การพัฒนาโปรแกรมของขั้นตอนวิธีอุบัติการณ์ร่วมกันบนตัวประมวลผลหลายแกน

นายวีรชิต ศรีมุข

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2554

AN IMPLEMENTATION OF COINCIDENCE ALGORITHM ON MULTI-CORE PROCESSORS

Mr.Weerachit Srimook

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 2011

วีรชิต ศรีมุข : การพัฒนาโปรแกรมของขั้นตอนวิธีอุบัติการณ์ร่วมกันบนตัวประมวลผลหลายแกน. (AN IMPLEMENTATION OF COINCIDENCE ALGORITHM ON MULTI-CORE PROCESSORS) อ. ที่ปรึกษาวิทยานิพนธ์หลัก : ศาสตราจารย์ ดร.ประภาส จงสถิตย์วัฒนา, 38 หน้า.

งานวิจัยชิ้นนี้นำเสนอ การแก้ปัญหาการเดินทางของผู้ขายสินค้า (*Traveling Salesman Problem*) ซึ่งอยู่ในกลุ่มของปัญหาเชิงวิวัฒนาการ โดยใช้ขั้นตอนวิธีอุบัติการณ์ร่วมกัน (*Coincidence Algorithm*) แก้ปัญหา โดยให้มีการทำงานแบบขนานเพื่อลดเวลาการประมวลผลในการหาคำตอบ ปัญหาที่นำมาทดสอบในงานวิจัยจำนวน 5 ปัญหา ที่มีจำนวนเมืองในการเดินทางไม่เกิน 200 เมือง พัฒนาโดยใช้ภาษา C++ ร่วมกับตัวบริการ *Intel Threading Building Blocks* และให้ทำงานบนเครื่องคอมพิวเตอร์ที่มีตัวประมวลผลกลางแบบ 4 แกน เปรียบเทียบกับการทำงานแบบลำดับ

ผลการวิจัยที่ได้สามารถเพิ่มความเร็วในการประมวลผลได้จริง เมื่อวิเคราะห์วิธีการแก้ปัญหา โดยแบ่งงานในขั้นตอนของวิธีการแก้ปัญหาให้ทำงานไปพร้อมๆกัน สามารถทำความเร็วได้มากกว่าแบบลำดับถึง 3 เท่า

ภาควิชาวิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต.....

สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก.....

ปีการศึกษา 2554.....

5371468421 : MAJOR COMPUTER SCIENCE
KEYWORDS : COINCIDENCE ALGORITHM / EVOLUTIONARY COMPUTATION / MULTI-CORE SYSTEM / PARALLEL PROCESSING

WEERACHIT SRIMOOK : AN IMPLEMENTATION OF COINCIDENCE ALGORITHM ON MULTI-CORE PROCESSORS. ADVISOR : PROF. PHASBHAS CHONGSATITWATTANA, Ph.D., 38 pp.

This research presents an evolutionary algorithm to solve Traveling Salesman Problem using Coincidence Algorithm. A parallel processing is implemented to reduce the execution time. Five problems which the number of cities is less than 200 are used to test the implementation. The parallel program is written in C++ language with Intel Threading Building Block tool. The test was conducted on a quad-core processor and the runtime is compared to a sequential program.

The result shows that by divide the computation task and run them in parallel the speed up over the sequential program is up to 3 times.

Department : Computer Engineering Student's Signature .....
Field of Study : Computer Science Advisor's Signature .....
Academic Year : 2011

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จสมบูรณ์ได้เนื่องจากได้รับคำแนะนำและให้คำปรึกษาจาก ศาตราจารย์ ดร.ประภาส จงสถิตย์วัฒนา เป็นผู้แนะนำแนวทางถึงองค์ความรู้ และวิธีการต่างๆที่ให้วิทยานิพนธ์ฉบับนี้เป็นรูปเล่มขึ้นมา ทั้งนี้ขอขอบคุณคณะกรรมการสอบวิทยานิพนธ์ ผู้ช่วยศาตราจารย์ ดร.สุกรี สินธุภิญโญ ผู้ซึ่งเป็นประธานกรรมการ ผู้ช่วยศาตราจารย์ ดร.วรเศรษฐ สุวรรณิก ทำหน้าที่กรรมการ ที่ได้สละเวลาอันมีค่ามาทำหน้าที่นี้ รวมถึงได้ให้คำแนะนำ ชี้ข้อบกพร่องรวมถึงแนวทางแก้ไขต่างๆ ให้กับข้าพเจ้า

ขอขอบพระคุณ คุณแม่ ภรรยา และน้องชาย ที่คอยเป็นกำลังใจและให้คำแนะนำต่างๆ ในการเผชิญกับปัญหาอุปสรรคต่างๆ เพื่อให้สามารถผ่านพ้นเหตุการณ์ดังกล่าวได้โดยดี เพื่อให้สามารถบรรลุถึงเป้าหมายในการเรียนได้จนมาถึงจุดที่สำคัญนี้ได้อย่างภาคภูมิใจ สิ่งสำคัญอีกประการหนึ่งคือ ต้องขอขอบพระคุณที่ให้การสนับสนุนทางด้านการเงินในการศึกษาระดับบัณฑิตศึกษา

สุดท้ายนี้ ขอขอบคุณ เพื่อนๆ พี่ๆ และน้องๆ ที่คอยเป็นกำลังใจ ให้คำปรึกษา ช่วยผลักดันให้เกิดสิ่งดีๆขึ้นมาในการเรียน และการใช้ชีวิตในรั้วจุฬาลงกรณ์มหาวิทยาลัยแห่งนี้

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

บทที่ 1

บทนำ

ความเป็นมาและความสำคัญของปัญหา

การหาคำตอบที่ดีที่สุดสำหรับปัญหาที่ซับซ้อน ถือเป็นงานที่มีความยุ่งยาก และโดยมากแล้วยังไม่มีประสิทธิภาพที่ดีพอเมื่อใช้กับคอมพิวเตอร์ในยุคปัจจุบัน วิธีการหนึ่งที่นิยมนำมาแก้ไขสำหรับปัญหาในกลุ่มนี้คือ อัลกอริทึมวิวัฒนาการ (*Evolutionary Algorithm (EA)*) [1] ซึ่งอยู่ในกลุ่มของการคำนวณแบบวิวัฒนาการ (*Evolutionary Computation*) [2]

Genetic Algorithm (GA) เป็นอัลกอริทึมที่นิยมมากตัวหนึ่งในกลุ่มของ *EA* ซึ่งมีแนวทางการแก้ไขปัญหาโดยแรงบันดาลใจจากวิวัฒนาการทางธรรมชาติ เรียกสั้นๆว่า ขั้นตอนวิธีทางพันธุกรรม อัลกอริทึมนี้มีกลุ่มของประชากร หรือเรียกว่า โครโมโซม โดยถูกเข้ารหัสไว้อาจจะอยู่ในรูปของตัวอักษรหรือตัวเลข ใน 1 ชุดของประชากรเรียกเป็น 1 รุ่น มีการพัฒนาการของตัวโครโมโซมเพื่อการหาคำตอบสำหรับรุ่นต่อไป เช่นการกลายพันธุ์และการข้ามสายพันธุ์ (*Mutation and Cross Over*) ประชากรชุดใหม่ที่ได้ถูกนำไปใช้ในรอบต่อไปของการแก้ปัญหา และจะหยุดการประมวลผลเมื่อประชากรรุ่นสุดท้ายได้ถูกสร้างขึ้น หรือได้ค่าความเหมาะสม (*Fitness value*) ที่ต้องการ *GA* นำไปใช้เพื่อแก้ปัญหาในหลายสาขาวิชา เช่น *Bioinformatics, Phylogenetics, Computational Science, Engineering, Economics, Chemistry, Manufacturing, Mathematics, Physics* และอื่นๆ

ขั้นตอนวิธีอุบัติการณ์ร่วมกัน (*Coincidence Algorithm (COIN)*) [3] เป็นขั้นตอนวิธีหนึ่งอยู่ในกลุ่มของ *EA* เพื่อในการหาคำตอบปัญหา ปัญหาที่นิยมคือ ปัญหาการเดินทางของผู้ขายสินค้า (*Traveling Salesman Problem (TSP)*) *COIN* มีวิธีการคือ แบ่งกลุ่มข้อมูลของประชากรออกเป็นกลุ่มดีและกลุ่มเลว มีการให้รางวัลและการทำโทษของแต่ละคู่ที่เส้นทางที่เกิดขึ้น ถึงแม้ว่ามีความเร็วในการหาคำตอบกว่าขั้นตอนวิธีอื่นๆ แต่มีการทำงานบางขั้นตอนสามารถกระจายงานให้ทำงานแบบขนานได้

ตัวประมวลผลหลายแกน (*Multi-core Processors*) ถือเป็นความก้าวหน้าทางเทคโนโลยีของหน่วยประมวลผลกลาง (*CPU*) ที่มีความเร็วเพิ่มมากขึ้นและจำนวนแกน (*Core*) เพิ่มจำนวนขึ้นจากเดิม 1 แกน เป็น 2 และ 4 แกน แบบก้าวกระโดดและในราคาที่ถูกลง สามารถพบเห็นได้ทั่วไปในเครื่องคอมพิวเตอร์ส่วนบุคคลปัจจุบัน และกำลังก้าวเข้าสู่ยุค *Tera-scale Computing* [4] เพื่อให้มีการใช้ตัวประมวลผลที่มีหลายแกนได้เต็มประสิทธิภาพ จำเป็นต้องมีการเขียนโปรแกรมให้มีการทำงานแบบขนานด้วย เครื่องมือสำหรับการเขียนโปรแกรมให้สามารถประมวลผลแบบขนาน มีด้วยกันหลายตัว เช่น *Intel TBB* [5], *OpenMP* [6], [7] หรือ เธรดปกติ (*Native Threads*)

งานวิจัยนี้จึงนำเสนอการแก้ปัญหาผู้ขายสินค้า ด้วย *COIN* โดยพัฒนาให้การทำงานเป็นแบบขนาน ประมวลผลบนทรัพยากรที่มีหน่วยประมวลผลเป็นแบบหลายแกน เพื่อเพิ่มประสิทธิภาพในการทำงานให้การได้มาซึ่งคำตอบได้รวดเร็วยิ่งขึ้น

วัตถุประสงค์ของการวิจัย

เพื่อต้องการลดเวลาในการหาคำตอบของปัญหาผู้ขายสินค้าที่กำหนดขึ้นโดยใช้ *Coincidence Algorithm* ในการหาคำตอบสำหรับการประมวลผลแบบขนานบนตัวประมวลผลหลายแกน

ขอบเขตของการวิจัย

พัฒนาโปรแกรมประยุกต์ ใช้ภาษา *C++* ร่วมกับชุดบริการ (*Library*) ของ *Intel* ชื่อ *Intel Threading Building Blocks 3.0* เพื่อทำงานแบบขนาน และประมวลผลบนตัวประมวลผลหลายแกน

เครื่องมือและทรัพยากรที่ใช้ ดังนี้

- Visual Studio 2010
- Intel Parallel Studio XE 2011
- Computer set
- CPU Quad Core
- RAM 8 GB

ปัญหาผู้ขายสินค้าที่นำมาทดสอบในงานวิจัย เป็นปัญหาที่กำหนดขึ้น ประกอบด้วย 5 ปัญหา โดยมีเมืองไม่เกิน 200 ผลการทดลองเปรียบเทียบกับการประมวลผลบนเครื่องคอมพิวเตอร์ส่วนบุคคลที่มีตัวประมวลผลแกนเดียว (*Single-core Processor*) ใช้โปรแกรมในการแก้ปัญหาเดียวกันแต่การทำงานเป็นแบบลำดับ โดยเปรียบเทียบการทำงานในแต่ละส่วนในการวัดผลด้านความเร็ว

ประโยชน์ที่คาดว่าจะได้รับ

ได้มีวิธีการเขียนโปรแกรมแบบขนานทำงานบนตัวประมวลผลแบบหลายแกน ที่สามารถแก้ปัญหาซับซ้อน

วิธีดำเนินการวิจัย

1. ศึกษาวิธีการทำงานของ *COIN*
2. ศึกษาวิธีการทำงานแบบขนานของ *Thread*
3. ศึกษาการเขียนโปรแกรมด้วยภาษา *C++* โดยใช้ชุดคำสั่งของ *Intel* ที่มีชื่อเรียกว่า *Intel® Threading Building Blocks*
4. พัฒนาโปรแกรมที่แก้ปัญหาผู้ขายสินค้าในงานวิจัยชิ้นนี้ เพื่อทดสอบและวัดผลในด้านความเร็ว เปรียบเทียบกับวิธีการประมวลผลแบบลำดับ
5. ทดลองการแก้ปัญหาและเปรียบเทียบผลที่ได้จากการทดลอง
6. สรุปผลและข้อเสนอแนะ
7. จัดทำวิทยานิพนธ์

โครงสร้างวิทยานิพนธ์

โครงสร้างวิทยานิพนธ์ฉบับนี้ประกอบด้วย 5 บทหลัก คือ บทนำ เอกสารและงานวิจัยที่เกี่ยวข้อง ระเบียบขั้นตอนวิธีที่เสนอ การทดลองและผล และสรุปผลการวิจัย อภิปรายผล

บทที่ 1 กล่าวถึงที่มาและความสำคัญของปัญหา วัตถุประสงค์ของงานวิจัย ขอบเขตและขั้นตอนในการทำวิจัย ผลประโยชน์ที่คาดว่าจะได้รับ โครงสร้างและผลงานที่วิทยานิพนธ์ได้ตีพิมพ์ บทที่ 2 กล่าวถึงทฤษฎีที่เกี่ยวข้องกับงานวิจัย และผลงานงานวิจัยที่เกี่ยวข้อง ในบทที่ 3 อธิบายถึงรายละเอียดของขั้นตอนและวิธีการของงานวิจัย บทที่ 4 อธิบายรายละเอียดการทดลอง และแสดงผลลัพธ์ที่ได้จากการทดลอง และบทสุดท้าย บทที่ 5 อภิปรายผลการทดลองและสรุปผล

ผลงานที่ตีพิมพ์จากวิทยานิพนธ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้รับการตอบรับให้ตีพิมพ์เป็นบทความทางวิชาการ ในหัวเรื่อง *“An Implementation of Coincidence Algorithm on Multi-core Processors”* โดย วีรชิต ศรีมุข และ ประภาส จงสถิย์วัฒนา ในงานประชุมวิชาการ *“International Conference on Computer and Information Technology (ICCIT'2012)”* ที่ประเทศไทย ระหว่างวันที่ 16-17 มิถุนายน พ.ศ. 2555

บทที่ 2

เอกสารและงานวิจัยที่เกี่ยวข้อง

ในบทนี้นำเสนอถึงเอกสารที่อ้างอิงในงานวิจัย รวมถึงทฤษฎีต่างๆ ที่นำมาใช้กับงานวิจัยชิ้นนี้ เพื่อให้เห็นถึงปัญหาและพื้นหลังของงานวิจัยและแนวทางสำหรับการแก้ปัญหา

2.1 แนวคิดและทฤษฎี

รายละเอียดที่สำคัญของหัวข้อนี้อธิบายถึงการทำงานของขั้นตอนวิธีอุบัติการณ์ร่วมกัน การทำงานแบบขนาน เครื่องมือที่ช่วยสำหรับการพัฒนาให้มีการทำงานแบบขนาน และสถาปัตยกรรมของตัวประมวลผลหลายแกน โดยมีรายละเอียดต่อไปนี้

2.1.1 ขั้นตอนวิธีอุบัติการณ์ร่วมกัน

ขั้นตอนวิธีอุบัติการณ์ร่วมกัน จัดอยู่ในกลุ่มของ *EA* ซึ่งเป็นการแก้ปัญหาในการหาคำตอบที่ดีที่สุด ตัวอย่างปัญหาในกลุ่มนี้ คือ *Traveling Salesman Problem (TSP), Minimum Spanning Tree Problem* เป็นต้น ขั้นตอนวิธี *COIN* ได้ถูกนำเสนอโดย วรินทร์ วัฒนพรพรหม [3] โดยนำค่าของตัวอย่างข้อมูลที่ไม่ดีและค่าตัวอย่างข้อมูลที่ดีมาใช้สำหรับการหาคำตอบให้กับปัญหา ซึ่งเป็นการนำความรู้เก่ามาใช้ในการสร้างประชากรสำหรับรุ่นต่อไป รูปแบบของ *COIN* เริ่มต้นที่ตารางข้อมูลความน่าจะเป็น โดยที่ค่าข้อมูลนี้ถูกใช้สำหรับการสร้างประชากร เรียกตารางนี้ว่า *Generator* โดยขนาดเท่ากับขนาดของปัญหาที่ต้องการหาคำตอบทั้งด้านกว้างและด้านยาว ข้อมูลในตารางนี้อยู่ในรูปของ *Markov Chain*

ตัวอย่างปัญหาที่หาคำตอบโดยใช้ *COIN* คือ ปัญหาผู้ขายสินค้า คำตอบของปัญหานี้คือเส้นทางการเดินทางที่สั้นที่สุดจากเส้นทางทั้งหมดที่เป็นไปได้ ให้สามารถเดินทางได้ครบทุกเมืองแล้ววนกลับมายังเมืองเริ่มต้น ข้อจำกัดคือสามารถผ่านเมืองหนึ่งได้แค่ครั้งเดียว ตัวอย่างเส้นทางของการเดินทางสำหรับปัญหาผู้ขายสินค้า ที่มีขนาด 10 เมือง ให้แต่ละเมืองถูกแทนด้วยตัวอักษรภาษาอังกฤษเป็น *A B C D E F G H I* และ *J* ตามลำดับ สมมุติเส้นทางที่ได้คือ

A-B-C-D-E-F-G-H-I-J-A

เมื่อแยกออกมาเป็นคู่ๆ ที่มีการเดินทาง จะได้ดังนี้ *A-B, B-C, C-D, D-E, E-F, F-G, G-H, H-I, I-J* และ *J-A* แต่ละคู่ที่จะเกิดในแต่ละเส้นทางถูกคัดเลือกจากค่าความน่าจะเป็นของแต่ละคู่จากตาราง *Generator* โดยที่ค่าความน่าจะเป็นอยู่ในช่วง 0.0 ถึง 1.0 ยกเว้นคู่เมืองตัวเองจะมีค่าเป็น 0.0 เสมอ

COIN มีลำดับในการทำงานแบ่งได้เป็น 6 ขั้น ดังนี้

1. กำหนดค่าเริ่มต้นของตาราง *Generator*
2. สร้างประชากรกลุ่มตัวอย่าง
3. ประเมินผลกลุ่มตัวอย่าง
4. เลือกข้อมูลกลุ่มตัวอย่าง
5. ปรับปรุงข้อมูลค่าความน่าจะเป็นในตาราง *Generator*
6. ประมวลผลซ้ำขั้นตอน 2-3-4-5 จนเข้าเงื่อนไขการหยุด

1. กำหนดค่าเริ่มต้นของตาราง Generator

ขบวนการแรกของขั้นตอนวิธีอุบัติการณ์ร่วมกัน คือขั้นตอนกำหนดค่าเริ่มต้นของตัวสร้างประชากร ค่าทุกค่าถูกกำหนดให้มีค่าเท่ากันทุกช่องข้อมูลคือ *1/(n-1)* โดยที่กำหนดให้ n คือขนาดของปัญหา ยกเว้นค่าข้อมูลที่จับคู่กับตัวเองให้มีค่าเป็นศูนย์เสมอ จากภาพที่ 2-1 เป็นตัวอย่างการกำหนดค่าเริ่มต้นของตัวสร้างประชากรที่มีขนาดปัญหา 5 ดังนั้นค่าข้อมูลเริ่มต้นเป็น 1/(5-1) มีค่าเท่ากับ 0.25

	1	2	3	4	5
1	0	0.25	0.25	0.25	0.25
2	0.25	0	0.25	0.25	0.25
3	0.25	0.25	0	0.25	0.25
4	0.25	0.25	0.25	0	0.25
5	0.25	0.25	0.25	0.25	0

ภาพที่ 2-1 กำหนดค่าเริ่มต้นของตัวสร้างประชากรขนาดปัญหาเท่ากับ 5

2. สร้างประชากรกลุ่มตัวอย่าง

ขั้นตอนนี้เป็นการค้นหาเส้นทางตามจำนวนที่กำหนด ในการค้นหาเส้นทางใช้หลักการสุ่มหาเมืองที่ต้องการตามค่าความน่าจะเป็นจนครบทุกเมือง เริ่มต้นที่การสุ่มเมืองเริ่มต้นในกรณีที่ไม่ได้กำหนดเมืองเริ่มต้นมาให้ เมืองแรกนี้แทนด้วย *x* ลำดับถัดไปคือหาว่าจากเมือง *x* จะเดินทางไปเมือง

ถัดไปควรจะเป็นเมืองใด หาได้จากการสุ่มเช่นกันแต่ในครั้งนี้ การสุ่มเมืองที่มีโอกาสเดินทางเป็นเมืองต่อไปสุ่มโดยดูค่าความน่าจะเป็นจากตารางข้อมูล *Generator* โดยเปรียบเทียบค่าตามแถวข้อมูลที่ตรงกับ *x* เมืองที่ได้ใหม่นี้ให้แทนด้วย *y* วนกระบวนการนี้จนได้ครบทุกเมืองโดยไม่มีการซ้ำกันของเมืองต่างๆ และทำจนได้เส้นทางครบตามข้อกำหนดถือเป็นประชากร 1 รุ่น

3. ประเมินผลกลุ่มตัวอย่าง

การหาค่าความเหมาะสมให้กับกลุ่มข้อมูลทุกค่าที่ได้จากขั้นตอนที่ 2 ค่าความเหมาะสม (*Fitness Value*) ขึ้นอยู่กับฟังก์ชันความเหมาะสม (*Fitness Function*) ปัญหาต่างประเภทกันฟังก์ชันในการหาก็อาจแตกต่างกันไป ในกรณีของปัญหาผู้ขายสินค้า ฟังก์ชันสำหรับการหาค่าความเหมาะสมคือ ฟังก์ชันหาระยะทางของแต่ละเส้นทาง ภาพที่ 2-2 แสดงการหาระยะทางรวมของแต่ละเส้นทาง โดยหาระยะทางระหว่างเมืองแต่ละเมืองที่มีการเดินทางแล้วนำค่าที่ได้รวมกัน ค่าที่ได้เป็นระยะทางรวมสำหรับเส้นทางนั้นๆ ข้อมูลเป็นการเดินทางของปัญหาผู้ขายสินค้า จำนวน 5 เมือง คือ *A B C D* และ *E* เส้นทางแรกคือ *A-B-C-D-E-A* ระยะทางของการเดินทางแต่ละคู่คือ A-B = 2, B-C = 1, C-D = 3, D-E = 1 และ E-A = 2 รวมระยะทางทั้งหมดเท่ากับ 9 หน่วย

A-B	B-C	C-D	D-E	E-A	Summary
2	1	3	1	2	9
A-E	E-D	D-B	B-C	C-A	
2	1	3	1	3	10
C-E	E-B	B-D	D-A	A-C	
5	4	3	4	3	19
D-A	A-C	C-B	B-E	E-D	
4	3	1	4	1	13
.					
.					
.					

ภาพที่ 2-2 การหาระยะทางรวมของแต่ละเส้นทาง

เมื่อประมวลผลหาค่าความเหมาะสมครบทุกเส้นทางแล้ว ลำดับถัดมาคือการเรียงข้อมูลตามค่าความเหมาะสม โดยเรียงจากน้อยไปหามาก เพื่อเป็นข้อมูลสำหรับใช้งานในขั้นตอนถัดไป

4. เลือกข้อมูลกลุ่มตัวอย่าง

การเลือกข้อมูลเพื่อนำไปใช้ในขั้นตอนปรับปรุงค่าความน่าจะเป็นในขั้นตอนถัดไปนั้น เลือกจากข้อมูลที่ได้จากขั้นตอนที่ 3 ที่มีการเรียงค่าข้อมูลแล้ว ข้อมูลจากการเลือกแบ่งออกเป็น 2 กลุ่ม คือ

กลุ่มตัวอย่างดี *(Better Group)* และ กลุ่มตัวอย่างเลว *(Worse Group)* วิธีการเลือกข้อมูลมีด้วยกัน 2 วิธีคือ

Uniform selection เป็นการเลือกโดยกำหนดจำนวนเปอร์เซ็นต์ เพื่อเลือกจากกลุ่มข้อมูลทั้งหมดโดยเลือกจากส่วนบน ให้เป็นกลุ่มตัวอย่างดีและส่วนล่างของข้อมูล ให้เป็นตัวอย่างเลว

Adaptive selection เป็นการเลือกข้อมูลจากกลุ่มที่มีค่ามากกว่าค่าเฉลี่ยและกลุ่มที่น้อยกว่าค่าเฉลี่ย ในกลุ่มของทั้งสองส่วนเบี่ยงเบนมาตรฐาน กล่าวคือหากจำนวนประชากรมีตัวอย่างดีมากกว่าตัวอย่างเลวแล้ว การเลือกจะเลือกเอาตัวอย่างเลวมามากกว่าจะเลือกเอาตัวอย่างดีมา ในทางกลับกันหากตัวอย่างเลวจากประชากรทั้งหมดมีมากกว่าตัวอย่างดี ก็จะเลือกตัวอย่างดีมากกว่าแทนที่จะเลือกเอาตัวอย่างที่ไม่ดีมา

ในการวิจัยครั้งนี้ ใช้หลักการเลือกแบบ *Uniform selection* โดยจำนวนเปอร์เซ็นต์ที่กำหนดเป็น 10 เปอร์เซ็นต์

5. ปรับปรุงข้อมูลค่าความน่าจะเป็นในตาราง *Generator*

ขั้นตอนนี้เป็นขั้นตอนที่สำคัญขั้นตอนหนึ่งเพราะค่าที่ได้หลังจากการปรับปรุงแล้วจะถูกนำไปใช้เพื่อสร้างประชากรในรุ่นถัดไป การปรับปรุงค่าความน่าจะเป็นมีด้วยกัน 2 แบบ คือ การปรับเพิ่มค่าความน่าจะเป็นและการปรับลดค่าความน่าจะเป็น เงื่อนไขของการปรับเป็นดังนี้ คู่ของ *xy* ใดๆ ที่เจอในกลุ่มตัวอย่างดีจะปรับค่าความน่าจะเป็นเพิ่มขึ้นหรือเรียกว่าการให้รางวัล ค่าในตำแหน่ง H_{xy} เพิ่มขึ้นโดยการรวบรวมค่าความน่าจะเป็นเท่ากับ $k/(n-1)^2$ จากตำแหน่ง H_{xz} อื่นๆ โดยที่ k คือจังหวะการเรียนรู้ n คือขนาดของปัญหา z มีค่าตั้งแต่ 1 ถึง n เงื่อนไขคือ $z \neq x$ และ $z \neq y$ และ r_{xy} คือจำนวนที่เจอคู่ xy ในกลุ่มตัวอย่างดี สูตรการให้รางวัล สามารถเขียนได้ดังนี้

$$H_{xy}(t+1) = H_{xy}(t) + \frac{k}{n-1}\left(r_{xy}(t+1)\right) - \frac{k}{(n-1)^2}\sum_{z=1}^{n} r_{xz}(t+1))$$

โดยที่ H_{xy} คือ ค่าความน่าจะเป็นที่ตำแหน่ง xy

k คือ ค่าจังหวะการเรียนรู้

n คือ ขนาดของปัญหา

r_{xy} คือ จำนวนคู่ของ xy ที่เจอในกลุ่มตัวอย่างดี

r_{xz} คือ จำนวนคู่ของ xz ที่เจอในกลุ่มตัวอย่างดี โดยที่ $z \neq x$ และ $z \neq y$

ในทางกลับกันส่วนของการลดค่าความน่าจะเป็นหรือการลงโทษนั้นเป็นการกระจายค่าความน่าจะเป็นของตัวเองไปยังตำแหน่งอื่นๆ ค่าที่กระจายเท่ากับ $k/(n-1)^2$ ในตำแหน่ง H_{xy} ใดๆ ให้กับสมาชิกในตำแหน่ง H_{xz} อื่นๆ โดยที่ $z \neq x$ และ $z \neq y$ เมื่อ k คือจังหวะการเรียนรู้ n คือขนาดของปัญหา p_{xy} คือจำนวนของ xy ที่เจอในกลุ่มตัวอย่างเลว สมการการลงโทษสำหรับข้อมูลที่ตำแหน่ง H_{xy} คือ

$$H_{xy}(t+1) = H_{xy}(t) - \frac{k}{n-1}\big(p_{xy}(t+1)\big) + \frac{k}{(n-1)^2}\sum_{z=1}^{n} p_{xz}(t+1))$$

กรณีที่สมาชิกคู่ใดๆที่เกิดขึ้นทั้งในกลุ่มตัวอย่างดีและกลุ่มตัวอย่างเลว การปรับค่าความน่าจะเป็นจะดูทั้งสองส่วนแล้วนำผลต่างมาคำนวณ เมื่อนำสูตรการปรับค่าทั้งสองรายการคือ การปรับเพิ่มค่าความน่าจะเป็นและการปรับลดค่าความน่าจะเป็น ได้สูตรการคำนวณสำหรับใช้ในกรณีที่คู่ xy ใดๆ เกิดขึ้นในทั้งสองกลุ่มตัวอย่างดังนี้

$$H_{xy}(t+1) = H_{xy}(t) + \frac{k}{n-1}\Big(r_{xy}(t+1) - p_{xy}(t+1)\Big) + \frac{k}{(n-1)^2}\Big(\sum_{z=1}^{n} p_{xz}(t+1) - \sum_{z=1}^{n} r_{xz}(t+1)\Big)$$

ในส่วนของ $r_{xy}(t+1) - p_{xy}(t+1)$ คือการคำนวณหาผลสุทธิของการให้รางวัลและการทำโทษของสมาชิกที่ตำแหน่ง xy จากทั้งสองกลุ่มตัวอย่าง และเทอมของ $\sum_{z=1}^{n} p_{xz}(t+1) - \sum_{z=1}^{n} r_{xz}(t+1)$ เป็นส่วนการปรับปรุงค่าความน่าจะเป็นสำหรับค่าข้อมูลในตำแหน่งอื่นๆ ของตารางค่าความน่าจะเป็น โดยที่ $z \neq x$ และ $z \neq y$

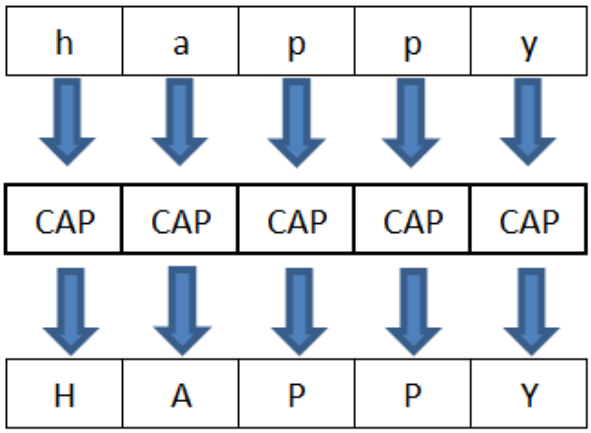
สำหรับการปรับค่าความน่าจะเป็นเมื่อทำการปรับค่าข้อมูลแถวใดๆแล้ว ผลรวมของค่าข้อมูลความน่าจะเป็นของแถวนั้นๆ ต้องยังคงค่าเดิมที่ 1.0 อยู่เสมอ ในขั้นตอนของการปรับปรุงค่าความน่าจะเป็นมีเงื่อนไขดังนี้

1. ห้ามค่าข้อมูลติดลบ
2. กรณีปรับลดค่าความน่าจะเป็นไม่อนุญาตให้ค่าข้อมูลน้อยกว่า 1 ใน 10 ของค่าเริ่มต้น

3. กรณีปรับเพิ่มค่าความน่าจะเป็นไม่อนุญาตให้ค่าข้อมูลมากกว่า 8 ใน 10 ของค่าข้อมูลรวม

2.1.1.1 COIN กับ Intel® TBB

การประมวลผลแบบขนานโดยใช้ *COIN* แก้ปัญหาผู้ขายสินค้า ในการพัฒนาโปรแกรมโดยใช้ *C++* ร่วมกับ *Intel TBB* เพื่อให้การทำงานคล่องตัวและรวดเร็วยิ่งขึ้น ได้ลดขั้นตอนของการจัดการเรื่องเธรดและการทำ *Load Balancing* ให้เครื่องมือที่ใช้เป็นตัวจัดการแทน ซึ่ง *Intel TBB* มีคุณสมบัติดังกล่าวครบถ้วน ผู้พัฒนามองการออกแบบให้โปรแกรมแก้ปัญหาที่กำหนดทำงานแบบขนานส่วนใดบ้าง ภาพที่ง่ายที่สุดคือการทำงานแบบขนานบนข้อมูล (*Data parallelism*) ทำงานเดียวกันบนข้อมูลที่สามารถทำไปพร้อมกันได้ ภาพที่ 2-3 เปลี่ยนตัวอักษรจากตัวพิมพ์เล็กให้เป็นตัวพิมพ์ใหญ่ โดยที่แต่ละตัวอักษรสามารถประมวลผลได้อย่างอิสระ หน้าที่ของผู้พัฒนาคือแบ่งการทำงานนี้ออกเป็นชิ้นย่อยๆ อย่างไร แต่ละชิ้นย่อยควรจะมีขนาดเท่าไหร่ ส่วนหลังจากนี้ *Intel TBB* จัดการให้โดยอัตโนมัติ โดยทำการจับคู่งานย่อยจากการแบ่งไปยังเธรดต่างๆของตัวประมวลผล



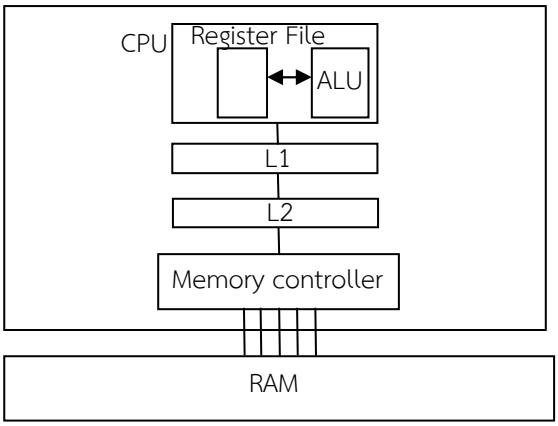
ภาพที่ 2-3 เปลี่ยนตัวพิมพ์เล็กเป็นตัวพิมพ์ใหญ่

2.1.2 ตัวประมวลผลหลายแกน

ในยุคปัจจุบันนี้หากกล่าวถึงความก้าวหน้าทางเทคโนโลยีแล้วนั้น สามารถกล่าวได้ว่ามีการพัฒนารุดหน้าแบบก้าวกระโดด เช่นเดียวกันกับเทคโนโลยีทางด้านตัวประมวลผลกลาง ตามท้องตลาดตัวประมวลผลกลางเป็นแบบหลายแกนแทบทั้งสิ้น โดยสามารถเห็นได้ทั่วไปเช่น 2 แกน 3 แกน หรือ 4 แกน ทางด้านบริษัทผู้ผลิตตัวประมวลผลเช่น *Intel* มีงานวิจัยว่าต่อไปในอนาคตอันใกล้นี้จะเข้าสู่

ยุคที่เรียกว่า *Tera-Scale Computing* [4] กล่าวคือเครื่องคอมพิวเตอร์ระดับพื้นฐานจะใช้ตัวประมวลผลกลางแบบหลายแกนตั้งแต่ 10 - 100 แกน

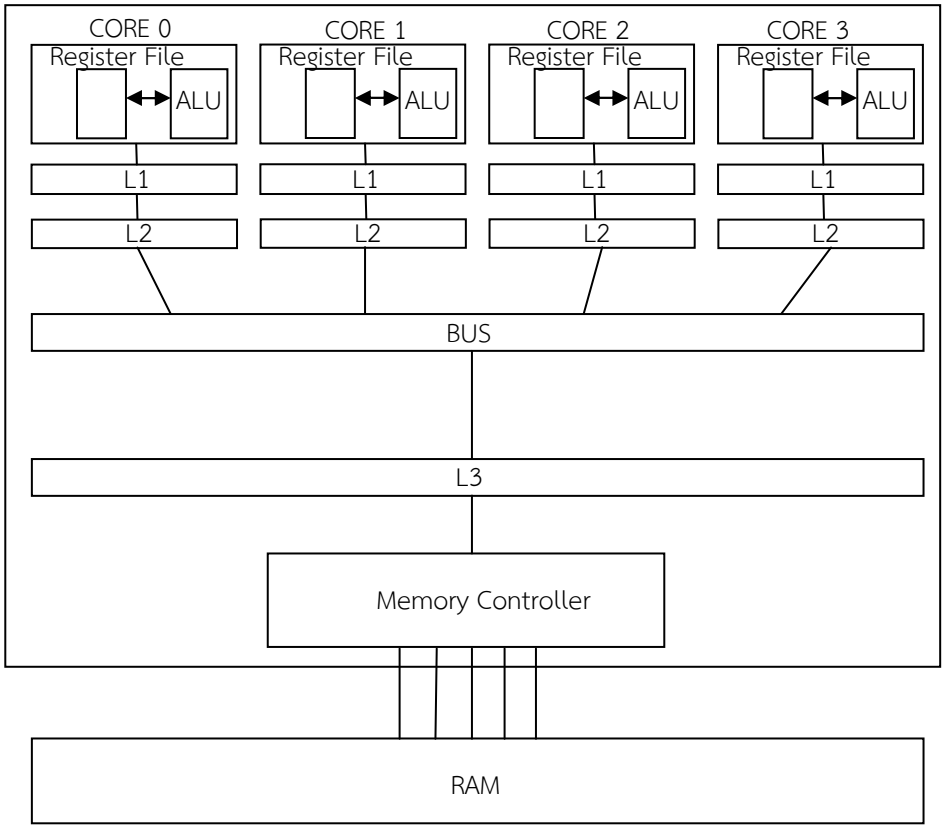
หากเปรียบเทียบโครงสร้างของตัวประมวลผลกลางแบบหลายแกนกับตัวประมวลผลแบบแกนเดียว ความแตกต่างสามารถเห็นได้ชัดเจน ภาพที่ 2-4 เป็นโครงสร้างของตัวประมวลผลแบบแกนเดียวจะมี *ALU* และ *Register* ชุดเดียว ในขณะที่โครงสร้างของตัวประมวลผลแบบหลายแกน ภายใต้ตัวประมวลผลจะแบ่งออกเป็น แกนซึ่งแต่ละแกนจะมี *ALU* และ *Register* เป็นของตัวเองสามารถทำงานได้อย่างอิสระแยกต่อกัน ดังแสดงในภาพที่ 2-5



ภาพที่ 2-4 โครงสร้างตัวประมวลผลแบบแกนเดียว

เหตุผลว่าทำไมจึงต้องเป็นตัวประมวลผลหลายแกน เนื่องจากว่าในปัจจุบันผู้ผลิตตัวประมวลผลจะทำการเพิ่มแกนการทำงานมากขึ้น แทนที่จะปรับทางด้ายความเร็วของสัญญาณนาฬิกาเหมือนดังเช่นอดีตก่อนหน้านี้ ดังนี้

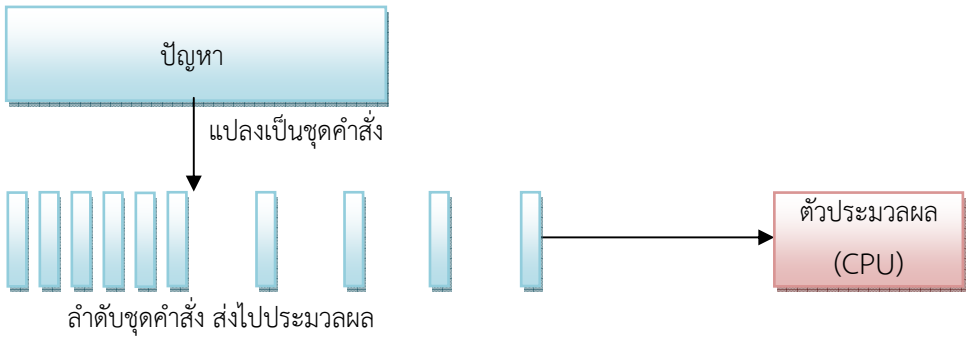
- มีความยุ่งยากมากขึ้นสำหรับการเพิ่มความเร็วสัญญาณนาฬิกาของตัวประมวลผลแบบแกนเดียว
- โปรแกรมต่างๆในปัจจุบันก็จะเน้นไปทางทำงานแบบขนาน หรือทำงานหลายๆงานพร้อมกัน (*Multithreaded*)
- แนวโน้มการการพัฒนาสถาปัตยกรรมของเครื่องคอมพิวเตอร์เน้นการทำงานแบบขนาน



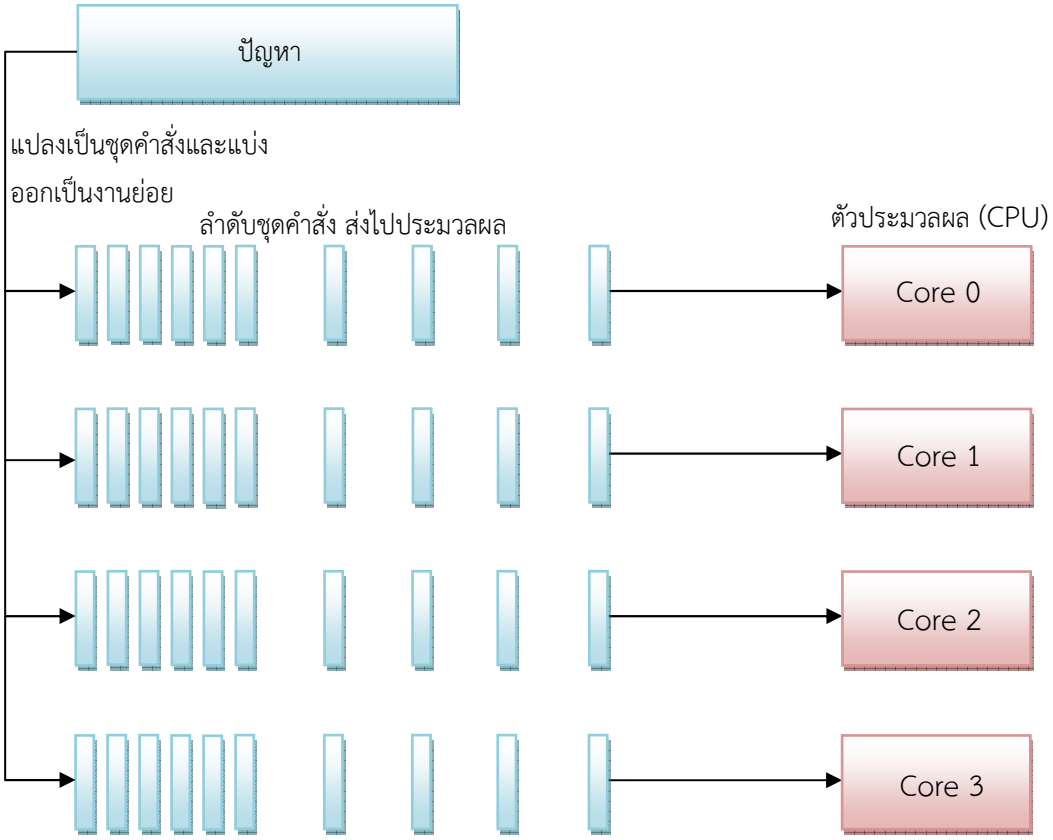
ภาพที่ 2-5 โครงสร้างตัวประมวลผลแบบหลายแกน

2.1.3 การโปรแกรมแบบขนาน

โดยทั่วไปแล้วโปรแกรมต่างๆ จะถูกเขียนมาให้ประมวลผลแบบลำดับ ซึ่งคำสั่งจะถูกประมวลผลบนเครื่องคอมพิวเตอร์เครื่องเดียวที่มีหน่วยประมวลผลกลางแบบแกนเดียว โดยปัญหาจะถูกแบ่งเป็นชุดคำสั่งต่อๆกันไป และประมวลผลตามลำดับคำสั่งที่เข้ามา การคำนวณแบบขนานคือแก้ปัญหาให้สามารถทำงานพร้อมกันบนทรัพยากรที่ใช้คำนวณ โดยการทำงานจะมีหน่วยประมวลผลกลางหลายตัวหรือหลายแกน ปัญหาจะถูกแบ่งออกเป็นส่วนย่อยๆ โดยปัญหานั้นสามารถแก้ควบคู่กันไปได้ ซึ่งปัญหาจากแต่ละส่วนถูกส่งไปประมวลผลพร้อมๆกันยังหน่วยประมวลผลกลางแตกต่างกันไป ภาพที่ 2-6 แสดงการทำงานการแก้ปัญหาในแบบลำดับ และภาพที่ 2-7 แสดงการทำงานสำหรับการแก้ปัญหาที่มีการแบ่งงานออกเป็นงานย่อย โดยที่ส่งชุดคำสั่งย่อยไปยังตัวประมวลผลที่แตกต่างกันพร้อมกัน เป็นการทำงานแบบขนาน



ภาพที่ 2-6 การประมวลคำสั่งการแก้ปัญหาแบบลำดับ



ภาพที่ 2-7 การประมวลคำสั่งการแก้ปัญหาแบบขนาน

โดยที่ทรัพยากรการคำนวณ อาจเป็นได้ดังนี้

- เครื่องคอมพิวเตอร์เครื่องเดียวแต่มีหน่วยประมวลผลกลางหลายตัว
- จำนวนของคอมพิวเตอร์ที่เชื่อมต่อกันด้วยระบบเครือข่าย

- ทั้งสองแบบรวมกัน

ปัญหาการคำนวณ สามารถที่จะกระทำได้ดังนี้

- สามารถแบ่งปัญหาออกเป็นส่วนย่อยๆได้ แล้วสามารถแก้ปัญหาได้พร้อมกัน
- สามารถประมวลผลคำสั่งโปรแกรมหลายๆคำสั่งในขณะใดขณะหนึ่งได้
- ในการแก้ปัญหาด้วยการใช้แหล่งประมวลผลหลายตัวเปรียบเทียบกับแหล่งประมวลผลเครื่องเดียวต้องใช้เวลาในการแก้ปัญหาที่น้อยกว่า

วิธีการในการเขียนโปรแกรมเพื่อให้ทำงานแบบขนาน มีหลายวิธีดังนี้

- Shared memory (without threads)
- Threads
- Distributed memory / message passing
- Data parallel
- Hybrid
- Single program multiple data (SPMD)
- Multiple program multiple data (MPMD)

การทำงานแบบขนาน มีการใช้เทคนิคได้หลายแบบ โดยที่ตัวแบบเธรด (*Threads model*) ก็เป็นอีกวิธีที่ใช้กันอย่างแพร่หลาย [8] การเขียนโปรแกรมในรูปแบบนี้ เป็นการเขียนโปรแกรมให้มีใช้งานหน่วยความจำร่วมกัน (*Shared memory*) รูปแบบของการแบ่งออกเป็นเธรด (*Thread*) เป็นการเขียนโปรแกรมให้ทำงานแบบขนาน โดยการทำงานเดียวกันในขณะการประมวลผลแบ่งการคำนวณเป็นหลายๆส่วน

ภาษาระดับสูงมีการพัฒนาให้รองรับสำหรับการเขียนโปรแกรม เพื่อให้มีการทำงานแบบขนาน ทำออกมาในรูปตัวตัวบริการ (*Library*) เช่น *OpenMP* หรือ *Intel TBB* โดย *OpenMp* พัฒนาโดย *OpenMP ARB* สามารถใช้ในการเขียนร่วมกับภาษา *C, C++* และ *Fortran* ในขณะที่ *Intel Threading Building Block (Intel TBB)* ออกแบบและพัฒนาโดยบริษัท *Intel* สามารถใช้เขียนร่วมกับ *C, C++* ทั้ง 2 วิธีที่ยกตัวอย่างมา มีการใช้งานและวิธีการเขียนต่างกันออกไป ในงานวิจัยชิ้นนี้ใช้ *C++* ร่วมกับ *Intel TBB*

เหตุผลสำหรับการเลือกใช้ *Intel TBB* เนื่องจากต้องการพัฒนาระบบด้วย *C++* และหากพิจารณาถึงคุณสมบัติเมื่อเปรียบเทียบกับชนิดอื่นแล้ว ผลที่ได้จากการเปรียบเทียบดังตารางที่ 2-1 คือมีคุณสมบัติโดดเด่นกว่าแบบอื่น

Capabilities	Intel® TBB	OpenMP	Threads
Task level parallelism	+	+	-
Data decomposition support	+	+	-
Complex parallel patterns (non-loops)	+	-	-
Broadly applicable generic parallel patterns	+	-	-
Scalable nested parallelism support	+	-	-
Built-in load balancing	+	+	-
Affinity support	-	+	+
Static scheduling	-	+	-
Concurrent data structures	+	-	-
Scalable memory allocator	+	-	-
I/O dominated tasks	-	-	+
User-level synchronization primitives	+	+	-
Compiler support is not required	+	-	+
Cross OS support	+	+	-

ตารางที่ 2-1 เปรียบเทียบด้านความสามารถ Intel TBB, OpenMP และ Threads โดยใช้ร่วมกับภาษา C++ [5]

ข้อดีของ *Intel TBB* แยกเป็นเรื่องๆ ดังนี้

1. มองให้อยู่ในรูปของงานย่อย (*Tasks*) แทนที่จะเป็นเธรด (*Threads*) นั่นคือ *TBB* จัดตารางเวลาให้กับงานย่อย ไปทำงานแบบเธรดแบบอัตโนมัติ โดยที่สามารถใช้ประโยชน์จากหน่วยประมวลผลกลางได้อย่างมีประสิทธิภาพ

2. เป้าหมายของ *Intel TBB* คือให้การทำงานแบบเธรดเพื่อประสิทธิภาพ (*Threading for performance*) โดยที่ *Intel TBB* มีเป้าหมายที่ชัดเจนคือ การคำนวณแบบขนานของงานหนักๆ แต่อยู่ในรูปของภาษาระดับสูงที่เรียบง่าย
3. *Intel TBB* สามารถเข้ากันได้กับการเขียนแบบเธรด ในแบบอื่นๆ ถือเป็นส่วนสำคัญมาก เพราะไม่ได้บังคับให้เลือกการใช้การเขียนเธรด แบบเดียวกันทั้งโปรแกรม โดยสามารถที่จะเพิ่มเติมการใช้งาน *Intel TBB* เข้าไปยังโปรแกรมที่มีการทำงานเธรดแบบอื่นๆ อยู่แล้วได้โดยอิสระ
4. *Intel TBB* เน้นการขยายขีดความสามารถของการเขียนโปรแกรม ที่มีการทำงานขนานในส่วนของข้อมูล (*Data-parallel programming*)

เมื่อเปรียบเทียบระหว่าง *Intel TBB, OpenMP* และ *Native Threads* ทั้ง 3 แบบนี้ หากชุดคำสั่งเขียนโดยใช้ภาษา *C++* ส่งผลให้ *Intel TBB* มีความเหมาะสมที่สุด [5]

2.1.3.1 Intel® Threading Building Blocks (Intel® TBB)

Intel® TBB เป็นเครื่องมือที่ใช้เขียนโปรแกรมให้รองรับการทำงานแบบขนาน พัฒนาโดยบริษัท *Intel* สามารถรองรับได้หลายระบบปฏิบัติการ ไม่ว่าจะเป็นระบบ *Windows*, *Linux* หรือ *Mac OS* ลักษณะการทำงานแบบขนานของ *Intel TBB* จะเป็นในแบบของงาน (Task Base) โดยการมองปัญหาออกเป็นงานย่อยที่สามารถประมวลผลไปพร้อมๆกันได้ คุณสมบัติเด่นอย่างหนึ่งของ *Intel TBB* ที่สามารถให้การทำงานนั้นเป็นไปอย่างมีประสิทธิภาพคือ การทำงานแบบขโมยกำหนดการ (Work-stealing task scheduler) การพัฒนาระบบโดยใช้ *Intel TBB* ต้องเขียนร่วมกับ *C++*

ส่วนประกอบหลักของ *Intel TBB* มีดังนี้

- *Generic parallel algorithm* คือ อัลกอริทึมทั่วไปสำหรับให้ทำงานแบบขนาน
- *Task scheduler* คือ ตัวจัดการ *Task* กับ *Thread*
- *Synchronization primitives* คือ ชุดการทำงานที่ประสานกันแบบดั้งเดิมกับคุณภาพที่ต่างกันซึ่งสามารถใช้ร่วมกันกับกลยุทธ์การประสามนร่วมกัน
- *Threads* คือ การทำงานเกี่ยวกับเธรด
- *Concurrent containers* คือ ทางเลือกที่สามารถปรับขนาดได้สำหรับเก็บข้อมูลแบบลำดับ

- *Thread local storage* คือ ตัวจัดการเกี่ยวกับการใช้เนื้อที่หน่วยความจำของเธรดแบบ *Local*
- *Memory allocation* คือ การจัดสรรหน่วยความจำเพื่อหลีกเลี่ยงปัญหาคอขวดโดยลดการเข้าถึงหน่วยความจำที่ใช้ร่วมกัน
- *Miscellaneous* คือ อื่นๆ เช่น ตัวนับเวลา

จากส่วนประกอบหลักแต่ละส่วน จะมีรายละเอียดย่อยดังแสดงในภาพที่ 2-8 ซึ่งบอกถึงฟังก์ชันย่อยหรือส่วนการติดต่อ (*Interface*)

Intel® Threading Building Blocks v3.0 (Intel® TBB v3.0)		
Generic Parallel Algorithms	**Concurrent Containers**	**Task Scheduler**
parallel_for(range)	concurrent_hash_map	task_group
parallel_reduce	concurrent_queue	structured_task_group
parallel_for_each(begin, end)	concurrent_bounded_queue	task_scheduler_init
parallel_do	concurrent_vector	task_scheduler_observer
parallel_invoke	concurrent_unordered_map	
pipeline		
parallel_pipeline		
parallel_sort		
parallel_scan		
Synchronization Primitives		**Memory Allocation**
atomic	queuing_rw_mutex	tbb_allocator
mutex	reader_writer_lock	cache_aligned_allocator
recursive_mutex	critical_section	scalable_allocator
spin_mutex	condition_variable	zero_allocator
spin_rw_mutex	null_mutex	
queuing_mutex	null_rw_mutex	
Thread Local Storage	**Threads**	**Miscellaneous**
enumerable_thread_specific	thread	tick_count
combinable		captured_exception
		moveable_exception

ภาพที่ 2-8 ส่วนประกอบต่างๆของ Intel TBB รุ่นที่ 3.0

ขั้นตอนวิธีแบบขนานทั่วไป

ในส่วนรายละเอียดตามภาพที่ 2-8 มีฟังก์ชันการใช้งานสำหรับการทำงานแบบขนาน ไม่ว่าจะเป็นการจัดการ งานชิ้นต่อชิ้น หรือการลดทอน หรือการตรวจสอบตามโครงสร้าง สามารถดำเนินการ

โดยใช้ตัวบริการของ *Intel TBB* คือ *parallel_for*, *parallel_reduce* และ *parallel_scan* ตามลำดับ การใช้งานฟังก์ชันดังกล่าวผู้ใช้ต้องมีการกำหนดค่าพารามิเตอร์ที่ต้องใช้สำหรับการเรียกใช้ฟังก์ชันดังกล่าว ค่าที่ต้องกำหนดมี 2 ค่าคือ ช่วงของข้อมูล (*Range*) และ ตัวโปรแกรมที่ปฏิบัติการ (*Body*) สำหรับข้อมูลที่อยู่ในช่วงที่กำหนด

สำหรับการจัดการกับการหยุดประมวลผลของการวนซ้ำ ผู้ใช้งานสามารถใช้ตัวจัดการแบ่ง (*partitioners*) มีทั้งหมด 3 แบบด้วยกัน *simple_parttitioner*, *auto_partitioner* และ *affinity_partitioner* ทั้งสามแบบนี้มีความต่างกันไปกล่าวคือ *simple_partitioner* การวนซ้ำแบ่งออกเป็นงานย่อยก็ต่อเมื่อ ขนาดของข้อมูลมีค่ามากกว่าค่าเริ่มต้นที่กำหนด ในการแบ่งงาน *auto_partitioner* โดยค่านี้เป็นค่าเริ่มต้นหากไม่มีการระบุ ทำหน้าที่เป็นตัวตรวจสอบถึงพฤติกรรมการขโมย หากการขโมยไม่มีการเกิดขึ้นแบ่งช่วงข้อมูลออกเป็นส่วนย่อยๆเท่ากับ $p \times 4$ ช่วง โดยที่ p คือจำนวนเธรดของอุปกรณ์ (*Hardware*) และ *affinity_partitioner* เป็นการเก็บประวัติของการทำงานเธรดต่างๆ ที่ประมวลผลก่อนหน้านี้ จะพยายามกระจายงานในการประมวลผลสำหรับครั้งต่อๆไป

ตัวอย่างการใช้งานของ *parallel_for* โดยนำตัวอย่างการหาค่าเฉลี่ยในบทที่ 3 จาก [9] แสดงถึงการเขียนโปรแกรมในรูปแบบขนาน ใช้ *Intel TBB* แสดงในภาพที่ 2-9

```
1  #include "tbb/parallel_for.h"
2  #include "tbb/blocked_range.h"
3  using namespace tbb;
4  struct Average {
5     float* input;
6     float* output;
7     void operator( )( const blocked_range<int>& range ) const {
8        for( int i=range.begin(); i!=range.end( ); ++i )
9           output[i] = (input[i-1]+input[i]+input[i+1])*(1/3.0f);
10       }
11 };
12 // Note: The input must be padded such that input[-1] and input[n]
13 // can be used to calculate the first and last output values.
14 void ParallelAverage( float* output, float* input, size_t n ) {
15    Average avg;
16    avg.input = input;
17    avg.output = output;
18    parallel_for( blocked_range<int>( 0, n, 1000 ), avg );
19 }
```

ภาพที่ 2-9 การเขียนโปรแกรมแก้ไขปัญหาการหาค่าเฉลี่ยโดยใช้ Intel TBB

จากภาพที่ 2-9 บรรทัดที่ 18 เป็นการเรียกให้มีการหาค่าเฉลี่ยแบบขนาน โดยไม่มีการระบุตัวจัดการแบ่งงาน โครงสร้างของ *parallel_for* คือ *parallel_for(range, body, partitioner)* ส่วนของ *range* เป็นการระบุช่วงของข้อมูลมีค่าตัวแปร 3 ค่า *range(begin, end, subrange)* ซึ่ง *begin* เป็นจุดเริ่มต้นของข้อมูล *end* เป็นค่าสิ้นสุดของข้อมูล และ *subrange* เป็นค่าเริ่มต้นที่ต้องแบ่งการทำงานเมื่อจำนวนข้อมูลทั้งหมดมากกว่าค่าที่กำหนด ค่าข้อมูลตัวที่สอง *body* เป็นส่วนที่เรียกเพื่อให้ทำงาน ในส่วนของ *body* เป็นวัตถุ โดยที่กระบวนการ (Method) ชื่อ *operator()* ถูกเรียกเพื่อประมวลผลในแต่ละช่วงย่อยของข้อมูล จากตัวอย่างอยู่บรรทัดที่ 7 ถึง 10 เมื่อพิจารณาจากบรรทัดที่ 8 เป็น *for loop* ธรรมดา ที่มี *range.begin()* คือค่าเริ่มต้นของช่วงย่อยนี้ *range.end()* คือค่าสุดท้ายของช่วงย่อยนี้

รายละเอียดเพิ่มเติมสำหรับการใช้งานภาษา *C++* ร่วมกับตัวบริการ *Intel TBB* สามารถอ่านเพิ่มเติมได้จาก [5], [9]

2.2 เอกสารและงานวิจัยที่เกี่ยวข้อง

Hongzhong Shan เสนองานวิจัยเรื่อง *Hybrid Programming for Multicore Processors* [10] โดยแก้ปัญหา 3 ปัญหาคือ *Lower-Upper Symmetric Gauss-Siedal (LU), Scalar Penta-diagonal (SP)*, และ *Block Tri-diagonal (BT)* ในการทำ *Hybrid programming* ใช้ *OpenMP* ในการทำงานภายในโหนด (*Node*) และใช้ *MPI* สำหรับการทำงานระหว่างโหนด การทดลองซึ่งใช้ *MPI* คู่กับ *OpenMP* ในสัดส่วนที่ต่างกันเพื่อวัดผลในเรื่องของ การใช้หน่วยความจำและประสิทธิภาพของการทำงาน

ผลการประเมินผลทางด้านการใช้หน่วยความจำ ปรากฏว่าจำนวนกระบวนการ (*Process*) ของ *MPI* จะแปรผันตามกับจำนวนการใช้หน่วยความจำ กล่าวคือเมื่อจำนวน *MPI Process* เพิ่มขึ้นแนวโน้มการใช้หน่วยความจำก็เพิ่มขึ้นด้วย จำนวนกระบวนการของ *MPI* ที่ใช้หน่วยความจำน้อยสุดจะอยู่ที่ 2 กระบวนการ และจะใช้หน่วยความจำสูงสุดเมื่อมีจำนวนกระบวนการของ *MPI* เป็น 16 กระบวนการ ซึ่งมี *OpenMP* เพียงแค่ 1 เธรด โดยที่ทั้ง *BT* และ *SP* ให้ผลลัพธ์ที่คล้ายกันคือ มีประสิทธิภาพสูงสุดเมื่อมีจำนวน *OpenMP Thread* เป็น 2 เธรด

ความยุ่งยากสามารถเกิดขึ้นได้ง่าย ในการจัดการเกี่ยวกับการทำงานแบบขนานที่ซ้อนกันและการใช้ทรัพยากรร่วมกัน เช่น ในหลายกรณีที่มีการใช้งาน *OpenMP* จะมีการสร้างเธรดใหม่เพิ่มขึ้นสำหรับแต่ละส่วนของการทำงานแบบขนานที่ซ้อนกัน

Shenshen Liang และคณะ เสนองานวิจัยเรื่อง *Parallel K-Nearest Neighbor Algorithm on CUDA-enabled GPU (CUKNN)* [11] โดยที่เลือกการทำงาน 2 รูปแบบ ที่ให้ไป

คำนวณที่หน่วยประมวลผลกราฟฟิกส์ (*GPU*) คือ การคำนวณระยะห่างและการเลือกเพื่อนบ้านใกล้ที่สุดจำนวน *k* ตัว ในการคำนวณหาระยะห่างแยกการคำนวณได้โดยอิสระในแต่ละคู่ ซึ่งในการทำงานแบบขนานใช้การทำ *Data-parallelism* ส่วนขั้นตอนการเลือกเพื่อนบ้านใกล้ *k* ตัว เมื่อคำนวณหาระยะห่างของแต่ละเธรด เรียบร้อยแล้ว จะได้ข้อมูลเป็นของแต่ละส่วน เรียกว่า *Local-k Nearest Neighbors* โดยที่ค่าระยะห่างจะถูกเรียงเรียบร้อยแล้ว และมีอีก 1 เธรดเพื่อค้นหาเพื่อนบ้านที่ใกล้ที่สุดจากทั้งหมด คือ *Global-k Nearest Neighbors* จาก ทุกๆ *Local-k Neighbors*

ผลการทดลอง ปรากฏว่าสามารถลดเวลาในการประมวลผลได้ 46.71% ในการทำชุดข้อมูลสังเคราะห์ และ 42.49% ในการทำชุดแบบจำลองข้อมูลกายภาพรวมถึงการใช้ส่วนการติดต่อทั้งการรับเข้าและส่งออกด้วย แต่ในการใช้งาน *CUDA* นั้นผู้ใช้ต้องมีความรู้เรื่องเธรดเป็นอย่างดี เพราะต้องบริหารจัดการเองทั้งหมดในทุกๆระดับ เช่นในเรื่องของการแบ่ง *Block, Thread* รวมถึงการทำสำเนาข้อมูลระหว่างหน่วยความจำหลักกับหน่วยความจำภายในของ *GPU*

Salman Yussof และคณะ ได้นำเสนอการแก้ปัญหา *Shortest Path Routing* โดยใช้ *Parallel Genetic Algorithm (PGA)* [12] เปรียบเทียบกับ *Non-parallel* โดยดำเนินการบน *MPI Cluster* ซึ่งในงานวิจัยนี้เลือกใช้ *Coarse-grained GA* เพราะมีใช้การติดต่อสื่อสารน้อยที่สุดเมื่อเทียบกับ PGA ประเภทอื่นๆ ทำงานบนเครื่องคอมพิวเตอร์ 6 เครื่อง แต่ละเครื่องใช้ *Dual-core CPU* สามารถทำงานได้พร้อมกันถึง 12 โหนด ในแต่ละโหนดจะสร้างประชากรย่อยของตัวเองและจะประมวลผล *GA* กับประชากรของตัวเอง แต่มีการกำหนดให้มีอีก 1 โหนด ทำหน้าที่รวบรวมผลลัพธ์จากทุกโหนดและเลือกผลลัพธ์ที่ดีที่สุดออกมา

ผลการทดลองสรุปว่าเวลาที่ใช้ในการคำนวณขึ้นอยู่กับขนาดของประชากรและจำนวนของโหนดที่ใช้ในการคำนวณ โดยเมื่อจำนวนโหนดในการคำนวณมากขึ้น ส่งผลให้ความถูกต้องแม่นยำลดลงแบบเส้นตรง ขณะที่เวลาที่ใช้คำนวณลดลงเช่นกันแต่ลดลงแบบยกกำลัง ในขณะที่เมื่อเพิ่มจำนวนของประชากรให้มากขึ้น ความถูกต้องแม่นยำจะเพิ่มขึ้นด้วย แต่เวลาที่ใช้ประมวลผลก็เพิ่มมากตามขึ้นด้วย อย่างไรก็ดียังสามารถแก้ได้โดยเพิ่มจำนวนโหนดการคำนวณให้มากขึ้น และหากเทียบกับการคำนวนแบบลำดับสามารถลดเวลาในการคำนวณได้อย่างมาก

Ami Marowka ได้นำเสนอการศึกษาการเขียนโปรแกรมและประสิทธิภาพ การทำงานของ *Intel Threading Building Blocks* [13] โดยได้พัฒนาโปรแกรมขึ้นมาเพื่อเปรียบเทียบกับอีก 2 วิธี นั่นคือ *POSIX Pthreads* และ *OpenMP* เพื่อดูความยากง่ายของการเขียนโปรแกรม โดยปัญหาที่กำหนดขึ้นคือ การหาค่าของพาย (π) และได้ทดสอบประสิทธิภาพของ *Intel TBB* โดยใช้โปรแกรมอีกชุดเพื่อทำการทดสอบ โดยผลการทดลองสรุปได้ดังนี้ ในส่วนของการเขียนโปรแกรม โดย *Pthreads* นั้น ตัวผู้พัฒนาเองต้องจัดการเกี่ยวกับเธรดเองทั้งหมด และต้องชัดเจนในการระบุงานให้กับเธรดและจะให้มีกี่เธรด ส่วนการใช้ *OpenMP* นั้นไม่ต้องระบุงานให้กับเธรดอย่างชัดแจ้งแต่ต้อง

ระบุว่า จะให้มีการทำงานกี่เธรด ในขณะที่ *Intel TBB* ไม่ต้องมีการระบุงานให้กับเธรดเอง และไม่ต้องระบุว่าจะให้มีการทำงานกี่เธรด โดยในส่วนนี้ *Intel TBB* จะจัดการให้ทั้งหมดเองโดยอัตโนมัติ

ผลการทดลองด้านประสิทธิภาพโดยทดลองบนเครื่อง *Pentium D, Core 2 Duo* และ *Core 2 Quad* โดยใช้เวลาในการประมวลผลเป็นตัวชี้วัด สำหรับ 1, 2 และ 4 เธรด ปรากฏว่าได้ผลการทำงานที่เร็วขึ้น เมื่อจำนวนแกนและเธรดเพิ่มขึ้น แต่มีบางปัญหาที่ทำงานบน *Core 2 Quad* และจำนวนเธรดเป็น 4 เธรด ผลลัพธ์ที่ได้ไม่ดีขึ้น

ผู้วิจัยได้สรุปว่ามีประสิทธิภาพ ถึงแม้ว่าในการเขียนแบบขนานโดยใช้ *C++* มีความยุ่งยากมากกว่าการพัฒนาโดยใช้ *C* และ *OpenMP*

บทที่ 3

ระเบียบขั้นตอนวิธีที่เสนอ

ระเบียบขั้นตอนวิธี

เวลาในการประมวลผลสำหรับการแก้ปัญหาผู้ขายสินค้า โดยใช้ *COIN* ใช้เวลาในการคำนวณมาก งานวิจัยนี้นำเสนอการแก้ปัญหาโดยให้ทำงานแบบขนานประมวลผลบนตัวประมวลผลหลายแกน พัฒนาโดยใช้ *C++* ร่วมกับ *Library* จาก *Intel* ชื่อว่า *Intel Threading Building Blocks* กระบวนการที่กระจายงานให้ทำแบบขนานมีดังนี้

1. ขั้นตอนการสร้างประชากร
2. ขั้นตอนการหาค่าความเหมาะสม
3. ขั้นตอนการปรับปรุงข้อมูลความน่าจะเป็น

3.1 ขั้นตอนการสร้างประชากร

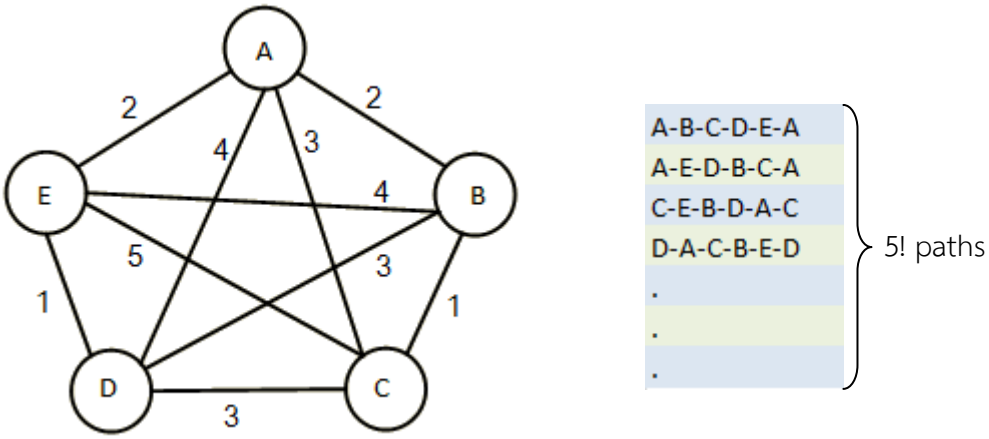
เป็นขั้นตอนที่หาเส้นทางการเดินทางจากเมืองเริ่มต้นไปยังทุกเมืองและวนกลับมายังเมืองที่เริ่มต้นโดยแต่ละเมืองผ่านได้เพียงครั้งเดียวถือเป็นเส้นทางที่สมบูรณ์ ในขั้นตอนนี้หาเส้นทางตามจำนวนเส้นทางที่กำหนดนับเป็นจำนวนประชากร 1 รุ่น ซึ่งสามารถหาเส้นทางแบบขนานได้เพราะในแต่ละเส้นทางเป็นอิสระจากกัน

ขั้นตอนการหาเส้นทาง เริ่มต้นทำการสุ่มเมือง 1 เมืองเพื่อให้เป็นเมืองเริ่มต้น ในลำดับถัดมาทำการสุ่มเมืองที่จะต้องเดินทางถัดไปจากเมืองเริ่มต้น การเลือกเมืองถัดไปนี้ได้โดยเลือกแบบสุ่ม ที่ต้องไม่ซ้ำกับเมืองเดิม การสุ่มมีตัวแปรคือค่าความน่าจะเป็นของแต่ละเมืองซึ่งหมายถึงโอกาสที่จะถูกเลือกขึ้นอยู่กับค่านี้ โดยหากมีค่าความน่าจะเป็นสูงโอกาสที่จะโดนเลือกก็มีสูงเช่นกัน และให้ทำในทำนองเดียวกันนี้จนได้เส้นทางที่ประกอบด้วยเมืองต่างๆครบทุกเมืองและวนกลับมายังเมืองเริ่มต้นจะได้เส้นทางที่สมบูรณ์

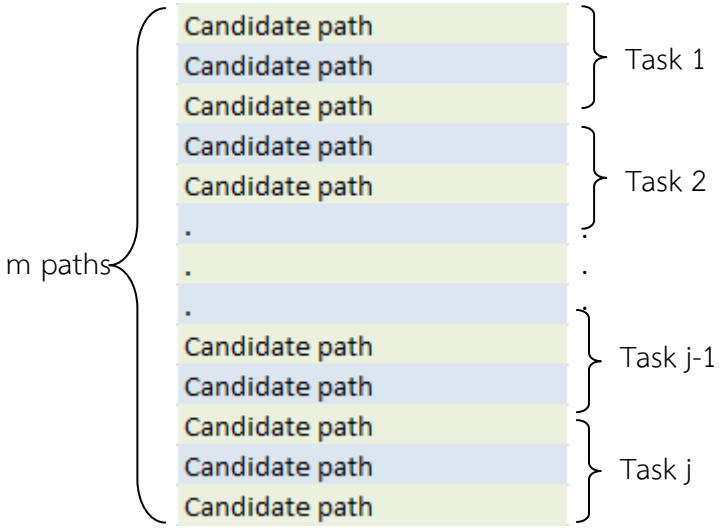
การแบ่งการทำงานในขบวนการหาเส้นทางใน 1 รุ่นประชากร แบ่งออกเป็นงานย่อย คือ เส้นทางใน 1 รุ่นประชากรแทนด้วย *n* ประชากร แบ่งเป็นกลุ่มย่อยๆ ได้ดังนี้

$$\textit{จำนวนกลุ่มย่อย} = \left\lfloor \frac{\textit{จำนวนประชากรทั้งหมด}}{\textit{จำนวนเส้นทางต่อกลุ่ม}} \right\rfloor$$

ภาพที่ 3-1 แสดงถึงเส้นทางที่สามารถเดินทางได้ จากตัวอย่างมีเมืองทั้งหมด 5 เมือง เส้นทางที่สามารถเป็นไปได้ทั้งหมด 120 เส้นทาง แต่ละเส้นทางคือตัวเลือก เมื่อกำหนดให้ใน 1 รุ่นประชากรมีทั้งหมด *m* เส้นทาง การคำนวณหาเส้นทางต้องหาให้ครบตามจำนวนที่กำหนดแต่ต้องไม่เกินไปกว่าจำนวนเส้นทางที่สามารถจะเกิดขึ้นได้ การแบ่งออกเป็นกลุ่มย่อยหรืองานย่อยเพื่อให้การทำงานสามารถทำงานพร้อมกันได้สำหรับขั้นตอนนี้ จากจำนวนเส้นทางทั้งหมดตามที่กำหนด แบ่งออกเป็นงานย่อยเป็น *j* งาน ดังแสดงในภาพที่ 3-2 เพื่อให้แต่ละงานย่อยทำงานต่างเธรดกัน

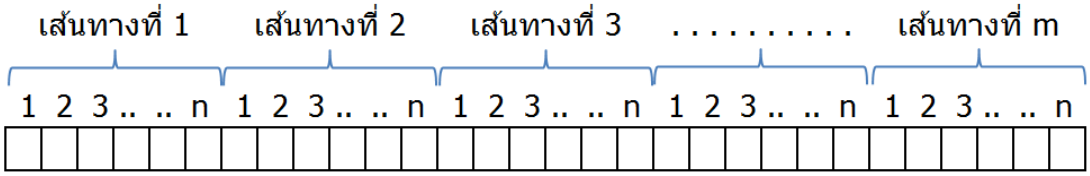


ภาพที่ 3-1 ตัวอย่างเส้นทางกรณีเดินทาง 5 เมือง



ภาพที่ 3-2 การแบ่งการสร้างเส้นทางออกเป็นกลุ่มงานย่อย

การทำงานเพื่อให้เป็นไปตามวัตถุประสงค์ ข้อมูลเส้นทางจัดเก็บในรูปของอะเรย์ 1 มิติ ที่จะมีความยาวเป็น *จำนวนเส้นทาง x จำนวนเมืองต่อเส้นทาง* ดังในแสดงในภาพที่ 3-3 ในขั้นตอนนี้ทำการสร้างเส้นทางการเดินทางตามข้อกำหนด 1 รุ่นประชากร โดยให้มีการทำงานแบบขนาน ตัวอย่างการสร้างระบบโดยใช้ *C++* ร่วมกับ Intel *TBB* แสดงในภาพที่ 3-4 แบ่งเป็น 2 ส่วนด้วยกันคือ ตัวระบบที่ทำการสร้างเส้นทาง และตัวเรียกให้มีการสร้างเส้นทางโดยทำงานแบบขนาน ตัวระบบที่สร้างเส้นทางต้องมีการเขียน*เป็น class object* ขึ้นมา โดยมีกระบวนการ (*Method*) *operator()* เป็นตัวที่บรรจุคำสั่งในการทำงานตามภาพที่ 3-4 ส่วนที่สองคือตัวเรียก *class object* ให้ทำงาน ใช้ฟังก์ชัน *parallel_for* ของ *Intel TBB* ต้องมีการกำหนดช่วงการทำงานทั้งหมดและจำนวนเส้นทางต่อการทำงาน 1 เธรด ดังภาพที่ 3-5



ภาพที่ 3-3 โครงสร้างข้อมูลสำหรับจัดเก็บเส้นทางหนึ่งรุ่น

```
1  class Generate{
2  public:
3      void operator()(const tbb::blocked_range<uint>& range) const{
4          for(uint i=range.begin();i!=range.end();++i){
5              firstTown = Random();
6              Population[i*(population_size+1)] = firstTown ;
7
8              for(uint j=1; j<population_size;j++){
9                  nextTown = RandomNextTown();
10
11                 Population[i*(population_size+1)+ j] = nextTown ;
12             }
13         }
14     }
15 };
```

ภาพที่ 3-4 รหัสเทียมคลาสของการสร้างเส้นทาง

```
1 tbb::task_scheduler_init init;
2 for(generation=0;i<MAXGENERATION;igeneration++) {
3     // Gen Population
4     tbb::parallel_for(tbb::blocked_range<uint>(0,MAXPOPULATION,MAXPOPULATION/10), Generate());
5 }
```

ภาพที่ 3-5 รหัสเทียมตัวเรียกคลาสสร้างเส้นทาง

3.2 ขั้นตอนการหาค่าความเหมาะสม

ค่าความเหมาะสมของปัญหาผู้ขายสินค้า คือการหาระยะทางสำหรับเส้นทางทุกเส้นทางที่ได้ทำการประมวลผลจากขั้นตอนการสร้างประชากร การคำนวณหาระยะทางโดยคิดจากระยะห่างของเมืองต่างๆ ตามการเดินทางเส้นทางนั้นๆ แล้วรวมผลที่ได้ทั้งหมดเป็นระยะทางรวมโดยที่ต้องมีการคิดระยะทางจากเมืองสุดท้ายกลับมายังเมืองเริ่มต้น

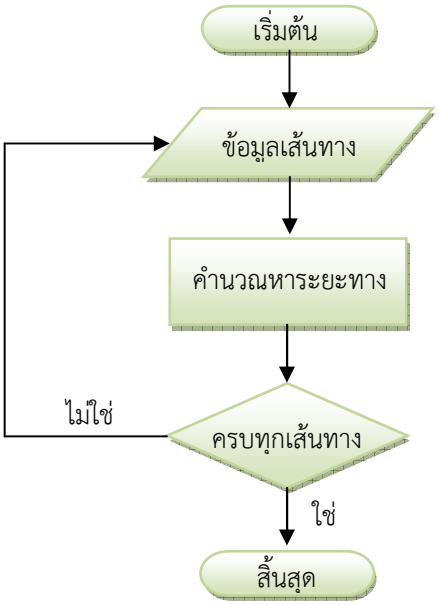
$$S = \left(\sum_{i=1}^{n-1} T_i T_{i+1}\right) + T_n T_1$$

เมื่อ S คือ ระยะทางรวม

T_i คือ เมืองตำแหน่งที่ i

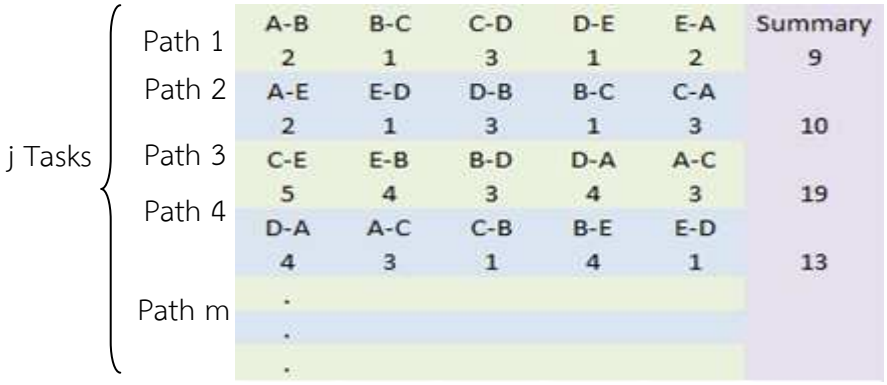
n คือ จำนวนเมือง

ขั้นตอนในการคำนวณหาระยะทางรวมของแต่ละเส้นทาง ภาพที่ 3-6 แสดงถึงวิธีการหาระยะทางทุกเส้นทาง โดยทำการวนหาทีละเส้นทางจนครบทุกเส้นทางตามจำนวนประชากรที่กำหนด



ภาพที่ 3-6 ขั้นตอนการหาระยะทางรวมของทุกเส้นทาง

เส้นทางแต่ละเส้นทางสามารถแยกการคำนวณได้อย่างเป็นอิสระต่อกัน เพื่อให้สามารถทำงานได้รวดเร็วมากขึ้น เราสามารถแบ่งการคำนวณออกเป็นงานย่อยได้ วิธีการแบ่งทำโดยนำจำนวนเส้นทางทั้งหมดมาแบ่งกลุ่มออกเป็นกลุ่มย่อยๆ แล้วสั่งให้ระบบทำงานแบบขนาน ดังแสดงในภาพที่ 3-7 ให้จำนวนประชากรเป็น *m* เส้นทาง แบ่งออกเป็นกลุ่มย่อยๆได้ *j* กลุ่ม

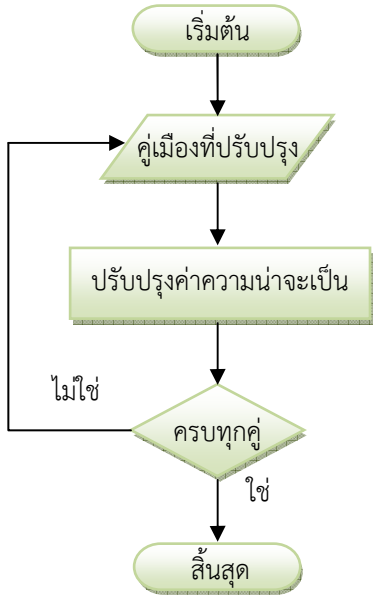


ภาพที่ 3-7 การแบ่งจำนวนเส้นทางเพื่อคำนวณหาระยะทางออกเป็นกลุ่มงานย่อย

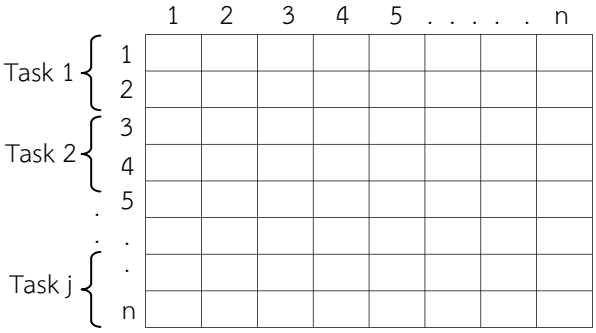
3.3 การปรับปรุงค่าความน่าจะเป็น

เป็นขั้นตอนที่มีความสำคัญมากขั้นตอนหนึ่งเพราะจะมีการนำค่าความน่าจะเป็นที่ได้รับการปรับปรุงข้อมูลแล้วไปใช้เพื่อทำการสร้างประชากรสำหรับรุ่นต่อไป ขั้นตอนการปรับปรุงตารางค่าความน่าจะเป็น เริ่มต้นโดย เลือกกลุ่มข้อมูลมา 2 กลุ่มจากตัวเลือกทั้งหมดที่มีค่าความเหมาะสมมากกว่าหรือเท่ากับค่าเฉลี่ย กลุ่มแรกเลือกมา 10 เปอร์เซ็นต์ที่มีค่าระยะทางน้อย เรียกว่ากลุ่มตัวอย่างดี กลุ่มที่ 2 เลือกมา 10 เปอร์เซ็นต์ที่มีค่าระยะทางในลำดับท้ายเรียกว่ากลุ่มตัวอย่างเลว นำค่าข้อมูลการเกิดขึ้นของเมืองคู่ใดๆ เพื่อไปเพิ่มหรือลดค่าความน่าจะเป็น โดยคู่ที่พบเจอในกลุ่มตัวอย่างดีจะเป็นการเพิ่มค่าความน่าจะเป็น ส่วนคู่ที่พบเจอในกลุ่มตัวอย่างเลวจะเป็นการลดค่าความน่าจะเป็น ทำจนครบทุกคู่ตามตารางค่าความน่าจะเป็น ดังแสดงดังภาพที่ 3-8

การทำงานเป็นแบบขนานสามารถแยกเป็นงานย่อย ในการปรับปรุงค่าความน่าจะเป็นของแถวใดแถวหนึ่งของข้อมูล สามารถทำได้เป็นอิสระต่อแถวข้อมูลอื่นๆ ดังนั้นแบ่งข้อมูลการทำงานเป็นกลุ่มย่อยตามแถวข้อมูล ดังภาพที่ 3-9 ข้อมูลค่าความน่าจะเป็นเก็บในรูปของตารางข้อมูลที่มีขนาด *n x n* โดยที่ *n* คือขนาดของปัญหา ปัญหาใหญ่ถูกแบ่งออกเป็นปัญหาย่อยได้ *j* งาน งานย่อยที่ได้จากการแบ่งถูกจัดการเพื่อให้ทำงานแบบขนานโดย *Intel TBB*



ภาพที่ 3-8 ขั้นตอนการปรับปรุงข้อมูลตารางค่าความน่าจะเป็น



ภาพที่ 3-9 การแบ่งกลุ่มข้อมูลสำหรับการปรับปรุงค่าความน่าจะเป็น

บทที่ 4

การทดลองและผลการทดลอง

4.1 เครื่องมือที่ใช้และปัจจัยแวดล้อม

การทดลองนี้ใช้เครื่องมือเพื่อวัดผลทางด้านประสิทธิภาพดังนี้ เครื่องคอมพิวเตอร์ส่วนบุคคลใช้ตัวประมวลผลกลาง *Intel Core i5-2500 3.3GHz* หน่วยความจำ *8 GB* ระบบปฏิบัติการ *Windows 7 64 bits* พัฒนาโดยใช้ *Parallel Studio XE 2011 with Visual Studio 2010* ร่วมกับ *Library* จาก *Intel* เพื่อเขียนโปรแกรมแบบขนานชื่อ *Intel Treading Building Blocks (Intel TBB)*

การทดลองทำการวัดผลการเปรียบเทียบระหว่าง การทำงานแบบลำดับกับการทำงานแบบขนาน โดยเปรียบเทียบเวลาที่ใช้ในการประมวลผลหาคำตอบจากโจทย์ที่กำหนด

4.2 ปัญหาที่ใช้เพื่อทำการทดลอง

ปัญหาที่นำมาทดลองเพื่อทำการเปรียบเทียบผลนั้น ใช้ปัญหาผู้ขายสินค้า ทั้งหมด 5 ปัญหาด้วยกัน ความแตกต่างของปัญหาคือจำนวนเมือง โดยแต่ละปัญหามีขนาดของปัญหาที่ไม่เท่ากัน ทั้งนี้จำนวนเมืองของทั้ง 5 ปัญหามีขนาดปัญหาที่ไม่เกิน 200 เมือง ปัญหาที่นำมาทดลองในครั้งนี้คือ *Dj38 Eil51 Eil76 Eil101* และ *Ch130* ที่มาจาก [14] รายละเอียดดังแสดงในตาราง 4-1

ปัญหา	รายละเอียด
DJ38	จำนวนเมืองเท่ากับ 38 เมือง
EIL51	จำนวนเมืองเท่ากับ 51 เมือง
EIL76	จำนวนเมืองเท่ากับ 76 เมือง
EIL101	จำนวนเมืองเท่ากับ 101 เมือง
CH130	จำนวนเมืองเท่ากับ 130 เมือง

ตาราง 4-1 ข้อมูลปัญหาผู้ขายสินค้าที่ใช้ในการทดลอง

ในการทดลอง ทำการเปรียบเทียบทีละปัญหาเพื่อดูเวลาที่ใช้ในการประมวลผล โดยเริ่มจากปัญหาที่เล็กที่สุดก่อน แล้วค่อยเพิ่มขนาดของปัญหาไปจนถึงปัญหาที่มีขนาดใหญ่ที่สุด การทดลองเปรียบเทียบการแก้ปัญหาโดยแยกย่อยตามการปรับให้มีการทำงานแบบขนาน กล่าวคือเมื่อ

1) สร้างประชากร
2) หาค่าความเหมาะสม
3) ปรับปรุงค่าความน่าจะเป็น
4) สร้างประชากรและหาค่าความเหมาะสม
5) สร้างประชากรและปรับปรุงความน่าจะเป็น
6) หาค่าความเหมาะสมและปรับปรุงความน่าจะเป็น
7) สร้างประชากร หาค่าความเหมาะสมและปรับปรุงค่าความน่าจะเป็น

4.3 ผลการทดลอง

การทดลองทั้ง 7 แบบ กำหนดเงื่อนไขการประมวลผลดังนี้ จำนวนประชากรกำหนดที่ 1,000 จำนวนรุ่นของประชากรกำหนดให้เป็น 500 รุ่น และจังหวะการเรียนรู้เป็น 0.1

การทดลองสำหรับแต่ละปัญหาและสำหรับแต่ละแบบการทดลอง ประมวลผลทั้งหมด 10 ครั้ง ผลการทดลองแสดงเป็นค่าเฉลี่ยที่ได้

การวัดผลด้านความเร็ววัดโดยคิดเป็นความเร็วที่เพิ่มขึ้น (Speedup) โดยเปรียบเทียบจากประมวลผลแบบลำดับกับประมวลผลแบบขนาน สูตรการคำนวณเป็นดังนี้

$$S = \frac{T_s}{T_p}$$

โดยที่ T_s คือ เวลาที่ใช้ในการประมวลผลแบบลำดับ
T_p คือ เวลาที่ใช้ในการประมวลผลแบบขนาน

4.3.1 สร้างประชากร

กำหนดให้การทำงานแบบขนานเฉพาะในส่วนของขั้นตอนการสร้างประชากรเท่านั้น เพื่อนำค่าเวลาที่ใช้ในการประมวลผลมาเปรียบเทียบ ผลการทดลองสำหรับแต่ละปัญหา ผลจากการทดลองตามตารางที่ 4-2 ข้อกำหนดต่างๆ ตามระบุใน 4.3

4.3.2 หาค่าความเหมาะสม

ขั้นตอนหาค่าความเหมาะสม (*Fitness value*) สำหรับปัญหาผู้ขายสินค้า นั้นเป็นการหาระยะทางที่ใช้สำหรับแต่ละเส้นทาง โดยเส้นทางใดที่ใช้ระยะทางสั้นที่สุดให้เป็นเส้นทางที่ดีที่สุดสำหรับจำนวนเส้นทางทั้งหมดในรุ่นนั้นๆ ดังนั้นในขั้นตอนนี้แบ่งออกเป็น 2 ส่วนคือ ส่วนแรกเป็นการหาค่าความเหมาะสม และส่วนที่สองเป็นการเรียงลำดับข้อมูลค่าความเหมาะสม โดยจะเรียงจากค่าน้อยสุดไปหามากสุด การเรียบเทียบเวลาที่ใช้ในการประมวลผลโดยให้การทำงานแบบขนานเฉพาะในส่วนของการหาค่าความเหมาะสมเท่านั้น ในขั้นตอนอื่นๆยังคงให้ทำงานแบบลำดับเช่นเดิม ผลจากการทดลองได้ตามข้อมูลในตารางที่ 4-3 สำหรับข้อกำหนดต่างๆ กำหนดตามรายละเอียดใน 4.3

4.3.3 ปรับปรุงค่าความน่าจะเป็น

ระบบทำงานแบบขนานเฉพาะในส่วนของการปรับปรุงค่าข้อมูล ในตารางค่าความน่าจะเป็นซึ่งเป็นการนำผลจากการหาค่าความเหมาะสมมาใช้งาน จากการแบ่งกลุ่มตัวอย่างออกเป็นสองกลุ่ม ในส่วนนี้มี 2 ขั้นตอนย่อยคือ ขั้นตอนแรกเป็นการนับความถี่ของการเกิดขึ้นพร้อมกันของคู่เมืองใดๆ ทั้งในกลุ่มตัวอย่างดีและกลุ่มตัวอย่างเลว ขั้นตอนที่สองคือ ใช้ค่าความถี่ของการเกิดพร้อมกันของคู่เมืองต่างๆ มาปรับปรุงค่าข้อมูลความน่าจะเป็น โดยปรับค่าเพิ่มขึ้นหากเป็นความถี่ที่เกิดขึ้นในกลุ่มตัวอย่างดี และปรับค่าลดลงในกลุ่มตัวอย่างเลว ผลการทดลองเป็นไปตามข้อมูลในตารางที่ 4-3 ข้อกำหนดต่างๆ ในการทดลองเป็นไปตามข้อมูลใน 4.3

4.3.4 สร้างประชากรและหาค่าความเหมาะสม

เปรียบเทียบการใช้เวลาในการประมวลผล ในขั้นตอนการสร้างประชากรและการหาค่าความเหมาะสม ผลการทดลองได้ดังตารางที่ 4-5 ข้อกำหนดสำหรับการทดลองเป็นไปตามรายละเอียดใน 4.3

4.3.5 สร้างประชากรและปรับปรุงความน่าจะเป็น

เปรียบเทียบเวลาในการประมวลผล การสร้างประชากรและการปรับปรุงค่าความน่าจะเป็น ผลการทดลองแสดงในตารางที่ 4-6 และในการทดลองข้อกำหนดเป็นไปตามรายการใน 4.3

4.3.6 หาค่าความเหมาะสมและปรับปรุงความน่าจะเป็น

เปรียบเทียบโดยตรวจสอบเวลาที่ใช้ในการหาคำตอบตามข้อกำหนดรายละเอียดใน 4.3 ในขั้นตอนการหาค่าความเหมาะสมและการปรับปรุงความน่าจะเป็น ผลที่ได้แสดงดังตารางที่ 4-7

4.3.7 สร้างประชากร หาค่าความเหมาะสมและปรับปรุงค่าความน่าจะเป็น

ผลการทดลองเปรียบเทียบการทำงานแบบลำดับกับการทำงานแบบขนานในทุกขั้นตอน รายละเอียดในการกำหนดค่าต่างๆ เป็นไปตามรายการที่ระบุใน 4.3 ผลการทดลองแสดงในตารางที่ 4-8

4.3.8 ผลการทดลองด้านความแม่นยำ

จากการทดลองทั้ง 7 แบบ จากแบบ 4.3.1 ถึง 4.3.7 เก็บประวัติเส้นทางการแก้ปัญหาของทั้ง 5 ปัญหาในการคำนวณแต่ละครั้ง เพื่อรายงานคำตอบในการแก้ปัญหาแต่ละปัญหา ผลการทดลองที่แสดงในตาราง 4-9 เป็นผลระยะทางที่สั้นที่สุดจากการให้โปรแกรมแก้ปัญหาแต่ละปัญหาจากทั้งหมด โดยเปรียบเทียบกับค่าที่ดีที่สุด (*Optimal Solution*) ของปัญหานั้นๆ ค่าคำตอบที่ดีที่สุดอ้างอิงจาก [14]

การประมวลผล	เวลา (วินาที)				
	ปัญหา				
	DJ 38	EIL 51	EIL 76	EIL 101	CH 130
แบบลำดับ	40.81	39.90	115.29	250.32	536.62
แบบขนาน	34.82	33.51	104.51	239.12	501.11
ความเร็วเพิ่มขึ้น (เท่า)	1.17	1.19	1.10	1.05	1.07

ตารางที่ 4-2 เปรียบเทียบเวลาที่ใช้ในการประมวลผลกรณีทำงานแบบขนานเฉพาะสร้างประชากร

การประมวลผล	เวลา (วินาที)				
	ปัญหา				
	DJ 38	EIL 51	EIL 76	EIL 101	CH 130
แบบลำดับ	40.81	39.90	115.29	250.32	536.62
แบบขนาน	40.57	39.47	113.70	248.25	519.66
ความเร็วเพิ่มขึ้น (เท่า)	1.01	1.01	1.01	1.01	1.03

ตาราง 4-3 เปรียบเทียบเวลาที่ใช้ในการประมวลผลกรณีทำงานแบบขนานเฉพาะหาค่าความเหมาะสม

การประมวลผล	เวลา (วินาที)				
	ปัญหา				
	DJ 38	EIL 51	EIL 76	EIL 101	CH 130
แบบลำดับ	40.81	39.90	115.29	250.32	536.62
แบบขนาน	18.39	17.11	45.50	94.47	185.54
ความเร็วเพิ่มขึ้น (เท่า)	2.22	2.33	2.53	2.65	2.89

ตารางที่ 4-4 เปรียบเทียบเวลาที่ใช้ในการประมวลผลกรณีทำงานแบบขนานเฉพาะปรับปรุงความน่าจะเป็น

การประมวลผล	เวลา (วินาที)				
	ปัญหา				
	DJ 38	EIL 51	EIL 76	EIL 101	CH 130
แบบลำดับ	40.81	39.90	115.29	250.32	536.62
แบบขนาน	34.48	34.04	103.91	239.12	501.11
ความเร็วเพิ่มขึ้น (เท่า)	1.18	1.17	1.11	1.05	1.07

ตารางที่ 4-5 เปรียบเทียบเวลาที่ใช้ในการประมวลผลกรณีทำงานแบบขนานเฉพาะสร้างประชากรและหาค่าความเหมาะสม

การประมวลผล	เวลา (วินาที)				
	ปัญหา				
	DJ 38	EIL 51	EIL 76	EIL 101	CH 130
แบบลำดับ	40.81	39.90	115.29	250.32	536.62
แบบขนาน	12.40	11.98	34.97	77.22	158.49
ความเร็วเพิ่มขึ้น (เท่า)	3.29	3.33	3.30	3.24	3.39

ตารางที่ 4-6 เปรียบเทียบเวลาที่ใช้ในการประมวลผลกรณีทำงานแบบขนานเฉพาะสร้างประชากรและปรับปรุงค่าความน่าจะเป็น

การประมวลผล	เวลา (วินาที)				
	ปัญหา				
	DJ 38	EIL 51	EIL 76	EIL 101	CH 130
แบบลำดับ	40.81	39.90	115.29	250.32	536.62
แบบขนาน	18.40	16.99	45.49	94.51	185.87
ความเร็วเพิ่มขึ้น (เท่า)	2.22	2.35	2.53	2.65	2.89

ตารางที่ 4-7 เปรียบเทียบเวลาที่ใช้ในการประมวลผลกรณีทำงานแบบขนานเฉพาะหาค่าความเหมาะสมและปรับปรุงค่าความน่าจะเป็น

การประมวลผล	เวลา (วินาที)				
	ปัญหา				
	DJ 38	EIL 51	EIL 76	EIL 101	CH 130
แบบลำดับ	40.81	39.90	115.29	250.32	536.62
แบบขนาน	12.07	11.90	34.57	76.72	157.60
ความเร็วเพิ่มขึ้น (เท่า)	3.38	3.35	3.33	3.26	3.40

ตารางที่ 4-8 เปรียบเทียบเวลาที่ใช้ในการประมวลผลกรณีทำงานแบบขนานในทุกขั้นตอน

การประมวลผล	ระยะทาง				
	ปัญหา				
	DJ 38	EIL 51	EIL 76	EIL 101	CH 130
ระยะทางสั้นที่สุด	6,656	426	538	629	6,110
ระยะทางคำนวณได้	7,711	483	733	992	14,246
ความถูกต้อง	86.32%	88.20%	73.40%	63.41%	42.89%

ตารางที่ 4-9 ผลการทดลองด้านความถูกต้อง

4.4 วิเคราะห์ผลการทดลอง

จากการทดลองใน 4.3 วิเคราะห์ผลการทดลองในส่วนของประสิทธิภาพ แบ่งเป็นสองส่วนคือด้านความเร็วและด้านความถูกต้องแม่นยำ

4.4.1 ประสิทธิภาพด้านความเร็ว

จากการทดลอง 4.3 ทั้งหมด 7 แบบด้วยกันตั้งแต่แบบที่ 4.3.1 ถึง แบบที่ 4.3.7 ซึ่งทำการประมวลผลแบบขนานจากหนึ่งขั้นตอน สองขั้นตอนแบบสลับครบทุกคู่และสุดท้ายให้มีการทำงาน

แบบขนานครบทั้งสามขั้นตอน พบว่าสามารถเพิ่มความเร็วได้ในทุกขั้นตอนที่ทำการทดลอง แต่ประสิทธิภาพแตกต่างกันออกไป กรณีที่ให้ทำงานแบบขนานเฉพาะขั้นตอนเดียว ปรากฏว่าในส่วนของการทำงาน การสร้างจำนวนประชากรและการหาค่าความเหมาะสม เพิ่มความเร็วในการประมวลผลได้ไม่ดีนักเมื่อให้ประมวลผลแบบขนาน โดยค่าที่ดีสุดของการสร้างจำนวนประชากร คือ 1.19 เท่า และประมาณ 1.01 เท่าสำหรับการหาค่าความเหมาะสม แต่สามารถเพิ่มความเร็วได้ดีประมาณ 2.89 เท่าสำหรับการทำงานแบบขนานเฉพาะในส่วนของการปรับปรุงค่าความน่าจะเป็น ส่วนการทดลองเมื่อมีการจับคู่ สลับคู่ของขั้นตอนการประมวลผลแบบขนานของทั้ง 3 ขั้นตอน ผลที่ได้คือ การทดลองกรณีขั้นตอนใดก็ตามที่จับคู่กับการปรับปรุงค่าความน่าจะเป็น สามารถเพิ่มความเร็วได้ 1.18 – 3.33 เท่า โดยที่การจับคู่ระหว่าง สร้างจำนวนประชากรกับปรับปรุงค่าความน่าจะเป็น สามารถเพิ่มความเร็วในการประมวลผลได้ดีกว่า การจับคู่ระหว่าง หาค่าความเหมาะสมกับปรับปรุงค่าความน่าจะเป็น ส่วนการจับคู่การประมวลผลแบบขนานระหว่าง สร้างจำนวนประชากรกับหาค่าความเหมาะสมเพิ่มความเร็วได้น้อยที่สุด ส่วนในการทดลองแบบสุดท้ายเมื่อให้ทั้งสามขั้นตอนทำงานแบบขนานประมวลผลเปรียบเทียบกับการทำงานแบบลำดับ ผลที่ได้สามารถเพิ่มความเร็วในการประมวลผลหาคำตอบของปัญหาผู้ขายสินค้า ทั้ง 5 ปัญหา สามารถเพิ่มความเร็วได้ 3.40 เท่า

การทดลองเมื่อพิจารณาถึงเวลาที่ใช้ในการหาคำตอบของทั้ง 5 ปัญหา พบว่ามีการใช้เวลาในการหาคำตอบเพิ่มมากขึ้นตามขนาดของปัญหา และเมื่อพิจารณาถึงเวลาที่ใช้สำหรับหาคำตอบเมื่อให้ทำงานแบบขนานของแต่ละขั้นตอนสามารถเรียงลำดับความเร็วที่สามารถเพิ่มได้ โดยเรียงจากน้อยไปหามาก ได้ดังนี้ หาค่าความเหมาะสม สร้างจำนวนประชากร และ ปรับปรุงค่าความน่าจะเป็น

4.4.2 ประสิทธิภาพด้านความถูกต้องแม่นยำ

จากการทดลองใน 4.3 ของการประมวลผลทั้ง 7 แบบ โดยกำหนดค่าตัวแปรให้มีค่าเดียวกันสำหรับการแก้ปัญหาทั้ง 5 ปัญหา พิจารณาผลของแต่ละปัญหาพบว่า ขนาดของปัญหาที่มีขนาดเล็กให้ค่าความถูกต้องแม่นยำมากที่สุด และค่าความถูกต้องแม่นยำลดลงเมื่อขนาดของปัญหาใหญ่มากขึ้น จากข้อมูลในตารางที่ 4-9 สามารถแสดงให้เห็นถึงการลดลงของความถูกต้องแม่นยำได้อย่างชัดเจน

4.4.3 สรุปผลการทดลอง

จากการทดลองทั้งทางด้านความเร็วและความถูกต้องแม่นยำ ผลการทดลองสามารถบ่งบอกได้อย่างชัดแจ้งว่า วิธีที่นำเสนอมีประสิทธิภาพในด้านความเร็วถึงแม้ปัญหาที่ต้องการหาคำตอบมีขนาดที่ใหญ่ขึ้น โดยขั้นตอนที่สามารถเพิ่มความเร็วได้ดีที่สุดคือ ขั้นตอนการปรับปรุงค่าความน่าจะเป็น ลำดับรองลงมาคือการสร้างจำนวนประชากร และลำดับสุดท้ายคือการหาค่าความเหมาะสม

ในส่วนของประสิทธิภาพความถูกต้องแม่นยำ หากเป็นการหาคำตอบสำหรับปัญหาที่มีขนาดเล็ก มีความถูกต้องแม่นยำสูง แต่ความถูกต้องแม่นยำลดลงเมื่อขนาดของปัญหาใหญ่ขึ้น เพราะว่าเมื่อปัญหาใหญ่ขึ้นจำนวนเส้นทางที่สามารถเป็นไปได้มีค่ามากขึ้น แต่จำนวนเส้นทางที่ระบบกำหนดและจำนวนรุ่นของประชากร ยังคงใช้ค่าเดียวกันกับการแก้ปัญหาที่เล็กกว่า ทำให้ข้อมูลที่ได้สำหรับแต่ละรุ่นประชากร ไม่ครอบคลุมเพียงพอ จึงกล่าวได้ว่าหากต้องการแก้ปัญหาที่มีขนาดแตกต่างกันต้องมีการกำหนดพารามิเตอร์ที่มีผลกับการหาคำตอบที่เหมาะสมด้วย

บทที่ 5

สรุปผลการวิจัย และข้อเสนอแนะ

สรุปผลการวิจัย

ปัญหาของการหาคำตอบให้กับปัญหาในกลุ่มของ *EA* นั้นคือเรื่องของเวลาเพื่อให้ได้มาซึ่งคำตอบที่ดีที่สุด การหาคำตอบใช้เวลามากในการประมวลผล และหากขนาดของปัญหามีขนาดใหญ่มากขึ้น ภาระในการประมวลผลหาคำตอบ เพิ่มขึ้นมากตามขนาดของปัญหาเช่นกัน

งานวิจัยชิ้นนี้เสนอการแก้ปัญหาในกลุ่มของ *EA* โดยใช้ขั้นตอนวิธี *COIN* ปัญหาที่นำมาทดสอบเป็นปัญหาผู้ขายสินค้าที่มีขนาดของปัญหาไม่เกิน 200 เมือง การแก้ปัญหาให้ระบบทำงานแบบขนานบนตัวประมวลผลหลายแกน เปรียบเทียบผลลัพธ์ในการใช้เวลาในการหาคำตอบกับการประมวลผลแบบลำดับ ปัญหาผู้ขายสินค้าในงานวิจัยนี้นี้ใช้ทั้งหมด จำนวน 5 ปัญหาด้วยกัน การวิจัยวิเคราะห์แยกรายการขั้นตอนของการประมวลผลแบบขนาน สลับจับคู่ขั้นตอนและ สุดท้ายรวมทุกขั้นตอน

จากผลการทดลองเพื่อเปรียบเทียบประสิทธิภาพด้านความเร็ว และความถูกต้องแม่นยำ สรุปได้ว่าวิธีที่นำเสนอสามารถลดระยะเวลาในการประมวลผลหาคำตอบสำหรับปัญหาที่กำหนดได้ เมื่อเปรียบเทียบกับการแก้ปัญหาโดยใช้การประมวลผลแบบลำดับที่ใช้ตัวประมวลผลกลางแบบแกนเดียว สามารถทำความเร็วได้เพิ่มขึ้น 3 เท่า เมื่อให้ทำงานแบบขนานมันตัวประมวลผลแบบ 4 แกน ในด้านความถูกต้องแม่นยำ จากการกำหนดค่าพารามิเตอร์ที่มีผลต่อการแก้ปัญหาไว้ตาม 4.3 ให้ความถูกต้องแม่นยำสูงถึง 80 เปอร์เซ็นต์ ในปัญหาที่มีขนาดเล็ก และความถูกต้องแม่นยำลดลงเมื่อแก้ปัญหาที่มีขนาดใหญ่ขึ้น

ข้อเสนอแนะ

สำหรับการทดลองแก้ไขปัญหาผู้ขายสินค้า ที่มีขนาดของปัญหาต่างกัน ควรมีการกำหนดค่าพารามิเตอร์ที่มีผลต่อการทดลองที่ต่างกันออกไป พารามิเตอร์ดังกล่าวคือ จำนวนรุ่น จำนวนประชากรในแต่ละรุ่น วิธีการเลือกกลุ่มตัวอย่าง และจำนวนประชากรที่เลือกเพื่อมาปรับปรุงค่าความน่าจะเป็นในแต่ละรุ่น ควรทดสอบกับตัวประมวลผลที่มีมากกว่า 4 แกนเปรียบเทียบการโตของความเร็วในการประมวลผล

รายการอ้างอิง

[1] Evolutionary Algorithm. [Online]. Available from : http://en.wikipedia.org/wiki/Evolutionary_algorithm [2012, January 20th]

[2] Evolutionary Computation. [Online]. Available from : http://en.wikipedia.org/wiki/Evolutionary_computation [2012, January 20th]

[3] Wattanapornprom, W. Olanviwitchai, P. Chutima, P. and Chongstitvatana, P. (2009). Multi-objective Combinatorial Optimisation with Coincidence Algorithm. 2009 IEEE Congress on Evolutionary Computation, pp. 1675-1682. Trondheim.

[4] Tera-Scale. [Online]. Available from : http://techresearch.intel.com/ResearchAreaDetails.aspx?Id=27 [2011, September 15th]

[5] Intel Threading Building Blocks. [Online]. Available from : http://threadingbuildingblocks.org [2011, September 22th]

[6] OpenMP. [Online]. Available from : http://openmp.org/wp/ [2012, March 3rd]

[7] OpenMP. [Online]. Available from : http://en.wikipedia.org/wiki/OpenMP [2012, March 3rd]

[8] Barney, B. Introduction to Parallel Computing. [Online]. Available from : https://computing.llnl.gov/tutorials/parallel_comp [2011, September 15th]

[9] Reinders, J. (2007). Intel Threading Building Blocks, Outfitting C++ for Multi-core Processor Parallelism. 1st ed. O`Reilly Press.

[10] Hongzhong Shan. (2010). Hybrid Programming for Multicore Processors. 2011 Fourth International Joint Conference on Computational Sciences and Optimization (CSO), pp. 261-262. Yunnan.

[11] Shenshen Liang, Ying Liu, Cheng Wang and Liheng Jian. (2010). Design and evaluation of a parallel k-nearest neighbor algorithm on CUDA-enabled GPU. 2010 IEEE 2nd Symposium on Web Society (SWS), pp. 53-60. Beijing.

[12] Yussof, S. Razali, R.A. and Ong Hang See. (2009). A Parallel Genetic Algorithm for Shortest Path Routing Problem. 2009 International Conference on Future Computer and Communication, pp. 268-273. Kuala Lumpur.

[13] Marowka, A. (2008). Towards High-Level Parallel Programming Models for Multicore Systems. Advance Software Engineering and Its Application, pp. 226-229. Hainan Island.

[14] Symmetric traveling salesman problem (TSP) data. [Online]. Available from : http://www2.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp [2012 January 10th]

ประวัติผู้เขียนวิทยานิพนธ์

นายวีรชิต ศรีมุข เกิดเมื่อวันที่ 29 กุมภาพันธ์ 2519 ที่จังหวัดนครศรีธรรมราช จบการศึกษาชั้นประกาศณียบัตรวิชาชีพ(ปวช.) ที่วิทยาลัยเทคนิคนครศรีธรรมราช ในสาขาวิชา ช่างอิเล็กทรอนิกส์ จังหวัดนครศรีธรรมราช และจบการศึกษาระดับปริญญาบัณฑิตจากภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาตร์ มหาวิทยาลัยรามคำแหง เมื่อปีการศึกษา 2545