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

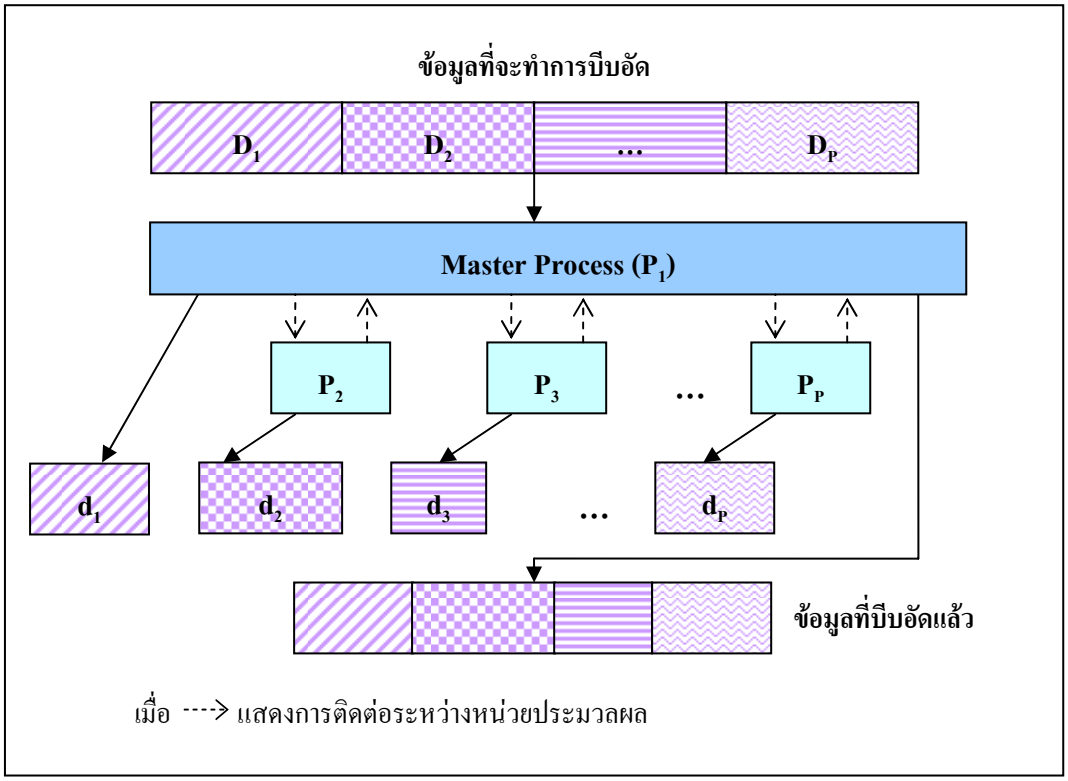
ให้ N คือ ขนาดของข้อมูลที่จะทำการบีบอัดข้อมูล D_P คือ บล็อกข้อมูลย่อยของหน่วยประมวลผลที่ i และ P คือ จำนวนหน่วยประมวลผลทั้งหมด สามารถเขียนได้ดังสมการ

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

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, ..., P_P ตามลำดับ บล็อกข้อมูลย่อยของแต่ละหน่วยประมวลผลที่ผ่านการบีบอัดข้อมูลแล้วจะมีลำดับ ดังนี้ d_1, d_2, d_3, ..., 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

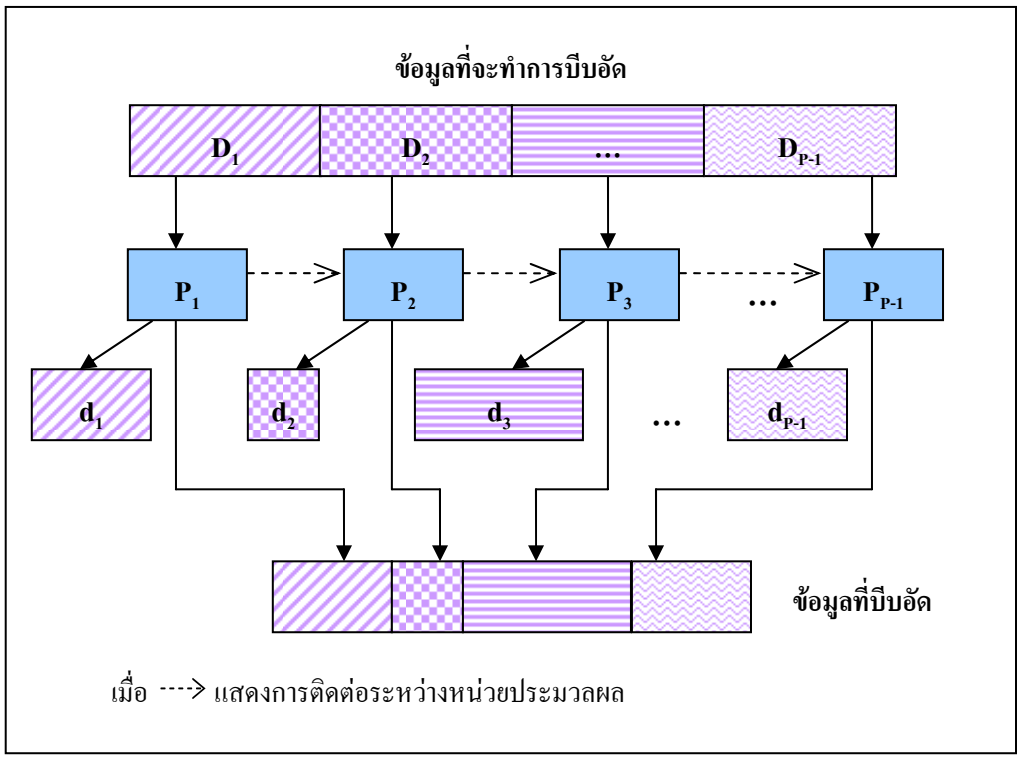
ให้ N คือ ขนาดของข้อมูลที่จะทำการบีบอัดข้อมูล D_P คือ บล็อกข้อมูลย่อยของหน่วยประมวลผลที่ i และ P คือ จำนวนหน่วยประมวลผลทั้งหมด สามารถเขียนได้ดังสมการ

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

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

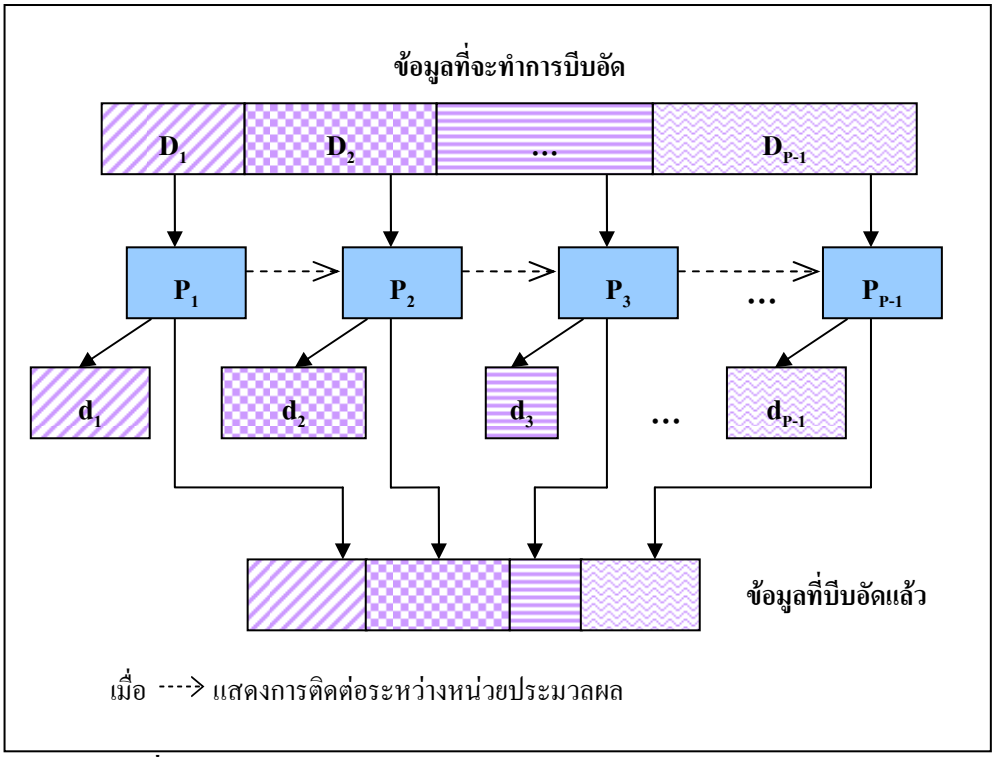
$$B = \frac{N - \sum_{i=1}^{P} W_i}{P}$$

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

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