ระบบคอมพิวเตอร์ช่วยกันทำงาน

PC CLUSTERING

นายจะเร เตือนสติ รหัส 40360224

นายวิษณุกรณ์ ครีบเขียว รหัส 40360497

นายเดชา กันลา รหัส 40360612

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวร

ปีการศึกษา 2544

ใบรับรองโครงงานวิจัย

หัวข้อโครงงาน ระบบคอมพิวเตอร์ช่วยกันทำงาน (PC Clustering)

ผู้ดำเนินโครงงาน	นายจะเร	เตือนสติ	รหัส 40360224
	นายวิษณุกรณ์	ครีบเขียว	รหัส 40360497
	นายเดชา	กันลา	รหัส 40360612
อาจารย์ที่ปรึกษา	อาจารย์วัชร์วีร์	พืชพันธ์	
สาขา	วิศวกรรมคอมพิวเตอร์		
ภาควิชา	วิศวกรรมไฟฟ้าและคอมพิวเตอร์		
ปีการศึกษา	2544		

.....

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวร อนุมัติให้โครงงานฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร วิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์

คณะกรรมการสอบโครงงานวิจัย

.....ประธานกรรมการ

(อ.สิทธิโชค เชาวกุล)

.....กรรมการ

(อ.ประทีป ตรีรณโอภาส)

.....กรรมการ

(อ.วัชร์วีร์ พืชพันธ์)

.....กรรมการ

(อ.ศิริพร เดชะศิลารักษ์)

สารบัญ

หน้า

สารบัญ(ต่อ)

สารบัญตาราง

สารบัญรูป

รูปที่ หน้า

บทที่ 1
บทนำ

1. ที่มาและความสำคัญของโครงงาน

ในปัจจุบันเทคโนโลยีของคอมพิวเตอร์ได้ก้าวหน้าไปอย่างรวดเร็ว ทำให้ระบบคอมพิวเตอร์ที่มีอยู่ล้าสมัยไปอย่างรวดเร็ว ไม่สามารถรองรับแอปพลิเคชั่นใหม่ ๆ ให้มีประสิทธิภาพได้ ดังนั้นจึงได้เกิดเทคโนโลยีที่เรียกว่า คลัสเตอร์คอมพิวติง ซึ่งก็คือ การนำคอมพิวเตอร์เหล่านั้นมาใช้ให้เกิดประโยชน์สูงสุด โดยการนำเอาคอมพิวเตอร์หลาย ๆ เครื่องมาเชื่อมต่อกันด้วยระบบเครือข่ายความเร็วสูงและใช้การโปรแกรมแบบขนานมาช่วยดึงเอาความสามารถของแต่ละเครื่องออกมา ทำให้ระบบคอมพิวเตอร์ที่สร้างขึ้นมีความเร็วสูงขึ้น และสามารถนำมาประมวลผลการคำนวณที่มีขนาดใหญ่ทางวิทยาศาสตร์และวิศวกรรมศาสตร์ เช่น Finite Element, Simulation หรืออาจจะนำไปใช้เป็นเซอร์เวอร์หลักขององค์กรแทนคอมพิวเตอร์ที่มีราคาแพง ๆ ได้ อีกทั้งระบบคอมพิวเตอร์ที่ถูกสร้างขึ้นด้วยเทคโนโลยีนี้ยังสามารถเพิ่มขีดความสามารถได้อย่างไม่จำกัด

2. วัตถุประสงค์ของโครงงาน

1. เพื่อศึกษาเทคโนโลยีที่จำเป็นในการติดตั้ง ออกแบบและบริหารระบบคอมพิวเตอร์ช่วยกันทำงาน
2. เพื่อเป็นต้นแบบให้โครงงานวิจัยอื่น ๆ ที่ต้องการใช้ระบบคอมพิวเตอร์ช่วยกันทำงาน
3. เพื่อศึกษาเทคโนโลยีในการพัฒนาโปรแกรมประยุกต์ โดยใช้เทคนิคการประมวลผลแบบขนาน

3. ขอบข่ายของโครงงาน

1. สร้างคอมพิวเตอร์โดยใช้เทคโนโลยีพีซีคลัสเตอริง โดยมีลีนุกซ์ เป็นระบบปฏิบัติการ
2. สร้างสภาพแวดล้อมการโปรแกรมแบบขนานบนระบบคอมพิวเตอร์ช่วยกันทำงาน โดยวิธีการโปรแกรมแบบส่งข้อความ (Message Passing Interface หรือ MPI)
3. พัฒนาโปรแกรมเพื่อสามารถทำงานบนระบบคอมพิวเตอร์ช่วยกันทำงาน

4. ขั้นตอนการดำเนินงาน

1. ศึกษาและออกแบบเครือข่ายคอมพิวเตอร์ให้เหมาะสมกับระบบคอมพิวเตอร์ช่วยกันทำงาน
2. สร้างระบบเครือข่ายคอมพิวเตอร์ตามระบบที่ถูกเลือกไว้
3. ติดตั้งระบบปฏิบัติการลีนุกซ์ลงในระบบคอมพิวเตอร์ช่วยกันทำงาน

4. ลงระบบการโปรแกรมแบบส่งผ่านข้อความ (Message Passing Interface) เพื่อสร้างภาพแวดล้อมการโปรแกรมแบบขนาน
5. ทำการพัฒนาโปรแกรมประยุกต์เพื่อใช้ความสามารถของระบบคอมพิวเตอร์ช่วยกันทำงาน

5. ผลที่คาดว่าจะได้รับ

1. ระบบพีซีคลัสเตอริง
2. ได้โปรแกรมประยุกต์ที่สามารถทำงานบนระบบคอมพิวเตอร์ช่วยกันทำงาน
3. เข้าใจระบบการโปรแกรมแบบขนานที่ใช้การโปรแกรมแบบส่งผ่านข้อความ (Message Passing Interface)

6. งบประมาณที่ใช้

ค่าวัสดุอุปกรณ์ 3,000 บาท (สามพันบาทถ้วน)

บทที่ 2

การประมวลผลแบบขนานและการประมวลผลภาพ

1. แบบจำลองคอมพิวเตอร์แบบขนาน(Parallel Computer Model)[1]

การจำแนกระบบคอมพิวเตอร์ของ Michael Flynn: (Flynn's Classification)

จำแนกโดยอาศัยการไหลของชุดคำสั่ง (Instruction Stream) และการไหลของข้อมูล (Data Stream) จำแนกออกได้เป็น 4 รูปแบบดังนี้

1. **SISD** (Single Instruction over Single Data stream) คือ เครื่องคอมพิวเตอร์ที่พบเห็นกันทั่ว ๆ ไป
2. **SIMD** (Single Instruction over Multiple Data stream) คือ เครื่องคอมพิวเตอร์ Vector Computer ซึ่งเป็นเครื่องคอมพิวเตอร์ที่ติดตั้งกับอุปกรณ์ที่เป็น Scalar และอุปกรณ์ที่เป็น Vector
3. **MIMD** (Multiple Instruction over Multiple Data stream) คือ Parallel Computer ที่เรากำลังสร้างและศึกษาอยู่
4. **MISD** (Multiple Instruction over Single Data stream) คือ ลักษณะของข้อมูลที่ไหลผ่านอาร์เรย์ของ processor ซึ่งทำงานกันต่างฟังก์ชัน เครื่องคอมพิวเตอร์ที่มีสถาปัตยกรรมแบบนี้รู้จักกันอีกอย่างหนึ่งว่า "Systolic Array" ใช้สำหรับทำ Pipeline เพื่อทำงานกับอัลกอลิทึมที่เฉพาะเจาะจง

ทั้ง 4 รูปแบบที่ได้กล่าวมา รูปแบบที่เป็นที่นิยมใช้เป็น Parallel Computer มากที่สุด คือ MIMD (General purpose computation) ส่วน SIMD และ MISD จะเหมาะสมกับงานที่ทำเฉพาะเท่านั้น (Special purpose computation) ด้วยเหตุนี้ MIMD จึงเป็นรูปแบบที่นิยมมากที่สุดและลองลงมาคือ SIMD และ MISD ตามลำดับ ส่วนสุดท้ายคือ SISD

ปัจจัยที่ใช้ในการวัด Performance ของระบบ (System Attribute to Performance)

1. **Clock Rate** และ **CPI** (Cycle Per Instruction)

I = รอบเวลาของสัญญาณนาฬิกา (Cycle Time) หน่วย nanosecond

F = ความเร็วของสัญญาณนาฬิกา (Clock Rate) หน่วย megahertz

I_c = จำนวนคำสั่งในโปรแกรมที่ใช้ทำงาน (Instruction)

CPI = จำนวนรอบเวลาของสัญญาณนาฬิกาที่ใช้ในการทำงาน 1 คำสั่งในโปรแกรมที่จะทำงาน

$$F = 1/T \qquad (1)$$

2. ปัจจัยที่ใช้ในการตรวจสอบประสิทธิภาพระบบ (Performance Factors)

T = เวลาของ CPU สำหรับในการทำงานกับโปรแกรม (CPU Time) หน่วย (Second/Program)

$$T = I_C * CPI * Z \qquad (2)$$

เนื่องจากการทำงานกับคำสั่งต้องการ Instruction fetch, decode, operand(s) fetch, execute และ store ผลลัพธ์

$$T = I_C * (p + m*k) * I \qquad (3)$$

เมื่อ

p = จำนวนรอบสัญญาณนาฬิกาที่ใช้ในการ decode และ execute instruction

m = จำนวนรอบสัญญาณนาฬิกาที่ใช้ในการเข้าถึง memory และทำงานแบบ memory จนเสร็จสมบูรณ์

k = ratio ระหว่าง memory cycle กับ Process Cycle

3. **MPI Rate**

$$\text{MPI Rate} = I_C/(T*10^6) = f/(CPI*10^6) \qquad (4)$$

4. **Throughput Rate**

คือ จำนวนโปรแกรมที่ถูกทำงานใน 1 Second

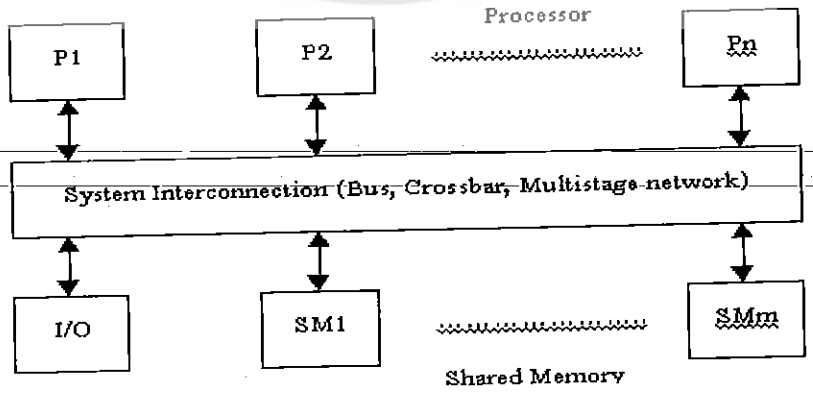
$$WP = f/(I_C*CPI) = (MIPS)*10^6/I_C \qquad (5)$$

Multiprocessor และ Multicomputer

1. **Share-Memory Multiprocessors**

- **UMA Model** (Uniform memory-Access Model)

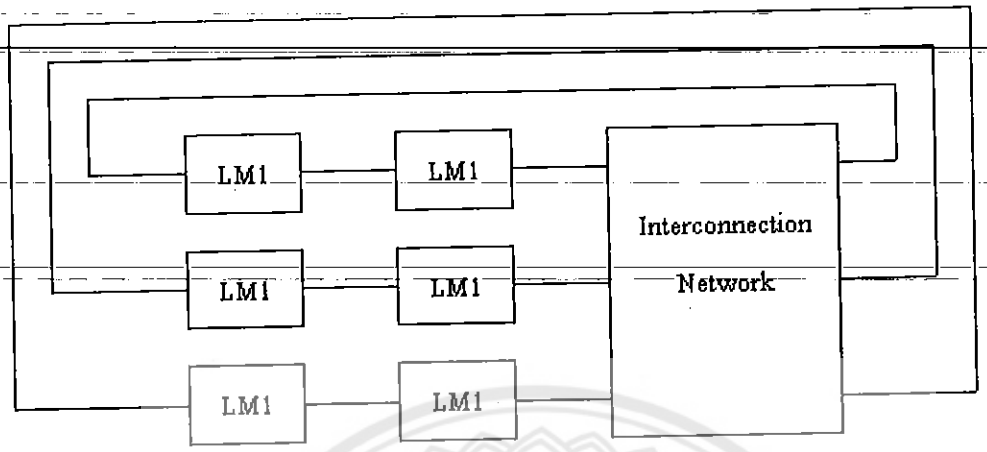
คือ ทุก Processor มีเวลาในการเข้าถึง memory เท่า ๆ กันหมด และทุก Processor จะใช้ cache ส่วนตัว ส่วน Peripherals จะถูก Share ในลักษณะเดียวกันกับ Memory และแต่ละ Processor จะเชื่อมต่อกันโดย Bus, Crossbar Switch หรือ Multistate Network



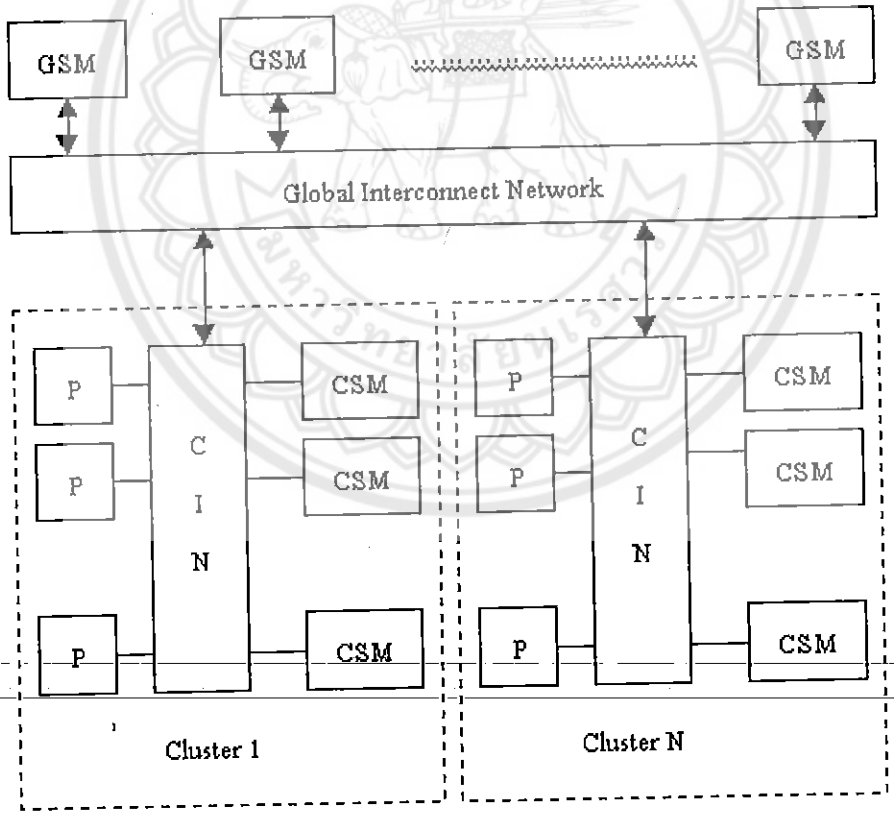
รูปที่ 2.1 แบบจำลองยูเอ็มเอ (UMA Model)

- NUMA Model (Nonuniform-Memory-Access Model)

คือ เวลาการเข้าถึงจะแปรเปลี่ยนไปตามที่อยู่ของ memory ซึ่งจะถูกเรียกใช้ (access) โดย Processor



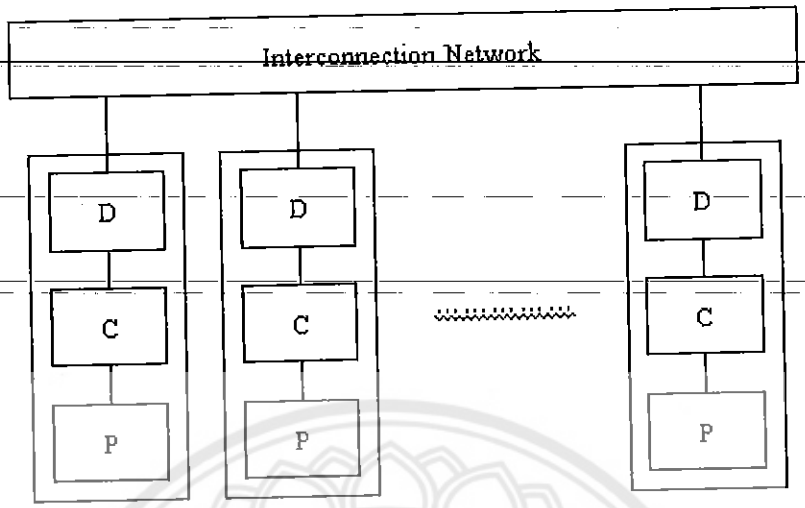
รูปที่ 2.2 การแชร์หน่วยความจำ (Shared local memories)



รูปที่ 2.3 แบบจำลองลำดับชั้นของคลัสเตอร์(A hierarchical cluster model)

2. **COMA Model (Cache-Only Memory Architecture Model)**

COMA Model คือ NUMA Model ที่มีลักษณะพิเศษนั่นเอง คือ Distribute main memories จะถูกเปลี่ยนแปลง เรียกว่า Cache ดังรูป

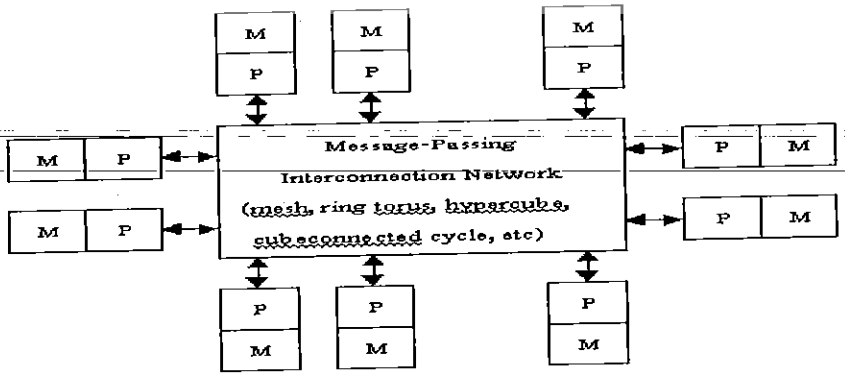


รูปที่ 2.4 แบบจำลองมัลติโพรเซสเซอร์แบบโคมา (The COMA model of a multiprocessors)

Distribute-Memory Multicomputers

ในระบบประกอบด้วยคอมพิวเตอร์หลายเครื่อง จะเรียกเครื่องคอมพิวเตอร์เหล่านี้ว่า "โหนด (node)" จะถูกเชื่อมต่อกันโดย Message Passing Network แต่ละโหนดจะมี Processor, Logical Memory และบางครั้งก็จะมี Harddisk หรือ I/O Peripherals เป็นของตนเอง

แต่ละโหนดจะเชื่อมต่อกันโดยตรง ทุก local memory จะถูกเข้าใช้โดย local access (NORMA) แต่อย่างไรก็ตามก็ได้มีการปรับปรุงให้สามารถใช้งานร่วมกันได้ (Distributed-Shared Memory) ในการสื่อสารระหว่างโหนดการสื่อสารผ่านระบบเครือข่าย (Message Passing)



รูปที่ 2.5 การกระจายหน่วยความจำ (Distribute-memory Multicomputer)

2. การส่งผ่านข้อความ(Message Passing Interface (MPI))[1]

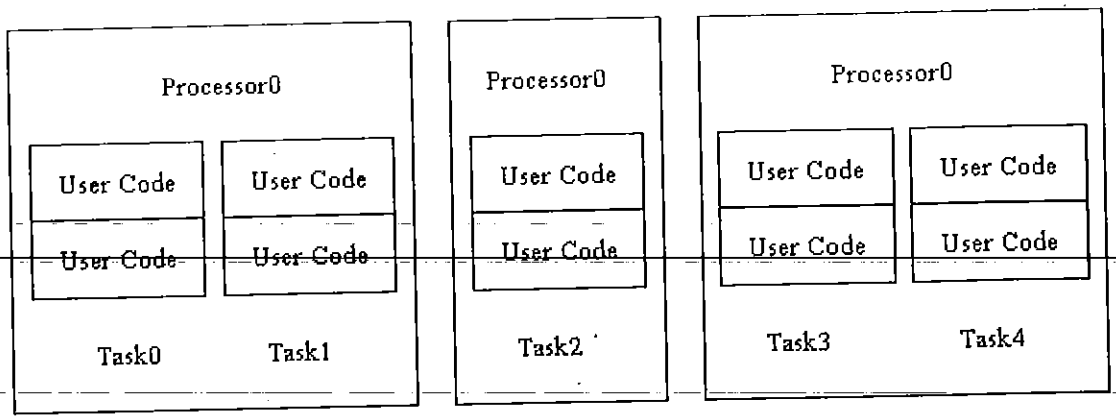
MPI เป็นมาตรฐานที่ใช้ในการเขียนโปรแกรมในเชิงลักษณะแบบส่งผ่านข้อความ (Message Passing Program) ซึ่ง MPI ได้ถูกเริ่มต้นพัฒนาเมื่อปี ค.ศ. 1933 และ ค.ศ. 1944 โดยกลุ่มของนักวิทยาศาสตร์ เจ้าของผู้ผลิตคอมพิวเตอร์ และนักพัฒนาโปรแกรมคอมพิวเตอร์ โดยเรียกตัวเองว่า "MPI Forum" จุดมุ่งหมายของ MPI คือ เพื่อจัดเตรียม library ของ routine มาตรฐาน มีไว้สำหรับเขียนโปรแกรมแบบส่งผ่านข้อความที่สามารถ portable และมีประสิทธิภาพได้ MPI นั้นไม่ใช้ภาษาคอมพิวเตอร์แต่อย่างใด แต่ MPI เป็นข้อกำหนดของ library ของ routine ที่สามารถถูกเรียกจากโปรแกรมอย่างภาษา C หรือ ภาษา FORTRAN77

MPI ได้จัดเตรียม routine ที่ใช้ในการสื่อสารแบบจุดต่อจุด (Point-to-Point Communication Routine) และการสื่อสารแบบกลุ่ม (Collective Communication Routines) การคำนวณทั่วไป (Global Computation) และการทำซิงโครไนเซชัน (Synchronization) MPI ได้กำหนดคุณลักษณะที่สำคัญ ๆ อีกหลายอย่าง เช่น derived data type และการสร้าง communication contexts

ในปัจจุบัน MPI Forum ได้ทำการพัฒนา MPI จนถึงเวอร์ชัน 2 (MPI-2) ซึ่งมีคุณสมบัติพิเศษที่เพิ่มเข้ามาจากเวอร์ชัน 1.0 (MPI-1.0) และเวอร์ชัน 1.2 (MPI-1.2) คือ dynamic processes สามารถให้การสนับสนุนการโปรแกรมแบบ Client/Server ได้ การสื่อสารทางเดียว (One-Side Communication) การทำ parallel I/O และ nonbrocking Collective Communication Function

ลักษณะของโปรแกรมประยุกต์แบบส่งผ่านข้อความ (MPI Application)

โปรแกรมของ MPI จะเป็นลักษณะเสมือนชุดคำสั่งต่าง ๆ จะทำงานไปพร้อม ๆ กัน โดยโปรแกรมที่เขียนขึ้นเป็นโปรแกรมที่ได้มีการใช้ routine ที่ได้มีการดึงเข้ามาและคำสั่งจะถูกกำหนดด้วยค่าเอกลักษณ์ค่าหนึ่ง (เรียกว่า rank ซึ่งเป็นตัวเลขจำนวนเต็มที่มีค่าอยู่ระหว่าง 0 – (n-1) เมื่อ n คือ จำนวนโพรเซสเซอร์ ในระบบ) โดย rank นี้จะถูกใช้โดยตัวส่ง (sender) ข้อความ ชุดคำสั่งของ MPI สามารถที่จะทำงานบนเครื่องเครื่องเดียวกัน (rank เดียวกัน) หรือ ต่างเครื่องกัน (rank ต่าง ๆ กัน) พร้อมเพียงกันก็ได้ สำหรับโปรแกรมการส่งข้อความไปที่โพรเซสเซอร์ เดียวกัน (sender และ receiver อยู่เครื่องเดียวกันและ rank เดียวกัน) กับการส่งข้อความไปที่ต่างโพรเซสเซอร์ จะมีลักษณะกลไกการทำงานเหมือน ๆ กัน โดย MPI จะเลือกกลไกในการส่งที่มีประสิทธิภาพที่สุดในการสื่อสารที่มีอยู่ในแต่ละเครื่อง



รูปที่ 2.6 ตัวอย่างของงานทั้ง 5 งานที่ทำงานบนโพรเซสเซอร์ 3 ตัว

ในการเรียกใช้ MPI routine ต่าง ๆ ได้นั้น จะต้องมีการเรียกใช้ routine ที่ใช้ในการเริ่มต้นระบบก่อน (Initialization Routine) โดย MPI ได้เตรียม routine ที่ใช้ในการเริ่มระบบ คือ

MPI_Init (&argc,&argv)

ฟังก์ชันนี้จะต้องถูกเรียกเป็นฟังก์ชันแรก ก่อนที่จะสามารถเรียกใช้ MPI routine อื่น ๆ ได้ แต่ก็มีเพียงฟังก์ชันเดียวเท่านั้นที่สามารถเรียกใช้ก่อน MPI_Init() คือ

MPI_Initialize(&Flag)

ฟังก์ชันนี้จะถูกเรียกเพื่อตรวจสอบว่า MPI_Init() นั้นถูกเรียกหรือยัง โดยจะให้ค่า flag ออกมาเป็น True ถ้า MPI_Init() ถูกเรียกแล้ว และ false ถ้า MPI_Init() ยังไม่ถูกเรียก

ในตอนท้ายของโปรแกรมที่เขียนโดยใช้ MPI โปรแกรมต้องทำการเรียก routine

MPI_Finalize()

เพื่อจะ cleanup สถานะ (state) ของ MPI เมื่อ MPI_Finalize() ถูกเรียกแล้วจะไม่มี MPI routine ที่ถูกเรียกได้อีก และก็ต้องมั่นใจว่าการสื่อสารทั้งหมดจะต้องสิ้นสุดลง ก่อนที่ MPI_Finalize() จะถูกเรียก

โดยสรุปโปรแกรม MPI จะมีการเริ่มต้นด้วย MPI_Init() routine และ มีการสิ้นสุดด้วย MPI_Finalize() routine ดังโปรแกรมข้างล่าง

```
#include "mpi.h"
main(int argc,char **argv)
   { // intialize MPI
     MPI_Init(&argc,&argv);
      :
     // Cleanup MPI
      MPI_Finalize();
   }
```

กลุ่มงาน (Task Group)

คำสั่งใน MPI สามารถที่จะถูกเรียกเป็นชื่อกลุ่มได้ (Named Group) กลุ่ม (Group) ใน MPI เป็น Object ที่จะถูก define เป็น Type MPI_Group ไว้ก่อนแล้วที่จะมีการเข้าใช้ ซึ่งการเข้าใช้ Object Group จะต้องใช้ผ่าน handle ของ MPI_Group

Task Group ก็คือ การรวบรวมกันของกลุ่มคำสั่งให้อยู่ใน Group เดียวกันเพื่อที่จะทำ operation ต่าง ๆ ต่อกันและกัน แต่ละคำสั่ง (task) ที่เป็นสมาชิก (member) ใน group จะได้ค่าเอกลักษณ์ที่เรียกว่า rank หรือเป็นเลขแสดงตัวตน โดย rank จะเป็นเลขที่เริ่มจาก 0 และเพิ่มขึ้นเรียงลำดับตามจำนวนของคำสั่งที่อยู่ใน group นั้น ๆ

MPI ได้จัดเตรียมฟังก์ชันที่สามารถจะสร้างกลุ่มใหม่ได้ โดยอาศัยกลุ่มที่มีอยู่ (MPI จะไม่สามารถสร้างกลุ่มใหม่ ๆ ได้ โดยไม่อาศัยกลุ่มที่มีอยู่เดิม) ในตอนเริ่มต้นทุกคำสั่งจะต้องเป็นสมาชิกของกลุ่มพื้นฐานที่มีอยู่แล้ว คำสั่งที่เป็นสมาชิกของกลุ่มที่สร้างขึ้นใหม่แต่ละคำสั่ง ต้องมาจากกลุ่มเดียวกันที่มีอยู่แล้วหรือมาจากต่างกลุ่มกันก็ได้ โดยกลุ่มใหม่จะเป็นการ excluding set ของคำสั่งและ /หรือ โดยการ including set ของคำสั่งกลุ่มที่มีอยู่แล้ว กลุ่มใหม่สามารถสร้างขึ้นจากกลุ่มที่มีอยู่เดิมโดยการ ใช้ทฤษฎีของ set คือ Union, Intersection และ Difference

การสร้างกลุ่มโดยเอ็กคลูสชัน (Creating a group by exclusion)

กลุ่มใหม่สร้างขึ้นโดย excluding คำสั่งบางคำสั่งจากกลุ่มที่อยู่แล้ว โดยการใช้ฟังก์ชัน

MPI_Group_excl(existing_group, size, rank_array, &new_group_

รูปแบบทั่วไป คือ MPI_Group_excl (existing_group, n, ranges, &new_group)

โดย

existing_group	คือ group ที่จะทำการ excluding
size	คือ ขนาดของ array ของ rank ที่จะทำการ excluding
array	คือ rank ของคำสั่ง ซึ่งจะถูก excluding จาก new_group
new_group	คือ group ใหม่ที่สร้างโดย excluding existing_group กับ array

array โดยรูปแบบแล้วจะเป็นลักษณะ (first rank, last rank, stride) ซึ่งก็คือ rank ที่จะไม่ได้เป็นสมาชิกของ new_group

ตัวอย่างการทำเอ็กคลูสชัน

Group G1 ประกอบด้วย 25 rank ซึ่งประกอบด้วย rank 0 – rank 24 เราต้องการที่จะสร้าง Group ใหม่ G2 ซึ่งประกอบด้วยคำสั่งทุกคำสั่งใน G1 ยกเว้นคำสั่งที่มี rank 3, 7, 11, 15, 22, 23 และ 24

เราสามารถที่จะหาคำตอบของปัญหานี้ได้โดยใช้ MPI_Group_excl() (ข้อตกลง G1 และ G2 ต่างก็เป็นสมาชิก MPI_Group)

1. ทำการเตรียม array ของ rank ที่บรรจุ rank ที่เราต้องการ excluding 3, 7, 11, 15, 22, 23 และ 24
2. ทำการเรียกฟังก์ชัน MPI_Group_excl (G1, 7, rank, &G2) โดย G2 จะประกอบด้วย rank สมาชิกที่ไม่ได้เป็นสมาชิกอยู่ใน array rank

ซึ่งการ excluding สามารถทำโดย

1. เตรียม array rank ซึ่งบรรจุ 2 element : (3, 15, 4) และ (22, 21, 1)
2. เรียก MPI_Group_excl(G1, 2, range, &G2)

Array ranges (3, 15, 4) สามารถที่จะ excluding 3, 7 และ 15 ในขณะที่ array ranges (22, 24, 1) สามารถทีจะ excluding 22, 23, 24 ดังนั้น G2 ซึ่งประกอบด้วย rank 1, 2, 4-6, 8-10, 12-14, และ 16-21

การสร้างกลุ่มโดยอินคลูสชัน (Create a group by inclusion)

ในส่วน inclusion จะใช้ฟังก์ชัน

MPI_Group_incl(existing_group, size, rank_array, &new_group)

รูปแบบทั่วไป คือ MPI_Group_incl (existing_group, n, range, &new_group)

existing_group คือ group ที่อยู่แล้วและจะเป็น groupที่จะนำมาทำ excluding

size คือ ขนาดของ array

rank_array คือ array ของ task rank ที่จะนำมาทำ inclusion กับ existing_group เพื่อให้ได้ n ew_group ใน rank_array จะอยู่ในรูปแบบ (first rank, last rank, stride)

new_group คือ group ใหม่ที่สร้างโดย including

ตัวอย่างการทำอินคลูสชัน

Group G1 ประกอบด้วย 25 rank ซึ่งประกอบด้วย rank 0 – rank 24 เราต้องการที่จะสร้าง group ใหม่ G2 โดย G2 ประกอบด้วย rank ที่เป็นสมาชิกของ G1 คือ 1, 2, 4-6, 8-10, 12-14 และ 16-21

เราสามารถสร้าง G2 โดยใช้ฟังก์ชัน MPI_Group_incl()

1. จัดเตรียม rank array ที่ประกอบด้วย 1, 2, 4, 5, 6, 8, 9, 10, 12, 13, 14, 16, 17, 18, 19, 20, 21
2. ทำการเรียกฟังชัน MPI_Group_incl (G1, 17, ranks, &G2) Group G2 ประกอบด้วย rank ที่มีอยู่ใน array rank

ดังนั้นเราจะใช้ฟังก์ชัน MPI_Group_range_incl() ดังนี้

1. จัดเตรียม array ranges ซึ่งประกอบด้วย element: (1, 2, 1),(4, 6, 1),(8, 10, 1) และ (16, 21, 1)
2. เรียกฟังก์ชัน :MPI_Group_range_incl(G1, 5, range,&G2) จากนั้นเราก็ได้ผลลัพธ์ G2 โดยมี rank สมาชิก คือ 1-2, 4-6, 8-10, 12-14 และ 16-21

การสร้างกลุ่มโดยการกระทำแบบเซต (create a group using a set operation)

ให้ group1 และ group2 เป็น group ที่มีอยู่แล้ว MPI ได้จัดเตรียม 3 ฟังก์ชัน เพื่อสร้าง group ใหม่จากการนำ union, intersection และ difference ของ group1 และ group2

(หมายเหตุ group1, group2 และ newgroup ต่างก็เป็น MPI_Group type)

1. **Union**

   MPI_Group_Union(group1,group2,&new_group)

   new_group ประกอบด้วย ทุก rank ที่มีอยู่ใน group1 ตามด้วยทุก rank ที่อยู่ใน group2 ที่ไม่อยู่ใน group1

2. **Intersection**

   MPI_Group_Intersection(group1, group2, &new_group)

   new_group ประกอบด้วย ทุก rank ที่อยู่ใน group1 และ group2 ที่มี ranks อยู่ใน group1

3. **Difference**

   MPI_Group_defference(group1,group2,&new_group)

   new_group ประกอบด้วย ทุก rank ที่มีอยู่ใน group1 ที่ไม่ม่อยู่ใน group2

คอมมูนิเคเตอร์ (Communicators)

Communicator คือ object ที่สามารถเข้าถึงได้โดยผ่าน handle ที่เป็นชนิด MPI_COMM Communicator สามารถจำแนกเป็น

- Intracommunicators: ใช้สำหรับสื่อสาร(communication) ระหว่างคำสั่งต่าง ๆ ใน group เดียวกัน
- Intercommunicators: ใช้สำหรับสื่อสารระหว่างคำสั่งต่าง ๆ ที่อยู่คนละ group กัน

คอมมูนิเคเตอร์โดยปริยาย (Default Communicator)

MPI ได้จัดเตรียม Communicator ให้แล้วโดย Default คือ MPI_COMM_WORLD เมื่อ MPI_Init() ถูกเรียก MPI_COMM_WORLD จะสร้างโดยอัตโนมัติ และจะเป็น group ของ rank ทุก rank ที่มีอยู่ทั้งหมดในระบบขณะนั้น MPI_COMM_WORLD จะมีค่าเหมือนกันในสำหรับทุก ๆ processes และจะไม่สามารถเปลี่ยนแปลงได้ และ MPI ก็ยังได้จัดเตรียม Communicator ที่มีชื่อว่า MPI_COMM_SELE ซึ่งเป็น group ที่ประกอบด้วย rank ที่เรียกว่า communicator เพียง rank เดียวเท่านั้น

ทาสแรงก์ (Task rank)

คำสั่งที่ถูกอ้างอิงโดย communicator จะถูกกำหนดโดยตัวเลข integer ที่มีค่าเพิ่มขึ้นเป็นลำดับโดยเริ่มจาก (0) จนถึง (ขนาด group ของ communicator – 1) ค่าตัวเลขเหล่านั้นจะเรียกว่า rank ซึ่งใช้เป็น

ค่าที่เป็นเอกลักษณ์ ในแต่ละคำสั่งที่อยู่ใน group เดียวกัน แต่ละคำสั่งจะสามารถที่จะหาค่าเอกลักษณ์ (rank) ของตัวเองได้โดยเรียกใช้ฟังก์ชัน MPI_Comm_rank โดย

```
MPI_Comm        communicator; // communicator handle
int  my_rank; // the rank at the calling task
MPI_Comm_rank (communicator, &my_rank);
```

ฟังก์ชันนี้จะให้ค่าของเอกลักษณ์ (rank) ของคำสั่งที่เรียกใน communicator นี้

กลุ่มของคอมมูนิเคเตอร์ (Communicator 's group)

Group ที่มีความสัมพันธ์กับ communicator นี้ นั้นสามารถที่จะใช้ฟังก์ชัน MPI_Comm_group() ในการหาค่าต่าง ๆ ของ group

```
MPI_Comm        communicator; // communicator handle
MPI_Group       corresponding_group; // group handle
MPI_Comm_group (communicator, &corresponding_group);
```

ขนาด group ที่สัมพันธ์กับ communicator สามารถหาได้โดยเรียกฟังก์ชัน

```
MPI_Comm        communicator; // communicator handle
Int number_at_tasks;
MPI_Comm_size(communicator, &number_at_tasks);
```

ตัวอย่างการสร้างกลุ่มของคอมมูนิเคเตอร์

สมมติว่าโปรแกรม MPI ได้ถูกเริ่มต้นด้วย 5 คำสั่ง คือ T0, T1, T2, T3 และ T4 และแต่ละคำสั่งมี rank คือ 0, 1, 2, 3 และ 4 ตามลำดับ ในตอนเริ่มต้น 5 คำสั่ง ทั้งหมดต่างก็ถูกอ้างอิงโดย MPI_COMM_WORLD Communicator

ถ้าคำสั่ง T3 เรียกใช้ฟังก์ชัน

```
MPI_Comm_rank(MPI_COMM_WORLD, &me);
```

ตัวแปร me จะถูกกำหนดค่าให้เป็น 3 ซึ่งก็มีความหมาย คือ คำสั่ง T3 มีเอกลักษณ์ (rank) เท่ากับ 3 ใน group ที่มีความสัมพันธ์กับ MPI_COMM_WORLD Communicator

ถ้าต้องการสร้าง group ที่รวมคำสั่งในโปรแกรมไว้ก็ใช้ฟังก์ชัน ดังนี้

```
MPI_Comm_group (MPI_COMM_WORLD, &World_group)
```

ซึ่งมันจะใช้ default MPI_COMM_WORLD เพื่อที่จะสร้าง group ที่สัมพันธ์กัน โดยให้ชื่อ group ว่า World_group โดยมีสมาชิกดังนี้ T0, T1, T2, T3 และ T4

ถ้าฟังก์ชันนี้ถูกเรียก

```
MPI_Comm_size(MPI_COMM_WORLD, &n)
```

ตัวแปร n จะมีค่าเท่ากับ 5 ซึ่งมีความหมาย คือ ขนาดของ group ที่สัมพันธ์กับ MPI_COMM_WORLD Communicator

การสร้างคอมมูนิเคเตอร์ใหม่ (Creating new communicator)

MPI ได้จัดเตรียมฟังก์ชันเพื่อให้คำสั่งที่สามารถสร้าง communicator ใหม่ได้ โดยใช้ communicator ที่มีอยู่แล้ว (โดย default คือ MPI_COMM_WORLD)

1. MPI_Comm_dup(oldcomm, &newcomm)
   ฟังก์ชันนี้จะสร้าง newcomm จาก oldcomm โดยทำการคัดลอก oldcomm มาทั้งหมด โดย group ต่าง ๆ ที่สัมพันธ์กับ oldcomm ก็จะถูกคัดลอกมาไว้ใน newcomm ด้วย แต่ทั้ง 2 communicator จะเป็นอิสระแก่กัน โดยสิ้นเชิง
2. MPI_Comm_create(oldcomm, group, &newcomm)
   ฟังก์ชันนี้จะสร้าง newcomm จาก oldcomm โดยที่ group ของ newcomm จะเป็น subset ของ oldcomm
3. MPI_Comm_split(oldcomm, split_key, rank_key,&newcomm)
   ฟังก์ชันจะทำการแบ่ง group ที่สัมพันธ์กับ oldcomm ออกเป็น 2 ส่วน (subgroup) โดยแต่ละ subgroup จะประกอบด้วยคำสั่งที่มีค่าเหมือนกันกับ split_key ในแต่ละ subgroup แต่ละคำสั่งจะถูกจัด rank เป็นลำดับ โดยจะถูกกำหนดโดย argument rank_key ดังนั้น subgroup แรกจะไปสัมพันธ์กับ oldcomm และอีก subgroup หนึ่งจะสัมพันธ์กับ newcomm

ตัวอย่างการสร้างคอมนิเคเตอร์ใหม่

สมมติ MPI โปรแกรมถูกเริ่มต้นด้วย 5 คำสั่ง คือ T0, T1, T2, T3 และ T4 โดยแต่ละคำสั่งมี rank คือ 0, 1,2, 3 และ 4 ตามลำดับ ถ้าต้องการสร้าง group ที่มีชื่อว่า small_group ซึ่งมีสมาชิกเพียง 2 ตัว คือ T0 และ T1 communicator ที่สัมพันธ์กับ small_group สามารถถูกสร้างเมื่อฟังก์ชันนี้ถูกเรียก โดยทุกคำสั่ง

MPI_Comm_create(MPI_COMM_WORLD, small_group, &small_comm)

ถ้า T0, T1, T2,T3 และ T4 เรียกฟังก์ชัน MPI_Comm_split() ดังต่อไปนี้

1. T0 เรียกโดย x = 8 และ me = 0
   MPI_Comm_split(MPI_COMM_WORLD, x, me, &newworld)
2. T1 เรียกโดย y = 5 และ me = 1
   MPI_Comm_split(MPI_COMM_WORLD, y, me, &newworld)
3. T2 เรียกโดย x = 8 และ me = 2
   MPI_Comm_split(MPI_COMM_WORLD, x, me, &newworld)
4. T3 เรียกโดย y = 5 และ me = 3

MPI_Comm_split(MPI_COMM_WORLD, y, me, &newworld)

5. T4 เรียกโดย MPI_UNDEFINED และ me = 4

MPI_Comm_split(MPI_COMM_WORLD, MPI_UNDEFINED, me, &newcomm)

ดังนั้น group ที่สัมพันธ์กับ newcomm จะประกอบด้วย {T1, T2} ในคำสั่งของ T0, T2 ในคำสั่งของ T1 และ T3 group ที่สัมพันธ์กับ newcomm จะประกอบด้วย {T1, T3} MPI_COMM_NULL จะถูก return ในคำสั่งของ T4 rank ใหม่ของแต่ละคำสั่ง ในแต่ละ subgroup จะเหมือนกับ rank ที่กำหนดโดย MPI_COMM_WORLD

โทโปรโลยีเสมือน (Virtual Topology)

MPI มี 2 virtual topologies ที่แตกต่างกัน คือ

- Cartesian topology
- Graph topology

โทโปรโลยีแบบคาทีเซียน (Cartesian topology)

จะใช้ MPI_Cart_create() ในการสร้าง Cartesian Structure dimension

MPI_Cart_create(oldcomm, ndims, sizeofdims, periods, mapping, newcomm)

oldcomm	คือ input communicator(handle)
ndims	คือ จำนวนมิติ (2 มิติ หรือ อื่น ๆ)
sizeofdims	คือ ขนาดของแต่ละมิติ
periods	คือ ระบุ structure จะมีการ period หรือไม่ crow period หรือ column period
mapping	คือ จะให้ cartesian structure เปลี่ยนแปลง rank oldcomm หรือไม่
newcomm	คือ handle ของ cartesian structure

ตัวอย่างการสร้างโทโปรโลยีแบบคาทีเซียน (cartesian topology)

สมมติ MPI โปรแกรมถูกเริ่มต้นด้วย 6 คำสั่ง คือ T0, T1, T2, T3, T4 และ T5 ซึ่งแต่ละคำสั่งมี rank 0, 1, 2, 3, 4 และ 5 ตามลำดับ ในตอนเริ่มต้น 6 คำสั่งถูกอ้างอิงโดย MPI_COMM_WORLD ถ้าต้องการ grid structure ขนาด 2 X 3 ด้วยคำสั่งของ MPI_COMM_WORLD โดย grid structure มีชื่อ communicator ใหม่ว่า gridcomm

```
MPI_Comm        gridcomm; // new communicator
int sizeofdims[2];
int periods[2];
int mapping = 0;
```

```
sizeofdims[0] = 2;
sizeofdims[1] = 3;
periods[0] =periods[1] = 0;
MPI_Cart_create(MPI_COMM_WORLD, 2, sizeofdims, periods, mapping, gridcomm);
```

ดังนั้น จะได้ผลลัพธ์ดังนี้

Task rank in the group	Task coordinate in gridcomm
0	(0,0)
1	(0,1)
2	(0,2)
3	(1,0)
4	(1,1)
5	(1,2)

0 (0,0)	1 (0,1)	2 (0,2)
3 (1,0)	4 (1,1)	5 (1,2)

รูปที่ 2.7 โครงสร้าง gridcomm

Retrieval of coordinates and rank

MPI_Cart_rank(communicator, coords, &rank)

ฟังก์ชันให้ค่ากลับมาเป็น rank ของผู้เรียกใช้ใน cartesian structure mapping โดยให้ cartesian coordinate coords เป็น input เข้าไป

ในทางกลับกันสามารถหา cartesian coordinate ที่สัมพันธ์กับ rank ที่ให้เข้าไป

Ndims คือ ขนาดของแต่ละ dimension

โทโปโลยีแบบกราฟ (Graph topology)

MPI ได้เตรียมฟังก์ชัน MPI_Graph_create() เพื่อให้สามารถสร้าง communicator ใหม่ที่มีลักษณะแบบ graph topology

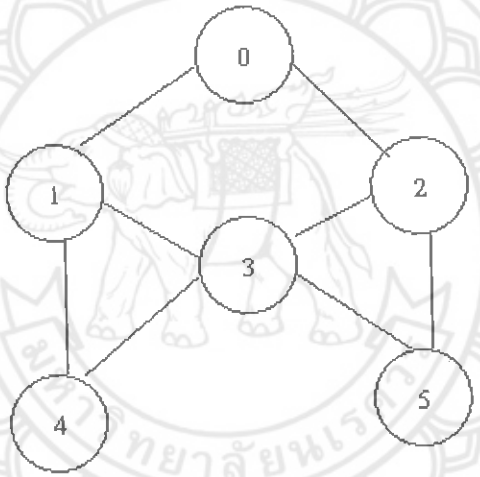
MPI_Craph_create(oldcomm,nnodes, index, edges, mapping, newcomm)

oldcomm	คือ Input communicator (handle)
nnodes	คือ จำนวนของ node ในกราฟ (0 ..... nnonds –1)
index	คือ array ของ integer ที่จะอธิบายถึง degree ของแต่ละ node
edges	คือ array ของ integer ซึ่งอธิบายเส้นของกราฟ
mapping	คือ จะให้ graph structure เปลี่ยนแปลง rank ของ oldcomm หรือไม่
newcomm	คือ handle ของ graph structure

ตัวอย่างการสร้างโทโปโลยีแบบกราฟ

สมมติ MPI เริ่มต้นมี 6 คำสั่ง คือ T1, T2, T3, T4 และ T5 ซึ่งมี rank 0, 1, 2, 3, 4 และ 5 ตามลำดับ

ถ้าต้องการสร้างกราฟที่มีลักษณะดังรูป จาก MPI_COMM_WORLD



รูปที่ 2.8 ลักษณะของ Virtual Topology

สามารถสร้างกราฟจากรูปได้โดยใช้ MPI_Graph_create()

```
MPI_Comm        Graphcomm; // row communicator
int nnodes[6] = {2, 5, 8, 12, 14, 16};
int eages[16] = {1, 2, 0, 3, 4, 0, 3, 5, 1, 2, 4, 5, 1, 3, 2, 3};
int mapping = 0;
MPI_Graph_Create(MPI_COMM_WORLD, nnodes, index, edges, mapping , grapcomm);
```

การสื่อสารคำสั่ง (Task communication)

ตารางที่ 2.1 MPI Data Type

Basic data type

MPI data type	C data type
MPI_BYTE	
MPI_CHAR	Signed char
MPI_DOUBLE	Double
MPI_FLOAT	Float
MPI_INT	Int
MPI_LONG	Long
MPI_LONG_DOUBLE	Long double
MPI_PACKED	
MPI_SHORT	Short
MPI_UNSIGNED_CHAR	Unsigned char
MPI_UNSIGNED_INT	Unsigned int
MPI_UNSIGNED_LONG	Unsigned long
MPI_UNSIGED_SHORT	Unsigned short

การสืบทอดชนิดข้อมูล (Derived data type)

MPI ได้เตรียมฟังก์ชันที่ทำให้ผู้ใช้สามารถสร้าง data type ขึ้นมาเองได้ ซึ่งก็มีอยู่หลายฟังก์ชันด้วยกัน เช่น

MPI_Type_struct(count, block_lengths, diplacements, array_of_types, nuwtype)

count	คือ จำนวนของ element ที่ต้องการจะรวมกลุ่มเข้าด้วยกัน
block_lengths	คือ array ของ integer ที่จะอธิบายถึงจำนวนของ element ในแต่ละ block
displacements	คือ array ของ integer ที่จะอธิบายถึงจำนวนของ element ในแต่ละ block
array_of_types	คือ array ของ handles ของ MPI data type ของแต่ละ block
newtype	คือ handle ของ newtype

ตัวอย่างการสร้างชนิดข้อมูล

ต้องการสร้าง data type ใหม่ ซึ่งประกอบด้วย 3 item คือ

1. 2 element ของ MPI_FLOAT ซึ่ง displacement = 0
2. 1 element ของ type my_type ซึ่ง displacement = 16 type my_type คือ derive type ที่มี type map {(MPI_DOUBLE, 0),(MPI_CHAR, 1)}
3. 4 element ของ type MPI_CHAR ซึ่ง displacement = 26 สามารถใช้ MPI_Type_struct() สร้าง data type ใหม่ ดังนี้

```
count = 3;
block_lengths[3] = {2, 1, 4};
displacement[3] = {0, 16, 26};
array_of_type[3] = {MPI_FLOAT, my_type};
MPI_Type_struct(count, block_lengths, displacements, array_of_type, newtype);
```

ดังนั้นเราจะได้ new type ที่มี type map ดังนี้

{(MPI_FLOAT,0),(MPI_FLOAT,4),(MPI_DOUBLE,16),(MPI_CHAR,24),(MPI_CHAR,26), (MPI_CHAR,27),(MPI_CHAR,28),(MPI_CHAR,29)}

การกระทำแบบคอเล็คชัน (Collective Operation)

Collective Operation ใน MPI คือ การทำ operation ต่าง ๆ ที่มีผลต่อคำสั่ง (task) ที่มีอยู่ใน group ของ communication โดยปกติแล้ว collective operation จะถูกเรียกว่า group of task โดย operation จะถูกกระทำก็ต่อเมื่อทุกคำสั่งที่อยู่ใน group ได้ทำการเรียก collective routine ที่มีค่า parameters นั้น match กับตัวเอง

ใน MPI ได้จำแนก collective operation ออกเป็น 3 ชนิด คือ

- task
- global computation
- data movement

การคำนวณทั่วไป (Global computation)

เป็น operation routine ที่จะถูกเรียกโดย rank ที่มีค่าเท่ากับ 0 หรือ root ส่วน rank ที่มีค่าอื่น ๆ จะไม่สามารถเรียกได้

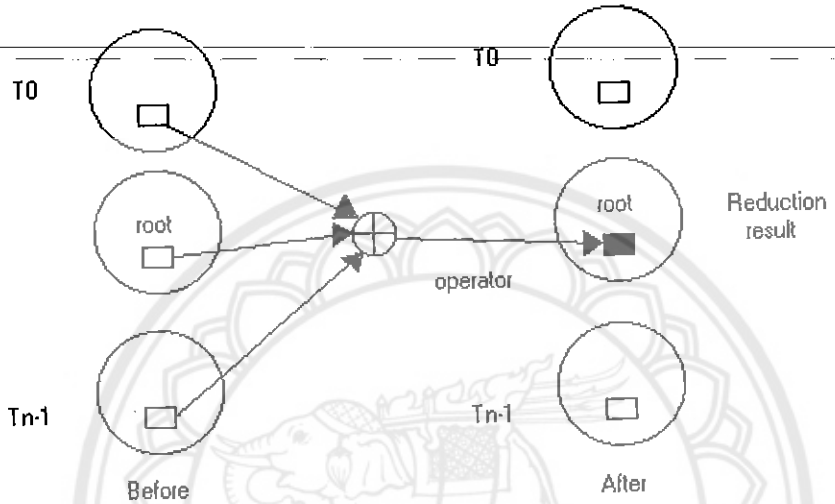
1. **การรวม (Global Combine)**

Reduction function ซึ่งเป็นการรวบรวมข้อมูลจากทุก ๆ rank แล้ว return กลับมาที่ root

MPI_Reduce(sbuf, rbuf, n, data_type, op, vt, communicator)

sbuf คือ ที่อยู่(address) ของ send buffer

rbuf	คือ ที่อยู่(address) ของ receive buffer
n	คือ จำนวนข้อมูลใน send buffer
data_type	คือ ชนิดของข้อมูลใน send buffer
op	คือ reduction operation
rt	คือ rank ของ root
communication	คือ communicator



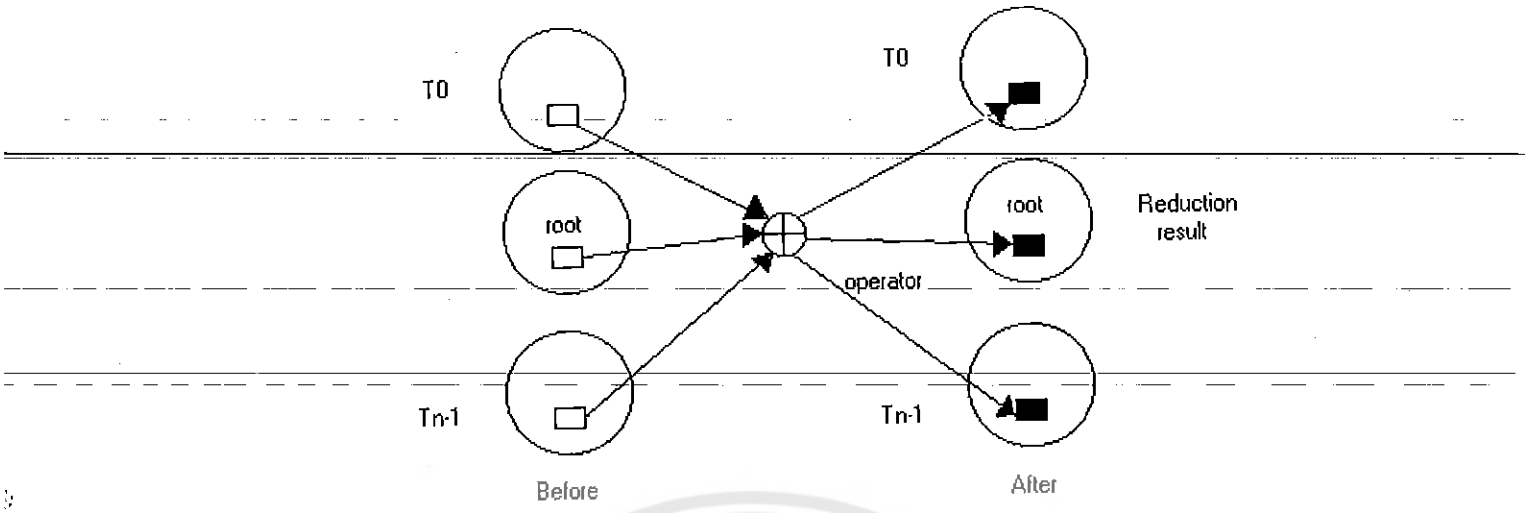
รูปที่ 2.9 การดำเนินการ Reduction

ตารางที่ 2.2 Reduction Operation

MPI name	Operation	MPI name	Operation
MPI_SUM	Sum	MPI_LOR	Logical or
MPI_RPOD	Product	MPI_LXOR	Logical exclusive
MPI_MIN	Minimum	MPI_BAND	Bitwise and
MPI_MAX	Maximum	MPI_BOR	Bitwise or
MPI_LAND	Logical	MPI_BXOR	Bitwise exclusive or

การรวมแบบแมนนีทูแมนนี (Many-to-many reduction)

มีลักษณะเหมือน MPI_Reduce() เพียงแต่ว่าผลของการ reduction จะไปเก็บไว้ทุก ๆ คำสั่ง ดังรูป

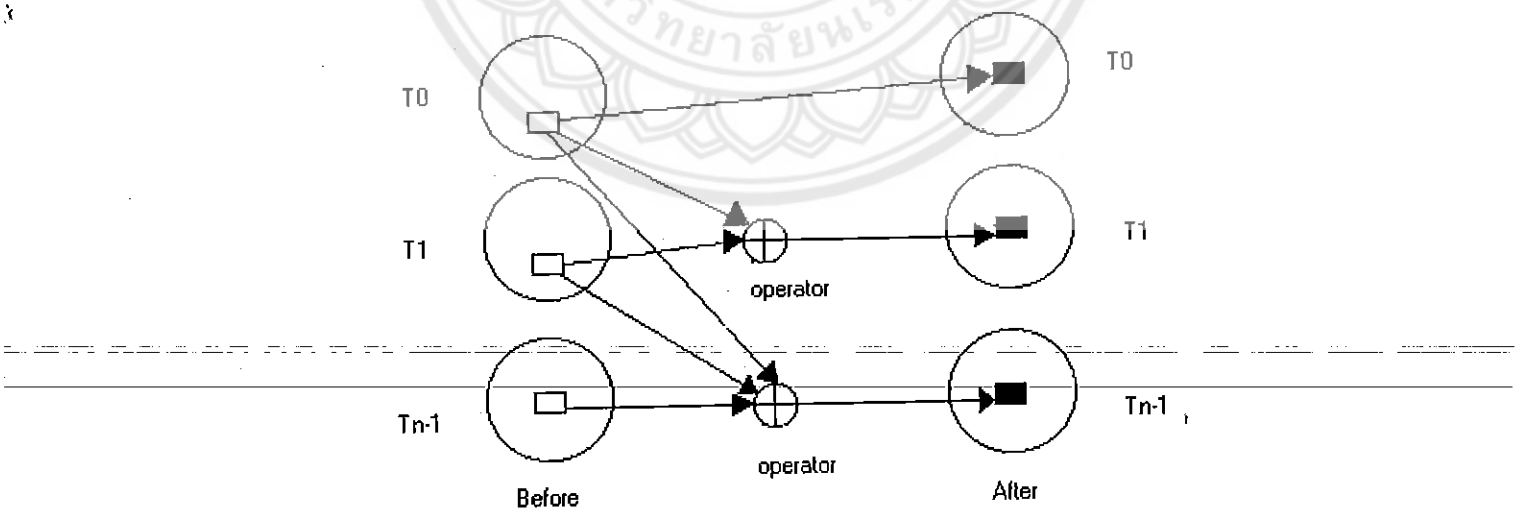


รูปที่ 2.10 All reduce operation

MPI_Allreduec(sbuf, rbuf, n, data_type, op, communicator)

การสแกน (Scan)

จะมีลักษณะเหมือน MPI_Allreduce() แต่ receive buffer ของ rank จะมีค่า reduction เท่ากับ send buffer ของคำสั่งของ rank 0, 1, ……,I ดังแสดงในรูป



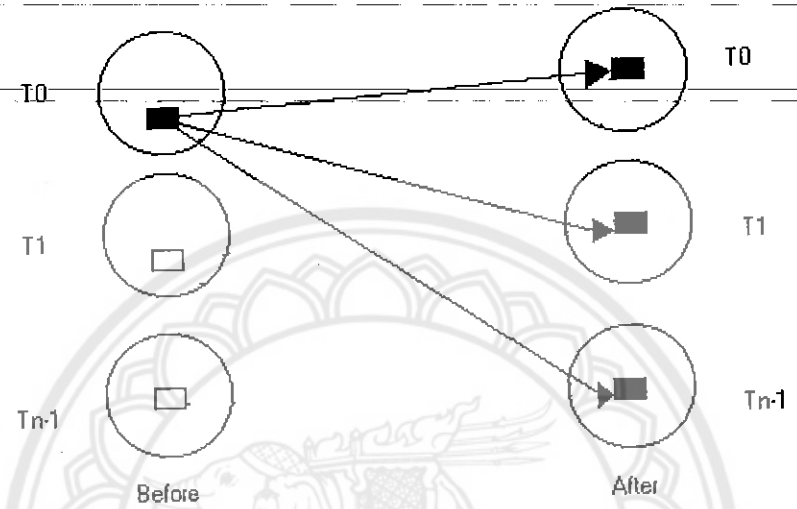
รูปที่ 2.11 การดำเนินการ Prefix scan

MPI_Scan(sbuf, rbuf, n, data_type, op, communicator)

การกระทำการโยกย้ายข้อมูล (Data Movement Operation)

การกระจาย (Broadcast)

Broadcast Message จาก root ไปที่ rank อื่น ๆ ใน group ของ communicator ดังแสดงในรูป



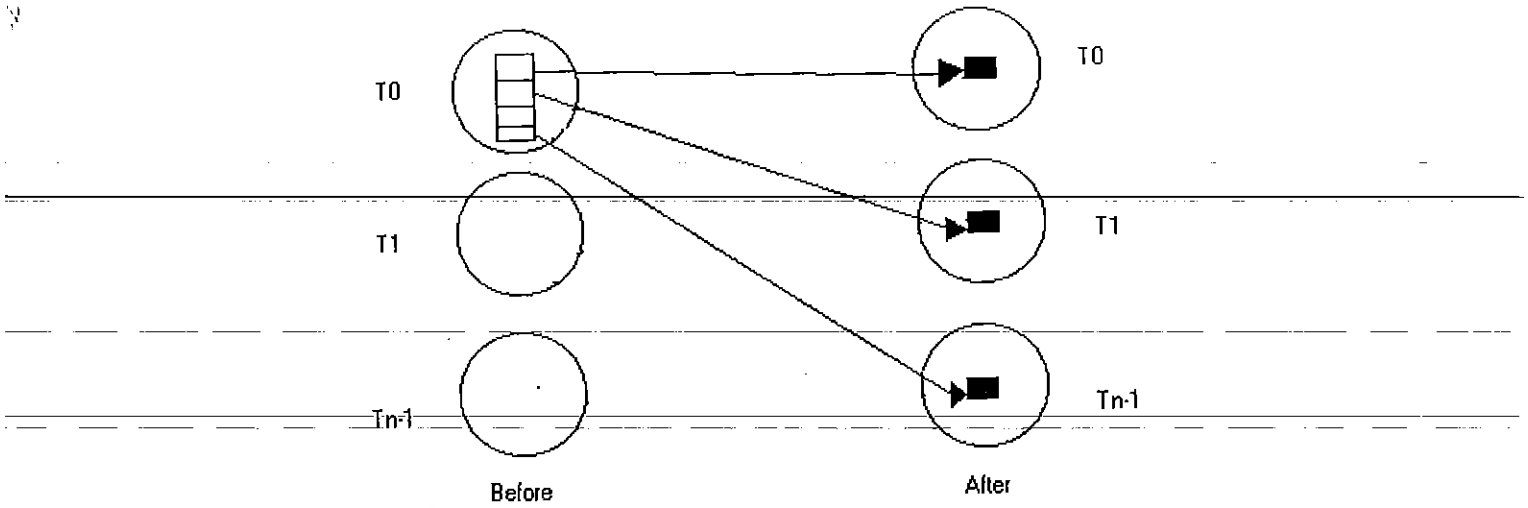
รูปที่ 2.12 การกระจายงานไปยังเครื่องต่าง ๆ (A broadcast from task To root)

MPI_Bcast(buffe, n, data_type, root, communicator)

buffer	คือ ตำแหน่งที่เริ่มต้นของ buffer
n	คือ จำนวนของข้อมูลใน buffer
data_type	คือ ชนิดของข้อมูลใน buffer
root	คือ rank ของ root
communicator	คือ communicator

การสแคทเทอร์และแกทเทอร์ (Scatter and Gather)

MPI_Scatter() เป็นการกระจายข้อมูลจาก root ออกไปที่ rank ต่าง ๆ ใน group ของ communicator โดยข้อมูลที่จะถูกส่งนั้นจะถูกแบ่งย่อยออกเป็นส่วนเท่า ๆ กัน ตามจำนวนของ rank ที่มีอยู่ใน group ดังแสดงในรูป

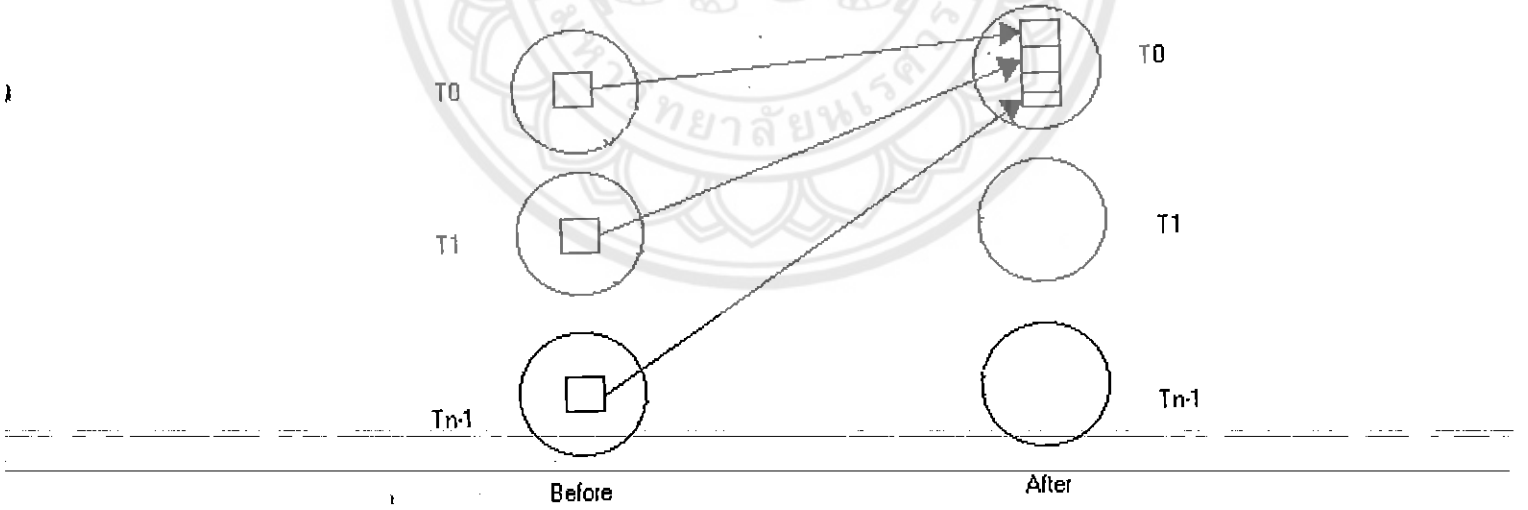


รูปที่ 2.13 การแบ่งส่วนงานไปยังเครื่องต่างๆ (Scatter from the task To)

MPI_Scatter(sbuf, n, stype, rbuf, m, rtype, rt, communicator)

MPI_Gather

เป็นฟังก์ชันที่ทำงานตรงข้ามกับ MPI_Scatter() คือ จะทำการประกอบข้อมูลย่อยจาก rank ต่างๆ ใน group ของ communicator ให้มาประกอบรวมกันที่ root ดังแสดงในรูป



รูปที่ 2.14 การประกอบข้อมูลย่อยไปยังรูท(Gather at the root task To)

MPI_Gather(sbuf, n, stype, rbuf, m, rtype, vt, communicator)

sbuf	คือ ตำแหน่งเริ่มต้นของ send buffer
n	คือ จำนวนของข้อมูลใน send buffer
stype	คือ ชนิดของข้อมูล send buffer
rbuf	คือ ตำแหน่งเริ่มต้นของ receive buffer
m	คือ จำนวนข้อมูลที่จะรับเข้า receive buffer
rtype	คือ ชนิดข้อมูลที่รับเข้าใน receive buffer
rt	คือ rank ของ root
communicator	คือ communicator

โครงสร้างโปรแกรมแบบมาสเตอร์/สลาฟ (Master/Slave application skeleton)

คือ ลักษณะของ MPI application

```
#include<mpi.h>
main(int argc, char *argv)
 {
   int myrank;
   MPI_Init(&argc, &argv); // Initialize MPI
   MPI_Comm_rank(MPI_COMM_WORLD, &myrank); // get my rank
   if(myrank == 0)
      {
         supervisor();
   }
   else{
      workw();
   }
   supervisor()
    {
      .....
      MPI_Comm_size(MPI_COMM_WORLD, &ntask); // get of task
```

```
for (i=1;I<ntask;++i)
  {
    // send some work to worker
  }
worker()
{
  ..............
  // Receive work from the supervisor (rank = 0)
  // do the work
  // Send the result to the supervisor(rank = 0)
}
```

3. การประมวลผลภาพ (Image Processing)[2]

เป็นกระบวนการที่ใช้ในการทำเอฟเฟ็กต์ กับรูปภาพ ด้วยวิธีการการดำเนินการ ซึ่งมีหลายวิธีด้วยกัน เช่น Edge detection, Median filter, Image smoothing, Convolution, Gray scale transformation, Image segmentation, Gaussians operator เป็นต้น ซึ่งรูปภาพที่ผ่านกระบวนการการดำเนินการ แล้วจะมีลักษณะเอฟเฟ็กต์ ต่างๆ ตามที่เราต้องการ การประมวลผลภาพ นำมาใช้งานได้หลายลักษณะ เช่น การกรองการรบกวนในรูปภาพ การทำให้ภาพมีความคมชัดขึ้นหรือมีความกลมกลืน การปรับความสว่างให้กับรูปภาพ การหาขอบของภาพเพื่อใช้ในด้านการดูองค์ประกอบของรูปภาพ มีการนำไปใช้งานหลายลักษณะเช่น ด้านดาวเทียม ด้านการทหาร การค้นหาทรัพยากร การจำแนกวัตถุและแร่ธาตุ การดูแบบแปลนของอาคาร การถ่ายรูป เป็นต้น

ในการบันทึกรูปภาพแบบดิจิตอล จะมีการเก็บในลักษณะจุดของภาพเรียกว่าพิกเซลมีลักษณะเป็นตารางเมตริกซ์ ที่ตำแหน่งพิกเซล (0,0) เป็นการอ้างอิงมุมซ้ายของภาพ มีการนำค่าตัวเลขจำนวนเต็มไปแทนในแต่ละพิกเซล ในการกำหนดระดับสีจะใช้ขนาด 8บิต ซึ่งค่า 0 จะเป็นสีดำ ค่า 255 จะเป็นสีขาว ตัวอย่างการเก็บรูปภาพแบบดิจิตอล

100	52	47	150	200	120	20	8	9	245
147	101	114	117	116	115	110	103	99	122
120	114	112	101	154	101	111	52	58	60
114	80	87	99	150	87	85	20	45	66
154	72	60	88	141	50	60	12	60	52

รูปที่ 2.15 ข้อมูลภาพดิจิตอล

วิธีการประมวลผลภาพ

1. คอนโวลูชัน (Convolution)

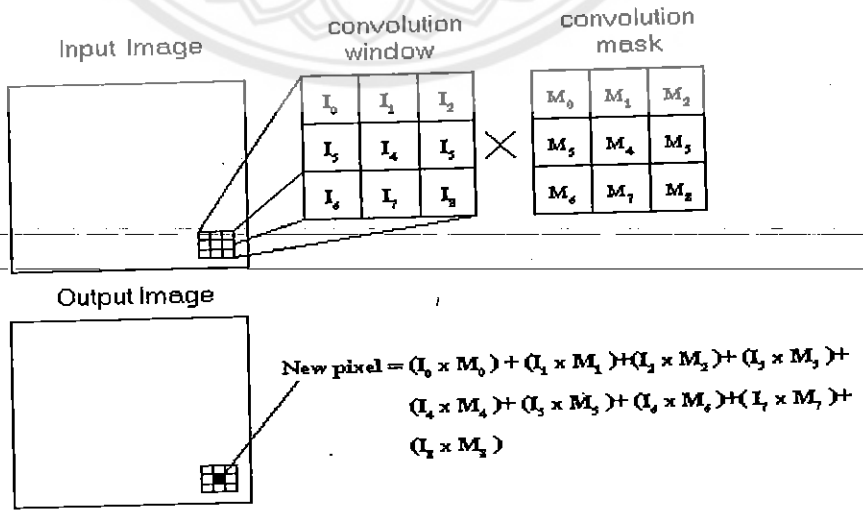
Discrete Convolution จะนำมาใช้มากในการทำการประมวลผลภาพให้ภาพออกมากลมกลืน , การหาขอบภาพและการทำ เอฟเฟ็กต์ต่าง ๆ คอนโวลูชัน จะใช้วิธีการหาผลบวกของพิกเซล ใกล้เคียงกันของของแหล่งกำเนิดพิกเซล จะใช้ convolution mask ขนาดเล็ก ๆ ในการหานำหนักของภาพ โดยทั่วไปจะใช้เมตริกซ์คี่ ดังนั้นจึงใช้จุดกึ่งกลางเมตริกซ์ ในการหาค่าโดยจะนำมาแทนที่เอาท์พุตพิกเซล

โดยจะเลื่อนหน้าต่างของ mask ไปซ้อนทับภาพอินพุต เมื่อประมวลผลแล้วจะได้เอาพุตพิกเซลใหม่ โดยจะใช้ค่าจากพิกเซลใกล้เคียงกับพิกเซล ที่ต้องการนำมาสร้าง

ผลบวกของนำหนักใน convolution mask จะมีผลต่อความสว่าง(intensity) ของภาพ ผลลัพธ์ของ convolution mask จะมีสัมประสิทธิ์ที่มีผลรวมเป็น 1 ในกรณีนี้เป็นการหาค่าเฉลี่ยของความสว่างของภาพที่นำมาใช้ บางชนิด (เช่น การหาขอบภาพ) จะมีค่าสัมประสิทธิ์เป็นค่าลบและมีผลรวมเป็น 0 โดยทั่วไปจะบวกค่าคงที่เข้าไปในการสร้างพิกเซลใหม่ (เช่น intensity สูงสุด / 2) ถ้าผลลัพธ์ออกมาเป็นลบก็ทำการกำหนดค่าเป็น 0

จะเริ่มทำการประมวลผลที่พิกเซล (1,1) แทนพิกเซล (0,0) เพราะถ้าเริ่มที่ (0,0) จะเกิดการ โอเวอร์แลปของภาพและทำการคัดลอกขอบภาพอินพุตมาใส่ที่ขอบภาพเอาท์พุตแทน โดยขอบของภาพเอาท์พุต จะเหมือนกับขอบภาพอินพุต

การขยายรูปเริ่มต้นให้ทำการคัดลอกขอบ โดยใช้ mask ขนาด 3x3 คัดลอกแถวด้านบน, ด้านข้างทางซ้าย, ด้านข้างทางขวา และด้านล่างของภาพ โดยเมื่อทำการคอนโวลูชัน ภาพที่จุด (1,1) จะทำการ คัดลอกขอบบนที่จุด (0,0) เมื่อทำการเลื่อน mask ไปตามแถวก็จะคัดลอกขอบบนที่จุด (1,0) แทนจุด (1,0) ในภาพเอาท์พุต ทำต่อไปเรื่อย ๆ จนสิ้นสุดขอบด้านขวา ทางด้านซ้าย, ด้านขวาและด้านล่างของภาพก็ทำเช่นเดียวกัน



รูปที่ 2.16 การทำคอนโวลูชัน(convolution)

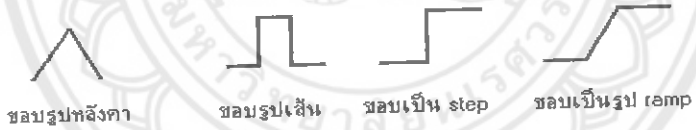
Algorithm ของ Convolution

```
For(i=1; i< nrow –1 ;i++)
 For(j=1; j<ncol –1 ;j++)
  {
    out[i][j] = In[i-1 ][j-1]*mask[0][0]
    +In[I-1][j]*mask[0][1]+In[i-1][j+1]*mask[0][2]
    +In[i][j-1]*mask[1][0]+In[i][j]*mask[1][1]
    +In[i][j+1]*mask[1][2]+In[i+1][j-1]*mask[2][0]
    +In[i+1][j]*mask[2][1]+In[i+1][j+1]*mask[2][2]
  }
```

2. การหาขอบภาพ (Edge Detection)

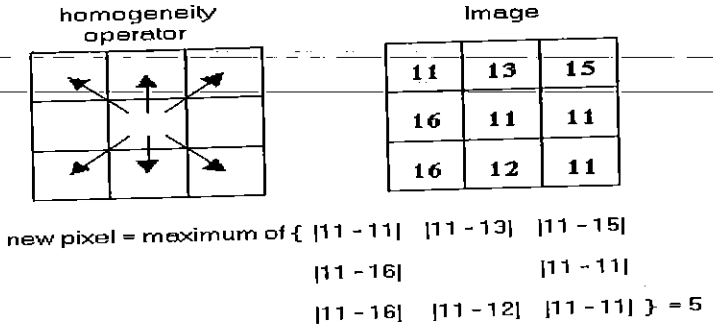
ขอบของภาพจะเป็นตัวบอกข้อมูลต่าง ๆที่อยู่ในภาพ ขอบภาพจะบอกรูปร่างและขนาดของมัน และบอกบางสิ่งที่เกี่ยวกับ texture ขอบของภาพจะเกิดจากการที่ความสว่างเปลี่ยนจากค่าน้อยไปเป็นค่ามาก

วิธีการหาขอบภาพวิธีแรกที่ใช้ในการแบ่งรูปภาพ โดยใช้กลุ่มของพิกเซลในบริเวณเดียวกัน เพื่อหาส่วนประกอบของภาพ



รูปที่ 2.17 รูปแบบของขอบภาพ

วิธีการหาขอบ คือ การหาค่ามากที่สุดจากการนำค่าระดับสีในพิกเซลมาลบกันวิธีการคำนวณดังรูป



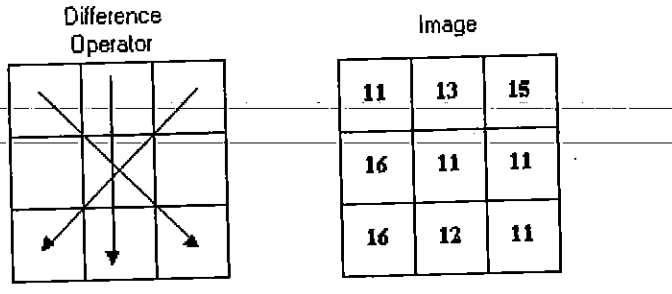
รูปที่ 2.18 การหาขอบภาพโดยใช้ Homogeneity Operator

อัลกอลิทึมการหาขอบภาพแบบใช้ Homogeneity Operator

```
{
        diff = abs(window[4] – window[0])
          max = diff
        diff = abs(window[4] – window[1])
          max = diff
        diff = abs(window[4] – window[2])
          max = diff
        diff = abs(window[4] – window[5])
          max = diff
        diff = abs(window[4] – window[8])
          max = diff
        diff = abs(window[4] – window[7])
          max = diff
        diff = abs(window[4] – window[6])
          max = diff
        diff = abs(window[4] – window[3])
          max = diff
          new_pixel = max+64;
}
```

วิธีที่ 2 คือ difference operator

วิธีการหาเอาท์พุตพิกเซลทำโดยแสดงดังรูป



รูปที่ 2.19 การหาขอบภาพโดยใช้ Difference Operator

อัลกอลิทึมแบบใช้ difference Operator

```
{
  Diff = abs (window [0]-window [8]);
  Max = diff;
  Diff = abs(window[1]-window[7]);
  max = diff;
  diff = abs(window[2]-window[6]);
  max = diff;
  diff = abs(window[5]-window[3]);
  max = diff;
new_pixel = max + 64;
}
```

3. **การหาอนุพันธ์อันดับที่ 1 (First Order Derivative)**

เป็นวิธีการหาขอบภาพตามแนวตั้งและแนวนอน วิธีการในการดำเนินการ คือ การใช้วิธีการแบบคอนโวลูชัน ซึ่งจะใช้ mask ในการนำมาหาขอบ การหาขอบตามแนวนอน จะใช้ Hr และการหาขอบตามแนวตั้งจะใช้ Hc

Hr

$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Hc

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

วิธีการของ Roberts มีผลต่อพื้นที่น้อย จะมีความไวต่อการเกิดการรบกวนมาก

การหาค่าสูงสุด หาได้ดังนี้

$$H(x,y) = \sqrt{Hr^2(x,y) + Hc^2(x,y)}$$

และหาค่า Edge Orientation ได้ดังนี้

$$\theta = \tan^{-1}(Hc(x,y) / Hr(x,y))$$

ใช้ อัลกอลิทึมเดียวกับคอลโวลูชัน

บทที่ 3

วิธีการดำเนินงาน

1.การออกแบบระบบ

เนื่องจากระบบที่คอมพิวเตอร์ช่วยกันทำงานมีความซับซ้อนอยู่พอสมควร ดังนั้นในการออกแบบระบบจึงต้องมีความรัดกุมและมีความตรงไปตรงมา เพื่อที่ว่าเมื่อเกิดปัญหาที่จุดใดจุดหนึ่งทำให้สามารถที่จะแก้ไขปัญหาต่าง ๆ ได้รวดเร็ว ดังนั้นในการออกแบบจึงแยกออกเป็น 5 ส่วน เพื่อให้การติดตั้งทำได้ง่ายขึ้น ดังนี้

- ออกแบบเครือข่าย
- ออกแบบการติดตั้งระบบปฏิบัติการ
- ออกแบบระบบไฟล์
- ออกระบบการจัดการผู้ใช้
- ออกแบบการใช้โปรแกรม MPICH

1.1 ออกแบบเครือข่าย

เนื่องจากระบบคลัสเตอร์ที่สร้างขึ้นมีความจำเป็นที่จะต้องเชื่อมต่อคอมพิวเตอร์เข้ากันด้วยระบบเครือข่ายคอมพิวเตอร์ ซึ่งลักษณะการเชื่อมต่อคอมพิวเตอร์ก็มีหลายลักษณะด้วยกัน เช่น ในเทคโนโลยีของอีเทอร์เน็ต เองก็จะมีการเชื่อมต่อแบบ 10Base5, 10BaseT,100BaseT, 100BaseF , Fast Ethernet, Gigabit Ethernet, 10Gigabit Ethernet ซึ่งในแต่ละแบบก็จะใช้อุปกรณ์เครือข่ายที่เกี่ยวข้องแตกต่างกันไป

นอกจากเทคโนโลยีอีเทอร์เน็ต ก็ยังมีเทคโนโลยีอื่น ๆ ที่น่าสนใจ ได้แก่ เทคโนโลยี Marynet, IBM Token Ring และอื่น ๆ ซึ่งเทคโนโลยีเหล่านี้ก็จะมีข้อดีข้อเสียแตกต่างกันไป

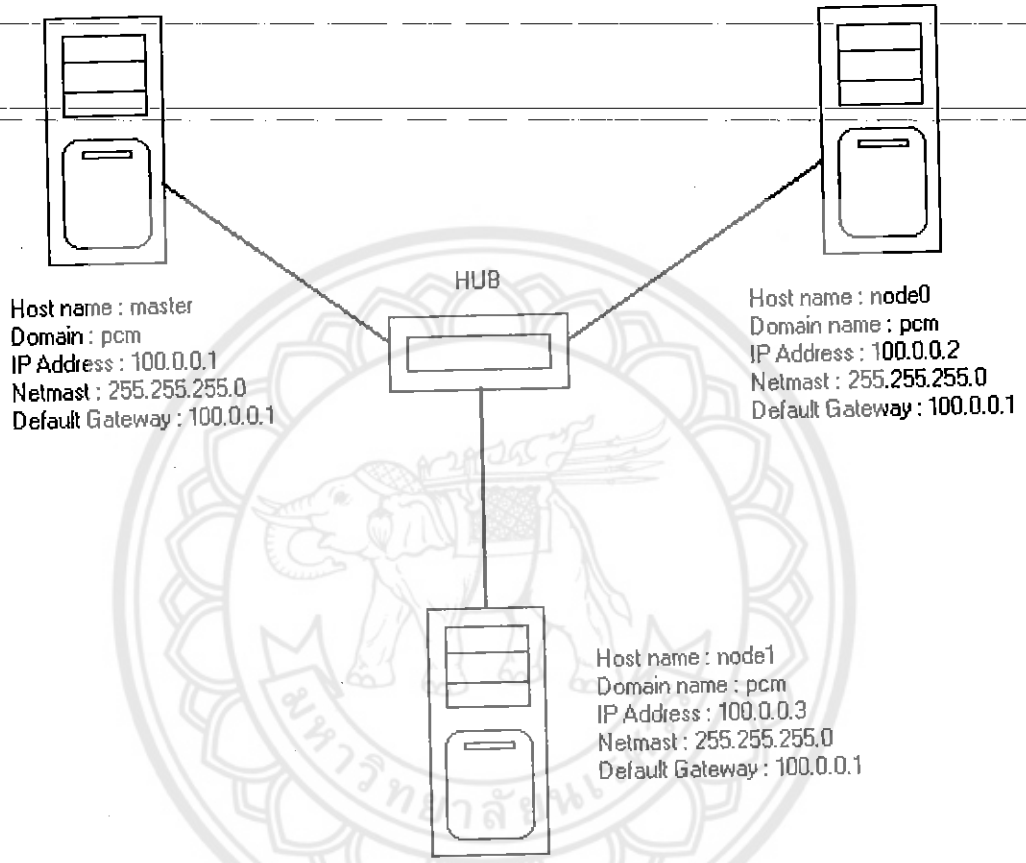
แต่ในระบบที่ใช้ในโครงงานนี้ใช้เทคโนโลยีของอีเทอร์เน็ต ซึ่งเป็นที่นิยมกันอย่างแพร่หลายและมีการติดตั้งที่ง่ายกว่า โดยรูปแบบการเชื่อมต่อจะเป็นแบบ 10BaseT

ลักษณะของ Ethernet 10BaseT

1. ใช้สายUTP-CAT5 ในการเชื่อมโยงระหว่างเครื่องคอมพิวเตอร์
2. ใช้หัว RJ-45
3. Hub 10/100 Mbits
4. Network Interface Card (NIC) 10/100 Mbits
5. ลักษณะการเชื่อมต่อแบบ Star โดยมี Hub เป็นตัวกระจาย Packet

6. ใช้โพรโตคอลสื่อสารในชั้น Data Link Layer คือ IP (Internet Protocol)ในการส่งข้อมูลระหว่างเครื่อง
7. ใช้โพรโตคอลสื่อสารในชั้น Network Layer คือ TCP ซึ่งเป็นโพรโตคอลที่มีความน่าเชื่อถือ ใช้ในการสร้างความน่าเชื่อถือให้กับการรับส่งข้อมูลระหว่างเครื่อง

ดังนั้นในการออกแบบเครือข่ายที่ใช้ในโครงงานนี้ มีลักษณะดังนี้



รูปที่ 3.1 การออกแบบเครือข่าย

ตารางที่ 3.1 คุณสมบัติของเครื่องคอมพิวเตอร์

Specification	Master	Node0	Node1
CPU	Inte Celeron 500 MHz	Inte Celeron 533 MHz	Intel Pentium III 450 MHz
RAM	KingTon 64 MB 100 MHz	- HITACHI 64 MB 133 MHz - HITACHI 32 MB 100 MHz	HITACHI 64 MB 100 MHz
HARDDISK	1.) 2.1 Gbyte Seagate 2.) 2.1 Gbyte Segate	8.4 Gbyte Seagate	8.4 Gbyte Qualtum
KEYBOARD	Standard 101/102 Key	Standard 101/102 Key	Standard 101/102 Key
MOUSE	PS/2 Compatible	PS/2 Compatible	PS/2 Compatible
MONITOR	IBM 15"	ADI 15"	TATUNG 14"
FLOPPY DISK	Standard Floppy disk 1.44 MB	Standard Floppy disk 1.44 MB	Standard Floppy disk 1.44 MB
CD-ROM	AFREEY 50X	AFREEY 50X	ASUS 50X

1.2 ออกแบบการติดตั้งระบบปฏิบัติการ

ในโครงงานได้เลือก Linux Redhat 6.2 เป็นระบบปฏิบัติการเครือข่าย โดยระบบปฏิบัติการได้ทำการติดตั้งบนทุก ๆ node รวมทั้ง master ด้วย แต่จำนวน Packet ที่ใช้ในการติดตั้งจะไม่เท่ากัน เนื่องจาก Harddisk ที่มีไว้สำหรับติดตั้ง Linux Redhat 6.2มีขนาดไม่เท่ากัน

ตารางที่ 3.2 พาร์ติชันที่ใช้ในการลงระบบปฏิบัติการลินุกซ์

	Node0	Node1	Master
เนื้อที่ลง Linux Redhat 6.2	1000 MB	1000 MB	2000 MB
Partition ที่ลงระบบปฏิบัติการ Linux Redhat 6.2	/dev/hda2 ขนาด 1000 MB	/dev/hda2 ขนาด 1000 MB	/dev/hda2 ขนาด 1000 MB
Partition สำหรับทำเป็น home directory (/home)	NONE (Mount จาก Master)	NONE (Mount จาก Master)	/dev/hda5 ขนาด 500 MB
Partition สำหรับเป็นที่เก็บ Solfware ของระบบคลัสเตอร์ (/usr/sofware)	NONE (Mount จาก Master)	NONE (Mount จาก Master)	/dev/hda7 ขนาด 500 MB
Partition สำหรับทำ Swap partion	/dev/hda9 ขนาด 50 MB	/dev/hda9 ขนาด 50 MB	/dev/hda9 ขนาด 50 MB

ตารางที่ 3.3 Packet ที่ลงแต่ละเครื่อง

Packet	Master	Node0	Node1
X Window System	●		
Gnome	●		
Network Workstation	●	●	●
NFS Server	●		
Anonymous FTP Server	●	●	●
Network Management Workstation	●	●	●
Development	●	●	●

1.3 ออกแบบระบบไฟล์

เนื่องจากโปรแกรม MPI ที่เขียนขึ้นและซอร์ฟแวร์ ของ MPI ที่ติดตั้งจริง ๆ จะต้องมีการติดตั้งให้ครบทุกโหนด รวมทั้ง master ด้วย ดังนั้นจึงก่อให้เกิดความยุ่งยากในการติดตั้ง เนื่องจากการติดตั้งทุกเครื่องเป็นการเสียเวลาเป็นอย่างมากและมีความยุ่งยากในการดูแลจัดการ ดังนั้นจึงมีความจำเป็นที่จะต้องลดปัญหาความยุ่งยากเหล่านี้ลง โดยการลงระบบไฟล์แบบ NFS โดยที่เครื่อง master จะทำหน้าที่เป็น NFS Server เพื่อทำหน้าที่ export directory ให้เครื่องอื่น ๆ ในเครือข่ายและในเครื่อง node อื่น ๆ ทำการ mount directory ของตนมาที่เครื่อง master ทำให้เกิดการใช้งานไฟล์และไดเร็คทอรีร่วมกัน เมื่อทำการปรับปรุงแก้ไขก็สามารถทำได้เลยไม่ต้องทำการคัดลอกข้ามเครือข่ายให้ยุ่งยาก

ข้อดีของระบบไฟล์ NFS

1. การจัดการจากศูนย์กลาง
2. การจัดการดูแลทำได้ง่าย
3. เครื่องโหนด ไม่จำเป็นต้องมีพื้นที่เยอะ
4. มีความเป็น Single Image

ข้อเสียของระบบไฟล์ NFS

1. เนื่องจากต้องมีการติดต่อระหว่าง master กับโหนด ตลอดเวลา ดังนั้นจึงอาจจะทำให้เครือข่ายมีความหนาแน่นสูง
2. เครื่องโหนดจะทำงานได้เมื่อ master ทำงานได้เท่านั้น
3. ต้องเปิดเครื่องเสมอตลอดการใช้งาน

ขั้นตอนการออกแบบระบบไฟล์ NFS

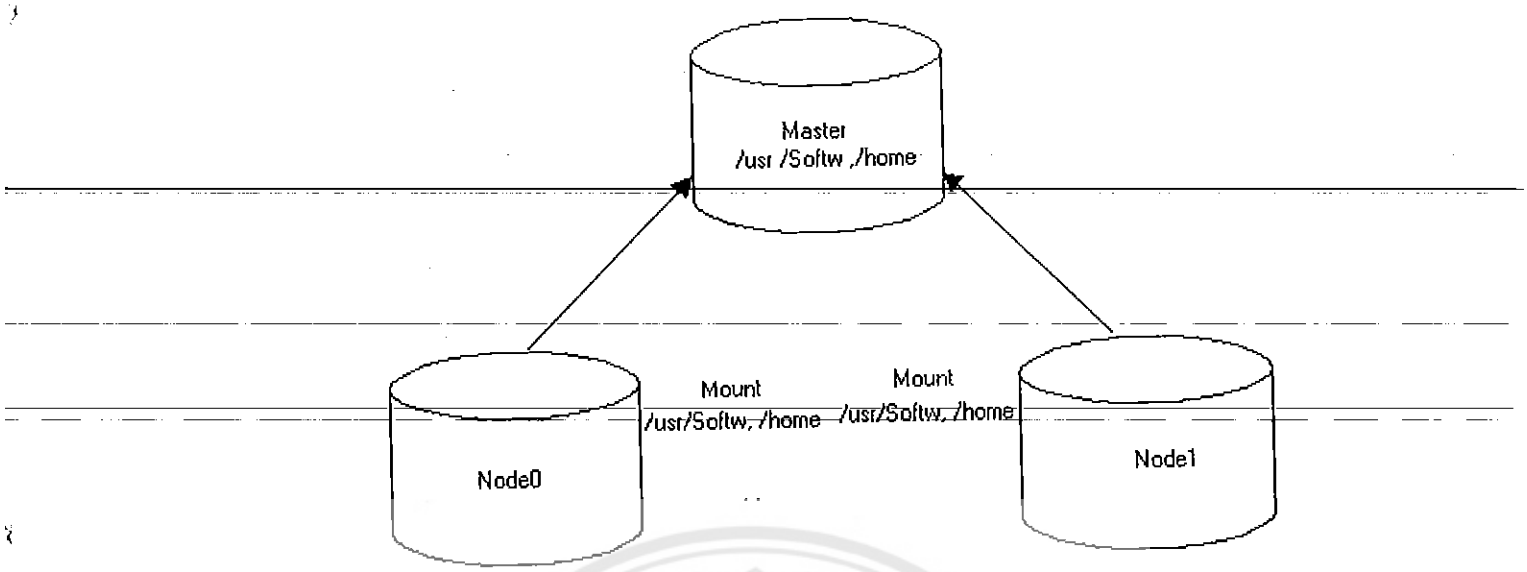
Master

เริ่มที่ master ก่อนจะ export directory เพื่อให้บริการ node ต่าง ๆ ในเครือข่าย โดย directory ที่ทำการ export คือ

/home	เพื่อทำหน้าที่เป็น home directory ของผู้ใช้ระบบทั้งหมด
/usr/Softw	เพื่อทำหน้าที่เก็บ Software ที่จำเป็นในการทำงานในระบบคลัสเตอร์

Node

ทำการ mount /home และ /usr/Softw จาก master เพื่อให้สามารถทำงานร่วมกันได้



รูปที่ 3.2 การลงระบบไฟล์

1.4 การออกแบบระบบการจัดการผู้ใช้

เนื่องจากผู้ใช้มีความจำเป็นที่จะต้องสามารถล๊อกอินไปที่โหนดใด ๆ ได้โดยไม่ต้องใส่รหัสผ่านเพื่อที่ว่าเวลารัน MPI Program ระบบจะได้ไม่ต้องถามพาสเวิร์ด ดังนั้น

1. ผู้ใช้จะต้องมีบัญชีรายชื่อในเครื่องแต่ละเครื่อง
2. ผู้ใช้ต้องสามารถลอกอินไปยังเครื่องทุกเครื่องได้
3. ผู้ใช้สามารถลอกอิน ไปยังเครื่องใด ๆ โดยไม่ต้องใส่รหัสผ่าน

1.5 ออกแบบการใช้โปรแกรม MPICH

เนื่องจากในการจัดการผู้ใช้ เราสามารถที่จะกำหนดให้ผู้ใช้คนใดสามารถใช้ MPICH ได้ดังนี้

1. ทำการเซต PATH ของผู้ใช้ที่สามารถใช้ MPICH ได้ มาที่ Directory ที่เก็บโปรแกรม MPICH
2. ทำให้ผู้ใช้สามารถใช้ MPICH ได้ สามารถลอกอินเข้าไปได้ทุกเครื่องในระบบ

2. ขั้นตอนการติดตั้งระบบ

2.1 ลงระบบปฏิบัติการ

master

ทำการติดตั้งระบบปฏิบัติการ Linux Redhat 6.2 จากซีดีรอมโดยทำการเซตข้อมูลเกี่ยวกับ Network เหมือนกับที่ได้ออกแบบไว้ในหัวข้อออกแบบเครือข่าย สร้างพาร์ติชัน และติดตั้งระบบไปที่ Partition ต่าง ๆ เหมือนกับที่ได้ออกแบบไว้ในหัวข้อออกแบบการติดตั้งและทำการติดตั้ง Packet ตามที่ได้ออกแบบไว้ในหัวออกแบบการติดตั้งและก็ทำการบูตระบบใหม่

Node

ที่เครื่องโหนด เนื่องจากเครื่องโหนดไม่มีซีดีรอม ดังนั้นจึงไม่สามารถติดตั้ง Linux จากซีดีรอมในเครื่องโหนดได้ ดังนั้นในการติดตั้ง Linux ที่เครื่อง node จึงสามารถติดตั้งได้โดยผ่านเครือข่ายโดยใช้ FTP Server และให้เครื่องโหนดเข้ามาข้อใช้บริการ

1. เปิดเครื่อง master แล้วลอกอินเข้ามาในฐานะ root
2. ใส่ซีดีรอม Linux Redhat 6.2 ที่ไดร์ฟซีดีรอมของเครื่อง master
3. ทำการ mount ซีดีรอมโดย

   # mount /dev/cdrom /mnt/cdrom

   #cd /mnt/cdrom

   #cd /images
4. ทำการเขียนไฟล์ bootnet.img จากซีดีรอม ลงในแผ่น floppy disk โดย

   # ใส่แผ่น floppy disk ลงใน floppy drive

   #dd if = bootnet.img of = /dev/fdo obs = 18k
5. เสร็จสิ้นการสร้างแผ่นบูต
6. นำแผ่นบูตที่สร้างขึ้นมาทำการบูตเครื่องโหนด
7. เปิดเครื่องโหนดพร้อมกับเซตให้ทำการบูตจากแผ่น
8. เมื่อถึงขั้นตอนการเซตเน็ตเวิร์ค ของเครื่องโหนด ให้เซตค่าตามที่ออกแบบไว้ในหัวข้อออกแบบเครือข่าย
9. ที่เครื่อง master ให้ทำการ unmount ซีดีรอมโดย

   #unmount /mnt/cdrom

   และทำการ mount ใหม่ โดย mount ไปที่ /home/ftp/pub/linux

   #mkdir /home/ftp/pub/linux

   #mount /dev/cdrom /home/ftp/pub/linux
10. มาที่เครื่องโหนด ; หลังจากนั้นการติดตั้งจะเลือก FTP
11. ทำการให้ IP Address ของเครื่อง FTP Server (master)

12. ใส่ PATH ของ Linux คือ /pub/linux
13. ระบบการติดตั้งก็จะเข้าสู่กระบวนการติดตั้งตามปกติ

ให้ทำการติดตั้ง Packet ที่จำเป็นๆ โดยดูจากหัวข้อออกแบบการติดตั้งและให้ทำการลงกับโหนดทุกโหนดที่มีอยู่ในระบบ

2.2 คอนฟิกระบบ (System Configuration)

เมื่อทำการติดตั้งระบบปฏิบัติการทุกเครื่องเรียบร้อยแล้ว ให้ทำการคอนฟิกระบบโดย

เครื่อง master

1. แก้ไขไฟล์ /etc/hosts โดยมีเนื้อหาดังนี้

   100.0.0.1 master.pcm master

   100.0.0.2 node0.pcm node0

   100.0.0.3 node1.pcm node1
2. สร้าง directory /usr/Solfw/ เพื่อเป็น directory ที่เก็บโปรแกรมต่าง ๆ

   #mkdir /usr/Softw/
3. สร้าง directory /usr/Solfw/mpich เพื่อเป็น directory ที่เก็บไฟล์และโปรแกรม MPICH

   #mkdir /usr/Softw/mpich
4. ทำการเพิ่ม user ชื่อ "upcm" เพื่อเป็น user ที่สามารถใช้ระบบคลัสเตอริงที่สร้างขึ้น

   #adduser upcm
5. ทำการเซตรหัสผ่านให้กับ user upcm โดย

   #passwd upcm

   NEW passwd: <พิมพ์ password แล้ว enter>

   Retry passwd: <พิมพ์ password อีกครั้งแล้ว enter>
6. ทำการ login เป็นผู้ใช้ชื่อ upcm โดย

   #su upcm
7. เข้ามายัง home directory ของผู้ใช้ upcm
8. ทำการแก้ไขไฟล์ ~/.profile เพื่อเพิ่ม PATH ของ user โดยเพิ่ม

   PATH = "$PATH : /usr/Softw/mpich/bin"
9. ทำการสร้างไฟล์ ~/.rhost โดยเพิ่ม

   master

   node0

node1

10. logon ออกจากผู้ใช้ upcm
11. ทำการลงโปรแกรม MPICH <ดูหัวข้อ "การลงโปรแกรม MPICH">
12. แก้ไขไฟล์ /etc/exports โดยเพิ่ม

    /home node0(rw) node1(rw)

    /usr/Softw node0(rw) node1(rw)

13. ทำการ setup deamo "nfs" เพื่อให้ start deamo เมื่อตอนบูตเครื่องโดย

    #setup

    #เลือก system service

    #check ที่ [*]nfs

    #exit

    #exit

14. ทำการ restart เครื่องโดย

    #shutdown –r 0

เครื่อง node

1. เปิดเครื่องและ login ในฐานะ root
2. แก้ไขไฟล์ /etc/hosts โดยดังนี้

    100.0.0.1 master.pcm master

    100.0.0.2 node0.pcm node0

    100.0.0.3 node1.pcm node1

3. สร้าง directory /usr/Softw เพื่อให้รับ mount กับ master:/usr/Softw

    #mkdir /usr/Softw

4. แก้ไขไฟล์ /etc/fstab โดยเพิ่ม

    Master:/home /home nfs rsize = 1024, wsize = 1024, time0 = 14 ,intr

    Master:/usr/Softw /usr/Softw nfs rsize=1024, wsize=1024, time0 = 14, intr

5. ทำการเพิ่ม user "upcm"

    #adduser upcm

6. ทำการให้ password ให้กับ user upcm

    #passwd upcm

    NEW password: <พิมพ์ password แล้ว enter>

    Retry password : <พิมพ์ password อีกครั้งแล้ว enter>

7. login เข้าเป็น user "upcm"

   #su upcm

   #cd ~

8. แก้ไขไฟล์ ~/.rhost โดยเพิ่ม

   master

   node0

   node1

9. แก้ไขไฟล์ ~/.Profile โดยเพิ่ม PATH

   PATH = "$PATH:/usr/Softw/mpich/bin"

10. logon จากผู้ใช้ upcm

    #exit

11. restart เครื่องโดย

    #shutdown –r 0

การติดตั้งโปรแกรม MPICH

1. ทำการ download ไฟล์ mpich.tar.gz แล้วเก็บไว้ที่ /home/src
2. เข้าไปที่ directory /home/src

   #cd /home/src
3. ทำการขยายไฟล์ mpich.tar.gz โดย

   #gunzip -cd mpich.tar.gz |tar xorf –
4. เข้าไปที่ directory mpich-1.2.4 โดย

   #cd mpich-1.2.4
5. ทำการ configuration โปรแกรม MPICH โดย

   #./configure --prefix = /usr/Softw/mpich

   เมื่อ /usr/Softw/mpich คือ ตำแหน่ง directory ที่จะทำการลงโปรแกรม MPICH
6. ทำการ compile โปรแกรม MPICH โดย

   #make
7. ทำการแก้ไขไฟล์ util/machines/machines.LINUX โดยเพิ่ม

   master

   node0

   node1

8. ทำการติดตั้งโปรแกรม MPICH ลงไปยังตำแหน่งที่กำหนด

   #make install

การใช้โปรแกรม MPICH

1. การ compile โปรแกรม MPI

   $mpicc –o prog prog.c

   เมื่อ prog.c คือ file source code ภาษา C

   และ prog คือ file ที่ได้จากการ compile file source code prog.c สามารถ run ได้

2. การ run โปรแกรม MPI

   $mpirun -np [n] prog

   เมื่อ n คือ จำนวนโพรเซสเซอร์ ที่ต้องการให้ทำการ run โปรแกรมและ prog คือ file ที่สามารถ run ได้

2.3 ทดสอบระบบ

โปรแกรมที่ใช้ในการทดสอบ คือ

1. ทำการ compile ไฟล์ systest.c

   #mpicc –o systest.c

2. ทำการ run โปรแกรม systest โดยใช้โพรเซสเซอร์1 ตัว

   #mpirun –np 1 systest

3. ทำการ run โปรแกรม systest โดยใช้โพรเซสเซอร์ 2 ตัว

   #mpirun –np 2 systest

4. ทำการ run โปรแกรม systest โดยใช้โพรเซสเซอร์ 3 ตัว

   #mpirun –np 3 systest

ผลการทดสอบระบบ

1. จากการทดสอบระบบโดยใช้โปรแกรม systest จะเห็นว่า เมื่อเราใช้โพรเซสเซอร์ 1 ตัว โปรแกรมจะเช็คว่าจะใช้โพรเซสเซอร์ตัวใด ซึ่งก็คือ ใช้ โพรเซสเซอร์ที่มีค่า rank = 0 มีชื่อว่า master.pcm
2. ถ้าต้องการทดสอบระบบโดยเพิ่มโพรเซสเซอร์เป็น 2 ตัว โปรแกรมจะทำการเช็คและแสดงชื่อของโพรเซสเซอร์ที่ใช้ออกมา ซึ่งก็คือ

   master.pcm

   node0.pcm

   ซึ่งมีค่า rank = 0 และ 1 ตามลำดับ

3. และถ้าทำการทดสอบระบบโดยเพิ่มโพรเซสเซอร์ เป็น 3 เครื่อง โปรแกรมจะทำการตรวจสอบ และ แสดงรายโพรเซสเซอร์ ชื่อโพรเซสเซอร์ออกมา คือ

   master.pcm

   node0.pcm

   node1.pcm

บทที่ 4

การสร้างและพัฒนาโปรแกรม

1. ขั้นตอนการทำงานของโปรแกรม

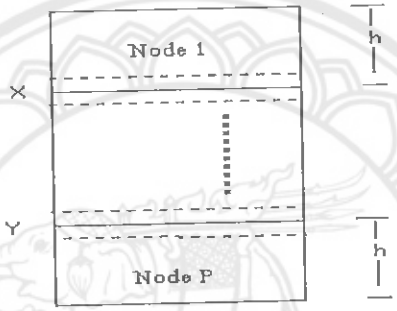
1.1การแบ่งภาพ จะทำที่เครื่อง master โดย master จะหาว่ามีโหนดที่รองรับการประมวลผลกี่เครื่อง เมื่อได้จำนวนโหนดแล้วนำมาแบ่งภาพ ตามสมการ

$$h = Row / p$$

เมื่อ h คือ ขนาดของภาพที่แต่ละโหนดได้รับ

Row คือ ขนาดตามแนวตั้งของภาพ

P คือ จำนวนเครื่อง



รูปที่ 4.1 การแบ่งภาพให้แต่ละเครื่อง

โหนดแรกจะได้รับภาพจากจุดเริ่มต้นถึง X+1

โหนดสุดท้ายจะได้รับภาพจาก Y-1 ถึงจุดสุดท้าย

ถ้าไม่ใช่โหนดแรกและโหนดสุดท้าย จะได้รับภาพจาก X-1 ถึง Y+1

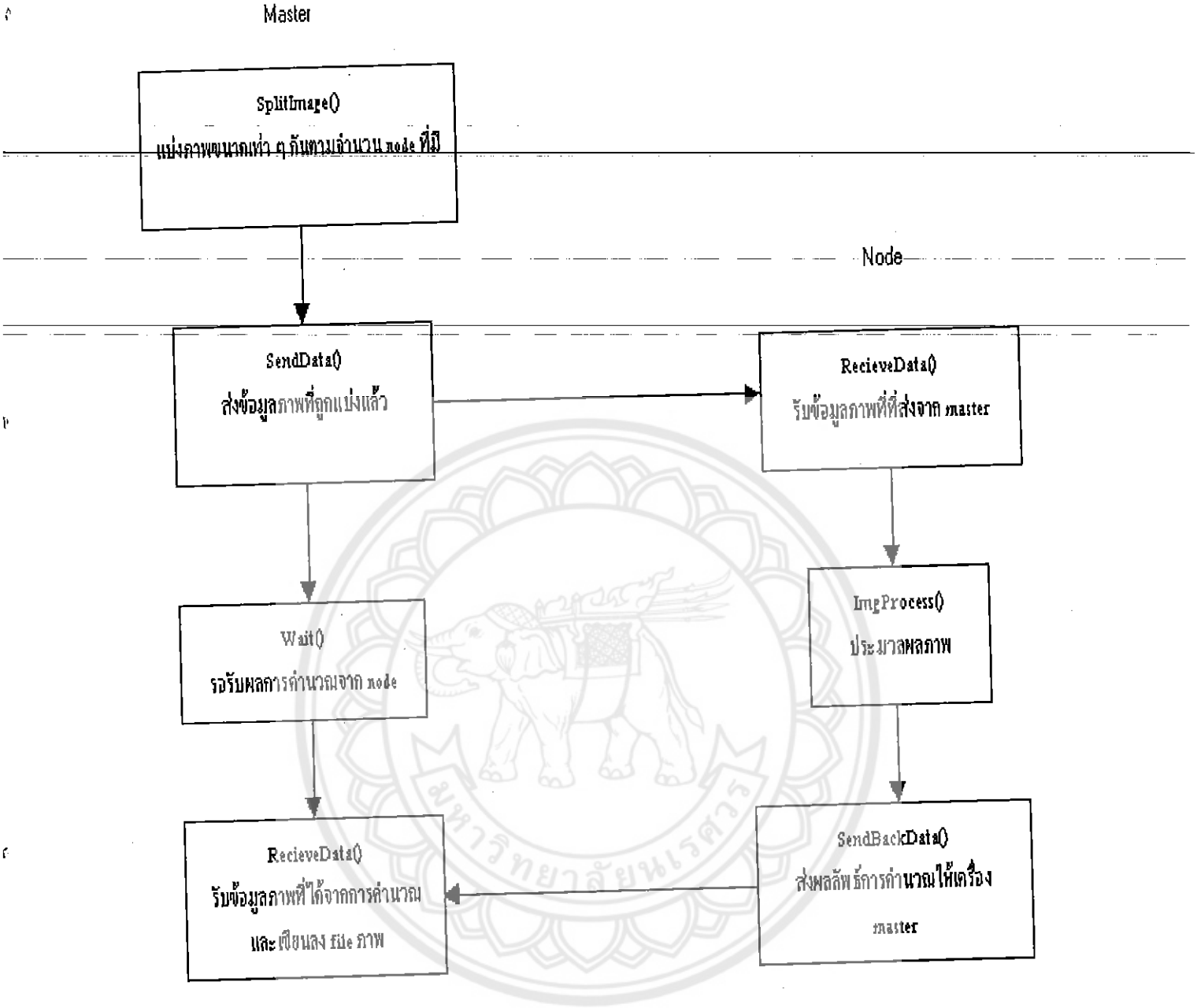
1.2 การทำงานของโปรแกรม

โดยโปรแกรมจะแบ่งการทำงานออกเป็น 2 ส่วน คือ

ส่วนของเครื่อง master และ ส่วนของเครื่องโหนด โดยที่เครื่อง master มีหน้าที่แบ่งภาพออกเป็นส่วน ๆ ตามจำนวนโหนดที่มีอยู่ แล้วก็ส่งภาพที่แบ่งแล้วนั้นไปให้โหนดทุกโหนดและก็รอรับภาพที่ได้จากการคำนวณของโหนด ทุกโหนดเขียนลงไฟล์

ส่วนโหนดทุกโหนดมีหน้าที่รับ ข้อมูลภาพจาก master แล้วจึงทำการประมวลผลภาพ แล้วก็ส่งผลคำนวณคืนให้กับ master

โดยจะมีไดอะแกรมแสดงการทำงานดังนี้



รูปที่ 4.2 การทำงานของโปรแกรม

บทที่ 5

ผลการทดลอง

ทำการทดลองโดยใช้โปรแกรมที่เขียนขึ้นเพื่อใช้ในระบบ PC Cluster โดยในการทดลองจะใช้ภาพที่มีขนาดคงที่ คือ 800 x 600 พิกเซล ซึ่งการทดลองประมวลผลภาพในโครงงานวิจัยทำ 3 อย่างด้วยกัน คือ การหาขอบภาพ(Edge Detection)โดยใช้ Homogeneity Operator และ Difference Operator, การหาค่าอนุพันธ์อันดับหนึ่ง (First Order Directive) โดยใช้ Robert Operator

1. ขั้นตอนการทดลอง

ขั้นตอนการทดลองทำได้ ดังนี้

1. เตรียมรูปภาพชนิด บิตแม็บ ขนาด 800 x 600 8 bit โดยให้ชื่อไฟล์รูปภาพเป็น in.bmp และไฟล์นี้ต้องอยู่ตำแหน่งเดียวกันกับโปรแกรมประมวลผลภาพ
2. ทำการ run โปรแกรม โดยใช้คำสั่ง

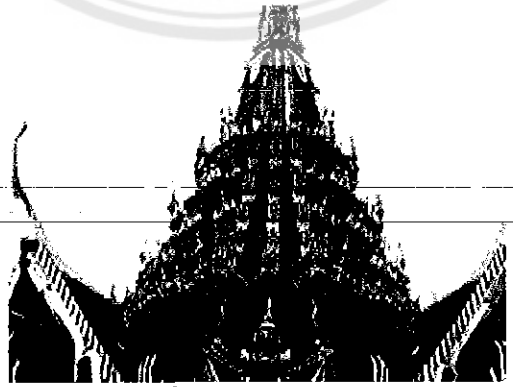
   $ mpirun -np N PROGRAMNAME

   โดยที่ N คือ จำนวน processor ที่ต้องการให้ทำงานมีค่าเริ่มจาก 1 เป็นต้นไป

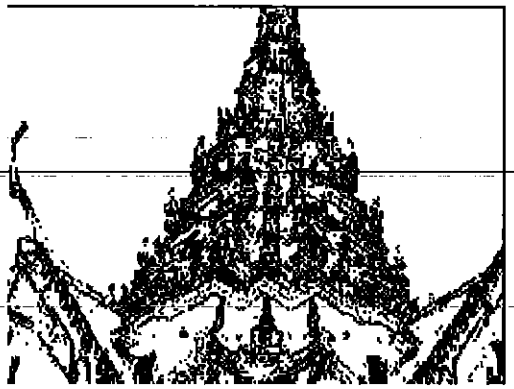
PROGRAMNAME	คือ โปรแกรมประมวลผลภาพ
Difference	คือ โปรแกรม Edge Detection โดยใช้ Difference Operator
Homogeneity	คือ โปรแกรม Edge Detection โดยใช้ Homogeneity Operator
Robert	คือ โปรแกรม First Order Directive โดยใช้ Robert Operator

3. บันทึกผลการทดลอง

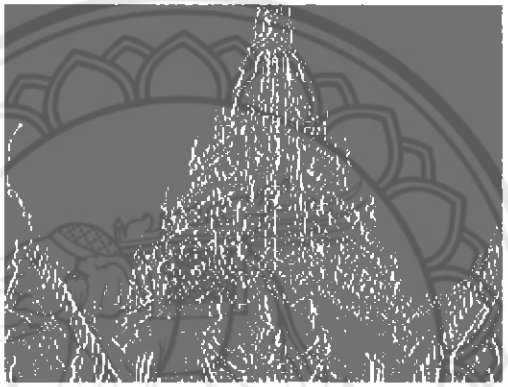
2. ผลการทดลอง



รูปที่ 5.1 ภาพต้นฉบับ



รูปที่ 5.2 ผลลัพธ์จากการหาขอบภาพโดยใช้ Homogeneity operator



รูปที่ 5.3 ผลลัพธ์จากการหาขอบภาพโดยใช้ Robert Operator

รูปที่ 5.4 ผลลัพธ์จากการหาขอบภาพโดยใช้ Difference Operator

ตารางที่ 5.1เวลาที่ใช้ในการประมวลผลการหาขอบภาพ โดยใช้ Homogeneity Operator หน่วย วินาที

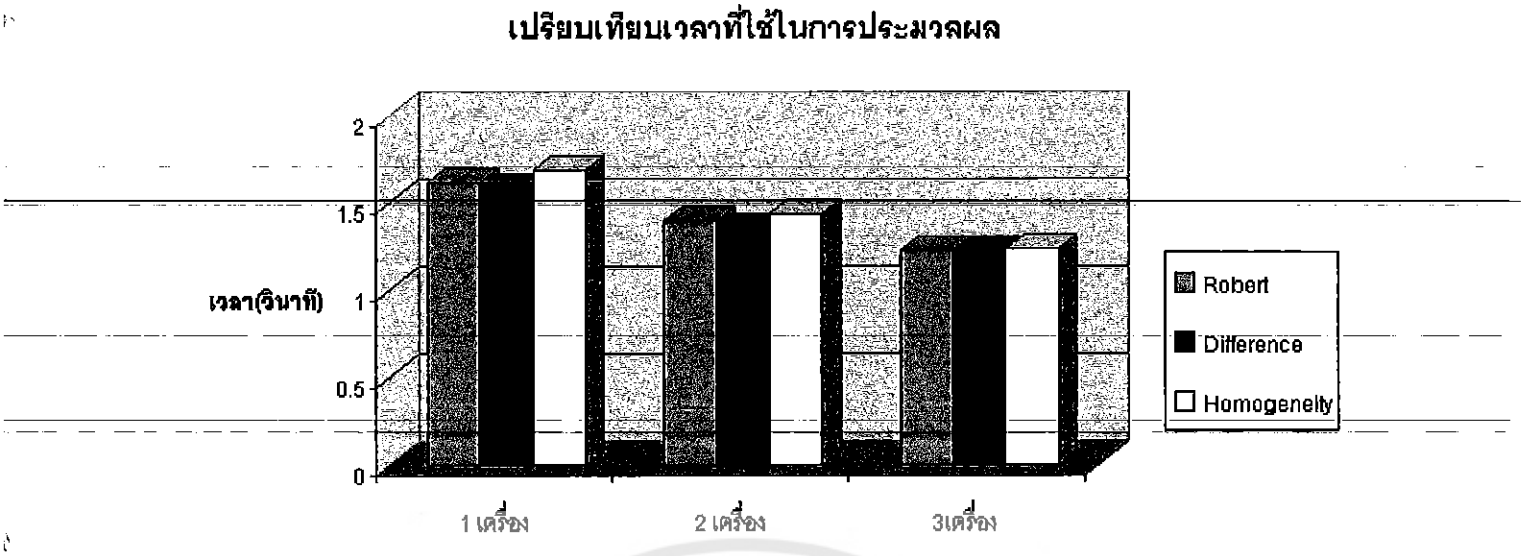
ครั้งที่ / จำนวนเครื่อง	1	2	3	4	5	เฉลี่ย
1	1.645213	1.652431	1.655245	1.650423	1.596423	1.639947
2	1.423046	1.441924	1.46947	1.460742	1.408613	1.440795
3	1.191845	1.280243	1.230213	1.253557	1.233827	1.237937

ตารางที่ 5.2 เวลาที่ใช้ในการประมวลผลการหาขอบภาพโดยใช้ Difference Operator หน่วย วินาที

ครั้งที่ / จำนวนเครื่อง	1	2	3	4	5	เฉลี่ย
1	1.621350	1.599872	1.600245	1.598745	1.568753	1.5166640
2	1.409982	1.426586	1.306596	1.311931	1.345849	1.3601888
3	1.234372	1.178319	1.261565	1.240753	1.221433	1.227289

ตารางที่ 5.3 เวลาที่ใช้ในการประมวลผลของ First Order Directive โดยใช้ Robert Operator หน่วย วินาที

ครั้งที่ / จำนวนเครื่อง	1	2	3	4	5	เฉลี่ย
1	1.487621	1.510259	1.566687	1.498732	1.458772	1.5044142
2	1.37078	1.384611	1.412278	1.348321	1.340152	1.371229
3	1.214136	1.205789	1.22345	1.161615	1.276423	1.276423



รูปที่ 5.5 ความสัมพันธ์ระหว่างจำนวนเครื่อง กับ เวลาที่ใช้ในการคำนวณแต่ละแบบ

2. สรุปผลการทดลอง

จากการทดลองสรุปได้ว่าเมื่อเราเพิ่มจำนวนโหนดที่ใช้ในการประมวลผลภาพ เราจะใช้เวลาในการประมวลผลน้อยลง แต่ถ้าในการประมวลผลนั้นใช้การคำนวณน้อยมาก เราควรจะใช้เพียโหนดเดียวในการประมวลผลเพราะเวลาที่ใช้ในการประมวลผลนั้นน้อยกว่าเวลาที่ใช้ในการรับส่งข้อมูลระหว่างโหนด ทั้งนี้เพราะเครือข่ายมีความเร็วต่ำ ทำให้เวลาส่วนใหญ่สูญเสียไปกับการรับส่งข้อมูลมากกว่าเวลาที่ใช้ในการประมวลผลมาก

และจากการทดลองจะเห็นว่าเวลาที่ใช้ในการประมวลผลของเครื่องคอมพิวเตอร์ 3 เครื่องจะไม่เป็นครึ่งหนึ่งของเวลาที่ในการประมวลผลด้วยเครื่องคอมพิวเตอร์ 2 เครื่อง ตามทฤษฎี แต่จะลดลงเพียงประมาณ 0.2 วินาทีเท่านั้น ทั้งนี้เพราะเวลาที่ใช้ในการประมวลผลจริงมีค่าน้อยมาก

ตามทฤษฎีเมื่อใช้เครื่องคอมพิวเตอร์ 1 เครื่อง ใช้เวลา s วินาที ถ้าจำนวนเพิ่มขึ้นเป็น n เครื่องจะใช้เวลาในการประมวลผลลดลงเป็น s/n วินาที แต่จากผลการทดลองไม่เป็นไปตามทฤษฎีเพราะเวลาส่วนใหญ่ (ประมาณ 1 วินาที) จะใช้ในการรับส่งข้อมูล จึงทำให้ผลการทดลองไม่เป็นไปตามทฤษฎี แต่เมื่อทำการวิเคราะห์เวลาที่ใช้ในการประมวลผลจริง(เวลาประมวลผลจริง = เวลาทั้งหมด-เวลาในการรับส่งข้อมูล) จะเห็นว่าเวลาที่ใช้มีค่าลดลง 0.5 เท่าซึ่งเป็นไปตามทฤษฎีที่ตั้งไว้

บทที่ 6
บทสรุป

1.สรุปผล

จากการทำการศึกษาเกี่ยวกับการทำงานของ PC Cluster ซึ่งเป็นทางเลือกหนึ่งของการประมวลผลที่มีประสิทธิภาพในการประมวลผลสูงแต่ราคาถูกเมื่อเทียบกับเครื่องที่ใช้การประมวลผลแบบขนาน ทำให้ PC Cluster มีการทดลองและวิจัยมากขึ้นเรื่อย ๆ ลักษณะการทำ PC Cluster สามารถทำได้หลายวิธี ไม่ว่าจะเป็นวิธี diskless system ซึ่งโหนด ภายใน cluster ไม่จำเป็นต้องมีฮาร์ดดิส และไม่ต้องมีระบบปฏิบัติการในทุกโหนด นอกจากนี้ยังมีอีกวิธีหนึ่งที่นิยมทำกันคือการติดตั้งระบบปฏิบัติการในทุกโหนดซึ่งในโครงงานวิจัยได้เลือกวิธีการที่สองนี้ ในการทำ PC Cluster จะต้องเข้าใจในสองส่วน คือ ฮาร์ดแวร์ และ ซอร์ฟแวร์ ในส่วนของฮาร์ดแวร์ต้องทำความเข้าใจเกี่ยวกับ ฮับ , การ์ดแลน การเซตระบบเน็ตเวิร์ค ให้สามารถติดต่อกันได้ ส่วนด้านซอร์ฟแวร์ต้องทำความเข้าใจเกี่ยวกับระบบปฏิบัติการ Linux (การเซต configuration ต่างๆ) รวมไปถึงการใช้งาน MPI ซึ่งเป็นมาตรฐานในการติดต่อกันภายใน PC Cluster และต้องศึกษากระบวนการการประมวลผลภาพ

เมื่อทำการเขียนโปรแกรมเสร็จเรียบร้อยแล้วก็นำโปรแกรมมารันบน PC Cluster ที่สร้างขึ้น เพื่อทำการทดลองตามจุดประสงค์ที่ได้ตั้งเอาไว้ โดยใช้การจับเวลาในการประมวลผลภาพ ในแต่ละชนิดของ operator ทำการทดลองโดยใช้เครื่องคอมพิวเตอร์ 2 เครื่อง และ 3 เครื่อง ผลที่ได้จากการทดลอง เมื่อนำมาเปรียบเทียบจะเห็นได้ว่าตัวดำเนินการที่มีการคำนวณมากจะใช้เวลาในการคำนวณมากกว่าดำเนินการที่มีการคำนวณน้อยกว่าและจากผลการทดลองสรุปได้ว่าเมื่อเพิ่มจำนวนโหนดที่ใช้ในการประมวลผลภาพก็จะใช้เวลาในการประมวลภาพน้อยลง

จากการศึกษาทำให้เข้าใจหลักการเขียนโปรแกรมแบบขนานและสามารถพัฒนาโปรแกรมประยุกต์ได้ เข้าใจโครงสร้างของระบบ PC Cluster และเข้าใจระบบ เน็ตเวิร์คได้มากยิ่งขึ้น

2.ปัญหาและวิธีการแก้ปัญหา

2.1 เครื่องคอมพิวเตอร์ที่ใช้ในการทดลองมีจำนวนน้อย จึงทำให้ไม่สามารถแสดงความแตกต่างของเวลาที่ใช้ในการประมวลผลได้อย่างชัดเจนนัก
แก้ไขได้โดย เพิ่มจำนวนเครื่องคอมพิวเตอร์

2.2 อุปกรณ์เชื่อมโยงเครือข่าย (Hub10/100 Mbit/sec,สาย UTP) มีความเร็วต่ำ ทำให้เวลาที่ใช้ในการ ประมวลผลไม่แตกต่างกันมากนัก เนื่องจากเวลาที่ใช้ส่งผ่านข้อมูลมีค่ามาก แต่เวลาที่ใช้คำนวณมีค่าน้อยมากเมื่อเทียบกับเวลาที่ใช้ส่งผ่านข้อมูล
แก้ไขโดยใช้เครือข่ายที่มีความเร็วมากกว่านี้ เช่น Fast Ethernet เป็นต้น

2.3 ตำรา และ เอกสารอ้างอิงมีน้อย ทำให้ความรู้ที่ได้ไม่กว้างขวางมากนัก

แก้ไขหาตำรา หรือ เอกสารอ้างอิงเพิ่มขึ้น

3.ข้อดี และ ข้อเสีย

ข้อดี

1.สามารถนำไปประยุกต์ใช้อย่างกว้างขวาง

2.ระบบมีต้นทุนต่ำ แต่ได้ประสิทธิภาพสูง

3.การเขียนโปรแกรมง่ายลง เนื่องจากปัจจุบันมีเครื่องมือช่วยมากขึ้น

ข้อเสีย

1.ใช้เนื้อที่ปฏิบัติการมาก

2.การเชื่อมโยงเครือข่ายยุ่งยาก

3.ระบบการเขียนโปรแกรมแบบขนาน มีความซับซ้อนกว่าการเขียนโปรแกรมแบบอนุกรม อย่างที่คุ้นเคย

4.แนวทางในการพัฒนา

สามารถนำไปประยุกต์ในงานที่ต้องความรวดเร็ว มีปริมาณข้อมูลมาก มีความซับซ้อนมาก และมีการประมวลผลมหาสาร หรือ สามารถประยุกต์ไปเป็นเซอร์เวอร์ต่าง ๆ ในองค์กร

เอกสารอ้างอิง

[1] Peter S.Pacheco. **Parallel Programming with MPI**. Morgan Kaufmann Publishers, Ins USA , 1997

[2] Crame Randy. **A Simplified Approach to Image Processing: Classical and Modern Techniques in C** .Upper Saddle River, N.J., Prentic Hall PTR, c1997

ประวัติผู้เขียนโครงงานวิจัย

1. นายจะเร เตือนสติ รหัส 40360224

เกิดวันที่ 1 ธันวาคม 2521

คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมคอมพิวเตอร์

มหาวิทยาลัยนเรศวร

จบการศึกษาระดับมัธยมศึกษาจาก ร.ร สวรรค์อนันต์วิทยา

บ้านเลขที่ 151/4 หมู่ 4 ต.หนองกลับ อ.สวรรคโลก จ.สุโขทัย 64110

email: krazip@hotmail.com

2. นายวิษณุกรณ์ ครีบเขียว รหัส 40360497

เกิดวันที่ 10 สิงหาคม 2521

คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมคอมพิวเตอร์

มหาวิทยาลัยนเรศวร

จบการศึกษาระดับมัธยมศึกษาจาก ร.ร ศรีสำโรงชนูปถัมภ์

บ้านเลขที่ 40/1 หมู่ 10 ต.สามเรือน อ.ศรีสำโรง จ.สุโขทัย 64120

email: willyken@thaimail.com

3. นายเดชา กันลา รหัส 40360612

เกิดวันที่ 9 ตุลาคม 2521

คณะวิศวกรรมศาสตร์ สาขาวิศวกรรมคอมพิวเตอร์

มหาวิทยาลัยนเรศวร

จบการศึกษาระดับมัธยมศึกษาจาก ร.ร ร้อยเอ็ดวิทยาลัย

บ้านเลขที่ 54 หมู่ 13 ต.สะอาดสมบูรณ์ อ.เมือง จ.ร้อยเอ็ด 45000

email: jennyk@thaimail.com