การพัฒนาการจัดตารางของระบบ PVM ในการคำนวณหาค่าฐานยิงของปืนใหญ่สนามเบาแบบลำกล้องยาว

SCHEDULING PLAN FOR PVM SYSTEM IN LIGHT-WEIGHT GUN FIRE-DIRECTIONS CALCULATIONS

เทพฤทิต จันทร์คา

TAPHIT CHANDA

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2544

ISBN 974-648-350-1

การพัฒนาการจัดการของระบบ PVM ในการคำนวณหาหลักฐานยิงของปืนใหญ่ขนาดเบากระสุนวิถีโค้ง

SCHEDULING PLAN FOR PVM SYSTEM IN LIGHT-WEIGHT GUN FIRE-DIRECTIONS CALCULATIONS

เทพจิต จันทร์ดา

TEPJIT CHANDA

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2544

ISBN 974-648-350-1

SCHEDULING PLAN FOR PVM SYSTEM IN LIGHT-WEIGHT GUN FIRE-DIRECTIONS CALCULATIONS

TEPJIT CHANDA

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF COMPUTER SCIENCES AND INFORMATION TECHNOLOGY

SCHOOL OF GRADUATE STUDIES

KING MONGUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2001

ISBN 974-648-350-1

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การพัฒนาการจัดการของระบบ PVM ในการคำนวณหาหลักฐานยิงของปืนใหญ่ขนาดเบากระสุนวิถีโค้ง
SCHEDULING PLAN FOR PVM SYSTEM IN LIGHT-WEIGHT GUN FIRE – DIRECTIONS CALCULATIONS

ชื่อนักศึกษา พันเอก เทพจิต จันทร์ดา

รหัสประจำตัว 37064415

ปริญญา วิทยาศาสตรมหาบัณฑิต

สาขาวิชา วิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ

อาจารย์ผู้ควบคุมวิทยานิพนธ์ รศ.บรรจง ปิยธำรง

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
รศ.บรรจง	ปิยธำรง	
รศ.ดร.วิเชียร	เปรมชัยสวัสดิ์	
ดร.ประจวบ	วานิชชัชวาล	
ดร.อาริต	ธรรมโน	
ดร.โชติพัชร์	ภรณวลัย	

วัน/เดือนปี ที่สอบ 2 พฤษภาคม 2544 เวลา 15.00 น. เป็นต้นไป

สถานที่สอบ ณ ห้อง LAB 316 ชั้น 3 คณะเทคโนโลยีสารสนเทศ

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.บุญวัฒน์ อัตชู)

คณบดีบัณฑิตวิทยาลัย

วันที่ 2 **เดือน** กรกฎาคม **พ.ศ.** 2544

หัวข้อวิทยานิพนธ์	การพัฒนาการจัดการของระบบ PVM ในการคำนวณหาหลักฐานยิงของปืนใหญ่ขนาดเบากระสุนวิถีโค้ง
นักศึกษา	พันเอก เทพจิต จันทร์คา
รหัสประจำตัว	37064415
ปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา	วิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ
พ.ศ.	2544
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.บรรจง ปิยธำรง

บทคัดย่อ

ปัจจุบันกลุ่มของเครื่องคอมพิวเตอร์ลูกข่ายในระบบเครือข่าย ได้กลายมาเป็นทางเลือกสำหรับการประมวลผลแบบขนาน โดยอาศัยโปรแกรมในการประมวลแบบกระจายเช่น PVM ซึ่งทำงานบนระบบเครือข่ายที่มีลูกข่ายเหมือนกันหรือต่างแบบกัน สามารถประมวลผลงานแบบขนานที่มีปริมาณมาก ๆ ได้ อย่างไรก็ตามยังมีปัญหาในการจัดการเพื่อจัดสรรทรัพยากรในการทำงานให้กับงานแบบขนานอยู่ดี โดยปกติผู้ใช้จะต้องเป็นผู้เลือกเครื่องลูกข่ายที่จะใช้ในการประมวลผลงานแบบขนานเอง การทำงานของเครื่องลูกข่ายมีการเปลี่ยนแปลงอยู่ตลอดเวลา ไม่สามารถคาดเดาได้ หากเครื่องลูกข่ายที่เลือกมีการทำงานหนักอยู่แล้ว จะทำให้เกิดปัญหาคอขวด และเสียเวลาในการรอคอยการประมวลผลของเครื่องนั้น เนื้อหาของวิทยานิพนธ์เล่มนี้ ได้นำเสนอถึงการพัฒนาระบบการจัดการของ PVM โดยใช้การคำนวณหาหลักฐานยิงของปืนใหญ่ขนาดเบากระสุนวิถีโค้งในการทดลอง เพื่อวัดประสิทธิภาพการจัดการงานแบบขนาน ซึ่งสามารถปรับปรุงให้เครื่องลูกข่ายได้รับปริมาณงานที่เหมาะสม และลดเวลาในการประมวลผลแบบขนานลง

Thesis Title	Scheduling Plan For PVM System In Light-Weight Gun Fire-Directions Calculations
Student	Colonel Tepjit Chanda
Student ID.	37064415
Degree	Master of Sciences
Programme	Computer Sciences and Information Technology
Year	2001
Thesis Advisor	Assoc.Prof.Banjong Piyathumrong

ABSTRACT

Clusters of powerful workstations have become a feasible alternative for Paralleling. Distributed computing environments such as PVM provide the integration of multiple heterogeneous computing platforms to support execution of multiple Parallel job. However, many of these environments do not properly address the problem of scheduling available resources. Usually, the user must choose the workstations to be used for the execution of a Parallel job. However, in real environments, the load of the system varies in an unpredictable way. When a Task of a Parallel job is allocated to a heavily-loaded machine, it may become a bottleneck for the entire job. The proposes of this thesis is the scheduling planning for PVM system in Light-Weight gun fire-directions calculations to measure performance of the Parallel job try to maintaining a balanced system-wide workload and reducing the average execution time for the job.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้อย่างดี ด้วยคำแนะนำและคำปรึกษาเกี่ยวกับระบบการจัดการโปรแกรมแบบขนานบนระบบ PVM จาก รศ.บรรจง ปียธำรง ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ผู้วิจัยรู้สึกซาบซึ้งในความอนุเคราะห์จากท่าน และขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ พันเอกเจิดศักดิ์ เจริญรุ่งเรือง ผู้อำนวยการกองควบคุมคุณวุฒิ กรมสารบรรณทหารบก ที่ให้คำแนะนำ และสนับสนุนอุปกรณ์ระบบคอมพิวเตอร์ เพื่อการทดลองจำนวนหนึ่ง

ขอขอบพระคุณ พันตรีหญิง กังสดาล ชุณหเพลส ซึ่งเป็นเพื่อนนักศึกษาที่ช่วยเหลือให้คำแนะนำต่าง ๆ พร้อมทั้งให้กำลังใจต่อผู้วิจัยอย่างใกล้ชิดตลอดมา

สุดท้ายขอขอบพระคุณบัณฑิตวิทยาลัย ที่ได้ให้ทุนสนับสนุนการทำวิทยานิพนธ์ครั้งนี้

คุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

เทพจิต จันทร์ดา

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

โปรแกรม PVM (Parallel Virtual Machine) เป็นโปรแกรมที่ใช้สำหรับการพัฒนาโปรแกรมแบบขนาน บนระบบเครือข่ายที่ใช้ระบบคอมพิวเตอร์ชนิดเดียวกัน หรือต่างชนิดกัน หรือที่เรียกว่า Heterogeneous Networks [3] PVM สามารถจัดการกับระบบเครือข่ายดังกล่าว โดยนำขีดความสามารถของหน่วยประมวลผลกลาง (CPU) ของเครื่องคอมพิวเตอร์ในระบบเครือข่าย มาช่วยทำการคำนวณงานที่มีปริมาณมากมายมหาศาล ด้วยการพัฒนาโปรแกรมแบบขนานบนระบบ PVM ทำให้ผลการคำนวณรวดเร็วขึ้น และยังเป็นการใช้ความสามารถของหน่วยประมวลผลกลางทุกเครื่องในระบบเครือข่ายได้อย่างเต็มที่ อย่างไรก็ตามโปรแกรม PVM ก็ยังคงทำหน้าที่ในการแบ่งงาน แปลงข้อมูล และส่งให้กับเครื่องในระบบเครือข่ายที่กำหนดให้เป็นเครื่องในระบบ PVM เท่านั้น นอกจากนี้ยังต้องอาศัยการปรับแต่งระบบด้วยมืออีกด้วย จึงทำให้โปรแกรมแบบขนานบนระบบ PVM ทำงานไม่มีประสิทธิภาพเท่าที่ควร

ระบบการจัดการส่วนใหญ่มุ่งเน้นไปในด้านการค้า และยังพัฒนาให้สามารถใช้งานกับโปรแกรมอื่น ๆ ในลักษณะเดียวกันกับ PVM อีกด้วย จึงทำให้ครอบคลุมขอบเขตที่กว้างเกินไป รวมทั้งไม่สามารถใช้บนระบบ PVM ได้อย่างสมบูรณ์ การที่เลือกศึกษาการจัดการบนระบบ PVM ก็เนื่องจาก PVM เป็นโปรแกรมที่ให้ใช้งานได้โดยไม่มีเงื่อนไข สามารถรับโปรแกรมได้จากระบบอินเทอร์เนต และได้ใช้โปรแกรมการยิงปืนใหญ่จำลอง พัฒนาให้เป็นโปรแกรมแบบขนาน เพื่อใช้ทดสอบโมเดลที่ได้พัฒนาแล้ว และเปรียบเทียบผลจากการทดลอง เพื่อให้สามารถนำโมเดลนี้ไปใช้ประโยชน์ได้ในอนาคต

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

เพื่อพัฒนาโปรแกรมสำหรับจัดการระบบ PVM ให้เป็นต้นแบบสำหรับการพัฒนาต่อไป โดยมุ่งเน้นศึกษาระบบการจัดการบนโปรแกรมแบบขนาน ที่ทำงานบนระบบเครือข่ายแบบ Heterogenous Networks ต่าง ๆ แล้วนำมาพัฒนาโมเดลการจัดการบนระบบ PVM จากนั้นจึงทดสอบด้วยโปรแกรมอำนวยการยิงปืนใหญ่ขนาดเบากระสุนวิถีโค้งแบบขนาน เพื่อทำการวัดผลประสิทธิภาพที่เพิ่มขึ้น

1.3 สมมุติฐานของการศึกษา

การจัดการระบบ PVM มีอยู่ด้วยกันหลายวิธี มีทั้งข้อดีและข้อเสีย ทั้งนี้ขึ้นอยู่กับระบบเครือข่าย และสภาพแวดล้อม สำหรับการศึกษานี้ได้มุ่งเน้นพัฒนาระบบการจัดการของ PVM ได้แก่ Resource Manager ของ PVM เอง เนื่องจากการจัดการของ PVM ยังไม่มีประสิทธิภาพ โดยผู้ใช้ต้องเป็นผู้เลือกเครื่องในระบบเสมือน (Virtual Machine) ซึ่งไม่สามารถทราบได้ว่าเครื่องใดในระบบที่มีการทำงานหนักบ้าง หากสามารถจัดการให้การแบ่งงานของโปรเซส แบ่งให้กับเครื่องที่มีความสามารถ จะทำให้การคำนวณเกิดความรวดเร็วยิ่งขึ้น และสามารถใช้ความสามารถของ CPU ที่ว่างมาช่วยทำการคำนวณได้อย่างเต็มที่

1.4 คำจำกัดความที่สำคัญ

Asynchronous คือการสื่อสารข้อมูลที่ผู้ส่ง และผู้รับไม่จำเป็นต้องดำเนินการในเวลาเดียวกันหรือพร้อมกัน

Barrier Synchronization เป็นเหตุการณ์ที่โปรเซสตั้งแต่สองโปรเซสขึ้นไปอยู่ในกลุ่มที่แน่ชัด และถูกกีดขวางจนกระทั่งสมาชิกทั้งหมดของกลุ่มถูกกีดขวาง เพื่อให้กระทำการใดได้ต่อไป ไม่มีสมาชิกของกลุ่มสามารถจะผ่านการกีดขวางไปได้ จนกว่าโปรเซสทั้งหมดในกลุ่มจะไปถึงมันแล้ว

Broadcast คือการส่งข่าวสารเดียวกันถึงผู้รับที่เป็นไปได้ทั้งหมดในคราวเดียวกัน สามารถส่งซ้ำ ๆ ได้ ซึ่งเป็นวิธีการที่มีประสิทธิภาพมากกว่า ตัวอย่างเช่น เครือข่ายแบบต้นไม้ ซึ่งแต่ละโหนดจะส่งข่าวสารไปยังลูกข่ายของมันในโครงข่ายแบบต้นไม้

Buffer คือที่เก็บชั่วคราวในหน่วยความจำ มีหลายวิธีในการหาเส้นทางการส่งข่าวสารระหว่างหน่วยประมวลผล เช่นใช้ Buffer ที่ต้นทางและปลายทาง หรือที่หน่วยประมวลผลระหว่างทาง

Bus เป็นตัวกลางการสื่อสารข้อมูล ที่ถูกใช้ร่วมกันระหว่างอุปกรณ์ตั้งแต่สองอย่างขึ้นไป

Communication Overhead คือหน่วยวัดปริมาณงานที่เพิ่มขึ้นในอัลกอริทึมแบบขนาน ซึ่งก็คือผลลัพธ์ของการสื่อสารข้อมูลระหว่างโหนดของระบบแบบขนานนั่นเอง

Computation-to-communication Ratio คืออัตราส่วนของจำนวนในการคำนวณของโปรเซสหนึ่งต่อจำนวนข่าวสารทั้งหมดที่มันส่งไป หรืออัตราส่วนของเวลาที่เสียไปในการคำนวณกับเวลาที่เสียไปในการส่งข่าวสาร ซึ่งขึ้นอยู่กับความสัมพันธ์ระหว่างความเร็วของหน่วยประมวลผล และตัวกลางในการสื่อสารข้อมูล

Contention คือการร้องขอทรัพยากรพร้อมกัน และไม่สามารถใช้ร่วมกันได้ ถ้าโปรเซสหลายตัวทำงานบนหน่วยประมวลผลเดียวกันอาจแย่งกันใช้ CPU หรือในเครือข่ายอาจเกิด Contention ได้ถ้าหลาย ๆ ข่าวสารพยายามที่จะใช้เส้นทางเชื่อมต่อเดียวกัน พร้อมกัน

Context Switching คือการเก็บสถานะของโปรเซสหนึ่ง และแทนที่มันด้วยสถานะอื่น ถ้าใช้เวลาน้อย ๆ ในการ Switch Context หน่วยประมวลผลที่ทำงานมากเกินไป ก็จะมีประสิทธิภาพมากขึ้นในระบบ Message-Passing

Daemon คือโปรเซสที่มีความมุ่งหมายพิเศษหนึ่งซึ่งทำงานอยู่เบื้องหลังของระบบเช่น Pvmd โปรเซส หรือ Group Server Task

Deadlock คือ ขนาดการกระทำของโปรเซสหนึ่งในเหตุการณ์การสื่อสารข้อมูล โปรเซสย่อย ๆ ที่ดีอาจมีการคำนวณทางคณิตศาสตร์เพียงเล็กน้อยเท่านั้น สำหรับ Message หนึ่ง และต่อไป ในขณะที่โปรเซสย่อยที่ไม่ดี อาจมีการคำนวณเป็นล้าน ๆ ครั้ง

บทที่ 2
งานวิจัยที่เกี่ยวข้อง

2.1 ความรู้เบื้องต้นของโปรแกรมแบบขนาน (Parallel Programming)

2.1.1 The Parallel System

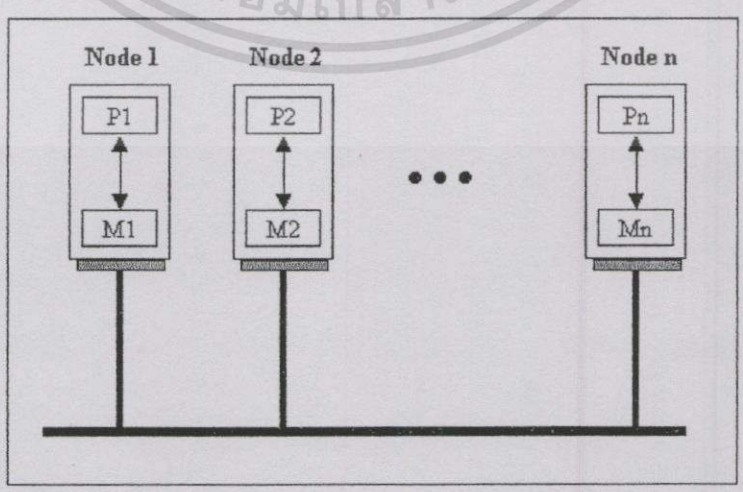
โดยทั่วไปมีสิ่งที่เกี่ยวข้อง และสัมพันธ์กันในระบบขนาน คือ หน่วยความจำกับหน่วยประมวลผลของคอมพิวเตอร์ ได้แก่

- Distributed Memory คือ แต่ละ Processor มีหน่วยความจำของตัวเอง
- Shared Memory คือ หน่วยความจำจะเป็นของส่วนกลาง ซึ่งให้ Processor แต่ละตัวใช้งานได้

สำหรับระบบคอมพิวเตอร์แบบขนานนั้น จะมีรูปแบบในการประมวลผล แต่ละแบบดังนี้

- MIMD (Multiple Instruction, Multiple Data)
- SIMD (Single Instruction, Multiple Data)

ระบบ MIMD นั้นแต่ละ Processor จะทำงานแต่ละคำสั่งที่ถูกแบ่งแล้ว (Task) พร้อม ๆ กัน ส่วน SIMD การทำงานทั้งหมดจะทำที่คำสั่งเดียวกัน พร้อม ๆ กัน ส่วนการ Synchronous นั้นแบบ SIMD จะกระทำโดยอัตโนมัติด้วยเครื่อง สำหรับแบบ MIMD ต้องเขียนโปรแกรมให้ระบบคอมพิวเตอร์แบบ MIMD สามารถจำลองให้เป็นแบบ SIMD ได้ แต่ไม่เหมาะสมเท่าใดนัก ทั้งนี้ยังสามารถจัดหน่วยความจำให้เป็นได้ทั้ง Distributed Memory และ Shared Memory แต่แบบ SIMD สามารถทำได้เฉพาะ Distributed Memory เท่านั้น



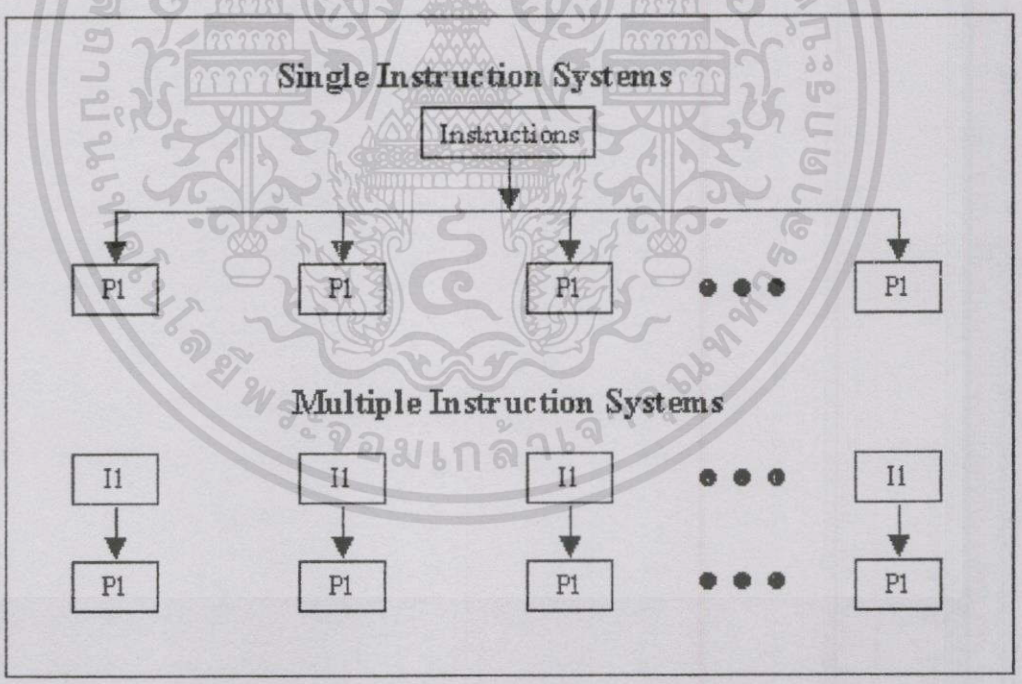
รูปที่ 2.1 การสื่อสารระหว่างเครื่องประมวลผลในเครือข่าย

ในระบบ Distributed Memory MIMD จัดเป็นชุดของ Node ซึ่งแต่ละ Node ประกอบด้วย หน่วยประมวลผลหลัก, หน่วยความจำ และการเชื่อมต่อในเครือข่าย แต่ละ Node ก็คือ เครื่องลูกข่าย แบบ Standalone Node ประมวลผลข้อมูลอย่างเป็นอิสระ และใช้การส่งและรับ Message กับ Node อื่นๆ ผ่านระบบเครือข่าย สถาปัตยกรรมแบบอิสระนี้ เรียกว่า "Loosely-Coupled"

2.1.2 สถาปัตยกรรมแบบ Loosely-Coupled

ในแต่ละ Node ประกอบด้วยคอมพิวเตอร์ที่มีหน่วยประมวลผล, หน่วยความจำ และมีการเชื่อมต่อกับระบบเครือข่าย ซึ่งอาจมีหน่วยคำนวณทางคณิตศาสตร์ถ้าจำเป็น แต่ละ Node จะคำนวณโปรแกรมและข้อมูลของตัวเอง โดยผู้ออกแบบโปรแกรมจะจัดแบ่งปัญหาออกเป็นส่วนย่อย (Decompose) และเขียนโปรแกรมส่งให้แต่ละ Node ทำการคำนวณ เพื่อแก้ปัญหานั้น ซึ่งจะดีกว่าระบบ SIMD คือ

- โปรแกรมเดียวสามารถแบ่งให้ Node ไปประมวลผลในส่วนที่แตกต่างของโปรแกรมได้
- โปรแกรมที่แบ่งย่อยแล้ว สามารถประมวลผลพร้อมกันได้ในแต่ละ Node

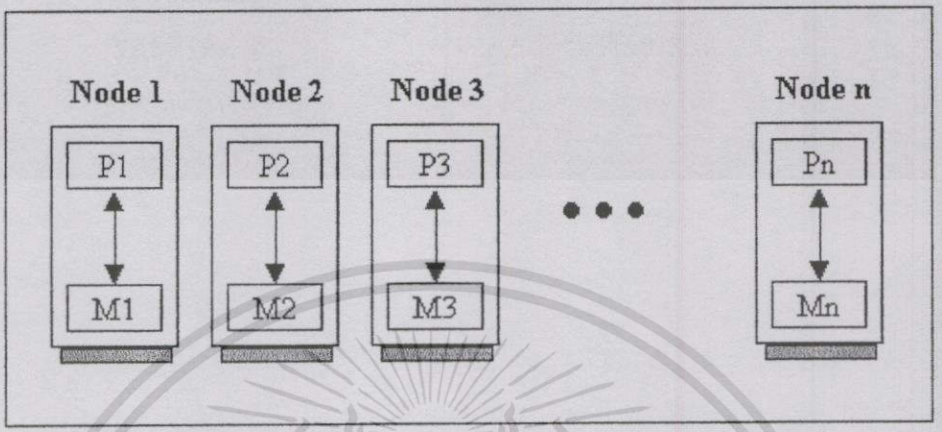


รูปที่ 2.2 ข้อแตกต่างระหว่าง SIMD และ MIMD

2.1.3 รูปแบบของหน่วยความจำ (Memory Schemes)

มีรูปแบบพื้นฐานในการจัดการ สำหรับคอมพิวเตอร์แบบขนาน คือ Distributed และ Shared ใน Distributed-Memory นั้น แต่ละ Node จะมีหน่วยความจำของตัวเอง และหน่วยประมวลผลที่ Node นั้น จะสามารถใช้งานได้ (คือหน่วยความจำของเครื่องนั้น) เมื่อ Node ใด

ต้องการข้อมูลจาก Node อื่น ข้อมูลจะถูกส่งจาก Node ที่มีข้อมูลไปยัง Node นั้น รูปที่ 2.3 แสดงรูปแบบของ Distributed-Memory



รูปที่ 2.3 Distributed-Memory Systems

ส่วนระบบ Shared-Memory นั้น จะมีการจัดหน่วยความจำส่วนกลาง สำหรับให้หน่วยประมวลผลทั้งหมดใช้งานได้ ในระบบคอมพิวเตอร์แบบขนาน มีการใช้งานหน่วยความจำ 2 แบบ คือ ใช้โดยตรง และใช้งานผ่านเครือข่าย (System Bus/Switching Network)

ในระบบ Bus-Based หน่วยประมวลผลทั้งหมด ต้องใช้ System Bus เพื่อเข้าใช้งานหน่วยความจำ ซึ่งจะใช้งานได้ดีและมีประสิทธิภาพมาก สำหรับผู้ใช้หลายคนที่ใช้โปรแกรมส่วนตัวที่ไม่ใหญ่และสับสนมากนัก ขีดจำกัดของระบบนี้อยู่ที่ จำนวนของ Processor หากมีเพิ่มมากขึ้นอาจทำให้เกิดปัญหาคอขวด (Bottleneck)

ระบบ Switching Network ออกแบบมาเพื่อหลีกเลี่ยงปัญหาคอขวด โดยจัดการแบ่งหน่วยความจำออกเป็นหลายส่วนที่ต่างกัน และมีการปรับปรุงและจัดหาพื้นที่หน่วยความจำให้ว่าง ในแบบนี้หน่วยประมวลผลกลางจะเชื่อมต่อกับหน่วยความจำผ่านระดับชั้น (Layers) ของสวิทช์ (โดยปกติเรียกว่า Butterfly Switches) จำนวนของระดับชั้น (Layers) เท่ากับ Logarithm ของจำนวนหน่วยประมวลผลในระบบ

ดังนั้นจึงมีช่องทาง (Bandwidth) ที่กว้างกว่าระบบ Bus-Based แต่ก็มีปัญหากับ "Hot Spots" ซึ่งจะเกิดนี้นเมื่อหน่วยความจำหลาย ๆ ส่วนถูกขอใช้พร้อม ๆ กัน ผ่าน Switch เดียวกัน จึงเป็นสาเหตุสำคัญที่ลดประสิทธิภาพของเครือข่ายโดยรวมลง เพราะว่าตำแหน่งของ Hot Spots นั้นขึ้นอยู่กับโปรแกรมว่าจะใช้หน่วยความจำอย่างไร จึงเป็นการยากสำหรับนักเขียนโปรแกรมที่จะทำนายหรือ หลีกเลี่ยง Hot Spots

ความยุ่งยากอื่น ๆ ของระบบ Switch ก็คือ การใช้หน่วยความจำที่มาก และไม่มีรูปแบบเวลาในการใช้ที่แน่นอน มีการจัดหน่วยความจำขึ้นใหม่เสมอ ดังนั้นแต่ละหน่วยประมวลผลมักจะ

มีหน่วยความจำของตัวเองที่ว่าง เพื่อให้หน่วยประมวลผลอื่นเรียกใช้ผ่าน Switching Network แต่การใช้งานด้วยข้อมูลในหน่วยความจำของตัวเองก็ยังมีประสิทธิภาพมากกว่าอยู่ดี สุดท้ายก็คือ ระบบ Switching Network เพิ่มความยุ่งยากสับสนให้กับระบบ มีผลกระทบในเรื่องราคา ความเชื่อถือได้ของระบบ และจำนวนของหน่วยประมวลผล ที่สามารถจะเพิ่มเข้ามาในระบบได้

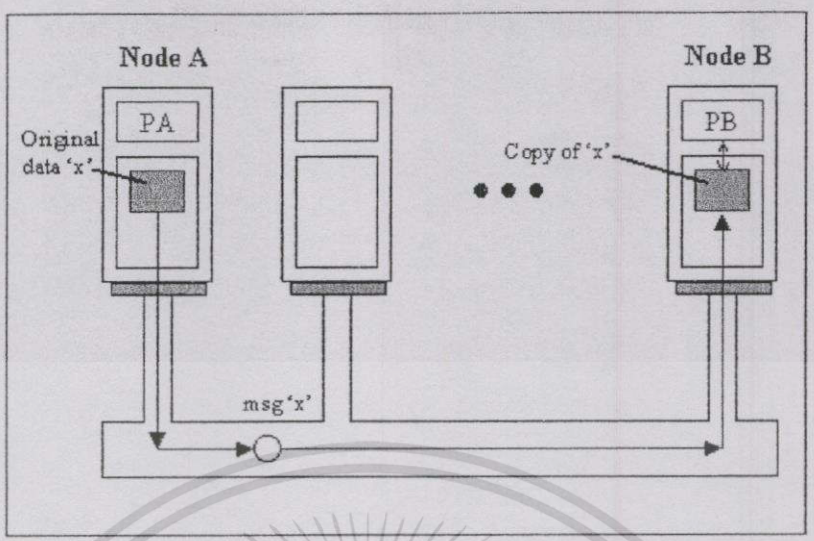
สำหรับโปรแกรมประยุกต์ด้านวิทยาศาสตร์ส่วนใหญ่ รูปแบบ Distributed-Memory จะให้ประสิทธิภาพสูงกว่า Shared-Memory การจัดรูปแบบที่เหมาะสม และใช้หน่วยความจำที่มี Access Time เร็วมากจะสามารถหลีกเลี่ยงปัญหาการแย่งใช้หน่วยความจำผ่านระบบ Bus ได้ อย่างไรก็ตาม Distributed-Memory ถูกออกแบบมาให้สามารถใช้กับหน่วยประมวลผลกลางจำนวนมากเป็นร้อยหรือพัน ที่มีแบบที่แตกต่างกันได้ แต่ก็ทำให้ทรัพยากรของส่วนกลางลดลง ซึ่งอาจเกิดปัญหาคอขวดได้ หรือทำให้ระบบมีความยุ่งยากสับสนมากขึ้น

ตัวแปรต่าง ๆ เหล่านี้ เป็นค่าที่จะเพิ่มประสิทธิภาพของ Loosely-Coupled สำหรับโปรแกรมประยุกต์ด้านวิทยาศาสตร์ที่มีขนาดใหญ่มากได้ อย่างไรก็ตามแต่ละ Node จะต้องสามารถสื่อสารกันได้ และยอมให้แต่ละหน่วยประมวลผลกลางใช้งานหน่วยความจำของหน่วยประมวลผลกลางอื่นได้ ซึ่งอาจเสี่ยงต่อการส่งข้อมูลผิดพลาด และล้มเหลวได้เช่นกัน

2.1.4 การสื่อสารข้อมูล (Communication)

อธิบายได้อย่างง่าย ๆ ก็คือ แต่ละหน่วยประมวลผลกลางมีหน่วยความจำเป็นของตัวเอง และต้องการการส่งข้อมูลที่เหมาะสมสำหรับการคำนวณ ถ้าข้อมูลเก็บอยู่ที่ Node ใด และ Node อื่นต้องการ Node นั้นจะต้องสามารถส่งให้ Node ที่ต้องการได้ และ Node ที่ต้องการข้อมูลก็ต้องได้รับมันด้วย

เครือข่าย Message-Passing สนับสนุนการส่งข้อมูลระหว่าง Node โดย Message จะสามารถนำข้อมูลไปให้แต่ละ Node ได้ Message ประกอบด้วยข้อมูล และรหัสโปรแกรมที่ถูกคัดลอกจาก Node ที่ส่ง และส่งผ่านเครือข่ายแบบ Message-Passing แสดงดังรูปที่ 2.4



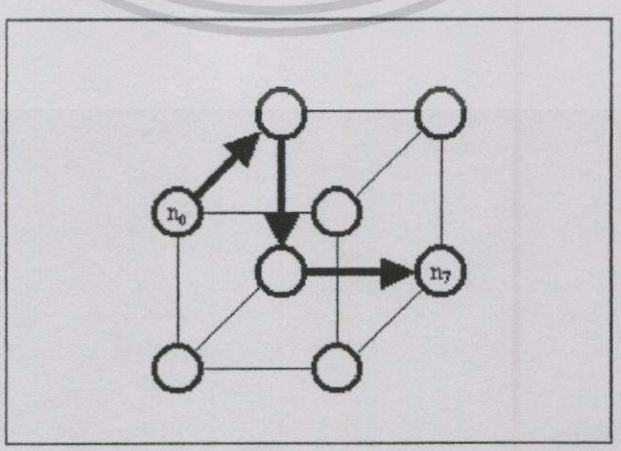
รูปที่ 2.4 การส่งผ่านเครือข่ายแบบ Message-Passing

โดยทั่วไปเวลาในการส่ง Message ขึ้นอยู่กับความเร็วของเครื่อง และระบบเครือข่าย ระบบที่สมดุลย์ (Balanced) ในเครือข่ายจะเพิ่มประสิทธิภาพในการคำนวณให้มากขึ้น

2.1.5 การเชื่อมต่อระหว่าง Node (Node Interconnection)

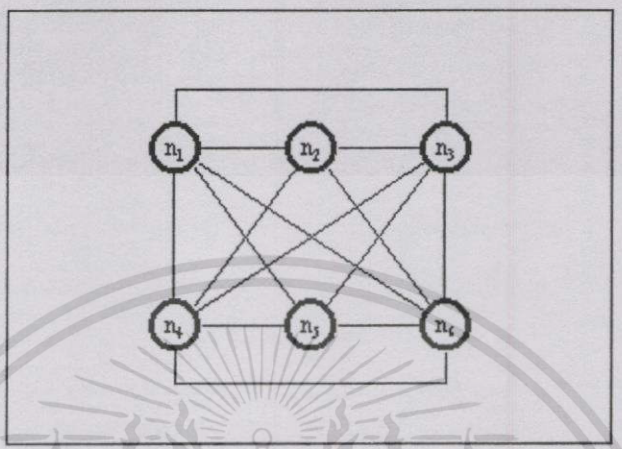
การเชื่อมต่อในระบบ Loosely-Coupled มีปัญหาบางประการ โดยทั่วไปสามารถเชื่อมต่อระบบเครือข่ายได้หลายแบบ เช่น Linear Arrays, Meshes, Tori, Hypercubes, Ring, Completely Connected Crossbars เป็นต้น

การเชื่อมต่อที่เป็นพื้นฐานที่สุดได้แก่ Linear Arrays ปัญหาของ Linear Arrays ก็คือมีช่องทางการส่ง Message จำกัด (Limit Message Bandwidth) และระยะทางในการส่ง Message ทำให้ประสิทธิภาพลดลง



รูปที่ 2.5 เครือข่าย Message-Passing แบบ Hypercube

การเชื่อมต่อแบบอื่น จะมีความอ่อนตัวมากกว่า เช่น Hypercube หรือ Mesh ซึ่งทำให้มีช่องทางในการเชื่อมต่อมากขึ้น และลดระยะในการส่ง Message ลง ตามรูปที่ 2.5 เป็นแบบ Hypercube



รูปที่ 2.6 เครือข่ายการเชื่อมต่อแบบเต็มรูปแบบ

เนื่องจากไม่สามารถทำนายเวลาของโปรแกรมประยุกต์ทั้งหมดได้ ดังนั้นการเชื่อมต่อที่ดีที่สุดคือ มีการเชื่อมต่อโดยตรงไปยัง Node ต่าง ๆ ได้ ซึ่งจะทำให้มีช่องทางการส่ง Message มากที่สุด และมีระยะทางในการส่งน้อยที่สุด การเชื่อมต่อแบบนี้ แสดงได้ตามรูป 2.6

แต่อย่างไรก็ตามหากมีการเชื่อมต่อหลาย Node มาก เช่น 1,000 Node ก็จะต้องสร้างเส้นทางเชื่อมต่อเป็นล้านเส้นทางระหว่าง Node ซึ่งเป็นไปได้ยากมาก

2.1.6 รูปแบบการเขียนโปรแกรมสำหรับ Loosely-Coupled

การพัฒนาโปรแกรมสำหรับ Distributed Memory Parallel Computer มีหลักที่คำนึงถึง 3 ประการ

- การคำนวณในแต่ละชุดของโปรเซสแบบขนาน
- ข้อมูลที่จะถูกใช้โดยโปรเซสหนึ่ง
- การสื่อสารโดย Passing Message

2.1.7 การคำนวณด้วยโปรเซสแบบขนานและข้อมูลส่วนตัว

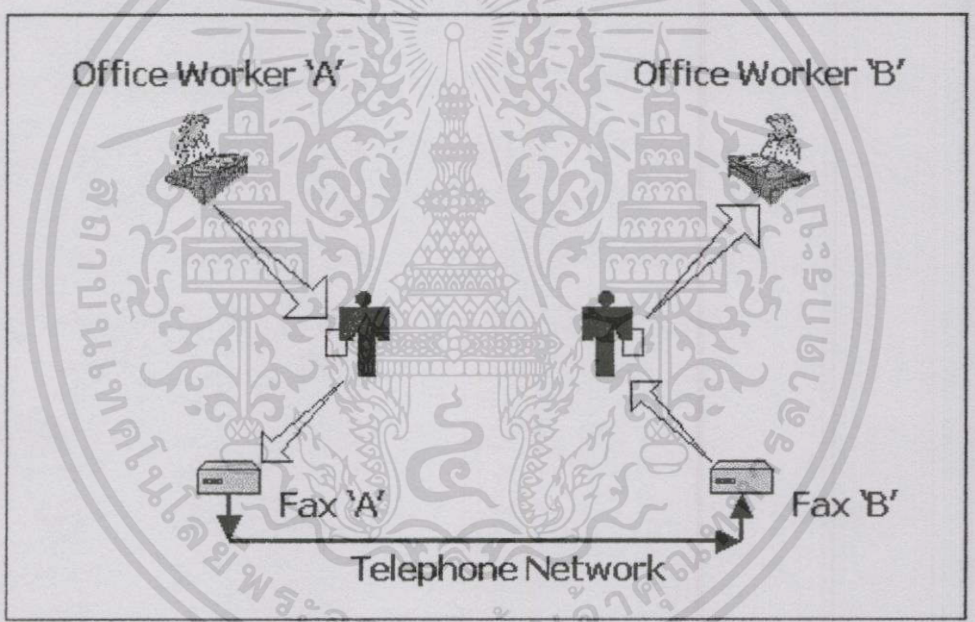
โปรเซสหนึ่งในระบบ Loosely-Coupled ก็คือ ส่วนที่เป็นแบบลำดับ (Sequential Part) ส่วนหนึ่งของโปรแกรมแบบขนาน ส่วนที่เป็นลำดับนี้จะทำงานบน Node หนึ่ง การทำให้เกิดประสิทธิภาพสูงสุด ก็คือการออกแบบโปรแกรมประยุกต์ให้ชุดของโปรเซสทำงานได้พร้อม ๆ กัน โดยไม่จำกัดที่จำนวนของหน่วยประมวลผล (Processor) หรือจำนวนของ Processor ที่มีความสามารถในการคำนวณ

โปรเซสทำงานในส่วนที่ต่างกัน และแลกเปลี่ยนข้อมูลที่ต้องการระหว่างกัน โปรเซสเหล่านี้ สามารถเขียนด้วยภาษาโปรแกรมที่เหมาะสมได้ โปรเซสจะทำงานอย่างอิสระบน Node ที่ต่างกัน และส่งผลลัพธ์ หรือแลกเปลี่ยนรหัสโปรแกรม และข้อมูลระหว่างโปรเซส

2.1.8 การสื่อสารข้อมูล (Communicating)

การสื่อสารระหว่างโปรเซส มีพื้นฐานง่าย ๆ เมื่อโปรเซสหนึ่งต้องการข้อมูลจากโปรเซสอื่นโปรเซสนั้นจะต้องส่งสำเนาของข้อมูลที่โปรเซสต้องการด้วย Message ให้ได้อย่างง่าย ๆ

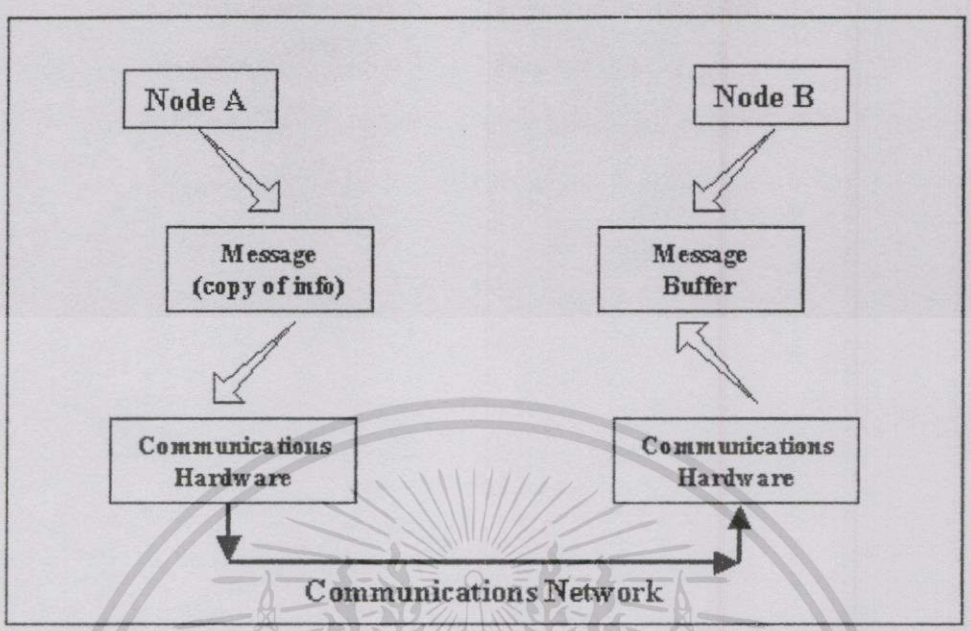
เปรียบเทียบได้กับการรับ - ส่ง แฟกซ์เอกสาร โดยทั่วไปผู้รับและผู้ส่งจะรู้ว่าเป็นเอกสารอะไร และผู้ส่งก็จะกำหนดเลขหมายของปลายทาง, ชื่อของผู้รับและชื่อเรื่องของเอกสาร เมื่อเชื่อมต่อกันได้ก็จะส่ง และรับข้อมูลตามรูป 2.7



รูปที่ 2.7 การรับ – ส่ง แฟกซ์เอกสาร

การส่ง Message ระหว่างโปรเซสก็มีขั้นตอนเช่นเดียวกัน โปรเซสหนึ่งจะส่ง Message ไปยังโปรเซสอื่น จะใช้ Message-Passing Routine ในการส่ง ซึ่งจะกำหนดชนิดของ Message กำหนดที่ตั้ง และความยาวของข้อมูลที่จะส่ง และสถานที่ที่จะส่ง การทำงานก็คือ คัดลอกข้อมูล (Document), ระบุ Node (The Destination) และโปรเซสที่จะรับ, ประเภทของ Message (The Subject) การสื่อสารติดต่อก็จะถูกจัดตั้งขึ้นเพื่อการส่ง Message

โปรเซสที่ต้องการข้อมูลจะต้องได้รับ Message ที่ต้องการ ดังนั้นการส่งจะต้องประกอบด้วยขั้นตอนการรับคือ ชนิดของ Message, ที่ซึ่งข้อมูลนั้นเก็บอยู่ และเมื่อใดที่ Message มาถึงโปรเซสสามารถรับได้ทันที



รูปที่ 2.8 การส่ง Message จากโปรเซสหนึ่งไปยังโปรเซสอื่น

ข้อควรจำที่สำคัญคือ Message ประกอบด้วย รหัส และข้อมูล คือ สำเนาของข่าวสารที่เก็บอยู่ในหน่วยความจำของ Node ที่ส่ง ไม่มีการอ้างถึงหรือตัวชี้ถึงตัวแปรต่าง ๆ ซึ่งแต่ละโปรเซสมีรหัสและโครงสร้างข้อมูลของตัวเอง และโปรเซสที่รับก็ไม่สามารถเข้าถึงหน่วยความจำของโปรเซสที่ส่งได้

2.1.9 แบบของ Message

แบบ ก็คือค่าของเลขจำนวนเต็ม (Integer) ที่แตกต่างกันในแต่ละ Message แต่ละ Message ถูกกำหนดแบบในขั้นตอนการส่ง และแบบเดียวกันนี้ต้องถูกกำหนดในขั้นตอนการรับเช่นเดียวกัน Message ที่ถูกต้องเท่านั้นที่จะรับไว้

แบบของ Message เป็นเลขจำนวนเต็มง่าย ๆ ไม่สับสน ซึ่งสัมพันธ์กับชนิดของข้อมูลในโปรแกรมคอมพิวเตอร์เช่น Integer, Real หรือ Complex โปรแกรมเมอร์จะเป็นคนกำหนดค่าของแบบ (Type) และใช้เป็นตัวระบุแบบของ Message ที่เชื่อถือได้ในโปรแกรม

การกำหนดแบบของ Message ทำให้สามารถเขียนโปรแกรมการรับโดยระบุ Message ที่เหมาะสมได้ ทำให้โปรเซสได้รับ Message เพื่อไปทำงานจนกระทั่งจบงานได้

2.1.10 การควบคุม Message (Message Handling)

บ่อยครั้งที่คนส่งเอกสารก่อนที่ผู้รับพร้อมจะอ่านมัน หรือบางทีก็ต้องการเอกสารก่อนที่มันจะมาถึง เช่นเดียวกันกับการส่งผ่าน Message (Passing Message) ระหว่างโปรเซสในโปรแกรมแบบขนาน

2.1.11 การควบคุมให้ส่งให้ถูกจังหวะตรงกัน (Synchronous Message Handling)

การหยุดการทำงานเพื่อรอ Message ที่จะส่งหรือรับเราเรียกว่า "Synchronous Message Handling" หรือ "Blocking" ในการส่งและรับ เมื่อโปรเซสหนึ่งต้องการ Message โปรเซสจะไม่ทำงานคำสั่ง (ถูก block แล้ว) จนกระทั่ง Message ตามแบบที่ระบุมาถึงแล้ว ถ้า Message ยังมาไม่ถึงในเวลาที่ต้องการ โปรเซสก็จะไม่ดำเนินการอะไรต่อไปจนกว่า Message จะมาถึง แต่ถ้า Message มาถึง Node ก่อนที่โปรเซสต้องการ โปรเซสก็จะคอยจนกระทั่ง Message ถูกคัดลอกจากระบบจัดเก็บ (System Buffer) ไปยังหน่วยความจำของโปรเซสซึ่งก็คือการพักรอชั่วคราวนั่นเอง

ในทำนองเดียวกันเมื่อโปรเซสหนึ่งต้องการที่จะส่ง Message แบบ Synchronously โปรเซสจะถูก blocked จนกระทั่ง Message ถูกระบบปฏิบัติการคัดลอกจากหน่วยความจำของ โปรเซสที่ส่งเข้าไปในระบบการส่งผ่าน Message ของเครือข่าย ซึ่งไม่ได้หมายความว่า Message นั้นไปถึงที่หมายแล้ว แต่หมายความว่า Message ที่จะส่งได้ถูกจัดเก็บแล้วในหน่วยความจำที่เก็บ Message ส่ง และตอนนี้ก็สามารถเปลี่ยนแปลงได้ถ้าต้องการ

Synchronous Message Passing ถือได้ว่ามีความจำเป็นต่อการพัฒนาอัลกอริทึมที่ต้องการความถูกต้อง และใช้ประโยชน์สำหรับการพัฒนาในช่วงแรก และการตรวจแก้โปรแกรมประยุกต์ แต่ในที่สุดแล้วการใช้ Asynchronous Message Passing จะทำให้มีความสะดวก และอิสระระหว่างโปรเซสมากกว่า

2.1.12 การควบคุมการส่งแบบไม่เป็นจังหวะ (Asynchronous Message Handling)

โดยปกติในขณะที่คอยเอกสาร เราสามารถทำงานอื่นๆ ได้ และสามารถเริ่มงานกับเอกสารนั้นเมื่อมันมาถึงและเราพร้อมจะรับมันได้ ลักษณะแบบนี้เหมือนกับการ Asynchronous Message Handling ซึ่งมีประสิทธิภาพมากกว่า การ Asynchronous ในการรับยอมให้โปรเซสแจ้งเตือนระบบปฏิบัติการถึง Message ที่ต้องการ และควรจะมีการนำส่งให้โปรเซสทันทีที่มาถึง มันจะทำงานต่อไปจนกระทั่งถึงคำสั่งที่ต้องการข้อมูลใน Message ที่ต้องการ แล้วโปรเซสก็จะตรวจสอบที่เครื่องของมันว่ามี Message มาถึงหรือยัง ถ้ามีก็สามารถใช้ได้ทันที ถ้ายังโปรเซสก็ต้องคอย

โปรเซสหนึ่งสามารถส่ง Message แบบไม่เป็นจังหวะได้ โดยแจ้งเตือนระบบปฏิบัติการว่าต้องการส่ง Message แต่โปรเซสจะไม่คอยจนกระทั่ง Message ถูกส่ง มันสามารถทำงานตามคำสั่งอื่นต่อไป แล้วจึงทำงานคำสั่งในการส่งต่อไป ทำให้เกิดความสะดวก ถ้า Message หนึ่งกำลังถูกส่ง บาง Message ไม่สามารถถูกส่งได้ทันที จึงเหมือนกับการเข้าคิวส่ง

2.1.13 การขัดจังหวะการส่ง Message (Interrupt Message Handling)

การควบคุมการส่งในแบบอื่น ๆ ที่นิยมเรียกว่า "Interrupt Handling" มีลักษณะคล้ายกับ Asynchronous Message Handling คือโปรเซส สามารถทำงานต่อไปได้ในขณะคอย Message ต่างกันที่ Interrupt Handling นั้นโปรเซสที่รับจะถูกขัดจังหวะทันทีที่ Message มาถึง ซึ่งสามารถนำไป

คำนวณได้ทันที เมื่อโปรเซสพร้อมจะรับมันจะประกาศบอกว่าพร้อมรับ หลังจากนั้น โปรเซสสามารถย้อนกลับมาทำงานนั้นได้เมื่อเกิดการขัดจังหวะ

2.1.14 Message ที่สูญเสีย (Lost Message)

แบบที่ 4 ในการควบคุมเอกสารที่เข้ามาง่าย ๆ คือ การยกเลิกมัน เพราะว่ามันไม่ต้องการในการเขียนโปรแกรมแบบขนาน Message ที่ถูกส่งไปยัง Node ที่ไม่ถูกต้อง Node นั้นจะไม่สามารถทำอะไรกับมันได้ เพราะมันไม่รู้ว่าจะมีโปรเซส อื่นต้องการ Message นั้นหรือไม่ ดังนั้น Message ที่ค้างอยู่ในที่พักข้อมูลของระบบ (System Message Buffer) เรียกว่า "Lost" กรณีที่เกิด Lost Message นั้นเกิดจากผลของโปรแกรมผิดพลาด และยากในการตรวจหา Message จากที่พักข้อมูลของระบบ โดยไม่ใช้เครื่องมือในการตรวจแก้โปรแกรม

2.1.15 ขีดความสามารถ (Performance)

โดยทั่วไปการวัดประสิทธิภาพของระบบแบบขนาน วัดได้ใน 2 ลักษณะคือ ความเร็วที่เพิ่มขึ้น (Speed-Up) และประสิทธิภาพ (Efficiency)

ความเร็วที่เพิ่มขึ้นวัดได้จากสูตร

$$Sp = T1/Tp$$

Sp คือ ความเร็วที่เพิ่มขึ้นบน P Node , T1 คือเวลาที่ดีที่สุดในอัลกอริทึมการแก้ปัญหาแบบลำดับ และ Tp คือเวลาที่ต้องการสำหรับอัลกอริทึมการแก้ปัญหาแบบขนานบน P Node

Linear Speed-Up = เวลาที่ต้องการในการแก้ปัญหาบนหนึ่ง Node จำนวน P ครั้ง/เวลาที่ต้องการในการแก้ปัญหาบน P Node

ประสิทธิภาพ ก็คือ ความเร็วที่เพิ่มขึ้นต่อ Node

$$Ep = Sp/P$$

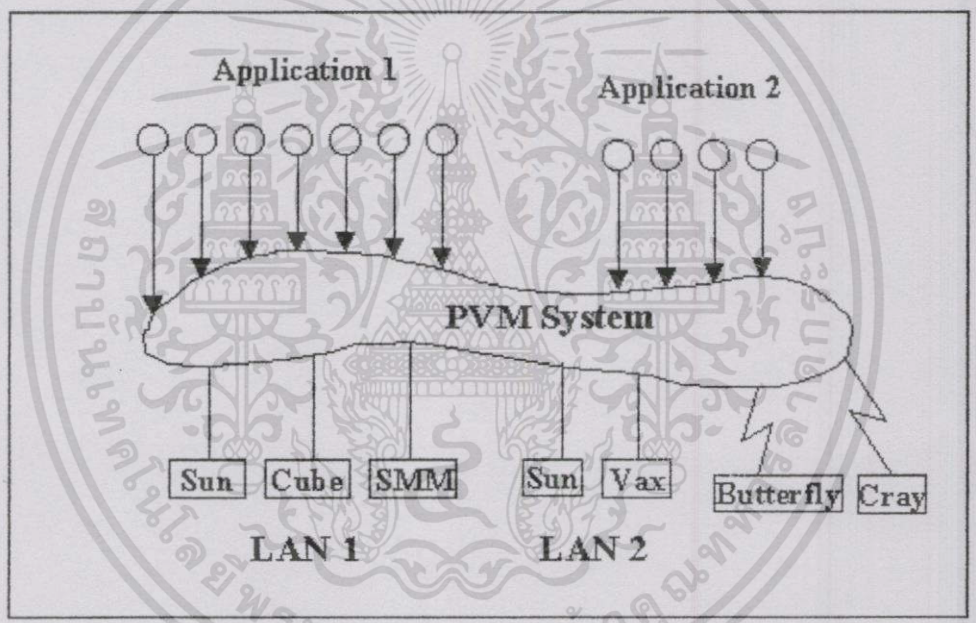
ขีดความสามารถที่ได้เปรียบของคอมพิวเตอร์แบบขนาน กับแบบลำดับ ขึ้นอยู่กับชนิดของปัญหา และวิธีการแบ่งปัญหาออกเป็นส่วนย่อย ซึ่งถ้าไม่ระวังจะทำให้เกิด OverHead และทำให้ขีดความสามารถของโปรแกรมแบบขนานลดลง

2.2 โปรแกรม PVM (Parallel Virtual Machine)

2.2.1 กล่าวทั่วไป

เป็นโปรแกรมที่สามารถทำงานในระบบเครือข่ายที่มีลักษณะหลากหลาย (Heterogeneous Networks) ซึ่งเป็นผลงานวิจัยจาก Oak Ridge National Laboratory and the University of Tennessee PVM เป็นโปรแกรมที่สามารถทำงานได้ทั้งบนเครื่องคอมพิวเตอร์แบบลำดับ (Serial) และแบบขนาน (Parallel) PVM เป็นที่นิยมมาก และเป็นมาตรฐานสำหรับการประมวลผลแบบ

กระจาย (Distributed Computing) [4] สามารถพัฒนาโปรแกรมแบบขนานขึ้นมาทำงานบนเครือข่ายที่มีโปรแกรม PVM ติดตั้งและทำงานอยู่ได้ ด้วยโปรแกรมภาษาซี และภาษาฟอร์แทรน สามารถทำการคำนวณแบบขนาน โดยโปรแกรม PVM จะแบ่งงานออกเป็นงานย่อย ๆ และจัดแบ่งให้กับเครื่องคอมพิวเตอร์ที่อยู่ในระบบเครือข่าย ทั้งที่เป็นคอมพิวเตอร์แบบลำดับ และแบบขนาน โดยวิธีการรับส่งข้อมูลแบบ Message-Passing ระหว่างเครื่องคอมพิวเตอร์ในระบบเครือข่าย เพื่อให้ช่วยกันทำการคำนวณงานที่แบ่งมอบให้ ดังนั้นการคำนวณที่มีปริมาณมาก ๆ และสามารถแบ่งงานออกเป็นงานย่อย ๆ ได้ จะสามารถแบ่งให้ทำงานไปพร้อมกันในแต่ละส่วนได้ โดยไม่ต้องรอกันแบบลำดับ ผู้วิจัยจึงได้เลือกใช้ PVM ในการพัฒนาโปรแกรมแบบขนาน ซึ่งมีพื้นฐานในการทำงานดังนี้ [6]



รูปที่ 2.9 รูปแบบการเชื่อมโยงเครือข่ายในระบบ PVM

2.2.1.1 ผู้ใช้สามารถกำหนดจำนวนเครื่องในระบบเครื่องแบบเสมือน (Virtual Machine) โดยสามารถเพิ่มหรือลบด้วยการสั่งจากบรรทัดคำสั่งได้ตลอดเวลา ว่าต้องการให้เครื่องใดบ้างช่วยทำการคำนวณ ด้วยการสั่งให้ Pvmd ซึ่งเป็นโปรเซสที่ทำงานอยู่เบื้องหลังของระบบปฏิบัติการของ PVM ให้ทำงานอยู่เบื้องหลังของเครื่องที่ต้องการเพิ่มเข้ามาในระบบเครื่องแบบเสมือน

2.2.1.2 โปรแกรม PVM จะทำการเลือกเครื่องที่มีขีดความสามารถ และส่งงานย่อยที่แบ่งแล้วให้แต่ละเครื่องทำงานอย่างเหมาะสมกับความสามารถของ CPU ของเครื่องนั้น

2.2.1.3 หน่วยพื้นฐานในการคำนวณแบบขนานคือ Task

2.2.1.4 โมเดลแบบ Message-Passing จะทำหน้าที่รวบรวมผลการคำนวณของ Task ในแต่ละส่วนโดยการส่ง และรับ Message จากเครื่องหนึ่งไปสู่อีกเครื่องหนึ่ง ซึ่งขนาดของ Message จะขึ้นอยู่กับขนาดของหน่วยความจำของเครื่องรับที่มีอยู่ในขณะนั้น

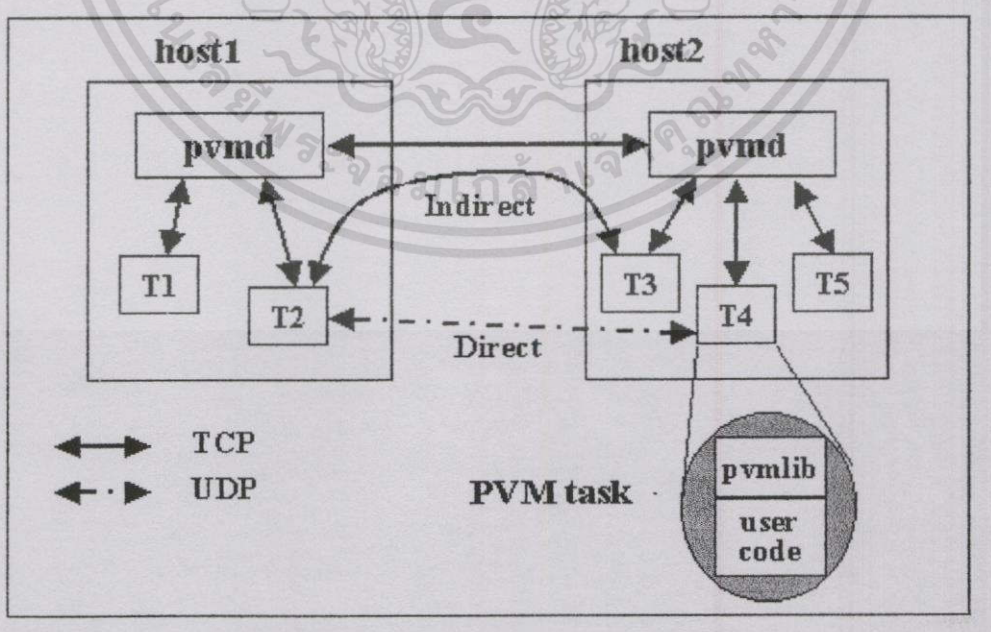
2.2.1.5 ระบบ PVM สามารถรับรู้แบบของข้อมูลต่าง ๆ ได้มากกว่า 1 แบบ ใน Message-Passing [7] เพื่อทำการเปลี่ยนให้เหมาะกับเครื่องในแบบต่าง ๆ

2.2.2 ระบบ PVM ประกอบด้วย 2 ส่วน ที่สำคัญ คือ

2.2.2.1 Daemon เรียกว่า Pvmd3 หรือเรียกสั้น ๆ ว่า Pvmd จะทำงานอยู่ในเบื้องหลังของระบบปฏิบัติการในระบบเครื่องแบบเสมือน ซึ่งการทำงานนี้สามารถจะสั่งให้ทำงานได้จากบรรทัดคำสั่งของระบบปฏิบัติการเช่น ยูนิกซ์ หรือ ลินุกซ์ ที่เครื่องใดก็ได้ที่พ่วงต่ออยู่ในระบบเครื่องแบบเสมือน

2.2.2.2 Library ของ PVM ซึ่งจะประกอบด้วยฟังก์ชันในการทำงานต่าง ๆ เช่น Message-Passing, Spawning Process, Coordinating Tasks และการปรับปรุงระบบเครื่องแบบเสมือน

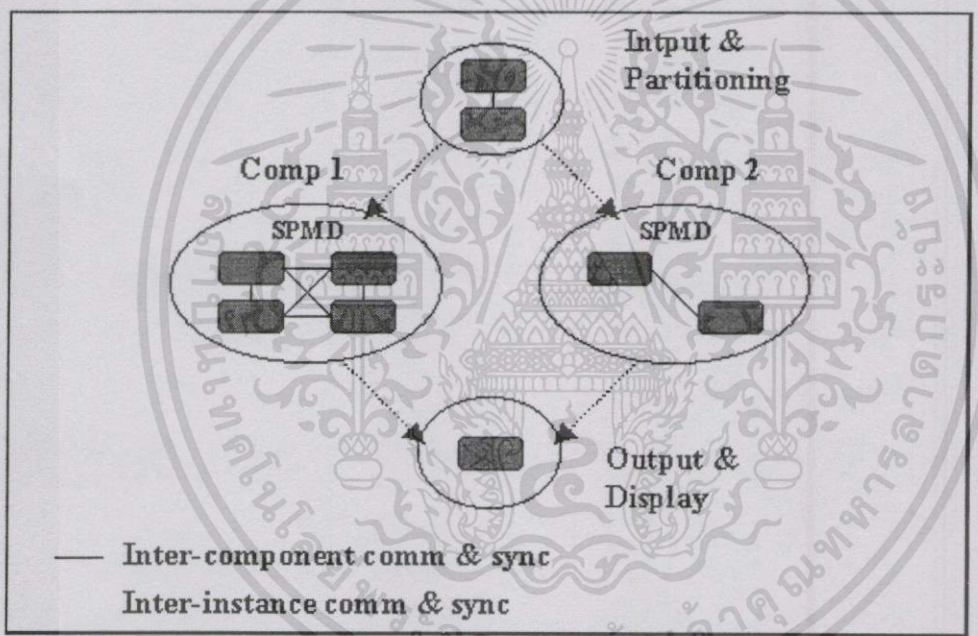
PVM มีรูปแบบในการจัดการเส้นทางการส่ง Message ของโปรแกรมประยุกต์ 2 แบบ คือ Indirect และ Direct Routing มีค่าโดยปริยายคือ Indirect Routing โดย Indirect คือการติดต่อระหว่าง Pvmd-Pvmd โดยใช้ UDP Socket และ Direct คือ การติดต่อระหว่าง Task-to-Pvmd โดยใช้ TCP Socket [5]



รูปที่ 2.10 เส้นทางการส่ง Message ของ PVM ทั้งสองแบบคือ Indirect และ Direct Routing

2.2.3 ส่วนประกอบของระบบ PVM

รูปแบบการคำนวณของ PVM เชื่อว่า โปรแกรมประยุกต์ประกอบด้วยงานหลายงาน (Task) แต่ละงานเป็นส่วนหนึ่งในการคำนวณของโปรแกรมประยุกต์ บางทีแต่ละงานก็กระทำต่อฟังก์ชันที่แตกต่างกันไปพร้อม ๆ กันได้ เช่น Input, Problem Setup, Solution, Output และ Display บางทีเรียกโปรเซสเหล่านี้ว่า "Functional Parallelism" แต่โดยทั่วไปยังมีอีกวิธีหนึ่งเรียกว่า "Data Parallelism" ซึ่งมีงานทั้งหมดเหมือนกัน แต่แตกต่างกันที่ข้อมูล มีลักษณะเช่นเดียวกับ SPMD (Single-Program Mulitiple-Data) PVM มีการทำงานที่สนับสนุนได้ทั้งสองแบบดังกล่าว ขึ้นอยู่กับฟังก์ชันและงานที่ทำงานแบบขนานและอาจมีการติดต่อกันหรือแลกเปลี่ยนข้อมูลซึ่งกันและกัน รูปที่ 2.11 เป็นโมเดลการคำนวณของ PVM



รูปที่ 2.11 โมเดลการคำนวณของ PVM

ปัจจุบัน ระบบ PVM สนับสนุนภาษาโปรแกรม คือ C, C++ และ FORTRAN โดยทั่วไปสามารถใช้โปรแกรมภาษา C ที่มีอยู่แล้วในระบบปฏิบัติการ UNIX ซึ่งสามารถคอมไพล์โปรแกรม PVM บนระบบยูนิกส์ได้ ซึ่งในวิทยานิพนธ์เล่มนี้ได้ติดตั้งและคอมไพล์โปรแกรม PVM บนระบบปฏิบัติการลีนุกส์

งานทั้งหมดของ PVM ถูกกำหนดเป็นเลขจำนวนเต็มเรียกว่า Task Identifier (TID) ดังนั้น Message จึงส่งและรับจากระบบ TID ซึ่ง TID จะต้องไม่ซ้ำกันในระบบ Virtual Machine เดียวกัน ซึ่งจะถูกควบคุมจากโปรเซสของ PVM เรียกว่า Pvmd จากเครื่องหลักโดยผู้ใช้ไม่ได้เป็นผู้กำหนด TID

โปรแกรมประยุกต์โดยทั่วไปสามารถจัดกลุ่มของงานได้ ซึ่งผู้ใช้มีความต้องการจัดกลุ่มของงาน เช่นโปรแกรมประยุกต์แบ่งงาน 0 - (p - 1) เป็นกลุ่มงานเดียวกันกลุ่มหนึ่ง โดย P คือจำนวนงาน PVM มีรูปแบบสนับสนุนกลุ่มงาน โดยผู้ใช้สามารถตั้งชื่อกลุ่ม และจัดงานเข้าไปอยู่ในกลุ่ม โดยกลุ่มจะกำหนดหมายเลขของงานที่ไม่ซ้ำกันในกลุ่ม ให้กับงานที่เข้ามาในกลุ่มงาน สามารถเข้ากลุ่มและออกจากกลุ่มเมื่อใดก็ได้ โดยที่จะไม่มีผลกระทบกับงานอื่น ๆ ในกลุ่ม กลุ่มงานสามารถเหลื่อมกันได้ โดยที่งานสามารถกระจาย Message ไปยังกลุ่มที่มันไม่ได้เป็นสมาชิกได้

โปรแกรมประยุกต์โดยทั่วไป ผู้ใช้สามารถเขียนด้วยภาษา C, C++ หรือ Fortran 77 และเรียกใช้ PVM library ได้ การจัดแบ่งงานขึ้นอยู่กับโปรแกรมประยุกต์ การทำงานของโปรแกรมจะเริ่มจากการคัดลอกงานหนึ่ง (โดยปกติคือ "Master" ที่เตรียมงาน) จากเครื่องๆ หนึ่ง ส่งไปยังเครื่องในระบบ PVM แต่ละเครื่อง โปรเซสจะไปสั่งให้งานในเครื่องอื่นเริ่มทำงาน โดยส่ง Message ให้แต่ละงานไปคำนวณ และรับผลการคำนวณมารวบรวมและแสดงผลในที่สุด

ตัวอย่างในรูป 2.12 คือ โปรแกรม hello เป็นพื้นฐานการเขียนโปรแกรมบนระบบ PVM โดยเริ่มต้นจะพิมพ์หมายเลขของงาน ซึ่งได้รับหมายเลขนี้จากฟังก์ชัน pvm_mytid () จากนั้นจะสั่งงานโปรแกรมอื่นเริ่มทำงานในที่นี้คือ hello_other ซึ่งมีอยู่ทั้งในเครื่องหลักและเครื่องอื่น ๆ ในระบบ PVM โดยใช้ฟังก์ชัน pvm_spawn () ถ้าสำเร็จมันจะเตรียมรับ Message (Blocking Receive) โดยฟังก์ชัน pvm_recv () เมื่อได้รับ Message แล้วจะพิมพ์ Message ที่ได้รับพร้อมกับ TID ของงานนั้น การคัดลอก Message จาก Bufter ใช้ฟังก์ชัน pvm_upkstr () สุดท้ายจบโปรแกรมด้วย pvm-exit

```
#include "pvm3.h"

main()
{
int cc, tid, msgtag;
char buf[100];

printf("i'm t%x\n", pvm_mytid();

cc = pvm_spawn("hello_other", (char**)0, 0, "", 1,  &tid);

if (cc = = 1) {
        msgtag = 1;
pvm_recv(tid, msgtag);
```

```
pvm_upkstr ( buf ) ;
printf ( "from t%x:  %s\n", tid, buf);
}else
printf ( "can't start hello_other \n");

pvm_exit () ;
}
```

รูปที่ 2.12 โปรแกรม hello.c [3]

รูปที่ 2.13 เป็นโปรแกรม "Slave" หรือรับการแบ่งงานจาก Master ลำดับแรกจะได้รับ TaskID จาก "Master" โดยใช้ฟังก์ชัน pvm_parent () จากนั้นหาชื่อเครื่องของมัน และส่ง Message ไปให้ Master ด้วยการทำงาน 3 ขั้นตอนคือ จัดเตรียม Buffter ที่จะส่ง (pvm_initsend), ใส่ค่า String เข้าไปใน Buffter ด้วยฟังก์ชัน pvm_pkstr () (ขึ้นอยู่กับแบบของข้อมูลที่จะส่ง) และส่งข้อมูลโดยระบุเครื่องรับคือ Ptid และ Tagging (เลขระบุ Message ซึ่งตรงกับตัวรับที่เครื่อง Master ในที่นี้กำหนดเป็นเลข 1) ด้วยฟังก์ชัน pvm_send ()

```
#include "pvm3.h"
main () {
int pid, msgtag;
char buf[100];
ptid = pvm_parent () ;
strcpy (buf,  "hello,  word from") ;
getHostname (buf  +  strlen (buf) , 64);
        msgtag  =  1;
pvm_initsend (PvmDataDefault) ;
pvm_pkstr (buf);
pvm_send (ptid,  msgtag);
pvm_exit () :
}
```

รูปที่ 2.13 โปรแกรม hello_other.c [3]

2.2.4 โปรแกรม PVM และแบบของเครื่องที่ใช้งาน

โปรแกรม PVM มีแจกฟรีให้ใช้งานได้ทั่วไปทางอินเตอร์เนตที่เวบไซต์ http://netlib.org/pvm3/index.html สามารถดาวน์โหลดมาติดตั้งใช้งานได้บนเครื่องแบบต่าง ๆ ที่ระบุตามตารางที่ 2.1

ตารางที่ 2.1 รายชื่อแบบของเครื่องที่ใช้โปรแกรม PVM ได้ [3]

PVM - ARCH	Machine	Notes
AFX8	Alliant FX/8	
ALPHA	DEC Alpha	DEC OSF - 1
BAL	Sequent Balance	DUNIX
BFLY	BBN Butterfly TC 2000	
BSD386	80386/486 PC running Unix	BSDI, 386BSD,NetBSD
CM2	Thinking Machines CM2	Sun front-end
CM5	Thinking Machines CM5	Uses native Messages
CNVX	Convex C-series	IEEE f.p.
CNVXN	Convex C-series	native f.p.
CRAY	C-90, YMP, T3D port available	UNICOS
CRAY2	Cray-2	
CRAYSMP	Cray S-MP	
DGAV	Data General Aviion	
E88K	Encore 88000	
HP300	HP - 9000 model 300	HPUX
HPPA	HP - 9000 PA-RISC	
I860	Intel iPSC/860	Uses native Messages
IPSC2	Intel iPSC/2 386 Host	SysV, Uses native Messages
KSR1	Kendall Square KSR-1	OSF-1, use shared Memory
LINUX	80386/486 PC running Unix	LINUX
MASPAR	Maspar	DEC front-end
MIPS	MIPS 4680	
NEXT	NeXT	
PGON	Intel Paragon	Uses native Messages
PMAX	DECstation 3100, 51000	Ultrix
RS6K	IBM/RS6000	AIX 3.2
RT	IBM RT	
SGI	Silicon Graphics IRIS	IRIX 4.x
SGI5	Silicon Graphics IRIS	IRIX 5.x
SGIMP	SGI multiprocessor	Use shared Memory
SUN3	sun 3	SunOS 4.2
SUN4	sun 4, SPARCstation	SunOS 4.2
SUN4SOL2	sun 4, SPARCstation	Solaris 2.x
SUNMP	SPARC multiprocessor	Solaris 2.x, use shared Memory
SYMM	Sequent Symmetry	
TITN	Stardent Titan	
U370	IBM 370	AIX
UVAX	DEC Micro VAX	

2.2.5 PVM User Interface

ใน PVM3 งานทั้งหมดของ PVM ถูกกำหนดเป็นเลขจำนวนเต็มโดย pvmd ของเครื่องหลัก (Local Pvmd) เรียกว่า TID ลักษณะคล้ายกับ ProcessID (PID) ที่ใช้ในระบบปฏิบัติการ Unix ผู้ใช้ไม่จำเป็นต้องรู้ TID โปรแกรม PVM จะเป็นตัวจัดการให้เอง โปรแกรมประยุกต์เขียนด้วยภาษา C/C++ สามารถเชื่อมโยงกับ PVM Library ได้ สำหรับโปรแกรมภาษาฟอร์แทรน ก็ใช้กับ PVM ที่สนับสนุนภาษาฟอร์แทรน

รูปแบบในการสื่อสารของ PVM กำหนดว่า งานใด ๆ สามารถส่ง Message ไปยังงานอื่น ๆ ของ PVM ได้ โดยไม่จำกัดขนาดหรือจำนวนของ Message ความจริงคือจำกัดที่หน่วยความจำหลัก และหน่วยความจำชั่วคราว (Buffer) ของแต่ละเครื่อง ส่วนรูปแบบการสื่อสารแบ่งเป็น Asynchronous Blocking Send, Asynchronous Blocking Receive และ Nonblocking Receive และมีฟังก์ชันสนับสนุนด้วย การ Asynchronus Blocking Send จะหยุดการส่งเพื่อให้หน่วยความจำชั่วคราวสำหรับการส่งว่างและใช้ส่งแบบ Asynchronus คือ ส่งแบบไม่ต้องรอให้ผู้รับพร้อมรับ ใน PVM3 มีตัวเลือกในการส่งข้อมูลโดยตรงจาก Task to Task ในกรณีนี้ ถ้า Message มีขนาดใหญ่มาก ผู้ส่งอาจ Block จนกระทั่งผู้รับพร้อมรับแล้ว สำหรับการ Nonblocking Receive เกิดขึ้นเมื่อมีข้อมูลหรือ Flag แสดงว่ายังไม่มีข้อมูลส่งมาถึง ในขณะที่ Blocking Receive จะส่งค่ากลับเมื่อมีข้อมูลเข้ามาอยู่ใน Receive Buffer แล้ว นอกจากนี้แล้วก็ยังมีฟังก์ชันเพิ่มเติมในการรับส่งอีกได้แก่ ฟังก์ชัน Point-to-Point Communication, รูปแบบที่สนับสนุน Multicast สำหรับกลุ่มของงานที่จะกระจายให้กับกลุ่มงานที่ผู้ใช้กำหนด และ ฟังก์ชันในการทำ Global Max, Global Sum ฯลฯ

รูปแบบในการรับส่ง Message ของ PVM สามารถรับประกันได้ เช่นถ้า Task 1 ส่ง Message ไปยัง Task 2 แล้ว Task 1 ส่ง Message B ไปยัง Task 2 อีก Message A จะถึง Task 2 ก่อน Message B นอกจากนั้นแล้วถ้าทั้งสอง Message มาถึงก่อนที่ Task 2 จะพร้อมรับ เมื่อพร้อมแล้วจะรับ Message A ก่อน

Buffer สำหรับรับ/ส่ง Message จะถูกจัดขึ้นแบบแปรเปลี่ยน ขนาดของ Message ที่ใหญ่ที่สุดที่สามารถ รับ/ส่งได้ ขึ้นอยู่กับขนาดหน่วยความจำที่เหลือใช้งานได้ของเครื่องที่รับ และเมื่อมี Message เข้ามากเกินกว่าขนาดหน่วยความจำจะรับได้ ผู้ใช้ก็จะไม่ได้ข่าวสารการผิดพลาดซึ่ง PVM เองก็จะไม่แจ้ง Task อื่นให้หยุดการส่งมาที่ Host นี้

2.2.6 Process Control

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

ฟังก์ชัน pvm_mytid () จะส่งค่า TID ของโปรเซสนี้กลับมา และสามารถเรียกได้หลายครั้ง มันก็คือการลงทะเบียนของโปรเซสนี้กับระบบ PVM ซึ่งเป็นสิ่งแรกที่ต้องกระทำก่อนจะดำเนินการอย่างใดอย่างหนึ่งต่อไป

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

ฟังก์ชัน pvm_exit () จะเป็นการบอก pvmd ที่เครื่องหลักว่าโปรเซสนี้กำลังออกจากระบบ PVM ฟังก์ชันนี้ไม่ได้เป็นการกำจัดโปรเซส ซึ่งมันสามารถจะทำงานกับ Task เหมือนกับโปรเซสอื่น ๆ ของ Unix ได้ต่อไป โดยปกติผู้ใช้จะต้องเรียก pvm_exit () ก่อนออกจากโปรแกรมภาษา C

```
int numt = pvm_spawn (char *Task, char **argv, int flag,
                      char *where, int ntask, int *tids)
```

ฟังก์ชัน pvm_spawn () จะทำการคัดลอกสำเนาไฟล์ประมวลผลตามที่ระบุในอากิวเมนต์ char *Task จำนวน ntask ไปยังเครื่องใน Virtual Machine เช่น argv คือ พอยน์เตอร์ที่ชี้ไปยัง Array ของ Task ถ้าเป็นท้าย away จะระบุเป็น NULL แต่ถ้าไม่มีอาร์กิวเมนต์ Task แล้ว argv จะมีค่าเป็น NULL ส่วนอาร์กิวเมนต์ flag เป็นตัวระบุตัวเลือกและผลรวมของตัวเลือก ดังนี้.-

Value	Option	Meaning
0	PvmTaskDefault	ให้ PVM เลือกตำแหน่งที่จะกระจายโปรเซสเอง
1	PvmTaskHost	กระจาย Task ให้กับ Host ที่ระบุตาม อาร์กิวเมนต์ where
2	PvmTaskArch	กระจาย Task ให้กับแบบของเครื่องที่ระบุตามอาร์กิวเมนต์ where
4	PvmTaskDebug	เริ่มกระจาย Task ภายใต้ตัวตรวจแก้โปรแกรม
8	PvmTaskTrace	จัดตั้งตัวติดตามข้อมูล
16	PvmMppFront	เริ่ม Task บนเครื่องประมวลผลแบบขนาน Mpp
32	PvmHostCompl	กระจาย Task ไปยังเครื่องที่เป็นชุดของ Host ที่ระบุตามอาร์กิวเมนต์ where

ชื่อของตัวเลือก (option) เหล่านี้ถูกระบุไว้แล้วใน pvm3/include/pvm3.h สามารถนำมาใช้งานได้เลย

PvmTaskTrace เป็นคุณลักษณะใหม่ใน PVM 3.3 มันจัดตั้งตัวตามข้อมูลเพื่อใช้สำหรับ XPVM ซึ่งผู้ใช้จะต้องระบุ where คือ ที่ซึ่งจะทำการติดตามในฟังก์ชัน pvm_setopt () ด้วย

ค่าของ numt จะได้เป็นเลขจำนวนเต็ม หมายถึง จำนวนของ Task ที่กระจายได้สำเร็จหรือถ้าไม่สำเร็จจะแสดงเป็นรหัสความผิดพลาด ถ้า Task เริ่มงานแล้ว pvm_spawn () จะส่งค่าเวคเตอร์ของ tid ของ Task ที่กระจายแล้ว และถ้ามีบาง Task ไม่สามารถเริ่มงานได้ รหัสความผิดพลาดจะถูกใส่ไว้ใน ntask_numt ซึ่งเป็นตำแหน่งสุดท้ายของเวคเตอร์

```
int info = pvm_kill (int tid)
```

ฟังก์ชัน pvm_kill () จะกำจัด Task ของ PVM ตามที่ถูกระบุใน Tid ปกติควรจะใช้ pvm_exit () แทน

```
int info = pvm_catchout (FILE *ff)
```

โดยทั่วไป PVM จะเขียน Stderr และ Stdout ของ Task ที่กระจายแล้วไปยังล็อคไฟล์ /tmp/pvml.<uid> ฟังก์ชันนี้จะเป็นการกำหนดให้เขียนลงไฟล์ตามที่ระบุแทน Pvmds ของเครื่องอื่นใน Virtual machine จะทำหน้าที่เขียนลงไฟล์ในเครื่องของตนและส่งผลนี้ให้กับเครื่องหลัก เพื่อนำไปเขียนเพิ่มเติมในแต่ละบรรทัดของไฟล์ในเครื่องหลัก

ถ้าฟังก์ชัน pvm_exit () ถูกเรียกโดยเครื่องหลักก่อนที่การส่ง Output ดังกล่าวจะเสร็จสิ้น มันจะ Block จนกระทั่งการส่งเสร็จสิ้นแล้วจึงจบงาน เพื่อที่จะหลีกเลี่ยงปัญหาดังกล่าว สามารถเรียก pvm_catchout (0) ก่อนเรียก pvm_exit ()

2.2.7 Information

```
int tid = pvm_parent (void)
```

ฟังก์ชัน pvm_parent () จะส่งกลับค่า TID ของโปรเซสที่กระจาย Task นี้มาให้ หรือถ้ามีค่าเท่ากับ PvmNoParent หมายความว่า Task นี้ไม่ได้ถูกกระจายมาโดย pvm_spawn () คือเป็นเครื่องหลักนั่นเอง

```
int dtid = pvm_tidtohost (int tid)
```

ฟังก์ชันนี้จะส่งค่า TID ของโปรเซสเบื้องหลัง (Daemon) ที่กำลังทำงานอยู่บนเครื่องเดียวกับ Tid ฟังก์ชันนี้ใช้ประโยชน์ในการหา Host ที่ Task หนึ่งกำลังทำงานอยู่ ข่าวสารมากกว่านี้ ซึ่งเกี่ยวกับข้อมูลของ Virtual Machine สามารถใช้ฟังก์ชันต่อไปนี้

```
int info = pvm_config (int *nhost, int *narch,
                       struct pvmhostinfo ** hostp)
```

ฟังก์ชัน pvm_config () จะส่งกลับเกี่ยวกับข้อมูลของ Virtual Machine รวมถึงจำนวนของ Host จาก nhost ,จำนวนของแบบข้อมูลที่แตกต่างกันจาก narch, hostp คือ ตัวชี้ไปยัง Array pvmhostinfo ที่กำหนดโดยผู้ใช้ จำนวน Array อย่างน้อยที่สุด ควรจะมีขนาดเท่ากับ nhost ค่าที่ส่งกลับให้กับ Structure pvmhostinfo แต่ละค่าประกอบด้วย TID ของ Pvmd, ชื่อของ Host, แบบของเครื่องและความเร็วของ CPU ที่สัมพันธ์กับระบบ PVM

```
int info = pvm_tasks (int which, int *ntask,
                      struct pvmtaskinfo ** taskp)
```

ฟังก์ชัน pvm_task () จะส่งกลับข้อมูล Task ของ PVM ที่กำลังทำงานอยู่บนเครื่อง Virtual Machine ค่าจำนวนเต็ม which หมายถึง หมายเลข Task ที่จะส่งกลับข้อมูลมาให้ มีตัวเลือกคือ 0 มีความหมายว่า ทุก ๆ Task Pvmd TID (dtid) หมายถึง Task ที่กำลังทำงานบน Host นั้น หรือหมายเลข TID คือ กำหนดว่าต้องการข้อมูลของ Task นั้น

จำนวนของ Task ส่งกลับมาใน ntask ส่วน taskp คือตัวชี้ไปยัง Array ของ Structure pvmtaskinfo ซึ่งจำนวนของ Array มีขนาดเท่ากับ ntask สำหรับ Structure pvmtaskinfo แต่ละค่าประกอบด้วย TID, Pvmd TID, Parent TID, สถานะของ flag และชื่อของไฟล์ที่กระจายมาให้ (ถ้า Started Task ด้วยมือแล้ว PVM จะไม่รู้ชื่อไฟล์ จึงแสดงค่านี้ด้วยช่องว่าง)

2.2.8 Dynamic Configuration Host

```
int info = pvm_addHosts (char **Host, int nhost, int *infos)
int info = pvm_delHosts (char **Host, int nhost, int *infos)
```

เป็นฟังก์ชันในภาษา C สำหรับเพิ่มหรือลบชุดของ Hosts ใน Virtual machine ส่วน info จะส่งกลับจำนวนของ Host ที่เพิ่มได้สำเร็จ อาร์กิวเมนต์ infos คือ Array ที่มีจำนวนเท่ากับ nhost ประกอบด้วย รหัสแสดงสถานะของแต่ละ Host ว่าเพิ่ม/ลบ ผู้ใช้สามารถตรวจสอบว่าชุดใด หรือ Host ใดมีปัญหาในการพยายามที่จะเพิ่มหรือลบ Host

บางครั้งก็ใช้สำหรับ Setup เครื่องใน Virtual Machine แต่บ่อยครั้งใช้เพื่อเพิ่มความอ่อนตัว และป้องกันความผิดพลาด ของโปรแกรมประยุกต์ขนาดใหญ่ ฟังก์ชันเหล่านี้ยอมให้โปรแกรมประยุกต์หนึ่งเพิ่มความสามารถในการคำนวณ (เพิ่ม Host)

2.2.9 Signaling

```
int info = pvm_pvm_sendsig (int tid, int signum)
int info = pvm_notify (int what, int msgtag, int cnt, int tids)
```

ฟังก์ชัน pvm_sendsig () จะส่ง signum ไปยัง PVM Task ที่ถูกระบุใน tid ส่วนฟังก์ชัน pvm_notify () จะร้องขอการแจ้งเตือนจาก PVM เมื่อมีการตรวจพบเหตุการณ์ต่าง ๆ ดังนี้

PvmTaskExit - แจ้งเตือนเมื่อ Task Exit

PvmHostDelete - แจ้งเตือนเมื่อ Host หนึ่งถูกลบแล้ว (หรือ Host ล้มเหลว)

PvmHostAdd - แจ้งเตือนเมื่อ Host ถูกเพิ่มแล้ว

ในการร้องขอการแจ้งเตือน หมายเลขของข่าวสารที่ร้องขอ จะถูกส่งโดย PVM กลับไปตามหมายเลขที่ระบุใน msgtag ส่วน tids เป็น Array ที่จะระบุ Task อะไรที่ต้องการดู ในกรณี Task Exit หรือ HostDelete แต่ถ้าเป็น HostAdd จะมีค่าว่างไว้เพราะไม่รู้ว่า Host อะไรที่จะเพิ่มซึ่งสามารถใช้ฟังก์ชัน pvm_config () และ pvm_task () เพื่อขอรับทราบ Task และ Pvmd Tasks

ถ้า Host ที่กำลังทำงาน Task A ล้มเหลว และ Task B ขอการแจ้งเตือนว่า Task A Exit หรือยัง Task B ก็จะได้รับการแจ้งเตือนให้ Exit เนื่องจากมีความสัมพันธ์กับ Task A โดยทางอ้อม

2.2.10 Setting and Getting Options

```
int info = pvm_setopt (int what, int Val)
int val  = pvm_getopt (int what)
```

ฟังก์ชัน pvm_setopt () เป็นฟังก์ชันที่มีความมุ่งหมาย เพื่อยอมให้ผู้ใช้ Set ตัวเลือกในระบบ PVM ได้ ใน PVM3, pvm_setopt () สามารถเรียกใช้ได้หลายครั้ง การกำหนดตัวเลือกนี้ ได้แก่ การพิมพ์

ข่าวสารการผิดพลาดโดยอัตโนมัติ, ระดับในการตรวจแก้ และวิธีการใช้เส้นทางในการสื่อสาร และจะส่งค่าการกำหนดครั้งที่แล้วใน oldval สำหรับอาร์กิวเมนต์ what มีค่าได้ดังต่อไปนี้

Option Value		Meaning
PvmRoute	1	กำหนดเส้นทางการสื่อสาร
PvmDebugMask	2	กำหนดการแก้ไข
PvmAutoErr	3	ให้รายงานความผิดพลาดโดยอัตโนมัติ
PvmOutputTid	4	Stdout ไปยัง Tid ลูก
PvmOutputCode	5	แสดงผล msgtag
PvmTraceTid	6	ติดตามที่ Tid ลูก
PvmTraceCode	7	ติดตาม msgtag
PvmFragSize	8	กำหนดขนาดของ Message
PvmResvTids	9	อนุญาตให้ Message สำรอง Tags และ Tids
PvmSelfOutputTid	10	Stdout ไปยังตัวเอง
PvmSelfOutputCode	11	แสดงผล msgtag ของตัวเอง
PvmSelfTraceTid	12	ติดตาม Tid ของตัวเอง
PvmSelfTraceCode	13	ติดตาม msgtag ของตัวเอง

โดยทั่วไปที่นิยมมากที่สุด ก็คือ การใช้ pvm_setopt () เพื่อให้สามารถติดต่อกันได้โดยตรงระหว่าง Task ของ PVM เช่น pvm_setopt (PvmRoute, PvmRouteDirect);

ข้อเสียของวิธีนี้คือ ถึงแม้ว่าการติดต่อเกิดความรวดเร็ว แต่ระบบ Unix ขนาดใหญ่อาจมีปัญหาได้ อาจจะไม่สามารถทำงานได้ ถ้ามี Task มากกว่า 60 Tasks ซึ่ง PVM จะตั้งค่ากลับไปที่ค่าโดยปริยาย ฟังก์ชันนี้สามารถเรียกใช้ได้หลายครั้ง ระหว่างการทำงานของโปรแกรมประยุกต์เพื่อกำหนดให้มีการติดต่อสื่อสารกันโดยตรงระหว่าง Task กับ Task แต่ต้องใช้หลังจากเรียกฟังก์ชัน pvm_mytid () แล้ว

2.2.11 Message Passing

การส่ง Message ประกอบด้วย 3 ขั้นตอน คือ หนึ่ง, หน่วยความจำชั่วคราวสำหรับการส่งต้องถูกต้องจัดเตรียม และว่างพร้อมส่งโดยการเรียกฟังกัน pvm_initsend () หรือ pvm_mkbuf () สอง, Message ต้องถูกจัดเก็บเข้าไปในหน่วยความจำชั่วคราวนี้ โดยใช้ฟังก์ชัน pvm_pk* () ตามแบบของข้อมูลที่จะส่ง สาม, การส่ง Message ไปยังโปรเซสอื่นโดยสมบูรณ์ด้วยฟังก์ชัน pvm_send () หรือส่งแบบกระจายด้วย pvm_mcast () Message หนึ่งจะถูกรับโดยการใช้ฟังก์ชัน

Blocking หรือ Nonblocking Receive และ "Unpacking" จากหน่วยความจำชั่วคราวสำหรับการรับ ซึ่งสามารถกำหนดให้รับ Message ใดๆ หรือ Message จากที่ใดตามที่ระบุ หรือระบุหมายเลขประจำ Message หรืออาจระบุหมายเลข Message จากแหล่งที่ส่งตามที่ระบุได้อีกด้วย

2.2.11.1 Message Buffer

```
int bufid = pvm_initsend (int encoding)
```

ถ้าผู้ใช้ใช้หน่วยความจำชั่วคราวสำหรับการส่งอันเดียวแล้ว สามารถใช้ฟังก์ชัน pvm_initsend () เท่านั้นก็เพียงพอ ซึ่งมันจะถูกเรียกใช้ก่อนที่จะมีการจัดเก็บ Message ใหม่เข้าไปใน Buffer ฟังก์ชันนี้จะล้าง Send Buffer ให้ว่าง และจัดเก็บ Message ใหม่ หมายเลขของหน่วยความจำชั่วคราวใหม่ส่งกลับไปใน bufid สำหรับ อาร์กิวเมนต์ encoding เป็นการระบุการเข้ารหัส Message ดังนี้.-

PvmDataDefault - เป็นการเข้ารหัส XDR ซึ่งถือเป็นค่าโดยปริยาย ใช้ในกรณีที่ไม่รู้แบบของข้อมูลระหว่าง เครื่องรับและส่ง

PvmDataRaw - หมายถึงไม่เข้ารหัส Message ที่ส่งมีรูปแบบตามความเป็นจริงเริ่มแรก ถ้าโปรเซสที่รับไม่สามารถอ่านได้ มันจะส่งกลับการผิดพลาดในระหว่างการ Unpacking

PvmDataInPlace - หมายถึง ข้อมูลจริงจะถูกนำไปจัดเก็บไว้ในที่ใดที่หนึ่งส่วน Buffer จะมีระบุเพียงขนาดและตัวชี้ไปยังข้อมูลที่จะถูกส่งเท่านั้น เมื่อฟังก์ชัน pvm_send () ถูกเรียก ข้อมูลจะถูกคัดลอกโดยตรงไปยังหน่วยความจำหลักของผู้ใช้ตัวเลือกนี้จะลดเวลาในการคัดลอก Message ได้ ซึ่งผู้ใช้ต้องไม่มีการปรับปรุงแก้ไขข้อมูลในขณะที่ถูกจัดเก็บ และในขณะที่ส่งไปแล้ว ใช้สำหรับการจัดเก็บ (pack) เพียงครั้งเดียว และมีการส่งไปยังโปรแกรมประยุกต์หลายๆ ครั้ง

ในกรณีที่ผู้ใช้ต้องการส่ง Message หลายๆ ครั้ง ก็ไม่จำเป็นต้องสร้างหน่วยความจำชั่วคราวขึ้นมาหลายๆ ครั้ง ใน PVM3 มี หน่วยความจำชั่วคราวสำหรับการส่งและการรับ ที่พร้อมใช้งานได้ตลอดเวลาอยู่แล้ว นักพัฒนาอาจสร้าง Message Buffer ขึ้นมาจำนวนหนึ่งเพื่อใช้สำหรับการ Packing, Sending, Recieve และ Unpacking ด้วยฟังก์ชันนี้

```
int bufid = pvm_mkbuf (int encoding)
```

ฟังก์ชัน pvm_mkbuf () จะสร้าง Send Buffer ว่างๆ ขึ้นมาใหม่ 1 อัน โดยระบุแบบของการเข้ารหัสตามที่กล่าวมาแล้ว มันจะส่งกลับค่าเลขประจำตัวของ Buffer ไว้ใน bufid

```
int info = pvm_freebuf (int bufid)
```

ฟังก์ชัน pvm_freebuf () จะทำให้ Buffer ตามเลขประจำตัวที่ระบุใน bufid ว่างเพื่อนำไปใช้งานต่อไป ควรเรียกใช้ Message ถูกส่งไปแล้วเป็นเวลานาน และไม่มีความต้องการ Message นั้นอีก เรียกใช้ pvm_mkbuf () เพื่อสร้าง Buffer ใหม่ สำหรับ Message ใหม่ ถ้าต้องการ ซึ่งไม่ใช้ร่วมกับ pvm_initsend () มีฟังก์ชันเพิ่มเติมเพื่อการใช้งาน Buffer แบบนี้ คือ

```
int bufid = pvm_getsbuf (void)
int bufid = pvm_getrbuf (void)
```

โดยที่ฟังก์ชัน pvm_getsbuf () จะส่งกลับค่าเลขประจำตัวของหน่วยความจำ สำหรับการส่งที่พร้อมใช้งานและฟังก์ชัน pvm_getrbuf () จะส่งกลับค่าเลขประจำตัวของหน่วยความจำชั่วคราวสำหรับการรับที่พร้อมใช้งาน

```
int oldbuf = pvm_setsbuf (int bufid)
int oldbuf = pvm_setrbuf (int bufid)
```

ฟังก์ชันด้านบนนี้ใช้ Set ค่า Active Send Buffer หรือ Receive Buffer ให้กับ bufid และเก็บสถานะของ Buffer ครั้งที่แล้ว โดยส่งกลับค่ารหัสประจำตัวของ Active Buffer ครั้งที่แล้วไว้ใน oldbuf

ถ้า bufid ถูกเซตเป็น 0 ใน pvm_setsbuf () หรือ pvm_setrbuf () แล้ว Buffer ปัจจุบันจะถูกจัดเก็บและไม่มี Active Buffer คุณลักษณะนี้สามารถใช้จัดเก็บสถานะ Message ของโปรแกรมประยุกต์ในปัจจุบัน เพื่อนำไปใช้ทดแทน Message ในกรณีที่ Message ของ PVM ไม่สอดคล้องกับ Buffer ของโปรแกรมประยุกต์หลังจากสมบูรณ์แล้ว Buffer ของโปรแกรมประยุกต์สามารถ reset ให้ Active ได้ มีโอกาสเป็นไปได้ว่า Message จะถูกส่งต่อได้โดยไม่ต้องทำการ Pack ใหม่ โดยใช้ฟังก์ชันดังนี้.-

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

2.2.11.2 Packing Data

ฟังก์ชันแต่ละตัวของภาษา C ต่อไปนี้ คือ Array ของข้อมูลแบบต่างๆ ใน Active Send Buffer ซึ่งสามารถถูกเรียกใช้ได้หลาย ๆ ครั้ง ในการ Pack ข้อมูลเข้าไปใน Message เดียว ดังนั้น Message หนึ่งจึงบรรจุหลาย ๆ Array ที่มีแบบข้อมูลแตกต่างกันไปได้

อาร์กิวเมนต์ ของแต่ละฟังก์ชัน คือ ตัวชี้ไปยังข้อมูลแรกที่ถูก Pack แล้ว nitem คือ จำนวนของข้อมูลทั้งหมดที่ถูก pack จาก Array นี้ และ stride คือ การนับในการ pack ถ้าเป็น 1 หมายถึงถูก Pack ชิดกันมาก และถ้าเป็น 2 ก็ห่างกันออกไป ยกเว้น pvm_pkstr () ซึ่งมีข้อจำกัดคือ ถ้า Pack ค่า String เป็น NULL แล้วจะไม่ต้องการ อาร์กิวเมนต์ nitem หรือ stride อีก

```
int info = pvm_pkbyte   (char   *cp, int nitem, int stride)
int info = pvm_pkcplx   (float  *xp, int nitem, int stride)
int info = pvm_pkcplx   (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    *np, int nitem, int stride)
int info = pvm_pklong   (long   *np, int nitem, int stride)
int info = pvm_pkshort  (short  *np, int nitem, int stride)
int info = pvm_pkstr    (char   *cp)
int info = pvm_pkckf    (const  char * fmt, ...)
```

2.2.11.3 Sending and Receive Data

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

ฟังก์ชัน pvm_send () จะทำการส่ง Message ที่มีเลขประจำตัว msgtag และส่งทันทีไปยังโปรเซสที่มี TID เท่ากับ tid ฟังก์ชัน pvm_mcast () จะทำการส่ง Message ที่มีเลขประจำตัว msgtag และ กระจายไปยัง Task ทั้งหมดที่ระบุใน Array ของ tid ซึ่งมีจำนวน Array เท่ากับ ntask

```
int info = pvm_psend (int tid, int msgtag,
                      void  *vp, int cnt, int type
```

ฟังก์ชัน pvm_psend () จะทำการ Pack และส่ง Array ของแบบข้อมูลตามที่ระบุ ไปยัง TID เท่ากับอาถิวเมนต์ tid แบบข้อมูลในอาถิวเมนต์ type กำหนดไว้แล้ว ดังนี้.-

PVM_STR	PVM_FLOAT
PVM-BYTE	PVM_CPLX
PVM_SHORT	PVM_DOUBLE
PVM_INT	PVM_DCPLX
PVM_LONG	PVM_ULNT
PVM_USHORT	PVM_ULONG

PVM มีหลายวิธีในการรับ Message ของ Task หนึ่ง ซึ่งอาจเป็นฟังก์ชันที่ไม่ตรงกับการส่งได้ ตัวอย่างเช่น pvm_psend () ต้องตรงกับ pvm_precv () แต่ฟังก์ชันต่อไปนี้สามารถเรียกใช้ในการรับ Message ใด ๆ ที่เข้ามาได้ไม่ว่าจะส่งมาแบบไหนก็ตาม หรืออาจจะส่งแบบกระจายได้

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

เป็น Blocking Receive ที่จะคอยจนกระทั่ง Message ที่มีเลขประจำตัวตามอาถิวเมนต์ msgtag มาถึงจากอาถิวเมนต์ tid ถ้าใส่ค่า msgtag หรือ tid เท่ากับ – 1 หมายความว่ารับทุก ๆ Message ที่มาถึง

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

ถ้า Message ที่ต้องการยังมาไม่ถึงแล้ว ฟังก์ชัน Nonblocking Receive, pvm_nrecv () จะส่งค่าให้ bufid = 0 ฟังก์ชันนี้สามารถเรียกใช้งานได้หลาย ๆ ครั้ง สำหรับ Message เดียวกัน เพื่อตรวจสอบว่ามันมาถึงหรือยัง เมื่อมาถึงแล้วสามารถใช้ ฟังก์ชัน Blocking Receive pvm_recv () กับ Message ตัวเดียวกันได้ ถ้า Message เลขประจำตัวตามอาถิวเมนต์ msgtag มาถึงแล้วจาก tid ฟังก์ชัน pvm_nrecv () จะจัดเก็บ Message นี้ไว้ใน Active Receive Buffer ที่สร้างขึ้นใหม่ และส่งค่ารหัส ID ของ Buffer นี้กลับออกมาแต่ถ้า Active Receive อันที่แล้วว่างอยู่จากการล้างและนำไปบันทึกเก็บด้วย pvm_setsbuf () แล้ว มันก็จะใช้ตัวนี้แทนการสร้างใหม่ ค่าของ msgtag หรือ tid เท่ากับ – 1 หมายถึงรับได้ทุก ๆ Message

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

ถ้า Message ที่ร้องขอยังมาไม่ถึง ฟังก์ชัน pvm_probe () จะส่งค่า 0 ให้ตัวแปร bufid ถ้ามาถึงแล้ว จะส่งค่าอื่นให้ bufid แต่ก็จะไม่รับมัน ฟังก์ชันนี้สามารถเรียกใช้ได้หลาย ๆ ครั้งกับ Message เดียวกัน เพื่อตรวจสอบว่า Message ที่ระบุมาถึงหรือยัง ใช้ประโยชน์เพื่อตรวจสอบในขณะทำงานระหว่าง Task นอกจากนี้ยังสามารถใช้ ฟังก์ชัน pvm_bufinfo () สามารถนำมาใช้เพื่อหาข้อมูลเกี่ยวกับ Message ก่อนที่จะรับมัน

```
int bufid = pvm_trecv (int tid, int msgtag, struct timeval *tmout)
```

PVM มีฟังก์ชันสนับสนุนการจำกัดเวลาในการรับ ในกรณีที่ Message ไม่เคยมาถึงเลยอาจเนื่องจากการส่งล้มเหลวหรือมีการผิดพลาด ฟังก์ชัน pvm_recv () จะ Block เพื่อรอรับตลอดไป เพื่อหลีกเลี่ยงสถานะดังกล่าว ผู้ใช้อาจกำหนดเวลาที่แน่ชัดว่าให้รอ Message ถึงเมื่อไร ฟังก์ชัน pvm_trecv () ยอมให้ผู้ใช้ระบุช่วงเวลาคอย Message ถ้ากำหนดเวลาคอยนานมาก pvm_trecv () จะทำงานเหมือนฟังก์ชัน pvm_recv () ถ้ากำหนดเวลาคอยเป็น 0 pvm_trecv () จะทำงานเหมือนฟังก์ชัน pvm_nrecv () ดังนั้น pvm_trecv () เป็น ฟังก์ชัน ที่ใช้สำหรับเติมช่องว่างระหว่างฟังก์ชัน Blocking กับ Nonblocking

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

ฟังก์ชันนี้จะส่งค่ากลับของ msgtag, tid และความยาวเป็นไบต์ของ Message ที่มีเลขประจำตัวตามอากิวเมนต์ bufid มันสามารถใช้หาป้ายชื่อ และ ต้นทางของการส่ง Message เพื่อที่จะระบุการรับที่ถูกต้อง และเหมาะสมได้

```
int info = pvm_precv (int tid, int msgtag, void *VP, int cnt,
                      int type, int *rtid, int *rtag, int *rcnt)
```

ฟังก์ชัน pvm_precv () จะเป็นฟังก์ชันรวม Blocking Receive และ Unpacking Receive Buffer เข้าด้วยกัน มันไม่ส่งกลับค่าให้กับ bufid แต่จะส่งกลับค่าที่แท้จริงของ tid, msgtag และ cnt

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

ฟังก์ชัน pvm_recvf () เป็นฟังก์ชันที่ใช้ปรับปรุงแก้ไข รายละเอียดของฟังก์ชันการรับ เพื่อเพิ่มขีดความสามารถให้ PVM โดยทั่วไปการรับจะรับ Message ที่ตรงกับการระบุต้นทาง และเลขประจำตัวของ Message (Message Tag) ฟังก์ชันนี้สามารถปรับปรุงแก้ไขรายละเอียด เพื่อทำการเปรียบเทียบ และนำไปใช้ร่วมกับฟังก์ชันอื่นได้

2.2.11.4 Unpacking Data

ฟังก์ชันในภาษา C ต่อไปนี้ใช้สำหรับ Unpack ข้อมูลแบบต่าง ๆ ใน Active Receive Buffer ซึ่งควรใช้ให้ถูกต้องตรงกันกับการใช้ฟังก์ชันในการ Pack ข้อมูลโดยระบุ type , จำนวนของ item และ stride ให้ตรงกัน อากิวเมนต์ nitem คือจำนวนของ Item ตามแบบข้อมูลการส่งเพื่อ Unpack และอาร์กิวเมนต์ stride คือ ช่วงของ Item

```
int info = pvm_upkbyte   ( char   *cp, int nitem, int stride)
int info = pvm_upkeplx   ( float  *xp, int nitem, int stride)
int info = pvm_upkeplx   (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    *np, int nitem, int stride)
int info = pvm_upklong   (long   *np, int nitem, int stride)
int info = pvm_upkshort  (short  *np, int nitem, int stride)
int info = pvm_upkstr    (char   *cp)
int info = pvm_unpkckf   (const  char * fmt, ...)
```

2.2.12 Dynamic Process Groups

ฟังก์ชัน Dynamic Process Group ถูกสร้างขึ้นมารวมอยู่ในแก่นของฟังก์ชัน PVM ซึ่งถูกแยกเป็น Library ต่างหาก คือ libgpvm3.a โปรแกรมของผู้ใช้จะต้องเชื่อมโยงกับ Library ก่อนจึงจะสามารถใช้งานฟังก์ชันของกลุ่มได้ Pvmd ไม่ทำงานกับฟังก์ชันกลุ่ม Task เหล่านี้ถูกควบคุมโดย Group Server ซึ่งจะ Start โดยอัตโนมัติเมื่อกลุ่มแรกถูกจัดตั้ง มีหลาย ๆ คนโต้แย้งว่ากลุ่มควรควบคุมด้วย Message-Passing ด้วยเหตุผลถึง ประสิทธิภาพ , ความมีเสถียรภาพ และความสมดุลย์ระหว่าง Static กับ Dynamic Group แต่บางคนก็ให้ความเห็นว่า Task ในกลุ่มเท่านั้นที่ควรจะสามารถเรียกใช้ฟังก์ชันกลุ่มได้

ฟังก์ชันกลุ่มใน PVM ออกแบบให้ง่ายต่อการใช้งานของผู้ใช้อย่างมีประสิทธิภาพ Task ของ PVM สามารถ Join และ Leave ออกจากกลุ่มในขณะใด ๆ ก็ได้โดยไม่ต้องแจ้งให้ Task อื่น

ทราบ และไม่มีผลกระทบกับ Task อื่นในกลุ่ม Task สามารถกระจาย Message ไปยังกลุ่มที่มันไม่ได้เป็นสมาชิกได้ โดยทั่วไป Task ของ PVM อาจจะเรียกใช้ฟังก์ชันกลุ่มในขณะใดก็ได้ ยกเว้นฟังก์ชัน pvm_lvgroup (), pvm_barrier () และ pvm_reduce () ซึ่งจะต้องถูกเรียกใช้ได้โดย Task ที่เป็นสมาชิกของกลุ่มที่ระบุเท่านั้น

```
int inum  = pvm_joingroup (char *group)
int info  = pvm_lvgroup (char *group)
```

ฟังก์ชันเหล่านี้ยอมให้ Task หนึ่ง Join หรือ Leave จากกลุ่มที่ผู้ใช้กำหนดฟังก์ชันแรก คือ pvm_joingroup () จะสร้างกลุ่มขึ้นด้วยชื่อตามที่ระบุใน Group และใส่ Task ที่เรียกฟังก์ชันนี้เข้าไปในกลุ่มนี้ ฟังก์ชันนี้จะส่งค่าจำนวนของโปรเซสในกลุ่มนี้ ซึ่งมีค่าจาก 0 ถึง (จำนวนของสมาชิกในกลุ่ม – 1) ให้กับตัวแปร inum ใน PVM Task หนึ่งสามารถ Join ได้หลาย ๆ กลุ่ม

ถ้าโปรเซสหนึ่งออกจากกลุ่ม และกลับเข้าไปในกลุ่มใหม่ โปรเซสนั้นอาจได้รับค่าตัวเลขที่ไม่เหมือนกัน ค่าตัวเลข Instance Number จะถูกจัดสรรใหม่ และให้ค่าตัวเลขน้อยที่สุดก่อนกับ Task ใหม่ แต่ถ้าเข้ากลุ่มทีละหลาย Task ก็รับประกันไม่ได้ว่าจะได้ค่าตามลำดับ

```
int tid   = pvm_gettid (char *group, int inum)
int inum  = pvm_getinst (char *group , int tid)
int size  = pvm_gsize (char *group)
```

ฟังก์ชัน pvm_gettid () จะส่งกลับค่า Tid ของโปรเซสให้กับตัวแปร tid รวมทั้งชื่อของกลุ่ม และเลข Instance Number ฟังก์ชัน pvm_getinst () จะส่งกลับค่า Instance Number ของ Tid ในกลุ่มที่ระบุ ฟังก์ชัน pvm_gsize () จะส่งกลับค่าจำนวนของสมาชิกในกลุ่มที่ระบุ

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

เมื่อเรียกฟังก์ชัน pvm_barrier () โปรเซสจะถูก Block จนกระทั่งจำนวนของสมาชิกตามที่ระบุในอาร์กิวเมนต์ count ของกลุ่มถูกเรียกครบแล้ว โดยทั่วไปแล้ว count ควรจะเท่ากับจำนวนสมาชิกทั้งหมดของกลุ่ม และที่ต้องแจ้งจำนวนสมาชิก เพราะว่าด้วย Dynamic Process Group PVM จะไม่รู้ว่าสมาชิกในกลุ่มมีจำนวนเท่าใด ณ เวลาหนึ่ง จะเกิดความผิดพลาด ถ้าโปรเซสที่เรียกใช้ฟังก์ชัน pvm_barrier () ไม่ใช่สมาชิกของกลุ่มหรือสมาชิกตัวหนึ่งของกลุ่มเรียก pvm_barrier () ด้วยอาร์กิวเมนต์ count=4 และสมาชิกตัวอื่นของกลุ่มเรียก pvm_barrier () ด้วยอาร์กิวเมนต์ count=5

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

ฟังก์ชันนี้จะกระจาย Message ที่มี ID ตามอาร์กิวเมนต์ msgtag ไปยัง Task ทั้งหมดของกลุ่มที่ระบุ ยกเว้นตัวมันเอง ถ้ามันเป็นสมาชิกภายในกลุ่ม ถ้า Task ใด Join กลุ่มในระหว่างที่กระจาย Message Task นั้นจะไม่ได้รับ Message ที่กระจาย ถ้า Task ใดออกจากกลุ่มระหว่างกระจาย Message สำเนาของ Message จะหยุดส่งถึง Task นั้น

```
int info  = pvm_reduce (void (*func )(), void *data,
                        int nitem, int datatype,
                        int msgtag, char *group, int root)
```

ฟังก์ชันนี้ใช้เพื่อทำการคำนวณทางคณิตศาสตร์ข้ามกลุ่ม ตัวอย่างเช่น Global Sum หรือ Global Max เพื่อลดการทำงานซ้ำซ้อนได้ ผลลัพธ์การลดการคำนวณจะแสดงในอาร์กิวเมนต์ root, PVM มีการกำหนดฟังก์ชันการคำนวณนี้ เพื่อให้ผู้ใช้นำไปแทนค่าในอาร์กิวเมนต์ func ดังนี้.-

PvmMax

PvmMin

PvmSum

PvmProduct

การลดการคำนวณนี้จะกระทำกับข้อมูลที่ใส่ในอาร์กิวเมนต์ data เช่น ถ้า Array ของ data ประกอบด้วยค่าเลขจำนวนจริง 2 ค่า และอาร์กิวเมนต์ func กำหนดเป็น PvmMax แล้วผลลัพธ์จะประกอบด้วย 2 จำนวน คือ ค่ามากที่สุดของสมาชิกในกลุ่มตามเลขจำนวนแรก และค่ามากที่สุดของสมาชิกในกลุ่มตามเลขที่สอง

ความสามารถเพิ่มเติมคือผู้ใช้สามารถกำหนดฟังก์ชันของตัวเองได้ แล้วแทนค่ามันลงในอาร์กิวเมนต์ func ตัวอย่างการใช้งานอยู่ใน Souce Code ของ PVM ดูใน PVM_ROOT/examples/gexamples.

2.3 พื้นฐานการเขียนโปรแกรมแบบขนาน

การคำนวณแบบขนานด้วยระบบ PVM มีโมเดลสำหรับการเขียนโปรแกรมอยู่ 3 ชนิด คือ

2.3.1 Crowd Computation

การทำงานแบบนี้จะรวบรวมโปรเซสที่สัมพันธ์กันอย่างใกล้ชิดเข้าไว้ด้วยกัน โดยทำการประมวลผลด้วยรหัสโปรแกรมที่เหมือนกันแต่มี Workloadในการคำนวณที่ต่างกัน โดยปกติแล้วจะใช้แลกเปลี่ยนผลลัพธ์ที่เกิดขึ้นระหว่างการคำนวณ การคำนวณแบบนี้แบ่งได้เป็น 2 ประเภทดังนี้

2.3.1.1 Master-Slave (Host-Node Model)

จะแบ่งการควบคุมโปรแกรมเทอมของ โปรแกรมหลักกับโปรแกรมรอง ซึ่งโปรแกรมหลักจะรับผิดชอบในการทำการ กระจายโปรเซส รวบรวม และ แสดงผลลัพธ์ บางทีก็ควบคุมในเรื่องเวลาด้วย โปรแกรมรองจะเป็นส่วนที่ทำการคำนวณจริง ซึ่งแต่ละโปรแกรมรองจะถูกจัดสรรงานให้โดยโปรแกรมหลัก รูปแบบของ Master-Slave

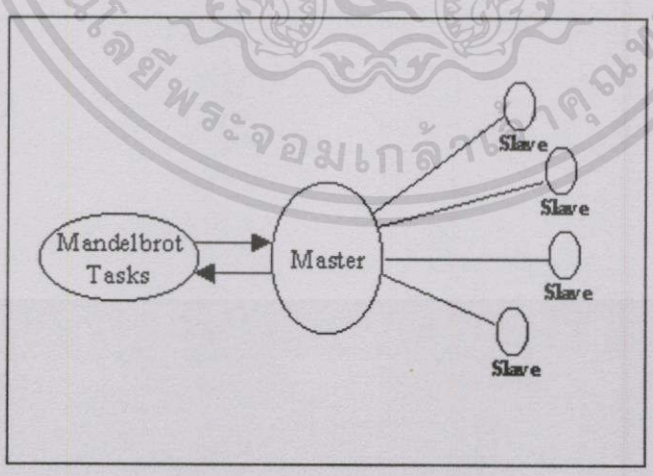
2.3.2.2 Node-Only Model

จะมีโปรแกรมทำงานเพียงโปรแกรมเดียว ซึ่งโปรแกรมหลักและโปรแกรมรองจะใช้โปรแกรมนี้ร่วมกัน โดยทั้งโปรแกรมหลักและโปรแกรมรองจะรู้เองว่าจะต้องทำงานในส่วนใดของโปรแกรม Crowd Computations มีการทำงานอยู่ 3 ขั้นตอนคือ

1) การเริ่มจัดกลุ่มของโปรเซส (กรณีของ Node-only จะเป็นการกระจายข้อมูลของกลุ่ม และปัจจัยปัญหาต่าง ๆ) จากนั้นจะกำหนดและจ่ายงานที่เหมาะสม (Workload Allocation)

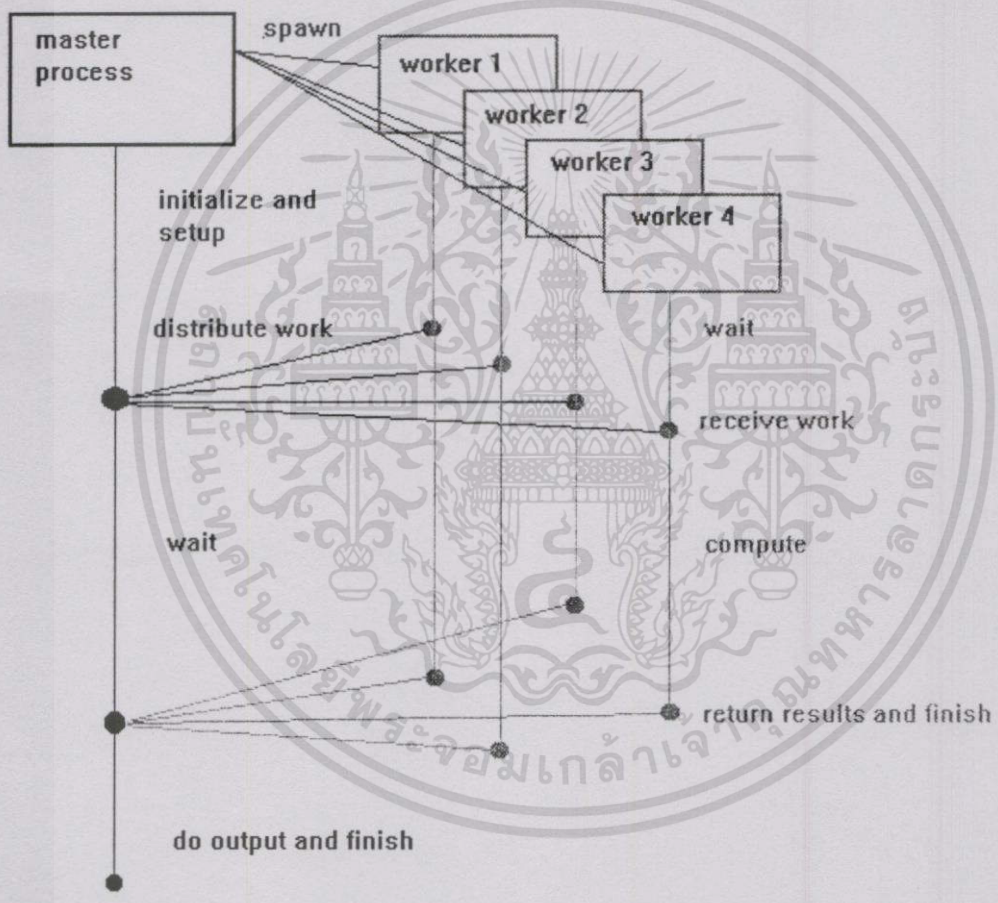
2) ขั้นการคำนวณ

3) การรวบรวมผลลัพธ์ และแสดงผล (ในขั้นนี้กลุ่มของโปรเซสจะถูกสลายไปหรือหยุดลง)



รูปที่ 2.14 รูปแบบของ Master-Slave Model [3]

สำหรับโมเดลแบบ Master-Slave ที่แสดงตามรูปที่ 2.14 จะทำการคำนวณตามแบบที่รู้จักทั่วไปว่า Mandelbrot ซึ่งสามารถแบ่งปัญหาเป็นส่วนย่อย ๆ เพื่อแยกไปทำการคำนวณ และรวบรวมผลลัพธ์แบบง่าย ๆ อย่างไรก็ตามโมเดลแบบนี้ยอมให้มีการจัดแบบไม่ตายตัว (Dynamic Load Balancing) ด้วยเหตุนี้จึงยอมให้การทำงานในแต่ละส่วนแบ่งงานให้กันได้ (Share the Workload Unevenly) ตัวอย่างขั้นตอนการทำงานแสดงได้ตามรูปที่ 2.15 และอัลกอริทึมโดยใช้ฟังก์ชันของ PVM ในการควบคุม Master-Slave แสดงตามรูปที่ 2.16



รูปที่ 2.15 ขั้นตอนการทำงานของโปรแกรม Master-Worker

```
{Master Mandelbrot algorithm.}
{Initial placement}
for I := 0 to NumWorkers - 1
pvm_spawn (<worker name>)          {Start up worker I}
```

```
pvm_send (<worker tid>), 999)        {Send Task to worker I}
endfor
{Receive-send}
while (WordToDo)
pvm_recv (888)                {Receive result}
pvm_send (<available worker tid >, 999)
{Send next Task to available worker}
display result
endwhile
{Gather remaining results.}
for I := 0 to Numworkers - 1
pvm_recv (888)                {Receive result}
pvm_kill (<worker tid I>)     {Terminate worker I}
display result
endfor

{Worker Mandelbrot algorithm}
while (true)
pvm_recv (999)                {Receive Task}
result := MandelbrotCalculations (Task) {Compute result}
pvm_send (<Master tid>, 888)         {Send result to Master}
endwhile
```

รูปที่ 2.16 อัลกอริทึมของการคำนวณ Mandelbrot โดยใช้ฟังก์ชันของ PVM ในแบบ Master-Slave [3]

ตัวอย่าง Master-Slave ตามที่แสดงจะไม่มีการติดต่อระหว่าง Slave แต่การคำนวณแบบ Crowd Computations โดยส่วนใหญ่แล้วมีความยุ่งยากสับสนมาก มักต้องการการติดต่อสื่อสารระหว่างโปรเซสในการคำนวณ ตัวอย่างเช่นการคำนวณเมตริกซ์ โดยใช้ Cannon's Algorithm [2] เป็นโครงสร้างแบบ Node-Only ตามรูปที่ 2.17 แสดงอัลกอริทึมการคำนวณเมตริกซ์

```
{Matrix Multiplication Using Pipe-Multiply-Roll Algorithm}

{Processor 0 starts up other processes}
```

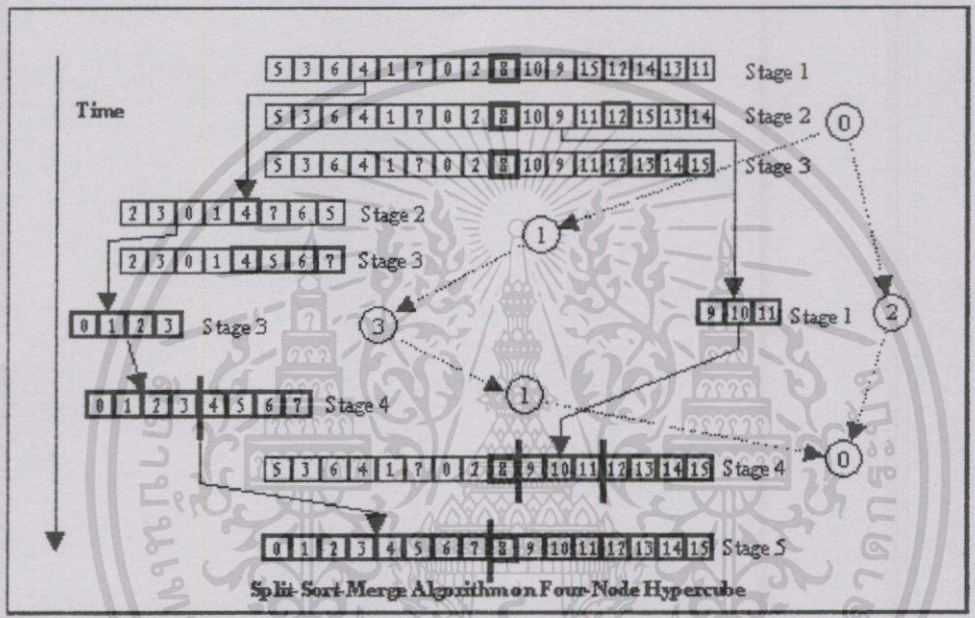
```
if (<my Processor number > = 0 ) then
        for i := 1 to MeshDimension*MeshDimension
                pvm_spawn (<component name>, . . )
        endfor
endif

forall Processors Pij, 0 <= i, j < MeshDimension
        for k := 0 to MeshDimension — 1
                {Pipe.}
                if myrow = (mycolumn + k) mod MeshDimension
                    {Send A to all Pxy, x = myrow, y <> mycolumn}
                    pvm_mcast ((Pxy, x = myrow, y <> mycolumn), 999)
                else
                    pvm_recv (999)      {Receive A}
                endif

                {Multiply. Running totals maintained in C.}
                Multiply (A, B, C)

                {Roll. }
                {Send B to Pxy, x = myrow-1, y = mycolumn}
                pvm_send ((Pxy, x = myrow-1, y = mycolumn , 888)
                pvm_recv (888)      {Receive B}
        endfor
endfor
```

รูปที่ 2.17 อัลกอริทึมการคำนวณเมตริกซ์ [3]

2.3.2 Tree Computation

การคำนวณแบบนี้โปรเซสที่ถูกแบ่งจะมีลักษณะเป็นแบบต้นไม้ (Tree) ซึ่งจะมีความสัมพันธ์กันกระหว่าง Parent-Child (ตรงข้ามกับ Crowd Computation ที่มีลักษณะความสัมพันธ์เหมือนกับดาว) โปรแกรมแบบนี้จะไม่รู้ตำแหน่งของการจ่ายงานทั้งหมด ตัวอย่างของการคำนวณแบบนี้ เช่น Branch-and-Bound Algorithms, Alpha-Beta Search และ Recursive

"Divide-and-Conquer" Algorithms ตัวอย่างที่แสดงให้ดูเป็นอัลกอริทึมการจัดเรียงข้อมูลแบบขนาน (Parallel Sorting Algorithm) ขั้นที่ 1 โปรเซสเริ่มทำงานใน PVM มีข้อมูลที่จะทำการจัดเรียงทั้งหมด สำหรับในขั้นที่ 2 จะแบ่งโปรเซสและส่งมันออกเป็นครึ่งหนึ่งของข้อมูล และถูกแบ่งออกเป็นครึ่งหนึ่งทุกระดับ ลักษณะแบบ Tree จนกว่าจะมีความลึกที่เหมาะสม คือแต่ละโปรเซสย่อยสามารถจัดเรียงข้อมูลของตนเองได้อย่างอิสระ จากนั้นเมื่อแต่ละโปรเซสทำการจัดเรียงเสร็จเรียบร้อยแล้วก็จะทำการส่งข้อมูลกลับมารวมกัน โครงสร้างการทำงานแสดงได้ตามรูปที่ 2.18



รูปที่ 2.18 ตัวอย่างโครงสร้างการจัดเรียงข้อมูลแบบ Tree Computational [3]

```
{ Spawn and partition list based on a broadcast tree pattern. }
for i := 1 to N, such that 2^N = NumProcs
for all Processors P such that P < 2^i
pvm_spawn ( . . . ) {process id P XOR 2^i }
if P < 2^ ( i-1) then
midpt : = PartitionList (list);
{Send list [0 . . midpt ] to P XOR 2^i }
pvm_send ((P XOR 2^i) , 999)
list := list [ midpt+1 . . MAXSIZE ]
else
pvm_recv (999)    {receive the list}
endif
```

```
endfor
endfor

{Sort remaining list.}
Quicksort (list[midpt+1 . . MAXSIZE])

{Gather/merge sorted sub-lists.}
for i := N downto 1, such that 2^N = NumProcs
for all Processors P such that P < 2^i
fi P > 2^(i-1) then
pvm_send ((P XOR 2^i), 888)
{Send list to P XOR 2^i}
else
pvm_recv (888)   {Receive temp list}
merge templist into list
endif
endfor
endfor
```

รูปที่ 2.19 อัลกอริทึมการจัดเรียงข้อมูลแบบ Tree Computational [3]

2.3.3 Hybrid model

เป็นการผสมผสานกันระหว่าง Tree Model กับ Crowd Model โมเดลทั้งสามนี้มีพื้นฐานของความสัมพันธ์ระหว่างโปรเซส (Process Relationships) ซึ่งจะเกี่ยวข้องกับการสื่อสารหรือส่งข้อมูลระหว่างโปรเซสเหล่านั้น ในอนาคตอาจจะมีโมเดลที่ดีกว่านี้ คือใกล้เคียงกับโครงสร้างของโปรแกรมแบบขนานมากยิ่งขึ้น

2.4 ระบบการจัดการงานแบบขนาน

2.4.1 กล่าวทั่วไป

การประมวลผลงานแบบขนานด้วยเครื่อง SuperComputers นั้น ถึงแม้ว่าจะมีความรวดเร็ว แต่ก็มีการลงทุนในการดำเนินการที่สูงมาก ในขณะที่ปัจจุบันมีการเชื่อมโยงเครื่องลูกข่ายในระบบเครือข่ายเพื่อความมุ่งหมายในการใช้งานแบบทั่วไปอยู่เป็นจำนวนมาก หากพัฒนาโปรแกรม

ประยุกต์ให้สามารถใช้งานหน่วยประมวลผลของเครื่องลูกข่าย ได้อย่างเต็มประสิทธิภาพก็จะสามารถนำมาใช้ประมวลผลงานแบบขนานได้ โดยมีการลงทุนในเกณฑ์ต่ำและคุ้มค่า จึงมีการพัฒนาระบบการจัดการงานแบบขนานเกิดขึ้นมากมาย ทั้งนี้ขึ้นอยู่กับสภาพแวดล้อมของระบบการประมวลผลแบบกระจาย และความมุ่งหมายในการพัฒนา แต่ก็ยังพิจารณาให้เครื่องที่มีอยู่ในระบบเครือข่ายเดิม สามารถทำงานของตัวเองต่อไปได้อย่างปกติ และไม่ทำให้ประสิทธิภาพเลวลงมากจนเกินไป จึงต้องมีการพัฒนาระบบการจัดการอย่างเหมาะสม อย่างไรก็ตามหลักในการพิจารณามักจะสนใจในเรื่องของการแบ่งปันทรัพยากรระหว่างงานแบบขนาน ซึ่งก็หมายความว่าเป็นการจัดการในระดับของระบบปฏิบัติการ โดยทั่วไปมีเทคนิคที่ได้รับความนิยมมากอยู่ 4 แบบด้วยกันคือ Global Queue, Variable Partitioning, Dynamic Partitioning และ Gang Scheduling

2.4.2 ความเป็นมา

ปัญหาของการจัดการบนเครื่องแบบขนาน มีความแตกต่างกันออกไป ตามความมุ่งหมายและความต้องการของแต่ละคน วัตถุประสงค์ที่แตกต่างกันนี้ทำให้เกิดการพัฒนาการจัดการบนระบบแบบขนาน เป้าหมายหลักของการจัดการก็คือ Job และ Threads สำหรับ Job ก็คือโปรแกรมที่จะต้องทำงานในเครื่องของตัวเอง Job หนึ่งไม่ควรให้สามารถใช้หน่วยความจำของ Job อื่นๆ ได้ Job อาจจะประกอบด้วยหลาย ๆ Threads ซึ่งสามารถจะทำงานไปพร้อม ๆ กันได้ในโปรแกรมแบบขนาน ดังนั้น Threads ก็คืองานย่อยที่ถูกแบ่งออกจาก Job นั่นเอง โดยทั่วไปเราเรียกว่า Task และเรียก Job ว่าโปรเซส

Job แบบขนานประกอบด้วยงานย่อยที่ทำงานตามที่ระบุในโปรแกรมแบบขนาน ตามตารางที่ 2.1 เป็นการแบ่งประเภทของภาษาโปรแกรมแบบขนาน งานย่อยสามารถถูกแบ่งเป็น Static หรือ Dynamic และการสื่อสารข้อมูลระหว่างงานย่อย ๆ นั้น ก็สามารถแบ่งได้เป็น Static หรือ Dynamic เช่นกัน ตัวอย่าง ในภาษาโปรแกรมแบบ Single Program, Multiple Data (SPMD) นั้น จำนวนของงานย่อยจะถูกแบ่งตายตัวเท่ากับจำนวนของหน่วยประมวลผล แต่การสื่อสารข้อมูลระหว่างงานย่อย สามารถเปลี่ยนแปลงได้ในระหว่างการคำนวณ

ตารางที่ 2.2 การแบ่งประเภทของโปรแกรมแบบขนานแบบง่าย ๆ [12]

		Communication Pattern	
		Static	*Dynamic*
Thread creation	*Static*	DAG	SPMD, HPF
	Dynamic	Dataflow Functional	Nested Parallelism Unix fork-join

รูปแบบของเครื่องในระบบเครือข่าย ก็มีผลกระทบที่สำคัญกับการจัดการงานแบบขนานเป็นอย่างมาก เช่น เครื่องแบบ Uniform Access to Shared Memory การจัดการในการหยุดงานย่อย และเริ่มทำงานมันอีกในเครื่องอื่น ๆ สามารถกระทำได้อย่างง่าย ๆ แต่เครื่องแบบที่มี Topology จำกัด จะสามารถทำงานย่อยได้จำกัด เช่น Mesh หรือ Hypercube เท่านั้น การจัดการให้เป็นไปตามที่ต้องการจะต้องพัฒนาโปรแกรมจัดการอีกมาก เป็นต้น

2.4.3 แบบของการจัดการโดยทั่วไป

แบบอื่นๆ ของระบบการจัดการ ซึ่งแต่ละแบบมีเป้าหมายของตัวเองที่แตกต่างกันไป ขึ้นอยู่กับเครื่อง และภาษาที่ใช้งาน [8]

2.4.3.1 Static Scheduling

เพื่อให้เข้าใจทฤษฎี และข้อจำกัดของการจัดการในโปรแกรมแบบขนาน โปรแกรมประยุกต์ถูกจัดการแบบ Directed Acyclic Graph (DAG) ซึ่งแต่ละโหนดจะถูกมองเป็นวงรอบ และถูกจัดงานให้โหนดละหนึ่งงาน งานที่ถูกจัดให้โหนดแรกจะต้องทำงานเสร็จเรียบร้อยก่อนโหนดที่ถูกจัดงานให้สุดท้าย และจะต้องทำงานแรกที่ถูกจัดให้เสร็จก่อน จึงจะเริ่มงานใหม่ได้ เมื่อระบบของเครื่องในเครือข่ายเป็นแบบเดียวกัน จะสามารถเริ่มงานย่อยได้พร้อม ๆ กัน ในบางกรณีการสื่อสารระหว่างอาจทำให้มีการชักช้าในการคำนวณได้ เพราะต้องเสียเวลาในการสื่อสารข้อมูลระหว่างงานย่อย การจัดการแบบนี้จะกำหนดลำดับความเร่งด่วนของแต่ละโหนด โดยขึ้นอยู่กับระยะทางในการเชื่อมต่อ โหนดที่อยู่ใกล้จะมีความเร่งด่วนในการได้รับงานก่อน การจัดการแบบนี้ไม่มีความอ่อนตัว เนื่องจากหากมีการสื่อสารหนาแน่นในระบบเครือข่าย จะทำให้การจัดลำดับไม่เป็นไปตามความเป็นจริง และเสียเวลามากยิ่งขึ้น

2.4.3.2 Scheduling in the Runtime System

ในขณะที่การจัดการแบบ DAG มีการวิจัยกันอย่างกว้างขวาง แต่ส่วนมากแล้วไม่สามารถพัฒนาโปรแกรมที่เหมาะสมกับสภาพแวดล้อมแบบนั้นได้ โปรแกรมเมอร์ส่วนใหญ่จึงเขียนโปรแกรมเข้าไปควบคุมเครื่องในขณะ Runtime ของระบบ

ตัวอย่างของการจัดการแบบนี้ก็คือการจัดการกับงานย่อย โปรแกรมประยุกต์ที่มีการจัดโครงสร้างโดยแบ่งชุดของงานย่อย เช่น ฟังก์ชันการแบ่งงานย่อย การสื่อสารข้อมูลระหว่างงานย่อย และการหยุดงานเหล่านั้น สามารถพัฒนาเป็นฟังก์ชันที่ทำงานในระหว่าง Runtime ของระบบ และมีการตอบสนองเป็นอย่างดี ตัวอย่างอื่น เช่นการใช้ตัวแปรภาษาของโปรแกรมแบบขนานจะทำให้การวนรอบในโปรแกรม (Loop) แบบลำดับน้อยลง เนื่องจากในขณะ Runtime จะมีการแบ่งการวนรอบในโปรแกรมที่ไม่เหมือนกัน ให้กับเครื่องประมวลผลที่ชัดเจนตามความสามารถ จะทำให้ลดรอบในการทำงานแบบวนรอบลงได้ และทำให้เกิดความสมดุลย์ได้ โดยโปรแกรมเมอร์ไม่ต้องกังวลเรื่องนี้อีก

2.4.3.3 Scheduling in the Operating System

การจัดการในขณะ Runtime ของโปรแกรมเมอร์นั้นมีความพอใจในการแก้ปัญหาได้ในระดับหนึ่ง ได้แก่การทำงานของโปรแกรมเดียวจะไม่มีปัญหา แต่เมื่อมีโปรแกรมแบบขนานทำงานพร้อมกันในระบบเดียวกัน มีความจำเป็นต้องพิจารณาในระดับระบบปฏิบัติการ

จริงอยู่ที่การจัดการในระดับระบบปฏิบัติการอาจมีความคล้ายคลึงกับการจัดการในระหว่าง Runtime แต่ในบางปัญหาก็ไม่ชัดเจน และอาจมีปัญหากับระบบปฏิบัติการ ดังนั้นการจัดการในระดับระบบปฏิบัติการ จะมีความอ่อนตัวกว่าเพราะระบบปฏิบัติการจะสามารถทำงานต่าง ๆ ได้ทั้งหมด ได้แก่ Interrupts, Context Switches และ Memory Swapping [9]

ความต้องการมากมายทำให้การจัดการยุ่งยากมาก ตัวอย่างข้อแตกต่างที่สำคัญ เช่น Job ที่ทำงานแตกต่างกัน Job หนึ่งไม่สามารถจะหน่วยความจำหรือพื้นที่การรับ-ส่ง Message ของอีก Job หนึ่งได้ ความยุ่งยากอีกอย่างคือ เครื่องมีแบบที่แตกต่างกันออกไป ทำให้บาง Job ไม่สามารถจัดสรรเครื่องที่มีแบบแตกต่างออกไปได้ เช่นงานย่อย Subcube สามารถทำงานกับเครื่อง Hypercube ได้เท่านั้น ปัญหาอื่น ๆ ได้แก่ ระบบ Message Passing ต่างแบบกัน ไม่สามารถใช้ร่วมกันได้ เครื่องบางแบบไม่สามารถโอนถ่ายงานระหว่างกันได้เป็นต้น

2.4.3.4 Administrative Scheduling

การจัดการแบบสุดท้ายก็คือการตกลงใจโดย ผู้จัดการระบบ ตัวอย่างเช่นโครงการบางอย่างมีความต้องการใช้งานเครื่องที่แน่นอน ในเวลาที่กำหนดเพื่อให้งานสำเร็จตามโครงการ จึงต้องมีการจัดการด้วยผู้จัดการระบบ หรืออาจจะมีการจัดลำดับความเร่งด่วนให้กับงานแบบขนานสั้น ๆ ก่อน งานแบบขนานที่มีขนาดใหญ่มาก หรือการจัดอัตราส่วนให้ระหว่าง Batch Jobs กับ Interactive Jobs [10] เป็นต้น

2.4.4 การจัดการที่นิยมในปัจจุบัน

เพื่อที่จะให้สามารถทำงานหลาย ๆ โปรแกรม ระบบปฏิบัติการตกลงใจให้แต่ละ Job ทำงานบนเครื่องที่ระบุ โดยทั่วไปสามารถกระทำได้สองแบบคือ การใช้ Time Slicing (แต่ละ Job แบ่งกันใช้เครื่องเดียวกัน) และ การใช้ Space Slicing (แต่ละเครื่องถูกจัดให้ Job ทำงานจนกว่าจะทำงานเสร็จ) หรืออาจใช้ผสมกันทั้งสองแบบ Time Slicing อาจไม่สามารถใช้ได้ในบางระบบ หรืออาจเป็นไปไม่ได้ที่จะป้องกันไม่ให้โปรแกรมประยุกต์อื่นเข้ามาใช้งาน และก็ไม่สามารถแน่ใจได้ว่าเครื่องที่ถูกจัดเวลาใช้ให้จะสามารถสนับสนุนงานนั้นได้อย่างเต็มที่ เช่นเดียวกันเครื่องแบบขนานหลายเครื่องได้ถูกนำไปใช้งานแบบลำดับ ที่ไม่เกิดประโยชน์ใด ๆ ดังนั้นการใช้ Partitioning จึงไม่ค่อยมีประโยชน์เท่าไรนัก แต่การจัดสรรทั้งสองแบบได้มีการพัฒนากันอย่างกว้างขวางหลายปีมาแล้ว ตารางที่ 2.3 แสดงการพัฒนาการจัดการทั้งสองแบบ

ตารางที่ 2.3 ตัวอย่างของระบบต่าง ๆ ที่ใช้การผสมผสานการจัดการแบบ Space Slicing และ Time Slicing [8]

			time slicing			
			yes			no
			independent PEs		gang scheduling	
			global queue	local queue		
Space slicing	yes	flexible	Mach	Meiko/timeshare Paragon/service KSR/interactive transputers Tera/streames Chrysalis	Medusa Butterfly@LLNL Meiko/gang Paragon/gang SGI/gang Tera/PB MAXI/gang	IBM SP2, Victor Meiko/batch Paragon/slice KSR/batch 2-level/bottom TRAC, MICROS Amoeba
		structured		NX/2 on iPSC/2 nCUBE	CM-5 Cedar DQT on RWC-1 DHC design	Cray T3D CM-2 PASM hypercubes
	no		IRIX on SGI NYU Ultra Dynix 2-level/top Hydra/C.mmp	StarOS Psyche Elxsi AP1000	MasPar MP2 Alliant FX/8 Chagori on K2	Illiac IV MPP GF11 Warp

สำหรับแบบการจัดการที่ได้รับความนิยมมากที่สุด 4 แบบ ได้แก่ Global Queue, Variable Partitioning, Dynamic Partitioning การจัดการแบบสองระดับ และ Gang Scheduling มี 2 แบบที่ใช้การจัดการแบบ Space Slicing คือ Variable Partitioning, Dynamic Partitioning และมี 2 แบบที่ใช้การจัดการแบบ Time Slicing คือ Global Queue, Gang Scheduling ซึ่งรูปแบบ ขีดความสามารถ และข้อดีข้อเสียดังนี้

2.4.4.1 Global Queue

การจัดการแบบนี้เป็นการจัดการแบบง่าย ๆ ในระดับระบบปฏิบัติการ คือการสั่งให้สำเนาของโปรแกรมทำงานในแต่ละเครื่อง แล้วแบ่งกันใช้ข้อมูล โดยจัดงานย่อยที่พร้อมจะทำงานเอาไว้ในคิวส่วนกลาง เครื่องที่มีความสามารถจะนำงานย่อยแรกที่อยู่ในคิวไปทำการประมวลผลตามห้วงเวลาที่กำหนด จากนั้นก็นำมันกลับเข้าไปในคิวต่อไป การจัดการแบบนี้ใช้กับเครื่องทั่ว ๆ ไป ที่มีระบบ Bus-Based ที่ไม่ใหญ่มากนัก, UMA Shared Memory Machines เช่น Sequent Multiprocessors , SGI Multiprocessor Workstations และ ระบบปฏิบัติการ Mach

ข้อดีของการจัดการแบบนี้ก็คือ มีการจัดการเกี่ยวกับ Load Sharing โดยอัตโนมัติ และไม่มีเครื่องใดว่างในระบบถ้ามีงานย่อยรออยู่ในคิว อย่างไรก็ตามก็มีปัญหาในเรื่องการแย่งกันใช้เครื่องของงานในคิวส่วนกลาง งานจะถูกจัดให้ทำงานบนเครื่องที่แตกต่างกัน ในแต่ละเวลา จึงไม่สามารถเก็บข้อมูลไว้ที่ Local Memory ได้ หน่วยเก็บเก็บข้อมูลชั่วคราว ก็จะถูกล้างข้อมูลทุกครั้งที่มีการจัดการงานใหม่ ผลกระทบนี้ขึ้นอยู่กับความหนาแน่นในการจัดการ ซึ่งโดยปกติตัวจัดการจะพยายามจัดงานย่อยให้ทำงาน และเริ่มงานใหม่บนเครื่องเดียวกันก่อน ปัญหาอีกอย่างหนึ่งก็คือ งานย่อยของโปรแกรมประยุกต์เดียว (Single Application) จะถูกจัดการในลักษณะรวมการด้วย มันจะเป็นการดีถ้างานย่อยนั้นไม่มีความเกี่ยวพันกับงานอื่น แต่ในโปรแกรมแบบขนานส่วนใหญ่แล้วงานย่อยมักจะมีการติดต่อสัมพันธ์กับงานอื่น ถ้ามีอัตราการทำงานร่วมกันสูง และส่วนใหญ่จะไม่สามารถทำงานพร้อมกันได้ จึงต้องมีการจัดการย่อย ๆ เป็นจำนวนมาก จึงเป็นเหตุให้เสียเวลาในการ Context Switches และ เกิด Overhead สูง

2.4.4.2 Variable Partitioning

เป็นการแบ่งเครื่องให้กับแต่ละ Job ได้ทำงานอย่างชัดเจน มีข้อแตกต่างในการจัดแบ่ง Partition โดยทั่วไป 4 ประการคือ **Fixed partitioning** คือการจัดแบ่งแบบตายตัว ขนานของ Partition ไม่สามารถเปลี่ยนแปลงได้ นอกจากการ Reboot ระบบใหม่ **Variable** คือ เมื่อชุดของโหนดถูกจัดให้กับผู้ใช้ตามที่ร้องขอ โดยยอมรับ Job แล้ว ส่วนของ Partition ที่ใหญ่มากสามารถแบ่งแยกย่อยออกไปเพื่อให้ Job เล็ก ๆ ทำงานได้พร้อม ๆ กัน การจัดแบ่งใหม่นี้จะถูกยกเลิกเมื่อ Job ทำงานเสร็จแล้ว **Adaptive** คือการจัดแบ่งให้ Job ที่ยอมรับแล้ว แบบอัตโนมัติด้วยระบบโดยสัมพันธ์กับปริมาณงานในขณะนั้น **Dynamic** คือขนาดที่จัดแบ่งให้สามารถเปลี่ยนแปลงได้ในระหว่าง Runtime ขึ้นอยู่กับความเปลี่ยนแปลงของความต้องการ และปริมาณงาน การจัดแบ่งที่เป็นที่นิยมมากที่สุดได้แก่ Variable และ Dynamic Partitioning

ตารางที่ 2.4 การแบ่งประเภทของการ Partitioning [16]

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

Variable Partitoning นิยมใช้งานกันมากบนระบบ Distributed Memory Machines และใช้กับเครื่อง Intel และ nCUBE Hypercubes, IBM SP2, Intel Paragon, Meiko CS-2 และ Cray T3D ไม่มีข้อจำกัดในการใช้งานหน่วยความจำ และปัญหาในการ Shared Address Space ของเครื่องเหมือนกับ Global Queue ข้อดีในการทำงานของ Job หนึ่ง บนเครื่องที่จัดแบ่งให้แบบ Variable Partitoning คือได้เครื่องทำงานอย่างเหมาะสม โปรแกรมสามารถควบคุมข้อมูลในหน่วยความจำของเครื่องที่มันทำงาน ไม่ต้องมีหน่วยเก็บข้อมูลสำรอง และไม่ทำให้ระบบเกิด Overhead ทั้งนี้ก็ขึ้นอยู่กับแบบของการเชื่อมต่อของระบบนั้น ๆ ด้วย

แน่นอน Variable Partitoning ก็มีข้อเสียเหมือนกัน ข้อเสียเปรียบที่เป็นไปได้ก็คือความไม่ลงตัวระหว่างหน่วยประมวลผลที่มีความสามารถ กับความต้องการของผู้ใช้ มีปัญหาที่เห็นได้อย่างชัดเจน 2 อย่าง ได้แก่ หนึ่งจะเกิดการแตกกระจาย (Fragmentation) ซึ่งจะเกิดขึ้นเมื่อหน่วยประมวลผลที่มีความสามารถมีไม่เพียงพอกับความต้องการของ Job ที่ยอมรับแล้ว ดังนั้นหน่วยประมวลจึงไม่มีการว่าง ปัญหาที่สองคือ Job ที่ยอมรับแล้วอาจต้องเข้าคิวคอยหน่วยประมวลผลเป็นเวลานาน โดยเฉพาะ Job ที่มีขนาดใหญ่มากซึ่งต้องการหน่วยประมวลผลทั้งหมดของระบบทำให้ไม่สามารถจัดให้ Job อื่นทำงานได้ เช่นเดียวกันหน่วยประมวลผลที่ว่างชั่วขณะเพื่อคอยการทำงานกับ I/O ก็จะไม่สามารถให้บริการแก่ Job อื่นๆ ได้ ดังนั้น Variable Partitoning จึงเหมาะกับ Batch Processing มากกว่า Interactive Work [12]

2.4.4.3 Dynamic Partitioning ด้วยการจัดการสองระดับ

หนทางหนึ่งเพื่อที่จะลดเวลาในการรอคอยของ Job ที่เข้าคิวอยู่ก็คือ ป้องกันไม่ให้มี Job ผูกขาดการใช้หน่วยประมวลผลมากจนเกินไป เมื่อระบบมีปริมาณงานมาก ซึ่งทำได้โดย Adaptive Partitioning โดยการจัดหน่วยประมวลผลจำนวนหนึ่ง ให้กับ Job หนึ่งแบบตายตัวจนกว่า Job นั้นจะจบการทำงาน ไม่มีการเปลี่ยนแปลงแม้ว่าปริมาณงานของระบบจะเปลี่ยนไปอย่างไรก็ไม่สามารถจัดให้กับ Job ใหม่ได้

ดังนั้นการจัดหน่วยประมวลผลในระหว่างการประมวลผล สามารถกระทำได้ด้วย Dynamic Partitioning มันสามารถสนับสนุนความต้องการของโปรแกรมประยุกต์ ด้วยการพัฒนาโปรแกรมให้สามารถบ่งบอกความต้องการ และจัดสรรให้ตามต้องการ ทำได้โดยใช้ Workpile Model ซึ่งมีการทำงานคือจัดงานย่อยโดยไม่ต้องจัดเรียงรวมไว้ใน Pile แล้วทำการคำนวณหากลุ่มของงานย่อยจาก Work Pile เรียกกลุ่มนี้ว่า Worker Threads ดังนั้นจะมีเพียงงานย่อยเดียวเท่านั้นจาก Work Pile ได้รับการประมวลผลจากหน่วยประมวลผลเดียว การตรวจจับการคำนวณ (Worker) และยอมให้จัดปรับจำนวนของหน่วยประมวลผลได้โดยการเปลี่ยนจำนวนของ Workers ผลลัพธ์นี้เป็นการจัดการในสองระดับ ในระดับระบบปฏิบัติการคือการจัดสรรหน่วยประมวลผลให้กับ Job ในขณะที่โปรแกรมประยุกต์จะควบคุมและจัดงานย่อย ๆ ให้กับหน่วยประมวลผลเหล่านั้น

Dynamic Partitioning ด้วยการจัดการสองระดับ เป็นที่นิยมมากโดยการพิจารณาปริมาณงานประกอบ มีตัวแปรที่เกี่ยวข้องดังนี้

- ไม่มีการแตกกระจายของทรัพยากรที่จะจัดสรรให้
- ไม่มีการ Overhead ในการ Context Switching ยกเว้นมีการสร้างโปรเซสขึ้นใหม่เมื่อมีปริมาณงานเปลี่ยนแปลง และการจัดการในระดับที่สองของโปรแกรมประยุกต์กำหนดว่ามี Overhead น้อยมาก
- ไม่มีการสูญเสียวงรอบของ CPU ในขณะที่คอยการ Synchronization และงานย่อยที่ถูกกั้นไว้เกิด Overhead เพียงเล็กน้อย
- การกำหนดทรัพยากรให้แต่ละ Job จะลดลงโดยอัตโนมัติภายใต้เงื่อนไขปริมาณงานที่เพิ่มขึ้น ทำให้ประสิทธิภาพดีขึ้น

อย่างไรก็ตาม Dynamic Partitioning ก็มีข้อด้อยคือไม่สนับสนุนโปรแกรมที่เป็นที่นิยมเช่น SPMD (ซึ่งใช้โดย MPI) และ Data Parallel (ซึ่งใช้โดย HPF) แต่มันก็สามารถนำไปใช้ในระบบคิวได้ถ้าหน่วยประมวลผลไม่เพียงพอ ซึ่งจะเกิด Overhead ในการ Repartitioning ขึ้นได้ รวมถึงการปรับเปลี่ยนหน่วยประมวลผลจากการป้องกัน เพื่อให้สามารถสนับสนุนงานอื่นได้

Dynamic Partitioning ต้องการการสัมพันธ์กันระหว่างระบบปฏิบัติการ กับตัวจัดการของโปรแกรมประยุกต์ด้วย ซึ่งจะทำให้การควบคุมความเปลี่ยนแปลงของหน่วยประมวลผลมีประสิทธิภาพมากขึ้น [15] ตัวอย่างเช่น ถ้าหน่วยประมวลผลไม่สามารถสนับสนุนโปรแกรมประยุกต์ อาจเกิดการ Deadlock ขึ้นได้เมื่องานย่อยทำงานบนหน่วยประมวลผลที่ป้องกันไว้ การป้องกันนี้อาจเกิดจากที่โปรแกรมประยุกต์หยุดเพื่อกระทำการอย่างใดอย่างหนึ่ง นั่นก็หมายความว่าระบบปฏิบัติการและการพัฒนาโปรแกรมจะต้องออกแบบไปพร้อม ๆ กัน ซึ่งทำให้เกิดข้อจำกัดในเรื่องความสามารถนำไปใช้งานในสภาพแวดล้อมอื่น แต่ก็อาจแก้ปัญหานี้ได้โดยการเปลี่ยนแบบของหน่วยประมวลผลในโปรแกรมได้

2.4.4.4 Gang Scheduling

เป็นการจัดการที่ให้เวลาตอบสนองที่ดีสำหรับการจัดการผ่าน Time Slicing อย่างไรก็ตามถ้าไม่จัดการในระดับเดียวกันกับ Global Queue แล้ว อาจจะทำให้ไม่มีประสิทธิภาพเลยก็ได้ ดังนั้นการ Context Switching ควรจะสามารถกระทำข้ามหน่วยประมวลผลได้ ดังนั้นงานย่อยทั้งหมดของ Job จะสามารถทำงานได้พร้อมกัน นอกจากนี้งานย่อยสามารถจัดหน่วยประมวลผลให้แบบตายตัวได้ นั่นคือจะยอมให้ใช้หน่วยความจำของเครื่องได้ ทำให้มีสถานะมั่นคงยิ่งขึ้น การแก้ปัญหาแบบนี้จะสำเร็จได้ด้วยการเขียนโปรแกรม ความจริงการจัดการแบบนี้จะไม่มีความสัมพันธ์กันระหว่างระบบปฏิบัติการกับโปรแกรมประยุกต์ ซึ่งจะถูกใช้ในทางการค้าได้แก่ CM-5 จาก Thinking Machines, IRIX system on SGI multiprocessors, Intel Paragon และ Meiko CS-2 [13] เป็นต้น

สิ่งที่น่าสนใจของ Gang Scheduling ตามที่สังเกตคือการจัดการในระดับเดียวกันเท่านั้นที่ต้องการ ถ้างานย่อยของ Job มีการกระทำบ่อยครั้ง ดังนั้นอัตราของการกระทำดังกล่าวสามารถจัดเข้าไปในกลุ่มของ Gang ได้ ส่วนอื่นๆ ได้แก่ Coscheduling เป็นการจัดการส่วนย่อยของ Gang Scheduling เมื่อไม่สามารถจัดการงานย่อยทั้งหมดได้ในครั้งเดียวกัน และอีกส่วนหนึ่งคือ Family Scheduling ซึ่งยอมให้งานย่อยมีมากกว่าหน่วยประมวลผล และใช้การจัดการระดับที่สองเป็น Internal Time Slicing

Gang Scheduling ไม่ได้ทำให้ประสิทธิภาพของระบบโดยรวมดีขึ้น (แม้ว่าจะจัดการกับ Job ได้ดีก็ตาม) ปัญหาเกิดจากมี Overhead ในการ Context Switching และอาจมีการแตกกระจายของหน่วยประมวลผล อย่างไรก็ตามขึ้นอยู่กับ Time Slicing ซึ่งจะมีผลกระทบต่อการให้บริการ Job น้อยกว่า Variable Partitioning แต่จากการศึกษาส่วนใหญ่พบว่า Gang Scheduling มีประสิทธิภาพใกล้เคียงกับ Dynamic Partitioning

2.5 หลักการอำนวยการยิง

หลักการอำนวยการยิงที่ใช้ในการพัฒนาโปรแกรมแบบขนานนี้ เป็นหลักการเบื้องต้น ที่ไม่มีการคำนวณในรายละเอียดที่ยุ่งยากมากนัก เนื่องจากจะทำให้ตัวโปรแกรมมีความยาวมากจนเกินความจำเป็น และอาจไม่สามารถวัดประสิทธิภาพได้อย่างถูกต้อง หากการคำนวณหลัก ๆ สามารถทำงานได้อย่างถูกต้องแล้ว การนำตัวแก้ต่าง ๆ เพิ่มเติมเข้าไปก็จะทำได้ง่ายยิ่งขึ้น และไม่เป็นปัญหาต่อตัวโปรแกรมหลัก

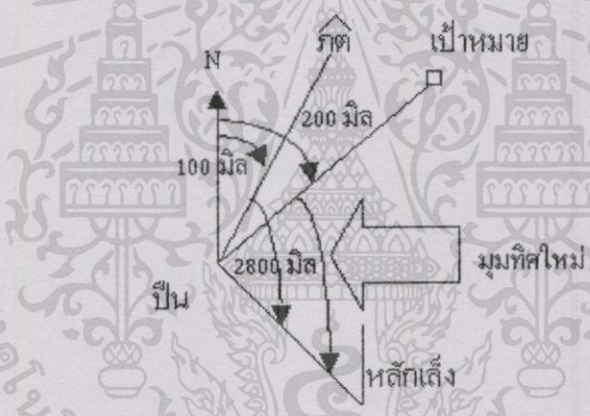
ตามที่กล่าวแล้วข้างต้นว่า อาวุธปืนใหญ่สนามนั้นใช้สนับสนุนการโจมตีต่อเป้าหมาย ที่อยู่ห่างไกลเกินกว่าอาวุธประจำหน่วยจะโจมตีได้ และใช้เพิ่มอำนาจการโจมตีของอาวุธยิงเล็งตรงประจำหน่วย โดยยิงมาจากระยะที่ห่างไกลจากเป้าหมาย ดังนั้นจึงไม่สามารถมองเห็นเป้าหมายและไม่สามารถยิง โดยวิธีเล็งตรงไปยังเป้าหมายได้ จึงต้องใช้การยิงโดยวิธีเล็งจำลอง การยิงเล็ง

จำลองจะต้องมีหลักฐานที่นำมาตั้งให้กับอาวุธและกระสุน หลักฐานนี้จะทำให้ลูกกระสุนไประเบิดณ เป้าหมาย ซึ่งมีหลักฐานยิงที่ต้องการสำหรับตั้งปืนดังนี้

2.5.1 มุมทิศ

ปกติจะไม่ใช้ค่ามุมภาคเป็นค่าตั้งทิศทางในการตั้งปืน แต่จะใช้วิธีการจำลองทิศ โดยตั้งปืนครั้งแรก จะตั้งด้วยมุมภาคทิศทางยิงของปืนไปยังกึ่งกลางเขตปฏิบัติการ เรียกว่า "มุมภาคทิศทางยิง" แล้วจำลองทิศทางนั้นไว้ด้วย มุมทิศ 2800 มิลเลียม มุมทิศจากปืนไปยังเป้าหมายจะเปลี่ยนไปตามพิกัดเป้าหมายที่เกิดขึ้น เมื่อเราจำลองทิศที่มุมภาคทิศทางยิงแล้ว จะคำนวณมุมทิศจากปืนไปยังเป้าหมายใหม่ดังนี้

- มุมทิศจากปืนไปยังเป้าหมาย = (มุมภาคทิศทางยิงเริ่มแรก + มุมจำลองทิศ) — มุมภาคทิศทางยิงใหม่ [2]



รูปที่ 2.20 การคำนวณหาค่ามุมทิศ

จากรูปที่ 2.20 กำหนดให้

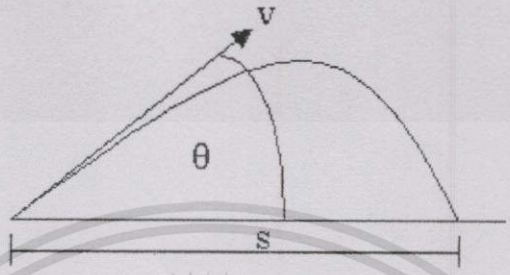
- มุมภาคทิศทางยิงเริ่มแรก (ภต)	=	100	มิลเลียม
- มุมจำลองทิศ	=	2800	มิลเลียม
- มุมภาคเป้าหมายใหม่	=	200	มิลเลียม
ดังนั้น มุมทิศ = (100+2800) — 200	=	2700	มิลเลียม

2.5.2 มุมยิง

ในการคำนวณจริง ๆ แล้ว จะต้องมีการพิจารณาผลกระทบหลายอย่าง เช่น มุมพื้นที่ยิง มุมพื้นที่เพิ่มเติม แตกต่างความสูงระหว่างปืนและเป้าหมาย ฯลฯ การคำนวณดังกล่าวจะต้องเกี่ยวข้องกับตารางยิง ซึ่งเป็นค่ามุมสูงที่ได้จากการทดลองยิงจริงในสภาพมาตรฐาน [1] ซึ่งจะเกี่ยวข้องกับ

ฐานข้อมูล และหากทำการคำนวณตามหลักการทั้งหมด จะทำให้ตัวโปรแกรมยาวจนเกินความจำเป็น และอาจทำการทดลองไม่ได้ จึงได้กำหนดสูตรในการคำนวณแบบง่าย ๆ ดังนี้

- มุมยิง = $\frac{1}{2}*\sin^{-1}(s*g/v^2)$



รูปที่ 2.21 สูตรที่ใช้เพื่อทดลองหาค่าของมุมยิง

เมื่อ S = ระยะจากปืนถึงเป้าหมาย

g = ค่าความโน้มถ่วงของโลก

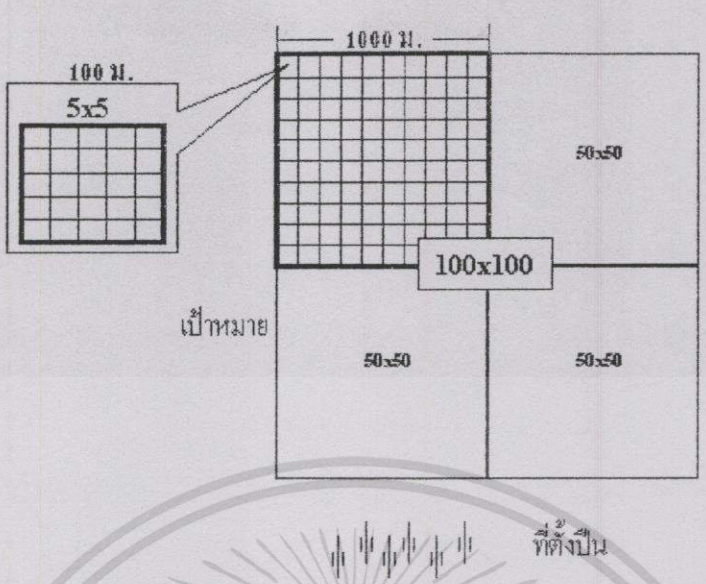
v = ค่าความเร็วต้นปากลำกล้องปืน = 550 m/s

2.6 วิธีดำเนินการ

สำหรับโปรแกรมแบบขนานด้วยระบบ PVM ที่ได้เขียนขึ้นเพื่อใช้ในการคำนวณหาหลักฐานยิงปืนใหญ่นี้ ได้ใช้โมเดลแบบ Crowd Computation ซึ่งจะมีโมเดลทั้งในแบบ Master-Slave และ Node Only ผสมกันอยู่ กล่าวคือ เขียนเป็นโปรแกรมเดียวแบบ Node Only แต่แบ่งงานแบบ Master-Slave แต่ละ Slave เป็นอิสระต่อกัน ไม่ต้องส่งผลลัพธ์ให้กันเพื่อคำนวณต่อ ซึ่งสามารถคำนวณเสร็จสมบูรณ์ได้ในแต่ละ Slave เอง

การคำนวณเป้าหมายตามพื้นที่ที่ได้รับมอบภารกิจให้กับปืนใหญ่ ได้กำหนดให้ทำการคำนวณตามภารกิจการยิงกวาดเป็นเขต ซึ่งปืนใหญ่จะต้องนำพื้นที่ที่ได้รับมอบให้ทำการยิง มาคำนวณหาจำนวนเป้าหมาย เพื่อทำการยิงให้ครอบคลุมเป้าหมาย เพื่อให้เกิดความเข้าใจแนวคิดการคำนวณเป้าหมายให้กับปืนใหญ่ จึงได้สมมุติให้ปืนใหญ่รับภารกิจยิงให้ครอบคลุมพื้นที่เป้าหมาย 4 ตารางกิโลเมตร (2x2) และกำหนดให้ตำบลระเบิดมีเส้นผ่าศูนย์กลาง 20 เมตร

ดังนั้นใน 100x100 ม. ปืนหนึ่งกระบอกจะมีเป้าหมายเท่ากับ 25 เป้าหมาย (100 ม./ตำบลระเบิด) และ 1000x1000 ม. จะมีเป้าหมายเท่ากับ 50x50 เป้าหมาย/กระบอก ดังนั้นในพื้นที่รับผิดชอบ 4 ตารางกิโลเมตร จึงมีเป้าหมายเท่ากับ 100x100 เป้าหมาย/กระบอก



รูปที่ 2.22 การแบ่งเป้าหมายในภารกิจยิงกวาดเป็นเขต

ดังนั้นสามารถเขียนความสัมพันธ์ระหว่าง ที่ตั้งปืนกับเป้าหมายได้ดังนี้

ปืน 1 [g 1] =[พิกัดตะวันออก พิกัดเหนือ] ->

$$\left[\begin{matrix}\textit{พิกัดตะวันออกเป้าหมาย}\\ 100x100\end{matrix}\right], \left[\begin{matrix}\textit{พิกัดเหนือเป้าหมาย}\\ 100x100\end{matrix}\right]$$

:

:

ปืน 2 [g 2] =[พิกัดตะวันออก พิกัดเหนือ] ->

$$\left[\begin{matrix}\textit{พิกัดตะวันออกเป้าหมาย}\\ 100x100\end{matrix}\right], \left[\begin{matrix}\textit{พิกัดเหนือเป้าหมาย}\\ 100x100\end{matrix}\right]$$

:

:

ปืน 3 [g 3] =[พิกัดตะวันออก พิกัดเหนือ] ->

$$\left[\begin{matrix}\textit{พิกัดตะวันออกเป้าหมาย}\\ 100x100\end{matrix}\right], \left[\begin{matrix}\textit{พิกัดเหนือเป้าหมาย}\\ 100x100\end{matrix}\right]$$

:

:

<u>หมายเหตุ</u> -> คือ การคำนวณมุมทิศ และ มุมยิงของปืนแต่ละกระบอกต่อเป้าหมายทุกเป้าหมาย ผลลัพธ์ที่ได้ จะเก็บเป็นอาร์เรย์ลักษณะคล้ายเมตริกซ์ดังนี้.-

$$\text{ปืน 1} = \begin{bmatrix} \textit{มุมทิศ} \\ 100x100 \end{bmatrix}, \begin{bmatrix} \textit{มุมยิง} \\ 100x100 \end{bmatrix}$$

:

:

:

$$\text{ปืน 2} = \begin{bmatrix} \textit{มุมทิศ} \\ 100x100 \end{bmatrix}, \begin{bmatrix} \textit{มุมยิง} \\ 100x100 \end{bmatrix}$$

:

:

:

<u>สรุป</u> การทำงานได้ดังนี้

1) กำหนดให้

- ที่ตั้งปืนเป็นอาร์เรย์หนึ่งมิติดังนี้ ปืน 1 = [g1], ปืน 2 = [g2], ... ปืน 5 = [g5]
- เป้าหมายเป็นอาร์เรย์หนึ่งมิติขนาด 100x100 เป้าหมาย1 = [T1], ... เป้าหมาย5 = [T5]

2) การคำนวณหามุมทิศ

- ใช้มุมภาคทิศทางยิง = 100 มิลเลียม
- มุมจำลองทิศ = 2800 มิลเลียม
- มุมภาคปืน-เป้าหมาย (FT)
  - * แตกต่างเหนือ = พิกัดเหนือของเป้าหมาย-พิกัดเหนือของที่ตั้งปืน = ΔN
  - * แตกต่างตะวันออก = พิกัดตะวันออกของเป้าหมาย-พิกัดตะวันออกของที่ตั้งปืน $= \Delta E$
  - * ระยะเป้าหมาย (R) = $\sqrt{(\Delta E^2 + \Delta N^2)}$
  - * มุมภาคปืน-เป้าหมาย

    กรณี $\Delta E > 0$, FT = $\cos^{-1}(\Delta N)/R$

    กรณี $\Delta E > 0$, FT = 2II-$\cos^{-1}(\Delta N)/R$
- ดังนั้น มุมทิศ = (100 + 2800) - FT

3) การคำนวณหามุมยิง

- กำหนดให้ค่าความเร็วต้นปากลำกล้องปืน 550 m/s
- มุมยิง = 1/2 $\sin^{-1}$ (s*g/2v^2)

สำหรับการพัฒนาระบบการจัดการของ PVM ได้พัฒนาโปรแกรมคำนวณหาหลักฐานยิงโดยพิจารณาว่ามีเครื่องในระบบเท่าไร แล้วแบ่งงานให้ทุกเครื่อง โดยแบ่งเป้าหมายที่ได้รับทั้งหมดเฉลี่ยให้กับแต่ละเครื่องจำนวนเท่ากัน โดยส่งพิกัดปืนทุกกระบอกให้กับทุกเครื่องไปคำนวณหามุมทิศ และมุมยิงต่อเป้าหมายที่ได้รับทั้งหมด จากนั้นส่งผลการคำนวณให้เครื่องหลักรวบรวมและแสดงผล เนื่องจากเครื่องที่ทำการทดลอง มีความแตกต่างกันอยู่บ้าง ทำให้ความสามารถในการคำนวณหาหลักฐานยิงของแต่ละเครื่องไม่เท่ากัน และเมื่อได้ความสามารถในการคำนวณหาหลักฐานยิง (จำนวนเป้าหมาย/วินาที) แล้ว ก็นำความเร็วที่ได้ไปปรับในโปรแกรมใหม่ เพื่อทำการคำนวณแบ่งเป้าหมายให้กับแต่ละเครื่องมากน้อย ตามความสามารถของเครื่อง ทั้งนี้เพื่อให้ทุกเครื่องเริ่มและจบการคำนวณหาหลักฐานยิงได้ในเวลาใกล้เคียงกัน ทำให้ลดเวลาในการคำนวณลงได้ มีประสิทธิภาพมากขึ้น

เนื่องจากปัจจุบันระบบ PVM มีการจัดการงานแบบง่าย ๆ กล่าวคือ เมื่อมีการแบ่งงานแบบขนานออกเป็นงานย่อย ๆ PVM ก็จะตรวจสอบดูว่ามีเครื่องในระบบอยู่กี่เครื่อง แล้วก็ทำการแบ่งงานย่อยให้ในปริมาณที่เท่ากันเรียงตามลำดับ หากมีเศษเกินเครื่องที่เป็นแม่ข่ายก็จะรับมาคำนวณเอง การรวบรวมผลลัพธ์ของเครื่องหลักก็จะต้องรอผลลัพธ์การคำนวณของแต่ละเครื่องในระบบ ถ้าเครื่องใดคำนวณหาหลักฐานยิงได้ช้า ก็จะทำให้คำนวณเสร็จช้าและส่งผลลัพธ์ช้าตามไปด้วย ทำให้การคำนวณงานในการหาหลักฐานยิงของปืนใหญ่ของทุกเครื่องโดยรวมช้าไปด้วย

2.7 การทดลอง

2.7.1 การจัดตั้งระบบเครือข่าย

ได้ทำการเชื่อมต่อคอมพิวเตอร์เป็นระบบเครือข่ายจำนวน 4 เครื่อง เชื่อมต่อด้วยสาย UTP ด้วย EtherFast 10/100 LAN Card แบบเดียวกันทั้ง 4 เครื่อง ต่อผ่าน Hub 8 Port ความเร็ว 10/100 เช่นกัน มีคุณลักษณะของแต่ละเครื่องดังนี้

- เครื่องคอมพิวเตอร์แบบเพนเทียมทู ความเร็ว 333 Mhz. หน่วยความจำ 64 MB. ติดตั้งระบบปฏิบัติการ LINUX Ziif v.0.99 และโปรแกรม PVM รุ่นที่ 3.4.3 (ติดตั้ง XPVM บน X — Windows ด้วย) จัดทำเป็นเครื่องแม่ข่าย (ให้ชื่อเครื่องว่า tepa)
- เครื่องคอมพิวเตอร์แบบ MMX ความเร็ว 233 Mhz. หน่วยความจำ 64 MB. ติดตั้งระบบปฏิบัติการ LINUX Ziif v.0.99 และโปรแกรม PVM รุ่นที่ 3.4.3 (ติดตั้ง XPVM บน X — Windows ด้วย) จัดทำเป็นเครื่องลูกข่ายเครื่องที่ 1 (ให้ชื่อเครื่องว่า tepb)
- เครื่องคอมพิวเตอร์แบบเพนเทียมทู ความเร็ว 333 Mhz. หน่วยความจำ 64 MB. ติดตั้งระบบปฏิบัติการ LINUX Ziif v.0.99 และโปรแกรม PVM รุ่นที่ 3.4.3 (ติดตั้ง XPVM บน X — Windows ด้วย) จัดทำเป็นเครื่องลูกข่ายเครื่องที่ 2 (ให้ชื่อเครื่องว่า tepc)

- เครื่องคอมพิวเตอร์แบบ MMX ความเร็ว 200 Mhz. หน่วยความจำ 48 MB. ติดตั้งระบบปฏิบัติการ LINUX Ziif v.0.99 และโปรแกรม PVM รุ่นที่ 3.4.3 (ติดตั้ง XPVM บน X — Windows ด้วย) จัดทำเป็นเครื่องลูกข่ายเครื่องที่ 3 (ให้ชื่อเครื่องว่า tepd)

2.8 การออกแบบโปรแกรม

2.8.1 โปรแกรมคำนวณหาหลักฐานยิงแบบขนาน

ทำการเขียนโปรแกรมโดยมีลำดับการทำงานดังนี้.-

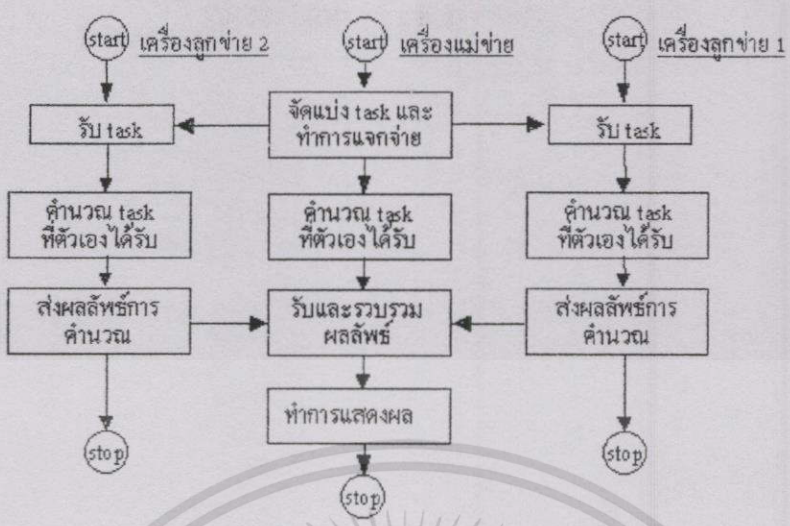
2.8.1.1 สั่งให้เครื่องในระบบ PVM ทำงานพร้อมกันโดยทำงานเบื้องหลังเครื่องทุก ๆ เครื่อง โดยสั่งจากเครื่องหลัก

2.8.1.2 นำจำนวนเป้าหมายที่ผู้ใช้กำหนด มาคำนวณแบ่งเป้าหมายให้กับเครื่องแต่ละเครื่อง โดยเฉลี่ยจากความเร็วของเครื่องหรือ Speed ที่ PVM กำหนดให้แต่ละเครื่อง ตัวอย่างเช่น

- เป้าหมาย = 100x100 = 10,000 เป้าหมาย
- มีเครื่องในระบบ PVM จำนวน 4 เครื่อง
- ความเร็วของเครื่อง 1000 เป็นค่าโดยปริยาย ซึ่งสัมพันธ์กับความสามารถในการคำนวณของเครื่อง เปรียบเทียบกับ Hosts อื่น ๆ ในระบบ ใช้สำหรับ Load Balancing เพื่อให้ PVM เลือก Hosts ที่จะทำการ Spawn Task มีค่าได้ตั้งแต่ 1 ถึง 1000,000
- ดังนั้นแต่ละเครื่องจะได้รับการแบ่งเป้าหมายในการคำนวณเท่ากับ 10,000x1000/4000 = 2,500 เป้าหมาย กรณีไม่ลงตัวมีเศษจะปัดไปให้เครื่องหลัก ซึ่งถือว่ามีประสิทธิภาพสูงที่สุดในเครือข่ายทดลอง

2.8.1.3 ทำการส่งพิกัดที่ตั้งปืน และเป้าหมาย ให้แต่ละเครื่องทำการคำนวณ และส่งผลกลับมาให้เครื่องหลักรวมรวมและแสดงผล

2.8.1.4 ทำการทดลองโดยเปลี่ยนจำนวนปืน และเป้าหมาย



รูปที่ 2.23 ผังของโปรแกรมแบบขนานบนระบบ PVM

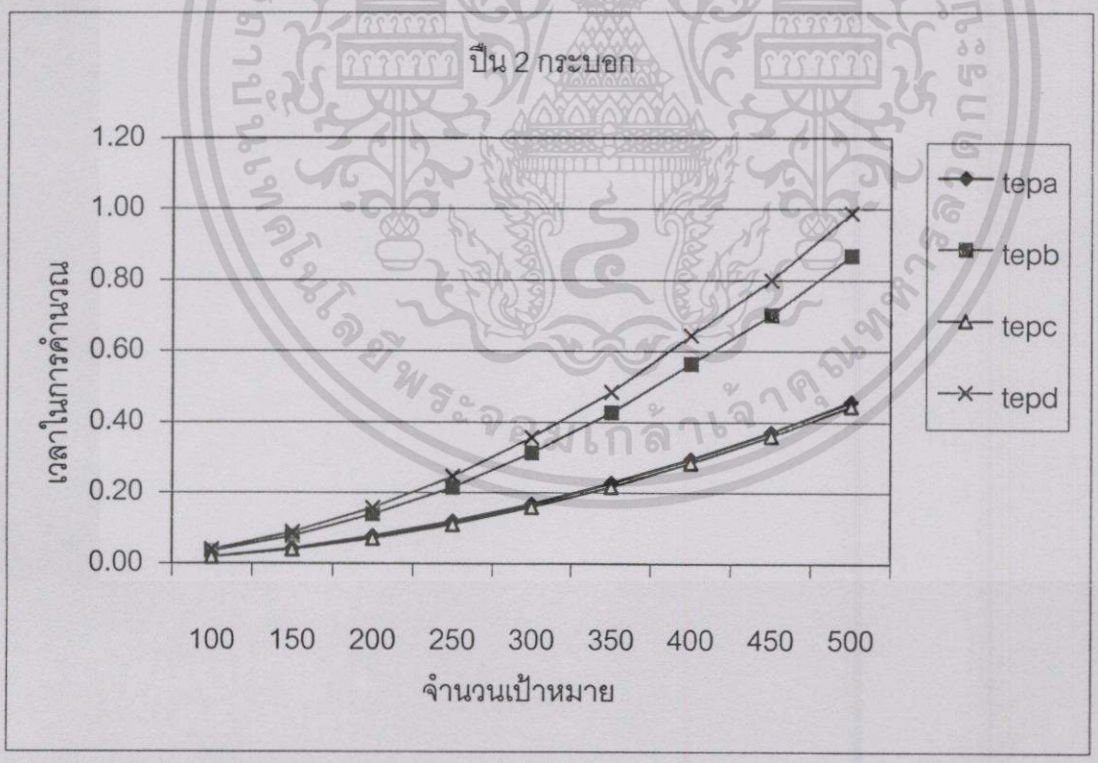
```
{Scheduling Plan for PVM System in Light-Weight Gun Fire-Directions Calculations Algorithm}
{เริ่มต้นการทำงาน}
mytid = pvm_mytid ();
myparent = pvm_parent ();
{ถ้าเป็นเครื่องหลัก}
if (myparent = PvmNoParent)
        {รับค่าพารามิเตอร์}
        จำนวนปืน = atoi (argv [1]);
        จำนวนเป้าหมาย = atoi (argv [2]);
        {ตั้งค่าความเร็วให้แต่ละเครื่องตามผลการทดลอง}
        setspeed [i];
        {คำนวณจำนวนเป้าหมายให้แต่ละเครื่อง}
        for i = 0 to Numworker -1
                target [i] = All target * speed [i] / All speed
        endfor
        {สุ่มค่าพิกัดที่ตั้งปืน}
        for i = 0 to Numgun
                g [i] = 40000 + rand () % 1000;  {พิกัดตะวันออกปืน}
                g [i + 1] = 60000 + rand () % 1000;  {พิกัดเหนือปืน}
        endfor
```

```
        {Start up Worker}
        info = pvm_spawn ();
        {Send Gun Location and Target}
        pvm_initsend ();
        pvm_pkint (NumTarget);
        pvm_pkint (NumGun);
        pvm_pkfloat (Gun Location);
        {รับค่าหลักฐานยิง มุมทิศ, มุมยิง}
        for k = 0 to Numworker - 1
                buf = pvm_recv (-1, -1);
                pvm_upkfloat (Target Location);
                pvm_upkfloat (มุมทิศ, มุมยิง);
                print Result;
        endfor
else    {กรณีเป็นเครื่องรองในเครือข่าย}
        {Receive Gun Location and Target}
        pvm_recv (Parent);
        pvm_upkint (NumTarget);
        pvm_upkint (NumGun);
        pvm_upkfloat (Gun Location);
        {สุ่มค่าพิกัดเป้าหมาย}
        InitBlock (NumTarget);
        {คำนวณ มุมทิศ, มุมยิง ระหว่างปืนกับเป้าหมาย}
        BlockMult ();
        {ส่งค่าหลักฐานยิง มุมทิศ, มุมยิง}
        pvm_initsend ();
        pvm_pkfloat (Target Location);
        pvm_pkfloat (มุมทิศ, มุมยิง);
endif
print Calculation Time;
```

รูปที่ 2.24 อัลกอริทึมโปรแกรมพัฒนาการจัดการ

ตารางที่ 2.5 เปรียบเทียบเวลาการคำนวณหาหลักฐานยิงปืนใหญ่ ปืน 2 กระบอก

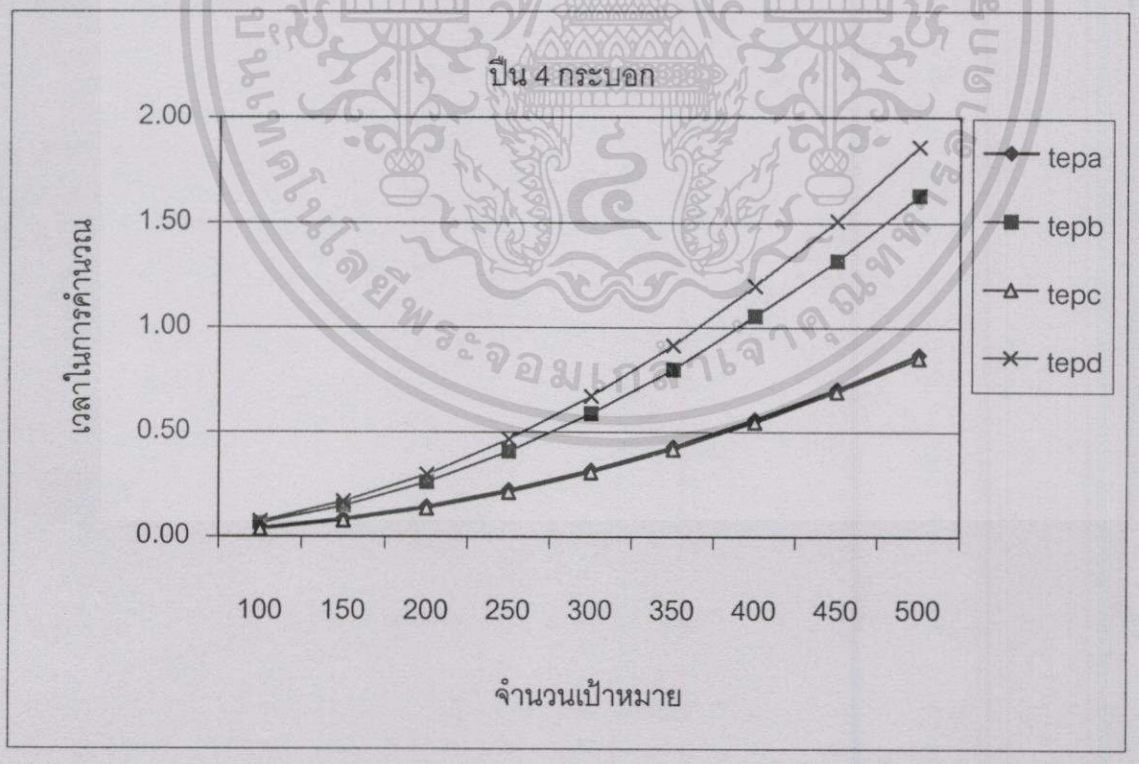
จำนวนเป้าหมาย	tepa	tepb	tepc	tepd	เวลารวม
100x100	0.019002	0.034239	0.017753	0.039513	0.122548
150x150	0.042025	0.077744	0.039810	0.088756	0.261834
200x200	0.078593	0.138612	0.070677	0.157462	0.462556
250x250	0.119146	0.216050	0.110454	0.246425	0.691634
300x300	0.168179	0.312245	0.159033	0.356765	0.975577
350x350	0.227180	0.427091	0.216500	0.484798	1.305386
400x400	0.294953	0.564093	0.283995	0.643468	1.716308
450x450	0.371232	0.702740	0.360078	0.800699	2.129926
500x500	0.457448	0.868091	0.444746	0.988954	2.631272



รูปที่ 2.25 เวลาการคำนวณหาหลักฐานยิงปืนใหญ่ ปืน 2 กระบอก

ตารางที่ 2.6 เปรียบเทียบเวลาการคำนวณหาหลักฐานยิงปืนใหญ่ ปืน 4 กระบอก

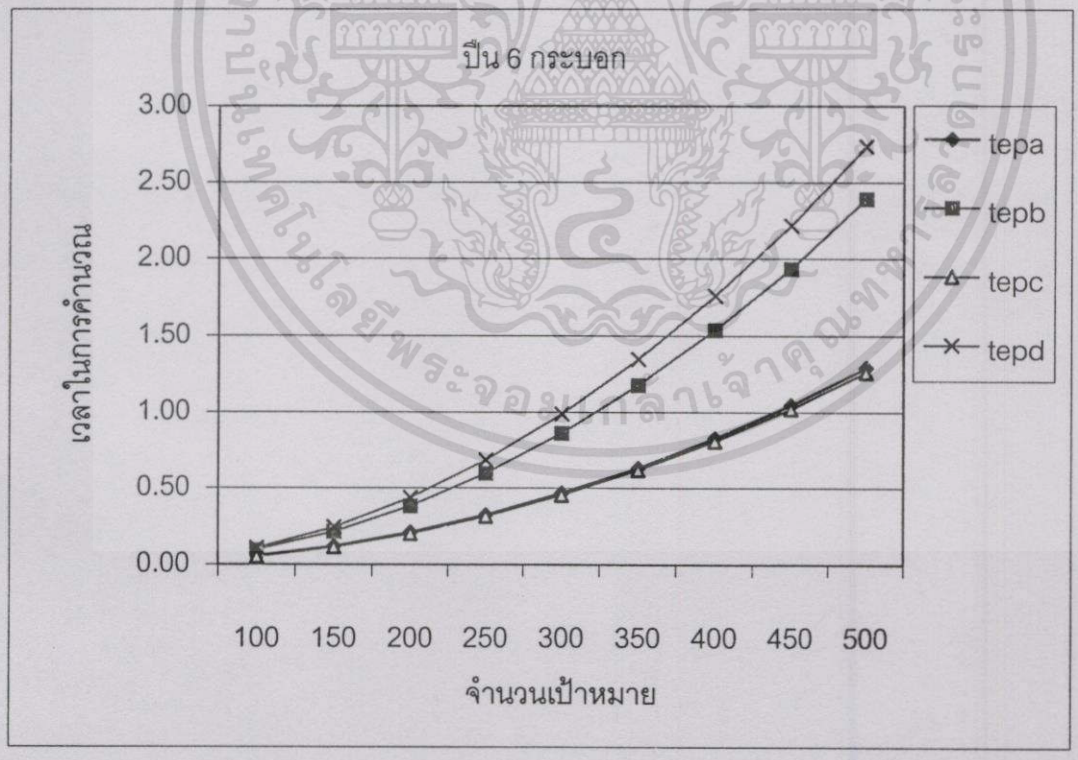
จำนวนเป้าหมาย	tepa	tepb	tepc	tepd	เวลารวม
100x100	0.040978	0.065148	0.033947	0.074179	0.218556
150x150	0.084620	0.146240	0.076403	0.167399	0.453647
200x200	0.144648	0.260298	0.135408	0.297190	0.774649
250x250	0.221773	0.405997	0.211699	0.463620	1.170939
300x300	0.316609	0.586231	0.304970	0.669862	1.667258
350x350	0.428665	0.798495	0.415703	0.913593	2.260703
400x400	0.558799	1.054494	0.544521	1.199702	2.957581
450x450	0.705015	1.319139	0.690400	1.507257	3.707900
500x500	0.871346	1.627777	0.852886	1.859750	4.613014



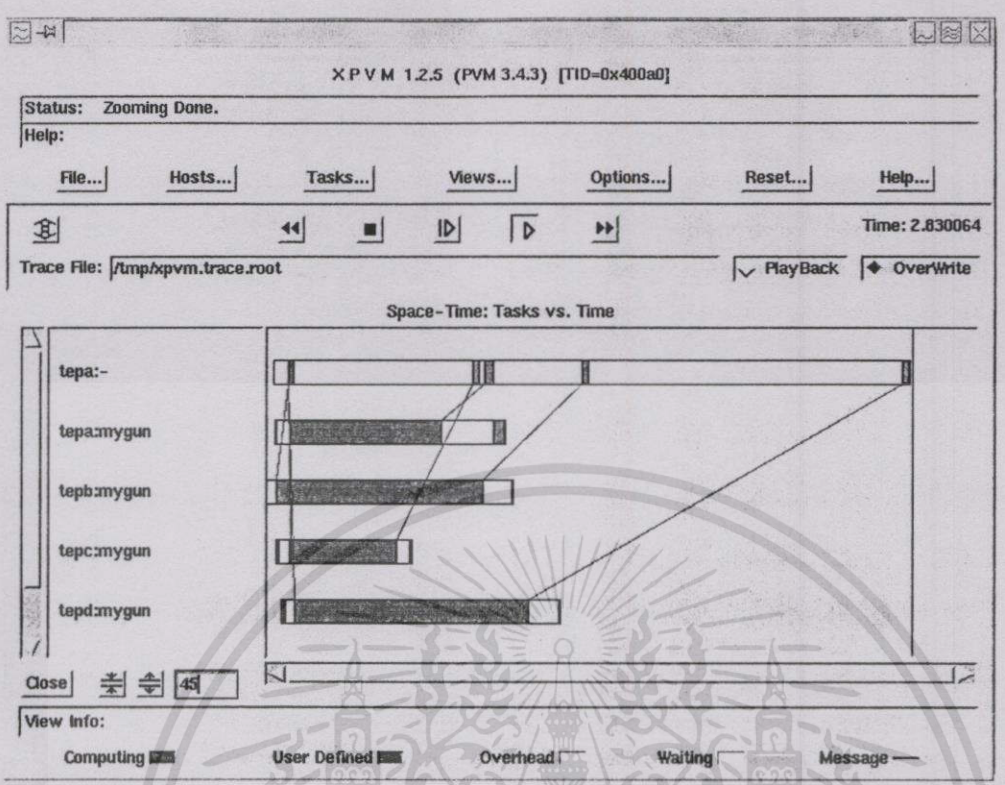
รูปที่ 2.26 เวลาการคำนวณหาหลักฐานยิงปืนใหญ่ ปืน 4 กระบอก

ตารางที่ 2.7 เปรียบเทียบเวลาการคำนวณหาหลักฐานยิงปืนใหญ่ ปืน 6 กระบอก

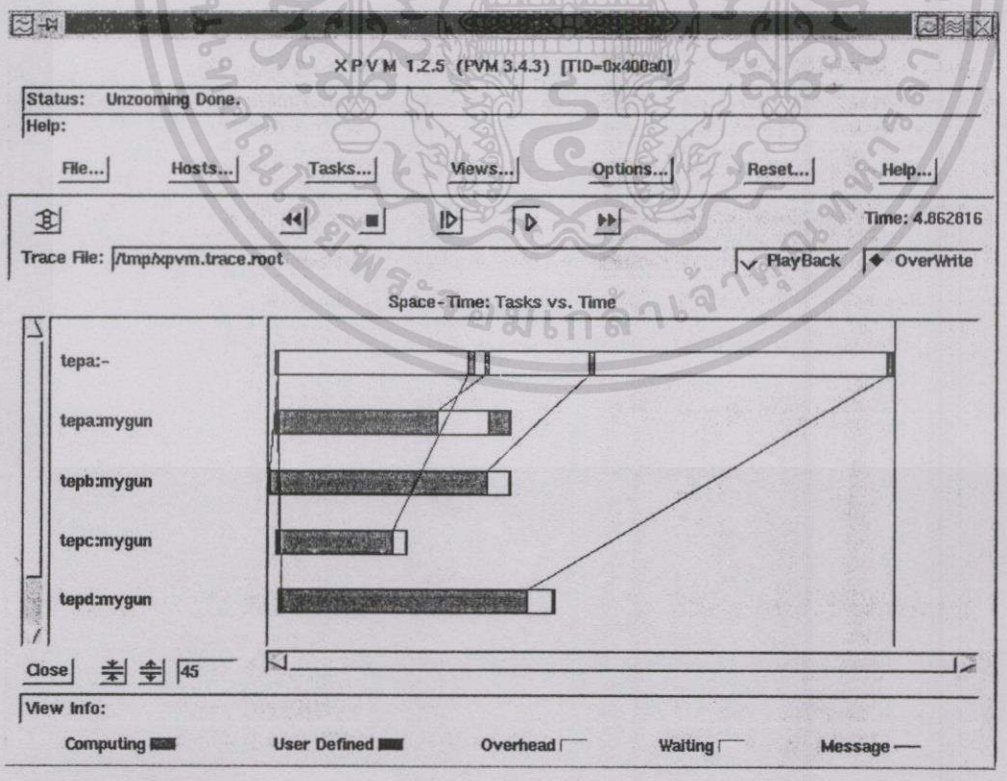
จำนวนเป้าหมาย	tepa	tepb	tepc	tepd	เวลารวม
100x100	0.056093	0.095223	0.050118	0.108681	0.291602
150x150	0.119452	0.213858	0.112492	0.244540	0.617276
200x200	0.209043	0.380770	0.200219	0.435869	1.072931
250x250	0.323713	0.596694	0.313038	0.683467	1.645746
300x300	0.463970	0.860840	0.451244	0.984422	2.373993
350x350	0.630098	1.173292	0.615117	1.342023	3.207845
400x400	0.826088	1.537068	0.805242	1.757765	4.205198
450x450	1.043136	1.933389	1.020580	2.214119	5.292057
500x500	1.287628	2.389740	1.260236	2.734567	6.544229



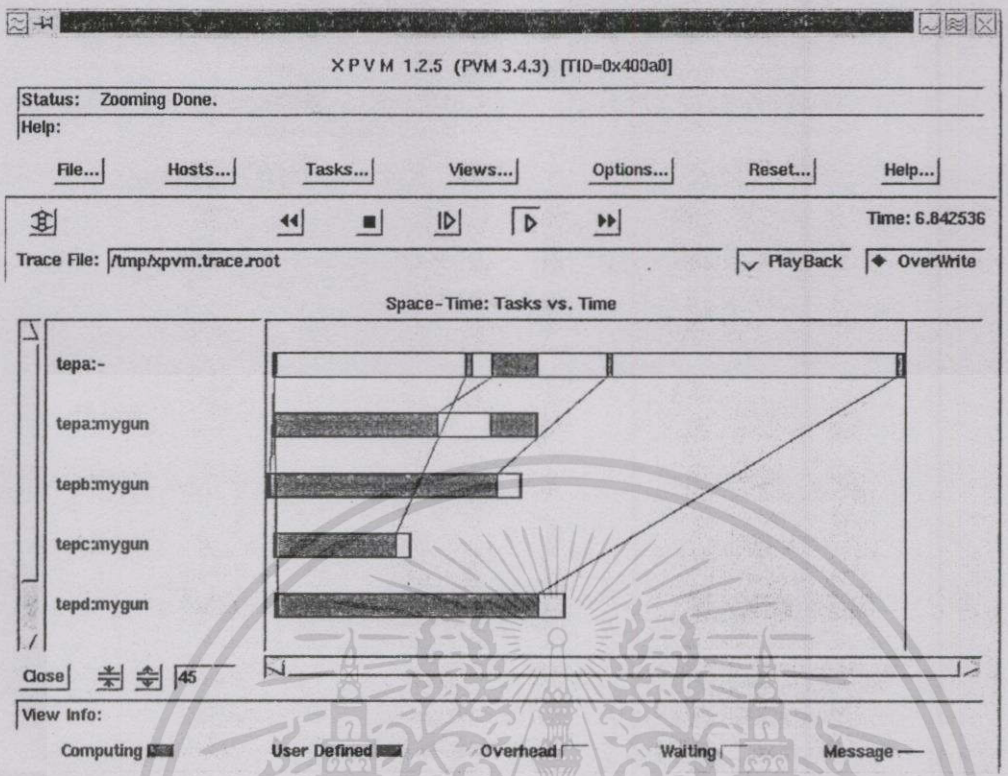
รูปที่ 2.27 เวลาการคำนวณหาหลักฐานยิงปืนใหญ่ ปืน 6 กระบอก



รูปที่ 2.28 การคำนวณหาหลักฐานยิงของปืน 2 กระบอกต่อเป้าหมาย 500X500



รูปที่ 2.29 การคำนวณหาหลักฐานยิงของปืน 4 กระบอกต่อเป้าหมาย 500X500



รูปที่ 2.30 การคำนวณหาหลักฐานยิงของปืน 6 กระบอกต่อเป้าหมาย 500X500

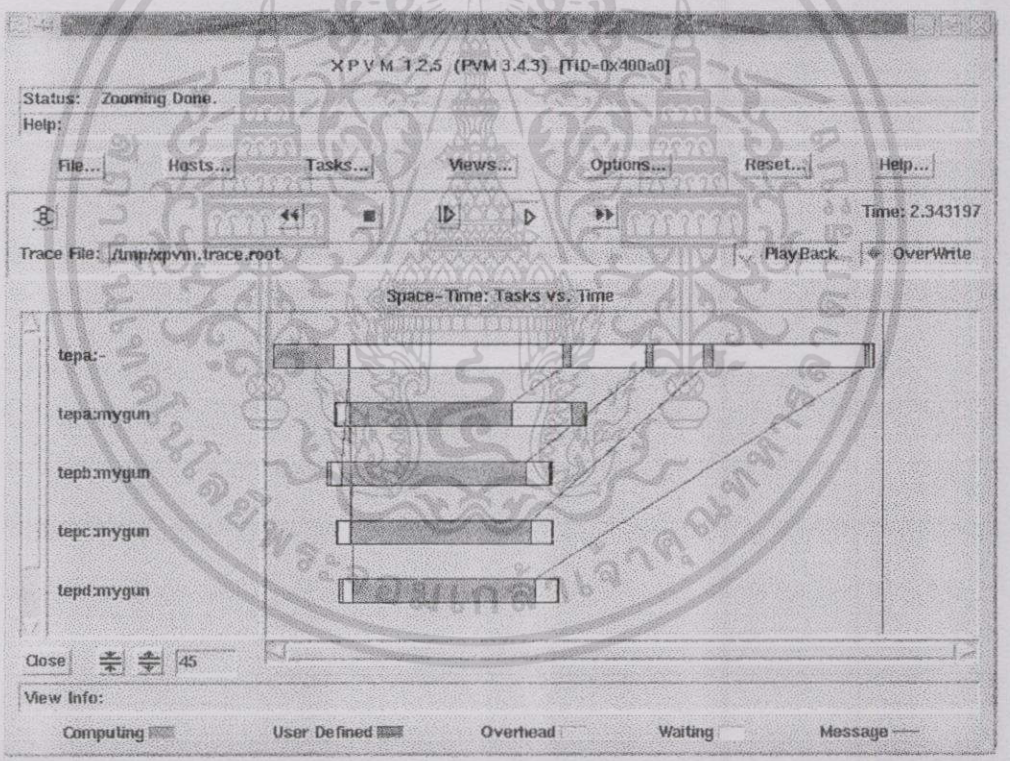
ตารางที่ 2.8 ค่าเฉลี่ยความสามารถในการคำนวณจำนวนเป้าหมายต่อวินาที

	tepa	tepb	tepc	tepd
2 กระบอก	1067552.950514	576485.047327	1128866.265806	505231.133909
4 กระบอก	1110570.222468	613656.239202	1177671.694827	537425.019334
6 กระบอก	1147609.231668	628282.278379	1195379.164192	549231.835896
เฉลี่ย	**1108577.468217**	**606141.188303**	**1167305.708275**	**530629.329713**

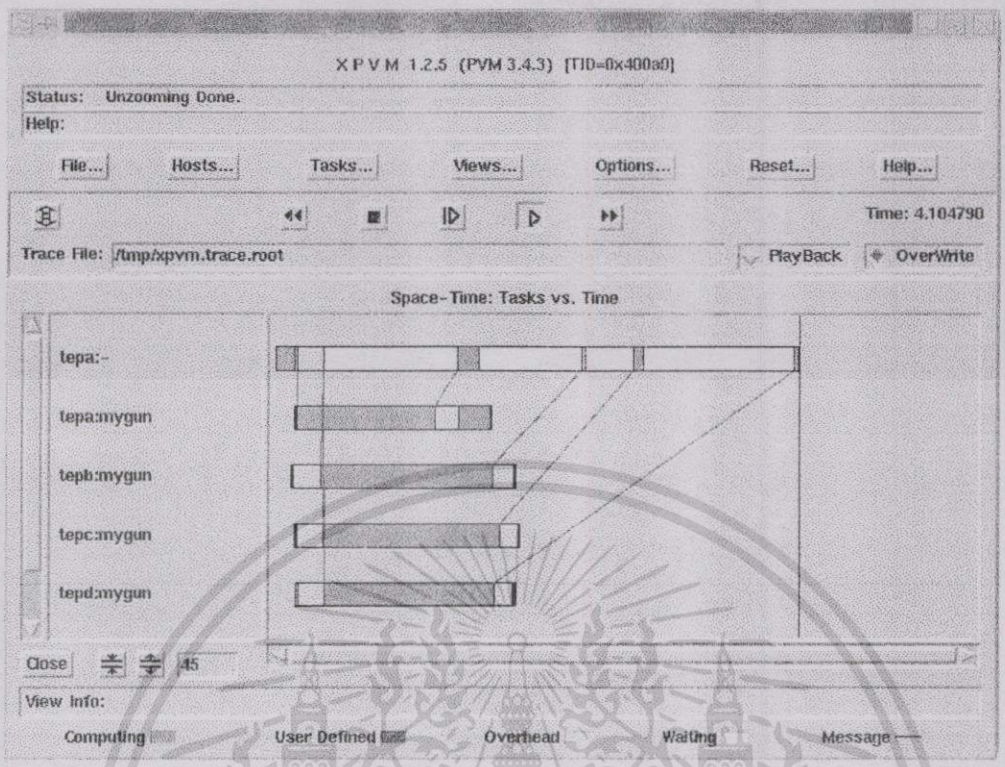
ซึ่งถ้าเครื่องมีความเร็วในการประมวลผลเท่ากัน ควรจะใช้เวลาในการคำนวณได้ใกล้เคียงกัน แต่เนื่องจากขีดความสามารถของเครื่องมีความแตกต่างกัน แต่ระบบ PVM จะจัดความเร็วของเครื่องเริ่มต้นที่ 1000 ดังนั้นผู้ใช้จึงต้องจัดปรับความเร็วนี้ให้เหมาะสมกับสภาพแวดล้อมที่ใช้งานอยู่ เมื่อพิจารณาแล้วโดยอาศัยข้อมูลจากการทดลอง จะเห็นได้ว่าแต่ละเครื่องมีขีดความสามารถในการคำนวณไม่เท่ากัน เครื่องที่ 3 (tepc), เครื่องที่ 1 (tepa), เครื่องที่ 2 (tepb) และเครื่องที่ 4 (tepd) มีความเร็วในการคำนวณมากไปหาน้อยตามลำดับ

2.8.3 โปรแกรมพัฒนาการจัดการของระบบ PVM

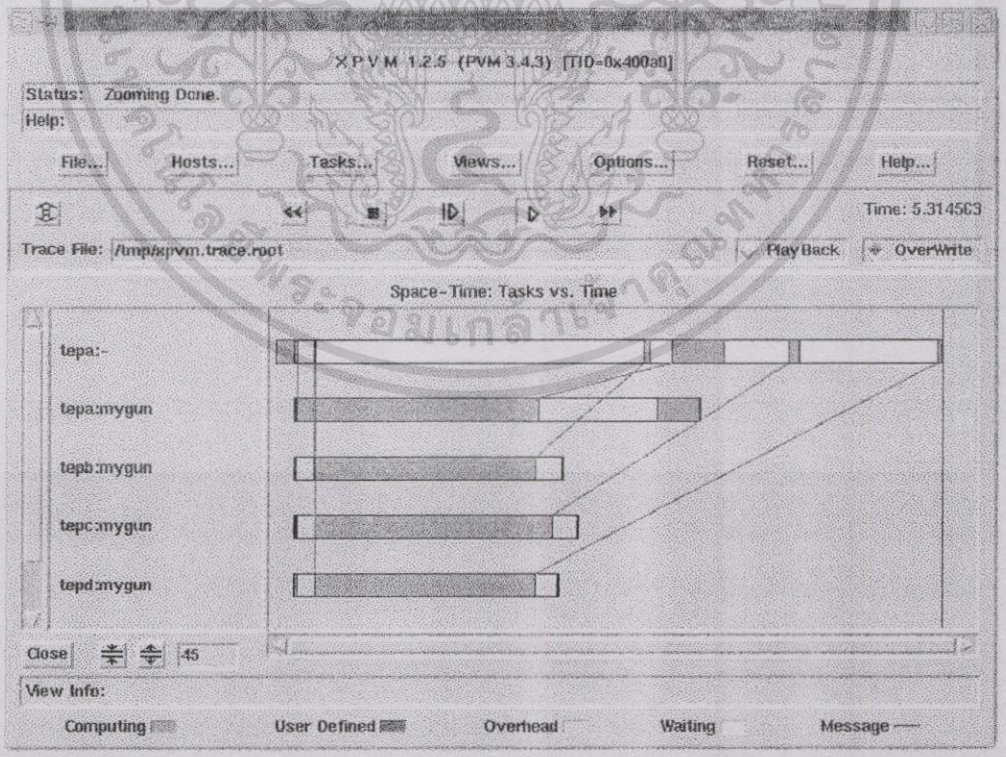
ทำการเขียนโปรแกรมพัฒนาการจัดการ โดยพิจารณาความสามารถในการคำนวณเป็นจำนวนเป้าหมายต่อวินาที ซึ่งได้จากการทดลองครั้งแรก แล้วนำมาคำนวณเพื่อแบ่งจำนวนเป้าหมายให้แต่ละเครื่องอย่างเหมาะสม



รูปที่ 2.31 การคำนวณหาหลักฐานยิงด้วยโปรแกรมการจัดการ พิจารณา Load Average สำหรับปืน 2 กระบอก 500X500 เป้าหมาย



รูปที่ 2.32 การคำนวณหาหลักฐานยิงด้วยโปรแกรมการจัดการ พิจารณา Load Average สำหรับปืน 4 กระบอก 500X500 เป้าหมาย



รูปที่ 2.33 การคำนวณหาหลักฐานยิงด้วยโปรแกรมการจัดการ พิจารณา Load Average สำหรับปืน 6 กระบอก 500X500 เป้าหมาย

ตารางที่ 2.9 เวลาการคำนวณด้วยโปรแกรมพัฒนาการจัดการปืน 2 กระบอก

จำนวนเป้าหมาย	tepa	tepb	tepc	tepd	เวลารวม
100x100	0.020484	0.034239	0.017753	0.039513	0.253401
150x150	0.050872	0.059590	0.060374	0.059587	0.335076
200x200	0.097775	0.095648	0.100611	0.099683	0.491120
250x250	0.152743	0.141871	0.155711	0.154170	0.619729
300x300	0.224993	0.212483	0.224434	0.221452	0.825756
350x350	0.291976	0.300864	0.304238	0.300373	1.012110
400x400	0.360974	0.418200	0.401373	0.400201	1.290092
450x450	0.444506	0.495506	0.524520	0.539605	1.653672
500x500	0.580162	0.588435	0.621412	0.661330	2.026143

ตารางที่ 2.10 เวลาการคำนวณด้วยโปรแกรมพัฒนาการจัดการปืน 4 กระบอก

จำนวนเป้าหมาย	tepa	tepb	tepc	tepd	เวลารวม
100x100	0.052713	0.044078	0.044475	0.048371	0.287309
150x150	0.111342	0.098754	0.100035	0.108414	0.441766
200x200	0.189905	0.190608	0.174921	0.190253	0.664438
250x250	0.284763	0.288943	0.291287	0.287592	0.937051
300x300	0.414981	0.414396	0.419718	0.414721	1.309197
350x350	0.554552	0.563638	0.573242	0.564781	1.619149
400x400	0.722990	0.742958	0.749247	0.760362	2.207280
450x450	0.888233	0.958272	0.961372	0.945868	2.659464
500x500	0.047050	0.201148	0.210733	0.206275	3.210531

ตารางที่ 2.11 เวลาการคำนวณด้วยโปรแกรมพัฒนาการจัดการปืน 6 กระบอก

จำนวนเป้าหมาย	tepa	tepb	tepc	tepd	เวลารวม
100x100	0.069660	0.069066	0.070681	0.069983	0.413630
150x150	0.153952	0.153478	0.156364	0.153930	0.567915
200x200	0.269706	0.271068	0.276514	0.271669	0.871116
250x250	0.429484	0.422312	0.430713	0.422459	1.315446
300x300	0.582707	0.616783	0.628831	0.619188	1.785735
350x350	0.838687	0.850907	0.828105	0.805469	2.263512
400x400	1.015307	1.165226	1.107363	1.113171	2.982553
450x450	1.295992	1.404720	1.437871	1.459851	3.787798
500x500	1.707730	1.709063	1.720630	1.683911	4.800309

บทที่ 3
บทสรุป

3.1 เรื่องที่ทำการวิจัย

การทำวิจัยนี้มุ่งเน้นเพื่อแก้ปัญหาการจัดการของระบบ PVM โดยพัฒนาโปรแกรมแบบขนานเพื่อแก้ปัญหาทางด้านหลักยิงปืนใหญ่ ที่ผู้วิจัยได้เลือกพัฒนาโปรแกรมแบบขนานคำนวณหาหลักฐานยิงปืนใหญ่ขนาดเบา ก็เนื่องจากว่าการวางแผนการยิงของทหารปืนใหญ่นั้น ในบางสถานการณ์ต้องการผลที่รวดเร็ว และถูกต้องแม่นยำ ซึ่งปัจจุบันก็ได้มีการพัฒนาเพื่อความมุ่งหมายดังกล่าว มีการจัดหาระบบเครื่องมืออำนวยการยิงระบบอัตโนมัติแบบคอมพิวเตอร์ มาใช้ในเหล่าทหารปืนใหญ่ สามารถเชื่อมโยงเป็นระบบเครือข่าย ผ่านทางสายโทรศัพท์สนาม และหรือทางวิทยุได้ ทำให้คำนวณได้รวดเร็วยิ่งขึ้น จึงเป็นพื้นฐานให้ผู้วิจัยเกิดแนวความคิดที่จะทดลองใช้ระบบเครือข่ายมาช่วยทำการคำนวณงานแบบขนาน ซึ่งในสนามนั้นมีเครื่องมือระบบคอมพิวเตอร์ที่มีคุณลักษณะทางทหาร เชื่อมโยงเป็นระบบเครือข่ายได้จริงแล้ว ดังนั้นการทดลองนี้จึงมีความเป็นไปได้สูงที่จะนำผลการวิจัยไปใช้ประโยชน์ หรือนำไปทำการวิจัยต่อไปให้เกิดประโยชน์ตามความต้องการเฉพาะด้านได้

เนื่องจากทหารปืนใหญ่เองก็ได้มีความพยายามที่จะนำระบบคอมพิวเตอร์ มาพัฒนาทางด้านเทคนิคและทางด้านยุทธวิธีโดยต่อเนื่อง แม้ว่าเครื่องมืออำนวยการยิงระบบอัตโนมัติแบบคอมพิวเตอร์ที่มีการจัดหาเข้ามาแล้วในบางระบบ จะทำงานได้ไม่ดีนัก แต่ก็เป็นพื้นฐานทำให้เกิดความตื่นตัวที่จะพัฒนาระบบอัตโนมัติดังกล่าว ให้เป็นไปโดยอัตโนมัติทั้งระบบ ไม่ว่าจะเป็นระบบค้นหาเป้าหมาย ระบบส่วนยิง ระบบอาวุธกระสุน หรือการทำงานแผนที่ ที่เมื่อได้ข้อมูลส่วนใดมาแล้วก็จะต้องมีการดำเนินกรรมวิธีทันทีโดยอัตโนมัติ หากต้องการที่จะสนับสนุนหน่วยดำเนินกลยุทธได้อย่างต่อเนื่อง ได้ผล และเอาตัวรอดอยู่ได้ในสนามรบแล้ว ทหารปืนใหญ่จะต้องคำนวณได้รวดเร็ว ในปริมาณที่มากพอนั่นคือต้องมีการคำนวณปรับปรุงแผนการยิงอยู่ตลอดเวลา เช่น มีการเปลี่ยนที่ตั้งยิง ต้องมีการปรับแผนการยิงตามที่ตั้งยิงใหม่ เมื่องานแผนที่ทำเสร็จเรียบร้อยก็ต้องมีการคำนวณใหม่เพื่อให้หลักฐานยิงมีความถูกต้องมากที่สุด หรือเมื่อมีข่าวสภาพอากาศใหม่แผนการยิงทั้งหมดต้องปรับตัวแก้ ก็ต้องมีการคำนวณตัวแก้เพื่อหาหลักฐานยิงเตรียมไว้ตลอดเวลา

ดังนั้นการวิจัยนี้จึงมีเหตุผลเพียงพอที่จะทำการพัฒนาให้ มีการคำนวณที่รวดเร็วได้ด้วยโปรแกรมแบบขนาน และใช้เครื่องที่อยู่ในเครือข่ายช่วยทำการคำนวณ อย่างไรก็ตามผู้วิจัยได้เชื่อมต่อเครื่องคอมพิวเตอร์เป็นระบบเครือข่าย โดยไม่ได้ใช้เครื่องมืออำนวยการยิงที่กองทัพบกมีอยู่ ก็เนื่องจากว่าเป็นสิ่งอุปกรณ์ของทางราชการ ซึ่งมีราคาแพงไม่สามารถนำออกมาเพื่อทำการทดลองได้ แต่ก็ถือว่าจำลองเป็นระบบที่คล้ายคลึงกัน ซึ่งขีดความสามารถอยู่ในเกณฑ์ที่ไม่แตกต่าง

กันมากนัก สำหรับจำนวนที่ใช้เพียงจำนวน 4 เครื่อง ก็มีเหตุผลว่า ใช้ในระดับกองพันทหารปืนใหญ่ กองร้อยปืนใหญ่กองร้อยละ 1 เครื่อง (3 กองร้อย) และ ศูนย์อำนวยการยิงกองพันอีกหนึ่งเครื่อง รวมเป็น 4 เครื่อง ซึ่งใช้ข่ายวิทยุในข่ายเดียวกันได้ หากจะมีเพิ่มเติมก็แล้วแต่สถานการณ์ เช่น อาจมีปืนเพิ่มเติมกำลังยิงให้กับกองพันปืนใหญ่ช่วยโดยตรง แต่ในชั้นต้นนี้ถือเอาตามมาตรฐานที่จัดในเริ่มต้นก่อน

จากเหตุผลดังกล่าวจึงได้ใช้เครื่องจำนวนน้อย และทำให้การพัฒนาระบบ PVM ในเครือข่ายเล็ก ๆ มีความแตกต่างไปจากการพัฒนาในระบบใหญ่ หรือตามความมุ่งหมายที่แตกต่างกันไป เนื่องจากสภาพแวดล้อมไม่เหมือนกัน เป้าหมายในการพัฒนาก็แตกต่างกันไป

นอกจากนี้ในประเทศไทยเองก็มีระบบเครือข่ายเล็ก ๆ อยู่เป็นจำนวนมาก โดยที่ไม่มีการพัฒนาโปรแกรมการใช้งานให้ใช้ประโยชน์จากเครื่องคอมพิวเตอร์ที่อยู่ได้เต็มประสิทธิภาพ ด้วยเหตุนี้การพัฒนาการจัดการของระบบ PVM อาจจะเป็นประโยชน์ต่อผู้สนใจทั่วไป และผู้สนใจเฉพาะด้านได้เป็นอย่างดี

ผู้วิจัยได้พยายามนำเครื่องคอมพิวเตอร์ที่มีการเชื่อมโยงเป็นระบบเครือข่าย ซึ่งเดิมก็ทำการเชื่อมโยงกันในความมุ่งหมายในการใช้งานทั่วไป ซึ่งในการใช้งานนั้นผู้ใช้อาจใช้งานได้ไม่เต็มประสิทธิภาพเท่าไรนัก อาจจะมีเหตุผลอยู่หลายประการเช่น

- เครื่องที่ซื้อมาใช้งานนั้น ไม่มีโปรแกรมที่เหมาะสมในการใช้งานของสำนักงานนั้น เนื่องจากงบประมาณไม่เพียงพอ ต่อการจัดหาโปรแกรมที่ใช้ในความมุ่งหมายแบบเฉพาะได้
- ผู้ใช้ไม่มีความสามารถในการพัฒนาโปรแกรมขึ้นใช้งานเอง เนื่องจากเป็นผู้ใช้งานทั่วไปในขั้นต้นเท่านั้น
- องค์กรไม่สามารถพัฒนาบุคลากรได้ทันและเพียงพอต่อความต้องการ
- เครื่องคอมพิวเตอร์มีความล้าสมัย ไม่สามารถใช้งานกับโปรแกรมประยุกต์ในปัจจุบันได้

การที่เราสามารถนำเครื่องในระบบเครือข่ายมาช่วยทำการคำนวณงาน ที่มีปริมาณการคำนวณสูงได้โดยไม่ต้องจัดหาเครื่องที่ทำงานแบบขนานได้เช่น SuperComputer ซึ่งมีราคาสูงมากนั้น นับได้ว่ามีความคุ้มค่ามาก ถึงแม้ว่าจะมีขีดความสามารถไม่เทียบเท่าเครื่องแบบขนานก็ตาม แต่หากได้มีการพัฒนาโปรแกรมให้ทำงานได้อย่างมีประสิทธิภาพแล้ว เราก็จะมีเครื่องคอมพิวเตอร์ในระบบเครือข่ายที่ผู้ใช้สามารถใช้งานได้ตามปกติ ความเร็วในการใช้งานไม่ลดลงไปมากนัก ซึ่งผู้ใช้อาจไม่รู้เลยว่า เครื่องที่ตนเองใช้งานนั้นเป็นเครื่องที่ระบบแบบเสมือน ซึ่งเมื่อว่างจากการใช้งานของผู้ใช้แล้ว ก็จะมีความสามารถในการช่วยกันประมวลผลงานแบบขนานทันที การวิจัยครั้งนี้จะเป็นประโยชน์อย่างมากต่อการศึกษาวิจัยทางด้านวิทยาศาสตร์ ซึ่งอาจมีการใช้การคำนวณที่มีปริมาณสูงมาก ซึ่งการประมวลผลจากเครื่องคอมพิวเตอร์โดยทั่วไป ไม่สามารถกระทำได้ หรืออาจกระทำได้แต่ต้องเสียเวลาและทรัพยากรจำนวนมาก

การที่จะทำให้เครื่องในระบบเครือข่ายมีความสามารถในการประมวลผลแบบขนานได้นั้น จะต้องอาศัยโปรแกรมที่มีความสามารถในการประมวลแบบขนาน รวมทั้งต้องมีความสามารถเป็นเหมือนระบบปฏิบัติการแบบเครือข่ายอีกด้วย ในปัจจุบันมีโปรแกรมตามความมุ่งหมายดังกล่าวแล้วเป็นจำนวนมาก เช่น P4, Linda, Condor, Piranya, MPI, PVM ฯลฯ การวิจัยครั้งนี้ได้เลือกใช้โปรแกรม PVM เนื่องจากเป็นโปรแกรมที่แจกจ่ายฟรี สามารถดาวน์โหลดมาใช้งานได้ตลอดเวลา อีกทั้งมีความอ่อนตัว สามารถทำงานได้กับเครื่องหลายชนิด และกับระบบปฏิบัติการหลายแบบ โดยเฉพาะอย่างยิ่ง ทำงานบนระบบยูนิกซ์ และ ลินุกซ์ ได้เป็นอย่างดี และเนื่องด้วยระบบเครือข่ายของหน่วยงานและองค์กรของรัฐ โดยส่วนมากมีการเชื่อมโยงที่ไม่กว้างขวางมากนัก และมีจำนวนน้อย สามารถติดตั้งระบบปฏิบัติการลินุกซ์ ซึ่งมีผู้ช่วยกันพัฒนาเป็นจำนวนมากโดยอาศัยเครือข่ายอินเทอร์เนต มีความแน่นอน มั่นคง เพิ่มขึ้นเรื่อย ๆ ใช้งานได้ดีกับเครื่องคอมพิวเตอร์แบบตั้งโต๊ะที่มีใช้งานอยู่เป็นจำนวนมากในประเทศไทย ได้เป็นอย่างดี ดังนั้นการวิจัยด้วยระบบดังกล่าวน่าจะมีความเป็นไปได้ในการนำไปพัฒนาจริงในประเทศได้

อย่างไรก็ตามโปรแกรม PVM ยังมีข้อจำกัดในเรื่องของการจัดการ เนื่องจากผู้ใช้จะต้องเป็นผู้คอยควบคุมเครื่องในระบบว่าต้องการใช้เครื่องใดบ้างในการคำนวณ ซึ่งถ้าเครื่องมีจำนวนมากขึ้นผู้ใช้ก็จะไม่สามารถทราบได้ว่า เครื่องใดมีผู้ใช้กำลังใช้งานอยู่บ้าง ซึ่งทำให้ลดประสิทธิภาพของโปรแกรม PVM ในการประมวลผลงานแบบขนานลง เพราะโปรแกรมจะแจกจ่ายงานให้กับเครื่องในระบบเสมือนแบบเป็นไปตามลำดับ และจ่ายให้ในปริมาณที่เท่ากัน หรือใกล้เคียงกัน หากเครื่องในระบบเป็นประเภทเดียวกัน มีความเร็ว และทรัพยากรเท่ากัน รวมทั้งไม่มีการใช้งานจากผู้ใช้ จะทำให้โปรแกรมแบบขนานที่ทำงานบนระบบ PVM สามารถทำงานได้อย่างมีประสิทธิภาพ แต่ถ้าเครื่องมีความแตกต่างทั้งแบบ หรือรุ่น ก็จะทำให้ประสิทธิภาพโดยรวมลดลง เพียงแต่เราอาศัยช่วงเวลาที่เครื่องว่างมาช่วยทำการประมวลผล โปรแกรม PVM ก็จะแจกจ่ายงานให้ในลักษณะเดิม เมื่อเครื่องที่มีประสิทธิภาพต่ำได้รับงานในจำนวนที่เท่ากับเครื่องอื่น ๆ ที่ว่าง ก็จะทำการคำนวณได้ช้ากว่า เครื่องแม่ข่ายที่เป็นตัวแจกจ่ายงานจะต้องรอผลการคำนวณจากเครื่องที่มีปริมาณงานมากนั้นจนกว่าจะประมวลผลเสร็จ และส่งผลกลับมาเพื่อให้เครื่องแม่ข่ายทำการรวมผลลัพธ์และแสดงผล ซึ่งอาจทำให้การประมวลผลงานแบบขนานช้ามากกว่างานแบบลำดับเสียอีก ดังนั้นผู้วิจัยจึงได้เลือกที่จะพัฒนาระบบการจัดการของโปรแกรม PVM เพื่อให้โปรแกรมมีความสามารถในการพิจารณาตกลงใจ แจกจ่ายงานให้กับเครื่องในระบบได้อย่างเหมาะสม โดยวิเคราะห์ก่อนทำการแจกจ่ายงานให้กับเครื่องเพื่อประมวลผลต่อไป ทั้งนี้จะต้องมีขีดความสามารถในการเพิ่มเครื่องที่ว่างเข้ามาในระบบได้โดยอัตโนมัติ โดยที่ผู้ใช้ไม่ต้องกังวลกับการพิจารณาใช้เครื่องในระบบอีกต่อไป

โดยทั่วไปได้มีโปรแกรมที่มีความสามารถ ในการจัดการงานแบบขนานในระบบเสมือนอยู่เป็นจำนวนมาก มีขีดความสามารถในการ Migrate งานแบบขนานจากเครื่องที่มีปริมาณงานมากใน

ระหว่าง Runtime ไปยังเครื่องที่ว่างเครื่องอื่นโดยสามารถเริ่มงานต่อจากเครื่องที่ถูก Migrate ไปได้ด้วย โดยเฉพาะมีการตรวจวัดปริมาณของทุกเครื่องในระบบอยู่ตลอดเวลา ซึ่งสามารถปรับปรุงเปลี่ยนแปลงสถานะของระบบเสมือนได้ตลอดเวลา แต่เนื่องจากโปรแกรมโดยส่วนมากมักเป็นโปรแกรมที่มีความมุ่งหมายในด้านการค้า อีกทั้งมีการพัฒนามาใช้งานกับระบบที่ใหญ่มากโดยตรง หากนำมาใช้กับเครือข่ายที่มีจำนวนน้อยจะไม่สามารถใช้งานได้เต็มประสิทธิภาพของโปรแกรมได้ จึงได้เลือกแบบและวิธีการที่คิดว่ามีความเหมาะสมกับระบบเครือข่ายขนานย่อม และทำการทดลองเพื่อวัดประสิทธิภาพที่เพิ่มขึ้นด้วยเครือข่ายเพียงแค่ 4 เครื่อง ซึ่งน่าจะครอบคลุมเครื่องจำนวนน้อย ๆ เช่น 10-30 เครื่อง

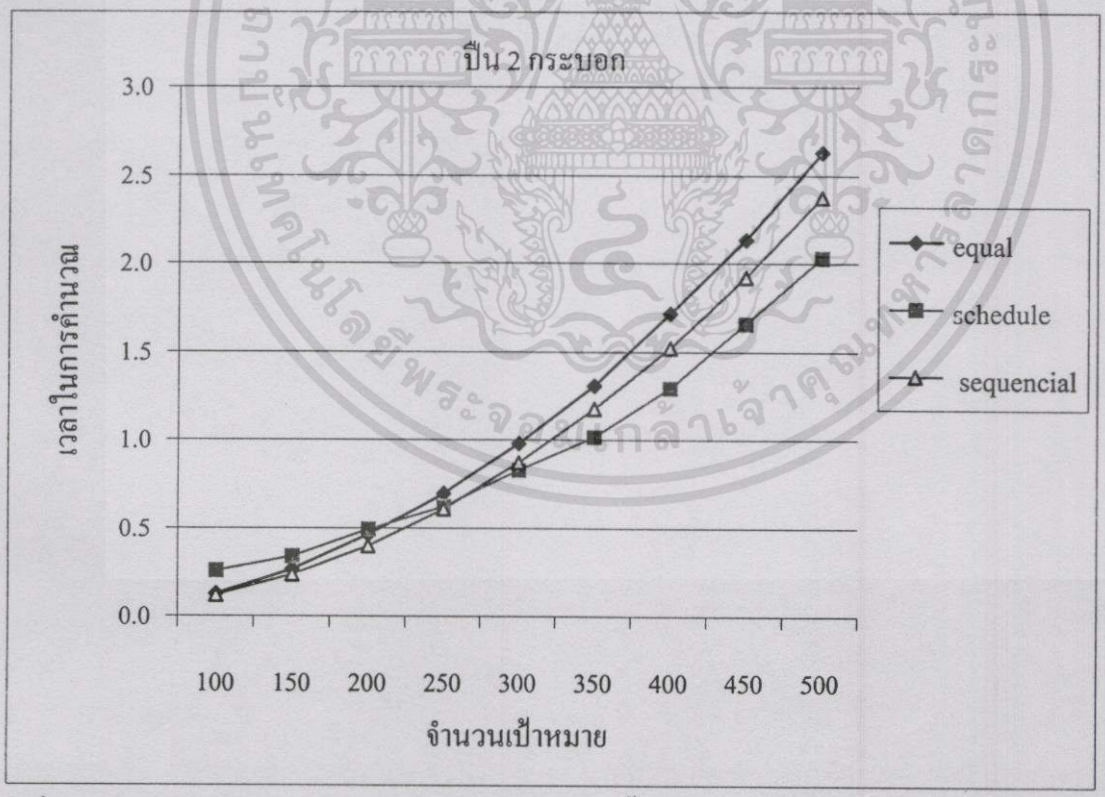
3.2 ผลการทดลอง

การทดลองในสภาวะแวดล้อมเครือข่ายส่วนตัว มีเครื่องคอมพิวเตอร์จำนวน 4 เครื่อง ซึ่งมีความเร็วในการคำนวณไม่เท่ากันนั้น สามารถวัดได้ด้วยการแบ่งเป้าหมายให้ทำการคำนวณ โดยเฉลี่ยเท่ากัน เมื่อทำการ Run โปรแกรมแบบขนาน จะสามารถหาค่าเฉลี่ยความสามารถในการคำนวณของเครื่องแต่ละเครื่องเป็นจำนวนเป้าหมายต่อวินาทีได้ จากนั้นนำค่าตัวแก้นี้ไปปรับแก้ที่ตัวแปรความเร็วของเครื่องในระบบ PVM ทำให้เครื่องในระบบสามารถเริ่มและจบงานได้ในเวลาที่ใกล้เคียงกัน รวมทั้งยังได้รับงานไปทำการคำนวณทุกเครื่อง ไม่มีเครื่องใดว่างจากการคำนวณ เป็นการใช้งานเครื่องคอมพิวเตอร์ได้อย่างมีประสิทธิภาพทุกเครื่อง

เมื่อได้พัฒนาโปรแกรมจัดการให้มีความสามารถในการแจกจ่ายงาน ให้กับเครื่องต่าง ๆ ด้วยจำนวนเป้าหมายที่เหมาะสม ทำให้แต่ละเครื่องในระบบใช้เวลาการคำนวณที่ใกล้เคียงกันแล้ว เพื่อให้สามารถวัดประสิทธิภาพที่เพิ่มขึ้น จึงได้บันทึกเวลาการทำงานของโปรแกรมทั้งสิ้น ตั้งแต่เริ่มงานแจกจ่ายงาน จนกระทั่งทำการคำนวณหาหลักฐานยิงเสร็จเรียบร้อยทุกเครื่อง และส่งผลให้กับเครื่องหลักทำการแสดงผล โดยทำการเปรียบเทียบการทำงานของโปรแกรมจำนวน 3 โปรแกรม คือ โปรการทำงานแบบลำดับ (Sequencial) ได้แก่ โปรแกรมคำนวณหาหลักฐานยิงของเป้าหมายทั้งหมดด้วยเครื่องเพียงเครื่องเดียว โปรแกรมแบบขนาน (Equal) ได้แก่ โปรแกรมแบบขนานที่แจกจ่ายงานในการคำนวณ (เป้าหมาย) ให้กับทุกเครื่องเท่า ๆ กัน และ โปรแกรมพัฒนาการจัดการ (Schedule) ได้แก่ โปรแกรมที่ได้แบ่งเป้าหมายในการคำนวณให้ทุก ๆ เครื่อง ไม่เท่ากันตามความสามารถในการคำนวณของแต่ละเครื่อง

ตารางที่ 3.1 เปรียบเทียบผลการคำนวณของโปรแกรมทั้งสามแบบ ด้วยปืน 2 กระบอก

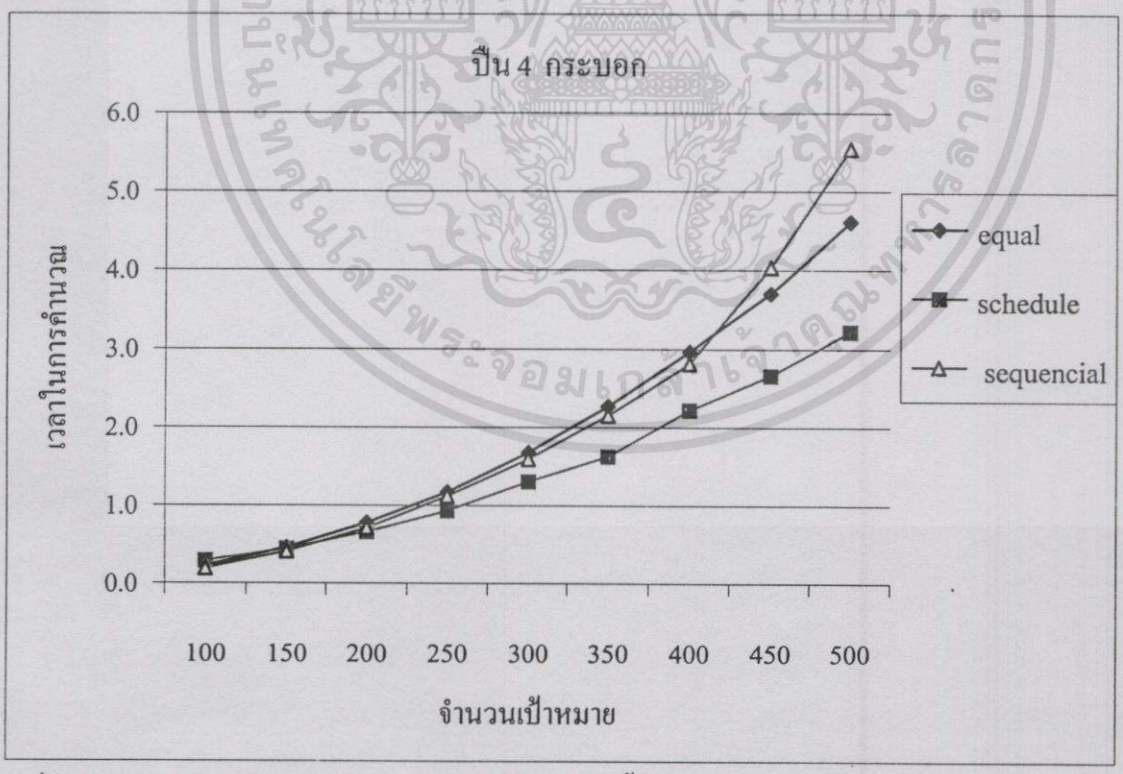
จำนวนเป้าหมาย	sequencial	equal	schedule
100x100	0.113324	0.122548	0.253401
150x150	0.230003	0.261834	0.335076
200x200	0.395956	0.462556	0.491120
250x250	0.606261	0.691634	0.619729
300x300	0.867341	0.975577	0.825756
350x350	1.174918	1.305386	1.012110
400x400	1.520326	1.716308	1.290092
450x450	1.920068	2.129926	1.653672
500x500	2.370532	2.631272	2.026143



รูปที่ 3.1 กราฟเปรียบเทียบผลการคำนวณของโปรแกรมทั้งสามแบบด้วยปืน 2 กระบอก

ตารางที่ 3.2 เปรียบเทียบผลการคำนวณของโปรแกรมทั้งสามแบบ ด้วยปืน 4 กระบอก

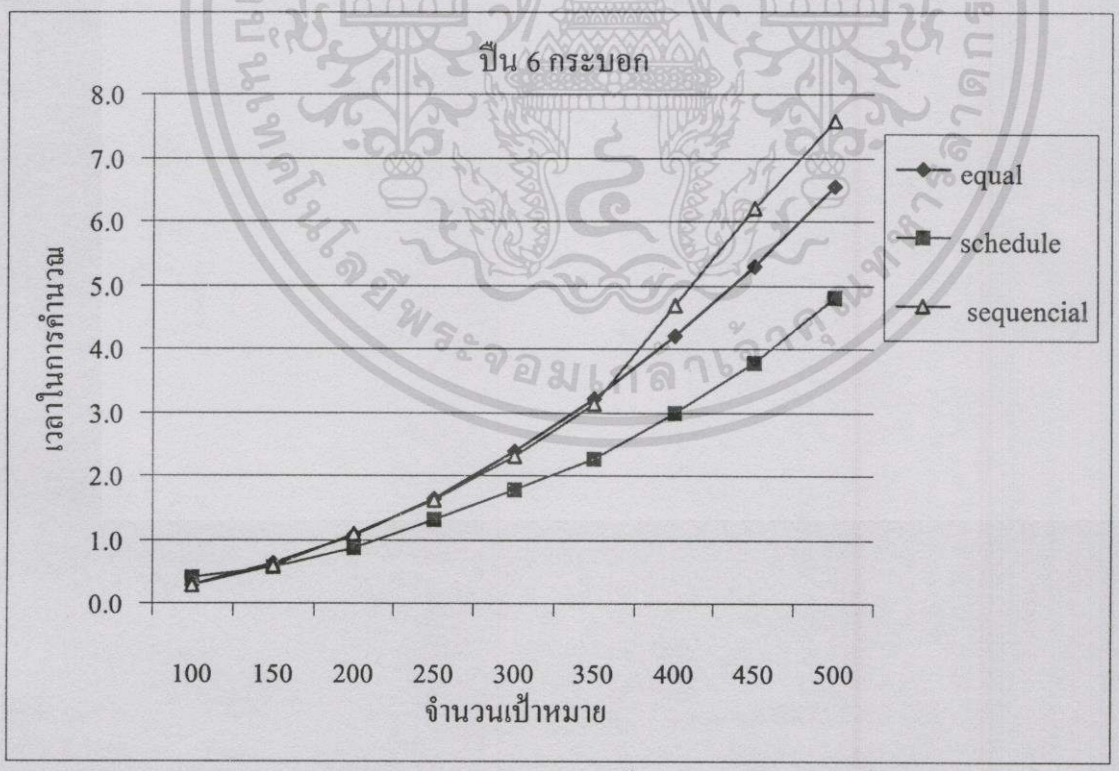
จำนวนเป้าหมาย	sequencial	equal	schedule
100x100	0.194213	0.218556	0.287309
150x150	0.410387	0.453647	0.441766
200x200	0.717940	0.774649	0.664438
250x250	1.108392	1.170939	0.937051
300x300	1.585453	1.667258	1.309197
350x350	2.151862	2.260703	1.619149
400x400	2.808341	2.957581	2.207280
450x450	4.046333	3.707900	2.659464
500x500	5.544394	4.613014	3.210531



รูปที่ 3.2 กราฟเปรียบเทียบผลการคำนวณของโปรแกรมทั้งสามแบบ ด้วยปืน 4 กระบอก

ตารางที่ 3.3 เปรียบเทียบผลการคำนวณของโปรแกรมทั้งสามแบบ ด้วยปืน 6 กระบอก

จำนวนเป้าหมาย	sequencial	equal	schedule
100x100	0.273929	0.291602	0.413630
150x150	0.590258	0.617276	0.567915
200x200	1.089500	1.072931	0.871116
250x250	1.609551	1.645746	1.315446
300x300	2.309682	2.373993	1.785735
350x350	3.140471	3.207845	2.263512
400x400	4.690057	4.205198	2.982553
450x450	6.204317	5.292057	3.787798
500x500	7.581817	6.544229	4.800309



รูปที่ 3.3 กราฟเปรียบเทียบผลการคำนวณของโปรแกรมทั้งสามแบบ ด้วยปืน 6 กระบอก

3.2.1 ปืนจำนวน 2 กระบอก

โปรแกรมพัฒนาการจัดการ สามารถทำงานได้เร็วกว่าโปรแกรมแบบลำดับ และโปรแกรมแบบขนาน เมื่อมีเป้าหมายมากกว่า 200x200 เป้าหมาย และโปรแกรมแบบขนานเมื่อยังไม่ได้มีการจัดการทำงานได้ช้ากว่าโปรแกรมแบบลำดับ ผลที่ได้รับจากการพัฒนาการจัดการทำให้โปรแกรมทำงานได้เร็วกว่าโปรแกรมแบบขนานเดิมโดยเฉลี่ยร้อยละ 3.80

3.2.2 ปืนจำนวน 4 กระบอก

โปรแกรมพัฒนาการจัดการ สามารถทำงานได้เร็วกว่าโปรแกรมแบบลำดับ และโปรแกรมแบบขนาน เมื่อมีเป้าหมายมากกว่า 150x150 เป้าหมาย และโปรแกรมแบบขนานเมื่อยังไม่ได้มีการพัฒนาการจัดการทำงานได้ไม่แตกต่างกับโปรแกรมแบบลำดับมากนัก แต่จะทำงานได้เร็วกว่าโปรแกรมแบบลำดับเมื่อมีเป้าหมายมากกว่า 400x400 เป้าหมาย ผลที่ได้รับจากการพัฒนาการจัดการทำให้โปรแกรมทำงานได้เร็วกว่าโปรแกรมแบบขนานเดิมโดยเฉลี่ยร้อยละ 15.47

3.2.3 ปืนจำนวน 6 กระบอก

โปรแกรมพัฒนาการจัดการ สามารถทำงานได้เร็วกว่าโปรแกรมแบบลำดับ และโปรแกรมแบบขนาน เมื่อมีเป้าหมายมากกว่า 150x150 เป้าหมาย และโปรแกรมแบบขนานเมื่อยังไม่ได้มีการพัฒนาการจัดการทำงานได้เร็วกว่าโปรแกรมแบบลำดับเมื่อมีเป้าหมายมากกว่า 350x350 เป้าหมาย ผลที่ได้รับจากการพัฒนาการจัดการทำให้โปรแกรมทำงานได้เร็วกว่าโปรแกรมแบบขนานเดิมโดยเฉลี่ยร้อยละ 15.93

3.2.4 การทำงานของโปรแกรมแบบขนาน

เมื่อยังไม่ได้มีการพัฒนาการจัดการ ทำงานได้ไม่เร็วกว่าโปรแกรมแบบลำดับ เมื่อเป้าหมายมีขนาดน้อย โดยเฉพาะเมื่อคำนวณด้วยปืน 2 กระบอก ทำงานได้ช้ากว่า แต่เมื่อเพิ่มจำนวนปืนและเป้าหมาย จะทำงานได้เร็วกว่าเมื่อใช้ปืนจำนวน 4 กระบอก และ 6 กระบอกโดยประมาณเมื่อมีเป้าหมายมากกว่า 350x350 เป้าหมาย

3.2.5 การทำงานของโปรแกรมพัฒนาการจัดการ

สำหรับโปรแกรมที่ได้พัฒนาการจัดการแล้ว โดยเฉลี่ยทำงานได้เร็วกว่าโปรแกรมแบบลำดับ และโปรแกรมแบบขนานที่ยังไม่ได้พัฒนาการจัดการ โดยสรุปแล้วเร็วกว่าโปรแกรมแบบลำดับร้อยละ 10.22 และเร็วกว่าโปรแกรมแบบขนานร้อยละ 11.74

3.3 ข้อเสนอแนะ

การพัฒนาการจัดการที่ได้ทำการทดลองนี้ เป็นการจัดการที่ยังไม่ได้มีการพิจารณาปริมาณการทำงานของเครื่องปกติ เนื่องจากเป็นระบบเครือข่ายส่วนตัว แต่ผลจากการทดลองก็สามารถนำไป

พัฒนาร่วมกับการพิจารณาปริมาณงานของแต่ละเครื่องได้เป็นอย่างดี อย่างไรก็ตามโปรแกรม PVM ก็ยังมีปัญหาที่ไม่สามารถควบคุมให้ทำการแบ่งงานให้กับเครื่องที่เป็นเครื่องหลักได้ หากสามารถทำได้ก็จะทำให้เครื่องหลักไม่ต้องรับเป้าหมายไปคำนวณ ซึ่งอาจทำให้ระบบการจัดการดีขึ้นได้ ทั้งนี้ขึ้นอยู่กับปริมาณงานที่แบ่งให้กับแต่ละเครื่อง จะต้องมีขนาดที่ใหญ่พอเหมาะ

ผู้วิจัยหวังเป็นอย่างยิ่งว่า การทำวิจัยในเรื่องระบบการจัดการสำหรับโปรแกรมแบบขนานบนระบบ PVM จะเป็นประโยชน์ต่อผู้สนใจในเรื่องโปรแกรมแบบขนาน ซึ่งใช้เครื่องในระบบเครือข่ายให้เกิดประโยชน์สูงสุด อย่างไรก็ตามโปรแกรมที่ใช้งานนั้น ในปัจจุบันมีอยู่มากมาย ควรเลือกใช้งาน และนำมาพัฒนาเพิ่มเติมให้เหมาะสมกับการใช้งานต่อไป

เอกสารอ้างอิง

[1] แผนกวิชาหลักยิง ศูนย์การทหารปืนใหญ่. “การอำนวยการยิง.” หลักสูตรชั้นนายพัน ป. รุ่นที่ 34, 2531. หน้า 1-52.

[2] แผนกวิชาหลักยิง ศูนย์การทหารปืนใหญ่. “ส่วนยิง.” หลักสูตรชั้นนายพัน ป. รุ่นที่ 34, 2531. หน้า 32.

[3] The MIT Press. "PVM:Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing." Massachusetts Institute of Technology, 1994.

[4] G.A.Geist. "PVM a Platform for Portable Distributed Computing." In Proceeding of the High erformance Computing Conference'94, September 1994., pp. 15-12

[5] J. Casas, D. Clark, R. Konuru, S. Otto, R. Prouty, and J. Walpole. "MPVM:A Migration Transparent Version of PVM." Technical report, Oregon Graduate Institute of Science & Technology. 1995.

[6] Clay Breshears. "A Beginer's Guide to PVM Parallel Virtual Machine." University of Tennessee Knoxville, February 1995.

[7] G.A.Geist,and V.S. Sunderam, "Network Based Concurrent Computing on the PVM System." Concurroncy:Practice & Experience, june 1992.

[8] Dror G.Feitelson, Larry Rudolph. “Parallel Job Scheduling: Issues and Approaches.” Technical report, Institute of Computer Science The Hebrew University Jerusalem, Israel. 1995.

[9] Virgilio Augusto, Fermandes Almeida, eduardo Ferreira Loures. “Scheduling Parallel Jobs on a Cluster of Heterogeneous Workstations.” In Proceeding of the High Performance Computing Conference'94, September 1994., pp. 103-108

[10] Ursula Maier, Georg Stellner. “Distributed Resource Management for Parallel Applications in Networks of Workstations.” Technical report, Institute of Informatik Technische Universitat Munchen. 1997.

[11] Ursula Maier, Georg Stellner, Ivan Zoraja. “Resource Allocation, Scheduling and Load Balancing Based on the PVM Resource Manager.” Technical report, Institute of Informatik Technische Universitat Munchen. 1997.

[12] Ursula Maier, Georg Stellner, Ivan Zoraja. "Batch Queuing and Resource Management for PVM Applications in a Network of Workstations." Technical report, Institute of Informatik Technische Universitat Munchen. 1997.

[13] Josef Schule, Axel Brandes. "On the Distribution of Parallel Tasks." Technical report, Rechenzentrum der Technischen Universitat Braunschweig. 1997.

[14] Georg Stellner. "Consistent Checkpoints of PVM Applications." Technical report, Institute of Informatik der Technischen Universitat Munchen. 1995.

[15] Georg Stellner. "Resource Management and Checkpointing for PVM." Technical report, Institute of Informatik der Technischen Universitat Munchen. 1995.

[16] T.E. Anderson, B. N. Bershad, E. D. Lazowsk, and H. M. Levy. "Scheduler activations: effective kernel support for the user-level management of Parallelism." ACM Trans. Comput. Syst. Feb 1992. pp. 53-79

ประวัติผู้เขียน

พันเอก เทพจิต จันทร์คา เกิดเมื่อวันที่ 9 ธันวาคม 2504 ที่จังหวัดสระแก้ว สำเร็จการศึกษาวิทยาศาสตร์บัณฑิต (วทบ.ทบ.) จากโรงเรียนนายร้อยพระจุลจอมเกล้า ปีการศึกษา 2527

ปี พ.ศ. 2527 เข้ารับราชการในตำแหน่ง ผู้ช่วยรองผู้บังคับกองร้อย กองพันทหารปืนใหญ่ที่ 9 ค่ายสุรสีห์ อำเภอเมือง จังหวัดกาญจนบุรี และปัจจุบันดำรงตำแหน่ง รองผู้อำนวยการกองกรมสารบรรณทหารบก ภายในกองบัญชาการกองทัพบก