การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูบนระบบพีซีคลัสเตอร์

PARALLEL DATA COMPRESSION WITH BZIP2 ON A CLUSTER OF PCs

อัศวิน นิ่มกร
ASSAWIN NIMKORN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2549

ISBN 974-8308-10-2

การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูบนระบบพีซีคลัสเตอร์

PARALLEL DATA COMPRESSION WITH BZIP2 ON A CLUSTER OF PCs

อัศวิน นิ่มกร

ASSAWIN NIMKORN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2549

ISBN 974-8308-16-2

PARALLEL DATA COMPRESSION WITH BZIP2 ON A CLUSTER OF PCs

ASSAWIN NIMKORN

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

2006

ISBN 974-8308-16-2

หัวข้อวิทยานิพนธ์	การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู บนระบบพีซีคลัสเตอร์
นักศึกษา	นายอัศวิน นิ่มกร
รหัสประจำตัว	47063707
ปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา	วิทยาการคอมพิวเตอร์
พ.ศ.	2549
อาจารย์ที่ปรึกษาวิทยานิพนธ์	ผศ. ดร. จีรพร ศรีสวัสดิ์

บทคัดย่อ

งานวิจัยนี้นำเสนอการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูบนระบบพีซีคลัสเตอร์ทั้งหมด 3 วิธี คือ 1) วิธีเอ็มเอสบีซิพทู (MSBzip2) 2) วิธีเอดับเบิลยูบีซิพทู (AWBzip2) และ 3) วิธีเอดับเบิลยูดับเบิลยูบีซิพทู (AWWBzip2) โดยวิธีแรกเป็นวิธีที่ใช้แนวคิดการบีบอัดข้อมูลแบบขนานด้วยวิธีพีบีซิพทู (PBzip2) ที่มีผู้ศึกษาไว้บนเครื่องคอมพิวเตอร์แบบขนานที่ใช้หน่วยความจำร่วม (Shared-Memory Parallel Architectures) โดยมีหน่วยประมวลผลควบคุม ส่วนวิธีที่ 2 และ 3 เป็นวิธีที่เสนอแนวคิดใหม่โดยไม่มีหน่วยประมวลผลควบคุม เพื่อลดการติดต่อสื่อสารแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลควบคุมกับหน่วยประมวลผลบีบอัดข้อมูล ดังนั้นทุกหน่วยประมวลผลจะทำหน้าที่ แบ่งข้อมูล บีบอัดข้อมูล และเขียนข้อมูล ส่วนวิธีที่ 3 จะต่างจากวิธีที่ 2 ที่มีการกำหนดน้ำหนักของการแบ่งบล็อกข้อมูลย่อย เพื่อให้บล็อกข้อมูลย่อยมีขนาดต่างกัน ทำให้แต่ละหน่วยประมวลผลใช้งานอุปกรณ์อินพุทเอาท์พุทศูนย์กลาง (Centralized I/O) ไม่พร้อมกัน จากผลการทดลองบนระบบพีซีคลัสเตอร์ที่มีขนาด 1 ถึง 5 หน่วยประมวลผล พบว่า วิธีเอดับเบิลยูดับเบิลยูพีบีซิพทู (AWWBzip2) และวิธีเอดับเบิลยูบีซิพทู (AWBzip2) มีสมรรถนะดีกว่าวิธีเอ็มเอสบีซิพทู (MSBzip2) อย่างน้อย 19.91% และ 18.75% ตามลำดับ

Thesis	Parallel Data Compression with Bzip2 on a Cluster of PCs
Student	Mr.Assawin Nimkorn
Student ID	47063707
Degree	Master of Science
Program	Computer Science
Year	2006
Thesis Advisor	Asst. Prof. Dr. Jeeraporn Srisawat

ABSTRACT

This research proposes the parallel data compression with Bzip2 on a cluster of PCs. In particular, three strategies are introduced: 1) The MSBzip2 strategy, 2) The AWBzip2 strategy and 3) The AWWBzip2 strategy. The first method (MSBzip2) adapts the idea from the existing PBzip2, which is performed on shared-memory parallel architectures, controlled by the master processor. The last two methods (AWBzip2 and AWWBzip2) are presented with no master processor in order to reduce the communication time between the master processor and the slave processors. Therefore, in this case all processors will split the data into a number of blocks of the same size, compress the block of data and write the block of compressed data in parallel. The major difference of the last one (AWWBzip2) is that all processors will split the data into a number of blocks of the different weighted sizes in order to concurrent access the centralized input/output (I/O) device (or first-finish first-write). On the cluster system of size up to 5 processors (Ps), the experimental results showed that the AWWBzip2 strategy and the AWBzip2 strategy improved performance over the MSBzip2 strategy at least 19.91%.and 18.75%, respectively.

กิตติกรรมประกาศ

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

ขอขอบพระคุณ รศ.ดร.วีระ บุญจริง ผศ.ดร.ศรัณย์ อินทโกสุม และดร.เฉลิมศักดิ์ เลิศวงศ์เสถียร คณะกรรมการสอบหัวข้อ และโครงร่างวิทยานิพนธ์ ที่กรุณาให้คำแนะนำตลอดจนข้อชี้แนะจนในที่สุดทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลงได้

ขอขอบพระคุณโครงการส่งเสริมการผลิตครูที่มีความสามารถพิเศษทางวิทยาศาสตร์และคณิตศาสตร์ (สควค.) และโครงการพัฒนาครูแกนนำในสาขาคณิตศาสตร์ วิทยาศาสตร์และเทคโนโลยีและสิ่งแวดล้อม (Project Guiding Light) ที่ให้ทุนการศึกษาตลอดมา

ขอขอบพระคุณบิดา มารดา และพี่ๆ ที่สนันสนุนให้ได้เรียนในระดับที่ได้ตั้งใจ อีกทั้งยังได้ดูแลเรื่องค่าใช้จ่ายต่างๆระหว่างศึกษาเป็นอย่างดีอีกด้วย

ขอขอบคุณ นายนพรัตน์ พันธุ์เสนา นายอภินันท์ เสริมศรี และเพื่อนๆทุกคนที่ให้คำปรึกษา และช่วยอำนวยความสะดวกในด้านต่างๆ

สำหรับคุณงามความดีและประโยชน์อันใดที่เกิดขึ้นจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดา มารดา อาจารย์ทุกท่านซึ่งเป็นที่เคารพรักยิ่ง ตลอดจนญาติพี่น้อง และเพื่อนๆทุกคน

อัศวิน นิ่มกร

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญตาราง (ต่อ)

สารบัญรูป

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

จากอดีตจนถึงปัจจุบันโลกของเรามีการพัฒนาทางด้านเทคโนโลยีไปอย่างมาก เริ่มตั้งแต่การเปลี่ยนแปลงจากยุคเกษตรกรรมไปสู่ยุคอุตสาหกรรม จนมาถึงปัจจุบัน คือยุคสารสนเทศ ซึ่งเป็นยุคที่ข้อมูลสารสนเทศเป็นสิ่งที่มีความจำเป็นกับชีวิตของมนุษย์อย่างขาดไม่ได้ และยังเป็นยุคที่มีการพัฒนาเทคโนโลยีสารสนเทศอย่างต่อเนื่อง ดังจะเห็นได้จาก การพัฒนาเทคโนโลยีคอมพิวเตอร์ ที่มีการผลิตไมโครโปรเซสเซอร์ (Microprocessor) ขึ้นมาใช้ ซึ่งทำให้คอมพิวเตอร์และอุปกรณ์มีขนาดเล็กลง แต่มีสมรรถนะสูงขึ้น มีการพัฒนาเทคโนโลยีเครือข่าย (Network Technology) ที่ช่วยให้สามารถติดต่อสื่อสารกันได้สะดวกและรวดเร็วขึ้น การพัฒนาเทคโนโลยีด้านการควบคุม ที่ทำให้เราสามารถควบคุมอุปกรณ์ต่างๆได้จากระยะไกล เป็นต้น ในขณะที่เทคโนโลยีสารสนเทศมีพัฒนาการเจริญก้าวหน้า มีการนำคอมพิวเตอร์เข้ามามีส่วนร่วมในการจัดการและจัดเก็บข้อมูล ซึ่งการจัดเก็บข้อมูลมีการจัดเก็บได้หลายรูปแบบ เช่น จัดเก็บในแฟ้มข้อมูล จัดเก็บในฐานข้อมูล โดยอุปกรณ์ที่ใช้ในการจัดเก็บข้อมูลอาจจะเป็น ฮาร์ดดิสก์ (Harddisk) หรือ เทป (Tape) แต่อุปกรณ์จัดเก็บข้อมูลของคอมพิวเตอร์ก็ยังมีข้อจำกัดระดับหนึ่ง รวมไปถึงแนวโน้มของข้อมูลสารสนเทศที่เพิ่มขึ้นอยู่ตลอดเวลา ส่งผลให้อุปกรณ์จัดเก็บข้อมูลไม่เพียงพอต่อความต้องการ วิธีการหนึ่งที่สามารถลดปัญหานี้ลงได้คือ การบีบอัดข้อมูล

การบีบอัดข้อมูล (Data Compression) [2] [7] [10] คือ การลดขนาดข้อมูลเพื่อให้ข้อมูลมีขนาดเล็กลง ทำให้สามารถเก็บข้อมูลได้มากขึ้นในขณะที่พื้นที่เก็บข้อมูลเท่าเดิม ซึ่งเทคโนโลยีนี้ถูกนำไปใช้งานในด้านต่างๆเช่น การรักษาพื้นที่ว่างสำหรับดิสก์ การสำรองข้อมูล รวมถึงรูปภาพต่างๆที่อยู่ในเว็บเพจ จะถูกบีบอัดข้อมูล ซึ่งส่วนใหญ่อยู่ในรูปแบบ JPEG หรือ GIF อีกทั้งในหลายๆระบบแฟ้มข้อมูลจะมีการบีบอัดแฟ้มข้อมูลอัตโนมัติเมื่อมีการบันทึกข้อมูล เทคโนโลยีการบีบอัดข้อมูลนี้ไม่ได้ถูกใช้แค่เพียงแต่ในเครื่องคอมพิวเตอร์เท่านั้น แต่ยังถูกใช้ในโมเด็ม เราท์เตอร์ กล้องดิจิตอล โทรสาร และอื่นๆ อีกมากมาย ดังนั้น การศึกษาวิจัยเกี่ยวกับการบีบอัดข้อมูล จึงเป็นที่สนใจอย่างแพร่หลาย การบีบอัดข้อมูลด้วยวิธีบีซิพทูเป็นการบีบอัดข้อมูลแบบอนุกรม (Sequential Compression) วิธีการหนึ่งที่ได้รับความนิยมและมีประสิทธิภาพ อย่างไรก็ตามการบีบอัดข้อมูลแบบอนุกรมบนเครื่องคอมพิวเตอร์ที่มีหน่วยประมวลผลเดียว (Sequential Computer) กับข้อมูลที่มีขนาดใหญ่ต้องใช้เวลาในการบีบอัดข้อมูลเป็นจำนวนมาก จึงเกิดแนวคิดที่จะนำเทคนิคของการประมวลผลแบบขนาน (Parallel Computing) บนเครื่องคอมพิวเตอร์ที่มีหลายหน่วยประมวล (Parallel Computer)[11] มาใช้ เพื่อลดเวลาในการบีบอัดข้อมูล

งานวิจัยนี้ได้นำเสนอการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู (BZIP2) [3] ที่มีสมรรถนะ (Performance) มากขึ้นบนระบบพีซีคลัสเตอร์ (Cluster of PCs) ซึ่งการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูนี้สามารถแบ่งเวลาที่ใช้ในการทำงานเป็น 2 ส่วน เช่นเดียวกับการประมวลผลแบบขนานโดยทั่วไป คือ เวลาที่ใช้ในการประมวลผลแบบขนาน (Parallel Computation Time) และเวลาที่ใช้ในการติดต่อสื่อสารแบบขนาน (Parallel Communication Time) ในงานวิจัยนี้จะทำการเพิ่มสมรรถนะของเวลาที่ใช้ในการติดต่อสื่อสารแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวล และลดการใช้งาน อุปกรณ์อินพุทเอาท์พุทศูนย์กลาง (Centralized I/O) พร้อมกันของแต่ละหน่วยประมวลผล โดยแสดงการเปรียบเทียบ ประเมินสมรรถนะจากเวลาที่ใช้ในการบีบอัดข้อมูล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) ซึ่งจะพัฒนาโปรแกรมการบีบอัดข้อมูลด้วยภาษาซี (C Language) และมาตรฐานภาษาเอ็มพีไอ (MPI Standard) บนระบบพีซีคลัสเตอร์ เพื่อใช้ในการทดลอง และหาผลสรุปของงานวิจัย

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

วิทยานิพนธ์ฉบับนี้มุ่งหวังเพื่อศึกษาการบีบอัดข้อมูลแบบขนานด้วยวิธีพีบีซิพทู(PBzip2) ที่มีผู้ศึกษาไว้บนเครื่องคอมพิวเตอร์แบบขนานที่ใช้หน่วยความจำร่วม (Shared-Memory Parallel Architectures) ซึ่งการบีบอัดข้อมูลแบบขนานด้วยวิธีพีบีซิพทูดังที่กล่าวมาแล้วข้างต้นเมื่อนำมาพัฒนาบนระบบพีซีคลัสเตอร์แล้วนั้นจะทำให้มีการติดต่อสื่อสารระหว่างหน่วยประมวลผลมาก ทำให้สมรรถนะของการบีบอัดข้อมูลต่ำลง ดังนั้นในวิทยานิพนธ์นี้จึงเสนอวิธีลดการติดต่อสื่อสารแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล และวิธีลดการใช้งานอุปกรณ์อินพุทเอาท์พุทศูนย์กลางพร้อมกัน ของการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู ซึ่งสามารถช่วยให้การบีบอัดข้อมูลมีสมรรถนะดีขึ้น

1.3 สมมติฐานของการศึกษา

1) การตัดหน่วยประมวลผลควบคุมออกเพื่อลดการติดต่อระหว่างหน่วยประมวลผลทำให้เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลลดลง มีผลทำให้การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูที่นำเสนอมีสมรรถนะเพิ่มขึ้น

2) การกำหนดน้ำหนักของการแบ่งบล็อกข้อมูลย่อย เพื่อให้บล็อกข้อมูลย่อยมีขนาดต่างกัน ทำให้แต่ละหน่วยประมวลผลใช้งานอุปกรณ์อินพุทเอาท์พุทศูนย์กลางไม่พร้อมกัน มีผลทำให้การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูที่นำเสนอมีสมรรถนะเพิ่มขึ้น

1.4 ขอบเขตการวิจัย

ขอบเขตของวิทยานิพนธ์ฉบับนี้มีดังต่อไปนี้

1) การบีบอัดข้อมูลในงานวิจัยนี้ เป็นการบีบอัดข้อมูลแบบที่ไม่มีการสูญเสียข้อมูล (Lossless Compression)

2) งานวิจัยนี้ครอบคลุมเฉพาะส่วนการบีบอัดข้อมูลเท่านั้น จะไม่ครอบคลุมถึงส่วนของการคลายข้อมูล

3) ข้อมูลที่ใช้สำหรับทดลองในงานวิจัยนี้ เป็นชุดข้อมูลที่นำมาจากชุดข้อมูลกูเทนเบริก (Gutenberg Corpus)

4) ข้อมูลที่ได้จากการบีบอัดข้อมูลแบบขนานในงานวิจัยนี้ สามารถคลายข้อมูลกลับด้วยวิธีแบบอนุกรม

5) สมรรถนะของการบีบอัดข้อมูลวัดจากเวลาที่ใช้ในการบีบอัดข้อมูล อัตราการเพิ่มขึ้นของความเร็ว และประสิทธิภาพ

6) งานวิจัยนี้จะเปรียบเทียบสมรรถนะของการบีบอัดข้อมูลข้อมูลแบบขนานด้วยวิธีบีซิพทูแบบใหม่ที่เสนอ กับการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูแบบเก่า ซึ่งพัฒนาบนระบบพีซีคลัสเตอร์

1.5 ขั้นตอนการศึกษาและดำเนินงานวิจัย

งานวิจัยนี้มีขั้นตอนการศึกษาและดำเนินงานวิจัยดังนี้

1) ศึกษางานวิจัยที่เกี่ยวข้องกับการบีบอัดข้อมูล

2) ศึกษาขั้นตอนวิธี (Algorithm) การบีบอัดข้อมูลด้วยวิธีบีซิพทู

3) ทำการตั้งสมมติฐาน ดังนี้ 1) ลดการติดต่อระหว่างหน่วยประมวลผลทำให้เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลลดลง 2) ลดการใช้อินพุทเอาท์พุทศูนย์กลางพร้อมกัน โดยให้บล็อกข้อมูลย่อยมีขนาดต่างกัน ทำให้แต่ละหน่วยประมวลผลใช้งานอุปกรณ์อินพุทเอาท์พุทศูนย์กลางไม่พร้อมกัน มีผลทำให้การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูที่นำเสนอมีสมรรถนะเพิ่มขึ้น

4) นำเสนอวิธีการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูที่มีสมรรถนะเพิ่มขึ้นดังที่ได้ตั้งสมมติฐาน

5) พัฒนาโปรแกรมบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู โดยใช้ภาษาซีบนระบบพีซีคลัสเตอร์ที่ใช้ระบบปฏิบัติการลีนุกซ์ (Linux) โดยใช้มาตรฐานภาษาเอ็มพีไอที่สนับสนุนการโปรแกรมแบบขนาน

6) วิเคราะห์ผลการทดลองโดยการเปรียบเทียบสมรรถนะของการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูนี้จากเวลาที่ใช้ในการบีบอัดข้อมูล อัตราการเพิ่มของความเร็ว และประสิทธิภาพ

7) สรุปผลการทดลองพร้อมเสนอแนวทางการพัฒนางานวิจัย

8) เขียนวิทยานิพนธ์

1.6 ประโยชน์ที่คาดว่าจะได้รับ

งานวิจัยนี้มีประโยชน์ที่คาดว่าจะได้รับจากการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูบนระบบพีซีคลัสเตอร์ ดังนี้

1) ลดเวลาที่ใช้ในการบีบอัดข้อมูล

2) นำมาใช้เป็นแนวทางในการพัฒนาระบบบีบอัดข้อมูลด้วยวิธีอื่น

3) นำมาใช้เป็นแนวทางในการพัฒนาโปรแกรมแบบขนานบนระบบพีซีคลัสเตอร์ เพื่อให้สามารถใช้ได้สะดวกและเข้าใจได้ง่ายมากขึ้น

4) นำมาใช้เป็นแนวทางในการพัฒนาระบบคอมพิวเตอร์แบบขนานชนิดคลัสเตอร์สำหรับหน่วยงานหรือสถานศึกษาที่ต้องการมีระบบคลัสเตอร์ของตนเอง

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในหัวข้อนี้จะกล่าวถึงทฤษฎีพื้นฐานต่างๆ ที่เกี่ยวข้องในการวิจัย ประกอบด้วย การบีบอัดข้อมูลแบบต่างๆ การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู และการวัดสมรรถนะ (Performance Measurement) ซึ่งเนื้อหาทั้งหมดนี้จำเป็นสำหรับการศึกษาวิจัย เพื่อใช้ในการทดลองและประเมินสมรรถนะของการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู

2.1 การบีบอัดข้อมูล (Data Compression)

การบีบอัดข้อมูล [2] [7] [10] คือ การลดขนาดข้อมูลเพื่อให้ข้อมูลมีขนาดเล็กลง ทำให้สามารถเก็บข้อมูลได้มากขึ้นในขณะที่พื้นที่เก็บข้อมูลเท่าเดิม ซึ่งเทคโนโลยีนี้ถูกนำไปใช้งานในด้านต่างๆเช่น การรักษาพื้นที่ว่างสำหรับดิสก์ การสำรองข้อมูล รวมถึงการลดเวลาที่ใช้ในการสื่อสารหรือเวลาที่ใช้ในการถ่ายโอนข้อมูล สามารถแบ่งออกเป็น 2 แบบ คือ

1) การบีบอัดข้อมูลแบบที่มีการสูญเสียข้อมูล (Lossy Compression) เมื่อบีบอัดข้อมูลแล้วคลายข้อมูลกลับจะมีความเพี้ยนและสูญหายของข้อมูลไปจากข้อมูลต้นฉบับด้วย ซึ่งการยอมสูญเสียข้อมูลบางส่วนนี้ก็เพื่อแลกกับการบีบอัดข้อมูลให้เล็กลงได้มากขึ้น [10] ข้อมูลประเภทรูปภาพและข้อมูลเสียงจัดเป็นข้อมูลที่เหมาะสำหรับการบีบอัดข้อมูลประเภทนี้

2) การบีบอัดข้อมูลแบบที่ไม่มีการสูญเสียข้อมูล (Lossless Compression) เมื่อบีบอัดข้อมูลแล้วคลายข้อมูลกลับจะต้องได้ข้อมูลเหมือนเดิมครบถ้วนทุกประการ ด้วยเหตุนี้ การบีบอัดข้อมูลแบบนี้จึงมักถูกใช้กับข้อมูลประเภทที่ข้อมูลจะสูญเสียไม่ได้ เช่น ข้อมูลประเภทข้อความหรือแฟ้มข้อมูลโปรแกรม เป็นต้น การบีบอัดข้อมูลรูปภาพก็ใช้แนวทางการบีบอัดแบบนี้โดยอาศัยหลักการลดการซ้ำซ้อนของข้อมูล (Redundancy Reduction) และการใช้จำนวนบิตที่เก็บค่าข้อมูลไม่คงที่ (Variable-Length Coding) เช่น การเข้ารหัสแบบฮัฟแมน (Huffman Coding) [4] และที่ใช้กันแพร่หลายก็คือการบีบอัดข้อมูลในรูปแบบแฟ้มข้อมูลรูปภาพของ PCX, GIF และ BMP ซึ่งให้อัตราการบีบอัดระหว่าง 10% ถึง 90 % สำหรับรูปภาพกราฟิก [10]

2.1.1 การเข้ารหัสแบบฮัฟแมน (Huffman Coding)

การเข้ารหัสแบบฮัฟแมน [4] เป็นการใช้รหัสที่สั้นกว่าแทนสัญลักษณ์ที่เกิดขึ้นบ่อย โดยจะใช้ต้นไม้สองทางในการสร้างรหัสของสัญลักษณ์แต่ละตัวกระจายไปกับข้อมูลของโหนดใบซึ่งเชื่อมอยู่กับต้นไม้สองทาง (Binary Tree) แต่ละโหนดจะมีน้ำหนักกำหนดอยู่ ซึ่งก็คือ ความถี่ หรือความน่าจะเป็นของการปรากฏของสัญลักษณ์นั้น

วิธีการสร้างต้นไม้สามารถทำได้ตามขั้นตอนดังนี้ [10]

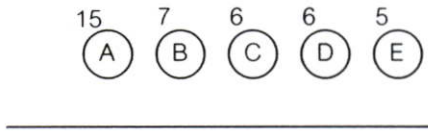
1) กำหนดให้ทุกๆสัญลักษณ์เป็นโหนดใดๆ
2) หา 2 โหนดใดๆ ที่มีน้ำหนักน้อยที่สุด
3) สร้างโหนดแม่สำหรับโหนดสองโหนดนี้ โดยมีน้ำหนักเท่ากับผลรวมของน้ำหนักของโหนดลูก
4) กำหนดให้โหนดแม่นี้เป็นโหนดใดๆ และนำโหนดลูกออกจากโหนดใดๆ
5) โหนดลูกโหนดหนึ่งจะถูกกำหนดให้มีทางผ่านจากโหนดแม่เมื่อทำการถอดรหัสด้วยบิต 0 และอีกโหนดจะถูกกำหนดด้วย บิต 1
6) กลับไปข้อ 1) จนกว่าจะเหลือโหนดใดๆ หนึ่งโหนด ซึ่งโหนดนี้จะถูกกำหนดให้เป็นรากของต้นไม้

ตัวอย่างที่ 2.1 กำหนดให้มีข้อมูลต่างๆดังนี้

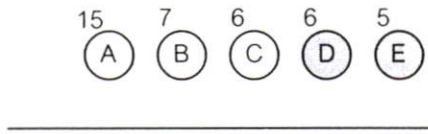
ตารางที่ 2.1 แสดงสัญลักษณ์และจำนวนของการปรากฏของข้อมูลสัญลักษณ์ทั้งหมด 5 สัญลักษณ์ คือ A, B, C, D และ E

Symbol	Count
A	15
B	7
C	6
D	6
E	5

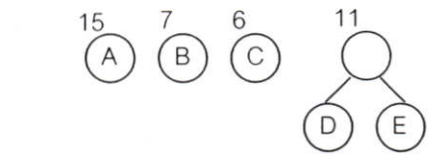
สัญลักษณ์ทั้ง 5 สัญลักษณ์นี้จะเป็นโหนดใบของต้นไม้ เมื่อกระบวนการเริ่มขึ้น โหนดทั้งหมดจะถูกกำหนดให้เป็นโหนดใดๆ ในรายการ (List)



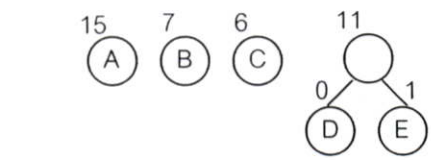
ขั้นแรก หา 2 โหนดที่มีน้ำหนักน้อยสุด คือ D และ E ซึ่งมีน้ำหนัก 6 และ 5



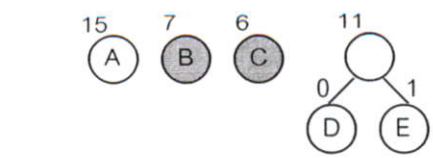
รวมโหนดทั้งสองเข้ากับโหนดแม่ และกำหนดให้โหนดแม่มีน้ำหนักเป็น 11 และนำโหนด D และ E ออกจากโหนดใดๆ ในรายการ



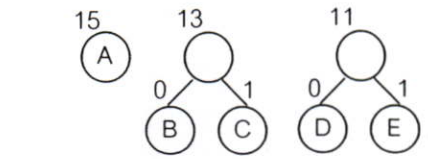
เมื่อขั้นตอนแรกเสร็จสิ้น เราจะรู้ว่าอะไรคือบิตที่น้อยที่สุดที่แสดงค่าของรหัส D และ E โดย D จะถูกแยกออกจากโหนดแม่ด้วย 0 (ซ้าย) และ E จะถูกแยกด้วย 1 (ขวา)



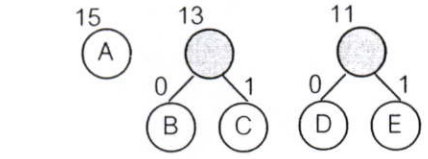
ขั้นต่อมา 2 โหนดที่มีค่าน้ำหนักน้อยที่สุดในโหนดใดๆ คือ โหนด B และ C



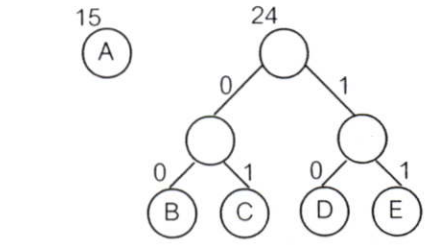
ดังนั้น ต้องเพิ่มโหนดแม่ใหม่และกำหนดให้มีน้ำหนักเป็น 13 และนำโหนด B และ C ออกจากโหนดใดๆ ในรายการ



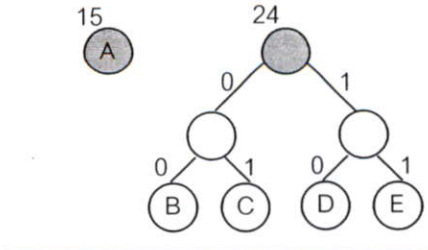
ขั้นต่อมา 2 โหนดที่มีค่าน้ำหนักน้อยที่สุด คือ โหนดแม่ของ B/C และ D/E



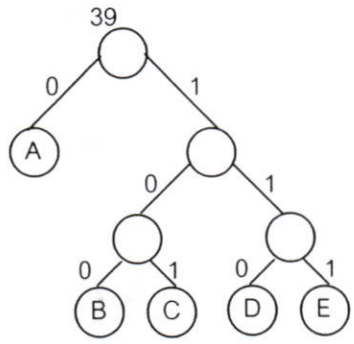
ดังนั้นต้องเพิ่มโหนดแม่ใหม่และกำหนดให้มีน้ำหนักเป็น 24 และนำโหนดลูกออกจากโหนดใดๆ ในรายการ



ในขั้นตอนสุดท้ายจะเหลือเพียง 2 โหนดใดๆ ในรายการ คือ โหนดแม่ที่มีน้ำหนัก 24 และโหนด A



สร้างโหนดแม่ใหม่ให้มีน้ำหนักเป็น 39 และนำโหนดลูกออกจากโหนดใดๆ ในรายการ ตอนนี้เหลือโหนดแม่เพียงโหนดเดียว นั้นหมายถึงการสร้างต้นไม้เสร็จสิ้น



ตัวอย่างโครงร่างโปรแกรมการเข้ารหัสแบบฮัฟแมน (Huffman Pseudocode)

```
Huffman (C)
    n = the size of C
    insert all the elements of C into Q,
    using the value of the node as the priority
    for i in 1..n-1 do
        z = a new tree node
        x = Extract-Minimum (Q)
        y = Extract-Minimum (Q)
        left node of z = x
        right node of z = y
        f[z] = f[x] + f[y]
        Insert (Q, z)
    end for
return Extract-Minimum (Q) as the complete tree
```

รูปที่ 2.1 ตัวอย่างโครงร่างโปรแกรมการเข้ารหัสแบบฮัฟแมน (Huffman Pseudocode)

ในการกำหนดรหัสให้กับสัญลักษณ์นั้น ทำโดยเดินจากรากของต้นไม้ไปตามโหนดใบของต้นไม้ฮัฟแมน สะสมบิตใหม่ถ้าเดินผ่านโหนดแม่แต่ละโหนด ด้วยวิธีนี้จะได้โครงสร้างของรหัสแสดงได้ดังนี้

ตารางที่ 2.2 แสดงรหัสแทนสัญลักษณ์ของการเข้ารหัสแบบฮัฟแมน

The Huffman Code Table	
A	0
B	100
C	101
D	110
E	111

จะเห็นว่ารหัสจะมีคุณสมบัติคือรหัสนำหน้าจะมีเพียงหนึ่งเดียว (Prefix code) ไม่ซ้ำกัน หมายความว่า ไม่มีรหัสใดเป็นรหัสนำหน้าของรหัสตัวอื่นๆ โดยรหัสฮัฟแมนสามารถถอดรหัสได้อย่างไม่กำกวมเมื่อรหัสนั้นมาในรูปแบบสายอักขระ สัญลักษณ์ที่มีความถี่หรือค่าความน่าจะเป็นสูงที่สุด คือ A จะถูกกำหนดด้วยจำนวนบิตที่น้อยที่สุด และสัญลักษณ์ที่มีความถี่หรือ ค่าความน่าจะเป็นน้อยที่สุด คือ E จะถูกกำหนดด้วยจำนวนบิตที่มากที่สุด

2.1.2 เบอร์โรวส์-วีเลอร์ (Burrows-Wheeler Transform)

วิธีเบอร์โรวส์-วีเลอร์ [1] เป็นวิธีที่ทำให้อัตราการบีบอัดข้อมูลมีประสิทธิภาพเพิ่มขึ้น โดยลักษณะการแปลงข้อมูลวิธีนี้ จะแบ่งข้อมูลต้นกำเนิดออกเป็นกลุ่ม หรือบล็อก (Block) ที่มีขนาดเท่าๆกัน แล้วทำการเปลี่ยนลำดับ ของอักษรภายในบล็อกดังกล่าว เพื่อให้อักษรที่เหมือนกันเรียงต่อกัน เช่น bacba ถูกเปลี่ยนเป็น bbcaa ซึ่งการเข้ารหัสและถอดรหัสด้วยวิธีเบอร์โรวส์-วีเลอร์มีดังต่อไปนี้

1) การเข้ารหัสด้วยวิธีเบอร์โรวส์-วีเลอร์

กำหนดให้ S เป็นบล็อกข้อมูลขนาด N ตัวอักษร โดยที่แต่ละตัวอักษรในบล็อกแทนด้วยสัญลักษณ์ B[0], B[1], B[2], ..., B[N-2], B[N-1] ตามลำดับ และกำหนดให้เมตริก M เป็นเมตริกจัตุรัสขนาด N x N ซึ่งมีรูปแบบการเก็บอักษรในเมตริกดังนี้

$$\begin{pmatrix} B[0] & B[1] & B[2] & \dots & B[N-3] & B[N-2] & B[N-1] \\ B[1] & B[2] & B[3] & \dots & B[N-2] & B[N-1] & B[0] \\ B[2] & B[3] & B[4] & \dots & B[N-1] & B[0] & B[1] \\ & \dots & & & \dots & & \\ & \dots & & & \dots & & \\ B[N-2] & B[N-1] & B[0] & \dots & B[N-5] & B[N-4] & B[N-3] \\ B[N-1] & B[0] & B[1] & \dots & B[N-4] & B[N-3] & B[N-2] \end{pmatrix}_{N \times N}$$

แถวแรกของเมตริก M ได้จากการนำเอาอักษรแต่ละตัวในบล็อก S มาวางแต่ละตำแหน่งในแถวตามลำดับ

แถวที่สองของเมตริก M ได้จากการย้ายตำแหน่งตัวอักษรที่อยู่หน้าสุดของแถวแรกไปไว้ในตำแหน่งหลังสุด แล้วเลื่อนอักษรทุกตัวที่เหลือในแถวไปทางซ้าย 1 ตำแหน่ง

แถวที่สามของเมตริก M ได้จากการย้ายอักษรที่อยู่หน้าสุดของแถวทีสองไปไว้ในตำแหน่งหลังสุด แล้วเลื่อนอักษรทุกตัวที่เหลือในแถวไปทางซ้าย 1 ตำแหน่ง ในทำนองเดียวกันแถวที่สี่จนถึงแถวที่ N จะทำลักษณะเดียวกันนี้ไปจนครบ ก็จะได้เมตริก M ขนาด N x N ดังแสดงในตัวอย่างที่ 2.2

ตัวอย่างที่ 2.2 สมมติว่าบล็อก S ในข้อมูลต้นกำเนิดมีอักษรเรียงต่อกันคือ bacba

จากโจทย์ S = bacba, N = 5

เลขแถว	เมตริก M
0	b a c b a
1	a c b a b
2	c b a b a
3	b a b a c
4	a b a c b

$$M = \begin{pmatrix} b & a & c & b & a \\ a & c & b & a & b \\ c & b & a & b & a \\ b & a & b & a & c \\ a & b & a & c & b \end{pmatrix}$$

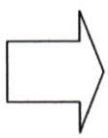
ข้อมูลแต่ละแถวในเมตริก M จะถูกมองเสมือนว่าเป็นกลุ่มข้อมูลแต่ละกลุ่ม เช่น ในตัวอย่าง 2.2 แถวทั้ง 5 แถว ในเมตริก M จะถูกมองเสมือนว่าเป็นกลุ่มข้อมูล 5 กลุ่ม โดยมีเลขแถวกำกับตั้งแต่ 0 ถึง 4 (N = 5) คือ bacba, acbab, cbaba, babac และ abacb ตามลำดับ ซึ่งในขั้นตอนต่อไปจะนำเอากลุ่มข้อมูลเหล่านี้ มาเรียงลำดับ (Sorting) ในลักษณะเดียวกับการเรียงคำศัพท์ในพจนานุกรม จากนั้นจะนำกลุ่มข้อมูลทั้งหมดที่เรียงลำดับแล้ว ไปใส่ไว้ในเมตริกใหม่ที่สร้างขึ้น โดยเริ่มใส่ข้อมูลที่ละแถวจากแถวบนมายังแถวล่างสุด และเรียกเมตริกใหม่ที่เกิดขึ้นดังกล่าวว่า เมตริก M' ดังแสดงในตัวอย่างที่ 2.3

ตัวอย่างที่ 2.3 สร้างเมตริก M' จากเมตริก M จากตัวอย่างที่ 2.2

เมตริก M มีกลุ่มข้อมูล 5 กลุ่ม และมีลำดับดังนี้ bacba, acbab, cbaba, babac และ abacb

เมื่อเรียงลำดับกลุ่มข้อมูลทั้ง 5 กลุ่มใหม่แล้ว ลำดับใหม่ของกลุ่มข้อมูลที่เกิดขึ้นจะเป็นดังนี้ abacb, acbab, babac, bacba และ cbaba

เลขแถว	เมตริก M
0	b a c b a
1	a c b a b
2	c b a b a
3	b a b a c
4	a b a c b

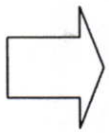


เลขแถว	เมตริก M'
4	a b a c b
1	a c b a b
3	b a b a c
0	b a c b a
2	c b a b a

ให้ L เป็นข้อมูลสดมภ์สุดท้ายของเมตริก M' ผลลัพธ์ของการเข้ารหัสคือ คู่ลำดับ (L, I) โดยที่ I คือ เลขแถวตัวแรกที่พบจากการนำเอาข้อมูลหลักสุดท้ายของแถวแรกในเมตริก M' ไปค้นหาข้อมูลที่มีค่าตรงกับหลักสุดท้ายของเมตริก M โดยจะเรียก I ว่าดัชนี ดังตัวอย่างที่ 2.4

ตัวอย่างที่ 2.4 หาผลลัพธ์การเข้ารหัสจากตัวอย่างที่ 2.3

เลขแถว	เมตริก M
0	b a c b a
1	a c b a **b**
2	c b a b a
3	b a b a c
4	a b a c b



เลขแถว	เมตริก M'
4	a b a c **b**
1	a c b a **b**
3	b a b a **c**
0	b a c b **a**
2	c b a b **a**

จากตัวอย่างที่ 2.4 ข้อมูลในหลักสุดท้ายของแถวแรกในเมตริก M' คือ b ('b' ที่ขีดเส้นใต้ในเมตริก M') ซึ่งเมื่อนำไปค้นในสดมภ์สุดท้ายของเมตริก M พบว่า 'b' ตัวแรกที่เจอ ('b' ตัวหนาในเมตริก M) มีค่าเลขแถวเท่ากับ 1 ฉะนั้น I มีค่าเป็น 1 ผลลัพธ์ของข้อมูลที่ผ่านการเข้ารหัสแล้วจึงเป็น (bbcaa, 1)

2) การถอดรหัสด้วยวิธีเบอร์โรวส์-วีเลอร์

จากการเข้ารหัสในหัวข้อที่ผ่านมา ข้อมูลสดมภ์สุดท้ายของเมตริก M' (คู่ลำดับแรกของผลลัพธ์) จะมีจำนวนตัวอักษรเท่ากับข้อมูลสดมภ์สุดท้ายของเมตริก M' เสมอ เช่นในตัวอย่างที่ 2.4 จะเห็นว่าสดมภ์แรกและสดมภ์สุดท้ายของเมตริก M' มีตัวอักษรเท่ากันคือ มี'a' 2 ตัว 'b' 2 ตัว และ 'c' 1 ตัว แต่จะแตกต่างที่ข้อมูลในสดมภ์แรกของเมตริกซ์ M' จะมีการเรียงลำดับตามตัวอักษร ฉะนั้นเมื่อทราบผลลัพธ์ซึ่งเป็นสดมภ์สุดท้ายของเมตริก M' ก็จะสามารถหาค่าของสดมภ์แรกของเมตริก M' ได้ทันที

ก่อนการถอดรหัสจะต้องมีข้อมูล 3 ส่วน ได้แก่ สดมภ์สุดท้ายของเมตริก M' (คู่อันดับแรกของผลลัพธ์) ดัชนี I และเวกเตอร์ V[N] ซึ่งเวกเตอร์ V คือ เมตริกขนาด N x 1 และมีค่าตัวเลขอยู่ในช่วง 0 ถึง N-1 เมื่อ N คือ ขนาดของบล็อกข้อมูลของสดมภ์สุดท้ายของเมตริก M' ซึ่งข้อมูล 2 ส่วนแรก คือ คู่อันดับของผลลัพธ์ที่ได้จากการเข้ารหัส (ไม่ต้องคำนวณหา) แต่เวกเตอร์ V จะสามารถหาได้จากการคำนวณโดยมีขั้นตอนดังต่อไปนี้

ในการสร้างเวกเตอร์ V จะนำเอาอักษรทีละตัวในสดมภ์สุดท้ายของเมตริก M' จากบนลงล่างมาค้นหาอักษรที่มีค่าเดียวกันในสดมภ์แรกของเมตริก M' เมื่อพบตัวอักษรตัวแรกที่ตรงกันก็จะกำหนดค่าตัวเลข ณ แถวเดียวกันกับการพบตัวอักษรที่ตรงกันในสดมภ์แรกของเมตริก M' โดยให้ตัวแรกที่พบจะมีค่าเป็น 0 ตัวที่สองจะมีค่าเป็น 1 ทำเช่นนี้ไปเรื่อยๆจนกระทั่งถึง N-1 ดังตัวอย่างที่ 2.5

<u>ตัวอย่างที่ 2.5</u> ทำการสร้างเวกเตอร์ V จากคู่อันดับ (bbcaa, 1)

กำหนดให้

L แทนข้อมูลสดมภ์สุดท้ายของเมตริกซ์ M' (L=bbcaa)

F แทนข้อมูลสดมภ์แรกของเมตริกซ์ M' (F=aabbc)

เมื่อนำมาทำการสร้างเวกเตอร์เพื่อใช้ในการถอดรหัสจะมีลักษณะการทำงานดังต่อไปนี้

L	**F**	**V**
[b]	a	
b	a	
c	[b]	0
a	b	
a	c	

⇨

L	**F**	**V**
b	a	
[b]	a	
c	b	0
a	[b]	1
a	c	

⇨

L	**F**	**V**
b	a	
b	a	
[c]	b	0
a	b	1
a	[c]	2

⇨

L	**F**	**V**
b	[a]	3
b	a	
c	b	0
[a]	b	1
a	c	2

⇨

L	**F**	**V**
b	a	3
b	[a]	4
c	b	0
a	b	1
[a]	c	2

ฉะนั้นเวกเตอร์ V = $\begin{pmatrix} 3 \\ 4 \\ 0 \\ 1 \\ 2 \end{pmatrix}$

เมื่อได้เวกเตอร์ในการถอดรหัสแล้ว จะสามารถหาบล็อกข้อมูลต้นกำเนิด S ได้โดยใช้ข้อมูลจากสดมภ์สุดท้ายของเมตริกซ์ M' เวกเตอร์ V และดัชนี I หาค่าอักษรของบล็อกข้อมูลต้นกำเนิดทีละตัวดังอัลกอริทึมต่อไปนี้ และแสดงในตัวอย่างที่ 2.6

```
For j = 0 to N-1
        row = I
        S[j] = L[row]
        I = V[row]
End For
```

ตัวอย่างที่ 2.6 ทำการหาบล็อกข้อมูลต้นกำเนิด โดยใช้ข้อมูลจากตัวอย่างที่ 2.5

จากคู่ลำดับ (bbcaa, 1)

จะได้ L = bbcaa, I = 1 และ V = (3, 4, 0, 1, 2)

การหาบล็อกข้อมูลต้นกำเนิด S จะมีขั้นตอนการทำงานดังต่อไปนี้

L	V	S
b	3	b
[b]	4	
c	0	
a	1	
a	2	

⇨

L	V	S
b	3	b
b	4	a
c	0	
a	1	
[a]	2	

⇨

L	V	S
b	3	b
b	4	a
[c]	0	c
a	1	
a	2	

⇨

L	V	S
[b]	3	b
b	4	a
c	0	c
a	1	b
a	2	

⇨

L	V	S
b	3	b
b	4	a
c	0	c
[a]	1	b
a	2	a

I	=	1

I	=	4

I	=	2

I	=	0

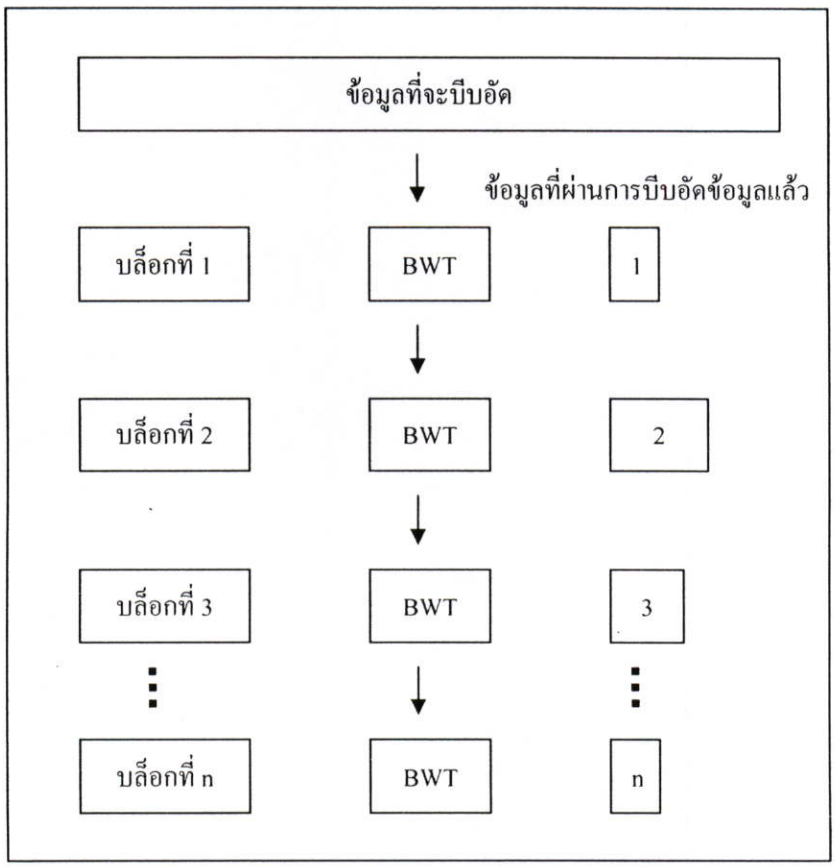
I	=	3

บล็อกข้อมูลต้นกำเนิด คือ bacba

2.1.3 บีซิพทู (Bzip2)

บีซิพทูเป็นโปรแกรมบีบอัดข้อมูลโอเพ่นซอส (Open Source) ที่มีความนิยมและหาใช้ได้ง่ายในระบบยูนิกส์ ใช้วิธีการเบอร์โรวส์-วีเลอร์ และการเข้ารหัสแบบฮัฟแมน โดยปกติการบีบอัดข้อมูลแบบนี้จะดีกว่าการบีบอัดข้อมูลแบบ LZ77/LZ78 และมีประสิทธิภาพใกล้เคียงแบบ PPM ของการบีบอัดข้อมูลโดยใช้สถิติ แต่เร็วกว่าทางด้านเวลา [14] โปรแกรมจะประมวลผลข้อมูลโดยแบ่งข้อมูลออกเป็นช่วงบล็อกขนาด 100,000 ไบต์ ถึง 900,000 ไบต์ ขึ้นอยู่กับคำสั่ง ปกติขนาดของบล็อกจะเท่ากับ 900,000 ไบต์ โปรแกรมจะอ่านข้อมูลครั้งละ 5,000 ไบต์ จนกว่าจะครบเท่ากับ

ขนาดของบล็อกที่กำหนด และประมวลผลบีบอัดข้อมูล เขียนข้อมูลที่บีบอัดแล้ว ลงบนหน่วยเก็บข้อมูล ทำเช่นนี้ต่อไปเรื่อยๆจนกว่าบล็อกข้อมูลถูกบีบอัดข้อมูลแล้วทั้งหมด ดังแสดงในรูปที่ 2.2

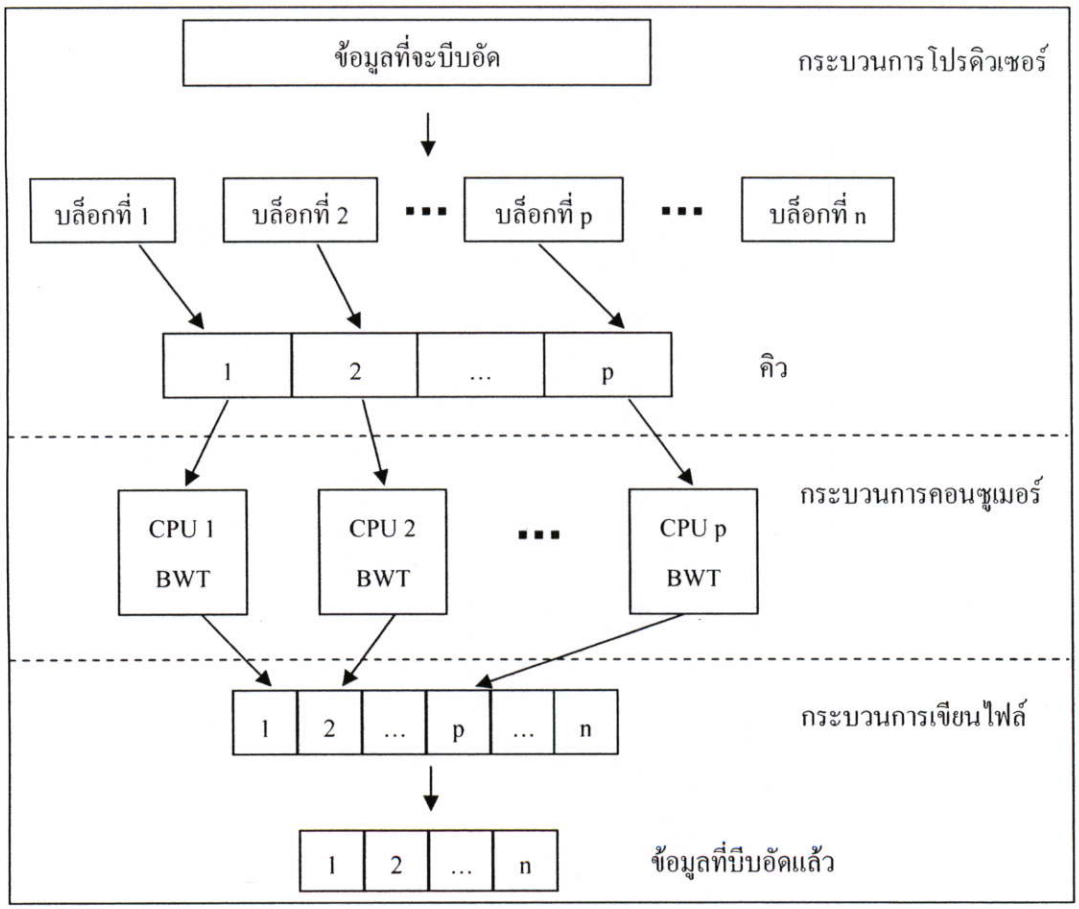


รูปที่ 2.2 แสดงการบีบอัดข้อมูลด้วยวิธีบีซิพทู

จากรูปที่ 2.2 จะพบว่า การบีบอัดข้อมูลด้วยวิธีบีซิพทูนี้เหมาะสมกับการนำไปพัฒนาเป็นการบีบอัดข้อมูลแบบขนาน เนื่องจากข้อมูลที่จะทำการบีบอัดข้อมูลถูกแบ่งออกเป็นบล็อกข้อมูลย่อยหลายๆ บล็อก จากนั้นบล็อกข้อมูลย่อยแต่ละบล็อกจะผ่านกระบวนการบีบอัดข้อมูล ซึ่งกระบวนการบีบอัดข้อมูลนี้ จะถูกใช้เหมือนกันกับทุกๆบล็อกข้อมูลย่อย ดังนั้น บล็อกข้อมูลย่อยแต่ละบล็อก สามารถนำมาประมวลผลบีบอัดข้อมูลพร้อมกันได้ในระบบประมวลผลแบบขนาน ดังจะกล่าวในหัวข้อต่อไป

2.2 การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู

การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู [3] ถูกพัฒนาขึ้นในปี ค.ศ. 2004 โดย เจฟ กิลเชริส์ท (Jeff Gilchrist) ซึ่งพัฒนาการบีบอัดข้อมูลนี้บนเครื่องคอมพิวเตอร์แบบขนานที่ใช้หน่วยความจำร่วม (Shared-Memory Parallel Architectures) วิธีนี้จะแบ่งข้อมูลออกเป็นบล็อกเท่าๆ กัน ซึ่งกำหนดโดยผู้ใช้ แทนที่จะอ่านข้อมูลครั้งละ 5,000 ไบต์ มีการนำระบบคิวและแบบจำลองโปรดิวเซอร์-คอนซูเมอร์(Producer-Consumer Model) ใช้ในการออกแบบพัฒนาระบบ โดยการประมวลผลแบบขนานจะประกอบด้วยคอนซูเมอร์หลายๆ คอนซูเมอร์ขึ้นอยู่กับหน่วยประมวลผล โดยโปรดิวเซอร์จะทำหน้าที่แบ่งข้อมูลออกเป็นบล็อกแล้วนำเข้าคิว และคอนซูเมอร์ จะทำหน้าที่อ่านข้อมูลจากคิว และประมวลผลบีบอัดข้อมูลด้วยวิธีบีซิพทู จากนั้นข้อมูลที่ถูกบีบอัดแล้วจะถูกเขียนลงไฟล์ผลลัพธ์ให้ถูกต้องตามลำดับโดยกระบวนการเขียนไฟล์ ดังแสดงในรูป 2.3 เมื่อแต่ละคอนซูเมอร์ทำแต่ละกระบวนการเสร็จแล้วจะไปอ่านข้อมูลจากคิว ทำเช่นนี้ไปเรื่อยๆจนกว่าคิวจะว่าง นั่นคือ ข้อมูลทั้งหมดถูกบีบอัดข้อมูลแล้ว



รูปที่ 2.3 แสดงการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู

2.3 การวัดสมรรถนะ (Performance Measurement)

วิธีการวัดสมรรถนะขั้นตอนวิธีการทางคอมพิวเตอร์แบบขนาน (Parallel algorithm) จะมีความซับซ้อนกว่าขั้นตอนวิธีการทางคอมพิวเตอร์แบบอนุกรม (Sequential algorithm) โดยสามารถประเมินผลจากเวลาที่ใช้ในการประมวลผล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) เมื่อใช้หน่วยประมวลผลเพิ่มขึ้น ในการประมวลผลจริงบนระบบคลัสเตอร์สามารถวัดเวลาที่ใช้ในการประมวลผลงานหนึ่งได้โดยจับเวลาที่ใช้ในการประมวลผลตั้งแต่เริ่มต้นจนสิ้นสุดการประมวลผล ซึ่งวัดได้ทั้งแบบอนุกรม (Sequential Programming) และแบบขนาน (Parallel Programming) โดยจะทำการวัดเวลาที่ใช้ในการประมวลผล ซึ่งไม่รวมเวลาที่ใช้ในการย้ายข้อมูลมาเก็บไว้ในหน่วยความจำสำรอง (Hard disk) และการตั้งค่าระบบ (Initialization) ของ MPI ซึ่งกระบวนการทั้งสองจะถูกทำครั้งเดียว

2.3.1 เวลาที่ใช้ในการประมวลผล (Response Time)

การวัดเวลาที่ใช้ในการประมวลผลสำหรับการบีบอัดข้อมูลแบบขนาน ในทางทฤษฎี (Theoretical Approach) เวลาที่ใช้ในการประมวลผลแบบขนาน (Parallel Computation) จะสามารถคำนวณได้ดังสมการที่ 2.1 [8] [13]

$$T_p = \frac{T_s}{P} \tag{2.1}$$

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 จนกระทั่งสิ้นสุดการประมวลผล หรือได้คำตอบที่ต้องการ ซึ่งเวลาที่วัดได้ดังกล่าวจะรวมเวลาที่ใช้ในการติดต่อสื่อสารด้วย ดังนั้นถึงแม้ว่าจำนวนของหน่วยประมวลผล (P) ในระบบจะมีเพิ่มขึ้นมากๆ แต่เวลาที่ได้ของระบบจะไม่ลดลงจนเข้าใกล้ศูนย์ เพราะสาเหตุมาจากเวลาที่ใช้ในการประมวลผลทั้งหมดประกอบด้วยเวลาที่ใช้ในการประมวลผล

(Computation Time) และเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล (Communication Time) ซึ่งปกติจะไม่เป็นศูนย์ ดังกรณีอุดมคติ โดยจะมีลักษณะตามสมการที่ 2.2

$$T_p = t_{computation} + t_{communication} \quad (2.2)$$

เมื่อ T_p คือ เวลาที่ใช้ในการประมวลผลแบบขนานทั้งหมดด้วย P หน่วย

$t_{computation}$ คือ เวลาที่ใช้ในการประมวลผลซึ่งไม่รวมเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผล

$t_{communication}$ คือ เวลาที่ใช้ในการสื่อสารระหว่างหน่วยประมวลผล (ถ้ามี)

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

2.3.2 อัตราการเพิ่มของความเร็ว (Speedup)

นอกจากการวัดเวลาที่ใช้ในการประมวลผลแล้วยังสามารถวัดอัตราการเพิ่มของความเร็ว (Speedup) ของการประมวลผลแบบขนานได้ ดังสมการที่ 2.3 [8] [13]

$$S_p = \frac{T_s}{T_p} \leq P \quad (2.3)$$

เมื่อ S_p คือ อัตราการเพิ่มของความเร็วที่ใช้ในประมวลผลแบบขนาน โดยใช้ P หน่วยประมวลผล ซึ่งมีค่าสูงสุดในอุดมคติเท่ากับ P

T_s คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วยหนึ่งหน่วยประมวลผล

T_p คือ เวลาที่ใช้ในการประมวลผลแบบขนานด้วย P หน่วยประมวลผล

2.3.3 ประสิทธิภาพ (Efficiency)

นอกจากการวัดเวลาที่ใช้ในการประมวลผล และอัตราการเพิ่มขึ้นของความเร็วแล้ว ยังสามารถวัดประสิทธิภาพของการประมวลผลแบบขนานได้ ดังสมการที่ 2.4 [8] [13]

$$E_p = \frac{S_p}{P} \leq 1 \qquad (2.4)$$

เมื่อ E_p คือ ประสิทธิภาพของการประมวลผลแบบขนานมีค่าในอุดมคติเท่ากับ 1

S_p คือ อัตราการเพิ่มของความเร็วที่คำนวณได้จากสมการที่ 2.3

P คือ จำนวนหน่วยประมวลผลที่ใช้บนระบบคลัสเตอร์

บทที่ 3
การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู

งานวิจัยนี้เป็นงานวิจัยทดลอง ซึ่งผู้วิจัยทำการศึกษาขั้นตอนวิธีการบีบอัดข้อมูลด้วยวิธีบีซิพทู และการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูดังที่กล่าวไปแล้วในบทที่ 2 โดยงานวิจัยนี้จะทำการลดเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลของการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู และลดการใช้งานอุปกรณ์ อินพุทเอาท์พุทศูนย์กลาง (Centralized I/O) พร้อมกันของแต่ละหน่วยประมวลผล เพื่อให้การบีบอัดข้อมูลมีสมรรถนะ (Performance) สูงขึ้นซึ่งผู้วิจัยแบ่งการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูที่เพิ่มประสิทธผลต่างๆดังกล่าวได้ 3 วิธี ดังนี้

1) การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูที่ใช้แนวความคิดเดิมที่มีอยู่ (Master-Slave Bzip2 : MSBzip2)

2) การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูที่ตัดหน่วยประมวลผลควบคุมออก (All-Worker Bzip2 : AWMBzip2) เพื่อลดการติดต่อสื่อสารเพื่อแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผลทำให้เวลาในที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลลดลง

3) การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูที่กำหนดน้ำหนักของการแบ่งบล็อกข้อมูลย่อย (All-Worker Weight Partition Bzip2 : AWWPBzip2) เพื่อให้บล็อกข้อมูลย่อยมีขนาดต่างกันทำให้แต่ละหน่วยประมวลผลใช้งานอุปกรณ์อินพุทเอาท์พุทศูนย์กลางไม่พร้อมกัน

โดยทุกวิธีดังกล่าวจะถูกพัฒนาขึ้นด้วยภาษาซี มาตราภาษาเอ็มพีไอ บนระบบพีซีคลัสเตอร์เพื่อใช้เป็นเครื่องมือในการทดลองเปรียบเทียบสมรรถนะ (Performance) ที่เพิ่มแตกต่างกัน โดยวัดผลขั้นต้น จากเวลาตอบสนอง (Response Time) จากนั้นสามารถแสดงอัตราการเพิ่มของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency) ของการบีบอัดข้อมูลทุกวิธีดังกล่าว

3.1 การบีบอัดข้อมูลแบบขนานด้วยวิธีเอ็มเอสบีซิพทู (MSBzip2)

การบีบอัดข้อมูลแบบขนานด้วยวิธีเอ็มเอสบีซิพทู (Master-Slave Bzip2 : MSBzip2) จะแบ่งหน่วยประมวลผลออกเป็น 2 ชนิด คือ หน่วยประมวลผลควบคุม (Master Processor) 1 หน่วยประมวลผล และหน่วยประมวลผลบีบอัดข้อมูล (Slave Processors) จำนวน P หน่วยประมวลผล โดยหน่วยประมวลผลทั้ง 2 ชนิด จะทำการบีบอัดข้อมูลไปพร้อมๆ กัน แต่หน่วยประมวลผลควบคุมจะประกอบด้วยกระบวนการเพิ่มขึ้นจากหน่วยประมวลผลบีบอัดข้อมูลอีก 2 กระบวนการ คือ กระบวนการแบ่งข้อมูลที่จะทำการบีบอัดข้อมูลออกเป็นบล็อกข้อมูลย่อย และกระบวนการรวมบล็อกข้อมูลย่อยที่ผ่านการบีบอัดข้อมูลแล้ว

บล็อกข้อมูลย่อยของแต่ละหน่วยประมวลผล สามารถคำนวณได้ ดังนี้

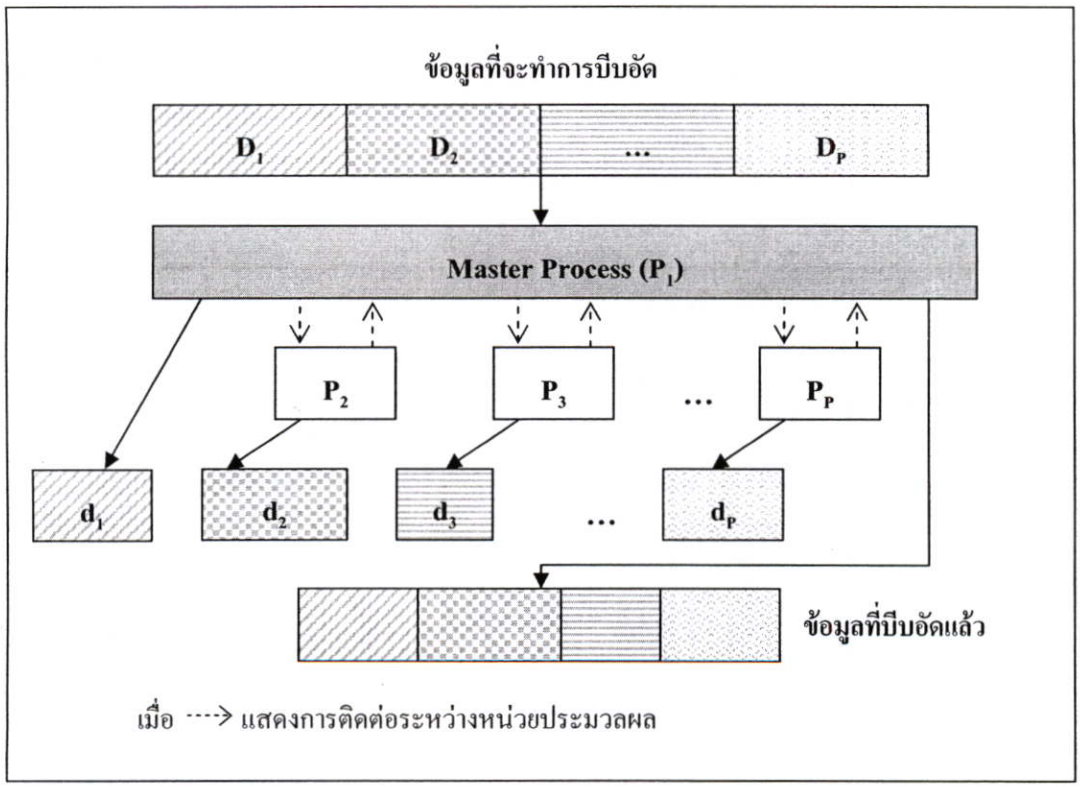
ให้ N คือ ขนาดของข้อมูลที่จะทำการบีบอัดข้อมูล D_P คือ บล็อกข้อมูลย่อยของหน่วยประมวลผลที่ i และ P คือ จำนวนหน่วยประมวลผลทั้งหมด สามารถเขียนได้ดังสมการ

$$N = \sum_{i=1}^{P} D_i$$

โดยที่

$$D_i = \frac{N}{P}$$

ข้อมูลที่ผ่านกระบวนการทั้งสอง จะมีการเรียงลำดับก่อนหลังคือ ข้อมูลที่จะทำการบีบอัดข้อมูลเมื่อผ่านกระบวนการแบ่งข้อมูลแล้วจะได้บล็อกข้อมูลย่อยจำนวน P บล็อกข้อมูล ซึ่งบล็อกข้อมูลย่อยจะมีลำดับ ดังนี้ D_1, D_2, D_3, ..., D_P บล็อกข้อมูลย่อยแต่ละบล็อก จะถูกบีบอัดข้อมูลด้วยหน่วยประมวลผลทุกหน่วยประมวลผล คือ $P_1, P_2, P_3, \ldots, P_P$ ตามลำดับ บล็อกข้อมูลย่อยของแต่ละหน่วยประมวลผลที่ผ่านการบีบอัดข้อมูลแล้วจะมีลำดับ ดังนี้ $d_1, d_2, d_3, \ldots, d_P$ จากนั้นบล็อกข้อมูลย่อยจะถูกรวมเป็นแฟ้มข้อมูลผลลัพท์โดยหน่วยประมวลผลควบคุมตามลำดับก่อนหลังของบล็อกข้อมูลย่อย ดังแสดงในรูปที่ 3.1



รูปที่ 3.1 การบีบอัดข้อมูลแบบขนานด้วยวิธีเอ็มเอสบีซิพทู

จากรูปที่ 3.1 หน่วยประมวลผลควบคุม (Master Processor) จะอ่านข้อมูลทีละบล็อกข้อมูลย่อย และส่งบล็อกข้อมูลย่อยไปยังหน่วยประมวลผลบีบอัดข้อมูล (Slave Processors) ทำเช่นนี้ไปเรื่อยๆ จนครบทุกบล็อกข้อมูลย่อย นั่นคือ ทุกหน่วยประมวลผลจะมีบล็อกข้อมูลย่อยของแต่ละหน่วยประมวลผล จากนั้นแต่ละหน่วยประมวลผลจะทำการบีบอัดข้อมูล หลังจากแต่ละหน่วยประมวลผลบีบอัดข้อมูลเสร็จ หน่วยประมวลผลบีบอัดข้อมูลจะส่งบล็อกข้อมูลย่อยที่ผ่านการบีบอัดแล้ว ไปยังหน่วยประมวลผลควบคุม และหน่วยประมวลผลควบคุมจะเขียนบล็อกข้อมูลย่อยที่ผ่านการบีบอัดแล้วของทุกๆ หน่วยประมวลผลเป็นแฟ้มข้อมูลผลลัพธ์ตามลำดับก่อนหลังของบล็อกข้อมูลย่อย

ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี (MPI-C like) ของการบีบอัดข้อมูลด้วยวิธีเอ็มเอสบีซิพทู

```
Start
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &procs);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
Master Process
 MPI_File_open(MPI_COMM_SELF, filename, MPI_MODE_RDONLY | MPI_MODE_UNIQUE_OPEN,
      MPI_INFO_NULL, &fh);
 MPI_File_get_size(fh, &filesize);
 /*calculate size of data for each process*/
 filesize = filesize/sizeof(char);
 length = filesize / nprocs + 1;
 for (i=1; i<=nprocs-1; i++ )
 {
  /*read and send data to another process */
  MPI_File_set_view(fh, i*length * sizeof(char),MPI_CHAR, MPI_CHAR, "native", MPI_INFO_NULL);
  MPI_File_read(fh, buffer, length, MPI_CHAR, &status);
  MPI_Get_count(&status, MPI_CHAR, &count);
  MPI_Send(&count, 1, MPI_INT, i, 1, MPI_COMM_WORLD );
  MPI_Send(buffer, count, MPI_CHAR, i, 1, MPI_COMM_WORLD )
 }
Slave Process
 /*receive data from master process */
 MPI_Recv(&count, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &status);
 MPI_Recv(buffer, count, MPI_CHAR, 0, 1, MPI_COMM_WORLD, &status);
All Process Compression (Bzip2)
 BZ2_bzBuffToBuffCompress (outbuf, &outsize, buffer, count, 9, 0, 30 );
Master Process
 MPI_File_open(MPI_COMM_SELF, filenamebuf, MPI_MODE_WRONLY | MPI_MODE_CREATE,
      MPI_INFO_NULL, &fh);
 MPI_File_set_view(fh, 0,MPI_CHAR, MPI_CHAR, "native", MPI_INFO_NULL);
 MPI_File_write(fh, outbuf, outsize, MPI_CHAR, MPI_STATUS_IGNORE);
 sizeview=outsize;
 for (i=1; i<=nprocs-1; i++ )
 {
  /*receive and write data from another process */
  MPI_File_set_view(fh, sizeview,MPI_CHAR, MPI_CHAR, "native", MPI_INFO_NULL);
  MPI_Recv(&outsize, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &status);
  sizeview=sizeview+outsize;
  outbuf = (char *)malloc(outsize* sizeof(char));
  MPI_Recv(outbuf, outsize, MPI_CHAR, i, 1, MPI_COMM_WORLD, &status);
  MPI_File_write(fh, outbuf, outsize, MPI_CHAR, MPI_STATUS_IGNORE);
 }
Slave Process
 /*send data to master process */
 MPI_Send(&outsize, 1, MPI_INT, 0, 1, MPI_COMM_WORLD );
 MPI_Send(outbuf, outsize, MPI_CHAR, 0, 1, MPI_COMM_WORLD );
```

รูปที่ 3.2 ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี การบีบอัดข้อมูลด้วยวิธีเอ็มเอสบีซิพทู

3.2 การบีบอัดข้อมูลแบบขนานด้วยวิธีเอดับเบิลยูบีซิพทู (AWBzip2)

การบีบอัดข้อมูลด้วยวิธีเอดับเบิลยูบีซิพทู (All-Worker Bzip2 : AWBzip2) ทุกๆ หน่วยประมวลผลจะประกอบด้วยกระบวนการต่างๆ 3 กระบวนการ ดังนี้ กระบวนการแรก คือ กระบวนการแบ่งข้อมูล ข้อมูลที่จะทำการบีบอัดข้อมูลจะถูกแบ่งออกเป็นบล็อกข้อมูลย่อยๆ โดยแต่ละหน่วยประมวลผลจะมีตำแหน่งและขนาดของบล็อกข้อมูลย่อยของหน่วยประมวลผลนั้นที่จะทำการบีบอัดข้อมูล

บล็อกข้อมูลย่อยของแต่ละหน่วยประมวลผล สามารถคำนวณได้ ดังนี้

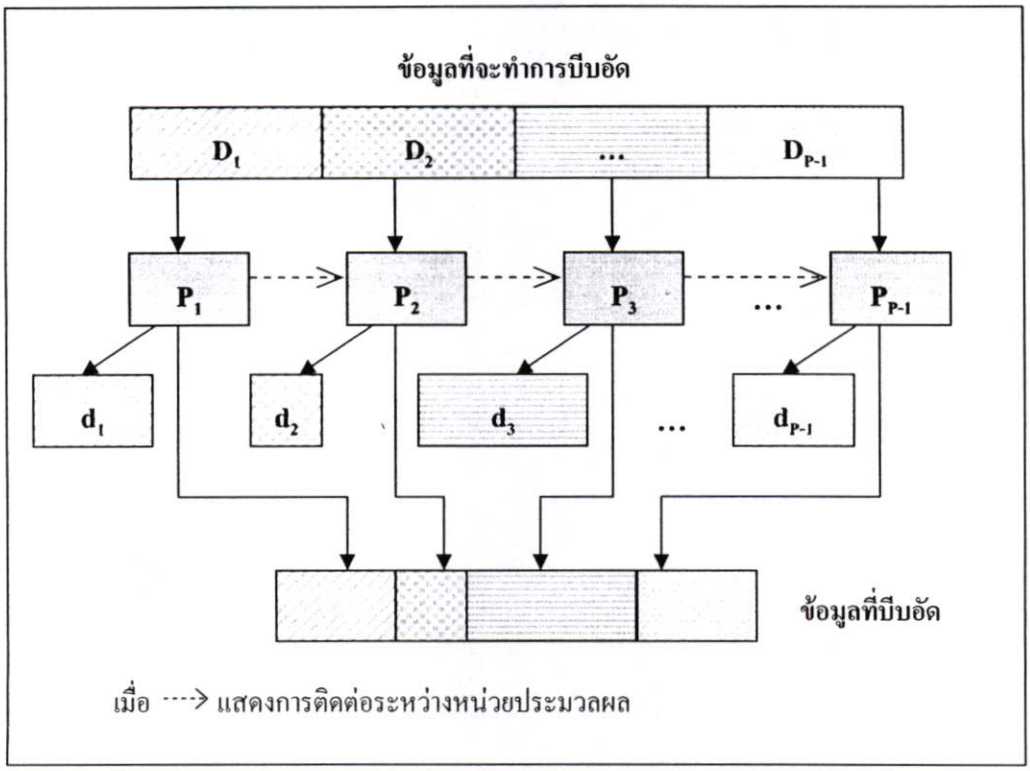
ให้ N คือ ขนาดของข้อมูลที่จะทำการบีบอัดข้อมูล D_P คือ บล็อกข้อมูลย่อยของหน่วยประมวลผลที่ i และ P คือ จำนวนหน่วยประมวลผลทั้งหมด สามารถเขียนได้ดังสมการ

$$N = \sum_{i=1}^{P} D_i$$

โดยที่

$$D_i = \frac{N}{P}$$

กระบวนการที่สองคือ กระบวนการบีบอัดข้อมูล บล็อกข้อมูลย่อยที่ได้จากกระบวนการแรก จะถูกทำการบีบอัดข้อมูล และจะได้บล็อกข้อมูลย่อยที่ผ่านการบีบอัดแล้ว จากนั้นแต่ละหน่วยประมวลผล จะมีการติดต่อสื่อสารกันระหว่างหน่วยประมวลผล คือ จะมีการส่งขนาดของบล็อกข้อมูลย่อยที่ทำการบีบอัดข้อมูลแล้วของหน่วยประมวลผลข้อมูลลำดับแรก (P_1) ไปยังหน่วยประมวลผลข้อมูลลำดับถัดไป คือ หน่วยประมวลผลข้อมูลลำดับที่สอง (P_2) จากนั้น หน่วยประมวลผลข้อมูลลำดับที่สองจะนำขนาดบล็อกข้อมูลย่อยที่ทำการบีบอัดข้อมูลแล้วของหน่วยประมวลผลลำดับก่อนหน้า รวมกับขนาดของบล็อกข้อมูลย่อยที่ทำการบีบอัดข้อมูลแล้ว ส่งไปยังหน่วยประมวลผลของบล็อกข้อมูลย่อยลำดับถัดไป ซึ่งก็คือ หน่วยประมวลผลข้อมูลลำดับที่สาม (P_3) จากนั้น หน่วยประมวลผลข้อมูลลำดับที่สามจะนำขนาดบล็อกข้อมูลย่อยที่ทำการบีบอัดข้อมูลแล้วของหน่วยประมวลผลลำดับก่อนหน้า รวมกับขนาดของบล็อกข้อมูลย่อยที่ทำการบีบอัดข้อมูลแล้ว ส่งไปยังหน่วยประมวลผลของบล็อกข้อมูลย่อยลำดับถัดไป ทำเช่นนี้ต่อไปจนกระทั่งถึงหน่วยประมวลผลของบล็อกข้อมูลลำดับสุดท้าย (P_p) กระบวนการสุดท้ายคือ กระบวนการเขียนแฟ้มข้อมูล หลังจากแต่ละหน่วยประมวลผลได้ติดต่อสื่อสารกันแล้ว ทำให้ทุกๆ หน่วยประมวลผลสามารถเขียนแฟ้มข้อมูลของแต่ละบล็อกข้อมูลย่อยที่ทำการบีบอัดแล้วได้ ดังแสดงในรูปที่ 3.3



รูปที่ 3.3 การบีบอัดข้อมูลแบบขนานด้วยวิธีเอดับเบิลยูบีซิพทู

ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี ของการบีบอัดข้อมูลด้วยวิธีเอดับเบิลยูบีซิพทู

```
Start
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &procs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
All Process Calculate size of data and Read data
  MPI_File_open(MPI_COMM_SELF, filename, MPI_MODE_RDONLY | MPI_MODE_UNIQUE_OPEN,
        MPI_INFO_NULL, &fh);
  MPI_File_get_size(fh, &filesize);
  filesize = filesize/sizeof(char);
  length = filesize / nprocs + 1;
  MPI_File_set_view(fh, myrank*length * sizeof(char),MPI_CHAR, MPI_CHAR, "native",
        MPI_INFO_NULL);
  MPI_File_read(fh, buffer, length, MPI_CHAR, &status);
  MPI_Get_count(&status, MPI_CHAR, &count);
Compression (Bzip2)
  BZ2_bzBuffToBuffCompress (outbuf, &outsize, buffer, count, 9, 0, 30 );
All Process Communication and write data to a single file
  MPI_File_open(MPI_COMM_SELF, filenamebuf, MPI_MODE_WRONLY  | MPI_MODE_CREATE,
        MPI_INFO_NULL, &fh);
  if (myrank==0)
    start=0;
  if (myrank>0)
    MPI_Recv(&start, 1, MPI_INT, myrank-1, 1, MPI_COMM_WORLD, &status);
  if (myrank<nprocs-1)
  {
    size=start+outsize;
    MPI_Send(&size, 1, MPI_INT, myrank+1, 1, MPI_COMM_WORLD );
  }
  MPI_File_set_view(fh, start, MPI_CHAR, MPI_CHAR, "native", MPI_INFO_NULL);
  MPI_File_write(fh, outbuf, outsize, MPI_CHAR, MPI_STATUS_IGNORE);
```

รูปที่ 3.4 ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี การบีบอัดข้อมูลด้วยวิธีเอดับเบิลยูบีซิพทู

3.3 การบีบอัดข้อมูลแบบขนานด้วยวิธีเอดับเบิลยูดับเบิลยูบีซิพทู (AWWBzip)

การบีบอัดข้อมูลด้วยวิธีเอดับเบิลยูดับเบิลยูบีซิพทู (All-Worker Weight Bzip2 : AWWBzip2) ทุกๆหน่วยประมวลผลจะประกอบด้วยกระบวนการต่างๆ 3 กระบวนการ ดังนี้ กระบวนการแรกคือกระบวนการแบ่งข้อมูล ข้อมูลที่จะทำการบีบอัดข้อมูลจะถูกแบ่งออกเป็นบล็อกข้อมูลย่อยๆ โดยแต่ละหน่วยประมวลผลจะมีตำแหน่งและขนาดของบล็อกข้อมูลย่อยของหน่วยประมวลผลนั้น

บล็อกข้อมูลย่อยของแต่ละหน่วยประมวลผล สามารถคำนวณได้ ดังนี้

ให้ N คือ ขนาดของข้อมูลที่จะทำการบีบอัดข้อมูล D_i คือ บล็อกข้อมูลย่อยของหน่วยประมวลผลที่ i และ P คือ จำนวนหน่วยประมวลผลทั้งหมด สามารถเขียนได้ดังสมการ

$$N = \sum_{i=1}^{P} D_i$$

โดยที่

$$D_i = B + W_i$$

เมื่อ B และ W_i คือ บล็อกข้อมูลย่อยเริ่มต้น และ น้ำหนักที่กำหนดให้บล็อกข้อมูลย่อยของหน่วยประมวลผลที่ i ตามลำดับ โดยบล็อกข้อมูลย่อยของหน่วยประมวลผล P_i จะมีขนาดเล็กกว่าบล็อกข้อมูลย่อยของหน่วยประมวลผล P_{i+1} ซึ่งขนาดของบล็อกข้อมูลย่อยจะมีการเพิ่มขึ้นแบบเชิงเส้น (Linear) เพราะการเพิ่มขึ้นแบบนี้เหมาะสมสำหรับการกำหนดน้ำหนักของการแบ่งบล็อกข้อมูลมากกว่าการเพิ่มขึ้นแบบอื่น เช่น การเพิ่มขึ้นแบบเอกซ์โพเนนเชียล (Exponential) การเพิ่มขึ้นแบบนี้ไม่เหมาะสม เนื่องจาก ถ้าแบ่งข้อมูลด้วยการเพิ่มขึ้นแบบนี้จะทำให้บล็อกข้อมูลย่อยของหน่วยประมวลท้ายๆ มีขนาดใหญ่มาก ทำให้ต้องใช้เวลาในการบีบอัดข้อมูลมากขึ้น หรือการเพิ่มขึ้นแบบลอการิทึม (Logarithm) การเพิ่มขึ้นแบบนี้ไม่เหมาะสม เนื่องจาก ถ้าแบ่งข้อมูลด้วยการเพิ่มขึ้นแบบนี้จะทำให้บล็อกข้อมูลย่อยของหน่วยประมวลผลท้ายๆ มีขนาดไม่แตกต่างกันเหมือนกับไม่ได้กำหนดน้ำหนักของการแบ่งบล็อกข้อมูล ทำให้ใช้อุปกรณ์อินพุทเอาท์พุทศูนย์กลางพร้อมกัน ซึ่งน้ำหนักที่กำหนดให้บล็อกข้อมูลย่อยที่เหมาะสม สามารถหาได้จากสมการ

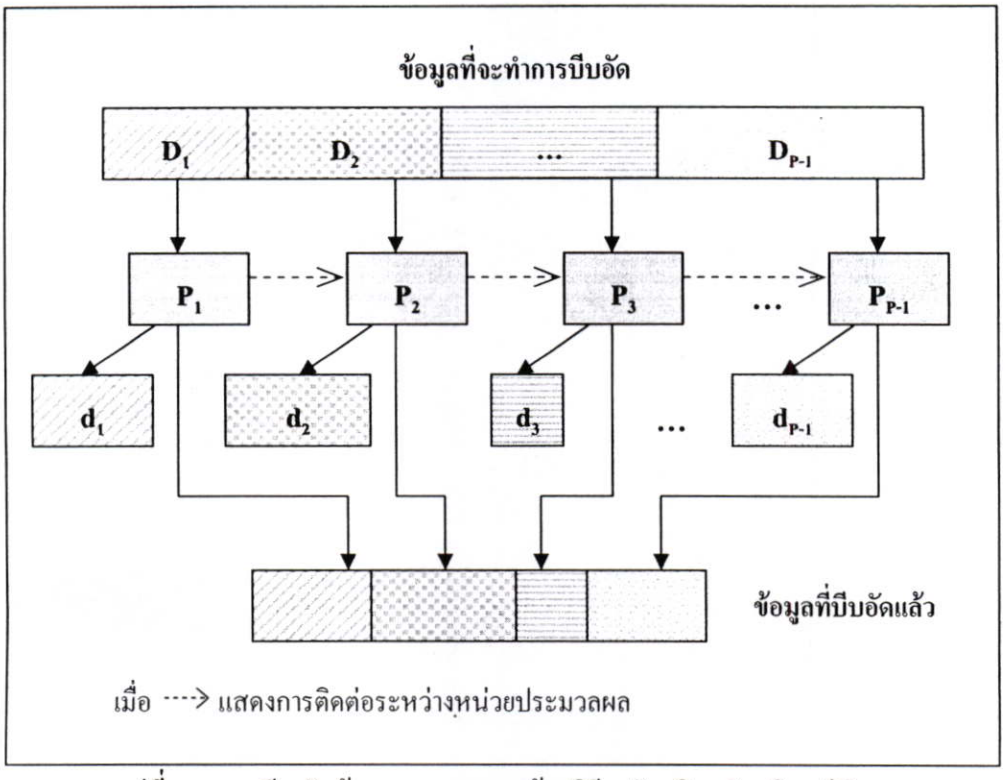
$$B = \frac{N - \sum_{i=1}^{P} W_i}{P}$$

$$W_i = \frac{i(i-1)}{2} \times w$$

เมื่อ w คือ น้ำหนักที่กำหนดให้บล็อกข้อมูลย่อยมีขนาดแตกต่างกัน โดยค่าน้ำหนักจะแตกต่างกันขึ้นอยู่กับประสิทธิภาพของระบบพีซีคลัสเตอร์ เช่น w = 0.01%, 0.02%, 0.03%, …, 0.99%, 1% ของขนาดข้อมูลที่จะทำการบีบอัดข้อมูล

วิธีการบีบอัดข้อมูลวิธีนี้ต่างกับการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูแบบเอดับเบิลยูบีซิพทู คือ วิธีเอดับเบิลยูบีซิพทูกำหนดให้ บล็อกข้อมูลย่อยของแต่ละหน่วยประมวลผลมีขนาดที่เท่ากัน แต่วิธีนี้บล็อกข้อมูลย่อยของแต่ละหน่วยประมวลผลจะมีขนาดต่างกันคือ บล็อกข้อมูลย่อยลำดับแรกจะมีขนาดเล็กที่สุด บล็อกข้อมูลย่อยลำดับที่สองจะมีขนาดเพิ่มขึ้นจากบล็อกข้อมูลย่อยลำดับแรกเท่ากับน้ำหนักที่กำหนดให้ บล็อกข้อมูลย่อยลำดับถัดไปจะมีขนาดเพิ่มขึ้นจากบล็อกข้อมูลย่อยลำดับก่อนหน้าเท่ากับน้ำหนักที่กำหนดให้ นั่นคือ บล็อกข้อมูลย่อยจะมีขนาดเพิ่มขึ้นเรื่อยๆ จนกระทั่งถึงบล็อกข้อมูลย่อยลำดับสุดท้าย

กระบวนการที่สองคือ กระบวนการบีบอัดข้อมูล บล็อกข้อมูลย่อยที่ได้จากกระบวนการแรก จะถูกทำการบีบอัดข้อมูล และจะได้บล็อกข้อมูลย่อยที่ผ่านการบีบอัดแล้ว จากนั้นแต่ละหน่วยประมวลผล จะมีการติดต่อสื่อสารกันระหว่างหน่วยประมวลผล คือ จะมีการส่งขนาดของบล็อกข้อมูลย่อยที่ทำการบีบอัดข้อมูลแล้วของหน่วยประมวลผลข้อมูลลำดับแรก (P_1) ไปยังหน่วยประมวลผลข้อมูลลำดับถัดไป คือ หน่วยประมวลผลข้อมูลลำดับที่สอง (P_2) จากนั้น หน่วยประมวลผลข้อมูลลำดับที่สองจะนำขนาดบล็อกข้อมูลย่อยที่ทำการบีบอัดข้อมูลแล้วของหน่วยประมวลผลลำดับก่อนหน้า รวมกับขนาดของบล็อกข้อมูลย่อยที่ทำการบีบอัดข้อมูลแล้ว ส่งไปยังหน่วยประมวลผลของบล็อกข้อมูลย่อยลำดับถัดไป ซึ่งก็คือ หน่วยประมวลผลข้อมูลลำดับที่สาม (P_3) จากนั้น หน่วยประมวลผลข้อมูลลำดับที่สามจะนำขนาดบล็อกข้อมูลย่อยที่ทำการบีบอัดข้อมูลแล้วของหน่วยประมวลผลลำดับก่อนหน้า รวมกับขนาดของบล็อกข้อมูลย่อยที่ทำการบีบอัดข้อมูลแล้ว ส่งไปยังหน่วยประมวลผลของบล็อกข้อมูลย่อยลำดับถัดไป ทำเช่นนี้ต่อไปจนกระทั่งถึงหน่วยประมวลผลของบล็อกข้อมูลลำดับสุดท้าย (P_p) กระบวนการสุดท้ายคือ กระบวนการเขียนแฟ้มข้อมูล หลังจากแต่ละหน่วยประมวลผลได้ติดต่อสื่อสารกันแล้ว ทำให้ทุกๆ หน่วยประมวลผลสามารถเขียนแฟ้มข้อมูลของแต่ละบล็อกข้อมูลย่อยที่ทำการบีบอัดแล้วได้ ดังแสดงในรูปที่ 3.5



รูปที่ 3.5 การบีบอัดข้อมูลแบบขนานด้วยวิธีเอดับเบิลยูดับเบิลยูบีซิพทู

ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี ของการบีบอัดข้อมูลด้วยวิธีเอดับเบิลยูดับเบิลยูบีซิพทู

```
Start
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &procs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
All Process Calculate size of data and Read data
  MPI_File_open(MPI_COMM_SELF, filename, MPI_MODE_RDONLY, MPI_INFO_NULL, &fh);
  MPI_File_get_size(fh, &filesize);
  filesize = filesize/sizeof(char);
  Pweight = filesize * weight / 1000;
  length = (filesize - nprocs * (nprocs - 1) * Pweight / 2 ) / nprocs + 1;
  start=0;
  for (i=0; i<myrank; i++ )
  {
    length = length + i * Pweight;
    start = start+length;
  }
  length = length + myrank * Pweight;
  MPI_File_set_view(fh, start, MPI_CHAR, MPI_CHAR, "native", MPI_INFO_NULL);
  MPI_File_read(fh, buffer, length, MPI_CHAR, &status);
  MPI_Get_count(&status, MPI_CHAR, &count);
Compression (Bzip2)
  BZ2_bzBuffToBuffCompress (outbuf, &outsize, buffer, count, 9, 0, 30 );
All Process Communication and write data to a single file
  MPI_File_open(MPI_COMM_SELF, filenamebuf, MPI_MODE_WRONLY | MPI_MODE_CREATE,
        MPI_INFO_NULL, &fh);
  if (myrank==0)
    start=0;
  if (myrank>0)
    MPI_Recv(&start, 1, MPI_INT, myrank-1, 1, MPI_COMM_WORLD, &status);
```

```
if (myrank<nprocs-1)
{
  size=start+outsize;
  MPI_Send(&size, 1, MPI_INT, myrank+1, 1, MPI_COMM_WORLD );
}
MPI_File_set_view(fh, start, MPI_CHAR, MPI_CHAR, "native", MPI_INFO_NULL);
MPI_File_write(fh, outbuf, outsize, MPI_CHAR, MPI_STATUS_IGNORE);
```

รูปที่ 3.6 ตัวอย่างรหัสเหมือนเอ็มพีไอ-ซี การบีบอัดข้อมูลด้วยวิธีเอดับเบิลยูดับเบิลยูบีซิพทู

บทที่ 4

การทดลองและผลการทดลอง

งานวิจัยนี้เป็นการนำเสนอการเพิ่มสมรรถนะ (Performance) โดยการลดเวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลของการบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทู และลดการใช้งานอุปกรณ์ อินพุทเอาท์พุทศูนย์กลาง (Centralized I/O) พร้อมกันของแต่ละหน่วยประมวลผล โดยวิธีต่างๆ 3 วิธี (MSBzip2, AWBzip2 และ AWWBzip2) ที่เสนอได้แสดงขั้นตอนวิธีไว้แล้วในบทที่ 3 ส่วนเนื้อหาในบทนี้ จะเป็นการเสนอผลการพัฒนาโปรแกรมการบีบอัดข้อมูลแบบขนานดังกล่าวเพื่อประมวลผลบนระบบพีซีคลัสเตอร์ ซึ่งเป็นระบบที่ได้รับความสนใจเป็นอย่างมากในปัจจุบัน เนื่องจากมีราคาไม่แพง และมีขั้นตอนการพัฒนาที่ไม่ยุ่งยากซับซ้อนมาก

4.1 ระบบคอมพิวเตอร์คลัสเตอร์ที่ใช้ในการทดลอง

ระบบคลัสเตอร์ที่ใช้ในการทดลอง เป็นระบบคลัสเตอร์แบบ "Homogenous Cluster" ที่มีส่วนประกอบ ประสิทธิภาพในการประมวลผล และภาระงานที่รับผิดชอบในแต่ละเครื่องมีจำนวนเท่ากัน นอกจากนั้นในขณะทำการทดลองระบบคลัสเตอร์ดังกล่าวจะไม่มีภาระงานอื่นใดประมวลผลอยู่ในระบบคลัสเตอร์ที่ใช้ในการทดลอง มีส่วนประกอบพื้นฐานของระบบซึ่งจะประกอบด้วยส่วนประกอบด้านฮาร์ดแวร์ ระบบปฏิบัติการและซอฟท์แวร์ ที่ทำงานในระดับของแกน (Kernel) ของระบบปฏิบัติการซึ่งเรียกซอฟท์แวร์นี้ว่า "คลัสเตอร์มิดเดิลแวร์" และสุดท้ายคือการเชื่อมต่อแต่ละโหนดข้อมูลเข้าด้วยกันผ่านทางเครือข่ายความเร็วสูง ดังนี้

1) ส่วนประกอบด้านฮาร์ดแวร์

โครงสร้างทางฮาร์ดแวร์ของระบบที่ใช้ในการทดลอง เป็นคลัสเตอร์ที่มีโครงสร้างเหมือนกัน (Homogenous Cluster) คือหน่วยประมวลผล ขนาดของหน่วยความจำ และส่วนประกอบอื่นๆ เหมือนกัน ระบบที่ใช้ในการทดลองเป็นระบบคลัสเตอร์ที่พัฒนาขึ้นโดยสำนักวิจัยและบริการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง โดยระบบในปัจจุบันนี้ ประกอบด้วยคอมพิวเตอร์ส่วนบุคคลจำนวน 5 เครื่อง โดยใช้เครื่องคอมพิวเตอร์ชนิดที่มีสองหน่วยประมวลผลในเครื่องเดียวกัน (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) หน่วยความจำแคช (Cache Memory) ขนาด 512 kb

ฉ) เชื่อมต่อผ่านเครือข่ายความเร็ว 1000 เมกกะบิตต่อวินาที (Mbps)

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 ซึ่งการเลือกใช้ระบบปฏิบัติการนั้นขึ้นอยู่กับระบบคลัสเตอร์ด้วยว่าเป็นแบบใด (PCs or Workstation) นอกจากนี้ยังขึ้นอยู่กับความสะดวก ความเชี่ยวชาญและอุปกรณ์ที่เลือกใช้ อีกอย่างหนึ่งคือจะขึ้นอยู่กับซอฟท์แวร์และชุดคำสั่งที่เลือกใช้ด้วย เช่น ถ้าเลือกใช้ OpenMosix ที่ทำงานได้บนระบบลีนุกซ์เท่านั้น ก็จำเป็นต้องเลือกใช้ลีนุกต์เป็นระบบปฏิบัติการอีกทั้งฮาร์ดแวร์ที่สนับสนุนด้วย โดยส่วนใหญ่ซอฟท์แวร์และชุดคำสั่งที่ถูกพัฒนาขึ้นมาเพื่อใช้ในระบบคลัสเตอร์นี้จะทำงานเข้ากันได้กับระบบปฏิบัติการตระกูลยูนิกซ์เกือบทุกชนิดอยู่แล้ว

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

3) คลัสเตอร์มิดเดิลแวร์

คลัสเตอร์มิดเดิลแวร์ คือซอฟท์แวร์ที่ทำงานในระดับเดียวกับแกน (Kernel) ของระบบปฏิบัติการ มีหน้าที่ในการกระจายงาน (Process) จากโหนดหนึ่งไปยังโหนดอื่นๆ ที่อยู่ในระบบคลัสเตอร์เดียวกัน โดยส่วนใหญ่แล้วคลัสเตอร์มิดเดิลแวร์นี้จะเป็นซอฟท์แวร์ที่เขียนเพิ่มเติม

เข้าไปในแกนของระบบปฏิบัติการ เพื่อให้ทุกเครื่องที่อยู่ในระบบคลัสเตอร์เสมือนเป็นเครื่องคอมพิวเตอร์หลายหน่วยประมวลผลขนาดใหญ่ ที่เสมือนมีหน่วยความจำ หน่วยประมวลผลอยู่ที่เดียวกันเหมือนกับระบบคอมพิวเตอร์แบบ SMP (Symmetric Multi Processor) หรือแบบ MMP (Massive Multi Processor)

4) การเชื่อมต่อเครือข่าย

การสื่อสารระหว่างหน่วยประมวลผลในระบบคลัสเตอร์จะทำผ่านระบบเครือข่ายความเร็วสูงซึ่งอุปกรณ์เครือข่ายแต่ละชนิดจะมีความเร็วและราคาแตกต่างกันไป ตัวอย่างอุปกรณ์เครือข่ายที่นิยมนำมาใช้ในการสร้างระบบคลัสเตอร์ มีดังนี้

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

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

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

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

5) เอ็มพีไอซีเอช (MPICH)

เอ็มพีไอซีเอช (MPICH) เป็นการพัฒนาชุดคำสั่งตามมาตรฐานขึ้นมาใช้งานจริงมากที่สุดคือมาตรฐานการส่งผ่านข้อความ (Message Passing Interface: MPI) ซึ่งสามารถพัฒนาโปรแกรมภาษาต่างๆ เช่น โปรแกรมภาษาซี (C Language) โปรแกรมภาษาฟอร์แทน (Fortran Language) และโปรแกรมภาษาจาวา (Java Language) เป็นต้น ที่สนับสนุนการโปรแกรมแบบขนาน ซึ่งมีฟังก์ชันพื้นฐานต่างๆ รวมทั้งมีคำสั่งใช้งานเพื่อทำการคอมไพล์โปรแกรมที่เขียนขึ้น โดยเอ็มพีไอซีเอชนี้จะสนับสนุนการพัฒนาโปรแกรมภาษาซี และโปรแกรมภาษาฟอร์แทนเท่านั้น

การทดลองในงานวิจัยนี้ จะนำโปรแกรมที่ได้พัฒนาขึ้นไปประมวลผลบนระบบพีซีคลัสเตอร์ ของสำนักวิจัยและบริการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง โดยคำสั่งที่ใช้ในการทดลองมีรูปแบบดังนี้

mpirun –nolocal –np <P1> bzipC <P2> <P3> <P4>

เมื่อ <P1> คือ จำนวนหน่วยประมวลผล

<P2> คือ ชื่อแฟ้มข้อมูลที่ทำการบีบอัด

<P3> คือ วิธีการบีบอัดข้อมูล โดยมีค่าดังนี้ 1 แทนวิธีเอ็มเอสบีซิพทู (MSBzip2) 2 แทนวิธีเอดับเบิลยูบีซิพทู (AWBzip2) และ 3 แทนวิธีเอดับเบิลยูดับเบิลยูบีซิพทู (AWWBzip2)

<P4> คือ น้ำหนักของการแบ่งบล็อกข้อมูลย่อย สำหรับการบีบอัดข้อมูลแบบขนานด้วยวิธีเอดับเบิลยูดับเบิลยูบีซิพทู (AWWBzip2)

ผลการทดลองในงานวิจัยนี้ จะทำการบันทึกเวลาที่ใช้ในการประมวลผล โดยนำเวลาที่มากที่สุดที่ใช้ในการประมวลผล มาหาค่าเฉลี่ยโดยทำการทดลองซ้ำทั้งหมด 3 ครั้ง และทำการบันทึกขนาดของข้อมูลที่ผ่านการบีบอัดข้อมูลแล้ว ของทุกๆวิธีการบีบอัดข้อมูล และตรวจสอบความถูกต้องของการบีบอัดข้อมูลโดยนำแฟ้มข้อมูลที่ผ่านการบีบอัดแล้วไปคลายข้อมูลด้วยวิธีการคลายข้อมูลแบบอนุกรม และนำแฟ้มข้อมูลที่คลายข้อมูลแล้ว ตรวจสอบความถูกต้องกับแฟ้มข้อมูลต้นฉบับก่อนการบีบอัดข้อมูล ด้วยโปรแกรม WinDiff ซึ่งสามารถดาวน์โหลดได้จาก http://www.grigsoft.com/windiff.zip ซึ่งถ้าแฟ้มข้อมูลที่ผ่านการบีบอัดถูกคลายข้อมูล แล้วนำมาเปรียบเทียบกับแฟ้มข้อมูลต้นฉบับจะต้องมีข้อมูลที่เหมือนกัน

4.2 ลักษณะข้อมูล การเลือกข้อมูลและเหตุผลการเลือก

ข้อมูลที่ถูกใช้ในงานวิจัยต่างๆ ที่เกี่ยวกับการบีบอัดข้อมูลแบบที่ไม่มีการสูญเสียข้อมูล (Lossless Compression) แบ่งออกเป็น 4 กลุ่ม ได้แก่

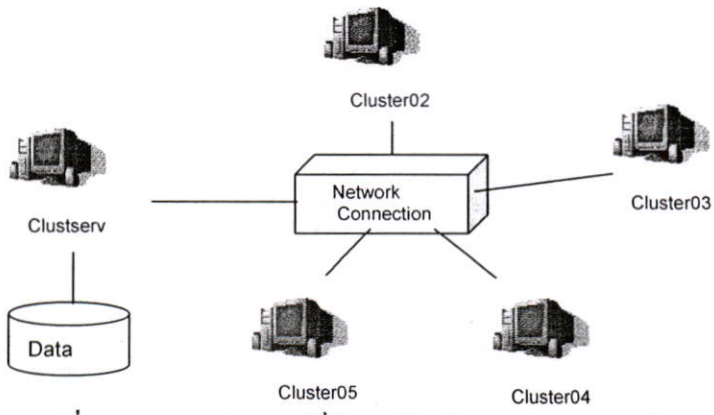
1) ชุดข้อมูลแคลกอรี่ (Calgary Corpus)
2) ชุดข้อมูลแคนเทอร์เบอรี่ (Canterbury Corpus)
3) ชุดข้อมูลโปรตีน (Protein Corpus)
4) ชุดข้อมูลซิลีเซีย (Silesia Corpus)

ชุดข้อมูลดังกล่าวถูกใช้ในการวัดสมรรถนะ (Performance Measurement) ของวิธีการบีบอัดข้อมูลแบบอนุกรมวิธีต่างๆ แต่งานวิจัยนี้เป็นการพัฒนาเพื่อเพิ่มสมรรถนะของการบีบอัดข้อมูลข้อมูลแบบขนาน ทำให้ชุดข้อมูลที่ใช้ในการวัดสมรรถนะของวิธีการบีบอัดข้อมูลแบบอนุกรม ไม่เหมาะสมสำหรับการวัดสมรรถนะของวิธีการบีบอัดข้อมูลแบบขนาน เนื่องจากชุดข้อมูลเหล่านี้มีขนาดเล็ก ดังนั้น งานวิจัยนี้ จึงทำการเพิ่มชุดข้อมูลที่มีขนาดใหญ่เพื่อใช้ในการวัดสมรรถนะ โดยชุด

ข้อมูลที่ใช้ในการวัดสมรรถนะประกอบด้วย แฟ้มข้อมูลต่างๆ ในชุดข้อมูลกูเทนเบริก (Gutenberg Corpus) ดังนี้

ชื่อข้อมูล	ขนาดข้อมูล (ไบต์)	ที่มาของข้อมูล
00ws110.txt	4,651,867	http://gutenberg.elib.com/gutenberg/etext00/
20hgp10.txt	12,066,257	http://gutenberg.elib.com/gutenberg/etext00/
22hgp11.txt	34,707,288	http://gutenberg.elib.com/gutenberg/etext00/
0xhgp10.txt	57,923,382	http://gutenberg.elib.com/gutenberg/etext00/
07hgp10.txt	73,370,264	http://gutenberg.elib.com/gutenberg/etext00/

งานวิจัยนี้ ข้อมูลที่ใช้ในการวัดสมรรถนะของการบีบอัดข้อมูลจะถูกจัดเก็บอยู่ในหน่วยความจำสำรองของหน่วยประมวลผลหลัก ซึ่งเรียกการเก็บข้อมูลแบบนี้ว่า "การเก็บข้อมูลแบบรวมศูนย์กลาง" (Centralized Data) ดังรูปที่ 4.1



รูปที่ 4.1 ระบบคลัสเตอร์ที่มีการเก็บข้อมูลแบบรวมศูนย์กลาง

4.3 ผลการทดลอง

จากการพัฒนาโปรแกรมบีบอัดข้อมูลแบบขนาน ได้นำไปทดลองบนระบบพีซีคลัสเตอร์ของสำนักวิจัยและบริการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ซึ่งประกอบด้วย หน่วยประมวลผล 10 หน่วย ขนาด 2.4 GHz จำนวน 5 เครื่องหน่วยความจำหลัก 1 Gb หน่วยความจำสำรอง 100 Gb หน่วยความจำแคช (Cache memory) 512 Kb เชื่อมต่อกันด้วยความเร็วขนาด 1000 Mbps ระบบปฏิบัติการ Suse และโปรแกรม MPICH 1.2 ทำการวัดสมรรถนะของวิธีการบีบอัดข้อมูลแบบขนานทั้ง 3 วิธีที่เสนอ คือ

1) การบีบอัดข้อมูลแบบขนานด้วยวิธีเอ็มเอสบีซิพทู (Master-Slave Bzip2 : MSBzip2)
2) การบีบอัดข้อมูลแบบขนานด้วยวิธีเอดับเบิลยูบีซิพทู (All-Worker Bzip2 : AWBzip2)
3) การบีบอัดข้อมูลแบบขนานด้วยวิธีเอดับเบิลยูดับเบิลยูบีซิพทู (All-Worker Weight Bzip2 : AWWBzip2)

รวมถึงการบีบอัดข้อมูลแบบอนุกรมเพื่อนำมาเปรียบเทียบกัน โดยการวัดสมรรถนะประกอบด้วย เวลาที่ใช้ในการประมวลผล (Response Time) อัตราการเพิ่มขึ้นของความเร็ว (Speedup) และประสิทธิภาพ (Efficiency)

หมายเหตุ

วิธีการบีบอัดข้อมูลแบบขนานด้วยวิธีเอดับเบิลยูดับเบิลยูบีซิพทู (AWWBzip2) จะทำการทดลองซ้ำๆ โดยให้น้ำหนักที่กำหนดให้บล็อกข้อมูลย่อยมีขนาดแตกต่างกัน (w) ดังต่อไปนี้ 0.01%, 0.02%, 0.03%, …, 0.2% ของขนาดข้อมูลที่จะทำการบีบอัดข้อมูล เพื่อหาค่า w ที่เหมาะสมกับระบบพีซีคลัสเตอร์ที่ใช้ในการทดลอง ซึ่งน้ำหนักที่กำหนดให้บล็อกข้อมูลย่อยมีขนาดแตกต่างกัน (w) ที่เหมาะสมกับระบบพีซีคลัสเตอร์ที่ใช้ในการทดลอง คือ 0.1 % ของขนาดข้อมูลที่จะทำการบีบอัดข้อมูล

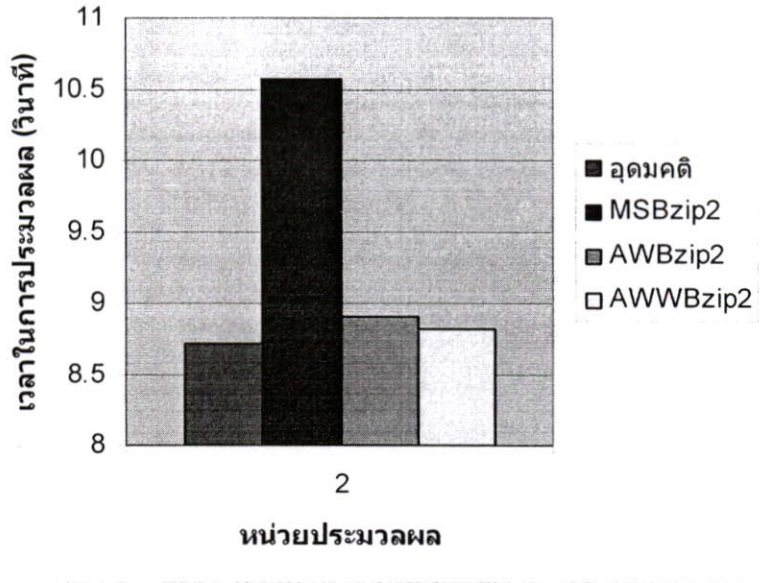
4.3.1 ผลการทดลองเปรียบเทียบกับเวลาในอุดมคติ

การทดลองเปรียบเทียบเวลาที่ประมวลผลจริงกับเวลาในอุดมคติ ซึ่งเป็นการวัดสมรรถนะของระบบ (System Performance) ที่ใช้ในการทดลอง โดยทำการทดลองบีบอัดข้อมูลแฟ้มข้อมูล 22hgp11.txt ซึ่งมีขนาดเท่ากับ 34,707,288 ไบต์ ประมวลผลด้วยหน่วยประมวลผล(P) 2 หน่วยประมวลผล มีผลการทดลองดังนี้ ตารางที่ 4.1 และรูปที่ 4.2 แสดงผลการเปรียบเทียบเวลาที่ใช้ในการประมวลผล ทั้ง 3 วิธี เปรียบเทียบกับเวลาที่ใช้ในการประมวลผลในอุดมคติ (T_S/P) ตารางที่ 4.2 และรูปที่ 4.3 แสดงอัตราการเพิ่มขึ้นของความเร็ว ทั้ง 3 วิธี ($S_P = T_S/T_P$) เปรียบเทียบกับอัตราการ

เพิ่มขึ้นของความเร็วในอุดมคติ (S_P=P) ตารางที่ 4.3 และรูปที่ 4.4 แสดงประสิทธิภาพในการบีบอัดข้อมูล ทั้ง 3 วิธี ($E_P = S_P/P$) เปรียบเทียบกับประสิทธิภาพในอุดมคติ (E_P =1)

ตารางที่ 4.1 เวลาที่ใช้ในการบีบอัดข้อมูล ทั้ง 3 วิธี เมื่อใช้หน่วยประมวลผล 2 หน่วยประมวลผล ประมวลผลบีบอัดแฟ้มข้อมูล 22hgp11.txt ซึ่งมีขนาดเท่ากับ 34,707,288 ไบต์

วิธีการบีบอัดข้อมูล	เวลาในการประมวลผล (วินาที)
อุดมคติ	8.7252
วิธีแบบขนาน	
MSBzip2	10.5782
AWBzip2	8.9070
AWWBzip2	8.8217

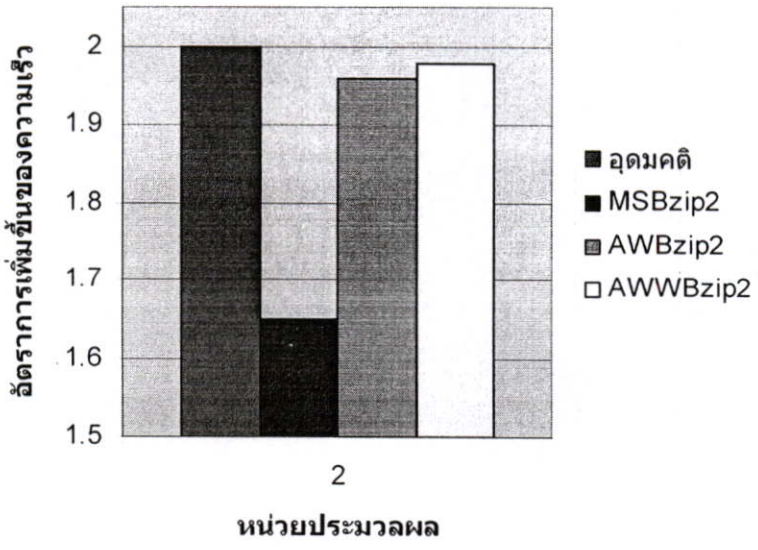


รูปที่ 4.2 เปรียบเทียบเวลาที่ใช้ในการบีบอัดข้อมูล ทั้ง 3 วิธี กับเวลาในอุดมคติ เมื่อ P=2

จากรูปที่ 4.2 สามารถอธิบายได้ว่าเวลาที่ใช้ในการประมวลผลวิธี AWWBzip2 ที่เสนอในวิทยานิพนธ์ เมื่อใช้หน่วยประมวลผล 2 หน่วยประมวลผล ประมวลผลบีบอัดแฟ้มข้อมูล 22hgp11.txt ซึ่งมีขนาดเท่ากับ 34,707,288 ไบต์จะใช้เวลาในการประมวลผล 8.8217 วินาทีใกล้เคียงกับเวลาในอุดมคติ คือ 8.7252 วินาที ซึ่งคิดเป็น 98.9% และวิธี MSBzip2 เป็นวิธีที่ใช้เวลาในการประมวลผลมากที่สุด คือ 10.5782 วินาที

ตารางที่ 4.2 อัตราการเพิ่มขึ้นของความเร็ว ทั้ง 3 วิธี เมื่อใช้หน่วยประมวลผล 2 หน่วยประมวลผล ประมวลผลบีบอัดแฟ้มข้อมูล 22hgp11.txt ซึ่งมีขนาดเท่ากับ 34,707,288 ไบต์

วิธีการบีบอัดข้อมูล	อัตราการเพิ่มขึ้นของความเร็ว
อุดมคติ	2
วิธีแบบขนาน	
MSBzip2	1.6496
AWMBzip2	1.9591
AWWPBzip2	1.9781

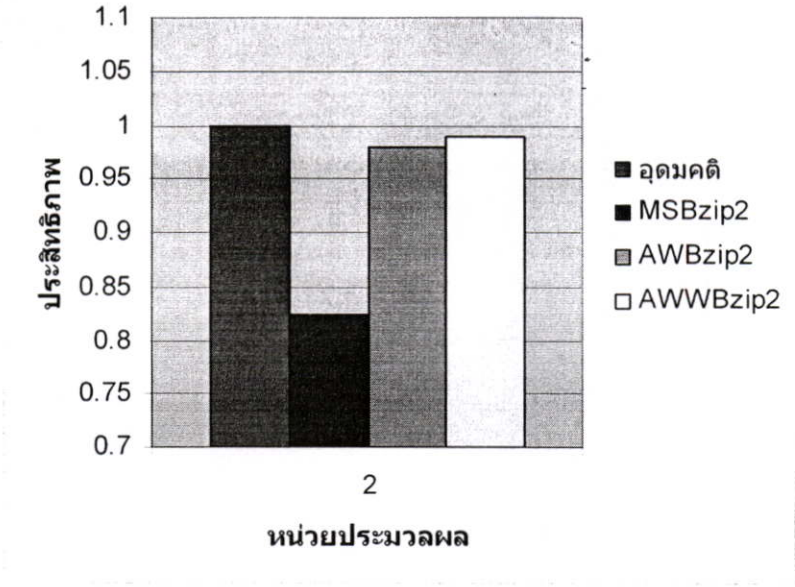


รูปที่ 4.3 เปรียบเทียบอัตราการเพิ่มขึ้นของความเร็วทั้ง 3 วิธี กับอัตราการเพิ่มขึ้นของความเร็วในอุดมคติ เมื่อ P=2

จากรูปที่ 4.3 สามารถอธิบายได้ว่าอัตราการเพิ่มขึ้นของความเร็วของการบีบอัดข้อมูล เมื่อใช้หน่วยประมวลผล 2 หน่วยประมวลผล ประมวลผลบีบอัดแฟ้มข้อมูล 22hgp11.txt ซึ่งมีขนาดเท่ากับ 34,707,288 ไบต์ วิธี AWWBzip2 เท่ากับ 1.9781 ซึ่งเป็นวิธีที่ดีที่สุด ส่วนวิธี MSBzip2 มีอัตราการเพิ่มขึ้นของความเร็วเท่ากับ 1.6496 ซึ่งน้อยกว่าทุกๆวิธี

ตารางที่ 4.3 ประสิทธิภาพ ทั้ง 3 วิธี เมื่อใช้หน่วยประมวลผล 2 หน่วยประมวลผล ประมวลผลบีบอัดแฟ้มข้อมูล 22hgp11.txt ซึ่งมีขนาดเท่ากับ 34,707,288 ไบต์

วิธีการบีบอัดข้อมูล	ประสิทธิภาพ
อุดมคติ	1
วิธีแบบขนาน	
MSBzip2	0.8248
AWMBzip2	0.9795
AWWPBzip2	0.9890



รูปที่ 4.4 เปรียบเทียบประสิทธิภาพทั้ง 3 วิธี กับประสิทธิภาพในอุดมคติ เมื่อ P=2

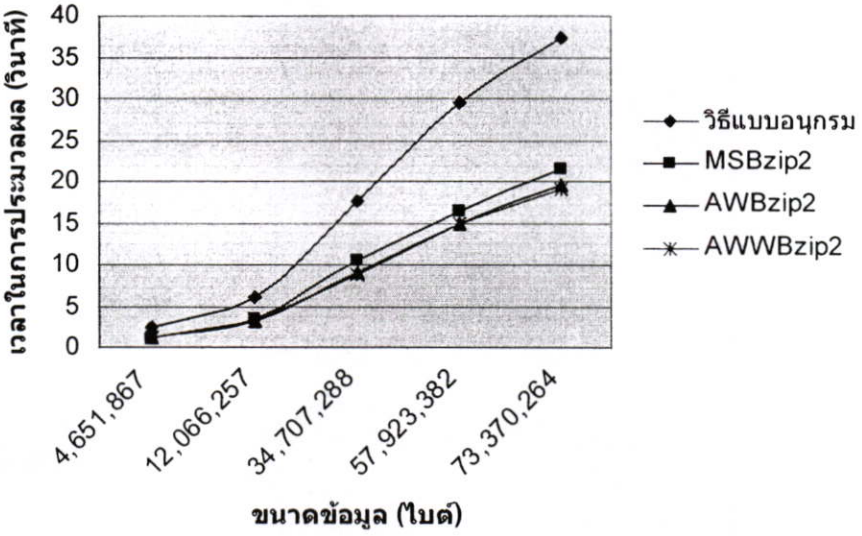
จากรูปที่ 4.4 แสดงประสิทธิภาพของการบีบอัดข้อมูลทั้ง 3 วิธี เมื่อใช้หน่วยประมวลผล 2 หน่วยประมวลผล ประมวลผลบีบอัดแฟ้มข้อมูล 22hgp11.txt ซึ่งมีขนาดเท่ากับ 34,707,288 ไบต์ จะเห็นว่าวิธี AWWBzip2 มีค่าประสิทธิภาพเท่ากับ 0.9890 จะให้ผลเข้าใกล้ 1 ส่วนวิธี MSBzip2 จะมีประสิทธิภาพเท่ากับ 0.8248 น้อยที่สุดเมื่อเทียบกับทุกๆวิธี

4.3.2 ผลการทดลองเปรียบเทียบเมื่อขนาดข้อมูลที่ใช้ทดลองต่างกัน

การทดลองเปรียบเทียบการบีบอัดข้อมูลเมื่อขนาดของข้อมูลที่ใช้ทดลองมีขนาดต่างกัน กำหนดให้จำนวนหน่วยประมวลผลที่ใช้ในการคำนวณเท่ากับ 2 หน่วยประมวลผล ทำการเปรียบเทียบวิธีการบีบอัดข้อมูลแบบอนุกรม (Sequential Compression) และวิธีการบีบอัดข้อมูลแบบขนานทั้ง 3 วิธี โดยบีบอัดแฟ้มข้อมูล ดังนี้ 00ws110.txt, 20hgp10.txt, 22hgp11.txt, 0xhgp10.txt และ 07hgp10.txt ซึ่งมีขนาด 4,651,867 ไบต์, 12,066,257 ไบต์, 34,707,288 ไบต์, 57,923,382 ไบต์ และ 73,370,264 ไบต์ ตามลำดับ ซึ่งผลการทดลองแสดงดังนี้ ตารางที่ 4.4 และรูปที่ 4.5 แสดงเวลาที่ใช้ในการประมวลผล นอกจากนี้เราสามารถนำเวลาที่ใช้ในการประมวลผลมาคำนวณหาอัตราการเพิ่มขึ้นของความเร็ว ดังแสดงในตารางที่ 4.5 และรูปที่ 4.6 จากนั้นนำค่าของอัตราการเพิ่มขึ้นของความเร็วมาคำนวณหาประสิทธิภาพ ดังแสดงในตารางที่ 4.6 และรูปที่ 4.7

ตารางที่ 4.4 เวลาที่ใช้ในวิธีการบีบอัดข้อมูลทั้ง 3 วิธี เมื่อใช้หน่วยประมวลผล 2 หน่วยประมวลผล บีบอัดแฟ้มข้อมูล ดังนี้ 00ws110.txt, 20hgp10.txt, 22hgp11.txt, 0xhgp10.txt และ 07hgp10.txt ซึ่งมีขนาด 4,651,867 ไบต์, 12,066,257 ไบต์, 34,707,288 ไบต์, 57,923,382 ไบต์ และ 73,370,264 ไบต์ ตามลำดับ

ขนาดข้อมูล (ไบต์)	วิธีแบบอนุกรม	วิธีแบบขนาน		
		MSBzip2	AWBzip2	AWWBzip2
4,651,867	2.4136	1.2826	1.2197	1.2157
12,066,257	6.1348	3.3675	3.1579	3.1059
34,707,288	17.4500	10.5782	8.9070	8.8217
57,923,382	29.4150	16.3253	14.9430	14.7927
73,370,264	37.2360	21.4778	19.4758	18.9934

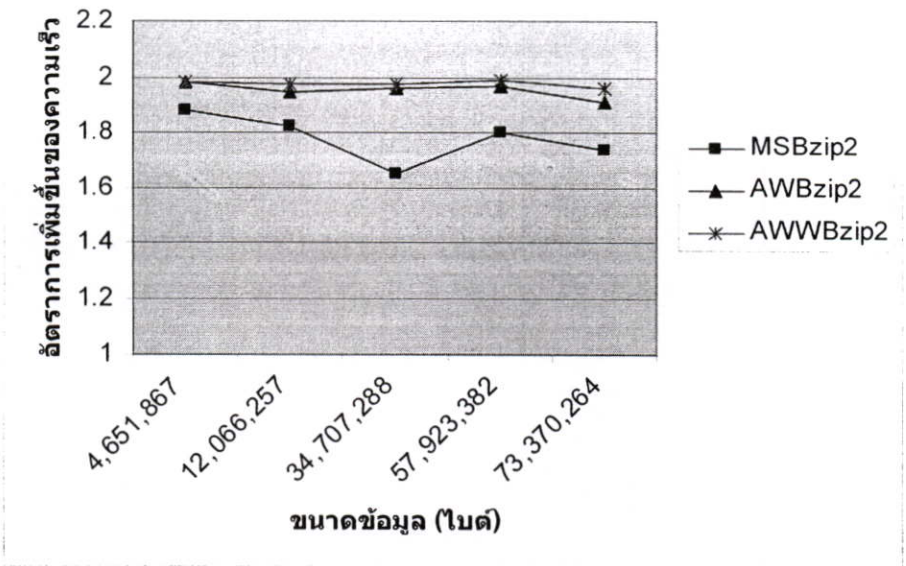


รูปที่ 4.5 เปรียบเทียบเวลาที่ใช้ในการบีบอัดข้อมูล ทั้ง 3 วิธี เมื่อขนาดข้อมูลที่ใช้ทดลองต่างกัน และP = 2

จากรูปที่ 4.5 แสดงเวลาที่ใช้ในการบีบอัดข้อมูล โดยทำการเปรียบเทียบเวลาทั้ง 3 วิธี เมื่อใช้หน่วยประมวลผล 2 หน่วยประมวลผล บีบอัดแฟ้มข้อมูล ดังนี้ 00ws110.txt, 20hgp10.txt, 22hgp11.txt, 0xhgp10.txt และ 07hgp10.txt ซึ่งมีขนาด 4,651,867 ไบต์, 12,066,257 ไบต์, 34,707,288 ไบต์, 57,923,382 ไบต์ และ 73,370,264 ไบต์ ตามลำดับ จะเห็นว่าเมื่อข้อมูลมีขนาดใหญ่ขึ้นเวลาที่ใช้ในการบีบอัดข้อมูลนั้นจะเพิ่มขึ้นตามไปด้วย โดยวิธี AWWBzip2 ใช้เวลาในการประมวลผลน้อยที่สุด และวิธี MSBzip2 ใช้เวลาในการประมวลผลมากกว่าวิธีอื่นๆที่เสนอ

ตารางที่ 4.5 อัตราการเพิ่มขึ้นของความเร็วทั้ง 3 วิธี เมื่อใช้หน่วยประมวลผล 2 หน่วยประมวลผล บีบอัดแฟ้มข้อมูล ดังนี้ 00ws110.txt, 20hgp10.txt, 22hgp11.txt, 0xhgp10.txt และ 07hgp10.txt ซึ่งมีขนาด 4,651,867 ไบต์, 12,066,257 ไบต์, 34,707,288 ไบต์, 57,923,382 ไบต์ และ 73,370,264 ไบต์ ตามลำดับ

ขนาดข้อมูล (ไบต์)	วิธีแบบขนาน		
	MSBzip2	AWBzip2	AWWBzip2
4,651,867	1.8817	1.9788	1.9853
12,066,257	1.8217	1.9426	1.9751
34,707,288	1.6496	1.9591	1.9781
57,923,382	1.8018	1.9684	1.9884
73,370,264	1.7336	1.9118	1.9604



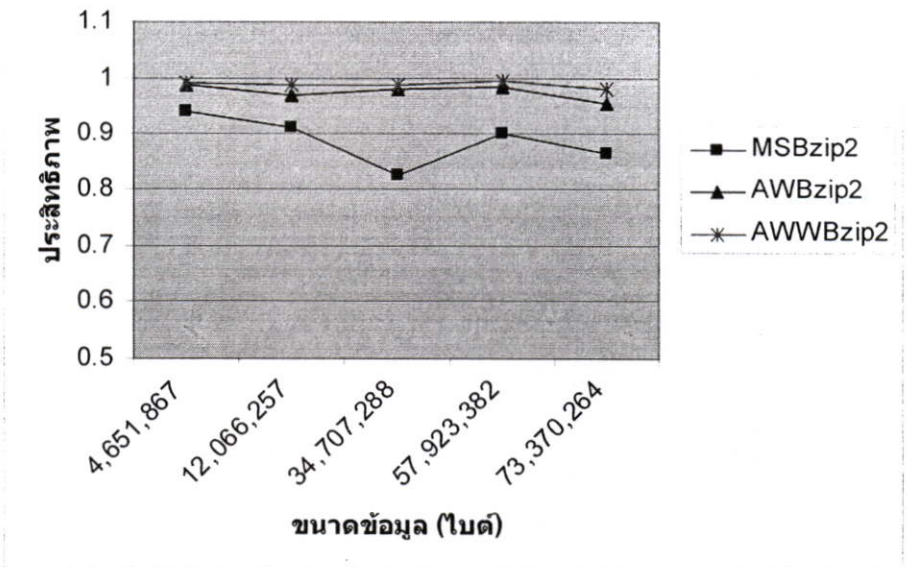
รูปที่ 4.6 เปรียบเทียบอัตราการเพิ่มขึ้นของความเร็วทั้ง 3 วิธี เมื่อขนาดข้อมูลที่ใช้ทดลองต่างกัน และP = 2

จากรูปที่ 4.6 แสดงการเปรียบเทียบอัตราการเพิ่มขึ้นของความเร็วทั้ง 3 วิธี เมื่อใช้หน่วยประมวลผล 2 หน่วยประมวลผล บีบอัดแฟ้มข้อมูล ดังนี้ 00ws110.txt, 20hgp10.txt, 22hgp11.txt, 0xhgp10.txt และ 07hgp10.txt ซึ่งมีขนาด 4,651,867 ไบต์, 12,066,257 ไบต์, 34,707,288 ไบต์,

57,923,382 ไบต์ และ 73,370,264 ไบต์ ตามลำดับ จะเห็นว่าวิธี AWWBzip2 มีอัตราการเพิ่มขึ้นของความเร็วมากที่สุด ส่วนวิธี MSBzip2 มีอัตราการเพิ่มขึ้นของความเร็วน้อยที่สุด

ตารางที่ 4.6 ประสิทธิภาพทั้ง 3 วิธี เมื่อใช้หน่วยประมวลผล 2 หน่วยประมวลผล บีบอัดแฟ้มข้อมูลดังนี้ 00ws110.txt, 20hgp10.txt, 22hgp11.txt, 0xhgp10.txt และ 07hgp10.txt ซึ่งมีขนาด 4,651,867 ไบต์, 12,066,257 ไบต์, 34,707,288 ไบต์, 57,923,382 ไบต์ และ 73,370,264 ไบต์ ตามลำดับ

ขนาดข้อมูล (ไบต์)	วิธีแบบขนาน		
	MSBzip2	AWBzip2	AWWBzip2
4,651,867	0.9408	0.9894	0.9926
12,066,257	0.9108	0.9713	0.9875
34,707,288	0.8248	0.9795	0.9890
57,923,382	0.9009	0.9842	0.9942
73,370,264	0.8668	0.9559	0.9802



รูปที่ 4.7 เปรียบเทียบประสิทธิภาพ ทั้ง 3 วิธี เมื่อขนาดข้อมูลที่ใช้ทดลองต่างกัน และP = 2

จากรูปที่ 4.7 แสดงการเปรียบเทียบประสิทธิภาพทั้ง 3 วิธี เมื่อใช้หน่วยประมวลผล 2 หน่วยประมวลผล บีบอัดแฟ้มข้อมูล ดังนี้ 00ws110.txt, 20hgp10.txt, 22hgp11.txt, 0xhgp10.txt และ 07hgp10.txt ซึ่งมีขนาด 4,651,867 ไบต์, 12,066,257 ไบต์, 34,707,288 ไบต์, 57,923,382 ไบต์

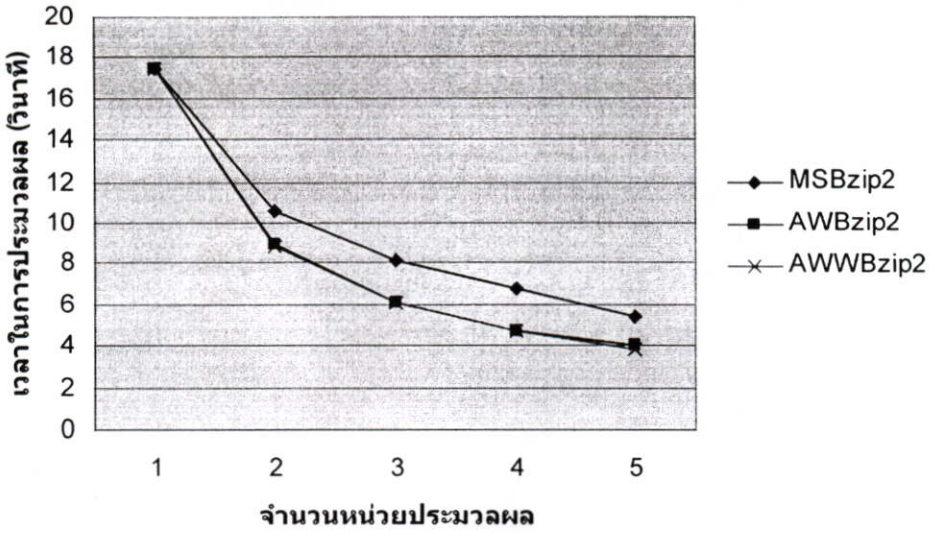
และ 73,370,264 ไบต์ ตามลำดับ จะเห็นว่าวิธี AWWBzip2 มีประสิทธิภาพมากที่สุด ส่วนวิธี MSBzip2 มีประสิทธิภาพน้อยที่สุด

4.3.3 ผลการทดลองเปรียบเทียบเมื่อจำนวนหน่วยประมวลผลที่ใช้ในการทดลองต่างกัน

ผลการทดลองเปรียบเทียบการบีบอัดข้อมูลเมื่อกำหนดแฟ้มข้อมูลคือ 22hgp11.txt ซึ่งมีขนาดเท่ากับ 34,707,288 ไบต์ และทำการเพิ่มจำนวนหน่วยประมวลผลตั้งแต่ 1 หน่วยประมวลผล 2 หน่วยประมวลผล 3 หน่วยประมวลผล 4 หน่วยประมวลผล และ 5 หน่วยประมวลผลตามลำดับ โดยจะพิจารณาเวลาทั้งหมดที่ใช้ในการประมวลผล ดังแสดงในตารางที่ 4.7 และรูปที่ 4.8 นอกจากนี้เราสามารถนำเวลาที่ใช้ในการประมวลผลมาคำนวณหาอัตราการเพิ่มขึ้นของความเร็ว ดังแสดงในตารางที่ 4.8 และรูปที่ 4.9 จากนั้นนำค่าของอัตราการเพิ่มขึ้นของความเร็วมาคำนวณหาประสิทธิภาพ ดังแสดงในตารางที่ 4.9 และรูปที่ 4.10

ตารางที่ 4.7 เวลาที่ใช้ในการประมวลผลทั้ง 3 วิธี โดยใช้หน่วยประมวลผลตั้งแต่ 1, 2, 3, 4 และ 5 หน่วยประมวลผล ตามลำดับ ทำการบีบอัดแฟ้มข้อมูล 22hgp11.txt ซึ่งมีขนาดเท่ากับ 34,707,288 ไบต์

จำนวนหน่วยประมวลผล	วิธีแบบขนาน		
	MSBzip2	AWBzip2	AWWBzip2
1	17.4504	17.4504	17.4504
2	10.5782	8.9070	8.8217
3	8.0914	6.0438	6.0379
4	6.7806	4.7017	4.6970
5	5.3943	4.0363	3.8658

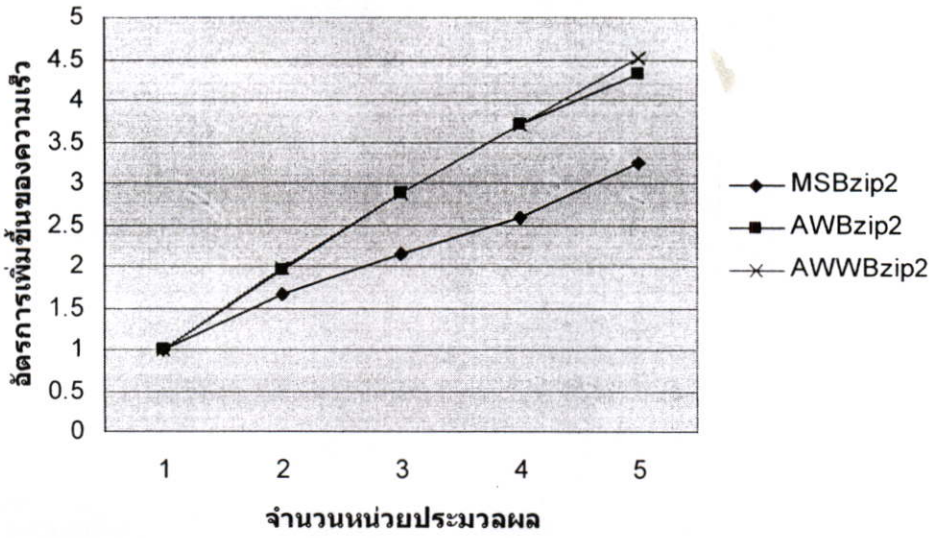


รูปที่ 4.8 เปรียบเทียบเวลาที่ใช้ในการประมวลผล เมื่อจำนวนหน่วยประมวลผลที่ใช้ในการทดลองต่างกัน

จากรูปที่ 4.8 แสดงเวลาที่ใช้ในการประมวลผล ซึ่งจะเห็นได้ว่าเมื่อเพิ่มจำนวนหน่วยประมวลผล เวลาในการประมวลผลจะลดลง โดยวิธี AWWBzip2 ใช้เวลาในการประมวลผลน้อยที่สุด เท่ากับ 8.8217 วินาที, 6.0379 วินาที, 4.6970 วินาที และ 3.8658 วินาที เมื่อจำนวนหน่วยประมวลผลเท่ากับ 2, 3, 4 และ 5 หน่วยประมวลผลตามลำดับ และวิธี MSBzip2 ใช้เวลาในการประมวลผลมากกว่าทุกๆวิธี

ตารางที่ 4.8 อัตราการเพิ่มขึ้นของความเร็ว ทั้ง 3 วิธี โดยใช้หน่วยประมวลผลตั้งแต่ 1, 2, 3, 4 และ 5 หน่วยประมวลผล ตามลำดับ ทำการบีบอัดแฟ้มข้อมูล 22hgp11.txt ซึ่งมีขนาดเท่ากับ 34,707,288 ไบต์

จำนวนหน่วยประมวลผล	**วิธีแบบขนาน**		
	MSBzip2	AWBzip2	AWWBzip2
1	1	1	1
2	1.6496	1.9591	1.9781
3	2.1566	2.8873	2.8901
4	2.5735	3.7114	3.7151
5	3.2349	4.3233	4.5139

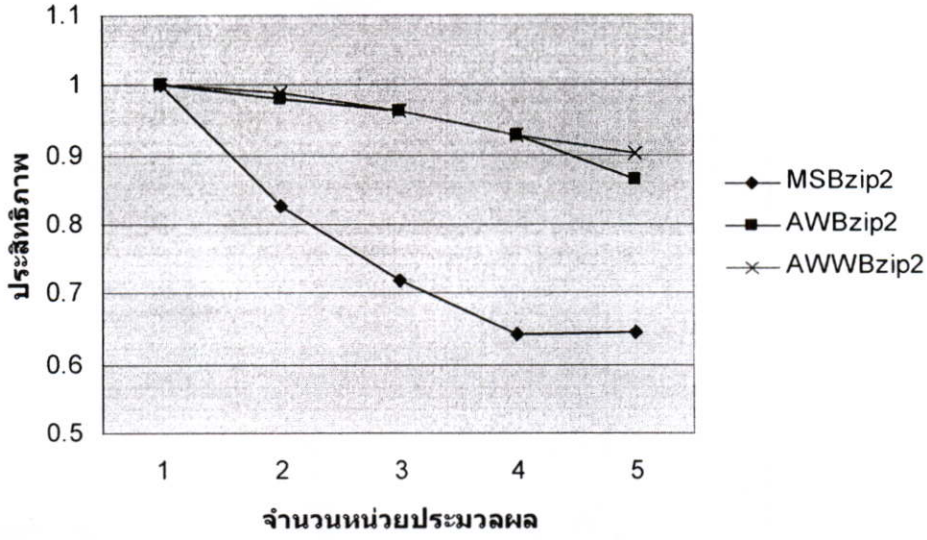


รูปที่ 4.9 เปรียบเทียบอัตราการเพิ่มขึ้นของความเร็ว เมื่อจำนวนหน่วยประมวลผลที่ใช้ในการทดลองต่างกัน

จากรูปที่ 4.9 แสดงอัตราการเพิ่มขึ้นของความเร็ว จะเห็นได้ว่าเมื่อเพิ่มจำนวนหน่วยประมวลผล อัตราการเพิ่มขึ้นของความเร็วจะเพิ่มขึ้นใกล้เคียงกับจำนวนหน่วยประมวลผล โดยวิธี AWWBzip2 เท่ากับ 1.9781, 2.8901, 3.7151 และ 4.5139 เมื่อจำนวนหน่วยประมวลผลเท่ากับ 2, 3, 4 และ 5 หน่วยประมวลผลตามลำดับ และวิธี MSBzip2 มีอัตราการเพิ่มขึ้นของความเร็วน้อยกว่าทุกๆวิธี

ตารางที่ 4.9 ประสิทธิภาพทั้ง 3 วิธี โดยใช้หน่วยประมวลผลตั้งแต่ 1, 2, 3, 4 และ 5 หน่วยประมวลผล ตามลำดับ ทำการบีบอัดแฟ้มข้อมูล 22hgp11.txt ซึ่งมีขนาดเท่ากับ 34,707,288 ไบต์

จำนวนหน่วยประมวลผล	**วิธีแบบขนาน**		
	MSBzip2	AWBzip2	AWWBzip2
1	1	1	1
2	0.8248	0.9795	0.9890
3	0.7188	0.9624	0.9633
4	0.6433	0.9278	0.9287
5	0.6469	0.8646	0.9027



รูปที่ 4.10 เปรียบเทียบประสิทธิภาพ

จากรูปที่ 4.10 แสดงผลการเปรียบเทียบประสิทธิภาพ วิธี AWWBzip2 ให้ประสิทธิภาพเข้าใกล้ 1 เท่ากับ 0.9890, 0.9633, 0.9287 และ 0.9027 เมื่อหน่วยประมวลผลเท่ากับ 2, 3, 4 และ 5 หน่วยประมวลผล ตามลำดับ เนื่องจากต้องใช้ระยะเวลาในการติดต่อระหว่างหน่วยประมวลผลเพิ่มมากขึ้นเมื่อเพิ่มจำนวนหน่วยประมวลผล ส่งผลให้ประสิทธิภาพลดลง ส่วนวิธี MSBzip2 มีค่าประสิทธิภาพน้อยกว่าทุกๆ วิธี

4.3.4 ผลการทดลองเปรียบเทียบขนาดข้อมูลที่ผ่านการบีบอัดข้อมูลด้วยวิธีต่างๆ

ผลการทดลองเปรียบเทียบขนาดข้อมูลที่ผ่านการบีบอัดข้อมูลด้วยวิธีต่างๆ เมื่อบีบอัดแฟ้มข้อมูล ดังนี้ 00ws110.txt, 20hgp10.txt, 22hgp11.txt, 0xhgp10.txt และ 07hgp10.txt ซึ่งมีขนาด 4,651,867 ไบต์, 12,066,257 ไบต์, 34,707,288 ไบต์, 57,923,382 ไบต์ และ 73,370,264 ไบต์ ตามลำดับ โดยประมวลผลตั้งแต่ 2 หน่วยประมวลผล 3 หน่วยประมวลผล 4 หน่วยประมวลผล และ 5 หน่วยประมวลผล ตามลำดับ มีผลแสดงในตารางที่ 4.10

ตารางที่ 4.10 ขนาดข้อมูลที่ผ่านการบีบอัดทั้ง 3 วิธี โดยใช้หน่วยประมวลผลตั้งแต่ 2, 3, 4 และ 5 หน่วยประมวลผล ตามลำดับ ทำการบีบอัดแฟ้มข้อมูล ดังนี้ 00ws110.txt, 20hgp10.txt, 22hgp11.txt, 0xhgp10.txt และ 07hgp10.txt ซึ่งมีขนาด 4,651,867 ไบต์, 12,066,257 ไบต์, 34,707,288 ไบต์, 57,923,382 ไบต์ และ 73,370,264 ไบต์ ตามลำดับ

ชื่อแฟ้มข้อมูล	ขนาดข้อมูลก่อนการบีบอัด	ขนาดข้อมูลหลังการบีบอัด ด้วยวิธีต่าง		
		msbzip2	awbzip2	awwbzip2
ประมวลผลการบีบอัดข้อมูลด้วย 2 หน่วยประมวลผล				
00ws110.txt	4,651,867	1,370,442	1,370,442	1,370,546
20hgp10.txt	12,066,257	3,296,022	3,296,022	3,295,629
22hgp11.txt,	34,707,288	9,241,934	9,241,934	9,239,995
0xhgp10.txt	57,923,382	15,774,308	15,774,308	15,772,779
07hgp10.txt	73,370,264	20,007,150	20,007,150	20,009,326
ประมวลผลการบีบอัดข้อมูลด้วย 3 หน่วยประมวลผล				
00ws110.txt	4,651,867	1,369,958	1,369,958	1,369,991
20hgp10.txt	12,066,257	3,296,072	3,296,072	3,296,145
22hgp11.txt,	34,707,288	9,240,212	9,240,212	9,239,808
0xhgp10.txt	57,923,382	15,771,424	15,771,424	15,770,780
07hgp10.txt	73,370,264	20,001,765	20,001,765	20,005,646
ประมวลผลการบีบอัดข้อมูลด้วย 4 หน่วยประมวลผล				
00ws110.txt	4,651,867	1,382,695	1,382,695	1,382,646
20hgp10.txt	12,066,257	3,297,507	3,297,507	3,297,233
22hgp11.txt,	34,707,288	9,240,465	9,240,465	9,241,781
0xhgp10.txt	57,923,382	15,778,491	15,778,491	15,773,721
07hgp10.txt	73,370,264	20,006,693	20,006,693	20,003,106
ประมวลผลการบีบอัดข้อมูลด้วย 5 หน่วยประมวลผล				
00ws110.txt	4,651,867	1,374,777	1,374,777	1,374,040
20hgp10.txt	12,066,257	3,296,830	3,296,830	3,297,506
22hgp11.txt,	34,707,288	9,240,722	9,240,722	9,238,140
0xhgp10.txt	57,923,382	15,768,194	15,768,194	15,769,714
07hgp10.txt	73,370,264	20,010,471	20,010,471	20,013,244

จากตารางที่ 4.10 พบว่า ขนาดข้อมูลที่ผ่านการบีบอัดข้อมูลด้วยวิธี MSBzip2 และการบีบอัดข้อมูลด้วยวิธี AWBzip2 มีขนาดข้อมูลที่ผ่านการบีบอัดแล้วเท่ากัน ส่วนวิธีการบีบอัดข้อมูลด้วยวิธี AWWBzip2 ขนาดข้อมูลที่ผ่านการบีบอัดแล้วอาจเพิ่ม หรือลดขนาดไปจากการบีบอัดข้อมูลวิธีอื่นๆ โดยเฉลี่ย 0.0013 % เมื่อเทียบกับขนาดแฟ้มข้อมูลที่ทำการบีบอัด ขนาดที่เพิ่มหรือลดของการบีบอัดข้อมูลวิธี AWWBzip2 เกิดขึ้นเนื่องจาก การแบ่งขนาดบล็อกข้อมูลย่อยที่จะบีบอัดข้อมูลไม่เท่ากัน นั่นคือ แต่ละหน่วยประมวลผลลำดับเดียวกันของการบีบอัดข้อมูลวิธี AWWBzip2 และวิธีการบีบอัดข้อมูลวิธีอื่นๆ จะมีขนาดข้อมูลที่จะทำการบีบอัดไม่เท่ากัน ทำให้ข้อมูลที่ผ่านการบีบอัดแล้วของแต่ละหน่วยประมวลผลลำดับเดียวกันมีขนาดไม่เท่ากัน ส่งผลให้ผลลัพธ์ของการบีบอัดข้อมูลวิธี AWWBzip2 มีขนาดแตกต่างกับการบีบอัดข้อมูลวิธีอื่นๆ

บทที่ 5
สรุปผลและแนวทางการพัฒนางานวิจัย

5.1 สรุปผลและวิเคราะห์ผลการทดลอง

การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูที่ได้เสนอไว้ในบทที่ 3 ซึ่งมีทั้งหมด 3 วิธี คือ 1) เอ็มเอสบีซิพทู (MSBzip2) 2) เอดับเบิลยูบีซิพทู (AWBzip2) และ 3) เอดับเบิลยูดับเบิลยูบีซิพทู (AWWBzip2) โดยวิธีแรกเป็นวิธีที่ใช้แนวคิดการบีบอัดข้อมูลแบบขนานวิธีพีบีซิพทู (PBzip2) ที่มีผู้ศึกษาไว้บนเครื่องคอมพิวเตอร์แบบขนานที่ใช้หน่วยความจำร่วม (Shared-Memory Parallel Architectures) แบบมีหน่วยประมวลผลควบคุม ส่วนวิธีที่ 2 และ 3 เป็นวิธีที่เสนอแนวคิดใหม่โดยไม่มีหน่วยประมวลผลควบคุม เพื่อลดการติดต่อระหว่างหน่วยประมวลผลควบคุมกับหน่วยประมวลผลบีบอัดข้อมูล ดังนั้นทุกหน่วยประมวลผลทำหน้าที่แบ่งข้อมูล บีบอัดข้อมูล และเขียนข้อมูล ส่วนวิธีที่ 3 จะต่างจากวิธีที่ 2 ที่มีการกำหนดน้ำหนักของการแบ่งบล็อกข้อมูลย่อย เพื่อให้บล็อกข้อมูลย่อยมีขนาดต่างกัน ทำให้แต่ละหน่วยประมวลผลใช้งานอุปกรณ์อินพุทเอาท์พุทศูนย์กลาง (Centralized I/O) ไม่พร้อมกัน

การทดลองที่ผ่านมาในบทที่ 4 ซึ่งเป็นการทดลองบนระบบพีซีคลัสเตอร์ เมื่อใช้หน่วยประมวลผลตั้งแต่ 1 หน่วยประมวลผล ถึง 5 หน่วยประมวลผล บีบอัดแฟ้มข้อมูล ดังนี้ 00ws110.txt, 20hgp10.txt, 22hgp11.txt, 0xhgp10.txt และ 07hgp10.txt ซึ่งมีขนาด 4,651,867 ไบต์, 12,066,257 ไบต์, 34,707,288 ไบต์, 57,923,382 ไบต์ และ 73,370,264 ไบต์ ตามลำดับ โดยนำผลการทดลองทั้งหมดมาหาค่าเฉลี่ย และสามารถสรุปผลการทดลองได้ดังต่อไปนี้

วิธีการบีบอัดข้อมูล	ประสิทธิภาพเฉลี่ย
MSBzip2	0.7376
AWBzip2	0.9402
AWWBzip2	0.9513

จากผลการทดลองบนระบบพีซีคลัสเตอร์ที่มีขนาด 1 ถึง 5 หน่วยประมวลผล พบว่า วิธีเอดับเบิลยูดับเบิลยูบีซิพทู (AWWBzip2) และวิธีเอดับเบิลยูเอ็มบีซิพทู (AWBzip2) มีสมรรถนะดีกว่าวิธีเอ็มเอสบีซิพทู (MSBzip2) อย่างน้อย 19.91% และ 18.75% ตามลำดับ ดังนั้น จากผลการทดลองสามารถสรุปได้ว่า การลดการติดต่อระหว่างหน่วยประมวลผลทำให้เวลาที่ใช้ในการติดต่อสื่อสารระหว่างหน่วยประมวลผลลดลง และการลดการใช้อินพุทเอาท์พุทศูนย์กลางพร้อมกัน โดยให้บล็อกข้อมูลย่อยมีขนาดต่างกัน ทำให้แต่ละหน่วยประมวลผลใช้งานอุปกรณ์อินพุทเอาท์พุทศูนย์กลางไม่

พร้อมกัน ส่งผลให้การบีบอัดข้อมูลแบบขนานด้วยวิธีบีซิพทูที่นำเสนอมีสมรรถนะ (Performance) เพิ่มขึ้น

5.2 แนวทางการพัฒนางานวิจัย

- ถ้านำโปรแกรมที่ได้พัฒนาขึ้นไปใช้กับระบบคลัสเตอร์แบบที่คอมพิวเตอร์ในระบบมีองค์ประกอบภายในที่แตกต่างกัน (Heterogeneous Cluster) จำเป็นต้องให้ความสำคัญในเรื่องการสร้างสมดุลของงาน (Load Balancing) ในระบบ เนื่องจากคอมพิวเตอร์ที่เชื่อมต่ออยู่ในระบบมีองค์ประกอบภายในที่แตกต่างกัน ทำให้คอมพิวเตอร์แต่ละเครื่องมีความสามารถและประสิทธิภาพในการประมวลผลงานที่แตกต่างกัน
- นำไปพัฒนาโปรแกรมให้สามารถใช้งานได้ง่ายขึ้นและใช้ได้หลายระบบปฏิบัติการ เช่น พัฒนาบนระบบปฏิบัติการไมโครซอฟท์วินโดวส์ เป็นต้น และสามารถนำโปรแกรมที่พัฒนาขึ้นไปประยุกต์ใช้งานกับระบบต่างๆ เช่น นำมาใช้งานร่วมกับระบบฐานข้อมูล เป็นต้น

เอกสารอ้างอิง

[1] Burrows, M. and Wheeler, D.J. **A Block-Sorting Lossless Data Compression Algorithm.** SRC Research Report 124. Digital Systems Research Center, Palo Alto, CA. 1994

[2] Gilbert, H. and Thomas, R. M. **Data Compression : techniques and applications hardware and software considerations.** 3rd ed. Chichester : John Wiley. 1991.

[3] Gilchrist, J. **Parallel Data Compression with BZIP2**. International Conference on Parallel and Distributed Computing and Systems (PDCS 2004). MIT Cambridge, USA. 2004.

[4] Gonzalez, R.C. and Wood, R.E. **Digital Image Processing.** Addison Wesley. 1992.

[5] Gonzalez-Smith, M.E. and Storer, J.A. **Parallel Algorithms for Data Compression**. Journal of the Association for Computing Machinery, Vol. 32, No. 2, April 1985. pp. 344-373.

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

[7] Harris, M. **The disk compression book.** Indianapolis, IN : Que. 1993.

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

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

[10] Nelson, M. **The Data Compression Book.** San Mateo, CA : M&T Books. 1991.

[11] Pfister, G.F. **In Search of Clusters - The Coming Battle in Lowly Parallel Computing**. New Jersey :Prentice Hall PTR. Prentice-Hall, Inc. 1995.

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

[13] Rosario, J.M. and Choudhary, A.N. **High-performance I/O for massively parallel computers: problems and prospects**. IEEE Computer Society, vol. 27, no. 3, March 1994. pp. 59-68.

[14] Seward, J. **The bzip2 and libbzip2.** [Online]. Available : http://www.bzip.org/. 2005.

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

[16] Timothy, C.B. John, G.C. and Ian, H.W. **Text compression.** Indianapolis, IN :Que. 1993.

[17] Wayner, P. **Compression algorithms for real programmers.** San Diego : Academic Press. 2000.

ภาคผนวก

ตัวอย่างโปรแกรม BzipC.c ที่ใช้ในงานวิจัย

วิธีการบีบอัดข้อมูลทุกวิธีในงานวิจัยนี้จะถูกเขียนอยู่ในโปรแกรมเดียวนี้ โดยวิธีการบีบอัดข้อมูลจะเรียกใช้แตกต่างกันขึ้นอยู่กับคำสั่งที่ใช้ในการทดลอง ดังกล่าวไว้แล้วในบทที่ 4

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <bzlib.h>
main(int argc, char *argv[])
{
        int start, end;
        int count;
        int length;
        int error;
        char* buffer;
        char* outbuf;
        int nprocs;
        int myrank;
        int outsize,insize,sizeview;
        double starttime, finishtime, starttime1, finishtime1;
        char *filename;
        char *filenamebuf;
        int i, size;
        double weight, Pweight;
        int algorithm;
        MPI_Status   status;
        MPI_File     fh;
        MPI_Offset   filesize;
        MPI_Init(&argc, &argv);
```

```
starttime = MPI_Wtime();
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
algorithm = atoi(argv[2]);
if (algorithm == 1) // MSBzip2
{
        if (myrank == 0)  //Master
        {
                filename = argv[1];
                filenamebuf = (char *)malloc((strlen(filename)+7)*sizeof(char));
                sprintf(filenamebuf,"%s.bz2", filename);
                MPI_File_open(MPI_COMM_SELF, filename,
MPI_MODE_RDONLY | MPI_MODE_UNIQUE_OPEN, MPI_INFO_NULL, &fh);
                MPI_File_get_size(fh, &filesize);
                filesize = filesize/sizeof(char);
                length = filesize / nprocs + 1;
                for (i=1; i<=nprocs-1; i++ )
                {
                        buffer = (char *)malloc(length * sizeof(char));
                        MPI_File_set_view(fh, i*length * sizeof(char),MPI_CHAR,
MPI_CHAR, "native", MPI_INFO_NULL);
                        MPI_File_read(fh, buffer, length, MPI_CHAR, &status);
                        MPI_Get_count(&status, MPI_CHAR, &count);
                        MPI_Send(&count, 1, MPI_INT, i, 1,
MPI_COMM_WORLD );
                        MPI_Send(buffer, count, MPI_CHAR, i, 1,
MPI_COMM_WORLD );
                }
                buffer = (char *)malloc(length * sizeof(char));
                MPI_File_set_view(fh, 0,MPI_CHAR, MPI_CHAR, "native",
MPI_INFO_NULL);
                MPI_File_read(fh, buffer, length, MPI_CHAR, &status);
```

```
                MPI_Get_count(&status, MPI_CHAR, &count);
                MPI_File_close(&fh);
        }
        else // Slave
        {
                MPI_Recv(&count, 1, MPI_INT, 0, 1, MPI_COMM_WORLD,
&status);
                buffer = (char *)malloc(count* sizeof(char));
                MPI_Recv(buffer, count, MPI_CHAR, 0, 1, MPI_COMM_WORLD,
&status);
        }
        outbuf = (char *)malloc((count + 600 +(count/100)) * sizeof(char));
        BZ2_bzBuffToBuffCompress (outbuf, &outsize, buffer, count, 9, 0, 30 );
        if (myrank == 0)  //Master
        {
                MPI_File_open(MPI_COMM_SELF, filenamebuf,
MPI_MODE_WRONLY | MPI_MODE_CREATE, MPI_INFO_NULL, &fh);
                MPI_File_set_view(fh, 0,MPI_CHAR, MPI_CHAR, "native",
MPI_INFO_NULL);
                MPI_File_write(fh, outbuf, outsize, MPI_CHAR,
MPI_STATUS_IGNORE);
                sizeview=outsize;
                for (i=1; i<=nprocs-1; i++ )
                {
                        MPI_File_set_view(fh, sizeview,MPI_CHAR, MPI_CHAR,
"native", MPI_INFO_NULL);
                        MPI_Recv(&outsize, 1, MPI_INT, i, 1,
MPI_COMM_WORLD, &status);
                        sizeview=sizeview+outsize;
                        outbuf = (char *)malloc(outsize* sizeof(char));
                        MPI_Recv(outbuf, outsize, MPI_CHAR, i, 1,
MPI_COMM_WORLD, &status);
```

```
                        MPI_File_write(fh, outbuf, outsize, MPI_CHAR,
MPI_STATUS_IGNORE);
                }
                MPI_File_close(&fh);
        }
        else  // Slave
        {
                MPI_Send(&outsize, 1, MPI_INT, 0, 1, MPI_COMM_WORLD );
                MPI_Send(outbuf, outsize, MPI_CHAR, 0, 1, MPI_COMM_WORLD );
        }
        finishtime = MPI_Wtime();
        printf ("Process %d Total Time is : %5.8f \n",myrank ,finishtime-starttime);
}
if (algorithm == 2) //AWBzip2
{
        filename = argv[1];
        filenamebuf = (char *)malloc((strlen(filename)+7)*sizeof(char));
        sprintf(filenamebuf,"%s.bz2", filename);
        MPI_File_open(MPI_COMM_SELF, filename, MPI_MODE_RDONLY |
MPI_MODE_UNIQUE_OPEN, MPI_INFO_NULL, &fh);
        MPI_File_get_size(fh, &filesize);
        filesize = filesize/sizeof(char);
        length = filesize / nprocs + 1;
        buffer = (char *)malloc(length * sizeof(char));
        MPI_File_set_view(fh, myrank*length * sizeof(char),MPI_CHAR, MPI_CHAR,
"native", MPI_INFO_NULL);
        MPI_File_read(fh, buffer, length, MPI_CHAR, &status);
        MPI_Get_count(&status, MPI_CHAR, &count);
        MPI_File_close(&fh);
        outbuf = (char *)malloc((count + 600 +(count/100)) * sizeof(char));
        BZ2_bzBuffToBuffCompress (outbuf, &outsize, buffer, count, 9, 0, 30 );
        if (myrank==0)
```

```
                {
                        start=0;
                }
                if (myrank>0)
                {
                        MPI_Recv(&start, 1, MPI_INT, myrank-1, 1, MPI_COMM_WORLD,
&status);
                }
                if (myrank<nprocs-1)
                {
                        size=start+outsize;
                        MPI_Send(&size, 1, MPI_INT, myrank+1, 1, MPI_COMM_WORLD );
                }
                MPI_File_open(MPI_COMM_SELF, filenamebuf, MPI_MODE_WRONLY |
MPI_MODE_CREATE, MPI_INFO_NULL, &fh);
                MPI_File_set_view(fh, start, MPI_CHAR, MPI_CHAR, "native",
MPI_INFO_NULL);
                MPI_File_write(fh, outbuf, outsize, MPI_CHAR, MPI_STATUS_IGNORE);

                MPI_File_close(&fh);
                finishtime = MPI_Wtime();
                printf ("Process %d Total Time is : %5.8f \n",myrank ,finishtime-starttime);
        }
        if (algorithm == 3) //AWWBzip2
        {
                filename = argv[1];
                filenamebuf = (char *)malloc((strlen(filename)+7)*sizeof(char));
                sprintf(filenamebuf,"%s.bz2", filename);
                weight = atoi(argv[3]);
                MPI_File_open(MPI_COMM_SELF, filename, MPI_MODE_RDONLY,
MPI_INFO_NULL, &fh);
                MPI_File_get_size(fh, &filesize);
```

```
filesize = filesize/sizeof(char);
Pweight = filesize * weight / 1000;
length = (filesize - nprocs * (nprocs - 1) * Pweight / 2 ) / nprocs + 1;
start=0;
for (i=0; i<myrank; i++ )
{
        length = length + i * Pweight;
        start = start+length;
}
length = length + myrank * Pweight;
buffer = (char *)malloc(length * sizeof(char));
MPI_File_set_view(fh, start, MPI_CHAR, MPI_CHAR, "native",
MPI_INFO_NULL);
MPI_File_read(fh, buffer, length, MPI_CHAR, &status);
MPI_Get_count(&status, MPI_CHAR, &count);
MPI_File_close(&fh);
outbuf = (char *)malloc((count + 600 +(count/100)) * sizeof(char));
BZ2_bzBuffToBuffCompress (outbuf, &outsize, buffer, count, 9, 0, 30 );
if (myrank==0)
{
        start=0;
}
if (myrank>0)
{
        MPI_Recv(&start, 1, MPI_INT, myrank-1, 1, MPI_COMM_WORLD,
&status);
}
if (myrank<nprocs-1)
{
        size=start+outsize;
        MPI_Send(&size, 1, MPI_INT, myrank+1, 1, MPI_COMM_WORLD );
}
```

```
                MPI_File_open(MPI_COMM_SELF, filenamebuf, MPI_MODE_WRONLY |
MPI_MODE_CREATE, MPI_INFO_NULL, &fh);
                MPI_File_set_view(fh, start, MPI_CHAR, MPI_CHAR, "native",
MPI_INFO_NULL);
                MPI_File_write(fh, outbuf, outsize, MPI_CHAR, MPI_STATUS_IGNORE);
                MPI_File_close(&fh);
                finishtime = MPI_Wtime();
                printf ("Process %d Total Time is : %5.8f \n",myrank ,finishtime-starttime);
        }
        MPI_Finalize();
}
```

ประวัติผู้เขียน

ชื่อ – สกุล	นายอัศวิน นิ่มกร
วัน เดือน ปีเกิด	13 มกราคม 2524
ที่อยู่	41/1 หมู่ 2 ตำบลประจันตคาม
	อำเภอประจันตคาม จังหวัดปราจีนบุรี 25130
ประวัติการศึกษา	
	2547 จบการศึกษาประกาศนียบัตรบัณฑิตวิชาชีพครู
	สาขาการศึกษา สถาบันราชภัฏพระนครศรีอยุธยา
	2546 จบการศึกษาปริญญาวิทยาศาสตรบัณฑิต (เกียรตินิยมอันดับ 2)
	สาขาวิทยาการคอมพิวเตอร์ สถาบันราชภัฏพระนครศรีอยุธยา
ทุนการศึกษา	
พ.ศ. 2547 - 2549	โครงการพัฒนาครูแกนนำในสาขาคณิตศาสตร์ วิทยาศาสตร์และ
	เทคโนโลยีและสิ่งแวดล้อม (Project Guiding Light)
พ.ศ. 2542 - 2547	โครงการส่งเสริมการผลิตครูที่มีความสามารถพิเศษทางวิทยาศาสตร์และ
	คณิตศาสตร์ (สควค.)