การหาเส้นทางที่สั้นที่สุดแบบเสถียรด้วยขั้นตอนวิธีแบบขนาน

PARALLEL ALGORITHMS FOR ALL-PAIR SHORTEST PATHS WITH RELIABLE ROUTING

นนทกร เกิดมงคล

ปฏิพล ทวีชาติ

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร

ปริญญาวิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)

ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2560

PARALLEL ALGORITHMS FOR ALL-PAIR SHORTEST PATHS WITH RELIABLE ROUTING

Nontakhon Kerdmongkol

Patipon Taweechat

A SPECIAL PROBLEM SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENT FOR

THE DEGREE OF BACHELOR OF SCIENCE (COMPUTER SCIENCE)

DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2017

หัวข้อปัญหาพิเศษ	การหาเส้นทางที่สั้นที่สุดแบบเสถียรด้วยขั้นตอนวิธีแบบขนาน PARALLEL ALGORITHMS FOR ALL-PAIR SHORTEST PATHS WITH RELIABLE ROUTING
ชื่อนักศึกษา	นาย นนทกร เกิดมงคล 57050252 นาย ปฏิพล ทวีชาติ 57050269
ปริญญา	วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชา	วิทยาการคอมพิวเตอร์
ปีการศึกษา	2560
อาจารย์ที่ปรึกษา	รศ.ดร.จีรพร วีระพันธุ์

คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (สจล.) อนุมัติให้ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตร์บัณฑิต (วิทยาการคอมพิวเตอร์) ประจำปีการศึกษา 2560

คณะกรรมการสอบ	ลายมือชื่อ
ผศ.ดร.ศรัณย์ อินทโกสุม ประธานกรรมการ	
ดร.กุลสวัสดิ์ จิตขจรวานิช กรรมการ	
รศ.ดร.จีรพร วีระพันธุ์ อาจารย์ที่ปรึกษา	

หัวข้อโครงงานพิเศษ	การหาเส้นทางที่สั้นที่สุดแบบเสถียรด้วยขั้นตอนวิธีแบบขนาน PARALLEL ALGORITHMS FOR ALL-PAIR SHORTEST PATHS WITH RELIABLE ROUTING
ชื่อนักศึกษา	นาย นนทกร เกิดมงคล 57050252 นาย ปฏิพล ทวีชาติ 57050269
ปริญญา	วิทยาศาสตรบัณฑิต
สาขา	วิทยาการคอมพิวเตอร์
คณะ	วิทยาศาสตร์
มหาวิทยาลัย	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (สจล.)
ปีการศึกษา	2560
อาจารย์ที่ปรึกษา	รศ. ดร. จีรพร วีระพันธุ์

บทคัดย่อ

งานวิจัยนี้ได้นำเสนอกลยุทธ์การปรับปรุงประสิทธิภาพของขั้นตอนวิธีของฟลอยด์และการประยุกต์การคูณเมทริกซ์ซึ่งเป็นขั้นตอนวิธีหาระยะทางที่สั้นที่สุดจากทุกจุด โดยจะพัฒนาขึ้นให้สามารถแสดงเส้นทางได้ มีความเสถียร และเพิ่มความเร็วด้วยขั้นตอนวิธีแบบขนาน จากงานวิจัยพบว่าการพัฒนาให้สามารถแสดงเส้นทางได้จะใช้พื้นที่หน่วยความจำเพิ่มขึ้นอีก n^2 สมาชิก เมื่อ n คือจำนวนโหนด ในขั้นตอนวิธีแบบฟลอยด์และประยุกต์การคูณเมทริกซ์พบว่าการประยุกต์การคูณเมทริกซ์ทำงานช้ากว่าฟลอย 6 เท่าที่ n=500 และเพิ่มขึ้น เมื่อ n เพิ่มขึ้นโดยจะเพิ่มขึ้นอย่างมีนัยสำคัญ สำหรับในการเพิ่มให้มีความสามารถแบบเสถียรจะทำให้ขั้นตอนวิธีสามารถหาเส้นทางที่สั้นที่สุดใหม่ (เมื่อเส้นทางเดิมเกิดขัดข้อง) กรณีดีที่สุดใช้ความซับซ้อนด้านเวลา $O(n^2)$ ในกรณีที่แย่ที่สุดยังต้องคำนวณใหม่ทั้งหมดโดยใช้ความซับซ้อนด้านเวลา $O(n^3)$ ทั้งสองกรณีจะใช้หน่วยความจำเพิ่มขึ้นเป็น n^3 สมาชิก ในการพัฒนาขั้นตอนวิธีและทำงานแบบขนานพบว่าจำนวนโปรเซสเซอร์มากขึ้นเวลาที่ใช้ในการประมวลผลจะลดลงอย่างมีนัยสำคัญ

TITLE	PARALLEL ALGORITHMS FOR ALL-PAIR SHORTEST PATHS WITH RELIABLE ROUTING
Student	Mr. Nontakhon Kerdmongkol 57050252
	Mr. Patipon Taweechat 57050269
Degree	Bachelor of Science
Department	Computer science
Faculty	Science
University	King Mongkut's Institute of Technology Ladkrabang (KMITL)
Academic Year	2017
Advisor	Assoc.Prof. Dr. Jeeraporn Werapun

ABSTRACT

This research proposes a method for improving the efficiency of Floyd Warshall and Matrix Multiplication algorithms which are All-Pair Shortest Paths algorithms. They are developed to be able to detect nodes along the path, add reliability, and increase the speed of the result by parallel computing. Based on the research, it was found that it required the memory space up to n^2 elements, where n is a number of nodes. Comparing between Floyd's and Matrix Multiplication algorithms, it was found that the Matrix Multiplication algorithm was 6 times slower than Floyd's algorithm when $n \geq 500$ and when n increase, the times increased significantly. The algorithm can recalculate the new shortest route when the latest route is not possible, to increase the reliability. Time complexity of the sequential process is $O(n^2)$ in best case and $O(n^3)$ in worst case and it sequential the memory space up to n^3 elements. In the development process, it was found that when the number of processors increased, the processing time decreased significantly.

กิตติกรรมประกาศ

ปัญหาพิเศษนี้สำเร็จลุล่วงได้ด้วยดี จากการได้รับคำแนะนำ คำชี้แจง ความรู้ แนวทางในการจัดทำ จาก รศ.ดร.จีรพร วีระพันธุ์ ผู้เป็นอาจารย์ที่ปรึกษา ซึ่งท่านได้สละเวลาให้กับข้าพเจ้าอย่างเต็มที่ จึงขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณ ผศ.ดร.ศรัณย์ อินทโกสุม ประธานคณะกรรมการสอบปัญหาพิเศษ และ ดร.กุลสวัสดิ์ จิตขจรวานิช กรรมการสอบปัญหาพิเศษที่กรุณาให้คำแนะนำตลอดจนชี้แนะข้อผิดพลาดต่าง ๆ ส่งเสริมให้ผู้วิจัยมีมุมมองรอบด้านทำงานอย่างรอบคอบ เก็บรายละเอียดในทุกขั้นตอน และทำให้งานออกมาได้สำเร็จ

ขอขอบคุณเพื่อนๆ และผู้ทีเกี่ยวข้องที่คอยให้คำปรึกษาปัญหาพิเศษเล่มนี้ คอยแนะนำ ติชม สนับสนุน และช่วยเหลือในบางด้านทำให้จัดทำปัญหาพิเศษออกมาได้อย่างดี

สำหรับคุณงามความดีและประโยชน์อันใดที่เกิดขึ้นจากปัญหาพิเศษฉบับนี้ ข้าพเจ้าขอมอบแด่ บิดา มารดา ผู้ซึ่งให้โอกาสทางการศึกษาและสนับสนุนทั้งแรงกาย แรงใจ เป็นผู้คอยห่วงใยเอาใจใส่เสมอมา พร้อมคอยอยู่ข้างหลังเป็นกำลังใจให้ข้าพเจ้าตั้งใจเรียนจนได้รับโอกาสที่จะแสวงหาความรู้และพัฒนาตนเองจนเกิดปัญหาพิเศษฉบับนี้ขึ้นมาได้ ผู้วิจัยสำนึกถึงพระคุณในข้อนี้เป็นอย่างสูง

นนทกร เกิดมงคล

ปฏิพล ทวีชาติ

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญ

ปัจจุบันปัญหาในการใช้ชีวิตประจำวันไม่ว่าจะเป็นด้านการขนส่ง การเดินทาง อุตสาหกรรม การเงิน และการคมนาคม พบว่ามีความสูญเสียด้านการขนส่ง ซึ่งการขนส่งเป็นกิจกรรมที่ไม่ก่อให้เกิดมูลค่าเพิ่มแก่วัสดุ ดังนั้นจึงต้องควบคุมและลดระยะทางในการขนส่งลงให้เหลือเท่าที่จำเป็นเท่านั้น และหนึ่งในขั้นตอนวิธีที่ใช้แก้ปัญหาดังกล่าวคือขั้นตอนวิธีการหาเส้นทางที่สั้นที่สุดจากทุกจุด (All-Pair Shortest Paths) ซึ่งขั้นตอนวิธีการหาเส้นทางที่สั้นที่สุดจากทุกจุดยังไม่มีการนำมาปรับใช้ให้เข้ากับเทคโนโลยีปัจจุบันได้อย่างเต็มที่ และยังขาดความสามารถบางอย่างที่จำเป็น อาทิเช่น การแสดงเส้นทาง และการหาเส้นใหม่ได้อย่างรวดเร็ว

ผู้วิจัยจึงได้นำขั้นตอนวิธีหาเส้นทางที่สั้นที่สุดแบบอนุกรม (Sequential) มาปรับปรุง โดยออกแบบขั้นตอนวิธีและนำมาพัฒนาเพิ่มให้สามารถแสดงเส้นทางได้ และสามารถนำทางได้อย่างเสถียร (Reliable) เพราะขั้นตอนวิธีแบบเดิมไม่มีการเก็บเส้นทาง และถ้าเส้นทางที่สั้นที่สุดเกิดความเสียหายจะไม่สามารถหาเส้นทางใหม่ได้ทันทีเนื่องจากต้องมีการคำนวณใหม่ทั้งหมด นอกจากนี้ทางผู้วิจัยจะเพิ่มประสิทธิภาพขั้นตอนวิธีเดิมให้มีความเร็วมากขึ้น

ดังนั้นผู้วิจัยจึงได้นำขั้นตอนวิธีแบบอนุกรมมาพัฒนาเพิ่มเพื่อรองรับฟังก์ชันสำคัญ 2 ฟังก์ชัน คือ 1.ทำให้มีความสามารถเก็บเส้นทางได้ และ 2.มีความเสถียรโดยหาเส้นทางใหม่ได้ทันทีซึ่งมีการคำนวณน้อยที่สุดหากเส้นทางปัจจุบันเกิดความผิดพลาด นอกจากนี้ยังมีการปรับขั้นตอนวิธีแบบเดิมให้เป็นขั้นตอนวิธีแบบขนาน (Parallel Algorithm) เพื่อเพิ่มประสิทธิภาพด้านความเร็วให้มากยิ่งขึ้น

1.2 วัตถุประสงค์

1.2.1 เพื่อออกแบบขั้นตอนวิธีและพัฒนาโปรแกรมการหาระยะทางที่สั้นที่สุดที่สามารถแสดงเส้นทางได้

1.2.2 ปรับปรุงขั้นตอนวิธีแบบเดิมให้มีความเสถียร เมื่อเส้นทางเดิมเสียหายจะสามารถหาเส้นใหม่ได้ทันที

1.2.3 เพื่อปรับขั้นตอนวิธีแบบอนุกรมให้เป็นขั้นตอนวิธีแบบขนานและพัฒนาโปรแกรมแบบขนาน

1.2.4 เปรียบเทียบประสิทธิภาพของการประมวลผลด้วยขั้นตอนวิธีแบบขนานและแบบอนุกรม

1.3 สมมุติฐาน

1.3.1 ในการประมวลผลขั้นตอนวิธีแบบขนาน การเพิ่มจำนวนโปรเซสเซอร์มากขึ้นเวลาในการประมวลผลจะลดลง จนกว่าจะถึงจุดหนึ่งที่เวลาที่ใช้ในการต่อติดสื่อสารมากกว่าการคำนวณ

1.3.2 ขั้นตอนวิธีแบบขนานและแบบอนุกรมที่พัฒนาขึ้นในงานวิจัยนี้จะสามารถหาเส้นทางสำรองได้ทันทีโดยคำนวณน้อยที่สุดเมื่อเส้นทางหลักเกิดความเสียหาย

1.3.3 ขั้นตอนวิธีใหม่ที่นำเสนอในงานวิจัยชิ้นนี้ (ขั้นตอนวิธีแบบเสถียร) จะมีความคุ้มค่าในการนำไปประยุกต์ใช้มากกว่าขั้นตอนวิธีหาระยะทางที่สั้นที่สุดแบบเดิม (ขั้นตอนวิธีแบบไม่เสถียร)

1.3.4 ขั้นตอนวิธีแบบขนานที่นำเสนอในงานวิจัยชิ้นนี้ จะมีความคุ้มค่าในการนำไปประยุกต์ใช้มากกว่าขั้นตอนวิธีหาระยะทางที่สั้นที่สุดแบบเดิม (ขั้นตอนวิธีแบบอนุกรม)

1.3.5 โปรแกรมแบบขนานและแบบอนุกรมที่สร้างขึ้นในงานวิจัยนี้จะสามารถหาเส้นทางสำรองได้ทันทีโดยคำนวณน้อยที่สุดเมื่อเส้นทางหลักเกิดความเสียหาย

1.3.6 โปรแกรมแบบขนานจะใช้เวลาในการประมวลผลน้อยกว่าโปรแกรมแบบอนุกรม

1.4 ขอบเขตการวิจัย

1.4.1 งานวิจัยนี้จะออกแบบขั้นตอนวิธีการหาระยะทางที่สั้นที่สุด จากทุกจุดดังนี้ 1.แบบอนุกรมที่แสดงเส้นทางได้ 2.แบบอนุกรมที่แสดงเส้นทางได้และเสถียร 3.แบบขนานที่แสดงเส้นทางได้ และ 4.แบบขนานที่แสดงเส้นทางได้และเสถียร

1.4.2 งานวิจัยนี้จะสร้างโปรแกรมหาระยะทางที่สั้นที่สุดจากขั้นตอนวิธีที่ในข้อ 1 เท่านั้น

1.4.3 เพื่อศึกษาประสิทธิภาพระหว่างการประมวลผลแบบขนานและแบบอนุกรม

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.5.1 สร้างขั้นตอนวิธีหาระยะทางที่สั้นที่สุดตามขอบเขตที่กำหนด

1.5.2 เปรียบเทียบประสิทธิภาพระหว่างการประมวลผลแบบขนานและแบบอนุกรมเพื่อเป็นข้อมูลในการเลือกใช้

1.5.3 สร้างโปรแกรมจากขั้นตอนวิธีที่พัฒนาขึ้นตามข้อ 1 โดยขั้นตอนวิธีใหม่จะถูกปรับปรุงให้มีความเสถียรมากขึ้น มีประสิทธิภาพมากขึ้น ซึ่งจะสามารถนำไปต่อยอดในงานอื่น ๆ ที่ใช้ขั้นตอนวิธีแบบเดียวกัน

1.6 นิยามศัพท์

1.6.1 **การประมวลผลแบบขนาน** (Parallel Processing) หมายถึง การทำงานชิ้นหนึ่งด้วยหลายหน่วยประมวลผลที่มีการแบ่งงานให้แต่ละหน่วยประมวลผลทำงานพร้อมกัน

1.6.2 **การประมวลผลแบบอนุกรม** (Sequential Processing) หมายถึง การทำงานที่ หน่วยประมวลผลใดหน่วยประมวลผลหนึ่งรับภาระงานไปแค่หน่วยประมวลผลเดียว

1.6.3 **ขั้นตอนวิธีการหาเส้นทางที่สั้นที่สุดจากทุกจุด** (All-Pair Shortest Paths Algorithm) หมายถึง การหาเส้นทางที่สั้นที่สุดโดยคำนวณจากทุกจุด

1.6.4 **APSP Seq** หมายถึง ขั้นตอนวิธีการหาระยะทางที่สั้นที่สุดจากทุกจุดแบบอนุกรมที่แสดงเส้นทาง (All-Pair Shortest Paths with Path Tracking by Sequential Processing)

1.6.5 **APSP-R Seq** หมายถึง ขั้นตอนวิธีการหาระยะทางที่สั้นที่สุดจากทุกจุดแบบอนุกรมที่แสดงเส้นทางและเสถียร (All-Pairs Shortest Paths with Reliable Path by Sequential Processing)

1.6.6 **APSP Par** หมายถึง ขั้นตอนวิธีการหาระยะทางที่สั้นที่สุดจากทุกจุดแบบขนานที่แสดงเส้นทาง (All-Pair Shortest Paths with Path Tracking by Parallel Processing)

1.6.7 **APSP-R Par** หมายถึง ขั้นตอนวิธีการหาระยะทางที่สั้นที่สุดจากทุกจุดแบบขนานที่แสดงเส้นทางและเสถียร (All-Pair Shortest Paths with Reliable Path by Parallel Processing)

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

ในงานวิจัยชิ้นนี้ได้มีการนำเอาองค์ความรู้ทางวิทยาการคอมพิวเตอร์หลายแขนงมาประยุกต์ใช้ โดยมีทฤษฎีที่เกี่ยวข้องดังนี้

2.1 ขั้นตอนวิธีการหาระยะทางที่สั้นที่สุดจากทุกจุด (All-Pair Shortest Paths algorithm)

All Pair Shortest Path algorithm คือการหาเส้นทางที่มีระยะทางสั้นที่สุด โดยจะเปรียบเทียบหาระยะทางที่สั้นที่สุดของทุก ๆ จุด ซึ่งปัญหานี้ปัจจุบันนิยมประยุกต์ใช้ขั้นตอนวิธีของ Dijkstra และ Floyd-Warshall

2.1.1 ขั้นตอนวิธีของฟรอยด์ (Floyd – Warshall Algorithm)

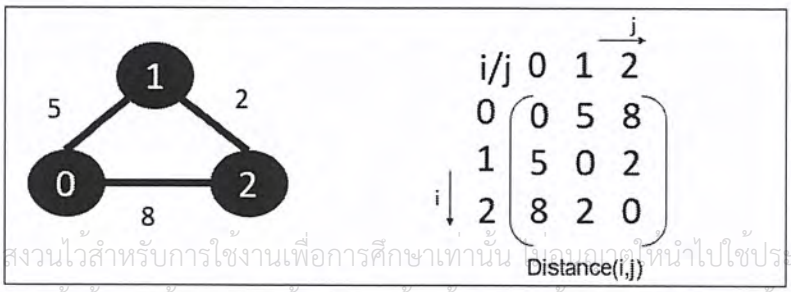
Floyd – Warshall Algorithm คือขั้นตอนวิธีที่ใช้สำหรับหาเส้นทางที่สั้นที่สุดที่เส้นเชื่อมมีระยะทางเป็นบวกหรือเป็นลบ (แต่ต้องไม่มีลบแบบที่เป็นวัฏจักร) วิธีการทำงานของขั้นตอนวิธีนี้คือการรวมค่าของระยะทางของเส้นทางที่สั้นที่สุดซึ่งได้จากการเทียบทุกโหนด ผลลัพธ์ที่ได้จากขั้นตอนวิธีนี้คือผลรวมระยะทางของเส้นทางที่สั้นที่สุด (แต่ไม่สามารถแสดงเส้นทางได้) โดยมีขั้นตอนวิธีดังนี้

สำหรับแต่ละการเทียบกันระหว่างโหนด จากโหนด i ไปหาโหนด j แล้ว k เป็นโหนดใด ๆ อาจเป็นได้ 2 กรณีคือ

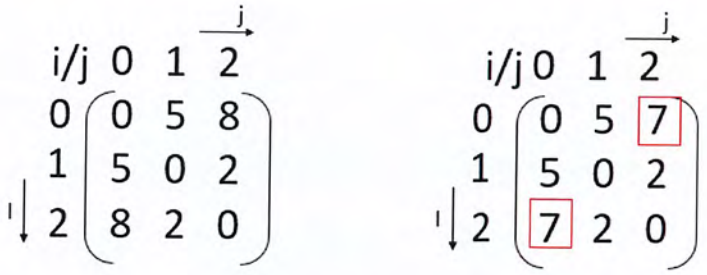
1. เส้นทางที่ไม่ได้ผ่านโหนด k (จากโหนด i ไปหาโหนด j ได้โดยตรง) หรือ
2. เส้นทางที่ผ่านโหนด k นั่นคือ จากโหนด i ไปหา k แล้ว k ไปหา j (โหนด k เป็นโหนดกลาง)

ให้ distance(i, j) คือระยะทางของเส้นเชื่อมของโหนด i ไป j ฉะนั้นระยะทางของเส้นเชื่อมที่น้อยที่สุดคือ

distance(i, j) = min(distance(i, j), distance (i, k) + distance(k, j))



รูปที่ 2.1 แสดงการนำข้อมูลเส้นทางที่เป็นกราฟ (n โหนด) เข้าสู่เมทริกซ์ (ขนาด n x n)

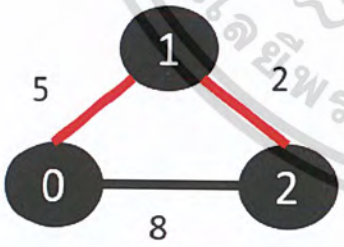


distance(i, j) = min(distance(i, j), distance(i, k) + distance(k, j))

distance(0, 2) = min(distance(0, 2), distance(0, 1) + distance(1, 2))
distance(0, 2) = min(8, 5+2)
distance(0, 2) = min(8, 7)
distance(0, 2) = 7

รูปที่ 2.2 แสดงตัวอย่างการหาเส้นทางที่สั้นที่สุดด้วยวิธีของฟลอยด์ (Floyd – Warshall All-pair shortest path)

ตัวอย่างเช่น จากรูปที่ 2.1 ต้องการหาเส้นทางที่สั้นที่สุดจากโหนด 0 ไปยังโหนด 2 ดังนั้นจึงต้องเปรียบเทียบดูว่าระหว่างกรณีที่ 1 ที่เส้นทางไม่ผ่านโหนด k ใด ๆ (จากโหนด 0 ไปยังโหนด 2 โดยตรง) มากกว่า กรณีที่ 2 ที่โหนด 0 ผ่านโหนด k ใด ๆ แล้วไปยังโหนด 2 ซึ่งถ้าลองผ่านโหนดที่ k = 1 จะได้ว่าโหนด 0 จะไปหาโหนด 1 ก่อนแล้วจึงไปยังโหนด 2 (0->1->2) เมื่อลองนำมาเปรียบเทียบระยะทางจะได้ว่า กรณีที่ 2 จะได้ระยะทางที่น้อยกว่าคือ 7 ดังนั้นเส้นทางนี้จึงเลือกที่จะผ่าน 1 แทนการที่จะเริ่มจากโหนด 0 และไปยังโหนด 2 โดยตรง แสดงเป็นกราฟได้ดังรูปที่ 2.2



กรณีที่ 1 distance(0,2) =8
กรณีที่ 2 distance(0,1) + distance(1,2) = 5+2 = 7

รูปที่ 2.3 แสดงตัวอย่างกราฟ (n=0) จากการหาเส้นทางใหม่ สำหรับโหนด 0 ไปยัง โหนด 2

แต่เนื่องจากการหาเส้นทางที่สั้นที่สุดนั้นจะต้องเปรียบเทียบการผ่านทุกโหนด ดังนั้นจะได้ว่า

distance (0, 2) = min (distance(0, 2), distance(0, 0) + distance(0, 2)) เมื่อ k=0

หรืออาจเป็น min (distance(0, 2), distance(0, 1) + distance(1, 2)) เมื่อ k=1

หรืออาจเป็น min (distance(0, 2), distance(0, 2) + distance(2, 2)) เมื่อ k=2 เขียนโดยรวมได้ว่า

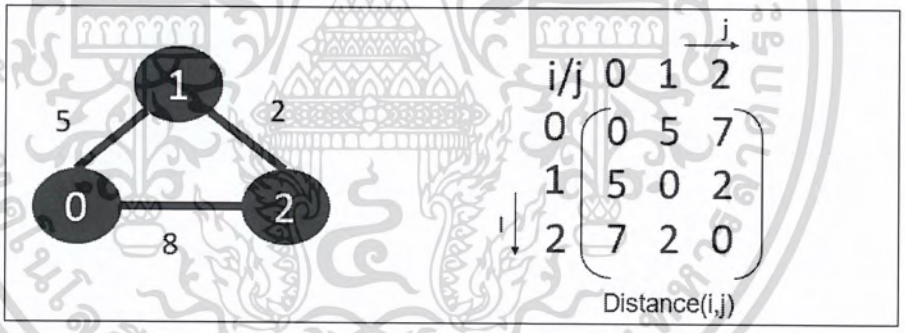
min(distance(0, 2), distance(0, 0) + distance(0, 2), distance(0, 1) + distance(1, 2), distance(0, 2)+ distance(2, 2))

และเนื่องจากจะต้องหาระยะทางที่สั้นที่สุดจากทุกจุด ดังนั้นจึงต้องคำนวณหาให้ครบทุกโหนดต้นทางและโหนดปลายทาง ดังขั้นตอนวิธีดังนี้

```
1. let distance be a n x n array
2. for k from 0 to n-1 do  // panning node(k)
3.   for i from 0 to n-1 do  // source (i)
4.     for j from 0 to n-1 do // destination
5.       if distance [i][j] < distance [i][k] + distance [k][j] then
6.       distance [i][j] = distance [i][k] + distance [k][j]
7.     end if
```

ขั้นตอนวิธีที่ 2.1 ขั้นตอนวิธีการหาเส้นทางที่สั้นที่สุดของฟลอยด์

(Floyd – Warshall All-pair Shortest Path)



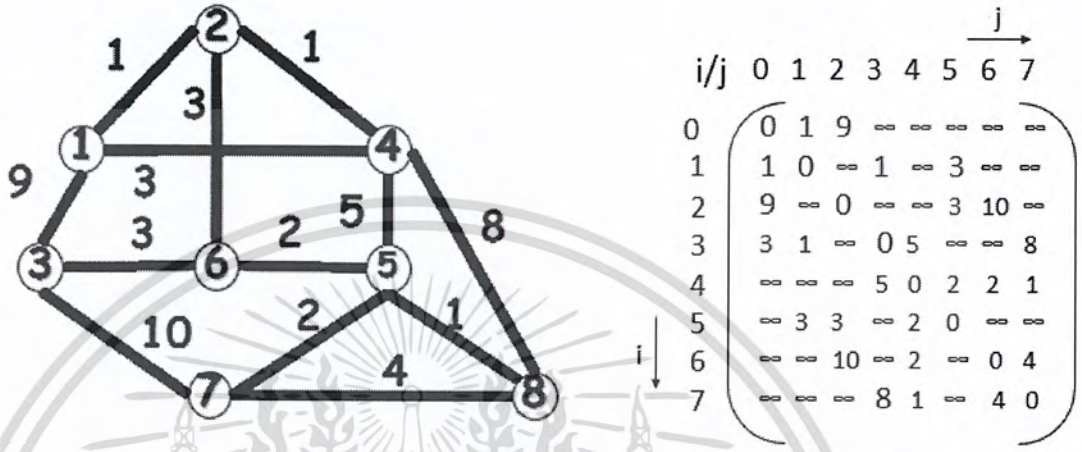
distance(0, 0) = 0 distance(0, 1) = 5 distance(0, 2) = 7
distance(1, 0) = 5 distance(1, 1) = 0 distance(1, 2) = 2
distance(2, 0) = 7 distance(2, 1) = 2 distance(2, 2) = 0

รูปที่ 2.4 แสดงผลลัพธ์การคำนวณที่เสร็จเรียบร้อย

จากรูปที่ 2.4 เมื่อคำนวณหาระยะทางที่สั้นที่สุดจากโหนด i ไปยังโหนด j สำหรับทุกจุดและนำทุกโหนดมาเป็นทางผ่านแล้ว(k=0,1,2,...,n-1) ผลลัพธ์จะได้ว่าเป็นระยะทางที่สั้นที่สุด

2.1.2 ขั้นตอนวิธีประยุกต์ใช้การคูณเมทริกซ์ (Matrix Multiplication Algorithm)

ขั้นตอนวิธีประยุกต์ใช้การคูณเมทริกซ์ เป็นการหาเส้นทางที่สั้นที่สุดโดยปรับมาจากขั้นตอนวิธีการคูณเมทริกซ์ และในลำดับแรกต้องนำข้อมูลเส้นทางที่เป็นกราฟเข้าสู่เมทริกซ์ และหลังจากนั้นจึงนำมาประมวลผลตามขั้นตอนวิธีที่ประยุกต์ขึ้นมา



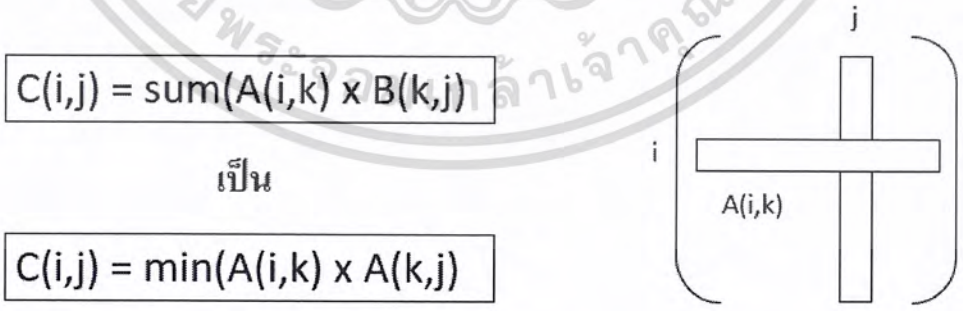
รูปที่ 2.5 แสดงการนำข้อมูลเส้นทางที่เป็นกราฟ (n=8) เข้าสู่เมทริกซ์ขนาด nxn

หลังจากที่มีการนำข้อมูลเส้นทางที่เป็นกราฟเข้าสู่เมทริกซ์แล้ว เราจะสามารถหาเส้นทางที่สั้นที่สุดโดยใช้ขั้นตอนการคูณเมทริกซ์มาประยุกต์ ซึ่งจะมีการปรับจากขั้นตอนการคูณเมทริกซ์ คือ

$C(i,j) = \text{sum}(A(i,k) \times B(k,j))$ เมื่อ $0 \le k < n$ นำมาปรับ

เปลี่ยนจากการคูณเป็นการบวกและจากการบวกเป็นหาค่า min แทน จะได้

$C(i,j) = \min(A(i,k) + A(k,j))$



รูปที่ 2.6 แสดงการการปรับจากขั้นตอนวิธีคูณเมทริกซ์มาใช้หาเส้นทางที่สั้นที่สุด

จากรูปที่ 2.6 แสดงความแตกต่างการคำนวณของขั้นตอนวิธีการคูณเมทริกส์กับการหาระยะทางที่สั้นที่สุดด้วยการประยุกต์ใช้การคูณเมทริกซ์

$$C(i,j) = sum(A(i,k) \times B(k,j))$$

C(0,0) = sum(2*2,5*4) = 24
C(0,1) = sum(2*5,5*1) = 15
C(1,0) = sum(2*4,1*4) = 12
C(1,1) = sum(5*4,1*4) = 21

$$\begin{array}{c} \text{i/j} \quad 0 \quad 1 \\ 0 \\ 1 \end{array} \begin{bmatrix} 2 & 5 \\ 4 & 1 \end{bmatrix}$$

$$\begin{array}{c} \text{i/j} \quad 0 \quad 1 \\ 0 \\ 1 \end{array} \begin{bmatrix} 2 & 5 \\ 4 & 1 \end{bmatrix} \quad \begin{bmatrix} 24 & 15 \\ 12 & 21 \end{bmatrix}$$

Matrix multiplication

$$C(i,j) = min(A(i,k) \times A(k,j))$$

C(0,0) = min(2+2,5+4) = 4
C(0,1) = min(2+5,5+1) = 6
C(1,0) = min(2+4,4+1) = 5
C(1,1) = min (5+4,1+1) = 2

$$\begin{array}{c} \text{i/j} \quad 0 \quad 1 \\ 0 \\ 1 \end{array} \begin{bmatrix} 2 & 5 \\ 4 & 1 \end{bmatrix}$$

$$\begin{array}{c} \text{i/j} \quad 0 \quad 1 \\ 0 \\ 1 \end{array} \begin{bmatrix} 2 & 5 \\ 4 & 1 \end{bmatrix} \quad \begin{bmatrix} 4 & 6 \\ 5 & 2 \end{bmatrix}$$

APSH apply by Matrix multiplication

รูปที่ 2.7 แสดงความแตกต่างระหว่างผลลัพธ์ของการคำนวณในขั้นตอนวิธีทั้งสองแบบ

ขั้นตอนวิธีประยุกต์ใช้การคูณเมทริกซ์สามารถเขียนเป็นโค้ดเทียมได้ดังนี้

```
1. let A,B array of size n x n
2. for r = 1 to log(n) do
3. B=A
4. for I = 0 to n-1 do
5. for j = 0 to n-1 do
6. for k = 0 to n-1 do
7. if A[i][j]  < B[i][k] + B[k][j] then
8. A[i][j] = B[i][k] + B[k][j]
```

ขั้นตอนวิธีที่ 2 ประยุกต์ใช้การคูณเมทริกซ์ (Matrix Multiplication)

จากขั้นตอนวิธีที่ 2 จะมีการทำงานทั้งหมด log(n) ครั้ง ซึ่งแต่ละครั้งของลูปจะเกิดการลองนำเอาโหนดต่าง ๆ ไปผ่านโหนดใดๆ แล้วจะได้เมทริกซ์ที่ผ่านการคำนวณในรอบนั้นๆ การผ่านการคำนวณ หรือ ผ่านแต่ละโหนดเราจะแทนด้วย A^k โดย A คือ เมทริกซ์ และ k-1 คือจำนวนรอบที่ทำงานหรือที่ผ่านโหนดมาแล้ว เมื่อทำงานเสร็จสิ้นค่าจะได้ค่า k=n ดังนั้นจะเห็นได้ว่า

เมื่อ k = 1 จะได้ว่าไม่ผ่านโหนดใด ๆ เป็นเมทริกซ์ตั้งต้นยังไม่ผ่านการคำนวณ

เมื่อ k = 2 ผ่านการคำนวณมาแล้ว 1 รอบ โดยลองผ่านโหนดแล้ว 1 โหนด

เมื่อ k = 3 ผ่านการคำนวณมาแล้ว 2 รอบ โดยลองผ่านโหนดแล้ว 2 โหนด

เมื่อ k = n นั่นคือผ่านมาแล้ว n-1 รอบ โดยลองผ่านโหนดแล้ว n-1 โหนด

และเมื่อได้ผลลัพธ์จากการคำนวณครบ 1 รอบจะได้ A^2 ให้นำค่าที่ได้มาคำนวณซ้ำตามขั้นตอนวิธีที่ 2 จะได้ $A^2 \times A^2 = A^4$ และ $A^4 \times A^4 = A^8$ ดังนั้นการทำวิธีดังกล่าวจะทำให้ลดจำนวนครั้งในการคำนวณ จาก 8 ครั้ง เหลือเพียง log(8) ซึ่งก็คือ เพียง 3 รอบ และเขียนเป็นสูตรได้ว่า จำนวนครั้งของการคำนวณจะมีค่าเป็น log(n)

ตัวอย่างการคำนวณโดยกำหนดเมทริกซ์ดังนี้

$A^1 =$

i/j	0	1	2	3	4	5	6	7
0	0	1	9	∞	∞	∞	∞	∞
1	1	0	∞	1	∞	3	∞	∞
2	9	∞	0	∞	∞	3	10	∞
3	3	1	∞	0	5	∞	∞	8
4	∞	∞	∞	5	0	2	2	1
5	∞	3	3	∞	2	0	∞	∞
6	∞	∞	10	∞	2	∞	0	4
7	∞	∞	∞	8	1	∞	4	0

$A^2 =$

i/j	0	1	2	3	4	5	6	7
0	0	1	9	2	8	4	19	11
1	1	0	6	1	5	3	∞	9
2	9	6	0	12	5	3	10	14
3	2	1	12	0	5	4	7	6
4	8	5	5	5	0	2	2	1
5	4	3	3	4	2	0	4	3
6	19	∞	10	7	2	4	0	3
7	11	9	14	6	1	3	3	0

รูปที่ 2.8 แสดงการคำนวณที่ A^1 และ A^2

$A^2 \times A^2 = A^4 =$

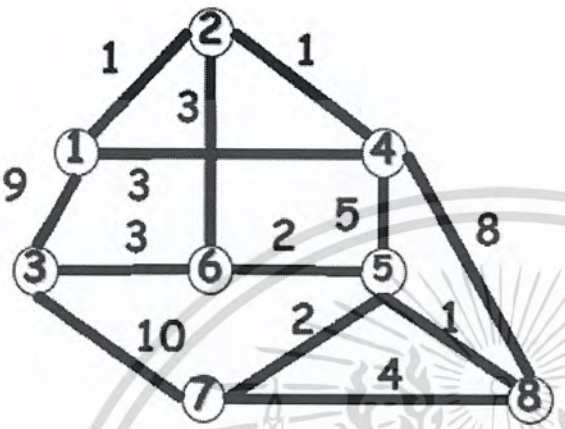
i/j	0	1	2	3	4	5	6	7
0	0	1	7	2	6	4	8	7
1	1	0	6	1	5	3	7	6
2	7	6	0	7	5	3	7	6
3	2	1	7	0	5	4	7	6
4	6	5	5	5	0	2	2	1
5	4	3	3	4	2	0	4	3
6	8	7	7	7	2	4	0	3
7	7	6	6	6	1	3	3	0

$A^4 \times A^4 = A^8 =$

i/j	0	1	2	3	4	5	6	7
0	0	1	7	2	6	4	8	7
1	1	0	6	1	5	3	7	6
2	7	6	0	7	5	3	7	6
3	2	1	7	0	5	4	7	6
4	6	5	5	5	0	2	2	1
5	4	3	3	4	2	0	4	3
6	8	7	7	7	2	4	0	3
7	7	6	6	6	1	3	3	0

รูปที่ 2.9 แสดงการคำนวณที่ A^4 และ A^8

จากรูปที่ 2.8 และ 2.9 เราได้กำหนด A^1 เป็นเมทริกซ์ที่เราแปลงมาจากกราฟ มีจำนวนโหนดทั้งหมด 8 โหนด และทำการคำนวณตามขั้นตอนวิธีที่ 2 ดังนั้นจะคำนวณทั้งหมด log(n) =log(8)=4 ครั้ง ซึ่งจะได้ผลลัพธ์ (ผ่านโหนด 7 โหนด) เป็นเมทริกซ์ที่คำนวณเสร็จสิ้นแล้ว เป็น A^8



i/j	0	1	2	3	4	5	6	7
0	0	1	7	2	6	4	8	7
1	1	0	6	1	5	3	7	6
2	7	6	0	7	5	3	7	6
3	2	1	7	0	5	4	7	6
4	6	5	5	5	0	2	2	1
5	4	3	3	4	2	0	4	3
6	8	7	7	7	2	4	0	3
7	7	6	6	6	1	3	3	0

รูปที่ 2.10 แสดงผลลัพธ์การคำนวณที่เสร็จเรียบร้อย

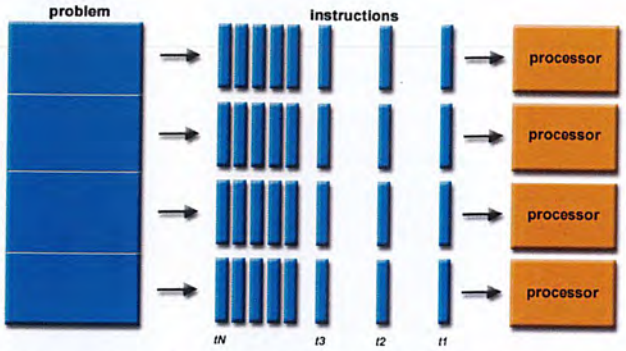
จากรูปที่ 2.10 หากเริ่มที่โหนด 1 ไป โหนด 3 จะพบว่าระยะทางที่สั้นที่สุดมีค่าเป็น 7 ซึ่งมีค่าน้อยที่สุดเป็นระยะทางสั้นที่สุด

2.2 การคำนวณแบบขนาน (Parallel computing)

การคำนวณแบบขนานเป็นประเภทของการคำนวณ เป็นการแบ่งปัญหาที่ใหญ่เป็นปัญหาที่เล็กลงหลายๆปัญหา แล้วจัดได้ปัญหาขนาดเล็กดังกว่าในเวลาเดียวกันหรือทำพร้อมกัน การคำนวณแบบขนานมีหลากหลายรูปแบบ เช่น ระดับบิต ระดับคำสั่ง ระดับข้อมูล และระดับงาน

ในการคำนวณแบบขนานงานย่อย ๆ ที่มีความคล้ายกันมาก ๆ สามารถประมวลผลแบบเป็นอิสระต่อกันได้แล้วนำผลลัพธ์มารวมกันทีหลัง ซึ่งในระหว่างประมวลผลบางครั้งอาจมีการติดต่อกัน

สำหรับในหลาย ๆ การคำนวณแบบขนานนี้ถูกใส่ไว้เป็นส่วนหนึ่งของ High Performance Computing ด้วยการเติบโตและกำลังที่ใช้กลุ่มของคอมพิวเตอร์ ได้รับการกล่าวถึงไม่กี่ปีมาการคำนวณแบบขนานกลายเป็นตัวอย่างมี่สำคัญในสถาปัตยกรรมคอมพิวเตอร์ โดยมีรูปแบบหลักๆคือ มัลติคอร์โปรเซสเซอร์

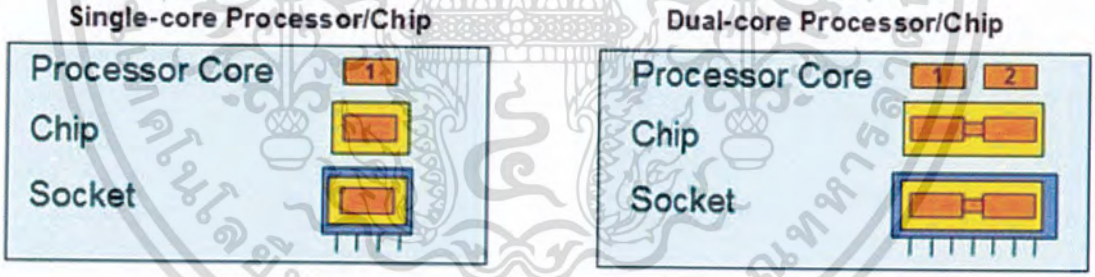


รูปที่ 2.11 แสดงการคำนวณแบบแบบขนาน (Parallel Computing)

2.3 สถาปัตยกรรมคอมพิวเตอร์ (Computer Architecture)

2.3.1 สถาปัตยกรรมแบบหนึ่งหน่วยประมวลผล (Single – Core Architecture)

สถาปัตยกรรมแบบหนึ่งหน่วยประมวลผลในไมโครโปรเซสเซอร์ คือ ไมโครโปรเซสเซอร์ที่มีเพียงคอร์เดียวต่อชิฟหนึ่งตัว ทำงานได้เพียงเธรด(Thread)เดียวในเวลาหนึ่งเท่านั้น แต่ในภายหลังมีการสร้างตัวประมวลผลแบบมัลติคอร์(Multi Core)ออกมา ซึ่งแต่ละโปรเซสเซอร์เป็นอิสระต่อกันในชิฟเดียว ตัวอย่างเช่น Intel มี Core 2 duo ซึ่งต่อยอดมาจากแบบซิงเกิลคอร์(Single core) จะแสดงให้เห็นถึงการเพิ่มประสิทธิภาพในการประมวลผลแบบขนาน

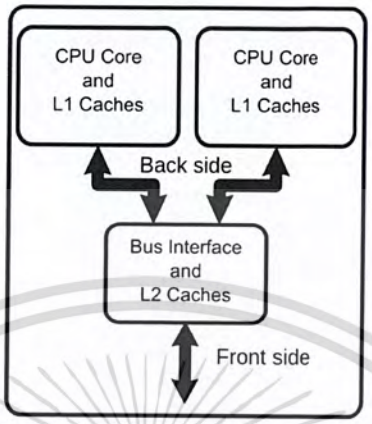


รูปที่ 2.12 สถาปัตยกรรมแบบหนึ่งหน่วยประมวลผลและแบบสองหน่วยประมวลผล (Dual - Core Processor)

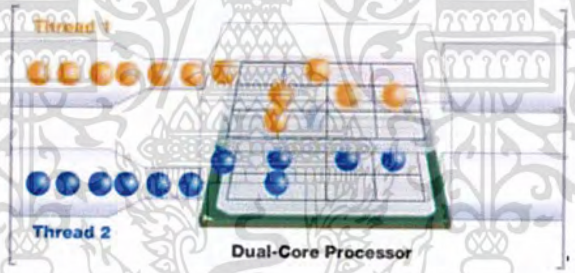
2.3.2 สถาปัตยกรรมหน่วยประมวลผลแบบมัลติคอร์ (Multi – Core Architecture)

การประมวลผลแบบมัลติคอร์ เป็นการประมวลผลที่มีส่วน processing unit อิสระต่อกัน (เรียกว่าคอร์) ซึ่งแต่ละยูนิตสามารถ อ่าน (Read) ดำเนินการ (Execute) คำสั่งต่าง ๆ เช่น เพิ่ม (Add) ย้าย (Move) พร้อมกันได้ แต่ในสถาปัตยกรรมแบบหนึ่งหน่วยประมวลผล ไม่สามารถแยกกระทำในเวลาเดียวกันได้ ซึ่งส่วนนี้จะช่วยเพิ่มความเร็วให้กับการคำนวณแบบขนาน ในการทำ

การประมวลผลแบบมัลติคอร์ นักออกแบบจะต้องออกแบบการใช้แคชร่วมกันดังแสดงในรูปที่ 2.13 หรืออาจทำแบบส่งข้อความระหว่างกันในระหว่างการติดต่อกันแต่ละคอร์



รูปที่ 2.13 แผนภาพของสองหน่วยประมวลผลที่มีแคชของตัวเองที่ระดับหนึ่ง และใช้แคชร่วมกันที่ระดับสอง

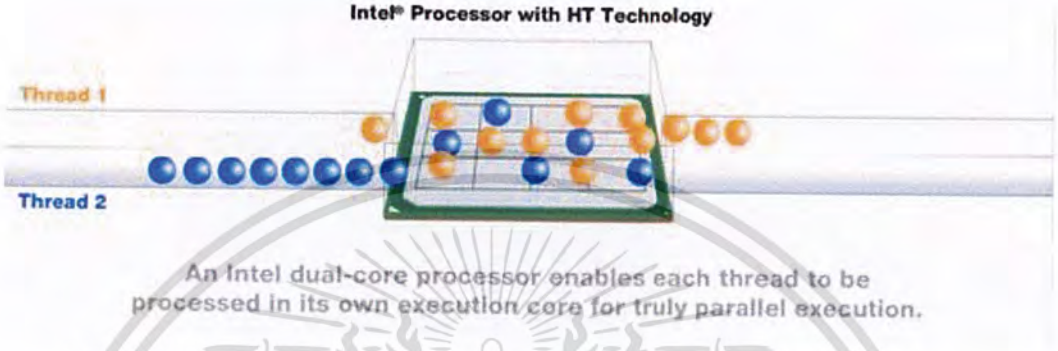


รูปที่ 2.14 การทำงานของสองหน่วยประมวล (Dual - Core Processor)

2.3.3 เทคโนโลยีไฮเปอร์เธรดดิ้ง (Hyper-Threading Technology)

เทคโนโลยีไฮเปอร์เธรดดิ้ง คือเทคโนโลยีของ Intel ในการเอาซีพียูที่มีแกนประมวลผลหลักเพียงแกนเดียว (คอร์เดียว) มาใช้ระบบปฏิบัติการเป็นตัวจัดการเธรดหลายๆเธรดในหนึ่งช่วงเวลา โดยเทคโนโลยีไฮเปอร์เธรดดิ้งจะจำลองซีพียูที่มีทรัพยากรเดียวกันมีแคชชุดเดียว และมีคล็อกชุดเดียว ออกเป็น 2 ซีพียู ซึ่งทำให้มีการดำเนินการได้ 2 เธรด (หรือมากกว่า) ในเวลาเดียวกัน อาศัยช่วงเวลาที่เธรดแรก ไม่ได้ทรัพยากรบางส่วนของซีพียู มาทำการดำเนินการอีกเธรดหนึ่งสลับกันไป แต่ก็ไม่เต็มประสิทธิภาพ เนื่องจากทั้งสองเธรด (หรือมากกว่า) ก็ยังคงใช้ทรัพยากรเดียวกัน ภายใต้ซีพียูคอร์เดียวกันอยู่ เช่น การประมวลผลแบบสองหน่วยประมวลผลซึ่ง มีแกนประมวลผลหลักถึง 2

แกน ทำให้แต่ละเธรดที่ต้องการดำเนินการนั้น สามารถใช้ทรัพยากรของแต่ละคอร์ได้อย่างเต็มที่ ซึ่งแน่นอนว่า ส่งผลดีกว่าการทำงานผ่านไฮเปอร์เธรดดิ้งเพราะแต่ละคอร์ต่างก็มีทรัพยากรเป็นของตัวเอง

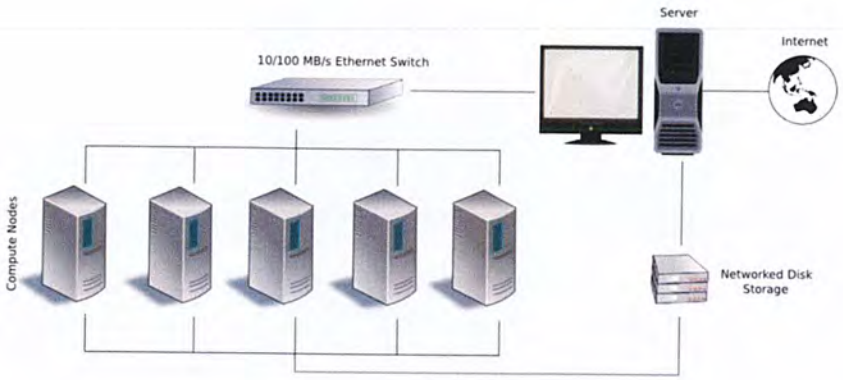


รูปที่ 2.15 การทำงานแบบหนึ่งหน่วยประมวลผลใช้ไฮเปอร์เธรดดิ้ง (Hyper - Threading)

2.4 คอมพิวเตอร์คลัสเตอร์ (Computer Cluster)

คอมพิวเตอร์คลัสเตอร์ ประกอบด้วยเซ็ตของคอมพิวเตอร์ที่เชื่อมต่อกันที่ทำงานร่วมกัน สามารถมองได้เป็นหนึ่งระบบคอมพิวเตอร์คลัสเตอร์มีเซ็ตของแต่ละโหนดในการทำงานเดียวกันและถูกควบคุมและจัดตารางงานโดยซอฟแวร์ส่วนประกอบของคลัสเตอร์ โดยทั่วไปจะเป็นการเชื่อมกันของแต่ละโหนดผ่านแลน (LAN) แต่ละโหนด คือ เครื่องคอมพิวเตอร์ที่ทำหน้าที่เป็นเหมือนเซิฟเวอร์ทำงานโดยมีระบบปฏิบัติการเป็นของตัวเอง ตามสภาพการณ์แล้วทุก ๆ โหนด จะใช้ฮาร์ดแวร์เดียวกันและระบบปฏิบัติการเดียวกัน แต่ถ้าใช้ระบบปฏิบัติการที่แตกต่างกัน ในการติดตั้งระบบอาจจะใช้ Open Source Cluster Application Resources (OSCAR) โดยทั่วไปจะใช้คลัสเตอร์ ในการเพิ่มประสิทธิภาพและความพร้อมใช้งาน ที่คอมพิวเตอร์เครื่องเดียวทำไม่ได้ เพราะว่าการทำคลัสเตอร์ จะได้ประสิทธิผลต่อราคามากกว่าคอมพิวเตอร์เครื่องเดียวทั้งในด้านความเร็วและความใช้งานได้

คอมพิวเตอร์คลัสเตอร์ พัฒนาออกมาจากผลลัพธ์ของแนวโน้มของไมโครโปรเซสเซอร์ที่มีราคาถูก และเครือข่ายความเร็วสูง รวมถึงซอฟแวร์ที่จัดการการประมวลผลแบบกระจายซึ่งถูกใช้กันอย่างกว้างขวางตั้งแต่ธุรกิจขนาดเล็ก เพราะว่าจัดการง่ายกว่าซุปเปอร์คอมพิวเตอร์เช่น IBM's Sequoia



รูปที่ 2.16 การออกแบบและองค์ประกอบของคลัสเตอร์

2.5 MPI (Message Passing Interface)

MPI คือ API ที่จัดการเกี่ยวกับการส่งผ่านข้อความโดย message - passing parallel programming model ซึ่งเป็นการที่ข้อมูลในแต่ละที่อยู่ย้ายจาก โปรเซสหนึ่งไปยังอีกโปรเซสหนึ่งในการทำงานร่วมกัน ซึ่งมีองค์ประกอบหลักๆ ดังนี้

1 รูปแบบการติดต่อในMPI

1) การติดต่อแบบ Point to Point คือ การส่งข้อความระหว่าง 2 โปรเซสโดยตรง

2) การติดต่อแบบเป็นกลุ่ม Collective คือการส่งข้อความให้กับกลุ่มของโปรเซส เช่น การส่ง แบบ Broadcast (1 : m) และการส่งแบบ Reduction (m : 1)

3) การติดต่อแบบ Asynchronous คือ การรับส่งโดยไม่ต้องรอให้ผู้รับ-ส่ง ได้รับข้อความก่อน ช่วยให้สามารถปรับปรุงสมรรถนะของโปรแกรมไปพร้อมๆกับการรับส่งข้อมูลได้

2 แรงค์ (Ranks)

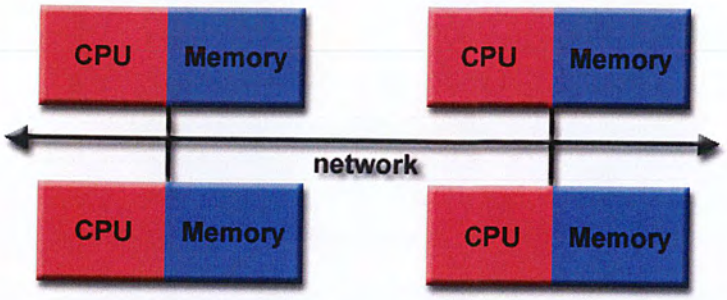
การแบ่งงานออกเป็นโปรเซสย่อย ๆ แต่ละโปรเซสจะมีหน่วยความจำเป็นของตัวเอง ทุกๆ โปรเซสจะได้รับหมายเลขประจำตัว (Rank) หมายเลขหนึ่งจะเริ่มต้นที่หมายเลข 0 เสมอ จนถึงหมายเลข n-1 ranks จะใช้งานในการระบุต้นทางและปลายทางของการรับส่งข้อความระหว่างโปรเซส

3 คอมมูนิเคเตอร์ (Communicator)

คือ เป็นตัวอ้างอิงสำหรับการทำงานเป็นกลุ่ม โดยที่แต่ละโปรเซสจะเชื่อมโยงกับคอมมูนิเคเตอร์ ซึ่งค่าเริ่มต้นของทุก ๆ โปรเซสจะอยู่ในกลุ่มที่ชื่อว่า COMM_WORLD

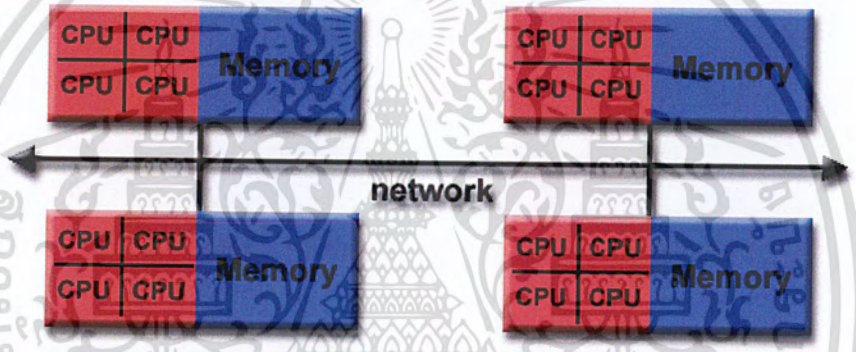
4 โปรแกรมมิ่งโมเดล (Programming Model)

ต้นกำเนิดของMPIถูกออกแบบให้เป็นแบบสถาปัตยกรรมที่มีหน่วยความจำแบบกระจาย (Distributed Memory Architectures)



รูปที่ 2.17 สถาปัตยกรรมที่มีหน่วยความจำแบบกระจาย (Distributed Memory Architectures)

แต่ในภายหลังมีการนำสถาปัตยกรรมที่มีหน่วยความจำแบบร่วมกัน (Shared Memory SMPs) มารวมระหว่างเน็ตเวิร์คจึงเกิดเป็นแบบไฮบริด (Hybrid)



รูปที่ 2.18 สถาปัตยกรรมที่มีหน่วยความจำแบบร่วมกันและกระจาย (Shared and Distributed memory architectures)

ในทุกวันนี้MPIสามารถรันได้บน 3 แพลตฟอร์มคือ สถาปัตยกรรมที่มีหน่วยความจำแบบกระจาย, สถาปัตยกรรมที่มีหน่วยความจำแบบร่วมกัน และสถาปัตยกรรมแบบไฮบริด

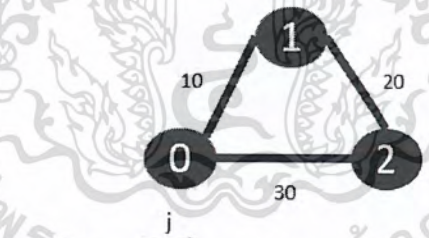
บทที่ 3
การหาเส้นทางที่สั้นที่สุดแบบเสถียรด้วยขั้นตอนวิธีแบบขนาน

การวิจัยครั้งนี้มีจุดประสงค์เพื่อออกแบบขั้นตอนวิธีการหาเส้นทางที่สั้นที่สุดจากทุกจุดแบบเสถียร เพิ่มความสามารถในการแสดงเส้นทาง โดยเริ่มจากวิธีขั้นตอนวิธีแบบอนุกรมและขยายเป็นขั้นตอนวิธีแบบขนานโดยจะนำขั้นตอนวิธีที่ออกแบบไปพัฒนาเป็นโปรแกรมเพื่อทำการทดลองศึกษาเปรียบเทียบหาประสิทธิภาพในการทำงานเพื่อสามารถนำผลลัพธ์ที่ได้ไปประยุกต์ใช้ในการตัดสินใจเพื่อเลือกใช้ขั้นตอนวิธีที่เหมาะสมและรวดเร็ว ในงานต่าง ๆ

3.1 การออกแบบขั้นตอนวิธีเพิ่มความสามารถในการเก็บเส้นทาง และขั้นตอนวิธีในการแสดงเส้นทาง

จากขั้นตอนวิธีการหาเส้นทางที่สั้นที่สุดจากทุกจุดเดิมซึ่งไม่สามารถแสดงเส้นทางได้ ดังนั้นในขั้นตอนวิธีใหม่นี้จึงเพิ่มส่วนของการเก็บเส้นทาง และส่วนของการแสดงเส้นทางในแต่ละโหนดต้นทางและปลายทาง

จากขั้นตอนวิธีเดิมจะเก็บข้อมูลระยะทางได้เพียงอย่างเดียวดังนั้นเราจะสร้างอาเรย์ขึ้นมาอีกชุดขนาด n x n เมื่อ n คือจำนวนโหนด โดยจะนำมาเก็บข้อมูลเส้นทางเพื่อทำให้ขั้นตอนวิธีสามารถแสดงระยะทางและเส้นทางที่สั้นที่สุดได้



i/j	0	1	2
0	0	20	30
1	10	0	20
2	30	20	0

Array collect distance

i/j	0	1	2
0	0	1	2
1	1	0	2
2	2	1	0

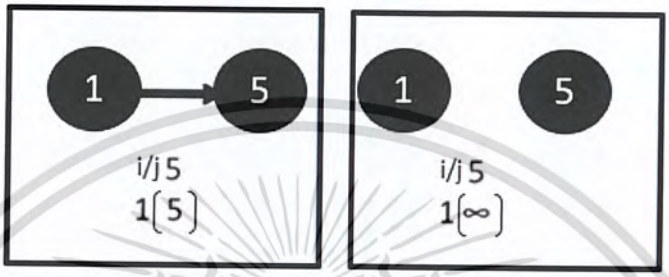
New Array collect path

รูปที่ 3.1 แสดงอาเรย์ใหม่ที่สร้างขึ้นมาเพื่อเก็บเส้นทาง

จากรูปที่ 3.1 แสดงการสร้างอาเรย์ใหม่โดยอาเรย์แรกเป็นอาเรย์ที่ในขั้นตอนวิธีเดิมมีอยู่แล้ว ส่วนอาเรย์ด้านขวาจะเป็นอาเรย์ที่เราจะสร้างขึ้นเพื่อนำมาเก็บข้อมูลเส้นทาง โดยจากขั้นตอนวิธีเดิมนำมาเพิ่มความสามารถดังนี้

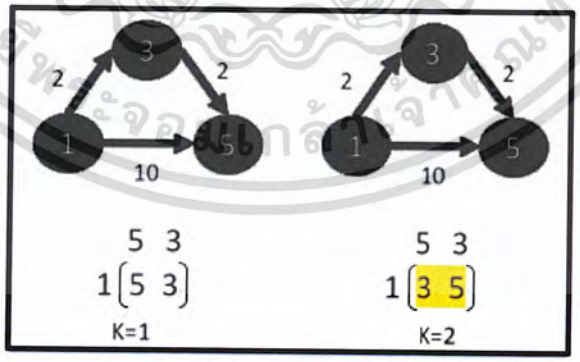
3.1.1 การเพิ่มความสามารถในการเก็บเส้นทาง

โครงสร้างที่ใช้เก็บเส้นทางจะใช้เป็นอาเรย์ขนาด n x n เมื่อ n คือจำนวนโหนด และประกาศค่าเริ่มต้นให้แต่ละเส้นทางมีค่าเท่ากับโหนดปลายทาง เช่น เส้นทางจากจากโหนด 1 ไปยังโหนด 5 ให้เก็บค่าในเส้นทางเท่ากับ 5 เป็นต้น หากเริ่มต้นโหนด 1 ไปยังโหนด 5 นั้นไม่มีเส้นไปหาโหนด 5 ให้ใส่ค่าอินฟินิตี้ (∞)



รูปที่ 3.2 แสดงตัวอย่างการเก็บเส้นทางจากโหนด 1 ไป โหนด 5 ที่มีเส้นทางไป (ซ้าย) และการเก็บเส้นทางจากโหนด 1 ไป โหนด 5 ที่ไม่มีเส้นทางไป (ขวา)

กระบวนการเก็บเส้นทางจะเก็บไปพร้อมกับการคำนวณหาระยะทางที่สั้นที่สุด โดยจะเก็บหมายเลขโหนดที่คำนวณแล้วพบว่าเมื่อผ่านโหนดนั้นจะทำให้ได้ระยะทางที่น้อยกว่าเดิมเช่น จากเดิมเส้นทางจากโหนด 1 ไปยังโหนด 5 เก็บค่าในเส้นทางเท่ากับ 10 เมื่ออยู่ในระหว่างการคำนวณหาเส้นทางที่สั้นที่สุด พบว่าเมื่อผ่านโหนด 3 แล้วจะทำให้ได้ค่าระยะทางที่น้อยกว่า คือ 1->3->5 แล้วจะเก็บค่าในเส้นทางเท่ากับ 4



รูปที่ 3.3 จำลองการเก็บเส้นทาง

โดยที่ k=1 ไม่มีการคำนวณ k=2 ผ่านการคำนวณโดยลองผ่านโหนด 1 โหนด

จากรูปที่ 3.3 สังเกตอาเรย์ด้านซ้ายและอาเรย์ด้านขวาตรงที่ทำการเน้นจะแสดงการเปลี่ยนเส้นทางจากโหนด 1 ไปโหนด 5 ต้องผ่านโหนด 3 ก่อน จะไปโหนด 5 จึงจะเป็นระยะทางที่สั้นที่สุด 1->5 ใช้ระยะ 10 แต่ถ้าผ่านโหนด 3 ก่อน คือ 1->3->5 ใช้ระยะเพียง 4 เท่านั้น

3.1.2 ขั้นตอนวิธีของฟลอยด์

ขั้นตอนวิธีเดิม

```
1. let distance be an array of size n x n
2. get distance
3. for k from 0 to n-1 do
4.   for i from 0 to n-1 do
5.     for j from 0 to n-1 do
6.       if distance[i][j] > distance[i][k] + distance[k][j] then
7.       distance[i][j] = distance[i][k] + distance[k][j]
```

ขั้นตอนวิธีที่ 3.1 ขั้นตอนวิธีการหาเส้นทางที่สั้นที่สุดของฟลอยด์ (Floyd–Warshall All-Pair Shortest Paths)

ขั้นตอนวิธีใหม่ที่มีการเก็บเส้นทาง

```
1. let path be an array of size n x n
2. for i=0 to n-1 do
3.   for j=0 to n-1 do
4.     path[i][j] =j
5. let distance be an array of size n x n
6. get distance
7.   for k = 0 to n-1 do
8.     for i = 0 to n-1 do
9.       for j = 0 to n-1 do
10.        if distance[i][j]  > distance[i][k] + distance[k][j] then
11.          distance[i][j] = distance[i][k] + distance[k][j]
12.          path[i][j] = path[i][k]
```

ความซับซ้อนของเวลา : $O(n^3)$

ขั้นตอนวิธีที่ 3.2 ขั้นตอนวิธีการหาเส้นทางที่สั้นที่สุดของฟลอยด์แบบเก็บเส้นทาง (Floyd–Warshall All-Pair Shortest Paths Path Tracking)

จากขั้นตอนวิธีที่ 3.2 ตรงที่มีการเน้น หรือ ตรงบรรทัดที่ 1,2,3,4,12 เป็นการเพิ่มที่ทำให้มีการเก็บเส้นทางได้โดยบรรทัดที่ 1 เป็นการสร้างอาเรย์ขนาด n x n โดยที่ n คือจำนวนโหนด บรรทัดที่ 2-4 เป็นการเข้าลูปเพื่อนำไปตั้งค่าเริ่มต้นโดยค่าเริ่มต้นของแต่ละค่าในนี้คือค่า j ซึ่งก็คือ ค่าของแต่ละแถว i โดยค่า j จะมีค่าเป็น 0,1,2,...,n-1 โดย n คือจำนวนโหนด และ ขึ้นแถว i ใหม่ จะทำการกำหนดค่า j ตามที่กล่าวไปข้างต้น

3.1.3 ขั้นตอนวิธีประยุกต์การคูณเมทริกซ์

ขั้นตอนวิธีเดิม

```
1.  let temp_distance, distance be an array of size n x n
2.  get distance
3.  for r = 0 to log(n)-1 do
4.    temp_distance = distance with transpose
5.    for i = 0 to n-1 do
6.      for j = 0 to n-1 do
7.        for k = 0 to n-1 do
8.          if distance[i][j]  > distance[i][k] + temp_distance[j][k]  then
9.            distance[i][j]  = distance[i][k] + temp_distance[j][k]
```

ขั้นตอนวิธีที่ 3.3 ประยุกต์ใช้การคูณเมทริกซ์ (Matrix Multiplication)

ขั้นตอนวิธีใหม่ที่มีการเก็บเส้นทาง

```
1.  let path be an array of size n x n
2.  for i=0 to n-1 do
3.    for j=0 to n-1 do
4.      path[i][j] = j
5.  let distance ,temp_distance be an array of size n x n
6.  get distance
7.  for r = 0 to log(n) do
8.    temp_distance = distance with transpose
9.    for I = 0 to n-1 do
10.     for j = 0 to n-1 do
11.       for k = 0 to n-1 do
12.         if distance[i][j]  > distance[i][k] + temp_distance[j][k] then
13.           distance[i][j] = distance[i][k] + temp_distance[j][k]
14.           path[i][j] =path[i][k]
```

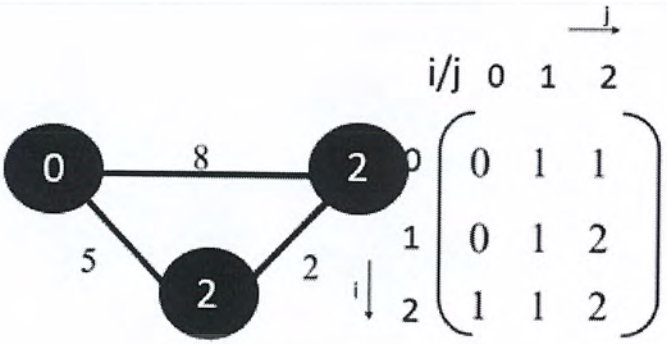
ความซับซ้อนของเวลา : $O(n^3 \log(n))$

ขั้นตอนวิธีที่ 3.4 ประยุกต์ใช้การคูณเมทริกซ์แบบเก็บเส้นทาง
(Matrix Multiplication Path Tracking)

จากขั้นตอนวิธีที่ 3.4 ในการเก็บเส้นทางจะเหมือนกับใน 3.3

3.1.4. ขั้นตอนวิธีในการแสดงเส้นทาง

การนำเส้นทางที่เก็บไว้ในอาเรย์ มาแสดงเส้นทางจะต้องรู้ก่อนว่าโหนดเริ่มต้นเริ่มจากโหนดใดและไปหาโหนดสิ้นสุดใดเช่น จากโหนดเริ่มต้น i ไปยังโหนดเริ่มต้น j โหนด i จะต้องผ่านโหนดอะไรก่อนจึงจะไปไปถึงโหนดสิ้นสุด j เช่นแสดงเส้นทางจากโหนดเริ่มต้น 1 ไปโหนดปลายทาง 5 จะเส้นทางได้เป็น 1->4->2->5 เป็นต้น



รูปที่ 3.4 แสดงตัวอย่างข้อมูลการเก็บเส้นทาง

จากรูปที่ 3.4 เมื่อต้องการแสดงเส้นทางจากโหนด 0 ไปยังโหนด 2 จะได้ว่า ค่าของ path[0][2] เท่ากับ 1 หมายความว่า จากโหนด 0 จะไปยังโหนด 2 ได้นั้นต้องผ่านโหนด 1 ก่อน จะได้เส้นทางเป็น 0->1-> จากนั้นให้ดูว่าโหนดที่ผ่านนั้นจะไปยังโหนด 2 อย่างไร และจากอาเรย์ที่เราเก็บเส้นทางไว้จะได้ path[1][2] เท่ากับ 2 นั่นคือไม่ผ่านโหนดใด ๆ จึงเป็น 0->1->2 คือ เส้นทางจากโหนดเริ่มต้น 0 ไปยังโหนดสิ้นสุด 2

สามารถเขียนเป็นขั้นตอนวิธีการแสดงเส้นทางจากโหนด u ไป v ได้ดังนี้

```
1. if path[u][v]= ∞ then
2.     print []
3.     exit
4. end if
5. print path[u][v]
6.  while u != v
7.     u = path[u][v]
8.     print path[u][v]
9.  end while
```

ขั้นตอนวิธีที่ 3.5 ขั้นตอนวิธีการแสดงเส้นทาง

จากขั้นตอนวิธีที่ 3.5 ในบรรทัดที่ 1-5 จะเป็นการทำงานโดยถ้าค่าเป็นอินฟินิตี้จะไม่ทำการพิมพ์ค่าใด ๆ แต่ถ้ามีค่าเราจะพิมพ์ค่า[u][v] ในบรรทัดที่ 6-9 จะเป็นการไล่เส้นทางในอาเรย์ที่เราเก็บเส้นทางไว้โดยเส้นทางที่เราจะไปจะไล่ตั้งแต่โหนดเริ่มต้น (u) ไปยังโหนดสิ้นสุด (v)

3.2 การออกแบบขั้นตอนวิธีเพิ่มความสามารถในการหาเส้นทางได้อย่างเสถียร และขั้นตอนวิธีในการเรียกใช้เส้นทางสำรอง

ในหัวข้อนี้เราจะเลือกใช้ขั้นตอนวิธีของฟลอยด์มาเพิ่มความสามารถเพราะในเรื่องของการทำงานจริงฟรอยด์จะมีประสิทธิภาพมากกว่าการประยุกต์การคูณเมทริกซ์ จะประกอบไปด้วย 2 ส่วน ได้แก่

3.2.1 การเพิ่มความสามารถในการหาเส้นทางได้อย่างเสถียร

จากขั้นตอนวิธีแบบเดิมจะมีการเก็บข้อมูล เป็นอาเรย์ 2 มิติ 2 อาเรย์ซึ่งประกอบไปด้วยอาเรย์ที่เก็บค่าเส้นทางและอาเรย์ที่เก็บค่าระยะทาง แต่สำหรับในการเพิ่มความสามารถให้หาเส้นทางได้อย่างเสถียรจะต้องมีการขยายอาเรย์เพิ่มเป็นสามมิติกลายเป็น ขนาด n x n x n โดยข้อมูลที่เก็บเพิ่มคือข้อมูลของตอนที่ขั้นตอนวิธีของเราผ่านค่า k จากที่เราเคยทิ้งเราจะมีการเก็บค่า k เอาไว้

$k=0$

i \ j	1	2	3	4
1	0	∞	−2	∞
2	4	0	3	∞
3	∞	∞	0	2
4	∞	−1	∞	0

$k=1$

i \ j	1	2	3	4
1	0	∞	−2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	∞	−1	∞	0

$k=2$

i \ j	1	2	3	4
1	0	∞	−2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	3	−1	1	0

$k=3$

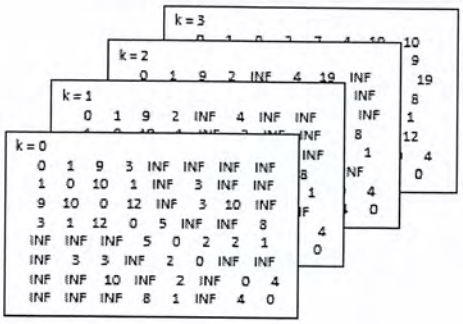
i \ j	1	2	3	4
1	0	∞	−2	0
2	4	0	2	4
3	∞	∞	0	2
4	3	−1	1	0

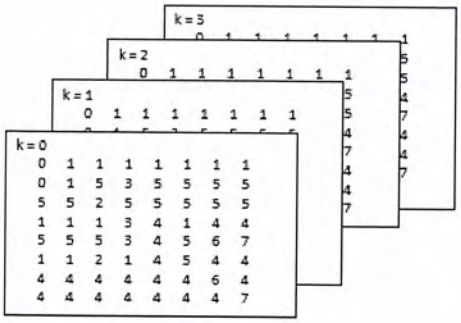
$k=4$

i \ j	1	2	3	4
1	0	−1	−2	0
2	4	0	2	4
3	5	1	0	2
4	3	−1	1	0

รูปที่ 3.5 แสดงการคำนวณของขั้นตอนวิธีแบบฟลอยด์โดยผ่านทีละโหนด

จากรูปที่ 3.5 โดยปกติข้อมูลของเราจะมีถึงแค่ k=4 และทิ้งข้อมูลตัวอื่น ๆ ดังนั้นแทนที่จะทิ้งข้อมูล k=1 ถึง k=3 จะมีการเก็บเอาไว้เพื่อนำมาใช้ในการทำการหาเส้นทางได้อย่างเสถียร





รูปที่ 3.6 แสดงการเก็บข้อมูลของขั้นตอนวิธีฟลอยด์แบบเสถียร

จากรูปที่ 3.6 แสดงอาเรย์ชื่อ alldistance เป็นอาเรย์ที่เก็บค่าเส้นทางและอาเรย์ที่ชื่อ allpath เป็นอาเรย์ที่เก็บเส้นทาง และจากวิธีเดิมกับแนวคิดใหม่ทำให้เราสามารถแสดงขั้นตอนวิธีได้ดังนี้

```
APSP using Floyd with keep path and reliable
1.let all_distance be an array of size n x n x n
2.let all_path be an array of size n x n x n
3.for i=0 to n-1 do
4.  for j=0 to n-1 do
5.    all_path[0][i][j] = j
6.get all_distance[0]
7.  for k=0 to n-1 do
8.    for i=0 to n-1 do
9.      for j=0 to n-1 do
10.       new_weight =  all_distance[k][i][k] + all_distance[k][k][j]
11.           if new_weight < all_distance[k][i][j] then
12.             all_distance[k][i][j] = new_weight
13.             all_path[k][i][j] = all_path[k][i][k]
14.           end if
15.     end for
16.   end for
17.   if k+1 < n then
18.     array_copy(all_distance[k], all_distance[k + 1]);
19.     array_copy(all_path[k], all_path[k + 1])
20.   end if
21. end for
```

ความซับซ้อนของเวลา : $O(n^3)$

ขั้นตอนวิธีที่ 3.6 ขั้นตอนวิธีของฟลอยด์แบบเก็บเส้นทางและเสถียร (Floyd with keep path and reliable)

จากขั้นตอนวิธีที่ 3.6 จะเป็นการพัฒนาขึ้นมาเพื่อรองรับการทำให้ขั้นตอนวิธีมีความเสถียรมากยิ่งขึ้น โดยจากอาเรย์สองมิติเราจะมีการเก็บเพิ่มเป็นสามมิติโดยเพิ่มมิติที่ k เข้าไป

3.2.2 การเรียกใช้เส้นทางสำรอง

สมมติเรามีการเก็บเส้นทางเรียบร้อยแล้วในข้อ 1) เราจะสมมุติให้เส้นทาง 4->5 เกิดความความเสียหายหรือเกิดเส้นทางที่ 4 ไป 5 พัง จะให้ alldistance[0][4][5] = infinity และหากพิจารณาอาเรย์มิติที่ k=4 ซึ่งเป็นมิติที่เก็บค่าโหนด 4 ไปยังโหนด 5 ดังนั้นเราจะเริ่มคิดที่อาเรย์มิติที่ k=3 เพราะยังไม่มีการนำมาผ่านโหนด 4 จากรูปที่ 3.8 จะเห็นว่าเส้นทาง 4->5 มีค่าระยะทางเป็น 2 แต่เส้นทางนี้เกิดความเสียหายทำให้จะไม่ใช้ค่าดังกล่าวและจะมีการหาค่าระยะทางที่สั้นที่สุดใหม่โดยคิดแค่ 1 เส้นเท่านั้น (ใช้ O(n)) โดยใช้วิธีการคำนวณใหม่โดยคิดเพียงเส้นเดียวดังรูปที่ 3.9

สมมติ ถ้าที่ k=3 ที่ 4->5 มีการคำนวณระยะทางใหม่จาก 2 เป็น 7 (เลขสมมติ) จะมีการแก้ไขไปถึงที่ k=4 โดยการคัดลอกคอลัมที่ 5 ของ k=3 ไปที่ k=4 ดังรูปที่ 3.10 และจาก k=4 จะมีการนำไปคำนวณใหม่ต่อไปตามขั้นตอนวิธีของฟลอยด์จนครบรอบ n - 1

k = 4

0	1	9	2	7	4	9	8
1	0	10	1	6	3	8	7
9	10	0	11	16	3	10	17
2	1	11	0	5	4	7	6
7	6	16	5	0	2	2	1
4	3	3	4	2	0	4	3
9	8	10	7	2	4	0	3
8	7	17	6	1	3	3	0

รูปที่ 3.7 แสดงการเก็บข้อมูลของขั้นตอนวิธีฟลอยด์แบบเสถียร

k = 3

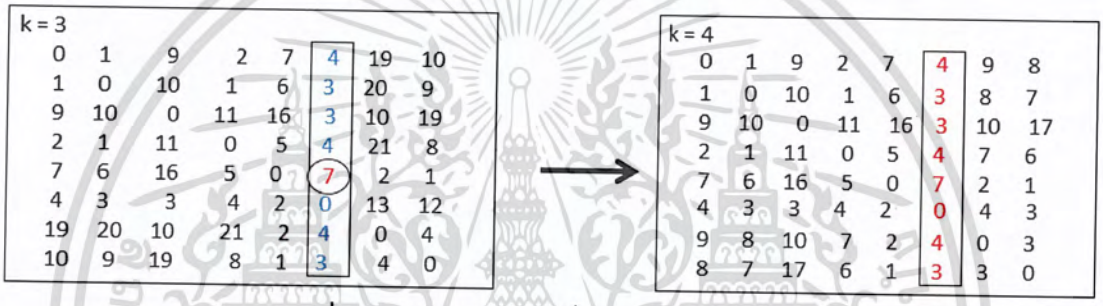
0	1	9	2	7	4	19	10
1	0	10	1	6	3	20	9
9	10	0	11	16	3	10	19
2	1	11	0	5	4	21	8
7	6	16	5	0	2	2	1
4	3	3	4	2	0	13	12
19	20	10	21	2	4	0	4
10	9	19	8	1	3	4	0

รูปที่ 3.8 แสดงการเก็บข้อมูลของขั้นตอนวิธีฟลอยด์แบบเสถียร

```
for (int k = 0; k < u; k++)
{
   int new_weight = alldistance[k][u][k] + alldistance[k][k][v];

   if (new_weight < alldistance[k][u][v])
   {
      alldistance[k][u][v] = new_weight;
      allpath[k][u][v] = allpath[k][u][k];
   }
   alldistance[k+1][u][v] = alldistance[k][u][v];
   allpath[k+1][u][v] = allpath[k][u][v];
}
```

รูปที่ 3.9 วิธีการคำนวณใหม่โดยคิดเพียงเส้นเดียว



รูปที่ 3.10 คัดลอกคอลัมที่ 5 ของ k=3 ไปที่ k=4

กรณีที่จะต้องซ่อมเส้นทาง

1) ถ้าโหนดต้นทางใดหรือโหนดปลายทางใดเกิดความเสียหายเราจะไปดูที่อาเรย์เส้นทางที่คำนวณเสร็จสิ้นแล้วว่ามีค่าเป็นโหนดปลายทางหรือไม่ ถ้าใช่จะต้องทำการคำนวณใหม่มิติที่ k=โหนดต้นทาง - 1 ถ้าไม่จะทำแค่เพียงกำหนดค่าที่ alldistance[0][โหนดต้นทาง][โหนดปลายทาง] = infinity
2) ถ้าโหนดต้นทางเป็นโหนดเริ่มต้น 0 ซึ่งเป็นขั้นแรกของการคำนวณหากโหนดต้นทาง ไปถึงโหนดปลายทางใด ๆ เกิดความเสียหายในกรณีนี้จะต้องคำนวณใหม่ทั้งหมด
3) กรณีที่เกิดความเสียหายจากโหนดสุดท้ายไปหาโหนดอื่น สมมติว่า 8 เป็นโหนดสุดท้ายที่เป็นทางผ่าน เช่น 8->5 , 8->7 เกิดความเสียหายจะได้ Best-Case

ความซับซ้อนของเวลา : Best-Case : $O(n^2)$

Worst-Case : $O(n^3)$

3.3 การออกแบบขั้นตอนวิธีเพิ่มประสิทธิภาพด้านความเร็วให้กับขั้นตอนวิธีที่สามารถเก็บเส้นทาง โดยปรับให้เป็นขั้นตอนวิธีแบบขนาน

ในขั้นตอนแบบขนานเราจะต้องมีการแบ่งงานเพื่อทำให้เป็นการทำงานแบบขนาน โดยจะได้จากสูตรดังนี้

สูตรคำนวณการแบ่งงาน คือ $b = w / P$ ——————— (3.1)

เมื่อ b คือ งานที่แต่ละโปรเซสเซอร์ต้องรับไปทำเท่า ๆกัน w คือจำนวนงานที่ทั้งหมด P คือ จำนวนโปรเซสเซอร์ที่มีในการทำงาน

X_{01}	x_{02}	...	X_{0j}
x_{11}	X_{12}	...	X_{1j}
X_{21}	X_{22}	...	X_{2j}
X_{31}	X_{32}	...	X_{3j}
X_{41}	X_{42}	...	X_{4j}
X_{51}	X_{52}	...	X_{5j}

รูปที่ 3.11 แสดงงานที่กำหนดขึ้นมาจำนวน 6 แถว

จากรูปที่ 3.11 เรากำหนดงานขึ้นมา 6 แถว และหากมีโปรเซสเซอร์ 3 ตัวเราจะได้ว่าแต่ละโปรเซสเซอร์จะต้องรับงานไปโดยคิดจากสูตร b=w/P หากแทนค่าจะได้ b=6/3= 2 ดังนั้น แต่ละโปรเซสเซอร์จะต้องรับงานไปคนละ 2 แถว เป็นต้น

สำหรับการคำนวณว่าจะให้งานไปที่โปรเซสเซอร์ใด ๆ อย่างละเอียดจะแสดงในรูปที่ 3.12 ซึ่งจากงาน W แถวจะถูกแบ่งเป็น b โดยที่โปรเซสเซอร์ 0 จะรับหน้าที่ตั้งแต่แถว 0 ถึงแถว b-1 โปรเซสเซอร์ 1 จะรับหน้าที่ตั้งแต่แถว b ถึงแถว b + b - 1 และ โปรเซสเซอร์ k ใด ๆจะรับหน้าที่ตั้งแต่แถวที่ kb ถึงแถว kb + b - 1 ซึ่งสามารเขียนการแบ่งงานที่โปรเซสเซอร์ k ใด ๆ ได้ดังนี้

แถวที่เริ่มทำงาน = (P * b)

แถวที่สุดท้ายที่ทำงาน = (P * b) + b - 1

เมื่อ P คือ หมายเลขโปรเซสเซอร์ตั้งแต่ 0 ถึง k

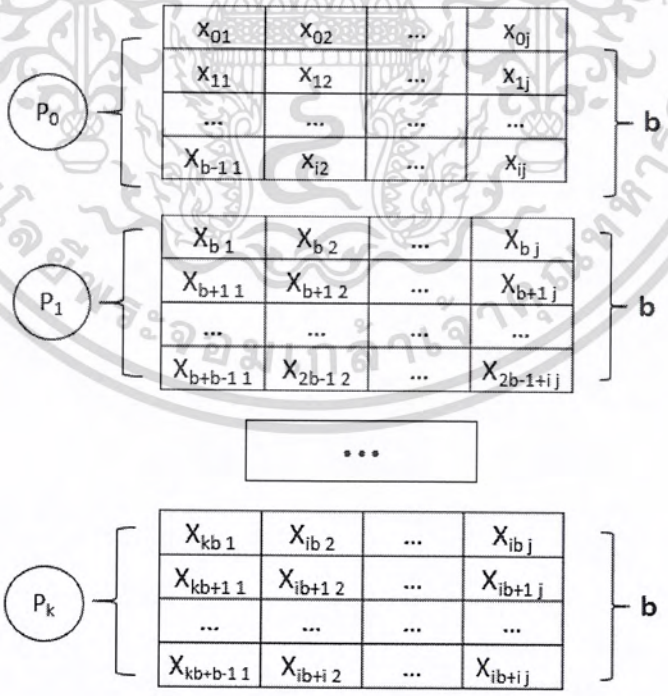
b คือ จำนวนงานที่ต้องทำที่ได้จาก สูตรที่ 3.1

นอกจากนี้จะเกิดปัญหาถ้าหากแบ่งงานแล้วไม่ลงตัวจะทำให้เราไม่รู้ว่างานที่เหลือควรให้ใครซึ่งงานที่เหลือสามารถคำนวณได้จากสูตรดังนี้

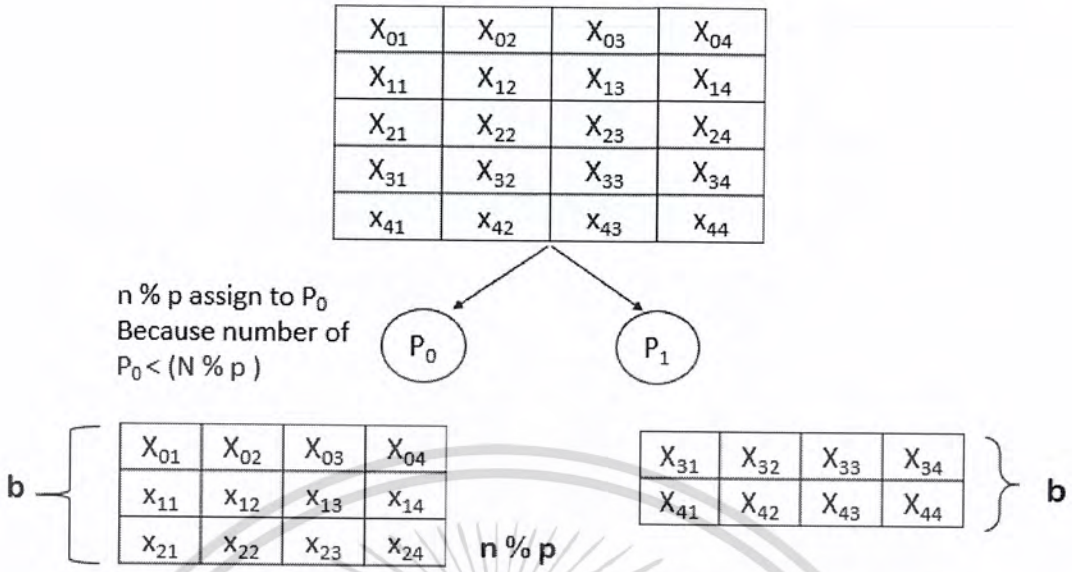
สูตรคำนวณงานที่เหลือ คือ $r = w \% P$ ——————— (3.2)

เมื่อ r คืองานที่เหลือ w คือจำนวนงานทั้งหมด P คือ จำนวนโปรเซสเซอร์ที่มีในการทำงาน

สำหรับในงานวิจัยนี้จะแบ่งให้โปรเซสเซอร์ที่หมายเลขน้อยกว่า r เป็นโปรเซสเซอร์ที่รับงานที่เหลือดังกล่าวไปโดยจะกระจายงานให้เท่า ๆกัน ดังรูปที่ 3.13



รูปที่ 3.12 แสดงการแบ่งงานที่โปรเซสเซอร์ใด ๆ



รูปที่ 3.13 แสดงการแบ่งงานที่โปรเซสเซอร์ใด ๆ ในกรณีที่มีงานเหลืออยู่

จากแนวคิดที่อธิบายไปในข้างต้นทำให้เราสามารถแบ่งงานและแก้ไขปัญหางานที่จะเกิดขึ้นได้และเป็นแนวคิดของการทำงานแบบขนาน ดังนั้นเราจะประยุกต์ขั้นตอนวิธีแบบขนานแบบนี้กับขั้นตอนวิธีการหาระยะทางที่สั้นที่สุดเพื่อเพิ่มประสิทธิภาพด้านความเร็ว โดยจะแบ่งเป็น 2 ขั้นตอนตามขั้นตามวิธีแบบเดิมที่โดยเราจะนำมาปรับปรุง ดังนี้

3.3.1 การประยุกต์ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีการหาระยะทางที่สั้นที่สุดด้วยวิธีการประยุกต์การคูณเมทริกซ์แบบเก็บเส้นทางได้

```
APSP using Matrix Multiplication  by parallel algorithms
1. for process=0 to number_of_processes-1 pardo
2.     let distance be an array of size n x n
3.     let part_of_distance, part_of_path be an array of size n x n
4.     If process=MASTER then
5.        set distance value
6.     end if
7.     for i=0 to row_per_process-1 do
8.        for j=0 to n-1
9.           part_of_path[i][j] = j
10.       end for
11.    end for
12.    broadcast (distance) from master to another process
13.    part_of_distance = partition(distance)
14.    array_transpose (distance)
15.    for r=0 to log(n)-1 do
16.       for i=0 to row_per_process-1 do
17.          for j=0 to n-1 do
18.             for k=0 to n-1 do
```

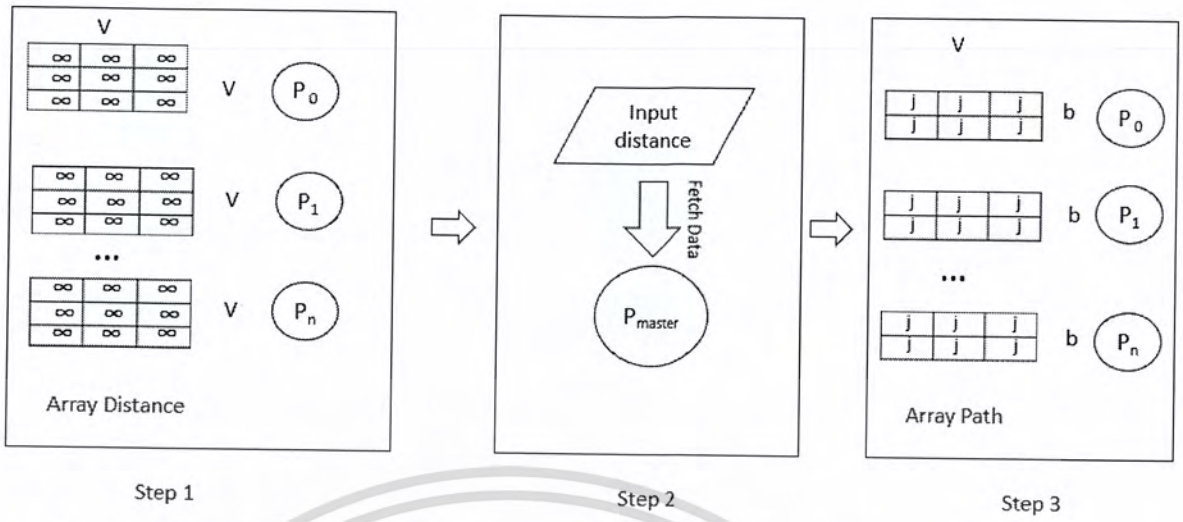
```
APSP using Matrix Multiplication  by parallel algorithms
19.              if part_of_distance[i][k] + distance[j][k] < part_of_distance[i][j] then
20.                part_of_distance[i][j] = part_of_distance[i][k] + distance[j][k]
21.                part_of_path[i][j] = path_of_path[i][k]
22.              end if
23.            end for
24.          end for
25.        end for
26.        if process != MASTER then
27.          send (part_of_distance) to MASTER
28.        else
29.          distance = receive(part_of_distance)
30.          array_transpose(distance)
31.        end if
32.        broadcast (distance) from master to another process
33.      end for
34. end for
```

ความซับซ้อนของเวลา $O(n^2 \log(n))$ ขณะที่ $P = n$

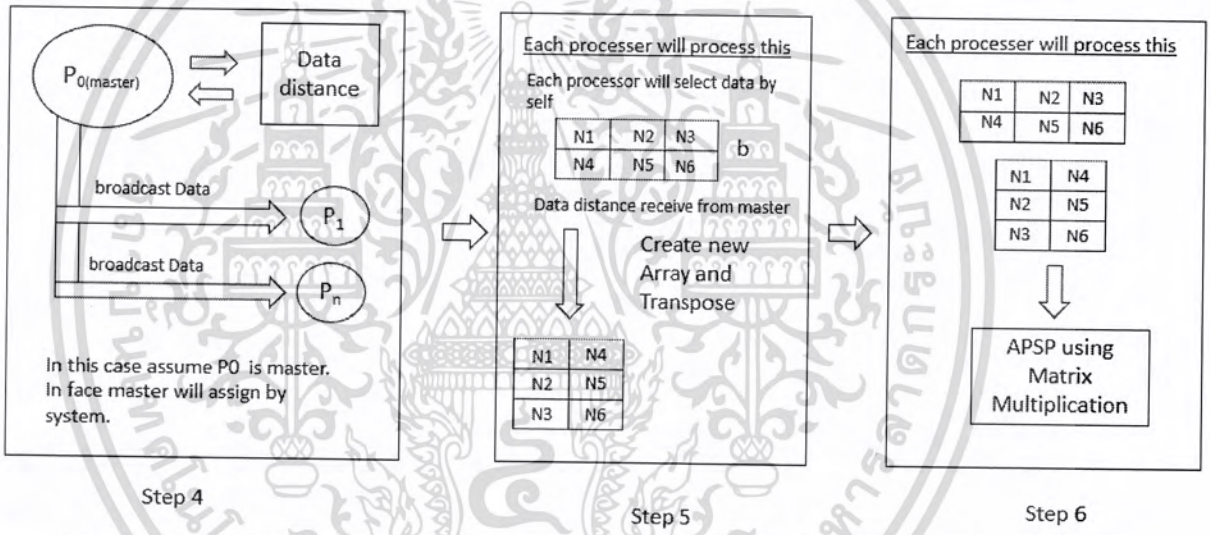
เวลาในการติดต่อสื่อสาร $p + p \log(n)$ ครั้ง

ขั้นตอนวิธีที่ 3.7 การประยุกต์ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีการประยุกต์การคูณเมทริกซ์ (Matrix Multiplication by parallel algorithms)

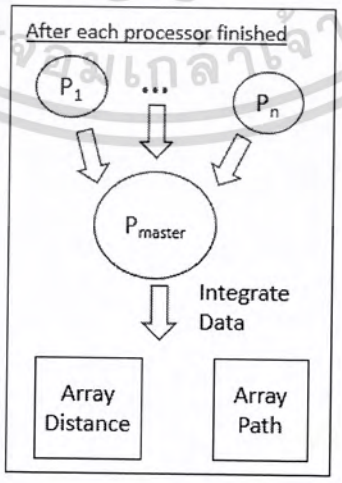
จากขั้นตอนวิธีการประยุกต์การคูณเมทริกซ์กับขั้นตอนวิธีแบบขนานเราจะมีการแบ่งงานเพื่อให้แต่ละโปรเซสเซอร์ไปทำงาน โดยจากโค้ดเทียมทุกโปรเซสเซอร์จะเข้ามาอ่านโค้ดเดียวกัน ตั้งแต่ บรรทัดที่ 2 - 6 จะเป็นการทำการสร้างอาเรย์ขนาด n X n เมื่อ n คือจำนวนโหนด และให้ค่าเริ่มต้น เป็นค่าอินฟินิตี้ หลังจากนั้นถ้าโปรเซสเซอร์ที่มาอ่านเป็น Master โดยที่ Master จะทำหน้าที่รับ input หรือและกระจายให้โปรเซสเซอร์ลูก บรรทัดที่ 7-11 เป็นขั้นตอนการตั้งค่าเริ่มต้นให้กับงานเส้นทางในแถวที่ตัวเองรับมา โดยจะเซตเป็นค่า j ซึ่งเป็นค่าเส้นทาง เรียกอีกอย่างว่าเป็นการทำการสร้างตัวเก็บเส้นทาง บรรทัดที่ 12-13 เป็นการส่งค่าและรับค่าของโปรเซสเซอร์ Master กับ ลูก และโปรเซสเซอร์จะทำการเลือกแถวที่ตัวเองต้องรับผิดชอบ บรรทัด 14-25 เป็นขั้นตอนวิธีการหาระยะทางที่สั้นที่สุดดังที่เคยกล่าวไปแล้ว แต่บรรทัดที่ 13 จะมีการทำการ transpose เพื่อทำให้โปรแกรมทำงานได้รวดเร็วโดยการแก้ปัญหาการเข้าถึงอาเรย์แบบ Row Major บรรทัดที่ 26 -32 จะให้โปรเซสเซอร์ตัวอื่น ๆ ที่ไม่ใช่มาสเตอร์ส่งข้อมูลที่คิดแล้วให้มาสเตอร์ บรรทัดที่ 32 จะเป็นการกระจายข้อมูลที่ทำการคำนวณแล้วจาก Master แล้ว ให้กับ โปรเซสเซอร์ตัวอื่น ๆ เพื่อโปรเซสเซอร์นำไปคำนวณรอบถัดไปนอกจากนี้ยังมีการ transpose ก่อนจะไปเข้าการคำนวณรอบถัดไป จนจบ log(n) รอบ เมื่อ n คือจำนวนโหนด หลังจากนั้นเวลานำข้อมูลที่ โปรเซสเซอร์ลูกคำนวณไปใช้งาน Master จะมีขั้นตอนวิธีการรวมข้อมูลที่เหมาะสมเพื่อนำไปใช้งานต่อไป



รูปที่ 3.14 แสดงการทำงานของ APSP using Matrix Multiplication by parallel algorithm



รูปที่ 3.15 แสดงการทำงานของ APSP using Matrix Multiplication by parallel algorithm



รูปที่ 3.16 แสดงการทำงานของ APSP using Matrix Multiplication by parallel algorithm

3.3.2 การประยุกต์ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีการหาเส้นทางที่สั้นที่สุดของฟลอยด์

```
APSP using Floyd by parallel algorithms
1. for process=0 to number_of_processes-1 pardo
2.     let part_of_distance, part_of_path be an array of size (row_per_process+1) x n
3.     If process=MASTER then
4.        let distance be an array of size n x n
5.        set distance value
6.        part_of_distance =  partition distance for MASTER
7.        partition data and send to another process
8.     else
9.        part_of_distance =  receive part_of_distance from MASTER
10.    for i=0 to row_per_process-1 do
11.       for j=0 to n-1
12.          part_of_path[i][j] = j
13.       end for
14.    end for
15.    for k=0 to  n-1 do
16.       for i=1 to row_per_process-1 do
17.          for j=0 to n-1 do
18.             if part_of_distance[i][j] >  part_of_distance[i][k] + part_of_distance[0][j]
then
19.                part_of_distance[i][j] =  part_of_distance[i][k] + part_of_distance[0][j]
20.                part_of_path[i][j] = j
21.             end if
22.          end for
23.       end for
24.       if process has row k then
25.          send data to another process
26.          part_of_distance[0] =  part_of_distance[index of row k]
27.       else
28.          part_of_distance[0] =  receive data from process that has row k
29.    end for
30. end for
```

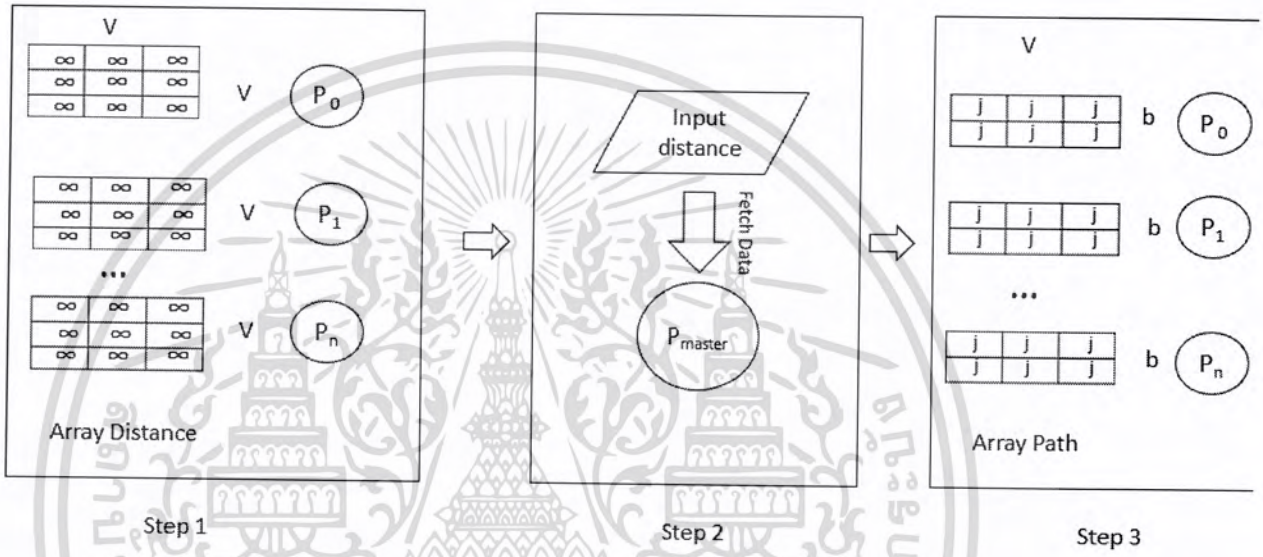
ความซับซ้อนของเวลา $O(n^2)$ ขณะที่ P = n

เวลาในการติดต่อสื่อสาร p + np ครั้ง

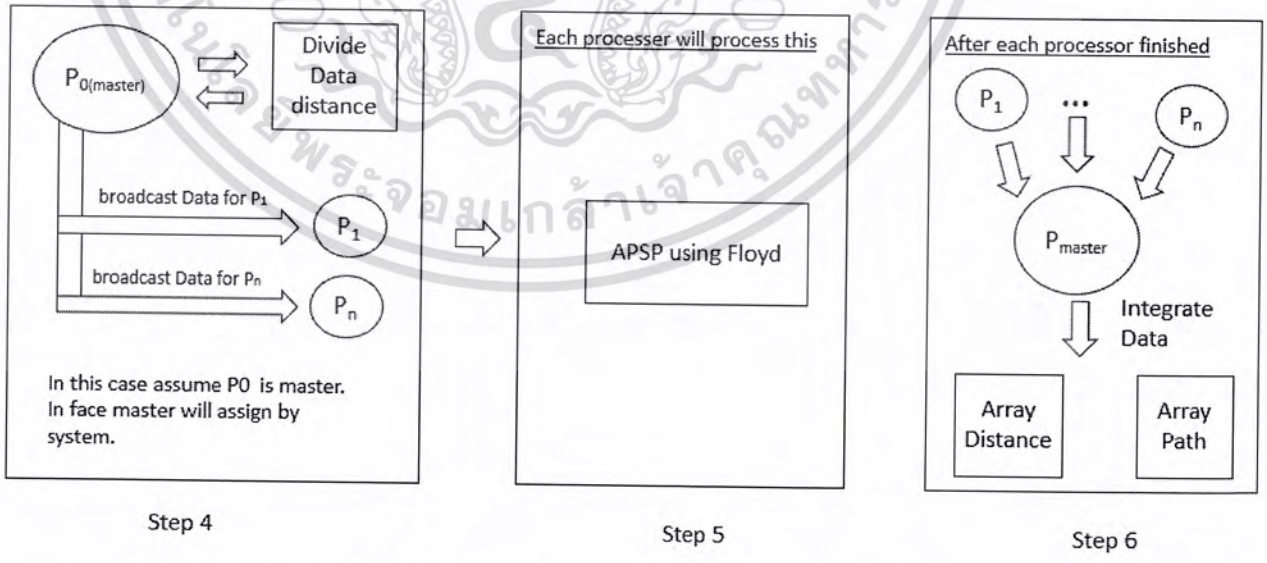
ขั้นตอนวิธีที่ 3.8 การประยุกต์ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีของฟลอยด์ (Floyd by parallel algorithm)

จากขั้นตอนวิธีการประยุกต์วิธีการหาเส้นทางที่สั้นที่สุดของฟรอยด์กับขั้นตอนวิธีแบบขนานเราจะมีการแบ่งงานเพื่อให้แต่ละโปรเซสเซอร์ไปทำงาน โดยจากโค้ดเทียมทุกโปรเซสเซอร์จะเข้ามาอ่านโค้ดเดียวกัน ตั้งแต่ บรรทัดที่ 2 - 9 จะเป็นการทำการสร้างอาเรย์ขนาด (row_per_process +1) x n เมื่อ n คือจำนวนโหนด และให้ค่าเริ่มต้น เป็นค่าอินฟินิตี้ หลังจากนั้นถ้าโปรเซสเซอร์ที่มาอ่านเป็น Master จะทำหน้าที่รับ รับข้อมูลระยะทางเข้ามาและ

แบ่งให้โปรเซสเซอร์อื่น ๆ บรรทัดที่ 10-14 เป็นขั้นตอนการทำการตั้งค่าเริ่มต้นให้กับงานเส้นทางในแถวที่ตัวเองรับมา โดยจะเซตเป็นค่า j ซึ่งเป็นค่าเส้นทาง เรียกอีกอย่างว่าเป็นการทำการสร้างตัวเก็บเส้นทางเอาไว้ บรรทัด 15-23 เป็นขั้นตอนวิธีการหาระยะทางที่สั้นที่สุดดังที่เคยกล่าวไปแล้วของฟลอยด์บรรทัดที่ 24-28 จะให้โปรเซสเซอร์ที่จะต้องส่งข้อมูลให้กับคนอื่นส่งข้อมูลให้กับโปรเซสเซอร์อื่น ๆ และสุดท้ายโปรเซสเซอร์ Master จะทำการรวมข้อมูลไว้เพื่อนำมาใช้งานต่อไป



รูปที่ 3.17 แสดงการทำงานของ APSP using Floyd by parallel algorithms



รูปที่ 3.18 แสดงการทำงานของ APSP using Floyd by parallel algorithms

3.4 การออกแบบขั้นตอนวิธีเพิ่มประสิทธิภาพด้านความเร็วให้กับขั้นตอนวิธีที่สามารถเก็บเส้นทางและเสถียร โดยปรับให้เป็นขั้นตอนวิธีแบบขนาน โดยจะประกอบไปด้วย 2 ส่วน คือ

3.4.1 การเก็บเส้นทาง สำหรับขั้นตอนนี้จะคล้ายๆกับขั้นตอนวิธีการเก็บเส้นทางและเสถียรแต่จะมีการแบ่งแต่ละ โปรเซสเซอร์เป็น row_per_process x n x n มีการเปลี่ยนโครงสร้างอาเรย์โดยแยกเก็บในแต่ละโปรเซสเซอร์เป็น part_of_distance , part_of_path ซึ่งเก็บค่าระยะทางและค่าเส้นทางตามลำดับและยังมีการเพิ่ม อาเรย์เพื่อมารอเก็บโหนดที่เป็นทางผ่านว่า temp_distance เป็นอาเรย์ขนาด n

APSP using Floyd by parallel algorithms with reliable

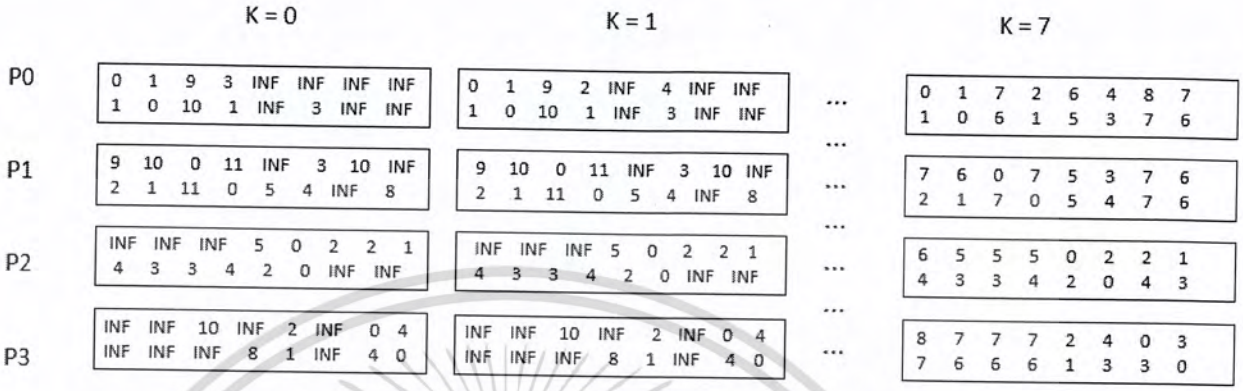
```
1. for process=0 to number_of_process-1 pardo
2.    let part_of_distance be a row_per_process x n x n array
3.    let part_of_path be a row_per_process x n x n array
4.    let temp_of_distance be a n array
5.     if process=MASTER then
6.       let distance be a n x n array
7.       set distance value
8.       part_of_distance[0] = partition distance for Master
9.       temp_of_distance = distance[0]
10.      partition distance and send to another process
11.    else
12.       part_of_distance[0] = receive data from master
13.       temp_of_distance = receive data from master
14.    for i=0 to row_per_process-1 do
15.       for j=0 to n-1 do
16.          part_of_path[0][i][j] = j
17.    for k=0 to n-1 do
18.      for i=0 to row_per_process-1 do
19.         for j=0 to n-1 do
20.            new_weight = part_of_distance[k][i][k] + temp_of_distance[j]
21.            if new_weight < part_of_distance[k][i][j] then
22.               part_of_distance[k][i][j] = new_weight
23.               part_of_path[k][i][j] = part_of_path[k][i][k]
24.            end if
25.         end for
26.      end for
27.      if k+1 < n then
28.         array_copy(part_of_distance[k], part_of_distance[k + 1]);
29.         array_copy(part_of_path[k], part_of_path[k + 1])
30.      end if
31.      if process have row k then
32.         send data to another process
33.         temp_of_distance = part_of_distance[k][index of row k]
34.      else
35.         temp_of_distance = receive data from process that have row k
36.    end for
37. end for
```

ความซับซ้อนของเวลา $O(n^2)$ ขณะที่ P = n

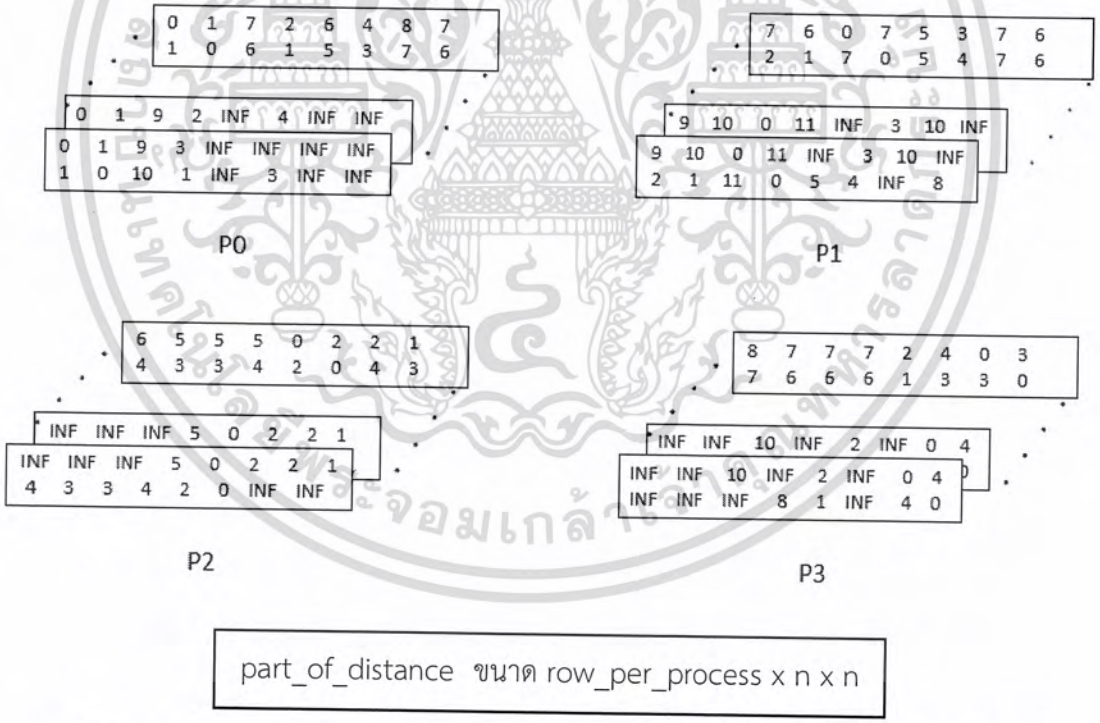
เวลาในการติดต่อสื่อสาร p + np ครั้ง

ขั้นตอนวิธีที่ 3.9 การประยุกต์ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีของฟลอยด์แบบเสถียร (Floyd by parallel algorithms)

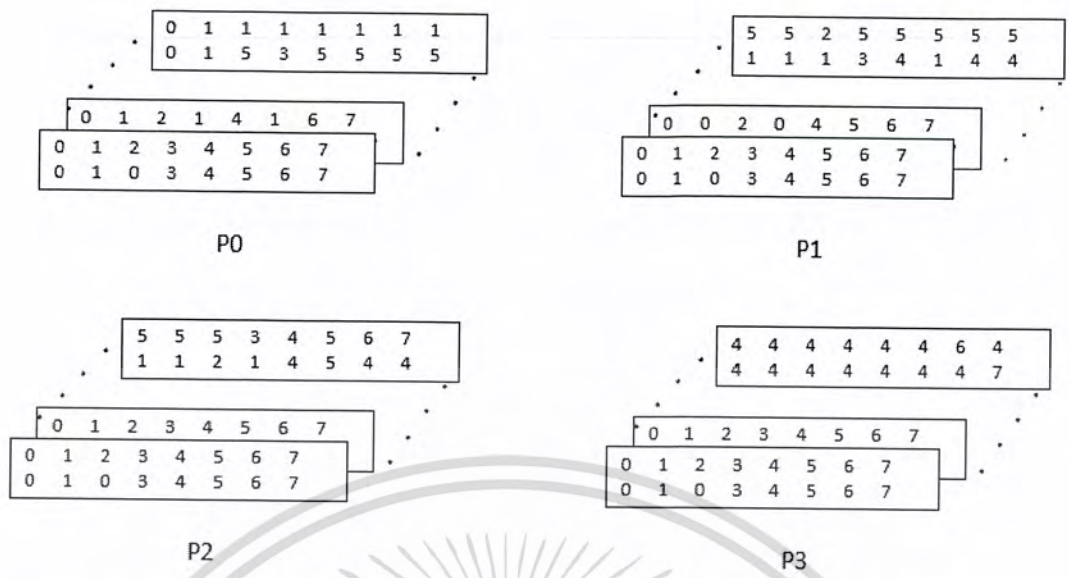
จากขั้นตอนที่ 3.9 หลังจากคำนวณเสร็จแต่ละโปรเซสเซอร์จะเก็บค่าที่สั้นที่สุดไว้ row_per_process แถวและเก็บ k แบบ ดังแสดงในรูปที่ 3.19 ดังนั้นในแต่ละโปรเซสเซอร์จะมีการเก็บข้อมูลเส้นทางและระยะทางดังในรูปที่ 3.20 – 3.21



รูปที่ 3.19 แสดงการเก็บข้อมูลของโปรเซสเซอร์ในขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีของฟลอยด์แบบเสถียร



รูปที่ 3.20 แสดงการเก็บข้อมูลระยะทางของโปรเซสเซอร์ในขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีของฟลอยด์แบบเสถียร ในรูปแบบแต่ละโปรเซสเซอร์



รูปที่ 3.21 แสดงการเก็บข้อมูลเส้นทางของโปรเซสเซอร์ในขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีของฟลอยด์แบบเสถียร ในรูปแบบแต่ละโปรเซสเซอร์

3.4.2 การใช้เส้นทางสำรอง สำหรับการใช้เส้นทางสำรองจะคล้าย ๆ กับฟลอยด์แบบเสถียรปกติ ถึงแม้ว่าหลังคำนวณข้อมูลจะเก็บอยู่ในแต่ละโปรเซสเซอร์ แต่จะไม่มีการรวมข้อมูลที่กระจายอยู่นี้ให้กับมาสเตอร์ จะให้มาสเตอร์ก็ต่อเมื่อมีการเรียกใช้การเรียกดูเส้นทาง สำหรับการซ่อมเส้นทางจะแสดงได้ดั้ง ขั้นตอนวิธีที่ 3.10

```
Function fix_path(u, v)
1.for process=0 to number_of_process-1 pardo
2.     if process have row u then
3.       part_of_distance[0][index of row u][v] ← infinity
4.       if part_of_path[n-1][index of row u][v] != v then
5.          call another process to end function
6.          return
7.       end if
8.       else
9.         waiting for process that has row u send status
10.      if u=0 then
11.         recalculate allpairshortestpath
12.         return
13.      end if
14.      for k=0 to u-1 do
15.         if process have a row k then
16.            send data to another process
17.            temp_of_distance =  part_of_distance[k][index of row k]
18.         else
19.            temp_of_distance =  receive data from process that has row k
```

```
Function fix_path(u, v)
20.       if process have row u then
21.         new_weight =   part_of_distance[k][index of row u][v] + temp_of_distance[k]
22.          if new_weight < part_of_distance[k][index of row u][v] then
23.            part_of_distance[k][index of row u][v] =  new_weight
24.            part_of_path[k][index of row u][v] = part_of_path[k][index of row u][v]
25.          end if
26.       if k+1 < n then
27.         part_of_distance[k+1][index of row u][v] =  part_of_distance[k][index of row
u][v]
28.         part_of_path[k+1][index of row u][v] =  part_of_path[k][index of row u][v]
29.       end if
30.     end for
31.     for i=0 to row_per_process-1 do
32.          part_of_distance[u][i][v] =  part_of_distance[u-1][i][v]
33.          part_of_path[u][i][v] = part_of_ path[u][i][v]
34.     end for
35.     for k=u to n-1 do
36.         if process have row k then
37.             send data to another process
38.           temp_of_distance = part_of_distance[k][index of row k]
39.         else
40.             temp_of_distance =  receive data from process that has row k
41.         for i=0 to row_per_process-1 do
42.           for j=0 to n-1 do
43.           new_weight = part_of_distance[k][i][k] + temp_of_distance[j]
44.             if new_weight < part_of_distance[k][i][j] then
45.               part_of_distance[k][i][j] =  new_weight
46.               part_of_path[k][i][j] = part_of_path[k][i][k]
47.             end if
48.           end for
49.       end for
50.       if k+1 < n then
51.           array_copy(part_of_distance[k], part_of_distance[k + 1]);
52.           array_copy(part_of_path[k], part_of_path[k + 1])
53.       end if
54.     end for
55.end for
```

ขั้นตอนวิธีที่ 3.10 แสดงขั้นตอนวิธีของการซ่อมเส้นทาง

จากขั้นตอนวิธที่ 3.10 ถ้าโหนดที่เสีย คือ u -> v เกิดความเสียหายขั้นตอนวิธีจะทำการหาว่าโปรเซสเซอร์ใดรับผิดชอบเส้นทางจาก u -> v พร้อมให้โปรเซสเซอร์นั้นทำการเปลี่ยนค่าระยะทางเป็นอินฟินิตี้และจากโปรเซสเซอร์ดังกล่าวจะเข้าไปดูข้อมูลที่ k=u-1 ที่คอลัม v เพื่อทำการคำนวณใหม่ตั้งแต่ตรงที่ดังกล่าว เพราะที่ u=k คือการที่โหนดต่าง ๆ ผ่านโหนด u มาแล้วดังนั้นเราจะย้อนไปที่ก่อนผ่านโหนด u และเราจำทำการคำนวณใหม่ที่ คอลัม v ตั้งแต่ โหนด k=0 ถึง k=u-1 โดยขณะที่คำนวณจะทำการเปลี่ยนค่าที่มิติ k หลังจากทำการคำนวณถึง k=u-1 แล้ว เมื่อถึง k=u จะทำการ

คัดลอกค่าคอลัมที่ v ของ k=u-1 ไปให้ที่ k=u และคำนวณใหม่โดยใช้ขั้นตอนวิธีของฟลอยด์แบบขนานแต่เริ่มที่ k=u ถึง k = n-1 และทุกโปรเซสเซอร์ทำงานพร้อมกัน

ตัวอย่าง เมื่อเส้นทางจากโหนด 4>5 เกิดความเสียหาย จะให้โปรเซสเซอร์ที่รับผิดชอบแถวนี้ ซึ่งก็คือโปรเซสเซอร์ 2 เปลี่ยนค่า 4>5 ที่ k=0 เป็นอินฟินิตี้ และโปรเซสเซอร์ 2 ย้อนกลับไปที่มิติอาเรย์ที่ k=u-1 ซึ่งก็คือ 3 ดังรูปที่ 3.22 ดังนั้นโปรเซสเซอร์ 2 จะเริ่มทำการคำนวณโดยย้อนกลับไปที่ k=0 และทำการคำนวณที่คอลัม v ซึ่งก็คือคอมลัมที่ 5 ใหม่จนถึง k=3 โดยในขณะคำนวณจะดึงค่าของคอลัมที่ k ของโปรเซสเซอร์ต่าง ๆ ผ่าน temp_of_distance ซึ่งเป็นอาเรย์ขนาด n เมื่อคำนวณเสร็จสิ้น (สมมติว่าได้ระยะทางใหม่คือ 7) จะทำการคัดลอก ค่าที่คอลัมที่ 5 ที่ k = 3 มาให้คอลัมที่ 5 ที่ k =4 ดังรูปที่ 3.23 และหลังจากนั้นเราจะเริ่มกระบวนการเส้นทางที่สั้นที่สุดใหม่โดยใช้ อาเรย์ที่ k=4 เป็นข้อมูลนำไปเข้าขั้นตอนวิธีแบบฟลอยด์เพื่อหาเส้นทางที่สั้นที่สุดใหม่

k = 3

P0	0	1	9	2	7	4	19	10
	1	0	10	1	6	3	20	9
P1	9	10	0	11	16	3	10	19
	2	1	11	0	5	4	21	8
P2	7	6	16	5	0	2	2	1
	4	3	3	4	2	0	13	12
P3	19	20	10	21	2	13	0	4
	10	9	19	8	1	12	4	0

รูปที่ 3.22 แสดงอาเรย์ระยะทางที่มิติ k=3

k = 3

0	1	9	2	7	7	19	10
1	0	10	1	6	3	20	9
9	10	0	11	16	3	10	19
2	1	11	0	5	4	21	8
7	6	16	5	0	7	2	1
4	3	3	4	2	0	13	12
19	20	10	21	2	13	0	4
10	9	19	8	1	12	4	0

Copy Colum 5 of k = 3

to Colum 5 of k = 4

k = 4

0	1	9	2	7	7	9	8
1	0	10	1	6	3	8	7
9	10	0	11	16	3	10	17
2	1	11	0	5	4	7	6
7	6	16	5	0	7	2	1
4	3	3	4	2	0	4	3
9	8	10	7	2	2	0	3
8	7	17	6	1	1	3	0

รูปที่ 3.23 แสดงการคัดลอกข้อมูลคอลัมที่ 5 ของ k=3 ไปให้ k=4

กรณีที่จะต้องซ่อมเส้นทาง

1) ถ้าโหนดต้นทางใดหรือโหนดปลายทางใดเกิดความเสียหายเราจะไปดูที่อาเรย์เส้นทางที่คำนวณเสร็จสิ้นแล้วว่ามีค่าเป็นโหนดปลายทางหรือไม่ ถ้าใช่จะต้องทำการคำนวณใหม่มิติที่ k=โหนดต้นทาง - 1 ถ้าไม่จะทำแค่เพียงกำหนดค่าที่

alldistance[0][โหนดต้นทาง][โหนดปลายทาง] = infinity

2) ถ้าโหนดต้นทางเป็นโหนดเริ่มต้น 0 ซึ่งเป็นขั้นแรกของการคำนวณหากโหนดต้นทาง ไปถึงโหนดปลายทางใด ๆ เกิดความเสียหายในกรณีนี้จะต้องคำนวณใหม่ทั้งหมด

3) กรณีที่เกิดความเสียหายจากโหนดสุดท้ายไปหาโหนดอื่น สมมติว่า 8 เป็นโหนดสุดท้ายที่เป็นทางผ่าน เช่น 8->5 , 8->7 เกิดความเสียหายจะได้ Best-Case

ความซับซ้อนของเวลา : Best-Case : $O(n)$ เมื่อ $p = n$

Worst-Case: $O(n^2)$ เมื่อ $p = n$

บทที่ 4
วิธีการทดลอง

4.1 การออกแบบการทดลอง

งานวิจัยนี้จะมีการสร้างโปรแกรมขึ้นมาจากขั้นตอนวิธีที่ออกแบบไว้ โดย โปรแกรมสำหรับการหาระยะทางที่สั้นที่สุดแบบอนุกรมจะเขียนโดยภาษา C ในขณะที่โปรแกรมสำหรับการหาระยะทางที่สั้นที่สุดแบบขนาน จะเขียนโดย ภาษา C และ มีการนำเอาไลบรารี MPI(message passing interface) มาใช้เพื่อช่วยให้สามารถเขียนโปรแกรมแบบขนานได้ โดยไลบรารี MPI นี้จะใช้ของบริษัทไมโครซอฟท์ ซึ่งมีชื่อว่า MS-MPI

4.1.1 สเปกเครื่องที่นำมาใช้รันโปรแกรม

OS : Windows 10 Education

RAM : 12 GB DDR4 (Bus Speed 2133)

CPU : Intel Core i7-6700HQ (4 physical cores and 8 logical cores

Clock speed 2.6 up to 3.5 GHz)

HDD : 1 TB (5400 rpm)

4.1.2 สมมุติฐาน

1) ในโปรแกรมแบบขนานหากมีการเพิ่มจำนวนโปรเซสเซอร์ขึ้นเวลาในการประมวลผลจะลดลง

2) จะมีจำนวนโปรเซสเซอร์ที่เหมาะสมในการใช้ประมวลผลโหนด

3) โปรแกรมแบบขนานจะใช้เวลาน้อยกว่าโปรแกรมแบบธรรมดา

4) ทั้งโปรแกรมแบบขนานและโปรแกรมแบบธรรมดาที่ใช้ขั้นตอนวิธีที่สร้างขึ้นในงานวิจัยนี้จะสามารถหาเส้นทางสำรองได้ทันทีคือไม่ต้องคำนวณอีกรอบเมื่อเส้นทางหลักเกิดความเสียหาย

5) ขั้นตอนวิธีที่คิดขึ้นจากงานวิจัยชิ้นนี้(ขั้นตอนวิธีแบบสเถียร) จะมีความคุ้มค่าในการเอาไปใช้มากกว่าอัลกริที่มการหาระยะทางที่สั้นที่สุดแบบเดิม(แบบไม่สเถียร)

4.1.3 วิธีการทดลอง

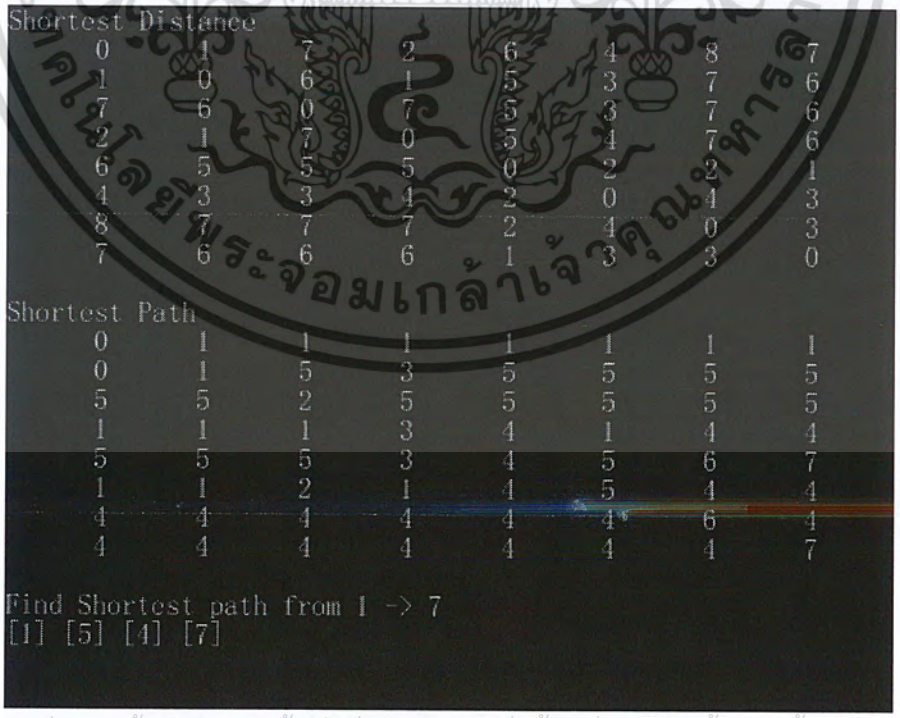
1.ทำการทดลองตามสมุติฐาน

2.เก็บและบันทึกผล

3.สรุปผลการทดลองตามสมุติฐาน

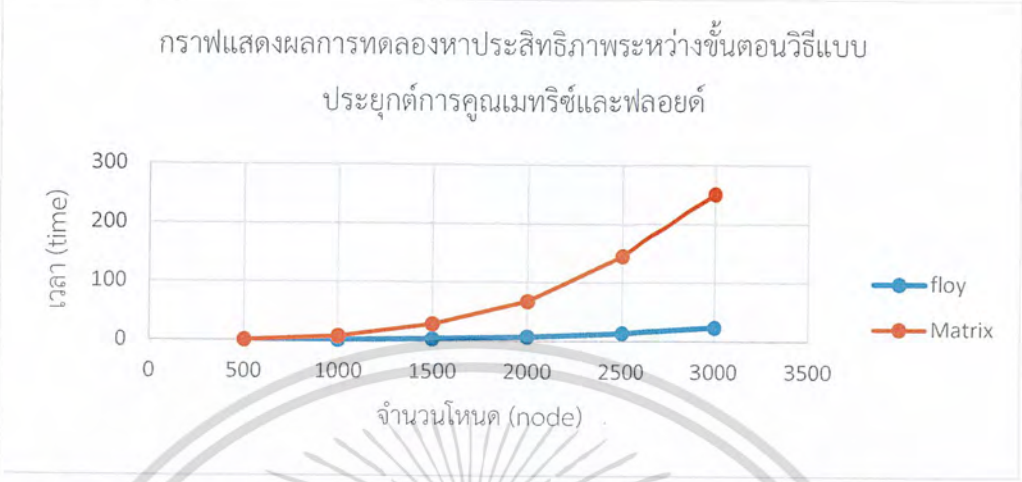
4.2 การพัฒนาขั้นตอนวิธีแบบเดิมให้เก็บเส้นทางได้

จากขั้นตอนวิธีที่พัฒนาขึ้นใน 3.1.1 จะทำให้ขั้นตอนวิธีของฟลอยด์และการประยุกต์เมทริกซ์เดิมสามารถแสดงเส้นทางได้โดยขั้นตอนวิธีทั้งจะใช้การแสดงเส้นทางรูปแบบเดียวกันดังรูปที่ 4.1 ซึ่งประกอบไปด้วย Shortest Distance เป็นเมทริกซ์ ที่แสดงข้อมูลระยะทางและ Shortest path เป็นเทริกซ์ที่แสดงข้อมูลเส้นทาง โดยการดูว่า จากโหนดเริ่มต้นไปที่โหนดปลายทางเราจะค่อยไล่ไปเรื่อย ๆ โดยดูจากเมทริซ์ Shortest Path เช่น จาก โหนด 1 ไป โหนด 7 เราจะไปดูที่โหน 1 ไป โหนด 7 ว่ามีค่าเท่าไหร่ ซึ่งในกรณีนี้ มีค่าเป็น 5 ต่อไป เราจะไปดูที่โหนด 5 ไป 7 ซึ่งมีค่าเป็น 4 และเราจะดู จาก 4 ไป 7 จะได้ เป็น 7 ดังนั้นแสดงว่าเราไล่เสร็จสิ้นเรียบร้อยและจะได้ระยะทางจากโหน 1 ไป 7 ต้องผ่านโหนด 5,4 และสามารถเขียนเป็นเส้นทางเต็ม ๆ ได้เป็น 1>5>4>7



รูปที่ 4.1 แสดงการเก็บเส้นทาง

4.3 ผลการทดลองหาประสิทธิภาพระหว่างขั้นตอนวิธีแบบประยุกต์การคูณเมทริซ์และฟลอยด์



รูปที่ 4.2 แสดงการเปรียบเทียบระหว่างขั้นตอนวิธีแบบประยุกต์การคูณเมทริกซ์และฟลอยด์

จากรูปที่ 4.2 จะแสดงให้เห็นว่าขั้นตอนวิธีแบบประยุกต์การคูณเมทริกซ์จะใช้เวลาในการทำงานมากกว่าฟลอยด์ ทำให้เราสามารถสรุปได้ว่าการคำนวณภายในอาเรย์ตัวมันเองจะใช้เวลาน้อยกว่าที่จะต้องไปใช้อาเรย์ของที่อื่นซึ่งจะเสียเวลามากกว่า

4.4 การพัฒนาขั้นตอนวิธีของฟลอยด์ให้มีการเก็บเส้นทางสำรองและใช้เส้นทางสำรอง

จาก 4.2 จะเห็นว่าฟลอยด์ใช้เวลาในการทำงานน้อยกว่าเมทริกซ์นั่นหมายถึงฟลอยด์มีประสิทธิภาพ มากกว่าของการประยุกต์การคูณเมริกซ์ จึงเลือกที่จะพัฒนาขั้นตอนวิธีของฟลอยด์มาพัฒนาให้เสถียรโดยสามารถซ่อมเส้นทางได้จากการพัฒนาเราจะเก็บอาเรย์เพิ่มขึ้นจากเดิมสองมิติเป็นสามมิติโดยมิติโดยที่เพิ่มมาเป็นการเก็บเส้นทางสำรองไว้ ตัวอย่างอาเรย์ของระยะทางและเส้นทางจะแสดงดังรูปที่ 4.3,4.4,4.5,4.6 ตามลำดับและตัวอย่างการเรียกใช้เส้นทางดังรูปที่ 4.7

k = 0

0	1	9	3	INF	INF	INF	INF
1	0	10	1	INF	3	INF	INF
9	10	0	12	INF	3	10	INF
3	1	12	0	5	INF	INF	8
INF	INF	INF	5	0	2	2	1
INF	3	3	INF	2	0	INF	INF
INF	INF	10	INF	2	INF	0	4
INF	INF	INF	8	1	INF	4	0

k = 1

0	1	9	2	INF	4	INF	INF
1	0	10	1	INF	3	INF	INF
9	10	0	11	INF	3	10	INF
2	1	11	0	5	4	INF	8
INF	INF	INF	5	0	2	2	1
4	3	3	4	2	0	INF	INF
INF	INF	10	INF	2	INF	0	4
INF	INF	INF	8	1	INF	4	0

k = 2

0	1	9	2	INF	4	19	INF
1	0	10	1	INF	3	20	INF
9	10	0	11	INF	3	10	INF
2	1	11	0	5	4	21	8
INF	INF	INF	5	0	2	2	1
4	3	3	4	2	0	13	INF
19	20	10	21	2	13	0	4
INF	INF	INF	8	1	INF	4	0

k = 3

0	1	9	2	7	4	19	10
1	0	10	1	6	3	20	9
9	10	0	11	16	3	10	19
2	1	11	0	5	4	21	8
7	6	16	5	0	2	2	1
4	3	3	4	2	0	13	12
19	20	10	21	2	13	0	4
10	9	19	8	1	12	4	0

รูปที่ 4.3 แสดงอาเรย์ของระยะทาง ที่ k=0 ถึง k=3

k = 4

0	1	9	2	7	4	9	8
1	0	10	1	6	3	8	7
9	10	0	11	16	3	10	17
2	1	11	0	5	4	7	6
7	6	16	5	0	2	2	1
4	3	3	4	2	0	4	3
9	8	10	7	2	4	0	3
8	7	17	6	1	3	3	0

k = 5

0	1	7	2	6	4	8	7
1	0	6	1	5	3	7	6
7	6	0	7	5	3	7	6
2	1	7	0	5	4	7	6
6	5	5	5	0	2	2	1
4	3	3	4	2	0	4	3
8	7	7	7	2	4	0	3
7	6	6	6	1	3	3	0

k = 6

0	1	7	2	6	4	8	7
1	0	6	1	5	3	7	6
7	6	0	7	5	3	7	6
2	1	7	0	5	4	7	6
6	5	5	5	0	2	2	1
4	3	3	4	2	0	4	3
8	7	7	7	2	4	0	3
7	6	6	6	1	3	3	0

k = 7

0	1	7	2	6	4	8	7
1	0	6	1	5	3	7	6
7	6	0	7	5	3	7	6
2	1	7	0	5	4	7	6
6	5	5	5	0	2	2	1
4	3	3	4	2	0	4	3
8	7	7	7	2	4	0	3
7	6	6	6	1	3	3	0

รูปที่ 4.4 แสดงอาเรย์ของระยะทาง ที่ k=4 ถึง k=7

k = 0

0	1	2	3	4	5	6	7
0	1	0	3	4	5	6	7
0	0	2	0	4	5	6	7
0	1	0	3	4	5	6	7
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

k = 1

0	1	2	1	4	1	6	7
0	1	0	3	4	5	6	7
0	0	2	0	4	5	6	7
1	1	1	3	4	1	6	7
0	1	2	3	4	5	6	7
1	1	2	1	4	5	6	7
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

k = 2

0	1	2	1	4	1	2	7
0	1	0	3	4	5	0	7
0	0	2	0	4	5	6	7
1	1	1	3	4	1	1	7
0	1	2	3	4	5	6	7
1	1	2	1	4	5	2	7
2	2	2	2	4	2	6	7
0	1	2	3	4	5	6	7

k = 3

0	1	2	1	1	1	2	1
0	1	0	3	3	5	0	3
0	0	2	0	0	5	6	0
1	1	1	3	4	1	1	7
3	3	3	3	4	5	6	7
1	1	2	1	4	5	2	1
2	2	2	2	4	2	6	7
3	3	3	3	4	3	6	7

รูปที่ 4.5 แสดงอาเรย์ของเส้นทาง ที่ k=0 ถึง k=3

k = 4

0	1	2	1	1	1	1	1
0	1	0	3	3	5	3	3
0	0	2	0	0	5	6	0
1	1	1	3	4	1	4	4
3	3	3	3	4	5	6	7
1	1	2	1	4	5	4	4
4	4	2	4	4	4	6	4
4	4	4	4	4	4	4	7

k = 5

0	1	1	1	1	1	1	1
0	1	5	3	5	5	5	5
5	5	2	5	5	5	5	5
1	1	1	3	4	1	4	4
5	5	5	3	4	5	6	7
1	1	2	1	4	5	4	4
4	4	4	4	4	4	6	4
4	4	4	4	4	4	4	7

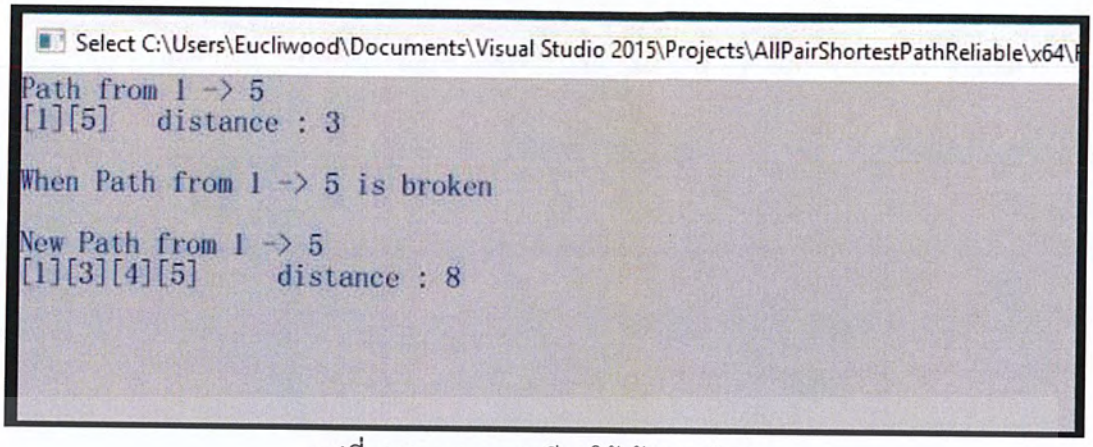
k = 6

0	1	1	1	1	1	1	1
0	1	5	3	5	5	5	5
5	5	2	5	5	5	5	5
1	1	1	3	4	1	4	4
5	5	5	3	4	5	6	7
1	1	2	1	4	5	4	4
4	4	4	4	4	4	6	4
4	4	4	4	4	4	4	7

k = 7

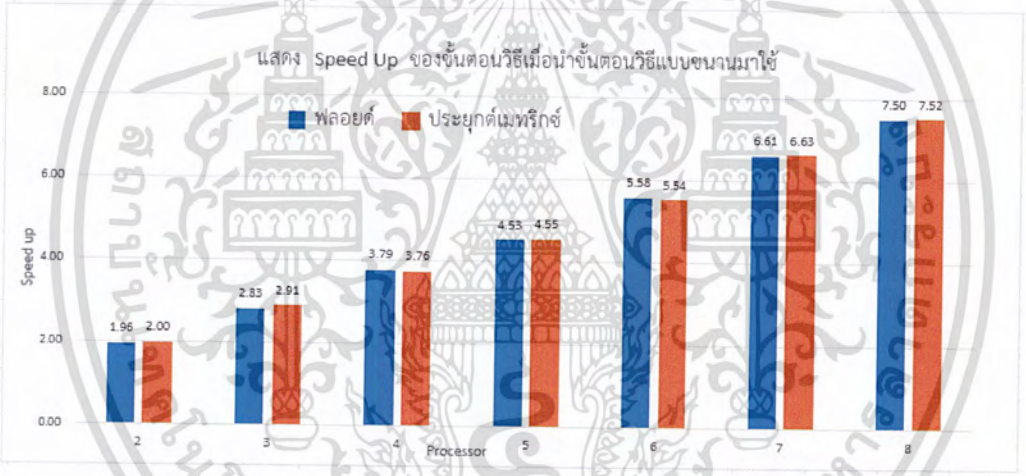
0	1	1	1	1	1	1	1
0	1	5	3	5	5	5	5
5	5	2	5	5	5	5	5
1	1	1	3	4	1	4	4
5	5	5	3	4	5	6	7
1	1	2	1	4	5	4	4
4	4	4	4	4	4	6	4
4	4	4	4	4	4	4	7

รูปที่ 4.6 แสดงอาเรย์ของเส้นทาง ที่ k=4 ถึง k=7

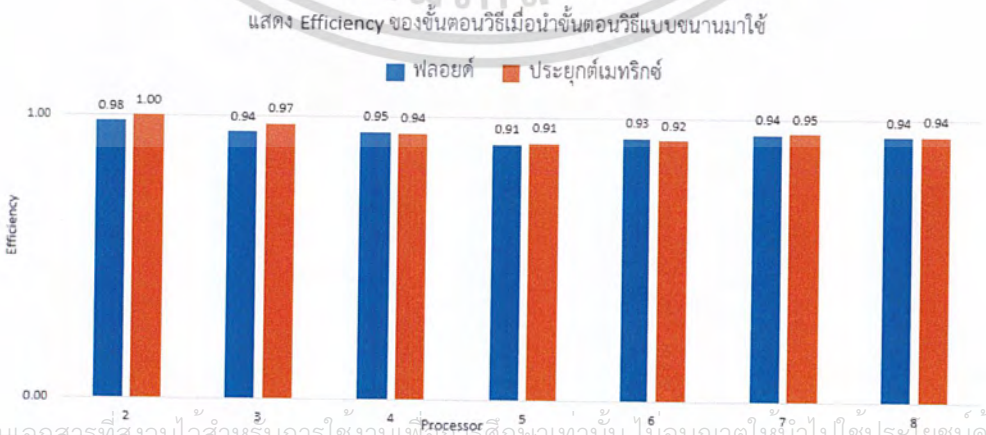


รูปที่ 4.7 แสดงการเรียกใช้เส้นทางสำรอง

4.5 การเปรียบเทียบประสิทธิภาพระหว่างการประยุกต์การคูณเมทริกซ์และฟลอยด์โดยพัฒนากับขั้นตอนวิธีแบบขนานโดยทดลองเพิ่มจำนวนโปรเซสเซอร์



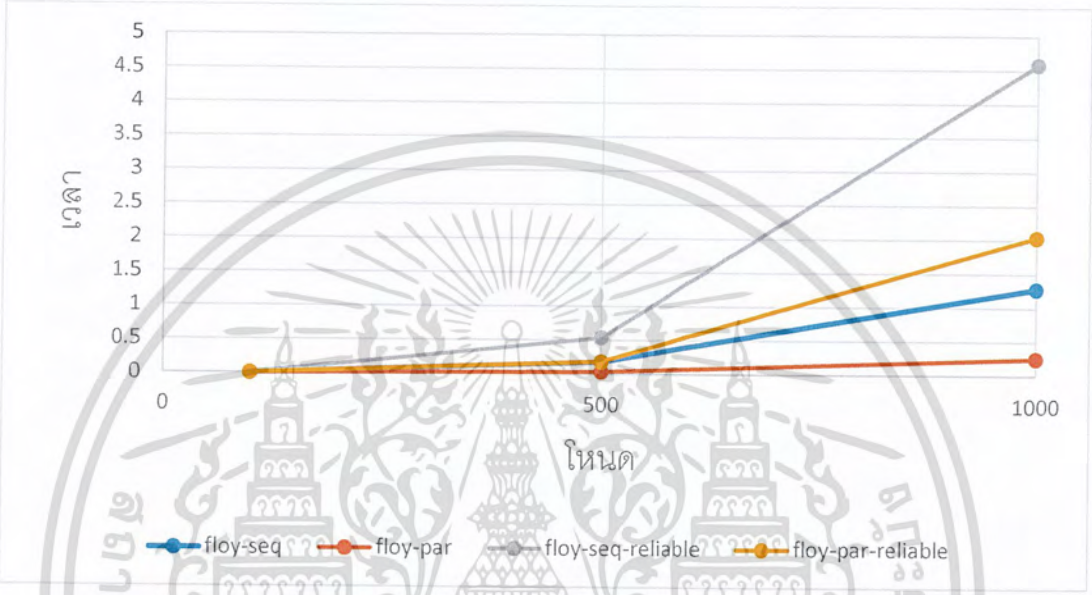
รูปที่ 4.8 แสดง speed up เมื่อเพิ่มจำนวนโปรเซสเซอร์ที่โหนด 3000 โหนด



รูปที่ 4.9 แสดง Efficiency เมื่อเพิ่มจำนวนโปรเซสเซอร์ที่โหนด 3000 โหนด

จากรูปที่ 4.8,4.9 จะได้ว่าทุก ๆ ครั้งที่มีการเพิ่มจำนวนโปรเซสเซอร์ประสิทธิภาพของโปรแกรมจะเพิ่มขึ้น ซึ่งสอดคล้องกับสมมุติฐานที่ตั้งไว้ที่ว่าโปรแกรมแบบขนานจะใช้เวลามากกว่าโปรแกรมแบบอนุกรม ในขณะเดียวกันยิ่งเพิ่มจำนวนโปรเซสเซอร์มากขึ้นเวลาในการประมวลผลจะลดลง

4.6 การเปรียบเทียบประสิทธิภาพระหว่างขั้นตอนวิธีของฟลอยด์แบบเสถียรกับแบบปกติ



รูปที่ 4.10 การเปรียบเทียบการทำงานระหว่างขั้นตอนวิธีต่าง ๆ

จากรูปที่ 4.10 จะได้ว่าการเพิ่มความสามารถให้สเถียรจะต้องใช้เวลาเพิ่มถ้าหากคิดระหว่างฟลอยด์ปกติ กับฟลอยด์แบบเสถียรจะได้ว่าต้องใช้เวลาเพิ่มเป็น 3 เท่าโดยประมาณและในขณะที่เทียบฟลอยด์แบบขนานกับฟลอยด์แบบขนานกับเสถียร จะต้องใช้เวลาเพิ่มขึ้น 7 เท่าโดยประมาณ ดังนั้นในการที่จะเลือกใช้ความสามารถในการซ่อมเส้นทางหรือเพื่อความสเถียรนั้นยังต้องคำนึงถึงระยะเวลาที่ต้องใช้ด้วยทั้งนี้ทั้งนั้นจะต้องนำไปเปรียบเทียบความคุ้มค่าระหว่างซ่อมเส้นทางกับไม่ซ่อมทางแล้วคำนวณใหม่หมดเพื่อหาจุดเหมาะสมในการใช้งานต่อไป

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

งานวิจัยชิ้นนี้ทำขึ้นมาเพื่อพัฒนาขั้นตอนวิธีการหาระยะทางที่สั้นที่สุดจากจุดทุกจุด ซึ่งในงานวิจัยนี้ได้หยิบขั้นตอนวิธีแบบฟลอยด์และแบบการประยุกต์เมทริกซ์เพื่อนำมาพัฒนาให้มีความสามารถเก็บเส้นทางได้ ซ่อมแซมทางได้ทำให้เกิดความเสถียรและเพิ่มประสิทธิภาพด้านความเร็วด้วยการนำขั้นตอนวิธีแบบขนานมาประยุกต์ ดังนั้น งานวิจัยชิ้นนี้จะประกอบไปด้วย

1) ขั้นตอนวิธีแบบฟลอยด์กับการเก็บเส้นทาง
2) ขั้นตอนวิธีแบบฟลอยด์กับการเก็บเส้นทางและเสถียร
3) ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีแบบฟลอยด์กับการเก็บเส้นทาง
4) ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีแบบฟลอยด์กับการเก็บเส้นทางและเสถียร
5) ขั้นตอนวิธีการประยุกต์เมทริกซ์กับการเก็บเส้นทาง
6) ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีการประยุกต์เมทริกซ์กับการเก็บเส้นทาง

จากการทดลองสร้างขั้นตอนวิธีต่าง ๆในงานวิจัยพบว่าขั้นตอนวิธีแบบฟลอยด์สามารถทำงานได้เร็วกว่าขั้นตอนวิธีการประยุกต์เมทริกซ์ นอกจากนี้การประยุกต์ขั้นตอนวิธีแบบขนานให้ขั้นตอนวิธีทั้งสองแบบสามารถเพิ่มประสิทธิภาพด้านความเร็วได้อย่างมีนัยสำคัญตามจำนวนโปรเซสเซอร์ที่เพิ่มขึ้น และสุดท้ายในการ
เพิ่มคุณสมบัติให้สามารถซ่อมเส้นทางได้ทำให้ขั้นตอนวิธีมีความเสถียรสามารถทำได้โดยใช้พื้นที่หน่วยความจำและเวลาในการคำนวณตั้งแต่เริ่มต้นความสามารถ
เมื่อเส้นทางพังระหว่างการใช้งานนั้นพบว่ายังใช้หน่วยความจำและเวลามากดังนั้นในการเลือกใช้ควรพิจารนาคุณสมบัตินี้กับหน่วยความจำและเวลาที่ใช้ในตอนเริ่มต้นให้เหมาะสม โดยข้อมูลของการหาระยะทางที่สั้นที่สุดจะสรุปข้อมูลตามตารางที่ 5.1 , 5.2 และข้อมูลของการซ่อมเส้นทางจะแสดงดังตารางที่ 5.3

ชื่อขั้นตอนวิธี	ความซับซ้อนด้านเวลา	เวลาประมวลผล (ที่ n = 1000)	หน่วยความจำ	เวลาในการติดต่อสื่อสาร
1) ขั้นตอนวิธีแบบฟลอยด์กับการเก็บเส้นทาง	$O(n^3)$	0.917	n^2	-
2) ขั้นตอนวิธีแบบฟลอยด์กับการเก็บเส้นทางและเสถียร	$O(n^3)$	4.472000122	n^3	-
3) ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีแบบฟลอยด์กับการเก็บเส้นทาง	$O(n^2)$,p=n	0.258569479	n^2	p + np
4) ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีแบบฟลอยด์กับการเก็บเส้นทางและเสถียร	$O(n^2)$,p=n	2.057008505	n^3	p + np

ตารางที่ 5.1 แสดงข้อมูลเปรียบเทียบของแต่ละขั้นตอนวิธีของฟลอยด์ในการหาเส้นทางที่สั้นที่สุด

ชื่อขั้นตอนวิธี	ความซับซ้อนด้านเวลา	เวลาประมวลผล (ที่ n =1000)	หน่วยความจำ	เวลาในการติดต่อสื่อสาร
1) ขั้นตอนวิธีการประยุกต์เมทริกซ์กับการเก็บเส้นทาง	$O(n^3\log(n))$	7.571	n^3	-
2) ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีการประยุกต์เมทริกซ์กับการเก็บเส้นทาง	$O(n^2\log(n))$,p=n	3.967983	n^3	p + p log(n)

ตารางที่ 5.2 แสดงข้อมูลเปรียบเทียบของแต่ละขั้นตอนวิธีในการหาเส้นทางที่สั้นที่สุด

ชื่อขั้นตอนวิธี	กรณีที่โหนดเสียหายเป็นโหนดเริ่มต้น	กรณีที่โหนดเสียหายเป็นโหนดสุดท้าย	กรณีที่โหนดเสียหายเป็นอื่นๆที่ใช้งานอยู่
1) ขั้นตอนวิธีแบบฟลอยด์กับการเก็บเส้นทาง	$O(n^3)$	$O(n^3)$	$O(n^3)$
2) ขั้นตอนวิธีแบบฟลอยด์กับการเก็บเส้นทางและเสถียร	$O(n^3)$	Best-Case: O(n) เมื่อ p = n Worst-Case : $O(n^2)$ เมื่อ p = n	
3) ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีแบบฟลอยด์กับการเก็บเส้นทาง	$O(n^2)$, p=n	$O(n^2)$, p=n	$O(n^2)$, p=n
4) ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีแบบฟลอยด์กับการเก็บเส้นทางและเสถียร	$O(n^2)$, p=n	Best-Case: O(n) เมื่อ p = n Worst-Case : $O(n^2)$ เมื่อ p = n	
5) ขั้นตอนวิธีการประยุกต์เมทริกซ์กับการเก็บเส้นทาง	$O(n^3\log(n))$	$O(n^3\log(n))$	$O(n^3\log(n))$
6) ขั้นตอนวิธีแบบขนานกับขั้นตอนวิธีการประยุกต์เมทริกซ์กับการเก็บเส้นทาง	$O(n^2\log(n))$,p=n	$O(n^2\log(n))$,p=n	$O(n^2\log(n))$,p=n

ตารางที่ 5.3 แสดงข้อมูลเปรียบเทียบของแต่ละขั้นตอนวิธีในการซ่อมเส้นทาง

5.2 ข้อเสนอแนะ

ขั้นตอนวิธีที่มีความสามารถแบบเสถียรจะต้องนำไปคิดกับความคุ้มค่าซึ่งจะต้องคิดจากสิ่งที่ต้องสูญเสียไปคือเวลาและพื้นที่หน่วยความจำให้มีความสอดคล้องกับการใช้งานจริงโดยสามารถนำไปเป็นหัวข้อในงานวิจัยต่อไป

แหล่งอ้างอิง

[1] รศ.ดร.จีรพร วีระพันธุ์. 2016. เอกสารประกอบการเรียนขั้นตอนวิธีแบบขนาน. คณะวิทยาศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง.

[2] Wikipedia. 2017. Floyd–Warshall_algorithm. [Online]. Available : https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm.

[3] Wikipedia. 2017. Parallel computing. [Online]. Available : https://en.wikipedia.org/wiki/Parallel_computing.

[4] Nontaporn Taptimhin. 2014. MPI. [Online]. Available : http://parallelcomputing57.blogspot.com/2014/12/blog-post.html.

[5] IBM. 2006. single-dual core. [Online]. Available : https://www.ibm.com/developerworks/data/library/techarticle/dm-0611zikopoulos2/.

[6] Wikipedia. 2017. Multi core. [Online]. Available : https://en.wikipedia.org/wiki/Multi-core_processor.

[7] overclockzone. 2017. HyperThread. [Online]. Available : https://www.overclockzone.com/spin9/review/cpu/intel/pd_820/index2.html.

[8] sujaneuy. 2017. Operating System. [Online]. Available : https://sujaneuy.wordpress.com/2015/09/14/operating-system/.

[9] Bernard Marr. 2017. Big Data. [Online]. Available : https://www.linkedin.com/pulse/20140306073407-64875646-big-data-the-5-vs-everyone-must-know.

[10] Wikipedia. 2017. Computer cluster. [Online]. Available : https://en.wikipedia.org/wiki/Computer_cluster.

[11] bthreiynbththi99. 2017. [Online]. Available : https://sites.google.com/site/bthreiynbththi99/1/kar-pramwl-phl-hrux-por-ses.

งานทะเบียนคณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

คำรับรองเล่มโครงงานพิเศษ/ปัญหาพิเศษ/สหกิจศึกษา

วันที่ 23 เดือน กรกฎาคม พ.ศ 2567

ข้าพเจ้า นาย/~~นาง/นางสาว~~ นนทกร เกิดมงคล รหัสประจำตัว 57050252

นาย/~~นาง/นางสาว~~ ปฏิพล ทวีชาติ รหัสประจำตัว 57050269

นักศึกษาหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชา วิทยาการ คอมพิวเตอร์ ภาควิชา วิทยาการ คอมพิวเตอร์

ขอรับรองว่าโครงงานพิเศษ/ปัญหาพิเศษ/สหกิจศึกษา เรื่อง

ชื่อภาษาไทย การ หาเส้นทางที่สั้นที่สุดแบบเสถียรด้วยขั้นตอนวิธีแบบขนาน

ชื่อภาษาอังกฤษ PARALLEL ALGORITHMS FOR ALL-PAIR SHORTEST PATHS WITH RELIABLE ROUTING

ปีการศึกษา 2560

เป็นผลงานวิจัยที่มิได้คัดลอกหรือละเมิดลิขสิทธิ์ของผู้อื่นและได้ผ่านการตรวจสอบความซ้ำซ้อนเรียบร้อยแล้ว และได้แนบเอกสารการตรวจสอบการลอกเลียนงานวรรณกรรมที่ตรวจสอบจากเล่มโครงงานพิเศษ/ปัญหาพิเศษ/สหกิจศึกษาฉบับสมบูรณ์แล้ว

โปรแกรมอักขราวิสุทธิ์ 0.00 % หรือโปรแกรม Turnitin .....%

ลงชื่อ ปฏิพล ทวีชาติ ลงชื่อ นนทกร เกิดมงคล

(นาย ปฏิพล ทวีชาติ) (นาย นนทกร เกิดมงคล)

นักศึกษา นักศึกษา

ข้าพเจ้า ศ. / รศ. / ผศ. / ดร. / อ..... อาจารย์ที่ปรึกษาโครงงานพิเศษ/ปัญหาพิเศษ/สหกิจศึกษา ได้ตรวจสอบโครงงานพิเศษ/ปัญหาพิเศษ/สหกิจศึกษาของนักศึกษาข้างต้น แล้ว ขอรับรองว่าเป็นผลงานวิจัยของนักศึกษาจริงและมีเนื้อหาสมบูรณ์ จึงลงชื่อไว้เป็นหลักฐาน

ลงชื่อ..... ลงชื่อ..... ลงชื่อ.....

อาจารย์ที่ปรึกษา อาจารย์ที่ปรึกษาร่วม อาจารย์ที่ปรึกษาร่วม