain);
                array_old_node[k] = random_destinate_node;
                m_pTempArc=
                        Create_Arc(((CNode*)pElement)->m_Task_no,
                                random_destinate_node,
                                random_communicate);
                pDoc->AddArc(m_pTempArc);
                m_pTempArc = 0;
        }// for
        delete [ ] array_old_node;
        Total_Arc += random_arc_node;
    }//if
}//while(aPos)

pDoc->m_TOTAL_ARC = Total_Arc; // Assume
}//if aDlg.DoModal() == IDOK
}// if( pDoc->m_StatGraph)
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CATCSView::Check_Duplicate_Node(int *pArrayNode, int aDestinate_Node, int aSize)
{       for(int i=0; i<aSize; i++)
        {       if( *(pArrayNode+i) == aDestinate_Node)
                        return TRUE;
        }
        return FALSE; // not duplicate
}

////////////////////////////////////////////////////////////////////////////////////////////////////
BOOL CATCSView::Difference_Level_DAG(int aDestinate_TaskNo, int aCurrentLevel)
{       CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        CElement *pElement;
        POSITION aPos = pDoc->GetListHeadPosition();
        while(aPos)
        {       pElement = pDoc->GetNext(aPos);
                if( ((CNode*)pElement)->m_Task_no == aDestinate_TaskNo )
                {       if( ((CNode*)pElement)->m_Vertex_level > aCurrentLevel )
                        {       if( ( ((CNode*)pElement)->m_Vertex_level - aCurrentLevel ) > 2)
                                        return FALSE;
                                else
                                        return TRUE;
                        }
                        else
                                return FALSE;
                }
        }
        return FALSE;
}
////////////////////////////////////////////////////////////////////////////////////////////////////
// To Create Communicate arc of a random DAG
int CATCSView::Get_Random_Communicate(int aRange_MinExecute, int aRange_MaxExecute, int aSource_Execute,
                                        int aDestinate_Execute, BOOL aFine_Grain, BOOL aCoarse_Grain)
{       int rand_commu = 0;
        if( aCoarse_Grain == TRUE) // for a coarse grain
        {       rand_commu = 0;
                if(aSource_Execute > aDestinate_Execute)
                {       while( (rand_commu > aDestinate_Execute) || (rand_commu <= (aDestinate_Execute/2)) )
                                rand_commu = rand() % 255;
                }
                else if(aSource_Execute < aDestinate_Execute)
                {       while( (rand_commu > (aSource_Execute) ) ||
```

```
                                        (rand_commu <= (aSource_Execute/2)) )
                                rand_commu = rand() % 255;
                }
                else // aSource_Execute == aDestinate_Execute
                        rand_commu = aSource_Execute;
                return rand_commu;
        }
        else // for a fine grain task
        {       rand_commu = 0;
                int aFine_Grain = 0;
                // Range random a Granularity is between 6 to 10
                while( (aFine_Grain < 6 ) || (aFine_Grain > 10))
                        aFine_Grain = rand() % 255;
                while( (rand_commu <= aSource_Execute * 4) ||
                        (rand_commu > (aRange_MinExecute * aFine_Grain) ) )
                        rand_commu = rand()%255;
                return rand_commu;
        }
}
////////////////////////////////////////////////////////////////////////////////////////////////////
int CATCSView::Get_ExecuteCost(int aTask_No)
{       CATCSDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        CElement *pElement;
        int Execute_Cost=0;
        POSITION aPos = pDoc->GetListHeadPosition();
        while(aPos)
        {       pElement = pDoc->GetNext(aPos);
                if( ((CNode*)pElement)->m_Task_no == aTask_No)
                {       Execute_Cost = ((CNode*)pElement)->m_Execute_cost;
                        break;
                }
        }
        return Execute_Cost;
}

////////////////////////////////////////////////////////////////////////////////////////////////////  END  ////////////////////////////////////////////////////////////////////////////////////////////////////
```

ภาคผนวก ข.
ผลงานวิจัยที่ได้รับการตีพิมพ์

1. Kantakajorn S., Piyatamrong B. "A Heuristic of Task Clustering Algorithm(HTCA) on Parallel Program." Lakrabang Engineering Journal, vol. 16, no. 3, September 1999. pp. 95-100.
2. Piyatamrong B., Ohara S. and Kantakajorn S. "Adaptive Task Clustering and Scheduling Algorithms for Parallel Processor Architecture." Proceedings 1999 IEEE International Symposium on Intelligent Signal Processing and Communication Systems(ISPACS'99), Phuket, Thailand, December 1999. pp. 485-488.
3. Piyatamrong B., Ohara S. and Kantakajorn S. "GTCS: A Greedy Task Clustering and Scheduling Algorithms for Distributed Memory Processors Architecture." Proceedings IEEE The Fourth International Conference/Exhibition on High Performance Computing in Asia-Pacific Region(IEEE HPC-Asia 2000), Beijing, China, May 2000. pp. 310-314.

ประวัติผู้เขียน

ชื่อผู้เขียน	นาย ศักดิ์ชัย คันธะขจร
วัน เดือน ปี เกิด	3 ธันวาคม 2515
วุฒิการศึกษาระดับปริญญาตรี	วิทยาศาสตร์บัณฑิต สาขาคณิตศาสตร์
สถานที่สำเร็จการศึกษา	มหาลัยมหิดล
ประสบการณ์ทำงาน	โปรแกรมเมอร์
	Software Engineer
	Senior Software Engineer
งานวิจัยที่สนใจ	การทำงานของตัวประมวลผลแบบขนานและแบบกระจาย

เฮียวลิสติก อัลกอริทึม การจัดกลุ่มงานบนโปรแกรมแบบขนาน

A HEURISTIC OF TASK CLUSTERING ALGORITHM (HTCA) ON PARALLEL PROGRAM

ศักดิ์ชัย คันธะขจร บรรจง ปิยธำรง

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

บทคัดย่อ

ปัญหาการแบ่งกลุ่มงาน(task clustering)บนโปรแกรมแบบขนาน(parallel program) คือปัญหาการจัดแบ่งโหนด(node)หรืองานบน directed acyclic task grap (DAG)ลงบนหลายๆโปรเซสเซอร์(multiprocessor) เพื่อลดเวลาที่ใช้ในการประมวลผลของงานทั้งระบบ ปัญหาการแบ่งกลุ่มงานถือได้ว่าเป็นปัญหาแบบ NP hard [4] ถึงแม้ว่าจะไม่มีการจำกัดจำนวนของโปร-เซสเซอร์และยอมให้เกิดงานซ้ำซ้อนกันระหว่างกลุ่มงานก็ตาม HTCAจะทำการจัดกลุ่มงานโดยการแบ่งงานหรือโหนดบน DAG HTCAจะพยายามลดโอเวอร์เฮด(overhead)ของค่าใช้จ่ายในการติดต่อสื่อสาร(communication cost)ระหว่างโปรเซส-เซอร์ โดย HTCAจะทำการแปลงลักษณะเม็ดเนื้องานแบบละเอียด(a fine grain task)เป็นลักษณะเม็ดเนื้องานแบบหยาบ(a coarse grain task) HTCAมีค่าความสลับซับซ้อน(complexity time)ของอัลกอริทึมเท่ากับ $O(|V^2|)$

Abstact

The optimization partitioning task on parallel program is a problem of partitioning the tasks on a directed acyclic task graph (DAG) onto the processors of a multiprocessor in a way that minimizes the execution time. The task clustering problem is NP hard problem [4], although a number of processor is not define and a duplicate task between clusters are allowed. The heuristic of task clustering algorithm(HTCA) implements the task clustering method by partition nodes on a DAG. The HTCA tries to reduce the overhead communication cost between processor by transforming a fine grain task to a coarse grain task. The HTCA has a complexity time of algorithm equals $O(|V^2|)$.

1) บทนำ

ลักษณะการทำงานของโปรแกรมแบบขนานมีทั้งแบบแจกแจงหน่วยความจำ(distributed memory)และแบบหน่วยความจำร่วม(shared memory) โดยงานวิจัยนี้จะเน้นสถาปัตยกรรมแบบแจกแจงหน่วยความจำเป็นหลัก สิ่งสำคัญที่เกิดขึ้นในการทำงานแบบขนานคือค่าใช้จ่ายในการติดต่อสื่อสารระหว่างโปรเซสเซอร์ ยิ่งมีค่าใช้จ่ายในการติดต่อสื่อสารระหว่างโปรเซสเซอร์มากเท่าใดการเกิดโอเวอร์-เฮดก็มากขึ้นตามมา การเกิดโอเวอร์เฮดทำให้เวลาที่ใช้ในประมวลผล(execution time)ของงานมากขึ้น ดังนั้นประสิทธิภาพในการประมวลผลของโปรแกรมแบบขนานขึ้นอยู่กับอัลกอริทึมในการแบ่งโปรแกรมออกเป็นกลุ่มๆ กลุ่มงานอาจจะเป็นกลุ่มของคำสั่ง(instruction)หรือโมดูล(module) การแบ่งกลุ่มงานทำให้เวลาในการประมวลผลน้อยลง เพราะเป็นการลดเวลาในการติดต่อระหว่างโปร-เซสเซอร์น้อยลง เทคนิคดังกล่าวทำให้เวลาที่ใช้ในการประมวลผลโดยรวมของ task granularity ของ DAG มีค่าน้อยลง โดยทั่วไปอัลกอริทึมในการจัดกลุ่มงานจะมีอยู่สองแบบ 1)แบบงานซ้ำซ้อน(task duplication) และ 2)แบบงานไม่ซ้ำซ้อน(non task duplication) โดยบทความนี้จะเน้นพิจารณาลักษณะเม็ดเนื้องานแบบละเอียดเป็นหลัก

2) ทฤษฎี

การแสดงขั้นตอนและทิศทางการทำงานของโปรแกรมแบบขนานจะใช้ weighted directed graph G = (V,E,μ,λ) หรือ weighted DAG G เมื่อ V คือเซต(set)ของโหนดหรืองานบน DAG E คือเซตของอาร์ค(arc)ระหว่างโหนดใน V ถ้ากำหนดให้ u และ v เป็นสมาชิกโหนดของ V และกำหนดให้ (u,v) $\in$ E แสดงให้เห็นว่างานของโหนดu จะเสร็จสิ้นสมบูรณ์ก่อน ต่อจากนั้นโหนดu จะต้องส่งข้อมูลมายังโหนด v จากนั้นโหนด v จึงจะสามารถเริ่มต้นทำงานได้ กำหนดให้ μ(v) เมื่อ v $\in$ V หมายถึงค่าของเวลาที่ใช้ในการประมวลผลของงานบนโหนด v หรือโปรเซสเซอร์ v และ λ(u,v)หมายถึงค่าใช้จ่ายการติดต่อสื่อสารระหว่างโปรเซสเซอร์ที่เกิดขึ้นในการส่งผ่านข้อมูลจากโหนด u ไปยังโหนด v โดยที่ λ(u,v)ไม่เท่ากับศูนย์เมื่อโหนด u และ v อยู่บนโปรเซสเซอร์ที่ต่างกัน และเท่ากับศูนย์เมื่อโหนด u และ v อยู่บนโปรเซสเซอร์เดียวกัน ส่วน makespan ของการกำหนดการ(scheduling)หรือเวลาที่ใช้ในการประมวลผลงานบน DAG คือค่าผลรวมระหว่างโหนดและเส้นทางระหว่างโหนดที่มีค่ามากที่สุดบน DAG นั้นเช่นค่า makespan ของการกำหนดการของ DAG รูปที่ 1 มีค่าเท่ากับ 56

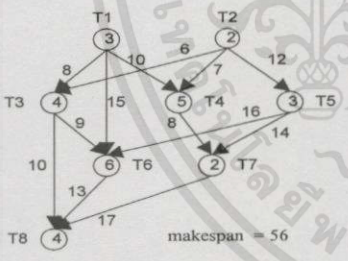


Figure 1) shows weight DAG

การสร้างกลุ่มงานก็คือวิธีการนำเอาโหนดหรืองานจาก DAG นำมารวมกลุ่มกันทำให้เกิดกลุ่มงาน กำหนดให้ Φ เป็นสัญลักษณ์ใช้แสดงเซตของกลุ่มงาน โดย Φ จะประกอบด้วยเซตของกลุ่มงานอย่างน้อยหนึ่งกลุ่ม ดังนั้น $\Phi = \{C_1, C_2, \ldots, C_n\}$ เมื่อ C_i คือกลุ่มงานใดๆ การจัดการกำหนดการของ Φ จะทำได้โดยการนำเอาสมาชิกกลุ่มงานแต่ละตัวใน Φ ไปใส่ในแต่ละโปรเซสเซอร์ ดังนั้นเวลาเริ่มต้นโหนดใดๆหรืองานใดๆในกลุ่มงานมีค่าเท่ากับ S(u,q)+μ(u)+λ(u,v) เมื่อ S(u,q) คือเวลาเริ่มต้นของโหนด u ที่โปรเซสเซอร์ q μ(u) เป็นเวลาที่ใช้ในการประมวลผลของโหนด u และ λ(u,v) เป็นเวลาที่เกิดขึ้นในการติดต่อสื่อสารระหว่างโหนด u และ v ในกรณีที่โหนด u และ v อยู่บนโปรเซสเซอร์เดียวกัน λ(u,v) จะเท่ากับศูนย์ สรุปได้ว่าเวลาเริ่มต้นของโหนดใดๆบนกลุ่มงานเท่ากับ $S(v_i, p) = S(v_{i-1}, p) + \mu(v_{i-1})$ เมื่อ $\forall v_i \in C$ และ $S(v_i, p)$ เท่ากับศูนย์เมื่อ v_i เป็นซอสโหนด

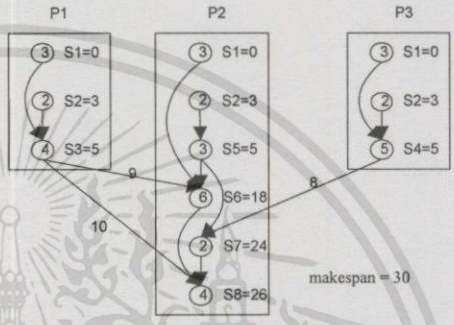


Figure 2) HTCA produces the makespan of a set of clusters

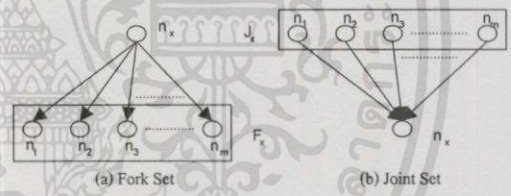


Figure 3) shows a Fork Set and Joint Set for any node on the DAG

DAG จะถูกกล่าวว่าเป็นเม็ดเนื้องานแบบหยาบเมื่อ Granularity ของ DAG มีค่ามากกว่าหรือเท่ากับ 1 ($g(G) \geq 1$) ถ้า DAG ถูกกล่าวว่าเป็นเม็ดเนื้องานแบบละเอียดเมื่อ Granularity ของ DAG มีค่าน้อยกว่า 1 ($g(G) < 1$) ความหมายของ Task Grain และ Granularity จะยึดตามนิยามจาก[3] โดยค่าของ Grain เป็นค่าของอัตราส่วนระหว่างค่าของ μ ต่อ λของ v_i เมื่อ $v_i \in V$ ค่า Grain เป็นค่าของเม็ดเนื้องานในระดับโลคัล(local) ส่วนค่าของ Granularity เป็นค่าของเม็ดเนื้องานในระดับโกลบอล(Grobal) ให้พิจารณาจากรูปที่3 เป็นการแสดงฟอกเซต(Fork Set) ของโหนด n_x(a) และ จอยย์เซต(Joint Set)

ของโหนด $n_x(b)$ จากข้อกำหนดสำหรับ DAG ใดๆ G = (V, E, μ, λ) สำหรับโหนด v ใดๆ Granularity ของ G ถูกแสดงด้วย g(G) และ grain ของงานหรือโหนดใดๆถูกแสดงด้วย g_v เมื่อ v $\in$ V กำหนดให้

Fork Set $F_x = \{n_1, n_2, \ldots, n_m\}$ จะได้

$$g(F_x) = \min_{i=1,2,\ldots,m} (\mu_i) / \max_{i=1,2,\ldots,m} (\lambda_i) \qquad (1)$$

โดยที่λ_iจะต้องเป็นค่าของเวลาที่ใช้ในการติดต่อระหว่างโหนด n_x กับโหนด n_i ใดๆ เมื่อ $\forall n_i \in F_x$

$$g(J_x) = \min_{i=1,2,\ldots,m} (\mu_i) / \max_{i=1,2,\ldots,m} (\lambda_i) \qquad (2)$$

Joint Set $J_x = \{n_1, n_2, \ldots, n_m\}$ จะได้

โดยที่λ_iจะต้องเป็นค่าของเวลาที่ใช้ในการติดต่อระหว่างโหนด n_i กับโหนด n_x ใดๆ เมื่อ $\forall n_i \in J_x$

ดังนั้นจาก (1) และ (2) ค่า Task Grain ของโหนดใดๆบน DAG มีค่าเท่ากับ

$$g_n = \min\{ g(F_x), g(J_x) \} \qquad (3)$$

เมื่อ n คือโหนดใดๆบน DAG ขณะเดียวกันค่า Granularity ของ DAG มีค่าเท่ากับ

$$g(G) = \min_{n=1,2,\ldots,m} \{g_n\} \qquad (4)$$

และ n คือโหนดใดๆบน DAG โดยที่ $\forall n \in V$ และ m คือจำนวนโหนดบน DAG

ปัญหาของการจัดแบ่งกลุ่มงานคือ NP-hard problem [4]แม้ว่าจะไม่มีการจำกัดจำนวนของโปรเซสเซอร์ที่ใช้ก็ตาม งานวิจัยนี้จะแสดงให้เห็นว่า HTCA สามารถพิสูจน์ได้ว่า

$$\tilde{}opt(M_s) \le (1+\varepsilon) opt(M_s) \qquad (5)$$

เมื่อ $\tilde{}$opt(M_s)คือค่าที่ได้จากใช้อัลกอริทึมหรือการทำ approximation algorithm ของ DAG หรือ makespan ของการกำหนดการของ DAG opt(M_s)คือค่าที่ได้จากการทำ optimal กลุ่มงานและ ε คือค่าของ relative approximation algorithm

3) HTCA อัลกอริทึม

การนำเสนอ HTCA จะใช้ back tracking อัลกอริทึมเป็นหลักในการหากลุ่มงานจาก DAG ที่กำหนดขึ้น ขั้นตอนในการสร้างกลุ่มงานจะประกอบด้วย 4 ขั้นตอน 1)การแปลง DAG ให้อยู่ในรูปแบบของต้นไม้กลับตัวชี้(Inverse Pointer tree) 2)การแปลงต้นไม้กลับตัวชี้ให้อยู่ในรูปแบบของต้นไม้แบบทวิภาคกลับตัวชี้(Inverse Pointer Binary Tree) 3)การใช้ Heuristic of Task Clustering อัลกอริทึมสร้างกลุ่มงาน และ 4)จัดการกำหนดการของ Φ ในการสร้างกลุ่มงานจะใช้ DAG จากรูปที่ 1

1) การแปลง DAG ให้อยู่ในรูปแบบของต้นไม้กลับตัวชี้ ในการแปลง DAG ให้อยู่ในรูปของต้นไม้กลับตัวชี้ ให้พิจารณาจากทุกๆจุดของฟอกโหนดบน DAG นั้นๆ รูปที่4แสดงการแปลง DAG รูปที่1ให้อยู่ในรูปของต้นไม้กลับตัวชี้และยังแสดงให้เห็นว่าหลังที่ DAG ถูกแปลงแล้วจะเกิดงานซ้ำซ้อนขึ้นบนต้นไม้ เช่นโหนด T1 และ T2 เป็นโหนดใบไม้(leaf node)

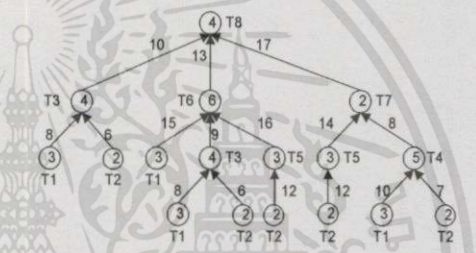


Figure 4) shows transforming the DAG on figure 1 into the inverse pointer tree

2) การแปลงต้นไม้กลับตัวชี้ให้อยู่ในรูปแบบของต้นไม้แบบทวิภาคกลับตัวชี้ จะทำการแปลงโหนดใดๆที่มีจำนวนโหนดลูกหลาน(successor node)มากกว่าสองโหนดขึ้นไปให้เหลือเพียงสองโหนดลูกหลาน ซึ่งโหนดนั้นอาจจะเป็นโหนดราก(root node)หรือโหนดรากต้นไม้ส่วนย่อย(sub-tree root node)ดังแสดงอยู่ในรูปที่5 หลักในการแปลงโหนดลูกหลานของโหนดรากหรือโหนดรากต้นไม้ส่วนย่อยให้เหลือเพียงสองโหนดลูกหลาน โดยการนำเอาโหนดลูกหลานสองตัวแรกที่มีค่าของ λ มากที่สุด(critical path)มารวมกัน จะได้ λ อันใหม่ซึ่งเกิดจากการรวมกันของ λ ของสองโหนดลูกหลาน และจะทำการรวมสองโหนดลูกหลานถัดไปหากยังมีโหนดลูกหลานเหลืออยู่มากกว่าหนึ่งตัว ขณะเดียวกันจะทำการสร้างสเปซโหนด(space node) ขึ้นมาเพื่อใช้เป็นโหนดเชื่อมต่อระหว่างโหนดรากต้นไม้ส่วนย่อยหรือโหนดรากกับโหนด

ลูกหลาน สเปซโหนดมีค่า μ เท่ากับศูนย์ ในขั้นตอนนี้จะใช้ MakeBinaryTree อัล-กอริทึมในการแปลง

```
MakeBinaryTree(v)  // v is root of tree or root of subtree
{  IF( node v is not leaf node)
 {  for( each child of node v)   // u_i = each child of node v
     CCu_i = MakeBinaryTree(u_i)
     sort critical path order of CCu_i
     While( child of v > 2 )
      (1) to combine each pair of child of node v that has largest CCu_i and
          to make CCsp_i = CCu_i + CCu_i+1
      (2) to create space node and link each pair of child of node v
          from (1) to space node then replace root subtree of each pair of
          child node with space node
      (3) to remove connection between node v with child from (1)
      (4) to link node v to space node from (2) with new connection whose
          communication cost equals to CCsp_i
      (5) sort critical path order of CCsp_i
 } Return λ that is communication cost between node v and parent of node v
}
```

ค่าความสลับซับซ้อนของเวลาที่ใช้ของ MakeBinaryTree เท่ากับ O(m log m) ในกรณีที่ m เท่ากับ V แล้วจะได้ O(V log V)

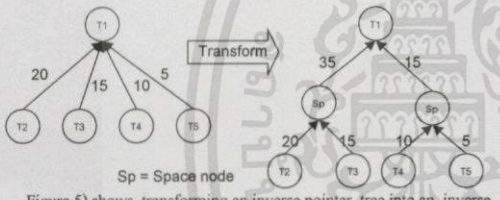


Figure 5) shows transforming an inverse pointer tree into an inverse pointer binary tree

3) การใช้ Heuristic of Task Clustering อัลกอริทึมสร้างกลุ่มงาน การทำงานของ Heuristic of Task Clustering(HTC)เหมือนกับ Back Tracking อัลกอริทึมมาประยุกต์ใช้งาน HTCจะการทำแบ่งโหนดบนต้นไม้ออกเป็นกลุ่มๆ ค่าความสลับซับซ้อนของ HTC เท่ากับ $O(|V^2|)$ เมื่อ V คือจำนวนโหนดบน DAG HTC จะพยายามลดช่วงเวลาหยุดนิ่งบนโปรเซสเซอร์ ในขณะเดียวกันจะพยายามรักษาลักษณะการทำการของโปรแกรมแบบขนานให้มากที่สุดและเวลาที่ใช้ในการประมวลผลงานแบบขนานให้น้อยที่สุดด้วย จากรูปที่6 แสดงวิธีการหาค่า MIN_{LR} ค่า MIN_{LR} เป็นค่าที่ได้จากการหาเวลาเริ่มต้นที่น้อยที่สุดสำหรับโหนดที่จะถูกนำเข้ามารวมในกลุ่มงาน

$$MIN_{LR} = \min(MAX_L, MAX_R) \qquad (6)$$

สมการที่ (6) แสดงถึง min-max criterion สำหรับกลุ่มงานใดๆ โดย (6) จะพยายามให้เกิดเวลาที่ใช้ในการประมวลผลให้น้อยที่สุดสำหรับกลุ่มงานหรือเรียกว่า" Optimum clustering"และกระบวนการทำงานแบบขนานให้มากที่สุด

L = left sub-tree, R = right sub-tree

μ_L = computation cost of left child node of node v

μ_R = computation cost of right child node of node v

μ_v = computation cost of node v

λ_L = communication cost between left child node and node v

λ_R = communication cost between right child node and node v

$Start_v$ = Started time of node v

C_v = Task clustering at node v

CR = Task clustering at right child sub-tree of node v

CL = Task clustering at left child sub-tree of node v

$Compl_v$ = Completion time of node v

```
AdaptiveTaskCluster(v, λ, start, compl, C)
{  IF(node v is not leaf node)
   { AdaptiveTaskCluster( v_L , λ_L , Start_L , Compl_L , CL)
     AdaptiveTaskCluster( v_R , λ_R , Start_R , Compl_L , CR )
     MAX_L = max(compl_L + λ_L , compl_R)
     MAX_R = max(compl_L , compl_R + λ_R)
     MAX_v = compl_(CR∩CL) + compl_~(CR∩CL)
     MIN_LR = min(MAX_L ,MAX_R)
     IF( node v is space node) //**** case 1
      {  C_v = C_v ∪ ( CR ∩ CL ) ∪ ~ ( CR ∩ CL)
         Start_v = max(compl_L , compl_R ) + ∃(μ(v)) when v ∉(either L or R)
                   but v∈(either L or R) that depends on max(compl_L , compl_R )
                   such as v ∉ L but v∈R
      } else IF( MIN_LR < MAX_v) // *** case 2
        { C_v = C_v ∪ {task clustering left or right child of node v that depends
          on MIN_LR }
          Start_v = MIN_LR
          Φ = Φ ∪ {task clustering that depends on condition of MIN_LR }
      } else IF(MIN_LR >= MAX_v) //*** case 3
        { IF( ( CL ∩ CR) = ϕ ) //  ϕ means empty set    *** case 3.1
            C_v = C_v ∪ CL ∪ CR
            Start_v = compl_R + compl_L
        } else IF (( ∀L_i ∈ CR) or ∀R_i ∈CL)) //when L_i ∈ CL and
                                              R_i ∈CL  ***case 3.2
          { C_v = C_v ∪ {CR or CL that depends on condition}
            Start_v = compl_CR or compl_CL // depends on condition
        } else  //*** case 3.3
          { C_v = C_v ∪ {task clustering left or right child of node v
                  that depends on  MIN_LR }
            Start_v = MIN_LR
            Φ = Φ ∪ {task clustering that depends on condition of MIN_LR }
          }  compl_v = Start_v + μ_v
}
```

}else { $\lambda = \lambda_v$

Start = $Start_v$

Compl = Start + μ_v

$C = C \cup C_v$ }

} // End

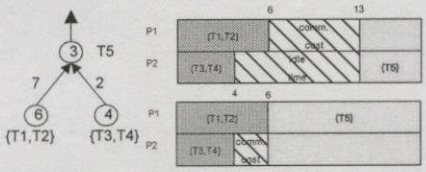


Figure 6) shows to find minimum started time for a new node

4) จัดการกำหนดการของ Φ ผลที่ได้จาก HTC จะได้เซตของกลุ่มงาน Φ เกิดขึ้น จากนั้นจะนำเอา Φ มาทำการจัดการกำหนดการให้กับ Φ(DAG) Φ ที่ได้จาก HTC อาจจะได้กลุ่มงานที่ซ้ำซ้อนกัน ดังนั้น Φ ที่ได้จาก HTC ในขั้นตอนที่3 ภายใต้โจทย์ของ DAG รูปที่ 1 Φ={{T1,T2,T3},{T1,T2,T3},{T1,T2,T5,T6,T7,T8},{T1, T2,T4}} จะใช้ ScheduleOfTask(C,Φ) จัดการกำหนดการให้กับ Φ โดยอินพุท(input)ของอัลกอริทึมคือกลุ่มงานที่มีซิงค์โหนดประกอบอยู่ด้วยในกลุ่มงาน(C)และΦ เช่นจาก Φ ของ DAG รูปที่ 1 จะใช้ C={T1,T2,T5,T6,T7,T8} เป็นอินพุท

ScheduleOfTask(C, Φ) // C is task clustering

{ Φ(DAG) = ϕ // Φ(DAG) is set of task clustering of a DAG

Φ(DAG) = Φ(DAG) $\cup$ C

for (each node or task of C) // $c_i \in C$ when i = 1,2,3,…,n

1) to pick mark node c_i

2) IF c_i has an arc connected to outside cluster ($\lambda_{u,w}$ when w = c_i and u $\notin$C but u $\in$ other cluster, w $\in$C)

Then union cluster of u to Φ(DAG) // u must be sink node of cluster

}

ค่าความสลับซับซ้อนของเวลาของ ScheduleOfTask มีค่าเท่ากับO(n+e) เมื่อ n คือ จำนวนโหนดและ e แสดงจำนวนอาร์คของ C ที่มีการติดต่อไปยังโหนดที่อยู่ภายนอกกลุ่มงาน ดังนั้นกรณี worst cast ของ ScheduleOfTask เท่ากับ O(|V|+|E|) โดยสรุปจากทั้ง4ขั้นตอนค่าความสลับซับซ้อนของเวลาของอัลกอริทึมเท่ากับ O(|V| log |V|) + O($|V^2|$) + O(|V| + |E|) ดังนั้นค่าความสลับซับซ้อนของเวลาของ HTCA เท่ากับ O($|V^2|$)

4) ผลการทดลอง

รูปที่7 แสดงเซตของกลุ่มงานที่ได้จาก HTC และยังแสดงให้เห็นว่ามีงานซ้ำซ้อนเกิดขึ้นในระหว่างกลุ่มงาน และแสดงให้เห็นค่าของเวลาเริ่มต้นที่น้อยที่สุดที่เป็นไปได้ของซิ้งค์โหนดในกลุ่มงานเช่น T3,T8 เป็นต้น รูปที่2แสดงการกำหนดการของเซตของกลุ่มงานโดยการกำหนดการที่ได้มาจากScheduleOfTask และแสดงอาร์คระหว่างกลุ่มงาน จากรูปที่2แสดงให้เห็นว่าจะใช้สามโปรเซสเซอร์สำหรับการประมวลของงานสำหรับ DAG รูปที่1 ขณะเดียวกันอัลกอริทึมทำการลดโอเวอร์เฮดระหว่างโปรเซสเซอร์และค่า makespan ของเซตของกลุ่มงานมีค่าเท่ากับ 30 ซึ่งเป็นค่าที่น้อยกว่าค่า makespan ของ DAG เท่ากับ 56 สรุปได้ว่า HTCA สามารถลดจำนวนโปรเซสเซอร์ จำนวนอาร์คระหว่างโปรเซสเซอร์ ช่วงเวลาหยุดนิ่ง และในขณะเดียวกันเป็นการเพิ่มค่าใช้จ่ายในการประมวลผล(computation cost)ที่โปรเซสเซอร์มากขึ้น

THEOREM 1 . From [2] for the granularity of a DAG that has g(G) $\geq (1-\varepsilon)/\varepsilon$, where $0 < \varepsilon \leq 1$, then

$$\tilde{}opt(M) \leq (1+\varepsilon)opt(M)$$

where $\tilde{}opt(M)$ is approximation algorithm to instance X or makespan of schedule of Φ, $opt(M)$ is optimal solution to instance X or makespan of an optimal schedule of task clustering, and ε is relative approximation algorithms.

PROOF . $\tilde{}opt(M) \leq (1+\varepsilon)opt(M)$

(makespan produces by HTCA) $\leq (1+\varepsilon)$(started time of task T8 + μ(T8) that produced by HTC), when the g(G) of a DAG on figure 1 is 1/7

(30) $\leq (1+\varepsilon)((S_8 = 26) + (\mu_8 = 4))$

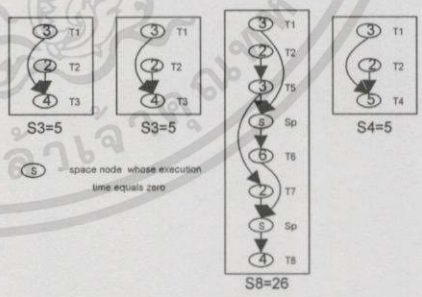


Figure 7) shows the result of HTC algorithm

5) การเปรียบเทียบ

การวัดประสิทธิภาพของ HTCA โดยการเปรียบเทียบกับอัลกอริทึมอื่นๆ โดยจะเปรียบเทียบ HTCA กับ [2] การเปรียบเทียบผลการทดลองจะใช้ DAG ที่มีลักษณะเป็นเม็ดเนื้องานแบบละเอียด รูปที่9(b)แสดงผลลัพธ์ของการ

ใช้[2] กับ DAG รูปที่1 จะพบว่า makespan ที่ได้มีค่าเท่ากับ makespan ของHTCA แต่จำนวนของโปรเซสเซอร์และช่วงเวลาหยุดนิ่งที่ได้จาก[2] จะมากกว่า HTCA ดังแสดงในรูปที่9 นอกเหนือจากการพิสูจน์ของ HTCA กับ [2] เมื่อใช้ DAG รูปที่1 ดังกล่าวแล้ว ได้ทำการทดลองกับ DAG ของ [2] ด้วยHTCA ซึ่งการพิสูจน์พบว่าผลที่ได้ยังคงให้ผลเช่นเดียวกัน รูปที่8และ10(a)แสดงผลลัพธ์ที่ได้จากการใช้ HTCA กับ DAG ใน[2] ค่า makespan ของHTCA ที่ได้รับมีค่าเท่ากับ makespan ของ [2] ขณะเดียวกันรูปที่ 10(b)แสดงจำนวนโปรเซสเซอร์และช่วงเวลาหยุดนิ่งที่ได้จาก[2] ยังคงมากกว่า HTCA แสดงให้เห็นว่าถึงแม้ค่าความสลับซับซ้อนของเวลาของ HTCA ($O(|V^2|)$) มากกว่า [2] ($O(|V|\,(\,|V|\log|V| + |E|\,))$) ก็ตาม แต่ประสิทธิภาพของเวลาที่ใช้หรือ makespan ของ HTCA ไม่มากกว่า [2]

THEOREM 2. If Φ_{HTCA} is optimal for a DAG G if for every other Φ'_{HTCA} for G, then $makespan(\Phi_{HTCA}) \le makespan(\Phi'_{HTCA})$

PROOF . From figure 9 and 10, Given the $\Phi_{[2]}$ is makespan of Φ for a DAG G and is produced by [2] algorithm, so that

$$makespan(\Phi_{HTCA}) \le makespan(\Phi_{[2]})$$

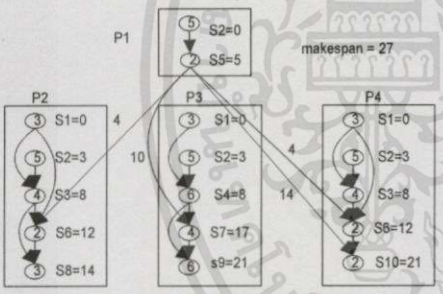


Figure 8) shows HTCA produces the task clustering on the DAG of [2]

6) สรุปผลการทดลอง

จากรูปที่9 และ 10 แสดงให้เห็นว่า HTCA สามารถลดโอเวอร์เฮด จำนวนโปรเซสเซอร์ที่ใช้ และช่วงเวลาหยุดนิ่งได้อย่างมีประสิทธิภาพ ขณะเดียวกันค่า makespan ของ HTCA ไม่ด้อยไปกว่า[2] ซึ่ง [2] จะใช้จำนวนโปรเซสเซอร์ และช่วงเวลาที่หยุดนิ่งมากกว่าHTCA ถึงแม้ HTCA จะมีค่าความสลับซับซ้อนของเวลามากกว่า [2] ก็ตาม

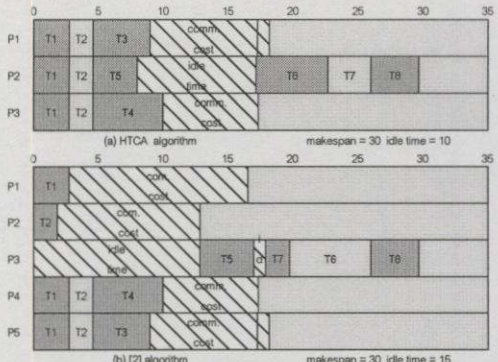


Figure 9) shows the number of processors, makespan, and idle time that are produced by HTCA and [2]

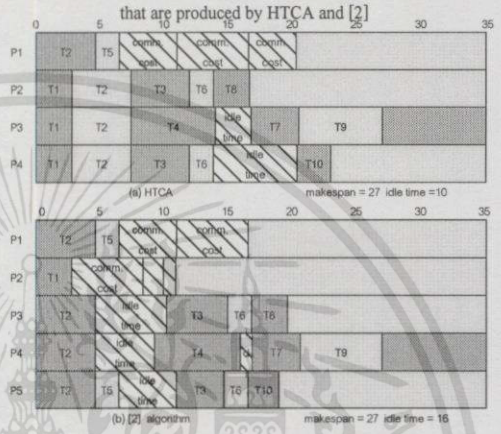


Figure 10) shows the result that produces by HTCA and [2]

7) หนังสืออ้างอิง

[1] C.H. Papadimitriou and M. Yannakakis, "Towards an architecture-indepent analysis of parallel algorithms," SIAM J. Computing, vol.19, no.2, pp. 322-328, April 1990.

[2] M.A. Palis, J.C. Liou and D.S.L. Wei, "A Greedy Task Clustering Heuristic that is Provably Good," Proc. IEEE ISPAN, pp. 398-405, December 1994.

[3] A. Gerasoulis and T. Yang, "On the granularity and clustering of directed acyclic task graphs," IEEE Trans. Parallel and Distributed Systems, vol.4, no.6, pp. 686-701, June 1993.

[4] M.R. Garey and D.S. Johnson, Computers and Intractabiliry A Guide to the theory of NP-Completeness, W.H. Freeman and Company, 1979.

Adaptive Task Clustering and Scheduling Algorithms for Parallel Processor Architecture

Banjong Piyatamrong* Shigeyuki Ohara** Sakchai Kantakajorn*
*Department of Computer Engineering, Faculty of Engineering,
King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand.
Email: kpbanjon@kmitl.ac.th kmitl_toon@hotmail.com
**Department of Electronics, Faculty of Engineering, School of Engineer, Tokai University, Japan.

Abstract

The optimization partition task on parallel program is a problem of partitioning the tasks on a directed acyclic task graph(DAG)[1] onto the processors of a multiprocessor in a way that minimizes the execution time. The task clustering problem is NP hard-problem[6][7], although a number of processor is not define and a duplicate task between clusters are allowed. The Adaptive Task Clustering and Scheduling Algorithms(ATCS) tries to reduce the overhead communication cost between processor by transforming a fine grain task to a coarse grain task. The ATCS has a time complexity of algorithm equals to $O(|V^2|)$.

1. Introduction

[5]introduces method called the "Task Clustering". The task clustering method is categorized into two types. The first type is called "non linear" and the second type is called "linear". The non linear is a task clustering, which has at least two independent parallel task and place on the same processor, otherwise it is called linear. The target of task clustering method is to reduce the execution time of parallel program on an unbound number of processors.

2. Theory

To show stage and oriented of parallel program for this paper will be using weighted directed graph $G=(V,E,\mu,\lambda)$ or weighted DAG G, where V is the set of nodes or tasks and each node $v_i \in V$ represents a task whose execution time is $\mu(v_i)$ and E is the set of arcs and each arc $e_i \in E$ represent task u should complete its execution time and transfer data from task u to v before task v can be started, where $\forall(u_i \text{ and } v_i) \in V$, so that a DAG shows precedence of tasks. If u_i and v_i are reside in difference processors, the $\lambda(u_i, v_i)$ is not zero and zero otherwise.

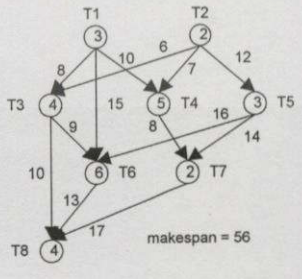


Figure 1) a weighted DAG

The figure 1 shows a weighted DAG G that is a fine grain task. Each node on a DAG resides in difference processors. The makespan(or critical path)of schedule of a DAG is a total time of a maximum execution cost of nodes and communication cost of arcs. Thus the makespan of schedule on a DAG of figure 1 is 56. A task clustering method is collection node on a DAG that a result of that method produces several task clusters. Assume algorithm produces a number of task clusters then each task cluster is placed on only one processor. To define Φ symbol is a set of task clustering that contains at least one task cluster. When Φ contains $\{C_1, C_2, \ldots, C_n\}$where C_i is a task cluster and subscribe $i = 1,2,\ldots,n$, if v_i is member of C_i ($v_i \in C_i$) that must be a member of $V(v_i \in V)$. The Φ may be consisting of a set of task clustering, which is either task clustering without duplication or with duplication.

To construct a schedule for Φ can be perform by placing each cluster C_i, where $C_i \in \Phi$, on only one processor. The started time of node v on processor p is presented by $S(v,p) = S(u,q) + \mu(u) + \lambda(u,v)$, where u and v are mapped to distinct processors. But when u and v are mapped on a same processor p that $\lambda(u,v)$ is zero.

To consider a DAG is either a fine grain or a coarse grain task that depends on the granularity of a DAG or g(G). If the granularity of a DAG is greater than or equal to $1(g(G) \geq 1)$that a DAG is a coarse grain task and otherwise is a fine grain task[5]. A DAG consists of several fork and joint set as shown in[5] shows a fork and joint set of node n_x. The fork set F_x consists of immediate successors of node n_x. The joint set J_x consists of immediate predecessors of node n_x. The task clustering with and without duplication is NP-hard problem although is not define a number of processors and allows task duplication between processors as other problems. This paper will be represented that ATCS can be performed[7]

$$\sim opt(M_s) \leq (1+\varepsilon)opt(M_s) \qquad (1)$$

where $\sim opt(M_s)$ is approximation algorithm to instance X or makespan of schedule for Φ that returns from algorithm, $opt(M_s)$ is optimal solution to instance X or makespan of optimal schedule of task clustering for G, and ε is a relative approximation algorithm[6][7]. From (1) we can prove that it is true for ATCS algorithm.

For this paper presents a new algorithm. Which named ATCS algorithms. The efficiency of ATCS algorithm is compared with [3][4] algorithms. The concept design of ATCS attempts to reduce the execution time of parallel

program with an unbound number of processors. The ATCS compacts criterion several tasks, which is maximum completion time, on a graph into task clustering. At the same time, It makes parallel time minimization as equation below:

$$\min_{\Phi}\{\max_{j=1,2,\ldots,v} CT(T_j)\} \qquad (2)$$

where i is a task on a graph, $CT(T_i)$, and $v \in V$.

3. ATCS Algorithms

To construct task clustering will be using ATCS algorithm that consists of four stages as the following:

1) Transforming a DAG into an inverse pointer tree that lets to consider at every fork node on a DAG. Figure 1 represents a result of transforming that DAG into the inverse pointer tree.

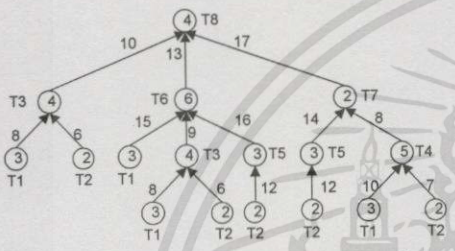


Figure 2) Transforming a DAG to an inverse pointer tree

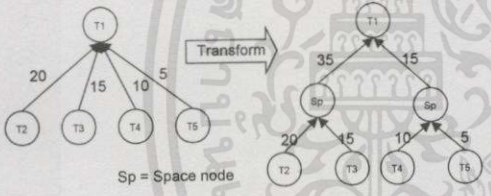


Figure 3) Transforming an inverse pointer tree into an inverse pointer binary tree

2) Transforming an inverse pointer tree into an inverse pointer binary tree. This stage is reducing a number of successor nodes of any node on a DAG that exceeds two successors into only two successor nodes. Figure 3 shows to transform an inverse pointer tree into an inverse pointer binary tree and note that in the figure has space node on an inverse pointer binary tree. The computation cost of space node is zero. We use MakeBinaryTree Algorithm to implement for this stage.

```
MakeBinaryTree(v) // v is root of tree or root of subtree
{ IF( node v is not leaf node)
  { for( each child of node v)  // u_i = each child of node v
      CCu_i = MakeBinaryTree(u_i)
      sort critical path order of CCu_i
      While( child of v > 2 )
       (1) to combine each pair of child of node v that has
           largest CCu_i and to make  CCsp_i = CCu_i + CCu_i + 1
       (2) to create space node and link each pair of child of node
           v from (1) to space node then replace root subtree of
           each pair of child node with space node
       (3) to remove connection between node v with child from
           (1)
       (4) to link node v to space node from (2) with new
           connection whose communication cost equals to CCsp_i
       (5) sort critical path order of CCsp_i
  }
  Return λ that is communication cost between node v and parent
         of node v
}
```

The complexity time of MakeBinaryTree algorithm is O(m lg m) time, where m is a number of successors of node v on a tree. In worst cast, if m equals to V, where V is a number of nodes on a DAG, the complexity time of MakeBinaryTree is O(V lg V) time.

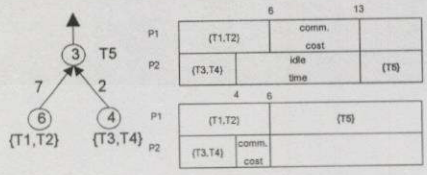


Figure 4) to find MIN_{LR}

3) Using Adaptive Task Clustering algorithm (ATC) produces task clustering. The ATC partitions all nodes on an inverse pointer binary tree into several clusters. The ATC produces a makespan of an optimal schedule for each task clustering. ATC finds a node on a tree by started from source node to sink node that represents as bottom up.

$$MIN_{LR} = \min(MAX_L, MAX_R) \qquad (3)$$

where MAX_L and MAX_R are a maximum completion time of left and right sub-tree respectively. Figure 4 shows an example to find MIN_{LR} of node T5, which is added to the cluster. From (3) explained min-max criterion for task clustering definition of (2). Thus (3) attempts the minimum parallel time is called the "Optimum clustering" and the maximum parallel processing. The complexity time of ATCS algorithm equals to $O(|V^2|)$ time, where V is a number of nodes on a DAG because each node can be successor of other nodes at most V time. The following is a symbol of variables of ATC algorithm:

L = left sub-tree, R = right sub-tree
μ_L = computation cost of left child node of node v
μ_R = computation cost of right child node of node v
μ_v = computation cost of node v
λ_L = communication cost between left child node and node v
λ_R = communication cost between right child node and node v
$Start_v$ = Started time of node v
C_v = Task clustering at node v
CR = Task clustering at right child sub-tree of node v
CL = Task clustering at left child sub-tree of node v
$Compl_v$ = Completion time of node v

```
AdaptiveTaskCluster(v, λ, start, compl, C)
{ IF(node v is not leaf node)
   { AdaptiveTaskCluster( v_L , λ_L , Start_L , Compl_L , CL)
     AdaptiveTaskCluster( v_R , λ_R , Start_R , Compl_L , CR )
     MAX_L = max(compl_L + λ_L , compl_R)
     MAX_R = max(compl_L , compl_R + λ_R)
     MAX_v = compl_(CR ∩ CL) + compl_~(CR ∩ CL)
     MIN_LR = min(MAX_L ,MAX_R)
     IF( node v is space node) //**** case 1
       {  C_v = C_v ∪ ( CR ∩ CL ) ∪ ~ ( CR ∩ CL)
```

$Start_v = \max(compl_L, compl_R) + \exists(\mu(v))$
when v ∉(either L or R) but v∈(either L or R) that depends on $\max(compl_L, compl_R)$ such as v ∉L but v∈R
} else IF($MIN_{LR} < MAX_v$) // *** case 2
{ $C_v = C_v \cup$ {task clustering left or right child of node v that depends on MIN_{LR} }
$Start_v = MIN_{LR}$
$\Phi = \Phi \cup$ {task clustering that depends on condition of MIN_{LR} }
} else IF($MIN_{LR} >= MAX_v$) //*** case 3
{ IF((CL ∩ CR) = ϕ) // ϕ means empty set *** case 3.1
$C_v = C_v \cup CL \cup CR$
$Start_v = compl_R + compl_L$
} else IF (($\forall L_i \in CR$) or $\forall R_i \in CL$)) //when $L_i \in CL$ and $R_i \in CL$ ***case 3.2
{ $C_v = C_v \cup$ {CR or CL that depends on condition}
$Start_v = compl_{CR}$ or $compl_{CL}$ // depends on condition
} else //*** case 3.3
{ $C_v = C_v \cup$ {task clustering left or right child of node v that depends on MIN_{LR} }
$Start_v = MIN_{LR}$
$\Phi = \Phi \cup$ {task clustering that depends on condition of MIN_{LR} }
} $compl_v = Start_v + \mu_v$
}
}else { $\lambda = \lambda_v$
Start = $Start_v$
Compl = Start + μ_v
C = C ∪ C_v }
} // End

lemma 1 *When a new node v_{new} is added to a cluster that the started time of v_{new} is always greater than or equal to started time of all nodes both left or right sub-tree.*
proof Given v_{new} is a new node, which is added into a cluster.

$$S(v_{new}, p) \geq \sum_{i=1,2,..,n} S(v_i, p) + \mu(v_i)$$

where $\forall v_i \in C_i$, $S(v_i, p)$is zero if v_i is source node on a DAG, C_i is cluster, n is a number of node in cluster, and p is processor which C_i is placed.

4) Constructing a schedule for Φ from the last stage, the ATC algorithm produces a set of task clustering Φ. In final stage, we will be constructing a schedule to Φ. The Φ of the DAG on figure 1 that consists of four task clusters and appears task duplication between processors. To construct schedule to Φ of performed by ScheduleOfTask algorithm. The input of ScheduleOfTask consists of two parameters, which are a task clustering C and Φ. The C must has a sink node on it, such as a task clustering C={T1,T2,T5,T6,T7,T8} is a member of Φ and is a parameter of ScheduleOfTask algorithm, while it has a T8 as a sink node on the DAG.

ScheduleOfTask(C, Φ) // C is task clustering
{ Φ(DAG) = ϕ // Φ(DAG) is set of task clustering of a DAG
Φ(DAG) = Φ(DAG) ∪ C
for (each node or task of C) // $c_i \in C$ when i = 1,2,3,…,n
1) to pick mark node c_i
2) IF c_i has an arc connected to outside cluster ($\lambda_{u,w}$ when w = c_i and u ∉C but u ∈ other cluster, w ∈C)
Then union cluster of u to Φ(DAG) // u must be sink node of cluster
}

The complexity time of ScheduleOfTask algorithm is O(|V| + |E|) time, where V is a number of nodes in a C_i and E is a number of arcs in C_i that is connected to several outside task clusters. In the summary, the over all of the complexity time takes O(|V| lg |V|) + O($|V^2|$) + O(|V| + |E|) time, thus the complexity time of ATCS algorithm equals to O($|V^2|$) time.

4. Result of experience

Figure 5 shows a set of task clusters that is produced by ATC algorithm, and it shows task duplication between clusters. Figure 6 represents a schedule of a set of task clusters, which is produced by ScheduleOfTask algorithm, and it shows connection or arc between several clusters. Figure 6 shows the makespan of the Φ that is 30, which is less than a makespan of a DAG that is 56, so that ATCS algorithm reduces a number of processors, arcs or connection, idle time, and increases computation cost at processors.

Theorem 1 *From [4] for the granularity of a DAG that has g(G) ≥ (1-ε)/ε, where 0<ε≤1, then*
$\sim opt(M_s) \leq (1+\varepsilon) opt(M_s)$
where $\sim opt(M_s)$ is approximation algorithm to instance X or makespan of schedule of Φ, $opt(M_s)$ is optimal solution to instance X or makespan of an optimal schedule of task clustering, and ε is a relative approximation algorithms.
proof.
$\sim opt(M_s) \leq (1+\varepsilon) opt(M_s)$
or (makespan produces by ATCS)≤(1+ε)(started time of task T8 + μ(T8) that produced by ATC)
$(30) \leq (1+\varepsilon)((S_8=26) + (\mu_8=4))$

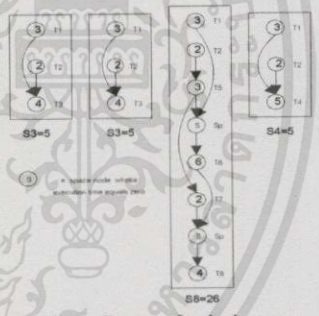


Figure 5) a set of task clusters

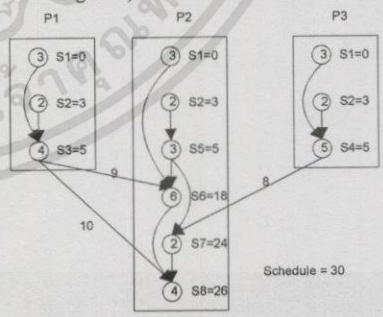


Figure 6) ATCS produces the makespan of schedule of Φ

5. Comparison with other algorithms

The efficiency of ATCS algorithm is compared with [4]. Figure 7 shows the number of processors, makespan, and idle time of both algorithms that ATCS algorithm can reduces more idle time at processors and the number of

processors than [4] performs. From a example shows that ATCS algorithm does not give lower efficiency to perform parallel program than [4], when does on a fine grain task.

6. A Coarse Grain Task

In this section discuss a result of a coarse grain task, which is performed by ATCS, [4], and [3] algorithms. A DAG is used by analyze that it appeared on [3], which represents a coarse grain task. The previous section does not discuss [3] because [3] can performs on a DAG that is only a coarse grain task. Figure 8 shows the number of processors, makespan, and idle time, which are produced by ATCS, [4], and [3] respectively. The result represents that ATCS can performs as well on a coarse grain task as [3] and [4] performed. But ATCS does not guarantee that it performs well on every case.

Theorem 2 If Φ_{ATCS} is optimal for a DAG if for every other Φ'_{ATCS} for G, then

$$makespan(\Phi_{ATCS}) \leq makespan(\Phi'_{ATCS})$$

proof From figure 8, Given the $\Phi_{[3]}$ is maespan of Φ for a DAG G and is produced by [3], and $\Phi_{[4]}$ is makespan of Φ for a DAG G and is produced by [4], so that

$$makespan(\Phi_{ATCS}) \leq makespan(\Phi_{[3]})$$
$$and$$
$$makespan(\Phi_{ATCS}) \leq makespan(\Phi_{[4]})$$

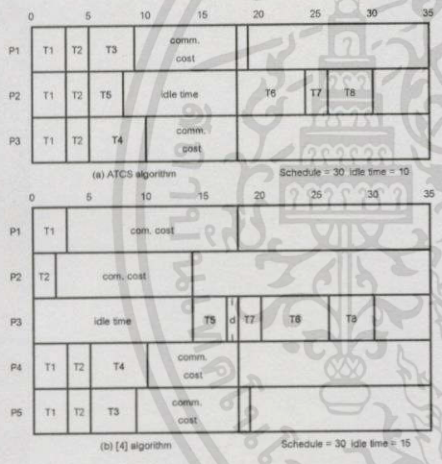


Figure 7) the number of processors, makespan, an idle time that are produced by ATCS and [4]

7. Conclusion

Algorithm	makespan	Idle time	A number of Processors	Complexity Time
ATCS	21	0	5	$O(\|V^2\|)$
[4]	21	0	5	$O(V(V \lg V + E))$
[3]	21	5	4	$O(\|V^2\|)$

Table 1) comparison result of ATCS, [4], [3] on a coarse grain task

The table 1 represents the comparison result of ATCS, [3], [4] on a coarse grain task. In a case of a coarse grain task, and the ATCS produces as a makespan of schedule as [4] and [3]. As you can see note that ATCS can solve problem both on a find grain and coarse grain task by does not modified any things.

References

[1] V. Sarkar, Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors. Cambrige, Mass.: MIT Press, 1989.
[2] C.H. Papadimitriou and M. Yannakakis, "Towards an architecture-indepent analysis of parallel algorithms," SIAM Journal Computing, vol.19, no.2, pp.322-328, April 1990.
[3] S. Darbha and D,P. Agrawal, "Optimal Scheduling Algorithm for Distributed-Memory Machines," IEEE Trans. Parallel and Distributed Systems, vol.9, no.1, Jan. 1998.
[4] M.A. Palis, J.C. Liou and D.S.L. Wei, "Task Clustering and Scheduling for Distributed Memory Parallel Architectures," IEEE Trans. Parallel and Distributed Systems, vol.7, no.1, Jan. 1996.
[5] A. Gerasoulis and T. Yang, "On the granularity and clustering of directed acyclic task graphs," IEEE Trans. Parallel and Distributed Systems, vol.4, no.6, pp. 686-701, June 1993.
[6] M.R. Garey and D.S. Johnson, Computers and Intractability A Guide to the theory of NP-Completeness, W.H. Freeman and Company, 1979.
[7] G. Brassard and P. Bratley, Fundamentals of Algorithms, Prentice-Hall International, Inc., 1996.

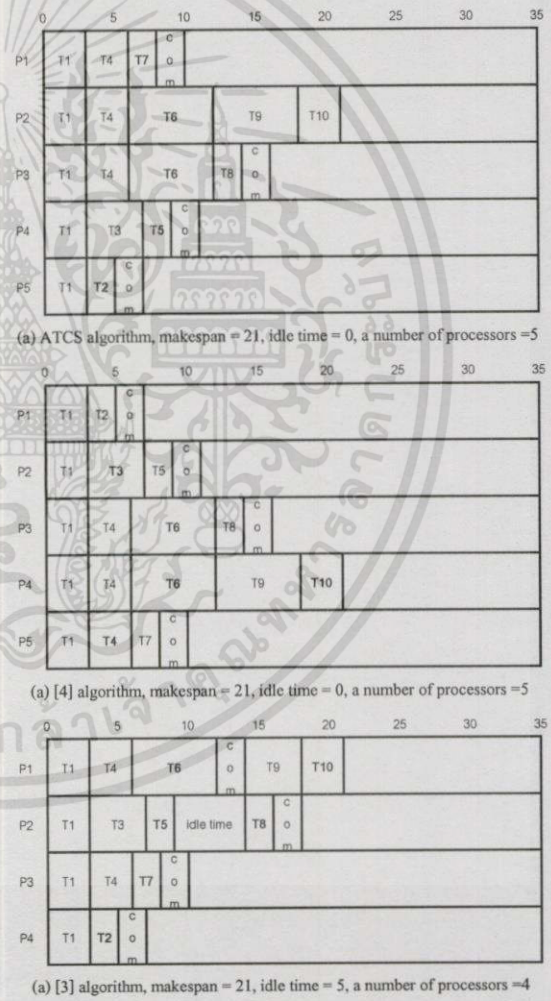


Figure 8) the number of processors, makespan, and idle time that are produced by (a) ATCS (b) [4] and (c) [3] algorithms

GTCS : A Greedy Task Clustering and Scheduling Algorithms for Distributed Memory Processors Architecture

*Banjong Piyatamrong
kpbanjon@kmitl.ac.th

**Shigeyuki Ohara
ohara@keyaki.cc.u-tokai.ac.jp

*Sakchai Kantakajorn
s0061063@kmitl.ac.th

* Department of Computer Engineering, Faculty of Engineering,
King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok 10520, Thailand.
** Department of Electronics, School of Engineering, Tokai University,
259-12 Kitakaname, Hiratsuka-shi, Kanagawa, Japan.

Abstract

This paper presents a task-clustering algorithm named A Greedy Task Clustering and Scheduling Algorithms (GTCS). The algorithm represents tasks of a parallel program as nodes and scheduling them with Directed Acyclic Task Graph (DAG). The parallel hardware architecture focused in this paper is the distributed memory architecture. The GTCS implements the task-clustering method by partitioning nodes on a DAG. Each group or cluster within a DAG is placed on only one processor. The algorithm partitions the DAG into several clusters to reduce the communication overhead between processors. Because the network has a lot of communication overhead that increases total execution time or makespan of the tasks, especially in the parallel program that are fine grain tasks on a network. The GTCS algorithm produces a schedule whose makespan is at most twice of the optimal cluster. The GTCS has a complexity time of algorithm equals to $O(m(|V|lg|V|+|E|))$ where m is a number of clusters and $1 \le m < V$.

1. Introduction

The efficiency of execution time of parallel program on parallel architecture depends on the algorithm, which partitions the parallel program into groups or clusters. The target of partitioning parallel program on parallel architecture is to reduce an execution time or makespan. It can be claimed that the result of transforming from a fine grain task to a coarse grain task using the partition algorithm is to reduce the total execution time of parallel program [5].

[1] represents a new algorithm that has a complexity time equals to $O(|V|(|V|lg|V| + |E|))$ time. [1] is proved to have a better performance guaranteed by explicitly taking into granularity of the DAG. For a DAG G with uniform task execution times (μ) and uniform communication cost (λ) then the granularity of G, g(G), equals to μ/λ. [1] shows that if $g(G) \ge (1- \varepsilon)/\varepsilon$ for some $0 < \varepsilon \le 1$, the algorithm can schedule the task cluster that produces a makespan of at most $(1+\varepsilon)$ times of the optimal schedule makespan. For a DAG with arbitrary granularity, $g(G) \ge 0$, the algorithm can be produced a schedule that is at most twice of the optimal one.

2. Theory

A weighted directed graph $G=(V,E,\mu,\lambda)$ or weighted DAG G, where V is the set of nodes or tasks and each node. $v_i \in V$ represents a task whose execution time is $\mu(v_i)$ and E is the set of arcs. Each arc $e_i \in E$ represent an arc whose communication cost is $\lambda(u_i , v_i)$. $\lambda(u_i , v_i)$ represents a cost that task u completes its execution and transfers the result from task u to task v, where $\forall(u_i$ and $v_i) \in V$. If u_i and v_i are resided in difference processors, the $\lambda(u_i ,v_i)$ is not zero and zero otherwise. Figure 1 shows a weighted DAG G that is a fine grain task. Each node or task on a DAG resides in difference processors. The makespan of a scheduled DAG is the maximum of the summation of the node execution cost and the arcs communication cost. Thus makespan of schedule on a DAG of figure 1 is 52. The granularity of a DAG or g(G) is used to determine that a DAG is either a fine grain or a coarse grain task. If the granularity of a DAG is greater than or equals to 1 ($g(G) \ge 1$) then the DAG is a coarse grain task but if $g(G) < 1$ that the DAG is a fine grain task [2]. A grain value is a ratio of the computation cost μ and the communication cost λ of v_i, where $v_i \in V$. A DAG $G=(V,E,\mu,\lambda)$ for node v has a granularity of g(G) and a task grain of g_v, where $v \in V$.

3. The GTCS Algorithm

This section presents the GTCS algorithm. The GTCS creates a set of task clustering, Φ, and schedules a set of task clustering, $\Phi(G)$, for a task graph with arbitrary granularity. The GTCS produces a schedule whose makespan is at most twice of the optimal cluster according to the definition: for any DAG G, which has $g(G) \ge 0$, then the makespan of $\Phi(G)$ is at most $(1+(1/(1+g(G))))$ times makespan of the optimal cluster for any DAG G[1]. As mention before, the GTCS algorithm is developed from [1] by employing a new method to find the started time e(v) on a DAG G that spends a less time complexity than the algorithm [1]. Because the algorithm [1] started time depends on the depth of DAG G. If a DAG G has more depth, it requires more running time. GTCS uses only the information of

0-7695-0589-2/00 $10.00

the immediate predecessors for finding started time e(v), therefore, the source node started time is always zero.

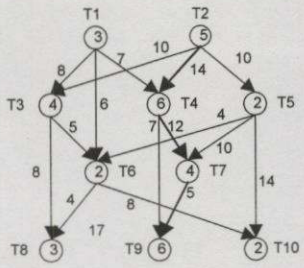


Figure 1 : A weight DAG

3.1 Finding Started time e(v) and Creating Φ from DAG G

Defines ω as a symbol that represents a dynamic array and ψ represents a binary tree that stores the sink node of C_i, which is any task cluster, on Φ. All sink nodes of C_i on ψ stored on the binary tree are initially considered. The first element of ω is selected to be the first initial node for creating a cluster. The ψ is used for checking the sink node duplication on Φ. During the creation of a set of task cluster, a new sink node on the DAG G is determined and checked for duplication on ψ. If it is duplicated then the size of ψ and ω is not grew. But if a new sink node is not duplicated on ψ then a new sink node will be inserted onto ψ and ω, respectively. To find the started time of every nodes v, or e(v), excepts sink node on the DAG G, the Algorithm Create-E-Value( ) is used. And to create a set of task clustering Φ, the Algorithm Create-Cluster( ) is used. Initially, ψ and ω store sink node on DAG G, such as ω = {T8, T9, T10} of the DAG G on figure 1 then a sink node T8 is selected for creating C_i, and the ψ contains the same element as ω. After finishing the creation of a task cluster for sink node T8, the sink node T9 is selected for creating the next task cluster. The process is iterated until there is not any sink node on ω is left.

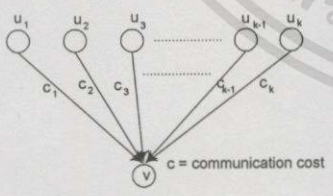


Figure 2 Immediate predecessors nodes point to node v

To find the started time of any node v, e(v), considers from the total value of communication cost, execution cost, and started time for each ancestor nodes, or $S(u_i, P) + \mu(u_i) + \lambda(u_i, v)$. Where each $S(u_i, P) + \mu(u_i) + \lambda(u_i, v)$ value of the ancestor node can be inserted onto the binary tree and searched by right→root→left or from the maximum to the minimum value as shown on figure 2. Figure 2 shows sorted $S(u_i, P) + \mu(u_i) + \lambda(u_i, v)$ value of ancestor from max to min, so the arc from u_1 is the maximum value and u_k is the minimum value, in decreasing order.

```
ALGORITHM Create-E-Value(v,G)
BEGIN
 If v is a source node return zero;
 C = {v};  m = 0;
 c = first arc(u_i , v) incoming ;where arc(u_i , v) ∈ψ
 e(v) = c;
 while( m < c)do
   C = C ∪ {u_i};
   m = GREEDY-SCHEDULE(C – {v});
   c = next arc(u_i , v) incoming ; where arc(u_i , v) ∈ψ
   e(v) = min{e(v), max(m,c) };
 endwhile
 return e(v);
END
```

The Algorithm Create-E-Value( ) can produce the minimum e(v) of any node v that will be considered from the maximum value of $S(u_i, P) + \mu(u_i) + \lambda(u_i, v)$ of any ancestors. Next, the Algorithm Create-E-Value( ) determines the next maximum value of the leave ancestors. The Algorithm Create-E-Value ( ) produces a minimum e(v) value if and only if m has a less value than c, and m is produced by the GREEDY-SCHEDULE(). The started time of the source node is always zero value. The GREEDY-SCHEDULE() function discussed has a communication cost equals to zero, as shown on figure 3. The GREEDY-SCHEDULE() schedules all ancestors u_i as those are on a single processor. Figure 3 represent all node u_i, which are on a frame, that are used by GREEDY-SCHEDULE(C-{v}) to create a scheduling of the ancestor u_i.

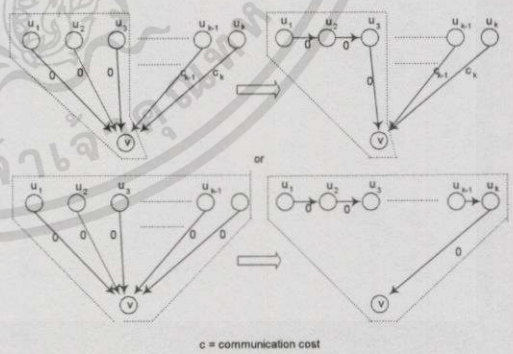


Figure 3 Finding started time e(v) for any node v

Theorem 1. Algorithm Create-E-Value(v,G) can find started time e(v) of any node or task of DAG G. The e(v) is possible minimum started time of node v that returns

0-7695-0589-2/00 $10.00 © 2000 IEEE

from Algorithm Create-E-Value(v,G). The source node v on a DAG has always e(v) equals to zero.

Proof The theorem is obviously true for source nodes. The proof is by induction on the depth of node v, that is

$$e(v^{l}) = \max_{i=1}^{m} (e(v_i^{l-1})) + \mu(v_i^{l-1})$$

Where m be a number of immediate predecessors, which are scheduled on a single processor, and l be a level of depth of node v on a DAG, where l is greater than 0.

induction base l = 1;

$$e(v^{1}) = \max_{i=1}^{m} (e(v_i^{0})) + \mu(v_i^{0})$$

induction hypothesis l = n;

$$e(v^{n}) = \max_{i=1}^{m} (e(v_i^{n-1})) + \mu(v_i^{n-1})$$

and implies that induction step l = n+1;

$$e(v^{n+1}) = \max_{i=1}^{m} (e(v_i^{(n+1)-1})) + \mu(v_i^{(n+1)-1})$$

The next step, to create a set of task clusters Φ by using e(v) of several nodes v that created from Algorithm Create-E-Value( ). To create Φ that is just initiated from a sink nodes on a DAG G, which are stored on ω. Each time when Φ is created, the size of ω and ψ are grew due to the sink node of a new task cluster is inserted onto ω and ψ, and may not be grew if the sink node of a new task cluster is the duplication on ψ. Algorithm Create-Cluster(v, G) is used to create Φ and yields C_i, which is inserted onto Φ, at the same time GET-TASK-CROSS-CLUSTER( ) function will get the sink node of a new task cluster, which is inserted onto ω and ψ.

```
ALGORITHM Create-Cluster(v,G)
BEGIN
 If v is a source node then return;
  C = {v};  m = 0;
  c = FIBANACCI-START-TIME(C);
  e(v) = c;
  while( m < c) do
   C = C ∪ {u};
   m = GREEDY-SCHEDULE(C-{v});
   //let (u,w) be an arc where u ∉C, w∈C, and
   //c=c(u,w)
   c = FIBANACCI-START-TIME(C);
   e(v) = min{e(v), max(m,c)};
  endwhile
GET-TASK-CROSS-CLUSTER();
 return e(v);
END
```

The concept of the AlgorithmCreate-Cluster(v,G) resembles the Algorithm Create-E-Value(v, G). The difference is that the Algorithm Create-E-Value(v, G) considers only ancestor's arcs that point to node v but the Algorithm Create-Cluster(v,G) considers every ancestor's arcs that point to every node of C_i. So that, to create C_i the Algorithm Create-Cluster(v, G) can be initialized by inserting the sink node of DAG G that is only on ω and ψ as the first node of C_i. A task cluster C_i is grew by inserting the ancestor node of node v. The result of the algorithm produces a minimum possible started time of sink node, e(v), and a task cluster C_i for node v. For example of Algorithm Create-Cluster(v, G), considers the sink node T8 as the first node of C_i and the first element of ω that is inserted on it. After the Algorithm Create-Cluster(v, G) is finished then a task cluster C_i will consist of a set of task nodes as the following C_i = {T8, T6, T3, T2}, at the same time ω and ψ grow as the following ω = {T8, T9, T10, T5, T1} and ψ = {T8, T9, T10, T5, T1} which are performed by GET-TASK-CROSS-CLUSTER( ). Therefore, the new nodes, T5 and T1, on ω are new sink nodes for creating the task cluster C_i.

As mentioned before, the Algorithm Create-Cluster performs the same as the Algorithm Create-E-Value except that the Algorithm Create-E-Value does not discuss a task cluster that only to find a minimum started time of node v. The step in creating C_i that consists of two stages as the following:

Stage 1) To find the makespan of a scheduling ancestor $\{u_1, u_2, \ldots, u_k\}$of node v, where every node $u_i \in C_i$ and node v is the first node of C_i by considering the started time and execution time of the ancestor node that are placed on one processor, i.e. $e(u_i)$ and $\mu(u_i)$ respectively. This scheduling can be done by the greedy algorithm to solve the problem of ancestor nodes $\{u_1, u_2, \ldots, u_k\}$. The relation between them can be expressed as :

$$e_C(v) \geq GREEDY\text{-}SCHEDULE(C - \{v\}) \qquad (1)$$

Stage 2) To consider all arcs that link from outside C_i, where $c(u_i,w) \in E$, $u_i \notin C$, and $w \in C$, if $c(u_i,w)=e(u_i)+\mu(u_i)+\lambda(u_i, v)$ then e(v) of C_i equals to

$$e_C(v) \geq FIBANACCI\text{-}START\text{-}TIME(C) \qquad (2)$$

where FIBANACCI-START-TIME( ) produces a maximum $c(u_i,w)$. From (1) and (2), a greedy algorithm yields the started time e(v) of C_i as follow :

$$e_C(v) \geq max(\ GREEDY\text{-}SCHEDULE(C-\{v\}),\ FIBANACCI\text{-}START\text{-}TIME(C)\ \} \qquad (3)$$

So that the minimum started time of node v, e(v), in C_i equals to

$$e(v) \geq min_C\{e_C(v)\} \qquad (4)$$

From (1)-(3) the minimum $e_C(v)$ of any node v by (4). Figure 4 shows Φ that is produced by the Algorithm Create-E-Value( ) and Create-Cluster( ) respectively.

Equations (1)-(4) and the Algorithm Create-Cluster explains a mechanism to create a cluster by considering from e(v). The whole algorithm uses greedy algorithm as the significant mechanism to create a cluster. The

0-7695-0589-2/00 $10.00 © 2000 IEEE

algorithm grows a task cluster one node at a time, if a new node is inserted onto a cluster then the started time of node v, e(v), on C_i can be reduced. But if a new node inserted onto a cluster can not potentially decrease the started time e(v), then the Algorithm Create-Cluster( ) is immediately stopped to grow the cluster. And the Algorithm will returns the minimum e(v) value that obtained. So that, it can be concluded that the creation of cluster C_i can be processed in two cases: the first case FIBANACCI-START-TIME(C) > GREEDY-SCHEDULE(C-{v}) represents that $c(u_i, w)$, which is an arc of node u_i from the outside cluster C_i, is greater than the makespan of all ancestors that is on C_i. Thus adding a new node onto C_i can potentially decrease the started time e(v), then a task cluster C_i grows. The second case FIBANACCI-START-TIME(C)≤GREEDY-SCHEDULE(C-{v}) indicates that the started time e(v) can not decrease, because $c(u_i, w)$ is less than or equal to the makespan of ancestors on C_i. So adding a new node u_i onto C_i can not decrease the started time e(v). The result of this stage makes the task cluster C_i stops growing and does not insert a new node onto task cluster C_i. Finally the Algorithm Create-Cluster( ) will return a started time e(v) for a sink node v as the Algorithm Create-E-Value( ) performs.

3.2 Scheduling of Φ

After the Algorithm Create-E-Value() and Create-Cluster( ) produces Φ then the next step is to define the scheduling of Φ, that is represented as Φ(G). The scheduling of Φ requires no any algorithm but only placing C_i onto each processor. Each task or node of C_i with a single processor will be executed by a sequence of started time e(v) with increasing value. Figure 5 shows Φ(G) of several C_i on Φ that are placed onto several processors. The Φ(G) shows that it can be produced makespan of scheduling of Φ with optimal value.

3.3 Time Complexity of Algorithm

To analyze time complexity of the algorithms used in this paper consists of time complexity of the Algorithm Create-E-Value() and Create-Cluster(). The Algorithm Create-E-Value() produces e(v) value by considering from the ancestor's arcs, which are only point to node v. The processing of the Algorithm Create-E-Value() performs on $S(u_i, P) + \mu(u_i) + \lambda(u_i, v)$, which are stored on a binary tree, therefore the time complexity of this algorithm equals to O(c lg c) for each node v_i, where c is a number of ancestor's arcs that point to node v and $c \in E$, $c << E$, and $v_i \in V$. The time complexity O(c lg c) shows an at most ancestor's arcs equal to c. So that the time complexity for every node v_i , except sink node on DAG G, that run in O(n(c lg c)), where n is every node v_i , except sink node on DAG G.

The running time of Algorithm Create-Cluster() consists of time complexity of GREEDY-SCHEDULE() and FIBANACCI-START-TIME( ). The GREEDY-SCHEDULE() function is maintained by 2-3 tree algorithm [1][4] that stores scheduling time of ancestor's node that is on C_i , the operation can be performed in O(|V| lg |V|) time. Because to insert a new node onto 2-3 tree that performed in O(lg|V|) time the same as required on the binary tree. The FIBANACCI-START-TIME() is maintained by fibanacci-heap algorithm [3] that is used for considering insert ancestor node onto C_i. The operation of FIBANACCI-START-TIME( ) that performs ATTRACK-MAX( ) and INCREASE-KEY()[3][1]. The ATTRACK-MAX() operation is to remove a maximum key node on a heap which is the most complicated process of the operation presented in fibanacci-heap. The INCREASE-KEY( ) operation is increasing the key of a node in a heap in O(1) amortized time.

The ATTRACK-MAX( ) runs at most V times by considers the value of $c(u_i,w)$. The INCREASE-KEY( ) is performed at most E times. When the operation of both ATTRACK-MAX() and INCREASE-KEY() can be performed in O(lg |V|) amortized time and O(1) amortized time, respectively. So the operation of FIBANACCI-START-TIME runs in O(|V| lg|V| + |E|) time. Therefore, the Algorithm Create-Cluster( ) runs in O(|V| lg|V| + |E|) + O(|V| lg|V|) for each cluster C_i. So that, The total time complexity of to create Φ(G) equals to O(m(|V| lg |V| + |E|)), where m is a number of clusters C_i , V and E is a number of nodes and edges on a DAG G, respectively, and $1 \le m < V$.

4. Result of the experiments

Figure 4 shows Φ that is produced by the Algorithm Create-E-Value( ) and Create-Cluster( ), respectively. It shows task duplication between clusters. Figure 5 shows a scheduling of Φ, or Φ(G), on several processors and the result shows that the started time of each node are changed. In addition , the Φ(G) has a less makespan than DAG G of figure 1. Therefore, A purpose of the GTCS algorithm can be achieved in reducing the execution time of parallel program, as expected.

Theorem 2. For a DAG with arbitrary granularity (g(G)≥0), we have

$$M_{opt} \le M_{GTCS} \le (1 + \frac{1}{1 + g(G)}) M_{opt}$$

Where M_{opt} be a makespan schedule of an optimal clustering of a DAG G, and M_{GTCS} be a makespan schedule of Φ(G).

Proof If M_{opt} is at least $max(e(v_i) + \mu(v_i))$ for all sink node v of C_i, $C_i \in \Phi$. From lemma 1[1], so that
and thus

$$M_{GTCS} \le \max_{i=1}^{m}\{(1+\frac{1}{1+g(G)})e(v_i)+\mu(v_i)\}$$

$$\le \max_{i=1}^{m}\{(1+\frac{1}{1+g(G)})[e(v_i)+\mu(v_i)]\}$$

$$\le (1+\frac{1}{1+g(G)})\max_{i=1}^{m}\{e(v_i)+\mu(v_i)\}$$

0-7695-0589-2/00 $10.00

$$\leq (1 + \frac{1}{1 + g(G)}) M_{opt}$$

5. Comparison with other algorithm

Figure 6 shows a set of task clustering Φ that is produced by the algorithm in [1] which produces a greater number of task clusters than GTCS algorithms. The scheduling of Φ(G) of [1] algorithm gives the same makespan as the GTCS. The Φ produced by [1] algorithm shows that some C_i is not used, while the Φ produced in the GTCS, all the C_i are used. So that, the GTCS algorithm needs only the time complexity for creating the clusters that are only uses, and the GTCS has not garbage running time to create a task cluster.

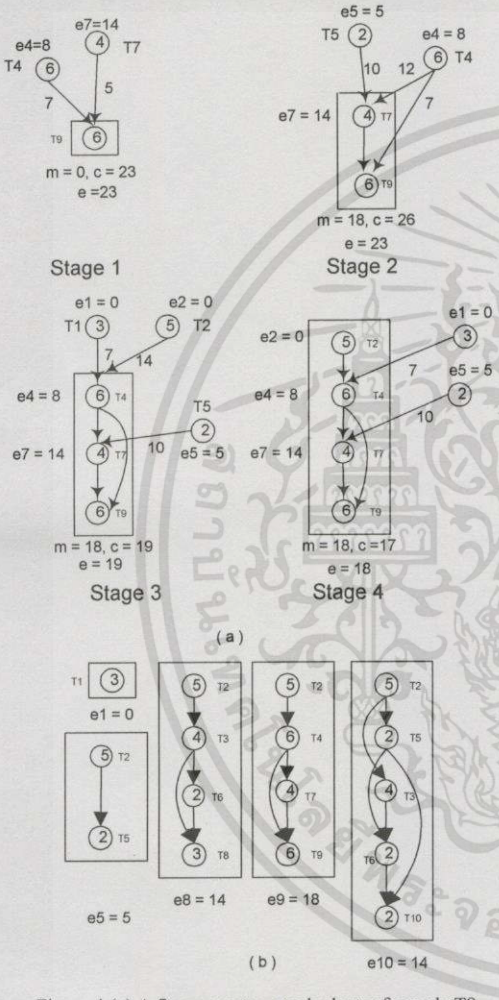


Figure 4 (a) A Stage creates a task cluster for node T9
(b) A Φ produced by GTCS

6. Conclusion

The GTCS algorithm can create an efficient set of task clusters, Φ, and has a time complexity equals to O(m(|V| lg|V| + |E|)), where m is a number of task clusters ($1 \leq m < V$) that produced from the GTCS . The efficiency of the GTCS algorithm is compared with [1], which has a time complexity equals to O(|V|(|V|lg|V| + |E|)). So if m equals to 0.5V, where V is a number of nodes on DAG G, the GTCS algorithm uses only a half time complexity of [1] algorithm .

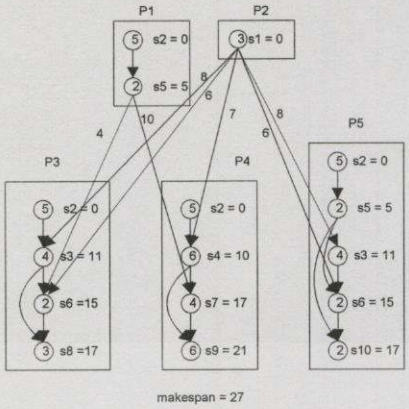


Figure 5 A Φ(G) is produced by GTCS and [1] algorithms

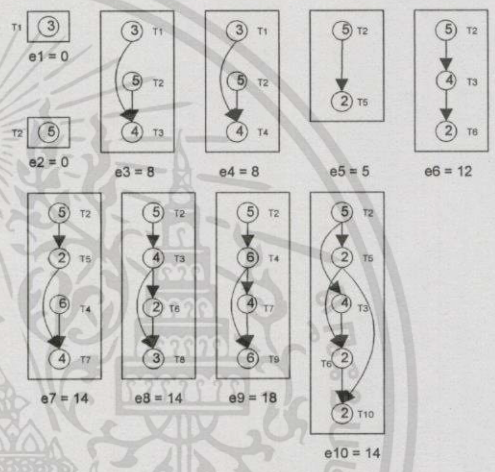


Figure 6 The Φ is produced by [1] algorithms

7. Reference

[1]. M.A. Palis, J.C. Liou and D.S.L. Wei, "Task Clustering and Scheduling for Distributed Memory Parallel Architectures," IEEE Trans. Parallel and Distributed Systems, vol.7, no.1, Jan. 1996.

[2]. A. Gerasoulis and T. Yang, "On the granularity and clustering of directed acyclic task graphs," IEEE Trans. Parallel and Distributed Systems, vol.4, no.6, pp. 686-701, June 1993.

[3]. M.L. Fredman and R.E. Tarjan, "Fibanacci heaps, and their uses in improved network optimization algorithms," Proc. 25th Annual Symposium on Foundations of Computer Science, pp. 338-346, Oct. 1984.

[4]. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, The Design and Analysis of Computer Algorithms, Reading, Mass.: Addison-Wesley, 1974.

[5] V. Sarkar, Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors. Cambrige, Mass.: MIT Press, 1989.

0-7695-0589-2/00 $10.00 © 2000 IEEE