การคูณเมตริกซ์แบบขนานบนระบบพีซีคลัสเตอร์

PARALLEL MATRIX MULTIPLICATION ON A CLUSTER OF PCs

ไพโรจน์ สมุทรักษ์

PHAIROJ SAMUTRAK

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2548

ISBN 974-15-2079-4

PARALLEL MATRIX MULTIPLICATION ON A CLUSTER OF PCs

PHAIROJ SAMUTRAK

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCE

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2005

ISBN 974-15-2079-4

หัวข้อวิทยานิพนธ์	การคูณเมตริกซ์แบบขนานบนระบบพีซีคลัสเตอร์
นักศึกษา	นายไพโรจน์ สมุทรักษ์
รหัสประจำตัว	46063611
ปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา	วิทยาการคอมพิวเตอร์
พ.ศ.	2548
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ผศ.ดร.จีรพร ศรีสวัสดิ์

บทคัดย่อ

การหาผลคูณของเมตริกซ์เป็นการประมวลผลขั้นพื้นฐานในการหาคำตอบทางวิทยาศาสตร์และวิศวกรรมศาสตร์หลายด้าน เช่น การหาระยะทางที่สั้นที่สุดโดยใช้เมตริกซ์ การวิเคราะห์ออกแบบโครงสร้างเหล็ก และงานทางด้านกราฟิค เป็นต้น ปกติในการหาผลคูณเมตริกซ์แต่ละครั้งต้องใช้ระยะเวลาในการคำนวณเป็นเวลานาน โดยเฉพาะสำหรับเมตริกซ์ขนาดใหญ่ๆ (nxn) เนื่องจากมีความซับซ้อนด้านเวลาเท่ากับ $O(n^3)$ งานวิจัยนี้จึงเสนอการหาผลคูณเมตริกซ์แบบขนานบนระบบพีซีคลัสเตอร์โดยใช้มาตรฐานภาษาเอ็มพีไอ (MPI Standard) เพื่อช่วยลดระยะเวลาในการคำนวณดังกล่าวให้น้อยลง การหาผลคูณเมตริกซ์แบบขนานที่เสนอมี 3 วิธีซึ่งต่างกันตามลักษณะของการจัดแบ่งข้อมูลเพื่อการประมวลผลแบบขนานและการติดต่อสื่อสารที่เหมาะสมรวมถึงนโยบายการเก็บข้อมูลที่ซ้ำซ้อนกันในแต่ละหน่วยประมวลผล คือ 1) การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmm_w (Row-Block Partitioning Matrix Multiplication with replicated data) 2) การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmm_wo (Row-Block Partitioning Matrix Multiplication without replicated data) และ 3) การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี cbmm_w (Checkerboard-Block Partitioning Matrix Multiplication with replicated data) โดยวิธีแรกเป็นวิธีที่มีผู้เสนอไว้แล้วบนระบบพีซีคลัสเตอร์ ส่วนสองวิธีหลังเป็นวิธีที่เสนอขึ้นในวิทยานิพนธ์ วิธีสุดท้ายเป็นการนำข้อดีของสองวิธีแรกมารวมกัน การวัดประสิทธิภาพของวิธีการคูณเมตริกซ์แบบขนานวิธีต่างๆ ที่เสนอรวมถึงการคูณเมตริกซ์แบบอนุกรม (Sequential Matrix Multiplication) ทำโดยการนำค่าเวลา ที่ใช้ในการประมวลผล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และ ประสิทธิภาพ (Efficiency) ของทุกวิธีมาเปรียบเทียบกัน จากผลการทดลองบนระบบพีซีคลัสเตอร์ที่มีขนาด 1 ถึง 9 หน่วยประมวลผล พบว่าวิธีใหม่ที่เสนอการหาผลคูณเมตริกซ์แบบขนานด้วยวิธี cbmm_w มีค่าอัตราการเพิ่มขึ้นของความเร็วเข้าใกล้กรณีอุดมคติมาก คือ 3.63 ในกรณีที่จำนวนหน่วยประมวลผลเท่ากับ 4 หน่วยและมีประสิทธิภาพดีกว่า 12% และ 37% เมื่อเปรียบเทียบกับวิธีการหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmm_w และวิธี rbmm_wo ตามลำดับ

Thesis	Parallel Matrix Multiplication on a Cluster of Pcs
Student	Mr. Phairoj Samutrak
Student ID	46063611
Degree	Master of Science
Programme	Computer Science
Year	2005
Thesis Advisor	Asst. Prof. Dr. Jeeraporn Srisawat

ABSTRACT

The matrix multiplication is one of the most common operation and solution of science and engineering applications i.e. an approach of all-pair shortest path with the modified matrix multiplication, steel-structure design, graphic design and computing, etc. Time complexity of the sequential matrix multiplication is $O(n^3)$ for any matrix size nxn. However when the applications need very large matrix size, the sequential matrix multiplication cannot finish in reasonable time and hence parallel matrix multiplication is introduced to reduce the sequential computing time. This study proposes and implements the practical coarse-grained parallel matrix multiplication on the cluster of PCs using MPI (Message Passing Interface) standard. In particular, three parallel matrix multiplication methods are presented, according to data partitioning schemes for decomposing matrix A and matrix B, data communication and replicated data policy namely: 1) rbmm_w (Row-Block Partitioning Matrix Multiplication with replicated data), 2) rbmm_wo (Row-Block Partitioning Matrix Multiplication without replicated data) and 3) cbmm_w (Checkerboard-Block Partitioning Matrix Multiplication with replicated data). The first method is the existing one on the cluster of PCs, while the last two methods are proposed in this thesis. Specially the last one is introduced by combining the advantages of the first two methods. Finally, the system performance of sequential and parallel processing of the matrix multiplication have been compared and evaluated in terms of response time, speedup and efficiency. On the cluster system of size up to 9 processors, the experimental results showed that the introduced cbmm_w strategy yielded nearly ideal speedup, which was 3.63 for P=4. In addition, the cbmm_w method performed the improved performance upto 12% over that of the existing rbmm_w method and 37% over that of the rbmm_wo method.

กิตติกรรมประกาศ

วิทยานิพนธ์นี้มิอาจจะสำเร็จลุล่วงไปได้ด้วยดี หากมิได้รับคำแนะนำ คำชี้แจง ความรู้ และความเอาใจใส่จาก ผศ. ดร. จีรพร ศรีสวัสดิ์ ผู้เป็นอาจารย์ที่ปรึกษา ซึ่งท่านได้สละเวลาให้กับข้าพเจ้าอย่างเต็มที่ จึงใคร่ขอขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ รศ. ดร. วีระ บุญจริง ผศ. ดร. ศรันย์ อินทโกสุม และดร.เฉลิมศักดิ์ เลิศวงศ์เสถียร คณะกรรมการสอบหัวข้อและโครงร่างวิทยานิพนธ์ที่กรุณาให้คำแนะนำตลอดจนข้อชี้แนะ จนในที่สุดทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลงได้

ขอขอบพระคุณ รศ. ดร. มนัส สังวรศิลป์ ผู้อำนวยการสำนักวิจัยและบริการคอมพิวเตอร์ที่ได้สนับสนุนในเรื่องของระบบคลัสเตอร์ เพื่อใช้ในการทดลองในวิทยานิพนธ์ฉบับนี้

ขอขอบพระคุณบิดา มารดา และพี่ๆ ที่สนันสนุนให้ได้เรียนในระดับที่ได้ตั้งใจ อีกทั้งยังได้ดูแลเรื่องค่าใช้จ่ายต่าง ๆ ระหว่างการศึกษาเป็นอย่างดีอีกด้วย

ขอขอบคุณ นายนพรัตน์ พันธุ์เสนา นางสาวจุรีพร บุญนิยม และเพื่อนๆทุกคนที่ให้คำปรึกษาในเรื่องของการใช้งานระบบคลัสเตอร์และช่วยอำนวยความสะดวกในด้านต่างๆ

ขอขอบคุณนางสาวอารีรัตน์ แสงถกล ที่ช่วยให้กำลังใจในการทำวิทยานิพนธ์จนสำเร็จลุล่วงไปด้วยดี

สำหรับคุณงามความดีและประโยชน์อันใดที่เกิดขึ้นจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดา มารดา อาจารย์ทุกท่านซึ่งเป็นที่เคารพรักยิ่ง ตลอดจนญาติพี่น้อง และเพื่อน ๆ ทุกคน

ไพโรจน์ สมุทรักษ์

ตุลาคม 2548

สารบัญ

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันคนเราสามารถทำงานขนาดใหญ่ชิ้นหนึ่งที่ได้รับมอบหมายให้เสร็จเร็วกว่าปกติได้โดยการเพิ่มประสิทธิภาพในการทำงาน ซึ่งสามารถทำได้โดย การทำงานหนักขึ้นกว่าเดิม (Work Harder) หรือใช้วิธีที่เหมาะสมกว่าเดิม (Work Smarter) และในที่สุดถ้าใช้วิธีที่ดีที่สุดและทำงานหนักขึ้นแล้วก็ยังทำงานได้ไม่เร็วดังที่ต้องการหรือทำงานไม่เสร็จตามที่กำหนด วิธีสุดท้ายคือหาผู้ช่วย (Get Help) ปัญหาในการประมวลผลงานขนาดใหญ่ด้วยคอมพิวเตอร์ก็เช่นเดียวกัน เราสามารถเพิ่มประสิทธิภาพของการประมวลผลงานขนาดใหญ่ๆ งานหนึ่งให้เสร็จเร็วขึ้นได้โดยการใช้คอมพิวเตอร์ที่มีความเร็วสูงขึ้นหรือคอมพิวเตอร์รุ่นใหม่ขึ้น (Work Harder) หรือใช้ขั้นตอนวิธีทางคอมพิวเตอร์ (Computer algorithm) ที่ดีกว่า (Work Smarter) แต่ถ้าใช้ขั้นตอนวิธีที่ดีที่สุดบนเครื่องรุ่นใหม่ที่มีอยู่ในปัจจุบันแล้วก็ยังประมวลผลได้ไม่เร็วพอ วิธีสุดท้าย คือ ใช้เครื่องที่มีหลายหน่วยประมวลผล (Get Help) หรือ คอมพิวเตอร์แบบขนาน (Parallel computer) นั่นเอง[6]

คอมพิวเตอร์แบบขนานเป็นระบบคอมพิวเตอร์ชนิดหนึ่งที่เป็นเทคโนโลยีสมัยใหม่ซึ่งมีการออกแบบทางด้านสถาปัตยกรรมระบบคอมพิวเตอร์และมีการศึกษาพัฒนาที่ต่อเนื่องอยู่ตลอดเวลา โดยระบบคอมพิวเตอร์ชนิดนี้ประกอบด้วยหน่วยประมวลผลหลายๆ หน่วยจำนวนมากรวมอยู่ภายในเครื่องเดียวกัน แนวคิดเริ่มแรกในการออกแบบและพัฒนาระบบคอมพิวเตอร์แบบขนาน คือ เพื่อใช้ในการประมวลผลสำหรับงานบางชนิดที่ต้องการสมรรถนะในการประมวลผลสูง โดยเฉพาะการประมวลผลงานทางด้านวิทยาศาสตร์และวิศวกรรมศาสตร์ ซึ่งต้องการผลที่รวดเร็วและทันต่อเวลา ตัวอย่างเช่น การประมวลผลเพื่อการพยากรณ์อากาศ การคำนวณผลภาพถ่ายทางดาวเทียม การวิเคราะห์ผลภาพสแกน 3 มิติทางการแพทย์ การจำลองแบบทางชีววิทยา เป็นต้น

คอมพิวเตอร์แบบขนานไม่ได้ถูกผลิตและใช้งานอย่างแพร่หลาย เนื่องจากต้นแบบแต่ละแบบถูกพัฒนาขึ้นในจำนวนที่จำกัดเพื่อใช้ศึกษาและวิจัยในหน่วยงานวิจัย และเปิดโอกาสให้นักวิจัยรวมถึงผู้ที่เกี่ยวข้องกับงานวิจัยในสถาบันการศึกษาและสถาบันวิจัยใหญ่ๆ ที่มีชื่อเสียงสำหรับประเทศซึ่งเป็นผู้ผลิตเท่านั้น เหตุผลสำคัญที่ทำให้ระบบคอมพิวเตอร์แบบขนานไม่ถูกผลิตออกจำหน่ายในเชิงธุรกิจ เนื่องจากต้นทุนในการผลิตเครื่องแต่ละเครื่องมีราคาสูงและใช้โปรแกรมภาษาแบบขนานที่ออกแบบมาใช้เฉพาะสำหรับแต่ละระบบเท่านั้น ทำให้การเขียนโปรแกรมเพื่อ

การประมวลผลแบบขนานมีความซับซ้อนมากกว่าการเขียนโปรแกรมโดยทั่วๆ ไป นอกจากนี้ ผู้เขียนโปรแกรมแบบขนานจะต้องมีความรู้พื้นฐานทางด้านสถาปัตยกรรมระบบคอมพิวเตอร์แบบขนานแต่ละแบบ จึงจะสามารถเขียนโปรแกรมแบบขนานได้อย่างมีประสิทธิภาพ

ในปัจจุบันมีการพัฒนาระบบคอมพิวเตอร์แบบขนานอีกชนิดหนึ่งด้วยต้นทุนที่ไม่แพง เรียกว่า "ระบบคลัสเตอร์ (Cluster System)" ซึ่งเป็นระบบคอมพิวเตอร์ที่ได้รับความนิยมสูงเพราะสามารถประมวลผลได้ทั้งแบบขนานและแบบทั่วๆ ไป ระบบคลัสเตอร์นี้เป็นระบบที่มีการนำเครื่องคอมพิวเตอร์ส่วนบุคคล (Personal Computer : PC) หรือเวิร์กสเตชั่น (Workstation) ที่มีประสิทธิภาพสูงหลายๆ เครื่องมาเชื่อมต่อกันด้วยเครือข่าย (Network) ที่มีความเร็วสูง ระบบคอมพิวเตอร์ดังกล่าวได้รับความสนใจอย่างกว้างขวางเนื่องจากสามารถพัฒนาขึ้นใช้ได้เองในห้องทดลองของหน่วยงานวิจัยทางคอมพิวเตอร์ด้วยราคาที่ไม่สูงมาก นอกจากนั้นการใช้งานและการพัฒนาโปรแกรมแบบขนานบนระบบคลัสเตอร์มีความซับซ้อนน้อยกว่าการพัฒนาโปรแกรมแบบขนานบนระบบคอมพิวเตอร์แบบขนานที่กล่าวถึงตั้งแต่ต้น โดยโปรแกรมภาษาแบบขนานสำหรับระบบคอมพิวเตอร์ชนิดนี้เป็นโปรแกรมภาษาที่ใช้ได้กับทุกระบบคลัสเตอร์ เช่น PVM (Parallel Virtual Machines) [7] และ MPI (Message Passing Interface) [7] [14] ซึ่งมีลักษณะการเขียนโปรแกรมในแบบ SPMD (Single Program over Multiple Data) หรือ MPMD (Multiple Programs over Multiple Data) ที่ง่ายกว่าโปรแกรมในแบบ SIMD (Single Instruction over Multiple Data) หรือ MIMD (Multiple Instructions over Multiple Data) บนระบบคอมพิวเตอร์แบบขนานทั่วๆ ไป

งานวิจัยนี้ได้เสนอการประยุกต์ใช้ขั้นตอนวิธีแบบขนานบนระบบพีซีคลัสเตอร์ โดยเน้นเรื่องการคูณของเมตริกซ์ (Matrix Multiplication) ที่มีขนาดใหญ่ เนื่องจากการคูณของเมตริกซ์เป็นส่วนประกอบในงานที่สำคัญๆ หลายด้าน ตัวอย่างเช่น งานทางด้านวิศวกรรม มีการวิเคราะห์หาคำตอบของสมการเชิงเส้นที่ประกอบด้วยตัวแปรเป็นจำนวนมากซึ่งเมตริกซ์เป็นวิธีการหนึ่งที่ช่วยในการคำนวณหาคำตอบดังกล่าว ดังนั้นถ้าสามารถลดระยะเวลาในการหาผลคูณเมตริกซ์จะทำให้การหาค่าคำตอบในสมการดังกล่าวใช้เวลาลดลงตามไปด้วย อีกทั้งยังสามารถรองรับการหาผลคูณของเมตริกซ์ที่มีขนาดใหญ่มากๆ ได้ ตัวอย่างอีกตัวอย่างที่ประยุกต์ใช้การคูณเมตริกซ์ในงานทางด้านวิศวกรรม คือ การวิเคราะห์โครงสร้างของโครงถักเหล็ก (Steel Truss) ซึ่งมีตัวแปรของสมการเป็นจำนวนมากทำให้ต้องใช้เวลาในการหาคำตอบมากด้วยเช่นกัน และตัวอย่างที่น่าสนใจอีกตัวอย่างในการประยุกต์ใช้การคูณเมตริกซ์ ในงานคำนวณเกี่ยวกับเส้นทางการติดต่อสื่อสาร เช่น อินเตอร์เน็ต คือ การหาระยะทางที่สั้นที่สุดของทุกจุดในระบบ (All Pair Shortest Path) เนื่องจากปกติมีจำนวนจุดหรือคอมพิวเตอร์เป็นจำนวนมาก ก็จะต้องใช้ระยะเวลาในการหาระยะที่สั้นที่สุด

มากด้วย ดังนั้นการหาผลคูณเมตริกซ์แบบขนานที่เร็วขึ้นและการนำไปประยุกต์ใช้ในเรื่องต่างๆ ดังกล่าวจะเป็นการลดเวลาในการประมวลผลลงได้

การหาผลคูณของเมตริกซ์แบบขนานสำหรับวิทยานิพนธ์นี้เป็นการประยุกต์เพื่อประมวลผลบนระบบพีซีคลัสเตอร์โดยนำเสนอวิธีการหาผลคูณของเมตริกซ์แบบขนานทั้งหมด 3 วิธี ซึ่งแตกต่างกันตามลักษณะของการจัดแบ่งข้อมูลและการใช้ข้อมูลที่ซ้ำซ้อนกันในแต่ละหน่วยประมวลผล คือ

1) การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmm_w
(Row-Block Partitioning Matrix Multiplication with replicated data)
2) การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmm_wo
(Row-Block Partitioning Matrix Multiplication without replicated data)
3) การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี cbmm_w
(Checkerboard-Block Partitioning Matrix Multiplication with replicated data)

การวัดประสิทธิภาพและการเปรียบเทียบวิธีต่างๆ ที่นำเสนอจะแสดงโดยใช้เวลาทั้งหมดที่ใช้ในการคูณเมตริกซ์ (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency)

1.2 วัตถุประสงค์ของการศึกษา

งานวิจัยนี้มีวัตถุประสงค์เพื่อศึกษาวิธีการหาผลคูณของเมตริกซ์แบบขนาน (Parallel matrix multiplication) พร้อมนำเสนอวิธีการเพิ่มประสิทธิภาพของการหาผลคูณของเมตริกซ์แบบขนานบนระบบพีซีคลัสเตอร์ ดังนี้

1) ลดระยะเวลาในการหาผลคูณของเมตริกซ์แบบอนุกรมที่ใช้หนึ่งหน่วยประมวลผลโดยการใช้หน่วยประมวลผลเพิ่มขึ้นและประมวลผลแบบขนาน ในกรณีที่เมตริกซ์มีขนาดใหญ่ๆ

2) ลดการใช้ข้อมูลในการหาผลคูณของเมตริกซ์แบบขนานที่ซ้ำซ้อนกันของแต่ละหน่วยประมวลผล

3) ทำการเปรียบเทียบประสิทธิภาพของการหาผลคูณของเมตริกซ์แบบขนาน เพื่อหาข้อดีและข้อจำกัดของวิธีต่างๆ ที่เสนอ

1.3 สมมติฐานการศึกษา

1) การหาผลคูณของเมตริกซ์แบบขนานโดยใช้จำนวนหน่วยประมวลผลเพิ่มขึ้น สำหรับเมตริกซ์ขนาดใหญ่เป็นการช่วยให้เวลาในการประมวลผลในการหาผลคูณของเมตริกซ์ลดลง

2) การหาผลคูณของเมตริกซ์แบบขนานมีประสิทธิภาพมากกว่าการหาผลคูณของเมตริกซ์แบบอนุกรม ในกรณีที่เมตริกซ์มีขนาดใหญ่ ๆ

1.4 ขอบเขตการวิจัย

วิทยานิพนธ์นี้มีขอบเขตของการวิจัย ดังต่อไปนี้

ส่วนที่ 1 ทำการศึกษาการหาผลคูณของเมตริกซ์แบบขนาน โดยขั้นแรกเป็นการลดระยะเวลาในการหาผลคูณของเมตริกซ์แบบอนุกรม ในกรณีที่เมตริกซ์มีขนาดใหญ่ๆ โดยใช้ข้อมูลที่ซ้ำซ้อนกันบ้างเพื่อให้มีการติดต่อสื่อสารระหว่างหน่วยประมวลผลเพื่อแลกเปลี่ยนข้อมูลน้อยที่สุดหรือไม่มีเลย และขั้นต่อมาเป็นการลดการใช้ข้อมูลที่ซ้ำซ้อนกันหรือไม่ใช้ข้อมูลซ้ำซ้อนกันเลยในการหาผลคูณของเมตริกซ์แบบขนานของแต่ละหน่วยประมวลผล แต่ต้องยอมเสียเวลาในการติดต่อสื่อสารระหว่างหน่วยประมวลผลเพื่อแลกเปลี่ยนข้อมูลระหว่างกัน

ส่วนที่ 2 ทำการพัฒนาโปรแกรม โดยนำทฤษฎีและวิธีการที่ได้ทำการศึกษาในส่วนแรกมาพัฒนา โดยใช้ภาษาซี (C language) และมาตรฐานภาษาเอ็มพีไอ (MPI Standard) ที่สนับสนุนการโปรแกรมแบบขนานบนระบบพีซีคลัสเตอร์เพื่อนำเสนอผลลัพธ์พร้อมทั้งวิเคราะห์และสรุปผล

1.5 ขั้นตอนการศึกษาและการดำเนินงานวิจัย

วิทยานิพนธ์นี้มีขั้นตอนการศึกษาและการดำเนินงานวิจัย ดังนี้

1) ศึกษาขั้นตอนวิธีการทางคอมพิวเตอร์ (Computer algorithm) ของการหาผลคูณของเมตริกซ์แบบขนาน

2) ศึกษางานวิจัยที่เกี่ยวข้องกับการหาผลคูณของเมตริกซ์แบบขนาน

3) ทำการตั้งสมมติฐานโดยคาดว่าวิธีการหาผลคูณของเมตริกซ์ที่ได้นำเสนอเมื่อใช้จำนวนหน่วยประมวลผลเพิ่มขึ้น จะเป็นการช่วยให้เวลาในการประมวลผลในการหาผลคูณของเมตริกซ์ลดลงโดยเฉพาะกรณีที่เมตริกซ์มีขนาดใหญ่ๆ และลดการใช้ข้อมูลที่ซ้ำซ้อนกันซึ่งเป็นการลดพื้นที่ในหน่วยความจำของแต่ละหน่วยประมวลผล และสรุปได้ว่าการหาผลคูณของเมตริกซ์แบบขนานมีประสิทธิภาพมากกว่าการหาผลคูณของเมตริกซ์แบบอนุกรม ในกรณีที่เมตริกซ์มีขนาดใหญ่ ๆ

4) นำเสนอวิธีการการหาผลคูณของเมตริกซ์แบบขนานที่มีประสิทธิภาพ ดังที่ได้ตั้งสมมติฐานไว้ในข้อ (3)

5) พัฒนาโปรแกรมการหาผลคูณของเมตริกซ์แบบขนานที่นำเสนอโดยใช้โปรแกรมภาษาซี (C Language) บนระบบพีซีคลัสเตอร์ที่ใช้ระบบปฏิบัติการลีนุกซ์ (Linux) และตรงตามมาตรฐานภาษาเอ็มพีไอ (MPI Standard) ที่สนับสนุนการโปรแกรมแบบขนาน

6) วิเคราะห์ผลการทดลองโดยการเปรียบเทียบประสิทธิภาพของการหาผลคูณของเมตริกซ์ จะประเมินผลจากเวลาที่ใช้ในการประมวลผล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency)

7) สรุปผลการทดลอง พร้อมเสนอแนวทางการพัฒนางานวิจัย

8) เขียนวิทยานิพนธ์

1.6 ประโยชน์ที่คาดว่าจะได้รับ

1) สามารถหาผลคูณของเมตริกซ์ กรณีที่มีเมตริกซ์มีขนาดใหญ่ๆ โดยใช้เวลาน้อยลงใช้หน่วยความจำน้อยลงและสามารถนำไปใช้ในการหาคำตอบของสมการที่มีจำนวนตัวแปรมากๆ ได้ต่อไป

2) นำมาใช้เป็นแนวทางในการพัฒนาโปรแกรมคอมพิวเตอร์แบบขนานเพื่อให้สามารถใช้งานได้สะดวกและเข้าใจได้ง่าย

3) นำไปประยุกต์ใช้กับการพัฒนาโปรแกรมด้านต่างๆ เช่น งานด้านธุรกิจ งานด้านเว็บเซิร์ฟเวอร์ งานด้านการพยากรณ์อากาศ คำนวณผลภาพถ่ายทางดาวเทียม การจำลองแบบทางชีววิทยา งานด้านการแพทย์ งานด้านกราฟิคดีไซน์ เป็นต้น

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 คอมพิวเตอร์แบบขนาน (Parallel Computer)

คอมพิวเตอร์แบบขนาน คือ เครื่องคอมพิวเตอร์ชนิดหนึ่งที่ประกอบด้วยหน่วยประมวลผลหลายๆ หน่วยจำนวนมากรวมอยู่ภายในเครื่องเดียวกันหรือระบบเดียวกัน เพื่อใช้ประมวลผลพร้อมๆ กัน

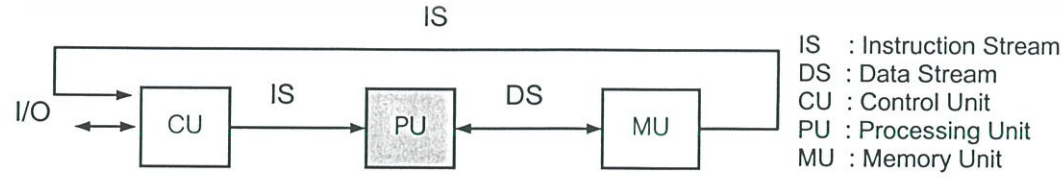
สถาปัตยกรรมคอมพิวเตอร์ทั่วๆ ไป สามารถจำแนกได้หลายรูปแบบโดยมีรายละเอียดดังนี้

2.1.1 สถาปัตยกรรมคอมพิวเตอร์ (Computer Architecture)

ในปี 1972 Flynn นักวิจัยทางด้านสถาปัตยกรรมคอมพิวเตอร์ ได้จำแนกสถาปัตยกรรมของระบบคอมพิวเตอร์เป็น 4 แบบ [9][15] คือ

1) สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอสไอเอสดี
SISD :Single Instruction stream over Single Data stream

สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอสไอเอสดีเป็นการใช้หนึ่งชุดคำสั่งกับหนึ่งชุดข้อมูล ซึ่งใช้กับคอมพิวเตอร์ทั่วไปที่มีหนึ่งหน่วยประมวลผล โดยแต่ละคำสั่งการคำนวน (IS : Instruction Stream) จะโหลดจากหน่วยความจำ (MU : Memory Unit) โดยหน่วยควบคุม (CU : Control Unit) และส่งไปให้หน่วยประมวลผล (PU : Processing Unit) สำหรับคำสั่งนั้น ส่วนข้อมูลสามารถรับและส่งจากหน่วยความจำได้ทีละชุดข้อมูล

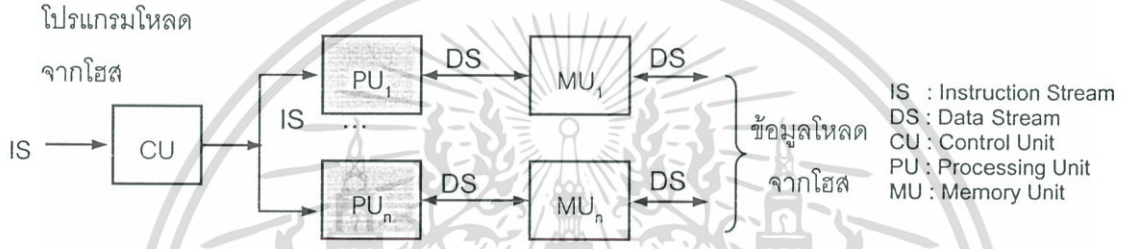


รูปที่ 2.1 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอสไอเอสดี

2) สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอสไอเอ็มดี

SIMD :Single Instruction stream over Multiple Data stream

สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอสไอเอ็มดีเป็นการใช้หนึ่งชุดคำสั่งกับหลายชุดข้อมูล คือ ระบบคอมพิวเตอร์แบบขนานชนิดหนึ่งซึ่งใช้ประมวลผลงานขนานแบบเฉพาะงาน ซึ่งคำสั่งในโปรแกรมจะโหลดจากโฮส (Host) ทีละคำสั่งจากหน่วยควบคุม (CU) และส่งไปให้หน่วยประมวลผล (PU) สำหรับคำสั่งนั้น ส่วนข้อมูลสามารถรับและส่งจากหน่วยความจำหลายๆ หน่วยได้พร้อมๆ กัน เพื่อประมวลผลพร้อมๆ กัน

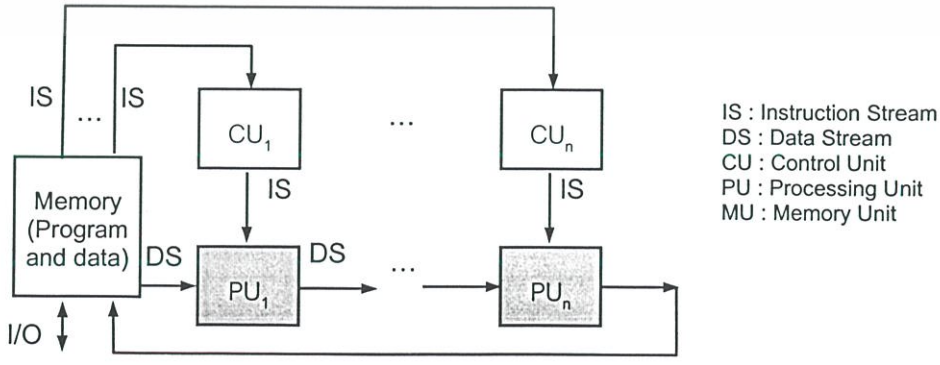


รูปที่ 2.2 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอสไอเอ็มดี

3) สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอ็มไอเอสดี

MISD : Multiple Instruction stream over Single Data stream

สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอ็มไอเอสดีเป็นการใช้หลายชุดคำสั่งกับหนึ่งชุดข้อมูล คือ ระบบคอมพิวเตอร์แบบขนานชนิดหนึ่งซึ่งใช้ประมวลผลงานขนานแบบเฉพาะงาน ดังนั้นแต่ละหน่วยประมวลผล (PU) จะมีหน่วยควบคุม (CU) ของตัวเอง ซึ่งทำหน้าที่โหลดคำสั่งที่ต่างกันได้พร้อมๆ กันเพื่อประมวลผลพร้อมๆ กัน ส่วนข้อมูลจะรับและส่งจากหน่วยความจำกลาง

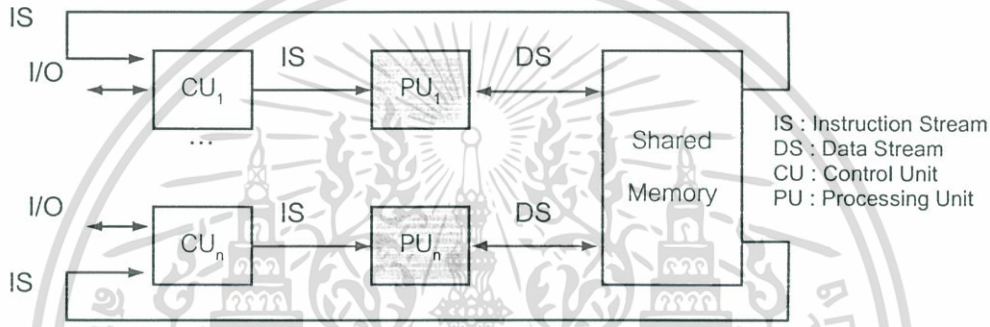


รูปที่ 2.3 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอ็มไอเอสดี

4) สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอ็มไอเอ็มดี

MIMD : Multiple Instruction stream over Multiple Data stream

สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอ็มไอเอ็มดีเป็นระบบคอมพิวเตอร์แบบขนานชนิดหนึ่งที่ได้รับความนิยมมากที่สุดสำหรับใช้ประมวลผลงานขนานแบบทั่วๆ ไป เป็นการใช้หลายชุดคำสั่งกับหลายชุดข้อมูลทำงานได้พร้อมๆ กัน โดยหลายๆ หน่วยประมวลผล (PUs) ซึ่งมีหน่วยควบคุม (CU) ของตัวเองทำหน้าที่ในการโหลดคำสั่ง (IS) ส่งให้หน่วยประมวลผล (PUs) และแต่ละหน่วยประมวลผล (PUs) สามารถรับและส่งข้อมูลของตัวเองจากหน่วยความจำร่วม (Shared Memory) ได้พร้อมๆ กัน



รูปที่ 2.4 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบเอ็มไอเอ็มดี

2.1.2 สถาปัตยกรรมคอมพิวเตอร์แบบขนาน (Parallel Computer Architecture)

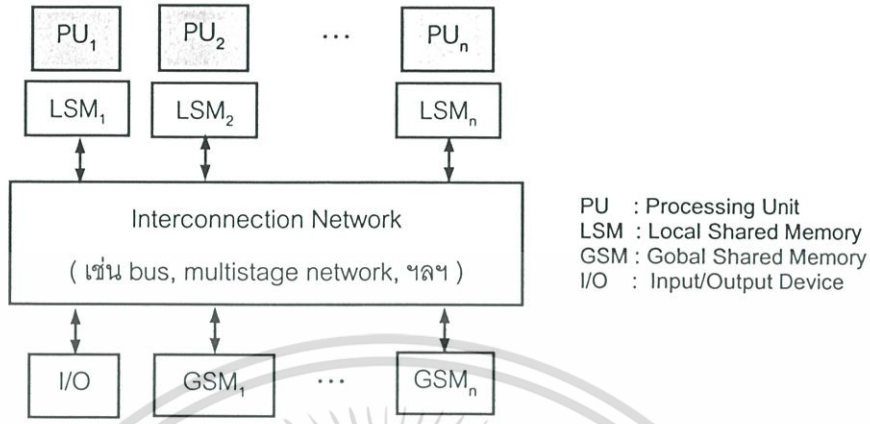
สถาปัตยกรรมคอมพิวเตอร์แบบขนาน สามารถแยกประเภทตามโครงสร้างของหน่วยความจำโดยสามารถแบ่งได้เป็น 2 แบบ คือ สถาปัตยกรรมแบบขนานที่ใช้หน่วยความจำร่วมกันและสถาปัตยกรรมแบบขนานที่มีการกระจายหน่วยความจำ โดยมีรายละเอียดดังนี้ [15]

1) สถาปัตยกรรมแบบขนานที่ใช้หน่วยความจำร่วมกัน

Shared-Memory Parallel Architecture

สถาปัตยกรรมของระบบคอมพิวเตอร์แบบขนานชนิดนี้ถูกออกแบบมาให้มีการใช้หน่วยความจำร่วมกัน[9][11] โดยที่หน่วยประมวลผลทั้งหมดในระบบ (Processors) สามารถเข้าถึงข้อมูลในหน่วยความจำ (Memory) ของระบบได้ทุกที่พร้อมๆ กัน ทำให้การพัฒนาโปรแกรมทำได้สะดวกมากขึ้น อย่างไรก็ตามจำนวนของหน่วยประมวลผลที่สามารถเข้าถึงข้อมูลในหน่วยความจำนั้นได้ถูกจำกัดด้วยขนาดของเส้นทางการติดต่อสื่อสาร จากปัญหาดังกล่าวได้มีความพยายามที่จะ

แก้ไขปัญหานี้โดยกำหนดให้แต่หน่วยประมวลผลมีหน่วยความจำส่วนตัว (Local Memory) และมีหน่วยความจำกลาง (Global Memory) ที่ใช้เก็บข้อมูล ดังแสดงในรูปที่ 2.5

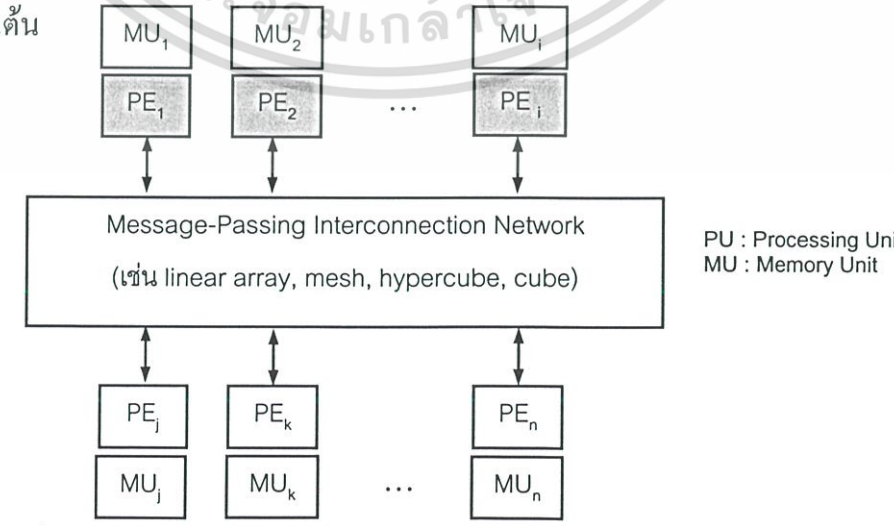


รูปที่ 2.5 สถาปัตยกรรมแบบขนานที่ใช้หน่วยความจำร่วมกัน

2) สถาปัตยกรรมแบบขนานที่มีการกระจายหน่วยความจำ

Distributed-Memory Parallel Architecture

สถาปัตยกรรมของระบบคอมพิวเตอร์แบบขนานชนิดนี้ออกแบบมาให้ใช้หน่วยความจำแบบกระจาย [9][11] คือ หน่วยประมวลผลแต่ละหน่วยจะมีหน่วยความจำส่วนตัว (Local Memory) ที่ไม่ใช้ร่วมกันดังแสดงในรูปที่ 2.6 ดังนั้นในระบบนี้ถ้าหน่วยประมวลผลใดต้องการข้อมูลจากหน่วยประมวลผลอื่นต้องติดต่อสื่อสารกันผ่านระบบส่งผ่านข้อความ (Message-Passing) โดยการพัฒนาโปรแกรมแบบขนานนี้จะทำการรับส่งข้อมูล (Send/Receive) ผ่านไลบรารี (Libraries) ของเอ็มพีไอ (MPI: Message Passing Interface) [7][14] หรือพีวีเอ็ม (PVM: Parallel Virtual Machine) [7] เป็นต้น



รูปที่ 2.6 สถาปัตยกรรมแบบขนานที่มีการกระจายหน่วยความจำ

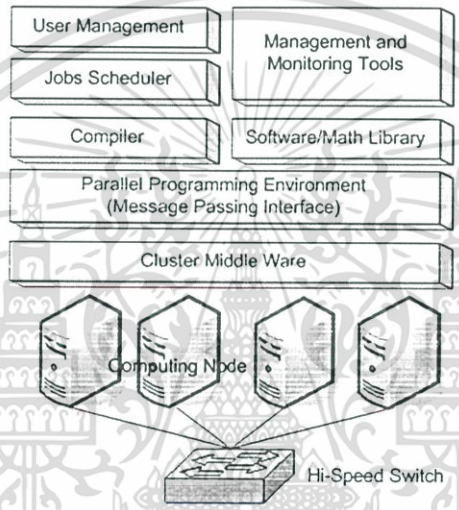
นอกจากนี้แล้วสถาปัตยกรรมคอมพิวเตอร์แบบขนาน ยังสามารถแบ่งแยกเป็นแบบย่อยๆลงไปได้อีกเป็นแบบซิมเมทริกซ์มัลติโปรเซสเซอร์ (Symmetric Multi-Processor: SMP)[9] ซึ่งโดยทั่วไปสถาปัตยกรรมแบบนี้เป็นคลัสเตอร์ (Cluster) ขนาดใหญ่ใช้หน่วยความจำแบบกระจายและมีหลายคลัสเตอร์ย่อย ภายในคลัสเตอร์ย่อยนี้จะจับกลุ่มหน่วยประมวลผลจำนวน 4-8 หน่วยประมวลผลที่มีประสิทธิภาพเท่ากันซึ่งจะใช้หน่วยความจำร่วมกัน ในทางปฏิบัตินั้นคลัสเตอร์แบบเอสเอ็มพี (SMP Cluster) จะเลือกใช้หน่วยความจำแบบกระจาย ทำให้การพัฒนาโปรแกรมแบบเอสเอ็มพีคลัสเตอร์และแบบกระจายหน่วยความจำมีความคล้ายคลึงกันมาก

ระบบคลัสเตอร์ คือ การนำคอมพิวเตอร์หลายๆ เครื่องมาเชื่อมต่อกันเพื่อให้ทำงานเหมือนเครื่องใหญ่เครื่องเดียว ความแตกต่างระหว่างระบบคลัสเตอร์กับระบบ LAN (Local Area Network) คือ ระบบ LAN จะเป็นการเชื่อมต่อคอมพิวเตอร์จำนวนมากเพื่อใช้ทรัพยากรร่วมกัน โดยแต่ละเครื่องยังเป็นเครื่องอิสระ แต่ระบบคลัสเตอร์เป็นการรวมเครื่องหลายหลายเครื่องที่ประสานงานกันอย่างแน่นแฟ้นจนเปรียบเสมือนเป็นเครื่องเดียวกัน ระบบคลัสเตอร์นี้สามารถนำมาประยุกต์ใช้งานได้แทนระบบเซิร์ฟเวอร์ขนาดใหญ่ได้เป็นอย่างดีในราคาที่ถูกกว่า [20]

โครงสร้างของระบบพีซีคลัสเตอร์ทั่วไปจะประกอบด้วยเครื่องพีซีสมรรถนะสูงจำนวนหนึ่งที่นำมาเชื่อมต่อกันด้วยเครือข่ายความเร็วสูง เช่น ระบบ ATM Switch, Fast Ethernet Switch, Myrinet เป็นต้น โดยปกติโครงสร้างพื้นฐานของระบบคลัสเตอร์และระบบ LAN ไม่ต่างกันมาก แต่ระบบคลัสเตอร์จะพึ่งพาซอฟท์แวร์พิเศษที่กระจายงานไปในกลุ่มของคอมพิวเตอร์ที่มารวมกันทำงานเป็นระบบคลัสเตอร์ ดังนั้นคุณสมบัติสำคัญของเทคโนโลยีคลัสเตอร์มีอยู่สามประการ คือ เครือข่ายความเร็วสูง ระบบซอฟท์แวร์ที่สนับสนุนการทำงานแบบคลัสเตอร์และโปรแกรมประยุกต์ที่ใช้ขีดความสามารถดังกล่าว

แนวความคิดของระบบคลัสเตอร์เกิดขึ้นในปี ค.ศ. 1960 โดยบริษัท IBM ต้องการเชื่อมต่อระบบเมนเฟรม (Mainframe) ขนาดใหญ่เข้าด้วยกันเพื่อใช้ในเชิงการค้า แต่ในตอนนั้นระบบคลัสเตอร์ยังไม่เป็นที่รู้จักอย่างแพร่หลายเนื่องจากข้อจำกัดทางด้านราคาและความแพร่หลายของฮาร์ดแวร์ (Hardware) นอกจากนั้นยังขาดเครื่องมือ (Tools) มาตรฐานสำหรับการเชื่อมต่อเครื่องในระบบเข้าด้วยกัน จนในช่วงกลางปี 1980 เป็นช่วงที่เทคโนโลยีระบบคลัสเตอร์สามารถเกิดขึ้นได้ เนื่องจากสามารถสร้างไมโครโปรเซสเซอร์ที่มีประสิทธิภาพได้ และเกิดระบบเครือข่ายความเร็วสูงขึ้น ซึ่งเครื่องคอมพิวเตอร์ในระบบเครือข่ายสามารถสื่อสารกันได้ภายในเวลาไม่เกินหนึ่งมิลลิวินาที นอกจากนั้นความต้องการความเร็วในการประมวลผลงานทางวิทยาศาสตร์และวิศวกรรมศาสตร์ซึ่งมีมากขึ้นอย่างต่อเนื่องก็มีส่วนผลักดันให้เกิดเทคโนโลยีคลัสเตอร์ขึ้น [20]

ระบบคลัสเตอร์ได้เกิดขึ้นอย่างเป็นรูปธรรมในปี 1994 เมื่อ Thomas Sterling และ Don Becker จาก CESDIS (The Center of Excellence in Space Data and Information Sciences) ต้องการที่จะวิเคราะห์ข้อมูลจากอวกาศที่มีจำนวนมากและซับซ้อนซึ่งต้องใช้เครื่องคอมพิวเตอร์ที่สามารถประมวลผลได้เร็วมากๆ และในขณะนั้นมีเพียงเครื่องซุปเปอร์คอมพิวเตอร์ซึ่งมีราคาแพงแต่เนื่องจากงบประมาณที่มีอยู่จำกัดทำให้ Stering และ Becker มีแนวความคิดที่จะใช้ระบบคลัสเตอร์มาทำการประมวลโดยได้สร้างระบบคลัสเตอร์จากเครื่องคอมพิวเตอร์ซึ่งใช้ชิพ Intel DX4 เป็นหน่วยประมวลผลกลาง (CPU) และส่วนประกอบอื่นๆ ซึ่งสามารถหาได้ง่ายจากท้องตลาดและได้ตั้งชื่อระบบคลัสเตอร์นี้ว่า Beowulf [20] ดังแสดงในรูปที่ 2.7



รูปที่ 2.7 โครงสร้างระบบคลัสเตอร์[20]

ระบบคลัสเตอร์ สามารถจัดแบ่งเป็น 2 ชนิดตามลักษณะที่ต่อระบบเข้ากับเครือข่ายความเร็วสูงภายนอก (High-Speed Network) คือ คลัสเตอร์แบบปิดและคลัสเตอร์แบบเปิด

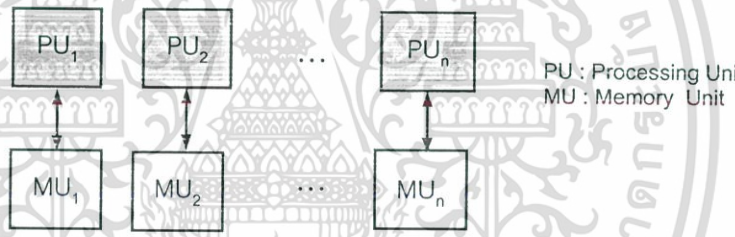
1) คลัสเตอร์แบบปิด ระบบนี้จะต่อคลัสเตอร์ผ่านเกตเวย์ที่ซ่อนทั้งระบบจากโลกภายนอก ข้อดีก็คือ มีความปลอดภัยสูงและจะใช้อินเตอร์เน็ตแอดเดรสเพียงแอดเดรสเดียวเท่านั้น ข้อเสียคือแต่ละโหนดในระบบไม่สามารถช่วยกันบริการข้อมูลกับโลกภายนอกได้ ดังนั้นคลัสเตอร์แบบปิดจึงเหมาะกับการใช้ทำงานคำนวณขนาดใหญ่ภายในเท่านั้น เช่น การคำนวณภาพถ่ายทางดาวเทียม การคำนวณทางชีววิทยา เป็นต้น

2) คลัสเตอร์แบบเปิด ระบบนี้จะต่อกับเน็ตเวิร์กภายนอกโดยตรง ทำให้ผู้ใช้เข้าถึงทุกโหนดในระบบคลัสเตอร์ได้โดยตรง ข้อเสียก็คือ ความปลอดภัยจะต่ำลงมากเพราะต้องคอยดูแลทุกเครื่องในระบบ และยังต้องการหมายเลขอินเตอร์เน็ตแอดเดรสจำนวนมาก แต่ข้อดีก็คือระบบ

สามารถช่วยกันบริการข้อมูลได้ ดังนั้นคลัสเตอร์แบบเปิดจึงเหมาะกับงานบริการข่าวสารจำนวนมากเช่นใช้เป็นระบบเซิร์ฟเวอร์ สำหรับ WWW หรือ ftp ที่ขยายตัวได้ ระบบนี้น่าจะเหมาะกับการทำเป็นฐานข้อมูลขนาดใหญ่ด้วย แต่ในขณะนี้ยังไม่มีซอฟท์แวร์ที่จะประยุกต์เป็นฐานข้อมูลที่ใช้ขีดความสามารถของระบบ คลัสเตอร์ได้

2.1.3 แบบจำลอง PRAM (Parallel Random Access Machine Model)

แบบจำลอง PRAM เป็นแบบจำลองการต่อแบบขนานในอุดมคติ (Idealized Parallel System) เพราะสมมติให้มีค่าเวลาในการติดต่อระหว่างหน่วยประมวลผลเท่ากับศูนย์ ซึ่งส่วนมากใช้สำหรับการพัฒนาขั้นตอนวิธีการทางคอมพิวเตอร์แบบขนาน (Parallel Algorithm) วิเคราะห์ขนาด (Scalability) และความซับซ้อน (Complexity) ในแบบจำลอง PRAM สมมติให้มีจำนวนหน่วยประมวลผลเท่ากับ n หน่วยประมวลผลและใช้หน่วยความจำร่วมกันโดยทุกหน่วยประมวลผลสามารถอ่านค่าจากหน่วยความจำคนละที่พร้อมๆ กันได้และสามารถเขียนไปยังหน่วยความจำคนละที่พร้อมๆ กันได้ ดังแสดงในรูปที่ 2.8



รูปที่ 2.8 แบบจำลอง PRAM

ในกรณีที่หน่วยประมวลผลหลายๆ หน่วยต้องการอ่านหรือเขียนข้อมูลบนหน่วยความจำ ณ จุดเดียวกัน จะต้องมีกลยุทธ์ในการจัดการ ดังนั้นแบบจำลอง PRAM ยังสามารถแยกประเภทได้ 4 ประเภท ตามลักษณะการอนุญาตให้มีการอ่านหรือเขียน ณ จุดเดียวกันโดยหลายๆ หน่วยประมวลผลได้ดังนี้

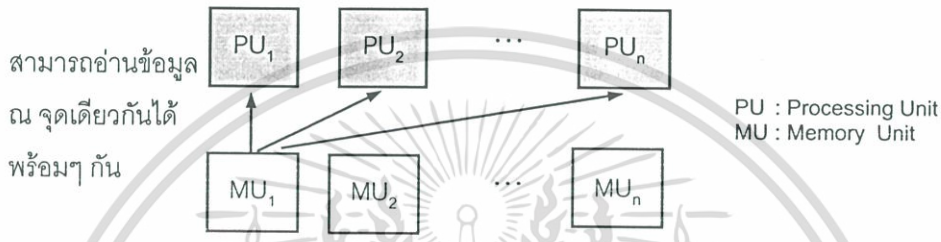
1) แบบจำลอง PRAM แบบอีอาร์อีดับเบิลยู
(EREW :Exclusive Read Exclusive Write)

แบบจำลอง PRAM แบบอีอาร์อีดับเบิลยู กรณีนี้อนุญาตให้เพียงหนึ่งหน่วยประมวลผลสามารถอ่านหรือเขียนข้อมูลในหน่วยความจำ ณ จุดๆ เดียวกันในหนึ่งรอบเวลา (Cycle) ถ้าหน่วยประมวลผลอื่นต้องการอ่านหรือเขียนข้อมูลต้องรอรอบเวลาถัดไป

2) แบบจำลอง PRAM แบบซีอาร์อีดับเบิลยู

(CREW :Concurrent Read Exclusive Write)

แบบจำลอง PRAM แบบซีอาร์อีดับเบิลยู กรณีนี้อนุญาตให้ทุกหน่วยประมวลผลสามารถอ่านข้อมูลในหน่วยความจำ ณ จุดๆ เดียวกันในหนึ่งรอบเวลา (Cycle) แต่ไม่สามารถเขียนข้อมูล ณ จุดๆ เดียวกันได้พร้อมกันในหนึ่งรอบเวลาได้ถ้าหน่วยประมวลผลอื่นต้องการเขียนข้อมูลต้องรอรอบเวลาถัดไป ดังแสดงในรูปที่ 2.9

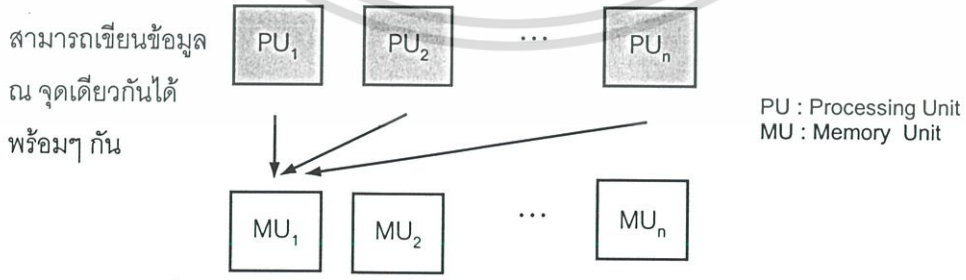


รูปที่ 2.9 แบบจำลอง PRAM แบบซีอาร์อีดับเบิลยู

3) แบบจำลอง PRAM แบบอีอาร์ซีดับเบิลยู

(ERCW :Exclusive Read Concurrent Write)

แบบจำลอง PRAM แบบอีอาร์ซีดับเบิลยู กรณีนี้อนุญาตให้ทุกหน่วยประมวลผลสามารถเขียนข้อมูลในหน่วยความจำ ณ จุดๆ เดียวกันในหนึ่งรอบเวลา (Cycle) แต่ไม่สามารถอ่านข้อมูล ณ จุดๆ เดียวกันได้พร้อมกันในหนึ่งรอบเวลา ถ้าหน่วยประมวลผลอื่นต้องการอ่านข้อมูลต้องรอรอบเวลาถัดไป ดังแสดงในรูปที่ 2.10

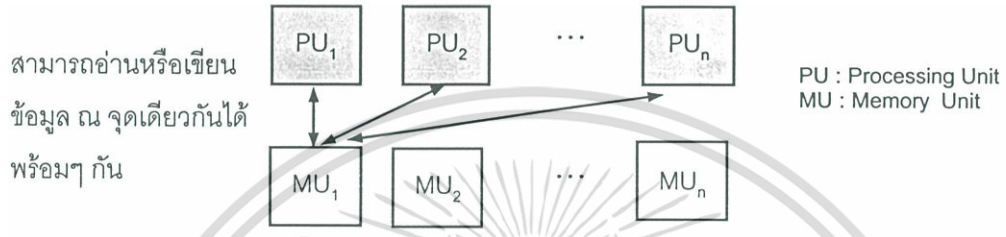


รูปที่ 2.10 แบบจำลอง PRAM แบบอีอาร์ซีดับเบิลยู

4) แบบจำลอง PRAM แบบซีอาร์ซีดับเบิลยู

(CRCW : Concurrent Read Concurrent Write)

แบบจำลอง PRAM แบบซีอาร์ซีดับเบิลยู กรณีนี้อนุญาตให้ทุกๆ หน่วยประมวลผลสามารถอ่านหรือเขียนข้อมูลในหน่วยความจำ ณ จุดๆ เดียวกันในหนึ่งรอบเวลา (Cycle) ได้พร้อมๆ กันซึ่งเป็นแบบที่มีความยืดหยุ่นมากที่สุด ดังแสดงในรูปที่ 2.11



รูปที่ 2.11 แบบจำลอง PRAM แบบซีอาร์ซีดับเบิลยู

2.2 โครงสร้างการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Interconnection Network)

การติดต่อระหว่างหน่วยประมวลผลสำหรับสถาปัตยกรรมแบบขนานที่มีกระจายหน่วยความจำมีหลากหลายชนิด เช่น การติดต่อแบบดาว (Star Connected) การติดต่อแบบอะเรย์เชิงเส้น (Linear Array) การติดต่อแบบเมสช์ (Mesh) และการติดต่อแบบไฮเปอร์คิว (Hypercube) เป็นต้น

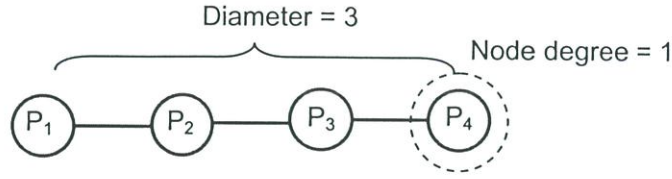
ในการศึกษาขั้นตอนวิธีทางคอมพิวเตอร์แบบขนานและการพัฒนาโปรแกรมแบบขนานที่มีประสิทธิภาพ นักวิจัยต้องทราบโครงสร้างการติดต่อสื่อสารระหว่างหน่วยประมวลผล คุณสมบัติพื้นฐานของแต่ละโครงสร้าง เช่น เส้นผ่าศูนย์กลางของโครงข่าย (Diameter of network) จำนวนช่องทางติดต่อของแต่ละโหนด (Node degree) เป็นต้น

<u>นิยามคำศัพท์</u>

เส้นผ่าศูนย์กลางโครงข่าย (Diameter of network) คือ จำนวนการเชื่อมต่อ (Connection) ที่สั้นที่สุดระหว่างโหนด 2 โหนดที่อยู่ไกลกันที่สุดในระบบ

โหนดดีกรี (Node degree) คือ จำนวนช่องทางที่ใช้ติดต่อสื่อสาร (Channel) ของโหนดแต่ละโหนดในระบบ

ตัวอย่างเช่น โครงสร้างการติดต่อสื่อสารชนิดหนึ่งซึ่งมีจำนวนโหนดทั้งหมดเท่ากับ 4 โหนด ดังแสดงในรูปที่ 2.12



รูปที่ 2.12 ตัวอย่างโครงสร้างการติดต่อสื่อสารชนิดหนึ่งและการหาเส้นผ่าศูนย์กลางโครงข่ายและโหนดดีกรี

จากรูปจะได้ ขนาดเส้นผ่าศูนย์กลางโครงข่ายเท่ากับ 3 ซึ่งวัดจากโหนดแรก (P_1) และโหนดสุดท้าย (P_4) ในระบบ

- จำนวนโหนดดีกรีเท่ากับ 1 สำหรับ P_1 และ P_4 สำหรับโหนดหัวและท้าย
- จำนวนโหนดดีกรีเท่ากับ 2 สำหรับ P_2 และ P_3 สำหรับโหนดกลาง

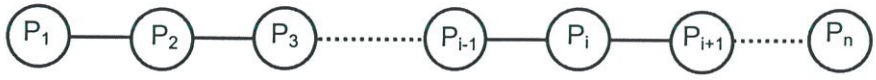
ในหัวข้อนี้เน้นการเสนอรายละเอียดของโครงสร้างการติดต่อระหว่างหน่วยประมวลผล 3 แบบที่นิยมใช้ในการพัฒนาระบบคอมพิวเตอร์แบบขนานและขั้นตอนวิธีการทางคอมพิวเตอร์แบบขนาน[9] คือ

2.2.1 โครงสร้างการติดต่อแบบอะเรย์เชิงเส้น (Linear Array Network)

โครงสร้างการติดต่อแบบอะเรย์เชิงเส้น มีจำนวน n หน่วยประมวลผล มีหน่วยประมวลผล P_1 ถึงหน่วยประมวลผล P_n เชื่อมต่อกันใน 1 มิติ ดังแสดงในรูปที่ 2.13

-หน่วยประมวลผล P_i ติดต่อกับหน่วยประมวลผล P_{i-1} (ทางซ้าย) และหน่วยประมวลผล P_{i+1} (ทางขวา) เมื่อ i = 2, 3, …, n-1

- หน่วยประมวลผล P_1 ติดต่อกับหน่วยประมวลผล P_2 (ทางขวา) เท่านั้น
- หน่วยประมวลผล P_n ติดต่อกับหน่วยประมวลผล P_{n-1} (ทางซ้าย) เท่านั้น



รูปที่ 2.13 โครงสร้างการติดต่อแบบอะเรย์เชิงเส้น

ขนาดของโครงสร้างหรือจำนวนโหนดทั้งหมดในโครงสร้าง เท่ากับ n

ขนาดเส้นผ่าศูนย์กลางโครงข่าย (Diameter of network) เท่ากับ n-1

จำนวนโหนดดีกรี (Node Degree)

- จำนวนโหนดดีกรีเท่ากับ 1 สำหรับโหนดแรกและโหนดสุดท้าย
- จำนวนโหนดดีกรีเท่ากับ 2 สำหรับโหนดกลาง

2.2.2 โครงสร้างการติดต่อแบบเม็สช์ (Mesh Network)

โครงสร้างการติดต่อแบบเม็สช์ (Mesh) เป็นการติดต่อใน 2 มิติ โดยมีจำนวน mxn หน่วยประมวลผล ดังแสดงในรูปที่ 2.14

กำหนดให้ $P_{i,j}$ คือ หน่วยประมวลผลแถวที่ i และหลักที่ j

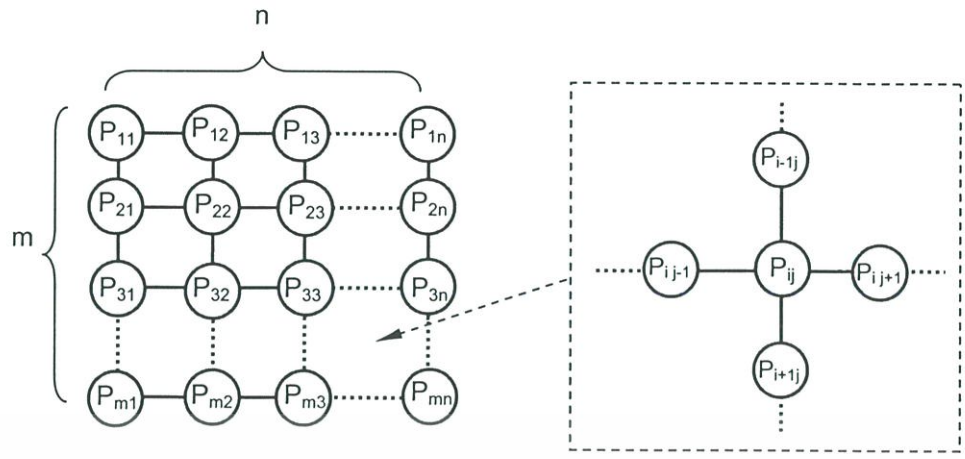
- สำหรับ $P_{i,j}$ ติดต่อกับ $P_{i\pm1,j}$ และ $P_{i,j\pm1}$ เมื่อ i = 2,3, … , m-1 และ j = 2,3, … , n-1
- ส่วนหน่วยประมวลผลแถวที่ 1 และหลักที่ j ($P_{1,j}$) ติดต่อกับ $P_{2,j}$ และ $P_{1,j\pm1}$
  เมื่อ j = 2,3, … , n-1
- ส่วนหน่วยประมวลผลแถวที่ i และหลักที่ 1 ($P_{i,1}$) ติดต่อกับ $P_{i,2}$ และ $P_{i\pm1,i}$
  เมื่อ i = 2,3, … , m-1
- ส่วนหน่วยประมวลผลแถวที่ m และหลักที่ j ($P_{m,j}$) ติดต่อกับ $P_{m-1,j}$ และ $P_{m,j\pm1}$
  เมื่อ j = 2,3, … , n-1
- ส่วนหน่วยประมวลผลแถวที่ i และหลักที่ n ($P_{i,n}$) ติดต่อกับ $P_{i,n-1}$ และ $P_{i\pm1,n}$
  เมื่อ i = 2,3, … , m-1
- ส่วนหน่วยประมวลผล 4 โหนดมุมจะติดต่อเพียง 2 โหนด ดังนี้

  $P_{1,1}$ ติดต่อกับ $P_{1,2}$ และ $P_{2,1}$

  $P_{1,n}$ ติดต่อกับ $P_{1,n-1}$ และ $P_{2,n}$

  $P_{m,1}$ ติดต่อกับ $P_{m-1,1}$ และ $P_{m,2}$

  $P_{m,n}$ ติดต่อกับ $P_{m-1,n}$ และ $P_{m,n-1}$



รูปที่ 2.14 โครงสร้างการติดต่อแบบเม็สช์

ขนาดของโครงสร้างหรือจำนวนโหนดทั้งหมดในโครงสร้างเท่ากับ m x n

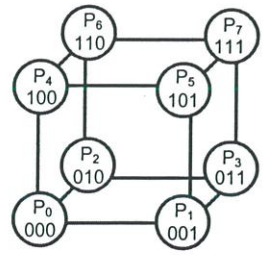
ขนาดเส้นผ่าศูนย์กลางโครงข่าย (Diameter of network) เท่ากับ m-1+n-1

จำนวนโหนดดีกรี (Node degree)

- จำนวนโหนดดีกรีเท่ากับ 2 สำหรับโหนดมุม (Corner nodes)
- จำนวนโหนดดีกรีเท่ากับ 3 สำหรับโหนดด้านข้าง (Side nodes)
- จำนวนโหนดดีกรีเท่ากับ 4 สำหรับโหนดกลาง (Middle nodes)

2.2.3 โครงสร้างการติดต่อแบบไฮเปอร์คิว (Hypercube Network)

โครงสร้างการติดต่อแบบไฮเปอร์คิวเป็นการติดต่อกันแบบ k มิติ มีจำนวน 2^k หน่วยประมวลผลโดยการติดต่อระหว่างโหนดจะขึ้นอยู่กับค่าเลขฐานสองของโหนด (Binary Representation) คือ $x_{k-1}x_{k-2}\ldots x_1x_0$ โดยหน่วยประมวลผลที่ติดต่อกันได้โดยตรงจะมีบิต (bit) ต่างกัน 1 บิตเท่านั้น เช่น หน่วยประมวลผลที่ P_{000} ติดต่อกับ P_{001} P_{010} และ P_{100} ในมิติที่ 1, 2 และ 3 ตามลำดับ ดังแสดงในรูปที่ 2.15



รูปที่ 2.15 โครงสร้างการติดต่อแบบไฮเปอร์คิว

ขนาดของโครงสร้างหรือจำนวนโหนดทั้งหมดในโครงสร้างเท่ากับ n หรือ 2^k

ขนาดเส้นผ่าศูนย์กลางโครงข่าย (Diameter of network) เท่ากับ k

จำนวนโหนดดีกรี (Node degree) สำหรับทุกโหนดในระบบเท่ากับ k

2.3 การวัดประสิทธิภาพของการประมวลผลแบบขนาน

วิธีการวัดประสิทธิภาพขั้นตอนวิธีการทางคอมพิวเตอร์แบบขนาน (Parallel algorithm) จะมีความซับซ้อนกว่าขั้นตอนวิธีการทางคอมพิวเตอร์แบบอนุกรม (Sequential algorithm) โดยสามารถประเมินผลจากเวลาที่ใช้ในการประมวลผล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) เมื่อใช้หน่วยประมวลผลเพิ่มขึ้น ในการประมวลผลจริงบนระบบคลัสเตอร์สามารถวัดเวลาที่ใช้ในการประมวลผลงานหนึ่งได้โดยจับเวลาที่ใช้ในการประมวลผลตั้งแต่เริ่มต้นจนสิ้นสุดการประมวลผล ซึ่งวัดได้ทั้งแบบอนุกรม (Sequential Programming) และแบบขนาน (Parallel Programming) โดยจะทำการวัดเวลาที่ใช้ในการประมวลผล ซึ่งไม่รวมเวลาที่ใช้ในการสร้างข้อมูลมาเก็บไว้ในหน่วยความจำสำรอง (Hard disk) และการตั้งค่าระบบ (Initialization) ของ MPI ซึ่งกระบวนการทั้งสองจะถูกทำครั้งเดียว

2.3.1 เวลาทั้งหมดที่ใช้ในการประมวลผล (Response Time)

การวัดเวลาที่ใช้ในการประมวลผลในทางทฤษฎี (Theoretical Approach) เวลาที่ใช้ในการประมวลผลแบบขนาน (Parallel Computation) จะสามารถคำนวณได้ดังสมการที่ 2.1 [10] [13]

$$T_p = \frac{T_s}{P} \tag{2.1}$$

เมื่อ T_p คือ เวลาที่ใช้ในการประมวลผลด้วย P หน่วยประมวลผล

T_s คือ เวลาที่ใช้ในการประมวลผลด้วยหนึ่งหน่วยประมวลผล

P คือ จำนวนหน่วยประมวลผล (Processor) ที่ใช้ในระบบ

ซึ่งกรณีนี้จะเรียกว่าเป็นกรณีอุดมคติ (Ideal Case) เพราะว่ามีการสมมติให้เวลาที่ใช้ในการติดต่อสื่อสาร (Communication Time) ระหว่างหน่วยประมวลผลมีค่าเป็นศูนย์ แต่ในทางปฏิบัติ (Practical Approach) บนระบบคลัสเตอร์ การวัดเวลาที่ใช้ในการประมวลผลแบบขนานจะวัดโดยการจับเวลา ตั้งแต่เริ่มประมวลผล จนกระทั่งสิ้นสุดการประมวลผล หรือได้คำตอบที่ต้องการ ซึ่ง

เวลาที่วัดได้ดังกล่าวจะรวมเวลาที่ใช้ในการติดต่อสื่อสารด้วย ดังนั้นถึงแม้ว่าจำนวนของหน่วยประมวลผล (P) ในระบบจะมีเพิ่มขึ้นมากๆ แต่เวลาที่ได้ของระบบจะไม่ลดลงจนเข้าใกล้ศูนย์ เพราะสาเหตุมาจากเวลาที่ใช้ในการประมวลผลทั้งหมดประกอบด้วยเวลาที่ใช้ในการประมวลผล (Computation Time) และเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Communication Time) ซึ่งปกติจะไม่เป็นศูนย์ ดังกรณีอุดมคติ โดยจะมีลักษณะตามสมการที่ 2.2

$$T_p = t_{computation} + t_{communication} \tag{2.2}$$

เมื่อ T_p คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วย P หน่วยประมวลผล

$t_{computation}$ คือ เวลาที่ใช้ในการประมวลผลซึ่งไม่รวม เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล

$t_{communication}$ คือ เวลาที่ใช้ในการสื่อสารระหว่างหน่วยประมวลผล

ดังนั้นในทางปฏิบัติเราสามารถเพิ่มหน่วยประมวลผลเข้าไปในระบบมากขึ้น และเวลาจะลดลงช่วงหนึ่งเท่านั้น เพราะเวลาที่ใช้ในการประมวลผล (Computation Time) มากกว่าเวลาที่ใช้ในการติดต่อสื่อสาร (Communication Time) ระหว่างหน่วยประมวลผล แต่ ณ จุดหนึ่งเมื่อเพิ่มหน่วยประมวลผลเข้าไปอีก เวลาจะไม่ลดลงแต่อาจจะเพิ่มขึ้นเนื่องจากเวลาที่ใช้ในการสื่อสารระหว่างหน่วยประมวลผล $t_{communication}$ มากขึ้นและมีค่ามากกว่า เวลาที่ใช้ในการประมวลผล

2.3.2 อัตราการเพิ่มของความเร็ว (Speedup)

การวัดอัตราการเพิ่มของความเร็ว (Speedup) แสดงได้ดังสมการที่ 2.3 [10] [13]

$$S_p = \frac{T_s}{T_p} \leq p \tag{2.3}$$

เมื่อ S_p คือ อัตราการเพิ่มของความเร็ว โดยใช้ P หน่วยประมวลผล ซึ่งมีค่าสูงสุดคือ P

T_s คือ เวลาที่ใช้ในการประมวลผลโดยใช้หนึ่งหน่วยประมวลผล

T_p คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วย P หน่วยประมวลผล

2.3.3 ประสิทธิภาพ (Efficiency)

การวัดประสิทธิภาพ แสดงดังสมการที่ 2.4 [10] [13]

$$E_p = \frac{S_p}{P} \leq 1 \tag{2.4}$$

เมื่อ E_p คือ ประสิทธิภาพของระบบมีค่าสูงสุดคือ 1

S_p คือ อัตราการเพิ่มของความเร็วที่คำนวณได้จากสมการที่ 2.3

P คือ จำนวนหน่วยประมวลผลที่ใช้บนระบบ

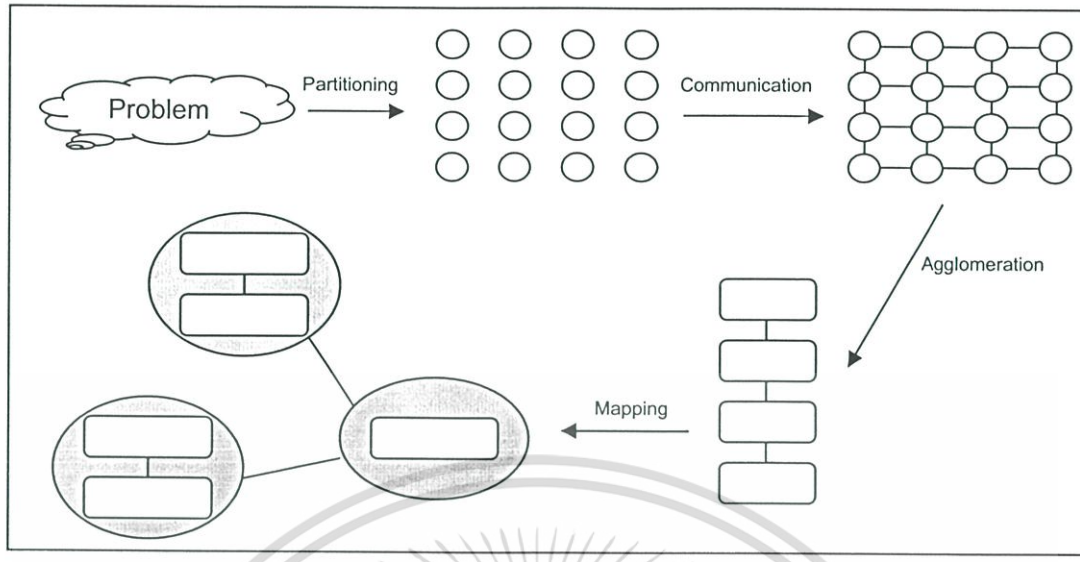
2.4 การออกแบบโปรแกรมและพัฒนาโปรแกรมแบบขนาน

2.4.1 การออกแบบโปรแกรมแบบขนาน

ปกติการเขียนโปรแกรมที่ดีนั้นจะต้องมีการออกแบบโปรแกรมที่ดีก่อน โดยโปรแกรมที่ดีนั้นจะต้องเป็นไปตามหลักการและหลีกเลี่ยงปัญหาต่างๆ ซึ่งวิธีการออกแบบโปรแกรมแบบขนานนี้ Foster [14] ได้นำเสนอหลักการต่างๆ ไว้ 4 ขั้นตอนด้วยกันคือ

1) การแบ่งงาน (Partition)
2) การติดต่อสื่อสาร (Communication)
3) การรวมกลุ่มงาน (Agglomerate)
4) การกำหนดงานไปยังหน่วยคำนวณที่เหมาะสม (Mapping)

ปัญหาขนาดใหญ่ที่ต้องคำนวณนั้น ในขั้นตอนของการแบ่งงานจะทำการแยกแยะปัญหาออกเป็นส่วนย่อยๆ ตามความเหมาะสมของแต่ละปัญหา หลังจากนั้นจึงทำการพิจารณาความเกี่ยวกันของแต่ละส่วน โดยอาจจะพิจารณาจากขอบเขต การอยู่ติดกันของส่วนย่อยเล็กๆ เหล่านั้น รวมทั้งพิจารณาถึงการรับส่งค่าของปัญหาต่างๆ อีกด้วย หลังจากนั้นจึงทำการรวมส่วนที่เกี่ยวข้องหรือมีความสัมพันธ์เข้าไว้ด้วยกันในขั้นตอนของการรวมกลุ่มงาน และสุดท้ายจะเป็นการกำหนดงานไปยังหน่วยคำนวณ (Processing Element) ที่เหมาะสมต่อไปรายละเอียดแสดงในรูปที่ 2.16 ซึ่งสามารถอธิบายขั้นตอนต่างๆ ได้ดังนี้



รูปที่ 2.16 ขั้นตอนการออกแบบโปรแกรมแบบขนาน

2.4.1.1 การแบ่งงาน (Partition)

การแบ่งงานเป็นจุดเริ่มต้นของการประมวลผลแบบขนาน เพราะจะต้องทำการแบ่งปัญหาขนาดใหญ่ที่ซับซ้อนออกเป็นส่วนย่อยที่สามารถคำนวณไปพร้อมๆ กัน หลังจากนั้นจึงทำขั้นตอนอื่นๆ ต่อไป โดยการแบ่งงานสามารถแบ่งได้เป็น 2 แบบ คือ

1) การแบ่งข้อมูลของปัญหาออกเป็นส่วนย่อย (Domain Decomposition or Data Decomposition) ในส่วนนี้จะเป็นการพิจารณาที่ข้อมูลเข้าว่าสามารถแบ่งเป็นส่วนย่อยหรือเป็นกลุ่มของข้อมูลที่อยู่ติดกัน แล้วทำการประมวลผลโดยใช้หลักการเอสพีเอ็มดี (SPMD: Single Program Multiple Data) ซึ่งโปรแกรมที่ทำงานอยู่บนแต่ละหน่วยประมวลผลจะเป็นโปรแกรมชุดเดียวกัน แต่ประมวลผลข้อมูลคนละชุดกัน เช่น การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค การบีบอัดข้อมูลภาพเคลื่อนไหวหรือภาพยนตร์ เป็นต้น

2) การแบ่งงานตามหน้าที่ของการคำนวณออกเป็นส่วนย่อย (Function Decomposition) ซึ่งการแบ่งงานแบบนี้จะกระทำตั้งแต่เริ่มลงมือเขียนส่วนของโปรแกรม เพราะจะต้องทำการแยกฟังก์ชั่นการทำงานของแต่ละส่วนอย่างชัดเจนตั้งแต่แรก เพื่อให้เป็นไปตามหลักการเอ็มพีเอ็มดี (MPMD: Multiple Program Multiple Data) เช่น การคูณเมตริกซ์ การสร้างแบบจำลองของสภาพภูมิอากาศ เป็นต้น

2.4.1.2 การติดต่อสื่อสาร (Communication)

การติดต่อสื่อสารนี้จะพิจารณาความสัมพันธ์ในเรื่องของการสื่อสารระหว่างงานที่แบ่งออกไปในงานบางอย่าง งานย่อยที่ถูกแบ่งไม่สามารถประมวลผลได้ในเวลาเดียวกัน เนื่องจากต้องอาศัยข้อมูลหรือผลลัพธ์จากงานย่อยอื่นเพื่อใช้ในการประมวลผล ดังนั้นจึงต้องมีการสื่อสารระหว่างงานย่อย การเลือกวิธีการสื่อสารที่เหมาะสมจะทำให้การประมวลผลมีประสิทธิภาพมากยิ่งขึ้น การออกแบบขั้นตอนการติดต่อสื่อสารจะแบ่งเป็นสองขั้นตอนคือ ขั้นแรกออกแบบโครงสร้างของการเชื่อมโยงว่างานใดต้องการข้อมูลและงานใดทำหน้าให้ข้อมูล ขั้นที่สองกำหนดโครงสร้างของข้อความ (Message) ที่ใช้ในการส่งและรับของแต่ละการเชื่อมโยง

2.4.1.3 การรวมกลุ่มงาน (Agglomeration)

การรวมกลุ่มงานจะถูกทำเพื่อลดเวลาที่ใช้ในการสื่อสารระหว่างงานให้ลดลงทำให้หน่วยประมวลผลไม่ต้องเสียเวลาในการติดต่อสื่อสาร และรอคอยการรับส่งข้อมูลทำให้ประสิทธิภาพในการคำนวณมากขึ้นด้วย

2.4.1.4 การกำหนดงานไปยังหน่วยประมวลผลที่เหมาะสม (Mapping)

การกำหนดงานไปยังหน่วยประมวลผลที่เหมาะสม เพื่อเป็นการใช้ประโยชน์ของหน่วยประมวลผลสูงสุด และลดการสื่อสารให้มีน้อยที่สุด การกำหนดงานนี้สามารถทำเป็นแบบคงที่ซึ่งได้กำหนดไว้ตายตัวตั้งแต่เริ่มสร้างโปรแกรม หรือทำการกำหนดตอนที่กำลังคำนวณโดยใช้เทคนิคของการสมดุลงาน (Load Balancing) เข้ามาช่วยก็ได้

2.4.2 การพัฒนาโปรแกรมแบบขนาน

การพัฒนาโปรแกรมแบบขนานแบบต่างๆ โดยลำดับแรกในการพัฒนาโปรแกรมแบบขนานนั้น ส่วนใหญ่ผู้เขียนโปรแกรมมักจะสร้างโปรแกรมแบบอนุกรม (Sequential Program) ให้สามารถทำงานได้อย่างถูกต้องเสียก่อน จากนั้นจึงจะพัฒนาโปรแกรมแบบอนุกรมนี้ไปเป็นโปรแกรมแบบขนานได้ตามวิธีต่างๆ [20] ดังนี้

2.4.2.1 การสร้างโปรแกรมแบบขนานโดยอัตโนมัติ

การสร้างโปรแกรมแบบขนานด้วยวิธีนี้เป็นวิธีที่ง่ายที่สุด แต่ก็จะมีประสิทธิภาพต่ำที่สุด โดยหน้าที่ของการสร้างโปรแกรมแบบขนานจะเป็นหน้าที่ของตัวแปรภาษาซึ่งตัวแปรภาษาจะตรวจสอบรหัสต้นฉบับ (Source Code) ซึ่งอาจประกอบด้วยส่วนของรหัสที่มีการวนซ้ำ และการประมวลผลกับข้อมูลที่ซ้ำๆ กันโดยตัวแปรภาษานี้จะทำการเปลี่ยนรหัสต้นฉบับไปเป็นรหัสที่สนับสนุนการประมวลผลแบบขนานจากนั้นใช้ตัวแปรภาษาเปลี่ยนรหัสที่ได้เป็นโปรแกรมแบบขนานเองโดยอัตโนมัติ แต่ข้อจำกัดของการสร้างโปรแกรมด้วยวิธีนี้จะขึ้นอยู่กับเทคโนโลยีที่ตัวแปรภาษาใช้ในการเปลี่ยนรหัสต้นฉบับเป็นรหัสที่สนุบสนุนการประมวลผลแบบขนาน ขั้นตอนการทำงานของการสร้างโปรแกรมแบบขนานโดยอัตโนมัติ สามารถแสดงได้ดังรูปที่ 2.17

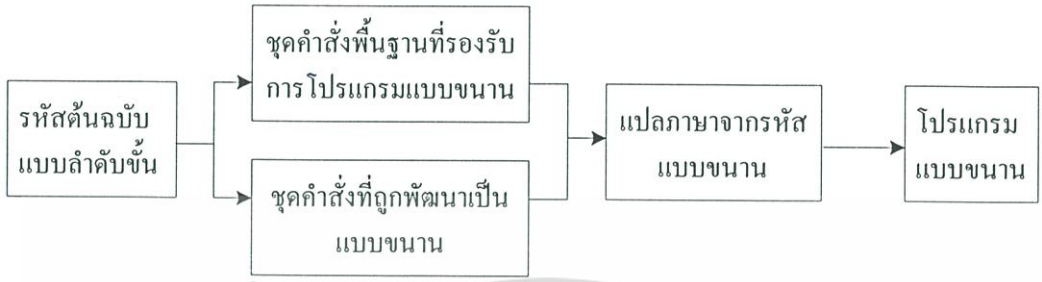
รูปที่ 2.17 การสร้างโปรแกรมแบบขนานโดยอัตโนมัติ

2.4.2.2 การสร้างโปรแกรมโดยใช้ชุดคำสั่งแบบขนาน

การสร้างโปรแกรมโดยใช้ชุดคำสั่งแบบขนานที่มีประสิทธิภาพมากกว่าวิธีแรกโดยเมื่อทำการสร้างโปรแกรมจะใช้ชุดคำสั่งที่ถูกพัฒนาขึ้นมาให้สนับสนุนการทำงานแบบขนานแล้ว เช่น ชุดคำสั่งของ ScaLAPACK หรือ PLAPACK ซึ่งชุดคำสั่งเหล่านั้นสนับสนุนการประมวลผลแบบขนานอยู่แล้ว ฟังก์ชั่นที่ชุดคำสั่งเหล่านี้เตรียมไว้ให้ใช้งาน เช่น ชุดคำสั่งของการหาผลเฉลยสมการต่างๆ แบบขนาน และการคูณเมตริกซ์แบบขนาน เป็นต้น โดยชุดคำสั่งที่สนับสนุนการทำงานแบบขนานนี้จะมีอยู่สองกลุ่ม คือ

1) กลุ่มของชุดคำสั่งพื้นฐานที่รองรับหรือทำให้เกิดสภาพแวดล้อมการประมวลผลแบบขนานตัวอย่างเช่น ฟังก์ชันที่ถูกเตรียมไว้โดย MPICH หรือ LAM ซึ่งจะเป็นคำสั่งหรือฟังก์ชันพื้นฐานที่ทำให้เกิดการประมวลผลแบบขนานได้ เพราะมีคำสั่งที่ทำให้เกิดการสื่อสารระหว่างหน่วยประมวลผลต่อหน่วยประมวลผล การส่งข้อความถึงหน่วยประมวลผลอื่นๆ ในระบบได้ เป็นต้น

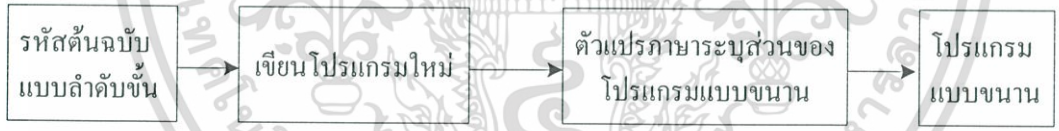
2) กลุ่มของชุดคำสั่งที่ได้รับการพัฒนาให้เป็นการประมวลผลแบบขนาน เช่น ScaLAPACK หรือ PLAPACK ซึ่งเป็นชุดคำสั่งทางคณิตศาสตร์ที่ได้รับการพัฒนาให้ทำงานแบบขนานได้โดยเรียกใช้ชุดคำสั่งของมาตรฐานภาษาเอ็มพีไอ (MPI Standard) อีกที



รูปที่ 2.18 การสร้างโปรแกรมโดยใช้ชุดคำสั่งแบบขนาน

2.4.2.3 การสร้างโปรแกรมแบบขนานด้วยตนเอง

การสร้างโปรแกรมแบบขนานด้วยวิธีนี้เป็นแบบยืดหยุ่นมากที่สุด ผู้สร้างโปรแกรมสามารถเลือกตัวแปรภาษาใดก็ได้ตามที่ต้องการ รวมถึงสามารถเลือกรูปแบบและวิธีการที่ใช้ในการสื่อสาร แต่วิธีนี้นับเป็นวิธีที่ยากและเสียเวลามากที่สุด ขั้นตอนการสร้างโปรแกรมแบบขนานด้วยตัวเองสามารถแสดงได้ดังรูปที่ 2.19



รูปที่ 2.19 การสร้างโปรแกรมแบบขนานด้วยตนเอง

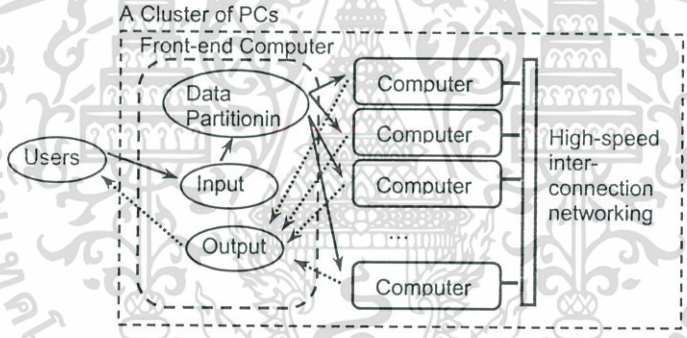
2.4.3 มาตรฐานภาษาเอ็มพีไอ (Message-Passing Interface)

ในปี ค.ศ. 1990 ได้มีการกำหนดมาตรฐานในการส่งผ่านข้อมูล (Message-Passing) [7] ขึ้นใหม่นั้นคือ เอ็มพีไอ (MPI) เนื่องจากมาตรฐานที่มีอยู่ก่อนหน้านี้คือ พีวีเอ็ม (PVM) ซึ่งเป็นมาตรฐานที่ได้มีการใช้งานอย่างแพร่หลายใน ค.ศ. 1980 แต่ด้วยจุดอ่อนของพีวีเอ็มในหลายๆ ด้าน ทำให้การพัฒนาโปรแกรมแบบขนานมีความยุ่งยากซับซ้อนมากขึ้น ด้วยเหตุนี้จึงทำให้เอ็มพีไอได้รับความนิยมในเวลาต่อมา

การพัฒนาโปรแกรมแบบขนานบนระบบคลัสเตอร์ซึ่งจะมีไลบรารี (Library) และฟังก์ชันพื้นฐานต่างๆ เป็นตัวกลางในการเชื่อมการทำงานของแต่ละหน่วยประมวลผลเข้าด้วยกันเพื่อให้หน่วยประมวลผลสามารถทำงานรวมกันได้อย่างสะดวกและมีประสิทธิภาพ ซึ่งจะทำให้นักพัฒนาโปรแกรมสามารถพัฒนาโปรแกรมได้ง่ายและสะดวกรวดเร็วมากขึ้น ด้วยภาษาต่างๆ ตามความถนัดของนักพัฒนา เช่น ภาษาฟอร์แทน (Fortan Language) ภาษาซี (C Language) เป็นต้น

เนื่องจากมาตรฐานภาษาเอ็มพีไอได้รับความนิยมจากนักพัฒนาโปรแกรมแบบขนาน ทำให้มีซอฟท์แวร์จำนวนมากที่สนับสนุนมาตรฐานภาษาเอ็มพีไอทั้งในเชิงพาณิชย์ (Commercial) เช่น เอ็มพีไอโปร (MPI/Pro) หรือแบบให้เปล่า (Open Source) เช่น เอ็มพีไอซีเอช (MPICH) เป็นต้น

งานวิจัยนี้ได้แบ่งการทำงานเป็น 2 ส่วน คือ การทำงานในส่วนหน่วยประมวลผลหลัก (Master computer or Front-end Computer) และส่วนของหน่วยประมวลผลทำงาน (Worker computer) ซึ่งติดต่อกันด้วยเครือข่ายความเร็วสูง รายละเอียดดังแสดงในรูปที่ 2.20



รูปที่ 2.20 ระบบคอมพิวเตอร์คลัสเตอร์ [16]

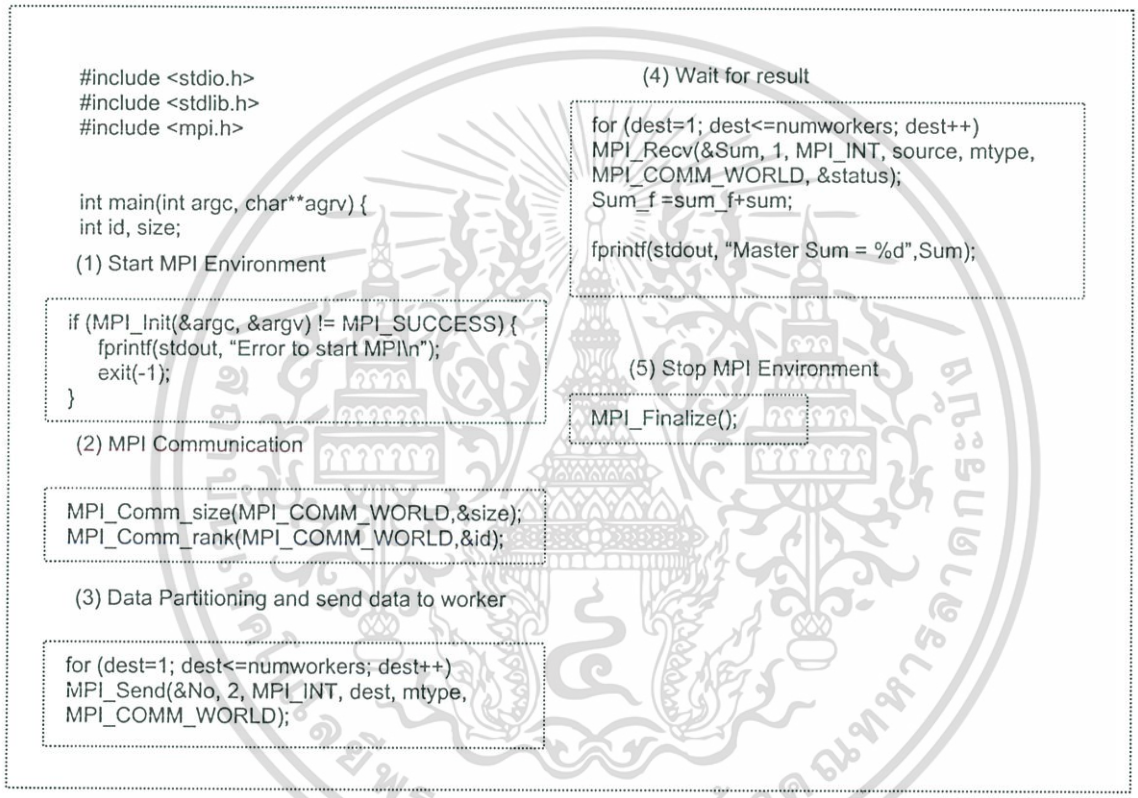
ขั้นตอนการพัฒนาโปรแกรมแบบขนานบนระบบคลัสเตอร์ โดยใช้เอ็มพีไอมีขั้นตอนการทำงานดังนี้

1) เริ่มต้นการทำงานของการเขียนโปรแกรมแบบขนาน
2) ผู้ใช้ (Users) ติดต่อกับหน่วยประมวลผลหลักและสั่งให้หน่วยประมวลผลหลักทำงาน
3) หน่วยประมวลผลหลักแบ่งการทำงาน (Data Partitioning) และส่งข้อมูลให้หน่วยประมวลผลทำงาน (Worker computer)
4) หน่วยประมวลผลทำงานทำการหาผลลัพธ์ซึ่งหน่วยประมวลผลหลักรออยู่
5) หน่วยประมวลผลทำงานส่งผลลัพธ์ให้กับหน่วยประมวลผลหลัก
6) หน่วยประมวลผลหลักทำการรวบรวมข้อมูลจากแต่ละหน่วยประมวลผล

7) หยุดการทำงานของการประมวลผลแบบขนานของแต่ละหน่วยประมวลผล

8) หน่วยประมวลผลหลักแสดงผลลัพธ์ให้ผู้ใช้

ตัวอย่างโปรแกรมการหาผลรวมของกลุ่มตัวเลข {1, 2, 3, 4, 5, 6, 7, 8} ที่เขียนตามมาตรฐานภาษาเอ็มพีไอ (MPI Standard) ซึ่งแสดงขั้นตอนการพัฒนาโปรแกรมแบบขนานที่ได้กล่าวในข้างต้น และแสดงผลจากการประมวลผลที่หน่วยประมวลผลหลัก โดยใช้หน่วยประมวลผลในการรัน (Run) โปรแกรมตัวอย่างนี้จำนวน 5 หน่วยประมวลผล



รูปที่ 2.21 ตัวอย่างโปรแกรมการหาผลรวมของกลุ่มตัวเลขส่วนหน่วยประมวลผลหลัก (Master computer)

ฟังก์ชัน MPI ที่ใช้ในโปรแกรมตัวอย่าง

- MPI_Send (void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

เมื่อ *buf คือ ตำแหน่งเริ่มต้นที่จะส่ง (buf initial address of send buffer (choice))

count คือ จำนวนที่จะส่ง (count number of elements in send buffer)

datatype คือ ชนิดของข้อมูล (datatype of each send buffer element (handle))

dest คือ เครื่องเป้าหมายที่จะส่ง (dest rank of destination (integer))

tag คือ ค่าในการส่งข้อความ (tag message tag (integer))

comm คือ การติดต่อ (comm communicator (handle))

- MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

เมื่อ *buf คือ ตำแหน่งเริ่มต้นที่จะรับ (initial address of receive buffer (choice))

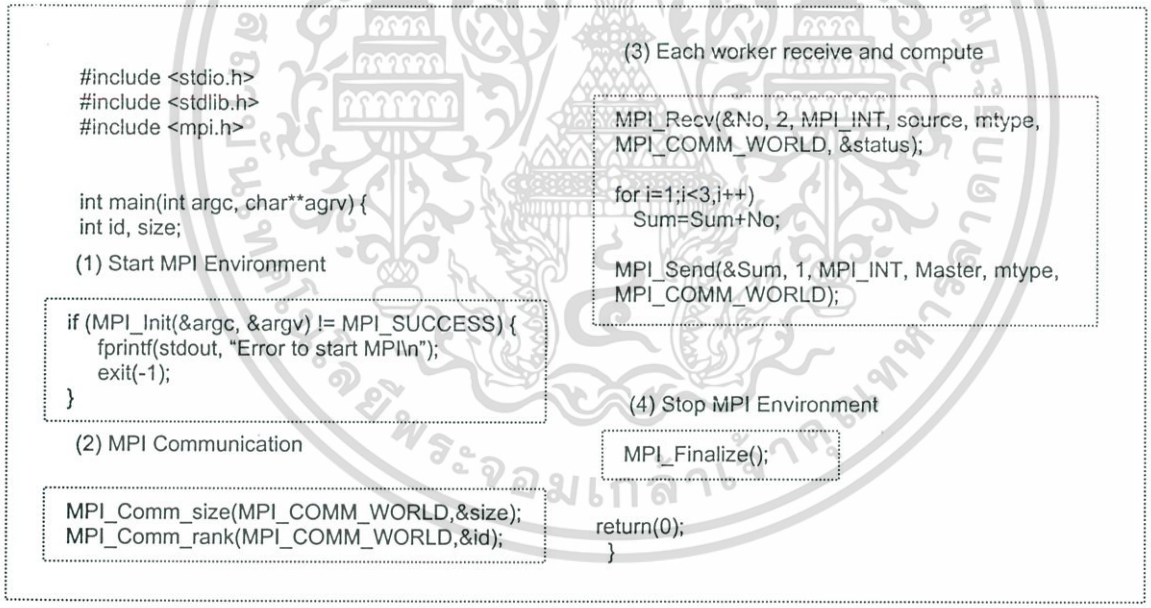
count คือ ค่าจำนวนที่จะรับ (count maximum number of elements in receive buffer (integer))

datatype คือ ชนิดของข้อมูล (datatype of each send buffer element (handle))

source คือ เครื่องที่ส่งมา (source rank of source (integer))

tag คือ ค่าในการส่งข้อความ (tag message tag (integer))

comm คือ การติดต่อ (comm communicator (handle))



รูปที่ 2.22 ตัวอย่างโปรแกรมการหาผลรวมของกลุ่มตัวเลขส่วนหน่วยประมวลผลทำงาน (Worker computer)

ผลจากการประมวลผลโปรแกรมแสดงหมายเลขประจำหน่วยประมวลผล คือ

```
Master            Sum =  36
Worker 1 Partial  Sum =   3
Worker 2 Partial  Sum =   7
Worker 3 Partial  Sum =  11
Worker 4 Partial  Sum =  15
```

2.5 การแบ่งชุดข้อมูล (Data Partitioning)

การแบ่งชุดข้อมูลของข้อมูลแบบเมตริกซ์หรือตาราง 2 มิติ มี 2 วิธี คือ การแบ่งชุดข้อมูลแบบเป็นส่วนๆ (Strip Partitioning) และการแบ่งชุดข้อมูลแบบตาราง (Checkerboard Partitioning)

2.5.1 การแบ่งชุดข้อมูลแบบเป็นส่วนๆ (Strip Partitioning)

วิธีการแบ่งชุดข้อมูลทำโดยแบ่งข้อมูลเป็นส่วนๆ ตามแถว (Row) หรือตามหลัก (Column) ติดกัน (Block) หรือเป็นวงรอบ (Cyclic) สามารถแบ่งย่อยเป็น 4 ประเภท พร้อมตัวอย่างเมตริกซ์ขนาด 16 x16 (16 แถว 16 หลัก) ที่ถูกแบ่งสำหรับ 4 หน่วยประมวลผล (P=4) ดังนี้ [10]

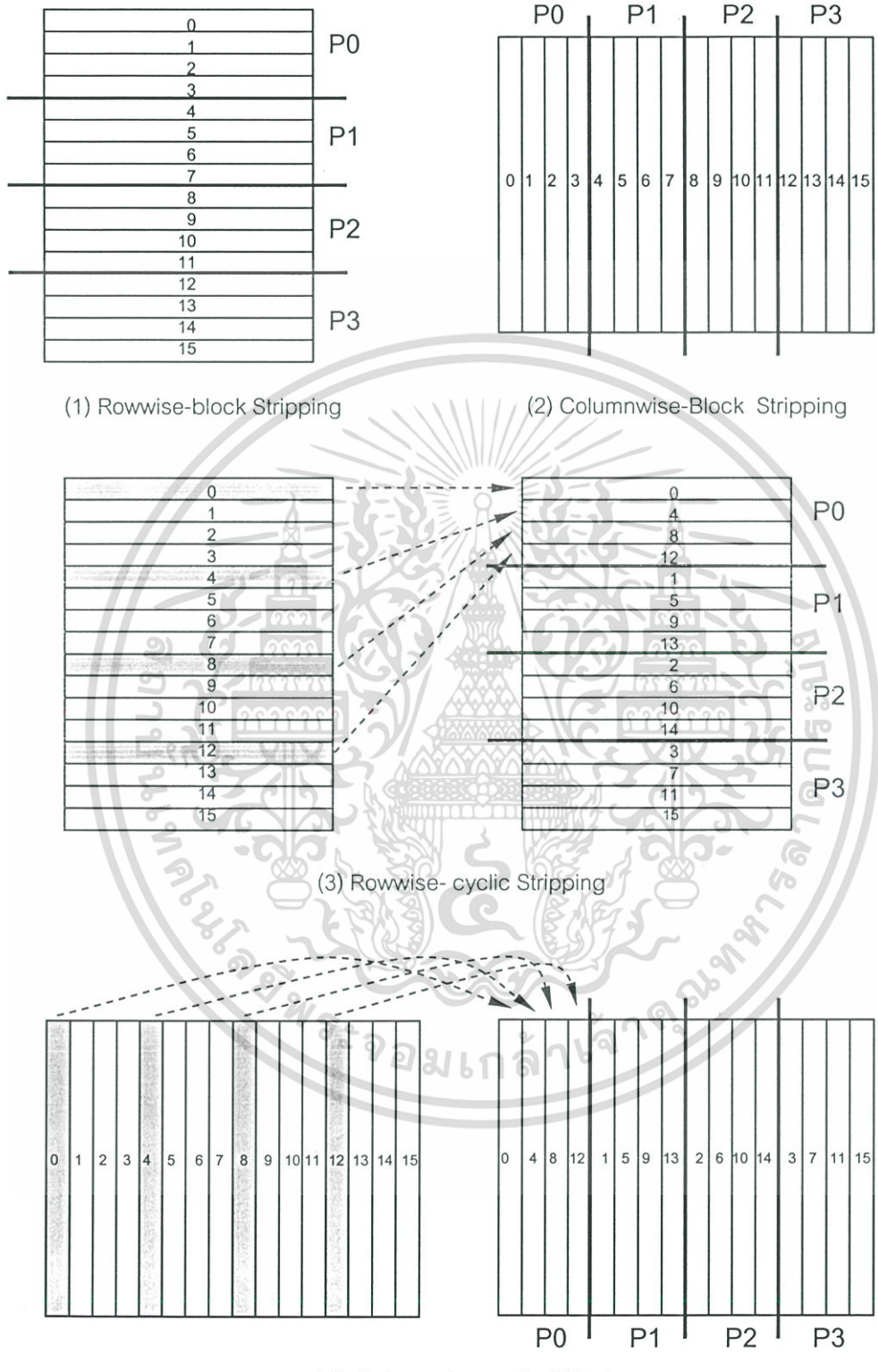
1) วิธีการแบ่งชุดข้อมูลแบบ Rowwise-Block Stripping คือ การแบ่งชุดข้อมูลในแต่ละแถวของชุดข้อมูลที่ติดกัน ถูกแบ่งให้แต่ละหน่วยประมวลผลมีขนาดเท่าๆ กัน รายละเอียดแสดงดังรูปที่ 2.23 (1) โดยที่หน่วยประมวลผล Pi ได้รับข้อมูลแถวติดกันจากแถวที่ 4 i ถึง 4(i+1)-1 เมื่อ i = 0, 1, 2, 3 ตามลำดับ

2) วิธีการแบ่งชุดข้อมูลแบบ Columnwise-Block Stripping คือ การแบ่งชุดข้อมูลในแต่ละหลักของชุดข้อมูลที่ติดกัน ถูกแบ่งให้แต่ละหน่วยประมวลผลมีขนาดเท่าๆ กัน รายละเอียดแสดงดังรูปที่ 2.23 (2) โดยที่หน่วยประมวลผล Pj ได้รับข้อมูลหลักติดกันจากหลักที่ 4j ถึง 4(j+1)-1 เมื่อ j = 0, 1, 2, 3 ตามลำดับ

3) วิธีการแบ่งชุดข้อมูลแบบ Rowwise-Cyclic Stripping คือ การแบ่งชุดข้อมูลในแต่ละแถวของชุดข้อมูลที่ไม่ติดกัน แบ่งเป็นวงรอบ (Cyclic) โดยเริ่มจากแบ่งแถวข้อมูล 1 แถวให้แต่ละหน่วยประมวลผลตามลำดับจนครบรอบ แล้วทำการเพิ่มแถวขึ้นอีกตามลำดับ เช่น หน่วยประมวลผลที่ 1 มีแถวที่ 0, 4, 8, 12 เป็นต้น โดยแต่ละหน่วยประมวลผลจะมีจำนวนแถวเท่าๆ กัน รายละเอียดแสดงดังรูปที่ 2.23 (3) โดยที่หน่วยประมวลผล Pi ได้รับข้อมูลแถวที่ไม่ติดกัน คือแถวที่ i, i+4, i+8 และ i+12 เมื่อ i = 0, 1, 2, 3 ตามลำดับ

4) วิธีการแบ่งชุดข้อมูลแบบ Columnwise-Cyclic Stripping คือ การแบ่งชุดข้อมูลในแต่ละหลักของชุดข้อมูลที่ไม่ติดกัน แบ่งเป็นวงรอบ (Cyclic) โดยเริ่มจากแบ่งหลักข้อมูล 1 หลักให้แต่ละหน่วยประมวลผลตามลำดับจนครบรอบ แล้วทำการเพิ่มหลักขึ้นอีกตามลำดับ เช่น หน่วยประมวลผลที่ 1 มีหลักที่ 0, 4, 8, 12 เป็นต้น โดยแต่ละหน่วยประมวลผลจะมีจำนวนหลักเท่าๆ กัน

รายละเอียดแสดงดังรูปที่ 2.23 (4) โดยที่หน่วยประมวลผล Pj ได้รับข้อมูลหลักที่ไม่ติดกัน คือ หลักที่ j, j+4, j+8 และ j+12 เมื่อ j = 0, 1, 2, 3 ตามลำดับ



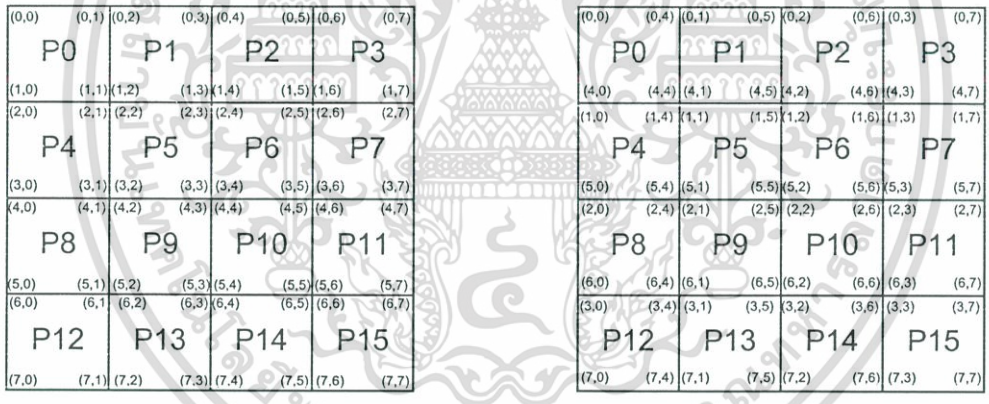
รูปที่ 2.23 การแบ่งชุดข้อมูลแบบเป็นส่วนๆ

2.5.2 การแบ่งชุดข้อมูลแบบตาราง (Checkerboard Partitioning)

วิธีการแบ่งชุดข้อมูลทำโดยแบ่งข้อมูลติดกัน (Block) หรือเป็นวงรอบ (Cyclic) สามารถแบ่งย่อยเป็น 2 ประเภท พร้อมตัวอย่างเมตริกซ์ขนาด 8x8 (8 แถว 8 หลัก) ที่ถูกแบ่งสำหรับ 16 หน่วยประมวลผล (P=16) รายละเอียดดังนี้ [10]

1) วิธีการแบ่งชุดข้อมูลแบบ Block-Checkerboard Partitioning คือ การแบ่งชุดข้อมูลเป็นเมตริกซ์ย่อยในแต่ละหลักและแต่ละแถวของชุดข้อมูลที่ติดกันโดยแต่ละหน่วยประมวลผลจะมีขนาดเมตริกซ์ย่อยเท่าๆ กัน รายละเอียดแสดงดังรูปที่ 2.24 (1)

2) วิธีการแบ่งชุดข้อมูลแบบ Cyclic-Checkerboard Partitioning คือ การแบ่งชุดข้อมูลเป็นเมตริกซ์ย่อยในหลักและตามด้วยแถวของชุดข้อมูล ซึ่งข้อมูลจะไม่ติดกันโดยแต่ละหน่วยประมวลผลจะมีขนาดเมตริกซ์ย่อยเท่าๆ กัน รายละเอียดแสดงดังรูปที่ 2.24 (2)



รูปที่ 2.24 การแบ่งชุดข้อมูลแบบตาราง

2.6 การหาผลคูณของเมตริกซ์แบบขนาน (Parallel Matrix Multiplication)

การหาผลคูณของเมตริกซ์เป็นการประมวลผลขั้นพื้นฐานในการหาคำตอบทางวิทยาศาสตร์และวิศวกรรมศาสตร์หลาย ๆ ด้าน เช่น การหาคำตอบของสมการโดยใช้เมตริกซ์ การวิเคราะห์โครงสร้างทางวิศวกรรม การประยุกต์การหาระยะทางที่สั้นที่สุดโดยใช้เมตริกซ์ งานด้านกราฟิค เป็นต้น

วิธีการหาผลคูณของเมตริกซ์ คือ เมตริกซ์ A และ เมตริกซ์ B จะสามารถคูณกันได้จะต้องมีจำนวนหลักของเมตริกซ์ A ต้องเท่ากับจำนวนแถวของ B (Compatible) โดยมีวิธีการดังนี้

กำหนดให้เมตริกซ์ C_{nxn} เป็นผลลัพธ์ที่เกิดจากการคูณของเมตริกซ์ A_{nxm} และ เมตริกซ์ B_{mxn} โดยมีสูตรในการคูณกันระหว่างเมตริกซ์ดังนี้

$$A_{nxm} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2m} \\ \vdots & & & & & \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{im} \\ \vdots & & & & & \\ a_{n1} & a_{n2} & \cdots & a_{nj} & \cdots & a_{nm} \end{pmatrix}_{nxm} \quad \text{และ } B_{mxn} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1j} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2j} & \cdots & b_{2n} \\ \vdots & & & & & \\ b_{i1} & b_{i2} & \cdots & b_{ij} & \cdots & b_{in} \\ \vdots & & & & & \\ b_{m1} & b_{m2} & \cdots & b_{mj} & \cdots & b_{mn} \end{pmatrix}_{mxn}$$

$$C_{nxn} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1j} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2j} & \cdots & c_{2n} \\ \vdots & & & & & \\ c_{i1} & c_{i2} & \cdots & c_{ij} & \cdots & c_{in} \\ \vdots & & & & & \\ c_{n1} & c_{n2} & \cdots & c_{nj} & \cdots & c_{nn} \end{pmatrix}_{nxn} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2m} \\ \vdots & & & & & \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{im} \\ \vdots & & & & & \\ a_{n1} & a_{n2} & \cdots & a_{nj} & \cdots & a_{nm} \end{pmatrix}_{nxm} \times \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1j} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2j} & \cdots & b_{2n} \\ \vdots & & & & & \\ b_{i1} & b_{i2} & \cdots & b_{ij} & \cdots & b_{in} \\ \vdots & & & & & \\ b_{m1} & b_{m2} & \cdots & b_{mj} & \cdots & b_{mn} \end{pmatrix}_{mxn}$$

รูปที่ 2.25 การหาผลคูณของเมตริกซ์ $C_{nxn} = A_{nxm} \times B_{mxn}$

ดังนั้นสำหรับค่า i และ j เมื่อ i=1,2,…,n และ j=1,2,…,n จะได้

$$c_{i,j} = \sum_{k=1}^{m} a_{i,k} b_{k,j} \qquad (2.5)$$

เมื่อ $c_{i,j}$ คือ ค่าในเมตริกซ์ C ซึ่งเป็นผลคูณของเมตริกซ์ A แถวที่ i และเมตริกซ์ B หลักที่ j

$a_{i,k}$ คือ ค่าในเมตริกซ์ A แถวที่ i หลักที่ k

$b_{k,j}$ คือ ค่าในเมตริกซ์ B แถวที่ k หลักที่ j

การหาผลคูณของเมตริกซ์แบบอนุกรม มีค่าความซับซ้อนด้านเวลา (Time Complexity) เท่ากับ $O(n^3)$ ซึ่งต้องใช้ระยะเวลาเป็นจำนวนมากเมื่อเมตริกซ์มีขนาดใหญ่ๆ (n มีค่าเพิ่มมากขึ้น) ดังนั้นการนำระบบคอมพิวเตอร์แบบขนานมาประยุกต์ใช้กับการหาผลคูณของเมตริกซ์เพื่อลดระยะเวลาดังกล่าว โดยเริ่มจากการแบ่งงานให้คอมพิวเตอร์หลายๆ เครื่องในระบบเพื่อทำการคูณเมตริกซ์ย่อยๆ ไปพร้อมๆ กัน แล้วนำผลลัพธ์ที่ได้ในแต่ละส่วนมารวมกันเป็นคำตอบ โดยขั้นตอนวิธีการทางคอมพิวเตอร์แบบขนาน (Parallel algorithm) สำหรับการหาผลคูณของเมตริกซ์มีผู้เสนอไว้หลายวิธีที่ต่างกันตามขนาดของระบบคอมพิวเตอร์แบบขนานและโครงสร้างการติดต่อสื่อสารของระบบ เช่น

1) กรณีที่มีจำนวนหน่วยประมวลผลมากกว่าจำนวนข้อมูลของเมตริกซ์ (P>N) เช่น จำนวนหน่วยประมวลผล (P) เท่ากับ nxnxn และขนาดเมตริกซ์ (N) เท่ากับ nxn

2) กรณีที่มีจำนวนหน่วยประมวลผลน้อยกว่าจำนวนข้อมูลของเมตริกซ์ (P<N) เช่น จำนวนหน่วยประมวลผล (P) เท่ากับ n และขนาดเมตริกซ์ (N) เท่ากับ nxn

3) กรณีที่มีจำนวนหน่วยประมวลผลเท่ากับจำนวนข้อมูลของเมตริกซ์ (P=N) เช่น จำนวนหน่วยประมวลผล (P) เท่ากับ nxn และขนาดเมตริกซ์ (N) เท่ากับ nxn

การหาผลคูณของเมตริกซ์แบบขนานแยกตามขนาดและโครงสร้างการติดต่อสื่อสารระหว่างหน่วยประมวลผลมีรายละเอียดดังนี้

2.6.1 การหาผลคูณของเมตริกซ์แบบขนานบนการติดต่อแบบพีแรม (n x n x n (cube))

การหาผลคูณของเมตริกซ์แบบขนานบนการติดต่อระหว่างหน่วยประมวลผลเป็นแบบพีแรมซีอาร์อีดับเบิลยู (PRAM-CREW-nxnxn(cube)) ซึ่งอนุญาตให้ทุกหน่วยประมวลผลสามารถอ่านข้อมูลในหน่วยความจำ ณ จุดๆ เดียวกันในหนึ่งรอบเวลา โดยใช้ n^3 หน่วยประมวลผล (ดังแสดงในรูปที่ 2.26 (1)) เมตริกซ์ C เป็นผลคูณระหว่าง เมตริกซ์ A และเมตริกซ์ B โครงสร้างของหน่วยความจำร่วมเป็น 3-D อะเรย์ ซึ่งมีเมตริกซ์ A และ เมตริกซ์ B เก็บอยู่ในแนวแกน x และ y (ดังแสดงในรูปที่ 2.26(2)) ในการคำนวณการหาผลคูณของเมตริกซ์แบบขนาน

ขั้นตอนที่ 1 ทุกหน่วยประมวลผล P_{ijk} เมื่อ i,j,k = 1,2,...,n อ่านค่า a_{ik} และ b_{kj} และทำการคำนวณค่า $c_{ijk} = a_{ik} \times b_{kj}$ หลังจากนั้นจะเก็บค่า c_{ijk} ในหน่วยความจำร่วม

ขั้นตอนที่ 2 ทุกหน่วยประมวลผล $P_{ijk/2}$ เมื่อ i,j,k = 1,2,...,n คำนวณหาผลบวก c_{ij} แบบบวกทีละคู่ จำนวน log n รอบ เพื่อเก็บไว้ในหน่วยความจำในแนวแกน x และ y

ตัวอย่างเช่น c_{11} (รูปที่ 2.26(3)) จะคำนวณจาก

รอบ 1 (P_{111}) $c_{111} \leftarrow c_{111} + c_{11n/2+1}$, (P_{112}) $c_{112} \leftarrow c_{112} + c_{11n/2+2}$,

$\ldots$, $(P_{11n/2})$ $c_{11n/2} \leftarrow c_{11n/2} + c_{11n}$

รอบ 2 (P_{111}) $c_{111} \leftarrow c_{111} + c_{11n/4+1}$, (P_{112}) $c_{112} \leftarrow c_{112} + c_{11n/4+2}$,

$\ldots$, $(P_{11n/4})$ $c_{11n/4} \leftarrow c_{11n/4} + c_{11n/2}$

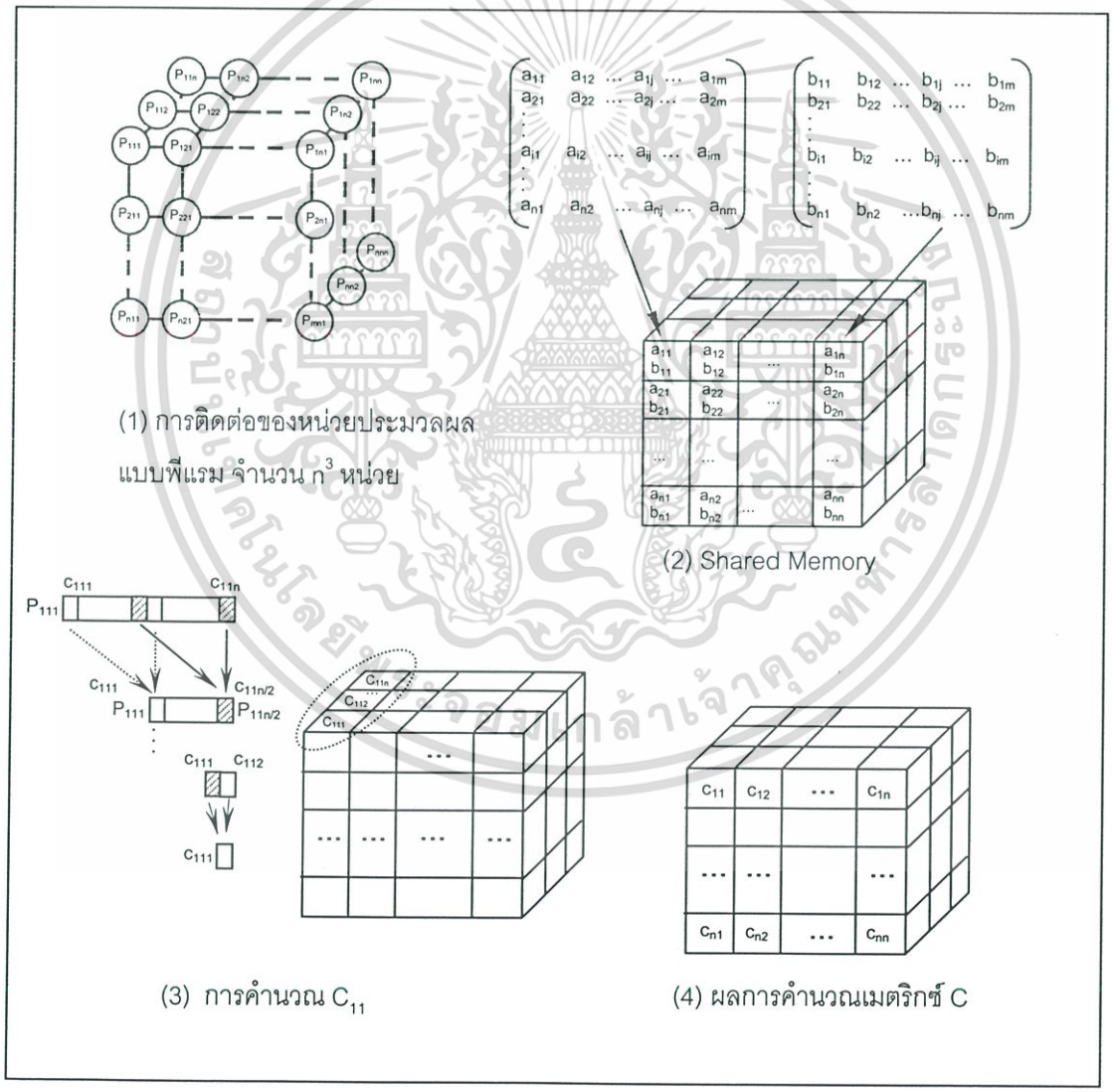
.

.

.

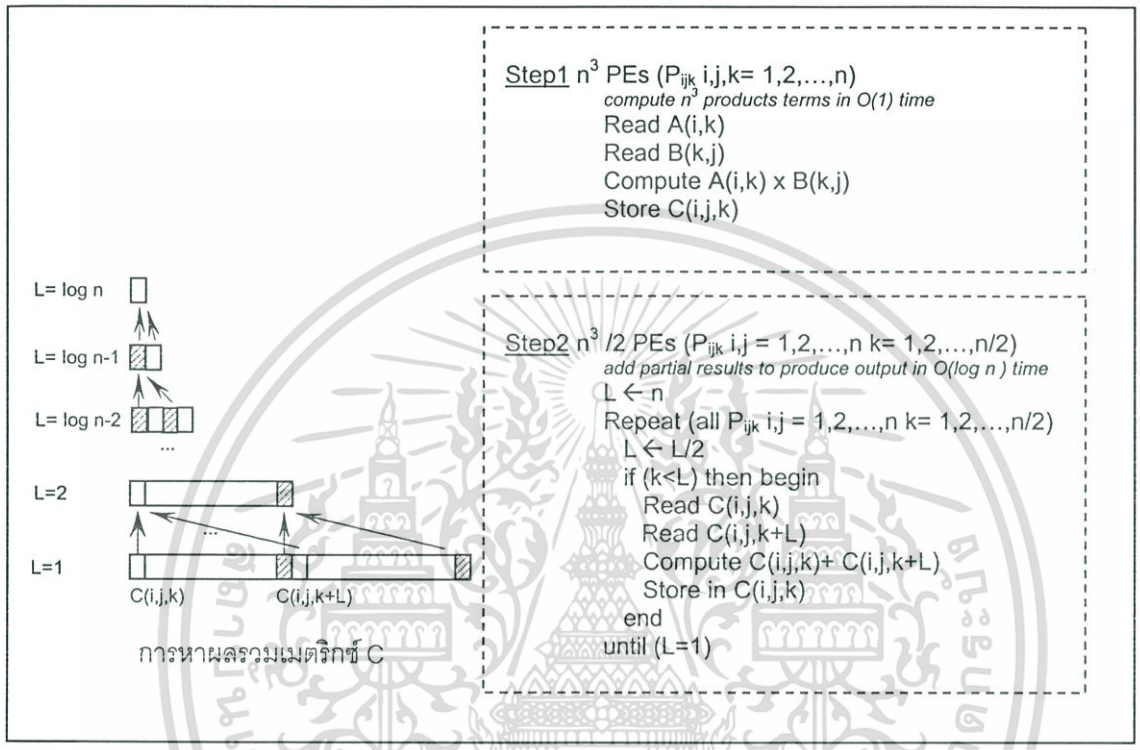
รอบ $\log n$ (P_{111}) $c_{111} \leftarrow c_{111} + c_{112}$

ทุกๆ หน่วยประมวลผล (P_{ijk}) ทำการประมวลผล c_{ij} พร้อมๆ กัน เมื่อ i,j,k = 1,2, ..., n/2 เป็นจำนวน log n รอบ จะได้ผลคูณเมตริกซ์ C เก็บอยู่ในแนวแกน x และ y ดังแสดงในรูปที่ 2.26(4) โดยมีค่าความซับซ้อนด้านเวลา (Time Complexity) เท่ากับ O(log n)



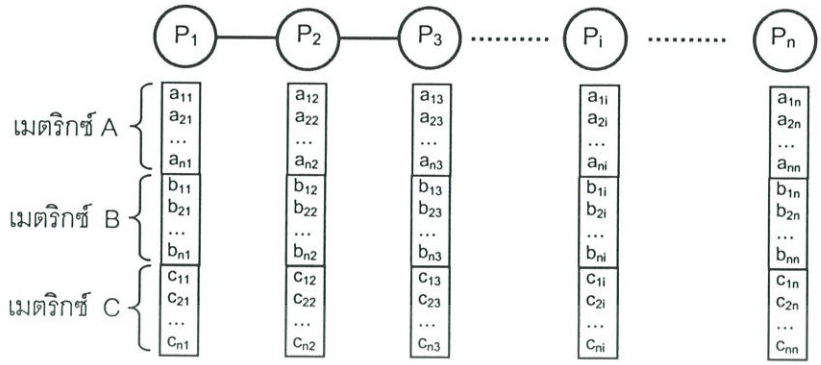
รูปที่ 2.26 การหาผลคูณของเมตริกซ์บนการติดต่อแบบพีแรมซีอาร์อีดับเบิลยู (PRAM-CREW- nxnxn (cube))

ขั้นตอนทางคอมพิวเตอร์ของการหาผลคูณเมตริกซ์แบบพีแรมซีอาร์อีดับเบิลยู (PRAM-CREW- nxnxn(cube)) แสดงได้ดังนี้



รูปที่ 2.27 ขั้นตอนทางคอมพิวเตอร์ของการหาผลคูณเมตริกซ์บนการติดต่อแบบพีแรมซีอาร์อีดับเบิลยู (PRAM-CREW-nxnxn(cube))

2.6.2 การหาผลคูณของเมตริกซ์แบบขนานบนการติดต่อแบบอะเรย์เชิงเส้น



รูปที่ 2.28 การหาผลคูณของเมตริกซ์แบบขนานบนการติดต่อแบบอะเรย์เชิงเส้น

การหาผลคูณของเมตริกซ์แบบขนาน โดยมีการติดต่อระหว่างหน่วยประมวลผลเป็นแบบ SIMD อะเรย์เชิงเส้น ขนาด P = n หน่วยประมวลผล ดังแสดงในรูป 2.28 เมตริกซ์ C เป็นผลคูณระหว่าง เมตริกซ์ A และเมตริกซ์ B ขนาด nxn ประกอบด้วย 2 ขั้นตอน คือ

ขั้นตอนที่ 1 แบ่งชุดข้อมูลให้แต่ละหน่วยประมวลผลโดยที่ P_i เก็บค่าเมตริกซ์หลักที่ i เมื่อ i = 1,2,…,n ของเมตริกซ์ A และเมตริกซ์ B

ขั้นตอนที่ 2 ทุกหน่วยประมวลผล P_i ทำการคำนวณหาค่าเฉพาะข้อมูลที่ได้รับ ดังนี้ $c_{ik} = c_{ik} + a_{ij} \times b_{jk}$ โดยที่ค่า a_{ij} จะถูกส่งมาจากหน่วยควบคุม (CU) ไปยังทุกหน่วยประมวลผลทำงาน (P_i)

<u>หมายเหตุ</u> หน่วยประมวลผล P_i คำนวณค่า c_{ij} ทุกค่า j เมื่อ j = 1,2,…,n ในหลักที่ i

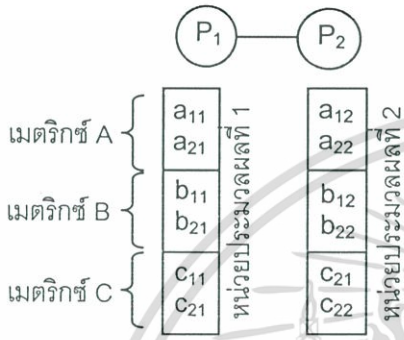
การพัฒนาโปรแกรมคอมพิวเตอร์ของการหาผลคูณของเมตริกซ์แบบขนาน โดยมีการติดต่อระหว่างหน่วยประมวลผลเป็นแบบอะเรย์เชิงเส้น [15]

```
for i =1 to n do
   for k =1 to n pardo
      cik = 0          //initialize
   end for all k
   for j =1 to n do
      for k =1 to n pardo
          cik = cik+aij x bjk
      end for all k
   end for j
end for i
```

รูปที่ 2.29 ขั้นตอนทางคอมพิวเตอร์ของการหาผลคูณเมตริกซ์บนการติดต่อแบบอะเรย์เชิงเส้น

ตัวอย่างที่ 2.1 การหาผลคูณของเมตริกซ์แบบขนานขนาด 2x2 โดยมีการติดต่อระหว่างหน่วยประมวลผลเป็นแบบอะเรย์เชิงเส้น จำนวน 2 หน่วยประมวลผล (P=2)

เมตริกซ์ C $\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$ = เมตริกซ์ A $\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ x เมตริกซ์ B $\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$



รอบที่ i	j	หน่วยประมวลผลที่ 1	หน่วยประมวลผลที่ 2
1	1	$c_{11} = c_{11}+a_{11}\ b_{11}$	$c_{12} = c_{12}+a_{11}\ b_{12}$
1	2	$c_{11} = c_{11}+a_{12}\ b_{21}$	$c_{12} = c_{12}+a_{12}\ b_{22}$
2	1	$c_{21} = c_{21}+a_{21}\ b_{11}$	$c_{22} = c_{22}+a_{21}\ b_{12}$
2	2	$c_{21} = c_{21}+a_{22}\ b_{21}$	$c_{22} = c_{22}+a_{22}\ b_{22}$

รูปที่ 2.30 ตัวอย่างการหาผลคูณของเมตริกซ์แบบขนานบนการติดต่อแบบอะเรย์เชิงเส้นขนาด P=2

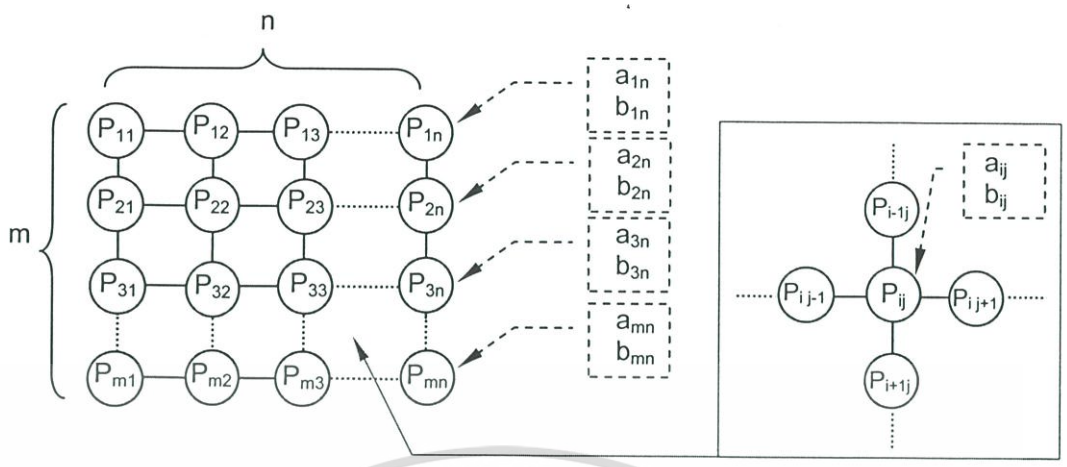
จำนวนที่ใช้ในการประมวลผลเท่ากับ 4 รอบ (i= 1,2,…,n และ j=1,2,…,n เมื่อ n=2) ดังนี้ (ดังแสดงในรูปที่ 2.30)

ขั้นตอนที่ 1 หน่วยประมวลผล P_1 เก็บข้อมูลหลักที่ 1 และหน่วยประมวลผล P_2 เก็บข้อมูลหลักที่ 2 ของเมตริกซ์ A และเมตริกซ์ B

รอบที่ i=1 j=1 ค่า a_{11} ถูกกระจาย (broadcast) จาก CU ไปยังหน่วยประมวลผล P_1 และหน่วยประมวลผล P_2 โดยหน่วยประมวลผล P_1 คำนวณ $c_{11}=a_{11}b_{11}$ และหน่วยประมวลผล P_2 คำนวณ $c_{12}=a_{11}b_{12}$ และเมื่อ j=2 ค่า a_{12} ถูกกระจายจาก CU ไปยังหน่วยประมวลผล P_1 และหน่วยประมวลผล P_2 โดยหน่วยประมวลผล P_1 คำนวณ $c_{11}= c_{11}+ a_{12}b_{21}$ และหน่วยประมวลผล P_2 คำนวณ $c_{12}= c_{12}+ a_{12}b_{22}$

รอบที่ i=2 j=1 ค่า a_{21} ถูกกระจายจาก CU ไปยังหน่วยประมวลผล P_1 และหน่วยประมวลผล P_2 โดยหน่วยประมวลผล P_1 คำนวณ $c_{21}=a_{21}b_{11}$ และหน่วยประมวลผล P_2 คำนวณ $c_{22}=a_{21}b_{12}$ และเมื่อ j=2 ค่า a_{22} ถูกกระจายจาก CU ไปยังหน่วยประมวลผล P_1 และหน่วยประมวลผล P_2 โดยหน่วยประมวลผล P_1 คำนวณ $c_{21}= c_{21}+ a_{22}b_{21}$ และหน่วยประมวลผล P_2 คำนวณ $c_{22}= c_{22}+ a_{22}b_{22}$

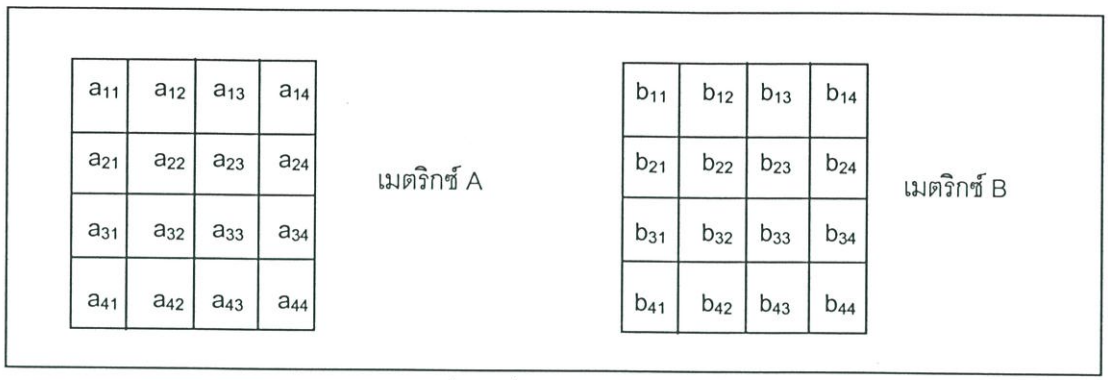
2.6.3 การหาผลคูณของเมตริกซ์แบบขนานบนการติดต่อแบบเมสช์ (Mesh)



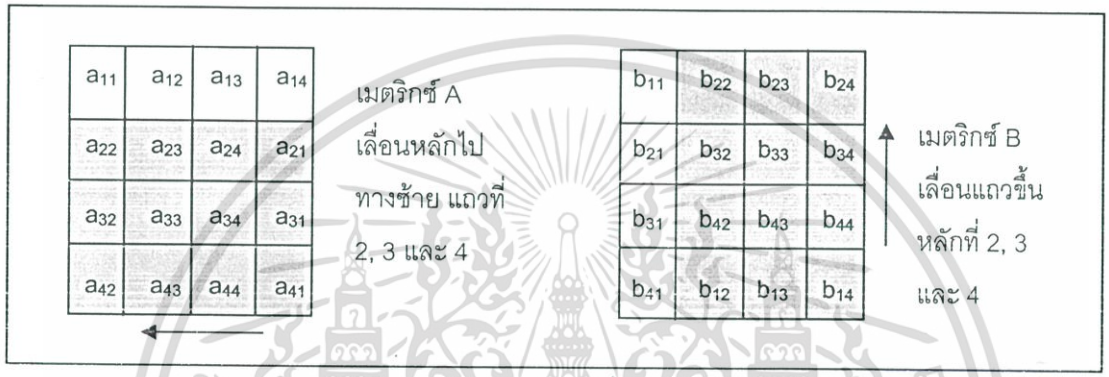
การหาผลคูณของเมตริกซ์แบบขนานโดยมีการติดต่อระหว่างหน่วยประมวลผลเป็นแบบเมสช์ มี 3 ขั้นตอนดังนี้

ขั้นตอนที่ 1 แต่ละหน่วยประมวลผล (P_{ij}) โหลดข้อมูล 1 ค่าจากเมตริกซ์ A (a_{ij}) และ 1 ค่าจากเมตริกซ์ B (b_{ij}) มาเก็บไว้ในหน่วยความจำของตัวเอง

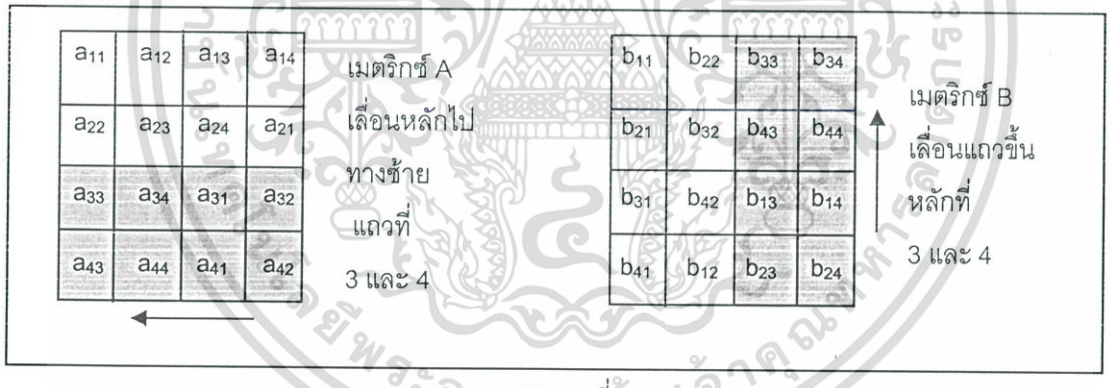
ขั้นตอนที่ 2 ทำการจัดเรียงข้อมูลให้อยู่ในรูปแบบที่หาผลคูณได้พร้อมๆ กัน โดยเลื่อนหลักของเมตริกซ์ A ไปทางซ้ายและในขณะเดียวกันเลื่อนแถวของเมตริกซ์ B ขึ้นด้านบน ทั้งหมด n-1 รอบ โดยมีเงื่อนไขดังนี้ แต่ละรอบทุกหน่วยประมวลผล P_{ij} ตรวจสอบเงื่อนไขและรับค่า $a_{i,j+1}$ จาก $P_{i,j+1}$ เมื่อ $i > k$ และ $b_{i+1,j}$ จาก $P_{i+1,j}$ เมื่อ $j > k$ ดังแสดงในรูปที่ 2.31 จากนั้น P_{ij} ทำการหาผลคูณเมตริกซ์ A และเมตริกซ์ B คู่แรกทั้งหมด n คู่ จากที่เก็บอยู่ในหน่วยความจำของแต่ละ P_{ij} ณ เวลานั้น คือ $a_{11}b_{11}$ ใน P_{11} , $a_{12}b_{22}$ ใน P_{12} , $a_{13}b_{33}$ ใน P_{13} , ... , $a_{43}b_{34}$ ใน P_{44} ดังแสดงในรูปที่ 2.32 (ซึ่งเป็นผลคูณบางส่วนเฉพาะที่ขีดเส้นใต้) ส่วนผลคูณที่เหลือทั้งหมดจะแสดงในขั้นตอนที่ 3



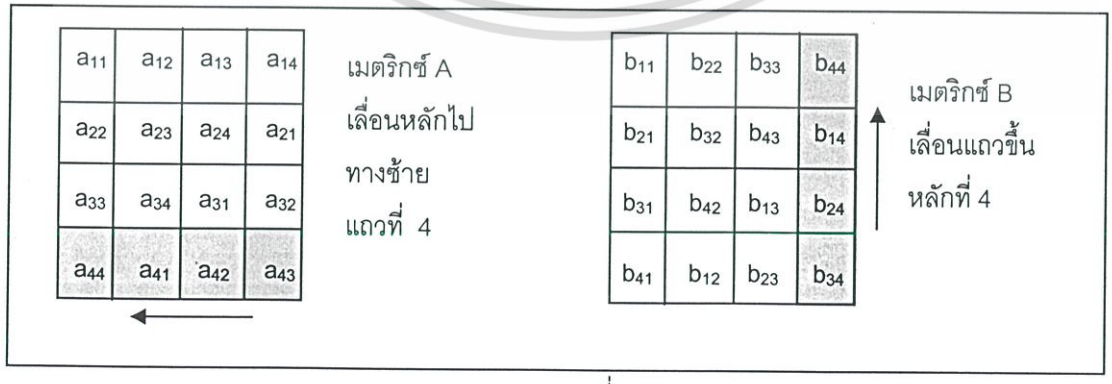
(1) ค่าเริ่มต้นที่เก็บก่อนการจัดเรียงข้อมูล



(2) รอบที่ 1

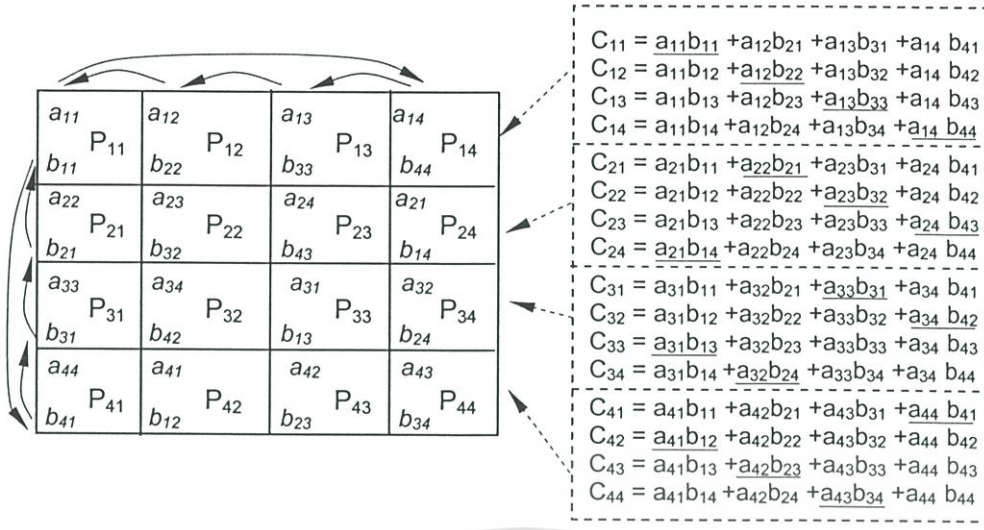


(3) รอบที่ 2



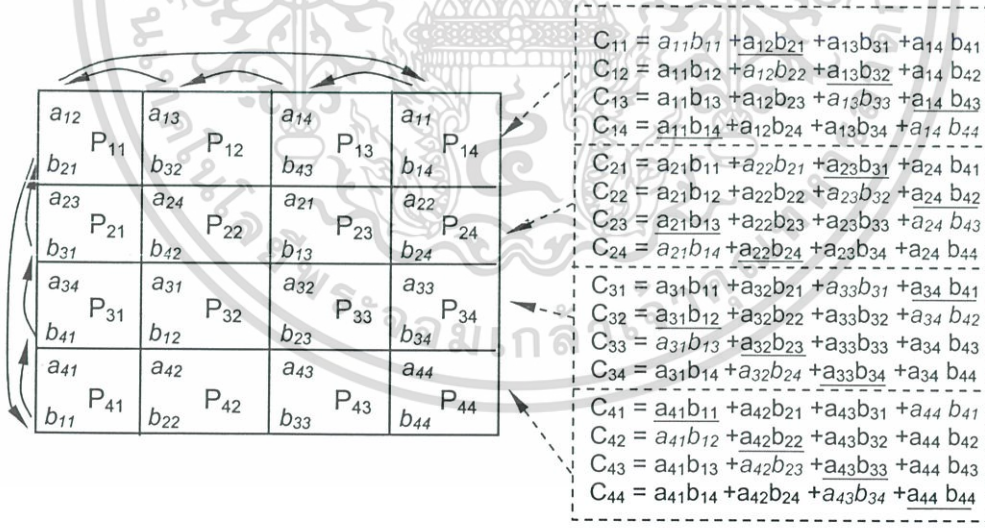
(4) รอบที่ 3

รูปที่ 2.31 การจัดข้อมูลในการหาผลคูณเมตริกซ์บนการติดต่อแบบเมสช์



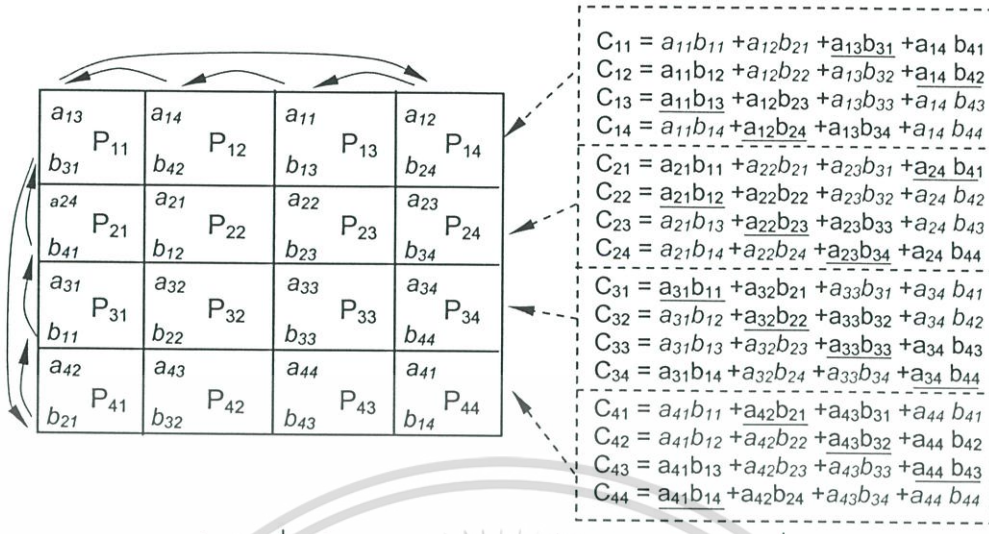
รูปที่ 2.32 การหาผลคูณเมตริกซ์บนการติดต่อแบบเมสช์ ขั้นตอนที่ 2

ขั้นตอนที่ 3 ทำการหาผลคูณอีก n-1 คู่ เป็นจำนวน n-1 รอบ เพื่อรวมเป็นค่า c_{ij} ในแต่ละ P_{ij} โดยในแต่ละรอบทำการเลื่อนข้อมูลไปทางซ้ายของเมตริกซ์ A และเลื่อนข้อมูลขึ้นของเมตริกซ์ B และทำการหาผลคูณเมตริกซ์ A และเมตริกซ์ B ทั้งหมด n คู่ จากที่เก็บอยู่ในหน่วยความจำของแต่ละ P_{ij} ณ เวลานั้น คือ $a_{11}b_{11}$ ใน P_{11}, $a_{12}b_{22}$ ใน P_{12}, $a_{13}b_{33}$ ใน P_{13}, ..., $a_{43}b_{34}$ $a_{12}b_{21}$ $a_{13}b_{32}$ ... $a_{14}b_{44}$ และทำต่อไปจนครบรอบ

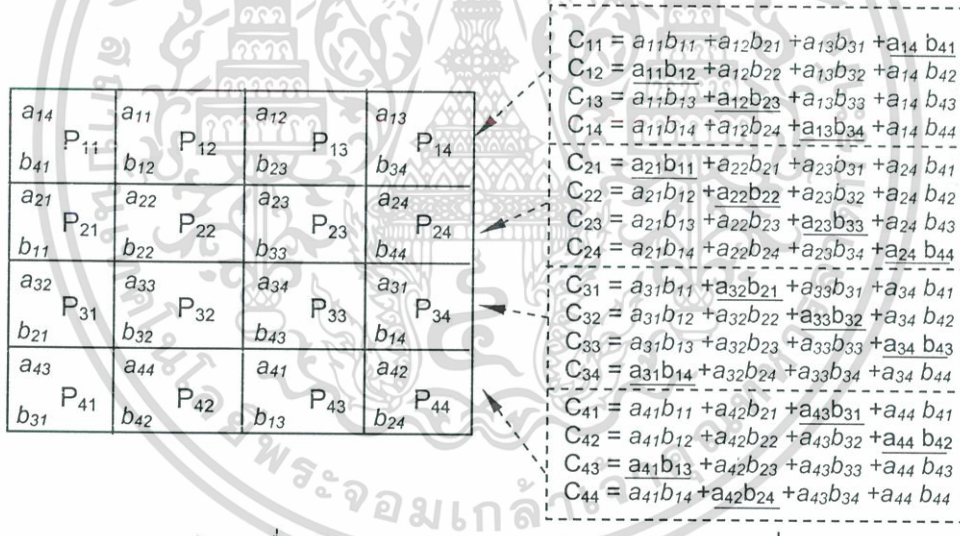


(1) ทำการเลื่อนข้อมูล และหาผลคูณเมตริกซ์รอบถัดไป (รอบที่1)

รูปที่ 2.33 การหาผลคูณเมตริกซ์บนการติดต่อแบบเมสช์ ขั้นตอนที่ 3



(2) ทำการเลื่อนข้อมูล และหาผลคูณเมตริกซ์รอบถัดไป (รอบที่2)



(3) ทำการเลื่อนข้อมูล และหาผลคูณเมตริกซ์รอบถัดไป (รอบที่3)

รูปที่ 2.33 การหาผลคูณเมตริกซ์บนการติดต่อแบบเมสช์ ขั้นตอนที่ 3 (ต่อ)

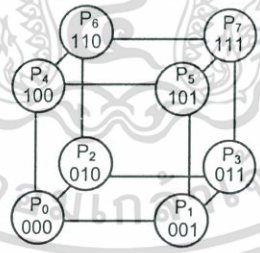
ขั้นตอนทางคอมพิวเตอร์ของการหาผลคูณเมตริกซ์แบบขนานโดยมีการติดต่อระหว่างหน่วยประมวลผลเป็นแบบเมสช์ [15] ดังแสดงในรูปที่ 2.34

```
Step 1: Stageger matrix (initialize)
 for  k = 1 to n-1 do
    for all Pij pardo
          if (i > k) A (i,j) ← A (i,j+1)
          if (j > k) B (i,j) ←  B (i+1,j)
     end for all
end for
for all Pij pardo
          C (i,j) = A (i,j) x B (i,j)
 end for all

Step 2
 for  k = 1 to n-1 do
    for all Pij pardo
          A (i,j) = A (i,j+1)
          B (i,j) = B (i+1,j)
          C (i,j) = A (i,j) x B (i,j)
     end for all
end for
```

รูปที่ **2.34** ขั้นตอนทางคอมพิวเตอร์ของการหาผลคูณเมตริกซ์บนการติดต่อแบบเมสช์

2.6.4 การหาผลคูณของเมตริกซ์แบบขนานบนการติดต่อแบบไฮเปอร์คิว (Hypercube)



การหาผลคูณของเมตริกซ์ (nxn) แบบขนานโดยมีการติดต่อระหว่างหน่วยประมวลผลเป็นแบบไฮเปอร์คิว ขนาด $n^3=2^{3q}$ หน่วย โดยแต่ละหน่วยประมวลผล Pi เมื่อ $0 \le i \le 2^{3q}-1$ มีตัวแปรในหน่วยความจำ คือ a, b, c, s, t และมีรายละเอียดขั้นตอนการหาผลคูณของเมตริกซ์ดังนี้

เมตริกซ์ C = เมตริกซ์ A × เมตริกซ์ B

$$\begin{pmatrix} c_{00} & c_{01} \dots c_{0j} \dots c_{0\,n-1} \\ c_{10} & c_{11} \dots c_{1j} \dots c_{1\,n-1} \\ \vdots & \\ c_{n-1\,0} & c_{n-1\,1} \dots c_{n-1\,j} \dots c_{n-1\,n-1} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \dots a_{0j} \dots a_{0\,n-1} \\ a_{10} & a_{11} \dots a_{1j} \dots a_{1\,n-1} \\ \vdots & \\ a_{n-1\,0} & a_{n-1\,1} \dots a_{n-1\,j} \dots a_{n-1\,n-1} \end{pmatrix} \begin{pmatrix} b_{00} & b_{01} \dots b_{0j} \dots b_{0\,n-1} \\ b_{10} & b_{11} \dots b_{1j} \dots b_{1\,n-1} \\ \vdots & \\ b_{n-1\,0} & b_{n-1\,1} \dots b_{n-1\,j} \dots b_{n-1\,n-1} \end{pmatrix}$$

row 1 col 1 : $c_{00} = a_{00}b_{00} + a_{01}b_{10} + \dots + a_{0\,n-1}b_{n-1\,0}$

... col n : $c_{0n-1} = a_{00}b_{0n-1} + a_{01}b_{1n-1} + \dots + a_{0\,n-1}b_{n-1\,n-1}$

row 2 col 1 : $c_{10} = a_{10}b_{00} + a_{11}b_{10} + \dots + a_{1\,n-1}b_{n-1\,0}$

... col n : $c_{1n-1} = a_{10}b_{1\,n-1} + a_{11}b_{1\,n-1} + \dots + a_{1\,n-1}b_{n-1\,n-1}$

...

row i col 1 : $c_{i\,0} = a_{i0}\,b_{00} + a_{i1}\,b_{10} + \dots + a_{in-1}\,b_{i0}$

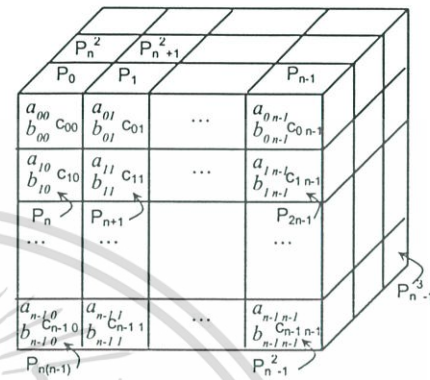
... col j : $c_{i\,j} = a_{i\,0}b_{0\,j} + a_{i\,1}\,b_{1\,j} + \dots + a_{ij}\,b_{j\,j}$

... col n : $c_{i\,n-1} = a_{i\,0}b_{0\,n-1} + a_{i\,1}\,b_{1\,n-1} + \dots + a_{i\,n-1}\,b_{i\,n-1}$

...

row n col 1 : $c_{n-1\,0} = a_{n-10}\,b_{00} + a_{n-11}\,b_{10} + \dots + a_{n-1n-1}\,b_{n-1\,0}$

... col n : $c_{n-1\,n-1} = a_{n-1\,0}b_{0\,n-1} + a_{n-1\,1}\,b_{1\,n-1} + \dots + a_{n-1\,n-1}\,b_{n-1\,n-1}$



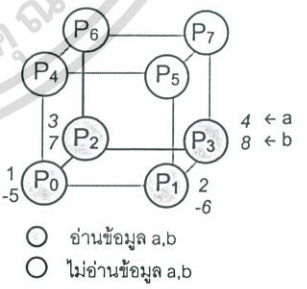
รูปที่ 2.35 การหาผลคูณเมตริกซ์บนการติดต่อแบบไฮเปอร์คิว

ขั้นตอนที่ 1 เริ่มแรกค่า a_{ij} และ b_{ij} เมื่อ $0 \le i,j \le n-1$ ถูกอ่านเข้าเก็บในตัวแปร a และ b ของหน่วยประมวลผล P_x เมื่อ $x = 2^q i+j = ni+j$ ดังนั้นถ้าสมมติ P=8, n=2 และ q=1 เพื่อใช้ในการคูณเมตริกซ์ A และเมตริกซ์ B ขนาด 2x2 ดังนี้

เมตริกซ์ A × เมตริกซ์ B = เมตริกซ์ C

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} -5 & -6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix}$$

P_0	$a_{00} = 1$	$b_{00} = -5$	P_1	$a_{01} = 2$	$b_{01} = -6$
P_2	$a_{10} = 3$	$b_{10} = 7$	P_3	$a_{11} = 4$	$b_{11} = 8$
P_4	-		P_5	-	
P_6	-		P_7	-	



ขั้นตอนที่ 2 กระจายข้อมูล (Broadcast) เมตริกซ์ A และเมตริกซ์ B โดยใช้ฟังก์ชัน BIT และ BIT.COMPLEMENT ดังนี้

- ฟังก์ชัน BIT รับค่าเลขจำนวนเต็ม m และ *l* และส่งค่าของบิทที่ *l* ในรูปแบบเลขฐาน 2 ของค่า m
- ฟังก์ชัน BIT.COMPLEMENT รับค่าเลขจำนวนเต็ม m และ *l* และส่งค่าเลขจำนวนเต็มที่ถูกสร้างโดยการ Complement ค่าในบิทที่ *l* (เลขฐาน 2) ของ m

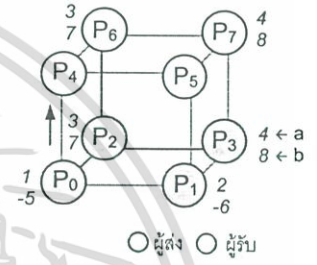
<u>หมายเหตุ</u> ฟังก์ชัน BIT และ BIT.COMPLEMENT ใช้ในขั้นตอนทางคอมพิวเตอร์ของการหาผลคูณเมตริกซ์ไฮเปอร์คิว (รูปที่ 2.36)

ขั้นตอนที่ 2.1 การรับ-ส่งค่า a และ b ใน Phase 1 (for $l \leftarrow 3q\text{-}1$ downto $2q$) ของขั้นตอนทางคอมพิวเตอร์ (รูปที่ 2.36) ผู้รับ คือ ทุก P_m ที่มี BIT(m,l) เท่ากับ 1, l=2 เมื่อ q=1 และผู้ส่ง คือ ทุก P_t เมื่อ t=BIT.COMPLEMENT (m,l) ตัวอย่างเช่น ทุก P_m คำนวณค่า BIT(m,l) เมื่อ m=0,1,2,3,…,7

P_0 BIT(0,2) = 0 เนื่องจากเลขฐาน 2 ของ 0 คือ $0\underline{000}_2$ ค่าในบิทที่ 2 คือ 0

BIT.COMPLEMENT (0,2) = 4 เนื่องจากเลขฐาน 2 ของ 0

คือ $0\underline{000}_2$ Complement ค่าในบิทที่ 2 คือ $0\underline{100}_2 = 4$



ในทำนองเดียวกัน

P_1 BIT(1,2) = 0 [เนื่องจาก $m = 0\underline{001}_2$]

P_2 BIT(2,2) = 0 [เนื่องจาก $m = 0\underline{010}_2$]

P_3 BIT(3,2) = 0 [เนื่องจาก $m = 0\underline{011}_2$]

ผู้รับ

P_4 BIT(4,2) = 1 [เนื่องจาก $m = 0\underline{100}_2$], BIT.COMPLEMENT (4,2) = 0_{10} [= $0\underline{000}_2$]

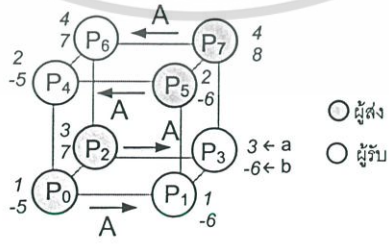
P_5 BIT(5,2) = 1 [เนื่องจาก $m = 0\underline{101}_2$], BIT.COMPLEMENT (5,2) = 1_{10} [= $0\underline{001}_2$]

P_6 BIT(6,2) = 1 [เนื่องจาก $m = 0\underline{110}_2$], BIT.COMPLEMENT (6,2) = 2_{10} [= $0\underline{010}_2$]

P_7 BIT(7,2) = 1 [เนื่องจาก $m = 0\underline{111}_2$], BIT.COMPLEMENT (7,2) = 3_{10} [= $0\underline{011}_2$]

ผู้ส่ง

ขั้นตอนที่ 2.2 การรับ-ส่งค่า a ใน Phase 1 (for $l \leftarrow q\text{-}1$ downto 0) ของขั้นตอนทางคอมพิวเตอร์ (รูปที่ 2.36) ผู้รับ คือ ทุก P_m ที่มี BIT(m,l) **ไม่เท่ากับ** BIT(m,2q+l), l=0 เมื่อ q=1 และผู้ส่ง คือ ทุก P_t เมื่อ t=BIT.COMPLEMENT (m,l) ตัวอย่างเช่น ทุก P_m คำนวณค่า BIT(m,l) และ BIT(m,2q+l) เมื่อ m=0,1,2,3,…,7



P_0 BIT(0,0) = 0 [เนื่องจาก $m = 000\underline{0}_2$], BIT(0,2*1+0) = 0 [เนื่องจาก $m = 0\underline{000}_2$] => เท่ากัน

P_1 BIT(1,0) = 1 [เนื่องจาก $m = 000\underline{1}_2$], BIT(1,2*1+0) = 0 [เนื่องจาก $m = 0\underline{001}_2$] => **ไม่เท่ากัน**

ดังนั้น P_1 เป็นผู้รับค่าที่ส่งมาจาก P_t เมื่อ t = BIT.COMPLEMENT (1,0) = 0_{10} [= $000\underline{0}_2$]

P_2 BIT(2,0) = 0 [เนื่องจาก $m = 001\underline{0}_2$], BIT(2,2*1+0) = 0 [เนื่องจาก $m = 0\underline{010}_2$] => เท่ากัน

P_3 BIT(3,0) = 1 [เนื่องจาก m = $001\underline{1}_2$], BIT(3,2*1+0) = 0 [เนื่องจาก m = $0\underline{0}11_2$] => **ไม่เท่ากัน**
ดังนั้น P_3 เป็นผู้รับค่าที่ส่งมาจาก P_t เมื่อ t=BIT.COMPLEMENT (3,0) = 2_{10} [= $001\underline{0}_2$]

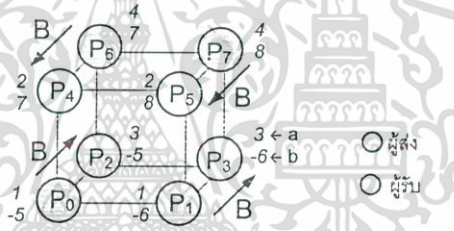
P_4 BIT(4,0) = 0 [เนื่องจาก m = $010\underline{0}_2$], BIT(4,2*1+0) = 1 [เนื่องจาก m = $0\underline{1}00_2$] => **ไม่เท่ากัน**
ดังนั้น P_4 เป็นผู้รับค่าที่ส่งมาจาก P_t เมื่อ t=BIT.COMPLEMENT (4,0) = 5_{10} [= $010\underline{1}_2$]

P_5 BIT(5,0) = 1 [เนื่องจาก m = $010\underline{1}_2$], BIT(5,2*1+0) = 1 [เนื่องจาก m = $0\underline{1}01_2$] => เท่ากัน

P_6 BIT(6,0) = 0 [เนื่องจาก m = $011\underline{0}_2$], BIT(6,2*1+0) = 1 [เนื่องจาก m = $0\underline{1}10_2$] => **ไม่เท่ากัน**
ดังนั้น P_6 เป็นผู้รับค่าที่ส่งมาจาก P_t เมื่อ t=BIT.COMPLEMENT (6,0) = 7_{10} [= $011\underline{1}_2$]

P_7 BIT(7,0) = 1 [เนื่องจาก m = $011\underline{1}_2$], BIT(7,2*1+0) = 1 [เนื่องจาก m = $0\underline{1}11_2$] => เท่ากัน

ขั้นตอนที่ 2.3 การรับ-ส่งค่า b ใน Phase 1 (for *l* ← 2q-1 downto q) ของขั้นตอนทางคอมพิวเตอร์ (รูปที่ 2.36) ผู้รับ คือ ทุก P_m ที่มี BIT(m,*l*) **ไม่เท่ากับ** BIT(m,q+*l*) , *l*=1 เมื่อ q=1 และผู้ส่ง คือ ทุก P_t เมื่อ t=BIT.COMPLEMENT (m,*l*) ตัวอย่างเช่น ทุก P_m คำนวณค่า BIT(m,*l*) และ BIT(m,q+*l*) เมื่อ m=0,1,2,3,…,7



P_0 BIT(0,1) = 0 [เนื่องจาก m = $000\underline{0}_2$], BIT(0,1+1) = 0 [เนื่องจาก m = $0\underline{0}00_2$] => เท่ากัน

P_1 BIT(1,1) = 0 [เนื่องจาก m = $00\underline{0}1_2$], BIT(1,1+1) = 0 [เนื่องจาก m = $0\underline{0}01_2$] => เท่ากัน

P_2 BIT(2,1) = 1 [เนื่องจาก m = $00\underline{1}0_2$], BIT(2,1+1) = 0 [เนื่องจาก m = $0\underline{0}10_2$] => **ไม่เท่ากัน**
ดังนั้น P_2 เป็นผู้รับค่าที่ส่งมาจาก P_t เมื่อ t=BIT.COMPLEMENT (2,1) = 0_{10} [= $000\underline{0}_2$]

P_3 BIT(3,1) = 1 [เนื่องจาก m = $001\underline{1}_2$], BIT(3,1+1) = 0 [เนื่องจาก m = $0\underline{0}11_2$] => **ไม่เท่ากัน**
ดังนั้น P_3 เป็นผู้รับค่าที่ส่งมาจาก P_t เมื่อ t=BIT.COMPLEMENT (3,1) = 1_{10} [= $000\underline{1}_2$]

P_4 BIT(4,1) = 0 [เนื่องจาก m = $010\underline{0}_2$], BIT(4,1+1) = 1 [เนื่องจาก m = $0\underline{1}00_2$] => **ไม่เท่ากัน**
ดังนั้น P_4 เป็นผู้รับค่าที่ส่งมาจาก P_t เมื่อ t=BIT.COMPLEMENT (4,1) = 6_{10} [= $011\underline{0}_2$]

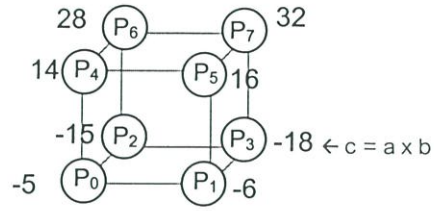
P_5 BIT(5,1) = 0 [เนื่องจาก m = $010\underline{1}_2$], BIT(5,1+1) = 1 [เนื่องจาก m = $0\underline{1}01_2$] => **ไม่เท่ากัน**
ดังนั้น P_5 เป็นผู้รับค่าที่ส่งมาจาก P_t เมื่อ t=BIT.COMPLEMENT (5,1) = 7_{10} [= $011\underline{1}_2$]

P_6 BIT(6,1) = 1 [เนื่องจาก m = $011\underline{0}_2$], BIT(6,1+1) = 1 [เนื่องจาก m = $0\underline{1}10_2$] => เท่ากัน

P_7 BIT(7,1) = 1 [เนื่องจาก m = $011\underline{1}_2$], BIT(7,1+1) = 1 [เนื่องจาก m = $0\underline{1}11_2$] => เท่ากัน

ขั้นตอนที่ 3 หาผลคูณเมตริกซ์ C ในแต่ละหน่วยประมวลผล ใน Phase 2 ของขั้นตอนทางคอมพิวเตอร์ (รูปที่ 2.36) ตัวอย่างเช่น ทุก P_m คำนวณค่า c = a x b เมื่อ m=0,1,2,3,...,7

<u>หมายเหตุ</u> ค่า a และ b เป็นค่าที่ได้จากขั้นตอนที่ 1 และขั้นตอนที่ 2



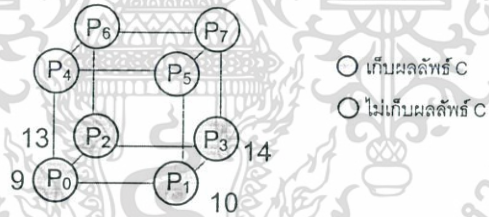
P_0 c = 1 x (-5) = -5 , P_1 c = 1 x (-6) = -6

P_2 c = 3 x (-5) = -15 , P_3 c = 3 x (-6) = -18

P_4 c = 2 x 7 = 14 , P_5 c = 2 x 8 = 16

P_6 c = 4 x 7 = 28 , P_7 c = 4 x 8 = 32

ขั้นตอนที่ 4 นำค่าเมตริกซ์ C ระหว่างหน่วยประมวลผลมารวมกันใน Phase 3 (for $l \leftarrow 2q$ downto 3q-1) ของขั้นตอนทางคอมพิวเตอร์(รูปที่ 2.36) l=2 เมื่อ q=1 ตัวอย่างเช่น ทุก P_m รับค่า c มาเก็บใน s จาก P_t และคำนวณค่า c = c+s เมื่อ m=0,1,2,3,...,7 ผลคูณเมตริกซ์ C จะอยู่ใน P_x เมื่อ x $= 2^q i+j$ และ $0 \leq i, j \leq n-1$ เมตริกซ์ C $\begin{bmatrix} 9 & 10 \\ 13 & 14 \end{bmatrix}$



P_0 t=BIT.COMPLEMENT (0,2) = 4_{10} [= $0\underline{1}00_2$],
รับค่า c จาก P_4 (ในขั้นตอนที่ 3) s <= [4]c = 14, c = c+s = -5+14 = 9

P_1 t=BIT.COMPLEMENT (1,2) = 5_{10} [= $0\underline{1}01_2$],
รับค่า c จาก P_5 (ในขั้นตอนที่ 3) s <= [5]c = 16, c = c+s = -6+16 = 10

P_2 t=BIT.COMPLEMENT (2,2) = 6_{10} [= $0\underline{1}10_2$],
รับค่า c จาก P_6 (ในขั้นตอนที่ 3) s <= [6]c = 28, c = c+s = -15+28 = 13

P_3 t=BIT.COMPLEMENT (3,2) = 7_{10} [= $0\underline{1}11_2$],
รับค่า c จาก P_7 (ในขั้นตอนที่ 3) s <= [7]c = 32, c = c+s = -18+32 = 14

P_4 t=BIT.COMPLEMENT (4,2) = 0_{10} [= $0\underline{0}00_2$],
รับค่า c จาก P_0 (ในขั้นตอนที่ 3) s <= [0]c = -5, c = c+s = 14-5 = 9

P_5 t=BIT.COMPLEMENT (5,2) = 1_{10} [= $0\underline{0}01_2$],
รับค่า c จาก P_1 (ในขั้นตอนที่ 3) s <= [1]c = -6, c = c+s = 16-6 = 10

P_6 t=BIT.COMPLEMENT (6,2) = 2_{10} [= $0\underline{0}10_2$],

รับค่า c จาก P_2 (ในขั้นตอนที่ 3) s <= [2]c = -15, c = c+s = 28-15 = 13

P_7 t=BIT.COMPLEMENT (7,2) = 3_{10} [= $0\underline{0}11_2$],

รับค่า c จาก P_3 (ในขั้นตอนที่ 3) s <= [3]c = -18, c = c+s = 32-18 = 14

ขั้นตอนทางคอมพิวเตอร์การพัฒนาโปรแกรมของการหาผลคูณเมตริกซ์ไฮเปอร์คิว[15] ดังแสดงในรูปที่ 2.36

```
MATRIX MULTIPLICATION (Hypercube SIMD)

Parameter q {Matrix size is 2^q x 2^q}  //n=2^q, q=log n
Global I
Local a,b,c,s,t
Begin

 {Phase1: Broadcast matrix A and B}
 for  I ← 3q-1 downto 2q do
  for all Pm where BIT(m,I)= 1 do
          t ← BIT.COMPLEMENT(m,I)
          a ⇐ [t]a
          b ⇐ [t]b
  endfor
 endfor
 for  I ← q-1 downto 0 do
  for all Pm where BIT(m,I) ≠BIT(m,2q+I)do
          t ← BIT.COMPLEMENT(m,I)
          a ⇐ [t]a
  endfor
 endfor

for  I ← 2q-1 downto q do
  for all Pm where BIT(m,I)≠BIT(m,q+I)do
          t ← BIT.COMPLEMENT(m,I)
          b ⇐ [t]b
   endfor
endfor
{Phase2: Do the multiplications in parallel}
for all Pm do
          c ← a x b
endfor

{Phase3:Sum the products}
for  I ← 2q to 3q-1  do
   for all Pm do
          t ← BIT.COMPLEMENT(m,I)
          s ⇐ [t]c
          c ← c+s
   endfor
endfor
End
```

รูปที่ 2.36 ขั้นตอนทางคอมพิวเตอร์ของการหาผลคูณเมตริกซ์บนการติดต่อแบบไฮเปอร์คิว

2.7 งานวิจัยที่เกี่ยวข้อง

ในปี 1994 Alonso Sanches C.A. และ Song S.W. [1] ทำการทดลองหาผลคูณของเมตริกซ์บนเครื่องคอมพิวเตอร์แบบขนานที่โครงสร้างของหน่วยประมวลผลติดต่อกันแบบไฮเปอร์คิวโดยใช้ขั้นตอนทางคอมพิวเตอร์แบบ SIMD และนำเสนอการมินิไมเซชั่นในเวลาที่ประมวลผล เรียกวิธี "MMM_1" โดยมีค่าความซับซ้อนด้านเวลาเท่ากับ $O(n^2/p^{2/3} \log p + n^\lambda/p^{\lambda/3})$ เมื่อ $1 \le p \le n^3$ มีประสิทธิภาพดีกว่าเมื่อเปรียบเทียบกับวิธีของ Dekel, Nassimi และ Sahni ซึ่งมีค่าความซับซ้อนด้านเวลาเท่ากับ $O(n^\lambda/p^{(\lambda-1)/2})$ เมื่อ $1 \le \lambda < 3$ และ $1 \le p \le n^2$และเท่ากับ $O(\log p/n^2+n^3/p)$ เมื่อ $n^2 \le p \le n^3$

ในปี 1996 Tasic J.F., Zajc M. และ Kosir A. [18] ทำการพัฒนาขั้นตอนทางคอมพิวเตอร์ในการหาผลคูณเมตริกซ์แบบ Systolic array โดยทำการเลื่อนลำดับการนำข้อมูลเข้าและเปลี่ยนลำดับของหน่วยประมวลผลในการคำนวณหาผลคูณของเมตริกซ์โดยทำให้เวลาในการประมวลผล (Computation Time) เร็วกว่าวิธีต้นแบบ (วิธีของ S. Y. Kung) เป็นจำนวน N-1 รอบ

ในปี 1997 Browne S., Dongarra J. และ London K. [3]ได้ทำการทดสอบและวิเคราะห์ประสิทธิภาพของเครื่องมือสำหรับโปรแกรมแบบขนานที่ใช้มาตรฐานภาษา MPI บนเครื่องคอมพิวเตอร์ IBM SP2 ประกอบด้วยเครื่องมือ AIMS, nupshot, Pablo, Paradyn, VAMPIR และ VTtrace พร้อมแสดงจุดเด่นและจุดด้อยของแต่ละเครื่องมือ

ในปี 1997 Choi Jaeyoung [4][5] สร้างขั้นตอนทางคอมพิวเตอร์ใหม่ในการหาผลคูณของเมตริกซ์บนระบบคอมพิวเตอร์แบบกระจายหน่วยความจำ เรียกว่า DIMMA (Distribution Independent Matrix Multiplication Algorithm) บนเครื่องคอมพิวเตอร์ Intel Paragon (XPS35) นำไปเปรียบเทียบกับวิธี SUMMA โดยใช้ 16x16 หน่วยประมวลผลให้ค่าประสิทธิภาพดีกว่าประมาณ 10 %

ในปี 1998 John Gunnels, Calvin Lin, Greg Morrow และ Robert van de Geijn [8] แสดงปัญหาของขั้นตอนทางคอมพิวเตอร์การหาผลคูณของเมตริกซ์โดยการใช้ขนาดเมตริกซ์ A และเมตริกซ์ B ที่ต่างกัน ซึ่งทำให้เวลาในการประมวลผลหาผลคูณของเมตริกซ์แตกต่างกัน พร้อมได้ทำการสร้างแบบจำลองและเปรียบเทียบประสิทธิภาพกับการทดลองบนเครื่องคอมพิวเตอร์ Cray T3E

ในปี 1998 Sengupta A. และ Raghavendra C.S [17] พัฒนาขั้นตอนทางคอมพิวเตอร์ของการหาผลคูณของเมตริกซ์บนระบบคอมพิวเตอร์ไฮเปอร์คิว โดยใช้เวลาในการประมวลผล $2n\alpha + 2NL\beta$ วินาที

ในปี 2000 Li Keqin [12] เสนอทฤษฎีในการหาผลคูณของเมตริกซ์บนคอมพิวเตอร์แบบขนานที่มีหน่วยความจำแบบกระจาย DMPC (Distributed memory parallel computer) โดยเปลี่ยนแปลงจำนวนหน่วยประมวลผล และยังคงมีค่าความซับซ้อนด้านเวลาเท่ากับ O(log N) เมื่อใช้ $N^\alpha/\log N$ หน่วยประมวลผลและ $2 < \alpha \le 3$

ในปี 2004 Typou T., Stefanidis V., Michailidis P. และ Margaritis K. [19] ได้ทำการหาผลคูณของเมตริกซ์บนระบบคลัสเตอร์โดยใช้มาตรฐานภาษา MPI โดยได้เสนอแบบจำลองการคาดการณ์ประสิทธิภาพเปรียบเทียบกับการทดลองบนระบบคลัสเตอร์ (จำนวน 32 เครื่อง หน่วยประมวลผล 200 MHz หน่วยความจำ 64 Mb และความเร็วในการติดต่อ 100 Mbps) โดยใช้ dynamic master – worker แยกเป็น 4 วิธี คือ

1. Dynamic allocation of columns of the matrix คือ เมตริกซ์ A และเมตริกซ์ B ถูกเก็บอยู่ที่หน่วยความจำสำรอง (local disk) ของมาสเตอร์ทำการแบ่งแต่ละบล๊อคและกระจายให้กับเวิร์กเกอร์ และส่งบางส่วนของแต่ละแถวของเมตริกซ์ A กระจายให้กับเวิร์กเกอร์ และเมื่อแต่ละเวิร์กเกอร์คำนวณเสร็จจะส่งค่าเมตริกซ์ C คืนให้มาสเตอร์

2. Dynamic allocation of matrix pointers (matrix B master) คือ เมตริกซ์ A และ เมตริกซ์ B ถูกเก็บอยู่ที่หน่วยความจำสำรองของมาสเตอร์และส่งพอยท์เตอร์ของหลักที่จะทำการคำนวณของเมตริกซ์ B ให้เวิร์กเกอร์ และส่งบางส่วนของแต่ละแถวของเมตริกซ์ A กระจายให้กับเวิร์กเกอร์ เมื่อแต่ละเวิร์กเกอร์คำนวณจะทำการอ่านค่าหลักของเมตริกซ์ B จากหน่วยความจำสำรองของมาสเตอร์โดยเริ่มตั้งแต่พอยท์เตอร์ที่ได้รับจากมาสเตอร์ เมื่อคำนวณเสร็จจะส่งค่าเมตริกซ์ C คืนให้มาสเตอร์

3. Dynamic allocation of matrix pointers (matrix B worker) คือ เมตริกซ์ B ถูกเก็บอยู่ที่หน่วยความจำสำรองของแต่ละเวิร์กเกอร์ ส่วนเมตริกซ์ A เก็บอยู่ที่หน่วยความจำสำรองของมาสเตอร์ และจะส่งบางส่วนของแต่ละแถวของเมตริกซ์ A กระจายให้กับเวิร์กเกอร์ เมื่อแต่ละเวิร์กเกอร์คำนวณจะทำการอ่านค่าหลักของเมตริกซ์ B จากหน่วยความจำสำรองโดยเริ่มตั้งแต่พอยท์เตอร์ที่ได้ส่งมาจากมาสเตอร์ เมื่อคำนวณเสร็จจะส่งค่าเมตริกซ์ C คืนให้มาสเตอร์

4. Dynamic allocation of matrix pointers (matrix A,B worker) คือ เมตริกซ์ A และเมตริกซ์ B ถูกเก็บอยู่ที่หน่วยความจำสำรองของแต่ละเวิร์กเกอร์ เมื่อแต่ละเวิร์กเกอร์คำนวณจะทำการอ่านค่าหลักของเมตริกซ์ B จากหน่วยความจำสำรองโดยเริ่มตั้งแต่พอยท์เตอร์ที่ได้ส่งมาจากมาสเตอร์ และอ่านค่าเมตริกซ์ A จากหน่วยความจำ เมื่อคำนวณเสร็จจะส่งค่าเมตริกซ์ C คืนให้มาสเตอร์

พร้อมทั้งได้สร้างแบบจำลองการวัดประสิทธิภาพ คาดการณ์เวลาในการประมวลผล และได้นำมาเปรียบเทียบกับเวลาในการประมวลผลบนระบบพีซีคลัสเตอร์ โดยนำวิธีที่ 2,3 และ 4 มาทดลองเปรียบเทียบ ผลที่ได้เวลาในการประมวลผลจากการคาดการณ์และเวลาในการประมวลผลบนระบบพีซีคลัสเตอร์ มีค่าความแตกต่างกันอยู่ในช่วง 10-20%

บทที่ 3
การหาผลคูณของเมตริกซ์แบบขนานบนระบบพีซีคลัสเตอร์

วิทยานิพนธ์นี้เป็นการศึกษาวิธีการหาผลคูณของเมตริกซ์แบบขนานบนระบบพีซีคลัสเตอร์ ซึ่งเน้นการประยุกต์ใช้จริงบนระบบคอมพิวเตอร์แบบขนานที่มีอยู่ได้อย่างมีประสิทธิภาพ

ขั้นตอนของการวิจัยการหาผลคูณของเมตริกซ์แบบขนานมี 8 ขั้นตอนดังนี้

1. ศึกษาขั้นตอนวิธีการทางคอมพิวเตอร์ (Computer algorithm) การหาผลคูณของเมตริกซ์ แบบขนานและศึกษางานวิจัยที่เกี่ยวข้องกับการหาผลคูณของเมตริกซ์แบบขนาน
2. ทำการตั้งสมมติฐานโดยคาดว่าการประมวลการหาผลคูณของเมตริกซ์ด้วยโปรแกรมคอมพิวเตอร์แบบขนาน (Parallel) ซึ่งมีประสิทธิภาพจะใช้เวลาน้อยกว่าการประมวลผลการหาผลคูณของเมตริกซ์ด้วยโปรแกรมคอมพิวเตอร์แบบอนุกรม (Sequential)
3. นำเสนอวิธีการคูณกันของเมตริกซ์แบบขนานที่มีประสิทธิภาพบนระบบพีซีคลัสเตอร์ดังที่ได้ตั้งสมมติฐานในข้อ 2 เช่น วิธีการลดระยะเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลและลดการใช้เนื้อที่ในการเก็บข้อมูลที่ซ้ำซ้อนกันในระบบ
4. พัฒนาโปรแกรมแบบขนานของการหาผลคูณของเมตริกซ์โดยใช้ภาษา C บนระบบคลัสเตอร์ที่ใช้ระบบปฏิบัติการยูนิกซ์ โดยใช้โปรแกรม MPICH
5. ประเมินผลการทดลองโดยการเปรียบเทียบประสิทธิภาพของการหาผลคูณของเมตริกซ์จากเวลาทั้งหมดที่ใช้ในการประมวลผล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) ของการใช้ P หน่วยประมวลผลเมื่อเทียบกับการใช้เพียง 1 หน่วยประมวลผล
6. วิเคราะห์ผลการทดลองโดยระบุถึงปัจจัยที่มีผลต่อโปรแกรมแบบขนานของการหาผลคูณของเมตริกซ์
7. สรุปผลการทดลอง และเสนอแนะการศึกษาและพัฒนาขั้นต่อไปเกี่ยวกับการประมวลผลแบบขนาน
8. เขียนวิทยานิพนธ์

จากการศึกษา (ขั้นตอนที่ 1-3) และการพัฒนาโปรแกรมการหาผลคูณของเมตริกซ์แบบขนานที่มีประสิทธิภาพและนำมาทดลองประมวลผลจริงในระบบพีซีคลัสเตอร์ (ขั้นตอนที่ 4) โครงงานวิจัยนี้มุ่งเน้นที่การเพิ่มประสิทธิภาพในการติดต่อสื่อสารระหว่างหน่วยประมวลผลหรือ

ลดเวลาที่ใช้ในการติดต่อสื่อสารให้มากที่สุดเท่าที่ทำได้ และเพิ่มประสิทธิภาพของเนื้อที่ในการเก็บข้อมูลในแต่ละหน่วยประมวลผล หรือลดเนื้อที่ที่เก็บข้อมูลซ้ำซ้อนในระบบให้มากที่สุดเท่าที่ทำได้

ในบทนี้ผู้วิจัยเสนอวิธีการหาผลคูณของเมตริกซ์แบบขนาน 3 วิธี ซึ่งเป็นวิธีที่เหมาะสมกับระบบคลัสเตอร์ คือ

1) การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmm_w
(Row-Block Partitioning Matrix Multiplication with replicated data)
2) การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmm_wo
(Row-Block Partitioning Matrix Multiplication without replicated data)
3) การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี cbmm_w
(Checkerboard-Block Partitioning Matrix Multiplication with replicated data)

โดยวิธีแรก (วิธี rbmm_w) เป็นวิธีที่มีผู้เสนอไว้แล้วบนระบบพีซีคลัสเตอร์ซึ่งเป็นวิธีที่มีประสิทธิภาพในการติดต่อสื่อสารมากที่สุด แต่ใช้เนื้อที่ในการเก็บข้อมูลซ้ำซ้อนมากที่สุด ส่วนสองวิธีหลัง (วิธี rbmm_wo และวิธี cbmm_w) เป็นวิธีใหม่ที่เสนอขึ้นในวิทยานิพนธ์นี้เพื่อเพิ่มประสิทธิภาพโดยการลดการใช้ข้อมูลที่ซ้ำซ้อนกันระหว่างหน่วยประมวลผล โดยวิธีที่สองไม่มีการเก็บข้อมูลที่ซ้ำซ้อนกันเลยแต่เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลมีค่าเพิ่มมากขึ้น ส่วนวิธีที่สาม (วิธี cbmm_w) เป็นการนำข้อดีของวิธีที่ 1 (วิธี rbmm_w) และวิธีที่ 2 (วิธี rbmm_wo) มารวมกัน คือ จัดเก็บข้อมูลที่ซ้ำซ้อนกันบ้างแต่น้อยกว่าวิธีที่ 1 (หรือดีกว่าวิธีที่ 1) แต่มากกว่าวิธีที่ 2 และยังคงรักษาประสิทธิภาพในการติดต่อสื่อสารระหว่างหน่วยประมวลผลได้มากที่สุดเท่าวิธีที่ 1 ซึ่งดีกว่าวิธีที่ 2

3.1 การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmm_w (Row-Block partitioning Matrix Multiplication With replicated data)

วิธีการหาผลคูณเมตริกซ์แบบขนานวิธี rbmm_w จะประกอบด้วยหน่วยประมวลผลซึ่งแบ่งเป็น 2 ชนิด คือ หน่วยประมวลผลหลัก (Master computer) จำนวน 1 หน่วย และหน่วยประมวลผลทำงาน (Worker computer) จำนวน P หน่วย มีขั้นตอนในการทำงานดังต่อไปนี้

<u>หน่วยประมวลผลหลัก จำนวน 1 หน่วย</u>

ขั้นตอนที่ 1 สร้างข้อมูล คือ เมตริกซ์ A และเมตริกซ์ B ซึ่งมีขนาดใหญ่มาก (nxn) เก็บลงในหน่วยความจำโดยใช้เป็นเทคนิคหน่วยความจำเสมือน (Virtual memory)

ขั้นตอนที่ 2 ทำการแบ่งข้อมูลตามแถวของเมตริกซ์ A ขนาด $k = \lceil n/p \rceil$ เพื่อให้ภาระงาน (Work load) มีขนาดเท่ากัน (แสดงในรูปที่ 3.1) สำหรับหน่วยประมวลผลทำงาน P หน่วย เพื่อคำนวณหาผลคูณเมตริกซ์ C จำนวน $\lceil n/p \rceil$ แถวแบบขนาน

ขั้นตอนที่ 3 แจ้งช่วงและขนาดของเมตริกซ์ A ที่แต่ละหน่วยประมวลผลทำงานต้องทำการอ่านจากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก ส่วนเมตริกซ์ B ทุกหน่วยประมวลผลทำงาน ต้องอ่านทั้งหมดเพื่อนำมาเก็บไว้ในหน่วยความจำของตัวเอง

ขั้นตอนที่ 4 รอรับผลคูณเมตริกซ์ C จากหน่วยประมวลผลทำงาน P หน่วย

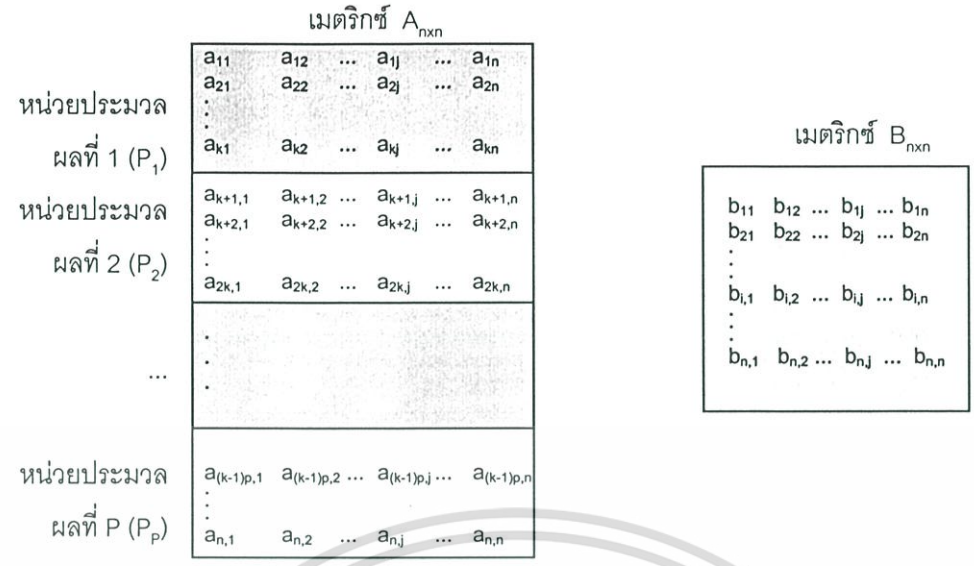
<u>หมายเหตุ</u> สำหรับ เมตริกซ์ A_{nxm} และ เมตริกซ์ B_{mxn} ค่า k เท่ากับ $\lceil n/p \rceil$

<u>หน่วยประมวลผลทำงาน จำนวน P หน่วย</u>

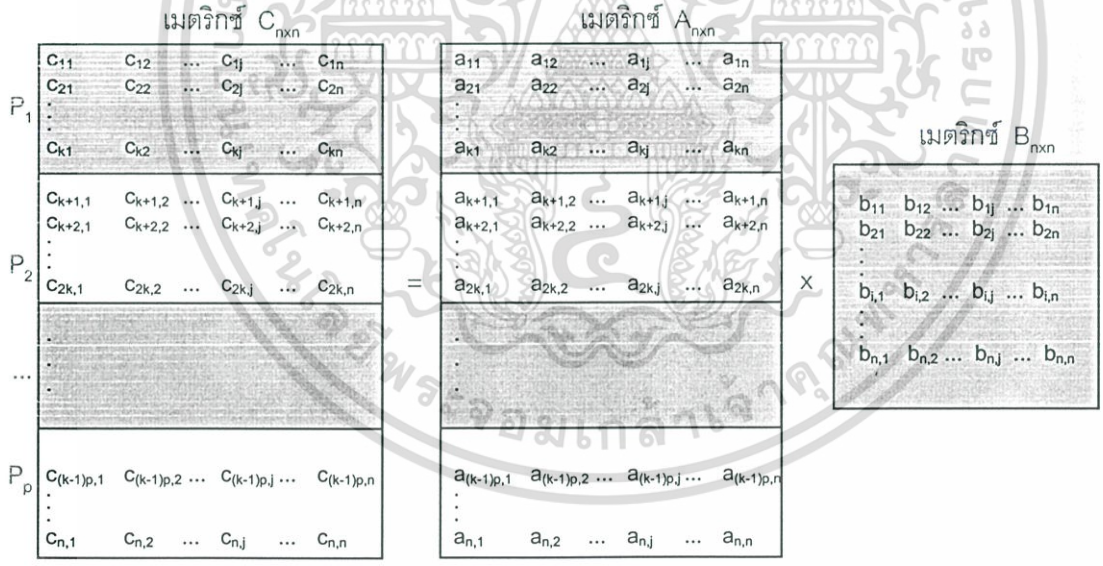
ขั้นตอนที่ 1 รอรับการติดต่อจากหน่วยประมวลผลหลักและหลังจากนั้น อ่านข้อมูลบางส่วนของเมตริกซ์ A ขนาด $k = \lceil n/p \rceil$แถว (แถวที่ (i-1)k+1-ik เมื่อ i หมายถึงไอดีของหน่วยประมวลผลทำงาน (Worker ID)) และเมตริกซ์ B ทั้งหมดจากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก

ขั้นตอนที่ 2 ทำการคำนวณผลคูณเมตริกซ์ C ในแถวที่ (i-1)k+1 ถึงแถวที่ ik

ขั้นตอนที่ 3 ส่งผลคูณของเมตริกซ์ C กลับไปยังหน่วยประมวลผลหลักดังแสดงในรูปที่ 3.2



รูปที่ 3.1 การแบ่งข้อมูลเมตริกซ์ A และ B สำหรับ P หน่วยประมวลผลของวิธี rbmm_w



รูปที่ 3.2 การแบ่งงานหาผลคูณเมตริกซ์ C ที่มี P หน่วยประมวลผลของวิธี rbmm_w

ตัวอย่างที่3.1 การหาผลคูณของเมตริกซ์แบบขนานระหว่างเมตริกซ์ A_{8x8} และ เมตริกซ์ B_{8x8} วิธี rbmm_wโดยใช้จำนวนหน่วยประมวลผล 4 หน่วยประมวลผล

เมตริกซ์ A

16	6	0	13	12	12	1	3
5	18	2	11	11	0	14	0
1	1	10	15	17	15	7	12
19	3	9	1	3	2	4	17
17	11	7	2	7	17	9	8
1	14	13	7	0	16	12	16
5	6	5	14	15	1	13	1
3	7	4	17	9	14	5	2

เมตริกซ์ B

3	11	18	4	5	4	10	18
7	16	0	12	11	4	3	3
3	9	12	8	15	11	14	4
11	4	16	11	16	1	19	2
18	12	11	9	4	6	11	6
5	0	4	8	9	16	7	6
13	14	12	10	18	17	4	8
14	17	11	7	6	3	17	13

วิธีทำ

ขั้นตอนที่ 1 หน่วยประมวลผลหลักแบ่งข้อมูลบางส่วนตามแถวของเมตริกซ์ A (แถวที่ (i-1)k+1-ik) และข้อมูลทั้งหมดของเมตริกซ์ B ให้แต่ละหน่วยประมวลผล i เมื่อ i= 1,2,…,p และ k = ⌈n/p⌉ โดยกรณีนี้ p=4 n=8 และ k=2

เมตริกซ์ A

16	6	0	13	12	12	1	3	หน่วยประมวลผลที่ 1 จำนวน 2 แถว (แถวที่ 1-2)
5	18	2	11	11	0	14	0	
1	1	10	15	17	15	7	12	หน่วยประมวลผลที่ 2 จำนวน 2 แถว (แถวที่ 3-4)
19	3	9	1	3	2	4	17	
17	11	7	2	7	17	9	8	หน่วยประมวลผลที่ 3 จำนวน 2 แถว (แถวที่ 5-6)
1	14	13	7	0	16	12	16	
5	6	5	14	15	1	13	1	หน่วยประมวลผลที่ 4 จำนวน 2 แถว (แถวที่ 7-8)
3	7	4	17	9	14	5	2	

เมตริกซ์ B

3	11	18	4	5	4	10	18	ทุกหน่วยประมวลผล
7	16	0	12	11	4	3	3	จำนวน 8 แถว 8 หลัก
3	9	12	8	15	11	14	4	
11	4	16	11	16	1	19	2	
18	12	11	9	4	6	11	6	
5	0	4	8	9	16	7	6	
13	14	12	10	18	17	4	8	
14	17	11	7	6	3	17	13	

ขั้นตอนที่ 2 หน่วยประมวลผลแต่ละหน่วยทำการอ่านข้อมูลบางส่วนตามแถวของเมตริกซ์ A และข้อมูลทั้งหมดของเมตริกซ์ B จากนั้นคำนวณหาผลคูณของเมตริกซ์ไปพร้อมๆ กัน

หน่วยประมวลผลที่ 1 (P_1) ทำการหาผลคูณของเมตริกซ์จำนวน 2 แถว คือ แถวที่ 1 และแถวที่ 2

เมตริกซ์ A							
16	6	0	13	12	12	1	3
5	18	2	11	11	0	14	0

เมตริกซ์ B							
3	11	18	4	5	4	10	18
7	16	0	12	11	4	3	3
3	9	12	8	15	11	14	4
11	4	16	11	16	1	19	2
18	12	11	9	4	6	11	6
5	0	4	8	9	16	7	6
13	14	12	10	18	17	4	8
14	17	11	7	6	3	17	13

=

เมตริกซ์ C							
564	533	721	514	546	391	696	523
648	733	579	612	725	429	518	352

เช่น C_{11} = 564 = (16x3)+(6x7)+(0x3)+(13x11)+(12x18)+(12x5) + (1x13) +(3x14)

หน่วยประมวลผลที่ 2 (P_2) ทำการหาผลคูณของเมตริกซ์จำนวน 2 แถว คือ แถวที่ 3 และแถวที่ 4

เมตริกซ์ A							
1	1	10	15	17	15	7	12
19	3	9	1	3	2	4	17

เมตริกซ์ B							
3	11	18	4	5	4	10	18
7	16	0	12	11	4	3	3
3	9	12	8	15	11	14	4
11	4	16	11	16	1	19	2
18	12	11	9	4	6	11	6
5	0	4	8	9	16	7	6
13	14	12	10	18	17	4	8
14	17	11	7	6	3	17	13

=

เมตริกซ์ C							
845	683	841	688	807	630	962	495
470	723	742	397	483	357	696	672

หน่วยประมวลผลที่ 3 (P_3) ทำการหาผลคูณของเมตริกซ์จำนวน 2 แถว คือ แถวที่ 5 และแถวที่ 6

เมตริกซ์ A							
17	11	7	2	7	17	9	8
1	14	13	7	0	16	12	16

เมตริกซ์ B							
3	11	18	4	5	4	10	18
7	16	0	12	11	4	3	3
3	9	12	8	15	11	14	4
11	4	16	11	16	1	19	2
18	12	11	9	4	6	11	6
5	0	4	8	9	16	7	6
13	14	12	10	18	17	4	8
14	17	11	7	6	3	17	13

=

เมตริกซ์ C							
611	780	763	623	734	682	707	691
677	820	670	713	922	718	799	526

หน่วยประมวลผลที่ 4 (P_4) ทำการหาผลคูณของเมตริกซ์จำนวน 2 แถว คือ แถวที่ 7 และแถวที่ 8

เมตริกซ์ A							
5	6	5	14	15	1	13	1
3	7	4	17	9	14	5	2

เมตริกซ์ B							
3	11	18	4	5	4	10	18
7	16	0	12	11	4	3	3
3	9	12	8	15	11	14	4
11	4	16	11	16	1	19	2
18	12	11	9	4	6	11	6
5	0	4	8	9	16	7	6
13	14	12	10	18	17	4	8
14	17	11	7	6	3	17	13

=

เมตริกซ์ C							
684	631	710	566	699	443	645	369
582	461	611	572	688	470	681	329

ขั้นตอนที่ 3 ส่งผลคูณของเมตริกซ์ (เมตริกซ์ C) คืนให้หน่วยประมวลผลหลัก

	เมตริกซ์ C (Worker)							
จากหน่วยประมวลผล ที่ 1 (P_1) จำนวน 2 แถว	564	533	721	514	546	391	696	523
	648	733	579	612	725	429	518	352
จากหน่วยประมวลผล ที่ 2 (P_2) จำนวน 2 แถว	845	683	841	688	807	630	962	495
	470	723	742	397	483	357	696	672
จากหน่วยประมวลผล ที่ 3 (P_3) จำนวน 2 แถว	611	780	763	623	734	682	707	691
	677	820	670	713	922	718	799	526
จากหน่วยประมวลผล ที่ 4 (P_4) จำนวน 2 แถว	684	631	710	566	699	443	645	369
	582	461	611	572	688	470	681	329

เมตริกซ์ C (Master)							
564	533	721	514	546	391	696	523
648	733	579	612	725	429	518	352
845	683	841	688	807	630	962	495
470	723	742	397	483	357	696	672
611	780	763	623	734	682	707	691
677	820	670	713	922	718	799	526
684	631	710	566	699	443	645	369
582	461	611	572	688	470	681	329

ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี (MPI-C like) โดยวิธี rbmm_w แสดงในรูปที่ 3.3

Master (Supervisor Node)

Start MPI

Create Data

Send Data to Workers

```
MPI_Send(&offset,1,MPI_INT,dest,mtype,MPI_COMM_WORLD);
MPI_Send(&rows,1,MPI_INT,dest,mtype,MPI_COMM_WORLD);
MPI_Send(&a, NRA/2*NCA, rows*NCA,MPI_DOUBLE,dest, mtype, MPI_COMM_WORLD);
MPI_Send(&b, NCA*NCB/2, MPI_DOUBLE, dest, mtype, PI_COMM_WORLD);
```

Wait for Result

```
MPI_Recv(&offset,1,MPI_INT,source,mtype,MPI_COMM_WORLD, &status);
MPI_Recv(&rows,1,MPI_INT,source,mtype,MPI_COMM_WORLD, &status);
MPI_Recv(&c[offset][0], rows*NCB, MPI_DOUBLE, source, mtype, MPI_COMM_WORLD,
&status);
```

Worker (Computing Node)

Receive Data from Master

```
MPI_Recv(&offset,1,MPI_INT,MASTER,mtype,MPI_COMM_WORLD, &status);
MPI_Recv(&rows,1,MPI_INT,MASTER,mtype,MPI_COMM_WORLD, &status);
MPI_Recv (&a, NRA/2*NCA, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,
&status);
MPI_Recv((&b, NCA*NCB/2, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,
&status);
```

Perform Matrix Multiplication

```
    for (k=0; k<NCB; k++)
      for (i=0; i<rows; i++) {
        c[i][k] = 0.0;
        for (j=0; j<NCA; j++)
          c[i][k] = c[i][k] + a[i][j] * b[j][k];     }
```

Send Data to Master

```
MPI_Send(&offset,1,MPI_INT,MASTER,mtype,MPI_COMM_WORLD);
MPI_Send(&rows,1,MPI_INT,MASTER,mtype,MPI_COMM_WORLD);
MPI_Send(&c, rows*NCB, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD);
```

รูปที่ 3.3 ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี (MPI-C like) วิธี rbmm_w

3.2 การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmm_wo

(Row-Block partitioning Matrix Multiplication Without replicated data)

วิธีการหาผลคูณเมตริกซ์แบบขนานวิธี rbmm_wo จะประกอบด้วยหน่วยประมวลผลซึ่งแบ่งเป็น 2 ชนิด คือ หน่วยประมวลผลหลัก (Master computer) จำนวน 1 หน่วย และหน่วยประมวลผลทำงาน (Worker computer) จำนวน P หน่วย

<u>หน่วยประมวลผลหลัก จำนวน 1 หน่วย</u>

ขั้นตอนที่ 1 สร้างข้อมูล เมตริกซ์ A และเมตริกซ์ B ซึ่งมีขนาดใหญ่มาก (nxn) เก็บลงในหน่วยความจำโดยใช้เทคนิคหน่วยความจำเสมือน (Virtual memory)

ขั้นตอนที่ 2 ทำการแบ่งข้อมูลตามแถวของเมตริกซ์ A ขนาด $k = \lceil n/p \rceil$ แถว และแบ่งข้อมูลตามหลักของเมตริกซ์ B ขนาด $r = \lceil n/p \rceil$ หลัก เพื่อให้ภาระงาน (Work load) มีขนาดเท่าๆ กัน (แสดงในรูปที่ 3.4) สำหรับหน่วยประมวลผลทำงาน P หน่วย เพื่อคำนวณหาผลคูณเมตริกซ์ C จำนวน $\lceil n/p \rceil$ แถวแบบขนาน

ขั้นตอนที่ 3 แจ้งช่วงและขนาดของเมตริกซ์ A เมตริกซ์ B ที่แต่ละหน่วยประมวลผลทำงานต้องทำการอ่านจากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก นำมาเก็บไว้ในหน่วยความจำของตัวเอง

ขั้นตอนที่ 4 รอรับผลคูณเมตริกซ์ C จากหน่วยประมวลผลทำงาน P หน่วย

<u>หมายเหตุ</u> สำหรับเมตริกซ์ A_{nxm} และเมตริกซ์ B_{mxn} ค่า $k = \lceil n/p \rceil$ และค่า $r = \lceil m/p \rceil$

<u>หน่วยประมวลผลทำงาน จำนวน P หน่วย</u>

ขั้นตอนที่ 1 รอรับการติดต่อจากหน่วยประมวลผลหลักและหลังจากนั้นอ่านข้อมูลบางส่วนของเมตริกซ์ A ขนาด $k = \lceil n/p \rceil$ แถว (แถวที่ (i-1)k+1 ถึงแถวที่ ik เมื่อ i คือ ไอดีของหน่วยประมวลผลทำงาน (Worker ID)) และบางส่วนของเมตริกซ์ B ขนาด $r = \lceil n/p \rceil$ หลัก (หลักที่ (i-1)r+1 ถึงหลักที่ ir) จากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก

ขั้นตอนที่ 2 ทำการคำนวณผลคูณเมตริกซ์ C ในแถวที่ (i-1)k+1 ถึงแถวที่ ik โดยมีรายละเอียดขั้นตอนย่อยดังนี้

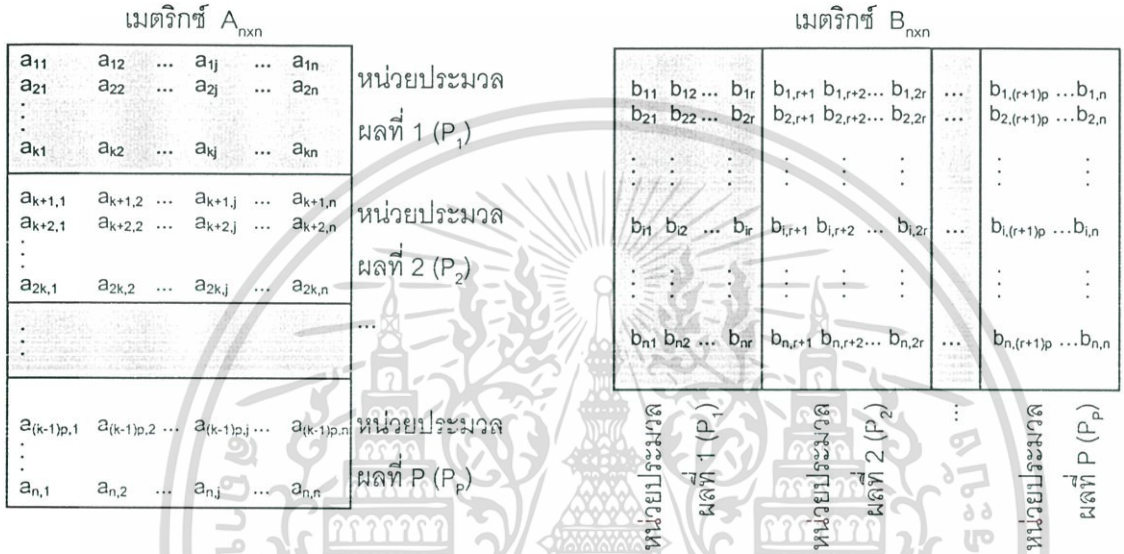
ขั้นตอนที่ 2.1 ทำการคำนวณผลคูณเมตริกซ์ C บางค่าในแถวที่ (i-1)k+1 ถึงแถวที่ ik และค่าในหลักที่ (i-1)r+1 ถึงหลักที่ ir (แสดงในรูปที่ 3.5)

ขั้นตอนที่ 2.2 ทำการส่งค่าเมตริกซ์ B หลักที่ (i-1)r+1 ถึงหลักที่ ir ให้หน่วยประมวลผลทำงานลำดับก่อนหน้า (P_{i-1}) และรับข้อมูลของเมตริกซ์ B หลักที่ ir+1 ถึงหลักที่ ir+r จากหน่วยประมวลผลทำงานลำดับถัดไป (P_{i+1}) เช่น กรณีที่มี P หน่วยประมวลผล P_i จะส่งค่าเมตริกซ์ B หลัก

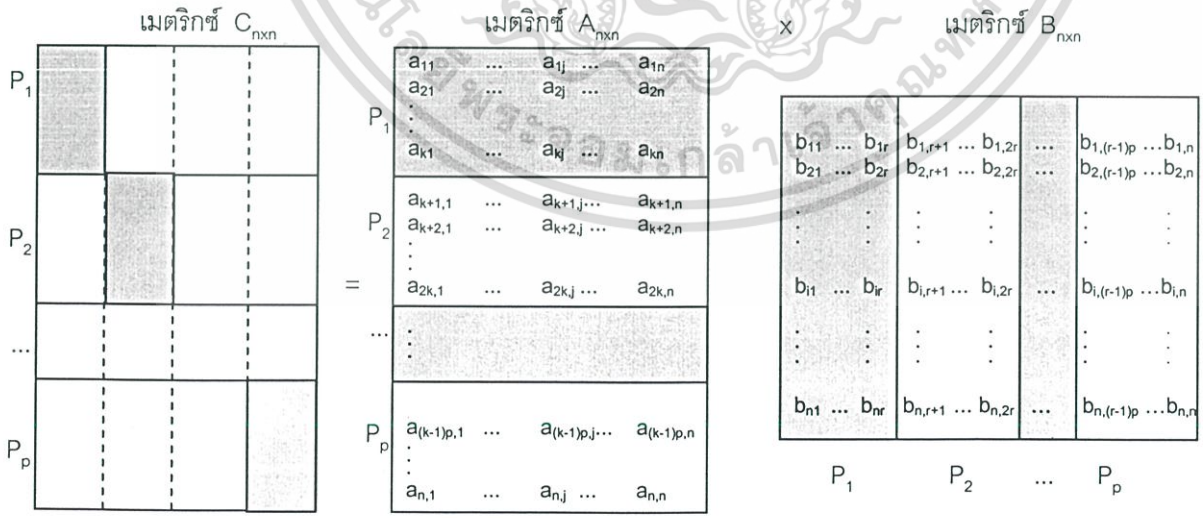
ที่ (i-1)r+1 ถึงหลักที่ ir ให้หน่วยประมวลผล P_{i-1} และรับหลักที่ ir +1 ถึงหลักที่ ir+r จากหน่วยประมวลผล P_{i+1} ยกเว้นหน่วยประมวลผล P_1 จะส่งหลักที่ 1 ถึงหลักที่ r ให้หน่วยประมวลผล P_p และหน่วยประมวลผล P_p ก็จะรับหลักที่ 1 ถึงหลักที่ r จากหน่วยประมวลผล P_1

ขั้นตอนที่ 2.3 ทำซ้ำขั้นตอนที่ 2.1 และขั้นตอนที่ 2.2 จนได้เมตริกซ์ C ครบทุกแถวที่ได้รับมอบภาระงานจากหน่วยประมวลผลหลัก

ขั้นตอนที่ 3 ส่งผลคูณของเมตริกซ์ C กลับไปยังหน่วยประมวลผลหลัก



รูปที่ 3.4 การแบ่งข้อมูลเมตริกซ์ A และ B สำหรับ P หน่วยประมวลผลของวิธี rbmm_wo



รูปที่ 3.5 การแบ่งงานหาผลคูณเมตริกซ์ C (บางส่วน) ที่มี P หน่วยประมวลผล ของวิธี rbmm_wo (ขั้นตอนที่ 2.1)

ตัวอย่างแสดงการแลกเปลี่ยนข้อมูลเมตริกซ์ B_{4x4} ที่ใช้ 4 หน่วยประมวลผลกระจายข้อมูลในแต่ละหลักของเมตริกซ์ B (ขั้นตอนที่ 2.2)

หน่วยประมวลผลที่ i (P_i) ส่งข้อมูลหลักที่ i ให้หน่วยประมวลที่ i-1 (P_{i-1})

หน่วยประมวลผลที่ i (P_i) รับข้อมูลหลักที่ i+1 จากหน่วยประมวลที่ i+1 (P_{i+1})

<u>รอบที่ 1</u> (แสดงในรูปที่ 3.6)

หน่วยประมวลผลที่ 2 (P_2) ส่งข้อมูลหลักที่ 2 ให้หน่วยประมวลที่ 1 (P_1)

หน่วยประมวลผลที่ 3 (P_3) ส่งข้อมูลหลักที่ 3 ให้หน่วยประมวลที่ 2 (P_2)

หน่วยประมวลผลที่ 4 (P_4) ส่งข้อมูลหลักที่ 4 ให้หน่วยประมวลที่ 3 (P_3)

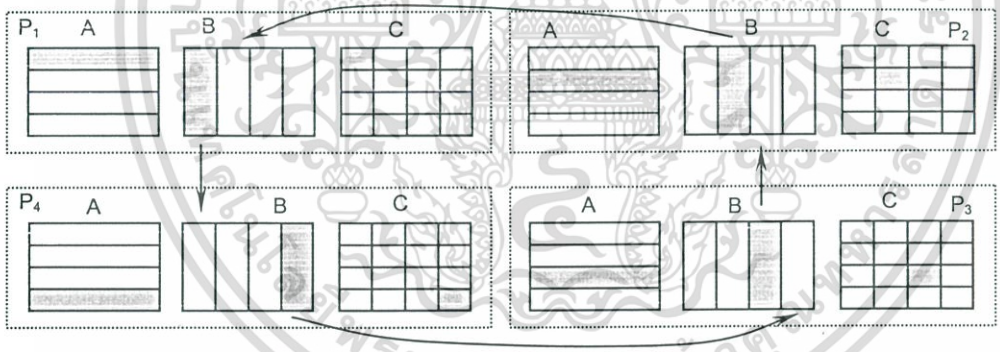
ยกเว้น หน่วยประมวลผลที่ 1 (P_1) ส่งข้อมูลหลักที่ 1 ให้หน่วยประมวลที่ 4 (P_4)

หน่วยประมวลผลที่ 1 (P_1) รับข้อมูลหลักที่ 2 จากหน่วยประมวลที่ 2 (P_2)

หน่วยประมวลผลที่ 2 (P_2) รับข้อมูลหลักที่ 3 จากหน่วยประมวลที่ 3 (P_3)

หน่วยประมวลผลที่ 3 (P_3) รับข้อมูลหลักที่ 4 จากหน่วยประมวลที่ 4 (P_4)

ยกเว้น หน่วยประมวลผลที่ 4 (P_4) รับข้อมูลหลักที่ 1 จากหน่วยประมวลที่ 1 (P_1)



รูปที่ 3.6 การแลกเปลี่ยนข้อมูลเมตริกซ์ B_{4x4} รอบที่ 1

<u>รอบที่ 2</u> (แสดงในรูปที่ 3.7)

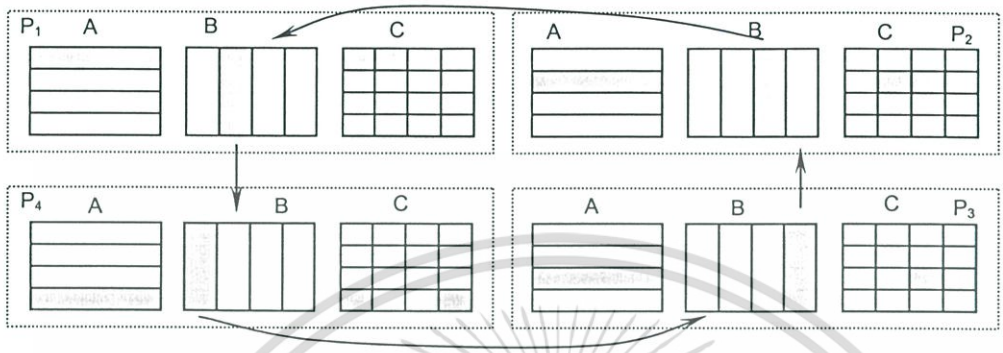
หน่วยประมวลผลที่ 2 (P_2) ส่งข้อมูลหลักที่ 3 ให้หน่วยประมวลที่ 1 (P_1)

หน่วยประมวลผลที่ 3 (P_3) ส่งข้อมูลหลักที่ 4 ให้หน่วยประมวลที่ 2 (P_2)

หน่วยประมวลผลที่ 4 (P_4) ส่งข้อมูลหลักที่ 1 ให้หน่วยประมวลที่ 3 (P_3)

ยกเว้น หน่วยประมวลผลที่ 1 (P_1) ส่งข้อมูลหลักที่ 2 ให้หน่วยประมวลที่ 4 (P_4)

หน่วยประมวลผลที่ 1 (P_1) รับข้อมูลหลักที่ 3 จากหน่วยประมวลที่ 2 (P_2)
หน่วยประมวลผลที่ 2 (P_2) รับข้อมูลหลักที่ 4 จากหน่วยประมวลที่ 3 (P_3)
หน่วยประมวลผลที่ 3 (P_3) รับข้อมูลหลักที่ 1 จากหน่วยประมวลที่ 4 (P_4)
ยกเว้น หน่วยประมวลผลที่ 4 (P_4) รับข้อมูลหลักที่ 2 จากหน่วยประมวลที่ 1 (P_1)

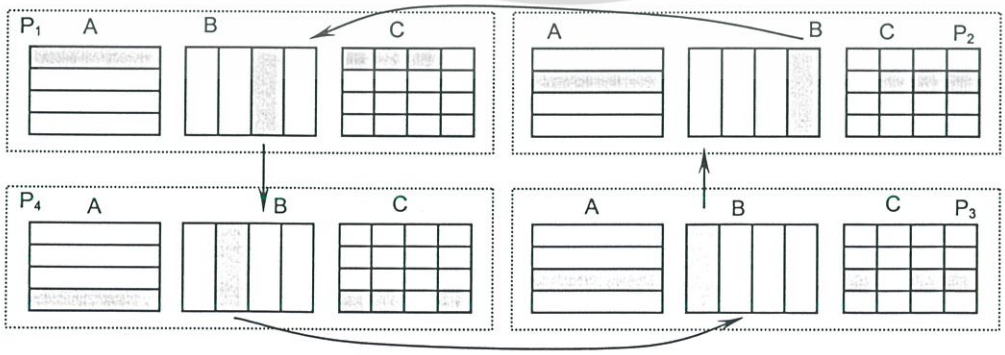


รูปที่ 3.7 การแลกเปลี่ยนข้อมูลเมตริกซ์ B_{4x4} รอบที่ 2

<u>รอบที่ 3</u> (แสดงในรูปที่ 3.8)

หน่วยประมวลผลที่ 2 (P_2) ส่งข้อมูลหลักที่ 4 ให้หน่วยประมวลที่ 1 (P_1)
หน่วยประมวลผลที่ 3 (P_3) ส่งข้อมูลหลักที่ 1 ให้หน่วยประมวลที่ 2 (P_2)
หน่วยประมวลผลที่ 4 (P_4) ส่งข้อมูลหลักที่ 2 ให้หน่วยประมวลที่ 3 (P_3)
ยกเว้น หน่วยประมวลผลที่ 1 (P_1) ส่งข้อมูลหลักที่ 3 ให้หน่วยประมวลที่ 4 (P_4)

หน่วยประมวลผลที่ 1 (P_1) รับข้อมูลหลักที่ 4 จากหน่วยประมวลที่ 2 (P_2)
หน่วยประมวลผลที่ 2 (P_2) รับข้อมูลหลักที่ 1 จากหน่วยประมวลที่ 3 (P_3)
หน่วยประมวลผลที่ 3 (P_3) รับข้อมูลหลักที่ 2 จากหน่วยประมวลที่ 4 (P_4)
ยกเว้น หน่วยประมวลผลที่ 4 (P_4) รับข้อมูลหลักที่ 3 จากหน่วยประมวลที่ 1 (P_1)



รูปที่ 3.8 การแลกเปลี่ยนข้อมูลเมตริกซ์ B_{4x4} รอบที่ 3

ตัวอย่างที่3.3 การหาผลคูณของเมตริกซ์ระหว่างเมตริกซ์ A_{8x8} และ เมตริกซ์ B_{8x8} วิธี rbmm_woโดยใช้จำนวนหน่วยประมวลผล 4 หน่วยประมวลผล

เมตริกซ์ A

16	6	0	13	12	12	1	3
5	18	2	11	11	0	14	0
1	1	10	15	17	15	7	12
19	3	9	1	3	2	4	17
17	11	7	2	7	17	9	8
1	14	13	7	0	16	12	16
5	6	5	14	15	1	13	1
3	7	4	17	9	14	5	2

เมตริกซ์ B

3	11	18	4	5	4	10	18
7	16	0	12	11	4	3	3
3	9	12	8	15	11	14	4
11	4	16	11	16	1	19	2
18	12	11	9	4	6	11	6
5	0	4	8	9	16	7	6
13	14	12	10	18	17	4	8
14	17	11	7	6	3	17	13

วิธีทำ

ขั้นตอนที่ 1 หน่วยประมวลผลหลักแบ่งข้อมูลตามแถวของเมตริกซ์ A และตามหลักของเมตริกซ์ B ให้แต่ละหน่วยประมวลผลโดยข้อมูลไม่ซ้ำกัน

เมตริกซ์ A

16	6	0	13	12	12	1	3	หน่วยประมวลผลที่ 1 จำนวน 2 แถว
5	18	2	11	11	0	14	0	
1	1	10	15	17	15	7	12	หน่วยประมวลผลที่ 2 จำนวน 2 แถว
19	3	9	1	3	2	4	17	
17	11	7	2	7	17	9	8	หน่วยประมวลผลที่ 3 จำนวน 2 แถว
1	14	13	7	0	16	12	16	
5	6	5	14	15	1	13	1	หน่วยประมวลผลที่ 4 จำนวน 2 แถว
3	7	4	17	9	14	5	2	

เมตริกซ์ B

3	11	18	4	5	4	10	18
7	16	0	12	11	4	3	3
3	9	12	8	15	11	14	4
11	4	16	11	16	1	19	2
18	12	11	9	4	6	11	6
5	0	4	8	9	16	7	6
13	14	12	10	18	17	4	8
14	17	11	7	6	3	17	13

หน่วยประมวลผลที่ 4 จำนวน 2 หลัก

หน่วยประมวลผลที่ 3 จำนวน 2 หลัก

หน่วยประมวลผลที่ 2 จำนวน 2 หลัก

หน่วยประมวลผลที่ 1 จำนวน 2 หลัก

ขั้นตอนที่ 2 หน่วยประมวลผลแต่ละหน่วยทำการคำนวณหาผลคูณของเมตริกซ์ โดยมีรายละเอียดขั้นตอนย่อยดังนี้

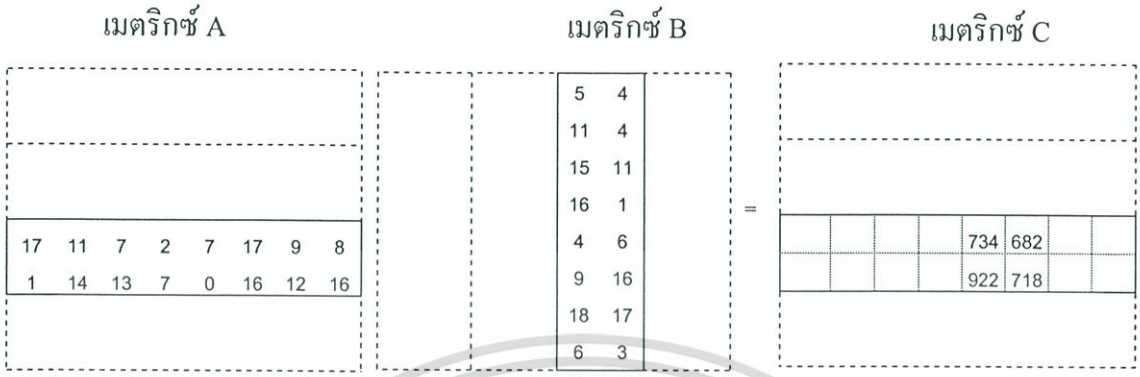
ขั้นตอนที่ 2.1 ทำการคำนวณผลคูณเมตริกซ์ C บางค่าในแถวที่ (i-1)k+1 ถึงแถวที่ ik และค่าในหลักที่ (i-1)r+1 ถึงหลักที่ ir

หน่วยประมวลผลที่ 1 (P_1) ทำการหาผลคูณของเมตริกซ์(เมตริกซ์ C) ขนาด 2x2 คือ บางส่วนแถวที่ 1 และบางส่วนแถวที่ 2

เช่น C_{11}= 564 = (16x3)+(6x7)+(0x3)+(13x11)+(12x18)+(12x5) + (1x13) +(3x14)

หน่วยประมวลผลที่ 2 (P_2) ทำการหาผลคูณของเมตริกซ์(เมตริกซ์ C) ขนาด 2x2 คือ บางส่วนแถวที่ 3 และบางส่วนแถวที่ 4

หน่วยประมวลผลที่ 3 (P_3) ทำการหาผลคูณของเมตริกซ์(เมตริกซ์ C) ขนาด 2x2 คือ บางส่วนแถวที่ 5 และบางส่วนแถวที่ 6



หน่วยประมวลผลที่ 4 (P_4) ทำการหาผลคูณของเมตริกซ์(เมตริกซ์ C) ขนาด 2x2 คือ บางส่วนแถวที่ 7 และบางส่วนแถวที่ 8

ขั้นตอนที่ 2.2 ส่งข้อมูลเมตริกซ์ B ให้หน่วยประมวลผลถัดไป(P_{i+1}) และรับข้อมูลเมตริกซ์ B จากหน่วยประมวลผลก่อนหน้า (P_{i-1}) ดังนี้

หน่วยประมวลผลที่ 1 ส่งให้หน่วยประมวลผลที่ 4 และรับจากหน่วยประมวลผลที่ 2
หน่วยประมวลผลที่ 2 ส่งให้หน่วยประมวลผลที่ 1 และรับจากหน่วยประมวลผลที่ 3
หน่วยประมวลผลที่ 3 ส่งให้หน่วยประมวลผลที่ 2 และรับจากหน่วยประมวลผลที่ 4
หน่วยประมวลผลที่ 4 ส่งให้หน่วยประมวลผลที่ 3 และรับจากหน่วยประมวลผลที่ 1

ขั้นตอนที่ 2.3 ทำซ้ำขั้นตอนที่ 2.1 และขั้นตอนที่ 2.2 จนได้เมตริกซ์ C ครบทุกแถวที่ได้รับมอบภาระงานจากหน่วยประมวลผลหลัก

ขั้นตอนที่ 3 ส่งผลคูณของเมตริกซ์ (เมตริกซ์ C) คืนให้หน่วยประมวลผลหลัก

ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี (MPI-C like) โดยวิธี rbmm_wo แสดงในรูปที่ 3.9

Master (Supervisor Node)

Start MPI

Create Data

Send Data to Workers

```
MPI_Send(&offset,1,MPI_INT,dest,mtype,MPI_COMM_WORLD);
MPI_Send(&rows,1,MPI_INT,dest,mtype,MPI_COMM_WORLD);
MPI_Send(&a, rows*nrow_a, rows*NCA,MPI_DOUBLE,dest, mtype,
MPI_COMM_WORLD);
MPI_Send(&b, rows*nrow_a, MPI_DOUBLE, dest, mtype, PI_COMM_WORLD);
```

Wait for Result

```
MPI_Recv(&offset,1,MPI_INT,source,mtype,MPI_COMM_WORLD, &status);
MPI_Recv(&rows,1,MPI_INT,source,mtype,MPI_COMM_WORLD, &status);
MPI_Recv(&c[offset][0], rows*nrow_a, MPI_DOUBLE, source, mtype,
MPI_COMM_WORLD, &status);
```

Worker (Computing Node)

Receive Data from Master

```
MPI_Recv(&offset,1,MPI_INT,MASTER,mtype,MPI_COMM_WORLD, &status);
MPI_Recv(&rows,1,MPI_INT,MASTER,mtype,MPI_COMM_WORLD, &status);
MPI_Recv (&a, rows*nrow_a, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,
&status);
MPI_Recv((&b, rows*nrow_a, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,
&status);
```

Perform Matrix Multiplication

```
  for (i=0; i<rows; i++)
          for (k=0; k<rows; k++)
          {
         cworker[i][k+offset_a] = 0.0;
            for (j=0; j<nrow_a; j++)
               cworker[i][k+offset_a] = cworker[i][k+offset_a] + a[i][j] * b[k][j];
}
```

Send Data to Master

```
MPI_Send(&offset,1,MPI_INT,MASTER,mtype,MPI_COMM_WORLD);
MPI_Send(&rows,1,MPI_INT,MASTER,mtype,MPI_COMM_WORLD);
MPI_Send(&c, rows*nrow_a, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD);
```

รูปที่ 3.9 ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี (MPI-C like) วิธี rbmm_wo

3.3 การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี cbmm_w

(Checkerboard-Block partitioning Matrix Multiplication With replicated data)

วิธีการหาผลคูณเมตริกซ์แบบขนานวิธี cbmm_w จะประกอบด้วยหน่วยประมวลผลซึ่งแบ่งเป็น 2 ชนิด คือ หน่วยประมวลผลหลัก (Master computer) จำนวน 1 หน่วย และหน่วยประมวลผลทำงาน (Worker computer) จำนวน P หน่วย โดยที่ P = 1,2,3,4,5,6, .. , P_rxP_c โดยการแบ่งข้อมูลในแต่ละหน่วยประมวลผลจะขึ้นอยู่กับภาระงาน (Work load) ที่ได้รับผิดชอบ เช่น หน่วยประมวลผลทำงานมี P_rxP_c หน่วยประมวลผล ก็จะแบ่งข้อมูลตามแถวของเมตริกซ์ A ขนาด $r = \lceil n/P_r \rceil$ หรือ $\sqrt{p}\lceil n/p \rceil$ หรือ $\lceil n/\sqrt{p} \rceil$ เมื่อ $P_r=P_c$ และแบ่งข้อมูลตามหลักของเมตริกซ์ B ขนาด $c = \lceil n/P_c \rceil$ หรือ $\sqrt{p}\lceil n/p \rceil$ หรือ $\lceil n/\sqrt{p} \rceil$ เมื่อ $P_r=P_c$

หมายเหตุ สำหรับเมตริกซ์ A_{nxm} และเมตริกซ์ B_{mxn} ค่า $r = \lceil n/P_r \rceil$ และค่า $c = \lceil m/P_c \rceil$

หน่วยประมวลผลหลัก จำนวน 1 หน่วย

ขั้นตอนที่ 1 สร้างข้อมูล คือ เมตริกซ์ A และเมตริกซ์ B ซึ่งมีขนาดใหญ่มาก (nxn) เก็บลงในหน่วยความจำโดยใช้เป็นเทคนิคหน่วยความจำเสมือน (Virtual memory)

ขั้นตอนที่ 2 ทำการแบ่งข้อมูลตามแถวของเมตริกซ์ A ขนาด $r = \sqrt{p}\lceil n/p \rceil$ แถว และแบ่งข้อมูลตามหลักของเมตริกซ์ B ขนาด $c = \sqrt{p}\lceil n/p \rceil$ หลัก เพื่อให้ภาระงานมีขนาดเท่าๆ กัน (แสดงในรูปที่ 3.10) สำหรับหน่วยประมวลผลทำงาน P หน่วย เพื่อคำนวณหาผลคูณเมตริกซ์ C ขนาด $n/\sqrt{p}$ x $n/\sqrt{p}$ แบบขนาน

ขั้นตอนที่ 3 แจ้งช่วงและขนาดของเมตริกซ์ A เมตริกซ์ B ที่แต่ละหน่วยประมวลผลทำงานต้องทำการอ่านจากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก นำมาเก็บไว้ในหน่วยความจำของตัวเอง

ขั้นตอนที่ 4 รอรับผลคูณเมตริกซ์ C จากหน่วยประมวลผลทำงาน P หน่วย

หน่วยประมวลผลทำงาน จำนวน P หน่วย

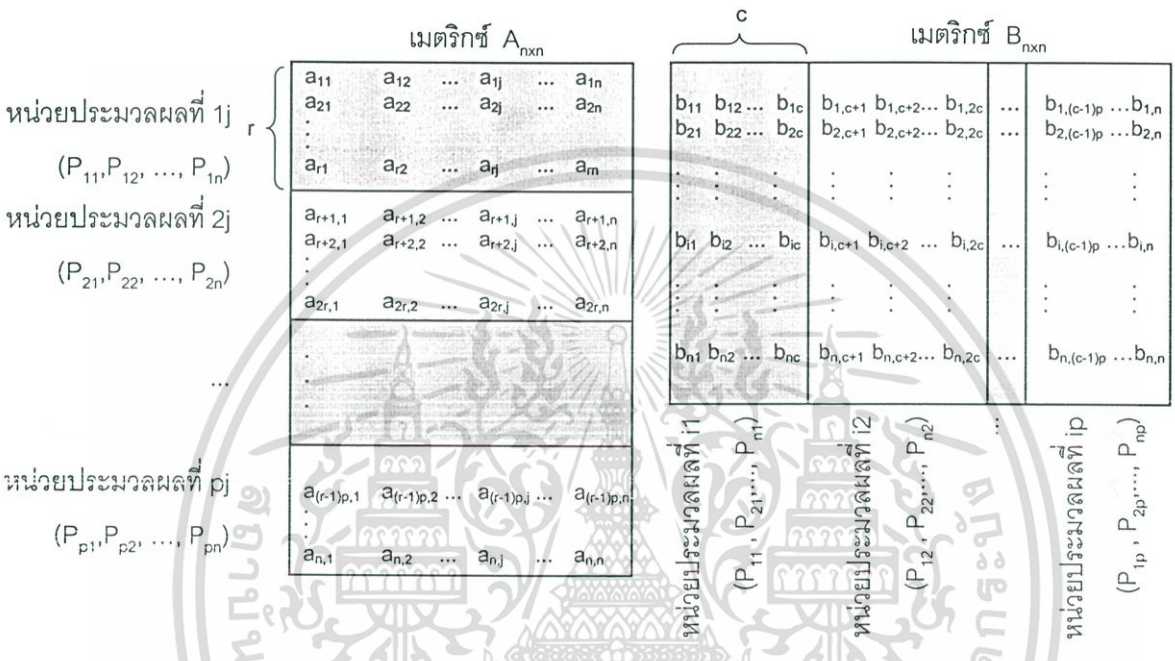
ขั้นตอนที่ 1 รอรับการติดต่อจากหน่วยประมวลผลหลักและหลังจากนั้นอ่านข้อมูลบางส่วนของเมตริกซ์ A ขนาด $r = \sqrt{p}\lceil n/p \rceil$ และบางส่วนของเมตริกซ์ B ขนาด $c = \sqrt{p}\lceil n/p \rceil$ หลักจากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก

หมายเหตุ - หน่วยประมวลผลที่ 1j (P_{1j}) เมื่อ j=1,2,…,n ใช้ข้อมูลเมตริกซ์ A แถวที่ (j-1)r+1 ถึงแถวที่ jr ซ้ำกันและหน่วยประมวลผลที่ 2j,3j,…, pj จะใช้ข้อมูลเมตริกซ์ A แถวถัดไป (แสดงในรูปที่ 3.10)

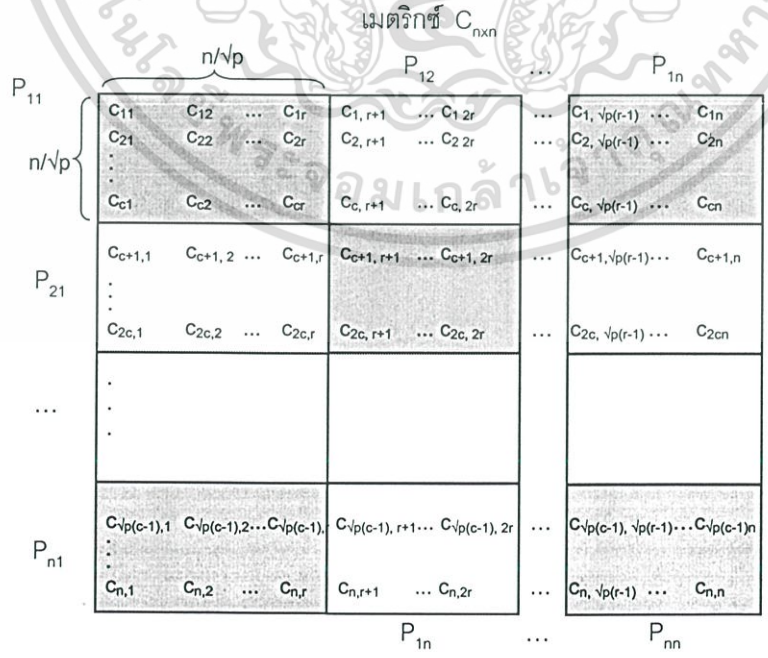
- หน่วยประมวลผลที่ i1 (P_{i1}) เมื่อ i=1,2,…,n ใช้ข้อมูลเมตริกซ์ B หลักที่ (i-1)c+1 ถึงหลักที่ ic ซ้ำกันและหน่วยประมวลผลที่ i2,i3,…, ip จะใช้ข้อมูลเมตริกซ B หลักถัดไป (แสดงในรูปที่ 3.10)

ขั้นตอนที่ 2 หน่วยประมวลผล P_{ij} ทำการคำนวณผลคูณเมตริกซ์ C แบบตารางย่อยแถวที่ (i-1)r+1 ถึงแถวที่ ir และหลักที่ (j-1)c+1 ถึงหลักที่ jc

ขั้นตอนที่ 3 ส่งผลคูณของเมตริกซ์ C กลับไปยังหน่วยประมวลผลหลัก(แสดงในรูปที่ 3.11)



รูปที่ 3.10 การแบ่งข้อมูลเมตริกซ์ A และ B สำหรับ P หน่วยประมวลผลของวิธี cbmm_w



รูปที่ 3.11 การแบ่งงานหาผลคูณเมตริกซ์ C ที่มี P หน่วยประมวลผลวิธี cbmm_w

ตัวอย่างที่3.2 การหาผลคูณของเมตริกซ์ระหว่างเมตริกซ์ A_{8x8} และ เมตริกซ์ B_{8x8} วิธี cbmm_wโดยใช้จำนวนหน่วยประมวลผล 4 หน่วยประมวลผล

เมตริกซ์ A

16	6	0	13	12	12	1	3
5	18	2	11	11	0	14	0
1	1	10	15	17	15	7	12
19	3	9	1	3	2	4	17
17	11	7	2	7	17	9	8
1	14	13	7	0	16	12	16
5	6	5	14	15	1	13	1
3	7	4	17	9	14	5	2

เมตริกซ์ B

3	11	18	4	5	4	10	18
7	16	0	12	11	4	3	3
3	9	12	8	15	11	14	4
11	4	16	11	16	1	19	2
18	12	11	9	4	6	11	6
5	0	4	8	9	16	7	6
13	14	12	10	18	17	4	8
14	17	11	7	6	3	17	13

<u>วิธีทำ</u>

ขั้นตอนที่ 1 หน่วยประมวลผลหลักแบ่งข้อมูลตามแถวของเมตริกซ์ A และตามหลักเมตริกซ์ B ให้แต่ละหน่วยประมวลผล

เมตริกซ์ A

16	6	0	13	12	12	1	3
5	18	2	11	11	0	14	0
1	1	10	15	17	15	7	12
19	3	9	1	3	2	4	17
17	11	7	2	7	17	9	8
1	14	13	7	0	16	12	16
5	6	5	14	15	1	13	1
3	7	4	17	9	14	5	2

หน่วยประมวลผลที่ 1 จำนวน 4 แถว
หน่วยประมวลผลที่ 2 จำนวน 4 แถว
หน่วยประมวลผลที่ 3 จำนวน 4 แถว
หน่วยประมวลผลที่ 4 จำนวน 4 แถว

เมตริกซ์ B

3	11	18	4	5	4	10	18
7	16	0	12	11	4	3	3
3	9	12	8	15	11	14	4
11	4	16	11	16	1	19	2
18	12	11	9	4	6	11	6
5	0	4	8	9	16	7	6
13	14	12	10	18	17	4	8
14	17	11	7	6	3	17	13

หน่วยประมวลผลที่ 4 จำนวน 4 หลัก
หน่วยประมวลผลที่ 2 จำนวน 4 หลัก
หน่วยประมวลผลที่ 3 จำนวน 4 หลัก
หน่วยประมวลผลที่ 1 จำนวน 4 หลัก

ขั้นตอนที่ 2 หน่วยประมวลผลแต่ละหน่วยทำการคำนวณหาผลคูณของเมตริกซ์

หน่วยประมวลผลที่ 1 (P_{11}) ทำการหาผลคูณของเมตริกซ์(เมตริกซ์ C) ขนาด 4x4

เมตริกซ์ A							
16	6	0	13	12	12	1	3
5	18	2	11	11	0	14	0
1	1	10	15	17	15	7	12
19	3	9	1	3	2	4	17

เมตริกซ์ B			
3	11	18	4
7	16	0	12
3	9	12	8
11	4	16	11
18	12	11	9
5	0	4	8
13	14	12	10
14	17	11	7

=

เมตริกซ์ C			
564	533	721	514
648	733	579	612
845	683	841	688
470	723	742	397

เช่น C_{11}= 564 = (16x3)+(6x7)+(0x3)+(13x11)+(12x18)+(12x5) + (1x13) +(3x14)

หน่วยประมวลผลที่ 2 (P_{12}) ทำการหาผลคูณของเมตริกซ์(เมตริกซ์ C) ขนาด 4x4

เมตริกซ์ A							
16	6	0	13	12	12	1	3
5	18	2	11	11	0	14	0
1	1	10	15	17	15	7	12
19	3	9	1	3	2	4	17

เมตริกซ์ B			
5	4	10	18
11	4	3	3
15	11	14	4
16	1	19	2
4	6	11	6
9	16	7	6
18	17	4	8
6	3	17	13

=

เมตริกซ์ C			
546	391	696	523
725	429	518	352
807	630	962	495
483	357	696	672

หน่วยประมวลผลที่ 3 (P_{21}) ทำการหาผลคูณของเมตริกซ์(เมตริกซ์ C) ขนาด 4x4

เมตริกซ์ A							
17	11	7	2	7	17	9	8
1	14	13	7	0	16	12	16
5	6	5	14	15	1	13	1
3	7	4	17	9	14	5	2

เมตริกซ์ B			
5	4	10	18
11	4	3	3
15	11	14	4
16	1	19	2
4	6	11	6
9	16	7	6
18	17	4	8
6	3	17	13

=

เมตริกซ์ C			
734	682	707	691
922	718	799	526
699	443	645	369
688	470	681	329

หน่วยประมวลผลที่ 4 (P_{22}) ทำการหาผลคูณของเมตริกซ์(เมตริกซ์ C) ขนาด 4x4

เมตริกซ์ A

17	11	7	2	7	17	9	8
1	14	13	7	0	16	12	16
5	6	5	14	15	1	13	1
3	7	4	17	9	14	5	2

เมตริกซ์ B

3	11	18	4
7	16	0	12
3	9	12	8
11	4	16	11
18	12	11	9
5	0	4	8
13	14	12	10
14	17	11	7

เมตริกซ์ C

=

611	780	763	623
677	820	670	713
684	631	710	566
582	461	611	572

<u>ขั้นตอนที่ 3</u> ส่งผลคูณของเมตริกซ์ (เมตริกซ์ C) คืนให้หน่วยประมวลผลหลัก

<u>หมายเหตุ</u> สำหรับโปรแกรมการหาผลคูณของเมตริกซ์วิธี cbmm_w ได้กำหนดให้เมตริกซ์ B อยู่ในรูปที่ผ่านการทรานสโพสมาแล้ว เพื่อง่ายในการส่งข้อมูลตามมาตรฐานภาษาเอ็มพีไอ (MPI) ตัวอย่างแสดงดังรูป

เมตริกซ์ B

3	11	18	4	5	4	10	18
7	16	0	12	11	4	3	3
3	9	12	8	15	11	14	4
11	4	16	11	16	1	19	2
18	12	11	9	4	6	11	6
5	0	4	8	9	16	7	6
13	14	12	10	18	17	4	8
14	17	11	7	6	3	17	13

เมตริกซ์ B ทรานสโพส

3	7	3	11	18	5	13	14
11	16	9	4	12	0	14	17
18	0	12	16	11	4	12	11
4	12	8	11	9	8	10	7
5	11	15	16	4	9	18	6
4	4	11	1	6	16	17	3
10	3	14	19	11	7	4	17
18	3	4	2	6	6	8	13

ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี (MPI-C like) โดยวิธี cbmm_w แสดงในรูปที่ 3.12

Master (Supervisor Node)

Start MPI

Create Data

Send Data to Workers

```
MPI_Send(&offset_a, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
MPI_Send(&rows, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
MPI_Send((void *)a[offset_a],rows*nrow_a, MPI_DOUBLE, dest,
mtype,MPI_COMM_WORLD);
MPI_Send((void *)bt[offset_b],rows*nrow_a, MPI_DOUBLE, dest,
mtype,MPI_COMM_WORLD);
```

Wait for Result

```
for (i=1; i<=numworkers; i++) {
    source = i;
    MPI_Recv(&offset,1,MPI_INT,source,mtype,MPI_COMM_WORLD, &status);
    MPI_Recv(&rows,1,MPI_INT,source,mtype,MPI_COMM_WORLD, &status);
    MPI_Recv(&c[0][0], NRA*NCB, MPI_DOUBLE, source, mtype,
    MPI_COMM_WORLD, &status);
}
```

Worker (Computing Node)

Receive Data from Master

```
mtype=FROM_MASTER;
MPI_Recv(&offset_a, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &status);
MPI_Recv(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &status);
MPI_Recv(*a, rows*nrow_a, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,
&status);
MPI_Recv(*bt, rows*nrow_a, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,
&status);
```

Perform Matrix Multiplication

```
for (i=0; i<rows; i++)
  for (k=0; k<rows; k++)
  {
          cworker[i][k] = 0.0;
          for (j=0; j<nrow_a; j++) {
             cworker[i][k] = cworker[i][k] + a[i][j] * bt[k][j];
          }
  }
```

Send Data to Master

```
MPI_Send(&offset,1,MPI_INT,MASTER,mtype,MPI_COMM_WORLD);
MPI_Send(&rows,1,MPI_INT,MASTER,mtype,MPI_COM_WORLD);
MPI_Send(*cworker, rows*nrow_a/2, MPI_DOUBLE, MASTER, mtype,
MPI_COMM_WORLD);
```

รูปที่ 3.12 ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี (MPI-C like) วิธี cbmm_w

บทที่ 4

การทดลองและผลการทดลอง

วิทยานิพนธ์นี้เป็นการนำเสนอการออกแบบขั้นตอนวิธีการหาผลคูณของเมตริกซ์แบบขนานเพื่อประมวลผลบนระบบพีซีคลัสเตอร์ โดยวิธีต่างๆ ที่เสนอเป็นวิธีที่ปรับปรุงประสิทธิภาพทั้งในด้านเวลาที่ใช้ในการประมวลผล รวมทั้งเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล และการใช้ข้อมูลที่ซ้ำซ้อนกันในแต่ละหน่วยประมวลผล ดังแสดงไว้แล้วในบทที่ 3 ส่วนเนื้อหาในบทนี้ จะเป็นการเสนอผลการพัฒนาโปรแกรมการหาผลคูณของเมตริกซ์แบบขนานดังกล่าว เพื่อประมวลผลบนระบบพีซีคลัสเตอร์ ซึ่งเป็นระบบที่ได้รับความสนใจเป็นอย่างมากในปัจจุบัน เนื่องจากมีราคาไม่แพง และมีขั้นตอนการพัฒนาที่ไม่ยุ่งยากซับซ้อนมากจนเกินไป ปกติการประมวลผลแบบขนานจะมีความซับซ้อนกว่าการประมวลผลแบบอนุกรม (Sequential Processing) เนื่องจากเวลาที่ใช้ในการประมวลผลแบบขนานทั้งหมด จะประกอบด้วยเวลาที่ใช้ในการประมวลผล (Computation Time) โดยใช้ P หน่วยประมวลผล และเวลาที่ใช้ในการติดต่อสื่อสาร (Communication Time) ระหว่างหน่วยประมวลผล ส่วนเวลาทั้งหมดที่ใช้ในการประมวลผลแบบอนุกรมจะเป็นเวลาที่ใช้ในการประมวลผลโดยใช้หนึ่งหน่วยประมวลผลเท่านั้น ถึงแม้ว่าการประมวลผลแบบขนานจะมีความซับซ้อนกว่าการประมวลผลทั่วๆ ไปที่เป็นแบบอนุกรม แต่การประมวลผลแบบขนานมีบทบาทสำคัญต่อการประมวลผลงานขนาดใหญ่ที่ใช้เวลานาน ดังนั้นวิธีการประมวลผลแบบขนานที่มีประสิทธิภาพจึงถูกเสนอขึ้น ทั้งขั้นตอนวิธีแบบขนานและการประยุกต์ใช้งานในหลายๆ ด้านแบบขนาน

4.1 ระบบคอมพิวเตอร์คลัสเตอร์ที่ใช้ในการทดลอง

ระบบคลัสเตอร์ที่ใช้ในการทดลอง เป็นระบบคลัสเตอร์แบบ “Homogenous Cluster” ที่มีส่วนประกอบ ประสิทธิภาพในการประมวลผล และภาระงานที่รับผิดชอบในแต่ละเครื่องมีจำนวนเท่ากัน นอกจากนั้นในขณะทำการทดลองระบบคลัสเตอร์จะไม่มีภาระงานอื่นใดประมวลผลอยู่ในระบบคลัสเตอร์ที่ใช้ในการทดลอง มีส่วนประกอบพื้นฐานของระบบซึ่งจะประกอบด้วยส่วนประกอบด้านฮาร์ดแวร์ ระบบปฏิบัติการและซอฟท์แวร์ ที่ทำงานในระดับของแกน (Kernel) ของระบบปฏิบัติการซึ่งเรียกซอฟท์แวร์นี้ว่า “คลัสเตอร์มิดเดิลแวร์” และสุดท้ายคือการเชื่อมต่อแต่ละโหนดข้อมูลเข้าด้วยกันผ่านทางเครือข่ายความเร็วสูง[20]

1) ส่วนประกอบด้านฮาร์ดแวร์

โครงสร้างทางฮาร์ดแวร์ของระบบที่ใช้ในการทดลอง เป็นคลัสเตอร์ที่มีโครงสร้างเหมือนกัน (Homogenous Cluster) คือหน่วยประมวลผล ขนาดของหน่วยความจำ และส่วนประกอบอื่นๆ เหมือนกัน ระบบที่ใช้ในการทดลองเป็นระบบคลัสเตอร์ที่พัฒนาขึ้นโดยสำนักวิจัยและบริการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง โดยระบบในปัจจุบันนี้ ประกอบด้วยคอมพิวเตอร์ส่วนบุคคลจำนวน 5 เครื่อง โดยใช้เครื่องคอมพิวเตอร์ชนิดที่มีสองหน่วยประมวลผลในเครื่องเดียวกัน (Dual CPU) ซึ่งมีองค์ประกอบด้านฮาร์ดแวร์ที่จำเป็นในการพิจารณาเพื่อสร้างเป็นระบบคลัสเตอร์ต่างๆ ดังนี้

ก) แผงวงจรรวม (Mother Board) หรือ System Board

ข) หน่วยประมวลผล (CPUs) ในงานวิจัยนี้ใช้เครื่องคอมพิวเตอร์ชนิดที่มีสองหน่วยประมวลผลในเครื่องเดียวกัน(Dual CPUs) ยี่ห้ออินเทล (Intel Xeon) ความเร็วต่อรอบ 2.4 กิกะเฮิร์ต

ค) หน่วยความจำหลัก (RAM) ใช้ 1 กิกะไบต์

ง) หน่วยความจำสำรอง (Disk Storage) มีขนาดเท่ากับ 100 กิกะไบต์

จ) หน่วยความจำแคช (Cache Memory) ขนาด 512 kb

ฉ) เชื่อมต่อผ่านเครือข่ายความเร็ว 100 เมกกะบิตต่อวินาที (Mbps)

2) ระบบปฏิบัติการ

ระบบปฏิบัติการจะเป็นส่วนสำคัญอย่างมากต่อการทำงานของคอมพิวเตอร์และระบบคลัสเตอร์เพราะคอมพิวเตอร์ในระบบคลัสเตอร์ต้องสามารถทำงานเองได้โดยอิสระไม่ขึ้นกับเงื่อนไขของเครื่องอื่นถึงแม้มีเครื่องใดเครื่องหนึ่งในระบบหยุดทำงาน ระบบคลัสเตอร์ก็ยังสามารถทำงานได้ดังนั้นระบบปฏิบัติการจึงมีความจำเป็นในส่วนที่จะทำให้คอมพิวเตอร์แต่ละเครื่องสามารถทำงานโดยอิสระต่อกันได้ และสามารถติดต่อสื่อสารกันได้ ระบบปฏิบัติการที่สามารถใช้กับระบบคลัสเตอร์แบบพีซี (Cluster of PCs) มีหลากหลาย เช่น Linux, SUSE, FreeBSD, HP-UX และระบบปฏิบัติการที่สามารถใช้กับระบบคลัสเตอร์แบบเวิร์กสเตชั่น (Cluster of Workstation) คือ Tru64 UNIX, Solaris เป็นต้น ซึ่งการเลือกใช้ระบบปฏิบัติการนั้นขึ้นอยู่กับระบบคลัสเตอร์ด้วยว่าเป็นแบบใด (PCs or Workstation) นอกจากนี้ยังขึ้นอยู่กับความสะดวก ความเชี่ยวชาญและอุปกรณ์ที่เลือกใช้ อีกอย่างหนึ่งคือจะขึ้นอยู่กับซอฟท์แวร์และชุดคำสั่งที่เลือกใช้ด้วย เช่น ถ้าเลือกใช้ OpenMosix ที่ทำงานได้บนระบบลีนุกซ์เท่านั้น ก็จำเป็นต้องเลือกใช้ลีนุกต์เป็นระบบปฏิบัติการอีก

ทั้งฮาร์ดแวร์ที่สนับสนุนด้วย โดยส่วนใหญ่ซอฟท์แวร์และชุดคำสั่งที่ถูกพัฒนาขึ้นมาเพื่อใช้ในระบบคลัสเตอร์นี้จะทำงานเข้ากันได้กับระบบปฏิบัติการตระกูลยูนิกซ์เกือบทุกชนิดอยู่แล้ว

ระบบคลัสเตอร์ที่ใช้ในงานวิจัยนี้เป็นระบบปฏิบัติลีนุกต์รุ่นซูซี (SUSE Version) ที่ใช้บนระบบคลัสเตอร์แบบพีซีเนื่องจากมีความสามารถและรองรับการทำงานได้หลากหลาย อีกทั้งยังมีความสามารถในการเฝ้าระวัง (Monitoring) ในส่วนงานของผู้ดูแลระบบ (Administrator) เป็นแบบออนไลน์ (Online) ทำให้เราสามารถตรวจสอบระบบคลัสเตอร์ผ่านทางอินเตอร์เน็ต (Internet) ได้ตลอดเวลาอีกด้วย

3) คลัสเตอร์มิดเดิลแวร์

คลัสเตอร์มิดเดิลแวร์ คือซอฟท์แวร์ที่ทำงานในระดับเดียวกับแกน (Kernel) ของระบบปฏิบัติการ มีหน้าที่ในการกระจายงาน (Process) จากโหนดหนึ่งไปยังโหนดอื่นๆ ที่อยู่ในระบบคลัสเตอร์เดียวกัน โดยส่วนใหญ่แล้วคลัสเตอร์มิดเดิลแวร์นี้จะเป็นซอฟท์แวร์ที่เขียนเพิ่มเติมเข้าไปในแกนของระบบปฏิบัติการ เพื่อให้ทุกเครื่องที่อยู่ในระบบคลัสเตอร์เสมือนเป็นเครื่องคอมพิวเตอร์หลายหน่วยประมวลผลขนาดใหญ่ ที่เสมือนมีหน่วยความจำ หน่วยประมวลผลอยู่ที่เดียวกันเหมือนกับระบบคอมพิวเตอร์แบบ SMP (Symmetric Multi Processor) หรือแบบ MMP (Massive Multi Processor)

4) การเชื่อมต่อเครือข่าย

การสื่อสารระหว่างหน่วยประมวลผลในระบบคลัสเตอร์จะทำผ่านระบบเครือข่ายความเร็วสูงซึ่งอุปกรณ์เครือข่ายแต่ละชนิดจะมีความเร็วและราคาแตกต่างกันไป ตัวอย่างอุปกรณ์เครือข่ายที่นิยมนำมาใช้ในการสร้างระบบคลัสเตอร์ มีดังนี้

ก) Ethernet ในปัจจุบันอุปกรณ์ Ethernet นั้นได้ถูกพัฒนาให้มีความเร็วในการรับส่งข้อมูลสูงมากขึ้นจนถึงระดับกิกะบิตอีเทอร์เน็ต (Gigabit Ethernet) หรือ 10 กิกะบิตอีเทอร์เน็ต คือมีความเร็วในการส่งผ่านข้อมูลประมาณ 1-10 พันล้านบิตต่อวินาที ซึ่งเป็นแบบที่ใช้ในระบบคลัสเตอร์ที่ใช้ทำวิจัยนี้ด้วย

ข) Myrinet มีความเร็วในการส่งผ่านข้อมูลประมาณ 2 พันล้านบิตต่อวินาที และมีค่า Latency Time ต่ำกว่าเครือข่ายแบบ Ethernet มาก แต่มีราคาแพงกว่าอีเทอร์เน็ตสูงมาก

ค) Quadrics มีความเร็วในการส่งข้อมูลอยู่ที่ 340-900 MB/Second หรือประมาณ 2.65-7 กิกะบิตต่อวินาที และมีค่า Latency Time ต่ำมาก

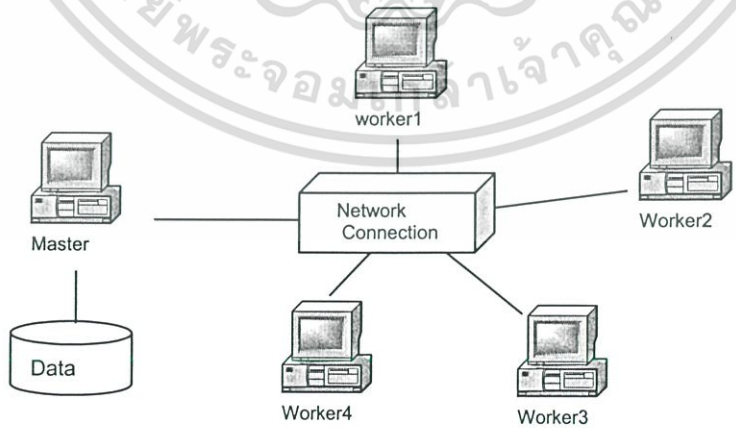
ง) InfiniBand เป็นเทคโนโลยีที่มีความเร็วในการสื่อสารข้อมูลสูงมากถึง 5 กิกะบิตต่อวินาที และมีค่า Latency Time น้อยกว่า 10 ไมโครวินาที (Microsecond)

5) เอ็มพีไอซีเอช (MPICH)

เอ็มพีไอซีเอช (MPICH) เป็นการพัฒนาชุดคำสั่งตามมาตรฐานขึ้นมาใช้งานจริงมากที่สุดคือ มาตรฐานการส่งผ่านข้อความ (Message Passing Interface: MPI) ซึ่งสามารถพัฒนาโปรแกรมภาษาต่างๆ เช่น โปรแกรมภาษาซี (C Language) โปรแกรมภาษาฟอร์แทน (Fortran Language) และโปรแกรมภาษาจาวา (Java Language) เป็นต้น ที่สนับสนุนการโปรแกรมแบบขนาน ซึ่งมีฟังก์ชันพื้นฐานต่างๆ รวมทั้งมีคำสั่งใช้งานเพื่อทำการคอมไพล์โปรแกรมที่เขียนขึ้น โดยเอ็มพีไอซีเอชนี้จะสนับสนุนการพัฒนาโปรแกรมภาษาซี และโปรแกรมภาษาฟอร์แทนเท่านั้น

4.2 ลักษณะข้อมูล การเลือกข้อมูลและเหตุผลการเลือก

ลักษณะของข้อมูลที่ใช้ในการทดลองเป็นข้อมูลชนิดเลขจำนวนจริงละเอียด 2 เท่า (Double Precision) โดยขนาดข้อมูลของเมตริกซ์จะอยู่ในช่วง 0.00-99.00 และกำหนดให้เมตริกซ์มีขนาดเป็นจัตุรัส คือ 512x512, 1024x1024, 2048x2048, 4096x4096 ทั้งของเมตริกซ์ A และเมตริกซ์ B ซึ่งในการวิจัยนี้ข้อมูลจะถูกจัดเก็บอยู่ในหน่วยความจำสำรองของหน่วยประมวลผลหลัก ซึ่งเรียกการเก็บข้อมูลแบบนี้ว่า "การเก็บข้อมูลแบบรวมศูนย์กลาง" (Centralized Data) ดังรูปที่ 4.1



รูปที่ 4.1 ระบบคลัสเตอร์ที่มีการเก็บข้อมูลแบบรวมศูนย์กลาง

4.3 ผลการทดลอง

จากการพัฒนาโปรแกรมแบบขนานการหาผลคูณของเมตริกซ์แบบขนาน ได้นำไปทดลองบนระบบพีซีคลัสเตอร์ ของสำนักวิจัยและบริการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ซึ่งประกอบด้วย หน่วยประมวลผล 10 หน่วย ขนาด 2.4 GHz จำนวน 5 เครื่อง หน่วยความจำหลัก 1 Gb หน่วยความจำสำรอง 100 Gb หน่วยความจำแคช (Cache memory) 512 Kb เชื่อมต่อกันด้วยความเร็วขนาด 1000 Mbps ระบบปฏิบัติการ Suse และโปรแกรม MPICH 1.2 ทำการวัดประสิทธิภาพของวิธีการคูณเมตริกซ์แบบขนานทั้ง 3 วิธีที่เสนอ รวมถึงการหาผลคูณเมตริกซ์แบบอนุกรมเมื่อนำมาเปรียบเทียบกัน การวัดประกอบด้วย เวลาที่ใช้ในการประมวลผล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) ซึ่งผลที่ได้จะแสดงได้ชัดเจนเมื่อเมตริกซ์มีขนาดใหญ่และมีจำนวนหน่วยประมวลผลเพิ่มมากขึ้น

การหาเวลาที่ใช้ในการประมวลผลของการหาผลคูณของเมตริกซ์แบบอนุกรมบนระบบคลัสเตอร์ที่อยู่ดังกล่าว จะสามารถหาระยะเวลาในการประมวลผลที่ถูกต้องเมื่อเมตริกซ์ขนาดน้อยกว่า 512x512 เนื่องมาจากสาเหตุ การประกาศตัวแปรเป็นแบบเลขจำนวนจริงละเอียด 2 เท่า (Double Precision) จะใช้พื้นที่ในหน่วยความจำ 8 ไบต์ เมื่อเมตริกซ์มีขนาด 512x512 จะใช้พื้นที่จำนวน 512x512x8 = 2,097,152 ไบต์ ซึ่งมีค่าเกินหน่วยความจำแคช (Cache memory) ทำให้ต้องใช้เวลาในการเข้าถึงข้อมูลนั้นๆ เพิ่มมากขึ้น ทำให้เวลาในการประมวลผลที่ได้มากกว่าเวลาปกติ ดังนั้นเวลาในการประมวลผลการหาผลคูณของเมตริกซ์แบบอนุกรมสำหรับเมตริกซ์ขนาดมากกว่า 512 x512 จะเป็นเวลาที่คำนวณขึ้นมา เพื่อนำไปใช้ในการเปรียบเทียบประสิทธิภาพของการหาผลคูณของเมตริกซ์แบบขนาน โดยมีรายละเอียดการเปรียบเทียบการหาผลคูณของเมตริกซ์ ดังนี้

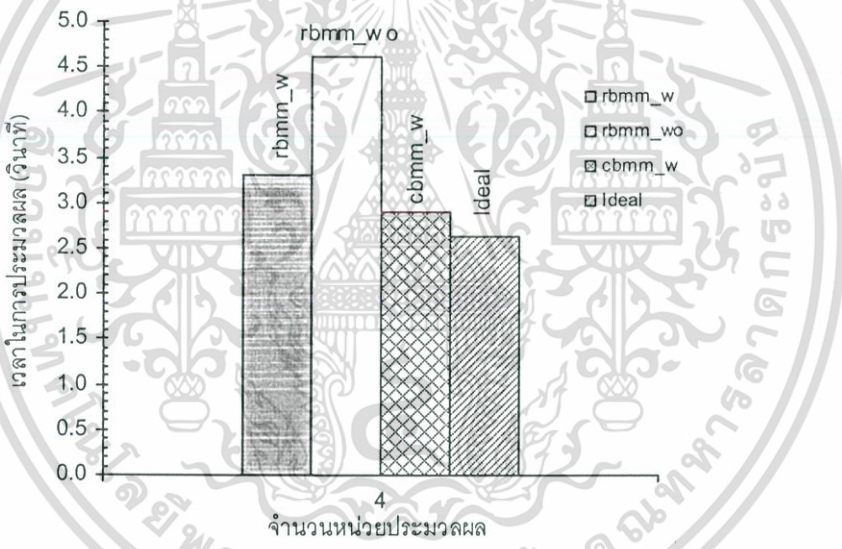
4.3.1 ผลการทดลองเปรียบเทียบกับเวลาในอุดมคติ

การทดลองเปรียบเทียบเวลาที่ประมวลผลจริงกับเวลาในอุดมคติเป็นการวัดประสิทธิภาพของระบบ (System Performance) ที่ใช้ในการทดลองดังนี้ ตารางที่ 4.1 และรูปที่ 4.2 แสดงผลการเปรียบเทียบเวลาที่ใช้ในการประมวลผล (Response Time) ทั้ง 3 แบบ ($T_P = T_S/P + T_C$) โดยในงานวิจัยนี้เน้นการเพิ่มประสิทธิภาพของเวลาที่ใช้ในการติดต่อสื่อสาร (Communication Time: T_C) ส่วนเวลาที่ใช้ในการประมวลผลข้อมูลในอุดมคติ (Response Time of Ideal Case) จะคำนวณโดยสมมติว่าเวลาที่ใช้ในการติดต่อสื่อสาร เพื่อแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลเท่ากับศูนย์ ($T_C = 0$) ดังนั้น $T_P = T_S/P$ ตามสมการที่ 2.1 ตารางที่ 4.2 และรูปที่ 4.3 แสดงอัตราการเพิ่มขึ้นของความเร็ว (Speedup) ทั้ง 3 วิธี ($S_P = T_S/T_P$) เปรียบเทียบกับอัตราการเพิ่มขึ้นของความเร็วในอุดมคติ

(S_P=P) ในตารางที่ 4.3 และรูปที่ 4.4 แสดงประสิทธิภาพในการหาผลคูณของเมตริกซ์ ทั้ง 3 วิธี (E_P = S_P/P) เปรียบเทียบกับประสิทธิภาพในอุดมคติ (E_P =1)

ตารางที่ 4.1 เวลาที่ใช้ในการหาผลคูณของเมตริกซ์ (Response Time) ทั้ง 3 วิธี เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล และเมตริกซ์ขนาด 1024x1024

วิธีการหาผลคูณของเมตริกซ์	เวลาในการประมวลผล (วินาที)
อุดมคติ	2.64
วิธีแบบขนาน	
rbmm_w	3.30
rbmm_wo	4.62
cbmm_w	2.90

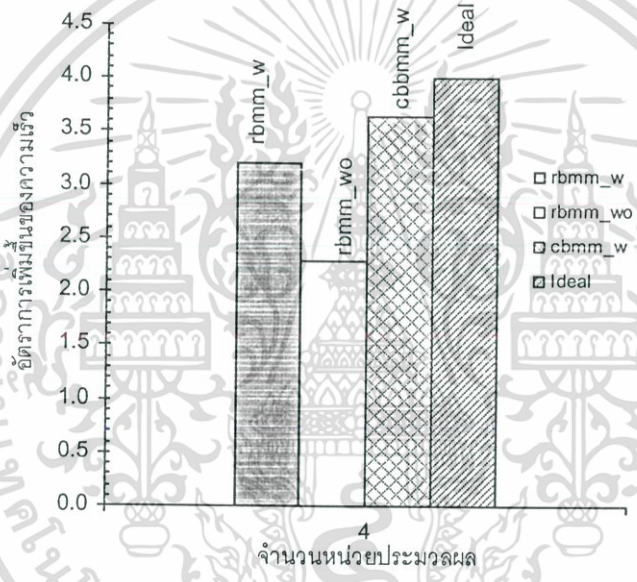


รูปที่ 4.2 เวลาที่ใช้ในการหาผลคูณของเมตริกซ์ ทั้ง 3 วิธี กับเวลาในอุดมคติ

จากรูปที่ 4.2 สามารถอธิบายได้ว่าเวลาที่ใช้ในการประมวลผลวิธี cbmm_w ที่เสนอในวิทยานิพนธ์ เมื่อจำนวนหน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล จะใช้เวลาในการประมวลผล 2.9 วินาทีใกล้เคียงกับเวลาในอุดมคติ คือ 2.64 วินาที ซึ่งคิดเป็น 91% เป็นวิธีที่ให้ผลดีที่สุด ส่วนวิธี rbmm_w ที่นิยมพัฒนาบนระบบพีซีคลัสเตอร์จะใช้เวลาในการประมวลผลเพิ่มขึ้น คือ 3.3 วินาทีและวิธีสุดท้ายคือวิธี rbmm_wo เป็นวิธีที่ใช้เวลาในการประมวลผลมากที่สุด คือ 4.62 วินาที

ตารางที่ 4.2 อัตราการเพิ่มขึ้นของความเร็ว (Speedup) โดยการเปรียบเทียบอัตราการเพิ่มขึ้นของความเร็วในอุดมคติ (Speedup of Ideal Case) เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล และเมตริกซ์ขนาด 1024x1024

วิธีการหาผลคูณของเมตริกซ์	อัตราการเพิ่มขึ้นของความเร็ว
อุดมคติ	4.00
วิธีแบบขนาน	
rbmm_w	3.19
rbmm_wo	2.28
cbmm_w	3.63

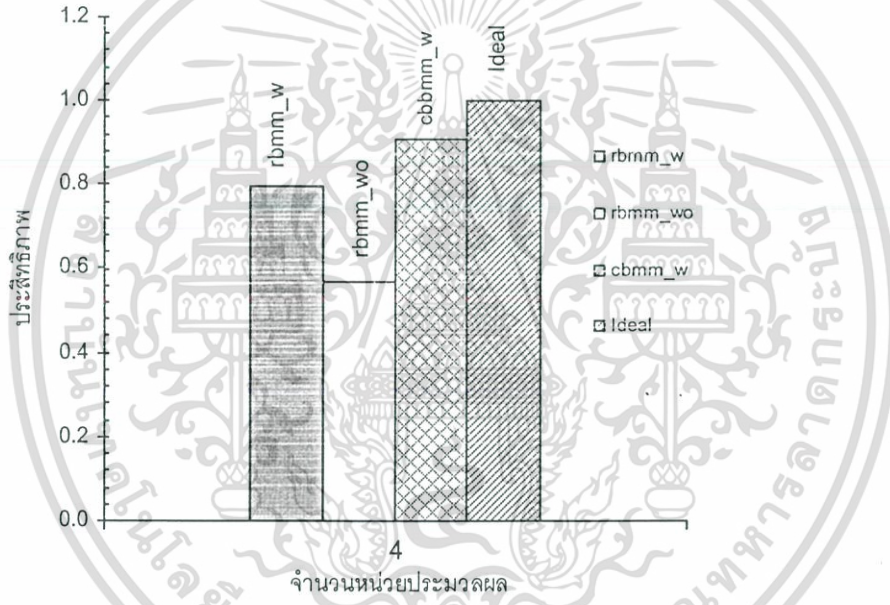


รูปที่ 4.3 อัตราการเพิ่มขึ้นของความเร็วทั้ง 3 วิธี เมื่อ P = 4

จากรูปที่ 4.3 สามารถอธิบายได้ว่าอัตราการเพิ่มขึ้นของความเร็วของการหาผลคูณของเมตริกซ์วิธี cbmm_w เท่ากับ 3.63 เมื่อใช้ 4 หน่วยประมวลผล ซึ่งเป็นวิธีที่ดีที่สุด ส่วนวิธี rbmm_w จะมีอัตราการเพิ่มขึ้นของความเร็วเท่ากับ 3.19 ลดลงน้อยกว่าวิธี cbmm_w และวิธี rbmm_wo มีอัตราการเพิ่มขึ้นของความเร็วเท่ากับ 2.28 ซึ่งน้อยกว่าสองวิธีแรกที่ได้กล่าวมาแล้ว

ตารางที่ 4.3 ประสิทธิภาพ (Efficiency) เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผลและเมตริกซ์ขนาด 1024x1024

วิธีการหาผลคูณของเมตริกซ์	ประสิทธิภาพ
อุดมคติ	1.00
วิธีแบบขนาน	
rbmm_w	0.80
rbmm_wo	0.57
cbmm_w	0.91



รูปที่ 4.4 เปรียบเทียบประสิทธิภาพทั้ง 3 วิธี เมื่อ P = 4

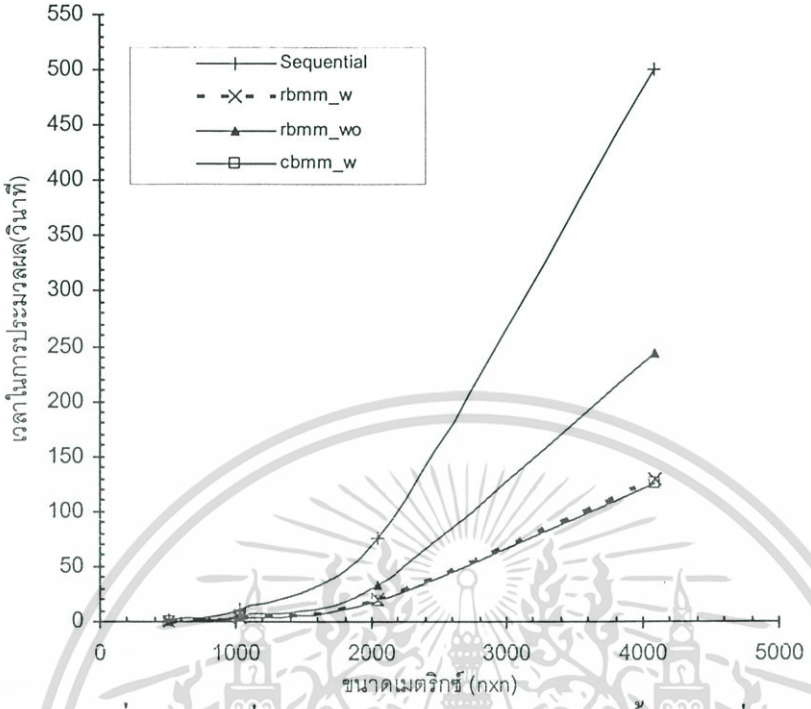
จากรูปที่ 4.4 แสดงประสิทธิภาพของการหาผลคูณของเมตริกซ์ทั้ง 3 วิธีจะเห็นว่าวิธี cbmm_w มีค่าประสิทธิภาพเท่ากับ 0.91 จะให้ผลเข้าใกล้ 1 เมื่อใช้ 4 หน่วยประมวลผล สำหรับประสิทธิภาพวิธี rbmm_w จะมีค่าประสิทธิภาพเท่ากับ 0.80 มีค่าน้อยกว่าวิธี cbmm_w ส่วนวิธี rbmm_wo จะมีประสิทธิภาพเท่ากับ 0.57 น้อยที่สุดเมื่อเทียบกับสองวิธีแรก

4.3.2 ผลการทดลองเปรียบเทียบเมื่อขนาดของเมตริกซ์ที่ใช้ทดลองมีขนาดต่างกัน

การทดลองเปรียบเทียบการหาผลคูณของเมตริกซ์เมื่อขนาดของเมตริกซ์ที่ใช้ทดลองมีขนาดต่างกัน กำหนดให้จำนวนหน่วยประมวลผลที่ใช้ในการคำนวณเท่ากับ 4 หน่วยประมวลผล ทำการเปรียบเทียบวิธีการหาผลคูณของเมตริกซ์แบบอนุกรม (Sequential) และวิธีการหาผลคูณของเมตริกซ์แบบขนานทั้ง 3 วิธี โดยใช้เมตริกซ์ขนาด 512x512, 1024x1024, 2048x2048 และ 4096x4096 ตามลำดับ ซึ่งผลการทดลองแสดงดังนี้ ตารางที่ 4.4 และรูปที่ 4.5 แสดงเวลาที่ใช้ในการประมวลผล (Response Time) ที่เป็นผลรวมของเวลาที่ใช้ในการติดต่อสื่อสารและเวลาที่ใช้ในการประมวลผล ($T_P = T_S/P+T_C$) นอกจากนี้เราสามารถนำเวลาที่ใช้ในการประมวลผลมาคำนวณหาอัตราการเพิ่มขึ้นของความเร็ว ตามสมการที่ 2.3 ($S_P = T_S/T_P$) ดังแสดงในตารางที่ 4.5 และรูปที่ 4.6 จากนั้นนำค่าของอัตราการเพิ่มขึ้นของความเร็วมาคำนวณหาประสิทธิภาพ ตามสมการที่ 2.4 ($E_P = S_P/P$) ดังแสดงในตารางที่ 4.6 และรูปที่ 4.7

ตารางที่ 4.4 เวลาที่ใช้ในวิธีการหาผลคูณของเมตริกซ์ (Response Time) ทั้ง 3 วิธี เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล โดยใช้เมตริกซ์ขนาด 512x512, 1024x1024, 2048x2048 และ 4096x4096 ตามลำดับ

ขนาดของเมตริกซ์	วิธีแบบอนุกรม	วิธีแบบขนาน		
		rbmm_w	rbmm_wo	cbmm_w
512 x 512	2.00	0.64	0.79	0.50
1024 x 1024	10.54	3.30	4.62	2.90
2048 x 2048	75.00	19.44	33.49	18.15
4096 x 4096	500.00	129.00	244.00	124.97

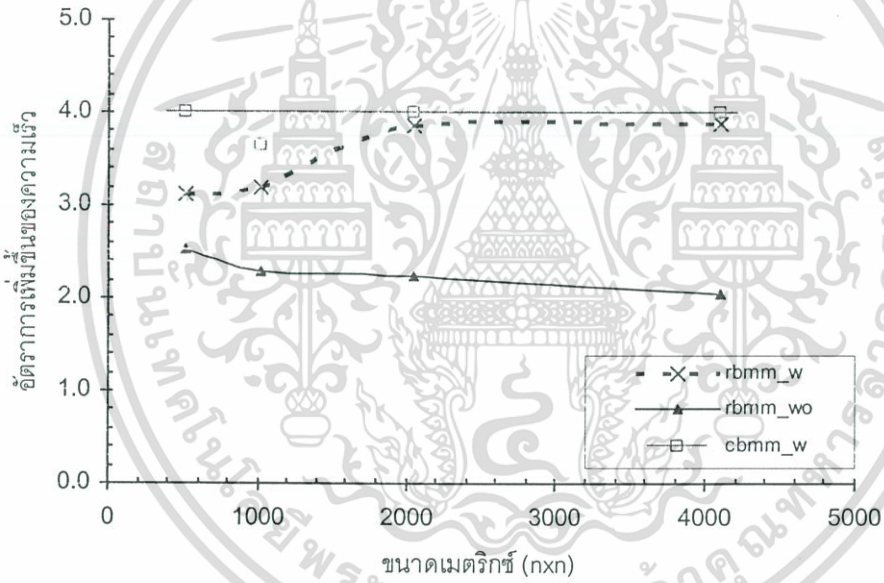


รูปที่ 4.5 เวลาที่ใช้ในการหาผลคูณของเมตริกซ์ ทั้ง 3 วิธี เมื่อ P = 4

จากรูปที่ 4.5 แสดงเวลาที่ใช้ในการหาผลคูณของเมตริกซ์ (Response Time) โดยทำการเปรียบเทียบเวลาทั้ง 3 วิธี ซึ่งเวลาที่ใช้ในการประมวลผลประกอบด้วยเวลา 2 ส่วนคือ ส่วนแรก เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Communication Time) ส่วนที่สองเวลาที่ใช้ในการประมวลผลข้อมูล (Computation Time) จากผลการทดลองรูปที่ 4.5 จะเห็นว่าเมื่อข้อมูลมีขนาดใหญ่ขึ้นเวลาที่ใช้ในการหาผลคูณของเมตริกซ์นั้นจะเพิ่มขึ้นตามไปด้วย โดยวิธี cbmm_w ใช้เวลาในการประมวลผลรวมน้อยที่สุดของทุกวิธี สำหรับวิธี rbmm_w ใช้เวลาในการประมวลผลรวมเพิ่มมากกว่าวิธี cbmm_w เล็กน้อย และวิธี rbmm_wo ใช้เวลาในการประมวลผลรวมมากกว่าสองวิธีดังกล่าวมาก เนื่องจากต้องการเวลาที่ใช้ในการติดต่อสื่อสารเพื่อแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล

ตารางที่ 4.5 อัตราการเพิ่มขึ้นของความเร็ว (Speedup) เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล โดยใช้เมตริกซ์ขนาด 512x512, 1024x1024, 2048x2048 และ 4096x4096 ตามลำดับ

ขนาดของเมตริกซ์	วิธีแบบขนาน		
	rbmm_w	rbmm_wo	cbmm_w
512 x 512	3.13	2.53	4.00
1024 x 1024	3.19	2.28	3.63
2048 x 2048	3.86	2.24	4.00
4096 x 4096	3.88	2.05	4.00

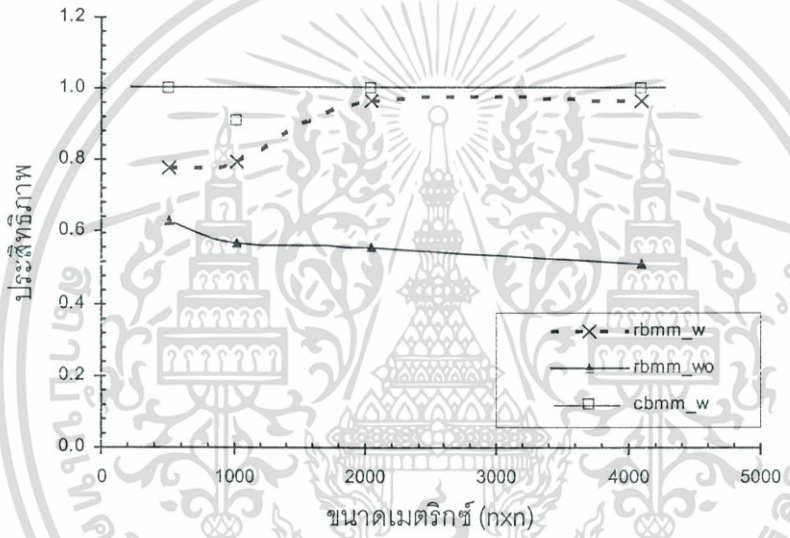


รูปที่ 4.6 อัตราการเพิ่มขึ้นของความเร็วทั้ง 3 วิธี เมื่อ P = 4

จากรูปที่ 4.6 แสดงการเปรียบเทียบอัตราการเพิ่มขึ้นของความเร็วทั้ง 3 วิธี เมื่อจำนวนหน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล เมตริกซ์มีขนาด 512x512, 1024x1024, 2048x2048 และ 4096x4096 ตามลำดับ จะเห็นว่าวิธี cbmm_w มีอัตราการเพิ่มขึ้นของความเร็วมากที่สุด ส่วนวิธี rbmm_w มีค่าลดลงตาม สุดท้ายคือวิธี rbmm_wo มีอัตราการเพิ่มขึ้นของความเร็วน้อยที่สุด

ตารางที่ 4.6 ประสิทธิภาพ (Efficiency) เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล โดยใช้เมตริกซ์ขนาด 512x512, 1024x1024, 2048x2048 และ 4096x4096 ตามลำดับ

ขนาดของเมตริกซ์	วิธีแบบขนาน		
	rbmm_w	rbmm_wo	cbmm_w
512 x 512	0.78	0.63	1.00
1024 x 1024	0.80	0.57	0.91
2048 x 2048	0.96	0.56	1.00
4096 x 4096	0.97	0.51	1.00



รูปที่ 4.7 ประสิทธิภาพทั้ง 3 วิธี เมื่อ P = 4

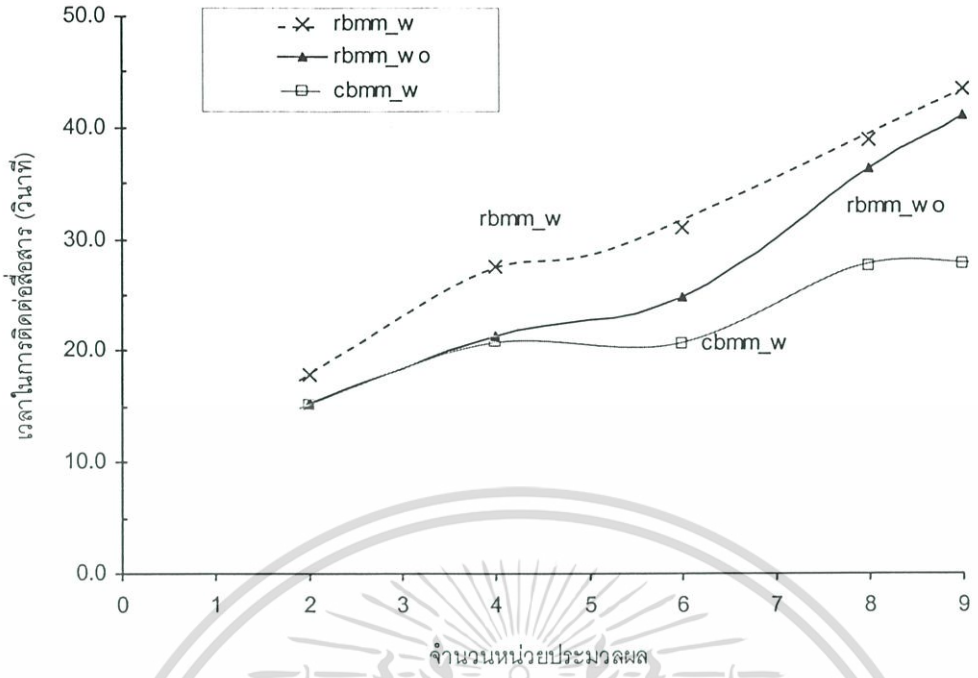
จากรูปที่ 4.7 แสดงการเปรียบเทียบประสิทธิภาพทั้ง 3 วิธี เมื่อจำนวนหน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล เมตริกซ์ขนาด 512x512, 1024x1024, 2048x2048 และ 4096x4096 ตามลำดับ จะเห็นว่าวิธี cbmm_w มีประสิทธิภาพมากที่สุด ส่วนวิธี rbmm_w มีค่าลดลง สุดท้ายคือวิธี rbmm_wo มีประสิทธิภาพน้อยที่สุด

4.3.3 ผลการทดลองเปรียบเทียบเมื่อจำนวนหน่วยประมวลผลที่ใช้ทดลองต่างกัน

ผลการทดลองเปรียบเทียบการหาผลคูณของเมตริกซ์เมื่อกำหนดขนาดของเมตริกซ์ 4010x4010 และทำการเพิ่มจำนวนหน่วยประมวลผลตั้งแต่ 2 หน่วยประมวลผล 4 หน่วยประมวลผล 6 หน่วยประมวลผล 8 หน่วยประมวลผล และ 9 หน่วยประมวลผลตามลำดับ โดยจะพิจารณาเวลาที่ใช้ในการหาผลคูณของเมตริกซ์แบบขนานได้เป็น 2 ส่วน คือ ส่วนแรกเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลเพื่อแลกเปลี่ยนข้อมูล (Communication Time) ดังแสดงในตารางที่ 4.7 และรูปที่ 4.8 ส่วนที่สองเวลาทั้งหมดที่ใช้ในการประมวลผล (Computation Time) ซึ่งเป็นผลรวมของเวลาที่ใช้ในการติดต่อสื่อสารและเวลาที่ใช้ในการประมวลผล ($T_P = T_S/P + T_C$) แสดงในตารางที่ 4.8 และรูปที่ 4.9 นอกจากนี้เราสามารถนำเวลาที่ใช้ในการประมวลผลมาคำนวณหาอัตราการเพิ่มขึ้นของความเร็ว ตามสมการที่ 2.3 ($S_p = T_S/T_p$) ดังแสดงในตารางที่ 4.9 และรูปที่ 4.10 จากนั้นนำค่าของอัตราการเพิ่มขึ้นของความเร็วมาคำนวณหาประสิทธิภาพตามสมการที่ 2.4 ($E_p = S_p/P$) ดังแสดงในตารางที่ 4.10 และรูปที่ 4.11

ตารางที่ 4.7 เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Communication Time) โดยใช้หน่วยประมวลผลตั้งแต่ 2, 4, 6, 8 และ 9 หน่วยตามลำดับ และขนาดของเมตริกซ์เท่ากับ 4010x4010

จำนวนหน่วยประมวลผล	วิธีแบบขนาน		
	rbmm_w	rbmm_wo	cbmm_w
2	17.81	15.32	15.10
4	27.56	21.32	20.84
6	31.11	24.79	20.69
8	38.92	36.29	27.63
9	43.39	41.08	27.66

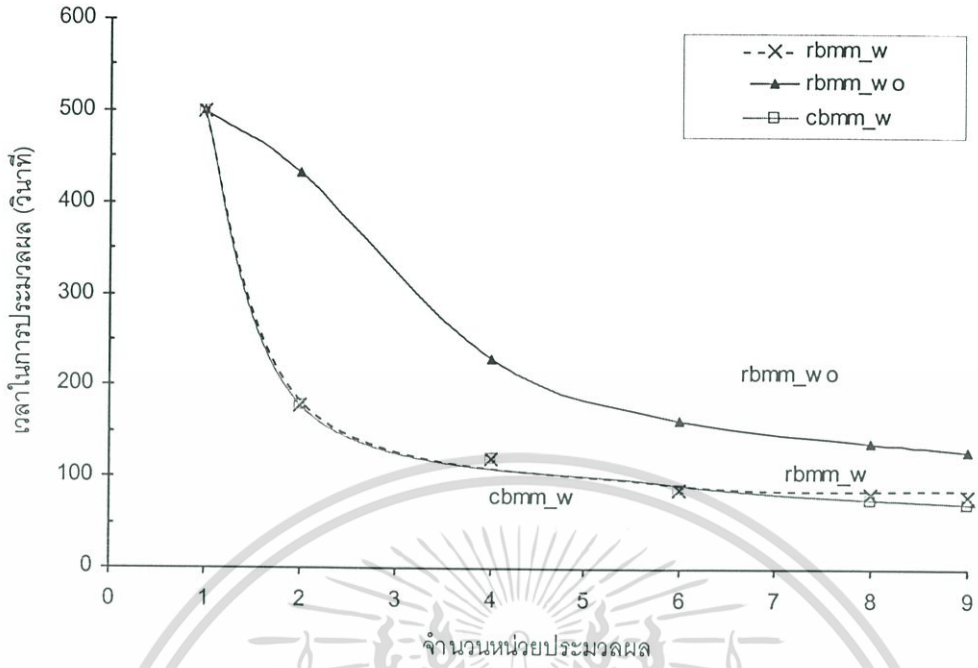


รูปที่ 4.8 เปรียบเทียบเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล

จากรูปที่ 4.8 แสดงเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล ซึ่งจะเห็นว่าวิธี rbmm_wo ใช้เวลาในการติดต่อสื่อสารระหว่างหน่วยประมวลผลน้อยกว่าวิธี rbmm_w เนื่องจากมีการใช้ข้อมูลที่ไม่ซ้ำซ้อนกันในแต่ละหน่วยประมวลผล ส่วนวิธี cbmm_w ใช้เวลาในการติดต่อสื่อสารระหว่างหน่วยประมวลผลน้อยที่สุด

ตารางที่ 4.8 เวลาที่ใช้ในการประมวลผล (Response Time) โดยใช้หน่วยประมวลผลตั้งแต่ 1, 2, 4, 6, 8 และ 9 หน่วยตามลำดับ และขนาดของเมตริกซ์ เท่ากับ 4010x4010

จำนวนหน่วยประมวลผล	วิธีแบบขนาน		
	rbmm_w	rbmm_wo	cbmm_w
1	500.00	500.00	500.00
2	178.19	432.09	178.00
4	121.24	229.23	121.11
6	86.58	162.84	86.48
8	81.58	138.50	76.09
9	79.32	130.28	70.35

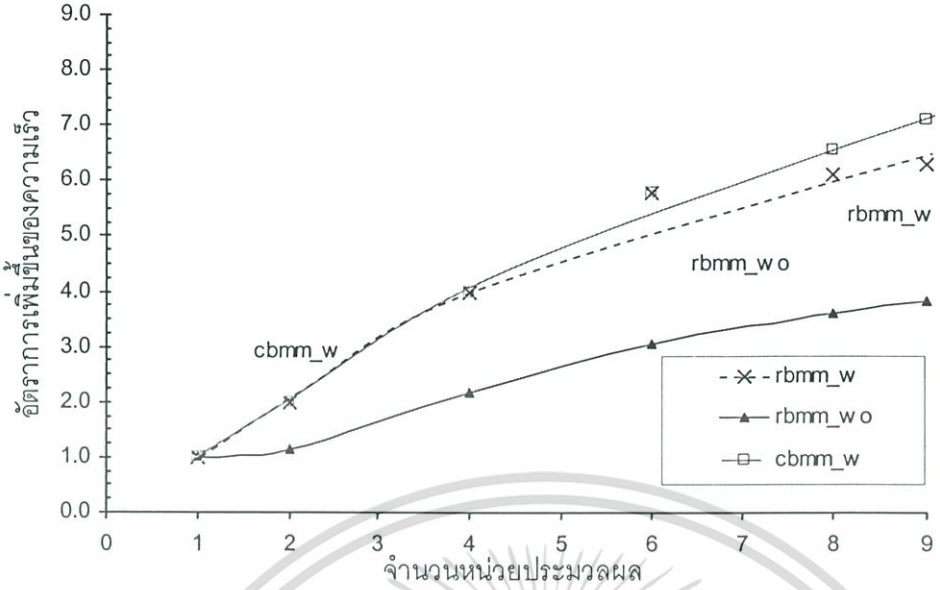


รูปที่ 4.9 เปรียบเทียบเวลาที่ใช้ในการประมวลผล

จากรูปที่ 4.9 แสดงเวลาที่ใช้ในการประมวลผล (Response Time) โดยวิธี cbmm_w ซึ่งจะเห็นได้ว่าเมื่อเพิ่มจำนวนหน่วยประมวลผล เวลาในการประมวลผลจะลดลงเท่ากับ 178.00 วินาที, 121.11 วินาที, 86.48 วินาที, 76.09 วินาทีและ 70.35 วินาที เมื่อจำนวนหน่วยประมวลผลเท่ากับ 2, 4, 6, 8 และ 9 หน่วยประมวลผลตามลำดับ สำหรับที่ 8 และ 9 หน่วยประมวลผลเวลาในการประมวลผลไม่ลดลงมาก เนื่องจากต้องใช้ระยะเวลาในการติดต่อระหว่างหน่วยประมวลผลเพิ่มมากขึ้น

ตารางที่ 4.9 อัตราการเพิ่มขึ้นของความเร็ว (Speedup) โดยใช้หน่วยประมวลผลตั้งแต่ 1, 2, 4, 6, 8 และ 9 หน่วยตามลำดับ และขนาดของเมตริกซ์ เท่ากับ 4010x4010

จำนวนหน่วยประมวลผล	วิธีแบบขนาน		
	rbmm_w	rbmm_wo	cbmm_w
1	1.00	1.00	1.00
2	2.00	1.16	2.00
4	4.00	2.18	4.00
6	5.78	3.07	5.78
8	6.13	3.61	6.57
9	6.30	3.84	7.11

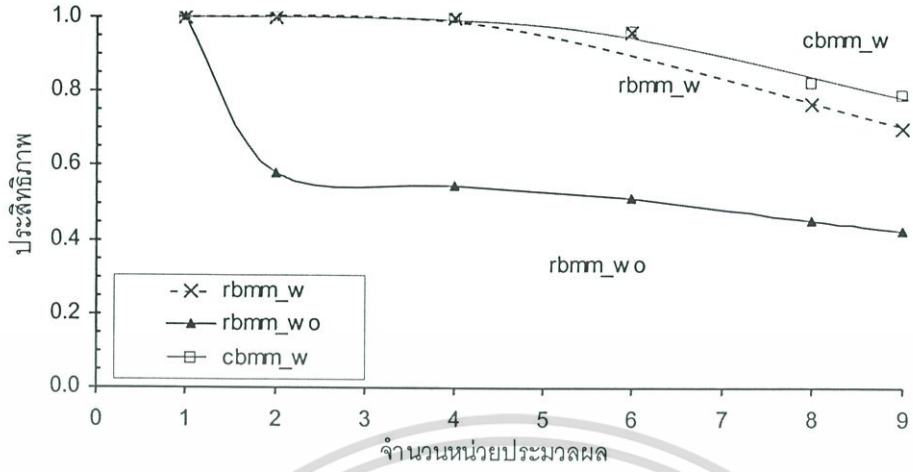


รูปที่ 4.10 เปรียบเทียบอัตราการเพิ่มขึ้นของความเร็ว

จากรูปที่ 4.10 แสดงอัตราการเพิ่มขึ้นของความเร็ว วิธี cbmm_w จะเห็นได้ว่าเมื่อเพิ่มจำนวนหน่วยประมวลผล อัตราการเพิ่มขึ้นของความเร็วจะเพิ่มขึ้นใกล้เคียงกับจำนวนหน่วยประมวลผล ซึ่งเท่ากับ 2.00, 4.00, 5.78, 6.57 และ 7.11 เมื่อจำนวนหน่วยประมวลผลเท่ากับ 2, 4, 6, 8 และ 9 หน่วยประมวลผลตามลำดับ สำหรับที่จำนวนหน่วยประมวลผลเท่ากับ 8 และ9 หน่วยประมวลผล อัตราการเพิ่มขึ้นของความเร็วไม่ใกล้เคียงกับจำนวนหน่วยประมวลผล เนื่องจากต้องใช้ระยะเวลาในการติดต่อระหว่างหน่วยประมวลผลเพิ่มมากขึ้น ส่วนวิธี cbmm_wo และวิธี rbmm_w มีอัตราการเพิ่มขึ้นของความเร็วลักษณะเช่นเดียวกับวิธี cbmm_w

ตารางที่ 4.10 ประสิทธิภาพ (Efficiency) โดยใช้หน่วยประมวลผลตั้งแต่ 1, 2, 4, 6, 8 และ 9 หน่วยตามลำดับ และขนาดของเมตริกซ์ เท่ากับ 4010x4010

จำนวนหน่วยประมวลผล	วิธีแบบขนาน		
	rbmm_w	rbmm_wo	cbmm_w
1	1.00	1.00	1.00
2	1.00	0.58	1.00
4	1.00	0.55	1.00
6	0.96	0.51	0.96
8	0.77	0.45	0.82
9	0.70	0.43	0.79



รูปที่ 4.11 เปรียบเทียบประสิทธิภาพ

จากรูปที่ 4.11 แสดงผลการเปรียบเทียบประสิทธิภาพ วิธี cbmm_w ให้ประสิทธิภาพเท่ากับ 1 เมื่อหน่วยประมวลผลเท่ากับ 2 และ 4 หน่วยประมวลผล ตามลำดับ ส่วนที่ 8 หน่วยประมวลผลให้ประสิทธิภาพเท่ากับ 0.82 และ 9 หน่วยประมวลผลให้ประสิทธิภาพเท่ากับ 0.79 เนื่องจากใช้ระยะเวลาในการติดต่อระหว่างหน่วยประมวลผลเพิ่มมากขึ้น ส่งผลให้ประสิทธิภาพลดลง ส่วนวิธี rbmm_w และวิธี rbmm_wo มีค่าประสิทธิภาพลักษณะเช่นเดียวกับวิธี cbmm_w

บทที่ 5
สรุปผลการทดลองและแนวทางการพัฒนางานวิจัย

5.1 สรุปผลและวิเคราะห์ผลการทดลอง

การหาผลคูณของเมตริกซ์แบบขนานบนระบบพีซีคลัสเตอร์ที่ได้เสนอไว้ในบทที่ 3 ซึ่งมีทั้งหมด 3 วิธี คือ วิธีที่ 1 การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmm_w (Row-Block Partitioning Matrix Multiplication with replicated data) วิธีที่ 2 การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmm_wo (Row-Block Partitioning Matrix Multiplication without replicated data) และวิธีที่ 3 การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี cbmm_w (Checkerboard-Block Partitioning Matrix Multiplication with replicated data) โดยแต่ละแบบดังกล่าวมีวัตถุประสงค์และความยากง่ายในขั้นตอนวิธีที่แตกต่างกันดังนี้คือ วิธีแรก (วิธี rbmm_w) เป็นวิธีที่มีผู้เสนอไว้แล้วบนระบบพีซีคลัสเตอร์ซึ่งเป็นวิธีที่มีประสิทธิภาพในการติดต่อสื่อสารมากที่สุด แต่ใช้เนื้อที่ในการเก็บข้อมูลซ้ำซ้อนมากที่สุด ส่วนสองวิธีหลัง (วิธี rbmm_wo และวิธี cbmm_w) เป็นวิธีใหม่ที่เสนอขึ้นในวิทยานิพนธ์นี้เพื่อเพิ่มประสิทธิภาพโดยการลดการใช้ข้อมูลที่ซ้ำซ้อนกันระหว่างหน่วยประมวลผล โดยวิธีที่สองไม่มีการเก็บข้อมูลที่ซ้ำซ้อนกันเลยแต่เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลมีค่าเพิ่มมากขึ้น ส่วนวิธีที่สาม (วิธี cbmm_w) เป็นการนำข้อดีของวิธีที่ 1 (วิธี rbmm_w) และวิธีที่ 2 (วิธี rbmm_wo) มารวมกัน คือ จัดเก็บข้อมูลที่ซ้ำซ้อนกันบ้างแต่น้อยกว่าวิธีที่ 1 (หรือดีกว่าวิธีที่ 1) แต่มากกว่าวิธีที่ 2 และยังคงรักษาประสิทธิภาพในการติดต่อสื่อสารระหว่างหน่วยประมวลผลได้มากที่สุดเท่าวิธีที่ 1 ซึ่งดีกว่าวิธีที่ 2

จากผลการทดลองที่ผ่านมาในบทที่ 4 ซึ่งเป็นการทดลองบนระบบพีซีคลัสเตอร์ของสำนักบริการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง จะเห็นว่าการหาผลคูณของเมตริกซ์แบบขนานในแต่ละวิธีจะมีการเพิ่มประสิทธิภาพในด้านต่างๆ ที่แตกต่างกันสอดคล้องกับทฤษฎีที่กล่าวมาแล้วในบทที่ 3 ซึ่งสามารถแบ่งได้เป็น 2 ด้าน ดังนี้

1) การเพิ่มประสิทธิภาพโดยการลดเวลาที่ใช้ในการประมวลผลทั้งหมด
2) การลดการใช้ข้อมูลที่ซ้ำซ้อนกันในแต่ละหน่วยประมวลผล

5.2 แนวทางการพัฒนางานวิจัย

นำโปรแกรมที่ได้พัฒนาขึ้นไปใช้กับระบบคลัสเตอร์แบบเนื้อผสม (Heterogeneous Cluster) จำเป็นต้องให้ความสำคัญในเรื่องของการสร้างสมดุลของงาน (Load Balancing) ในระบบ เนื่องจากคอมพิวเตอร์ที่เชื่อมต่ออยู่ในระบบนี้ สามารถมีองค์ประกอบภายในที่แตกต่างกันได้ ทำให้คอมพิวเตอร์แต่ละเครื่องมีความสามารถและประสิทธิภาพในการประมวลผลงานที่แตกต่างกัน หรือนำโปรแกรมไปพัฒนาบนการประมวลผลแบบกริด (Grid computing) ซึ่งเป็นเทคโนโลยีการประมวลผลรวมศักยภาพจากเครื่องที่อยู่แยกกันตามที่ต่างๆ เข้าด้วยกัน โดยมีความร่วมมือสร้างกันระหว่างสถาบันการศึกษาในประเทศไทย (http://www.thaigrid.net)

นำไปพัฒนาโปรแกรมให้สามารถใช้งานได้ง่ายขึ้นและใช้ได้หลายระบบปฏิบัติการ เช่น พัฒนาบนระบบปฏิบัติการไมโครซอฟท์วินโดวส์ เป็นต้น และสามารถนำโปรแกรมที่พัฒนาขึ้นไปประยุกต์ใช้งานร่วมกับแบบจำลองด้านต่างๆ เช่น แบบจำลองทางด้านพยากรณ์อากาศอุตุนิยมวิทยา เพื่อช่วยคาดการณ์ภัยธรรมชาติ เป็นต้น

เอกสารอ้างอิง

[1] Alonso Sanches C.A. and Song S.W. "SIMD Algorithms for Matrix Multiplication on the Hypercube" **The 8th Int'l Proceedings on Parallel Processing Symposium**. 1994. pp.490-496.

[2] Beaumont O., Boudet V., Rastello F. and Robert Y., "Matrix Multiplication on Heterogeneous Platforms." **IEEE Trans. on Parallel and Distributed Systems**. v.12 (10), 2001. pp.1033-1051.

[3] Browne S., Dongarra J. and London K. **Review of Performance Analysis Tools for MPI Parallel programs**. [Online]. Available:http://www.cs.utk.edu/~browne/ perftools - review/. 1997.

[4] Choi J. "A Fast Scalable Universal Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers." **IEEE Trans. on Parallel and Distributed Systems 3**. 1997. pp.310-314.

[5] Choi J. "A New Parallel Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers." **HPC Asia '97 High Performance Computing on the Information Superhighway**. 1997. pp.224-229.

[6] Grefory F. Pfister. **In Search of Clusters The Coming Battle in Lowly Parallel Computing**. New Jersey :Prentice Hall PTR. Prentice-Hall, Inc. 1995.

[7] Gropp W. Lusk E. and Skjellum A. **Using MPI: Portable Parallel Programming with the Massage Passing Interface.** Cambridge, MA :MIT Press. 1994.

[8] Gunnels J., Lin C., Morrow G. and Geijn R. **Analysis of a Class of Parallel Matrix MultiplicationAlgorithms.** [Online]. Available :http:// www.cs.utexas.edu /users /plapack /papers /ipps98 /ipps98.html. 1998.

[9] Hwang K. **Advance Computer Architecture : Parallelism Scalability Programmability**. New York : Mcgraw-Hill. 1993.

[10] Kumar V. **Introduction to parallel computing : design and analysis of algorithms.** Redwood City, CA : Benjamin/Cummings. 1994.

[11] Le T.T. and Huu T.C. **Advances in Parallel Computing For the Year 2000 and Beyond.** [Online]. Available : http://www.vacets.org/vtic97/ttle.htm. 2005.

[12] Li K. "Scalable Parallel Matrix Multiplication on Distributed Memory Parallel Computers." **Parallel and Distributed Processing Symposium IPDPS 2000.** Proceedings. 14th International, 2000. pp.307-314.

[13] Miller R. **Algorithms sequential and parallel : a unified approach.** Upper Saddle River, NJ : Prentice Hall. 2000.

[14] Quinn J. M. **Parallel Programming in C with MPI and OpenMP.** International Edition. Singapore : McGraw Hill. 2003.

[15] Quinn J. M. **Parallel computing : theory and practice.** 2nd ed. New York : McGraw-Hill. 1994.

[16] Samutrak P., Boonniyom J. and Srisawat J. "Parallel Matrix Multiplication and Application on a Cluster of PCs." **The 2nd International Symposium on Mathematical, Statistical and Computer Sciences 2005.** 19-20 February 2005. pp.34-42.

[17] Sengupta A. and Raghavendra C.S. "All-To-All Broadcast and Matrix Multiplication in Faulty SIMD Hypercubes." **IEEE Trans. on Parallel and Distributed Systems.** v.9(6),1998. pp.550-560.

[18] Tasic J.F., Zajc M. and Kosir A. "Comparison of Some Parallel Matrix Multiplication Algorithms." **Electrotechnical Conference MELECON '96.** 8th Mediterranean. v.1, 1996. pp.155-158.

[19] Typou T., Stefanidis V., Michailidis P. and Margaritis K. "Implementing Matrix Multiplication on an MPI Cluster of Workstations." **The 1st Int'l Conference from Scientific Computing to Computational Engineering (IC-SCCE).** Athens. 2004.

[20] นพรัตน์ พันธุ์เสนา "**การสร้างภาพเชิงปริมาตรบนระบบคลัสเตอริ่งโดยวิธีการแปลงเฉือนและบิด.**" วิทยานิพนธ์วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์ บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2547.

[21] ภุชงค์ อุทโยภาส, "แนะนำเทคโนโลยีระบบพีซีคลัสเตอร์" **วิศวกรรมสาร มก.** ปีที่ 16, ฉบับที่47, สิงหาคม-พฤศจิกายน 2545. หน้า 1-8.

[22] อำไพ พรประเสริฐกุล. **Introduction to Computer Organization.** กรุงเทพมหานคร : บ.ซีเอ็ดยูเคชั่น จำกัด (มหาชน). 2543.

ประวัติผู้เขียน

ชื่อ – สกุล นายไพโรจน์ สมุทรักษ์

วัน เดือน ปีเกิด 9 กรกฎาคม พ.ศ. 2520

ที่อยู่ 195/4 ถนนณรงค์วิถี ตำบลอุทัยใหม่
อำเภอเมือง จังหวัดอุทัยธานี 61000

ประวัติการศึกษา

พ.ศ. 2546 จบการศึกษาปริญญาครุศาสตรบัณฑิต
สาขาคอมพิวเตอร์ศึกษา สถาบันราชภัฏจันทรเกษม

พ.ศ. 2542 จบการศึกษาปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิศวกรรมทรัพยากรน้ำ มหาวิทยาลัยเกษตรศาสตร์

ประวัติการทำงาน

พฤศจิกายน พ.ศ. 2542 วิศวกรทรัพยากรน้ำ บริษัท แมคโครคอนซัลแตนท์ จำกัด