การจัดงานด้วยพื้นฐาน ซีพียู บน PVM

CPU BASED SCHEDULING ON PVM

เธียรชัย พัฒนศิริเวทิน

THIANCHAI PATANASIRIVETIN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาเทคโนโลยีสารสนเทศ

บัณฑิตวิทยาลัย

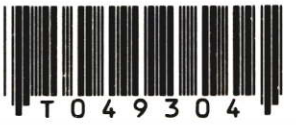
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2546

ISBN 974-524-857-9

การจัดงานด้วยพื้นฐาน ซีพียู บน PVM

CPU BASED SCHEDULING ON PVM



เธียรชัย พัฒนศิริเวทิน

THIANCHAI PATANASIRIVETIN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาเทคโนโลยีสารสนเทศ

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2546

ISBN 974-324-857-9

CPU BASED SCHEDULING ON PVM

THIANCHAI PATANASIRIVETIN

A THESIS SUBMITTED IN PATIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2003

ISBN 974-324-857-9

หัวข้อวิทยานิพนธ์	**การจัดงานด้วยพื้นฐาน ซีพียู บน PVM**
ชื่อนักศึกษา	**นาย เธียรชัย พัฒนศิริเวทิน**
รหัสประจำตัว	42067018
ปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา	เทคโนโลยีสารสนเทศ
พ.ศ.	2546
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ. ดร. วิเชียร เปรมชัยสวัสดิ์
อาจารย์ผู้ควบคุมวิทยานิพนธ์ร่วม	ผศ. ดร. วรพจน์ กรีสุระเดช

บทคัดย่อ

ความจำเป็นในงานคำนวณทางวิทยาศาสตร์และวิศวกรรมศาสตร์ที่ต้องใช้กำลังในการคำนวณของเครื่องคอมพิวเตอร์อย่างมหาศาลของเครื่องซุปเปอร์คอมพิวเตอร์ราคาสูง จึงเกิดแนวคิดของการประมวลผลแนวใหม่ที่โดยแบ่งงานหลักออกเป็นงานย่อยและกระจายไปยังเครื่องคอมพิวเตอร์หลายๆเครื่องบนระบบที่เชื่อมต่อผ่านเครือข่าย, เป็นแนวทางในการสร้างเครื่องซุปเปอร์คอมพิวเตอร์ที่มีราคาถูกกว่าและสามารถใช้งานได้จริง แต่ปัญหาและอุปสรรค์สำคัญของการประมวลแบบคู่ขนานคือ "ภาระงานไม่สมดุลย" ที่เกิดขึ้นเนื่องจากสภาพแวดล้อมที่แตกต่างกันของเครื่องคอมพิวเตอร์บนเครือข่าย เช่น ความเร็วซีพียู, ขนาดและความเร็วของหน่วยความจำของแต่ละเครื่องที่แตกต่างกันเป็นต้น ในกรณีที่แย่สุดอาจทำให้การทำงานแบบคู่ขนานประมวลผลได้ช้ากว่าการทำงานแบบเรียงลำดับบนเครื่องๆเดียว

Thesis Title	CPU Based Scheduling on PVM
Student	Mr. Thianchai Patanasirivatin
Student ID.	42067018
Degree	Master of Science
Programme	Information Technology
Year	2003
Thesis Advisor	Assoc.Prof.Dr.Wichian Premchaiswadi
Co-Thesis Advisor	Asst.Prof.Dr. Dr. Worapoj Kreesuradej

ABSTRACT

Computing in scientific and engineering need high speed computers. The computing is usually performed by using supercomputers which are very expensive. In the modern computing concept, A large problem is divided into many tasks and distributed to machines which connected via network. In idea is to build the supercomputer from cheaper computers that can really work. However, there is the main problem of using the system called "load imbalance". The load imbalance occurs because the specification of machines such as CPU speed and main memory size are different. In the worst case, the parallel processing time is slower than that of a single host. The problem appears on a heterogeneous environment of a PVM system. This thesis purposes dynamic scheduling schemes on PVM, namely: AWFGSS, AWGSS and AWFS to solve the problem of load imbalance. The experimental results show that the propose schemes give the correct result and having speed up factor better than that of other methods. The scheme is shown by comparing the experimental results with other methods.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ไม่อาจจะสำเร็จลุล่วงได้ด้วยดีถ้าปราศจากคำปรึกษาและแนวทางการทำงานวิจัยจาก รศ.ดร.วิเชียร เปรมชัยสวัสดิ์ อาจารย์ผู้ควบคุมวิทยานิพนธ์ ผู้วิจัยรู้สึกซาบซึ้งและกราบขอบพระคุณในความอนุเคราะห์และสนับสนุนเป็นอย่างสูง

กราบขอบพระคุณด้วยความเคารพและศรัทธาอย่างสูงสุด ด้วยพระคุณจากบิดา, มารดา และพี่ๆในครอบครัว พัฒนศิริเวทิน ทุกๆท่านที่ให้ความรัก, ความเข้าใจและให้กำลังใจแก่ผู้วิจัยด้วยดีเสมอมา

และขอขอบพระคุณคณาจารย์ทุกท่าน ผู้ประสิทธิประสาทวิชาความรู้ ทำให้สามารถนำความรู้มาใช้ในงานวิจัย และขอบคุณบุคลากรคณะเทคโนโลยีสารสนเทศ, บัณฑิตวิทยาลัยทุกท่านที่อำนวยความสะดวกและประสานงานต่าง ๆ ด้วยดี

สุดท้ายนี้ขอขอบคุณ คุณสุธาสินี ฉิมเล็ก ที่ให้คำแนะนำและอนุญาติให้ผู้วิจัยทำการดัดแปลงและแก้ไขซอฟท์แวร์สืบค้นข้อมูลรูปภาพเพื่อนำมาเป็นแอ็พพิเคชั่นทดสอบในงานวิจัยนี้ และขอขอบคุณ คุณกอบชัย นิยมธรรม, คุณผุสดี พรผล, คุณพิบูลย์ เทพบุตร รวมถึงเพื่อนนักศึกษา รุ่น 7.1 ทุกๆท่านที่เป็นกำลังใจและคอยให้ความช่วยเหลือและสนับสนุนการทำวิจัยด้วยดีตลอดมา

เธียรชัย พัฒนศิริเวทิน

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

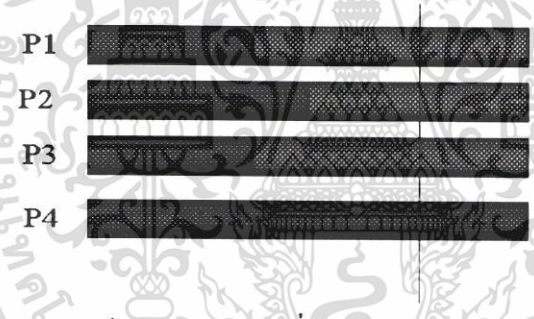
สารบัญรูป (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

PVM (Parallel Vritual Machine) เริ่มพัฒนาตั้งแต่ปี 1989 โดยความร่วมมือระหว่างห้องแล็บวิจัย ORNL (Oak Ridge National Laboratory) และ University of Tennessee ที่ Knoxville เพื่อสร้างเครื่องซุปเปอร์คอมพิวเตอร์ราคาถูกและความเร็วสูงจากเครื่อง UNIX Workstation หลายๆ เครื่องที่เชื่อมต่อผ่านระบบเครือข่ายเดียวกัน เสมือนเป็นเครื่องจักรที่มีกำลังในการคำนวณมหาศาล ต่อมาได้มีการพัฒนากับเครื่องคอมพิวเตอร์ส่วนบุคคลที่มีราคาถูกกว่าจนกระทั่งในปัจจุบันเทคโนโลยีของคอมพิวเตอร์ส่วนบุคคลพัฒนาไปอย่างรวดเร็วดังนั้นในการทำงานร่วมกันของเครื่องรุ่นใหม่ๆกับเครื่องรุ่นเก่าที่มีอยู่เดิมแล้วปัญหาที่มักจะเกิดขึ้นตามมาก็คือ ภาระงานไม่สมดุลย (Load Imbalance)



รูปที่ 1.1 การเกิดปัญหาภาระไม่สมดุลยบนเครื่อง 4 โพรเซสเซอร์

ปัญหาดังกล่าวเกิดขึ้นเนื่องจากแต่ละเครื่องบนระบบ PVM นั้นได้ถูกกำหนดความเร็วไว้เท่าเทียมกันหมด แต่ในความเป็นจริงความเร็วของซีพียูแต่ละเครื่องในระบบไม่เท่ากัน รวมถึงการออกแบบระบบจัดงานบน PVM นั้นจะใช้การวนรอบ (Round Robin Scheduling) โดยเรียงลำดับตามตารางเครื่อง (Host Table) ในระบบ

ปัญหานี้ส่งผลกระทบต่อระบบในแง่ประสิทธิภาพการประมวลผลคู่ขนานอย่างมากเนื่องจากเครื่องที่เร็วสุดในระบบจะต้องรอเครื่องที่ช้าที่สุดทำงานเสร็จก่อนจึงจะเสร็จสิ้นการทำงาน ดังนั้นเพื่อลดปัญหาดังกล่าวจึงจำเป็นต้องมีการพัฒนาอัลกอลิซึมสำหรับจัดงานที่เหมาะสมเพื่อให้แต่ละเครื่องสามารถทำงานเสร็จได้อย่างพร้อมๆกันหรือเรียกว่าการทำภาระงานสมดุลยโดยอัลกอลิทึมที่ใช้สำหรับจัดงานดังกล่าวเรียกว่า **"อัลกอลิทึมการจัดงานแบบพลวัต (Dynamic Scheduling)"**

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

1.2.1 ศึกษาเทคนิคการจัดงานแบบพลวัตเพื่อนำมาแก้ปัญหาภาระงานที่ไม่สมดุลยบนระบบ

1.2.2 เพื่อเป็นแนวทางการศึกษา, ออกแบบและพัฒนาระบบจัดงานสำหรับแอ็พพิเคชั่นที่เกี่ยวข้องทำงานได้อย่างมีประสิทธิภาพบนระบบที่มีสภาพแวดล้อมที่แตกต่าง

1.2.3 เพื่อนำการจัดงานแบบพลวัตมาประยุกต์ใช้บนระบบ PVM ได้อย่างเหมาะสม

1.3 สมมุติฐานในการศึกษา

แม้นว่า PVM จะทำให้เกิดการทำงานร่วมกันของเครื่องคอมพิวเตอร์หลายแพลตฟอร์ม แต่ปัญหาภาระงานไม่สมดุลยก็เป็นสิ่งที่เกิดขึ้นตามมาเช่นกัน ทั้งนี้เนื่องจากความไม่เหมาะสมในการเลือกใช้อัลกอลิทึมสำหรับการจัดแบ่งงานบนระบบเสมือนคู่ขนาน (Parallel Virtual Machine)

อัลกอลิซึมการจัดงานพลวัตคืออัลกอลิทึมการจัดแบ่งงานให้แต่ละโพรเซสเซอร์บนระบบในช่วงรันทั้งนี้เพื่อให้เกิดการปรับภาระสมดุลย์บนเครื่องระบบหลายโพรเซสเซอร์ที่ใช้พื้นที่หน่วยความจำร่วมกัน (Share Memory Multiprocessor System) ดังนั้นการนำอัลกอลิทึมเหล่านี้มาประยุกต์ใช้บนเครื่องระบบส่งผ่านแมสเส็จ (Message Passing Machines) โดยเฉพาะอย่างยิ่งบนระบบที่มีสภาพแวดล้อมที่แตกต่าง (Heterogeneous) ซึ่งแต่ละเครื่องบนระบบมีความสามารถในการรองรับภาระงานไม่เท่ากัน

โดยทั่วไปค่าความเร็วซีพียู (CPU Speed) จะเป็นพารามิเตอร์ที่บ่งบอกถึงความสามารถสูงสุดในการประมวลผลของเครื่องนั้นๆได้ดีที่สุดและในความเป็นจริงประสิทธิภาพการคำนวณของเครื่องคอมพิวเตอร์มักจะนิยมวัดผลเป็น MFLOPS (Mega Floating Point Operations) ซึ่งจะบ่งบอกความสามารถในการคำนวณของแต่ละเครื่องได้อย่างชัดเจน ซึ่งหากจะพิจารณาถึงกำลังการคำนวณสูงสุด (Peak of Computing Power) ของแต่ละเครื่องก็สามารถที่จะประเมินด้วยวิธีง่ายๆคือ หาว่าใน 1 Clock Cycle ของเครื่องนั้นๆสามารถประมวลผลเลขทศนิยมได้กี่ครั้ง เช่น ถ้า 1 Clock Cycle บนเครื่องคอมพิวเตอร์นั้นๆสามารถประมวลผลเลขทศนิยมได้ 3 ครั้ง แสดงว่ากำลังในการคำนวณของเครื่องนั้นก็คือ 3 เท่าของความเร็วซีพียู (3 x CPU Speed) นั้นเอง

สำหรับบนเครื่อง PC แล้วโดยทั่วไป 1 Clock Cycle มักจะประมวลผลกับเลขทศนิยมได้ทีละ 1 flop (ยกเว้นในกรณีที่เครื่องนั้นเป็น DUAL CPU โดยอิงสถาปัตยกรรม CPU แบบ 32 บิต) ดังนั้นในงานวิจัยนี้ตั้งสมมุติฐานว่าสำหรับ PC คลัสเตอริ่งทุกเครื่องบนระบบแตกต่างที่ทดสอบนี้จะมีกำลังในการคำนวณสูงสุดเท่ากับความเร็วซีพียูของเครื่องนั้นๆ (โดยยังไม่พิจารณาเทคโนโลยีของ CPU สมัยใหม่ที่สามารถให้ 1 Clock Cycle ทำงานได้มากกว่า 1 flop)

ในงานวิจัยนี้จะนำเสนอการประยุกต์ระบบการจัดงานพลวัตบนระบบส่งผ่านแมสเส็จ PVM โดยจะนำแนวคิดของแต่ละอัลกอริทึมการจัดงานพลวัตแบบเดิมมาออกแบบ/พัฒนาอัลกอริทึมการ

จัดงานพลวัตใหม่ให้สัมพันธ์กับความเร็วแต่ละเครื่องบนระบบแทน รวมถึงการวิเคราะห์ความเหมาะสมของแต่ละอัลกอริทึมเพื่อนำไปใช้แก้ปัญหาภาระงานไม่สมดุลย์ที่เกิดขึ้นบนระบบ

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 ได้ศึกษาถึงงานวิจัยที่เกี่ยวข้องกับการจัดงานแบบพลวัตบนเพื่อนำมาแก้ปัญหาภาระงานไม่สมดุลยบนระบบ

1.4.2 ได้ศึกษาคุณลักษณะสำคัญต่างๆของระบบ PVM เพื่อสร้างเครื่องซุปเปอร์คอมพิวเตอร์ที่มีราคาถูกและมีประสิทธิภาพ

1.4.3 เพื่อพัฒนาระบบจัดงานแบบพลวัตบนเครื่อง PVM ที่มีสภาพแวดล้อมที่แตกต่างกัน ทั้งนี้เพื่อให้เกิดการใช้งานทรัพยากรบนระบบได้อย่างมีประสิทธิภาพ

1.4.4 สามารถนำแนวคิดการจัดงานแบบพลวัตรวมถึงอัลกอริทึมประยุกต์ที่ออกแบบไว้ทั้งสามแบบไปใช้งานได้จริงกับหลายๆแอ็พพิเคชั่นที่เกี่ยวข้อง

1.5 แผนการดำเนินงานวิจัย

1.5.1 ศึกษาจุดอ่อนสำคัญที่ทำให้ระบบ PVM เกิดปัญหาภาระงานไม่สมดุลย

1.5.2 ศึกษาและออกแบบการทำงานของมิดเด็ลแวร์เพื่อสนับสนุนการทำภาระงานสมดุลย์พลวัตบนระบบ PVM

1.5.3 วิเคราะห์, ออกแบบและพัฒนาอัลกอลิซึมการจัดงานแบบพลวัตที่เหมาะสมกับระบบ

1.5.4 ออกแบบการทดลองและพัฒนาโปรแกรมสำหรับการทดสอบเพื่อทดสอบประสิทธิภาพของแต่ละอัลกอลิซึมที่ออกแบบ

1.5.5 ทดสอบและเก็บผลการทดลองพร้อมวิเคราะห์ข้อดีและข้อเสีย

1.5.6 สรุปผลการดำเนินการ และข้อเสนอแนะในงานวิจัย พร้อมรวบรวมจัดทำเอกสารนำเสนอโดยได้กำหนดระยะเวลาในการดำเนินการแต่ละขั้นตอนดังนี้

ขั้นตอนการทำงาน	ระยะเวลาการดำเนินงาน (เดือนที่)											
	1	2	3	4	5	6	7	8	9	10	11	12
1. ศึกษาจุดอ่อนสำคัญที่ทำให้ระบบ PVM เกิดปัญหาภาระงานไม่สมดุลย	▒											
2. ศึกษาบทความและเกี่ยวข้องกับงานวิจัย		▒	▒									

3. ศึกษาและออกแบบการทำงานของมิดเด็ลแวร์เพื่อสนับสนุนการทำภาระงานสมดุลยพลวัตบนระบบ PVM												
4. วิเคราะห์ , ออกแบบและพัฒนาอัลกอลิซึมการจัดงานพลวัตที่เหมาะสมกับระบบ												
5. ออกแบบการทดลองและพัฒนาโปรแกรมสำหรับการทดสอบเพื่อทดสอบประสิทธิภาพของแต่ละอัลกอลิซึมที่ออกแบบไว้												
6. ทดสอบและเก็บผลการทดลองพร้อมวิเคราะห์ข้อดีและข้อเสีย												
7. สรุปผลการดำเนินการ และข้อเสนอแนะในงานวิจัย พร้อมรวบรวมจัดทำเอกสารนำเสนอ												

1.6 โครงสร้างของวิทยานิพนธ์

โครงสร้างของวิทยานิพนธ์ ประกอบด้วย 5ส่วนคือ

บทที่ 1 บทนำ จะกล่าวถึงความสำคัญและที่มาของปัญหาในงานวิจัย รวมถึงสมมุติฐานที่ใช้ในงานวิจัย

บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง บทนี้จะกล่าวถึง ทฤษฎีต่างๆที่เกี่ยวข้องกับระบบ PVM และการจัดงานพลวัต รวมถึงนำเสนองานวิจัยที่เกี่ยวข้อง

บทที่ 3 การออกแบบและประยุกต์การจัดงานพลวัตบนระบบ PVM บทนี้จะกล่าวถึงการแนวทางการประยุกต์ระบบจัดงานพลวัตบน PVM รวมถึงแนวคิดของการออกแบบและดัดแปลงอัลกอลิซึมการจัดงานพลวัตที่เหมาะสมกับการประยุกต์ใช้งานจริง

บทที่ 4 การประยุกต์งานจริงกับแอ็พพิเคชั่นโปรแกรมสืบค้นข้อมูลรูปภาพแบบคู่ขนาน

บทที่ 5 การทดลองและผลการทดลอง บทนี้จะกล่าวถึงวิธีการกระบวนการของการออกแบบและทดลองต่างๆพร้อมแสดงผลลัพธ์จากการทดลองในงานวิจัยนี้

บทที่ 6 สรุปผลและแนวทางในการพัฒนาในอนาคต

คำจำกัดความที่สำคัญ (Terminology)

Chunks	คือขนาดภาระงาน หรือ จำนวนรอบของลูปโปรแกรมที่ถูกจัดแบ่งตามอัลกอริทึมการจัดงานพลวัต
Collective communication	คือการจัดการส่งเมสเส็จที่มีผู้รับและผู้ส่งมากกว่า 1 คู่บนระบบ เช่น การติดต่อสื่อสารภายในกลุ่ม
Collective operation	คือการแพร่กระจายข้อมูลให้แก่งานภายในกลุ่มเดียวกัน
Daemond	คือโปรแกรมที่คอยทำงานอยู่เบื้องหลังบนระบบ
Data Dependency	คือการเข้าถึงข้อมูลตามลำดับซึ่งไม่สามารถหลีกเลี่ยงได้
DOALL	คือประเภทของลูปที่สามารถทำงานแบบคู่ขนานได้โดยที่ไม่เกิดปัญหา ความพึ่งพาข้อมูลในแต่ละรอบที่ทำซ้ำๆกัน
HOST	คือเครื่องคอมพิวเตอร์บนระบบ
IPC	เป็นคำย่อมาจาก Inter Process Communication
Load Balancing	คือการทำสมดุลย์เพื่อให้แต่ละโพรเซสเซอร์บนระบบทำงานเสร็จอย่างพร้อมเพียงกัน
Message	คือลำดับของการจัดส่งข้อมูลระหว่างแต่ละงาน
Message Passing	คือรูปแบบการติดต่อสื่อสารระหว่างโพรเซสด้วยวิธีการจัดส่งเมสเส็จที่ให้ฝ่ายหนึ่งทำหน้าที่จัดส่ง ส่วนอีกฝ่ายหนึ่งคอยรอรับ
Process	คือหน่วยที่ทำงานบนเครื่องต่างๆซึ่งมี 3 องค์ประกอบหลักคือ โปรแกรม, ข้อมูล และ สแต็ก
Task	คือโพรเซส หรือ งานบนระบบ PVM
Task Granularity	คืออัตราส่วนความสัมพันธ์ระหว่างเวลารวมของการประมวลผลบนแต่ละเครื่องคอมพิวเตอร์กับเวลาที่สูญเสียไปจากการติดต่อสื่อสาร
Tasks Pool	คือแหล่งเก็บงานระบบชั่วคราวเพื่อรอการกระจายไปตามเครื่อง/โพรเซสเซอร์ต่างๆบนระบบ
TID	ย่อมาจาก Task Identifier ใช้ระบุตัวงานบนระบบเสมือน
Virtual Machine	ระบบเสมือนหรือก็คือระบบที่สร้างให้เกิดความเป็นหนึ่งเดียวกันจากหลายๆ Host ที่ใช้ในการประมวลผล
Workload	คือภาระงานทั้งหมดที่เกิดขึ้นจากการประมวลผลบนเครื่องเดียว

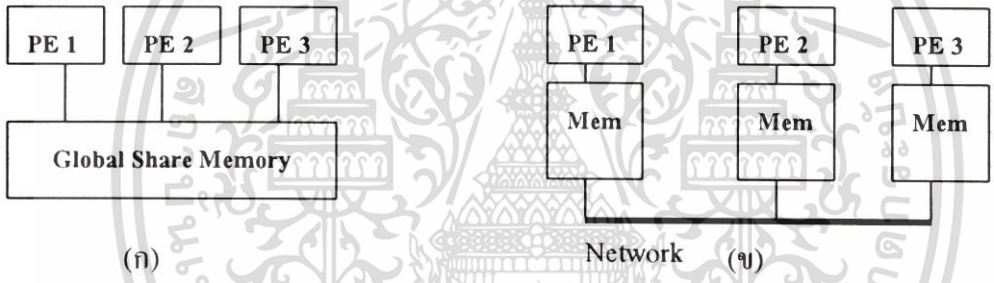
บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 หลักการพื้นฐานของการคำนวณแบบคู่ขนาน (Basic Parallel Computing)

2.1.1 สถาปัตยกรรมระบบคู่ขนาน

ในการจำแนกประเภทสถาปัตยกรรมระบบคู่ขนานสามารถทำได้หลายแบบทั้งนี้ขึ้นอยู่กับการพิจารณาองค์ประกอบใดเป็นสำคัญ ตัวอย่างเช่นถ้าใช้ความสัมพันธ์ระหว่าง PE (Processing Equipment) กับ หน่วยความจำของเครื่องในระบบ ก็จะแยกได้เป็น 2 ประเภทหลักๆดังนี้

- Share Memory คือ ระบบที่แต่ละ PE จะใช้หน่วยความจำร่วมกัน
- Distributed Memory คือระบบที่แต่ละ PE จะมีหน่วยความจำเป็นของตัวเอง



รูปที่ 2.1 (ก) สถาปัตยกรรมแบบ Share Memory (ข) สถาปัตยกรรมแบบ Distributed Memory

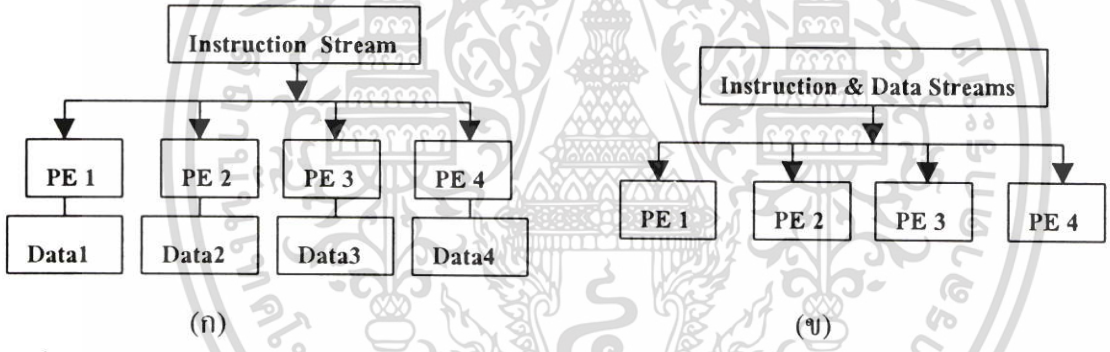
ระบบในรูป (ข) แต่ละ PE จะใช้ระบบเครือข่ายเป็นช่องทางการสื่อสารและหากพิจารณาจากความสัมพันธ์ระหว่าง ชุดคำสั่ง (Instruction Stream) และ ชุดข้อมูล (Data Stream) ที่ใช้ประมวลผลตามการจัดหมวดหมู่ของ Flynn สามารถจำแนก 4 ประเภทหลักๆดังแสดงในรูปที่ 2.2

Flynn 's Classification	
SISD	**MISD**
SIMD	**MIMD**

รูปที่ 2.2 แยกประเภทสถาปัตยกรรมตามแนวคิดของ Flynn

- SISD (Single Instruction Single Data) คือสถาปัตยกรรมระบบที่มีชุดคำสั่งเดียว เพื่อประมวลผลกับข้อมูลชุดเดียวกัน หรือก็คือระบบ UNI-Processor System เช่น บนเครื่อง PC นั้นเอง
- MISD (Multiple Instruction Single Data) คือสถาปัตยกรรมระบบที่ประกอบด้วยหลายชุดคำสั่งเพื่อประมวลผลกับข้อมูลชุดเดียวกัน อย่างไรก็ตามสถาปัตยกรรมระบบแบบนี้นักวิจัยบางกลุ่มมองว่าไม่น่าจะทำคู่ขนานได้ แต่บางกลุ่มให้มองว่าการทำงานแบบ Pipeline ก็เป็นสถาปัตยกรรมระบบนี้เช่นกัน
- SIMD (Single Instruction Multiple Data) คือสถาปัตยกรรมที่ประมวลผลชุดคำสั่งเดียวกันแต่ต่างชุดข้อมูลกัน
- MIMD (Multiple Instruction Multiple Data) คือสถาปัตยกรรมระบบที่ประมวลผลกับชุดข้อมูลและชุดคำสั่งที่แตกต่างกัน

สถาปัตยกรรมแบบ SIMD และ MIMD เป็นสถาปัตยกรรมระบบที่ได้รับความนิยมมากของระบบประมวลผลแบบคู่ขนานดังแสดงในรูปที่ 2.3



รูปที่ 2.3 (ก) แสดงสถาปัตยกรรมแบบ SIMD (ข) แบบ MIMD

ตัวอย่างเครื่องระบบ MIMD เช่น IBM SP2, SGI Origin , HP CONVEX SPP, Paragon, CRAY-C90, CRAY-T3D และ CM-5 เป็นต้น ส่วนเครื่องระบบ SIMD บางครั้งอาจเรียกว่าเครื่องอาเรย์โพรเซสเซอร์ (Array Processor) ตัวอย่างเครื่องระบบนี้เช่น X-MP, Y-MP,CM-2 และ MasPar เป็นต้น

ในปัจจุบันเทคโนโลยีระบบเครือข่าย (Network) พัฒนาไปอย่างรวดเร็วทำให้เกิดแนวคิดของการนำเครื่องสถาปัตยกรรมแบบ SISD เช่น PC หลายๆเครื่องเชื่อมต่อผ่านระบบเครือข่ายความเร็วสูงเรียกว่า "คลัสเตอร์ (Cluster)" หรือบางครั้งเรียกว่าระบบหลายเครื่องคอมพิวเตอร์ (Multicomputer) โดยจุดเด่นที่สำคัญของระบบนี้ก็คือ ความสามารถในการปรับ ย่อ/ขยายระบบ (Scalable) ตามความเหมาะสมกับการใช้งาน อย่างไรก็ตามระบบนี้อาจยังมีประสิทธิภาพไม่เทียบเท่ากับระบบอื่นๆในเชิงพาณิชย์ (commercial) แต่ก็ถือได้ว่าระบบนี้มีราคาถูกกว่ามาก

2.1.2 ซอฟท์แวร์ระบบคู่ขนาน (Parallel System Software)

เนื่องจากในปัจจุบันยังไม่มีตัวแปรภาษาแบบคู่ขนาน (Parallel compiler) ใดๆที่จะวิเคราะห์โปรแกรมได้สมบูรณ์แบบเหมือนผู้เชี่ยวชาญลงมือปฏิบัติเอง การออกแบบและพัฒนาซอฟท์แวร์ระบบจึงเป็นสิ่งจำเป็นเพื่อให้สามารถทำงานแบบคู่ขนานได้กับระบบใดระบบหนึ่งโดยเฉพาะ ซึ่งสามารถจำแนกได้ 4 ประเภทหลักๆดังนี้

- Share Memory เช่น UNIX IPC, Pthread, OpenMP
- Message Passing เช่น PVM,MPI
- Object Oriented เช่น CORBA, Microsoft DCOM
- Procedural เช่น RPC, Java RMI

ซอฟท์แวร์ระบบ Share Memory ทำงานบนเครื่องหลายโพรเซสเซอร์ที่ใช้หน่วยความจำระบบร่วมกัน (SMP – Symatic Multi Processor) โดยซอฟท์แวร์ประเภทนี้อนุญาตให้เกิดการพัฒนาแอ็พพิเคชั่นโปรแกรมระบบได้ 2 แบบใหญ่ๆ คือ หลายโพรเซส (Multi-process programming) และ หลายเทรด (Multi -Threaded programming) ซึ่งมีการกำหนดพื้นที่หน่วยความจำที่ใช้ร่วมกันสำหรับการแลกเปลี่ยนข้อมูลบนระบบ รวมถึงการกำหนดจังหวะการทำงาน (Synchronization) ด้วย semaphores, Mutex และ locks เป็นต้น ในปัจจุบันซอฟท์แวร์สำหรับพัฒนาแอ็พพิเคชั่นระบบประเภทนี้อย่างเช่น OpenMP มีความสามารถเกือบเท่า Parallel Compiler (หลายคนจัดว่า OPENMP เป็น Semi Compiler ของระบบ) โดยอาศัยการกำหนด Directive ของภาษา C แทนทำให้สะดวกในการพัฒนาแอ็พพิเคชั่นโปรแกรม

ซอฟท์แวร์ระบบประเภท Message Passing ปัจจุบันมี 2 ชนิดที่ได้รับความนิยมมากที่สุดในโลกคือ PVM และ MPI โดยหลักการของซอฟท์แวร์นี้คือโพรเซสต้นทางจะจัดส่งเมสเส็จไปยังโพรเซสปลายทางที่อยู่บนเครื่องเดียวกันหรือต่างกัน ซึ่งการพัฒนาแอ็พพิเคชั่นด้วยซอฟท์แวร์ประเภทนี้ผู้พัฒนาโปรแกรมจำเป็นจะต้องจัดการเกี่ยวกับติดต่อสื่อสารเองทั้งหมด ซึ่งมีอยู่ 2 รูปแบบคือ Blocking (ผู้ส่งและผู้รับคอยรอรับเมสเส็จที่ถูกส่งมาอยู่ตลอด) และ Non Blocking (ผู้รับไม่ต้องคอยรอรับเมสเส็จอยู่ตลอดจนเมสเส็จนั้นมาถึง) ในปัจจุบันซอฟท์แวร์ประเภทนี้ได้นำหลักการทำงานร่วมกันเป็นกลุ่มโพรเซส (Process Group) บนระบบเพื่อความสะดวกในการจัดการติดต่อสื่อสารแบบกลุ่มโดยเฉพาะบนระบบ "คลัสเตอร์ริ่ง"

Object Oriented เป็นการนำแนวคิดเชิงวัตถุมาประยุกต์ใช้กับการพัฒนาโปรแกรมรูปแบบไคล์เอ็นท์ / เซริฟ์เวอร์ (client/Server) เพื่อให้เกิดการทำงานได้บนหลากหลายแพลต์ฟอร์ม โดยเครื่องไคล์เอ็นท์ (client) สามารถจะเรียกใช้ method และ attribute ของวัตถุที่รันอยู่บนเครื่อง เซริฟ์เวอร์ (Server) ได้ด้วยข้อกำหนดตามรูปแบบภาษา IDL (Interface Definition Language) ซึ่งทำหน้าที่เสมือนภาษากลางที่ใช้ในการติดต่อสื่อสารบนระบบปฏิบัติการที่แตกต่างกันได้ ตัวอย่างเช่น Microsoft DCOM อนุญาตให้ระบบปฏิบัติการ WIN32 และ UNIX สามารถติดต่อกันได้, ส่วน

CORBA จะใช้ ORB (Object Request Broker) สำหรับสร้างการติดต่อระหว่าง ไคล์เอ็นท์ และ เซริฟ์เวอร์ เป็นต้น

สำหรับซอฟท์แวร์ระบบประเภท Procedural นั้นจะให้โพรเซสหนึ่งสามารถเรียกใช้ โพรซีเยอร์ที่อยู่ในอีกโพรเซสหนึ่ง ด้วยการส่งผ่านพารามิเตอร์ให้ ซึ่งอาจจะมีการส่งผลลัพธ์กลับคืนในแต่ละครั้งที่เรียกหรือไม่ก็ได้ ตัวอย่าง ซอฟท์แวร์ประเภทนี้ที่มีชื่อเสียงมากสุดคือ RPC ที่ถูกพัฒนาโดยบริษัท SUM microsystem โดยกลุ่มของโพรซีเยอร์ที่ถูกใช้นี้จะถูกพัฒนาด้วย ภาษา RPC และใช้เครื่องมือที่เรียกว่า rpcgen ในการสร้างไฟล์ stub บนเครื่องทั้ง 2 ฝั่ง โดยไฟล์ stub ตัวแรกจะเป็นของโพรเซสที่เรียก (Client) โดยจะเก็บฟังก์ชั่นที่สามารถเรียกใช้ และจะทำการสร้างเมสเส็จส่งให้โพรเซสปลายทาง (Server) ส่วนไฟล์ stub ตัวที่ 2 จะอยู่ที่ฝั่ง Server จะคอยตีความเมสเส็จที่ถูกส่งมาเพื่อดูว่ามีการเรียกใช้บริการโพรซีเยอร์ใดตามที่ประกาศไว้

2.2 เทคนิคการพัฒนาโปรแกรมคู่ขนาน (Parallel Programming Techniques)

สิ่งสำคัญอย่างแรกของการพัฒนาโปรแกรมคู่ขนานคือ วิธีการแบ่งงานซึ่งการแบ่งงานสามารถแยกได้ 2 ประเภทหลักๆคือ ฟังก์ชั่น (Function Decomposition) และ ข้อมูล (Data Decompositon) ทั้งนี้ขึ้นอยู่กับลักษณะของแอ็พพิเคชั่น เช่นบางแอ็พพิเคชั่นที่ต้องประมวลผลกับข้อมูลที่มีจำนวนมหาศาลด้วยวิธีการคำนวณที่เหมือนๆ (Data Parallelism) ตัวอย่าง เช่น การประมวลผลของ Neural Network, Weather Modeling, Fluid Dynamics และ Finite Element Analysis เป็นต้น สำหรับบางตัวอย่างของแอ็พพิเคชั่นที่ต้องผ่านหลายกระบวนการ (Function Parallelism) ได้แก่ โปรแกรมจำลองการทำงาน (Simulation) และ Ecosystem Modeling เป็นต้น

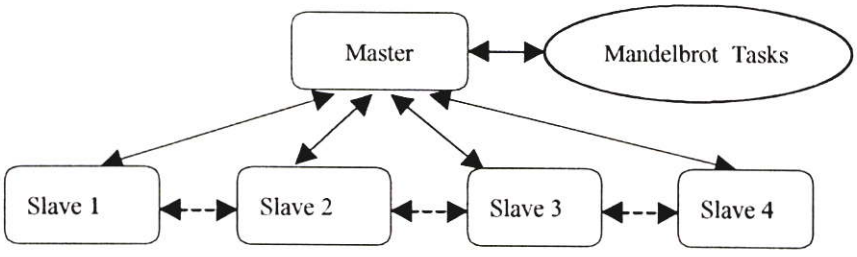
2.2.1 รูปแบบการพัฒนาโปรแกรมคู่ขนาน (Parallel Programming Paradigm)

ในการพัฒนาแอ็พพิเคชั่นโปรแกรมคู่ขนานจำเป็นจะต้องมีการกำหนดรูปแบบ (Model) ที่ใช้สำหรับการแลกเปลี่ยนข้อมูลระหว่าง โพรเซส (process) หรือ เทรด (thread) ที่ประมวลผลอยู่บนทรัพยากรของระบบนั้นๆ ปัจจุบันมีหลากหลายรูปแบบแต่สำหรับระบบส่งผ่านเมสเส็จแล้วสามารถจำแนกได้เป็น 3 ประเภทหลักๆ ดังนี้

2.2.1.1 Master/Slaves Paradigm (Mandelbrot algorithm)

การพัฒนาโปรแกรมในรูปแบบนี้จะต้องมีอย่างน้อย 2 โปรแกรมหลักๆคือ Master โปรแกรม และ Slave โปรแกรม โดย Master โปรแกรมจะเป็นตัวสร้าง Slave โพรเซสและควบคุมการติดต่อสื่อสารของตัว Slave โปรแกรม และส่วน Slave โปรแกรมจะเป็นส่วนคำนวณเป็นหลัก

แต่บางครั้งอาจจะออกแบบให้ Slave โปรแกรมสามารถติดต่อสื่อสารระหว่างกันได้ดังแสดงในรูปที่ 2.4

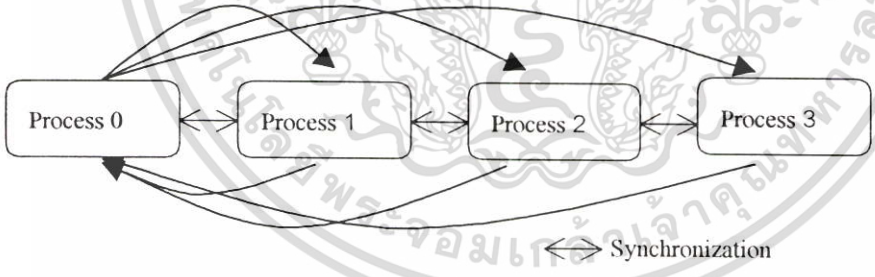


รูปที่ 2.4 การพัฒนาโปรแกรมรูปแบบ Master/Slaves

สำหรับอัลกอริทึมรูปแบบนี้ถ้าให้ Slave โปรแกรมสามารถสร้าง Slave โพรเซสของมันได้อีกเป็นทอดๆ ผลที่ได้คือโครงสร้างแบบ Tree และจะเรียกการคำนวณในลักษณะนี้ว่า Tree computing ซึ่งก็จะคล้ายกับ Divide and Conquer ของ Sequential Algorithm นั้นเอง

2.2.1.2 Node Only Paradigm (Group Computing)

การพัฒนาโปรแกรมในรูปแบบนี้แต่ละโพรเซสจะต้องจัดการควบคุมการติดต่อสื่อสารเองทั้งหมดโดยอาจจะกำหนดให้โพสเซสหนึ่งโพรเซสใดเป็น Master ขึ้นมาก็ได้ดังแสดงรูปข้างล่างนี้



รูปที่ 2.5 การพัฒนาโปรแกรมแบบ Group

ทุกๆโพรเซสในระบบจะเกิดการทำงานร่วมกันเป็นกลุ่มใหญ่ หรือ ทีมขึ้นบนระบบ โดยทั่วไปมักจะกำหนด Process 0 เป็น Master ตัวอย่างของการพัฒนาโปรแกรมในรูปแบบนี้ก็คือ SPMD (Single Program Multiple Data) โดยพัฒนาขึ้นมาเพียงโปรแกรมเดียวแต่ประมวลผลกับข้อมูลที่มีความแตกต่างกัน

2.2.1.2 Hybrid

เป็นการรวมแนวคิดทั้ง 2 รูปแบบที่ได้กล่าวมาแล้วข้างต้นซึ่งอาจจะเป็นโครงสร้างแบบ Tree โดยที่โหนดระดับเดียวกันสามารถจะแลกเปลี่ยนข้อมูลระหว่างกันได้ หรืออาจเป็นตามรูปที่ 2.5 ถ้าทุกๆ Slave โพรเซสสามารถติดต่อถึงกันได้เอง

2.2.2 การวิเคราะห์การพึ่งพาข้อมูล (Data Dependency Analysis)

ในการแปลงจากโปรแกรมเรียงลำดับให้เป็นโปรแกรมคู่ขนานสิ่งสำคัญคือการพิจารณาจากความสัมพันธ์ข้อมูลและชุดคำสั่งที่จะถูกแบ่ง ซึ่งส่วนใหญ่จะเริ่มที่การวิเคราะห์ข้อมูลก่อนเพื่อออกแบบโครงสร้างข้อมูลให้เหมาะสมก่อนจึงจะพิจารณาต่อไปว่าควรจะทำการแบ่งชุดคำสั่งอีกหรือไม่

โดยทั่วไปคำสั่งที่กระทำกับข้อมูล 2 อย่างคือ การอ่าน (Read) คือการนำข้อมูลนั้นมาใช้งาน และ การเขียน (Write) คือการแสดงผลลัพธ์ลงบนหน่วยความจำของเครื่อง ดังนั้นความพึ่งพาข้อมูลที่ใช้ร่วมกันระหว่างชุดคำสั่ง จะเกิดได้ 3 รูปแบบหลักๆตามนิยามดังนี้

นิยาม 2.1 (ความสัมพันธ์เชิงพึ่งพากันของข้อมูล)

กำหนดให้ 2 ชุดคำสั่งคือ S1 และ S2 มีการใช้ข้อมูลร่วมกันดังนั้นความพึ่งพาของข้อมูลในชุดคำสั่งทั้งสอง สามารถเกิดได้ 3 ลักษณะดังนี้

1. **Flow Dependence** : ถ้าชุดคำสั่งที่ S2 มีการใช้ข้อมูลร่วม A ซึ่งเป็นผลลัพธ์ของ S1 ดังนั้นสามารถกล่าวได้ว่า S2 เกิดการพึ่งพาข้อมูลโดยตรงจาก S1 โดยสามารถเขียนสัญลักษณ์แทนดังนี้ S1-f->S2

ตัวอย่างเช่น S1: A=B+C

S2: D=A+10

จากตัวอย่างแสดงให้เห็นว่า S2 จะต้องรอผลลัพธ์ข้อมูลร่วม A จาก S1 ก่อน

2. **Anti Dependence** : ถ้าชุดคำสั่งที่ S2 ให้ผลลัพธ์ข้อมูลร่วม B หลังจากชุดคำสั่ง S1 ได้ใช้ข้อมูลร่วม B แล้ว ดังนั้นสามารถกล่าวได้ว่า S2 เป็นความพึ่งพาเชิงกลับกับ S1 โดยสามารถเขียนสัญลักษณ์แทนดังนี้ S1-a->S2

ตัวอย่างเช่น S1: A=B+C

S2: B=10

ในตัวอย่างนี้แสดงให้เห็นว่าข้อมูลร่วม B นี้เกิดผลลัพธ์ที่ S2 หลังจาก S1 เรียกใช้เสร็จแล้ว

3. **Output Dependence** : ถ้าชุดคำสั่ง S2 พยายามให้ผลลัพธ์ข้อมูลร่วม A อีกครั้งหลังจากชุดคำสั่ง S1 ให้ผลลัพธ์ร่วม A นี้ไปแล้ว นั้นคือแสดงการเกิดความพึ่งพาผลลัพธ์ร่วมกันระหว่างชุดคำสั่งทั้งสอง โดยสามารถเขียนสัญลักษณ์แทนดังนี้ S1-o->S2

ตัวอย่างเช่น S1: A=B+C

S2: A=D-10

ในตัวอย่างนี้แสดงให้เห็นว่า S2 จะให้ผลลัพธ์ข้อมูลร่วม A อีกครั้งหลังจาก S1 ให้ไปแล้ว

จากลักษณะความสัมพันธ์พึ่งพาข้อมูลทั้ง 3 แบบจะเห็นได้ว่า ความพึ่งพาข้อมูลแบบที่ 2 และ 3 นี้สามารถที่จะแยก 2 ชุดคำสั่งนี้ให้ทำงานควบคู่กันได้โดยการเปลี่ยนชื่อตัวแปร (Variable renaming)

2.2.3 การวิเคราะห์การพึ่งพาภายในลูปโปรแกรม

แอ็พพิเคชั่นส่วนมากจะประมวลผลกับข้อมูลอาเรย์ ดังนั้นการระบุความพึ่งพาข้อมูลระหว่าง 2 ชุดคำสั่งเป็นเรื่องสำคัญมาก ตัวอย่างเช่น

A[2*I+1] =B[I] S1

D[I] = A[2*I] S2

นั้นคือ S1 และ S2 สามารถทำงานควบคู่กันได้หรือไม่ ซึ่งทั้งสองชุดคำสั่งใช้ตัวแปรอาเรย์ A ร่วมกัน แต่ทั้งสองชุดคำสั่งมีการอ้างอิงตำแหน่งอาเรย์ A แตกต่างกัน ดังนั้นจะมีวิธีตรวจสอบว่าทั้งสองชุดคำสั่งนี้สามารถแยกทำงานได้อย่างอิสระหรือไม่ ซึ่งถ้าเกิดการพึ่งพาขึ้นกับตัวแปรอาเรย์ A จะเรียกการพึ่งพาแบบนี้ว่า "*Loop carried data dependency*" (การพึ่งพาเกิดขึ้นที่ I มีค่าต่างกันภายในลูปโปรแกรม) โดยจะมีวิธีที่นิยมใช้ในการตรวจสอบการพึ่งพาดังนี้

2.2.3.1. GCD test

เป็นวิธีง่ายที่สุดโดยการหาตัวที่สามารถหารทุกๆจำนวนได้ลงตัว ซึ่งสามารถแทนด้วยสัญลักษณ์ดังนี้ $GCD(a_1,a_2,a_3,\ldots,a_n) \mid C$ (a – ค่าสัมประสิทธิ์ของตัวแปร I จำนวน n และ C- ผลต่างของค่าคงที่) โดยถ้ามีตัวหารที่ไม่เท่ากับ 1 และไม่มากกว่าจำนวนน้อยสุดที่แสดงทุกๆค่าโพลิโนเมียล a และ C ได้แสดงว่าเกิดความพึ่งพาข้อมูลระหว่าง 2 ชุดคำสั่ง ตัวอย่างเช่นลูปโปรแกรมนี้

```
For I=1 to 100
    A[2I-1] = ...                    S1
        = A[4I – 7] +.....           S2
End for
```

เมื่อหา GCD test จะได้

$GCD(2,4) \mid -1-(-7)$ หรือ $GCD(2,4) \mid 6$

โดยตัวหารร่วมมากที่หาได้คือ 2 แสดงว่าเกิดความพึ่งพากันระหว่าง S1,S2 แต่วิธีนี้มีข้อจำกัดคือไม่สามารถระบุประเภทการพึ่งพาที่เกิดขึ้นได้

2.2.3.2. Diophatine Equation test

เป็นการแก้สมการของ Diphantine ด้วย 2 ตัวแปรคือ x,y ดังนี้

$$ax+by = C$$

จากนั้นหาค่า GCD(a,b)

$$g = gcd(a_1, a_2, \ldots, a_n) \mid C$$

และหาค่า g ของ a,b ด้วยสมการดังนี้

$$g = au + bv.$$

ดังนั้นสมการที่ใช้หาค่าของ x,y คือ

$$x(t) = u.c/g + b.t/g$$

$$y(t) = v.c/g - a.t/g$$

ซึ่งพิจารณาจากตัวอย่างลูปโปรแกรมนี้

```
For I=1  to  100
        A[2I-1] = …..                    S1
                = A[4I – 7] +……          S2
End for
```

เริ่มต้นด้วยการจัดสมการของตัวแปรอาร์เรย์ A ในรูปแบบสมการ x,y ใหม่คือ

$$2x-1 = 4y-7$$

$$4y – 2x = 6$$

หา GCD(4,-2) = 2 นั้นคือ g=2 และจากสมการ

$$g = au + bv$$

$$x(t) = u^* c/g + b^* t/g$$

$$y(t) = v^* c/g - a^* t/g$$

แทนค่า g=2 , a = -2,b=4 ,c=6

$$2 = -2u+4v \qquad (1)$$

$$x(t) = u^*6/2 + 4^*t/2 = 3u + 2t \qquad (2)$$

$$y(t) = v^*6/2 –(-2)^*t/2=3v+t \qquad (3)$$

จากสมการ (1) เมื่อเราแทนค่า u,v = 1 ใน (1) จะทำให้เป็นจริง ดังนั้นจะใช้ u=1,v=1 คือ

$$x(t)=3+2t$$

$$y(t) = 3+t$$

แทนค่า t =1 จะได้ x = 5, y=4 นั้นคือมีการเกิดความพึ่งพากันข้อมูลระหว่างรอบที่ I=4 และ I=5 โดยเมื่อลองทำการแทนค่าดังกล่าวในลูปโปรแกมจะพบว่าที่ตัวแปร A[9] ในรอบที่ I=5 จะเกิด Anti Data Dependency กับที่ I=4 นั้นเอง

อย่างไรก็ตามวิธีนี้ยังไม่มีประสิทธิภาพเท่าใดนักแม้นสามารถหาประเภทความพึ่งพาข้อมูลได้ก็ตาม (หลังจากหาสมการ x(t) และ y(t) ได้แล้ว) และรู้ตำแหน่ง I ที่เกิดด้วย แต่ไม่ได้คิดค่าขอบเขตของลูปโปรแกรม I

2.2.3.3 Exact Test

วิธีนี้เพิ่มเติมจากวิธีที่ 2 คือคิดช่วงของลูปเพื่อดูค่า t ที่เกิดการซ้อนทับกัน แสดงการเกิดความพึ่งพาข้อมูล จากตัวอย่างลูปโปรแกรมที่แล้วทำให้ทราบช่วงของค่า x,y คือ 1<=x,y,<=100 ดังนั้นเมื่อทำการกำหนดช่วงในสมการเดิมจะแสดงได้ดังนี้

$$1 <= 3+2t <= 100 \quad (1)$$

$$1 <= 3+t <= 100 \quad (2)$$

แก้สมการ (1) ได้ $-1 <= t <= 48$

แก้สมการ (2) ได้ $-2 <= t <= 97$

โดยค่าที่ทับกันคือช่วง –1 ถึง 48 แสดงว่าเกิดความพึ่งพากันข้อมูลที่ค่า t = 1 ถึง 48 แม้นว่าวิธีนี้สามารถทราบช่วงที่เกิดความพึ่งพาข้อมูลของลูปโปรแกรมได้แต่วิธีนี้ไม่สามารถใช้ได้กับกรณีลูปซ้อนกันหลายชั้น (nested loop)

2.2.3.4 Bounds Test

จากทั้ง 3 วิธีที่กล่าวมาจะมีข้อจำกัดคือไม่สามารถใช้ได้กับกรณีลูปซ้อนกัน ดังนั้นวิธีการนี้เป็นวิธีที่ง่ายสุดในการทดสอบเพื่อดูการพึ่งพากันข้อมูลที่จะทดสอบว่ามีหรือไม่ ซึ่งจะคล้ายกับ GCD test เพียงแต่จะคิดขอบเขตของตัวแปร I,J เพื่อความถูกต้องโดยจะทำการแทนค่าสูงสุดของขอบเขตลงในสมการซึ่งถ้าค่าน้อยกว่า 0 แสดงว่าเป็นอิสระต่อดังตัวอย่างโปรแกรมนี้

```
For I=1 to 30
        For J=1 to 30
                a(200-I) =.....
                .......  =a(I+J)
        End for
End for
```

จากโปรแกรมกำหนดให้ i1 ของ f(i1) = 200-i1 และ i2,j2 ของ f(i2,j2) = i2+j2 สามารถเขียนความสัมพันธ์ของสมการได้ดังนี้

```
f(i1)   = f(i2,j2)
200-i1  =i2+j2
```

หรือ i2+i1+j2-200 = 0 และพิจารณาค่าสูงสุดจะได้ 1<=i1,i2,j2 <=30 ซึ่งค่าสูงสุดที่ได้จากสมการคือ i2=i1=j2 = 30 และเมื่อแทนค่าเพื่อหาค่าสูงสุดจะได้ -110 นั้นแสดงให้เห็นว่าทั้ง 2 ชุดคำสั่งนี้เป็นอิสระต่อกันและสามารถทำงานแบบคู่ขนานได้

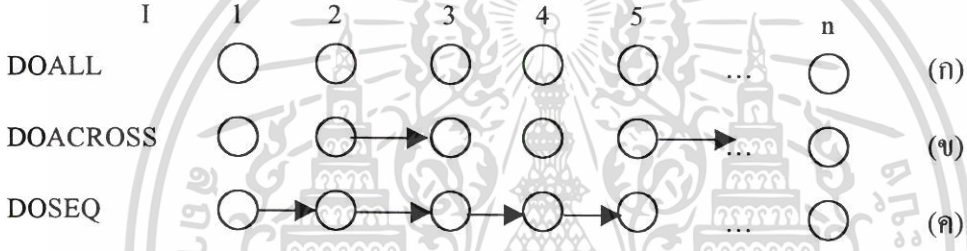
2.2.4 การจำแนกประเภทของลูปโปรแกรม

ลูปโปรแกรมสามารถแยกได้เป็น 3 ประเภทหลักๆ ได้ดังนี้

1. DOALL ทุกๆรอบของลูปสามารถทำงานได้พร้อมๆกันนั้นคือตัวแปรเว็กเตอร์ I ขนาน n รอบ <I1,I2,…,In> นั้นสามารถประมวลผลได้พร้อมๆกัน

2. DOACROSS แสดงว่าลูปนี้จะมีเฉพาะบางค่าของเว็กเตอร์ I ที่ต้องทำงานแบบเรียงลำดับ

3. DOSEQ ใช้เพื่อแสดงว่าลูปดังกล่าวทำงานแบบเรียงลำดับเท่านั้น



รูปที่ 2.6 แสดงการทำงานของลูปทั้ง 3 ประเภทคือ (ก) DOALL (ข) DOACROSS (ค) DOSEQ

ในภาษาฟอร์แทนสามารถระบุได้ตามตัวอย่างลูปโปรแกรมนี้

```
DO  I=2 to n
        D0 J = 2  to m
                a(I,J) = a(I-1 ,J) +1
        ENDDO
ENDDO
```

สามารถเขียนใหม่ได้ดังนี้

```
DOSEQ  I=2  to n
        DOALL J=2,m
                A(I,J) = A(I-1,J) +1
        ENDDO
ENDDO
```

2.2.5 เทคนิคการแปลงลูปโปรแกรม (Loop Transformation)

เทคนิคต่างๆที่นำเสนอจะช่วยปรับโครงสร้างลูปเพื่อให้เหมาะสมกับการทำงานแบบคู่ขนาน โดยที่ผลลัพธ์ยังคงเหมือนเดิม ซึ่งมีดังนี้

1. Loop Distribution

เป็นการแตกแต่ละชุดคำสั่งที่ไม่มีการพึ่งพากันข้อมูลภายในลูปออกเพื่อทำงานแบบคู่ขนาน ดังแสดงตามตัวอย่างลูปโปรแกรมนี้

```
DO  I=1 to N
        A(I+1) = B(I-1) + C(I)          S1
        B(I) = A(I)*10                  S2
        C(I) = B(I) –1                  S3
ENDDO
```

เมื่อทดสอบโปรแกรมลูปจะพบว่าตัวแปร A ระหว่างชุดคำสั่ง S1 กับ S2 เกิด Loop carried data dependency และตัวแปร B(I) ระหว่างชุดคำสั่ง S1 กับ S2 ก็เกิด Loop carried data dependency เช่นกัน สำหรับตัวแปร B ในชุดคำสั่ง S2 กับ S3 ไม่เกิด และในส่วนตัวแปร C ในชุดคำสั่งระหว่าง S1 กับ S3 ไม่เกิดด้วยเช่นกัน ดังนั้นสามารถทำ Loop Distribution ได้ดังนี้

```
DO I=1 to N
        A(I+1) = B(I-1) + C(I)          S1
        B(I)   =  A(I) *K               S2
ENDDO


DO  I=1  to N
        C(I) = B(I) –1                  S3
ENDDO
```

2. Loop Fusion (Loop Jamming)

เทคนิคนี้จะรวม 2 ลูปโปรแกรมให้เหลือเพียง 1 เพื่อลดความสูญเสียของการทำงานคู่ขนาน ดังแสดงในตัวอย่างลูปโปรแกรมต่อไปนี้

```
DOALL  I=1 to n                 ลูปที่ 1
        D(I) = E(I) + X(I)
ENDDO
```

```
DOALL  J=1 to n                    ลูปที่ 2
        E(J) = D(J) + 10
ENDDO
```

โดยทั้งสองลูปสามารถรวมกันได้เพราะมีขอบเขตที่เหมือนกันคือ n ดังนี้

```
DOALL  I=1 to n
        D(I) = E(I) + X(I)
        E(I) = D(I) + 10
ENDDO
```

3. Loop Coalescing

เทคนิคนี้จะแปลงลูปต่อเนื่องหลายชั้น (nested loop) ให้เหลือเพียงลูปเดียวดังที่แสดงให้เห็นจากตัวอย่างโปรแกรมข้างล่างนี้

```
DOALL  j=1 to n
        DOALL  k=1 to n
                A(j,jk) = ……
        ENDDO  k
ENDDO  j
```

ดังนั้นจะลดเหลือเพียงลูปเดียวดังนี้

```
DOALL  I =1  to n*n
        A(ceil(i/n), I-n(bottom(I-1)/N)) =……..
ENDDO  I
```

โดยเทคนิคนี้ลดความสูญเสียในการออกแบบแอ็พพิเคชั่นคู่ขนานแต่ขนาดภาระงานจะเพิ่มขึ้นด้วย

4. Loop Interchange

เทคนิคนี้จะสลับตัวแปรเว็กเตอร์เพื่อให้เกิดการทำงานแบบคู่ขนานมากที่สุดดังแสดงให้เห็นจากตัวอย่างลูปโปรแกรมดังนี้

```
DOSEQ  I=2  to  N
   DOALL J=1 to N
        A(I,J) = A(I-1,J) + B(I)
   ENDDO
ENDDO
```

ซึ่งสามารถสลับลูปโปรแกรมใหม่ได้ดังนี้

```
DOALL  J=1  to N
    DOSEQ I=2 to N
        A(I,J) = A(I-1,J) + B(I)
    ENDDO
ENDDO
```

5. Loop Shrinking

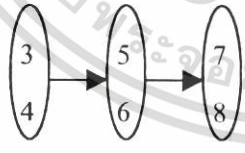
จากลูปโปรแกรมตัวอย่างนี้

```
DOSEQ  I=3  TO  N
    A(I) = B(I-2) –1          S1
    B(I) =  A(I-3)*K          S2
ENDDO
```

สามารถแสดงการพึ่งพาที่เกิดกับตัวแปรอาเรย์ A และ B ของลูปโปรแกรมระหว่างทั้งสองชุดคำสั่ง S1 และ S2 ได้ดังรูปที่ 2.7

รูปที่ 2.7 แสดงการพึ่งพาที่เกิดขึ้นของลูปโปรแกรมตัวอย่าง

เทคนิคนี้จะเพิ่มให้เกิดการทำงานแบบคู่ขนานขึ้นด้วยการทำงานตามที่แสดงในรูปที่ 2.8



รูปที่ 2.8 แสดงการพึ่งพาใหม่จากการปรับโครงสร้างลูปด้วยเทคนิค Loop Shrinking

สามารถแปลงโครงสร้างของลูปโปรแกรมใหม่ได้ดังนี้

```
DOSEQ  J=3 to  N Step 2
    DOALL I=J ,J+1
        A(I) = B(I-2) –1;  B(I) = A(I-3) *K
    ENDDO
ENDDO
```

6. Loop Unrolling

เทคนิคนี้ช่วยลดความพึ่งพากันข้อมูลแบบ Loop Carried Data Dependency ด้วยการเพิ่มชุดคำสั่งภายในลูปโปรแกรมเพิ่มขึ้นตัวอย่างเช่นลูปโปรแกรมนี้

```
DO I = 1 to N
        A(I) = A(I-1) *10
ENDDO
```

เมื่อทำการ Unrolling จะเป็นดังนี้

```
DO I=1 to N  Step 2
        A(I) = A(I-1)*10
        A(I+1) = A(I-1)*100
ENDDO
```

7. Loop Spreading

คือเทคนิคการกระจายตัวแปรเว็กเตอร์ที่แตกต่างกันให้กับแต่ละโพรเซสเซอร์ที่อยู่บนระบบเพื่อให้เกิดความสมดุลยกัน โดยการกำหนด PAR ดังนี้

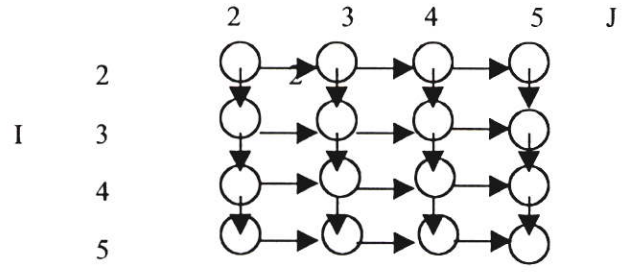
```
PAR I=1 to P
        S1(I)
        S2(I)
ENDPAR
```

8. Loop Skewing

เทคนิคนี้เรียกอีกชื่อหนึ่งว่า Wave Font ซึ่งจะปรับการทำงานของโปรแกรมลูปต่อเนื่องใหม่เพื่อเพิ่มการทำงานแบบคู่ขนานมากขึ้นดังตัวอย่างโปรแกรมที่แสดงนี้

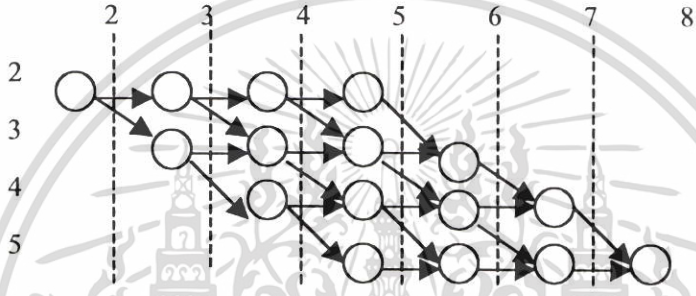
```
DO  I=2 to 5
        DO J=2 to 5
                A(I,J) = (A(I+1,J)+A(I-1,J)+A(I,J+1)+A(I,J-1))/4
        ENDDO
ENDDO
```

โดยการทำงานของโปรแกรมดังกล่าวแสดงได้ดังนี้



รูปที่ 2.9 แสดงการทำงานของลูปโปรแกรมดังกล่าว

ซึ่งสามารถปรับเปลี่ยนการทำงานของโปรแกรมดังกล่าวให้ทำงานแบบ Wave Font ได้ดังนี้



รูปที่ 2.10 แสดงการทำงานของลูปโปรแกรมจากการปรับโครงสร้างแบบ Wave Font

โดยจะเขียนลูปโปรแกรมใหม่ได้ดังนี้

```
DOALL I=2 to 5
    DOSEQ J=I+2,I+6
        A(I,J-1) = (A(I+1,J-I)+A(I-1,J-I)+A(I,J+1-I)+A(I,J-1-I)) /4
    ENDDO
ENDDO
```

2.3 ทฤษฎีการพัฒนาโปรแกรมระบบ PVM (Parallel Virtual Machine)

PVM คือ ซอฟท์แวร์ระบบที่ทำให้เกิดการทำงานร่วมกันของเครื่องคอมพิวเตอร์หลายเครื่องบนระบบเครือข่ายเดียวกันเสมือนเป็นเครื่องจักรที่มีกำลังในการคำนวณมหาศาลเพื่อแก้ปัญหาแอ็พพิเคชั่นระดับใหญ่ๆ ทั้งนี้โดยอาศัยหลักการของการส่งผ่านเมสเส็จระหว่างกลุ่มโพรเซสที่ทำงานอยู่บนระบบเสมือนเดียวกันด้วยพื้นฐานของการใช้เมสเส็จสำหรับการติดต่อสื่อสารกัน โดยเมสเส็จสามารถจะถูกนำไปประยุกต์ใช้สำหรับการพัฒนาแอ็พพิคชั่นระบบได้ 3 รูปแบบหลักๆคือ 1.ใช้ส่งผ่านข้อมูลระหว่างกัน (Data delivery), 2.เป็นสัญญาณ (Signal) เพื่อกำหนดจังหวะการทำงานร่วมกัน และ 3.ออกแบบให้เป็นชุดคำสั่ง (Command) ของโพรเซสปลายทาง เป็นต้น

องค์ประกอบระบบ PVM มี 2 ส่วนหลักๆคือ 1.PVMD – ซึ่งจะคอยรันเป็นโพรเซสเบื้องหลังตามเครื่องต่างๆบนระบบเสมือน (Virtual Machine) เพื่อสร้างช่องทางสำหรับการส่งผ่านเมสเส็จบนระบบ รวมถึงการเตรียมคุณลักษณะความทนทานต่อการล่ม (Fault Tolerance) 2. Libpvm – เป็นไลบรารีที่เก็บรูทีนฟังก์ชั่น (routine functions) ต่างๆของการรับส่งเมสเส็จรวมถึงการเข้าถึงทรัพยากรบนระบบเสมือน

2.3.1 การเชื่อมต่อระบบ PVM กับผู้ใช้ (PVM User Interface)

งานทั้งหมดบนระบบ PVM จะถูกกำหนดด้วยเลขจำนวนเต็ม 32 บิตที่เรียกว่า TID ซึ่งจะคล้ายกับ ProcessID (PID) ที่ใช้บนแต่ละระบบปฏิบัติการ ซึ่งในการพัฒนาแอ็พพิเคชั่นโปรแกรมระบบสามารถใช้ C/C++ หรือ ภาษาฟอร์แทรน ได้เชื่อมต่อกับไลบรารีฟังก์ชั่น (LINK) ที่ PVM จัดเตรียมไว้ (libpvm3.a บนระบบปฏิบัติการ Linux และ libpvm3.lib บนระบบปฏิบัติการ WIN32) สำหรับภาษา C,C++ และ libfpvm3 สำหรับภาษาฟอร์แทรน เป็นต้น ซึ่งในภาษาฟอร์แทรนนั้นจะเป็นต้อง LINK กับไลบรารี livpvm3 เข้าไปด้วย (libfpvm3.a / libfpvm3.lib เก็บฟังก์ชั่นที่ห่อหุมรูทีนฟังก์ชั่นต่างๆของ livpvm3.a อีกชั้นหนึ่ง เพื่อการแปลงโค๊ดเป็นภาษา C) โดยรูทีนต่างๆที่เก็บในไลบรารีเหล่านี้สามารถจำแนกได้เป็น 5 กลุ่มหลักๆดังนี้

1. **Process Control**

```
int tid = pvm_mytid(void)
call pvmfmytid(void)
```

ฟังก์ชั่นนี้จะให้ค่า TID (Task Identifier) สำหรับโพรเซสที่ลงทะเบียนเข้าสู่ระบบเรียบร้อย ซึ่งค่าที่ได้ควร > 0 แต่ถ้า <0 แสดงว่าเกิด Error ดังนั้นระบบจะให้ค่า error code แทนเพื่อตรวจหา Error ที่เกิดขึ้นต่อไป

```
int info = pvm_exit(void)
call pvmfmytid(void)
```

ฟังก์ชั่นนี้จะบอกให้ PVMD บนเครื่องหลักว่าโพรเซสนี้กำลังจะออกจากระบบ และ PVMD บนเครื่องหลักจะแจ้งให้ทุกๆ PVMD บนเครื่องลูกข่ายรับทราบถึงการถอดโพรเซสออกจากระบบ ซึ่งฟังก์ชั่นนี้ไม่ได้กำจัดโพรเซสออกจากเครื่อง ดังนั้นมันยังคงทำงานต่อไปได้เหมือนโพรเซสอื่นๆบนเครื่องนั้นๆ ซึ่งผู้ใช้จะต้องจัดการเรียกฟังก์ชั่น exit() ในโปรแกรมภาษา C หรือ คำสั่ง STOP ของภาษาฟอร์แทรนด้วยตัวเอง

```
int  numt = pvm_spawn(char *task,char **argv,int flag,
char *where,int ntask,int *tids)
call pvmf spawn (task,flag,where,ntask,numt)
```

ฟังก์ชั่นนี้เป็นการสั่งโปรแกรมชื่อ task ทำงานตามจำนวน ntask บนระบบเสมือน (virtual machine) โดยการส่งพารามิเตอร์ผ่านตัวแปรพอยน์เตอร์ argv ซึ่งถ้าไม่มีอาร์กิวเมนต์นี้ ค่า argv ควรเป็น NULL ส่วนพารามิเตอร์ flag ใช้เป็นตัวเลือกและเป็นผลรวมของตัวเลือก ดังนี้

Value	Option	Meaning
0	PvmTaskDefault	ให้ PVM สั่งรันโพรเซสเอง
1	PvmTaskHost	สั่งรันงานนั้นๆตามอาร์กิวเมนต์ where
2	PvmTaskArch	สั่งรันงานบนเครื่องที่มีสถาปัตยกรรมตรงตามที่กำหนดใน PVM_ARCH
4	PvmTaskDebug	สั่งรันงานภายใต้ตัวดีบัก (Debugger)
8	PvmTaskTrace	สร้างตัวติดตามข้อมูล (Tracer)
16	PvmMppFront	PvmMppFront
32	PvmHostCompl	ไม่รันงานบนเครื่องที่กำหนดใน where

โดยค่าตัวเลือก (option) ถูกระบุไว้ก่อนแล้วใน PVM3/include/pvm3.h และ fpvm3.h สำหรับภาษาฟอร์แทรน

PvmTaskTrace เป็นตัวเลือกใหม่ที่มีตั้งแต่ PVM3.3 ขึ้นไป โดยจะสร้างตัวติดตามข้อมูลซึ่งจะทำงานร่วมกับโปรแกรม XPVM เพื่อให้ผู้ใช้สามารถไล่เหตุการณ์ต่างๆได้ ซึ่งพัฒนาจาก PvmTaskDebug ที่ใช้ใน pvm_setopt() เพื่อสั่งรันตัวดีบักโปรแกรมบน PVM ซึ่งเป็นโปรแกรมง่ายๆสำหรับค้นหาข้อผิดพลาดบนระบบ

ค่า numt คือจำนวนของ Task ที่ทำงานได้จริงบนระบบ ซึ่งถ้าสำเร็จก็จะแสดงหมายเลข Tid แต่ถ้าไม่สำเร็จก็จะแสดงค่าผิดพลาดผ่านตัวแปรเว็กเตอร์ tids : นั้นคือถ้างานที่ไม่สามารถสร้างได้สำเร็จจะมีการเก็บค่าผิดพลาดต่างๆอยู่ที่ตัวแปรเว็กเตอร์ที่ตำแหน่ง ntask-numt เป็นต้นไป

```
int  info = pvm_kill(tid)
call pvmfkill(tid,info)
```

ฟังก์ชั่นนี้ใช้กำจัดงานบนระบบตามที่ระบุค่าใน tid แต่ฟังก์ชั่นนี้ไม่ได้ถูกออกแบบมาเพื่อกำจัดงานที่มันกำลังเรียกใช้อยู่ ดังนั้นถ้าทำงานเสร็จสิ้นแล้วควรเรียก pvm_exit() ตามด้วย exit()

```
int info = pvm_catchout(FILE *ff)
call pvmfcatchout( onoff)
```

โดยทั่วไป PVM จะเขียน stderr และ stdout ของงานบนระบบไว้ในล็อกไฟล์ (log file) /tmp/pvml.<uid> ซึ่งฟังก์ชั่นเหล่านี้จะกำหนดให้เขียนลงไฟล์ตามที่ระบุแทน โดย pvmd ของเครื่องลูกข่ายในระบบเสมือนจะทำหน้าที่เขียนไฟล์ในเครื่องตนก่อนส่งให้เครื่องหลัก เพื่อนำไปเขียนเพิ่มเติมในแต่ละบรรทัดของไฟล์บนเครื่องหลัก

ซึ่งถ้าฟังก์ชั่น pvm_exit() ถูกเรียกโดยโพรเซสแม่ (parent process) ก่อนที่จะส่ง output เสร็จ ระบบจะ Block จนกว่าการส่งเสร็จสิ้นจึงจะสิ้นสุดงานได้ หรือสามารถหลีกเลี่ยงปัญหาดังกล่าว ได้โดยเรียก pvm_catchout(0) ก่อนเรียก pvm_exit()

2. Information

```
int tid = pvm_parent()
call pvmfparent(tid)
```

ฟังก์ชั่นนี้จะให้ค่า TID ของโพรเซสแม่ที่สั่งรันโพรเซสนี้บนระบบ ซึ่งถ้าให้ค่า PvmNoParent แสดงว่าโพรเซสนี้ไม่ได้ถูกสั่งรันโดยฟังก์ชั่น pvm_spawn()

```
int dtid = pvm_tidtohost(int tid)
call pvmf tidtohost (tid,dtid)
```

ฟังก์ชั่นนี้จะให้ค่า TID ของ pvmd ที่กำลังทำงานอยู่บนเครื่องเดียวกับ tid เพื่อหาว่างานดังกล่าวทำงานบนเครื่องใด

```
int info = pvm_config(int *nhost,int *narch,struct
pvmhostinfo **hostp)
call pvmfconfig(nhost,narch,dtid,name,arch,speed,info)
```

ฟังก์ชั่นนี้จะให้รายละเอียดระบบเสมือน ตั้งแต่จำนวนเครื่องที่ใช้บนระบบ nhost, และจำนวนของรูปแบบข้อมูลที่แตกต่างกัน narch โดย hostp คือพอยนทเตอร์ของตัวแปรอาเรย์ pvmhostinfo

ที่จะเก็บค่า TID ของ pvmd, ชื่อเครื่อง, สถาปัตยกรรมเครื่อง และ ความเร็วของ CPUของเครื่องนั้นๆในค่าติดตั้งของระบบ

```
int  info = pvm_tasks(int which,int *ntask,struct
pvmtaskinfo **taskp)
call pvmfctasks(which,ntask,tid,ptid,dtid,flag,aout,info)
```

ฟังก์ชั่นนี้จะให้รายละเอียดเกี่ยวกับ task บน PVM ที่กำลังทำงานอยู่ which คือ หมายเลข TID ของ task ที่ให้รายละเอียดกลับมา ถ้ากำหนด which เป็น 0 หมายถึงทุกๆงานที่รันอยู่ทั้งหมด ณ ขณะนี้ แต่ถ้าเป็นค่า pvmd TID (dtid) หมายถึง งานที่กำลังรันอยู่บนเครื่องนั้นๆ หรือบางทีอาจจะระบุ TID ของ task ที่รันอยู่โดยตรงก็ได้

โดยจำนวนงานจะถูกส่งกลับอยู่ใน ntask และ taskp คือพอยนทเตอร์ของตัวแปรอารเรย์ pvmtaskinfo ที่เก็บค่า TID, dtid, ptid, สถานะของ flag และ ชื่อไฟล์ของงานที่ถูกสร้างขึ้น (ถ้างานดังกล่าวถูกสั่งรันแบบผู้ใช้สั่งเอง ชื่อไฟล์จะไม่แสดงให้เห็น)

```
int code = pvm_archcode( char *arch )
call pvmfarchcode( arch, cod )
```

ฟังก์ชั่นนี้จะแสดงรหัสของชื่อสถาปัตยกรรมในตัวแปร พอยนทเตอร์ arch ซึ่งรหัสที่ส่งคืนนี้แสดงให้ทราบถึงความเข้ากันได้ของข้อมูลไบนารีบนเครื่องนั้นตัวอย่างเช่นบนเครื่อง SUN4 และ RS6K จะใช้รหัสเดียวกัน ดังนั้นสามารถเข้ารหัสการส่งข้อมูลแบบ PvmDataRaw ได้ แทนที่จะส่งแบบ PvmDataDefault ระหว่าง 2 งานบน 2 เครื่อง

```
int status = pvm_pstat( tid )
call pvmfpstat( tid, status )
```

ฟังก์ชั่นนี้แสดงสถานะของโพรเซสบนระบบด้วยการระบุพารามิเตอร์ TID โดยค่าสถานะที่ส่งคืนกลับนี้มี 3 ค่าหลักๆคือ PvmOk ถ้างานนั้นยังคงรันอยู่ในระบบ, PvmNoTask ถ้าไม่มีงานนั้นบนระบบ และ PvmBadParam แสดงค่าระบุค่า TID ไม่ถูกต้อง ซึ่งฟังก์ชั่นนี้สามารถใช้งานร่วมกับ pvm_notify() เมื่อโพรเซสที่เรียกนั้นเกิดการล่มบนระบบ

3. Virtual Machine Control

ความสามารถที่สำคัญของ PVM คือการเพิ่มและลดจำนวนเครื่องที่ใช้งานบนระบบเสมือน ซึ่งสร้างความเป็นหนึ่งเดียว (Single image) ด้วยฟังก์ชั่นทั้งสองดังนี้

```
int info = pvm_addhosts (char **hosts,int nhost,int *infos)
int info = pvm_delhosts(char **hosts,int nhost,int *infos)
call pvmfaddhost(host,info)
call pvmfdelhost(host,info)
```

ฟังก์ชั่นนี้จะส่งค่าคืนกลับในตัวแปรอาเรย์ infos ที่มีขนาด nhost เพื่อแสดงสถานะว่าการคำสั่งทำงานสำเร็จหรือไม่ ซึ่งถ้าไม่สำเร็จก็จะส่งค่ารหัสผิดพลาด (Error Code) ให้ตรวจสอบต่อไป ซึ่งฟังก์ชั่นเหล่านี้ใช้สร้างระบบเสมือน (Virtual Machine) เพื่อเพิ่มความยืดหยุ่นและความทนทานสำหรับแอ็พพิเคชั่นใหญ่ๆ ที่ต้องใช้กำลังการคำนวนสูงมากๆ โดยการเพิ่มเครื่องที่ใช้งาน

ตัวอย่างเช่นแอ็พพิเคชั่นโปรแกรมประเภท CAD/CAM ที่ในช่วงของการคำนวณ จะต้องขยายขนาดของตัวแปร grid อาเรย์อยู่ตลอดในช่วงรันหรือขนาดของปัญหาเพิ่มขึ้น รวมถึงในกรณีที่ต้องการเพิ่มความทนทานให้แก่แอ็พพิเคชั่นเมื่อมีบางเครื่องล่มในช่วงนั้น ด้วยการเพิ่มจำนวนเครื่องที่ใช้บนระบบ

```
int mstat = pvm_mstat( char *host )
call pvmfmstat( host, mstat )
```

ฟังก์ชั่นนี้ใช้ตรวจสอบสถานะของเครื่องที่ทำงานบนระบบเสมือน โดยระบุตัวแปรสตริง host ซึ่งเป็นชื่อของเครื่องและฟังก์ชั่นนี้สามารถส่งค่าสถานะคืนกลับได้ 3 ค่าคือ PvmOk –แสดงว่าเครื่องดังกล่าวยังทำงานอยู่, PvmNoHost –แสดงว่าขณะนี้เครื่องดังกล่าวไม่ได้อยู่บนระบบเสมือนแล้ว และ PvmHostFail-แสดงว่าเครื่องดังกล่าวอาจจะล่มได้ โดยฟังก์ชั่นดังกล่าวนี้มักจะทำงานร่วมกับ pvm_notify() เพื่อแจ้งให้โพรเซสอื่นๆบนระบบได้รับทราบและทำการปรับโครงสร้างระบบเสมือนใหม่

```
int cc = pvm_reg_tasker()
int cc = pvm_reg_hoster()
int cc = pvm_reg_rm( struct pvmhostinfo **hip )
```

ทั้งสามฟังก์ชั่นเป็นฟังก์ชั่นขั้นสูงสำหรับการจัดการทรัพยากรบนระบบ โดยฟังก์ชั่นแรกเป็นการร้องขอขึ้นทะเบียนกับ pvmd เป็น Tasker ดังนั้นเมื่อแต่ละ pvmd ได้รับเมสเส็จ DM_EXEC แทนที่จะทำการสร้างโพรเซสด้วยฟังก์ชั่น fork() หรือ execv() ก็จะให้ตัว Tasker บนระบบทำงานแทน

ซึ่งในการสร้างโพรเซสของ pvmd ทันทีที่ได้รับเมสเส็จ DM_EXEC นั้น pvmd จะทำการค้นหาตาม PATH ที่ถูกกำหนดไว้ในตัวแปร epath หรือ default path ก่อน ดังนั้นถ้ามีไฟล์ดังกล่าว pvmd ก็จะสั่งรันงานนั้นทันที สำหรับในกรณีที่มีโพรเซสขอขึ้นทะเบียนเป็นตัว Tasker บนระบบ pvmd จะส่งเมสเส็จ SM_STTASK ให้ tasker ทำงานแทนโดยมีรูปแบบเมสเส็จดังนี้

```
int tid          // of task
int flags        // as passed to spawn()
string path      // absolute path of the executable
int  argc        // number of args to process
string  argv[argc]       // args
int nenv         // number of envars to pass to task
string env[nenv]         // environment strings
```

ตัว tasker จะไม่ส่งเมสเส็จ SM_STTASKACK เพื่อแสดงการตอบรับให้ pvmd แต่จะทำการสร้างการเชื่อมต่อใหม่ระหว่าง pvmd กับงานที่ถูกสร้างขึ้น แต่ Tasker จะส่ง SM_STTASKX ให้ pvmd เมื่องานที่มันสร้างขึ้นมาได้ออกจากระบบแล้ว หรือ ในกรณีที่การสร้างงานนี้ไม่สมบูรณ์ โดยมีรูปแบบเมสเส็จดังนี้

```
int tid          // of task
int status               // the Unix exit status (from \fIwait()\fR)
int u_sec                // user time used by the task, seconds
int u_usec       // microseconds
int s_sec                // system time used by the task, seconds
int s_usec       // microseconds
```

ฟังก์ชั่นนี้มีประโยชน์สำหรับการพัฒนาโปรแกรมดีบักบนระบบซึ่งจะทำหน้าที่ควบคุมดูแลโพรเซสอื่นๆบนระบบในการอ่านและเขียนเข้าไปในพื้นที่หน่วยความจำรวมถึงการสั่งทำงานและหยุดการทำงานของ IP (Instruction Program Counter) ของโพรเซสนั้นๆ โดยโพรเซสที่ทำหน้าที่เป็นตัวดีบักโปรแกรมนี้ทำหน้าที่เป็น Parent โพรเซสอื่นๆโดยตรงของโพรเซสที่ถูกสร้าง

ฟังก์ชั่น pvm_reg_hoster() เป็นการขึ้นทะเบียนเป็น hoster เพื่อทำหน้าที่จัดการดูแลการเพิ่ม/ลดจำนวนเครื่องที่ใช้งานบนระบบเสมือน หรือทำหน้าที่ในการสั่งรัน/ยกเลิก สลาฟโพรเซส pvmd แทนที่มาสเตอร์ pvmd จัดการเอง โดยทั่วไปเมื่อมาสเตอร์ pvmd ได้รับเมสเส็จเช่น DM_ADD เพื่อ

ร้องขอเพิ่มเครื่องเข้าสู่ระบบเสมือนนั้น จะมีการตรวจดูหมายเลข IP ของเครื่องที่ถูกเพิ่มก่อน จากนั้นก็จะทำการกำหนดค่าพารามิเตอร์ต่างๆตามที่กำหนดใน Host File หรือ default parameter ก่อนจะสั่งรันโพรเซสรีโมท (เช่น rsh หรือ rexec) เพื่อสร้างสลาฟ pvmd โพรเซสบนเครื่องใหม่ ส่วนในกรณีที่มีงานที่ขอขึ้นทะเบียนเป็น hoster นั้น pvmd จะส่งเมสเส็จ SM_STHOST ซึ่งมีรูปแบบเมสเส็จดังนี้

```
int nhosts                  // number of hosts
{
        int tid             // of host
        string options      // from hostfile \fIso=\fR field
        string login        // in form ``[username@]hostname.domain"
        string command // to run on remote host
} [nhosts]
```

โดย hoster จะรันแต่ละคำสั่งที่อยู่ตามเครื่องต่างๆที่คล้ายๆกันดังนี้

$PVM_ROOT/lib/pvmd -s -d8 -nhonk 1 80a9ca95:0f5a 4096 3 80a95c43:0000

และสลาฟ pvmd จะตอบรับกลับมาด้วยรูปแบบคล้ายๆกันด้วยดังนี้

ddpro<2312> arch<ALPHA> ip<80a95c43:0b3f> mtu<4096>

ซึ่งเมื่อเพิ่มเครื่องได้สำเร็จ hoster จะส่งเมสเส็จ SM_STHOSTACK ให้แก่มาสเตอร์ pvmd โดยเมสเส็จดังกล่าวจะมีรูปแบบดังนี้

```
{
    int tid                 // of host, must match request
    string status           // result line from slave or error code
} []            // implied count
```

โดยค่า TID ที่จะตอบกลับนี้จะเป็น TID ของสลาฟ pvmd นั้นๆซึ่งจะมีการรายงานสถานะที่เกิดขึ้นจากการเพิ่มแต่ละเครื่องติดด้วยสำหรับในกรณีที่เกิดข้อผิดพลาดดังเช่น PvmCantStart หรือ PvmDSysErr และ PvmDupHost เป็นต้น แต่ในกรณีที่ไม่มีข้อผิดพลาดก็จะส่งค่า PvmOk แทน

สำหรับฟังก์ชั่น pvm_reg_rm() ต้องกำหนดพารามิเตอร์เป็นตัวแปรพอทย์เตอร์โครงสร้าง pvmhostinfo ที่ภายในโครงสร้างมีองค์ประกอบดังนี้

```
struct pvmhostinfo {
  int hi_tid;
  char *hi_name;
  char *hi_arch;
  int hi_speed; };
```

โดยฟังก์ชั่นนี้จะขอขึ้นทะเบียนการเป็นตัวจัดการทรัพยากรระบบ (Resource Manager) หรือเรียกย่อๆว่า RM ซึ่ง RM โพรเซสนี้จะคอยรอรับเมสเส็จจากงานอื่นๆบนระบบที่ได้ร้องขอการบริการเมื่อมีการเรียกใช้งานไลบารีฟังก์ชั่นบางตัว ซึ่งเมสเส็จแท็กที่จะส่งให้แก่ RM ดังนี้

Libpvm call	RM message	Normal message
pvm_addhosts()	SM_ADDHOST	TM_ADDHOST
pvm_config()	SM_CONFIG	TM_CONFIG
pvm_delhosts()	SM_DELHOST	TM_DELHOST
pvm_notify()	SM_NOTIFY	TM_NOTIFY
pvm_spawn()	SM_SPAWN	TM_SPAWN
pvm_tasks()	SM_TASK	TM_TASK
pvm_reg_rm()	SM_SCHED	TM_SCHED

โดยทั่วไปเมื่องานบนระบบเรียกใช้ฟังก์ชั่นข้างต้นไลบารี Libpvm จะสร้าง Normal เมสเส็จให้แต่ละ pvmd ช่วยจัดการแต่เมื่อมีภายในระบบมีบางงานที่ถูกขึ้นทะเบียนเป็น RM ตัว pvmd ก็จะสร้าง RM เมสเส็จและส่งไปยัง RM แทน แต่อย่างไรก็ตาม RM จำเป็นจะต้องส่งเมสเส็จแท็กเหล่านี้เพื่อตอบกลับ pvmd ซึ่งสามารถเทียบได้กับเมสเส็จที่ใช้งานทั่วไปดังนี้

RM message	Normal message
SM_EXEC	DM_EXEC
SM_EXECACK	DM_EXECACK
SM_ADD	DM_ADD
SM_ADDACK	DM_ADDACK
SM_HANDOFF	(none - change of resource manager)

และมีเมสเส็จที่ตัวจัดการระบบจะต้องส่งเพื่อแจ้งเหตุให้ระบบทราบดังนี้

RM message	ความหมาย
SM_TASKX	แจ้งเมื่องานบนระบบตกหรือออกไป
SM_HOSTX	แจ้งเมื่อมีบางเครื่องถูกลบหรือตกจากระบบ

โดยฟังก์ชั่นทั้ง 3 นี้จำเป็นจะต้องเรียกใช้ pvm_setopt(PvmResvTids, 1) เพื่อให้งานทั้งหมดบนระบบสามารถส่งเมสเส็จต่างๆที่กล่าวมาข้างต้นได้ด้วยการเข้ารหัสแบบ PvmDataDefault เพื่อให้สามารถถอดข้อมูลได้ทุกที่ในระบบ และต้องเรียกใช้ #include <pvmsdpro.h> เพื่อสามารถเรียกแท็กของเมสเส็จระบบตามที่กำหนดไว้ได้

```
int info = pvm_hostsync( int host, struct
timeval *clk, struct timeval *delta )
call pvmfhostsync( host, clksec, clkusec,
deltasec, deltausec, info )
```

ฟังก์ชั่นนี้แสดงความแตกต่างทางเวลาระหว่างเครื่องที่เป็นลอคัลกับเครื่องที่เป็นรีโมท ด้วยตัวแปรโครงสร้างเวลา timeval ที่กำหนดไว้ใน #include <sys/time.h> โดยจะให้ค่าคืนกลับในตัวแปรพอทย์เตอร์ clk ซึ่งแสดงเวลาของเครื่องที่ถูกเรียก และ delta แสดงความแตกต่างของนาฬิกาทั้งสอง

4. Message Passing

รูปแบบการส่งเมสเส็จระหว่าง 2 งานบน PVM จะไม่จำกัดขนาดหรือจำนวนเมสเส็จที่รับส่งแต่อย่างใด ทั้งนี้ขึ้นอยู่กับขนาดหน่วยความจำหลักและหน่วยความจำเสมือน (Virtual Memory) บนเครื่องนั้นๆ ซึ่ง PVM ได้เตรียมฟังก์ชั่นสำหรับการจัดการติดต่อสื่อสารบนระบบไว้ 2 ลักษณะคือ 1. Blocking –เป็นการสั่งหยุดการทำงานภายในโปรแกรมชั่วคราวเพื่อให้การรับส่งเมสเส็จก่อน โดยฝั่งผู้ส่งจะทำการ Blocking Send เพื่อให้เมสเส็จบัสเฟอร์ว่างก่อนทำการส่งครั้งต่อไป ส่วนฝั่งผู้รับจะทำ Blocking Receive เพื่อรอให้เมสเส็จดังกล่าวมาถึงก่อนจะทำงานต่อไปได้ ซึ่งเหมาะสมกับการรับส่งเมสเส็จขนาดใหญ่ 2. NonBlocking –เป็นการตรวจสอบว่าเมสเส็จที่รออยู่นั่นมาถึงหรือยังถ้ายังก็โปรแกรมทำงานต่อได้เรียกว่า NonBlocking Receive (คล้ายกับการทำ Polling นั้นเอง)

ในการรับส่งเมสเส็จบน PVM สามารถรับประกันได้ว่า ถ้า Task1 ส่งเมสเส็จ A ให้ Task2 และต่อมาส่งเมสเส็จ B อีกครั้ง เมสเส็จ A จะถึง Task2 ก่อนเมสเส็จ B นอกจากนั้นแล้วในกรณีที่ทั้งสองเมสเส็จมาถึงก่อน Task2 จะพร้อมรับแล้วเมสเส็จ A จะถูกรับใน Task2 ก่อน

การส่งเมสเส็จประกอบด้วย 3 ขั้นตอนหลักๆคือ 1.เตรียมเมสเส็จบัฟเฟอร์พร้อมสำหรับการส่งเมสเส็จ 2. ทำการบรรจุข้อมูลลงในเมสเส็จบัฟเฟอร์ 3.จัดการส่งเมสเส็จไปยังผู้รับตาม TID ที่ระบุโดยมีฟังก์ชั่นต่างๆที่เกี่ยวข้องดังนี้

```
int  bufid = pvm_initsend(int encoding)
call pvmfinitsend(encoding,bufid)
```

เนื่องจาก PVM จะสร้างเมสเส็จให้แต่ละงานสามารถใช้จัดส่งข้อมูลได้ทีละครั้ง ดังนั้นก่อนจะจัดเก็บข้อมูลลงในเมสเส็จบัฟเฟอร์ดังกล่าวจำเป็นจะต้องใช้ฟังก์ชั่นนี้ก่อนเพื่อล้างข้อมูลที่คงค้าง

ในเมสเส็จบัฟเฟอร์ดังกล่าว ซึ่งจะส่งค่า bufid คืนกลับเพื่อให้ทราบเมสเส็จที่ถูกใช้สำหรับส่งข้อมูล ณ ขณะนี้ และใช้ตัวแปรอาร์กิวเมนต์ encoding สำหรับระบุวิธีเข้ารหัสของเมสเส็จดังนี้

PvmDataDefault – เป็นการเข้ารหัส XDR (eXternal Data Representation) ที่สามารถส่งข้อมูลให้เครื่องต่างๆบนระบบที่มีสถาปัตยกรรมแตกต่างกันสามารถรับข้อมูลได้

PvmDataRaw – ไม่ทำการเข้ารหัสแต่จะนำข้อมูลดิบของเครื่องผู้ส่งบรรจุในเมสเส็จบัฟเฟอร์เลย

PvmDataInPlace- ข้อมูลจะถูกจัดเก็บไว้ที่หนึ่งก่อนเพื่อลดค่าสูญเสียในช่วงบรรจุ (packing) โดยในเมสเส็จบัฟเฟอร์จะเก็บเฉพาะตัวแปรพอยนทเตอร์ของข้อมูลและขนาด และเมื่อมีการเรียกใช้ฟังก์ชั่น pvm_send() ข้อมูลดังกล่าวจะถูกสำเนาโดยตรงจากหน่วยความจำหลักที่กำหนดไว้ ซึ่งจะลดเวลาในการสำเนาข้อมูลลงในเมสเส็จบัฟเฟอร์ได้ แต่จะไม่สามารถทำการแก้ไขข้อมูลในขณะบรรจุ รวมถึงขณะที่ส่งไปแล้วได้ ซึ่งมักจะใช้ในกรณีที่ต้องการทำการบรรจุเพียงครั้งเดียวแต่สามารถทำการจัดส่งได้หลายครั้ง

สำหรับกรณีที่ต้องมีการจัดส่งหลายๆเมสเส็จ PVM ได้เตรียมฟังก์ชั่นสำหรับสร้างเมสเส็จให้แก่แอ็พพิเคชั่นโปรแกรมระบบด้วยฟังก์ชั่นนี้

```
int  bufid = pvm_mkbuf(int encoding)
call pvmfmkbuf (encoding,bufid)
```

ฟังก์ชั่นนี้จะสร้างเมสเส็จที่ว่างขึ้นใหม่ โดยระบุวิธีเข้ารหัสเหมือนเดิม และจะทำการคืนค่า bufid เพื่อระบุหมายเลขเม็สเส็จที่สร้างขึ้นนี้ ถ้าไม่เกิดข้อผิดพลาดใดๆก็จะส่งค่าคืนกลับที่ > 0

```
int  info = pvm_freebuf(int bufid)
call pvmffreebuf(bufid,info)
```

ฟังก์ชั่นนี้จะล้างเมสเส็จหมายเลข bufid ออกจากงานทันที ซึ่งฟังก์ชั่นนี้จะใช้เมื่อเมสเส็จดังกล่าวไม่ใช้งานอีกต่อไป ดังนั้นฟังกชั่น pvm_mkbuf() สามารถสร้างเมสเส็จหมายเลข bufid อีกครั้ง

```
int  bufid = pvm_getsbuf()
call pvmfgetsbuf (bufid)
int bufid = pvm_getrbuf()
call pvmfgetrbuf(bufid)
```

เนื่องจาก PVM ออกแบบให้งานสามารถมีเมสเส็จที่ใช้รับและส่งได้เพียงอย่างละหนึ่ง (Active Send/Receive message) ดังนั้นฟังก์ชั่นดังกล่าวจะคืนค่าหมายเลขเมสเส็จที่ใช้ขณะนั้น โดย

ฟังกชั่น pvm_getsbuf() คืนค่าเมสเส็จที่ใช้ส่งเมสเส็จ ส่วน pvm_getrbuf() คืนค่าเมสเส็จที่ใช้รอรับข้อมูล

```
int  oldbuf= pvm_setsbuf(bufid)
call pvmfsetsbuf (bufid, oldbuf)
int oldbuf = pvm_setrbuf(bufid)
call pvmfgetrbuf(bufid, oldbuf)
```

ฟังก์ชั่นเหล่านี้เป็นการกำหนดให้เมสเส็จหมายเลข bufid ทำงานสำหรับการรับ/ส่งเมสเส็จ โดย pvm_setsbuf() กำหนดให้เป็นเมสเส็จสำหรับส่งข้อมูล และ pvm_setrbuf() เป็นเมสเส็จสำหรับรอรับข้อมูล ซึ่งฟังก์ชั่นเหล่านี้ใช้งานร่วมกับ pvm_mkbuf() ที่สร้างไว้หลายๆเมสเส็จ โดยจะส่งค่าคืนกลับ oldbuf เป็นหมายเลขเมสเส็จเดิมที่ถูกใช้งาน

ถ้าค่า bufid กำหนดให้เป็น 0 ใน pvm_setsbuf() หรือ pvm_setrbuf() นั้นแสดงว่าสถานะของบัฟเฟอร์ในปัจจุบันถูกจัดเก็บอยู่ และ จะไม่มีการทำงานใดๆของบัฟเฟอร์ (no active buffer) ซึ่งคุณลักษณะดังกล่าวนี้สามารถใช้เก็บสถานะปัจจุบันของเมสเส็จ เพื่อให้ ไลบรารีทางคณิตศาสตร์ หรือ กราฟฟิกต่างๆ (เป็น Third Party ของแอ็พพิเคชั่นระบบ) สามารถใช้เมสเส็จบน PVM ได้โดยไม่ไปยุ่งเกี่ยวกับบัฟเฟอร์ของแอ็พพิเคชั่นและหลังจากใช้งานเมสเส็จเสร็จแล้วบัฟเฟอร์ของแอ็พพิเคชั่นนั้นๆต้องถูก รีเซ็ทเพื่อให้ทำงานได้ใหม่อีกครั้ง

ถ้าต้องการจะส่งเมสเส็จอย่างต่อเนื่อง (forward message) โดยไม่ต้องทำการบรรจุ (repacking)ข้อมูลใหม่สามารถทำได้โดยการใช้ฟังก์ชั่นเหล่านี้ตามลำดับ

```
bufid = pvm_recv(src,tag);
oldid = pvm_setsbuf(bufid);
info = pvm_send(dst,tag);
info = pvm_freebuf(oldid);
```

การบรรจุข้อมูลลงบัฟเฟอร์เป็นขั้นตอนต่อมาหลังจากงานนั้นๆมีเมสเส็จที่ใช้สำหรับส่งข้อมูลแล้ว โดยฟังก์ชั่นแต่ละตัวของภาษา C ต่อไปนี้จะบรรจุข้อมูลอารเรย์เข้าสู่บัฟเฟอร์ของเมสเส็จที่ใช้ส่งข้อมูล (Active send buffer) ซึ่งสามารถทำการบรรจุข้อมูลได้หลายครั้งในเมสเส็จเดียวกันนี้ ดังนั้นแอ็พพิเคชั่นจะไม่มีความซับซ้อนในเรื่องของการบรรจุข้อมูล แต่แอ็พพิเคชั่นจะต้องปลดข้อมูล (unpack) ของเมสเส็จนั้นๆได้อย่างถูกต้อง (เหมือนกับตอนที่ถูกบรรจุมา)

อาร์กิวเมนต์ของแต่ละฟังก์ชั่นคือ พอยนทเตอร์ของข้อมูลตัวแรกที่ถูกบรรจุ และ nitem คือขนาดของอารเรย์ข้อมูลที่จะบรรจุลงไปและ stride คือการเว้นช่วงการบรรจุข้อมูล โดยทั่วไปจะกำหนดไว้ที่ 1 เพื่อทำการบรรจุข้อมูลเว็กเตอร์อย่างต่อเนื่อง และ 2 คือทุกๆข้อมูลจะมีการเว้นระยะ

ห่างไว้ 1 (มีการเว้นระยะข้อมูลด้วยตัว space ไว้ 1 ตัว) และค่าอื่นๆอีกเช่นกัน ซึ่งประโยชน์ของการกำหนด stride ที่ 2 เช่นการบรรจุข้อมูลที่เป็น จำนวนจินตรภาพเช่น 3+4i นั้นเอง โดยมีข้อยกเว้นกับฟังก์ชั่น pvm_str() ซึ่งตัวแปรสตริงจะมีค่า NULL ต่อท้ายอยู่แล้วจึงไม่จำเป็นต้องมี nitem หรือ stride อีก

```
int info = pvm_pkbyte( char *xp, int nitem, int stride )
int info = pvm_pkcplx( float *cp, int nitem, int stride )
int info = pvm_pkdcplx( double *zp, int nitem, int stride )
int info = pvm_pkdouble( double *dp, int nitem, int stride )
int info = pvm_pkfloat( float *fp, int nitem, int stride )
int info = pvm_pkint( int *ip, int nitem, int stride )
int info = pvm_pkuint( unsigned int *ip, int nitem, int stride )
int info = pvm_pkushort( unsigned short *ip, int nitem, int stride )
int info = pvm_pkulong( unsigned long *ip, int nitem, int stride )
int info = pvm_pklong( long *ip, int nitem, int stride )
int info = pvm_pkshort( short *jp, int nitem, int stride )
int info = pvm_pkstr( char *sp )
int info = pvm_packf( const char *fmt, ... )
```

นอกจากนี้ PVM ได้เตรียมฟังก์ชั่นบรรจุข้อมูลที่มีรูปแบบคล้ายกับ printf() ของภาษา C อีกด้วย และสำหรับภาษาฟอร์แทรนการกำหนดชนิดของข้อมูลที่จะบรรจุในตัวแปร xp ดังนี้

```
call pvmfpack( what, xp, nitem, stride, info )
```

และฟังก์ชั่นสำหรับจัดส่งข้อมูลของเมสเสจที่ใช้ขณะนั้น (Active send message) จะมีทั้งแบบ Point to Point และ Multicast ดังนี้

```
int  info = pvm_send(int tid,int msgtag)
call pvmfsend(tid,msgtag,info)
int  info = pvm_mcast(int *tids,int ntask,int msgtag)
call pvmfmcast(ntask,tids,msgtag,info)
```

ฟังก์ชั่น pvm_send() จะทำการจัดส่งเมสเส็จไปถึง tid ปลายทางและกำหนด msgtag ซึ่งเป็นเสมือนลาเบล (label) ของเมสเส็จนั้นๆ สำหรับกรณีที่มีหลายปลายทางจะใช้ pvm_mcst() แทน ซึ่งฟังก์ชั่น pvm_mcast() นี้จะกระจายเมสเส็จตามที่ระบุในตัวแปรอารเรย์ tid ขนาด ntask ยกเว้นตัวมันเอง (tid ของตัวส่งเอง)

```
int  info = pvm_psend(int tid,int msgtag,void *vp,
int cnt,int type)
call pvmfpsend(tid,msgtag,xp,cnt,type,info)
```

โดยฟังก์ชั่น pvm_send() และ pvm_psend() ใช้รูปแบบการส่งแบบ อะสิงโคนัส (Asynchronous) ซึ่งมีข้อแตกต่างจาก สิงโคนัส (Synchronous) คือในช่วงที่คำนวณบนโพรเซสเซอร์ตัวส่งจะยังคงค้างอยู่อย่างนั้นจนกว่าจะมีการจับคู่ของเมสเส็จบนโพรเซสเซอร์ตัวรับ ดังนั้นการคำนวณบนโพรเซสเซอร์ตัวส่งจะกลับคืนในทันทีที่เมสเส็จนั้นถูกเก็บอย่างปลอดภัยบนโพรเซสเซอร์ตัวรับ สำหรับฟังก์ชั่นนี้ใช้บรรจุและส่งข้อมูลอารเรย์ที่ได้ระบุไว้ในตัวแปรอาร์กิวเมนต์ type ดังนี้

PVM_STR	PVM_FLOAT
PVM_BYTE	PVM_CPLX
PVM_SHORT	PVM_DOUBLE
PVM_INT	PVM_DCPLX
PVM_LONG	PVM_UINT
PVM_USHORT	PVM_ULONG

PVM จะมีการรับเมสเส็จได้หลายวิธี โดยไม่มีการกำหนดตายตัว ตัวอย่างเช่น ผู้ส่งใช้ pvm_psend() แต่ผู้รับไม่จำเป็นต้องใช้ pvm_precv() ให้ตรงกันเสมอไป ฟังก์ชั่นต่อไปนี้สามารถเรียกใช้ในการรอรับเมสเส็จใดๆที่มาได้ไม่ว่าจะส่งมาแบบใดก็ตาม

```
int  bufid = pvm_recv(int tid,int  msgtag)
call pvmfrecv (tid,msgtag,bufid)
```

ฟังก์ชั่นนี้จะเป็นการรอรับเมสเส็จแบบ Blocking Receive โดยจะคอยจนกว่าเมสเส็จที่ประกาศไว้จาก tid ที่ระบุมาถึง ถ้าค่า msgtag หรือ tid เท่ากับ –1 แสดงว่าให้เปิดรับทุกๆเมสเส็จไม่ว่าจะมาจากต้นทางใดๆก็ตาม

```
int  bufid = pvm_nrecv(int tid,int  msgtag)
call pvmfnrecv (tid,msgtag,bufid)
```

ฟังก์ชั่นนี้จะรอรับเมสเส็จแบบ Nonblocking Receive โดยถ้าเมสเส็จที่รอนั้นยังมาไม่ถึงจะส่งค่า bufid =0 ซึ่งฟังก์ชั่นนี้สามารถเรียกใช้งานได้หลายครั้งเพื่อตรวจสอบการมาถึงของเมสเส็จดังกล่าว ก่อนการใช้ pvm_recv() กับเมสเส็จตัวเดียวกันนี้ โดยทั่วไปจะมีการสร้างเมสเส็จเพื่อเตรียมรอรับตลอดและจะนำเมสเส็จเดิมกลับมาใช้ใหม่อีกครั้งหลังจากทำการล้างเมสเส็จนั้นแล้ว หรืออาจจะเรียกใช้ฟังก์ชั่น pvm_setrbuf() แทนก็ได้

```
int  bufid = pvm_probe(int tid,int  msgtag)
call pvmfprobe (tid,msgtag,bufid)
```

ฟังก์ชั่นนี้ใช้ตรวจสอบดูว่าเมสเส็จที่กำลังรอนั้นมาถึงหรือยัง ซึ่งมันจะไม่ทำการรับเมสเส็จที่เข้ามาในทันทีเหมือน pvm_recv() และ pvm_nrecv() และฟังก์ชั่นนี้ก็สามารถเรียกใช้งานได้หลายครั้งเช่นกัน โดยจะใช้งานร่วมกับ pvm_bufinfo() เพื่อตรวจสอบรายละเอียดของเมสเส็จที่มานี้ก่อนทำการรับเมสเส็จดังกล่าว

```
int  bufid = pvm_trecv(int tid,int  msgtag,struct
timeval *tmout)
call pvmftrecv (tid,msgtag,sec,usec,bufid)
```

ฟังก์ชั่นนี้จะเหมือนกับ pvm_recv() แต่อนุญาตให้ผู้ใช้สามารถกำหนดช่วงเวลาในการรอรับเมสเส็จเพื่อใช้ตรวจสอบความล้มเหลวหรือความผิดพลาดของเมสเส็จที่รอรับอยู่ได้ (ป้องกันการ Block อยู่ตลอดเวลา) ซึ่งการกำหนดระยะเวลานานมากๆ (เกือบเป็น INFINITY) ฟังก์ชั่นนี้จะทำงานเหมือน pvm_recv() แต่ถ้ากำหนดเป็น 0 ฟังก์ชั่นนี้จะทำงานเหมือน pvm_nrecv()

```
int  info = pvm_bufinfo(int bufid,int *bytes,int
*msgtag,int *tid)
call pvmfbufinfo (bufid,bytes,msgtag,tid,info)
```

ฟังก์ชั่นนี้จะให้ค่าคืนกลับของ bytes (จำนวนไบต์ของข้อมูล) , msgtag และ tid ของเมสเส็จต้นทางที่ส่งมา ด้วยการกำหนดอาร์กิวเมนต์ bufid จากฟังก์ชั่นที่รอรับเมสเส็จต่างๆ

```
int  info = pvm_precv (int tid,int msgtag,void *vp,
int cnt,int type,int *rtid,int *rtag,int *rcnt)
call pvmfpsend(tid,msgtag,xp,cnt,type,rtid,rtag,
rcnt,info)
```

ฟังก์ชั่นนี้รวมการ Blocking Receive และ การถอดรหัสข้อมูล (UNPACKING) ของเมสเส็จบัฟเฟอร์ที่รับเข้ามา ซึ่งจะไม่ส่งค่าคืนกลับเป็น bufid เหมือนฟังก์ชั่นรอรับเมสเส็จอื่นๆ แต่จะส่งค่าคืนกลับเป็น TID,msgtag และ cnt แทน

```
int info = pvm_unpackf( const char *fmt, ... )
int info = pvm_upkbyte( char *xp, int nitem, int stride)
int info = pvm_upkcplx( float *cp, int nitem, int stride)
int info = pvm_upkdcplx( double *zp, int nitem, int stride)
int info = pvm_upkdouble( double *dp, int nitem, int stride)
int info = pvm_upkfloat( float *fp, int nitem, int stride)
int info = pvm_upkint( int *ip, int nitem, int stride)
int info = pvm_upkuint( unsigned int *ip, int nitem, int stride )
int info = pvm_upkushort( unsigned short *ip, int nitem, int stride )
int info = pvm_upkulong( unsigned long *ip, int nitem, int stride )
int info = pvm_upklong( long *ip, int nitem, int stride)
int info = pvm_upkshort( short *jp, int nitem, int stride)
```

ฟังก์ชั่นเรานี้ใช้ถอดข้อมูลในเมสเส็จบัฟเฟอร์ที่รับเข้ามาซึ่งควรให้ตรงกันกับฟังก์ชั่นที่บรรจุข้อมูลลงเมสเส็จนั้นๆด้วย ตั้งแต่เลือกใช้ฟังก์ชั่นตรงกับชนิดของข้อมูลที่รับเข้ามา ,ขนาด (nitem) และ stride

```
int info = pvm_sendsig( int tid, int signum )
call pvmfsendsig( tid, signum, info )
```

ฟังก์ชั่นนี้เป็นการส่งสัญญาณ signum ให้แก่โพรเซสต่างๆตามที่ระบุใน tid โดยถ้าส่งสำเสร็จก็จะให้ค่าคืนกลับเป็น 0 แต่ถ้ามี Error ก็จะให้ค่าคืนกลับ < 0 ซึ่งฟังก์ชั่นนี้ควรใช้กับโปรแกรมเมอร์ที่มีประสบการณ์ในการเขียนโปรแกรมส่งสัญญาณต่างๆบน Unix โพรเซส

```
int info = pvm_notify( int what, int msgtag, int
cnt, int *tids )
call pvmfnotify( what, msgtag, cnt, tids, info )
```

ฟังก์ชั่นนี้ใช้แจ้งเหตุการณ์ต่างๆที่เกิดขึ้นบนระบบเสมือนด้วยพารามิเตอร์ what ซึ่งมีค่าดังนี้

PvmTaskExit	เมื่อจบงานบนระบบหรืองานบนระบบล่มไป
PvmHostDelete	เมื่อมีบางเครื่องในระบบถูกลบหรือล่มไป
PvmHostAdd	เมื่อมีการเพิ่มเครื่องเข้าไปในระบบ

msgtag เป็นแท็กที่ใช้สำหรับการแจ้งเหตุ ส่วน cnt จะขึ้นอยู่กับ what กำหนดอะไรเช่น PvmTaskExit และ PvmHostDelete ค่า cnt จะระบุความยาวของอาเรย์ tids แต่ใน PvmHostAdd จะใช้ระบุจำนวนครั้งที่มีการแจ้งเหตุ และตัวแปรอาเรย์ tids PvmTaskExit และ PvmHostDelete จะเก็บ Tid ของงานที่ออกจากระบบหรือ pvmd TID ที่ถูกลบออกไป ซึ่งใน PvmHostAdd ไม่ใช้ ซึ่งถ้าไม่สามารถแจ้งเหตุได้ค่า info < 0

5. Group Library Functions

ความสามารถในการรวมกลุ่มของโพรเซสบนระบบแบบไดนามิก (Dynamic process group functions) นี้ถูกสร้างบนแกนหลักของ PVM โดยแยกพัฒนาไลบรารีขึ้นมาใช้เองต่างหากคือ libgpvm3.a ดังนั้นการนำฟังก์ชั่นเหล่านี้มาใช้จำเป็นต้องเชื่อมไลบรารีนี้ก่อน โดยฟังก์ชั่นเหล่านี้จะไม่ถูกจัดการโดย pvmd แต่จะใช้ Group Server เข้ามาจัดการ ซึ่งจะเริ่มทำงานโดยอัตโนมัติเมื่อกลุ่มแรกถูกตั้งบนระบบ (ด้วยการันโปรแกรม pvmgs ตามเครื่องต่างๆที่งานนั้นๆใช้ฟังก์ชั่นเหล่านี้) มีหลายคนโต้แย้งถึงวิธีการจัดการกลุ่มในระบบเชื่อมต่อของการส่งผ่านเมสเส็จ (message passing interface) เพื่อให้ได้ทั้งประสิทธิภาพ (efficiency) และความน่าเชื่อถือ (Reliability) ซึ่งมีข้อดีและข้อเสียกันระหว่างการเลือกจัดกลุ่มทั้งแบบสถิตย์และไดนามิก โดยบางคนก็ให้ความเห็นว่าเฉพาะงานในกลุ่มเท่านั้นที่สามารถจะเรียกใช้ฟังก์ชั่นเหล่านี้ได้

และเพื่อยังคงรักษาแนวคิดหลักของ PVM ฟังก์ชั่นกลุ่มเหล่านี้จึงถูกออกแบบให้ง่ายต่อการใช้งานและก่อให้เกิดประสิทธิภาพมากสุดแก่ผู้ใช้ งานบนระบบสามารถเชื่อม (Join) และออกนอกกลุ่มได้ (Leave) ตลอดเวลาโดยไม่จำเป็นต้องแจ้งให้ งานอื่นๆในกลุ่มเดียวกันทราบ ซึ่งไม่เกิดผลกระทบใดๆกับงานอื่นๆเช่นกัน งานใดๆบนระบบสามารถส่งเมสเส็จกระจายให้แก่กลุ่มที่มันไม่ได้

เป็นสมาชิกได้ โดยมีข้อยกเว้นที่บางฟังก์ชั่น pvm_lvgroup(), pvm_barrier() และ pvm_reduce() จะต้องถูกเรียกใช้ได้เฉพาะงานที่เป็นสมาชิกของกลุ่มนั้นๆเท่านั้น

```
int inum = pvm_joingroup(char *group)
int info = pvm_lvgroup(char *group)
call pvmfjoingroup(group,inum)
call  pvmflvgroup(group,info)
```

ฟังก์ชั่นเหล่านี้ให้ งาน หนึ่งๆเชื่อม (Join) และ ออก (leave) จากกลุ่มที่ผู้ใช้กำหนดได้ ฟังก์ชั่น pvm_joingroup() จะสร้างกลุ่มตามชื่อที่ระบุ ซึ่งถ้าไม่เกิดข้อผิดพลาดใดๆจะคืนค่าเลขจำนวนเต็ม (ตั้งแต่ 0 เป็นต้นไป) ในการบ่งบอกถึงหมายเลขสมาชิกในกลุ่มนั้นๆ และถ้ามีหมายเลขที่ว่างอยู่ก็จะถูกกำหนดให้แก่งานที่เข้ามาใหม่ (ในกรณีที่บางงานในกลุ่มออกไป แล้วมีงานใหม่เข้ามาในกลุ่ม งานใหม่จะได้หมายเลขว่างที่มีค่าน้อยสุดก่อน) อย่างไรก็ตามข้อควรระวังก็คือถ้ามีแต่งานออกนอกกลุ่มและไม่มีงานใหม่เข้ามาจะทำให้เกิดช่องว่างได้ ทั้งนี้ผู้ใช้จะต้องจัดการดูแลหมายเลขดังกล่าวให้ต่อเนื่องตามความต้องการของอัลกอริทึมที่ใช้งานอยู่

```
int tid = pvm_gettid(char *group,int inum)
int inum = pvm_getinst(char *group,int tid)
int size = pvm_gsize(char *group)
call  pvmfgettid(group,inum,tid)
call  pvmfgetinst(group,tid,info)
call  pvmfgsize(group,size)
```

ฟังก์ชั่น pvm_gettid() จะให้ค่า tid ของโพรเซสบน PVM คืนกลับตามอาร์กิวเมนต์ชื่อกลุ่ม (char *group) และ หมายเลขสมาชิก (inum) ฟังก์ชั่น pvm_getinst() จะให้หมายเลขสมาชิกภายในกลุ่มของ tid ส่วนฟังก์ชั่น pvm_gsize() จะให้จำนวนสมาชิกที่อยู่ในกลุ่มตามที่ระบุไว้

```
int  info = pvm_barrier(char *group,int count)
call pvmfbarrier(group,count,info)
```

เมื่อเรียกใช้ฟังก์ชั่นนี้โพรเซสจะถูก Block จนกว่าจะมีจำนวนสมาชิกในกลุ่มครบตามจำนวน count ที่ระบุไว้ อย่างไรก็ตามจำนวน count ควรกำหนดให้เท่ากับจำนวนโพรเซสที่มีในกลุ่มพอดี เนื่องจาก เป็นเรื่องยากที่จะทราบจำนวนสมาชิกในกลุ่ม ณ ขณะนั้น และอาจจะเกิดข้อผิดพลาดได้

ง่ายถ้าโพรเซสที่เรียกใช้ฟังก์ชั่นดังกล่าวไม่ใช่สมาชิกในกลุ่ม หรือ สมาชิกในกลุ่มเรียกฟังก์ชั่นนี้ด้วยจำนวน count=4 แต่สมาชิกตัวอื่นเรียกด้วยค่า count=5

```
int  info = pvm_bcast(char *group,int msgtag)
call pvmfbcast(group,msgtag,info)
```

ฟังก์ชั่นนี้จะกระจายเมสเส็จที่กำหนดแท็กลาเบล msgtag ไปยังสมาชิกทั้งหมดในกลุ่มที่ระบุไว้ยกเว้นตัวมันเอง ถ้ามันเป็นสมาชิกภายในกลุ่มด้วย และถ้างานใดๆพยายามเข้ากลุ่มในระหว่างที่กระจายเมสเส็จอยู่ งานนั้นจะไม่ได้รับเมสเส็จที่กระจายด้วยฟังก์ชั่นนี้ และถ้างานใดออกจากกลุ่มระหว่างที่กระจายเมสเส็จนี้ สำเนาของเมสเส็จนี้จะหยุดส่งให้งานนั้นทันที

```
int info = pvm_scatter( void *result, void *data, int
count, int datatype, int msgtag, char *group, int
rootginst)
call pvmfscatter(result, data, count, datatype, msgtag,
group, rootginst, info)
```

ฟังก์ชั่นนี้จะกระจายข้อมูล data จากสมาชิกที่เป็นแหล่งกระจายงาน (มักเป็น root หรือสมาชิกหมายเลข 0 ของกลุ่มนั้นเอง) ไปยังทุกๆสมาชิกในกลุ่มและรวมตัวมันเองด้วย โดยทุกๆสมาชิกจะเรียก ฟังก์ชั่นนี้พร้อมกัน ซึ่งผู้รับแต่ละตัวจะได้ข้อมูลอาเรย์แต่ละส่วนจากแหล่งจ่ายไปเก็บในตัวแปรอาเรย์ result มันเอง โดยทุกๆสมาชิกหมายเลข i ใดๆของกลุ่มจะได้รับข้อมูลเริ่มต้นที่ตำแหน่ง (offset) ที่ i*count เป็นต้นไปจากจุดเริ่มต้นของตัวแปร data อาเรย์

```
int info = pvm_gather( void *result,  void *data,
                int count, int datatype, int msgtag,
                char *group, int rootginst)
call pvmfgather(result, data, count, datatype,
                        msgtag, group, rootginst, info)
```

ฟังก์ชั่นนี้เป็นการส่งเมสเส็จจากแต่ละสมาชิกในกลุ่มไปยังแหล่งกำเนิด (root) ของกลุ่ม โดยทุกๆสมาชิกในกลุ่มจำเป็นต้องเรียกฟังก์ชั่นนี้พร้อมๆกันด้วย ซึ่งผู้ส่งแต่ละตัวจะส่งชุดข้อมูล data ที่มีความยาวเท่ากับ count ของชนิดข้อมูลที่ระบุ datatype ให้ที่ root ทำการเก็บรวบรวมเมสเส็จดัง

กล่าวไว้ในตัวแปรอารเรย์ result นั้นคือที่แหล่ง root จะเปิดรับข้อมูลแต่ละชนิดข้อมูลที่มีขนาด count จากแต่ละสมาชิกหมายเลข i ใดๆไปเก็บในตัวแปร result โดยเริ่มต้นที่ตำแหน่ง (offset) i*count เป็นต้นไปนับจากจุดเริ่มต้นของตัวแปรอารเรย์ result โดยค่า rootinst คือหมายเลขของสมาชิกในกลุ่มนั้นๆที่กำหนดให้เป็นแหล่งจ่ายงาน

ในการใช้ฟังก์ชั่นทั้งสองนี้จะต้องระวังเรื่องของการเรียงลำดับซึ่งในภาษา C เก็บอารเรย์หลายมิติตามลำดับแถว โดยทั่วไปจะเริ่มนับข้อมูลอารเรย์ที่ตำแหน่ง 0 ก่อนและในส่วนภาษาฟอรแทรนเก็บอารเรย์ตามลำดับคอลัมน์ ซึ่งจะใช้ 1 เป็นตำแหน่งเริ่มต้น

ข้อควรระวังของฟังก์ชั่น pvm_gather() คือการส่งไม่ได้ถูก Block ไว้ดังนั้นถ้างานที่เรียก pvm_gather นั้นออกจากกลุ่มไปก่อนที่แหล่งกำเนิด (root) จะเรียกฟังก์ชั่นนี้ก็อาจทำให้เกิดข้อผิดพลาดขึ้นได้ง่าย

```
int info = pvm_reduce( void (*func)(), void *data, int
          count, int datatype, int msgtag, char *group,
          int rootginst)
call pvmfreduce(func, data, count, datatype,
     msgtag, group, rootginst, info)
```

ฟังก์ชั่น pvm_reduce() จะสั่งให้เกิดคำนวณคณิตศาสตร์ทั่วถึง (Global arithmetic operation) ข้ามกลุ่ม ตัวอย่างเช่น GLOBAL SUM หรือ GLOBAL MAX ผลลัพธ์ของการทำงานฟังก์ชั่นนี้จะเก็บอยู่ที่ root ตามที่กำหนดไว้ PVM ได้เตรียม func มาตรฐานไว้ 4 ฟังก์ชั่นให้ผู้ใช้เรียกใช้งานดังนี้

PvmMax

PvmMin

PvmSum

PvmProduct

โดยการลดทอนการคำนวณของฟังก์ชั่นนี้จะกระทำกับข้อมูลที่เก็บในตัวแปรอารเรย์ data เช่น ถ้าอารเรย์ข้อมูล data ประกอบด้วยเลขจำนวนจริง 2 ค่า และอาร์กิวเมนต์ของ func คือ PvmMax แล้วผลลัพธ์ของทั้ง 2 ค่าคือ ค่ามากสุดของสมาชิกในกลุ่มตามเลขจำนวนแรก และค่ามากสุดของสมาชิกในกลุ่มตามเลขที่สอง

อย่างไรก็ตามฟังก์ชั่น pvm_reduce() นี้ผู้ใช้สามารถเขียนขึ้นมาเองก็ได้เพื่อขยายขีดความสามารถของการคำนวณแบบโกลบอล ซึ่งสามารถดูตัวอย่างโปรแกรม PVM ได้ที่ PVM_ROOT/examples/gexamples แต่ข้อควรระวังของ ฟังก์ชั่น pvm_reduce() นี้คือฟังก์ชั่นนี้ไม่มี

การ Block ดังนั้นเวลาที่งานในกลุ่มเรียกใช้ฟังก์ชั่นนี้และออกจากกลุ่มก่อนที่ root จะเรียกฟังก์ชั่นนี้ อาจทำให้เกิดข้อผิดพลาดได้

อย่างไรก็ตามแม้นว่าการจัดกลุ่มแบบไดนามิกทำให้เกิดความยืดหยุนของการใช้งาน และ เพิ่มความทนทานบนระบบในการเพิ่ม/ลดจำนวนงานที่ทำในกลุ่มตามภาระงานที่เกิดบนเครื่องนั้นๆได้ แต่การจัดการรวมกลุ่มแบบไดนามิกบนระบบ PVM นั้นจะใช้ Group Server จะมีศูนย์กลางในการจัดการ (centralize) อยู่ที่เดียว ดังนั้นผลที่ตามมาคือทำให้เกิดปัญหาคอขวดของการติดต่อสื่อสารแบบรวมกลุ่ม (Collective communication – ที่มีการส่งเมสเส็จจากหลายผู้ส่งและผู้รับ ในเวลาเดียวกัน) รวมถึงการอะสิงโคนัส (Asynchronous) ระหว่างโพรเซสในกลุ่มที่มักทำให้เกิดข้อผิดพลาดง่าย

ในปัจจุบันห้องแล็บ ORNL พยายามเพิ่มขีดความสามารถของ PVM ในการรวมกลุ่มแบบคงที่ (Static Group) ซึ่งจะเหมาะสมกับหลายๆอัลกอริทึม/แอ็พพิเคชั่นระบบที่ใช้การสิงโคนัส (synchronize) แทน ซึ่งได้มีการทดสอบหลายครั้งในการทำงานระหว่างแบบไดนามิก และ คงที่ และมักจะมีข้อสรุปที่ตรงกันว่าการรวมกลุ่มแบบคงที่จะให้ประสิทธิภาพที่ดีกว่า โดยเฉพาะเรื่องของการกระจายข่าวสารอย่างทั่วถึง (Broadcasting) ระหว่างโพรเซสภายในกลุ่ม ทั้งนี้เนื่อง ผู้ส่งจะต้องทราบรายชื่อของทุกๆผู้รับในกลุ่มจาก Group Server ส่วนกลางก่อนจะกระจายเมสเส็จดังนั้นทำให้ค่า Round Trip สูง

6. การเลือกวิธีติดตั้งและดูค่า (Setting and Getting Option)

```
int oldval = pvm_setopt( int what, int val )
call pvmfsetopt( what, val, oldval )
```

ฟังก์ชั่นนี้สำหรับการกำหนดตัวเลือกต่างๆที่ระบบ PVM ให้งานบนระบบสามารถทำได้ และจะส่งค่าคืนกลับคือค่าเก่า oldval ที่เคยตั้งไว้ โดยมีให้ติดตั้งไว้หลายตัวเลือกเช่นกันดังนี้

Option	Value	ความหมาย
PvmRoute	1	กำหนดนโยบายการค้นหาเส้นทาง
PvmDebugMask	2	ทำการมาสต์ในไลบรารีฟังก์ชั่นเพื่อการแก้ไขโปรแกรม
PvmAutoErr	3	ให้รายงานข้อผิดพลาดโดยอัตโนมัติ
PvmOutputTid	4	ให้ Stdout ของงานไปยัง TID ที่ระบุไว้
PvmOutputCode	5	แสดงผลของ msgtag
PvmTraceTid	6	ไล่ติดตามข้อมูลของโพรเซสลูก
PvmTraceCode	7	ไล่ติดตาม msgtag

PvmFragSize	8	กำหนดขนาดของเมสเส็จ
PvmResvTids	9	อนุญาติให้ใช้งานเมสเส็จแท็กที่จองไว้ของระบบ
PvmSelfOutputTid	10	กำหนด Stdout ของตัวเอง
PvmSelfOutputCode	11	แสดงผลเมสเส็จแท็กของตัวเอง
PvmSelfTraceTid	12	ไล่ติดตาม Tid ของตัวเอง
PvmSelfTraceCode	13	ไล่ติดตาม msgtag ของตัวเอง
PvmShowTids	14	ให้ฟังก์ชั่น pvm_catchout แสดงค่า tid ออกมา
PvmPollType	15	นโยบายรอรับเมสเส็จ (ใช้เฉพาะบนเครื่อง Share Memory)
PvmPollTime	16	ให้เมสเส็จทำการโพรลิ่งตามช่วงเวลาที่กำหนดไว้

ตัวเลือกส่วนใหญ่ที่ใช้กันมากก็คือ PvmRoute ซึ่งสามารถกำหนดค่าในตัวเลือกนี้ได้ 3 ค่าหลักๆคือ

PvmDontRoute	1	ไม่ส่งร้องขอหรือให้สร้างการเชื่อมต่อ
PvmAllowDirect	2	ไม่ให้ทำการร้องขอแต่สามารถสร้างการเชื่อมต่อได้ (Default)
PvmRouteDirect	3	ให้ทำการร้องขอและสามารถสร้างการเชื่อมต่อได้

การเชื่อมต่อโดยตรงระหว่างงานจะไม่สามารถทำงานได้บนระบบ UNIX ขนาดใหญ่เมื่อมีจำนวนงานที่มากกว่า 60 task ซึ่ง PVM จะติดตั้งค่านี้คืนกลับโดยอัตโนมัติ แม้ว่าฟังก์ชั่นนี้สามารถจะเรียกใช้ได้หลายครั้งระหว่างการทำงานของโปรแกรมประยุกต์เพื่อกำหนดให้มีการติดต่อสื่อสารกันโดยตรงระหว่าง task กับ task แต่ต้องใช้หลังจากเรียกใช้ฟังก์ชั่น pvm_mytid()

โดยฟังก์ชั่นนี้จะให้ค่าเก่าที่ได้ถูกติดตั้งไว้ก่อนหน้า (oldval) ส่งคืนกลับ ซึ่งถ้าหากเกิดข้อผิดพลาดในการติดตั้งใดๆก็ตามก็จะส่งค่าผิดพลาดคืนกลับแทน สำหรับรายละเอียดของแต่ละตัวเลือกนั้นสามารถดูได้จากคู่มือเอกสารประกอบการใช้งานบน PVM หรือจาก help ไฟล์ที่ติดมาให้

```
int  info = pvm_perror(char *msg)
call  pvmfperror(msg,info)
```

ฟังก์ชั่นนี้แจ้งให้ทราบถึงข้อผิดพลาดที่เกิดขึ้นคล้ายกับ stderr ของภาษา C โดยการส่งเมสเส็จข้อความ *msg ไปเก็บไว้ยังไฟล์ /tmp/pvml.<uid> บนเครื่องหลัก (เครื่องที่ master pvmd รันอยู่)

2.3.2 เพิ่มเติมฟังก์ชั่นขั้นสูงของ PVM (Addition advance PVM Function)

ฟังก์ชั่นขั้นสูงเหล่านี้มีอยู่ใน PVM 3.4 ขึ้นไปเพื่ออำนวยความสะดวกแก่ผู้พัฒนาแอ็พพิเคชั่น โปรแกรมระบบที่ซับซ้อนสูงและขยายขีดความสามารถของระบบ เช่น โปรแกรมควบคุมงาน, Debuger, Tracer, agent และ ตัวจัดงานระบบ เป็นต้น ฟังก์ชั่นขั้นสูงเหล่านี้มีดังต่อไปนี้

```
C       int ctx = pvm_newcontext( void )
        int old_ctx = pvm_setcontext( int new_ctx )
        int info = pvm_freecontext( ctx )
        int ctx = pvm_getcontext( void )
Fortran
        call pvmfnewcontext( ctx )
        call pvmfsetcontext( new_ctx, old_ctx )
        call pvmffreecontext( ctx, info )
        call pvmfgetcontext( ctx )
```

แนวคิดของ context นำมาแก้ปัญหาของการพัฒนาแอ็พพิเคชั่นระบบโดยเฉพาะ Third party parallel library ซึ่งเป็นเรื่องยุ่งยากมากในการกำหนดเมสเส็จแท็ก (message tag) สำหรับจัดการติดต่อสื่อสารบนแอ็พพิเคชั่นขนาดใหญ่ดังนั้นผู้เขียนไลบารีคู่ขนานนี้จะต้องแน่ใจได้ว่าจะไม่เกิดการแทรกทรอด (inference) กับแอ็พพิเคชั่น ตัวอย่างเช่นถ้ามีการพัฒนาไลบารีลูทีนทางคณิตศาสตร์เป็น Third party ให้กับแอ็พพิเคชั่นที่รันปัญหาที่มักเกิดขึ้นเสมอคือถ้าในไลบารีดังกล่าวพยายามเปิดรับทุกๆเมสเส็จแท็ก หรือ มีการใช้แท็กที่เหมือนกันกับแอ็พพิเคชั่น ผลที่ตามมาก็คืองานต้นทางอาจส่งเมสเส็จให้ที่ไลบารีนั้นแทน (ขึ้นอยู่กับว่าช่วงนั้นมีการสั่งรอรับเมสเส็จก่อนกัน) ซึ่งเป็นสาเหตุให้แอ็พพิเคชั่นเกิดการล่มได้

context จะเป็นสิ่งที่ระบบสร้างขึ้นแทนการกำหนดขึ้นเองของผู้พัฒนาโปรแกรม ทำให้สามารถแยกความแตกต่างของเมสเส็จทางตรรกะได้ (Logically) โดยมีฟังก์ชั่นที่เกี่ยวข้องกับ context ดังนี้ 1.ฟังก์ชั่น pvm_newcontext() จะให้ค่า context ใหม่แทนค่า context ที่ถูกกำหนดไว้ก่อนหน้า 2.pvm_setcontext() สั่งเปลี่ยนค่า context ที่ใช้ก่อนหน้าให้เป็นค่าใหม่ โดยจะให้ค่าเก่าคืนกลับ 3.pvm_freecontext() เป็นการสั่งลบ context นั้นออกจากระบบ และ 4.pvm_getcontext() จะคืนค่า context ที่ใช้อยู่ ณ ปัจจุบัน (การทำงานของ context ถ้ามองอย่างง่ายๆก็เหมือนกับการเปลี่ยน mode การทำงานของอุปกรณ์ไฟฟ้าบางตัวนั้นเอง)

```
int mhid = pvm_addmhf( int src, int tag, int ctx, int (*func)(int mid) )
int info = pvm_delmhf( int mhid )
```

PVM เพิ่มความสามารถให้มีการจัดการเมสเส็จที่รับเข้ามา (message handler) เพื่อใช้สำหรับการควบคุมระบบตามสภาพแสดล้อมต่างๆและความสามารถในการถอดใส่ (Plug-in) เครื่องมือในเลเยอร์กลาง (middle layer) ของระบบที่เรียกว่า GPM (Generalized Plug-In Machine) โดยฟังก์ชั่นเหล่านี้ทำงานคล้ายกับ Active Message (Active Message คือแนวคิดของการนำ code ฝั่งในเมสเส็จเพื่อให้เครื่องที่ปลายทางสามารถทำงานตาม code นั้นๆได้ แต่เดิมเมสเส็จแทนที่จะส่งเฉพาะข้อมูลเพียงอย่างเดียว) โดยฟังก์ชั่นเหล่านี้จะอำนวยความสะดวกในการสร้าง event-driven code (หรือ Active ให้เกิดการทำงานตาม code นั้นๆ) ซึ่งง่ายกว่าระบบเมสเส็จเดิมที่ใช้กันอยู่ ตัวอย่างเช่น Active Document จะใส่ code การจัดการเอาไว้แก้ปัญหาเอกสารไม่ตรง version กันทำให้ไม่เสียเวลาลองเปิดเอกสารนั้น

ฟังก์ชั่น pvm_addmhf() เป็นการระบุฟังก์ชั่นที่จะถูกเรียกใช้ทันทีเมื่อ libpvm สำเนาเมสเส็จที่กำหนด header ฟิวด์ของ src,tag และ ctx ตรงกับที่ได้ระบุไว้ในอาร์กิวเมนต์ ส่วน ฟังก์ชั่น pvm_delmhf() ทำการเคลียร์ฟังก์ชั่นที่ระบุไว้ใน pvm_addmhf() แต่อย่างไรก็ตามฟังก์ชั่นเหล่านี้จะมีข้อจำกัดอยู่บ้างในการเรียกใช้ฟังก์ชั่นต่างๆของ PVM ซึ่งจะเรียกใช้ได้เฉพาะฟังก์ชั่นของการส่งผ่านเมสเส็จเท่านั้น ยกเว้นฟังก์ชั่นที่ใช้ในการรับเมสเส็จ และ ฟังก์ชั่นการทำมัลติคาสติ้ง

```
int index = pvm_putinfo( char *name, int bufid, int flags )
int bufid = pvm_recvinfo( char *name, int index, int flags )
int info = pvm_delinfo( char *name, int index, int flags )
int info = pvm_getmboxinfo( char *pattern, int *nclasses,
struct pvmmboxinfo **classes )
int cc = pvm_insert( char *name, int index, int data )
int cc = pvm_delete( char *name, int index )
int cc = pvm_lookup( char *name, int index, int *data )
```

แนวคิดของการสร้างชื่อให้เป็นที่รู้จัก (Well know name) ทำให้ง่ายในการอ้างอิงถึงสิ่งต่างบนระบบ เช่นในระบบเมล์บนระบบปฏิบัติการ UNIX ใช้ socket เบอร์ 25 ซึ่งเป็นที่รู้จักดีว่าเป็นพอร์ทสำหรับการเปิดซอกเก็ตเชื่อมต่อ (binding) กับโปรแกรมเมล์ Server เป็นต้น สำหรับฟังก์ชั่นเหล่านี้บนระบบ PVM ใช้เพื่อกำหดชื่อบริการต่างๆ (Name Service) ที่จัดเตรียมไว้ ทั้งนี้เพื่อให้โปรแกรมระบบสามารถมองผ่านชื่อนั้นเพื่อได้รับข่าวสาร (information) ต่างๆที่จำเป็น ซึ่ง "well know name" เป็นเสมือน key ในระบบฐานข้อมูลสารสนเทศของระบบ

ฟังก์ชั่น pvm_putinfo()จะทำการแทรก (insert) ข่าวสารต่างๆที่เก็บอยู่แล้วในเมสเส็จหมายเลข bufid ให้ไปเก็บฐานข้อมูลระบบ (หรือเพิ่มจำนวนเรคอร์ด) ส่วนฟังก์ชั่น pvm_recvinfo() จะทำ

การเรียกเมสเส็จที่เก็บในฐานข้อมูลระบบออกมา สำหรับฟังก์ชั่นนี้จะคล้ายกับฟังก์ชั่น pvm_recv() แต่ฟังก์ชั่นเหล่านี้จะเป็นการติดต่อสื่อสารแบบทางเดียว (one-side communication) ซึ่งก็คือฐานข้อมูลระบบนั้นเอง ส่วนฟังก์ชั่น pvm_delinfo() และ pvm_delete() เป็นคำสั่งที่ใช้ลบเมสเส็จออกจากเรคอร์ดฐานข้อมูลระบบนั้นเอง โดยการระบุชื่อตารางและหมายเลข index ของเรคอร์ดนั้นๆ สำหรับฟังก์ชั่น pvm_getmboxinfo() จะให้รายละเอียดตามชื่อตารางที่มีบนฐานข้อมูลระบบ ส่วนฟังก์ชั่น pvm_lookup() ใช้สำหรับดูข้อมูลที่เก็บในตารางซึ่งใช้ควบคู่กับ pvm_insert() ในกรณีที่ข้อมูลที่เก็บมีฟิวด์เดียวเฉพาะเลขจำนวนเต็ม

ฟังก์ชั่นเหล่านี้สร้างให้เกิดการแชร์ข้อมูลบนระบบกระจาย (Distributed Shared Memory) ที่แต่ละเครื่องบนระบบกระจายหน่วยความจำสามารถรับทราบข้อมูลของเครื่องต่างๆได้โดยกำหนด "Well know name" แต่อย่างไรก็ตาม การติดต่อสื่อสารบนระบบด้วยช่องทางนี้จะเกิดค่าสูญเสียสูง ดังนั้นจึงจำเป็นต้องออกแบบการให้ถูกต้องตรงกับการใช้งานจริง จะช่วยให้ฟังก์ชั่นเหล่านี้ทำงานได้อย่างมีประสิทธิภาพ

```
int (*old)() = pvm_recvf( int (*new)( int bufid, int tid, int tag ))
```

ฟังก์ชั่นนี้เพิ่มขีดความสามารถในการรับเมสเส็จของ PVM โดยให้ผู้ใช้สามารถเขียนฟังก์ชั่นการตรวจสอบเมสเส็จที่รับเข้ามาแทนการใช้ฟังก์ชั่นที่ PVM เตรียมไว้ให้ โดยฟังก์ชั่นเหล่านี้ใช้งานร่วมกับ pvm_recv() , pvm_nrecv() และ pvm_probe() เพื่อสั่งให้รอรับเมสเส็จที่มาถึง ซึ่งฟังก์ชั่นนี้ให้ผู้ใช้กำหนดอาร์กิวเมนต์เป็นชื่อของฟังก์ชั่นที่ประกาศอาร์กิวเมนต์ bufid,tid,tag ไว้แล้ว แต่อย่างไรก็ตามการใช้งานฟังก์ชั่นนี้ค่อนข้างจะซับซ้อนมาก ต้องเป็นผู้ที่เข้าใจภาษา C อย่างลึกซึ้งและเข้าใจรูทีนฟังก์ชั่นของการรับเมสเส็จเป็นอย่างดี (ฟังก์ชั่นนี้ถูกออกแบบให้คล้ายกับการดักจับ สัญญาณบนระบบ)

```
int nfds = pvm_getfds( int **fds )
```

เนื่องจากงานบนระบบ PVM จะใช้ ซ็อกเก็ต ในการติดต่อสื่อสารกับงานอื่นๆรวมถึง PVMD ผ่านไลบรารี libpvm ฟังก์ชั่นนี้ทำให้ทราบจำนวนของ File Descriptor ของ ซ็อกเก็ตที่ใช้งานอยู่เพื่อทำการรอรับ input จากเมสเส็จของ PVM หรือจากที่อื่นๆ ตัวอย่างที่เห็นได้ชัดเช่นโปรแกรม PVM Console ที่คอยรอรับ input จาก คีย์บอร์ด และ เมสเส็จที่แจ้งเหตุมา โดยจะคอยทำการโพรลิ่ง (Polling) จาก INPUT ทั้งสองสลับกันคล้าย Multiplexing อย่างไรก็ตามวิธีนี้จะทำให้เสีย cpu cycle สูงมากกว่าโดยจะใช้คู่กับคำสั่ง select() ซึ่งเป็น system call ที่ถูกใช้เพื่อรอหลายๆ INPUT พร้อม

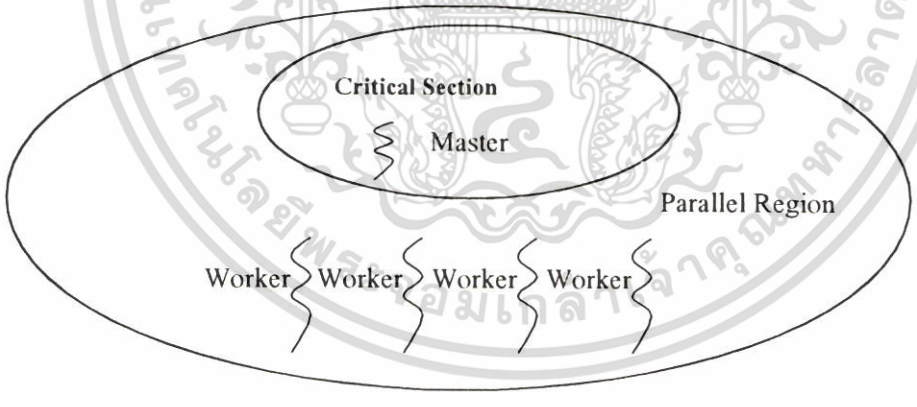
ฟังก์ชั่นนี้จะส่งหมายเลขของ File Descriptor เก็บไว้ในตัวแปรอาเรย์พอยน์เตอร์ fds แล้ว (ถูกแบ่งและลบออกโดย libpvm) ซึ่งอย่างน้อยต้องมี fds[0] ของ PVMD โดยจำนวนซ็อกเก็ตจะ

มากหรือน้อยขึ้นอยู่กับเส้นทางที่ถูกสร้างเพื่อใช้ในการติดต่อกับงานอื่นๆบนระบบ ซึ่งฟังก์ชั่นนี้จะใช้ควบคู่กับ pvm_nrecv หรือ pvm_probe ในการรอเมสเส็จบนระบบโดยไม่ทำการ blocking แต่อย่างไก็ตามฟังก์ชั่นนี้ยังมีข้อจำกัดคือจะใช้งานได้เฉพาะบนระบบปฏิบัติการ UNIX หรือคล้ายๆกันเท่านั้น

ฟังก์ชั่นขั้นสูงที่กล่าวมาทั้งหมดเป็นการเพิ่มขีดความสามารถการทำงานบนระบบ PVM ให้เหมาะสมกับการประมวลผลคำนวณแบบกระจาย (Distributed Computing) กับหลายๆแอ็พพิเคชั่นระบบขั้นสูงที่ซับซ้อนมากๆเช่น Parallel IO, Distributed Database System ซึ่งต้องจัดการ Synchronization และ Locking ทรัพยากรบนระบบ รวมถึง สนับสนุน Transaction Processing บนระบบ, และ การจัดการสืบค้นข่าวสารแบบกระจาย (Distributed Information Retrieval) เป็นต้น

2.4 ทฤษฎีอัลกอลิทึมการจัดงานพลวัต (Dynamic Scheduling Algorithms)

Tang และ Yew [2] นำเสนอแนวคิดการออกแบบคอมไพล์เลอร์สำหรับจัดงานที่ลูปโปรแกรม (แบบ DOALL) ซึ่งเป็นส่วนที่ทำให้เกิดภาระงานหนักที่สุดในโปรแกรมเรียงลำดับบนเครื่อง SMP (Symmetric Multi- Processor) โดยแบ่งการทำงานออกเป็น 2 เทรดหลักๆคือ Master และ Workers เทรด Master เทรดมีหน้าที่ที่หลักในการควบคุมการเข้าใช้ทรัพยากรระบบที่เก็บใน Critical Section และ Workers เทรดมีหน้าที่ในการคำนวณหาผลลัพธ์ ดังแสดงรูปที่ 2.11



รูปที่ 2.11 การทำงานแบบ Master/Worker ภายใน Parallel Region บนเครื่อง SMP

โดยมีจำนวนของ Worker เทรดเท่ากับจำนวนโพรเซสเซอร์ที่ใช้งานบนระบบ Master เทรดจะควบคุมและจัดแบ่งสรรทรัพยากรต่างๆให้กับแต่ละ Worker เทรดใช้ประมวลผล แต่อย่างไรก็ตามปัญหาภาระไม่สมดุลยบนเครื่อง SMP เกิดขึ้นเนื่องจากแต่ละ Worker เทรดไม่สามารถเริ่มทำได้อย่างพร้อมกัน ทั้งนี้เนื่องจากภาระงานหนักของเครื่องในช่วงเวลาขณะนั้น ทำให้แต่ละ Worker เทรดไม่สามารถเข้าใช้งานโพรเซสเซอร์บนระบบได้พร้อมเพียงกัน ดังนั้น Master เทรดจะต้องเสียเวลารอให้ Worker เทรดตัวสุดท้ายได้ทำงานก่อนจึงจะเสร็จสิ้นการทำงานได้

การจัดงานพลวัตคือแนวคิดของการแบ่งภาระงานให้มีหลากหลายขนาดในช่วงรันเพื่อลดภาระไม่สมดุลยบนระบบ โดยมีอัลกอริทึมที่ใช้ในการกำหนดขนาดของภาระงานในแต่ละครั้งที่เกิดการจัดงาน/กระจายงานบนระบบ ซึ่งแต่ละอัลกอริทึมจะมีข้อดีและข้อเสียแตกต่างกันทั้งนี้ขึ้นอยู่กับแนวทางการแก้ปัญหาแต่ละแบบ ซึ่งสามารถแยกได้ 5 อัลกอริทึมหลักๆดังนี้

2.4.1 Self Scheduling (SS)

จะแบ่งงานให้เป็นหน่วยย่อยที่สุด (Atomic) และแต่ละหน่วยย่อยของงานจะถูกส่งให้แก่โพรเซส หรือ เทรด ที่พร้อมทำงานในขณะนั้น ดังนั้นเมื่ออัลกอริทึมนี้ทำการจัดแบ่งปัญหาที่มีขนาด N (ขนาดของลูป N รอบ) จะเกิดการกระจายงานบนระบบถึง N ครั้ง แต่อย่างไรก็ตามอัลกอริทึมนี้จะมีค่าสูญเสียเนื่องจากการจัดงาน/การกระจายงานบนระบบมากกว่าอัลกอริทึมอื่นๆ แม้นว่าจะดีในเรื่องของการทำภาระงานสมดุลยระบบ (แต่ละโพรเซสเซอร์ทำงานเสร็จได้พร้อมเพียงกัน)

2.4.2 Chunk Scheduling (CS)

เนื่องจากอัลกอริทึมการจัดแบ่งงานที่ 2.4.1 จะมีค่าสูญเสียในการจัดงานสูงมากดังนั้นอัลกอริทึมนี้จะรวมหน่วยย่อยของงานให้ใหญ่ขึ้นเป็นชุดของปัญหา/ชุดของภาระงาน (chunk) ที่มีขนาดเท่าๆกัน ซึ่งสมการที่ง่ายสุดในการแบ่งขนาดภาระงานคือ

$$C = Ceil(N/P)$$

N – คือขนาดภาระงาน หรือ ขนาดลูปโปรแกรมเรียงลำดับ

P – คือจำนวนโพรเซสเซอร์ที่ใช้งานบนระบบ

อัลกอริทึมนี้จะเกิดการจัดงาน/กระจายงานบนระบบสุด (เท่ากับจำนวน P ครั้ง) แต่อัลกอริทึมนี้จะคล้ายกับการจัดงานแบบสถิตย์ ดังนั้นอัลกอริทึมนี้เสี่ยงต่อการเกิดภาระไม่สมดุลยบนระบบได้สูงเช่นกัน

2.4.3 Guide Self Scheduling (GSS)

แนวคิดของอัลกอริทึมนี้โดย [3] คือเหลืองานสำหรับการทำภาระงานสมดุลยระบบโดยให้ขนาดภาระงานมากสุดคือ N/P ในตอนเริ่มต้นของการจัดงานและจะลดลงขนาดภาระงานที่เหลือในแต่ละครั้งที่เกิดการจัดงาน/กระจายงานบนระบบโดยอัตราส่วน 1/P ตามสมการดังนี้

$$C_i = Ceil(R_i/P)$$

R_i – คือขนาดภาระงานที่เหลือสำหรับการจัดงานรอบที่ i

จากสมการนี้แสดงให้เห็นว่าเริ่มต้นของจัดงานบนระบบ (i=1) ค่า R_1 จะมีเท่ากับ N โดยแต่ละครั้งที่เกิดการจัดงานบนระบบขนาดภาระงานจะลดลง ตัวอย่างเช่น ถ้า N=20 บนเครื่องที่มี 4 โพรซสเซอร์ การจัดงานแต่ละครั้งที่เกิดขึ้นแสดงได้ดังนี้

ตารางที่ 2.1 แสดงการจัดงานบนเครื่อง 4 โพรเซสเซอร์ด้วย GSS

การจัดงานครั้งที่	1	2	3	4	5	6	7	8	9
ขนาดภาระงาน	5	4	3	2	2	1	1	1	1

จากตารางที่ 2.1 แสดงให้เห็นว่าจะเกิดการจัดงาน/กระจายงานบนระบบถึง 9 ครั้งเพื่อให้เกิดการทำภาระงานสมดุลยระบบ

2.4.4 Factor Scheduling (FS)

แนวคิดของอัลกอริทึมนี้เหลือภาระงานครึ่งหลังสำหรับการทำภาระงานสมดุลยระบบ โดยการเปลี่ยนขนาดภาระงานแต่ละครั้งจะทำการแบ่งงานแบบเท่าๆกันเก็บไว้ในระบบแบ็ชที่มีขนาดเท่ากับจำนวนโพรเซสเซอร์บนระบบ ทั้งนี้เพื่อลดจำนวนครั้งในการจัดงาน/กระจายงานบนระบบ

เนื่องจากขนาดภาระงานตอนเริ่มต้นจัดงานคือ N/2*P ซึ่งถ้าเกิดการเปลี่ยนภาระงานแต่ละครั้งที่เกิดการจัดงาน/กระจายงานบนระบบจะทำให้เกิดการจัดงานบนระบบที่มากกว่า GSS เกือบ 2 เท่า

$$C = Ceil(R_i / (2*P))$$

ตารางที่ 2.2 แสดงการจัดงานบนเครื่อง 4 โพรเซสเซอร์ด้วย FS

การจัดงานครั้งที่ i	1	2	3	4	5	6	7	8	9	10	11	12
ขนาดภาระงาน	3	3	3	3	1	1	1	1	1	1	1	1

จากตารางที่ 2.2 แสดงให้เห็นว่าการจัดงานด้วยอัลกอริทึมนี้จะเกิดการจัดงาน/กระจายงานบนระบบมากกว่า GSS คือ 12 ครั้ง ขนาดภาระงานที่มากสุดของอัลกอริทึมนี้คือ N/2*P และจะมีการกำหนดขนาดภาระงานใหม่เมื่อการจัดงาน/กระจายงานบนระบบ (i) ครบจำนวนโพรเซสเซอร์หรือ แบ็ช (Batch) ของระบบว่าง

2.4.5 Trapezoid Self Scheduling (TSS)

แนวคิดการแบ่งภาระงานของอัลกอริทึมนี้จะคล้ายกับ 2.4.3 แต่การจัดงาน/กระจายงานบนระบบในแต่ละครั้งจะลดลงด้วยค่าคงที่ D ตามที่แสดงในสมการนี้

$$n = Ceil (2*N/(F+L))$$

$$F = Ceil (N/2*P)$$

$$D = (F-L)/(n-1)$$

$$C_i = F - (i-1)*D$$

โดย L ถูกกำหนดให้เป็นค่าคงที่ ส่วนใหญ่จะกำหนดไว้เท่ากับ 1 โดยจะเห็นได้ว่าอัลกอริทึมนี้มีการคำนวณหาจำนวนครั้งของการจัดงาน/กระจายงานบนระบบด้วยค่า n และขนาดภาระงานตอนเริ่มต้นจัดงาน (F) จะถูกกำหนดไว้เท่ากับ N/2*P แล้ว ดังนั้นจึงสามารถกำหนดค่าคงที่ D เพื่อใช้กำหนดขนาดภาระงาน ตัวอย่างเช่นถ้า N =20 บนเครื่อง 4 โพรเซสเซอร์ การจัดงานของอัลกอริทึมนี้สามารถแสดงได้ดังนี้

ตารางที่ 2.3 แสดงการจัดงาน N=20 บนเครื่อง 4 โพรเซสเซอร์ด้วย TSS

การจัดงานครั้งที่ i	1	2	3	4	5	6	7	8	9	10
ขนาดภาระงาน	3	3	3	2	2	2	2	1	1	1

จากตารางที่ 2.3 ขนาดภาระงานที่มากสุดตอนเริ่มต้นของการจัดงานคือ N/2*P และแต่ละครั้งภาระงานจะลดลงด้วยอัตราส่วน 2 / 9 (จากตัวอย่างนี้ N=20) ซึ่งสมการกำหนดค่าภาระงานนี้จะทำการปัดเศษขึ้นเมื่อเศษส่วนมากกว่า 0.5

2.5 การประเมินประสิทธิภาพ (Performance Evaluation)

2.5.1 ตัววัด (Metric)

-เวลารันโปรแกรม (Runtime) คือการวัดเวลาในช่วงโปรแกรมประมวลผลบนเครื่องนั้นๆ ได้แก่

Tseq คือเวลาที่ใช้ในการประมวลผลบนเครื่องเดียว

Tpar คือเวลามี่ใช้ในการประมวลผลบนหลายๆเครื่อง

-ความเร็วที่เพิ่มขึ้น (Speed Up) คืออัตราส่วนระหว่างเวลาที่ประมวลผลบนเครื่องเดียวกับเวลาที่ประมวลผลด้วยหลายเครื่อง

$$S=Tseq/Tpar$$

-ประสิทธิภาพ (Efficiency) เป็นการวัดเพื่อบ่งบอกถึงความคุ้มค่าในการใช้ทรัพยากรระบบ

$$E = S/P$$

ซึ่งในทางอุดมคติควรจะได้ S = P, E=1 เมื่อ P คือจำนวนโพรเซสเซอร์ที่ใช้งานบนระบบ

2.5.2 ข้อจำกัดของระบบคอมพิวเตอร์คู่ขนาน (Limited Parallel Computer)

- สำหรับเครื่องคอมพิวเตอร์คู่ขนานจะมีข้อจำกัดอยู่ที่ระบบเครือข่ายที่ใช้งาน
- T_{com} เวลาที่สูญเสียจากการติดต่อสื่อสารบนเครือข่าย

- โดยเครือข่ายมีแบนด์วิธ (B) มีหน่วยเป็น บิต/sec ดังนั้นจะได้

$$T_{com} = 1/B$$

2.5.3 ขนาดงาน (Granurality)

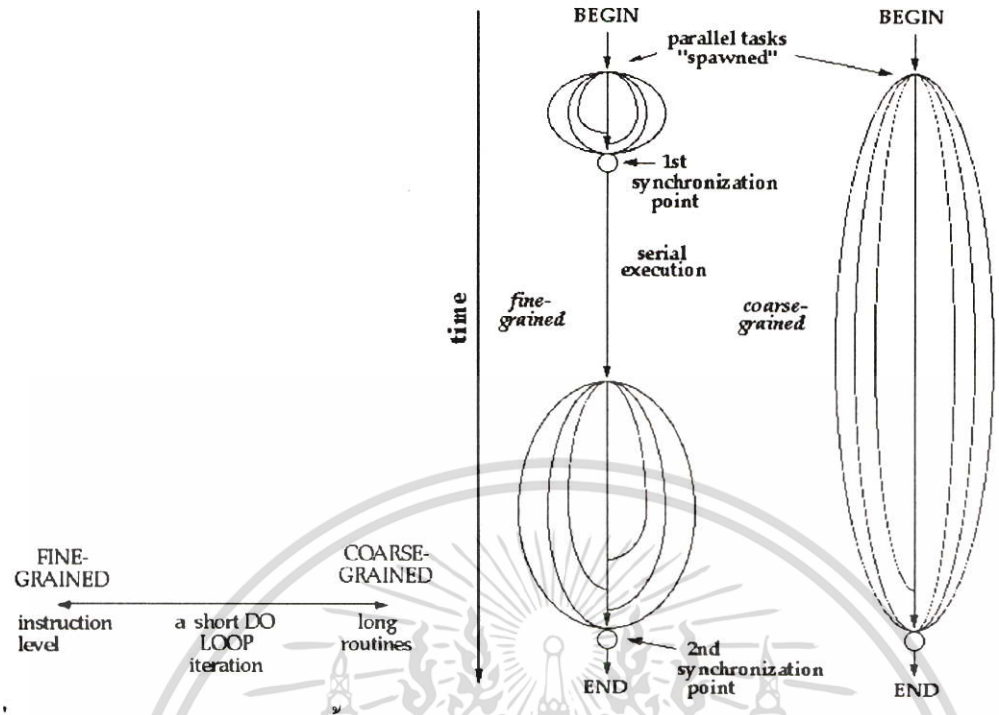
คืออัตราส่วนระหว่างเวลาที่ใช้ในการคำนวณบนเครื่องนั้นๆกับเวลาที่สูญเสียไปเนื่องจากการติดต่อสื่อสารบนเครือข่าย

$$G = T_{exe}/T_{com}$$

ค่านี้มีความสำคัญมากในการแบ่งงานเพื่อให้เกิดการความเหมาะสมในการทำงานแบบคู่ขนาน (parallelism) ตามสถาปัตยกรรมที่ใช้ ซึ่งแยกได้เป็น 2 ระดับคือ

1. **Fine-gain parallelism** ในแต่ละรอบที่มีติดต่อสื่อสารจะมีระยะเวลาในการประมวลผลสั้น ซึ่งหน่วยของงานในระดับนี้จะดีมากในแง่การปรับภาระงานสมดุลยระบบ แต่ก็มีค่าสูญเสียในการกระจายงานบนระบบมากเช่นกัน ดังนั้นการกำหนดหน่วยของงานในระดับนี้จะเป็นจะต้องทำงานบน Hardware ที่สนับสนุนเป็นอย่างดี โดยหน่วยของงานในระดับนี้มักจะทำบนสถาปัตยกรรม Share Memory เช่น บนเครื่อง SMP (Symmetric Multi- Processor) เป็นต้น
2. **Coarse-gain parallelism** ในแต่ละรอบที่มีการติดต่อสื่อสารบนระบบจะมีระยะเวลาในการประมวลผลที่นานกว่า ทั้งนี้เพื่อลดค่าสูญเสียที่เกิดขึ้น แต่อย่างไรก็ตามหน่วยของงานในระดับนี้ทำให้ลดประสิทธิภาพของทำงานแบบคู่ขนาน และ การทำภาระงานสมดุลยบนระบบ ดังนั้นหน่วยของงานในระดับนี้เหมาะสำหรับทำงานบนสถาปัตยกรรมที่ใช้ระบบเครือข่ายความเร็วต่ำ

บางครั้งการแยกระดับขนาดของงานอาจใช้การนับชุดคำสั่งแทน โดยถ้าทำงานเพียงไม่กี่ชุดคำสั่งแล้วจำเป็นต้องส่งเมสเส็จออกในแต่ละครั้งระหว่างที่ประมวลผล เช่นการทำ Synchronize เป็นต้น ก็จะถูกจัดให้เป็น FINE-GRAINED แต่ถ้าต้องประมวลผลกับชุดคำสั่งเป็นล้านๆก่อนจึงจะเกิดการส่งเมสเส็จในแต่ละครั้ง ก็ถือได้ว่าเป็น COARSE-GRAINED



รูปที่ 2.12 แสดงขนาดของงานทั้ง FINE GRAINED และ COARSE GRAINED

อย่างไรก็ตามนักวิจัยบางกลุ่มและตำราบางเล่มอาจจะแยกได้เป็น 3 ระดับ โดยเพิ่มระดับกลางคือ Medium-gain parallelism ขึ้น ทั้งนี้เนื่องจากเทคโนโลยีในปัจจุบันฮาร์ดแวร์มีขีดความสามารถที่สูงรวมถึงระบบเครือข่ายที่มีความเร็วสูงแต่ต้นทุนต่ำ ซึ่งช่วยลดค่าสูญเสียลงได้มาก

และค่านี้ก็มีความสำคัญอย่างมากในการออกแบบแอ็พพิเคชั่นโปรแกรมคู่ขนาน รวมถึงการออกแบบฮาร์ดแวร์ระบบ เพื่อกำหนดที่ตั้งข้อมูล (data locality) ตามโพรเซสเซอร์ต่างๆที่ใช้งานบนระบบ ทั้งนี้เพื่อลดการขนส่งข้อมูลระหว่างโพรเซสเซอร์ เช่นการกำหนดขนาดหน่วยความจำความเร็วสูง (cache) ตามโพรเซสเซอร์ที่ใช้งานบนระบบเป็นต้น

2.6 ทฤษฎีการจัดงานระบบเบื้องต้น (Basic Scheduling Theory)

การจัดงานคือมอบหมายงาน/กระจายงานให้แก่โพรเซสเซอร์บนระบบทำงานแบบคู่ขนาน โดยยังคงให้ผลลัพธ์ที่ถูกต้องเหมือนกับการประมวลผลแบบเรียงลำดับทุกประการ ซึ่งมีเกณฑ์ต่างๆที่สามารถนำมาใช้พิจารณาเพื่อจำแนกประเภทของการจัดงานบนระบบ แต่ส่วนใหญ่มักจะใช้ ช่วงเวลาของการทราบรายละเอียดงานและระบบ (Task & System Information) เป็นเกณฑ์หลักดังนั้นทำให้การจัดงานสามารถจำแนกได้เป็น 2 ลักษณะใหญ่ๆดังนี้

1. การจัดงานแบบสแตติก (Static Scheduling)

จะทราบรายละเอียดของงานและระบบก่อนเริ่มประมวลผล (รู้ในช่วงคอมไพล์โปรแกรม) ซึ่งบางครั้งจะเรียกอัลกอริทึมที่ใช้จัดงานแบบนี้ว่า offline algorithm โดยโปรแกรมแบบคู่ขนานจะ

ถูกประยุกต์ในรูปแบบ DAG (Directed Acyclic Graph) ที่กำหนดให้แต่ละโหนดคืองานหนึ่งๆบนระบบ และ โหนดที่สัมพันธ์กันจะถูกเชื่อมต่อด้วยลิงก์ (Link) โดยโหนดแรกจะต้องถูกจัดให้ทำงานเสร็จเรียบร้อยก่อนโหนดสุดท้ายจะเริ่มงาน และแต่ละโหนดสามารถจะถูกกำหนดลำดับความเร่งด่วนได้ แต่โหนดที่อยู่ก่อนหน้าโหนดเร่งด่วนจะถูกจัดงานก่อนดังนั้นการจัดงานแบบนี้จะไม่มีความอ่อนตัว เนื่องจากหากมีการสื่อสารหนาแน่นบนเครือข่ายจะทำให้การจัดลำดับไม่เป็นไปตามความเป็นจริงและทำให้เสียเวลามากขึ้น ดังนั้นจึงจำเป็นต้องใช้งานร่วมกับ clustering algorithms เพื่อรวมบางโหนดที่จะทำให้เกิดค่าสูญเสียในการติดต่อสื่อสารสูงๆเข้าด้วยกัน ในการลดค่าสูญเสียบนเครือข่ายให้น้อยสุดและให้เกิดการใช้ cpu idle time ของแต่ละโพรเซสเซอร์ได้มากสุด

2. การจัดงานแบบไดนามิก (Dynamic Scheduling)

แม้นว่ามีงานวิจัยที่เกี่ยวข้องกับการจัดงาน DAG มากมาย แต่ในความเป็นจริงส่วนใหญ่การพัฒนาแอ็พพิเคชั่นโปรแกรมให้เหมาะสมกับสภาพแวดล้อมระบบนั้นเป็นเรื่องยากมากๆ ดังนั้นผู้พัฒนาแอ็พพิเคชั่นโปรแกรมส่วนใหญ่จึงเกิดแนวคิดการพัฒนาโปรแกรมเพื่อการจัดการกับงานและทรัพยากรระบบ (การเพิ่ม/ลดจำนวนโพรเซสเซอร์หรือจำนวนเครื่องที่ใช้งานระบบ) ในช่วงที่โปรแกรมรันแทน (Runtime) ตัวอย่างของการจัดงานแบบนี้เช่น การแตกเป็นงานย่อยรวมถึงการสั่งหยุดทำงานบนระบบซึ่งจะมีการตอบสนองได้ดีกว่า รวมไปถึงการใช้ตัวแปรภาษาคู่ขนานบางตัว (parallel compiler) สำหรับการจัดแบ่งงานของลูปโปรแกรมเพื่อลดจำนวนรอบของการทำงานแบบวนซ้ำกัน และ ทำให้เกิดภาระงานสมดุลยระบบอีกด้วย โดยที่ผู้พัฒนาแอ็พพิเคชั่นโปรแกรมไม่ต้องวิตกเรื่องความไม่สมดุลยระบบที่เกิดขึ้นอีกต่อไป ซึ่งจะเพิ่มความยืดหยุ่นในการจัดงานระบบอย่างมากสำหรับบาง DAG ที่ไม่สามารถแสดงหลายละเอียดบางโหนด หรือ ลิงค์ได้อย่างครบถ้วน (เนื่องจากบางโหนดที่เกิดขึ้นช่วงรันโปรแกรมนั้นจะขึ้นอยู่กับเงื่อนไขบางอย่างของข้อมูล เช่นชุดคำสั่ง if...else, jump และ goto เป็นต้น รวมถึงขนาดข้อมูลที่มีความแปรผันได้ในบางลิงค์) โดยจะเรียกอัลกอริทึมที่ใช้จัดงานเหล่านี้ว่า online algorithm

อย่างไรก็ตามมีนักวิจัยบางกลุ่มได้เพิ่มประเภทของการจัดงานใหม่ขึ้นมาเรียกว่า การจัดงานแบบกึ่งสแตติก (Semi-Static Scheduling) โดยจะทราบรายละเอียดในช่วงก่อนโปรแกรมเริ่มทำงาน (Program startup) ซึ่งต้องอาศัยหลักการทางสถิติเพื่อเก็บข้อมูลของงานที่รันและทำนายโครงสร้าง DAG กราฟที่มีความแน่นอนก่อนรันโปรแกรม

2.6.1 เทคนิคการแบ่งงาน (Partition Technique)

ในการแบ่งปัญหาหรือภาระงานระบบสามารถจำแนกได้ 4 วิธีดังนี้

1. **Fixed Partition** คือการจัดแบ่งงานแบบตายตัวโดยขนาดงานไม่สามารถเปลี่ยนแปลงได้
2. **Variable Partition** คือการแตกงานที่มีขนาดใหญ่ให้เป็นงานย่อยๆหลายงานประมวลผลอย่างพร้อมเพียงกันบนทรัพยากรระบบตามที่ผู้ใช้ทำการร้องขอ
3. **Adaptive Partition** คือการจัดแบ่งงานโดยอัตโนมัติตามความสัมพันธ์กับปริมาณงานที่มีอยู่แล้วบนระบบ
4. **Dynamic Partition** คือ ขนาดที่จัดแบ่งสามารถเปลี่ยนแปลงได้ในช่วง Runtime ขึ้นอยู่กับความเปลี่ยนแปลงความต้องการ และ ปริมาณงานที่อยู่บนระบบ

โดยสามารถสรุปประเภทของการแบ่งงานตามพารามิเตอร์ต่างๆที่เกี่ยวข้องได้ดังตารางนี้

ตารางที่ 2.4 แสดงการแบ่งประเภทของการทำ Partition โดยพิจารณาจากพารามิเตอร์เหล่านี้

	Parameters taken into account		
Schema	User request	System load	Changes
Fixed	No	No	No
Variable	Yes	No	No
Adaptive	Yes	Yes	No
Dynamic	Yes	Yes	Yes

Variable partition จะนิยมมากบนเครื่อง Distributed Memory เช่น เครื่อง Ncube Hypercubes, IBM SP2, Intel Paragon, Meiko CS-2 และ Cray T3D โดยไม่มีข้อจำกัดในเรื่องการจัดการหน่วยความจำระบบ และ แก้ปัญหาการ Share Address Space ของเครื่อง ข้อดีของการแบ่งงานแบบนี้คือจะจัดทรัพยากรระบบให้ทำงานได้อย่างเหมาะสมกับงานที่ถูกแบ่ง และ โปรแกรมสามารถจัดการควบคุมข้อมูลในหน่วยความจำของเครื่องที่รันอยู่ได้ โดยไม่จำเป็นต้องมีหน่วยเก็บข้อมูลสำรอง แต่ข้อจำกัดของการแบ่งงานแบบนี้ก็คือ ความไม่ลงตัวในการแตกงาน ทั้งนี้เนื่องจากบางกรณีที่ผู้ใช้ร้องไปนั้นอาจจะมีบางเครื่องที่ไม่สามารถจะตอบสนองได้ทันที ดังนั้น งาน หรือ Job ดังกล่าวอาจจะเข้าไปอยู่ในระบบคิวบนเครื่องนั้นๆเป็นเวลานาน ซึ่งการแบ่งงานแบบ Variable partition นี้จึงเหมาะกับระบบ Batch Processing มากกว่างานในลักษณะ Interactive Work หรือ Realtime

Adaptive partition แก้ปัญหาของการแบ่งงานแบบตายตัว (fixed partition) และ Variable partition โดยจะพิจารณาภาระระบบปัจจุบัน (current system load) ก่อนทำการแบ่งกลุ่มงานทั้ง

หมด N ให้เหลือ K กลุ่ม ซึ่งแต่ละกลุ่มจะมีขนาดเท่าๆกัน S (S=N/K) โดยจะใช้วิธีการ FCFS (First Come First Serve) ในการจัดงานเข้ากลุ่ม ซึ่งค่า K ควรอยู่ระหว่าง P < K<N เมื่อ P คือ จำนวนโพรเซสเซอร์หรือจำนวนหน่วยประมวลผลบนระบบ แต่อย่างไรก็ตามการแบ่งแบบนี้ยังถือว่าคล้ายกับการแบ่งแบบ Variable partition อยู่ดี เพราะจะต้องจัดแบ่งให้เรียบร้อยก่อนจะสั่งรันได้

Dynamic partition จะทำการแบ่งงานในช่วงรันโดยที่ขนาดอาจจะไม่คงที่ทั้งนี้ขึ้นอยู่กับความสามารถของระบบที่จะตอบสนองได้ในขณะนั้น ส่วนงานที่เหลือจะถูกเก็บไว้ใน Tasks Pool ของระบบ ซึ่งจะรอจนกว่า Worker thread หรือ Worker process พร้อมจะตอบสนองอีกครั้ง (ในบางกรณีอาจหมายถึงการที่ระบบสามารถสร้าง Thread หรือ Process ขึ้นใหม่เพื่อรันงานที่เพิ่มขึ้นได้) โดยข้อดีที่สำคัญของการแบ่งงานในลักษณะนี้คือจะมีความยืดหยุ่นมากในการปรับภาระงานสมดุลยระบบและเหมาะสมกับแอ็พพิเคชั่นโปรแกรมประยุกต์ต่างๆ แต่ข้อเสียที่สำคัญคือจะเกิดค่าสูญเสียของการแบ่งงานใหม่อย่างมาก แต่อย่างไรก็ตามถ้างานที่ถูกแบ่งในช่วงที่รันมีความเหมาะสมที่ระบบสามารถตอบสนองได้แล้วก็ถือได้ว่าคุ้มค่ามากกว่าการแบ่งงานทั้ง 3 แบบที่กล่าวมา

บทที่ 3
งานวิจัยต่างๆที่เกี่ยวข้อง

ในช่วงแรกงานวิจัยเกี่ยวกับการจัดงานจะเป็นการจัดงานตามกราฟ DAG แต่ส่วนใหญ่เมื่อนำไปประยุกต์ใช้งานจริงพบว่ายังเป็นเรื่องที่ยุ่งยากอย่างมากในการพัฒนาแอ็พพิเคชั่นโปรแกรมและการทำภาระงานแบบสมดุลยบนระบบ ดังนั้นผู้พัฒนาโปรแกรมส่วนใหญ่จึงมีแนวคิดของการออกแบบระบบจัดงานเพื่อให้สอดคล้องกับการใช้งานได้จริงบนระบบนั้นๆ ทั้งนี้เพื่อเป็นการทำภาระงานสมดุลยบนระบบ โดยมีงานวิจัยต่างๆที่เกี่ยวข้องดังนี้

3.1 A PVM Code Generator for the Fortran Parallel Transformer

Erik H. D'Hollander และ Fubo Zhang แห่งภาควิชาวิศวกรรมไฟฟ้า ณ.มหาวิทยาลัย Ghent ซึ่งทั้งคู่อยู่ในกลุ่มศึกษาวิจัยทางด้าน Parallel Information System ได้นำเสนอเทคนิคของการแปลงโปรแกรมเพื่อใช้งานบนระบบ PVM โดยตัวแปลงโปรแกรมที่พวกเขาพัฒนาขึ้นนี้ใช้งานจริงกับภาษาฟอร์แทน เรียกว่า FPT (Fortran Parallel Transformer) เพื่อทำการค้นหาส่วนของโปรแกรมที่เป็น DOALL และ สร้างโปรแกรมขึ้นมา 2 ส่วนคือ ส่วนที่เป็นมาสเตอร์ และ สลาฟ ซึ่งจะทำการจัดแบ่งส่วนของที่โปรแกรม DOALL โดยออกแบบการกระจายข้อมูล (Data Distribution) รวมไปถึงการจัดแบ่งส่วนของการคำนวณที่สลาฟโปรแกรมประมวลผลพร้อมกัน

3.2 A Load Balanced Scheduler for PVM Jobs

José Nagib Cotrim Árabe และ Virgilio Augusto Fernades Almeida แห่งมหาวิทยาลัย Federal de Minas Gerais ประเทศบราซิล ได้นำเสนอแนวคิดการออกแบบและประยุกต์โปรแกรมจัดงานแบบไดนามิกบน PVM โดยใช้อัลกอริทึม sender-initiated เพื่อทำการแลกเปลี่ยนข้อมูลของแต่ละเครื่อง WORKSTATION บนระบบ ซึ่งบนแต่ละเครื่อง WORKSTATION จะรันโปรแกรม daemon ไว้ 2 ตัว (pvmd และ ตัวแลกเปลี่ยนข้อมูลภาระของแต่ละเครื่อง) โดยโปรแกรม daemon ตัวที่สองจะทำหน้าที่คอยแลกเปลี่ยนข้อมูลภาระงานระหว่างเครื่องตัวเองกับเครื่องอื่นๆบนระบบตามช่วงเวลาที่ถูกตั้งไว้ นอกจากนี้แล้วโปรแกรม daemon ตัวที่สองนี้ยังมีหน้าที่หลักในการตัดสินใจสั่งสร้างงานใหม่โดยให้สิทธิแก่เครื่องที่มีภาระน้อยที่สุดในขณะนั้นก่อน ซึ่งค่าภาระแต่ละเครื่องจะถูกทำเป็นดัชนีไว้ (load index) แล้วจากการเทียบอัตราส่วนของ idletime และ ความเร็วโพรเซสเซอร์ ทั้งนี้เพื่อให้ฟังก์ชันการสร้างโพรเซสระบบได้นำไปประยุกต์ใช้ตามจำนวนงานที่ถูกจัดเก็บอยู่ใน ready queue โดยทั้งคู่ได้ทำการทดสอบ 4 นโยบายหลักสำหรับการจัดงานบนระบบ PVM เพื่อวิเคราะห์เปรียบเทียบประสิทธิภาพของแต่ละนโยบายดังนี้ 1.ใช้นโยบายเดิมที่ PVM

กำหนดไว้ 2. ใช้นโยบายกระจายงานให้แก่เครื่องที่มีค่าดัชนีภาระน้อยที่สุด 3.เป็นนโยบายที่แก้ไขเพิ่มเติมจากนโยบายที่ 2 เพื่อเลี่ยงการเกินภาระ (overloading) ของโพรเซสเซอร์ในกรณีที่งานมาถึงเร็วกว่าระหว่างที่มีการแลกเปลี่ยนข้อมูลภาระงานบนระบบ 4.นโยบายให้ผู้ใช้ระบุสิทธิบางงานเพื่อขอใช้บริการก่อน โดยนำแอ็พพิเคชั่นโปรแกรมคู่ขนาน Harmonics summation, Bessel's equation และ Matrix Multiplication มาเป็นภาระงานสำหรับการทดสอบและผลลัพธ์ที่ได้คือสามารถลดเวลาเฉลี่ยของการประมวลผลได้จาก 20% จนถึง 60 %

3.3 A load Balancing extension for the PVM Software System

CHRISTOPHER WADE HUMPHERS แห่งมหาวิทยาลัย Alabama ได้ประยุกต์การทำภาระสมดุลยพลวัตในระบบ PVM โดยกระจาย Slave โพรเซสตามเครื่องต่างๆ เพื่อเก็บข้อมูล CPU Load ของแต่ละเครื่อง (ด้วยชุดคำสั่ง vmstat บนระบบปฏิบัติการ UNIX) ให้ Master รวบรวมเพื่อเชื่อมต่อ (interface) ในรูปแบบ API (Application Program Interface) กับแอ็พพิเคชั่นโปรแกรมต่างๆได้ ดังนั้นแอ็พพิเคชั่นจะทราบสารสนเทศเหล่านี้ก่อนจะทำการจัดงานบนระบบโดยการสร้างงานให้เครื่องที่มีค่า CPU Load น้อยสุดมากกว่าเครื่องอื่นๆ และ เครื่องที่มีค่า CPU Load สูงๆจะมีการสร้างงานบนเครื่องนั้นน้อยที่สุด ซึ่งเป็นการทำภาระงานสมดุลยบนระบบพร้อมกันด้วย โดยงานวิจัยของเขาสามารถใช้งานได้จริงและมีประสิทธิภาพอย่างมากกับแอ็พพิเคชั่นทดสอบอย่างเช่น vector-matrix multiply , และ แอ็พพิเคชั่นของการบีบอัดรูปภาพ (fractal image compression application) บนเครือข่ายของเครื่อง WORKSTATION ของ SUN และ IBM

3.4 Dynamic Load Balancing under PVM

L. Hluch y และ J. Astalos แห่ง Institute of Computer Systems, Slovak Academy นำเสนอเทคนิคของการจัดแบ่งงานแบบพลวัตเพื่อทำภาระงานสมดุลยแบบพลวัตบนระบบ PVM โดยใช้วิธีกึ่งการกระจาย (Semi-Distrubuted Approach) ที่พวกเขาเรียกว่า Sphere แทนการจัดงานแบบ Global organization (หรือก็คือ Distributed Scheduler) โดยออกแบบให้มีตัวจัดงาน (S) บนระบบอยู่ตัวเดียว และตัวจัดสรรทรัพยากระบบ (RM) กับ ตัวแทน (Agent) กระจายอยู่ตามเครื่องต่างๆ โดยตัวแทนจะทำหน้าที่ในการวัดความเร็วของแต่ละเครื่อง (Speed Testing) และ ภาระงานที่เกิดขึ้นของแต่ละเครื่องเพื่อส่งให้ตัวจัดงานเพื่อใช้เป็นสารสนเทศในการกำหนดนโยบายการจัดงาน (Scheduling policy) โดยตัวจัดสรรทรัพยากรทำการสร้างงานตามนโยบายที่ได้รับนั้นคือถ้าแอ็พพิเคชั่นใดๆมีการเรียกใช้ฟังก์ชั่น pvm_spawn() ตัวจัดสรรทรัพยากรระบบจะรับหน้าที่ในการสั่งรันงานบนเครื่องต่างๆในระบบแทน

3.5 Dynamic Load Balancing in Heterogeneous Environment under PVM

Markus Fischer แห่งมหาวิทยาลัย Paderborn ทำการศึกษาค้นคว้าเรื่องของการทำภาระงานแบบสมดุลยภายใต้สภาพแวดล้อมระบบแตกต่างบนระบบ PVM โดยสภาพแวดล้อมที่แตกต่างนอกจากเครื่องคอมพิวเตอร์แล้วยังรวมไปถึงระบบเครือข่ายที่ใช้งานด้วย ดังนั้นจะนำค่าสูญเสียที่เกิดขึ้นบนเครือข่ายมาเกี่ยวข้อง โดยเขาได้ทำการทดสอบกับแต่ละอัลกอริทึมที่ใช้ในการทำภาระงานแบบสมดุลยบนระบบทั้งนี้เพื่อดูความเหมาะสมในการนำมาประยุกต์ใช้งานจริง

3.6 Dynamic Scheduling on Distributed-Memory Multiprocessor และ Combining Static and Dynamic Scheduling on Distributed-Memory Multiprocessor

Oscar Plata และ Francisco F. Rivera แห่งมหาวิทยาลัย Santiago ประเทศสเปน ได้ทำการศึกษาค้นคว้าถึงความเหมาะสมในการออกแบบอัลกอริทึมการจัดงานแบบพลวัตที่เหมาะสมกับเครื่องหลายโพรเซสเซอร์ที่ใช้หน่วยความจำแบบกระจาย โดยนำแนวคิดการจัดงานแบบ Heuristic มาใช้ร่วมกับอัลกอริธึมการจัดงานแบบพลวัตเดิม รวมถึงในงานวิจัยถัดมาได้ทำการประยุกต์ระบบจัดงานแบบกระจายโดยใช้อัลกอริทึมการจัดงานแบบสแตกติกในช่วงแรกก่อน จากนั้นทำการดัดแปลงโดยใช้อัลกอริทึมการจัดงานแบบพลวัตเดิมเพื่อปรับภาระสมดุลยบนระบบ

3.7 Design Multiprocessor Scheduling Algorithm Using a Distributed Genetic Algorithm System

Kelvin K. Yue และ David J. Lilja แห่งมหาวิทยาลัย Minnessota ได้ร่วมกันศึกษาและออกแบบอัลกอริทึมการจัดงานพลวัตบนระบบหลายโพรเซสเซอร์ด้วยเจเนติกอัลกอริทึม สำหรับการกำหนดขนาดภาระงานให้แต่ละ Worker เทรดที่จะรอรันอยู่ตามโพรเซสเซอร์ต่างๆ และทันทีที่พร้อมประมวลผล (Worker เทรดจับจองโพรเซสเซอร์ได้) แต่ละ Worker เทรดจะมีการประเมินขนาดภาระงานที่จะร้องขอให้ Master เทรดจัดสรรให้ด้วยเจเนติกอัลกอริทึมแทน โดยทั้งคู่ทำการวิเคราะห์เปรียบเทียบกับอัลกอริทึมการจัดงานพลวัตแบบเดิมที่ใช้กันอยู่ พบว่าการจัดงานด้วยวิธีนี้มีประสิทธิภาพมากกว่าอัลกอริทึมการจัดงานพลวัตแบบเดิม

บทที่ 4
การประยุกต์ระบบจัดงานพลวัตบน PVM

4.1 ปัญหาการเกิดภาระงานไม่สมดุลยบนระบบ

การจัดงานแบบพลวัตคือแนวคิดของการแบ่งภาระงานให้มีหลากหลายขนาดในช่วงรันโปรแกรม ทั้งนี้เพื่อแก้ปัญหาภาระงานไม่สมดุลยบนระบบ SMP ที่เกิดขึ้นเนื่องจากแต่ละ Worker เทรดไม่สามารถทำได้อย่างพร้อมเพียงกัน เป็นเหตุให้ Master เทรดต้องเสียเวลารอให้ Worker เทรดตัวสุดท้ายได้ทำงานก่อนจะสิ้นสุดการทำงาน ทั้งนี้ขึ้นอยู่กับภาระของเครื่อง SMP ในขณะนั้นๆ ซึ่งเป็นสาเหตุหลักที่ทำให้แต่ละ Worker เทรดไม่สามารถเริ่มต้นทำงานได้อย่างพร้อมเพียงกัน โดยเฉพาะอย่างยิ่งในช่วงที่ภาระหนักๆ (Heavy load) แต่ละ Worker เทรดจะแย่งกันจับจองโพรเซสเซอร์ระบบที่สามารถใช้ได้เพียงตัวเดียว ณ เวลานั้น

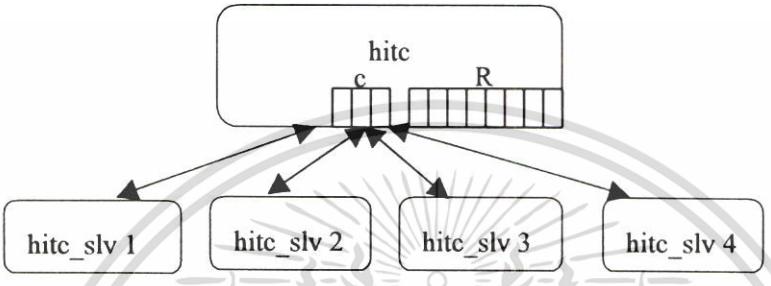
ในปัจจุบันมีเครื่องมือพัฒนาแอ็พพิเคชั่นโปรแกรมระบบเชิงพาณิชย์บางตัวเช่น OpenMP [5] มีความสามารถในการจัดงานแบบพลวัตบนระบบเป็นต้น แต่การจัดงานบนระบบแตกต่างของ PVM ปัญหาภาระงานไม่สมดุลยระบบมักเกิดขึ้นเสมอแม้นว่าแต่ละ Worker โพรเซสสามารถเริ่มทำงานได้อย่างพร้อมเพียงกัน แต่ไม่สามารถทำงานเสร็จได้อย่างพร้อมเพียงกันเนื่องจากสเป็กที่แตกต่างกัน โดยอัตโนมัติของการจัดงานบนระบบ PVM นั้นจะใช้วิธีการสั่งรันโพรเซสแบบวนครบรอบ (Round Robin) ตามตารางเครื่อง (Host Table) บนระบบ PVM แต่เนื่องจากในทางลอจิคอลได้กำหนดให้แต่ละเครื่องมีความเร็วเท่ากันหมด (SP=1000) ทั้งที่จริงแล้วในทางกายภาพแต่ละเครื่องจะมีความเร็วซีพียู และ ขนาดหน่วยความจำที่แตกต่างกันอย่างสิ้นเชิง ดังนั้นจึงจำเป็นต้องมีการแก้ไขค่าพารามิเตอร์เหล่านี้ให้ถูกต้องเพื่อให้ตัวจัดงานระบบ (Scheduler) ได้รายละเอียดของทรัพยากรระบบที่ถูกต้องในการจัดงานบนระบบ

4.2 การออกแบบและประยุกต์มิดเด็ลแวร์ระบบ

จากปัญหาเรื่องการได้รับรายละเอียดของทรัพยากรระบบไม่ถูกต้องทั้งที่มีความเร็วไม่เท่ากันกับความเป็นจริง เช่น ทุกเครื่องได้ sp=1000 ดังนั้นจึงจำเป็นต้องมีมิดเด็ลแวร์ตัวหนึ่งเพื่อแก้ไขค่าติดตั้งเหล่านี้ให้ถูกต้อง รวมไปถึงสนับสนุนการจัดงานแบบพลวัตบนระบบได้อย่างมีประสิทธิภาพ ในปัจจุบันมีมิดเด็ลแวร์เชิงพาณิชย์หลายตัวที่ทำงานบน PVM อย่างเช่น CONDOR, DQS (Distributed Queue System) และ PRM (The Prospero Resource Manager) เป็นต้น ซึ่งซอฟท์แวร์ระบบเหล่านี้จะมีคุณลักษณะที่แตกต่างกัน

4.2.1 รูปแบบแอ็พพิเคชั่นโปรแกรมที่เหมาะสมกับการจัดงานแบบพลวัต

การจัดงานแบบพลวัตบนระบบ กับการพัฒนาแอ็พพิเคชั่นโปแกรมระบบแบบ Master/Slaves (Mandelblot algorithm) จึงมีความเหมาะสมอย่างมาก โดยให้โปรแกรม Master ทำหน้าที่ในการจัดงานแบบพลวัต ซึ่งมี tasks Pool เป็นที่เก็บงานคล้ายกับส่วนของ Critical Section บนเครื่อง SMP โดย Masterโพรเซสจะจัดส่งงานให้ Slave โพรเซสประมวลผล และ Slave โพรเซสจะส่งผลลัพธ์คืนกลับหลังจากประมวลผลเสร็จสิ้นดังแสดงรูปที่ 4.1

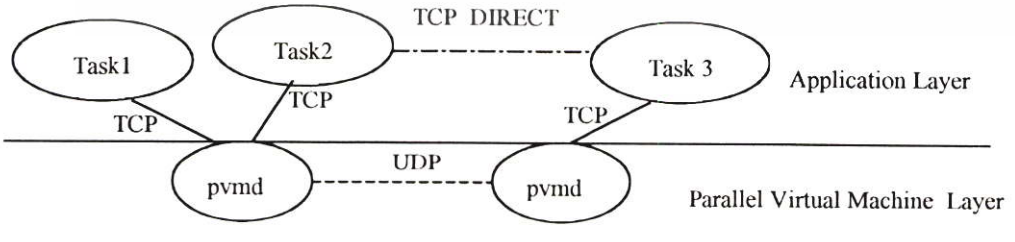


รูปที่ 4.1 ตัวอย่างการรันโปรแกรม hitc/hitc_slv บนระบบ PVM

รูปที่ 4.1 เป็นตัวอย่างของโปรแกรมที่ทำภาระงานสมดุลยพลวัต (Dynamic Load Balancing) บนระบบ PVM โดย hitc ทำหน้าที่เป็น Master ที่คอยจัดแบ่งภาระงานย่อยขนาด c ให้แก่ทุกๆ Slave โพรเซสบนระบบ (โปรแกรม hitc_slv ที่รันตามเครื่องต่างๆบนระบบ) ที่ร้องของาน และจะจัดส่งผลลัพธ์คืนกลับเมื่อประมวลผลเสร็จพร้อมรอรับงานครั้งต่อไป ซึ่งตัวอย่างโปรแกรมนี้ยังคงเหลือภาระงาน R ไว้ใน Task Pool สำหรับการทำภาระงานสมดุลยกับทุกๆ Slave โพรเซสทำงานอย่างพร้อมเพียงกันจนกระทั่งสิ้นสุดการจัดงาน (R=0)

4.2.2 แนวคิดการออกแบบมิดเด็ลแวร์

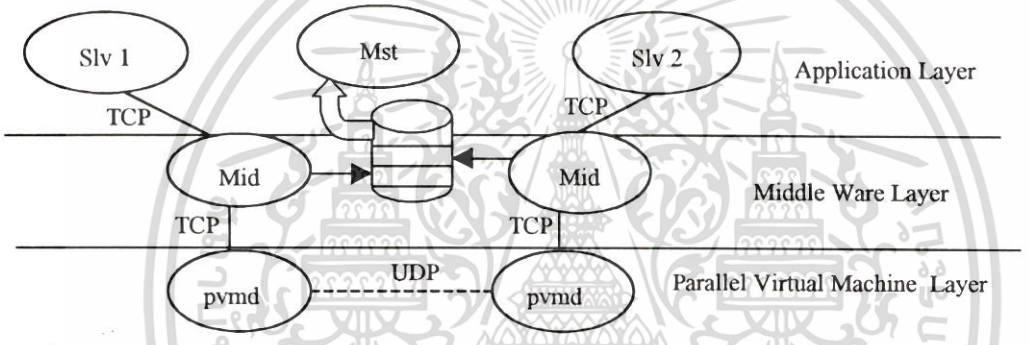
ในการทำงานบนระบบ PVM แยกการทำงานออกเป็น 2 เลเยอร์หลักๆ ดังแสดงรูปที่ 4.2 นี้



รูปที่ 4.2 การทำงานในแต่ละเลเยอร์บนระบบ PVM

ในรูปที่ 4.2 ระดับแอ็พพิเคชั่นเลเยอร์แต่ละงานสามารถติดต่อสื่อสารถึงกันได้ โดยการทำงานร่วมกันของแต่ละ pvmd ในเลเยอร์ล่างสุดที่สร้างความเป็นหนึ่งเดียวกัน (Single image) ในการเข้าใช้งานทรัพยากรระบบ ซึ่งแต่ละ pvmd จะรันตามเครื่องต่างๆและติดต่อสื่อสารกันด้วยโพรโตคอล UDP และซึ่งแต่ละ pvmd จะคอยดูแลงานบนระบบที่อยู่บนเครื่องของตนเอง โดยแต่ละ task จะคอยติดต่อกับ pvmd ของตัวเองด้วย TCP โพรโตรคอล โดยแอ็พพิเคชั่นระบบ (task) นั้นจะมี libpvm3 เป็นตัวเชื่อมตัว (interface) กับตัว pvmd ของระบบในการเชื่อมเลเยอร์ทั้งสอง อย่างไรก็ตามแต่ละ task ในระดับแอ็พพิเคชั่นเลเยอร์สามารถสร้างการเชื่อมต่อถึงกันได้เพื่อช่วยลดภาระการจัดส่งเมสเส็จของ pvmd ได้อีกทางหนึ่ง

และเพื่อสนับสนุนการจัดงานแบบพลวัตบนระบบให้กับแอ็พพิเคชั่นแบบ Master/Slave ดังนั้นมิดเด็ลแวร์ระบบควรจะทำงานในระดับเลเยอร์กลางของทั้งสองตามรูปที่ 4.3



รูปที่ 4.3 สถาปัตยกรรมระบบผ่านมิดเด็ลแวร์

ในรูปที่ 4.3 ที่แอ็พพิเคชั่นเลเยอร์จะมี Masterโพรเซสที่จะทำการจัดงานแบบพลวัตบนเครื่องหลัก (Master Host) ส่วน Slave โพรเซสจะเป็นโพรเซสที่รอทำงานซึ่งจะอยู่ตามเครื่องต่างๆบนระบบเสมือน (Virtual Machine) โดยเลเยอร์กลางที่ถูกพัฒนาขึ้นนี้คือจะเป็นเลเยอร์การทำงานร่วมกันของมิดเด็ลแวร์โพรเซส (Mid) ซึ่งจะอาศัยอยู่ตามเครื่องต่างๆเหมือน pvmd ซึ่งมิดเด็ลแวร์จะมี 2 อย่างดังนี้ 1. ปรับค่าติดตั้งของทรัพยากรระบบให้ถูกต้อง (ความเร็วแต่ละเครื่อง) 2. คอยตรวจสอบสถานะการทำงานของแต่ละ Slave โพรเซสและรายงานผ่านทาง MailBox ระบบ ซึ่งเป็นหน่วยความจำร่วมแบบกระจาย (Distributed Share Memory) ด้วยฟังก์ชั่น pvm_putinfo() และ pvm_getinfo() ตามที่กล่าวไว้แล้วในบทที่ 2

4.2.3 การประยุกต์ใช้มิดเด็ลแวร์

เนื่องจาก Master โพรเซสจะเป็นตัวจัดงานระบบ ดังนั้นมิดเด็ลแวร์จำเป็นจะต้องกำหนดค่าติดตั้งทรัพยากรระบบให้ถูกต้องสำหรับนำไปใช้ในการจัดงานบนระบบ (แม้นว่า PVM จะมีฟังก์ชั่น pvm_config() ที่ให้รายละเอียดของทรัพยากรระบบที่ใช้งานอยู่ก็ตาม แต่ค่าพารามิเตอร์ทางกาย

ภาพเช่น sp นั้นไม่ตรงความเป็นจริง) โดยค่าติดตั้งระบบที่แก้ไข้ใหม่นี้จะรายงานข้อมูลแต่ละเครื่องตาม Schema ที่แสดงดังรูปที่ 4.4 นี้ผ่านทางพื้นที่หน่วยความจำร่วมแบบกระจายของระบบ

HID	Host Name	Arch	Speed	Physical Mem(MB)	Logical Mem(MB)

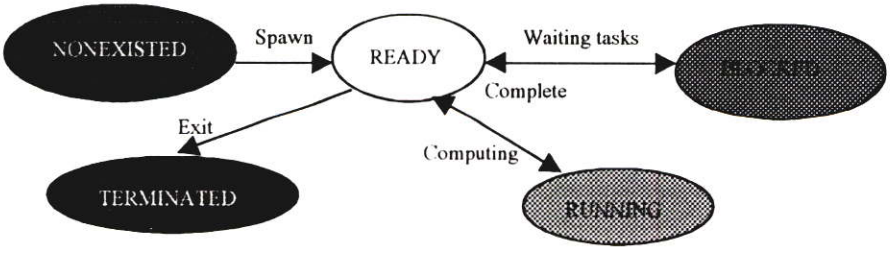
รูปที่ 4.4 Schema ของตาราง Host Config ใหม่

ตารางที่ 4.1 ตัวอย่างข้อมูลตาม Schema ของตาราง Host Config

HID	Host Name	Arch	Speed	Phy Mem	Log Mem
0	Mst	WIN32	755	128	2044
1	Slv1	WIN32	500	64	2044
2	Slv2	WIN32	400	64	2044
3	Slv3	WIN32	233	32	2044

โดย HID 0 จะเป็นข้อมูลของเครื่องหลัก (Master Host) และ หมายเลขอื่นๆก็คือเครื่องลูกข่ายบนระบบเสมือน ซึ่งในตอนเริ่มต้นทำงานโปรแกรม Master จะไปดึงข้อมูลต่างๆที่เก็บในตาราง Host Config นี้ทันที ต่อมา Master โพรเซสจึงจะสร้าง Slave โพรเซสตามเครื่องต่างๆ บนระบบ

ก่อนที่ Master โพรเซสจะส่งงานให้แต่ละ Slave โพรเซสตามเครื่องต่างๆทำงาน Master โพรเซสจะทราบความพร้อมในการทำงานของแต่ละ Slave ก่อน ดังนั้นข้อมูลสถานะการทำงานของ Slave แต่ละตัวจึงเป็นสิ่งจำเป็น แม้ว่า pvmd จะสามารถตรวจสอบสถานะได้ด้วยฟังก์ชั่น pvm_pstat(tid) ก็ตาม แต่ค่าที่แสดงนั้นทำให้ทราบแค่เพียงว่าโพรเซสดังกล่าวยังคงอยู่บนระบบหรือไม่เท่านั้น ซึ่งยังไม่เพียงพอสำหรับนำไปประยุกต์ใช้ในการจัดงานแบบพลวัตบนระบบ ดังนั้นเพื่อให้ชัดเจนมากขึ้น มิดเด็ลแวร์ระบบที่พัฒนาขึ้นนี้จะคอยรายงานสถานะของแต่ละ Slave โพรเซสใหม่ตามรูปที่ 4.5



รูปที่ 4.5 การเปลี่ยนสถานะของ Slave โพรเซส

โดยในตอนเริ่มต้น Slave โพรเซสที่ถูกสร้างมาอยู่ในสถานะ READY ทันทีโดยถ้ามีงานที่ส่งจาก Master โพรเซส สถานะ Slave โพรเซสจะเปลี่ยนเป็น BLOCKED (เพื่อแสดงให้ทราบว่าณขณะนี้ภาระอยู่บนเครือข่ายที่ใช้งาน) และจะเปลี่ยนสถานะกลับเป็น READY เช่นเดิม ในช่วงที่ Slave โพรเซสคำนวณ สถานะจะเปลี่ยนเป็น RUNNING (เพื่อให้ทราบถึงถึงการใช้ทรัพยากรบนเครื่องนั้นๆที่ Slave โพรเซสทำงานอยู่อันได้แก่ CPU time และ หน่วยความจำระบบเป็นต้น)และจะพร้อมกลับมารอรับงานได้อย่างเดิมเป็น READY หลังจากเสร็จสิ้นการคำนวณ จนกว่า Slave โพรเซสจะออกจากระบบ (TERMINATED) หลังจากได้รับสัญญาณให้หยุดการทำงานจาก Master แล้ว โดยจะรายงานสถานะบนระบบหน่วยความจำร่วมแบบกระจายในรูปแบบดังแสดงในรูปที่ 4.6 และตารางที่ 4.2

Tid	Status	Load (sec)	HID

รูปที่ 4.6 Schema ของตาราง Slv Task Status

ตารางที่ 4.2 ตัวอย่างการรายงานสถานะของแต่ละ Slave

Tid	Status	Load (sec)	HID
40001	Ready	20.0	0
80001	Running	0.0	1
C0001	Blocked	18.0	2
100001	Ready	0.0	3

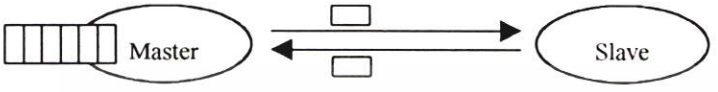
Master โพรเซสจะทราบความพร้อมในการทำงานของแต่ละ Slave โพรเซสในเครื่องต่างๆ ในช่วงที่ทำการจัดงาน/กระจายงานบนระบบ โดยจะแสดงค่าของภาระที่เกิดขึ้นของงานที่ทำเสร็จก่อนหน้านั้นไว้ด้วย สารสนเทศเหล่านี้มีประโยชน์ในการนำไปใช้ปรับภาระงานสมดุลยของบางอัลกอริทึมการจัดงานแบบพลวัตแบบใหม่ที่จะกล่าวในหัวข้อถัดไป

และเพื่อลดค่าสูญเสียบนเครือข่ายให้น้อยที่สุด ดังนั้นถ้าเครื่องทั้งสองมีสถาปัตยกรรมระบบเหมือนกัน Master โพรเซสควรที่จะใช้วิธีเข้ารหัสเมสเส็จแบบ PvmDataInplace แทน (ตามที่กล่าวไว้ในฟังก์ชั่น pvm_mkbuf() และ pvm_initsend() ในบทที่2) และในการส่งผลลัพธ์กลับ Slave โพรเซสควรจะใช้วิธีเข้ารหัสเมสเส็จแบบ PvmDataRaw เพื่อให้ข้อมูลผลลัพธ์คงอยู่ ซึ่งจะช่วยลดค่าสูญได้มากกว่าการใช้ PvmDataDefault อย่างมาก

ในการจัดส่งงานจาก Master โพรเซสแต่ละครั้ง เมสเส็จควรที่จะมีการสร้างหมายเลขแท็กให้เองโดยอัตโนมัติรวมไปถึงที่ Slave โพรเซสเองควรใช้เมสเส็จตัวเดียวกันนี้ (มีหมายเลขแท็กเบอร์เดียวกันกับที่ Master ส่งมาให้) สำหรับบรรจุผลลัพธ์เพื่อส่งกลับคืน เพื่อช่วยในเรื่องของการจัดเรียง

ลำดับผลลัพธ์ทั้งหมดที่ Master โพรเซสโดยดูจากหมายเลขแท็กของเมสเส็จผลลัพธ์ที่ถูกส่งกลับคืน ซึ่งสิ่งเหล่านี้ควรจะมีการจัดการเองโดยอัตโนมัติเพื่อลดความยุ่งยากของการพัฒนาโปรแกรมระบบแทนที่ผู้ใช้ต้องจัดการเองทั้งหมดเหมือนแต่ก่อน

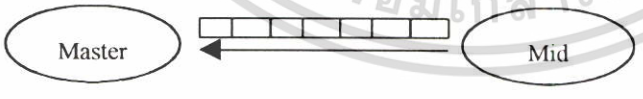
ซึ่งหลังจากการประมวลผลเสร็จในแต่ละครั้ง Slave โพรเซสต้องส่งคืนผลลัพธ์ที่ได้ให้ Master โพรเซสอยู่ตลอดจึงเป็นการเพิ่มภาระบนเครือข่ายอย่างมากตามที่แสดงในรูปที่ 4.7



รูปที่ 4.7 แสดงการส่งผลลัพธ์ของ Slave หลังประมวลผลเสร็จในทันที

ในรูปที่ 4.7 เมื่อ Slave โพรเซสประมวลผลเสร็จก็จะส่งเมสเส็จผลลัพธ์กลับคืนทันทีและในขณะเดียวกัน Master โพรเซสก็จัดส่งงานใหม่ถัดมา ซึ่งจะเห็นได้ว่าทำให้เกิดภาระการใช้งานบนเครือข่ายที่ใช้งานอย่างมาก โดยเฉพาะเป็นช่วงเวลาที่ Master โพรเซสกำลังจัดงาน/กระจายงานด้วย รวมไปถึงมีความเสี่ยงสูงของการเกิดปัญหาการสูญหายเมสเส็จ (Message Loss) ตามมา (ทั้งนี้เนื่องจากในการติดต่อสื่อสารระหว่าง pvmd นั้นใช้ UDP โพรโตคอล แม้ว่าจะทำงานเร็วขึ้นแต่ก็มีความเสี่ยงที่จะเกิดปัญหาการสูญหายของแพ็คเก็จได้ง่าย)

ดังนั้นเพื่อลดค่าสูญเสียที่เกิดขึ้นรวมไปถึงความเสี่ยงของการสูญหายเมสเส็จบนระบบ เทคนิคของการทำ Store&Forward สามารถนำมาใช้แก้ปัญหานี้ได้ โดยการออกแบบให้มิดเด็ลแวร์มีระบบเมสเส็จคิวที่จะคอยเก็บเมสเส็จผลลัพธ์ของ Slave โพรเซสบนเครื่องเดียวกันก่อน ทั้งนี้เพื่อรอส่งกลับคืนเป็นชุดหลังจากที่ Master โพรเซสเสร็จสิ้นการจัดงานบนระบบ และ Slave โพรเซสทำงานชุดหลังสุดที่ถูกส่งมาเสร็จสิ้นดังแสดงรูปที่ 4.8



รูปที่ 4.8 แสดงการส่งเมสเส็จผลลัพธ์จากมิดเด็ลแวร์ด้วยเทคนิค Store&Forward

4.3 การออกแบบและประยุกต์อัลกอริทึมการจัดงานแบบพลวัตบนระบบ PVM

Hummer และ คณะ [4] ได้ศึกษาวิจัยการจัดงานบนระบบที่แตกต่างกัน (Heterogeneous system) พบว่าเป็นเรื่องที่ยากมากที่จะทำการจัดงานบนระบบโดยไม่มีรายละเอียดของทรัพยากรระบบประกอบ ดังนั้นในการจัดงานบนระบบแตกต่างให้พิจารณาจากกำลังการคำนวณของแต่ละเครื่องที่ร้องของานเป็นหลัก โดยทั่วไปกำลังในการคำนวณมักจะเทียบออกมาเป็นเชิงทศนิยม

FLOPS (floating point operation per second) ซึ่งจะหาว่าในช่วงเวลา 1 วินาทีเครื่องนั้นสามารถประมวลผลได้กี่ flop

ในช่วงที่ Slave โพรเซสมีสถานะ RUNNING จะเป็นจุดที่เกิดการคำนวณได้สูงที่สุด (Peak of Computation) ทั้งนี้เนื่องจากระบบปฏิบัติการบนเครื่องนั้นๆจะให้ CPU Time แก่ Slave โพรเซสตัวนั้นได้สูงสุดเกือบ 100% และในคำนวณหาความเร็วสูงสุดของการคำนวณ (peak of speed) บนเครื่องคอมพิวเตอร์ตัวนั้นๆ จะพิจารณาอย่างง่ายๆ คือ 1 flop จะใช้เวลาอย่างน้อยสุดเคือ 1 CPU Clock Cycle time (1 Hz) พอดี ดังนั้นกำลังสูงสุดของการคำนวณที่จะเกิดบนเครื่องนั้นๆก็จะเทียบเท่ากับ ความเร็วซีพียูบนเครื่องนั้น (ในที่นี้จะคิดเฉพาะเครื่อง PC ที่ใช้ 1 CPU และ มีสถาปัตยกรรมแบบ 32 บิต)

ดังนั้นในงานวิจัยนี้จะทำการประยุกต์อัลกอริทึมการจัดงานแบบพลวัตใหม่ให้สัมพันธ์กับความเร็วที่แท้จริงของแต่ละเครื่องบนระบบ รวมไปถึงการประยุกต์บางอัลกอริทึมที่อาจจะนำค่าภาระงานที่เกิดขึ้นจริงบนเครื่องนั้นๆมาใช้ในการปรับภาระงานสมดุลยระบบ ซึ่งจะใช้วิธีกระจายงานบนระบบตาม [10] โดยจะมอบหมายงานให้แก่ Slave โพรเซส ที่มีสถานะเป็น Ready และ อยู่บนเครื่องที่มีความเร็วสูงที่สุดในขณะนั้นก่อน ทั้งนี้เพื่อให้การมอบหมายงานบนระบบได้สอดคล้องกับการทำงานของบางอัลกอริทึมเดิมอย่างเช่น GSS เป็นต้น ดังนั้นในงานวิจัยนี้จึงได้ออกแบบอัลกอริทึมสำหรับการจัดงานแบบพลวัตใหม่ไว้ 3 อัลกอริทึมดังนี้

4.3.1 AWFGSS (Applied Weight Fast Guide Self Scheduling)

เนื่องจากแนวคิดการจัดงานของอัลกอริทึม GSS ดังนั้นแนวคิดของอัลกอริทึม AWFGSS นี้จะให้เกิดการจัดงาน/กระจายงานบนระบบน้อยที่สุด (เท่ากับจำนวนเครื่อง P ที่ใช้งานบนระบบ) โดยการแบ่งภาระงานจะสัมพันธ์กับความเร็วของแต่ละเครื่อง ดังนั้นสมการที่ใช้ในการแบ่งภาระงานแสดงได้ดังนี้

$$C_i = \lceil (RSlv->Sp / HT)\, N \rceil \qquad (1)$$

โดยที่ RSlv คือพอทย์เตอร์ที่ชี้ไปยัง Slaveโพรเซสที่ถูกเลือกให้ได้รับงานในขณะนั้น

RSlv->Sp คือ ความเร็วของการคำนวณบนเครื่องที่ Slave โพรเซสนั้นถูกเลือก

HT คือ ผลรวมความเร็วของทุกๆเครื่องบนระบบเสมือนที่สร้างขึ้น

N คือ ขนาดปัญหาหรือภาระงานที่จะถูกแบ่งสำหรับการประมวลผล

C_i คือ ขนาดของภาระงานที่ถูกแบ่งให้แก่ RSlv ได้ประมวลผล

i คือ จำนวนครั้งที่เกิดการจัดงาน/กระจายงานบนระบบ

อย่างไรก็ตามจะเห็นว่าอัลกอริทึมประยุกต์นี้จะมีความเสี่ยงของการเกิดภาระงานไม่สมดุลยสูงกว่า GSS อัลกอริทึม ทั้งนี้เนื่องจากจะไม่เหลือภาระงานไว้สำหรับปรับภาระงานสมดุลยเลย แม้นว่าจะดีกว่าอัลกอริทึมการจัดงานแบบ CS ก็ตาม

4.3.2 AWGSS (Applied Weight Guide Self Scheduling)

อัลกอริทึมนี้จะแตกต่างจากอัลกอริทึมประยุกต์ที่ 4.3.1 โดยจะเปลี่ยนค่าภาระงานตอนเริ่มต้นของการจัดงานใหม่เพื่อเหลืองานไว้สำหรับปรับภาระงานสมดุลย รวมถึงการลดจำนวนครั้งของการจัดงาน/กระจายงานบนระบบในช่วงทำภาระงานสมดุลยโดยการกำหนดค่าเริ่มต้นภาระงานบนระบบใหม่ด้วยตามสมการที่ (2) และ (3)

$$F = \begin{cases} \lceil N/P \rceil & i=1 \\ \lceil (Slv[0].Sp / HT)\ R_i \rceil & i = P+1 \text{ เริ่มทำภาระงานสมดุลย} \end{cases} \quad (2)$$

$$C_i = \begin{cases} F & RSlv = Slv[0] \\ (RSlv->Sp / Slv[0].Sp)\ F & \end{cases} \quad (3)$$

โดยที่ F – คือขนาดภาระงานระบบตอนเริ่มต้นของการจัดงานและทำภาระงานสมดุลย

R_i – คือขนาดภาระงานที่คงเหลือใน Tasks Pool สำหรับการจัดงานครั้งที่ i

ซึ่งการจัดงานแบบพลวัตมักจะกำหนดขนาดภาระงานไว้สูงสุดคือ $\lceil N/P \rceil$ ดังนั้นอัลกอริทึมนี้จะกำหนดขนาดภาระงานเริ่มต้น (F) ให้แก่ Slave โพรเซสบนเครื่องหลักก่อน (RSlv=Slv[0]) และสำหรับ Slave โพรเซสตัวอื่นๆจะได้ขนาดภาระงานลดลงตามอัตราส่วนเปรียบเทียบความเร็ว (RSlv->Sp/Slv[0].Sp) จากขนาดภาระงานเริ่มต้น ในช่วงที่เริ่มทำภาระงานสมดุลยโอกาสที่ Slv[0] จะพร้อมทำงานก่อนมีมากกว่าดังนั้นขนาดของภาระงานจะถูกปรับตามอัตราส่วนความเร็วเครื่องหลักต่อความเร็วทั้งหมด (HT) จากงานที่คงเหลืออยู่ (R_i) เพื่อลดการจัดงาน/กระจายงานบนระบบ ส่วนSlave โพรเซสบนเครื่องอื่นๆก็ยังคงใช้สมการการแบ่งภาระงานเหมือนเดิม

4.3.3 AFWS (Applied Factor Weight Scheduling)

อัลกอริทึมนี้ได้นำแนวคิดการจัดงานของ Factor Scheduling โดยเหลืองานไว้ครึ่งหนึ่ง (N/2) สำหรับการทำภาระงานสมดุลยบนระบบ ซึ่งงานครึ่งแรกของอัลกอริทึมนี้จะใช้สมการกำหนดขนาดภาระงานคล้ายๆกับอัลกอริทึมที่ 4.3.1 ดังนี้

$$C_i = \lceil (RSlv->Sp / HT)\ N/2 \rceil \quad (4)$$

$$RSlv->Chunks = C_i \quad (5)$$

และสำหรับในช่วงทำภาระงานแบบสมดุลยอัลกอริทึมนี้จะใช้ค่าภาระงานที่เกิดขึ้นจริงของแต่ละ Slave โพรเซสในเครื่องนั้นๆมาปรับขนาดภาระงานใหม่ตามสมการที่ (6) และ (7) นี้

$$minload = Min(\ Slv[1..P].load) \quad (6)$$

$$C_i = \lceil (minload / RSlv\text{->}load)\ RSlv\text{->}Chunks \rceil \quad (7)$$

ในสมการที่ (6) จะคำนวณหาค่าภาระจริงของแต่ละโพรเซส Slv[1..P].load ที่น้อยสุดตั้งแต่เริ่มทำภาระงานแบบสมดุลยบนระบบ (minload > 0) โดยโพรเซสที่ถูกเลือก (RSlv) จะมีการปรับขนาดภาระงานใหม่จากเดิม (RSlv->Chunks) ด้วยอัตราส่วน minload/RSlv->load แทน ทั้งนี้จะทำให้การทำภาระงานแบบสมดุลยมีประสิทธิภาพยิ่งขึ้น

บทที่ 5
การประยุกต์แอ็พพิเคชั่นบนระบบจัดงานพลวัตใน PVM

ในงานวิจัยนี้ได้นำไปประยุกต์ใช้งานจริงกับโปรแกรมสืบค้นรูปภาพซึ่งใช้วิธีการออร์โตโครีโลแกรมและโครีโลแกรมความแตกต่างของค่าสี ทั้งนี้เพื่อให้แอ็พพิเคชั่นโปรแกรมสามารถใช้งานบนระบบ PVM ที่มีสภาพแวดล้อมที่แตกต่างอย่างได้มีประสิทธิภาพ

5.1 การพัฒนาแอ็พพิเคชั่นโปรแกรมคู่ขนาน

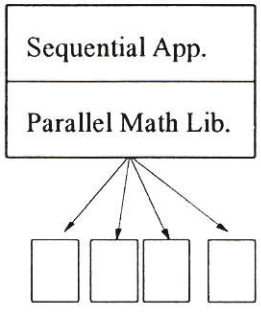
ปัจจุบันมีวิธีการเขียนโปรแกรมแบบคู่ขนาน (Parallel Programming Method) 4 วิธีดังนี้

1. พัฒนาเป็น Parallel Complier เองแต่ค่อนข้างยากเพราะในปัจจุบันการวิเคราะห์โปรแกรมยังไม่ฉลาดพอเหมือนกับให้ผู้เชี่ยวชาญลงมือปฏิบัติเอง แต่ก็มีการออกแบบภาษาสมัยใหม่โดยผู้พัฒนาโปรแกรมสามารถใส่คำแนะนำ (directive) ให้คอมไพล์เลอร์ ตัวอย่างเทคนิคนี้คือ HPF (High Performance Fortran) และ Open MP จึงทำให้การเขียนโปรแกรมได้เร็ว ข้อดีคือเขียนโปรแกรมง่ายแต่ราคาสูงมากๆ เช่นบน IBM SP และบนระบบ Linux Cluster เช่น PGHPF ของบริษัท Porland Group ในสหรัฐ)

2. ออกแบบภาษาใหม่สำหรับใช้เขียนแอ็พพิเคชั่นโปรแกรมคู่ขนานอย่างเช่น OCCAM

3. Message Passing เช่น PVM(Parallel Virtual Machine) และ MPI(Message Passing Interface) ซึ่งเป็นที่นิยมใช้งานมากบนเครื่องซุปเปอร์คอมพิวเตอร์และ Cluster เป็น Free Software และยัง Portable ที่ใช้ได้กับทุกเครื่อง แต่ข้อเสียคือการพัฒนาโปรแกรมยากและต้องใช้อัลกอลิซึมที่มีประสิทธิภาพสูง

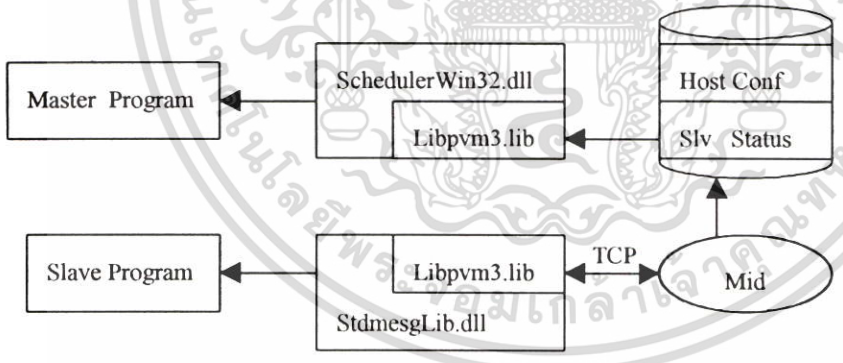
4. เป็นรูปแบบ Library ฟังก์ชั่นที่สามารถทำงานคู่ขนานได้ ซึ่งส่วนใหญ่มักจะเป็นไลบรารีทางคณิตศาสตร์มากกว่าตัวอย่างเช่น PetSc,Scalapack เป็นต้น ข้อดีคือส่วนใหญ่จะเป็นไลบรารีเฉพาะทางโดยเฉพาะคณิตศาสตร์ ซึ่งส่วนใหญ่ไลบรารีเหล่านี้จะรองรับงานพื้นฐานเช่น การแก้สมการทาง Linear Algebra หรือ Differential Equation ตามที่แสดงในรูปที่ 5.1



รูปที่ 5.1 แสดงการพัฒนาแอ็พพิเคชั่นคู่ขนานโดยใช้ไลบรารีทางคณิตศาสตร์

สำหรับบนระบบ Message Passing อย่าง PVM นั้นจะใช้ไลบรารีซึ่งเป็น API (Application Program Interface) สำหรับเชื่อมต่อเพื่อพัฒนาแอ็พพิเคชั่นโปรแกรมคู่ขนาน ดังนั้นเพื่อให้แอ็พพิเคชั่นของระบบสามารถทำภาระงานสมดุลย์พลวัตตามที่กล่าวไว้ในบทที่ 3 และ ให้การพัฒนาแอ็พพิเคชั่นระบบสะดวกมากยิ่งขึ้น ในงานวิจัยนี้จึงได้พัฒนาไลบรารี 2 ชุด คือ ไลบรารี SchedulerWIN32.dll สำหรับพัฒนา Master โปรแกรม และ ไลบรารี Stdmesglib.dll สำหรับพัฒนา Slave โปรแกรมบนระบบ

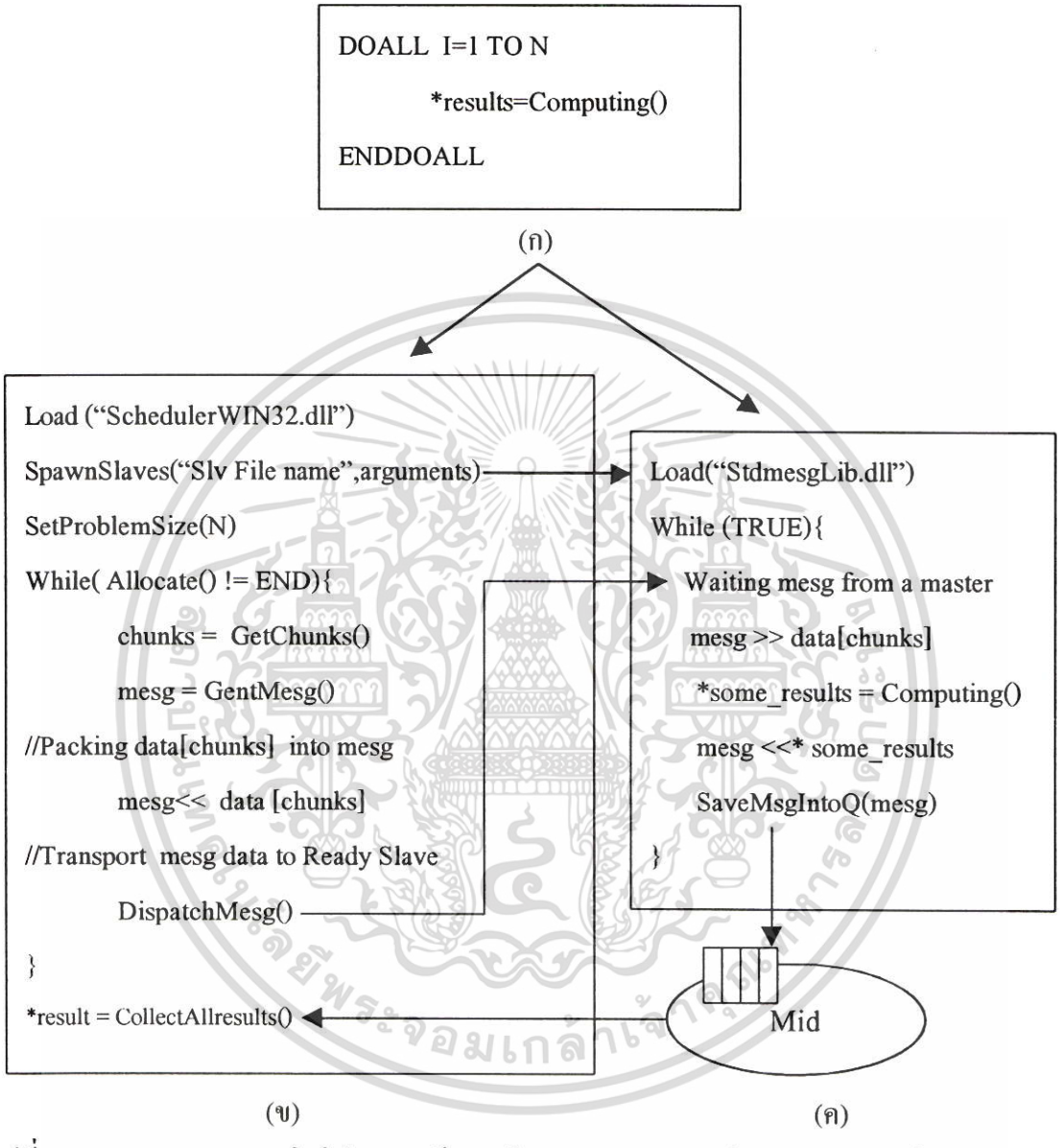
โดยไลบรารีทั้งสองจะห่อหุ้มฟังก์ชั่น API ต่างๆของ libpvm3.lib ไว้ เพื่อลดความยุ่งยากในการจัดการติดต่อสื่อสารระหว่าง Master/Slave ด้วยตัวเอง เช่นการกำหนดและตรวจสอบ แมสเส็จแท็ก (message tag) ที่ใช้งานระหว่าง Master/Slaves โดยอัตโนมัติ และ ซ่อนการจัดการ TID ของโพรเซสทั้งสองด้วยไลบรารี Stdmesglib.dll และ การเข้าถึง MailBox ระบบตามที่ มิดเด็ลแวร์ โพรเซส (Mid) บนเครื่องต่างๆได้จัดเตรียมไว้สำหรับตรวจสอบสถานะความพร้อมการทำงานของ Slave โพรเซส รวมถึง API สำหรับผู้พัฒนาโปรแกรมระบบเรียกใช้งานอัลกอริทึมจัดงานพลวัตสำหรับการจัดงานบนระบบที่มีอยู่แล้วใน SchedulerWIN32.dll (เนื่องจากในงานวิจัยนี้การพัฒนาแอ็พพิเคชั่นต่างๆอยู่บนแพลต์ฟอร์มของ WIN32 ด้วยเครื่องมือ VC++ เวอร์ชั่น 6.0 เป็นหลัก ดังนั้นการพัฒนาไดนามิกไลบรารีระบบจึงสะดวกและการใช้งานจะมีประสิทธิภาพมากกว่าแบบ สแตติกไลบรารี) ดังนั้นการพัฒนาแอ็พพิเคชั่นระบบสามารถแสดงได้รูปที่ 5.2



รูปที่ 5.2 แสดงการเรียกใช้ไลบรารีของ มาสเตอร์และสลาฟโปรแกรมบนระบบ

รูปที่ 5.2 แสดงการเชื่อมต่อของไลบรารีทั้งสองในการพัฒนาแอ็พพิเคชั่นโปรแกรมทั้ง มาสเตอร์และ สลาฟ ซึ่งไลบรารีทั้งสองเสมือนตัวเชื่อมของสองเลเยอร์ตามที่แสดงไว้แล้วในรูปที่ 3.4 Slave โปรแกรมจะสร้าง Connection โดยตรงกับ มิดเด็ลแวร์บนเครื่องเดียวกันทันทีที่ถูกสั่งให้ทำงานจากคำสั่ง SpawnSlvProcess() ซึ่งเป็น API ที่ถูกพัฒนาขึ้นมาใหม่ใน SchedulerWin32.dll (ฟังก์ชั่นดังกล่าวจะห่อหุ้มฟังก์ชั่น pvm_spawn() อีกทีเพื่อสั่งรัน Slave โปรแกรมที่อยู่ตามเครื่องต่างๆแล้ว รวมถึงการจัดการกับข้อผิดพลาดต่างๆเอาไว้แล้ว) ดังนั้นฟังก์ชั่น API ต่างๆที่ถูกพัฒนาไว้แล้วในไลบรารีทั้งสองจะช่วยจัดการให้โปรแกรมทั้งสองสามารถทำงานได้ครบถ้วนตามคุณ

ลักษณะที่ได้กล่าวไว้แล้วในบทที่ 4 โดยสามารถแสดง psudo code ของ Master และ Slave เพื่อนำไปพัฒนาแอ็พพิเคชั่นโปรแกรมบนระบบได้ดังรูปที่ 5.3



รูปที่ 5.3 (ก) จาก DOALL ในโปรแกรมเรียงลำดับ (ข) psudo code ส่วนของ Master โปรแกรม (ค) psudo code ส่วนของ Slave โปรแกรม

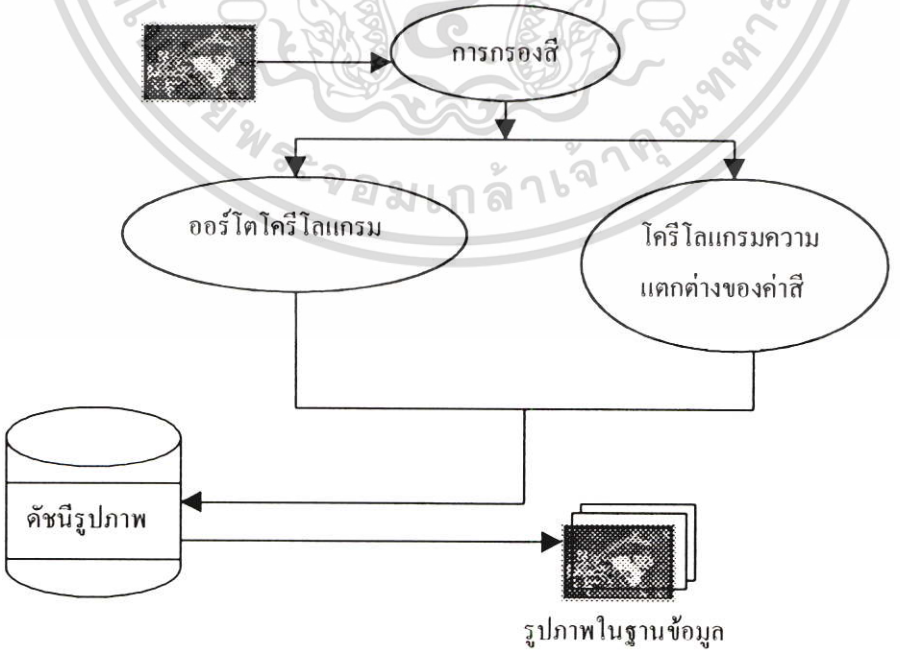
จากรูปที่ 5.3 psudo code ในส่วนของ Master โปรแกรมจะกำหนดขนาดภาระงาน หรือขนาดปัญหาที่จะถูกแบ่งด้วย SetProblemSize(N) โดยการจัดงานบนระบบด้วยฟังก์ชั่น Allocate() ซึ่งขนาดภาระงานย่อย (chunks) หาได้จาก GetChunks() และ ฟังก์ชั่น GentMesg() จะสร้างเมสเส็จเพื่อให้ผู้ใช้ทำการจัดสรรข้อมูลให้สัมพันธ์กับขนาดภาระงานย่อยลงในเมสเส็จดังกล่าวสำหรับใการกระจายงานบนระบบ โดยใช้ฟังก์ชั่น DispatchMesg() เพื่อทำการจัดส่งเมสเส็จสร้างให้ถึง Slave

โพรเซสที่พร้อมทำงานในขณะนั้นๆ เมื่อ Slave ทำงานเสร็จ Mid จะทำการจัดส่งเมสเสจผลลัพธ์ทั้งหมดคืน Master ทันที และเมื่อเมสเสจผลลัพธ์ทั้งหมดมาครบ Master ก็จะทำการจัดเรียงลำดับเมสเสจผลลัพธ์ทั้งหมดใหม่ CollectAllresults() เพื่อให้ผลลัพธ์ทั้งหมดถูกต้อง

5.2 ความเป็นมาของงานวิจัยสืบค้นข้อมูลรูปภาพโดยใช้ออร์โตโครีโลแกรมและโครีโลแกรมความแตกต่างของค่าสี

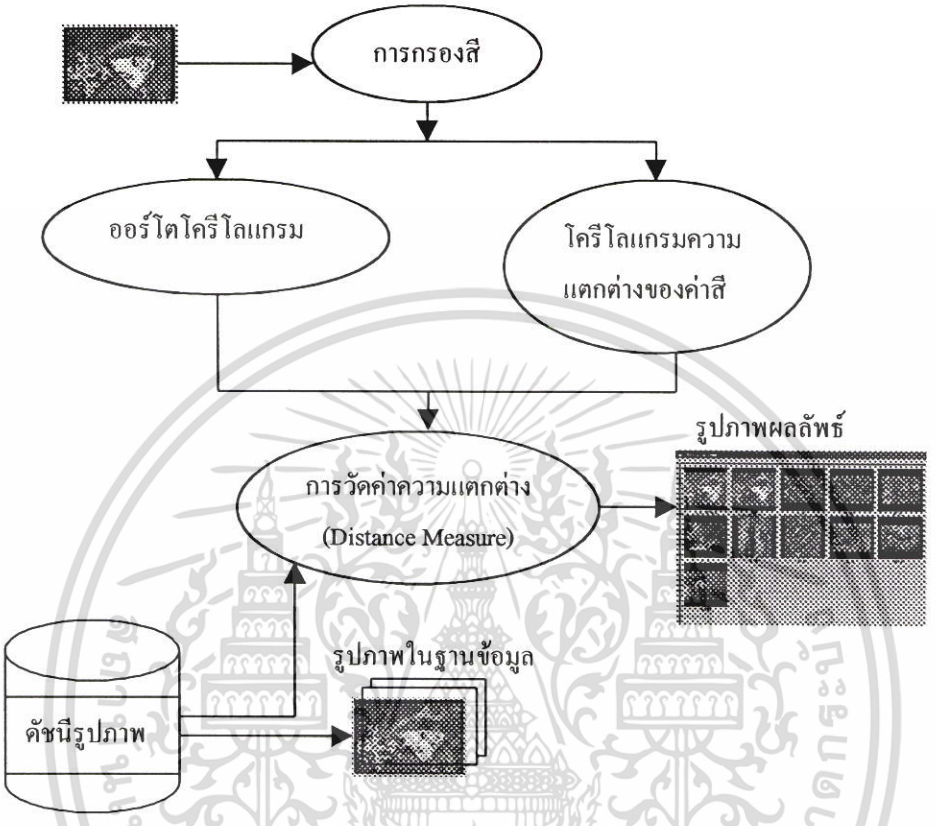
ในปัจจุบันงานวิจัยเกี่ยวกับระบบการสืบค้นคืนข้อมูลรูปภาพมากมาย ซึ่งส่วนใหญ่จะเน้นการสืบค้นรูปภาพโดยใช้คอนเท็นท์ของรูปภาพที่ระบบสามารถให้ผู้ใช้เลือกรูปตัวอย่างที่ต้องการใช้ในการค้นหารูปภาพในฐานข้อมูล ออร์โตโครีโลแกรมและโครีโลแกรมความแตกต่างของค่าสี คือผลงานวิจัยของสุธาสินี [12] ในการสร้างดัชนีและค้นคืนข้อมูลรูปภาพ โดยการดึงลักษณะสำคัญที่ได้รวมข้อมูลสีและข้อมูลเชิงพื้นที่ของรูปภาพเข้าด้วยกัน ทำให้สามารถค้นคืนข้อมูลรูปภาพที่อยู่สถานที่เดียวกัน แต่มุมมองแตกต่างกัน รวมไปถึงรูปภาพที่มีการเปลี่ยนแปลงไปด้วยวิธีต่างๆได้ดี เช่น การหมุน และ การย่อขยาย เป็นต้น

ซึ่งงานวิจัยนี้จะประกอบด้วยขั้นตอนหลักๆ 2 ขั้นตอนคือ การทำดัชนี และ การสืบค้นข้อมูลรูปภาพ โดยขั้นตอนของการทำดัชนีนั้นจะเป็นการคำนวณหาเว็กเตอร์ข้อมูลของออร์โตโครีโลแกรมและโครีโลแกรมความแตกต่างของค่าสีของรูปภาพที่จะถูกจัดเก็บลงฐานข้อมูลระบบ ซึ่งมีขั้นตอนต่างๆดังรูปที่ 5.4



รูปที่ 5.4 แสดงขั้นตอนการทำดัชนีรูปภาพ

สำหรับกระบวนการสืบค้นข้อมูลรูปภาพโดยใช้ออร์โตโครีโลแกรมและโครีโลแกรมความแตกต่างของค่าสีจะให้ผู้ใช้นำรูปภาพตัวอย่างมาเพื่อค้นหารูปภาพในฐานข้อมูลระบบที่คล้ายคลึงกับรูปภาพตัวอย่างมากที่สุด โดยขั้นตอนของการค้นคืนข้อมูลรูปภาพสามารถแสดงได้ดังรูปที่ 5.5



รูปที่ 5.5 แสดงขั้นตอนในการค้นคืนรูปภาพโดยวิธีออร์โตโครีโลแกรมและโครีโลแกรมความแตกต่างของค่าสี

จากรูปที่ 5.5 แสดงให้เห็นว่ามีการคำนวณของรูปภาพตัวอย่างเหมือนกับขั้นตอนในการทำดัชนีรูปภาพ แต่ในการค้นคืนข้อมูลรูปภาพจะนำค่าที่ได้ไปเปรียบเทียบกับค่าออร์โตโครีโลแกรมและโครีโลแกรมความแตกต่างของค่าสีของรูปภาพในฐานข้อมูลด้วยกระบวนการที่เรียกว่า "การวัดความแตกต่าง (Distance Measure)" ซึ่งรูปภาพในฐานข้อมูลที่มีความคล้ายคลึงกับรูปภาพตัวอย่างมากๆจะมีค่าความแตกต่างน้อยๆ ดังนั้นก่อนการแสดงรูปภาพผลลัพธ์จะมีการทำ Ranking เพื่อให้ครบตามจำนวนรูปภาพที่ถูกกำหนดไว้สำหรับการแสดงผล ซึ่งสมการต่างๆที่ใช้การคำนวณในแต่ละขั้นตอนที่ได้กล่าวมาทั้งหมดนี้สามารถดูได้จาก [12]

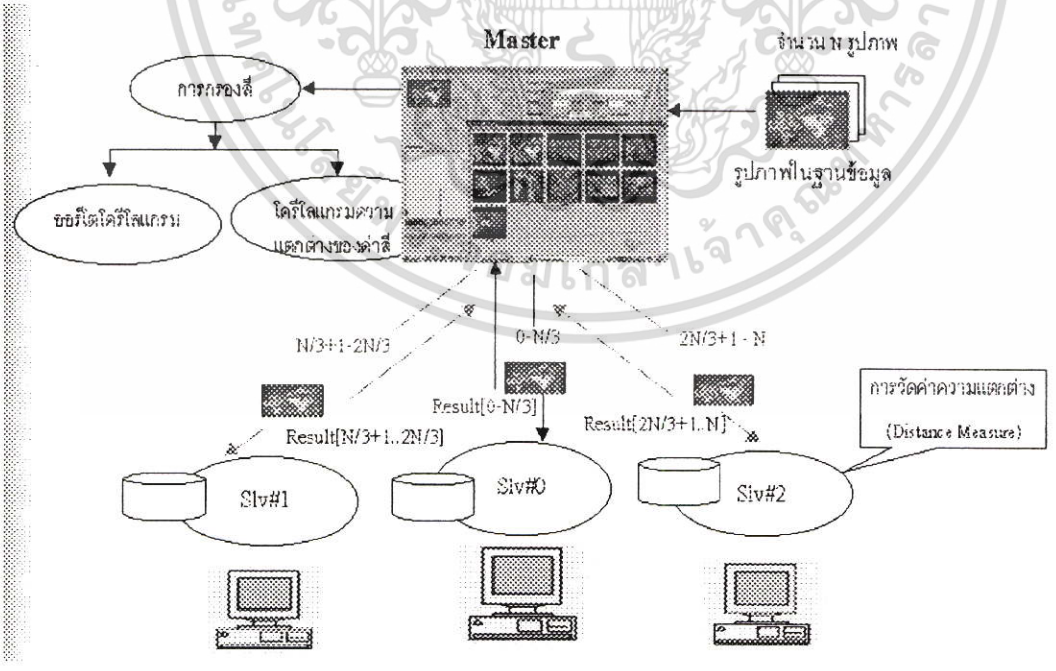
อย่างไรก็ตามในจากงานวิจัยนี้แสดงให้เห็นถึงการคำนวณที่เพิ่มมากขึ้น แม้ว่าวิธีนี้จะมีประสิทธิภาพดีกว่า Histogram และ ออร์โตโครีโลแกรม หรือ โครีโลแกรมความแตกต่างของค่าสีของรูปอย่างใดอย่างหนึ่งซึ่งจะใช้เวลาในการค้นคืนรูปภาพจากฐานข้อมูลเพิ่มขึ้นเกือบ 2 เท่า โดย

เฉพาะอย่างยิ่งเมื่อมีการเก็บรูปภาพลงในฐานข้อมูลมากขึ้น ดังนั้นการประมวลผลแบบคู่ขนานด้วย PVM จะช่วยให้การทำงานดังกล่าวเร็วขึ้น

5.3 การออกแบบแอ็พพิเคชั่นโปรแกรมสืบค้นข้อมูลรูปภาพคู่ขนาน

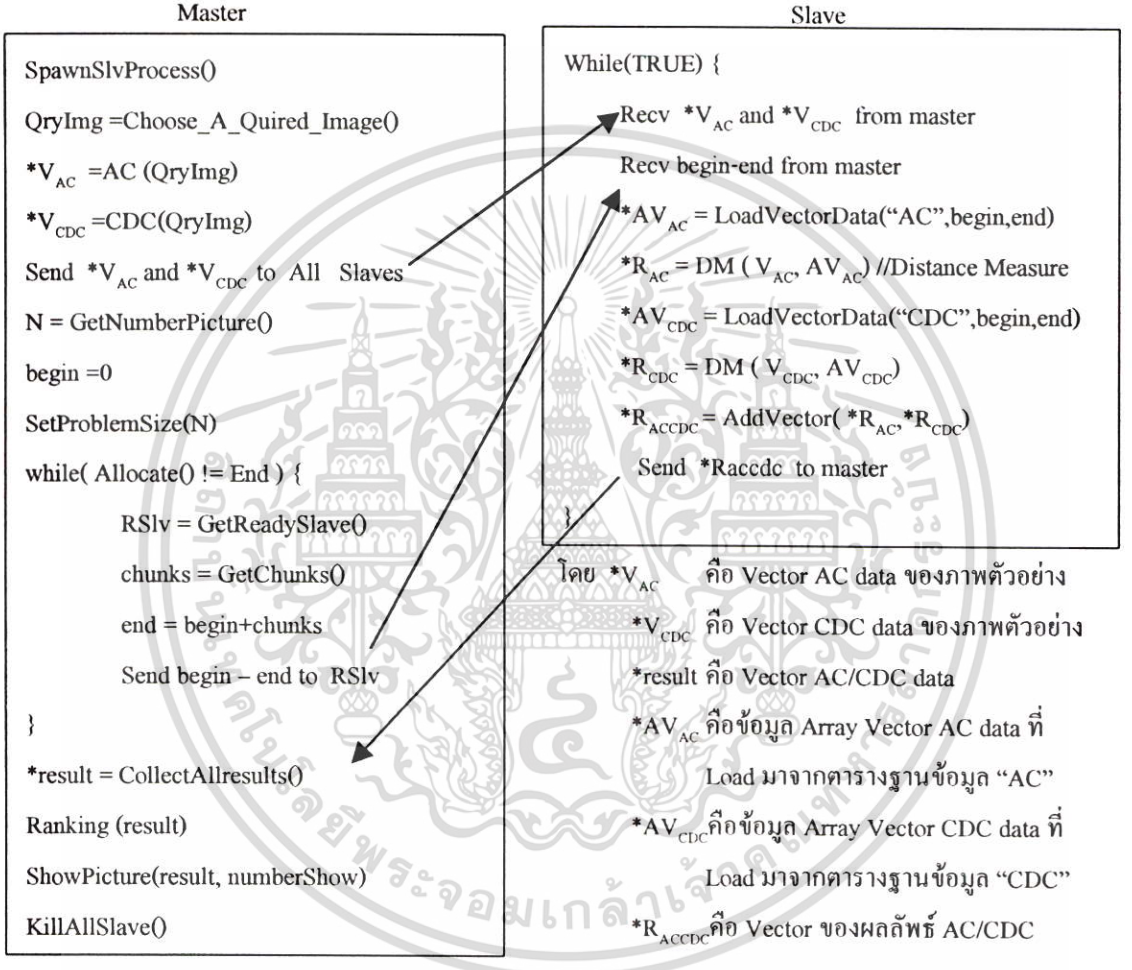
จากรูปที่ 5.5 จะเห็นได้ว่าส่วนที่เป็นภาระงานมากที่สุดของโปรแกรมเรียงลำดับจะอยู่ที่กระบวนการวัดค่าความแตกต่าง (Distance Measure) ซึ่งต้องประมวลผลกับข้อมูลเวิ็กเตอร์ของรูปภาพตัวอย่างและข้อมูลเวิ็กเตอร์ที่ถูกจัดเก็บลงในฐานข้อมูลระบบซึ่งมีจำนวนมากตามจำนวนรูปภาพที่ถูกจัดเก็บลงในฐานข้อมูลระบบ

ดังนั้นการออกแบบแอ็พพิเคชั่นคู่ขนานควรให้แต่ละเครื่องช่วยกันประมวลผลในกระบวนการวัดค่าความแตกต่างนี้อย่างพร้อมเพียงกันโดยนำจำนวน N รูปภาพที่ถูกจัดเก็บอยู่แล้วในฐานข้อมูลมาเป็นภาระงานที่จะถูกแบ่งให้แต่ละเครื่อง แต่เนื่องจากแอ็พพิเคชั่นโปรแกรมเรียงลำดับดังกล่าวสามารถให้ผู้ใช้ทำการเลือกรูปภาพตัวอย่างสำหรับการค้นหาและต้องผ่านกระบวนการคำนวณของ ออร์โตโครีโลแกรมและโครีโลแกรมความแตกต่างของค่าสีก่อน ดังนั้นรูปแบบ Master/Slave (Mandelbrot Algorithm) จึงเหมาะสมสำหรับการพัฒนาแอ็พพิเคชั่นโปรแกรมระบบ โดยให้ Master โปรแกรมสามารถแสดงผลแบบกราฟฟิก (GUI) เพื่อให้ผู้ใช้เลือกรูปภาพตัวอย่าง รวมถึงจัดแบ่งงานให้แต่ละ Slave โพรเซสประมวลผล และทำการรวบรวมผลลัพธ์จาก Slave ทั้งหมดก่อนจัดแสดงรูปภาพออกมา ดังแสดงรูปที่ 5.6



รูปที่ 5.6 แสดงการทำงานแบบคู่ขนานของโปรแกรมสืบค้นรูปภาพด้วยวิธีออร์โตโครีโลแกรมและโครีโลแกรมวัดความแตกต่างของค่าสี

รูปที่ 5.6 หลังจากที่ Master ผ่านกระบวนการออร์โตโคลีโรแกรม และ โครีโลแกรมค่าความแตกต่างของสีของรูปภาพตัวอย่างแล้ว ก็จะทำการจัดส่งข้อมูลเว็กเตอร์ของรูปตัวอย่างให้ทุกๆ Slave โพรเซส จากนั้นก็จะทำการแบ่งงานขนาด N และกำหนดช่วงการเข้าถึงข้อมูลของแต่ละ Slave เพื่อโหลดข้อมูลจากฐานข้อมูลที่กระจายอยู่แล้วตามเครื่องต่างๆ แล้ว (ทั้งนี้เพื่อลดค่าสูญเสียของการติดต่อสื่อสารบนเครือข่ายให้น้อยที่สุด) ก่อนทำการประมวลผลกระบวนการวัดค่าความแตกต่างอย่างพร้อมเพียงกันตาม psuedo code ที่แสดงในรูปที่ 5.7



รูปที่ 5.7 แสดง psudo code ทั้ง Master และ Slave

รูปที่ 5.7 ขนาดปัญหาที่ถูกแบ่งมีขนาด N โดยฟังก์ชั่น GetReadySlave() จะเป็น API ฟังก์ชั่นระบบที่จะเลือก Slave โพรเซสที่เหมาะสมจะได้งานมากที่สุด ส่วน API ฟังก์ชั่น Allocate() จะทำการแบ่งภาระงานให้แก่ Slave โพรเซสที่ถูกเลือกโดยอัตโนมัติตามอัลกอริทึมการจัดงานแบบพลวัตที่ผู้ใช้เลือก ซึ่งสำหรับฐานข้อมูลระบบ และ รูปภาพต่างๆที่ใช้ทดสอบในแอ็พพิเคชั่นนี้มาจากงานวิจัย [12]

บทที่ 6

การทดลองและผลการทดลอง

6.1 ออกแบบการทดลอง

6.1.1 โปรแกรมทดสอบภาระงานบนระบบ

ส่วนใหญ่แอ็พพิเคชั่นโปรแกรมคู่ขนานนำไปใช้แก้ปัญหา2 ประเภทหลักๆดังนี้

1. ปัญหาเล็ก (Small Problem) เป็นแอ็พพิเคชั่นโปรแกรมที่ใช้ทรัพยากร CPU Time ของเครื่องนั้นๆมากกว่า ตัวอย่างของแอ็พพิเคชั่นเหล่านี้ได้แก่ฟังก์ชั่นการคำนวณค่าอินทรีเกตต่างๆและแอ็พพิเคชั่นประเภทประมวลผลสัญญาณ เป็นต้น
2. ปัญหาใหญ่ (Large Problem) เป็นแอ็พพิเคชั่นโปรแกรมที่ต้องใช้ทรัพยากรระบบจำนวนมากในการแก้ปัญหาโดยเฉพาะ CPU Time และ ขนาดพื้นที่หน่วยความจำของเครื่อง เป็นต้น โดยตัวอย่างแอ็พพิเคชั่นประเภทนี้ได้แก่ Vector Processing และแอ็พพิเคชั่นที่เกี่ยวข้องกับการประมวลผลข้อมูลที่มีจำนวนมหาศาลเช่น Data Mining หรือ Image Processing เป็นต้น

ดังนั้นในงานวิจัยนี้จะพัฒนาแอ็พพิเคชั่นระบบเพื่อประเมินประสิทธิภาพของแต่ละอัลกอริทึมการจัดงานแบบพลวัต ซึ่งโปรแกรมที่นิยมใช้ทดสอบอย่างแพร่หลายของปัญหาแต่ละประเภทดังนี้

1. โปรแกรมคำนวณค่า π

เป็นตัวอย่างของแอ็พพิเคชั่นโปรแกรมที่แก้ปัญหาซับซ้อนทางคณิตศาสตร์ด้วยวิธีการคำนวณอย่างง่ายๆแต่ต้องทำการคำนวณแบบวนซ้ำหลายๆรอบเพื่อให้ได้ผลลัพธ์ ตามฟังกชั่นอินทริเกรตนี้

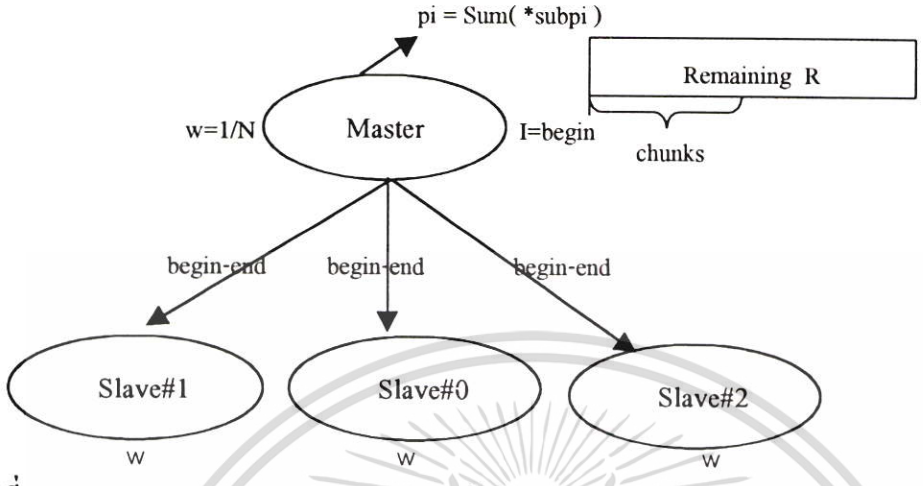
$$\int_0^N \frac{4}{(1+x^2)} dx = \pi$$

สามารถแสดงโปรแกรมได้ดังรูปที่ 6.1

```
N = 1000000000, w=1/N
DOALL  I=1,N
      t=(I +0.5)* w
      pi =pi +4.0 /( 1.0 +t*t)
ENDDO
```

รูปที่ 6.1 โปรแกรมเรียงลำดับของการคำนวณค่า π

จากโปรแกรมเรียงลำดับในรูปที่ 6.1 สามารถออกแบบการกระจายงานของแอ็พพิเคชั่นโปรแกรมคู่ขนานได้ตามรูปที่ 6.2



รูปที่ 6.2 แสดงการกระจายงานบนระบบของโปรแกรมคำนวณค่า π

ในรูปที่ 6.2 หลังจาก Master สั่งรัน Slave โปรแกรมตามเครื่องต่างๆแล้วจะทำการคำนวณ w=1/N และส่งให้ทุกๆ Slave โพรเซสก่อนจะทำการแบ่งงานบนระบบโดยจะกำหนดช่วงเริ่มต้นและสิ้นสุด (end=begin+chunks) ให้แก่แต่ละ Slave โพรเซส จนกระทั่งไม่เหลืองาน (R=0) และสุดท้ายก็จะนำแต่ละผลลัพธ์ย่อย (*subpi) ทั้งหมดมารวมกันเป็นคำตอบค่า pi ในการทดลองนี้ได้กำหนดขนาดภาระงาน (N) ไว้ 3 ระดับคือ 1.0E+09, 2,147,483,647 (ค่าสูงสุดของเลขจำนวนเต็ม) และ 4,294,967,295 (ค่าสูงสุดของข้อมูลเลขจำนวนเต็มทสนิยมไม่ติดเครื่อง)

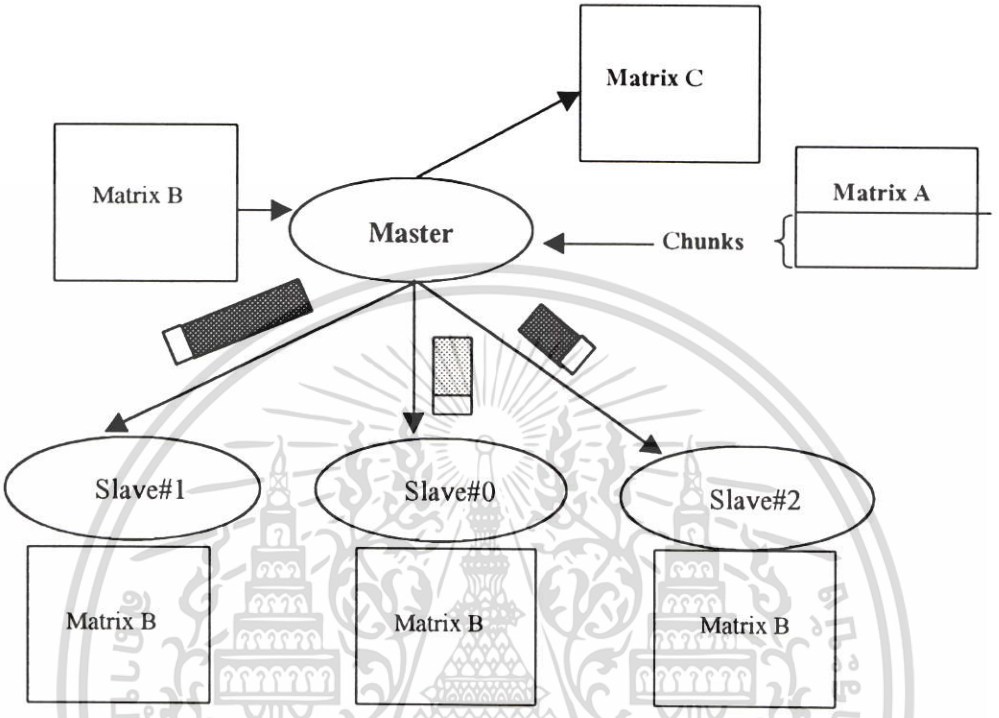
2. โปรแกรมคูณเมตริกซ์ NxN

เป็นแอ็พพิเคชั่นที่ต้องประมวลผลกับข้อมูลจำพวก เว็กเตอร์, อาเรย์ โดยโปรแกรมจะทำการคำนวณแมตริกซ์ A,B และ C ที่มีขนาดเท่ากันหมดคือ NxN ดังแสดงในรูปที่ 6.3

```
DOALL I=1,N
        DO J= 1,N
          DO K=1,N
                  C[I,J] += A[I,K] * B[K,J]
          ENDDO
        ENDDO
ENDOALL
```

รูปที่ 6.3 โปรแกรมคำนวณเมตริกซ์ขนาด Nx N

ในการทดลองนี้ได้กำหนดขนาดของภาระงานสำหรับทดสอบไว้ 3 ระดับคือ 800,1000 และ 1500 ซึ่งจะต้องมีการจับจองพื้นที่หน่วยความจำของเครื่องอย่างมหาศาลในการประมวลผลของแอ็พพิเคชั่นนี้ ดังนั้นการทำงานของแอ็พพิเคชั่นระบบนี้สามารถแสดงได้ดังรูปที่ 6.4



รูปที่ 6.4 แสดงการกระจายภาระงานของโปรแกรมคูณเมตริกซ์

รูปที่ 6.4 Master โปรแกรมจะกระจาย Matrix B ให้แก่ทุกๆ Slave โพรเซสบนระบบก่อนจะทำการแบ่ง Matrix A เพื่อส่งให้แต่ละ Slave ทำงานจนกระทั่งการแบ่งงานสิ้นสุด แต่ละผลลัพธย่อยของ Matrix C จะถูกจัดเรียงลำดับใหม่มีขนาด NxN พอดี

นอกจากนี้ในงานวิจัยนี้ได้ทดสอบกับโปรแกรมสืบค้นข้อมูลรูปภาพด้วยวิธีการออร์โตโครีโลแกรมและโครีโลแกรมค่าความแตกต่างของค่าสีตามที่กล่าวไว้ในบทที่ 5 แล้ว ทั้งนี้เพื่อแสดงให้เห็นถึงการนำไปประยุกต์ใช้งานกับแอ็พพิเคชั่นจริงๆ ซึ่งลักษณะปัญหาของแอ็พพิเคชั่นดังกล่าวจะคล้ายๆกับโปรแกรมคูณแมตริกซ์นั้นเอง

6.1.2 ตัววัดประสิทธิภาพระบบ

ค่าความเร็วที่เพิ่มขึ้น (Speed Up) จะเป็นพารามิเตอร์ที่บ่งบอกถึงประสิทธิภาพการจัดงานแบบพลวัตของแต่ละอัลกอริทึมได้ดีที่สุด ซึ่งถ้าค่าน้อยกว่า 0 แสดงให้เห็นว่าเกิดกรณีแย่ที่สุดบนระบบ (เวลาที่ใช้ในการประมวลผลหลายเครื่องนานกว่าเวลาที่ประมวลผลบนเครื่องเดียว) โดยเวลาประมวลผลของแต่ละโปรแกรมเรียงลำดับ (T_{seq}) จะวัดที่เครื่องบนระบบที่มีความเร็วมากสุด (เครื่อง Master ของระบบ) และ เวลาในการประมวลผลแบบคู่ขนาน (T_{par}) จะวัดตั้งแต่เวลาที่โปรแกรม Master เริ่มทำงานจนกระทั้งสิ้นสุด

6.1.3 เครื่องมือและการจัดสภาพแวดล้อมสำหรับทดสอบ

ตารางที่ 6.1 สเป็กของแต่ละเครื่องที่ใช้ทดสอบ

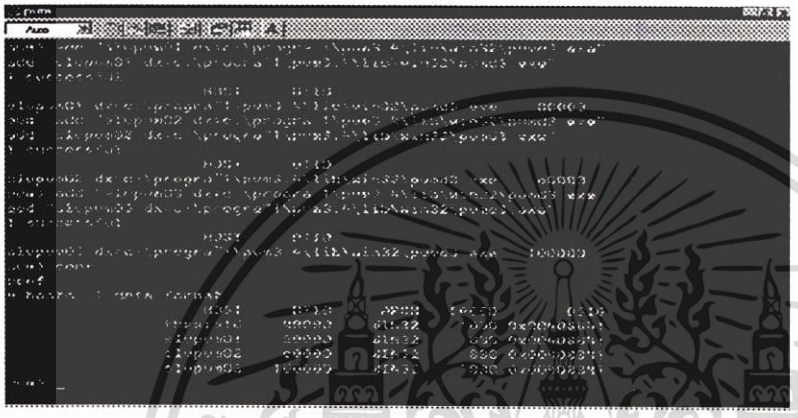
ชื่อเครื่อง	CPU Spec	RAM
Master	AMD THUNDERBIRD 755 Mhz	SDRAM 128 MB
Slvpvm01	ASUS ATHLON 550 Mhz	EDO 64 MB
Slvpvm02	ASUS CELERON 400 Mhz	EDO 64 MB
Slvpvm03	Pentium MMX 233 Mhz	EDO 32 MB

แต่ละเครื่องจะเชื่อมต่อบนเครือข่ายเดียวกัน ที่ห้องวิจัย คณะเทคโนโลยีสารสนเทศ ซึ่งมีความเร็ว 10 Mbit/Sec โดยแต่ละเครื่องลงซอฟท์แวร์ระบบ PVM เวอร์ชั่น 3.4.3 ที่สามารถทำงานร่วมกันได้กับระบบปฏิบัติการ WIN32 และ UNIX ซึ่งทุกๆเครื่องที่ทดสอบนี้ใช้ระบบปฏิบัติการ Windows 98 เหมือนกันหมด สำหรับการพัฒนาแอ็พพิเคชั่นระบบใช้ Microsoft Visual C++ 6.0 ส่วนแอ็พพิเคชั่นโปรแกรมสืบค้นรูปภาพด้วยวิธี AC/CDC นั้นโปรแกรม Master จะถูกพัฒนาโดย Microsoft Visual Basic 6.0 ซึ่งยังคงเป็นโปรแกรมรูปแบบเดิมอยู่เพียงแต่ได้แก้ไขเพิ่มโดยการเรียกใช้ API ที่ถูกพัฒนาในไลบรารี SchedulerWIN32.dll และ สำหรับโปรแกรม Slave จะถูกพัฒนาขึ้นใหม่ด้วย VC++ 6.0 โดยระบบฐานข้อมูลที่ใช้คือ Microsoft Access 97 และ ไฟล์รูปภาพทั้งหมด 15,694 ภาพที่มีขนาดและรูปแบบแตกต่างกันเช่น BMP,JPEG และ GIF เป็นต้น โดยโปรแกรมสืบค้นรูปภาพด้วยวิธี AC/CDC นี้ยังคงประมวลผลแบบเรียงลำดับบนเครื่องเดียวได้ทั้งนี้เพื่อให้ผู้ใช้สามารถใช้เปรียบเทียบผลลัพธ์ของทั้งโปรแกรมเรียงลำดับและโปรแกรมคู่ขนาน รวมถึงความแตกต่างของเวลาที่ตอบสนองจากการรันทั้งสองแบบได้อย่างชัดเจน

ในการทดลองนี้ได้ออกแบบสภาพแวดล้อมระบบ (Virtual Machine) ที่เครื่องหลัก (Master Host) จะต้องทำงานร่วมกับแต่ละเครื่องที่มีสเป็กแตกต่างกันด้วยแอ็พพิเคชั่นต่างๆ ตามตารางที่ 6.2

ตารางที่ 6.2 สภาพแวดล้อมแต่ละ Virtual Machine (VM) สำหรับทดสอบ

Het Env	เครื่องลูกข่ายที่ทำงานร่วม
VM1	Slvpvm01
VM2	Slvpvm01,Slvpvm03
VM3	Slvpvm01,Slvpvm02 และ Slvpvm03



รูปที่ 6.5 แสดงการเพิ่มเครื่องบนระบบ PVM ด้วยโปรแกรม PVM Console เพื่อสร้าง Virtual Machine (VM) ตามที่ออกแบบไว้

6.2 ผลการทดลอง

6.2.1 เวลาประมวลผลแบบเรียงลำดับของแต่ละแอ็พพิเคชั่น

ตารางที่ 6.3 โปรแกรมคำนวณค่า π และโปรแกรมคูณแมตริกซ์ขนาด NxN

N	T_{seq} (Sec)
1.0E+09	113
2,147,483,647	242
4,294,967,295	481

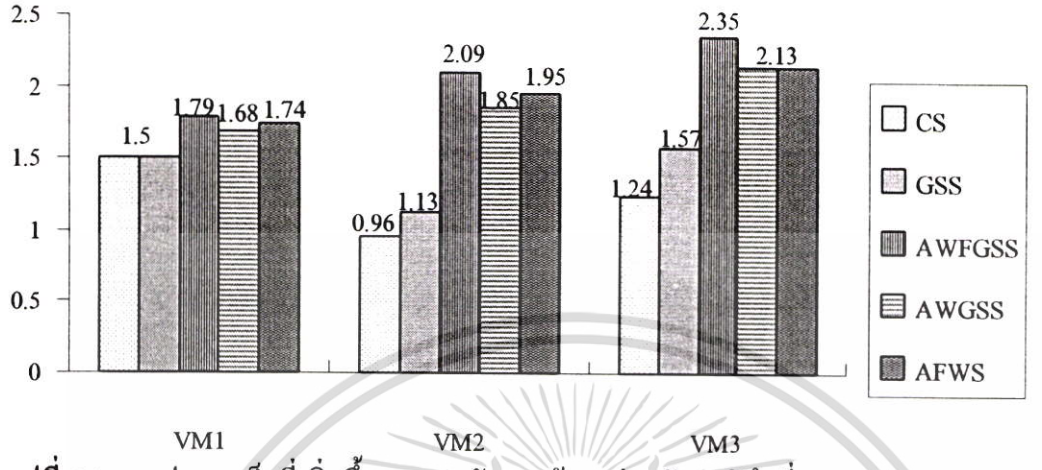
N	T_{seq} (Sec)
800	197
1000	386
1500	1340

ตารางที่ 6.4 โปรแกรมสืบค้นข้อมูลรูปภาพด้วยวิธีการ AC/CDC

N	T_{seq} (Sec)
15,694	260

6.2.2 อัตราเร็วที่เพิ่มขึ้นจากการจัดงานบนระบบของแต่ละอัลกอริทึม

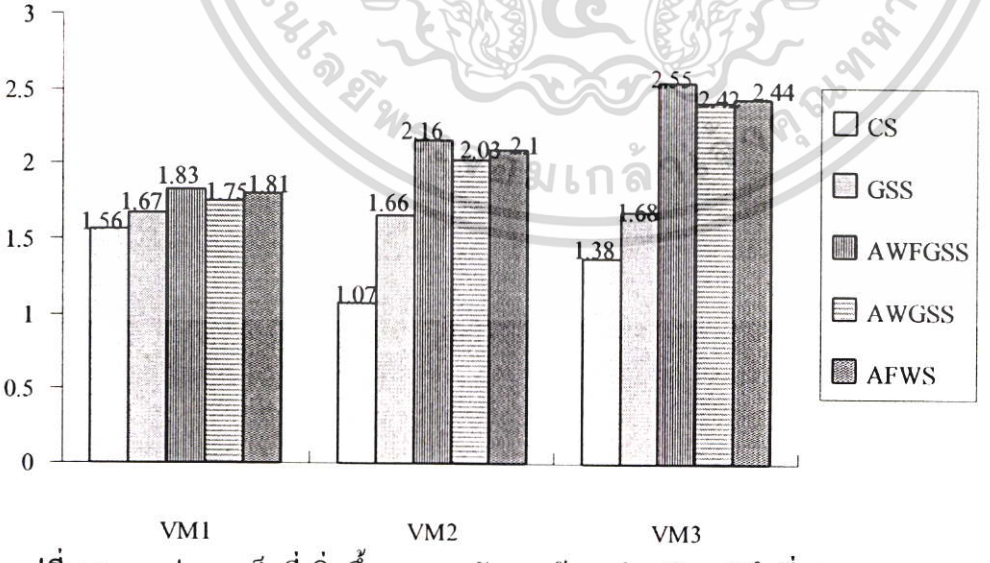
1. โปรแกรมคำนวณค่า π



รูปที่ 6.6 กราฟความเร็วที่เพิ่มขึ้นจากการจัดงานด้วยแต่ละอัลกอริทึมที่ N=1.0E+09

ตารางที่ 6.5 ค่าความเร็วที่เพิ่มขึ้นของแต่ละอัลกอริทึมที่ N=1.0E+09

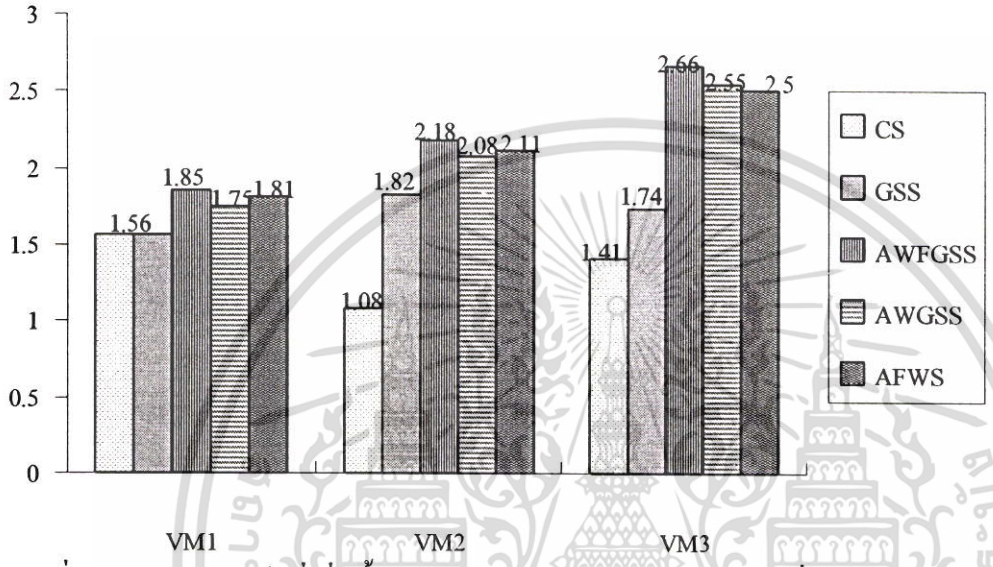
	CS	GSS	AWFGSS	AWGSS	AFWS
VM1	1.5	1.5	1.79	1.68	1.74
VM2	0.96	1.13	2.09	1.85	1.95
VM3	1.24	1.24	2.35	2.13	2.13



รูปที่ 6.7 กราฟความเร็วที่เพิ่มขึ้นจากการจัดงานด้วยแต่ละอัลกอริทึมที่ N=2147483647

ตารางที่ 6.6 ค่าความเร็วที่เพิ่มขึ้นของแต่ละอัลกอริทึมที่ N=2147483647

	CS	GSS	AWFGSS	AWGSS	AFWS
VM1	1.56	1.67	1.83	1.754	1.806
VM2	1.07	1.66	2.16	2.03	2.1
VM3	1.38	1.68	2.55	2.42	2.44



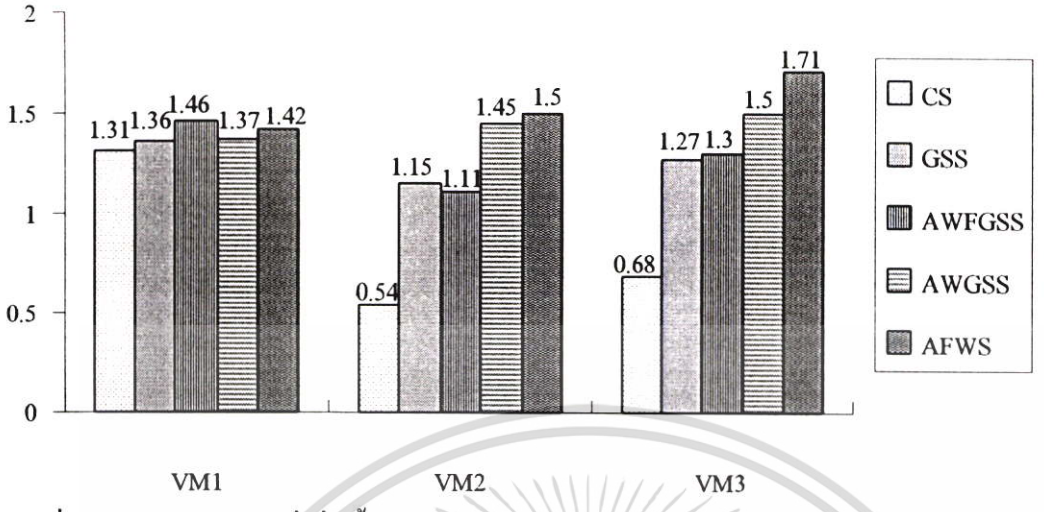
รูปที่ 6.8 กราฟความเร็วที่เพิ่มขึ้นจากการจัดงานด้วยแต่ละอัลกอริทึมที่ N=4294967295

ตารางที่ 6.7 ค่าความเร็วที่เพิ่มขึ้นของแต่ละอัลกอริทึมที่ N=4294967295

	CS	GSS	AWFGSS	AWGSS	AFWS
VM1	1.56	1.56	1.85	1.7	1.8
VM2	1.08	1.82	2.18	2.08	2.11
VM3	1.41	1.74	2.66	2.55	2.5

จากผลการทดลองของแอ็พพิเคชั่นนี้จะเห็นได้ว่าอัลกอริทึมประยุกต์การจัดงานแบบพลวัตทั้งหมดจะมีประสิทธิภาพเหนือกว่าอัลกอริทึมการจัดงานแบบพลวัตแบบเดิม GSS และ CS อย่างมาก โดยจะเห็นได้อัลกอริทึมประยุกต์การจัดงานแบบพลวัต AWFGSS จะมีประสิทธิภาพเหนือกว่าอัลกอริทึมทั้งหมด ส่วนอัลกอริทึมประยุกต์ทั้งสองจะเห็นได้ว่าประสิทธิภาพในการจัดงานกับแอ็พพิเคชั่นประเภทนี้พอๆกัน ไม่ว่าที่ขนาดปัญหาเพิ่มขึ้นเท่าไรก็ตาม

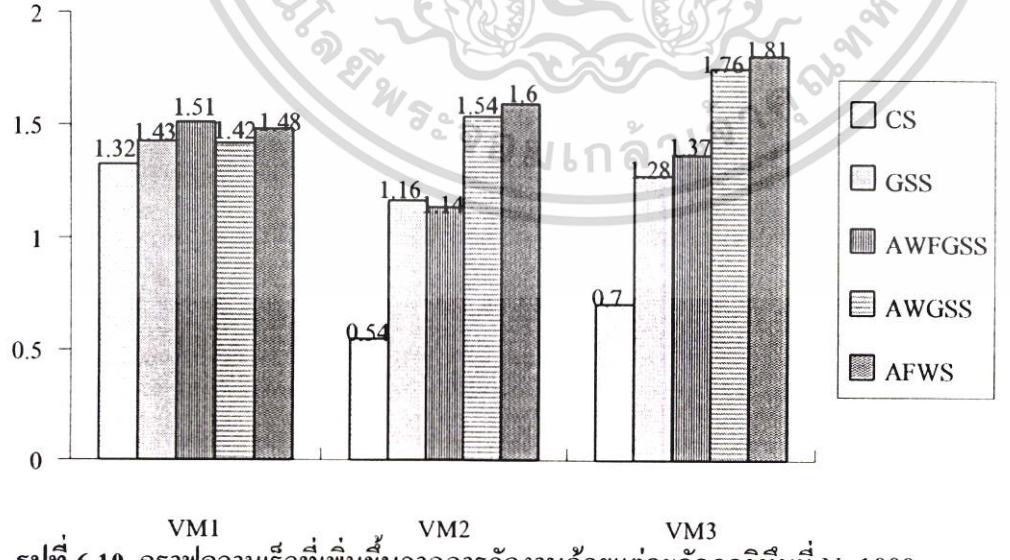
2. โปรแกรมคูณแมตริกซ์ขนาด NxN



รูปที่ 6.9 กราฟความเร็วที่เพิ่มขึ้นจากการจัดงานด้วยแต่ละอัลกอริทึมที่ N=800

ตารางที่ 6.8 ค่าความเร็วที่เพิ่มขึ้นของแต่ละอัลกอริทึมที่ N=800

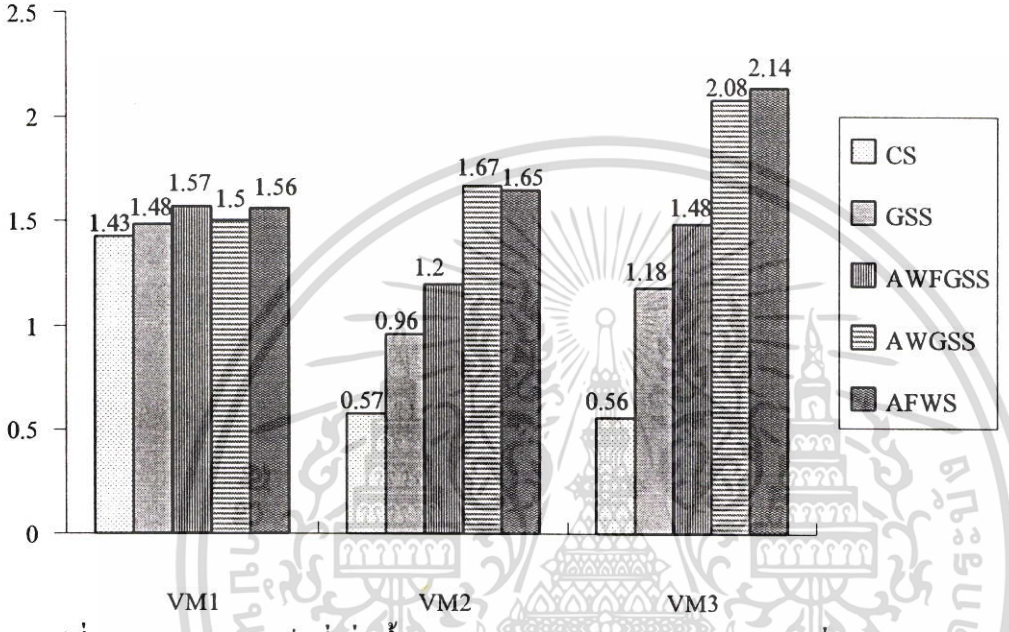
	CS	GSS	AWFGSS	AWGSS	AFWS
VM1	1.31	1.36	1.46	1.37	1.42
VM2	0.54	1.15	1.11	1.45	1.5
VM3	0.68	1.27	1.3	1.64	1.71



รูปที่ 6.10 กราฟความเร็วที่เพิ่มขึ้นจากการจัดงานด้วยแต่ละอัลกอริทึมที่ N=1000

ตารางที่ 6.9 ค่าความเร็วที่เพิ่มขึ้นของแต่ละอัลกอริทึมที่ N=1000

	CS	GSS	AWFGSS	AWGSS	AFWS
VM1	1.32	1.43	1.51	1.42	1.48
VM2	0.54	1.16	1.14	1.54	1.6
VM3	0.7	1.28	1.37	1.76	1.81



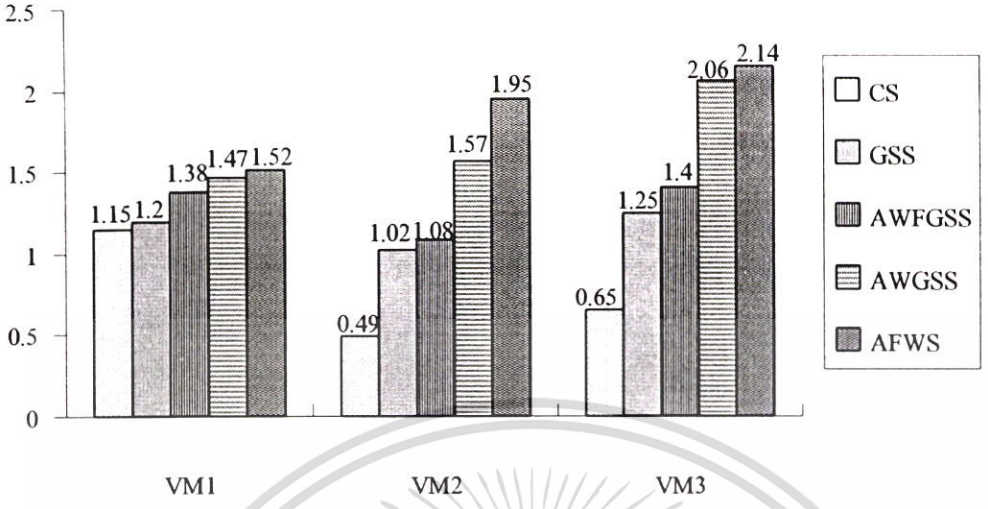
รูปที่ 6.11 กราฟความเร็วที่เพิ่มขึ้นจากการจัดงานด้วยแต่ละอัลกอริทึมที่ N=1500

ตารางที่ 6.10 ค่าความเร็วที่เพิ่มขึ้นของแต่ละอัลกอริทึมที่ N=1500

	CS	GSS	AWFGSS	AWGSS	AFWS
VM1	1.43	1.48	1.57	1.5	1.56
VM2	0.57	0.96	1.2	1.67	1.65
VM3	0.56	1.18	1.48	2.08	2.14

สำหรับแอ็พพิเคชั่นนี้จะเห็นได้ว่าอัลกอริทึมแบบ CS เกิดกรณีแย่สุด ส่วน GSS จะมีประสิทธิภาพด้อยกว่าอัลกอริทึมประยุกต์อย่างมากโดยเฉพาะที่สภาพแวดล้อมแบบ VM2 และ VM3 ตามลำดับ ส่วนอัลกอริทึมประยุกต์ AWFGSS จะทำงานได้ดีเฉพาะสภาพแวดล้อมแตกต่างแบบ VM1 เท่านั้น ซึ่งตรงข้ามกับอัลกอริทึมประยุกต์แบบ AWGSS และ AFWS ที่ยังคงมีประสิทธิภาพไม่ว่าขนาดปัญหาหรือภาระงานเพิ่มขึ้นและทำงานบนสภาพแวดล้อมระบบที่แตกต่างแบบใดก็ตาม ซึ่งจากผลการทดลอง AFWS ให้ผลที่ดีกว่า

3. โปรแกรมสืบค้นข้อมูลรูปภาพด้วยวิธีการ AC/CDC



รูปที่ 6.12 กราฟความเร็วที่เพิ่มขึ้นจากการจัดงานด้วยแต่ละอัลกอริทึมที่ N=15694 ภาพ

ตารางที่ 6.11 ค่าความเร็วที่เพิ่มขึ้นของแต่ละอัลกอริทึมที่ N=15694 ภาพ

	CS	GSS	AWFGSS	AWGSS	AFWS
VM1	1.15	1.2	1.38	1.47	1.52
VM2	0.49	1.02	1.08	1.57	1.95
VM3	0.65	1.25	1.4	2.06	2.14

สำหรับการทดสอบกับแอ็พพิเคชั่นโปรแกรมนี้เพื่อสนับสนุนให้เห็นถึงการนำไปประยุกต์ใช้งานจริงกับหลายๆแอ็พพิเคชั่นที่แก้ปัญหาคล้ายๆกับโปรแกรมคูณเมตริกซ์ เช่นโปรแกรมคำนวณแบบคู่ขนานเพื่อหาหลักฐานยิงของปืนใหญ่ขนาดเบากระสุนวิถีโค้ง [21] เป็นต้น ซึ่งจากผลการทดลองจะเห็นได้ว่าผลการทดลองนี้จะให้ผลคล้ายกับของโปรแกรมคูณเมตริกซ์ NxN

6.3 ปัจจัยที่มีผลต่อการทำงานคู่ขนานบนระบบ

ในการทำงานคู่ขนานของแอ็พพิเคชั่นโปรแกรมถ้าพิจารณาเปรียบเทียบกับการทำงานแบบ Client/Server จะเห็นได้ว่า Slave โปรแกรมที่รันอยู่ตามเครื่องต่างๆจะทำหน้าที่เหมือนกับเป็น Server ของเครื่องนั้นๆที่คอยให้บริการงานที่ส่งมาจาก Master เสมือนเป็น client ของระบบ ดังนั้นการกำหนดค่าพารามิเตอร์ต่างๆบนระบบปฏิบัติการมีส่วนสำคัญอย่างมากในการที่จะให้ Slave โพรเซสทำงานได้อย่างประสิทธิภาพบนเครื่องนั้นๆ ตัวอย่างเช่นบนระบบปฏิบัติการ Windows ที่กำหนดความละเอียดของจอภาพที่ใช้งานมากกว่า 16 bit ก็สามารถให้น้อยลงได้ เพราะ Slave โพร

เซสจะทำงานอยู่เบื้องหลัง (background) ดังนั้นจึงไม่จำเป็นต้องใช้ความละเอียดสูงซึ่งเป็นการสิ้นเปลืองหน่วยความจำระบบ นอกจากนั้นในส่วนของฮาร์ดิสก์ที่ใช้งานตามเครื่องต่างๆถ้าสามารถกำหนดให้ใช้เทคโนโลยี DMA (Direct Memory Access) และ เลือกใช้ File System ระบบเป็น Network Server (พารามิเตอร์เหล่านี้สามารถตั้งได้บนระบบปฏิบัติการ Windows 98 SE) ได้จะช่วยเพิ่มประสิทธิภาพการทำงานของ Slave โพรเซสได้อย่างมาก รวมถึงการทำ Defragment ฮาร์ดิสก์อย่างสม่ำเสมอ เป็นต้น

นอกจากนั้นความเร็วของระบบเครือข่ายก็เป็นปัจจัยสำคัญอย่างมาก ซึ่งถ้าระบบเครือข่ายที่ใช้งานในขณะนั้นมีแบนด์วิธ (Bandwidth) สูงก็จะช่วยลดค่าสูญเสียเนื่องจากการติดต่อสื่อสารบนเครือข่ายน้อยลง รวมไปถึงการออกแบบอัลกอริทึมและการพัฒนาแอ็พพิเคชั่นคู่ขนานที่ดีก็จะช่วยให้การทำงานบนคู่ขนานบนระบบมีประสิทธิภาพมากยิ่งขึ้น ตัวอย่างเช่นการเข้ารหัสเมสเส็จที่มีสถาปัตย์กรรมของผู้ส่งและผู้รับเหมือนกันด้วย PvmDataInPlace/PvmDataRaw จะช่วยลดค่าสูญเสียของการจัดเก็บข้อมูลและแตกข้อมูลของเม็สเส็จตัวนั้น (ไม่ต้องผ่านกระบวนการเข้ารหัส หรือ ถอดรหัสข้อมูล) ซึ่งจะเป็นการลดจำนวนข้อมูลที่รับส่งบนเครือข่ายอีกด้วย แต่ถ้าบางเครื่องลูกข่ายใช้ระบบปฏิบัติการอื่นที่ต่างจากเครื่องหลัก (เครื่องที่ Master โพรเซสที่ทำงานอยู่และจัดส่งงานมาให้) ดังนั้นแล้วคุณลักษณะดังกล่าวก็จะไม่ได้ช่วยอะไร

บทที่ 7
สรุปผลและข้อเสนอแนะ

7.1 สรุปผลการทดลอง

เนื่องจากในการพัฒนาโปรแกรมบนระบบส่งผ่านเมสเสจอย่าง PVM นั้นต้องใช้อัลกอริทึมการจัดงานระบบที่มีประสิทธิภาพสูง โดยเฉพาะอย่างยิ่งเมื่อทำงานบนสภาพแวดล้อมระบบแตกต่างปัญหาภาระงานไม่สมดุลย์มักจะเกิดขึ้นตามมา ซึ่งอาจจะส่งผลถึงกรณีแย่ที่สุดที่เวลาในการตอบสนองแบบคู่ขนาน (T_{par}) จะนานกว่าเวลาที่ประมวลผลบนเครื่องเดียวที่มีความเร็วมากสุดบนระบบ (T_{seq}) ดังนั้นแนวคิดการจัดงานพลวัตคือการจัดแบ่งงาน/กระจายงานให้มีหลายขนาดภาระงานในช่วงรันโปรแกรมระบบทั้งนี้เพื่อให้เกิดการทำภาระงานสมดุลย์บนระบบ (ลดภาระงานไม่สมดุลย์ที่เกิดขึ้น)

สำหรับอัลกอริทึมประยุกต์การจัดงานพลวัตที่ออกแบบนี้จะสัมพันธ์กับความเร็วของแต่ละเครื่องที่ Slave โพรเซสทำงานอยู่ ซึ่งจากผลการทดลองจะเห็นได้ว่าอัลกอริทึมประยุกต์จะมีประสิทธิภาพเหนือกว่าอัลกอริทึมการจัดงานแบบพลวัตเดิมที่ใช้บนเครื่อง SMP อย่างมาก ทั้งนี้เนื่องจากประสิทธิภาพของการจัดงาน/กระจายงานบนระบบส่งผ่านเมสเสจขึ้นอยู่กับระบบเครือข่ายที่ใช้ ดังนั้นการจัดงาน/กระจายงานบนระบบที่เกิดขึ้นแต่ละครั้งจะมีค่าสูญเสียที่มากกว่าบนเครื่อง SMP รวมถึงการแบ่งภาระงานของอัลกอริทึมเดิมนั้นจะไม่สัมพันธ์กับความเร็วที่แต่ละเครื่องสามารถตอบสนองได้ ดังนั้นเวลาในการตอบสนองแบบคู่ขนานจึงไม่ดีเท่าที่ควร และ มีความเสี่ยงต่อการเกิดภาระงานไม่สมดุลย์บนระบบสูงด้วย

จากผลการทดลองของทั้งสามแอ็พพิเคชั่นจะเห็นได้ว่าอัลกอริทึมประยุกต์การจัดงานแบบพลวัตแบบ AWFGSS จะให้ผลลัพธ์ที่ดีสุดในการรันแอ็พพิเคชั่นเพื่อแก้ปัญหาเล็ก (Small Problem) อย่างโปรแกรมคำนวณค่า π ตามขนาดปัญหาหรือขนาดภาระงานเพิ่มขึ้นและยังสามารถทำงานได้ดีกับทุกๆสภาพแวดล้อมระบบแตกต่าง แต่สำหรับอีก 2 แอ็พพิเคชั่นจะพบว่าอัลกอริทึมประยุกต์แบบนี้ทำงานได้ดีเฉพาะบนสภาพแวดล้อมแบบ VM1 ซึ่งความเร็วของเครื่องลูกข่ายไม่แตกต่างกับเครื่องหลักมากนัก ดังนั้นบนสภาพแวดล้อมแตกต่างแบบ VM2 และ VM3 เมื่อต้องรันแอ็พพิเคชั่นคู่ขนานอย่างโปรแกรมคูณเมตริกซ์ และ โปรแกรมสืบค้นข้อมูลรูปภาพด้วยวิธีการ AC/CDC อัลกอริทึมประยุกต์แบบ AWGSS และ AFWS จะมีประสิทธิภาพเหนือกว่า โดยอัลกอริทึมประยุกต์ AWFS นี้จะปรับขนาดของภาระงานให้สัมพันธ์กับค่าภาระ (load) ที่เกิดขึ้นจริงของแต่ละ Slave โพรเซสในช่วงทำภาระงานสมดุลยระบบ นั้นแสดงให้เห็นว่าประเภทของแอ็พพิเคชั่นที่รันบนระบบมีส่วนสำคัญในการเลือกอัลกอริทึมประยุกต์สำหรับการจัดงานแบบพลวัตด้วย

ปัญหาสำคัญของการออกแบบอัลกอริทึมการจัดงานแบบพลวัตคือถ้าเกิดการจัดงาน/กระจายงานบนระบบที่มากเกินไปทำให้เกิดค่าสูญเสียของการจัดงานและกระจายงานที่สูงแม้นว่าจะดีในแง่ของการทำภาระงานสมดุลย์ระบบก็ตาม แต่ถ้าออกแบบอัลกอริทึมให้เกิดการจัดงาน/กระจายงานบนระบบที่น้อยไป (เกิดการกระจายงานบนระบบอย่างน้อยเท่ากับจำนวนเครื่องหรือจำนวนโพรเซสเซอร์ที่ใช้งานพอดี) ก็จะมีความเสี่ยงของการเกิดปัญหาภาระงานไม่สมดุลย์บนระบบได้สูงเช่นกัน ดังนั้นในการออกแบบอัลกอริทึมการจัดงานแบบพลวัตบนระบบส่งผ่านเมสเส็จที่ต้องทำงานคู่ขนานบนสภาพแวดล้อมแตกต่างนั้นควรให้สัมพันธ์กับความเร็วของแต่ละเครื่องที่ใช้งานบนระบบ รวมไปถึงในช่วงของการทำภาระงานสมดุลย์ระบบถ้าขนาดของภาระงานสามารถปรับเปลี่ยนให้สัมพันธ์กับค่าภาระที่เกิดขึ้นจริงจากการทำงานก่อนหน้าบนเครื่องนั้นๆได้ จะทำให้เวลาในการตอบสนองดีขึ้นด้วย

7.2 ข้อจำกัดของงานวิจัย

เนื่องจากการจัดงานระบบในงานวิจัยนี้จะใช้ได้เฉพาะแอ็พพิเคชั่นคู่ขนานที่ใช้วิธีการแบ่งข้อมูลเป็นหลัก (Data Decomposition) ดังนั้นจึงไม่สามารถใช้ได้กับแอ็พพิเคชั่นคู่ขนานที่ใช้วิธีการแบ่งงานตามหน้าที่ (Function Decomposition) ทั้งนี้เนื่องจากในการพัฒนาโปรแกรมบนระบบส่งผ่านเมสเส็จนั้นเป็นเรื่องที่ยากมากที่จะทำการแบ่งโค้ดโปรแกรมทีละบรรทัดหรือเป็นโมดูลฟังก์ชั่นและส่งให้แต่ละเครื่องบนระบบประมวลผลทันที ดังนั้นแนวทางการพัฒนาแอ็พพิเคชั่นโปรแกรมคู่ขนานบนระบบส่งผ่านเมสเส็จก็คือพัฒนาโปรแกรมระบบที่สามารถสั่งทำงานได้ เช่นไฟล์ .exe ต่างๆ ให้กระจายอยู่ตามเครื่องต่างๆไว้ก่อนเพื่อรอการสั่งประมวลผลอย่างพร้อมเพียงกัน เช่นการสั่งฟังก์ชั่น pvm_spawn() จาก Master โปรแกรมบนระบบ PVM หรือ การใช้โปรแกรม mpirun บนระบบ MPI เพื่อสั่งให้โปรแกรมดังกล่าวเริ่มทำงานอย่างพร้อมเพียงกัน ดังนั้นบนระบบส่งผ่านเมสเส็จจึงมักนิยมพัฒนาแบบโปรแกรมเดียวแต่จัดการกับข้อมูลได้หลากหลายเรียกว่า SPMD (Single Program Multiple Data)

และเนื่องจากในงานวิจัยนี้ทดลองกับระบบปฏิบัติการ WIN32 เป็นหลักโดยเฉพาะ Windows 98 แต่ก็ยังสามารถทำงานร่วมกันได้กับทุกๆตระกูลของ WIN32 อย่าง Windows 2000 และ NT 4.0ได้ด้วย ดังนั้นจึงไม่ได้ทำการทดสอบร่วมกับระบบปฏิบัติการตัวอื่นๆอย่าง Linux หรือ UNIX Free BSD เป็นต้น ซึ่งอาจจะทำให้ผลลัพธ์ที่ได้มีความเคลือนไปได้บ้างเมื่อแต่ละเครื่องใช้ระบบปฏิบัติการที่แตกต่างกัน (เป็นผลจากต้องทำบางกระบวนการเพิ่มขึ้นอย่างการเข้ารหัส/ถอดรหัส และ จำนวนข้อมูลที่ส่งผ่านบนเครื่อข่ายมีมากขึ้น) แต่สิ่งเหล่านี้สามารถชดเชยได้โดยการกำหนดค่าพารามิเตอร์บนระบบปฏิบัติการนั้นๆให้เหมาะสมกับการทำงานเบื้องหลังของ Slave โพรเซส รวมถึงการเชื่อมต่อของแต่ละเครื่องบนระบบ ถ้าเป็นแบบ Dedicate จะดีกว่า เพราะลดปัญหา

การจราจรบนระบบเครือข่ายที่ต้องใช้งานร่วมกับผู้ใช้คนอื่นๆ และควรเลือกใช้ Hub ที่ทำงานแบบ cut though จะลดค่าสูญเสียได้มากกว่าแบบ Store&Forward

7.3 ข้อเสนอแนะในการพัฒนางานวิจัยต่อไป

จากผลการทดลองจะเห็นได้ว่าอัลกอริทึมประยุกต์การจัดงานแบบพลวัตสามารถแก้ปัญหาบนระบบส่งผ่านเมสเส็จที่มีสภาพแวดล้อมแตกต่างกันได้ ดังนั้นความเป็นได้ในการพัฒนาต่อไปก็คือถ้าตัวจัดงานระบบถูกพัฒนาให้สามารถเก็บข้อมูลของการจัดงาน/กระจายงานด้วยอัลกอริทึมการจัดงานแบบพลวัตแต่ละแบบ ทั้งนี้เพื่อนำข้อมูลเหล่านี้มาเป็นสถิติไปประยุกต์ใช้งานร่วมกับอัลกอริทึมประยุกต์ (Adaptive algorithms) สำหรับการรันงานคู่ขนานในครั้งต่อไป โดยที่ Master หรือโปรแกรมจัดงานระบบสามารถนำข้อมูลเหล่านี้มาใช้ก่อนเริ่มทำงาน (Start Up) จะทำให้ลดค่าสูญเสียของการจัดงาน/กระจายงานในช่วงรัน และ การแบ่งงานระบบก็จะมีความแม่นยำมากยิ่งขึ้น

และเนื่องจากบางอัลกอริทึมประยุกต์การจัดงานแบบพลวัตอย่าง AWFGSS นั้นสามารถตอบสนองได้ดีที่สุดกับบางแอ็พพิเคชั่นและบางสภาพแวดล้อมระบบเท่านั้น ดังนั้นถ้าสามารถพัฒนาต่อไปให้ตัวจัดงานระบบสามารถสร้างสภาพแวดล้อมด้วยการเลือกเพิ่ม/ลด เครื่องที่ใช้บนระบบได้เอง ทั้งนี้เพื่อให้ได้เวลาการตอบสนองของแต่ละแอ็พพิเคชั่นโปรแกรมคู่ขนานที่ดีที่สุด โดยอาจจะต้องมีการเก็บข้อมูลเพื่อนำมาวิเคราะห์ก่อนตัดสินใจจะเพิ่มหรือลดเครื่องใดๆบนระบบ รวมไปถึงคุณลักษณะเด่นของ PVM ที่สามารถทำการเพิ่ม/ลดจำนวนเครื่องบนระบบได้ง่ายกว่าระบบส่งผ่านเมสเส็จแบบอื่นๆอย่าง MPI เป็นต้น

และสำหรับบางแอ็พพิเคชั่นที่ต้องการความทนทานสูง (fault tolerance) ในกรณีเช่นการป้องกันการล้มของเครื่องที่ใช้งานอยู่ (Host Fail) ดังนั้นถ้าสามารถทำการโยกย้ายงานจากเครื่องเก่าไปยังเครื่องใหม่และให้โพรเซสที่รันบนเครื่องใหม่เริ่มทำงานต่อจากที่ทำงานค้างอยู่ได้ (Process Migration) ซึ่งถ้าพัฒนาโปรแกรม Mid โพรเซสต่อเพื่อให้มีความสามารถดังกล่าวก็จะช่วยให้แอ็พพิเคชั่นดังกล่าวทำงานบนระบบได้อย่างมีประสิทธิภาพยิ่งขึ้น โดยทั้งนี้ PVM ก็มีคุณลักษณะของการแจ้งข้อผิดพลาดต่างๆ (notify) ให้ระบบรับทราบได้อยู่แล้วเช่นกัน

และเนื่องจากแอ็พพิเคชั่นระบบคู่ขนานที่ทดสอบนั้นเป็นแบบ DOALL แต่ถ้าบางแอ็พพิเคชั่นที่เป็น DOACROSS ซึ่งจะเกิดความพึ่งพาข้อมูลในช่วงที่ประมวลผลแบบคู่ขนานได้ ดังนั้นจะหาวิธีการจัดการความพึ่งพาข้อมูลที่จะเกิดขึ้นอย่างไร อย่างเช่นให้ Master โปรแกรมสามารถวิเคราะห์ช่วงของแต่ละรอบ (I) ที่จะเกิดความพึ่งพาข้อมูลได้ก่อนจากนั้นจึงจัดให้อยู่ในกลุ่มงานเดียวกันก่อนส่งให้ประมวลผลหรืออาจให้ Slave โพรเซสสามารถแลกเปลี่ยนข้อมูลได้ ซึ่งจะต้องใช้ฟังก์ชั่นที่เกี่ยวข้องกับ Collective communication ที่มีประสิทธิภาพสูง ในการกำหนดจังหวะการทำงานร่วมกันของ Slave โพรเซสที่อยู่ในกลุ่มเดียวกัน

7.4 แนวโน้มงานวิจัยที่เกี่ยวข้องในอนาคต

ในอนาคตอันใกล้นี้มีหลายงานวิจัยที่พยายามจะการออกแบบและพัฒนาเลเยอร์ Parallel Virtual Machine บน Infrastructure ระดับใหญ่อย่าง INTERNET ซึ่งเป็นแหล่งทรัยากรระบบที่มีขนาดมหาศาลในการตอบสนองภาระงานที่หลากหลายได้ทั่วโลก ในปัจจุบันหลายงานวิจัยพยายามกำหนดมาตรฐานในการพัฒนาเลเยอร์ดังกล่าว เช่น GRID Computing , GDC (Generalized Distributed Computing) ที่ทำงานบนระบบเครือข่ายแตกต่างของ PVM เป็นต้น ซึ่งจะทำให้แต่ละระบบคลัสเตอร์ที่ใช้งานแตกต่างกันทั่วโลก (เช่น PVM,MPI หรือ WINWOLF ของ MicroSoft) เกิดความเป็นหนึ่งเดียวกัน (Single Image)

ดังนั้นจึงมีความเป็นไปได้ที่จะเกิดงานวิจัยหลายๆด้านเพื่อการจัดการทรัพยกรระบบขนาดใหญ่ดังกล่าวเช่น การพัฒนาตัวจัดงานระบบแบบกระจาย (Distributed Scheduler) , การควบคุมงานและทรัพยากรระบบ (Resource Management) , การจัดการภาระงานสมดุลย์แบบอัตโนมัติ (Automatic Load Balancing) และ การโยกย้ายงานไปตามเครื่องบนระบบ (Procress Migration) เพื่อให้สามารถเริ่มงานที่ค้างอยู่จากเครื่องเดิมได้ รวมไปถึงการออกแบบและพัฒนาระบบหน่วยความจำแบบกระจาย (Distributed Share Memory) เพื่อความสะดวกในการพัฒนาแอ็พพิเคชั่นใช้งานบนระบบ (ลดความยุ่งยากในการออกแบบและจัดการติดต่อสื่อสารของแอ็พพิเคชั่นโปรแกรมด้วยตัวเองบนระบบส่งผ่านเมสเส็จ)

หนังสืออ้างอิง

[1] G.A. Geist. **PVM: Parallel Virtual Machine A User's Guide and Tutorial**, MIT Press , London , 1994.

[2] Tang and Yew, "Scheduling loops by compiler (really the runtime-system)", ICPP 1986

[3] Polychronopolous, "Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers" ,IEEE Transactions on Computers, Dec. 1987

[4] Hummel, Schmit, Uma, และ Wein, "Weighted Factoring under heterogeneous system", SPAA 1996

[5] Paul Graham, "OpenMP A Parallel Programming Model for Share Memory Architectures", Edinburgh Parallel Computing Centre, University of Edinburgh, March 1999 Version 1.1

[6] Oscar Plata , and Francisco F. Rivera , " Dynamic Scheduling on Distributed-Memory Multiprocessor ", and "Combining Static and Dynamic Scheduling on Distributed-Memory Multiprocessor " , Dept Electronics and Computing University of Santiago SPAIN , 1998

[7] Kelvin K. Yue ,and David J. Lilja , "Design Multiprocessor Scheduling Algorithm Using a Distributed Genetic Algorithm System " , Dept Comp Sci and Electrical Engineering , University of Minnessota , 1997

[8] CHRISTOPHER WADE HUMPHRES , "A Load Balancing for the PVM Software System", Dept Electrical Engineering, University of Alabama ,1995

[9] L. Hluchy, J. Astalos, "Dynamic Load Balancing under PVM", Institute of Computer System, Slovak Academy of Sciences

[10] J.C. FABERO, I. MARTIN A. BAUTISTA , S. MOLINA, "Dynamic Load Balancing in Heterogeneous Environment under PVM ", Institute of Computer System , Slovak Academy of Sciences , IEEE , 1996

[11] Pawel Czarnul ,"DAMPVM:Dynamic Allocation and Migration on PVM" , Euro PVM/MPI-99 Conference, OAK Springer , 1999

[12] สุธาสินี ฉิมเล็ก, "การค้นคืนข้อมูลรูปภาพโดยใช้ออร์โตโครีโลแกรมและโครรีโลแกรมความแตกต่างของค่าสี", สาขาเทคโนโลยีสารสนเทศ บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, พ.ศ.2545 หน้าที่ 19-32

[13] G.A.Geist, J.A.Kohl, P.M.Papadopoulos and S.L.Scott , "Beyond PVM 3.4: What We 've Learned, What 's Next , and Why " ,Oak Ridge National Laboratory, TN, 37831-6367, USA

[14] Markus Fischer, Jack Dongarra, "Another Architecture: PVM on Windows 95/NT", Mathematical Science Section, Oak Ridge National Laboratory, TN, 37831-6367, USA

[15] Adam Beguelin, Jack Dongarra, AI Geist, Robert Manchek and Vaidy Sunderam, "Recent Enhancements to PVM", University of Tennessee and Oak Ridge National Laboratory, USA

[16] J.J. Dongarra, G.A.Geist, R.J.Manchek and P.M. Papadoponlos, "Adding Context and Static Groups into PVM", Computer Science and Mathematics Division, Oak Ridge National Laboratory, TN, 37831-6367, USA

[17] G.A.Geist, J.A.Kohl, R.J.Manchek and P.M. Papadoponlos, "New Features of PVM 3.4 and Beyond" , Computer Science and Mathematics Division, Oak Ridge National Laboratory, TN, 37831-6367, USA, 1995

[18] G.A.Geist, J.A.Kohl, R.J.Manchek and P.M. Papadoponlos, "PVM and MPI: a Comparison of Features" , Computer Science and Mathematics Division, Oak Ridge National Laboratory, TN, 37831-6367, USA, 1996

[19] Alexandre Alves, Luis Silva, Joao Carrerira and Joao Gabriel Silva, "WPVM: Parallel Computing for the People", Department of Information Engineering, University of de Coimbra, Portugal

[20] Michael Wolfe. **High Performance Compilers for Parallel Computing**, Oregon Graduate Institute of Science & Technology. 1995. pp.307-383

[21] เทพจิต จันทร์ดา, "การพัฒนาการจัดงานของระบบ PVM ในการคำนวณหาหลักฐานยิงของปืนใหญ่ขนาดเบากระสุนวิถีโค้ง", สาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, พ.ศ.2544 หน้าที่ 44-50

[22] Sam-Jin Jeong and Mann-Ho Lee, "REGION PARTITIONING METHOD OF NESTED LOOPS WITH NON-UNIFORM DEPENDENCES", Department of Computer Science, Chonan University, 1999 IEEE TENCON, pp 447-450

ประวัติผู้เขียน

ชื่อผู้เขียน	เธียรชัย พัฒนศิริเวทิน
วัน/เดือน/ปีเกิด	31 พฤษภาคม 2518 ที่จังหวัดแพร่
วุฒิการศึกษา	ระดับปริญญาตรีวิศวกรรมศาสตร์บัณฑิต (วิศวกรรมคอมพิวเตอร์) มหาวิทยาลัยเชียงใหม่ ปีการศึกษา 2540
ประสบการณ์ทำงาน	ปี 2538 ผู้ช่วยฝึกอบรม การใช้งานอินเทอร์เน็ทแด่บุคลากรภายในสถาบัน ซึ่งจัดโดยภาควิชาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเชียงใหม่ ปี 2540 ร่วมงานกับบริษัทโอซีที (O.C.T- Open Computing Technology) จำกัดแผนก Technical Support Engineering