ตัวสร้างรหัสสำหรับโปรแกรมแบบขนานเพื่อไมโครคอนโทรลเลอร์แบบหลายแกน

นายวัฒนา พรสูงส่ง

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2554

A CODE GENERATOR FOR PARALLEL PROGRAMS FOR MULTI-CORE MICROCONTROLLERS

Mr. Wattana Pornsoongsong

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering Program in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2011

หัวข้อวิทยานิพนธ์	ตัวสร้างรหัสสำหรับโปรแกรมแบบขนานเพื่อไมโครคอนโทรลเลอร์แบบหลายแกน
โดย	นายวัฒนา พรสูงส่ง
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ศาสตราจารย์ ดร.ประภาส จงสถิตย์วัฒนา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

…….. คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร.บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

……. ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.วีระ เหมืองสิน)

…….. อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(ศาสตราจารย์ ดร.ประภาส จงสถิตย์วัฒนา)

……. กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร.วรเศรษฐ สุวรรณิก)

วัฒนา พรสูงส่ง : ตัวสร้างรหัสสำหรับโปรแกรมแบบขนานเพื่อไมโครคอนโทรลเลอร์แบบหลายแกน. (A Code Generator for Parallel Programs for Multi-Core Microcontrollers) อ. ที่ปรึกษาวิทยานิพนธ์หลัก : ศ. ดร.ประภาส จงสถิตย์วัฒนา, 86 หน้า.

ปัจจุบันไมโครคอนโทรลเลอร์ได้รับการพัฒนาให้มีขีดความสามารถสูงมากขึ้น ไมโครคอนโทรลเลอร์แบบหลายแก่นเป็นไมโครคอนโทรลเลอร์ที่มีสถาปัตยกรรมแบบใหม่โดยแต่ละหน่วยประมวลผลสามารถทำงานได้อย่างพร้อม ๆ กัน ในการเขียนโปรแกรมแบบขนานบนระบบสถาปัตยกรรมแบบใหม่ด้วยภาษาที่เฉพาะกับสถาปัตยกรรมนั้นจึงเป็นสิ่งที่ยากและต้องการทรัพยากรต่าง ๆ เป็นอันมากโดยเฉพาะสำหรับผู้เริ่มต้นรวมถึงการเขียนโปรแกรมที่อาจมีความซับซ้อนสูงและมีข้อจำกัดทางด้านเวลาในการพัฒนาโปรแกรม ดังนั้นการมีตัวแปลโปรแกรมที่สามารถแปลภาษาที่ได้รับความนิยมสูง เช่น ภาษาซี ไปเป็นภาษาที่ใช้อยู่บนสถาปัตยกรรมแบบใหม่ที่สามารถแปลโปรแกรมให้สามารถสั่งการหน่วยประมวลผลต่าง ๆ ให้ทำงานในแบบขนานได้อย่างอัตโนมัติโดยใช้คำสั่งตัวแปลภาษาหรือคอมไพเลอร์ไดเรคทีฟ จะช่วยเพิ่มศักยภาพในการพัฒนาโปรแกรม ดึงดูดผู้ที่สนใจในการศึกษา พัฒนารวมถึงการวิจัยบนสถาปัตยกรรมแบบใหม่นี้ อีกทั้งยังแสดงให้เห็นถึงความก้าวหน้าในการวิจัยที่เป็นการต่อยอดงานวิจัยบนสถาปัตยกรรมแบบใหม่ที่สามารถทำงานในแบบขนานกันได้

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา 2554

5270493821 : MAJOR COMPUTER ENGINEERING

KEYWORDS : COMPILER / MICROCONTROLLER / PARALLEL

WATTANA PORNSOONGSONG : A CODE GENERATOR FOR PARALLEL PROGRAMS FOR MULTI-CORE MICROCONTROLLERS. ADVISOR : PROF. PRABHAS CHONGSTITVATANA, Ph.D., 86 pp.

Nowadays, microcontroller has been developed for the higher capabilities. Multi-core microcontroller has the new architecture which each core can be run in parallel. For creating a program on the new architecture with the specific language to the architecture, it is very complicated and required much resources especially the beginner and building a complex program with the limited time resources. Thus having the well-known compiler language such as C language on the new architecture which can compile any program to be executed and run in every core in parallel automatically by using compiler directive will increase potential and performance in the program development, attract the program developer for learning, developing and also researching on this new multi-core architecture. This also reflects the advancement in the research which is the research on the latest multi-core architecture which capable of running programs in parallel and simultaneously.

Department : Computer Engineering Student's Signature .....

Field of Study : Computer Engineering Advisor's Signature .....

Academic Year : 2011

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดีจากความช่วยเหลืออย่างดียิ่งของศาสตราจารย์ ดร. ประภาส จงสถิตย์วัฒนา อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ได้ให้ความรู้ ความคิด คำชี้แนะและแนะนำ ตลอดจนข้อคิดเห็นต่าง ๆ รวมถึงแรงบันดาลใจอันสำคัญยิ่ง ผู้วิจัยจึงขอขอบพระคุณเป็นอย่างสูง

ผู้วิจัยขอขอบพระคุณภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สำหรับทุนอัจฉริยะคืนรังสนับสนุนการเรียนการวิจัยในระดับปริญญามหาบัณฑิตของผู้วิจัยที่ทำให้ลดภาระค่าใช้จ่ายในการเรียนได้เป็นอย่างมาก อีกทั้งผู้วิจัยขอขอบคุณรุ่นพี่ เพื่อน และรุ่นน้องนิสิตปริญญามหาบัณฑิต และรุ่นน้องปริญญาบัณฑิตที่คอยให้ความช่วยเหลือ ให้ความคิดเห็นและให้กำลังใจ ซึ่งเป็นแรงผลักดันที่สำคัญสำหรับวิทยานิพนธ์ฉบับนี้เป็นอย่างมาก

ท้ายสุดนี้ผู้วิจัยขอกราบขอบพระคุณบิดามารดา ครอบครัว และทุก ๆ คนที่ได้ให้กำลังใจและให้การสนับสนุนมาโดยตลอดและเสมอมาแก่ผู้วิจัยจนกระทั่งสำเร็จการศึกษา

สารบัญ

บทที่ หน้า

สารบัญตาราง

สารบัญภาพ

ภาพที่ หน้า

ภาพที่ หน้า

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันหน่วยประมวลผลต่าง ๆ ได้รับการพัฒนาอย่างมาก จากแต่เดิมที่มีขนาดบิตน้อย ๆ จนมาเป็นแบบ 64 บิตแบบ amd-64 ที่กำลังได้รับความนิยมแพร่หลายเป็นอย่างมากในปัจจุบัน สิ่งที่โดดเด่นของหน่วยประมวลผลในปัจจุบันคงจะต้องเป็นสถาปัตยกรรมแบบหลายหน่วยประมวลผลที่ทำให้ความเร็วในการทำงานของเครื่องคอมพิวเตอร์ทั้งหลายเร็วมากขึ้นกว่าในอดีตเป็นอย่างมาก ในลักษณะเดียวกันหน่วยประมวลผลขนาดเล็กที่เรียกว่าไมโครคอนโทรลเลอร์ซึ่งในปัจจุบันได้มีการพัฒนาสถาปัตยกรรมให้มีหลายหน่วยประมวลผลอยู่ภายใน ทำงานสามารถทำงานแบบขนานไปพร้อมกันได้ ด้วยสถาปัตยกรรมในลักษณะนี้สามารถออกแบบให้มีความเร็วตอบสนองเพิ่มขึ้นได้โดยการแบ่งงานไปให้กับหน่วยประมวลผลตัวอื่นช่วยกันคำนวณประมวลผลข้อมูล หรืออาจทำให้การส่งสัญญาณควบคุมกลไกต่าง ๆ เป็นไปได้อย่างสะดวกมากขึ้นโดยการให้หน่วยประมวลผลแต่ละตัวกำกับการทำงานของแต่ละส่วนที่แยกออกจากกันได้ไม่ต้องทำงานหลาย ๆ งานในหน่วยประมวลผลเพียงหน่วยเดียว อีกทั้งยังสามารถลดต้นทุนหากในงานเดิมต้องใช้ไมโครคอนโทรลเลอร์หลายหน่วยซึ่งยากต่อการควบคุมมาเป็นการใช้หน่วยประมวลผลแบบหลายหน่วยที่อยู่ภายในชิปเพียงตัวเดียวทำให้พัฒนาโปรแกรมได้ง่ายยิ่งขึ้น

สำหรับในการเขียนโปรแกรมบนไมโครคอนโทรลเลอร์ซึ่งโดยทั่วไปจะมีโปรแกรมที่มีมาให้หรือสามารถหาได้ในอินเทอร์เนท จะเป็นภาษาเฉพาะที่ใช้ในไมโครคอนโทรลเลอร์ของแต่ละบริษัทที่ผลิต เช่นเดียวกันกับชิปพร็อพเพลเลอร์[1] ที่พัฒนาโดยบริษัทพาราแล็กซ์ ผู้พัฒนาโปรแกรมจะใช้ภาษาสปินในการสร้างและพัฒนาโปรแกรมซึ่งเป็นภาษาที่ไม่ซับซ้อนมากนักและมีโครงสร้างภาษาที่คล้ายกับภาษาสคริปท์ในปัจจุบันเช่น ภาษาวิชวลเบสิก แต่ภาษานี้ก็จะมีความซับซ้อนเพิ่มมากขึ้นในกรณีที่สร้างโปรแกรมให้ทำงานในแบบขนานที่ผู้เขียนโปรแกรมจะต้องควบคุมการทำงาน

ของหน่วยประมวลผลแต่ละหน่วยเอง ซึ่งงานวิจัยนี้ได้เห็นความสำคัญของภาษาที่ใช้ในการสร้างและพัฒนาโปรแกรมที่ควรจะเป็นภาษาที่คุ้นเคยทำให้ช่วยดึงดูดความสนใจในการเริ่มต้นศึกษาพัฒนาไปจนถึงการทำการวิจัย ซึ่งจะช่วยให้การสร้างและพัฒนาโปรแกรมเป็นไปโดยไม่ยากนัก ใช้เวลาในการเริ่มต้นศึกษาไม่มากอีกทั้งตัวภาษาเองจะมีคำสั่งพิเศษที่จะช่วยให้การสร้างโปรแกรมที่ทำงานแบบขนานได้ง่ายขึ้น ไม่ต้องใช้ภาษาสปินที่มีความยากลำบากมากกว่า

1.2 วัตถุประสงค์ของการวิจัย

พัฒนาตัวสร้างรหัสที่ใช้สำหรับการแปลงเป็นภาษาสปินทั้งในโปรแกรมที่ทำงานแบบอนุกรมและแบบขนาน

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1. ช่วยลดระยะเวลาและความซับซ้อนในการสร้างและพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์ทั้งในการทำงานแบบปกติและแบบขนาน
2. เพื่อช่วยเปิดและขยายโอกาสในการวิจัยและพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์แบบหลายแก่นทั้งในการทำงานแบบปกติและแบบขนาน
3. ลดความยุ่งยากและซับซ้อนรวมถึงเพิ่มความน่าสนในในการศึกษาไมโครคอนโทรลเลอร์ที่สามารถทำงานแบบขนานได้
4. โปรแกรมที่สร้างสามารถทำงานในแบบขนานได้อย่างอัตโนมัติ
5. เพิ่มโอกาสและความน่าสนใจในการพัฒนาและวิจัยโปรแกรมบนไมโครคอนโทรลเลอร์แบบหลายหน่วยประมวลผล

1.4 วิธีดำเนินการวิจัย

1. ศึกษาไมโครคอนโทรลเลอร์ทั่วไป
2. ศึกษาไมโครคอนโทรลเลอร์พร็อพเพลเลอร์ที่มีสถาปัตยกรรมพิเศษแบบหลายหน่วยประมวลผล
3. ศึกษาการทำงานของโปรแกรมบนไมโครคอนโทรลเลอร์ทั้งในแบบทั่วไปและแบบหลายหน่วยประมวลผล
4. ออกแบบและสร้างโปรแกรมแบบปกติและแบบขนานบนไมโครคอนโทรลเลอร์พร็อพเพลเลอร์
5. ศึกษาการแปลโปรแกรม และตัวแปลโปรแกรมสำหรับโปรแกรมแบบขนาน
6. ออกแบบคำสั่งบนตัวแปลโปรแกรมสำหรับโปรแกรมแบบขนาน
7. สร้างและทดสอบคำสั่งแบบขนานบนตัวแปลโปรแกรมเพื่อสร้างโปรแกรมทำงานแบบขนาน

บทที่ 2

หลักการและทฤษฎีที่เกี่ยวข้อง

2.1 ไมโครคอนโทรลเลอร์

พร็อพเพลเลอร์เป็นไมโครคอนโทรลเลอร์ที่ผลิตขึ้นโดยบริษัทพาราแล็กซ์ โดยไมโครคอนโทรลเลอร์นี้มีสถาปัตยกรรมพิเศษคือมีหลายหน่วยประมวลผลอยู่ภายในโดยมีถึง 8 หน่วยประมวลผลสามารถทำงานที่แยกกันและร่วมกันทำงานได้ตามที่โปรแกรมไว้

2.1.1 หน่วยความจำหลักและหน่วยความจำภายใน

สถาปัตยกรรมของไมโครคอนโทรลเลอร์พร็อพเพลเลอร์มีหน่วยความจำอยู่สองชนิด คือ หน่วยความจำหลักและหน่วยความจำภายใน โดยหน่วยความจำหลักมีขนาด 64 กิโลไบต์ ประกอบไปด้วยหน่วยความจำชนิดแรม (RAM) อยู่ 32 กิไลไบต์ หรือ 8,192 เวิร์ด และหน่วยความจำชนิดรอม (ROM) อยู่ 32 กิไลไบต์ หรือ 8,192 เวิร์ด เช่นเดียวกัน และหน่วยความจำภายในซึ่งอยู่ในทุก ๆ หน่วยประมวลผลที่เรียกว่า ค็อก (cog) จะมีอยู่ทั้งสิ้น 2 กิโลไบต์ หรือ 512 เวิร์ด

2.1.2 รายละเอียดของพร็อพเพลเลอร์

ชิปพร็อพเพลเลอร์ที่ใช้นี้เป็นโมเดลหรือรุ่น P8x32A ซึ่งต้องการใช้พลังงานไฟฟ้ากระแสตรงซึ่งมีแรงดัน 3.3 โวลต์ในการทำงานและกระแสไฟฟ้าสูงสุดจำกัดอยู่ที่ 300 มิลลิแอมป์ ความเร็วรอบสัญญาณนาฬิกาของระบบอยู่ระหว่างความถี่ของไฟฟ้ากระแสตรงจนถึง 80 เมกะเฮิร์ตและความเร็วรอบสัญญาณนาฬิกาภายนอกอยู่ระหว่างความถี่ของไฟฟ้ากระแสตรงจนถึง 80 เมกะเฮิร์ต โดยใช้ร่วมกับสัญญาณนาฬิกา PLL (Phase-Locked Loop) Oscillator แบบตัว

ต้านทาน-ตัวเก็บประจุภายในมีความเร็ว 12 เมกะเฮิร์ตถึง 20 กิโลเฮิร์ต โดยประมาณซึ่งอาจอยู่ในช่วง 8 ถึง 20 เมกะเฮิร์ตและ 13 ถึง 33 กิโลเฮิร์ตตามลำดับ มีขาอินพุทและเอาท์พุทอยู่ 32 แบบสัญญาณ CMOS โดยมีค่าเธรชโฮลที่ครึ่งหนึ่งของความต่างศักดิ์ VDD ที่ใช้จ่ายไฟเพื่อให้ระบบทำงาน กระแสไฟฟ้าที่ใช้สำหรับแต่ละอินพุทและเอาท์พุทคือ 40 มิลลิแอมป์ อัตราการใช้ไฟฟ้าที่ 3.3 โวลต์ ณ อุณหภูมิ 70 องศาฟาเรนไฮต์ ต่อคำสั่งคือ 500 ไมโครแอมป์

พร็อพเพลเลอร์จะเริ่มต้นการทำงานด้วยการนำข้อมูลจากหน่วยความจำหลักซึ่งบรรจุตัวโปรแกรมไว้โหลดไปยังหน่วยความจำในหน่วยประมวลผลแรก จากนั้นหน่วยประมวลผลแรกนี้ก็จะเริ่มทำงานเพียงหน่วยเดียวก่อนสำหรับหน่วยประมวลผลหน่วยอื่น ๆ จะทำงานภายหลังที่ถูกกำหนดโดยโปรแกรมจากผู้เขียนโปรแกรม การใช้หน่วยความจำหลักที่ใช้ร่วมกันระหว่างหน่วยประมวลผลนั้นจะสามารถอ่านหรือเขียนได้เพียงหน่วยประมวลผลเดียว ณ เวลาหนึ่ง ๆ เท่านั้นซึ่งจะใช้เวลาโดยประมาณ คือ 7 ถึง 22 รอบนาฬิกา การเขียนโปรแกรมบนพร็อพเพลเลอร์จะสามารถเขียนได้โดยใช้ภาษาสปิน[2] (Spin) ที่สร้างและพัฒนาโดยบริษัทพาราแล็กซ์เพื่อใช้งานบนชิปพร็อพเพลเลอร์ลักษณะของภาษาจะอยู่ในระดับที่ใกล้เคียงกับภาษาเบสิกหรือผู้เขียนโปรแกรมอาจเขียนโปรแกรมด้วยภาษาแอสเซมบลีได้โดยเขียนร่วมกันกับภาษาสปินก็ได้เช่นกัน ตัวอย่างของโปรแกรมที่เขียนด้วยภาษาสปินแสดงในภาพที่ 2.1

```
CON
   N = 32

VAR
   long a[N]

PUB main | i, j, temp
   repeat i from 1 to N
      repeat j from i to N-1
         if a[j] > a[j+1]
            temp := a[j]
            a[j] := a[j+1]
            a[j+1] := a[j]
```

ภาพที่ 2.1 แสดงตัวอย่างโปรแกรมภาษาสปิน

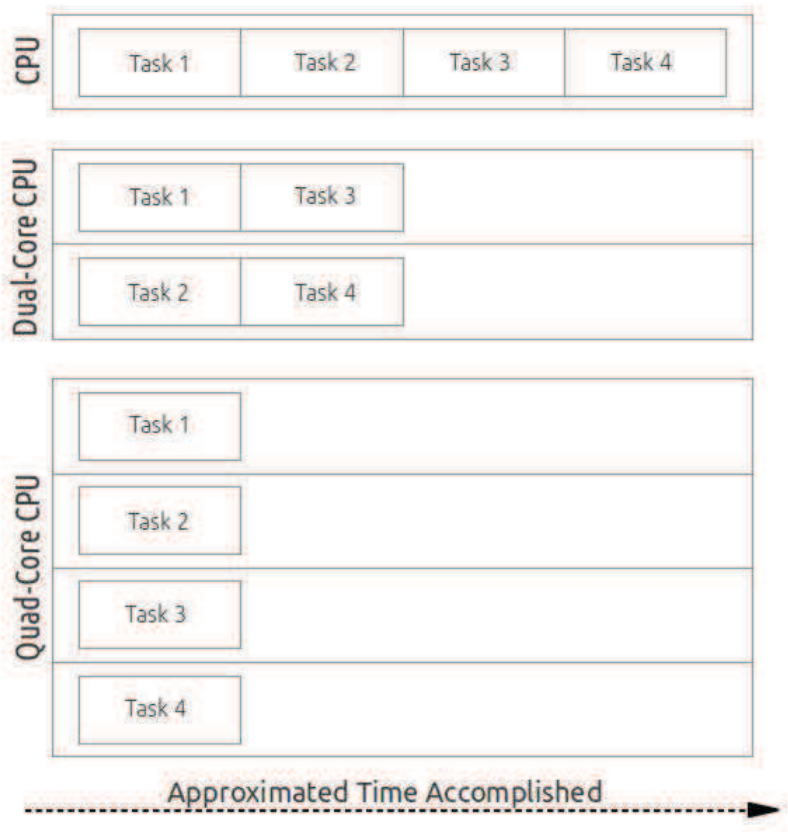
2.2 การทำงานแบบขนาน

โดยทั่วไปโปรแกรมโดยเฉพาะบนหน่วยประมวลผลเดี่ยวจะทำงานแบบอนุกรมทีละคำสั่ง จากคำสั่งที่อยู่ลำดับแรกไปยังลำดับสุดท้ายโดยอาจมีการวนลูปหรือกระโดดไปยังตำแหน่งข้างหน้าหรือย้อนหลังอยู่บ้าง ตามแต่อัลกอริทึมที่ผู้เขียนโปรแกรมนั้นสร้างขึ้นสำหรับในกรณีที่มีหลายหน่วยประมวลผลที่สามารถทำงานได้พร้อมกัน ก็จะสามารถใช้หน่วยประมวลผลเหล่านี้ให้ได้ประโยชน์สูงมากขึ้นได้โดยการออกแบบอัลกอริทึมให้แบ่งและกระจายการทำงานให้แต่ละหน่วยประมวลผลช่วยกันทำงาน[3] ซึ่งจะทำให้ลดเวลารวมที่ใช้ในการทำงานลงไปได้

การทำงานของคอมพิวเตอร์จะทำงานตามคำสั่งที่อยู่ในหน่วยความจำของเครื่องโดยในแต่ละคำสั่งจะถูกส่งไปยังหน่วยประมวลผลกลางเพื่อทำการประมวลผล เครื่องที่มีหน่วยประมวลผลเดี่ยวก็จะมีการทำงานในแบบอนุกรมซึ่งก็คือคำสั่งที่อยู่ในหน่วยความจำก็จะถูกประมวลผลทีละคำสั่งเรียงลำดับไปตามปกติซึ่งอาจมีการกระโดดข้ามหรือกระโดดวนลูปทำคำสั่งเฉพาะในบางลักษณะแต่ก็เป็นการทำงานทีละคำสั่งแบบอนุกรมเช่นเดิม สำหรับในกรณีเครื่องที่มีหลายหน่วยประมวลผลสามารถทำงานได้พร้อม ๆ กัน จะสามารถทำงานได้พร้อม ๆ กันในลักษณะทำงานแบบขนานกันไป ภาพที่ 2.2 แสดงตัวอย่างโปรแกรมที่ทำการหาค่าของสมการ f(x) = x*x - 2*x + 1 โดยหาค่าของสมการสำหรับทุกค่า x ที่อยู่ในช่วงตั้งแต่ 1 ถึง 128 ในโปรแกรมนี้จะเห็นว่าจะเกิดการวนรอบทำงานเป็นจำนวน 128 ครั้ง เพื่อคำนวณหาค่าให้กับตัวแปรอาร์เรย์ทั้ง 128 ตัว โดยการทำงานทั้งหมดที่ทำงานบนหน่วยประมวลผลเดี่ยวก็จะเกิดขึ้นบนหน่วยประมวลผลนั้น ๆ ทั้งหมด แต่ถ้าหากทำงานอยู่บนหลายหน่วยประมวลผลก็จะสามารถกระจายงานแบ่งออกเป็นส่วน ๆ ให้กับแต่ละหน่วยประมวลผลทำการคำนวณในส่วนที่ได้รับกระจายงานมาได้โดยถ้าหากมีหน่วยประมวลผลคู่ก็จะแบ่งให้แต่ละหน่วยประมวลผลย่อยทำการคำนวณหน่วยละ 64 รอบ ถ้าหากทำงานบนหน่วยประมวลผลแบบสี่หน่วยก็จะเกิดการคำนวณในแต่ละหน่วยประมวลผลย่อยเป็นจะนวน 32 รอบ ซึ่งเมื่อรวมการทำงานของทุก ๆ หน่วยประมวลผลแล้วก็จะเป็น 128 ที่จะเท่ากันกับจำนวนรอบในการคำนวณแบบปกติบนหน่วยประมวลผลเดี่ยว ดังภาพที่ 2.3

```
double f[128];
for (x=1; x<=128; ++x)
   f[x] = x*x - 2*x + 1;
```

ภาพที่ 2.2 แสดงโปรแกรมการหาค่าของสมการ



ภาพที่ 2.3 แสดงการกระจายการทำงานในหน่วยประมวลผลหลายหน่วย

2.3 การแปลภาษา

ภาษาที่เรานำมาใช้ในการเขียนโปรแกรมได้นั้นเราจะต้องสร้างคำสำคัญที่จะนำมาใช้สั่งงานและทำงานในระดับภาษาเครื่อง เช่น ภาษาแอสเซมบลี ก็ยังต้องใช้คำสั่งถึงแม้ว่าคำสั่งจะเป็นคำสั่งที่แปลเป็นคำเพื่อให้ใช้งานได้ง่ายเป็นหลัก ดังนั้นในการใช้คำสั่งจึงเป็นสิ่งที่สำคัญและจะมีความสำคัญมากยิ่งขึ้นในระดับที่สูงขึ้นเนื่องจากคำสั่งจะช่วยให้การเขียนโปรแกรมที่มีความยากและซับซ้อนได้ง่ายขึ้นเนื่องจากคำสั่งเหล่านี้จะช่วยแปลงเป็นภาษาเครื่องอย่างอัตโนมัติและโครงสร้างของคำสั่งจะสามารถอ่านและทำความเข้าใจได้ง่ายรวดเร็วและซับซ้อนมากยิ่งขึ้น

โปรแกรมตัวอย่างในภาพที่ 2.4 เป็นโปรแกรมที่อยู่ในรูปของภาษาระดับสูงโดยโปรแกรมนี้จะทำการวนทำงานเพื่อเปรียบเทียบค่าของตัวแปรในอาเรย์เพื่อเรียงค่าจากน้อยไปมากโดยใช้ลูปสองชั้น ชั้นแรกจะวนตั้งแต่รอบที่ 1 ถึงรอบที่ N-1 ส่วนลูปชั้นในจะวนตั้งแต่รอบจากลูปชั้นแรกจนถึงรอบที่ N-1 โดยในแต่ละรอบจะทำการเปรียบเทียบค่าในอาร์เรย์ โดยถ้าหากคำที่อยู่ในอันดับที่น้อยกว่ามีค่ามากกว่าก็จะเกิดการสลับค่าของอันดับที่เปรียบเทียบกันนั้นแต่ถ้าหากไม่ใช่ก็จะไม่เกิดการสลับใด ๆ สำหรับในภาพที่ 2.5 เป็นโปรแกรมที่อยู่ในรูปของภาษาในระดับภาษาเครื่องภาษาแอสเซมบลีของไมโครคอนโทรลเลอร์พร็อพเพลเลอร์โดยการทำงานจะทำงานในลักษณะเดียวกันกับโปรแกรมในภาพที่ 2.4 ซึ่งหากเปรียบเทียบกันแล้วก็จะพบว่าจะมีความยากง่ายในการอ่านที่แตกต่างกันค่อนข้างชัดเจนโดยตัวอย่างแรกจะมีความใกล้เคียงกับภาษาที่ใช้อ่านเขียนมากกว่าแต่ถ้าหากโปรแกรมอยู่ในรูปแบบเลขฐานสิบหกหรือเลขฐานสองก็จะอ่านได้อยากกว่ามากดังแสดงในภาพที่ 2.6

```
for (i=1; i < N; ++i)
   for (j=i; j < N-1; ++j)
      if (data[j] > data[j+1]) {
         temp = data[j];
         data[j] = data[j+1];
         data[j+1] = temp;
      }
```

ภาพที่ 2.4 โปรแกรมการเรียงแบบฟองภาษาซี

```
DAT
:loop_i
        cmp     i, #32          wz
if_nz   jmp     #:end

        mov     ptr_a, i

:loop_j
        cmp     j, #31          wz
if_nz   jmp     #:end_loop_j

        rdlong  ptr_a, var_a
        add     ptr_a, 4
        rdlong  ptr_a, var_b
        cmp     var_a, var_b    wz, wc
if_z    jmp     #:end_if
if_c    jmp     #:end_if
        mov     var_c, var_a
        mov     var_a, var_b
        mov     var_b, var_c

:end_if
        add     ptr_a, 4
        jmp     #:loop_j
```

ภาพที่ 2.5 รูปการเรียงแบบฟองภาษาแอสเซมบลี

```
:end_loop_j
        add     i, 4
        jmp     #:loop_i


:end
        nop


i       res
j       res
ptr_a   res
var_a   res
var_b   res
var_c   res
```

ภาพที่ 2.5 **(ต่อ)** รูปการเรียงแบบฟองภาษาแอสเซมบลี

```
0AF3:0100  00 1B B7 00 00 CC 10 00-4C 00 D4 00 18 00 E4 00
0AF3:0110  3C 00 02 00 08 00 0C 00-36 65 64 69 68 D8 00 68
0AF3:0120  36 EC D8 00 FA 0A 14 68-D8 00 6D 68 36 EC D8 00
0AF3:0130  68 D9 00 68 D8 00 68 36-EC D9 00 64 37 04 36 ED
0AF3:0140  6A 02 59 36 37 04 66 02-51 32 00 00 00 00 00 00
0AF3:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0AF3:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0AF3:0170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0AF3:0180  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0AF3:0190  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0AF3:01A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0AF3:01B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0AF3:01C0  00 00 00 00 00 00 00 00-00 00 00 00 FF FF F9 FF
0AF3:01D0  FF FF F9 FF 00 00 00 00-00 00 00 00 00 00 00 00
```

ภาพที่ 2.6 รูปการเรียงแบบฟองแบบเลขฐาน

โทเคนเป็นค่าที่ใช้ภายในตัวแปลโปรแกรมเพื่อใช้แทนคำสั่งที่อ่านเข้ามาใช้ในการพาร์สข้อมูลและในช่วงการสร้างรหัสภายหลังอีกด้วย ในการสร้างโทเคนจะต้องสร้างชื่อโทเคนและระบุหมายเลขให้โดยที่หมายเลขจะต้องไม่ซ้ำกันในเฮดเดอร์ไฟล์และเพิ่มโค้ดให้กับส่วนสแกนเนอร์เพื่อให้ตัวแปลโปรแกรมรู้จัก

โครงสร้างภาษาของภาษาที่จัดอยู่ในประเภทอินเทอร์พรีเตอร์อย่างเช่นภาษาเบสิกจะมีโครงสร้างที่ไม่ซับซ้อนมากนักในแต่ละบรรทัดของคำสั่งอาจมีคำสั่งได้มากกว่าหนึ่งรูปแบบของคำสั่งจะประกอบไปด้วยคำสำคัญคำสั่งแล้วตามด้วยพารามิเตอร์สำหรับคำสั่งนั้น ๆ แต่ละคำสั่งจะแยกกันด้วยเครื่องหมายคั่นคำสั่งหรือการขึ้นบรรทัดใหม่ก็ได้เช่นกัน โครงสร้างดังที่กล่าวนี้เป็น

โครงสร้างที่มีความซับซ้อนไม่มากเพียงแค่ใช้การแบ่งคำสำคัญและกำหนดเครื่องหมายก็สามารถสร้างตัวแปลโปรแกรมได้ไม่ยากนัก สำหรับภาษาในระดับสูง เช่น ภาษาซี ภาษาปาสคาล จะมีโครงสร้างภาษาที่มีความซับซ้อนมากกว่าดังจะเห็นได้จากการมีฟังก์ชันที่สามารถมีพารามิเตอร์ได้ตามต้องการและมีส่วนภายในที่บรรจุคำสั่งการทำงานและมีการคืนค่ากลับได้เมื่อฟังก์ชันทำงานเสร็จสิ้นแล้ว มีโครงสร้างเช่นฟอร์ลูปที่สามารถวนทำงานได้ตามจำนวนที่กำหนดภายในวงเล็บที่ต่อจากคำสำคัญฟอร์โดยภายในวงเล็บจะระบุการให้ค่าเริ่มต้นกับตัวแปรที่เกี่ยวข้อง ตามด้วยเงื่อนไขที่ใช้ในการตัดสินใจการทำคำสั่งสเตทเมนท์หรือการคำสั่งที่อยู่ภายในบล็อคและการทำงานภายหลังการเสร็จสิ้นการทำงานในแต่ละรอบเช่นการอินครีเมนต์หรือเพิ่มค่าตัวแปรที่ใช้เป็นอินเด็กซ์หรือใช้ในเงื่อนไขตัดสินใจดังกล่าวข้างต้น การมีตัวแปรโครงสร้างที่สามารถกำหนดตัวแปรและชนิดของตัวแปรแต่ละตัวได้ตามแต่การใช้งานสามารถสร้างได้โดยประกาศคีย์เวิร์ดสตรัคและชื่อของสตรัคที่จะใช้เป็นไทป์แล้วตามด้วยบล็อคที่ภายในเป็นการประกาศชนิดตัวแปรและชื่อของตัวแปรได้ตามจำนวนที่ต้องการแล้วยังสามารถประกาศชื่อตัวแปรที่จะสามารถนำไปใช้งานภายหลังจากบล็อคได้อีกด้วย จะเห็นได้ว่าความซับซ้อนของภาษาในระดับที่สูงกว่าก็จะมีความซับซ้อนที่สูงกว่าเช่นกัน

คำชี้แนะตัวแปลโปรแกรม (compiler directive) เป็นคำสั่งชนิดหนึ่งในการเขียนโปรแกรมซึ่งแตกต่างจากคำสั่งแบบอื่นที่จะเป็นคำสั่งที่สั่งงานทั่วไป โดยคำชี้แนะตัวแปลโปรแกรมจะเป็นคำสั่งที่จะถูกใช้เป็นข้อมูลที่ตัวแปลโปรแกรมจะนำไปใช้ควบคู่กับรหัสคำสั่งโดยเฉพาะกับส่วนที่อยู่ถัดมาเป็นส่วนใหญ่ ดังตัวอย่างในภาพที่ 2.7 การใช้คำชี้แนะตัวแปลโปรแกรม parallel for เพื่อให้ข้อมูลแก่ตัวแปลโปรแกรมว่าในโค้ดสเตทเมนท์ต่อไปนี้ต้องการให้ทำงานในแบบขนานกับคำสั่ง for ซึ่งตัวแปลโปรแกรมก็จะเกิดการทำงานภายในแบบพิเศษเพื่อให้เกิดการทำงานแบบขนาน โทเคนพาราเลลจะถูกสร้างขึ้นเพื่อใช้ในขั้นตอนการสร้างพาร์สทรีสำหรับคำสั่งที่ต้องการให้เกิดการทำงานแบบขนานนี้ โดยโทเคนพาราเลลจะถูกวางบนสแตกก่อนแล้วจึงตามด้วยคำสั่งฟอร์ลูปที่ประกอบไปด้วยเอ็กเพรสชันสำหรับการวนลูปและต่อท้ายด้วยสเตทเมนท์หรือบล็อคคำสั่ง

```
#pragma parallel for
  for ( i=0; i < 10; ++i )
    a[i] = b[i];
```

ภาพที่ 2.7 แสดงการใช้คำชี้แนะตัวแปลโปรแกรม

2.4 ภาษา RZ

ภาษา RZ[4] เป็นภาษาที่มีลักษณะต่าง ๆ คล้ายกันกับภาษา C/C++ เช่น การประกาศตัวแปร การมีบล็อคของคำสั่งด้วยเครื่องหมายปีกกาเปิดและเครื่องหมายปีกกาปิด เครื่องหมายตัวดำเนินการทางคณิตศาสตร์ต่าง ๆ เป็นต้น โดยเริ่มแรกภาษา RZ นี้มีจุดมุ่งหมายเพื่อเป็นภาษาสำหรับการเรียนการสอน ใช้เพื่อแสดงภาพของระบบการทำงานของเครื่องคอมพิวเตอร์ในการทำการแปลภาษา สร้างรหัสการทำงาน และทำงานบนตัวจำลองสถาปัตยกรรมชุดคำสั่งของเครื่องคอมพิวเตอร์ ผู้เรียนสามารถเรียนรู้การทำงานต่าง ๆ ได้อีกทั้งยังเปลี่ยนแปลงแก้ไขเพื่อศึกษาการทำงานได้อีกด้วย[5]

ภาษา RZ มีคำสำคัญทั้งหมด 5 คำด้วยกันคือ if, else, while, return และ print โดยคำสั่ง if และ else ใช้สำหรับตรวจสอบเงื่อนไขสำหรับการทำงาน คำสั่ง while ใช้สำหรับการทำลูป คำสั่ง return ใช้สำหรับคืนค่าและจบการทำงานของฟังก์ชัน และคำสั่ง print ใช้สำหรับแสดงผลลัพธ์ไปยังเอาท์พุท สำหรับตัวดำเนินการของ ภาษา RZ มีตัวดำเนินการทั้งหมด 15 ตัว ดังนี้ เครื่องหมายบวก (+), เครื่องหมายลบ (-), เครื่องหมายคูณ (*), เครื่องหมายหาร (/), เครื่องหมายเท่ากับ (==), เครื่องหมายไม่เท่ากับ (!=), เครื่องหมายน้อยกว่า (<), เครื่องหมายน้อยกว่าเท่ากับ (<=), เครื่องหมายมากกว่า (>), เครื่องหมายมากกว่าเท่ากับ (>=), เครื่องหมายน็อต (!), เครื่องหมายตรรกะและ (&&), เครื่องหมายตรรกะหรือ (||), เครื่องหมายดีเรฟเฟอร์เรนซ์ (*) และเครื่องหมายแอ็ดเดรส (&)

ภาษา RZ เป็นภาษาที่ไม่มีชนิดของตัวแปรเนื่องจากมีชนิดของตัวแปรเพียงชนิดเดียวซึ่งก็คือชนิดจำนวนเต็มหรือ integer ทุกตัวแปรที่สร้างขึ้นจะมีชนิดเป็นจำนวนเต็มเท่านั้น ตัวแปรของภาษา RZ นี้แบ่งออกได้เป็น 2 แบบ คือ ตัวแปรแบบโกลบอล และตัวแปรแบบโลคอล ตัวแปรแบบโกลบอลจะมีลักษณะต่าง ๆ คือ ตัวแปรแบบโกลบอลจะต้องถูกประกาศก่อนการใช้งานเสมอซึ่งต่างจากตัวแปรแบบโลคอล ตัวแปรแบบโกลบอลสามารถเป็นตัวแปรแบบสเกล่าหรือเวคเตอร์ก็ได้ โดยตัวแปรแบบสเกลาหมายถึงตัวแปรแบบทั่วไปและตัวแปรแบบเวคเตอร์หมายถึงตัวแปรแบบอาร์เรย์หลายมิติ แต่สำหรับตัวแปรแบบเวคเตอร์ของภาษา RZ สามารถมีได้เพียงมิติเดียวและจะต้องระบุขนาดของตัวแปรไว้เมื่อประกาศตัวแปรด้วย สำหรับตัวแปรแบบโลคอลนั้น สามารถเป็นตัวแปรแบบสเกล่าได้เท่านั้นไม่สามารถเป็นแบบเวคเตอร์ได้ ผู้เขียนโปรแกรมไม่จำเป็นต้องประกาศตัวแปรแบบภายใน จะทำการประกาศตัวแปรหรือไม่ก็ได้ ถ้าหากผู้เขียนโปรแกรมไม่ได้ทำการประกาศตัวแปรไว้สำหรับตัวแปรแบบภายใน ตัวแปลรหัสก็จะสร้างตัวแปรให้เองอย่างอัตโนมัติ

ตัวอย่างโปรแกรมการเรียงแบบฟองแสดงในภาพที่ 2.8 เป็นโปรแกรมที่จะเรียงค่าข้อมูลทั้งหมด 32 จำนวนจากมากไปหาน้อย โดยเริ่มจากการกำหนดค่าเริ่มต้นให้กับตัวแปร i เพื่อเริ่มการทำงานจากข้อมูลลำดับแรกที่เริ่มที่ตำแหน่ง 0 ลูป while ตรวจสอบเงื่อนไข i<32 เพื่อทำการวนทำงานทั้งหมด 32 รอบโดยเริ่มตั้งแต่รอบที่ 0 จนถึงรอบที่ 31 ภายในลูปจะเริ่มต้นด้วยการกำหนดค่าให้กับตัวแปร j ที่จะใช้ในการวนเปรียบเทียบแล้วจึงเริ่มลูป while ชั้นในซึ่งจะตรวจสอบเงื่อนไข j < 32-1 เพื่อจำกัดการทำงานในแต่ละรอบให้อยู่ในช่วงที่กำหนด และที่ต้องเป็น 32-1 นั้นเพราะเนื่องจากการตรวจสอบค่าจะทำการบวกค่าของตัวแปร j ไปอีกหนึ่งสำหรับข้อมูลที่อยู่ถัดไปอีกหนึ่งตำแหน่งซึ่งถ้าหากค่า j เป็นค่าของตำแหน่งสุดท้ายจะทำให้ค่าของ j+1 เกินกว่าข้อมูลที่มีอยู่ซึ่งอาจทำให้ทำงานผิดพลาดได้ ภายในลูปจะมีคำส่ง if เพื่อเปรียบเทียบข้อมูลตัวที่ j และ j+1 ถ้าหากข้อมูลในตำแหน่งที่น้อยกว่า คือ ตำแหน่งที่ j มีค่ามากกว่าข้อมูลในตำแหน่งที่ j+1 ก็จะทำคำสั่งถัดไปที่อยู่ภายในคำสั่ง if คือ เก็บค่าของข้อมูลในตำแหน่ง j ไว้ในตัวแปร temp แล้ว

กำหนดค่าของข้อมูลในตำแหน่งที่ j ให้เป็นค่าในตำแหน่งที่ j+1 แล้วจึงกำหนดค่าให้กับข้อมูลในตำแหน่งที่ j+1 เป็นค่า temp ที่เป็นค่าข้อมูลในตำแหน่งที่ j ที่ได้เก็บเอาไว้ แล้วจึงเพิ่มค่าตัวแปร j ขึ้นหนึ่งและเมื่อลูป while ชั้นในเสร็จสิ้นการทำงานก็จะเพิ่มค่าของตัวแปร i ขึ้นอีกหนึ่งเช่นกัน

```
main() {
   i = 0;
   while (i<32) {
      j = i;
      while (j < 32-1) {
         if (a[j] > a[j+1]) {
            temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp;
         }
         j = j+1;
      }
      i = i+1;
   }
}
```

ภาพที่ 2.8 แสดงโปรแกรมการเรียงแบบฟอง

การทำงานของตัวแปลโปรแกรมดังภาพที่ 2.9 และ 2.10 จะได้ผลลัพธ์ออกมาเป็นโครงสร้างของโครงสร้างข้อมูลภายในที่อยู่ในรูปของลิสต์และโค้ดที่อยู่ในรูปแบบภาษาที่ต้องการ เช่น ภาษาสปินดังในภาพ

```
 main

 (fun main (do (= #1 0 )(while (< #1 32 )(do (= #2 #1 )(while (< #2 (-
32 1 ))(do (if (> (vec #3 #2 )(vec #3 (+ #2 1 )))(do (= #4 (vec #3 #2
))(= (vec #3 #2 )(vec #3 (+ #2 1 )))(= (vec #3 (+ #2 1 ))#4 )))(= #2 (+
#2 1 ))))(= #1 (+ #1 1 ))))))


 pub main | i, j, a, temp

    i := 0

    repeat until not (i < 32)

       j := i

       repeat until not (j < (32 - 1))

          if (a[j] > a[(j + 1)])

             temp := a[j]

             a[j] := a[(j + 1)]

             a[(j + 1)] := temp

          j := (j + 1)

       i := (i + 1)
```

ภาพที่ 2. 9 แสดงผลการทำงานของตัวแปลโปรแกรม

```
(fun main
 (do
  (= #1 0 )
  (while
   (< #1 32 )
   (do
    (= #2 #1 )
    (while
     (< #2 (- 32 1 ))
     (do
      (if (> (vec #3 #2 )(vec #3 (+ #2 1 )))
       (do
        (= #4 (vec #3 #2 ))
        (= (vec #3 #2 )(vec #3 (+ #2 1 )))
        (= (vec #3 (+ #2 1 ))#4 )
       )
      )
      (= #2 (+ #2 1 ))
     )
    )
    (= #1 (+ #1 1 ))
   )
  )
 )
)
```

ภาพที่ 2.10 แสดงโครงสร้างข้อมูลแบบต้นไม้ที่ได้จากการทำงานของตัวแปลโปรแกรม

2.5 งานวิจัยที่เกี่ยวข้อง

2.5.1 แมพรีดิวซ์

แมพรีดิวซ์ (MapReduce) เป็นอัลกอริทึมที่ใช้ในการทำงานแบบขนานโดยผู้เขียนโปรแกรมสามารถสั่งทำงานในลักษณะการแมพและการรีดิวซ์ที่สามารถกระจายการทำงานไปยังเครื่องต่าง ๆ ในระบบที่มีอยู่เป็นจำนวนมากให้สามารถช่วยกันทำงานได้อย่างพร้อมกันทำให้ใช้ลดเวลาในการทำงานลงได้เป็นอย่างมาก ลักษณะของโปรแกรมดังตัวอย่างในภาพที่ 2.11 เป็นโปรแกรมที่ตัวอย่างแสดงการนับจำนวนคำทั้งหมดของเอกสาร จะเป็นการสร้างคู่ข้อมูลในฟังก์ชันแมพ และใช้ข้อมูลที่ได้จากการแมพในฟังก์ชันรีดิวซ์ ตัวอย่างการแมพจะรับพารามิเตอร์สองตัวคือตัวแปร key และ value ที่เป็นสตริงโดย key ใช้สำหรับชื่อของเอกสาร (document) และ value ใช้สำหรับเนื้อหาในเอกสารนั่นเอง คำสั่ง for จะทำการวนค่าในแต่ละค่าทีและ word ของตัวแปร value โดยให้ตัวแปร w เป็นตัวแปรที่ใช้แทนค่าในแต่ละรอบแล้วทำการสร้างคู่ข้อมูลใหม่ด้วยคำสั่ง EmitIntermediate คือคู่ข้อมูลของแต่ละ w และค่า 1 นั่นเอง สำหรับฟังก์ชันรีดิวส์ที่มีสองพารามิเตอร์คือตัวแปร key ที่มีไทป์เป็นสตริงและ values ที่มีไทป์เป็น Iterator โดยตัวแปร key ใช้แทน word และตัวแปร values ใช้แทนลิสต์ของจำนวนนับ ตัวแปรชนิด integer ชื่อ result ถูกกำหนดค่าเริ่มต้นเป็น 0 ถูกสร้างขึ้นเพื่อใช้ในการนับคำสั่ง for ใช้วนแต่ละค่าภายในตัวแปร values โดยมีตัวแปร v เป็นตัวแทนค่าในแต่ละรอบที่จะทำการบวกค่าเพิ่มเป็นจำนวน integer ของตัวแปร v ให้กับตัวแปร result และทำการส่งค่า result ที่บวกค่าในแต่ละรอบเรียบร้อยแล้วในรูปแบบสตริงเป็นผลลัพธ์

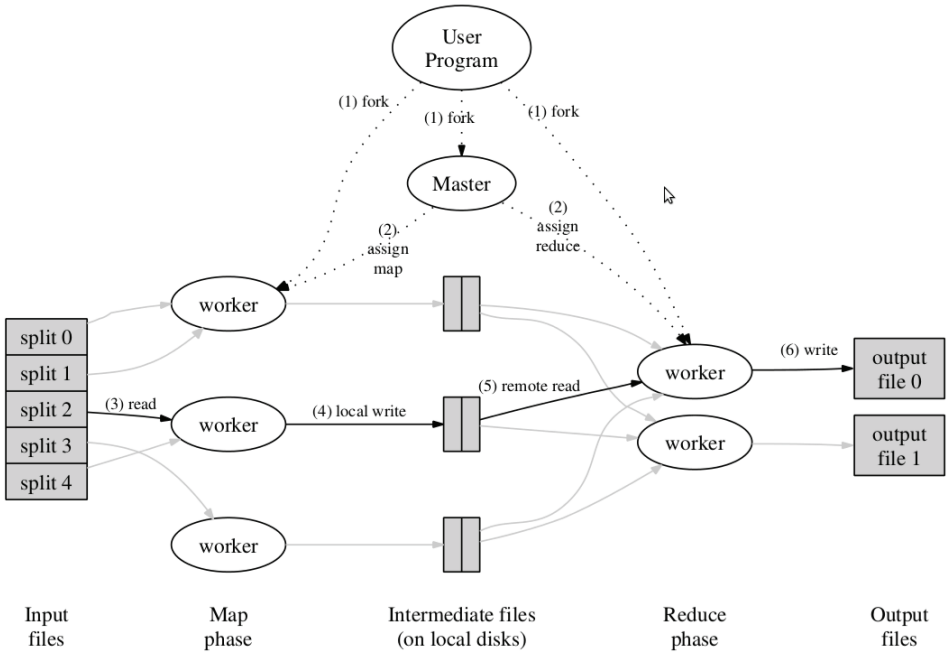
```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

ภาพที่ 2.11 แสดงตัวอย่างโปรแกรมแมพรีดิวซ์

ตัวอย่างโปรแกรมที่สามารถใช้แมพรีดิวซ์ได้ คือ คำสั่งเกร็พแบบกระจาย (Distributed Grep) เป็นโปรแกรมที่จะแมพหารูปแบบที่ผู้ใช้ต้องการและโปรแกรมรีดิวซ์ก็เพียงแค่ส่งข้อมูลที่แมพได้ออกเป็นเอาท์พุทเท่านั้น โปรแกรมการนับความถี่ของการเรียก URL (Count of URL Access Frequency) จะสร้างคู่ข้อมูลของเว็บ URL และค่า 1 ซึ่งหมายถึงการเรียกเว็บเพจในแต่ละครั้งในฟังก์ชันแมพ และจะรวมค่าจำนวนครั้งของการเรียกเว็บเพจเป็นผลลัพธ์ที่ได้จากการทำงาน โปรแกรมกราฟของการเชื่อมโยงของเว็บแบบกลับ (Reverse Web-Link Graph) จะเกิดการสร้างคู่ข้อมูลระหว่างเป้าหมายและต้นทางของการเชื่อมโยงบนเว็บในการแมพ สำหรับการรีดิวซ์จะเป็นการนำข้อมูลจากการแมพมาสร้างเป็นคู่ข้อมูลของเพจเป้าหมายและลิสต์ของแหล่งต้นทางทั้งหมดนั่นเอง โปรแกรมเทอมเวคเตอร์ของแต่ละโฮสต์ (Term-Vector per Host) เป็นโปรแกรมที่จะเก็บคำสำคัญและความถี่ของเอกสารซึ่งฟังก์ชันแมพจะทำการสร้างคู่ข้อมูลระหว่างชื่อโฮสต์ที่ได้และเทอมเวคเตอร์ และฟังก์ชันรีดิวซ์จะสร้างคู่ข้อมูลระหว่างชื่อโฮสต์และเทอมเวคเตอร์ที่ได้ทำการ

รวบรวมและนำคำสำคัญที่มีความสำคัญน้อยออกไปแล้ว โปรแกรมอินเด็กซ์แบบกลับ (Inverted Index) จะทำการสร้างคู่ข้อมูลระหว่างคำสำคัญและหมายเลขของเอกสารในการแมพ และจะทำการรวบรวมหมายเลขเอกสารของแต่ละคำสำคัญเป็นคู่ข้อมูลคือคู่ข้อมูลระหว่างคำสำคัญและลิสต์ของหมายเลขเอกสารซึ่งจะทำให้สามารถตามคำสำคัญที่อยู่ในเอกสารในที่ต่าง ๆ ได้ โปรแกรมการเรียงแบบกระจาย (Distributed Sort) จะทำการแมพระหว่างคีย์ (Key) และเรคอร์ด (Record) และในส่วนของรีดิวซ์ก็จะเป็นการส่งผ่านข้อมูลที่ได้จากการแมพไปเป็นผลลัพธ์โดยตรง ซึ่งจะเป็นผลที่เกิดการเรียงลำดับแล้วเนื่องจากการแมพนั้นจะเกิดการสร้างคีย์เป็นไปตามลำดับที่เป็นลำดับที่เรียงอยู่แล้วไปในตัวเอง

การทำงานของแมพรีดิวซ์ดังภาพที่ 2.12 จะเริ่มต้นจากการสร้างโพรเสซ (process) ย่อยเป็นจำนวนมากเพื่อให้เป็นตัวทำงาน (worker) ที่จะทำงานเป็นทั้งการแมพและการรีดิวซ์โดยทำงานอยู่บนระบบแบบกระจายนั่นเอง ในการสร้างโพรเสซย่อยจะมีโพรเสซย่อยพิเศษอยู่โพรเสซหนึ่งเรียกว่าโพรเสซตัวหลัก (master) ซึ่งจะทำหน้าที่ในการกำหนดการทำงานให้กับตัวทำงานอื่น ๆ ให้ทำการแมพหรือการรีดิวซ์ โดยตัวทำงานที่เป็นการแมพจะทำการนำข้อมูลที่ถูกแบ่งให้เป็นส่วน ๆ ในส่วนที่จะใช้สำหรับทำงานมาทำการสร้างคู่ข้อมูลจากการกำหนดโดยผู้เขียนโปรแกรมซึ่งผลลัพธ์ที่ได้ก็จะถูกเขียนลงบนดิสก์แบบภายในที่เรียกว่าอินเทอร์มิเดียทไฟล์ โดยไฟล์เหล่านี้จะถูกอ่านโดยโพรเสซที่อาจอยู่บนหน่วยอื่นที่ไม่จำเป็นต้องอยู่ในหน่วยเดียวกันซึ่งโพรเสซเหล่านี้จะอ่านข้อมูลไปทำการรีดิวซ์ตามที่ผู้เขียนโปรแกรมได้กำหนดไว้แล้วบันทึกผลลัพธ์ที่ได้ไปยังไฟล์ที่ถูกใช้เป็นเอาท์พุท



ภาพที่ 2.12 แสดงรูปการทำงานของแมพรีดิวซ์

2.5.2 โอเพ็นเอ็มพี

โอเพ็นเอ็มพี[6] (OpenMP) เป็นส่วนต่อประสานการเขียนโปรแกรม (Application Programming Interface) ที่สามารถสร้างการทำงานแบบขนานได้ด้วยการใช้คำชี้แนะตัวแปลโปรแกรม (compiler directive) สำหรับโอเพ็นเอ็มพีที่ผู้เขียนโปรแกรมสามารถใช้ให้ทำงานแบบขนานหรือทำงานแบบตามลำดับปกติได้ตามความต้องการโดยโค้ดการทำงานในแบบขนานอาจได้ผลลัพธ์ไม่ตรงกับการทำงานในแบบตามลำดับธรรมดาก็เป็นได้

โปรแกรมที่เขียนด้วยโอเพ็นเอ็มพีจะเริ่มทำงานด้วยเทรดแบบเดี่ยวเรียกว่าเทรดเริ่มต้นที่เป็นส่วนที่ถูกสร้างขึ้นมาเองโดยโอเพ็นเอ็มพีครอบโปรแกรมทั้งหมด และเมื่อพบคำสั่ง parallel ก็จะเกิดการสร้างเทรดขึ้นมาในทีม โปรแกรมในเทรดจะเป็นโปรแกรมที่ถูกระบุไว้ภายใต้คำสั่ง parallel ที่ผู้เขียนโปรแกรมได้เขียนขึ้นมาและทำงานตามที่ผู้เขียนกำหนดไว้แบบขนานโดยส่วนโปรแกรมหลักจะถูกหยุดการทำงานเอาไว้ให้เทรดทำงานเสียก่อน ในกรณีที่มีหลายเทรดที่ทำงาน

พร้อมกันแต่เสร็จสิ้นการทำงานไม่พร้อมกันในกรณีทั่วไปก่อนการกลับไปทำงานยังโปรแกรมเดิมจะเกิดการรอให้ทุก ๆ เทรดทำงานจะเสร็จเสียก่อนแล้วจึงกลับเว้นแต่ว่าผู้เขียนโปรแกรมได้กำหนดไว้ เทรดนั้นสามารถที่จะสร้างในลักษณะที่ซ้อนกันได้ขึ้นอยู่กับผู้เขียนโปรแกรม โดยการทำงานของเทรดนั้นจะร่วมกันทำงาน (cooperative) คือจะทำงานอย่างเป็นระบบมีการจัดการการทำงานซึ่งจะไม่เกิดการทำงานของทุกเทรดที่สร้างขึ้นมาพร้อม ๆ กันทั้งหมดซึ่งมีข้อเสียหลายประการ ผู้เขียนโปรแกรมสามารถสร้างเทรดขึ้นมาหลายเทรดและซิงโครไนซ์ (synchronize) กันระหว่างเทรดได้โดยใช้โปรแกรมในไลบรารีและตัวแปรสิ่งแวดล้อมของโอเพนเอ็มพีได้ ส่วนในการเขียนข้อมูลลงในไฟล์เดียวกันนั้นโอเพ็นเอ็มพีไม่สามารถรับรองได้ว่าจะเกิดการเขียนที่ถูกต้องซึ่งจะต้องเป็นหน้าที่ของผู้เขียนโปรแกรมเองแต่สำหรับในกรณีที่เกิดการเขียนไฟล์ที่แตกต่างกันของแต่ละเทรดก็จะไม่เกิดปัญหาดังที่กล่าวมาข้างต้น

รูปแบบของคำสั่งของโอเพ็นเอ็มพีดังภาพที่ 2.13 จะเริ่มด้วย #pragma omp แล้วตามด้วยไดเรคทีฟที่ผู้เขียนโปรแกรมต้องการ เช่น parallel สำหรับสร้างเทรด for สำหรับสร้างคำสั่งลูปที่ทำงานในแบบขนาน sections สำหรับการสร้างเทรดในแบบที่ไม่เกิดการติดต่อกัน เป็นต้น และตามด้วย clause ที่อาจมีเพียงหนึ่งหรือมากกว่าหรือจะไม่มีก็ได้ขึ้นอยู่กับผู้เขียนโปรแกรมจะกำหนด โดย clause จะเป็นการกำหนดการทำงานที่จะแตกต่างกันไปในแต่ละคำสั่งไดเรคทีฟที่ประกาศไว้ก่อนหน้าโดยผู้เขียนโปรแกรม ตัวอย่างของ clause เช่น private สำหรับประกาศตัวแปรที่ใช้เฉพาะภายใน reduction สำหรับการรวมค่าตัวแปรจากการกระจายการทำงานในแต่ละเทรด และ nowait สำหรับการบ่งบอกถึงการไม่ต้องเกิดการรอระหว่างที่เทรดอื่นยังทำงานไม่เสร็จเป็นต้น ตัวอย่างโปรแกรมแสดงในภาพที่ 2.14

```
#pragma omp directive-name [clause[ [,] clause]...] new-line
```

ภาพที่ 2.13 แสดงรูปแบบคำสั่งของโอเพ็นเอ็มพี

```
#include <omp.h>

void subdomain(float *x, int istart, int ipoints)
{
  int i;
  for (i = 0; i < ipoints; i++)
    x[istart+i] = 123.456;
}

void sub(float *x, int npoints)
{
  int iam, nt, ipoints, istart;

#pragma omp parallel default(shared) private(iam,nt,ipoints,istart)
  {
    iam = omp_get_thread_num();
    nt = omp_get_num_threads();
    ipoints = npoints / nt;    /* size of partition */
    istart = iam * ipoints; /* starting array index */
    if (iam == nt-1)    /* last thread may do more */
      ipoints = npoints - istart;
    subdomain(x, istart, ipoints);
  }
}
```

ภาพที่ 2.14 ตัวอย่างโปรแกรมที่เขียนด้วยโอเพ็นเอ็มพี

```
int main()
{
  float array[10000];
  sub(array, 10000);
  return 0;
}
```

ภาพที่ 2.14 (ต่อ) ตัวอย่างโปรแกรมที่เขียนด้วยโอเพ็นเอ็มพี

2.5.3 อินเทลทีบีบี

อินเทลทีบีบี[7] หรือ Inter Threading Building Blocks เป็นไลบราลีที่สร้างและพัฒนาโดยบริษัทอินเทลเพื่อใช้สำหรับการเขียนโปรแกรมแบบในแบบขนานบนภาษาซีพลัสพลัสซึ่งอินเทลทีบีบีได้สร้างคลาสเท็มเพลตสำหรับการเขียนโปรแกรมแบบขนานสำหรับผู้เขียนโปรแกรมเพื่อให้การออกแบบและเขียนโปรแกรมเกิดความสะดวกและมีประสิทธิภาพที่สูงยิ่งขึ้น อินเทลทีบีบีได้ถูกสร้างขึ้นจากเทคโนโลยีและการพัฒนาการเขียนโปรแกรมแบบขนาน เช่น ซิลค์[8] (Cilk) โอเพ็นเอ็มพี (OpenMP) และ STAPL เป็นต้น

โปรแกรมแบบขนานที่เขียนด้วยไลบรารี่ (library) ของอินเทลทีบีบี (Intel Threading Building Block) จะเป็นการใช้เทมเพลตของการเขียนโปรแกรมแบบขนานดังตัวอย่างในภาพที่ 2.15 ซึ่งเป็นตัวอย่างการทำงานแบบขนานของฟังก์ชัน Foo โดยคลาส ApplyFoo ประกอบด้วยตัวแปร my_a ที่มีชนิดเป็น float *const และฟังก์ชันสองฟังก์ชันคือ ฟังก์ชัน operator และฟังก์ชัน ApplyFoo ที่เป็นคอนสตรัคเตอร์ซึ่งจะถูกเรียกใช้เฉพาะขณะที่คลาสถูกสร้างโดยจะรับพารามิเตอร์หนึ่งค่าที่มีไทป์เป็นอาเรย์ของ float เพื่อใช้ในการกำหนดให้เป็นค่าเริ่มต้นของตัวแปร my_a สำหรับฟังก์ชัน operator จะรับพารามิเตอร์หนึ่งค่าคือตัวแปร r ที่มีไทป์เป็นคลาส blocked_range โดยโค้ดภายในจะเป็นการสร้างตัวแปร a ที่มีไทป์เป็นพอยน์เตอร์ของ float และชี้

ไปยังตัวแปร my_a เพื่อใช้งานภายในตามด้วยลูป for ที่ใช้ตัวแปร i ไทป์เป็น size_t เป็นตัวนับโดยเริ่มตั้งแต่เมื่อตัวแปร i มีค่าเท่ากับค่าที่ได้จากฟังก์ชัน r.begin() ไปจนถึงเมื่อค่า i มีค่าเท่ากับค่า r.end() โดยตัวแปร r คือตัวแปรของคลาส blocked_range ที่ทำหน้าที่ในการกำหนดช่วงให้ทำงาน ในการเรียกคลาส ApplyFoo ให้ทำงานแบบขนานนั้นจะใช้คำสั่ง parallel_for แล้วตามด้วยพารามิเตอร์สองตัวคือ พารามิเตอร์ที่ใช้เป็น range และคลาสที่ต้องการให้ทำงานแบบขนาน โดยพารามิเตอร์ที่ใช้ในตัวอย่างนี้คือ คลาส blocked_range โดยกำหนดชนิดของขนาดเป็น size_t ตั้งแต่ 0 จนถึง n โดยมีลักษณะการแบ่งเป็นแบบ IdealGrainSize และคลาสที่ต้องการให้ทำงานแบบขนานคือ คลาส ApplyFoo ที่รับพารามิเตอร์ a เพื่อใช้เป็นค่าเริ่มต้นให้กับการทำงานภายในคลาส

```
#include "tbb/blocked_range.h"


class ApplyFoo {
   float *const my_a;
public:
   void operator()( const blocked_range<size_t>& r ) const {
      float *a = my_a;
      for( size_t i=r.begin(); i!=r.end(); ++i )
         Foo(a[i]);
   }
   ApplyFoo( float a[] ) :
      my_a(a)
   {}
};
```

ภาพที่ 2.15 แสดงการเขียนโปรแกรมด้วยอินเทลบีบี

```
 #include "tbb/parallel_for.h"

 void ParallelApplyFoo( float a[], size_t n ) {

    parallel_for(blocked_range<size_t>(0,n,IdealGrainSize),
ApplyFoo(a) );

 }
```

ภาพที่ 2.15 **(ต่อ)** แสดงการเขียนโปรแกรมด้วยอินเทลบีบี

บทที่ 3

วิธีดำเนินการวิจัยและการทดลอง

การทดลองการแปลโปรแกรมจะเป็นการทดลองแปลโปรแกรมซอร์สโค้ดต้นฉบับที่เขียนด้วยภาษา RZ แล้วนำผลที่ได้อัพโหลดไปยังไมโครคอนโทรลเลอร์บนบอร์ดทดลองและสั่งให้ทำงานแล้วจึงตรวจสอบการทำงานและผลการทำงาน โดยซอร์สโค้ดที่นำมาทำการทดสอบคือโปรแกรมการคูณเมทริกซ์ โปรแกรมการบวกแบบรีดักชัน และโปรแกรมการเรียงแบบคี่-คู่ โปรแกรมตัวอย่างนี้เป็นตัวอย่างของโปรแกรมที่ใช้งานบนระบบแบบขนานทั่วไป ในการคูณเมทริกซ์โดยเฉพาะกับเมทริกซ์ที่มีขนาดใหญ่จะต้องใช้ทรัพยากรจำนวนมากในการทำงาน เช่น การคูณเมทริกซ์ขนาด 10x10 จะต้องเกิดเฉพาะการคูณ 1000 ครั้ง การคูณเมทริกซ์ขนาด 100x100 จะเกิดการคูณ 1,000,000 และสำหรับเมทริกซ์ขนาด 1000x1000 จะต้องทำการคูณทั้งสิ้น 1,000,000,000 ครั้ง เป็นต้น โปรแกรมการบวกแบบรีดักชันเป็นการหาผลรวมที่พบได้บ่อยในระบบการทำงานแบบขนานเนื่องจากการหาผลรวมสามารถพบได้ทั่วไปในอัลกอริทึมต่าง ๆ และเป็นการใช้ประโยชน์จากการทำงานแบบขนานได้อย่างมีประสิทธิภาพ การหาผลรวมในลักษณะนี้จะเป็นการจับคู่บวกของทุกค่าที่ต้องการบวกโดยไม่ซ้ำกันเป็นรอบ ๆ จะได้ผลลัพธ์ซึ่งในแต่ละรอบจะได้ผลจากการบวกของทุก ๆ คู่เป็นจำนวนครึ่งหนึ่งของจำนวนที่นำมาบวกทำให้การบวกในรอบต่อ ๆ ไปจะทำการบวกลดลงจนกระทั่งรอบสุดท้ายที่จะเป็นการบวกเพียงคู่สุดท้ายที่ได้จากผลบวกของทุก ๆ ค่าในการบวกรอบก่อน ๆ จึงเป็นผลลัพธ์ของการบวกทั้งหมดทุกค่า สำหรับการเรียงแบบคี่-คู่เป็นอัลกอริทึมการเรียงอีกแบบหนึ่งที่สามารถนำไปใช้งานบนระบบแบบขนานได้โดยง่ายและรวดเร็วในกรณีที่มีจำหน่วยประมวลผลมากพอ ในการเรียงข้อมูลจะเกิดการจับคู่ที่อยู่ใกล้กันของข้อมูลทุกตัวโดยจะแบ่งเป็นรอบที่เริ่มจากค่าแรกหรือรอบคี่และรอบที่เริ่มจากค่าที่สองหรือรอบคู่สลับกันไปเป็นจำนวนทั้งสิ้น N-1 รอบโดย N คือจำนวนของข้อมูลทั้งหมด การเรียงในลักษณะนี้จะมีความคล้ายกับการเรียงแบบการเรียงแบบฟองที่มีการเปรียบเทียบค่าแล้วทำการสลับตำแหน่งเมื่อเงื่อนไขเป็นจริงเช่นค่าที่อยู่ในตำแหน่งที่น้อยกว่ามีค่ามากกว่า แต่การเรียงแบบคี่-คู่จะทำการเปรียบเทียบคู่ข้อมูลทุก ๆ ตัวที่อาจทำงานแบบขนานกันไปพร้อม ๆ กันโดยใช้ความสามารถของ

ระบบแบบขนานที่สามารถทำงานแบบขนานกันได้ซึ่งจะทำให้เกิดการเปรียบเทียบข้อมูลได้จำนวนมากพร้อม ๆ กันซึ่งก็จะทำให้ใช้เวลาในการทำงานที่ลดลงด้วย

3.1 หลักการการสร้างตัวแปลโปรแกรมแบบขนาน

ด้วยสาเหตุที่โปรแกรมนั้นมีความซับซ้อนตั้งแต่ระดับง่ายไปจนถึงยาก เช่น โปรแกรมหาผลรวมของอาร์เรย์และโปรแกรมการเรียงแบบควิกซอร์ท การสร้างโปรแกรมต่าง ๆ ให้สามารถทำงานได้ในแบบขนานจะต้องใช้การวิเคราะห์และทดลองในการสร้างและแบ่งการทำงานให้สามารถทำงานแยกกันได้ซึ่งอาจจะผลที่ได้นั้นอาจสร้างผลการทำงานที่มีประสิทธิภาพที่สูงขึ้นมากหรืออาจจะมีประสิทธิภาพไม่เป็นไปตามที่คาดได้ซึ่งขึ้นอยู่กับลักษณะของโปรแกรมเป็นสำคัญ ดังนั้นในการสร้างภาษาที่ช่วยในการเขียนโปรแกรมแบบขนานจึงมีข้อจำกัดอย่างสูง การสร้างฟังก์ชันหรือ API (Application Programming Interface) ที่ทำงานแบบขนานเฉพาะแบบ เช่น โปรแกรมเสริช โปรแกรมการจัดเรียง โปรแกรมโครงสร้างข้อมูลมาตรฐาน ถึงแม้จะมีปัญหาน้อยแต่ก็ทำการประยุกต์ใช้ได้น้อยเช่นเดียวกัน สำหรับการสร้างคำสั่งประเภท All-in-One ที่สามารถทำให้โปรแกรมต่าง ๆ ทำงานได้ในแบบขนานนั้นจะเกิดปัญหาขึ้นเนื่องมาจากความซับซ้อนของตัวโปรแกรมเองจึงไม่สามารถทำได้ ดังนั้นการสร้างคำสั่งช่วยการทำงานแบบขนานพื้นฐานจึงเป็นวิธีการที่มีความเป็นไปได้สูงสุด

3.1.1 ขั้นตอนการสร้างตัวแปลโปรแกรม

1. สร้างโทเคนเพื่อใช้ในการสแกน พาร์สและตัวสร้างรหัส ดังนี้ tkFOR, tkPRAGMA, tkPARALLEL
2. เพิ่มโครงสร้างแกรมมาสำหรับคำสั่งใหม่ คือ แกรมมา for, แกรมมาคำชี้แนะตัวแปลโปรแกรม pragma
3. สร้างและปรับแต่งแอคชันรูทีนหรือคำสั่งภายในตัวแปลโปรแกรม
4. จัดเก็บข้อมูลจากแกรมมาลงในโครงสร้างข้อมูลการพาร์สแบบลิงค์ลิสต์
5. สร้างโครงสร้างข้อมูลของฟังก์ชันใหม่
6. สร้างตัวสร้างรหัสในส่วนของคำสั่งเดิม คือ แกรมมา for, แกรมมาคำชี้แนะตัวแปลโปรแกรม pragma
7. สร้างตัวสร้างรหัสในส่วนของคำสั่งใหม่ คือ if-then, if-then-else, assignment and expression

คำสั่ง #pragma parallel for จะสร้างการทำงานแบบขนานขึ้นโดยจะทำการกระจายการทำงานของลูปให้แต่ละหน่วยประมวลผลทำงาน สำหรับโปรแกรมการคูณเมทริกซ์ที่มีลูปวนตามแถว ตามหลัก และตามจำนวนของสมาชิกที่จะทำการคูณซึ่งจะเกิดการแบ่งงานที่ลูปชั้นในสุดโดยในกรณีที่มีจำนวนสมาชิกอยู่ k และมีจำนวนหน่วยประมวลผลอยู่ N จะได้ว่าแต่ละหน่วยประมวลผลจะทำการคำนวณทั้งหมด k/N สำหรับการบวกแบบรีดักชันในลูปชั้นแรกจะเป็นการวนตามจำนวนชั้นของต้นไม้ที่จำลองการบวกที่มีความสูง log k โดย k เป็นจำนวนข้อมูลที่ต้องการหาผลรวม และลูปชั้นในเป็นการทำการบวกตามจำนวนการบวกในแต่ละชั้นซึ่งจะเกิดการกระจายการทำงานที่ลูปชั้นนี้ สมมติว่าในแต่ละชั้นต้องทำการบวกเป็นจำนวน j ข้อมูล และมีจำนวนหน่วยประมวลผลอยู่ N หน่วย จะได้ว่าในแต่ละหน่วยประมวลผลจะทำการบวกอยู่ j/N ครั้งนั่นเอง และสำหรับโปรแกรมการเรียงแบบคี่-คู่ที่มีลูปชั้นแรกเป็นจำนวนรอบที่น้อยกว่าจำนวนข้อมูลทั้งหมดอยู่หนึ่งและลูปชั้นในที่จะทำงานอยู่ k/2 ครั้งสลับกันเริ่มจากตัวคู่และตัวคี่ ดังนั้นถ้ามีหน่วยประมวลผล N หน่วย จะได้ว่าแต่ละหน่วยประมวลผลจะเกิดการทำงานอยู่รอบละ (k/2)/N ครั้ง

ลักษณะการทำงานแบบขนานของไมโครคอนโทรลเลอร์พร็อพเพลเลอร์จะต้องสั่งให้แต่ละหน่วยประมวลผลย่อยทำงานด้วยคำสั่ง COGINIT หรือ COGNEW ซึ่งผู้เขียนโปรแกรมจะต้องระบุพารามิเตอร์ที่เป็นฟังก์ขันหรือโปรแกรมย่อยที่ผู้เขียนโปรแกรมต้องการให้ทำงานบนหน่วยประมวลผลย่อย ดังนั้นตัวสร้างรหัสที่สร้างจะต้องสร้างฟังก์ชันใหม่เพื่อให้ทำงานแบบขนานได้โดยการนำรหัสการทำงานภายใต้บล็อกคำสั่งของคำสั่ง for ที่ต้องการให้ทำงานแบบขนานไปเป็นคำสั่งของฟังก์ขันที่สร้างขึ้นใหม่ที่จะถูกเรียกใช้โดยคำสั่ง COGINIT เพื่อให้เกิดการทำงานอยู่บนแต่ละหน่วยประระมวลผลย่อยตามต้องการ

เนื่องจากตำสั่ง COGINIT ใช้เวลาในการทำงานมากเพราะต้องทำการนำรหัสการทำงานที่ผู้เขียนโปรแกรมกำหนด ไปไว้ยังหน่วยประมวลผลย่อยที่ต้องการ ทำให้ใช้เวลามากกว่าคำสั่งทั่วไป และถ้าหากคำสั่งนี้อยู่ภายในลูปที่มีรอบการทำงานจำนวนมาก ก็จะทำให้ใช้เวลาทำงานที่สูงมาก ด้วยเหตุนี้การสั่งให้หน่วยประมวลผลย่อยทำงานด้วยคำสั่ง COGINIT จึงนำไปไว้ในช่วงแรกของการทำงาน ซึ่งจะเกิดการทำงานเพียงครั้งเดียวไม่จำเป็นต้องให้คำสั่งนี้อยู่ในลูปที่จะทำให้เกิดการเรียกคำสั่งนี้หลายครั้งซึ่งจะทำให้เสียเวลาในการทำงานมาก

คำสั่ง #pragma parallel for จะทำการแปลการทำงานแบบลูปจากในรูปที่ทำงานในแบบตามลำดับให้เป็นการทำงานในแบบขนาน โดยจะทำการแบ่งการทำงานให้กับแต่ละหน่วยประมวลผลด้วยวิธีที่มีความซับซ้อนน้อยเพื่อให้สามารถประยุกต์ใช้งานได้กว้าง โดยจะให้แต่ละหน่วยประมวลผลทำงานด้วยรหัสการทำงานเดียวกันแต่จะกำหนดรอบของการทำงานของลูปที่แตกต่างกัน โดยที่งานของแต่ละหน่วยประมวลผลย่อยจะถูกแบ่งออกจะงานรวมเป็นปริมาณที่เท่า ๆ กัน และตัวแปรทุกตัวรวมถึงตัวแปรแบบอาร์เรย์จะถูกเก็บบนโปรแกรมด้วยชนิด long

เนื่องจากการทำงานแบบขนานจะเกิดการกระจายงานให้กับหน่วยปรมวลผลต่าง ๆ ที่มีอยู่ในระบบและจะต้องจัดการไม่ให้เกิดการทำงานผิดพลาด เช่น การเขียนข้อมูลทับซ้อนลงในตำแหน่งเดียวกันหรืออ่านข้อมูลที่ไม่ใช้ข้อมูลล่าสุดที่ถูกต้อง เป็นต้น ซึ่งสำหรับงานนี้ก็จะใช้ฟังก์ชัน lockset และ lockclr สำหรับการขอทำการล็อคและยกเลิกการล็อคตามลำดับ ที่เป็นฟังก์ชั่นพื้นฐานที่มีให้ใช้ในไมโครคอนโทรลเลอร์พร็อพเพลเลอร์ ควบคู่กับตัวแปรที่ใช้นับ คือ busy_core และ busy_data โดยทั้งสองตัวแปรจะถูกล็อคในทุก ๆ ครั้งที่จะถูกเพิ่มค่าหรือลดค่า ทั้งนี้เพื่อไม่ให้เกิดความผิดพลาดทั้งในการอ่านและเขียนค่าภายในตัวแปรทั้งสอง ตัวแปร busy_core จะถูกใช้ในการแสดงสถานะการทำงานของหน่วยประมวลผล ถ้าหากมีค่ามากกว่า 1 ก็จะหมายถึงว่ามีการทำงานของหน่วยประมวลผลอยู่ แต่ถ้าหามีค่าเป็น 0 ก็จะแสดงว่าไม่มีหน่วยประมวลผลที่กำลังทำงานอยู่หรือการทำงานเสร็จสิ้นแล้วพร้อมที่จะทำงานต่อไปได้ และตัวแปร busy_data จะถูกใช้สำหรับเป็นสถานะความพร้อมของข้อมูล โดยถ้าหากค่าของตัวแปรเป็น 1 จะหมายถึงข้อมูลยังไม่พร้อมใช้งานแต่ถ้าหากมีค่าเป็น 0 จะหมายถึงข้อมูลพร้อมใช้งานแล้วแต่ละหน่วยประมวลผลสามารถเริ่มทำการประมวลผลได้

การแปลรหัสของตัวอย่างการทำงานของคำสั่ง #pragma parallel for ดังภาพที่ 3.1 จะได้รหัสการทำงานเป็นสองส่วน คือ ส่วนของการกระจายงาน และส่วนของฟังก์ชันที่จะใช้คำนวณในแต่ละหน่วยประมวลผลย่อย สำหรับในส่วนของการกระจายงานดังภาพที่ 3.2 จะมีลูปเพื่อวนกระจายงานโดยแต่ละรอบจะกระจายงานได้เท่ากับจำนวนของหน่วยประมวลผลย่อยที่มีดังนั้นถ้ามีงานทั้งหมดคือ N จะมีจำนวนรอบการทำงานเท่ากับ N/core_count ภายในลูปจะมีส่วนที่ใช้สำหรับ synchronize การทำงานระหว่างหน่วยประมวลผล คือ ตัวแปร busy__core แสดงสถานะการทำงานของหน่วยประมวลผลย่อย และ ตัวแปร busy__data แสดงสถานะความพร้อมของข้อมูล โดยก่อนที่จะทำงานจะรอให้หน่วยประมวลผลย่อยทำงานให้เสร็จเสียก่อนโดยการตรวจสอบตัวแปร busy__core ถ้าหากมีค่ามากกว่า 1 แสดงว่าหน่วยประมวลผลย่อยยังไม่เสร็จสิ้นการทำงานแต่ถ้าหากเป็น 0 ก็แสดงว่าทำงานเสร็จเรียบร้อยแล้ว ก็จะทำการกำหนดค่าให้กับตัวแปร g_i ซึ่งเป็นตัวแปรแบบโกลบอลที่จะถูกนำไปใช้ในฟังก์ชันการคำนวณโดยหน่วยประมวลผล

ย่อย โดยค่าของตัวแปรนี้จะเปลี่ยนค่าไปตามรอบการวนของลูปเพื่อคำนวนตามที่ผู้เขียนโปรแกรมกำหนด แล้วจึงทำการกำหนดค่าของตัวแปร busy__data ให้เป็น 0 เพื่อเป็นตัวบ่งบอกให้แก่หน่วยประมวลผลย่อยว่าข้อมูลพร้อมใช้งานแล้ว สามารถเริ่มต้นทำงานได้ และสุดท้ายจะเป็นการตรวจสอบว่าแต่ละหน่วยประมวลผลได้นำข้อมูลไปใช้ครบแล้วหรือไม่ โดยตัวแปรแต่ละตัวจะทำการเพิ่มค่าให้กับตัวแปร busy__data เพื่อแสดงให้เห็นว่าเกิดการใช้งานข้อมูลแล้ว

```
#pragma parallel for
   for (i = 1; i <= 256; i = i+1)
      x[i] = x[i] + i;
```

ภาพที่ 3.1 โปรแกรมตัวอย่างกระจายการทำงานคำสั่ง for ในภาษา RZ

```
repeat _i from 1 to ( N / core_count )
  repeat until busy__core == 0

  g_i := ( ( _i - 1 ) * core_count )

  busy__data := 0
  repeat until busy__data == core_count
```

ภาพที่ 3.2 ผลการแปลโปรแกรมตัวอย่างกระจายการทำงานคำสั่ง for

สำหรับในส่วนของฟังก์ชันที่ใช้คำนวณในแต่ละหน่วยประมวลผลย่อยแสดงในภาพที่ 3.3 จะเป็นลูปวนทำงานอยู่ตลอดเวลาแต่จะมีการ synchronize เพื่อให้ทำงานสอดคล้องกันกับหน่วยประมวลผลอื่น ๆ ที่ทำงานร่วมกัน โดยเริ่มแรกจะทำการตรวจสอบว่าข้อมูลพร้อมแล้วหรือไม่ โดย

การตรวจสอบค่าของตัวแปร busy__data ถ้ามากกว่า 1 แสดงว่ายังไม่พร้อมต้องรอจนว่าจะมีค่าเป็น 0 จึงจะทำงานต่อไป ซึ่งก็คือ ทำการเพิ่มค่าของตัวแปร busy__core และ busy__data เพื่อแสดงว่าหน่วยประมวลผลย่อยกำลังทำงานอยู่และได้รับข้อมูลแล้วตามลำดับ แต่เนื่องจากตัวแปรทั้งสองนี้เป็นตัวแปรที่ใช้ระหว่างกันระหว่างหน่วยประมวลผลย่อยทั้งหมด จึงอาจเกิดการอ่านเขียนผิดพลาดขึ้นได้ ดังนั้นจึงต้องมีการ synchronize ด้วยคำสั่ง lock ให้รอจนว่าจะว่างไม่มีหน่วยประมวลผลย่อยทำงานอยู่กับตัวแปรนี้ ด้วยคำสั่ง lockset ที่จะให้ค่าคืนเป็น 1 ถ้าหากยังคงมีการล็อค อยู่ และจะมีค่าเป็น 0 ถ้าหากไม่ได้ถูกล็อคอยู่ แล้วจึงทำการเพิ่มหรือลดค่าให้กับตัวแปร และเนื่องจากมีตัวแปรที่จะทำการเพิ่มค่าอยู่สองตัวแปรก็จึงมี lock อยู่สองตัวเช่นเดียวกัน เพราะถ้าหากใช้ lock ตัวเดียวกันจะทำให้การทำงานช้าลงเนื่องจากต้อง lock ตัวแปรเป็นเวลานาน และเมื่อข้อมูลพร้อมเรียบร้อยแล้ว จึงทำการคำนวณตามที่ผู้เขียนโปรแกรมกำหนดอยู่ภายในบล็อคของคำสั่ง for ที่ถูกนำมาไว้ในฟังก์ชันการคำนวณสำหรับหน่วยประมวลผลย่อย และสุดท้ายจะเป็นการลดค่าของตัวแปร busy__core เพื่อบ่งบองถึงการทำงานของหน่วยประมวลผลย่อยนี้ได้เสร็จสิ้นลงแล้ว

```
PUB function_calculation | i
  repeat
    repeat until busy__data == 0


    repeat until not lockset ( l_id )
    busy__core += 1
    lockclr ( l_id )


    repeat until not lockset ( l_id2 )
    busy__data += 1
    lockclr ( l_id2 )


    i := g_i + cogid - 1
    x [i - 1] = x [i - 1] + i


    repeat until not lockset ( l_id )
    busy__core -= 1
    lockclr ( l_id )
```

ภาพที่ 3.3 ฟังก์ชันการคำนวณในผลการแปลโปรแกรมตัวอย่างกระจายการทำงานคำสั่ง for

สำหรับการแปลรหัสของตัวอย่างการทำงานของคำสั่ง #pragma parallel for reduction ดังภาพที่ 3.4 จะได้ผลการแปลรหัสดังภาพที่ 3.5 เป็นสองส่วนเช่นเดียวกัน ในส่วนของการกระจายงานจะประกอบไปด้วยลูปสองชั้นโดยรอบแรกจะเป็นการวนเท่ากับจำนวนชั้นของโครงสร้างต้นไม้รีดักชันเนื่องจากการหาผลลัพธ์แบบรีดักชันจะมีโครงสร้างแบบต้นไม้ที่มีความสูงเท่ากับ log N เมื่อ N คือจำนวนทั้งหมด ลูปชั้นถัดมาจะเป็นการวนรอบตามจำนวนของโหนดในชั้นนั้น ๆ โดยแบ่งเป็นรอบ ๆ ตามจำนวนของหน่วยประมวลผลย่อยที่ใช้ในการกระจายงาน โดยภายในลูปจะเป็นการกำหนดค่าตัวแปรต่าง ๆ ที่จะใช้ในการคำนวน คือ ตัวแปร g_i สำหรับเป็นค่า

ตำแหน่งเริ่มต้นสำหรับแต่ละรอบการทำงาน และ g_j เป็นความกว้างของต้นไม้ในแต่ละชั้น ในฟังก์ชันการคำนวณสำหรับแต่ละหน่วยประมวลผลย่อย

```
#pragma parallel for reduction
   for (i = 1; i <= 256; i = i+1)
      x = x + data[i];
```

ภาพที่ 3.4 โปรแกรมการบวกแบบรีดักชันภาษา RZ

```
repeat _i from _LOG2N to 1
  repeat _j from 1 to ( _POW [ _i - 1 ] /
     core_count ) + 1
       repeat until busy__core == 0
       g_i := ( ( _j - 1 ) * core_count )
       g_j := _POW [ _i - 1 ]
       busy__data := 0
       repeat until busy__data == core_count
```

ภาพที่ 3.5 ผลการแปลโปรแกรมโปรแกรมการบวกแบบรีดักชัน

สำหรับส่วนของฟังก์ชันการคำนวณดังภาพที่ 3.6 จะมีลักษณะคล้ายกับโปรแกรมตัวอย่างของ #pragma parallel for โดยจะมีลูปชั้นนอกที่วนอยู่ตลอดเวลา และภายในมีการตรวจสอบค่าตัวแปร busy__data เพื่อความพร้อมของข้อมูลที่จะนำมาใช้ซึ่งถ้าหากข้อมูลพร้อมใช้แล้วก็จะทำงานต่อไปก็คือ การเพิ่มค่าของตัวแปร busy__core และ busy__data เพื่อแสดงว่าหน่วย

ประมวลผลย่อยกำลังทำงานอยู่และได้ข้อมูลแล้วตามลำดับ โดยการเพิ่มค่าของตัวแปรทั้งสองนี้จะต้อง synchornize ด้วยฟังก์ชัน lockset และ lockclr เช่นเดียวกัน ถัดมาจะเป็นการกำหนดค่า i และ j เพื่อใช้กำหนดค่าตำแหน่งของแต่ละหน่วยประมวลผลย่อยใช้คำนวณตามแต่ละตำแหน่งที่หน่วยประมวลผลย่อยแต่ละตัวจะต้องคำนวณ และสุดท้ายจะเป็นการลดค่าของตัวแปร busy__core เพื่อแสดงว่าการทำงานได้เสร็จสิ้นแล้ว

```
PUB function_calculation | i, j

  repeat

    repeat until busy__data == 0


    repeat until not lockset ( l_id )

    busy__core += 1

    lockclr ( l_id )


    repeat until not lockset ( l_id2 )

    busy__data += 1

    lockclr ( l_id2 )


    i := g_i + cogid - 1

    j := g_j

    if ( i - 1 < j )

      data [ i - 1 ] = data [ i - 1 ] +
         data [ ( i - 1 ) + j ]


    repeat until not lockset ( l_id )

    busy__core -= 1

    lockclr ( l_id )
```

ภาพที่ 3.6 ฟังก์ชันการคำนวณในผลการแปลโปรแกรมโปรแกรมการบวกแบบรีดักชัน

3.2 การทำงานแบบตามลำดับ

โปรแกรมแบบตามลำดับ (sequential) เป็นโปรแกรมแบบทั่วไปที่เขียนขึ้นเพื่อให้ทำงานบนหน่วยประมวลผลแบบเดี่ยว ไม่มีการแบ่งงานออกเป็นส่วน ๆ เพื่อกระจายให้หน่วยประมวลผลอื่น ๆ ทำงานในส่วนที่กำหนด ไมโครคอนโทรลเลอร์พร็อพเพลเลอร์มีสถาปัตยกรรมแบบหลายแกนสามารถทำงานได้ทั้งในแบบขนานและแบบตามลำดับ ตัวอย่างโปรแกรมการคูณเมทริกซ์ที่ทำงานแบบตามลำดับด้วยภาษาสปินแสดงดังภาพที่ 3.7 โปรแกรมนี้ประกอบด้วยส่วนประกาศตัวแปรค่าคงที่ ส่วนประกาศตัวแปรแบบโกลบอล และส่วนฟังก์ชันการทำงาน

```
CON
  size = 3
VAR
  long m[size * size]
  long a[size * size]
  long b[size * size]
OBJ
 vga : "vga_text"


PUB main | i, j, k
  i := 0
  repeat until not (i < size)
    j := 0
    repeat until not (j < size)
      k := 0
      repeat until not (k < size)
        m[((i * size) + j)] :=
          (m[((i * size) + j)]
           + (a[((i * size) + k)]
           * b[((k * size) + j)]))
        k := (k + 1)
      j := (j + 1)
    i := (i + 1)
```

ภาพที่ 3.7 โปรแกรมการคูณเมทริกซ์ที่ทำงานแบบตามลำดับด้วยภาษาสปิน

ตารางที่ 3.1 แสดงผลเวลาที่ใช้ในการทำงานโปรแกรมการคูณเมทริกซ์ในแต่ละขนาดของเมทริกซ์ที่ทำงานแบบขนานแบบ 3 และ 6 หน่วย และแบบตามลำดับ มีหน่วยเป็นจำนวนรอบการ

ทำงานหรือความถี่ของสัญญาณนาฬิกา และตารางที่ 3.2 แสดงอัตราส่วนเวลาที่เพิ่มขึ้นระหว่างเวลาที่ใช้ในโปรแกรมการคูณเมทริกซ์แบบขนานที่รวมส่วนการกระจายและควบคุมการทำงานหน่วยประมวลผลอื่นด้วย และเวลาที่ใช้ในแบบตามลำดับ ซึ่งจะเห็นได้ว่าการทำงานแบบขนานยังคงใช้เวลาในการทำงานที่น้อยกว่าและมีความเร็วเพิ่มขึ้นประมาณ 1.01 ถึง 1.82 เท่า

ตารางที่ 3.1 แสดงแสดงเวลาที่ใช้ในโปรแกรมการคูณเมทริกซ์ในแบบขนานและแบบตามลำดับ

NO. CORE USED	MATRIX SIZE				
	3x3	6x6	12x12	24x24	48x48
6-core	664,768	1,858,752	10,731,328	77,923,744	601,103,776
3-core	664,624	2,553,200	16,655,936	125,952,176	983,617,616
Sequential	673,280	2,805,344	18,613,664	140,127,776	1,092,852,512

ตารางที่ 3.2 แสดงอัตราส่วนเวลาที่เพิ่มขึ้นระหว่างเวลาที่ใช้ในโปรแกรมการคูณเมทริกซ์แบบขนาน และเวลาที่ใช้ในแบบตามลำดับ

NO. CORE USED	MATRIX SIZE				
	3x3	6x6	12x12	24x24	48x48
6-core	1.01	1.51	1.73	1.80	1.82
3-core	1.01	1.10	1.12	1.11	1.11

ตารางที่ 3.3 แสดงผลเวลาที่ใช้ในการทำงานของโปรแกรมการบวกแบบรีดักชันในแต่ละขนาดของข้อมูลที่ทำงานแบบขนานแบบ 3 และ 6 หน่วย และแบบตามลำดับ มีหน่วยเป็นจำนวนรอบการทำงานหรือความถี่ของสัญญาณนาฬิกา ตารางที่ 3.4 แสดงอัตราส่วนความเร็วเปรียบเทียบระหว่างการทำงานแบบขนานและแบบตามลำดับ ซึ่งจะเห็นว่าการทำงานแบบขนานทำงานได้ช้ากว่าแต่ถ้าพิจารณาปัญหาที่ขนาดใหญ่มากขึ้น การทำงานแบบขนานก็จะทำงานได้เร็วยิ่งขึ้นเนื่องจากโปรแกรมส่วนการกระจายงานมีการเรียกใช้ข้อมูลค่า logarithm และค่าเลขยกกำลังเพื่อกำหนดจำนวนรอบของการกระจายงานซึ่งการอ่านและเขียนข้อมูลบนหน่วยความจำเป็นส่วนที่ใช้เวลามากกว่าคำสั่งอื่น ๆ การปรับปรุงส่วนนี้ให้เร็วขึ้นก็จะสามารถทำให้การทำงานโดยรวมเร็วขึ้นได้ และจะเห็นว่าคำสั่ง #pragma parallel for ไม่มีการกระจายงานแบบรีดักชันจึงไม่เกิดการเรียกค่าข้อมูล logarithm และเลขยกกำลังจึงทำงานได้รวดเร็วกว่า

ตารางที่ 3.3 แสดงผลเวลาที่ใช้ในการทำงานของโปรแกรมการบวกแบบรีดักชันในแบบขนาน และแบบตามลำดับ

NO. CORE USED	DATA SIZE			
	512	1024	2048	4096
6-core	2,598,528	4,912,656	9,661,872	18,994,672
3-core	3,413,920	6,672,080	13,219,264	26,252,944
Sequential	1,534,032	3,065,936	6,129,744	12,257,360

ตารางที่ 3.4 แสดงอัตราส่วนเวลาที่เพิ่มขึ้นระหว่างเวลาที่ใช้ในโปรแกรมการบวกแบบรีดักชัน และเวลาที่ใช้ในแบบตามลำดับ

NO. CORE USED	DATA SIZE			
	512	1024	2048	4096
6-core	0.59	0.62	0.63	0.65
3-core	0.45	0.46	0.46	0.47

ตารางที่ 3.5 แสดงผลเวลาที่ใช้ในการทำงานของโปรแกรมการเรียงแบบคี่-คู่ในแต่ละขนาดของข้อมูลที่ทำงานแบบขนานแบบ 3 และ 6 หน่วย และแบบตามลำดับ มีหน่วยเป็นจำนวนรอบการทำงานหรือความถี่ของสัญญาณนาฬิกา และตารางที่ 3.6 แสดงอัตราส่วนความเร็วที่เพิ่มขึ้นที่เพิ่มขึ้นเป็น 1.93 ถึง 1.54 เท่าสำหรับข้อมูลขนาด 32 ถึง 512 จำนวน ตามลำดับ

ตารางที่ 3.5 แสดงผลเวลาที่ใช้ในการทำงานของโปรแกรมการเรียงแบบคี่-คู่ในแบบขนาน และแบบตามลำดับ

NO. CORE USED	DATA SIZE				
	32	64	128	256	512
6-core	3,333,888	11,973,168	42,336,923	167,622,768	651,919,984
3-core	4,919,696	17,250,560	67,424,000	262,783,664	1,048,402,160
Sequential	6,442,304	20,160,864	70,335,568	261,391,776	1,001,311,712

ตารางที่ 3.6 แสดงอัตราส่วนเวลาที่เพิ่มขึ้นระหว่างเวลาที่ใช้ในโปรแกรมการเรียงแบบคี่-คู่ และเวลาที่ใช้ในแบบตามลำดับ

NO. CORE USED	DATA SIZE				
	32	64	128	256	512
6-core	1.93	1.68	1.66	1.56	1.54
3-core	1.31	1.17	1.04	0.99	0.96

ในการสร้างโปรแกรมให้สามารถทำงานได้ในแบบขนาน โดยทั่วไปมีความจำเป็นที่จะต้องมีส่วนของโปรแกรมที่ทำหน้าที่ในการกระจายการทำงานและควบคุมการทำงานให้แต่ละหน่วยประมวลผลประสานการทำงานกันได้อย่างถูกต้อง สำหรับในภาษาสปินส่วนกระจายการทำงานคือส่วนการเรียกใช้ฟังก์ชัน COGINIT และส่วนลูปที่ใช้กระจายการทำงานออกเป็นรอบ ๆ ในส่วนควบคุมและประสานงานจะเป็นส่วนที่ใช้ฟังก์ชัน lockset และ lockclr และการใช้ตัวแปร busy__core และ busy__data นั่นเอง

ตารางที่ 3.7 แสดงเวลาที่ใช้ในการทำงานของโปรแกรมสำหรับทำงานแบบขนานที่ใช้หน่วยประมวลผลในการคำนวณเพียงหน่วยประมวลผลเดียวและโปรแกรมที่ทำงานแบบตามลำดับ มีหน่วยเป็นจำนวนรอบการทำงานหรือความถี่ของสัญญาณนาฬิกา โปรแกรมแบบขนานที่แสดงในตารางถูกกำหนดให้ใช้เพียงหน่วยประมวลผลเดียวในการคำนวณเพื่อใช้ในการวัดส่วนควบคุมการทำงานแบบขนานหรือส่วนโอเวอร์เฮด (overhead) ซึ่งถึงแม้ว่าการทำงานแบบขนานจะใช้เวลาในการทำงานที่น้อยกว่า แต่การทำงานแบบขนานมีโอเวอร์เฮดอยู่สูงมากประมาณระหว่างร้อยละ 32.8 ถึง 65.0 สำหรับเมทริกซ์ขนาด 3x3 ถึง 48x48 ดังแสดงไว้ในตาราง และเนื่องจากหน่วยความจำของไมโครคอนโทรลเลอร์มีขนาดค่อนข้างเล็กและจำกัด ขนาดของปัญหาจึงไม่สามารถกำหนดให้มีขนาดที่ใหญ่มาก ๆ ได้

ตารางที่ 3.7 แสดงเวลาที่ใช้ในการทำงานของโปรแกรมสำหรับทำงานแบบขนานที่ใช้หน่วยประมวลผลในการคำนวณเพียงหน่วยประมวลผลเดียวและโปรแกรมที่ทำงานแบบตามลำดับ และส่วนโอเวอร์เฮด

NO. CORE USED	MATRIX SIZE				
	3x3	6x6	12x12	24x24	48x48
1-core	1,001,888	5,506,224	40,596,128	317,650,768	3,119,684,672
Sequential	673,280	2,805,344	18,613,664	140,127,776	1,092,852,512
Overhead	32.8%	49.1%	54.1%	55.9%	65.0%

3.3 ภาษาสปินแบบขนาน

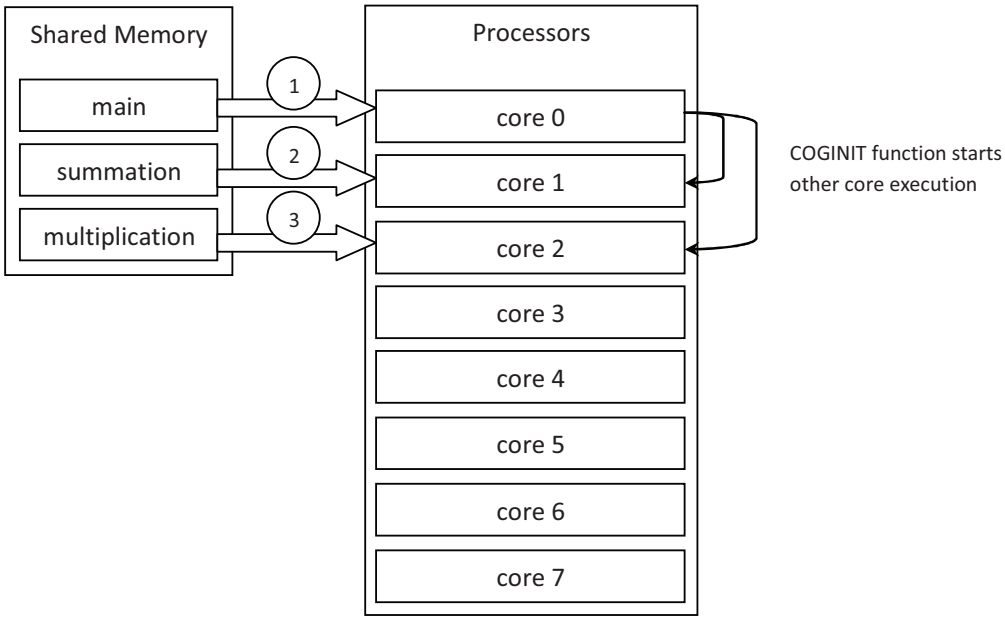
ไมโครคอนโทรลเลอร์พร็อพเพลเลอร์สามารถทำงานได้ทั้งในแบบเดี่ยวและแบบขนานครบทั้ง 8 หน่วยประมวลผลหรือเพียงบางหน่วยประมวลผลก็สามารถทำได้ ลักษณะการทำงานแบบขนานจะทำได้โดยการกำหนดให้หน่วยประมวลผลหลักหรือหน่วยประมวลผลอื่นที่ถูกสั่งทำงานแล้วสั่งให้หน่วยประมวลผลที่ว่างอยู่หรือหน่วยที่กำหนดให้ทำงานโดยการกำหนดคำสั่งที่อยู่ในรูปของฟังก์ชันตามที่ผู้เขียนโปรแกรมกำหนดดังภาพที่ 3.8 แสดงตัวอย่างโปรแกรมภาษาสปินที่ใช้หาค่าผลรวมและผลคูณของอาร์เรย์ โดยหลังจากโปรแกรมนี้ได้ถูกส่งไปยังหน่วยความจำภายในไมโครคอนโทรลเลอร์พร็อพเพลเลอร์ เมื่อเริ่มการทำงาน คำสั่งในฟังก์ชัน main จะถูกส่งเข้าไปยังหน่วยความจำภายในของหน่วยประมวลผลตัวที่ 0 ซึ่งเป็นตัวแรก และด้วยการเรียกใช้คำสั่ง COGINIT จะทำให้คำสั่งในฟังก์ชัน summation ที่ถูกระบุไว้เป็นพารามิเตอร์ ถูกส่งไปยังหน่วยประมวลผลตัวที่ 1 ที่ถูกระบุไว้เป็นพารามิเตอร์ของคำสั่ง COGINIT เช่นเดียวกัน และสำหรับคำสั่งถัดมาในทำนองเดียวกันก็จะถูกส่งฟังก์ชัน multiplication เข้าไปยังหน่วยประมวลผลตัวที่ 2 ซึ่งจะทำให้เกิดการทำงานแบบขนานที่มีสามหน่วยประมวลผลทำงานอยู่ไปพร้อม ๆ กัน โดยหน่วย

ประมวลผลตัวที่ 0 เป็นตัวกระจายงานและหน่วยประมวลผลตัวที่ 1 และ 2 เป็นตัวหาผลบวกและผลคูณตามลำดับ

```
VAR
  long sum_stack[8]
  long mul_stack[8]

  long data[16]
  long sum, mul

PUB main
  COGINIT (1, summation, @sum_stack)
  COGINIT (2, multiplication, @mul_stack)

PUB summation | i
  sum := 0
  repeat i from 0 to 15
    sum = sum + data[i]

PUB multiplication | i
  mul := 1
  repeat i from 0 to 15
    mul = mul * data[i]
```

ภาพที่ 3.8 ตัวอย่างโปรแกรมภาษาสปินทำงานแบบขนาน

แผนภาพการทำงานแสดงไว้ดังภาพที่ 3.9 เมื่อเริ่มการทำงาน โปรแกรมฟังก์ชัน main จากหน่วยความจำร่วมถูกส่งเข้าไปยังหน่วยความจำภายในของประมวลผลหน่วยที่ 0 และเมื่อหน่วยประมวลผลนี้ทำงานก็จะทำให้โปรแกรมฟังก์ชัน summation ถูกส่งเข้าไปยังหน่วยประมวลผลที่ 1 และเริ่มการทำงาน และโปรแกรมฟังก์ชัน multiplication ถูกส่งเข้าไปยังหน่วยประมวลผลที่ 2 และเริ่มการทำงานเช่นเดียวกัน

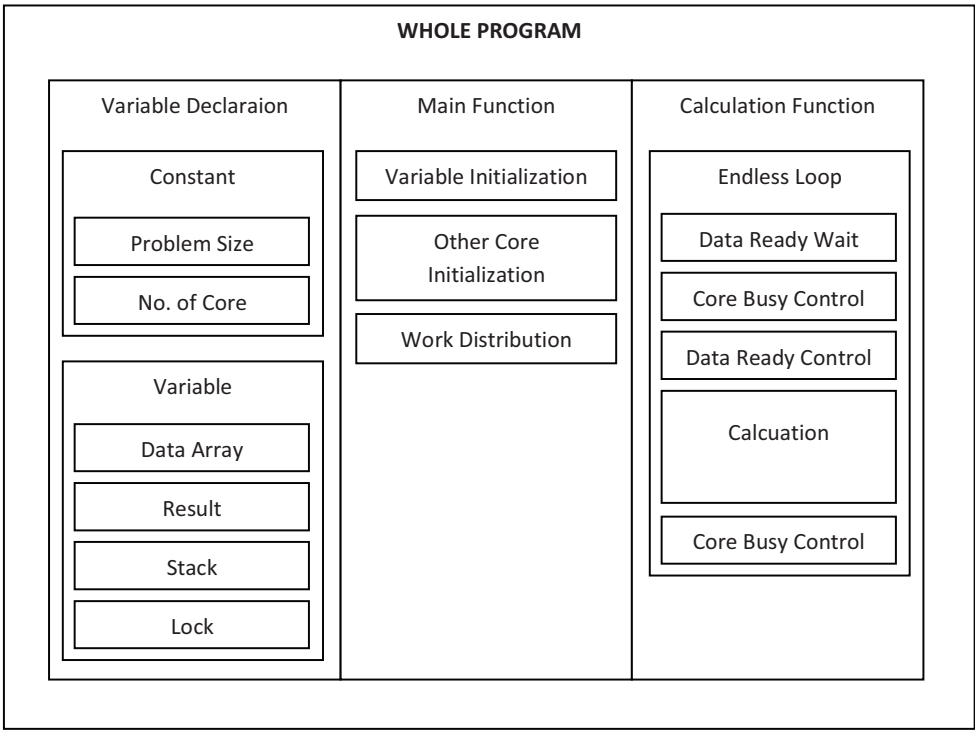


ภาพที่ 3.9 แผนภาพการทำงานของโปรแกรมภาษาสปินทำงานแบบขนาน

ฟังก์ชัน COGINIT เป็นฟังก์ชันที่ใช้ในการสั่งให้หน่วยประมวลผลที่กำหนดทำงานตามฟังก์ชันที่กำหนดมีพารามิเตอร์ทั้งหมด 3 ตัว คือ หมายเลขหน่วยประมวลผล ฟังก์ชันที่ต้องการให้ทำงานพร้อมระบุพารามิเตอร์ และตำแหน่งหน่วยความจำที่จะถูกใช้เป็นหน่วยความจำสแต็กสำหรับการทำงานภายในและการคืนค่าตัวแปร เมื่อฟังก์ชันนี้ถูกเรียกทำงาน จะเกิดการส่งโปรแกรมฟังก์ชันที่ผู้เขียนโปรแกรมกำหนดไว้ในพารามิเตอร์ไปยังหน่วยประมวลผลที่กำหนดไว้ในพารามิเตอร์เช่นเดียวกัน ในกรณีที่หน่วยประมวลผลที่กำหนดไว้กำลังทำงานอยู่ การทำงานเดิมจะ

ถูกเปลี่ยนให้เป็นโปรแกรมฟังก์ชันใหม่โดยทันทีซึ่งต่างจากฟังก์ชัน COGNEW ที่จะไม่ทำงานถ้าหากไม่มีหน่วยประมวลผลที่ว่างอยู่เท่านั้น

สำหรับโปรแกรมแบบขนานที่ใช้สำหรับการแปลภาษานี้จะมีลักษณ์ที่คล้ายกับการทำงานที่กล่าวมาข้างต้น ส่วนต่าง ๆ ของโปรแกรมแสดงไว้ดังภาพที่ 3.10 ซึ่งประกอบไปด้วยส่วนประกาศตัวแปร ส่วนฟังก์ชันหลัก และส่วนฟังก์ชันคำนวณ โดยส่วนประกาศตัวแปรจะประกอบไปด้วยส่วนประกาศค่าคงที่ต่าง ๆ เช่น ขนาดของปัญหา และจำนวนหน่วยประมวลผลย่อยที่จะใช้ช่วยกันทำงาน ส่วนประกาศตัวแปรต่าง ๆ เช่น ตัวแปรอาร์เรย์สำหรับเก็บข้อมูล ตัวแปรสำหรับเก็บผลการคำนวณ ตัวแปรสแต็ก ตัวแปรสำหรับการล็อค เป็นต้น ส่วนฟังก์ชันหลักนั้นภายในจะประกอบไปด้วย ส่วนกำหนดค่าเริ่มต้นของตัวแปร ส่วนการสั่งให้หน่วยประมวลผลย่อยทำงาน ส่วนการแบ่งและกระจายการทำงานให้กับแต่ละหน่วยประมวลผล สำหรับส่วนฟังก์ชันคำนวณจะประกอบด้วยส่วนต่าง ๆ ที่อยู่ภายใน คือ ส่วนวนลูปแบบไม่รู้จบที่ภายในประกอบไปด้วย ส่วนควบคุมประสานเวลาสำหรับรอให้ข้อมูลพร้อม ส่วนกำหนดค่าของตัวแปรสำหรับบ่งบอกถึงหน่วยประมวลผลกำลังทำงาน ส่วนกำหนดค่าของตัวแปรสำหรับบ่งบอกถึงการได้รับข้อมูลเรียบร้อยแล้ว ส่วนทำการคำนวณสำหรับข้อมูลที่ได้รับ และส่วนกำหนดค่าของตัวแปรสำหรับบ่งบอกถึงการเสร็จสิ้นการทำงาน

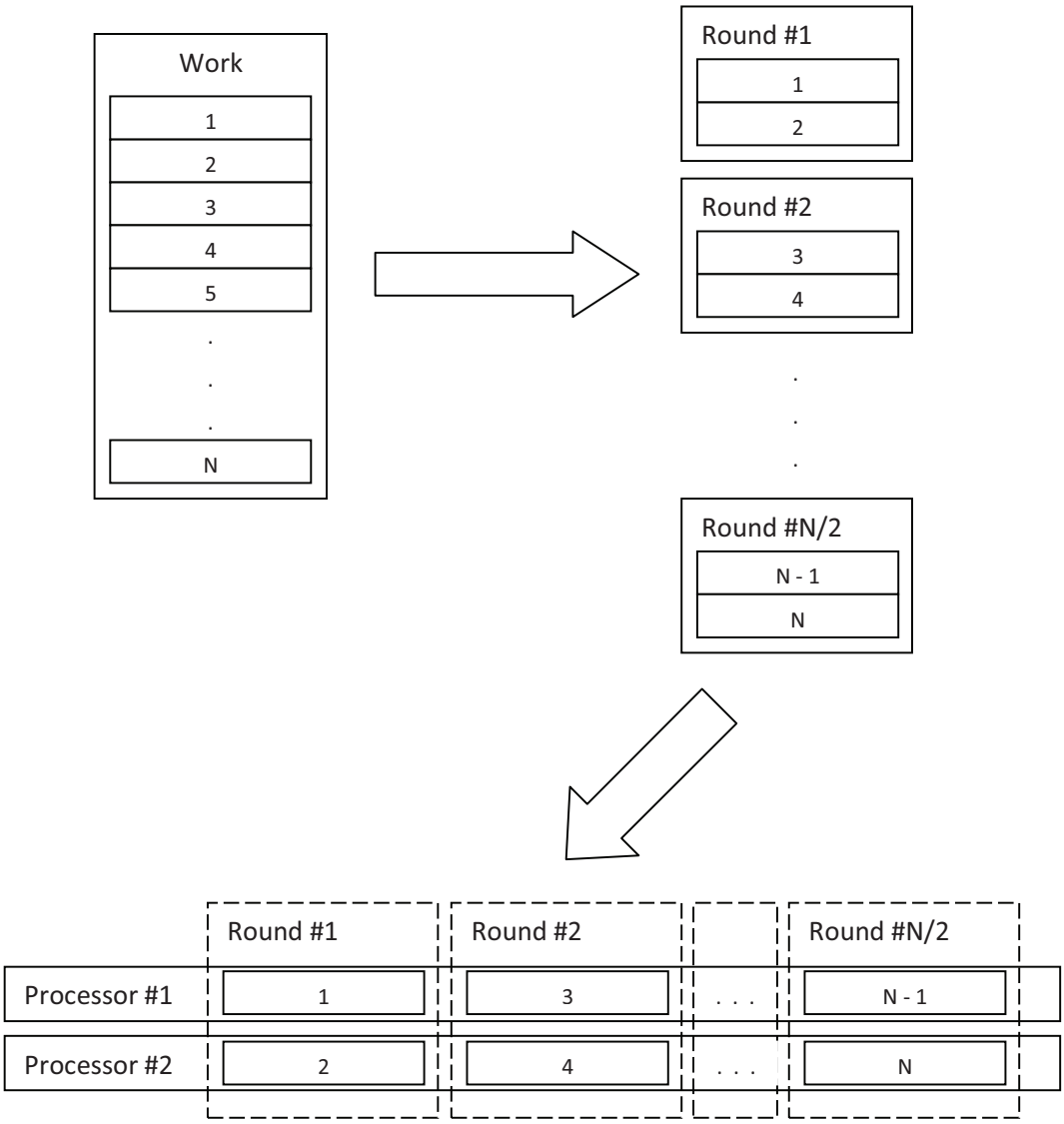


ภาพที่ 3.10 ส่วนประกอบของโปรแกรมแบบขนานโดยตัวแปลโปรแกรม

ในการสร้างโปรแกรมแบบขนานผู้เขียนโปรแกรมสามารถใช้คำชี้แนะตัวแปลภาษา (compiler directive) เพื่อช่วยให้โปรแกรมทำงานได้ในแบบขนานโดยตัวแปลภาษาจะทำการแปลให้เกิดการทำงานแบบขนานอย่างอัตโนมัติ คำชี้แนะตัวแปลภาษาที่สร้างขึ้น 2 แบบ คือ #pragma parallel for และ #pragma parallel for reduction สำหรับการกำหนดส่วนของโปรแกรมที่ต้องการให้เกิดการทำงานแบบขนาน ผู้เขียนโปรแกรมจะเป็นผู้กำหนดเองได้ สำหรับตัวอย่างการใช้งานจะกำหนดให้เกิดการทำงานที่ลูปชั้นในสุดจะทำให้โปรแกรมเข้าใจได้ง่าย และทำให้โปรแกรมดูไม่ซับซ้อนจนเกินไป ซึ่งผู้เขียนโปรแกรมสามารถทำความเข้าใจและนำไปประยุกต์ใช้ได้ตามที่ต้องการ และการทำงานในส่วนของการคำนวณของแต่ละหน่วยประมวลผลย่อย เนื่องจากโปรแกรมในภาษาสปินจะมีตัวจัดการโปรแกรมภายในที่จะคอยดึงคำสั่งครั้งละหนึ่งคำสั่ง ซึ่งคำสั่งในการคำนวนข้อมูลและจัดเก็บข้อมูลลงในตัวแปรเดิมจะไม่ถูกแทรกการทำงานจึงทำให้ไม่เกิดการทำงานผิดพลาดในการอ่านและเขียนข้อมูลระหว่างการทำงานของหลาย ๆ หน่วยประมวลผลร่วมกันได้

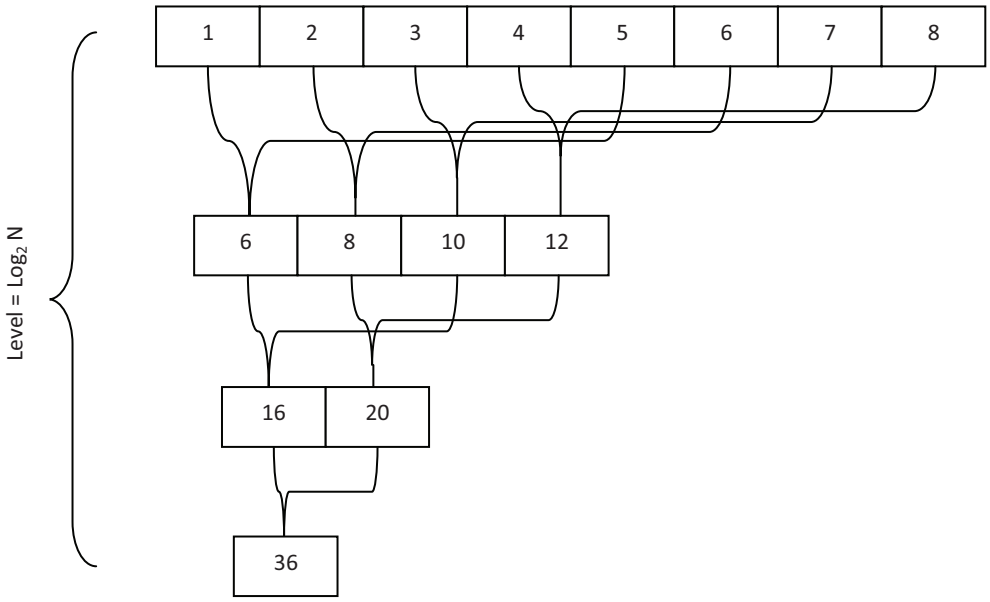
3.4 การทำงานของคำชี้แนะตัวแปลโปรแกรม

คำชี้แนะตัวแปลภาษา #pragma parallel for เป็นคำสั่งที่ผู้เขียนโปรแกรมสามารถใช้เพื่อให้คำสั่ง for เกิดการทำงานแบบขนาน กระจายงานให้หน่วยประมวลผลอื่น ๆ ช่วยกันทำงาน โดยการกระจายการทำงานจะถูกแบ่งออกเป็นรอบ ๆ โดยรอบการทำงานหนึ่งจะเกิดการทำงานเท่ากับจำนวนหน่วยประมวลผลที่ใช้ ดังนั้นถ้าหากคำสั่ง for ที่ผู้เขียนโปรแกรมต้องการมีการทำงานทั้งหมด 1000 ครั้ง ในการทำงานแบบขนานที่ใช้หน่วยประมวลผล 2 หน่วย ก็จะเกิดการทำงานทั้งหมด 500 รอบ แต่ละรอบจะทำการประมวลผลบนหน่วยประมวลผลทั้งสอง ซึ่งเมื่อทำงานครบ 500 รอบ ก็จะเกิดการทำงาน 1000 ครั้งตามที่ผู้เขียนโปรแกรมต้องการ ดังภาพที่ 3.11



ภาพที่ 3.11 แผนภาพการกระจายงาน

ในส่วนของคำชี้แนะตัวแปลภาษา #pragma parallel for reduction เป็นคำสั่งที่จะทำให้คำสั่ง for เกิดการทำงานแบบขนานเช่นเดียวกัน แต่จะทำการคำนวณหาผลลัพธ์ด้วยวิธีการแบบรีดักชันที่จะเกิดการหาผลลัพธ์แบบโครงสร้างต้นไม้ที่มีความสูงเป็น log2 (N) เมื่อ N คือจำนวนของข้อมูลทั้งหมดที่ต้องการหาผลลัพธ์ แสดงดังภาพที่ 3.12



ภาพที่ 3.12 แผนภาพการกระจายงานแบบรีดักชัน

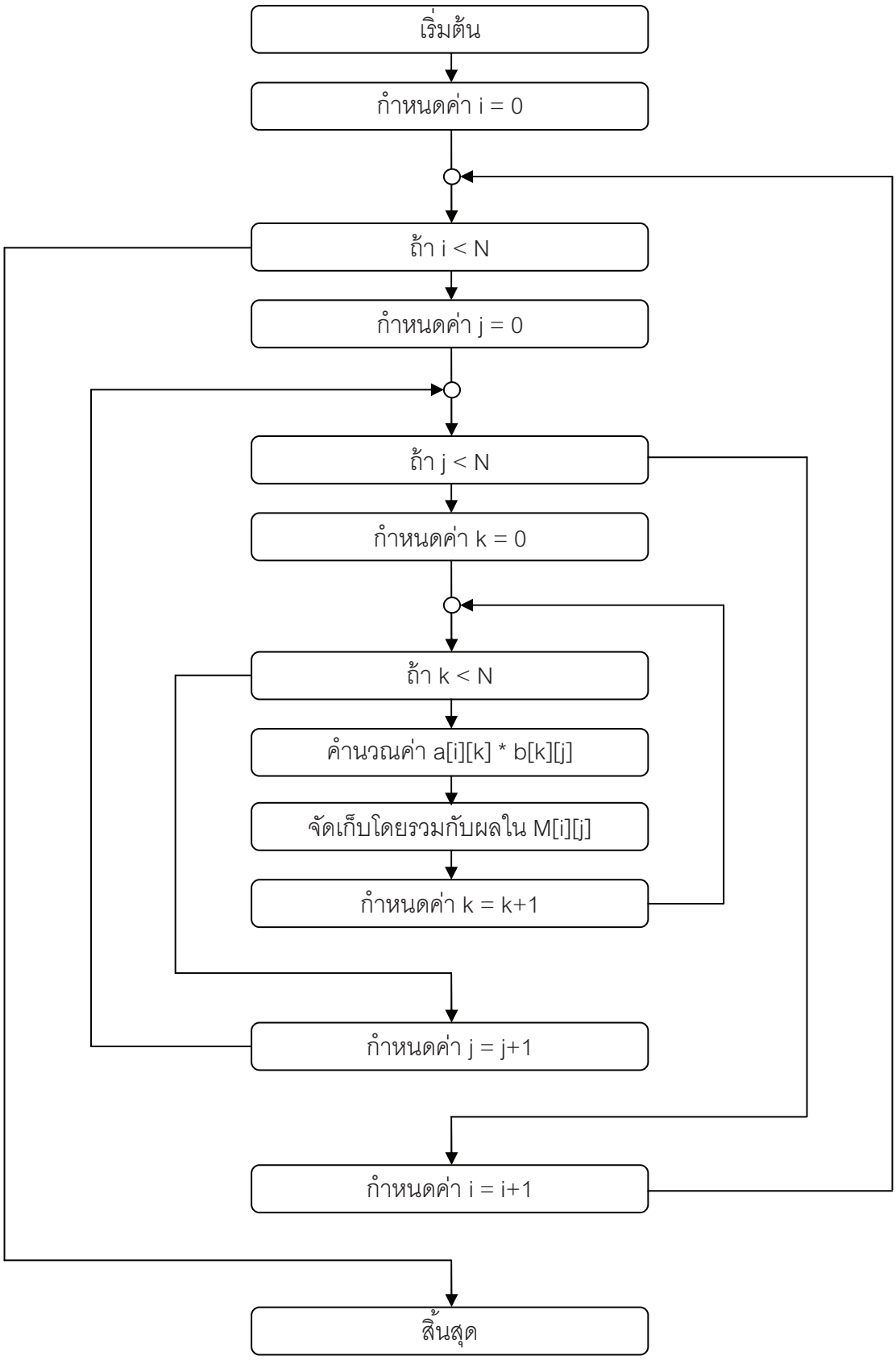
3.5 การคูณเมทริกซ์

การคูณเมทริกซ์เป็นการคูณของเมทริกซ์สองตัวโดยเมทริกซ์ที่สามารถคูณกันได้นั้นจะต้องมีขนาดที่คูณกันได้ซึ่งก็คือจำนวนหลักของเมทริกซ์ตัวแรกจะต้องเท่ากับจำนวนแถวของเมทริกซ์ตัวหลัง ผลการคูณของเมทริกซ์จะได้เป็นเมทริกซ์ที่มีจำนวนแถวเท่ากับจำนวนแถวของเมทริกซ์ตัวแรกและมีจำนวนหลักเท่ากับจำนวนหลักของเมทริกซ์ตัวหลังโดยสมาชิกตัวที่ i, j ของเมทริกซ์ผลลัพธ์จะได้มากจากผลรวมของผลคูณของทุกสมาชิกของแถว i ขอเมทริกซ์ตัวแรกกับทุกสมาชิกของหลัก j ของเมทริกซ์ตัวหลังดังภาพที่ 3.13

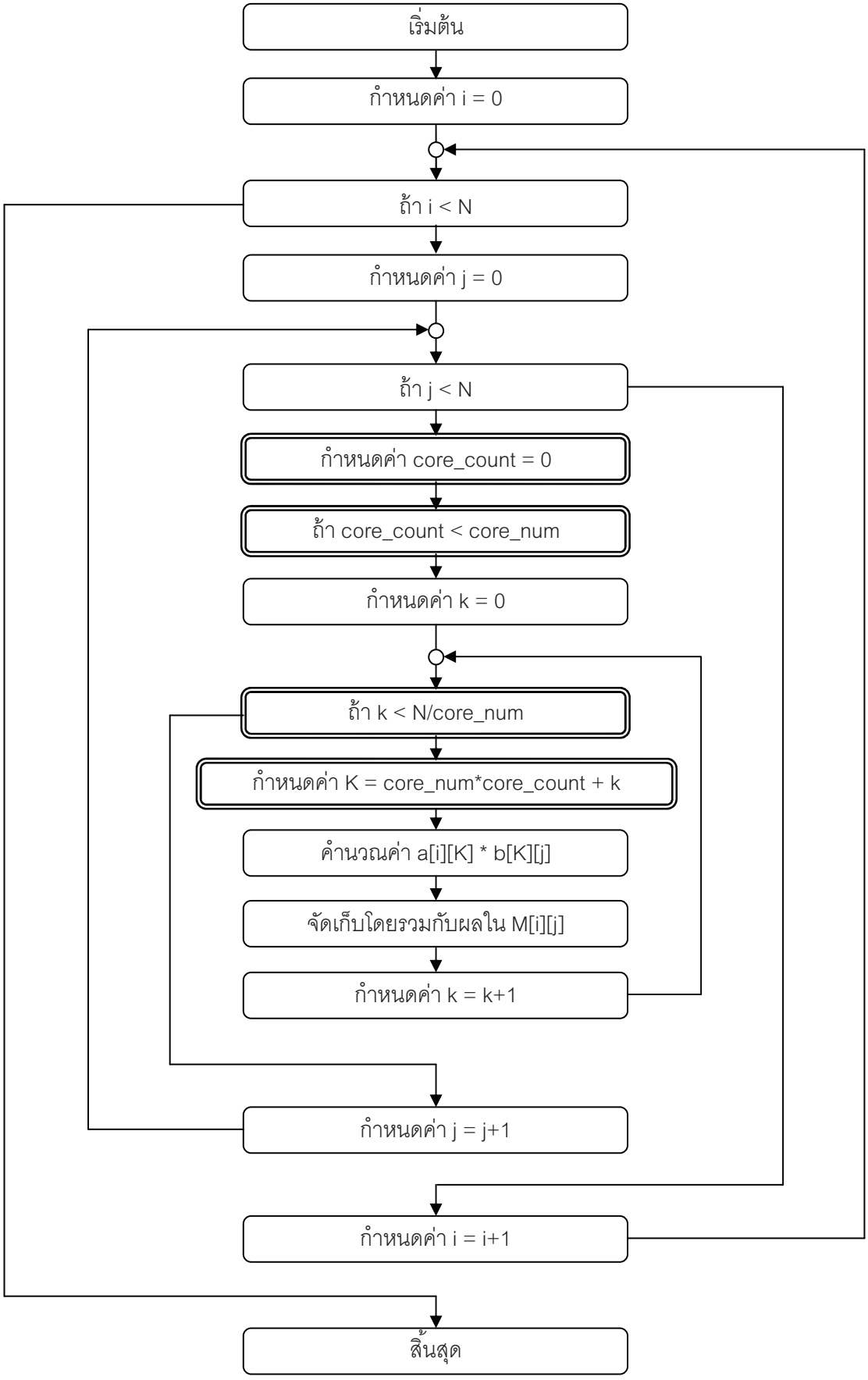
```
for (i=0; i<N; ++i)
  for (j=0; j<N; ++j)
  {
    M[i][j] = 0;
     for (k=0; k<N; ++k)
       M[i][j] += A[i][k] + B[k][j];
  }
```

ภาพที่ 3.13 แสดงโปรแกรมการคูณเมทริกซ์

ภาพที่ 3.14 แสดงโฟล์วชาร์ตการทำงานของโปรแกรมการคูณเมทริกซ์ โปรแกรมการคูณเมทริกซ์จะเริ่มต้นด้วยการกำหนดค่าให้กับตัวแปร i ที่ใช้เป็นตัวนับตัวแปรนี้จะเป็นตัวที่ใช้ในการนับค่าเท่ากับจำนวนแถวของเมทริกซ์ตัวแรก ต่อมาจะเป็นการตรวจสอบเงื่อนไข i < N เพื่อใช้ตรวจจำนวนรอบซึ่งจะสิ้นสุดการทำงานเมื่อเงื่อนไขไม่เป็นจริง ต่อมาในลักษณะเดียวกันจะเป็นการกำหนดค่าให้กับตัวแปร j ที่ใช้เป็นตัวนับโดยจะนับค่าเท่ากับจำนวนหลักของเมทริกซ์ตัวท้ายซึ่งจะวนทำงานเมื่อเงื่อนไข j < N เป็นจริง และรอบในสุดจะใช้ตัวแปร k ที่ใช้ในการนับเท่ากับจำนวนหลักของเมทริกซ์ตัวแรกหรือจำนวนแถวของเมทริกซ์ตัวหลังที่ต้องมีขนาดที่เท่ากัน ภายในสุดจะเป็นการบวกผลคูณระหว่างสมาชิกแถวที่ i หลักที่ k ของเมทริกซ์ตัวแรกและสมาชิกแถวที่ k หลักที่ j ของเมทริกซ์ตัวหลัง เมื่อการทำงานเสร็จสิ้นก็จะได้ผลคูณของเมทริกซ์ A และ B เป็นเมทริกซ์ M สำหรับโปรแกรมการคูณเมทริกซ์แบบขนานในภาพที่ 3.15 จะมีลักษณะใกล้เคียงกับโปรแกรมแบบปรกติแต่หลังจากการตรวจสอบเงื่อนไขของตัวแปร j จะเป็นการสร้างลูปการทำงานอีกชั้นหนึ่งที่จะเป็นส่วนการแบ่งการทำงานให้กับแต่ละหน่วยประมวลผลย่อยที่มีทั้งหมดเท่ากับค่าของตัวแปร core_num และใช้ตัวแปรที่ชื่อ core_count เป็นตัวนับ ถัดไปจะเป็นการกำหนดค่าเริ่มต้นของตัวแปร k และเงื่อนไขการทำงานลูปซึ่งจะมีความแตกต่างที่จำนวนรอบโดยจะถูกหารด้วยจำนวนหน่วยประมวลผลย่อยที่แบ่งงานกัน ซึ่งก็จะทำงานในลักษณะเดียวกันกับโปรแกรมแบบปกติแต่การทำงานจริงจะทำงานแบบขนานกันในทุก ๆ หน่วยประมวลผลย่อย



ภาพที่ 3.14 โฟล์วชาร์ตการทำงานของโปรแกรมการคูณเมทริกซ์



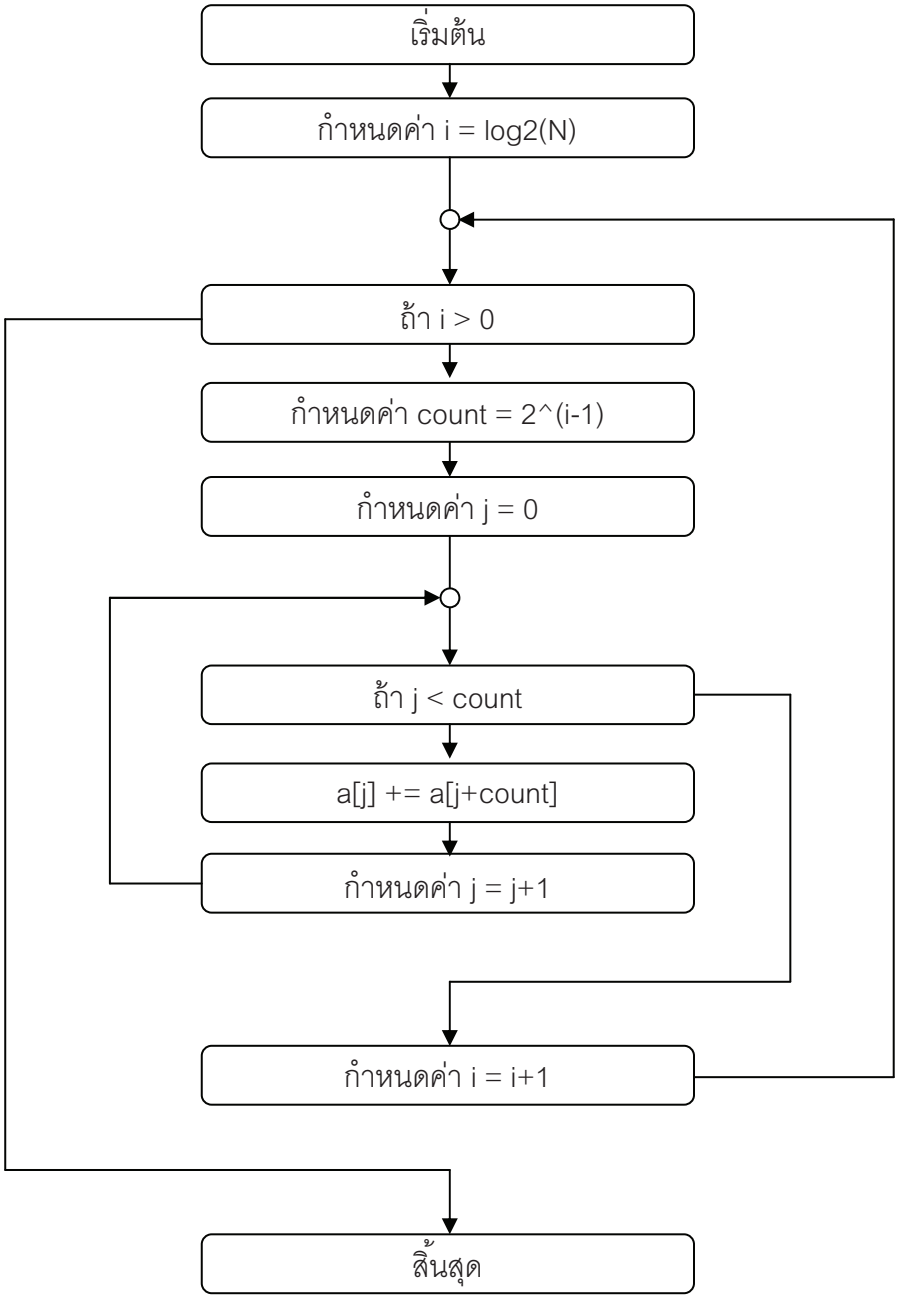
ภาพที่ 3.15 โฟล์วชาร์ตการทำงานของโปรแกรมการคูณเมทริกซ์สำหรับการทำงานแบบขนาน (ส่วนกรอบสองชั้นคือส่วนที่เพิ่มสำหรับการทำงานแบบขนาน)

3.6 การบวกแบบรีดักชัน

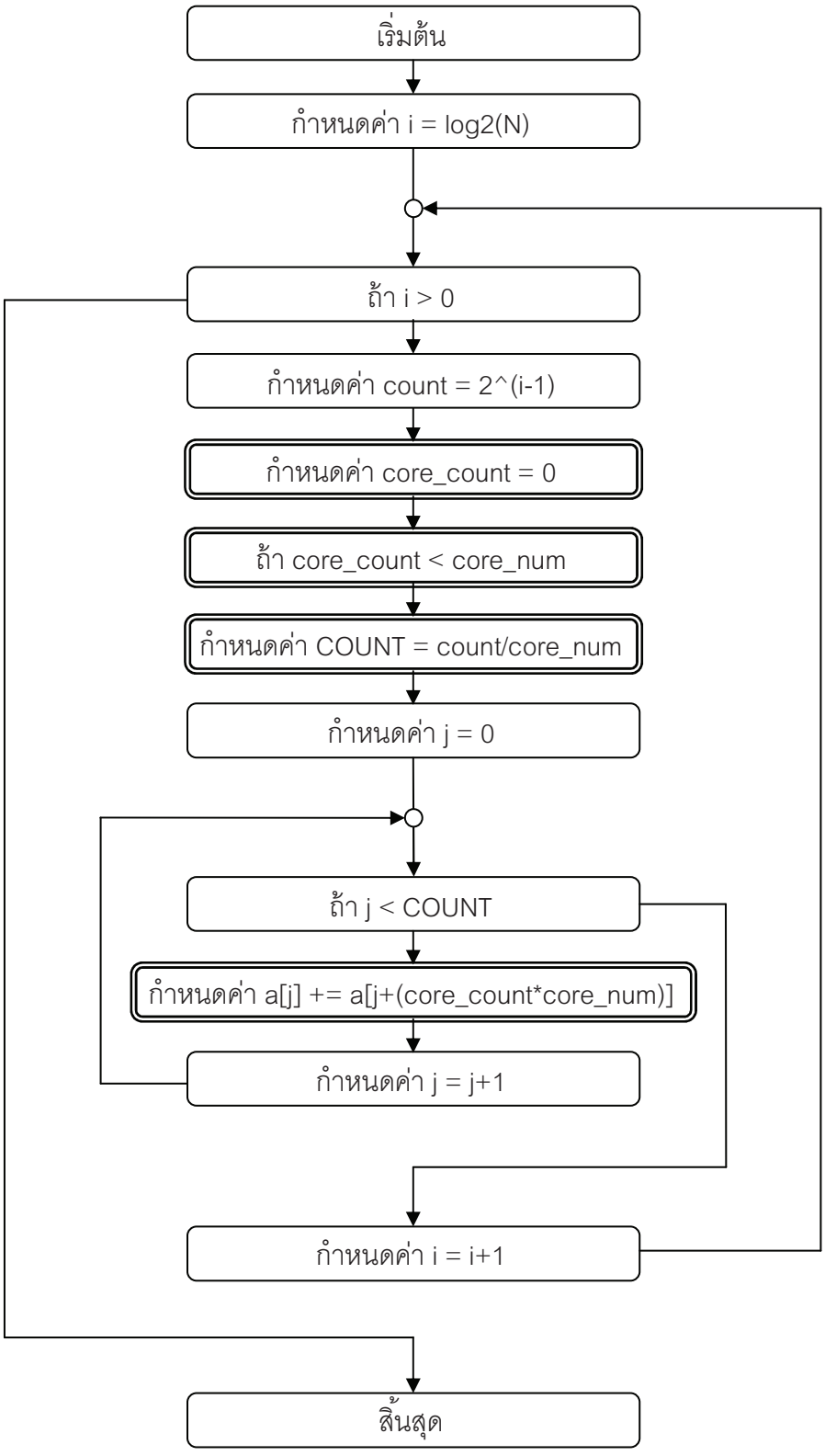
การซัมหรือการหาผลรวมจะสามารถพบได้ทั่วไปในการเขียนโปรแกรม เช่น การหาผลรวมเพื่อใช้ในการหาค่าเฉลี่ย การหาผลรวมของตัวเลขทางบัญชี การหาผลรวมของคะแนนที่ได้ในการทำข้อสอบเป็นต้น การเขียนโปรแกรมในการหาผลรวมโดยทั่วไปจะทำการสร้างลูปเพื่อวนบวกค่าของตัวแปรทุกตัวครั้งละหนึ่งตัวจนครบ ซึ่งก็จะใช้เวลาบวกทั้งหมด n รอบ แต่สำหรับในกรณีที่มีหน่วยประมวลผลที่สามารถทำงานได้พร้อม ๆ กันในช่วงเวลาหนึ่ง ๆ หรือทำงานในแบบขนาน ก็จะสามารถใช้ประโยชน์ได้อย่างมากหากใช้วิธีการบวกแบบรีดักชัน การบวกแบบรีดักชันนั้นในแต่ละรอบจะแบ่งข้อมูลทั้งหมดออกเป็นคู่ ๆ แล้วทำการบวกเพื่อหาผลรวมของคู่นั้น ๆ ซึ่งถ้าหากสมมติว่ามีข้อมูลอยู่ 8 ข้อมูลเมื่อหาผลรวมแบบแบ่งคู่แล้วก็จะเปรียบเสมือนว่ามีข้อมูลเหลืออยู่ 4 ข้อมูลที่ยังไม่ได้หาผลรวม ซึ่งถ้าทำซ้ำอีกสองรอบ ก็จะได้ผลลัพธ์ที่เป็นผลรวมของข้อมูลทั้งหมด

ในการหาผลบวกแบบการบวกแบบรีดักชันจะใช้การลูปสองชั้นดังภาพที่ 3.16 โดยจะกำหนดค่าเริ่มต้นตัวแปร i ให้เป็น log2(N) ซึ่งจะนำไปใช้ในการคำนวณในลูปถัดไป ลูปในชั้นนอกสุดนี้จะวนรอบตามจำนวนชั้นทั้งหมดของต้นไม้ที่สามารถครอบคลุมจำนวนของค่าทั้งหมดที่ต้องการหาผลรวมซึ่งก็คือ log2(N) โดยการวนรอบจะเรียงจากค่ามากไปค่าน้อยเพื่อนำไปใช้ในการคำนวนหาจำนวนรอบทั้งหมดที่จะวนในแต่ละชั้นของต้นไม้ของลูปชั้นในที่เริ่มจากการกำหนดค่าเริ่มต้นตัวแปร j เป็นศูนย์ กำหนดให้ตัวแปร count เท่ากับ 2^(i-1) และจะทำงานภายในลูปก็ต่อเมื่อเงื่อนไข j < count เป็นจริง ซึ่งจะทำการหาผลบวกระหว่างสองจำนวนที่ห่างกันอยู่ 2^(i-1) ภายในอาร์เรย์ ซึ่งเมื่อทำงานเสร็จสิ้นจะทำให้ได้ค่าผลรวมของจำนวนทั้งหมดอยู่ที่ตำแหน่งแรกของตัวแปรอาร์เรย์ที่ใช้ในการบวกนั้นเอง ซึ่งกระบวนการดังกล่าวนี้เป็นการทำงานแบบปกติแต่สำหรับในการทำงานแบบขนานดังภาพที่ 3.17 จะมีความคล้ายคลึงกันโดยจะเพิ่มลูปการทำงานอีกชั้นหนึ่งโดยในชั้นนี้จะเป็นการแบ่งงานแต่ละส่วนให้กับแต่ละหน่วยประมวลผล เริ่มจากการกำหนดค่าเริ่มต้นให้กับตัวแปรที่ใช้นับ core_count และตรวจสอบเงื่อนไข core_count <

core_num เพื่อทำงานภายในที่จะเป็นการกระจายการบวกให้กับแต่ละหน่วยประมวลผลเพื่อหาผลรวม



ภาพที่ 3.16 โฟล์วชาร์ตการทำงานของโปรแกรมการบวกแบบรีดักชัน



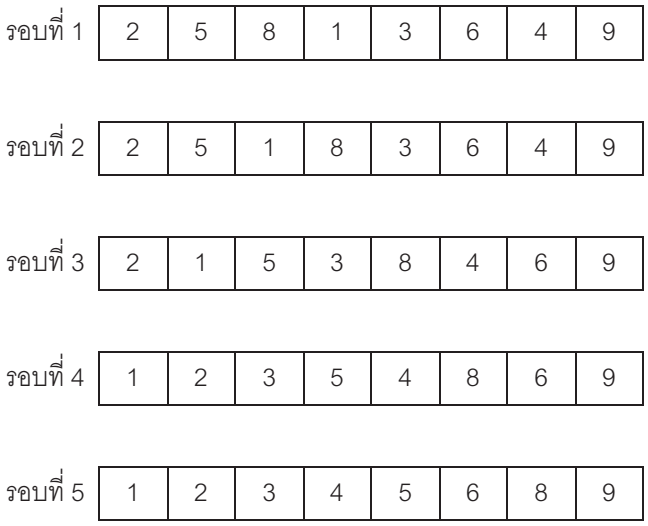
ภาพที่ 3.17 โฟล์วชาร์ตการทำงานของโปรแกรมการบวกแบบรีดักชันสำหรับการทำงานแบบขนาน (ส่วนกรอบสองชั้นคือส่วนที่เพิ่มสำหรับการทำงานแบบขนาน)

3.7 การจัดเรียงแบบคี่-คู่

การจัดเรียงเป็นการเรียงข้อมูลให้อยู่ในลำดับที่ต้องการ เช่น การเรียงจากมากไปหาน้อย หรือการเรียงจากน้อยไปหามาก เป็นต้น อัลกอริทึมการจัดเรียงมีอยู่หลายอัลกอริทึม ตัวอย่างเช่น การเรียงแบบฟอง อินเสิร์ทชันซอร์ท ซีเลคชันซอร์ท เมอร์จซอร์ท และควิกซอร์ท ซึ่งในแต่ละอัลกอริทึมก็จะมีวิธีที่แตกต่างกันออกไป ซึ่งทำให้มีความแตกต่างกันในเรื่องของทรัพยากรที่ใช้ เช่น หน่วยความจำ และเวลาที่ใช้ในการทำงาน ซึ่งจะเห็นได้ว่าการเรียงแบบฟองที่ใช้วิธีการวนเปรียบเทียบทุกคู่ข้อมูลในทุก ๆ รอบทำให้ต้องใช้เวลาค่อนข้างมากเป็น O(n^2) ส่วนควิกซอร์ทที่ใช้หลักการการแบ่งแยกและเอาชนะทำให้ใช้เวลาที่น้อยกว่าเป็น O(n log2 n) สำหรับการจัดเรียงแบบคี่-คู่[9] นั้นจะเป็นอัลกอริทึมการจัดเรียงที่ทำงานได้ดีกว่าเมื่อทำงานบนหน่วยประมวลผลที่สามารถทำคำสั่งหลาย ๆ คำสั่งได้พร้อม ๆ กัน เนื่องจากอัลกอริทึมนี้ที่จะเปรียบเทียบข้อมูลที่อยู่ใกล้กันทุกข้อมูลโดยสลับกันเริ่มจากข้อมูลตัวคี่-คู่ในแต่ละรอบของการทำงาน ทำให้สามารถทำงานได้เร็วโดยถ้าหากมีจำนวนหน่วยประมวลผลมากพอก็จะทำงานได้เร็วถึง O(n) เนื่องจากสามารถแจกงานในการเปรียบเทียบข้อมูลทุก ๆ คู่ให้กับแต่ละหน่วยประมวลผลในการทำงานแต่ละรอบจึงทำให้ทำงานได้อย่างรวดเร็ว

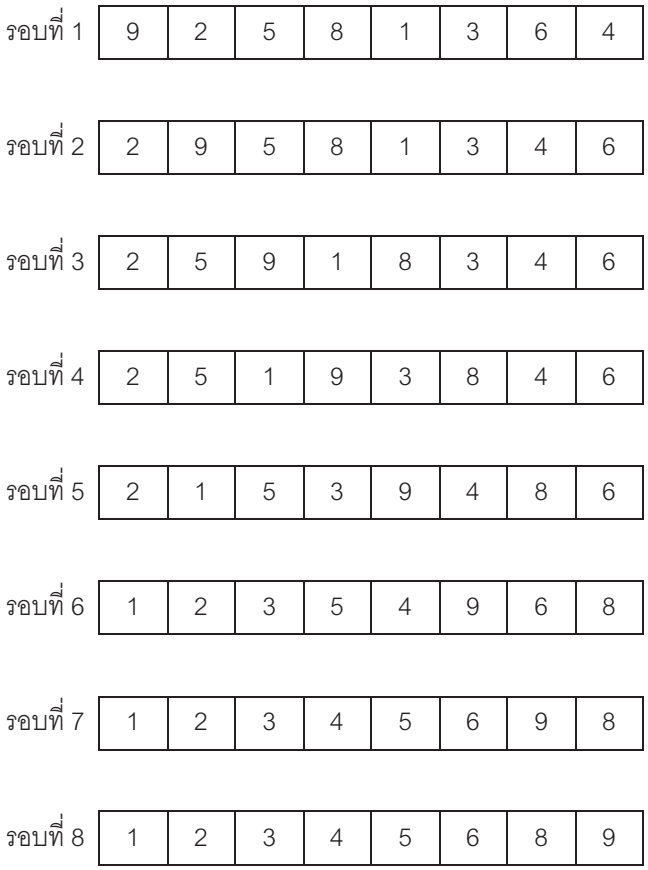
ภาพที่ 3.18 เป็นภาพแสดงตัวอย่างการทำงานของการเรียงแบบคี่-คู่ ในขั้นตอนแรกตัวเลขเรียงกันดังนี้ 2 5 8 1 3 6 4 9 ซึ่งในขั้นตอนแรกนี้จะเกิดการเปรียบเทียบค่าของคู่ข้อมูลที่ติดกันโดยเริ่มจากตัวแรกคู่กับตัวที่สอง ตัวที่สามคู่กับตัวที่สี่ ตัวที่ห้าคู่กับตัวที่หกและตัวที่เจ็ดคู่กับตัวที่แปด ซึ่งจะเกิดการสลับในแต่ละคู่เมื่อค่าที่อยู่ในตำแหน่งแรกมากกว่าค่าที่อยู่ในตำแหน่งหลัง ซึ่งจะเกิดการสลับค่า 8 กับ 1 จะได้เป็น 2 5 1 8 3 6 4 9 ในรอบต่อมาจะจับคู่กันในลักษณะเดิมแต่เริ่มจากข้อมูลในตำแหน่งที่สอง คือ ตัวที่สองคู่กับตัวที่สาม ตัวที่สี่คู่กับตัวที่ห้าและตัวที่หกคู่กับตัวที่เจ็ด ซึ่งจะเกิดการสลับของทุกคู่เนื่องจากค่าในตำแหน่งแรกของแต่ละคู่มากกว่าค่าในตำแหน่งหลังจะได้เป็น 2 1 5 3 8 4 6 9 สำหรับรอบที่สามนี้จะจับคู่คล้ายกับในรอบแรกซึ่งก็คือ ตัวแรกคู่กับตัวที่สอง ตัวที่สามคู่กับตัวที่สี่ ตัวที่ห้าคู่กับตัวที่หกและตัวที่เจ็ดคู่กับตัวที่แปด จะเกิดการ

สลับค่าของสามคู่แรกจะได้เป็น 1 2 3 5 4 8 6 9 และในรอบต่อมาจะคล้ายกับในรอบที่สองซึ่งจะเกิดการสลับในสองคู่หลังได้เป็น 1 2 3 4 5 6 8 9 ซึ่งเรียงลำดับจากน้อยไปมากเป็นที่เรียบร้อย



ภาพที่ 3.18 แสดงขั้นตอนการเรียงแบบคี่-คู่แบบง่าย

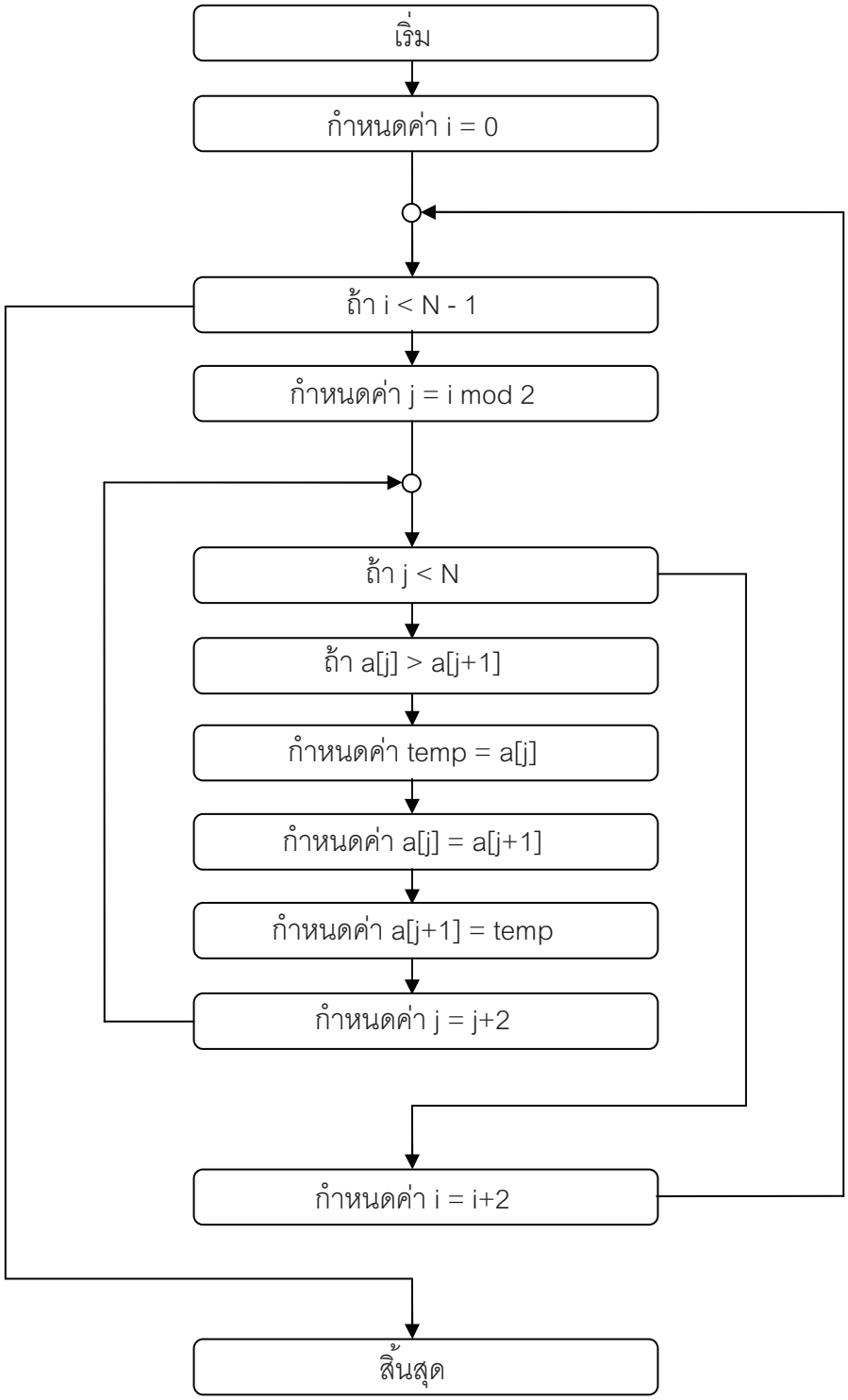
ตัวอย่างในภาพที่ 3.19 นี้จะเป็นการเรียงแบบคี่-คู่ที่ใช้การเรียงทั้งหมด N-1 ครั้งจึงจะสามารถเรียงได้เสร็จสิ้น โดย N คือจำนวนข้อมูลทั้งหมดที่นำมาเรียง เนื่องจากค่าในตำแหน่งแรกสุดซึ่งก็คือเป็นค่าที่มากที่สุดจะต้องใช้การสลับทั้งหมด N-1 ครั้งเพื่อให้อยู่ในลำดับท้ายสุดนั่นเอ



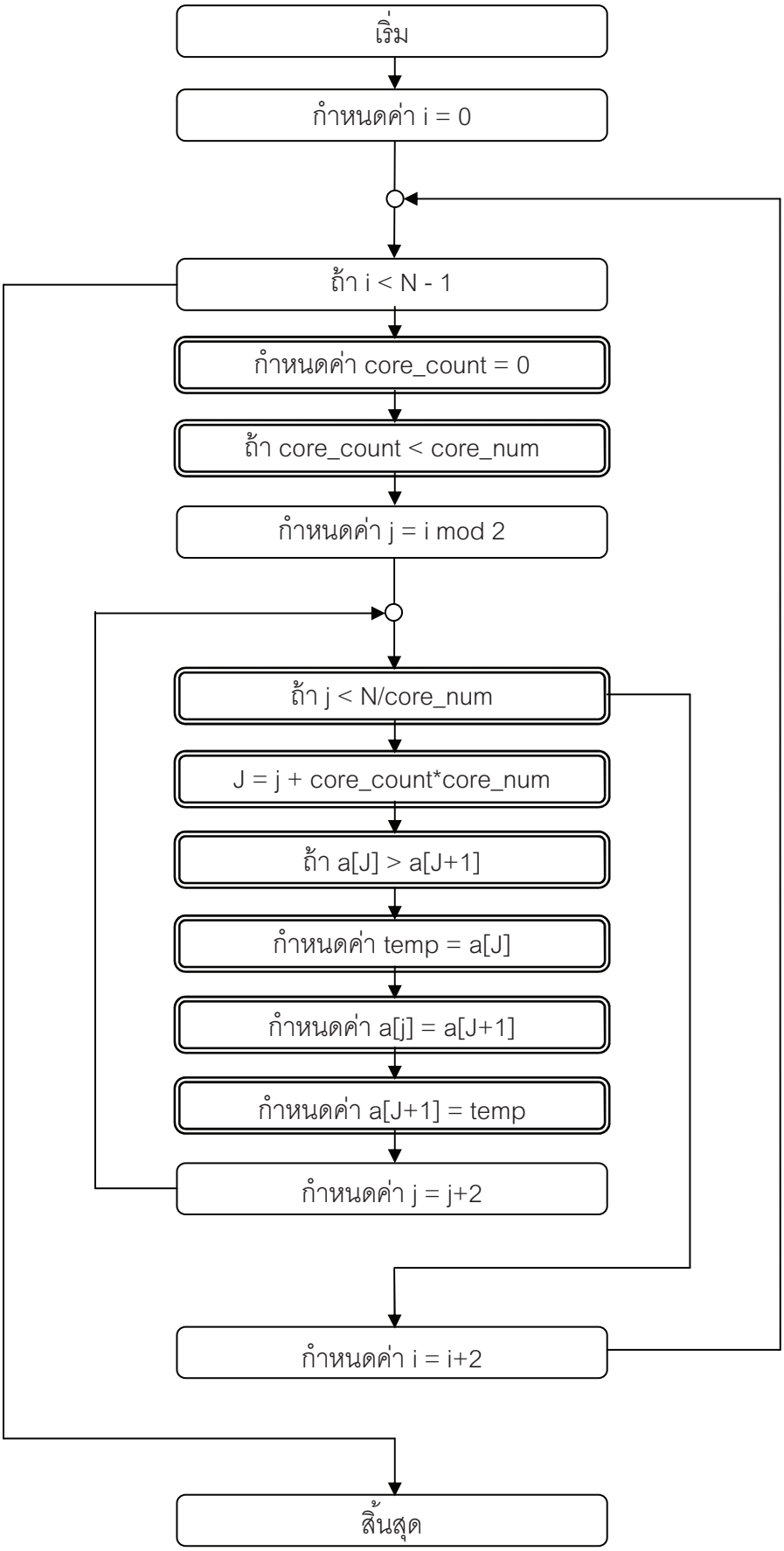
ภาพที่ 3.19 แสดงขั้นตอนการเรียงแบบคี่-คู่แบบจำนวนรอบสูงสุด

การเรียงแบบคี่-คู่ดังภาพที่ 3.20 จะเริ่มต้นด้วยการกำหนดค่าเริ่มต้นให้กับตัวแปร i และตรวจสอบเงื่อนไข i < N-1 เพื่อทำงานภายในเพียง N-1 รอบ ซึ่งภายในลูปจะเริ่มจากการกำหนดค่าา j เป็น i mod 2 เพื่อหาค่าตัวเริ่มต้นในแต่ละรอบว่าจะเป็นตัวคู่หรือตัวคี่แล้วจึงทำการตรวจสอบเงื่อนไข j < N เพื่อจำกัดจำนวนรอบการทำงานสำหรับปัญหาขนาด N โดยภายในจะเป็นการตรวจสอบค่าของอาร์เรย์ที่อยู่ติดกัน a[j] > a[j+1] ซึ่งถ้าหากค่าของอาร์เรย์ตำแหน่งที่น้อยกว่ามีค่ามากกว่าค่าของอาร์เรย์ที่อยู่ในตำแหน่งที่มากกว่าก็จะเกิดการสลับค่าโดยจะกำหนดค่าในอาร์เรย์ a[j] ให้กับตัวแปร temp แล้วกำหนดค่า a[j] ให้เท่ากับค่าของอาร์เรย์ a[j+1] แล้วจึงกำหนดค่าของอาร์เรย์ a[j+1] ให้เท่ากับค่าของตัวแปร temp และทำการเพิ่มค่าให้กับตัวแปรตัวนับ j = j+2 สำหรับลูปชั้นใน i = i+2 สำหรับตัวแปรชั้นนอกตามลำดับ ในส่วนของการทำงาน

แบบขนานดังภาพที่ 3.21 จะเพิ่มการทำงานของการแบ่งกระจายงานให้แต่ละหน่วยประมวลผลช่วยทำงาน โดยจะเพิ่มตัวแปร core_count และกำหนดค่าเริ่มต้นให้เป็น 0 ในกรณีที่ core_count < core_num ซึ่งเป็นการแบ่งงานให้ครบเท่ากับจำหน่วยหน่วยประมวลผล จะทดสอบค่า j < N/core_num และกำหนดค่า J=j+core_count*core_num เพื่อคำนวณหาค่าตัวชี้สำหรับแต่ละหน่วยประมวลผลที่ร่วมกันทำงาน



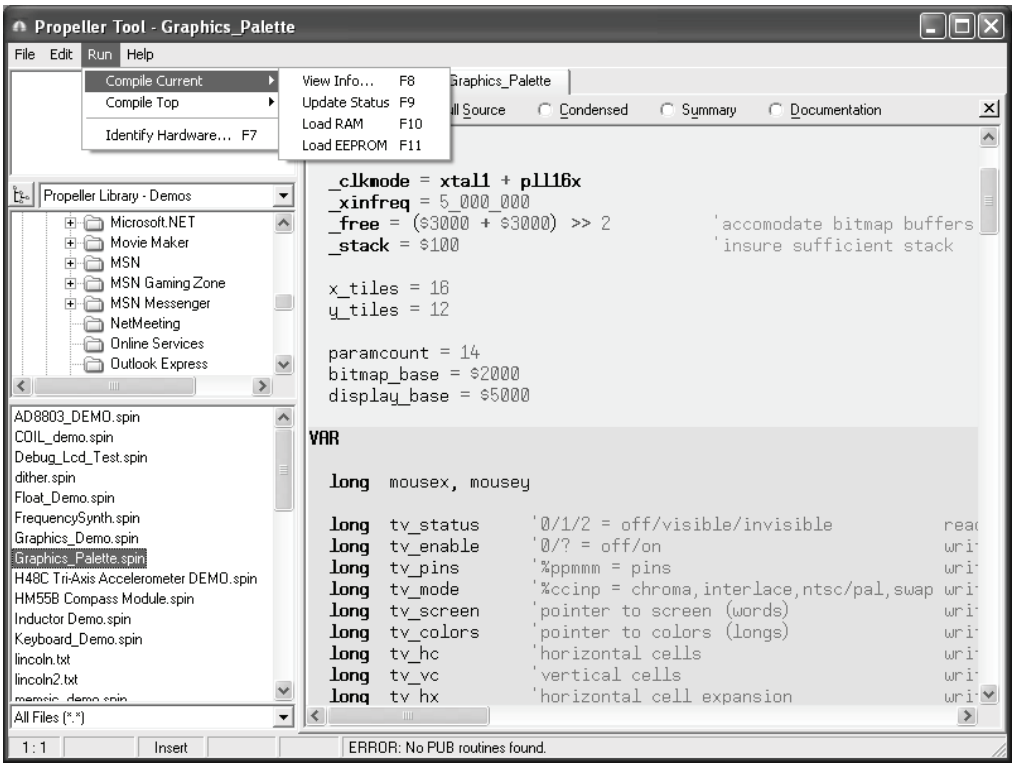
ภาพที่ 3.20 โฟล์วชาร์ตการทำงานของโปรแกรมการเรียงแบบคี่-คู่



ภาพที่ 3.21 โฟล์วชาร์ตการทำงานของโปรแกรมการเรียงแบบอ็อด-อีเว็นสำหรับการทำงานแบบขนาน (ส่วนกรอบสองชั้นคือส่วนที่เพิ่มสำหรับการทำงานแบบขนาน)

3.8 ขั้นตอนการทดลอง

การวัดประสิทธิผลของการแปลโปรแกรมสำหรับโปรแกรมตัวอย่างทั้งสามจะทำการโหลดข้อมูลจากเครื่องคอมพิวเตอร์โดยใช้โปรแกรม Propeller Tool v1.2 ที่สามารถดาวร์โหลดได้จากเวบไซต์ของพาราแล็กซ์ผู้พัฒนา ในการโหลดข้อมูลจะทำการเปิดไฟล์โปรแกรมแล้วใช้ฟังก์ชัน F8 เพื่อทำการแปลโปรแกรมและตรวจสอบสถานะการแปลโปรแกรมให้พร้อมก่อนการโหลดข้อมูลไปยังบอร์ดพร็อพเพลเลอร์ที่ใช้ในการทดลองและตรวจสอบสายส่งข้อมูลที่มีอินเทอร์เฟสแบบยูเอสบี แล้วจึงใช้ฟังก์ชัน F11 ดังภาพที่ 3.22 เพื่อโหลดโปรแกรมไปยังหน่วยความจำ EEPROM ภายในชิปพร็อพเพลเลอร์และเริ่มต้นการทำงาน



ภาพที่ 3.22 แสดงหน้าจอการทำงานในการทดลอง

บทที่ 4

ผลการวิเคราะห์ข้อมูล

4.1 ผลการแปลโปรแกรม

เมื่อนำโค้ดของโปรแกรมตัวอย่างการคูณเมทริกซ์ การบวกแบบรีดักชันและการเรียงแบบคี่-คู่ไปแปลโปรแกรมจะได้ผลเป็นดังภาพที่ 4.1 ถึง 4.9 ตามลำดับ ผลการแปลโปรแกรมเป็นผลที่ได้มากจากการนำโค้ดของโปรแกรมที่เขียนด้วยภาษา RZ พร้อมกับการใช้คำสั่งทำงานแบบขนานที่ใช้ช่วยในการเขียนโปรแกรมเพื่อให้การเขียนโปรแกรมง่ายขึ้น ซับซ้อนน้อยลงและใช้เวลาลดลง นำมาให้ตัวแปลโปรแกรมแปลเป็นโค้ดที่สามารถนำไปทำงานบนหน่วยประมวลผลแบบหลายแก่นได้

โปรแกรมการคูณเมทริกซ์ภาษา RZ แสดงในภาพที่ 4.1 ผลการแปลโปรแกรมแสดงดังภาพที่ 4.2 และฟังก์ชันการคำนวณในผลการแปลโปรแกรมโปรแกรมการคูณเมทริกซ์แสดงในภาพที่ 4.3 ตามลำดับ

```
for (i = 1; i <= N; i = i+1)
   for (j = 1; j <= N; j = j+1) {
   #pragma parallel for
      for (k = 1; k <= N; k = k+1)
         C[i*N+j] += A[i*N+k]*B[k*N+j];
   }
```

ภาพที่ 4.1 โปรแกรมการคูณเมทริกซ์ภาษา RZ

```
repeat i from 1 to N
  repeat j from 1 to N
    repeat k from 1 to ( N / core_count )
      repeat until busy__core == 0
      g_i := i
      g_j := j
      g_k := ( ( k - 1 ) * core_count )
      busy__data := 0
      repeat until busy__data == core_count
```

ภาพที่ 4.2 ผลการแปลโปรแกรมโปรแกรมการคูณเมทริกซ์

```
PUB f_3 | i, j, k

  repeat

    repeat until busy__data == 0

    repeat until not lockset ( l_id )

    busy__core += 1

    lockclr ( l_id )

    repeat until not lockset ( l_id2 )

    busy__data += 1

    lockclr ( l_id2 )

    i := g_i

    j := g_j

    k := g_k + cogid - 1

    M [(((i-1)*N)+(j-1))] +=
      A [(i-1)*N+(k-1)] * B [(k-1)*N+(j-1)]

    repeat until not lockset ( l_id )

    busy__core -= 1

    lockclr ( l_id )
```

ภาพที่ 4.3 ฟังก์ชันการคำนวณในผลการแปลโปรแกรมโปรแกรมการคูณเมทริกซ์

โปรแกรมการบวกแบบรีดักชันภาษา RZ แสดงในภาพที่ 4.4 ผลการแปลโปรแกรมแสดงดังภาพที่ 4.5 และฟังก์ชันการคำนวณในผลการแปลโปรแกรมโปรแกรมการบวกแบบรีดักชันแสดงในภาพที่ 4.6 ตามลำดับ

```
#pragma parallel for reduction
   for (i = 1; i <= N; i = i+1)
      sum = sum + V[i];
```

ภาพที่ 4.4 โปรแกรมการบวกแบบรีดักชันภาษา RZ

```
  repeat t_1 from _LOG2N to 1
    repeat t_2 from 1 to ( _POW [ t_1 - 1 ] /
__CORE ) + 1
        repeat until busy__core == 0
        g_a_1 := ( ( t_2 - 1 ) * __CORE )
        g_a_2 := _POW [ t_1 - 1 ]
        busy__data := 0
        repeat until busy__data == __CORE
```

ภาพที่ 4.5 ผลการแปลโปรแกรมการบวกแบบรีดักชัน

```
PUB f_1 | a_1, a_2
  repeat
    repeat until busy__data == 0
    repeat until not lockset ( l_id )
    busy__core += 1
    lockclr ( l_id )
    repeat until not lockset ( l_id2 )
    busy__data += 1
    lockclr ( l_id2 )
    a_1 := g_a_1 + cogid - 1
    a_2 := g_a_2
    if ( a_1 - 1 < a_2 )
      V [ a_1 - 1 ] += V [ ( a_1 - 1 ) + a_2 ]
    repeat until not lockset ( l_id )
    busy__core -= 1
    lockclr ( l_id )
```

ภาพที่ 4.6 ฟังก์ชันการคำนวณในผลการแปลโปรแกรมโปรแกรมการบวกแบบรีดักชัน

โปรแกรมการเรียงแบบคี่-คู่ภาษา RZ แสดงในภาพที่ 4.7 ผลการแปลโปรแกรมแสดงดังภาพที่ 4.8 และฟังก์ชันการคำนวณในผลการแปลโปรแกรมโปรแกรมการเรียงแบบคี่-คู่แสดงในภาพที่ 4.9 ตามลำดับ

```
for (i = 1; i <= N; i = i+1)
#pragma parallel for
   for (j = (i mod 2)+1; j < N; j = j+2)
      if (A[j] > A[j+1]) {
         temp = A[j]
         A[j] = A[j+1]
         A[j+1] = temp
      }
```

ภาพที่ 4.7 โปรแกรมการเรียงแบบคี่-คู่ภาษา RZ

```
   repeat t_0 from 1 to _N
     repeat t_1 from 1 to ( _N / ( 2 * _CORE ) )
+ 1
         repeat until busy__core == 0
         g_p_0 := ( ( t_0 // 2 ) + ( ( t_1 - 1 )
* ( 2 * _CORE ) ) )
         busy__data := 0
         repeat until busy__data == _CORE
```

ภาพที่ 4.8 ผลการแปลโปรแกรมโปรแกรมการเรียงแบบคี่-คู่

```
PUB f_1 | p_0, t_0
  repeat
    repeat until busy__data == 0
    repeat until not lockset ( l_id )
    busy__core += 1
    lockclr ( l_id )
    repeat until not lockset ( l_id2 )
    busy__data += 1
    lockclr ( l_id2 )
    p_0 := g_p_0 + ( 2 * ( ( cogid - 1 ) - 1 ) )

    if p_0 < _N - 1
      if D [ p_0 ] > D [ p_0 + 1 ]
        t_0 := D [ p_0 ]
        D [ p_0 ] := D [ p_0 + 1 ]
        D [ p_0 + 1 ] := t_0

    repeat until not lockset ( l_id )
    busy__core -= 1
    lockclr ( l_id )
```

ภาพที่ 4.9 ฟังก์ชันการคำนวณในผลการแปลโปรแกรมโปรแกรมการเรียงแบบคี่-คู่

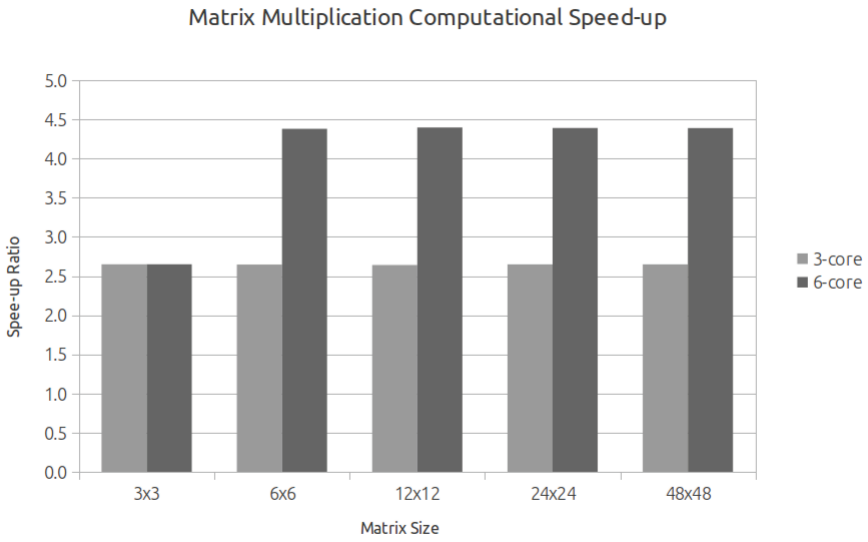
4.2 ผลการทำงานบนบอร์ดทดลอง

ตารางที่ 4.1 แสดงการอัตราส่วนเวลาที่ใช้ในการคูณเมทริกซ์ขนาดต่าง ๆ เปรียบเทียบกับเวลาที่ใช้ในการทำงานบนหน่วยประมวลผลเดี่ยวซึ่งก็คืออัตราความเร็วที่เพิ่มขึ้น (Speed-up) การทดลองจะทำการคูณเมทริกซ์ขนาดต่าง ๆ ดังนี้ เมทริกซ์ขนาด 3x3, เมทริกซ์ขนาด 6x6, เมทริกซ์ขนาด 12x12, เมทริกซ์ขนาด 24x24 และเมทริกซ์ขนาด 48x48 เมื่อนำโปรแกรมการคูณเมทริกซ์

ไปทำงานบนบอร์ดทดลองโดยใช้หน่วยประมวลผล 3 หน่วยจะได้อัตราส่วนของความเร็วที่เพิ่มขึ้นในการทำงานเป็นดังนี้ 2.650, 2.647, 2.641, 2.649 และ 2.649 ตามลำดับ และเมื่อนำโปรแกรมการคูณเมทริกซ์ไปทำงานบนบอร์ดทดลองโดยใช้หน่วยประมวลผล 6 หน่วยจะได้อัตราส่วนของความเร็วที่เพิ่มขึ้นในการทำงานเป็นดังนี้ 2.650, 4.381, 4.398, 4.391 และ 4.389 ตามลำดับ ภาพที่ 4.1 แสดงผลการทำงานในรูปกราฟแท่งเปรียบเทียบกันระหว่างการทำงานที่ใช้จำนวนหน่วยประมวลผลที่แตกต่างกันซึ่งจะเห็นได้ว่าการใช้จำนวนหน่วยประมวลผลที่มากกว่าจะยังคงมีอัตราความเร็วที่เพิ่มขึ้นใกล้เคียงกันสาเหตุเนื่องมาจากการกระจายของงานที่เมื่อกระจายมากขึ้นจะทำให้ปริมาณของงานที่แต่ละหน่วยประมวลผลต้องทำมีปริมาณที่ลดน้อยลงตามอัตราส่วนของจำนวนหน่วยประมวลผลที่ใช้ซึ่งปัญหาที่มีขนาดใหญ่ขึ้นจะสามารถทำงานได้เร็วยิ่งขึ้นแต่ยังคงมีอัตราส่วนที่ใกล้เคียงกันเนื่องจากการกระจายงานถูกจำกัดด้วยจำนวนหน่วยประมวลผลที่ใช้ในการประมวลผลนั่นเอง

ตารางที่ 4.1 ตารางแสดงเวลาเปรียบเทียบในการทำงานของโปรแกรมการคูณเมทริกซ์

NO. CORE USED	MATRIX SIZE				
	3x3	6x6	12x12	24x24	48x48
3-core	2.650	2.647	2.641	2.649	2.649
6-core	2.650	4.381	4.398	4.391	4.389

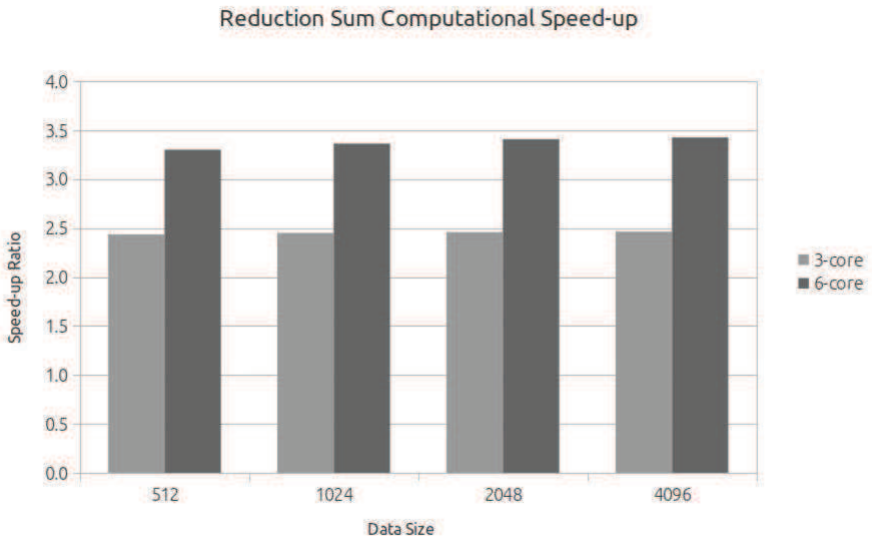


ภาพที่ 4.10 กราฟแสดงเวลาเปรียบเทียบในการคูณเมทริกซ์สำหรับเมทริกซ์ขนาดต่าง ๆ

ระยะเวลาที่ใช้ในการทำการการบวกแบบรีดักชันบนหน่วยประมวลผลหลายหน่วย คือ 3 และ 6 หน่วย เปรียบเทียบกับระยะเวลาที่ใช้ทำงานบนหน่วยประมวลผลเพียงหน่วยเดียวแสดงดังตารางที่ 4.2 ซึ่งได้จากการทดลองโดยการทำงานของโปรแกรมที่ปรับเปลี่ยนขนาดของปัญหาและจำนวนของหน่วยประมวลผลที่ใช้ในการทำงานแบบขนาน โดยขนาดของปัญหาหรือปริมาณข้อมูลที่ใช้ในการการบวกแบบรีดักชัน คือ 512, 1024, 2048 และ 4096 ข้อมูล อัตราความเร็วที่เพิ่มขึ้นในการทำงานบนหน่วยประมวลผลหลายหน่วยแบบ 3 หน่วย คือ 2.436, 2.451, 2.459 และ 2.464 และอัตราความเร็วที่เพิ่มขึ้นในการทำงานบนหน่วยประมวลผลหลายหน่วยแบบ 6 หน่วย คือ 3.302, 3.364, 3.408 และ 3.427 โดยแสดงผลเป็นรูปกราฟแท่งเปรียบเทียบการทำงานดังภาพที่ 4.2 ซึ่งจะเห็นว่าการใช้จำนวนหน่วยประมวลผลที่มากขึ้นจะทำให้มีการใช้เวลาในการทำงานที่ลดลง โดยที่ไม่ขึ้นกับขนาดของปัญหาที่มีอัตราความเร็วที่เพิ่มขึ้นใกล้เคียงกัน

ตารางที่ 4.2 ตารางแสดงเวลาเปรียบเทียบในการทำงานของโปรแกรมการบวกแบบรีดักชัน

NO. CORE USED	SIZE OF DATA			
	512	1024	2048	4096
3-core	2.436	2.451	2.459	2.464
6-core	3.302	3.364	3.408	3.427



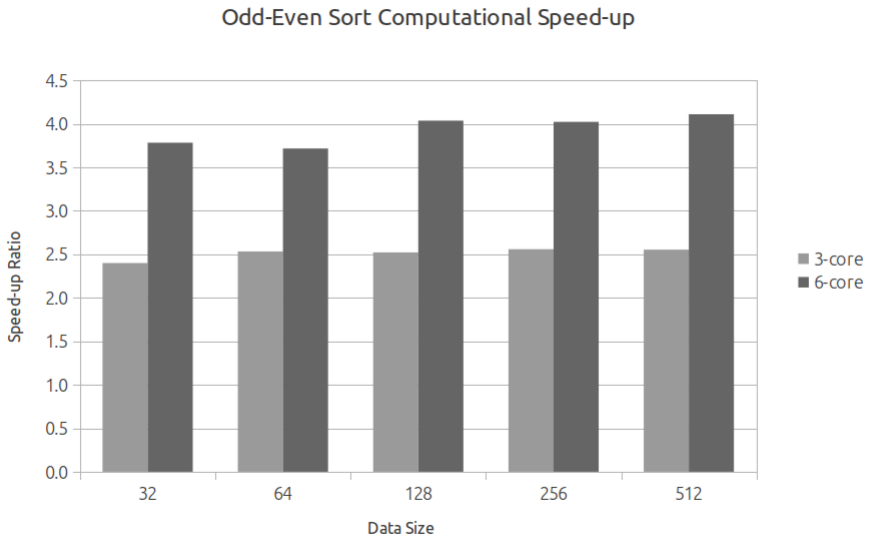
ภาพที่ 4.11 กราฟแสดงเวลาเปรียบเทียบโปรแกรมการบวกแบบรีดักชันสำหรับข้อมูลขนาดต่าง ๆ

ตารางแสดงการทำงานของโปรแกรมการเรียงแบบคี่-คู่ ดังตารางที่ 4.3 แสดงการทำงานสำหรับปัญหาขนาดต่าง ๆ จากขนาดเล็กไปขนาดใหญ่ คือ 32, 64, 128, 256 และ 512 โดยทำงานเปรียบเทียบกันระหว่างจำนวนหน่วยประมวลผลที่แตกต่างกันคือ 3 และ 6 หน่วยประมวลผล โดยระยะเวลาเปรียบเทียบกับหน่วยประมวลผลเดี่ยวจากผลการทดลองบนบอร์ดทดลองที่ทำงานบนหน่วยประมวลผล 3 หน่วย คือ 2.399, 2.532, 2.521, 2.558 และ 2.553 และสำหรับระยะเวลาเปรียบเทียบที่ใช้จากการทดลองบนบอร์ดทดลองที่ทำงานบนหน่วยประมวลผล

6 หน่วย คือ 3.783, 3.715, 4.036, 4.022 และ 4.110 ตามลำดับ เมื่อนำข้อมูลข้างต้นเพื่อสร้างเป็นกราฟแสดงการเปรียบเทียบจะได้เป็นกราฟในภาพที่ 4.3 ซึ่งจากภาพจะเห็นว่าอัตราความเร็วในการทำงานบนหน่วยประมวลผล 3 หน่วยเปรียบเทียบกับระยะเวลาการทำงานบนหน่วยประมวลผลเดียวน้อยกว่าระยะเวลาการทำงานบนหน่วยประมวลผล 6 หน่วยเปรียบเทียบกับระยะเวลาการทำงานบนหน่วยประมวลผลเดียวเนื่องจากการกระจายงานที่ทำให้การทำงานบนแต่ละหน่วยประมวลผลทำงานลดลงจึงใช้เวลาน้อยลงเมื่อกระจายงานไปยังหน่วยประมวลผลต่าง ๆ ได้มากขึ้นนั่นเอง

ตารางที่ 4.3 ตารางแสดงเวลาเปรียบเทียบในการทำงานของโปรแกรมการเรียงแบบคี่-คู่

NO. CORE USED	SIZE OF DATA				
	32	64	128	256	512
3-core	2.399	2.532	2.521	2.558	2.553
6-core	3.783	3.715	4.036	4.022	4.110



ภาพที่ 4.12 กราฟแสดงเวลาเปรียบเทียบโปรแกรมการเรียงแบบคี่-คู่สำหรับข้อมูลขนาดต่าง ๆ

บทที่ 5

สรุปผลการวิจัย อภิปรายผล และข้อเสนอแนะ

5.1 สรุปผลการวิจัย

โปรแกรมที่ทำงานอยู่บนหน่วยประมวลผลแบบเดี่ยวนั้น ณ เวลาใด ๆ จะสามารถทำคำสั่งได้เพียงคำสั่งเดียวเท่านั้น แต่สำหรับสถาปัตยกรรมในแบบหลายหน่วยประมวลผลจะมีความสามารถในการทำงานแบบขนานซึ่งสามารถทำคำสั่งได้หลายคำสั่ง ณ เวลาหนึ่ง ๆ ซึ่งสถาปัตยกรรมในลักษณะนี้ได้รับการพัฒนาและใช้งานอย่างแพร่หลายโดยเฉพาะคอมพิวเตอร์แบบตั้งโต๊ะที่ได้รับความนิยมอย่างสูงเพราะมีความเร็วและความสามารถรวมทั้งประสิทธิภาพที่สูงขึ้นแต่ไม่เพียงเท่านั้นไมโครคอนโทรลเลอร์ก็ได้มีการออกแบบและพัฒนาให้มีสถาปัตยกรรมแบบขนานเช่นเดียวกัน ไมโครคอนโทรลเลอร์พร็อพเพลเลอร์ได้รับการพัฒนาให้มีหน่วยประมวลผลทั้งสิ้นแปดหน่วยที่มีหน่วยความจำเฉพาะภายในและหน่วยความจำร่วมกันภายนอกทำให้สามารถกระจายการทำงานและร่วมกันทำงานระหว่างกันได้ ช่วยให้การทำงานมีประสิทธิภาพดียิ่งขึ้น

จากการนำโปรแกรมทั้งสามคือ โปรแกรมการคูณเมทริกซ์ โปรแกรมการบวกแบบรีดักชันและโปรแกรมการเรียงแบบคี่-คู่ที่เขียนด้วยภาษา RZ ที่ได้เพิ่มคำสั่งใหม่ที่รวมถึงคำสั่งการทำงานแบบขนานมาทำการแปลโปรแกรมด้วยตัวแปลโปรแกรมที่ได้รับการพัฒนาและปรับปรุงแล้วได้ผลเป็นดังนี้คือสามารถแปลโปรแกรมได้ทั้งสามโปรแกรมได้ผลการแปลโปรแกรมเป็นโค้ดที่อยู่ในภาษาสปินซึ่งเป็นภาษาที่ใช้ในการพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์พร็อพเพลเลอร์และโค้ดดังกล่าวนี้ได้รับการทดสอบจริงบนบอร์ดทดลอง โดยแต่และโปรแกรมเมื่อแปลโปรแกรมเสร็จเรียบร้อยจะนำโค้ดที่ได้โหลดไปยังบอร์ดทดลองเพื่อทำการทดลองและบันทึกผลการทดลองโดยมีการกำหนดรูปแบบการทดลองเพื่อตรวจวัดประสิทธิภาพ คือ การปรับขนาดของปัญหาและการปรับจำนวนของหน่วยประมวลผลที่ใช้ซึ่งผลการทดลองที่ได้เป็นไปตามความคาดหมายคือเมื่อนำ

โปรแกรมดังตัวอย่างมาทำงานแบบขนานแล้วจะบรรลุประสิทธิผลและได้ประสิทธิภาพที่ดีขึ้น เช่น เมื่อกระจายการทำงานให้กับแต่ละหน่วยประมวลผลก็จะทำให้การทำงานเสร็จสิ้นได้เร็วยิ่งขึ้น เมื่อเพิ่มจำนวนของหน่วยประมวลผลที่ใช้ในการทำงานก็จะทำให้การทำงานเสร็จได้เร็วเพิ่มยิ่งขึ้นอีกด้วย

5.2 อภิปรายผลการวิจัย

ตัวแปลโปรแกรมสามารถทำการแปลโปรแกรมโปรแกรมในภาษา RZ ที่เพิ่มเติมคำสั่งแบบขนานได้เป็นโค้ดที่อยู่ในภาษาสปินที่สามารถโหลดไปยังบอร์ดทดลองพร็อพเพลเลอร์และทำงานได้โดยการทำงานของโปรแกรมที่นำมาใช้ทดลองสามารถทำงานตามความคาดหมาย

5.2.1 การสร้างเทรดและตัวจัดการเทรด

เทรดเป็นส่วนหนึ่งของโปรแกรมที่ผู้เขียนโปรแกรมสร้างไว้ใช้งานโดยโปรแกรมที่โดยส่วนใหญ่เป็นโพรเสซหนึ่งของระบบปฏิบัติการสร้างเทรดขึ้นเพื่อให้เกิดการแบ่งและกระจายการทำงานให้แก่หน่วยประมวลผลที่มีอยู่หลายหน่วยที่อยู่บนระบบและสามารถเข้าถึงได้ โปรแกรมที่ทำงานอยู่บนเทรดเดียวหรือทำงานแบบอนุกรมจะมีข้อจำกัดในการทำงานตามขีดความสามารถของหน่วยประมวลผลที่ทำงานได้ครั้งละหนึ่งคำสั่ง ณ เวลาหนึ่ง ๆ แต่โปรแกรมที่สามารถทำงานโดยการสร้างเทรดหลาย ๆ เทรดเพื่อกระจายการทำงาน ช่วยกันทำงานบนหน่วยประมวลผลหลายหน่วยบนระบบก็จะสามารถทำให้งานกระจายกันไปทำบนแต่ละหน่วยประมวลผลพร้อม ๆ กันไป ซึ่งจะช่วยลดเวลาการทำงานจากที่มีเพียงหน่วยประมวลผลเดียวทำงานทั้งหมด

5.2.2 การเชื่อมต่อระหว่างไมโครคอนโทรลเลอร์หลายตัว

ในกรณีถ้าหากเราต้องการหน่วยประมวลผลจำนวนมากให้ช่วยร่วมกันทำงานหรือคำนวณงานใด ๆ เราก็อาจสามารถที่จะนำหน่วยประมวลผลหลาย ๆ หน่วยมาเชื่อมต่อกันเป็นจำนวนมากยิ่งขึ้นได้ การใช้หน่วยประมวลผลมาต่อเชื่อมกันให้สามารถทำงานร่วมกันเป็นการสร้างระบบกระจายการทำงานทำให้สามารถแบ่งงานให้กับหน่วยประมวลผลที่มีอยู่จำนวนมากให้ร่วมกันทำงานซึ่งมีโอกาสความเป็นไปได้สูงที่จะเกิดประสิทธิผลที่ทำให้การทำงานเร็วมากขึ้นมีประสิทธิภาพสูงขึ้น โดยการเชื่อมต่อระหว่างกันนั้นสามารถทำได้หลายรูปแบบ เช่น การเชื่อมต่อแบบสตาร์ (star) ที่มีตัวจัดการการเชื่อมต่อเพียงหน่วยเดียว การเชื่อมต่อแบบบัส (bus) ที่แต่ละหน่วยมีการจัดการการรับส่งข้อมูลเพื่อการทำงานและการเชื่อมต่อภายในเอง การเชื่อมต่อแบบสตาร์จะมีลักษณะการทำงานแบบรวมศูนย์คือจะมีตัวจัดการการเชื่อมต่อเพียงหน่วยเดียวซึ่งหน่วยนี้ก็จะมีหน้าที่หลักในการรับส่งข้อมูลซึ่งอาจเป็นหน่วยที่เก็บข้อมูลการทำงานของหน่วยต่าง ๆ ไว้เองหรือกระจายข้อมูลไปยังหน่วยที่ทำหน้าที่เฉพาะในการเก็บข้อมูลหรือกระจายให้เก็บข้อมูลไว้ในทุก ๆ หน่วยเองก็เป็นได้ สำหรับการเชื่อมต่อแบบบัสนั้นจะมีส่วนควบคุมการรับส่งข้อมูลและตัวจัดการการเชื่อมต่ออยู่ในทุกหน่วยที่อยู่ในระบบซึ่งโดยทั่วไปแล้วจะเกิดการส่งข้อมูลจากหน่วยที่ต้องการไปยังหน่วยปลายทางที่ต้องการส่งข้อมูลเฉพาะส่วนได้เลยไม่จำเป็นต้องผ่านหน่วยกลางที่ทำการเชื่อมระหว่างหน่วย ซึ่งวิธีแบบบัสนี้อาจไม่จำเป็นต้องมีหน่วยความจำร่วมอันเนื่องมาจากการที่แต่ละหน่วยสามารถส่งข้อมูลถึงกันได้แต่การทำงานในลักษณะนี้ต้องอาศัยการระบบซอฟท์แวร์ที่มีความซับซ้อนมากกว่า

5.2.3 การเชื่อมการทำงานกับเครื่องคอมพิวเตอร์ตั้งโต๊ะ

เนื่องจากไมโครคอนโทรลเลอร์จะมีส่วนอินเทอร์เฟส (interface) ที่สามารถเชื่อมต่อเข้ากับพอร์ตมาตรฐาน เช่น พอร์ตอนุกรม พอร์ตยูเอสบี เป็นต้นจึงสามารถที่จะเชื่อมต่อกันระหว่างคอมพิวเตอร์ตั้งโต๊ะกับไมโครคอนโทรลเลอร์ได้อย่างไม่ยากนัก ทำให้มีความน่าสนใจที่จะขยายความสามารถของไมโครคอนโทรลเลอร์แบบขนานนี้ โดยสามารถที่จะต่อไมโครคอนโทรลเลอร์

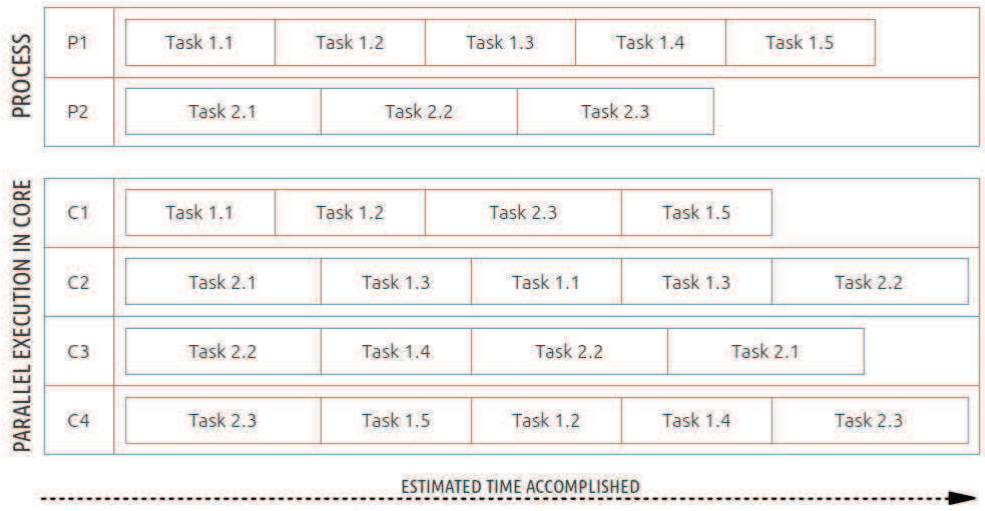
เพื่อให้ทำงานที่ได้รับมาจากคอมพิวเตอร์ตั้งโต๊ะแล้วทำงานแบบขนานบนไมโครคอนโทรลเลอร์หลาย ๆ ตัวในลักษณะที่ให้ช่วยกันทำงานซึ่งอาจจะเชื่อมต่อไมโครคอนโทรลเลอร์เป็นจำนวนมากเพื่อกระจายการทำงานทำให้งานสามารถเสร็จได้เร็วยิ่งขึ้น ซึ่งข้อดีสำหรับการต่อเชื่อมกับไมโครคอนโทรลเลอร์นี้คือมีความซับซ้อนที่น้อยกว่าระบบคอมพิวเตอร์ขนาดใหญ่ สามารถทำงานเฉพาะได้ง่ายกว่าคือโปรแกรมเฉพาะงานที่ต้องการไม่ต้องจัดการกับส่วนอื่น ๆ ที่มีส่วนเกี่ยวข้องน้อยหรือไม่มี ที่อาจพบได้กับการทำงานบนคลัสเตอร์ (cluster) และด้วยราคาที่ไม่แพงทำให้การขยายขนาดของระบบสามารถทำได้ด้วยต้นทุนที่ต่ำกว่าแต่สามารถสร้างระบบที่มีความสามารถในระดับสูงได้

5.3 ข้อเสนอแนะ

ความสามารถอีกประการหนึ่งของภาษาที่ช่วยในการทำงานแบบขนานคือการสร้างเทรด (Thread) หรือโปรแกรมย่อยภายในโพรเสซ (Process) ที่ถูกสร้างขึ้นมาเพื่อให้ช่วยการทำงานของโปรแกรมและทำงานในแบบขนาน โดยผู้เขียนโปรแกรมเป็นผู้กำหนดการทำงานภายในของเทรดเองทั้งหมดโดยให้ตัวแปลโปรแกรมและเอพีไอ (API : Application Programming Interface) จัดการการทำงานแบบขนานระหว่างเทรดทั้งหมดที่สร้างขึ้นมาให้เองอย่างอัตโนมัติ ความสามารถนี้เป็นที่น่าสนใจอย่างมากเนื่องจากสามารถนำไปใช้ประโยชน์ได้กว้างและหลากหลายมาก เนื่องจากผู้ใช้สามารถกำหนดการทำงานใด ๆ ได้เองภายในเทรดและสามารถสร้างได้ตามจำนวนที่ต้องการใช้ตามอัลกอริทึมที่ใช้งาน

ตัวอย่างการทำงานของเทรดแสดงในภาพที่ 5.1 ซึ่งประกอบไปด้วยสองโพรเสซ คือ โพรเสซ 1 และโพรเสซ 2 ที่มีห้าเทรดและสามเทรดตามลำดับ ทำงานอยู่บนหน่วยประมวลผลควอดคอร์หรือหน่วยประมวลผลสี่หน่วยดังภาพแสดงแต่ละหน่วยประมวลผลที่แต่ละเทรดทำงานอยู่ ณ เวลาหนึ่ง ๆ ซึ่งจากภาพจะเห็นว่าหน่วยประมวลผลที่หนึ่งจะมีเทรด 1.1, 1.2, 2.1, 2.3 และ 1.5

ทำงานอยู่ หน่วยประมวลผลที่สองจะมีเทรด 2.1, 1.3, 1.1, 1.3 และ 2.2 ทำงานอยู่ หน่วยประมวลผลที่สามจะมีเทรด 2.2, 1.4, 2.2 และ 2.1 ทำงานอยู่และหน่วยประมวลผลที่สี่จะมีเทรด 2.3, 1.5, 1.2, 1.4 และ 2.3 ทำงานอยู่ตามลำดับ ซึ่งจากตัวอย่างดังภาพนี้แสดงให้เห็นถึงการทำงานในแบบขนานซึ่งจะแตกต่างกับการทำงานแบบอนุกรมแบบปกติซึ่ง ณ เวลาหนึ่ง ๆ จะมีเพียงเทรดเดียวที่ทำงานได้เท่านั้นดังนั้นในทุก ๆ เทรดจะทำงานได้ก็ต่อเมื่อเทรดที่กำลังทำงานอยู่ได้เสร็จสิ้นการทำงานลงเท่านั้นแต่ในการทำงานแบบขนานจะสามารถให้แต่ละหน่วยประมวลผลช่วยกันทำงานไปพร้อม ๆ กันได้ โดยจากในภาพจะเห็นว่าเทรด 1.1, 2.1, 2.2 และ 2.3 ได้เริ่มทำงานและทำงานไปโดยพร้อม ๆ กัน



ภาพที่ 5.1 แสดงการทำงานของเทรดของสองโพรเสซบนหน่วยประมวลผลแบบหลายหน่วย

รายการอ้างอิง

[1] Parallax Inc. *Parallax Propeller* [Online]. 2012. Available from : http://www.parallax.com/tabid/407/Default [2012]

[2] Martin, J. and Lindsay, S. *Parallax Propeller Manual*. 2006.

[3] Daniel Hillis, W., Steele, Guy L., Jr.. Data parallel algorithms. *Communications of the ACM - Special issue on parallelism, IEEE Magazine* (Dec 1986).

[4] Chongstitvatana, P. *RZ language and its compiler* [Online]. 2012. Available from : http://www.cp.eng.chula.ac.th/faculty/pjw/project/rz3/index-rz3.htm [2012]

[5] Chongstitvatana, P. *RZ compiler tools kit* [Online]. 2012. Available from : http://www.cp.eng.chula.ac.th/faculty/pjw /project/rz/rz2compiler.htm [2012]

[6] Dagum, L. and Menon, R. OpenMP: An Industry Standard Api for Shared Memory Programming. *IEEE Computational Science and Engineering* (Mar 1998) : 46-55.

[7] Blumofe, R., Joerg, C., Kuszmaul, B., Leiserson, C., Randall, K., and Zhou, Y. Cilk: An efficient multithreaded runtime system. *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)* (July 1995) : 207-216.

[8] Popovici, N. and Willhalm, T. Putting Intel Threading Building Blocks to work. *Proc. of the International Workshop on Multicore Software Engineering (IWMSE 2008)* (2008).

[9] Grama, A., Karypis, G., Kumar, V., Gupta, A. *Introduction to parallel computing*. Addison-Wesley, 2003.

ประวัติผู้เขียนวิทยานิพนธ์

นายวัฒนา พรสูงส่ง เกิดเมื่อวันที่ 8 มกราคม พ.ศ. 2526 ที่กรุงเทพมหานคร เข้าศึกษาในระดับปริญญาบัณฑิตเมื่อปีการศึกษา 2544 และสำเร็จการศึกษาในหลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัยในปีการศึกษา 2547 และเข้าศึกษาต่อในระดับปริญญามหาบัณฑิตในปีการศึกษา 2552 หลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย