การพัฒนาการประมวลผลแบบขนานบนคลัสเตอร์

Parallel Processing On Cluster

นาย สรพงษ์ จรัญญากรณ์

นาย สราวุธ อมราสิงห์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

การพัฒนาการประมวลผลแบบขนานบนคลัสเตอร์

Parallel Processing On Cluster

โดย

นาย สรพงษ์ จรัญญากรณ์

นาย สราวุธ อมราสิงห์

อาจารย์ที่ปรึกษา

ผศ.ดร. บรรจง ปิยธำรง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

ปริญญานิพนธ์ปีการศึกษา 2543

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาการประมวลผลแบบขนานบนคลัสเตอร์

Parallel Processing On Cluster

ผู้จัดทำ

1. นาย สรพงษ์ จรัญญากรณ์ รหัสประจำตัว 40010819
2. นาย สราวุธ อมราสิงห์ รหัสประจำตัว 40010827

อาจารย์ที่ปรึกษา

(รศ. บรรจง ปิยธำรง)

การพัฒนาการประมวลผลแบบขนานบนคลัสเตอร์

นาย สรพงษ์ จรัญญากรณ์
นาย สราวุธ อมราสิงห์
รศ. บรรจง ปิยธำรง อาจารย์ที่ปรึกษา
ปีการศึกษา 2543

บทคัดย่อ

ในปัจจุบันเทคโนโลยีทางด้านคอมพิวเตอร์พัฒนาขึ้นอย่างรวดเร็ว ทำให้โปรแกรมต่าง ๆ มีความซับซ้อนมากขึ้น รวมไปถึงการประมวลผลข้อมูลที่มีขนาดใหญ่ก็มีใช้อย่างมากขึ้น ทำให้เกิดความต้องการในการใช้งานคอมพิวเตอร์ที่มีประสิทธิภาพสูง แต่เครื่องที่มีความสามารถดังกล่าวนั้นในทุกวันนี้มีราคาค่อนข้างสูง การประมวลผลแบบขนานบนระบบคลัสเตอร์จึงเริ่มมีการนำมาใช้ขึ้น ระบบคลัสเตอร์เป็นระบบที่ทำขึ้นมาเพื่อรองรับการประมวลผลแบบขนาน การประมวลผลแบบขนานนั้นได้สนับสนุนแนวคิดของการนำกลับมาใช้ใหม่ โดยสามารถนำเครื่องที่เก่าและไม่ได้ใช้ประโยชน์แล้วมาทำการใช้ประมวลผลได้ แต่การจะเชื่อมต่อเครื่องเหล่านั้นให้มีประสิทธิภาพเท่าเครื่องที่มีความเร็วสูง ๆ นั้น ต้องใช้เครื่องเป็นจำนวนมาก ซึ่งนี่ก็คือข้อเสียข้อหนึ่ง

การประมวลผลแบบขนานเป็นการนำพีซี หลาย ๆ เครื่องมาเชื่อมต่อกันและช่วยกันประมวลผลงานแต่ละงานจะไม่ทำอยู่บนโพรเซสเซอร์เพียงตัวเดียว แต่จะทำการแบ่งไปประมวลผลในโพรเซสเซอร์ตัวอื่นด้วย การประมวลผลแบบขนานส่วนสำคัญคือการแบ่งงาน ในส่วนการแบ่งงานบนคลัสเตอร์นั้นจะใช้ Message Parallel Interface (MPI) ซึ่งเป็นมาตรฐานการแบ่งงานชนิดหนึ่งเข้ามาช่วยกระจายงานไปยังโพรเซสเซอร์อื่น ๆ ในหลายรูปแบบของการสื่อสาร เช่น Point-to-point communication, Collective communication MPI จะใช้หลักการการสื่อสารระหว่างโพรเซสเซอร์ในรูปแมสเซจ แต่การพัฒนาโปรแกรมเชิงขนานนั้นยังไม่แพร่หลายนัก เนื่องจากโปรแกรมเมอร์ส่วนใหญ่มักคุ้นเคยกับการเขียนโปรแกรมซีเควนเชียลมากกว่า ทำให้เข้าใจแนวคิดเชิงขนานได้ยาก

วิทยานิพนธ์ฉบับนี้ได้ทำการทดสอบเพื่อแสดงให้เห็นถึงประสิทธิภาพและเปรียบเทียบการทำงานระหว่างการทำงานบนเครื่องเดียวด้วยโปรแกรมแบบซีเควนเชียล กับการทำงานเชิงขนาน ซึ่งการทำงานแบบขนานจะช่วยให้การทำงานรวดเร็วขึ้น การประมวลผลแบบขนานนั้นประสิทธิภาพของการทำงานขึ้นอยู่กับ อัลกอริทึมในการแบ่งงานและสเปคของเครื่องที่ใช้ในการคำนวณเป็นอย่างมาก สเปคของเครื่องในการประมวลผลควรมีขนาดที่ใกล้เคียงกันเพื่อลดปัญหาการรอกันของงาน และอัลกอริทึมก็ควรจะสนับสนุนการแบ่งงานอย่างเหมาะสมด้วย

Parallel Processing On Cluster

Mr.Sorapong Charunyakorn
Mr.Sarawut Amarasing
Assoc.Prof. Banjong Piyatamrong
2000

Abstact

According to the rapid improvement in computer technology nowadays, many programs have become more complicated as well as the increasing size of data used to process at a time. This leads to the increased demand of high-efficient computer which is costly. The invention of parallel processing on cluster helps us to facilitate this difficulty. Cluster is the system invented to support the parallel processing which gets the idea from reuse. By reusing, we can use the old-fangeld computers. But the advantage of using the old computer is that — we have to connect many computers together in order to make them to have as high capabilities as the high-efficient computers.

The parallel processing is to connect many PCs together in order to share the processing so that each task will not be on only one processor; but it will share the processing to other processors as well. The most important point of parallel processing is the division of task. The division of task on cluster uses Message Parallel Interface (MPI) which is one of standard divisions used to divide the task to other processors in the forms of communication such as Point-to-point communication and Collective communication. MPI communicates through processors by messaging. The parallel processing is not yet well-known because the majority of programmers do not get used to the parallel processing but the sequential processing instead.

This thesis shows the benchmarking of parallel processing and the comparison of the response-time between processing on single computer by sequential processing and by parallel processing. The processing parallel processing helps us to work faster and its efficiency depends mostly on the algorithm of the division of task and the spec of computer used for computing. Spec of computer should be similar in order to reduce the problem of queuing of task and the algorithm should support the division suitably.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้ปริญญานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์บรรจง ปิยธำรง อาจารย์ที่ปรึกษาปริญญานิพนธ์ ที่ให้ความเอาใจใส่ แนะนำ และช่วยเหลือเสมอมา และอีกท่านหนึ่งก็คือ อาจารย์อัครเดช วัชรภูพงษ์ รวมถึงอาจารย์ทุก ๆ ท่านที่ให้คำปรึกษาและชี้แนะแนวทาง ทำให้โครงงานวิจัยนี้สำเร็จลุล่วงไปด้วยดี ซึ่งก็ต้องขอขอบพระคุณทุกท่านไว้เป็นอย่างมาก

และต้องขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ เอาใจใส่เสมอมา ในทุก ๆ ด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

นาย สรพงษ์ จรัญญากรณ์
นาย สราวุธ อมราสิงห์

ผู้จัดทำ

สารบัญ

	หน้าที่
บทคัดย่อภาษาไทย	I
บทคัดย่อภาษาอังกฤษ	II
กิตติกรรมประกาศ	III
สารบัญตาราง	VI
สารบัญภาพ	VII
บทที่ 1 บทนำ	1
1.1 ความสำคัญและที่มา	1
1.2 วัตถุประสงค์ของงานวิจัย	2
1.3 ขอบเขตของงานวิจัย	2
1.4 อุปกรณ์ที่ใช้ในงานวิจัย	2
1.5 โครงร่างของงานวิจัย	2
บทที่ 2 ทฤษฎีและหลักการของคลัสเตอร์	4
2.1 ความหมายและหลักการทำงานของคลัสเตอร์	4
2.2 ชนิดของคลัสเตอร์	5
บทที่ 3 หลักการเขียนโปรแกรมแบบซีเควนเชียลและการเขียนโปรแกรมแบบขนาน	6
3.1 หลักการของการเขียนโปรแกรมแบบซีเควนเชียล	6
3.2 หลักการของการเขียนโปรแกรมแบบขนาน	6
3.2.1 เอนวิรอนเมนต์ของการเขียนโปรแกรมแบบขนาน	7
3.2.2 แนวทางการพัฒนาโปรแกรมแบบขนาน	8
3.3 การเปรียบเทียบระหว่าง การเขียนโปรแกรมแบบซีเควนเชียลกับ การเขียนโปรแกรมแบบขนาน	9
บทที่ 4 หลักการของ MPI	11
4.1 ความหมายของ MPI (Message Passing Interface)	11
4.2 องค์ประกอบต่าง ๆ ของ MPI	11
4.2.1 MPI โพรเซส	11
4.2.2 กลุ่มโพรเซส	12
4.2.3 คอมมิวนิเคชั่นคอนเท็ก	12
4.2.4 คอมมิวนิเคเตอร์	12
4.2.5 เวอร์ชวล ทอโปโลจี	13
4.3 การสื่อสาร	13
4.3.1 การสื่อสารแบบจุดต่อจุด	14

สารบัญ (ต่อ)

	หน้าที่
4.3.2 การสื่อสารแบบกลุ่ม	17
4.4 ความรู้พื้นฐานในการเขียนโปรแกรม MPI	19
บทที่ 5 แนวคิดในการออกแบบคลัสเตอร์	22
5.1 หลักการติดตั้งระบบคลัสเตอร์	22
บทที่ 6 แนวคิดและอัลกอริทึมในการทำงานของโปรแกรมแบบขนาน	24
6.1 ลักษณะของโปรแกรม	24
6.2 อัลกอริทึม	25
6.3 ขั้นตอนการเขียนโปรแกรม	26
บทที่ 7 การทดสอบระบบ	28
7.1 แนวคิดในการทดสอบระบบ	28
7.2 การทดสอบระบบ	29
7.2.1 สมมติฐานก่อนการทดลอง	29
7.2.2 การทดลอง	30
7.2.3 สรุปผลการทดลอง	42
บทที่ 8 บทสรุปและวิจารณ์	44
บรรณานุกรม	46
ภาคผนวก ก การติดตั้งโปรแกรมที่จำเป็นสำหรับการวิจัย	48
ภาคผนวก ข การวัดความเร็วในแบบขนาน	51
ภาคผนวก ค ซอร์สโค้ดของโปรแกรม	54

สารบัญตาราง

หน้าที่

ตารางที่ 3.3.1 แสดงข้อเปรียบเทียบระหว่างซีเควนเชียลโปรแกรมและโปรแกรมแบบขนาน 10
ตารางที่ 4.4.1 แสดงประเภทข้อมูลของ MPI ในภาษาซี 20
ตารางที่ 7.2.2.1 ตารางเปรียบเทียบความเร็วการประมวลผลระหว่างซีเควนเชียลกับแบบขนาน 1 โหนด 31
ตารางที่ 7.2.2.2 ตารางเปรียบเทียบความเร็วการประมวลผลแบบขนาน 2 โหนด 32
ตารางที่ 7.2.2.3 ตารางเปรียบเทียบความเร็วการประมวลผลแบบขนาน 3 โหนด 33
ตารางที่ 7.2.2.4 ตารางเปรียบเทียบความเร็วการประมวลผลแบบขนาน 4 โหนด 34
ตารางที่ 7.2.2.5 เปรียบเทียบความเร็วระหว่างการประมวลผลแบบซีเควนเชียลและแบบขนาน 1 โหนด 37
ตารางที่ 7.2.2.6 แสดงเวลาที่ใช้ในการประมวลผลแบบขนาน 2 โหนดที่ใช้โหนดช้าเข้าไป 38
ตารางที่ 7.2.2.7 แสดงเวลาที่ใช้ในการประมวลผลแบบขนาน 2 โหนดที่ใช้โหนดเร็วเข้าไป 39
ตารางที่ 7.2.2.8 แสดงเวลาที่ใช้ในการประมวลผลแบบขนาน 3 โหนด 40

สารบัญภาพ

		หน้าที่
ภาพที่ 2.1.1	แสดงลักษณะทั่วไปของระบบคลัสเตอร์	4
ภาพที่ 3.2.1	แสดงประเภทของ การเขียนโปรแกรมแบบขนาน	7
ภาพที่ 3.2.1.1	แสดงโครงสร้างการเขียนโปรแกรมแบบขนาน	8
ภาพที่ 3.3.1	การเปรียบเทียบระหว่างการเขียนโปรแกรมแบบขนานและแบบซีเควนเชียล	9
ภาพที่ 4.2.3.1	แสดงลักษณะของ คอมมิวนิเคเตอร์	12
ภาพที่ 4.2.4.1	แสดงลักษณะเวอร์ชวล ทอโปโลจี	13
ภาพที่ 4.3.1	การสื่อสารแบบนอน-บล็อกกิ้ง เวอร์ชัน	14
ภาพที่ 4.3.1.1	การสื่อสารแบบจุดต่อจุด	14
ภาพที่ 4.3.1.2	การเกิดเดดล็อกในการส่งแมสเซจ	15
ภาพที่ 4.3.1.3	แสดงภาพ การรับ-ส่งแมสเซจตามลำดับ	16
ภาพที่ 4.3.1.4	แสดงการผิดพลาดในการรับ-ส่ง	16
ภาพที่ 4.3.2.1	แสดงลักษณะการบรอดคาส	17
ภาพที่ 4.3.2.2	แสดงสแคทเทอร์และเกเทอร์	18
ภาพที่ 4.3.2.3	แสดงรีดิวซ์และสแกน	18
ภาพที่ 4.3.2.4	แสดงเกเทอร์-ทู-ออล	18
ภาพที่ 5.1.1	แสดงหลักการทำงานของดิสก์เลสคลัสเตอร์	23
ภาพที่ 6.2.1	แสดงหลักการออกแบบการทำงานของโปรแกรมแบบขนาน	24
ภาพที่ 6.2.2	แสดงอัลกอริทึมของโปรแกรม	25
ภาพที่ 7.2.1.1	แสดงการทดสอบระบบคลัสเตอร์อื่น ๆที่เผยแพร่บนอินเทอร์เน็ต	30
ภาพที่ 7.2.1.2	แสดงผลการทดสอบระบบคลัสเตอร์ในต่างประเทศ	30
ภาพที่ 7.2.2.1	แสดงความสัมพันธ์ระหว่างเวลาที่ใช้คำนวณและจำนวนโพรเซสที่ใช้	35
ภาพที่ 7.2.2.2	แสดงลักษณะการทำงานจากโหนดที่ความเร็วต่างกัน	41
ภาพที่ ก-1	แสดงหน้าจอโปรแกรม SCMS	50
ภาพที่ ข-1	แสดงหน้าจอการใช้งาน MPI	51
ภาพที่ ข-2	แสดงหน้าจอเมื่อเลือก Build & Run	52
ภาพที่ ข-3	แสดงหน้าจอช่วงเวลาในการทำงานของแต่ละโพรเซสที่เวลาใด ๆ	52
ภาพที่ ข-4	แสดงหน้าจอสถานะการทำงานของแต่ละโพรเซส	53

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มา

ปัจจุบันนี้ คอมพิวเตอร์พีซี มีการพัฒนาไปเร็วมากเพื่อให้ทันกับพัฒนาการของเทคโนโลยีในปัจจุบัน ทำให้เครื่องที่ตกรุ่นและไม่ได้ใช้งาน และเสียประโยชน์ไปอย่างน่าเสียดาย ซึ่งระบบการประมวลผลแบบคลัสเตอร์นั้นเป็นการนำพีซีที่ไม่ได้ใช้งานเหล่านั้นนำกลับมาใช้ใหม่ โดยการนำเครื่องรุ่นเก่า เช่น 386, 486 มาทำเป็นคลัสเตอร์ นอกจากจะทำให้เราได้โพรเซสเซอร์ ที่มีประสิทธิภาพสูงจากเครื่องเหล่านี้แล้ว ยังลดการสิ้นเปลืองจากการซื้อซูเปอร์คอมพิวเตอร์ที่มีราคาแพงจากต่างประเทศ อีกทั้งยังลดค่าใช้จ่ายในการบำรุงรักษาอีกด้วย ซึ่งทั้งหมดนี้จะเป็นการช่วยส่งเสริมการนำทรัพยากรกลับมาใช้ใหม่ในระดับประสิทธิภาพคงเดิมด้วย

จากเหตุผลดังกล่าวปัจจุบันทำให้มีผู้คนสนใจในการศึกษาในเรื่องคลัสเตอร์มากขึ้น เพื่อลดค่าใช้จ่ายในการสั่งซื้อเครื่องซูเปอร์คอมพิวเตอร์ที่มีราคาแพง ๆเข้ามา ทั้ง ๆที่เมื่อเปรียบเทียบกันแล้วประสิทธิภาพในการคำนวณแทบจะไม่แตกต่างกันเลย โดยในการศึกษาก็พยายามหาโอเอสที่มีความเหมาะสมในการทำคลัสเตอร์มาใช้ ซึ่งก็เลือกใช้ลีนุกซ์เนื่องจากเหตุผลหลายประการ เช่น ลีนุกซ์เป็นฟรีซอร์ฟแวร์ซึ่งเป็นการเหมาะกับผู้ที่ทดลองใช้งาน และสามารถที่จะพัฒนาเพิ่มเติมจากที่เป็นอยู่ได้ง่าย และอีกอย่างคือลีนุกซ์เป็นระบบปฏิบัติการที่เหมาะกับการใช้งานด้านเครือข่ายทั้งในด้านความมั่นคง ปลอดภัย และมีเสถียรภาพในการทำงานสูงด้วย

เมื่อต้องการที่จะใช้งานก็จำเป็นต้องมีการเรียนรู้ถึงภาษาโปรแกรมที่สามารถนำมาใช้งานได้ ซึ่งภาษาโปรแกรมที่มีลักษณะเป็นภาษาโปรแกรมแบบขนานเหมาะสำหรับการนำมาใช้ โดยในการวิจัยนี้ได้ใช้ MPI มาเป็นมาตรฐานในการทำงาน เนื่องจากมีข้อดีหลายประการ เช่น มีฟังก์ชันการทำงานที่รองรับการทำงานได้กว้างขวาง และ สามารถที่จะเขียนซีเควนเชียลโปรแกรมเพื่อมาใช้แปลงเป็นโปรแกรมแบบขนานโดยใช้ MPI นี้ได้

จากที่กล่าวมาทั้งหมด ในการวิจัยครั้งนี้จึงได้สร้างโปรแกรมในการคำนวณทางคณิตศาสตร์ คือการคำนวณเมตริกซ์ขึ้นมารันบนระบบลีนุกซ์คลัสเตอร์ที่ติดตั้งขึ้น เพื่อชี้ให้เห็นถึงความแตกต่างทางด้านประสิทธิภาพการทำงานระหว่างซีเควนเชียลโปรแกรมที่ทำงานบนเครื่องเครื่องเดียวกับโปรแกรมภาษา MPI ซึ่งจะแปลงเป็นโปรแกรมแบบขนานในการทำงานเป็นระบบคลัสเตอร์

1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 เพื่อศึกษาความแตกต่างระหว่างซีเควนเชียลโปรแกรมกับโปรแกรมแบบขนาน

1.2.2 เพื่อทดสอบติดตั้งระบบลีนุกซ์คลัสเตอร์ ว่าจะต้องมีการติดตั้งอย่างไร และระบบมีการทำงานอย่างไรบ้าง

1.2.3 เพื่อศึกษาการเขียนโปรแกรมแบบขนานด้วย MPI

1.2.4 เพื่อเป็นแนวทางสำหรับใช้ในการคำนวณงานที่ต้องการใช้การคำนวณสูง ๆ เช่น งานทางด้านสามมิติ งานด้านการคำนวณคณิตศาสตร์ชั้นสูง เป็นต้น

1.3 ขอบเขตของงานวิจัย

จากงานวิจัยในครั้งนี้ ได้ทำการติดตั้งระบบลีนุกซ์คลัสเตอร์ โดยลงระบบลีนุกซ์ เรดแฮท 6.2 ลงบนเครื่องเซิร์ฟเวอร์ และทำการติดตั้งระบบคลัสเตอร์โดยใช้วิธีการ ดิสก์เลสคลัสเตอร์ เสร็จแล้วได้มีการเขียนแอพพลิเคชั่นที่เขียนด้วยภาษา MPI ซึ่งเป็นโปรแกรมการคำนวณทางด้านเมตริกซ์ โดยเหตุผลที่ใช้เมตริกซ์เนื่องจากสามารถประยุกต์ใช้กับงานต่าง ๆ ทั้งทางด้านคณิตศาสตร์และทางด้านวิทยาศาสตร์ได้อย่างหลากหลาย อีกทั้งยังเป็นงานที่ต้องการการประมวลผลด้วยความเร็วสูง จึงเหมาะกับการวิจัยในครั้งนี้ได้เป็นอย่างดี

แต่ถึงอย่างไรก็ตาม การวิจัยครั้งนี้ก็มีข้อจำกัดอยู่มาก เช่น ปริมาณเครื่องที่ใช้ทำคลัสเตอร์มีจำนวนที่ไม่มากนักรวมทั้งอัลกอริทึมที่เขียนก็ยังทำให้การคำนวณที่มีประสิทธิภาพที่ไม่ดีนัก แต่ก็หวังว่าจะเป็นแนวทางสำหรับผู้สนใจได้ทำการพัฒนาให้ดียิ่งขึ้นต่อไป

1.4 อุปกรณ์ที่ใช้ในการวิจัย

1. เครื่องคอมพิวเตอร์ 4 เครื่องประกอบไปด้วย
   1.1 เครื่องเซิร์ฟเวอร์ ที่ได้ทำการติดตั้งระบบ ดิสก์เลสคลัสเตอร์ไว้แล้ว
   1.2 เครื่องไคลเอนท์อีก 3 เครื่องที่จะใช้เป็นโหนดในระบบคลัสเตอร์
2. แผ่นบูตเคอร์เนล เพื่อใช้ในการบูตเครื่องไคลเอนท์ทำการติดตั้งคลัสเตอร์
3. โปรแกรมทำการประมวลผลเมตริกซ์ ที่เขียนขึ้นด้วยโปรแกรม MPI
4. ฮับ เพื่อใช้ในการเชื่อมต่อระหว่างเครื่องเซิร์ฟเวอร์และเครื่องไคลเอนท์

1.5 โครงร่างของงานวิจัย

รายละเอียดของงานวิจัยประกอบไปด้วยบทต่าง ๆ ดังนี้

บทที่ 1 บทนำจะกล่าวถึงความเป็นมาและวัตถุประสงค์ของงานวิจัยนี้

บทที่ 2 จะกล่าวถึงทฤษฎีและหลักการของคลัสเตอร์

บทที่ 3 จะกล่าวถึงหลักการของการเขียนโปรแกรมแบบซีเควนเชียล(Sequencial programming)และ การเขียนโปรแกรมแบบขนาน (Parallel Programming)

บทที่ 4 กล่าวถึงหลักการของ MPI

บทที่ 5 กล่าวถึงแนวคิดในการออกแบบคลัสเตอร์

บทที่ 6 กล่าวถึงแนวคิดและอัลกอริทึมการทำงานของโปรแกรมแบบขนาน

บทที่ 7 กล่าวถึงการทดลองของงานวิจัยนี้

บทที่ 8 กล่าวถึงสรุปและวิจารณ์ของงานวิจัยนี้

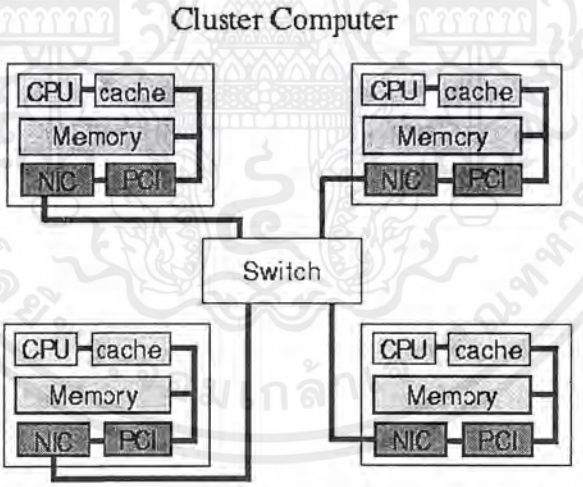
บทที่ 2

ทฤษฎีและหลักการทำงานของคลัสเตอร์

2.1 ความหมายและหลักการทำงานของคลัสเตอร์

ระบบการประมวลผลแบบคลัสเตอร์เป็นการนำเอาระบบพีซีคอมพิวเตอร์หลาย ๆตัวมาช่วยในการประมวลผลเพื่อที่จะทำให้การประมวลผลข้อมูลต่าง ๆทำได้รวดเร็วขึ้นกว่าการนำเอาคอมพิวเตอร์เครื่องเดียวมาประมวลผล ซึ่งในระบบคลัสเตอร์นี้ต่างจากระบบแลน คือ ระบบแลน จะเป็นการเชื่อมต่อคอมพิวเตอร์จำนวนมากเพื่อใช้ทรัพยากรร่วมกัน โดยแต่ละเครื่องยังเป็นเครื่องอิสระ แต่ระบบคลัสเตอร์เป็นการรวมเครื่องหลาย ๆเครื่องที่ประสานงานกันอย่างแน่นแฟ้นจนเปรียบเสมือนเป็นเครื่องเดียวกัน

โครงสร้างของระบบพีซีคลัสเตอร์ประกอบด้วยเครื่องพีซีสมรรถนะสูงจำนวนหนึ่งที่แต่ละเครื่องต่างก็มีหน่วยความจำเป็นของตนเองแล้วนำมาเชื่อมต่อกันด้วยเครือข่ายความเร็วสูง เช่น ระบบ ATM Switch,Fast Ethenet Switch,Myrinet ถ้าดูจากโครงสร้างแล้วจะเห็นได้ว่าโครงสร้างของระบบคลัสเตอร์ก็คือ คอมพิวเตอร์แบบขนานที่ใช้หน่วยความจำแยกนั่นเอง



รูปภาพที่ 2.1.1 แสดงลักษณะทั่วไปของระบบคลัสเตอร์

บางคนอาจจะสงสัยว่า ถ้าอย่างนั้นระบบแลน ทุกที่ก็เป็นคอมพิวเตอร์แบบขนานหรือไม่ คำตอบคือใช่และไม่ใช่ ที่ว่าใช่เพราะว่าโครงสร้างพื้นฐานของระบบคลัสเตอร์และระบบแลน ไม่ได้ต่างกันเลย ส่วนที่ว่าไม่ใช่นั้นเนื่องจากระบบแลน ธรรมดาไม่มีซอฟท์แวร์ที่จะนำความสามารถของการประมวลผลแบบขนานและกระจายมาใช้ได้ ซึ่งหัวใจของระบบคลัสเตอร์นั้นมีอยู่ด้วยกัน 3 อย่าง คือ

เครือข่ายความเร็วสูง ระบบซอฟท์แวร์ที่สนับสนุนการทำงานแบบคลัสเตอร์ และโปรแกรมประยุกต์ที่ใช้ขีดความสามารถ ดังกล่าว

ในปัจจุบันมีแนวโน้มทางเทคโนโลยีหลายประการที่สนับสนุนการก้าวไปสู่ระบบคลัสเตอร์ เช่น

1. เนื่องจากโพรเซสเซอร์ของเครื่องพีซีในปัจจุบันมีสมรรถนะแทบจะไม่แตกต่างจากโพรเซสเซอร์ เครื่องซูเปอร์คอมพิวเตอร์ดังนั้นการเชื่อมระบบพีซีเข้าด้วยกันเพื่อทำงานเป็นซูเปอร์คอมพิวเตอร์จึงมีสมรรถนะไม่แตกต่างจากซูเปอร์คอมพิวเตอร์ที่ขายกันอยู่เลย
2. เนื่องจากเทคโนโลยีของพีซีมีการขายจำนวนมหาศาล ดังนั้นผลที่ได้คือ ราคาที่ถูกมากเมื่อเทียบกับประสิทธิภาพ นอกจากนั้นการแข่งขันที่รุนแรงยังทำให้เงินทุนที่ทุ่มให้การวิจัยและพัฒนานั้นมหาศาล การพัฒนาจึงเป็นไปอย่างรวดเร็ว ทำให้ได้ของที่มีประสิทธิภาพสูงขึ้นมากในราคาเท่าเดิมตลอดเวลา
3. ปัญหาอย่างหนึ่งที่พบอยู่เสมอเมื่อใช้ระบบราคาแพงคือ ค่าบำรุงรักษาที่สูงมาก นอกจากนั้นเมื่อเทคโนโลยีนั้นเก่าหรือช้าไปแล้ว การหาเงินทุนมาเพิ่มระบบแทบจะเป็นไปได้ยาก ในขณะที่ในระบบพีซีคลัสเตอร์การเพิ่มความสามารถทำได้ทีละน้อยในราคาที่ถูกกว่า นอกจากนั้นเครื่องที่นำออกจากระบบยังเอาไปใช้ต่อได้ ถ้าเราซื้อซูเปอร์คอมพิวเตอร์ เวลาซีพียู เก่าก็ต้องเปลี่ยนทิ้ง แต่พีซีคลัสเตอร์ยังเอาของเก่าไปลงซอฟท์แวร์ใช้งานในสำนักงานได้ด้วย
4. ความก้าวหน้าของซอฟท์แวร์สำคัญ เช่น ลีนุกซ์ ที่เป็นระบบปฏิบัติการฟรีที่มีประสิทธิภาพสูง ระบบโปรแกรมแบบขนาน MPI และ PVM ทำให้สามารถสร้างและใช้ขีดความสามารถของระบบคลัสเตอร์ได้
5. ระบบพีซีเป็นเทคโนโลยีที่คนส่วนใหญ่คุ้นเคย ทำให้สามารถบำรุงรักษาระบบได้โดยไม่ต้องจ้างบริษัทในราคาแพง

2.2 ชนิดของคลัสเตอร์

โครงสร้างของระบบคลัสเตอร์ในโลกแบ่งออกได้เป็น 2 แบบ คือ ระบบคลัสเตอร์แบบปิด และระบบคลัสเตอร์แบบเปิด ซึ่งรายละเอียดของการวางระบบในแต่ละแบบมีดังนี้

คลัสเตอร์แบบปิด

ระบบนี้จะต่อคลัสเตอร์ผ่านเกตเวย์ที่ซ่อนทั้งระบบจากภายนอก ข้อดีก็คือ มีความปลอดภัยสูง และจะเปลืองเนื้อที่อินเทอร์เน็ตแอดเดรสเพียงแอดเดรสเดียวเท่านั้น ข้อเสียคือ แต่ละโหนดในระบบไม่สามารถช่วยกันบริการข้อมูลกับโลกภายนอกได้ จึงเหมาะกับการใช้งานคำนวณขนาดใหญ่เท่านั้น

คลัสเตอร์แบบเปิด

ระบบนี้จะต่อกับเครือข่ายภายนอกโดยตรง ทำให้ผู้ใช้เข้าถึงทุกโหนดในระบบคลัสเตอร์ได้โดยตรง ข้อเสียก็คือ ความปลอดภัยจะต่ำลงมาก เพราะต้องคอยดูแลทุกเครื่องในระบบ และยังต้องการหมายเลขอินเทอร์เน็ตแอดเดรสจำนวนมาก แต่ข้อดีก็คือระบบสามารถช่วยกันบริการข้อมูลได้ จึงเหมาะกับงานบริการข่าวสารเป็นจำนวนมาก

บทที่ 3

หลักการเขียนโปรแกรมแบบซีเควนเชียลและการเขียนโปรแกรมแบบขนาน

3.1 หลักการของการเขียนโปรแกรมแบบซีเควนเชียล

การเขียนโปรแกรม ในลักษณะซีเควนเชียลมีมานานแล้ว ตั้งแต่ที่ผู้ใช้ต้องการสร้างแอพพลิเคชั่นโปรแกรมบนเครื่องเครื่องหนึ่ง ผู้ใช้ก็มีการเขียนซอร์สโค้ด โปรแกรมขึ้นบนเครื่องนั้น และทำการคอมไพล์ โปรแกรมเพื่อให้สามารถใช้งานได้ โดยเมื่อต้องการจะพัฒนาโปรแกรมใหม่ก็จำเป็นต้องมี อัลกอริทึม ของตัวโปรแกรมเดิม ไม่อย่างนั้นก็จำเป็นต้องมีการใช้เครื่องมือ (Tool) ต่าง ๆมาช่วย สาเหตุที่ผู้ใช้มักเขียนโปรแกรมในลักษณะซีเควนเชียลเนื่องจาก

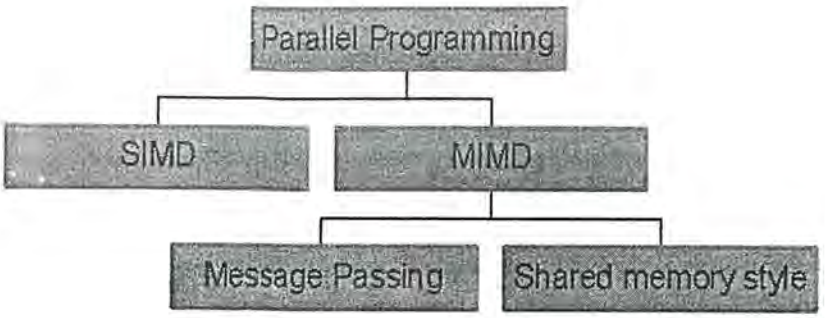
1. เนื่องจากว่ามีรูปแบบอัลกอริทึมต่าง ๆมากมายหลายแบบซึ่งได้คิดค้นมานานแล้ว และช่วยผู้ใช้ในการออกแบบโปรแกรม
2. เนื่องจากมีการใช้แมชีนโมเดลรูปแบบเดียวกันคือ von Neumann model ซึ่งสามารถรองรับได้กับทุก ๆรูปแบบอัลกอริทึมและทุก ๆ แพลตฟอร์ม
3. เนื่องจากมีภาษาทางซีเควนเชียลหลายตัวที่เป็นที่นิยมใช้กัน เช่น ฟอร์แทรน, โคบอล, ซี เป็นต้น

โอเปอเรชัน ที่สำคัญในการเขียนโปรแกรมแบบซีเควนเชียล มี 3 อย่าง คือ

1. ซีเควนเชียล โอเปอเรชัน เป็นการเขียนคำสั่งตามลำดับของการประมวลผลจากบนลงล่าง
2. คอนดิชัน โอเปอเรชันเป็นการเขียนตามเงื่อนไขต่าง ๆและประมวลผลตามเงื่อนไขที่ถูกเลือกนั้น
3. อิเทอเรทีฟ โอเปอเรชัน เป็นการเขียนคำสั่งให้ประมวลผลซ้ำกันภายในบล็อกคำสั่งหนึ่ง

3.2 หลักการของการเขียนโปรแกรมแบบขนาน

ในระบบซีเควนเชียลคอมพิวเตอร์ จะมีเพียงหนึ่งโปรแกรมที่ทำการประมวลผลในเวลาที่กำหนด ซึ่งคอมพิวเตอร์จะใช้งานเพียง 1 ซีพียู หรือ 1 โพรเซสเซอร์ในการทำงานแต่ละครั้ง ซึ่งโปรแกรมที่ถูกประมวลผลนี้เรียกว่า โพรเซส โพรเซสจะประกอบไปด้วยโปรแกรม ข้อมูล สแตก และระบบปฏิบัติการเป็นโครงสร้าง โดยแต่ละโพรเซสจะมีเพียงหนึ่งหมายเลขโพรเซส (process ID) ซึ่งเป็นค่าข้อมูลประเภทอินทีเจอร์ โดยซีเควนเชียลคอมพิวเตอร์สามารถทำมัลติโปรแกรมมิ่งได้เพียงกำหนดให้งานหนึ่งมีหลาย ๆโพรเซสและทำโพรเซสสลับกันไปมา แต่ก็ไม่ได้เป็นการทำงานที่เป็นแบบขนานที่แท้จริง เนื่องจากการประมวลผลใช้ตัวโพรเซสเซอร์แค่ตัวเดียว ซึ่งต่างจากแบบขนานที่มีโพรเซสเซอร์หลายตัวมาช่วยกันประมวลผล ซึ่งจากรูปที่ 3.2.1 การเขียนโปรแกรมแบบขนานสามารถแบ่งได้เป็น 2 ประเภท คือ



รูปภาพที่ 3.2.1 แสดงประเภทของ การเขียนโปรแกรมแบบขนาน

1.SIMD(Single Instruction Multiple Data) หมายถึง ระบบคอมพิวเตอร์แบบขนานที่แต่ละโพรเซสเซอร์สามารถประมวลผลคำสั่งเดียว โดยใช้ข้อมูลหลาย ๆตัวในเวลาเดียวกัน

2.MIMD(Multiple Instruction Multiple Data) หมายถึง ระบบคอมพิวเตอร์แบบขนานที่แต่ละโพรเซสเซอร์สามารถประมวลผลหลาย ๆ คำสั่ง ซึ่งอยู่บนข้อมูลต่างชนิดกันได้ ซึ่งเป็นการแบ่งงานออกเป็นส่วน ๆและแต่ละส่วนจะถูกประมวลผลโดยโพรเซสเซอร์ขนาน ทำให้การประมวลผลเป็นไปอย่างรวดเร็ว

ในการประมวลผลโพรเซสแบบขนาน นี้จะให้แต่ละส่วนที่มีอยู่สามารถแลกเปลี่ยนข้อมูลกันในช่วงเวลาหนึ่งได้ ดังนั้นจึงมีการติดต่อสื่อสารระหว่างโพรเซส ซึ่งมีอยู่ 2 วิธี คือ

1. ใช้วิธีการส่งแมสเซจ (Message Passing) คือจะนำการเชื่อมต่อเครือข่ายมาใช้ในการแลกเปลี่ยนข้อมูลระหว่างโพรเซสเซอร์ ซึ่งได้แก่ MPI และ PVM เป็นต้น
2. ใช้วิธีการแชร์หน่วยความจำ (Share Memory) ซึ่งเมื่อประมวลผลโปรแกรมแบบขนานและตัวโพรเซสเซอร์จะต้องระบุถึงจำนวนโพรเซสที่ต้องมีการประมวลผล และพร้อมที่จะส่งโพรเซสเหล่านี้ไปให้กับตัวโพรเซสเซอร์ที่ว่างอยู่ทำงานทันที

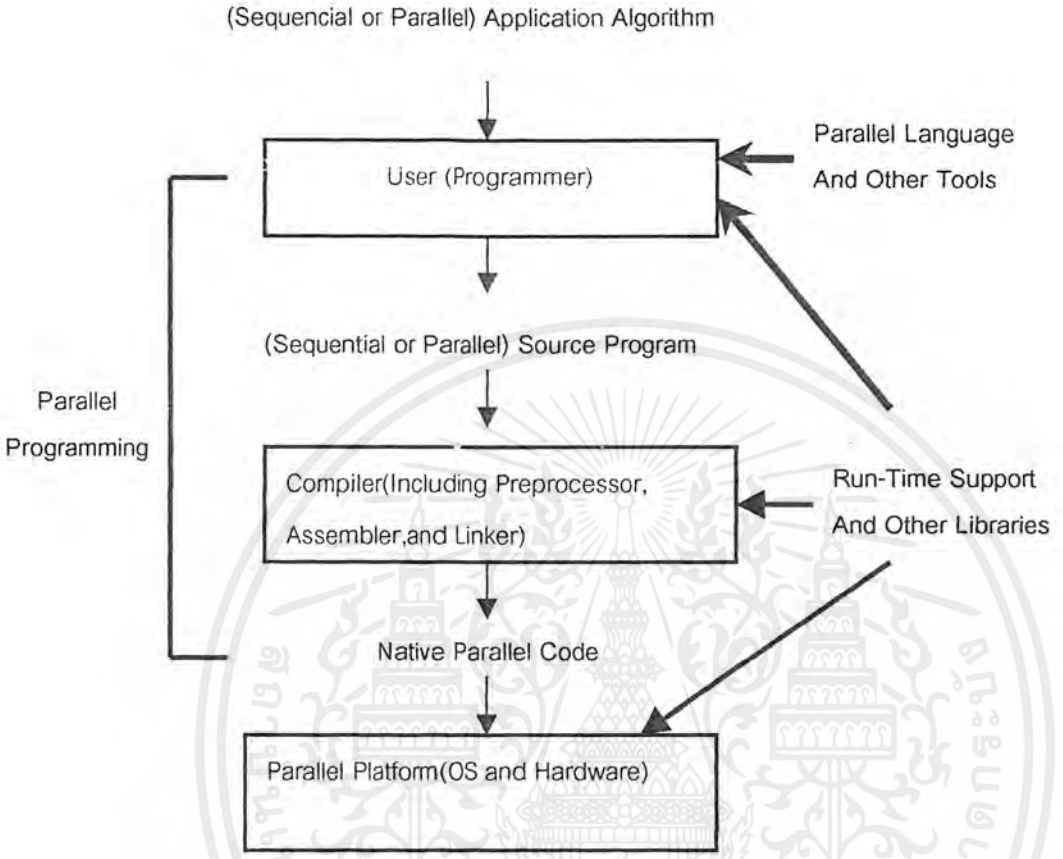
โดยส่วนมากการเขียนโปรแกรมแบบขนานนี้จะไม่สนับสนุนเครื่องมือ ต่าง ๆที่มีอยู่ และยังไม่เป็นที่เข้าใจหรือยอมรับจากผู้ใช้เป็นส่วนมาก โดยจะประกอบด้วยโมเดล 2 ระดับ คือ

1. โปรแกรมมิ่ง โมเดล เป็นโมเดลที่เกี่ยวข้องกับสิ่งที่โปรแกรมเมอร์มองเห็นและใช้ในการพัฒนา โปรแกรมแบบขนาน
2. เนทีฟ โมเดล เป็นโมเดลระดับต่ำกว่าระดับโปรแกรมมิ่ง โมเดล ซึ่งเป็นสิ่งที่โปรแกรมเมอร์มองเห็น ตามแต่ละแพลตฟอร์มของคอมพิวเตอร์แบบขนาน (parallel computer)

3.2.1 เอนวิรอนเมนท์ของการเขียนโปรแกรมแบบขนาน (Parallel Programming Environment)

จากมุมมองของผู้ใช้ ปกติระบบโครงสร้างของระบบประมวลผลแบบขนานเป็นไปดังรูปที่ 3.2.1.1 ซึ่งในตอนแรกจะนำส่วนของอัลกอริทึม ทั้งแบบ ซีเควนเชียลหรือแบบขนานมาทำการอิมพลีเมนต์ โดยใช้ส่วนของภาษาแบบขนาน หรือ ทูลต่าง ๆเข้าช่วยในการทำให้ได้ ซอร์สโปรแกรม ซึ่งจะผ่านส่วนของตัวคอมไพเลอร์ ซึ่งทำหน้าที่แปลง ซอร์สโค้ด ให้เป็นเนทีฟโค้ด(native code) ซึ่งจะอยู่ในลักษณะ โค้ดแบบ

ขนาน (Parallel code)เพื่อจะนำมาใช้กับแพลตฟอร์มที่เป็นแบบขนานได้ โดยในทุกขั้นตอนจะมีส่วนของรันไทม์ และ ไลบรารี เพื่อใช้ในการเรียกโหลดรูทีน ที่ได้อ้างอิงไว้ในตัวโค้ด



รูปภาพที่ 3.2.1.1 แสดงโครงสร้างของการเขียนโปรแกรมแบบขนาน

3.2.2 แนวทางการทำ Parallel Programming

ปัจจุบันมีรูปแบบโมเดลของการเขียนโปรแกรมแบบขนานอยู่ 4 รูปแบบที่นิยมใช้กัน คือ

1. Implicit
2. Data parallel
3. Message-passing
4. Shared variable

โดยโมเดลทั้งหลายนี้มักจะใช้กับภาษาฟอร์แทรนหรือซี เป็นส่วนขยายรูปแบบ โดยจะแบ่งได้เป็น 3 แนวทาง คือ

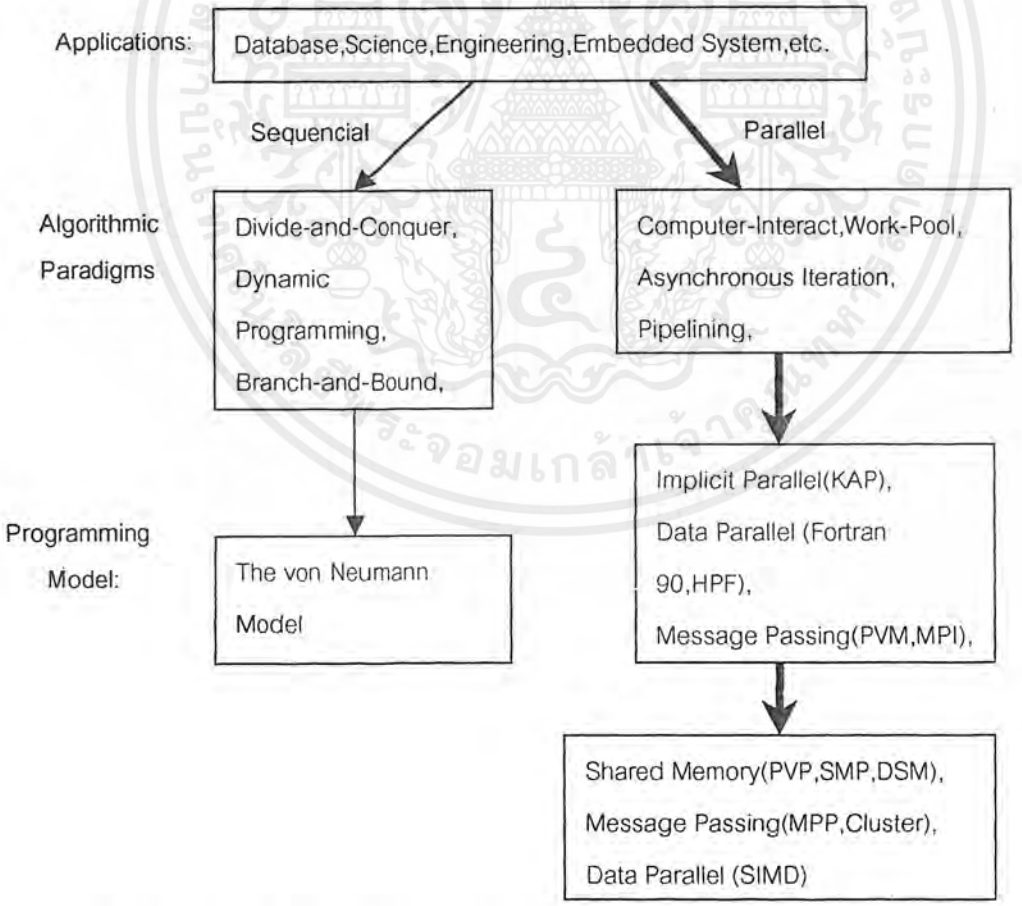
1. ไลบรารีซับรูทีน นอกจากไลบรารีของโปรแกรมประเภท ซีเควนเชียลโปรแกรม แล้วยังมีการเพิ่ม ไลบรารี ที่ทำให้สามารถประมวลผลคำสั่งแบบขนาน ยกตัวอย่างเช่น MPI ,PVM ,Cray Craft โดยข้อดีของแนวทางประเภทนี้คือ ง่ายในการ ใช้งานโปรแกรม และไม่จำเป็น

ต้องมีการ อิมพลีเมนต์คอมไพเลอร์ใหม่ แต่มีข้อเสียคือ ไม่มีตรงส่วนของการตรวจสอบเวลาในการคอมไพล์ การตรวจสอบความถูกต้อง หรือการออปติไมซ์ ซอร์สโค้ด

2. นิวคอนสตรัค แนวทางนี้จะต่างจากแนวทางแรกคือจะสร้างส่วนคอมไพเลอร์ขึ้นใหม่ให้มีการสนับสนุนการประมวลผลแบบขนาน โดยวิธีนี้มีข้อดีตรงที่เราจะได้ส่วนของการตรวจสอบความถูกต้องจากคอมไพเลอร์ การตรวจสอบเวลาการคอมไพล์ หรือการออปติไมซ์ ซอร์สโค้ด แต่มีข้อเสียที่จะต้องมีการสร้างคอมไพเลอร์ขึ้นใหม่ ทำให้ยากในการใช้โปรแกรมคอมไพเลอร์ใหม่
3. ไดเรกทีฟจะอยู่ระหว่างกลางของทั้งสองแนวทางที่กล่าวมา คือสามารถทำให้ใช้ซีเควนเชียลโปรแกรมในรูปแบบที่กำหนดเพื่อให้กลายเป็นโปรแกรมแบบขนานได้ โดยใช้ตัวไดเรกทีฟเข้ามาแทรกในส่วนซีเควนเชียลโปรแกรม โดยถ้านำเอาส่วนไดเรกทีฟออกก็สามารถใช้งานกับซีเควนเชียลแพลตฟอร์มอื่น ๆได้

3.3 การเปรียบเทียบระหว่างการเขียนโปรแกรมแบบซีเควนเชียลกับการเขียนโปรแกรมแบบขนาน

จากรูปที่ 3.3.1 จะเห็นได้ว่าการเขียนโปรแกรมแบบซีเควนเชียลจะมีโมเดลอยู่โมเดลเดียว แต่แบบขนานจะมีอยู่สองโมเดล



รูปภาพที่ 3.3.1 การเปรียบเทียบระหว่างการเขียนโปรแกรมแบบขนานและแบบซีเควนเชียล

จากตารางที่ 3.3.1 จะเป็นการเปรียบเทียบข้อดีข้อเสียของซีเควนเชียลโปรแกรมและโปรแกรมแบบขนาน

รูปแบบ	ข้อดี	ข้อเสีย
1.ซีเควนเชียลโปรแกรม	1.ผู้คนใช้งานเป็นจำนวนมาก เพราะคุ้นเคยกับการเขียนในรูปแบบนี้มาก่อน	1.การประมวลผลทำได้ช้ากว่าโปรแกรมแบบขนานบนเครื่องที่มีการทำงานเป็นแบบขนาน
	2.มีเครื่องมือ สนับสนุนอยู่เป็นจำนวนมาก	2. ทำให้ตัวโพรเซสเซอร์ต้องทำงานหนัก
	3. รูปแบบในการเขียนค่อนข้างง่าย	
2. โปรแกรมแบบขนาน	1. เมื่อมีการประมวลผลในงานที่ต้องใช้ความเร็วโพรเซสเซอร์สูง จะทำงานได้รวดเร็วกว่าซีเควนเชียลโปรแกรม	1.ผู้คนมีความคุ้นเคยต่อรูปแบบการเขียนประเภทนี้น้อย
	2. เป็นการใช้งานตัวโพรเซสเซอร์ได้อย่างมีประสิทธิภาพ	2.มีทูลสนับสนุนอยู่น้อยมาก
		3. รูปแบบการเขียนค่อนข้างซับซ้อน

ตารางที่ 3.3.1 แสดงข้อเปรียบเทียบระหว่างซีเควนเชียลโปรแกรมและโปรแกรมแบบขนาน

บทที่ 4

หลักการของ MPI

4.1 ความหมายของ MPI (Message Passing Interface)

MPI เป็นมาตรฐานในการประมวลผลแบบกลุ่มชนิดหนึ่ง โดยได้รับการพัฒนามาจากใน MPI Forum ซึ่งเป็นเหมือนการรวมกันต่าง ๆ ระหว่างผู้ขายคอมพิวเตอร์แบบขนาน ผู้เขียนไลบรารี และผู้เชี่ยวชาญด้านแอพพลิเคชัน (Application Specialist) โดย MPI ได้รับความสำเร็จจากการเป็นมาตรฐานสากล และการที่สามารถใช้ได้กับทุกแพลตฟอร์มอย่างอิสระ โดยไลบรารี ของตัว MPI นี้จะใช้ภาษาฟอร์แทรน และ ซี เป็นภาษาหลัก เนื่องจากเหตุผลนี้ทำให้ MPI ได้รับการยอมรับจากกลุ่มของการประมวลผลแบบขนาน โดยได้ถูกประยุกต์ใช้บน ไอบีเอ็ม พีซี บน วินโดว์ และส่วนของยูนิกซ์ สเตชัน รวมทั้งคอมพิวเตอร์แบบขนานประเภทอื่น ๆ โดยข้อดีอีกอย่างหนึ่งของ MPI คือ เราสามารถที่จะเขียนโปรแกรมที่เป็นมาตรฐานแบบ ซี หรือ ฟอร์แทรน และใช้ไลบรารี MPI ในการส่ง ก็สามารถนำไปรันกับทุก ๆ แพลตฟอร์มของคอมพิวเตอร์ ไม่ว่าจะเป็นคอมพิวเตอร์เครื่องเดี่ยว ๆ เครื่องเวิร์คสเตชัน หรือเป็นเครือข่ายของเครื่องเวิร์คสเตชัน โดยโปรแกรม MPI จะทำการจัดการทั้งเรื่องการติดต่อสื่อสาร การสร้างโพรเซสในการคำนวณขึ้นมา ให้เราทั้งหมดทำให้การเขียนโปรแกรมง่ายขึ้น

นอกจาก MPI แล้วยังมีไลบรารีที่สำคัญคือ PVM ซึ่งเป็นโปรแกรมแบบขนานเช่นเดียวกัน โดยได้มีการพัฒนามาก่อนไลบรารี MPI แต่ปัจจุบันนี้คนนิยมใช้ไลบรารีของ MPI มากกว่าเนื่องจากมีฟังก์ชันการทำงานที่มากกว่า PVM และใช้ง่ายกว่าด้วย

4.2 องค์ประกอบต่าง ๆ ของ MPI

4.2.1 MPI โพรเซส

MPI โพรเซสเป็นส่วนของเอ็นติตี้ที่ทำงานเป็นคอมพิวเตอร์ทากส์ ถ้าพิจารณา MPI ในรูปโปรแกรมมิ่งโมเดล แล้วจะเห็นว่าแต่ละ โพรเซส จะเชื่อมโยงกันโดยแต่ละโพรเซสจะมีหน่วยความจำของแต่ละตัวเรียกว่า หน่วยความจำโลคอล ในโมเดลจะให้โพรเซสเชื่อมโยงกันด้วยลักษณะหน่วยความจำที่คล้าย ๆ กันเป็นการทำงานลักษณะ Multiple Precesses Multiple Data (MPMD)

โพรเซสใน MPI จะสื่อสารกันด้วย แมสเซจ ซึ่งแมสเซจจะประกอบไปด้วยส่วนของข้อมูลเรียกว่า body ประกอบกับข้อมูลที่เพิ่มเข้าไปบางข้อมูลเรียกว่าข้อมูลหน้าซอง (Envelop) โดยข้อมูลหน้าซองจะเป็นส่วนที่ชี้เฉพาะให้ โพรเซสทราบถึงความแตกต่างกันระหว่างแต่ละแมสเซจ และผู้รับข้อมูลสามารถตัดสินใจและเลือกรับข้อมูลได้อย่างถูกต้อง

การแลกเปลี่ยนแมสเซจระหว่างโพรเซส ใน MPI จะหมายถึงการสื่อสาร (communication) กล่าวคือ MPI โพรเซสจะรับข้อมูลไปประมวลผลในหน่วยความจำโลคอลหรือทำการซิงโครนัสกับโพรเซส อื่น โดยการสื่อสารจะส่งการเรียกในรูปฟังก์ชันการทำงานของภาษาซี หรือฟอร์แทรน

4.2.2 กลุ่มโพรเซส (process group)

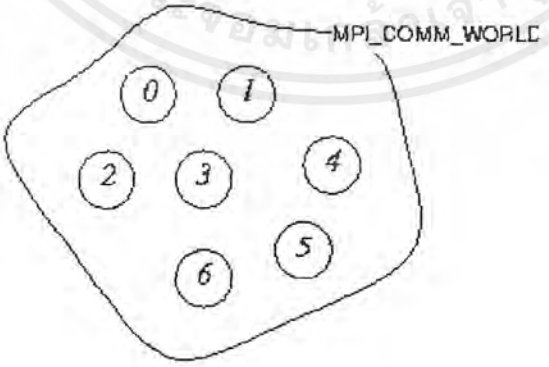
กลุ่มโพรเซสใน MPI เป็นเซตของโพรเซสซึ่งเป็นกลุ่มอยู่และบ่งบอกถึงแต่ละโพรเซสด้วยเลขเรียงกันตั้งแต่ 0 เป็นต้นไป โดยจำนวนของโพรเซสที่ต่อเชื่อมกันในกลุ่ม จะเรียกว่าแรงค์ ของกลุ่ม ลักษณะการทำงานของโพรเซสในกลุ่มจะทำงานร่วมกัน ในแต่ละงานจะมีโพรเซสที่ทำงานอยู่เป็นจำนวนแน่นอน และไม่สามารถเพิ่มจำนวนของโพรเซสระหว่างทำการรันโปรแกรมได้ แต่ MPI สามารถขยายแรงค์ของกลุ่มได้

4.2.3 คอมมิวนิเคชัน คอนเท็ก

ในแต่ละโพรเซส MPI จะสามารถตัดสินใจเลือกที่จะรับหรือไม่รับแมสเซจได้ขึ้นอยู่กับข้อมูลหน้าซอง ของแมสเซจนั้น ๆ ซึ่งข้อผิดพลาดทั่ว ๆ ไปของการส่งแมสเซส คือ โพรเซส ได้รับแมสเซจที่ส่งมาโดยผิด โพรเซส ในจุดนี้เราสามารถกำหนดค่าโดยตรงให้โดยผู้ใช้ หรือจะใช้การเรียกใช้คอนเท็ก ในแมสเซจ พาสซิ่ง อินเทอร์เฟซ จะสนับสนุนการใช้แมสเซจคอนเท็ก ที่มาช่วยให้การส่ง แมสเซจไม่เกิดความเสี่ยงและความสับสนขึ้น

4.2.4 คอมมิวนิเคเตอร์

คอมมิวนิเคเตอร์จะถูกสร้างมาเชื่อมโยงแต่ละ MPI โพรเซส ในแต่ละกลุ่มโพรเซส จากแนวคิดของ เวอร์ชวล ทอโปโลจี โดยจะทำหน้าที่จัดการกับการสื่อสาร ทุก ๆ การสื่อสารของ MPI ต้องเรียกผ่านคอมมิวนิเคเตอร์และ โพรเซส ที่จะทำการสื่อสารกันได้นั้นต้องมีการแชร์ตัวกันเสียก่อน ซึ่งในแต่ละคอมมิวนิเคเตอร์จะประกอบด้วยลิสท์ของ โพรเซส ในกลุ่ม

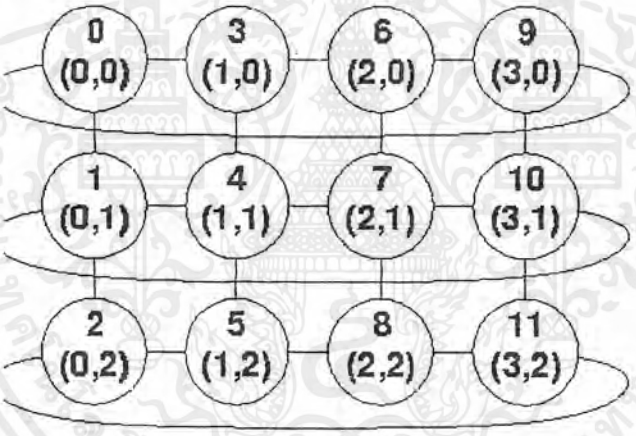


รูปภาพที่ 4.2.3.1 แสดงลักษณะของคอมมิวนิเคเตอร์

4.2.5 เวอร์ชวล ทอโปโลจี

เวอร์ชวล ทอโปโลจี เป็นกลไกสำหรับการตั้งชื่อของโพรเซสในกลุ่ม หรือในคอมมิวนิเคเตอร์เดียวกัน เพื่อกำหนดรูปแบบ (pattern) ของการสื่อสารกัน ถ้าในสภาพปกติแล้วการเข้าถึง โพรเซส ที่ถูกเชื่อมโยงในกลุ่ม จะดูไม่เป็นธรรมชาติในการทำงาน แต่จะเป็นการง่ายกว่าถ้าจะมองโพรเซสเป็นรูปอินเด็กซ์หรือตาม ค่าลำดับคาร์ทีเซียน(Cartesian coordinate) เรียกว่าคาร์ทีเซียน ทอโปโลจี และโดยทั่วไปแล้ว MPI จะทำการใช้ลักษณะของกราฟทอโปโลจี มาใช้เพื่อแสดงและหาเส้นทางใกล้สุดระหว่าง โพรเซส ด้วยกันเพื่อทำการสื่อสาร โดยไม่ต้องทำการรัน เพื่อหาเส้นทางจากโพรเซสเซอร์ดังนั้น MPI จึงใช้คาร์ทีเซียน และกราฟ ทอโปโลจี มาใช้เชื่อมโยง แทนการเชื่อมโยงปกติและเรียกรวมว่าเวอร์ชวล ทอโปโลจีเข้ามาใช้งาน โดยจะนำมาใช้ตอนรันไทม์ระบบเพื่อทำออปติไมซ์ เส้นทางการสื่อสาร หรือใช้เพื่อแจ้งให้โหลดเดอร์ ทราบว่าควรจะคอนฟิกโพรเซส อย่างไร

เวอร์ชวล ทอโปโลจี จะเชื่อมเข้ากับคอมมิวนิเคเตอร์ โดยเมื่อเวอร์ชวล ทอโปโลจีถูกสร้างขึ้นบนคอมมิวนิเคเตอร์ ที่มีอยู่แล้ว จะเกิดการสร้างคอมมิวนิเคเตอร์ตัวใหม่ขึ้นมาโดยอัตโนมัติ แล้วส่งกลับไปยังผู้ใช้ และให้ผู้ใช้ทำงานกับคอมมิวนิเคเตอร์ตัวนี้แทน

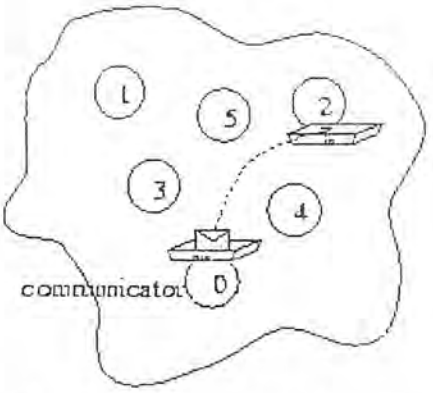


รูปภาพที่ 4.2.4.1 แสดงลักษณะเวอร์ชวล ทอโปโลจี

4.3 การสื่อสาร

การทำงานของ MPI จะอยู่บนพื้นฐานอยู่บนการสื่อสารกันระหว่างโพรเซส โดยใช้การส่งแมสเซจไปทำการประมวลผลในโหนดต่าง ๆ มีลักษณะการสื่อสารกันหลายลักษณะเป็นบล็อกกิ้ง เวอร์ชัน และ นอน-บล็อกกิ้ง เวอร์ชัน ในรูปแบบบล็อกกิ้ง เวอร์ชันจะมีการสร้างรูทีนในการรับส่งขึ้นและทำการส่งค่ากลับจากรูทีน เมื่อเสร็จสิ้นแล้ว ส่วนในแบบนอน-บล็อกกิ้ง เวอร์ชันจะมีการสร้างรูทีนขึ้นมาเช่นกันแต่จะมีการส่งค่ากลับมาก่อนที่การสื่อสารจะเสร็จสิ้น โดยโพรเซสที่ทำการส่งและรับจะมีการสร้างบัฟเฟอร์ขึ้น ก่อนที่โพรเซสผู้ส่งจะทำการอัพเดท บัฟเฟอร์ได้ต้องตรวจสอบรูทีน ก่อนว่าการสื่อสารเสร็จสมบูรณ์ขึ้นจริงหรือยังโดยสามารถเรียกรูทีนทำการทดสอบได้ ส่วนโพรเซสผู้รับจะสามารถทำงานอื่น

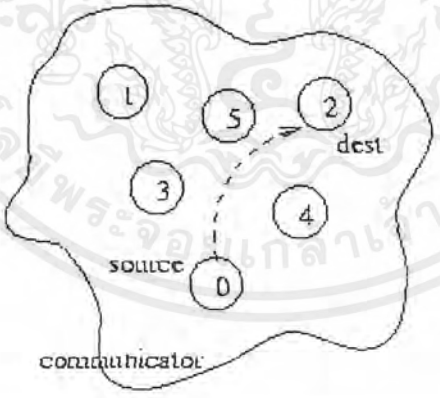
หรือประมวลผลอย่างอื่นก่อนได้จนกว่าจะถึงเวลาที่ต้องการใช้ข้อมูลก็จะไปตรวจสอบจากบัฟเฟอร์ว่ามีข้อมูลอยู่หรือไม่



รูปภาพที่ 4.3.1 การสื่อสารแบบ นอน-บล็อกกิ้ง เวอร์ชัน

4.3.1 การสื่อสารแบบจุดต่อจุด(Point-to-Point Communication)

เป็นการสื่อสารที่มีโพรเซสเข้ามาเกี่ยวข้องกัน2โพรเซสเท่านั้นโดยโพรเซสหนึ่งจะส่งแมสเซจและอีกโพรเซสหนึ่งจะเป็นผู้รับแมสเซจ โดยตัวส่งจะใช้การเรียก MPI Call ไปยังปลายทางในแรงค์ ของคอมมิวนิเคเตอร์ ที่เหมาะสม สำหรับแมสเซจที่ส่งในตัวของแมสเซจจะประกอบไปด้วย แอดเดรสเริ่มต้นของข้อมูล, ขนาดข้อมูล, ประเภทข้อมูล, จุดหมาย และ อาร์กิวเมนท์ ของคอมมิวนิเคเตอร์หลักการในการส่งจะเรียกว่า "pushing" คือการส่งแมสเซจไปยังโพรเซสต่าง ๆ โดยตัวโพรเซสไม่สามารถ "fetch" แมสเซจเองได้ ทำได้แค่เพียงรอรับแมสเซจที่ส่งมาแล้วเท่านั้น โดยในแบบจุดต่อจุดจะสามารถทำการสื่อสารได้ 4 รูปแบบ

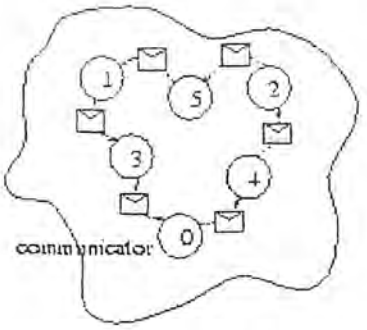


รูปภาพที่ 4.3.1.1 การสื่อสารแบบจุดต่อจุด

1 การส่งแบบซิงโครนัส(Synchronous send)

การส่งแมสเซจในแบบนี้จะต้องรู้ว่าผู้รับได้รับแมสเซจแล้วโดยตัวโพรเซสผู้รับจะส่งสัญญาณ ตอบกลับมาเพื่อบอกว่าการสื่อสารสมบูรณ์แล้วการส่งในโหมดนี้จะช่วยให้ทำการดีบักได้ง่ายและจำไม่ทำให้เกิดการโอเวอร์โหลดและส่งแมสเซจผิดได้ เนื่องจากต้องมีการซิงโครนัสกันระหว่างผู้ส่งกับผู้รับ แต่ถ้าใช้ร่วมกับ บล็อกกิ้ง

จะทำให้เกิดปัญหาในลักษณะเดดล็อกได้คือผู้ส่งและผู้รับจะมีการซิงโครนัสต่อกันไปเรื่อยๆทำให้มีการรอแมสเซจในแต่ละโพรเซส สามารถแก้ไขได้โดยใช้นอน-บล็อกกิ้งแทน



รูปภาพที่ 4.3.1.2 การเกิดเดดล็อกในการส่งแมสเซจ

2 การส่งแบบใช้บัฟเฟอร์(Buffer send)

จะมีการรับประกันการส่งแมสเซจทันทีที่ทำการส่ง โดยจะใช้วิธีการทำสำเนาแมสเซจไว้ในระบบบัฟเฟอร์หลังจากทำการส่ง บัฟเฟอร์ หลังจากทำการส่ง วิธีนี้จะรับประกันการทำงานโดยไม่ต้องมีการซิงโครนัสกันระหว่างโพรเซสทำให้สามารถทำงานได้เร็วและสะดวกขึ้น แต่จะสามารถเกิดการโอเวอร์โหลดของเครือข่ายขึ้นได้จากการส่งมาก ๆ ข้อเสียของการส่งแบบบัฟเฟอร์คือโปรแกรมเมอร์จะไม่สามารถกำหนดขนาดบัฟเฟอร์ได้แน่นอนและจะต้องมีบัฟเฟอร์ที่มีขนาดเพียงพอกับโปรแกรมด้วย

3 การส่งแบบพร้อมทำงาน (Ready send)

มีลักษณะคล้ายกับการส่งแบบบัฟเฟอร์ คือจะรับประกันการส่งแมสเซจทันที การสื่อสารแบบนี้จะรับประกันความถูกต้อง ถ้ามีผู้รับที่ถูกต้องมารอรับ โดยโพรเซสผู้ส่งจะส่งแมสเซจไปยังเครือข่ายการสื่อสารและหวังว่าผู้รับจะรออยู่เพื่อรับแมสเซจ ถ้าโพรเซสผู้รับพร้อมที่จะรับแมสเซส ก็จะทำการรับแต่ถ้าไม่เป็นเช่นนั้นแมสเซจจะเงียบหายไปและเกิดความผิดพลาด

แนวคิดนี้จะทำการหลีกเลี่ยงการทำแฮนเชคและบัฟเฟอร์ริ่งระหว่างผู้ส่งและผู้รับในโหมดนี้ยากในการทำ ดีบัก และต้องการการดูแลในการเขียนโปรแกรม

4 การส่งแบบมาตรฐาน(Standard send)

โหมดนี้จะสามารถเลือกใช้ระหว่างซิงโครนัสและบัฟเฟอร์ คือการเสร็จการส่งอาจจะก่อนที่จะมีการตรวจรับจากผู้รับ หรือไม่ก็ได้ โดยในการส่งแบบมาตรฐานจะมีหลักว่า

- ไม่ควรตั้งสมมติฐานว่าการส่งแมสเซจเสร็จก่อนที่ผู้รับจะเริ่มทำการรับ เพราะถ้าใช้การ บล็อกกิ้ง อาจเป็นสาเหตุให้เกิดเดดล็อกได้
- ไม่ควรตั้งสมมติฐานว่าการส่งเสร็จสิ้นเมื่อผู้รับเริ่มทำการรับ

กล่าวโดยสรุปคือการส่งแบบมาตรฐานอาจจะนำวิธีซิงโครนัสหรือวิธีบัฟเฟอร์มาใช้ก็ได้และผู้ใช้ไม่ควรตั้งสมมติฐานทั้ง 2 กรณีเพราะไม่สามารถรู้ได้

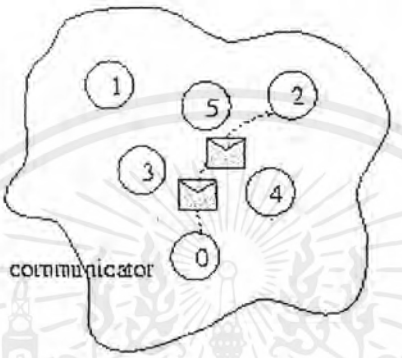
การรับส่งแมสเซจผู้รับจะพิจารณาจากข้อมูลหน้าซองซึ่งเป็นส่วนที่แสดงความแตกต่างระหว่างแมสเซจและในส่วนแทกบนแมสเซจจะเป็นตัวบ่งบอกถึงประเภทของแมสเซจต่าง ๆ กันเช่น ข้อมูลเริ่มต้น

(initial data), ลักษณะการร้องขอจากโพรเซสอื่น ๆ ในส่วนแทกนี้จะทำให้ผู้รับสามารถเลือกแมสเซจที่ต้องการรับได้อย่างถูกต้อง

การสื่อสารแบบจุดต่อจุดควรจะรับประกันสิ่งเหล่านี้

1. ความถูกต้องของลำดับการส่ง(Message order prevention)

แมสเซจจะต้องไม่มีการแซงกันขึ้นในการส่ง ต้องเป็นไปตามลำดับ เช่น โพรเซส A ส่ง 2 แมสเซจไปยัง โพรเซส B แล้วโพรเซส B ทำการโพสเพื่อรับทั้งสองแมสเซจต้องมีการรับประกันว่าการรับจะเป็นไปตามลำดับที่ทำการส่งมา



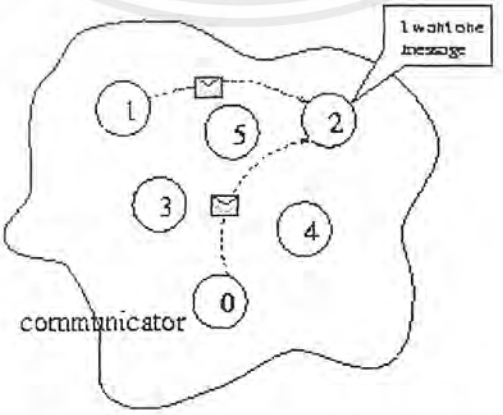
รูปภาพที่ 4.3.1.3 แสดงภาพ การรับ-ส่งแมสเซจตามลำดับ

2. ความคืบหน้าในการส่ง(Progress)

จะไม่มีการเหลือโพรเซสที่กันในการรับส่งนอกจาก 2 โพรเซสที่เกี่ยวข้องกันเท่านั้น การรับส่งจะเสร็จสมบูรณ์อย่างถูกต้องเมื่อโพรเซสที่ 1 ส่ง แมสเซจไปยังโพรเซสที่ 2 ที่มีคุณสมบัติตรงกับแมสเซจที่ส่งไป

แต่อาจมี 2 ลักษณะที่เป็นไปได้และไม่ควรเกิดขึ้น

- แมสเซจที่ส่งถูกรับไปโดยโพรเซสที่3และตรงกับการรับแมสเซจที่ส่งมาทำให้การส่งสมบูรณ์แต่โพรเซสที่ 2 ไม่ได้รับ
- แมสเซจโพรเซสที่ 3 ส่ง แมสเซจออกไปและได้รับโดยโพรเซสที่ 2 ซึ่งการสื่อสารก็จะสมบูรณ์ แต่โพรเซสที่ 1 ไม่ได้ทำการส่ง



รูปภาพที่ 4.3.1.4 แสดงการผิดพลาดในการรับ-ส่ง

4.3.2 การสื่อสารแบบกลุ่ม(Collective communication)

จะเป็นการสื่อสารที่ทุก ๆ โพรเซสภายในคอมมิวนิเคเตอร์(คือ ทุก ๆ โพรเซสในกลุ่ม ที่เชื่อมโยงกันกับคอมมิวนิเคเตอร์) และถ้าหากมีการใช้การสื่อสารแบบกลุ่มในส่วนสับเซตของคอมมิวนิเคเตอร์จะต้องมีการสร้างคอมมิวนิเคเตอร์ใหม่ขึ้น

คุณลักษณะของการสื่อสารแบบกลุ่ม

- การสื่อสารแบบกลุ่มจะไม่มีการเข้าไปเกี่ยวข้องกับการสื่อสารแบบจุดต่อจุด โดยจะแยกส่วนกันอย่างชัดเจนตัวอย่างเช่น การสื่อสารแบบกลุ่มจะไม่สามารถส่งแมสเซจและรับโดยจะไม่สามารถส่งแมสเซจและรับโดยผู้รับที่ใช้การสื่อสารแบบจุดต่อจุดได้ แต่ใน คอมมิวนิเคเตอร์หนึ่งที่มี 2 คอมมิวนิเคเตอร์ย่อย ทั้ง 2 ตัวจะสามารถแยก เป็นการสื่อสารแบบจุดต่อจุดกับการสื่อสารแบบกลุ่มได้
- การสื่อสารแบบกลุ่มจะใช้การซิงโครนัสหรือไม่ก็ได้
- ในการสื่อสารแบบกลุ่มจะมีลักษณะเป็นบล็อกกิ้ง เวอร์ชันได้เท่านั้น
- ทุก ๆ โพรเซสในคอมมิวนิเคเตอร์ต้องเรียกใช้การสื่อสารแบบกลุ่ม

ลักษณะที่เหมือนกันกับการสื่อสารแบบจุดต่อจุด

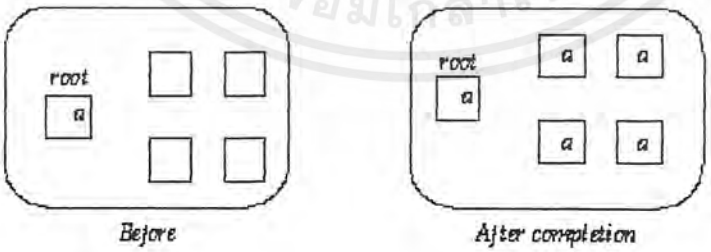
- แมสเซจเป็นประเภทข้อมูลของอาร์เรย์ ชนิดหนึ่ง ๆ
- ประเภทข้อมูลจะต้อง มีคุณสมบัติตรงกันระหว่างผู้ส่งและผู้รับ

ลักษณะที่ต่างกันกับการสื่อสารแบบจุดต่อจุด

- ไม่มีหลักการเกี่ยวกับแทกของแมสเซจ
- แมสเซจที่ถูกส่งมาจะต้องถูกใส่ลงในบัฟเฟอร์ของผู้รับนั้น ๆ

การส่งในแบบกลุ่มมีหลายลักษณะ

แบบบรอดคาสจะทำการกระจายแมสเซจที่เหมือนกันจากโพรเซสหนึ่งที่เรียกว่า โพรเซสรูท ไปยังโพรเซสอื่น ๆ ที่เชื่อมกับคอมมิวนิเคเตอร์

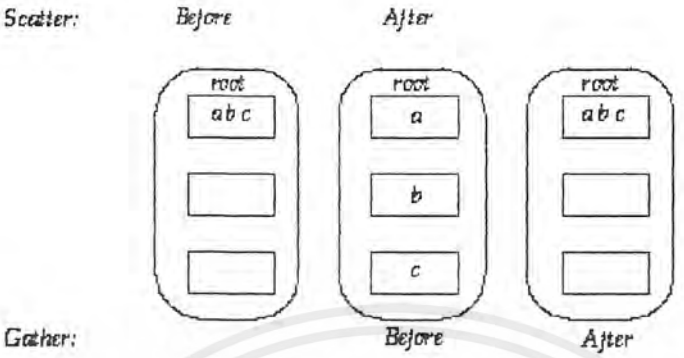


รูปภาพที่ 4.3.2.1 แสดงลักษณะการบรอดคาส

สแคทเทอร์และเกเทอร์ ทั้งสแคทเทอร์ และเกเทอร์จะเป็นโอเปอเรชันที่มีลักษณะตรงข้ามกัน

- สแคทเทอร์ : โพรเซสรูทจะส่งแมสเซจที่ต่างกันที่มีลักษณะของข้อมูล เหมือนกันไปยังแต่ละโพรเซส

- เกเทอร์ : โพรเซสรูทจะรับแมสเซจที่ต่างกันจากแต่ละโพรเซส มาทำการต่อกันเป็นแมสเซจในรูปประเภทข้อมูลเดียวกัน

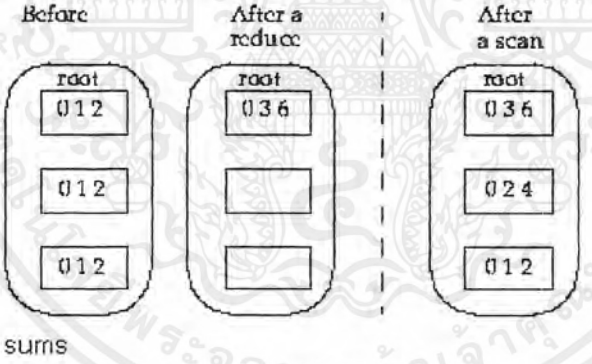


รูปภาพที่ 4.3.2.2 แสดงสแคทเทอร์และเกเทอร์

รีดิวซ์ และสแกน : ทุก ๆ อีลีเมนท์ในตำแหน่งเดียวกันจะเป็นอินพุทจากทุก ๆ โพรเซสที่จะทำการรีดิวซ์มารวมกัน โดยผลลัพธ์จากการรีดิวซ์จะเก็บเป็นผลลัพธ์ในโพรเซสรูท

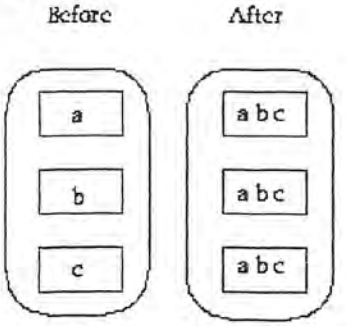
- รีดักชันโอเปอเรชัน จะทำการรวมหรือเชื่อมต่อแมสเซจ คล้ายกับการเกเทอร์

- สแกนโอเปอเรชัน จะทำแบ่งส่วน(Partial)ผลลัพธ์ไปเก็บในทุก ๆ โพรเซสในรูปลำดับบางอย่างโดยการสแกน จะใช้ประโยชน์ในการสร้างค่าลักษณะเฉพาะ(unique) สำหรับแต่ละโพรเซส



รูปภาพที่ 4.3.2.3 แสดงรีดิวซ์และสแกน

เกเทอร์-ทู-ออล จะทำการเกเทอร์แมสเซจมาไว้ที่โพรเซสรูท และตามด้วยการทำบรอดคาสข้อมูลที่ได้จากการเกเทอร์ไปยังทุกโพรเซส



รูปภาพที่ 4.3.2.4 แสดง เกเทอร์-ทู-ออล

4.4 ความรู้พื้นฐานในการเขียนโปรแกรม Mpi

โครงสร้างหลักของการเขียน โปรแกรม Mpi คือ

```
#include <Mpi.h>
void main(int argc,char *argv[]) {
 int  Rank,Size;
 MPI_Init(&argc,&argv);
MPI_Comm_Rank(MPI_COMM_WORLD,Rank);
MPI_Comm_Size(MPI_COMM_WORLD,Size);
/* Code การทำงาน */
MPI_Finalize();
}
```

ประเภทข้อมูลของ MPI

ประเภทข้อมูลของ MPI นำพื้นฐานของประเภทข้อมูลจากภาษาซี และ ฟอร์แทรน มาใช้โดยในมาตรฐานของ MPI นั้นจะใช้ ประเภทข้อมูลของ MPI แสดงอยู่บนประเภทข้อมูลของภาษาซี และฟอร์แทรน ซึ่งประเภทข้อมูลรูปแบบง่าย ๆ คือส่วนประเภทข้อมูลอะทอมิค ของภาษาซี และ ฟอร์แทรนนำหน้าด้วยประเภทข้อมูลของ MPI

MPI Datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

ตารางที่ 4.4.1 แสดงประเภทข้อมูลของ MPI ในภาษาซี

ฟังก์ชันที่สำคัญใน MPI

1. MPI_Init(&argc,&argv) เป็นฟังก์ชันที่ใช้ในการเริ่มต้นการใช้งาน MPI
2. MPI_COMM_RANK(comm,rank) เป็นฟังก์ชันที่ใช้หาว่าโหนดที่ทำงานอยู่เป็นโหนดใด เป็นมาสเตอร์หรือสเลฟ
3. MPI_COMM_SIZE(comm,size) เป็นฟังก์ชันที่ใช้ในการหาจำนวนโพรเซสที่จะใช้ในการประมวลผล
4. MPI_SEND(buf,count,datatype,dest,tag,comm) ซึ่งค่าในอาร์กิวเมนท์ต่าง ๆ อธิบายได้ดังนี้

   buf เป็นแอดเดรสเริ่มต้นของบัฟเฟอร์ผู้ส่ง

   count เป็นจำนวนที่ต้องการจะส่ง

   datatype เป็นชนิดของข้อมูลที่จะส่ง

   dest เป็นโหนดที่เป็นผู้รับ

   tag เป็นแมสเซจแทกที่ใช้ในการส่ง

   comm เป็นการบอกตัวคอมมิวนิเคเตอร์ ซึ่งส่วนใหญ่จะใช้ MPI_COMM_WORLD
5. MPI_RECV(buf,count,datatype,source,tag,comm,status)ซึ่งส่วนใหญ่จะเหมือนฟังก์ชัน MPI_SEND ต่างกันตรงที่บัฟเฟอร์เป็นของผู้รับ ,source จากผู้ส่ง และ status ที่บอกสถานะในการส่ง
6. MPI_Finalize() เป็นฟังก์ชันสำหรับจบการทำงาน MPI

ฟังก์ชันในส่วนของMPI_SEND กับ MPI_RECV จะเป็นการสื่อสารแบบจุดต่อจุด แต่นอกจากนี้ยังมีฟังก์ชันที่ทำการสื่อสารแบบกลุ่มด้วย ซึ่งประกอบด้วยฟังก์ชันดังต่อไปนี้

1. MPI_Reduce(void *data,void *result,int count,MPI_Datatype datatype,MPI_op,int target MPI_Comm comm) โดยเป็นฟังก์ชันที่นำผลรวมของทุกโพรเซสเซอร์ ในคอมมิวนิเคเตอร์มาทำการประมวลผลตาม โอเปอเรชั่น ที่กำหนดโดย Mpi_op แล้วส่งผลไปยังส่วนที่ต้องการ โดยโอเปอเรชั่นที่สำคัญมีดังต่อไปนี้

MPI_MAX

MPI_MIN

MPI_SUM

MPI_PROD (Product)

MPI_LAND (Logical AND)

MPI_BAND (Bitwise AND)

MPI_LOR (Logical OR)

2. MPI_Bcast(void *message,int count,MPI_Datatype,int root,MPI_Comm comm) เป็นฟังก์ชันที่จะส่งค่าที่เหมือน ๆกันไปยังทุกๆโพรเซสภายในตัวคอมมิวนิเคเตอร์ ในขณะที่อยู่ในโพรเซสนั้น ๆที่กำหนด

บทที่ 5

แนวคิดในการออกแบบระบบคลัสเตอร์

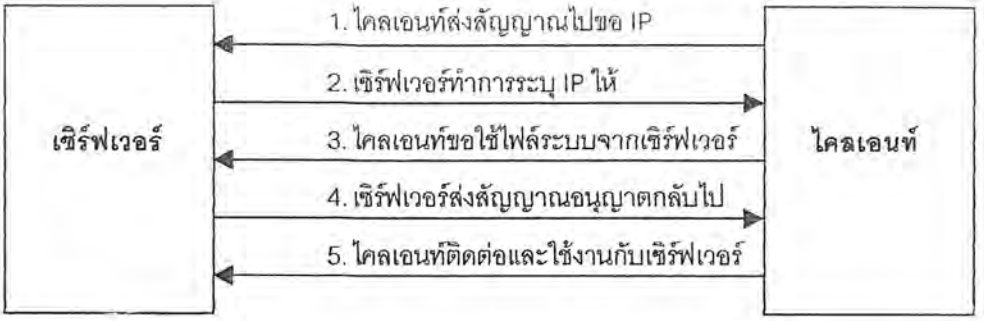
5.1 หลักการติดตั้งระบบคลัสเตอร์

ก่อนจะกล่าวถึงการเขียนโปรแกรม และการใช้งานโปรแกรมแบบขนานนั้น เราต้องมีระบบที่สามารถประมวลผลแบบขนานได้ นั่นคือระบบคลัสเตอร์นั่นเอง ลักษณะของระบบคลัสเตอร์ที่นิยมทำกันนั้นมี 2 ลักษณะด้วยกัน

- ลักษณะที่เครื่องไคลเอนท์จะมีฮาร์ดดิสก์ หรือเนื้อที่ในการเก็บไฟล์ระบบ (File system) รวมถึงเคอร์เนลของแต่ละเครื่องในตัวเอง ส่วนเครื่องเซิร์ฟเวอร์จะทำหน้าที่เชื่อมต่อและให้ไอพีแอดเดรสกับเครื่องไคลเอนต์เท่านั้น
- ลักษณะที่เครื่องไคลเอนท์จะไม่ใช้เนื้อที่ความจุในการเก็บไฟล์ระบบของตัวมันเลย แต่จะเรียกใช้ผ่านไปยังเซิร์ฟเวอร์ โดยในแบบนี้เซิร์ฟเวอร์จะทำตัวเองเป็น เน็ตเวิร์คไฟล์ซิสเต็มเซิร์ฟเวอร์ ทำการเก็บไฟล์ระบบของเครื่องไคลเอนท์ในแต่ละเครื่องไว้ และจะอนุญาตให้เครื่องไคลเอนท์เข้ามาใช้เนื้อที่นั้นได้ ส่วนใหญ่แล้วระบบแบบนี้มักจะเรียกว่าดิสก์เลสคลัสเตอร์

ซึ่งทั้ง 2 แบบนี้จะมีข้อดีและข้อเสียต่างกัน โดยในแบบแรกจะมีข้อดีตรงที่สามารถทำงานได้ประสิทธิภาพมากกว่า มีการติดต่อกันในระบบเครือข่ายน้อยกว่า ในระบบดิสก์เลสคลัสเตอร์ เพราะจะสามารถเรียกใช้ไฟล์ระบบได้จากตัวเอง ไม่ต้องเสียเวลาในการเรียกผ่านเครือข่าย และลดปัญหาการติดขัดในการใช้ช่องสัญญาณของระบบเครือข่าย ส่วนในระบบดิสก์เลสนั้นจะมีการติดต่อแลกเปลี่ยนข้อมูลกันไปมาบ่อยกว่าในแบบแรก ทำให้มีการใช้งานในระบบเครือข่ายสูงและอาจจะเกิดการติดขัดได้ และจะเสียเวลาในการทำงานจากส่วนของการติดต่อกันไปมานี้เอง แต่จะช่วยในการประหยัดค่าใช้จ่าย เพราะเครื่องไคลเอนท์ไม่จำเป็นต้องมีฮาร์ดดิสก์เพื่อเก็บไฟล์ระบบ

โดยส่วนใหญ่แล้วระบบดิสก์เลสคลัสเตอร์จะเป็นที่นิยมใช้กันมากกว่าเนื่องจากเป็นระบบที่ประหยัดและสามารถ นำเครื่องเก่าที่ไม่ได้ใช้งานมาใช้ทำได้ ดังนั้นในโครงงานชิ้นนี้จะทำระบบคลัสเตอร์โดยใช้วิธีการดิสก์เลสเพราะต้องการใช้แค่ความสามารถของโพรเซสเซอร์ในการช่วยกันประมวลผลแบบขนานและเพื่อความประหยัดในการนำไปใช้งานจริง หลักการทำงานและแนวทางของระบบดิสก์เลสคลัสเตอร์มีดังนี้



รูปภาพที่ 5.1.1 แสดงหลักการทำงานของดิสก์เลสคลัสเตอร์

1. ขั้นตอนแรกเครื่องไคลเอนท์จะทำการร้องขอไอพีแอดเดรสจากเซิร์ฟเวอร์เพื่อนำมาระบุใช้ในการติดต่อ โดยไคลเอนท์จะส่งค่าแมคแอดเดรส ของเน็ตเวิร์คการ์ดในเครื่องไปยังเซิร์ฟเวอร์
2. เมื่อเซิร์ฟเวอร์ได้รับสัญญาณการขอไอพีแอดเดรส พร้อมกับแมคแอดเดรสจากเครื่องไคลเอนท์มาแล้วจะนำไปตรวจสอบและหาไอพีแอดเดรสที่จะระบุให้กับไคลเอนท์ แล้วจึงแจ้งกลับไป
3. ในขั้นตอนต่อไปไคลเอนท์จะร้องขอไฟล์ระบบจากเครื่องเซิร์ฟเวอร์เพราะในระบบคลัสเตอร์นี้เป็นดิสก์เลส ทำให้ในตัวไคลเอนท์ไม่มีส่วนของไฟล์ระบบในตัวเองจึงจำเป็นต้องมีการขอใช้จากเซิร์ฟเวอร์
4. ในตัวเซิร์ฟเวอร์จะมีการสร้างไฟล์ที่สำคัญและจำเป็นต้องใช้ในแต่ละไคลเอนท์ทุก ๆ เครื่องเก็บไว้เพื่อที่จะให้เรียกใช้ได้ และในขณะนี้เครื่องเซิร์ฟเวอร์จะทำตัวเองเป็นเน็ตเวิร์คไฟล์ซิสเต็มเซิร์ฟเวอร์นั่นก็คือ เป็นเซิร์ฟเวอร์ที่ทำการแชร์ไฟล์ให้ใช้งานได้ผ่านทางระบบเครือข่าย ดังนั้นเมื่อเซิร์ฟเวอร์ได้สัญญาณร้องขอจากเครื่องไคลเอนท์แล้วก็ตรวจสอบและจะส่งสัญญาณตอบรับไปยังเครื่องไคลเอนท์
5. เมื่อเครื่องไคลเอนท์ได้รับสัญญาณส่งกลับมาแล้วก็จะสามารถเข้าไปใช้งานไฟล์ระบบในเครื่องเซิร์ฟเวอร์ในส่วนที่ตนเองมีสิทธิ์เข้าใช้ได้โดยผ่านทางเครือข่าย และจะต้องมีการส่งสัญญาณแลกข้อมูลต่าง ๆ กันระหว่างไคลเอนท์และเซิร์ฟเวอร์ตลอดเวลาที่ใช้งานอยู่

หลังจากขั้นตอนเหล่านี้แล้วระบบคลัสเตอร์ก็จะพร้อมสมบูรณ์ที่จะใช้งาน แต่การทำงานต่าง ๆ ก็ยังเป็นลักษณะซีเควนเชียลอยู่ เพราะระบบคลัสเตอร์นั้นเป็นเพียงส่วนที่สร้างขึ้นเพื่อรองรับการทำงานแบบขนานเท่านั้น แต่ถ้าโปรแกรมที่เรานำมาใช้งานยังไม่ได้ถูกสร้างหรือออกแบบมาในหลักการการประมวลผลแบบขนานระบบก็ยังไม่สามารถใช้งานแบบขนานได้

บทที่ 6

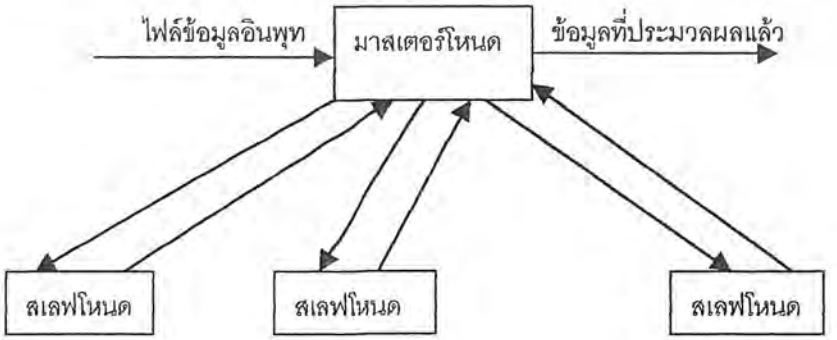
แนวคิดและอัลกอริทึมการทำงานของโปรแกรมแบบขนาน

ในส่วนของโปรแกรมที่ควรนำมาใช้งานกับระบบคลัสเตอร์นั้น ควรเป็นโปรแกรมที่ใช้กับงานที่มีการคำนวณอย่างหนักจากโพรเซสเซอร์ ซึ่งจะทำให้โพรเซสเซอร์สามารถช่วยกันประมวลผลได้ในลักษณะการประมวลผลแบบขนาน และทำให้มีประสิทธิภาพในการทำงานสูงขึ้น และที่สำคัญต้องออกแบบในหลักของการเขียนโปรแกรมแบบขนาน โดยทั่วไปแล้วการได้มาซึ่งโปรแกรมที่ทำงานในลักษณะขนานนั้นอาจจะได้มาจากการที่เขียนโปรแกรมและออกแบบโปรแกรมมาในลักษณะขนานเลย หรืออาจจะใช้การเขียนโปรแกรมในแบบซีเควนเชียลและใช้คอมไพเลอร์แบบขนาน ที่มีอยู่ทั่วไปมาทำการคอมไพล์โปรแกรมซีเควนเชียลนั้นก็จะได้โปรแกรมแบบขนานออกมาเช่นเดียวกัน

ในโครงงานนี้จะใช้การเขียนโปรแกรมในลักษณะแรกก็คือ การออกแบบและเขียนโปรแกรมในลักษณะขนานโดยตรงเพื่อที่จะได้เป็นแนวทางในการศึกษาและพัฒนาการเขียนโปรแกรมแบบขนานต่อไป ลักษณะนี้จะมีข้อดีคือเราสามารถกำหนดการทำงานต่าง ๆ ของโปรแกรมได้ด้วยตนเองทำให้สามารถออกแบบโปรแกรมได้อย่างเหมาะสมและมีประสิทธิภาพกับระบบเราอย่างเต็มที่ ในส่วนของการเขียนโปรแกรมจะใช้ไลบรารีของ MPI (Message Passing Interface) มาช่วยในการเขียนตัวโปรแกรมด้วย โดยโปรแกรมแบบขนานที่เขียนขึ้นนั้นมาจะใช้โปรแกรม MPI ในการรัน โดยโปรแกรม MPI จะทำการสร้างโพรเซสในการคำนวณขึ้นตามที่เรากำหนด และจัดการสร้างโพรเซสเซอร์ขึ้นมารวมทั้งหมดมาทำงานให้เองทำให้สะดวกสบายในการเขียนมาขึ้น

6.1 ลักษณะของโปรแกรม

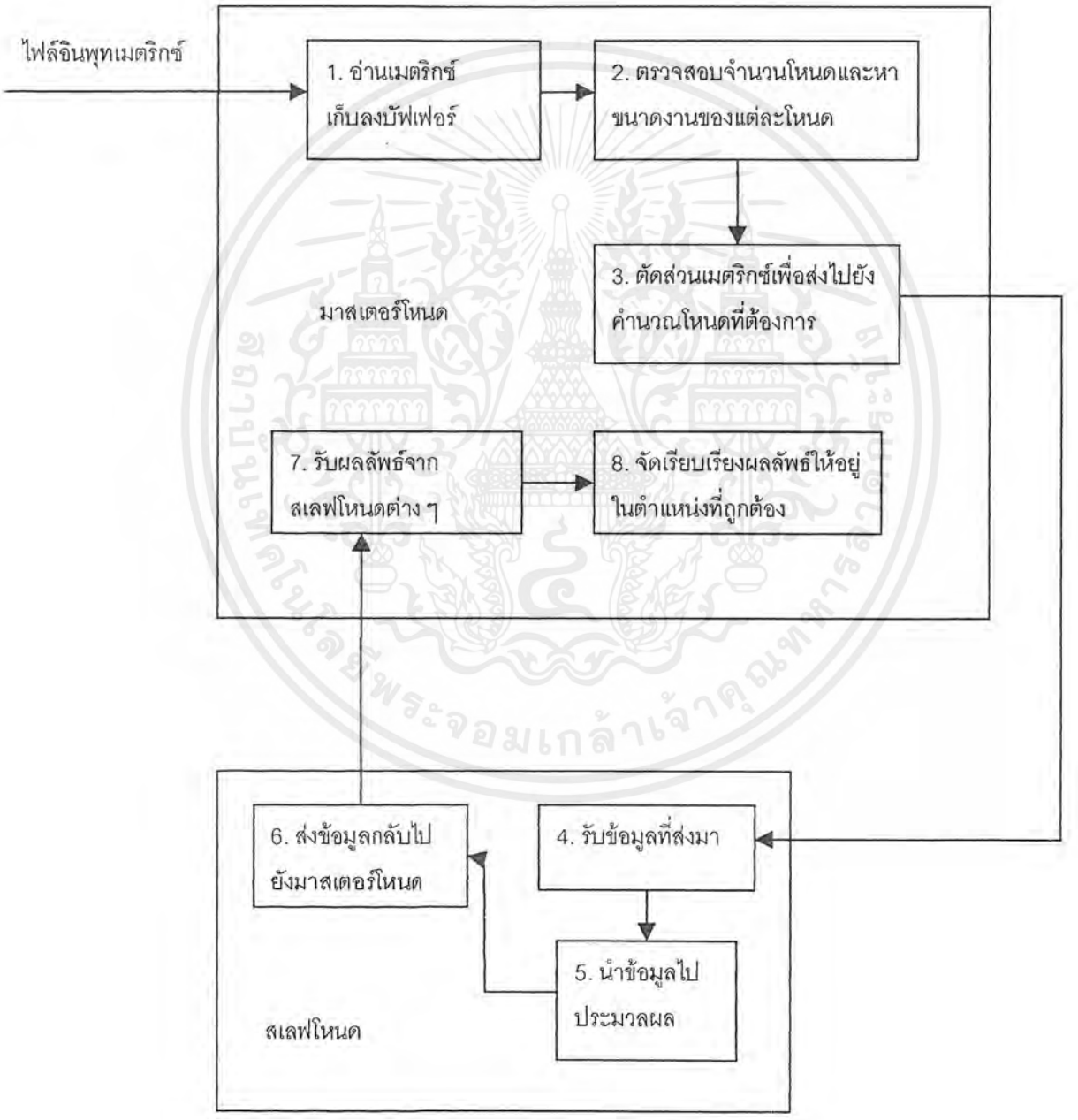
ลักษณะของโปรแกรมจะเป็นโปรแกรมการคำนวณทางเมตริกซ์ต่าง ๆ ได้แก่ บวก, ลบ, คูณ และดีเทอร์มินันท์เมตริกซ์ ซึ่งการคำนวณทางเมตริกซ์นี้เป็นที่นิยมนำมาใช้ในการทดสอบการประมวลผลแบบขนานของระบบคลัสเตอร์ เพื่อบ่งบอกประสิทธิภาพของระบบนั้น ๆ โปรแกรมจะรับค่าเมตริกซ์จากไฟล์ และนำไปประมวลผลแบบขนานต่อไป ดังรูปภาพที่ 6.2.1



รูปภาพที่ 6.2.1 แสดงหลักการออกแบบการทำงานของโปรแกรมแบบขนาน

6.2 อัลกอริทึม

โดยทั่วไปแล้วหลักการสำคัญของการเขียนโปรแกรมแบบขนานจะอยู่ที่การตัดข้อมูลและส่งไปทำการคำนวณยังโหนดต่าง ๆ หลักการส่วนใหญ่ของโปรแกรมนี้คือจะให้มาสเตอร์โหนดอ่านค่าเมตริกซ์จากไฟล์และนำไปเก็บใส่ไว้ในบัฟเฟอร์ จากนั้นก็จะตัดส่วนของเมตริกซ์บางส่วนแล้วส่งไปคำนวณที่โหนดต่าง ๆ เมื่อคำนวณเสร็จแล้วแต่ละโหนดก็จะส่งผลลัพธ์ที่ได้กลับมายังมาสเตอร์โหนดเพื่อทำการรวบรวมผลลัพธ์ให้ถูกต้องต่อไป มาสเตอร์โหนดจะทำหน้าที่คล้ายกับตัวควบคุมโหนดต่าง ๆ แบ่งงานไปประมวลผลยังโหนดต่าง ๆ และก็เก็บรวบรวมผลกลับมา ดังนั้นส่วนสำคัญ ๆ ก็จะอยู่ที่การโปรแกรมตัวมาสเตอร์โหนดนั่นเอง ดังรูปภาพที่ 6.2.2



รูปภาพที่ 6.2.2 แสดงอัลกอริทึมของโปรแกรม

6.3 ขั้นตอนการทำงานของโปรแกรม

1. โปรแกรมจะอ่านค่าจากไฟล์แต่ละชุดข้อมูลมาแปลงข้อมูลให้อยู่ในรูปดับเบิลและจัดเก็บลงสู่บัฟเฟอร์ซึ่งเป็นอาร์เรย์ 2 มิติ
2. ในขั้นตอนนี้จะเกิดการตรวจสอบจำนวนโหนดของคลัสเตอร์ที่จะนำมาทำการประมวลผลโดยจะตัดตัวมาสเตอร์ออกเพราะจะไม่ได้ใช้มาสเตอร์เป็นตัวคำนวณด้วย แต่จะใช้เป็นตัวแบ่งงาน และดูทากส์ของงานทั้งหมดว่ามีกี่ทากส์ ในที่นี้ทากส์จะใช้เป็นจำนวนแถวของเมตริกซ์ตัวตั้งที่นำมาคำนวณ เช่น เราจะหาดีเทอร์มินันท์ของเมตริกซ์ 4x4 จำนวนทากส์จะเป็น 4 หรือ ถ้าเราจะนำเมตริกซ์ 3x3 บวกกับเมตริกซ์ 3x3 จำนวนทากส์จะเป็น 3 เป็นต้น เมื่อได้จำนวนทากส์และจำนวนโหนดทั้งหมดแล้วเมื่อเรานำค่าของทากส์หารด้วยค่าโหนดจะได้จำนวนครั้งที่แต่ละโหนดต้องรับข้อมูลไปประมวลผล เช่น ทากส์ = 4 , โหนด = 2 ดังนั้นแต่ละโหนดจะต้องรับงานไปทำ 2 ชุดข้อมูล
3. ขั้นตอนต่อไปเป็นขั้นตอนที่จะตัดข้อมูลในบัฟเฟอร์เพื่อส่งไปให้โหนดต่าง ๆ คำนวณอัลกอริทึมในส่วนนี้จะขึ้นอยู่กับแต่ละฟังก์ชันการทำงานดังนี้

3.1 การบวกและลบ จะตัดแถวเดียวกันของแต่ละเมตริกซ์ส่งไปทำการบวกหรือลบกันที่สเลฟโหนด โดยจะส่งไปทั้ง 2 แถว

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad \begin{bmatrix} 9 & 8 & 7 \end{bmatrix}$$
$$\begin{matrix} 4 & 5 & 6 \end{matrix} \quad + \quad \begin{matrix} 4 & 3 & 1 \end{matrix}$$
$$\begin{matrix} 7 & 8 & 9 \end{matrix} \quad \quad \begin{matrix} 4 & 6 & 2 \end{matrix}$$

3.2 การคูณ จะส่งแถวหนึ่งแถวของตัวตั้งไปและ ส่งเมตริกซ์ตัวคูณไปทั้งหมด

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad \begin{bmatrix} 9 & 8 & 7 \\ 4 & 3 & 1 \\ 4 & 6 & 2 \end{bmatrix}$$
$$\begin{matrix} 4 & 5 & 6 \end{matrix} \quad +$$
$$\begin{matrix} 7 & 8 & 9 \end{matrix}$$

3.3 ดีเทอร์มินันท์ ตัวมาสเตอร์จะตัดแถวและหลักตามตำแหน่งของตัวหลักด้านหน้าออกแล้วส่งชุดเมตริกซ์จัตุรัสที่ได้ออกไปยังโหนดเป้าหมาย

$$\begin{matrix} (1) & 2 & 3 & 4 \end{matrix}$$
$$\begin{matrix} 5 \\ 4 \\ 8 \end{matrix} \begin{bmatrix} 4 & 9 & 7 \\ 8 & 6 & 2 \\ 9 & 1 & 3 \end{bmatrix}$$

เมื่อแบ่งงานตามฟังก์ชันแล้วมาสเตอร์โหนดก็จะกำหนดโหนดเป้าหมายและส่งข้อมูลไปยังสเลฟโหนดต่าง ๆ เพื่อทำการประมวลผลต่อไป

4. สเลฟโหนดจะรับค่าข้อมูลที่ส่งมาให้มาเก็บไว้ที่ตัวเองเพื่อประมวลผลต่อไป
5. สเลฟโหนดจะนำค่าข้อมูลไปประมวลผลตามฟังก์ชันการคำนวณต่าง ๆ ดังนี้
   5.1 การบวกและลบ จะนำชุดข้อมูล 2 แถวที่ได้รับ มาบวก ลบกันในตำแหน่งเดียวกัน
   5.2 การคูณจะนำแถวที่รับมาคูณกับเมตริกซ์ตัวคูณที่ส่งมาด้วยที่ละแถว
   5.3 ดีเทอร์มีนันท์จะนำชุดเมตริกซ์จัตุรัสที่ได้มาหาตามวิธีรีเดอร์ซีฟในแต่ละโหนดที่ได้รับ โดยการตัดตามตำแหน่งหลักด้านหน้าแล้วรีเดอร์ซีฟต่อไปเรื่อย ๆ
6. เมื่อสเลฟโหนดแต่ละโหนดทำการประมวลผลจนได้ผลลัพธ์ออกมาแล้วก็จะส่งกลับไปยังมาสเตอร์โหนด
7. ในขณะนี้มาสเตอร์โหนดจะทำการรอรับผลลัพธ์ของข้อมูลจากแต่ละโหนดเพื่อนำมาเก็บในตำแหน่งที่ถูกกำหนดไว้แล้วโดยในการส่งและรับข้อมูลนี้จะเป็นไปตามลำดับ
8. ทำการเรียบเรียงผลลัพธ์เพื่อการแสดงผลต่อไป

ในขั้นตอนที่ 1-3 และ 7-8 จะเป็นขั้นตอนที่โปรแกรมให้มาสเตอร์โหนดทำงาน โดยจะมีส่วนของการแบ่งงานอยู่ ส่วนในขั้นตอนที่ 4-6 จะเป็นขั้นตอนที่โปรแกรมให้สเลฟโหนดทำงานรวมถึงฟังก์ชันการทำงานที่จะสั่งให้ประมวลผลก็จะอยู่ในส่วนนี้ด้วย โดยเมื่อแต่ละสเลฟโหนดได้รับข้อมูลมาแล้วก็จะเริ่มประมวลผลกันไปในลักษณะขนาน โดยไม่เกี่ยวเนื่องกับโหนดอื่น

บทที่ 7

การทดลอบระบบ

7.1 แนวคิดในการทดสอบระบบ

ระบบที่ทำงานด้วยการประมวลผลแบบขนานนั้น ส่วนใหญ่แล้วจะมีการทดสอบและวัดผลถึงความสามารถและประสิทธิภาพของระบบ โดยส่วนใหญ่แล้วมักจะใช้งานที่มีขนาดใหญ่และมีการคำนวณสูง ๆ เข้ามาใช้ในการทดสอบ เช่น การคำนวณทางคณิตศาสตร์ การทำงานด้านกราฟฟิกสูง ๆ เป็นต้น

ในโครงงานชิ้นนี้จะทำการทดสอบระบบคลัสเตอร์ที่เราได้ทำการติดตั้งไว้ โดยนำโปรแกรมที่เขียนโดยใช้หลักการแบบขนานมาทำการทดสอบ และจะใช้การทดสอบถึงความเร็วของการประมวลผลระหว่างโปรแกรมซีเควนเชียลกับโปรแกรมขนานในจำนวนโหนดต่าง ๆ กันออกไปเพื่อนำมาสรุปผล โดยหลักทั่วไปแล้วการวัดผลของการทำงานแบบขนานนั้นจะวัดตามจุดประสงค์ของการใช้งานการประมวลผลแบบขนาน 3 ประการนี้คือ

- ความเร็วในการประมวลผล ซึ่งอาจนำมาเปรียบเทียบกันได้โดย

$$\text{ค่าเปรียบเทียบความเร็ว (speed up.sp)} = \frac{\text{เวลาที่ใช้ในการประมวลผลบน 1 โพรเซสเซอร์}}{\text{เวลาที่ใช้ในการประมวลผลบน จำนวน p โพรเซสเซอร์}}$$

- ประสิทธิภาพของการใช้ทรัพยากร

$$Ep = \frac{Sp}{P}$$

- ความประหยัดขึ้นของราคาค่าใช้จ่าย

โดยส่วนใหญ่แล้วการวัดผลโดยทั่วไปแล้วมักจะมองกันที่ความเร็วในการประมวลผลมากกว่า

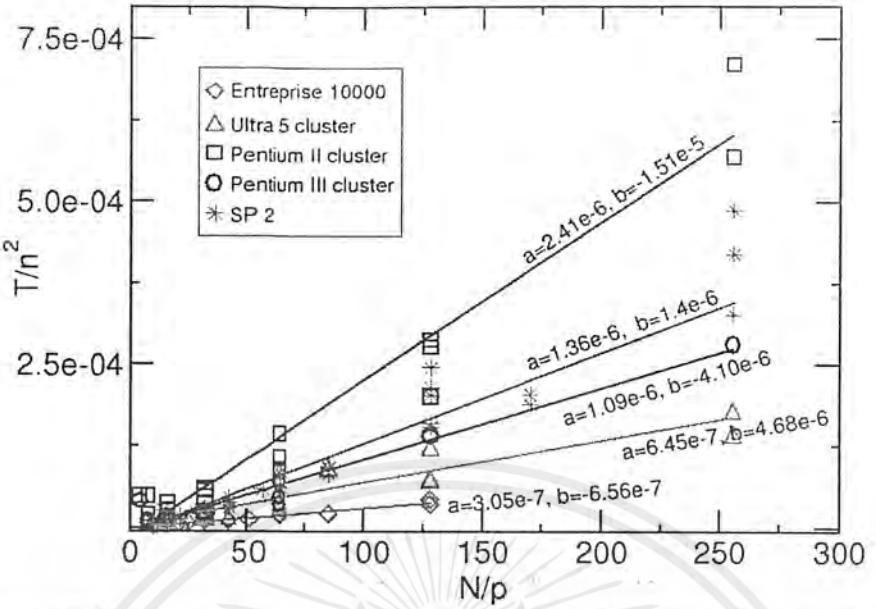
7.2 การทดสอบระบบ

ในโครงงานชิ้นนี้จะทำการทดสอบระบบคลัสเตอร์ที่เราได้ทำการติดตั้งไว้ โดยนำโปรแกรมที่เขียนโดยใช้หลักการแบบขนานมาทำการทดสอบ และจะใช้การทดสอบถึงความเร็วของการประมวลผลระหว่างโปรแกรมซีเควนเชียลกับโปรแกรมขนานในจำนวนโหนดต่าง ๆ กันออกไปเพื่อนำมาสรุปผล

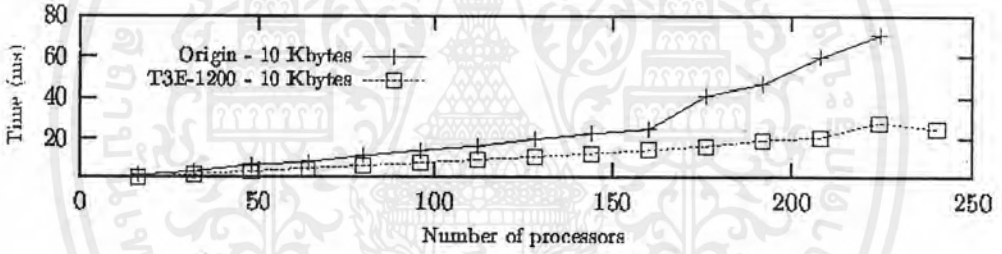
การทดสอบในโครงงานนี้จะทำโดยใช้การคำนวณหาดีเทอร์มินันท์ของเมตริกซ์มาใช้ ซึ่งเป็นที่นิยมในการทดสอบระบบเพื่อตรวจสอบความเร็ว เพราะโปรแกรมจะต้องใช้การคำนวณมาก ๆ โปรแกรมที่ใช้ในการทดสอบจะใช้ 2 โปรแกรมคือ โปรแกรมการหาดีเทอร์มินันท์ที่ออกแบบและเขียนด้วยวิธีทางด้านซีเควนเชียลกับ โปรแกรมแบบขนาน ในการทดสอบบางครั้งคนส่วนใหญ่มักจะใช้การใช้โปรแกรมแบบขนานที่เขียนขึ้นและให้ทำงานบนโพรเซสเดียวมาแทนผลของโปรแกรมแบบซีเควนเชียล ซึ่งวิธีนี้จะเป็นวิธีที่ผิดเพราะว่าอัลกอริทึมและการทำงานจะต่างกันจึงอาจทำให้เกิดความคลาดเคลื่อนได้ วิธีที่ถูกต้องคือเราต้องเขียนโปรแกรมแบบซีเควนเชียลที่มีอัลกอริทึมในการคำนวณเหมือนกับโปรแกรมแบบขนานมาทำการทดสอบ โปรแกรมทั้ง 2 จะถูกนำมาทำงานบนระบบคลัสเตอร์เพื่อเปรียบเทียบผล

7.2.1 สมมติฐานก่อนการทดลอง

ตามหลักการประมวลผลแบบขนานแล้ว เมื่อมีการเพิ่มจำนวนโพรเซสเซอร์ที่ทำการประมวลผลแล้วจะทำให้การประมวลผลมีความเร็วในการทำงานสูงขึ้น มากกว่าการประมวลผลในจำนวนโพรเซสเซอร์ที่น้อยกว่า เนื่องจากโพรเซสเซอร์จะช่วยแบ่งงานกันทำ โดยพิจารณาจากการทดลองซึ่งมีผู้ทำการทดลองในลักษณะนี้ และเผยแพร่ผลการทดลองบนอินเทอร์เน็ต ผลการทดลองที่ได้เป็นดังรูปที่ 7.2.1.1



รูปภาพที่ 7.2.1.1 แสดงการทดสอบระบบคลัสเตอร์อื่น ๆที่เผยแพร่บนอินเทอร์เน็ต



รูปภาพที่ 7.2.1.2 แสดงผลการทดสอบระบบคลัสเตอร์ในต่างประเทศ

ซึ่งทั้ง 2 รูปเป็นผลการทดลองในระบบคลัสเตอร์แต่ละแพลตฟอร์ม จะเห็นได้ว่าโดยส่วนใหญ่แล้วเมื่อเพิ่มจำนวนโพรเซสเซอร์เข้าไปแล้วนั้นความเร็วในการประมวลผลจะเพิ่มขึ้นในรูปที่ค่อนข้างจะเป็นลักษณะเชิงเส้น โดยในแต่ละแพลตฟอร์มมีเครื่องลูกที่ต่ออยู่เป็นสเปคเดียวกัน

7.2.2 การทดลอง

ในการทดลองจะทดสอบใน 2 รูปแบบ คือ แบบที่จำนวนโพรเซสเซอร์มีขนาดสเปคเดียวกัน และแบบที่โพรเซสเซอร์มีขนาดสเปคที่ใช้ต่างกัน ดังต่อไปนี้

การทดลองที่ 1

วัตถุประสงค์การทดลอง

- เพื่อแสดงประสิทธิภาพการทำงานและเปรียบเทียบความเร็วในการประมวลผลที่เพิ่มขึ้นเมื่อเพิ่มจำนวนโพรเซสเซอร์ที่มีสเปคเดียวกัน

- เพื่อเปรียบเทียบความเร็วของการทำงานแบบซีเควนเชียลและแบบขนาน

ลักษณะการทดลอง

การทดลองนี้ทำในระบบคลัสเตอร์ที่เราทำขึ้นจะใช้ 4 โหนดที่มีสเปคเดียวกันทั้งหมด ซึ่งการวัดผลจะเริ่มที่การทำงานด้วยโปรแกรมซีเควนเชียล, โปรแกรมแบบขนานด้วยขนาด 1 โหนด, 2 โหนด, 3 โหนด และ 4 โหนดตามลำดับในหลาย ๆ ขนาดเมตริกซ์เพื่อนำผลไปเปรียบเทียบกัน โดยในแต่ละโหนดมีสเปคเป็นเครื่อง Compaq 486 มีหน่วยความจำ 16 Mbyte

ผลการทดลอง

ผลการทดลองที่ได้เป็นไปตามตารางดังต่อไปนี้

การประมวลผล / ขนาด	ซีเควนเชียล(sec)	แบบขนาน 1 โหนด(sec)
เมตริกซ์ 8x8	6.44	8.476
	6.60	8.443
	6.61	8.469
	6.62	8.467
	6.58	8.477
ค่าเฉลี่ย	6.61	8.466
เมตริกซ์ 9x9	58.72	71.607
	58.58	71.650
	58.74	71.684
	58.65	71.617
	58.67	71.608
ค่าเฉลี่ย	58.672	71.6332
เมตริกซ์ 10x10	604.54	736.054
	602.36	736.117
	602.51	735.859
	602.55	736.294
	601.83	735.520
ค่าเฉลี่ย	602.758	735.9688

ตารางที่ 7.2.2.1 ตารางเปรียบเทียบความเร็วการประมวลผลระหว่างซีเควนเชียลกับแบบขนาน 1 โหนด

การประมวลผล / ขนาด	แบบขนาน 2 โหนด(sec)			
	ความเร็วโหนดเร็วสุด	ความเร็วโหนดช้าสุด	ช่วง Idle	เวลาการประมวลผล
เมตริกซ์ 8x8	4.099	4.280	0.181	4.270
	4.096	4.283	0.187	4.275
	4.100	4.280	0.18	4.278
	4.087	4.276	0.189	4.274
	4.094	4.277	0.183	4.282
ค่าเฉลี่ย	4.0952	4.2792	0.184	4.2758
เมตริกซ์ 9x9	31.882	39.131	7.249	39.139
	31.847	39.093	7.246	39.101
	31.895	39.133	7.238	39.141
	31.850	39.089	7.239	39.098
	31.842	39.067	7.255	39.075
ค่าเฉลี่ย	31.8632	39.1026	7.2454	39.108
เมตริกซ์ 10x10	358.776	367.817	9.041	367.814
	358.981	367.796	8.815	367.797
	359.304	368.125	8.821	368.126
	359.073	367.914	8.841	367.911
	358.781	367.059	8.278	367.957
ค่าเฉลี่ย	358.983	367.7422	8.7592	367.921

ตารางที่ 7.2.2.2 ตารางเปรียบเทียบความเร็วการประมวลผลแบบขนาน 2 โหนด

การประมวลผล / ขนาด	แบบขนาน 3 โหนด(sec)			
	ความเร็วโหนดเร็วสุด	ความเร็วโหนดช้าสุด	ช่วง Idle	เวลาการประมวลผล
เมตริกซ์ 8x8	2.129	3.061	0.932	3.064
	2.125	3.062	0.937	3.067
	2.130	3.059	0.929	3.075
	2.124	3.050	0.926	3.077
	2.131	3.062	0.931	3.067
ค่าเฉลี่ย	2.1278	3.0588	0.931	3.07
เมตริกซ์ 9x9	23.160	23.965	0.805	23.966
	23.183	23.992	0.809	23.992
	23.189	23.998	0.809	23.999
	23.212	24.003	0.791	24.003
	23.195	24.005	0.81	24.006
ค่าเฉลี่ย	23.1878	23.9926	0.8048	23.9932
เมตริกซ์ 10x10	212.034	285.692	73.658	285.325
	211.035	285.548	73.613	285.585
	211.974	285.609	73.629	285.637
	212.104	285.796	73.692	285.833
	212.183	285.883	73.65	285.870
ค่าเฉลี่ย	212.046	285.6938	73.6484	285.6496

ตารางที่ 7.2.2.3 ตารางเปรียบเทียบความเร็วการประมวลผลแบบขนาน 3 โหนด

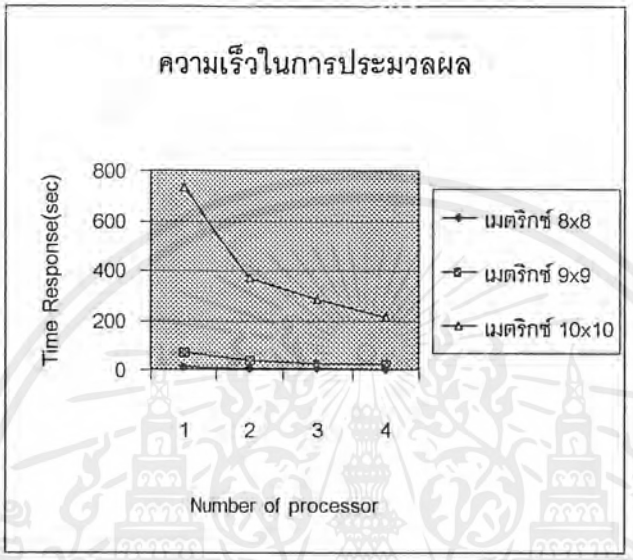
การประมวลผล / ขนาด	แบบขนาน 4 โหนด(sec)			
	ความเร็วโหนดเร็วสุด	ความเร็วโหนดช้าสุด	ช่วง Idle	เวลาการประมวลผล
เมตริกซ์ 8x8	2.002	2.231	0.229	2.237
	2.006	2.227	0.221	2.234
	2.019	2.249	0.23	2.253
	1.998	2.204	0.206	2.208
	1.996	2.222	0.226	2.226
ค่าเฉลี่ย	2.0042	2.2266	0.2224	2.2316
เมตริกซ์ 9x9	15.213	23.243	8.03	23.248
	15.185	23.230	8.045	23.235
	15.244	23.244	8.00	23.249
	15.201	23.252	8.051	23.257
	15.252	23.270	8.018	23.275
ค่าเฉลี่ย	15.219	23.2478	8.0288	23.2528
เมตริกซ์ 10x10	138.449	212.139	73.69	212.167
	138.470	212.145	73.675	212.128
	138.388	212.026	73.638	212.038
	138.490	212.201	73.711	212.230
	138.480	212.133	73.653	212.145
ค่าเฉลี่ย	138.4554	212.1288	73.6734	212.1416

ตารางที่ 7.2.2.4 ตารางเปรียบเทียบความเร็วการประมวลผลแบบขนาน 4 โหนด

สรุปผลการทดลองที่ 1

จากการทดลองเราสามารถนำค่าเฉลี่ยของความเร็วในการประมวลผลในเมตริกซ์ขนาดต่าง ๆ มาทำเป็นกราฟได้ดังรูปต่อไปนี้

รูปกราฟแสดงความเร็วในการประมวลผล



รูปภาพที่ 7.2.2.1 แสดงความสัมพันธ์ระหว่างเวลาที่ใช้คำนวณและจำนวนของโพรเซสที่ใช้

จากรูปจะเห็นว่า ความเร็วในการประมวลผลแบบขนานจะมีความเร็วมากกว่าการประมวลผลแบบซีเควนเชียลและ เมื่อเราเพิ่มโพรเซสเซอร์เข้าไปทำงานมากขึ้นจะทำให้การประมวลผลเร็วขึ้นแต่ว่ารูปกราฟไม่เป็นลักษณะเชิงเส้นดังที่ได้ตั้งสมมติฐานไว้ในตอนต้นทั้งนี้อาจจะมีสาเหตุมาจากหลาย ๆ ตัวแปรด้วยกันในด้านการทดลอง และสาเหตุที่สำคัญอย่างหนึ่ง ซึ่งโปรแกรมแบบขนานมักจะพบกันโดยทั่วไปนั่นก็คือปัญหาด้านโหลดบาลานซ์ ซึ่งก็คือปัญหาที่เมื่อมีการแบ่งงานกันไปทำที่โพรเซสเซอร์ต่าง ๆ แล้ว ต้องมีการรอผลลัพธ์ของแต่ละตัว โดยเหตุนี้จะทำให้โพรเซสเซอร์ตัวที่ทำงานเสร็จก่อนต้องรอโพรเซสเซอร์ที่ทำงานช้ากว่าโดยว่างงานไป ทำให้เสียประโยชน์และเสียเวลาในการทำงานมาก ซึ่งจะเห็นผลมากในเครื่องที่ใช้คำนวณมีประสิทธิภาพต่างกันมาก ๆ

การทดลองที่ 2

วัตถุประสงค์การทดลอง

- เพื่อแสดงให้เห็นถึงปัญหาของโหลดบาลานซ์ในการเขียนโปรแกรมเชิงขนาน
- เพื่อแสดงผลกระทบที่เกิดจากอัลกอริทึมของโปรแกรมที่ขาดการทำโหลดบาลานซ์

ลักษณะการทดลอง

ขั้นตอนการทดลองที่ 2.2 นี้จะเหมือนกับในการทดลองที่ 2.1 โดยจะใช้โปรแกรมในการหาดีเทอร์มินันท์โปรแกรมเดียวกับที่ทำการทดลองไปในการทดลองที่ 2.1 มาใช้เปรียบเทียบความเร็วเช่นเดียวกัน แต่ในครั้งนี้จะใช้เครื่องที่สเปคต่างกัน 3 เครื่องในการทดลองโดยประกอบด้วย

- Pentium 200 MHz หน่วยความจำ 80 Mbyte
- Pentium II 333 MHz หน่วยความจำ 64 Mbyte
- Pentium Celeron 566 MHz หน่วยความจำ 96 Mbyte

โดยจะใช้ผลการทดลองมาเปรียบเทียบความเร็วเมื่อเพิ่มโพรเซสเซอร์เข้าไปในลักษณะที่ต่าง ๆ กัน โดยแบ่งการทดลองออกเป็น 4 ส่วน

2.2.1 จะทำการทดลองให้ทำการเปรียบเทียบความเร็วของโปรแกรมซีเควนเชียลกับโปรแกรมเชิงขนานในเครื่องมาสเตอร์เพียงเครื่องเดียวซึ่งในการทดลองจะใช้เครื่อง Pentium II 333 MHz

2.2.2 ทดลองใช้ 2 เครื่องในการประมวลผลแบบขนานโดยใช้เครื่องมาสเตอร์กับเครื่องที่มีประสิทธิภาพต่ำกว่าเครื่องมาสเตอร์เข้ามาช่วยประมวลผลในการทดลองนี้ใช้ Pentium II 333 MHz กับเครื่อง Pentium 200 MHz มาประมวลผล

2.2.3 ทดลองใช้ 2 เครื่องในการประมวลผลแบบขนานโดยใช้เครื่องมาสเตอร์กับเครื่องที่มีประสิทธิภาพสูงกว่าเครื่องมาสเตอร์เข้ามาช่วยประมวลผลในการทดลองนี้ใช้ Pentium II 333 MHz กับเครื่อง Pentium Celeron 566 MHz มาประมวลผล

2.2.4 นำเครื่องทั้งสามคือ Pentium II 333 , Pentium 200 MHz และ Pentium Celeron 566 มาประมวลผลร่วมกัน

ผลการทดลอง

จะแบ่งผลการทดลองออกเป็น 4 ส่วนดังที่กล่าวมาข้างต้น

ผลการทดลองที่ 2.1 เปรียบเทียบความเร็วของการประมวลผลแบบขนานกับการประมวลผลแบบซีเควนเชียลบนเครื่องมาสเตอร์ (Pentium II 333 MHz)

การประมวลผล / ขนาด	ซีเควนเชียล(sec)	แบบขนาน 1 โหนด(sec)
เมตริกซ์ 8x8	0.55	0.6719
	0.55	0.6721
	0.54	0.6743
	0.54	0.6761
	0.54	0.6759
ค่าเฉลี่ย	0.544	0.6740
เมตริกซ์ 9x9	5.00	5.927
	5.00	5.9156
	5.02	5.9066
	5.03	5.929
	5.04	5.905
ค่าเฉลี่ย	5.018	5.916
เมตริกซ์ 10x10	49.44	60.3470
	49.55	60.339
	49.58	60.3458
	49.37	60.3412
	49.49	60.3649
ค่าเฉลี่ย	49.486	60.3475

ตารางที่ 7.2.2.5 เปรียบเทียบความเร็วระหว่างการประมวลผลแบบซีเควนเชียลและแบบขนาน 1 โหนด

ผลการทดลองที่ 2.2 แสดงผลการทำงานแบบขนานบนเครื่องมาสเตอร์(Pentium II 333 MHz) ที่ทำงานกับเครื่องสเลฟที่มีประสิทธิภาพต่ำกว่า(Pentium 200 MHz)

การประมวลผล / ขนาด	แบบขนาน 2 โหนด(sec)			
	ความเร็วโหนดเร็วสุด	ความเร็วโหนดช้าสุด	ช่วง Idle	เวลาการประมวลผล
เมตริกซ์ 8x8	0.772	0.918	0.146	0.919
	0.778	0.923	0.145	0.924
	0.774	0.919	0.145	0.920
	0.770	0.916	0.148	0.917
	0.771	0.9159	0.1449	0.916
ค่าเฉลี่ย	0.773	0.9183	0.14578	0.9192
เมตริกซ์ 9x9	5.759	8.502	2.743	8.503
	5.755	8.500	2.745	8.501
	5.753	8.497	2.744	8.498
	5.754	8.496	2.742	8.497
	5.751	8.491	2.74	8.492
ค่าเฉลี่ย	5.7544	8.4972	2.7428	8.4982
เมตริกซ์ 10x10	71.091	81.315	10.224	81.320
	71.135	81.358	10.223	81.363
	71.108	81.334	10.226	81.339
	71.128	81.315	10.187	81.320
	71.084	81.322	10.238	81.327
ค่าเฉลี่ย	71.1092	81.3288	10.2196	81.3338

ตารางที่ 7.2.2.6 แสดงเวลาที่ใช้ในการประมวลผลแบบขนาน 2 โหนดที่ใช้โหนดช้าเข้าไป

การทดลองที่ 2.3 แสดงผลการทำงานแบบขนานบนเครื่องมาสเตอร์(Pentium II 333 MHz) ที่ทำงานกับเครื่องสเลฟที่มีประสิทธิภาพต่ำกว่า(Pentium Celeron 566 MHz)

การประมวลผล / ขนาด	แบบขนาน 2 โหนด(sec)			
	ความเร็วโหนดเร็วสุด	ความเร็วโหนดช้าสุด	ช่วง Idle	เวลาการประมวลผล
เมตริกซ์ 8x8	0.313	0.350	0.037	0.3496
	0.310	0.3466	0.0366	0.3481
	0.302	0.339	0.037	0.338
	0.303	0.339	0.036	0.339
	0.304	0.3406	0.0366	0.3401
ค่าเฉลี่ย	0.3064	0.34304	0.03664	0.34116
เมตริกซ์ 9x9	2.630	3.0191	0.3891	3.0198
	2.663	3.051	0.388	3.052
	2.627	3.0162	0.3892	3.0170
	2.629	3.017	0.388	3.018
	2.6304	3.0187	0.3883	3.0195
ค่าเฉลี่ย	2.63855	3.0244	0.38852	3.0252
เมตริกซ์ 10x10	27.6734	30.211	2.5376	30.210
	27.671	30.208	2.537	30.207
	27.673	30.214	2.541	30.213
	27.676	30.2152	2.538	30.2149
	27.6824	30.220	2.5376	30.220
ค่าเฉลี่ย	27.6751	30.2136	2.538	30.2128

ตารางที่ 7.2.2.7 แสดงเวลาที่ใช้ในการประมวลผลแบบขนาน 2 โหนดที่ใช้โหนดเร็วเข้าไป

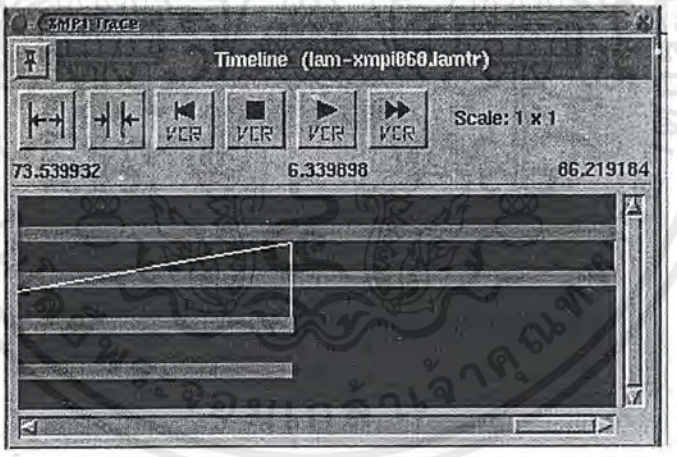
การทดลองที่ 2.4 แสดงผลการทำงานแบบขนานบนเครื่องมาสเตอร์(Pentium II 333 MHz) ที่ทำงานกับเครื่องสเลฟทั้งสองเครื่อง(Pentium Celeron 566 MHz และ Pentium 200 MHz)

การประมวลผล / ขนาด	แบบขนาน 3 โหนด(sec)			
	ความเร็วโหนดเร็วสุด	ความเร็วโหนดช้าสุด	ช่วง Idle	เวลาการประมวลผล
เมตริกซ์ 8x8	0.320	0.697	0.377	0.698
	0.313	0.689	0.376	0.690
	0.316	0.692	0.376	0.693
	0.322	0.697	0.375	0.698
	0.316	0.6913	0.3753	0.6922
ค่าเฉลี่ย	0.3174	0.68526	0.3758	0.6942
เมตริกซ์ 9x9	3.793	5.106	1.313	5.107
	3.797	5.109	1.312	5.110
	3.793	5.107	1.314	5.108
	3.797	5.109	1.312	5.110
	3.788	5.101	1.313	5.102
ค่าเฉลี่ย	3.7936	5.1064	1.3128	5.1074
เมตริกซ์ 10x10	38.5620	52.300	13.7371	52.302
	38.578	52.313	13.735	52.315
	38.570	52.3064	13.7364	52.3085
	38.575	52.306	13.731	52.308
	38.633	52.369	13.736	52.371
ค่าเฉลี่ย	38.5836	52.3188	13.7351	52.3209

ตารางที่ 7.2.2.8 แสดงเวลาที่ใช้ในการประมวลผลแบบขนาน 3 โหนด

สรุปผลการทดลองที่ 2

ตามทฤษฎีแล้วถ้าหากโหนดหรือโพรเซสเซอร์ในการประมวลผลเพิ่มขึ้นแล้วนั้น ความเร็วในการประมวลผลจะต้องเร็วขึ้น แต่เมื่อเรานำผลการทดลองตารางที่ 2.2.2 มาพิจารณา ๆ แล้วจะพบว่าความเร็วในการประมวลผลแบบขนานที่เพิ่มเครื่องเข้าไปช่วยในการประมวลผลนั้นไม่ได้ทำให้ความเร็วของการทำงานเพิ่มขึ้นจากการทำงานในเครื่องเดียวเลย และยังทำให้การหาความเร็วในการประมวลผลช้าลงอีกด้วย แต่เมื่อเราดูผลการทดลองของตารางที่ 2.2.3 จะเห็นว่าความเร็วในการประมวลผลมีความเร็วสูงขึ้นซึ่งในส่วนนี้ได้เป็นไปตามสมมติฐาน แต่เมื่อเราทดลองใช้เครื่องที่มีประสิทธิภาพต่ำกว่าเพิ่มเข้าไปอีกเป็น 3 เครื่องกลับทำให้ความเร็วในการประมวลผลลดลง หากเรานำความเร็วในการประมวลผลมาสร้างกราฟ จะไม่มีลักษณะเป็นเชิงเส้น หรือแนวโน้มด้านใดเหมือนในการทดลองที่ 2.1 เลย ซึ่งเราสามารถอธิบายผลการทดลองในส่วนนี้ได้ว่ามีสาเหตุมาจากผลของโหลดบาลานซ์นั่นเอง ซึ่งเมื่อเครื่องที่เร็วกว่าทำงานเสร็จสิ้นแล้ว การประมวลผลยังคงต้องรอเครื่องที่ช้ากว่าด้วยโดยเครื่องที่เร็วกว่าเหล่านั้นไม่สามารถช่วยเครื่องที่มีความเร็วช้ากว่าทำงานได้เลย ซึ่งในส่วนนี้ทำให้เมื่อเพิ่มเครื่องที่ประสิทธิภาพต่ำกว่าเข้าไปก็จะทำให้ช้าลงไปอีกนั่นเอง ลักษณะของปัญหานี้เป็นดังรูป โดยจากรูปจะเห็นว่าแถวบนสุดเป็นมาสเตอร์ซึ่งในแถวที่ 3,4 นั้นทำงานเสร็จแล้ว ส่วนในแถวที่ 2 นั้นยังทำงานไม่เสร็จทำให้ที่เหลือต้องรอโดยเสียเวลาไปอย่างเปล่าประโยชน์



รูปภาพที่ 7.2.2.2 แสดงลักษณะการทำงานจากโหนดที่ความเร็วต่างกัน

ปัญหาโหลดบาลานซ์นี้ในระบบที่มีเครื่องที่ร่วมกันประมวลผลมีสเปคเหมือนกันหรือใกล้เคียงกัน จะมีผลกระทบน้อยกว่าระบบที่ใช้เครื่องสเปคต่างกันมาก ๆ ในการประมวลผล เนื่องจากแบบที่มีสเปคเดียวกัน สเลฟโหนดจะประมวลผลเสร็จในแต่ละครั้งใกล้เคียงกันจึงไม่เกิดปัญหาในเรื่องการรอกันของแต่ละโหนด

ปัญหาโหลดบาลานซ์นี้สามารถแก้ไขได้ที่อัลกอริทึมในการทำงานของโปรแกรมโดยทำอัลกอริทึมส่วนของโหลดบาลานซ์เพิ่มเข้าไปคือต้องมีการตรวจสอบสถานะของการทำงานในแต่ละโหนดด้วย

หากว่ามีโหนดใดเสร็จงานก่อนแล้วนั้น เราต้องทำการให้งานที่เหลือไปยังโหนดนั้นก่อนเพื่อไม่ทำให้โหนดนั้นต้องว่างงาน และเสียเวลาไป

7.2.3 สรุปผลการทดลอง

1. การเพิ่มโพรเซสเซอร์ในการประมวลผล ไม่ได้ให้ความเร็วเพิ่มขึ้นเสมอไป เนื่องจากตัวแปรที่มีผลกระทบต่อความเร็วในการประมวลผลมีหลายอย่างไม่ว่าจะเป็นความเร็วของระบบเครือข่ายในการติดต่อ อัลกอริทึมในการทำงาน ขนาดของงานและความเหมาะสมกับโหนดที่ใช้งานด้วย และนอกจากนี้ตัวแปรที่สำคัญยิ่งที่พบในการทดลองนี้คือความสามารถของซีพียูแต่ละตัวในการประมวลผล

จากเหตุผลเหล่านี้เองทำให้เราเห็นว่าเมื่อเพิ่มจำนวนโหนดมากขึ้น ความเร็วในการทำงานจะไม่เร็วขึ้นเสมอไป โดยส่วนใหญ่แล้วการทำระบบคลัสเตอร์เพื่อมาใช้ในการประมวลผลแบบขนานนั้นจะใช้เครื่องไคลเอนท์ที่มีสเปคเท่า ๆ กัน หรือใกล้เคียงกันเพื่อลดปัญหาในส่วนนี้นั่นเอง

สรุปตัวแปรที่มีผลกับความเร็วของการ ประมวลผลมีดังต่อไปนี้

- ความเร็วของเครือข่ายในการติดต่อสื่อสาร
- ความสามารถของซีพียูที่นำมาเป็นสเลฟโหนดในการประมวลผล
- ความเหมาะสมของขนาดงานที่แบ่งออกไป
- อัลกอริทึมของโปรแกรมในการแบ่งงาน

2. ในแง่ของประโยชน์และความคุ้มค่าแล้ว การนำซีพียูเก่า ๆ และประสิทธิภาพต่ำมาเชื่อมต่อเพื่อทำการประมวลผลแบบขนานนั้น ไม่ได้ให้ประสิทธิภาพที่น่าพอใจเท่าไรนัก เนื่องจากต้องใช้การเพิ่มซีพียูเข้าไปเป็นจำนวนมากจึงจะได้ประสิทธิภาพเท่ากับเครื่องที่มีความเร็วสูง ซึ่งเมื่อเปรียบเทียบแล้วการใช้เครื่องที่มีประสิทธิภาพสูงเพียงเครื่องเดียวในการทำงานเลยนั้น อาจจะทำให้เราสะดวกสบายในการติดตั้งรวมถึงประหยัดค่าใช้จ่ายในบางแง่ด้วย เช่น ค่าใช้จ่ายด้านเครือข่าย ค่าไฟฟ้า และอื่น ๆ

3. อัลกอริทึมของโปรแกรมเชิงขนาน เป็นหัวใจของการทำงานเลยทีเดียวก็ว่าได้ เพราะจะมีผลเป็นอย่างมากต่อความเร็วในการประมวลผล ส่วนที่สำคัญ ๆ ของอัลกอริทึมคือการแบ่งงานไปคำนวณยังโพรเซสต่าง ๆ ซึ่ง ถ้าหากในส่วนนี้เราแบ่งงานได้ไม่ดีเท่าที่ควร ความเร็วของการประมวลผลนอกจากจะไม่เร็วขึ้นแล้วยังอาจทำให้ช้าลงอีกด้วย ดังตัวอย่างปัญหาของโหลดบาลานซ์ข้างต้นนั้น เป็นปัญหาใหญ่ที่หลาย ๆ ระบบต้องทำการแก้ไข เพราะปัญหานี้จะทำให้การใช้งานของซีพียูทำได้อย่างไม่มีประสิทธิภาพและ ได้ไม่เต็มที่ ซึ่งหากอัลกอริทึมของโปรแกรมสามารถแก้ปัญหาในส่วนนี้ได้จะทำให้ใช้งานระบบได้อย่างเต็มประสิทธิภาพเลยทีเดียว

แนวทางในการแก้ปัญหาโหลดบาลานซ์

การใช้สเปคของซีพียูในการประมวลผลใกล้เคียงกันนั้นเป็นวิธีแก้ปัญหาที่หลาย ๆ ระบบคลัสเตอร์นำไปใช้ การแก้ปัญหาแบบนี้อาจจะช่วยให้การใช้งานทำได้ดีขึ้นก็จริง แต่ว่าเป็นการแก้ปัญหาที่ปลายเหตุมากกว่าเนื่องจากว่าการคำนวณอาจเกิดขึ้นจากโจทย์ในหลาย ๆ แบบทำให้จะการทำงานจะดีขึ้นได้บางกรณีเท่านั้น ดังนั้นวิธีที่ถูกต้องของการแบ่งงานก็ควรจะต่างกันไปตามแบบต่าง ๆ เราไม่ควรจะกำหนดลงไปแน่นอนในการแบ่งงาน ซึ่งการแก้ปัญหาที่ถูกจริง ๆ แล้วเราควรแก้ปัญหาที่อัลกอริทึมมากว่า นั่นคือเราควรมีการตรวจสอบสถานะการทำงานในแต่ละโหนด และแต่ละโหนดโดยใช้มาสเตอร์เป็นศูนย์กลางเพื่อทำการแลกเปลี่ยนข้อมูลในการทำงาน เมื่อโหนดใดทำงานเสร็จก่อนเราก็ควรจะให้งานที่เหลือแก่โหนดนั้นต่อ ซึ่งวิธีนี้จะทำให้ประสิทธิภาพการทำงานของระบบดีขึ้นไม่ว่าจะกรณีใด ๆ ก็ตาม

บทที่ 8

บทสรุปและวิจารณ์

สรุป

จากการศึกษาการทำงานของระบบคลัสเตอร์แบบดิสก์เลส และการประมวลผลแบบขนาน นั้นเราสามารถสรุปได้ดังนี้

1. ระบบดิสก์เลสคลัสเตอร์เป็นแนวความคิดการทำงานที่มีประโยชน์เป็นอย่างมากเนื่องจากเราไม่จำเป็นต้องใช้โหนดฝ่ายไคลเอนท์ที่มีฮาร์ดดิสก์มาใช้ เพียงแต่ใช้ซีพียูและหน่วยความจำของส่วนนั้นมาใช้เท่านั้น จึงเป็นการประหยัดทรัพยากรลงได้มาก และยังสามารถนำมาใช้กับแนวความคิดแบบนำกลับมาใช้ใหม่ (reuse) อีกด้วย นั่นคือการนำเครื่องที่ไม่ได้ใช้งานและตกรุ่นมาใช้ให้เกิดประโยชน์ได้
2. การเขียนโปรแกรมเชิงขนานนั้นยังไม่แพร่หลายและคนพัฒนานั้นก็ยังไม่มีมากนัก รวมถึงโปรแกรมที่เขียนขึ้นมาในแบบขนานจริง ๆ นั้นยังมีน้อย ซึ่งอาจจะเนื่องจากว่าโปรแกรมเมอร์หลาย ๆ คนคุ้นเคยกับการเขียนโปรแกรมแบบซีเควนเชียลมากกว่า จึงทำให้เป็นการยากหากมาเขียนโปรแกรมเชิงขนาน และปัจจุบันก็ยังมีคอมไพเลอร์แบบขนานขึ้นมาอีกมากมาย ซึ่งทำเราสามารถเขียนโปรแกรมแบบซีเควนเชียลแล้วคอมไพล์ให้ได้โปรแกรมเชิงขนานออกมา หากแต่ว่าในวิธีนี้เราไม่สามารถกำหนดการทำงานของโปรแกรมได้อย่างเต็มที่และเหมาะสมกับระบบของเรา ต่างกับการที่เราเขียนขึ้นมาเอง
3. ในส่วนของอัลกอริทึมของโปรแกรมนั้นมีความสำคัญเป็นอย่างมากในการทำงาน เพราะโปรแกรมเชิงขนานนั้น เป็นโปรแกรมที่ต้องทำการควบคุมไปยังโพรเซสเซอร์อื่น ๆ อีกด้วย ดังนั้นการทำงานนอกจากจะเขียนขึ้นเพื่อการทำงานของเครื่องมาสเตอร์แล้ว เรายังต้องใส่ใจและกำหนดการทำงานของเครื่องสเลฟอีกด้วย การควบคุม กับ การแบ่งงานก็เป็นอีกส่วนที่สำคัญมากในการเขียนโปรแกรม เพราะความเร็วในการทำงานและประสิทธิภาพในการทำงานจริง ๆ แล้วขึ้นอยู่กับจุดนี้นั่นเอง ในปัจจุบันอัลกอริทึมการแบ่งงานมีหลายแบบให้เราเลือกใช้ได้เหมาะกับการและระบบแบบต่าง ๆ อีกด้วย
4. MPI ซึ่งเป็นไลบรารี ที่ช่วยในการเขียนโปรแกรมแบบขนานนั้น ได้ถูกพัฒนาขึ้นมาหลาย ๆ เวอร์ชันเพื่อให้สะดวกสบายมากขึ้นในการเขียนโปรแกรมและ ยังรองรับมาตรฐานต่าง ๆ ซึ่งเป็นที่นิยมใช้เป็นอย่างมากในการนำมาพัฒนาโปรแกรมเชิงขนาน
5. จากการที่ได้ทดสอบกับโปรแกรมการทำงานของเมตริกซ์ จะพบว่าเมื่อมีจำนวนโหนดไคลเอนท์มากขึ้น จะทำให้ประสิทธิภาพการทำงานของโปรแกรมเมตริกซ์ทำงานได้อย่างรวดเร็วกว่าการใช้คอมพิวเตอร์เครื่องเดียวมาประมวลผล ซึ่งผลที่ได้รับก็เป็นที่น่าพอใจในระดับหนึ่งแต่ก็มีข้อที่ควรปรับปรุงและพัฒนาต่อตรงที่ถ้าสามารถใช้อัลกอริทึมที่จะทำให้การ

ทำงานเร็วขึ้นได้ระบบคลัสเตอร์ก็น่าจะประมวลผลความเร็วได้สูงกว่านี้ และประสิทธิภาพก็จะดีขึ้นกว่านี้

วิจารณ์

จากการศึกษาที่ผ่านมาพบว่ามีผู้คนจำนวนน้อยที่สนใจในการเขียนโปรแกรมแบบขนาน เพราะเข้าใจยาก และซอฟท์แวร์ที่มีอยู่ก็มักจะเป็นแบบซีเควนเชียล อีกทั้งการใช้ระบบปฏิบัติการลีนุกซ์ซึ่งคนมักไม่นิยมใช้มากนักถ้าเทียบกับระบบปฏิบัติการวินโดว์ที่ใช้ง่ายกว่า ซึ่งจากเหตุผลข้างต้นก็เลยทำให้การใช้ลีนุกซ์คลัสเตอร์เป็นไปได้ยากมากขึ้น แต่ถึงอย่างไรปัจจุบันก็ได้มีการรวมกลุ่มคนในเมืองไทยที่ศึกษาในด้านของลีนุกซ์ขึ้น ที่ทำการพัฒนาสร้างซอฟท์แวร์หรือทูลต่าง ๆในการทำให้การติดตั้งลีนุกซ์คลัสเตอร์และการใช้งานทำได้โดยง่าย ซึ่งก็คงจะทำให้ในอนาคตคงมีคนสนใจที่จะทำลีนุกซ์คลัสเตอร์แทนการสั่งซื้อซูเปอร์คอมพิวเตอร์ที่มีอยู่ซึ่งทำให้ประหยัดค่าใช้จ่ายลงได้

ข้อเสนอแนะ

ในส่วนของโปรแกรมเมตริกซ์เป็นการเขียนโปรแกรมโดยใช้อัลกอริทึมทั่วไป คือ การทำงานในส่วนของโปรแกรมยังไม่มีโหลดบาลานซ์ ซึ่งเป็นการตรวจสอบการทำงานของโพรเซสอื่น ๆ ระหว่างทำงานช่วยให้เราสามารถนำมากำหนดการแบ่งงาน ณ เวลาต่าง ๆ ได้อย่างเหมาะสมกับโหลดและความสามารถของซีพียู การแบ่งงานก็ไม่ควรมีการกำหนดไว้อย่างตายตัวเนื่องจากงานที่เข้ามาอาจมีหลายลักษณะงานซึ่งควรจะแบ่งงานไปตามความเหมาะสมมากกว่า นอกจากนี้เครื่องที่ใช้คำนวณก็ควรจะมีประสิทธิภาพที่ใกล้เคียงกันเพื่อลดปัญหาด้านโหลดบาลานซ์ของการประมวลผล ซึ่งจะทำให้ประสิทธิภาพและการใช้งานซีพียูเป็นไปได้ดีมากขึ้น

บรรณานุกรม

[1] Neil MacDonald , Elspeth Minty , Joel Malard , Tim Harding, Simon Brown,Mario Antonioletti "*Writing Message Passing Parallel Programs with MPI A Two Day Course on MPI Usage*", Edinburgh Parallel Computing Centre The University of Edinburgh

[2] ดร.ภูชงค์ อุทโยภาศ "*เทคโนโลยีคลัสเตอร์กับลีนุกส์*",สำนักบริการคอมพิวเตอร์ สถาบันวิจัยและพัฒนา ฯ ภาควิชาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเกษตรศาสตร์ ร่วมกับ ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC) มูลนิธิวิจัยเทคโนโลยีสารสนเทศ และมูลนิธิเพื่อการศึกษาคอมพิวเตอร์และการสื่อสาร

[3] "*MPI: A Message-Passing Interface Standard*",Message Passing Interface Forum May 5,1994

[4] Jack Radajewski and Douglas Eadline "*Beowulf Installation and Administration HOWTO*",Version 0.1.2 2 June 1999

[5] Kai Hwang,Zhiwei Xu "*Scalable Parallel computing Technology ,Architecture , Programming*" ,Mcgraw Hill,c1998

[6] จากเว็ปไซต์ http://www.beowulf.org/

[7] จากเว็ปไซต์ http://www.epcc.ed.ac.uk/epcc-tec/documents/techwatch-mpi/

[8] จากเว็ปไซต์ http://www.mpi.nd.edu/lam/

[9] จากเว็ปไซต์ http://www.latech.edu/~kurtz/cs240/mpi/intro_mpi_hjl/sld002.htm

[10] จากเว็ปไซต์ http://info.mcs.kent.edu/docs/mcsparlmach/encorec/section3.4.html

ภาคผนวก

ภาคผนวก ก การติดตั้งโปรแกรมที่เกี่ยวข้องกับการวิจัย

วิธีการติดตั้ง LAM

LAM (Local Area Multicomputer) เป็นซอฟท์แวร์ที่ช่วยในการคอมไพล์ภาษา MPI ที่ใช้ในการประมวลผลแบบขนาน โดยเราสามารถ ดาวน์โหลดได้จากเว็ปไซต์

http://www.mpi.nd.edu/downloads/lam/lam-6.3.2.tar.gz

สำหรับขั้นตอนการติดตั้ง มีดังต่อไปนี้

1. ใช้คำสั่ง gunzip -c lam-6.4-a3.tar.gz | tar xf - เพื่อแตกไฟล์ที่ดาวน์โหลดมา
2. ใช้คำสั่ง cd lam-6.4-a3 เพื่อเข้าสู่ไดเรกทอรีของไฟล์ที่ทำการแตกไฟล์ออกมา
3. ใช้คำสั่ง ./configure --prefix=/path/to/install/in เพื่อทำการติดตั้งระบบตามแต่ละเครื่อง โดยต้องการติดตั้งลงบนเส้นทาง(Path) ที่ต้องการ ถ้าไม่ระบุโปรแกรมจะติดตั้งใน Path /usr/local
4. ใช้คำสั่ง make ในการคอมไพล์โปรแกรม
5. หลังจากนั้นเราต้องทำการกำหนดเส้นทางโดยใช้คำสั่ง

   $ export PATH=$PATH:/ <PATH ที่ไฟล์ lam อยู่ >

   เช่น ถ้าไฟล์ที่ใช้งานอยู่ที่ /usr/local/lam/bin ก็ใช้คำสั่ง

   $ export PATH=$PATH:/usr/local/lam/bin
6. ถ้าต้องการให้ในครั้งต่อไปไม่ต้องมากำหนดเส้นทางทุกครั้งให้กำหนดเพิ่มคำสั่ง

   export PATH=$PATH:/ <PATH ที่ไฟล์ lam อยู่ >

ในไฟล์ .bashrc ของไดเรกทอรี home ของผู้ใช้ที่เราจะใช้งาน

การใช้งานโปรแกรม LAM MPI

1. การใช้งานก่อนการรันโปรแกรมแบบขนานนั้น เราต้องทำการสร้างเส้นทางการเชื่อมต่อในการประมวลผลขึ้นมาก่อนโดยใช้คำสั่ง

   lamboot

โดยคำสั่งนี้จะเป็นการสร้างการเชื่อมต่อของเครื่องคอมพิวเตอร์หรือทอโปโลจีที่เราต้องการนำมาร่วมในการประมวลผลด้วย คำสั่งนี้ถ้ารันโดยปกติจะทำการเรียกไฟล์ชื่อว่า bhost.def ซึ่งอยู่ในไดเรกทอรี lam/boot ซึ่งลักษณะของไฟล์นี้จะเก็บชื่อเครื่องที่เราต้องการนำมาคำนวณ ในตอนเริ่มแรกไฟล์จะมีลักษณะดังนี้

localhost

ซึ่งในไฟล์ที่มีคำว่า localhost ก็คือหมายถึงจะสร้าง การเชื่อมต่อขึ้นบนเครื่องเดียว โดยถ้าเราต้องการเปลี่ยนเป็นหลายเครื่องก็ทำการแก้ไฟล์นี้ เช่น ถ้าเราใช้เครื่อง node1, node2 และ node3 ในการประมวลผลก็ไปแก้ไฟล์เป็น

localhost

node1

node2

node3

เมื่อแก้ไฟล์แล้วต่อไปถ้าหากเรียก lamboot ก็จะได้ทอโปโลจีเป็นเครื่องทั้งหมดตามไฟล์นั้น

2. การคอมไพล์โปรแกรมแบบขนานที่เราทำขึ้น จะใช้คำสั่ง

hcc <ชื่อไฟล์แบบขนาน >

เช่น hcc test

3. ในการรันโปรแกรมจะใช้คำสั่ง

mpirun –np<จำนวนโพรเซสในการคำนวณ> <ชื่อไฟล์>

เช่น mpirun -np 4 a.out

วิธีการติดตั้งโปรแกรม XMPI

โปรแกรม XMPI เป็นโปรแกรมที่สำหรับตรวจสอบดูการทำงานในแต่ละโหนดว่าใช้เวลาในการคำนวณเท่าใด โดยเราจะสามารถดาวน์โหลดได้ที่เว็ปไซต์

http://www..mpi.nd.edu/downloads/lam/xmpi-2.2-6.3.2.i386.rpm

โดยวิธีติดตั้งก็รันคำสั่ง RPM –ivh xmpi-2.2-6.3.2.i386.rpm

วิธีการติดตั้งโปรแกรม SCMS

โปรแกรม SCMS(Smile Cluster Management System) เป็นโปรแกรมที่ทางมหาวิทยาลัยเกษตรศาสตร์ร่วมมือกับโครงการแบวูลฟ์ของนาซ่า ร่วมกันจัดทำขึ้น โดยเป็นโปรแกรมที่ช่วยในการบริหารและการดูแลระบบคลัสเตอร์เพื่อความสะดวกรวดเร็วยิ่งขึ้น โดยสามารถไปดาวน์โหลดได้ที่เว็ปไซต์

http://smile.cpe.ku.ac.th/software/

ซึ่งวิธีการติดตั้งโปรแกรม มีดังนี้

1. ใช้คำสั่ง gzip -dc scms-1.2.2.tar.gz | tar -xvf - or หรือคำสั่ง tar xvzf scms-1.2.2.tar.gz ในการแตกไฟล์

2. ใช้คำสั่ง make ตามลักษณะแพลตฟอร์มที่เราใช้อยู่

- ถ้าเป็น x86 แพลตฟอร์ม ใช้คำสั่ง make x86 SRC=$PWD
- ถ้าเป็น alpha แพลตฟอร์ม ใช้คำสั่ง make alpha SRC=$PWD

3. ติดตั้งโปรแกรม SCMS ตาม Path ที่เราต้องการโดยใช้คำสั่ง make install DEST= <target directory>

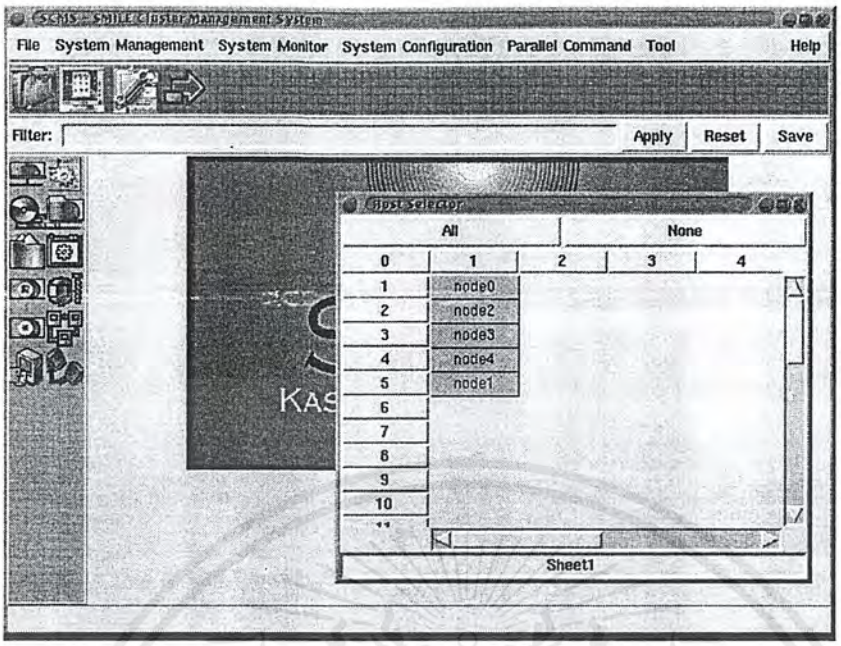
เช่น ถ้าเราใช้คำสั่ง make install DEST=/usr/local/scms โปรแกรมจะถูกติดตั้งลงใน Path /usr/local/scms

4. เพิ่มส่วนของตัวแปร CMS_ROOT ไว้ในจุดที่เราติดตั้งโปรแกรม SCMS โดย

- ถ้าเป็น bash หรือ ksh Shell ให้ใช้คำสั่ง export CMS_ROOT=/usr/local/scms
- ถ้าเป็น c-shell ให้ใช้คำสั่ง setenv CMS_ROOT /usr/local/scms

5. เพิ่มคำสั่ง export PATH=$PATH:$CMS_ROOT/bin ลงในส่วนสคริปท์/etc/bashrc หรือ ~/.bashrc เพื่อให้ครั้งต่อไปสามารถใช้งานโปรแกรมได้ทันที

6. การเรียกใช้โปรแกรมใช้คำสั่ง scms และจะปรากฏหน้าจอโปรแกรมดังรูปที่ ก-2



รูปภาพที่ ก-1 แสดงหน้าจอโปรแกรม SCMS

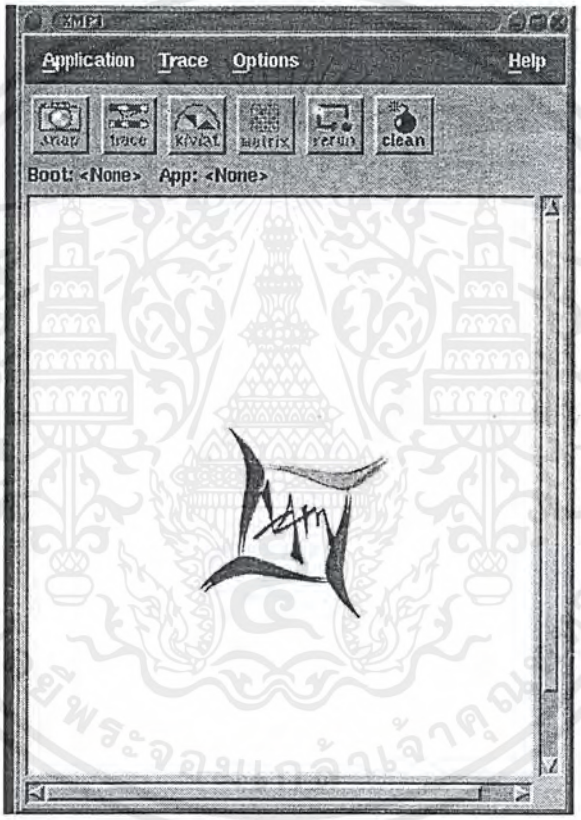
ภาคผนวก ข การวัดความเร็วในแบบขนาน

จะใช้โปรแกรม XMPI ซึ่งเป็นโปรแกรมที่จะแสดงเวลาในการประมวลผลโปรแกรมและสามารถแสดงช่วงเวลาในแต่ละโหนดได้ด้วยว่าในช่วงเวลาขณะใดกำลังทำอะไรอยู่ถึงช่วงเวลาไหนถึงเวลาเท่าไร ซึ่งเป็นโปรแกรมที่มีประโยชน์ในการวัดความเร็วเป็นอย่างมาก และนอกจากนี้ยังเป็นดีบักเกอร์ของโปรแกรมที่เขียนด้วย MPI อีกด้วย

การใช้งานโปรแกรม XMPI

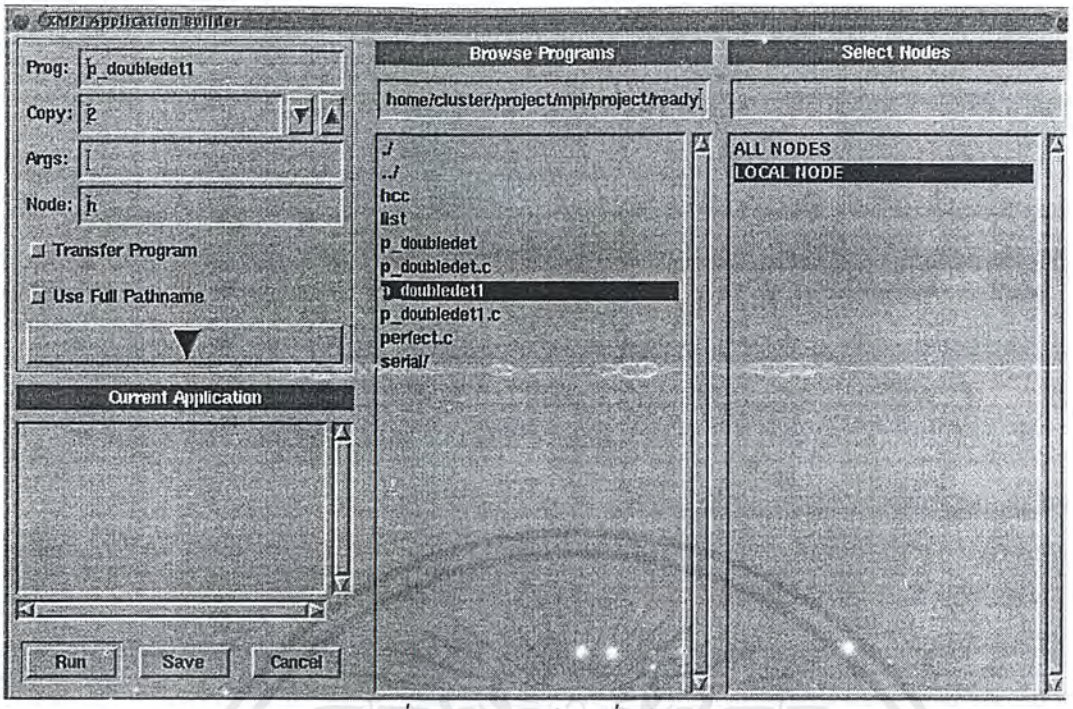
1. เริ่มใช้งานโปรแกรมโดยใช้คำสั่ง

   $ xmpi จะเห็นหน้าจอดังรูปที่ ข-1



รูปภาพที่ ข-1 แสดงหน้าจอการใช้งาน MPI

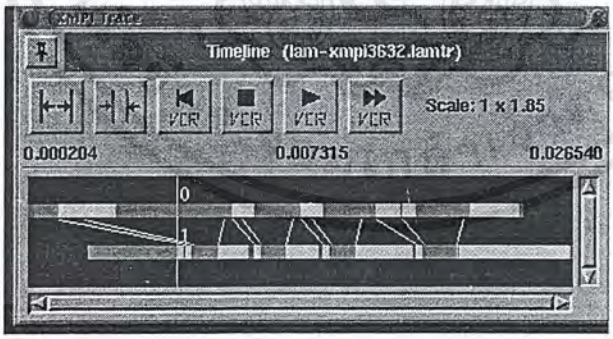
2. ในส่วนหน้าจอนี้เราสามรถเลือกที่จะรันโปรแกรมแบบขนานได้โดยเลือกเมนู Application จากนั้นก็เลือก Build & Run จะขึ้นหน้าจอดังนี้



รูปภาพที่ ข-2 แสดงหน้าจอเมื่อเลือก Build & Run

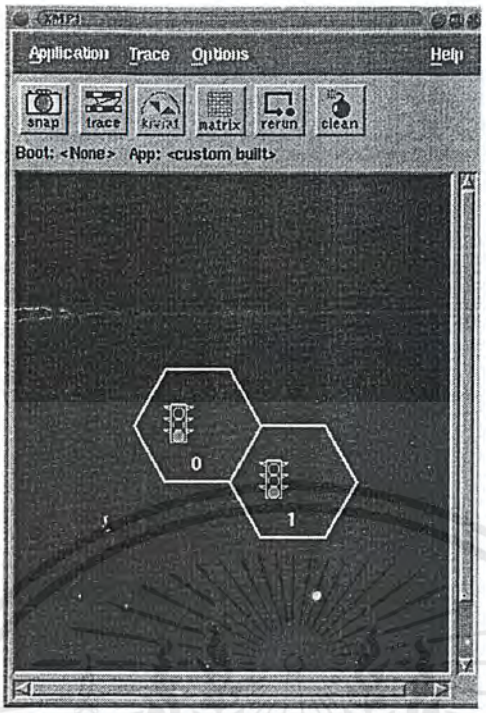
ซึ่งในหน้าจอนี้จะให้เราเลือกโปรแกรมที่จะนำมาเล่นและดูการทำงานโดยสามารถรันโปรแกรมแบบขนานได้ทั้งในโหนดเดียว และ หลายโหนดโดยเลือก ALL Nodes หรือ Local Node พร้อมทั้งยังสามารถเลือกจำนวนโพรเซสได้จากจำนวนก็อปปี้

3. เมื่อรันโปรแกรมแล้วเราจะได้หน้าจอที่แสดงสถานะการทำงานของแต่ละโพรเซสดังรูปที่ ข-3



รูปภาพที่ ข-3 แสดงหน้าจอช่วงเวลาในการทำงานของแต่ละโพรเซสที่เวลาใด ๆ

4. หากเราต้องการที่จะดูเวลาในการทำงานของแต่ละโพรเซส ที่เวลาขณะใด ๆ เราสามารถเลือกดูได้ที่ไอคอน Trace บนหน้าจอแรกจะได้ผลดังรูปที่ ข-4



รูปภาพที่ ข-4 แสดงหน้าจอสถานะการทำงานของแต่ละโพรเซส

การวัดผลในส่วนโปรแกรมซีเควนเชียล

จะใช้การวัดผลจากคอมไพเลอร์ gcc ที่สามารถใช้เลือกพารามิเตอร์ตอนคอมไพล์โปรแกรมเลย โดยคำสั่ง

```
$ gcc –pg  program.c
```

หลังจากนั้นจะได้ไฟล์ a.out ออกมา ซึ่งเป็นโปรแกรมที่สามรถรันให้ทำงานได้ออกมาเมื่อรันแล้วจะได้ไฟล์ชื่อ gmon.out ออกมาต่อจากนั้นเมื่อเราสั่งรัน

```
$ gprof
```

จะแสดงผลของเวลาในการทำงานของโปรแกรมนั้นออกมา รวมถึงยังแสดงเวลาของการทำงานในแต่ละฟังก์ชันย่อยของโปรแกรม ลักษณะการแสดงผลเป็นดังนี้

time	seconds	seconds	calls	ms/call	ms/call	name
100.00	1.89	1.89	28960	0.07	0.07	cut
0.00	1.89	0.00	1	0.00	1890.00	deter
0.00	1.89	0.00	1	0.00	1890.00	read

คอลัมภ์ที่สำคัญ ๆ นั้นจะมี name ซึ่งเป็นชื่อฟังก์ชันแต่ละฟังก์ชันในโปรแกรม คอลัมภ์ seconds แรกจะแสดงถึงเวลาที่ใช้ในแต่ละฟังก์ชันรวมกับฟังก์ชันข้างบนด้วย ส่วนในคอลัมภ์ seconds ที่สองจะแสดงเวลาที่ทำงานในแต่ละฟังก์ชันอย่างเดียวเท่านั้น

ภาคผนวก ค ซอร์สโค้ดของโปรแกรม

แบ่งเป็น 3 ไฟล์ ดังนี้คือ

1. **s_dou.c** เป็นโปรแกรมหาค่าดีเทอร์มินันท์ซึ่งเขียนด้วยวิธีแบบซีเควนเชียล มีซอร์สโค้ด ดังนี้

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define x 10

double deter(double buff[x][x],int R);
int CA,RA,CB,RB;

int main()
{
 read();
 return 0;
}

int read(void)
{
 double buffer1[x][x],sumdet,tmp,buffer[x][x];
 FILE *fp;
 char ch,temp[90];
 int i,j,k,l,is_M2;
 for(l =0;l<90;l++) temp[l] = '&';
 for(i=0; i < x; i++)
  for(j=0; j < x; j++)
        buffer[i][j] = 0;
 if((fp = fopen("list","r")) == NULL)
  {
  puts("cannot open file\n");
  exit(1);
  }
 i=0; j=0;is_M2=0;
 ch = getc(fp);
```

```
while (ch != EOF)
 {
if (ch != '\n')
  {
        if (ch != ' ')
         {
          k =0;
          while (ch != ' ' && ch != '\n')
           {

                temp[k] = ch;
                k++;
                ch = getc(fp);
           }
          buffer[i][j]=atof(temp);
          if (ch == '\n') {
                i++;
                if(is_M2 == 1) CB=j+1;else CA =j+1;
                j=0;
          }else j++;
          for(l =0;l<90;l++) temp[l] = '&';
        }
  }
else
  {
       ch= getc(fp);
        j=0;
        if (ch == '\n') {i++;RA=i-1;is_M2=1;}
       else {
          if (ch != ' ')
                {
                 k=0;
                 while (ch != ' '&& ch != '\n')
                   {
```

```
                temp[k] = ch;
                k++;
                ch = getc(fp);
               }
               buffer[i][j]=atof(temp);
               if (ch == '\n'){
                i++;j=0;
               }else j++;
               for(l =0;l<90;l++) temp[l] = '&';
              }
          }
     }
    ch=getc(fp);
   }
 RB = i-1-RA;
 fclose(fp);
 printf("\n\n          ...Calculating Now...\n");
 sumdet=deter(buffer,CA);
  printf("\n\n          determinant is %f\n\n",sumdet);
  return 0;
}

int cut(int PR,int PC,double buff[x][x],double buff1[x][x],int R)
{ int k,l,i,j,tmp;
k=-1; l=0;
 for (i=0;  i < x; i++)
  for(j=0;   j < x; j++)
    buff1[i][j]=0;

    for(i=0; i < x; i++) { l=0; if ( i != PR) k++;
          for(j=0;   j < x; j++)
            if ((i == PR) || (j == PC))  {}
            else { buff1[k][l]=buff[i][j];
                   l++;  }
```

```
        }   j=0;
    while (buff1[0][j] != 0)
        j++;
    return j;

}

double deter(double buff[x][x],int R)
{ double de[x],buff1[x][x],sdet=0,sps;
  int Co,i,z,j,l,k;
for (z=0; z < x; z++)
  de[z]=0;
  for( i=0; i < R; i++)
  {  sps=1;
    cut(i,0,buff,buff1,R);
    Co = R-1;
     if ((i%2) != 0) sps=-1; else;
   if (Co==2)
    de[i]=sps*buff[i][0]*(buff1[0][0]*buff1[1][1]-buff1[1][0]*buff1[0][1]);
    else de[i]=sps*buff[i][0]*deter(buff1,Co);
  }

for(l=0; l < CA; l++)
  sdet=sdet+de[l];
  return sdet;
}
```

2. **p_dou.c** เป็นโปรแกรมหาค่าดีเทอร์มินันท์ซึ่งเขียนด้วยวิธีแบบขนาน มีซอร์สโค้ดดังนี้

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define x 11

double deter(double buff[x][x],int R);
int RA,RB,CA,CB;

main(int argc,char** argv)
{
 int i,j,l,rank,size,dest,worker,n,extra,R,loop,task,numworker,Co,is_M2,k;
 FILE *fp;
 char ch,temp[90];
 double buffer[x][x],e[x][x],sdet,buff1[x][x],sd[x],det,start,stop,start1,stop1;
 MPI_Status status;
 MPI_Init(&argc,&argv);
 MPI_Comm_rank(MPI_COMM_WORLD,&rank);
 MPI_Comm_size(MPI_COMM_WORLD,&size);



 if (rank == 0)
 {
        start=MPI_Wtime();
        first(temp,fp,buffer);
        task=CA; numworker=size-1;
        loop=task/numworker;
        extra=task%numworker;
        printf("\n\n            ...Calculating Now...\n");
  i=1;
  while (i <= loop)
  {
    if (i==1) n=0; else n=(i-1)*numworker;
```

```
for(dest=1; dest <= numworker; dest++)
  {
  cut(n,0,buffer,buff1,CA);
  Co=CA-1;
  MPI_Send(&Co,1,MPI_INT,dest,10,MPI_COMM_WORLD);
  MPI_Send(&buff1[0][0],x*x,MPI_DOUBLE,dest,10,MPI_COMM_WORLD);
  MPI_Send(&loop,1,MPI_INT,dest,10,MPI_COMM_WORLD);
  MPI_Send(&extra,1,MPI_INT,dest,10,MPI_COMM_WORLD);
  n++;
  }
  if (i==1) l=0;
    else l=(i-1)*numworker;

for(dest=1; dest <= numworker; dest++)
  {
   MPI_Recv(&sd[l],1,MPI_DOUBLE,dest,10,MPI_COMM_WORLD,&status);
   l++;
  }
 i++;
 }

if (extra > 0)
 {
 n=loop*numworker;
 for(dest=1; dest <= extra; dest++)
  {
  cut(n,0,buffer,buff1,CA);
  Co=CA-1;
```

```
      MPI_Send(&Co,1,MPI_INT,dest,10,MPI_COMM_WORLD);
      MPI_Send(&buff1[0][0],x*x,MPI_DOUBLE,dest,10,MPI_COMM_WORLD);
      MPI_Send(&loop,1,MPI_INT,dest,10,MPI_COMM_WORLD);
      MPI_Send(&extra,1,MPI_INT,dest,10,MPI_COMM_WORLD);
       n++;
      }
     if (i==1) l=0;
     else l=(i-1)*numworker;


    for(dest=1; dest <= extra; dest++)
     {
       MPI_Recv(&sd[l],1,MPI_DOUBLE,dest,10,MPI_COMM_WORLD,&status);
       l++;
     }
    }
  for(i =0; i< CA; i++)
   if ((i%2) == 1) sd[i]=-1*sd[i];

  for(i =0; i< CA; i++)
    sd[i]=sd[i]*buffer[i][0];

  det=0;
  for(i =0; i< CA; i++)
  det=det+sd[i];
  printf("\n\n          determinant is %f\n\n",det);
 stop=MPI_Wtime();
  printf("          Response Time = %f\n\n",stop-start);


}
```

```
if (rank > 0)
{ start1=MPI_Wtime(); i=1;
while (i <= loop+1)
 {
 if (i == loop+1)
   if (rank <= extra)
    {  MPI_Recv(&R,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
       MPI_Recv(&e[0][0],x*x,MPI_DOUBLE,0,10,MPI_COMM_WORLD,&status);
       MPI_Recv(&loop,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
       MPI_Recv(&extra,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);


          sdet=deter(e,R);


       MPI_Send(&sdet,1,MPI_DOUBLE,0,10,MPI_COMM_WORLD);
    }
   else ;
else {
       MPI_Recv(&R,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
       MPI_Recv(&e[0][0],x*x,MPI_DOUBLE,0,10,MPI_COMM_WORLD,&status);
       MPI_Recv(&loop,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
       MPI_Recv(&extra,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);


          sdet=deter(e,R);


       MPI_Send(&sdet,1,MPI_DOUBLE,0,10,MPI_COMM_WORLD);
    }
 i++;
 }
 stop1=MPI_Wtime(); printf("\n   Response time of node %d is %f\n",rank,stop1-start1); }
MPI_Finalize();
```

```
}

int cut(int PR,int PC,double buff[x][x],double buff1[x][x],int size)
{ int k,l,i,j;
    k=-1; l=0;
 for (i=0;  i < x; i++)
  for(j=0;   j < x; j++)
   buff1[i][j]=0;

    for(i=0; i < x; i++) { l=0; if ( i != PR) k++;
          for(j=0;   j < x; j++)
           if ((i == PR) || (j == PC))  {}
           else { buff1[k][l]=buff[i][j];
                  l++;  }
         }   j=0;
   while (buff1[0][j] != 0)
          j++;
   return size-1;

}

double deter(double buff[x][x],int R)
{ double sdet=0,de[x],buff1[x][x];
 int Co,i,z,j,l,k,sps;
for (z=0; z < x; z++)
 de[z]=0;
 for( i=0; i < R; i++)
 {  sps=1;
   cut(i,0,buff,buff1,R);
   Co=R-1;
    if ((i%2) != 0) sps=-1; else;
  if (Co==2)
   de[i]=sps*buff[i][0]*(buff1[0][0]*buff1[1][1]-buff1[1][0]*buff1[0][1]);
   else de[i]=sps*buff[i][0]*deter(buff1,Co);
```

```
 }
for(l=0; l < x; l++)
  sdet=sdet+de[l];
  return sdet;
}
int first(char temp[90],FILE *fp,double buffer[x][x])
{
 int i,j,k,l,is_M2;
 char ch;


 for(l =0;l<90;l++) temp[l] = '&';
 for(i=0; i < x; i++)
  for(j=0; j < x; j++)
        buffer[i][j] = 0;
 if((fp = fopen("list","r")) == NULL)
 {
 puts("cannot open file\n");
 exit(1);
 }



 i=0; j=0;is_M2=0;
 ch = getc(fp);
while (ch != EOF)
 {
 if (ch != '\n')
   {
        if (ch != ' ')
         {
          k =0;
          while (ch != ' ' && ch != '\n')
```

```
        {

            temp[k] = ch;
            k++;
            ch = getc(fp);
        }
        buffer[i][j]=atof(temp);
        if (ch == '\n') {
            i++;
            if(is_M2 == 1) CB=j+1;else CA =j+1;
            j=0;
        }else j++;
        for(l =0;l<90;l++) temp[l] = '&';
      }
  }
else
  {
     ch= getc(fp);
      j=0;
      if (ch == '\n') {i++;RA=i-1;is_M2=1;}
     else {
      if (ch != ' ')
          {
           k=0;
           while (ch != ' '&& ch != '\n')
             {
              temp[k] = ch;
              k++;
              ch = getc(fp);
             }
           buffer[i][j]=atof(temp);
           if (ch == '\n'){
            i++;j=0;
           }else j++;
```

```
                    for(i =0;i<90;i++) temp[i] = '&';
                }
            }
        }
    ch=getc(fp);
    }
 RB = i-1-RA;
fclose(fp);
return 0;
}
```

3. **matrix.c** เป็นโปรแกรมที่เขียนขึ้นเพื่อทดลองใช้งานจริงบนระบบคลัสเตอร์ และเพื่อเป็นแนวทางในการเขียนและพัฒนาโปรแกรมเชิงขนานขึ้นมาใช้งานต่อไป โดยจะประกอบด้วยฟังก์ชันการคำนวณทางเมตริกซ์ ซึ่งได้แก่ บวก, ลบ, คูณ และ ดีเทอร์มินันท์ โดยมีซอร์สโค้ดดังนี้

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define x 10

double deter(double buff[x][x],int R);
int CA,CB,RA,RB;
int loop1,extra1;

main(int argc,char** argv)
{
 int i,j,l,rank,size,dest,worker,n,extra,R,loop,task,numworker,Co,k,check,sign=4;
 FILE *fp;
 char ch,temp[90];
 double buffer[x][x],e[x][x],sdet,buff1[x][x],sd[x],det,result[x][x],start,stop;
 MPI_Status status;
 MPI_Init(&argc,&argv);
 MPI_Comm_rank(MPI_COMM_WORLD,&rank);
 MPI_Comm_size(MPI_COMM_WORLD,&size);



/*------------Master--------------*/

 if (rank == 0)
 {
        start=MPI_Wtime();
        sign=first(temp,fp,buffer,sign);
          switch(sign) {
      case 1:printf("\n        Operation Addition...\n\n");;break;
```

```
        case 2:printf("\n       Operation Subtraction...\n \n");;break;
       case 3:printf("\n       Operation Multiple...\n\n");;break;
       default:printf("\n      Operation Determinant...\n\n");}

    printf("        Calculating Now...\n\n");

        task=CA; numworker=size-1;
        loop=task/numworker;
        extra=task%numworker;

i=1;
 while (i <= loop)
 {
   if (i==1) n=0; else n=(i-1)*numworker;

for(dest=1; dest <= numworker; dest++)
   {
      MPI_Send(&sign,1,MPI_INT,dest,10,MPI_COMM_WORLD);
      switch(sign) {
                case 1:check=sendAdd(buffer,dest,n,loop,extra);break;
                case 2:check=sendAdd(buffer,dest,n,loop,extra);break;
               case 3:check=sendmulti(buffer,dest,n,loop,extra);break;
               default:check=sendDet(Co,buff1,dest,n,buffer,loop,extra);
        }
   n++;
   }
   if (i==1) l=0;
      else l=(i-1)*numworker;

for(dest=1; dest <= numworker; dest++)
   {

      switch(sign) {
      case 1:check=M_RecvAdd(result,dest,l,&status);break;
```

```
        case 2:check=M_RecvAdd(result,dest,l,&status);break;
        case 3:check=M_RecvMulti(result,dest,l,&status);break;
        default:check=M_RecvDet(sd,dest,l,&status);
          }
      l++;
     }
    i++;
    } /*end while*/


/*----------------extra process of master----------------*/

 if (extra > 0)
 {
   n=loop*numworker;
   for(dest=1; dest <= extra; dest++)
    {   MPI_Send(&sign,1,MPI_INT,dest,10,MPI_COMM_WORLD);
        switch(sign) {
        case 1:check=sendAdd(buffer,dest,n,loop,extra);break;
            case 2:check=sendAdd(buffer,dest,n,loop,extra);break;
        case 3:check=sendmulti(buffer,dest,n,loop,extra);break;
        default:check=sendDet(Co,buff1,dest,n,buffer,loop,extra);
          }
      n++;
     }
   if (i==1) l=0;
   else l=(i-1)*numworker;
   for(dest=1; dest <= extra; dest++)
    {
        switch(sign) {
        case 1:check=M_RecvAdd(result,dest,l,&status);break;
        case 2:check=M_RecvAdd(result,dest,l,&status);break;
        case 3:check=M_RecvMulti(result,dest,l,&status);break;
        default:check=M_RecvDet(sd,dest,l,&status);
```

```
            }
          l++;
        }
      }
 if (sign==4) {
 for(i =0; i<CA; i++)
 { if ((i%2) == 1) sd[i]=-1*sd[i];
   sd[i]=sd[i]*buffer[i][0];

 }
 for(i =0; i<CA; i++)
 det=det+sd[i];
 printf("\n\n       determinant is %f\n\n",det); }
 else
 for(i=0;i < RA;i++)
 { printf("\n");
 for(j=0; j < CB; j++)
    printf("%3.2f ",result[i][j]);
    } printf("\n\n");
stop=MPI_Wtime();
 printf("         Response Time = %f\n\n",stop-start);
}
/*--------------------------------------*/

/*-----------------Slave----------------*/
 if (rank > 0)
 { i=1;
 while (i <= loop+1)
 {
  if (i == loop+1)
    if (rank <= extra) {
          MPI_Recv(&sign,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
        switch(sign) {
        case 1:check=slaveAdd(status);break;
```

```
            case 2:check=slaveSub(status);break;
          case 3:check=slavemulti(&status);break;
          default:check=slaveDet(&status);
            }          }
     else ;
   else {
           MPI_Recv(&sign,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
          switch(sign) {
          case 1:check=slaveAdd(status);break;
              case 2:check=slaveSub(status);break;
          case 3:check=slavemulti(&status);break;
          default:check=slaveDet(&status);
     }
    }
    loop=loop1; extra=extra1;
   i++;
   }
  }
MPI_Finalize();
}


int cut(int PR,int PC,double buff[x][x],double buff1[x][x],int size)
{ int k,l,i,j,tmp;
k=-1; l=0;
 for (i=0;  i < x; i++)
  for(j=0;   j < x; j++)
   buff1[i][j]=0;

    for(i=0; i < x; i++) { l=0; if ( i != PR) k++;
          for(j=0;   j < x; j++)
           if ((i == PR) || (j == PC))  {}
           else { buff1[k][l]=buff[i][j];
                    l++;   }
```

```
        }   j=0;
    while (buff1[0][j] != 0)
        j++;
    return size-1;

}


double deter(double buff[x][x],int R)
{ double sdet=0,de[x],buff1[x][x];
 int Co,i,z,j,l,k,sps;
for (z=0; z < x; z++)
 de[z]=0;
 for( i=0; i < R; i++)
 {  sps=1;
    cut(i,0,buff,buff1,R);
    Co=R-1;
     if ((i%2) != 0) sps=-1; else;
  if (Co==2)
    de[i]=sps*buff[i][0]*(buff1[0][0]*buff1[1][1]-buff1[1][0]*buff1[0][1]);
    else de[i]=sps*buff[i][0]*deter(buff1,Co);
 }
for(l=0; l < x; l++)
  sdet=sdet+de[l];
  return sdet;
}


int sendDet(int Co,double buff1[x][x],int dest,int n,double buffer[x][x],int loop,int extra)
{
     Co=cut(n,0,buffer,buff1,CA);
     MPI_Send(&Co,1,MPI_INT,dest,10,MPI_COMM_WORLD);
     MPI_Send(&buff1[0][0],x*x,MPI_DOUBLE,dest,10,MPI_COMM_WORLD);
     MPI_Send(&loop,1,MPI_INT,dest,10,MPI_COMM_WORLD);
     MPI_Send(&extra,1,MPI_INT,dest,10,MPI_COMM_WORLD);
```

```
    return 1;
}

int M_RecvDet(double sd[x],int dest,int l,MPI_Status status)
{
    MPI_Recv(&sd[l],1,MPI_DOUBLE,dest,10,MPI_COMM_WORLD,&status);
    return 1;
}

int slaveDet(MPI_Status status)
{ int R;
  double sdet,e[x][x];
    MPI_Recv(&R,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
    MPI_Recv(&e[0][0],x*x,MPI_DOUBLE,0,10,MPI_COMM_WORLD,&status);
    MPI_Recv(&loop1,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
    MPI_Recv(&extra1,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);

      sdet=deter(e,R);

    MPI_Send(&sdet,1,MPI_DOUBLE,0,10,MPI_COMM_WORLD);
    return 1;
}

int sendAdd(double buffer[x][x],int dest,int n,int loop,int extra)
{
    MPI_Send(&CA,1,MPI_INT,dest,10,MPI_COMM_WORLD);
    MPI_Send(&CB,1,MPI_INT,dest,10,MPI_COMM_WORLD);
    MPI_Send(&buffer[n][0],CA,MPI_DOUBLE,dest,10,MPI_COMM_WORLD);
    MPI_Send(&buffer[n+RA+1][0],CB,MPI_DOUBLE,dest,10,MPI_COMM_WORLD);
    MPI_Send(&loop,1,MPI_INT,dest,10,MPI_COMM_WORLD);
    MPI_Send(&extra,1,MPI_INT,dest,10,MPI_COMM_WORLD);
    return 1;
}
```

```
int M_RecvAdd(double result[x][x],int dest,int l,MPI_Status status)
{
     MPI_Recv(&result[l][0],CA,MPI_DOUBLE,dest,10,MPI_COMM_WORLD,&status);
     return 1;
}

int slaveAdd(MPI_Status status)
{    double d[x],e[x];
     int l;
     MPI_Recv(&CA,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
     MPI_Recv(&CB,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
     MPI_Recv(&d[0],CA,MPI_DOUBLE,0,10,MPI_COMM_WORLD,&status);
     MPI_Recv(&e[0],CB,MPI_DOUBLE,0,10,MPI_COMM_WORLD,&status);
     MPI_Recv(&loop1,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
     MPI_Recv(&extra1,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);

          for(l=0; l <= CA; l++)
            d[l]=d[l]+e[l];

     MPI_Send(&d[0],CA,MPI_DOUBLE,0,10,MPI_COMM_WORLD);
     return 1;
}

int slaveSub(MPI_Status status)
{    double d[x],e[x];
     int l;
     MPI_Recv(&CA,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
     MPI_Recv(&CB,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
     MPI_Recv(&d[0],CA,MPI_DOUBLE,0,10,MPI_COMM_WORLD,&status);
     MPI_Recv(&e[0],CB,MPI_DOUBLE,0,10,MPI_COMM_WORLD,&status);
     MPI_Recv(&loop1,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
     MPI_Recv(&extra1,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
```

```
            for(l=0; l <= CA; l++)
               d[l]=d[l]-e[l];

      MPI_Send(&d[0],CA,MPI_DOUBLE,0,10,MPI_COMM_WORLD);
      return 1;
}

int sendmulti(double buffer[x][x],int dest,int n,int loop,int extra)
{
      MPI_Send(&CA,1,MPI_INT,dest,10,MPI_COMM_WORLD);
      MPI_Send(&CB,1,MPI_INT,dest,10,MPI_COMM_WORLD);
      MPI_Send(&RB,1,MPI_INT,dest,10,MPI_COMM_WORLD);
      MPI_Send(&buffer[n][0],CA,MPI_DOUBLE,dest,10,MPI_COMM_WORLD);
      MPI_Send(&buffer[RA+1][0],RB*x,MPI_DOUBLE,dest,10,MPI_COMM_WORLD);
      MPI_Send(&loop,1,MPI_INT,dest,10,MPI_COMM_WORLD);
      MPI_Send(&extra,1,MPI_INT,dest,10,MPI_COMM_WORLD);

      return 1;
}

int slavemulti(MPI_Status status)
{  double d[x],e[x][x],c[x];
   int i,j,l;
      MPI_Recv(&CA,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
      MPI_Recv(&CB,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
      MPI_Recv(&RB,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
      MPI_Recv(&d[0],CA,MPI_DOUBLE,0,10,MPI_COMM_WORLD,&status);
      MPI_Recv(&e[0][0],RB*x,MPI_DOUBLE,0,10,MPI_COMM_WORLD,&status);
      MPI_Recv(&loop1,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);
      MPI_Recv(&extra1,1,MPI_INT,0,10,MPI_COMM_WORLD,&status);

            for(j=0; j < CB; j++)
            { c[j]=0;
```

```
            for(l=0; l < CA; l++)
              c[j]=c[j]+d[l]*e[l][j]; }

  MPI_Send(&c[0],CB,MPI_DOUBLE,0,10,MPI_COMM_WORLD);
  return 1;
}

int M_RecvMulti(double result[x][x],int dest,int l,MPI_Status status)
{
     MPI_Recv(&result[l][0],CB,MPI_DOUBLE,dest,10,MPI_COMM_WORLD,&status);
     return 1;
}


int first(char temp[90],FILE *fp,double buffer[x][x],int sign)
{
 int i,j,k,l,is_M2;
 char ch;

/*------open file & initial variable-------*/

 for(l =0;l<90;l++) temp[l] ='&';
  for(i=0; i < x; i++)
  for(j=0; j < x; j++)
        buffer[i][j] = 0;
  if((fp = fopen("list","r")) == NULL)
  {
  puts("cannot open file\n");
  exit(1);
  }
/*--------------------------------*/


/*---read matrix to buffer---*/
```

```
i=0; j=0;is_M2=0;
ch = getc(fp);
while (ch != EOF)
 {
if (ch != '\n')
  {
        if (ch != ' ')
         {
          k =0;
          while (ch != ' ' && ch != '\n')
          {

              temp[k] = ch;
              k++;
              ch = getc(fp);
          }
          buffer[i][j]=atoi(temp);
          if (ch == '\n') {
              i++;
              if(is_M2 == 1) CB=j+1;else CA =j+1;
              j=0;
          }else j++;
          for(l =0;l<90;l++) temp[l] = '&'; /* Refresh Temp */
        }
  }
else
  {
       ch= getc(fp);
        j=0;
     switch(ch) {
     case'+':sign=1;break;
         case'-':sign=2;break;
     case'*':sign=3;break;
     defaults:sign=4;}
```

```
        ch= getc(fp);
            if (ch == '\n') {i++;RA=i-1;is_M2=1;}
          else {
              if (ch != ' ')
                    {
                     k=0;
                     while (ch != ' '&& ch != '\n')
                       {
                        temp[k] = ch;
                        k++;
                        ch = getc(fp);
                       }
                      buffer[i][j]=atoi(temp);
                      if (ch == '\n'){
                       i++;j=0;
                      }else j++;
                      for(l =0;l<90;l++) temp[l] = '&'; /* Refresh Temp */
                    }
              }
     }
    ch=getc(fp);
   }
 RB = i-1-RA;
fclose(fp);

/*-------------------------*/

  return sign;
}
```