การพัฒนาการแปลโปรแกรมเชิงขนานสำหรับไมโครคอนโทรลเลอร์หลายแกน

นายนิกร มนัส

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2556

DEVELOPMENT OF A PARALLEL COMPILER FOR MULTI-CORE MICROCONTROLLER

Mr. Nikorn Manus

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Computer Science
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2013

หัวข้อวิทยานิพนธ์	การพัฒนาการแปลโปรแกรมเชิงขนานสำหรับไมโครคอนโทรลเลอร์หลายแกน
โดย	นายนิกร มนัส
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ศาสตราจารย์ ดร. ประภาส จงสถิตย์วัฒนา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(ศาสตราจารย์ ดร. บัณฑิต เอื้ออาภรณ์)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร. สุกรี สินธุภิญโญ)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(ศาสตราจารย์ ดร. ประภาส จงสถิตย์วัฒนา)

.....กรรมการภายนอกมหาวิทยาลัย

(รองศาสตราจารย์ ดร. วรเศรษฐ สุวรรณิก)

นิกร มนัส : การพัฒนาการแปลโปรแกรมเชิงขนานสำหรับไมโครคอนโทรลเลอร์หลายแกน. (DEVELOPMENT OF A PARALLEL COMPILER FOR MULTI-CORE MICROCONTROLLER) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ศ. ดร. ประภาส จงสถิตย์วัฒนา, 45 หน้า.

การเขียนโปรแกรมในปัจจุบันเป็นการเขียนและประมวลผลการทำงานบนแกนเดียว การพัฒนาหน่วยประมวลผลกลางให้มีหลายแกน เพื่อที่จะทำให้การประมวลผลมีประสิทธิภาพมากขึ้น ตัวแปลภาษาที่ใช้งานอยู่ไม่สนับสนุนการเขียนโปรแกรมให้กับหน่วยประมวลผลกลางชนิดหลายแกน งานวิจัยนี้สร้างตัวแปลภาษาเพื่อให้สามารถประมวลแบบขนานได้ โดยใช้ Parallax Propeller ซึ่งเป็นมัลติคอร์ไมโครคอนโทรลเลอร์ เนื่องจากมีถึงแปดโปรเซสเซอร์หรือแกน ซึ่งสามารถทำงานพร้อมๆกันหรือแยกกันทำงานอย่างอิสระ โดยใช้โครงสร้างภาษาแบบเดียวกันกับภาษา C มีเพิ่มสัญลักษณ์พิเศษ @ และ # เข้ามาเพื่อทำให้ผู้พัฒนาสามารถระบุควบคุมการแบ่งข้อมูลและระบุแกนของหน่วยประมวลผลได้

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา 2556

5371464921 : MAJOR COMPUTER SCIENCE
KEYWORDS: MULTICORE PROCESSOR / DATA PARALLELISM

NIKORN MANUS: DEVELOPMENT OF A PARALLEL COMPILER FOR MULTI-CORE MICROCONTROLLER. ADVISOR: PROF. PRABHAS CHONGSTITVATANA, Ph.D., 45 pp.

Presently, programming is done on single-core processors. The development of multicore processors has increased the performance. However, current compilers do not support programming for multicore processors. This research developed a compiler for parallel programming . The target machine is Palallax Propeller which is a multicore microcontroller that has eight cores. Each core works concurrently and independently. The proposed language has a similar structure to C language with additional special symbols @, # to allow programmers to control the division of data and specify the core to execute tasks.

Department: Computer Engineering Student's Signature .....
Field of Study: Computer Science Advisor's Signature .....
Academic Year: 2013

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดี ด้วยความช่วยเหลือของศาสตราจารย์ ดร. ประภาส จงสถิตวัฒนา อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งท่านได้ให้คำแนะนาและข้อคิดเห็นต่างๆ อันเป็นประโยชน์อย่างยิ่งในการทำวิจัย อีกทั้งยังช่วยแก้ปัญหาต่างๆ ที่เกิดขึ้นระหว่างการดาเนินงานอีกด้วย ขอกราบขอบพระคุณเป็นอย่างสูง

ขอกราบขอบพระคุณคณะกรรมการสอบวิทยานิพนธ์ทุกท่านเป็นอย่างสูง ได้แก่ผู้ช่วยศาสตราจารย์ ดร.สุกรี สินธุภิญโญ และ รองศาสตราจารย์ ดร. วรเศรษฐ สุวรรณิก ที่สละเวลาอันมีค่ามาชี้ให้เห็นถึงข้อบกพร่อง พร้อมทั้งให้ข้อคิดและคำแนะนำอันเป็นประโยชน์อย่างยิ่งต่องานวิจัย

ขอขอบคุณเจ้าหน้าที่ประจาภาควิชาวิศวกรรมคอมพิวเตอร์ทุกท่านที่มีส่วนช่วยเหลือทาให้วิทยานิพนธ์ฉบับนี้สำเร็จเรียบร้อยลงด้วยดีทุกประการ

สุดท้ายนี้ ผู้วิจัยขอขอบพระคุณบิดามารดา และครอบครัว ซึ่งเปิดโอกาสให้ได้รับการศึกษาเล่าเรียน ตลอดจนคอยช่วยเหลือและให้กาลังใจผู้วิจัยเสมอมาจนสำเร็จการศึกษา

สารบัญ

หน้า

หน้า

สารบัญภาพ

หน้า

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

Parallax Propeller [1] เป็นมัลติคอร์ไมโครคอนโทรลเลอร์แบบที่สามารถที่ประมวลผลแบบหลายแกนได้ดีตัวหนึ่ง เนื่องจากมีถึงแปดโปรเซสเซอร์หรือแกน (เรียกว่า Cogs) ซึ่งสามารถทำงานพร้อมๆกันหรือแยกกันทำงานอย่างอิสระก็สามารถกระทำได้ การใช้งาน Propeller นั้นจะเปรียบได้เป็นการจ้างทีมงานแปดคนทำงานในงานๆเดียวกัน และคนในทีมงานนั้นก็สามารถทำงานในแบบคู่ขนานกันไปตามงานที่ได้รับมอบหมายและจะประสานงานตามความจำเป็นเพื่อให้บรรลุเป้าหมายร่วมกัน ดังนั้นผลงานที่ได้จะถูกต้อง ยืดหยุ่นและมีประสิทธิภาพ

เนื่องจากการพัฒนาโปรแกรมให้สามารถใช้งาน Propeller ไมโครคอนโทรลเลอร์ให้ได้เต็มประสิทธิภาพตามที่ผู้ผลิตได้ออกแบบไว้นั้นผู้พัฒนาโปรแกรมจำเป็นต้องเรียนรู้ภาษาสปินซึ่งเป็นภาษาระดับสูงหรือแอสเซมบลี เพื่อใช้งานแต่ละ Cogs ให้ดีนั้นยังต้องศึกษาวิธีการโปรแกรมแบบขนาน เพื่อทำให้การทำงานร่วมกันของแต่ละ Cogs ได้อย่างเต็มประสิทธิภาพ งานวิจัยนี้ทำการสร้างคอมไพเลอร์แบบขนานด้วยภาษา RZ ซึ่งเป็นการใช้โครงสร้างภาษาแบบเดียวกันกับภาษา C ซึ่งเป็นภาษาระดับสูงที่เป็นที่นิยมกันมากในการพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์อื่นๆ ภาษา RZ นั้นจะเป็นภาษาที่ปรับรูปแบบไวยากรณ์ภาษาให้มีขนาดเล็กเหมาะกับการพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์ซึ่งมีหน่วยความจำไม่มากนัก หลักการทำงานเบื้องต้นของคอมไพเลอร์นั้นจะทำสร้างโปรแกรมแอสเซมบลีที่มีคำสั่งควบคุมแต่ละ Cogs โดยที่ไม่ต้องกังวลกับการเขียนคำสั่งเพื่อควบคุมแต่ละ Cogs ทำให้การพัฒนาโปรแกรมทำได้ง่ายขึ้น

การประมวลผลแบบขนานนั้นทำให้สามารถใช้งานแต่ละ Cogs ได้อย่างมีประสิทธิภาพ อัลกอริทึมหรือรูปแบบของการโปรแกรมแบบขนานที่นำมาประยุกต์ใช้ก็ต้องเหมาะสมกับสถาปัตยกรรมของ Propeller ดังนั้นแนวทางวิจัยนี้ใช้การพัฒนาโปรแกรมแบบขนานเชิงข้อมูลมาประยุกต์ เนื่องจากง่ายต่อการเรียนรู้ การประมวลผลแบบขนานนั้นจะเกิดในเฉพาะมิติของข้อมูลเท่านั้นคือคำสั่งชุดเดียวกันจะถูกประมวลผลโดยแบ่งข้อมูลออกเป็นหลายชุดๆในเวลาเดียวกัน โดยจะประมวลผลบนหลายหน่วยประมวลผล ซึ่งเหมาะสมกับสถาปัตยกรรมของ Propeller เพราะว่าแต่ละ Cogs นั้นจะมีหน่วยความจำ, I/O Control, Program Counter เป็นของตัวเอง ดังนั้นการประมวลจะเป็นอิสระต่อกัน ลดโอกาสในการเกิดปัญหาการจองหน่วยความจำซ้อนทับกันของแต่ละ Cogs ลงได้ งานวิจัยนี้มีขอบเขตการศึกษาเพียงการประยุกต์ใช้ Data Parallel Algorithm และสร้างคอมไพเลอร์แบบขนานเท่านั้นไม่ครอบคลุมถึงการประมวลแบบขนานแบบอื่นๆ

1.2 วัตถุประสงค์ของการวิจัย

การสร้างคอมไพเลอร์แบบขนานที่ประยุกต์ใช้หลักการของ Data Parallel Algorithm สามารถประมวลผลบน Propeller ไมโครคอนโทรลเลอร์ได้

1.3 ขอบเขตของการวิจัย

งานวิจัยนี้เป็นการสร้างคอมไพเลอร์แบบขนานที่ประยุกต์ใช้หลักการของ Data Parallel Algorithm เพื่อให้สามารถประมวลผลประเภทลูปเท่านั้น ซึ่งการสร้างคอมไพเลอร์แบบขนานโดยใช้การประมวลแบบขนานในรูปแบบอื่นๆ งานวิจัยนี้จะไม่ครอบคลุมถึง ซึ่งเป็นข้อจำกัดของงานวิจัยนี้ที่สามารถนำไปพัฒนาต่อไปได้

1.4 ประโยชน์ที่ได้รับ

ผลที่คาดว่าจะได้รับคือคอมไพเลอร์ที่สามารถประมวลผลแบบขนานได้ใช้กับ Propeller ไมโครคอนโทรลเลอร์

1.5 ลำดับการจัดเรียงเนื้อหาในวิทยานิพนธ์

วิทยานิพนธ์นี้แบ่งเนื้อหาออกเป็น 6 บทดังต่อไปนี้ บทที่ 1 เป็นบทนำซึ่งกล่าวถึง ความเป็นมาและความสำคัญของปัญหา รวมถึงวัตถุประสงค์ของการวิจัย บทที่ 2 กล่าวถึงทฤษฎีพื้นฐานและงานวิจัยที่เกี่ยวข้องในงานวิจัยนี้ บทที่ 3 กล่าวถึงระเบียบขั้นตอนวิธีที่เสนอ บทที่ 4 กล่าวถึงการทดลองและผลการทดลอง บทที่ 5 กล่าวถึงการประยุกต์ใช้ และบทที่ 6 กล่าวถึงสรุปผลการวิจัยและข้อเสนอแนะ

1.6 ผลงานที่ตีพิมพ์จากวิทยานิพนธ์

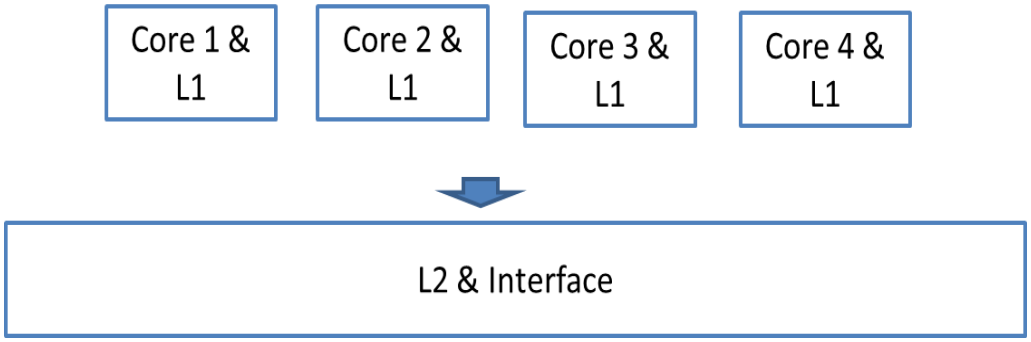
ส่วนหนึ่งของวิทยานิพนธ์นี้ได้รับการตีพิมพ์เป็นผลงานวิชาการในหัวข้อเรื่องดังต่อไปนี้ “Compiler on the Web: Remote Laboratory for e-Learning” โดย นิกร มนัส และประภาส จงสถิตย์วัฒนา ในงานประชุมวิชาการของ “The Fourth TCU International e-Learning Conference” ที่ประเทศไทย ระหว่างวันที่ 14-15 มิถุนายน 2555

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 ตัวประมวลผลหลายแกน (Multi-core Processors)

ในระบบคอมพิวเตอร์ในปัจจุบันนั้นการประมวลผลแบบหลายแกนเป็นที่แพร่หลายอย่างยิ่งไม่ว่าจะเป็นคอมพิวเตอร์ตั้งโต๊ะ คอมพิวเตอร์พกพาและเครื่องคอมพิวเตอร์แม่ข่าย ไมโครคอนโทรลเลอร์ก็สามารถประมวลแบบหลายแกนได้เหมือนกับระบบคอมพิวเตอร์ทั่วไป ซึ่งองประกอบภายในของหน่วยประมวลผลกลางนั้นจะมีหน่วยประมวลผลย่อย ที่เราเรียกว่าแกนมากกว่า 1 แกน แต่ละแกนมีหน่วยความจำหลักเป็นของตัวเอง เรียกว่าแคชระดับที่1 หรือ L1 (Cache L1) แต่ละแกนอาจจะมีการใช้หน่วยความจำร่วมกันเรียกว่าแคช L2 การเข้าถึงข้อมูลที่อยู่ภายในแคช L1 นั้นสามารถทำได้รวดเร็วกว่าการเข้าถึงแคช L2 หรือการเข้าถึงข้อมูลจากหน่วยความจำหลัก แต่การออกแบบโครงสร้างซีพียูนั้น จะต้องมีส่วนที่ทำงานร่วมกันได้ด้วย เพื่อที่จะทำให้สามารถประมวลผลร่วมกันได้ ซึ่งส่วนที่จำทำงานร่วมกันก็คือแคช L2 นั่นเอง



ภาพที่ 2.1 โครงสร้างโดยคร่าวของมัลติคอร์

1. **Cache Memory** แคช (cache) คือหน่วยความจำขนาดเล็กที่มีความเร็วสูงซึ่งเก็บข้อมูลหรือคำสั่งที่ถูกเรียกใช้หรือเรียกใช้บ่อยๆ ข้อมูลและคำสั่งที่เก็บอยู่ในแคชซึ่งทำงานโดยใช้ SRAM (STATIC RAM) จะถูกดึงไปใช้งานได้เร็วกว่าการดึงข้อมูลจากหน่วยความจำหลัก (MAIN MEMORY) ซึ่งใช้ DRAM (DYNAMIC RAM)หลายเท่าตัวโดยในแคชของซีพียูแบบมัลติคอร์นั้น ก็มีถึงสองระดับคือแคช L1 และแคช L2 ซึ่ง ซึ่งขนาดของแคชนั้นก็จะแตกต่างกันออกไปแล้วแต่รุ่นของซีพียูนั้นๆ

2. **Clock Speed** ใช้อธิบายความถึงสมรรถนะของซีพียู บอกถึงความเร็วในการประมวลผล ตัวอย่างเช่น ซีพียูที่มีความเร็วสัญญาณนาฬิกา(Clock Speed) 3.2 GHz. สามารถทำงาน

ได้เร็ววกว่า ซีพียูที่มีความเร็วสัญญาณนาฬิกา 2.8 GHz ความเร็วสัญญาณนาฬิกาเรียกอีกอย่างหนึ่งว่า Clock rate คือ ความเร็วที่ซีพียูประมวลผลคำสั่ง มีหน่วยวัดเป็นรอบต่อวินาทีหรือเฮิร์ท(Hz) เครื่องคอมพิวเตอร์ทุกเครื่อง ต้องมีระบบสร้างสัญญาณนาฬิกาภายใน เพื่อใช้เป็นสัญญาณอ้างอิง ในการจัดระเบียบการประมวลผลคำสั่ง และควบคุมการทำงานของอุปกรณ์ต่างๆ ให้สอดคล้องกันอีกด้วย

3. ในซีพียูแบบมัลติคอร์นั้น มิได้เน้นการพัฒนาทางด้านการเพิ่มความเร็วของสัญญาณนาฬิกามากมายนักเนื่องจากการเพิ่มความเร็วของสัญญาณนาฬิกา จะเป็นการเพิ่มความต้องการพลังงาน ซึ่งเป็นผลทำให้ความร้อนก็เพิ่มขึ้นด้วย

2.1.2 การโปรแกรมแบบขนาน (Parallel Programming)

โปรแกรมต่างๆในปัจจุบันนี้จะถูกพัฒนาและประมวลผลแบบลำดับ ซึ่งการประมวลผลนั้นจะประมวลผลบนหน่วยประมวลผลกลางแบบแกนเดียวและชุดคำสั่งจะประมวลต่อๆกันไปและตามลำดับที่กำหนดไว้ ซึ่งจะเห็นว่าหน่วยประมวลผลกลางตัวอื่นๆยังสามารถมาใช้งานได้ ในกรณีที่เครื่องคอมพิวเตอร์นั้นมีหน่วยประมวลผลหลายแกน การประมวลผลแบบขนาน [3] สามารถแก้ปัญหาให้สามารถทำงานบนหน่วยประมวลผลกลางหลายแกนได้พร้อมๆกัน หรือประมวลผลคำสั่งพร้อมๆกันบนหน่วยประมวลผลกลางแบบแกนเดียวได้

โดยงานถูกแบ่งออกเป็นส่วนย่อยๆ และงานนั้นๆต้องสามารถแก้ไขควบคู่กันไปได้ ซึ่งปัญหาจากแต่ละส่วนถูกส่งไปประมวลผลพร้อมๆกัน โดยที่ทรัพยากรที่ใช้ในการประมวลผลนั้น อาจเป็นได้ดังนี้

1. .เครื่องคอมพิวเตอร์ที่มีหน่วยประมวลผลกลางหลายตัว ซึ่งการประมวลจะส่งงานไปให้แต่ละหน่วยประมวลพร้อมๆกันข้อดีคือใช้ทรัพยากรน้อยแต่ก็มีข้อเสียคือมีการใช้งานหน่วยความจำร่วมกันทำให้การประมวลผลมาสามารถทำงานเป็นอิสระต่อกันได้
2. คอมพิวเตอร์หลายตัวที่เชื่อมต่อกันด้วยระบบเครือข่าย ซึงจะเป็นการนำคอมพิวเตอร์มาเชื่อมโยงกันแล้วกระจายงานไปให้แต่ละเครื่อง ข้อดีก็คือว่าทรัพยากรที่ใช้ประมวลผลจะแยกกันเป็นอิสระสามารถประมวลผลได้เร็วเพราะไม่ต้องใช้หน่วยความจำร่วมกัน ข้อเสียคือใช้พื้นที่และทรัพยากรมากในการประมวลผล

ส่วนของการแบ่งงาน สามารถที่จะกระทำได้ดังนี้

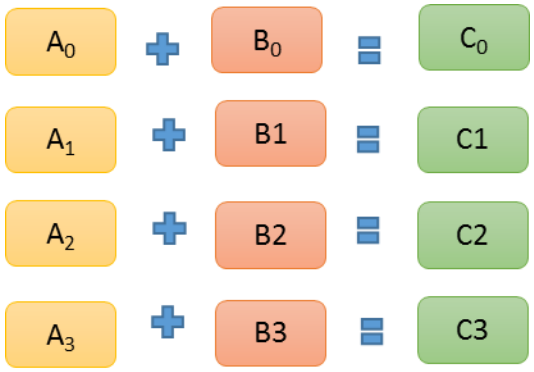
- สามารถแบ่งงานออกเป็นส่วนย่อยๆได้ แล้วสามารถประมวลผลได้พร้อมกัน
- สามารถประมวลผลคำสั่งโปรแกรมหลายๆคำสั่งในขณะใดขณะหนึ่งได้

วิธีการในการพัฒนาโปรแกรมเพื่อให้ทำงานแบบขนานได้นั้น มีหลายวิธีดังนี้

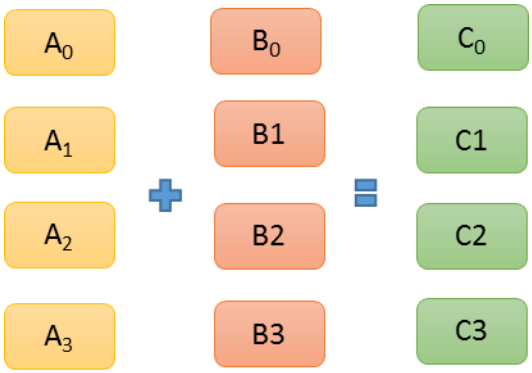
- Shared memory (without threads) ในการพัฒนาโปรแกรมแบบนี้จะเป็นการแบ่งปันหรือใช้งานหน่วยความจำร่วมกันและมีการอ่านเขียนได้โดยไม่ต้องรอว่างานอื่นจะทำงานเสร็จหรือไม่ ดังนั้นข้อดีก็คือไม่มีงานใดเป็นเจ้าของหน่วยความจำ แต่จะมีปัญหาเรื่องการจัดการการล็อคหน่วยความจำ
- Threads การเขียนโปรแกรมนี้จะเป็นการประมวลครั้งเดียวแต่สามารถแบ่งงานย่อยๆแล้วประมวลผลพร้อมกันได้ ทำให้เขียนโปรแกรมง่าย โดย Thread แต่ละตัวของหน่วยประมวลผลกลางเดียวกันจะทำงานแตกต่างกันแต่มีความเกี่ยวข้องกันบางอย่างและต้องทำงานอยู่ภายใต้สภาพแวดล้อมเดียวกัน ซึ่งก็มีข้อจำกัดในการจัดการล็อกทรัพยากรที่ใช้ในการประมวลผล
- Distributed memory / message passing การเขียนโปรแกรมแบบนี้จะให้ชุดคำสั่งของที่ประมวลให้ใช้หน่วยความจำของตัวเองหรือจะกระจายงานให้กับหน่วยประมวลผลกลางอื่นๆก็ได้
- Data parallel การเขียนโปรแกรมโดยจะใช้การแบ่งข้อมูลแล้วนำไปประมวลผลบนหน่วยประมวลกลางโดยชุดคำสั่งที่ต้องประมวลผลจะต้องมีโครงสร้างเดียวกัน

2.1.3 การพัฒนาโปรแกรมแบบขนานเชิงข้อมูล (Data Parallel Programming)

การพัฒนาโปรแกรมแบบขนานเชิงข้อมูล [4] นั้น จัดเป็นวิธีที่ง่ายที่สุดสำหรับการเรียนรู้ เนื่องจากการประมวลผลขนานจะเกิดในมิติของข้อมูลเท่านั้น หรือกล่าวอีกนัยหนึ่งคือ คำสั่งชุดเดียวกันถูกประมวลผลบนข้อมูลหลายชุดในเวลาเดียวกันโดยอาศัยหน่วยประมวลผลจำนวนมากกว่าหนึ่งหน่วย เทคนิคการพัฒนานี้จึงเหมาะสำหรับการประมวลผลบนสถาปัตยกรรมแบบ Single Instruction Multiple Data หรือ SIMD ซึ่งการประมวลผลนั้นจะแตกต่างจากการประมวลผลแบบลำดับ ซึ่งจะประมวลผลข้อมูลชุดเดียวต่อหนึ่งหน่วยซึ่งจะเรียกว่า Scalar Operation ภาพที่ 2-2 แสดงการประมวลผลแบบ Scalar Operation ซึงเป็นการประมวลผลแบบลำดับ นั่นคือนำ $A_0 + B_0$ แล้วค่อยทำคำสั่งถัดมาจนกว่าจะเสร็จทั้งหมด

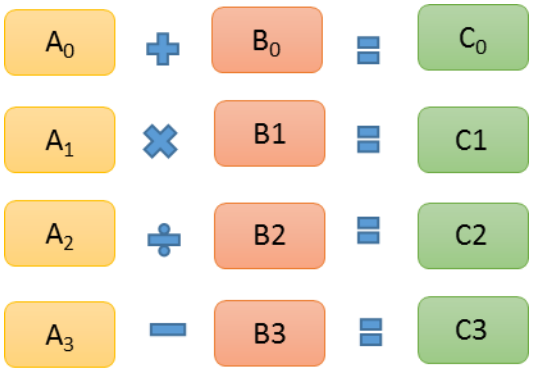


ภาพที่ 2.2 การประมวลผลแบบ Scalar



ภาพที่ 2.3 การประมวลผลแบบ SIMD

การประมวลผลแบบ SIMD จะนำข้อมูลเข้าไปประมวลผลพร้อมกันโดยไม่ต้องเรียงแถวกันประมวลผลและแม้จะมีข้อได้เปรียบของความสามารถในการประมวลผลข้อมูลหลายๆครั้งพร้อมกันได้แต่ว่าจำเป็นต้องใช้กับรูปแบบการประมวลผลที่ได้มีการจัดเตรียมเอาไว้ล่วงหน้ามาก่อนแล้ว



ภาพที่ 2.4 รูปแบบที่ไม่สามารถการประมวลผลแบบ SIMD

ตามภาพที่ 2.3 รูปแบบนี้ไม่สามารถจะทำการประมวลผลแบบ SIMD ได้เพราะเครื่องหมายทางคณิตศาสตร์ไม่ตรงกันจึงไม่สามารถที่จะจัดเตรียมข้อมูลเพื่อประมวลผลได้ต้องใช้การประมวลผลแบบ Scalar เท่านั้น

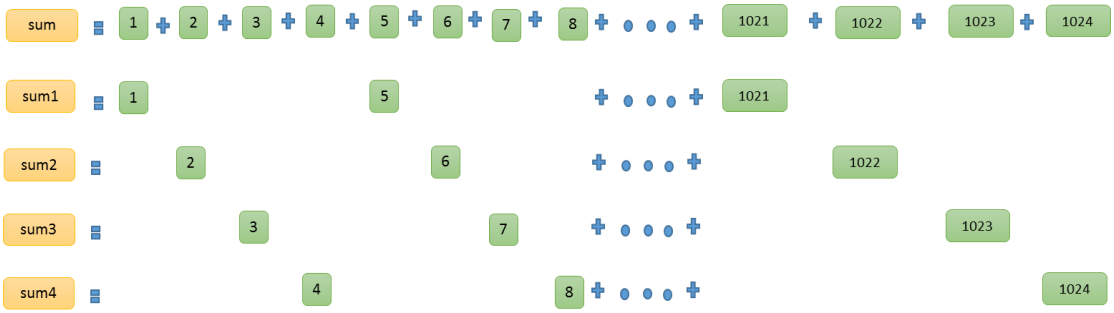
ตัวอย่างของการประมวลผลแบบ Scalar และ SIMD ยกตัวอย่างการหาผลรวมในอาร์เรย์ในกรณีเช่นนี้การประมวลผลแบบ Scalar ธรรมดาจะแสดงได้ดังนี้

```
int a[4] = { 1, 3, 5, 7 };
int b[4] = { 2, 4, 6, 8 };
int c[4];
c[0] = a[0] + b[0];      // 1 + 2
c[1] = a[1] + b[1];      // 3 + 4
c[2] = a[2] + b[2];      // 5 + 6
c[3] = c[3] + c[3];      // 7 + 8
```


ซึ่งก็จะบวกซ้ำๆกันไปถึง 4 ครั้งจะเห็นได้ว่าระบบนั้นทำงานซ้ำซ้อนและทำต้องรอคำสั่งก่อนหน้าให้ทำงานเสร็จก่อนจึงจะประมวลผลได้ ถ้าเป็นโปรแกรมแบบ SIMD จะเป็นในรูปแบบดังนี้

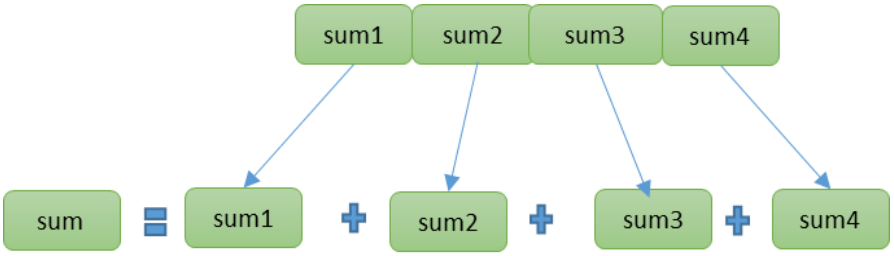
```
a[4]  = { 1, 3, 5, 7 };
b[4]  = { 2, 4, 6, 8 };
c[4];
vector  va = a;
vector  vb = b;
vector vc  = c;
vc = va + vb;     // 1 + 2, 3 + 4, 5
+ 6, 7 + 8
```

มาพิจารณาตัวอย่างการประมวลผลและการแบ่งข้อมูลของการบวกเลข 1-1024 ในตัวอย่างของการแบ่งข้อมูลตามภาพ 2.5



ภาพที่ 2.5 ตัวอย่างการแบ่งข้อมูล

เมื่อแบ่งข้อมูลเสร็จก็ทำการประมวลผลหาผลลัพธ์ และสุดท้ายแล้วก็จะต้องมาบวกกันทั้งหมดอีกครั้ง



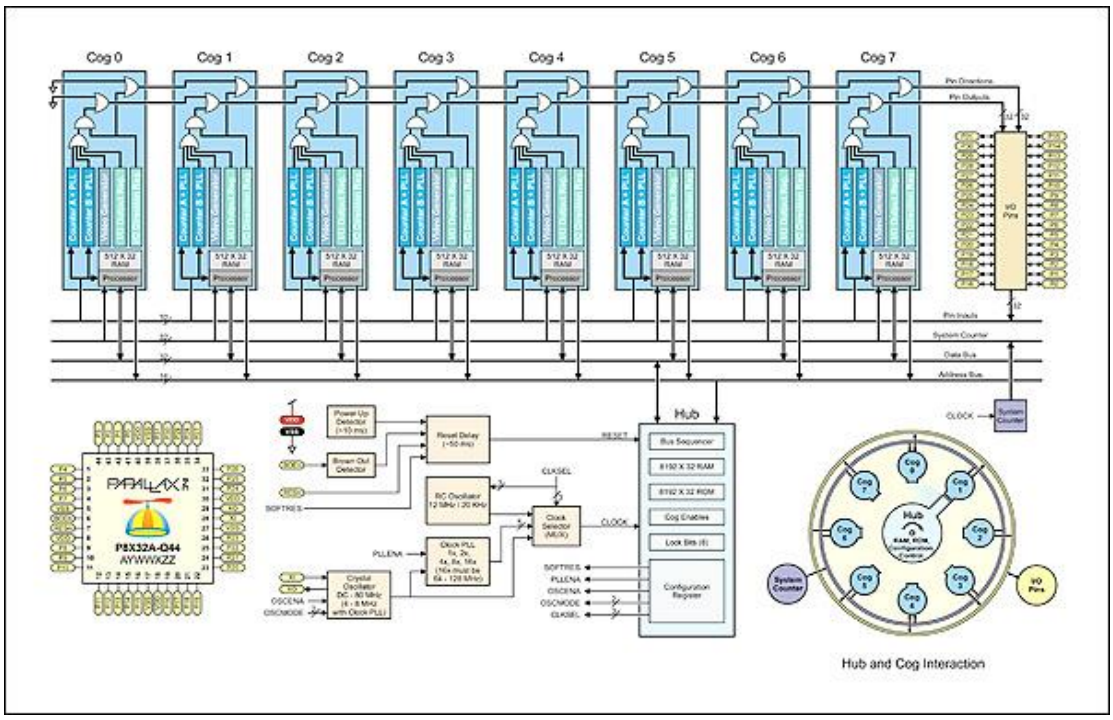
ภาพที่ 2.6 การหาผลลัพธ์รวม

จากตัวอย่างข้างต้นนี้สามารถที่เขียนเป็นโปรแกรมต้นแบบได้ดังนี้

```
MAX_NUM = N
Va       = {1, 2, 3, 4 ,… N};
Vsum     = {0, 0, 0, 0,… , N};
Vstep    = {4, 4, 4, 4 ,… 4};
For (I = 1; I <= MAX_NUM; I += 4) {
     vsum = vsum + va ;
     va   = va+ vstep ;
}
sum  = psum[0] + psum[1] + psum[2] +
psum[3];
```

2.1.4 Multi-Core Parallax Propeller

Multi-Core Parallax Propeller [1] เป็นไมโครคอนโทรลเลอร์แบบ 32 บิตซึ่งผลิตและออกแบบโดยบริษัท Parallax Inc. ที่มีสถาปัตยกรรมภายในที่พิเศษคือมีจำนวนโปรเซสเซอร์อยู่ถึงแปดตัวที่สามารถทำงานแยกจากกันอย่างอิสระหรือร่วมกันทำงานก็ได้ นับเป็นแนวคิดที่นับเป็นการปฏิวัติวงการไมโครคอนโทรลเลอร์ 32 ครั้งสำคัญอีกครั้งหนึ่ง

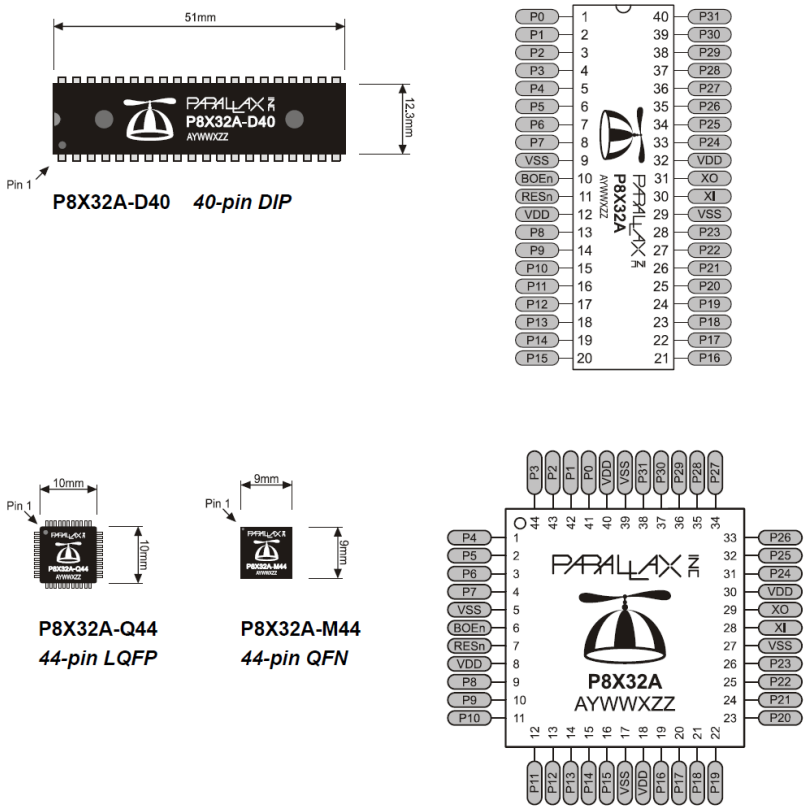


ภาพที่ 2.7 สถาปัตยกรรมของ Propeller (Block Diagram)

คุณสมบัติเด่นของ Propeller [2] มีดังนี้

- ประกอบไปด้วยโปรเซสเซอร์ 8 ตัวหรือเรียกอีกอย่างว่า Cogs ซึ่งสามารถทำงานพร้อมๆกันอย่างเป็นอิสระโดยมีการควบคุมการใช้ทรัพยากรร่วมกันผ่านตัวกลางที่เรียกว่า Central hub
- มีความรวดเร็วในการประมวลสูงเนื่องจากว่าแต่ละ Cogs นั้นจะทำงานเป็นอิสระต่อกันทำให้สามารถรองรับการตอบสนองต่อเหตุการณ์ต่างๆที่เกิดในระบบอย่างรวดเร็วจึงไม่ต้องใช้กระบวนการ Interrupt เข้ามาช่วยประมวลผล ทำให้การเขียนโปรแกรมลดความซับซ้อนลงได้อย่างมาก

- มีการใช้ System Clock ร่วมกันทำให้สามารถอ้างอิงค่าเวลาเดียวกันได้ทำให้การทำงานในแต่ละ Cogs นั้นสอดคล้องกัน
- ภาษาสปินซึ่งมีลักษณะเป็นโปรแกรมภาษาระดับสูงแบบออบเจ็คได้รับการออกแบบให้สามารถรองรับการทำงานของ Propeller อย่างมีประสิทธิภาพ
- ภาษาแอสเซมบลีของ Propeller นั้นได้ทำการจัดเตรียมคำสั่งเพื่อตรวจสอบเงื่อนไขและตัวแปรในการตรวจสอบการทำงานได้อย่างดี ทั้งยังสามารถรองรับงานในลักษณะที่ต้องตัดสินใจพร้อมๆกันหลายๆเงื่อนไข
- ในแต่ละ Cogs นั้นจะประกอบด้วยตัวโปรเซสเซอร์ที่ทำงานเป็นอิสระต่อกันโดยแต่ละตัวจะมีหน่วยความจำ 2 กิโลไบต์ที่เมื่อกำหนดให้ทำงานเป็นรีจิสเตอร์ 32 บิต จะได้ทั้งสิ้น 512 ตัวและมี Program Counter ที่มีความสามารถสูงที่จะทำงานร่วมกันกับเฟสล็อกลูป ทำให้การทำงานในแต่ละ Cogs ทำงานได้เร็วถึง 80 MHz รวมถึงยังมีวงจรกำเนิดสัญญาณภาพและส่วนควบคุม I/O อีกด้วย
- มี Port I/O ทั้งหมด 32 ขาโดยจะกำหนดให้ใช้ 2 ขาสำหรับติดต่อหน่วยความจำ EEPROM สำหรับเก็บโปรแกรมของผู้ใช้งาน และอีก 2 ขาสำหรับการดาวน์โหลดโปรแกรม
- Propeller เก็บข้อมูลโปรแกรมของผู้ใช้บน EEPROM ภายนอก ทำให้อายุการใช้งานของตัวชิปนั้นไม่ขึ้นกับจำนวนครั้งในการลบหรือบันทึกโปรแกรมเข้าไปใหม่
- ด้วยความสามารถในส่วนกำเนิดสัญญาณภาพที่มีมากถึง 8 ชุด (แต่ละ Cogs จะมีตัวกำเนิดสัญญาณเป็นของตัวเอง) ทำให้มีความสะดวกในการแสดงผลภาพได้หลายๆแบบพร้อมๆกัน
- สามารถเชื่อมต่อกับ Mouse และ Keyboard ได้
- มี 3 แบบคือ DIP 40 ขา LQFP 44 ขา และ QFN 44 ขา



ภาพที่ 2.8 รูปแบบของ Propeller ไมโครคอนโทรลเลอร์

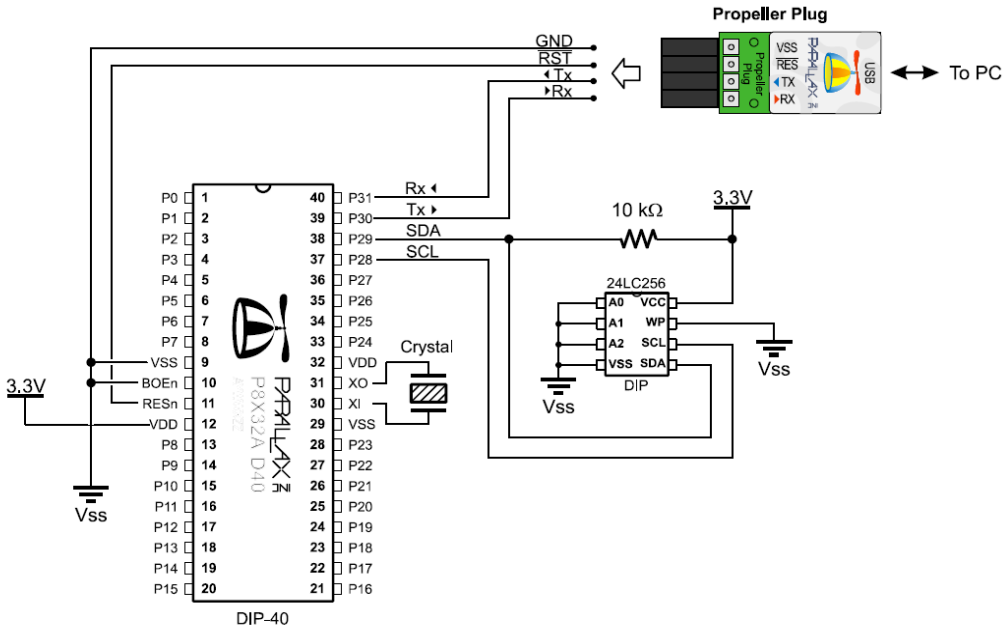
คุณสมบัติทางด้านเทคนิคของ Propeller

- เป็นไมโครคอนโทรลเลอร์ที่ภายในประกอบด้วยโปรเซสเซอร์ขนาด 32 บิตถึง 8 ชุด
- ทำงานด้วยแรงดันเฉลี่ย 3.3 โวลต์ (2.7V – 3.6V)
- แต่ละขาและ Port สามารถจ่ายกระแสไฟได้ 40mA ต่อขาและ 100mA ต่อ Port (8 ขา)
- มีวงจรกำเนิดสัญญาณนาฬิกาภายใน 12 MHz หรือ 20 kHz เลือกกำหนดค่าได้
- ทำงานด้วยสัญญาณนาฬิกาจากภายนอกความถี่ตั้งแต่ DC – 80 MHz
- สามารถใช้คริสตอล 4 MHz ถึง 8 MHz ร่วมกับตัวคูณสัญญาณนาฬิกาซึ่งมีความถี่สูงสุด 80 MHz
- หน่วยความจำทั้งระบบแบ่งเป็นหน่วยความจำ EEPROM 32 กิโลไบต์ และมีหน่วยความจำชั่วคราว 32 KB
- ในแต่ละ Cogs มีหน่วยความจำชั่วคราวตัวละ 2KB

- การจัดการหน่วยความจำเป็นแบบ 32 บิต
- มีจำนวน I/O 32 ขา

ตำแหน่งขา	หน้าที่	รายละเอียด
P0-P31	I/O	มีวัตถุประสงค์ทั่วไปเพื่อ เป็น Port A ที่สามารถเป็นแหล่งกำเนิดไฟฟ้าได้ 40 mA ที่แรงดัน 3.3 VDC ระดับ Logic Threshold ที่ประมาณ ½ VDD หรือ 1.6 VDC ที่แรงดันไฟเลี้ยง 3.3 V ขาบางขายังมีหน้าที่พิเศษ เมื่อเปิดหรือรีเซ็ตระบบดังนี้ • P28 เป็นขา SCL ของ I2C สำหรับการเชื่อมต่อกับ EEPROM ภายนอก • P29 เป็นขา SDA ของ I2C สำหรับการเชื่อมต่อกับ EEPROM ภายนอก • P30 เป็นขา Tx ส่งข้อมูลกับ Serial Port • P31 เป็นขา Rx รับส่งข้อมูลกับ Serial Port
VDD	-	ขาไฟบวก (2.7V - 3.6V)
VSS	-	ขากราวด์
BOEn	I	ขา Enable Brown Out และจำเป็นต้องเชื่อมต่อกับ VDD หรือ VSS อย่างได้อย่างหนึ่ง และจะทำงานที่ Logic “0” ถ้าขานี้เป็น “0” จะทำให้ขารีเซ็ททำหน้าที่เป็น Output แต่ยังคงสามารถส่ง Logic “0” ให้ขารีเซ็ตเพื่อรีเซ็ทไมโครคอนโทรลเลอร์ได้ ถ้าขานี้เป็น “1” จะทำหน้าที่ Input แบบ Schmitt Trigger

RESn	I/O	ขารีเซ็ท ทำงานที่ลอจิก “0” เมื่อขานี้เป็น “0” Propeller จะถูกรีเซ็ท Cogs ทั้งหมดจะถูก Disable I/O อยู่ในสถานะลอย Propeller ใช้เวลาในการรีเซ็ททั้งหมด 50 มิลลิวินาที หลังจากนั้นจะเปลี่ยนค่าจาก 0 เป็น 1
XI	I	ขา Input ของคริสตอลใช้ต่อกับแหล่งกำเนิดสัญญาณนาฬิกาจากภายนอก (ในกรณีนี้ขา XO จะไม่ได้ใช้งาน) หรือต่อเข้ากับขาด้านใดด้านหนึ่งเข้ากับคริสตอล หรือ โรเซเตอร์ (ขาอีกข้างต่อกับ XO) โดยไม่จำเป็นต้องต่อตัวต้านทานหรือตัวเก็บประจุภายนอก
XO	O	ขา Output ของคริสตอลออกแบบเพื่อส่งข้อมูลกลับไปที่คริสตอลการต่อคริสตอลนั้นจะสัมพันธ์กันกับการกำนดค่าในรีจิสเตอร์ CLK ด้วย



ภาพที่ 2.9 การเชื่อมต่อ Hardware ภายนอก

2.1.5 คอมไพเลอร์ (Compiler)

คอมไพเลอร์คือตัวแปลโปรแกรมที่พัฒนาขึ้นด้วยภาษาระดับสูง ไปเป็นภาษาเครื่องต่อไปนี้จะแสดงถึงขั้นตอนการทำงานหลัก ๆ ของคอมไพเลอร์ซึ่งในปัจจุบัน คอมไพเลอร์สมัยใหม่ อาจมีขั้นตอนมากกว่า และมีเทคนิคเพิ่มเติมขึ้นมาก ซึ่งแต่ละผลิตภัณฑ์จะไม่บอกรายละเอียดการทำงาน เนื่องจากเป็นความลับของแต่ละผลิตภัณฑ์ของตน แต่ต่อไปนี้จะเป็นการกล่าวถึงหลักการทำงาน ที่ทุก ๆ คอมไพเลอร์จะต้องมีใช้งานเป็นอย่างน้อย

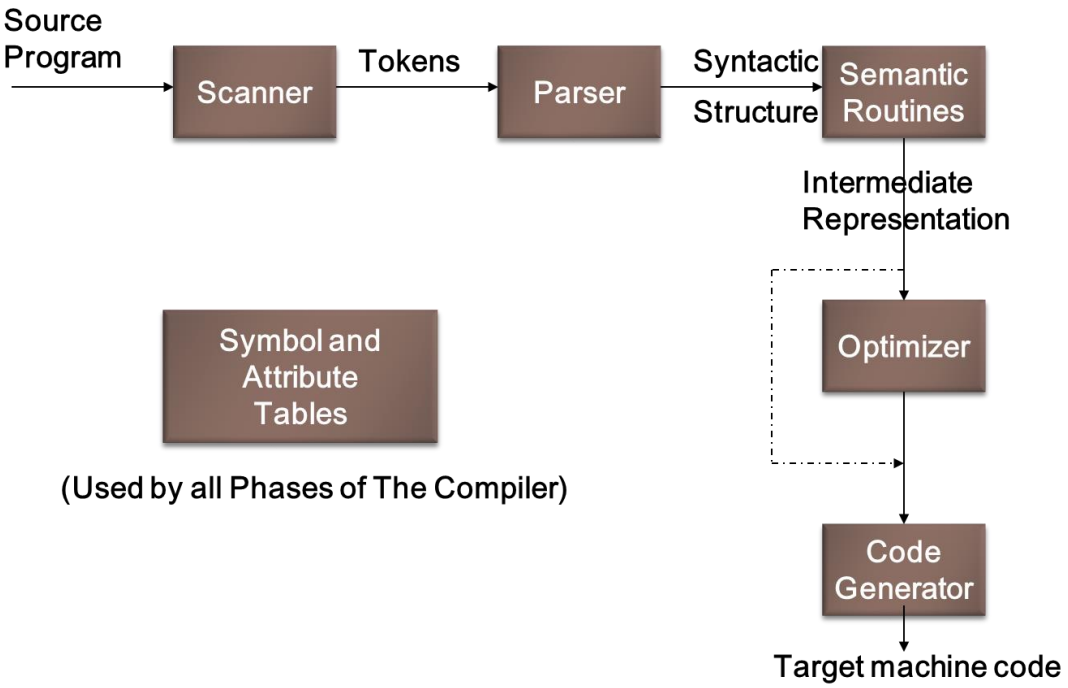
2.1.4.1 ความหมายของคอมไพเลอร์

ภาษาเครื่อง (Machine Language) นั้นเป็นภาษาหรือชุดคำสั่งที่เครื่องคอมพิวเตอร์สามารถเข้าใจและสามารถติดต่อได้โดยตรง อยู่ในรูปแบบของเลขฐานสอง (Bit/Binary) ซึ่งเป็นเรื่องยากที่มนุษย์จะทำความเข้าใจและเขียนคำสั่งดังกล่าว จึงทำให้เกิดการพัฒนาภาษาสัญลักษณ์ (Symbolic Language) ได้แก่ ภาษาแอสเซมบลี (Assembly Language) แต่ภาษาสัญลักษณ์นั้นยังคงยากสำหรับมนุษย์ จึงได้เกิดการพัฒนาภาษาในระดับสูง ได้แก่ ภาษาซี (C language) และภาษาจาวา (Java language) เป็นต้น และใช้คอมไพเลอร์แปลภาษาระดับสูงดังกล่าวให้เป็นภาษาเครื่องอีกครั้ง

2.1.4.2 ขั้นตอนการทำงานของคอมไพเลอร์

เริ่มตั้งแต่การตรวจสอบความถูกต้องของคำสั่งจากภาษานั้น ๆ ว่าถูกต้องตามกฎเกณฑ์หรือไม่ ตัวอย่างเช่นกฎเกณฑ์ภาษาจาวาไม่สามารถเขียนเพียงตัวแปรเพียงตัวเดียวได้ เช่น A; แต่สามารถเขียนการกระทำกับตัวแปร เช่น A++; หรือ A=5; ได้ และเมื่อคำสั่งถูกต้องตามกฎเกณฑ์ จึง

ดำเนินการแปลเป็นภาษาเครื่อง และทำงานได้ผลลัพธ์เช่นเดียวกับโปรแกรมที่เขียนขึ้นจากภาษาต้นแบบ ซึ่งขั้นตอนการทำงานของคอมไพเลอร์นั้นประกอบด้วยขั้นตอน 5 ขั้นตอนหลัก ดังนี้



ภาพที่ 2.10 แสดงการทำงานของคอมไพเลอร์

1. Lexical Analyzer ทำหน้าที่อ่านอักขระจากโปรแกรมต้นแบบแล้วแยกอักขระเหล่านั้นออกเป็นกลุ่ม ๆ เรียกว่า โทเคน (Token) ตามลักษณะหน้าที่ที่ได้ระบุไว้ในโครงสร้างและกฎเกณฑ์ของภาษา ซึ่งเมื่อแยกอักขระแล้วจะได้ ตัวแปร และตัวดำเนินการ
2. Syntax Analyzer ทำหน้าที่ตรวจสอบความถูกต้องโครงสร้างของภาษา ต้นแบบว่าถูกต้องตามกฎไวยากรณ์ของภาษานั้น ๆ หรือไม่
3. Semantic Analyzer ทำหน้าตรวจสอบความหมายของภาษาและโครงสร้างของรหัสชุดคำสั่งที่ผ่านการตรวจสอบความถูกต้องของโครงสร้างแล้ว นำมาทำรหัสกลาง ตัวอย่างรหัสกลาง ได้แก่ สัญกรณ์คณิตศาสตร์ เป็นต้น ซึ่งรหัสกลางสามารถที่จะถูกแปลงเป็นภาษาเครื่องต่าง ๆ ที่ไม่เหมือนกันได้ง่าย
4. Code Optimization เป็นขั้นตอนการแก้ไข ปรับปรุง รหัสกลาง ให้มีความสามารถมากยิ่งขึ้น เช่น หากโปรแกรมมีการคำนวณค่าเดิม ๆ ซ้ำ โค้ดออฟติไมเซอร์จะทำการลดขั้นตอนการคำนวณลง โดยให้ใช้ค่าที่ได้คำนวณแล้วแทน
5. Code Generation เป็นส่วนที่เพิ่มเติมเพื่อปรับปรุงภาษากลางให้เป็นภาษาเครื่อง ซึ่งจะขึ้นอยู่กับซีพียู และเครื่องปลายทาง เช่น การกำหนดตำแหน่งของข้อมูลในหน่วยความจำ การกำหนดรีจิสเตอร์ต่าง ๆ ที่จะใช้

2.4.1.3 การจัดตารางของคอมไพเลอร์

เป็นอีกหน้าที่ที่สำคัญของคอมไพเลอร์ จะต้องทำการบันทึก ชื่อตัวแปร ชื่อโพรซีเดอร์ ชื่อฟังก์ชัน ที่ปรากฏอยู่ในโปรแกรมต้นแบบไว้ในตาราง รวมทั้งบันทึกรายละเอียดต่าง ๆ เช่น ชนิด และแอดเดรสของแต่ละตัวแปร

2.4.1.4 การตรวจจับและดำเนินการต่อเมื่อพบความผิดพลาด

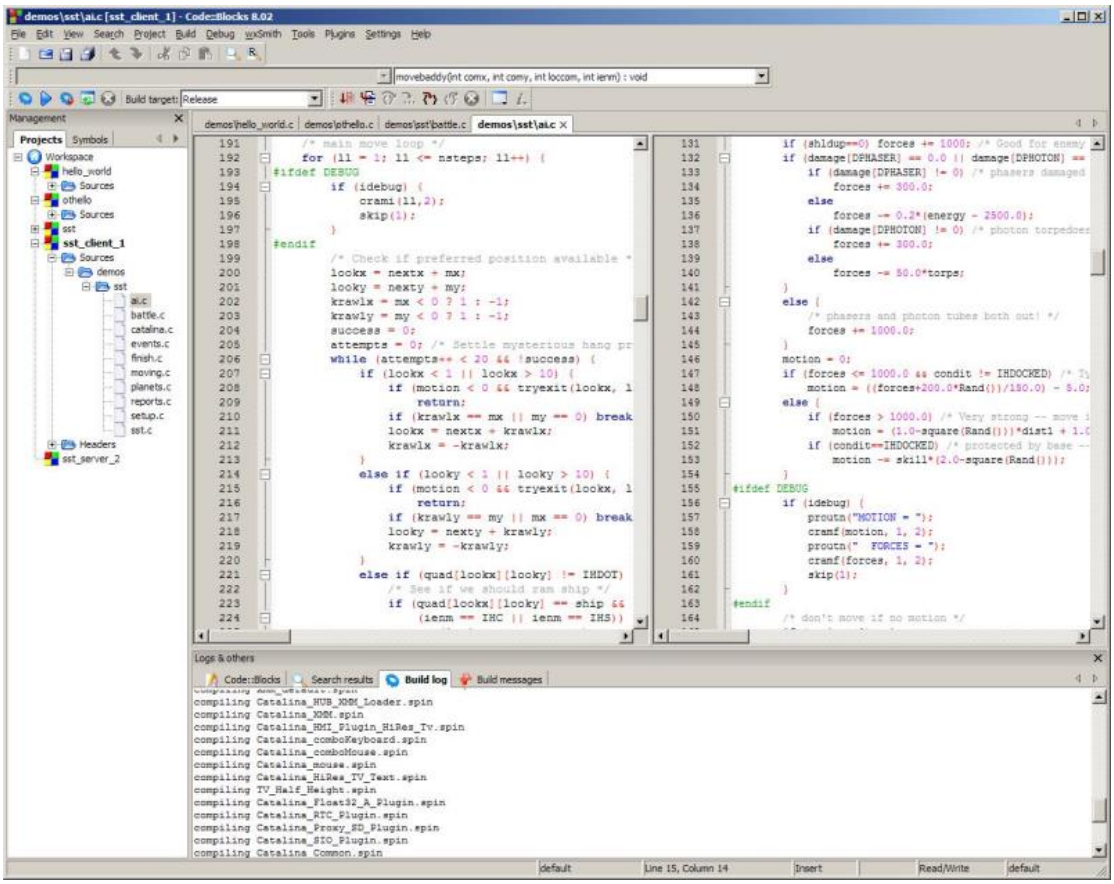
คอมไพเลอร์ที่ดีหากโปรแกรมต้นแบบเขียนผิดหลักภาษา จะต้องแสดงข้อผิดพลาดที่ใกล้เคียงความจริงมากที่สุดให้กับผู้พัฒนาโปรแกรมทราบ นอกจากการตรวจจับข้อผิดพลาดแล้ว คอมไพเลอร์ต้องสามารถดำเนินการต่อเพื่อตรวจสอบข้อผิดพลาดอื่น ๆ ได้อีก ไม่หยุดการทำงานเมื่อพบข้อผิดพลาดเพียงครั้งเดียว คอมไพเลอร์ต้องพยายามแก้ไขความผิดพลาดที่เกิดขึ้น (Error Correction) เพื่อให้สามารถคอมไพล์โปรแกรมต้นแบบต่อไป และนำส่วนที่พยายามแก้ไขให้ถูกต้องนั้นรายงานให้กับผู้พัฒนาโปรแกรมทราบเมื่อเสร็จสิ้นการคอมไพล์

2.2 เอกสารและงานวิจัยที่เกี่ยวข้อง

2.2.1 Catalina C

Catalina C [5] เป็นคอมไพเลอร์ภาษา C ของ Propeller ที่ได้สร้างขึ้นโดย Ross Higson ซึ่งมีคุณสมบัติหลักดังนี้

- ใช้มาตรฐาน ANSI C C89 Library และ บาง Function ใน C99
- สนับสนุน Floating point (32 bit IEEE 754)
- มีเครื่องมือในการ Debug Program
- ไม่ขึ้นต่อ Platform ใดๆ สามารถใช้ได้ทั้ง Windows , Linux
- สนับสนุนการทำงานแบบ Multi-Core , Multi-Thread แบบ Concurrency
- สนับสนุนการทำงานบน Code::Blocks IDE .



ภาพที่ 2.11 การใช้งาน Catalina C บน Code::Blocks IDE

Catalina C นั้นยังไม่สามารถเขียนโปรแกรมแบบขนานได้เนื่องจากยังไม่มีคำสั่งที่สามารถควบคุมการทำงานของแต่ละ Cogs ให้เพราะว่า Catalina C ทำการประมาณผลแบบ Concurrency โดยการ Run 8 Cogs พร้อมๆกัน

2.2.2 RZ Language

RZ Language [6] มีวัตถุประสงค์เพื่อการเรียนการสอนเป็นภาษาสำหรับการเขียนโปรแกรมระบบในวิชาสถาปัตยกรรมคอมพิวเตอร์ซึ่งเน้นภาษาขนาดเล็ก ที่สามารถใช้ในการแสดงถึงการทำงานภายในทั้งหมดของระบบคอมพิวเตอร์ ช่วยให้นักเรียนได้ศึกษาเกี่ยวกับระบบคอมพิวเตอร์ที่มาจากภาษาระดับสูงลงไปที่บิตของโปรเซสเซอร์ และ RZ เป็นภาษาที่ง่ายเนื่องโครงสร้างภาษานั้นเป็นแบบพื้นฐานที่สุด และจะดูเหมือน C (ที่ไม่มีการกำหนด Type)

คุณสมบัติหลักของภาษา RZ นั้นมีดังนี้

1. เป็นภาษาที่เป็นกลุ่มย่อยของภาษา C แต่ตัวแปรจะไม่มี Type
2. มีเพียงตัวแปร Integer เท่านั้น

3. การใช้ตัวแปร Global จำเป็นต้องประกาศก่อนการใช้งาน แต่ถ้าเป็นตัวแปร Local นั้นจะสามารถใช้ได้เลย
4. ตัวแปร Global ประกาศเป็นอาร์เรย์ได้แต่ต้องกำหนดขนาดให้ชัดเจนตั้งแต่ตอน Compile และประกาศได้เพียงอาร์เรย์มิติเดียว
5. มีชื่อสงวนดังต่อไปนี้ if, else, while, return, print
6. มี Operators ดังนี้ +, -, *, /, ==, !=, <, <=, >, >=, ! , &&, ||, * (dereference), & (address).

ตัวอย่างการเขียนโปรแกรม RZ

```
// find max in an array
    a[10], N;
    init(){
    i = 0;
    while ( i < N ) {
        a[i] = I;
         i = i + 1;
         }
     }
     main(){
     N = 10;
     init();
     max = a[0];
     I = i;
     while( I < N ) {
          if( max < a[i] ) max = a[i];
```

```
        i = i + 1;
            }
    print( "the max value is ", max );
        }
```

จะเห็นได้ว่า RZ นั้นมีขนาดเล็ก ดังนั้นจึงเหมาะกับการนำมาเขียนโปรแกรมบนไมโครคอนโทรลเลอร์ ซึ่งมีขนาดของหน่วยความจำจำกัด และเมื่อนำสร้างเป็นคอมไพเลอร์ของ Propeller นั้นทำให้สามารถGenerate Code หรือ Optimize Code เองได้ตามต้องการ

บทที่ 3
ระเบียบขั้นตอนวิธีที่เสนอ

แนวทางการวิจัยนี้จะนำเสนอวิธีการสร้างคอมไพเลอร์โดยใช้หลักการพัฒนาโปรแกรมแบบขนาน และใช้ การโปรแกรมแบบขนานเชิงข้อมูล ใช้การวิเคราะห์ข้อมูลเพื่อจะทำการประมวลผลแบบขนานได้อย่างมีประสิทธิภาพ โดยจะทำการพัฒนาภาษา RZ ให้สามารถสร้างโค้ดแอสเซมบลีของ Propeller

3.1 การคอมไพล์โปรแกรม

ในงานวิจัยนี้จะใช้ทำการสร้างโปรแกรมต้นแบบภาษาแอสเซมบลี โดยจะเพิ่ม Syntax พิเศษเข้าไปในภาษา RZ เพื่อให้คอมไพเลอร์แปลงเป็นโปรแกรมแบบขนานได้โดยจะเพิ่มดังนี้

1. #num_of_cpu เพื่อระบุจำนวน Cogs ที่ต้องการประมวลผล
2. @ var { ... code } โดยให้ var = {0,1,2,3....num_of_cpu -1} เพื่อระบุว่าจะให้ Cog ไหนเป็นตัวประมวลผล

จากนี้จะเป็นการอธิบายการคอมไพล์โปรแกรมแบบขนานของคอมไพเลอร์ที่ได้พัฒนา

3.1.1 โปรแกรมหาค่าสูงสุดใน Array

```
// find max in an array
    N = 1024;
    #no_of_cpu = 8
    @i while( i < N ) {
          if( max < a[i] ) max = a[i];
              i = i + 1;
           }
         }
```

ในขั้นตอนเริ่มต้นนั้นคอมไพเลอร์จะทำการแปลง #, @ ให้อยู่ในรูป Normal Form และคำนวณหาจำนวนข้อมูลที่จะต้องแบ่งเป็นสิ่งแรก

“no_of_data = Roundup (length (N)/no_of_cpu) “

จากตัวอย่างจะเท่ากับ 1024/8 = 128 เมื่อได้จำนวนข้อมูลที่จะต้องแบ่งแล้วก็จะทำแบ่งข้อมูลที่ต้องไปประมวลผลในแต่ละ Cog

@0 [แทน i ด้วย 0]

@1 [แทน i ด้วย no_of_data (1) -1]

...

@no_of_cpu -1 [แทน i ด้วย (no_of_data * no_of_cpu –1)]

ซึ่งคอมไพเลอร์จะยังคงไม่แปลงเป็นแอสเซมบลีแต่จะแปลงเป็น RZ โปรแกรมแบบกระจาย Loop ออกไปทำงานในแต่ละ Cog


```
// find max in an array parallel
    N = 1024;
    no_of_data = 128
    i_0 =127
    i_1 = 255
    ...
    i_no_of_cpu-1 = no_of_data * no_of_cpu-1
    @0 while( i_0 < 127 ) {
          if( max[0] < a[i] ) max[0] = a[i];
              i = i + 1;
            }
        }
    ...
```

```
@no_of_cpu-1   while(   i_   <   no_of_data   *
no_of_cpu-1 ) {

       if   (   max[no_of_cpu-1]   <   a[i]   )
max[no_of_cpu-1] = a[i];

          i_ no_of_cpu-1   = i_ no_of_cpu-1   +
1;

         }

     }
```

จากโปรแกรมข้างต้นบนสามารถสร้างเป็น Assembly Code โดยมีขั้นตอนและกระบวนการดังนี้จะต้องทำการ Set Header ของแอสเซมบลีเพื่อให้สามารถทำงานบน Propeller ได้ โดยจะต้องระบุ Cog เป็น 0 เสมอ

```
CON 'Spin setup code

    _clkmode = xtal1 + pll16x

    _xinfreq = 5_000_000

PUB Start

cognew(@main,0)                 'Main cogs
```

ประกาศตัวแปร Global ซึ่งประเภทข้อมูลของ Propeller โดยประเภทข้อมูลที่ประกาศจำนวนคือ Long

```
'RZ a [1024], N, max [8], maxs = 0;
'------- [Global Variables] -----------------
max  long
N    long
a    long
maxs long
'------- [Global Variables] -----------------
```


เมื่อเจอ Statement Assignment จะแปลง ด้วยคำสั่ง mov <X>, #value ยกตัวอย่างเช่น N = 1024 ก็จะได้เป็น mov N, #1024

```
'RZ N = 1024
 mov N,#1024
```

และเมื่อต้องแปลง Flow control Statement ที่จำเป็นต้องเปรียบเทียบค่า ก็จะใช้คำสั่ง CMP แล้วก็ตรวจสอบ และใช้คำสั่ง if_c และ if_nc เป็นคำสั่งในการตรวจสอบ bit C เพื่อตรวจสอบว่าเป็นจริงหรือเท็จ ซึ่งการ set bit C =1 ถ้า value1 > value 2 C= 0 ถ้า value1 <=value2

```
' RZ (if( max[0] < a[i] ) max[0] = a[i];
 cmp          #max, #a      wc " Check bit c
 if_nc   mov  #max, #a
```

การสร้าง Loop จากโปรแกรมในตัวอย่างที่ผ่านมาผนวกเข้ากับการแบ่งข้อมูลด้วย # และ @ ดังนั้นข้อมูลจึงมีการแบ่งเป็น 8 ส่วนโดยคำนวณจากจำนวนของข้อมูลจากที่ระบุไว้โปรแกรมดังนั้นจึงข้อมูลได้แบ่งได้ เท่าๆกันแล้วส่งไปประมวลผลใน Cogs ที่ระบุไว้ในโปรแกรมโดยคำสั่ง cognew (@procedure ,<no_cogs>)

```
...
 @0 while( i_0 < 127 ) {
        if( max[0] < a[i] ) max[0] = a[i];
            i = i + 1;
           }
       }
 ...
```

จากโปรแกรมด้านบนเราจะเห็นว่ามีการกระจาย Loop มาเรียบร้อยและมีการสร้างตัวแปร Local เพื่อให้การทำงานเป็นอิสระต่อกัน เมื่อคอมไพเลอร์สร้างแอสเซมบลีแล้วจะได้ดังนี้

```
...
cognew(0 , @loop_0)
                    mov      temp,#127
: loop_0            cmp      temp_0, #j_1 wc
     if_c cmp       #max,#a      wc
     if_c jmp       loop_0
     if_nc          mov  #max,#a
: add               add  j_1,#1
                    jmp  loop_0
}
...
```

3.1.2 โปรแกรมหายอดรวมในอาร์เรย์

```
a [1024], N ,sum[8];#no_of_cpu = 8
init(){
      i = 0;
      while ( i < N ) {
         a[i] = i;
         i = i + 1;
         }
    }
 main(){
     N = 1024;
     init();
     sums = 0;
     @i  for ( i = 0; i < N) ;i++ ) {
          sum = sum + a[i] ;
       }
    }
 }
```

จากตัวอย่างในข้อที่ 3.1.1 จะใช้หลักการเดียวกันนั่นคือต้องแบ่งข้อมูลออกเป็น 8 ส่วนเช่นเดียวกัน และเมื่อแปลงเป็นแอสเซมบลีได้ดังนี้

```
CON 'Spin setup code
_clkmode = xtal1 + pll16x
_xinfreq = 5_000_000
PUB Start
cognew(@main,0)
DAT
main     mov  N,#1024
         jmp, #init
init          mov  i,#0
              mov  temp_0,#8
: loop        cmp  temp_0,  #i    wc
    if_c mov  sum,#0
         add  i,#1
         shl  sum,#1
         jmp  loop
: data   if_nc
         mov  temp_0,#127
         mov  j_0,#0
cognew(0 , @loop_0)

         mov  temp_1,#255
mov  j_1,#127
```

```
            mov  j_1,#127
    cognew(1 ,  @loop_1)
            mov  temp_2,#383
            mov  j_2,#255
    cognew(2 ,  @loop_2)
            mov  temp_3,#511
            mov  j_3,#383
    cognew(3 ,  @loop_3)
            mov  temp_4,#639
            mov  j_4,#511
    cognew(4 ,  @loop_4)
            mov  temp_5,#767
            mov  j_5,#511
    cognew(5 ,  @loop_5)
            mov  temp_6,#895
            mov  j_5,#767
    cognew(6 ,  @loop_6)
            mov  temp_7,#1023
            mov  j_5,# 895
    cognew(7 ,  @loop_7)
```

```
cognew(7 , @loop_7)
: loop_0      cmp  temp_0, #j_0      wc
    if_c add  #sum,#a
: add         add  j_0,#1
          jmp  loop_0
: loop_1      cmp  temp_0, #j_1      wc
    if_c add  #sum,#a
: add         add  j_1,#1
          jmp  loop_1
loop_2    cmp  temp_2, #j_2      wc
    if_c add  #sum,#a
: add         add  j_2,#1
          jmp  loop_2
: loop_3      cmp  temp_3, #j_3      wc
    if_c add  #sum,#a
: add         add  j_3,#1
          jmp  loop_3
loop_4    cmp  temp_4, #j_4      wc
    if_c add  #sum,#a
: add         add  j_4,#1
```

```
: add         add j_4,#1
              jmp loop_4
: loop_5      cmp temp_0, #j_1      wc
    if_c add  #sum,#a
: add         add j_1,#1
          jmp loop_0
loop_6   cmp  temp_0, #j_1      wc
    if_c add  #sum,#a
: add         add j_1,#1
              jmp loop_6
: loop_7      cmp temp_7, #j_7      wc
    if_c add  #sum,#a
: add         add j_7,#1
              jmp loop_7
              mov i,#8
              mov maxs,#0
              shl max,#8
loop_for :
              add ,sums,#sum
```

```
              shr sum,#1
    if_c      mov maxs,#max
              djnz i,#: loop_for
    : main_ret   ret
'------- [Global Variables] -----------------
    sum  long
    sums long
    N    long
    a    long
'------- [Global Variables] -----------------
```

บทที่ 4
การทดลองและผลการทดลอง

4.1 สภาพแวดล้อมและเครื่องมือที่ใช้ในการพัฒนา

- การทดลองนี้ใช้เครื่องมือเพื่อวัดผลทางด้านประสิทธิภาพ BASIC Stamp Activity Kit ใช้ตัวไมโครคอนโทรลเลอร์ Propeller ที่พัฒนาโดย Parallax Inc.
- การทดลองวัดผลจากเวลาในการประมวลผลเปรียบเทียบกันระหว่าง การทำงานแบบลำดับกับการทำงานแบบขนาน

4.2 โปรแกรมที่ใช้ทดลอง

ในส่วนนี้จะนำเสนอรายละเอียดของโปรแกรมที่ใช้ในการทดสอบประสิทธิภาพของตัวแปลภาษา โดยมีรายละเอียดดังนี้

- โปรแกรม Sum หาผลรวมของจำนวนเต็มในอาร์เรย์ทั้งหมด 16348 จำนวน โดยเริ่มจากกำหนดค่าของจำนวนในอาร์เรย์ 16348 จำนวน จากนั้นจึงหาผลรวมของจำนวนทั้งหมด
- โปรแกรม Find max หาจำนวนที่มีค่ามากที่สุดในอาร์เรย์จากทั้งหมด 16348 จำนวน โดยเริ่มจากกำหนดค่าในอาร์เรย์ 16348 จำนวน จากนั้นจึงทำการหาจำนวนที่มีค่ามากที่สุด

4.3 การประเมินผล

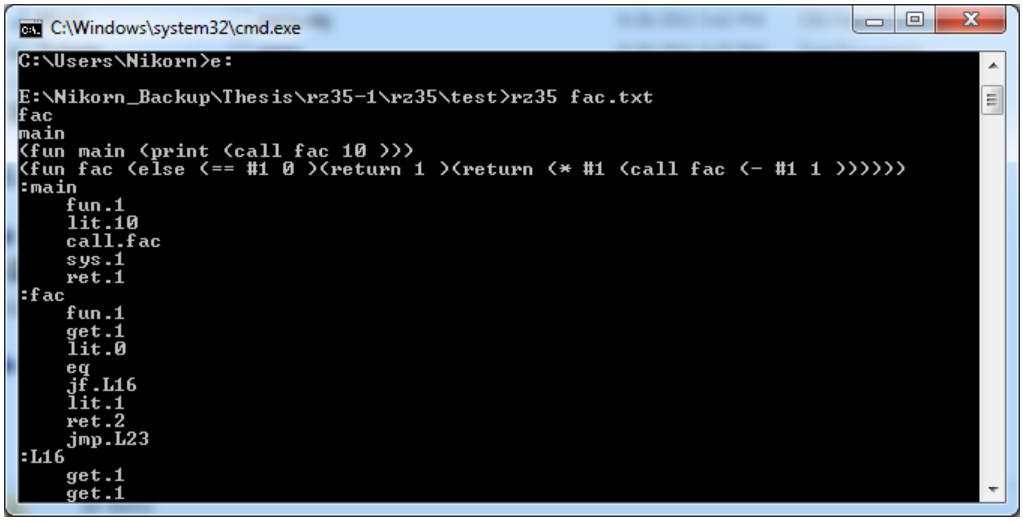
ในการทดลองเพื่อเปรียบเทียบประสิทธิภาพระหว่างการประมวลผลแบบลำดับและการประมวลผลแบบขนาน ของการทดลองทั้ง 2 โปรแกรม โดยประมวลผลทั้งหมด 10 ครั้ง โดยประเมินจากความเร็วในการประมวลผลในแบบลำดับและขนาน ส่วนการทำงานที่แตกต่างกันของโปรแกรมลำดับและขนานนั้นก็คือการระบุ # และ @

```
// find max in an array
     N = 1024;
     #no_of_cpu = 8
     @i while( i < N ) {
            if( max < a[i] ) max = a[i];
                i = i + 1;
              }
           }
```


ซึ่งถ้าเมื่อต้องการให้เป็นโปรแกรมแบบลำดับก็เพียงแต่ลบ # และ @ ออก และเขียนใหม่ได้ดังนี้

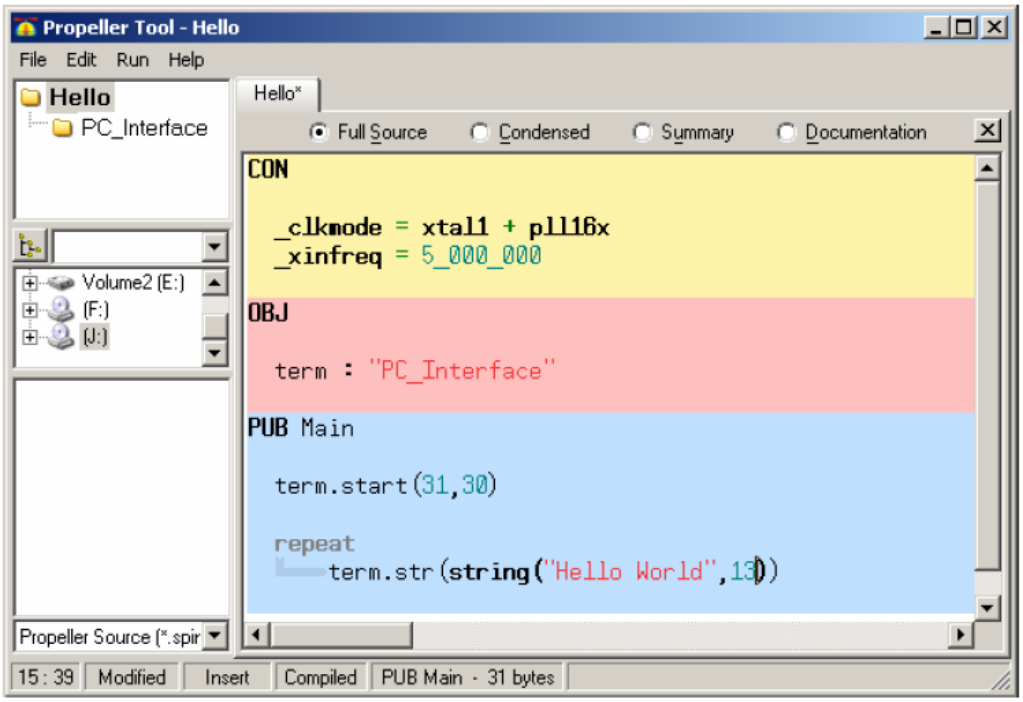
```
// find max in an array
     N = 1024;
      while( i < N ) {
            if( max < a[i] ) max = a[i];
                i = i + 1;
              }
           }
```

หลังจากนั้นจะนำโปรแกรมไปคอมไพล์ด้วยคำสั่งของ RZ Compiler ดังภาพที่ 4.1



ภาพที่ 4-1 คอมไพล์ RZ โปรแกรม

จะได้ Output เป็นแอสเซมบลีและเมื่อแปลงเป็นภาษาแอสเซมบลีแล้วนำเข้าไปแปลงเป็นภาษาเครื่องของ Propeller โดยใช้เครื่องมือพัฒนาของ Palarax Propeller Spin Development ดังภาพที่ 4.2



ภาพที่ 4.2 Propeller Spin Development

และเมื่อแปลงเป็นภาษาเครื่องเสร็จแล้วนั้นจะนำไป Flash ลงบน Propeller และในขั้นตอนการบันทึกเวลานั้นจะใช้ Spin Clock

```
CON
        _clkmode=xtal1 + pll16x
        _xinfreq = 5_000_000
OBJ
        pst : "Parallax Serial Terminal"
PUB Main: ticks
        pst.Start(9600)
        Pause (100) ' 1 ms
```

```
    flag := true
    repeat  while flag
      Pause (1)
      counter++
      pst.newline
      call "Test Program"
      …
      flag = false
  pst.dec(counter)
  PRI Pause(ms)
    waitcnt(clkfreq/1000 * ms + cnt)
```


4.4 การประเมินผล

จำนวนข้อมูลที่ใช้เป็นสิ่งที่แปรผันตรงกับเวลาที่ใช้ในการประมวลผลนั่นคือถ้าจำนวนข้อมูลที่ต้องใช้งานมีมาก เวลาที่ใช้ในการประมวลผลจะมากตามไปด้วย การประมวลผลแบบขนานโดยแบ่งงานไปประมวลบนแต่ละ Cogs ย่อมใช้เวลาน้อยกว่าแบบลำดับ ดังนั้นจึงจะวัดประสิทธิภาพการทำงานโดยวัดเวลาโดยเฉลี่ยที่ใช้ประมวลผลบน Propeller ซึ่งผลลัพธ์ที่ได้เป็นไปตามตาราง 4.1

โปรแกรม	เวลาโดยเฉลี่ยที่ใช้ในการประมวลผล (มิลลิวินาที)	
	แบบลำดับ	แบบขนาน
Sum	112	32
Find max	125	33

ตารางที่ 4.1 ผลการทดลอง

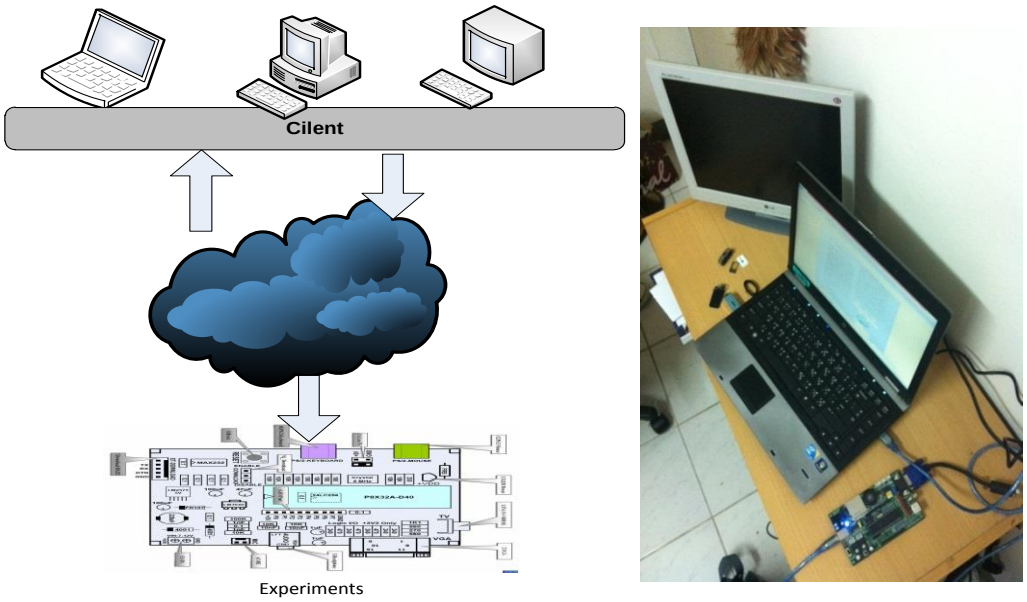
จากการวิเคราะห์ประสิทธิภาพความเร็วในการประมวลผลของโปรแกรมแบบขนานและแบบลำดับจะพบว่าการประมวลผลแบบลำดับจะประมวลผลได้ช้ากว่าเนื่องจากการประมวลผลแบบ single core ดังนั้นเมื่อเราสามารถแบ่งข้อมูลให้อิสระต่อกันและแล้วนำไปประมวลผลแบบแบบขนาน ซึ่งเป็นการประมวลผลแบบ Multi-Core ทำให้เวลาในการประมวลผลลดลง

บทที่ 5
การประยุกต์ใช้

เมื่อทำการพัฒนาเครื่องมือเรียบร้อยแล้วผู้วิจัยได้เล็งเห็นถึงประโยชน์ที่จะได้รับจากการนำประยุกต์ใช้คอมไพเลอร์แบบขนานไปใช้ในการเรียนการสอนได้จึงได้สร้างโปรแกรมประยุกต์ที่สามารถให้นักเรียนนักศึกษาเข้ามาฝึกฝนการเขียนโปรแกรมรายละเอียดจะได้กล่าวในลำดับถัดไป

5.1 ห้องปฏิบัติการระยะไกล

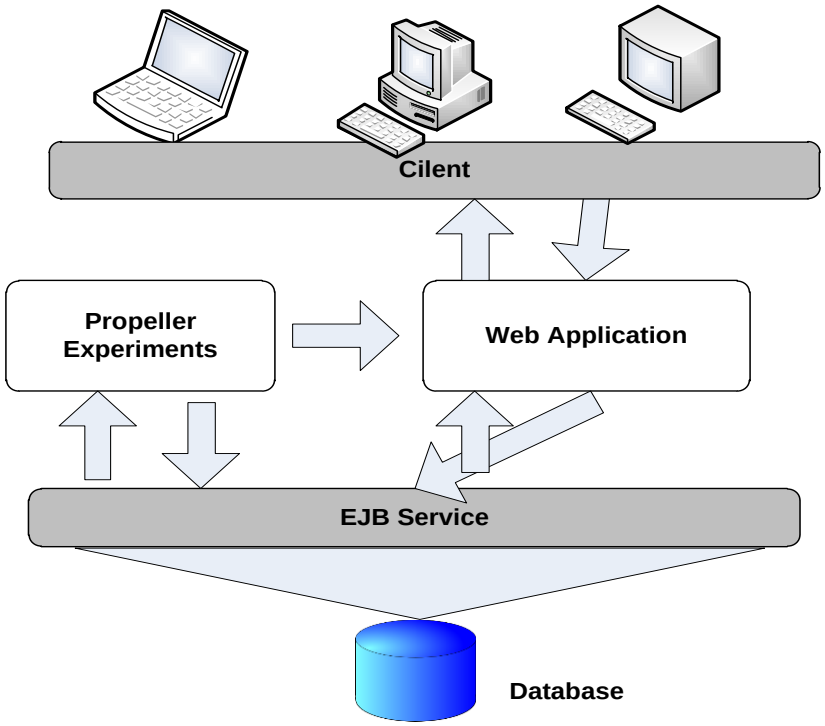
ห้องปฏิบัติการระยะไกลถูกกำหนดไว้บนพื้นฐานว่าการดำเนินการของการทดลองจะสามารถดำเนินการจากระยะไกลโดยนักเรียนไม่จำเป็นต้องอยู่ในห้องปฏิบัติการจริง แต่พวกเขาสามารถเข้าถึงและการควบคุมผ่านเครือข่าย การทำงานของห้องปฏิบัติการระยะไกลที่แสดงในรูปที่ 5.1 นักเรียนสามารถเข้าถึงได้จากระยะไกลและดำเนินการทดลองในห้องปฏิบัติการในห้องปฏิบัติการผ่านทางอินเทอร์เน็ต ผลทั้งหมดถูกจัดเก็บไว้ในฐานข้อมูลที่สามารถเรียกดูได้วิเคราะห์และแสดงภายหลัง



ภาพที่ 5.1 ควบคุมเครือข่ายแบบจำลองห้องปฏิบัติการระยะไกล

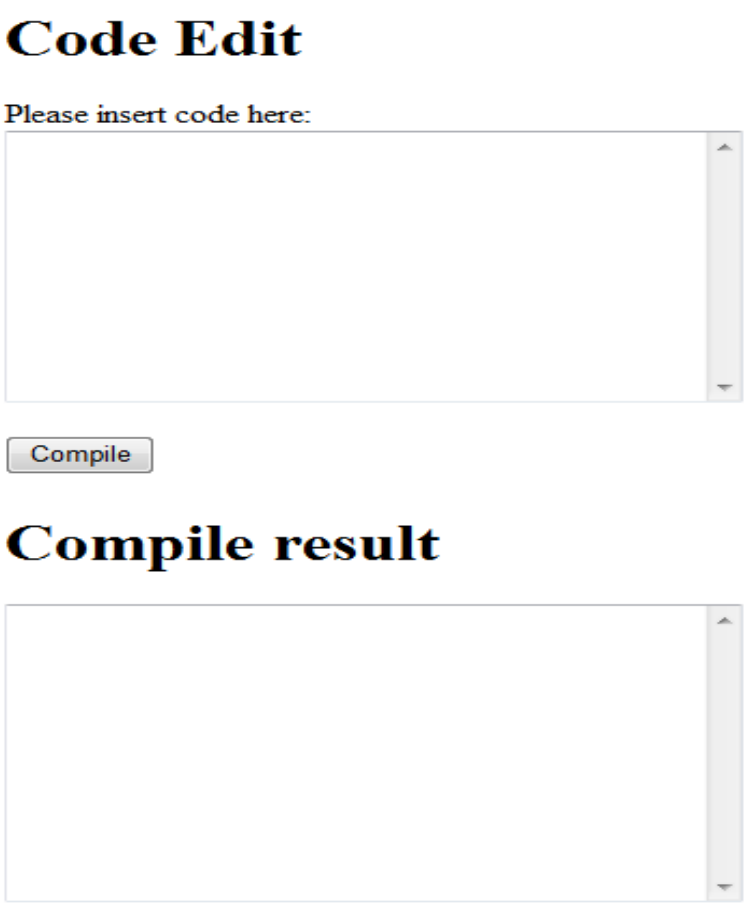
5.2 โปรแกรมควบคุมห้องปฏิบัติการระยะไกล

โปรแกรมควบคุมจะดำเนินงานระยะไกลบนเว็บแอพลิเคชันรูปแบบที่ผู้ใช้จะเข้ามาใช้ห้องปฏิบัติการ แสดงในภาพที่ 5-2 โดยจะแสดงให้เห็นส่วนของห้องปฏิบัติการระยะไกลที่มีโครงสร้างพื้นฐานบนเว็บ ที่พัฒนาโดยภาษา Java



ภาพที่ 5.2 สถาปัตยกรรมโปรแกรมควบคุม

นักเรียนสามารถเข้าถึงไปยังห้องปฏิบัติการผ่านคอมพิวเตอร์ลูกข่ายรวมถึงการเขียนโปรแกรมในหน้าเว็บซึ่งประกอบด้วยพื้นที่ ปุ่ม และ ส่วนประกอบอื่น ๆ ที่จะรองรับ นักเรียนที่จะทำงาน ได้ง่าย แสดงในภาพที่ 5.3



ภาพที่ 5.3 หน้าเว็บสำหรับการเขียนโปรแกรม

หลังจากนั้นนักเรียนจะส่งโปรแกรมไปคอมไพล์ และคำสั่ง RZ คอมไพเลอร์ถูกเรียกโดย EJB Service ของภาษา JAVA และเก็บรวบรวมผลของการคอมไพล์โปรแกรมลงในฐานข้อมูล หลังจากนั้นนักเรียนจะสามารถทดสอบโปรแกรมและเก็บผลการทดสอบลงในฐานข้อมูล

บทที่ 6
สรุปผลการวิจัยและข้อเสนอแนะ

6.1 สรุปผลการวิจัย

การเขียนโปรแกรมในปัจจุบันเป็นการเขียนและประมวลผลการทำงานบนแกนเดียว แต่การพัฒนาหน่วยประมวลผลกลางให้มีหลายแกน เพื่อที่จะทำให้การประมวลผลมีประสิทธิภาพมากขึ้นแต่ตัวแปลภาษาที่ใช้งานอยู่ไม่สนับสนุนให้การเขียนโปรแกรมเพื่อนำประสิทธิภาพของหน่วยประมวลผลกลางมาใช้งานอย่างเต็มที่ งานวิจัยนี้จึงสร้างตัวแปลภาษาเพื่อให้สามารถประมวลแบบขนานได้โดยจะใช้ Parallax Propeller ซึ่งเป็นมัลติคอร์ไมโครคอนโทรลเลอร์แบบที่สามารถที่ประมวลผลแบบหลายแกนได้ที่ดีตัวหนึ่ง เนื่องจากมีถึงแปดโปรเซสเซอร์หรือแกน (เรียกว่า Cogs) ซึ่งสามารถทำงานพร้อมๆ กันหรือแยกกันทำงานอย่างอิสระ

เนื่องจากการพัฒนาโปรแกรมให้สามารถใช้งาน Propeller ไมโครคอนโทรลเลอร์ให้สามารถประมวลผลแบบขนาน จำเป็นต้องเรียนรู้ภาษาสปินหรือแอสเซมบลีและเพื่อใช้งานแต่ละแกน ดังนั้นแนวทางวิจัยนี้จะทำการสร้างคอมไพเลอร์แบบขนานด้วยภาษา RZ ซึ่งเป็นการใช้โครงสร้างภาษาแบบเดียวกันกับภาษา C ซึ่งเป็นภาษาระดับสูงที่เป็นที่นิยมกันมากในการพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์อื่นๆ แต่จะมีเพิ่มสัญลักษณ์พิเศษ @ และ # เข้ามาเพื่อทำให้ผู้พัฒนาสามารถระบุควบคุมการแบ่งข้อมูลและระบุแกนของหน่วยประมวลผลกลางที่ต้องการประมวลผลได้ ยกตัวอย่างเช่น ถ้าต้องการแบ่งข้อมูลออกเป็น 8 สามารถเขียนได้ดังนี้

“# no_of_cpu = 8 “

ถ้าต้องการให้ Loop ได้ๆไปประมวลผลในแกนที่เราต้องการก็ใช้ @ ยกตัวอย่างต่อจากด้านบน

@i for (i=0; i< N; i++)

เมื่อคอมไพลแล้วจะได้แอสเซมบลีของ Propeller ซึ่งเป็นภาษาที่ใกล้เคียงกับภาษาเครื่อง รวมถึงยังมีคำสั่งที่พิเศษสามารถให้ผู้พัฒนาควบคุมหน่วยความจำ ระบุหน่วยที่ประมวลได้เลย ในขั้นตอนการเปรียบเทียบว่าการทำงานที่เป็นแบบขนานและเป็นแบบลำดับจะทำโดยการนำโปรแกรมที่เขียนด้วยภาษา RZ สองโปรแกรมนั่นคือ

- โปรแกรม Sum เป็นโปรแกรมหาผลรวมของจำนวนเต็มในอาร์เรย์ทั้งหมด 16348 จำนวน
- โปรแกรม Find max เป็นโปรแกรมหาจำนวนที่มีค่ามากที่สุดในอาร์เรย์จากทั้งหมด 16348 จำนวน

ประมวลเปรียบเทียบความเร็วในการประมวลผล ซึ่งข้อแตกต่างของการเขียนโปรแกรมแบบขนานลำดับก็คือ สัญลักษณ์ # และ @ กล่าวคือถ้าไม่มีการระบุ @ หรือ # โปรแกรมจะประมวลผลบนแกนหลักของหน่วยประมวลผลกลางเท่านั้น ซึ่งผลลัพธ์ที่ได้จากตาราง 4-2 ก็คือว่า การประมวลผลแบบขนานเร็วกว่าประมาณ 3.5 เท่า

6.2 ข้อจำกัด

ในการดำเนินงานวิจัยนี้มีข้อจำกัดในการใช้งานระบบดังต่อไปนี้

1. รองรับเฉพาะการแบ่งงานแบบเชิงข้อมูลเท่านั้น
2. การพัฒนาโปรแกรมผู้พัฒนายังคงต้องทำการวิเคราะห์เองยังไม่สามารถทำให้การแบ่งงานเป็นไปแบบอัตโนมัติได้

6.3 แนวทางการวิจัยต่อไป

งานวิจัยนี้ยังไม่ครอบคลุมการประมวลแบบขนานแบบอื่นๆนอกเหนือจากการประมวลผลแบบขนานเชิงข้อมูล ดังนั้นการพัฒนาในอนาคตควรทำให้ตัวแปลภาษาสามารถที่วิเคราะห์ว่าจะต้องแบ่งข้อมูลแล้วนำไปประมวลผลที่แกนไหนของหน่วยประมวลผลกลางได้เอง โดยไม่ต้องพึ่งพานักพัฒนาโปรแกรม รวมถึงความสามารถใช้งานได้กับการประมวลผลแบบขนานแบบอื่นๆได้ด้วย

รายการอ้างอิง

1] (2011, October) Parallax Inc. Web Site. [Online]. http://www.parallax.com/

2] Jeff Martin. Format & Editing, Stephanie Lindsay, Propeller Manual. Rocklin, United States of America: Parallax Inc., 2009.

3] ผศ. ดร. ธีรณี อจลากุล, "การพัฒนาโปรแกรมแบบอิมพลิสิต (Implicit Parallel Programming)," การประมวลผลแบบขนานฺ แบบกระจาย. กรุงเทพมหานคร, ไทย: สกอ, 2009, pp. 32-33.

4] ผศ. ดร. ธีรณี อจลากุล, "การพัฒนาโปรแกรมขนานเชิงข้อมูล (Data Parallel Programming)," การประมวลผลแบบขนานและแบบกระจาย. กรุงเทพมหานคร, ไทย: สกอ, 2009, pp. 33-36.

5] (2011, October) Catalina - a FREE C Compiler for the Propeller Chip. [Online]. http://propeller.wikispaces.com

6] Prabhas Chongstitvatana. (2011, September) Rz language. [Online]. http://www.cp.eng.chula.ac.th

ภาคผนวก

ประวัติผู้เขียนวิทยานิพนธ์

นายนิกร มนัส เกิดวันที่ 20 สิงหาคม พ.ศ. 2523 ที่จังหวัดร้อยเอ็ด สำเร็จการศึกษาระดับปริญญาตรีวิศวกรรมศาสตร์บัณฑิต ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น ในปีการศึกษา 2546 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตร์มหาบัณฑิต ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2553


1483634190