

การใช้งานระบบคอมพิวเตอร์สมรรถนะสูง เบื้องต้น

โภมัสร์ สเตอร์วิ่ง[†]
กษิติศ ชาญเชี่ยว

สารบัญ

บทที่ 1 แนะนำ High Performance Computing

- 1.1 ปัจจัยที่มีความต้องการในการคำนวณสูง
- 1.2 ที่มาของความสามารถในการประมวลผลสมรรถนะสูง
 - 1.2.1 Supercomputer
 - 1.2.2 Commodity Cluster
 - 1.2.3 Volunteer Computing
 - 1.2.4 Grid Computing

บทที่ 2 สถาปัตยกรรมของระบบคอมพิวเตอร์สมรรถนะสูง

- 2.1 HPC system Stack
- 2.2 ตัวแปรที่เกี่ยวข้องกับประสิทธิภาพ
- 2.3 สถาปัตยกรรมคอมพิวเตอร์
- 2.4 ประเด็นเกี่ยวกับประสิทธิภาพของ Parallel Structures
 - 2.4.1 Scalability
 - 2.4.2 Performance Matrices
- 2.5 Basic Uni-processor Architecture Element
- 2.6 Multiprocessor
- 2.7 Massively Parallel Processor
- 2.8 Pipeline Structures
- 2.9 Vector Processor
- 2.10 SIMD
- 2.11 Special Purpose Device
- 2.12 Systolic Array
- 2.14 Introduction to SMP
- 2.15 Challenge to Computer Architecture
- 2.16 Commodity Cluster

บทที่ 3 ระบบ Commodity Cluster

- 3.1 แนะนำระบบ Commodity Cluster
- 3.2 ประวัติของระบบคลัสเตอร์
- 3.3 คุณลักษณะของระบบคลัสเตอร์

3.4 Cluster System

3.4.1 ระบบ Hardware และ Software ของคลัสเตอร์หนึ่ง

3.4.2 การ Programming ระบบ Cluster

3.4.3 Interconnection Network

3.5 Software สำหรับคลัสเตอร์

3.6 การวัดประสิทธิภาพของระบบคลัสเตอร์

บทที่ 4 การใช้งานระบบ Cluster Computer เป็นต้นและ ssh โพรโตคอล

4.1 แนะนำระบบ hpccluster

4.2 การเข้าใช้ user account

4.3 การพัฒนา MPI โปรแกรมบนระบบ hpccluster

4.3.1 การ compile โปรแกรมด้วย mpicc

4.3.2 การ run โปรแกรมด้วย mpirun

4.3.3 การ transfer files ไปยัง hpccluster

4.4 การสื่งงานเครื่องคอมพิวเตอร์หลายเครื่องผ่าน ssh โดยไม่ใช้รหัสผ่าน

4.4.1 telnet และ rsh

4.4.2 การใช้ ssh

4.4.3 การติดตั้งและใช้งาน public และ private keys เพื่อการ remote login

4.4.4 การใช้งาน ssh-agent เพื่อการ remote login โดยไม่ใช้ passphrase

4.4.5 การใช้คำสั่ง ssh และ scp

4.5 ทำไม่ใช้ agent และ public และ private key ถึงปลอดภัยกว่า password

บทที่ 5 การใช้งาน Message Passing Interfaces

5.1 MPI Standard และ ประวัติของ MPI

5.2 MPI Runtime Environment

5.3 หลักการพื้นฐานของ MPI

5.4 MPI Point-to-Point Communication

5.5 Deadlock

5.6 ตัวอย่าง MPI program ที่ใช้ Point-to-point Communication

บทนำ

ตำราเล่มนี้มีจุดประสงค์เพื่อแนะนำให้ผู้อ่านรู้จักกับระบบคอมพิวเตอร์สมรรถนะสูง ระบบ Commodity Cluster และการใช้งานระบบ Message Passing Interfaces เพื่อเขียน Parallel Program บนระบบ Cluster

เนื้อหาของตำราเล่มนี้ได้แก่บทที่ 2 และบทที่ 3 เป็นการแปลจากคลิป lecture วิชา High Performance Computing ที่ Louisiana State University ของ Professor Thomas Sterling ผู้ที่ได้รับการยกย่องว่าเป็นผู้ริเริ่มสร้างระบบ Cluster Computer ขึ้น โดยผู้แปลและเรียบเรียง คือ กษิติ ชาญเชี่ยว สำหรับ บทที่ 1 บทที่ 4 และ บทที่ 5 นั้นแต่งโดย กษิติ ชาญเชี่ยว จาก เนื้อหาวิชา Introduction to Cluster Computing ที่ ภาควิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยธรรมศาสตร์

ตำราเล่มนี้กำลังอยู่ในระหว่างปรับปรุงแก้ไข ผู้แต่งและเรียบเรียงหวังว่าจะปรับปรุงให้ดีขึ้นเพื่อให้เป็นตำราที่ได้มาตรฐานในอนาคต



บทที่ 1 แนะนำ High Performance Computing

แต่งและเรียบเรียง กษิจิต ชาญเชี่ยว

ระบบคอมพิวเตอร์ได้รับการพัฒนามาเพื่อแก้ปัญหาที่มีความต้องการในการคำนวณหรือประมวลผลเป็นปริมาณมากได้อย่างรวดเร็วให้ทันต่อความต้องการของมนุษย์ เมื่อความสามารถของคอมพิวเตอร์มีมากขึ้น เราก็นำมันไปใช้แก้ปัญหาต่างๆได้มากขึ้น แต่อย่างไรก็ตามเนื่องจากความต้องการในการตรวจสอบความรู้ของมนุษย์นั้น ไร้ขีดจำกัด คอมพิวเตอร์ที่ได้รับการพัฒนาขึ้นมาก็มักจะพบกับขีดจำกัดในการใช้งานเสมอ

ในปัจจุบัน ถึงแม้ว่าความสามารถในการประมวลผลของระบบคอมพิวเตอร์แต่ละเครื่องจะมีมาก แต่ก็มีปัญหาหลายอย่างที่คอมพิวเตอร์เหล่านั้นไม่สามารถแก้ปัญหาการประมวลผลให้เราได้ตามที่ต้องการ ทำให้มีการพัฒนาระบบคอมพิวเตอร์สมรรถนะสูงขึ้นมาเพื่อแก้ปัญหาเหล่านั้น และระบบคอมพิวเตอร์ดังกล่าวก็มีความสามารถในการประมวลผลในลักษณะที่เราเรียกว่า Parallel Processing หรือการประมวลผลแบบขนานที่ทำให้มันแก้ปัญหาขนาดใหญ่เหล่านี้ได้

เราจะยกตัวอย่างเปรียบเทียบสำหรับ Parallel Processing ดังนี้สมมติว่าคุณเป็นเจ้าของบริษัทรับเหมาสร้างบ้านบริษัทหนึ่งซึ่งมีพนักงานอยู่ 10 คนและสามารถสร้างบ้านได้ 1 หลังใช้เวลาหนึ่งเดือน ถ้ามีบริษัท Real Estate บริษัทหนึ่งสนใจจะจ้างบริษัทของคุณให้สร้างบ้านให้เขาหนึ่งหลังบ้านมี 100 หลังคาดเดือน ภายในได้เงินไว้ว่าบริษัทก่อสร้างของคุณต้องสร้างบ้านทั้ง 100 หลังให้เสร็จภายใน 1 เดือน คุณจะแก้ปัญหานี้อย่างไร แน่นอนคุณต้องจ้างคนงานเพิ่ม ซึ่งในการนี้คุณต้องจ้างคนเป็น 100 เท่าคือ 1000 คนนั่นเอง และให้คนงานกลุ่มละ 10 คนสร้างบ้านแต่ละหลังไป นี่เป็นหลักการเดียวกันกับที่เราใช้ในการแก้ปัญหาแบบ Parallel Processing

1.1 ปัญหาที่มีความต้องการในการคำนวณสูง

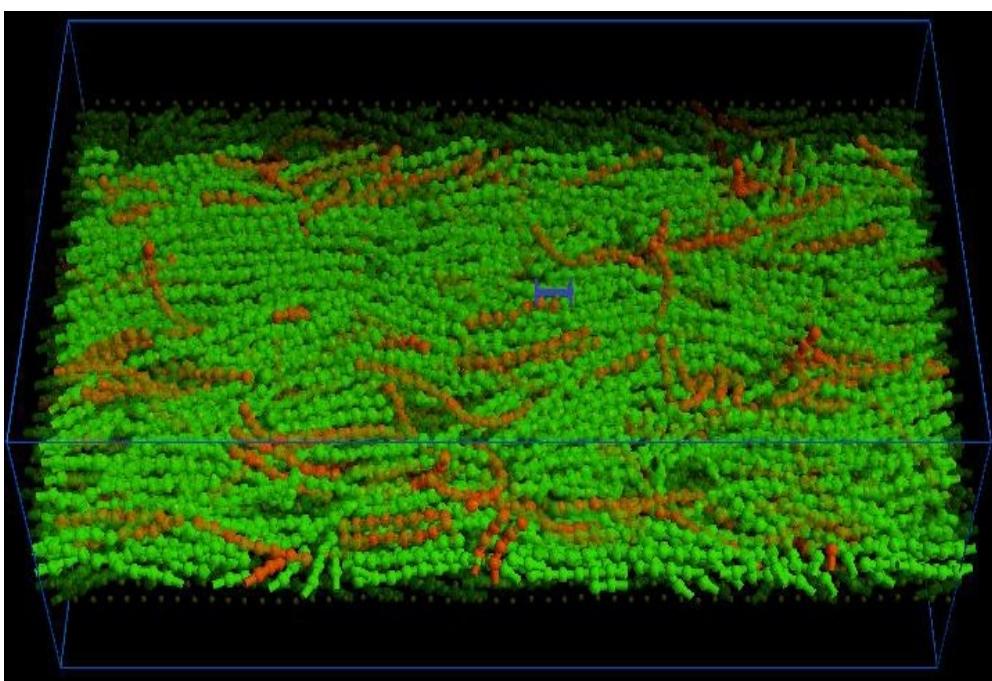
ตัวอย่างของปัญหาที่มีความจำเป็นต้องใช้คอมพิวเตอร์ที่มีความสามารถสูงอีกอย่างหนึ่ง ได้แก่ ในการที่เราจะพยากรณ์อากาศในอีก 2 วันถัดไปในเมริกาได้อย่างแม่นยำเราจำเป็นต้องใช้ข้อมูลสภาพอากาศในแต่ละตำแหน่งในชั้นบรรยากาศเป็นจำนวนมากๆมาใช้ในการประมวลผล เนื่องจากพื้นที่ของทวีปอเมริกาเนื้อที่มีประมาณ 20 ล้านตารางกิโลเมตรและต้องใช้ข้อมูลที่วัดสูงขึ้นไปในชั้นบรรยากาศเป็นระยะทาง 20 กิโลเมตร และในแต่ละพื้นที่สามารถมีความกว้าง 1 km ยกกำลัง 3 (หรือ cube) จะถูกแบ่งเป็น cube อยู่ๆอีก 1000 cube นั่นหมายความว่ามีข้อมูลอยู่ 1000 จุดใน 1 km cube ซึ่งจะทำให้มีข้อมูลที่จะต้องนำมาคำนวณทั้งหมดเป็น 4×10^{11} grid point

สมมุติว่าต้องใช้การคำนวณ 100 คำสั่งเพื่อประมวลผล 1 grid point ก็จะทำให้เราต้องใช้การคำนวณทั้งสิ้น 2×10^{15} คำสั่งเพื่อพยากรณ์สภาพอากาศของทวีปอเมริกาเนื้อใน 2 วันข้าง

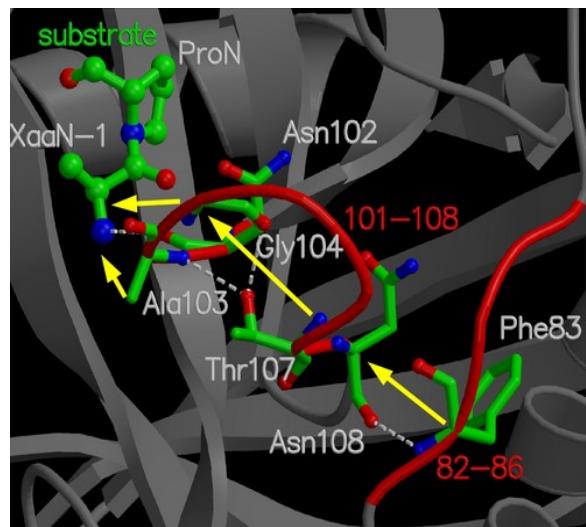
หน้า ในปัจจุบันคอมพิวเตอร์เครื่องหนึ่งมีความสามารถในการประมวลผลประมาณ 10^9 คำสั่งใน 1 วินาที ดังนั้นในการพยากรณ์อากาศจะต้องใช้ 2×10^{15} หาร 10^9 วินาทีหรือประมาณ 23 วัน ซึ่งจะเห็นได้ว่าไม่ทันต่อการใช้งาน ดังนั้นเราจึงจำเป็นต้องมีระบบคอมพิวเตอร์ที่สามารถประมวลผลได้เร็วกว่าที่เช่นมีความสามารถ 10^{12} คำสั่งต่อ 1 วินาที ก็จะทำให้สามารถพยากรณ์อากาศตามที่ต้องการได้ในครึ่งชั่วโมง

อีกตัวอย่างหนึ่งที่เกิดขึ้นในอุตสาหกรรมภาพยนต์ก็คือการใช้คอมพิวเตอร์ที่มีความสามารถสูงเพื่อสร้างภาพ Graphic ที่มีความละเอียดสูงและเหมือนจริงในการทำ Special Effect ในภาพยนตร์ต่างๆ ในสตูดิโอภาพยนตร์ใหญ่หลายแห่งนั้นมักจะมีการใช้เครื่องคอมพิวเตอร์สมรรถนะสูงเพื่อการสร้างภาพยนตร์ของตน

ในปัจจุบันมีปัญหามากมายที่ใช้ Parallel Processing เข้าไปช่วยในการคำนวณโดยเฉพาะอย่างยิ่งปัญหาทางวิทยาศาสตร์ ภาพ 1-1 แสดงภาพที่ได้จากการจำลองสภาวะการเปลี่ยนแปลงของโมเลกุลของสารเมื่อสภาวะแวดล้อมเปลี่ยนไป เนื่องจากการจำลองนั้นมีข้อมูลหรือ data point เป็นจำนวนมากจึงต้องใช้ระบบคอมพิวเตอร์สมรรถนะสูง ผลที่แสดงในภาพได้มาจากการคำนวณโดยใช้เครื่อง Cray X1 ที่ Oak Ridge National Laboratory (ORNL) การจำลองทางวิทยาศาสตร์โดยใช้คอมพิวเตอร์นั้นลดค่าใช้จ่ายจากการทดลองในห้อง lab จริงเป็นอย่างมาก ในตัวอย่างนี้แทนที่นักวิทยาศาสตร์จะต้องสังเคราะห์สตูดิวิชันมาแล้วทำการทดสอบมันซึ่งต้องใช้เงินเยอะและใช้เวลานาน เขาก็สามารถจำลองการทดลองขึ้นในคอมพิวเตอร์ก่อนแล้วค่อยทำการทดลองจริงเพื่อ confirm ผลของจากคอมพิวเตอร์

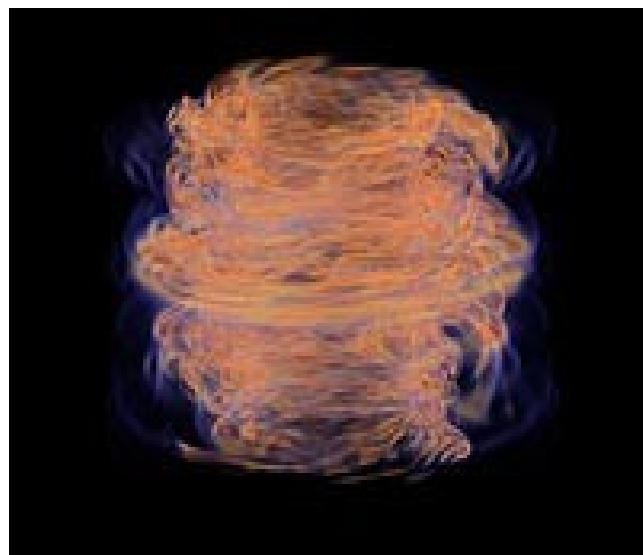


ภาพ 1-1 Large-scale molecular dynamics simulations (ORNL Cray X1) –source ORNL



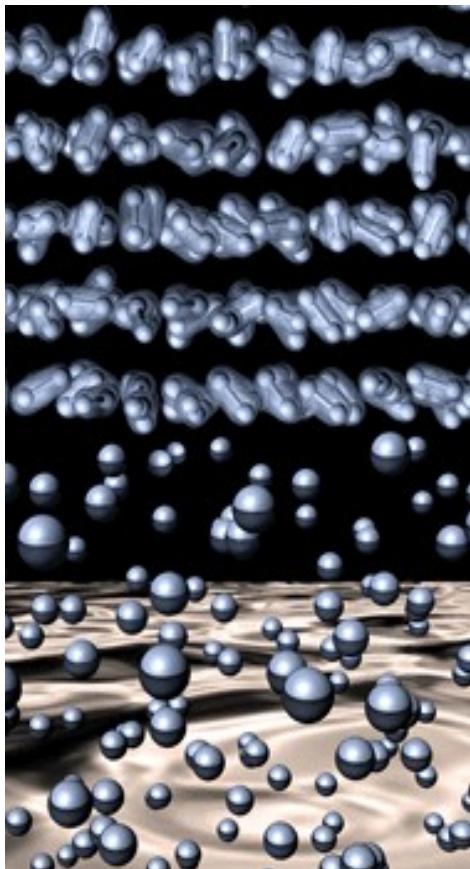
ภาพ 1-2 Multiscale modelling of protein vibrations in HIV infection related protein cyclophilin A. IBM power4 machine—source: ORNL

ภาพ 1-2 แสดงผลจากการจำลองการสั่นของสายโมเลกุลของโปรตีนที่อาจมีความเกี่ยวข้องกับการติดเชื้อ HIV การจำลองในภาพเป็นผลจากการประมวลผลของเครื่อง IBM power 4 ที่ ORNL การจำลองนี้ลดค่าใช้จ่ายในการทดลองจริงใน Wet lab เป็นอย่างมาก



ภาพ 1-3 Volume rendering of the energy dissipation through the current in the development of magneto-rotational instability, believed to be important in accretion of matter onto a central compact object such as a black hole. (Source: Argonne National Lab)

ภาพ 1-3 แสดงการจำลองการเกิดของหลุมดำโดยที่นักวิทยาศาสตร์เชื่อวามันเกิดจากการหมุนอย่างรวดเร็วของพลังงานแม่เหล็กและได้จำลองสถานะการณ์นี้ขึ้นโดยใช้ Supercomputers ที่ Argonne National Laboratory (ANL) การจำลองนี้ช่วยนักวิทยาศาสตร์ศึกษาทฤษฎีทางวิทยาศาสตร์ใหม่ๆได้



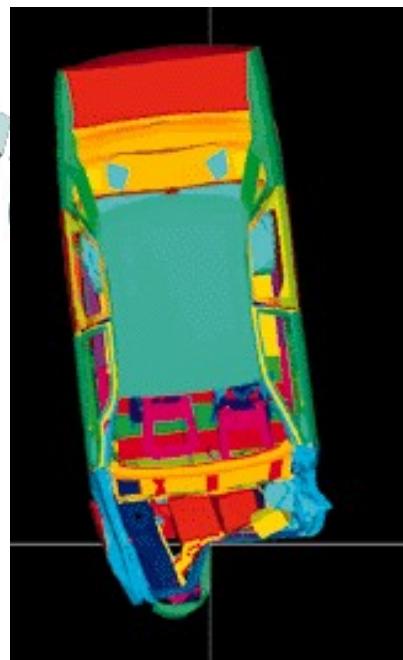
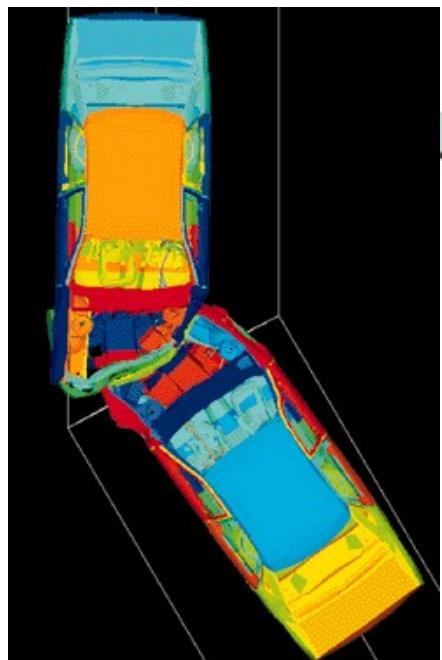
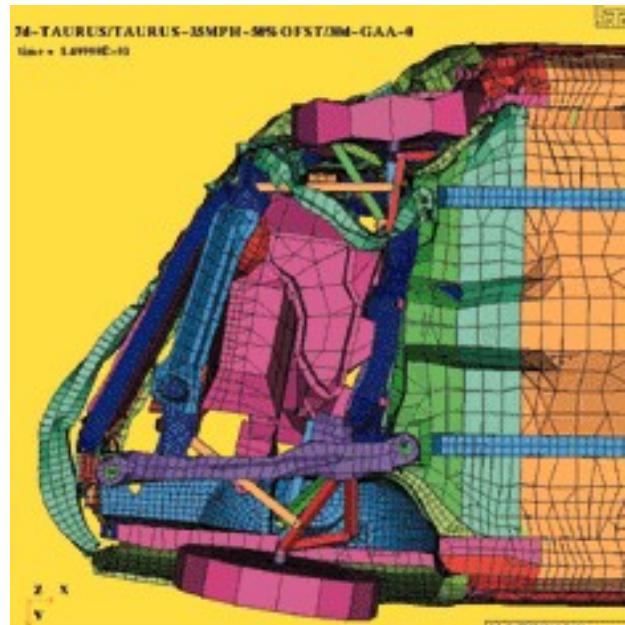
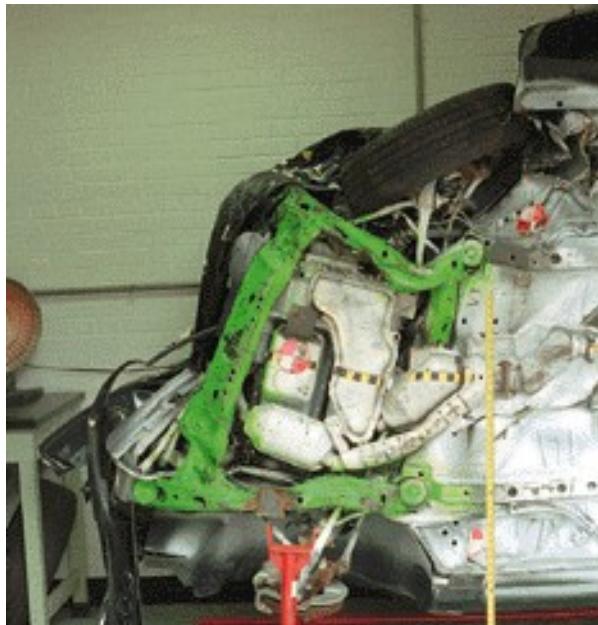
Molecular simulation (2005). Produced by the IBM BlueGene/L (BG/L) supercomputer at the Lawrence Livermore National Laboratory. Simulation of the transition from a molecular solid (top) to a quantum liquid (bottom) that is expected to occur in hydrogen under high pressure. BG/L set a world record for computing with a sustained performance of 207.3 trillion floating-point operations per second (teraFLOPS) using 131,072 processors. (Source: Lawrence Livermore National Laboratory).

ภาพ 1-4

ภาพ 1-4 แสดงการจำลองการเปลี่ยนสถานะของ Hydrogen จากของแข็งเป็นของเหลวภายในได้ภาวะความดันสูง ผลของการจำลองนี้ได้มาจากการประมวลผลโดยใช้เครื่อง IBM Bluegene Supercomputer ที่ Lawrence Livermore National Laboratory (LLNL) ซึ่งสามารถประมวลผลด้วยความเร็ว 207.3 Trillion คำสั่งใน 1 วินาที

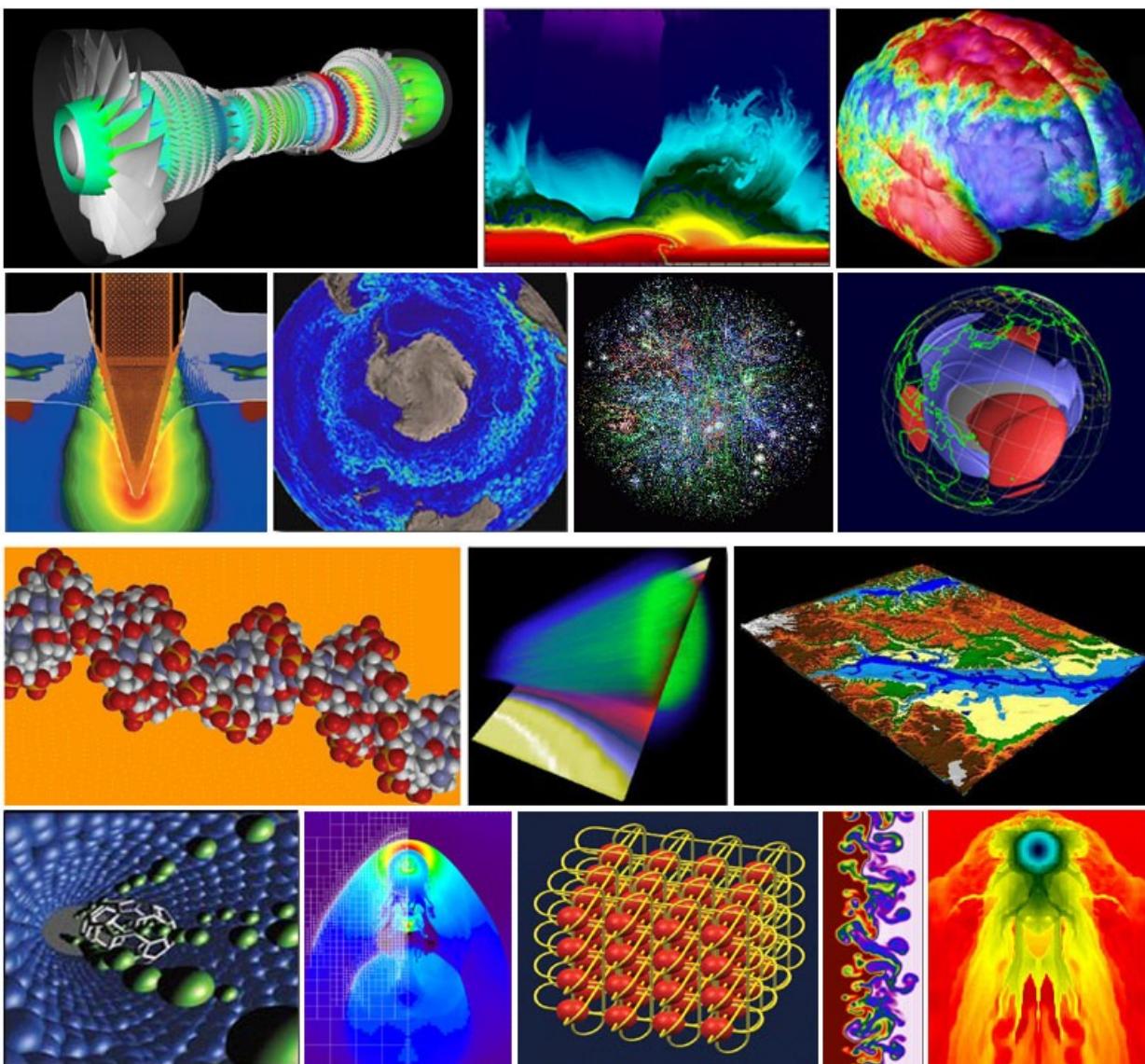
นอกจากการจำลองทางวิทยาศาสตร์แล้วการจำลองโครงสร้างต่างๆทางวิศวกรรมศาสตร์ ก็ต้องใช้การประมวลผลเป็นจำนวนมากเช่นกัน ภาพ 4-5 แสดงการจำลองการชนกันของรถยนต์ โดยใช้ Supercomputer ที่ ORNL และนำผลนั้นไปเทียบกับการชนจริง จากภาพจะเห็นว่าผลทั้งสองนั้นใกล้เคียงกันมาก การทดลองโดยใช้คอมพิวเตอร์จำลองแบบนี้จะช่วยลดค่าใช้จ่ายให้กับบริษัทผลิตรถยนต์อย่างมากในการออกแบบโครงสร้างของรถยนต์ให้ปลอดภัยก่อนที่จะสร้างต้นแบบรถยนต์จริง เพราะคอมพิวเตอร์จะช่วยให้บริษัทสามารถเลือกการออกแบบที่ทดสอบบน

คอมพิวเตอร์แล้วไม่กี่แบบมาทำการสร้างรถต้นแบบไม่กี่คันเพื่อนำไปทดสอบจริง แทนที่จะต้องสร้างรถต้นแบบมาใช้สำหรับการทดลองซึ่งຈึงหายๆคัน



ภาพ 1-5 Vehicle impact simulation based on Finite Element Modeling using supercomputers at ORNL

นอกจากตัวอย่างที่กล่าวมาข้างต้นยังมีปัญหาอีกมาก many ที่มีปริมาณการประมวลผลมาก และมีความจำเป็นที่ต้องใช้ระบบคอมพิวเตอร์สมรรถนะสูงเข้ามาแก้ปัญหา



ภาพ 1-6 ภาพที่ได้จากการ visualization ผลของการประมวลผลโดยใช้เครื่องคอมพิวเตอร์สมรรถนะสูง ภาพนี้มาจากการ https://computing.llnl.gov/tutorials/parallel_comp/

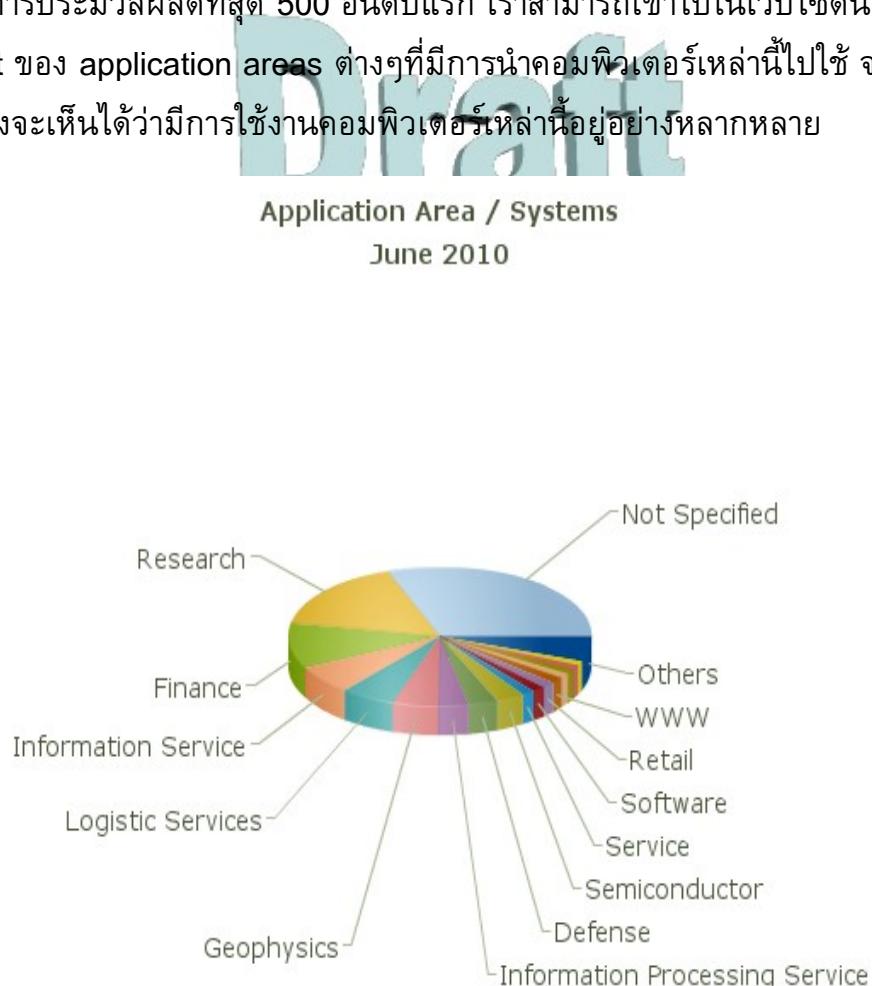
Blaise Barney นักวิจัยจาก LLNL ได้ลิสต์ปัญหาด้านต่างๆที่มีการใช้ระบบคอมพิวเตอร์สมรรถนะสูง เว็บไซต์ของเข้า (https://computing.llnl.gov/tutorials/parallel_comp/) ในภาพ 1-7 นี้ (ตัวอย่างในภาพ 1-6 แสดงผลจากการประมวลผลในด้านต่างๆเหล่านั้น)

<ul style="list-style-type: none"> • Physics - applied, nuclear, particle, condensed matter, high pressure, fusion, photonics • Bioscience, Biotechnology, Genetics • Chemistry, Molecular Sciences 	<ul style="list-style-type: none"> • Web search engines, web based business services • Medical imaging and diagnosis • Pharmaceutical design • Management of national and multi-
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<ul style="list-style-type: none"> • Geology, Seismology • Mechanical Engineering - from prosthetics to spacecraft • Electrical Engineering, Circuit Design, Microelectronics • Computer Science, Mathematics • Databases, data mining • Oil exploration 	<ul style="list-style-type: none"> national corporations • Financial and economic modelling • Advanced graphics and virtual reality, particularly in the entertainment industry • Networked video and multi-media technologies • Collaborative work environments
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

ภาพ 1-7 ตารางแสดงหัวข้อสาขาวิชาการใช้งานต่างๆของเครื่องคอมพิวเตอร์สมรรถนะสูง

จากเว็บไซต์ www.top500.org ซึ่งเป็นเว็บไซต์ที่ list ข้อมูลของคอมพิวเตอร์ที่มีความสามารถในการประมวลผลดีที่สุด 500 อันดับแรก เราสามารถเข้าไปในเว็บไซต์นี้แล้วสั่งให้มันแสดง chart ของ application areas ต่างๆที่มีการนำคอมพิวเตอร์เหล่านี้ไปใช้ จะได้ผลมาดังในภาพ 1-8 ซึ่งจะเห็นได้ว่ามีการใช้งานคอมพิวเตอร์เหล่านี้อยู่อย่างหลากหลาย



ภาพ 1-8

1.2 ที่มาของความสามารถในการประมวลผลสมรรถนะสูง

จากความต้องการในการประมวลผลปริมาณมากดังที่ได้กล่าวมาข้างต้น เราสามารถจัดหมวดหมู่ของระบบคอมพิวเตอร์ที่นำมาใช้ตอบสนองความต้องการดังกล่าวได้ดังนี้

1. Supercomputer
2. Commodity Cluster
3. Volunteer Computing
4. Grid Computing

1.2.1 Supercomputer

คอมพิวเตอร์ชนิดนี้เป็นคอมพิวเตอร์ที่มีประสิทธิภาพสูงที่สุดและมีราคาแพงที่สุดด้วยเช่นกัน และใช้งบประมาณมหาศาลในการพัฒนาและสร้างขึ้นมา คอมพิวเตอร์ชนิดนี้เป็นตัวบ่งชี้ว่า ความสามารถสูงในการประมวลผลของมนุษยชาติ ในปัจจุบันเครื่องคอมพิวเตอร์ชนิดนี้ที่มีความสามารถสูงสุดในโลกคือเครื่อง Cray XT5 ชื่อ Jaguar ที่ The National Center for Computational Science ที่ ORNL ดังแสดงในภาพ 1-9



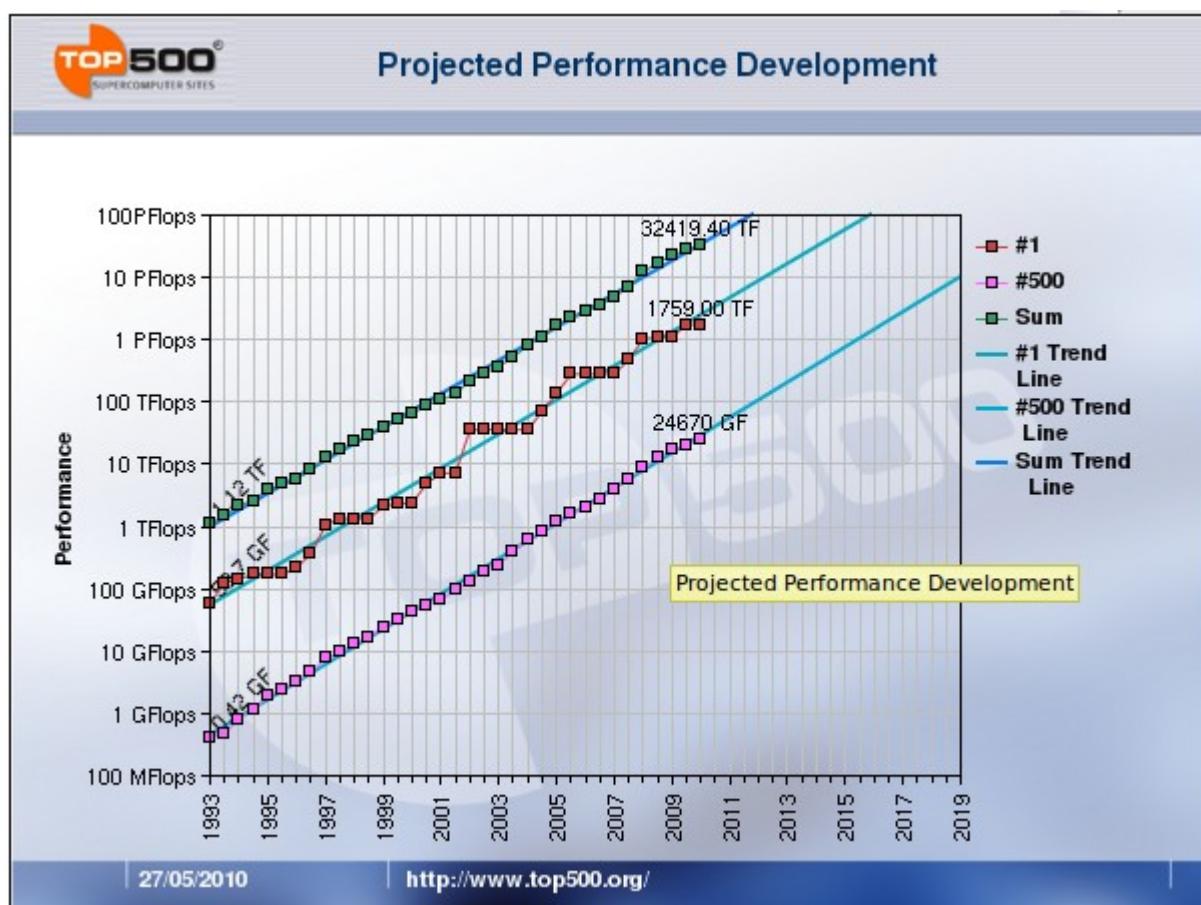
ภาพ 1-9

เครื่อง jaguar ในภาพนั้นมีความเร็วในการประมวลผลสูงสุดทางทฤษฎี Peak Performance เท่ากับ 1.75 Petaflops (10^{15} Floating point operations per second) ประกอบไปด้วย AMD Opteron processor core เป็นจำนวน 244,256 core เป็นเครื่องที่สร้างขึ้นมาจากการร่วมมือของกระทรวงพลังงานสหรัฐกับบริษัท Cray เราจะพูดถึงสถาปัตยกรรมของเครื่อง Jaguar และเครื่อง Supercomputer ในบทถัดไป การจัดอันดับของเครื่อง

คอมพิวเตอร์เหล่านี้ทำโดยวัดประสิทธิภาพในการรัน Linpack Benchmark

นอกจากเครื่อง jaguar และเครื่อง supercomputer ที่มีความเร็วเป็นอันดับสองก็คือ เครื่อง Dawning Nabulae ของประเทศจีน ตั้งอยู่ที่ the National Supercomputing Center, in Shenzhen มีประสิทธิภาพ 1.26 Petaflops ประกอบไปด้วย Processor Intel Xeon 5650 และ NVIDIA Tesla

ในแห่งของประสิทธิภาพนั้นระบบคอมพิวเตอร์สมรรถนะสูงได้รับการพัฒนาให้มี ประสิทธิภาพสูงขึ้นอย่างต่อเนื่องตลอดมาสอดคล้องกับ Moore's law หรือคำพูดของนัก วิทยาศาสตร์คอมพิวเตอร์ที่มีชื่อเสียงมากคนหนึ่งของโลกคือ Gordon Moore ที่ว่าจำนวนของ transistor ที่ถูกบรรจุลงสู่ Microprocessor นั้นจะเพิ่มเป็นสองเท่าทุกๆ สองปี เราอาจตีความ อย่างหยาบๆ ได้ว่าประสิทธิภาพของคอมพิวเตอร์จะเพิ่มขึ้นเป็นสองเท่าทุกๆ สองปี ภาพ 1-10 แสดงพัฒนาการของประสิทธิภาพของคอมพิวเตอร์ที่อยู่ใน Top 500 list ซึ่งสอดคล้องกับกฎดัง กล่าว



ภาพ 1-10 แสดง chart จาก www.top500.org

ระบบ Supercomputer นั้นมีอยู่หลายแบบดังที่จะได้พูดถึงมากขึ้นในบทที่ 2 แต่อย่างไร ก็ตามระบบคอมพิวเตอร์เหล่านี้จะมีลักษณะบางอย่างร่วมกันคือมีขนาดใหญ่ใช้พื้นที่มากดัง แสดงในภาพ 1-11 และประกอบไปด้วย Processors จำนวนมาก (บางที่เราจะเรียกเครื่อง คอมพิวเตอร์เหล่านี้ว่าเป็นระบบ Massively Parallel Processors หรือ MPP) และมี High Speed Network เชื่อมต่อส่วนประกอบต่างๆเข้าด้วยกัน



ภาพ 1-11 Supercomputers จากบริษัทผู้ผลิตที่แตกต่างกัน ภาพนำมายจาก

https://computing.llnl.gov/tutorials/parallel_comp/

จากภาพจะเห็นว่าเครื่องคอมพิวเตอร์เหล่านี้มีขนาดใหญ่และใช้พลังงานไฟฟ้าค่อนข้างมากในการประมวลผลและทำความเย็นให้กับระบบ จากเว็บ www.top500.org เครื่อง supercomputer 10 อันดับแรกใช้พลังงานโดยเฉลี่ย 1.32 Mega Watt ในการรัน Linpack Benchmark

1.2.2 Commodity Cluster

ในขณะที่องค์กรใหญ่สามารถซื้อหาเครื่อง supercomputer มาใช้ได้ องค์กรระดับกลาง หรือนักวิจัยทั่วๆไปก็สามารถสร้างเครื่องคอมพิวเตอร์เพื่อสนับสนุนการทำงานแบบ Parallel Processing ของตนเองขึ้นมาได้เช่นกัน แนวคิดที่จะนำพลังจากการทำงานแบบ Parallel มาใช้แก้ปัญหาขนาดใหญ่ในแนวทางนี้เน้นการนำอุปกรณ์คอมพิวเตอร์ที่หาซื้อได้ตามท้องตลาด ไม่จำเป็นต้องสร้างขึ้นมาพิเศษเหมือนใน supercomputer และนำอุปกรณ์เหล่านั้นมาสร้างเป็น

เครื่องคอมพิวเตอร์สมรรถนะสูงขึ้นมา ซึ่งก็เป็นแนวคิดพื้นฐานในการพัฒนาระบบ Cluster Computer นั่นเอง

ในช่วงปี 1980 ได้มีโครงการในแนวนี้สองโครงการได้แก่ โครงการ Beowulf นำโดย Thomas Sterling จาก CalTech และโครงการ Network of Workstations (NOW) นำโดย David Culler แห่ง UC Berkeley ซึ่งทั้งสองโครงการได้วางรากฐานของระบบคลัสเตอร์ในปัจจุบัน และโครงการ Beowulf ได้กลายมาเป็นโครงสร้างพื้นฐานของระบบคลัสเตอร์ คอมพิวเตอร์ที่แพร่หลายเป็นอย่างมากในปัจจุบัน

ระบบคลัสเตอร์ที่ได้รับการพัฒนาขึ้นในโครงการ Beowulf นั้นนำคอมพิวเตอร์ที่มีราคาไม่แพงและ Network ที่ราคาไม่แพงมากและหาได้จากท้องตลาดทั่วไป มาประกอบกันเข้าเป็น คอมพิวเตอร์ที่ทำงานร่วมกันแบบ Parallel ภาพ 1-12 แสดงเครื่องคลัสเตอร์คอมพิวเตอร์ชื่อ HiTorque ที่ประกอบขึ้นที่ ORNL จากคอมพิวเตอร์แบบ Pentium III หลายเครื่องและเชื่อมต่อกันด้วย Fast Ethernet Network



ภาพ 1-12 HighTORC cluster (Source: ornl): An affordable Linux cluster

ในปัจจุบันระบบคลัสเตอร์ได้รับการพัฒนาขึ้นโดยผู้ผลิตหลายอาทิ Dell HP และ IBM และมีใช้กันอยู่ทั่วไปเราจะพูดถึงโครงสร้างและการใช้งานมันในบทที่ 3 ถึง 5 ต่อไป นอกจาความสะดวกในการซื้อหาอุปกรณ์ hardware และระบบคลัสเตอร์ยังได้รับการออกแบบให้ใช้พื้นที่ได้มากน้อยขึ้นอยู่กับความต้องการของผู้ใช้ ภาพ 1-13 แสดงระบบคลัสเตอร์ที่ TACC

ที่ The University of Texas เป็นระบบที่พัฒนาขึ้นโดยบริษัท Dell (ซึ่งเป็นบริษัทที่เน้นการผลิต PC เป็นหลัก) และบริษัท Cray ในขณะเดียวกัน นักวิจัยจาก Sandia National Laboratory ก็ได้พัฒนาเครื่องคลัสเตอร์คอมพิวเตอร์ขนาดเล็กขึ้นมาซึ่งเรียกว่า Cluster in the bread box ดังแสดงในภาพ 1-14

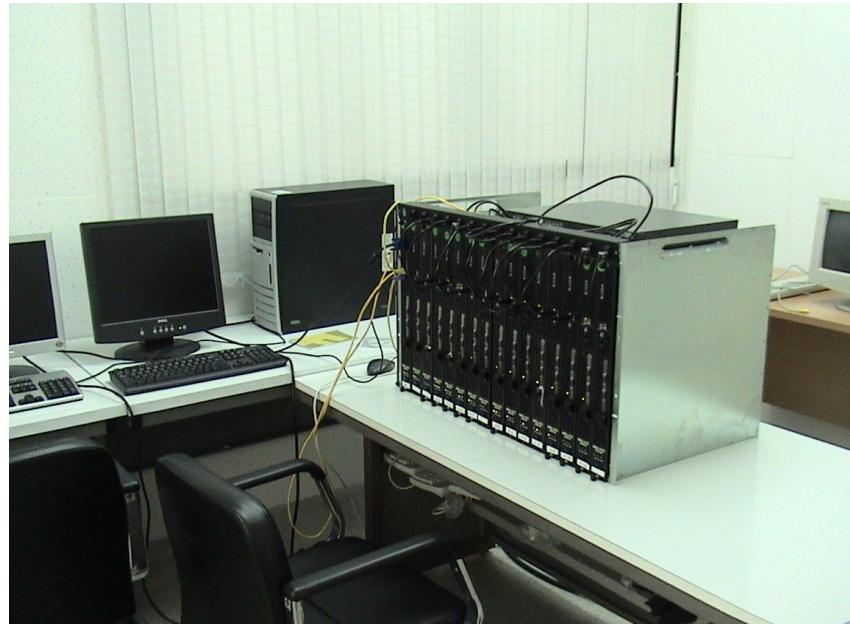


ภาพ 1-13 Lonestar Cluster ผลิตโดยบริษัท Dell และ Cray จาก TACC



ภาพ 1-14 Cluster in the bread box จาก Sandia National Laboratory
<http://www.sandia.gov/media/NewsRel/NR2002/cluster-computer.html>

ภาพ 1-15 แสดงระบบคลัสเตอร์ที่ภาควิชาวิทยาการคอมพิวเตอร์ ม.ธรรมศาสตร์ ชื่อ hpccluster ซึ่งประกอบไปด้วยคอมพิวเตอร์แบบ Blade (เครื่อง PC ที่ทำให้บางเพื่อจะได้เสียบลงในตู้ใส่หรือ Chasis ได้หลายเครื่อง) 16 เครื่อง แต่ละเครื่องใช้ Processor Intel Xeon 2 อัน และเชื่อมต่อกันด้วย 1 Gigabit Ethernet Network



ภาพ 1-15 A 16-Nodes 32 CPU cluster computer contains blade servers in a chassis.

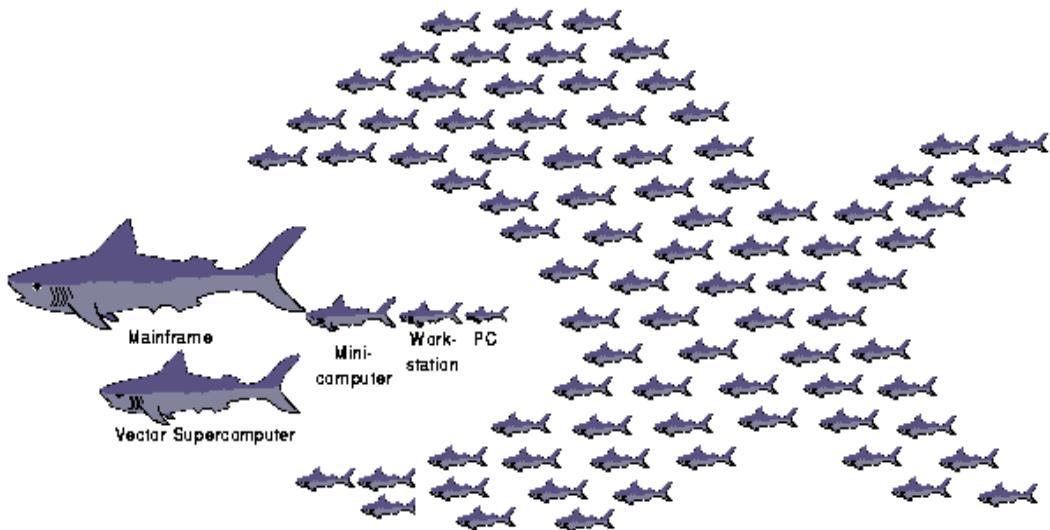
ระบบคอมพิวเตอร์อีกรูปแบบหนึ่งที่ได้รับพัฒนาขึ้นในช่วง 1980 และเป็นพื้นฐานของระบบ Cluster และ MPP ในปัจจุบันได้แก่ระบบ Network Of Workstations (NOW) ซึ่งในช่วงนั้น เป็นหนึ่งในระบบคอมพิวเตอร์ที่อยู่ใน 500 อันดับแรกด้วย ปริมาณพื้นฐานของระบบ NOW นั้น แสดงดังภาพ 1-15 คือเช่นว่าการนำพลังของเครื่องคอมพิวเตอร์ระดับ Workstations ซึ่งเทียบเท่ากับ High-end PC ในปัจจุบัน

จากภาพ 1-15 ปลาใหญ่ที่สุดในภาพจะเป็น Supercomputer ในขณะนั้นและตัวรองลงมา ก็เป็น mainframe และ minicomputer ตามอันดับ ถัดจากนั้นก็เป็น Workstations และ PC เล็กที่สุด จุดประสงค์ของโครงการนี้คือการสร้าง Supercomputer จาก Workstations ดังนั้นเป้าหมายคือระบบที่มีความสามารถสูงและเอาชนะ Supercomputer ในระดับนั้นได้ ซึ่งโครงการนี้ก็ประสบความสำเร็จตามนั้นโดยที่เครื่อง NOW สามารถ sort ค่าได้เร็วกว่าระบบอื่นๆ

ส่วนประกอบของ NOW ดังแสดงในภาพ 1-16 นั้นจะแตกต่างจากระบบ Commodity Cluster เพราะส่วนประกอบทุกส่วนจะมีราคาแพงและมีประสิทธิภาพสูง เช่นใช้ SUN workstation และใช้ Network ราคาแพงมากได้แก่ Myrinet Network แทนที่จะใช้ชิ้นส่วนที่ซื้อหาได้ง่ายและราคาถูกเหมือนโครงการ Beowulf

ระบบ NOW ทำให้เกิดการพัฒนา software ขึ้นมาเพื่อควบคุมการทำงานของ workstation เหล่านี้ให้มีประสิทธิภาพดังแสดงในภาพ 1-17

The Berkeley NOW Project

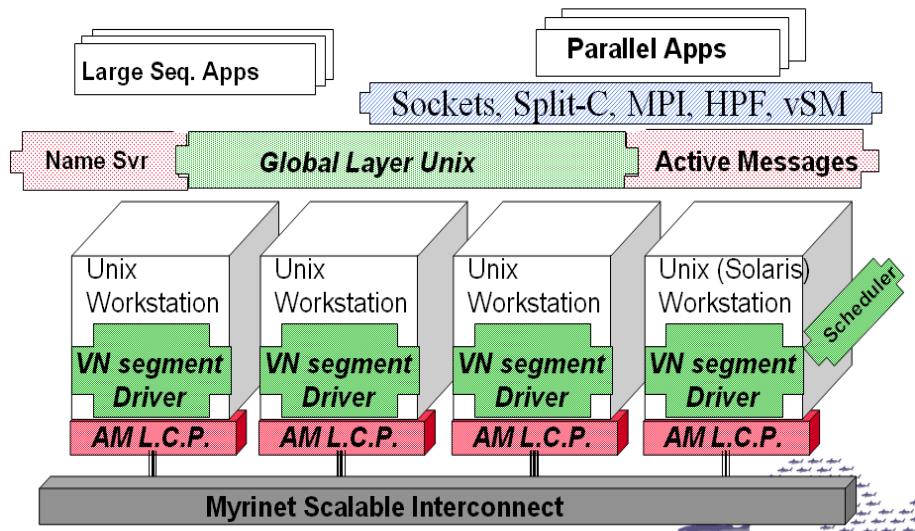


ภาพ 1-15 <http://now.cs.berkeley.edu/>



ภาพ 1-16 ระบบ NOW

NOW Software Components



ภาพ 1-17 NOW's Software Stack จาก Source:

<http://www.cs.berkeley.edu/~pattrsn/Arch/prototypes2.html>

1.2.3 Volunteer Computing

ในแต่ละวันมีเครื่องคอมพิวเตอร์จำนวนนับพันล้านเครื่องที่ต่ออยู่บน Internet เครื่องคอมพิวเตอร์เหล่านี้อาจเป็น Notebook หรือ PC หรือแม้แต่ game console เครื่องคอมพิวเตอร์เหล่านี้มีพลังในการประมวลผลรวมกันเป็นจำนวนมากมหาศาล David Anderson ผู้นำโครงการ Berkeley Open Infrastructure for Network Computing (BOINC) ที่ UC Berkeley ได้เปรียบเทียบอย่างคร่าวๆ ไว้ว่าเครื่องคอมพิวเตอร์ PC จำนวน 100,000 เครื่องมีความสามารถในการประมวลผลเท่ากับ Supercomputer 1 เครื่อง (สมมุติ 1 เครื่อง PC มีความเร็ว 1×10^9 Floating point operations per second หรือ 1 Giga-FLOPs และเครื่อง Supercomputer มีความเร็วประมาณ 1 Tera-FLOPs หรือ 10^{15} Floating point operations per second ดังนั้น ต้องใช้เครื่อง PC คอมพิวเตอร์ 10^5 เครื่องจึงจะได้ความสามารถที่เท่าเทียมกัน) แต่โดยประมาณแล้วเราคาดว่าจะมีเครื่อง PC และอุปกรณ์ประมวลผลอื่นๆ เช่น game console และ GPU ประมาณกว่า 1 พันล้านเครื่องทำงานอยู่บน Internet ดังนั้นถ้าเราสามารถนำเอาพลังของเครื่องเหล่านี้มาใช้ในการประมวลผลร่วมกันได้ ก็จะได้พลังเป็นจำนวนมากมหาศาล

Volunteer computing เป็นการประมวลผลที่นำพลังในการประมวลผลมาจากการอาสาสมัครบริจากเวลาในการทำงานของเครื่องคอมพิวเตอร์ที่ติดต่ออยู่ในระบบ Internet เหล่านั้นมา ร่วมกันแก้ปัญหาที่จำเป็นต้องประมวลผลเป็นปริมาณมาก ในปัจจุบันมีโครงการหลายโครงการที่ใช้เทคโนโลยี Volunteer computing นี้ และได้รับความร่วมมือจากเจ้าของคอมพิวเตอร์ทั่วโลก

ซอฟต์แวร์ BOINC เป็น middleware จากโครงการ BOINC ที่จัดการและให้บริการ Volunteer Computing บน Internet มันได้รับการพัฒนาขึ้นมาที่ UC Berkeley และได้รับความร่วมมือเป็นอย่างดีจากผู้บริจากทั่วโลกทำให้มีจำนวนคอมพิวเตอร์ที่เข้าร่วมอยู่ถึง 680,000 คัน จาก 245 ประเทศ และมีคอมพิวเตอร์โดยรวมอยู่ถึง 1,000,000 เครื่อง และมี Disk Space โดยรวมถึง 12 Peta-bytes

มี applications ทางวิทยาศาสตร์หลายอย่างที่ได้ใช้งานเครื่องคอมพิวเตอร์เหล่านี้โดยใช้ ซอฟต์แวร์ BOINC อาทิเช่น โครงการ [Seti@home](#) โครงการ [folding@home](#) โครงการ [LHC@home](#) เป็นต้น ในโครงการ [folding@home](#) นั้นมีรายงานว่าใช้ความสามารถของ คอมพิวเตอร์ที่ Volunteer โดยรวมถึง 650 Teraflops ใน การประมวลผลซึ่งเป็นประสิทธิภาพ ที่มากกว่าเครื่อง Bluegene/L supercomputer เสียอีก โดยที่ 650 teraflops นี้แบ่งออกเป็น 200 teraflops จากเครื่อง PC และ 50 Teraflops จาก GPU และ 400 Teraflops จาก Play Station 3 นอกนั้นในช่วงเวลา 7 ปีที่ผ่านมาโครงการ [Seti@home](#) ได้ใช้ปริมาณการ ประมวลผลจากคอมพิวเตอร์ที่เดียวเป็นเวลาถึง 27 ล้านปี

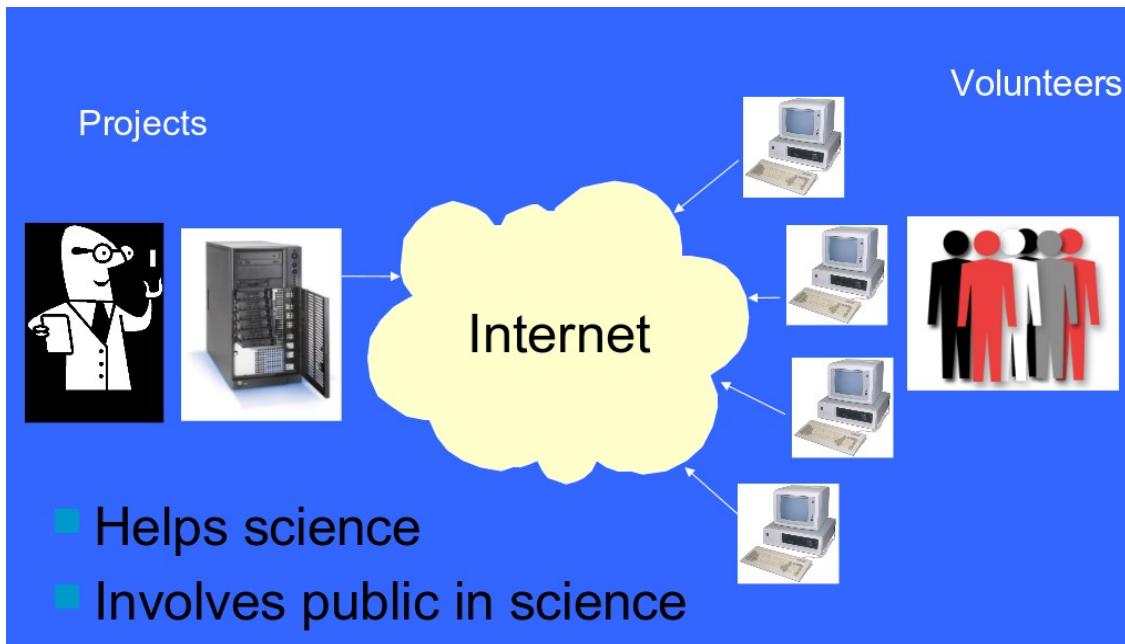
ในทางธุรกิจนั้น ได้มีบริษัทที่ซื้อเวลาจากคอมพิวเตอร์บน Internet และนำเอาความสามารถในการประมวลผลปริมาณมากนั้นไปขายต่อกับผู้ที่มีต้องการในการประมวลมากๆอีกต่อหนึ่ง



สถาปัตยกรรมของ Volunteer Computing

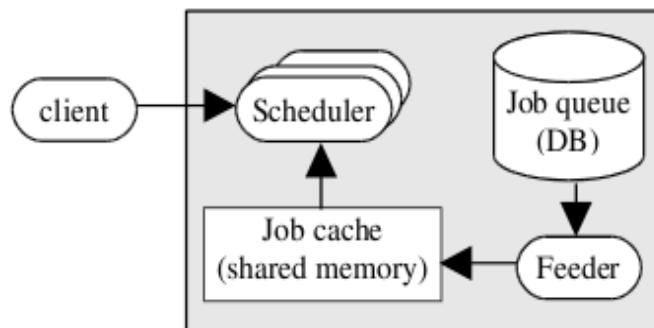
สถาปัตยกรรมของ Volunteer computing ประกอบไปด้วยสองส่วนใหญ่ๆทำงานอยู่บน Internet ได้แก่ server และ client ดังแสดงในภาพ 1-18 client จะส่ง request มาของงานจาก server เมื่อมันพร้อมที่จะทำงาน ในการทำงานแบบนี้มักจะมี client เป็นจำนวนมากบน internet ทำให้มีความสามารถโดยรวมเป็นปริมาณมากไปด้วย

server นั้นรับແຜบคอมพิวเตอร์ของโครงการที่ต้องการอาสาสมัครมาช่วยการประมวลผลและเป็นผู้ส่งงานซึ่งประกอบไปด้วยข้อมูลที่ใช้ในการประมวลผลและโปรแกรมที่จะรับบน client ภาพ 1-19 แสดงโครงสร้างของ server ซึ่งมันต้องมีความสามารถในการจ่ายงานให้กับ client ด้วยความเร็วที่สูง ถ้า server ต้องค้นหางานที่เหมาะสมให้กับ client โดยการ search database ทุกรังก์จะต้องใช้เวลาในการหาค่อนข้างมาก ดังนั้น server จึงได้รับการออกแบบให้เลือกงานจาก job cache ที่มีขนาดไม่ใหญ่มากและมีโปรแกรม Feeder แยกออกมาเป็นโปรแกรมที่ทำหน้าที่เลือกงานจาก database ไปใส่ใน cache

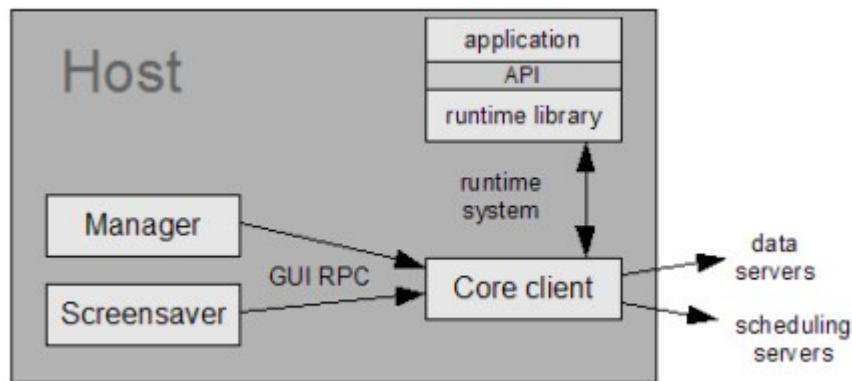


ภาพ 1-20

ส่วน client นี้จะรันงานให้กับ server เมื่อมันพร้อม โดยส่ง request ไปยัง server และรอรับ ส่งผลกลับให้ server เมื่อทำงานให้เสร็จ บน client จะมี client core ที่จะฝังตัวและดำเนินการต่างๆโดยที่มันจะทำงานเมื่อนโยบายที่ตั้งไว้บนเครื่อง client เป็นจริง เช่นทำงานเมื่อเครื่อง Idle หรือในช่วงไดช่วงหนึ่งของวันเป็นต้น เมื่อมันทำงานมันจะเรียก job ที่ได้รับมาจาก server ขึ้นมาทำงานในส่วนของ application ในภาพ 1-20 โดยที่ job จะติดต่อกับ client core ผ่าน API ของ BOINC ซึ่ง API เหล่านี้จะเรียกใช้ library ของ BOINC เพื่อดูแลการเข้าถึง resource ของ client หรือติดต่อกับ server หลังจากประมวลผลแล้วโปรแกรมอาจแสดงผลของมันออกมายัง screen saver ส่วนโปรแกรม manager นั้นมีไว้ให้เจ้าของเครื่อง client เรียกดูสถานะของงานที่รันอยู่บนเครื่องของเขากำหนดนโยบายต่างๆ



ภาพ 1-19



ภาพ 1-20

โครงการที่ใช้ Volunteer computing

ตัวอย่างของโครงการหนึ่งที่ได้รับประโยชน์อย่างมากจาก Volunteer computing คือ โครงการ [Seti@home](#) ซึ่งเป็นความพยายามที่จะหาสัญญาณวิทยุของสิ่งมีชีวิตที่มีความคลาดนอกโลกที่อาจส่งสัญญาณวิทยุในรูปแบบที่มีความหมาย โครงการนี้มีจานรับสัญญาณวิทยุขนาดบันทองฟ้าเพื่อรับสัญญาณทุกวัน นอกจากนั้นมันยังมีจานรับสัญญาณขนาดใหญ่ที่ชุดบนยอดเขา ดังแสดงในภาพ 1-22 เนื่องจากสัญญาณที่ได้รับมาปริมาณมหาศาลการวิเคราะห์มันจึงต้องใช้การประมวลผลจำนวนมากและ volunteer computing ก็มีความสามารถที่ช่วยได้



ภาพ 1-22 The 300 meter (1000 foot) Arecibo Radio Telescope in Puerto Rico, where SETI@home collects its data and the reobservations were conducted.

อีกโครงการหนึ่งที่ได้รับประโยชน์จาก Volunteer computing มากก็คือโครงการ [folding@home](#) ที่เป็นการจำลองและวิเคราะห์การพับตัวของโปรตีน ซึ่งเป็นเรื่องที่เกิดขึ้นตาม

ธรรมชาติ การพับตัวของโปรตีนที่มีความผิดปกตินั้นมีความเกี่ยวข้องกับโรคหลายอย่างเช่น Alzheimer และ Parkinson และ BSE เป็นต้น การพับตัวที่ผิดปกติทำให้การทำงานของเซลล์ผิดปกติไปด้วย แต่การจำลองการพับของโปรตีนนั้นจึงเป็นเรื่องที่ยากและต้องการความสามารถในการประมวลผลมาก เพราะมีรายละเอียดมากและมีสเกลของเวลาที่ละเอียดมาก โดยเฉลี่ยแล้ว เครื่องคอมพิวเตอร์หนึ่งเครื่องจะใช้เวลา 1 วันในการจำลองการพับของโปรตีนขนาดเล็กเป็นเวลา 20 nanosecond (10^{-9}) และเมื่อใช้ cell processor ประมวลผลหนึ่งวันก็อาจจำลองการพับได้เป็นช่วงเวลาประมาณ 500 nanosecond แต่การพับของโปรตีนที่มีประโยชน์นั้นจะเป็นการพับในช่วงเวลา หลาย millisecond หรือหลายวินาที ดังนั้นการจำลองการพับของโปรตีนจึงเป็นงานที่ต้องใช้ความสามารถในการคำนวนมหาศาล

โครงการ [folding@home](#) ได้ใช้ volunteer computing มาตั้วแต่ปี 2000 และในปี 2009 มีความเร็วในการประมวลผลถึง 5 Petaflops ซึ่งมากกว่า Supercomputer ที่เร็วที่สุดในโลก ภาพ 1-23 แสดงประสิทธิภาพจากการประมวลผลที่ได้จาก client ชนิดต่างๆ จะเห็นได้ว่ามันได้พลังมาจาก Playstation 3 และ GPU ค่อนข้างมาก

Client Type	TFLOPS	Clients
Windows x86/x64	275	280,000
Mac OS X ppc	5	6,000
Mac OS X x86	25	9,000
Linux x86	50	28,000
ATI GPU	1,125	10,000
NVIDIA GPU	1,900	17,000
PLAYSTATION 3	1,400	50,000
Total (approximate)	4,800	400,000

ภาพ 1-23 Approximate [folding@home](#) host statistics (Feb 2009)

1.2.4 Grid Computing

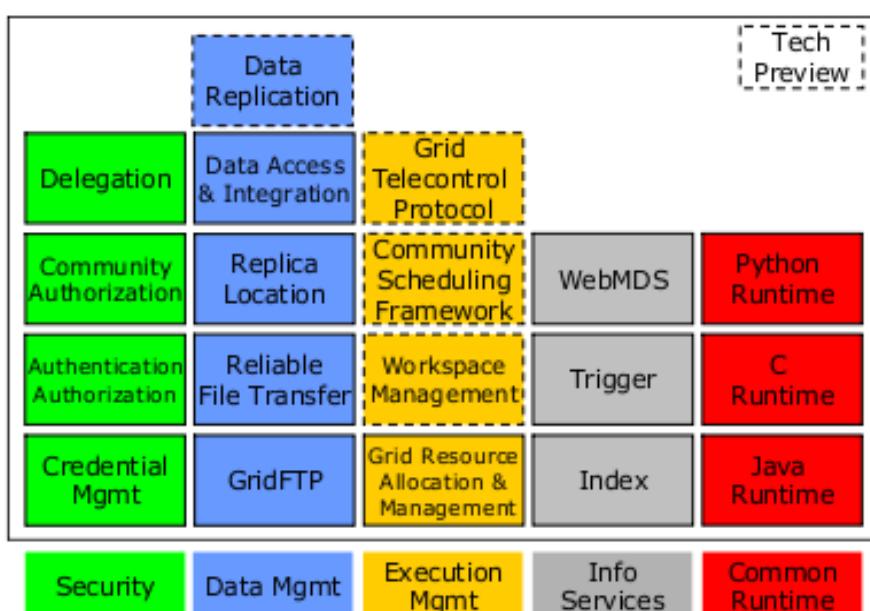
Grid computing เป็นรูปแบบของ Distributed computing แบบหนึ่งที่ต้องการนำพลังความสามารถในการประมวลผลจากหลายองค์กรที่ต่างองค์กรต่างมีการบริหารจัดการ computing resources และนโยบายในการรักษาความปลอดภัยที่แตกต่างกันมาใช้งานร่วมกัน การใช้งาน resource ร่วมกันแบบนี้นั้นสอดคล้องกับระบบธุรกิจสมัยใหม่ที่มีการสร้างทีมงานจากบริษัทหลายบริษัทมาทำงานร่วมกันในโครงการเฉพาะกิจบางอย่างเป็นลักษณะของ Virtual Team และเนื่องจากโครงการเฉพาะกิจเหล่านั้นอาจต้องใช้ resource ในการประมวลผลและแทนที่จะซื้อหา resources มาใหม่ทั้งหมด Grid computing ก็มีเครื่องมือให้องค์กรสามารถนำเอา resources ที่แต่ละองค์กรมีอยู่มาใช้งานร่วมกันอย่างปลอดภัยเรียกว่า Virtual

Organization เพื่อตอบสนองความต้องการของ virtual team

คำว่า Grid computing เป็นคำที่แต่งขึ้นโดย Ian Foster และ Carl Kesselman ในช่วงปี 90 เป็นการเปรียบเทียบ computing infrastructure แบบนี้ว่าเหมือนกับระบบ Electrical grid ที่เป็นการเชื่อมต่อของแหล่งจ่ายไฟเป็น grid ที่แผ่กระจายอยู่ทั่วประเทศและคนที่ใช้ไฟไม่จำเป็นต้องรู้ว่าพลังงานไฟฟ้าที่ตนใช้นั้นจะได้มาจากแหล่งจ่ายไฟไหน ถ้าแหล่งจ่ายไฟแหล่งหนึ่งไม่ทำงาน ไฟฟ้าก็จะถูกนำมายังผู้ใช้จากแหล่งอื่นแทน เช่นเดียวกัน พลังงานในการประมวลผลก็ควรที่จะสามารถได้จากแหล่งประมวลผลที่อยู่บน Grid ของการประมวลผล หรือ Computational Grid Infrastructure

ระบบ Grid software ได้รับการพัฒนาขึ้นภายใต้โครงการ Globus ที่ ANL ซึ่งมี Ian Foster เป็นผู้นำโครงการ โครงการนี้ได้รับการพัฒนาอย่างต่อเนื่องตั้งแต่ช่วงปี 90 จนในปัจจุบัน มีซอฟต์แวร์ Globus 4 เป็นเวอร์ชันที่ 4 และมีมาตรฐานของ Interface และ protocol ที่ได้รับการกำหนดโดย Grid forum

สิ่งที่เป็นความท้าทายในการสร้าง Grid software ก็คือการเข้าถึง resource ที่มีอยู่เบย และวิธีการรักษาความปลอดภัยที่แตกต่างกัน เพราะผู้ที่เป็นเจ้าของ resource ย่อมไม่ต้องการให้ resource ของตนถูกใช้โดยที่ตนไม่ได้รับรู้ว่าถูกนำไปใช้อย่างไร Grid software ต้องการันตีความปลอดภัยในการเข้าถึง resources และต้องมีเครื่องมือสำหรับส่งงานให้มีการ execute งานบน resources ต่างๆในระบบ นอกจากนั้นก็ต้องมีเครื่องมือสำหรับการส่งข้อมูลจำนวนมากอย่างมีประสิทธิภาพบน Internet อีกด้วย



ภาพ 1-24 จาก paper ของ Ian Foster เรื่อง Globus Toolkits version 4: Software for service-oriented systems

ภาพ 1-24 แสดง component ต่างๆของ Globus 4 software ประกอบไปด้วย 4 ส่วนดังนี้

Security: ระบบซอฟต์แวร์ที่จัดการเรื่องความปลอดภัยในการใช้งาน resources ต่างๆในระบบ Grid นั้นเรียกว่า Grid Security Infrastructure (GSI) มันสร้างอยู่บนพื้นฐานของ Public-Key Infrastructure หรือ PKI ซึ่งใช้การออก X.509 certificate ให้กับสมาชิกใน Virtual Organization และมีการอนุญาติให้ผู้ใช้ที่ได้รับการออก certificate ให้สามารถเข้าถึง resource ที่อยู่ใน virtual organization โดยไม่ต้องใช้ password ทำให้สามารถสร้าง applications ที่ทำงานบน resources เหล่านี้ได้โดยง่าย

GSI ใช้หลักการ certificate delegation ของ PKI ในการ access resource โดยไม่ใช้ password หลักการนี้ได้รับการขยายไปใช้ใน myproxy software เพื่อให้ web portal สามารถเข้าถึง remote resource ได้ด้วย

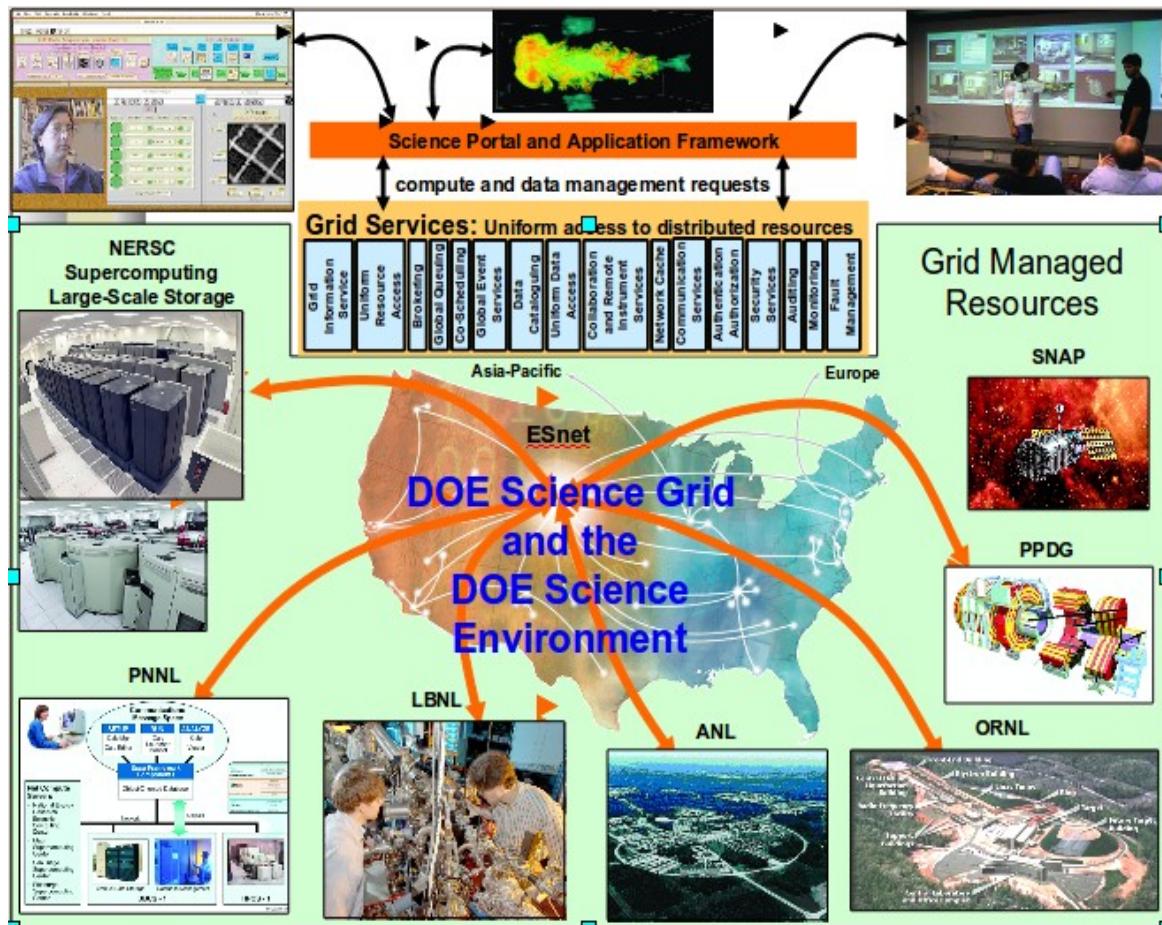
Data Management: เนื่องจากข้อมูลที่ใช้ในการประมวลผลในระบบ grid นั้นมีเป็นจำนวนมาก และมักจะต้องมีการออนไลน์ข้อมูลข้าม Internet ดังนั้น Globus 4 software จึงมี GridFTP software สำหรับถ่ายโอนข้อมูลเหล่านั้น GridFTP ทำงานภายใต้ GSI โดยที่มันจะเช็คสิทธิในการเข้าถึง resources ทั้งบนผู้ส่งและผู้รับแล้วถึงจะเริ่มต้นการถ่ายโอน ผู้ใช้สามารถเพิ่มประสิทธิภาพของการถ่ายโอนได้โดยการสร้าง TCP connection หลายๆ connection เพื่อช่วยกันถ่ายโอนข้อมูล นอกจากนั้น Globus 4 ยังมีซอฟต์แวร์ชื่อ Reliable File Transfer (RFT) ซึ่งจะทำงานบน GridFTP อีกด้วยและมีความสามารถที่จะคุ้มครองการทำงานได้เมื่อเกิดความผิดพลาดในระหว่างที่มีการถ่ายโอนข้อมูลขึ้น โดยที่ RFT จะทำการถ่ายโอนข้อมูลจากตำแหน่งล่าสุดที่ GridFTP ได้ถ่ายโอนข้อมูลไปแล้วแทนที่จะเริ่มต้นใหม่ทั้งหมด

Execution Management: Globus 4 มีบริการให้ผู้ใช้สามารถ รัน job ข้ามเครื่องคอมพิวเตอร์ในระบบ Grid ได้โดยใช้ GSI ตรวจสอบสิทธิของผู้ใช้ในการเข้าถึง resource เหล่านั้นแล้วจึงส่ง job ไปยัง resource นั้นเพื่อรัน ซอฟต์แวร์ที่ใช้ประกอบไปด้วย client interface library และ GSI-enabled server ที่ถูกสร้างขึ้นโดยใช้โปรโตคอล Web Services

Info Service: เนื่องจาก resource ภายใต้ VO หนึ่งๆอาจมีเป็นจำนวนมาก Globus software จึงจำเป็นต้องมี Directory Service protocol เพื่อเก็บข้อมูลเกี่ยวกับ resource เหล่านั้น ข้อมูลเหล่านี้ได้แก่ชื่อของ resource ชนิดของ CPU และ workloads ขนาดของ memory ขนาดของ Disk และ Network Bandwidth ของ resource เหล่านั้นเป็นต้น Globus 4 software มีเครื่องมือที่ใช้ในการ search หา resource ตามที่ผู้ใช้ต้องการด้วย

Common Runtime: ผู้ใช้สามารถพัฒนาซอฟต์แวร์เพื่อใช้งาน Grid resource โดยใช้ภาษา C หรือ Java หรือ Python โดยที่ Globus 4 ได้กำหนด API สำหรับภาษาเหล่านี้ไว้และมี programming library สำหรับให้ผู้พัฒนาซอฟต์แวร์ได้เรียกใช้บริการต่างๆของ Globus ดังที่ได้กล่าวมาแล้วข้างต้นด้วย

ภาพ 1-25 แสดงตัวอย่างของโครงการ DOE Science Grid ที่ต้องการจะนำเอา Supercomputer หลายเครื่องจากศูนย์วิจัยต่างๆ ของกระทรวงพลังงานประเทศไทยและรัฐอเมริกามาใช้งานร่วมกันเพื่อการประมวลผลทางวิทยาศาสตร์ เครื่อง supercomputer เหล่านี้อยู่ภายใต้การบริหารจัดการที่แตกต่างกันและมีนโยบายรักษาความปลอดภัยที่แตกต่างกัน แต่ทั้งหมดก็จะรัน Globus 4 software เพื่อสร้าง middleware runtime system บนเครื่องคอมพิวเตอร์เหล่านี้



ภาพ 1-25 จาก <http://www.doesciencegrid.org>

ก่อนที่จะมีการใช้งาน resource ร่วมกันได้โครงการนี้ได้กำหนดให้มี DOE Science Grid VO และสร้าง Certificate Authority ขึ้นเพื่อออก certificate ให้กับผู้ใช้ในองค์กร

ผู้ใช้สามารถพัฒนา application ของตนเพื่อใช้งาน grid resource เหล่านี้ร่วมกัน โดยที่ application สามารถเรียกใช้ Grid services หรือบริการต่างๆ ของ Globus ได้แก่ GSI และ remote execution และ Data management ผ่าน API และ common runtime system ดังที่แสดงในภาพ 1-24 เพื่อเข้าถึง resource ต่างๆ

บทที่ 2 สถาปัตยกรรมของระบบคอมพิวเตอร์สมรรถนะสูง

เนื้อหา เรียนเรียงจากวิศวอุกคิปชอง Prof. Dr. Thomas Sterling ผู้เชี่ยวชาญทางด้าน High Performance Computing จาก Louisiana State University
แปลและเรียนเรียง กษิดิศ ชาญเชี่ยว

High performance computing เป็นวิชาที่เป็น มัลไทริสสิพลินนารี คือเกี่ยวข้องกับ 略有สาขาวิชา แล้วเราจะศึกษาความสัมพันธ์ระหว่างวิชาเหล่านี้อย่างไร?

วิธีการแรก เราอาจใช้แนวทางศึกษาแบบ Bottom Up คือการเริ่มศึกษาตั้งแต่การสร้าง และออกแบบ logic circuit และสูงขึ้นไปเรื่อยๆจนถึงระดับบันคือ application software แต่ข้อเสียของวิธีการนี้คือเราจะมองไม่เห็นภาพรวมจนกว่าจะศึกษาถึงระดับบันสุด

ในแนวทางที่สอง ซึ่งเป็นวิธีที่ใช้ในการเรียนการสอน supercomputing เป็นต้นเป็นส่วนใหญ่ ก็คือการสร้างโปรแกรมโดยใช้ภาษาโปรแกรม หรือ programming library หรือใช้ทั้งสองสิ่ง ร่วมกัน แต่ข้อเสียของแนวทางนี้คือเราจะไม่ได้ศึกษาเรื่องประสิทธิภาพหรือ performance ของระบบ

จากประวัติศาสตร์ของการประมวลผลข้อมูล นักวิทยาศาสตร์ได้สังเกตเห็นความสัมพันธ์ระหว่างสามองค์ประกอบสำคัญของวิทยาการคอมพิวเตอร์ได้แก่ เทคโนโลยีพื้นฐาน สถาปัตยกรรมคอมพิวเตอร์ และโปรแกรมมิ่งโมเดล เราจะเห็นว่าสถาปัตยกรรมคอมฯคือการใช้ประโยชน์จากเทคโนโลยีพื้นฐาน และ โปรแกรมมิ่งโมเดลคือการใช้ประโยชน์จาก สถาปัตยกรรมคอม อีกต่อหนึ่ง

ด้วยเหตุนี้ เราจะใช้แนวทางการศึกษา high performance computing วิธีที่สามคือเริ่มต้นจากการศึกษาสถาปัตยกรรมคอมพิวเตอร์ ซึ่งเป็นโซ่อุปกรณ์ แล้วขยายไปศึกษาเทคโนโลยีพื้นฐานและโปรแกรมมิ่งโมเดลหลังจากนั้น

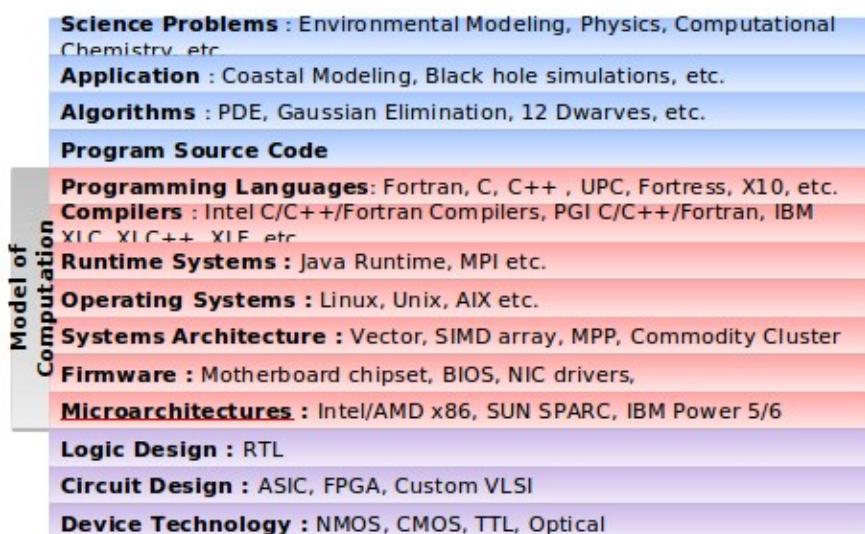
2.1 HPC system Stack

เพื่อให้คุณมีความเข้าใจเนื้อหาวิชานี้ ผมจะยกเรื่องสำคัญเกี่ยวกับสถาปัตยกรรมคอมพิวเตอร์มา พูดช้าเป็นระยะๆตลอดวิชานี้ สำหรับวันนี้ผมจะพูดถึงภาพรวมของสถาปัตยกรรมคอมพิวเตอร์ที่ คุณควรจะรู้จัก ผมจะบอกก่อนนะครับว่า ในวิชานี้ เราไม่ได้ศึกษาการออกแบบวงจร แต่เราจะ พิจารณาฟังชั่นการทำงานของวงจรและอุปกรณ์ และประสิทธิภาพของมัน

ภาพ 2-1 แสดงภาพรวมของ System Stack ของ high performance computing ซึ่ง
ล่างสุดแสดงเทคโนโลยีพื้นฐาน ประกอบไปด้วยการออกแบบจรรยาและ Device technology ซึ่ง
อยู่ในส่วนสิ่งของภาพ การเปลี่ยนแปลงของเทคโนโลยีในระดับนี้จะส่งผลกระทบต่อ
ประสิทธิภาพการทำงานทั้งหมดของระบบ



HPC System Stack



Department of Computer Science CSC 7600 Lecture 2 : Parallel Computer
Louisiana State University Architecture, Spring 2009

5

ภาพ 2-1

ถ้าคุณมองที่ด้านบนของ System Stack ก็จะประกอบไปด้วย Application ทาง
วิทยาศาสตร์หรือวิศวกรรมศาสตร์หรือด้านอื่นๆที่ผู้ใช้งานใจ เรายังมีอัลกอริทึ่มสำหรับแก้ปัญหา
ให้กับ Application เหล่านั้น และในที่สุดเราจะต้องนำเอาอัลกอริทึ่มเหล่านั้นมาเขียนเป็น
โปรแกรมอุปกรณ์สำหรับระบบคอมพิวเตอร์ทำงาน

ส่วนกลางของ System Stack ก็คือระบบคอมพิวเตอร์นั้นเอง ซึ่งเป็นส่วนที่รับคำสั่งจากผู้
ใช้และประกอบไปด้วยกลไกและนวัตกรรมที่ถูกสร้างขึ้นเพื่อใช้งานเทคโนโลยีพื้นฐานให้มี
ประสิทธิภาพและคุ้มค่ามากที่สุดเท่าที่จะทำได้

ในวันนี้เราจะพูดถึง layer สอง layer ในระดับกลางได้แก่ Micro Architecture และ
System Architecture ผู้คนไม่สามารถพูดถึงรายละเอียดทั้งหมดของทั้งสองเรื่องได้ในเวลา

อันสั้น คงต้องใช้เวลาเป็นทั้งเทอม แต่ผมจะพูดถึงหลักการที่สำคัญของมันเพื่อให้คุณสามารถประเมินสถาปัตยกรรมคอมพิวเตอร์หลายแบบได้ และรู้ว่าแต่ละสถาปัตยกรรมมีประสิทธิภาพในการทำงานอย่างไร

Micro Architecture คือสถาปัตยกรรมละเอียดที่กำหนดการทำงานร่วมกันของ logic gate ต่างๆ ส่วน System Architecture นั้นหมายถึงสถาปัตยกรรมโดยรวมของระบบคอมพิวเตอร์ทั้งระบบ จากภาพคุณจะเห็นว่ามี layer อื่นๆ เหนือขึ้นไปจาก System Architecture อีกด้วยเฉพาะอย่างยิ่ง Operating Systems และ Runtime Systems และ Compiler ซึ่งเราจะศึกษามันในอันดับถัดไป

2.2 ตัวแปรที่เกี่ยวข้องกับประสิทธิภาพ

ก่อนอื่นเราจำเป็นต้องเข้าใจวิธีการพื้นฐานในการวัดประสิทธิภาพของระบบสองอย่าง อันดับแรกคือ Response Time ซึ่งหมายถึงระยะเวลาตั้งแต่เริ่มต้นการกระทำอย่างใดอย่างหนึ่งจนกระทั่งการกระทำนั้นเสร็จสิ้น หน่วยของมันจะเป็นเศษส่วนของวินาที เช่น นาโน seconds หรือ ไมโคร seconds ขึ้นอยู่กับสิ่งที่เราจะวัด



Performance Factors: Technology Speed

LSU

- Latencies
 - Logic latency time
 - Processor to memory access latency
 - Memory access time
 - Network latency
- Cycle Times
 - Logic switching speed
 - On-chip clock speed (clock cycle time)
 - Memory cycle time
- Throughput
 - On-chip data transfer rate
 - Instructions per cycle
 - Network data rate
- Granularity
 - Logic density
 - Memory density
 - Task Size
 - Packet Size

อีกวิธีการหนึ่งสำหรับวัดประสิทธิภาพคือการวัดจำนวน operations ที่ทำเสร็จภายในหนึ่งหน่วยเวลา ในการทำงานแบบ sequential การวัดประสิทธิภาพทั้งสองแบบเป็นสัดส่วนผกผันซึ่งกันและกัน

แต่มันไม่เป็นเช่นนั้นในการทำงานแบบ parallel เราแยกการวัดทั้งสองแบบออกจากกัน ซึ่งเราจะได้ศึกษาความแตกต่างของทั้งสองวิธีนี้ต่อไป

จากสไลด์ Latency (ลาเทนซี) คือระยะเวลาที่สิ่งใดสิ่งหนึ่งเดินทางเป็นระยะทางหนึ่ง ยกตัวอย่างเช่น Memory Access Time หรือเวลาที่ข้อมูลเดินทางจาก Memory までの Processor หรือเวลาที่ packet เดินทางจากคอมพิวเตอร์เครื่องหนึ่งไปที่อีกเครื่องหนึ่งบนระบบ Network

Cycle Time หมายถึงระยะเวลาที่ระบบหรือส่วนใดส่วนหนึ่งของระบบใช้ก่อนที่มันจะสามารถให้บริการได้อีกรอบหนึ่ง ใน Memory บางชนิด Cycle Time และ Access Time ของ Memory นั้นไม่เหมือนกัน การเข้าถึงและดึงข้อมูลจาก Memory แต่ละครั้งนั้นทำได้เร็วมากแต่หลังจากนั้นระบบ Memory ต้องทำอะไรบางอย่างก่อนที่มันจะสามารถให้บริการได้อีกรอบหนึ่ง

ในอันดับถัดไป Throughput (ทรัพุท) คือจำนวน operations ที่ระบบทำงานเสร็จต่อหนึ่งหน่วยเวลา ในระบบที่ทำงานแบบ sequential ค่าที่ได้คือค่าผกผันของ Cycle Time นั้นเอง แต่ มันไม่เป็นเช่นนั้นในการทำงานแบบ parallel เพราะระบบหลาย ๆ ระบบสามารถทำงานได้พร้อมๆ กัน

การวัดประสิทธิภาพการทำงานของระบบหนึ่งขึ้นกับสิ่งที่เรียกว่า Granularity ซึ่งหมายถึงขนาดของการประมวลผล และขนาดของข้อมูลสำหรับการประมวลผลบนระบบใดระบบหนึ่ง ยกตัวอย่างเช่น คุณอาจมี Global System หรือระบบหภาคอยู่ระบบหนึ่งซึ่งประกอบไปด้วยระบบย่อยหลาย ๆ ระบบ เมื่อการประมวลผลของระบบหภาคเกิดขึ้นจะมีการแบ่งงานไปยังระบบย่อยและประสานงานระหว่างระบบย่อยเหล่านั้น ในตัวอย่างนี้ขนาดของการประมวลผลของระบบย่อยแต่ละระบบก็คือ Granularity นั้นเอง

Granularity นั้นมีความสัมพันธ์กับประสิทธิภาพของระบบหภาค เพราะการแบ่งงานและประสานงานระหว่างระบบย่อยจะทำให้เกิด Overhead และ Latency ซึ่งเป็นปัจจัยสำคัญต่อประสิทธิภาพในการประมวลผลโดยรวมของระบบ

เราสามารถประเมินค่าคร่าวๆ ของ Response Time Cycle Time และ Throughput ของระบบคอมพิวเตอร์ โดยสังเกตจากค่า Parameters ต่างๆ ของระบบคอมพิวเตอร์แต่ละระบบ



Machine Parameters affecting Performance

- Peak floating point performance
- Main memory capacity
- Bi-section bandwidth
- I/O bandwidth
- Secondary storage capacity
- Organization
 - Class of system
 - # nodes
 - # processors per node
 - Accelerators
 - Network topology
- Control strategy
 - MIMD
 - Vector, PVP
 - SIMD
 - SPMD



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

8

ภาพ 2-3

Draft

ค่า Parameter ค่าหนึ่งที่พูดถึงกันมากคือ Peak floating point performance (พีค โฟลติ้ง พอตติ้ง เพอฟอร์แมนน์) ของระบบคอมพิวเตอร์ซึ่งหมายถึงประสิทธิภาพสูงสุดในการคำนวณแบบ floating point ทางทฤษฎีซึ่งไม่สามารถสูงได้ขนาดนั้นเชิงปฏิบัติ

ก่อนอื่นเรามาทบทวนกันก่อนว่า floating point นั้นหมายถึงอะไร floating point คือการประมาณค่าของเลข Real Number บนระบบคอมพิวเตอร์ ซึ่งต่างจากค่าของตัวเลขแบบ Integer Number ซึ่งไม่ต้องประมาณ

floating point operations นั้นสำคัญมากสำหรับการคำนวณทางวิทยาศาสตร์และวิศวกรรมศาสตร์ วงจรสำหรับการคำนวณแบบนี้สร้างยากและมีราคาแพงในอดีต แต่ไม่เป็นเช่นนั้นแล้ว ในปัจจุบัน

สำหรับ Parameters ที่แสดง Organization ของระบบ เช่น จำนวน node และจำนวน processors ต่อ node นั้นแสดงให้เห็นโครงสร้างที่มีหลายระดับของระบบ ที่ทำให้มีการทำงานแบบ concurrency มากขึ้น อันนำไปสู่ Throughput ที่สูงขึ้นและในบางกรณีอาจทำให้เราได้ Response Time ที่ลดลง

ในเรื่องของ Topology คือการเชื่อมต่อชิ้นส่วนต่างของระบบ ซึ่งลักษณะการเชื่อมต่อแบบต่างก็มีผลกระทบกับค่า Bandwidth หรือปริมาณข้อมูลที่เราสามารถส่งผ่าน Network ที่เชื่อมชิ้นส่วนเหล่านั้น ณ. เวลาใดเวลาหนึ่ง สุดท้าย วิธีการควบคุมการทำงานของส่วนประกอบ

ต่างๆของระบบก็เป็นอีก Parameter หนึ่งที่มีผลต่อประสิทธิภาพ

ระบบคอมพิวเตอร์อาจควบคุมการทำงานของส่วนประกอบต่างๆร่วมกัน หรือกำหนดให้แต่ละส่วนประกอบมีการควบคุมเป็นของตนเองและแยกจากกัน



Performance Factors: Parallelism

- Fully independent processing elements operating concurrently on separate tasks
 - Coarse grained
 - Communicating Sequential Processes (CSP),
 - Single Program Multiple Data stream (SPMD)
- Instruction Level Parallelism (ILP)
 - Fine grained
 - Single instruction performs multiple operations
- Pipelining
 - Fine grained
 - Overlapping sequential operations in execution pipeline
 - Vector pipelines
- SIMD operations
 - Fine / Medium grained
 - Single Instruction stream, Multiple Data stream
 - ALU arrays
- Overlapping of computation and communication
 - Fine / Medium grained
 - Asynchronous
 - Prefetching
- Multithreading
 - Medium grained
 - Separate instruction streams serve single processor



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

9

ภาพ 2-4

ผมอยากรู้ให้คุณตั้งใจศึกษาสไลด์นี้ เพราะเราจะพูดถึงวิธีการต่างๆที่ทำให้เกิดการประมวลแบบขนาน หรือ Parallel Processing ขึ้น เรื่องนี้มีพื้นฐานมาจากเรื่องของ Granularity ที่ผมได้พูดถึงก่อนหน้านี้ ต่อไปคุณจะเห็นภาพของมันมากขึ้นเมื่อเราทำการทดลองและวัดค่าจริงๆ แต่ตอนนี้ผมอยากรู้ให้คุณจินตนาการอย่างง่ายๆว่า Granularity คือขนาดของข้อมูลอินพุทของการประมวลผลขนาดหนึ่ง ถ้าขนาดใหญ่ก็ต้องประมวลผลมาก ขนาดเล็กก็ประมวลผลน้อย จากที่เคยกล่าวมาแล้วว่าคุณอาจมองว่าระบบของคุณเป็นระบบมหาศาลหรือ Global system ซึ่งประกอบไปด้วยระบบย่อยหรือ local systems หลายระบบ Granularity ก็หมายถึงขนาดของการประมวลผลที่ระบบ local และระบบทำโดยปราศจากการปฏิสัมพันธ์หรือประสานงานกับระบบ local อื่นๆนั่นเอง

ระบบเหล่านี้มีโครงสร้างที่แตกต่างกันไปเพื่อสนับสนุนการทำงานแบบพร้อมเพียงหรือ Concurrency ส่วนใหญ่แล้วจะเป็นแบบ Hierarchy ซึ่งเราจะได้ศึกษามันในรายละเอียดต่อไป ถ้าจะว่าไปแล้วโครงสร้างแบบที่ง่ายที่สุดก็คือการเอาคอมพิวเตอร์จำนวนหนึ่งมาไว้ในห้อง

เดียวกันแล้วให้แต่ละเครื่องทำงานของมันไป การทำงานแบบนี้เป็นการทำงานแบบ Parallel และแบบ Concurrent ด้วย เพราะโปรแกรมที่ทำงานบนแต่ละเครื่องไม่มีการปฏิสัมพันธ์กัน ขนาดของการทำงานบนแต่ละเครื่องนั้นเราเรียกว่าเป็นแบบ Coarse Grain คือเป็นหน่วยของการทำงานที่มีขนาดใหญ่ประกอบไปด้วยคำสั่งจำนวนมาก การทำงานแบบนี้จะได้ค่า Throughput ที่สูงอันเกิดจากการนำค่า Throughput ของแต่ละเครื่องมารวมกัน

คุณอาจเพิ่มการติดต่อสื่อสารหรือเพิ่มกลไกการประสานงานระหว่างการทำงานบนระบบคอมพิวเตอร์เหล่านั้นในลักษณะของ Communicating Sequential Processes และ Single Program Multiple Data Stream ซึ่งการทำงานบนแต่ละเครื่องก็ยังเป็นแบบ Coarse grain ออยู่ ถึงแม้ขนาดของ granularity จะเล็กลง แต่อย่างไรก็ตามการปฏิสัมพันธ์ก็ทำให้เราสามารถนำเครื่องคอมพิวเตอร์เหล่านั้นมาร่วมกันแก้ปัญหาอย่างโดยย่างหนักได้

ในขณะเดียวกันก็มีการทำงานแบบที่เป็น Instruction Level Parallelism หรือ ILP ซึ่งเป็นลักษณะการทำงานแบบ Fine grain ที่การทำงาน 1 instruction ทำให้เกิด operations มากกว่า 1 operations ทำงานพร้อมๆ กัน

ใน Microprocessor สมัยใหม่ย่างเช่นในเครื่อง Laptop ของคุณ มักจะมี ALU หลายอัน อาจเป็น Floating point unit 2 units และ Integer unit 2 units รวมเป็น 4 units ซึ่งทั้งหมดสามารถทำงานได้พร้อมๆ กันด้วย instruction เดียว ถึงแม้ว่า Applications ทั่วๆ ไปจะไม่สามารถใช้ประโยชน์จาก ILP ได้เต็มที่แต่บาง Applications ก็สามารถทำเช่นนี้ได้เป็นอย่างดี

โครงสร้างอีกรูปแบบหนึ่งที่เป็นแบบ Fine grain คือโครงสร้างแบบ Pipelining ซึ่งเป็นการแบ่งเส้นทางการทำงานหนึ่งออกเป็นเส้นทางย่อยหลายช่วง Pipelining เป็นวิธีการอย่างหนึ่งที่เราใช้กับกิจกรรมบางอย่างในชีวิตประจำวัน เราจะพูดถึงรายละเอียดและ Performance Model ของมันต่อไป

โครงสร้างแบบ Fine grain แบบสุดท้ายที่เราจะพูดถึงก็คือ Single Instruction Stream Multiple Data Stream หรือ SIMD ซึ่งเป็นระบบที่ Instruction เดียวกันทำงานบน Computing Units หลายช่วงพร้อมๆ กัน

ในการทำงานอีกระดับหนึ่งแบบ Medium grain ที่ประกอบไปด้วยคำสั่งหลายคำสั่งแต่จำนวนไม่มากเหมือนกับแบบ Coarse grain เราอาจจัดให้มีการทำงานแบบขนานในลักษณะที่ชุดคำสั่งสำหรับ computation และ communication ถูกจัดให้เกิดขึ้นพร้อมๆ กันในลักษณะควบคู่กัน หรือ overlap การทำงานลักษณะนี้จะคล้ายๆ กับแบบ Pipelining มีระดับของ grain ที่ใหญ่กว่า

โครงสร้างแบบสุดท้ายคือแบบ Multithreading ที่ประกอบไปด้วย Thread หลายๆ Thread ทำงานไปพร้อมๆ กัน โดยที่แต่ละ Thread ทำงานตามชุดคำสั่งซึ่งอาจมีขนาดเป็นแบบ Medium หรือ Coarse grain ก็ได้

เราได้พูดถึงโครงสร้างหลายๆ โครงสร้างในสไลด์เดียวโดยที่จุดประสงค์ของโครงสร้างต่างๆ นั้นก็คือความพยายามที่จะใช้การทำงานแบบขนาดเพื่อเพิ่ม Throughput และถ้าเป็นไปได้ลด Response Time



Performance Degradation

- Overhead

- Critical-path work for management of concurrent tasks and parallel resources not required for sequential execution
- e.g. : Synchronization and scheduling

- Latency

- Time required for response of access to remote data or services
- e.g. : Local memory access, Network communication

- Contention

- Idle cycles waiting for access to shared resources
- e.g. : Memory bank conflicts, shared network channels

- Starvation

- Insufficient parallelism
- Inadequate load balancing
- e.g. : Amdahl's law



Department of Computer Science

Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer Architecture, Spring 2009

10



ภาพ 2-5

ในสไลด์นี้ Overhead คือเวลาที่คุณไม่ต้องการเสียแต่만ักจำเป็นต้องเสียเพื่อจัดการการทำงานแบบ Parallel ยกตัวอย่างเช่นเวลาที่ใช้ในการ Synchronization และ Scheduling การประมวลผล

Latency คือเวลาที่ใช้ในการเข้าถึงข้อมูลหรือบริการซึ่งอยู่ที่ใดที่หนึ่งซึ่งอาจอยู่ไกลหรือใกลอกันไป

สำหรับ Contention นั้นคือเวลาที่คุณต้องเสียไปเพื่อรอที่จะใช้งาน resources ยกตัวอย่างเมื่อ processor หลายๆ processor ต้องการเข้าถึง memory bank หนึ่งในเวลาเดียวกัน แต่มันสามารถเข้าถึงได้ทีละ process เท่านั้น

สุดท้ายคือ Starvation ซึ่งเกิดขึ้นเมื่อมีงานเพียงพอให้ processor หลายๆ processor ทำซึ่งอาจเกิดจากที่ไม่มีการกระจายการทำงานแบบขนาดไปยัง processor เหล่านั้นอย่างเหมาะสม สมทำให้บาง processor ไม่มีงานทำ

ต่อไปเราจะพูดถึงเรื่องของกฎของ Amdahl ซึ่งคร่าวๆแล้วก็หมายถึงการประมวลผลของ

โปรแกรมๆหนึ่งหรือส่วนหนึ่งของโปรแกรมนั้นที่ไม่สามารถทำให้เป็นการทำงานแบบ Parallel ได้ หรือการประมวลผลดังกล่าวอาจถูกทำให้เป็นแบบ Parallel ได้แต่เราไม่ได้กระจายมันไปยัง processor ต่างๆหรือทำ load balancing ไม่ดีพอ

2.3 สถาปัตยกรรมคอมพิวเตอร์

ก่อนที่เราจะพูดถึง Architecture แบบใดแบบหนึ่งเราจะทำความเข้าใจถึงความหมายของมันก่อน สมมุติว่าคุณจะสร้างบ้านคุณอาจใช้ CAD โปรแกรมออกแบบบ้านของคุณซึ่งคุณอาจเอห้องหลายห้องมารวมกันให้เป็นบ้านในรูปแบบใดแบบหนึ่ง และนั่นก็คือโครงสร้างของบ้านของคุณนั่นเอง เมื่อเปรียบเทียบกับคอมพิวเตอร์แล้วนั่นก็เหมือนกับการที่คุณเอาส่วนประกอบต่างๆ ของมันมาจัดวางเป็นโครงสร้างรูปแบบใดรูปแบบหนึ่ง



Computer Architecture

- **Structure**
 - Functional elements
 - Organization and balance
 - Interconnect and Data flow paths
- **Semantics**
 - Meaning of the logical constructs
 - Primitive data types
 - Manifest as Instruction Set Architecture abstract layer
- **Mechanisms**
 - Primitive functions that are usually implemented in hardware or sometimes firmware
 - Determines preferred actions and sequences
 - Enables efficiency and scalability
- **Policy**
 - Actions and decisions outside the ISA or name space



ภาพ 2-6

จากสไลด์ Semantic คือ “ความหมาย” ของโครงสร้างหรือส่วนประกอบของโครงสร้าง นั่นๆ จากตัวอย่างโครงสร้างของบ้านที่เราได้พูดถึง คุณคงไม่ทำอาหารในห้องนอน และไม่นอนในห้องครัว เพราะห้องแต่ละห้องนั้นมี “ความหมาย” และหน้าที่เฉพาะของมัน ห้องนอน ห้องครัว ห้องน้ำ เล่น ทางเดิน ห้องน้ำ ล้วนแต่มีหน้าที่ของมันทั้งสิ้น

ในการที่จะทำให้ห้องเหล่านั้นทำงานได้ตามความหมายของมันนั้น แต่ละห้องจำเป็นต้องมี Mechanism หรือกลไกการทำงานของมัน ยกตัวอย่างเช่น ในห้องนอนต้องมีเตียง ในครัวต้องมีเตา กลไกเหล่านี้ทำให้ห้องแต่ละห้องทำงานได้ตามความหมายของมันได้ ซึ่งทุกๆห้องทำงานร่วมกันเพื่อสนับสนุนโครงสร้างรวมของทั้งระบบ

สุดท้ายคือเรื่องของ Policy หรือนโยบายที่แตกต่างกันในการใช้งานและการควบคุมดูแล การใช้งานระบบ ยกตัวอย่างเช่นในบ้านของผม ผมมีนโยบายว่าจะนอนในห้องนอนเวลากลางคืน และมีนโยบายว่าจำนวนคนที่จะทานข้าวในห้องทานข้าวมีได้จำนวนหนึ่งเท่านั้น เช่นเดียวกันกับระบบคอมพิวเตอร์

สรุป ก็คือส่วนประกอบ hardware ของคอมพิวเตอร์แต่ละชิ้นทำงานที่ของมันได้ด้วย กลไกของส่วนประกอบนั้นๆ แต่การที่เราจะใช้งาน hardware เหล่านั้นอย่างไรก็ชี้แจงอยู่กับการตัดสินใจของ hardware หรือ software หรือโปรแกรมเมอร์ที่ควบคุมการใช้งานระบบ

Draft



Structure

- Functional elements
 - The form of functional elements made up of more primitive logical modules
 - e.g. vector arithmetic unit comprising a pipeline of simple stages
- Organization and balance
 - Number of major elements of different types
 - Hierarchy of collections of elements
- Data flow
 - Interconnection of functional, state, and communication elements
 - Control of dataflow paths determines actions of processor and system



Department of Computer Science CSC 7600 Lecture 2 : Parallel Computer Architecture, Spring 2009

13

Draft
ภาพ 2-7

Structure

เราได้พูดถึงโครงสร้างชั้นประกอบไปด้วยชิ้นส่วนต่างๆ ของระบบคอมพิวเตอร์รวมถึงการเชื่อมต่อ กันและความสัมพันธ์ของชิ้นส่วนต่างๆ เหล่านั้น ใน การออกแบบโครงสร้างของระบบนั้น ชิ้นส่วน ต่างๆ จะต้องได้รับการจัดวางอย่างสมดุล ยกตัวอย่าง เช่น ใน การออกแบบ เราต้องพิจารณาว่า จะ ต้องใช้ memory bank กี่อัน สำหรับ processor core และจะต้องมีกี่ processors ใน 1 node เป็นต้น รวมทั้งต้องพิจารณาการจัดวางส่วนประกอบเหล่านี้อย่างสมดุลด้วย

Semantics

Semantics คือ เรื่องของความหมายของส่วนประกอบแต่ละชิ้นของระบบ ยกตัวอย่าง เช่น

floating point number นั้นจริงๆแล้วเรามันก็คือชุดของข้อมูลแบบไบนารีหรือบิต ซึ่งก็คือกระแสไฟฟ้าชุดหนึ่งนั่นเอง เมื่อเราชุดของข้อมูลนี้มาทำการบวก อุปกรณ์ hardware ที่ทำหน้าที่ดังกล่าวก็จะนำชุดของบิตที่เป็นตัวแทนของค่าของตัวเลขที่เกี่ยวข้องมาทำการอย่างโดยประมาณ แล้วให้ผลลัพธ์เป็นชุดของบิตชุดใหม่ออกมานะ การทำงานของ hardware นี้ก็คือการแปลความหมายของชุดของบิตที่มันนำมาปฏิบัติการว่าเป็นค่าของเลข floating point และทำงานตาม semantics หรือหน้าที่ของมัน

Draft



Semantics

- Meaning of the logical constructs
 - Basic operations that can be performed on data
- Primitive data types
 - Defines actions that can be performed on binary strings
 - Cache coherence mechanisms
- Instruction Set Architecture
 - Defined set of actions that can be performed on data object on which they can be applied
- Parallel control constructs
 - Hardware implemented : vector operations,
 - Software implemented : MPI libraries



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

14

ภาพ 2-8

ในการสั่งงาน hardware ให้ทำงานตามหน้าที่ของมันนั้นเกิดจากการแปลความหมายของชุดของบิตที่เป็นตัวแทนของคำสั่งและค่าของตัวเลขที่เกี่ยวข้อง เราเรียกโครงสร้างสำหรับการปฏิบัติการแบบนี้ว่าสถาปัตยกรรมชุดคำสั่งหรือ Instruction Set Architecture

Instruction Set Architecture คือสถาปัตยกรรมที่รับชุดคำสั่งที่เป็นชุดของบิตมาแปลความหมายและปฏิบัติการในระดับพื้นฐานของเครื่องหรือ Machine-level operations การแปลความหมายนั้นไม่ได้เกิดขึ้นในระดับพื้นฐานเท่านั้น แต่เกิดขึ้นในระดับที่สูงขึ้นด้วย เช่นการที่เราใช้ compiler แปลชุดคำสั่งในรูปแบบภาษาโปรแกรมให้เป็นชุดคำสั่งในระดับ Machine-level เป็นต้น นอกจากนั้นเรายังมีระบบที่ประมวลผลชุดคำสั่งในระดับ Machine level ที่ทำให้เกิดการทำงานแบบ parallel ในระดับ fine grain เช่น pipelining และ Instruction Level Parallelism ด้วย

Mechanism

Mechanism หมายถึงกลไกการทำงานของส่วนประมวลพื้นฐานของระบบคอมพิวเตอร์ซึ่งส่วนใหญ่แล้วจะเป็น hardware หรือ firmware เพื่อลด overhead ในการประมวลผล กลไกเหล่านี้ถูกสร้างขึ้นมาเพื่อความคุ้มการทำงานที่เกิดขึ้นบ่อยและต้องการประมวลผลอย่างรวดเร็ว ส่วนใหญ่แล้วการทำงานของกลไกเหล่านี้เกิดขึ้นโดยที่ผู้ใช้ไม่รู้ ยกตัวอย่างเช่นการแปลง Address ของ Translation Look Aside buffer และการทำงานของกลไกที่นำข้อมูลเข้าหรือออกจาก

cache เป็นต้น กลไกของระบบอาจเป็นการทำงานของ Software ก็ได้ยกตัวอย่างเช่นการส่งข้อมูลระหว่าง Compute Nodes เป็นต้น



Mechanisms

- Primitive functions that are usually implemented in hardware or sometimes firmware
 - Lower level than instruction set operations
 - Multiple such mechanisms contribute to execution of operation
- Determines preferred actions and sequences
 - Usually time effective primitives
 - Usually widely used by many instructions
- Enables efficiency and scalability
 - Establishes basic performance properties of machine
- Examples
 - Basic arithmetic and logic unit functions
 - Thread context switching
 - TLB (Translation Lookaside Buffer) address translation
 - Cache line replacement



Policy

ภาพ 2-9

Draft

นโยบาย หรือ Policy ไม่ใช่การสั่งงานให้คอมพิวเตอร์ทำงานโดยใช้ชุดคำสั่งแบบ Machine-level instruction set แต่เป็นการออกคำสั่งในอิกรูปแบบหนึ่งสำหรับการประมวลผล Applications ของระบบคอมพิวเตอร์ ยกตัวอย่างเช่น ในระดับ low-level ได้แก่การกำหนดนโยบายในการทำงานของระบบ cache หรือนโยบายในการทำ Branch prediction ซึ่งอยู่นอกเหนือจากการสั่งงานโดยใช้ instruction set ตัวอย่างของการกำหนดนโยบายในระดับ High-level ก็ได้แก่การกำหนดนโยบายในการ Schedule jobs หรือการแบ่งข้อมูลที่ใช้ในการประมวลผลให้กับ Processors ต่างๆ ซึ่งได้รับการกำหนดโดยผู้ใช้

Draft



Policy

- Hardware architecture policies
 - Not all machine decisions are visible to the instructions of the system
 - Not all machine choices are available to the name space of the operands
 - Examples
 - Cache structure, size, and speed
 - Cache replacement policies
 - Order of operation execution
 - Branch prediction
 - Allocation of shared resources
 - Network routers
- Software system management policies
 - Scheduling,
 - Data allocation : partitioning of a problem



Department of Computer Science

CSC 7600 Lecture 2 : Parallel Computer
Louisiana State University

16

ภาพ 2-10
Draft

2.4 ประเด็นเกี่ยวกับประสิทธิภาพของ Parallel Structures

เราจะพูดถึงโครงสร้างสามแบบใหญ่ๆ ซึ่งอาจนำมาใช้ร่วมกันได้ในการออกแบบระบบคอมพิวเตอร์ แบบแรกคือโครงสร้างระบบแบบ Pipelining ซึ่งเป็นระบบที่ทำการประมวลผลในระดับต่ำที่สุด แบบที่สองคือโครงสร้างแบบ Instruction Level Parallelism ที่ทำการประมวลผลบนหลายๆ ALU ได้พร้อมๆ กัน และแบบสุดท้ายคือการประมวลผลโดยใช้ processor หลายๆ processor ประมวลผลไปพร้อมๆ กันและมีการประสานงานหรือติดต่อสื่อสารกันเป็นครั้งคราว ซึ่งเราจะได้พูดถึงรายละเอียดของโครงสร้างเหล่านี้ต่อไป



Parallel Structures

- Pipelining
 - Vector processing
 - Execution pipeline
- Multiple Arithmetic Units
 - Instruction level parallelism
 - Systolic arrays
- Multiple processors
 - MIMD: Separate control
 - SIMD: Single controller
 - Multicore
 - Accelerators



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

18

Draft
ภาพ 2-11

ในแง่ของประสิทธิภาพนั้น การทำงานแบบ pipeline ทำให้ Throughput ของระบบสูงขึ้น และไม่ทำให้ Response Time ลดลงแต่อย่างใด ในกรณีของ Instruction Level Parallelism การใช้ ALU หลายๆ ALU ก็ทำให้ค่า Throughput เพิ่มขึ้นเช่นกันและใน Application บางอย่าง Response Time ก็อาจลดลงด้วย

สำหรับการประมวลผลโดยใช้หลายๆ Processors นั้นความสามารถเพิ่ม Throughput ได้แต่ยากที่จะลด Response Time เพราะ ในระหว่างประมวลผลจะมี Overhead ที่เกิดจากการประสานงานส่วนประกอบต่างๆของระบบซึ่งจะส่งผลให้ Execution Time เพิ่มมากขึ้น ด้วยเหตุนี้ Granularity จึงมีความสำคัญกับการประมวลผลแบบนี้ เพราะมันคือขนาดของการประมวลผลที่ส่วนประกอบใดๆทำโดยไม่มีการปฏิสัมพันธ์กับองค์ประกอบอื่นของระบบ ดังนั้นเราจำเป็นต้องจัดให้ Granularity ของการประมวลผลมีขนาดใหญ่เพียงพอที่จะทำให้ Overhead ของระบบโดยรวมอยู่ในระดับที่น้อยมากเมื่อเทียบกับการประมวลผลที่ได้

Draft



Performance Issues

- Pipelining
 - Pipelining increases throughput : More operations per unit time
 - Pipelining increases latency time : Operation on single operand pair can take longer than non-pipelined functional unit
- Multiple ALUs
 - Increases peak performance
 - Requires application instruction level parallelism
 - Average usually significantly lower than peak
- Multiple processors require overhead operations
 - Synchronization
 - Communications
 - Cache Coherence



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

19

ภาพ 2-12
Draft

2.4.1 Scalability

slide นี้ เป็น slide ที่สำคัญ และคุณจะเห็นมันบ่อยด้วย ซึ่งหลักการพื้นฐานของมันคือหลักการเกี่ยวกับ response time และ throughput นั้นเอง เราจะพิจารณา Scalability สองชนิดได้แก่ Strict scaling และ Weak scaling

Scalability นั้นเกิดจากໄ奥地ียง่ายๆว่า เมื่อเราเพิ่ม Resource หรือ Hardware สำหรับ การประมวลผล เราคาดว่าจะเห็นประสิทธิภาพในการทำงานของ Applications หรือ Workload ที่เราสนใจในระดับที่สูงขึ้นด้วย ซึ่งประสิทธิภาพที่สูงขึ้นนั้นวัดได้จาก Throughput หรือจำนวนงานที่ทำเสร็จต่อหนึ่งหน่วยเวลา และจาก Response Time หรือเวลาที่ใช้ในการทำงานนั้นเอง

จากที่เคยอธิบายไปแล้วว่า เราจะต้องแบ่งการประมวลผลไปยัง Computing resources หลายๆ resource ในการทำงานแบบ Parallel ซึ่งทำให้เกิด Overhead ในการประสานงาน การประมวลผลบน Resource เหล่านั้นตามมา

ถ้าเราต้องการเพิ่มประสิทธิภาพโดยการลด response time เป็นหลัก เราต้องกำหนดให้ขนาดของการประมวลผลโดยรวมนั้นคงที่ในขณะที่จำนวนของ Computing resource เพิ่มมากขึ้น ดังนั้นขนาดของ granularity ของการประมวลผลของแต่ละ resource ก็จะลดลงและส่งผล

ให้การประมวลใน granularity ดังกล่าวเสร็จเร็วขึ้น แต่การเปลี่ยนแปลงของค่า Overhead เมื่อ resource เพิ่มขึ้นมิได้เป็นไปในรูปแบบเดียวกันกับการเปลี่ยนแปลงขนาด Granularity เมื่อจำนวน resource มากขึ้น Overhead ในการประสานงานนั้นมักจะมีค่าคงที่หรือเพิ่มขึ้น ดังนั้น ในที่สุดแล้วเมื่อเราเพิ่มจำนวน resource เป็นจำนวนมากๆ เวลาที่ใช้ในการทำงานส่วนใหญ่ก็จะกลับเป็นเวลาที่สูญเสียไปกับ Overhead ทำให้เราได้ค่า Throughput ที่น้อยมากตามมา

Draft



Scalability

- The ability to deliver proportionally greater sustained performance through increased system resources
- Strong Scaling
 - Fixed size application problem
 - Application size remains constant with increase in system size
- Weak Scaling
 - Variable size application problem
 - Application size scales proportionally with system size
- Capability computing
 - in most pure form: strong scaling
 - Marketing claims tend toward this class
- Capacity computing
 - Throughput computing
 - Includes job-stream workloads
 - In most simple form: weak scaling
- Cooperative computing
 - Interacting and coordinating concurrent processes
 - Not a widely used term
 - Also: coordinated computing



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

21

ภาค 2-13

ในทางตรงกันข้าม ถ้าเราต้องการเพิ่มประสิทธิภาพโดยการเพิ่ม Throughput เป็นหลัก เราจะมองว่าเมื่อจำนวน computing resource เพิ่มมากขึ้น ขนาดของการประมวลผลโดยรวมก็ เพิ่มขึ้นมากด้วย โดยที่การเพิ่มขึ้นของทั้งสองอย่างนั้นต้องเป็นไปในทิศทางที่ทำให้สัดส่วนของ Overhead และขนาดของ Granularity บน resource เหล่านั้นคงเดิม ด้วยการเพิ่มประสิทธิภาพแบบนี้ เมื่อเพิ่มจำนวน resource เราจะได้ Throughput ที่สูงขึ้นตามมา แต่ในการนี้คุณจะไม่ได้ ค่า Response Time ที่ลดลง

ความแตกต่างของ Strict scaling และ Weak scaling นั้นคือ Strict scaling จะพิจารณา ขนาดของปัญหาที่คงที่ และคาดว่าเมื่อ Computing resource เพิ่มขึ้นแล้ว Response Time จะ ลดลง ในขณะที่ Weak scaling นั้นเป็นการเพิ่ม resource และในขณะเดียวกันเพิ่มขนาดของ ปัญหาเพื่อควบคุมสัดส่วนของ Overhead ต่อขนาดของการประมวลผลให้อยู่ในระดับที่เหมาะสม เพื่อเพิ่ม Throughput

โดยทั่วไปแล้วเราจะพูดถึงชนิดของการประมวลผลสองแบบคือ Capability computing และ Capacity computing

Capability computing นั้นใช้หลักการของ Strict scaling ในขณะที่ Capacity computing มองว่าเราสามารถเพิ่มจำนวนเครื่องคอมพิวเตอร์มาช่วยแก้ปัญหาที่มีอยุ่มาก many ซึ่ง

ปัญหาเหล่านี้อาจไม่ได้มีความเกี่ยวข้องกันเลยเพียงแต่มีปริมาณโดยรวมเป็นจำนวนมาก ซึ่งถ้าจะว่าไปแล้วนี่คือรูปแบบที่ง่ายที่สุดของ Weak scaling นั่นเอง เพราะมันແບບจะไม่มีการประสานงานระหว่างเครื่องเกิดขึ้น

ผมอยากระบุถึงชนิดของการประมวลอิกอย่างหนึ่งที่เรารายไม่ค่อยได้พูดถึงกันคือ Cooperative computing การประมวลผลชนิดนี้เป็นการเอาปัญหางานมาเด็กให้เกิดการประมวลผลอยู่ๆ หลายส่วนที่เกี่ยวข้องกัน เมื่อเราเพิ่มขนาดของปัญหาให้มีขนาดใหญ่ขึ้นเราก็เพิ่มจำนวน computing resource ขึ้นตามด้วย เราจะพูดถึง Cooperative Computing ต่อไปในวิชานี้

เวลาที่คุณซื้อคอมพิวเตอร์ ผู้ขายส่วนใหญ่จะพยายามขายระบบโดยเน้นที่ Capability นั่นคือเขาจะพยายามขายเครื่องที่มีคุณภาพและมีอุปกรณ์ใหม่ๆ ราคาแพงเพื่อให้คุณทำงานได้เร็วขึ้น ในทางตรงกันข้าม ผู้ขายต้องการขายระบบที่เน้น Capacity เขาจะโฆษณาว่าเครื่องที่ต้องการขายมีความสามารถประมวลผลต่อหนึ่งหน่วยเม็ดเงินที่คุณลงทุนมากกว่าเครื่องอื่นๆ

Draft



Performance Metrics

- Peak floating point operations per second (flops)
- Peak instructions per second (ips)
- Sustained throughput
 - flops, Mflops, Gflops, Tflops, Pflops
 - flops, Megaflops, Gigaflops, Teraflops, Petaflops
 - ips, Mips, ...
- Cycles per instruction
 - cpi
 - Alternatively: instructions per cycle, ipc
- Memory access latency
 - cycles per second
- Memory access bandwidth
 - bytes per second
 - or Gigabytes per second, GBps, GB/s
- Bi-section bandwidth
 - bytes per second



ภาพ 2-14

2.4.1 Performance Matrices

Draft

ใน High performance computing มี performance matrices หลายอันที่เราใช้ อย่างแรกที่ผมจะพูดถึงก็คือ flops หรือ Floating Point Operation per Second ผมขอเน้นนะครับว่า “s” ตัวสุดท้ายนั้นแทน Second

ถ้าคุณได้ยินว่าบริษัท Foo-Bar คือเครื่องระดับ Peta-flops นั้นหมายความว่าเครื่องคอมพิวเตอร์เครื่องนั้นทำงานด้วยความเร็ว Quadrillion operations ต่อหนึ่งวินาที หรือ พันล้านล้านคำสั่งต่อหนึ่งวินาที นั่นเอง สำหรับ Matrices อื่นๆก็ขอให้ศึกษาจากสไลด์นะครับ ขอให้สังเกตด้วยนะครับว่า cycle per instruction ก็คือบทกลับของ instructions per cycle นั่นเอง

2.5 Basic Uni-processor Architecture Element

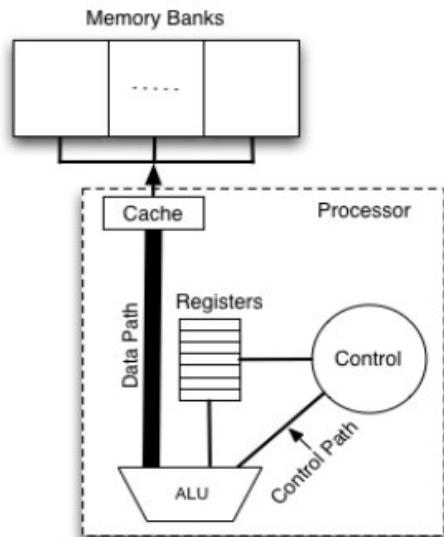
เราจะศึกษา Architecture กันเริ่มต้นจากรูปที่แสดงส่วนประกอบของเครื่องที่มี Processor เดียว และหลังจากนั้นเราจะไปศึกษาเครื่องแบบ Parallel และพิจารณาว่ามันนำส่วนประกอบเหล่านี้ไปใช้อย่างไร จากภาพคุณจะเห็นว่า processor นั้นมี register อยู่กลุ่มหนึ่งซึ่งเชื่อมต่อกับ ALU และมี cache ที่เราใช้เก็บข้อมูลที่มีการใช้งานบ่อยๆ ซึ่ง cache นี้ก็มี interface เชื่อมต่อไปยัง main memory นอกจากนั้นก็มี controller ซึ่งทำหน้าที่ควบคุมการทำงานให้เกิดขึ้นเป็นอันดับ

Draft



Basic Uni-processor Architecture elements

- Cache hierarchy
- Arithmetic Logic Units
- Register Sets
- Control
- Memory interface
- I/O Interface
- Execution pipeline



Department of Computer Science

Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

24

幻燈 2-15

2.6 Multiprocessor

Draft



Multiprocessor

- A general class of system
- Integrates multiple processors in to an interconnected ensemble
- MIMD: Multiple Instruction Stream Multiple Data Stream
- Different memory models
 - Distributed memory
 - Nodes support separate address spaces
 - Shared memory
 - Symmetric multiprocessor
 - UMA – uniform memory access
 - Cache coherent
 - Distributed shared memory
 - NUMA – non uniform memory access
 - Cache coherent
 - PGAS
 - Partitioned global address space
 - NUMA
 - Not cache coherence
 - Hybrid : Ensemble of distributed shared memory nodes
 - Massively Parallel Processor, MPP



Department of Computer Science

Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

25

ภาพ 2-16

Multiprocessor เป็น processor ที่เอาส่วนประกอบใน uni-processor architecture มา เชื่อมต่อเข้าด้วยกันในอิกรูปแบบหนึ่ง จริงๆแล้วคำว่า multiprocessor นั้นมีนานานแล้วและมี ความหมายหลายอย่างในปัจจุบัน ยกตัวอย่างเช่น Distributed Memory Multiprocessor ก็คือ multiprocessor ที่ไม่ได้แชร์ memory ซึ่ง processor เหล่านี้ติดต่อสื่อสารกันผ่าน I/O

ในขณะที่ Shared memory multiprocessor คือระบบที่ processor ทุกๆตัวแชร์ Main memory อันเดียวกัน ดังนั้นเวลาที่ใช้ในการเข้าถึง memory จากแต่ละ processor นั้นจะเท่ากัน แต่ในระบบที่ใหญ่ขึ้นนั้นเราจะคิดถึงระบบที่เรียกว่า Distributed Shared Memory

Multiprocessor ซึ่ง processor ทุกตัวจะมี local memory ของมัน และก็สามารถเข้าถึง local memory ของ processor อื่นๆได้ด้วย ดังนั้นเวลาในการเข้าถึง memory ในแต่ละครั้งอาจไม่เท่า กันได้ ขึ้นอยู่ว่า memory ที่เข้าถึงนั้นอยู่ที่ไหน

เมื่อพูดถึงการเข้าถึง memory และเราจะแบ่งออกเป็นสองแบบคือ UMA หรือ Uniform Memory Access และ NUMA หรือ Non-Uniform Memory Access การเข้าถึงที่แตกต่างนี้ใช้ ระบุชนิดของคอมพิวเตอร์ได้ด้วย

อีกระบบที่น่าสนใจคือ PGAS หรือ Partitioned Global Address Space ซึ่ง Processor แต่ละ Processor จะมี Memory ที่มันเข้าถึงได้ Processor แต่ละตัวก็จะมี cache ของมันแต่จะไม่มีระบบจัดการระบบจัดการข้อมูลใน cache แบบอัตโนมัติ Application programs จะต้องจัดการการเคลื่อนย้ายข้อมูลระหว่าง memory และ cache เอง โดยมีจุด ประสงค์ที่จะลด Overhead และเพิ่มประสิทธิภาพในการทำงาน

ในความเป็นจริงแล้วสถาปัตยกรรมในปัจจุบันก็เกิดมาจากการผสมผสานของหลักการเหล่านี้กันเอง

2.7 Massively Parallel Processor



Massively Parallel Processor

- MPP
- General class of large scale multiprocessor
- Represents largest systems
 - IBM BG/L
 - Cray XT3
- Distinguished by memory strategy
 - Distributed memory
 - Distributed shared memory
 - Cache coherent
 - Partitioned global address space
- Custom interconnect network
- Potentially heterogeneous
 - May incorporate accelerator to boost peak performance



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

26

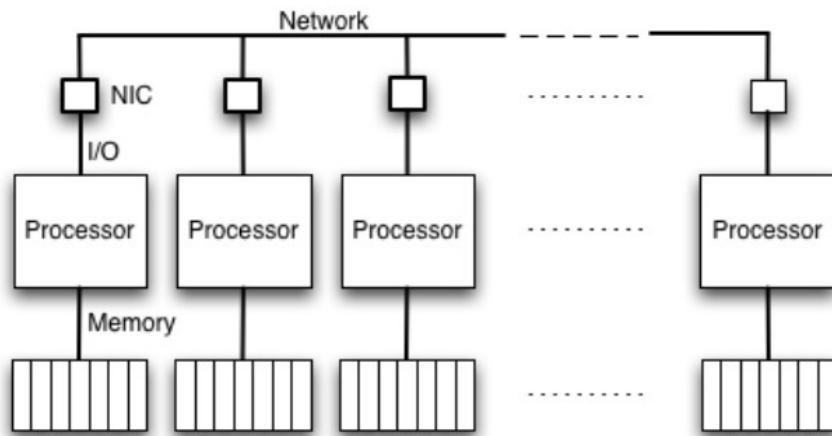
ภาพ 2-17

Draft

เราจะมาดูเครื่องอีกแบบหนึ่งคือระบบ Massively Parallel Processor หรือ MPP ซึ่งเป็นรูปแบบของเครื่องที่ใหญ่ที่สุด เครื่องแบบนี้จะเป็นแบบ Distributed Memory คือ Memory ของ node หนึ่งจะแยกจากอีก Node หนึ่งและไม่สามารถเข้าถึงกันได้ ถ้า process บน node หนึ่งต้องการข้อมูลจาก memory ของ remote node ไดๆ process นั้นต้องขอข้อมูลผ่าน software ของ remote node นั้น ระบบ MPP บางระบบมีสถาปัตยกรรมแบบ Distributed Shared Memory ยกตัวอย่างเช่นเครื่องของบริษัท SGI ซึ่งมีระบบจัดการ cache coherency และการเข้าถึงข้อมูลที่ใช้เวลาแตกต่างกัน ระบบๆนี้เป็นระบบที่ค่อนข้างแพงและซับซ้อน นี่คือภาพของ diagram ซึ่งทำให้เข้าใจระบบ MPP ง่ายขึ้น คุณจะเห็นว่าแต่ละ processor มี local memory ของมัน และไม่สามารถเข้าถึง memory ของ processor อื่นๆได้ ถ้ามันต้องการเข้าถึงข้อมูลของ processor อื่นมันต้องร้องขอไปยัง processor นั้นผ่านระบบ Network



DM - MPP



CSC

Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

27

ภาพ 2-18

เครื่องคอมพิวเตอร์แบบ MPP ที่ใหญ่ที่สุดเครื่องหนึ่งในปัจจุบันก็คือเครื่อง IBM BlueGene ดังภาพ



IBM Blue Gene/L



CSC

Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

28

ภาพ 2-19

และมันก็เป็นหนึ่งใน Top 500 list ซึ่งเป็นการจัดอันดับเครื่องที่เร็วที่สุดในโลก 500 อันดับแรก ซึ่งได้รับการติด

ตั้งอยู่ที่ Lawrence Livermore National Laboratory และมีความเร็วอยู่ที่ 0.5 Peteflop

Draft

Draft



Historical Top-500 List



CSC

Department of Computer Science
Louisiana State University

csc 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

29

Draft

ภาพ Z-20

ภาพนี้แสดงสถาปัตยกรรมของเครื่อง BlueGene/L เครื่องในภาพมีความเร็วกว่า 300 teraflops แต่เครื่องที่เร็วที่สุดในปัจจุบันมีความเร็วถึง 600 teraflops ซึ่งมีจำนวน processors เพิ่มมากขึ้นกว่าในภาพถึง 40,000 processors เครื่องในภาพใช้พลังงาน 2.5 Megawatt ซึ่งถือว่าน้อยสำหรับเครื่องใหญ่ๆระดับนี้

เครื่อง MPP เครื่องแรกคือเครื่อง ASCI Red เมื่อประมาณ 10 ปีที่แล้ว เป็นเครื่องแรกที่ทำงานได้ถึงระดับ Teraflops วัดจาก Linpack benchmark



ASCI RED

Compute Nodes	4,536
Service Nodes	32
Disk I/O Nodes	32
System Nodes (Boot)	2
Network Nodes (Ethernet, ATM)	10
System Footprint	1,600 Square Feet
Number of Cabinets	85
System RAM	594 Mbytes
Topology	38x32x2
Node to Node bandwidth - Bi-directional	800 Mbytes/sec
Bi-directional - Cross section Bandwidth	51.6 Gbytes/sec
Total number of Pentium® Pro Processors	9,216
Processor to Memory Bandwidth	533 Mbytes/sec
Compute Node Peak Performance	400 MFLOPS
System Peak Performance	1.8 TFLOPS
RAID I/O Bandwidth (per subsystem)	1.0 Gbytes/sec
RAID Storage (per subsystem)	1 Tbyte



Department of Computer Science
Louisiana State University

31

Draft
ภาพ 2-21

นี่คือภาพของ Board หนึ่งของเครื่อง ASCI Red ซึ่งคุณจะเห็นว่ามี Module แยกกันอยู่ 2 Module ใน Board นี้เราใช้ processor ของบริษัท Intel

ในเครื่องจริงเราเอา Board เหล่านี้ไปใส่ใน Rack ซึ่งมีเป็นจำนวนมาก

ผนชอนะสมของพวgnicรับ ในปัจจุบันความสามารถของ Rack หนึ่ง Rack ของเครื่อง BlueGene/L ก็มีค่าเทียบมากกว่าประสิทธิภาพของเครื่อง ASCI Red ทั้งเครื่อง จริงๆแล้วเครื่อง BlueGene/L มีความเร็วมากกว่าเครื่อง ASCI Red ถึงกว่า 500 เท่า



ASCI RED : I/O Board



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

32

ภาพ 2-22

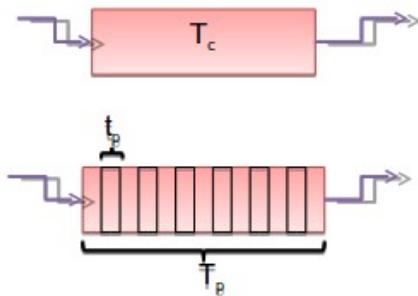
2.8 Pipeline Structures

ในสไลด์นี้เราจะศึกษา parallelism อีกแบบหนึ่งเรียกว่า Pipelining ซึ่งทำให้เรานำถึง Assembly Line แต่ผิดมัจจะนึกถึงเวลาที่เราเข้าคิวตักอาหารในโรงอาหารของโรงเรียนหรือใน food court ซึ่งทุกคนจะเข้าแถวและหยิบอาหารที่เข้าต้องการจากโต๊ะวางอาหารแบบต่างๆ และท้ายที่สุดเขาก็ไปจ่ายเงินกับ cashier ก่อนจะออกไปทานอาหารของเขารอไป ในขณะที่คนๆหนึ่งกำลังตักอาหารชนิดหนึ่งอยู่ อีกคนหนึ่งก็อาจหยิบอีกชนิดหนึ่ง และอีกคนหนึ่งซึ่งหยิบอาหารที่เขาต้องการเสร็จแล้วกำลังจ่ายเงินกับ cashier ในการทำงานแบบนี้เรารู้ว่าค่า Throughput ที่จะได้ก็คืออัตราเร็วของการจ่ายเงินที่ cashier นั่นเอง

Draft



Pipeline : Concepts



$$t_p \ll T_c$$

$$T_p = N \times t_p$$

$$T_c < T_p$$

$$Perf_c = \frac{1}{T_c}$$

$$Perf_p = \frac{1}{t_p}$$

Where :

- T_c is the Logic Latency
- T_p is the aggregated pipeline latency
- t_p is the latency for each pipelined step



CSC

Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer Architecture, Spring 2009

35

ภาพ 2-23

Draft

ในทางตรงกันข้ามแทนที่จะใช้วิธีเข้าແກตากาหาร คุณอาจใช้วิธีสั่งอาหารแล้วรอจนกว่าอาหารเสร็จ คนถัดไปจะสั่งอาหารของเข้าได้ก็ต่อเมื่อคุณได้รับอาหารของคุณแล้ว

ภาพด้านบนคือภาพ model ของการทำงานแบบที่สั่งอาหารและรอจนอาหารเสร็จ ซึ่งใช้เวลาในการรอทั้งหมดคือ T ในส่วน c แต่การทำงานแบบ pipeline เป็นแบบภาพที่ดำเนินมาคือเราจะแบ่งงานให้เป็นงานย่อยหลายๆ งาน ในความเป็นจริงแล้ว ถ้ามองจากมุมมองของคนที่จะทานอาหารแต่ละคน การสั่งอาหารและรอจนอาหารเสร็จอาจจะเร็วกว่าการทำงานแบบ pipeline เพราะแบบ pipeline นั้นต้องทำงานเป็นขั้นๆ ไปร่วบรัดตัดตอนไม่ได้ แต่อย่างไรก็ตาม การทำงานแบบ pipeline ก็จะทำให้เราได้ค่า Throughput ที่สูงกว่าแบบการสั่งอาหารเพราเราแบ่งการทำงานออกเป็นขั้นๆ หรือ stages ซึ่งในขณะที่คนๆ หนึ่งทำงานใน stage หนึ่ง คนอื่นๆ ก็สามารถทำงานใน stage อื่นๆ ได้พร้อมๆ กัน

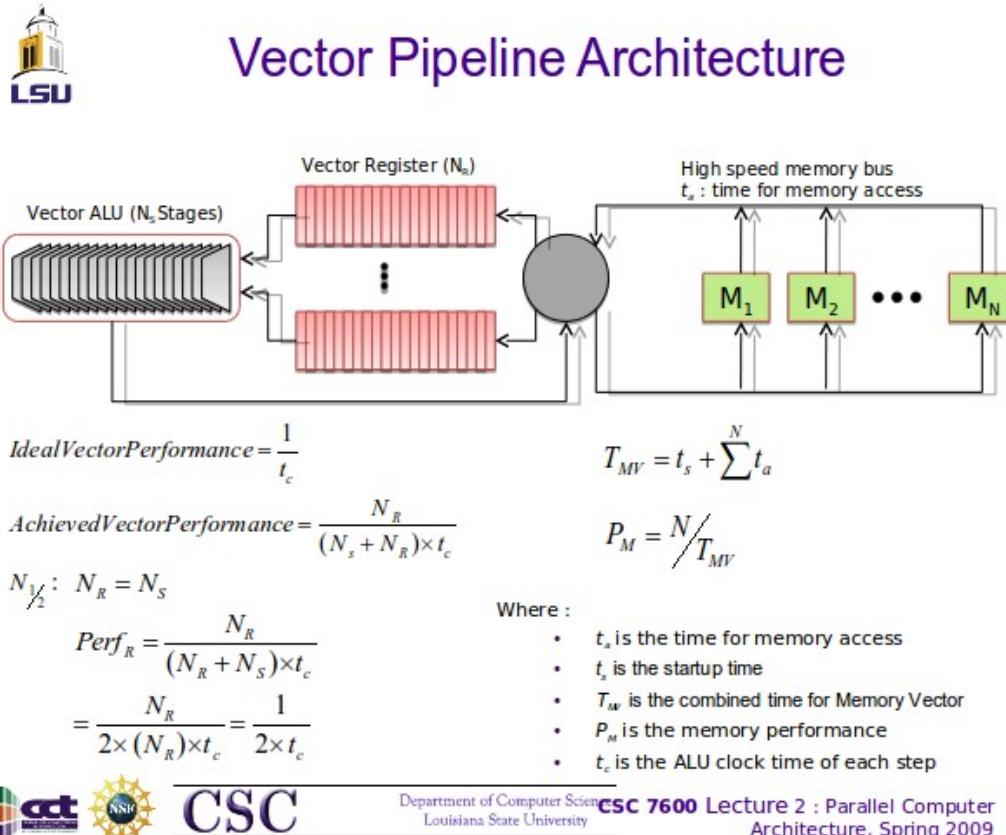
จากภาพข้างล่าง ถ้าให้เวลาในแต่ละ stage นั้นมีค่าเท่ากับ t เล็ก p อัตราการทำงานเสร็จจาก pipeline ก็จะเป็นหนึ่งงานทุกๆ ระยะเวลา t เล็ก p นั่นเอง

นอกจากนั้นถ้าคุณเอาเวลา t เล็ก p ในทุก stage มารวมกันคุณก็จะได้เวลา T ในส่วน p ซึ่งมากกว่าเวลา T ในส่วน c

ดังนั้นเราสรุปได้ว่าการทำงานแบบ pipeline นั้นเป็นการเพิ่ม Throughput ทำให้ประสิทธิภาพในการทำงานดีขึ้น แต่ว่า Response Time ก็จะมีค่ามากขึ้นเช่นกัน

2.9 Vector Processor

Pipelining เป็นเทคนิคที่ดีอันหนึ่งในการออกแบบ processor ซึ่งได้รับการนำไปใช้สม Parsons กับโครงสร้างสถาปัตยกรรมหลายแบบ รวมถึงโครงสร้างแบบ Vector Processor ที่เป็น processor ที่ประกอบไปด้วย execution unit หลายๆ unit ซึ่งจะทำงานไปพร้อมๆกันเพื่อ ประมวลผลข้อมูลแบบ vector



ภาพ 2-24

ในแต่ละ execution unit ส่วนใหญ่จะเป็น Pipeline execution unit ซึ่ง stage ต่างๆมัก จะเริ่มต้นด้วยการ fetch instruction ตามด้วยการร้องขอ registers และตามด้วยการ fetch data ตามด้วยการประมวลผลโดยใช้ ALU ซึ่งอาจใช้เวลาหลาย cycles หลังจากนั้นก็เป็นการส่งค่ากลับไปยัง registers และ commit ค่านั้น การแบ่งการทำงานเป็น stages เช่นนี้ทำให้เราสามารถ overlap การทำงานของแต่ละ stage ให้เกิดขึ้นพร้อมๆกันและได้ Throughput ที่สูงขึ้น ในที่สุด

ในภาพนี้เป็นตัวอย่างง่ายๆของ Vector Pipeline Architecture ขอให้สังเกตว่าภาพนี้ ประกอบไปด้วยสามส่วนใหญ่ๆ ทางด้านซ้ายคือ ALU ซึ่งเราตั้งสมมุติฐานไว้ว่าการทำงานของมัน ประกอบไปด้วย N_s Stages ภาพตรงกลางคือ Register bank หรือกลุ่มของ Registers ซึ่งเรา

ใช้เก็บค่าแบบ vector ซึ่งมีลักษณะเป็นชุดอาจประกอบไปด้วยข้อมูล 16 หรือ 64 ค่าก็ได้ ในภาพคุณจะเห็นว่าเรามี register bank สองชุดสำหรับการประมวลผลที่ต้องใช้ค่าสองค่าหรือ binary operations

ในภาพทางด้านขวาจะเป็นกลุ่มของ memory สามชุดที่เราใช้กลุ่มของ memory นั้นเนื่องจากการเข้าถึง memory unit เดียวันนี้ใช้เวลามากเนื่องจาก memory cycle time นั้นมักจะมีค่ามากกว่า memory access time ดังนั้นเราจะต้องใช้กลุ่มของ memory unit หลายๆ unit เพื่อที่เราจะได้สามารถ overlap การเข้าถึง memory unit หลายๆ unit เพื่อให้การเข้าถึงข้อมูลหลายๆข้อมูลทำได้เร็วขึ้น

ในการทำงานแบบ pipeline นั้นประกอบไปด้วยหลาย stages แต่การทำงานของแต่ละ stage อาจไม่เท่ากัน Stage ที่ช้าที่สุดจะเป็น bottle neck ของระบบ ดังนั้นอัตราความเร็วในการประมวลผลแบบ pipeline จะขึ้นอยู่กับเวลาในการทำงานของ stage ที่ช้าที่สุดนั้น

การทำงานนั้นจะเริ่มต้นจากการ load ค่าเข้าสู่ vector register ซึ่งมีขนาดจำกัดเช่น 16 หรือ 64 ค่าเป็นต้น สมมุติว่าเราจะทำการคำนวน dot product นั่นคือการจับคู่ค่าในอันดับเดียวกันใน vector สอง vectors มาคูณกัน และเอาผลคูณทั้งหมดมารวมกันเป็นค่าผลลัพธ์ ถ้าเราพิจารณาการคูณ vector และเราจะต้องสร้าง vector ใหม่จากการคูณ vector ตั้งตัน 2 vectors ซึ่งทำโดย vector unit หรือ vector ALU ในภาพ

เราจะหาว่าต้องใช้เวลาเท่าไรเพื่อให้การคูณเสร็จสิ้น ตอนแรกนั้น ทุกๆ stage ใน vector unit นั้นจะว่าง แต่ก่อนที่เราจะได้ vector ผลลัพธ์ชุดแรกของเราต้องทยอยเติมการประมวลผลในทุกๆ stage ให้เต็มซึ่งก็จะใช้เวลาเท่ากับเวลาที่ใช้ในการประมวลผลผ่านทุก stage ของ pipeline และเมื่อการทำงานมาถึง stage สุดท้ายมันก็จะเริ่มผลิตผลลัพธ์แบบ pipelining

เนื่องจากการทำงานของ register ก็มีหลาย stage เช่นกันดังนั้นก่อนที่เราจะเติม stage ของ pipeline ได้เติมเราต้องเติมค่าในทุก stage ของ vector register มาแล้วก่อนหน้า ดังนั้นหลังจากข้อมูลชุดแรกจาก vector register เริ่มผลิตผลลัพธ์ ซึ่งก็ใช้เวลาเท่ากับการทำงานในทุก stage และเวลาที่เหลือในการประมวลผล vector operations จนเสร็จสิ้นก็ขึ้นอยู่กับจำนวน stage ของ vector register หรือขนาดของ vector register นั้นเอง

จากภาพถ้าเราให้ N_s เป็นจำนวน stage ใน vector unit และ N_r คือจำนวน stage ของ vector register และ ประสิทธิภาพที่จะได้รับจากสถาปัตยกรรมนี้ก็คือ N_r หารด้วย ผลคูณของ t เล็ก c ซึ่งก็คือระยะเวลาในการทำงานแต่ละ stage กับผลรวมของ stage ทั้งหมด หรือ N_r บวก N_s นั้นเอง ถ้าจะพูดอีกอย่างหนึ่งก็คือ จำนวนค่าผลลัพธ์ที่ได้จากการประมวลผล ต่อเวลาที่ใช้ในการผลิตผลลัพธ์เหล่านั้น

มีรูปแบบของความสัมพันธ์ที่นำเสนใจอิกอย่างหนึ่งดังที่แสดงในภาพด้านขวาแบบ N หนึ่ง ส่วน สอง ซึ่งได้มาจากการที่เรากำหนดให้ N_r เท่ากับ N_s ทำให้เราได้ประสิทธิภาพใน

การทำงานเท่ากับครีงหนึ่งของ กรณีที่เป็น Ideal คือ 1 ผลลัพธ์ต่อเวลา t เล็ก c

ประสิทธิภาพในลักษณะนี้เป็นผลมาจากการที่เราต้องเติม pipeline ให้เต็มในช่วงเริ่มต้นของการประมวลผลค่าใน vector register การที่เราจำเป็นต้องเติม pipeline ให้เต็มนั้นเป็นตัวอย่างหนึ่งของ starvation ในระดับที่ fine grain มากรๆ



Cray 1

- First announced in 1975-6
- 80 MHz Clock rate
- Theoretical peak performance (160 MIPS), average performance 136 megaflops, vector optimized peak performance 150 megaflops
- 1-million 64 bit words of high speed memory
- Manufactured by Cray Research Inc.
- First Customer was National Center for Atmospheric Research (NCAR) for 8.86 million dollars.



src : <http://en.wikipedia.org/wiki/Cray-1>



Department of Computer Science
Louisiana State University

The Cray 1 System



Cray 1 logic boards



38

ภาพ 2-25

เครื่องในสไลเดอร์คือเครื่อง Cray 1 เป็น Supercomputer ที่สร้างในช่วงกลาง 1970 และใช้งานในปี 1976 โดยได้รับการติดตั้งที่ Los Alamos National Laboratory ชื่อเครื่องๆนี่เรียกตามชื่อผู้ประดิษฐ์คือ Seymour Cray ผู้ก่อตั้งบริษัท Cray Research Inc. เครื่องนี้มี Peak performance กว่า 100 Mflops ซึ่งน้อยกว่าประสิทธิภาพของเครื่อง Laptop ของคุณในปัจจุบัน

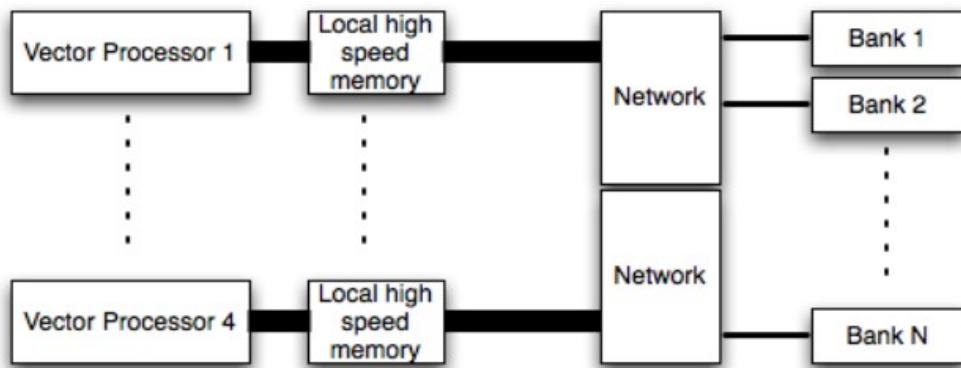
ปัญหอย่างหนึ่งของ Pipelining คือมันไม่ scale หรือเพิ่มประสิทธิภาพได้ยาก ดูนูกอกแบบและทำให้มันทำงานเร็วขึ้นได้เพียงแค่ระดับหนึ่งเท่านั้น แต่ Steve Chen จากบริษัท Cray Research ค้นพบว่าความสามารถเพิ่มประสิทธิภาพของมันโดยการเพิ่ม processor หลายๆ processor และออกแบบให้มันแชร์ memory กัน

นี่คือภาพง่ายๆของ PVP หรือ Parallel Vector Processor ที่มี 4 processors นี่คือโครงสร้างของเครื่อง Cray XMP ซึ่งออกแบบต่อจาก Cray 1 โดยการนำเอา Cray 1 สี่เครื่องมาต่อเข้าด้วยกัน หลังจากนั้นก็มีการสร้างเครื่องแบบนี้ต่ออีกดือเครื่อง Cray YMP เครื่อง C90

และ T90 คุณ Steve Chen เป็นผู้นำในการออกแบบเครื่องเหล่านี้ในขณะที่ Seymour Cray นั้นได้ไปออกแบบเครื่องอีกชุดหนึ่งคือ Cray 2 ซึ่งมีสถาปัตยกรรมที่แตกต่างกันจาก Cray 1 มาก



PVP (e.g. Cray – XMP)



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

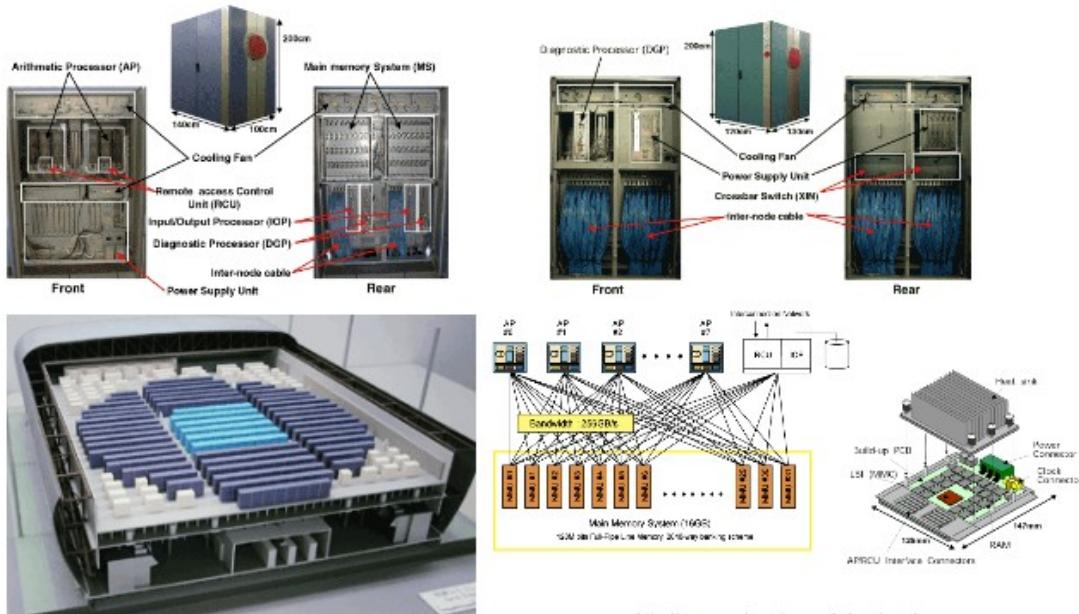
41

ภาพ 2-26
Draft

เครื่อง PVP ขนาดใหญ่มากในภาพนี้คือเครื่อง Earth Simulator ซึ่งเป็นเครื่องคอมพิวเตอร์ที่เร็วที่สุดเครื่องหนึ่งของโลก



Earth Simulator



src : <http://www.es.jamstec.go.jp/esc/eng/>



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer Architecture, Spring 2009

42

ภาพ 2-27
เครื่อง Earth Simulator มีความเร็วสูงสุดคือ 35 Teraflops วัดโดยใช้ Linpack benchmark ใช้ งบประมาณในการสร้าง 400 ล้านเหรียญ ซึ่งจริงๆแล้วที่แพงมาก เพราะเราสร้างตึกๆหนึ่งขึ้นมา ครอบเครื่องนี้ไว้ เหมือนกับเป็น chassis แบบหนึ่ง เครื่องนั้นรองอันดับหนึ่งของโลกอยู่หลายปี



EarthSimulator (Facts)

- Located in Yokohoma, Japan
- Size of the entire center about 4 tennis courts
- Can execute 35.86 trillion (35,860,000,000,000) FLOPS, or 35.86 TFLOPS.
- Consists of 640 nodes with each node consisting of 8 vector processors and 16 GB of memory
- Totaling 5120 processors and 10 Terabytes of memory
- Aggregated disk storage of 700 Terabytes and around 1.6 Petabytes of storage in tape drives
- Costs about 350 million dollars
- First on the Top500 list for 5 consecutive times. Surpassed by IBM's BlueGene/L prototype on September 24, 2004



ภาพ 2-28

ในปัจจุบันมีหลายบริษัทได้ผลิตเครื่อง PVP ออกมาราย ในญี่ปุ่นก็มีเครื่อง SX-8 บริษัท Cray เองก็ผลิตเครื่อง Cray X1 ซึ่งออกแบบโดย Steve Scott ภาพด้านบนเป็นภาพที่เผยแพร่รูป กับเครื่อง Cray X1 ที่ ORNL และภาพข้างล่างคือเครื่อง NEC

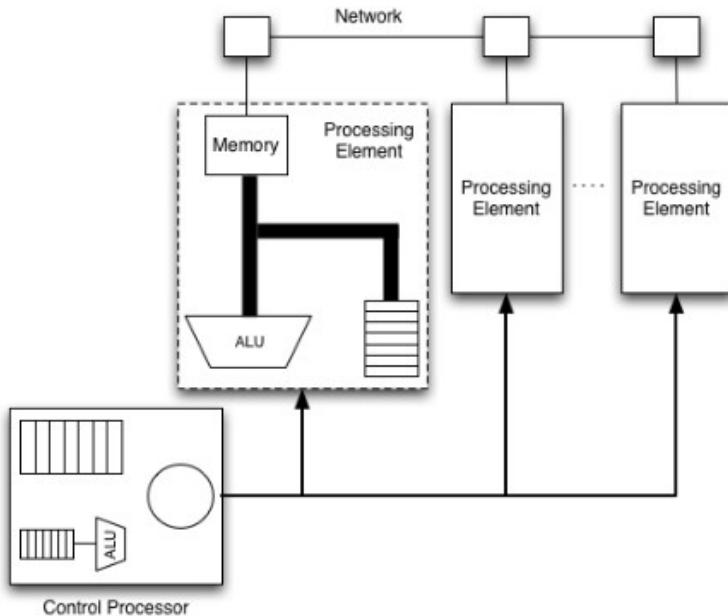
2.10 SIMD

เราจะพูดถึงสถาปัตยกรรมอีกแบบหนึ่งคือแบบ Single Instruction Stream Multiple Data Stream (SIMD) ที่ประกอบไปด้วย ALU จำนวนมาก ซึ่งแต่ละ ALU จะประมวลผลของมัน แต่ทุก ALU จะพึ่งคำสั่งจาก Control Unit เดียวกัน

Draft



SIMD



CSC

Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

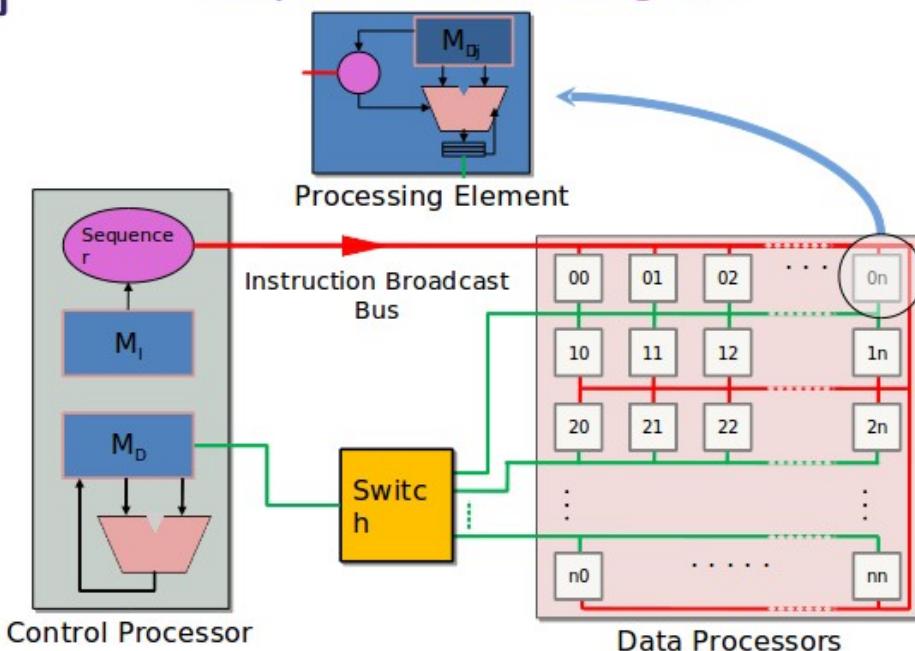
47

ภาพ 2-29
Draft

จาก diagram ในภาพนี้ คุณจะเห็นว่า control unit นั้นอยู่ที่ด้านล่างของภาพบางทีเราก็จะเรียก มันว่า Instruction Unit ซึ่งจะส่ง Instruction หรือคำสั่งไปให้แก่ Processing Element ทุกตัว โดยที่ Processing Element แต่ละตัวนั้นเป็นหน่วยประมวลผลหรือ Processor แบบง่ายๆ ซึ่ง ประกอบไปด้วย Memory bank และ ALU และ register bank ทุกๆ ALU จะนำข้อมูลจาก memory ของมันมาประมวลผล



Simplified SIMD Diagram



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

48

Draft

ภาพ 2-30

นี่ก็เป็นอีกภาพหนึ่งที่แสดงสถาปัตยกรรมแบบ SIMD คุณจะเห็นว่า Control Unit นั้นอยู่ทางซ้าย และมีชุดของ ALU อยู่ทางด้านขวาของภาพซึ่งเราเรียกว่ามันอีกอย่างว่า SIMD array processor

นี่เป็นการขยายขนาดของเครื่องเพื่อรับงานที่มีการประมวลผลขนาดใหญ่โดยใช้วิธีการเพิ่ม processors แบบนี้มากๆ และมีราคาที่คุ้มค่า เพราะทุกๆ processor แซร์ control unit และได้รับการกำหนดให้ทำงานด้วยความเร็วในระดับ Mega Hertz ต่ำๆ ซึ่งจะทำให้ทุกๆ processor ประมวลผลด้วย latency ในระดับ micro seconds หรือส่วนของ micro second แทนที่จะเป็นระดับ nano second หรือส่วนหนึ่งของ nano second

การประมวลผลของ processor ด้วย latency ในระดับ micro seconds นี้เป็นระดับความเร็วที่เหมาะสมกับสถาปัตยกรรมนี้ เพราะคำสั่งจาก control unit ต้องใช้เวลาในการเดินทางมาสู่ processor เนื่องจากในความเป็นจริงนั้นแสดงเดินทาง 1 พุตก์ใช้เวลาในระดับ nano seconds และ electron จะเดินทางด้วยความเร็วประมาณ หนึ่งส่วนสามของความเร็วแสง ดังนั้นมันจึงเป็นไปไม่ได้ที่ control unit จะ broadcast คำสั่งสู่ Processors ต่างๆ ได้ภายในช่วงเวลาระดับ nano second

ตัวอย่างของเครื่องแบบนี้ตัวอย่างหนึ่งคือเครื่อง CM-2 ที่ได้รับการพัฒนาโดยบริษัท Thinking Machine Corporation

อิกเครื่องหนึ่งที่ใช้กันเยอะในช่วงปี 80 คือเครื่องที่ชื่อว่า Maspar ซึ่งได้รับการพัฒนาขึ้น เพื่อใช้ในการคำนวณแบบ floating point เป็นหลัก ประสิทธิภาพของเครื่องชนิดนี้อยู่ในระดับ Gigaflops ในปัจจุบันเราเลิกใช้เครื่อง Maspar นี้ไปแล้วเนื่องจากการ broadcast คำสั่งนั้นมี latency มาก แต่อย่างไรก็ตามถ้าเราพิจารณา ราคาของมันเทียบกับประสิทธิภาพที่ได้แล้ว คุณจะเห็นว่ามันเป็นเครื่องที่มีราคาไม่แพง

Draft



CM-2

CM-2 General Specifications :

- Processors 65,536
- Memory 512 Mbytes
- Memory Bandwidth 300Gbits/Sec
- I/O Channels 8 Capacity per Channel 40 Mbytes/Sec Max.
- Transfer Rate 320 Mbytes/Sec
- Performance in excess of 2500 MIPS
- Floating Point performance in excess of 2500 MFlops



DataVault Specifications :

- Storage Capacity 5 or 10 Gbytes
- I/O Interfaces 2 Transfer Rate,
- Burst 40 Mbytes/Sec Max.
- Aggregate Rate 320 Mbytes/Sec
- Originated at MIT,
- Commercialized at Thinking Machines Corp.



src : <http://www.svisions.com/sv/cm-dv.html>



CSC

Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

49

ภาพ 2-31

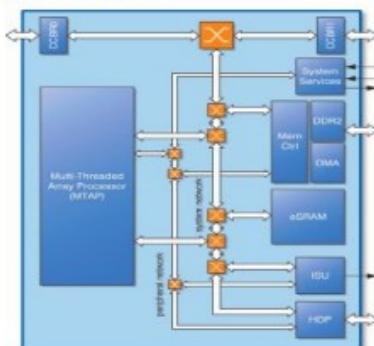
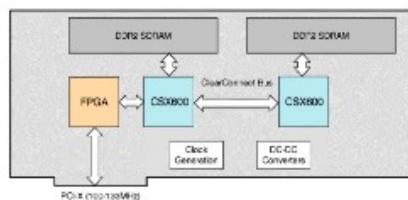
ปัจจุบันนี้เราใช้การประมวลผลแบบ SIMD ในรูปแบบที่เปลี่ยนไป ในภาพเป็นตัวอย่างของ board สร้างขึ้นโดยบริษัท Clearspeed ซึ่งเป็น SIMD board รับคำสั่งแบบ Coarse grained SIMD คือชุดคำสั่งที่ส่งให้แต่ละ Processing unit ทำนั้นมีจำนวนหลายคำสั่งและมีการ synchronize การทำงานระหว่าง processor เป็นระยะๆ ใน board นั้นมี controller ซึ่งจะส่งคำสั่งไปยัง processing unit หลายๆ processing unit ให้ประมวลผลพร้อมๆ กัน

สำหรับคนที่ใช้ Graphic card เช่นของ NVIDIA ก็มีสถาปัตยกรรมเป็นแบบ SIMD ซึ่งทำงานตามคำสั่งที่ส่งมาจาก Instruction unit



ClearSpeed SIMD Accelerator

- Medium-Coarse grained SIMD
- 130nm fabrication technology
- 250 MHz clock rate
- 100 Gflops peak, 66 Gflops sustained



- 1997 Intel ASCI Red Supercomputer
 - 1TFLOPS, 2,500 sq. ft., 800KW, \$55Million
- 2007 ClearSpeed + Intel Dense Cluster
 - 1 TFLOPS, 25 sq. ft., <7 KW, <\$200K



CSC

Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer Architecture, Spring 2009

50

ภาพ 2-32

นี่คือภาพของระบบที่ชื่อว่า Tsubame ซึ่งเป็นเครื่องที่ใหญ่ที่สุดเครื่องหนึ่งในญี่ปุ่น มีประสิทธิภาพประมาณ 100 Teraflops เป็นเครื่องแบบ MPP ในภาพนี้ผมยืนคู่กับ Professor Satoshi Matsuimoto ผู้ออกแบบเครื่อง จุดเด่นอย่างหนึ่งของเครื่องนี้ก็คือมันได้รับการติดตั้ง clearspeed accelerator เป็นร้อยๆ card



Tsubame

- Heterogeneous computing : Added ClearSpeed Boards
- 648 nodes resulting in 38.5 TFLOPS
- 648 nodes with 360 ClearSpeed boards to 47.38 TFLOPS



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer Architecture, Spring 2009

51

ภาพ 2-33

2.11 Special Purpose Device

ถ้า course นี้มีเวลามากกว่านี้เราจะศึกษาสถาปัตยกรรมอิกแบบหนึ่งชื่อว่า Special Purpose Devices เครื่องคอมพิวเตอร์ที่เร็วที่สุดในโลกในปัจจุบันก็เป็นเครื่อง Special Purpose Device ในญี่ปุ่นชื่อว่า MD-Grape ซึ่งถูกสร้างมาสำหรับ Algorithm แบบหนึ่งโดยเฉพาะและมีความเร็วในระดับ Petaflops ตั้งแต่ปีที่แล้ว

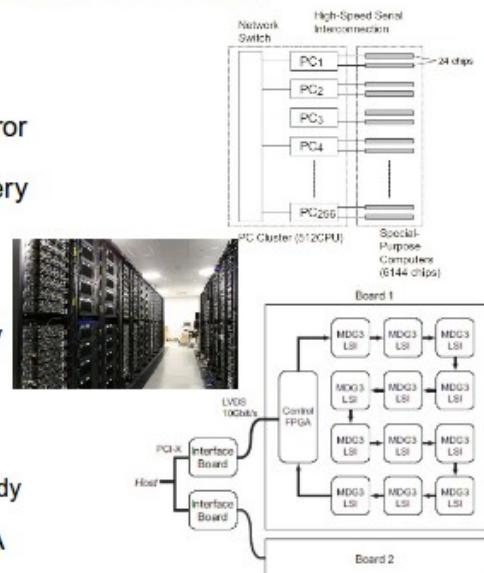
นอกจากนั้น จะว่าไปแล้ว Graphic card เช่น NVIDIA และระบบที่ใช้เทคโนโลยี FPGA ก็ถือได้ว่าเป็น Special Purpose Hardware ซึ่งทำงานเฉพาะด้าน



Special Purpose Devices

• SPD

- Optimized for a given algorithm or class of problems
- Functional elements and dataflow path mirror the requirements of a specific algorithm
- Usually exploits fine grain parallelism for very high parallelism
- Best for arithmetic (or logic) intensive applications with limited memory access requirements
- Best for strong temporal and spatial locality
- Systolic Arrays** are one class of such machines widely used in digital signal processing
- Examples
 - MD-Grape first Petaflops machine, for N-body problem
 - GPU Graphics Processing Unit, e.g. NVIDIA
 - FPGA field programmable gate array
 - Allows reconfiguration of logic array



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

53

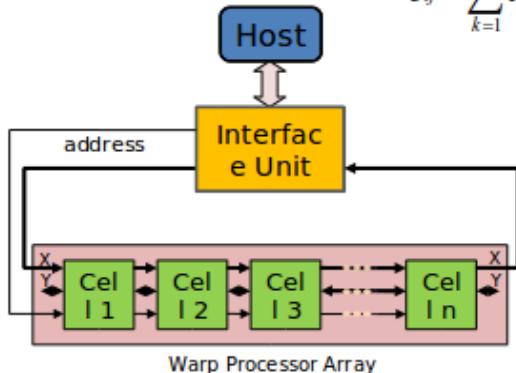
ภาพ 2-34

2.12 Systolic Array

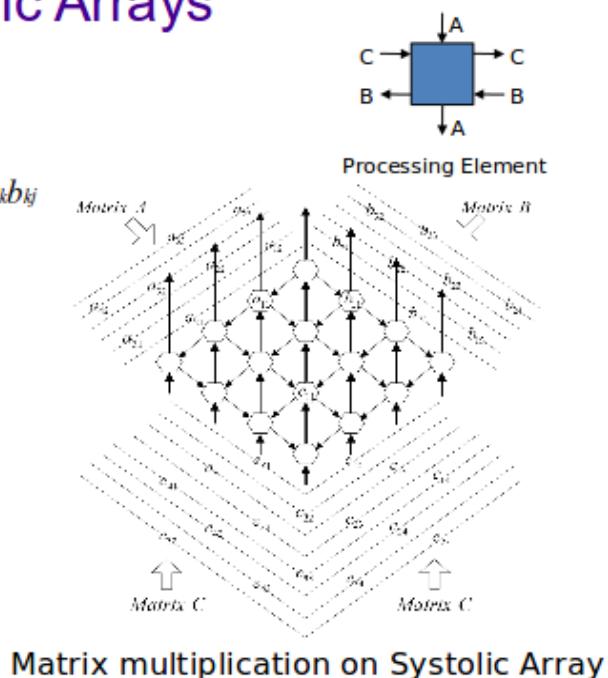
slide นี้แสดงภาพของเครื่อง Special Purpose Device ที่มีสถาปัตยกรรมเป็นแบบ Systolic Array และได้รับการออกแบบสำหรับการประมวลผล dot product ในภาพทางซ้ายเป็น hardware ที่ถูกสร้างขึ้นมาทำงานแบบหนึ่งมิติ และทางด้านขวาเป็นระบบที่ถูกออกแบบมาให้ทำงานแบบสองมิติ



Systolic Arrays



$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$



Example implementation:
Warp architecture

Matrix multiplication on Systolic Array

References:

- M. Annaratone, E. Arnould, et al, "The Warp Computer: Architecture, Implementation, and Performance"
Y. Yang, W. Zhao, and Y. Inoue, "High-Performance Systolic Arrays for Band Matrix Multiplication"



CSC

Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

54

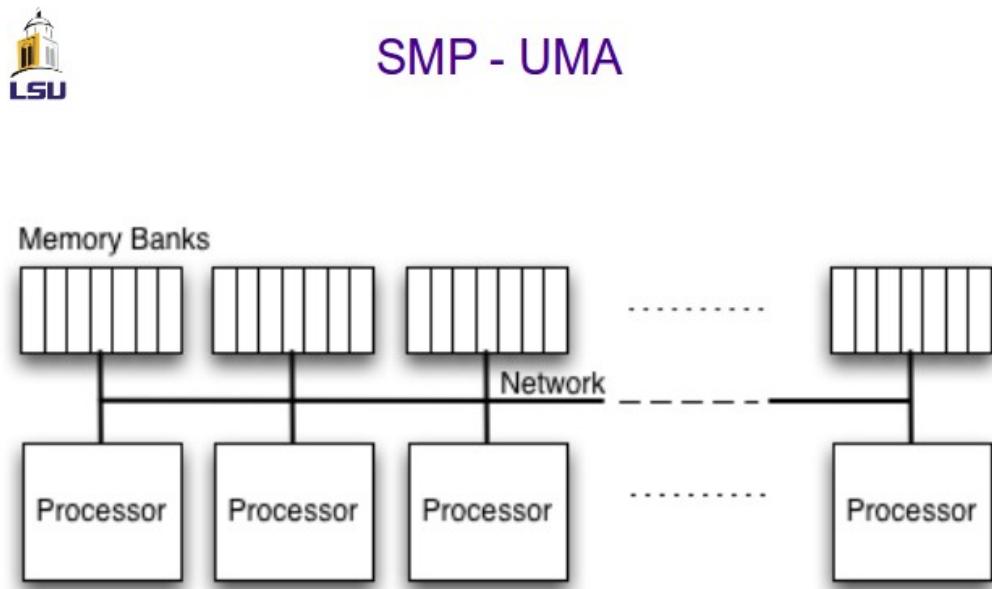
ภาพ 2-35

ในวันนี้ผมจะแนะนำให้คุณรู้จัก-Ideas หลายๆ Ideas เกี่ยวกับสถาปัตยกรรมคอมพิวเตอร์

2.13 Introduction to SMP

เราจะพูดถึงชนิดของ High Performance Computer อีกชนิดหนึ่งชื่อว่า Symmetric Multiprocessor หรือ SMP ซึ่งประกอบไปด้วย Processor จำนวนหนึ่งที่ใช้ Global Memory เดียวกัน ทุกๆ Processor จะใช้เวลาเข้าถึง Module ได้เท่ากัน processor เหล่านี้อาจมี cache 1 ถึง 2 cache ซึ่งเป็น memory buffer ความเร็วสูงและมีการบริหารจัดการข้อมูลที่ Transparent คือมันจะทำงานโดยอัตโนมัติ ถ้า Processor ต้องการข้อมูลจาก memory address ได้ก็ตามมันก็จะจัดการการโอนถ่ายข้อมูลระหว่างตัวมันกับ memory โดยอัตโนมัติ และส่งข้อมูลใน address นั้นให้กับ processor ในกรณีที่มีข้อมูลของ address เดียวกันอยู่ใน cache ของหลายๆ processor เราต้องมีวิธีการที่จะทำให้มันใจได้ว่าข้อมูลเหล่านั้นมีค่าเหมือน กัน ซึ่งเราเรียกคุณสมบัติแบบนี้ว่า cache coherent หรือ cache consistency ระบบที่ใช้จัดการเรื่อง cache coherent นี้ส่วนใหญ่จะได้รับการติดตั้งเป็น hardware เพื่อให้การทำงานเป็นไปอย่างรวดเร็ว

ปัญหาของสถาปัตยกรรมแบบ SMP คือ มันไม่ scale คุณอาจสามารถออกแบบให้ SMP มีจำนวน processor ได้ระดับหนึ่ง 4 หรือ 8 หรือ 16 หรือ 32 processors แต่ถ้าคุณต้องการจะให้ระบบของคุณมี processor มากกว่าหนึ่ง การออกแบบก็จะยากขึ้นมากโดยเฉพาะอย่างยิ่งถ้าคุณต้องการให้ทุกๆ processors เข้าถึง memory แบบ uniform memory access คือใช้เวลาในการเข้าถึงเท่ากัน



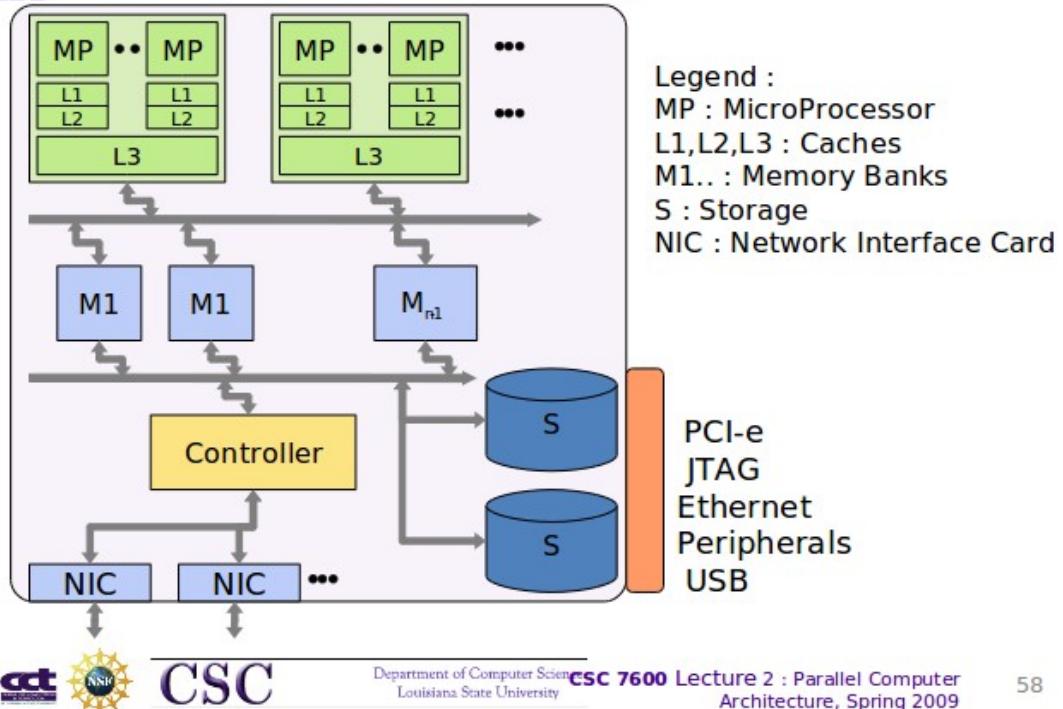
ภาพ 2-36

Diagram นี้แสดงภาพการเชื่อมต่อของ processor กลุ่มหนึ่งกับ memory bank อีกกลุ่มหนึ่ง ผ่านระบบ network คุณอาจสังสัยว่าจะเกิดอะไรขึ้นเมื่อ processor มากกว่าหนึ่ง processor พยายามเข้าถึง memory location เดียวกัน processor ไหนจะได้เข้าถึงก่อนหลังและเราจะปริหารจัดการการ synchronization ระหว่างการทำงานของส่วนประกอบของระบบเหล่านี้อย่างไร

ในระบบแบบ SMP การบริหารจัดการตั้งกล่าวสามารถทำได้โดยใช้ hardware หรือ software แต่อย่างไรก็ตามมันก็เป็นส่วนที่ทำให้ระบบซับซ้อนขึ้นและทำให้เกิด Overhead ใน การเข้าถึงข้อมูล



SMP Node Diagram



CSC

Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

58

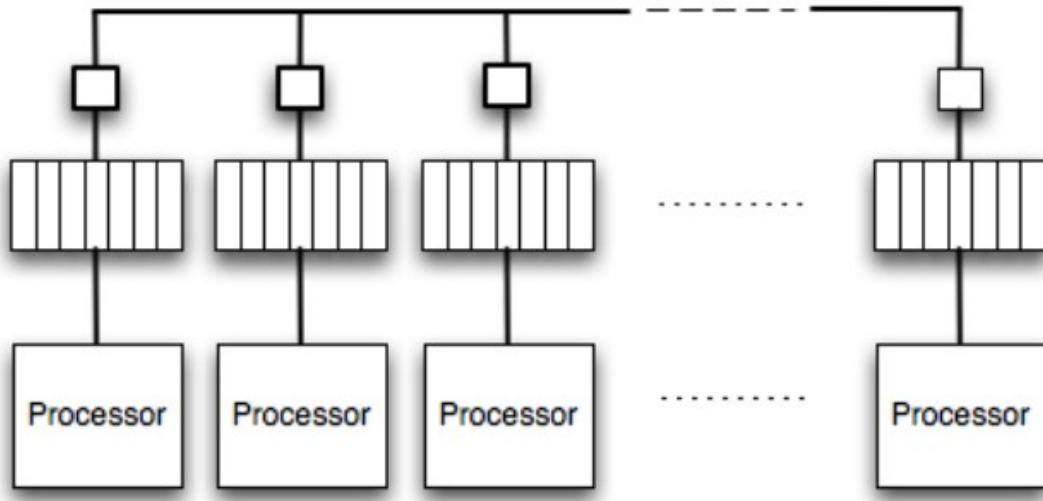
ภาพ 2-37

ภาพนี้แสดงรายละเอียดของระบบ SMP คุณจะสังเกตเห็นว่ากล่องสีเขียวในภาพซึ่งมีอักษร MP ที่เรามายถึง Micro processor แต่ในปัจจุบันเรารู้จะเรียกมันว่า core มากกว่า คุณจะเห็นจากภาพได้ว่ากล่องสีเขียวทุกกล่องได้รับการเชื่อมต่อเข้ากับ Bus และในขณะเดียวกันก็มี memory bank กลุ่มหนึ่งเชื่อมต่อเข้ากับ Bus ด้วย การเข้าถึงข้อมูลนั้น processor แต่ละ processor สามารถเข้าถึง memory bank ได้ ก็ได้ นอกจากนั้นระบบในภาพยังมี controller สำหรับบริหารจัดการ I/O และ Network Interface Controller หรือ NIC ซึ่งอาจมีอันเดียวหรือมากกว่านั้นก็ได้ ในระบบของคุณอาจมี Disks อยู่หนึ่งถึงสอง Disk ต่อหนึ่ง node ของระบบ SMP แบบนี้ และนอกจากนั้นก็ระบบอาจมี interfaces สำหรับอุปกรณ์ต่อพ่วงอื่นๆอยู่ด้วย

ทั้งหมดนี้เป็นตัวอย่างของระบบที่เราจะได้ใช้งานมันต่อไปในวิชานี้ แต่ขอให้ระลึกไว้เสมอว่าระบบ SMP นั้นไม่ scale เราไม่สามารถมี processor จำนวนมากเป็นพันๆ processors ในระบบแบบนี้ได้ เพราะมันจะมีปัญหาในการเข้าถึง memory แบบ uniform memory access เนื่องจากความซับซ้อนในการบริหารจัดการ การมี overhead สูง และมีราคาแพง



DSM - NUMA



ภาพ 2-38

Draft

เราสามารถทำให้ระบบ SMP มี scalability มากขึ้นโดยการใช้สถาปัตยกรรมแบบ Distributed Shared Memory ซึ่งมีการเข้าถึง memory แบบ Non-Uniform Memory Access หรือ NUMA ในภาพนี้คุณจะเห็นว่า processor ทุก processor มี local memory bank ประจำตัวของมัน processor แต่ละ processor สามารถเข้าถึง memory bank ได้ในระบบก็ได้ แต่ถ้า memory bank ที่มันต้องการเข้าถึงไม่ใช่ local memory bank มันก็จะใช้เวลาในการเข้าถึงนานกว่าที่ใช้ในการเข้าถึง local memory bank

SGI เป็นบริษัทที่ผลิตเครื่องชั้นนำนี้ ในการใช้งานมันคุณต้องคิดถึงเรื่องประสิทธิภาพในระหว่างที่ application ทำงานด้วย เพราะถึงแม้ว่าสถาปัตยกรรมนี้จะเป็นแบบ shared memory แต่เนื่องจากเวลาที่ใช้เข้าถึง memory ที่แตกต่างกัน คุณต้องคำนึงถึงการจัดวางใน memory bank ต่างๆในระบบรวมทั้งการ schedule การเข้าถึงข้อมูลของ processor ด้วย

ระบบแบบนี้อาจเหมาะสมกับงานบางอย่าง แต่สำหรับบาง applications ผู้ผลิตว่ามันไม่ได้มีประสิทธิภาพสูงกว่าระบบแบบ Distributed Memory เลย

2.14 Challenge to Computer Architecture

ในช่วงหลังของเรื่องสถาปัตยกรรมคอมพิวเตอร์นี้เราจะมาพิจารณาความท้าทายหรือปัญหาในการออกแบบระบบและความพยายามที่จะแก้ปัญหาเหล่านั้น

ระบบ high performance computer นั้นมีประสิทธิภาพเพิ่มมากขึ้นเรื่อยๆ ในอดีตเราจะพูดถึงความพยายามที่จะสร้างเครื่องที่มีความเร็วในระดับ Teraflops ในปัจจุบันเรามาทำลังจะมีเครื่องในระดับ Petaflops และเราคาดว่าต่อไปในอนาคตภายในปี 2020 เราจะมีเครื่องคอมพิวเตอร์ที่มีความเร็วขนาด Exaflops ซึ่งมีความเร็วประมาณสองยกกำลัง ยี่สิบ หรือ สิบ ยกกำลังสิบแปด ก็คือประมาณล้านล้าน flops

จากประวัติศาสตร์แล้วเรามักจะมีรอบของการออกแบบและพัฒนาเครื่องคอมพิวเตอร์รอบหนึ่งใช้เวลาประมาณ 7 ปี ภายในปลายทศวรรษหน้าหรือสองรอบของการออกแบบสถาปัตยกรรมคอมพิวเตอร์แบบใหม่ เรายังไม่มีเครื่องระดับ Exaflops อกมما

เรื่องที่เป็นความท้าทายอีกอย่างหนึ่งในการสร้างเครื่องแบบ Exaflops นี้ก็คือเรื่องของการใช้พลังงาน เราคาดว่ามันอาจต้องใช้พลังงานสูงถึง 100 mega watts ดังนั้นการออกแบบระบบให้มีการใช้พลังงานอย่างประหยัดก็เป็นเรื่องที่สำคัญมากเรื่องหนึ่ง

สิ่งสำคัญมากๆ อีกสิ่งหนึ่งที่เราต้องพิจารณาคือเรื่องของขนาดของ memory ในระบบ ถ้าดูจากสัดส่วนของงบประมาณที่ใช้ในการลงทุนกับส่วนประกอบต่างๆ ของระบบแล้ว คุณจะพบว่าเงินส่วนใหญ่ส่วนหนึ่งจะถูกใช้เพื่อการสร้างระบบ memory

หลักเกณฑ์ที่เราใช้พิจารณาสร้างระบบดังกล่าวได้แก่ การพิจารณา memory bandwidth และความจุของ memory โดยรวมของระบบ ในเรื่องของความจุนั้นเราอาจเริ่มต้นจากการประมาณว่าเราจะใช้ memory 1 byte ต่อ 1 flop ซึ่งสำหรับระบบใหญ่ๆ แล้วเราต้องใช้ memory เป็นจำนวนมากและเสียค่าใช้จ่ายมากด้วย เมื่อเทียบราคากับ ALU และราคาของ memory จะสูงกว่ามาก เพื่อความเหมาะสมเราจึงตั้งเกณฑ์ไว้ที่ 1 byte ต่อ 5 หรือ 10 flops แต่แนวโน้มน่าจะเป็นที่ 1 byte ต่อ 20 flops หากกว่า ซึ่งสิ่งเหล่านี้ส่งผลต่อคุณภาพและราคากลางๆ ของเครื่อง

สิ่งที่จะต้องพิจารณาอีกอย่างหนึ่งคือเรื่อง memory latency ซึ่งก็เป็นปัญหาในการออกแบบระบบคอมพิวเตอร์ที่นับวันก็ยกขึ้น ในการทำงานของ CPU นั้น การทำงาน 1 clock cycle จะใช้เวลาประมาณส่วนหนึ่งของหนึ่ง second แต่เวลาในการเข้าถึง memory นั้นจะใช้เวลากว่า 200 cycles ยิ่งถ้าเป็นการเข้าถึง remote memory หรือ memory ของ processor อื่นด้วยแล้วอาจต้องใช้เวลากว่า 1000 หรือ 10000 cycles ใน การเข้าถึง ความแตกต่างเช่นนี้ทำให้เกิดปัญหา memory wall ดังที่ได้กล่าวมาแล้ว

เนื่องจากการใช้เทคโนโลยี multicore ในปัจจุบัน อัตรา clock rate จะไม่เพิ่มขึ้นมาก

นัก แต่ในทางตรงกันข้าม จำนวน processors ในระบบจะเพิ่มจำนวนมากขึ้น จำนวน core ก็จะมีมากขึ้น เป็นผลให้ขนาดของเครื่อง supercomputer ใหญ่ขึ้น ดังนั้นเวลาที่ใช้ในการสื่อสารระหว่างส่วนประกอบต่างๆก็มากขึ้นด้วยนั่นก็หมายถึงค่า latency ของระบบที่สูงขึ้นนั่นเอง

ปัญหาอีกเรื่องหนึ่งที่เราจะต้องพิจารณาคือเรื่องของ overheads ในการทำงานแบบ parallel ผู้อยากรู้ว่าให้เห็นนะครับว่า granularity หรือความละเอียดของการแบ่งขนาดของการประมวลแบบขนาดนั้นจะละเอียดได้มากเท่าไรก็ขึ้นอยู่กับความเล็กของ overhead ยิ่ง overhead เล็กมาก granularity ก็จะละเอียดมากตาม แต่ถ้าเราไม่สามารถลด overhead ของระบบลงได้ มันก็ไม่มีประโยชน์ที่จะทำให้ granularity ของการประมวลผลละเอียดขึ้น

คุณอาจนึกเซ่นนี่นะครับว่าในการแก้ปัญหาอย่างโดยย่างหนึ่งซึ่งมีขนาดค่อนข้างใหญ่ซึ่งต้องใช้การประมวลผลจำนวนหนึ่งเพื่อแก้ปัญหานั้น ถ้าคุณมี processor เพียง processor เดียว ก็อาจใช้เวลานาน แต่สมมุติว่าคุณมี processors หลาย processors คุณก็สามารถแบ่งการประมวลผลไปยังแต่ละ processor ในปริมาณที่เท่าๆกัน ดังนั้น สมมุติว่าคุณมีปริมาณของการประมวลผลเป็นปริมาณคงที่ปริมาณหนึ่ง ยิ่งคุณมี processors มากราคาเท่าไร ขนาดของการประมวลผลที่ถูกแบ่งไปยังแต่ละ processor ก็มีจำนวนน้อยลงเท่านั้น เราอาจมองง่ายๆว่า ปริมาณการประมวลผลที่น้อยลงของแต่ละ processor ดังกล่าวจะน้อยกว่า granularity ที่น้อยลงนั่นเอง

ในขณะเดียวกัน การใช้งาน processor หลายๆ processor ก็คือการเพิ่ม parallelism ใน การประมวลผล การทำงานแบบ parallel จะทำให้เกิด overhead ในการแบ่งงาน การประสานงาน การรวมรวมผลงาน และอื่นๆตามมา ซึ่งความสัมพันธ์ระหว่าง overhead กับ granularity นั้นก็เป็นดังที่ผมกล่าวมาก่อนหน้า ก็คือ granularity ของการประมวลแบบขนาดนั้นจะละเอียดได้มากเท่าไรก็ขึ้นอยู่กับความเล็กของ overhead ยิ่ง overhead เล็กมาก granularity ก็จะสามารถละเอียดได้มากเป็นสัดส่วนกันไป

ในกรณีที่เราลด overhead ไม่ได้เราก็สามารถเพิ่มประสิทธิภาพของการประมวลผลของระบบได้โดยใช้วิธีเพิ่มขนาดของปัญหา เพื่อให้อัตราส่วนของ granularity ของการประมวลผล กับ overheads อยู่ในระดับที่เหมาะสม

ในอนาคตถ้าเรามีการประมวลผลที่มี granularity ละเอียดมากๆ ความเป็น parallelism ของระบบก็จะมีมากขึ้นด้วยเพื่อรับการประมวลผลนั้น ความเป็น parallelism ดังกล่าวไม่ใช่แค่ในระดับ 100 หรือ 1,000 หรือ 10,000 parallelism แต่จะเป็นในระดับ หลายล้าน parallelism

อีกประเด็นหนึ่งซึ่งมีความสำคัญต่อการออกแบบสถาปัตยกรรมคอมพิวเตอร์ก็คือเรื่องของพลังงานหรือ power ซึ่ง power นั้นมีความสัมพันธ์กับ clock rate และมันก็เป็นเหตุผลหลักที่ทำให้เราไม่สร้างระบบที่มี clock rate สูงขึ้นเรื่อยๆ ในปัจจุบันเราจะเห็นได้ว่า clock rate จะอยู่

ในระดับคงที่หรือเพิ่มเล็กน้อยเท่านั้น

ความนำเชื้อถือเป็นอีกเรื่องหนึ่งที่มีผลต่อการออกแบบระบบฯ เมื่อเราเพิ่มส่วนประกอบหรือ node เข้าสู่ระบบมากขึ้น โอกาสที่ส่วนประกอบหรือ node เหล่านั้นจะเกิดความผิดพลาดหรือเสียไปมากขึ้น ทำให้เวลาโดยเฉลี่ยก่อนที่จะมีความผิดพลาดเกิดขึ้นหรือ Mean Time Between Failures นั้นลดลง ดังนั้นเราต้องมีวิธีการสำหรับจัดการความผิดพลาดเหล่านี้หรือ Fault Tolerance นั้นเอง

ความท้าทายในการออกแบบสถาปัตยกรรมอันสุดท้ายก็คือเรื่องของความซับซ้อนในการออกแบบ ซึ่งส่งผลกระทบต่อค่าใช้จ่ายในการค้นคว้าและพัฒนาสถาปัตยกรรมนั้นขึ้นมา จริงๆ แล้วสำหรับเครื่องคอมพิวเตอร์ราคาร้อยล้านเหรียญ ค่าใช้จ่ายในการออกแบบระบบที่ซับซ้อนนั้น ขึ้นมาอาจต้องใช้ถึง พันล้านเหรียญก็ได้ ความซับซ้อนที่ว่านี้ยกตัวอย่างเช่นถ้าคุณต้องการเพิ่ม parallelism จำนวนมากในระบบของคุณ คุณอาจต้องพัฒนาระบบที่มีจำนวน transistor เป็นร้อยล้าน transistor ในระบบนั้น ซึ่งไม่ใช่เรื่องง่ายเลย

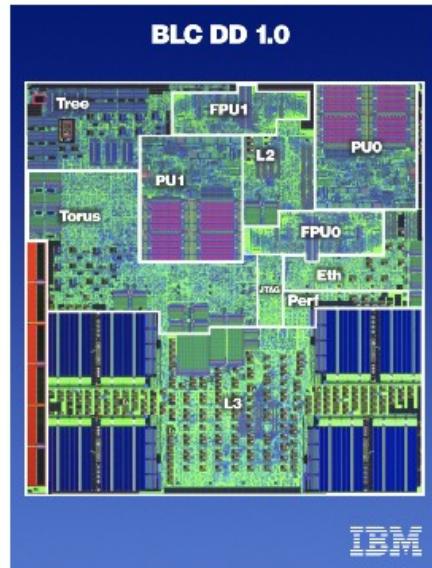
ในสไตล์นี้เราจะพูดถึงเทคโนโลยีในปัจจุบัน ในอดีตที่ผ่านมาเราจะเห็นการเปลี่ยนแปลงของสถาปัตยกรรมของ supercomputer หลายแบบ เช่นการเปลี่ยนแปลงจากเครื่อง mainframe ไปเป็นเครื่อง vector processor และหลังจากนั้นก็เปลี่ยนเป็น SIMD processor ต่อจากนั้นก็มีกระแสของสถาปัตยกรรมแบบ Data flow มาแรงอยู่พักหนึ่งแต่ก็ไม่ได้ถูกผลิตมาขายในห้องตลาด หลังจาก SIMD processor ก็เป็นสถาปัตยกรรมแบบ MPP ที่ได้รับความนิยมในปัจจุบัน การเปลี่ยนแปลงเหล่านี้เกิดขึ้นทุกๆ 15 ปีโดยประมาณ

ในปัจจุบันเรารู้อยู่ในช่วงหัวเลี้ยวหัวต่ออีกรังหนึ่ง สาเหตุเนื่องมาจากความซับซ้อนในการออกแบบสถาปัตยกรรมของ processor ที่ซับซ้อนขึ้นเรื่อยๆ จนเราแบบจะหมดได้เดียวที่จะทำให้มันซับซ้อนมากไปกว่านี้ นอกจากนั้นยังมีเรื่องของการใช้พลังงานของ processor ที่เราพบว่า การเพิ่ม clock rate ทำให้ processor ต้องใช้พลังงานเพิ่มขึ้นมาก ซึ่งปัญหาเหล่านี้เป็นสาเหตุให้เราต้องออกแบบสถาปัตยกรรมแบบใหม่ขึ้น

ซึ่งก็คือยุคของสถาปัตยกรรมแบบ multicore นั้นเอง สถาปัตยกรรมแบบนี้เป็นการบรรจุ processor หลายๆ processors เข้าไปใน chip ๆ เดียวหรือในแผ่น semiconductor หรือ die ๆ เดียวทั้งหมด การทำเช่นนี้ทำให้ processor มีความสามารถมากขึ้นตามจำนวน core ที่มันมี



IBM Blue Gene/L



Department of Computer Science
Louisiana State University

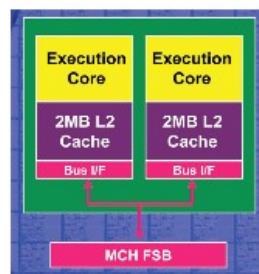
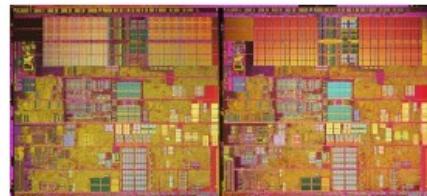
CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

63

ภาพ 2-39
นี่เป็นภาพของ Chip ของเครื่อง Bluegene Light ซึ่งมี core 2 core ใน chip นี้ ในสไลด์
Pentium ของ Intel ก็มี 2 core ซึ่งในปัจจุบัน บริษัท Intel ผลิต chip แบบ 4 cores ออกมากด้วย



Intel Pentium Extreme



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

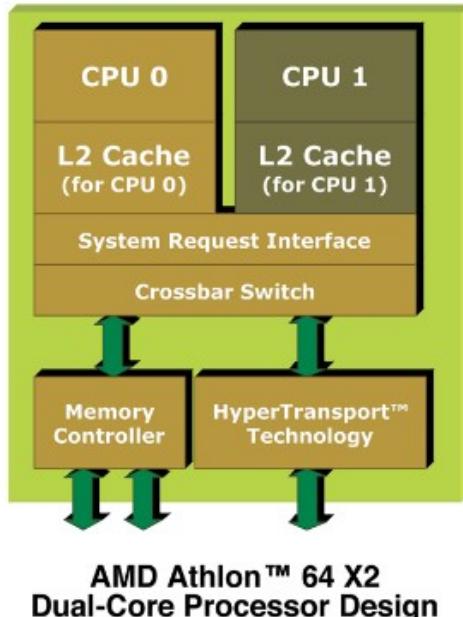
64

ภาพ 2-40

AMD เองก็ผลิต chip ที่มี 2 core ดังในภาพ และมี chip แบบ Quad core ด้วย



AMD Dual Core Architecture



AMD Athlon™ 64 X2
Dual-Core Processor Design



Department of Computer Science

Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

65



ภาพ 2-41

Chip ที่กล่าวมาแล้วล้วนแต่ใช้ชุดคำสั่งแบบเดียวกันคือแบบ x86 ซึ่งมีความซับซ้อนของการออกแบบในระดับหนึ่งที่เดียว ตรงนี้มีประดิษฐ์ที่นำเสนออยู่นะครับว่าถ้าเราเลือกที่จะออกแบบสถาปัตยกรรมชุดคำสั่งใหม่ที่ไม่ซับซ้อนแต่ออกแบบ core ให้ง่ายๆและใช้เป็นจำนวนมากแล้วจะเกิดอะไรขึ้น เช่นมีประดิษฐ์ภาพมากขึ้นหรือไม่ มีความน่าเชื่อถือมากขึ้นหรือไม่ และใช้พลังงานลดลงหรือไม่

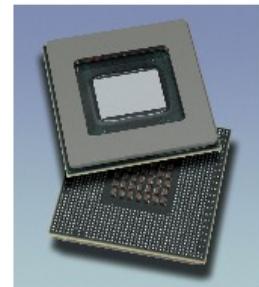
ใน slide นี้เราพูดถึง chip ที่บริษัท Toshiba SONY และ IBM ได้ร่วมกันพัฒนาขึ้นมา ข้อว่าสถาปัตยกรรมแบบ cell

Chip นี้เป็น chip ที่ใช้ในเครื่อง playstation 3 บริษัท Toshiba จะนำมามาใช้ผลิต Desktop คอมพิวเตอร์ ส่วน IBM ก็จะใช้มันสำหรับการประมวลผลที่มีความต้องการความสามารถในการประมวลผลสูงๆหรือ High End Computing



IBM/SONY Cell Architecture

- Product of the “STI” alliance: SCEI (Sony), Toshiba and IBM
- Budget estimate ~\$400 mil
- Primary design center in Austin, TX (March 2001)
- Modified POWER4 toolchain
- The effort took 4 years, with over 400 engineers and 11 IBM centers involved
- Original target applications:
 - Sony Playstation 3
 - IBM blade server
 - Toshiba HDTV



ภาพ 2-42

Draft

ภาพนี้เป็นภาพของสถาปัตยกรรมแบบ cell ซึ่งมี core ถึง 9 core ประกอบไปด้วย core เล็กที่ชื่อว่า Synergetic Processing Elements หรือ SPE 8 core และ core แบบที่เป็น PowerPC core อีก 1 core สถาปัตยกรรมแบบนี้มีประสิทธิภาพสูงมากเพราะ chip ๆ เดียวสามารถทำงานได้เร็วถึง 1 ส่วน 4 Teraflops

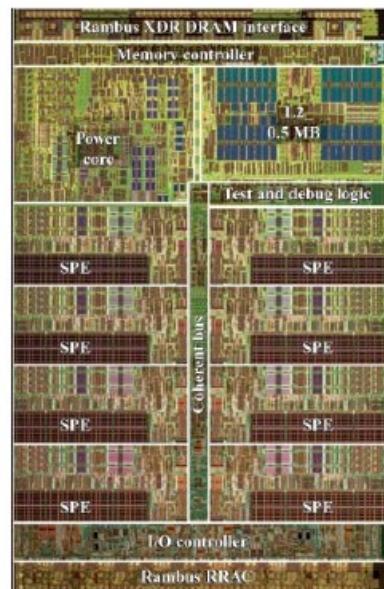
เมื่อพูดถึงเรื่องราคาแล้วมันก็ไม่แพงเกินไปนักเพราะคุณสามารถหาชื้อ Playstation 3 ได้ในราคาย่อมเยา 400 เหรียญ...ใช่ไหมครับ คุณลองไปเช็คที่ร้านดูได้ หรือถ้าคุณจะเครื่อง playstation 3 หลายๆเครื่องมาต่อเชื่อมเข้าด้วยกันแล้วทำให้เป็น cluster และใช้ประมวลผลเกมส์หรือ render ภาพก็ได้

จริงๆแล้วได้มีการนำ chip แบบนี้มาใช้สร้าง supercomputer ชื่อก็คือเครื่อง Road Runner ที่ศูนย์วิจัย Los Alamos และในปัจจุบันเครื่องๆนึงก็เป็นเครื่องที่มีประสิทธิภาพมากที่สุดในโลก



Cell Components and Layout

- One Power Processing Element (PPE)
- Multiple Synergistic Processing Elements (SPE)
- Element Interconnect Bus (EIB)
- Dual channel XDR memory controller
- FlexIO external I/O interface



CSC

Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

70

ภาพ 2-43

ต่อไปในอนาคตเราจะพูดถึง Multicore และ core แบบอื่นๆที่มีลักษณะเป็น Heterogeneous Processing มาขึ้น และเราจะศึกษาด้วยว่า core เหล่านี้ประสานงานกันอย่างไร

Draft



Cell Processor in Numbers

- 234 mil transistors
- 221mm² die on 90nm process
- SOI, low-k dielectrics, copper interconnects
- 3.2GHz clock speed (over 5Ghz in lab)
- Peak performance:
 - over 256Gflops @4GHz, single precision
 - ~26Gflops, double precision
 - memory bandwidth: 25.6Gbytes/s
 - I/O bandwidth: 76.8Gbytes/s (48.8 outbound, 32 inbound)
- Power consumption undisclosed, estimated at 30W (MacWorld) or 50-80W (other sources); 5 power states



CSC

Department of Computer Science
Louisiana State University

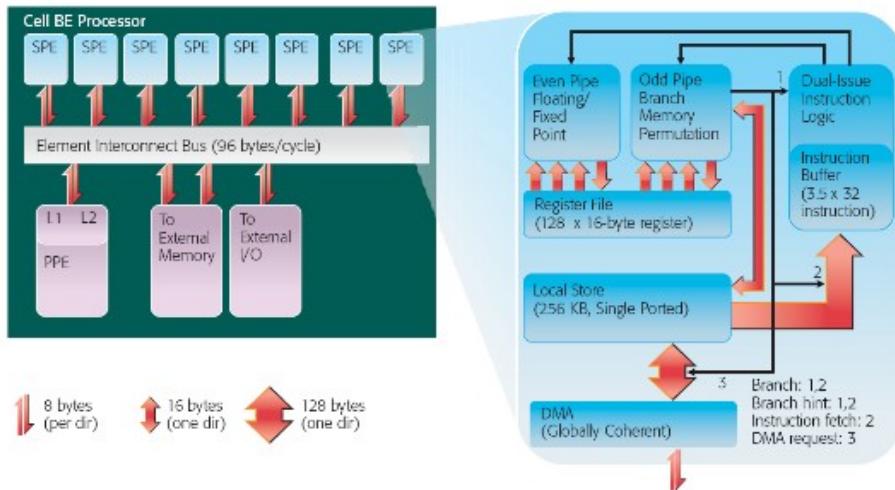
CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009

68

ภาพ 2-44



Internal Structure



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 2 : Parallel Computer Architecture, Spring 2009

69

ภาพ 2-45
Draft

2.15 Commodity Cluster

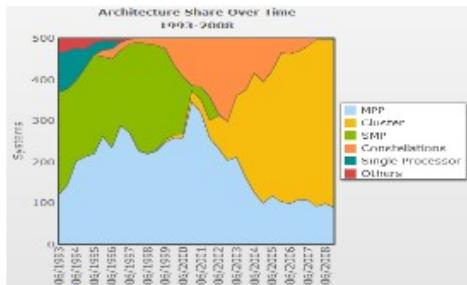
มีสถาปัตยกรรมอิกแบบหนึ่งที่ผลิตไม่ได้พุดถึงและผลจะพุดถึงมันในรายละเอียดในบทถัดไป สถาปัตยกรรมที่ว่านี้ก็คือ Commodity Cluster

ในพัฒนาการของ supercomputer มีคอมพิวเตอร์สองแบบที่ไม่เปลี่ยนไปมากนักและมีการใช้งานอยู่ในหลายปีที่ผ่านมา อย่างแรกคือ MPP หรือ Massively Parallel Processor ก็ได้แก่ เครื่องคอมพิวเตอร์แบบเครื่อง ASCII White เครื่อง BlueGene และเครื่อง Road Runner เป็นต้น คอมพิวเตอร์อิกแบบคือ supercomputer สำหรับคนจน หรือ poorman supercomputer หรือเรียกอิกอย่างว่า commodity cluster ซึ่งเป็นการนำเอาคอมพิวเตอร์ที่มีขายอยู่ทั่วไปในห้องตลาดหรือ off-the-shelves คอมพิวเตอร์มาเชื่อมต่อกันด้วย System Area Network หรือ Local Area Network ที่มีขายอยู่ทั่วไปเช่นกัน



Commodity Clusters

- Distributed Memory systems
- Superior performance to cost
- Dominant parallel systems architecture on the Top 500 List
- Combines off the shelf systems in scalable structure
- Employs commercial high-bandwidth networks for integration
- Message Passing programming model used (e.g. MPI)
- First cluster on Top500 : Berkley Now, 1997



CSC

Department of Computer Science
Louisiana State University

**CSC 7600 Lecture 2 : Parallel Computer
Architecture, Spring 2009**

73



ในภาพคุณจะเห็นผมยืนอยู่หน้าเครื่อง cluster รุ่นที่ 4 ที่เราเรียกว่าระบบ Beowulf Cluster ระบบในภาพนี้ติดตั้งอยู่ที่มหาวิทยาลัย California Institute of Technology หรือ Caltech เครื่องนี้มีประสิทธิภาพประมาณ 10 Gigaflops

ในปี 1997 เครื่องที่เป็น commodity cluster เครื่องแรกคือเครื่อง Berkeley NOW หรือ Network of Workstations ซึ่งติดอันดับเป็นเครื่องคอมพิวเตอร์ที่เร็วที่สุดในโลก 500 เครื่องแรก ในอีก 10 ปีถัดมา เครื่อง supercomputer ส่วนใหญ่ในโลกนั้นเป็น commodity cluster กว่า 70 % ของเครื่อง 500 อันดับแรก หรือ Top 500 list นั้นก็เป็น commodity cluster ทั้งสิ้น เราจะพูดถึง commodity cluster ในรายละเอียดกันต่อไป สำหรับในวันนี้ขอบคุณทุกคนมากครับ

บทที่ 3 ระบบ Commodity Cluster

เนื้อหา เรียนเรียงจากวิศวอุปัชชอง Prof. Dr. Thomas Sterling ผู้เชี่ยวชาญทางด้าน High Performance Computing จาก Louisiana State University
แปลและเรียนเรียง กษิติ ชาญเชี่ยว

ในบทที่แล้วเราพูดถึงเรื่อง Supercomputer Architecture เราพูดว่า Supercomputing คือ การใช้เทคโนโลยีที่มีอยู่ และใช้ Parallelism ที่มีอยู่ในการประมวลผลต่าง ๆ ในเรื่องของ โครงสร้างของสถาปัตยกรรมคอมพิวเตอร์นั้น เราจะพิจารณาความเป็น Parallelism สำหรับ สถาปัตยกรรมแบบ Parallel ใน 3 รูปแบบ คือ

1. แบบ Coarse grain คือการที่มีโปรแกรมทำงานบน OS ของเครื่องต่าง ๆ และ โปรแกรมเหล่านั้นทำงานร่วมกัน
2. ในแบบที่ Medium Grain จะอ่อนตัวขึ้น คือ การที่มีหน่วยประมวลผล ALU หลาย ๆ อันทำงานร่วมกัน
3. และท้ายที่สุด คือ แบบ Fine grain อย่างเช่น Pipelining คือ การที่มีส่วนหนึ่ง ของชุดคำสั่ง ที่ทำงานพร้อม ๆ กันใน function unit หลาย ๆ อันที่แตกต่างกัน

สำหรับการใช้สถาปัตยกรรมเหล่านี้ สำหรับ Parallelism แบบที่เป็น Coarse grain นั้น เราจะใช้งานมันในลักษณะที่เป็น Throughput Computing หรือ Capacity Computing นอกจากนั้นเรายังสามารถใช้งานมันได้ในลักษณะของ Cooperative Computing ซึ่งเป็นลักษณะ ที่ application มีการปฏิสัมพันธ์กันซึ่งก็ยังเป็นในระดับ Coarse grain และ นอกจากนั้นก็มีอีก แนวทางหนึ่งคือ แบบ Shared Memory Programming

ในเรื่องของประสิทธิภาพนั้นเราจะพูดถึงเรื่อง Throughput หมายถึง จำนวนของ Operations ทั้งหมดที่เราสามารถ execute ได้ในหนึ่งหน่วยเวลา และต่อไปเราจะพูดถึงรูปแบบ ของ Performance ที่ยกขึ้นที่จะทำคือ การทำอย่างไรที่จะลดเวลา execution time ของปัญหา ที่มีขนาดของปัญหาคงที่ และเวลา execution time จะลดลงอย่างไร เมื่อเราเพิ่มจำนวน Computing Resource

ในวันนี้เราจะพูดถึงแนวทาง 3 แนวทางที่จะ Model การประมวลผลที่แตกต่างกัน นั้นคือ Throughput Computing และ Capacity Computing และ Cooperative Computing (ซึ่งใช้ หลักการส่วนหนึ่งจาก Capacity Computing แต่เน้นไปที่การทำงานของ Application เพียง Application เดียว)

นอกจากนั้นเราจะพูดถึงหลักการ Weak Scaling ซึ่งเป็นการเพิ่มขนาดของ Computing resources ไปพร้อม ๆ กันกับการเพิ่มขนาดของปัญหา

ดังนั้นในบทนี้เราจะมาศึกษาชนิดของ Architecture แบบหนึ่ง และในขณะเดียวกันเราจะ

พิจารณาว่า โมเดลต่าง ๆ ที่ว่ามานั้นมันเกี่ยวข้องกับ Architecture ชนิดนี้อย่างไร ซึ่งชนิดของ Architecture ที่ว่านี้ก็คือ Commodity Cluster ก่อนอื่น มองจะพูดเล็กน้อยเกี่ยวกับความเข้าใจว่า อะไร เป็นสิ่งที่ทำให้ Commodity Cluster สำคัญและมีประสิทธิภาพ และเป็นที่ใช้กันอย่างแพร่หลายในปัจจุบัน

เราจะพูดถึงประวัติของมัน และหลังจากนั้นจะพูดถึงโครงสร้างของระบบแบบนี้ ไม่ใช่เพียงแค่โครงสร้างทาง Hardware เมื่อนอกบุคลากรที่แล้ว แต่เราจะพูดถึงโครงสร้างทาง Software เราจะพูดถึง Stack ของ layer ต่าง ๆ ที่เกี่ยวข้องตั้งแต่ Hardware register ไปจนถึง Application เราจะแยก layer ออกมาและพิจารณาแต่ละอัน และหลังจากนั้นเราจะมีการ Demo คุณจะได้เห็นการเข้าใช้งานระบบ Cluster ที่ LSU ซึ่งเป็นระบบที่มีชื่อว่า เชลอริทั่ส

3.1 แนะนำระบบ Commodity Cluster

ระบบ Commodity Cluster คืออะไร จริง ๆ แล้วเราสามารถเปรียบมันเหมือนกลุ่มของผลไม้ที่ห้อยจากต้นอยู่ในระดับต่ำ ๆ Commodity Cluster คือ ระบบ Distributed System ซึ่งประกอบไปด้วยระบบย่อย ซึ่งแต่ละส่วนนั้นเป็นระบบคอมพิวเตอร์ที่มีความเป็นเอกเทศของตนเอง และที่สำคัญระบบคอมพิวเตอร์เหล่านี้มีราคาถูกและหาซื้อได้ทั่วไป เหตุผลนี้เป็นตัวผลักดันให้เกิดการใช้งานอย่างแพร่หลาย หรือ Economy of Scale ขึ้น

ถึงแม้ว่า Supercomputer จะมีประสิทธิภาพมากและสำคัญ แต่ว่าติดตั้งค่อนข้างเล็ก และมีค่าใช้จ่ายในการพัฒนาระบบที่แพง และสูงมาก

ในขณะที่ Commodity Cluster นั้นส่วนมากมาจากส่วนประกอบต่าง ๆ ที่มีของ hardware ที่ใช้กับ PC และ Workstation ซึ่งมีขนาดค่อนข้างใหญ่นำมาประกอบกันด้วยวิธีการทาง hardware และ Software ที่เราจะพูดถึงในบทนี้ แล้วทำให้เราได้ประสิทธิภาพของการทำงานที่สูงขึ้น โดยที่ส่วนใหญ่แล้วจะเป็นในเรื่องของ Throughput และในบางระดับในเรื่องของ Response Time ทำให้เราได้เครื่องที่มีประสิทธิภาพสูงในราคาน้ำเงินมากเกินไป เราจะพูดถึง Component หลักสองอย่างของ Commodity Cluster ได้แก่ Node ซึ่งก็คือ เครื่องคอมพิวเตอร์ที่ประมวลผลได้ด้วยตัวของมันเอง ส่วนที่ 2 ก็คือ Network ซึ่งเป็น System Area Network หรือ System Interconnect ที่เชื่อมต่อ Node หลาย ๆ Node เข้าด้วยกัน นอกจากนั้นยังมีสิ่งอื่นด้วย อย่างเช่น Secondary Storage ซึ่งเชื่อมต่อกับ Local Area Network เป็นต้น แต่ส่วนใหญ่แล้วเราจะพูดถึง 2 ส่วนใหญ่นี้



ภาพ 3-1

ภาพ 3-1 เป็นภาพของ Linux Cluster รุ่นที่ 4 หรือ แบบ Beowulf Cluster ซึ่ง Thomas Sterling เป็นผู้พัฒนาขึ้น เขารายงานปี 1997 หรือ 98 เครื่องนี้มีประสิทธิภาพประมาณ 10 Gigaflops ซึ่งเล็กน้อยมากเมื่อเทียบกับประสิทธิภาพของเครื่องคอมพิวเตอร์ในปัจจุบัน แต่ประสิทธิภาพของมันก็ยังเป็นประสิทธิภาพที่ค่อนข้างสูงในช่วงนั้น

Draft

Draft

Draft

Draft



ภาพ 3-2

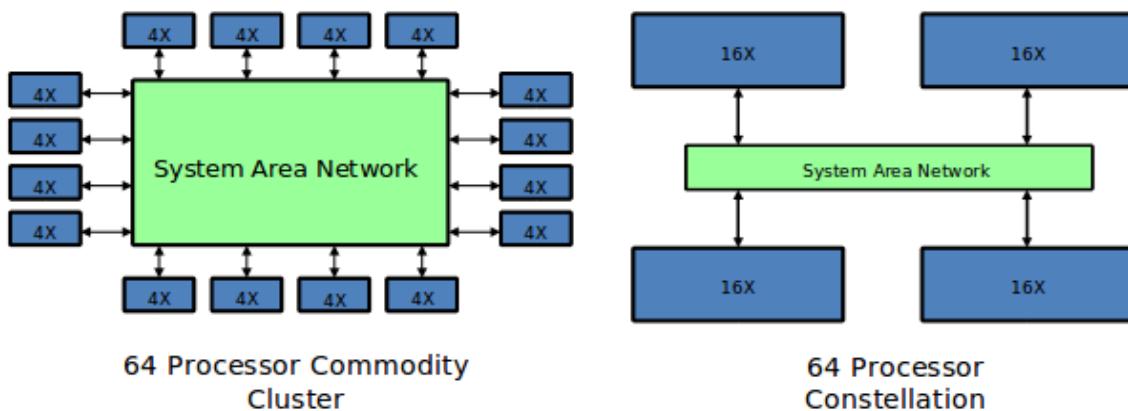
ตัวอย่างของเครื่องคลัสดเตอร์ในภาพ 3-2 คือเครื่อง TSUBAME ผู้ได้ถ่ายภาพกับคุณ Satoshi ผู้ออกแบบเครื่องนี้ตอนที่ผมได้ไปเยี่ยมชมเมื่อเดือนที่แล้วที่ Tokyo Institute of Technology ภาพข้างล่างคือเครื่อง TSUBAME นั้น เป็นเครื่องที่ใช้ Clear Speed Accelerator หลายร้อยอัน เป็นเครื่องที่เร็วที่สุดในญี่ปุ่น ในปัจจุบัน และมีประสิทธิภาพประมาณ 100 Teraflops เครื่องนี้เป็นอีกด้าวย่างหนึ่งของ Commodity Cluster



ภาพ 3-3

ตัวอย่างถัดไปเป็นภาพของของเครื่อง Cluster อีกเครื่องหนึ่งคือเครื่อง Marenostrum

ของประเทคโนโลยี เป็นเครื่องระดับกลางด้วยประสิทธิภาพกว่า 30 Teraflops เครื่องนี้ใช้ hardware และ Software คล้ายกันกับที่เราใช้ในวิชานี้ ผู้คิดว่าที่นี่มีสถานที่ติดตั้งเครื่อง Cluster ที่สวยที่สุด เพราะติดตั้งอยู่ในโบส์ต์ และตัว Cluster เองก็ตั้งอยู่ในกล่องกระจายกลาง โบส์ต์



ภาพ 3-4

ระบบ Commodity Cluster นั้นมีอยู่หลายแบบ ผู้จะพูดถึงแบบหนึ่งผู้เคยบอกว่า Parallelism นั้นคือจำนวน Node ใน Cluster แต่ในแต่ละ Node ก็มี Parallelism ด้วย และเรา จะเห็นถัดไปว่า อาจมี Processor จำนวนมากอยู่ใน node หนึ่ง ๆ จริง ๆ และ มีระบบที่ประกอบไปด้วย Node จำนวนหนึ่ง แต่ปรากฏว่า จำนวน Processors ในแต่ละ Node นั้นกลับมีมากกว่า จำนวน Node ด้วยข้อความสมดุลของ Parallelism ในรูปแบบนี้นั้นต่างจากที่เราคุ้นเคยกัน แจ็ก ดองการร่า จากศูนย์วิจัย ORNL เรียกเครื่องคอมพิวเตอร์แบบนี้ว่า Constellation (ภาพ 3-4)



ภาพ 3-5

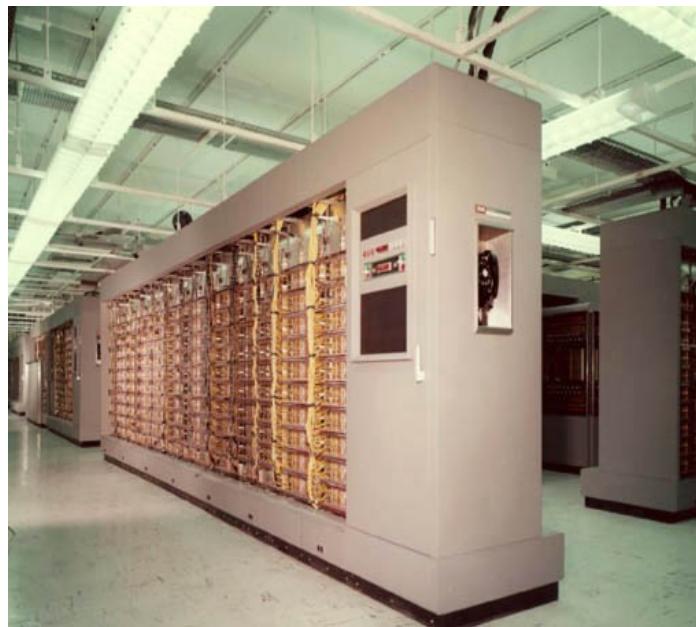
ภาพ 3-5 แสดงตัวอย่างของระบบเรียกว่า Constellation เป็นเครื่องที่ชื่อว่า Columbia ตั้งอยู่ที่ NASA Ames research center ในคалиฟอร์เนีย มันเป็นเครื่องชนิด SGI Altrix มีจำนวน Node 20 Node และแต่ละ Node มี Processors จำนวน 512 processors (เป็น Intel Itanium-2) เครื่องนี้ถูกนำมาใช้ในการศึกษาข้อมูลเกี่ยวกับการระเบิดของกระสวยอวกาศ Columbia และคำนวณหารือการแก้ไขข้อผิดพลาดเหล่านั้น

3.2 ประวัติของระบบคลัสเตอร์

Cluster ไม่ใช่สิ่งใหม่จริง ๆ แล้วคอมพิวเตอร์รุ่นแรก ๆ ก็เป็นเครื่อง Cluster เท่าที่ผมทราบ เครื่องแรกนั้นสร้างในปี 1957 จากที่บอกในความแรกว่าเครื่อง Whirlwind เป็นเครื่องคอมพิวเตอร์ที่ได้รับการพัฒนาขึ้นที่ MIT เพื่อรับระบบ flight Simulator และมีประสิทธิภาพประมาณ 5000 operation per sec และใช้เทคโนโลยี Core Memory (ซึ่ง IBM ซื้อลิขสิทธิ์ไป) เครื่อง Whirlwind นี้ถูกนำมาใช้เป็น พื้นฐานในการสร้างระบบ Cluster ระบบแรก ของโลก ประกอบไปด้วยเครื่องคอมพิวเตอร์ที่ใช้หลอดศูนยากาศ หลายเครื่อง ใช้สำหรับการป้องกันภัยทางอากาศ เครื่องนี้ได้รับการติดตั้งใช้งานในปี 1957 ที่ NORAD มันเป็นเครื่องคอมพิวเตอร์ที่ใช้ระบบหลอดศูนยากาศเป็นเครื่องสุดท้าย แต่ถูกใช้งานมากกว่า 20 ปี และเลิกใช้งานในปี 1982 นับว่าคุ้มค่ามาก



ภาพ 3-6 <http://museum.mit.edu/150/entries/1344>



ภาพ 3-7 <http://reference.findtarget.com/search/Semi%20Automatic%20Ground%20Environment/>

คำว่า Cluster นั้นถูกใช้เป็นครั้งแรกในโครงการชื่อว่า M31 โดยบริษัท DEC ซึ่งในอดีตเคยเป็นบริษัทที่ผลิตคอมพิวเตอร์เป็นอันดับสองของโลก แต่ไม่มีอยู่แล้วในปัจจุบัน เพราะถูกซื้อโดยบริษัท Compaq ซึ่งถูก Take over โดยบริษัท HP อีกต่อหนึ่ง เครื่อง M31 นั้นมี processor VAX 11/750 32 processors เป็นเครื่องแบบ minicomputer

ในช่วงปี 1990 Software ที่แพร่หลายที่สุดสำหรับออกแบบโครงสร้างให้ระบบคอมพิวเตอร์ที่มี Processor หลาย Processor ทำงานร่วมกัน ที่มีชื่อเสียงมากก็ได้แก่ระบบ PVM ซึ่งเปิดให้คนใช้ได้ทั่ว ๆ ไป

ในปี 1993 มี Project อよ' 2 project ที่นำสู่ Project หนึ่งคือ NOW หรือ Networks of Workstations Project จาก UC Berkeley และอีก Project หนึ่งคือ Beowulf project ที่ Dr. Thomas Sterling เป็นผู้ริเริ่มขึ้นขณะที่ทำงานอยู่ที่ NASA's Goddard Space Flight Centre โดยความร่วมมือกับ University of Maryland



UC-Berkeley NOW Project

- NOW-1 1995
- 32-40 SparcStation 10s and 20s
- originally ATM
- first large myrinet network



- NOW-2 1997
- 100+ Ultra Sparc 170s
- 128 MB, 2 2GB disks, ethernet, myrinet
- largest Myrinet configuration in the world
- First cluster on the TOP500 list



CSC

Department of Computer Science
Louisiana State University

CSC 7600 Lecture 3 : Commodity Clusters,
Fall 2009

12

ภาพ 3-8

ในกลางปี 1993 ระบบคอมพิวเตอร์แบบ NOW ก็ได้รับการพัฒนาขึ้นที่ UC Berkeley และในกลางปี 1994 ระบบคอมพิวเตอร์แบบ Beowulf Cluster รุ่นแรกก็เกิดขึ้นตามมา ในปัจจุบันระบบ Beowulf Cluster หรือ Linux Cluster นั้นได้รับการยอมรับจากผู้ผลิตส่วนใหญ่ และได้รับรางวัล Gordon Bell ซึ่งเป็นรางวัลที่มีเกียรติอันเป็นการแสดงให้เห็นคุณค่าของมัน และในปีเดียวกันนั้นระบบ Berkley NOW ก็เป็นระบบ Cluster ระบบแรกที่มีประสิทธิภาพติด อันดับ Top 500 list ด้วย ทั้งหมดนี้คือจุดเริ่มต้นของการที่ระบบ Cluster ได้กลายเป็นระบบที่ใช้งานกันมากที่สุดใน การประมวลผล สมรรถภาพสูงในปัจจุบัน

ระบบ NOW และ ระบบ Cluster นั้นมีความแตกต่างกัน NOW นั้นมุ่งสร้างระบบที่มี ความสามารถสูง และมีราคาสูง ดังนั้น เขาจึงใช้ Workstations (ในขณะนั้นเครื่องแบบ Workstation กับ PC นั้นไม่เหมือนกัน แต่ในปัจจุบันจะเหมือนกัน) NOW ใช้ Workstation ราคาแพง และ ใช้ระบบ Network ที่มีราคาแพงด้วยคือระบบ Myrinet



NASA Beowulf Project



- | | | |
|---------------------------|----------------------------------|-------------------------------------|
| • Wiglaf - 1994 | • Hrothgar - 1995 | • Hyglac-1996 (Caltech) |
| • 16 Intel 80486 100 MHz | • 16 Intel Pentium 100 MHz | • 16 Pentium Pro 200 MHz |
| • VESA Local bus | • PCI | • PCI |
| • 256 Mbytes memory | • 1 Gbyte memory | • 2 Gbytes memory |
| • 6.4 Gbytes of disk | • 6.4 Gbytes of disk | • 49.6 Gbytes of disk |
| • Dual 10 base-T Ethernet | • 100 base-T Fast Ethernet (hub) | • 100 base-T Fast Ethernet (switch) |
| • 72 Mflops sustained | • 240 Mflops sustained | • 1.25 Gflops sustained |
| • \$40K | • \$46K | • \$50K |



Department of Computer Science
Louisiana State University

CSC 7600 Lecture 3 : Commodity Clusters,
Fall 2000

14

Draft

ภาพ 3-9

ในทางตรงกันข้าม โครงการ Beowulf ใช้ PC ที่ถูกทิ划สุดเท่าที่จะเป็นไปได้ และใช้ Network ที่ถูกทิ划สุดด้วย ในภาพนี้คุณจะเห็นเครื่อง Beowulf 3 รุ่น สร้างขึ้นในปี 1994 1995 และ 1996 ตามลำดับ ถ้าดูที่ความแตกต่างคุณจะเห็นว่า Clock rate ของเครื่องในปี 1994 และ 1995 นั้นมีประมาณ 100 MHz ซึ่งน้อยกว่าของเครื่อง laptop ของคุณถึงประมาณ 20 ถึง 30 เท่า ในภาพ 3-9 เครื่องทางขวาของคุณ คือเครื่อง hyglac ซึ่งมีความเร็ว 100 Mbps เป็น Non-blocking switch ซึ่งราคาในตอนนั้น น้อยกว่า 5000 US Dollars เพียงเล็กน้อยเท่านั้น และเป็นเพียงแค่ Fast Ethernet เท่านั้นด้วย

เราใช้เครื่องนี้สำหรับการคำนวณทางวิทยาศาสตร์ที่ Cal Tech และ JPL เป็นเวลาถึง 5 ปี โดยที่มันมี Clock rate เท่ากับ 200 MHz ผมจามไม่ได้ว่าแต่ละเครื่องมี Memory เท่าไร แต่อย่างไรก็ตามมันสามารถทำงานได้ด้วยประสิทธิภาพถึงระดับ Gigaflops โดยการรัน Applications จริง ๆ เพื่อแก้ปัญหาแบบ N-Body และราคาโดยรวมของมันก็คือประมาณ 5000 US Dollars ซึ่งเมื่อเทียบกับประสิทธิภาพที่ได้ในขณะนั้นแล้ว ถือได้ว่าเป็นพัฒนาการที่สำคัญมากอันหนึ่ง

นิตยสาร Science ได้ถ้าพิมพ์เรื่องของระบบนี้ และออกขายในการประชุม Super Computing Conference ซึ่งก็น่าสนใจมากนะครับ เพราะว่าในขณะที่ Conference นั้นมีแต่บริษัทใหญ่ ๆ ออกแสดงเครื่องคอมพิวเตอร์ของตน หนังสือนิตยสารฉบับนี้กลับตีพิมพ์เรื่อง เกี่ยว

กับเครื่องที่ประกอบไปด้วย PC ราคาถูก ๆ อยู่ก่อนหนึ่ง และเรียกเครื่องแบบนี้ว่า “Super Computer” ที่คุณสร้างได้ด้วยตนเอง”

ในปัจจุบัน ระบบ Cluster เป็นระบบที่ใช้งานมากที่สุดในโลกของ Super Computer บริษัทคอมพิวเตอร์ใหญ่ ๆ แทบทุกบริษัทมีผลิตภัณฑ์ Cluster Super computer และมีบางบริษัทเกิดขึ้นเพื่อสร้างระบบ Cluster โดยเฉพาะ ผู้รวมบริษัท Dell เป็นบริษัทในกลุ่มนั้น เพราะ HPC product ของ Dell มีอยู่อย่างเดียวคือ Commodity Cluster

จากประวัติศาสตร์ของระบบคลัสเตอร์ จะเห็นได้ว่า Cluster เริ่มมีอยู่ใน TOP 500 list (ลิสของระบบคอมพิวเตอร์ที่เร็วที่สุด 500 อันดับแรก) ตั้งแต่ปี 97 และในปัจจุบันกว่า 70% ของเครื่องใน list นี้ก็เป็น Cluster มีอยู่ช่วงหนึ่งในอดีตที่มีระบบแบบ Constellation มากพอสมควร แต่ในปัจจุบันแทบไม่มีแล้ว ผู้คนคาดเดาไว้ว่า Constellation จะกลับมาอีกครั้งหนึ่ง เพราะ Multicore Technology ต่อไปคุณอาจมี 1000 core ใน 1 node แต่คุณอาจไม่มี Node ถึง 1000 node ก็ได้ซึ่งนั่นก็คือ Constellation

Cluster นั้นกล้ายเป็นระบบคอมพิวเตอร์สมรรถนะสูงที่ได้รับการนำไปใช้งานอย่างแพร่หลายเหตุผลหนึ่งเพราะมันค่อนข้างจะ Cost-effective คือ ราคาไม่แพง เมื่อเทียบกับ Throughput ที่ได้และเมื่อเทียบกับสถาปัตยกรรม Super computer แบบอื่น ๆ ส่วนใหญ่

แต่ก็มีเหตุผลอย่างอื่นด้วย นั่นคือระบบคลัสเตอร์นั้นมีความ flexibility ค่อนข้างมาก เพราะ hardware ที่นำมาใช้สร้างระบบคลัสเตอร์นั้นเป็น hardware ที่ใช้กับ PC ทั่วไปและหาได้ตามท้องตลาด การสร้างหรือซ่อมแซมระบบคลัสเตอร์ไม่ต้องซื้อนายกับบริษัทผู้ผลิตกลุ่มใดกลุ่มหนึ่งที่จะเป็นผู้กำหนดว่าระบบจะต้องมี hardware พิเศษอย่างไร คุณมีทางเลือกที่ค่อนข้างมาก มีคำพูดหนึ่งที่หลายคนพูดคือ “ป้อยครั้งที่เทคโนโลยีที่ดีที่สุดนั้นเกิดมาจากผู้ผลิตภัณฑ์ที่ถูกที่สุด” ทำไมหรือ? เพราะตลาดที่ใหญ่ที่สุดนั้นอยู่ที่นี่นั่น ถ้าคุณต้องการทำเงินจากผู้ผลิตภัณฑ์อะไรก็ตามที่คุณจะลงทุนสร้างขึ้น สมมุติว่าใช้เงินหนึ่งพันล้าน US Dollars หลังจากผลิตมาแล้ว คุณก็จะต้องพยายามขายมันให้หมดให้มากที่สุดเท่าที่จะมากได้ใช่ไหมครับเพื่อให้ได้ค่าตอบแทนและกำไรจากการผลิตสิ่งนั้น ยกตัวอย่างเช่น Processor Chip ที่มีความสามารถสูงสุดในวันนี้มีความสามารถประมาณ $\frac{1}{4}$ Teraflops (พันล้านล้าน flops) ซึ่งก็ได้แก่ IBM Cell architecture ซึ่งถูกออกแบบมาสำหรับใช้ใน Play Station 3 เพื่อการเล่นเกมส์เป็นจุดประสงค์หลัก ไม่ได้ถูกสร้างมาเพื่อใช้สร้าง คอมพิวเตอร์ IBM Blue Gene (ที่ประกอบไปด้วย Cell Processor หลายหมื่นอัน) ซึ่งเป็นคอมพิวเตอร์เครื่องหนึ่งที่เร็วที่สุดในโลก ซึ่งก็เป็นตัวอย่างที่ชัดเจน ของเทคโนโลยีที่ดีที่สุดต่อราค่าต่ำที่สุด

นอกจากนั้นก็ยังมีเหตุผลอื่นที่บังช้อนกันว่านั่นคือ การใช้งานระบบคลัสเตอร์นั้นทำให้อำนาจในการตัดสินใจขึ้นอยู่กับผู้ใช้ เพราะผู้ใช้เป็นผู้กำหนดเองว่าเขาต้องการอะไร ไม่ต้องให้ Vendor มาเป็นผู้กำหนดให้ ผู้ใช้เป็นผู้ตัดสินใจว่าจะเอา hardware แบบใดหรือ Software อะไร

บางครั้งมันทำงานได้อย่างมีประสิทธิภาพ

ในปี 1999 Professor Thomas Sterling ได้เขียนหนังสือชื่อว่า “การสร้าง Beowulf Cluster” พิมพ์โดยสำนักพิมพ์ MIT ซึ่งอธิบายการสร้างระบบคลัสเตอร์ขึ้นด้วยตนเอง

3.3 คุณลักษณะของระบบคลัสเตอร์

เหตุผลที่ lecture นี้สำคัญ ก็ เพราะในความคุณจะต้องไปใช้เครื่องแบบนี้จริง ๆ มีบางสิ่งที่คุณจะต้องรู้เกี่ยวกับ Commodity Cluster อย่างแรกคือคุณจำเป็นต้องรู้จักส่วนประกอบของมัน ได้แก่ โหนดหรือคอมพิวเตอร์แต่ละเครื่อง และการเชื่อมต่อโหนดต่างๆ หรือ Interconnection Network และ Software บนแต่ละโหนด ที่เราจะพูดถึงมันต่อไป หลังจากนั้นคุณจำเป็นต้องเข้าใจ Programming Model ที่คุณจะใช้สำหรับเขียนโปรแกรม และรันโปรแกรมของคุณ ยิ่งไปกว่านั้นคุณยังต้องรู้จัก Performance Matrices ที่เราจะใช้ในการวัดประสิทธิภาพ และเครื่องมือที่เราสามารถใช้ในการวัดประสิทธิภาพดังกล่าวเพื่อช่วยให้คุณวิเคราะห์ว่าระบบ Scale หรือไม่อย่างไร และมีประสิทธิภาพหรือไม่อย่างไร

มีพารามิเตอร์หลายอย่างที่เราใช้เป็นตัวกำหนดความสามารถของระบบคลัสเตอร์ ยกตัวอย่างเช่น ในระบบ Network ที่เชื่อมต่อโหนดทั้งหมดเข้าด้วยกัน ค่า Bisection bandwidth (ค่าจำนวน link ใน Network Topology ที่คุณตัดเมื่อแบ่งระบบเป็นสองส่วนเหมือน ๆ กัน) เป็นค่า bandwidth ที่ต่ำที่สุดที่คุณสามารถส่งข้อมูลจากฝั่งหนึ่งไปยังอีกฝั่งหนึ่งข้ามระบบคลัสเตอร์ได้ พารามิเตอร์ที่บ่งบอกคุณลักษณะของระบบเช่นนี้เป็นสิ่งที่คุณจำเป็นต้องเข้าใจ เพราะมันจะมีผลกับการสร้างโปรแกรมของคุณ

นอกจากนั้นคุณจำเป็นต้องรู้ว่าจำนวน Node ที่ระบบมีและจำนวน Processor ในแต่ละ Node เป็นจำนวนเท่าไร เพราะคุณจะได้รู้ว่าคุณสามารถสร้าง Thread การทำงานในโปรแกรมเป็นจำนวนกี่ Thread ที่สามารถทำงานไปพร้อม ๆ กัน ในกรณีที่ระบบคลัสเตอร์มี Accellerator Hardware หรือ GPU จำนวน Thread ของการทำงานก็จะมีมากขึ้นซึ่งเราจำเป็นต้องเข้าใจการใช้งานมันด้วย

โดยทั่วไปแล้วระบบคลัสเตอร์ที่ใช้กันส่วนใหญ่นั้นค่อนข้างเล็ก เราอาจไม่จำเป็นต้องกังวลเกี่ยวกับพื้นที่ติดตั้งระบบหรือพลังงานไฟฟ้าที่ใช้ แต่สำหรับระบบคลัสเตอร์ขนาดใหญ่ ๆ นั้น การสร้างระบบเหล่านี้ต้องคำนึงถึงเรื่องสถานที่ตั้ง และการใช้พลังงานด้วยเป็นอย่างยิ่งเลยที่เดียว

สิ่งที่ทำให้ระบบคลัสเตอร์ทำงานได้เร็วกว่าคอมพิวเตอร์ทั่วไปก็คือระดับของ Parallelism ของมัน ซึ่งเมื่อพิจารณาดูที่ระบบเหล่านี้แล้ว Parallelism อยู่ที่ไหน

- แน่นอนเนื่องจากมันเป็น Cluster ดังนั้น Parallelism ก็คือจำนวน Node
- แต่ในแต่ละ Node มันก็จะมีจำนวนของ Socket ของ Processor จำนวนหนึ่ง ซึ่งนั่นก็

เป็น Parallelism

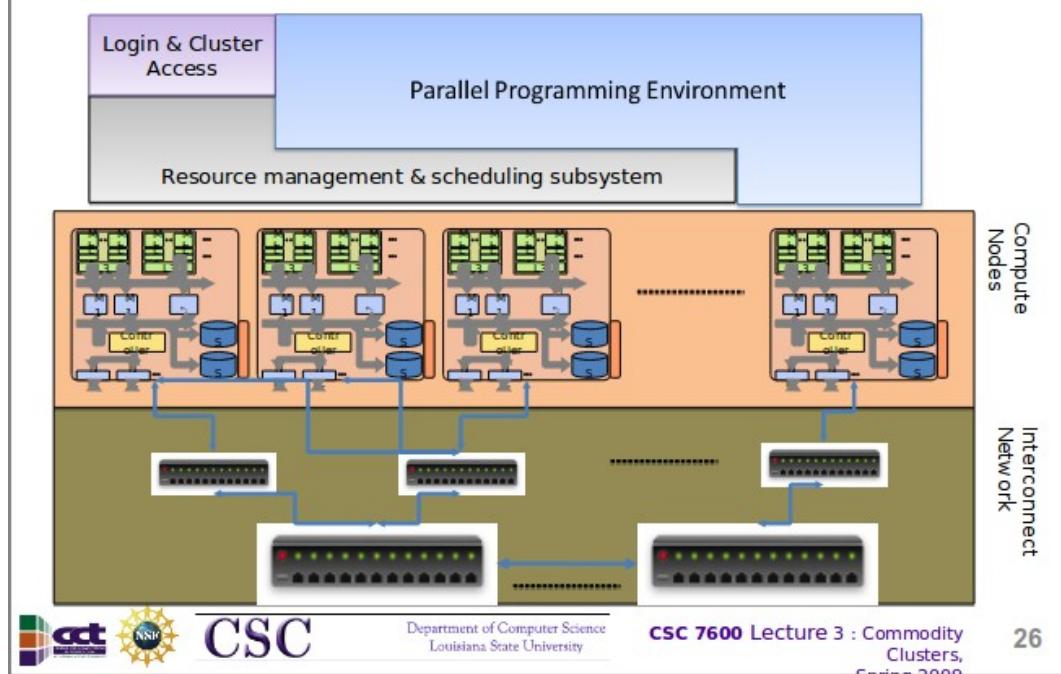
- ในอีกอันดับหนึ่งแต่ในแต่ละ Socket ในปัจจุบันก็จะมีอย่างน้อย ๆ 2 Core และนั่นก็เป็น Parallelism ในระดับที่ 3
- และในแต่ละ Core เองก็มี Parallelism อญี่อีกไม่ว่าจะเป็น Instruction Level Parallelism และการใช้ ALU หลาย ๆ อัน การทำงานของ Pipeline Execution Unit และ Pipeline Floating Point Unit ซึ่งทั้งหมดนี้อยู่ใน Core เหล่านั้น
- นอกจากนั้น ยังมีระบบ pre-fetching ซึ่งเป็นการประมวลผลที่คุณไม่ได้ควบคุมมันแต่เป็น ระบบที่ Overlap Computation และการติดต่อสื่อสารกัน Main Memory ในบางครั้งการทำงานหลายสถานการณ์ใน processor ก็มี latency ที่แตกต่างกัน การทำงาน Pre-fetching ก็จะมีอุปกรณ์ที่จะทำการประมวลผลคำสั่งต่างๆโดยจัดให้มันทำงานไปพร้อม ๆ กัน หรือในอันดับที่แตกต่างจากลำดับการทำงานตามปกติ
- และยิ่งไปกว่านั้น คุณยังสามารถเพิ่ม Parallelism ให้มากขึ้นโดยการใช้ Accelerator หรือ GPU ในแต่ละหน่วยได้ด้วย

3.4 Cluster System

ภาพ 3-10 เป็น Diagram ที่แสดงโครงสร้างของระบบ Cluster ครอบคลุมทั้ง hardware และ Software ในส่วนล่างสุดคุณจะเห็น interconnection Network ที่เราจะพูดถึงมันต่อไป ในส่วนของการ Processing ก็จะมี Set ของ Computer Node ซึ่งในภาพได้แสดงโครงสร้างในแต่ละ Node ด้วย เห็นอีกนั้นขึ้นไปก็เป็น Interface ของระบบที่มีไว้ให้ผู้ใช้งานระบบหรือสร้างโปรแกรมเพื่อใช้งานระบบ คุณจะเห็นว่า Programming Environment ในภาพประกอบไปด้วย Tools ต่างๆที่สนับสนุนการสร้างโปรแกรมบนระบบ Cluster อย่างน้อยๆ 3 วิธี และสุดท้ายในแต่ละ Node ในระบบคลัสเตอร์จะมีซอฟแวร์สำหรับจัดการ Resource Scheduling และ Resource Management ด้วย



Cluster System



ภาพ 3-10
จากภาพสำหรับ Hardware ของระบบคลัสเตอร์นั้นประกอบไปด้วย Node และ Network นอกจากระบบที่มีส่วนสำคัญที่ใช้ติดต่อกับโลกภายนอกนั่นก็คือ External I/O และ File System

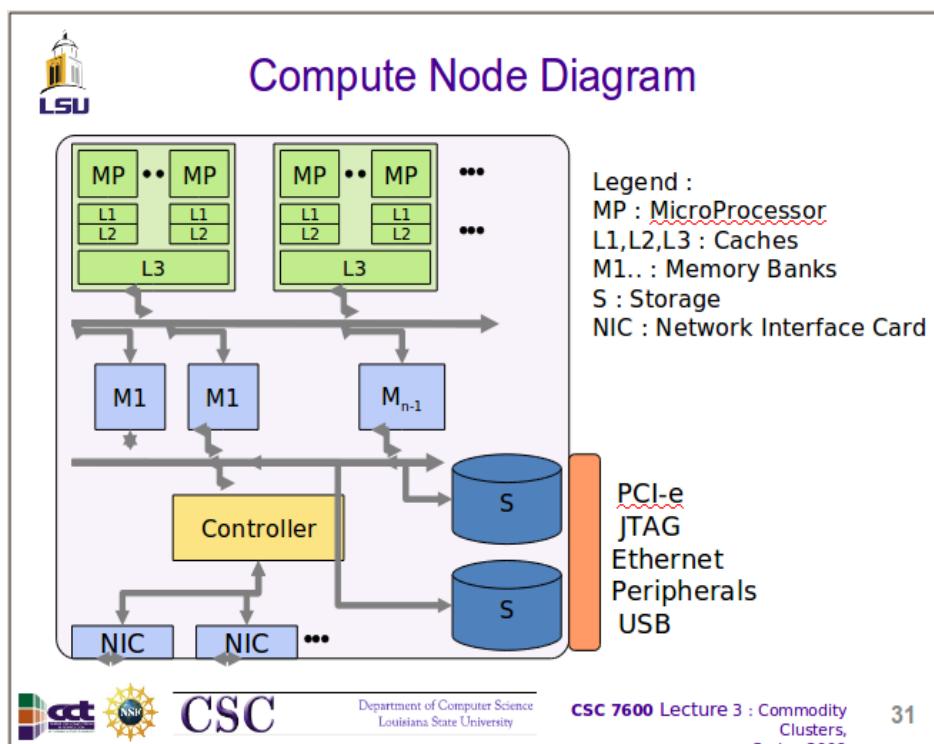
3.4.1 ระบบ Hardware และ Software ของคลัสเตอร์โน๊ต

ความสามารถของโน๊ตบนระบบคลัสเตอร์นั้นขึ้นอยู่กับความสามารถของ Processor ที่ใช้ ในปัจจุบันเครื่อง Laptop ของคุณมี Clock rate อย่างน้อย ๆ 10 เท่าของเครื่องคอมพิวเตอร์ที่เราใช้เป็น Cluster node ในยุคแรกๆ (ที่มี Clock rate ประมาณ 200 MHz) ปัจจุบันเรามี Clock rate ที่มากกว่า 3 GHz แล้ว ซึ่งนั่นก็เป็นการเพิ่มขึ้นเป็นอย่างมากในช่วงระยะเวลา 10 ปี

แต่เมื่อเราพิจารณาการเติบโตที่เป็นไปอย่างต่อเนื่องของเทคโนโลยีเราจะเห็น trend หรือแนวโน้มที่เป็นปัญหา การที่ clock rate สูงขึ้นนั้นเกิดมาจากการสามารถในการเพิ่มจำนวน Transistor ลงบน Processor Chip จำนวน transistor ที่เรารับรู้ในพื้นที่ 1 Unit ของ Chip นั้นเพิ่มขึ้นเป็นอย่างมากในแต่ละปี ซึ่งก็เป็นไปตามกฎของ Moore

แต่อย่างไรก็ตามถึงแม้ว่าเราจะสามารถเพิ่มความสามารถทาง hardware ของคอมพิวเตอร์เครื่องเดียวได้เช่นนั้น เรา ก็ยังไม่สามารถที่จะใช้ประโยชน์จากมัน หรือใช้งานมันให้ได้ประสิทธิภาพเพิ่มขึ้นแบบ exponential ในรูปแบบเดียวกัน ยกตัวอย่างเช่น เราพบว่าหลังจาก

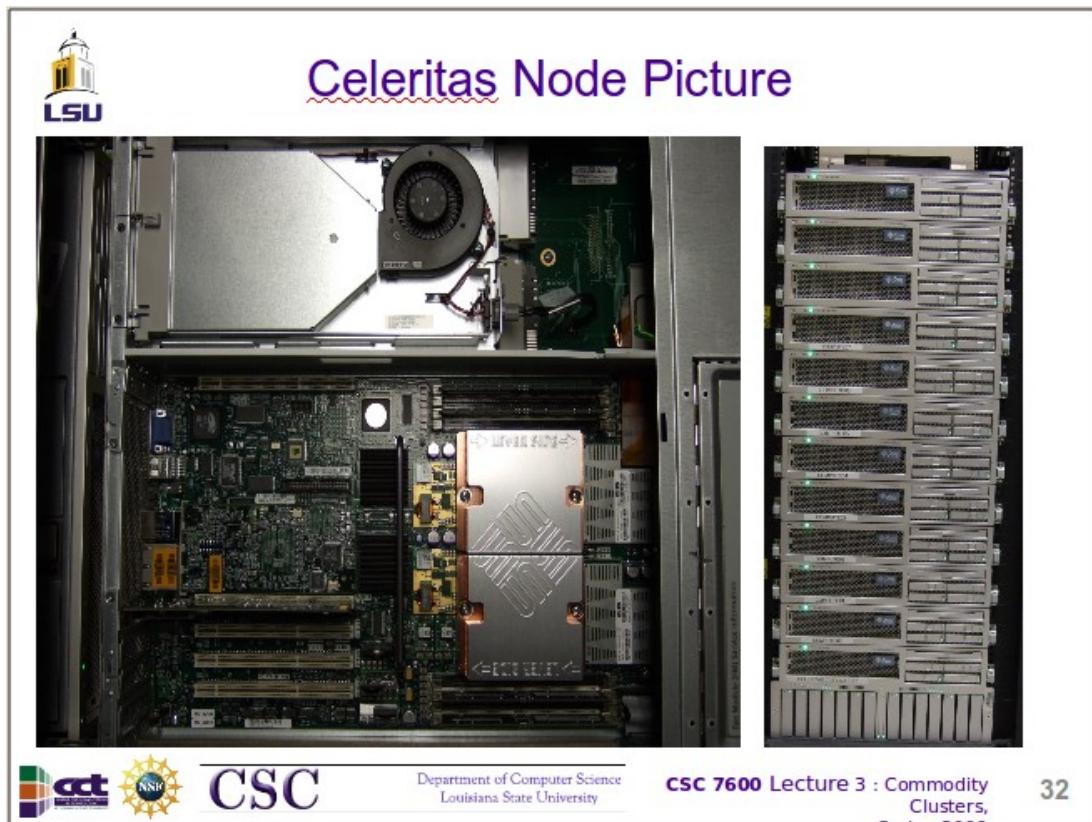
การเพิ่มขึ้นของ Clock rate อย่างรวดเร็วใน 10 ปีที่ผ่านมา ตอนนี้มันเริ่มจะอยู่คงที่ เหตุผลส่วนหนึ่งก็เนื่องมาจาก พลังงานที่ใช้ต่อ 1 processor จะเป็นประมาณ 150 Watt ต่อ Clock rate ปกติ และนั่นก็เป็นพลังงานที่ใช้สำหรับ Chip แค่นั้น ไม่ว่าจะมีร่วมที่ต้องใช้สำหรับ Memory หรือ Communication ดังนั้นระดับพลังงานที่ Processor ใช้อยู่ในปัจจุบันก็ถือว่ามากอยู่แล้ว เราไม่สามารถจ่ายค่าพลังงาน ซึ่งเป็นภาระในการใช้งานระบบคอมพิวเตอร์ได้มากไปกว่านี้ ในขณะเดียวกันจำนวนของ instructions ที่ Processor ทำงานได้ต่อ 1 Clock Cycle จำนวนของ Operations ต่อ Clock cycle นั้นไม่เพิ่มขึ้น



ภาพ 3-11 แสดงแผนภาพโครงสร้างของระบบ Cluster ที่เราจะใช้ในวิชานี้ องค์ประกอบหลัก ก็คือ Processor Chip ดังแสดงใน กล่องสีเขียวในภาพ ซึ่งมี Core อยู่ 2 Core และมี Memory Bank หลาย Memory Bank และในเครื่อง Cluster ที่คุณจะใช้ก็จะมี 4 memory Bank ใน 1 node ในบอร์ดจะมี Controllers ที่ควบคุม interfaces เรามักจะเรียกมันว่า North Bridge และ South Bridge Controller นอกจากนั้นก็มี Network Controller หรือ NIC ซึ่งถ้าอกสู่ Network ต่าง ๆ ซึ่งเรามี 2 Network และเราก็มี local Disk ซึ่งใน Cluster ของเรามี Disk เดียว และที่เหลือก็เป็นอุปกรณ์สำหรับติดต่อสื่อสารอื่นๆ (เช่น PCI-e, JTAG, USB ในภาพ)

ภาพ 3-12 แสดงโครงสร้างของ Cluster ทางด้านซ้ายและระบบทั้งหมดที่ดูเหมือนหอคอย แห่งการประมวลผลทางขวา ทางซ้ายคุณจะเห็นภาพ Chip ที่มี 2 Cores จำนวน 2 Chips ซึ่งจะ

มี memory Dim 2 คู่อยู่ด้านบน แผ่นการ์ดในภาพเป็น Network Interface card และ Network Interface ที่ฝังตัวในบอร์ดด้วย ในส่วนบนของภาพทางซ้ายที่มีพัดลมคือ Power Supply เมื่อพิจารณาภาพทางขวา จะเห็นว่าระบบ Cluster นี้มี 8 nodes และ head node อีก 1 node ซึ่งจะเป็นคอมพิวเตอร์คุณจะ login เข้าไปเพื่อใช้งานดหนดอื่นที่เหลือ ในส่วนล่างของระบบในภาพทางขวาจะเป็นระบบ Storage ขนาด 7 terabytes เป็นระบบ RAID 5



ภาพ 3-12

กล่องที่ใส่ cluster หรือ Chasis นั้นมีหลายแบบ ปกติจะเป็นแบบเล็กหรือ 1 U Chasis Box สำหรับใส่หน่วยที่มีลักษณะเป็น Blade คอมพิวเตอร์ แต่ในภาพที่คุณเห็นนั้นแต่ละหน่วยอยู่ใน 4 U Box ซึ่งมีขนาดใหญ่ขึ้น และทำให้เราสามารถปรับเปลี่ยน Component ตามที่ต้องการได้ง่าย

History of Linux

สำหรับ Software นั้นประกอบไปด้วยหลาย Layer มีสิ่งแปร逈สิ่งหนึ่งที่เกิดขึ้นในช่วงเริ่มต้นของ Cluster และ Beowulf นั้นก็คือ วิวัฒนาการของ Open Source Software ซึ่งก็คือ การพัฒนา Software ที่ได้รับการสนับสนุนทางการเงินจากที่ได้ที่หนึ่งแล้วให้ Software นั้น รวมทั้ง Source code ฟรีแก่สังคมซึ่งทำให้เกิดการใช้งาน Software และการ Share Software กันอย่างกว้างขวาง

ระบบ Linux ไม่ใช่ Software ชุดแรก ๆ ที่ออกแบบจากโครงการ GNU จาก MIT อย่างเช่น Emacs และ gcc ที่เราใช้งานกันในปัจจุบัน แต่ Linux มีประวัติของมันเอง โดยเดลของ OS ที่ใช้ในการประมวลผลมาก ๆ นั้น โดยมากแล้วจะเป็น OS ที่เป็นระบบ UNIX ซึ่งได้รับการพัฒนาขึ้นที่ Bell Lab โดยบริษัท AT&T และได้รับการสนับสนุนจากการ DARPA ต่อมา UC Berkeley ได้พัฒนาระบบ BSD UNIX ขึ้น ซึ่งก็ได้รับการใช้งานอย่างกว้างขวาง และกลายเป็น Open Source ไปในที่สุด การเป็น Open Source ทำให้ระบบ UNIX ได้รับการพัฒนาเป็นอย่างมากในเรื่อง Virtual Memory และ การเชื่อมต่อ Internet ซึ่งในตอนนั้นเราไม่ได้เรียกมันว่า Internet อย่างในปัจจุบัน ดังนั้น BSD จึงเป็นที่ทั้งฟรีและดี แต่โดยรายที่บริษัท AT&T คิดว่าเข้าจำเป็นที่จะต้องได้ลิขสิทธิ์ในระบบ UNIX นี้จะด้วยเหตุผลอะไรก็ไม่ทราบได้ อาจเป็นในเรื่องของกฎหมายหรือการไม่อยากมีคู่แข่งทางการค้า ก็ไม่ทราบแน่ชัด

แต่อย่างไรก็ตามมันทำให้คนที่เป็นนักวิจัยและผู้พัฒนาระบบไม่แน่ใจที่จะพัฒนา Software บนระบบ BSD เพราะว่าเรื่องทางกฎหมายเหล่านี้ ดังนั้น นักเรียนระดับบัณฑิตศึกษาที่ประเทคโนโลยีและคณหนึ่งชื่อ Linus Trovald จึงได้สร้างโปรแกรม UNIX ขนาดเล็กขึ้นมาและแจกฟรีบน Internet และเขาถูกเรียกว่า “Linux” ไปให้มาช่วยกันพัฒนามัน

ด้วยความขัดแย้งและไม่รู้ว่าจะทำอย่างไรกัน AT&T รวมทั้งความต้องการในการวิจัยและพัฒนา Software ของคนเหล่านั้น ทำให้คนหันมาใช้โปรแกรมของนาย Linus ซึ่งถูกเรียกว่า Linux กันอย่างแพร่หลาย โครงการ Beowulf ก็เป็นโครงการหนึ่งที่ได้ช่วยพัฒนาระบบ Linux เราได้ช่วยเขียน Ethernet Driver ถ้าคุณดู Linux Source Code แล้วคุณจะเห็นคำอ้างอิงมากที่ Beowulf project ด้วย นอกจากนี้เรายังได้ช่วยเขียน Driver อื่น ๆ ด้วยอีกหลายชิ้น ทีมวิจัยของผมได้ช่วยพัฒนา Ethernet Driver ดังเดิมของระบบ Linux Driver ที่ใช้ในปัจจุบันก็เป็น Driver ที่ได้รับการพัฒนาต่อยอดมาจากการของพวกราในครั้งนั้น

ในปัจจุบัน ประมาณ 15 ปีหลังจากนั้น เรายังคงพูดถึงความสัมพันธ์ระหว่าง Commercial Software และ Open Source Software อยู่เสมอ เพราะ ทั้งสองอย่างนั้นมีข้อดีทั้งคู่ ขึ้นอยู่กับว่าเราจะใช้ทั้งสองสิ่งนี้ให้ลงตัวอย่างไร

หน้าที่หลักของ OS หน้าที่หนึ่งคือการให้บริการ Interface ระหว่างผู้ใช้กับระบบคอมพิวเตอร์ และให้บริการ Service ต่างๆ และเป็นผู้จัดการ Resource ต่างๆ OS เป็นผู้ควบคุมการ execute งานในระบบ และป้องกันงานของผู้ใช้หลายคน ๆ คน ไม่ให้มาก้าวเข้ากัน มันช่วยให้การใช้งานระบบทำได้ง่าย และในบางครั้งมันก็ทำให้ระบบทำงานมีประสิทธิภาพมากขึ้น ด้วย นอกจากนั้น OS ยังให้บริการ Interface ในการเข้าถึงข้อมูลใน File Systems และบริการอื่น ๆ ก็ได้แก่ บริการตรวจสอบความผิดพลาด และการตอบสนองต่อความผิดพลาด เมื่อมีสิ่งผิดปกติเกิดขึ้นในการทำงานของระบบ เช่น การพยายามหารด้วยศูนย์ ซึ่ง OS ก็จะถูกเรียกเข้ามาทำงานเมื่อมี Exception เกิดขึ้น และมันก็ทำให้การทำงานของระบบเป็นไปได้อย่างไม่เลว

รายงานเกินไป เมื่อ Application ได ๆ ทำงานผิดพลาดขึ้น OS ก็จะพยายามจัดการกับความผิดพลาดเหล่านั้นอย่างเหมาะสมเพื่อพยายามไม่ให้ความผิดพลาดส่งผลกระทบต่อระบบโดยรวม

3.4.2 การ Programming ระบบ Cluster

computation โมเดล ในการใช้งานระบบ cluster นั้นมี 3 อย่างได้แก่ Throughput หรือ Capacity Computing, Cooperative Computing, และ Capability Computing

แบบแรกเป็นการรันงานแต่ละงานแยกจากกันและรันแต่ละงานต่างโหนดกัน โมเดลนี้เป็นโมเดลง่ายๆที่ใช้กันอย่างกว้างขวาง เราเรียกมันว่า Throughput หรือ Capacity Computing

แบบที่ 2 หรือ Cooperative Computing เป็นการแยกชิ้นงานของ Application หนึ่งออกเป็นส่วนๆแล้วกระจายการทำงานของแต่ละส่วนออกไปที่โหนดหลายๆโหนดและในระหว่างที่ Application นั้นรันชิ้นงานส่วนต่างๆของมันก็มีการประสานการทำงานกันเป็นระยะๆ

แบบสุดท้าย Capability Computing เป็นรูปแบบของการทำงานที่ชิ้นงานอยู่ติดกันมาก และมีการประสานงานกันมาก เราไม่สามารถ application แบบนี้ข้ามโหนดได้ เพราะแต่ละโหนดในระบบคลัสเตอร์เป็นเครื่องคอมพิวเตอร์ที่แยกจากกัน application ในโมเดลนี้ต้องรันบนเครื่องคอมพิวเตอร์เครื่องเดียวกันซึ่งอาจมีหลาย processors และมีการแลกเปลี่ยนข้อมูลผ่าน shared memory

Throughput Computing มีสองรูปแบบ แบบแรกคือการรันโปรแกรมต่างโปรแกรมกันบนโหนดต่างโหนดกัน โปรแกรมเหล่านี้ไม่มีอะไรเกี่ยวข้องกันเลยและอาจมาจากผู้ใช้ต่างคนกัน ก็ได้ แบบที่สองคือการรันโปรแกรมๆเดียวกัน แต่ แต่ละโปรแกรมประมวลผลข้อมูลที่แตกต่างกันและไม่มีการปฏิสัมพันธ์ระหว่างกันเลย ยกตัวอย่างเช่นวิศวกรอาจรันโปรแกรม simulation เดียวยหลายๆอันโดยที่แต่ละอันมี input parameters ที่แตกต่างกันและใช้พื้นที่ใน memory ที่แตกต่างกัน เราเรียก simulation แบบนี้ว่า parametric simulation งานเหล่านี้จะรันอยู่บนโหนดต่างๆพร้อมๆกัน

ระบบ Decoupling Work Queue ก็เป็นระบบที่ให้บริการ Throughput computing แบบหนึ่งซึ่งใช้วิธีการง่ายๆที่อนุญาตให้เรา assign jobs หลายๆ jobs ให้ทำงานบน node หลายๆ node ได้ Condor ก็เป็นระบบแบบนี้

3.4.3 Interconnection Network

ในระบบคลัสเตอร์เราใช้ Interconnection network หรือ System area network เพื่อเชื่อมต่อโหนดต่างๆ ในระบบที่เราจะพิจารณาเป็นตัวอย่างต่อไปก็มี Network ถึง 4 Network ระบบ Network แรกเป็นระบบพื้นฐานที่ใช้กันอยู่ใน LAN โดยทั่วๆไปคือ ระบบ Ethernet ซึ่งแต่ก่อนมี Bandwidth ต่ำประมาณ 10 Mbps ที่อาจต้องใช้หลาย Network ในการส่งข้อมูลจำนวน

มาก แต่ในปัจจุบัน Ethernet มีความเร็วถึง 1 Gbps และ 10 Gbps ซึ่งดีขึ้นมาก

Previous Technology	
Fast Ethernet	~100 Mbps
SCI	~4000 Mbps
OC-12 ATM	~622 Mbps
Fiber Channel	~100 MB/s
USB	12 Mbps
Firewire (IEEE 1394)	400 Mbps
Current Technology	
Gigabit Ethernet	~1000 Mbps
10 Gigabit Ethernet	10 Gbps
Myrinet	~1600 Mbps
Infiniband	10 – 20 Gbps

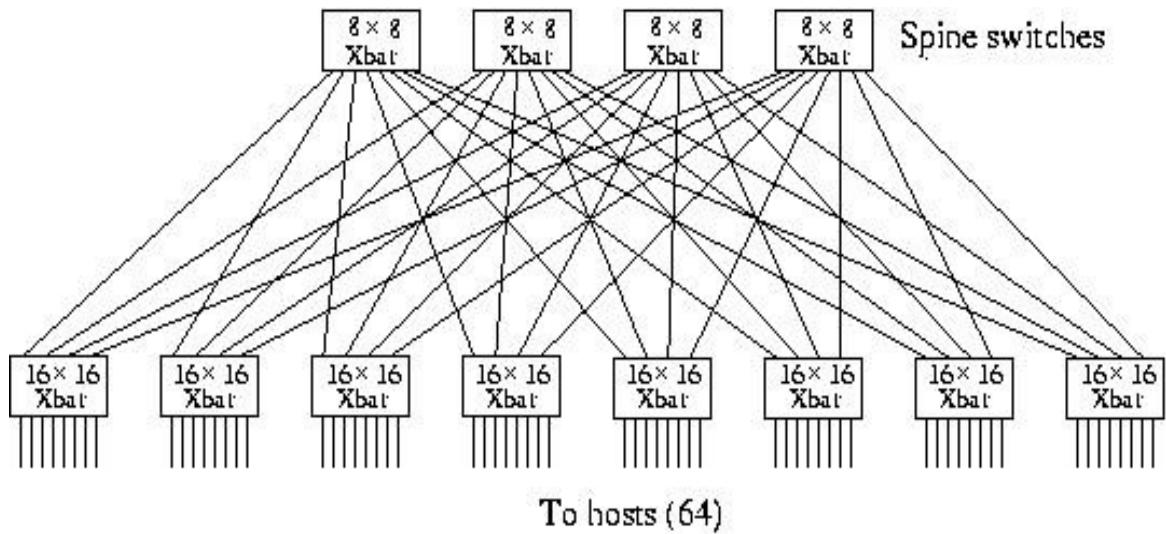
ภาพ 3-13

Network แบบที่สองคือระบบ Network ความเร็วสูง เช่นระบบ Myrinet ซึ่งได้รับการพัฒนาขึ้นโดยบริษัท Mericom ซึ่งเป็นบริษัทที่ Spin off จาก Cal Tech ระบบ Myrinet นั้นมี low latency และมี Bandwidth สูง นอกจากนั้นก็มีเทคโนโลยีใหม่ๆ เช่น Infiniband ระบบคลัสเตอร์ส่วนใหญ่จะใช้ 1 Gigabit Ethernet และระบบ Network ความเร็วสูงอีกอันหนึ่ง เช่น Infiniband ด้วยกัน

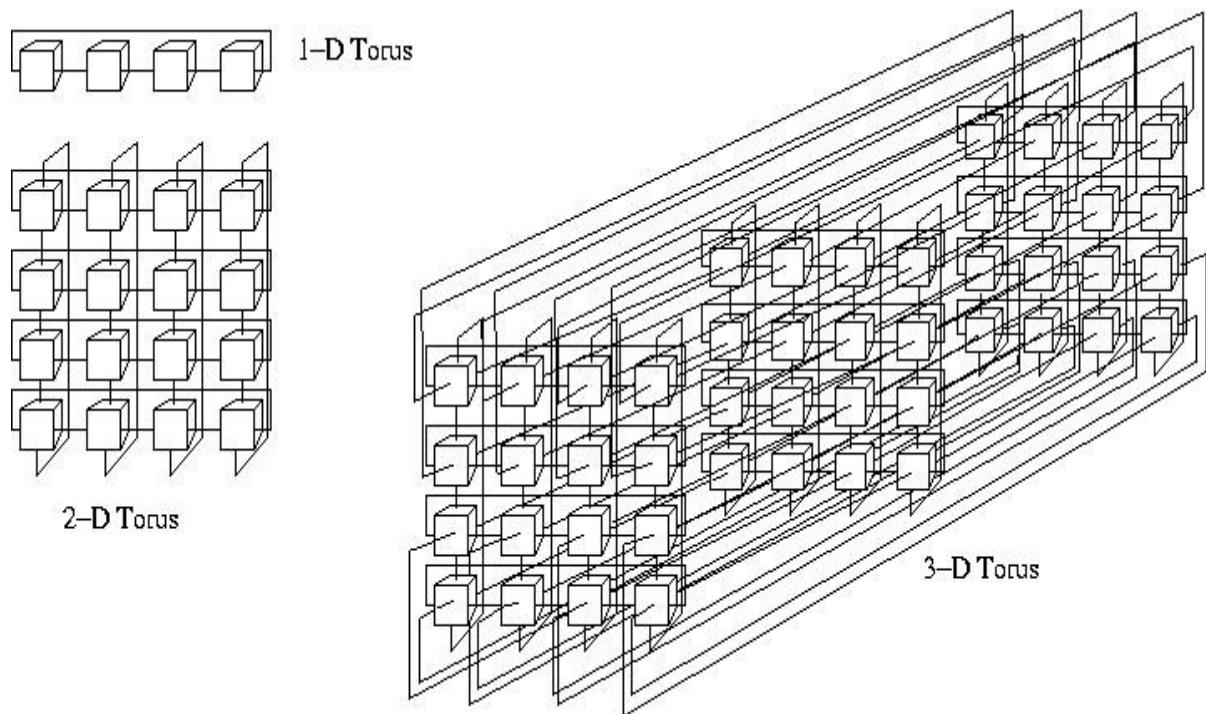
Network Topology

จริงๆแล้วรูปแบบของการเชื่อมต่อในทางทฤษฎีนั้นมีมากมาย แต่เรามักจะใช้อยู่ 4 Topology ใน การใช้งานจริง สองแบบที่ได้รับความนิยมมากที่สุดได้แก่ CLOS Network ซึ่งได้รับการพัฒนาต่อเป็นแบบ FAT TREE (ดังภาพ 3-) ซึ่งได้รับการพัฒนาขึ้นที่ MIT และได้ถูกนำไปใช้ในเครื่อง CM5 ของบริษัท Thinking Machine ระบบ FAT TREE เป็น Network ที่ทำงานแบบ non-blocking คือเมื่อโหนดใดต้องการส่งข้อมูลไปยังโหนดอื่นก็สามารถส่งได้ทันทีโดยไม่ต้องมีการ synchronization ก่อน

Network Topology อีกชนิดหนึ่งที่ใช้กันมากคือ MESH หรือ Toroidal Network ซึ่งมีลักษณะการเชื่อมต่อจากโหนดหนึ่งไปยังโหนดเพื่อนบ้านที่อยู่ใกล้ๆกันและต่อไปเรื่อยๆจนสุดพากหนึ่งแล้วก็วนกลับมาเชื่อมต่อกับโหนดอีกด้านหนึ่ง ภาพ 3- แสดง Toroidal แบบ 1-D หรือ 1 มิติ หรือ 2-D หรือ 2 มิติ หรือ 3-D หรือ 3 มิติ Network แบบ 3-D นั้นได้ถูกนำไปใช้ในเครื่อง Cray XT3 และ Cray XT4 Supercomputers

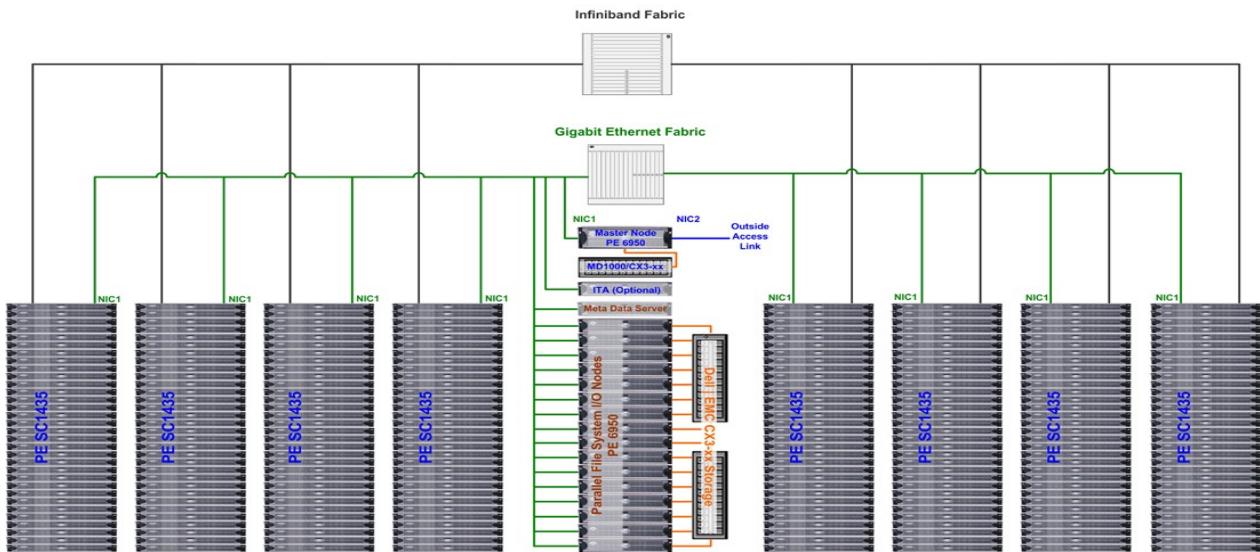


גראף 3-14



גראף 3-15

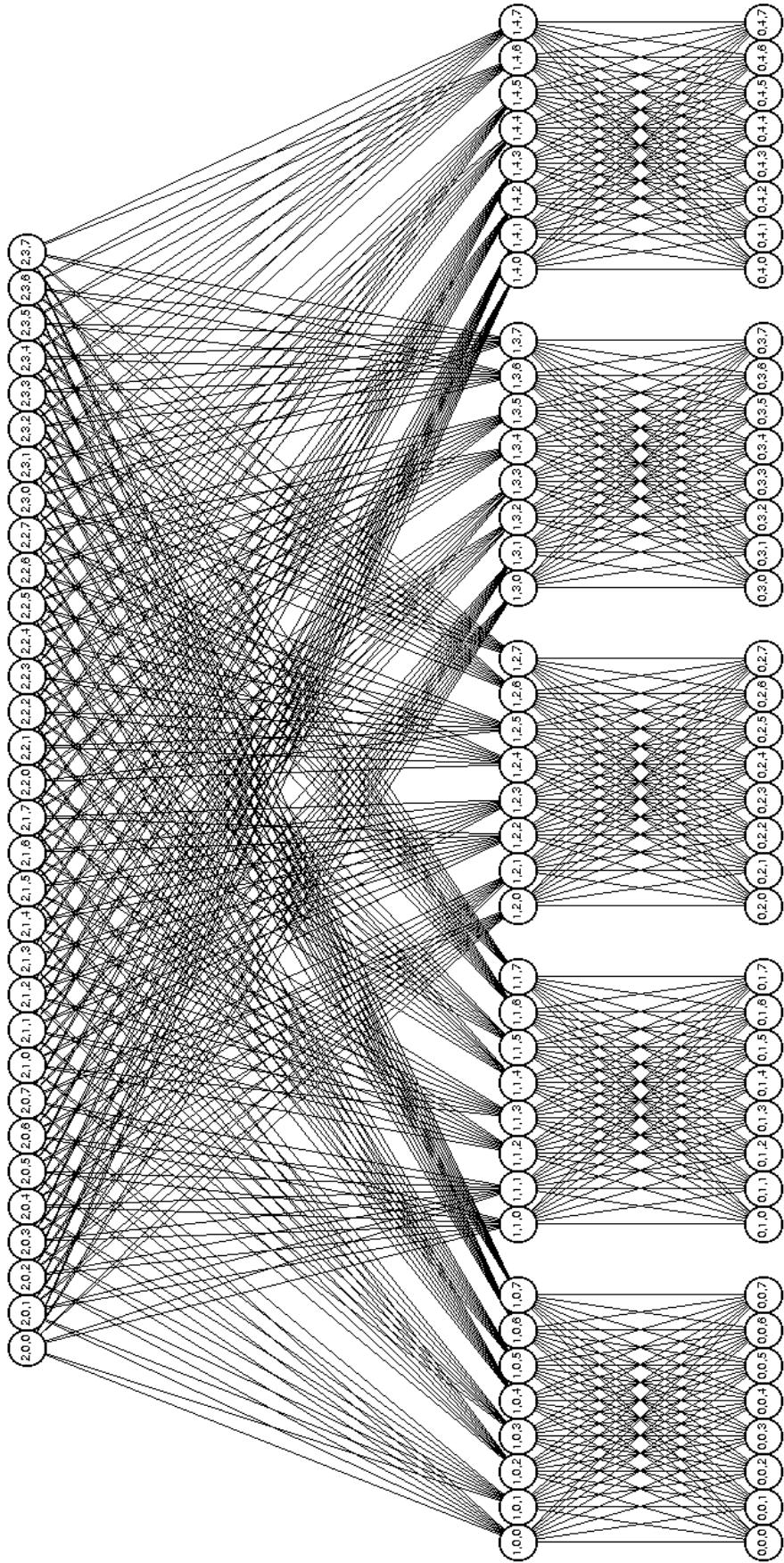
256-Node HPCC Bundle



ภาพ 3-16

ภาพ 3-16 เป็นรูปของเครื่องคลัสเตอร์ของบริษัท Dell ขนาดกลางซึ่งใช้ Network 2 แบบคือ Infiniband Network (Infiniband Fabric) ที่ใช้ในการติดต่อสื่อสารของ Parallel Applications และใช้ Gigabit Ethernet Network (Gigabit Ethernet Fabric) สำหรับการ maintenance ซึ่งทำให้การทำงานในหน้าที่ๆ เดียวกันเป็นไปได้โดยไม่มีการแบ่ง Network Bandwidth กัน จากภาพดูจะเห็นว่ามี Rack แบบ 1U case หลายๆ อันซึ่งเชื่อมต่อกันด้วย Switch ที่มี Topology แบบ FAT TREE 2 ระดับ

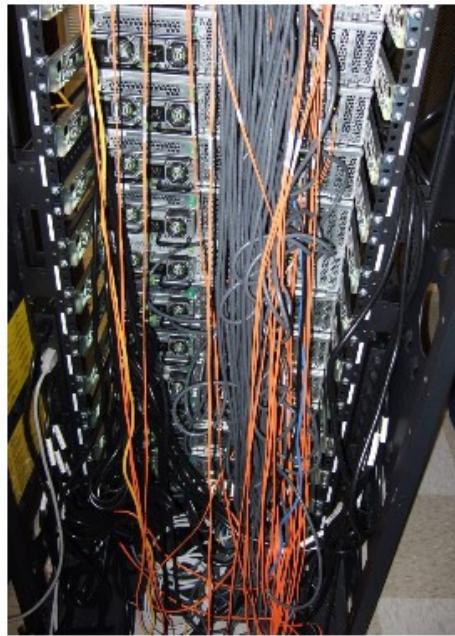
ภาพ 3-17 ถัดไปเป็นภาพของ Network FAT TREE 2 ระดับของบริษัท Myricomm ซึ่งเป็นแบบ Non-blocking, Fully-connected Topology และภาพถัดไป 3-18 แสดงระบบ Network ของเครื่องคลัสเตอร์ของเรา สายเคเบิลสีส้มในภาพทางด้านขวามีเป็นสาย Optical สำหรับ Myrinet ส่วนสีเทาเป็นสายเคเบิลธรรมด้าสำหรับ Ethernet ภาพข้างบนจะเป็น Network Interface Controller สำหรับแต่ละโหนดส่วนภาพล่างชี้ยังนั่นคือ Network switches ประกอบไปด้วย switch สำหรับ Ethernet (ชั้นบนสุด) และ switch สำหรับ Myrinet (ชั้นถัดลงมา)



מראג 3-17



Celeritas Myrinet Network



CSC

Department of Computer Science
Louisiana State University

CSC 7600 Lecture 3 : Commodity Clusters, Spring 2009

55

Draft

ภาพ 3-18

3.5 Software สำหรับคลัสเตอร์

ในระบบคลัสเตอร์นั้นจะมีทั้ง software ที่เป็น open source และที่เป็น commercial ถ้าเป็นแบบ open source มันจะมีแต่พัฒนาพื้นฐานในการใช้งานระบบคลัสเตอร์เท่านั้น ส่วนแบบ commercial นั้นจะมีพัฒนาที่ใช้งานได้ดีและง่ายรวมทั้งมี customer service supports ด้วยการใช้งานระบบคลัสเตอร์ที่ดีควรเป็นการใช้ software ทั่วสองประเทเวรร่วมกันอย่างสมดุลย์

PBS Scheduler

PBS เป็น scheduler software ที่กำหนดสภาวะแวดล้อมในการทำงานของระบบคลัสเตอร์และทำหน้าที่บริหารจัดการ jobs ที่รันอยู่ในนั้น PBS จะจัดการ job execution แบบ Batch Processing และอนุญาติให้ผู้ใช้กำหนด priority ของ job ต่างๆ นอกจากนั้นยังให้บริการเกี่ยวกับ Accounting เก็บประวัติการทำงานของ job ต่างๆรวมทั้งให้บริการ User Interfaces สำหรับดูแลและสั่งการระบบด้วย PBS นั้นได้รับการใช้งานอย่างแพร่หลายและบรรจุอยู่ใน Cluster Management Systems ที่นิยมกันอย่างกว้างขวางเช่นระบบ Rock จาก San Diego Supercomputing Center

Maui Scheduler

Scheduler software อีกเจ้าหนึ่งที่จัดการ job execution ในระดับที่ละเอียดกว่า PBS คือ Maui scheduler ซึ่งมันจะถูกใช้ควบคุมการรัน jobs ภายใต้ PBS อีกต่อหนึ่ง Maui เป็น scheduler ที่เหมาะสมกับการ scheduling job execution บนระบบ SMP แต่อย่างไรก็ตามเราสามารถใช้มันเพื่อ schedule การทำงานของ MPI process ข้ามโนําเดียวกันได้ด้วย Maui ได้รับการบรรจุไว้ใน Rock เช่นเดียวกัน

Condor Scheduler

ระบบ Condor Scheduler เป็น Scheduler เพื่อสนับสนุนการทำงานแบบ Throughput computing สำหรับ Serial Job และ Job แบบ SPMD ระบบ Condor มีพัฒนาการค่อนข้างยาวนานตั้งแต่ปี 1980 จากมหาวิทยาลัย วิสคอนซิน เมดิสัน และได้ถูกปรับปรุงให้มีความทันสมัยและนำมาใช้กับระบบ HPC และ Grid computing ในปัจจุบัน นอกจากนั้นมันยังถูกนำไปใช้กับ Megacomputing หรือ Peer-to-peer computing ซึ่งเป็นการนำ PC จากทั่วโลกบน internet มาช่วยกันทำงานอีกด้วย



MPI Software

ในการรัน application บนระบบคลัสเตอร์นั้นเป็นการรัน application ที่อาจประกอบไปด้วย process หลาย process รันอยู่บนโนําเดียวกันหรือต่างโนําเดียวกันและประสานงานกันด้วยการติดต่อสื่อสารแลกเปลี่ยนข้อมูล MPI เป็น software ในรูปแบบของ programming library ที่อนุญาตให้ process เหล่านั้นติดต่อกันและทำงานร่วมกันได้ เราจะพูดถึงรายละเอียดของ MPI ในบทที่ 5

Compiler และ Debugger

ในระบบคลัสเตอร์นั้นมี compiler และ debugger เป็น software tools ที่สำคัญสำหรับการพัฒนาซอฟต์แวร์ เราใช้ compiler ในการคอมไพล์ source code โดยอาจมีการ link โปรแกรมเข้ากับ library ต่างๆด้วย หลังจากนั้นเมื่อรันโปรแกรมเราอาจต้องใช้ debugger เช่น gdb เพื่อแก้ไขความผิดพลาดของโปรแกรม

เนื่องจากการทำงานในระบบคลัสเตอร์เป็นการทำงานที่เน้นเรื่องประสิทธิภาพ จึงต้องมีเครื่องมือสำหรับวัดและแสดงผลการวัดประสิทธิภาพการทำงานของโปรแกรมอุปกรณ์ เครื่องมือสองอย่างที่รู้จักกันดีได้แก่ PAPI ซึ่งมี library interfaces ที่ทำให้คุณสามารถอ่านข้อมูลจาก counter ที่ฝังอยู่ใน Microprocessor เพื่อที่จะจับเวลา function การทำงานต่างๆของ hardware และเครื่องมืออีกอย่างหนึ่งคือ TAU ซอฟต์แวร์ซึ่งให้บริการ User Interfaces เพื่อแสดงผลเหล่า

นั้น เครื่องมือเหล่านี้จะช่วยให้ผู้ใช้เห็นประสิทธิภาพของโปรแกรม นอกจานั้นยังมีเครื่องมืออื่นๆ อีก เช่น gprof และ prof counter

Distributed File System

ในระบบคลัสเตอร์นั้นมี persistent storage สำหรับเก็บข้อมูลอย่างถาวร ซึ่งบนระบบของเราใช้ Network file system หรือ NFS บนระบบ disk แบบ RAID 5 ซึ่งมีความจุประมาณ 7 terabytes นอกจากนั้น NFS ยังอนุญาติให้เราเก็บไฟล์แยกจากกันบนโหนดต่างๆ กันได้ และเราสามารถกำหนดให้โปรแกรมที่รันอยู่บนโหนดหนึ่งเข้าถึงไฟล์ที่อยู่ต่างโหนดกันได้ด้วย

ระบบคลัสเตอร์อาจใช้ File System ชนิดอื่นๆ ได้ ยกตัวอย่างเช่น PVFS ที่ได้รับการพัฒนาจาก Clemson University และที่ ANL เป็นระบบ file System ที่แยกกระจาย contents ของไฟล์ๆ เดียวไปเก็บไว้บนโหนดต่างๆ หลายๆ โหนดเพื่อเพิ่ม Bandwidth ในการอ่านและเขียนข้อมูล

3.6 การวัดประสิทธิภาพของระบบคลัสเตอร์

การวัดประสิทธิภาพการทำงานของระบบเป็นเรื่องที่สำคัญมากในการใช้งานระบบคลัสเตอร์ ในอันดับแรกคือเวลาในการทำงานของ application หรือ Wall clock time ซึ่งเป็นการวัดเวลาตั้งแต่เริ่มต้นการทำงานจนกระทั่งทำงานจบและเก็บ output ลงสู่ file systems

เมื่อเราต้องการวัดประสิทธิภาพของโปรแกรมบนระบบคลัสเตอร์ 2 ระบบ หรือแม้แต่บนเครื่องคอมพิวเตอร์ 2 เครื่อง เราจะใช้โปรแกรมที่เรียกว่า Benchmark ซึ่งเป็นโปรแกรมที่ได้รับการสร้างขึ้นมาเพื่อวัดประสิทธิภาพของเครื่อง ในการเปรียบเทียบนั้นเราจะรันโปรแกรมเหล่านี้บนเครื่องคลัสเตอร์แต่ละเครื่องแล้วเอาผลมาเปรียบเทียบกัน

การวัดอันดับถัดไปคือการวัด Efficiency ซึ่งเป็นการเปรียบเทียบว่าระบบทำงานได้มากขนาดไหน เมื่อเทียบกับที่คาดหวังไว้ มาตรวัด Efficiency นั้นมีหลายอย่าง

อย่างแรกที่ใช้กันมากที่สุดคือ Sustained Floating Point Performance หรือประสิทธิภาพในการคำนวนแบบ Floating point ที่ application ทำงานได้ ส่วน Peak floating point performance คือประสิทธิภาพในการทำงานสูงสุดของเครื่องคอมพิวเตอร์เครื่องนั้น ถ้าเราใช้ Linpack Benchmark เราอาจได้ประสิทธิภาพสูงกว่า 50 % แต่ถ้าเรารัน Linpack บนเครื่องที่มีประสิทธิภาพสูงอย่างเช่น the Earth Simulator ซึ่งมี Processor ที่อยู่ในกลุ่มกันมาก เราอาจได้ Efficiency ประมาณ 80 % หรือมากกว่านั้น แต่สำหรับ Application ที่เราใช้งานจริงอาจได้ค่า Efficiency ที่ต่ำลงมากอาจประมาณ 25 %

ในหลาย ๆ กรณีอย่างเช่นบนเครื่องที่ใช้งานสำหรับ Homeland Security เราอาจได้

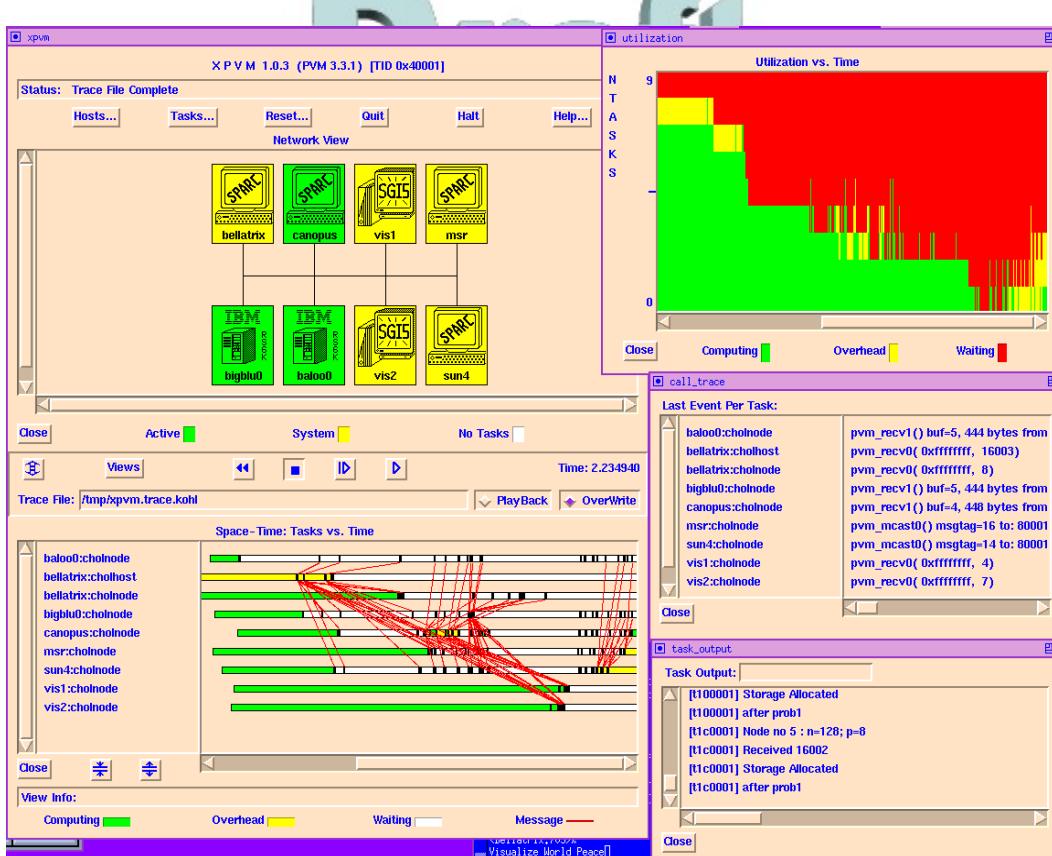
ประสิทธิภาพที่น้อยลงกว่านั้น เช่น 10 % หรือแม้แต่หลักเดียว

ในการวัดประสิทธิภาพนั้นเราต้องมีความเข้าใจพื้นฐานเกี่ยวกับประสิทธิภาพด้วย อย่างน้อยเราต้องรู้ว่าเมื่อเราเพิ่มจำนวนของ Resources ในระบบคอมพิวเตอร์ทำให้เครื่องมีขนาดใหญ่ขึ้น ขนาดของปัญหาที่ใช้ในการทดสอบต้องมีขนาดใหญ่ขึ้นด้วย

ประสิทธิภาพที่ลดลงนั้นอาจเจื่องมากจากการรัน operations หลายแบบซึ่งใช้เวลาในการทำงานไม่เท่ากันบางอันใช้เวลามากกว่าอันอื่น ยกตัวอย่างเช่น การทำ Memory Access จะใช้เวลานานกว่าการ บวก Interger ใน ALU ซึ่งมีจำนวน stage ประมาณ 7 ถึง 9 stages เท่านั้น ซึ่งเรานับเวลาตั้งแต่ CPU ส่งค่าจาก registers เข้าสู่ pipeline และนำค่าจาก Pipeline ไปไว้ใน registers ในทางตรงกันข้ามการบวกค่าที่อยู่ใน memory สองค่าอาจใช้เวลาถึง 100 หรือ 200 cycles ซึ่งเป็นความแตกต่างที่สูงมาก สำหรับการ Access Memory นั้น ถ้าข้อมูลที่เราต้องการใช้นั้นอยู่ใน cache เราจะใช้เวลา Access ข้อมูลเร็วขึ้นมากเพียงไม่กี่ cycles เท่านั้น

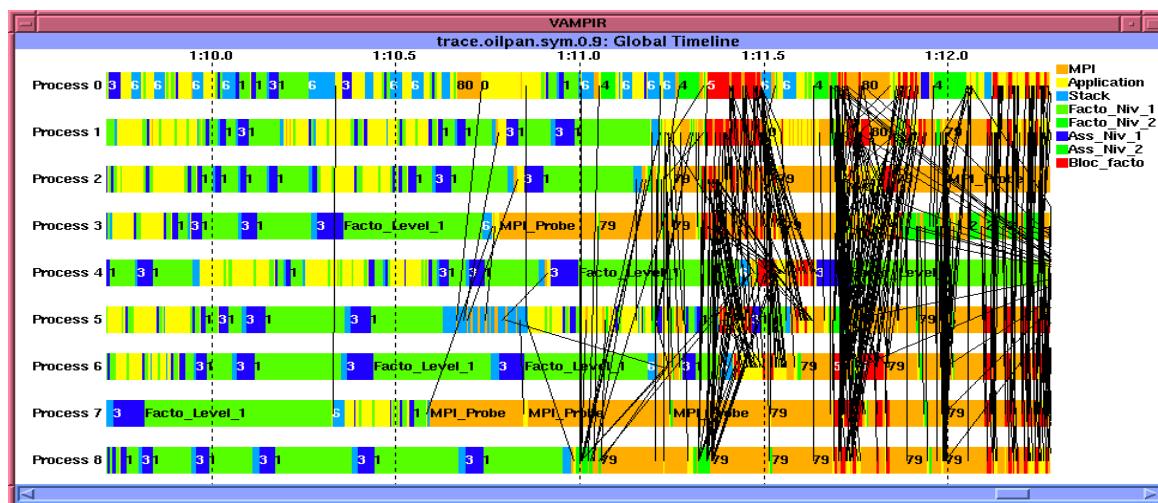
จากตัวอย่างข้างต้นจะเห็นว่าการวัดประสิทธิภาพนั้นเป็นเรื่องซับซ้อนและมีรายละเอียดมาก นอกจากระยะเวลาวัดประสิทธิภาพเรายังรู้ด้วยว่าข้อมูลที่จะนำมาประเมินผลนั้นอยู่ที่ไหน และมีการ Prefetch ข้อมูลหรือไม่

ตัวอย่างของเครื่องมือที่ใช้ในการวัดประสิทธิภาพเช่น XPVM



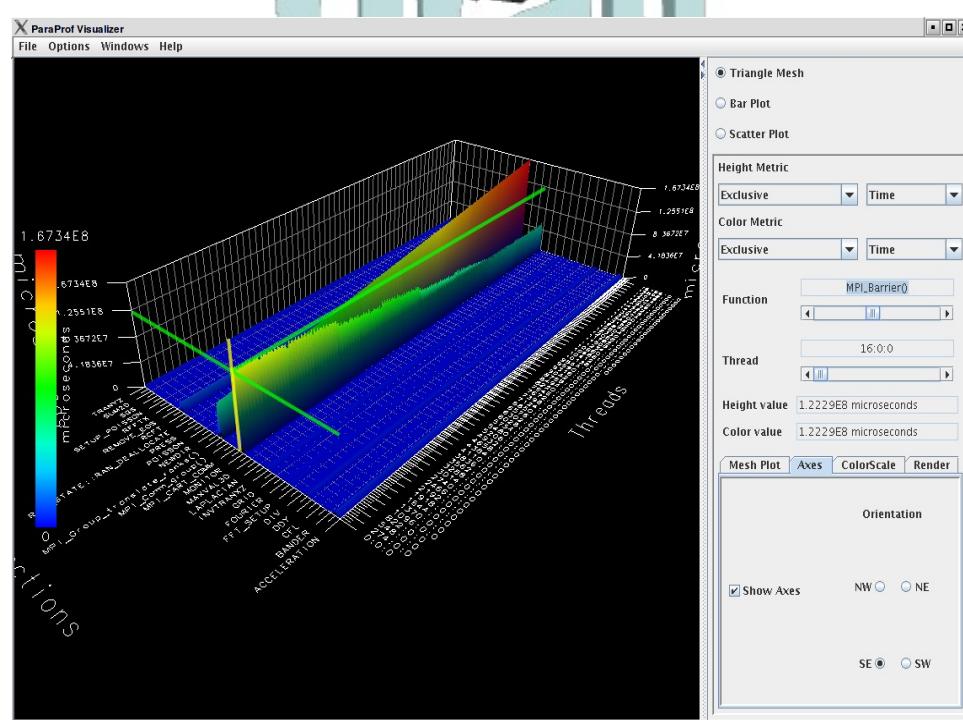
ภาพ 3-19

ระบบ XPVM เป็นระบบที่แสดงแผนภาพการติดต่อสื่อสารระหว่าง Process ใน Parallel Application และแสดงปริมาณเวลาที่ใช้ในการคำนวนและเวลาที่ระบบต้องรอการติดต่อสื่อสารในระหว่างที่มันทำงาน ระบบนี้ใช้สำหรับโปรแกรมที่ใช้ PVM library ในการติดต่อสื่อสาร



ภาพ 3-20

ภาพ 3-20 แสดงระบบ Vampire แสดงแผนภาพการติดต่อสื่อสารระหว่าง Process ของ MPI applications



ภาพ 3-21

ภาพ 3-21 แสดงเครื่องมือวัดประสิทธิภาพคือ TAU แสดงการ Visualization ประสิทธิภาพการทำงานของเครื่องในภาวะต่างๆ เป็นเครื่องมือไดร์บบาร์พัฒนาขึ้นที่ The university of Oregon

ตัวอย่างการ Compile และ Run โปรแกรมบนระบบ Cluster

ต่อไปนี้จะเป็นการอธิบายวิธีการเข้าใช้ระบบ Cluster ของเราและแสดงให้คุณดูว่าคุณจะ RUN Program อย่างไร เราจะใช้โปรแกรม High Performance Linpack

ในการเข้าใช้ระบบ Cluster ผู้จะแสดงให้คุณเห็นการเข้าใช้ระบบแบบ Remote โดยใช้โปรแกรม ssh และคุณจะเห็นสภาวะแวดล้อมอะไรบางอย่างตอนคุณ login เข้าใช้ระบบ ซึ่งเป็นระบบ UNIX หลังจากนั้นเราจะพูดถึง Compiler และ library ต่าง ๆ ซึ่งก็คือ Software Stack ที่คุณจะใช้ Compile โปรแกรมของคุณ

เราจะศึกษา MPI Compile time และ Runtime environment ซึ่งเป็นเรื่องที่สำคัญ ผู้เองเคยพลาダメาแล้ว คือ ผู้ที่จะ Compile โปรแกรมโดยใช้ MPI implementation ที่แตกต่างกัน ทำให้ผลที่ได้พิพากษา ผู้จะแนะนำวิธีที่คุณจะสามารถเช็ค version ของ MPI Software ของคุณ รวมทั้ง Ethernet Driver ที่ใช้งานด้วย เราจะทำการ Run งานหรือ job ง่ายๆ บนระบบ Cluster ด้วย ซึ่ง job ที่เราจะ Run ก็คือโปรแกรม HPL หรือ High Performance Linpack ซึ่งจริง ๆ แล้วนี้จะเป็น Assignment หนึ่งที่เราจะให้คุณทำ

1. การ login เข้าใช้งานระบบ

มีโปรแกรม ssh หลายโปรแกรมที่คุณสามารถใช้เข้าถึง Remote resource สำหรับ Windows คุณสามารถใช้ Utility ชื่อว่า Putty (พัตตี้) ซึ่งผู้จะใช้มันในวันนี้ แต่ก็มีโปรแกรมอื่น ๆ ด้วย เช่น ssh Mac OS X และ UNIX คุณสามารถใช้ ssh client จากคอมพิวเตอร์ของคุณเข้าถึงระบบ cluster

ผู้จะ login เข้าสู่ Account ของผู้ เนื่องจาก ระบบ Cluster ของเราใช้ระบบ UNIX และระบบของเราจะเป็นระบบ 64 bit Linux Cluster หลังจาก login แล้ว คุณสามารถหาข้อมูลเพิ่มเติมเกี่ยวกับระบบโดยใช้คำสั่ง “uname – a” คุณสามารถใช้คำสั่งอื่นได้ด้วย เช่น คำสั่ง ls ซึ่งเหมือนกับคำสั่ง dir ของระบบ Windows มันจะทำการ list รายชื่อของสิ่งที่อยู่ใน directory ปัจจุบัน แต่ถ้าคุณต้องการ directory ได้เป็นพิเศษ คุณก็สามารถพิมพ์ ls และตามด้วยชื่อ directory นั้นได้

คำสั่งอื่นที่คุณอาจใช้มันได้แก่ wget ซึ่งเป็นคำสั่งสำหรับ download file จาก internet ในวิชานี้ คุณจะใช้คำสั่งนี้สำหรับ download configuration file แต่อย่างไรก็ตามส่วนใหญ่แล้ว ในการทำงานคุณจะใช้คำว่า ls และ cd

คุณจำเป็นต้องรู้จักการใช้ Command line editor ด้วย คุณอาจใช้โปรแกรม nano หรือ pico ซึ่งใช้งานง่าย หรือคุณอาจใช้ vi ก็ได้ อีกวิธีหนึ่งที่ทำได้คือคุณเขียนโปรแกรมของคุณโดยใช้ editor บนเครื่องคอมพิวเตอร์ของคุณแล้ว ค่อยส่ง source code มาไว้บนเครื่องคลัสเตอร์ภายหลัง

ในการ Compile โปรแกรมของคุณ คุณจะใช้คำสั่ง Make ผิดๆ ว่าผู้ใช้ระบบ UNIX ส่วนใหญ่น่าจะคุ้นเคยกับคำสั่งเหล่านี้ ถ้าคุณอยากรู้รายละเอียดของคำสั่งใด ๆ ก็ให้ใช้คำสั่ง Man ตามด้วยชื่อของคำสั่งที่คุณต้องการหารายละเอียดนั้น

ผมจะแสดงการใช้คำสั่ง man ให้คุณดูนะครับ คือ ถ้าคุณต้องการรู้ว่า ls ทำอะไร ผมก็จะเขียนว่า man ls และคุณจะเห็น output ดังแสดงในภาพนะครับ

ตอนนี้สิ่งที่เราจะทำก็คือ เราจะ Compile โปรแกรม HPL ก่อนอื่น ผมจะ Copy files ต่าง ๆ เกี่ยวกับ HPL ซึ่งผมได้เก็บ file เหล่านี้ไว้ใน Compress file ชื่อว่า hpl.tgz นามสกุล tgz นี้หมายถึง tar และ gzip format

ผมจะใช้คำสั่ง untar เพื่อ uncompress file และตอนนี้คุณจะเห็นว่าเรามี directory ใหม่ชื่อว่า hpl ผมจะเข้าไปใน directory นั้นโดยใช้คำสั่ง cd คุณจะเห็นว่ามี Source file หลาย ๆ file ซึ่งเป็น Source Code ของโปรแกรม hpl หรือ high performance linpack

ผมมี make file ที่ทำไว้อยู่แล้ว และผมจะใช้คำสั่ง wget เพื่อ copy file นี้มาจากเครื่องคอมพิวเตอร์ของผม การใช้ wget คือ wget ตามด้วย http URL ของ makefile นั้นหลังจากนั้นให้ใช้ nano เพื่อ edit ไฟล์นี้ ถ้าไม่ใช้ nano ก็ใช้ vi แทนก็ได้ ให้เปลี่ยน directory เป็นที่ๆไฟล์นี้อยู่ หลังจากนั้นเราจะใช้ compiler เพื่อ compile ไฟล์

จากที่ได้บอกไปแล้วว่าเครื่อง cluster ของเราทั้งหมดรองรับด้วยคอมพิวเตอร์ 8 เครื่องซึ่งเชื่อมต่อกันด้วย Network 2 แบบ อย่างแรกคือ Ethernet ใช้ตัวย่อว่า eth และแบบที่สองคือ Myrinet network ใช้ตัวย่อว่า mx ซึ่ง MPI library ที่มีจะมีอยู่สี่อย่าง ตาม compiler ที่เรามีบน

เครื่อง และชนิดของ Network ที่มีอยู่

library แรกคือ mpich-eth ซึ่งหมายถึง library ที่ใช้กับ GNU Compiler และ Ethernet network ซึ่งถ้าเรา compile Parallel applications และ link เข้ากับ library นี้ Parallel applications นั้นจะใช้ Ethernet network ในการแลกเปลี่ยนข้อมูล แต่ถ้าเรา compile applications และ link เข้ากับ mpich-mx library การแลกเปลี่ยนข้อมูลก็จะเกิดขึ้นบน Myrinet network สำหรับ library ที่เหลืออีก 2 อันก็เป็น combination ระหว่าง network ทั้งสองกับ compiler ของ Intel

ในการใช้ compiler และ library ต่างๆนั้นเราจะกำหนดใน Makefile และใช้คำสั่ง make ในการกำหนดว่าจะใช้ library แบบไหนนั้นทำได้โดยการกำหนดค่าของ Environment variable MPI_PATH (ให้เท่ากับ gcc-eth ซึ่งเป็นการใช้ library แรก) ซึ่งสามารถกำหนดได้ในไฟล์ .bashrc

หลังจากนั้นเราก็จะได้ binary file ชื่อว่า xhpl ใน directory ชื่อ bin/linux

เราจะทำการรันโปรแกรม 3 โปรแกรมโดยใช้ processors 4 8 และ 16 processors ตามลำดับ ซึ่งเราสามารถกำหนดจำนวน processors ที่จะใช้ในไฟล์ HPL.dat ซึ่งเราต้องกำหนดค่าของ variables P และ Q ในไฟล์นั้น และจำนวน processors ทั้งหมดที่จะใช้คือผลคูณของค่าทั้งสอง

แต่ว่าเราจะบอกระบบ cluster อย่างไรว่าเราต้องการใช้ processors กี่ processors?

เราจะใช้ซอฟแวร์ที่ชื่อว่า PBS ในการสั่งงานนี้ โดยที่เราจะต้องสร้าง script file เพื่อกำหนดว่าเราต้องการ resource ใดบ้างของระบบ Cluster ในการรันโปรแกรม ในบางระบบเราราจุใช้โปรแกรม load leveler แทนที่จะเป็น PBS

สำหรับรายละเอียดการกำหนดค่าใน script file นั้น คุณสามารถกำหนดค่าต่างๆ เช่น maximum wallclock time หรือระยะเวลาที่มากที่สุดที่โปรแกรมจะรัน นอกจากนั้นก็สามารถกำหนดค่าของจำนวน processors หรือ cores ที่จะใช้ในแต่ละ node และกำหนด path ของ binary file ที่ต้องการจะรัน และกำหนดคำสั่งที่จะรันก็คือ mpirun เมื่อกำหนดค่าต่างๆแล้วเราจะป้อน script นี้ให้กับระบบ PBS เพื่อใช้รันโปรแกรมต่อไป

สมมุติว่าเราสร้าง script file ชื่อว่า hpl.pbs เมื่อเราจะรันโปรแกรมเราจะใช้คำสั่ง

\$ qsub hpl.pbs

และถ้าอยากรู้ว่ามีโปรแกรมอะไรที่รันอยู่ในคิวของระบบ cluster บ้างให้ใช้คำสั่ง

\$ qsub -a

หลังจากนั้นเราจะได้ output file ออกมากับรายงานประสททิภาพในการทำงานของโปรแกรม
รายงานออกมาด้วย

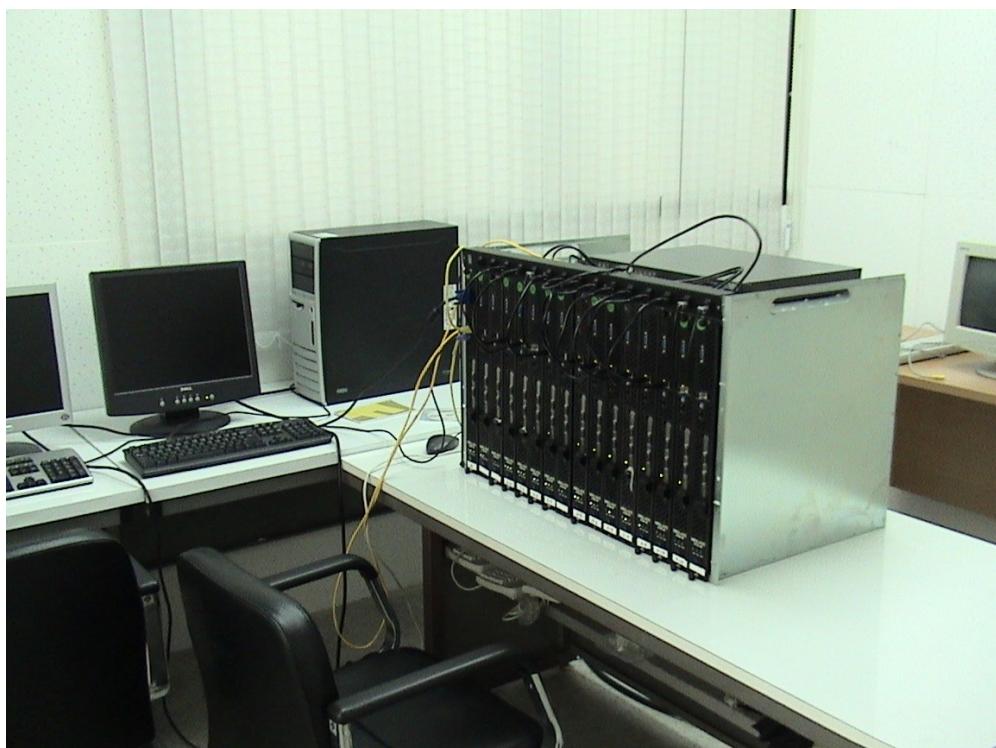
Draft

บทที่ 4 การใช้งานระบบ Cluster Computer เป็นต้น และ ssh โพรโตคอล

แต่งและเรียนเรียง กษิดิศ ชาญเชี่ยว

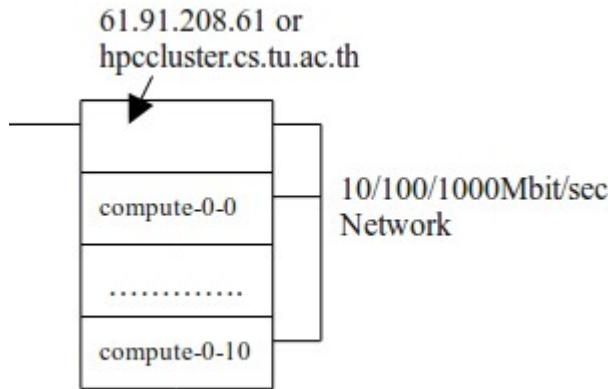
4.1 แนะนำระบบ hpccluster

เครื่อง hpccluster.cs.tu.ac.th เป็นเครื่องคอมพิวเตอร์สมรรถนะสูงที่ภาควิชาคอมพิวเตอร์และคณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์ ด้วยความร่วมมือกับ eXtreme Computing Laboratory ที่ The Department of Computer Science, Louisiana Tech University ได้รับการบริจาดจากบริษัท Intel โดยที่คอมพิวเตอร์เครื่องนี้ประกอบไปด้วยเครื่องคอมพิวเตอร์ Intel Xeon 2 CPU จำนวน 10 เครื่องติดตั้งอยู่ที่ห้อง 301/005 ดังที่แสดงในภาพ 4-1



ภาพ 4-1

โครงสร้างของเครื่อง hpccluster นั้นประกอบไปด้วยเครื่อง hpccluster และระบบ Gigabit Ethernet Networks ดังแสดงในภาพ 4-2



ภาพ 4-2

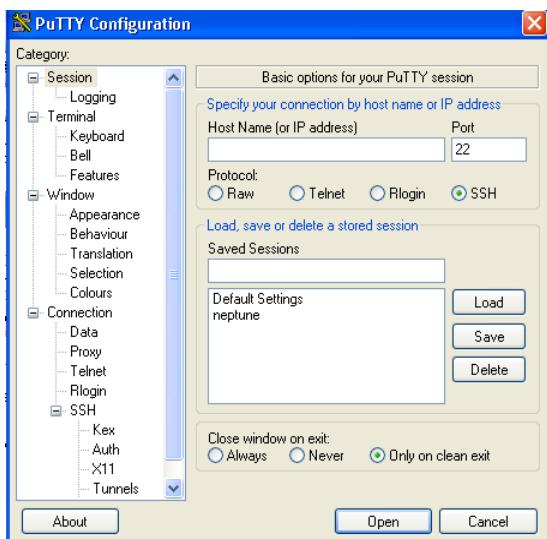
จากภาพ 4-2 คุณสามารถเข้าใช้งานเครื่อง cluster ได้ผ่าน Internet โดยการ login เข้าสู่เครื่อง hpccluster.cs.tu.ac.th (หรือ 61.91.208.61) ซึ่งเป็น front node ของ hpccluster ซึ่งจะเชื่อมต่อกับเครื่อง compute-node อีก 10 เครื่องผ่าน 10/100/1000Mbit/s Ethernet Switch ที่ทำหน้าที่เป็น network ภายในหอندต่างๆของระบบคลัสเตอร์ ส่วนเครื่อง front node นั้นมี NIC 2 อัน อันหนึ่งต่อกับ Network ภายนอก ส่วนอีกอันเชื่อมต่อกับ Internet

เครื่อง front node และแต่ละ Node ของ cluster จะมี 2 Intel Xeon CPU มี RAM 2 Gigabytes มี Hard disk 40 Gigabytes ทุกเครื่องติดตั้ง ROCKS cluster management system เครื่อง hpccluster จะมีข้อจำกัดในเรื่องของ harddisk ในปัจจุบันเนื่องจากผู้ใช้ทุกๆคน จะใช้ harddisk จากเครื่อง front node เท่านั้น ดังนั้นมีการใช้งานมากขึ้นเนื่องที่ harddisk ที่ใช้ร่วมกันอาจมีจำกัด เครื่อง compute-node ทุกๆเครื่องจะ share file systems บนเครื่อง front-node ผ่าน NFS ดังนั้นคุณสามารถติดตั้ง software หรือโปรแกรมของคุณครั้งเดียวที่ front node และเรียกใช้งานจาก compute-node ได้ก็ได้ ในการแก้ปัญหาข้อจำกัดของ hard disk นั้นคุณอาจใช้เครื่องของคุณ เพื่อช่วยเก็บ files ของคุณแล้ว transfer files มายังเครื่อง hpccluster เมื่อต้องการ

การเปิดให้บริการของเครื่อง hpccluster นั้นจะมีตารางเวลาของการเปิดให้บริการซึ่งทางผู้จัดการระบบจะแจ้งให้ทราบทาง email การให้บริการสามารถทำตามการร้องขอของผู้ใช้ได้ เช่น กันในกรณีที่ผู้ใช้ต้องใช้งานเป็นระยะเวลานาน ตามปกติแล้วเราจะพยายามเปิดเครื่องให้มีการใช้งานได้นานที่สุดเท่าที่จะทำได้ แต่ถึงอย่างไรก็ตามก็จะต้องมี Down time ในกรณีของการซ่อมบำรุง

4.2 การเข้าใช้ user account

ในการที่คุณจะใช้เครื่อง hpccluster.cs.tu.ac.th (ถ้า log in โดยใช้ชื่อไม่ได้ให้ใช้ IP addresses แทน) คุณจำเป็นต้อง login และใส่ข้อมูลเริ่มต้นที่จำเป็นอย่างเช่น passphrase สำหรับ public/private keys ของคุณบนเครื่องนั้นก่อน ยกตัวอย่างเช่น สมมุติว่าเครื่อง account ของคุณคือ user01 บนเครื่อง hpccluster และคุณต้องการเข้าใช้เครื่อง hpccluster จากคอมพิวเตอร์ที่ติดตั้งระบบ MS Windows คุณต้อง remote login เข้าสู่เครื่องนั้นโดยใช้โปรแกรมอย่างเช่น putty.exe¹ ดังแสดงในภาพ 4-3 และภาพ 4-4



ภาพ 4-3

```
user01@hpccluster:~$ login as: user01
user01@61.91.208.61's password:
Rocks 4.2.1 (Cydonia)
Profile built 16:45 27-Apr-2007

Kickstarted 00:19 28-Apr-2007
Rocks Frontend Node - hpccluster Cluster

It doesn't appear that you have set up your ssh key.
This process will make the files:
  /home/user01/.ssh/id_rsa.pub
  /home/user01/.ssh/id_rsa
  /home/user01/.ssh/authorized_keys

Generating public/private rsa key pair.
Enter file in which to save the key (/home/user01/.ssh/id_rsa):
Created directory '/home/user01/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user01/.ssh/id_rsa.
Your public key has been saved in /home/user01/.ssh/id_rsa.pub.
The key fingerprint is:
89:3e:b2:01:89:39:b0:1d:9c:36:0f:57:81:85:de:a2 user01@hpccluster.cs.tu.ac.th
[user01@hpccluster ~]$
```

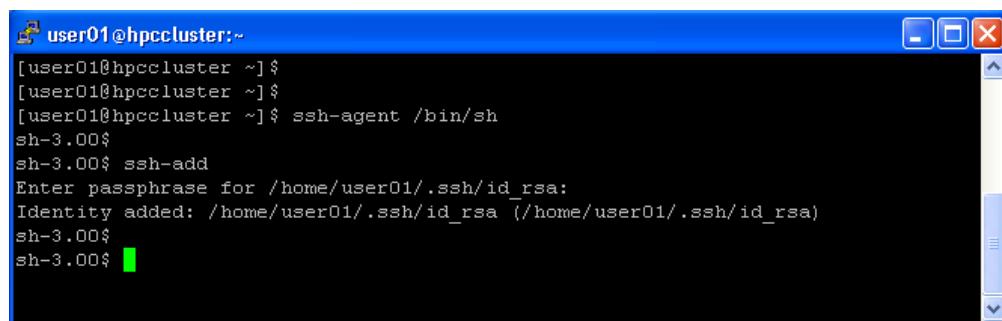
ภาพ 4-4

¹ ดาวน์โหลดได้จาก <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

ในกรณีที่คุณยังไม่มี Account บนระบบ hpccluster คุณสามารถส่ง email ไปยังผู้จัดการระบบเพื่อสร้าง Account ของคุณบนเครื่อง hpccluster การใช้งานระบบจะต้องสอดคล้องกับ Account usage policy ของภาควิชาวิทยาการคอมพิวเตอร์ฯ

เมื่อคุณได้รับ Account และ password และให้ login เข้าสู่ Account ของคุณ จะเห็นข้อความดังในภาพ 4-4 ซึ่ง rocks จะถามตำแหน่งที่คุณจะเก็บ private key ซึ่งคุณจะกด enter แล้ว rocks จะถาม passphrase สำหรับ encrypt private key ของคุณให้คุณใส่ passphrase สองครั้งและเข้าสู่การทำงานของ shell ที่จะรอรับคำสั่งของคุณต่อไป

ในการที่จะสั่งงานระบบ hpccluster ให้ทุกๆ nodes ทำงานตามที่คุณต้องการโดยไม่ต้องใส่ password หรือ passphrase ทุกครั้งที่มีการเข้าถึงแต่ละ compute-node นั้น คุณต้องใช้คำสั่ง ssh-agent เพื่อทำให้คุณสามารถออกคำสั่งข้ามเครื่องทุกๆ compute nodes ได้โดยไม่ต้องใช้ password ในขั้นแรกให้คุณออกคำสั่ง ssh-agent ดังแสดงในภาพ 4-5² และหลังจากนั้นให้ใช้คำสั่ง ssh-add เพื่อดึงเอา private key มาไว้ใน memory ซึ่ง ssh-add จะถามหา passphrase เพื่อเขามา decrypt private key ที่คุณสร้างเก็บไว้



```
[user01@hpccluster ~]$ [user01@hpccluster ~]$ [user01@hpccluster ~]$ ssh-agent /bin/sh sh-3.00$ sh-3.00$ ssh-add Enter passphrase for /home/user01/.ssh/id_rsa: Identity added: /home/user01/.ssh/id_rsa (/home/user01/.ssh/id_rsa) sh-3.00$ sh-3.00$
```

ภาพ 4-5

หลังจากนั้นคุณน่าจะสามารถออกคำสั่งให้เครื่อง compute nodes ทั้งหลายทำงานได้โดยใช้คำสั่ง cluster-fork ดังในภาพ 4-6 เราออกคำสั่ง “cluster-fork uptime” ซึ่งจะแสดง workloads ของ compute-nodes ทุกเครื่องบนหน้าจอ

² จากภาพถ้าใช้คำสั่ง ssh-agent /bin/bash ในตัวอย่างนี้ คุณจะได้ shell แบบเดียวกันกับ login shell ของคุณ

```

sh-3.00$ cluster-fork uptime
compute-0-0:
Warning: Permanently added 'compute-0-0' (RSA) to the list of known hosts.
20:02:54 up 11:46, 0 users, load average: 0.00, 0.00, 0.00
compute-0-1:
Warning: Permanently added 'compute-0-1' (RSA) to the list of known hosts.
20:02:55 up 11:46, 0 users, load average: 0.01, 0.02, 0.00
compute-0-2:
Warning: Permanently added 'compute-0-2' (RSA) to the list of known hosts.
20:02:55 up 11:46, 0 users, load average: 0.08, 0.10, 0.09
compute-0-3:
Warning: Permanently added 'compute-0-3' (RSA) to the list of known hosts.
20:02:56 up 11:46, 0 users, load average: 0.00, 0.00, 0.00
compute-0-5:
Warning: Permanently added 'compute-0-5' (RSA) to the list of known hosts.
20:02:56 up 11:46, 0 users, load average: 0.07, 0.10, 0.07
compute-0-6:
Warning: Permanently added 'compute-0-6' (RSA) to the list of known hosts.
20:02:57 up 11:46, 0 users, load average: 0.00, 0.00, 0.00
compute-0-7:
Warning: Permanently added 'compute-0-7' (RSA) to the list of known hosts.
20:02:58 up 11:46, 0 users, load average: 0.00, 0.01, 0.00
compute-0-8:
Warning: Permanently added 'compute-0-8' (RSA) to the list of known hosts.
20:02:58 up 11:32, 0 users, load average: 0.02, 0.02, 0.00
compute-0-9:
Warning: Permanently added 'compute-0-9' (RSA) to the list of known hosts.
20:02:59 up 11:46, 0 users, load average: 0.01, 0.01, 0.00
compute-0-10:

```

ภาพ 4-6

4.3 การพัฒนา MPI โปรแกรมบนระบบ hpccluster

ในการพัฒนา MPI program นั้นเราจะใช้ LAM/MPI [8] ซึ่งเป็นหนึ่งใน MPI

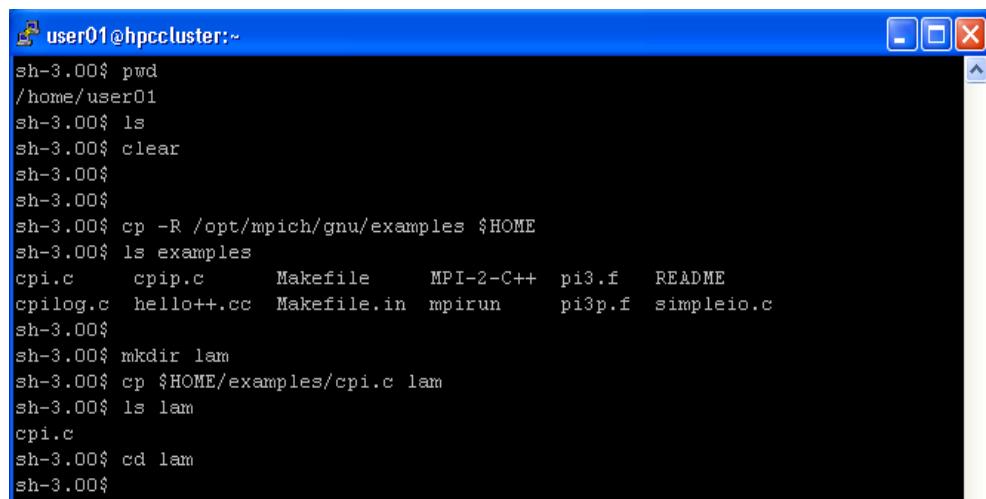
Implementations บนเครื่อง hpccluster³ ในเอกสารนี้เราจะแสดงการใช้งาน LAM/MPI อย่างง่ายๆ จริงๆ แล้วในการพัฒนาโปรแกรม MPI คุณสามารถทดลองพัฒนาได้หลายวิธี

1. ใช้ DEINO MPI software ที่ <http://mpi.deino.net/> คุณสามารถดาวน์โหลดแล้วติดตั้ง และใช้งานบนเครื่อง windows ของคุณก่อนเอา source code มา compile บนเครื่อง cluster [4]
2. ติดตั้ง vmware หรือ qemu และติดตั้ง linux อย่างเช่น fedora core และติดตั้ง MPI บน linux อีกที ให้ดู <http://www.openmpi.org> [5] หรือ <http://www-unix.mcs.anl.gov/mpi/mpich/> [6] เกี่ยวกับการติดตั้ง MPI
3. login เข้าไปที่เครื่อง ft.cs.tu.ac.th และทดลองที่นั่นแล้วค่อย transfer files ไปที่ hpccluster ดังที่กล่าวข้างต้น
4. login เข้าสู่ hpccluster.cs.tu.ac.th โดยตรงและพัฒนาโปรแกรมใน account ของคุณ

³ ดูรายละเอียดการใช้งาน LAM/MPI ได้ที่ <http://www.lam-mpi.org/download/files/7.1.3-user.pdf>

4.3.1 การ compile โปรแกรมด้วย mpicc

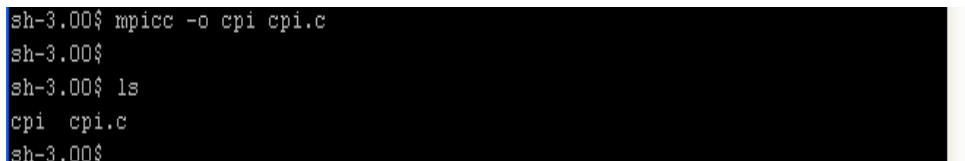
คุณสามารถ copy ตัวอย่าง MPI programs มาจาก MPI directory ได้ดังในภาพ 4-8 ซึ่งคุณจะ copy files จาก directory /opt/mpich/gnu/examples มาไว้ที่ใดที่หนึ่งใน \$HOME directory ของคุณ ในตัวอย่างนี้ผมจะ copy มาไว้ที่ \$HOME คุณจะสังเกตได้ว่าตัวอย่างโปรแกรมเหล่านี้เป็นตัวอย่างที่มากับ MPICH Implementation ดังนั้นเรายังไม่สามารถนำ Makefile ใน directory ของมันมาใช้ตรงๆได้เราจะเอา source code มาทดลองเฉยๆ หลังจากนั้นผมจะสร้าง sub directory ชื่อ lam ไว้ภายใน \$HOME และ copy โปรแกรม cpi.c ไปไว้ที่นั่น



```
user01@hpccluster:~$ sh-3.00$ pwd
/home/user01
sh-3.00$ ls
sh-3.00$ clear
sh-3.00$
sh-3.00$ sh-3.00$ cp -R /opt/mpich/gnu/examples $HOME
sh-3.00$ ls examples
cpi.c      cpi.c      Makefile      MPI-2-C++  pi3.f      README
cpilog.c   hello++.cc  Makefile.in  mpirun     pi3p.f    simpleio.c
sh-3.00$ sh-3.00$ mkdir lam
sh-3.00$ cp $HOME/examples/cpi.c lam
sh-3.00$ ls lam
cpi.c
sh-3.00$ cd lam
sh-3.00$
```

ภาพ 4-8

ต่อไปในภาพ 4-9 คุณสามารถใช้คำสั่ง mpicc เพื่อ compile โปรแกรม cpi.c ซึ่งเราแสดง source code ในภาพที่ 10 โปรแกรมที่ถูก compile และจะสร้าง file executable ชื่อว่า cpi ใน \$HOME/lam directory



```
sh-3.00$ mpicc -o cpi cpi.c
sh-3.00$ sh-3.00$ ls
cpi  cpi.c
sh-3.00$
```

ภาพ 4-9

```

#include "mpi.h"
#include <stdio.h>
#include <math.h>
double f( double );
double f( double a )
{return (4.0 / (1.0 + a*a));}

int main( int argc, char *argv[] )
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

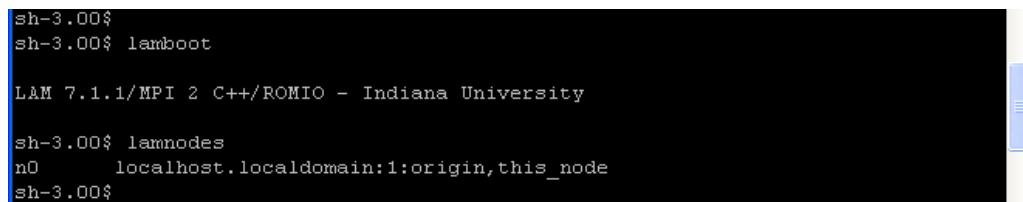
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);
    fprintf(stderr,"Process %d on %s\n",
            myid, processor_name);
    n = 0;
    while (!done){
        if (myid == 0){
            if (n==0) n=100; else n=0;
            startwtime = MPI_Wtime();
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0)
            done = 1;
        else{
            h = 1.0 / (double) n;
            sum = 0.0;
            for (i = myid + 1; i <= n; i += numprocs){
                x = h * ((double)i - 0.5);
                sum += f(x);
            }
            mypi = h * sum;

            MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
            if (myid == 0){
                printf("pi is approximately %.16f, Error is %.16f\n",
                       pi, fabs(pi - PI25DT));
                endwtime = MPI_Wtime();
                printf("wall clock time = %f\n",
                       endwtime-startwtime);
            }
        }
    }
    MPI_Finalize();
    return 0;
}

```

4.3.2 การ run โปรแกรมด้วย mpirun

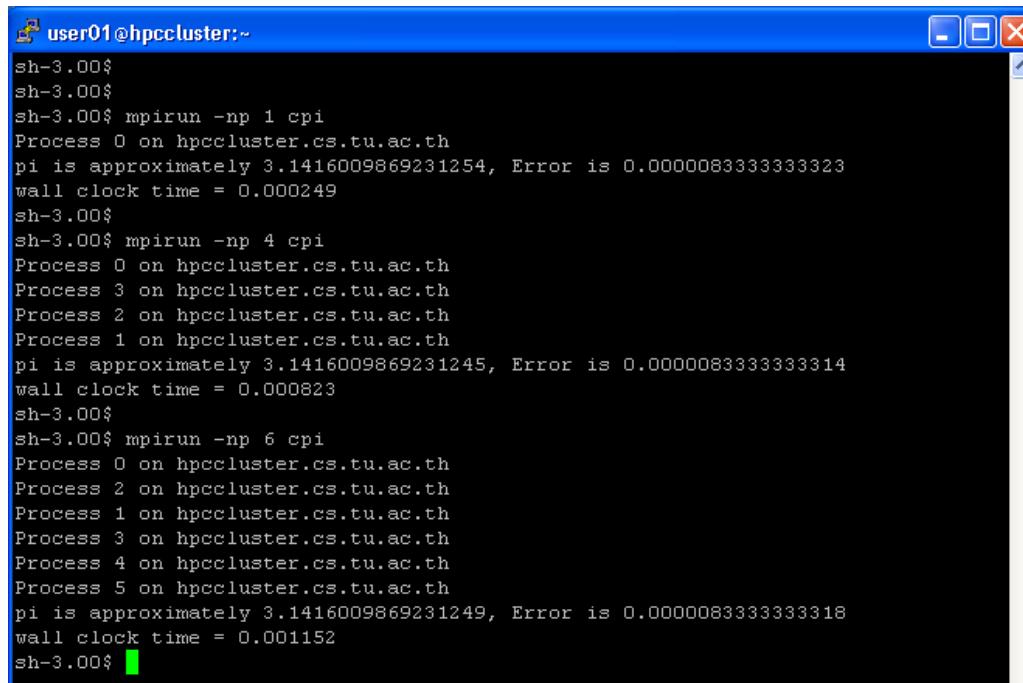
ในการที่จะรันโปรแกรม MPI ได้คุณต้องสร้าง LAM runtime system ก่อนซึ่งทำได้โดยการออกคำสั่ง lamboot และหลังจากนั้นคุณสามารถใช้คำสั่ง lamnodes เพื่อเช็คว่ามีเครื่องใดอยู่ในระบบ LAM runtime system (or LAM universe) บ้าง ในกรณีที่คุณรัน lamboot โดยไม่มี parameter นั้น LAM จะนำ localhost มาเป็นเครื่องในระบบและให้รหัสเครื่องเป็น n0 ดังภาพ 4-11 (โปรดอย่าลืมว่าคุณได้ทำ ssh-agent และ ssh-add มา ก่อนแล้วจึงทำให้คุณสามารถสร้าง LAM runtime system ที่ทำงานบนหลายเครื่องได้โดยไม่ต้องใส่ password หรือ passphrase)



```
sh-3.00$  
sh-3.00$ lamboot  
  
LAM 7.1.1/MPI 2 C++/ROMIO - Indiana University  
  
sh-3.00$ lamnodes  
n0      localhost.localdomain:1:origin,this_node  
sh-3.00$
```

ภาพ 4-11

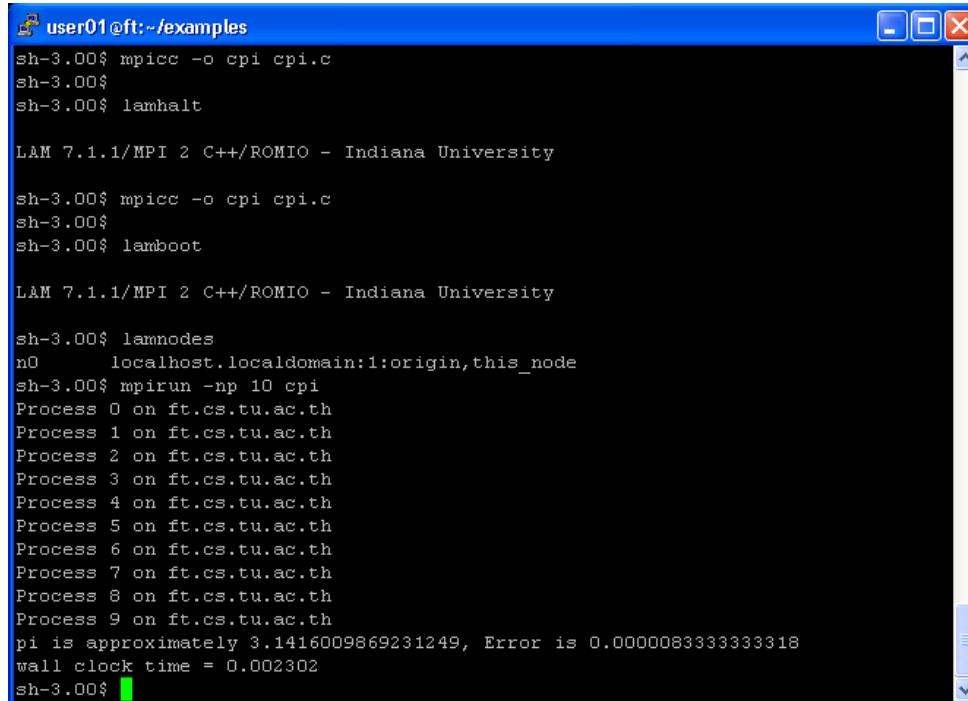
คุณสามารถออกคำสั่ง “mpirun -np 1 cpi” ดังเช่นในภาพ 4-12 เพื่อรันโปรแกรม cpi 1 process บนเครื่อง n0 ซึ่งก็ได้ผลดังภาพ ถ้าคุณต้องการรัน cpi 4 processes หรือ 6 processes คุณก็สามารถทำได้ดังในตัวอย่าง



```
user01@hpccluster:~  
sh-3.00$  
sh-3.00$  
sh-3.00$ mpirun -np 1 cpi  
Process 0 on hpccluster.cs.tu.ac.th  
pi is approximately 3.1416009869231254, Error is 0.0000083333333323  
wall clock time = 0.000249  
sh-3.00$  
sh-3.00$ mpirun -np 4 cpi  
Process 0 on hpccluster.cs.tu.ac.th  
Process 3 on hpccluster.cs.tu.ac.th  
Process 2 on hpccluster.cs.tu.ac.th  
Process 1 on hpccluster.cs.tu.ac.th  
pi is approximately 3.1416009869231245, Error is 0.0000083333333314  
wall clock time = 0.000823  
sh-3.00$  
sh-3.00$ mpirun -np 6 cpi  
Process 0 on hpccluster.cs.tu.ac.th  
Process 2 on hpccluster.cs.tu.ac.th  
Process 1 on hpccluster.cs.tu.ac.th  
Process 3 on hpccluster.cs.tu.ac.th  
Process 4 on hpccluster.cs.tu.ac.th  
Process 5 on hpccluster.cs.tu.ac.th  
pi is approximately 3.1416009869231249, Error is 0.0000083333333318  
wall clock time = 0.001152  
sh-3.00$
```

ภาพ 4-12

ในภาพ 4-13 นั้นเราแสดงให้เห็นว่าถ้าคุณใช้ account ของคุณบนเครื่อง ft.cs.tu.ac.th ทำแบบเดียวกันกับที่ทำตลอดมาบนเครื่อง hpccluster และคุณก็สามารถ compile และรันโปรแกรม mpi ของคุณและทดสอบการทำงานของมั่นบนเครื่อง ft.cs.tu.ac.th ได้เช่นกัน



```

user01@ft:~/examples
sh-3.00$ mpicc -o cpi cpi.c
sh-3.00$
sh-3.00$ lamhalt

LAM 7.1.1/MPI 2 C++/ROMIO - Indiana University

sh-3.00$ mpicc -o cpi cpi.c
sh-3.00$
sh-3.00$ lamboot

LAM 7.1.1/MPI 2 C++/ROMIO - Indiana University

sh-3.00$ lamnodes
n0      localhost.localdomain:1:origin,this_node
sh-3.00$ mpirun -np 10 cpi
Process 0 on ft.cs.tu.ac.th
Process 1 on ft.cs.tu.ac.th
Process 2 on ft.cs.tu.ac.th
Process 3 on ft.cs.tu.ac.th
Process 4 on ft.cs.tu.ac.th
Process 5 on ft.cs.tu.ac.th
Process 6 on ft.cs.tu.ac.th
Process 7 on ft.cs.tu.ac.th
Process 8 on ft.cs.tu.ac.th
Process 9 on ft.cs.tu.ac.th
pi is approximately 3.1416009869231249, Error is 0.000008333333318
wall clock time = 0.002302
sh-3.00$ 

```

ภาพ 4-13

เมื่อคุณเสร็จสิ้นการใช้งาน MPI และ LAM runtime system และคุณต้องเรียกคำสั่ง lamhalt เพื่อกำจัด LAM runtime system ออกไป ดังแสดงในภาพ 4-14



```

sh-3.00$ lamhalt

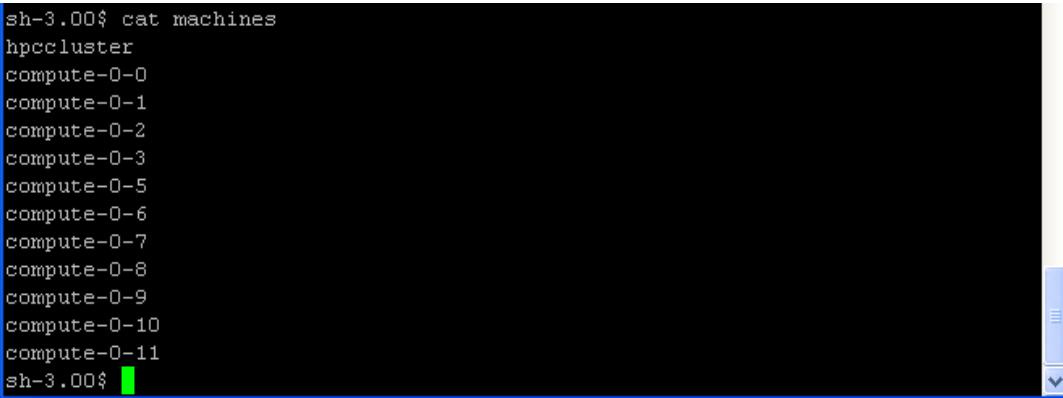
LAM 7.1.1/MPI 2 C++/ROMIO - Indiana University

sh-3.00$ 

```

ภาพ 4-14

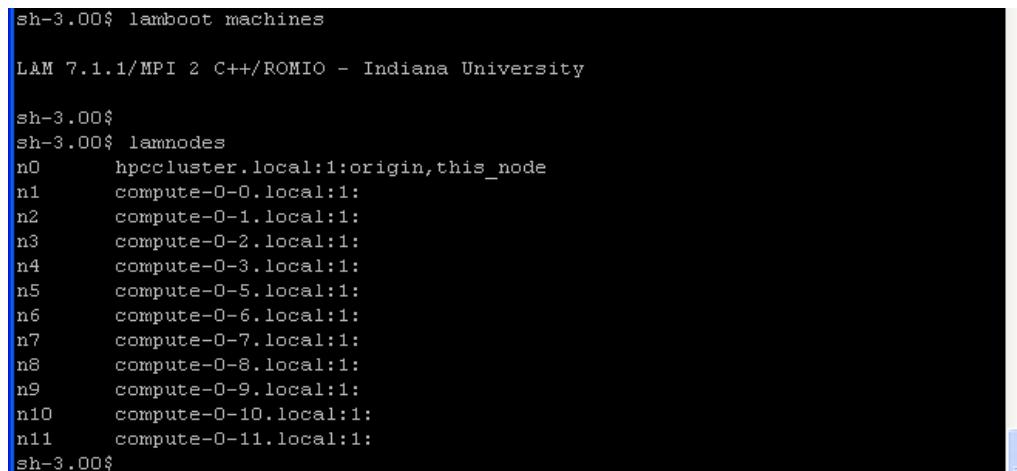
ในอันดับถัดไปเราจะพิจารณาการรันโดยใช้เครื่องหล่ายๆ เครื่องซึ่งทำได้บน hpccluster เท่านั้น ในขั้นแรกนั้นคุณจำเป็นต้องสร้าง hostfile ซึ่งเป็น file ที่ระบุจำนวนเครื่องที่ใช้ในการทำงานทั้งหมด ในระบบ LAM นั้น hostfile จะต้องมีชื่อหรือ IP ของ localhost ที่คุณใช้อยู่ และเป็นตัวสั้นๆ งานเพื่อรันโปรแกรมของคุณอยู่ในนั้นเสมอ จากตัวอย่างในภาพ 4-15 เราตั้งชื่อ hostfile ว่า “machines” ซึ่งมีรายการของ front node และ compute nodes ของระบบ hpccluster ของเราอยู่



```
sh-3.00$ cat machines
hpccluster
compute-0-0
compute-0-1
compute-0-2
compute-0-3
compute-0-5
compute-0-6
compute-0-7
compute-0-8
compute-0-9
compute-0-10
compute-0-11
sh-3.00$
```

ภาพ 4-15

หลังจากนั้นคุณสามารถสร้าง LAM runtime system ได้โดยใช้คำสั่ง “lamboot machines” ดังในภาพ 4-16 คำสั่งนี้จะใช้เวลาสักพัก เมื่อมันทำงานเสร็จแล้วคุณควรลองใช้คำสั่ง lamnodes เพื่อเช็คว่ามี nodes อะไรอยู่ใน LAM runtime system



```
sh-3.00$ lamboot machines
LAM 7.1.1/MPI 2 C++/ROMIO - Indiana University

sh-3.00$
sh-3.00$ lamnodes
n0      hpccluster.local:1:origin,this_node
n1      compute-0-0.local:1:
n2      compute-0-1.local:1:
n3      compute-0-2.local:1:
n4      compute-0-3.local:1:
n5      compute-0-5.local:1:
n6      compute-0-6.local:1:
n7      compute-0-7.local:1:
n8      compute-0-8.local:1:
n9      compute-0-9.local:1:
n10     compute-0-10.local:1:
n11     compute-0-11.local:1:
sh-3.00$
```

ภาพ 4-16

ถ้าคุณต้องการรันโปรแกรมของคุณโดยใช้ compute nodes เท่านั้น คุณสามารถใช้คำสั่ง “mpirun n1-11 –np 11 cpi” (ภาพ 4-17) เพื่อออกคำสั่งให้ LAM รันโปรแกรม cpi ของคุณโดยใช้ nodes n1 ถึง n11 เท่านั้น จะเห็นว่าเครื่อง localhost หรือ hpccluster.cs.tu.ac.th จะไม่ได้ถูกนำมาใช้งาน ในกรณีที่คุณไม่ระบุขอบเขตของการใช้งาน nodes ต่างๆในคำสั่ง mpirun ระบบ LAM/MPI ก็จะจัดการส่ง process ไปรันตาม list ที่มันมีด้วยการกำหนดเครื่องที่จะใช้ของมัน เองดังในคำสั่ง “mpirun N –np 4 cpi” ในภาพ

```

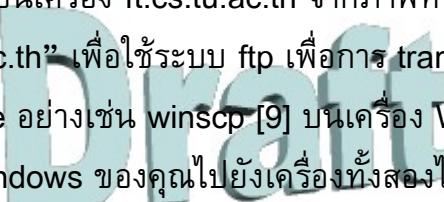
sh-3.00$ mpirun n1-11 -np 11 cpi
Process 2 on compute-0-2.local
Process 8 on compute-0-9.local
Process 1 on compute-0-1.local
Process 4 on compute-0-5.local
Process 3 on compute-0-3.local
Process 7 on compute-0-8.local
Process 10 on compute-0-11.local
Process 6 on compute-0-7.local
Process 9 on compute-0-10.local
Process 5 on compute-0-6.local
Process 0 on compute-0-0.local
pi is approximately 3.1416009869231249, Error is 0.0000083333333318
wall clock time = 0.001293
sh-3.00$
sh-3.00$ mpirun N -np 4 cpi
Process 0 on hpccluster.cs.tu.ac.th
Process 1 on compute-0-0.local
Process 2 on compute-0-1.local
Process 3 on compute-0-2.local
pi is approximately 3.1416009869231245, Error is 0.0000083333333314
wall clock time = 0.000868
sh-3.00$

```

ภาพ 4-17

4.3.3 การ transfer files ไปยัง hpccluster

สมมุติว่าคุณทำงานอยู่บนเครื่อง ft.cs.tu.ac.th จากภาพที่ 18 คุณสามารถใช้คำสั่ง “sftp user01@hpccluster.cs.tu.ac.th” เพื่อใช้ระบบ ftp เพื่อการ transfer files ระหว่างสองเครื่อง จริงแล้วคุณติดตั้ง software อย่างเช่น winscp [9] บนเครื่อง Windows ของคุณคุณก็สามารถ Transfer files จากเครื่อง Windows ของคุณไปยังเครื่องทั้งสองได้โดยตรง



```

user01@ft:~/examples
sh-3.00$
sh-3.00$
sh-3.00$
sh-3.00$ sftp user01@61.91.208.61
Connecting to 61.91.208.61...
Warning: Permanently added '61.91.208.61' (RSA) to the list of known hosts.
user01@61.91.208.61's password:
Permission denied, please try again.
user01@61.91.208.61's password:
sftp>
sftp> ls
examples  lam
sftp> cd lam
sftp> put myprog.c
Uploading myprog.c to /home/user01/lam/myprog.c
myprog.c                                100% 1640      1.6KB/s   00:00
sftp>
sftp> ls
cpi      cpi.c    machines  myprog.c
sftp> get machines
Fetching /home/user01/lam/machines to machines
/home/user01/lam/machines                100%  145      0.1KB/s   00:00
sftp>
sftp> close
Invalid command.
sftp> exit
sh-3.00$

```

ภาพ 4-18

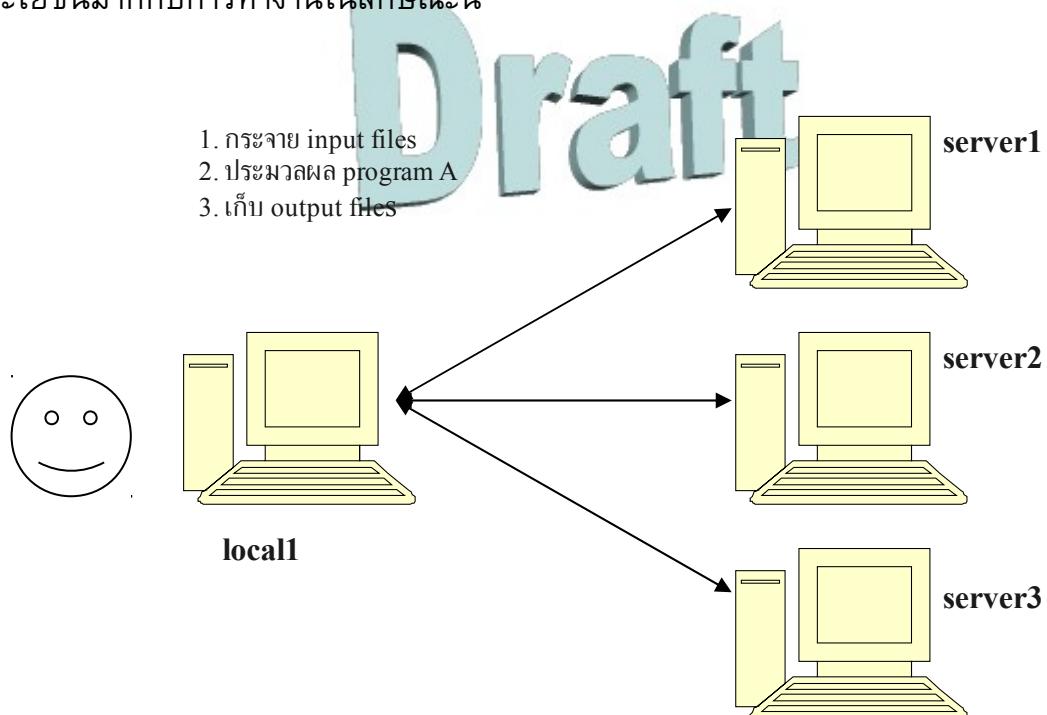
4.4 การสั่งงานเครื่องคอมพิวเตอร์หลายเครื่องผ่าน ssh โดยไม่ใช้รหัสผ่าน

ในการที่จะใช้งานเครื่องคอมพิวเตอร์ที่ใช้ระบบ Unix หลายเครื่องให้ประมวลผลร่วมกันในแบบ parallel computing, distribute computing, หรือ cluster computing ได้นั้น เราจำเป็นต้องสามารถสั่งงานให้เครื่องคอมพิวเตอร์เหล่านั้นประมวลผลหรือถ่ายไฟล์ข้อมูลระหว่างกันได้โดยไม่ต้องใช้ password

ยกตัวอย่างเช่น สมมุติว่าคุณใช้เครื่องคอมพิวเตอร์ชื่อ local1 อยู่และมี เครื่องคอมพิวเตอร์อื่นๆ อีก 3 เครื่องได้แก่เครื่อง server1, server2, และ server3 ดังภาพที่ 1 ข้างล่าง และคุณต้องการที่จะทำงานต่อไปนี้

1. copy ข้อมูล input files จากเครื่อง local1 ไปยัง server1, server2, และ server3
2. หลังจากนั้นต้องการ execute program A บนเครื่องดังกล่าว, และ
3. copy ข้อมูล output files กลับมายังเครื่อง local1 ของคุณ

การทำงานนี้คงจะ naïve มาถ้าคุณต้องพิมพ์ password ทุกครั้งเมื่อต้องการจะทำ remote execution หรือ transfer files ดังนั้นการทำงานร่วมกันโดยไม่ต้องพิมพ์ password จึงมีประโยชน์มากกับการทำงานในลักษณะนี้



ภาพ 4-19

4.4.1 telnet และ rsh

ในการติดต่อเข้าใช้งานเครื่องคอมพิวเตอร์ในระบบเครือข่ายนั้นเราสามารถทำได้หลายวิธี วิธีที่เก่าแก่และเป็นที่รู้จักกันทั่วไปก็คือการใช้โปรแกรม telnet ยกตัวอย่างเช่นถ้าคุณต้องการล็อกอินเข้าใช้เครื่อง server ชื่อ ft.cs.tu.ac.th ซึ่งมี IP คือ 10.0.2.1 คุณก็จะใช้คำสั่ง telnet ft.cs.tu.ac.th หรือ telnet 10.0.2.1 เพื่อทำการดังกล่าว นอกจากคำสั่ง telnet แล้วคุณยังสามารถใช้คำสั่ง rsh (remote shell) หรือ rlogin (remote login) ในระบบ Unix เพื่อ log in เข้าสู่ระบบได้ด้วยยกตัวอย่างเช่น

```
$ rsh ft.cs.tu.ac.th -l kasidit
```

หรือ

```
$ rlogin ft.cs.tu.ac.th -l kasidit
```

โดยที่ kasidit คือ account ของคุณ

ข้อเสียของทั้งสองวิธีนั้นคือมันไม่ปลอดภัย โปรแกรม telnet และ rsh นั้นจะส่งชื่อ account และ password ไปที่เครื่อง server ไปบนระบบเครือข่ายโดยไม่มีการเข้ารหัสซึ่งทำให้ hacker สามารถโนยข้อมูลดังกล่าวจากระบบเครือข่ายได้

ในการที่จะทำงานโดยไม่ใส่ password นั้นเราสามารถใช้ rsh และ rlogin ได้ โดยการสร้าง file ชื่อ .rhosts ไว้ที่ \$HOME directory ของ user บนเครื่อง server และระบุชื่อของ local คอมฯ แต่ก็ไม่ปลอดภัยอีกเพรราะถ้า hacker เข้ามาที่เครื่อง local คอมฯ ได้เขาก็สามารถเข้าใช้เครื่อง server ได้ทันที

4.4.2 การใช้ ssh

เพื่อความปลอดภัยในการใช้งานเครื่อง server และในการสื่อสารระหว่างเครื่อง local คอมพิวเตอร์กับ server เราสามารถใช้โปรแกรม secured shell หรือ ssh ใน การ login โดยใช้คำสั่ง

```
$ ssh kasidit@ft.cs.tu.ac.th
```

ซึ่งจะสร้างการติดต่อที่มีการเข้ารหัสข้อมูลระหว่างโปรแกรม ssh บนเครื่อง local คอมพิวเตอร์ กับโปรแกรม ssh server หรือ sshd ซึ่งทำงานอยู่บนเครื่อง ft.cs.tu.ac.th

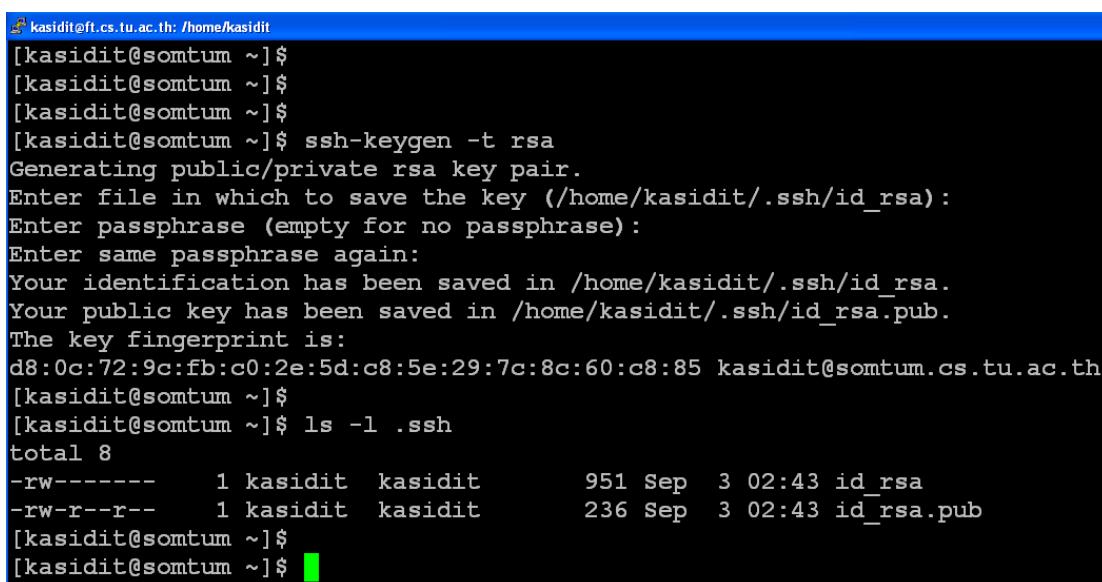
ในการทำงานของ ssh นั้น ssh จะให้บริการการ remote login ผู้ใช้เครื่อง server ในหลายรูปแบบตามการติดตั้งของผู้ใช้ โดยคร่าวๆ ssh จะตรวจสอบว่าผู้ใช้ติดตั้ง ssh-agent ไว้หรือเปล่า? ถ้าไม่มันก็จะให้ ssh-agent ช่วยจัดการการ remote login แต่ถ้าไม่มี ssh-agent มันก็จะดูว่าผู้ใช้ติดตั้ง public และ private keys หรือไม่? ถ้ามีมันก็จะใช้ keys ในการ remote login ถ้าไม่มี มันก็จะถามหา password เป็นทางเลือกสุดท้าย

วิธีการเหล่านี้มีระดับของความปลอดภัยแตกต่างกันไป (จะพูดถึงใน section ถัดไป) แต่อย่างไรก็ตามทุกๆวิธีก็จะส่งข้อมูลระหว่างเครื่องแบบเข้ารหัสซึ่งปลอดภัยกว่า telnet กับ rsh อよู่ดี

4.4.3 การติดตั้งและใช้งาน public และ private keys เพื่อการ remote login

การ login โดยใช้ ssh แบบใช้ public และ private keys นั้นเราต้องติดตั้ง keys ก่อนซึ่งมีขั้นตอนดังตัวอย่างต่อไปนี้

1. สมมุติว่าเครื่อง local คอมฯของคุณชื่อ somtum.cs.tu.ac.th และเครื่อง server ชื่อ ft.cs.tu.ac.th ในขั้นแรกนั้นคุณต้องสร้าง private key และ public key โดยใช้คำสั่ง ssh-keygen



```
kasidit@ft.cs.tu.ac.th: /home/kasidit
[kasidit@somtum ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/kasidit/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/kasidit/.ssh/id_rsa.
Your public key has been saved in /home/kasidit/.ssh/id_rsa.pub.
The key fingerprint is:
d8:0c:72:9c:fb:c0:2e:5d:c8:5e:29:7c:8c:60:c8:85 kasidit@somtum.cs.tu.ac.th
[kasidit@somtum ~]$ ls -l .ssh
total 8
-rw-----    1 kasidit  kasidit      951 Sep  3 02:43 id_rsa
-rw-r--r--    1 kasidit  kasidit     236 Sep  3 02:43 id_rsa.pub
[kasidit@somtum ~]$
```

ภาพ 4-20

จากภาพ 4-20 โปรแกรม ssh-keygen จะสร้าง private key ในรูปแบบ binary และจะให้ user ใส่ข้อความเรียกว่า passphase เพื่อนำไปเข้ารหัส private key นั้นก่อนที่จะจัดเก็บมันลงใน file \$HOME/.ssh/id_rsa โปรแกรม ssh-keygen จะใช้ private key มาสร้าง public key ใน file \$HOME/.ssh/id_rsa.pub ด้วย โปรดสังเกตด้วยว่า permission flag ของ files id_rsa นั้นคือ 600 และของ id_rsa.pub คือ 644 และของ directory .ssh คือ 700 โปรแกรม ssh, scp, และอื่นๆในตระกูลของมันจะเครื่องครัดกับเรื่อง permission มาก

ถ้าคุณไม่ใส่ passphase โปรแกรม ssh-keygen จะไม่เข้ารหัส private key ของคุณใน \$HOME/.ssh/id_rsa ซึ่งไม่ปลอดภัยเพราะถ้า hacker เข้ามาใน account ของคุณใน local คอมฯได้เขาจะสามารถเอา private key ของคุณไปใช้ได้ทันที แต่ถ้าคุณเข้ารหัส passphase ไว้ ครบถ้วนที่เขายังไม่มี passphase เขาก็ไม่สามารถใช้ \$HOME/.ssh/id_rsa ทำอะไรได้

2. คุณต้อง copy public key ของคุณใน file \$HOME/.ssh/id_rsa.pub ไปที่ เครื่อง ssh-server และเก็บข้อมูลลงใน file \$HOME/.ssh/authorized_keys ใน account ของคุณบนเครื่อง ssh-server นั้น

```
[kasidit@ft.cs.tu.ac.th: /home/kasidit/.ssh
[kasidit@somtum ~]$
[kasidit@somtum ~]$ cd $HOME/.ssh
[kasidit@somtum ~/.ssh]$ scp id_rsa.pub kasidit@ft.cs.tu.ac.th:id_rsa_somtum.pub
Warning: Permanently added 'ft.cs.tu.ac.th,203.146.19.24' (RSA) to the list of known hosts.
kasidit@ft.cs.tu.ac.th's password:
id_rsa.pub
[kasidit@somtum ~/.ssh]$ 100% 236      3.3MB/s   00:00
[kasidit@somtum ~/.ssh]$ ssh kasidit@ft.cs.tu.ac.th
kasidit@ft.cs.tu.ac.th's password:
[kasidit@ft kasidit]$ 
[kasidit@ft kasidit]$ mkdir .ssh
[kasidit@ft kasidit]$ chmod 700 .ssh
[kasidit@ft kasidit]$ cd .ssh
[kasidit@ft .ssh]$ 
```

ภาพ 4-21

จากภาพ 4-21 ตอนแรกคุณอยู่บนเครื่อง somtum.cs.tu.ac.th และ cd ไปที่ \$HOME/.ssh และ copy file id_rsa.pub ไปไว้ที่เครื่อง ft.cs.tu.ac.th ภายใต้ชื่อ file id_rsa_somtum.pub ใน \$HOME directory ของ account ของคุณที่นั่น โปรดสังเกตว่า โปรแกรม scp จะถาม password ของคุณเพื่อใช้ในการ access เครื่อง ft.cs.tu.ac.th

3. หลังจากนั้นคุณต้อง log in เข้าสู่เครื่อง ft.cs.tu.ac.th และสร้าง directory .ssh ถ้าไม่มีอยู่แล้ว และต้องเปลี่ยน permission flag ของ .ssh เป็น 700 (or drwx-----) และก็ cd ไปใน .ssh เพื่อ copy file id_rsa_somtum.pub ไปไว้ใน file ชื่อ authorized_keys และเปลี่ยน permission ของ authorized_keys เป็น 600 (หรือ -rw-----) ดังภาพ 4-22

```
[kasidit@somtum ~/.ssh]$ 
[kasidit@somtum ~/.ssh]$ ssh kasidit@ft.cs.tu.ac.th
kasidit@ft.cs.tu.ac.th's password:
[kasidit@ft kasidit]$ 
[kasidit@ft kasidit]$ mkdir .ssh
[kasidit@ft kasidit]$ chmod 700 .ssh
[kasidit@ft kasidit]$ cd .ssh
[kasidit@ft .ssh]$ 
[kasidit@ft .ssh]$ cp ../id_rsa_somtum.pub authorized_keys
[kasidit@ft .ssh]$ 
[kasidit@ft .ssh]$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAOspSariA84XvppWp+1KCJ7YFOAIRc9xrvW/cNb8jXCanBH
Emm5xE94f2i3FvNtG7hff8d/rqA4iAp974l5wxo3VLGbPQpkJ5Pw8ZvCfsZpEqfSqvqro940j0eI9jJxju
hfdtWYPA2iRC0VMvpEa0wNG5Q6bPyhI3M5E= kasidit@somtum.cs.tu.ac.th
[kasidit@ft .ssh]$ 
[kasidit@ft .ssh]$ chmod 600 authorized_keys
[kasidit@ft .ssh]$ 
[kasidit@ft .ssh]$ 
```

ภาพ 4-22

4. logout จากเครื่อง ft.cs.tu.ac.th ดังภาพ 4-23

```
[kasidit@ft .ssh]$  
[kasidit@ft .ssh]$ exit  
logout  
Connection to ft.cs.tu.ac.th closed.  
[kasidit@somtum ~/.ssh]$  
[kasidit@somtum ~/.ssh]$
```

ภาพ 4-23

5. หลังจากนั้นมีคุณ log in เข้าสู่เครื่อง ft.cs.tu.ac.th ด้วยโปรแกรม ssh ดังภาพ 4-24 มันก็จะถามหา *passphase* (ข้อความที่คุณใส่ตอนที่คุณสร้าง keys โดยใช้คำสั่ง ssh-keygen) แทนที่จะเป็น password

```
[kasidit@somtum ~/.ssh]$  
[kasidit@somtum ~/.ssh]$ ssh kasidit@ft.cs.tu.ac.th  
Enter passphrase for key '/home/kasidit/.ssh/id_rsa':  
[kasidit@ft kasidit]$  
[kasidit@ft kasidit]$
```

ภาพ 4-24

ถ้า ssh ยังถามหา password อยู่ นั่นหมายถึงการติดตั้ง files และ directories ต่างๆบนเครื่อง somum.cs.tu.ac.th และ ft.cs.tu.ac.th ยังไม่ถูกต้องให้ไปตรวจสอบชื่อ files และ permission flag ให้เป็นไปตามที่ ssh กำหนด

6. logout ออกมายก ft.cs.tu.ac.th กลับมาที่ somtum ดังภาพ 4-25

```
[kasidit@ft kasidit]$ exit  
logout  
Connection to ft.cs.tu.ac.th closed.  
[kasidit@somtum ~/.ssh]$  
[kasidit@somtum ~/.ssh]$ cd  
[kasidit@somtum ~]$  
[kasidit@somtum ~]$
```

ภาพ 4-25

4.4.4 การใช้งาน ssh-agent เพื่อการ remote login โดยไม่ใช้ passphrase

ถึงจุดนี้คุณได้สร้าง public และ private keys และติดตั้ง keys ทั้งสองเพื่อการ remote login เรียบร้อยแล้ว ในขั้นตัดไปเราจะใช้โปรแกรม ssh-agent เพื่อทำให้คุณสามารถใช้คำสั่ง ssh เพื่อ run command หรือ remote login และใช้คำสั่ง scp เพื่อ copy files ข้ามเครื่องได้โดยไม่ต้องใส่ passphrase

บนเครื่อง local คอมฯ (หรือ somtum.cs.tu.ac.th ตามตัวอย่าง) คุณต้องใช้คำสั่ง

```
$ eval `ssh-agent`
```

หรือ

```
$ ssh-agent tcsh
```

คำสั่งแรกจะ run โปรแกรม ssh-agent เป็น background process และให้โปรแกรม eval ทำการ set environment variables \$SSH_AUTH_SOCK และ \$SSH_AGENT_PID เป็นตามค่าที่ ssh-agent ตั้งขึ้น หลังจากนั้นจะกลับมารอรับคำสั่งจาก shell ตามเดิม ส่วนคำสั่งที่สองจะทำงานคล้ายกันแต่ว่ามันจะ run subshell ตัวใหม่ซึ่งในตัวอย่างข้างบนเป็น tcsh มารอรับคำสั่งจากผู้ใช้ต่อไป คุณสามารถเรียกดูข้อมูลของ ssh-agent ได้โดยใช้คำสั่ง

```
$ ps -fp $SSH_AGENT_PID
```



```
[kasidit@somtum ~] $ ssh-agent tcsh
[kasidit@somtum ~] $ ps -fp $SSH_AGENT_PID
UID      PID  PPID  C STIME TTY          TIME CMD
kasidit  1605  1604  0 03:12 ?        00:00:00 ssh-agent tcsh
[kasidit@somtum ~] $ ssh-add -l
The agent has no identities.
[kasidit@somtum ~] $ ssh-add
Enter passphrase for /home/kasidit/.ssh/id_rsa:
Identity added: /home/kasidit/.ssh/id_rsa (/home/kasidit/.ssh/id_rsa)
[kasidit@somtum ~] $ ssh-add -l
1024 d8:0c:72:9c:fb:c0:2e:5d:c8:5e:29:7c:8c:60:c8:85 /home/kasidit/.ssh/id_rsa (RSA)
[kasidit@somtum ~] $
```

ภาพ 4-26

เมื่อโปรแกรม ssh-agent อย่างเดียวนั้นไม่พอ ในการที่ ssh-agent จะมีประโยชน์ขึ้นมาได้นั้นเราจำเป็นต้อง load private key เข้าไปไว้ใน memory ของมัน ในการ load นี้เราใช้โปรแกรม ssh-add ดังตัวอย่างข้างบน คำสั่ง “ssh-add -l” จะ list finger print ของ private key ที่มีอยู่ใน memory ของ ssh-agent

เราจะใช้คำสั่ง ssh-add เลยๆเพื่อ load private key ไปไว้ใน memory ของ agent จากตัวอย่างข้างบน ssh-add จะไปมองหา private key จาก file \$HOME/.ssh/id_rsa และจะถูก

passphase จากผู้ใช้เพื่อเอาไปถอดรหัสของ private key และเอาไปเก็บไว้ใน memory ของ ssh-agent โปรแกรม ssh-agent จะทำตัวเป็นตัวแทนของผู้ใช้ในการใช้ private key เพื่อทำการพิสูจน์ตัวตนของผู้ใช้กับเครื่อง server ในระหว่างที่มีการใช้งานคำสั่ง ssh และ scp ต่อไป

4.4.5 การใช้คำสั่ง ssh และ scp

หลังจากนี้เมื่อเราใช้คำสั่ง ssh และ scp เราไม่จำเป็นต้องใส่ passphase อีกต่อไป เพราะโปรแกรมทั้งสองจะทำงานร่วมกับ ssh-agent เพื่อให้การสั่งงานข้ามเครื่องเป็นไปโดยอัตโนมัติ การใช้งาน ssh มีรูปแบบ

```
$ ssh account@remote-machine command
```

โดยที่มันจะส่งคำสั่ง command ไปทำงานบนเครื่อง remote-machine ภายใต้ account และแสดง output หรือ error บน screen ยกตัวอย่างเช่น

```
$ ssh kasidit@ft.cs.tu.ac.th "ls -l"
```

จะ run คำสั่ง “ls -l” บนเครื่อง ft.cs.tu.ac.th ภายใต้ account “kasidit” และแสดงผลบน screen



```
$ ssh kasidit@ft.cs.tu.ac.th "cat /etc/abcd"
```

จะ run คำสั่ง “cat /etc/passwd” บนเครื่อง ft.cs.tu.ac.th ภายใต้ account “kasidit” และแสดงผลบน screen

```
$ ssh kasidit@ft.cs.tu.ac.th "cat /etc/abcd" > output-local
```

จะ run คำสั่ง “cat /etc/abcd” บนเครื่อง ft.cs.tu.ac.th ภายใต้ account “kasidit” และ redirect output ไปเก็บใน file output-local บน local คอมฯ

```
$ ssh kasidit@ft.cs.tu.ac.th "cat /etc/passwd > output-remote"
```

จะ run คำสั่ง “cat /etc/abcd” บนเครื่อง ft.cs.tu.ac.th ภายใต้ account “kasidit” และ redirect output ไปเก็บใน file output-remote บน remote คอมฯ (ft.cs.tu.ac.th)

การใช้งาน scp มีรูปแบบ

```
$ scp source-location destination-location
```

และมีการทำงานดังตัวอย่างต่อไปนี้

```
$ scp fileA kasidit@ft.cs.tu.ac.th:fileAA
```

copy file ชื่อ “fileA” จาก local คอมฯ ไปที่เครื่อง remote คอมฯ (ft.cs.tu.ac.th) ไปไว้ใน \$HOME directory ของ account “kasidit” ในชื่อ “fileAA”

```
$ scp kasidit@ft.cs.tu.ac.th:/tmp/fileB mystuffs/fileB
```

copy file ชื่อ “fileB” ใน /tmp directory จากเครื่อง ft.cs.tu.ac.th มาไว้ที่ directory “mystuffs” ในชื่อเดิม

```
$ scp -r mystuffs kasidit@ft.cs.tu.ac.th:mystuffs
```

copy ข้อมูลใน directory mystuffs ทั้งหมดไปไว้ที่ directory mystuffs บนเครื่อง ft.cs.tu.ac.th option “-r” เป็นการระบุลักษณะการ copy แบบ recursion

4.5 ทำให้การใช้ agent และ public และ private key ถึงปลอดภัยกว่า password

การใช้ public และ private keys ในการ login สู่ remote server หรือสั่งงานบน remote server นั้นปลอดภัย กว่าการใช้ password เพราะว่าในการใช้ password นั้นถ้า account name และ password ถูกขโมยไปคนที่ขโมยสามารถใช้ข้อมูลเหล่านี้ login เข้าสู่ remote server ได้

ในการนี้ที่ใช้ public และ private keys นั้นผู้ใช้ต้องใช้ account name, private key, และ passphrase เพื่อ login เข้าสู่ remote server โปรแกรม ssh จะมองหา private key จาก file \$HOME/.ssh/id_rsa ซึ่งถูกเข้ารหัสไว้ด้วย passphrase ดังนั้นในขณะที่ ssh ทำงานมันจึงต้องการ passphrase มาถอดรหัสเพื่อนำ private key มาใช้งาน คนที่ต้องการเข้า account ของคุณต้องมีทั้งสองอย่างนี้ถึงจะ login เข้า remote server ได้

ส่วนในกรณีที่คุณใช้ ssh-agent คนที่ต้องการขโมย private key ต้องสามารถเข้าไปดึงเอาข้อมูลมาจาก memory ของ ssh-agent โดยตรงซึ่งเป็นเรื่องที่ยากมาก

4.4 การทำงานของ ssh

การทำงานของ ssh protocol นั้นผู้เขียนอ้างอิงมาจาก

<http://www.itc.virginia.edu/desktop/security/ssh1.html> และอธิบายการทำงานของมันโดยใช้ตัวอย่างข้างล่าง



จากภาพ 4-27 ssh มีขั้นตอนการทำงานดังนี้

- (1) โปรแกรม ssh สร้าง connection กับโปรแกรม sshd (หรือ ssh server) บน remote คอมฯแล้วส่ง public key จาก file \$HOME/.ssh/id_rsa.pub ไปให้ sshd โดยที่ sshd จะรับ public key ไว้แล้วส่ง public key ของตัวเองกลับไป
- (2) โปรแกรม ssh จะเช็คว่ามันมี public key ของ sshd อุปถัมภ์ไว้หรือไม่ ถ้าไม่มีมันก็จะเอา public key ที่ได้รับมาไปเข้า Hash function เพื่อสร้าง finger print ออกมารอแล้วตามผู้ใช้ ว่าผู้ใช้ยอมรับ finger print ของ remote ssh server อันนี้หรือไม่
- (3) ถ้าผู้ใช้ยอมรับ ssh ก็จะตอบ ok กลับไปให้ sshd ซึ่งหลังจากนั้น sshd จะไปเช็คว่ามี account cs446xx อุปถัมภ์บนเครื่อง remote server หรือไม่ ถ้ามีมันก็จะสร้าง ข้อความ (msg) ขึ้นมาอันหนึ่งเพื่อใช้ส่งระหว่างเครื่องเพื่อทดสอบความถูกต้อง public และ private keys ของทั้งผู้ client และ server

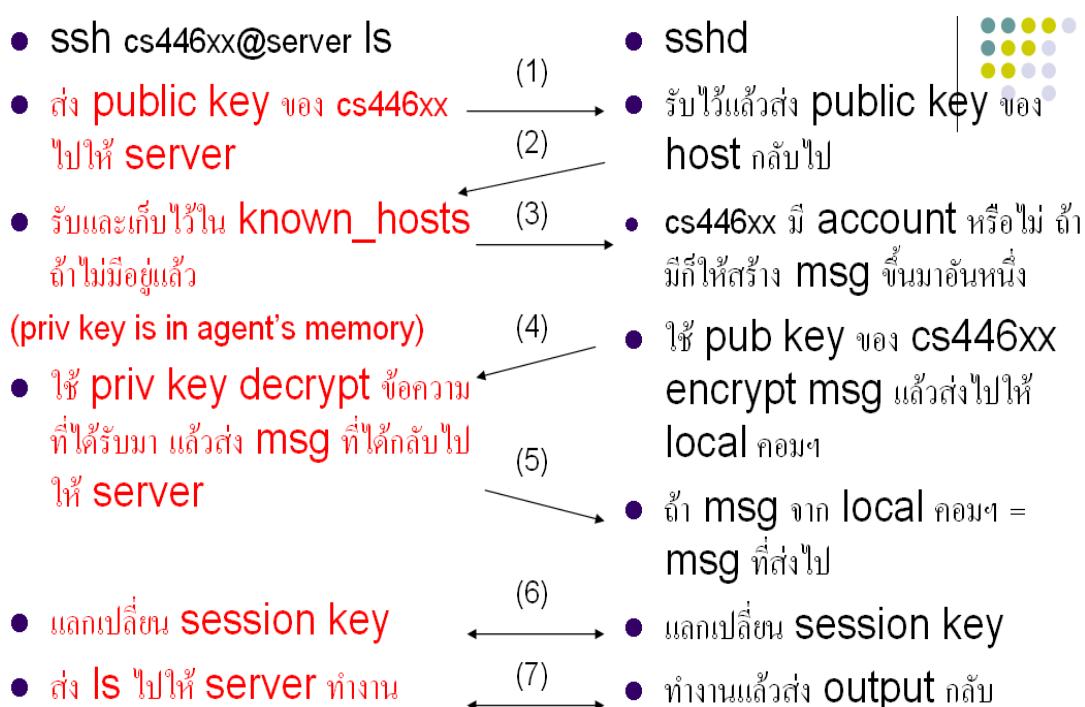
(4) โปรแกรม sshd จะไปอ่านเอา public key ของ cs446xx ออกมายก

~cs446xx/.ssh/authorized_keys file และเขามันไปเข้ารหัส msg แล้วส่ง encrypted msg ไปให้โปรแกรม ssh บน local คอมฯ

(5) โปรแกรม ssh จะไปดึงเอา private key ออกมายก file \$HOME/.ssh/id_rsa บนเครื่อง local คอมฯแต่มันต้องถาม passphase จากผู้ใช้เพื่อนำไปถอดรหัส file \$HOME/.ssh/id_rsa เพื่อให้ได้ private key ที่แท้จริงออกมายังงาน เมื่อได้ private key นั้นมาแล้ว ssh ก็จะใช้มันถอดรหัสของ encrypted msg ที่ได้รับมา จาก sshd และส่งผลลัพธ์ (ซึ่งควรจะเป็นตัว msg) กลับไปให้ sshd โปรแกรม sshd จะนำเข้าข้อความที่ ssh ส่งมาไปเปรียบเทียบกับ msg ที่มันสร้างขึ้นใน step 3 ถ้าเหมือนกันก็แสดงว่า ssh มี private key ที่ถูกต้องและ sshd มีข้อมูล public key ที่ถูกต้อง

(6) หลังจากนั้นทิ้ง ssh และ sshd จะแลกเปลี่ยนข้อมูลกันและสร้าง secret key ขึ้นมาตัวหนึ่งซึ่งเป็นที่รักกันทั้ง 2 ฝั่ง เราเรียกว่า session key ทั้ง ssh กับ sshd จะใช้ session key นี้สำหรับการเข้าและถอดรหัสข้อมูลที่ส่งระหว่างเครื่องทั้งสองต่อไป

(7) จากตัวอย่างนี้ ssh ก็จะใช้ session key เข้ารหัส “ls” และส่งข้อมูลไปให้ sshd ซึ่งจะใช้ session key ของมันถอยดูรหัสข้อมูลที่ส่งมา เมื่อ sshd ได้ “ls” ออกมานั้นก็จะออกคำสั่ง ls บน remote คอมฯ และเอา output ที่ได้ส่งกลับไปให้ ssh ซึ่งในการส่งนี้ sshd และ ssh ก็จะใช้ session key ในการเข้าและถอยรหัสเช่นกัน



ภาค 4-8

ในกรณีที่เราใช้ ssh-agent นั้น ssh จะ ส่งต่อคำสั่งที่มันได้รับให้กับ ssh-agent ซึ่งจะทำงานแทน ssh ตั้งแต่ step ที่ 1 จนจบดังแสดงด้วยตัวอักษรสีแดง

โปรดสังเกตว่าใน step 4 นั้น ssh-agent ไม่จำเป็นต้องถูก ผู้ใช้เพื่อนำไปถอดรหัส private key เพราะว่ามันมี private key ที่ได้รับการถอดรหัสแล้วเก็บไว้ใน memory และพร้อมที่จะใช้งานอยู่แล้ว

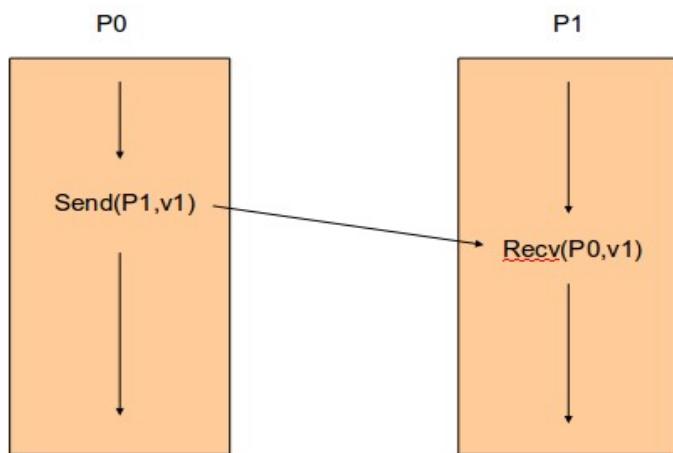
Draft

บทที่ 5 การใช้งาน Message Passing Interfaces

แต่งและเรียบเรียง กษิติ ชาญเชี่ยว

เครื่องมือสำหรับเขียนโปรแกรมแบบ parallel ที่ถือได้ว่าเป็นเสมือนภาษา Assembly ของการเขียนโปรแกรม parallel ก็คือระบบ Message Passing Interfaces หรือ MPI

MPI เป็นเครื่องมือมาตรฐานในการเขียนโปรแกรมที่อยู่ในรูปแบบของ programming library สำหรับ link เข้ากับโปรแกรมแบบ sequential ภาษา C หรือ Fortran โดยที่มันมี Interfaces สำหรับส่งและรับข้อมูลระหว่าง sequential โปรแกรมเหล่านั้น เนื่องจากโปรแกรมทั้งหลายที่สื่อสารกันด้วยการรับส่งข้อมูลในระบบ MPI เป็นโปรแกรมที่แยกจากกันเป็นเดอกเทศ ดังนั้น MPI จึงเหมาะสมสำหรับการสร้าง parallel applications บนระบบ cluster เป็นอย่างยิ่ง เราแสดงลักษณะการสื่อสารพื้นฐานของโปรแกรม MPI ดังภาพที่ 1 คือมี Process สองอันและ Process P0 ส่งข้อมูล v1 ไปให้ process P1 ด้วยคำสั่ง send() และ Process P1 รับข้อมูลจาก P0 โดยใช้คำสั่ง recv()



ภาพ 5-1

5.1 MPI Standard และ ประวัติของ MPI

ก่อนหน้าที่จะมี MPI บริษัทและศูนย์วิจัยต่างๆได้ผลิตเครื่องมือสำหรับทำการสื่อสารแบบ message passing อยู่แล้วหลายอย่างอาทิเช่นระบบ Parallel Virtual Machine (PVM) จาก Oak Ridge National Laboratory และระบบ P4 parallel system จาก Argonne National Laboratory เป็นต้น บริษัทต่างๆ เช่น Intel ก็มีเครื่องมือของตนเอง

เนื่องจากความหลากหลายของเครื่องมือเหล่านี้ การพัฒนา applications จึงต้องขึ้นอยู่กับเครื่องมือใด เครื่องมือหนึ่งโดยเฉพาะและเมื่อต้องการย้าย application ไปรันบนเครื่อง

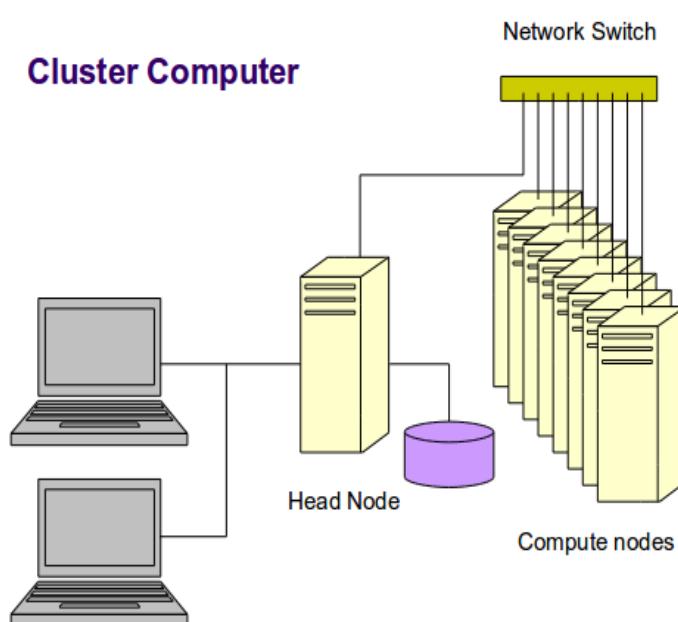
คอมพิวเตอร์ใหม่หรือบนระบบคอมพิวเตอร์ที่มีระบบซอฟแวร์และเครื่องมือสำหรับพัฒนา application ที่แตกต่างกันออกໄไป ก็จะต้องเขียน application ใหม่ทุกครั้งไป นั่นคือ source code ของ application นั้นไม่มี portability นั่นเอง ซึ่งจะทำให้การพัฒนา application มีความล่าช้าและมีการลงทุนที่ซ้ำซ้อน

ด้วยเหตุนี้ผู้ผลิตเครื่องมือที่หลากหลายต่างๆจึงได้มาร่วมมือกันเพื่อกำหนดมาตรฐานของ Message Passing Interfaces หรือ MPI ขึ้นมาซึ่งก็ได้ออกมาถึงสองเวอร์ชัน และมีจำนวน Programming Interfaces สำหรับการสื่อสารแบบ Message passing มากถึงกว่าสองร้อย functions การที่มีมาตรฐาน MPI ทำให้ผู้พัฒนาเครื่องมือสำหรับการทำ message passing สามารถพัฒนาเครื่องมือขึ้นมาสำหรับเครื่องคอมพิวเตอร์หรือระบบซอฟแวร์ระบบที่แตกต่างโดยที่เครื่องมือเหล่านั้นมี Implementation ที่แตกต่างกันแต่จะสนับสนุนการทำงานของ MPI Interfaces ชุดมาตรฐานเดียวกัน

ผลจากการมีมาตรฐาน Interfaces คือ portability ที่สูงขึ้น ทำให้สามารถเขียนโปรแกรม MPI ครั้งเดียวแล้วสามารถนำไปปรับบน MPI implementation เจ้าไหนก็ได้ รายละเอียดเกี่ยวกับ มาตรฐาน MPI หาดูได้ที่ <http://www mpi-forum.org>

5.2 MPI Runtime Environment

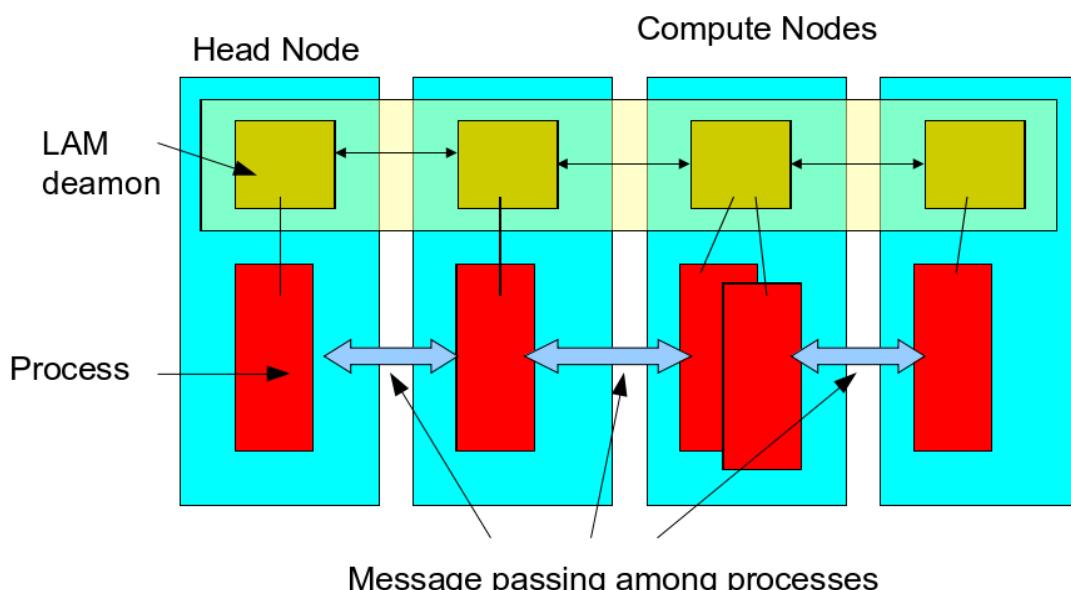
MPI เป็นเครื่องมือที่เหมาะสมกับการใช้งานระบบ Cluster เป็นอย่างยิ่ง โดยที่เราจะรัน process บนเครื่องคอมพิวเตอร์หรือโหนดของระบบคลัสเตอร์ จากภาพของระบบคลัสเตอร์ข้างล่าง ผู้ใช้จะล็อกอินเข้าใช้งานระบบที่เสดโนนดของระบบคลัสเตอร์ซึ่งมี Ethernet Network ภายในที่เชื่อมต่อมันกับโหนดต่างๆ



ภาพ 5-2

ระบบคลัสเตอร์มี software stack ที่ประกอบไปด้วย OS และ middle-ware หรือซอฟแวร์ระบบซึ่งรับนอยู่บน OS และทำหน้าที่เป็นตัวกลางในการประสานงานโหนดต่างๆ เราได้พูดไปแล้วในบทที่ 4

ในที่นี้เราจะพูดถึงสภาพแวดล้อมทางซอฟแวร์ของระบบที่จำเป็นต้องสร้างขึ้นมาก่อนที่จะมีการใช้งานระบบ MPI สภาพแวดล้อมนี้ขึ้นอยู่กับ MPI implementation ซึ่งในวิชานี้เราจะใช้ MPI implementation ที่ชื่อว่า LAM MPI จาก Indiana University จริงๆแล้ว MPI implementation ก็มีอยู่สองค่ายได้แก่ MPI จาก Argonne National Laboratory ซึ่งเป็น MPICH และ OpenMPI ซึ่งเป็นการรวมตัวกันของผู้ผลิต message passing software ที่ได้รับเงินสนับสนุนจากการท่องเที่ยวและงานของประเทศไทย LAM MPI นั้นก็ได้เข้าไปเป็นส่วนหนึ่งของ OpenMPI ในตอนนี้ แต่อย่างไรก็ตาม LAM MPI ก็ยังมีให้ใช้กันอยู่ทั่วไป



ภาพ 5-3

จากภาพ 5-3 ระบบ LAM MPI ประกอบไปด้วยสามส่วนใหญ่ๆ ส่วนแรกคือ LAM runtime system ซึ่งประกอบไปด้วยกลุ่มของ LAM daemon และ command line utility สำหรับออกคำสั่งต่างๆ LAM daemon นั้นเป็น daemon process รันอยู่บนโหนดแต่ละโหนดในระบบคลัสเตอร์และประสานงานกันเพื่อให้รู้ว่าในระบบคลัสเตอร์นั้นมีโหนดที่ทำงานได้อยู่ก็โหนด daemon เหล่านี้จะทำหน้าที่เช็คสภาพของโหนดว่าเป็นอย่างไรและสามารถติดต่อกับโหนดอื่นได้หรือไม่ มันจะประสานงานกับ daemon โดยส่งคำสั่งต่างๆระหว่างกันและเช็คการสื่อสารระหว่างกันและกันเป็นระยะๆ คำสั่งต่างๆที่ใช้ในการสั่งงานระบบ LAM โดยใช้ command line utilities

เช่น lamboot และ lamnodes จะส่งคำสั่งไปยัง LAM daemon บนเครื่องที่มีการออกคำสั่งแล้ว LAM daemon นั้นจะสื่อสารกับ daemon ตัวอื่นๆเพื่อปฏิบัติการตามคำสั่งที่ได้รับมา นอกจากนั้น LAM runtime system ยังเป็นผู้จัดการงานอื่นๆ เช่นการแสดงผลที่ application process ต่างๆ ที่อาจอยู่บนโน๊ตเดียวกันหรือต่างโน๊ตเดพิมพ์สูงน้ำใจ และการรายงานความผิดพลาดต่างๆ ของผู้ใช้ อีกด้วย

ส่วนที่สองคือ MPI Applications ซึ่งประกอบไปด้วย application process จำนวนหนึ่ง รันอยู่บนโน๊ตต่างๆ ของระบบคลัสเตอร์ application process เหล่านี้ได้รับการ link เข้ากับ MPI library เพื่อสนับสนุนการสื่อสารข้อมูลระหว่างกันและติดต่อกับ LAM runtime system เพื่อประสานงานการทำงานแบบ parallel ระหว่าง process เหล่านั้น ผู้ใช้จะออกแบบโปรแกรมของตนและใช้ MPI interfaces เพื่อเพื่อแก้ไขปัญหาต่างๆ ที่เราอาจจะต้องตั้งค่าให้พอดีกับต่อไป

ส่วนสุดท้ายซึ่งนับได้ว่าเป็นหัวใจของระบบ MPI คือ MPI library ซึ่งเป็น implementation ของ MPI interfaces ที่ต้องได้รับการ link เข้ากับ application process ทุกอันของ MPI application โดยหน้าที่หลักของ MPI library นี้คือจัดการการสื่อสารข้อมูลระหว่าง MPI process ที่อยู่ใน MPI application เดียวกัน และติดต่อกับ LAM daemon เพื่อรายงานสถานะการทำงานของ application process ต่อ LAM runtime system

5.3 หลักการพื้นฐานของ MPI

ในการสร้างโปรแกรม MPI นั้นมีหลักการพื้นฐานที่โปรแกรมเมอร์ต้องรู้ได้แก่ เรื่องของ communicator และโครงสร้างของ MPI message และลักษณะของการสื่อสารระหว่าง process ต่างๆ ในระบบ

ก่อนอื่น MPI กำหนดฐานโปรแกรมให้กับโลกแห่งการติดต่อสื่อสารใน MPI application หนึ่งๆ ซึ่งประกอบไปด้วยหลาย process ว่า Communicator ซึ่ง MPI application แต่ละ application จะมี communicator ที่แยกออกจากกัน และข้อความที่ส่งจาก process หนึ่งไปยัง process อื่น จะทำได้ระหว่าง process ที่อยู่ภายใต้ communicator เดียวกันเท่านั้น ด้วยเหตุนี้ การใช้ communicator จึงเป็นการป้องกันความผิดพลาดจากการส่งข้อมูลข้าม MPI applications อีก ด้วย communicator พื้นฐานของ MPI application ทุกๆ อันคือ MPI_COMM_WORLD อย่างไร ก็ตามในการใช้งาน communicator ขั้นสูงนั้นผู้ใช้สามารถสร้าง communicator ย่อยภายใน communicator ได้ และถ้าผู้ใช้ต้องการให้มีการติดต่อระหว่าง communicator จริงๆ สามารถใช้เครื่องมือชื่อ inter-communicator เข้ามาช่วยได้แต่เราจะไม่พูดถึงหัวข้อนี้ในที่นี้

ใน communicator หนึ่งนั้นจะประกอบไปด้วย process จำนวนหนึ่งสมมุติว่าเป็นค่า N และแต่ละ process จะได้รับการกำหนดค่า rank number ซึ่งเป็นค่า Integer เริ่มต้นตั้งแต่ 0 จนถึง N-1

สำหรับเรื่อง message ใน MPI นั้น message ประกอบไปด้วย message envelop และ message body ในส่วนแรกคือ envelop นั้นเป็นเหมือนซองจดหมายที่ประกอบไปด้วยข้อมูลสี่อย่างได้แก่

- sender rank number คือ rank number ของ process ผู้ส่ง
- receiver rank number คือ rank number ของ process ผู้รับ
- tag เป็นค่า integer ที่เป็นค่าที่ผู้ส่งกำหนดมาเป็นการเฉพาะสำหรับ message นั้นๆ
- communicator คือค่าที่แสดงขอบเขตหรือโลกของการสื่อสารสำหรับ process ในกลุ่มเดียวกัน

เมื่อ process ผู้ส่งส่งข้อมูลไปให้ process ผู้รับ ผู้ส่งจะสร้าง message envelop ขึ้นมาและเอาข้อมูลที่ต้องการส่งบรรจุลงใน message body หลังจากนั้นก็จะส่งออกไป เมื่อข้อมูลมาถึง process ผู้รับ มันจะรับข้อมูลนั้นไปใช้ก็ต่อเมื่อเงื่อนไขการรับที่กำหนดใน MPI interfaces สำหรับรับข้อมูลบนฝั่งผู้รับ match กับค่าใน message envelop เท่านั้น ดังที่เราจะได้ดูต่ออย่างในหัวข้อถัดไป

ในอันดับต่อไปการติดต่อสื่อสารระหว่าง process ใน MPI นั้นมีสองแบบคือแบบ point-to-point communication กับ collective communication การติดต่อสื่อสารแบบ point-to-point เป็นการสื่อสารระหว่างสอง process โดยมี process ผู้ส่งและ process ผู้รับโดยที่ผู้ส่งใช้ MPI_send() และผู้รับใช้ MPI_recv() interfaces สำหรับส่งและรับข้อมูลตามลำดับ ส่วน collective communication เป็นการสื่อสารที่ process ทุก process ใน communicator เดียวกันร่วมกันสื่อสารข้อมูลในรูปแบบต่างๆ MPI interfaces สำหรับ collective communication มีหลายอย่างอาทิเช่น MPI_Bcast() MPI_Reduce() MPI_gather() MPI_Scatter() เป็นต้น

A Simple MPI Program

ถึงแม้ว่า MPI จะมี Interfaces มากมายแต่คุณสามารถเขียนโปรแกรม MPI ได้โดยใช้ Interfaces พื้นฐาน 4 Interfaces ดังตัวอย่างในภาพข้างล่าง

```
#include <stdio.h>
#include <mpi.h>
main(int argc, char *argv[])
{
    int s, size, rank;

    s = MPI_Init(&argc, &argv); /* Initialize MPI */
    if (s != MPI_SUCCESS) {
        printf("MPI initialization failed!\n");
        exit(1);
    }
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) { /* root process */
        printf("This is the root process rank=%d\n", rank);
    } else {
        printf("This is a non-root process rank=%d\n", rank);
    }
    MPI_Finalize();
    exit(0);
}
```



ภาพ 5-4

จากภาพเป็นตัวอย่างโปรแกรมภาษา C ที่เป็นโค๊ดที่แต่ละ process ใน MPI application รัน การที่จะทำให้โค๊ดโปรแกรมภาษา C ใช้งานระบบ MPI ได้อย่างน้อยโปรแกรมต้องเรียกใช้ Interface s สองอันคือ mpi_init() เพื่อบอกกับ MPI ว่าโค๊ดส่วนถัดไปของโปรแกรมนั้นจะมีการใช้ระบบ MPI เพื่อการติดต่อสื่อสาร และ mpi_finalize() เพื่อบอกระบบ MPI ว่าการใช้งาน MPI ของโปรแกรมนั้นเสร็จสิ้นแล้ว ส่วน mpi_comm_size() นั้นจะรีเทรินค่าจำนวนของ process ทั้งหมดที่ใน MPI application นี้ และ mpi_comm_rank() นั้นจะรีเทรินค่า rank number คือค่า Id ประจำตัวของ process แต่ละ process ว่ามี Id อะไร

การทำงานของโปรแกรม MPI จะเป็นแบบ Single Program Multiple Data คือเป็นโปรแกรมที่ทุกๆ Process ในหนึ่ง MPI application จะแชร์โค๊ดๆเดียวกัน ในภาพ 5-5 หลังจากที่เขียนและเชฟโปรแกรมในภาพ 5-4 ส్క్రైพ์ฟิล์ชื่อ greeting.c เรียบร้อยแล้ว ผู้ใช้ก็คอมไพล์โปรแกรมโดยใช้คำสั่ง mpicc ซึ่งเป็นคอมไฟล์เลอร์ของ MPI ที่ไปเรียกใช้คอมไฟล์ร์ภาษา C ต่อ ก็ที่หนึ่งและทำการ link โปรแกรมที่คอมไฟล์เข้ากับ MPI library โดยอัตโนมัติ และคอมไฟล์ร์จะสร้างไฟล์ชื่อ greeting เป็น executable file สำหรับโปรแกรมนี้

หลังจากนั้นดเมื่อผู้ใช้ออกคำสั่ง mpirun ในภาพ 5-5 ซึ่งเป็นการออกคำสั่งให้ดลดและ

รัน executable โดยจากไฟล์ greeting เข้าสู่โหนดต่างๆเป็นจำนวน 4 process (ระบุด้วย option “-np 4”) การกำหนดว่า process ใดรันบนโหนดไหนนั้นก็ขึ้นอยู่กับระบบ MPI เมื่อ process ทั้งหมดรัน มันจะพิมพ์ข้อความออกสู่ standard output ของแต่ละ process ซึ่งระบบ MPI จะรวบรวมข้อความเหล่านี้มาพิมพ์ออกสู่เทอร์มินอลที่ผู้ใช้ออกคำสั่งให้รัน mpi application ดังแสดงในภาพ 5-5

```
$ mpicc -o greeting greeting.c
$ mpirun -np 4 greeting
This is a non-root process rank=2
This is the root process rank=0
This is a non-root process rank=1
This is a non-root process rank=3
$
```

ภาพ 5-5

5.4 MPI Point-to-Point Communication

การสื่อสารแบบ point-to-point เป็นการสื่อสารพื้นฐานระหว่าง process สอง process ระบบ MPI ใช้ MPI_Send() interface ในการส่งข้อมูลและมีรายละเอียดของ interface ดังนี้

```
MPI_Send(void *sendbuf,
         int count,
         MPI_Datatype datatype,
         int destination,
         int tag,
         MPI_Comm comm);
```

ภาพ 5-6

จากการ sendbuf คือ memory address ของ variable ที่เก็บข้อมูลที่ต้องการส่ง ส่วน count คือค่าของจำนวน data element ของข้อมูลที่ต้องการส่งนั้น datatype คือชนิดของ data element ของข้อมูลนั้น เราจะระบุค่าของ datatype ตามชนิดของข้อมูลโดยที่ MPI_INT หมายถึงชนิดที่เป็น Integer MPI_FLOAT หมายถึงข้อมูล Floating point number นอกจากนี้ยังมีค่าชนิดต่างๆอีกดังที่กำหนดในรายละเอียดของมาตรฐาน MPI ส่วน destination คือ ค่า rank number ของ process ผู้รับ tag คือค่า integer จำเพาะของข้อมูลที่ส่ง สุดท้ายคือ comm หรือ communicator ซึ่งโดย default แล้วคือ MPI_COMM_WORLD สำหรับทุกๆ MPI application ผู้ส่งจะต้องกำหนดค่าเหล่านี้ทุกค่าเมื่อเรียก MPI_Send() เพื่อส่งข้อมูล

การส่งข้อมูลของ MPI_Send() จะเป็นแบบ FIFO คือถ้าผู้ส่งใช้คำสั่งเดียวกันส่งข้อมูลที่มีค่า rank number และ tag เหมือนกัน มากกว่าหนึ่งครั้ง ข้อมูลเหล่านั้นจะได้รับโดยผู้รับแบบ FIFO คือข้อมูลที่ส่งก่อนจะถูกรับไปใช้งานก่อนข้อมูลที่ส่งมาที่หลังเสมอ

ภาพ 5-7 แสดงชนิดของ datatype พื้นฐานที่ได้รับการกำหนดโดย MPI คุณสามารถสร้าง MPI data type ขึ้นเองได้ด้วยโดยใช้ MPI interfaces สร้างสร้าง user-defined data type ซึ่งเราจะพูดถึงในอนาคต

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
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_INTEGER	INTEGER
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER
MPI_BYTE	
MPI_PACKED	

ภาพ 5-7

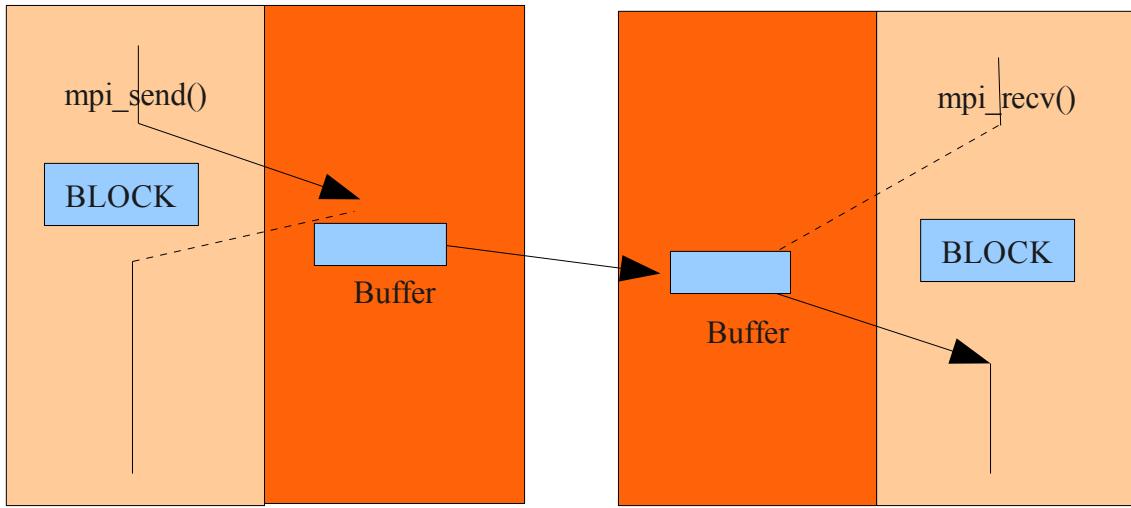
```
MPI_Recv(void *recvbuf,
       int count,
       MPI_Datatype datatype,
       int source,
       int tag,
       MPI_Comm comm,
       MPI_Status status);
```

ภาพ 5-8

บนฝั่ง process ผู้รับ เราต้องเรียก MPI_recv() interface ดังภาพ 5-8 พารามีเตอร์ตัวแรกของ interface นี้คือ recvbuf ซึ่งเป็นค่าของ address ในหน่วยความจำของ process ผู้รับที่ผู้ใช้ต้องการใช้เก็บข้อมูลที่ได้รับมา ส่วนค่า count นั้นเป็นจำนวน element ของข้อมูลที่ recvbuf สามารถรองรับได้ และข้อมูลที่ได้รับเข้ามายังมีชนิดเป็น datatype ซึ่งจะมีค่าเป็นแบบเดียวกันดังที่ได้กล่าวมาแล้วใน MPI_Send() สำหรับ source นั้นผู้ใช้ต้องระบุ rank number ของ process ที่เป็นผู้ส่งข้อมูล และค่า tag ก็เป็นค่า integer จำเพาะของข้อมูลที่จะรับ ส่วน comm ก็คือ communicator ซึ่งค่า default คือ MPI_COMM_WORLD สุดท้ายคือตัวแปร status นั้นจะใช้เก็บค่าสภาวะการทำงานของคำสั่ง MPI_Recv() นี้

ใน parameters เหล่านี้ผู้ใช้จะต้องกำหนดค่าของ recvbuf count datatype และ comm ให้แน่นอน ส่วนค่า source นั้นถ้ากำหนดค่าใดค่าหนึ่งแล้ว MPI_Recv() จะรับค่าเข้าสู่ recvbuf ข้อมูลถูกส่งมาจาก process ที่มีค่า rank number ตรงกับที่ระบุเท่านั้น แต่ในกรณีที่ผู้รับไม่ต้องการจะรับข้อมูลเป็นครั้ง ผู้รับสามารถกำหนดให้ source มีค่าเป็น MPI_ANY_SOURCE ได้ ส่วนค่า tag นั้นก็เช่นเดียวกันคือถ้ามีการกำหนดค่าจะลงเป็นตัวเลข ได้ตัวเลขหนึ่ง MPI_Recv() จะรับค่าข้อมูลเข้าสู่ recvbuf ก็ต่อเมื่อข้อมูลที่ส่งมานั้นมีค่า tag ตรงกับที่ระบุเท่านั้น แต่ถ้าผู้ใช้ต้องการรับข้อมูลที่มีค่า tag เป็นอะไรก็ได้ ผู้ใช้ก็สามารถกำหนด parameter tag ของ MPI_Recv() ให้เป็น MPI_ANY_TAG สำหรับค่าของ status นั้นจะรายงานค่าสภาวะของการทำงานของ MPI_Recv() และรายละเอียดของการรับข้อมูล เช่น ข้อมูล จำเพาะเกี่ยวกับ rank และ tag ของข้อมูลที่ได้รับในกรณีที่ผู้รับใช้ MPI_ANY_SOURCE หรือ MPI_ANY_TAG

สำหรับการทำงานของ MPI interfaces ทั้งสองนั้นเป็นการทำงานในแบบที่เรียกว่า Blocking Buffered communication ซึ่งในระหว่างที่ทั้งสอง Interfaces ทำงานนั้นโปรแกรมจะต้องหยุดรอเป็นระยะเวลาหนึ่งเพื่อให้การทำงานบางอย่างของ Interface ทำงานเสร็จจึงจะทำงานต่อไป การรอตรงนี้เราเรียกว่ามีการ Blocking เกิดขึ้นคือ block การทำงานของโปรแกรม ส่วนที่ว่าเป็น Buffered communication นั้นก็คือการส่งและรับข้อมูลของ Interfaces ทั้งสองจะใช้ Buffer ของ OS ของเครื่องผู้ส่งและเครื่องผู้รับมาช่วยในการส่งข้อมูลระหว่างกัน



ภาพ 5-8

ภาพ 5-8 แสดงลักษณะของ Blocking Buffered communication ในฝั่งของผู้ส่งจะเห็นว่าหลังจากที่โปรแกรมผู้ส่งเรียกใช้ MPI_Send() และ process จะดำเนินการทำงานอยู่ที่นั่นจนกระทั่งข้อมูลซึ่งสุดท้ายจาก sendbuf นั้นได้รับการคัดลอกไปไว้ใน Buffer ของ OS ผู้ส่งจะหมดก่อนแล้วจึงจะทำงานต่อ ส่วนในโปรแกรมผู้รับนั้น เมื่อมีการเรียก MPI_Recv() โปรแกรมจะ block การทำงานอยู่ที่นั่นจนกว่าข้อมูลที่ต้องการจะมาถึง และได้รับการคัดลอกจาก Buffer ของ OS ของเครื่องผู้รับมาไว้ใน variable recvbuf จนหมด และโปรแกรมผู้รับจึงจะทำงานคำสั่งต่อไป

ทั้งนี้ขอให้สังเกตว่าในการส่งข้อมูลแบบ Buffered communication นั้นข้อมูลจะถูกทยอยคัดลอกไปยัง buffer ของ OS ซึ่งจะทำการส่งข้อมูลออก Network ไปยังเครื่องปลายทางอีกต่อหนึ่ง เมื่อข้อมูลเดินทางมาถึงผู้รับแล้วมันจะถูกนำมาเก็บไว้ใน Buffer ของ OS จนกระทั่งโปรแกรมผู้รับทำงานมาถึงคำสั่ง MPI_Recv() และมีการคัดลอกค่าไปไว้ในหน่วยความจำของโปรแกรมผู้รับอีกต่อหนึ่ง ลักษณะการส่งแบบนี้คล้ายคลึงกับ TCP และ MPI implementation MPI ส่วนใหญ่มักจะใช้ TCP เข้ามายัดการการส่งข้อมูล ดังนั้นถ้าข้อมูลมีขนาดใหญ่มากและการจัดส่งข้อมูลของระบบ Network หรือผู้รับทำงานช้า หรือ buffer ของผู้รับเต็ม TCP จะทำ Flow control โดยระงับการส่งข้อมูลเพิ่มเติมจนกว่าโปรแกรมผู้รับจะได้รับข้อมูลที่ส่งไปแล้วก่อนหน้า เป็นสาเหตุหนึ่งที่อาจทำให้เกิดการรอการทำงานข้ามเครื่องและเกิดสถานะการณ์ Deadlock ได้ดังที่เราจะได้กล่าวถึงต่อไป

ภาพ 5-9 แสดงตัวอย่างโปรแกรม MPI ที่มีลักษณะเป็นแบบ Single Program Multiple Data หรือ SPMD คือในโปรแกรมประกอบไปด้วย code สำหรับการทำงานของ Process หลาย Process ในกรณีนี้คือ Process 0 (มีค่า rank number = 0) และ Process 1 โดยที่เราจะใช้ if และ else statements เพื่อแยก code สำหรับแต่ละ Process

โปรแกรมในภาษาจะได้รับการ compile เป็น executable file เดียวและถูกเก็บอยู่ใน shared file system และหลังจากนั้นจึงจะถูก load ขึ้นมาบนโหนดของระบบคลัสเตอร์ที่ MPI เลือกที่จะรันโปรแกรมบนนั้น สมมุติว่ามีการรันโปรแกรมนี้ใน 2 process แต่ละ process จะเริ่มต้นการทำงานโดยเรียก MPI_init() เพื่อ register ตนเองเข้าสู่ MPI runtime system และเรียก MPI_Comm_Rank() ก็จะได้ค่า rank number ของแต่ละ process ใน variable myrank และหลังจากนั้นก็เรียก MPI_Comm_size() ก็จะทำให้ได้จำนวนของ process ทั้งหมดเก็บไว้ใน variable p

หลังจากนั้นเมื่อแต่ละ process ทำงานถึง if statement มันจะแยกการทำงานไปทำใน code ส่วนที่แตกต่างกันคือ Process 0 จะทำงานในส่วนที่เป็น body ของ if ส่วน Process 1 จะไปทำงานในส่วนที่เป็น body ของ else นั่น Process 0 จะเรียก MPI_Send() เพื่อส่งข้อความ "Greeting..." ใน variable message ให้กับ Process 1 และในขณะเดียวกัน Process 1 ก็จะเรียก MPI_Recv() มารับข้อมูลไปไว้ใน variable message ของตน หลังจากนั้น Process ทั้งสองจะเรียก MPI_Finalize() และจบการทำงาน ขอให้สังเกตการกำหนดค่า parameters ต่างๆของ interfaces ทั้งสองจะเห็นได้ว่าเป็นไปตามกฎที่เราได้กล่าวถึงก่อนหน้า



```

#include <stdio.h>
#include <string.h>
#include "mpi.h"

main(int argc, char* argv[]) {
    int      myrank;          /* rank of process      */
    int      p;               /* number of processes */
    int      source;          /* rank of sender       */
    int      dest;            /* rank of receiver     */
    int      tag = 0;          /* tag for messages     */
    char    message[100];     /* storage for message */
    MPI_Status status;        /* return status for   */
                             /* receive              */

    /* Start up MPI */
    MPI_Init(&argc, &argv);

    /* Find out process rank */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* Find out number of processes */
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    if (myrank == 0) {
        /* Create message */
        sprintf(message, "Greetings from process %d!",
               my_rank);
        dest = 1;
        /* Use strlen+1 so that '\0' gets transmitted */
        MPI_Send(message, strlen(message)+1, MPI_CHAR,
                 dest, tag, MPI_COMM_WORLD);
    } else if (myrank == 1) { /* my_rank != 0 */
        source = 0;
        MPI_Recv(message, 100, MPI_CHAR, source, tag,
                 MPI_COMM_WORLD, &status);
        printf("%s\n", message);
    }

    MPI_Finalize();
} /* main */

```

ภาพ 5-9

นอกจากการสื่อสารแบบ blocking และ MPI ยังมีการสื่อสารแบบ non-blocking communication ซึ่งมี interface คล้ายคลึงกับ MPI_Send() และ MPI_Recv() ที่ได้กล่าวถึง ก่อนหน้า เราจะใช้ MPI_Isend() บนผู้ส่งซึ่ง Process ผู้ส่งจะส่งข้อมูลจาก variable sendbuf ที่มีค่าตามที่ระบุใน count และ datatype ไปยัง process ที่มี rank เท่ากับค่าใน destination เมื่อโปรแกรมผู้ส่งเรียก MPI_Isend() แล้วมันจะส่งค่า parameters ต่างๆไปให้ MPI

implementation ให้ทำการส่งต่อให้แล้ว return กลับมาทำงานคำสั่งถัดไปทันทีไม่รอให้การคัดลอกข้อมูลเสร็จสิ้น

```
MPI_Isend(void *sendbuf,  
          int count,  
          MPI_Datatype datatype,  
          int destination,  
          int tag,  
          MPI_Comm comm,  
          MPI_Request req);
```

ภาพ 5-10

จากภาพ 5-10 จะเห็นว่า MPI_Isend() มี parameter สำคัญที่แตกต่างจาก MPI_Send() คือ req ซึ่งจะเก็บค่าของ request ticket สำหรับใช้อ้างอิงเรียกดูสภาพการส่งข้อมูลครั้งนั้นๆ ว่า เสร็จหรือยัง เพราะ MPI_Isend() จะ return ทันที ดังนั้นถ้าโปรแกรมต้องการเช็คว่าการส่งข้อมูลเสร็จหรือยังหรือล้มเหลวด้วยประการใด โปรแกรมผู้ส่งต้องเรียกใช้คำสั่ง MPI_Test() และใส่ req เป็น parameter เพื่อเช็คสถานะของการส่งข้อมูลนั้น MPI_Test() จะเช็คแล้ว return ทันที เช่น กันไม่รอ ถ้าผู้ใช้ต้องการให้รอเพื่อให้การส่งข้อมูลเสร็จ (คือคัดลอกข้อมูลไปยัง buffer ของ OS จนหมด) เราต้องใช้ MPI_Wait() และผ่าน parameter req ให้กับมัน MPI_Wait() จะ block รอจนกว่าทั้งการส่งข้อมูลแบบ Buffered communication เสร็จ ในการส่งข้อมูลแบบ non-blocking นั้นถึงหนึ่งที่ต้องระวังคือตราบได้ที่การคัดลอกค่าจาก sendbuf ไปยัง Buffer ยังไม่เสร็จ จะต้องไม่มีการเปลี่ยนแปลงค่าของ sendbuf ในโปรแกรมผู้ส่ง มีฉะนั้นอาจเกิดความผิดพลาดของข้อมูลที่ส่งได้

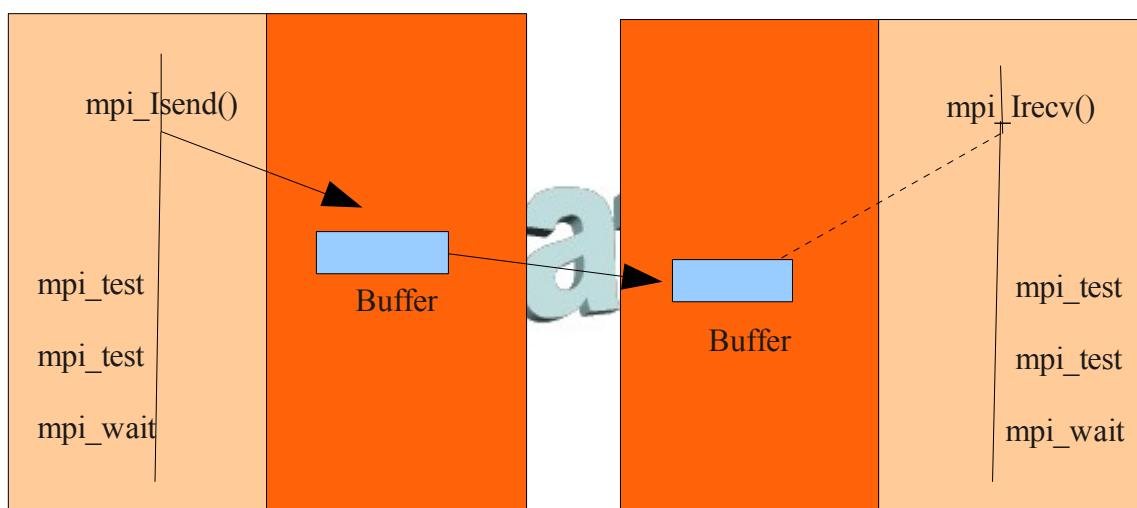
```
MPI_Irecv(void *recvbuf,  
          int count,  
          MPI_Datatype datatype,  
          int source,  
          int tag,  
          MPI_Comm comm,  
          MPI_Request *req);
```

ภาพ 5-11

เช่นเดียวกัน บนฝั่งผู้รับ เราสามารถเรียก MPI_Irecv() เพื่อให้ MPI รับข้อมูลแบบ non-blocking ให้โดยที่ Interface นี้จะส่งข้อมูล parameter ของมันให้กับ MPI implementation และบอกระบบ MPI ทำการรับข้อมูลให้แล้วไม่รอการรับข้อมูลใดๆ ทั้งสิ้น แต่จะ return จากการ

ทำงานของ MPI_Irecv() ไปรันคำสั่งตัดไปทันที สิ่งที่โปรแกรมจะได้รับมาพร้อมกับการ return นี้ ก็คือค่า request ซึ่งเป็นเหมือนกับ ticket ไว้วางอิงการรับข้อมูลที่โปรแกรมนี้ได้ขอให้ MPI จัดการให้ ซึ่ง MPI implementation จะทำการรอให้ข้อมูลมาถึงและคัดลอกข้อมูลไปยัง recvbuf ในหน่วยความจำของโปรแกรมผู้รับโดยอัตโนมัต ในขณะเดียวกันโปรแกรมผู้รับสามารถไปทำงานคำสั่งอื่นๆได้ ทำให้มีการ overlap ระหว่างการทำงานและการติดต่อสื่อสาร

หลังจากนั้น ผู้ใช้สามารถใช้ MPI_test() และให้ค่า req นี้เป็น parameter เพื่อเช็คสถานะของการรับข้อมูลว่าเสร็จสิ้นหรือยัง MPI_test() จะ return ค่าສภาวะการทำงานกลับสู่โปรแกรมผู้รับในทันที แต่ถ้าผู้ใช้ต้องการที่จะให้โปรแกรม block รอจนกว่าการรับจะเสร็จสิ้น ก็ต้องใช้ ? MPI_wait() Interface (และให้ req) เป็น parameter ก็จะทำให้โปรแกรม block รออยู่จนกว่าการรับข้อมูลและคัดลอกข้อมูลจาก Buffer ของ OS มา�ัง recvbuf เสร็จสิ้น สิ่งที่ผู้ใช้ต้องระวังในการใช้งาน MPI_Irecv() คือต้องระวังไม่ให้มีการอ่านหรือเขียนข้อมูลใน recvbuf ในระหว่างที่การรับข้อมูลยังไม่เสร็จสิ้น เพราะอาจทำให้ได้ค่าข้อมูลที่ผิดพลาด



ภาพ 5-12

ภาพ 5-12 แสดงการทำงานของ MPI non-blocking buffered communication โดยที่ การส่งข้อมูลจะเกิดขึ้นโดยที่ MPI จะเป็นผู้จัดการคัดลอกข้อมูลลงสู่ Buffer ของผู้ส่งและส่ง ข้อมูลไปยัง process ผู้รับ การทำงานในฝั่งผู้ส่งจะเป็นแบบ Buffered communication ซึ่งจะ เสร็จสิ้นเมื่อข้อมูลที่ต้องการส่งได้รับการคัดลอกไปสู่ Buffer จนหมด ในขณะเดียวกัน Process ผู้ ส่ง ก็สามารถทำงานอื่นๆได้ บนฝั่งผู้รับก็เช่นเดียวกัน MPI จะทำหน้าที่รับข้อมูลที่ส่งมาจากที่อื่นให Process ผู้รับ และ Process ผู้รับก็สามารถทำงานอื่นๆได้ในเวลาเดียวกัน การรับข้อมูลจะเสร็จ สิ้นเมื่อข้อมูลจาก Buffer ได้รับการคัดลอกมาไว้ใน variable ใน Process ผู้รับเรียบร้อยแล้ว

5.5 Deadlock

ในการส่งข้อมูลแบบ Blocking Buffered communication นั้น ถ้าข้อมูลมีขนาดใหญ่กว่าขนาด Buffer บนฝั่งผู้ส่งและผู้รับ อาจเกิดการ block บนฝั่งผู้ส่งขึ้นได้เนื่องจาก control flow การ block เนื่องจาก control flow นี้จะเสร็จสิ้นเมื่อผู้รับ drain ข้อมูลออกจาก communication channel ที่ใช้ในการส่งข้อมูลระหว่างสอง process นั้น แต่ถ้าบนฝั่งผู้รับไม่มีการรับข้อมูล ผู้ส่งก็จะ block รอจนกระทั่งเกิดการรับขึ้น

การรออันเนื่องมาจาก control flow นี้อาจทำให้เกิด Deadlock ได้ในกรณีติดต่อสื่อสารระหว่าง process ดังตัวอย่างในภาพ 5-13 ที่ Process 0 ส่งข้อมูลไปให้ Process 1 ซึ่งส่งข้อมูลไปให้กับ Process 0 พร้อมๆ กัน ถ้า Process ทั้งสองส่งข้อมูลขนาดใหญ่ มันอาจทำให้เกิดการรอขึ้นที่ MPI_Send() ของทั้งสอง Process และไม่เกิดการรับข้อมูลลักษณะที่ เมื่อเกิดการรอเป็นวงกลม หรือ circular wait ก็จะเกิดสถานะการณ์ deadlock ขึ้น

```
If (myrank == 0) {  
    MPI_Send(sendbuf, count, datatype, 1, tag, comm);  
    MPI_Recv(recvbuf, count, datatype, 1, tag, comm, &status);  
} else {  
    MPI_Send(sendbuf, count, datatype, 0, tag, comm);  
    MPI_Recv(recvbuf, count, datatype, 0, tag, comm, &status);  
}
```



ภาพ 5-13

วิธีการแก้ไขข้อนี้มีหลายวิธี วิธีหนึ่งคือการเปลี่ยนอันดับการส่งและรับข้อมูลของแต่ละ Process ไม่ให้เกิด circular wait ดังในภาพ 5-14 ซึ่ง Process 0 จะส่งข้อมูลไปให้ Process 1 ในขณะที่ process 1 จะรับข้อมูลจาก 0 ก่อนแล้วค่อยส่งข้อมูลให้กับ Process 0 ภายหลัง

```
If (rank == 0) {  
    MPI_Send(sendbuf, count, datatype, 1, tag, comm);  
    MPI_Recv(recvbuf, count, datatype, 1, tag, comm, &status);  
} else {  
    MPI_Recv(recvbuf, count, datatype, 0, tag, comm, &status);  
    MPI_Send(sendbuf, count, datatype, 0, tag, comm);  
}
```

ภาพ 5-14

อีกวิธีหนึ่งคือการใช้ non-blocking communication เข้ามาช่วยในการส่งข้อมูล ในภาพ 5-15 ทั้งสอง process จะส่งข้อมูลโดยใช้ MPI_Isend() interfaces ซึ่งจะบอกให้ MPI ส่งข้อมูลให้แล้วแต่ละ Process ก็จะไปรับคำสั่งถัดไปคือ MPI_Recv() ซึ่งจะเป็นการ drain ข้อมูลออกจาก communication channel ทำให้ไม่เกิด circular wait ขึ้น หลังจากนั้นจะเห็นว่าทั้งสอง process จะเรียกใช้ MPI_Wait() เพื่อรอให้การส่งเสร็จสิ้น ก่อนที่จะไปรับคำสั่งอื่นๆ

```

If (rank == 0) {
    MPI_Isend(sendbuf, count, datatype, 1, tag, comm, &req);
    MPI_Recv(recvbuf, count, datatype, 1, tag, comm);
    MPI_Wait(req, &status);
} else {
    MPI_Isend(sendbuf, count, datatype, 0, tag, comm, &req);
    MPI_Recv(recvbuf, count, datatype, 0, tag, comm);
    MPI_Wait(req, &status);
}

```

ภาพ 5-15

5.6 ตัวอย่าง MPI program ที่ใช้ Point-to-point Communication

ต่อไปเราจะแสดงตัวอย่างของโปรแกรม MPI ซึ่งประกอบไปด้วย P Process โดยที่ P เป็นค่า Integer และเราจะให้ Process 1 ถึง P-1 สร้างข้อมูล "Greeting from..." ขึ้นมาแล้วส่งให้กับ Process 0 จากภาพจะเห็นว่าโปรแกรมเป็นแบบ SPMD และให้ส่วนที่เป็น body ของ if statement เป็นส่วนที่ Process 1 ถึง P-1 รันคือ สร้างข้อความและใช้ MPI_Send() ส่งข้อความนั้นมาให้กับ Process 0 ส่วน code ใน body ของ else statement นั้นก็จะเป็นส่วนที่ Process 0 ใช้รัน เพื่อวนลูปรับข้อมูลโดยใช้คำสั่ง MPI_Recv() และการทำงานใน loop body แต่ละครั้งก็จะรับข้อมูลเริ่มต้นจากข้อมูลจาก Process 1 และพิมพ์ออกหน้าจอ การทำงานใน loop จะทำซ้ำจนมันได้รับและพิมพ์ข้อมูลจาก Process P-1

```

#include <stdio.h>
#include <string.h>
#include "mpi.h"

main(int argc, char* argv[]) {
    int          my_rank;           /* rank of process      */
    int          p;                /* number of processes */
    int          source;           /* rank of sender      */
    int          dest;             /* rank of receiver    */
    int          tag = 0;           /* tag for messages    */
    char         message[100];     /* storage for message */
    MPI_Status   status;           /* return status for   */
                                  /* receive             */

    /* Start up MPI */
    MPI_Init(&argc, &argv);

    /* Find out process rank */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* Find out number of processes */
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    if (my_rank != 0) {
        /* Create message */
        sprintf(message, "Greetings from process %d!",
               my_rank);
        dest = 0;
        /* Use strlen+1 so that '\0' gets transmitted */
        MPI_Send(message, strlen(message)+1, MPI_CHAR,
                 dest, tag, MPI_COMM_WORLD);
    } else { /* my_rank == 0 */
        for (source = 1; source < p; source++) {
            MPI_Recv(message, 100, MPI_CHAR, source, tag,
                     MPI_COMM_WORLD, &status);
            printf("%s\n", message);
        }
    }

    /* Shut down MPI */
    MPI_Finalize();
} /* main */

```