การพาร์ทิชันข้อมูลสำหรับการคูณเมตริกซ์ทรานสโพสแบบขนาน

DATA PARTITIONING FOR PARALLEL TRANSPOSE OF MATRIX MULTIPLICATION

นันทิพัฒน์ พิศุทธางกูร

NUNTIPAT PHISUTTHANGKOON

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2556

KMITL – 2013 – SC – M – 002 – 031

DATA PARTITIONING FOR PARALLEL TRANSPOSE OF MATRIX MULTIPLICATION

NUNTIPAT PHISUTTHANGKOON

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCE

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2013

KMITL – 2013 – SC – M – 002 – 031

หัวข้อวิทยานิพนธ์	การพาร์ทิชันข้อมูลสำหรับการคูณเมตริกซ์ทรานสโพสแบบ
ขนานนักศึกษา	นายนันทิพัฒน์ พิศุทธางกูร
รหัสประจำตัว	54650804
ปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา	วิทยาการคอมพิวเตอร์
พ.ศ.	2556
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.ดร.จีรพร วีระพันธุ์

บทคัดย่อ

เมตริกซ์ (Matrix) เป็นเทคนิคขั้นพื้นฐานที่สำคัญต่องานวิจัย และพัฒนาทางเทคโนโลยีในหลายๆ สาขาวิชา เช่น งานทางด้านวิศวกรรมศาสตร์ และทางด้านวิทยาศาสตร์ อย่างไรก็ตาม จุดอ่อนของเมตริกซ์ คือ การคำนวณที่ใช้เวลานาน โดยเฉพาะอย่างยิ่งเมื่อเมตริกซ์มีขนาดใหญ่ ดังนั้นงานวิจัยนี้ได้นำเสนอการประมวลผลเมตริกซ์แบบหนึ่ง คือ การคูณเมตริกซ์ ($C = A \times B$) ที่นำวิธีการไทลิงก์ (Tiling Technique) มาประยุกต์ในขั้นตอนการทรานสโพสเมตริกซ์ ก่อนที่จะดำเนินการคูณเมตริกซ์ โดยในการทรานสโพส เมตริกซ์จะทำเฉพาะข้อมูลเข้าเมตริกซ์ที่สอง (เมตริกซ์ B) ซึ่งวิธีการนี้จะช่วยลดจำนวนในการย้ายข้อมูลของแต่ละหน่วยประมวลผล และเพื่อทำให้ง่ายต่อการเข้าถึงข้อมูลตามมาตรฐานเอ็มพีไอ (MPI Standard) ซึ่งเป็นมาตรฐานในการเขียนโปรแกรมแบบขนานที่งานวิจัยนี้ได้นำมาใช้ การหาผลคูณเมตริกซ์ทรานสโพสที่นำเสนอมี 3 วิธี คือ 1) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w (row-block partitioning matrix-transpose multiplication with replicated data) 2) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี cbmtm_w (checkerboard-block partitioning matrix-transpose multiplication with replicated data) 3) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w/o (row-block partitioning matrix-transpose multiplication withoutreplicated data) โดยแต่ละวิธีจะแตกต่างกันตามลักษณะการจัดแบ่งข้อมูล และการเก็บข้อมูลที่ซ้ำซ้อนกันในแต่ละหน่วยประมวลผล จากผลการทดลองแสดงให้เห็นว่าวิธี cbmtm_wใช้เวลาในการประมวลผลดีที่สุดที่ 1.5 วินาที เมื่อใช้เมตริกซ์ขนาด 1024×1024 และวิธีที่มีประสิทธิภาพน้อยที่สุดคือวิธี rbmtm_w/o ซึ่งใช้เวลาในการประมวลผล 3.53 วินาที

Thesis	Data Partitioning for Parallel Transpose of Matrix Multiplication
Student	Mr.Nuntipat Phisutthangkoon
Student ID	54650804
Degree	Master of Science
Programme	Computer Science
Year	2013
Thesis Advisor	Assoc. Prof. Dr. Jeeraporn Werapun

ABSTRACT

Matrix is a fundamentally important technique in various fields of technological research and development such as engineering works and scientific researches. However, the weakness of matrixes is an expensive computation, especially in the large data set. Therefore, the research proposes a parallel matrix multiplication C = A×B by using matrix transpose with tiling technique. This technique suggests the tiling-based transpose for the second matrix (B) only. The proposed method can decrease a number of data moving in each processor. Afterward, matrix B is changed to transpose for simple and fast access by MPI standard, a standard of parallel programming design used in this research. Then, three methods (row-block partitioning matrix-transpose multiplication with replicated data, checkerboard-block partitioning matrix-transpose multiplication with replicated data, and row-block partitioning matrix-transpose multiplication without replicated data) of parallel matrix multiplication are introduced, which are different in data parting and data replicating in each processor. To evaluate method's performance, the response time of three methods are compared to the matrix multiplication with the ideal transpose technique. Among all methods, the experimental results show that the checkerboard-block partitioning matrix-transpose multiplication with replicated data method (cbmtm_w) achieves highest response time, which are 1.5sec, when the matrix size is 1024×1024.In other hand, the worst method was performed by row-block partitioning matrix-transpose multiplication without replicated data (rbmtm_w/o), the method achieves response time 3.53 sec.

กิตติกรรมประกาศ

วิทยานิพนธ์นี้มิอาจจะสำเร็จลุล่วงไปได้ด้วยดี หากมิได้รับคำแนะนำ คำชี้แจง ความรู้ และความเอาใจใส่จาก รศ.ดร.จีรพร วีระพันธุ์ ผู้เป็นอาจารย์ที่ปรึกษา ซึ่งท่านได้สละเวลาให้กับข้าพเจ้าอย่างเต็มที่ จึงใคร่ขอขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ รศ.ดร.วีระ บุญจริง ดร.สายชล ใจเย็น และดร.เฉลิมศักดิ์ เลิศวงศ์เสถียร คณะกรรมการสอบหัวข้อ และโครงร่างวิทยานิพนธ์ ที่กรุณาให้คำแนะนำตลอดจนข้อชี้แนะจนในที่สุดทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลงได้

ขอขอบพระคุณบิดา มารดา ที่สนันสนุนให้ได้เรียนในระดับที่ได้ตั้งใจ และเป็นกำลังใจในระหว่างการศึกษาเป็นอย่างดียิ่ง

ขอขอบคุณ นายเสฏฐนันท์ ทองสุวรรณ์ และนางสาวปาริฉัตร นาคสิทธิ์ พี่ๆ และเพื่อนๆทุกคนที่ให้คำปรึกษา และช่วยอำนวยความสะดวกในด้านต่างๆ

สำหรับคุณงามความดีและประโยชน์อันใดที่เกิดขึ้นจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดา มารดา อาจารย์ทุกท่านซึ่งเป็นที่เคารพรักยิ่ง ตลอดจนญาติพี่น้อง และเพื่อนๆทุกคน

นันทิพัฒน์ พิศุทธางกูร

พฤษภาคม 2556

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในช่วงไม่กี่ทศวรรษที่ผ่านมาความก้าวหน้าทางด้านเทคโนโลยีการประมวลผลเกิดขึ้นในอัตราที่ก้าวหน้าเป็นอย่างมาก นับตั้งแต่ปี พ.ศ. 2531 ถึงปี พ.ศ.2555 นี้ ความเร็วของไมโครโพรเซสเซอร์เพิ่มจาก 40 MHz เป็น 3.9 GHz ความก้าวหน้าของเทคโนโลยีทางด้านการประมวลผลดังกล่าว ส่งผลให้เทคโนโลยีด้านการพัฒนาโปรแกรมในสาขาวิชาต่างๆ เกิดขึ้นอย่างรวดเร็วตามไปด้วย โปรแกรมในปัจจุบันนี้มีแนวโน้มที่จะมีการใช้ทรัพยากรมากกว่าในอดีตอย่างมาก จนกระทั่งเทคโนโลยีไมโครโพรเซสเซอร์ตามไม่ทัน ในหมู่นักวิทยาศาสตร์ และนักวิศวกรที่ต้องการความเร็วในการประมวลผลสูง จึงใช้คอมพิวเตอร์แบบขนาน (Parallel Computer) หรือใช้การประมวลผลแบบขนาน (Parallel Processing) ซึ่งเป็นศาสตร์การเขียนโปรแกรมแบบหนึ่งที่ได้รับความนิยมเป็นอย่างสูงมาแก้ปัญหาเหล่านี้ นอกจากนี้แล้วเทคโนโลยีการประมวลผลแบบขนานยังสามารถลดเวลาในการประมวลผลข้อมูลที่มีขนาดใหญ่ ซึ่งปกติต้องใช้เวลานานในการคำนวณ ทำให้ได้ผลข้อมูลที่สามารถนำไปใช้ได้อย่างทันท่วงที

คำนิยามสำหรับการประมวลผลแบบขนาน คือ “การใช้เครื่องคอมพิวเตอร์ประมวลผลกลุ่มหนึ่งในการคำนวณ หรือแก้ปัญหาจากโจทย์เดียวกัน เพื่อให้ได้คำตอบที่เร็วขึ้นกว่าการคำนวณบนเครื่องประมวลผลเครื่องเดียว” [13] คำนิยามดังกล่าวได้แนวคิดมาจากการศึกษาธรรมชาติของการเขียนโปรแกรม ซึ่งพบว่าปัญหาส่วนใหญ่นั้น สามารถแบ่งเป็นปัญหาย่อยๆได้ และแต่ละปัญหาย่อยนั้นสามารถประมวลผลได้พร้อมๆกัน โดยอาศัยการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลหรือเครื่องประมวลผล ดังนั้นการประมวลผลแบบขนานจึงเปรียบเหมือนการทำงานแบบกลุ่ม ซึ่งต้องอาศัยทั้งประสิทธิภาพของสมาชิกในกลุ่ม และการสื่อสารที่ดีระหว่างสมาชิกในกลุ่ม

ปัจจุบันมีระบบคอมพิวเตอร์ชนิดหนึ่งที่เรียกว่า “คอมพิวเตอร์แบบขนาน” ซึ่งระบบคอมพิวเตอร์ชนิดนี้เป็นเทคโนโลยีสมัยใหม่ซึ่งมีการออกแบบทางด้านสถาปัตยกรรมระบบคอมพิวเตอร์และมีการศึกษาพัฒนาที่ต่อเนื่องอยู่ตลอดเวลา ระบบคอมพิวเตอร์ชนิดนี้ประกอบด้วยหน่วยประมวลผลหลายๆ หน่วยจำนวนมากรวมอยู่ภายในเครื่องเดียวกัน คอมพิวเตอร์แบบขนานมักจะถูกใช้ในการประมวลผลโปรแกรมที่เน้นการคำนวณเป็นหลัก (Compute – Intensive) ตัวอย่าง

ของโปรแกรมประยุกต์ (Application) ที่เหมาะสมสำหรับการใช้งานคอมพิวเตอร์แบบขนาน คือ การสร้างแบบจำลองทางวิทยาศาสตร์สำหรับงานวิจัยต่างๆ เช่น งานทางด้านการสร้างแบบจำลองดีเอ็นเอ และการค้นคว้าเกี่ยวกับยีนในเซลล์ของสิ่งมีชีวิต งานวิจัยด้านปรากฏการณ์ของพลังงาน (Quantum Phenomena) การศึกษาโครงสร้างและการเคลื่อนที่ของโมเลกุล การวิเคราะห์ทางด้านดาราศาสตร์ กลศาสตร์ เป็นต้น งานวิจัยด้านต่างๆ ดังที่กล่าวมาล้วนแล้วแต่ต้องใช้อัลกอริทึมแบบใหม่ ซึ่งมีการคำนวณ และจำนวนข้อมูลในปริมาณมาก กำลังในการประมวลผลระดับสูงเช่นนี้จำเป็นต้องอาศัยคอมพิวเตอร์แบบขนานในการทำงาน

งานวิจัยนี้ได้เสนอการใช้ขั้นตอนแบบขนานมาช่วยในเรื่องการคูณเมตริกซ์ (Matrix Multiplication) ที่มีขนาดใหญ่เนื่องจากเมตริกซ์เป็นทฤษฎีที่นำไปประยุกต์ใช้เป็นลำดับต้นๆในงานสำคัญหลายๆด้าน เช่น งานในสายคณิตศาสตร์และวิศวกรรมที่มีการวิเคราะห์หาคำตอบของสมการเชิงเส้นที่ประกอบด้วยตัวแปรจำนวนมาก ซึ่งงานลักษณะนี้อาจจะใช้เมตริกซ์มาช่วยหาคำตอบในเรื่องดังกล่าวได้ ดังนั้นถ้าสามารถลดระยะเวลาในการหาผลคูณเมตริกซ์ จะเป็นผลดีสำหรับการหาคำตอบในสมการดังกล่าวตามไปด้วย และถ้าเป็นกรณีที่เมตริกซ์ที่มีขนาดใหญ่มากๆก็ยังสามารถหาผลคูณได้ อีกตัวอย่างของงานทางด้านวิศวกรรมที่มีการประยุกต์ใช้เมตริกซ์ คือ การวิเคราะห์โครงสร้างของโครงถักเหล็ก (Steel Truss) ซึ่งสมการในงานประเภทนี้จะประกอบด้วยตัวแปรจำนวนมาก ทำให้ต้องใช้เวลามากในการหาคำตอบด้วยเช่นกัน

การหาผลคูณเมตริกซ์แบบขนานสำหรับวิทยานิพนธ์นี้มีการนำวิธีการไทลิงก์ (Tiling Technique) มาประยุกต์ใช้ในขั้นตอนการทรานสโพสเมตริกซ์ เพื่อที่จะได้ข้อมูลที่เหมาะสมก่อนที่จะทำการคูณเมตริกซ์ โดยนำเสนอวิธีการหาผลคูณเมตริกซ์ทรานสโพสแบบขนานทั้งหมด 3 แบบ ที่แตกต่างกันตามลักษณะการจัดแบ่งข้อมูล และการใช้ข้อมูลที่ซ้ำซ้อนกันของแต่ละหน่วยประมวลผล ดังนี้

1) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w
   (row-block partitioning matrix-transpose multiplication with replicated data)
2) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี cbmtm_w
   (checkerboard-block partitioning matrix-transpose multiplication with replicated data)
3) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w/o
   (row-block partitioning matrix-transpose multiplication withoutreplicated data)

การวัดประสิทธิภาพและการเปรียบเทียบวิธีต่างๆ ที่นำเสนอจะเปรียบเทียบวิธีที่กล่าวมาข้างต้นกับวิธีการคูณเมตริกซ์ทรานสโพสด้วยวิธีไทลิงก์ และวิธีแบบอนุกรม โดยวัดจากเวลาทั้งหมดที่ใช้ในการทรานสโพสเมตริกซ์ และคูณเมตริกซ์ (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speed Up) และประสิทธิภาพ (Efficiency)

1.2 วัตถุประสงค์ของการศึกษา

งานวิจัยนี้มีวัตถุประสงค์เพื่อศึกษาวิธีการหาผลคูณของเมตริกซ์แบบขนาน (parallel transpose of matrix multiplication) พร้อมทั้งนำเสนอวิธีการเพิ่มประสิทธิภาพของการหาผลคูณของเมตริกซ์แบบขนาน ดังนี้

1) ลดระยะเวลาในการหาผลคูณของเมตริกซ์แบบอนุกรมที่ใช้หน่วยประมวลผล หนึ่งหน่วย โดยการใช้หน่วยประมวลผลเพิ่มขึ้น และทำการประมวลผลแบบขนาน
2) ลดการใช้ข้อมูลในการหาผลคูณของเมตริกซ์แบบขนานที่ซ้ำซ้อนกันของแต่ละหน่วยประมวลผล
3) ทำการเปรียบเทียบประสิทธิภาพของการหาผลคูณของเมตริกซ์แบบขนาน เพื่อหาข้อดีและข้อจำกัดของวิธีต่างๆที่จะนำเสนอ

1.3 สมมติฐานการศึกษา

1) การหาผลคูณของเมตริกซ์แบบขนานโดยใช้จำนวนหน่วยประมวลผลเพิ่มขึ้นสำหรับการคูณเมตริกซ์ขนาดใหญ่เป็นการช่วยให้เวลาในการประมวลผลในการหาผลคูณของเมตริกซ์ลดลง
2) ในกรณีที่เมตริกซ์มีขนาดใหญ่มากๆ การหาผลคูณเมตริกซ์แบบขนานมีประสิทธิภาพที่ดีกว่าการหาผลคูณเมตริกซ์แบบอนุกรม
3) การทรานสโพสเมตริกซ์ที่สองก่อนที่จะมีการคูณเมตริกซ์ จะทำให้เวลาในการประมวลผลดีขึ้น เมื่อใช้กับมาตรฐาน MPI (MPI Standard)

1.4 ขอบเขตการวิจัย

วิทยานิพนธ์นี้มีขอบเขตของการวิจัย ดังต่อไปนี้

1) ทำการศึกษาการหาผลคูณเมตริกซ์ที่ทำการทรานสโพสแบบขนาน ที่มีวิธีแตกต่างกัน 3 วิธี วิธีแรกเป็นการลดระยะเวลาในการหาผลคูณของเมตริกซ์แบบอนุกรม ในกรณีที่เมตริกซ์มีขนาดใหญ่ๆ โดยใช้ข้อมูลที่ซ้ำซ้อนกันเล็กน้อยเพื่อที่จะมีการติดต่อสื่อสารระหว่างหน่วยประมวลผลให้น้อยที่สุด หรือไม่มีเลย วิธีต่อมาเป็นการลดการใช้ข้อมูลที่ซ้ำซ้อนกัน หรือไม่มีการใช้ข้อมูลที่ซ้ำซ้อนกันเลย แต่ยอมเสียเวลาในการติดต่อสื่อสารระหว่างหน่วยประมวลผลเพื่อทำให้มีการแลกเปลี่ยนข้อมูลระหว่างกัน
2) ทำการเขียนโปรแกรม โดยนำทฤษฎีและวิธีการที่ได้ทำการศึกษาในส่วนแรกมาพัฒนา โดยใช้ภาษา C (C Language) และมาตรฐานภาษาเอ็มพีไอ (MPI Standard) ที่สนับสนุนการเขียนโปรแกรมแบบขนาน เพื่อนำเสนอผลลัพธ์พร้อมทั้งวิเคราะห์ และสรุปผลการวิจัย

1.5 ขั้นตอนการศึกษา และการดำเนินงานวิจัย

วิทยานิพนธ์นี้มีขั้นตอนการศึกษา และการดำเนินงานวิจัย ดังนี้

1) ศึกษาขั้นตอนวิธีการทางคอมพิวเตอร์ (Computer algorithm) ของการหาผลคูณของเมตริกซ์แบบขนาน
2) ศึกษางานวิจัยที่เกี่ยวข้องกับการทรานสโพสเมตริกซ์ และการหาผลคูณของเมตริกซ์แบบขนาน
3) ทำการตั้งสมมติฐานโดยคาดว่าการเพิ่มหน่วยประมวลผล และใช้วิธีแบบขนานเข้ามาช่วยประมวลผลในการคูณเมตริกซ์จะทำให้เวลาในการหาผลคูณเมตริกซ์ลดลง โดยเฉพาะกรณีเมตริกซ์มีขนาดใหญ่รวมถึงการลดพื้นที่ในหน่วยความจำของแต่ละหน่วยประมวลผล ด้วยการลดใช้ข้อมูลที่ซ้ำซ้อนกัน และสรุปได้ว่าการหาผลคูณเมตริกซ์ทรานสโพสแบบขนานมีประสิทธิภาพมากกว่าวิธีการหาผลคูณเมตริกซ์แบบอนุกรมในกรณีที่เมตริกซ์มีขนาดใหญ่ๆ
4) นำเสนอวิธีการหาผลคูณของเมตริกซ์ทรานสโพสแบบขนานที่มีประสิทธิภาพ ดังที่ได้ตั้งสมมติฐานไว้ในข้อ3)

5) พัฒนาโปรแกรมที่นำเสนอโดยใช้โปรแกรมภาษา ซี (C Language) บนระบบปฏิบัติการลีนุกต์ (Linux) และตรงตามมาตรฐานภาษาเอ็มพีไอ (MPI Standard) ที่สนับสนุนการโปรแกรมแบบขนาน
6) วิเคราะห์ผลการทดลอง โดยการเปรียบเทียบประสิทธิภาพของการหาผลคูณของเมตริกซ์ จะประเมินผลจากเวลาที่ใช้ในการประมวลผล (Response Time) อัตราการเพิ่มของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency)
7) สรุปผลการทดลอง พร้อมเสนอแนวทางการพัฒนางานวิจัย
8) เขียนวิทยานิพนธ์

1.6 ประโยชน์ที่คาดว่าจะได้รับ

1) สามารถหาผลคูณของเมตริกซ์กรณีที่เมตริกซ์มีขนาดใหญ่ๆ โดยใช้เวลาน้อยลงใช้หน่วยความจำน้อยลง
2) นำมาใช้เป็นแนวทางในการพัฒนาโปรแกรมคอมพิวเตอร์แบบขนานเพื่อให้สามารถใช้งานได้สะดวกและเข้าใจได้ง่าย
3) นำไปประยุกต์ใช้กับการพัฒนาโปรแกรมด้านต่างๆ เช่นงานด้านการพยากรณ์อากาศ, งานด้านเว็บเซิร์ฟเวอร์, งานทางด้านธุรกิจ, คำนวณผลภาพถ่ายทางดาวเทียม และงานด้านกราฟิคดีไซน์เป็นต้น

บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 คอมพิวเตอร์แบบขนาน (Parallel Computer)

ระบบคอมพิวเตอร์ที่มีประสิทธิภาพสูง ในปัจจุบัน จะมีหน่วยประมวลผล (Processing Elements : PEs) รวมกันอยู่เป็นจำนวนมาก ซึ่งหน่วยประมวลผลเหล่านี้จะช่วยเพิ่มประสิทธิภาพในการประมวลผลที่มีขนาดใหญ่ๆ ระบบคอมพิวเตอร์ประเภทนี้จะถูกเรียกว่า คอมพิวเตอร์แบบขนาน (Parallel Computer) ในระบบคอมพิวเตอร์ดังกล่าว หน่วยประมวลผลทุกๆ หน่วยประมวลผลสามารถประมวลผลได้พร้อมๆกันไปทีละงานได้อย่างเต็มประสิทธิภาพ โดยระบบคอมพิวเตอร์แบบขนาน จะแตกต่างกันตามแต่ละสถาปัตยกรรม ซึ่งโดยทั่วไปสามารถจำแนกได้หลายรูปแบบ โดยมีรายละเอียดดังนี้

2.1.1 สถาปัตยกรรมคอมพิวเตอร์ (Computer Architecture)

การจัดแบ่งประเภทสถาปัตยกรรมคอมพิวเตอร์ที่นำเสนอโดย ไมเคิล เจ.ฟลินน์ ในปี 1972 หรือที่รู้จักอีกชื่อหนึ่งว่า Flynn's Taxonomy ถือเป็นวิธีการจัดแบ่งที่ได้รับความนิยมสูงและใช้กันอย่างแพร่หลาย ฟลินน์ได้จัดแบ่งระบบคอมพิวเตอร์ออกเป็น 4 ประเภท ตามลักษณะของความสามารถในการประมวลผลแบบขนาน ทั้งในมิติของคำสั่ง (Instructions) และข้อมูล (Data) สถาปัตยกรรมทั้ง 4 แบบคือ

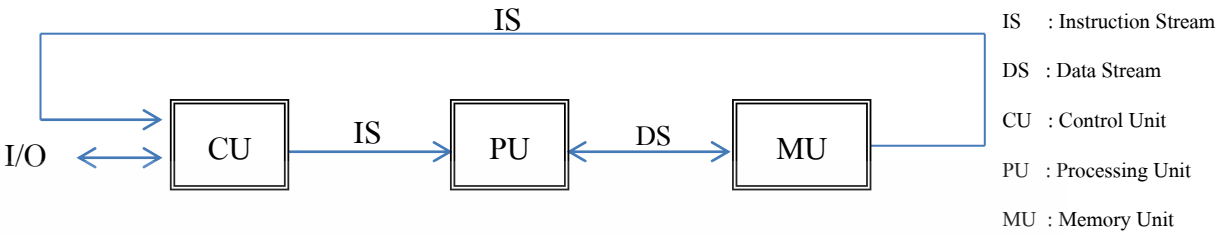
1) สถาปัตยกรรมคอมพิวเตอร์แบบ SISD

(Single Instruction stream over Single Data stream)

SISD เป็นต้นแบบของระบบคอมพิวเตอร์ที่ใช้โดยทั่วไป (Conventional Sequential Machines) ที่ประกอบด้วย หน่วยควบคุม (CU), หน่วยประมวลผล (PU) และหน่วยความจำ (MU) อย่างละหนึ่งหน่วยดังที่แสดงในรูปที่ 2.1

ในการประมวลผลบนระบบคอมพิวเตอร์ SISD นี้ หน่วยควบคุม CU จะนำคำสั่งการคำนวณ (IS : Instruction Stream) เข้าทีละคำสั่งจากหน่วยความจำ MU และส่งไปให้หน่วยประมวลผล PU ทำการประมวลผล โดยหน่วยประมวลผล PU สามารถอ่านข้อมูลจากหน่วยความจำ MU ได้ทีละค่า

สถาปัตยกรรมลักษณะนี้ไม่จัดเป็นแบบขนาน เนื่องจากระบบคอมพิวเตอร์จะประมวลผลคำสั่งทีละคำสั่งบนข้อมูลทีละชุด ตัวอย่างของคอมพิวเตอร์แบบ SISD คือ เครื่องคอมพิวเตอร์ส่วนบุคคล (Personal Computer) ทั่วๆไปหรือเครื่องเมนเฟรมในสมัยเก่า



รูปที่ 2.1 สถาปัตยกรรมคอมพิวเตอร์แบบ SISD

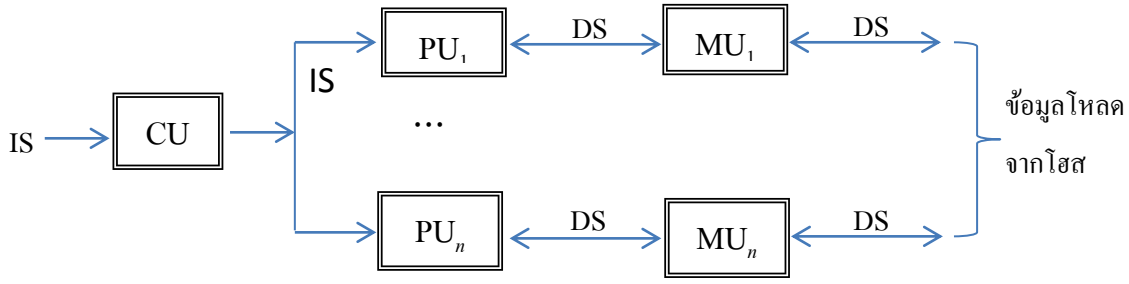
2) สถาปัตยกรรมคอมพิวเตอร์แบบ SIMD

(Single Instruction stream over Multiple Data stream)

ระบบคอมพิวเตอร์แบบ SIMD เป็นต้นแบบของระบบคอมพิวเตอร์แบบขนาน (Parallel Machine) ซึ่งเป็นแบบที่กระจายหน่วยความจำ (Distributed Memory) หรือเป็นระบบคอมพิวเตอร์แบบเวคเตอร์ (Vector Computer) การออกแบบระบบคอมพิวเตอร์ชนิดนี้จะประกอบด้วย หน่วยควบคุม (CU) หนึ่งหน่วย, หน่วยประมวลผล (PU) *n* หน่วย และหน่วยความจำ (MUs) *n* หน่วย (แสดงในรูป 2.2)

ในการประมวลผลนั้น หน่วยควบคุม CU จะนำเข้าคำสั่งทีละคำสั่งจากโฮสคอมพิวเตอร์ (Host computer) และถ้าคำสั่งนั้นเป็นแบบเวคเตอร์ (Vector Instruction) จะถูกส่งไปให้หน่วยประมวลผลที่มี *n*หน่วย เพื่อทำการประมวลผลพร้อมๆกัน โดยแต่ละหน่วยประมวลผลสามารถอ่านข้อมูลแต่ละค่าจากหน่วยความจำได้พร้อมๆกัน

สถาปัตยกรรมที่สามารถประมวลผลชุดคำสั่งชุดเดียวบนข้อมูลหลายชุดพร้อมกันได้ ลักษณะนี้จะเหมาะสมกับโปรแกรมประยุกต์แบบขนานจำนวนมาก ตัวอย่างเช่น โพสเซสเซอร์อาร์เรย์ (Processor Array)



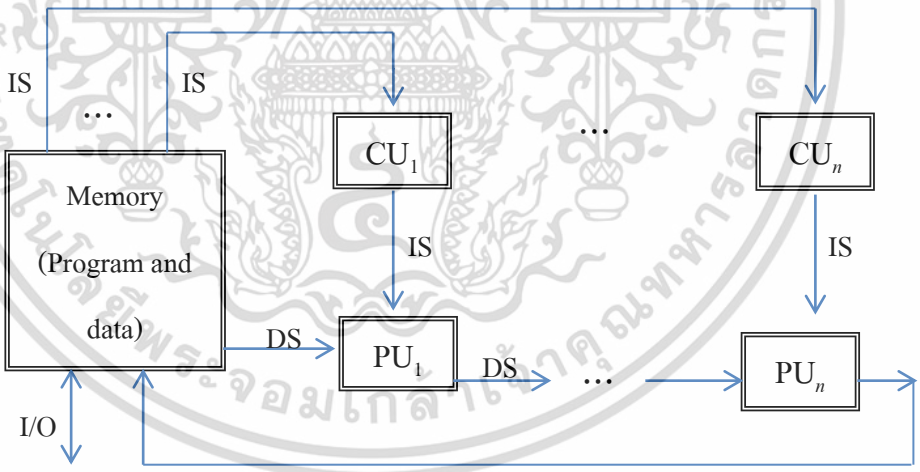
รูปที่ 2.2 สถาปัตยกรรมคอมพิวเตอร์แบบ SIMD

3) สถาปัตยกรรมคอมพิวเตอร์แบบ MISD

(Multiple Instruction stream over Data stream)

ระบบคอมพิวเตอร์แบบ MISD นี้ เป็นระบบคอมพิวเตอร์แบบขนานที่ถูกออกแบบมาสำหรับการประมวลผลแบบขนานที่เหมาะกับงานเฉพาะ โดยประกอบด้วยหน่วยควบคุม (CUs) *n* หน่วย, หน่วยประมวลผล (PUs) *n* หน่วย และหน่วยความจำร่วม (MU) หนึ่งหน่วย (แสดงในรูปที่ 2.3)

ในการประมวลผลโปรแกรมคอมพิวเตอร์บนระบบ MISD นี้ แต่ละหน่วยประมวลผล PU จะมีหน่วยควบคุม CU ของตัวเอง ทำหน้าที่โหลดคำสั่งที่ต่างกันได้พร้อมๆ กันเพื่อประมวลผลพร้อมๆ กัน ในระบบนี้แต่ละหน่วยประมวลผลจะได้รับข้อมูลชุดเดียวกันจากหน่วยความจำซึ่งเป็นกรณีที่เกิดขึ้นน้อย เนื่องจากการประมวลผลคำสั่งหลายๆชุดให้มีประสิทธิภาพมักต้องการข้อมูลหลายชุด อย่างไรก็ดี สถาปัตยกรรมแบบนี้เหมาะสมกับโปรแกรมประยุกต์ที่ต้องการความน่าเชื่อถือสูง เนื่องจากระบบมีการประมวลผลสำรองเผื่อในกรณีที่เกิดความผิดพลาด ตัวอย่างคือ ซิสทอริดอาร์เรย์ (Systolic Array)



รูปที่ 2.3 สถาปัตยกรรมคอมพิวเตอร์แบบ MISD

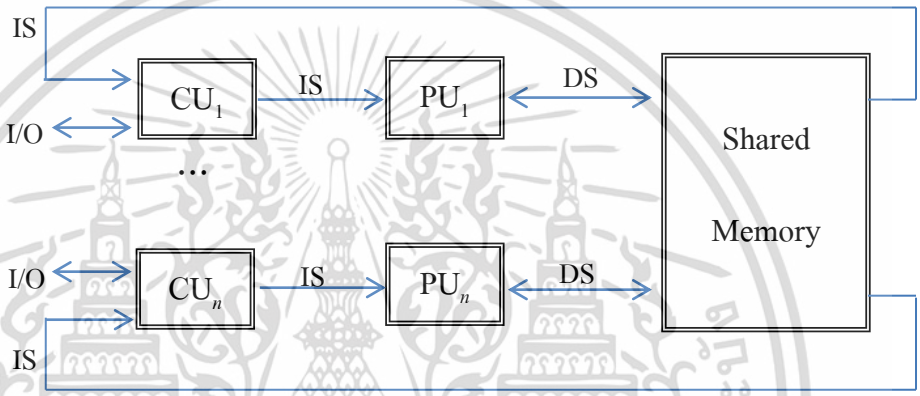
4) สถาปัตยกรรมของคอมพิวเตอร์แบบ MIMD

(Multiple Instruction stream over Multiple Data stream)

ระบบคอมพิวเตอร์แบบ MIMD เป็นระบบคอมพิวเตอร์แบบขนานทั่วๆไป ที่ถูกออกแบบสำหรับการประมวลผลแบบขนานที่เหมาะกับงานแบบขนานทั่วๆไป (General-Purpose

Computations) ที่ประกอบด้วยหน่วยควบคุม (CUs) n หน่วย, หน่วยประมวลผล (PUs) n หน่วย และหน่วยความจำร่วม (Shared Memory) ดังแสดงในรูปที่ 2.4

ในการประมวลผลโปรแกรมบนระบบ MIMD นี้ หน่วยควบคุม CU แต่ละหน่วยจะนำเข้าคำสั่งทีละคำสั่งไปพร้อมๆกัน และส่งให้หน่วยประมวลผล PU เพื่อทำการประมวลผลไปพร้อมๆกัน ในกรณีที่คำสั่งที่ถูกประมวลผลพร้อมๆกัน อาจจะแตกต่างกันได้ ซึ่งแต่ละหน่วยประมวลผลสามารถอ่านข้อมูลจากหน่วยความจำร่วมได้พร้อมๆกันระบบคอมพิวเตอร์แบบกระจาย (Distributed System) ทั่วๆไปในปัจจุบันมักจะสร้างตามสถาปัตยกรรมชนิดนี้



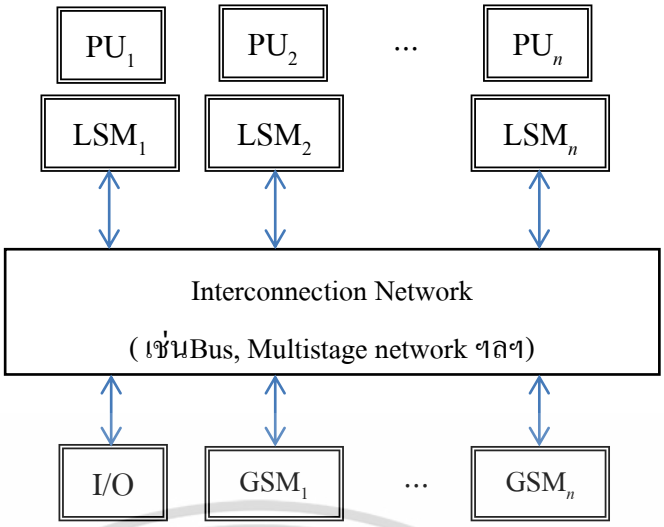
รูปที่ 2.4 สถาปัตยกรรมของระบบคอมพิวเตอร์แบบ MIMD

2.1.2 สถาปัตยกรรมคอมพิวเตอร์แบบขนาน (Parallel Computer Architecture)

สถาปัตยกรรมคอมพิวเตอร์แบบขนานสามารถแบ่งได้ 2 ประเภท ตามประเภทโครงสร้างของหน่วยความจำ [12]โดยมีรายละเอียดดังนี้

1) สถาปัตยกรรมแบบขนานที่ใช้หน่วยความจำร่วมกัน (Shared-Memory Parallel Architecture)

สถาปัตยกรรมแบบนี้จะถูกออกแบบมาให้มีการใช้หน่วยความจำร่วมกัน โดยที่หน่วยประมวลผลทั้งหมดของระบบสามารถเข้าถึงข้อมูลในหน่วยความจำของระบบได้ทุกที่พร้อมกัน ทำให้การพัฒนาประสิทธิภาพทำได้สะดวกมากขึ้น แต่อย่างไรก็ตามสถาปัตยกรรมแบบนี้ยังมีการจำกัดขนาดของเส้นทางที่ติดต่อสื่อสารของหน่วยประมวลผลที่จะเข้าถึงข้อมูลในหน่วยความจำอยู่ จากปัญหาดังกล่าวจึงได้มีการพยายามที่จะแก้ไขปัญหานี้ โดยให้แต่ละหน่วยประมวลผลมีหน่วยความจำส่วนตัว (Local Memory) และมีหน่วยความจำกลาง (Global Memory) ที่ใช้เก็บข้อมูล

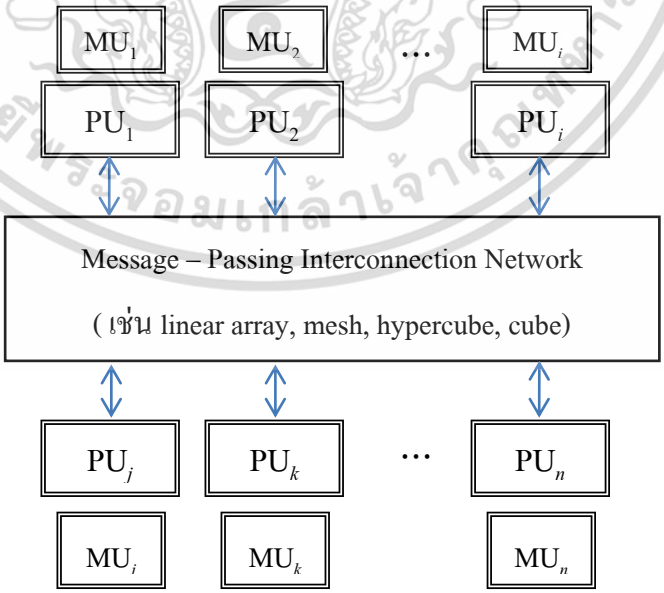


รูปที่ 2.5 สถาปัตยกรรมแบบขนานที่ใช้หน่วยความจำร่วมกัน

2) สถาปัตยกรรมแบบขนานที่มีการกระจายหน่วยความจำ

(Distributed-Memory Parallel Architecture)

สถาปัตยกรรมชนิดนี้ออกแบบมาให้ใช้หน่วยความจำแบบกระจาย คือ หน่วยประมวลผลแต่ละหน่วยจะมีความจำส่วนตัว (Local Memory) ที่ไม่สามารถใช้ร่วมกันได้ ซึ่งการติดต่อสื่อสารกันของหน่วยประมวลผลนั้นจะติดต่อสื่อสารกันผ่านระบบส่งผ่านข้อความ (Message-Passing) โดยจะทำการรับส่งข้อมูล ผ่านไลบรารี (Libraries) ของเอ็มพีไอ (Message Passing Interface)[11] หรือพีวีเอ็ม (Parallel Virtual Machine) เป็นต้น



รูปที่ 2.6 สถาปัตยกรรมแบบขนานที่มีการกระจายหน่วยความจำ

2.1.3 ชนิดของคอมพิวเตอร์แบบขนาน (Types of Parallel Computers)

ชนิดของคอมพิวเตอร์แบบขนานที่ถูกพัฒนาขึ้นมาใช้งานจริง นั้นมีหลายชนิด ซึ่งบางชนิดนั้นมีแนวคิดคล้ายกับวิธีการของ Flynn ที่เคยได้เสนอมาก่อนหน้านี้ ดังนี้

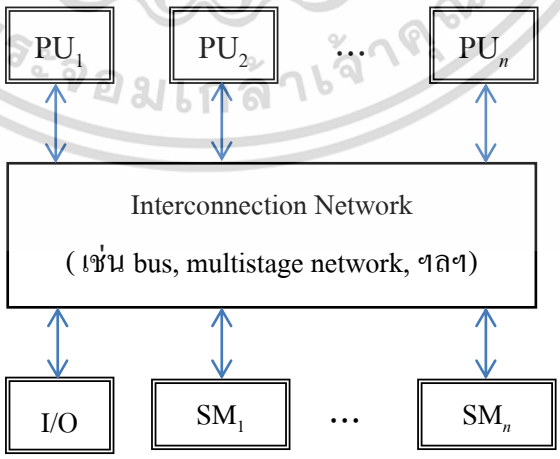
1) มัลติโพรเซสเซอร์ (Multiprocessors)

ระบบคอมพิวเตอร์แบบขนานชนิดนี้ จะมีลักษณะของการประมวลผลแบบขนาน ที่เรียกว่า คำสั่งขนาน (Instruction Parallelism) เหมือนกับระบบคอมพิวเตอร์แบบ MIMD ของ Flynn โดยส่วนมากจะเป็นแบบมัลติโพรเซสเซอร์ที่มีหน่วยความจำร่วม (Shared-Memory Multiprocessors) ซึ่งเน้นการประมวลผลหลายๆ คำสั่งที่อาจแตกต่างกันได้พร้อมๆกัน

ระบบคอมพิวเตอร์แบบขนานชนิดนี้ สามารถแบ่งเป็นโมเดลของคอมพิวเตอร์แบบขนานที่แตกต่างกันได้อีก 2 แบบ คือ

แบบที่ 1 : โมเดลคอมพิวเตอร์แบบ Uniform Memory Access (UMA)

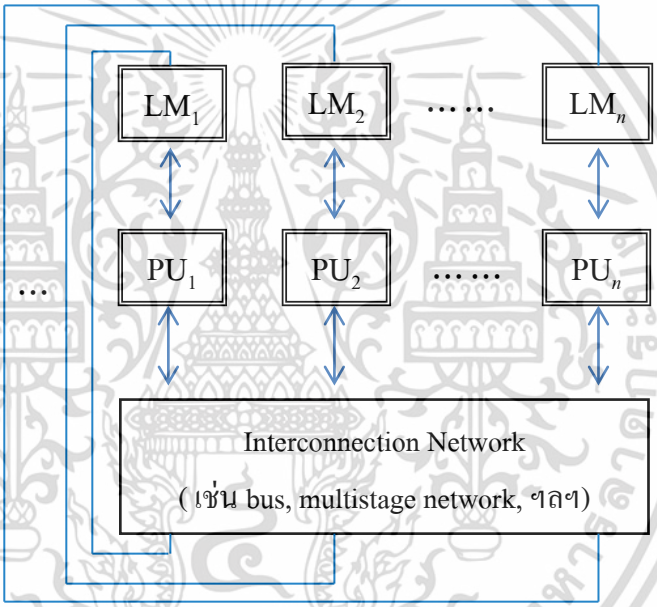
โมเดลคอมพิวเตอร์แบบ UMA (แสดงในรูป 2.7) เป็นมัลติโพรเซสเซอร์ (Multiprocessors) ชนิดที่ใช้หน่วยความจำร่วม (Shared Memory) แบบรวมศูนย์ โมเดล UMA นี้ ทุกๆหน่วยประมวลผล (P_i) สามารถเข้าถึงข้อมูลในหน่วยความจำ ผ่านเครือข่ายการเชื่อมต่อ (Interconnection Network) ด้วยเวลาที่เท่าๆกัน เนื่องจากมีการใช้ทรัพยากรร่วมกันสูง ระบบคอมพิวเตอร์แบบนี้จึงถูกเรียกว่า “Tightly Coupled System”



รูปที่ 2.7 แสดงองค์ประกอบของระบบคอมพิวเตอร์แบบ UMA

แบบที่ 2 : โมเดลคอมพิวเตอร์แบบ Non-Uniform Memory Access (NUMA)

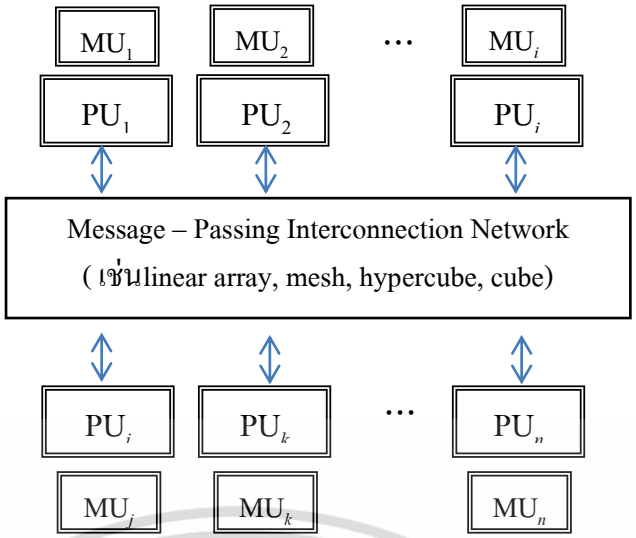
โมเดลคอมพิวเตอร์แบบ NUMA (แสดงในรูปที่2.8) หน่วยประมวลผลแต่ละหน่วยประมวลผลสามารถเข้าถึงข้อมูลที่อยู่ในหน่วยความจำที่ใกล้ หรือที่อยู่ในหน่วยประมวลผลของตัวเอง (Local Access) ได้โดยตรง เพราะว่าหน่วยความจำเป็นแบบหน่วยความจำร่วมที่กระจายอยู่ในทุกๆ หน่วยประมวลผล (Local Memory) แต่ใช้การกำหนดตำแหน่งแบบร่วม (Global Address Space) เพื่อให้หน่วยประมวลผลสามารถเข้าถึงข้อมูลได้ทุกตำแหน่งที่อยู่เครื่องใดก็ได้ในระบบ การเข้าถึงข้อมูลในหน่วยความจำประเภทดังกล่าวจะสามารถทำได้เร็วกว่าการเข้าถึงหน่วยความจำที่เป็นของหน่วยประมวลผลอื่นๆ หรือที่เรียกหน่วยความจำแบบรีโมต (Remote Memory)



รูปที่ 2.8 แสดงองค์ประกอบของระบบคอมพิวเตอร์แบบ NUMA

2) มัลติคอมพิวเตอร์ (Multicomputers)

ระบบคอมพิวเตอร์แบบขนานชนิดนี้ (รูปที่ 2.9) ประกอบด้วยหน่วยความจำแบบกระจาย (Distributed Memory) ซึ่งไม่สามารถใช้ร่วมกันได้ เพราะเป็นหน่วยความจำของแต่ละหน่วยประมวลผล ดังนั้นหน่วยประมวลผลแต่ละหน่วยสามารถเข้าถึงข้อมูลเฉพาะที่อยู่ในหน่วยความจำของตัวเองเท่านั้น และไม่สามารถเข้าถึงข้อมูลของหน่วยประมวลผลอื่น (No Remote Memory Access) หรือเรียกอีกอย่างว่า โมเดลคอมพิวเตอร์แบบ NORMA

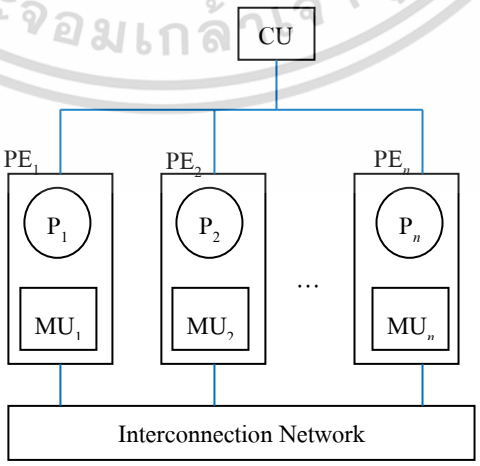


รูปที่ 2.9 แสดงองค์ประกอบของระบบคอมพิวเตอร์แบบมัลติคอมพิวเตอร์

3) อะเรย์โพรเซสเซอร์ (Array Processors)

ระบบคอมพิวเตอร์แบบขนานชนิดนี้ (รูปที่2.10) เป็นระบบคอมพิวเตอร์ที่เน้นการประมวลผลข้อมูลหลายๆ ค่าต่อหนึ่งคำสั่งได้พร้อมๆกัน หรือเป็นแบบข้อมูลขนาน (Data Parallelism) โดยระบบคอมพิวเตอร์ชนิดนี้จะมีหน่วยควบคุม (Control Unit: CU) หนึ่งหน่วย, หน่วยประมวลผล (Processing Elements) *n* หน่วย และประกอบด้วยคำสั่ง 2 แบบ คือ

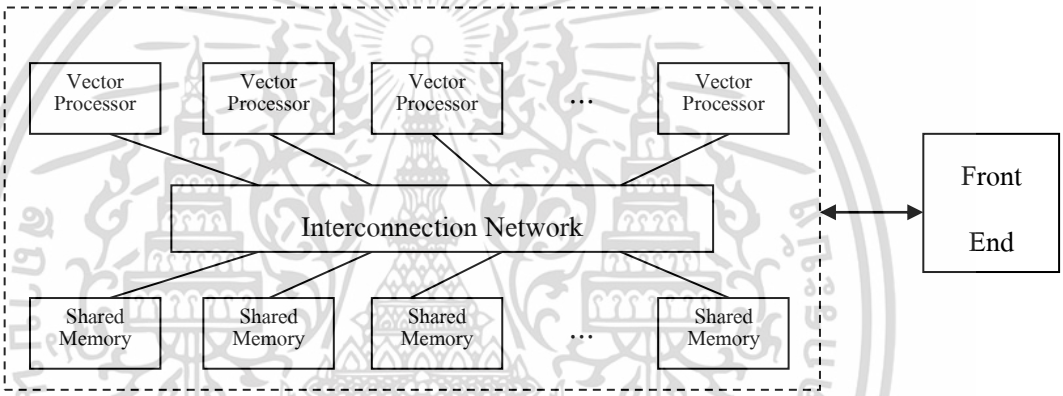
- คำสั่งแบบ C (C Instructions) เป็นคำสั่งแบบลำดับ ซึ่งจะประมวลผลด้วยหน่วยควบคุม (CU)
- คำสั่งแบบ I (I Instructions) เป็นคำสั่งแบบขนาน โดยจะถูกส่งแบบกระจาย (Broadcast) จากหน่วยควบคุม ไปยังทุกๆ หน่วยประมวลผล (PEs)



รูปที่ 2.10 แสดงองค์ประกอบของระบบคอมพิวเตอร์แบบอะเรย์โพรเซสเซอร์

4) ไพพ์ลายน์ และเวคเตอร์โพรเซสเซอร์ (Pipeline and Vector Processors)

ระบบคอมพิวเตอร์แบบไพพ์ลายน์ และเวคเตอร์โพรเซสเซอร์ จะเรียกอีกอย่างว่า ซุปเปอร์คอมพิวเตอร์ (Supercomputers) ในปัจจุบันประกอบด้วยหน่วยประมวลผลจำนวนหลายหน่วยประมวลผลทำงานร่วมกัน หน่วยประมวลผลทั้งหมดนี้จะเชื่อมต่อกับหน่วยความจำกลางดังรูปที่ 2.11 ซึ่งแต่ละหน่วยมีประสิทธิภาพมาก การเรียกใช้กำลังประมวลผลของระบบลักษณะนี้จะทำได้โดยผ่านเครื่องฟรอนต์เอนด์(Front End Computer) เท่านั้น ผู้ใช้ทั่วไปไม่สามารถติดต่อกับหน่วยประมวลผลกลางได้โดยตรง การออกแบบระบบนี้จะเน้นความสามารถในการประมวลผลแบบเวคเตอร์ ดังนั้นระบบประเภทนี้จะออกแบบมาเพื่อสนับสนุนการประมวลผลแบบขนานในระดับฮาร์ดแวร์ มากกว่าที่จะออกแบบมาเพื่อการประมวลผลแบบขนานบนซอฟแวร์ (Software)



รูปที่ 2.11 สถาปัตยกรรมแบบเวกเตอร์โพรเซสเซอร์แบบขนาน (PVP)

ข้อมูลที่อ่านเข้ามาในเวกเตอร์โปรเซสเซอร์ จะเป็นคิวแบบ FIFO (First-In First-Out) โดยชุดคำสั่งจะประกอบด้วยคำสั่งที่สามารถ โหลดเวกเตอร์รีจีสเตอร์จากตำแหน่งในหน่วยความจำ, ทำโอเปอร์เรชั่นต่างๆ ในเวกเตอร์รีจีสเตอร์ และเก็บค่าข้อมูลจากเวกเตอร์รีจีสเตอร์กลับลงหน่วยความจำ ประเภทของเวกเตอร์โปรเซสเซอร์ มีดังนี้

- Memory-to-memory Vector Processor

  เวกเตอร์โปรเซสเซอร์ประเภทนี้มีความสามารถในการประมวลผลเวกเตอร์ที่ยาวมากๆ มีข้อเสียคือมีค่าเสียเวลา (Startup Time) สูง

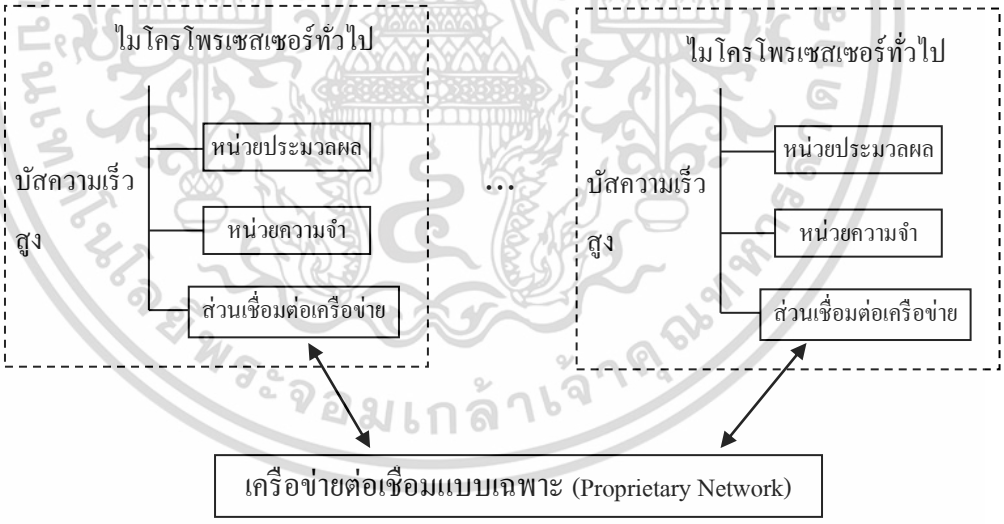
- Register-to-register Vector Processor

  เวกเตอร์โปรเซสเซอร์ประเภทนี้จะแบ่งเวกเตอร์ออกเป็นชิ้นย่อยๆ และทำงานคำนวณเป็นชุดๆ

ซุปเปอร์คอมพิวเตอร์ในยุคแรก (ช่วง พ.ศ. 2520 – 2540) ส่วนมากจะสร้างจากหน่วยประมวลผลแบบเวกเตอร์ ในปัจจุบันการใช้งานหน่วยประมวลผลประเภทนี้มีน้อยลงเนื่องจากความก้าวหน้าของเทคโนโลยีโพรเซสเซอร์ ทำให้ความเร็วของหน่วยประมวลผลแบบทั่วไป (Commodity Processor) สามารถรองรับการคำนวณจำนวนมากได้ด้วยราคาที่ต่ำกว่า

5) แมสซิฟลีพาราลเลลโพรเซสเซอร์ (Massively Parallel Processors)

แมสซิฟลีพาราลเลลโพรเซสเซอร์ (MPP) คือ ระบบที่มีหน่วยประมวลผลในระดับ 100 หน่วยขึ้นไปจนถึงหลายพันหน่วย สถาปัตยกรรม MPP จัดเป็นแบบ NUMA กล่าวคือ การเข้าถึงข้อมูลในหน่วยความจำแต่ละโมดูล อาจจะใช้เวลาไม่เท่ากัน การที่มีหน่วยประมวลผลจำนวนมากอยู่ในระดับเดียวกันส่งผลให้หน่วยความจำต้องเป็นแบบกระจาย (Distributed Memory) ระบบขนาดใหญ่ลักษณะนี้สามารถสร้างได้จากการเชื่อมต่อไมโครโพรเซสเซอร์ที่มีในท้องตลาด หรือที่ออกแบบเฉพาะซึ่งมีทั้งหน่วยความจำและส่วนเชื่อมต่อเครือข่ายอยู่ในตัว ไมโครโพรเซสเซอร์เหล่านี้สามารถทำงานขนานไปในระบบคอมพิวเตอร์เดียวกันโดยสื่อสารผ่านทางเครือข่ายความเร็วสูง ดังแสดงในรูปที่ 2.12



รูปที่ 2.12 สถาปัตยกรรมแบบแมสซีฟลีพาราลเลลโพรเซสเซอร์ (MPP)

การพัฒนาโปรแกรมประยุกต์สำหรับเครื่อง MPP นั้นหน่วยประมวลผลจะไม่สามารถสื่อสารแลกเปลี่ยนข้อมูลกันได้ผ่านการเขียน และอ่านตัวแปลชุดเดียวกันกับ PVP เนื่องจากหน่วยความจำเป็นแบบกระจาย ดังนั้นการสื่อสารจึงต้องอาศัยการส่งผ่านข้อมูล (Message Passing)

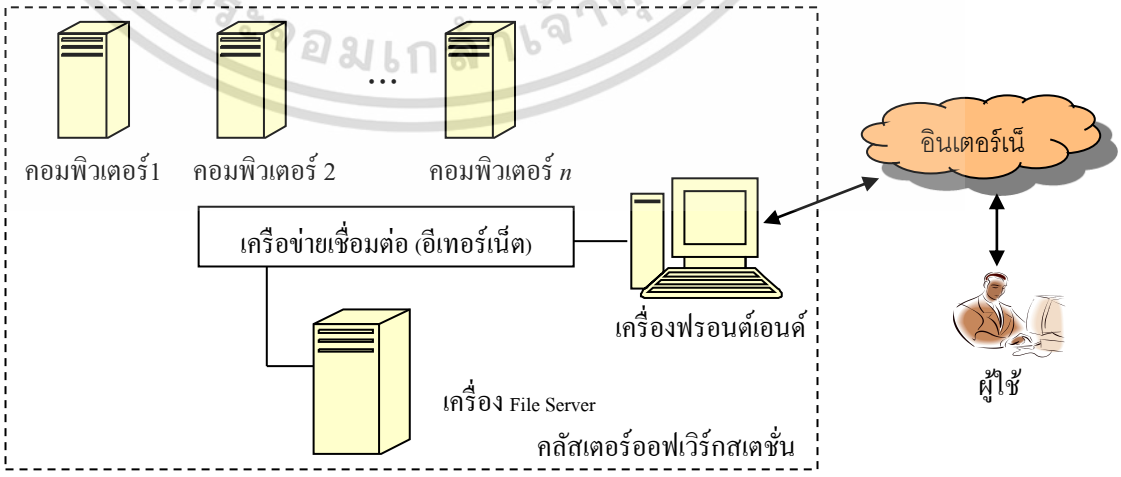
6) คลัสเตอร์ออฟเวิร์กสเตชัน (Cluster of Workstation)

คลัสเตอร์ออฟเวิร์กสเตชัน (COW) จัดเป็นสถาปัตยกรรมแบบ NUMA เช่นเดียวกับระบบคอมพิวเตอร์แบบ MPP กล่าวคือ หน่วยความจำจะกระจายกันอยู่ และเวลาที่ใช้ในการเข้าถึงหน่วยความจำแต่ละโมดูลอาจไม่เท่ากัน ความแตกต่างของ COW เมื่อเทียบกันกับ MPP คือ หน่วยประมวลผลแต่ละหน่วยของสถาปัตยกรรมแบบนี้จะเป็นเครื่องคอมพิวเตอร์ที่สมบูรณ์ โดยมีอุปกรณ์อินพุต (Input) และเอาต์พุต (Output) รวมอยู่ด้วย ไม่ได้มีเฉพาะส่วนประมวลผล หน่วยความจำ และส่วนต่อเชื่อมกับเครือข่ายเท่านั้น การเชื่อมต่อกันระหว่างเครื่องคอมพิวเตอร์จะใช้เครือข่ายเชื่อมต่อราคาต่ำที่มีอยู่ในท้องตลาดทั่วไป (Commodity Network) การแลกเปลี่ยนข้อมูลระหว่างเครื่องคอมพิวเตอร์ทำได้โดยส่งผ่านข้อความ (Message Passing) เท่านั้น การสร้างระบบคอมพิวเตอร์แบบ COW จะมีค่าใช้จ่ายต่ำจึงสามารถขยายขนาดได้ง่ายเมื่อเทียบกับการสร้างเครื่องซุปเปอร์คอมพิวเตอร์

สถาปัตยกรรมแบบ COW สามารถแบ่งออกได้เป็น 2 ชนิด คือ

- ระบบคลัสเตอร์คอมพิวเตอร์แบบไม่สมมาตร (Asymmetrical Cluster)

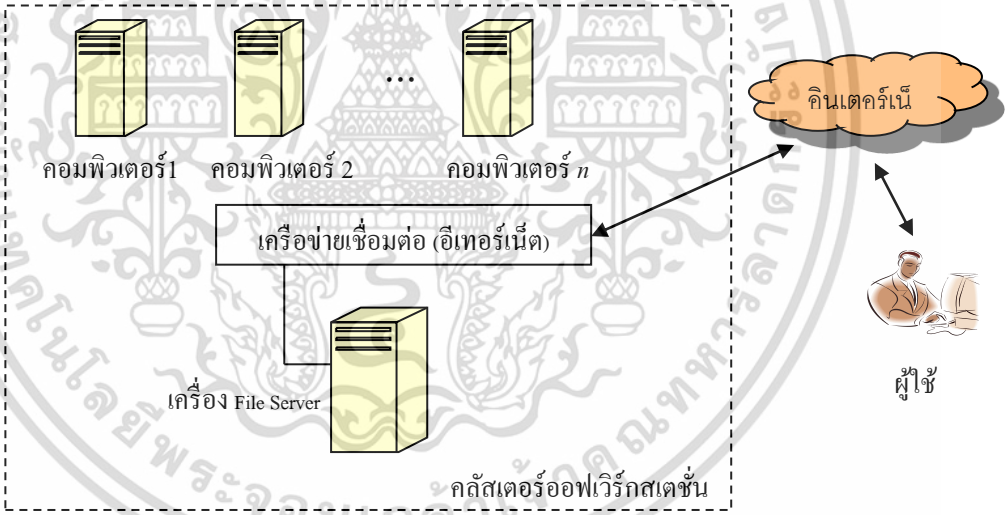
ระบบคอมพิวเตอร์นี้จะประกอบด้วยเครื่องคอมพิวเตอร์สำหรับประมวลผล เครื่องสำหรับเก็บข้อมูล และเครื่องคอมพิวเตอร์แบบฟรอนต์เอนด์ สำหรับติดต่อกับผู้ใช้ แสดงในรูปที่ 2.13 จากรูป ผู้ใช้จะสามารถใช้งานระบบคลัสเตอร์ได้ผ่านเครื่องฟรอนต์เอนด์ และอุปกรณ์ อินพุต/เอาต์พุตต่างๆ ก็จะต่อกับเครื่องฟรอนต์เอนด์เท่านั้น เครื่องที่เหลือจะทำหน้าที่ในการประมวลผลโปรแกรมแบบขนานเพียงอย่างเดียวนอกจากนี้ ตัวอย่างของระบบปฏิบัติการพื้นฐานดังกล่าว คือ ระบบ Vertex



รูปที่ 2.13 สถาปัตยกรรมคลัสเตอร์คอมพิวเตอร์แบบไม่สมมาตร

- ระบบคลัสเตอร์คอมพิวเตอร์แบบสมมาตร (Sysmetrical Cluster)

สำหรับคลัสเตอร์ประเภทนี้ เครื่องคอมพิวเตอร์ทุกเครื่องจะถูกควบคุมด้วยระบบปฏิบัติการเดียวกัน และทำหน้าที่เหมือนกันในการประมวลผล ผู้ใช้สามารถที่จะล๊อกอินเข้าไปที่เครื่องใดในระบบก็ได้เพื่อเขียนคอมไพล์ หรือ ประมวลผลโปรแกรม ลักษณะของคลัสเตอร์ดังกล่าวทำให้ยากต่อการกระจายงานให้สมดุลระหว่างเครื่องคอมพิวเตอร์ เนื่องจากผู้ใช้สามารถเลือกเครื่องที่ต้องการใช้งานได้โดยตรง รูปที่ 2.14 แสดงสถาปัตยกรรมระบบคอมพิวเตอร์คลัสเตอร์แบบสมมาตร โดยโพรเซสเซอร์ของเครื่องคอมพิวเตอร์แต่ละเครื่องไม่ได้อุทิศให้กับการประมวลผลแบบขนานเท่านั้น แต่ยังถูกใช้งานในการทำงานอื่น ๆ ได้อีก สถาปัตยกรรมแบบนี้สามารถพัฒนาต่อให้เป็นห้องปฏิบัติการคอมพิวเตอร์ทั่วไปได้ โดยการต่อหน้าจอและคีย์บอร์ด เข้ากับเครื่องคอมพิวเตอร์ทุกตัวโดยตรง และปล่อยให้ผู้ใช้ใช้งานเครื่องได้อย่างเต็มที่ โดยการประมวลผลแบบขนานถูกคำนวณอยู่เบื้องหลัง

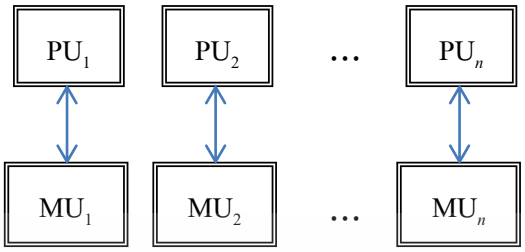


รูปที่ 2.14 สถาปัตยกรรมระบบคอมพิวเตอร์คลัสเตอร์แบบสมมาตร

2.1.4 แบบจำลอง PRAM (Parallel Random Access Machine Model)

แบบจำลอง RPAM จะมีค่าเวลาในการติดต่อระหว่างหน่วยประมวลผลเท่ากับศูนย์ ดังนั้นจึงเรียก แบบจำลองชนิดนี้ว่า แบบจำลองการต่อแบบขนานในอุดมคติ (Idealized Parallel System) ซึ่งโมเดลนี้จะถูกใช้ในการเริ่มต้นสำหรับการออกแบบขั้นตอนวิธีแบบขนานในขั้นแรก เนื่องจาก

ในโมเดลแบบพีแรมจะสมมติให้เวลาในการติดต่อสื่อสารมีค่าเป็นศูนย์ และสมมติให้หน่วยประมวลผลสามารถอ่านหรือเขียนค่าจากหน่วยความจำคนละที่พร้อมๆกันได้ ดังรูปที่ 2.15



รูปที่ 2.15 แบบจำลอง PRAM

แบบจำลอง PRAM สามารถแยกได้ 4 ประเภท ตามลักษณะการอ่านหรือเขียน ณ จุดเดียวกันโดยหลายๆ หน่วยประมวลผล เพราะแบบจำลอง PRAM จะต้องมีวิธีการในการจัดการในกรณีที่มีหน่วยประมวลผลหลายๆ หน่วย ที่ต้องการอ่านหรือเขียนข้อมูลบนหน่วยความจำ ณ จุดเดียวกันดังนี้

1) แบบจำลอง PRAM แบบ EREW (Exclusive Read Exclusive Write)

PRAM แบบ EREW เป็นโมเดลที่มีข้อจำกัดมากที่สุด เพราะว่าอนุญาตให้เพียงหนึ่งหน่วยประมวลผลสามารถอ่านหรือเขียนข้อมูลในหน่วยความจำ ณ จุดใดๆ เดียวกันในหนึ่งรอบเวลา ถ้าหน่วยประมวลผลอื่นๆต้องการอ่านหรือเขียนข้อมูลต้องรอรอบการทำงานถัดไป

2) แบบจำลอง PRAM แบบ CREW (Concurrent Read Exclusive Write)

PRAM แบบ CREW เป็นโมเดลที่มีความยืดหยุ่นสำหรับการอ่านข้อมูลจากหน่วยความจำ (Memory Unit) คือ จะอนุญาตให้ทุกหน่วยประมวลผลสามารถอ่านข้อมูลในหน่วยความจำ ณ จุดๆ เดียวกันในหนึ่งรอบการทำงาน แต่ไม่สามารถเขียนข้อมูล ณ จุดๆ เดียวกันได้พร้อมกันในหนึ่งรอบการทำงานถ้าหน่วยประมวลผลอื่นๆต้องการเขียนข้อมูลต้องรอรอบการทำงานถัดไป

3) แบบจำลอง PRAM แบบ ERCW (Exclusive Read Concurrent Write)

PRAM แบบ ERCW เป็นโมเดลที่มีความยืดหยุ่นสำหรับการเขียนข้อมูลคือจะอนุญาตให้ทุกหน่วยประมวลผลสามารถเขียนข้อมูลในหน่วยความจำ ณ จุดใดๆ เดียวกันในหนึ่งรอบการทำงาน แต่ไม่สามารถอ่านข้อมูล ณ จุดๆ เดียวกันได้พร้อมกันได้ ถ้าหน่วยประมวลผลอื่นๆ ต้องการอ่านข้อมูลต้องรอรอบการทำงานถัดไป

4) แบบจำลอง PRAM แบบ CRCW (Concurrent Read Concurrent Write)

PRAM แบบ CRCW เป็นโมเดลที่มีความยืดหยุ่นสำหรับการอ่านและเขียนข้อมูลคือ จะอนุญาตให้ทุกหน่วยประมวลผลสามารถอ่านหรือเขียนข้อมูลในหน่วยความจำ ณ จุดใดๆ เดียวกันในหนึ่งรอบการทำงานได้พร้อมๆกัน

2.2 การวัดประสิทธิภาพของการประมวลผลแบบขนาน

การวัดประสิทธิภาพขั้นตอนของการประมวลผลแบบขนานนั้นจะประเมินผลจาก 3 การประเมิน คือ เวลาที่ใช้ในการประมวลผล (Response Time), อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) เพราะว่าขั้นตอนการประมวลผลแบบขนาน มีความซับซ้อนกว่าขั้นตอนของวิธีการแบบอนุกรม (Sequential algorithm)

2.2.1 เวลาทั้งหมดที่ใช้ในการประมวลผล(Response Time)

การวัดเวลาที่ใช้ในการประมวลผลแบบขนานจะวัดโดยการจับเวลา ตั้งแต่เริ่มประมวลผลจนกระทั่งสิ้นสุดการประมวลผล หรือได้คำตอบที่ต้องการ ซึ่งเวลาที่วัดได้จะรวมเวลาที่ใช้ในการติดต่อสื่อสารด้วย

เวลาที่ใช้ในการประมวลผลแบบขนานสามารถคำนวณได้ดังสมการ

$$T_p = T_{computation} + T_{communication}$$

เมื่อ T_p คือ เวลาที่ใช้ในการประมวลผลด้วย p หน่วยประมวลผล

$T_{computation}$ คือ เวลาที่ใช้ในการประมวลผลซึ่งไม่รวมเวลาที่ใช้ในการติดต่อสื่อสาร

$T_{communication}$ คือ เวลาที่ใช้ในการสื่อสารระหว่างหน่วยประมวลผล

2.2.2 อัตราการเพิ่มของความเร็ว (Speedup)

วิธีวัดประสิทธิภาพของการประมวลผลขนานที่ใช้กันทั่วไปในปัจจุบัน คือ การหาค่าสปีดอัป(Speedup) ซึ่งค่านี้จะบ่งบอกถึงอัตราส่วนของประสิทธิภาพที่เพิ่มมากขึ้น จากการเขียนโปรแกรมแบบขนาน เมื่อเทียบกันกับโปรแกรมแบบตามลำดับ

อัตราการเพิ่มของความเร็วแบบขนานสามารถคำนวณได้ดังสมการ

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

เมื่อ S_p คือ อัตราการเพิ่มความเร็ว โดยใช้ p หน่วยประมวลผลซึ่งมีค่าสูงสุด คือ p

T_s คือ เวลาที่ใช้ในการประมวลผลโดยใช้หนึ่งหน่วยประมวลผล

T_p คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วย p หน่วยประมวลผล

2.2.3 ประสิทธิภาพ (Efficiency)

ค่าสปีดอัปสามารถนำมาใช้ในการคำนวณค่าประสิทธิภาพการประมวลผล (Efficiency) ได้ โดยค่าดังกล่าววัดได้จากสัดส่วนของเวลาที่หน่วยประมวลผลถูกใช้งานคำนวณที่เป็นประโยชน์เทียบกับเวลาว่างที่ต้องหยุดรอการแลกเปลี่ยนข้อมูล หรือโอเวอร์เฮดประเภทอื่นๆ หรือกล่าวอีกนัยหนึ่งก็คือ ค่าประสิทธิภาพการประมวลผลจะบอกถึงอัตราการใช้ประโยชน์ของหน่วยประมวลผล (Processor Utilization)

ประสิทธิภาพแบบขนานสามารถคำนวณได้ดังสมการ

$$E_p = \frac{S_p}{p} \leq 1$$

เมื่อ E_p คือ ประสิทธิภาพสูงสุดของระบบมีค่าสูงสุดคือ 1

S_p คือ อัตราการเพิ่มความเร็วที่คำนวณได้จากสมการหัวข้ออัตราการเพิ่มของความเร็ว

p คือ จำนวนหน่วยประมวลผลที่ใช้ในระบบ

2.3 การออกแบบอัลกอริทึม และเทคโนโลยีการพัฒนาโปรแกรมแบบขนาน

2.3.1 การออกแบบอัลกอริทึมแบบขนาน

ในหัวข้อนี้จะเป็นหลักการออกแบบอัลกอริทึมแบบขนานที่เหาะสมเพื่อเพิ่มประสิทธิภาพในการประมวลผล เนื้อหาในหัวข้อนี้จึงนำเสนอหลักการออกแบบโดยอ้างอิงจากวิธีการที่ได้รับ

ความยอมรับกันอย่างแพร่หลายในปัจจุบัน คือ วิธีการออกแบบของ เอียน ฟอสเตอร์ (Ian Foster) ซึ่งได้กำหนดหลักการสำหรับการพัฒนาอัลกอริทึมแบบขนานโดยมุ่งเน้นการทำงานบนสถาปัตยกรรมหน่วยความจำแบบกระจาย

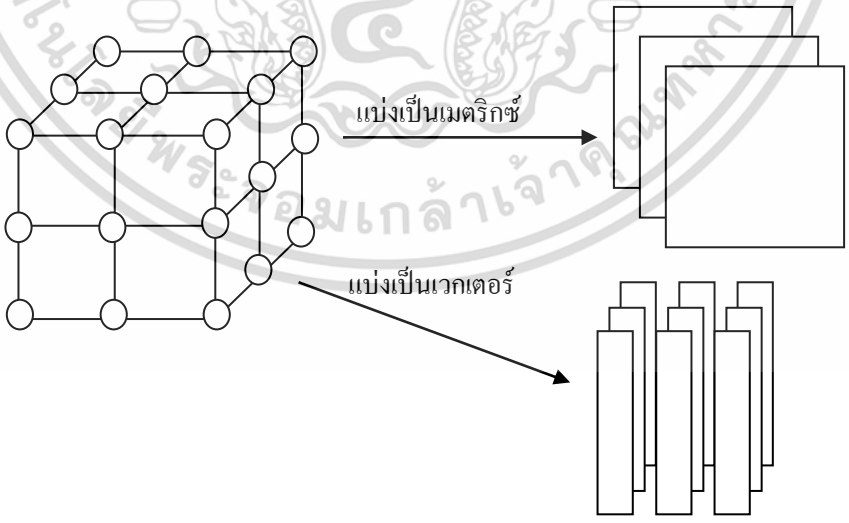
การออกแบบของฟอสเตอร์ ผู้พัฒนาโปรแกรมแบบขนานจะต้องคำนึงถึงขั้นตอนต่างๆ ในการออกแบบอัลกอริทึมดังต่อไปนี้ การแบ่งงาน (Partitioning), การสื่อสารระหว่างงาน (Communication), การจับงานย่อยรวมเป็นกลุ่ม (Agglomeration) และ การมอบหมายงาน (Mapping)

2.3.1.1 การแบ่งงาน (Partitioning)

เมื่อได้รับโจทย์ปัญหา และเริ่มต้นออกแบบอัลกอริทึม จะพบว่ามีวิธีการแบ่งโจทย์ปัญหาได้หลายรูปแบบเพื่อการประมวลผลแบบขนาน โดยการแบ่งงานสามารถทำได้ 2 รูปแบบหลักๆ คือ

1) การแบ่งงานเชิงข้อมูล (Data Decomposition)

การแบ่งงานเชิงข้อมูล คือ การแบ่งข้อมูลหรือโครงสร้างข้อมูลจากปัญหาออกเป็นส่วนๆ ซึ่งสามารถนำไปประมวลผลพร้อมกันได้ จากนั้นจึงหาวิธีที่จะเชื่อมโยงการคำนวณเข้ากับข้อมูลแต่ละส่วนนั้น วิธีการแบ่งแบบนี้จะเหมาะสมกับปัญหาเชิงวิทยาศาสตร์ ที่ต้องการประมวลผลข้อมูลจำนวนมาก

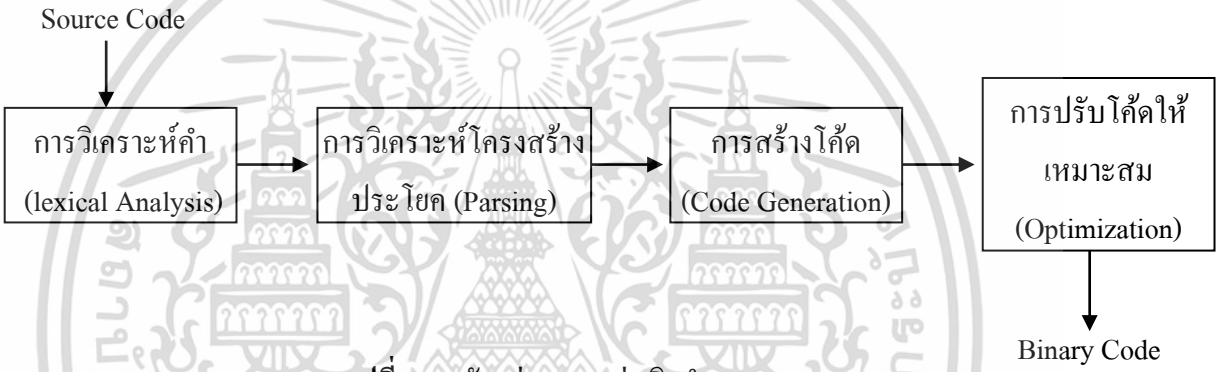


รูปที่ 2.16 ตัวอย่างการแบ่งงานเชิงข้อมูล

รูปที่ 2.16 แสดงตัวอย่างการแบ่งโครงสร้างข้อมูลเมตริกซ์แบบสามมิติ ซึ่งเป็นโครงสร้างข้อมูลที่ใช้มากในปัจจุบัน การแบ่งทำได้หลายแบบ เช่น แบ่งออกเป็นกลุ่มของเมตริกซ์ 2 มิติ จะได้งานย่อยทั้งสิ้น 3 ชิ้น หากแบ่งโครงสร้างข้อมูลออกเป็นกลุ่มเวกเตอร์ก็จะได้งานย่อย 9 ชิ้น ดังรูป

2) การแบ่งงานเชิงคำนวณ (Functional Decomposition)

จะมีลักษณะตรงข้ามกับการแบ่งเชิงข้อมูล คือ จะทำการแบ่งการคำนวณจากโจทย์ปัญหาออกเป็นส่วนย่อย ที่สามารถประมวลผลได้พร้อมกัน จากนั้นจึงค่อยหาวิธีเชื่อมโยงข้อมูลเข้ากับการคำนวณแต่ละส่วน วิธีนี้เหมาะสมกับกรณีที่ปัญหาเน้นที่อัลกอริทึม หรือฟังก์ชันการทำงานมากกว่าข้อมูล ตัวอย่างเช่น การทำงานของคอมไพเลอร์



รูปที่ 2.17 ตัวอย่างการแบ่งเชิงคำนวณ

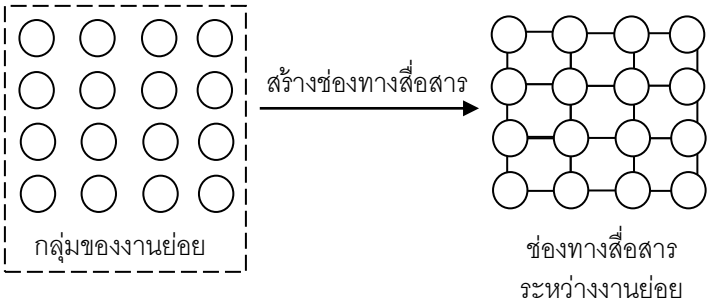
จากรูปที่ 2.17 การทำงานของคอมไพเลอร์แบ่งออกเป็น 4 ส่วนซึ่งสามารถทำงานขนานกันได้ โดยที่แต่ละส่วนจะประมวลผลบนข้อมูล หรือคำสั่งที่ต่างกันในลักษณะของไพพ์ไลน์

2.3.1.2 การสื่อสารระหว่างงาน (Communication)

การสื่อสารระหว่างโพรเซสในการประมวลผลแบบขนานแบ่งได้เป็น 2 ประเภทหลัก ดังนี้

1) ในกรณีที่งานย่อยจำนวนน้อยต้องการแลกเปลี่ยนข้อมูลระหว่างกัน เพื่อใช้ข้อมูลประกอบการคำนวณเราสามารถสร้างช่องทางสื่อสารเฉพาะสำหรับงานย่อยได้ เรียกว่า การสื่อสารแบบโลคอล (Local Communication)

2) หากงานย่อยจำนวนมาก หรือเกือบทั้งหมดต้องทำการแลกเปลี่ยนข้อมูลระหว่างกันเพื่อการคำนวณร่วม เช่น การหาผลบวกของข้อมูลที่กระจายอยู่ในพื้นที่หน่วยความจำของงานย่อยต่างๆ ในกรณีนี้ เราต้องสร้างช่องทางการสื่อสารแบบโกลบอล (Global Communication) ดังแสดงในรูป 2.18



รูปที่ 2.18 การออกแบบช่องทางการสื่อสาร

2.3.1.3 การจับงานย่อยรวมเป็นกลุ่ม (Agglomeration)

ในบางกรณีอัลกอริทึมแบบขนานที่ได้มาจากการออกแบบใน 2 ขั้นตอนแรกนั้นอาจมีประสิทธิภาพไม่สูงนัก เนื่องจากการออกแบบในช่วงแรกมักจะมุ่งเน้นการแบ่งงานให้ได้จำนวนมากเพื่อที่จะใช้ประโยชน์จากเครื่องคอมพิวเตอร์แบบขนานอย่างเต็มที่ ดังนั้นในหลายกรณีก่อนที่จะจัดสรรงานให้กับหน่วยประมวลผล เราอาจตัดสินใจจับงานย่อยรวมเป็นกลุ่ม เพื่อลดเวลาที่ต้องใช้ในการสื่อสาร และสร้างงานย่อย ซึ่งอาจเพิ่มประสิทธิภาพในการประมวลผล และทำให้เขียนโปรแกรมง่ายขึ้นอีกด้วย การรวมกลุ่มงานของงานย่อย มีเกณฑ์ดังนี้

- การจับกลุ่มงานย่อยที่เหมาะสมควรจะลดปริมาณการสื่อสารระหว่างหน่วยประมวลผลของโปรแกรมแบบขนาน
- การคำนวณซ้ำซ้อนที่มักจะเกิดขึ้นเมื่อมีการจับกลุ่มงานย่อย ต้องเสียเวลาน้อยว่าการสื่อสารที่ขจัดออกไปจากการจับกลุ่ม ไม่เช่นนั้นก็แสดงว่าการจับกลุ่มงานทำให้ประสิทธิภาพโดยรวมตกลง
- การจับกลุ่มงานย่อยไม่ควรทำให้เกิดความซ้ำซ้อนของข้อมูลจำนวนมาก
- กลุ่มงานต่างๆ ควรจะมีปริมาณการคำนวณและการสื่อสารใกล้เคียงกัน
- จำนวนกลุ่มงานควรเพิ่มขึ้นตามขนาดของปัญหา
- จำนวนกลุ่มงานควรมีน้อยที่สุดเท่าที่จะเป็นไปได้ แต่มีเพียงพอสำหรับจำนวนหน่วยประมวลผล

2.3.1.4 การมอบหมายงาน (Mapping)

การมอบหมายงานให้กับหน่วยประมวลผลอย่างเหมาะสมนั้นเป็นขั้นตอนที่มีความสำคัญอย่างยิ่งต่อประสิทธิภาพของระบบโดยรวม ในกรณีที่คอมพิวเตอร์มีหน่วยประมวลผลที่ใช้หน่วยความจำร่วมกันมากกว่า 1 หน่วย การมอบหมายงานจะเกิดขึ้นโดยอัตโนมัติภายใต้การดูแลของระบบปฏิบัติการ (Operating System) แต่สำหรับระบบคอมพิวเตอร์ที่มีหน่วยความจำแบบกระจายนั้น การมอบหมายงานไปยังหน่วยประมวลผลต่างๆ เป็นหน้าที่รับผิดชอบของผู้พัฒนาโปรแกรม การมอบหมายงานอย่างเหมาะสมจะส่งผลให้โปรแกรมสามารถใช้ประโยชน์จากหน่วยประมวลผลทั้งหมดได้อย่างเต็มที่ (Maximum Utilization)

การเลือกวิธีมอบหมางานแบ่งออกได้เป็น 2 กลุ่มคือ

1) เมื่อทราบจำนวนงานที่แน่นอนก่อนเริ่มการประมวลผล เราสามารถรวมกลุ่มงานย่อยให้เหลือ 1 กลุ่มสำหรับแต่ละหน่วยประมวลผล โดยพยายามลดปริมาณการสื่อสารให้เหลือน้อยๆ

2) เมื่อทราบจำนวนงานสามารถเปลี่ยนแปลงได้ขณะประมวลผล และงานมีการสื่อสารกันอย่างสม่ำเสมอ เทคนิคการจัดสรรงานให้สมดุลระหว่างการประมวลผล (Dynamic Load Balancing) สามารถนำมาใช้เพื่อเพิ่มประสิทธิภาพของการคำนวณ

2.3.2 เทคโนโลยีการพัฒนาโปรแกรมแบบขนาน

เนื่องจากสถาปัตยกรรมของเครื่องประมวลผลแบบขนานในปัจจุบันมีหลายรูปแบบ เทคโนโลยีการพัฒนาโปรแกรมแบบขนานจึงต้องมีหลากหลาย เพื่อตอบสนองต่อความต้องการของเครื่องแต่ละประเภท เทคโนโลยีสามารถแบ่งออกได้เป็น 3 ประเภทดังนี้

1) การใช้คอมไพเลอร์พิเศษ

วิธีการใช้คอมไพเลอร์พิเศษนี้ สามารถแปลงโปรแกรมแบบตามลำดับ (Sequential Program) ให้กลายเป็นแบบขนาน (Parallel Program) ผู้พัฒนาโปรแกรมไม่จำเป็นต้องเรียนรู้เทคนิคที่เกี่ยวข้องกับการเขียนโปรแกรมแบบขนาน เนื่องจากคอมไพเลอร์จะสามารถทำการสร้างโค้ดไบนารี่แบบขนานได้โดยอัตโนมัติจากโปรแกรมอนุกรมทั่วๆไป คอมไพเลอร์พิเศษนี้จะทำการปรับโครงสร้าง (Restructuring) ของโปรแกรม โดยการวิเคราะห์การขึ้นอยู่กับกันและกัน (Dependence Analysis) ของโปรแกรมแต่ละส่วนก่อนที่จะสร้างโค้ดไบนารี

ข้อดีของการใช้คอมไพเลอร์แบบนี้ คือ ผู้พัฒนาโปรแกรมสามารถใช้โค้ดเก่าที่มีอยู่แล้วมาประมวลผลแบบขนานได้เลยโดยไม่ต้องเขียนโค้ดใหม่ จึงเป็นการประหยัดเวลา ค่าใช้จ่าย อย่างไรก็

ดี ในท้องตลาดปัจจุบันยังไม่มีคอมไพเลอร์ที่สามารถสร้างโค้ดแบบขนานที่มีประสิทธิภาพเพียงพอ ส่งผลให้วิธีการนี้ไม่เป็นที่นิยมนัก

2) การใช้ภาษาใหม่

ภาษาใหม่ถูกออกแบบมาเพื่อการเขียนโปรแกรมแบบขนานโดยเฉพาะ ซึ่งชุดคำสั่งสามารถทำงานบนเวกเตอร์ หรือเมตริกซ์ของข้อมูลได้ การสร้างภาษาใหม่สามารถทำได้ 2 วิธีด้วยกันคือ

- สร้างภาษาใหม่จากจุดเริ่มต้นโดยไม่อ้างอิงภาษาเก่าใดๆ ซึ่งจะต้องมีการสร้างคอมไพเลอร์สำหรับภาษาใหม่โดยเฉพาะ ตัวอย่างเช่น ภาษา OCCAM พัฒนาโดยบริษัท INMOS จำกัด
- สร้างภาษาโดยการขยายผลจากภาษาเก่าที่มีอยู่แล้ว โดยการเพิ่มส่วนคำที่ใช้ผูกประโยคแบบขนาน (Parallel Construct) ซึ่งจะใช้ในการระบุส่วนของโปรแกรมที่ต้องการประมวลผลแบบขนาน ยกตัวอย่างเช่น ภาษา FORTRAN 90 และ C

ข้อดีของการใช้ภาษาใหม่ คือ ผู้พัฒนาโปรแกรมสามารถระบุส่วนที่ต้องการประมวลผลแบบขนานได้เอง ซึ่งจะมีประสิทธิภาพกว่าการให้คอมไพเลอร์ตัดสินใจ อย่างไรก็ดี ภาษาใหม่อาจไม่ได้รับการยอมรับอย่างกว้างขวางจากผู้พัฒนาโปรแกรม ทำให้ไม่สามารถพัฒนาต่อไปเป็นมาตรฐานของการเขียนโปรแกรมแบบขนานได้

3) การใช้ไลบารี่

การใช้ไลบารี่เป็นวิธีที่ได้รับความนิยมมากที่สุดในปัจจุบัน เนื่องจากผู้พัฒนาโปรแกรมไม่จำเป็นต้องเรียนรู้ภาษาใหม่ แต่สามารถเขียนโปรแกรมแบบขนานได้โดยการเรียก ใช้ฟังก์ชันต่างๆ ที่สร้างไว้ในไลบารี่ตามองค์ประกอบของภาษาเก่า เช่น ภาษา C หรือ ภาษา JAVA

การเขียนโปรแกรมแบบขนานต่างจากแบบลำดับคือ โปรแกรมต้องมีการสร้างโพรเซสสำหรับประมวลผลมากว่า 1 โพรเซส ซึ่งทำงานไปพร้อมๆกัน ในไลบารี่จึงต้องมีชุดฟังก์ชันสำหรับการติดต่อสื่อสารระหว่างโพรเซสอีกด้วย ตัวอย่างของไลบารี่ที่ถือว่าเป็นมาตรฐานในปัจจุบันคือ MPI (Message Passing Interface) ซึ่งใช้ในการพัฒนาโปรแกรมสำหรับสถาปัตยกรรมแบบหน่วยความจำกระจาย (Distributed Memory) นอกจากนี้ไลบารี่ OpenMP ยังได้รับความนิยมเป็นอย่างมากสำหรับใช้คู่กับสถาปัตยกรรมแบบหน่วยความจำร่วม (Shared Memory)

2.3.3 การเขียนโปรแกรมขนานแบบส่งผ่านข้อความ โดยใช้ MPI

การพัฒนาโปรแกรมแบบขนานอย่างชัดเจน (Explicit Parallel Model) เป็นที่รู้จักกันในหมู่นักพัฒนาโปรแกรมมานานแล้ว เทคนิคการพัฒนาโปรแกรมซึ่งเป็นที่ยอมรับกันอย่างกว้างขวางและยังคงใช้งานจนถึงทุกวันนี้ คือ การใช้ไลบารี่สำหรับพัฒนาโปรแกรมขนานแบบส่งผ่านข้อความ หรือ MPI (Message Passing Interface)

ไลบารี่ MPI สามารถใช้ได้กับการพัฒนาโปรแกรมด้วยภาษา C, FORTRAN และ JAVA โปรแกรมที่พัฒนาด้วย MPI มักจะนิยมใช้ประมวลผลบนระบบคอมพิวเตอร์แบบขนานที่มีหน่วยความจำแบบกระจาย ซึ่งแต่ละหน่วยประมวลผลจะเชื่อมต่อกันด้วยระบบเครือข่าย ดังแสดงในรูปที่ 2.19 อย่างไรก็ตาม โปรแกรมที่พัฒนาด้วย MPI สามารถประมวลผลบนระบบคอมพิวเตอร์แบบอื่นๆ ได้โดยใช้โค้ดเดียวกัน

รูปที่ 2.19 รูปแบบของระบบคอมพิวเตอร์ที่ไลบารี่ MPI นิยมนำไปใช้งาน

เนื่องจากไลบารี่ MPI มีฟังก์ชันให้เลือกใช้งานเป็นจำนวนมาก ซึ่งอาจทำให้เริ่มการเรียนรู้เทคนิคพัฒนาโปรแกรมทำได้ยาก หัวข้อนี้จึงอธิบายถึงฟังก์ชันกลุ่มหนึ่งที่สามารถใช้ในการเขียนโปรแกรมได้เกือบทุกรูปแบบของการติดต่อสื่อสาร และง่ายต่อการทำความเข้าใจ กลุ่มของฟังก์ชันมีดังนี้

1) ฟังก์ชัน MPI_Init และ MPI_Finalize

ฟังก์ชันของไลบารี่ MPI จะเขียนขึ้นต้นด้วย “MPI_” เสมอ และก่อนที่จะเรียกใช้ได้จะต้องมีการเรียกฟังก์ชัน MPI_Init เพื่อประกาศตัวแปรสิ่งแวดล้อม และจองพื้นที่ในหน่วยความจำสำหรับให้ฟังก์ชันอื่นๆ ใน MPI ใช้งาน จากนั้นเมื่อการทำงานของฟังก์ชัน MPI สิ้นสุด ควรมีการเรียก

ฟังก์ชัน MPI_Finalizeเพื่อคืนพื้นที่ของตัวแปรต่างๆที่ได้จองไว้ รูปแบบของทั้งสองฟังก์ชันมีลักษณะดังนี้

```
intMPI_Init(
                int*     argc     //in and out
                char**   argv[]   //in and out
)
intMPI_Finallize(void)
```

2) ฟังก์ชัน MPI_Comm_rank และ MPI_Comm_size

ฟังก์ชัน MPI_Comm_size สามารถถูกเรียกใช้เพื่อค้นหาว่ามีโพรเซสถูกสร้างอยู่ในกลุ่มเดียวกันทั้งสิ้นกี่โพรเซส ตัวอย่างเช่นในกรณีที่ระบบคอมพิวเตอร์ที่ใช้ในการประมวลผลมีทั้งหมด 5 เครื่อง โดยกำหนดให้แต่ละเครื่องมี 1 โพรเซส ฟังก์ชัน MPI_Comm_size จะส่งผลที่มีค่าเท่ากับ 5 กลับมา ส่วนฟังก์ชันMPI_Comm_rank จะถูกเรียกใช้เมื่อต้องการทราบลำดับของโพรเซสที่ถูกสร้างในกลุ่ม ในกรณีตัวอย่าง ฟังก์ชัน MPI_Comm_rank จะส่งค่ากลับมาเป็นจำนวนเต็มที่มีค่าอยู่ระหว่าง 0 ถึง 4 ซึ่งแต่ละโพรเซสจะได้รับค่ามาไม่เหมือนกัน

ฟังก์ชัน MPI_Comm_rank และ MPI_Comm_size มีลักษณะดังต่อไปนี้

```
voidMPI_Comm_size(
        MPI_Comm    comm                //in
        int*        number_of_process   //out
)

voidMPI_Comm_rank(
        MPI_Comm    comm                //in
        int*        rank                //out
)
```

3) ฟังก์ชัน MPI_Send และ MPI_Recv

ฟังก์ชัน MPI_Send เป็นฟังก์ชันที่ใช้ส่งข้อความ โดยข้อความที่ว่านี้ก็ คือ ตัวแปรหรือกลุ่มของตัวแปร แต่มีข้อแม้ว่าตัวแปรที่ใช้ส่งข้อความได้จะต้องเป็นตัวแปรที่มีการจองพื้นที่ในหน่วยความจำแบบต่อเนื่อง ฟังก์ชันสำหรับ MPI_Send มีลักษณะดังนี้

```
intMPI_Send(
        void*           message,        //in
        int             count,          //in
        MPI_Datatype    datatype,       //in
        int             dest,           //in
        int             tag,            //in
        MPI_Comm        comm            //in
)
```

ฟังก์ชัน MPI_Recv เป็นฟังก์ชันที่ใช้ในการรับข้อความซึ่งมักต้องปรากฏคู่กับฟังก์ชัน MPI_Send เสมอ พารามิเตอร์ของทั้งสองจะแตกต่างกันอยู่ 2 ประการ ประการแรกคือ ฟังก์ชัน MPI_Recv ต้องระบุว่า โพรเซส rank ใดเป็นตัวที่ส่งข้อความมาให้โดยระบุในพารามิเตอร์ source และประการที่สองคือ ฟังก์ชัน MPI_Recv จะเพิ่มตัวแปร MPI_Status เข้าไปเป็นพารามิเตอร์ตัวสุดท้าย โดยพารามิเตอร์ตัวนี้จะเป็นตัวแสดงสถานะของการรับข้อความ

```
intMPI_Recv(
        void*           message,        //in
        int             count,          //in
        MPI_Datatype    datatype,       //in
        int             dest,           //in
        int             tag,            //in
      MPI_Status*       status          //out
      MPI_Comm          comm,           //in
)
```

2.4 หลักการแบ่งชุดข้อมูล (Data Partitioning Methods)

การแบ่งชุดข้อมูลของข้อมูลแบบเมตริกซ์ มี 2 วิธี คือ การแบ่งชุดข้อมูลแบบเป็นส่วนๆ (Strip Partitioning) และการแบ่งชุดข้อมูลแบบตาราง (Checkerboard Partitioning)

2.4.1 การแบ่งชุดข้อมูลแบบเป็นส่วนๆ (Strip Partitioning)

วิธีการนี้ทำโดยแบ่งชุดข้อมูลเป็นส่วนๆ ตามแถว (Row) หรือหลัก (Column) ที่ติดกันโดยสามารถแบ่งย่อยได้เป็น 4 ประเภท

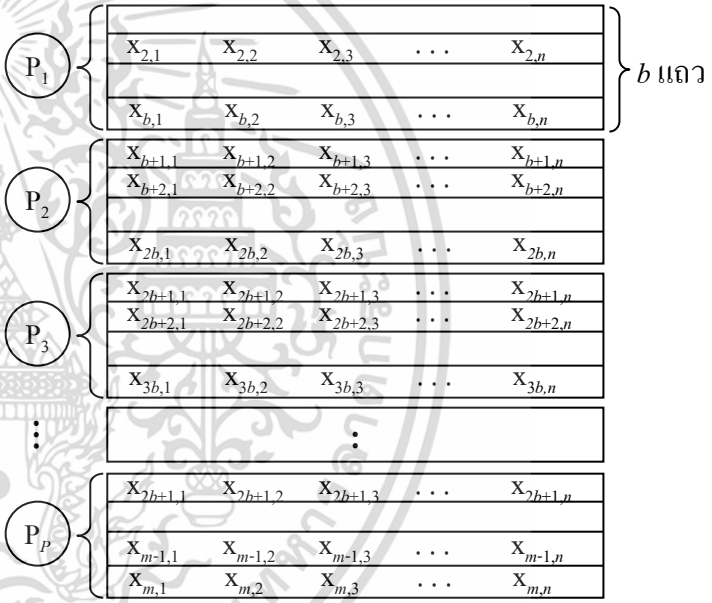
1) วิธีการแบ่งชุดข้อมูลแบบ Rowwise-Block Stripping คือ การแบ่งข้อมูลในแต่ละแถวของชุดข้อมูลที่ติดกัน โดยจะถูกแบ่งให้แต่ละหน่วยประมวลผลมีขนาดเท่าๆกัน

โดยขนาดของข้อมูลในแต่ละกลุ่มจะ

กำหนดให้เท่ากับ $b = \left\lceil \frac{n}{P} \right\rceil$

เมื่อ b = จำนวนแถวสำหรับแต่ละหน่วยประมวลผล

m และ n คือ ขนาดเมตริกซ์ $m \times n$



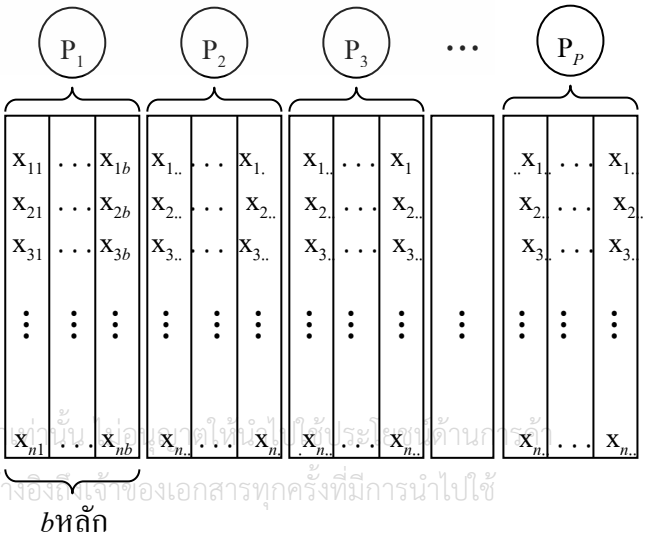
2) วิธีการแบ่งชุดข้อมูลแบบ Columnwise-Block Stripping คือ การแบ่งชุดข้อมูลในแต่ละหลักของชุดข้อมูลที่ติดกัน ถูกแบ่งให้แต่ละหน่วยประมวลผลมีขนาดเท่าๆกัน

โดยขนาดของข้อมูลในแต่ละกลุ่มจะ

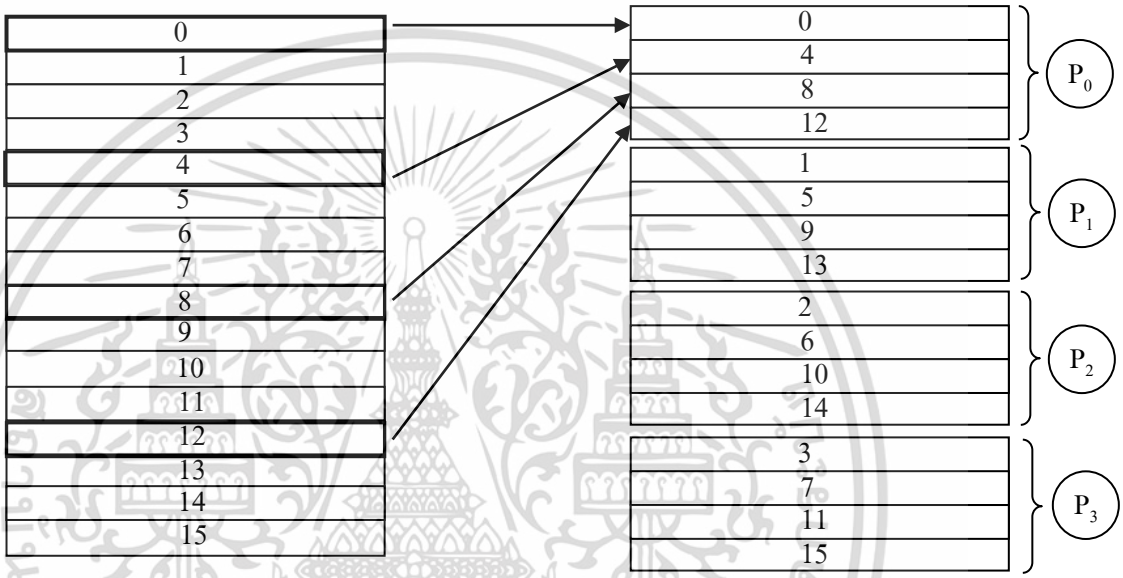
กำหนดให้เท่ากับ $b = \left\lceil \frac{n}{P} \right\rceil$

เมื่อ b = จำนวนหลักสำหรับแต่ละ PE

m และ n คือ ขนาดเมตริกซ์ $m \times n$

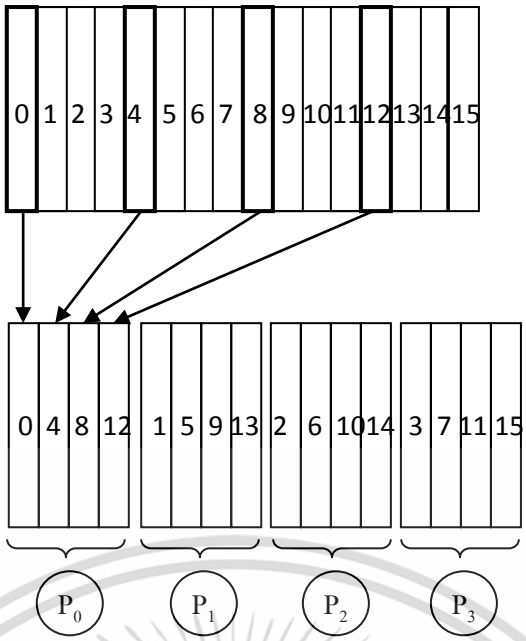


3) วิธีการแบ่งชุดข้อมูลแบบ Rowwise-Cyclic Stripping คือ แต่ละ PE จะรับผิดชอบแถวของข้อมูลที่มีค่าไม่ติดกัน เป็นจำนวนเท่ากับปริมาณงานหรือจำนวนแถวที่แต่ละ PE ต้องรับผิดชอบ คือ $p = \left\lceil \frac{n}{P} \right\rceil$ กรณีนี้ข้อมูลที่อยู่ติดกัน Pแถวแรกจะถูกจัดสรรให้ P PEs ในรอบแรกของการทำงาน และข้อมูล P แถวถัดมา จะถูกจัดให้ P PEs ในรอบที่ 2 ทำซ้ำแบบเดิมไปเรื่อยๆ จนครบทุกแถว ดังตัวอย่างเมตริกซ์ขนาด 16×16 (รูปที่ 2.20)



รูปที่ 2.20 การแบ่งชุดข้อมูลด้วยวิธี Rowwise – cyclic Stripping

4) วิธีการแบ่งชุดข้อมูลแบบ Columnwise-Cyclic Stripping คือแต่ละ PE จะรับผิดชอบหลักที่มีค่าไม่ติดกัน เป็นจำนวนเท่ากับจำนวนหลักที่แต่ละ PE ต้องรับผิดชอบ คือ $p = \left\lceil \frac{n}{P} \right\rceil$ กรณีนี้ข้อมูล P หลักแรก ที่อยู่ติดกัน จะถูกจัดสรรให้ P PEs ในรอบแรกของการทำงาน และข้อมูลใน P แถวถัดมา จะถูกจัดให้ P PEs ในรอบที่ 2 ทำซ้ำเช่นนี้ต่อไปเรื่อยๆ จนครบทุกหลัก ดังตัวอย่างเมตริกซ์ขนาด 16×16 (รูปที่ 2.21)



รูปที่ 2.21 การแบ่งชุดข้อมูลด้วยวิธี Columnwise – cyclic Stripping

2.4.2 การแบ่งชุดข้อมูลแบบตาราง (Checkerboard Partitioning)

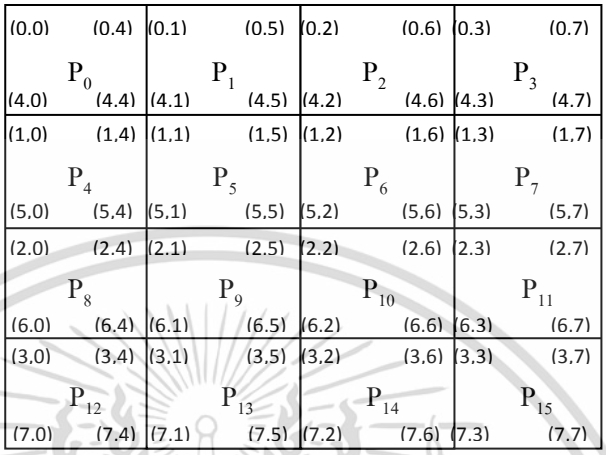
วิธีการนี้ทำโดยแบ่งข้อมูลติดกัน (Block) หรือวงรอบ (Cyclic) สามารถแบ่งเป็น 2 ประเภท ซึ่งมีรายละเอียดดังนี้

1) วิธีการแบ่งชุดข้อมูลแบบ Block – checkerboard partitioning คือ การแบ่งชุดข้อมูลข้อมูลเป็นกลุ่มๆเท่าๆกัน และจัดสรรให้หนึ่งกลุ่มต่อหนึ่งหน่วยประมวลผล ดังตัวอย่างเมตริกซ์ขนาด 16×16 (รูปที่ 2.22)

(0,0) (0,1) (1,0) (1,1) P_0	(0,2) (0,3) (1,2) (1,3) P_1	(0,4) (0,5) (1,4) (1,5) P_2	(0,6) (0,7) (1,6) (1,7) P_3
(2,0) (2,1) (3,0) (3,1) P_4	(2,2) (2,3) (3,2) (3,3) P_5	(2,4) (2,5) (3,4) (3,5) P_6	(2,6) (2,7) (3,6) (3,7) P_7
(4,0) (4,1) (5,0) (5,1) P_8	(4,2) (4,3) (5,2) (5,3) P_9	(4,4) (4,5) (5,4) (5,5) P_{10}	(4,6) (4,7) (5,6) (5,7) P_{11}
(6,0) (6,1) (7,0) (7,1) P_{12}	(6,2) (6,3) (7,2) (7,3) P_{13}	(6,4) (6,5) (7,4) (7,5) P_{14}	(6,6) (6,7) (7,6) (7,7) P_{15}

รูปที่ 2.22 การแบ่งชุดข้อมูลแบบ Block - checkerboard partitioning

2) วิธีการแบ่งชุดข้อมูลแบบ Cyclic – checkerboard partitioning คือ การแบ่งชุดข้อมูลเป็นเมตริกซ์ย่อยในหลักและตามด้วยแถวของชุดข้อมูล ซึ่งข้อมูลจะไม่ติดกันโดยแต่ละหน่วยประมวลผลจะมีขนาดของเมตริกซ์ย่อยเท่าๆกันดังรูปที่ 2.23



รูปที่ 2.23 การแบ่งชุดข้อมูลแบบตาราง Cyclic - checkerboard partitioning

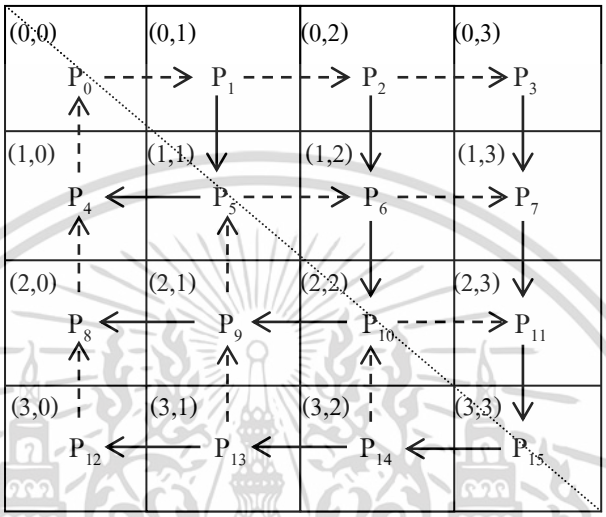
2.5 หลักการทรานสโพสเมตริกซ์แบบขนาน (Parallel Matrix Transpose)

การทรานสโพสเมตริกซ์ A ที่มีขนาด $n \times n$ คือ เมตริกซ์ A^T ด้วยขนาดที่เหมือนกัน ตัวอย่างเช่น $A^T[i,j] = A[j,i]$ เมื่อ $0 \le i, j < n$ ในกระบวนการทรานสโพสเมตริกซ์ ทุกๆตำแหน่งที่อยู่ใต้เส้นทแยงมุม จะถูกย้ายมาตำแหนงบนเส้นทแยงมุม ถ้าเราสมมุติว่าในการย้ายตำแหน่งหนึ่งตำแหน่งนั้นจะใช้เวลาหนึ่งหน่วยเวลา แล้วเวลาในการทรานสโพสเมตริกซ์ที่มีขนาด $n \times n$ คือ $\frac{(n^2-n)}{2}$ เมื่อ n คือขนาดของเมตริกซ์ สำหรับ การทรานสโพสเมตริกซ์ มีหลายวิธี ในหัวข้อนี้จะเลือกวิธีการทรานสโพสเมตริกซ์แบบขนานด้วยการเชื่อมต่อแบบเมสช์ (Mesh)

การทรานสโพสเมตริกซ์แบบขนานด้วยการเชื่อมต่อแบบเมสช์

เราสมมติว่า เมตริกซ์ขนาด $n \times n$ ถูกจัดเก็บในการเชื่อมต่อหน่วยประมวลผลแบบเมสช์ด้วยขนาด $n \times n$ ดังนั้น หนึ่งหน่วยประมวลผลจะถือครองเพียงหนึ่งค่าภายในเมตริกซ์ จากรูปที่ 2.24 และรูปที่ 2.25 แสดงขั้นตอนการ ทรานสโพสเมตริกซ์ สำหรับเมตริกซ์ขนาด 4×4 บนการเชื่อมต่อหน่วยประมวลผล 16 หน่วยประมวลผลแบบเมสช์ และมีเส้นทแยงมุมที่ถูกครอบครองโดย P_0, P_5, P_{10} และ P_{15} เป็นแกนการ ทรานสโพสนั้นทำได้โดย ย้ายเมตริกซ์ที่อยู่ข้างล่างเส้นทแยงมุม

ไปในตำแหน่งที่ตรงกันข้ามที่อยู่เหนือเส้นทแยงมุม หน่วยประมวลผลที่อยู่ใต้เส้นทแยงมุมจะถูกส่งขึ้นผ่านไปในตำแหน่งเส้นทแยงมุม แล้วก็จะถูกส่งไปทางขวาไปยังหน่วยประมวลผลปลายทางเช่นเดียวกับ หน่วยประมวลผลที่อยู่บนเส้นทแยงมุมจะถูกส่งลงผ่านเส้นทแยงมุม แล้วก็ส่งไปทางซ้าย ไปยังหน่วยประมวลผลปลายทางเช่นกัน



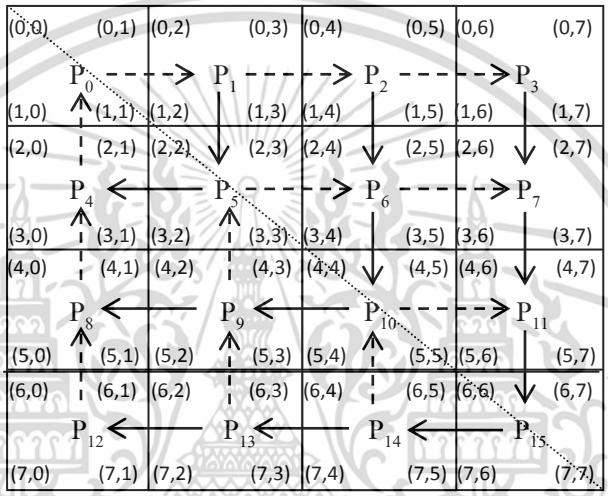
รูปที่ 2.24 ทิศทางการสื่อสารของการทรานสโพสเมตริกซ์ด้วยการเชื่อมต่อแบบเมสช์ ($P = n^2$)

(0,0) P_0	(1,0) P_1	(2,0) P_2	(3,0) P_3
(0,1) P_4	(1,1) P_5	(2,1) P_6	(3,1) P_7
(0,2) P_8	(1,2) P_9	(2,2) P_{10}	(3,2) P_{11}
(0,3) P_{12}	(1,3) P_{13}	(2,3) P_{14}	(3,3) P_{15}

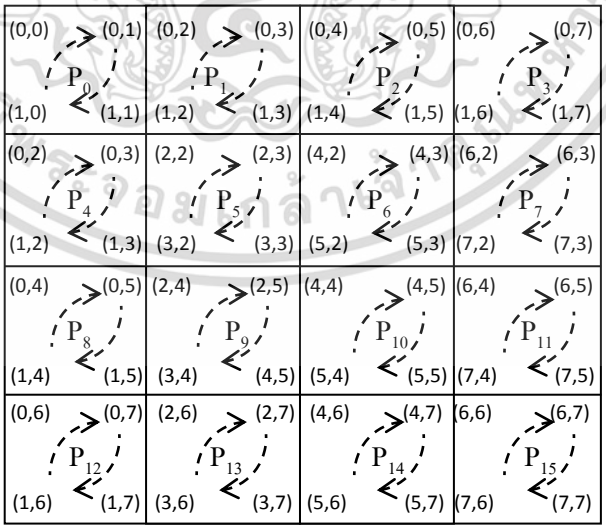
รูปที่ 2.25 ผลการทรานสโพสเมตริกซ์ 4×4 บน 16 หน่วยประมวลผลแบบที่เชื่อมต่อแบบเมสช์

จากรูปที่ 2.24 แสดงลำดับการสื่อสารของหน่วยประมวลผลสำหรับการทรานสโพสเมตริกซ์ 4×4 บน 16 หน่วยประมวลผลที่เชื่อมต่อกันด้วยเครือข่ายเมสช์ สำหรับตัวอย่าง เริ่มแรกค่าที่อยู่ใน P_8ต้องการย้ายไปที่ P_2 โดยย้ายจาก P_8 ผ่าน P_4, P_0และ P_1 และค่าที่อยู่ใน P_2ต้องการย้ายมา P_8 โดยย้ายจาก P_2ผ่าน P_6, P_{10}และ P_9 จากรูปที่ 2.25 เป็นผลลัพธ์ของการทรานสโพสด้วยวิธีนี้

ก่อนหน้านี้เราพิจารณาในกรณีที่จำนวนของหน่วยประมวลผล P เท่ากับ n^2 ($P = n^2$) สำหรับกรณีที่ P น้อยกว่า n^2 ($P < n^2$) นั้นการทรานสโพสทั้งเมตริกซ์สามารถทำได้สองขั้นตอน โดยใช้วิธีการแบ่งข้อมูลแบบ Checkerboard partitioning เข้ามาช่วย ดังแสดงตัวอย่างในกรณีที่เมตริกซ์ขนาด 8×8 ในขั้นตอนแรกจะเป็นการย้ายค่าที่อยู่ในหน่วยประมวลผลเหมือนกับกรณีที่ $P = n^2$ (รูปที่ 2.26) และขั้นตอนต่อมา คือ การทำการทรานสโพสในแต่ละหน่วยประมวลผล (Transpose Locally) ดังรูปที่ 2.27



รูปที่ 2.26 ทิศทางการสื่อสารของการทรานสโพสเมตริกซ์ด้วยการเชื่อมต่อแบบเมสช์ ($P < n^2$)



รูปที่ 2.27 การทรานสโพสในแต่ละหน่วยประมวลผล (Transpose Locally)

2.6 การหาผลคูณของเมตริกซ์แบบขนาน (Parallel Matrix Multiplication)

การหาผลคูณเมตริกซ์เป็นปัญหาคลาสสิกสำหรับการประมวลผลแบบขนาน เพราะเมตริกซ์เป็นพื้นฐานในการหาคำตอบทางวิทยาศาสตร์ และวิศวกรรมหลายๆด้าน เช่น การหาคำตอบของสมการหลายตัวแปร โดยใช้เมตริกซ์ การวิเคราะห์โครงสร้างทางวิศวกรรม การหาระยะทางที่สั้นที่สุดโดยใช้เมตริกซ์ หรือแม้แต่งานทางด้านกราฟิค เป็นต้น

การคูณเมตริกซ์แบบขนาน เป็นการคูณเมตริกซ์ ถ้าสองเมตริกซ์ที่มีขนาด $n \times n$ โดยกำหนดให้หน่วยประมวลผลหลายๆหน่วยทำงานไปพร้อมๆกัน เพื่อลดความซับซ้อนทางด้านเวลา (Time Complicity) ในการประมวลผลจาก $O(n^3)$ เป็น $O(\log_2 n)$ เมื่อใช้หน่วยประมวลผลจำนวน $n \times n \times n$ และ $O(n)$ เมื่อใช้หน่วยประมวลผล $n \times n$ หรือ $O(n^2)$ เมื่อใช้หน่วยประมวลผล n หน่วย โดยวิธีการหาผลคูณเมตริกซ์ มีวิธีการดังนี้

กำหนดให้เมตริกซ์ $C_{n \times n}$ เป็นผลลัพธ์ที่เกิดจากการคูณของเมตริกซ์ $A_{n \times n}$ และเมตริกซ์ $B_{n \times n}$ โดยมีสูตรในการคูณกันระหว่างเมตริกซ์ ดังนี้

$$C_{n\times n} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1j} & \cdots & c_{1n} \\ c_{21} & c_{22} & & c_{2j} & & c_{2n} \\ \vdots & & & & & \\ c_{i1} & c_{i2} & \cdots & c_{ij} & \cdots & c_{in} \\ \vdots & & & & & \\ c_{n1} & c_{n2} & \cdots & c_{nj} & \cdots & c_{nn} \end{pmatrix}_{n\times n} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{2j} & & a_{2n} \\ \vdots & & & & & \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & & & & & \\ a_{n1} & a_{n2} & \cdots & a_{nj} & \cdots & a_{nn} \end{pmatrix}_{n\times n} \times \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1j} & \cdots & b_{1n} \\ b_{21} & b_{22} & & b_{2j} & & b_{2n} \\ \vdots & & & & & \\ b_{i1} & b_{i2} & \cdots & b_{ij} & \cdots & b_{in} \\ \vdots & & & & & \\ b_{n1} & b_{n2} & \cdots & b_{nj} & \cdots & b_{nn} \end{pmatrix}_{n\times n} = A_{n\times n} \quad B_{n\times n}$$

รูปที่ 2.28 การหาผลคูณของเมตริกซ์ $C_{n \times n} = A_{n \times n} \times B_{n \times n}$

จากรูปที่ 2.28 สามารถสร้างเป็นสมการในการหาผลคูณเมตริกซ์แบบลำดับได้ ดังนี้

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$

เมื่อ c_{ij} คือ ค่าของเมตริกซ์ C ซึ่งเป็นผลคูณของเมตริกซ์ A แถวที่ i และเมตริกซ์ B หลักที่ j

a_{ik} คือ ค่าในเมตริกซ์ A แถวที่ i หลักที่ k

b_{ki} คือ ค่าในเมตริกซ์ B แถวที่ k หลักที่ j

โดย $i = 1, 2, \ldots, n$ และ $j = 1, 2, \ldots, n$

```
Matrix Multiplication: C = A×B : O(n^3)
        Begin
        for i = 1 to n do                       //elements in row i
            for j = 1 to n do                   //elements in col j
                        c[i, j] = 0
                        for k = 1 to n do       //sum of C = A×B (on element k)
                                c[i,j] = c[i,j] + a[i,k]*b[k,j]
                        end for k
                end for j
            end for i
         end
```

รูปที่ 2.29 ขั้นตอนวิธีสำหรับการคูณเมตริกซ์แบบลำดับ : $O(n^3)$

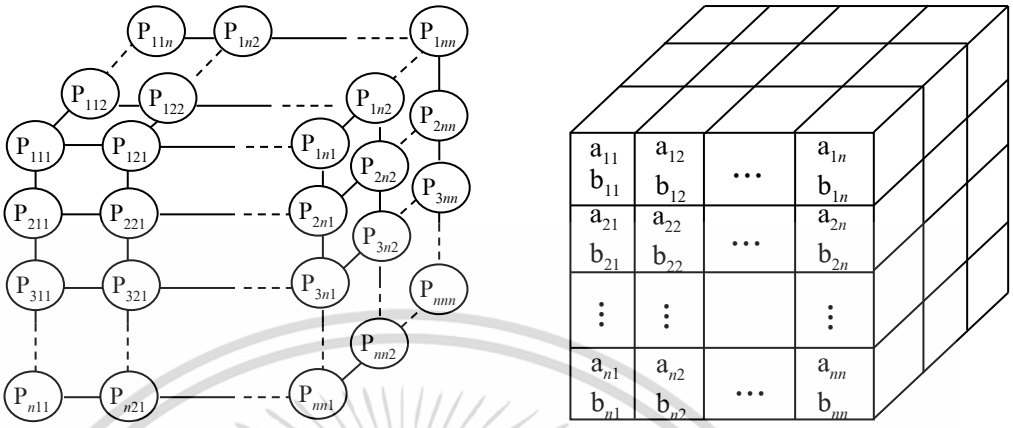
การหาผลคูณแบบลำดับจะเริ่มคำนวณจากแถวแรก จากซ้ายไปขวา สำหรับสมาชิก n สมาชิก ของเมตริกซ์ C ซึ่งจะเป็นการคูณแถวแรกของเมตริกซ์ A กับทุกๆ หลักของเมตริกซ์ B ดังรูป

สำหรับการคูณเมตริกซ์แบบขนาน จะมีดำเนินการคูณกันทีละหลายๆ สมาชิกไปพร้อมๆ กัน ซึ่งสามารถออกแบบขั้นตอนวิธีแบบขนานได้หลายวิธีที่แตกต่างกันตามขนาดของระบบ และชนิดของเครือข่ายการเชื่อมต่อที่ประยุกต์ใช้ ด้วยความซับซ้อนด้านเวลาที่แตกต่างกันโดยมีรายละเอียดดังนี้

2.6.1 การหาผลคูณของเมตริกซ์แบบขนานบนการติดต่อแบบ 3 – D คิวบ์

วิธีการหาผลคูณเมตริกซ์แบบขนานด้วยการเชื่อมต่อแบบ 3-D CUBE ด้วยโมเดลแบบพีแรม (CREW - PRAM) ซึ่งอนุญาตให้ทุกหน่วยประมวลผลสามารถอ่านข้อมูลในหน่วยความจำ ณ จุด เดียวกันในหนึ่งรอบเวลา โดยใช้ $n \times n \times n$ หน่วยประมวลผล ด้วยความซับซ้อนด้านเวลา เท่ากับ $O(log_2 n)$ ซึ่งจะพิจารณาดังนี้ สมมติให้หน่วยประมวลผล PE(i, j, k) ดังรูป 2.30(ก) เมื่อ i, j, k แทนหมายเลขของ PE ซึ่งระบุตำแหน่งในมิติที่ 1, 2, 3 ของ PE และ $1 \leq i, j, k \leq n$; n แทนจำนวนเต็ม

แบบสองยกกำลัง (2^k) และสมมติให้หน่วยความจำ (Memory) ของระบบเป็นอะเรย์ (Array) 3 มิติ ดังรูป 2.30(ข) ซึ่งจะเก็บค่าของเมตริกซ์ $A_{n\times n}$ และ $B_{n\times n}$ ในหน่วยความจำ 2 มิติแรก

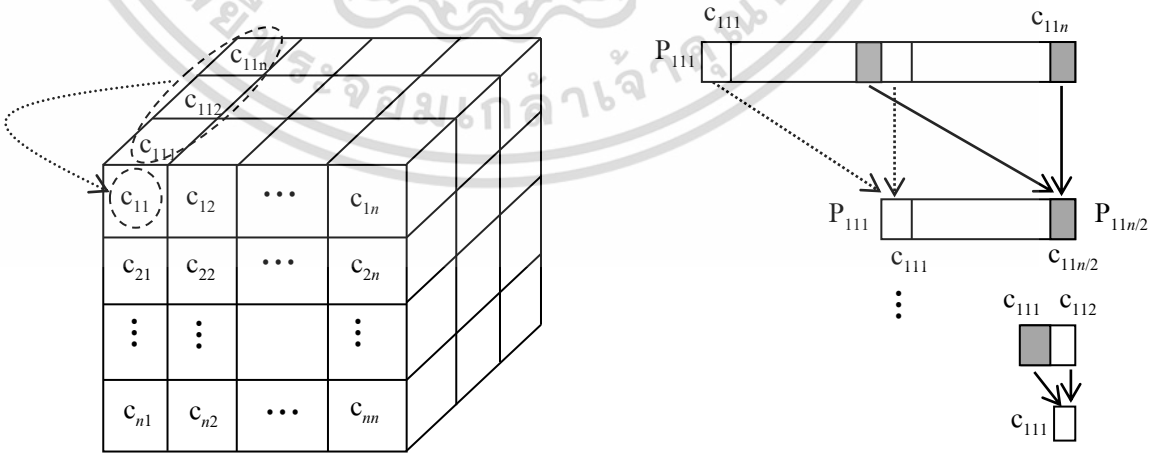


(ก) การติดต่อของหน่วยประมวลผลแบบพีแรม จำนวน n^3 หน่วย

(ข) หน่วยความจำร่วม

รูปที่ 2.30 การติดต่อของหน่วยประมวลผล และหน่วยความจำร่วมของการคูณเมตริกซ์แบบขนานด้วยการเชื่อมต่อแบบ 3-D CUBE

สำหรับขั้นตอนการคูณเมตริกซ์แบบขนานบนโมเดลแบบ CREW – PRAM จะประกอบด้วย 2 ขั้นตอน ขั้นตอนแรกจะเป็นการคำนวณหา ค่า $c_{i,j,k} = a_{i,k} \times b_{k,j}$ แล้วนำค่า $c_{i,j,k}$ ไปเก็บในหน่วยความจำร่วม และขั้นตอนที่สอง ทุกหน่วยประมวลผลจะนำค่า $c_{i,j,k}$ มารวมกันเป็น $c_{i,j}$ โดยการบวกทีละคู่ จำนวน $\log_2 n$ รอบดังรูปที่ 2.31



รูปที่ 2.31 ผลการคำนวณเมตริกซ์ C ของการคูณเมตริกซ์แบบขนานด้วยการเชื่อมต่อแบบ 3-D CUBE

ขั้นตอนที่ 1:O(1) ทุกๆn^3 PEs (P_{ijk}เมื่อ $i, j, k = 1, 2,\ldots,n$)

CRead A(i, k)
CRead B(k, j)
Compute A(i, k) * B(k, j)
Store C(i, j, k)

ขั้นตอนที่ 2 :O($log_2 n$) ทุกๆ $n^3/2$ PEs (P_{ijk} $i, j = 1,2,\ldots,n$; $k = 1,2,\ldots,n/2$)

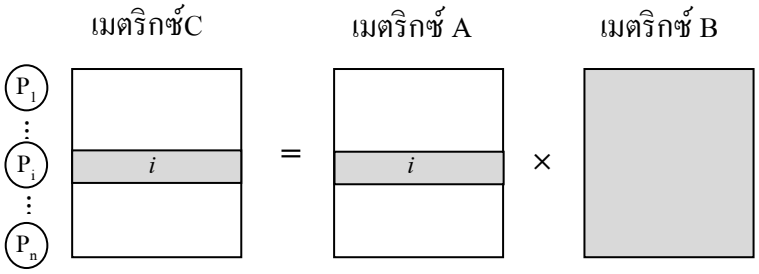
L←n
Repeat (all P_{ijk} $i, j = 1,2,\ldots,n$; $k = 1,2,\ldots,n/2$)
L←L/2
if (k< L) then begin
Read C(i, j, k)
Read C(i, j, k+L)
Compute C(i, j, k) + C(i, j, k+L)
Store in C(i, j, k)
End
until (L=1)

รูปที่ 2.32 ขั้นตอนทางคอมพิวเตอร์ของการหาผลคูณเมตริกซ์บนการติดต่อแบบพีแรมซีอาร์อีดับเบิลยู (PRAM-CREW on n^3CUBE)

2.6.2 การหาผลคูณของเมตริกซ์แบบขนานบนการติดต่อแบบอะเรย์เชิงเส้น

การหาผลคูณของเมตริกซ์ในหัวข้อนี้จะแบ่งวิธีการคูณเมตริกซ์ออกเป็น 2 แบบ ที่แตกต่างกันตามการติดต่อสื่อสารของหน่วยประมวลผล และการใช้เนื้อที่เก็บข้อมูลที่ซ้ำซ้อนกันในแต่ละหน่วยประมวลผล โดยรายละเอียดในแต่ละแบบมีดังนี้

แบบที่ 1:O(n^2) การคูณเมตริกซ์แบบขนานบนเครือข่ายอะเรย์เชิงเส้นแบบที่ 1 นี้ จะมีการเก็บข้อมูลซ้ำซ้อนในหน่วยประมวลผลมากกว่าแบบที่ 2 โดยวิธีนี้มีหน่วยประมวลผลจำนวน n PEs และเมตริกซ์มีขนาด $n \times n$ (N = $n \times n$) ดังนั้นในแต่ละหน่วยประมวลผลจะมีปริมาณงาน (Work Load) เท่ากับ N/P สมาชิก ของเมตริกซ์ C การคูณเมตริกซ์แบบขนานกรณีนี้ ก่อนการประมวลผลแต่ละหน่วยประมวลผล P_iจะเก็บค่าแถวที่ i ของเมตริกซ์ A และเก็บทุกค่าของเมตริกซ์ B ดังรูป 2.33

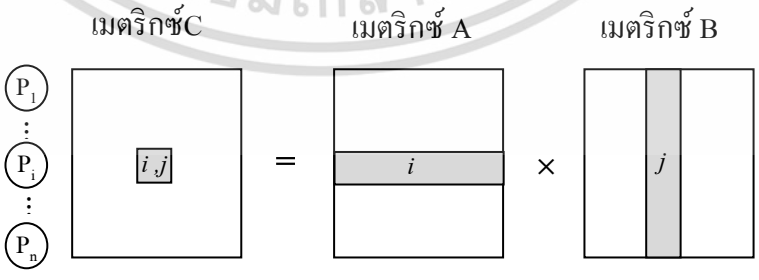


รูปที่ 2.33 การหาผลคูณของเมตริกซ์แบบขนานบนการติดต่อแบบอะเรย์เชิงเส้นแบบที่ 1

```
Begin
    for  all i = 1 to npardo                    //P_i Computes row i
        for j = 1 to n
            c[i, j] = 0
            for k = 1 to n
                c[i, j] = c[i, j] + a[i, k] * b[k, j]
            end for k
        end for j
    end for all
end
```

รูปที่ 2.34 ขั้นตอนทางคอมพิวเตอร์ของการหาผลคูณเมตริกซ์บนการติดต่อแบบอะเรย์เชิงเส้นแบบที่ 1

แบบที่ 2:O(n^2) การคูณเมตริกซ์แบบขนานบนเครือข่ายแบบอะเรย์เชิงเส้นแบบที่ 2 จะมีการสื่อสารระหว่างหน่วยประมวลผลมากกว่าแบบที่ 1 แต่ไม่มีการเก็บข้อมูลที่ซ้ำซ้อนในแต่ละหน่วยประมวลผลเลย โดยวิธีนี้ก่อนการประมวลผลแต่ละหน่วยประมวลผล P_i จะเก็บค่าแถวที่ *i* ของเมตริกซ์ A และเก็บหลักที่ *j* ของเมตริกซ์ B ดังรูปที่ 2.35



รูปที่2.35 การหาผลคูณของเมตริกซ์แบบขนานบนการติดต่อแบบอะเรย์เชิงเส้นแบบที่ 2

เริ่มแรกหน่วยประมวลผล P_i มีค่าในแถว i ของเมตริกซ์ A และหลัก j ของเมตริกซ์ B และต้องการคำนวณแถว i ของเมตริกซ์ C แต่เนื่องจากข้อมูลที่มีอยู่แบบไม่ซ้ำซ้อนกัน จะสามารถคำนวณได้เพียง 1 ค่าในตำแหน่งที่ i ของแถว j เท่านั้น ส่วนค่าในตำแหน่งที่เหลือจะทำได้โดยการสื่อสารกันระหว่างหน่วยประมวลผลด้วยการรับ และส่งข้อมูล ของเมตริกซ์ B หลักที่ j +1 ของเมตริกซ์ B พร้อมๆกัน (Parallel Data Transfer) จากหน่วยประมวลผลที่อยู่ติดกัน (P_{j+1}) ตามเงื่อนไขดังนี้

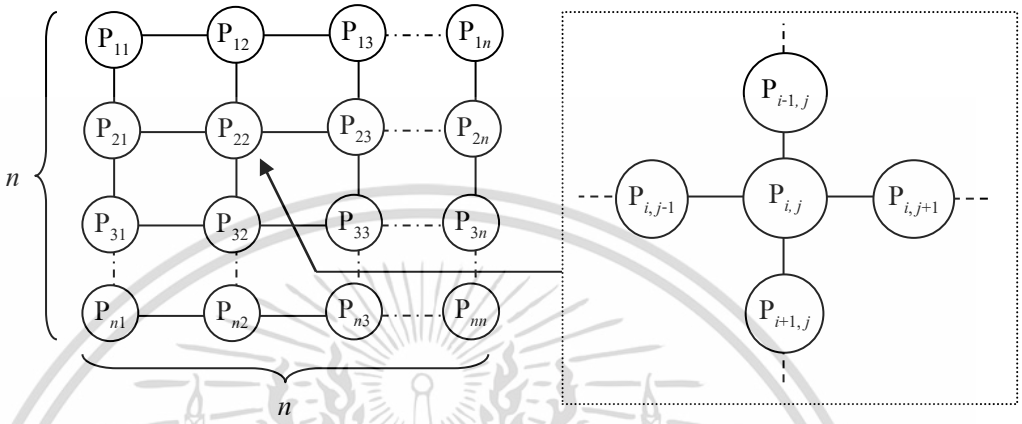
P_j : รับข้อมูลหลัก j ของเมตริกซ์ B ←จาก P_{j+1} : หลักที่ j + 1 ของเมตริกซ์ B

```
Begin
  for all i = 1 to npardo                    //P_i Computes row i
    for j = 1 to n
      c[i, j] = 0
      for k = 1 to n
        c[i, j] = c[i, j] + a[i, k] * b[k, j]
      end for k
      b[i, j] ←b[i, j + 1]                   // P_i ←P_i +1
    end for j
  end for all
end
```

รูปที่ 2.36 ขั้นตอนทางคอมพิวเตอร์ของการหาผลคูณเมตริกซ์บนการติดต่อแบบอะเรย์เชิงเส้นแบบที่ 2

2.6.3 การหาผลคูณของเมตริกซ์แบบขนานบนการติดต่อแบบ2 – D เมสช์ (Mesh)

การคูณเมตริกซ์แบบขนานบนเครือข่ายแบบ 2 - D เมสช์ ขนาด $n \times n$ หน่วยประมวลผล (PEs) โดยลักษณะการเชื่อมต่อกันของแต่ละหน่วยประมวลผลเป็นดังรูปที่ 2.37



รูปที่ 2.37 หน่วยประมวลผลที่เชื่อมต่อแบบเมสช์

ก่อนการประมวลผลแต่ละหน่วยประมวลผล $P_{i,j}$ จะเก็บค่า $a_{i,j}$ ของเมตริกซ์ A และเก็บค่า $b_{i,j}$ ของเมตริกซ์ B ซึ่งกรณีนี้จะไม่มีการเก็บข้อมูลซ้ำซ้อนในแต่ละหน่วยประมวลผล แต่ต้องมีการติดต่อสื่อสารระหว่างหน่วยประมวลผล สำหรับข้อมูลระหว่างแถว i ของเมตริกซ์ A พร้อมๆ กัน และข้อมูลระหว่างหลักที่ j ของเมตริกซ์ B ไปพร้อมๆ กันดังนี้

ในขั้นแรก หน่วยประมวลผล $P_{i,j}$ มีค่า $a_{i,j}$ และ $b_{i,j}$ ของเมตริกซ์ A และ B และต้องการคำนวณ $c_{i,j}$ ของเมตริกซ์ C แต่ข้อมูลที่มีอยู่ในแต่ละหน่วยประมวลผลดังกล่าว ยังไม่สามารถคำนวณผลคูณได้โดยตรงต้องมีการรับ - ส่งค่าในแต่ละแถว และหลัก ที่เหมาะสมก่อน โดยการรับหรือส่งข้อมูลพร้อมๆกัน (Parallel Data Transfer) จากหน่วยประมวลผลในแถวที่ติดกัน และหลักที่ติดกันเป็นจำนวน n - 1 ครั้งก่อน ดังนี้

$P_{i,j}$: $A(i,j) \leftarrow A(i,j+1)$ จาก $P_{i,j+1}$ (หลัก j ของเมตริกซ์ A ใน$P_{i,j}$ รับมาจากหลัก $j+1$ ของเมตริกซ์ A ใน $P_{i,j+1}$)

$P_{i,j}$: $B(i,j) \leftarrow B(i+1,j)$ จาก $P_{i+1,j}$ (หลัก i ของเมตริกซ์ B ใน$P_{i,j}$ รับมาจากหลัก $i+1$ ของเมตริกซ์ B ใน $P_{i,j+1}$)

ทิศทางการรับ และส่งข้อมูล สำหรับเมตริกซ์ A และ B ที่อยู่ในแต่ละ $P_{i,j}$ ดังกล่าว จะดำเนินการพร้อมๆ กัน ก็ต่อเมื่อเงื่อนไขต่อไปนี้เป็นจริงสำหรับแต่ละค่าของ $k = 1, 2, 3, \ldots, n-1$

- ถ้า $i > k$ จะมีการรับ และส่งข้อมูลทุกคู่ $P_{i,j} \leftarrow P_{i,j+1}$ ในทิศทาง ←A (สำหรับแถว i)
- ถ้า $j > k$ จะมีการรับ และส่งข้อมูลทุกคู่ $P_{i,j} \leftarrow P_{i+1,j}$ ในทิศทาง ↑ B (สำหรับแถว j)

เมื่อดำเนินการสับหลีกแบบขนานจนครบทุกค่าของ k แล้ว จะได้ $P_{i,j}$ มีค่า $a_{i,j}$ และ $b_{i,j}$ ของเมตริกซ์ A และ B ที่เหมาะสมสำหรับในแต่ละแถว i และหลัก j ที่สัมพันธ์กับ $P_{i,j}$ ดังนั้นจะสามารถคำนวณผลคูณคู่แรกของค่า $c_{i,j}$ ของเมตริกซ์ C ได้ แต่ยังเหลือผลคูณอีก $n-1$ ค่า สำหรับแต่ละค่าของ $c_{i,j}$ ในขั้นที่ 2 ซึ่งในแต่ละรอบ ทุกๆ หน่วยประมวลผล $P_{i,j}$ จะส่งค่า ← $a_{i,j}$ ของเมตริกซ์ A และ ↑$b_{i,j}$ ของเมตริกซ์ B ไปยังหน่วยประมวลผลที่ติดกันโดยไม่มีเงื่อนไข จากนั้นจะคำนวณผลคูณและบวกสะสมเพิ่มค่า $c_{i,j}$

2.7 งานวิจัยที่เกี่ยวข้อง

ในปี 1994 Alonso Sanches และ Song[7] ทำการทดลองหาผลคูณของเมตริกซ์บนเครื่องคอมพิวเตอร์แบบขนานที่โครงสร้างของหน่วยประมวลผลติดต่อกับแบบไฮเปอร์คิวบ์ โดยใช้ขั้นตอนทางคอมพิวเตอร์แบบ SIMD และนำเสนอการมินิไมเซชันในเวลาที่ประมวลผล เรียกวิธี "MMM" โดยมีค่าความซับซ้อนด้านเวลาเท่ากับ $O(n^2/p^{2/3}\log p + n^{\lambda}/p^{\lambda/3})$ เมื่อ $1 \le p \le n^3$ มีประสิทธิภาพดีกว่าเมื่อเปรียบเทียบกับวิธีของ Dekle, Nassimi และ Sahni ซึ่งมีความซับซ้อนด้านเวลาเท่ากับ $O(n^{\lambda}/p^{(\lambda-1)/2})$ เมื่อ $1 \le \lambda \le 3$ และ $1 \le p \le n^2$

ในปี 1996 Tosic และคณะ[8] ทำการพัฒนาขั้นตอนในการหาผลคูณเมตริกซ์ทางคอมพิวเตอร์แบบ Systolic array โดยการเปลี่ยนลำดับของหน่วยประมวลผล และทำการเลื่อนลำดับการนำเข้าข้อมูลในการคำนวณผลคูณเมตริกซ์ โดยนำเวลาในการประมวลผลมาเปรียบเทียบกับวิธีของ S.Y. Kung ซึ่งได้ผลว่าเวลาของ Tasicj และคณะ เร็วกว่าวิธีของ S.Y. Kung เป็นจำนวน N-1 รอบ

ในปี 1998 Gunnels และคณะ[9] แสดงปัญหาผลคูณเมตริกซ์ทางคอมพิวเตอร์ด้วยการใช้ขนาดเมตริกซ์ A และ B ที่แตกต่างกัน ซึ่งนั้นทำให้เวลาของการประมวลผลต่างกัน โดยการสร้างแบบจำลองและเปรียบเทียบประสิทธิภาพของขนาดที่ต่างกันบนเครื่องซุปเปอร์คอมพิวเตอร์ Cray T3E

ในปี 2004 Krishnan และNieplocha [4] ได้พัฒนาวิธีการหาผลคูณเมตริกซ์แบบขนานจากวิธีการที่เรียกว่า "SRUMMA" [5] โดยปรับปรุงประสิทธิภาพการคูณเมตริกซ์ทรานสโพส และ

เมตริกซ์สี่เหลี่ยมมุมฉาก โดยดำเนินการบน “ระบบคลัสเตอร์” (Cluster System) วิธี SRUMMA โดยทั่วไปนั้นจะมีการจัดการกับหน่วยความจำ และการส่งข้อมูลระหว่างกันอย่างมีประสิทธิภาพ อีกทั้งยังมีความยืดหยุ่นทั้งบนระบบคลัสเตอร์ และระบบหน่วยความจำร่วม (Shared Memory) สำหรับวิธีการหาผลคูณของเมตริกซ์สี่เหลี่ยมมุมฉากนั้น จะแบ่งได้เป็น 3 กรณีที่ต่างกันตามขนาดของเมตริกซ์ คือ กรณีแรกคือ m มีขนาดเล็ก, n และ k มีขนาดใหญ่ กรณีที่สอง คือ k มีขนาดเล็ก, m และ n มีขนาดใหญ่ กรณีสุดท้ายคือ n มีขนาดเล็ก, m และ k มีขนาดใหญ่โดย m, n, k คือ ขนาดของเมตริกซ์ A_{mxk} xB_{kxn} ส่วนวิธีการหาผลคูณของเมตริกซ์ ทรานสโพส มี 3 กรณีที่แตกต่างกันดังนี้ กรณีแรก $C = A^TB$ กรณีที่สอง$C = AB^T$และกรณีสุดท้าย $C = A^TB^T$ตามลำดับ

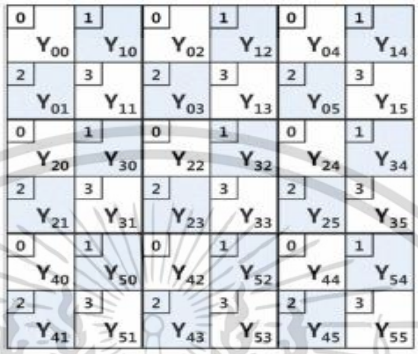
ในปี 2010 Chen และคณะ [2] ได้ออกแบบสถาปัตยกรรมแบบขนานที่ชื่อว่า ESCA (Engineering and science computing accelerator) โดยพัฒนาต่อมาจากสถาปัตยกรรม SIMD (Single-instruction multiple-data-stream)ซึ่งมีจุดมุ่งหมายเพื่อเร่งการคำนวณหาผลลัพธ์ของการคูณเมตริกซ์ โดยมีหน่วยควบคุม (CU: Control Unit) เพียงหน่วยเดียวแต่จะมีหน่วยประมวลผลย่อย (PE: Processing Element) หลายๆ หน่วยทำงานร่วมกัน ในส่วนข้างต้นเป็นส่วนที่เกี่ยวกับฮาร์ดแวร์ (Hardware) สำหรับวิธีการคูณเมตริกซ์นั้นจะใช้วิธีการที่เรียกว่า “แคนนอน” (Cannon’s Algorithm) ซึ่งเป็นขั้นตอนวิธีการคูณเมตริกซ์ที่มีมิติ $n \times n$ ด้วยการทำงานแบบขนาน

ทางด้านเวลาการทำงาน ถ้ากระบวนการทุกกระบวนการทำงานพร้อมกันแบบขนาน และกระบวนการแต่ละกระบวนการสามารถส่งเมตริกซ์ย่อยไปยังกระบวนการข้างเคียงได้เพียงหนึ่งกระบวนการต่อครั้งจะเกิดการส่งเมตริกซ์ย่อยทั้งหมด $4\sqrt{p}-2$ครั้ง และการคูณเมตริกซ์ย่อยแต่ละครั้งใช้การคูณตัวเลขทั้งหมด $\left(\frac{n}{\sqrt{p}}\right)^3$และจะได้เวลาการทำงาน คือ $O\left(\frac{n}{\sqrt{p}}\right)^3$

ในปี 2011 Kimและคณะ[1] ได้ทำการคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธีแบบไทลิงก์ (Tiling Technique) โดยไทลิงก์ คือ กระบวนการแบ่งเมตริกซ์ที่มีขนาดใหญ่ๆ ให้เป็นเมตริกซ์ย่อย (Sub-Matrix) โดยที่เมตริกซ์ที่ย่อยมาได้นั้นจะถูกเรียกว่า“ไทล์” (Tile) และในแต่ละไทล์ที่ได้แบ่งย่อยมาจากเมตริกซ์ใหญ่นั้น จะมีการดำเนินการทรานสโพสแบบขนานซึ่งจะเห็นได้ว่าถ้าเมตริกซ์ที่มีขนาดใหญ่ การ ทรานสโพสเพียงในไทล์นั้นมีค่าใช้จ่าย (Overhead) ที่น้อยกว่าเมื่อเปรียบเทียบกันกับวิธีการที่นำเมตริกซ์ทั้งหมดมาทรานสโพส โดยการทรานสโพสนั้นจะดำเนินการในเมตริกซ์ที่สอง คือ เมตริกซ์ Y เท่านั้นสมการ2.1 เป็นสมการที่ใช้หาจำนวนครั้งของการทราน

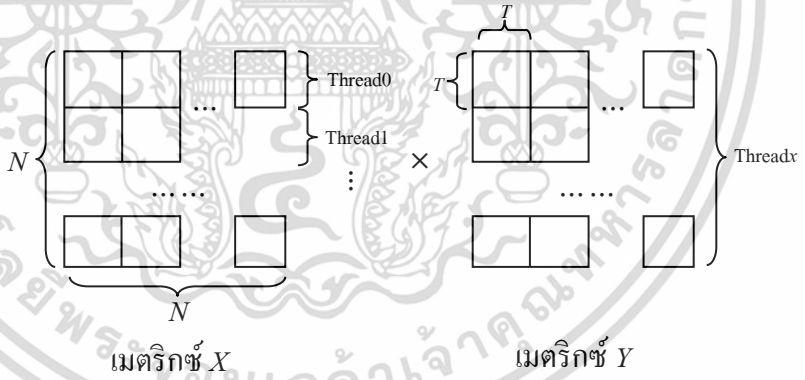
สโพสในข้อมูลเข้า (Input) ของเมตริกซ์ Y เมื่อ N คือ ขนาดของเมตริกซ์ ($N \times N$) และ T คือ ขนาดของไทล์

$$\text{Transposed elements (TF)} = 2\left(\frac{N^2}{T^2}\right)\cdot\left(\sum_{i=1}^{(T-1)} i\right) \tag{2.1}$$



รูปที่ 2.38 การทรานสโพสในเมตริกซ์ Y ของเมตริกซ์ขนาด 6×6

ยกตัวอย่างเช่น ข้อมูลเข้าในเมตริกซ์ Y มี $N = 6$ และ $T = 2$ เมื่อนำไปแทนในสมการดังกล่าวเราจะได้จำนวนครั้งของการทรานสโพสเป็น 18 ดังรูปที่ 2.38



รูปที่ 2.39 การแบ่งเมตริกซ์ลงในหลายๆไทล์

จากรูปที่ 2.39 เมตริกซ์ถูกแบ่งลงในหลายๆไทล์ซึ่งขนาดของไทล์แต่ละอันเป็น T ดังนั้นแต่ละเธรด(Thread) จะถือครองแถวหนึ่งแถวในเมตริกซ์ X และทั้งหมดของเมตริกซ์ Y โดยผลคูณเมตริกซ์จะถูกเก็บไว้ในเมตริกซ์ Z ดังสมการ 2.2

$$Z_{i,j} = \sum_{k=1}^{n} X_{i,k} Y_{k,j} \tag{2.2}$$

และได้ออกแบบวิธีการคูณเมตริกซ์แบบขนานได้ตามสมการ 2.3

$$\text{Thread } i : Z_i = X_i \times Y_{n \times n}$$
$$= [X_i \times Y_1, X_i \times Y_2, \ldots, X_i \times Y_n] \quad (2.3)$$

เมื่อ X_i และ Z_i คือ แถวที่ i ใดๆ ที่ถือครองด้วยเธรด (Thread) ที่ i^{th} ของเมตริกซ์ X และ Z ตามลำดับ และ Y_j คือ หลักที่ j ของ เมตริกซ์ Y ส่วนประสิทธิภาพที่วัดได้จากการสร้างแบบจำลองนั้น ได้นำมาเปรียบเทียบกับการคูณเมตริกซ์ที่ทำการทรานสโพสทั้งเมตริกซ์ ซึ่งจากผลการทดลองวิธีการคูณเมตริกซ์ ทรานสโพสด้วยวิธีการแบบไทลิงก์นั้นเร็วกว่าวิธีการคูณเมตริกซ์ที่ทำการทรานสโพสทั้งเมตริกซ์ 4.76% และ 6.61% เมื่อดำเนินการบนหน่วยประมวลผล Core2 9400 และ Phenom 9550 ตามลำดับ

บทที่ 3

การพาร์ทิชันข้อมูลสำหรับการคูณเมตริกซ์ทรานสโพสแบบขนาน

วิทยานิพนธ์นี้เป็นการศึกษาวิธีการหาผลคูณของเมตริกซ์แบบขนานที่ทำการทรานสโพสข้อมูลเข้า (เมตริกซ์ B) ก่อนที่จะนำข้อมูลนั้นมาหาผลคูณ โดยจะเน้นการประยุกต์ใช้จริงบนระบบคอมพิวเตอร์แบบขนานที่มีอยู่อย่างมีประสิทธิภาพ

ขั้นตอนการวิจัยการหาผลคูณของเมตริกซ์ทรานสโพสแบบขนานมี 8 ขั้นตอนดังนี้

1. ศึกษาขั้นตอนวิธีการหาผลคูณของเมตริกซ์ทรานสโพสแบบขนาน และศึกษางานวิจัยที่เกี่ยวข้องกับการหาผลคูณของเมตริกซ์ทรานสโพสแบบขนาน
2. ทำการตั้งสมมติฐาน โดยคาดว่าการประมวลผลคำตอบของการคูณเมตริกซ์ทรานสโพสด้วยวิธีแบบขนาน (Parallel) จะใช้เวลาน้อยกว่าการประมวลผลของการคูณเมตริกซ์ด้วนวิธีแบบอนุกรม (Sequential) และจะมีจำนวนครั้งของการทรานสโพสเมตริกซ์ ที่น้อยกว่าการทรานสโพสแบบปกติ
3. นำเสนอวิธีการคูณกันของเมตริกซ์ทรานสโพสแบบขนานที่มีประสิทธิภาพ
4. พัฒนาโปรแกรมแบบขนานของการหาผลคูณของเมตริกซ์ โดยใช้ภาษา C และโปรแกรม MPICH บนระบบปฏิบัติการ Linux
5. ประเมินผลการทดลองโดยการเปรียบเทียบประสิทธิภาพของการหาผลคูณของเมตริกซ์ จะประเมินผลจากเวลาที่ใช้ในการประมวลผล (Response Time) อัตราการเพิ่มของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) ของการใช้ 4 หน่วยประมวลผลเมื่อเทียบกับการใช้เพียง 1 หน่วยประมวลผล
6. วิเคราะห์ผลการทดลอง
7. สรุปผลการทดลอง และเสนอแนะการศึกษาและพัฒนาขั้นต่อไปเกี่ยวกับการประมวลผลแบบขนาน
8. เขียนวิทยานิพนธ์

โครงงานวิจัยนี้มุ่งเน้นที่การเพิ่มประสิทธิภาพในการลดเวลาที่ใช้ในการติดต่อสื่อสารให้มากที่สุดเท่าที่ทำได้ และมีการจัดการกับเนื้อที่ในการเก็บข้อมูลอย่างมีประสิทธิภาพให้มากที่สุด

ในบทนี้ผู้วิจัยเสนอวิธีการพาร์ทิชันข้อมูลเพื่อทำการหาผลคูณเมตริกซ์ทรานสโพสแบบขนาน โดยจะแบ่งขั้นตอนออกเป็น 2 ส่วน คือ การทรานสโพสเมตริกซ์แบบขนาน และการหาผลคูณเมตริกซ์ทรานสโพสแบบขนาน โดยทั้งสองส่วนเป็นกระบวนการที่ต่อเนื่องกัน

3.1 การทรานสโพสเมตริกซ์แบบขนาน (Parallel Transpose of Matrix)

ในหัวข้อนี้เป็นวิธีการทรานสโพสเมตริกซ์แบบขนาน ซึ่งเป็นวิธีที่พัฒนาต่อมาจากวิธีการทรานสโพสด้วยวิธีไทลิงก์ โดยจะทำการทรานสโพสเฉพาะเมตริกซ์ B เท่านั้น ซึ่งวิธีการนี้จะทำให้จำนวนครั้งของการทรานสโพสมีจำนวนครั้งน้อยกว่าการทรานสโพสแบบปกติ และทำให้ข้อมูลที่ได้หลังจากการทำการทรานสโพส มีความเหมาะสมสำหรับการเข้าถึงข้อมูลตามมาตรฐานภาษา MPI โดยวิธีการทรานสโพสนั้นจะประกอบด้วย หน่วยประมวลผล 2 ชนิด คือ หน่วยประมวลผลหลัก (Master Processor) จำนวน 1 หน่วยประมวลผล และหน่วยประมวลผลทำงาน (Worker Processor) จำนวน 4 หน่วยประมวลผล โดยแต่ละขั้นตอนในการทำงานมีดังต่อไปนี้

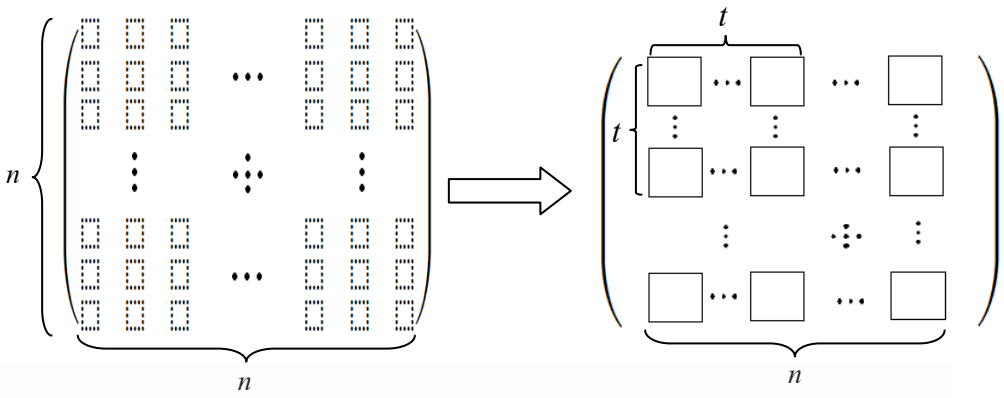
<u>หน่วยประมวลผลหลัก จำนวน 1 หน่วยประมวลผล</u>

ขั้นตอนที่ 1 สร้างข้อมูล ของเมตริกซ์ A และ เมตริกซ์ B ที่มีขนาด $n \times n$ เก็บลงในหน่วยความจำเสมือน (Virtual Memory)

ขั้นตอนที่ 2 ทำการแบ่งข้อมูลตามแถว และหลักของเมตริกซ์ B ให้มีขนาด $t \times t$ เมื่อ $t = \frac{n}{2}$ เพื่อให้ภาระงาน (Work load) สำหรับหน่วยประมวลผลทำงานทั้ง 4 หน่วย มีขนาดเท่ากัน (รูป 3.1)

ขั้นตอนที่ 3 แจ้งช่วงและขนาดของเมตริกซ์ B ที่แต่ละหน่วยประมวลผลทำงาน (Worker Processor) ต้องทำการอ่าน จากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก ที่ทำการแบ่งไว้ในขั้นตอนที่ 2

ขั้นตอนที่ 4 รอรับข้อมูลเมตริกซ์ B ที่ทำการทรานสโพสจากหน่วยประมวลผลทำงานทั้ง 4 หน่วยประมวลผล



รูปที่ 3.1 การแบ่งเมตริกซ์ขนาดใหญ่ ($n \times n$) เป็นเมตริกซ์ย่อย ($t \times t$)

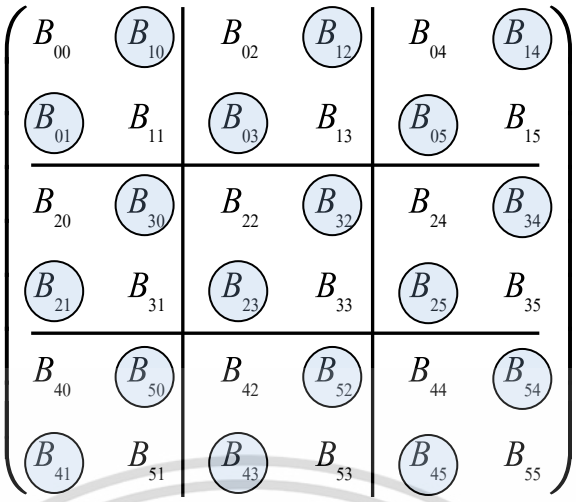
<u>หน่วยประมวลผลทำงาน 4 หน่วยประมวลผล</u>

ขั้นตอนที่ 1 รอรับการติดต่อจากหน่วยประมวลผลหลัก และอ่านข้อมูลบางส่วนของเมตริกซ์ B ขนาด $t \times t$

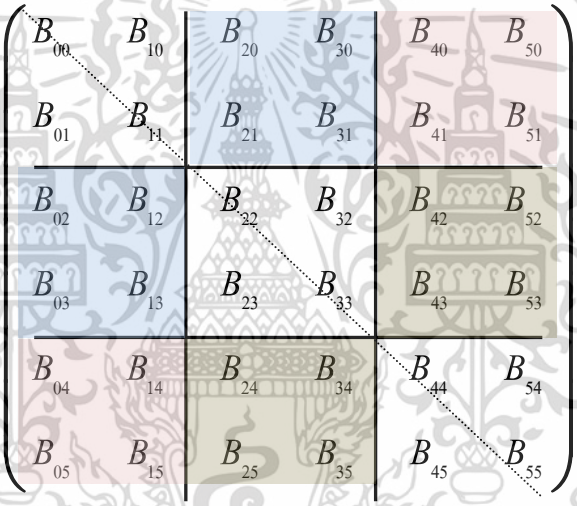
ขั้นตอนที่ 2 ทำการทรานสโพสภายในแต่ละหน่วยประมวลผลทำงานของตัวเอง ดังรูปที่ 3.2 (ก)

```
for (i = 0; i < t ; i++)
    for (j = 0; j < t ; j++)
        B[j][i] = B[i][j];
```

ขั้นตอนที่ 3 ส่งผลการทรานสโพสกลับไปยังหน่วยประมวลผลหลัก โดย P_2 และ P_3 จะส่งข้อมูลสลับกัน เพื่อจะทำให้เป็นการทรานสโพสที่สมบูรณ์



(ก) การทรานสโพสภายในเมตริกซ์ย่อย



(ข) การทรานสโพสระหว่างเมตริกซ์ย่อย

รูปที่ 3.2 การทรานสโพสในเมตริกซ์ย่อย (ก) และการทรานสโพสระหว่างเมตริกซ์ย่อย (ข) เมื่อเมตริกซ์มีขนาด 6×6

สมการต่อไปนี้ คือจำนวนครั้งของการทรานสโพสแบบดั้งเดิมแทนโดย TF จำนวนครั้งของวิธีการ ทรานสโพสแบบขนานโดยการใช้วิธีไทลิงก์แทนโดย PT และจำนวนครั้งของการทรานสโพสด้วยวิธีการใหม่ในงานวิจัยนี้แทนโดย NPT ตามลำดับ

$$TF = 2\left(\sum_{i=1}^{(n-1)} i\right) \tag{3.1}$$

$$PT = 2\left(\frac{n^2}{t^2}\right)\cdot\left(\sum_{i=1}^{(t-1)} i\right) \tag{3.2}$$

$$NPT = \left[2\left(\frac{n^2}{t^2}\right)\cdot\left(\sum_{i=1}^{(t-1)} i\right)\right] + 2\left(\sum_{i=1}^{(\frac{n}{t}-1)} i\right) \tag{3.3}$$

จากสมการ 3 สมการข้างต้น แสดงให้เห็นว่าจำนวนครั้งของการทรานสโพสในข้อมูลนำเข้าเมตริกซ์ B เมื่อใช้วิธีทรานสโพสแบบดั้งเดิม คือ $n^2 - n$ และ $n^2 - \frac{n^2}{t}$ เมื่อใช้วิธีทรานสโพสแบบขนานด้วยวิธีไทลิงก์ส่วนจำนวนครั้งของการทรานสโพสเมื่อใช้วิธีใหม่ในงานวิจัยนี้จะสมการที่ได้เป็นไปตามสมการที่ต่อไปนี้

$$\left(n^2 - \frac{n^2}{t}\right) + \left(\left(\frac{n^2}{t^2}\right) - \frac{n}{t}\right) \tag{3.4}$$

สำหรับตัวอย่างเมื่อกำหนดให้ $n = 6$ และ $t = 2$ จำนวนของการทรานสโพสในแต่ละวิธีมีดังนี้ วิธีทรานสโพสแบบดั้งเดิมโดยแทน $n = 6$ ในสมการที่ (3.1) คือ 30 ครั้งวิธีทรานสโพสแบบขนานด้วยวิธีไทลิงก์แทน $n = 6$ และ $t = 2$ ในสมการที่ (3.2) คือ 18 ครั้ง แต่วิธีการนี้ยังคงมีการย้ายข้อมูลบางส่วนในขั้นตอนการคูณเมตริกซ์ และวิธีการใหม่ในงานวิจัยนี้ โดยแทน $n = 6$ และ $t = 2$ ในสมการที่ (3.3) คือ 24 ครั้งโดยที่ไม่ต้องมีการย้ายข้อมูลในขั้นตอนการคูณเมตริกซ์

ตัวอย่างที่ 3.1 การทรานสโพสเมตริกซ์แบบขนาน ของเมตริกซ์ $B_{8\times8}$ โดยใช้หน่วยประมวลผลจำนวน 4 หน่วยประมวลผล

เมตริกซ์ B

$$\begin{pmatrix} 6 & 7 & 3 & 11 & 18 & 4 & 12 & 12 \\ 8 & 16 & 9 & 14 & 15 & 0 & 4 & 7 \\ 18 & 0 & 12 & 6 & 11 & 14 & 2 & 1 \\ 4 & 12 & 2 & 1 & 9 & 8 & 10 & 7 \\ 4 & 1 & 15 & 6 & 4 & 19 & 8 & 16 \\ 7 & 4 & 11 & 13 & 6 & 16 & 17 & 13 \\ 14 & 3 & 17 & 19 & 11 & 7 & 14 & 7 \\ 13 & 3 & 4 & 2 & 6 & 16 & 18 & 13 \end{pmatrix}$$

ขั้นตอนที่ 1 หน่วยประมวลผลหลักทำการแบ่งแถว และหลักของเมตริกซ์ B ให้มีขนาด $t \times t$ เมื่อ $t = \frac{n}{2} = 4$

หน่วยประมวลผลที่ 1 จำนวน 4×4

$$\begin{pmatrix} 6 & 7 & 3 & 11 \\ 8 & 16 & 9 & 14 \\ 18 & 0 & 12 & 6 \\ 4 & 12 & 2 & 1 \end{pmatrix}$$

หน่วยประมวลผลที่ 2 จำนวน 4×4

$$\begin{pmatrix} 18 & 4 & 12 & 12 \\ 15 & 0 & 4 & 7 \\ 11 & 14 & 2 & 1 \\ 9 & 8 & 10 & 7 \end{pmatrix}$$

หน่วยประมวลผลที่ 3 จำนวน 4×4

$$\begin{pmatrix} 4 & 1 & 15 & 6 \\ 7 & 4 & 11 & 13 \\ 14 & 3 & 17 & 19 \\ 13 & 3 & 4 & 2 \end{pmatrix}$$

หน่วยประมวลผลที่ 4 จำนวน 4×4

$$\begin{pmatrix} 4 & 19 & 8 & 16 \\ 6 & 16 & 17 & 13 \\ 11 & 7 & 14 & 7 \\ 6 & 16 & 18 & 13 \end{pmatrix}$$

ขั้นตอนที่ 2 หน่วยประมวลผลทำงานแต่ละหน่วยประมวลผลทำการอ่านข้อมูลตามแถวและหลักของเมตริกซ์ B จำนวน 4×4 จากหน่วยประมวลผลหลัก และทำการทรานสโพสงานที่ได้รับมอบหมาย

หน่วยประมวลผลที่ 1 (P_1) ทำการทรานสโพสข้อมูลแถวที่ 1 ถึง 4 และหลักที่ 1 ถึง 4

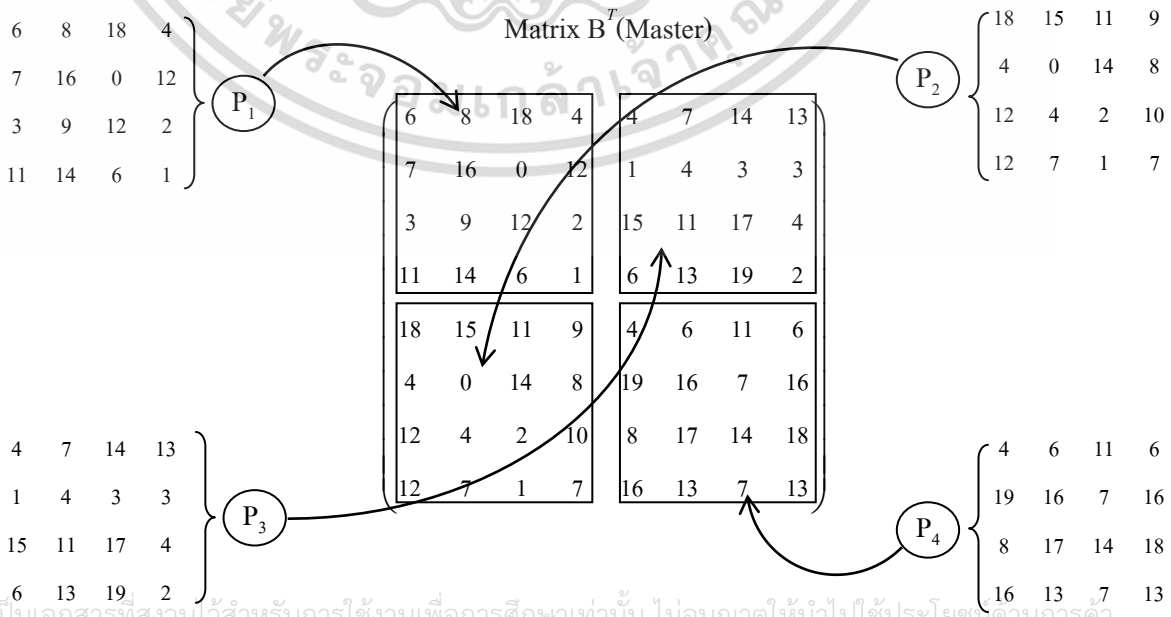
$$\begin{pmatrix} 6 & 7 & 3 & 11 \\ 8 & 16 & 9 & 14 \\ 18 & 0 & 12 & 6 \\ 4 & 12 & 2 & 1 \end{pmatrix} \xrightarrow{\text{ทรานสโพส}} \begin{pmatrix} 6 & 8 & 18 & 4 \\ 7 & 16 & 0 & 12 \\ 3 & 9 & 12 & 2 \\ 11 & 14 & 6 & 1 \end{pmatrix}$$

หน่วยประมวลผลที่ 2 (P_2) ทำการทรานสโพสข้อมูลแถวที่ 1 ถึง 4 และหลักที่ 5 ถึง 8

หน่วยประมวลผลที่ 3 (P_3) ทำการทรานสโพสข้อมูลแถวที่ 5 ถึง 8 และหลักที่ 1 ถึง 4

หน่วยประมวลผลที่ 4 (P_4) ทำการทรานสโพสข้อมูลแถวที่ 5 ถึง 8 และหลักที่ 5 ถึง 8

ขั้นตอนที่ 3 หน่วยประมวลผลทำงานส่ง ข้อมูลที่ทำการทรานสโพสเสร็จแล้วคืนให้หน่วยประมวลหลัก โดย P_2 และ P_3 จะส่งข้อมูลสลับกัน เพื่อจะทำให้เป็นการทรานสโพสที่สมบูรณ์



ตัวอย่างรหัสเหมือน เอ็มพีไอ-ซี (MPI – C like) ของการทรานสโพสแบบขนาน แสดงดังรูป 3.3

Master Processor

Start MPI

Create Data (A and B)

Send Data to Workers

```
MPI_Isend(&mat_bC1[0][0], PARTION*PARTION, MPI_DOUBLE, 1, MASTER_TO_SLAVE_TAG,
MPI_COMM_WORLD, &request);
MPI_Isend(&mat_bC2[0][0], PARTION*PARTION, MPI_DOUBLE, 2, MASTER_TO_SLAVE_TAG + 1,
MPI_COMM_WORLD, &request);
MPI_Isend(&mat_bC3[0][0], PARTION*PARTION, MPI_DOUBLE, 3, MASTER_TO_SLAVE_TAG + 2,
MPI_COMM_WORLD, &request);
MPI_Isend(&mat_bC4[0][0], PARTION*PARTION, MPI_DOUBLE, 4, MASTER_TO_SLAVE_TAG + 3,
MPI_COMM_WORLD, &request);
```

Wait for Result

```
MPI_Recv(&mat_bC1_T[0][0], PARTION*PARTION, MPI_DOUBLE, 1, MASTER_TO_SLAVE_TAG,
MPI_COMM_WORLD, &status);
MPI_Recv(&mat_bC2_T[0][0], PARTION*PARTION, MPI_DOUBLE, 3, MASTER_TO_SLAVE_TAG,
MPI_COMM_WORLD, &status);
MPI_Recv(&mat_bC3_T[0][0], PARTION*PARTION, MPI_DOUBLE, 2, MASTER_TO_SLAVE_TAG,
MPI_COMM_WORLD, &status);
MPI_Recv(&mat_bC4_T[0][0], PARTION*PARTION, MPI_DOUBLE, 4, MASTER_TO_SLAVE_TAG,
MPI_COMM_WORLD, &status);
```

Worker Processor

Receive Data from Master

```
rank :1    MPI_Recv(&mat_bC1[0][0], PARTION*PARTION, MPI_DOUBLE, 0, MASTER_TO_SLAVE_TAG,
MPI_COMM_WORLD, &status);
rank :2   MPI_Recv(&mat_bC2[0][0], PARTION*PARTION, MPI_DOUBLE, 0, MASTER_TO_SLAVE_TAG +
1, MPI_COMM_WORLD, &status);
rank :3 MPI_Recv(&mat_bC3[0][0], PARTION*PARTION, MPI_DOUBLE, 0, MASTER_TO_SLAVE_TAG +
2, MPI_COMM_WORLD, &status);
rank :4 MPI_Recv(&mat_bC4[0][0], PARTION*PARTION, MPI_DOUBLE, 0, MASTER_TO_SLAVE_TAG +
3, MPI_COMM_WORLD, &status);
```

Perform Transpose Matrix

```
for (i = 0; i < PARTION; i++)
    for (j = 0; j < PARTION; j++)
        mat_bC_T[j][i] = mat_bC4[i][j];
```

Send Data to Master

```
rank :1 MPI_Isend(&mat_bC1_T[0][0], PARTION*PARTION, MPI_DOUBLE, 0, MASTER_TO_SLAVE_TAG,
MPI_COMM_WORLD, &request);
rank :2 MPI_Isend(&mat_bC2_T[0][0], PARTION*PARTION, MPI_DOUBLE, 0, MASTER_TO_SLAVE_TAG,
MPI_COMM_WORLD, &request);
rank :3 MPI_Isend(&mat_bC3_T[0][0], PARTION*PARTION, MPI_DOUBLE, 0, MASTER_TO_SLAVE_TAG,
MPI_COMM_WORLD, &request);
rank :4 MPI_Isend(&mat_bC4_T[0][0], PARTION*PARTION, MPI_DOUBLE, 0, MASTER_TO_SLAVE_TAG,
MPI_COMM_WORLD, &request);
```

รูปที่ 3.3 ตัวอย่างรหัสเหมือน เอ็มพีไอ-ซี (MPI – C like) ของการทรานสโพสแบบขนาน

3.2 การหาผลคูณเมตริกซ์ทรานสโพสแบบขนาน (Parallel Matrix Transpose Multiplication)

ในหัวข้อนี้ผู้วิจัยได้เสนอวิธีการหาผลคูณเมตริกซ์ทรานสโพสแบบขนาน 3 วิธีที่แตกต่างกันดังนี้

1) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w (Row-Block Partitioning Matrix Transpose Multiplication with replicated data)
2) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี cbmtm_w (Checkerboard-Block Partitioning Matrix Transpose Multiplication with replicated data)
3) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w/o (Row-Block Partitioning Matrix Transpose Multiplication without replicated data)

โดยวิธี rbmtm_w เป็นวิธีที่มีประสิทธิภาพในการติดต่อสื่อสารมากที่สุด แต่ใช่เนื้อที่ในการเก็บข้อมูลซ้ำซ้อนกันมากที่สุด ส่วนวิธีที่สองจะมีการเก็บข้อมูลที่ซ้ำซ้อนกันบ้าง แต่น้อยกว่าวิธีแรก และยังมีประสิทธิภาพในการติดต่อสื่อสารระหว่างหน่วยประมวลผลได้น้อยกว่าวิธีแรก ต่อมาวิธีสุดท้ายจะไม่มีการเก็บข้อมูลซ้ำซ้อนกันระหว่างหน่วยประมวลผลเลย แต่เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลมีค่ามากที่สุด โดยรายละเอียดในแต่ละวิธีมีดังนี้

3.2.1 การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w (Row-Block Partitioning Matrix-Transpose Multiplication with replicated data)

วิธีการหาผลคูณเมตริกซ์แบบขนาน ด้วยวิธี rbmtm_w จะมีหน่วยประมวลผล 2 ชนิด ช่วยกันทำงานอย่างสอดคล้องกัน คือ หน่วยประมวลผลหลัก (Master Processor) จำนวน 1 หน่วย และหน่วยประมวลผลทำงาน (Worker Processor) จำนวน 4 หน่วย ซึ่งหน่วยประมวลผลแต่ละชนิดจะมีขั้นตอนการทำงานดังต่อไปนี้

<u>**หน่วยประมวลผลหลัก จำนวน 1 หน่วย**</u>

ขั้นตอนที่ 1 ทำการแบ่งข้อมูลตามแถวของเมตริกซ์ A ขนาด $k = \frac{n}{4}$ เพื่อให้ภาระงาน (Work

Load) มีขนาดเท่าๆกัน (แสดงในรูป 3.4) สำหรับหน่วยประมวลผลทำงาน P หน่วยประมวลผล เพื่อคำนวณหาผลคูณเมตริกซ์ C จำนวน $\frac{n}{4}$ แถวแบบขนาน

ขั้นตอนที่ 2 แจ้งช่วง และขนาดของเมตริกซ์ A ที่แต่ละหน่วยประมวลผลทำงานต้องทำการอ่านจากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก ส่วนเมตริกซ์ B ที่ได้ทำการทรานสโพสมาแล้วนั้น ทุกหน่วยประมวลผลทำงาน ต้องอ่านทั้งหมดเพื่อมาเก็บไว้ในหน่วยความจำของตัวเอง

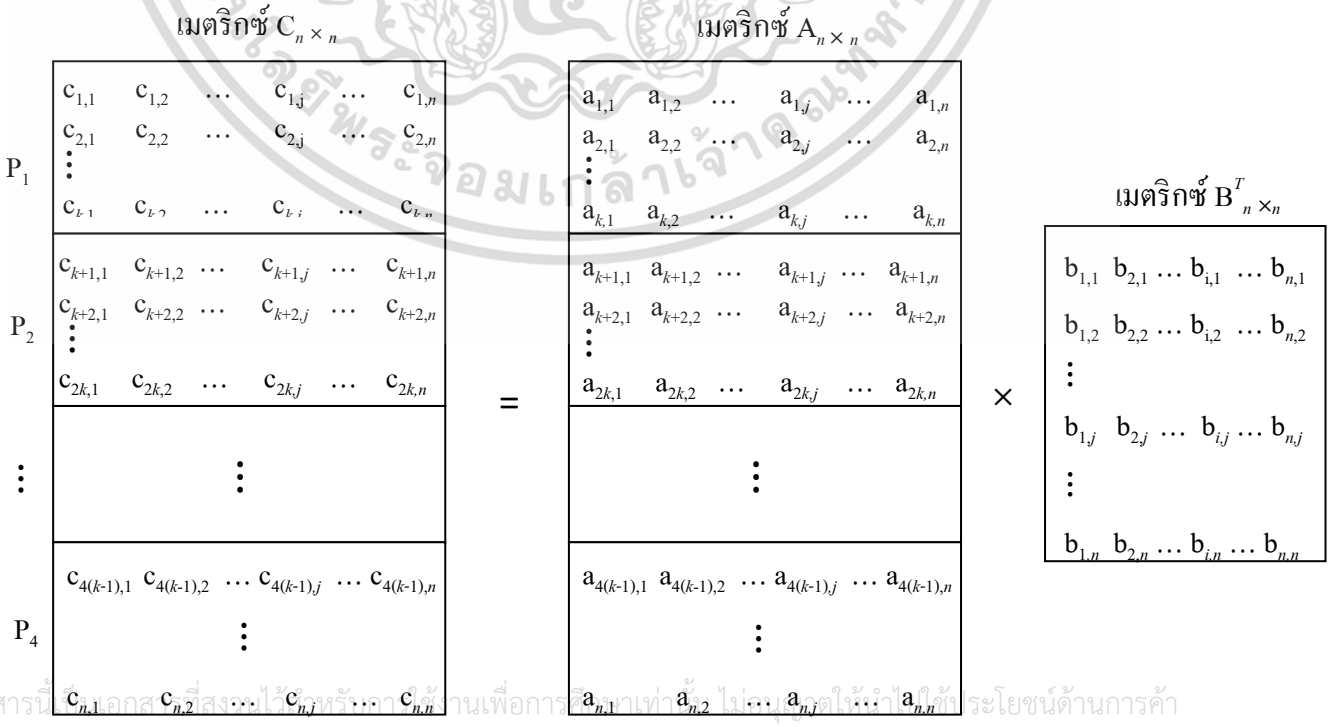
ขั้นตอนที่ 3 รอรับผลคูณเมตริกซ์ C จากหน่วยประมวลผลทำงาน 4 หน่วยประมวลผล

<u>หน่วยประมวลผลทำงาน จำนวน 4 หน่วยประมวลผล</u>

ขั้นตอนที่ 1 รอรับการติดต่อจากหน่วยประมวลผลหลัก และต่อจากนั้น อ่านข้อมูลบางส่วนของเมตริกซ์ A ขนาด $k = \frac{n}{4}$ แถว (แถวที่ $(i-1)k+1$ ถึง ik เมื่อ i คือไอดีของหน่วยประมวลผลทำงาน (Worker ID)) และอ่านข้อมูลเมตริกซ์ B ที่ได้ทำการทรานสโพสแล้ว ทั้งหมดจากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก

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

ขั้นตอนที่ 3 ส่งผลคูณเมตริกซ์ของเมตริกซ์ C กลับไปยังหน่วยประมวลผลหลัก (รูป 3.4)



รูปที่ 3.4 การแบ่งงานเพื่อหาผลคูณของเมตริกซ์ C ที่มี 4 หน่วยประมวลผล ของวิธี rbmtm_w

ตัวอย่างที่ 3.1 การหาผลคูณเมตริกซ์ทรานสโพสแบบขนาน ระหว่างเมตริกซ์ $A_{8\times8}$ และ เมตริกซ์ $B^T{}_{8\times8}$ วิธี rbmtm_w โดยใช้หน่วยประมวลผลจำนวน 4 หน่วยประมวลผล

เมตริกซ์ A

$$\begin{pmatrix} 3 & 6 & 7 & 17 & 11 & 8 & 1 & 8 \\ 13 & 12 & 3 & 11 & 11 & 8 & 14 & 0 \\ 1 & 1 & 10 & 15 & 17 & 15 & 7 & 12 \\ 12 & 13 & 9 & 1 & 3 & 2 & 4 & 17 \\ 19 & 11 & 7 & 15 & 7 & 3 & 9 & 8 \\ 3 & 11 & 3 & 7 & 2 & 16 & 6 & 16 \\ 5 & 3 & 5 & 16 & 13 & 1 & 13 & 9 \\ 3 & 8 & 2 & 13 & 17 & 14 & 5 & 2 \end{pmatrix}$$

เมตริกซ์ B^T

$$\begin{pmatrix} 6 & 8 & 18 & 4 & 4 & 7 & 14 & 13 \\ 7 & 16 & 0 & 12 & 1 & 4 & 3 & 3 \\ 3 & 9 & 12 & 2 & 15 & 11 & 17 & 4 \\ 11 & 14 & 6 & 1 & 6 & 13 & 19 & 2 \\ 18 & 15 & 11 & 9 & 4 & 6 & 11 & 6 \\ 4 & 0 & 14 & 8 & 19 & 16 & 7 & 16 \\ 12 & 4 & 2 & 10 & 8 & 17 & 14 & 18 \\ 12 & 7 & 1 & 7 & 16 & 13 & 7 & 13 \end{pmatrix}$$

<u>วิธีทำ</u>

ขั้นตอนที่ 1 หน่วยประมวลผลหลักแบ่งข้อมูลบางส่วนตามแถวของเมตริกซ์ A (แถวที่ $(i-1)k+1$ ถึง ik) และข้อมูลทั้งหมดของเมตริกซ์ B ให้แต่ละหน่วยประมวลผล i เมื่อ $i = 1, 2, 3, 4$ และ $k = n/4$

เมตริกซ์ A

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

หน่วยประมวลผลที่ 1 จำนวน 2 แถว (แถวที่ 1 และ 2)

หน่วยประมวลผลที่ 2 จำนวน 2 แถว (แถวที่ 3 และ 4)

หน่วยประมวลผลที่ 3 จำนวน 2 แถว (แถวที่ 5 และ 6)

หน่วยประมวลผลที่ 4 จำนวน 2 แถว (แถวที่ 7 และ 8)

เมตริกซ์ B^T

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

ทุกหน่วยประมวลผล จำนวน 8 แถว 8 หลัก

ขั้นตอนที่ 2 หน่วยประมวลผลแต่ละหน่วยทำการอ่านข้อมูลบางส่วนตามแถวของเมตริกซ์ A และข้อมูลทั้งหมด ของเมติรกซ์ B จากนั้นคำนวณหาผลคูณของเมตริกซ์ไปพร้อมๆกัน

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

เช่น $C_{11} = (3\times6) + (6\times8) + (7\times18) + (17\times4) + (11\times4) + (8\times7) + (1\times14) + (8\times13) = 606$

เมตริกซ์ A

3	6	7	17	11	8	1	8
13	12	3	11	11	8	14	0

×

เมตริกซ์ B^T

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

=

เมตริกซ์ C

606	646	483	344	557	658	749	435
690	698	597	516	483	747	851	685

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

เมตริกซ์ A

1	1	10	15	17	15	7	12
12	13	9	1	3	2	4	17

×

เมตริกซ์ B^T

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

=

เมตริกซ์ C

802	691	651	478	846	933	946	710
515	579	416	425	556	587	601	576

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

เมตริกซ์ A

19	11	7	15	7	3	9	8
3	11	3	7	2	16	6	16

×

เมตริกซ์ B^T

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

=

เมตริกซ์ C

719	798	661	470	567	796	983	694
545	491	406	475	726	767	589	682

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

เมตริกซ์ A × เมตริกซ์ B^T = เมตริกซ์ C

5	3	5	16	13	1	13	9
3	8	2	13	17	14	5	2

×

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

=

744	667	438	400	513	742	863	571
669	641	551	454	534	681	716	539

ขั้นตอนที่ 3 ส่งผลคูณเมตริกซ์ C คืนให้หน่วยประมวลผลหลัก

เมตริกซ์ C (Worker)

P_1	606	646	483	344	557	658	749	435
	690	698	597	516	483	747	851	685
P_2	802	691	651	478	846	933	946	710
	515	579	416	425	556	587	601	576
P_3	719	798	661	470	567	796	983	694
	545	491	406	475	726	767	589	682
P_4	744	667	438	400	513	742	863	571
	669	641	551	454	534	681	716	539

เมตริกซ์ C (Master)

606	646	483	344	557	658	749	435
690	698	597	516	483	747	851	685
802	691	651	478	846	933	946	710
515	579	416	425	556	587	601	576
719	798	661	470	567	796	983	694
545	491	406	475	726	767	589	682
744	667	438	400	513	742	863	571
669	641	551	454	534	681	716	539

ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี (MPI – C like) โดยวิธี rbmtm_w แสดงในรูปที่ 3.5

Master Processor

Start MPI

Create Data (A and B)

Perform Transpose Matrix B

Send Data to Workers

```
MPI_Isend(&low_bound, 1, MPI_INT, i, MASTER_TO_SLAVE_TAG, MPI_COMM_WORLD, &request);
MPI_Isend(&upper_bound, 1, MPI_INT, i, MASTER_TO_SLAVE_TAG + 1, MPI_COMM_WORLD, &request);
MPI_Isend(&mat_a[low_bound][0], (upper_bound - low_bound) * NUM_COLUMNS_A, MPI_DOUBLE, i,
MASTER_TO_SLAVE_TAG + 2, MPI_COMM_WORLD, &request);
MPI_Bcast(&mat_bT, NUM_ROWS_B*NUM_COLUMNS_B, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

Wait for Result

```
MPI_Recv(&low_bound, 1, MPI_INT, i, SLAVE_TO_MASTER_TAG, MPI_COMM_WORLD, &status);
MPI_Recv(&upper_bound, 1, MPI_INT, i, SLAVE_TO_MASTER_TAG + 1, MPI_COMM_WORLD, &status);
MPI_Recv(&mat_result[low_bound][0], (upper_bound - low_bound) * NUM_COLUMNS_B, MPI_DOUBLE, i,
SLAVE_TO_MASTER_TAG + 2, MPI_COMM_WORLD, &status);
```

Worker Processor

Receive Data from Master

```
MPI_Recv(&low_bound, 1, MPI_INT, 0, MASTER_TO_SLAVE_TAG, MPI_COMM_WORLD, &status);
MPI_Recv(&upper_bound, 1, MPI_INT, 0, MASTER_TO_SLAVE_TAG + 1, MPI_COMM_WORLD, &status);
MPI_Recv(&mat_a[low_bound][0], (upper_bound - low_bound) * NUM_COLUMNS_A, MPI_DOUBLE, 0,
MASTER_TO_SLAVE_TAG + 2, MPI_COMM_WORLD, &status);
```

Perform Matrix Multiplication

```
for (i = low_bound; i < upper_bound; i++) {
  for (j = 0; j < NUM_COLUMNS_B; j++) {
      for (k = 0; k < NUM_ROWS_B; k++)
                mat_result[i][j] += (mat_a[i][k] * mat_bT[j][k]);
  }
}
```

Send Data to Master

```
MPI_Isend(&low_bound, 1, MPI_INT, 0, SLAVE_TO_MASTER_TAG, MPI_COMM_WORLD, &request);
MPI_Isend(&upper_bound, 1, MPI_INT, 0, SLAVE_TO_MASTER_TAG + 1, MPI_COMM_WORLD, &request);
MPI_Isend(&mat_result[low_bound][0], (upper_bound - low_bound) * NUM_COLUMNS_B, MPI_DOUBLE, 0,
SLAVE_TO_MASTER_TAG + 2, MPI_COMM_WORLD, &request);
```

รูปที่ 3.5 ตัวอย่างรหัสเหมือน เอ็มพีไอ-ซี (MPI – C like) ของวิธี rbmtm_w

3.2.2 การหาผลคูณเมตริกซ์ทรานสโพสแบบขนาน ด้วยวิธี cbmtm_w (checkerboard - block partitioning matrix - transpose multiplication with replicated data)

วิธีการหาผลคูณเมตริกซ์แบบขนานด้วยวิธี cbmtm_w ประกอบด้วยหน่วยประมวลผล 2 ชนิดด้วยกัน คือ หน่วยประมวลผลหลัก (Master Processor) จำนวน 1 หน่วยประมวลผล และหน่วยประมวลผลทำงาน (Worker Processor) จำนวน 4 หน่วยประมวลผล ซึ่งมีขั้นตอนวิธีดังต่อไปนี้

<u>หน่วยประมวลผลหลัก จำนวน 1 หน่วยประมวลผล</u>

ขั้นตอนที่ 1 ทำการแบ่งข้อมูลตามแถวของเมตริกซ์ A ขนาด $k = \frac{n}{2}$ แถว และแบ่งข้อมูลตามแถวของเมตริซ์ B ขนาด $q = \frac{n}{2}$ แถว เพื่อทำให้การแบ่งภาระงานไปสู่หน่วยประมวลผลทำงานจะมีขนาดเท่าๆกัน (ดังรูป 3.6) สำหรับหน่วยประมวลผลทำงานทีมี 4 หน่วยประมวลผล เพื่อคำนวณหาผลคูณเมตริกซ์ C แบบขนาน

ขั้นตอนที่ 2 แจ้งช่วง และขนาดของเมตริกซ์ A และเมตริกซ์ B ที่แต่ละหน่วยประมวลผลทำงานต้องทำการอ่านจากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก นำมาเก็บไว้ในหน่วยความจำของตัวเอง

ขั้นตอนที่ 3 รอรับผลการคูณเมตริกซ์ C จากหน่วยประมวลผลทำงานทั้ง 4 หน่วยประมวลผล

<u>หน่วยประมวลผลทำงาน จำนวน 4 หน่วยประมวลผล</u>

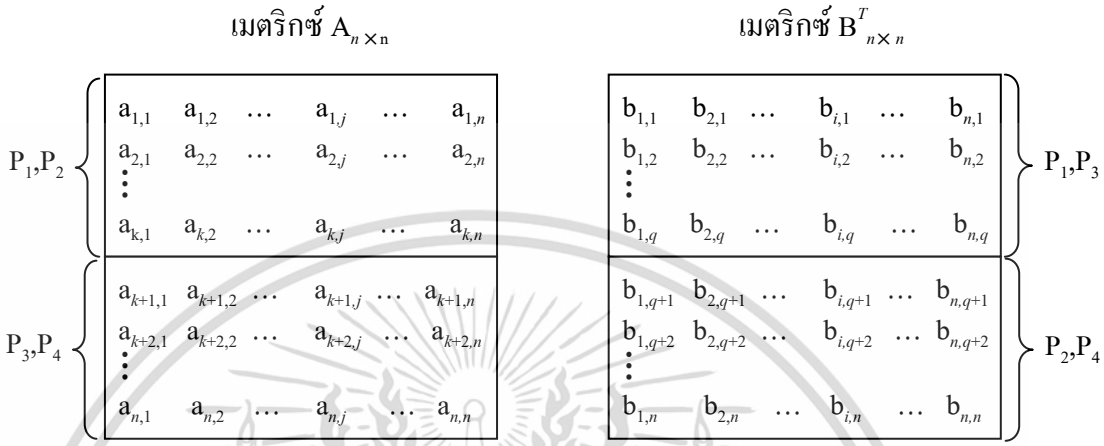
ขั้นตอนที่ 1 รอรับการติดต่อจากหน่วยประมวลผลหลัก และอ่านข้อมูลบางส่วนของเมตริกซ์ A ขนาด $k = \frac{n}{2}$ แถว และบ่างส่วนของเมตริกซ์ B ขนาด $q = \frac{n}{2}$ แถว จากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก

<u>หมายเหตุ</u> ในกรณีที่มี 4 หน่วยประมวลผล

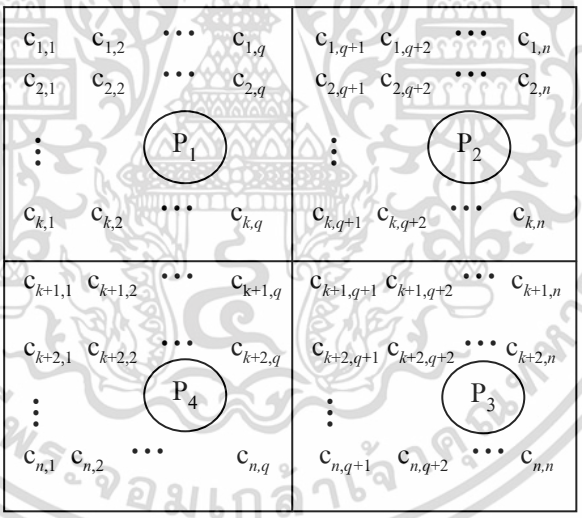
- หน่วยประมวลผลที่ 1 และ 2 จะใช้ข้อมูลเมตริกซ์ A ซ้ำกัน คือแถวที่ 1 ถึงแถวที่ $\frac{n}{2}$ และหน่วยประมวลผลที่ 3 และ 4 จะใช้ข้อมูลเมตริกซ์ A ในแถวถัดไป
- หน่วยประมวลผลที่ 1 และ 3 จะใช้ข้อมูลเมตริกซ์ B ซ้ำกัน คือแถวที่ 1 ถึงแถวที่ $\frac{n}{2}$ และหน่วยประมวลผลที่ 2 และ 4 จะใช้ข้อมูลเมตริกซ์ B ในแถวถัดไป ดังรูปที่ 3.6

ขั้นตอนที่ 2 หน่วยประมวลผลแต่ละหน่วยประมวลผล ทำการคำนวณผลคูณเมตริกซ์ C จากข้อมูลที่ได้รับมอบหมาย

ขั้นตอนที่ 3 ทำการส่งผลคูณของเมตริกซ์ C กลับไปที่หน่วยประมวลผลหลัก ดังรูปที่ 3.7



รูปที่ 3.6 การแบ่งข้อมูลเมตริกซ์ A และ B สำหรับ 4 หน่วยประมวลผลของวิธี cbmtm_w



รูปที่ 3.7 การแบ่งงานหาผลคูณเมตริกซ์ C ที่มี 4 หน่วยประมวลผล ด้วยวิธี cbmtm_w

ตัวอย่างที่ 3.2 การหาผลคูณเมตริกซ์ทรานสโพสแบบขนาน ระหว่างเมตริกซ์ $A_{8\times8}$ และ เมตริกซ์ $B^T{}_{8\times8}$ วิธี cbmtm_w โดยใช้หน่วยประมวลผลจำนวน 4 หน่วยประมวลผล

เมตริกซ์ A

$$\begin{pmatrix} 3 & 6 & 7 & 17 & 11 & 8 & 1 & 8 \\ 13 & 12 & 3 & 11 & 11 & 8 & 14 & 0 \\ 1 & 1 & 10 & 15 & 17 & 15 & 7 & 12 \\ 12 & 13 & 9 & 1 & 3 & 2 & 4 & 17 \\ 19 & 11 & 7 & 15 & 7 & 3 & 9 & 8 \\ 3 & 11 & 3 & 7 & 2 & 16 & 6 & 16 \\ 5 & 3 & 5 & 16 & 13 & 1 & 13 & 9 \\ 3 & 8 & 2 & 13 & 17 & 14 & 5 & 2 \end{pmatrix}$$

เมตริกซ์ B^T

$$\begin{pmatrix} 6 & 8 & 18 & 4 & 4 & 7 & 14 & 13 \\ 7 & 16 & 0 & 12 & 1 & 4 & 3 & 3 \\ 3 & 9 & 12 & 2 & 15 & 11 & 17 & 4 \\ 11 & 14 & 6 & 1 & 6 & 13 & 19 & 2 \\ 18 & 15 & 11 & 9 & 4 & 6 & 11 & 6 \\ 4 & 0 & 14 & 8 & 19 & 16 & 7 & 16 \\ 12 & 4 & 2 & 10 & 8 & 17 & 14 & 18 \\ 12 & 7 & 1 & 7 & 16 & 13 & 7 & 13 \end{pmatrix}$$

<u>วิธีทำ</u>

ขั้นตอนที่ 1 หน่วยประมวลผลหลักแบ่งข้อมูลตามแถวของเมตริกซ์ A และเมตริกซ์ B ให้แต่ละหน่วยประมวลผล

เมตริกซ์ A

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

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

หน่วยประมวลผลที่ 1 จำนวน 4 แถว

หน่วยประมวลผลที่ 3 จำนวน 4 แถว

หน่วยประมวลผลที่ 2 จำนวน 4 แถว

หน่วยประมวลผลที่ 4 จำนวน 4 แถว

ขั้นตอนที่ 2 หน่วยประมวลผลแต่ละหน่วย ทำการคำนวณหาผลคูณของเมตริกซ์

หน่วยประมวลผลที่ 1 (P_1) ทำการหาผลคูณของเมตริกซ์ C ขนาด 4×4

เช่น $C_{11} = (3\times6) + (6\times8) + (7\times18) + (17\times4) + (11\times4) + (8\times7) + (1\times14) + (8\times13) = 606$

เมตริกซ์ A

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

× เมตริกซ์ B^T

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

= เมตริกซ์ C

606	646	483	344
690	698	597	516
802	691	651	478
515	579	416	425

หน่วยประมวลผลที่ 2 (P_2) ทำการหาผลคูณของเมตริกซ์ C ขนาด 4×4

เมตริกซ์ A

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

× เมตริกซ์ B^T

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

= เมตริกซ์ C

557	658	749	435
483	747	851	685
846	933	946	710
556	587	601	576

หน่วยประมวลผลที่ 3 (P_3) ทำการหาผลคูณของเมตริกซ์ C ขนาด 4×4

เมตริกซ์ A

19	11	7	15	7	3	9	8
3	11	3	7	2	16	6	16
5	3	5	16	13	1	13	9
3	8	2	13	17	14	5	2

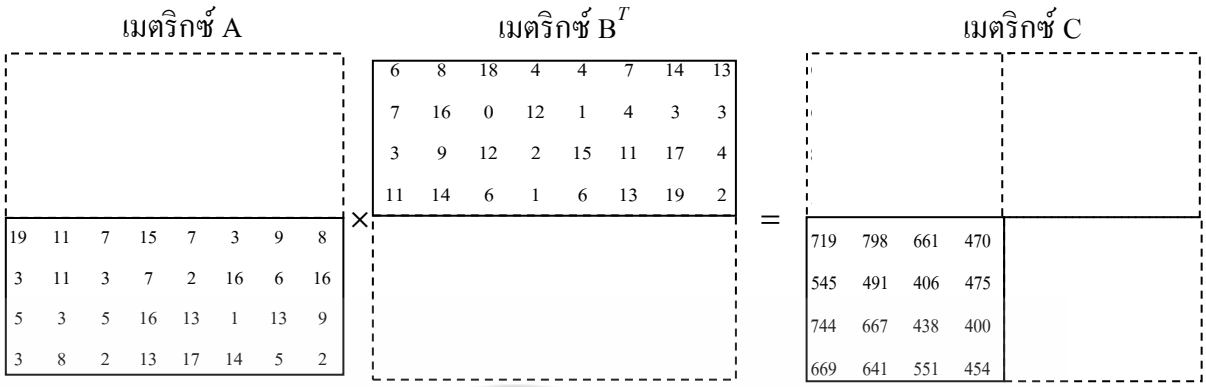
× เมตริกซ์ B^T

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

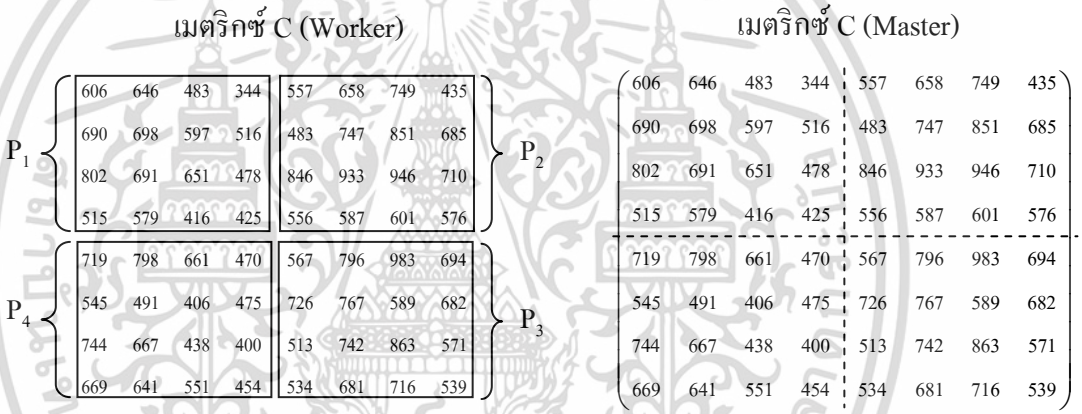
= เมตริกซ์ C

567	796	983	694
726	767	589	682
513	742	863	571
534	681	716	539

หน่วยประมวลผลที่ 4 (P_4) ทำการหาผลคูณของเมตริกซ์ C ขนาด 4×4



ขั้นตอนที่ 3 ส่งผลคูณของเมตริกซ์ C คืนให้หน่วยประมวลผลหลัก



ตัวอย่าง รหัสเหมือนเอ็มพีไอ-ซี (MPI – C like) โดยวิธี cbmtm_w แสดงในรูปที่ 3.8

Master Processor

Start MPI

Create Data (A and B)

Perform Transpose Matrix B

Send Data to Workers

```
MPI_Isend(&low_boundA, 1, MPI_INT, i, MASTER_TO_SLAVE_TAG, MPI_COMM_WORLD, &request);
MPI_Isend(&upper_boundA, 1, MPI_INT, i, MASTER_TO_SLAVE_TAG + 1, MPI_COMM_WORLD, &request);
MPI_Isend(&mat_a[low_boundA][0], (upper_boundA - low_boundA) * NUM_COLUMNS_A, MPI_DOUBLE, i,
    MASTER_TO_SLAVE_TAG + 2, MPI_COMM_WORLD, &request);
MPI_Isend(&low_boundB, 1, MPI_INT, i, MASTER_TO_SLAVE_TAG, MPI_COMM_WORLD, &request);
MPI_Isend(&upper_boundB, 1, MPI_INT, i, MASTER_TO_SLAVE_TAG + 1, MPI_COMM_WORLD, &request);
MPI_Isend(&mat_bT[low_boundB][0], (upper_boundB - low_boundB) * NUM_COLUMNS_B, MPI_DOUBLE, i,
    MASTER_TO_SLAVE_TAG + 2, MPI_COMM_WORLD, &request);
```

Wait for Result

```
MPI_Recv(&mat_result_C1[0][0], PARTION*PARTION, MPI_DOUBLE, 1, SLAVE_TO_MASTER_TAG,
    MPI_COMM_WORLD, &status);
MPI_Recv(&mat_result_C2[0][0], PARTION*PARTION, MPI_DOUBLE, 2, SLAVE_TO_MASTER_TAG + 1,
    MPI_COMM_WORLD, &status);
MPI_Recv(&mat_result_C3[0][0], PARTION*PARTION, MPI_DOUBLE, 3, SLAVE_TO_MASTER_TAG + 2,
    MPI_COMM_WORLD, &status);
MPI_Recv(&mat_result_C4[0][0], PARTION*PARTION, MPI_DOUBLE, 4, SLAVE_TO_MASTER_TAG + 3,
    MPI_COMM_WORLD, &status);
```

Worker Processor

Receive Data from Master

```
MPI_Recv(&low_boundA, 1, MPI_INT, 0, MASTER_TO_SLAVE_TAG, MPI_COMM_WORLD, &status);
MPI_Recv(&upper_boundA, 1, MPI_INT, 0, MASTER_TO_SLAVE_TAG + 1, MPI_COMM_WORLD, &status);
MPI_Recv(&mat_a[low_boundA][0], (upper_boundA - low_boundA) * NUM_COLUMNS_A, MPI_DOUBLE, 0,
    MASTER_TO_SLAVE_TAG + 2, MPI_COMM_WORLD, &status);
MPI_Recv(&low_boundB, 1, MPI_INT, 0, MASTER_TO_SLAVE_TAG, MPI_COMM_WORLD, &status);
MPI_Recv(&upper_boundB, 1, MPI_INT, 0, MASTER_TO_SLAVE_TAG + 1, MPI_COMM_WORLD, &status);
MPI_Recv(&mat_bT[low_boundB][0], (upper_boundB - low_boundB) * NUM_COLUMNS_B, MPI_DOUBLE, 0,
    MASTER_TO_SLAVE_TAG + 2, MPI_COMM_WORLD, &status);
```

Perform Matrix Multiplication

```
for (i = low_bound; i < upper_bound; i++) {
    for (j = 0; j < NUM_COLUMNS_B; j++) {
        for (k = 0; k < NUM_ROWS_B; k++)
            mat_result_C[i%PARTION][j%PARTION] += (mat_a[i][k] * mat_bT[j][k]);
    }
}
```

Send Data to Master

```
MPI_Isend(&mat_result_C1[0][0], PARTION*PARTION, MPI_DOUBLE, 0, SLAVE_TO_MASTER_TAG ,
    MPI_COMM_WORLD, &request);
MPI_Isend(&mat_result_C2[0][0], PARTION*PARTION, MPI_DOUBLE, 0, SLAVE_TO_MASTER_TAG + 1,
    MPI_COMM_WORLD, &request);
MPI_Isend(&mat_result_C3[0][0], PARTION*PARTION, MPI_DOUBLE, 0, SLAVE_TO_MASTER_TAG + 2,
    MPI_COMM_WORLD, &request);
MPI_Isend(&mat_result_C4[0][0], PARTION*PARTION, MPI_DOUBLE, 0, SLAVE_TO_MASTER_TAG + 3,
    MPI_COMM_WORLD, &request);
```

รูปที่ 3.8 ตัวอย่างรหัสเหมือน เอ็มพีไอ-ซี (MPI – C like) ของวิธี cbmtm_w

3.2.3 การหาผลคูณเมตริกซ์ทรานสโพสแบบขนาน ด้วยวิธี rbmtm_w/o (row - block partitioning matrix - transpose multiplication without replicated data)

วิธีการหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmtm_w/o ประกอบด้วยหน่วยประมวลผล 2 ชนิดด้วยกัน คือ หน่วยประมวลผลหลัก (Master Processor) จำนวน 1 หน่วยประมวลผล และหน่วยประมวลผลทำงาน (Worker Processor) จำนวน 4 หน่วยประมวลผล ซึ่งมีขั้นตอนวิธีดังต่อไปนี้

<u>หน่วยประมวลผลหลัก จำนวน 1 หน่วยประมวลผล</u>

ขั้นตอนที่ 1 ทำการแบ่งข้อมูลตามแถวของเมตริกซ์ A และ B ขนาด $k = \frac{n}{4}$ แถว เพื่อให้แต่ละภาระงาน (Work Load) มีขนาดเท่าๆกัน สำหรับหน่วยประมวลผลทำงาน 4 หน่วยประมวลผล เพื่อคำนวณหาผลคูณเมตริกซ์ C จำนวน $\frac{n}{4}$ แถวแบบขนาน (ดังรูป 3.9)

ขั้นตอนที่ 2 แจ้งช่วง และขนาดของเมตริกซ์ A และเมตริกซ์ B ที่แต่ละหน่วยประมวลผลทำงานต้องทำการอ่านจากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก นำมาเก็บไว้ในหน่วยความจำของตัวเอง

ขั้นตอนที่ 3 รอรับผลคูณเมตริกซ์ C จากหน่วยประมวลผลทำงาน 4 หน่วยประมวลผล

<u>หน่วยประมวลผลทำงาน จำนวน 4 หน่วยประมวลผล</u>

ขั้นตอนที่ 1 รอรับการติดต่อจากหน่วยประมวลผลหลัก และอ่านข้อมูลบางส่วนของเมตริกซ์ A และ B จากหน่วยความจำเสมือนของหน่วยประมวลผลหลัก ขนาด $k = \frac{n}{4}$ แถว คือแถวที่ (i-1)k+1 ถึงแถวที่ ik เมื่อ i คือ ไอดีของหน่วยประมวลผลทำงาน (Worker ID)

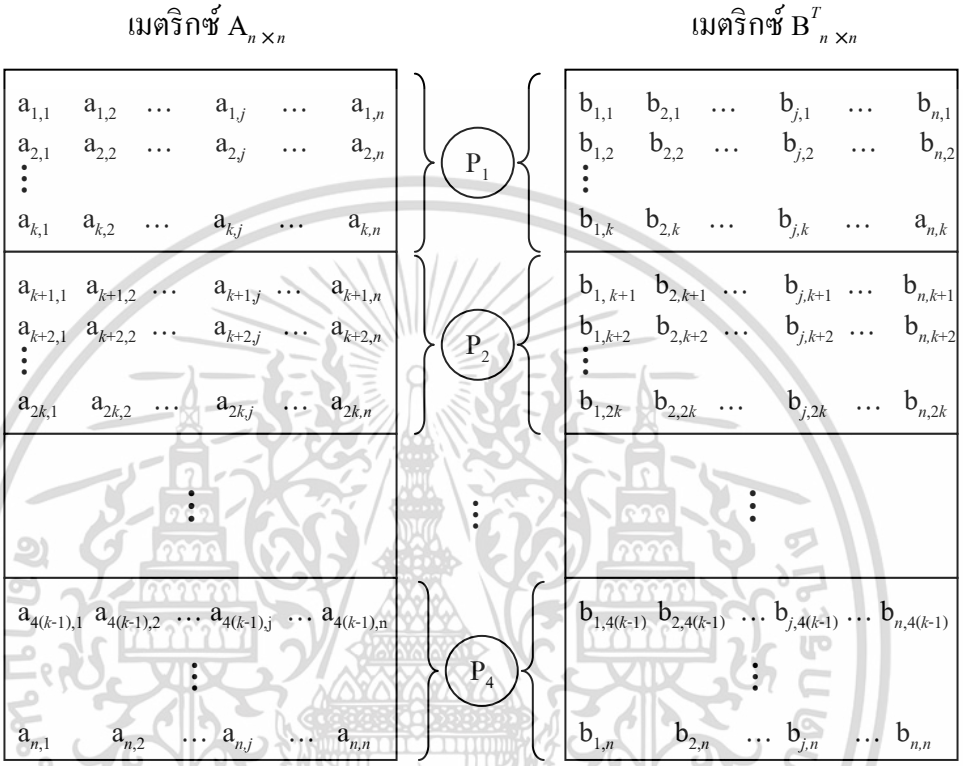
ขั้นตอนที่ 2 ทำการคำนวณผลคูณเมตริกซ์ C บางค่าในแถวที่แต่ละหน่วยประมวลผลได้รับมอบหมาย โดยจะมีขั้นตอนย่อยๆ ดังต่อไปนี้

ขั้นตอนที่ 2.1 คำนวณผลคูณเมตริกซ์ C ในแถวที่ (i-1)k+1 ถึงแถวที่ ik

ขั้นตอนที่ 2.2 ทำการส่งค่าของเมตริกซ์ B แถวที่ (i-1)k+1 ถึงแถวที่ ik ให้หน่วยประมวลผลทำงานที่อยู่ลำดับก่อนหน้า (P_{i-1}) และทำการรับข้อมูลของเมตริกซ์ B แถวที่ ik+1 ถึงแถวที่ ik+k จากหน่วยประมวลผลทำงานลำดับถัดไป (P_{i+1})

ขั้นตอนที่ 2.3 ทำซ้ำในขั้นตอนที่ 2.1 และ 2.2 จนได้ผลคูณของเมตริกซ์ C ครบทุกแถวที่ได้รับมอบภาระงานจากหน่วยประมวลผลหลัก

ขั้นตอนที่ 3 ส่งผลคูณของเมตริกซ์ C กลับไปยังหน่วยประมวลผลหลัก



รูปที่ 3.9 การแบ่งข้อมูลเมตริกซ์ A และ B สำหรับ 4 หน่วยประมวลผลของวิธี rbmtm_w/o

ตัวอย่างที่ 3.3 การหาผลคูณเมตริกซ์ทรานสโพสแบบขนาน ระหว่างเมตริกซ์ $A_{8\times8}$ และ เมตริกซ์ $B^T{}_{8\times8}$ วิธี rbmtm_w/o โดยใช้หน่วยประมวลผลจำนวน 4 หน่วยประมวลผล

เมตริกซ์ A

$$\begin{pmatrix} 3 & 6 & 7 & 17 & 11 & 8 & 1 & 8 \\ 13 & 12 & 3 & 11 & 11 & 8 & 14 & 0 \\ 1 & 1 & 10 & 15 & 17 & 15 & 7 & 12 \\ 12 & 13 & 9 & 1 & 3 & 2 & 4 & 17 \\ 19 & 11 & 7 & 15 & 7 & 3 & 9 & 8 \\ 3 & 11 & 3 & 7 & 2 & 16 & 6 & 16 \\ 5 & 3 & 5 & 16 & 13 & 1 & 13 & 9 \\ 3 & 8 & 2 & 13 & 17 & 14 & 5 & 2 \end{pmatrix}$$

เมตริกซ์ B^T

$$\begin{pmatrix} 6 & 8 & 18 & 4 & 4 & 7 & 14 & 13 \\ 7 & 16 & 0 & 12 & 1 & 4 & 3 & 3 \\ 3 & 9 & 12 & 2 & 15 & 11 & 17 & 4 \\ 11 & 14 & 6 & 1 & 6 & 13 & 19 & 2 \\ 18 & 15 & 11 & 9 & 4 & 6 & 11 & 6 \\ 4 & 0 & 14 & 8 & 19 & 16 & 7 & 16 \\ 12 & 4 & 2 & 10 & 8 & 17 & 14 & 18 \\ 12 & 7 & 1 & 7 & 16 & 13 & 7 & 13 \end{pmatrix}$$

<u>**วิธีทำ**</u>

ขั้นตอนที่ 1 หน่วยประมวลผลหลักแบ่งข้อมูลตามแถวของเมตริกซ์ A และเมตริกซ์ B ให้แต่ละหน่วยประมวลผล

เมตริกซ์ A

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

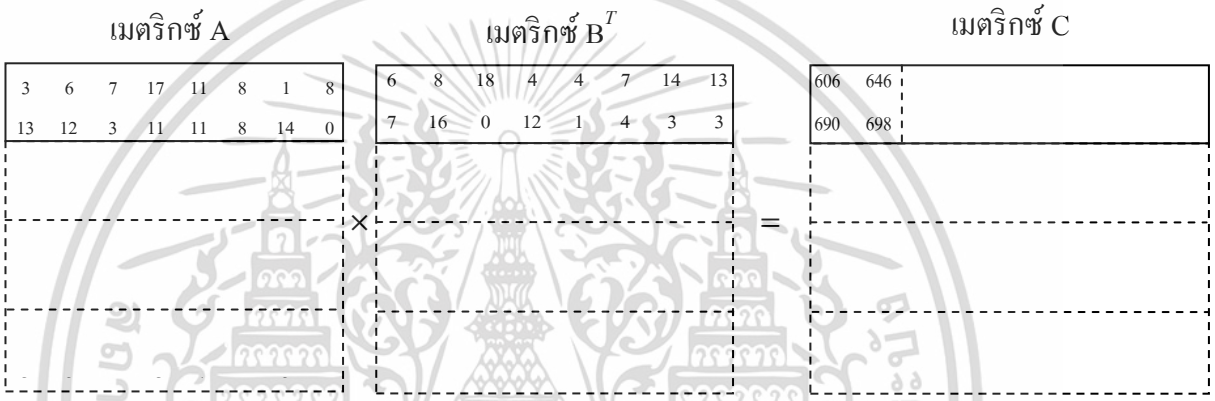
เมตริกซ์ B^T

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

ขั้นตอนที่ 2 หน่วยประมวลผลแต่ละหน่วยทำการคำนวณหาผลคูณของเมตริกซ์ โดยมีรายละเอียดขั้นตอนย่อยดังนี้

ขั้นตอนที่ 2.1 ทำการคำนวณผลคูณเมตริกซ์ C บางค่าในแถวที่ $(i-1)k+1$ ถึงแถวที่ ik หลักที่ $(i-1)k+1$ ถึงหลักที่ ik

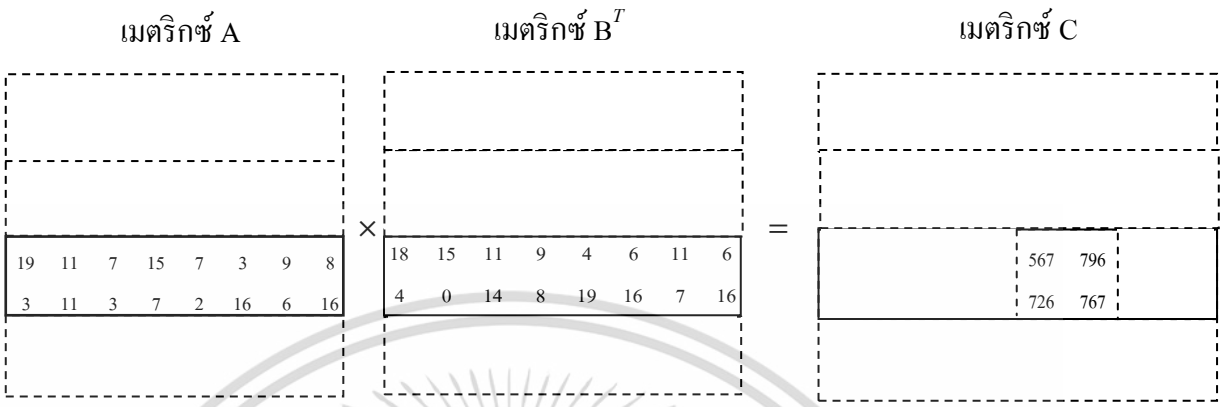
หน่วยประมวลผลที่ 1(P_1) ทำการหาผลคูณของเมตริกซ์ C ขนาด 2×2 คือ บางส่วนของแถวที่ 1 และแถวที่ 2



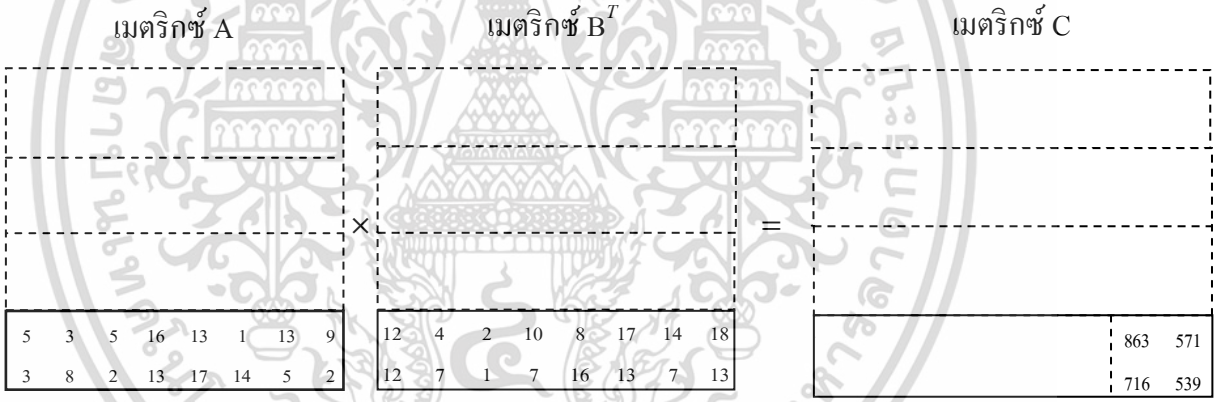
เช่น $C_{11} = (3\times6) + (6\times8) + (7\times18) + (17\times4) + (11\times4) + (8\times7) + (1\times14) + (8\times13) = 606$

หน่วยประมวลผลที่ 2(P_2) ทำการหาผลคูณของเมตริกซ์ C ขนาด 2×2 คือ บางส่วนของแถวที่ 3 และแถวที่ 4

หน่วยประมวลผลที่ 3(P_3) ทำการหาผลคูณของเมตริกซ์ C ขนาด 2×2 คือ บางส่วนของแถวที่ 5 และแถวที่ 6



หน่วยประมวลผลที่ 3(P_3) ทำการหาผลคูณของเมตริกซ์ C ขนาด 2×2 คือ บางส่วนของแถวที่ 5 และแถวที่ 6



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 C คืนให้หน่วยประมวลผลหลัก

ตัวอย่าง รหัสเหมือนเอ็มพีไอ-ซี (MPI – C like) โดยวิธี rbmtm_w/o แสดงในรูปที่ 3.10

Master Processor

Start MPI

Create Data (A and B)

Perform Transpose Matrix B

Send Data to Workers

```
MPI_Isend(&low_boundA, 1, MPI_INT, i, MASTER_TO_SLAVE_TAG, MPI_COMM_WORLD, &request);
MPI_Isend(&upper_boundA, 1, MPI_INT, i, MASTER_TO_SLAVE_TAG + 1, MPI_COMM_WORLD, &request);
MPI_Isend(&low_boundB, 1, MPI_INT, i, MASTER_TO_SLAVE_TAG+2, MPI_COMM_WORLD, &request);
MPI_Isend(&upper_boundB, 1, MPI_INT, i, MASTER_TO_SLAVE_TAG + 3, MPI_COMM_WORLD, &request);
MPI_Isend(&mat_a[low_boundA][0], (upper_boundA - low_boundA) * NUM_COLUMNS_A, MPI_DOUBLE, i,
    MASTER_TO_SLAVE_TAG + 4, MPI_COMM_WORLD, &request);
MPI_Isend(&mat_bT[low_boundB][0], (upper_boundB - low_boundB) * NUM_COLUMNS_B, MPI_DOUBLE, i,
    MASTER_TO_SLAVE_TAG + 5, MPI_COMM_WORLD, &request);
```

Wait for Result

```
MPI_Recv(&low_boundA, 1, MPI_INT, i, SLAVE_TO_MASTER_TAG, MPI_COMM_WORLD, &status);
MPI_Recv(&mat_result[low_boundA][0], portion* NUM_COLUMNS_B, MPI_DOUBLE, i,
   SLAVE_TO_MASTER_TAG + 1 , MPI_COMM_WORLD, &status);
```

Worker Processor

Receive Data from Master

```
MPI_Isend(&low_boundB, 1, MPI_INT, rank-1, SLAVE_TO_SLAVE_TAG+1, MPI_COMM_WORLD, &request);
MPI_Isend(&mat_bT[low_boundB][0],portion * NUM_COLUMNS_A, MPI_DOUBLE, rank-1,
     SLAVE_TO_SLAVE_TAG , MPI_COMM_WORLD,   &request);
MPI_Recv(&low_boundB, 1, MPI_INT, rank+1, SLAVE_TO_SLAVE_TAG+ 1, MPI_COMM_WORLD, &status);
MPI_Recv(&mat_bT[low_boundB][0], portion* NUM_COLUMNS_B, MPI_DOUBLE, rank+1,
    SLAVE_TO_SLAVE_TAG, MPI_COMM_WORLD, &status);
```

Perform Matrix Multiplication

```
for (i = low_boundA; i < upper_boundA; i++) {
     for (j = low_boundB; j < upper_boundB; j++) {
          for (k = 0; k <NUM_COLUMNS_B; k++)
               mat_result[i][j] += (mat_a[i][k] * mat_bT[j][k]);
     }
}
```

Send Data to Master

```
MPI_Isend(&low_boundA, 1, MPI_INT, 0,  SLAVE_TO_MASTER_TAG, MPI_COMM_WORLD, &request);
MPI_Isend(&mat_result[low_boundA][0], portion* NUM_COLUMNS_B, MPI_DOUBLE, 0,
     SLAVE_TO_MASTER_TAG + 1 , MPI_COMM_WORLD, &request);
```

รูปที่ 3.10 ตัวอย่างรหัสเหมือน เอ็มพีไอ-ซี (MPI – C like) ของวิธี rbmtm_w/o

บทที่ 4

การทดลองและผลการทดลอง

วิทยานิพนธ์นี้เป็นการนำเสนอการออกแบบขั้นตอนวิธีการหาผลคูณของเมตริกซ์ทรานสโพสแบบขนาน โดยวิธีต่างๆที่นำเสนอเป็นวิธีที่ปรับปรุงประสิทธิภาพทั้งในด้านเวลาที่ใช้ในการประมวลผล และการใช้ข้อมูลที่ซ้ำซ้อนกันในแต่ละหน่วยประมวลผลเนื้อหาในบทนี้ จะเป็นผลการพัฒนาโปรแกรมการหาผลคูณเมตริกซ์ทรานสโพสแบบขนานดังกล่าว 3 วิธี คือ 1) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w (row-block partitioning matrix-transpose multiplication with replicated data) 2) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี cbmtm_w (checkerboard-block partitioning matrix-transpose multiplication with replicated data) 3) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w/o (row-block partitioning matrix-transpose multiplication withoutreplicated data) โดยมีการทดลอง และผลการทดลองดังต่อไปนี้

4.1 ระบบคอมพิวเตอร์ที่ใช้ในการทดลอง

ระบบคอมพิวเตอร์ที่ใช้ในการทดลองจะไม่มีภาระงานอื่นใดประมวลผลอยู่ในระบบคอมพิวเตอร์ที่ใช้ในการทดลอง มีส่วนประกอบพื้นฐานของระบบซึ่งประกอบด้วย ส่วนประกอบด้านฮาร์ดแวร์ ระบบปฏิบัติการ และไลบารีเอ็มพีไอซีเอช(MPICH)สำหรับการเขียนโปรแกรมแบบขนาน

1) ส่วนประกอบด้านฮาร์ดแวร์

โครงสร้างทางฮาร์ดแวร์ของระบบที่ใช้ในการทดลอง โดยใช้เครื่องคอมพิวเตอร์ชนิดที่มี 4 หน่วยประมวลผลในเครื่องเดียวกัน (Quad-Core Processor) แยกกันทำงานอย่างอิสระ ซึ่งมีองค์ประกอบทางด้านฮาร์ดแวร์ที่จำเป็นดังนี้

- หน่วยประมวลผล (CPUs) ในงานวิจัยนี้ใช้เครื่องคอมพิวเตอร์ชนิดที่มี 4 หน่วยประมวลผลในเครื่องเดียวกัน (Quad-Core Processor) ยีห้ออินเทลI5 (Intel I5) ความเร็วต่อรอบ 2.8 กิกะเฮิรตซ์
- แผงวงจรหลัก (Main Board)

- หน่วยความจำหลัก (RAM) ชนิด DDR3 ขนาด 4 กิกะไบต์
- หน่วยความจำสำรอง (Disk Storage) มีขนาดเท่ากับ 500 กิกะไบต์
- หน่วยความจำแคช (Cache Memory) มีขนาดเท่ากับ 1024 กิโลไบต์

2) ระบบปฏิบัติการ

ระบบปฏิบัติการจะเป็นส่วนสำคัญอย่างมากต่อการทำงานของคอมพิวเตอร์ ในวิทยานิพนธ์นี้เลือกใช้ระบบปฏิบัติการ Linux เพราะเป็นระบบปฏิบัติการที่ใช้กันอย่างกว้างขวาง และมีประสิทธิภาพสูงในการใช้ฮาร์ดแวร์ (Hardware)

3) เอมพีไอซีเอช (MPICH)

เอ็มพีไอซีเอช(MPICH) เป็นชุดคำสั่งสำหรับมาตรฐานการส่งผ่านข้อความ (Message Passing Interface: MPI) ซึ่งมีไว้สำหรับพัฒนาโปรแกรมแบบขนาน สำหรับภาษาที่สามารถนำมาใช้พัฒนาโปรแกรมแบบขนาน ด้วยเอ็มพีไอซีเอชมีหลายภาษา เช่น ฟอร์แทน (Fortran Language),โปรแกรมภาษาซี (C Language) และโปรแกรมภาษาจาวา (Java Language) เป็นต้น โดยภาษาดังกล่าวนี้มีฟังก์ชันพื้นฐานต่างๆ รวมทั้งมีคำสั่งใช้งานเพื่อทำการคอมไพล์โปรแกรมที่ได้พัฒนาขึ้น

4.2 ลักษณะของข้อมูลที่ใช้ในการทดลอง

ลักษณะข้อมูลที่ใช้ในการทดลองเป็นข้อมูลชนิดเลขจำนวนจริงละเอียด 2 เท่า (Double Precision) โดยขนาดข้อมูลของเมตริกซ์จะกำหนดให้เป็นเมตริกซ์จัตุรัส ทั้งเมตริกซ์ A และเมตริกซ์ B โดยมีขนาดดังนี้ 256×256, 512×512, 1024×1024 และ 2048×2048

4.3 ผลการทดลอง

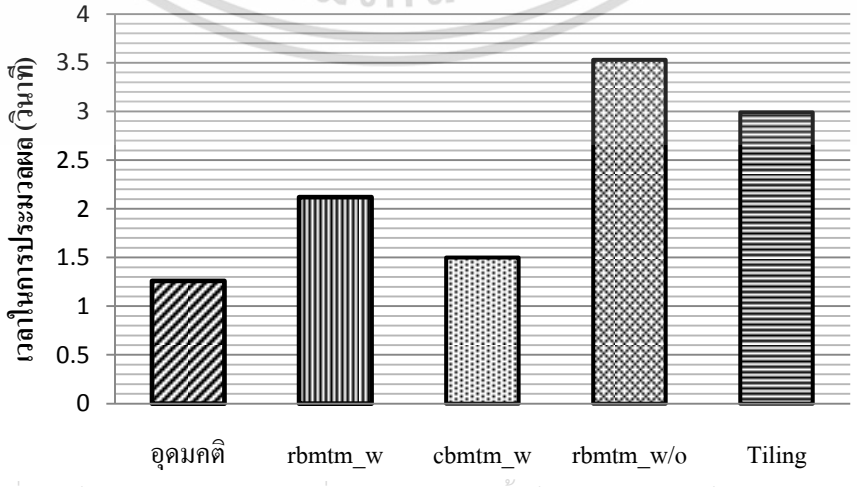
จากการพัฒนาโปรแกรม โดยไลบารี่ MPICH 3.0.3 และภาษา C (C Language) ทำการวัดประสิทธิภาพของวิธีการคูณเมตริกซ์ทรานสโพสแบบขนานทั้ง 3 วิธีที่เสนอ รวมถึงการหาผลคูณเมตริกซ์แบบอนุกรม และการหาผลคูณเมตริกซ์แบบขนานด้วยวิธีไทลิงก์ [1] เมื่อนำมาเปรียบเทียบกับการวัดผลที่ประกอบด้วย เวลาที่ใช้ในการประมวลผล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speed-Up) และประสิทธิภาพ (Efficiency) ซึ่งผลที่ได้จะแสดงได้ชัดแจนเมื่อเมตริกซ์มีขนาดใหญ่ขึ้น

4.3.1 ผลการทดลองเมื่อเปรียบเทียบกับเวลาในอุดมคติ

การทำการทดลองเปรียบเทียบเวลาในการประมวลผลจริงกับเวลาในอุดมคติเป็นการวัดประสิทธิภาพของระบบ (System Performance) ที่ใช้ในการทดลองดังนี้ ตารางที่ 4.1 และ รูปที่ 4.1 แสดงผลการเปรียบเทียบเวลาที่ใช้ในการประมวลผล (Response Time) ทั้ง 3 แบบ ($T_p = T_s + T_c$) เวลาที่ใช้ในการประมวลผลข้อมูลในอุดมคติ (Response Time of Ideal Case)จะคำนวณโดยสมมติว่าเวลาที่ใช้ในการติดต่อสื่อสาร เพื่อแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลเท่ากับศูนย์ ดังนั้น $T_p = T_s/p$ ตารางที่ 4.2 และรูปที่ 4.2 แสดงอัตราการเพิ่มขึ้นของความเร็ว (Speed-Up) ทั้ง 3 วิธี ($S_p = T_s / T_p$) เปรียบเทียบกับอัตราการเพิ่มขึ้นของความเร็ว ในอุดมคติ ($S_p = p$) ในตารางที่ 4.3 และรูปที่ 4.3 แสดงประสิทธิภาพในการหาผลคูณของเมตริกซ์ ทั้ง 3 วิธี ($E_p = S_p / p$) เปรียบเทียบกับประสิทธิภาพในอุดมคติ ($E_p = 1$)

ตารางที่ 4.1 เวลาที่ใช้ในการประมวลผลของผลคูณเมตริกซ์ทรานสโพส (Response Time) เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล และเมตริกซ์ขนาด 1024 × 1024

วิธีการหาผลคูณเมตริกซ์	เวลาในการประมวลผล (วินาที)
อุดมคติ	1.26
วิธีแบบขนาน	
rbmtm_w	2.12
cbmtm_w	1.50
rbmtm_w/o	3.53
Tiling	2.99

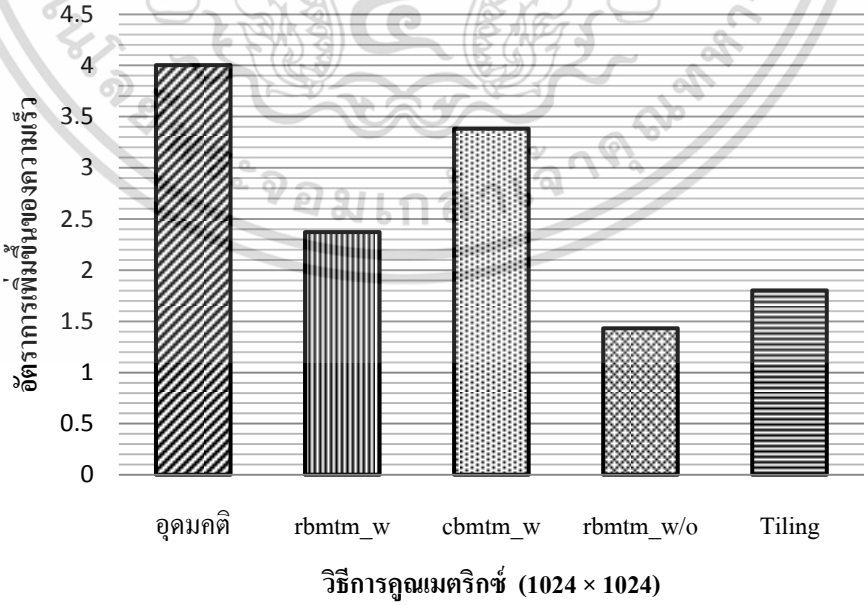


รูปที่ 4.1 เวลาที่ใช้ในการหาผลคูณของเมตริกซ์ ทั้ง 4 วิธี กับเวลาในอุดมคติเมื่อใช้ 4 หน่วยประมวลผล

จากรูปที่ 4.1 สามารถอธิบายได้ว่าเวลาที่ใช้ในการประมวลผลวิธี cbmtm_w ที่เสนอในวิทยานิพนธ์ เมื่อจำนวนหน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล จะใช้เวลาในการประมวลผล 1.50 วินาที ใกล้เคียงกับเวลาในอุดมคติมากที่สุด คือ 1.26 ซึ่งคิดเป็น 84% ส่วนวิธีที่มีประสิทธิภาพรองลงมาคือ วิธี rbmtm และ วิธีไทลิงก์ ส่วนวิธีที่ใช้เวลาในการประมวลผลมากที่สุด คือ วิธี rbmtm_w/o

ตารางที่ 4.2 อัตราการเพิ่มขึ้นของความเร็ว (Speed-Up) โดยการเปรียบเทียบกับความเร็วในอุดมคติ (Speedup of Ideal Case) เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล และเมตริกซ์ขนาด 1024 × 1024

วิธีการหาผลคูณเมตริกซ์	**เวลาในการประมวลผล (วินาที)**
อุดมคติ	4.00
วิธีแบบขนาน	
rbmtm_w	2.37
cbmtm_w	3.38
rbmtm_w/o	1.43
Tiling	1.68

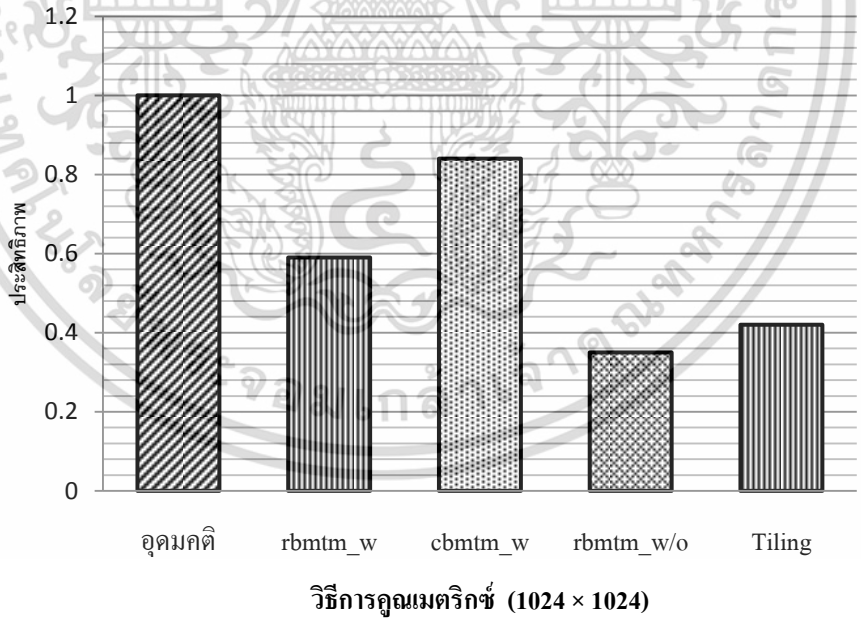


รูปที่ 4.2 อัตราการเพิ่มขึ้นของความเร็วทั้ง 4 วิธี กับเวลาในอุดมคติเมื่อใช้ 4 หน่วยประมวลผล

จากรูปที่ 4.2 สามารถอธิบายได้ว่าอัตราการเพิ่มขึ้นของความเร็วของการหาผลคูณของเมตริกซ์วิธี cbmtm_wเป็นวิธีที่ดีที่สุดซึ่งเท่ากับ3.38 เมื่อใช้ 4 หน่วยประมวลผล ส่วนวิธีที่รองลงมาคือวิธี rbmtm_wและวิธีไทลิงก์ ส่วนวิธีที่มีอัตราเพิ่มขึ้นของความเร็วน้อยที่สุดคือวิธี rbmtm_w/o

ตารางที่ 4.3 ประสิทธิภาพ (Efficiency) เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล และเมตริกซ์มีขนาด 1024 × 1024

วิธีการหาผลคูณเมตริกซ์	เวลาในการประมวลผล (วินาที)
อุดมคติ	1.00
วิธีแบบขนาน	
rbmtm_w	0.59
cbmtm_w	0.84
rbmtm_w/o	0.35
Tiling	0.42



รูปที่ 4.3 เปรียบเทียบประสิทธิภาพทั้ง 4 วิธี กับเวลาในอุดมคติเมื่อใช้ 4 หน่วยประมวลผล

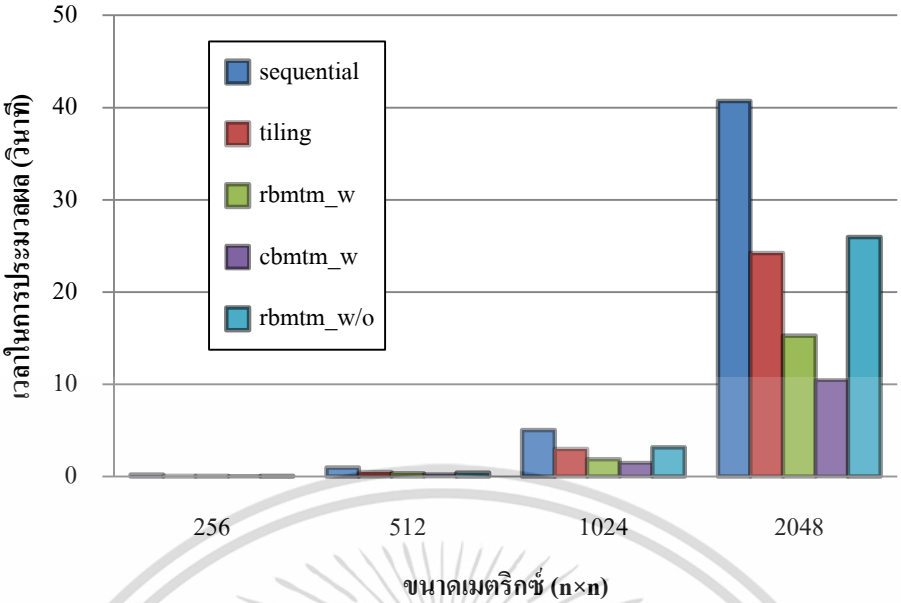
จากรูปที่ 4.3 แสดงประสิทธิภาพของการหาผลคูณของเมตริกซ์ทั้ง 3 วิธีจะเห็นว่าวิธีที่มีประสิทธิภาพมากที่สุด คือวิธี cbmtm_wมีค่าประสิทธิภาพเท่ากับ 0.84 สำหรับประสิทธิภาพวิธี rbmtm_wจะมีค่าประสิทธิภาพเท่ากับ 0.59 ซึ่งมีค่าน้อยกว่าวิธี cbmtm_wส่วนวิธีที่มีประสิทธิภาพน้อยที่สุดคือวิธี rbmtm_w/o จะมีประสิทธิภาพเท่ากับ 0.35

4.3.2 ผลการทดลองเปรียบเทียบเมื่อขนาดของเมตริกซ์ที่ใช้ทดลองมีขนาดต่างกัน

การเปรียบเทียบการหาผลคูณของเมตริกซ์ทรานสโพสเมื่อขนาดของเมตริกซ์ที่ใช้ในการทดลองมีขนาดต่างกัน โดยกำหนดให้จำนวนของหน่วยประมวลผลที่ใช้ในการคำนวณ เท่ากับ 4 หน่วยประมวลผล และทำการเปรียบเทียบทั้งวิธีการหาผลคูณของเมตริกซ์แบบอนุกรม (Sequential) และวิธีการคูณเมตริกซ์แบบไทลิงก์ กับวิธีการหาผลคูณของเมตริกซ์ทรานสโพสแบบขนานทั้ง 3 วิธี โดยใช้เมตริกซ์ขนาด 256×256, 512×512, 1024×1024 และ 2048×2048 ตามลำดับ ซึ่งผลการทดลองแสดงดังนี้ ตาราง 4.4 และรูปที่ 4.4 แสดงเวลาที่ใช้ในการประมวลผล (Response Time) และนำเวลาที่ใช้ในการประมวลผลมาคำนวณหาอัตราเพิ่มขึ้นของความเร็ว (Speed-Up) ดังแสดงในตารางที่ 4.5 และรูปที่ 4.6 จากนั้นนำค่าของอัตราการเพิ่มขึ้นของความเร็วมาคำนวณหาประสิทธิภาพ (Efficiency) ดังแสดงที่ 4.6 และรูปที่ 4.6

ตารางที่ 4.4 เวลาที่ใช้ในวิธีการหาผลคูณของเมตริกซ์ (Response Time) ทั้ง 5 วิธี

ขนาดของเมตริกซ์	วิธีแบบอนุกรม	วิธีไทลิงก์	วิธีแบบขนาน		
			rbmtm_w	cbmtm_w	rbmtm_w/o
256×256	0.208	0.079	0.077	0.053	0.084
512×512	0.996	0.492	0.373	0.257	0.412
1024×1024	5.049	2.995	1.882	1.496	3.169
2048×2048	40.718	24.236	15.274	10.469	25.981

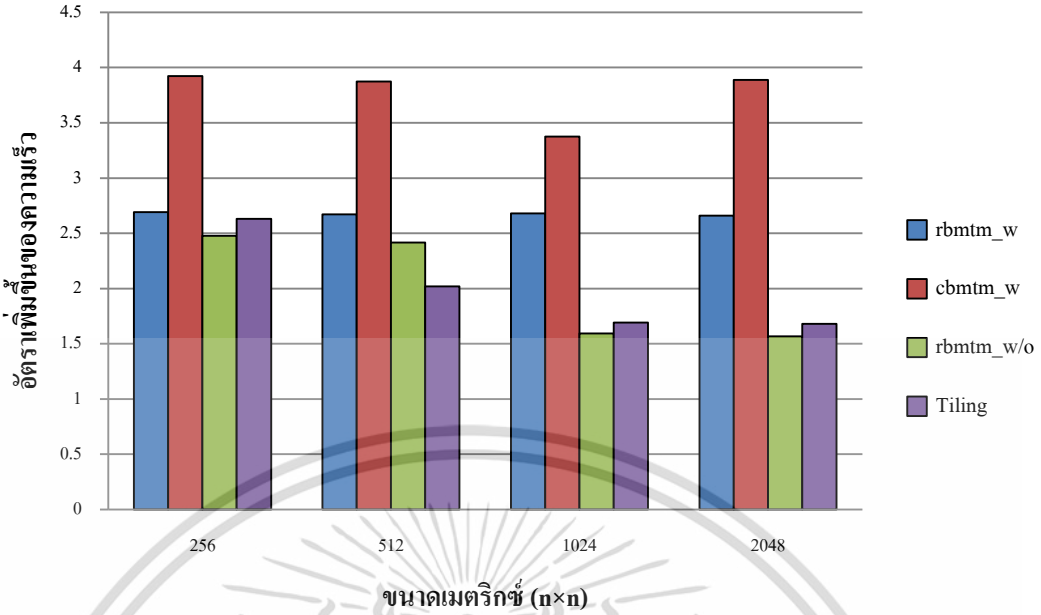


รูปที่ 4.4 เวลาที่ใช้ในการหาผลคูณเมตริกซ์ ทั้ง 5 วิธี เมื่อ P = 4

จากรูปที่ 4.4 แสดงเวลาที่ใช้ในการประมวลผลของเมตริกซ์ (Response Time) โดยทำการเปรียบเทียบเวลาทั้ง 5 วิธี สังเกตว่าเมื่อข้อมูลมีขนาดใหญ่ขึ้นเวลาที่ใช้ในการหาผลคูณของเมตริกซ์นั้นจะเพิ่มขึ้นตามไปด้วย โดยวิธี cbmtm_w ใช้เวลาในการประมวลผลรวมน้อยที่สุดเมื่อเทียบกับทุกวิธี และรองลงมาคือวิธี rbmtm_w และวิธีไทลิงก์ ส่วนวิธีที่ใช้เวลาในการประมวลผลมากที่สุดคือวิธีแบบอนุกรม (Sequential) สังเกตว่าวิธี rbmtm_w/o เป็นวิธีที่ใช้เวลาในการประมวลผลมากที่สุดใน 3 วิธีที่เสนอในวิทยานิพนธ์นี้ เนื่องจากวิธีนี้ต้องการเวลาที่ใช้ในการสื่อสารเพื่อแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล

ตารางที่ 4.5 อัตราการเพิ่มขึ้นของความเร็ว (Speed-Up) เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล โดยใช้เมตริกซ์ ขนาด 256×256, 512×512, 1024×1024 และ 2048×2048 ตามลำดับ

ขนาดของเมตริกซ์	วิธีไทลิงก์	วิธีแบบขนาน		
		rbmtm_w	cbmtm_w	rbmtm_w/o
256×256	2.63	2.69	3.924	2.476
512×512	2.02	2.67	3.875	2.417
1024×1024	1.69	2.68	3.375	1.593
2048×2048	1.68	2.66	3.889	1.567

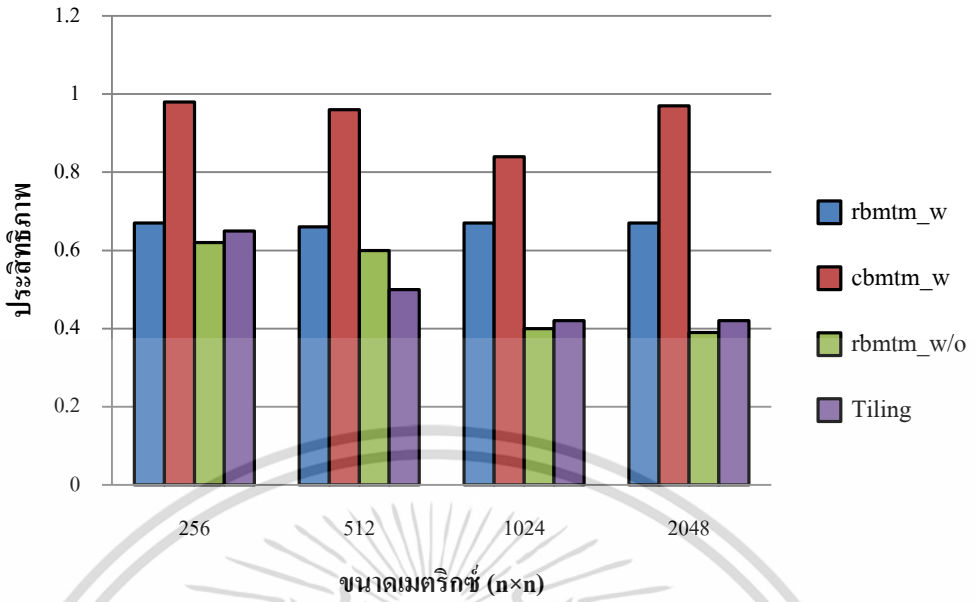


รูปที่ 4.5 อัตราการเพิ่มขึ้นของความเร็วทั้ง 4 วิธี เมื่อ P = 4

จากรูปที่ 4.5 แสดงการเปรียบเทียบอัตราการเพิ่มขึ้นของความเร็วทั้ง 4 วิธี เมื่อมีจำนวนหน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล และเมตริกซ์มีขนาด 256×256, 512×512, 1024×1024 และ 2048×2048 ตามลำดับ จะเห็นว่าวิธี cbmtm_w มีอัตราการเพิ่มขึ้นของความเร็วมากที่สุด ส่วนวิธีที่มีค่ารองลงมาคือ วิธี rbmtm_w และวิธีไทลิงก์ สำหรับวิธีที่มีอัตราเพิ่มขึ้นของความเร็วน้อยที่สุดคือ วิธี rbmtm_w/o

ตารางที่4.6 ประสิทธิภาพ (Efficiency) เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล โดยใช้เมตริกซ์ขนาด 256×256, 512×512, 1024×1024 และ 2048×2048 ตามลำดับ

ขนาดของเมตริกซ์	วิธีไทลิงก์	วิธีแบบขนาน		
		rbmtm_w	cbmtm_w	rbmtm_w/o
256×256	0.65	0.67	0.98	0.62
512×512	0.51	0.66	0.96	0.60
1024×1024	0.42	0.67	0.84	0.40
2048×2048	0.42	0.67	0.97	0.39



รูปที่ 4.6 ประสิทธิภาพทั้ง 4 วิธีเมื่อ P = 4

จากรูปที่ 4.6 แสดงการเปรียบเทียบประสิทธิภาพทั้ง 4 วิธีเมื่อจำนวนหน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล และเมตริกซ์ 256×256, 512×512, 1024×1024 และ 2048×2048 ตามลำดับ จะเห็นว่าวิธี cbmtm_w มีประสิทธิภาพดีที่สุด วิธีที่มีประสิทธิภาพรองลงมาคือวิธี rbmtm_w และวิธีไทลิงก์ ส่วนวิธีที่มีประสิทธิภาพน้อยที่สุดคือวิธี rbmtm_w/o

บทที่ 5

สรุปผลการทดลอง และแนวทางการพัฒนาการวิจัย

5.1 สรุปผลและวิเคราะห์ผลการทดลอง

การพาร์ทิชันเพื่อหาผลคูณของเมตริกซ์ทรานสโพสแบบขนานที่ได้เสนอไว้ในบทที่ 3 โดยก่อนที่จะมีการคูณเมตริกซ์นั้นจะมีการทรานสโพสเมตริกซ์เฉพาะเมตริกซ์ B โดยนำวิธีไทลิงก์ (Tiling Technique) มาช่วยในการทรานสโพสเมตริกซ์ ทั้งนี้การทรานสโพสเมตริกซ์ B นั้นจะทำให้เป็นการง่ายต่อการเข้าถึงข้อมูลตามมาตรฐานเอ็มพีไอ (MPI Standard) วิธีการคูณเมตริกซ์มีทั้งหมด 3 วิธี คือ วิธีที่ 1 การหาผลคูณเมตริกซ์ ทรานสโพสแบบขนานด้วยวิธี rbmtm_w (row-block partitioning matrix-transpose multiplication with replicated data) วิธีที่ 2 การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี cbmtm_w (checkerboard-block partitioning matrix-transpose multiplication with replicated data) และวิธีที่สุดท้าย คือ การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w/o (row-block partitioning matrix-transpose multiplication without replicated data) โดยแต่ละวิธีที่กล่าวมา มีวัตถุประสงค์ และความยากง่ายในขั้นตอนวิธีที่ต่างกันดังนี้ คือ วิธีแรก (วิธี rbmtm_w) เป็นวิธีที่มีประสิทธิภาพทางด้านการสื่อสารมากที่สุด แต่ใช้เนื้อที่ในการเก็บข้อมูลที่ซ้ำซ้อนในแต่ละหน่วยประมวลผลมากที่สุดเมื่อเทียบกับ 3 วิธี ส่วนวิธีที่สอง (วิธี cbmtm_w) เป็นวิธีที่ใช้เนื้อที่ในการเก็บข้อมูลซ้ำซ้อนกันในแต่ละหน่วยประมวลผลน้อยกว่าวิธีแรก และมีการสื่อสารระหว่างหน่วยประมวลผลบ้างเล็กน้อย ส่วนวิธีสุดท้ายเป็นวิธีที่ไม่ใช้เนื้อที่ในการเก็บข้อมูลซ้ำซ้อนกันเลยในแต่ละหน่วยประมวลผล แต่มีประสิทธิภาพทางด้านการสื่อสารน้อยที่สุด เพราะจะมีการสื่อสารกันระหว่างหน่วยประมวลผลมากกว่าวิธีอื่นๆ

จากผลการทดลองที่ผ่านมาในบทที่ 4 จะเห็นว่าการหาผลคูณแบบขนานแต่ละวิธีมีประสิทธิภาพที่น่าพอใจมาก โดยวิธีที่มีประสิทธิภาพที่สุดคือวิธี cbmtm_w รองลงมาคือวิธี rbmtm_w ส่วนวิธีแบบขนานที่แย่ที่สุดในการทดลองครั้งนี้คือวิธี rbmtm_w/o เนื่องจากว่าวิธีนี้ต้องใช้เวลาในการติดต่อสื่อสารเพื่อแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลมากว่าวิธีอื่นๆ

5.2 แนวทางการพัฒนาการวิจัย

จากการดำเนินการวิจัยพบว่ามีข้อจำกัดของการวิจัยดังนี้

- ขนาดของเมตริกซ์ที่ใช้ในการวิจัยจะต้องเป็นเมตริกซ์จัตุรัสที่มีขนาด $n \times n$ เท่านั้น
- ขนาดของเมตริกซ์ย่อย (Sub-Matrix) ที่ใช้ในการวิจัยจะต้องเป็นเมตริกซ์จัตุรัสที่มีขนาด $t \times t$ เท่านั้น และจะมีจำนวนเมตริกซ์ย่อยทั้งหมด $\left(\frac{n}{t}\right)^2$ เมตริกซ์
- การออกแบบโปรแกรมที่งานวิจัยนี้นำเสนอ เหมาะสมกับหน่วยประมวลที่มีจำนวน 4 หน่วยประมวลผล

จากข้อจำกัดในงานวิจัยที่กล่าวมาข้างต้นจึงมีข้อเสนอแนะแนวทางสำหรับพัฒนางานวิจัยดังต่อไปนี้

- พัฒนาโปรแกรมที่สามารถทำให้ขนาดของเมตริกซ์หลัก และเมตริกซ์ย่อยมีความยืดหยุ่นมากขึ้น เช่น สามารถประมวลผลในเมตริกซ์หลักและเมตริกซ์ย่อยที่เป็นเมตริกซ์สี่เหลี่ยมผืนผ้า ($t \times k$) ได้
- ปรับปรุงการออกแบบโปรแกรมให้เข้ากับโปรแกรมที่ใช้หน่วยประมวลผลหลายหน่วยประมวลผล (มากกว่า 4 หน่วยประมวลผล) เพื่อที่จะได้ประสิทธิภาพที่ดียิ่งขึ้น
- นำไปพัฒนาโปรแกรมให้สามารถใช้งานได้ง่ายขึ้น และใช้งานได้หลายระบบปฏิบัติการ เช่น พัฒนาบนระบบปฏิบัติการไมโครซอฟท์วินโดวส์ เป็นต้น และสามารถนำโปรแกรมที่พัฒนาขึ้นไปใช้งานกับโปรแกรมประยุกต์ต่างๆ เช่น การจำลองผลภาพสแกน 3 มิติ ทางการแพทย์ หรือ การจำลองแบบทางชีววิทยา เป็นต้น

เอกสารอ้างอิง

[1] M. Kim, Y. J. Jang and W. W. Ro. “Parallel Transpose of Matrix Multiplication Based on the Tiling Algorithms,” IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS), 2011, pp.1-3.

[2] P. Chen, K. Dai, D. Wu, J. Rao and X. Zou. “The Parallel Algorithm Implementation of Matrix Multiplication Based on ESCA,” IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2010, pp.1091-1094.

[3] S. Hunoid, T. Rauber and G. Runger. “Combining buliding blocks for parallel multi-level matrix multiplication,” Parallel Computing, Volume 34, Issues 6-8, July 2008, pp.411-426.

[4] M. Krishnan and J. Nieplocha. “Optimizing Parallel Multiplication Operation for Rectangular and Transposed Matrices,” 10th International Conference on Parallel and Distributed Systems (ICPADS), IEEE Computer Society ,7-9 July 2004, USA, pp.257-266.

[5] M. Krishnan and J. Nieplocha. “SRUMMA: A Matrix Multiplication Algorithm Suitable for Clusters and Scalable Shared Memory Systems,” 18th International Conference on Parallel and Distributed Processing Sysmpsium (IPDPS), IEEE Computer Society ,26-30 April 2004, USA, pp.70-80.

[6] H. J. Lee and P. Robertson. “Generalized Cannon’s algorithm for parallel matrix multiplication,” Proceedings of the 11th international conference on Supercomputing (ICS’97),07-11 July 97, Vienna, Austria, pp.44-51.

[7] C. A. Alonso Sanches and S. W. Song. “SIMD Alorithms for Matrix Multiplication on the Hyper cube,” The 8th int’l Proceedings on Parallel Processing Symposium. 1994. pp.490-496.

[8] J. F. Tasic, M. Zajc, and A. Kosir. “Comparison of Some Parallel Matrix Multiplication Algorithms,” 8th Mediterranean Electrotechnical Conference MELECON’96. V.1, 1996. pp.155-158.

[9] J. Gunnels, C. Lin, G. Morrow, and R. Geijn. “A Flexible Class of parallel Matrix Multiplication Algorithms,” [Online], Available :http://www.cs.utexas.edu/users/plapack/papers/ipps98/ipps98.html. 1998.

[10] V. Kumar, A. Grama, A. Gupta, and G. Karypis. Introduction to parallel Computing, Canada : The Benjamin/Cummings Pubishing Company, Inc. 1994.

[11] M. J. Quinn. Parallel Programming in C with MPI and OpenMP. United States: The McGraw – Hill Companies, Inc. 2003.

[12] ไพโรจน์ สมุทรักษ์. การคูณเมตริซ์แบบขนานบนระบบพีซีคลัสเตอร์ วิทยานิพนธ์วิทยาศาสตรมหา บัณฑิต สาขาวิทยาการคอมพิวเตอร์. สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2548.

[13] ธีรณี อจลากุล และ ราชวิชช์ สโรชวิกสิต. เทคโนโลยีการประมวลผลแบบขนานและแบบกระจาย. กรุงเทพมหานคร: Top Publishing Co.,Ltd. 2551.

[14] จีรพร วีระพันธุ์. ขั้นตอนวิธีแบบขนาน (เอกสารประกอบการสอน) สาขาวิทยาการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2554.

ภาคผนวก ก
งานวิจัยที่ตีพิมพ์

(Thai Track)

The 10th International Joint Conference on Computer Science and Software Engineering (JCSSE2013)

The next decade of Computer Science & Software Engineering

May 29th – 31st, 2013

Department of Computer Science, Mahasarakham University

Maha Sarakham, THAILAND

การพาร์ทิชันข้อมูลสำหรับการคูณเมตริกซ์ทรานสโพสแบบขนาน
Data Partitioning for Parallel Transpose of Matrix Multiplication

นันทิพัฒน์ พิศุทธางกูร[1] และ จีรพร วีระพันธุ์[2]
ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ถนนฉลองกรุง เขตลาดกระบัง กรุงเทพมหานคร 10520
E-mail: n.pisuttangkoon@gmail.com[1] และ ksjeerap@kmitl.ac.th[2]

บทคัดย่อ

งานวิจัยฉบับนี้นำเสนอการคูณเมตริกซ์ทรานสโพสแบบขนาน $C = A \times B$ ที่นำวิธีการไทลิงก์มาประยุกต์ในขั้นตอนการทรานสโพสเมตริกซ์ โดยในการทรานสโพสจะทำเฉพาะข้อมูลเข้าเมตริกซ์ที่สอง (เมตริกซ์ B) ซึ่งวิธีการนี้จะช่วยลดจำนวนในการย้ายข้อมูลของแต่ละหน่วยประมวลผล และการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล การทำให้เมตริกซ์ B อยู่ในรูปการทรานสโพสจะทำให้ง่ายต่อการเข้าถึงข้อมูลตามมาตรฐานภาษาเอ็มพีไอ ซึ่งเป็นมาตรฐานในการเขียนโปรแกรมแบบขนานที่งานวิจัยฉบับนี้นำมาใช้ การหาผลคูณเมตริกซ์แบบขนานที่นำเสนอมี 3 แบบ แตกต่างกันตามลักษณะของการจัดแบ่งข้อมูล และการเก็บข้อมูลที่ซ้ำซ้อนกันในแต่ละหน่วยประมวลผล ในส่วนของการวัดประสิทธิภาพจะนำเวลาในการหาผลคูณของทั้ง 3 วิธีมาเปรียบเทียบกันกับการทรานสโพสแบบดั้งเดิม และการคูณเมตริกซ์ทรานสโพสโดยใช้วิธีการไทลิงก์

คำสำคัญ: การคูณเมตริกซ์, วิธีการไทลิงก์, การจัดแบ่งเมตริกซ์

Abstract

This paper presents parallel matrix multiplication $C = A \times B$ by using matrix transpose with tiling technique. This approach suggests the tiling-based transpose for the second matrix (B) only. Our proposed method can decrease a number of data moving in each processor. Before processing parallel multiplication, matrix B is changed to transpose for simple and fast access by MPI standard, a standard of parallel programming design used in this paper. Then, three methods of parallel matrix multiplication are introduced, which are different in data partitioning and data replicating in each processor. Finally in performance evaluation, the response time (results) of our three methods are compared to those of the matrix multiplication with the original transpose and with the existing tiling-transpose technique.

Keywords: parallel matrix multiplication, tiling technique, matrix partitioning

1. บทนำ

การหาผลคูณของเมตริกซ์เป็นการประมวลผลขั้นพื้นฐานในการหาคำตอบทางวิทยาศาสตร์ และวิศวกรรมศาสตร์หลายด้าน เช่น การหาระยะทางที่สั้นที่สุดโดยใช้เมตริกซ์ การวิเคราะห์ออกแบบโครงสร้างเหล็ก และงานทางด้านกราฟิก เป็นต้น โดยปกติการหาผลคูณเมตริกซ์ แต่ละครั้งต้องใช้ระยะเวลาในการคำนวณเป็นเวลานาน โดยเฉพาะ ถ้าเมตริกซ์มีขนาดใหญ่ ที่เป็นเช่นนี้เนื่องจากการคูณเมตริกซ์นั้นมีความซับซ้อนด้านเวลา (Time Complexity) เท่ากับ $O(n^3)$ ดังนั้นในงานวิจัยฉบับนี้ได้นำเอาวิธีการแบบขนานมาใช้ในการหาคำตอบของผลคูณของเมตริกซ์ โดยมาตรฐานของวิธีการแบบขนานที่งานวิจัยฉบับนี้นำมาใช้คือ มาตรฐานภาษาเอ็มพีไอ (MPI Standard) ซึ่งเป็นมาตรฐานที่มีการใช้งานอย่างแพร่หลายอยู่ในขณะนี้

การหาผลคูณเมตริกซ์แบบขนานมีนักวิจัยนำเสนอไว้หลายวิธี [1-7] โดยในปี 2005, Samutrak [6] ได้เสนอวิธีการคูณเมตริกซ์แบบขนานบนระบบคลัสเตอร์ (Cluster System) ส่วนการคูณเมตริกซ์แบบทรานสโพสด้วยวิธีการไทลิงก์ (Tiling Technique) ถูกเสนอโดย Kim และคณะ [1] ในปี 2011 ซึ่งได้ผลลัพธ์จากการทดลองบนระบบมัลติคอร์ (Multicore) ที่เร็วกว่าวิธีการคูณเมตริกซ์ที่ทำการทรานสโพสเมตริกซ์ทั้งเมตริกซ์ 4.76% และ 6.61% เมื่อดำเนินการบนหน่วยประมวลผลที่แตกต่างกัน 2 ชนิด

เนื่องจากในการคูณเมตริกซ์ $C = A \times B$ การทำการทรานสโพสทั้งเมตริกซ์ B จะมีจำนวนครั้งในการย้ายข้อมูลมากกว่าวิธีการทรานสโพสแบบไทลิงก์ แต่การทรานสโพสทั้งเมตริกซ์จะช่วยลดเวลาในการคูณเมตริกซ์แบบขนานเพราะเข้าถึงข้อมูลที่จัดเก็บแบบต่อเนื่องกัน ในขณะที่การทรานสโพสแบบไทลิงก์แบบเดิมนั้นเป็นการทรานสโพสเฉพาะในไทล์เล็กๆ ทำให้ข้อมูลในเมตริกซ์ B ยังเป็นการทรานสโพสที่ยังไม่สมบูรณ์ และเมื่อมีการคูณเมตริกซ์ยังคงมีการย้ายข้อมูลบางส่วนอยู่

งานวิจัยฉบับนี้ได้มีการปรับปรุงข้อมูลนำเข้า โดยประยุกต์ใช้วิธีการไทลิงก์ (Tiling Technique) กับการทรานสโพสเมตริกซ์ วิธีการใหม่นี้ทำให้มีจำนวนการทรานสโพสน้อยกว่าวิธีการดั้งเดิม และไม่ต้องทำการย้ายข้อมูลในระหว่างดำเนินการคูณเมตริกซ์ การหาผลคูณเมตริกซ์แบบขนานที่นำเสนอมี 3 แบบ ซึ่งแตกต่างกันตามลักษณะการจัดแบ่งข้อมูลเพื่อการประมวลผลแบบขนาน และการติดต่อสื่อสารที่เหมาะสม รวมถึง

นโยบายการเก็บข้อมูลที่ซ้ำซ้อนกันในแต่ละหน่วยประมวลผล โดย 3 แบบดังกล่าวมีดังนี้

1) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w (row-block partitioning matrix-transpose multiplication with replicated data)
2) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี cbmtm_w (checkerboard-block partitioning matrix-transpose multiplication with replicated data)
3) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w/o (row-block partitioning matrix-transpose multiplication without replicated data)

วิธีแรก เป็นวิธีที่มีประสิทธิภาพในการติดต่อสื่อสารมากที่สุด แต่ใช้เนื้อที่ในการเก็บข้อมูลซ้ำซ้อนมากที่สุด วิธีที่สองเป็นวิธีที่มีประสิทธิภาพในการติดต่อสื่อสารพอๆกับวิธีแรก แต่มีการใช้เนื้อที่ในการเก็บข้อมูลที่ซ้ำซ้อนกันน้อยกว่า ส่วนวิธีสุดท้ายไม่มีการเก็บข้อมูลที่ซ้ำซ้อนกันเลย แต่มีเวลาในการติดต่อสื่อสารระหว่างหน่วยประมวลผลมากขึ้น การวัดประสิทธิภาพของวิธีการคูณเมตริกซ์แบบขนานที่เสนอ คำนวณโดยนำค่าเวลาที่ใช้ในการประมวลผล (Response Time) ของทุกวิธีมาเปรียบเทียบกัน

ส่วนที่เหลือของบทความวิจัยนี้มีดังนี้ ส่วนที่ 2 คือทฤษฎีและงานวิจัยที่เกี่ยวข้อง ส่วนที่ 3 คือ วิธีการคูณเมตริกซ์แบบขนานที่นำเสนอ ส่วนที่ 4 คือ การเปรียบเทียบประสิทธิภาพของการคูณเมตริกซ์ทรานสโพสแบบขนาน และการสรุปผลงานวิจัยในส่วนที่ 5

2. งานวิจัยที่เกี่ยวข้อง

ในปี 2004 Krishnan และ Nieplocha [4] ได้พัฒนาวิธีการหาผลคูณเมตริกซ์แบบขนานจากวิธีการที่เรียกว่า “SRUMMA” [5] โดยปรับปรุงประสิทธิภาพการคูณเมตริกซ์ทรานสโพส และเมตริกซ์สี่เหลี่ยมมุมฉาก โดยดำเนินการบน “ระบบคลัสเตอร์” (Cluster System) วิธี SRUMMA โดยทั่วไปนั้นจะมีการจัดการกับหน่วยความจำ และการส่งข้อมูลระหว่างกันอย่างมีประสิทธิภาพ อีกทั้งยังมีความยืดหยุ่นทั้งบนระบบคลัสเตอร์ และระบบหน่วยความจำร่วม (Shared Memory) สำหรับวิธีการหาผลคูณของเมตริกซ์สี่เหลี่ยมมุมฉากนั้น จะแบ่งได้เป็น 3 กรณีที่ต่างกันตามขนาดของเมตริกซ์ คือ กรณีแรกคือ m มีขนาดเล็ก, n และ k มีขนาดใหญ่ กรณีที่สอง คือ k มีขนาดเล็ก, m และ n มีขนาดใหญ่ กรณีสุดท้ายคือ n มีขนาดเล็ก, m และ k มีขนาดใหญ่โดย m, n, k คือขนาดของเมตริกซ์ $A_{mxk} \times B_{kxn}$ ส่วนวิธีการหาผลคูณของเมตริกซ์ ทรานสโพส มี 3 กรณีที่แตกต่างกันดังนี้ กรณีแรก $C = A^T B$ กรณีที่สอง $C = AB^T$ และกรณีสุดท้าย $C = A^T B^T$ ตามลำดับ

สำหรับผลการทดลองนั้นจะนำเอาวิธี SRUMMA มาเปรียบเทียบกับ ชุดการคำนวณทางพีชคณิตเชิงเส้นที่เรียกว่า ScaLAPACK บนระบบคลัสเตอร์ และระบบหน่วยความจำร่วม ในส่วนของการหาผลคูณเมตริกซ์สี่เหลี่ยมมุมฉากนั้น มีขนาดของเมตริกซ์ที่ใหญ่ที่สุดคือ 4,000 และเล็กที่สุดคือ 1,000 จากผลการทดลองแสดงให้เห็นว่า SRUMMA ดำเนินการได้ดีกว่า ScaLAPACK ในทุกกรณี ในส่วนของการหาผลคูณเมตริกซ์ทรานสโพสนั้น วิธี SRUMMA มีประสิทธิภาพสูงกว่า ScaLAPACK อย่างต่อเนื่องบนทั้งสองระบบ คือ ระบบคลัสเตอร์ และระบบหน่วยความจำร่วม

ในปี 2010 Chen และคณะ [2] ได้ออกแบบสถาปัตยกรรมแบบขนานที่ชื่อว่า ESCA (Engineering and science computing accelerator) โดยพัฒนาต่อมาจากสถาปัตยกรรม SIMD (Single-instruction multiple-data-stream) ซึ่งมีจุดมุ่งหมายเพื่อเร่งการคำนวณหาผลลัพธ์ของการคูณเมตริกซ์ โดยมีหน่วยควบคุม (CU: Control Unit) เพียงหน่วยเดียว แต่จะมีหน่วยประมวลผลย่อย (PE: Processing Element) หลายๆหน่วยทำงานร่วมกัน ในส่วนข้างต้นเป็นส่วนที่เกี่ยวกับฮาร์ดแวร์ (Hardware) สำหรับวิธีการคูณเมตริกซ์นั้นจะใช้วิธีการที่เรียกว่า “แคนนอน” (Cannon’s Algorithm) [7] ซึ่งเป็นขั้นตอนวิธีการคูณเมตริกซ์ที่มีมิติ $n \times n$ ด้วยการทำงานแบบขนาน

ทางด้านเวลาการทำงาน ถ้ากระบวนการทุกกระบวนการทำงานพร้อมกันแบบขนาน และกระบวนการแต่ละกระบวนการสามารถส่งเมตริกซ์ย่อยไปยังกระบวนการข้างเคียงได้เพียงหนึ่งกระบวนการต่อครั้งจะเกิดการส่งเมตริกซ์ย่อยทั้งหมด $4\sqrt{p} - 2$ ครั้ง และการคูณเมตริกซ์ย่อยแต่ละครั้งใช้การคูณตัวเลขทั้งหมด $\left(\frac{n}{\sqrt{p}}\right)^3$ และจะได้เวลาการทำงานคือ $O\left(\frac{n}{\sqrt{p}}\right)^3$

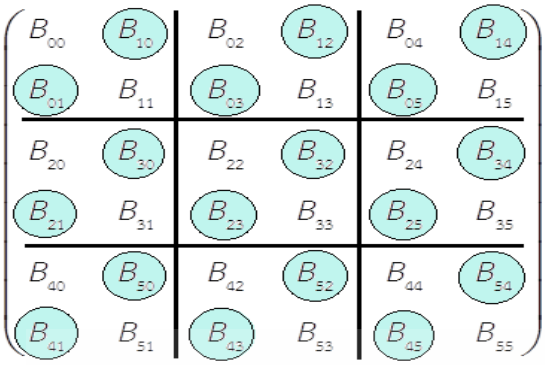
อย่างไรก็ตาม วิธีนี้จะใช้ได้ก็ต่อเมื่อจำนวนของกระบวนการเป็นตัวเลขที่เกิดจากการยกกำลังสอง หรือเมตริกซ์ที่นำมาคูณกันเป็นเมตริกซ์จัตุรัสเท่านั้น

ในปี 2011 Kim และคณะ [1] ได้เสนอวิธีการคูณเมตริกซ์แบบทรานสโพส โดยใช้วิธีการไทลิงก์ (Tiling Technique) จากการดำเนินการคูณเมตริกซ์ปกติ $C = A \times B$ และขนาดของเมตริกซ์ A และ B มีขนาด $n \times n$ ผลลัพธ์ของเมตริกซ์จะถูกเก็บไว้ในเมตริกซ์ C โดยดูได้จากสมการที่ (1)

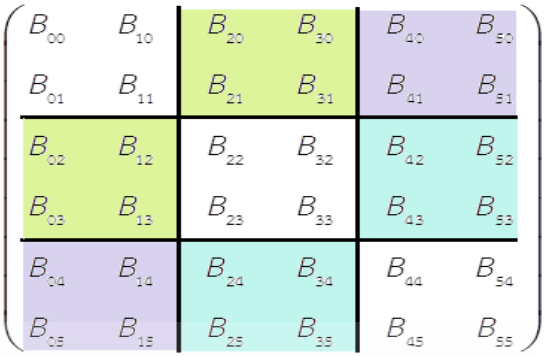
$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j} \quad (1)$$

และได้ออกแบบวิธีการคูณเมตริกซ์แบบขนานไว้ตามสมการที่ (2) ดังนี้

$$\text{Thread } i : C_i = A_i \times B_{nxn}$$
$$= [\, A_i \times B_1, A_i \times B_2, \ldots, A_i \times B_n \,] \quad (2)$$

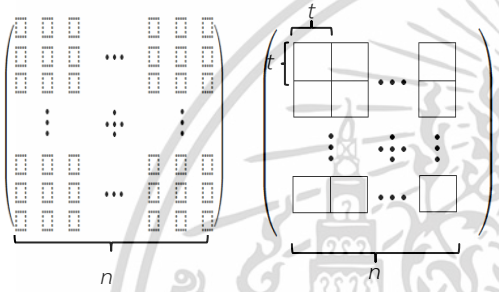


ก. การทรานสโพสในแต่ละเมตริกซ์ย่อย



ข. การทรานสโพสระหว่างเมตริกซ์ย่อย

รูปที่ 1 วิธีการทรานสโพสในข้อมูลนำเข้าเมตริกซ์ *B*



ก. เมตริกซ์ก่อนการแบ่งย่อย ข. เมตริกซ์หลังแบ่งย่อย

รูปที่ 2 เมตริกซ์ก่อน และหลังการแบ่งย่อย

เมื่อ A_i และ C_i คือ แถวที่ i ใดๆ ที่ถือครองด้วยเธรด (Thread) ที่ i^{th} ของเมตริกซ์ A และ C ตามลำดับ และ B_j คือ หลักที่ j ของเมตริกซ์ B ไทลิงก์ คือกระบวนการแบ่งเมตริกซ์ที่มีขนาดใหญ่ให้เป็นเมตริกซ์ย่อย (Sub-Matrix) โดยที่ เมตริกซ์ที่ย่อยมาได้นั้นจะถูกเรียกว่า “ไทล์” (Tile) และในแต่ละไทล์ ที่ได้แบ่งย่อยมาจากเมตริกซ์ใหญ่นั้น จะมีการดำเนินการทรานสโพสภายในแต่ละไทล์ ส่วนประสิทธิภาพที่วัดได้จากการสร้างแบบจำลองนั้น ได้นำมาเปรียบเทียบกับวิธีการคูณเมตริกซ์ที่ทำการทรานสโพสทั้งเมตริกซ์ ซึ่งจากผลการทดลองวิธีการคูณเมตริกซ์ทรานสโพสด้วยวิธีการแบบไทลิงก์นั้นเร็วกว่าวิธีการคูณเมตริกซ์ที่ทำการทรานสโพสทั้งเมตริกซ์ 4.76% และ 6.61% เมื่อดำเนินการบนหน่วยประมวลผล Core2 9400 และ Phenom 9550 ตามลำดับ

3. การพาร์ทิชันข้อมูลสำหรับการคูณเมตริกซ์ทรานสโพสแบบขนาน

ในงานวิจัยนี้ได้เสนอการปรับปรุงวิธีการคูณเมตริกซ์ทรานสโพสแบบขนานจากวิธีการที่มีอยู่เดิม ซึ่งก็คือการทรานสโพสเมตริกซ์แบบขนานด้วยวิธีการไทลิงก์ [1] โดยจะแบ่งวิธีดำเนินการทดลองออกเป็นสองส่วน ส่วนแรกคือ การทำการทรานสโพสเมตริกซ์แบบขนาน และส่วนที่สอง คือ การหาผลคูณเมตริกซ์ทรานสโพสแบบขนาน ดังนี้

ส่วนที่ 1) เป็นวิธีการทรานสโพสเมตริกซ์แบบขนาน โดยจะทำการแบ่งเมตริกซ์ที่มีขนาด $n \times n$ เป็นเมตริกซ์ย่อย (Sub-Matrix) ขนาด $t \times t$ จำนวน $\left(\frac{n}{t}\right)^2$ เมตริกซ์ หลังจากทำการแบ่งเป็นเมตริกซ์ย่อยดังรูปที่ 2(ข.) แล้วจะทำการทรานสโพส ในเมตริกซ์ย่อยแต่ละเมตริกซ์ (รูปที่1(ก.)) หลังจากนั้นทำการทรานสโพส อีกครั้งแต่เป็นการทรานสโพสระหว่างเมตริกซ์ย่อยทั้งหมด โดยใช้การส่งข้อมูลสลับระหว่างหน่วยประมวลผลเพื่อง่ายต่อการเข้าถึงข้อมูลตามมาตรฐานภาษาเอ็มพีไอ เพราะว่าหลังจากการทรานสโพสอีกครั้งนั้น เมตริกซ์จะเป็นเมตริกซ์ที่ทรานสโพสโดยสมบูรณ์

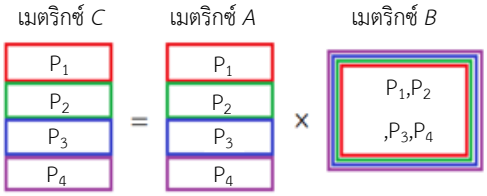
จากรูปที่ 1 เป็นวิธีการทรานสโพสข้อมูลนำเข้าในเมตริกซ์ B โดยรูปที่ 1(ก.) จะเป็นการทรานสโพสครั้งแรกในเมตริกซ์ย่อย และรูปที่ 1(ข.) เป็นการทรานสโพสระหว่างเมตริกซ์ย่อย

สมการต่อไปนี้ คือ จำนวนครั้งของการทรานสโพสแบบดั้งเดิม แทนโดย TF (3) จำนวนครั้งของวิธีการทรานสโพสแบบขนานโดยการใช้วิธีไทลิงก์ แทนโดย PT (4) และจำนวนครั้งของการทรานสโพสด้วยวิธีการใหม่ในงานวิจัยนี้แทนโดย NPT (5) ตามลำดับ

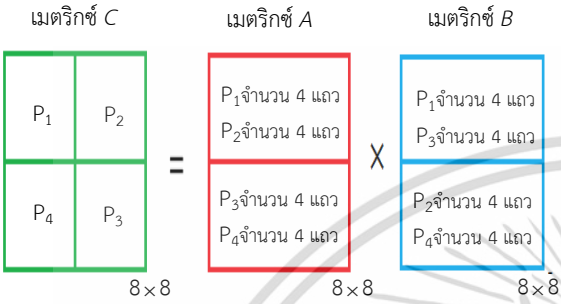
$$TF = 2\left(\sum_{i=1}^{(n-1)} i\right) \tag{3}$$

$$PT = 2\left(\frac{n^2}{t^2}\right)\cdot\left(\sum_{i=1}^{(t-1)} i\right) \tag{4}$$

$$NPT = \left[2\left(\frac{n^2}{t^2}\right)\cdot\left(\sum_{i=1}^{(t-1)} i\right)\right] + 2\left(\sum_{i=1}^{(\frac{n}{t}-1)} i\right) \tag{5}$$



รูปที่ 3 การแบ่งงานเพื่อหาผลคูณเมตริกซ์ *C* ในกรณีที่มี 4 หน่วยประมวลผล ของวิธี rbmtm_w



รูปที่ 4 การแบ่งงานเพื่อหาผลคูณเมตริกซ์ *C* ในกรณีที่มี 4 หน่วยประมวลผล ของวิธี cbmtm_w

จำนวนครั้งของการทรานสโพสในข้อมูลนำเข้าเมตริกซ์ *B* คือ $n^2 - n$ เมื่อใช้วิธีทรานสโพสแบบดั้งเดิม และ $n^2 - \frac{n^2}{t}$ เมื่อใช้วิธีทรานสโพสแบบขนานด้วยวิธีไทลิงก์ ส่วนจำนวนครั้งของการทรานสโพสเมื่อใช้วิธีใหม่ในงานวิจัยนี้จะสมการที่ได้เป็นไปตามสมการที่ (6)

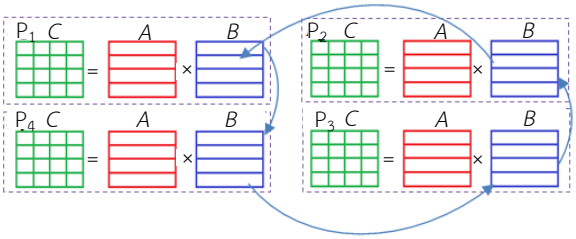
$$\left(n^2 - \frac{n^2}{t}\right) + \left(\left(\frac{n^2}{t^2}\right) - \frac{n}{t}\right) \quad (6)$$

สำหรับตัวอย่างเมื่อกำหนดให้ $n = 6$ และ $t = 2$ จำนวนของการ ทรานสโพสในแต่ละวิธีมีดังนี้ วิธีทรานสโพสแบบดั้งเดิมโดยแทน $n = 6$ และ $t = 2$ ในสมการที่ (3) คือ 30 ครั้ง วิธีทรานสโพสแบบขนานด้วยวิธีไทลิงก์ โดยแทน $n = 6$ และ $t = 2$ ในสมการที่ (4) คือ 18 ครั้ง แต่วิธีการนี้ ยังคงมีการย้ายข้อมูลบางส่วนในขั้นตอนการคูณเมตริกซ์ และวิธีการใหม่ในงานวิจัยนี้ โดยแทน $n = 6$ และ $t = 2$ ในสมการที่ (5) คือ 24 ครั้ง โดยที่ไม่ต้องมีการย้ายข้อมูลในขั้นตอนการคูณเมตริกซ์

ส่วนที่ 2) การคูณเมตริกซ์ทรานสโพสแบบขนาน จะมีวิธีการหาผลคูณ 3 วิธีที่แตกต่างกันตามลักษณะการจัดแบ่งข้อมูล และการใช้ข้อมูลที่ซ้ำซ้อนกันในแต่ละหน่วยประมวลผล

งานวิจัยนี้จะใช้หน่วยประมวลผลย่อย 4 หน่วยประมวลผล โดยมีรายละเอียดแต่ละวิธีดังนี้

1) การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี rbmtm_w (row-block partitioning matrix-transpose multiplication with replicated data) จะประกอบด้วยหน่วยประมวลที่แบ่งเป็น 2 ชนิดคือ หน่วยประมวลผลหลัก (Master Processor) จำนวน 1 หน่วยประมวลผล และหน่วยประมวลผลทำงาน (Worker Processor) จำนวน 4 หน่วย



รูปที่ 5 การแลกเปลี่ยนข้อมูลของเมตริกซ์ B รอบแรกของวิธี cbmtm_w ในกรณีที่มี 4 หน่วยประมวลผล

หน่วยประมวลผลหลัก มีหน้าที่แบ่งข้อมูลตามแถวของเมตริกซ์ *A* ขนาด $k = \frac{n}{4}$ เพื่อให้ภาระงาน (Work Load) มีขนาดเท่ากัน ดังรูปที่ 3 และทำการรับผลคูณเมตริกซ์ *C* จากหน่วยประมวลผลทำงาน 4 หน่วย

หน่วยประมวลผลทำงาน มีหน้าที่อ่านข้อมูลบางส่วนของเมตริกซ์ *A* ขนาด *k* และเมตริกซ์ *B* ทั้งหมด และทำการคำนวณผลคูณเมตริกซ์ *C* สุดท้ายจะเป็นการส่งผลลัพธ์ของเมตริกซ์ *C* กลับไปยังหน่วยประมวลผลหลัก

2) การหาผลคูณเมตริกซ์แบบขนานด้วยวิธี cbmtm_w (checkerboard block partitioning matrix-transpose multiplication with replicated data) วิธีนี้จะประกอบไปด้วยหน่วยประมวลผล 2 ชนิด เหมือนแบบแรก โดยหน่วยประมวลผลหลักมีหน้าที่คือ ทำการแบ่งข้อมูลตามแถวของเมตริกซ์ *A* และ *B* ให้มีขนาด $r = \frac{n}{\sqrt{p}}$ เมื่อ $p = 4$ จะได้ $r = \frac{n}{2}$ เพื่อให้ภาระงานมีขนาดเท่าๆกัน และทำการรอรับผลคูณเมตริกซ์ *C* จากหน่วยประมวลผลทำงาน 4 หน่วยประมวลผล สำหรับหน่วยประมวลผลทำงานนั้นมีหน้าที่รอรับการติดต่อจากหน่วยประมวลผลหลัก และอ่านข้อมูลบางส่วนของเมตริกซ์ *A* และ *B* ขนาด $r = \frac{n}{2}$ จากนั้นทำการคำนวณผลคูณเมตริกซ์ *C* และส่งกลับไปยังหน่วยประมวลผลหลัก

สมมติให้เมตริกซ์ *A* และ *B* มีขนาด 8×8 และใช้หน่วยประมวลผล 4 หน่วยประมวลผล จะแบ่งข้อมูลตามแถวของเมตริกซ์ *A* และ *B* คือ $r = \frac{n}{\sqrt{p}} = \frac{8}{\sqrt{4}} = 4$ ดังรูปที่ 4

3) การหาผลคูณเมตริกซ์ทรานสโพสแบบขนานด้วยวิธี rbmtm_w/o (row-block partitioning matrix-transpose multiplication without replicated data) จะประกอบด้วยหน่วยประมวลผล 2 แบบ เหมือนทั้งสองแบบที่ผ่านมา โดยวิธีนี้จะไม่มีการเก็บข้อมูลที่ซ้ำซ้อนกัน แต่เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลมีค่าเพิ่มมากขึ้น หน่วยประมวลผลหลักจะมีหน้าที่ ทำการแบ่งแถวของเมตริกซ์ *A* และ *B* ขนาด $r = \frac{n}{p}$ แถว เพื่อให้ภาระงาน (Work Load) มีขนาดเท่าๆกัน และรอรับข้อมูลจากเมตริกซ์ *C* จากหน่วยประมวลผลทำงานทั้ง 4 หน่วย สำหรับหน้าที่ของหน่วยประมวลผลทำงานจำนวน 4 หน่วยนั้น มีหน้าที่รอรับการติดต่อจากหน่วยประมวลผลหลัก หลังจากนั้นอ่านข้อมูลบางส่วนของเมตริกซ์ *A* และ *B* ตามขนาด *r* แถว (แถวที่ $(i-1)k+1$ ถึงแถวที่ *ik* เมื่อ *i* คือไอดีของหน่วยประมวลผลทำงาน (Worker ID) มีค่า 1-4) ที่หน่วย

ประมวลผลหลักได้แบ่งไว้ก่อนหน้านี้ และทำการคำนวณผลคูณบางส่วนของเมตริกซ์ C หลังจากนั้นทำการส่งค่าเมตริกซ์ B แถวที่ $(i\text{-}1)k+1$ ถึงแถวที่ ik ให้หน่วยประมวลผลทำงานลำดับก่อนหน้า (P_{i-1}) และรับข้อมูลของเมตริกซ์ B แถวที่ $ir+1$ ถึงแถวที่ $ir+r$ จากหน่วยประมวลผลทำงานลำดับถัดไป (P_i) ดังรูปที่ 5 หลังจากนั้นทำซ้ำในขั้นตอนการถ่ายโอนหน่วยประมวลผลและคำนวณผลคูณเมตริกซ์ C ไปเรื่อยๆ จนได้เมตริกซ์ C ครบทุกแถว และหน้าที่สุดท้ายของหน่วยประมวลผลทำงาน คือ ส่งผลคูณของเมตริกซ์ C กลับไปยังหน่วยประมวลผลหลัก

4. ผลการวิจัย

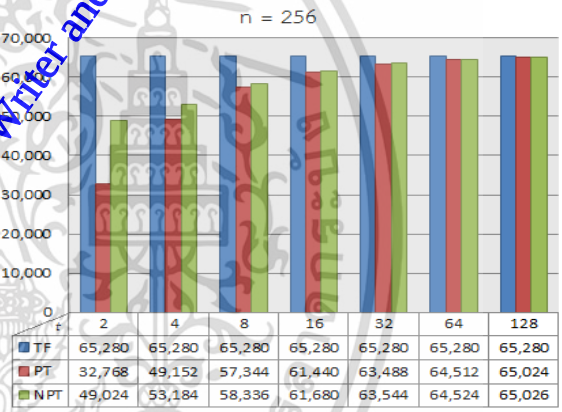
ในหัวข้อนี้จะเป็นการเปรียบเทียบผลการวิจัยในสองส่วน คือ จำนวนครั้งในการทรานสโพสเมตริกซ์ และเวลาที่ใช้ในการหาผลคูณเมตริกซ์ (Response Time) โดยดำเนินการบนเครื่อง CPU Intel core i5

ส่วนแรกเป็นการเปรียบเทียบจำนวนครั้งของวิธีการทรานสโพสทั้ง 3 แบบ คือ การทรานสโพสเมตริกซ์แบบดั้งเดิม (TF), การทรานสโพสเมตริกซ์ด้วยวิธีไทลิงก์ (PT) และการทรานสโพสเมตริกซ์ด้วยวิธีการใหม่ในงานวิจัยนี้ (NPT) จากสมการที่ (3), (4) และ (5) เมื่อพิจารณาจากเมตริกซ์ $(n \times n)$ ที่มีขนาด 256×256, 512×512 และ 1024×1024 โดยมีเมตริกซ์ย่อย $(t \times t)$ มีขนาด t เป็น 2, 4, 8, 16, 32, 64, 128, 256 และ 512 ตามลำดับ จากรูปที่ 6, 7 และ 8 จะเห็นว่าเมื่อ t มีขนาดเล็กๆ และ n มีขนาด 256, 512 และ 1024 การทรานสโพสเมตริกซ์ด้วยวิธีการใหม่ในงานวิจัยนี้กับการทรานสโพสเมตริกซ์ด้วยวิธีไทลิงก์ มีประสิทธิภาพมากกว่าการทรานสโพสเมตริกซ์แบบดั้งเดิมทุกกรณี และสังเกตว่าในช่วงแรกวิธีการทรานสโพสเมตริกซ์ด้วยวิธีไทลิงก์ มีประสิทธิภาพดีที่สุด หลังจากนั้นการทรานสโพสเมตริกซ์ด้วยวิธีไทลิงก์ กับการทรานสโพสเมตริกซ์ด้วยวิธีการใหม่ในงานวิจัยนี้ จะมีประสิทธิภาพไม่ต่างกันมากนัก เมื่อขนาดของ t คือ 4, 8, 16 และเมื่อขนาดของ t เป็น 32 เป็นต้นไป วิธีการทั้งสองนี้จะมีประสิทธิภาพใกล้เคียงกันมากที่สุด แต่ข้อมูลของเมตริกซ์ B ที่ได้จากการทรานสโพสโดยวิธีการใหม่จะเป็นการทรานสโพสที่สมบูรณ์

ส่วนที่สอง เป็นการเปรียบเทียบเวลาที่ใช้ในการประมวลผลคูณเมตริกซ์ โดยรวมเวลาตั้งแต่ขั้นตอนการทรานสโพสเมตริกซ์ และเปรียบเทียบกัน 5 วิธี คือ การคูณเมตริกซ์แบบอนุกรม (Sequential), การคูณเมตริกซ์แบบวิธี rbmtm_w, การคูณเมตริกซ์แบบวิธี cbmtm_w, การคูณเมตริกซ์แบบวิธี rbmtm_w/o และการคูณเมตริกซ์ทรานสโพสโดยวิธีไทลิงก์ ใช้เมตริกซ์ขนาด 256×256, 512×512, 1024×1024 และ 2048×2048 ตามลำดับ และกำหนดจำนวนหน่วยประมวลผลที่ใช้ในการคำนวณเท่ากับ 4 หน่วยประมวลผล ผลการทดลองจากตารางที่ 1 และ รูปที่ 9 จะเห็นว่า เมื่อข้อมูลมีขนาดใหญ่ขึ้น เวลาที่ใช้ในการประมวลผลเมตริกซ์จะเพิ่มขึ้นตามไปด้วย โดยวิธี cbmtm_w ใช้เวลาในการประมวลผลเร็วที่สุดเมื่อเทียบกับทุกวิธี รองลงมา คือ วิธี rbmtm_w, วิธีไทลิงก์, rbmtm_w/o และวิธีแบบอนุกรม ตามลำดับ สังเกตว่าวิธี rbmtm_w/o มีการประมวลผลช้ากว่าวิธีไทลิงก์ เนื่องจากต้องใช้เวลาในการติดต่อสื่อสารเพื่อแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล

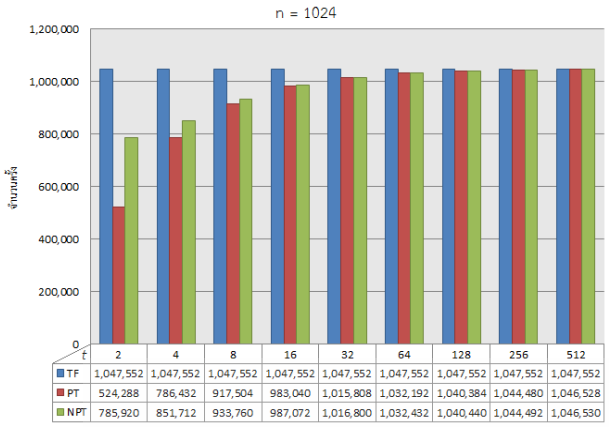
5. สรุปผลงานวิจัย

ในงานวิจัยนี้เสนอการทรานสโพสเมตริกซ์ และคูณเมตริกซ์ทรานสโพส โดยนำวิธีไทลิงก์ (Tiling Technique) มาช่วยในการทรานสโพสเมตริกซ์ โดยประมวลผลบนระบบมัลติคอร์ (Multicore System) การทรานสโพสเมตริกซ์ด้วยวิธีการใหม่มีจำนวนการทรานสโพสน้อยกว่าการทรานสโพสเมตริกซ์แบบดั้งเดิมในทุกขนาดของ t และมีจำนวนการทรานสโพสเทียบเท่ากับวิธีการไทลิงก์ แต่ข้อมูลของเมตริกซ์ B จะเป็นการทรานสโพสที่สมบูรณ์ ซึ่งเหมาะกับการคูณเมตริกซ์แบบขนานที่ใช้ MPI ในส่วนของการคูณเมตริกซ์นั้นพบว่าเวลาในการประมวลผลด้วยวิธีการใหม่ที่ได้จากงานวิจัยนี้ มีประสิทธิภาพดีกว่าวิธีการคูณเมตริกซ์ทรานสโพสโดยใช้วิธีไทลิงก์ ในทุกขนาดของเมตริกซ์ และวิธีที่มีประสิทธิภาพที่สุดคือวิธี cbmtm_w



รูปที่ 6 การเปรียบเทียบจำนวนการทรานสโพส ด้วยวิธี TF,PT และ NPT เมื่อ n = 256

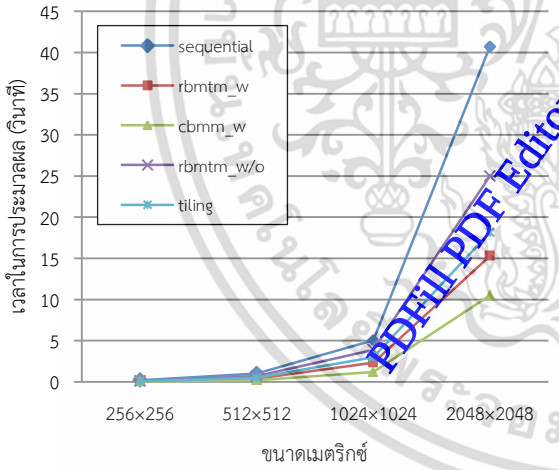
รูปที่ 7 การเปรียบเทียบจำนวนการทรานสโพส ด้วยวิธี TF, PT และ NPT เมื่อ n = 512



รูปที่ 8 การเปรียบเทียบจำนวนการทรานสโพส ด้วยวิธี TF, PT และ NPT เมื่อ n = 1024

ตารางที่ 1 เวลาที่ใช้ (Response Time) ในการคูณเมตริกซ์ทั้ง 5 วิธี

ขนาดของเมตริกซ์	วิธีแบบอนุกรม	วิธีไทลิงก์	วิธีแบบขนาน		
			rbmtm_w	cbmtm_w	rbmtm_w/o
256×256	0.208	0.079	0.077	0.053	0.084
512×512	0.996	0.492	0.373	0.257	0.412
1024×1024	5.049	2.995	1.882	1.496	3.169
2048×2048	40.718	18.131	15.274	10.469	25.981



รูปที่ 9 เวลาที่ใช้ในการคูณเมตริกซ์ ทั้ง 5 วิธี

เอกสารอ้างอิง

[1] M.Kim, Y.Jang and W.Ro. "Parallel Transpose of Matrix Multiplication Based on the Tiling Algorithms," IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS), 2011, pp.1-3.

[2] P.Chen, K.Dai, D.Wu, J.Rao and X.Zou. "The Parallel Algorithm Implementation of Matrix Multiplication Based on ESCA," IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2010, pp.1091-1094.

[3] S.Hunoid, T.Rauber and G.Runger. "Combining buliding blocks for parallel multi-level matrix multiplication," Parallel Computing, Volume 34, Issues 6-8, July 2008, pp.411-426.

[4] M.Krishnan and J.Nieplocha, "Optimizing Parallel Multiplication Operation for Rectangular and Transposed Matrices," 10th International Conference on Parallel and Distributed Systems (ICPADS), IEEE Computer Society ,7-9 July 2004, USA, pp.257-266.

[5] M.Krishnan and J.Nieplocha. "SRUMMA: A Matrix Multiplication Algorithm Suitable for Clusters and Scalable Shared Memory Systems," 18th International Conference on Parallel and Distributed Processing Sysmpsium (IPDPS), IEEE Computer Society ,26-30 April 2004, USA, pp.70-80.

[6] P. Samutrak, "Parallel Matrix Multiplication on a Cluster of PCs," A master of science thesis in computer science, school of graduate studies, King Mongkut's Institute of Technology ladkrabang (KMITL), 2005.

[7] H.J.Lee and P.Robertson, "Generalized Cannon's algorithm for parallel matrix multiplication," Proceedings of the 11th international conference on Supercomputing (ICS'97),07-11 July 97, Vienna, Austria, pp.44-51.

ประวัติผู้เขียน

ชื่อ – สกุล	นายนันทิพัฒน์ พิศุทธางกูร
วัน เดือน ปีเกิด	3 มีนาคม 2532
ที่อยู่	230 หมู่6 ต.บ้านโพธิ์ อ.เมือง จ.ตรัง 92000
ประวัติการศึกษา	
2553	จบการศึกษาปริญญาวิทยาศาสตรบัณฑิต สาขาคณิตศาสตร์ประยุกต์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
2554 - ปัจจุบัน	กำลังศึกษาวิทยาศาสตรมหาบัณฑิต สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง