การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคบนระบบพีซีคลัสเตอร์

PARALLEL BITONIC SORTING ON A CLUSTER OF PCs

จุรีพร บุญนิยม

JUREEPORN BOONNIYOM

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2548

ISBN 974-15-1777-7

PARALLEL BITONIC SORTING ON A CLUSTER OF PCs

JUREEPORN BOONNIYOM

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCE

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2005

ISBN 974-15-1777-7

หัวข้อวิทยานิพนธ์	**การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคบนระบบพีซีคลัสเตอร์**
นักศึกษา	**นางสาวจุรีพร บุญนิยม**
รหัสประจำตัว	**46063605**
ปริญญา	**วิทยาศาสตรมหาบัณฑิต**
สาขาวิชา	**วิทยาการคอมพิวเตอร์**
พ.ศ.	**2548**
อาจารย์ผู้ควบคุมวิทยานิพนธ์	**ผศ. ดร. จีรพร ศรีสวัสดิ์**

บทคัดย่อ

การเรียงลำดับข้อมูลเป็นการประมวลผลขั้นพื้นฐานของงานด้านต่างๆ หลายด้านอาทิ งานด้านวิทยาศาสตร์และงานด้านวิศวกรรมศาสตร์ เป็นต้น ดังนั้นการศึกษาวิจัยเกี่ยวกับการเรียงลำดับข้อมูลจึงเป็นที่สนใจอย่างแพร่หลายทั้งด้านทฤษฎีและการประยุกต์ใช้ อย่างไรก็ตามการเรียงลำดับข้อมูลแบบอนุกรม ไม่สามารถตอบสนองความต้องการใช้ข้อมูลได้อย่างเต็มประสิทธิภาพ เนื่องจากปัจจุบันนี้ข้อมูลมีขนาดใหญ่ขึ้นและมีอัตราเพิ่มขึ้นอย่างรวดเร็ว จึงเกิดแนวความคิดที่จะนำการประมวลผลแบบขนานมาประยุกต์ใช้ในการเรียงลำดับข้อมูล งานวิจัยนี้ได้นำเสนอการเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (Bitonic Sort) บนระบบพีซีคลัสเตอร์โดยใช้มาตรฐานภาษาเอ็มพีไอ ซึ่งการเรียงลำดับข้อมูลแบบขนานด้วยวิธีไบโทนิคนี้สามารถแบ่งเวลาที่ใช้ในการทำงานเป็น 2 ส่วน เช่นเดียวกับการประมวลผลแบบขนานโดยทั่วไป คือ เวลาที่ใช้ในการประมวลผลแบบขนานและเวลาที่ใช้ในการติดต่อสื่อสารแบบขนาน ในงานวิจัยนี้จะทำการเพิ่มประสิทธิภาพของเวลาที่ใช้ในติดต่อสื่อสารระหว่างหน่วยประมวลผลในการเรียงลำดับแบบขนานด้วยวิธีไบโทนิค และลดการใช้หน่วยความจำในการเก็บข้อมูลที่ซ้ำซ้อนกันในระบบ ซึ่งเสนอไว้ 4 วิธี คือ 1) วิธีบีเอส (BS) 2) วิธีซีอีบีเอส (CEBS) 3) วิธีเอสอีบีเอส (SEBS) 4) วิธีซีเอสอีบีเอส (CSEBS) โดยสองวิธีแรกเป็นวิธีที่มีผู้เสนอไว้แล้ว ส่วนสองวิธีหลังเป็นวิธีใหม่ที่เสนอขึ้นในวิทยานิพนธ์นี้ นอกจากนั้นทุกวิธีดังกล่าวถูกพัฒนาด้วยภาษาซีและมาตรฐานภาษาเอ็มพีไอบนระบบพีซีคลัสเตอร์เพื่อใช้ในการทดลอง เปรียบเทียบ และหาผลสรุปของงานวิจัย โดยแสดงการเปรียบเทียบวิธีที่เสนอและวิธีเดิมด้วยการประเมินผลจากเวลาที่ใช้ในการเรียงลำดับข้อมูล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) จากผลการทดลองบนระบบพีซีคลัสเตอร์ที่มีขนาด 1 ถึง 8 หน่วยประมวลผล (P) พบว่าวิธีใหม่ที่เสนอซีเอสอีบีเอสมีค่าอัตราการเพิ่มขึ้นของความเร็วเข้าใกล้กรณีอุดมคติมาก คือ 1.9, 3.3 และ 7.8 ในกรณีที่จำนวนหน่วยประมวลผลเท่ากับ 2, 4 และ 8 หน่วยตามลำดับ และเมื่อเปรียบเทียบกับวิธีที่มีอยู่เดิม พบว่าวิธีซีเอสอีบีเอสที่เสนอมีประสิทธิภาพดีกว่าวิธีบีเอสอย่างน้อย 20% และดีกว่าวิธีซีอีบีเอสอย่างน้อย 5%

Thesis	Parallel Bitonic Sorting on a Cluster of PCs
Student	Miss Jureeporn Boonniyom
Student ID	46063605
Degree	Master of Science
Programme	Computer Science
Year	2005
Thesis Advisor	Asst. Prof. Dr. Jeeraporn Srisawat

ABSTRACT

Sorting is one of the most common processing performed by computers in many applications such as scientific and engineering applications. Sorting technique is very popular in both the theoretical study and its widely utilized applications. However sequential sorting on large data sets is very time consuming and cannot properly manage data, which have rapidly growth rate. Hence parallel sorting has been introduced for such sorting applications. The purpose of this research is to present "parallel Bitonic sorting" for the very large data sets on a cluster of PCs by using MPI standard. Like any parallel processing, time complexity of the parallel Bitonic sorting consists of both the parallel computation time and the parallel communication time. This study focuses on the improvement of the communication time of the existing parallel Bitonic sorting and also the efficient used of the memory overhead. In particular, 4 strategies are purposed: 1) the BS (Bitonic Sort) strategy, 2) the CEBS (Communication-Efficient Bitonic Sort) strategy, 3) the SEBS (Space-Efficient Bitonic Sort) strategy and 4) the CSEBS (Communication-Space Efficient Bitonic Sort) strategy. The first two methods are the existing ones, whereas the last two strategies are introduced in this thesis. In order to measure system performance, all four methods were developed by using C language and MPI standard on the cluster of PCs. In such cluster environment, the system performance of new parallel Bitonic sorting (SEBS and CSEBS strategies) and existing Bitonic sorting (BS and CEBS strategies) were compared in terms of response time, speedup, and efficiency. On the cluster system of size up to 8 processors (P), the experimental results showed that the CSEBS strategy yielded nearly ideal speedup, which were 1.9, 3.3 and 7.8 for P = 2, 4 and 8, respectively. In addition, the CSEBS method performed the improved performance at least 20% over that of the original BS method and at least 5% over that of the existing CEBS method.

กิตติกรรมประกาศ

วิทยานิพนธ์นี้มิอาจจะสำเร็จลุล่วงไปได้ด้วยดี หากมิได้รับคำแนะนำ คำชี้แจง ความรู้ และความเอาใจใส่จาก ผศ. ดร. จีรพร ศรีสวัสดิ์ ผู้เป็นอาจารย์ที่ปรึกษา ซึ่งท่านได้สละเวลาให้กับข้าพเจ้าอย่างเต็มที่ จึงใคร่ขอขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ รศ. ดร. วีระ บุญจริง ดร. กรกช ประชุมรักษ์ และดร. เฉลิมศักดิ์ เลิศวงศ์เสถียร คณะกรรมการสอบหัวข้อและโครงร่างวิทยานิพนธ์ที่กรุณาให้คำแนะนำตลอดจนข้อชี้แนะจนในที่สุดทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลงได้

ขอขอบพระคุณ รศ. ดร. มนัส สังวรศิลป์ ผู้อำนวยการสำนักวิจัยและบริการคอมพิวเตอร์ที่ได้สนับสนุนในเรื่องของระบบคลัสเตอร์ เพื่อใช้ในการทดลองในวิทยานิพนธ์ฉบับนี้

ขอขอบพระคุณ มารดา ที่สนันสนุนให้ได้เรียนในระดับที่ได้ตั้งใจ อีกทั้งยังได้ดูแลเรื่องค่าใช้จ่ายต่าง ๆ ระหว่างการศึกษาเป็นอย่างดีอีกด้วย

ขอขอบคุณ นายนพรัตน์ พันธุ์เสนา และนายไพโรจน์ สมุทรักษ์ ที่ให้คำปรึกษาในเรื่องของการใช้งานระบบคลัสเตอร์และช่วยอำนวยความสะดวกในด้านต่างๆ

ขอขอบคุณ นางสาวแก่นจันทร์ ธรรมรักษ์ และนายนพดล ปลื้มเปรม ที่คอยให้ความช่วยเหลือในเรื่องต่างๆ และให้กำลังใจในการทำงานมาโดยตลอด

สำหรับคุณงามความดีและประโยชน์อันใดที่เกิดขึ้นจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดา มารดา อาจารย์ทุกท่านซึ่งเป็นที่เคารพรักยิ่ง ตลอดจนญาติพี่น้อง และเพื่อน ๆ ทุกคน

จุรีพร บุญนิยม

พ.ค. 2548

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ หน้า

สารบัญตาราง (ต่อ)

ตารางที่ หน้า

สารบัญรูป

สารบัญรูป (ต่อ)

รูปที่ หน้า

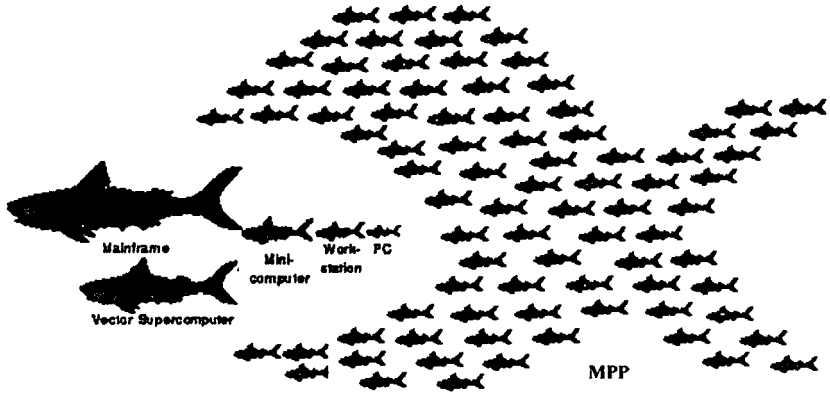
บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

คอมพิวเตอร์แบบขนาน (Parallel Computer) เป็นระบบคอมพิวเตอร์ซึ่งประกอบด้วยหน่วยประมวลผลหลายๆ หน่วยจำนวนมากรวมอยู่ภายในเครื่องเดียวกัน เพื่อประมวลผลงานขนาดใหญ่ไปพร้อมๆ กัน [10] แนวความคิดเริ่มแรกในการพัฒนาระบบคอมพิวเตอร์แบบขนานคือ เพื่อใช้ในการประมวลผลงานทางด้านวิทยาศาสตร์และวิศวกรรมศาสตร์ ซึ่งงานเหล่านี้ต้องการผลที่รวดเร็วและทันต่อเวลา ตัวอย่างเช่น การประมวลผลเพื่อการพยากรณ์อากาศ การคำนวณผลภาพถ่ายทางดาวเทียม การวิเคราะห์ผลภาพสแกน 3 มิติทางการแพทย์ การจำลองแบบทางชีววิทยา เป็นต้น ระบบคอมพิวเตอร์แบบขนานโดยทั่วไปแบ่งได้เป็น 2 ประเภทใหญ่ๆ คือ แบบใช้หน่วยความจำร่วมกัน (Shared-Memory Parallel System) และแบบกระจายหน่วยความจำ (Distributed-Memory Parallel System)

ในปัจจุบันระบบคลัสเตอร์ (Cluster System) เป็นระบบคอมพิวเตอร์แบบขนานชนิดหนึ่งที่ได้รับความนิยมสูงเพราะราคาไม่แพง โดยระบบนี้มีการนำเครื่องคอมพิวเตอร์ส่วนบุคคล (Personal Computer: PC) หรือเวิร์กสเตชั่น (Workstation) ที่มีประสิทธิภาพสูงหลายๆ เครื่องมาเชื่อมต่อกันด้วยเครือข่ายที่มีความเร็วสูง [31] เนื่องจากปัจจุบันการพัฒนาเทคโนโลยีของชิบ (Chip) ในเครื่องคอมพิวเตอร์ส่วนบุคคลหรือเวิร์กสเตชั่น เพื่อให้มีประสิทธิภาพมากขึ้นนั้น ได้มีการพัฒนาที่เปลี่ยนแปลงตลอดเวลาและมีราคาถูก ทำให้ผู้ใช้สามารถเปลี่ยนระบบใหม่ได้ภายในระยะเวลาสั้นๆ เช่น ทุกๆ 3 ปี ต่างจากเครื่องขนาดใหญ่เช่น เมนเฟรมคอมพิวเตอร์ (Mainframe Computer) ซึ่งมีราคาแพงและไม่สามารถซื้อทดแทนได้ภายในระยะเวลา 5 ปี เมื่อเร็วๆ นี้นักวิจัยทางด้านสถาปัตยกรรมคอมพิวเตอร์ [20] ได้เสนอความสัมพันธ์ของระบบคอมพิวเตอร์ทุกประเภทที่มีอยู่ในปัจจุบันและแนวโน้มของระบบคอมพิวเตอร์แบบขนาน ได้อย่างน่าสนใจดังรูปที่ 1.1

ในแผนภาพดังกล่าวมีความหมายโดยรวมดังนี้ เครื่องคอมพิวเตอร์ขนาดใหญ่อาทิ ซุปเปอร์คอมพิวเตอร์ (Super Computer) เมนเฟรมคอมพิวเตอร์ (Mainframe Computer) และมินิคอมพิวเตอร์ (Minicomputer) ซึ่งเป็นเครื่องคอมพิวเตอร์ที่ปกติมีหน่วยประมวลผลเพียง 1-2 หน่วยประมวลผล และมีราคาแพง อาจจะถูกแทนด้วยเครื่องคอมพิวเตอร์แบบขนาน (Massive Parallel Processor : MPP) ซึ่งประกอบด้วยหน่วยประมวลผลเป็นจำนวนมาก



รูปที่ 1.1 แนวโน้มของระบบคอมพิวเตอร์แบบขนาน (MPP: Massive Parallel Processor) [20]

ปัจจุบันนี้คอมพิวเตอร์แบบขนานไม่ได้ถูกใช้งานอย่างแพร่หลาย เหตุผลสำคัญประการหนึ่งคือ การเขียนโปรแกรมเพื่อการประมวลผลแบบขนานมีความซับซ้อนมากกว่าการเขียนโปรแกรมทั่วๆ ไป แต่สำหรับระบบคลัสเตอร์ได้มีการใช้งานและพัฒนาโปรแกรมที่มีความซับซ้อนน้อยกว่าการพัฒนาโปรแกรมแบบขนานทั่วๆ ไป โดยโปรแกรมภาษาแบบขนานสำหรับระบบคอมพิวเตอร์ชนิดนี้เป็นโปรแกรมภาษาที่ใช้ได้กับทุกระบบคลัสเตอร์ เช่น พีวีเอ็ม (PVM: Parallel Virtual Machines) [7] และ เอ็มพีไอ (MPI: Message Passing Interface) [8] ซึ่งมีลักษณะการเขียนโปรแกรมแบบเอสพีเอ็มดี (SPMD: Single Program over Multiple Data) หรือการเขียนโปรแกรมแบบเอ็มพีเอ็มดี (MPMD: Multiple Programs over Multiple Data) ที่ง่ายกว่าการเขียนโปรแกรมแบบเอสไอเอ็มดี (SIMD: Single Instruction over Multiple Data) หรือการเขียนโปรแกรมแบบเอ็มไอเอ็มดี (MIMD: Multiple Instructions over Multiple Data) [15] ของระบบคอมพิวเตอร์แบบขนานทั่วไป

ในด้านการประมวลผลแบบขนานซึ่งสามารถเพิ่มประสิทธิภาพในการประมวลผลสำหรับข้อมูลปริมาณมากๆ ปัจจุบันนี้ทุกหน่วยงานมีการจัดเก็บข้อมูลเป็นจำนวนมาก โดยใช้คอมพิวเตอร์จัดเก็บในรูปแบบเรียงลำดับ (Sorting) ตัวอย่างเช่น ในส่วนของหน่วยงานราชการ มีการจัดเก็บข้อมูลต่างๆ หลายชนิดเช่น ข้อมูลทะเบียนราษฎร ข้อมูลงานวิจัย ข้อมูลทางด้านสถิติ หรือแม้แต่ภาคเอกชนที่ดำเนินธุรกิจในเชิงพาณิชย์ก็ยังมีความจำเป็นในการจัดเก็บข้อมูลต่าง ๆ อาทิ ข้อมูลของธุรกิจการสื่อสาร ข้อมูลของธุรกิจการเงินการธนาคาร ซึ่งข้อมูลต่างๆ ดังกล่าวมีเป็นจำนวนมากและมีอัตราเพิ่มขึ้นอย่างรวดเร็วเป็นแบบเอ็กโพเนนเชียล (Exponential Growth Rate) ทำให้การเรียงลำดับข้อมูลแบบอนุกรม (Sequential Sorting) บนเครื่องคอมพิวเตอร์ที่มีหน่วยประมวลผลเดียว (Sequential Computer) ใช้เวลานาน เป็นเหตุให้ประสิทธิภาพในการประมวลผลลดลง จึงเกิดแนวความคิดที่จะนำเทคนิคของการประมวลผลแบบขนาน (Parallel Computing) บนเครื่องคอมพิวเตอร์ที่มีหลายหน่วยประมวลผล (Parallel Computer) มาใช้ในการเรียงลำดับข้อมูลเพื่อช่วยลดเวลาและเพิ่มประสิทธิภาพในการประมวลผลข้อมูลดังกล่าว

งานวิจัยนี้ได้นำเสนอการเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (Bitonic Sort) [2] [14] [19] [26] ที่มีประสิทธิภาพมากขึ้นบนระบบพีซีคลัสเตอร์ (Cluster of PCs) ซึ่ง การเรียงลำดับข้อมูลแบบขนานด้วยวิธีไบโทนิคนี้สามารถแบ่งเวลาที่ใช้ในการทำงานเป็น 2 ส่วน เช่นเดียวกับการประมวลผลแบบขนานโดยทั่วไป คือ เวลาที่ใช้ในการประมวลผลแบบขนาน (Parallel Computation Time) และเวลาที่ใช้ในการติดต่อสื่อสารแบบขนาน (Parallel Communication Time) ในงานวิจัยนี้จะทำการเพิ่มประสิทธิภาพของเวลาที่ใช้ในติดต่อสื่อสารระหว่างหน่วยประมวลผลในการเรียงลำดับแบบขนานด้วยวิธีไบโทนิค และลดการใช้หน่วยความจำในการเก็บข้อมูลที่ซ้ำซ้อนกันในระบบ โดยแสดงการเปรียบเทียบประสิทธิภาพด้วยการประเมินผลจากเวลาที่ใช้ในการเรียงลำดับข้อมูล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) ซึ่งจะพัฒนาโปรแกรมการเรียงลำดับแบบขนานด้วยภาษาซี (C Language) และมาตรฐานภาษาเอ็มพีไอ (MPI Standard) บนระบบพีซีคลัสเตอร์ เพื่อใช้ในการทดลองและหาผลสรุปของงานวิจัย

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

วิทยานิพนธ์นี้มีวัตถุประสงค์เพื่อศึกษาวิธีการเรียงลำดับข้อมูลแบบขนานด้วยวิธีไบโทนิคแบบเดิมที่มีอยู่และนำเสนอวิธีการเพิ่มประสิทธิภาพของการเรียงลำดับด้วยวิธีไบโทนิคแบบใหม่ดังนี้

1) เพิ่มประสิทธิภาพในการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล โดยจะทำการลดการแลกเปลี่ยนข้อมูลในบางช่วงเวลาที่ไม่จำเป็นออกจากกระบวนการเรียงลำดับ

2) ช่วยลดการใช้พื้นที่ในหน่วยความจำที่ไม่จำเป็น เนื่องจากวิธีไบโทนิคแบบเดิมจะใช้พื้นที่ในหน่วยความจำเก็บข้อมูลที่มีความซ้ำซ้อนอยู่ในระบบ

1.3 สมมติฐานของการศึกษา

การเพิ่มประสิทธิภาพ ข้อแรกการลดปริมาณข้อมูลที่ต้องมีการแลกเปลี่ยนกันในบางช่วงเวลาที่ไม่จำเป็น จะทำให้เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลลดลง และการเพิ่มประสิทธิภาพ ข้อสองโดยการใช้พื้นที่ในหน่วยความจำของระบบลดลง ซึ่งจะช่วยให้การเรียงลำดับข้อมูลใช้เวลาลดลง และเพิ่มประสิทธิภาพในการทำงานมากขึ้น

1.4 ขอบเขตการวิจัย

วิทยานิพนธ์นี้มีขอบเขตของการวิจัย ซึ่งแบ่งเป็น 2 ส่วนดังนี้

<u>ส่วนที่ 1</u> ด้านทฤษฎี

ทำการศึกษาการเรียงลำดับแบบขนานด้วยวิธีไบโทนิค แล้วนำวิธีการดังกล่าวมาปรับปรุงเพื่อเพิ่มประสิทธิภาพในการทำงานดังนี้

1) การเพิ่มประสิทธิภาพในการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล

2) การเพิ่มประสิทธิภาพโดยทำการลดการใช้พื้นที่ในหน่วยความจำลดลง

<u>ส่วนที่ 2</u> ด้านการประยุกต์ใช้งาน

นำทฤษฎีที่ได้ทำการศึกษาในส่วนแรกมาพัฒนาโดยใช้ภาษาซี (C Language) และมาตรฐานภาษาเอ็มพีไอ (MPI Standard) ที่สนับสนุนการโปรแกรมแบบขนานบนระบบพีซีคลัสเตอร์ (Cluster of PCs) และผลลัพธ์ที่ได้จากการทดลองใช้เป็นแนวทางในการวิเคราะห์เพื่อหาผลสรุป

1.5 ขั้นตอนการศึกษาและการดำเนินงานวิจัย

วิทยานิพนธ์นี้มีขั้นตอนการศึกษาและการดำเนินงานวิจัย ดังนี้

1) ศึกษาขั้นตอนวิธี (Algorithm) การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค

2) ศึกษางานวิจัยที่เกี่ยวข้องกับการเรียงลำดับแบบขนาน

3) ทำการตั้งสมมติฐานโดยคาดว่าการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคที่ได้นำเสนอสามารถลดการใช้พื้นที่ในหน่วยความจำและลดเวลาที่ใช้ในการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล ส่งผลให้การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคที่นำเสนอมีประสิทธิภาพเพิ่มขึ้น

4) นำเสนอวิธีการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคที่มีประสิทธิภาพ ดังที่ได้ตั้งสมมติฐานไว้ในข้อ (3)

5) พัฒนาโปรแกรมการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบเดิมและแบบที่นำเสนอโดยใช้ภาษาซี (C Language) บนระบบพีซีคลัสเตอร์ที่ใช้ระบบปฏิบัติการลีนุกซ์ (Linux) โดยใช้มาตรฐานภาษาเอ็มพีไอ (MPI Standard) ที่สนับสนุนการโปรแกรมแบบขนาน

6) วิเคราะห์ผลการทดลองโดยการเปรียบเทียบประสิทธิภาพของการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคนี้จะประเมินผลจากเวลาที่ใช้ในการเรียงลำดับข้อมูล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency)

7) สรุปผลการทดลอง พร้อมเสนอแนวทางการพัฒนางานวิจัย

8) เขียนวิทยานิพนธ์

1.6 ประโยชน์ที่คาดว่าจะได้รับ

วิทยานิพนธ์นี้มีประโยชน์ที่คาดว่าจะได้รับจากการเรียงลำดับข้อมูลแบบขนานด้วยวิธีไบ-โทนิคบนระบบพีซีคลัสเตอร์ ดังนี้

1) ลดเวลาที่ใช้ในแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล ในขั้นตอนของการเรียงลำดับข้อมูลแบบขนานด้วยวิธีไบโทนิค

2) ลดการใช้พื้นที่ในหน่วยความจำเก็บข้อมูลที่ซ้ำซ้อน ในขั้นตอนของการเรียงลำดับข้อมูลแบบขนานด้วยวิธีไบโทนิค

3) นำมาใช้เป็นแนวทางในการพัฒนาโปรแกรมแบบขนานบนระบบพีซีคลัสเตอร์เพื่อให้สามารถใช้ได้สะดวกและเข้าใจได้ง่ายมากขึ้น

4) นำมาใช้เป็นแนวทางในการพัฒนาระบบคอมพิวเตอร์แบบขนานชนิดคลัสเตอร์ สำหรับหน่วยงานหรือสถานศึกษาที่ต้องการมีระบบคลัสเตอร์ของตนเองในราคาที่ไม่แพง

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 สถาปัตยกรรมคอมพิวเตอร์แบบขนาน (Parallel Computer Architecture)

สถาปัตยกรรมคอมพิวเตอร์แบบขนานนี้สามารถจำแนกได้หลากหลายรูปแบบขึ้นอยู่กับสิ่งที่เราให้ความสนใจเช่น องค์ประกอบของพื้นที่การใช้งาน (Address-Space Organization) การติดต่อสื่อสารระหว่างหน่วยประมวลผล (Interconnection Network) ขนาดและจำนวนของหน่วยประมวลผล (Granularity of Processors) ในปี 1972 Flynn's [10] [25] ได้จำแนกสถาปัตยกรรมของคอมพิวเตอร์ออกเป็น 4 แบบ คือ

1) สถาปัตยกรรมแบบเอสไอเอสดี (SISD: Single Instruction over Single Data) คือ เครื่องคอมพิวเตอร์แบบทั่วไปที่ใช้หน่วยประมวลผลเพียงหนึ่งหน่วยประมวลผลเท่านั้น ซึ่งมีลักษณะการประมวลผลเป็นแบบการประมวลผลชุดคำสั่งทีละหนึ่งคำสั่งกับหนึ่งข้อมูล

2) สถาปัตยกรรมแบบเอสไอเอ็มดี (SIMD: Single Instruction over Multiple Data) คือ ระบบคอมพิวเตอร์แบบขนานสำหรับใช้งานเฉพาะด้าน ซึ่งมีลักษณะการประมวลผลเป็นการประมวลผลชุดคำสั่งทีละคำสั่งกับหลายข้อมูล ในหลายหน่วยประมวลผลพร้อมกัน

3) สถาปัตยกรรมแบบเอ็มไอเอสดี (MISD: Multiple Instructions over Single Data) คือ ระบบคอมพิวเตอร์แบบขนานสำหรับใช้งานเฉพาะด้าน ซึ่งมีลักษณะการประมวลผลเป็นแบบการประมวลผลชุดคำสั่งทีละหลายคำสั่งกับหนึ่งข้อมูล ในหลายหน่วยประมวลผลพร้อมกัน

4) สถาปัตยกรรมแบบเอ็มไอเอ็มดี (MIMD: Multiple Instructions over Multiple Data) คือ ระบบคอมพิวเตอร์แบบขนานสำหรับงานทั่วๆ ไป ซึ่งมีลักษณะการประมวลผลเป็นแบบการประมวลผลชุดคำสั่งทีละหลายคำสั่งกับหลายข้อมูล ในหลายหน่วยประมวลผลพร้อมกัน ซึ่งได้รับความนิยมอย่างสูงในปัจจุบัน

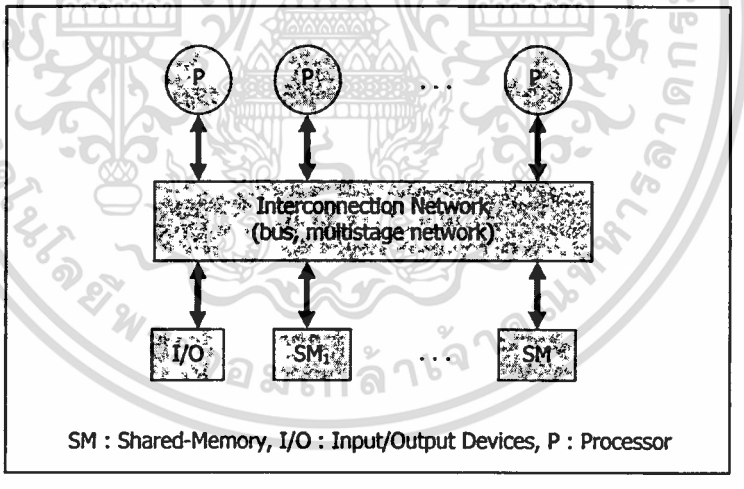
งานวิจัยนี้ให้ความสนใจที่สถาปัตยกรรมแบบเอ็มไอเอ็มดี (MIMD) ซึ่งจะเน้นที่โครงสร้างของหน่วยความจำโดยสามารถแบ่งได้เป็น 2 แบบดังนี้

2.1.1 สถาปัตยกรรมแบบขนานที่ใช้หน่วยความจำร่วมกัน (Shared-Memory Parallel Architecture)

สถาปัตยกรรมของระบบคอมพิวเตอร์แบบขนานชนิดนี้ถูกออกแบบมาให้มีการใช้หน่วยความจำร่วมกัน [10] [15] โดยที่หน่วยประมวลผลทั้งหมดในระบบ (Processors) สามารถเข้าถึงข้อมูลในหน่วยความจำ (Memory) ของระบบได้ทุกที่พร้อมๆ กัน ทำให้การพัฒนาโปรแกรมทำได้สะดวกมากขึ้น อย่างไรก็ตามจำนวนของหน่วยประมวลผลที่สามารถเข้าถึงข้อมูลในหน่วยความจำนั้นได้ถูกจำกัดด้วยขนาดของเส้นทางการติดต่อสื่อสาร จากปัญหาดังกล่าวได้มีความพยายามที่จะแก้ไขปัญหานี้โดยกำหนดให้แต่หน่วยประมวลผลมีหน่วยความจำส่วนตัว (Local Memory) และมีหน่วยความจำกลาง (Global Memory) ที่ใช้เก็บข้อมูลร่วมกัน ซึ่งโดยทั่วไปแล้วเราสามารถแบ่งสถาปัตยกรรมแบบใช้หน่วยความจำร่วมกันได้ 3 โมเดล คือ

2.1.1.1 ยูเอ็มเอโมเดล (UMA Model: Uniform Memory-Access Model)

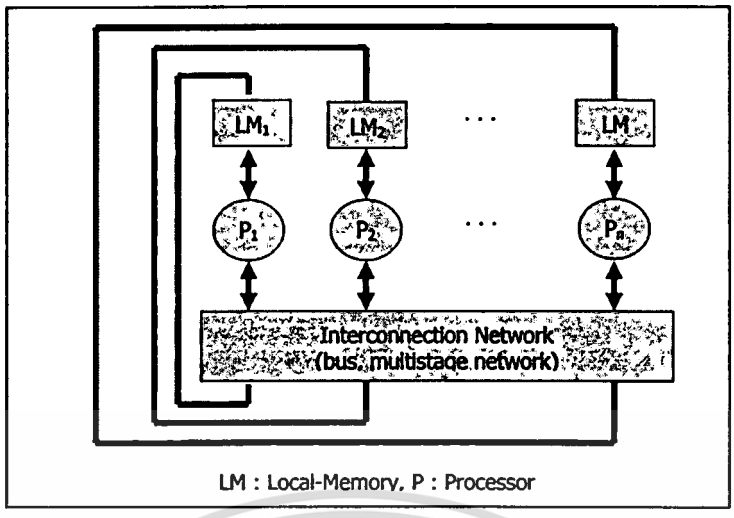
สถาปัตยกรรมแบบที่ใช้หน่วยความจำร่วมกันในยูเอ็มเอโมเดล (UMA Model) [10] นี้เป็นโมเดลที่ทุกๆ หน่วยประมวลผลสามารถเข้าถึงหน่วยความจำใดได้ด้วยเวลาที่เท่าๆ กัน และจำนวนของหน่วยประมวลผลไม่จำเป็นต้องเท่ากับจำนวนของหน่วยความจำ ดังรูปที่ 2.1



รูปที่ 2.1 ยูเอ็มเอโมเดล (UMA Model)

2.1.1.2 เอ็นยูเอ็มเอโมเดล (NUMA: Non-Uniform Memory Access Model)

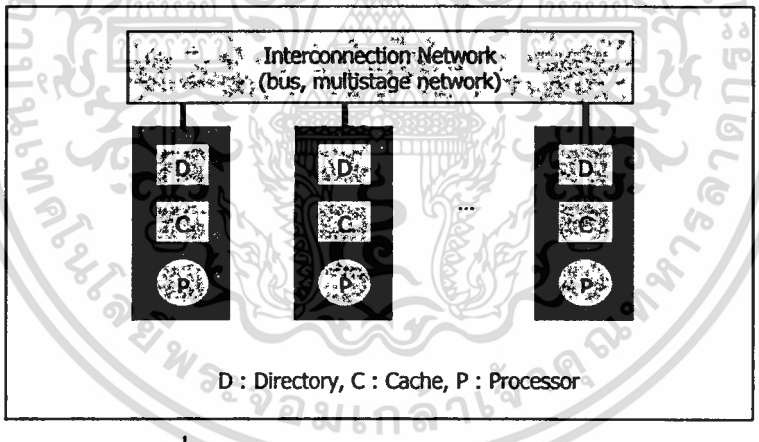
สถาปัตยกรรมแบบที่ใช้หน่วยความจำร่วมกันในเอ็นยูเอ็มเอโมเดล (NUMA Model) [10] เป็นโมเดลที่มีหน่วยความจำส่วนตัวที่ใช้ร่วมกัน (Local Shared Memory: LM) โดยที่การเข้าถึงข้อมูลในหน่วยความจำส่วนตัวที่ใช้ร่วมกันของตัวเองจะทำได้เร็วกว่าการเข้าถึงข้อมูลในหน่วยความจำส่วนตัวที่ใช้ร่วมกันของหน่วยประมวลผลอื่น ดังรูปที่ 2.2



รูปที่ 2.2 เอ็นยูเอ็มเอโมเดล (Non-UMA Model)

2.1.1.3 ซีโอเอ็มเอโมเดล (COMA: Cache-Only Memory Access Model)

สถาปัตยกรรมแบบที่ใช้หน่วยความจำร่วมกันในซีโอเอ็มเอโมเดล (COMA Model) [10] เป็นโมเดลที่มีการนำแคช (Cache) เข้ามาใช้แทนหน่วยความจำ (Memory) ทำให้การเข้าถึงข้อมูลเร็วขึ้นดังรูปที่ 2.3

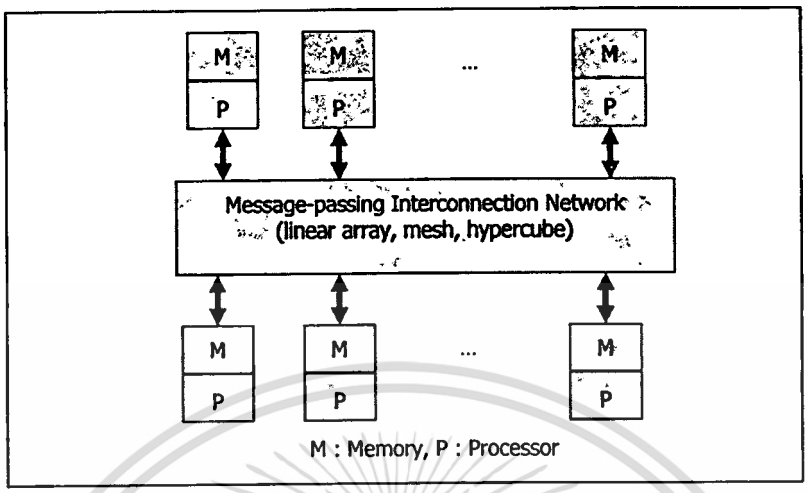


รูปที่ 2.3 ซีโอเอ็มเอโมเดล (COMA Model)

2.1.2 สถาปัตยกรรมแบบขนานที่มีการกระจายหน่วยความจำ (Distributed-Memory Parallel Architecture)

สถาปัตยกรรมของระบบคอมพิวเตอร์แบบขนานชนิดนี้ออกแบบมาให้ใช้หน่วยความจำแบบกระจาย [10] [15] คือ หน่วยประมวลผลแต่ละหน่วยจะมีหน่วยความจำส่วนตัว (Local Memory) ที่ไม่สามารถใช้ร่วมกันดังรูปที่ 2.4 ดังนั้นในระบบนี้ถ้าหน่วยประมวลผลใดต้องการข้อมูลจากหน่วยประมวลผลอื่น ต้องติดต่อสื่อสารกันผ่านระบบส่งผ่านข้อความ (Message Passing) โดยการพัฒนาโปรแกรมแบบขนานบนระบบนี้จะต้องทำการรับส่งข้อมูล (Send/Receive) ผ่าน

ไลบรารี (Libraries) ของเอ็มพีไอ (MPI: Message Passing Interface) [8] [18] หรือพีวีเอ็ม (PVM: Parallel Virtual Machine) [7] เป็นต้น



รูปที่ 2.4 สถาปัตยกรรมแบบกระจายหน่วยความจำ

นอกจากนี้แล้วสถาปัตยกรรมแบบเอ็มไอเอ็มดี (MIMD) ยังสามารถแบ่งออกเป็นแบบย่อยๆ ได้อีกเป็นแบบซิมเมทริกซ์มัลติโปรเซสเซอร์ (Symmetric Multi-Processor: SMP) [9] ซึ่งโดยทั่วไปสถาปัตยกรรมแบบนี้เป็นคลัสเตอร์ (Cluster) ขนาดใหญ่ที่ใช้หน่วยความจำแบบกระจายและมีหลายคลัสเตอร์ย่อยภายในคลัสเตอร์ย่อยนี้จะจับกลุ่มหน่วยประมวลผลจำนวน 4-8 หน่วยประมวลผลที่มีประสิทธิภาพเท่ากันซึ่งจะใช้หน่วยความจำร่วมกัน ในทางปฏิบัตินั้นคลัสเตอร์แบบเอสเอ็มพี (SMP Cluster) จะเลือกใช้หน่วยความจำแบบกระจาย ทำให้การพัฒนาโปรแกรมแบบเอสเอ็มพีคลัสเตอร์และแบบกระจายหน่วยความจำมีความคล้ายคลึงกันมาก

2.1.3 สถาปัตยกรรมแบบคลัสเตอร์ (Cluster Architecture)

แนวความคิดของระบบคลัสเตอร์เกิดขึ้นในปี ค.ศ. 1960 โดยบริษัท IBM ต้องการเชื่อมต่อระบบเมนเฟรมคอมพิวเตอร์ (Mainframe Computer) ขนาดใหญ่เข้าด้วยกันเพื่อใช้ในเชิงการค้า แต่ในตอนนั้นระบบคลัสเตอร์ยังไม่เป็นที่รู้จักอย่างแพร่หลายเนื่องจากข้อจำกัดทางด้านราคาและความแพร่หลายของฮาร์ดแวร์ (Hardware) นอกจากนั้นยังขาดเครื่องมือ (Tools) มาตรฐานสำหรับการเชื่อมต่อเครื่องในระบบเข้าด้วยกัน จนในช่วงกลางปี 1980 เป็นช่วงที่เทคโนโลยีระบบคลัสเตอร์สามารถเกิดขึ้นได้เนื่องจากสามารถสร้างไมโครโปรเซสเซอร์ที่มีประสิทธิภาพสูง และเกิดระบบเครือข่ายความเร็วสูงขึ้น ซึ่งเครื่องคอมพิวเตอร์ในระบบเครือข่ายสามารถสื่อสารกันได้ภายในเวลาไม่เกินหนึ่งมิลลิวินาที นอกจากนั้นความต้องการเพื่อประมวลผลงานทางวิทยาศาสตร์และวิศวกรรมศาสตร์ซึ่งมีมากขึ้นอย่างต่อเนื่องก็มีส่วนผลักดันให้เกิดเทคโนโลยีคลัสเตอร์ขึ้น [31]

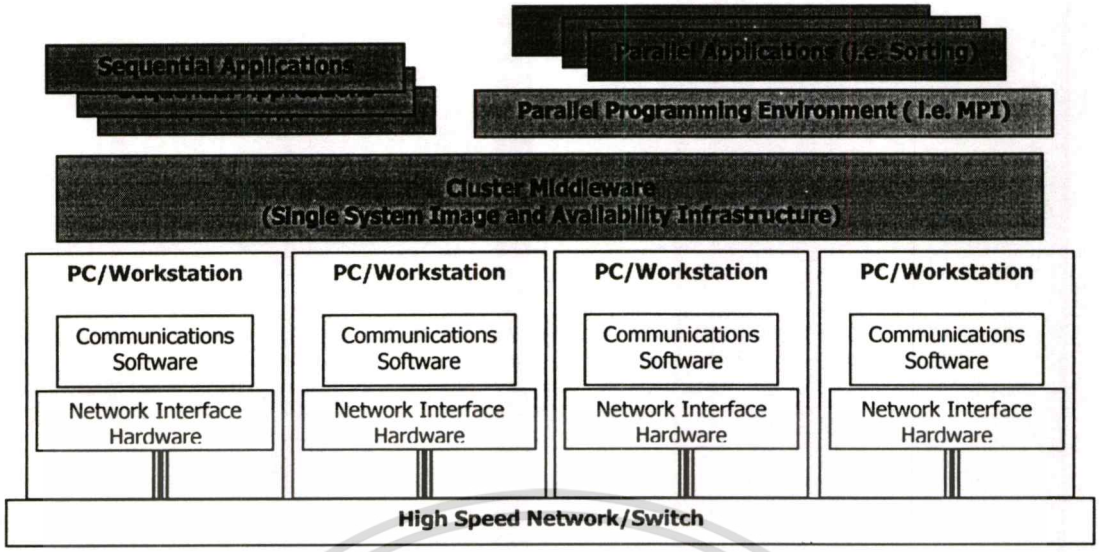
ระบบคลัสเตอร์ได้เกิดขึ้นอย่างเป็นรูปธรรมในปี 1994 เมื่อ Thomas Sterling และ Don Becker จาก CESDIS (The Center of Excellence in Space Data and Information Sciences) ต้องการที่จะวิเคราะห์ข้อมูลจากอวกาศที่มีจำนวนมากและซับซ้อนซึ่งต้องใช้เครื่องคอมพิวเตอร์ที่สามารถประมวลผลได้เร็วมากๆ และในขณะนั้นมีเพียงเครื่องซูปเปอร์คอมพิวเตอร์ (Super Computer) ซึ่งมีราคาแพงแต่เนื่องจากงบประมาณที่มีอยู่จำกัดทำให้ Sterling และ Becker มีแนวความคิดที่จะใช้ระบบคลัสเตอร์มาทำการประมวลโดยได้สร้างระบบคลัสเตอร์จากเครื่องคอมพิวเตอร์ซึ่งใช้ชิพ Intel DX4 เป็นหน่วยประมวลผลกลาง (CPU) และส่วนประกอบอื่นๆ ซึ่งสามารถหาได้ง่ายจากท้องตลาดและได้ตั้งชื่อระบบคลัสเตอร์นี้ว่า "Beowulf" [31]

ระบบคลัสเตอร์ [30] คือ สถาปัตยกรรมย่อยชนิดหนึ่งของสถาปัตยกรรมแบบขนานที่มีการกระจายหน่วยความจำ โดยระบบนี้เป็นการนำคอมพิวเตอร์ขนาดเล็กเช่น เครื่องคอมพิวเตอร์ส่วนบุคคล (Personal Computer) หรือเวิร์กสเตชั่น (Work Station) หลายๆ หน่วยมาเชื่อมต่อกันเพื่อให้ทำงานเหมือนเครื่องใหญ่เครื่องเดียว ความแตกต่างระหว่างระบบคลัสเตอร์กับระบบแลน (LAN: Local Area Network) คือ ระบบแลนจะเป็นการเชื่อมต่อคอมพิวเตอร์จำนวนมากเพื่อใช้ทรัพยากรร่วมกันโดยแต่ละเครื่องยังเป็นเครื่องอิสระและทำงานอิสระกัน แต่ระบบคลัสเตอร์เป็นการรวมเครื่องหลายๆ เครื่องที่ประสานงานกันอย่างแน่นแฟ้นจนเปรียบเสมือนเป็นเครื่องเดียวกันและสามารถทำงานได้ทั้งแบบขนานกันและแบบต่ออิสระกัน ในปัจจุบันระบบคลัสเตอร์สามารถนำมาประยุกต์ใช้งานได้แทนระบบเซิร์ฟเวอร์ขนาดใหญ่สำหรับประมวลผลงานอิสระทั่วๆ ไป และสำหรับประมวลผลงานขนาดใหญ่แบบขนานได้เป็นอย่างดีในราคาที่ถูก

2.1.3.1 ส่วนประกอบโดยรวมของคอมพิวเตอร์แบบคลัสเตอร์

เราอาจกล่าวได้โดยง่ายว่าระบบคลัสเตอร์คือ กลุ่มของคอมพิวเตอร์ที่มีการเชื่อมต่อผ่านเครือข่ายความเร็วสูงและสามารถคำนวณงานที่ถูกแบ่งออกเป็นงานย่อย ๆ แล้วร่วมกันได้ ซึ่งรูปแบบของเครือข่ายและการเชื่อมต่อนี้มีด้วยกันหลายวิธีและหลายชนิดเครือข่าย ส่วนการที่จะทำให้กลุ่มของคอมพิวเตอร์สามารถทำงานร่วมกันได้นั้นจะต้องมีซอฟท์แวร์เป็นตัวกลางในการเชื่อมโยงการทำงานของหน่วยประมวลผล แต่ละหน่วยเข้าด้วยกัน [27] ซึ่งมาตรฐานที่นิยมใช้คือ ระบบการส่งผ่านข้อความ (Message Passing Interface) และคลัสเตอร์มิดเดิลแวร์ (Cluster Middle Ware)

ส่วนประกอบโดยรวมทั้งหมดของระบบคอมพิวเตอร์แบบคลัสเตอร์นั้นจะประกอบไปด้วยส่วนต่าง ๆ ดังรูปที่ 2.5 [30]



รูปที่ 2.5 แสดงส่วนประกอบของระบบคลัสเตอร์

จากรูปที่ 2.5 สามารถที่จะแบ่งการทำงานออกได้ทั้งหมดสามส่วนด้วยกัน คือ

1) โครงสร้างพื้นฐานของระบบคลัสเตอร์ ซึ่งประกอบไปด้วยส่วนประกอบทางฮาร์ดแวร์ การเชื่อมต่อเครือข่าย ระบบปฏิบัติการ และคลัสเตอร์มิดเดิลแวร์

2) เครื่องมือและโปรแกรมอัตถะประโยชน์ ประกอบด้วย โปรแกรมสื่อสารระหว่างโหนดโดยการส่งข้อความ คอมไพเลอร์ และชุดคำสั่งทางคณิตศาสตร์

3) การจัดการระบบคลัสเตอร์ ประกอบด้วยตัวจัดการคิวงาน การจัดการสิทธิของผู้ใช้งานในระบบ และระบบตรวจสอบและจัดการคลัสเตอร์

2.1.3.2 ชนิดของคลัสเตอร์

ระบบคลัสเตอร์ สามารถจัดแบ่งเป็น 2 ชนิดตามลักษณะที่ต่อระบบเข้ากับเครือข่ายความเร็วสูงภายนอก (High-Speed Network) คือ

1) คลัสเตอร์แบบปิด (Closed Cluster) ระบบนี้จะต่อคลัสเตอร์ผ่านเกตเวย์ (Gateway) ที่ซ่อนทั้งระบบจากโลกภายนอก ข้อดีก็คือมีความปลอดภัยสูงและจะใช้หมายเลขอินเตอร์เน็ตแอดเดรส (IP Address) เพียงแอดเดรสเดียวเท่านั้น ข้อเสียคือแต่ละโหนดในระบบไม่สามารถช่วยกันบริการข้อมูลกับโลกภายนอกได้ ดังนั้นคลัสเตอร์แบบปิดจึงเหมาะกับการใช้ทำงานคำนวณขนาดใหญ่โดยเฉพาะงานที่ต้องการประมวลผลแบบขนาน เช่น การคำนวณภาพถ่ายทางดาวเทียม การคำนวณทางชีววิทยา เป็นต้น

2) คลัสเตอร์แบบเปิด (Open Cluster) ระบบนี้จะต่อกับเน็ตเวิร์คภายนอกโดยตรง ทำให้ผู้ใช้เข้าถึงทุกโหนดในระบบคลัสเตอร์ได้โดยตรง ข้อเสียก็คือ ความปลอดภัยจะต่ำกว่าแบบ

ปิดมากเพราะต้องคอยดูแลทุกเครื่องในระบบ และยังต้องการหมายเลขอินเตอร์เน็ตแอดเดรสจำนวนมาก แต่ข้อดีก็คือระบบสามารถช่วยกันบริการข้อมูลได้ ดังนั้นคลัสเตอร์แบบเปิดจึงเหมาะกับงานบริการข่าวสารจำนวนมากเช่นใช้เป็นระบบเซิร์ฟเวอร์ สำหรับเวิร์ดไวท์เวบ (WWW) หรือเอฟทีพี (FTP) ที่ขยายตัวได้ ระบบนี้น่าจะเหมาะกับการทำเป็นฐานข้อมูลขนาดใหญ่ด้วย แต่ในขณะนี้ยังไม่มีซอฟท์แวร์แวร์ที่จะประยุกต์เป็นฐานข้อมูลที่ใช้ขีดความสามารถของระบบคลัสเตอร์ได้

2.2 การเรียงลำดับ (Sorting)

โจทย์ปัญหาส่วนใหญ่ทางคอมพิวเตอร์ทั้งที่เป็นทฤษฎีขั้นตอนวิธีและการประยุกต์ใช้ จะเกี่ยวข้องอย่างมากกับการเรียงลำดับข้อมูลชนิดตัวเลขจากน้อยไปหามากหรือจากมากไปหาน้อย ตลอดจนการจัดลำดับกลุ่มของตัวอักษรต่างๆ การเรียงลำดับนี้มีประโยชน์โดยตรง ในการประยุกต์ใช้เพื่อแก้ปัญหาเกี่ยวกับการใช้งานฐานข้อมูล โดยเฉพาะอย่างยิ่งการค้นหาข้อมูลที่ต้องการภายในระยะเวลาอันรวดเร็ว นอกจากนี้การเรียงลำดับข้อมูลยังมีความสำคัญต่องานด้านต่างๆ ทางธุรกิจ ทางวิทยาศาสตร์ และวิศวกรรมศาสตร์ ที่ประมวลผลด้วยคอมพิวเตอร์ ดังนั้นการเรียนรู้เทคนิคการเรียงลำดับจึงจำเป็น เพื่อ

1) สามารถเลือกวิธีการเรียงลำดับที่เหมาะสมกับงานที่ทำมากที่สุด
2) เป็นแนวทางในการคิดค้นวิธีการเรียงลำดับใหม่ ๆ ที่ดียิ่งขึ้น (ไม่ซ้ำแบบเก่า)

นอกจากนี้การเรียงลำดับยังมีประโยชน์อื่นๆ อีกหลายประการดังนี้

1) ช่วยในการจัดหมวดหมู่ข้อมูล (Classify) ให้สะดวกขึ้น
2) ช่วยในการค้นหาข้อมูล (Searching) ให้เร็วขึ้น
3) ช่วยให้การจับคู่ข้อมูล (Matching) เช่น การปรับปรุงแฟ้มลำดับมีประสิทธิภาพยิ่งขึ้น

การเรียงลำดับข้อมูลในปัจจุบันสามารถแบ่งได้เป็น 2 แบบใหญ่ๆ คือ

2.2.1 การเรียงลำดับแบบอนุกรม (Sequential Sort)

การเรียงลำดับข้อมูลโดยทั่วไปเป็นการเรียงลำดับข้อมูลแบบอนุกรม ซึ่งเป็นกระบวนการจัดเรียงลำดับข้อมูลในตาราง หรือแฟ้มข้อมูลให้เรียงตามลำดับข้อมูลจากน้อยไปหามาก (Ascending Sequence) หรือเรียงลำดับข้อมูลจากมากไปหาน้อย (Descending Sequence) ซึ่งขั้นตอนวิธี (Algorithm) สำหรับการเรียงลำดับข้อมูลแบบอนุกรมแบบต่างๆ นี้ มีผู้คิดขึ้นมากมายด้วยความซับซ้อนด้านเวลา (Time Complexity) เท่ากับ $O(N \log_2 N)$ เมื่อ N เป็นขนาดของข้อมูล อาทิ การเรียงลำดับแบบเร็ว (Quick Sort) การเรียงลำดับแบบรวมกัน (Merge Sort) การเรียงลำดับ

แบบฮีป (Heap Sort) โดยที่แต่ละวิธีต่างก็มีข้อดีข้อเสีย ขึ้นอยู่กับลักษณะและปริมาณข้อมูลที่จะเรียงลำดับรวมถึงประสิทธิภาพของเครื่องคอมพิวเตอร์ที่ใช้ในการเรียงลำดับข้อมูลอีกด้วย

2.2.2 การเรียงลำดับแบบขนาน (Parallel Sort)

การเรียงลำดับแบบขนานถูกเสนอขึ้นสำหรับการเรียงลำดับข้อมูลขนาดใหญ่ เพราะถ้าข้อมูลมีขนาดใหญ่มากๆ การเรียงลำดับข้อมูลแบบอนุกรม (Sequential Sort) ที่ใช้เพียงหนึ่งหน่วยประมวลผล ดังที่ได้กล่าวไว้ในข้อ 2.2.1 จะใช้เวลาในการเรียงลำดับข้อมูลเป็นเวลานาน จากปัญหาข้างต้นจึงนำการเรียงลำดับแบบขนานโดยใช้หลายๆ หน่วยประมวลผลและการจัดเก็บข้อมูลจำนวนที่เหมาะสมในแต่ละหน่วยประมวลผล จึงทำให้ความเร็วในการเรียงลำดับข้อมูลสูงขึ้น

การเรียงลำดับแบบขนาน (Parallel Sort) [3] เป็นปัญหาค่อนข้างใหม่ที่ต้องศึกษา เพราะจะต้องทำความเข้าใจในเรื่องของการติดต่อสื่อสาร (Communication) ระหว่างหน่วยประมวลผล ที่ซับซ้อน และการประมวลผลแบบขนาน (Parallel Computing) ที่ทำให้เร็วขึ้น โดยขั้นตอนวิธี (Algorithm) การเรียงลำดับแบบขนานที่นิยมใช้ในปัจจุบัน เช่น การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (Bitonic Sort) [2] [19] [26] การเรียงลำดับแบบขนานด้วยวิธีคู่คี่ (odd-Even Sort) [19] [26] ด้วยความซับซ้อนด้านเวลา (Time Complexity) เท่ากับ $O(\log_2 N)^2$ เมื่อจำนวนหน่วยประมวลผลเท่ากับขนาดของข้อมูล (P=N) เป็นต้น

2.3 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (Parallel Bitonic Sorting)

การเรียงลำดับแบบไบโทนิค (Bitonic Sort) [2] [14] [19] [26] นี้ได้ถูกคิดค้นขึ้นเมื่อ ค.ศ. 1968 โดย Kenneth E. Batcher ซึ่งเป็นการเรียงลำดับแบบขนานที่ได้รับความนิยมอย่างมาก เนื่องจากความซับซ้อนด้านเวลา (Time Complexity) เพียง $O(\log_2 N)^2$ นับว่าเป็นเวลาที่ดีที่สุด โดยเราสามารถให้คำนิยามของการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคได้ดังนี้

นิยามที่ 1 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคเป็นการเรียงลำดับชุดของข้อมูลที่เรียงจากน้อยไปหามาก ($s_0 \le s_1 \le \ldots \le s_{N-1}$) หรือเรียงจากมากไปหาน้อย ($s_0 \ge s_1 \ge \ldots \ge s_{N-1}$) โดยก่อนที่มีการเรียงข้อมูลนั้น ข้อมูลเดิมต้องอยู่ในรูปแบบที่เรียกว่า "Bitonic Sequence" $a_0, a_1, a_2, \ldots, a_{N-1}$ ซึ่งประกอบด้วยข้อมูล 2 ชุดข้อมูลที่เรียงต่อกัน เมื่อ $a_0, \ldots, a_i$ เป็นชุดของข้อมูลย่อยที่เรียงจากน้อยไปหามาก และ $a_{i+1}, \ldots, a_{N-1}$ เป็นชุดของข้อมูลย่อยที่เรียงจากมากไปหาน้อย

ตัวอย่าง ชุดข้อมูล 5, 6, 10, 15, 30, 28, 20, 12, 3, 1 เป็นข้อมูลแบบ "Bitonic Sequence" เนื่องจากประกอบด้วยข้อมูล 2 ชุด ที่มีคุณสมบัติดังนี้ คือ

5, 6, 10, 15, 30 เป็นชุดของข้อมูลย่อยที่เรียงจากน้อยไปหามากและ

28, 20, 12, 3, 1 เป็นอีกชุดของข้อมูลย่อยที่เรียงจากมากไปหาน้อย

นิยามที่ 2 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (กรณีที่ข้อมูลอยู่ในรูปแบบของ "Bitonic Sequence") เมื่อกำหนดให้จำนวนข้อมูล (N) เท่ากับจำนวนหน่วยประมวลผล (P) หรือ P=N สามารถคำนวณหาจำนวนรอบในการทำงานได้ ดังสมการที่ 2.1 [14]

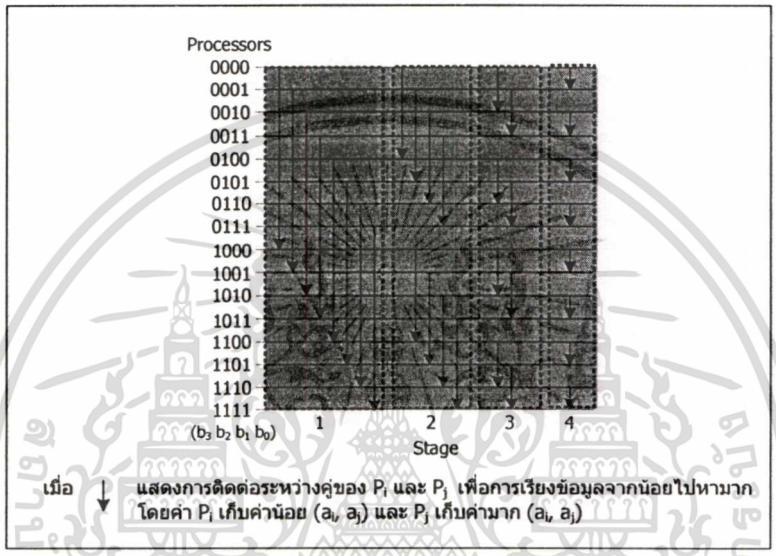
$$\text{จำนวนรอบ} = \log_2 P \qquad (2.1)$$

ในแต่ละรอบ (Stage) ทุกคู่ของหน่วยประมวลผล (P_i, P_j) จะประมวลผลไปพร้อมๆ กัน การคำนวณค่า P_i และ P_j ที่ติดต่อสื่อสารเพื่อแลกเปลี่ยนข้อมูลระหว่างกันพิจารณาจากหลักการทางด้านการออกแบบสถาปัตยกรรม (Architecture Design) ก่อน ซึ่งจะพิจารณาค่า i และ j เป็นเลขฐานสอง (Binary Number) แล้วจึงนำมาประยุกต์สำหรับการออกแบบขั้นตอนวิธี (Algorithm Design) ซึ่งพิจารณาค่า i และ j เป็นเลขฐานสิบ (Decimal Number) ในระบบเลขฐานสองค่า i และ j แสดงด้วยเลขฐานสิบจำนวน m บิต (Bits) โดย $m = \log_2 P$ คือ $b_{m-1}\, b_{m-2} \ldots b_2\, b_1\, b_0$ เมื่อ b มีค่าเท่ากับศูนย์หรือหนึ่ง (0/1)

- ในรอบที่ 1 คู่ (P_i, P_j) มีค่าแตกต่างกันในบิตที่ b_{m-1} โดยที่ P_i มีค่า $b_{m-1} = 0$ หรือ $i_{10} = 0\, b_{m-2}\, b_{m-3} \ldots b_2\, b_1\, b_0$ และค่า P_j มีค่า $b_{m-1} = 1$ หรือ $j_{10} = 1\, b_{m-2}\, b_{m-3} \ldots b_2\, b_1\, b_0$
- ในรอบที่ 2 คู่ (P_i, P_j) มีค่าแตกต่างกันในบิตที่ b_{m-2} โดยที่ P_i มีค่า $b_{m-2} = 0$ หรือ $i_{10} = b_{m-1}\, 0\, b_{m-3} \ldots b_2\, b_1\, b_0$ และค่า P_j มีค่า $b_{m-2} = 1$ หรือ $j_{10} = b_{m-1}\, 1\, b_{m-3} \ldots b_2\, b_1\, b_0$
- ในรอบที่ s ใดๆ คู่ (P_i, P_j) มีค่าแตกต่างกันในบิตที่ b_{m-s} โดยที่ P_i มีค่า $b_{m-s} = 0$ หรือ $i_{10} = b_{m-1} \ldots b_{m-s-1}\, 0\, b_{m-s+1} \ldots b_0$ และค่า P_j มีค่า $b_{m-s} = 1$ หรือ $j_{10} = b_{m-1} \ldots b_{m-s-1}\, 1\, b_{m-s+1} \ldots b_0$
- ในรอบที่ m (รอบสุดท้าย) คู่ (P_i, P_j) มีค่าแตกต่างกันในบิตที่ b_0 โดยที่ P_i มีค่า $b_0 = 0$ หรือ $i_{10} = b_{m-1} \ldots b_2\, b_1\, 0$ และค่า P_j มีค่า $b_0 = 1$ หรือ $j_{10} = b_{m-1} \ldots b_2\, b_1\, 1$

ในระบบเลขฐานสิบค่า i และ j คำนวณจากเงื่อนไขดังนี้คือ ในรอบที่ s ใดๆ (s = 1, 2, ... , m) สำหรับทุกค่าของ P_i เมื่อ i = 0, 1, 2, ..., P-1 (จำนวนหน่วยประมวลผล) จะได้ค่า j = i+d สำหรับ P_j ถ้า (i mod 2d) < d โดยที่ $d = 2^{m-s}$

ตัวอย่างเช่น รูปที่ 2.6 แสดงการเรียงลำดับแบบขนานด้วยวิธีไบโทนิค โดยใช้หน่วยประมวลผล 16 หน่วย (P=16) ดังนั้นจำนวนรอบเท่ากับ $\log_2 16 = 4$ รอบ ในแต่ละรอบทุกคู่ของหน่วยประมวลผล (P_i, P_j) จะประมวลผลไปพร้อมๆ กัน ส่วนการคำนวณค่า (P_i, P_j) ในแต่ละรอบแสดงไว้ในตารางที่ 2.1 เช่น ในรอบที่ 1 คู่(P_0, P_8), คู่(P_1, P_9), ..., คู่(P_6, P_{14}), คู่(P_7, P_{15}) จะทำการติดต่อกันแบบขนานกัน ในรอบที่ 2 คู่(P_0, P_4), คู่(P_1, P_5), ..., คู่(P_{10}, P_{14}), คู่(P_{11}, P_{15}) ในรอบที่ 3 คู่(P_0, P_2), คู่(P_1, P_3), ..., คู่(P_{12}, P_{14}), คู่(P_{13}, P_{15}) และรอบที่ 4 คู่(P_0, P_1), คู่(P_2, P_3), ..., คู่(P_{12}, P_{13}), คู่(P_{14}, P_{15}) เป็นต้น



รูปที่ 2.6 การเรียงลำดับแบบไบโทนิคใช้ 16 หน่วยประมวลผล (กรณีข้อมูลเป็น Bitonic Sequence)

ตารางที่ 2.1 แสดงการหาค่า (P_i, P_j) จำนวน $m = \log_2 P$ รอบ กรณีที่จำนวนหน่วยประมวลผลเท่ากับขนาดข้อมูล (N=P=16)

รอบ Stage (s)	การหาค่า i และ j เมื่อ x เท่ากับศูนย์หรือหนึ่ง (0/1)			(P_i, P_j) เมื่อ i = 0, 1, ..., 15 และ j = i+d ถ้า (i mod 2d) < d	
	$d = 2^{m-s}$	i	j	ฐานสอง	ฐานสิบ
1	$2^3 = 8$	0xxx	1xxx	(0000, 1000) (0001, 1001) ... (0111, 1111)	(P_0, P_8) (P_1, P_9) ... (P_7, P_{15})
2	$2^2 = 4$	x0xx	x1xx	(0000, 0100) (0001, 0101) ... (1011, 1111)	(P_0, P_4) (P_1, P_5) ... (P_{11}, P_{15})
3	$2^1 = 2$	xx0x	xx1x	(0000, 0010) (0001, 0011) ... (1101, 1111)	(P_0, P_2) (P_1, P_3) ... (P_{13}, P_{15})
4	$2^0 = 1$	xxx0	xxx1	(0000, 0001) (0010, 0011) ... (1110, 1111)	(P_0, P_1) (P_2, P_3) ... (P_{14}, P_{15})

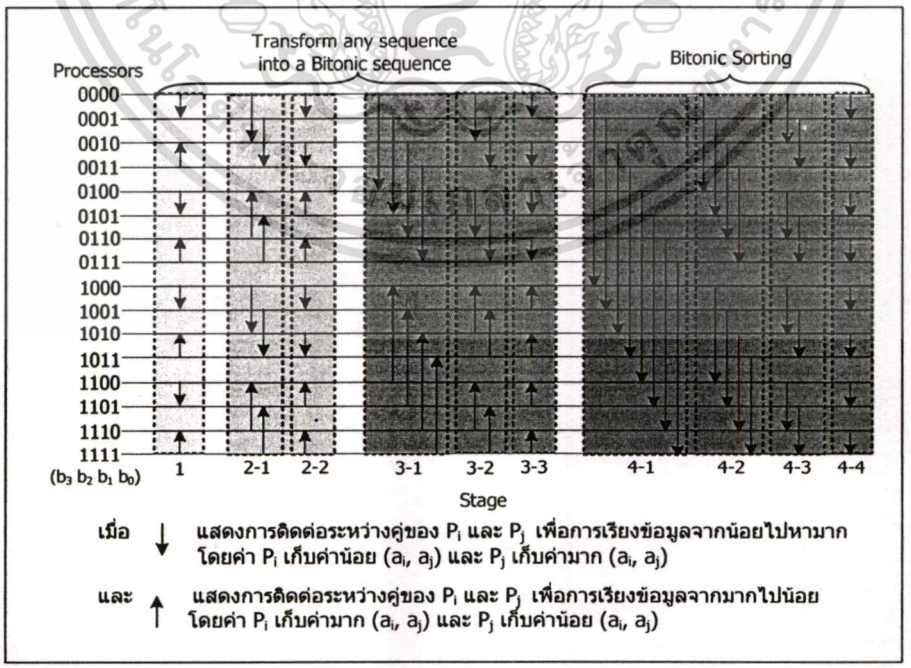
นิยามที่ 3 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (กรณีที่ข้อมูลนำเข้าเป็นแบบทั่วไป) ซึ่งประกอบด้วย 2 ขั้นตอนคือ ขั้นแรกจะต้องทำการแปลง (Transformation) ข้อมูลทั่วไป (Any Sequence) ให้อยู่ในรูปของ "Bitonic Sequence" เสียก่อน และขั้นตอนที่สองเป็นการเรียงข้อมูลแบบขนานด้วยวิธีไบโทนิค (Bitonic Sorting) โดยกรณีทั่วไปนี้ การคำนวณหาจำนวนรอบในการเรียงลำดับข้อมูลแบบขนานด้วยวิธีไบโทนิค สามารถคำนวณได้ตามสมการที่ 2.2 [14]

$$\text{จำนวนรอบ} = \frac{(1 + \log_2 P)(\log_2 P)}{2} \qquad (2.2)$$

เนื่องจากจำนวนรอบในขั้นตอนที่สอง (ตามนิยามที่ 2) เท่ากับ $\log_2 P$ ดังนั้น จำนวนรอบในขั้นตอนแรกเท่ากับ $(1+\log_2 P)(\log_2 P)/2 - (\log_2 P) = (1+\log_2 P)/2-1)(\log_2 P)$

ในแต่ละรอบ (Stage) ทุกคู่ของหน่วยประมวลผล (P_i, P_j) จะประมวลผลไปพร้อมๆ กัน การคำนวณค่า P_i และ P_j ที่ติดต่อสื่อสารแลกเปลี่ยนข้อมูลระหว่างกัน จะพิจารณาได้ในทำนองเดียวกับที่กล่าวมาในนิยามที่ 2

ตัวอย่างเช่น รูปที่ 2.7 แสดงการเรียงลำดับข้อมูลแบบขนานด้วยวิธีไบโทนิค โดยใช้หน่วยประมวลผล 16 หน่วย (P=16) ดังนั้นจำนวนรอบเท่ากับ (1+4)(4)/2 = 10 รอบ ในแต่ละรอบทุกคู่ของหน่วยประมวลผล (P_i, P_j) จะประมวลผลไปพร้อมๆ กัน โดยการคำนวณค่า (P_i, P_j) ที่ติดต่อสื่อสารเพื่อแลกเปลี่ยนข้อมูลระหว่างกัน ซึ่งแสดงไว้ในตารางที่ 2.2



รูปที่ 2.7 การเรียงลำดับแบบไบโทนิคใช้ 16 หน่วยประมวลผล (กรณีข้อมูลอยู่ในรูปทั่วไป)

ตารางที่ 2.2 แสดงการหาค่า (P_i, P_j) จำนวน $(1+\log_2 P)(\log_2 P)/2$ รอบ กรณีที่จำนวนหน่วยประมวลผลเท่ากับขนาดข้อมูล (N=P=16)

รอบนอก Stage	รอบย่อย (s)	การหาค่า i และ j เมื่อ x เท่ากับศูนย์หรือหนึ่ง (0/1)			(P_i, P_j) เมื่อ i = 0, 1, ..., 15 และ j = i+d ถ้า (i mod 2d) < d	
		$d = 2^{m-s}$	i	j	ฐานสอง	ฐานสิบ
1	1	$2^0 = 1$	xxx0	xxx1	(0000, 0001), ..., (1110, 1111)	(P_0, P_1), ..., (P_{14}, P_{15})
2	1	$2^1 = 2$	xx0x	xx1x	(0000, 0010), ..., (1101, 1111)	(P_0, P_2), ..., (P_{13}, P_{15})
	2	$2^0 = 1$	xxx0	xxx1	(0000, 0001), ..., (1110, 1111)	(P_0, P_1), ..., (P_{14}, P_{15})
3	1	$2^2 = 4$	x0xx	x1xx	(0000, 0100), ..., (1011, 1111)	(P_0, P_4), ..., (P_{11}, P_{15})
	2	$2^1 = 2$	xx0x	xx1x	(0000, 0010), ..., (1101, 1111)	(P_0, P_2), ..., (P_{13}, P_{15})
	3	$2^0 = 1$	xxx0	xxx1	(0000, 0001), ..., (1110, 1111)	(P_0, P_1), ..., (P_{14}, P_{15})
4	1	$2^3 = 8$	0xxx	1xxx	(0000, 1000), ..., (0111, 1111)	(P_0, P_8), ..., (P_7, P_{15})
	2	$2^2 = 4$	x0xx	x1xx	(0000, 0100), ..., (1011,1111)	(P_0, P_4), ..., (P_{11}, P_{15})
	3	$2^1 = 2$	xx0x	xx1x	(0000, 0010), ..., (1101,1111)	(P_0, P_2), ..., (P_{13}, P_{15})
	4	$2^0 = 1$	xxx0	xxx1	(0000, 0001), ..., (1110,1111)	(P_0, P_1), ..., (P_{14}, P_{15})

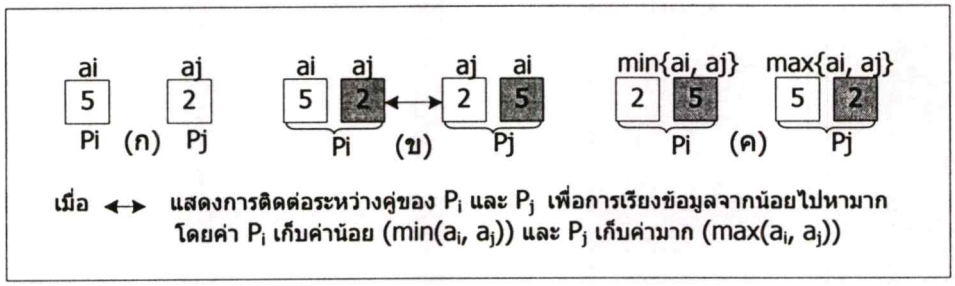
หมายเหตุ กรณีนี้ m เป็นจำนวนรอบนอก โดย $m = 1, 2, 3, ..., \log_2 P$ และเมื่อ m=4 ในตารางที่ 2.2 จะเหมือนกับกรณีที่แสดงไว้ในตารางที่ 2.1

การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค โดยปกติจะเป็นกรณีที่สมมติว่ามีจำนวนหน่วยประมวลผล (P) มากเพียงพอสำหรับข้อมูลขนาด N (P=N) แต่ในทางปฏิบัติหรือกรณีทั่วๆ ไป จำนวนหน่วยประมวลผลมักจะน้อยกว่าขนาดของข้อมูล (P<N) รายละเอียดของแต่ละกรณีจะแสดงในหัวข้อต่อไปนี้

2.3.1 กรณีที่จำนวนหน่วยประมวลผลเท่ากับจำนวนข้อมูล (P = N)

ในกรณีที่จำนวนหน่วยประมวลผล (P) เท่ากับจำนวนข้อมูล (N) ดังนั้นเริ่มแรกหน่วยประมวลผลแต่ละหน่วย (P_i) จะเก็บค่าของข้อมูลไว้เพียงหนึ่งค่า (a_i) ขั้นต่อไปจะทำการเปรียบเทียบค่าในหน่วยประมวลผลที่ P_i กับ P_j เป็นคู่ๆ ($i < j$) โดยค่าที่ได้หลังจากเปรียบเทียบนั้นจะกำหนดให้ค่าน้อยถูกเก็บที่หน่วยประมวลผล P_i และค่ามากถูกเก็บไว้ที่หน่วยประมวลผล P_j ซึ่งเรียกตัวดำเนินการ (Operation) แบบนี้ว่า "Compare-Exchange" [14] หลักการของ Compare-Exchange สามารถอธิบายได้ง่ายๆ ดังรูปที่ 2.8 โดยรูปที่ 2.8 (ก) แสดงการโหลดข้อมูลเข้าสู่หน่วยประมวลผลของ P_i และ P_j โดย P_i เก็บค่า 5 และ P_j เก็บค่า 2 รูปที่ 2.8 (ข) แสดงการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล P_i และ P_j โดยที่ข้อมูลในกรอบสีเทาคือข้อมูลที่ถูกส่งมาจากหน่วยประมวลผล

อื่น รูปที่ 2.8(ค) เปรียบเทียบค่าที่รับมากับค่าที่มีอยู่โดย P_i จะเก็บค่าน้อย ($\min(a_i, a_j)$) ส่วน P_j จะเก็บค่ามาก ($\max(a_i, a_j)$)



รูปที่ 2.8 ตัวดำเนินการแบบ "Compare-Exchange"

สมมติให้หน่วยประมวลผลที่ P_i และ P_j อยู่ติดกันและการติดต่อกันสามารถติดต่อถึงกันได้โดยตรง ทำให้เวลาที่ใช้ในการติดต่อสื่อสารในขั้นตอนของ Compare-Exchange ตามสมการที่ 2.3

$$T_s + T_w \tag{2.3}$$

เมื่อ T_s คือ เวลาเริ่มต้นในการติดต่อกับหน่วยประมวลผลอื่น

T_w คือ เวลาที่ใช้ในการแลกเปลี่ยนข้อมูล

การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคสำหรับกรณีที่ 1 เป็นข้อมูล "Bitonic Sequence" และกรณีที่ 2 เป็นข้อมูลทั่วไป นั้นจะมีขั้นตอนวิธี [26] โดยทั่วไปดังนี้ เมื่อจำนวนหน่วยประมวลผล (P) เท่ากับจำนวนข้อมูล (N) หรือ P = N

```
Global      d: Distance between elements being compare
Local       a: One of the elements to be sorted
            t: Element retrieved from adjacent processor
Begin
        for J = m-1 downto 0 do
           d = 2^J
           // For All Processor
           for   all Processor k where 0 <= k <= 2m-1 pardo
                 if k mod 2d < d then
                       t = [k + d]a
                       k+d]a = max(t, a) //sort low to high
                       a =  min(t, a)
                 end if
           end for  all
        end for J
end
```

ขั้นตอนวิธีที่ 1 ขั้นตอนวิธีการเรียงลำดับด้วยวิธีไบโทนิค (P=N) สำหรับข้อมูล "Bitonic Sequence"

ขั้นตอนวิธีการเรียงลำดับด้วยวิธีไบโทนิค เมื่อข้อมูลนำเข้าอยู่ในรูปทั่วไปๆ แสดงได้นี้

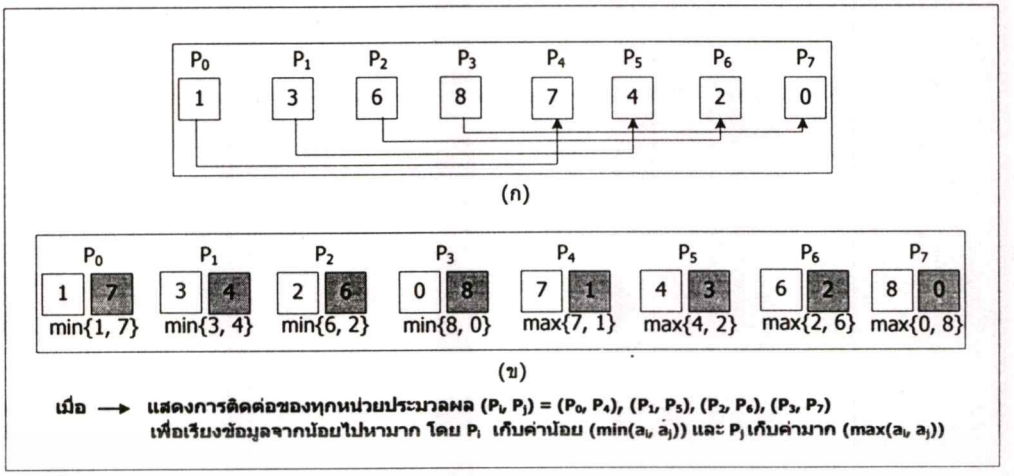
```
Global      d: Distance between elements being compare
Local       a: One of the elements to be sorted
            t: Element retrieved from adjacent processor
Begin
      for l = 0 up to m-1 do   // m = log2 N
            for J = l downto 0 do
               d = 2^J
               // For All Processor
               for   all Processor k where 0 <= k <= 2m-1 pardo
                     if k mod 2d < d then
                        t =[k + d]a
                        if  k mod  2l+2 < 2l+1  then
                            k+d]a =  max(t, a) //sort low to high
                                 a =  min(t, a)
                        else
                            [k+d]a =  min(t, a) //sort high to low
                                 a =  max(t, a)
                        end if
                     end if
               end for all
            end for J
      end for l
end
```

ขั้นตอนวิธีที่ 2 ขั้นตอนวิธีการเรียงลำดับด้วยวิธีไบโทนิค (P=N) สำหรับข้อมูลทั่วไป

<u>ตัวอย่างที่ 2.1</u> กรณีที่ข้อมูลนำเข้าอยู่ในรูปแบบของ "Bitonic Sequence"

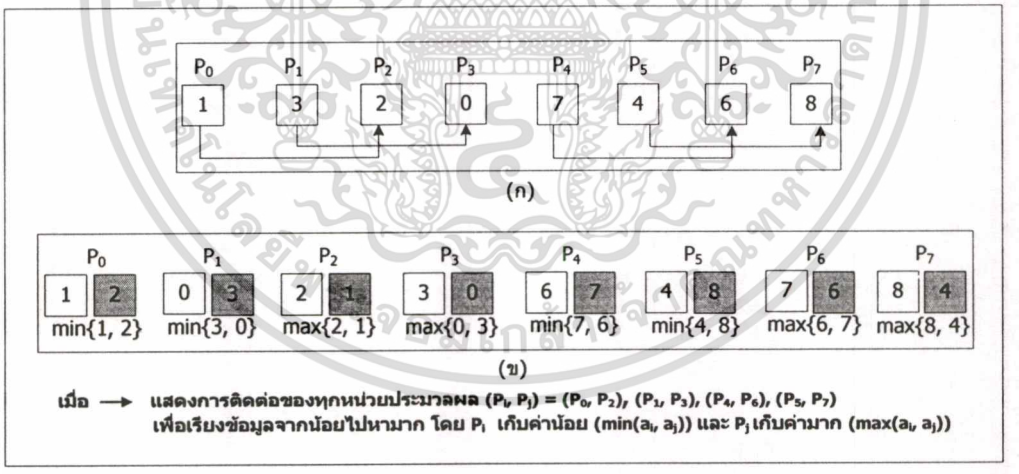
สมมติให้ข้อมูลนำเข้า คือ 1, 3, 6, 8, 7, 4, 2, 0 (N=8) ซึ่งอยู่ในรูปของ "Bitonic Sequence" เรียบร้อยแล้ว ขั้นตอนการเรียงลำดับข้อมูลแบบขนานด้วยวิธีไบโทนิคในกรณีที่จำนวนหน่วยประมวลผลเท่ากับจำนวนข้อมูล (P=N) สามารถคำนวณหาจำนวนรอบที่ใช้ในการเรียงลำดับข้อมูลได้ตามนิยามที่ 2 ดังนี้ จำนวนรอบ $m = (\log_2 P) = (\log_2 8) = 3$ รอบ ดังรายละเอียดที่จะได้กล่าวถึงต่อไปนี้คือ

รอบที่ 1 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคในรอบที่ 1 (s=1) ดังรูปที่ 2.9 (ก) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล P=8 ($P_0, P_1, P_2, \ldots, P_7$) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน เมื่อ $d = 2^{m-s} = 4$ สำหรับทุกค่า i = 0, 1, 2, 3, ..., 7 และ j = i+d ถ้า (i mod 2d) < d เช่น คู่ (P_0, P_4), คู่ (P_1, P_5), ..., คู่ (P_3, P_7) และรูปที่ 2.9 (ข) เปรียบเทียบข้อมูลระหว่างหน่วยประมวลผล (P_i, P_j) ซึ่งค่า P_i เก็บค่าน้อย ($\min(a_i, a_j)$) และ P_j เก็บค่ามาก ($\max(a_i, a_j)$)



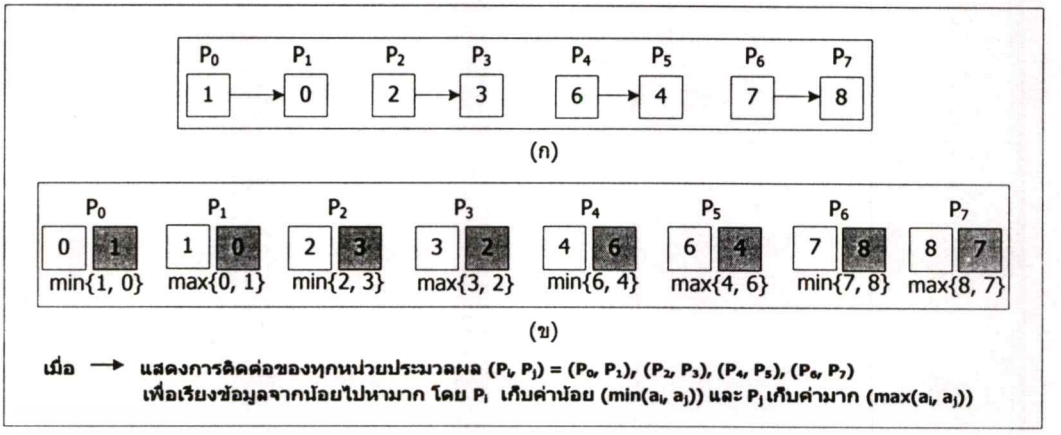
รูปที่ 2.9 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (ข้อมูล "Bitonic Sequence") รอบที่ 1

รอบที่ 2 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคในรอบที่ 2 (s=2) ดังรูปที่ 2.10 (ก) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล P=8 (P_0,P_1,P_2,..., P_7) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน เมื่อ $d = 2^{m-s} = 2$ สำหรับทุกค่า i = 0, 1, 2, 3, ..., 7 และ j = i+d ถ้า (i mod 2d) < d เช่น คู่ (P_0, P_2), คู่ (P_1, P_3), ..., คู่ (P_5, P_7) และรูปที่ 2.10 (ข) เปรียบเทียบข้อมูลระหว่างหน่วยประมวลผล (P_i, P_j) ซึ่ง P_i เก็บค่าน้อย ($\min(a_i, a_j)$) และ P_j เก็บค่ามาก ($\max(a_i, a_j)$)



รูปที่ 2.10 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (ข้อมูล "Bitonic Sequence") รอบที่ 2

รอบที่ 3 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคในรอบที่ 3 (s=3) ดังรูปที่ 2.11 (ก) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล P=8 (P_0,P_1,P_2,..., P_7) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน เมื่อ $d = 2^{m-s} = 1$ สำหรับทุกค่า i = 0, 1, 2, 3, ..., 7 และ j = i+d ถ้า (i mod 2d) < d เช่น คู่ (P_0, P_1), คู่ (P_2, P_3), ..., คู่ (P_6, P_7) และรูปที่ 2.11 (ข) เปรียบเทียบข้อมูลระหว่างหน่วยประมวลผล (P_i, P_j) ซึ่ง P_i เก็บค่าน้อย ($\min(a_i, a_j)$) และ P_j เก็บค่ามาก ($\max(a_i, a_j)$)



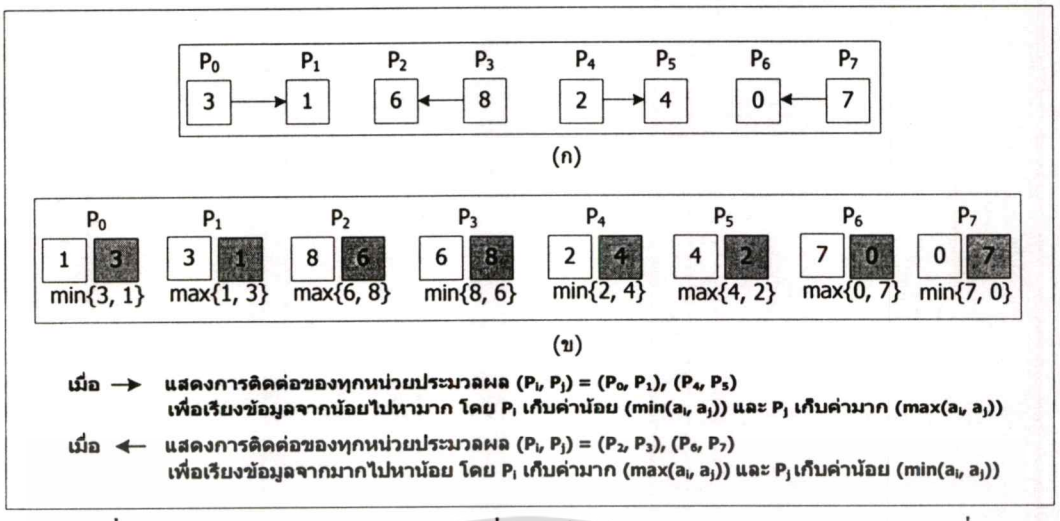
รูปที่ 2.11 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (ข้อมูล "Bitonic Sequence") รอบที่ 3

ตัวอย่างที่ 2.2 กรณีที่ข้อมูลนำเข้าอยู่ในรูปทั่วไป (ไม่อยู่ในรูปของ "Bitonic Sequence")

สมมติให้ข้อมูลนำเข้า คือ 3, 1, 6, 8, 2, 4, 0, 7 (N=8) ยังไม่ได้ถูกจัดให้อยู่ในรูปของ "Bitonic Sequence" ดังนั้นการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคกรณีที่จำนวนหน่วยประมวลผลเท่ากับขนาดของข้อมูล (P=N) จึงจำเป็นต้องใช้ 2 ขั้นตอนในการเรียงลำดับข้อมูล ตามนิยามที่ 3 และสามารถคำนวณหาจำนวนรอบในการเรียงลำดับข้อมูลได้ ดังนี้ $(1+\log_2 P)(\log_2 P)/2 = (1+\log_2 8)(\log_2 8)/2 = 6$ รอบ โดยมีรายละเอียดที่จะได้กล่าวดังต่อไปนี้คือ

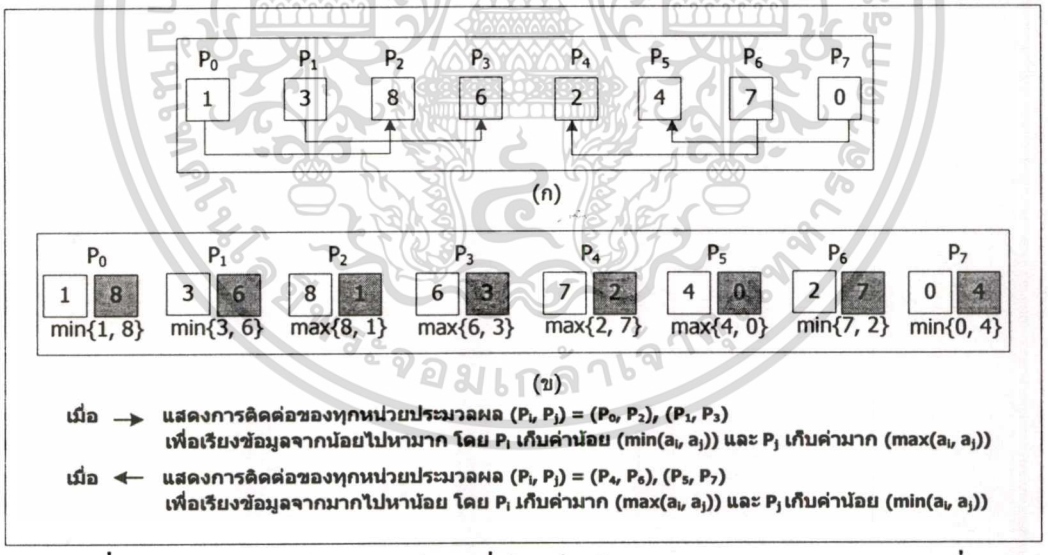
ขั้นแรก การแปลงข้อมูลทั่วไปให้อยู่ในรูปของ "Bitonic Sequence" ซึ่งสามารถคำนวณหาจำนวนรอบการทำงานได้ โดยนำจำนวนรอบในขั้นที่สอง ($\log_2 P$) มาหักออกจากจำนวนทั้งหมดซึ่งเท่ากับ $(1+\log_2 P)(\log_2 P)/2 - (\log_2 P) = 6 - 3 = 3$ รอบ

รอบที่ 1 การแปลงข้อมูลจากข้อมูลทั่วไปให้อยู่ในรูปแบบของ "Bitonic Sequence" ดังรูปที่ 2.12 (ก) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล P=8 ($P_0, P_1, P_2, \ldots, P_7$) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน เมื่อ m=1, s=1, $d = 2^{m-s} = 1$ สำหรับทุกค่า i = 0, 1, 2, 3, ..., 7 และ j = i+d ถ้า (i mod 2d) < d เช่น คู่ (P_0, P_1), คู่ (P_2, P_3), …, คู่ (P_6, P_7) และรูปที่ 2.12 (ข) เปรียบเทียบข้อมูลระหว่างหน่วยประมวลผล (P_i, P_j)



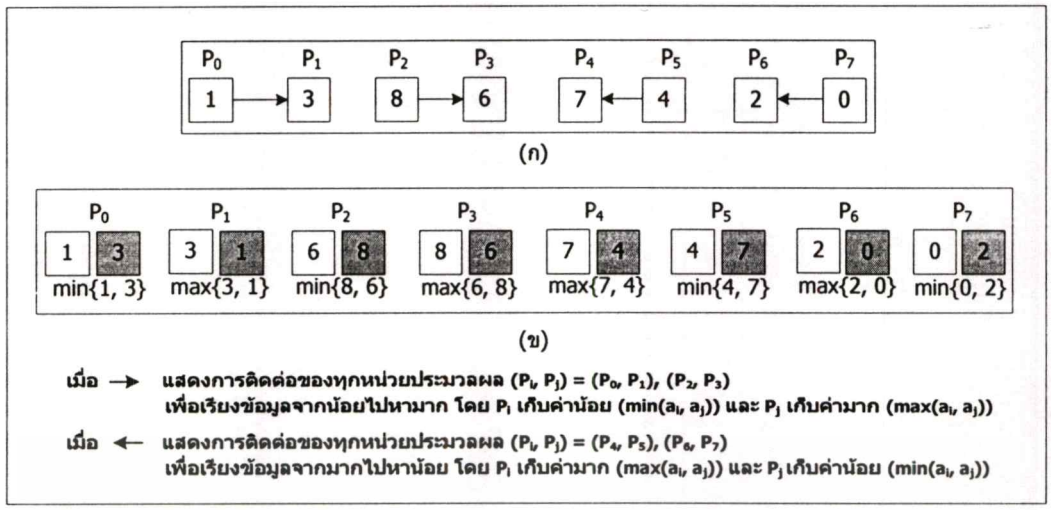
รูปที่ 2.12 การแปลงข้อมูลจากข้อมูลทั่วไปเป็นข้อมูล "Bitonic Sequence" รอบที่ 1

รอบที่ 2 การแปลงข้อมูลจากข้อมูลทั่วไปให้อยู่ในรูปแบบของ "Bitonic Sequence" ดังรูปที่ 2.13 (ก) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล P=8 (P_0,P_1,P_2,..., P_7) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน เมื่อ m=2, s=1, $d = 2^{m-s} = 2$ สำหรับทุกค่า i = 0, 1, 2, 3, ..., 7 และ j = i+d ถ้า (i mod 2d) < d เช่น คู่ (P_0, P_2), คู่ (P_1, P_3), ..., คู่ (P_5, P_7) และรูปที่ 2.13 (ข) เปรียบเทียบข้อมูลระหว่างหน่วยประมวลผล (P_i, P_j)



รูปที่ 2.13 การแปลงข้อมูลจากข้อมูลทั่วไปเป็นข้อมูล "Bitonic Sequence" รอบที่ 2

รอบที่ 3 การแปลงข้อมูลจากข้อมูลทั่วไปให้อยู่ในรูปแบบของ "Bitonic Sequence" ดังรูปที่ 2.14 (ก) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล P=8 (P_0,P_1,P_2,..., P_7) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน เมื่อ m=2, s=2, $d = 2^{m-s} = 1$ สำหรับทุกค่า i = 0, 1, 2, 3, ..., 7 และ j = i+d ถ้า (i mod 2d) < d เช่น คู่ (P_0, P_1), คู่ (P_2, P_3), ..., คู่ (P_6, P_7) และรูปที่ 2.14 (ข) เปรียบเทียบข้อมูลระหว่างหน่วยประมวลผล (P_i, P_j)



รูปที่ 2.14 การแปลงข้อมูลจากข้อมูลทั่วไปเป็นข้อมูล "Bitonic Sequence" รอบที่ 3

ขั้นที่สอง การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค สำหรับข้อมูลจากขั้นแรก (1, 3, 6, 8, 7, 4, 2,0) ซึ่งอยู่ในรูปของ "Bitonic Sequence" สามารถคำนวณหาจำนวนรอบการทำงานได้จากนิยามที่ 2 คือ $(\log_2 P) = (\log_2 8) = 3$ รอบ เช่นเดียวกับที่ได้แสดงไว้แล้วในตัวอย่างที่ 2.1

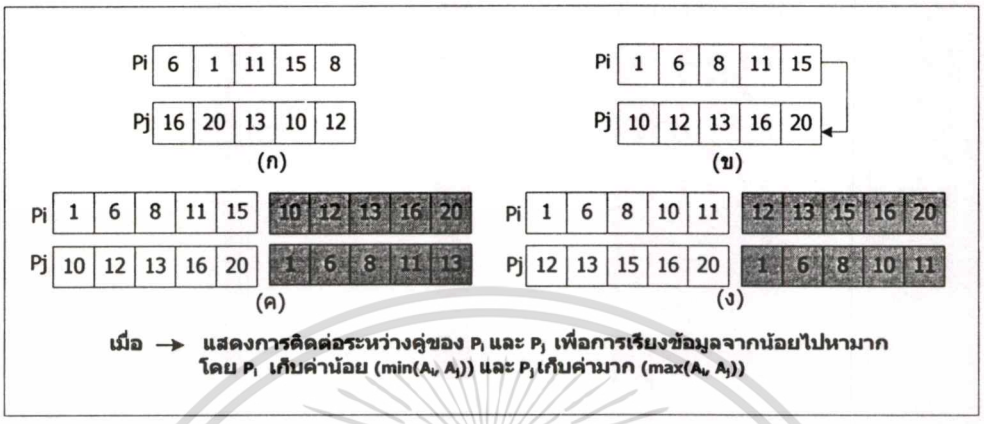
2.3.2 กรณีที่จำนวนหน่วยประมวลผลน้อยกว่าจำนวนข้อมูล (P < N)

ในกรณีที่จำนวนหน่วยประมวลผลน้อยกว่าจำนวนข้อมูล (P < N) กำหนดให้ P เป็นจำนวนของหน่วยประมวลผลโดยเรียงตั้งแต่ $P_0, P_1, P_2, \ldots, P_{P-1}$ และให้ N เป็นจำนวนข้อมูล $(a_0, a_1, \ldots, a_{N-1})$ ซึ่งข้อมูลทั้งหมดจะถูกแบ่งออกเป็นกลุ่มของข้อมูลขนาด n ซึ่งเท่ากับ N/P ค่า โดยกำหนดให้ $A_0, A_1, A_2, \ldots, A_{P-1}$ เป็นชุดของข้อมูลจำนวน P กลุ่มๆ ละ n จำนวน เมื่อ $A_i = a_{in}, a_{in+1}, a_{in+2}, \ldots, a_{in+(n-1)}$ และ i = 0, 1, 2, ..., P-1 (เช่น $A_0 = a_0, a_1, \ldots, a_{n-1}$, $A_1 = a_n, a_{n+1}, \ldots, a_{2n-1}, \ldots, A_{P-1} = a_{(P-1)n}, a_{(P-1)n+1}, \ldots, a_{N-1}$) ที่ส่งไปให้แต่ละหน่วยประมวลผล (P_i)

ในการเปรียบเทียบการเรียงลำดับแบบไบโทนิค เมื่อจำนวนหน่วยประมวลผลน้อยกว่าจำนวนข้อมูล ในกรณีข้อมูลทั่วๆ ไป ต้องพยายามจัดข้อมูลเป็น 2 ชุด ตามเงื่อนไขดังนี้คือ $A_i <= A_j$ (ซึ่งทุกๆ ค่าที่อยู่ใน A_i จะน้อยกว่าหรือเท่ากับค่าที่อยู่ใน A_j) หรือ $A_i >= A_j$ (ซึ่งทุกๆ ค่าที่อยู่ใน A_i จะมากกว่าหรือเท่ากับค่าที่อยู่ใน A_j) ตามรูปแบบของไบโทนิค ในการจัดการเพื่อให้ได้ข้อมูลในรูปแบบดังกล่าวจะใช้ตัวดำเนินการในการเปรียบเทียบค่า ซึ่งเรียกว่า "Compare-Split" [14]

ตัวอย่างเช่น รูปที่ 2.15 (ก) แสดงการโหลดข้อมูลขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_i เก็บค่า (6, 1, 11, 15, 8) และ P_j เก็บค่า (16, 20, 13, 10, 12) ไว้ในหน่วยความจำส่วนตัว (Local Memory) และทำการเรียงลำดับข้อมูลขนาด N/P ด้วยวิธีแบบเร็ว (Quick Sort) ในทุกหน่วยประมวลผลพร้อมๆ กัน รูปที่ 2.15 (ข) แสดงข้อมูลที่เรียงแล้วและการติดต่อสื่อสารระหว่าง P_i กับ

P_j รูปที่ 2.15 (ค) แสดงการแลกเปลี่ยนชุดข้อมูลระหว่าง P_i กับ P_j รวมถึงการรวมชุดข้อมูลทั้ง 2 ชุดเข้าด้วยกัน (Merge Sort) รูปที่ 2.15 (ง) เปรียบเทียบค่าที่รับมากับค่าที่มีอยู่โดย P_i เก็บค่าน้อย ($\min(A_i, A_j)$) และ P_j เก็บค่ามาก ($\max(A_i, A_j)$)



รูปที่ 2.15 ตัวดำเนินการแบบ "Compare-Split"

สมมติให้ P_i กับ P_j เป็นหน่วยประมวลผลที่เชื่อมต่อถึงกันโดยตรง ดังนั้นเวลาที่ใช้ในการเปรียบเทียบค่าโดยใช้ตัวดำเนินการ ตามสมการที่ 2.4

$$T_s + n T_w \qquad (2.4)$$

เมื่อ T_s คือ เวลาเริ่มต้นในการติดต่อกับหน่วยประมวลผลอื่น

$n = \frac{N}{P}$ คือ จำนวนข้อมูลต่อชุดข้อมูลสำหรับแต่ละหน่วยประมวลผล

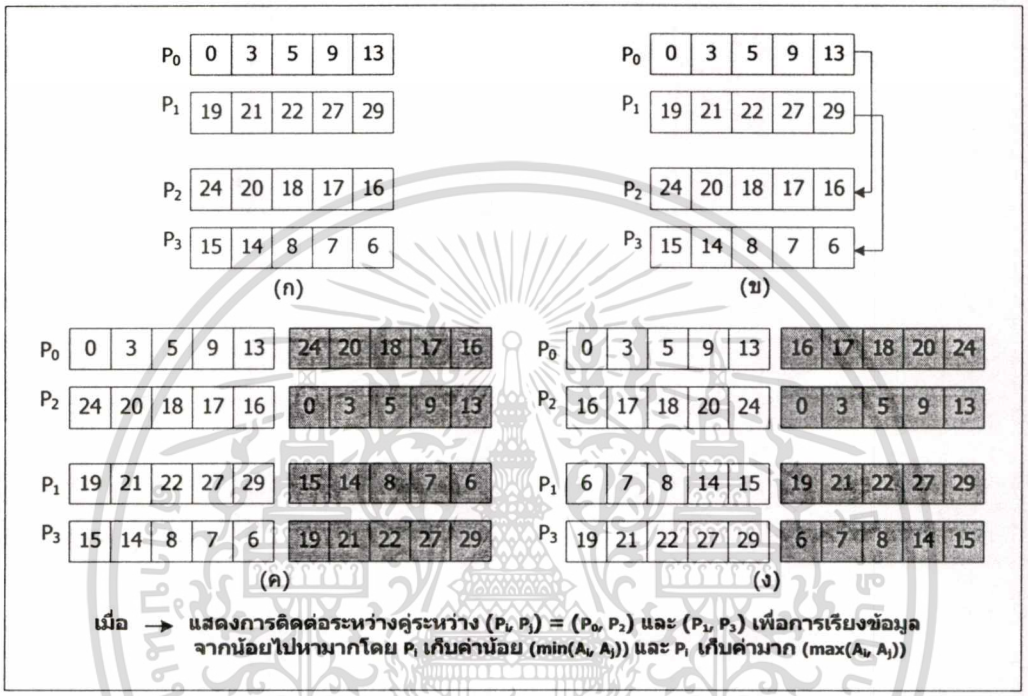
T_w คือ เวลาที่ใช้ในการแลกเปลี่ยนข้อมูล

ตัวอย่างที่ 2.3 กรณีที่ข้อมูลนำเข้าอยู่ในรูปแบบของ "Bitonic Sequence"

สมมติข้อมูลนำเข้า คือ 0, 3, 5, 9, 13, 19, 21, 22, 27, 29, 24, 20, 18, 17, 16, 15, 14, 8, 7, 6 (N=20) และจำนวนหน่วยประมวลผลเท่ากับ 4 (P=4) ขั้นตอนการเรียงลำดับข้อมูลแบบขนานด้วยวิธีไบโทนิค กรณีที่จำนวนหน่วยประมวลผลน้อยกว่าจำนวนข้อมูล (P<N) และข้อมูลนำเข้าอยู่ในรูปของ "Bitonic Sequence" สามารถคำนวณหาจำนวนรอบที่ใช้ในการเรียงลำดับข้อมูลได้ตามนิยามที่ 2 ดังนี้ $(\log_2 P) = (\log_2 4) = 2$ รอบ ดังรายละเอียดที่จะกล่าวถึงต่อไปนี้ คือ

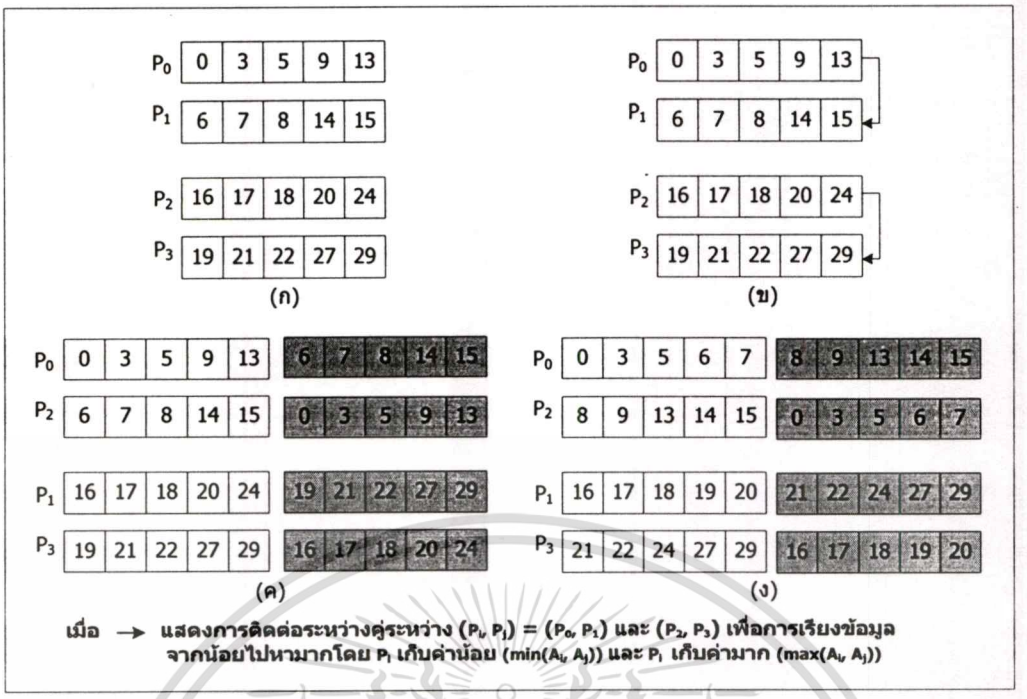
รอบที่ 1 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค ดังรูปที่ 2.16 (ก) แสดงการโหลดข้อมูลขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_0 เก็บค่า (0, 3, 5, 9, 13) P_1 เก็บค่า (19, 21, 22, 27, 29) P_2 เก็บค่า (24, 20, 18, 17, 16) และ P_3 เก็บค่า (15, 14, 8, 7, 6) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) โดยไม่ต้องทำการเรียงลำดับข้อมูลขนาด N/P

เพราะข้อมูลนี้อยู่ในรูปของ "Bitonic Sequence" และมีการเรียง 2 ชุดข้อมูลย่อยไว้แล้ว รูปที่ 2.16 (ข) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล (P_0,P_1,..., P_3 : P=4) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน เช่น คู่ (P_0, P_2) และคู่ (P_1, P_3) ในรูปที่ 2.16 (ค) แสดงการรวมชุดข้อมูลทั้ง 2 ชุดเข้าด้วยกัน (Merge Sort) และรูปที่ 2.16 (ง) เปรียบเทียบข้อมูลระหว่างหน่วยประมวลผล (P_i, P_j) ซึ่ง P_i เก็บค่าน้อย (min(A_i, A_j)) และ P_j เก็บค่ามาก (max(A_i, A_j))



รูปที่ 2.16 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (ข้อมูล "Bitonic Sequence") รอบที่ 1

รอบที่ 2 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค ดังรูปที่ 2.17 (ก) แสดงข้อมูลจากรอบที่ 1 เมื่อแต่ละหน่วยประมวลผล P_0 เก็บค่า (0, 3, 5, 9, 13) P_1 เก็บค่า (6, 7, 8, 14, 15) P_2 เก็บค่า (16, 17, 18, 20, 24) และ P_3 เก็บค่า (19, 21, 22, 27, 29) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) รูปที่ 2.17 (ข) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล (P_0,P_1,..., P_3 : P=4) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน เช่น คู่ (P_0, P_1) และคู่ (P_2, P_3) ในรูปที่ 2.17 (ค) แสดงการรวมชุดข้อมูลทั้ง 2 ชุดที่เรียงแล้วเข้าด้วยกัน (Merge Sort) และรูปที่ 2.17 (ง) เปรียบเทียบข้อมูลระหว่างหน่วยประมวลผล (P_i, P_j) ซึ่ง P_i เก็บค่าน้อย (min(A_i, A_j)) และ P_j เก็บค่ามาก (max(A_i, A_j))



รูปที่ 2.17 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (ข้อมูล "Bitonic Sequence") รอบที่ 2

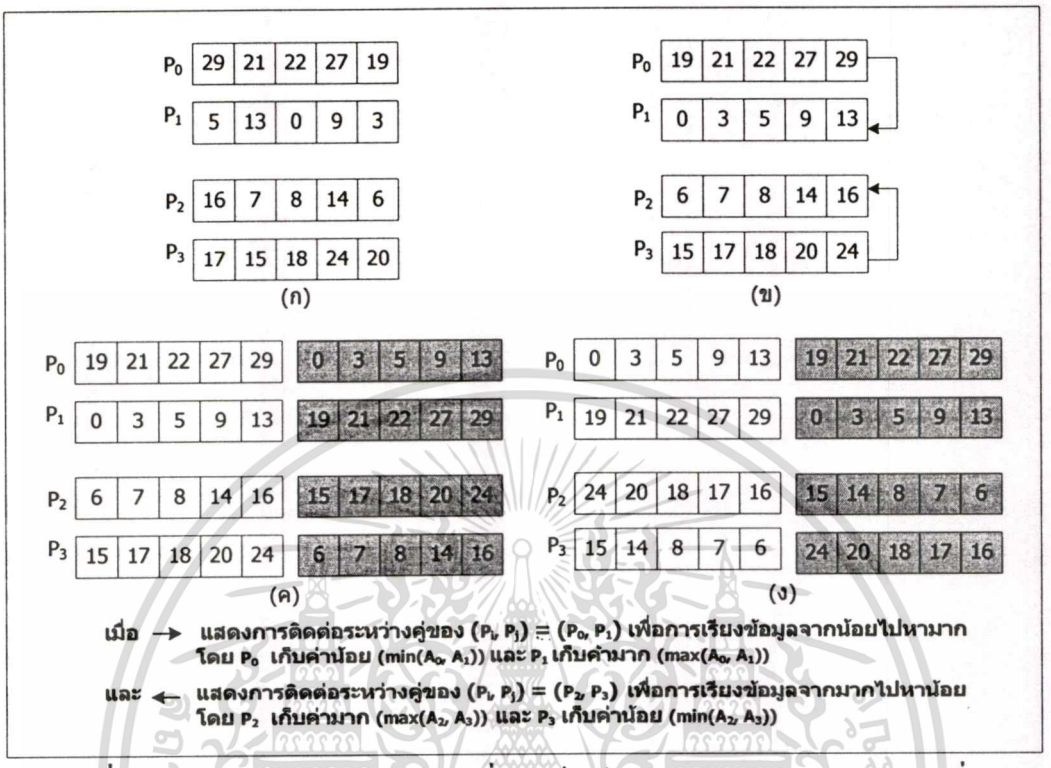
ตัวอย่างที่ 2.4 กรณีที่ข้อมูลนำเข้าอยู่ในรูปทั่วไป (ไม่อยู่ในรูปของ "Bitonic Sequence")

สมมติข้อมูลนำเข้า (N=20) คือ 29, 21, 22, 27, 19, 5, 13, 0, 9, 3, 16, 7, 8, 14, 6, 17, 15, 18, 24, 20 ซึ่งจำนวนหน่วยประมวลผล (P=4) น้อยกว่าจำนวนข้อมูล (P<N) และข้อมูลยังไม่ได้ถูกเรียงให้อยู่ในรูปของ "Bitonic Sequence" ดังนั้นการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคจึงจำเป็นต้องใช้ 2 ขั้นตอนในการเรียงลำดับข้อมูล ตามนิยามที่ 3 และสามารถคำนวณหาจำนวนรอบในการเรียงลำดับข้อมูลได้ คือ $(1+\log_2 P)(\log_2 P)/2 = (1+\log_2 4)(\log_2 4)/2 = 3$ รอบโดยมีรายละเอียดดังต่อไปนี้คือ

ขั้นแรก การแปลงข้อมูลทั่วไปให้อยู่ในรูปของ "Bitonic Sequence" ซึ่งสามารถคำนวณหาจำนวนรอบการทำงานได้ โดยนำจำนวนรอบในขั้นที่สอง ($\log_2 P$) มาหักออกจากจำนวนทั้งหมดซึ่งเท่ากับ $(1+\log_2 P)(\log_2 P)/2 - (\log_2 P) = 3 - 2 = 1$ รอบ

รอบที่ 1 การแปลงข้อมูลทั่วไปให้อยู่ในรูปของ "Bitonic Sequence" ดัง รูปที่ 2.18 (ก) แสดงการโหลดข้อมูลขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_0 เก็บค่า (29, 21, 22, 27, 19) P_1 เก็บค่า (5, 13, 0, 9, 3) P_2 เก็บค่า (16, 7, 8, 14, 6) และ P_3 เก็บค่า (17, 15, 18, 24, 20) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) และทำการเรียงลำดับข้อมูลขนาด N/P ด้วยวิธีแบบเร็ว (Quick Sort) ในทุกหน่วยประมวลผลพร้อมๆ กัน รูปที่ 2.18 (ข) แสดงข้อมูลที่เรียงแล้วและแสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล ($P_0, P_1, \ldots, P_3$: P=4) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน เช่น คู่(P_0, P_1) และ

คู่ (P_2, P_3) ในรูปที่ 2.18 (ค) แสดงการรวมชุดข้อมูลทั้ง 2 ชุดเข้าด้วยกัน (Merge Sort) และรูปที่ 2.18 (ง) เปรียบเทียบข้อมูลระหว่างหน่วยประมวลผล (P_i, P_j)



รูปที่ 2.18 การแปลงข้อมูลจากข้อมูลทั่วไปเป็นข้อมูล "Bitonic Sequence" รอบที่ 1

ขั้นที่สอง การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค สำหรับข้อมูลที่ได้รับการแปลงจากขั้นแรกคือ 0, 3, 5, 9, 13, 19, 21, 22, 27, 29, 24, 20, 18, 17, 16, 15, 14, 8, 7, 6 ซึ่งอยู่ในรูปของ "Bitonic Sequence" สามารถคำนวณหาจำนวนรอบการทำงานได้จากนิยามที่ 2 คือ $(\log_2 P) = (\log_2 4) = 2$ รอบ เช่นเดียวกับที่ได้แสดงไว้แล้วในตัวอย่างที่ 2.3

2.3.3 งานวิจัยที่เกี่ยวข้องกับการเรียงลำดับแบบขนาน

การเรียงลำดับแบบขนานใช้เวลาในการเรียงลำดับแบ่งเป็น 2 ส่วนคือ เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลแบบขนาน (Parallel Communication Time) และเวลาที่ใช้ในการประมวลผลแบบขนาน (Parallel Computation Time) ซึ่งส่วนใหญ่เวลาที่เสียไปในการเรียงลำดับแบบขนานนี้จะเกิดจากการติดต่อสื่อสารระหว่างหน่วยประมวลผล เนื่องจากขั้นตอนวิธีของการเรียงลำดับแบบขนานนั้นมีการติดต่อสื่อสารระหว่างหน่วยประมวลผลตลอดเวลาเพื่อแลกเปลี่ยนข้อมูลกัน ดังนั้นจากงานวิจัยการเรียงลำดับแบบขนานที่ได้ศึกษาจากเริ่มแรกจนถึงปัจจุบันส่วนใหญ่จะเน้นที่การติดต่อสื่อสารระหว่างหน่วยประมวลผลแบบขนาน ซึ่งสามารถแบ่งได้เป็น 3 กลุ่ม ดังนี้

1) การเรียงลำดับแบบขนานบนระบบการใช้หน่วยความจำร่วมกัน
2) การเรียงลำดับแบบขนานบนระบบการใช้หน่วยความจำแบบกระจาย
3) การเรียงลำดับแบบขนานบนระบบคลัสเตอร์

การเรียงลำดับด้วยวิธีไบโทนิค [2] นี้ได้รับความสนใจจากนักวิจัยรุ่นต่อๆ มาอย่างมาก โดยงานวิจัยส่วนใหญ่จะให้ความสนใจในเรื่องของการเพิ่มประสิทธิภาพด้านการติดต่อสื่อสารระหว่างหน่วยประมวลผลซึ่งสามารถสรุปได้ 3 แบบ ดังนี้

2.3.3.1 การเรียงลำดับแบบขนานบนระบบการใช้หน่วยความจำร่วมกัน

การเรียงลำดับแบบขนานบนระบบการใช้หน่วยความจำร่วมกัน มีโครงสร้างการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Interconnection Network) เป็นแบบไดนามิค (Dynamic) เช่น Omega, Baseline, Crossbar Switch และ Butterfly เป็นต้น ดังตัวอย่างงานวิจัยต่อไปนี้

ในปี ค.ศ. 1989 T. Nakatani และคณะ [22] เสนองานวิจัยที่อธิบายการเรียงลำดับแบบไบโทนิคด้วยวิธีเคเวย์ (K-way Bitonic Sort) โดยหลักการพื้นฐานของการเรียงลำดับแบบไบโทนิคด้วยวิธีเคเวย์นี้ได้ขยายมาจากวิธีทูเวย์ (2-way) ซึ่งหลักการนี้จะนำไปสู่การประยุกต์ (Application) ที่เป็นแบบไม่มีข้อจำกัดนั้นเอง

ในปี ค.ศ. 2000 Adler, M. Byers, J.W. Karp, R.M [1] ได้นำเสนอผลงานวิจัยที่พยายามจะปรับปรุงประสิทธิภาพในการติดต่อสื่อสารระหว่างหน่วยประมวลผล โดยใช้โมเดลที่เรียกว่า "ER-PRAM Model" โดยจะแลกเปลี่ยน (Tradeoff) ระหว่างเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Communication Time) ที่อาจจะเพิ่มขึ้นเมื่อใช้หน่วยประมวลผล (P) มากขึ้น กับเวลาที่ใช้ในการประมวลผล (Computation Time) ที่ลดลงเมื่อใช้หน่วยประมวลผลมากขึ้น ด้วยวิธี "Lower Bound" และนำการเรียงลำดับแบบคอลัมน์ (Column Sort) ที่เสนอโดย Leighton [17] มาทำการจับคู่ (Matching) เป็นวิธี "Upper Bound" ซึ่งสามารถประยุกต์เป็นโมเดลใหม่ที่เรียกว่า "Bridging Model" ที่สามารถใช้ในสถาปัตยกรรมอื่นๆ ได้

ในปี ค.ศ. 2000 Jae-Dong Lee และ Kenneth E. Batcher [16] เสนองานวิจัยนี้โดยทำการศึกษาการเรียงลำดับแบบไบโทนิค บนสถาปัตยกรรมแบบใช้หน่วยความจำร่วมกัน (Shared Memory) แต่การเข้าถึงข้อมูลในหน่วยความจำที่ใช้ร่วมกันนี้ค่อนข้างใช้เวลานาน ดังนั้นจึงเกิดแนวความคิดของกลยุทธ์แบบพาริตี้ (Parity Strategy) ขึ้น โดยการลดเวลาในการติดต่อสื่อสารระหว่างหน่วยประมวลผลเพื่อช่วยให้การเรียงลำดับข้อมูลมีประสิทธิภาพมากขึ้น

2.3.3.2 การเรียงลำดับแบบขนานบนระบบการใช้หน่วยความจำแบบกระจาย

การเรียงลำดับแบบขนานบนระบบการใช้หน่วยความจำแบบกระจายนี้ จะมีโครงสร้างการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Interconnection Network) แบบคงที่ (Static) เช่น แบบอะเรย์เชิงเส้น (Linear Array) แบบดาว (Star) แบบแหวน (Ring) แบบเม็สซ์ (Mesh) แบบไฮเปอร์คิว (HyperCube) เป็นต้น ซึ่งมีงานวิจัยที่น่าสนใจดังนี้

ในปี ค.ศ. 1977 C. D. Thompson and H. T. Kung [28] ได้แสดงการปรับปรุงเวลาที่ใช้ในการเรียงลำดับแบบไบโทนิคบนโครงสร้างการติดต่อสื่อสารของหน่วยประมวลผลแบบเม็สซ์ (Mesh) โดยในงานวิจัยนี้จะกล่าวถึงการเปลี่ยนขั้นตอนวิธีของไบโทนิคเมิจ (Bitonic Merge) ไปเป็น "Shuffle Row-Major" บนระบบคอมพิวเตอร์แบบ "ILLIAC-IV" ทำให้เวลาที่ใช้ในการประมวลผล ลดลงและในปี ค.ศ. 1983 C. D. Thompson ได้ศึกษาการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคบนระบบคอมพิวเตอร์แบบ "VLSI" (Vary Large Scale Integration) อีกด้วย

ในปี ค.ศ. 1997 M. F. Ionescu และ K. E. Schauser [11] เสนองานวิจัยที่ได้ศึกษาการเรียงลำดับแบบไบโทนิค บนระบบที่ทันสมัย ซึ่งมีความสัมพันธ์กันแบบคอร์สเกรน (Coarse grained) ซึ่งประกอบด้วยจำนวนโหนดที่เหมาะสมและแต่ละโหนดมีประสิทธิภาพสูง ดังนั้นจึงต้องทำการกำหนดข้อมูลจำนวนมากไปให้หน่วยประมวลผลแต่ละตัว (Mapping) ผลที่ตามมาก็คือเวลาที่ใช้ในการติดต่อระหว่างหน่วยประมวลผลจะสูงทำให้เกิดแนวความคิดที่จะลดขั้นตอนของการติดต่อสื่อสารและลดเวลาที่ใช้ในการคำนวณให้น้อยที่สุด โดยเสนอขั้นตอนวิธีแบบฟาสเตอร์ (Faster Algorithm) ซึ่งได้ทำการทดลองบนระบบคอมพิวเตอร์ Meiko CS-2 64 โหนด

ในปี ค.ศ. 2001 Y. C. Kim และคณะ [13] เสนองานวิจัยที่แสดงการเพิ่มประสิทธิภาพของการติดต่อสื่อสารระหว่างหน่วยประมวลผล และเวลาที่ใช้ในการเปรียบเทียบ ซึ่งบ่อยครั้งที่หน่วยประมวลผลไม่มีความจำเป็นที่จะแลกเปลี่ยนข้อมูลกัน หรือต้องการแลกเปลี่ยนข้อมูลเพียงบางส่วนเท่านั้น ในงานวิจัยนี้จะเน้นที่การแลกเปลี่ยนข้อมูลเฉพาะส่วนที่ต้องใช้ในกระบวนการเรียงลำดับข้อมูลเท่านั้น โดยหลักการนี้จะช่วยลดเวลาในการติดต่อสื่อสารระหว่างหน่วยประมวลผลลงได้อย่างน้อย 20% บนระบบคอมพิวเตอร์ Cray T3E

2.3.3.3 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคบนระบบคลัสเตอร์

การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคบนระบบคลัสเตอร์ ระบบนี้ใช้หน่วยความจำแบบกระจายเช่นกัน และมีโครงสร้างการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Interconnection Network) ส่วนใหญ่เป็นแบบ สวิตซ์เน็ตเวิร์ก (Switch Network) ดังตัวอย่างงานวิจัยต่อไปนี้

ในปี ค.ศ. 1997 Helman, D. R. and JaJa, J. [9] ได้นำเสนอผลงานวิจัยบนระบบคลัสเตอร์แบบเอสเอ็มพี (SMPs: Symmetric Multiprocessors) ซึ่งได้คิดขั้นตอนวิธีที่เรียกว่า "SMP Algorithm" โดยขั้นตอนวิธีนี้ใช้หลักการในการผสมผสานระหว่างวิธี "Random Sampling" กับวิธี "Deterministic Sampling" งานวิจัยนี้พัฒนาด้วยภาษาซี (C Language) โดยใช้โปรแกรมโพสิก (POSIX)

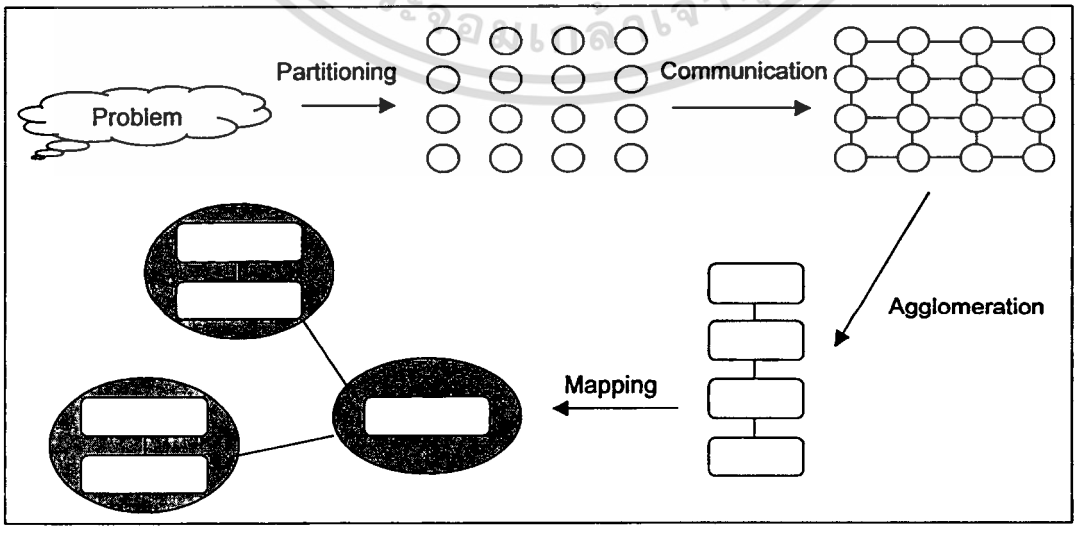
ในปี ค.ศ. 2000 J. Brest และคณะ [4] ทำงานวิจัยร่วมกันโดยนำเสนอวิธีการติดต่อสื่อสารข้อมูลระหว่างงานย่อย (Subtask) ของโปรแกรมประยุกต์ และแสดงผลการทดลองขั้นตอนวิธีการเรียงลำดับแบบขนาน (Parallel Sorting Algorithms) ด้วยวิธีการเรียงลำดับแบบเร็ว (Quick Sort) บนระบบคลัสเตอร์

2.4 การออกแบบโปรแกรมและพัฒนาโปรแกรมแบบขนาน

2.4.1 การออกแบบโปรแกรมแบบขนาน

การเขียนโปรแกรมที่ดีนั้น จะต้องมีการออกแบบโปรแกรมที่ดีก่อน โดยการออกแบบโปรแกรมที่ดีนั้นจะต้องเป็นไปตามหลักการของขั้นตอนวิธีที่ออกแบบและสามารถจัดการปัญหาต่างๆ ได้ ซึ่งวิธีการออกแบบโปรแกรมแบบขนานนี้ Foster [25] ได้นำเสนอหลักการต่างๆ ไว้ 4 ขั้นตอนด้วยกันคือ

1) การแบ่งงาน (Partition)
2) การติดต่อสื่อสาร (Communication)
3) การรวมกลุ่มงาน (Agglomerate)
4) การกำหนดงานไปยังหน่วยคำนวณที่เหมาะสม (Mapping)



รูปที่ 2.19 ขั้นตอนการออกแบบโปรแกรมแบบขนาน

ในรูปที่ 2.19 จะเห็นได้ว่าจากปัญหาขนาดใหญ่ที่ต้องการประมวลผลแบบขนานนั้น ในขั้นตอนของการแบ่งงาน (Partition) จะทำการแยกแยะปัญหาออกเป็นส่วนย่อยๆ ตามความเหมาะสมของแต่ละปัญหา หลังจากนั้นจึงทำการพิจารณาความเกี่ยวกันของแต่ละส่วนหรือรูปแบบของการติดต่อสื่อสาร (Communication) โดยอาจจะพิจารณาจากขอบเขต การอยู่ติดกันของส่วนย่อยเล็กๆ เหล่านั้น รวมทั้งพิจารณาถึงการรับส่งค่าของปัญหาต่างๆ อีกด้วย หลังจากนั้นจึงทำการรวมส่วนที่เกี่ยวข้องหรือมีความสัมพันธ์เข้าไว้ด้วยกัน ในขั้นตอนของการรวมกลุ่มงาน (Agglomeration) และสุดท้ายจะเป็นการกำหนดงานไปยังหน่วยคำนวณ (Processing Element) ที่เหมาะสมต่อไป (Mapping) ซึ่งสามารถอธิบายขั้นตอนต่างๆ ได้ดังนี้

2.4.1.1 การแบ่งงาน (Partition)

การแบ่งงานเป็นจุดเริ่มต้นของการประมวลผลแบบขนาน เพราะจะต้องทำการแบ่งปัญหาขนาดใหญ่ที่ซับซ้อนออกเป็นส่วนย่อยที่สามารถคำนวณไปพร้อมๆ กัน หลังจากนั้นจึงทำขั้นตอนอื่นๆ ต่อไป โดยการแบ่งงานสามารถแบ่งได้เป็น 2 แบบคือ

1) การแบ่งข้อมูลของปัญหาออกเป็นส่วนย่อย (Domain Decomposition or Data Decomposition) ในกรณีนี้จะเป็นการพิจารณาที่ข้อมูลเป็นหลัก ซึ่งปกติมีข้อมูลอยู่เป็นจำนวนมาก และสามารถแบ่งเป็นกลุ่มของข้อมูลย่อยได้ ตัวอย่างการประมวลผลแบบนี้เช่น โดยโปรแกรมประยุกต์ที่ใช้หลักการเขียนโปรแกรมแบบเอสไอเอสดี (SISD: Single Instruction over Single Data) และโปรแกรมที่ใช้หลักการเขียนโปรแกรมแบบเอสพีเอ็มดี (SPMD: Single Program over Multiple Data) สำหรับกรณีหลังจะเหมาะกับการประมวลผลบนระบบคลัสเตอร์ ซึ่งโปรแกรมที่ทำงานอยู่ในแต่ละหน่วยประมวลผลจะเป็นโปรแกรมชุดเดียวกัน แต่ประมวลผลข้อมูลคนละชุดกัน ในหลายๆ หน่วยประมวลผลพร้อมกัน เช่น การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค การบีบอัดข้อมูลภาพเคลื่อนไหวหรือภาพยนตร์ เป็นต้น

2) การแบ่งงานตามหน้าที่ของการคำนวณ ออกเป็นส่วนย่อย (Function Decomposition) ซึ่งการแบ่งงานแบบนี้จะกระทำตั้งแต่เริ่มลงมือเขียนส่วนของโปรแกรม เพราะจะต้องทำการแยกฟังก์ชั่นการทำงานของแต่ละส่วนอย่างชัดเจนตั้งแต่แรก ตัวอย่างการประมวลผลแบบนี้เช่น โปรแกรมประยุกต์ที่ใช้หลักการเขียนโปรแกรมแบบเอ็มไอเอ็มดี (MIMD: Multiple Instructions over Multiple Data) และโปรแกรมประยุกต์ที่ใช้หลักการเขียนโปรแกรมแบบเอ็มพีเอ็มดี (MPMD: Multiple Programs over Multiple Data) สำหรับกรณีหลังจะเหมาะกับการประมวลผลบนระบบคลัสเตอร์ ซึ่งเป็นโปรแกรมที่ต่างกันหลายๆ โปรแกรมที่ประมวลผลข้อมูลคนละชุดกัน ในหลายๆ หน่วยประมวลผลพร้อมกัน เช่น การคูณเมทริกซ์ การสร้างแบบจำลองของสภาพภูมิอากาศ ฯลฯ

2.4.1.2 การติดต่อสื่อสาร (Communication)

การติดต่อสื่อสารนี้จะพิจารณาความสัมพันธ์ในเรื่องของการสื่อสารระหว่างงานที่แบ่งออกไปในขั้นแรก แต่สำหรับงานบางชนิด งานย่อยๆ ที่ถูกแบ่งไม่สามารถประมวลผลอย่างอิสระพร้อมกันได้ในทุกขั้นตอนย่อยของการประมวลผล เนื่องจากในแต่ละขั้นตอนต้องอาศัยข้อมูลหรือผลลัพธ์จากงานย่อยอื่นๆ เพื่อใช้ในการประมวลผลขั้นต่อไป ดังนั้นจึงอาจต้องมีการสื่อสารระหว่างงานย่อย และการเลือกวิธีการสื่อสารที่เหมาะสมกับระบบคอมพิวเตอร์แบบขนานที่มีรูปแบบการติดต่อสื่อสารเฉพาะจะทำให้การประมวลผลมีประสิทธิภาพมากยิ่งขึ้น การออกแบบขั้นตอนการติดต่อสื่อสารจะแบ่งเป็นสองขั้นตอนคือ ขั้นแรกออกแบบโครงสร้างของการเชื่อมโยงว่างานใดต้องการข้อมูลและงานใดทำหน้าที่ให้ข้อมูล ขั้นที่สองกำหนดโครงสร้างของข้อความ (Message) ที่ใช้ในการส่งและรับของแต่ละการเชื่อมโยง โดยให้มีการเวลาติดต่อสื่อสารกันน้อยที่สุดเท่าที่จะทำได้

2.4.1.3 การรวมกลุ่มงาน (Agglomeration)

การรวมกลุ่มงานที่มีการติดต่อสื่อสารระหว่างกันมากๆ เข้าไว้ด้วยกัน และให้มีการติดต่อสื่อสารกันน้อยๆ ในกลุ่มใหม่หลังจากรวมกลุ่มงานจะสามารถลดเวลาที่ใช้ในการสื่อสารระหว่างงานนั้นๆ ให้ลดลงได้ ทำให้หน่วยประมวลผลเสียเวลาในการติดต่อสื่อสารและรอคอยการรับส่งข้อมูลน้อยลง ทำให้ประสิทธิภาพในการคำนวณมากขึ้นด้วย

2.4.1.4 การกำหนดงานไปยังหน่วยคำนวณที่เหมาะสม (Mapping)

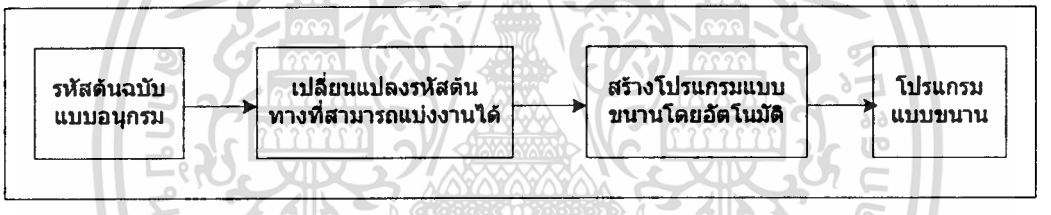
การกำหนดงานไปยังหน่วยคำนวณที่เหมาะสม เนื่องจากจำนวนกลุ่มของข้อมูล(N) อาจไม่เท่ากับจำนวนหน่วยประมวลผล (P) ซึ่งปกติจำนวนหน่วยประมวลผลจะน้อยกว่าจำนวนข้อมูล (P<N) เพื่อเป็นการใช้ประโยชน์ของหน่วยประมวลผลสูงสุด และลดการสื่อสารให้มีน้อยที่สุด การกำหนดงานนี้สามารถทำเป็นแบบคงที่ซึ่งได้กำหนดไว้ตายตัวตั้งแต่เริ่มสร้างโปรแกรม หรือทำการกำหนดตอนที่กำลังคำนวณโดยใช้เทคนิคของการสมดุลงาน (Load Balancing) เข้ามาช่วยก็ได้

2.4.2 การพัฒนาโปรแกรมแบบขนาน

การพัฒนาโปรแกรมแบบขนานสำหรับประยุกต์กับขั้นตอนวิธีแบบต่างๆ สามารถทำได้หลายแบบ ขึ้นกับวัตถุประสงค์ของผู้พัฒนาและเงื่อนไขความยากง่ายสำหรับผู้ใช้ [30] ดังนี้

2.4.2.1 การสร้างโปรแกรมแบบขนานโดยอัตโนมัติ

การสร้างโปรแกรมแบบขนานด้วยวิธีนี้เป็นวิธีที่ง่ายที่สุดสำหรับผู้ใช้ แต่จะยากที่สุดสำหรับผู้พัฒนาโดยในปัจจุบันจะมีประสิทธิภาพต่ำกว่าแบบอื่นๆ ที่จะกล่าวต่อไป กรณีนี้ผู้ใช้จะเขียนโปรแกรมแบบอนุกรม แต่หน้าที่การสร้างโปรแกรมแบบขนานจะเป็นหน้าที่ของตัวแปลภาษาแบบขนาน ซึ่งถูกพัฒนาโดยผู้พัฒนาโปรแกรมแบบขนานที่สามารถแปลงทุกชุดคำสั่งพื้นฐานแบบอัตโนมัติ โดยที่ตัวแปลภาษาดังกล่าวจะตรวจสอบรหัสต้นฉบับ (Source Code) ซึ่งอาจประกอบด้วยส่วนของรหัสที่มีการวนซ้ำและการประมวลผลกับข้อมูลที่ซ้ำๆ กันโดยตัวแปลภาษานี้จะทำการเปลี่ยนรหัสต้นฉบับไปเป็นรหัสที่สนับสนุนการประมวลผลแบบขนาน จากนั้นใช้ตัวแปลภาษาเปลี่ยนรหัสที่ได้เป็นโปรแกรมแบบขนานเองโดยอัตโนมัติ แต่ข้อจำกัดของการสร้างโปรแกรมด้วยวิธีนี้จะขึ้นอยู่กับเทคโนโลยีที่ตัวแปลภาษาแบบขนานใช้ในการเปลี่ยนรหัสต้นฉบับเป็นรหัสที่สนุบสนุนการประมวลผลแบบขนาน ขั้นตอนการทำงานของการสร้างโปรแกรมแบบขนานโดยอัตโนมัติ สามารถแสดงได้ดังรูปที่ 2.20



รูปที่ 2.20 การสร้างโปรแกรมแบบขนานโดยอัตโนมัติ

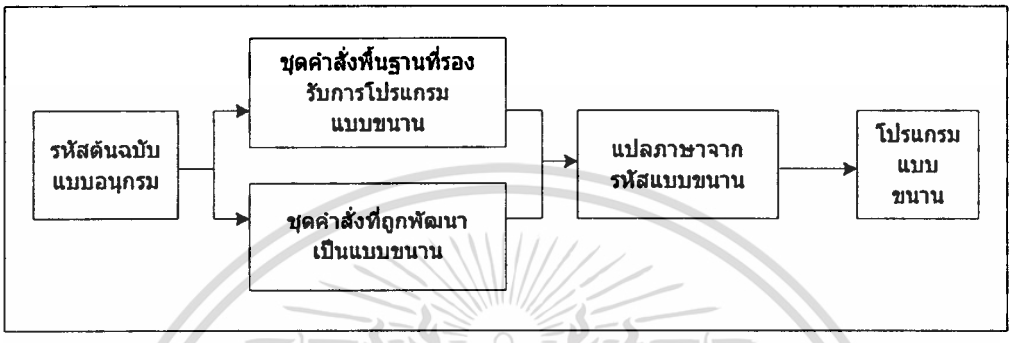
2.4.2.2 การสร้างโปรแกรมโดยใช้ชุดคำสั่งแบบขนาน

การสร้างโปรแกรมโดยใช้ชุดคำสั่งแบบขนานเป็นวิธีที่มีประสิทธิภาพมากกว่า วิธีแรกโดยเมื่อผู้ใช้ทำการสร้างโปรแกรมจะใช้ชุดคำสั่งที่ถูกพัฒนาขึ้นมาให้สนับสนุนการทำงานแบบขนานอยู่ก่อนแล้ว โดยบริษัทซอฟต์แวร์ที่พัฒนาซอฟต์แวร์ประยุกต์แบบขนานสำหรับฟังก์ชันทางคณิตศาสตร์ เช่น ชุดคำสั่งของ ScaLAPACK หรือ PLAPACK ซึ่งชุดคำสั่งเหล่านั้นสนับสนุนการประมวลผลแบบขนานอยู่แล้ว ฟังก์ชันที่ชุดคำสั่งเหล่านี้เตรียมไว้ให้ใช้งาน เช่น ชุดคำสั่งของการหาผลเฉลี่ยสมการต่างๆ แบบขนาน และการคูณเมตริกซ์แบบขนาน เป็นต้น โดยชุดคำสั่งที่สนับสนุนการทำงานแบบขนานนี้จะมีอยู่สองกลุ่ม คือ

1) กลุ่มของชุดคำสั่งพื้นฐานที่รองรับหรือทำให้เกิดสภาพแวดล้อมการประมวลผลแบบขนานตัวอย่างเช่น ฟังก์ชั่นที่ถูกเตรียมไว้โดย MPICH หรือ LAM ซึ่งจะเป็นคำสั่งหรือฟังก์ชั่นพื้นฐานที่ทำให้เกิดการประมวลผลแบบขนานได้ เพราะมีคำสั่งที่ทำให้เกิดการสื่อสาร

ระหว่างหน่วยประมวลผลต่อหน่วยประมวลผล การส่งข้อความถึงหน่วยประมวลผลอื่นๆ ในระบบได้ และอื่นๆ อีกมาก

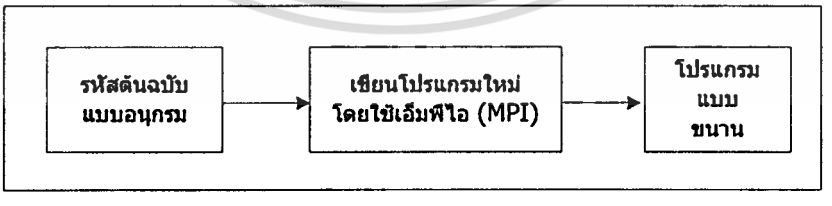
2) กลุ่มของชุดคำสั่งที่ได้รับการพัฒนาให้เป็นการประมวลผลแบบขนาน เช่น ScaLAPACK หรือ PLAPACK ซึ่งเป็นชุดคำสั่งทางคณิตศาสตร์ที่ได้รับการพัฒนาให้ทำงานแบบขนานได้โดยเรียกใช้ชุดคำสั่งของมาตรฐานภาษาเอ็มพีไอ (MPI Standard) อีกที



รูปที่ 2.21 การสร้างโปรแกรมโดยใช้ชุดคำสั่งแบบขนาน

2.4.2.3 การสร้างโปรแกรมแบบขนานด้วยตนเอง

การสร้างโปรแกรมแบบขนานด้วยวิธีนี้เป็นวิธีที่ยืดหยุ่นมากที่สุด ผู้สร้างโปรแกรมจะเขียนโปรแกรมที่สนับสนุนการทำงานแบบขนาน รวมถึงสามารถเลือกรูปแบบและวิธีการที่ใช้ในการสื่อสาร แต่วิธีนี้นับเป็นวิธีที่ยากและเสียเวลามากที่สุดสำหรับผู้พัฒนาและผู้ใช้ซึ่งมักจะเป็นคนเดียวกัน ขั้นตอนการสร้างโปรแกรมแบบขนานด้วยตัวเองสามารถแสดงได้ดังรูปที่ 2.22 โดยผู้เขียนส่วนใหญ่จะสร้างโปรแกรมแบบอนุกรม (Sequential Program) ก่อน จากนั้นจึงพัฒนาต่อโดยใช้เอ็มพีไอ (MPI) หรือพีวีเอ็ม (PVM) ซึ่งในวิทยานิพนธ์นี้ใช้เอ็มพีไอ ในการพัฒนาโปรแกรมแบบขนาน (Parallel Program)



รูปที่ 2.22 การสร้างโปรแกรมแบบขนานด้วยตนเอง

2.4.3 มาตรฐานภาษาเอ็มพีไอ (MPI Standard: Message-Passing Interface Standard)

ในปี ค.ศ. 1990 ได้มีการกำหนดมาตรฐานในการส่งผ่านข้อมูล (Message-Passing) [8] [18] [25] ขึ้นใหม่นั้นคือ มาตรฐานภาษาเอ็มพีไอ (MPI Standard) เนื่องจากมาตรฐานที่มีอยู่ก่อนหน้านี้

คือมาตรฐานพีวีเอ็ม (PVM Standard) ซึ่งเป็นมาตรฐานที่ได้มีการใช้งานอย่างแพร่หลายใน ค.ศ. 1980 แต่ด้วยจุดอ่อนของพีวีเอ็มในหลายๆ ด้านทำให้การพัฒนาโปรแกรมแบบขนานมีความยุ่งยากซับซ้อนมากกว่าเอ็มพีไอ ด้วยเหตุนี้จึงทำให้มาตรฐานภาษาเอ็มพีไอได้รับความนิยมในเวลาต่อมา

การพัฒนาโปรแกรมแบบขนานบนระบบคลัสเตอร์ ซึ่งจะมีไลบรารี (Library) และฟังก์ชันพื้นฐานต่างๆ เป็นตัวกลางในการเชื่อมการทำงานของแต่ละหน่วยประมวลผลเข้าด้วยกัน เพื่อให้หน่วยประมวลผลสามารถทำงานรวมกันได้อย่างสะดวกและมีประสิทธิภาพ ซึ่งจะทำให้นักพัฒนาโปรแกรมสามารถพัฒนาโปรแกรมได้ง่ายและสะดวกรวดเร็วมากยิ่งขึ้น ด้วยภาษาต่างๆ ตามความถนัดของนักพัฒนา เช่น ภาษาฟอร์แทน (Fortan Language) ภาษาซี (C Language) เป็นต้น

เนื่องจากเอ็มพีไอได้รับความนิยมจากนักพัฒนาโปรแกรมแบบขนาน ทำให้มีซอฟท์แวร์จำนวนมากที่สนับสนุนมาตรฐานภาษาเอ็มพีไอทั้งในเชิงพาณิชย์ (Commercial) เช่น เอ็มพีไอโปร (MPI/Pro) [5] หรือแบบให้เปล่า (Open Source) เช่น เอ็มพีไอซีเอช (MPICH) เป็นต้น

ขั้นตอนการพัฒนาโปรแกรมแบบขนานบนระบบคลัสเตอร์ โดยใช้เอ็มพีไอซึ่งทุกหน่วยประมวลผลทำงานไปพร้อมๆ กัน มีขั้นตอนการทำงานอธิบายกว้างๆ ดังนี้

1) เริ่มต้นการทำงานของการเขียนโปรแกรมแบบขนาน (Start MPI)
2) การติดต่อสื่อสารระหว่างหน่วยประมวลผลในการแลกเปลี่ยนส่ง/รับข้อมูล
3) การประยุกต์ใช้ เช่น การเรียงลำดับข้อมูลที่มีอยู่และที่ได้รับจากหน่วยประมวลผลอื่นๆ
4) รอผลลัพธ์ (ถ้ามี)
5) หยุดการทำงานของการเขียนโปรแกรมแบบขนาน (Stop MPI)

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main(int argc, char**agrv) {

int id, size;

(1) Start MPI Environment
if (MPI_Init(&argc, &argv) !=
   MPI_SUCCESS) {
    fprintf(stdout,
    "Error to start MPI\n");
    exit(-1);
}

(2) MPI Communication
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&id);

(4) Wait for result
fprintf(stdout, "Hi,  i am id %d of %d CPU",id,size);

(5) Stop MPI Environment
MPI_Finalize();

  return(0);
}
```

รูปที่ 2.23 ตัวอย่างโปรแกรมแสดงหมายเลขประจำหน่วยประมวลผล ใช้ 5 หน่วยประมวลผล

รูปที่ 2.23 เป็นตัวอย่างโปรแกรมแบบขนานขั้นพื้นฐาน ที่แสดงหมายเลขประจำหน่วยประมวลผล (Processor ID) โดยเขียนตามมาตรฐานภาษาเอ็มพีไอ (MPI Standard) ซึ่งแสดงขั้นตอนการพัฒนาโปรแกรมแบบขนานที่ได้กล่าวในข้างต้น ตามข้อ 1, 2, 4 และ 5 โดยไม่รวมการประยุกต์ใช้ในขั้นที่ 3 พร้อมทั้งแสดงผลจากการประมวลผลโปรแกรมที่แสดงหมายเลขประจำหน่วยประมวลผล โดยใช้หน่วยประมวลผลในการรัน (Run) โปรแกรมตัวอย่างมีจำนวน 5 หน่วยประมวลผล

ผลจากการประมวลผลโปรแกรมแสดงหมายเลขประจำหน่วยประมวลผล รูปที่ 2.23 คือ

```
Hi, I am id 0 of 5 CPU
Hi, I am id 1 of 5 CPU
Hi, I am id 2 of 5 CPU
Hi, I am id 3 of 5 CPU
Hi, I am id 4 of 5 CPU
```

2.5 การวัดประสิทธิภาพ

การวัดประสิทธิภาพของการเรียงลำดับข้อมูลแบบขนานในงานวิจัยนี้จะประเมินผลจากเวลาที่ใช้ในการเรียงลำดับข้อมูล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) ในการประมวลผลจริงบนระบบคลัสเตอร์สามารถวัดเวลาที่ใช้ได้โดยการจับเวลาที่ใช้ในการเรียงลำดับ (Sorting) ทั้งการเรียงลำดับแบบอนุกรม (Sequential Sort) และการเรียงลำดับแบบขนาน (Parallel Sort) โดยจะทำการวัดเวลาที่ใช้ในการเรียงลำดับ ซึ่งไม่รวมเวลาที่ใช้ในการอ่านข้อมูลมาเก็บไว้ในหน่วยความจำและการตั้งค่าระบบ (Initialization) ของ MPI ซึ่งกระบวนการทั้งสองจะถูกทำครั้งเดียวในการเรียงลำดับข้อมูล

2.5.1 เวลาที่ใช้ในการประมวลผล (Response Time)

การวัดเวลาที่ใช้ในการประมวลผลสำหรับการเรียงลำดับแบบขนาน ในทางทฤษฎี (Theoretical Approach) เวลาที่ใช้ในการประมวลผลแบบขนาน (Parallel Computation) จะสามารถคำนวณได้ดังสมการที่ 2.5 [19]

$$T_p = \frac{T_s}{P} \tag{2.5}$$

เมื่อ T_p คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วย P หน่วยประมวลผล

T_s คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วยหนึ่งหน่วยประมวลผล หรือเวลาที่ใช้ในการประมวลผลข้อมูลแบบอนุกรมนั้นเอง

P คือ จำนวนหน่วยประมวลผล (Processor) ที่ใช้ในระบบคลัสเตอร์

ซึ่งกรณีนี้จะเรียกว่าเป็นกรณีอุดมคติ (Ideal Case) เพราะว่ามีการสมมติให้เวลาที่ใช้ในการติดต่อสื่อสาร (Communication Time) ระหว่างหน่วยประมวลผลมีค่าเป็นศูนย์ แต่ในทางปฏิบัติ (Practical Approach) บนระบบคลัสเตอร์ การวัดเวลาที่ใช้ในการประมวลผลแบบขนานจะวัดโดยการจับเวลา ตั้งแต่เริ่มประมวลผล จนกระทั้งสิ้นสุดการประมวลผล หรือได้คำตอบที่ต้องการ ซึ่งเวลาที่วัดได้ดังกล่าวจะรวมเวลาที่ใช้ในการติดต่อสื่อสารด้วย ดังนั้นถึงแม้ว่าจำนวนของหน่วยประมวลผล (P) ในระบบจะมีเพิ่มขึ้นมากๆ แต่เวลาที่ได้ของระบบจะไม่ลดลงจนเข้าใกล้ศูนย์ เพราะสาเหตุมาจากเวลาที่ใช้ในการประมวลผลทั้งหมดประกอบด้วยเวลาที่ใช้ในการประมวลผล (Computation Time) และเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Communication Time) ซึ่งปกติจะไม่เป็นศูนย์ ดังกรณีอุดมคติ โดยจะมีลักษณะตามสมการที่ 2.6

$$T_p = t_{computation} + t_{communication} \qquad (2.6)$$

เมื่อ T_p คือ เวลาที่ใช้ในการประมวลผลแบบขนานทั้งหมดด้วย P หน่วย

$t_{computation}$ คือ เวลาที่ใช้ในการประมวลผล (เช่น การเรียงลำดับข้อมูล) ซึ่งไม่รวมเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล

$t_{communication}$ คือ เวลาที่ใช้ในการสื่อสารระหว่างหน่วยประมวลผล (ถ้ามี)

ดังนั้นในทางปฏิบัติเราสามารถเพิ่มหน่วยประมวลผลเข้าไปในระบบมากขึ้น และเวลาจะลดลงช่วงหนึ่งเท่านั้น เพราะเวลาที่ใช้ในการประมวลผล (Computation Time) มากกว่าเวลาที่ใช้ในการติดต่อสื่อสาร (Communication Time) ระหว่างหน่วยประมวลผล แต่ ณ จุดหนึ่งเมื่อเพิ่มหน่วยประมวลผลเข้าไปอีก เวลาจะไม่ลดลงแต่อาจจะเพิ่มขึ้นเนื่องจากเวลาที่ใช้ในการสื่อสารระหว่างหน่วยประมวลผล $t_{communication}$ มากขึ้นและมีค่ามากกว่า เวลาที่ใช้ในการประมวลผล

2.5.2 อัตราการเพิ่มของความเร็ว (Speedup)

นอกจากการวัดเวลาที่ใช้ในการประมวลผลแล้วยังสามารถวัดอัตราการเพิ่มของความเร็ว (Speedup) ของการประมวลผลแบบขนานได้ ทั้งในทางทฤษฎีและทางปฏิบัติจากสมการที่ 2.7 [19]

$$S_p = \frac{T_s}{T_p} \leq P \tag{2.7}$$

เมื่อ S_p คือ อัตราการเพิ่มของความเร็วที่ใช้ในประมวลผลแบบขนาน โดยใช้ P หน่วยประมวลผล ซึ่งมีค่าสูงสุดในอุดมคติเท่ากับ P

T_s คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วยหนึ่งหน่วยประมวลผล

T_p คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วย P หน่วยประมวลผล

2.5.3 ประสิทธิภาพ (Efficiency)

นอกจากการวัดเวลาที่ใช้ในการประมวลผล และอัตราการเพิ่มขึ้นของความเร็วแล้ว ยังสามารถวัดประสิทธิภาพของการประมวลผลแบบขนาน โดยใช้สมการที่ 2.8 ดังนี้

$$E_p = \frac{S_p}{P} \leq 1 \tag{2.8}$$

เมื่อ E_p คือ ประสิทธิภาพของการประมวลผลแบบขนานมีค่าในอุดมคติเท่ากับ 1

S_p คือ อัตราการเพิ่มของความเร็วที่คำนวณได้จากสมการที่ 2.7

P คือ จำนวนหน่วยประมวลผลที่ใช้บนระบบคลัสเตอร์

บทที่ 3
การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค

งานวิจัยนี้เป็นการวิจัยทดลอง โดยผู้วิจัยจะศึกษาขั้นตอนวิธีการเรียงลำดับแบบขนานด้วยวิธีไบโทนิค ซึ่งเน้นกรณีทั่วไปที่มีจำนวนหน่วยประมวลผลน้อยกว่าจำนวนข้อมูล ($P < N$) โดยงานวิจัยนี้จะมุ่งเน้นที่การเพิ่มประสิทธิภาพของวิธีการติดต่อสื่อสารระหว่างหน่วยประมวลผลของการเรียงลำดับแบบขนานด้วยวิธีไบโทนิค และการลดการใช้เนื้อที่ในการเก็บข้อมูลที่ซ้ำซ้อนกันในระบบ ซึ่งผู้วิจัยแบ่งวิธีการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคที่เพิ่มประสิทธิภาพต่างๆ ดังกล่าวได้ 4 วิธีดังนี้

1) การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบเดิม (Bitonic sort: BS)

2) การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคที่ทำการเพิ่มประสิทธิภาพเพื่อลดการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล (Communication-Efficient Bitonic Sort: CEBS) ในบางช่วงเวลาที่ไม่จำเป็นออกจากกระบวนการเรียงลำดับ

3) การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคที่ทำการเพิ่มประสิทธิภาพเพื่อลดการใช้พื้นที่ในหน่วยความจำเก็บข้อมูลที่ซ้ำซ้อนในระบบ (Space-Efficient Bitonic Sort: SEBS)

4) การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคโดยนำข้อดีของข้อ 2) และข้อ 3) มาประยุกต์รวมกัน ทำให้สามารถเพิ่มประสิทธิภาพของการใช้พื้นที่ในหน่วยความจำและลดการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล โดยจะเลือกแลกเปลี่ยนชุดข้อมูลที่มีขนาดเล็กที่สุดเท่าที่จำเป็นเท่านั้น (Communication-Space Efficient Bitonic Sort: CSEBS)

โดยทุกวิธีดังกล่าวที่เสนอจะถูกพัฒนาขึ้นโดยใช้เทคนิคการโปรแกรมแบบขนานด้วยมาตรฐานภาษาเอ็มพีไอ เพื่อใช้เป็นเครื่องมือในการทดลองเปรียบเทียบประสิทธิภาพที่แตกต่างกัน โดยวัดผลขั้นต้นจากเวลาที่ใช้ในการประมวลผล (Response Time) จากนั้นสามารถแสดงอัตราการเพิ่มของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) ของการเรียงลำดับข้อมูลทุกวิธีดังกล่าว

3.1 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (Bitonic Sort : BS)

วิธีการเรียงลำดับด้วยวิธีไบโทนิคแบบบีเอส (BS) [2] [23] จะใช้ตัวดำเนินการที่เรียกว่า "Compare-Split Operation" [14] ซึ่งเป็นตัวดำเนินการสำหรับเปรียบเทียบข้อมูลที่ติดต่อสื่อสารกันระหว่างหน่วยประมวลผลแต่ละคู่ (P_i, P_j) ดังรูปที่ 3.1 (ก) แสดงการโหลดข้อมูลขนาด N/P แต่ละหน่วยประมวลผล P_i เก็บค่า (6, 1, 11, 15, 8) และ P_j เก็บค่า (16, 20, 13, 10, 12) ไว้ในหน่วยความจำส่วนตัว (Local Memory) และทำการเรียงลำดับข้อมูลขนาด N/P ด้วยวิธีแบบเร็ว

(Quick Sort) ในทุกหน่วยประมวลผลพร้อมๆ กัน รูปที่ 3.1 (ข) แสดงผลจากการเรียงลำดับข้อมูลในข้อ (ก) และการติดต่อสื่อสารระหว่าง P_i กับ P_j รูปที่ 3.1 (ค) แสดงการแลกเปลี่ยนชุดข้อมูลระหว่าง P_i กับ P_j รวมถึงการรวมชุดข้อมูลทั้ง 2 ชุดเข้าด้วยกัน (Merge Sort) รูปที่ 3.1 (ง) เปรียบเทียบค่าที่รับมากับค่าที่มีอยู่ โดย P_i จะเก็บค่าน้อย ($\min(A_i, A_j)$) ส่วน P_j จะเก็บค่ามาก ($\max(A_i, A_j)$)

รูปที่ 3.1 ตัวดำเนินการที่เรียกว่า "Compare-Split Operation"

ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี (MPI-C like) ของการเรียงลำดับแบบขนานด้วยวิธีไบโทนิค

```
Start
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &p);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
Generate Data
 srand(my_rank);
 for (i = 0; i < list_size; i++)  local_list[i] = rand() % KEY_MAX;
Local Sort
 qsort(local_keys, list_size, sizeof(KEY_T),
       int(*)(const void*, const void*))(Key_compare));
Bitonic Sort(BS)
 if ((myrank & and_bit) == 0) // increase
    Bitonic_increasing(list_size, num, proc_set_size); // increasing
 else // decrease
    Bitonic_decreasing (list_size, num, proc_set_size); // decreasing
Partner Communication (send and receive data)
 MPI_Comm_rank(comm, &myrank);
    proc_set_dim = log_base2(proc_set_size);
    eor_bit = 1 << (proc_set_dim - 1);
    for (stage = 0; stage < proc_set_dim; stage++) {
       partner = my_rank ^ eor_bit;
        if (myrank > partner) // OR myrank > partner for Bitonic_decreasing
              Merge_list_low(list_size, num, partner);
        else
              Merge_list_high(list_size, num, partner);
        eor_bit = eor_bit >> 1; }
    end for
```

รูปที่ 3.2 ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี การเรียงลำดับแบบขนานด้วยวิธี "BS"

ตัวอย่างการเรียงลำดับแบบขนานด้วยวิธีไบโทนิค (BS) กรณีที่ข้อมูลนำเข้าอยู่ในรูปทั่วๆ ไป (ไม่ได้อยู่ในรูปแบบของ "Bitonic Sequence") แสดงไว้แล้วในบทที่ 2 ดังตัวอย่างที่ 2.4 เมื่อ N=20 และ P=4

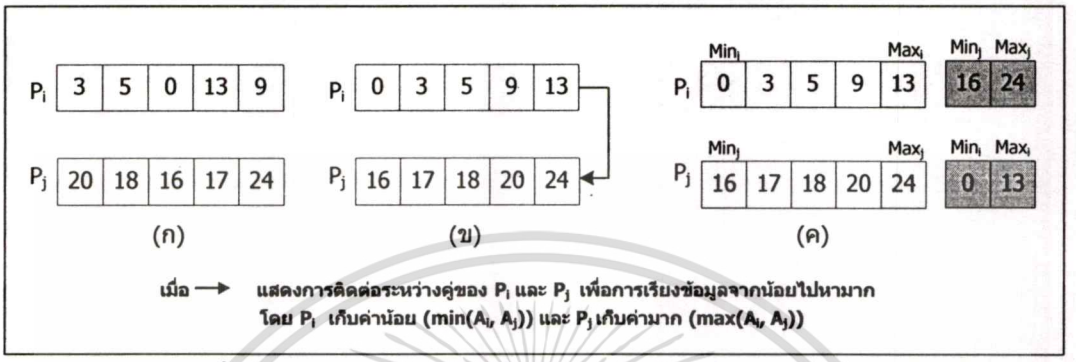
3.2 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบ "Communication-Efficient Bitonic Sort: CEBS"

วิธีการเรียงลำดับแบบซีอีบีเอส (Communication-Efficient Bitonic Sort: CEBS) เป็นวิธีที่เน้นในเรื่องของการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล เพื่อให้การแลกเปลี่ยนข้อมูลเกิดขึ้นเฉพาะช่วงเวลาที่จำเป็นเท่านั้น เพราะเวลาในการแลกเปลี่ยนข้อมูลมีผลอย่างมากต่อเวลาในการเรียงลำดับข้อมูล กรณีนี้สามารถแบ่งลักษณะการแลกเปลี่ยนข้อมูลได้ 3 แบบ (Pattern) โดยมีแนวความคิดจากงานวิจัยการเรียงลำดับข้อมูลแบบขนานด้วยวิธีไบโทนิคบนระบบคอมพิวเตอร์ Cray T3E [13] ซึ่งสามารถเพิ่มประสิทธิภาพในการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลได้อย่างน้อย 20% ซึ่งแต่ละแบบมีรายละเอียดดังนี้

3.2.1 การแลกเปลี่ยนข้อมูลแบบคงที่ (Hold Pattern)

การแลกเปลี่ยนข้อมูลแบบคงที่นี้เป็นกรณีที่ดีที่สุดสำหรับการเรียงลำดับ เนื่องจากข้อมูลได้ถูกเรียงลำดับไว้อย่างเหมาะสมแล้วตามวิธีการของไบโทนิค ดังนั้นจึงไม่ต้องมีการแลกเปลี่ยนข้อมูลทำให้เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Communication Time) ลดลง และไม่ต้องทำการเรียงลำดับข้อมูลใหม่แบบรวมชุดข้อมูล 2 ชุดเข้าด้วยกัน (Merge Sort) เนื่องจากข้อมูลทั้ง 2 ชุดถูกจัดเรียงไว้เหมาะสมแล้วใน P_i และ P_j ทำให้เวลาที่ใช้ในการประมวลผล (Computation Time) ลดลง ดังรูปที่ 3.3 (ก) แสดงการโหลดข้อมูลขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_i เก็บค่า (0, 3, 5, 9, 13) และ P_j เก็บค่า (16, 17, 18, 20, 24) ไว้ในหน่วยความจำส่วนตัว (Local Memory) และทำการเรียงลำดับข้อมูลขนาด N/P ด้วยวิธีแบบเร็ว (Quick Sort) ในทุกหน่วยประมวลผลพร้อมๆ กัน รูปที่ 3.3 (ข) แสดงข้อมูลที่เรียงแล้วและแสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล P_i กับ P_j รูปที่ 3.3 (ค) แสดงค่าน้อย (Min) และค่ามาก (Max) ในกรอบสีเทาเท่านั้นที่ต้องส่งและรับระหว่าง P_i และ P_j เพื่อตรวจสอบว่าเป็นการแลกเปลี่ยนข้อมูลแบบคงที่หรือไม่ โดยมีข้อกำหนดดังต่อไปนี้

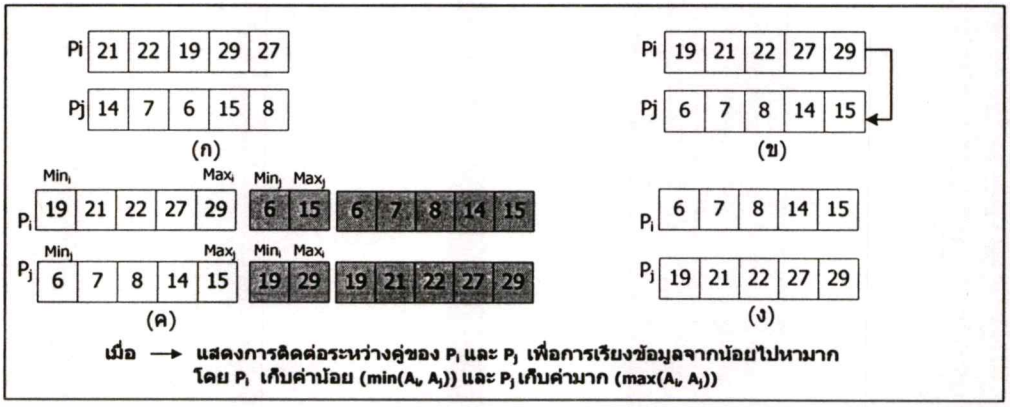
เมื่อ Min_j ที่รับมาจาก P_j มีค่ามากกว่าหรือเท่ากับ Max_i ใน P_i (หรือทุกๆ ค่าใน P_i นั้นเอง) และ Max_i ที่รับมาจาก P_i มีค่าน้อยกว่าหรือเท่ากับ Min_jใน P_j (หรือทุกๆ ค่าใน P_j นั้นเอง) แสดงว่าเป็นการแลกเปลี่ยนข้อมูลแบบคงที่ ดังนั้น ในกรณีนี้ไม่ต้องมีการแลกเปลี่ยนชุดข้อมูลขนาด N/P ข้อมูลระหว่าง P_i และ P_j



รูปที่ 3.3 แสดงวิธีการแลกเปลี่ยนข้อมูลแบบคงที่ "Hold Pattern"

3.2.2 การแลกเปลี่ยนข้อมูลแบบสลับที่ (Swap Pattern)

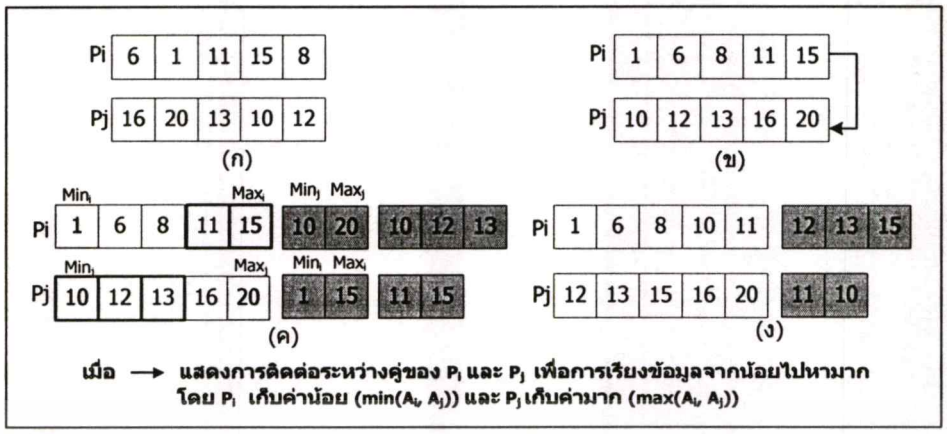
การแลกเปลี่ยนข้อมูลแบบสลับที่เป็นการแลกเปลี่ยนข้อมูลทั้งชุดโดยไม่ต้องทำการเรียงลำดับข้อมูลใหม่เนื่องจากข้อมูลทั้ง 2 ชุดถูกจัดเรียงไว้เหมาะสมแล้ว แต่ไม่ได้อยู่ใน P_i และ P_j ตามที่ต้องการ ดังนั้นกรณีนี้ P_i และ P_j เพียงส่งชุดข้อมูลขนาด N/P ไปให้กับหน่วยประมวลผลที่ทำการติดต่อด้วยเท่านั้น ทำให้เวลาที่ใช้ในการประมวลผล (Computation Time) ลดลง ดังแสดงในรูปที่ 3.4 (ก) แสดงการโหลดข้อมูลขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_i เก็บค่า (21, 22, 19, 29, 27) และ P_j เก็บค่า (14, 7, 6, 15, 8) ไว้ในหน่วยความจำส่วนตัว (Local Memory) และทำการเรียงลำดับข้อมูลขนาด N/P ด้วยวิธีแบบเร็ว (Quick Sort) ในทุกหน่วยประมวลผลพร้อมๆ กัน รูปที่ 3.4 (ข) แสดงข้อมูลที่เรียงแล้วและแสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล P_i กับ P_j รูปที่ 3.4 (ค) แสดงค่าน้อย (Min) และค่ามาก (Max) ในกรอบสีเทาเท่านั้นที่ต้องส่งและรับระหว่าง P_i และ P_j เพื่อตรวจสอบว่าเป็นการแลกเปลี่ยนข้อมูลแบบสลับที่ โดยมีเงื่อนไขดังนี้ Min_i ใน P_i มีค่ามากกว่าหรือเท่ากับ Max_j ที่รับมาจาก P_j (หรือทุกๆ ค่าใน P_j นั้นเอง) และ Max_j ใน P_j มีค่าน้อยกว่าหรือเท่ากับ Min_i ที่รับมาจาก P_i (หรือทุกๆ ค่าใน P_i นั้นเอง) ถ้าเป็นไปตามเงื่อนไขดังกล่าว แสดงว่าเป็นการแลกเปลี่ยนแบบสลับที่ (Swap Pattern) ดังนั้นในกรณีนี้จะทำการส่งและรับข้อมูลระหว่างหน่วยประมวลผล (P_i, P_j) ที่ทำการติดต่อด้วยเท่านั้น แต่ไม่ต้องทำการรวมหรือเรียงลำดับข้อมูลใหม่ และรูปที่ 3.4 (ง) แสดงค่าที่รับมาเก็บแทนค่าที่มีอยู่เดิม โดย P_i จะเก็บค่าน้อย ($\min(A_i, A_j)$) ส่วน P_j จะเก็บค่ามาก ($\max(A_i, A_j)$)



รูปที่ 3.4 แสดงวิธีการแลกเปลี่ยนข้อมูลแบบสลับที่ "Swap Pattern"

3.2.3 การแลกเปลี่ยนข้อมูลแบบเฉพาะส่วน (Partial Pattern)

โดยทั่วไปแล้วขนาดของข้อมูลในแต่ละหน่วยประมวลผลจะมีขนาดเท่ากับ N/P เมื่อ N คือจำนวนข้อมูล และ P คือจำนวนของหน่วยประมวลผล ดังนั้นการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลปกติจะเท่ากับ N/P แต่สำหรับการแลกเปลี่ยนข้อมูลแบบเฉพาะส่วน จะต้องคำนวณหาชุดข้อมูลย่อยเฉพาะที่จำเป็นต้องส่งและรับเท่านั้นโดยส่วนใหญ่แล้วจะมีจำนวนน้อยกว่า N/P ทำให้สามารถลดทั้งเวลาที่ใช้ในการติดต่อสื่อสาร (Communication Time) และเวลาที่ใช้ในการประมวลผล (Computation Time) ดังรูปที่ 3.5 (ก) แสดงการโหลดข้อมูลขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_i เก็บค่า (6, 1, 11, 15, 8) และ P_j เก็บค่า (16, 20, 13, 10, 12) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) และทำการเรียงลำดับข้อมูลขนาด N/P ด้วยวิธีแบบเร็ว (Quick Sort) ในทุกหน่วยประมวลผลพร้อมๆ กัน รูปที่ 3.5 (ข) แสดงการติดต่อสื่อสารระหว่าง P_i กับ P_j รูปที่ 3.5 (ค) แสดงการส่งค่าน้อย (Min) และค่ามาก (Max) ไปให้กับหน่วยประมวลผลที่ติดต่อด้วย เพื่อตรวจสอบหาชุดข้อมูลย่อยที่ต้องการแลกเปลี่ยนเท่านั้น จากนั้นนำค่า Min_j มาตรวจสอบกับค่าใน P_i เริ่มจากค่า Max_i ไปยังค่า Min_i โดยที่ค่า Min_j ต้องมีค่ามากกว่าค่าใน P_i ดังนั้น ชุดข้อมูลที่ต้องถูกส่งไปให้กับ P_j เพื่อทำการเรียงลำดับข้อมูลคือค่าใน P_i ที่มากกว่า Min_j (กรณีนี้กรอบเข้มใน P_i เป็นค่าส่งและกรอบสีเทาใน P_j เป็นค่ารับ) และในขณะเดียวกันนำค่า Max_i มาตรวจสอบกับค่าใน P_j เริ่มจากค่า Min_j ไปยังค่า Max_j โดยที่ค่า Max_i ต้องมีค่าน้อยกว่าค่าใน P_j ดังนั้นชุดข้อมูลที่ต้องถูกส่งไปให้กับ P_i เพื่อทำการเรียงลำดับข้อมูลคือค่าใน P_j ที่น้อยกว่าค่า Max_i (กรณีนี้กรอบเข้มใน P_j เป็นค่าส่ง และกรอบสีเทาใน P_i เป็นค่ารับ) รูปที่ 3.5 (ง) เปรียบเทียบค่าที่รับมากับค่าที่มีอยู่ โดย P_i จะเก็บค่าน้อย ($\min(A_i, A_j)$) ส่วน P_j จะเก็บค่ามาก ($\max(A_i, A_j)$)



รูปที่ 3.5 แสดงวิธีการแลกเปลี่ยนข้อมูลแบบเฉพาะส่วน "Partial Pattern"

หมายเหตุ ในขั้นตอนการพัฒนาโปรแกรมด้วยวิธีซีอีบีเอส (CEBS) การแลกเปลี่ยนข้อมูลแบบคงที่จะถูกตรวจสอบก่อน ถ้าไม่เป็นไปตามเงื่อนไขดังกล่าว การแลกเปลี่ยนแบบสลับที่ จะถูกตรวจสอบลำดับถัดมา และถ้าไม่เป็นไปตามเงื่อนไขทั้งสองนี้ จึงทำการแลกเปลี่ยนแบบเฉพาะส่วน

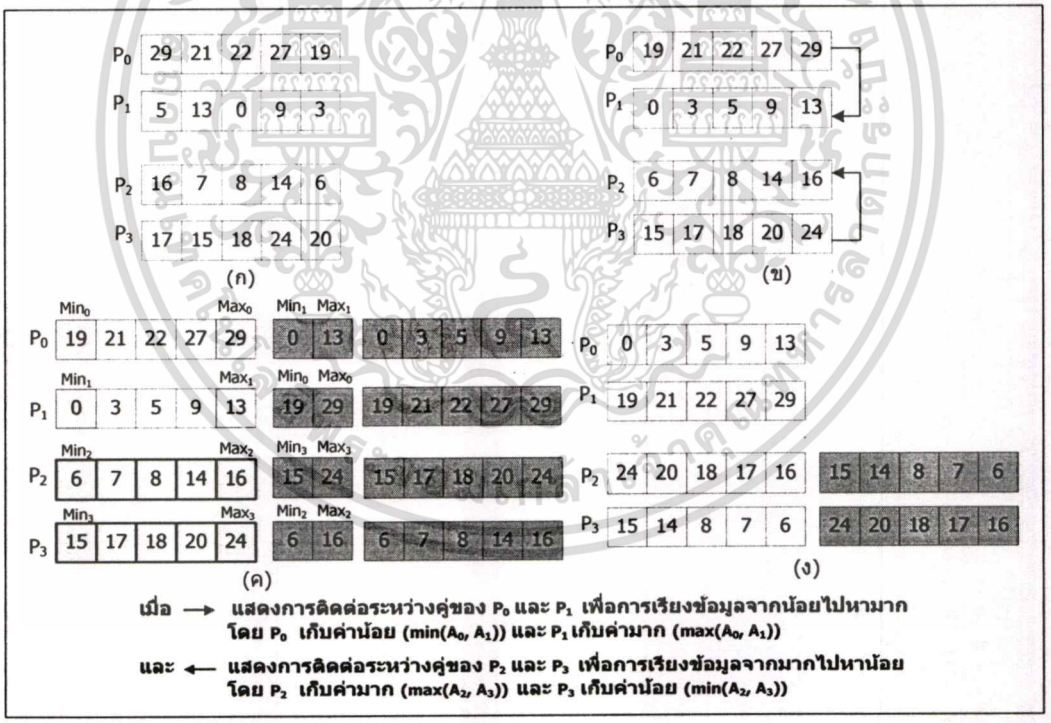
ตัวอย่างที่ 3.1 กรณีที่ข้อมูลนำเข้าอยู่ในรูปทั่วไป (ไม่อยู่ในรูปของ "Bitonic Sequence")

สมมติข้อมูลนำเข้าคือ 29, 21, 22, 27, 19, 5, 13, 0, 9, 3, 16, 7, 8, 14, 6, 17, 15, 18, 24, 20 (N=20) ซึ่งจำนวนหน่วยประมวลผล (P=4) น้อยกว่าจำนวนข้อมูล (P<N) และข้อมูลยังไม่ได้ถูกเรียงลำดับให้อยู่ในรูปของ "Bitonic Sequence" ดังนั้นการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคจึงจำเป็นต้องใช้ 2 ขั้นตอนในการเรียงลำดับข้อมูล ตามนิยามที่ 3 ซึ่งสามารถคำนวณหาจำนวนรอบในการเรียงลำดับข้อมูลได้ดังนี้ $(1+ \log_2 P)(\log_2 P)/2 = (1+ \log_2 4)(\log_2 4)/2 = 3$ รอบ โดยมีรายละเอียดที่จะได้กล่าวดังต่อไปนี้คือ

ขั้นแรก การแปลงข้อมูลทั่วไปให้อยู่ในรูปของ "Bitonic Sequence" ซึ่งสามารถคำนวณหาจำนวนรอบการทำงานได้ โดยนำจำนวนรอบในขั้นที่สอง ($\log_2 P$) มาหักออกจากจำนวนทั้งหมดซึ่งเท่ากับ $(1+ \log_2 P)(\log_2 P)/2 - (\log_2 P) = 3 - 2 = 1$ รอบ

รอบที่ 1 การแปลงข้อมูลจากข้อมูลทั่วไปให้อยู่ในรูปแบบของ "Bitonic Sequence" ดังรูปที่ 3.6 (ก) แสดงการโหลดข้อมูลขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_0 เก็บค่า (29, 21, 22, 27, 19) P_1 เก็บค่า (5, 13, 0, 9, 3) P_2 เก็บค่า (16, 7, 8, 14, 6) และ P_3 เก็บค่า (17, 15, 18, 24, 20) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) และทำการเรียงลำดับข้อมูลขนาด N/P ด้วยวิธีแบบเร็ว (Quick Sort) ในทุกหน่วยประมวลผลพร้อมๆ กัน รูปที่ 3.6 (ข) แสดงข้อมูลที่เรียงแล้วและแสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล ($P_0, P_1, \ldots, P_3$: P=4) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน รูปที่ 3.6 (ค) ทำการ

ส่งค่าน้อย (Min) และค่ามาก (Max) ไปให้กับหน่วยประมวลผลที่ติดต่อด้วย เพื่อตรวจสอบหารูปแบบของการส่งข้อมูลดังที่ได้กล่าวมาแล้วข้างต้น (Swap Pattern, Hold Pattern, Partial Pattern) เช่น คู่ (P_0, P_1) เป็นการแลกเปลี่ยนชุดข้อมูลแบบสลับที่ (Swap Pattern) โดยนำค่า Max_1 มาตรวจสอบกับค่า Min_0 ใน P_0 โดยที่ค่า Max_1 ต้องมีค่าน้อยกว่าค่า Min_0 ใน P_0 ในขณะเดียวกันนำค่า Min_0 มาตรวจสอบกับค่า Max_1 ใน P_1 โดยที่ค่า Min_0 ต้องมีค่ามากกว่าค่า Max_1 ใน P_1 ส่วนคู่ (P_2, P_3) เป็นการแลกเปลี่ยนข้อมูลแบบเฉพาะส่วน (Partial Pattern) โดยนำค่า Min_2 มาตรวจสอบกับค่าใน P_3 เริ่มจากค่า Max_3 ไปยังค่า Min_3 โดยที่ค่า Min_2 ต้องมีค่ามากกว่าค่าใน P_3 ดังนั้นชุดข้อมูลที่ต้องถูกส่งไปให้กับ P_2 เพื่อทำการเรียงลำดับข้อมูลคือ ค่าใน P_3 ที่มากกว่า Min_2 (กรอบเข้มใน P_3 เป็นค่าส่งและกรอบสีเทาใน P_2 เป็นค่ารับ) และในขณะเดียวกันนำค่า Max_3 มาตรวจสอบกับค่าใน P_2 เริ่มจากค่า Min_2 ไปยังค่า Max_2 โดยที่ค่า Max_3 ต้องมีค่าน้อยกว่าค่าใน P_2 ดังนั้นชุดข้อมูลที่ต้องถูกส่งไปให้กับ P_3 เพื่อทำการเรียงลำดับข้อมูลคือ ค่าใน P_2 ที่น้อยกว่าค่า Max_3 (กรอบเข้มใน P_2 เป็นค่าส่งและกรอบสีเทาใน P_3 เป็นค่ารับ) รูปที่ 3.6 (ง) เปรียบเทียบค่าที่รับมากับค่าที่มีอยู่ (ถ้ามี) และแสดงผลการเรียงลำดับข้อมูลที่เหมาะสม



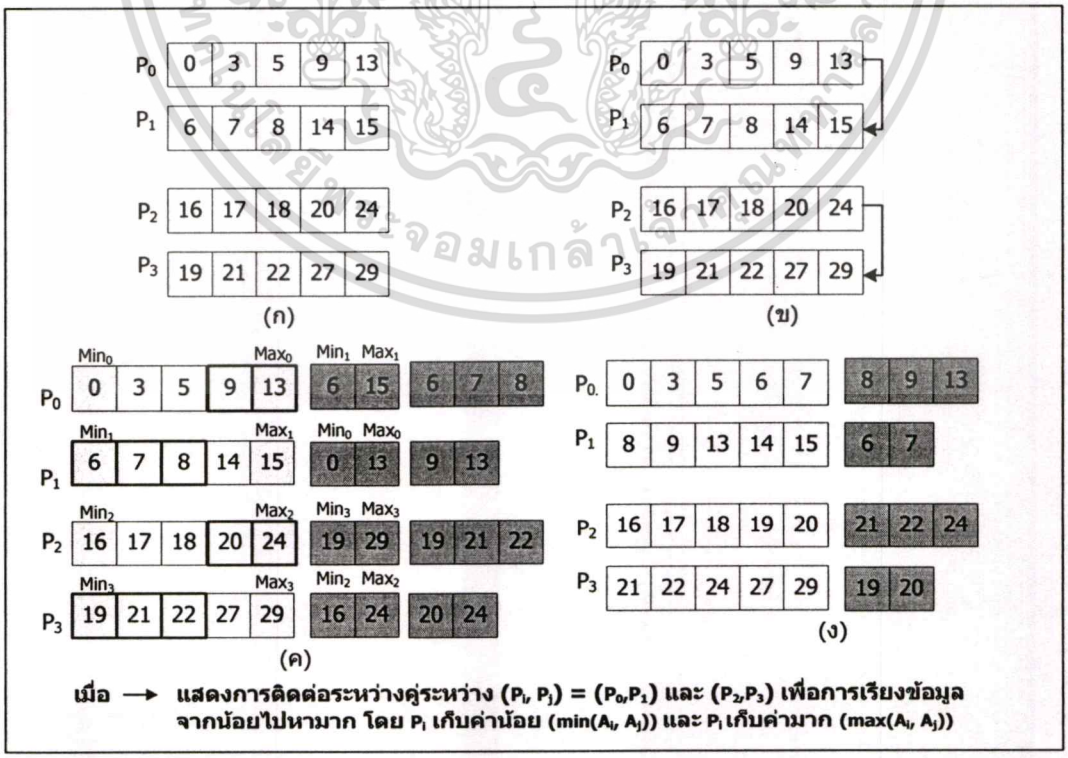
รูปที่ 3.6 การแปลงข้อมูลจากข้อมูลทั่วไปเป็นข้อมูล "Bitonic Sequence" รอบที่ 1

<u>ขั้นที่สอง</u> การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค สำหรับข้อมูลที่ได้รับการแปลงจากขั้นแรก 0, 3, 5, 9, 13, 19, 21, 22, 27, 29, 24, 20, 18, 17, 16, 15, 14, 8, 7, 6 ซึ่งอยู่ในรูปของ "Bitonic Sequence" สามารถคำนวณหาจำนวนรอบการทำงานได้จากนิยามที่ 2 คือ $(\log_2 P) = (\log_2 4) = 2$ รอบ ดังนี้คือ

รอบที่ 2 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค ดังรูปที่ 3.7 (ก) แสดงข้อมูลในรอบที่ 1 ขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_0 เก็บค่า (0, 3, 5, 9, 13) P_1 เก็บค่า (19, 21, 22, 27, 29) P_2 เก็บค่า (24, 20, 18, 17, 16) และ P_3 เก็บค่า (15, 14, 8, 7, 6) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) รูปที่ 3.7 (ข) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล (P_0,P_1,..., P_3 : P=4) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน รูปที่ 3.7 (ค) ทำการส่งค่าน้อย (Min) และค่ามาก (Max) ไปให้กับหน่วยประมวลผลที่ติดต่อด้วย เพื่อตรวจสอบหารูปแบบของการส่งข้อมูลดังที่ได้กล่าวมาแล้วข้างต้น (Swap Pattern, Hold Pattern, Partial Pattern) เช่น คู่ (P_0, P_2) เป็นการแลกเปลี่ยนชุดข้อมูลแบบคงที่ (Hold Pattern) โดยค่า Min_2 ที่รับมาจาก P_2 มีค่ามากกว่าหรือเท่ากับ Max_0 ใน P_0 (หรือทุกๆ ค่าใน P_0 นั้นเอง) และค่า Max_0 ที่รับมาจาก P_0 มีค่าน้อยกว่าหรือเท่ากับ Min_2ใน P_2 (หรือทุกๆ ค่าใน P_2 นั้นเอง) ในกรณีนี้ไม่ต้องมีการแลกเปลี่ยนชุดข้อมูลขนาด N/P ข้อมูลระหว่าง P_0 และ P_2 ส่วนคู่ (P_1, P_3) เป็นการแลกเปลี่ยนข้อมูลแบบสลับที่ (Swap Pattern) นำค่า Min_1 ใน P_1 มีค่ามากกว่าหรือเท่ากับ Max_3 ที่รับมาจาก P_3 (หรือทุกๆ ค่าใน P_3 นั้นเอง) และ Max_3 ใน P_3 มีค่าน้อยกว่าหรือเท่ากับ Min_1 ที่รับมาจาก P_1 (หรือทุกๆ ค่าใน P_1 นั้นเอง) ดังนั้นในกรณีนี้จะทำการส่ง (กรอบเข้ม) และรับ (กรอบสีเทา) ข้อมูลระหว่างหน่วยประมวลผล (P_1, P_3) ที่ทำการติดต่อด้วยเท่านั้น แต่ไม่ต้องทำการเรียงลำดับข้อมูลใหม่ รูปที่ 3.7 (ง) แสดงค่าที่รับมา (ถ้ามี) และแสดงผลการเรียงลำดับข้อมูลที่เหมาะสม

รูปที่ 3.7 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบ "CEBS" รอบที่ 2

รอบที่ 3 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค ดังรูปที่ 3.8 (ก) แสดงข้อมูลในรอบที่ 2 ขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_0 เก็บค่า (0, 3, 5, 9, 13) P_1 เก็บค่า (6, 7, 8, 14, 15) P_2 เก็บค่า (16, 17, 18, 20, 24) และ P_3 เก็บค่า (19, 21, 22, 27, 29) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) รูปที่ 3.8 (ข) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล (P_0,P_1,..., P_3 : P=4) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน รูปที่ 3.8 (ค) ทำการส่งค่าน้อย (Min) และค่ามาก (Max) ไปให้กับหน่วยประมวลผลที่ติดต่อด้วย เพื่อตรวจสอบหารูปแบบของการส่งข้อมูลดังที่ได้กล่าวมาแล้วข้างต้น (Swap Pattern, Hold Pattern, Partial Pattern) เช่น คู่ (P_0, P_1) เป็นการแลกเปลี่ยนชุดข้อมูลแบบเฉพาะส่วน (Partial Pattern) โดยนำค่า Min_1 มาตรวจสอบกับค่าใน P_0 เริ่มจากค่า Max_0 ไปยังค่า Min_0 โดยที่ค่า Min_1 ต้องมีค่ามากกว่าค่าใน P_0 ดังนั้นชุดข้อมูลที่ต้องถูกส่งไปให้กับ P_1 เพื่อทำการเรียงลำดับข้อมูลคือค่าใน P_0 ที่มากกว่า Min_1 (กรอบเข้มใน P_0 เป็นค่าส่ง และกรอบสีเทาใน P_1 เป็นค่ารับ) และในขณะเดียวกันนำค่า Max_0 มาตรวจสอบกับค่าใน P_1 เริ่มจากค่า Min_0 ไปยังค่า Max_0 โดยที่ค่า Max_0 ต้องมีค่าน้อยกว่าค่าใน P_1 ดังนั้นชุดข้อมูลที่ต้องถูกส่งไปให้กับ P_0 เพื่อทำการเรียงลำดับข้อมูลคือค่าใน P_1 ที่น้อยกว่าค่า Max_0 (กรอบเข้มใน P_1 เป็นค่าส่ง และกรอบสีเทาใน P_0 เป็นค่ารับ) ส่วนคู่ (P_2, P_3) เป็นการแลกเปลี่ยนข้อมูลแบบเฉพาะส่วน (Partial Pattern) ซึ่งมีขั้นตอนการตรวจสอบเช่นเดียวกับคู่ (P_0, P_1) รูปที่ 3.8 (ง) เปรียบเทียบค่าที่รับมากับค่าที่มีอยู่ และแสดงผลการเรียงลำดับข้อมูลที่เหมาะสม



รูปที่ 3.8 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบ "CEBS" รอบที่ 3

ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบ "CEBS"

```
Start
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &p);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

Generate Data
  srand(my_rank);
  for (i = 0; i < list_size; i++)  local_list[i] = rand() % KEY_MAX;

Local Sort
  qsort(local_keys, list_size, sizeof(KEY_T),
        int(*)(const void*, const void*))(Key_compare));

Bitonic Sort(BS)
  if ((myrank & and_bit) == 0) // increase
      Bitonic_increasing(list_size, num, proc_set_size); // increasing
  else // decrease
      Bitonic_decreasing (list_size, num, proc_set_size); // decreasing

Partner Communication
  MPI_Comm_rank(comm, &myrank);
      proc_set_dim = log_base2(proc_set_size);
      eor_bit = 1 << (proc_set_dim - 1);
      for (stage = 0; stage < proc_set_dim; stage++) {
         partner = my_rank ^ eor_bit;
         if (myrank < partner) // OR myrank > partner for Bitonic_decreasing
             Check_pattern(list_size, num, myrank, partner,0,stage);
         }
         else {
             Check_pattern(list_size, num, myrank, partner,1,stage);
         }
         eor_bit = eor_bit >> 1;
   }
Check_pattern
   if (value ==0)
      if (min_p >= num[last])
         printf("STAGE = %d Hold Pattern partner %d of myrank %d\n",stage ,partner,myrank);
      else if (max_p <= num[0])
         Merge_list_swap(list_size, num, partner);
      else
         Merge_list_low(list_size, num, partner,min_p, myrank);
    else
      if(max_p <= num[0])
         printf("STAGE = %d Hold Pattern partner %d of myrank %d\n",stage ,partner,myrank);
      else if (min_p >= num[last])
        Merge_list_swap(list_size, num, partner);
      else
         Merge_list_high(list_size, num, partner,max_p, myrank);

Pattern Send and Receive data
   if (value ==0)
      if (min_p >= num[last])
        printf("STAGE = %d Hold Pattern partner %d of myrank %d\n",stage ,partner,myrank);
      else if (max_p <= num[0])
        Merge_list_swap(list_size, num, partner);
      else
        Merge_list_low(list_size, num, partner,min_p, myrank);
    else
       if(max_p <= num[0])
         printf("STAGE = %d Hold Pattern partner %d of myrank %d\n",stage ,partner,myrank);
       else if (min_p >= num[last])
         Merge_list_swap(list_size, num, partner);
       else
         Merge_list_high(list_size, num, partner,max_p, myrank);
```

รูปที่ 3.9 ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบ "CEBS"

3.3 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบ "Space-Efficient BS: SEBS"

วิธีการเรียงลำดับแบบเอสอีบีเอส (SEBS) นี้ เป็นวิธีการใหม่ที่นำเสนอในวิทยานิพนธ์นี้ โดยจะใช้ตัวดำเนินการที่เรียกว่า "Efficient Compare-Split Operation" เป็นตัวดำเนินการสำหรับเปรียบเทียบข้อมูลที่ติดต่อสื่อสารกันระหว่างหน่วยประมวลผลแต่ละคู่ (P_i, P_j) ซึ่งวิธีการนี้ช่วยเพิ่มประสิทธิภาพโดยลดการใช้พื้นที่ในหน่วยความจำที่เก็บข้อมูลที่ซ้ำซ้อนในระบบ ซึ่งสามารถลดลงได้ถึง 50% เทียบกับแบบบีเอส (BS) เดิม นอกจากนี้มีการเพิ่มประสิทธิภาพในขั้นตอนของการรวมข้อมูลทั้ง 2 ชุดข้อมูลเข้าด้วยกัน (Merge Sort) โดยก่อนการรวมจะทำการตรวจสอบข้อมูลโดยใช้หลักการเช่นเดียวกับแบบซีอีบีเอส (CEBS) แต่ต่างกันที่กรณีนี้มีจำนวนครั้งในการรับและส่งข้อมูลระหว่าง P_i และ P_j น้อยกว่าแบบซีอีบีเอส (CEBS) ซึ่งครั้งแรกจะส่งค่ามาก (Max) และค่าน้อย (Min) ไปก่อนเพื่อตรวจสอบ และเฉพาะกรณีการแลกเปลี่ยนข้อมูลแบบสลับที่และแบบเฉพาะส่วน (Swap and Partial Pattern) จะต้องส่งข้อมูลทั้งชุดอีกครั้งระหว่าง P_i และ P_j แต่แบบเอสอีบีเอส (SEBS) ที่นำเสนอนี้ P_j เท่านั้นที่ส่งข้อมูลทั้งชุดไปให้กับ P_i เพียงครั้งเดียวโดยไม่ต้องส่งค่ามาก (Max) และค่าน้อย (Min) ไปก่อน (หรือหน่วยประมวลผลที่มีหมายเลขมากกว่า (P_j) จะเป็นผู้ส่งข้อมูลไปให้กับหน่วยประมวลผลที่มีหมายเลขน้อยกว่า (P_i)) และทำการตรวจสอบรูปแบบการแลกเปลี่ยนข้อมูล (Hold Pattern, Swap Pattern and Partial Pattern) ใน P_i ซึ่งวิธีการดังกล่าวนี้จะทำให้เวลาที่ใช้ในการเรียงลำดับข้อมูลลดลง เนื่องจากกรณีการแลกเปลี่ยนข้อมูลแบบคงที่ (Hold Pattern) จะไม่ต้องมีการส่งข้อมูลทั้งชุดกลับจาก P_i ไปยัง P_j แต่จะส่งแค่ผลที่แสดงว่าเป็นการรวมข้อมูลแบบคงที่ (Hold Pattern) เช่น H เท่านั้น และการส่งข้อมูลจะเกิดเฉพาะกรณีการแลกเปลี่ยนข้อมูลแบบสลับที่และแบบเฉพาะส่วน (Swap and Partial Pattern) เท่านั้น นอกจากนี้กรณีการแลกเปลี่ยนข้อมูลแบบเฉพาะส่วน (Partial Pattern) ข้อมูลที่ถูกส่งกลับอาจมีค่าน้อยกว่า N/P ทำให้เวลาที่ใช้ในการเรียงลำดับข้อมูลลดลงอีกด้วย ดังรูปที่ 3.10 (ก) แสดงการโหลดข้อมูลขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_i เก็บค่า (6, 1, 11, 15, 8) และP_j เก็บค่า (16, 20, 13,10, 12) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผลและทำการเรียงลำดับข้อมูลขนาด N/P ด้วยวิธีแบบเร็ว (Quick Sort) ในทุกหน่วยประมวลผลพร้อมๆ กัน รูปที่ 3.10 (ข) แสดงผลจากการเรียงลำดับข้อมูลในข้อ (ก) และแสดงการติดต่อสื่อสารระหว่าง P_i กับ P_j รูปที่ 3.10 (ค) แสดงการแลกเปลี่ยนชุดข้อมูลโดยที่ P_j ส่งข้อมูลไปให้ P_i เพื่อทำการรวม (Merge Sort) ชุดข้อมูลทั้ง 2 ชุดเข้าด้วยกัน โดยจะประยุกต์วิธีการของซีอีบีเอส (CEBS) ที่ได้กล่าวมาแล้วในข้างต้นมาใช้ในการรวมข้อมูลซึ่งมีวิธีการตรวจสอบ (Hold, Swap and Partial Pattern) เช่น คู่ (P_i, P_j) เป็นการแลกเปลี่ยนข้อมูลแบบเฉพาะส่วน (Partial Pattern) โดยนำค่า Max_i มาตรวจสอบ โดยเริ่มจากค่า Min_j ไปยังค่า Max_j โดยที่ค่า Max_i ต้องมีค่าน้อยกว่าค่าในกรอบสีเทาที่ถูกส่งมาจาก P_j ดังนั้นค่าที่

อยู่ในกรอบเข้มใน P_i และกรอบเข้มสีเทาในส่วนที่รับมาคือ ข้อมูลที่ใช้ในการรวมเข้าด้วยกัน และจำนวนข้อมูลที่ต้องส่งให้ P_j คือจำนวนในกรอบเข้มสีเทา ส่วน P_j จะรอรับค่าจาก P_i หลังกระบวนการรวมชุดข้อมูลทั้ง 2 ชุดเข้าด้วยกันเสร็จสิ้น รูปที่ 3.10 (ง) ข้อมูลในกรอบสีเทา (กรณีน้อยกว่า N/P) จะถูกส่งคืนหลังการเปรียบเทียบค่าให้กับหน่วยประมวลผลที่ติดต่อ (ส่งคืนให้กับหน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลมากกว่า) พร้อมกับค่าที่ระบุว่าเป็น P (Partial Pattern)

รูปที่ 3.10 ตัวดำเนินการที่เรียกว่า "Efficient Compare-Split Operation"

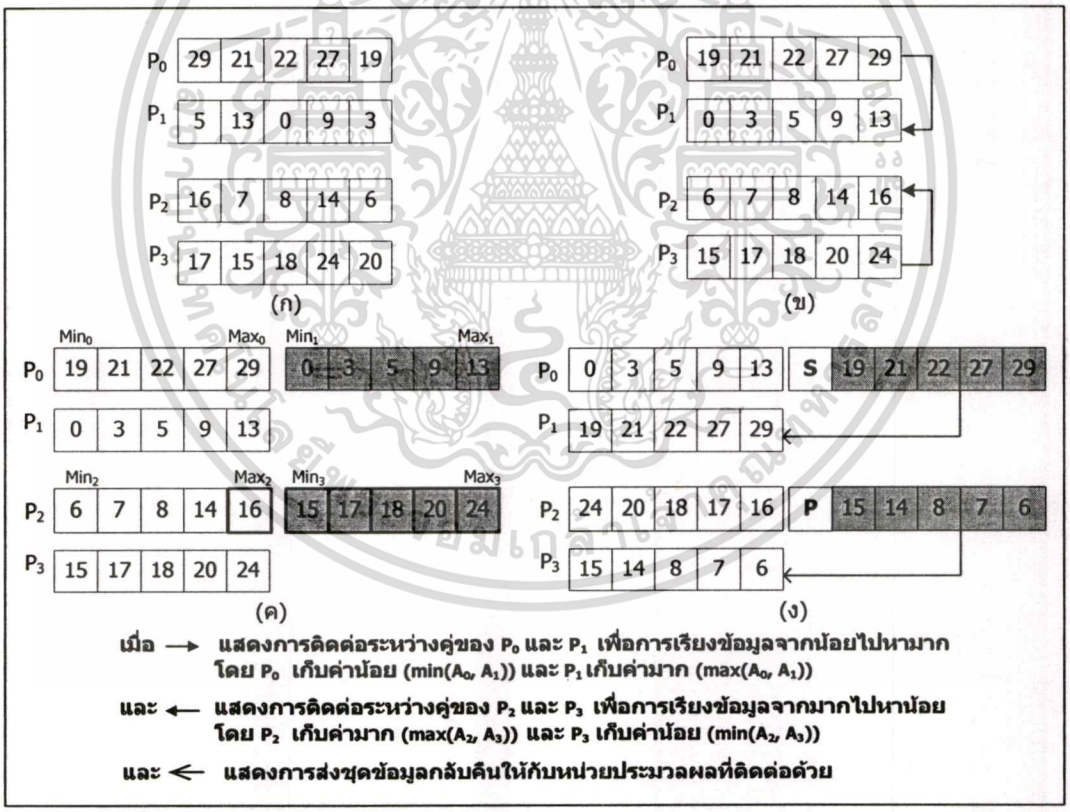
ตัวอย่างที่ 3.2 กรณีที่ข้อมูลนำเข้าอยู่ในรูปทั่วไป (ไม่อยู่ในรูปของ "Bitonic Sequence")

สมมติข้อมูลนำเข้า (N=20) คือ 29, 21, 22, 27, 19, 5, 13, 0, 9, 3, 16, 7, 8, 14, 6, 17, 15, 18, 24, 20 ซึ่งจำนวนหน่วยประมวลผล (P=4) น้อยกว่าจำนวนข้อมูล (P<N) และข้อมูลยังไม่ได้ถูกเรียงลำดับให้อยู่ในรูปของ "Bitonic Sequence" ดังนั้นการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคจึงจำเป็นต้องใช้ 2 ขั้นตอนในการเรียงลำดับข้อมูล ตามนิยามที่ 3 ซึ่งสามารถคำนวณหาจำนวนรอบในการเรียงลำดับข้อมูลได้ดังนี้ $(1+\log_2 P)(\log_2 P)/2 = (1+\log_2 4)(\log_2 4)/2 = 3$ รอบ โดยมีรายละเอียดดังต่อไปนี้

ขั้นแรก การแปลงข้อมูลทั่วไปให้อยู่ในรูปของ "Bitonic Sequence" ซึ่งสามารถคำนวณหาจำนวนรอบการทำงานได้ โดยนำจำนวนรอบในขั้นที่สอง ($\log_2 P$) มาหักออกจากจำนวนทั้งหมดซึ่งเท่ากับ $(1+\log_2 P)(\log_2 P)/2 - (\log_2 P) = 3 - 2 = 1$ รอบ

รอบที่ 1 การแปลงข้อมูลจากข้อมูลทั่วไปให้อยู่ในรูปแบบของ "Bitonic Sequence" ดังรูปที่ 3.11 (ก) แสดงการโหลดข้อมูลขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_0 เก็บค่า (29, 21, 22, 27, 19) P_1 เก็บค่า (5, 13, 0, 9, 3) P_2 เก็บค่า (16, 7, 8, 14, 6) และ P_3 เก็บค่า (17, 15, 18, 24, 20) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) รูปที่ 3.11 (ข)) แสดงข้อมูลที่เรียงแล้วและแสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล ($P_0, P_1, \ldots, P_3$: P=4) โดย

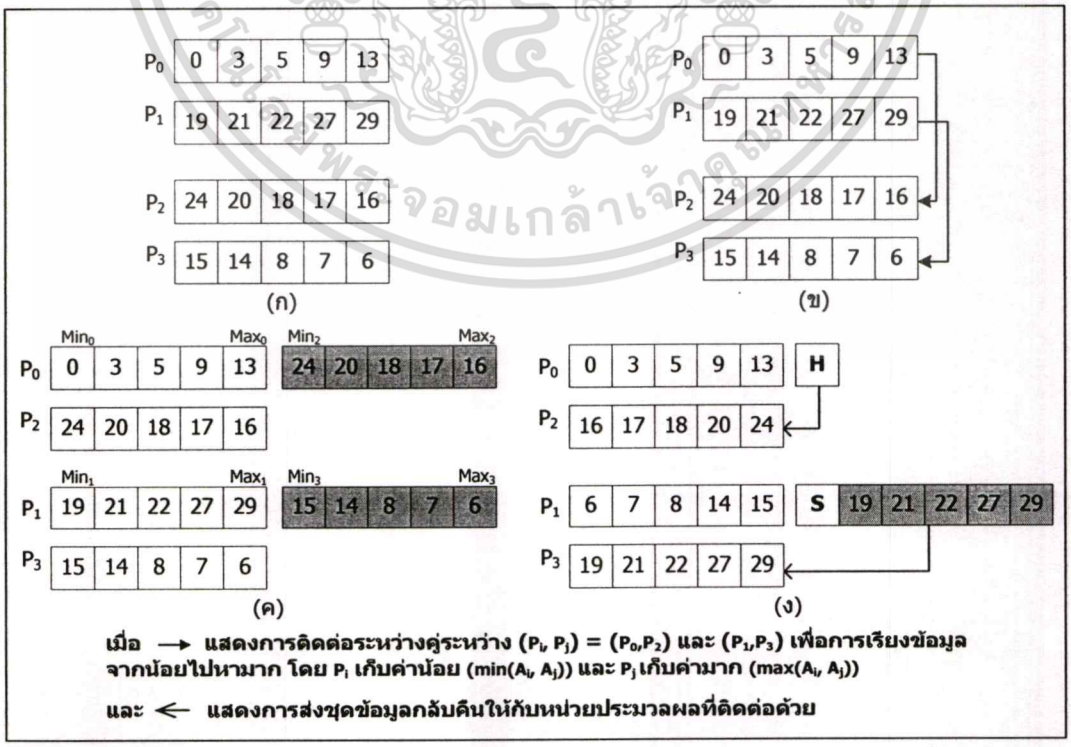
ทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน ซึ่งหน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลมากกว่าจะส่งข้อมูลไปให้กับหน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลน้อยกว่า เช่น คู่ (P_0, P_1) หน่วยประมวลผล P_1 จะส่งข้อมูลไปให้กับหน่วยประมวลผล P_0 ในรูปที่ 3.11(ค) ทำการรวมชุดข้อมูลทั้ง 2 ชุดเข้าด้วยกันโดยใช้หลักการตรวจสอบชุดข้อมูลก่อนการรวมเช่นเดียวกับแบบซีอีบีเอส (CEBS) โดยที่ คู่ (P_0, P_1) เป็นการรวมข้อมูลแบบสลับที่ (Swap Pattern) โดยนำค่า Max_1 มาตรวจสอบกับค่า Min_0 โดยที่ค่า Max_1 ต้องมีค่าน้อยกว่าค่า Min_0 ส่วนคู่ (P_2, P_3) เป็นการรวมข้อมูลแบบเฉพาะส่วน (Partial Pattern) โดยนำค่า Max_3 มาตรวจสอบค่าโดยเริ่มจากค่า Min_2 ไปยังค่า Max_2 โดยที่ค่า Max_3 ต้องมีค่าน้อยกว่าค่าใน P_2 ดังนั้นค่าที่อยู่ในกรอบเข้มคือ ข้อมูลที่ใช้ในการรวมข้อมูลเข้าด้วยกัน และรูปที่ 3.11 (ง) ข้อมูลในกรอบสีเทาจะถูกส่งคืนหลังการเปรียบเทียบค่าให้กับหน่วยประมวลผลที่ติดต่อ (ส่งคืนให้กับหน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลมากกว่า) พร้อมกับค่าที่ระบุว่าเป็น S (Swap Pattern) และ P (Partial Pattern)



รูปที่ 3.11 การแปลงข้อมูลทั่วไปให้อยู่ในรูปของ "Bitonic Sequence" รอบที่ 1

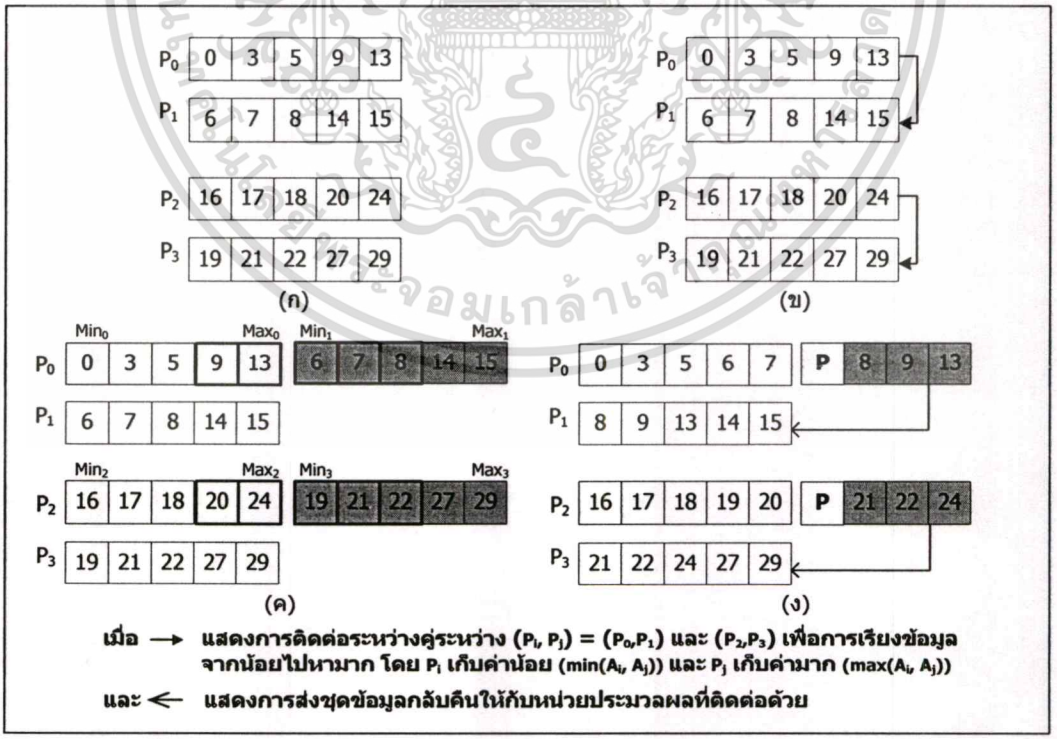
ขั้นที่สอง การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค สำหรับข้อมูลที่ได้รับการแปลงจากขั้นแรก 0, 3, 5, 9, 13, 19, 21, 22, 27, 29, 24, 20, 18, 17, 16, 15, 14, 8, 7, 6 ซึ่งอยู่ในรูปของ "Bitonic Sequence" สามารถคำนวณหาจำนวนรอบการทำงานได้จากนิยามที่ 2 คือ ($\log_2 P$) = ($\log_2 4$) = 2 รอบ

รอบที่ 2 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค ดังรูปที่ 3.12 (ก) แสดงข้อมูลในรอบที่ 1 ขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_0 เก็บค่า (0, 3, 5, 9, 13) P_1 เก็บค่า (19, 21, 22, 27, 29) P_2 เก็บค่า (24, 20, 18, 17, 16) และ P_3 เก็บค่า (15, 14, 8, 7, 6) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) รูปที่ 3.12 (ข) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล (P_0,P_1,..., P_3 : P=4) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน หน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลมากกว่าจะส่งข้อมูลไปให้กับหน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลน้อยกว่า เช่น คู่ (P_0, P_2) หน่วยประมวลผล P_2 จะส่งข้อมูลไปให้กับหน่วยประมวลผล P_0 ในรูปที่ 3.12 (ค) ทำการรวมชุดข้อมูลทั้ง 2 ชุดเข้าด้วยกันโดยใช้หลักการตรวจสอบชุดข้อมูลก่อนการรวมเช่นเดียวกับแบบซีอีบีเอส (CEBS) โดยคู่ (P_0, P_2) เป็นการรวมข้อมูลแบบคงที่ (Hold Pattern) โดยนำค่า Min_2 ที่รับมาจาก P_2 มีค่ามากกว่าหรือเท่ากับ Max_0 ใน P_0 (หรือทุกๆ ค่าใน P_0 นั้นเอง) ในกรณีนี้ไม่ต้องส่งข้อมูลขนาด N/P กลับไปยัง P_0 เพียงแต่ส่งผล H เพื่อแสดงว่าเป็นแบบคงที่เท่านั้น ส่วนคู่ (P_1, P_3) เป็นการรวมข้อมูลแบบสลับที่ (Swap Pattern) นำค่า Max_3 ที่รับมาจาก P_3 มีค่าน้อยกว่าหรือเท่ากับ Min_1 ใน P_1 (หรือทุกๆ ค่าใน P_1 นั้นเอง) ดังนั้นในกรณีนี้ไม่ต้องทำการรวมข้อมูล เพียงแต่รับข้อมูลมาเก็บแทนที่ข้อมูลเดิม และส่งข้อมูลเดิมไปยัง P_3 และรูปที่ 3.12 (ง) ข้อมูลในกรอบสีเทา (ถ้ามี) จะถูกส่งคืนหลังการเปรียบเทียบค่าให้กับหน่วยประมวลผลที่ติดต่อ (ส่งคืนให้กับหน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลมากกว่า) พร้อมกับค่าที่ระบุว่าเป็น H (Hold Pattern) และ S (Swap Pattern)



รูปที่ 3.12 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบ "SEBS" รอบที่ 2

รอบที่ 3 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค ดังรูปที่ 3.13 (ก) แสดงข้อมูลในรอบที่ 2 ขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_0 เก็บค่า (0, 3, 5, 9, 13) P_1 เก็บค่า (6, 7, 8, 14, 15) P_2 เก็บค่า (16, 17, 18, 20, 24) และ P_3 เก็บค่า (19, 21, 22, 27, 29) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) รูปที่ 3.13 (ข) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล (P_0,P_1,..., P_3 : P=4) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน โดยหน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลมากกว่าจะส่งข้อมูลไปให้กับหน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลน้อยกว่า เช่น คู่ (P_0, P_1) หน่วยประมวลผล P_1 จะส่งข้อมูลไปให้กับหน่วยประมวลผล P_0 ในรูปที่ 3.13 (ค) ทำการรวมชุดข้อมูลทั้ง 2 ชุดเข้าด้วยกันโดยใช้หลักการตรวจสอบชุดข้อมูลก่อนการรวมเช่นเดียวกับแบบซีอีบีเอส (CEBS) โดยคู่ (P_0, P_1) เป็นการรวมข้อมูลแบบเฉพาะส่วน (Partial Pattern) โดยนำค่า Max_0 มาตรวจสอบค่าโดยเริ่มจากค่า Min_1 ไปยังค่า Max_1 โดยที่ค่า Max_0 ต้องมีค่าน้อยกว่าค่าที่รับมาจากค่าใน P_1 ดังนั้นค่าที่อยู่ในกรอบเข้มคือ ข้อมูลที่ใช้ในการเรียงลำดับแบบรวมเข้าด้วยกัน ส่วนคู่ (P_2, P_3) เป็นการรวมข้อมูลแบบเฉพาะส่วน (Partial Pattern) ซึ่งมีขั้นตอนการตรวจสอบเช่นเดียวกับคู่ (P_0, P_1) และรูปที่ 3.13 (ง) ข้อมูลในกรอบสีเทาจะถูกส่งคืนหลังการเปรียบเทียบค่าให้กับหน่วยประมวลผลที่ติดต่อ (ส่งคืนให้กับหน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลมากกว่า) พร้อมกับค่าที่ระบุว่าเป็น P (Partial Pattern)



รูปที่ 3.13 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบ "SEBS" รอบที่ 3

ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี (MPI-C like) การเรียงลำดับด้วยวิธีไบโทนิคแบบ “SEBS”

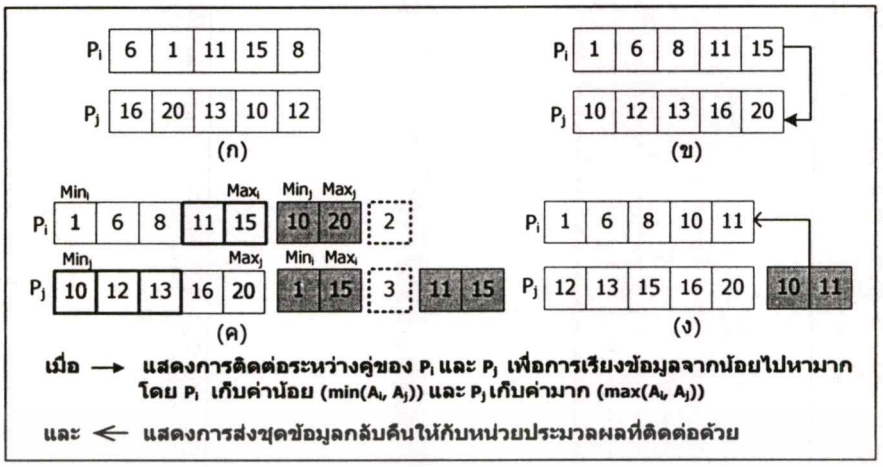
```
Start
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &p);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

Generate Data
  srand(my_rank);
  for (i = 0; i < list_size; i++)  local_list[i] = rand() % KEY_MAX;

Local Sort
  qsort(local_keys, list_size, sizeof(KEY_T),
        int(*)(const void*, const void*))(Key_compare));

Bitonic Sort(BS)
  if ((myrank & and_bit) == 0) // increase
      Bitonic_increasing(list_size, num, proc_set_size); // increasing
  else // decrease
      Bitonic_decreasing (list_size, num, proc_set_size); // decreasing

Partner Communication (send and receive data)
  MPI_Comm_rank(comm, &myrank);
    proc_set_dim = log_base2(proc_set_size);
    eor_bit = 1 << (proc_set_dim - 1);
    for (stage = 0; stage < proc_set_dim; stage++) {
      partner = my_rank ^ eor_bit;
      if (myrank > partner) // OR myrank > partner for Bitonic_decreasing
        MPI_Send(&num[0], list_size, MPI_INT, partner, data_in, MPI_COMM_WORLD);
        MPI_Recv(&num[0], list_size, MPI_INT, partner, result, MPI_COMM_WORLD, &status);
      }
      else {
        MPI_Recv(&num[list_size], list_size, MPI_INT, partner, data_in, MPI_COMM_WORLD,&status);
        qsort(num, list_size*2, sizeof(int), // local quick sort(int(*)
            (const void*, const void*))(Key_compare));
        MPI_Send(&num[list_size], list_size, MPI_INT, partner, result, MPI_COMM_WORLD);
      }
      eor_bit = eor_bit >> 1; }
Check_pattern
  if (value ==0)
    if (min_p >= num[last])
      printf("STAGE = %d Hold Pattern partner %d of myrank %d\n",stage ,partner,myrank);
    else if (max_p <= num[0])
      Merge_list_swap(list_size, num, partner);
    else
      Merge_list_low(list_size, num, partner,min_p, myrank);
  else
    if(max_p <= num[0])
      printf("STAGE = %d Hold Pattern partner %d of myrank %d\n",stage ,partner,myrank);
    else if (min_p >= num[last])
      Merge_list_swap(list_size, num, partner);
    else
      Merge_list_high(list_size, num, partner,max_p, myrank);
```

รูปที่ 3.14 ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี การเรียงลำดับแบบขนานด้วยวิธี “SEBS”

3.4 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบ "Communication-Space Efficient Bitonic Sort : CSEBS"

วิธีการเรียงลำดับแบบซีเอสอีบีเอส (CSEBS) นี้เป็นวิธีการใหม่ที่เสนอในวิทยานิพนธ์นี้ โดยวิธีการนี้จะประยุกต์ข้อดีของวิธีการที่ได้กล่าวมาข้างต้นคือแบบเอสอีบีเอส (SEBS) ที่ช่วยเพิ่มประสิทธิภาพทั้งการลดการใช้พื้นที่ในหน่วยความจำที่เก็บข้อมูลซ้ำซ้อนในระบบ ซึ่งสามารถลดได้ประมาณ 50% และแบบซีอีบีเอส (CEBS) ที่ช่วยลดเวลาที่ใช้ในการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลซึ่งสามารถลดได้อย่างน้อย 20% นอกจากนี้แล้ววิธีนี้ยังได้เพิ่มประสิทธิภาพของการแลกเปลี่ยนข้อมูลแบบเฉพาะส่วน (Efficient Partial Pattern) โดยกำหนดให้หน่วยประมวลผลใดก็ได้ระหว่าง P_i และ P_j ที่มีจำนวนข้อมูลที่ต้องส่ง (Partial Data) เป็นจำนวนน้อยกว่าหน่วยประมวลผลที่ติดต่อด้วยเป็นผู้ส่งข้อมูล ซึ่งต่างจากแบบเอสอีบีเอส (SEBS) ที่กำหนดให้ P_j (หน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลมากกว่า) เท่านั้น เป็นผู้ส่งข้อมูลไปยัง P_i (หน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลน้อยกว่า) ดังนั้นวิธีที่ได้นำเสนอนี้สามารถลดการใช้พื้นที่ในหน่วยความจำที่เก็บข้อมูลที่ซ้ำซ้อนในระบบได้มากกว่า 50% (ซึ่งมากกว่าแบบเอสอีบีเอส (SEBS)) และลดเวลาที่ใช้ในการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลได้มากกว่า 20% (ซึ่งมากกว่าแบบซีอีบีเอส (CEBS)) ดังรูปที่ 3.15 โดยที่ (ก) แสดงการโหลดข้อมูลข้อมูลขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_i เก็บค่า (6, 1, 11, 15, 8) และ P_j เก็บค่า (16, 20, 13,10, 12) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) และทำการเรียงลำดับข้อมูลขนาด N/P ด้วยวิธีแบบเร็ว (Quick Sort) ในทุกหน่วยประมวลผลพร้อมๆ กันรูปที่ 3.15 (ข) แสดงข้อมูลที่เรียงแล้วและแสดงการติดต่อสื่อสารระหว่าง P_i กับ P_j รูปที่ 3.15 (ค) แสดงการส่งค่าน้อย (Min) และค่ามาก (Max) ไปให้กับหน่วยประมวลผลที่ติดต่อด้วย เพื่อตรวจสอบหาชุดข้อมูลที่ต้องการแลกเปลี่ยนเท่านั้น จากนั้นแสดงการตรวจสอบเพื่อหาข้อมูลขนาดเล็กที่สุดที่ต้องส่ง โดยการส่งและรับค่าจำนวนข้อมูลที่ต้องส่งระหว่าง P_i และ P_j (เช่น ค่า 2 สำหรับ P_i และ 3 สำหรับ P_j ในกรอบเส้นประ) ซึ่งในกรณีนี้ P_i มีจำนวนชุดข้อมูลที่ต้องส่งน้อยกว่า P_j (แสดงในกรอบเข้ม) ดังนั้น P_i จึงส่งข้อมูลบางส่วน (11, 15) ไปให้กับ P_j เพื่อทำการเรียงลำดับข้อมูลแบบรวมเข้าด้วยกัน รูปที่ 3.15 (ง) นำค่าใน P_j ทำการเปรียบเทียบค่าที่รับมากับค่าที่มีอยู่ ซึ่ง P_j จะเก็บค่ามาก (max(Ai, Aj)) ไว้ และค่าน้อย (min(Ai, Aj)) จะถูกส่งกลับคืนให้กับ P_i

ส่วนการแลกเปลี่ยนข้อมูลแบบคงที่ (Hold Pattern) และการแลกเปลี่ยนข้อมูลแบบสลับที่ (Swap Pattern) ยังคงเป็นแบบเดียวกับแบบซีอีบีเอส (CEBS) ในรูปที่ 3.3 และ รูปที่ 3.4 ตามลำดับ



รูปที่ 3.15 แสดงวิธีการแบบ "Efficient Partial Pattern"

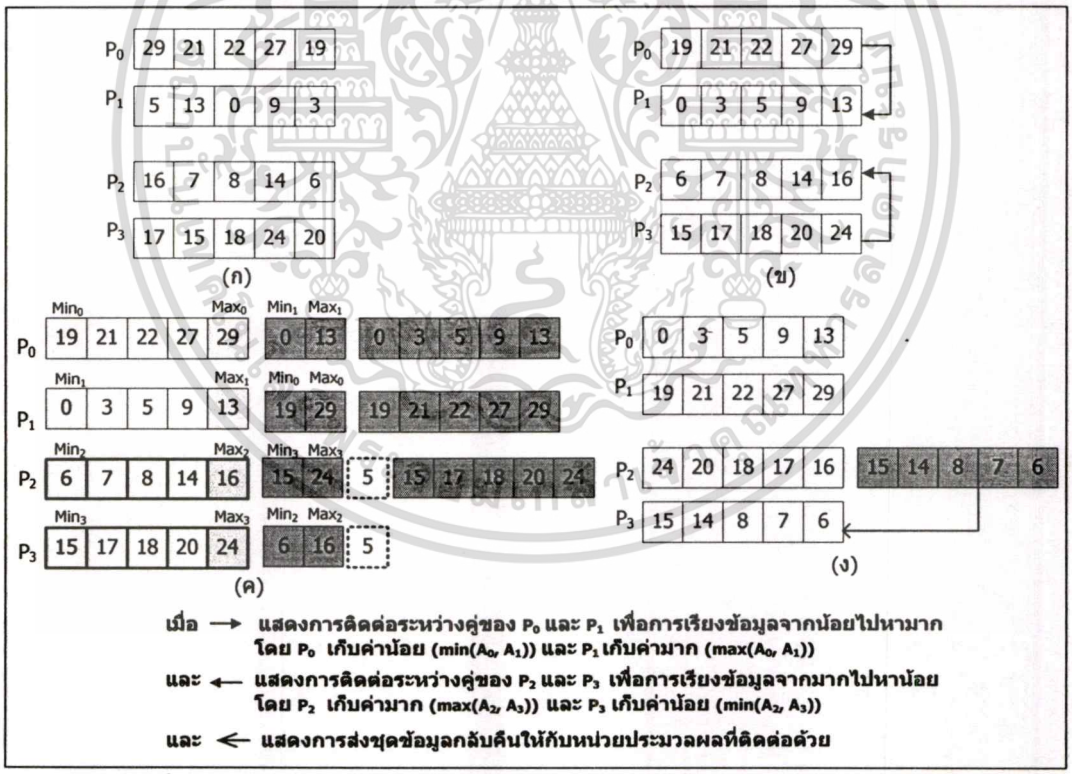
ตัวอย่างที่ 3.3 กรณีที่ข้อมูลนำเข้าอยู่ในรูปทั่วไป (ไม่อยู่ในรูปของ "Bitonic Sequence")

สมมติข้อมูลนำเข้า (N=20) คือ 29, 21, 22, 27, 19, 5, 13, 0, 9, 3, 16, 7, 8, 14, 6, 17, 15, 18, 24, 20 ซึ่งจำนวนหน่วยประมวลผล (P=4) น้อยกว่าจำนวนข้อมูล (P<N) และข้อมูลยังไม่ได้ถูกเรียงลำดับให้อยู่ในรูปของ "Bitonic Sequence" ดังนั้นการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคจึงจำเป็นต้องใช้ 2 ขั้นตอนในการเรียงลำดับข้อมูล ตามนิยามที่ 3 ซึ่งสามารถคำนวณหาจำนวนรอบในการเรียงลำดับข้อมูลได้ดังนี้ $(1+\log_2 P)(\log_2 P)/2 = (1+\log_2 4)(\log_2 4)/2 = 3$ รอบ โดยมีรายละเอียดดังต่อไปนี้คือ

ขั้นแรก การแปลงข้อมูลทั่วไปให้อยู่ในรูปของ "Bitonic Sequence" ซึ่งสามารถคำนวณหาจำนวนรอบการทำงานได้ โดยนำจำนวนรอบในขั้นที่สอง (log2 P) มาหักออกจากจำนวนทั้งหมดซึ่งเท่ากับ $(1+\log_2 P)(\log_2 P)/2 - (\log_2 P) = 3 - 2 = 1$ รอบ

รอบที่ 1 การแปลงข้อมูลจากข้อมูลทั่วไปให้อยู่ในรูปแบบของ "Bitonic Sequence" ดังรูปที่ 3.16 (ก) แสดงการโหลดข้อมูลขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_0 เก็บค่า (29, 21, 22, 27, 19) P_1 เก็บค่า (5, 13, 0, 9, 3) P_2 เก็บค่า (16, 7, 8, 14, 6) และ P_3 เก็บค่า (17, 15, 18, 24, 20) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) และทำการเรียงลำดับข้อมูลขนาด N/P ด้วยวิธีแบบเร็ว (Quick Sort) ในทุกหน่วยประมวลผลพร้อมๆ กัน รูปที่ 3.16 (ข) แสดงข้อมูลที่เรียงแล้วและแสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล ($P_0, P_1, \ldots, P_3$: P=4) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน รูปที่ 3.16 (ค) แสดงการส่งค่าน้อย (Min) และค่ามาก (Max) ไปให้กับหน่วยประมวลผลที่ติดต่อด้วย เพื่อตรวจสอบหารูปแบบของการส่งข้อมูลดังที่ได้กล่าวมาแล้วข้างต้น (Swap Pattern, Hold Pattern, Partial Pattern) เช่น คู่ (P_0, P_1) เป็นการแลกเปลี่ยนชุดข้อมูลแบบสลับที่ (Swap Pattern) โดยนำค่า Min_0 ใน P_0 มีค่า

มากกว่าหรือเท่ากับ Max_1 ที่รับมาจาก P_1 (หรือทุกๆ ค่าใน P_1 นั้นเอง) และ Max_1 ใน P_1 มีค่าน้อยกว่าหรือเท่ากับ Min_0 ที่รับมาจาก P_0 (หรือทุกๆ ค่าใน P_0 นั้นเอง) ในกรณีนี้จะทำการส่งและรับข้อมูลระหว่างหน่วยประมวลผล (P_0, P_1) เพื่อเก็บแทนข้อมูลที่มีอยู่เดิม ส่วนคู่ (P_2, P_3) เป็นการแลกเปลี่ยนข้อมูลแบบเฉพาะส่วน (Partial Pattern) โดยนำค่า Max_3 มาตรวจสอบค่าโดยเริ่มจากค่า Min_2 ไปยังค่า Max_2 โดยที่ค่า Max_3 ต้องมีค่าน้อยกว่าค่าใน P_2 ดังนั้นค่าที่อยู่ในกรอบเข้มคือข้อมูลที่ต้องส่ง (ซึ่งมีจำนวนข้อมูลที่ต้องส่งเท่ากับ 5) และในขณะเดียวกันนำค่า Min_2 มาตรวจสอบค่าโดยเริ่มจากค่า Max_3 ไปยังค่า Min_3 โดยที่ค่า Min_2 ต้องมีค่ามากกว่าค่าใน P_3 ดังนั้นค่าที่อยู่ในกรอบเข้มคือข้อมูลที่ต้องส่ง (ซึ่งมีจำนวนข้อมูลที่ต้องส่งเท่ากับ 5) จากนั้นทำการส่งและรับค่าของจำนวนข้อมูลที่ต้องส่งที่คำนวณได้ใน P_2 และ P_3 (แสดงในกรอบเส้นประ) มาตรวจสอบ เพื่อหาว่าใครเป็นผู้ส่งข้อมูล ในกรณีนี้ P_2 และ P_3 มีจำนวนข้อมูลที่ต้องส่งเท่ากัน ดังนั้นยังคงใช้เงื่อนไขเดียวกับแบบเอสอีบีเอสที่กำหนดให้หน่วยประมวลผลที่มีหมายเลขประจำหน่วยประมวลผลมากกว่าเป็นผู้ส่งข้อมูล รูปที่ 3.16 (ง) เปรียบเทียบค่าที่รับมากับค่าที่มีอยู่ และแสดงผลการเรียงลำดับข้อมูลที่เหมาะสม



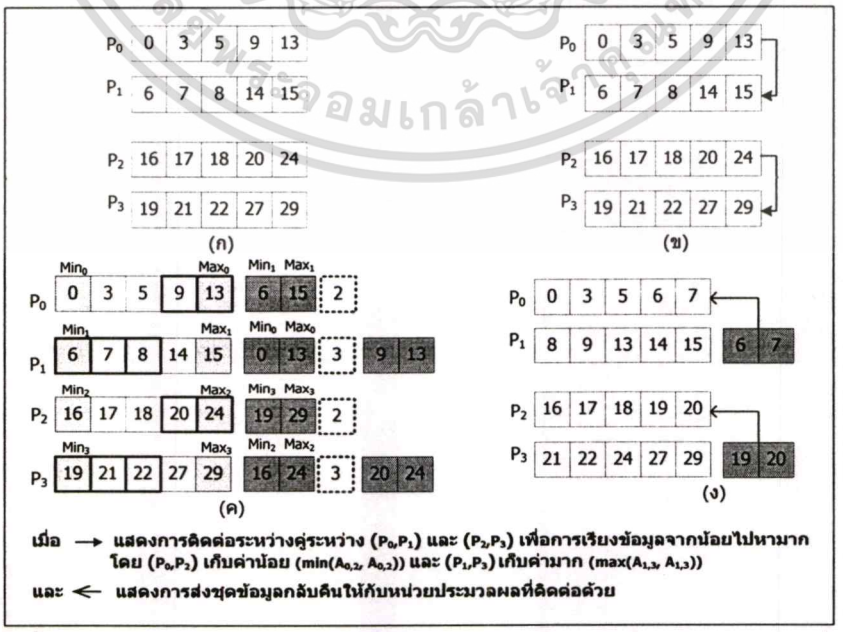
รูปที่ 3.16 การแปลงข้อมูลทั่วไปให้อยู่ในรูป "Bitonic Sequence" รอบที่ 1

ขั้นที่สอง การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค สำหรับข้อมูลที่ได้รับการแปลงจากขั้นแรก 0, 3, 5, 9, 13, 19, 21, 22, 27, 29, 24, 20, 18, 17, 16, 15, 14, 8, 7, 6 ซึ่งอยู่ในรูปของ "Bitonic Sequence" สามารถคำนวณหาจำนวนรอบการทำงานได้จากนิยามที่ 2 คือ (log P) = (log 4) = 2 รอบ

รอบที่ 2 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค ดังรูปที่ 3.17 (ก) แสดงข้อมูลในรอบที่ 1 ขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_0 เก็บค่า (0, 3, 5, 9, 13) P_1 เก็บค่า (19, 21, 22, 27, 29) P_2 เก็บค่า (24, 20, 18, 17, 16) และ P_3 เก็บค่า (15, 14, 8, 7, 6) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) รูปที่ 3.17 (ข) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล (P_0,P_1,..., P_3 : P=4) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน รูปที่ 3.17 (ค) แสดงการส่งค่าน้อย (Min) และค่ามาก (Max) ไปให้กับหน่วยประมวลผลที่ติดต่อด้วย เพื่อตรวจสอบหารูปแบบของการส่งข้อมูลดังที่ได้กล่าวมาแล้วข้างต้น (Swap Pattern, Hold Pattern, Partial Pattern) เช่น คู่ (P_0, P_2) เป็นการแลกเปลี่ยนชุดข้อมูลแบบคงที่ (Hold Pattern) โดยนำค่า Min_2 ที่รับมาจาก P_2 มีค่ามากกว่าหรือเท่ากับ Max_0 ใน P_0 (หรือทุกๆ ค่าใน P_0 นั้นเอง) และ Max_0 ที่รับมาจาก P_0 มีค่าน้อยกว่าหรือเท่ากับ Min_2ใน P_2 (หรือทุกๆ ค่าใน P_2 นั้นเอง) ในกรณีนี้ไม่ต้องมีการแลกเปลี่ยนชุดข้อมูลขนาด N/P ข้อมูลระหว่าง P_0 และ P_2 ส่วนคู่ (P_1, P_3) เป็นการแลกเปลี่ยนข้อมูลแบบสลับที่ (Swap Pattern) นำค่า Min_1 ใน P_1 มีค่ามากกว่าหรือเท่ากับ Max_3 ที่รับมาจาก P_3 (หรือทุกๆ ค่าใน P_3 นั้นเอง) และ Max_3 ใน P_3 มีค่าน้อยกว่าหรือเท่ากับ Min_1 ที่รับมาจาก P_1 (หรือทุกๆ ค่าใน P_1 นั้นเอง) ดังนั้นในกรณีนี้จะทำการส่งและรับข้อมูลระหว่างหน่วยประมวลผล (P_1, P_3) เพื่อเก็บแทนข้อมูลที่มีอยู่เดิม รูปที่ 3.17 (ง) เปรียบเทียบค่าที่รับมากับค่าที่มีอยู่ และแสดงผลการเรียงลำดับข้อมูลที่เหมาะสม

รูปที่ 3.17 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบ "CSEBS" รอบที่ 2

รอบที่ 3 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิค ดังรูปที่ 3.18 (ก) แสดงข้อมูลในรอบที่ 2 ขนาด N/P เมื่อแต่ละหน่วยประมวลผล P_0 เก็บค่า (0, 3, 5, 9, 13) P_1 เก็บค่า (6, 7, 8, 14, 15) P_2 เก็บค่า (16, 17, 18, 20, 24) และ P_3 เก็บค่า (19, 21, 22, 27, 29) ไว้ในหน่วยความจำส่วนตัว (Local Memory) ของแต่ละหน่วยประมวลผล (P_i) รูปที่ 3.18 (ข) แสดงการติดต่อสื่อสารระหว่างหน่วยประมวลผล (P_0,P_1,..., P_3 : P=4) โดยทุกๆ คู่ของหน่วยประมวลผล (P_i, P_j) จะแลกเปลี่ยนข้อมูลไปพร้อมๆ กัน รูปที่ 3.18 (ค) แสดงการส่งค่าน้อย (Min) และในขณะเดียวกันค่ามาก (Max) ไปให้กับหน่วยประมวลผลที่ติดต่อด้วย เพื่อตรวจสอบหารูปแบบของการส่งข้อมูลดังที่ได้กล่าวมาแล้วข้างต้น เช่น คู่ (P_0, P_1) เป็นการแลกเปลี่ยนข้อมูลแบบเฉพาะส่วน (Partial Pattern) โดยนำค่า Max_0 มาตรวจสอบค่าโดยเริ่มจากค่า Min_1 ไปยังค่า Max_1 โดยที่ค่า Max_0 ต้องมีค่าน้อยกว่าค่าใน P_1 ดังนั้นค่าที่อยู่ในกรอบเข้มคือข้อมูลที่ต้องส่ง (ซึ่งมีจำนวนข้อมูลที่ต้องส่งเท่ากับ 3) และในขณะเดียวกันนำค่า Min_1 มาตรวจสอบค่าโดยเริ่มจากค่า Max_0 ไปยังค่า Min_0 โดยที่ค่า Min_1 ต้องมีค่ามากกว่าค่าใน P_0 ดังนั้นค่าที่อยู่ในกรอบเข้มคือข้อมูลที่ต้องส่ง (ซึ่งมีจำนวนข้อมูลที่ต้องส่งเท่ากับ 2) จากนั้นทำการส่งและรับค่าของจำนวนข้อมูลที่ต้องส่งที่คำนวณได้ใน P_0 และ P_1 (แสดงในกรอบเส้นประ) มาตรวจสอบ เพื่อหาว่าใครเป็นผู้ส่งข้อมูล ในกรณีนี้ P_0 มีจำนวนข้อมูลที่ต้องส่งน้อยกว่า P_1 ดังนั้น P_0 จะทำการส่งข้อมูลบางส่วน (9, 13) ไปให้กับ P_1 เพื่อทำการเรียงลำดับข้อมูลแบบรวมเข้าด้วยกัน ส่วนคู่ (P_2, P_3) เป็นการแลกเปลี่ยนข้อมูลแบบเฉพาะส่วน (Partial Pattern) เช่นกัน ซึ่งมีขั้นตอนการตรวจสอบเช่นเดียวกับคู่ (P_0, P_1) โดยข้อมูลที่ต้องส่งใน P_2 เท่ากับ 2 (ค่าที่อยู่ในกรอบเข้ม 20, 24) และข้อมูลที่ต้องส่งใน P_3 เท่ากับ 3 (ค่าที่อยู่ในกรอบเข้ม 19, 21, 22) ดังนั้น P_2 จะเป็นผู้ส่งข้อมูลให้กับ P_3 เพื่อทำการเรียงลำดับข้อมูล รูปที่ 3.18 (ง) เปรียบเทียบค่าที่รับมากับค่าที่มีอยู่ และแสดงผลการเรียงลำดับข้อมูลที่เหมาะสม



รูปที่ 3.18 การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบ "CSEBS" รอบที่ 3

ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี (MPI-C like) การเรียงลำดับด้วยวิธีไบโทนิคแบบ "CSEBS"

```
Start
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &p);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

Generate Data
 srand(my_rank);
 for (i = 0; i < list_size; i++)  local_list[i] = rand() % KEY_MAX;

Local Sort
 qsort(local_keys, list_size, sizeof(KEY_T),
       int(*)(const void*, const void*))(Key_compare));

Bitonic Sort(BS)
 if ((myrank & and_bit) == 0) // increase
    Bitonic_increasing(list_size, num, proc_set_size); // increasing
 else // decrease
    Bitonic_decreasing (list_size, num, proc_set_size); // decreasing

Partner Communication (send and receive data)
 MPI_Comm_rank(comm, &myrank);
   proc_set_dim = log_base2(proc_set_size);
   eor_bit = 1 << (proc_set_dim - 1);
   for (stage = 0; stage < proc_set_dim; stage++) {
       partner = my_rank ^ eor_bit;
       if (myrank < partner) // OR myrank > partner for Bitonic_decreasing
          Check_pattern(list_size, num, myrank, partner,0,stage);
       }
       else {
          Check_pattern(list_size, num, myrank, partner,1,stage);
       }
       eor_bit = eor_bit >> 1;
   }
Check_pattern
 if (value ==0)
   if (min_p >= num[last])
       printf("STAGE = %d Hold Pattern partner %d of myrank %d\n",stage ,partner,myrank);
   else if (max_p <= num[0])
       Merge_list_swap(list_size, num, partner);
   else
       Merge_list_low(list_size, num, partner,min_p, myrank);
 else
   if(max_p <= num[0])
       printf("STAGE = %d Hold Pattern partner %d of myrank %d\n",stage ,partner,myrank);
   else if (min_p >= num[last])
       Merge_list_swap(list_size, num, partner);
   else
       Merge_list_high(list_size, num, partner,max_p, myrank);

Merge_list_high (Pattern Send and Receive data)
 MPI_Send(&partial_high, 1, MPI_INT, partner, 1, MPI_COMM_WORLD);
   MPI_Recv(&temp_low, 1, MPI_INT, partner, 0, MPI_COMM_WORLD, &status);
   if (partial_high <= temp_low  ){ //for Merge low if(partial_low < temp_high)
       MPI_Send(&temp[0], partial_high, MPI_INT, partner, 4, MPI_COMM_WORLD);
       MPI_Recv(&num[0], list_size, MPI_INT, partner, 5, MPI_COMM_WORLD, &status);
   }
   else {
       MPI_Recv(&temp[0], list_size, MPI_INT, partner, 3, MPI_COMM_WORLD, &status);
       Local_sort_high(list_size, num, temp, partial_high, partner, myrank); // Local_sort_low();
       MPI_Send(&merge[list_size], partial_high, MPI_INT, partner, 6, MPI_COMM_WORLD);

   }
```

รูปที่ 3.19 ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคแบบ"CSEBS"

```
Merge_list_low (Pattern Send and Receive data)
 MPI_Send(&partial_low, 1, MPI_INT, partner, 0, MPI_COMM_WORLD);
    MPI_Recv(&temp_high, 1, MPI_INT, partner, 1, MPI_COMM_WORLD, &status);
    partial = list_size - temp_high;
    if (partial_low < temp_high ){
        MPI_Send(&temp[0], partial_low, MPI_INT, partner, 3, MPI_COMM_WORLD);
        MPI_Recv(&num[partial], list_size, MPI_INT, partner, 6, MPI_COMM_WORLD, &status);
    }
    else{
        MPI_Recv(&temp[0], list_size, MPI_INT, partner, 4, MPI_COMM_WORLD, &status);
        Local_sort_low(list_size, num, temp, partial_low, partner, myrank);
        MPI_Send(&merge[list_size], partial_low, MPI_INT, partner, 5, MPI_COMM_WORLD);
    }
```

รูปที่ 3.19 (ต่อ)

บทที่ 4

การทดลองและผลการทดลอง

การประมวลผลแบบขนาน (Parallel Processing) เป็นการประมวลผลสำหรับงานที่มีขนาดใหญ่โดยใช้หลายหน่วยประมวลผล ปกติการประมวลผลแบบขนานจะมีความซับซ้อนกว่าการประมวลผลแบบอนุกรม (Sequential Processing) เนื่องจากเวลาที่ใช้ในการประมวลผลแบบขนานทั้งหมด จะประกอบด้วยเวลาที่ใช้ในการประมวลผล (Computation Time) โดยใช้ P หน่วยประมวลผล และเวลาที่ใช้ในการติดต่อสื่อสาร (Communication Time) ระหว่างหน่วยประมวลผล ส่วนเวลาทั้งหมดที่ใช้ในการประมวลผลแบบอนุกรมจะเป็นเวลาที่ใช้ในการประมวลผลโดยใช้หนึ่งหน่วยประมวลผลเท่านั้น ถึงแม้ว่าการประมวลผลแบบขนานจะมีความซับซ้อนกว่าการประมวลผลทั่วๆไปที่เป็นแบบอนุกรม แต่การประมวลผลแบบขนานมีบทบาทสำคัญต่อการประมวลผลงานขนาดใหญ่ที่ใช้เวลานาน ดังนั้นวิธีการประมวลผลแบบขนานที่มีประสิทธิภาพจึงถูกเสนอขึ้น ทั้งขั้นตอนวิธีแบบขนานและการประยุกต์ใช้งานในหลายๆ ด้านแบบขนาน เช่น การเรียงลำดับแบบขนาน การคูณเมตริกซ์แบบขนาน เป็นต้น

งานวิจัยนี้เน้นเรื่องขั้นตอนวิธีการเรียงลำดับแบบขนาน ดังนั้นจึงขอกล่าวถึงการออกแบบและการเปรียบเทียบประสิทธิภาพของระบบ (System Performance) ในการเรียงลำดับแบบขนาน ซึ่งสามารถจำแนกได้เป็น 3 กลุ่มกว้างๆ คือ

1) การออกแบบระบบคอมพิวเตอร์แบบขนานเพื่อรองรับการเรียงลำดับข้อมูลแบบขนานโดยเฉพาะ (Specific Parallel Machine) เช่น Sorting Network และการออกแบบขั้นตอนวิธีสำหรับระบบที่มีรูปแบบการติดต่อสื่อสารเฉพาะ เช่น Hypercubes และ Meshes ซึ่งโดยปกติจะสมมติว่า มีจำนวนหน่วยประมวลผล (P) เท่ากับจำนวนข้อมูล (N) หรือ P = N ดังนั้นการวัดประสิทธิภาพของระบบคอมพิวเตอร์แบบขนานดังกล่าว ส่วนใหญ่จะเป็นการเปรียบเทียบฟังก์ชันของจำนวนหน่วยประมวล (P) และจำนวนเส้นที่เชื่อมต่อ (Links) ระหว่างหน่วยประมวลผลทั้งหมดในระบบ ซึ่งฟังก์ชันของค่าดังกล่าวสามารถบอกฟังก์ชันของราคาในการสร้างระบบ โดยที่ต้องการให้มีค่าน้อยๆ ซึ่งต้องแลกกับ (Tradeoff) เวลาประมวลผลที่ต้องการให้ลดลง หรือมีประสิทธิภาพมากขึ้นนั่นเอง ส่วนการออกแบบขั้นตอนวิธีสำหรับระบบที่มีรูปแบบการติดต่อสื่อสารเฉพาะ จะใช้การเปรียบเทียบความซับซ้อนด้านเวลา (Time Complexity) เป็นเกณฑ์

2) การออกแบบขั้นตอนวิธีการเรียงลำดับแบบขนานบนระบบคอมพิวเตอร์แบบขนานที่มีอยู่หรือถูกสร้างขึ้นมาแล้วและสามารถประมวลผล หรือทำการทดลองได้จริง เช่น ระบบคลัสเตอร์ (Cluster System) หรือคอมพิวเตอร์แบบกริด (Grid Computer) ระบบคอมพิวเตอร์ Cray T3E ระบบคอมพิวเตอร์ Meiko CS-2 และระบบคอมพิวเตอร์แบบ SMP เป็นต้น ซึ่งปกติจะมีจำนวน

หน่วยประมวลผล (P) น้อยกว่า จำนวนข้อมูล (N) หรือ P < N เนื่องจากระบบดังกล่าวเป็นระบบที่มีใช้อย่างกว้างขวาง ดังนั้นการวัดประสิทธิภาพ ส่วนใหญ่จะวัดจากเวลาที่ใช้ในการประมวลผลจริง (Response Time) บนระบบต่างๆ ดังกล่าว

3) การเสนอขั้นตอนวิธีการเรียงลำดับแบบขนานแบบใหม่ ซึ่งอาจจะเป็นการปรับปรุงจากแบบที่มีอยู่เดิมให้มีประสิทธิภาพมากขึ้น โดยส่วนใหญ่จะเป็นการเพิ่มประสิทธิภาพด้านเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล ดังนั้นกรณีนี้การวัดประสิทธิภาพจะเป็นแบบกรณีที่ 1 ถ้าวิธีการใหม่นั้นเปรียบเทียบกับวิธีต่างๆในกรณีที่ 1 หรือเป็นการวัดประสิทธิภาพแบบกรณีที่ 2 ถ้าวิธีการใหม่นั้นเปรียบเทียบกับวิธีต่างๆในกรณีที่ 2

วิทยานิพนธ์นี้เป็นการนำเสนอการออกแบบขั้นตอนวิธีการเรียงลำดับแบบไบโทนิคซึ่งเป็นการเรียงลำดับแบบขนานชนิดหนึ่งที่ได้รับความนิยมสูง โดยวิธีต่างๆที่เสนอเป็นวิธีใหม่ที่ปรับปรุงประสิทธิภาพทั้งในด้านเวลาที่ใช้ในการติดต่อสื่อสาร และการใช้เนื้อที่ในหน่วยความจำที่ซ้ำซ้อนกันในระบบ ดังแสดงไว้แล้วในบทที่ 3 ส่วนเนื้อหาในบทนี้ จะเป็นการเสนอผลจากการพัฒนาโปรแกรมการเรียงลำดับแบบขนานของขั้นตอนวิธีดังกล่าว เพื่อประมวลผลบนระบบพีซีคลัสเตอร์ ซึ่งเป็นระบบที่ได้รับความสนใจเป็นอย่างมากในปัจจุบัน เนื่องจากมีราคาไม่แพง และมีขั้นตอนการพัฒนาที่ไม่ยุ่งยากซับซ้อนมากจนเกินไป

4.1 เครื่องมือที่ใช้ในการทดลอง

ระบบคลัสเตอร์ที่ใช้ในการทดลอง เป็นระบบคลัสเตอร์แบบ “Homogenous Cluster” ที่มีส่วนประกอบ ประสิทธิภาพในการประมวลผล และภาระงานที่รับผิดชอบในแต่ละเครื่องมีจำนวนเท่ากัน ขณะทำการทดลองระบบคลัสเตอร์จะไม่มีภาระงานอื่นใดประมวลผลอยู่ในระบบนอกจากการประมวลผลการเรียงลำดับข้อมูลแบบขนานเท่านั้น โดยมีส่วนประกอบพื้นฐานของระบบซึ่งจะประกอบด้วยส่วนประกอบด้านฮาร์ดแวร์ ระบบปฏิบัติการและซอฟท์แวร์ ที่ทำงานในระดับของแกน (Kernel) ของระบบปฏิบัติการซึ่งเรียกซอฟท์แวร์นี้ว่า “คลัสเตอร์มิดเดิลแวร์” และสุดท้ายคือการเชื่อมต่อแต่ละโหนดข้อมูลเข้าด้วยกันผ่านทางเครือข่ายความเร็วสูง

1) ส่วนประกอบด้านฮาร์ดแวร์

โครงสร้างทางฮาร์ดแวร์ของระบบที่ใช้ในการทดลอง เป็นคลัสเตอร์ที่มีโครงสร้างเหมือนกัน (Homogenous Cluster) คือหน่วยประมวลผล ขนาดของหน่วยความจำ และส่วนประกอบอื่นๆ เหมือนกัน ระบบที่ใช้ในการทดลองเป็นระบบคลัสเตอร์ที่พัฒนาขึ้นโดยสำนักวิจัยและบริการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง โดยระบบในปัจจุบันนี้ ประกอบด้วยคอมพิวเตอร์ส่วนบุคคลจำนวน 5 เครื่อง โดยใช้เครื่องคอมพิวเตอร์

ชนิดที่มีสองหน่วยประมวลผลในเครื่องเดียวกัน (Dual CPU) ซึ่งมีองค์ประกอบด้านฮาร์ดแวร์ที่จำเป็นในการพิจารณาเพื่อสร้างเป็นระบบคลัสเตอร์ต่างๆ ดังนี้

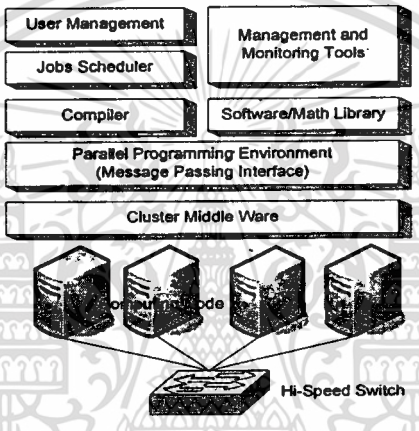
ก) แผงวงจรรวม (Mother Board) หรือ System Board

ข) หน่วยประมวลผล (CPUs) ในงานวิจัยนี้ใช้เครื่องคอมพิวเตอร์ชนิดที่มีสองหน่วยประมวลผลในเครื่องเดียวกัน(Dual CPUs) ยี่ห้ออินเทล (Intel Xeon) ความเร็วต่อรอบ 2.4 กิกะเฮิร์ต

ค) หน่วยความจำหลัก (RAM) ใช้ 1 กิกะไบต์

ง) หน่วยความจำสำรอง (Disk Storage) มีขนาดเท่ากับ 100 กิกะไบต์

จ) เชื่อมต่อผ่านเครือข่ายความเร็ว 100 เมกกะบิตต่อวินาที (Mbps)



รูปที่ 4.1 องค์ประกอบของระบบคลัสเตอร์ [30]

2) ระบบปฏิบัติการ

ระบบปฏิบัติการจะเป็นส่วนสำคัญอย่างมากต่อการทำงานของคอมพิวเตอร์และระบบคลัสเตอร์เพราะคอมพิวเตอร์ในระบบคลัสเตอร์ต้องสามารถทำงานเองได้โดยอิสระไม่ขึ้นกับเงื่อนไขของเครื่องอื่นถึงแม้มีเครื่องใดเครื่องหนึ่งในระบบหยุดทำงาน ระบบคลัสเตอร์ก็ยังสามารถทำงานได้ดังนั้นระบบปฏิบัติการจึงมีความจำเป็นในส่วนที่จะทำให้คอมพิวเตอร์แต่ละเครื่องสามารถทำงานโดยอิสระต่อกันได้ และสามารถติดต่อสื่อสารกันได้ ระบบปฏิบัติการที่สามารถใช้กับระบบคลัสเตอร์แบบพีซี (Cluster of PCs) มีหลากหลาย เช่น Linux, SUSE, FreeBSD, HP-UX และระบบปฏิบัติการที่สามารถใช้กับระบบคลัสเตอร์แบบเวิร์กสเตชั่น (Cluster of Workstation) คือ Tru64 UNIX, Solaris เป็นต้น [29] ซึ่งการเลือกใช้ระบบปฏิบัติการนั้นขึ้นอยู่กับระบบคลัสเตอร์ด้วยว่าเป็นแบบใด (PCs or Workstation) นอกจากนี้ยังขึ้นอยู่กับความสะดวก ความเชี่ยวชาญและอุปกรณ์ที่เลือกใช้ อีกอย่างหนึ่งคือจะขึ้นอยู่กับซอฟท์แวร์และชุดคำสั่งที่เลือกใช้ด้วย เช่น ถ้าเลือกใช้ OpenMosix ที่ทำงานได้บนระบบลีนุกซ์เท่านั้น ก็จำเป็นต้องเลือกใช้ลีนุกซ์เป็นระบบปฏิบัติการอีกทั้งฮาร์ดแวร์ที่สนับสนุนด้วย

โดยส่วนใหญ่ซอฟท์แวร์และชุดคำสั่งที่ถูกพัฒนาขึ้นมาเพื่อใช้ในระบบคลัสเตอร์นี้จะทำงานเข้ากันได้กับระบบปฏิบัติการตระกูลยูนิกซ์เกือบทุกชนิดอยู่แล้ว

ระบบคลัสเตอร์ที่ใช้ในงานวิจัยนี้เป็นระบบปฏิบัติลีนุกต์รุ่นซูซี (SUSE Version) ที่ใช้บนระบบคลัสเตอร์แบบพีซีเนื่องจากมีความสามารถและรองรับการทำงานได้หลากหลาย อีกทั้งยังมีความสามารถในการเฝ้าระวัง (Monitoring) ในส่วนงานของผู้ดูแลระบบ (Adminitrator) เป็นแบบออนไลน์ (Online) ทำให้เราสามารถตรวจสอบระบบคลัสเตอร์ผ่านทางอินเตอร์เน็ต (Internet) ได้ตลอดเวลาอีกด้วย

3) คลัสเตอร์มิดเดิลแวร์

คลัสเตอร์มิดเดิลแวร์ คือซอฟท์แวร์ที่ทำงานในระดับเดียวกับแกน (Kernel) ของระบบปฏิบัติการ มีหน้าที่ในการกระจายงาน (Process) จากโหนดหนึ่งไปยังโหนดอื่นๆ ที่อยู่ในระบบคลัสเตอร์เดียวกัน โดยส่วนใหญ่แล้วคลัสเตอร์มิดเดิลแวร์นี้จะเป็นซอฟท์แวร์ที่เขียนเพิ่มเติมเข้าไปในแกนของระบบปฏิบัติการ เพื่อให้ทุกเครื่องที่อยู่ในระบบคลัสเตอร์เสมือนเป็นเครื่องคอมพิวเตอร์หลายหน่วยประมวลผลขนาดใหญ่ ที่เสมือนมีหน่วยความจำ หน่วยประมวลผลอยู่ที่เดียวกันเหมือนกับระบบคอมพิวเตอร์แบบ SMP (Symmetric Multi Processor) หรือแบบ MMP (Massive Multi Processor)

4) การเชื่อมต่อเครือข่าย

การสื่อสารระหว่างหน่วยประมวลผลในระบบคลัสเตอร์จะทำผ่านระบบเครือข่ายความเร็วสูงซึ่งอุปกรณ์เครือข่ายแต่ละชนิดจะมีความเร็วและราคาแตกต่างกันไป ตัวอย่างอุปกรณ์เครือข่ายที่นิยมนำมาใช้ในการสร้างระบบคลัสเตอร์ มีดังนี้

ก) Ethernet [6] ในปัจจุบันอุปกรณ์ Ethernet นั้นได้ถูกพัฒนาให้มีความเร็วในการรับส่งข้อมูลสูงมากขึ้นจนถึงระดับกิกะบิตอีเทอร์เน็ต (Gigabit Ethernet) หรือ 10 กิกะบิตอีเทอร์เน็ต คือมีความเร็วในการส่งผ่านข้อมูลประมาณ 1-10 พันล้านบิตต่อวินาที ซึ่งเป็นแบบที่ใช้ในระบบคลัสเตอร์ที่ใช้ทำวิจัยนี้ด้วย

ข) Myrinet [21] มีความเร็วในการส่งผ่านข้อมูลประมาณ 2 พันล้านบิตต่อวินาที และมีค่า Latency Time ต่ำกว่าเครือข่ายแบบ Ethernet มาก แต่มีราคาแพงกว่าอีเทอร์เน็ตสูงมาก

ค) Quadrics [23] มีความเร็วในการส่งข้อมูลอยู่ที่ 340-900 MB/Second หรือประมาณ 2.65-7 กิกะบิตต่อวินาที และมีค่า Latency Time ต่ำมาก

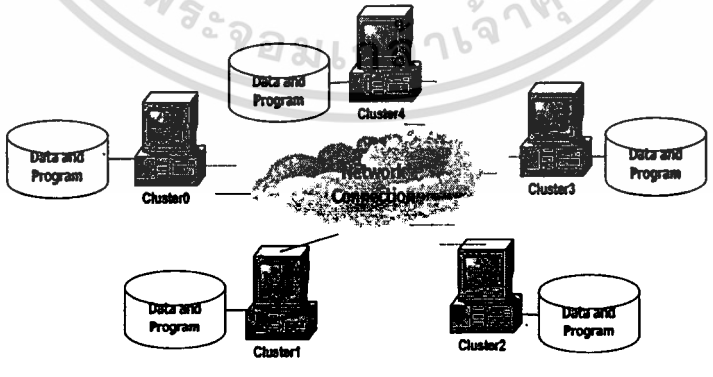
ง) InfiniBand [12] เป็นเทคโนโลยีที่มีความเร็วในการสื่อสารข้อมูลสูงมากถึง 5 กิกะบิตต่อวินาที และมีค่า Latency Time น้อยกว่า 10 ไมโครวินาที (Microsecond)

5) เอ็มพีไอซีเอช (MPICH)

เอ็มพีไอซีเอช (MPICH) [5] เป็นการพัฒนาชุดคำสั่งตามมาตรฐานขึ้นมาใช้งานจริงมากที่สุดคือ มาตรฐานการส่งผ่านข้อความ (Message Passing Interface: MPI) ซึ่งสามารถพัฒนาโปรแกรมภาษาต่างๆ เช่น โปรแกรมภาษาซี (C Language) โปรแกรมภาษาฟอร์แทน (Fortran Language) และโปรแกรมภาษาจาวา (Java Language) เป็นต้น ที่สนับสนุนการโปรแกรมแบบขนาน ซึ่งมีฟังก์ชันพื้นฐานต่างๆ รวมทั้งมีคำสั่งใช้งานเพื่อทำการคอมไพล์โปรแกรมที่เขียนขึ้นโดยเอ็มพีไอซีเอชนี้จะสนับสนุนการพัฒนาโปรแกรมภาษาซี และโปรแกรมภาษาฟอร์แทนเท่านั้น

4.2 ลักษณะข้อมูล การเลือกข้อมูลและเหตุผลการเลือก

ลักษณะของข้อมูลที่ใช้ในการทดลองเป็นข้อมูลชนิดตัวเลข (Integer) ที่ได้จากการสุ่มค่าตัวเลข (Random Number) จากโปรแกรมคอมพิวเตอร์ตามจำนวนที่ต้องการ โดยข้อมูลจะอยู่ในช่วง 0-999,999 ซึ่งจำนวนข้อมูลที่ใช้ในการทดลองตั้งแต่ 10-100 ล้านข้อมูล เนื่องจากข้อมูลจำนวนนี้แสดงให้เห็นถึงความแตกต่างในแต่ละแบบได้อย่างชัดเจน ถ้าใช้ข้อมูลที่มีจำนวนน้อยกว่าที่ได้กล่าวมา เช่น 1 ล้านข้อมูล จะไม่สามารถแสดงให้เห็นถึงความแตกต่างในแต่ละแบบได้เด่นชัดมากนัก ผู้วิจัยทำการเลือกข้อมูลชนิดตัวเลข เนื่องจากมีความใกล้เคียงกับการใช้งานในชีวิตประจำวันอย่างยิ่ง เช่น การเรียงลำดับของเลขที่บัญชีธนาคาร เลขที่บัตรประจำตัวประชาชน เลขที่บัตรสังคม การสร้างดัชนีแฟ้มข้อมูล เป็นต้น ซึ่งในการวิจัยนี้ (การเรียงลำดับ) ข้อมูลจะถูกจัดเก็บอยู่ในทุกๆ เครื่องของระบบคลัสเตอร์ที่มีประสิทธิภาพและมีความสามารถในการประมวลผลงานเท่ากัน ซึ่งเรียกการเก็บข้อมูลแบบนี้ว่า "การเก็บข้อมูลแบบกระจาย" (Distributed Data) ดังรูปที่ 4.2



รูปที่ 4.2 ระบบคลัสเตอร์ที่มีการเก็บข้อมูลและโปรแกรมแบบกระจาย

4.3 ผลการทดลอง

การวัดประสิทธิภาพของการเรียงลำดับข้อมูลนี้จะประเมินผลจากเวลาที่ใช้ในการเรียงลำดับข้อมูล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) โดยในผลการทดลองนี้จะทำการเปรียบเทียบประสิทธิภาพของการเรียงลำดับข้อมูลที่ได้นำเสนอทั้ง 4 แบบ คือ

1) การเรียงลำดับแบบบีเอส (BS: Bitonic Sort)
2) การเรียงลำดับแบบซีอีบีเอส (CEBS: Communication-Efficient BS)
3) การเรียงลำดับแบบเอสอีบีเอส (SEBS: Space-Efficient BS)
4) การเรียงลำดับแบบซีเอสอีบีเอส (CSEBS: Communication-Space Efficient BS)

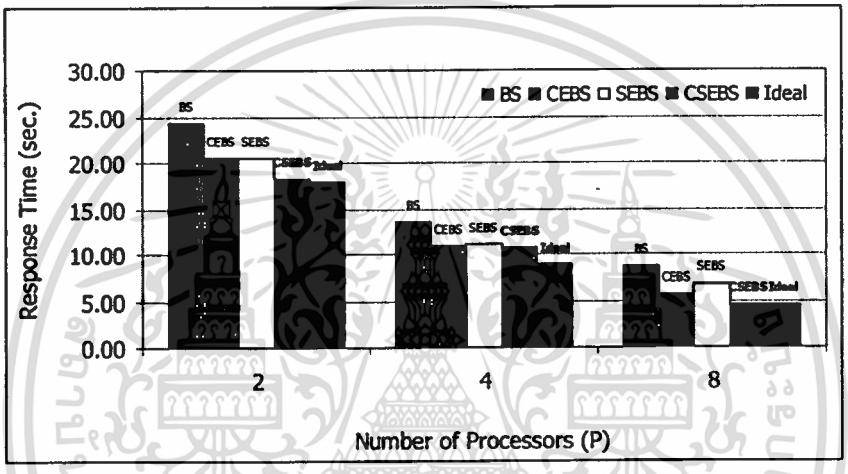
4.3.1 ผลการทดลองเปรียบเทียบกับเวลาในอุดมคติ

การทดลองเปรียบเทียบเวลาที่ประมวลผลจริงกับเวลาในอุดมคติ เป็นการวัดประสิทธิภาพของระบบ (System Performance) ที่ใช้ในการทดลองดังนี้ ตารางที่ 4.1 และรูปที่ 4.3 แสดงผลการเปรียบเทียบเวลาที่ใช้ในการเรียงลำดับข้อมูล (Response Time) ทั้ง 4 แบบ ($T_P = T_S/P + T_C$) โดยในงานวิจัยนี้เน้นการเพิ่มประสิทธิภาพของเวลาที่ใช้ในการติดต่อสื่อสาร (Communication Time: T_C) ส่วนเวลาที่ใช้ในการเรียงลำดับข้อมูลในอุดมคติ (Response Time of IdeaI Case) จะคำนวณโดยสมมติว่าเวลาที่ใช้ในการติดต่อสื่อสาร เพื่อแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลเท่ากับศูนย์ (T_C=0) ดังนั้น $T_P = T_S/P$ ตามสมการที่ 2.5 ตารางที่ 4.2 และรูปที่ 4.4 แสดงอัตราการเพิ่มขึ้นของความเร็ว (Speedup) ทั้ง 4 แบบ ($S_P = T_S/T_P$) เปรียบเทียบกับอัตราการเพิ่มขึ้นของความเร็วในอุดมคติ (S_P=P) ในตารางที่ 4.3 และรูปที่ 4.5 แสดงประสิทธิภาพในการเรียงลำดับข้อมูล ทั้ง 4 แบบ ($E_P = S_P/P$) เปรียบเทียบกับประสิทธิภาพในอุดมคติ (E_P =1)

ตารางที่ 4.1 เวลาที่ใช้ในการเรียงลำดับข้อมูล (Response Time) ของ 4 แบบโดยเปรียบเทียบกับเวลาที่ใช้ในการเรียงลำดับข้อมูลในอุดมคติ (Response Time of Ideal Case) เมื่อข้อมูลมีขนาด 50 ล้านข้อมูล และหน่วยประมวลผลตั้งแต่ 2, 4 และ 8

วิธีการ/จำนวนหน่วยประมวลผล	2	4	8
เวลาในอุดมคติ	17.88	8.94	4.47
เวลาในการเรียงลำดับแบบ "BS"	24.36	13.54	8.77
เวลาในการเรียงลำดับแบบ "CEBS"	20.57	11.01	5.57
เวลาในการเรียงลำดับแบบ "SEBS"	20.61	11.12	6.76
เวลาในการเรียงลำดับแบบ "CSEBS"	18.22	10.71	4.54

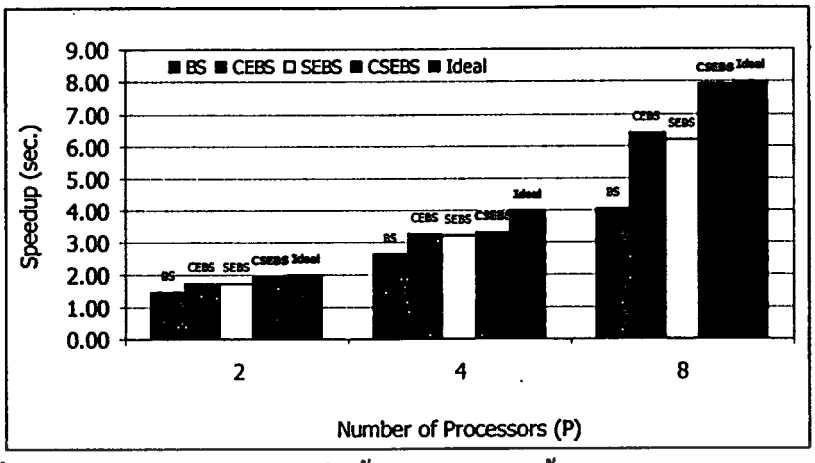
จากรูปที่ 4.3 สามารถอธิบายได้ว่าเมื่อจำนวนหน่วยประมวลผลเพิ่มขึ้นจะทำให้เวลาที่ใช้ในการเรียงลำดับข้อมูลลดลง โดยเวลาที่ใช้ในการเรียงลำดับข้อมูลแบบซีเอสอีบีเอส (CSEBS) เมื่อจำนวนหน่วยประมวลผลเท่ากับ 2, 4 และ 8 หน่วยประมวลผล จะให้ผลดังนี้ 18.22, 10.71 และ4.54 วินาทีใกล้เคียงกับเวลาในอุดมคติคือ 17.88, 8.94 และ 4.47 วินาที ซึ่งคิดเป็น 98%, 83% และ 98% ตามลำดับ ซึ่งเป็นแบบที่ให้ผลดีที่สุด ส่วนแบบอื่นๆ เช่น แบบซีอีบีเอส (CEBS) และแบบเอสอีบีเอส (SEBS) จะใช้เวลาในการเรียงลำดับข้อมูลใกล้เคียงกัน และแบบสุดท้ายคือแบบบีเอส (BS) เป็นแบบที่ใช้เวลาในการเรียงลำดับข้อมูลมากที่สุด



รูปที่ 4.3 เปรียบเทียบเวลาที่ใช้ในการเรียงลำดับทั้ง 4 แบบกับเวลาในอุดมคติ

ตารางที่ 4.2 อัตราการเพิ่มขึ้นของความเร็ว (Speedup) ของ 4 แบบ โดยเปรียบเทียบกับอัตราการเพิ่มขึ้นของความเร็วในอุดมคติ (Speedup of Ideal Case) เมื่อข้อมูลมีขนาด 50 ล้านข้อมูล และหน่วยประมวลผลตั้งแต่ 2, 4 และ 8

วิธีการ/จำนวนหน่วยประมวลผล	2	4	8
อัตราการเพิ่มขึ้นของความเร็วในอุดมคติ	2.00	4.00	8.00
อัตราการเพิ่มขึ้นของความเร็วแบบ "BS"	1.47	2.64	4.08
อัตราการเพิ่มขึ้นของความเร็วแบบ "CEBS"	1.74	3.25	6.42
อัตราการเพิ่มขึ้นของความเร็วแบบ "SEBS"	1.74	3.22	6.21
อัตราการเพิ่มขึ้นของความเร็วแบบ "CSEBS"	1.96	3.34	7.88

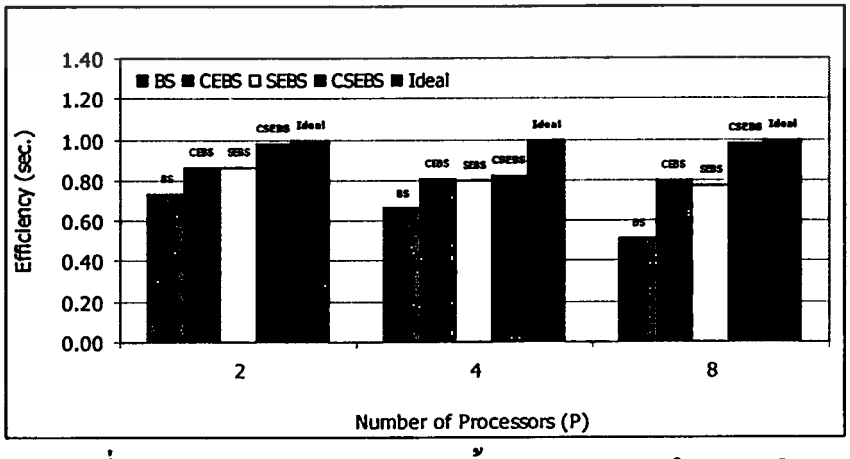


รูปที่ 4.4 เปรียบเทียบอัตราการเพิ่มขึ้นของความเร็วทั้ง 4 แบบ กับเวลาในอุดมคติ

จากรูปที่ 4.4 สามารถอธิบายได้ว่าอัตราการเพิ่มขึ้นของความเร็วของการเรียงลำดับแบบซีเอสอีบีเอส (CSEBS) เพิ่มขึ้นเป็น 1.96 เมื่อใช้ 2 หน่วยประมวลผล และ 3.34 เมื่อใช้ 4 หน่วยประมวผล สุดท้ายคือ 7.88 เมื่อใช้ 8 หน่วยประมวลผล ซึ่งเป็นแบบที่ดีที่สุด ส่วนแบบซีอีบีเอส (CEBS) และแบบเอสอีบีเอส (SEBS) จะให้ผลใกล้เคียงกัน และแบบสุดท้ายคือแบบบีเอส (BS) มีอัตราการเพิ่มขึ้นของความเร็วน้อยกว่าสามแบบแรกที่ได้กล่าวมาแล้ว

ตารางที่ 4.3 ประสิทธิภาพ (Efficiency) ทั้ง 4 แบบ โดยเปรียบเทียบกับประสิทธิภาพในอุดมคติ (Efficiency of Ideal Case) เมื่อข้อมูลมีขนาด 50 ล้านข้อมูล และหน่วยประมวลผลตั้งแต่ 2, 4 และ 8 ตามลำดับ

วิธีการ/จำนวนหน่วยประมวลผล	2	4	8
ประสิทธิภาพในอุดมคติ	1.00	1.00	1.00
ประสิทธิภาพแบบ "BS"	0.73	0.66	0.51
ประสิทธิภาพแบบ "CEBS"	0.87	0.81	0.80
ประสิทธิภาพแบบ "SEBS"	0.87	0.80	0.78
ประสิทธิภาพแบบ "CSEBS"	0.98	0.83	0.98



รูปที่ 4.5 เปรียบเทียบประสิทธิภาพทั้ง 4 แบบ กับเวลาในอุดมคติ

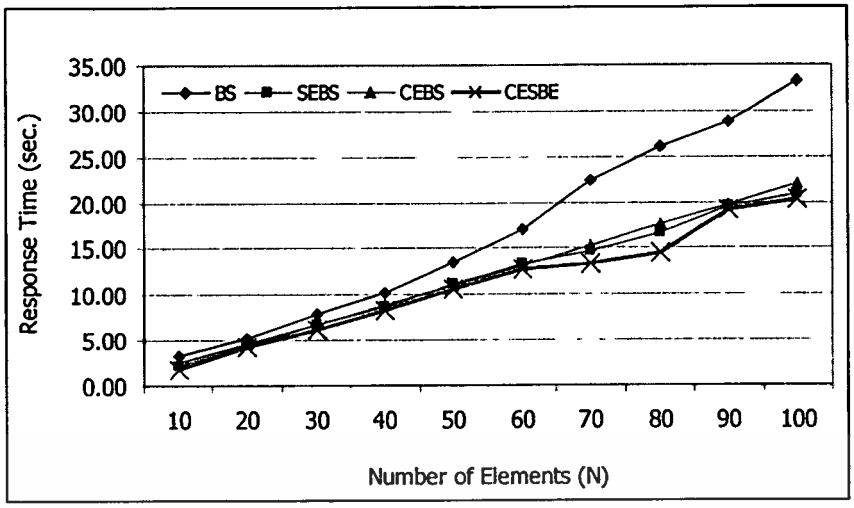
จากรูปที่ 4.5 แสดงประสิทธิภาพของการเรียงลำดับข้อมูลทั้งสี่แบบจะเห็นว่าการเรียงลำดับแบบซีเอสอีบีเอส (CSEBS) จะให้ผลเข้าใกล้ 1 เมื่อใช้ 2 หน่วยประมวลผล ประสิทธิภาพเป็น 0.98 และเมื่อใช้ 4 หน่วยประมวลผล ประสิทธิภาพเป็น 0.83 สุดท้ายคือ 0.98 เมื่อใช้ 8 หน่วยประมวลผล สำหรับประสิทธิภาพแบบซีอีบีเอส (CEBS) และแบบเอสอีบีเอส (SEBS) จะให้ผลใกล้เคียงกัน ส่วนแบบบีเอส (BS) จะมีประสิทธิภาพน้อยที่สุดเมื่อเทียบกับสามแบบแรก

4.3.2 ผลการทดลองเปรียบเทียบเมื่อขนาดข้อมูลที่ใช้ทดลองต่างกัน

การทดลองเปรียบเทียบการเรียงลำดับข้อมูลทั้งสี่แบบที่ได้กล่าวมาในข้างต้น สำหรับข้อมูลที่มีขนาดต่างกัน โดยทดลองที่ขนาดของชุดข้อมูลตั้งแต่ 10, 20, 30, 40, 50, 60, 70, 80, 90 และ 100 ล้านข้อมูล และจะกำหนดให้จำนวนหน่วยประมวลผลที่ใช้ในการคำนวณเท่ากับ 4 หน่วยประมวลผล ซึ่งแสดงผลการทดลองดังนี้ ตารางที่ 4.4 และรูปที่ 4.6 แสดงเวลาที่ใช้ในการเรียงลำดับข้อมูล (Response Time) ที่เป็นผลรวมของเวลาที่ใช้ในการติดต่อสื่อสารและเวลาที่ใช้ในการประมวลผล ($T_p = T_S/P+T_C$) นอกจากนี้เราสามารถนำเวลาที่ใช้ในการเรียงลำดับข้อมูลมาคำนวณหาอัตราการเพิ่มขึ้นของความเร็ว ตามสมการที่ 2.7 ($S_p = T_S/T_p$) ดังแสดงในตารางที่ 4.5 และรูปที่ 4.7 จากนั้นนำค่าของอัตราการเพิ่มขึ้นของความเร็วมาคำนวณหาประสิทธิภาพ ตามสมการที่ 2.8 ($E_p = S_p/P$) ดังแสดงในตารางที่ 4.6 และรูปที่ 4.8

ตารางที่ 4.4 เวลาที่ใช้ในการเรียงลำดับข้อมูล (Response Time) ทั้ง 4 แบบ เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล และขนาดข้อมูลตั้งแต่ 10, 20, 30, 40, 50, 60, 70, 80, 90 และ 100 ล้านชุดข้อมูลตามลำดับ

จำนวนชุดข้อมูล/วิธีการ	BS	CEBS	SEBS	CSEBS
10,000,000	3.32	2.44	2.20	1.77
20,000,000	5.15	4.68	4.40	4.19
30,000,000	8.01	6.70	6.68	6.24
40,000,000	10.21	8.94	8.65	8.37
50,000,000	13.54	11.01	11.12	10.71
60,000,000	16.99	13.18	13.31	12.83
70,000,000	22.49	15.18	14.78	13.33
80,000,000	26.11	17.68	16.63	14.50
90,000,000	28.81	19.72	19.51	19.19
100,000,000	33.17	21.96	20.84	20.24

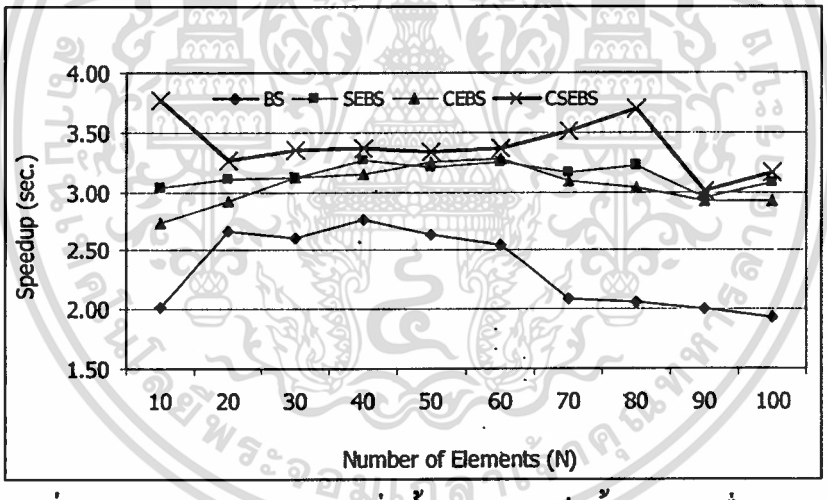


รูปที่ 4.6 เปรียบเทียบเวลาที่ใช้ในการเรียงลำดับ ทั้ง 4 แบบ เมื่อ P = 4

จากรูปที่ 4.6 แสดงเวลาที่ใช้ในการเรียงลำดับข้อมูล (Response Time) โดยทำการเปรียบเทียบเวลาทั้ง 4 แบบ ซึ่งเวลาที่ใช้ในการเรียงลำดับข้อมูลประกอบด้วยเวลา 2 ส่วนคือ ส่วนแรกเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Communication Time) ส่วนที่สองเวลาที่ใช้ในการประมวลผลข้อมูล (Computation Time) ซึ่งประกอบด้วยเวลาที่ใช้ในการประมวลผลแบบขนาน (Parallel Computation Time) และเวลาที่ใช้ในการประมวลผลแบบอนุกรม (Sequential Computation Time) เนื่องจาก P < N หรือเวลาที่ใช้ในการเรียงลำดับข้อมูลในแต่ละหน่วยประมวลผล (Local Sequential Sort: Quick sort) ซึ่งในส่วนนี้จะทำตอนโหลดข้อมูลเข้าสู่หน่วยความจำในครั้งแรกเท่านั้น ผลการทดลองจากรูปที่ 4.6 จะเห็นว่าเมื่อข้อมูลมีขนาดใหญ่ขึ้นเวลาที่ใช้ในการเรียงลำดับข้อมูลนั้นจะเพิ่มขึ้นตามไปด้วย โดยการเรียงลำดับข้อมูลแบบบีเอส (BS) และแบบเอสอีบีเอส (SEBS) จะใช้เวลาในการแลกเปลี่ยนข้อมูลมากกว่าการเรียงลำดับแบบซีอีบีเอส (CEBS) และแบบซีเอสอีบีเอส (CSEBS) เนื่องจากขนาดของชุดข้อมูลที่ส่งและรับในแต่ละรอบนั้น สองแบบแรก (BS, SEBS) จะส่งและรับข้อมูลมีขนาดเท่ากับ N/P ในขณะที่สองแบบหลัง (CEBS, CSEBS) โดยส่วนใหญ่จะส่งและรับข้อมูลที่มีขนาดน้อยกว่า N/P หรือในบางรอบเวลาไม่จำเป็นต้องมีการแลกเปลี่ยนข้อมูลเกิดขึ้นทำให้เวลาในการติดต่อสื่อสารระหว่างหน่วยประมวลผลลดลง ดังนั้นทำให้เวลาที่ใช้ในการประมวลผลทั้งหมดลดลงด้วย ซึ่งเมื่อเปรียบเทียบกับเวลาที่ใช้ในการติดต่อสื่อสารของทั้ง 4 แบบ พบว่าวิธีซีเอสอีบีเอสมีค่าน้อยที่สุดในทุกๆ ผลการทดลอง

ตารางที่ 4.5 อัตราการเพิ่มขึ้นของความเร็ว (Speedup) ทั้ง 4 แบบ เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล และขนาดข้อมูลตั้งแต่ 10, 20, 30, 40, 50, 60, 70, 80, 90 และ 100 ล้านชุดข้อมูล ตามลำดับ

จำนวนชุดข้อมูล/วิธีการ	BS	CEBS	SEBS	CSEBS
10,000,000	2.01	2.74	3.04	3.78
20,000,000	2.66	2.92	3.11	3.27
30,000,000	2.61	3.12	3.13	3.35
40,000,000	2.76	3.16	3.26	3.37
50,000,000	2.64	3.25	3.22	3.34
60,000,000	2.55	3.28	3.25	3.37
70,000,000	2.09	3.09	3.17	3.52
80,000,000	2.06	3.04	3.23	3.71
90,000,000	2.00	2.92	2.95	3.00
100,000,000	1.93	2.92	3.07	3.17

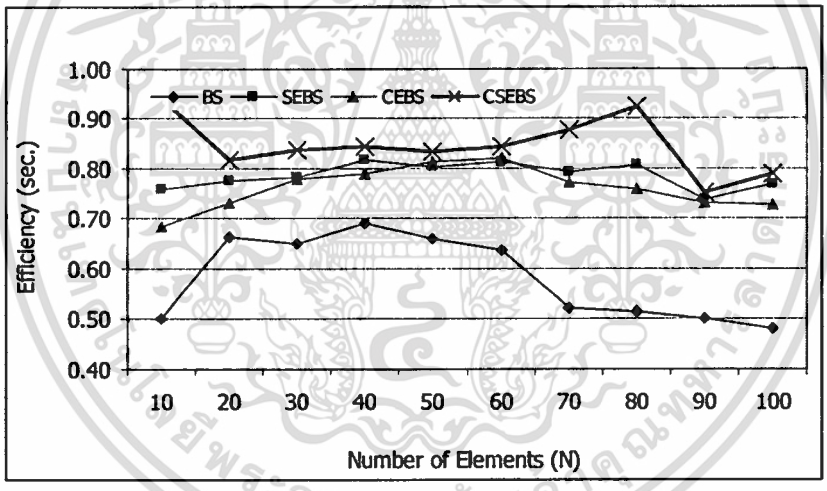


รูปที่ 4.7 เปรียบเทียบอัตราการเพิ่มขึ้นของความเร็วทั้ง 4 แบบ เมื่อ P = 4

จากรูปที่ 4.7 แสดงการเปรียบเทียบอัตราการเพิ่มขึ้นของความเร็วทั้ง 4 แบบ เมื่อจำนวนหน่วยประมวลผลเท่ากับ 4 หน่วยประมวลผล และขนาดข้อมูลตั้งแต่ 10 20 30 จนถึง 100 ล้านชุดข้อมูล จะเห็นว่าการเรียงลำดับแบบซีเอสอีบีเอส (CSEBS) เมื่อข้อมูลมีขนาดเท่ากับ 10 และ 80 ล้านชุดข้อมูล อัตราการเพิ่มขึ้นของความเร็วเพิ่มขึ้นเป็น 3.78 และ 3.71 ซึ่งเป็นจำนวนข้อมูลที่เหมาะสมที่สุดสำหรับจำนวนหน่วยประมวลผล 4 หน่วยประมวลผล ส่วนแบบซีอีบีเอส (CEBS) และแบบเอสอีบีเอส (SEBS) จะให้ผลใกล้เคียงกัน สุดท้ายคือแบบบีเอส (BS) มีอัตราการเพิ่มขึ้นของความเร็วน้อยที่สุด ซึ่งเมื่อเปรียบเทียบทั้ง 4 แบบ พบว่าแบบซีเอสอีบีเอสมีค่ามากที่สุด

ตารางที่ 4.6 ประสิทธิภาพ (Efficiency) ทั้ง 4 แบบ เมื่อใช้หน่วยประมวลผลเท่ากับ 4 หน่วย และขนาดข้อมูลตั้งแต่ 10, 20, 30, 40, 50, 60, 70, 80, 90 และ 100 ล้านชุดข้อมูล

จำนวนชุดข้อมูล/วิธีการ	BS	CEBS	SEBS	CSEBS
10,000,000	0.50	0.68	0.76	0.94
20,000,000	0.66	0.73	0.78	0.82
30,000,000	0.65	0.78	0.78	0.84
40,000,000	0.69	0.79	0.82	0.84
50,000,000	0.66	0.81	0.80	0.83
60,000,000	0.64	0.82	0.81	0.84
70,000,000	0.52	0.77	0.79	0.88
80,000,000	0.51	0.76	0.81	0.93
90,000,000	0.50	0.73	0.74	0.75
100,000,000	0.48	0.73	0.77	0.79



รูปที่ 4.8 เปรียบเทียบประสิทธิภาพทั้ง 4 แบบ เมื่อ P = 4

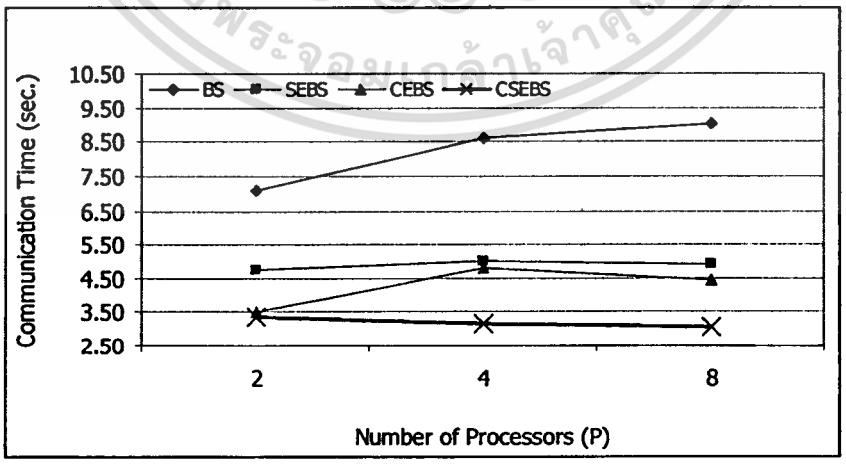
จากรูปที่ 4.8 แสดงประสิทธิภาพของการเรียงลำดับข้อมูลทั้ง 4 แบบ จะเห็นว่าการเรียงลำดับแบบซีเอสอีบีเอส (CSEBS) เมื่อข้อมูลเท่ากับ 10 และ 80 ล้านชุดข้อมูล จะให้ประสิทธิภาพคิดเป็น 94% และ 93% ตามลำดับ เมื่อใช้ 4 หน่วยประมวลผล ส่วนแบบซีอีบีเอส (CEBS) และแบบเอสอีบีเอส (SEBS) จะให้ผลใกล้เคียงกัน สุดท้ายคือแบบบีเอส (BS) มีอัตราการเพิ่มขึ้นของความเร็วน้อยที่สุด

4.3.3 การเปรียบเทียบประสิทธิภาพโดยปรับเปลี่ยนที่จำนวนของหน่วยประมวลผล

การเรียงลำดับของทั้งสี่แบบจะกำหนดให้ขนาดข้อมูล (N) เท่ากับ 50 ล้านข้อมูล และจะทำการเพิ่มจำนวนหน่วยประมวลผลตั้งแต่ 2, 4 และ 8 ตามลำดับ โดยในงานวิจัยนี้เน้นการเพิ่มประสิทธิภาพของเวลาที่ใช้ในการติดต่อสื่อสาร ดังนั้นผลการทดลองทั้ง 4 แบบนี้ จะพิจารณาเวลาที่ใช้ในการเรียงลำดับได้เป็น 2 ส่วนคือ ส่วนแรกเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลเพื่อแลกเปลี่ยนข้อมูล (Communication Time) ดัง แสดงในตารางที่ 4.7 และรูปที่ 4.9 ส่วนที่สองเวลาทั้งหมดที่ใช้ในการประมวลผลเพื่อเรียงลำดับข้อมูล (Computation Time) ซึ่งเป็นผลรวมของเวลาที่ใช้ในการติดต่อสื่อสารและเวลาที่ใช้ในการประมวลผล ($T_P = T_S/P + T_C$) แสดงในตารางที่ 4.8 และรูปที่ 4.10 นอกจากนี้เราสามารถนำเวลาที่ใช้ในการเรียงลำดับข้อมูลมาคำนวณหาอัตราการเพิ่มขึ้นของความเร็ว ตามสมการที่ 2.7 ($S_P = T_S/T_P$) ดัง ในตารางที่ 4.9 และรูปที่ 4.11 จากนั้นนำค่าของอัตราการเพิ่มขึ้นของความเร็วมาคำนวณหาประสิทธิภาพ ตามสมการที่ 2.8 ($E_P = S_P/P$) ดัง แสดงในตารางที่ 4.10 และรูปที่ 4.12

ตารางที่ 4.7 เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Communication Time) ทั้ง 4 แบบ เมื่อข้อมูลมีขนาด 50 ล้านข้อมูล และหน่วยประมวลผลตั้งแต่ 2, 4 และ 8

วิธีการ/จำนวนหน่วยประมวลผล	2	4	8
เวลาใช้ในการติดต่อสื่อสารแบบ "BS"	7.07	8.60	9.03
เวลาใช้ในการติดต่อสื่อสารแบบ "CEBS"	3.53	4.79	4.46
เวลาใช้ในการติดต่อสื่อสารแบบ "SEBS"	4.76	5.00	4.89
เวลาใช้ในการติดต่อสื่อสารแบบ "CSEBS"	3.38	3.17	3.08

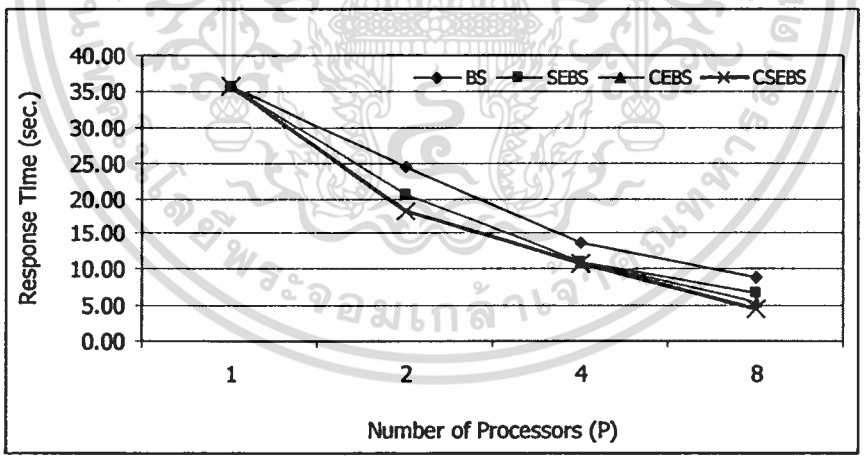


รูปที่ 4.9 เปรียบเทียบเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลทั้ง 4 แบบ

จากรูปที่ 4.9 แสดงเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล ซึ่งจะเห็นว่าการเรียงลำดับแบบซีอีบีเอส (CEBS) และแบบซีเอสอีบีเอส (CSEBS)ใช้เวลาในการแลกเปลี่ยนข้อมูลน้อยกว่าอีกสองแบบ (SEBS, BS) เนื่องจากก่อนการแลกเปลี่ยนข้อมูลทั้งสองแบบ (CEBS, CSEBS) นี้จะตรวจสอบหารูปแบบการแลกเปลี่ยนข้อมูลก่อน (Hold, Swap and Partial) ทำให้เวลาในการติดต่อสื่อสาร (Communication Time) ระหว่างหน่วยประมวลผลลดลง ซึ่งเมื่อเปรียบเทียบทั้ง 4 แบบ พบว่าแบบซีเอสอีบีเอส (CSEBS) ใช้เวลาในการติดต่อสื่อสารน้อยที่สุดในทุกๆ ผลการทดลอง และเมื่อเทียบกับแบบบีเอส (BS) เดิม พบว่าแบบซีเอสอีบีเอส (CSEBS) แบบซีอีบีเอส (CEBS) และแบบเอสอีบีเอส (SEBS) สามารถลดเวลาในการติดต่อสื่อสารระหว่างหน่วยประมวลผลลงได้อย่างน้อย 52%, 44% และ 33% ตามลำดับ

ตารางที่ 4.8 เวลาที่ใช้ในการเรียงลำดับข้อมูล (Response Time) ทั้ง 4 แบบ เมื่อข้อมูลมีขนาด 50 ล้านข้อมูล และหน่วยประมวลผลตั้งแต่ 1, 2,4 และ 8 ตามลำดับ

วิธีการ/จำนวนหน่วยประมวลผล	1	2	4	8
เวลาในการเรียงลำดับแบบ "BS"	35.76	24.36	13.54	8.77
เวลาในการเรียงลำดับแบบ "CEBS"	35.76	20.57	11.01	5.57
เวลาในการเรียงลำดับแบบ "SEBS"	35.76	20.61	11.12	5.76
เวลาในการเรียงลำดับแบบ "CSEBS"	35.76	18.22	10.71	4.54



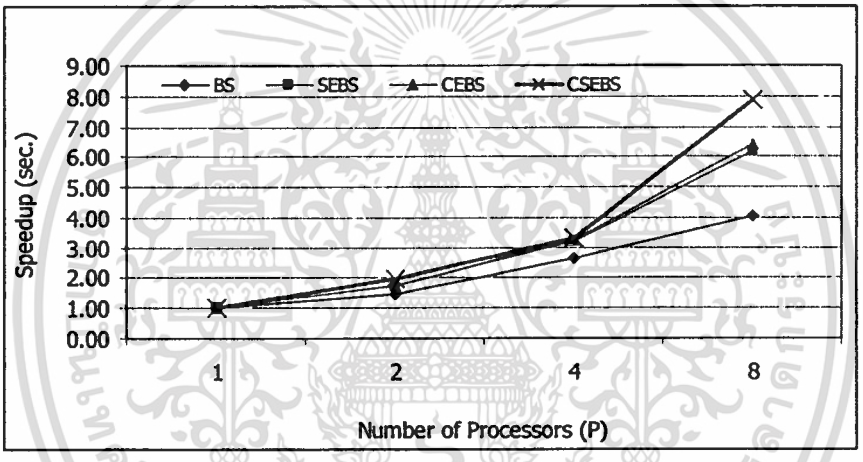
รูปที่ 4.10 เปรียบเทียบเวลาที่ใช้ในการเรียงลำดับทั้ง 4 แบบ เมื่อ N = 50

จากรูปที่ 4.10 แสดงเวลาที่ใช้ในการเรียงลำดับข้อมูล (Response Time) ทั้ง 4 แบบ ซึ่งจะเห็นว่าการเรียงลำดับแบบซีอีบีเอส (CEBS) กับแบบเอสอีบีเอส (SEBS) นี้ใช้เวลาในการเรียงลำดับข้อมูลให้ผลใกล้เคียงกัน ส่วนการเรียงลำดับข้อมูลแบบซีเอสอีบีเอส (CSEBS) จะใช้เวลาในการเรียงลำดับข้อมูลน้อยที่สุด เนื่องจากเวลาในการติดต่อสื่อสารลดลง (ดังตารางที่ 4.7 และรูปที่ 4.9) และเมื่อเทียบกับแบบบีเอส (BS) เดิม พบว่าแบบซีเอสอีบีเอส (CSEBS) แบบซีอีบีเอส (CEBS) และ

แบบเอสอีบีเอส (SEBS) สามารถลดเวลาในการติดต่อสื่อสารระหว่างหน่วยประมวลผลลงได้อย่างน้อย 21%, 16% และ 15% ตามลำดับ

ตารางที่ 4.9 อัตราการเพิ่มขึ้นของความเร็ว (Speedup) ทั้ง 4 แบบ เมื่อข้อมูลมีขนาด 50 ล้านข้อมูล และหน่วยประมวลผลตั้งแต่ 1, 2, 4 และ 8 ตามลำดับ

วิธีการ/จำนวนหน่วยประมวลผล	1	2	4	8
อัตราการเพิ่มขึ้นของความเร็วแบบ "BS"	1.00	1.47	2.64	4.08
อัตราการเพิ่มขึ้นของความเร็วแบบ "CEBS"	1.00	1.74	3.25	6.42
อัตราการเพิ่มขึ้นของความเร็วแบบ "SEBS"	1.00	0.74	3.22	3.22
อัตราการเพิ่มขึ้นของความเร็วแบบ "CSEBS"	1.00	1.96	3.34	7.88

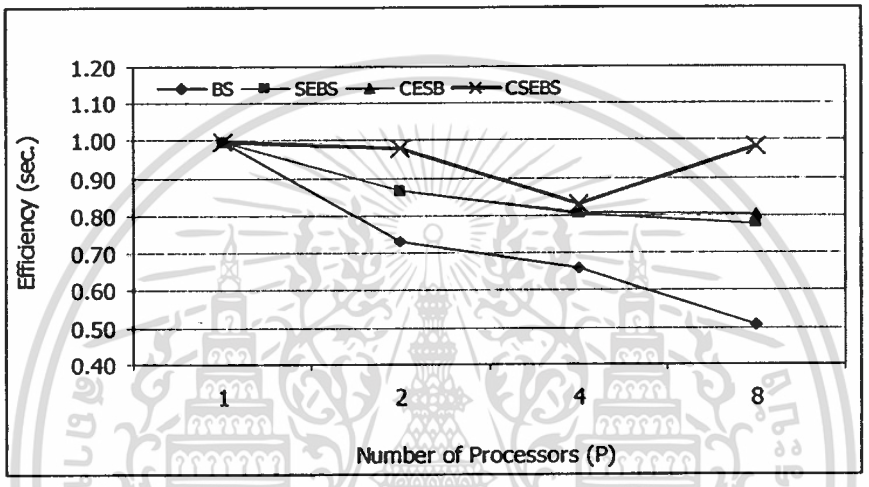


รูปที่ 4.11 เปรียบเทียบอัตราการเพิ่มขึ้นของความเร็วทั้ง 4 แบบ เมื่อ N = 50

จากรูปที่ 4.11 แสดงอัตราการเพิ่มขึ้นของความเร็ว ซึ่งการเรียงลำดับแบบเอสอีบีเอส (CEBS) และแบบซีอีบีเอส (SEBS) จะมีอัตราการเพิ่มขึ้นของความเร็วให้ผลใกล้เคียงกันมาก ส่วนการเรียงลำดับแบบบีเอส (BS) และแบบซีเอสอีบีเอส (CSEBS) จะให้ผลของอัตราการเพิ่มขึ้นของความเร็วคิดเป็น 74% เมื่อใช้ 2 หน่วยประมวลผล และอัตราการเพิ่มขึ้นของความเร็วคิดเป็น 79% เมื่อใช้ 4 หน่วยประมวลผล สุดท้ายคือเมื่อใช้ 8 หน่วยประมวลผล อัตราการเพิ่มขึ้นของความเร็วคิดเป็น 52% ซึ่งเมื่อเปรียบเทียบทั้ง 4 แบบ พบว่าการเรียงลำดับข้อมูลแบบซีเอสอีบีเอส (CSEBS) เป็นแบบที่ให้ผลดีที่สุด ในทุกๆ ผลการทดลอง

ตารางที่ 4.10 ประสิทธิภาพ (Efficiency) ทั้ง 4 แบบ เมื่อข้อมูลมีขนาด 50 ล้านข้อมูล และหน่วยประมวลผลตั้งแต่ 1, 2, 4 และ 8 ตามลำดับ

วิธีการ/จำนวนหน่วยประมวลผล	1	2	4	8
ประสิทธิภาพแบบ "BS"	1.00	0.73	0.66	0.51
ประสิทธิภาพแบบ "CEBS"	1.00	0.87	0.81	0.80
ประสิทธิภาพแบบ "SEBS"	1.00	0.87	0.80	0.78
ประสิทธิภาพแบบ "CSEBS"	1.00	0.98	0.83	0.98



รูปที่ 4.12 เปรียบเทียบประสิทธิภาพทั้ง 4 แบบ เมื่อ N = 50

จากรูปที่ 4.12 แสดงผลการเปรียบเทียบประสิทธิภาพของการเรียงลำดับทั้ง 4 แบบ ซึ่งการเรียงลำดับแบบซีเอสอีบีเอส (CSEBS) จะให้ประสิทธิภาพเข้าใกล้ 1 มากที่สุดคิดเป็น 0.98 และ 0.98 เมื่อใช้หน่วยประมวลผลเท่ากับ 2 และ 8 ตามลำดับ สำหรับข้อมูลขนาด 50 ล้านข้อมูล ส่วนแบบซีอีบีเอส (CEBS) และแบบเอสอีบีเอส (SEBS) จะให้ผลใกล้เคียงกัน และแบบสุดท้ายคือแบบบีเอส (BS) จะให้ผลน้อยที่สุดเมื่อเปรียบกับสามแบบแรก ซึ่งการทดลองที่ผ่านมาพบว่าแบบซีเอสอีบีเอสเป็นแบบที่ให้ผลดีที่สุด

บทที่ 5
สรุปผลและแนวทางการพัฒนางานวิจัย

5.1 สรุปผลและวิเคราะห์ผลการทดลอง

การเรียงลำดับแบบขนานด้วยวิธีไบโทนิคที่ได้เสนอไว้ในบทที่ 3 ซึ่งมีทั้งหมด 4 แบบคือ แบบที่1) บีเอส (BS) แบบที่ 2) ซีอีบีเอส (CEBS) แบบที่ 3) เอสอีบีเอส (SEBS) และแบบที่ 4) ซีเอสอีบีเอส (CSEBS) โดยแต่ละแบบดังกล่าวมีวัตถุประสงค์และความยากง่ายในขั้นตอนวิธีที่แตกต่างกันดังนี้คือ วิธีการเรียงลำดับแบบที่ 1) บีเอส (BS) เป็นแบบดั้งเดิม และง่ายที่สุดเมื่อเทียบกับแบบอื่นๆ วิธีการเรียงลำดับแบบที่ 2 ซีอีบีเอส (CEBS) เป็นแบบที่เน้นการเพิ่มประสิทธิภาพในการติดต่อสื่อระหว่างหน่วยประมวลผล ซึ่งมีความซับซ้อนกว่าแบบแรก แบบที่ 3 เอสอีบีเอส (SEBS) เป็นแบบที่เน้นการลดการใช้พื้นที่ในหน่วยความจำในการเก็บข้อมูลที่ซ้ำซ้อน และเพิ่มประสิทธิภาพในการรวมข้อมูลในแต่ละหน่วยประมวลผล และแบบที่ 4 ซีเอสอีบีเอส (CSEBS) เป็นแบบที่นำข้อดีของแบบที่ 2 และแบบที่ 3 มารวมกัน โดยแบบสุดท้ายนี้ เป็นแบบที่มีประสิทธิภาพมากที่สุด

จากผลการทดลองที่ผ่านมาในบทที่ 4 ซึ่งเป็นการทดลองบนระบบพีซีคลัสเตอร์จะเห็นว่าการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคในแต่ละแบบจะมีการเพิ่มประสิทธิภาพในด้านต่างๆ ที่แตกต่างกันสอดคล้องกับทฤษฎีที่กล่าวมาแล้วในบทที่ 3 ซึ่งสามารถแบ่งได้เป็น 3 ด้าน ดังนี้

1) การเพิ่มประสิทธิภาพโดยการลดเวลาที่ใช้ในการติดต่อสื่อสาร

2) การเพิ่มประสิทธิภาพโดยการลดเวลาที่ใช้ในการประมวลผลทั้งหมด

3) การเพิ่มประสิทธิภาพโดยการลดการใช้พื้นที่ในหน่วยความจำในการเก็บข้อมูลที่ซ้ำซ้อน

ตารางที่ 5.1 แสดงเปรียบเทียบการเพิ่มประสิทธิภาพของการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคทั้ง 4 แบบ

แบบที่	การเพิ่มประสิทธิภาพ (เทียบกับวิธีบีเอส)		
	เวลาที่ใช้ในการติดต่อสื่อสาร	เวลาที่ใช้ในการประมวลผลทั้งหมด	หน่วยความจำที่ใช้ในการเก็บข้อมูล
1) บีเอส (BS)	-	-	-
2) ซีอีบีเอส (CEBS)	อย่างน้อย 44%	อย่างน้อย 16%	ประมาณ 20%
3) เอสอีบีเอส (SEBS)	อย่างน้อย 33%	อย่างน้อย 15%	50%
4) ซีเอสอีบีเอส (CSEBS)	อย่างน้อย 52%	อย่างน้อย 21%	มากกว่า 50%

ตารางที่ 5.1 แสดงการเปรียบเทียบการเพิ่มประสิทธิภาพของการเรียงลำดับแบบขนานด้วยวิธีไบโทนิคทั้ง 4 แบบ โดยแบบที่สองซีอีบีเอส (CEBS) สามารถเพิ่มประสิทธิภาพในการติดต่อสื่อสารระหว่างหน่วยประมวลผลอย่างน้อย 44% ส่งผลให้เวลาที่ใช้ในการประมวลผลลดลง และพื้นที่ในหน่วยความจำลดลงประมาณ 20% (เมื่อเทียบกับแบบบีเอสเดิม) แบบที่ 3 เอสอีบีเอส (SEBS) เป็นแบบที่เน้นการใช้พื้นที่ในหน่วยความจำซึ่งลดลงประมาณ 50% นอกจากนี้แล้วยังเพิ่มเทคนิคการรวมข้อมูลในแต่ละหน่วยประมวลผล ส่งผลให้เวลาที่ใช้ในการประมวลผลลดลง และเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลลดลงอย่างน้อย 33% (เมื่อเทียบกับแบบบีเอสเดิม) และแบบสุดท้ายคือ ซีเอสอีบีเอส (CSEBS) เป็นแบบที่ให้ผลดีที่สุด เมื่อเทียบกับสามแบบแรกที่ได้กล่าวมาดังนี้ เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลลดลงอย่างน้อย 52% เวลาที่ใช้ในการประมวลผลลดลงอย่างน้อย 21% (เมื่อเทียบกับแบบบีเอสเดิม) และลดการใช้พื้นที่ในหน่วยความจำมากกว่า 50% (เมื่อเทียบกับแบบเอสอีบีเอส) ซึ่งเป็นวิธีที่มีประสิทธิภาพมากที่สุด แต่มีความยุ่งยากซับซ้อนมากในการพัฒนาโปรแกรม

5.2 แนวทางการพัฒนางานวิจัย

- ถ้านำโปรแกรมที่ได้พัฒนาขึ้นไปใช้กับระบบคลัสเตอร์แบบเนื้อผสม (Heterogeneous Cluster) จำเป็นต้องให้ความสำคัญในเรื่องของการสร้างสมดุลของงาน (Load Balancing) ในระบบ เนื่องจากคอมพิวเตอร์ที่เชื่อมต่ออยู่ในระบบนี้ สามารถมีองค์ประกอบภายในที่แตกต่างกันได้ ทำให้คอมพิวเตอร์แต่ละเครื่องมีความสามารถและประสิทธิภาพในการประมวลผลงานที่แตกต่างกัน

- นำเสนอขั้นตอนวิธีการใหม่สำหรับการเรียงลำดับข้อมูล ที่ง่ายและให้ผลดีกว่าแบบอื่นๆ ที่ได้มีการนำเสนอไว้แล้ว เช่น วิธีที่มีความซับซ้อนด้านเวลาน้อยกว่า $O(\log_2 N)^2$

เอกสารอ้างอิง

[1] Adler M., Byers J.W. and Karp R.M. "Parallel Sorting with Limited Bandwidth." **SIAM Journal Computer .** 2000. pp. 1997-2015.

[2] Batcher K.E. " Sorting networks and their applications." **Proceedings Spring Joint Computing Conference AFIPS.** Washington DC, 1968. pp.307-314.

[3] Bitton D., DeWitt D.J., Hsiao D.K. and Menon J. "A Taxonomy of Parallel Sorting." **ACM Computer Survey.** 1984. pp. 287-318.

[4] Brest J., Vreze A. and Zumer V. " A Sorting Algorithm on a PC Cluster." **Proceedings 2000 ACM Symposium on Applied Computing (SAC'00).** Como, Italy. pp. 710-715.

[5] Dimitrov R. and Skjellum A. "Software Architecture and Performance Comparison of MPI/Pro and MPICH." **International Conference on Computational Science.** 2003. pp. 307-315.

[6] **"Gigabit Ethernet."** [Online]. Available : http://www.gigabit-ethernet.org/. 2005.

[7] Gropp W. and Lusk E. "Why are PVM and MPI so different?." **PVM/MPI User's Group Meeting.** Cracow, Poland. 1997.

[8] Gropp W., Lusk E. and Skjellum A. **Using MPI: Portable Parallel Programming with the Massage Passing Interface.** Cambridge : MIT Press. 1994.

[9] Helman D.R. and JaJa J. "Sorting on Cluster of SMPs." **12th International Parallel Processing Symposium.** University of Maryland, College Park, MD, USA. 1997.

[10] Hwang K. **Advanced Computer Architecture: Parallelism, Scalability, Programmability.** New York : McGraw-Hill. 1993.

[11] Ionescu M.F. and Schauser K.E. "Optimizing Parallel Bitonic Sort." **Proceedings 11th Int'l Parallel Processing Symposium,** 1997. pp. 303-309.

[12] **"InfinBand."** [Online]. Available : http://www.infinibandta.org/. 2005

[13] Kim Y.C., Jeon M., Kim D. and Sohn A. "Communication-Efficient Bitonic Sort on a Distributed Memory Parallel Computer." **Int' l Conference Parallel and Distributed Systems.** 2001. pp. 165-170.

[14] Kumar V. **Introduction to parallel computing : design and analysis of algorithms.** Redwood City, CA : Benjamin/Cummings. 1994.

[15] Le T.T. and Huu T.C. **"Advances in Parallel Computing For the Year 2000 and Beyond."** [Online]. Available : http://www.vacets.org/vtic97/ttle.htm. 2005.

[16] Lee J.D. and Batcher K.E. "Minimizing Communication in the Bitonic Sort." **IEEE Transaction on Parallel and Distributed Systems.** 2000. pp. 459-473.

[17] Leighton T. "Tight Bounds on the Complexity of Parallel Sorting." **IEEE Transaction on Computers.** 1985. pp. 344-354.

[18] Message Passing Interface Forum. "MPI: A message passing interface standard." **Int'l Journal of Supercomputer Applications.** 1994.

[19] Miller R. **Algorithms sequential and parallel : a unified approach.** Upper Saddle River, NJ : Prentice Hall. 2000.

[20] **"Massive Parallel Processor: MPP."** [Online]. Available : www.buyya.com/cluster/LinuxClusters.ppt. 2005.

[21] **"Myrinet."** [Online]. Available : http://www.myri.com/myrinet/. 2005.

[22] Nakatani T., Huang S.T., Arden B.W. and Tripathi S.K. "K-Way Bitonic Sort." **IEEE Transaction Computers.** 1989. pp. 283-288.

[23] **"Parallel_Bitonic.c"** [Online]. Available : http://www.cs.panam.edu/. 2004.

[24] **"Quadrics."** [Online]. Available : http://www.quadrics.com/. 2005.

[25] Quinn J.M. **Parallel Programming in C with MPI and OpenMP,** International Edition 2003. Singapore : McGraw Hill. 2003.

[26] Quinn J.M. **Parallel computing : theory and practice.** 2nd ed. New York : McGraw-Hill. 1994.

[27] Spector D.H.M. **Building linux clusters.** Beijing : O'Reilly. 2000.

[28] Thompson C.D., and Kung H.T. "Sorting on a Mesh-Connected Parallel Computer." **Communication ACM.** 1997. pp. 263-271.

[29] ชิดชนก เหลือสินทรัพย์. **Analysis & Design of Algorithms.** กรุงเทพมหานคร : ซีเอ็ดยูเคชั่น. 2543.

[30] นพรัตน์ พันธุ์เสนา. **"การสร้างภาพเชิงปริมาตรบนระบบคลัสเตอริ่งโดยวิธีการแปลงเฉือนและบิด."** วิทยานิพนธ์วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์ บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2547.

[31] ภุชงค์ อุทโยภาส. **"แนะนำเทคโนโลยีระบบพีซีคลัสเตอร์."** [Online]. Available : http://www.cpe.ku.ac.th/. 2005.

ประวัติผู้เขียน

ชื่อ – สกุล	นางสาวจุรีพร บุญนิยม
วัน เดือน ปีเกิด	3 พฤษภาคม 2523
ที่อยู่	37/1 หมู่ 7 ตำบลตาก้อง อำเภอเมือง จังหวัดนครปฐม 73000
ประวัติการศึกษา	
2547	จบการศึกษาปริญญาวิทยาศาสตรบัณฑิต
	สาขาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยมหิดล
ประวัติการทำงาน	
มิถุนายน 2544	เจ้าหน้าที่ Administrator บริษัททรู
ตุลาคม 2544	เจ้าหน้าที่ Consult Internet มหาวิทยาลัยมหิดล