ใบรับรองวิทยานิพนธ์
บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์

วิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมคอมพิวเตอร์)

ปริญญา

วิศวกรรมคอมพิวเตอร์	วิศวกรรมคอมพิวเตอร์
สาขา	**ภาควิชา**

เรื่อง การปรับปรุงสมรรถนะของโปรแกรมจำลองโมเลกุลแบบไดนามิค ในระดับซอร์สโค้ด

Source-Level Optimization for Molecular Dynamic Simulator

นามผู้วิจัย นายวีระพงษ์ แก้วเทศ

ได้พิจารณาเห็นชอบโดย

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก ....

(อาจารย์ภารุจ รัตนวรพันธุ์, Ph.D.)

อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม ....

(รองศาสตราจารย์อานนท์ รุ่งสว่าง, Ph.D.)

หัวหน้าภาควิชา ....

(รองศาสตราจารย์อนันต์ ผลเพิ่ม, Ph.D.)

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์รับรองแล้ว

....

(รองศาสตราจารย์กัญจนา ธีระกุล, D.Agr.)

คณบดีบัณฑิตวิทยาลัย

วันที่ .... **เดือน** .... **พ.ศ.** ....

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

วิทยานิพนธ์

เรื่อง

การปรับปรุงสมรรถนะของโปรแกรมจำลองโมเลกุลแบบไดนามิคในระดับซอร์สโค้ด

Source-Level Optimization for Molecular Dynamic Simulator

โดย

นายวีระพงษ์ แก้วเทศ

เสนอ

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์
เพื่อความสมบูรณ์แห่งปริญญาวิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมคอมพิวเตอร์)
พ.ศ. 2557

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

วีระพงษ์ แก้วเทศ 2557: การปรับปรุงสมรรถนะของโปรแกรมจำลองโมเลกุลแบบไดนามิคในระดับซอร์สโค้ด ปริญญาวิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมคอมพิวเตอร์) สาขาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก: อาจารย์ภารุจ รัตนวรพันธุ์, Ph.D. 45 หน้า

โปรแกรม LAMMPS (Steve *et al.*, 1990) ซึ่งเป็นโปรแกรมประยุกต์ที่ทำงานเกี่ยวกับการจำลองการเคลื่อนที่ของโมเลกุลแบบไดนามิ ค ทั้งยังเป็นโปรแกรม ที่ถูกใช้งานอย่างแพร่หลายและเป็นที่นิยม อย่างมากโดยเฉพาะอย่างยิ่งกับนักเรียนและนักศึกษา แต่การทำงานของโปรแกรม LAMMPS นั้นยังติดปัญหาในเรื่องของสมรรถ นะในการประมวลผลเพราะใช้เวลาในการประมวลผลนานมากบางครั้งใช้เวลาหลายอาทิตย์เลยทีเดียว ตัวแปล ภาษา และ สถาปัตยกรรมคอมพิวเตอร์แบบซุปเปอร์สเกลาร์ ในปัจจุบันนี้มีกระบวน ในการ การเพิ่มสมรรถนะการ ประมวลผลให้กับโปรแกรมอยู่แล้ว แต่ว่าก็ยังไม่สามารถเพิ่มสมรรถนะได้อย่างเต็มที่เนื่องจากยังติดปัญหาหลายอย่างของสถาปัตยกรรมคอมพิวเตอร์ ซึ่งส่งผลให้สมรรถนะ ในการประมวลผลโปรแกรมยังไม่ดีเท่าที่ควร วิธีการเพิ่มสมรรถนะให้กับโปรแกรมนั้นมีอยู่ด้วยกันหลากหลายวิธี ซึ่งแต่ละวิธีจะต้องถูกนำไปใช้ในสถานการณ์ที่เหมาะสมตามลักษณะและพฤติกรรมการทำงานของโปรแกรม

งานวิจัยนี้นำ เสนอ วิธีการ แบบต่างๆ เพิ่ม สมรรถนะการประมวลผลใ ห้กับ โปรแกรม LAMMPS ซึ่งตัวแปลภาษาและสถาปัตยกรรมคอมพิวเตอร์แบบซุปเปอร์สเกลาร์ ไม่สามารถทำให้ได้ จุดประสงค์ของงานวิจัยชิ้นนี้ประกอบด้วย การแสดงให้เห็นว่าเรายังมีโอกาสในการสามารถเพิ่มสมรรถนะของโปรแกรม LAMMPS โดยใช้วิธีง่ายๆ หลังจากเพิ่มสมรรถนะแล้วโปรแกรมยังสามารถทำงานบนเครื่องคอมพิวเตอร์ทั่วๆไป และท้ายสุดงานวิจัยชิ้นนี้ได้อธิบายว่าสาเหตุใดที่ทำให้โปรแก รมไม่สามารถทำงานได้อย่างเต็มสมรรถนะ ผลการทดลองทำให้เห็นว่า งานวิจัยชิ้นนี้สามารถเพิ่มสมถรรนะให้กับ LAMMPS ได้มากกว่างานวิจัยที่ใช้เทคนิคเดียวกัน แต่ว่ายังคงน้อยกว่างานวิจัยที่ใช้เทคนิคพิเศษ ซึ่งงานวิจัยที่ใช้เทคนิคพิเศษจะมีข้อด้อยกว่าคือโปรแกรมที่ใช้เทคนิคพิเศษจะทำงานได้เฉพาะเครื่องที่ทำการปรับปรุงสมรรถนะสำหรับเครื่องเครื่องนั้นเท่านั้น ไม่สามารถนำไป Execute บนเครื่องอื่นๆได้

__________ ____________________ ___ / ___ / ___

ลายมือชื่อนิสิต ลายมือชื่ออาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

Veerapong Kaewtes 2014: Source-Level Optimization for Molecular Dynamic Simulator. Master of Engineering (Computer Engineering), Major Field: Computer Engineering, Department of Computer Engineering. Thesis Advisor: Mr. Paruj Ratanaworabhan, Ph.D. 45 pages.

LAMMPS is a classical molecular dynamics written in C++. It is used heavily for simulation of solid-state materials and soft matter. It can also be used to model atoms or, more generically, as a parallel particle simulator at the atomic scale. LAMMPS can run on a single processor or on multiprocessors. For the latter, it uses the message-passing techniques for parallel computation. LAMMPS is distributed as an open source code under the terms of the GPL by Sandia National Laboratories, a US Department of Energy laboratory.

This study investigates the runtime behavior of LAMMPS, aiming to further improve its performance. We focus on floating-point operations because they are responsible for a large percentage of LAMMPS execution time. Floating-point operations in LAMMPS have many data (true) dependencies so their execution must necessarily be serialized. Even the superscalar out-of-order processor that extracts instruction-level parallelism (ILP) cannot help much in such a situation. A more sophisticated source code transformation like software pipelining can alleviate the situation. In this paper, however, we focus on a much simpler transformation; instruction reordering that tries to set apart the producers away from the consumers as much as possible. The others optimization methods used in this paper are common techniques; these are loop unrolling, eliminating control flow to reduce branching (an, hence, the penalty from misprediction), and strength reduction.

		___ / ___ / ___
Student's signature	Thesis Advisor's signature	

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

กิตติกรรมประกาศ

ข้าพเจ้าขอกราบขอบพระคุณอาจารย์ ภารุจ รัตนวรพันธุ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก ที่ได้ประสิทธิ์ ประสาทวิชาความรู้และทฤษฎีต่าง ๆ ตลอดจนเป็นที่ปรึกษาในการวางแนวทาง วางแผนงานแก้ปัญหา และตรวจสอบข้อบกพร่องต่าง ๆ จนงานวิจัยนี้สำเร็จลุล่วง ตลอดจนอาจารย์ในภาควิชาวิศวกรรมคอมพิวเตอร์ ที่กรุณาให้คำปรึกษา และข้อเสนอแนะต่าง ๆ จนส่งผลให้งานวิจัยนี้สมบูรณ์ยิ่งขึ้นในทุกๆ ด้าน และประสบผลสำเร็จลุล่วงไปด้วยดี

ขอขอบคุณสมาชิกห้องปฏิบัติการ INOX (Innovative Extremist) คุณท่านที่คอยแนะนำให้คำปรึกษาและคำอธิบายที่เป็นประโยชน์ต่องานวิจัย ไม่ว่าจะเป็นการให้ยืมใช้อุปกรณ์สำหรับทำงานวิจัย รวมไปถึงการติดตามและช่วยแก้ปัญหาที่เกิดจากอุปกรณ์ดังกล่าว ที่จำเป็นสำหรับงานวิจัยและให้กำลังใจที่ดีเสมอมา

ขอขอบคุณเจ้าหน้าที่โครงการบัณฑิตศึกษา และเจ้าหน้าที่ธุรการภาควิชาวิศวกรรมคอมพิวเตอร์มหาวิทยาลัยเกษตรศาสตร์ที่ช่วยเหลือในการประสานงาน และดำเนินงานด้านเอกสารต่างๆ ให้เป็นไปอย่างสะดวกลุล่วงไปด้วยดี

คุณงามความดีหรือประโยชน์อันใดเนื่องจากวิทยานิพนธ์เล่มนี้ ขออุทิศให้แก่ บิดา มารดา ญาติสนิทมิตรสหาย ตลอดทั้งครูอาจารย์และผู้มีพระคุณทุกท่าน ที่ได้อบรมและให้กำลังใจเสมอมา

วีระพงษ์ แก้วเทศ

พฤษจิกายน 2557

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

สารบัญ

หน้า

สารบัญ	(1)
สารบัญตาราง	(2)
สารบัญภาพ	(3)
คำอธิบายสัญลักษณ์และคำย่อ	(5)
คำนำ	1
วัตถุประสงค์	3
การตรวจเอกสาร	4
อุปกรณ์และวิธีการ	19
อุปกรณ์	19
วิธีการ	20
ผลและวิจารณ์	29
ผล	29
วิจารณ์	40
สรุปและข้อเสนอแนะ	41
สรุป	41
ข้อเสนอแนะ	42
เอกสารและสิ่งอ้างอิง	43
ประวัติการศึกษาและการทำงาน	45

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

สารบัญตาราง

ตารางที่ **หน้า**

1 แสดงถึงข้อมูลการประมวลผลแต่ละส่วนของโปรแกรม LAMMPS (LAMMPS Profile) 21

2 แสดงข้อมูลของแต่ละแพลตฟอร์มที่นำมาใช้ในการทดลองครั้งนี้ 29

3 แสดงถึงข้อมูลของ Rock Cluster ที่มีการทำงานแบบขนานโดยใช้ MPI จำนวน 16 โหนด 33

4 แสดงถึงข้อมูลของ Rock Cluster ที่มีการทำงานแบบลำดับ 34

5 แสดงถึงข้อมูลของ Mac OSX ที่มีการทำงานแบบขนานโดยใช้ MPI จำนวน 4 โหนด 34

6 แสดงถึงข้อมูลของ Mac OSX ที่มีการทำงานแบบลำดับ 34

7 แสดงถึงข้อมูลของ Ubuntu ที่มีการทำงานแบบขนานโดยใช้ MPI จำนวน 4 โหนด 35

8 แสดงถึงข้อมูลของ Ubuntu ที่มีการทำงานแบบลำดับ 35

9 แสดงถึงเวลาในการประมวลผลแบบเฉลี่ยของทุกแพลตฟอร์ม 36

10 แสดงถึงเปอร์เซ็นต์ที่ดีขึ้นสำหรับการประมวลผลในแต่ละแพลตฟอร์ม 36

11 แสดงถึง Line of Code ของโปรแกรมก่อนถูกแก้ไข 37

12 แสดงถึง Line of Code ของโปรแกรมที่ถูกแก้ไขไปเพื่อให้ได้ Best performance 37

13 แสดงถึง Line of Code ของโปรแกรมที่ถูกแก้ไขไปด้วยวิธี Floating-point reordering 38

14 แสดงถึง Line of Code ของโปรแกรมที่ถูกแก้ไขไปแต่ละวิธีในรูปแบบเปอร์เซ็นต์ 38

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

สารบัญภาพ

ภาพที่		หน้า
1	ตัวอย่างปัญหาการขึ้นต่อกันของข้อมูลในภาษาซี	7
2	ตัวอย่างปัญหาการขึ้นต่อกันของข้อมูลในภาษาแอสแซมบลี (Assembly)	7
3	ตัวอย่างปัญหาการขึ้นต่อกันของข้อมูลแบบ RAW WAR และ WAW	8
4	ตัวอย่างประกอบการทำงานของ Window of Execution	8
5	โครงสร้างของฮาร์ดแวร์ภายในสถาปัตยกรรมแบบซุปเปอร์สเกลาร์	9
6	ตัวอย่างของวิธี Register renaming	10
7	ตัวอย่างของ Reorder Buffer	11
8	ตัวอย่างของ Register renaming ที่ทำงานร่วมกับ Reorder Buffer	11
9	ตัวอย่างคำสั่งภาษาแอสแซมบลีของ Regiter renaming	12
10	ตัวอย่างของ Single share queue	13
11	ตัวอย่างของ Muliple queue; one per instruction type	13
12	ตัวอย่างของ Muliple reservation stations; one per instruction type	14
13	ตัวอย่างคำสั่งวนซ้ำแบบ For-loop	22
14	ตัวอย่างคำสั่งวนซ้ำหลังจากทำ Loop unrolling แล้ว	22
15	ตัวอย่างชุดคำสั่งก่อนทำการสลับลำดับคำสั่ง	23
16	ตัวอย่างชุดคำสั่งหลังทำการสลับลำดับคำสั่ง	23
17	แสดงถึงส่วนที่เป็นข้อจำกัดของสถาปัตยกรรมแบบซุปเปอร์สเกลาร์	24
18	แสดงถึงปัญหาการขึ้นต่อกันของข้อมูลของคำสั่งโปรแกรมก่อนการสลับลำดับคำสั่ง	24
19	แสดงถึงคำสั่งโปรแกรมที่สามารถทำงานแบบขนานไปพร้อมๆได้	25
20	แสดงถึงคำสั่งกระโดดต่อเนื่องหลายคำสั่ง	26
21	แสดงถึงคำสั่งโปรแกรมหลังจากคำสั่งกระโดดได้ถูกจำกัดไป	26
22	แสดงถึงการทำงานของคำสั่งโปรแกรมแบบละเอียดหลังจากคำสั่งกระโดดได้ถูกจำกัดออกไป	27
23	แสดงถึงคำสั่งโปรแกรมก่อนทำการแก้ไขปัญหา Lazy evaluation	28

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

ภาพที่		หน้า
24	แสดงถึงคำสั่งโปรแกรมหลังทำการแก้ไขปัญหา Lazy evaluation	28
25	แสดงถึงคำสั่งแปลภาษาของโปรแกรม LAMMPS แบบลำดับ	30
26	แสดงถึงไฟล์ตั้งค่าของ Solvated 5-mer peptide	31
27	แสดงถึงคำสั่งที่สั่งให้โปรแกรม LAMMPS ทำงานแบบลำดับ	32
28	แสดงถึง Logging ของโปรแกรม LAMMPS ขณะที่ทำงาน	32
29	แสดงถึง Logging ของโปรแกรม LAMMPS หลังจากสิ้นสุดการทำงาน	33
30	แผนภาพแสดงการเปรียบเทียบผลลัพธ์ของแพลตฟอร์มแบบต่างๆ	39

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

คำอธิบายสัญลักษณ์และคำย่อ

LAMMPS	=	Large-scale Atomic/Molecular Massively Parallel Simulator
OoO	=	Out of Order
CPI	=	Clock Per Instruction
ROB	=	Reorder Buffer
MPI	=	Message Passing Interface
NOP	=	No-Operation
ILP	=	Instruction Level Parallelism
RISC	=	Reduced Instruction Set Computer
VLIW	=	Very-Large Instruction Word
FUs	=	Function Unit
RS	=	Reservation Station

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

การปรับปรุงสมรรถนะของโปรแกรมจำลองโมเลกุลแบบไดนามิคในระดับซอร์สโค้ด

Source-Level Optimization for Molecular Dynamic Simulator

คำนำ

LAMMPS เป็นโปรแกรมที่ใช้จำลอ งการเคลื่อนที่ของโมกุลแบบไดนามิ คที่ได้รับความนิยมและมีผู้ใช้งานเป็นจำนวนมาก โดยเฉพาะนักเรียนนักศึกษา ซึ่ง LAMMPS เป็นโปรแกรมที่มีขนาดใหญ่และมีการ คำนวณ ทางคณิตศาสตร์สูง โดยเฉพาะเลขทศนิยมสูงมาก ดังนั้นเวลาในการประมวลผลของ LAMMPS จึงใช้เวลานานมากขึ้นอยู่กับปริมา ณข้อมูลที่นำมาคำนวณ ซึ่งบางครั้งอาจต้องใช้เวลาเป็นอาทิตย์ ยิ่งถ้าหากใช้คอมพิวเตอร์ส่วนบุคคลที่มีสมรรถนะ ไม่สูงมากด้วยแล้วจะยิ่งทำให้เวลาในการรอผลลัพธ์ของการคำนวณยาวนานยิ่งขึ้น

ตัวแปล ภาษาในปัจจุบันได้นั้นได้มีการออกแบ บให้สามารถปรับปรุง สมรรถนะ ของโปรแกรมให้ดี ยิ่งขึ้นได้ โดยจะทำการเปลี่ยนแปลงปรับปรุงชุดคำสั่งขณะแปลภาษาเพื่อให้มีสมรรถนะดีขึ้น แต่ก็ยังไม่เพียงพอเพราะยังไม่สามารถทำให้โปรแกรมใช้งาน ทรัพยากร ของเครื่องคอมพิวเตอร์ได้อย่างเต็มที่ ได้เทียบเท่ากับ การเปลี่ยนแปลงปรับปรุงชุดคำสั่งที่เกิดจากฝีมือ มนุษย์ เนื่องจากตัวแปลภาษาเหล่านั้นไม่สามารถรู้ถึงลักษณะและพฤติกรรมของโปรแกรมได้ ลึกซึ้งเท่ากับมนุษย์

สถาปัตยกรรมแบบซุปเปอร์สเกลาร์นั้นได้ออกแบบมาเพื่อช่วยเพิ่มสมรรถนะในการประมวลผลแบบขนาน ซึ่งมีผลทำให้สมรรถนะของโปรแกรมดียิ่งขึ้น แต่ก็ยังมีในบางกรณีที่เป็นกรณีที่ซับซ้อนเกินกว่าสถาปัตยกรรมแบบซุปเปอร์สเกลาร์จะจัดการได้ ซึ่งส่งผลให้โปรแกรมยังคงประมวลผลได้ไม่เต็มที่ งานวิจัยชิ้นนี้จึงมุ่งเน้นแก้ปัญหาตรงจุดนี้ รวมไปถึงอธิบายอย่างละเอียดถึงปัญหาและวิธีแก้ปัญหา

ผลการทดลองของงานวิจัยชิ้นนี้สามารถเพิ่มสมรรถนะ ด้านเวลาการประมวลผล ให้กับโปรแกรมได้มากสุดประมาณ 28% เมื่อเทียบกับสมรรถก่อนการปรับปรุง ซึ่งแสดงให้เห็นว่ายังคงมีช่องว่างที่จะสามารถเพิ่มสมรรถนะให้กับโปรแกรมได้ โดยที่ตัวแปลภาษาและสถาปัตยกรรมแบบ

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ซุปเปอร์สเกลาร์ไม่สามารถทำให้ได้ โดยที่วิธีการต่างๆในงานวิจัยชิ้ นนี้เป็นเทคนิคอย่างง่ายที่ยังคงทำให้โปรแกรมสามารถทำงานได้บนเครื่องคอมพิวเตอร์ชนิดต่างๆทั่วไป

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

วัตถุประสงค์

1. เพิ่มสมรรถนะในการประมวลผลให้กับโปรแกรม LAMMPS โดยใช้เทคนิคง่ายๆที่มีใช้ทั่วไป โดยที่โปรแกรมยังคงสามารถทำงานบนคอมพิวเตอร์ทุกแพลตฟอร์มได้เหมือนเดิม

2. อธิบายถึงเหตุผลว่าทำไมตัวแปลภาษาและสถาปัตยกรรมแบบซุปเปอร์สเกลาร์ไม่สามารถจัดการกับปัญหาเหล่านั้นให้หมดไปได้ทั้งหมด

ขั้นตอนการวิจัย

1. ศึกษาวิธีการเพิ่มสมรรถนะทางด้านความเร็วในการประมวลผล

2. ศึกษางานวิจัยก่อนหน้า เพื่อวิเคราะห์ปัญหาและรวบรวมข้อดีและข้อด้อยเพื่อนำมาเป็นข้อมูลในการพัฒนาวิธีการใหม่ๆ

3. รวบรวมและวิเคราะห์ข้อมูล เพื่อศึกษาลักษณะและพฤติกรรมของโปรแกรม เพื่อที่จะสามารถแก้ปัญหาได้ตรงจุด

4. ทดลองเพิ่มสมรรถนะของโปรแกรมโดยใช้วิธีการที่ได้ศึกษามา

5. ทดลองเพิ่มสมรรถนะของโปรแกรมโดยใช้วิธีการแก้ปัญหาเลขทศนิยมที่มีต่อซีพียูแบบซุปเปอร์สเกลาร์

6. ทดสอบและวัดผลการทดลอง

7. สรุปผลการวิจัยและประโยชน์ที่ได้รับ

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

การตรวจเอกสาร

ความรู้พื้นฐาน

การเพิ่มสมรรถนะของโปรแกรมเป็นสิ่งที่มีความสำคัญเป็นอย่างมากทั้งทางด้านการพัฒนาซอฟต์แวร์และ สถาปัตยกรรมทางคอมพิวเตอร์ ปัจจุบันผู้ออกแบบ ตัวแปล ภาษาคอมพิวเตอร์ได้พยายามทำการเพิ่มสรรถนะของซอ ฟต์แวร์ขณะเวลาที่กำลังดำเนินการแปลภาษาโปรแกรมอยู่ เพื่อให้รหัสโปรแกรมที่เป็น คำสั่งภาษาเครื่องมี สมรรถนะดีที่สุดในหลายๆด้าน ทั้ง ในด้านของเวลาในการประมวลผล เทคนิคการเพิ่มส มรรถนะของโปรแกรมที่ ตัวแปลภาษาจัดการให้แบบอัตโนมัตินั้นไม่สามารถเพิ่มสรรถนะได้อย่างเพียงพอ เพราะยังมีช่องว่างที่ยังสามารถเพิ่มสมรรถนะได้อยู่ เนื่องจากตัวแปลภาษาเหล่านั้นไม่สามารถรับรู้ถึงพฤติกรรมของโปรแกรม ทั้งหมดขณะประมวลผลในขั้นตอนการแปลภาษาได้

สำหรับในส่วนของ สถาปัตยกรรมแบบซุปเปอร์สเกลาร์นั้นได้มีความพยายามที่จะเพิ่มสมรรถนะการประมวลผลของโปรแกรม โดยการเพิ่มเทคนิคการประมวลผลแบบขนานเข้ามาช่วยในการประมวลผล ซึ่งในบางกรณีการประมวลผลแบบขนานไม่สามารถทำได้ดีเท่าที่ควรจะเป็นเนื่องจากปัญหาหลายๆด้าน โดยเฉพาะปัญหาการขึ้นต่อกันของข้อมูลแบบ True dependency ซึ่งถือเป็นอุปสรรคสำคัญอย่างยิ่งของการประมวลผลแบบขนาน เนื่องจากการขึ้นต่อกันของข้อมูลแบบ True dependency จำเป็นจะต้องประมวลผลแบบตามลำดับเท่านั้นเพราะ การอ่านผลลัพธ์ที่เกิดขึ้นหลังจากการเขียนข้อมูล หากเกิดขึ้นแบบไม่เป็นลำดับจะทำให้ข้อมูลผิดพลาดได้

1. สถาปัตยกรรมของหน่วยประมวลผลแบบแบบซุปเปอร์สเกลาร์ (The Microarchitecture of Superscalar Processors)

สถาปัตยกรรมของหน่วยประมวลผลแบบซุปเปอร์สเกลาร์ (James *et al.*, 1995) นั้นสามารถที่จะประมวลผลคำสั่งได้มากกว่าหนึ่งคำสั่งต่อ หนึ่งรอบสัญญาณนาฬิกาหนึ่งสัญญาณ โดยวิธีการที่นำมาใช้นั้นประกอบไปด้วยสิ่งที่สำคัญหลายอย่างคือ การดึงคำสั่งเข้ามาประมวลทีละหลายคำสั่งพร้อมกัน การทำนายคำสั่งกระโดด (Branch Prediction) การจัดลำดับคำสั่ง แบบไดนามิค (Silvera *et al.*, 1997) (Dynamic Instruction Scheduling) และการทำงานแบบขนานกันระดับ คำสั่ง (ILP)

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ความเป็นมาของสถาปัตยกรรมแบบซุปเปอร์สเกลาร์นั้นเริ่มต้นมาจากกระบวนการประมวลผลคำสั่งแบบ สายการผลิต (Jones *et al*., 1990; Haitao et al., 2012) (Pipelining) ซึ่งเริ่มต้นขึ้นเมื่อปี 1950 รูปแบบของการประมวลผ ลแบบนี้นั้นเปรียบเส มือนส ายการ ผลิตในโรง งานอุตสาหกรรม ซึ่งจะมีรูปแบบการทำงานคือแบ่งห น้าที่ของการทำงานออกเป็นหลายๆขั้ นตอน และแต่ละขั้นตอนก็มีหน้าที่เฉพาะอย่าง เท่านั้น และการทำงานจะมีการส่งต่อการทำงานจากขั้นตอนหนึ่งไปยังขั้นตอนต่อๆไป จนกระทั่งสิ้นสุดการทำงาน แต่จำนวนสายการผลิตที่มีเพียงหนึ่งนั้นสามารถประมวลผลคำสั่งได้มากที่สุดเพียงหนึ่งคำสั่งต่อหนึ่ง รอบสัญญาณนาฬิกา ต่อมาได้มีการเพิ่มจำนวนส ายการผลิตให้มากขึ้นเพื่อให้สามารถประมวลผลคำสั่งได้มากกว่าหนึ่งคำสั่งต่อหนึ่งสัญญาณนาฬิกา ซึ่งประสบความสำเร็จในปี ค.ศ. 1960 ในคอมพิวเตอร์สมรรถนะ สูงที่มีชื่อว่า CDC 6600 สำหรับปัญหาสำคัญที่ทำให้สถาปัตยกรรมแบบซุปเปอร์สเกลาร์ไม่สาม ารถประมวลผลได้แบบเต็มสมรรถนะ คือ ปัญหาความขึ้นต่อกันของข้อมูล ภายในโปรแกรม จะเป็นอุปสรรคต่อการประมวลผลแบบ ขนานซึ่งถือเป็น การลดประสิทธิภาพการทำงานของ สถาปัตยกรรมแบบซุปเปอร์สเกลาร์

รูปแบบของกระบวนการประมวลผลคำสั่งของสถาปัตยกรรมแบบซุปเปอร์สเกลาร์นั้นก็มีวิวัฒนาการมาจากสถาปัตยกรรมก่อนหน้านี้คือ การประมวลผลโปรแกรมจากการ ถอดรหัสคำสั่งที่เป็นเลขฐานสอง โดยที่รูปแบบการทำงานของคำสั่งของสถาปัตยกรรมแบบซุปเปอร์สเกลาร์นั้น ถูกออกแบบให้รองรับการทำงานของคอมพิวเตอร์ในยุคก่อนหน้านั้น รวมไปถึงเครื่องคอมพิวเตอร์ที่ต่างสถาปัตยกรรมแต่อยู่ในยุคเดียวกันนั้นได้ด้วย การประมวลผลคำสั่งนั้นจะต้องเป็นไปตามลำดับอย่างถูกต้อง โดยมีรูปแบบคือดึงคำสั่งที่ต้องการประมวลผลเข้ามาด้วยตัวชี้ลำดับคำสั่ง ต่อมาก็ทำการประมวลผลคำสั่งโดยที่หลังจากประมวลผลคำสั่งเสร็จก็ดำเนินการให้ตัวชี้คำสั่งชี้ไปยังคำสั่งถัดไปและวนแบบนี้ไปเรื่อยๆตามลำดับ หากมีการเปลี่ยน ทิศทางของโปรแกรม อันเนื่องมาจากการทำนายคำสั่งประเภทเงื่อนไขผิดพลาด โปรแกรมจะต้องยังคงทำงานถูกต้องอยู่

สำหรับสิ่ง สำคัญในการออกแบบ ที่หน่วยประมวลผลสรรถนะสูงนั้นคือ การแก้ไขปัญหาเรื่องความไม่เท่ากันของเวลาในการประมวลผลของคำ สั่งบางประเภท และความสามารถใน การประมวลผลคำสั่งต่างๆแบบ คู่ขนานกันได้อย่างถูกต้อ ง สิ่งที่เป็นองค์ประกอบหลัก ของหน่ว ยประมวลผลสรรมถนะสูงคือ

1. กลยุทธ์ในการดึงคำสั่งเข้ามาประมวลผลทีละหลายๆคำสั่ง และบางครั้งดึงคำสั่งที่คาดว่าจะประมวลผลเป็นคำสั่งต่อไปมารอไว้ หากเป็นคำสั่งประเภทเงื่อนไข

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

2. วิธีการในการคำนวณว่าคำสั่งใดควรนำมาประมวลผลถึงจะสามารถทำให้โปรแกรมประมวลผลไม่ผิดพลาด รวมกลไกการจัดการกับคำสั่งเหล่านั้นในขณะประมวลผล

3. กระบวนการในการจัดการให้การประมวลผลคำสั่งแบบขนานเกิดขึ้นได้

4. ทรัพยากรที่ใช้ในการประมวลผลแบบขนานซึ่งประกอบไปด้วย สายการผลิตแบบหลายสายและหน่วยความจำแบบอ้างอิงได้หลายรูปแบบ

5. วิธีการในการจัดการดึงข้อมูล และบันทึกข้อมูลลงในหน่วยความจำ

6. วิธีการในการยืนยันถึงความถูกต้องของลำดับของคำสั่งในการประมวลผล

ปัญหาที่สถาปัตยกรรมแบบซุปเปอร์สเกลาร์ได้ทำการแก้ไขเพื่อให้สามารถเปลี่ยนโปรแกรมที่ประมวลผลแบบตามลำดับไปเป็นการประมวลผลแบบขนานได้

1. ปัญหาการขึ้นต่อกันของคำสั่งในการประมวลผลแบบขนาน

โปรแกรมทั่วๆไปเกิดจากการแปล ภาษาโปรแกรมระดับสูงไปเป็นรหัสการทำงานของโปรแกรมแบบเลขฐานสอง (Program Binary) ซึ่งรหัสเลขฐานสองเหล่านี้จะประกอบไปด้วยชุดของคำสั่ งที่ใช้ในการประมวลผลโปรแกรม ซึ่งจะถูกประมวลผลไปตามลำดับการทำงานของโปรแกรม เมื่อเริ่มต้นประมวลผลโปรแกรมคำสั่งงานทำงานจะถูกดึงเข้าไปประมวลผลที่ละหลายคำสั่ง โดยที่คำสั่งเหล่านั้นจะมีการจั ดรูปแบบเป็นแบบไดนามิค และถูกส่ งไปประมวลผล ระบุโดยตัวชี้คำสั่ง (Program Counter) แต่ถ้าหากลำดับการประมวลผลของโปรแกรมมีการเปลี่ยนแปลงโดยอาจจะเกิดจากคำสั่งกระโดด ตัวชี้คำสั่งจะต้องสามารถชี้ไปยังคำสั่งในตำแหน่งถัดใหม่ได้

ตัวอย่างของปัญหาการขึ้นต่อกันของข้อมูลแบบภาษาระดับสูงแสดงให้เห็นดังภาพที่ 1 และภาษาแอสแซมบลีดังภาพที่ 2 ปัญหาการขึ้นต่อกันของข้อมูล (Data Dependency Hazard) สามารถมีอยู่หลายปัญหาด้วยกันดังนี้ คือ การอ่านข้อมูลหลังการเขียนข้อมูล (RAW) เขียนข้อมูลหลังการอ่านข้อมูล (WAR) และ การเขียนข้อมูลหลังการเขียนข้อมูล (WAW) ดังตัวอย่างในภาพที่ 3

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

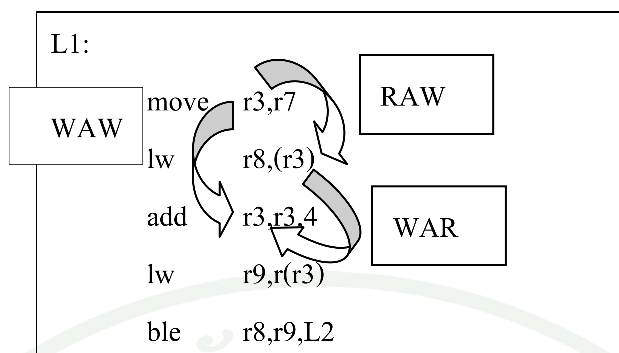
```
for ( i=0 ; i<last ; i++ ) {
        if(array[i] > array[i+1]) {
                temp       =      array[i];
                array[i]   =      array[i+1];
                array[i+1] =      temp;
                change++;
        }
}
```

ภาพที่ 1 ตัวอย่างปัญหาการขึ้นต่อกันของข้อมูลในภาษาซี

```
L1:
        move    r3,r7           #r3->array[1]
        lw      r8,(r3)         #load array[i]
        add     r3,r3,4         #r3->array[i+1]
        lw      r9,r(r3)        #load array[i+1]
        ble     r8,r9,L2        #branch array[i]->array[i+1]
        move    r3,r7           #r3->array[i]
        sw      r9,(r3)         #store array[i]
        add     r3,r3,4         #r3->array[i+1]
        sw      r8,(r3)         #store array[i+1]
        add     r5,r5,1         #change++
L2:
        add     r6,r6,1         #i++
        add     r7,r7,4         #r4->array[i]
        blt     r6,r4,L2        #branch i<last
```

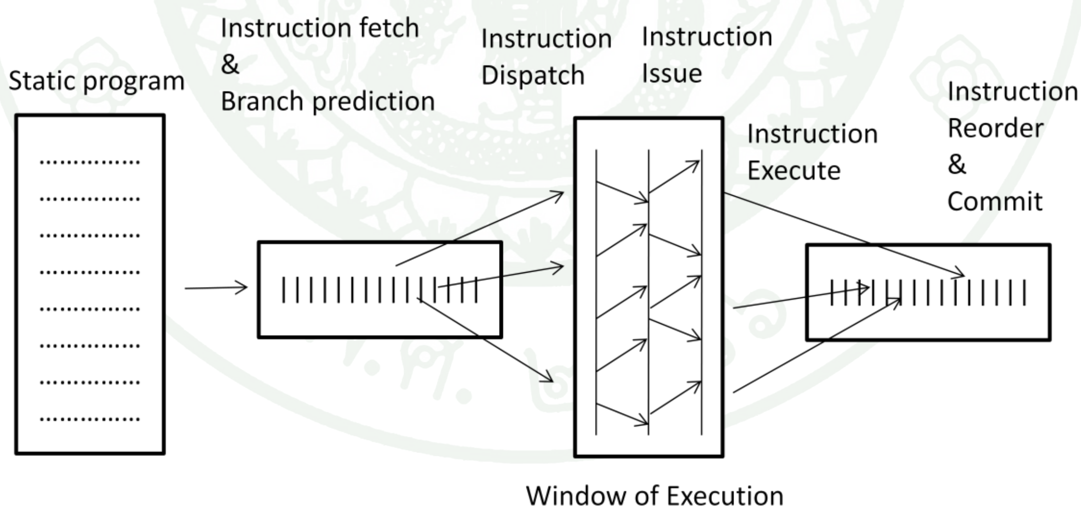
ภาพที่ 2 ตัวอย่างปัญหาการขึ้นต่อกันของข้อมูลในภาษาแอสแซมบลี (Assembly)

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์



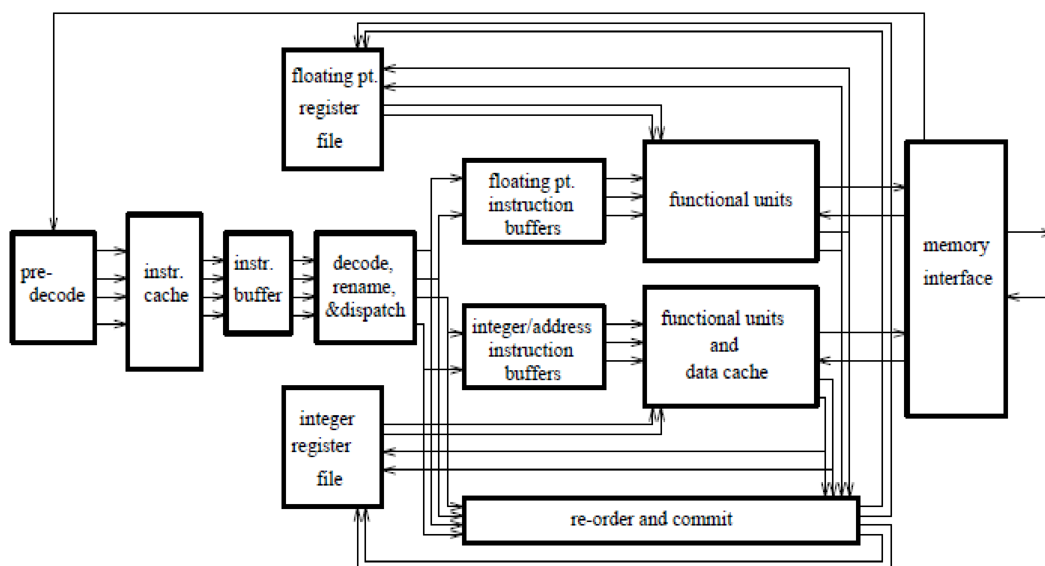
ภาพที่ 3 ตัวอย่างปัญหาการขึ้นต่อกันของข้อมูลแบบ RAW WAR และ WAW

กระบวนการทำงานแบบคู่ขนานของสถาปัตยกรรมแบบซุปเปอร์สเกลาร์นั้น เริ่มต้นจากโปรแกรมที่อยู่ในรูปแบบของรหัสเครื่องถูกดึงเข้าไปทำงานในสายการผลิต ซึ่งคำสั่งต่างๆที่ถูกดึงเข้าไปนั้นจะผ่านกระบวนการทำนายคำสั่งเงื่อนไข และมีการจัดลำดับคำสั่งใหม่แบบไดนามิคโดยส่วนอัจฉริยะเพื่อให้สามารถหลีกเลี่ยงปัญหาการขึ้นต่อกันของข้อมูลได้ โดยที่ส่วนที่ทำการประมวลผลอัจฉริยะดังกล่าวจะอยู่ในส่วนที่เรียกว่า Window of Execution ดังภาพที่ 4 สำหรับโครงสร้างฮาร์ดแวร์ของสถาปัตยกรรมแบบซุปเปอร์สเกลาร์นั้นแสดงให้เห็นดังภาพที่ 5



ภาพที่ 4 ตัวอย่างประกอบการทำงานของ Window of Execution

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์



ภาพที่ 5 โครงสร้างของฮาร์ดแวร์ภายในสถาปัตยกรรมแบบซุปเปอร์สเกลาร์

ที่มา: James *et al*., (1995)

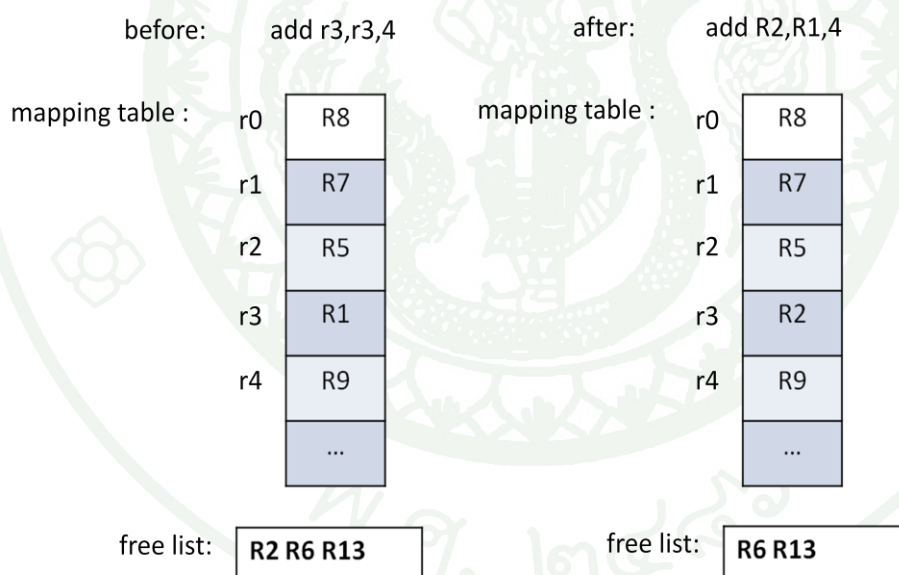
สำหรับกระบวนการในการดึงคำสั่งเข้ามาประมวลผลนั้น คำสั่งจะถูกค้นหาจาก หน่วยความจำแคชเป็นอันดับแรกซึ่งคำสั่งที่อยู่ในหน่วยความจำแคชนั้นจะเป็นคำสั่งที่ถูกใช้งาน บ่อย ซึ่งหน่วยความจำแคชนั้นจะแยกออกเป็นสองส่วน โดยที่ส่วนหนึ่งจะเป็นส่วนที่ใช้เก็บข้อมูล (Data Cache) และอีกส่วนหนึ่งจะใช้เป็นคำสั่งที่ใช้งานบ่อยซึ่งเรียกว่า Instruction Cache หากว่า คำสั่งที่ต้องการไม่ได้อยู่ในหน่วยความจำแคช หน่วยปฏิบัติการจะค้นหาในหน่วยความจำหลัก (RAM) และหน่วยความจำถาวร (Hard Disk) ตามลำดับของหน่วยความจำ โดยที่คำสั่งที่ถูกอ่าน ขึ้นมาจากถูกอ่านขึ้นมาเป็นกลุ่มก้อน (Block or Line)

อีกส่วนหนึ่งที่สำคัญคือส่วนของการทำนายคำสั่งกระโดด เพื่อให้โปรแกรมทำงานได้อย่าง ราบรื่นโดยที่ไม่มีการรอกันเมื่อเจอกับคำสั่งกระโดดและมีการเปลี่ยนทิศทางของโปรแกรม วิธีการ ของสถาปัตยกรรมแบบซุปเปอร์สเกลาร์คือการทำนายและเตรียมผลลัพธ์หากว่าการทำนายผิดพลาด เอาไว้รอ โดยทำทั้งสองอย่างในช่วงเวลาเดียวกัน ทำให้ไม่เกิดการรอกันเป็นเวลานานเกินไปหากมี การเปลี่ยนทิศทางการประมวลผลของโปรแกรม และผลของการทำนายจะถูกเก็บเป็นสถิติเอาไว้ เพื่อนำกับอัลกอริทึมที่ใช้ในการทำนายผลลัพธ์ของคำสั่งกระโดดคำสั่งถัดไป โดยผลลัพธ์จากการ

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ทำแบบนี้จะส่งผลให้โปรแกรมทำนายได้แม่นยำขึ้น ส่วนที่ใช้กับผลผลัพธ์การทำนายของคำสั่งเงื่อนไขเอาไว้มีชื่อเรียกว่า Branch History Table หรือเรียกอีกอย่างว่า Branch Prediction Table

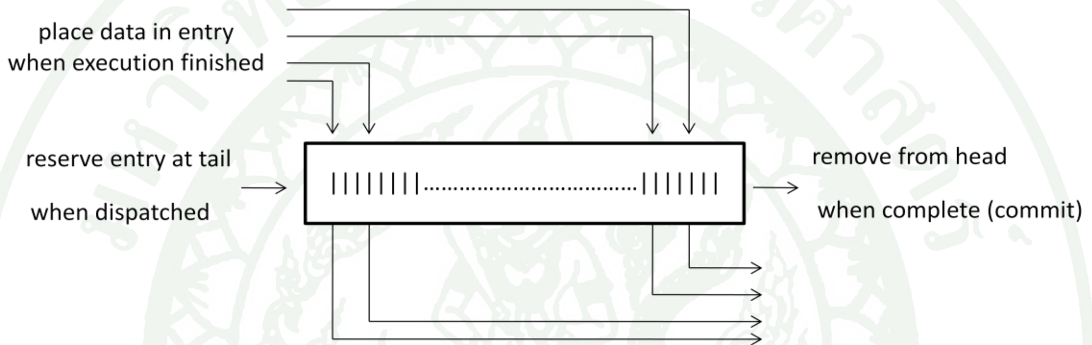
สำหรับส่วนของการแปลคำสั่งนั่นเริ่มจากการดึงคำสั่งที่อยู่ในส่วนของที่เก็บคำสั่งชั่วคราว (Instruction Fetch Buffer) ออกมาและทำการคำนวณเพื่อควบคุมการขึ้นต่อกันของข้อมูล (RAW, WAR and WAW Hazards) เพื่อใช้สามารถใช้ Register ได้อย่างราบรื่นเพราะ Register มีจำนวนจำกัดและต้องมีการนำกลับมาใช้ใหม่ โดยวิธีการแก้ไขปัญหาการขึ้นต่อกันของข้อมูลนั้นจะใช้เทคนิคที่เรียกว่า Register renaming โดยหลักการของวิธีการนี้คือการ หากมีการขึ้นต่อกันของข้อมูลหนึ่งเราจะทำการเก็บค่าของข้อมูลนั้นเอาไว้ใน Physical register ต่างๆ ที่ยังว่างอยู่และส่ง Register เหล่านั้นไปประมวลผลแยกกัน ซึ่งจะทำให้สามารถประมวลผลข้อมูลที่ขึ้นต่อกันได้ หลังจากนั้นก็จะทำงานรวมผลลัพธ์สุดท้ายของการประมวลผลข้อมูลดังกล่าวกลับมายัง Logical register เพื่อให้ข้อมูลถูกต้อง ซึ่งตัวอย่างของวิธีการนี้อยู่ในภาพที่ 6



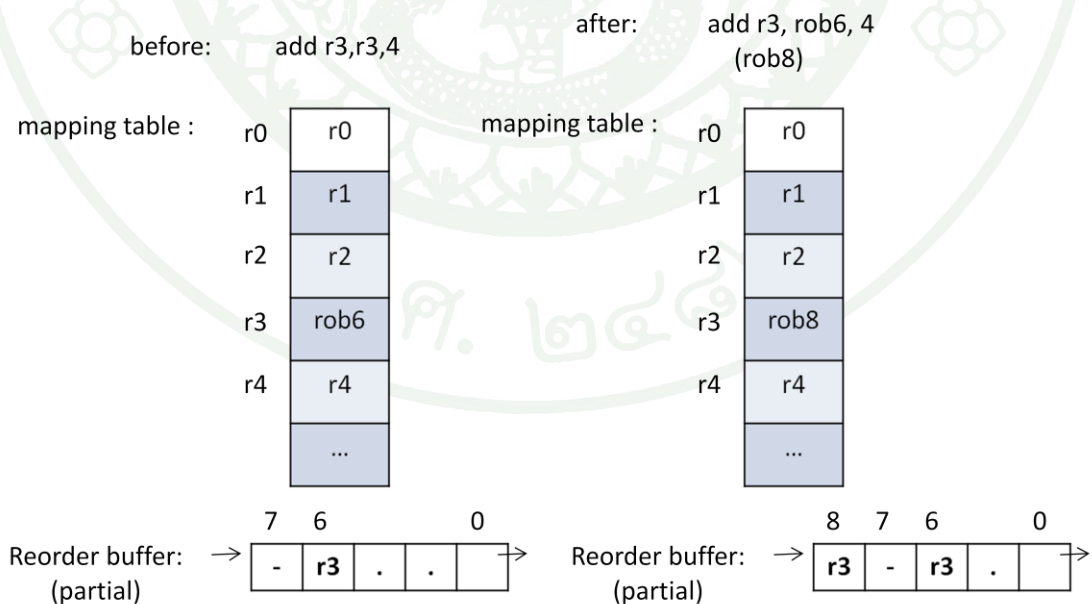
ภาพที่ 6 ตัวอย่างของวิธี Register renaming

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

สำหรับส่วนของ Physical register ที่ยังไม่ได้ Commit นั้นจะถูกเก็บใน Reorder Buffer (ROB) คำอธิบายอย่างง่ายของ ROB คือหน่วยคำจำแบบใครมาก่อนได้ใช้ก่อนหรือ FIFO ซึ่งสร้างขึ้นมาจากฮาร์ดแวร์ที่เป็นวงข้อมูลที่มีด้านหัวและด้านท้าย หากว่าคำสั่งไหนทำงานเสร็จแล้ว ROB จะถูกลบออก และผลลัพธ์ของการประมวลผลจะถูกเก็บลงใน Register file หากมีคำสั่งใหม่ถูกส่งเข้ามาประมวลผลก็จะมีข้อมูลใหม่ถูกส่งเข้ามาเก็บที่ ROB และก็จะมีการวนใช้งานแบบนี้ไปเรื่อยๆ ดังตัวอย่างประกอบดังภาพที่ 7 ส่วนตัวอย่างการใช้งาน Register renaming นั้นจะแสดงให้เห็นดังภาพที่ 8



ภาพที่ 7 ตัวอย่างของ Reorder Buffer



ภาพที่ 8 ตัวอย่างของ Register renaming ที่ทำงานร่วมกับ Reorder Buffer

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ในอดุมคตินั้นคำสั่งที่พร้อมจะประมวลผลคือคำสั่งที่ข้อมูลพร้อมประมวลผล แต่ก็ยังมีส่วนอื่นๆที่ยังมีข้อจำกัด ส่วนเหล่านั้นคือ หน่วยประมวลผล Register file และฮาร์ดแวร์ต่างๆ ตัวอย่างของคำสั่งที่ถูกประมวลผลแบบขนานในสถาปัตยกรรมแบบซุปเปอร์สเกลาร์แสดงดังภาพที่ 9

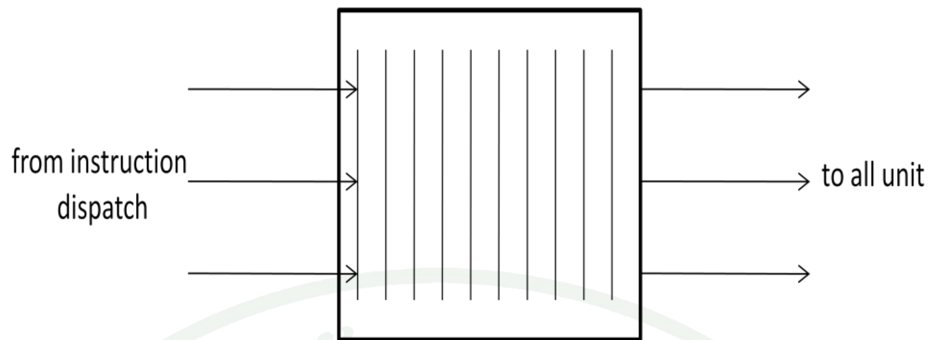
time ↓

INTEGER UNIT 1	INTEGER UNIT 2	MEMORY UNIT	BRANCH UNIT
move R1,r7			
add R2,R1,4		lw r8,(R1)	
move R3,r7		lw r9,(R2)	
add R4,R3,4			
add r5,r5,1	add r6,r6,1	sw r9,(R3)	ble r8,r9,L2
add r7,r7,4		sw r8,(R4)	blt r6,r4,L1

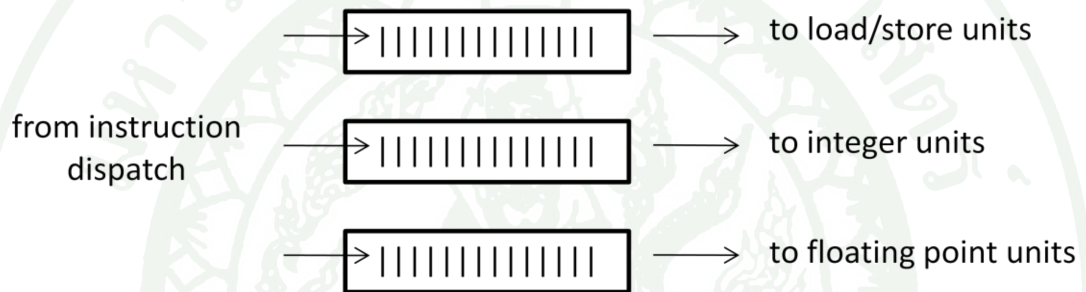
ภาพที่ 9 ตัวอย่างคำสั่งภาษาแอสแซมบลีของ Regiter renaming

การดึงคำสั่งเข้ามาประมวลผลในหน่วยความจำชั่วคราวนั้นมีหลักๆอยู่สามคือ การดึงเข้าสั่งแบบคำสั่งเดียว การดึงแบบหลายๆคำสั่งโดยแบ่งชนิดของคำสั่ง และการดึงแบบหลายๆคำสั่งโดยแบ่งชนิดของคำสั่งเข้ามาแบบขนาน

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

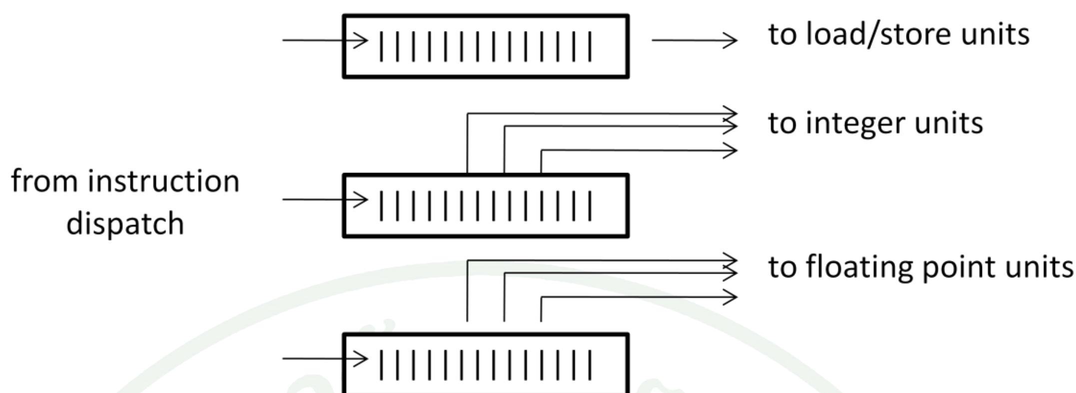


ภาพที่ 10 ตัวอย่างของ Single share queue



ภาพที่ 11 ตัวอย่างของ Muliple queue; one per instruction type

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์



ภาพที่ 12 ตัวอย่างของ Muliple reservation stations; one per instruction type

การทำงานของ Single share queue นั้นมีการทำงานแบบง่ายๆและไม่ซับซ้อน ไม่มีการเปลี่ยนลำดับคำสั่ง และไม่มีการทำ Register renaming การเลือกใช้ Register นั่นจะใช้วิธีการตรวจสอบข้อมูลในระดับบิทแบบง่ายๆเท่านั้นดังภาพที่ 10

การทำงานของ Muliple queue; one per instruction type นั้นมีการทำงานที่ซับซ้อนขึ้น โดยการดึงคำสั่งเข้ามาทำงานยังคงทำตามลำดับ แต่อาจจะมีการเปลี่ยนลำดับของคำสั่งแต่ละชนิดได้ และมีการใช้งานกระบวนการ Register renaming ดังภาพที่ 11

การทำงานของ Muliple reservation stations; one per instruction type นั้นมีการทำงานที่ซับซ้อนมากที่สุดเมื่อเทียบกับสองวิธีการก่อนหน้า อัลกอริทึมที่นำมาใช้มีชื่อเรียกว่า Tomasulo ซึ่งจะไม่ยึดติดกับการประมวลผลแบบลำดับอีกต่อไป และ มีการใช้วิธีเปลี่ยนลำดับคำสั่งในการเพิ่มสมรรถนะ มีการสำรวจข้อมูลทั้งหมดที่อาจจะนำมาใช้กับคำสั่งและดึงข้อมูลเหล่านั้ นเข้ามาเก็บไว้ใน Reservation station ก่อนการดึงคำสั่งเข้ามาประมวลจะมีการตรวจสอบว่าข้อมูลใดที่จะเอาไปใช้ทำให้ข้อมูลต่างๆที่จำเป็นต้องใช้ในการประมวลผลพร้อมเรียบร้อย เมื่อถึงเวลาที่คำสั่งถูกดึงเข้าไปประมวลผลดังแสดงตัวอย่างในภาพที่ 12

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

งานวิจัยที่เกี่ยวข้อง

สำหรับวิทยานิพนธ์นี้เป็นการวิจัยเกี่ยว กับการทดลองเพื่อเพิ่มสมรรถนะ ทางด้านเวลาในการประมวลผลของโปรแกรม LAMMPS ซึ่งเป็นโปรแกรมที่ใช้ในการจำลองการเคลื่อนที่ของโมเลกุลแบบไดนามิค ซึ่งมีงานวิจัยที่เกี่ยวข้องกับ การเพิ่มสมรรถนะ ของ LAMMPS ก่อนหน้านี้อยู่ 2 งานที่เป็นที่ยอมรับ โดยในส่วนนี้จะสรุปข้อดีข้อเสียของแต่ละวิธีพร้อมทั้งเสนอวิธีการ ที่เหมาะสมสำหรับการพัฒนาเพื่อใช้ในงานวิจัยนี้

1. งานวิจัยที่เกี่ยวข้องกับการเพิ่มสมรรถนะในการประมวลผลของ LAMMPS ด้วยวิธีการปรับปรุงโค้ดภาษาโปรแกรม (Optimization of LAMMPS)

การเพิ่มสมรรถนะในการประมวลผล ของ LAMMPS ด้วยวิธีการทุก อย่างที่จะสามารถทำได้จะทำให้โปรแกรมทำงานเร็วขึ้น เป็นอย่างมาก (Fischer *et al*., 2006) ได้ทำการเพิ่ม สมรรถนะ ในการประมวลผลด้วยวิธีการต่างๆ ซึ่งประกอบด้วยการใช้งานตัวแปลภาษาแบบเพิ่มสรร ถนะและการแก้ไขโค้ดของโปรแกรมด้วยวิธีต่างๆ คือ Loop-static branching conditions, Multi-dimensional C-arrays, Redundant integer, Floating-point operations และ Machine-specific instructions ก็รวมอยู่ด้วย

สำหรับการเพิ่มสมรรถนะด้วยวิธีทั้ง หมดที่กล่าวมานั้นสามารถเพิ่มส มรรถนะได้ 1.3 เท่าไปจนถึง 3.5 เท่า โดยที่สมรรถนะที่เพิ่มขึ้น มากนั้นมาจากการใช้คำสั่งพิเศษ ที่มีให้เฉพาะเครื่อง (Machine Specific Instructions) รวมอยู่ด้วย ซึ่งการใช้วิธีนี้ จะส่งผล เสียคือทำ ให้โปรแกรมที่ปรับปรุงสมรรถนะแล้วสามารถทำงานได้บนเครื่องที่ทำการปรับปรุงสมรรถนะได้เพียงเครื่องเดียวเท่านั้น ไม่สามารถนำไปประมวลผลยังเครื่องอื่นได้ แต่วิธีการนี้จะมีข้อดีคือสามารถเพิ่มสมรรถนะได้มากกว่าวิธีอื่นๆ

2. งานวิจัยที่เกี่ยวข้องกับการวัดและเพิ่มสมรรถนะของ LAMMPS บนแพลตฟอร์มแบบต่างๆ รวมไปถึงการเพิ่มสมรรถนะด้วยวิธีการปรับปรุงแพลตฟอร์มและโค้ดภาษาโปรแกรม (Performance analysis and optimization of LAMMPS on XCmaster, HPCx and Blue Gene)

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

การเพิ่มสมรรถนะการประมวลผล ของ LAMMPS ด้วยวิธีการ ปรับปรุงโค้ดโปรแกรม โดยใช้วิธีอย่างง่ายซึ่งใช้กันโดยทั่วไป (McKenna *et al*., 2007) ได้ทำการเพิ่ม สมรรถนะการประมวลผล ของ LAMMPS ด้วยวิธีการต่างๆ ซึ่งประกอบด้วย การเปิดใช้งาน ตัวแปลภาษาแบบเพิ่มสรรถนะ และการแก้ไข โค้ดของโปรแกรมด้วยวิธี อย่างง่ายเช่น Loop unrolling (Stoodley *et al*., 1996; Meisam *et al*., 2013) และ Lazy evaluation เป็นต้น

งานวิจัยของ McKenna มีความคลายคลึงกับงานวิจัยชิ้นนี้ แต่งานวิจัยชิ้นนี้แสดงให้เห็นว่ายังสามารถปรับปรุงสมรรถนะเพิ่มเติมได้อีกมาก รวมไปถึงยังอธิบายถึงรายละเอียดว่าเหตุใดผลลัพธ์ถึงได้ดีขึ้น

3. งานวิจัยที่เกี่ยวข้องกับการเพิ่มสมรรถนะการประมวลผลของโปรแกรมโดยวิธีการเปลี่ยนลำดับของคำสั่งดึงข้อมูลจากหน่วยความจำ (Load Instruction Characterization and Acceleration of the BioPerf Programs)

การทดลองเพิ่ม สมรรถนะ ทางด้านความเร็วของ โปรแกรม โดยการเปลี่ ยนลำดับของคำสั่งดึงข้อมูลจากหน่ว ยความจำ (Ratanaworabhan *et al*., 2006) ได้ศึกษาถึงวิธีแก้ปัญหาการขึ้นต่อกันของข้อมูล (True Dependency) โดยใช้วิธีเปลี่ยนแปลงลำดับคำสั่งของคำสั่งดึงข้อมูล จากหน่วยความจำ (Load Instruction) งานวิจัยนี้ได้นำเอาเทคนิคและวิธีการจากงานของ Ratanaworab-han มาใช้ รวม ไปถึงนำไปประยุกต์ใช้กับคำ สั่งที่ทำงานกับเลขทศนิยม (Floating-point operation) ซึ่งเป็นคำสั่งที่ต้องใช้หลายสัญญาณนาฬิกาเพื่อให้สามารถทำง านได้สำเร็จเช่นเดียวกับคำสั่ง Load ข้อมูลจากหน่วยความจำ หรือเรียกได้ว่าคำสั่งดึงข้อมูลและคำสั่งที่ทำงานกับเลขทศนิยมเป็นคำสั่งที่สิ้นเปลืองสัญญาณนาฬิกาแบบเดียวกันนั่นเอง

4. สรุปข้อดีข้อเสียของงานวิจัยก่อนหน้า

4.1 งานวิจัยที่เกี่ยวข้องกับการเพิ่มสมรรถนะของ LAMMPS ด้วยวิธีการปรับปรุงโค้ดภาษาโปรแกรม (Fischer *et al*., 2006)

4.1.1. ข้อดี ที่เกี่ยวข้องกับการ เพิ่มสมรรถนะ ของ LAMMPS ด้วยวิธีการปรับปรุงโค้ดภาษาโปรแกรม (Fischer *et al*., 2006) คือสามารถปรับปรุงสมรรถนะของโปรแกรมได้มากกว่าวิธี

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

อื่นๆ เพราะมีการใช้งานคำสั่งเฉพาะที่มีเฉพาะเครื่องใดเครื่องหนึ่งเท่านั้น ซึ่ง วิธีการนี้ทำให้ได้สมรรถนะเหนือกว่าวิธีอื่นๆเป็นอย่างมาก

4.1.2. ข้อเสีย การปรับปรุงสมรรถนะโดยใช้คำสั่งพิเศษที่มีเฉพาะเครื่องใดเครื่องหนึ่งนั้นนั้นสามารถทำได้ยากมากๆ และข้อเสียอีกประการคือ หากใช้คำสั่งพิเศษเหล่านี้แล้ว โปรแกรมหลังจากการเพิ่มสมรรถนะเป็นที่เรียบร้อยแล้วจะไม่สามารถนำไปทำงานยังเครื่องอื่นๆได้

4.2 งานวิจัยที่เกี่ยวข้องกับการวัดและเพิ่มสมรรถนะของ LAMMPS บนแพลตฟอร์มแบบต่างๆ รวมไปถึงการเพิ่มสมรรถนะด้วยวิธีการปรับปรุงโค้ดภาษาโปรแกรมโดยใช้วิธีการอย่างง่าย (Geraldine McKenna, 2007)

4.2.1 ข้อดี ใช้วิธีการปรับปรุงสมรรถนะอย่างง่าย ที่มีใช้โดยทั่วไป ทำให้โปรแกรมยังคงสามารถทำงานบนแพลตฟอร์มใดก็ได้

4.2.2 ข้อเสีย งานวิจัยชิ้นนี้ ยังไม่สามารถปรับปรุงสมรรถนะ ได้น่าพอใจนัก เพราะผลลัพธ์ยังเป็นตัวเลขที่น้อยเกินไปเพียงแค่ประมาณ 2 เปอร์เซ็นต์เท่านั้น เนื่องจากวิธีการที่ใช้เป็นวิธีง่ายๆที่มีการใช้งานทั่วไป ซึ่ง บางวิธีตัวแปลภาษาโปรแกรมก็ได้ทำการเพิ่มสมรรถนะให้ในระหว่างการแปลภาษา

4.3 งานวิจัยที่เกี่ยวข้องกับการปรับปรุงสมรรถนะการประมวลผลของโปรแกรมโดยวิธีการเปลี่ยนลำดับของคำสั่งดึงข้อมูลจากหน่วยความจำ

4.3.1 ข้อดี สามารถนำมาใช้ในการปรับปรุง สมรรถนะของโปรแกรมที่มีพฤติ กรรมของการขึ้นต่อกันของข้อมูล อันเนื่องมาจากคำสั่งที่มีลักษณะสิ้นเปลืองจำนวนสัญญาณนาฬิกา ในงานวิจัยชิ้นนี้สนใจคำสั่งประเภทดึงข้อมูลจากหน่วยความจำและคำสั่งที่ทำงานกับเลขทศนิยม

4.3.2 ข้อเสีย หลังจากปรับปรุงลำดับคำสั่งแล้วอาจจะทำให้โค้ดโปรแกรมเข้าใจได้ยากขึ้น ซึ่งอาจจะส่งผลกระทบต่อการพัฒนาโปรแกรมในอนาคตได้

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

5. แนวทางแก้ไขปัญหา

จากที่ได้สรุปมานั้นจะเห็นได้ว่า วิธีการปรับปรุง สมรรถนะการประมวลผล ของ LAMMPS ด้วยวิธีอย่างง่าย นั้นยังไม่ ดีเท่าที่ควร โดยมีข้อ สงสัย ว่าอาจจะยังทำไม่ถึงที่สุดและน่าจะยังมี จุดที่น่าจะสามารถ ทำการเพิ่มสมรรถนะเพิ่มเติมได้อีก ในขณะที่การปรับปรุงโดยใช้ชุดคำสั่งพิเศษนั้นเห็นผลดีขึ้นเป็นอย่างมาก แต่กลับส่งผลเสียคือโปรแกรมสามารถทำงานได้บนเครื่องที่ทำการปรับปรุงสมรรถนะได้เพียงแพลตฟอร์มเดียวเท่านั้น

ในวิทยานิพนธ์เล่มนี้ ผู้วิจัยจึงได้นำเสนอ วิธีการปรับปรุงสมรรถนะการประมวลผลของ LAMMPS โดยใช้เทคนิคอย่างง่ายที่มีใช้กันอย่างแพร่หลาย โดยที่ผลลัพธ์หลังจากการปรับปรุงสมรรถนะจะต้องดีขึ้นอย่างน่าพอใจ รวมไปถึงต้องสามารถอธิบายถึงสาเหตุที่ทำให้สมรรถนะของโปรแกรมดีขึ้นได้ด้วยว่ามาจากสาเหตุใด

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

อุปกรณ์และวิธีการ

อุปกรณ์

1. ฮาร์ดแวร์

1.1. เครื่องคอมพิวเตอร์แบบคลัสเตอร์ที่ทำงานรวมกับเทคโนโลยี CUDA จำนวน 1 เครื่อง ซึ่งประกอบด้วยรายละเอียดของอุปกรณ์ดังต่อไปนี้

1.1.1. ซีพียู2xCPU Intel(R) Xeon(R) CPU E5620 @ 2.40GHz

1.1.2. จีพียูNvidia Cuda compilation tools, release 3.2, V0.2.1221 (448 Cores)

1.1.3. หน่วยความจำหลัก 16.00 GB

1.1.4. ฮาร์ดดิสก์ขนาด 20TB

1.2. เครื่องคอมพิวเตอร์ชนิดพกพาได้จำนวน 1 เครื่อง ประกอบด้วย รายละเอียดของอุปกรณ์ดังต่อไปนี้

1.2.1. ซีพียูIntel 2.9 GHz Intel Core i7

1.2.2. หน่วยความจำหลัก 8.00 GB

1.2.3. ฮาร์ดดิสก์ขนาด 500 MB

1.3. เครื่องคอมพิวเตอร์ชนิดพกพาได้จำนวน 1 เครื่อง ประกอบด้วย รายละเอียดของอุปกรณ์ดังต่อไปนี้

1.3.1. ซีพียูIntel 2.9 GHz Intel Core i7

1.3.2. หน่วยความจำหลัก 2.00 GB

1.3.3. ฮาร์ดดิสก์ขนาด 500 MB

2. ซอฟต์แวร์

2.1. ซอร์สโค้ดของโปรแกรม LAMMPS ที่อ้างอิงวันที่ 2 กุมภาพันธ์ 2556

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

2.2. ระบบปฏิบัติการ IBM Rocks Cluster 5.4 (x86-64)

2.2.1. คอมไพเลอร์ภาษา C - gcc version 4.1.2 20080704 (Red Hat 4.1.2-48)

2.2.2. OpenMPI 1.4.3 Libraries

2.2.3. จีพียูคอมไพเลอร์ Nvidia Cuda compilation tools, release 3.2, V0.2.1221 (448 Cores)

2.3. ระบบปฏิบัติการ Mac OS X 10.8.5

2.3.1. คอมไพเลอร์ภาษา C - gcc version 4.2.1 (Based on Apple Inc. build 5658)

2.3.2. OpenMPI 1.6.5 Libraries

2.4. ระบบปฏิบัติการ Ubuntu 64 12.04.3 LTS

2.4.1. คอมไพเลอร์ภาษา C - gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5)

2.4.2. OpenMPI 1.4.1 Libraries

วิธีการ

การเพิ่มสมรรถนะของ LAMMPS ในงานวิจัยชิ้นนี้ประกอบด้วยขั้นตอนดังต่อไปนี้

1. ศึกษาพฤติกรรมของของโปรแกรม LAMMPS โดยการโปรแกรมวัดผลโปรไฟล์ของโปรแกรม

1.1. ใช้โปรแกรม Instrument tool 5.0.1 ซึ่งเป็นโปรแกรมที่ใช้วัดพฤติกรรมการประมวลผลของโปรแกรม มาทำการ วัดพฤติกรรมการประมวลผลของโปรแกรม LAMMPS ว่ามีการประมวลผลส่วนใดของโปรแกรมบ้าง และ ส่วนใดที่มีการประมวลผลมากกว่าส่วนอื่นๆ ก็ควรที่จะได้รับการปรับปรุงสมรรถนะ ซึ่งจากตารางที่ 1 ด้านล่างจะเห็นว่ามีอยู่ 3 ส่วนหลักๆ ด้วยกันคือ Pair, Kspce และ Neigh โดยที่ทั้งสามส่วนมีการใช้งาน CPU รวมกันเกินกว่า 92 เปอร์เซ็นต์

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ตารางที่ 1 แสดงถึงข้อมูลการประมวลผลแต่ละส่วนของโปรแกรม LAMMPS (LAMMPS Profile)

LAMMPS Profile	
Modules	***CPU time Percentage***
Pair	67.36
Kspce	16.69
Neigh	8.51
Other	3.66
Comm	3.60
Bond	0.17
Output	0.01

1.2. ทำการเพิ่มสมรรถนะโดยวิธีการลดจำนวนการวนซ้ำของคำสั่งวนซ้ำ (Loop unrolling)

Loop unrolling เป็นวิธีการเพิ่มสมรรถนะของโปรแกรมโดยวิธีการลดจำนวนคำสั่งตรวจสอบเงื่อนไข (Conditional instruction) ที่จะทำให้คำสั่งในการทำงานลดลง และยังลดจำนวนของคำสั่งเปลี่ยนแปลงทิศทางของโปรแกรม (Branch instruction) อีกด้วย ซึ่งจะได้ข้อดีจากการที่คำสั่งลดน้อยลงและยังลดปัญหาการทำนาย Branch ผิด (Branch mis-prediction penalty) ของหน่วยประมวลผลอีกด้วย

สำหรับวิธี Loop unrolling นั้นโดยปรกติสามารถทำได้โดยอัตโนมัติด้วยตัวแปลภาษาที่สามารถปรับปรุงสมรรถนะ ของโปรแกรมได้ แต่ว่าการให้ตัวแปลภาษาเป็นตัวจัดการนั้นจะทำให้เราไม่สามารถเข้าถึงสมรรถนะสูงสุดของการประมวลผลได้ เนื่องจากตัวแปลภาษานั้นจะใช้ค่าคงที่บางค่าสำหรับค่าการวนซ้ำ (Loop step number) ในการทำ Loop unrolling เท่านั้น แต่งานวิจัยชิ้นนี้จะใช้ตัวเลขค่าการวนซ้ำที่ทำให้ได้สมรรถนะดีที่สุดจากการทดลองมาใช้ในโปรแกรม ซึ่งจะส่งผลให้ได้สมรรถนะที่ดีกว่าการที่ตัวแปลภาษาเป็นตัวจัดการให้

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

จากตัวอย่าง ในภาพที่ 13 ด้านล่างแสดงให้เห็นคำสั่งการวนซ้ำ ก่อนที่จะทำ Loop unrolling และในภาพที่ 14 หลังจากที่ทำการ Loop unrolling เรียบร้อยแล้ว ซึ่งจะเห็นว่าคำสั่งการวนซ้ำถูกเปลี่ยนไปเป็นคำสั่ง แบบไม่มีการวนซ้ำเลย แม้แต่รอบเดียว ทำให้คำสั่งตรวจสอบเงื่อนไขและคำสั่งเปลี่ยนแปลงทิศทางของโปรแกรมถูกกำจัดออกไปจนหมด

```
(1) for (i = 0; i < 6; i++) {
(2)     virial[i]=0.5*qscale*volume*virial_all[i];
(3) }
```

ภาพที่ 13 ตัวอย่างคำสั่งวนซ้ำแบบ For-loop

```
(1) tmp_double0=0.5*qscale*volume;
(2) virial[0]=tmp_double0*virial_all[0];
(3) virial[1]=tmp_double0*virial_all[0];
(4) virial[2]=tmp_double0*virial_all[0];
(5) virial[3]=tmp_double0*virial_all[0];
```

ภาพที่ 14 ตัวอย่างคำสั่งวนซ้ำหลังจากทำ Loop unrolling แล้ว

1.3. ทำการเพิ่มสมรรถนะ โดยวิธีเปลี่ยนลำดับคำสั่งดึงข้อมูลจากหน่วยความจำและคำสั่งที่กระทำกับเลขทศนิยม ที่มีการขึ้นต่อกันของข้อมูล

วิธีการเปลี่ยนแปลงลำดับคำสั่งที่มีการขึ้นต่อกันของข้อมูล (Grossman *et al.*, 2000; Wang *et al.*, 2002; Mutlu *et al.*, 2005; Srinivasan *et al.*, 2010) (Out-of-Order) ในงานวิจัยชิ้นนี้มุ่งเน้นไปที่คำสั่งดึงข้อมูลจากหน่วยความจำและคำสั่งที่กระทำกับเลขทศนิยม เนื่องจากคำสั่งเหล่านี้เป็นคำสั่งที่ใช้ CPU time เป็นจำนวนมากหรือเรียกอีกอย่างว่าเป็นคำสั่งที่สิ้นเปลืองนั่นเอง (Expensive instruction)

เมื่อใดก็ตามที่คำสั่งเหล่านี้มีการขึ้นต่อกันของข้อมูลเป็นอย่างมากจะทำให้มีโอกาสที่หน่วยประมวลผล จะหยุดทำงาน (Processor stall) เนื่องจากต้องรอผลลัพธ์จากคำสั่งที่สิ้นเปลืองเหล่านั้น จากตัวอย่างในภาพที่ 15 เป็นตัวอย่างโค้ดก่อนทำการเปลี่ยนแปลงตำแหน่ง และในภาพที่

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

16 เป็นโค้ดที่ได้ทำการเปลี่ยนแปลงตำแหน่งเรียบร้อยแล้ว ถ้าสังเกตดูจะเห็นว่าชุดคำสั่งในกล่องสีเหลืองและกล่องสีฟ้านั้นจะเป็นอิสระต่อกัน

```
(1) table = etable[itable]+fraction*detable[itable];
(2) ecoul = qtmp*q[j]*table;
```

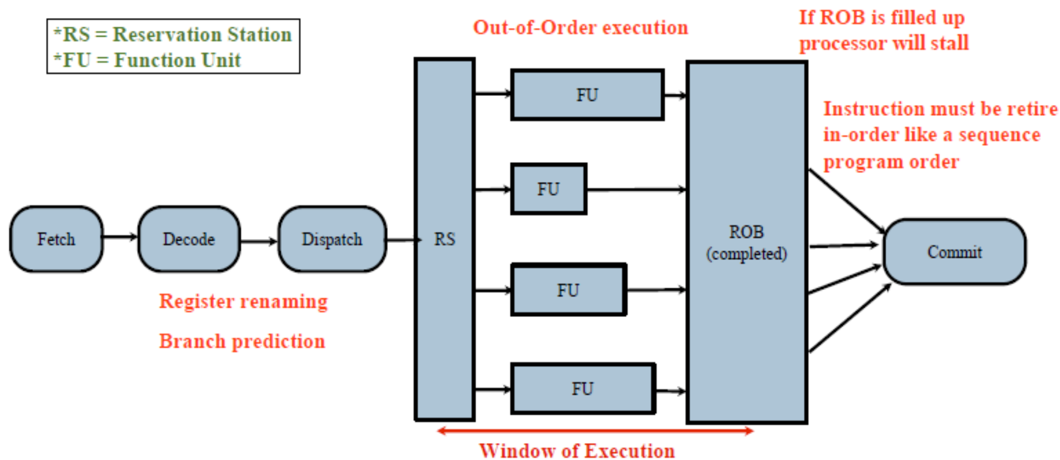
ภาพที่ 15 ตัวอย่างชุดคำสั่งก่อนทำการสลับลำดับคำสั่ง

```
(1)  tmp_double0 = fraction * detable[itable];
(2)  tmp_double1=qtmp*q[j];
(3)  table = etable[itable]+tmp_double0;
(4)  ecoul = tmp_double1*table;
```

ภาพที่ 16 ตัวอย่างชุดคำสั่งหลังทำการสลับลำดับคำสั่ง

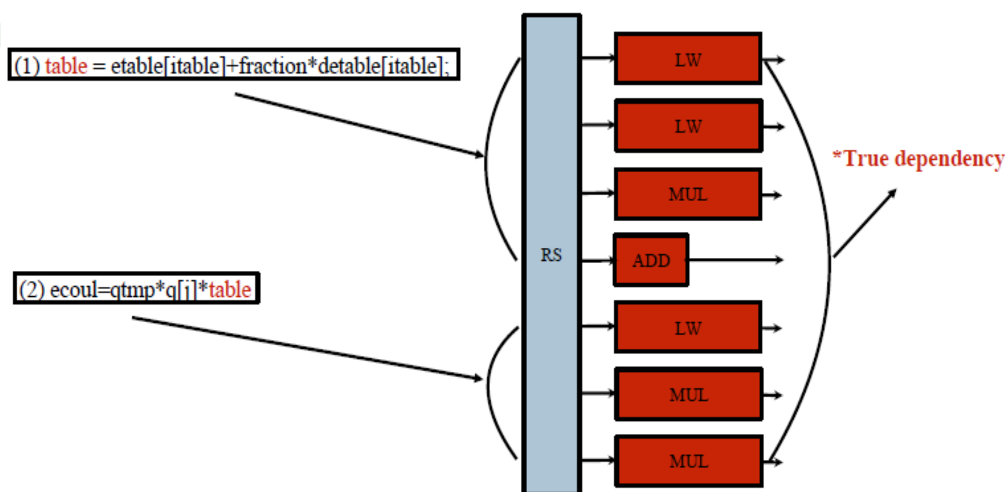
ต่อไปนี้จะเป็นการอธิบายอย่างละเอียดว่าเหตุใดการเปลี่ยนแปลงตำแหน่งของคำสั่งจึงทำให้โปรแกรมทำงานรวดเร็วขึ้น เริ่มจากการระบุถึงจุดที่เป็นข้อจำกัดของสถาปัตยกรรมแบบซุปเปอร์สเกลาร์กันก่อน จากภาพที่ 17 แสดงให้เห็น ว่าจุดที่เป็นข้อจำกัด มีอยู่ด้วยกันสองจุด คือ FU (Function Unit) และขนาดของ ROB (Reorder buffer) ซึ่งในสถานการณ์ที่คำสั่งที่สิ้นเปลืองมีการขึ้นต่อกัน ของข้อมูล มากๆก็จะมีโอกาสทำให้ ROB เต็ม ซึ่งจะเป็นสาเหตุให้หน่วยประมวลผลหยุดทำงานลง

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์



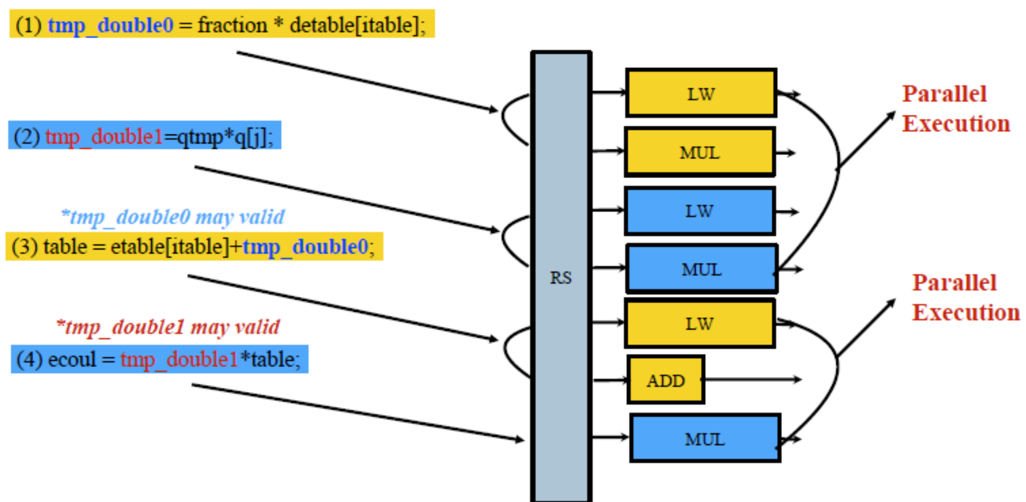
ภาพที่ 17 แสดงถึงส่วนที่เป็นข้อจำกัดของสถาปัตยกรรมแบบซุปเปอร์สเกลาร์

จากภาพที่ 18 จะเห็นว่า เมื่อชุดคำสั่งเดิมที่ยังไม่ได้ทำการสลับตำแหน่งถูกดึงเข้าไปประมวลผลยังหน่วยประมวลผลแบบซุปเปอร์สเกลาร์ ก็จะเกิดปัญหาการขึ้นต่อกันของข้อมูล (True dependency) ซึ่งจะทำให้หน่วยประมวลผลหยุดชะงักลง แต่ในภาพที่ 19 จะเห็นว่าหลังจากที่มีการสลับตำแหน่งของชุดคำสั่งแล้ว ส่งผลให้ชุดคำสั่งในกล่องสีเหลืองและสีฟ้าเป็นอิสระต่อกัน ซึ่งจะส่งผลให้ชุดคำสั่งทั้งสองสามารถทำงานไปพร้อมๆกันได้ในหน่วยประมวลผล ทำให้โปรแกรมมีสมรรถนะที่ดีขึ้น



ภาพที่ 18 แสดงถึงปัญหาการขึ้นต่อกันของข้อมูลของคำสั่งโปรแกรมก่อนการสลับลำดับคำสั่ง

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์



ภาพที่ 19 แสดงถึงคำสั่งโปรแกรมที่สามารถทำงานแบบขนานไปพร้อมๆได้

1.4. Reducing branch misprediction

เทคนิคการลดจำนวนคำสั่งเปลี่ยนแปลงทิศทางของโปรแกรมจะส่งผลทำให้โปรแกรมลดจำนวนคำสั่งประเภทเงื่อนไขลงไปได้ รวมไปถึงการลดเวลาที่ต้องสูญเสียไปเนื่องจากการเดาคำสั่งเปลี่ยนแปลงทิศทางของโปรแกรมผิดพลาดลงไปได้ จากภาพที่ 20 แสดงให้เห็นโค้ดของโปรแกรมที่ยังไม่ได้ทำการปรับปรุง จะเห็นว่าชุดคำสั่งเหล่านี้ประกอบไปด้วย คำสั่งประเภท if-else ซึ่งเป็นคำสั่งประเภทเงื่อนไขจำนวนมาก รวมไปถึงทิศทางของโปรแกรมที่สามารถเป็นไปได้มีถึง 3 ทิศทางด้วยกัน

ซึ่งหลังจากทำการปรับปรุงสมรรถนะของโปรแกรมแล้วดังภาพที่ 21 จะเห็นว่าคำสั่งประเภท if-else นั้นถูกกำจัดออกจนหมด โดยการใช้เทคนิคที่เรียกว่า Bitwise operation (Kraeling *et al*., 1996; Henry *et al*., 2012) และทิศทางการประมวลผลของโปรแกรมก็มีเพียงทิศทางเดียวเท่านั้น หรือสรุปได้ว่าการประมวลผลของชุดคำสั่งหลังการปรับปรุง จะมีเพียงแต่การคำนวณ ณเพียงอย่างเดียวเท่านั้น และการทำงานของคำสั่งใหม่จะแสดงดังภาพที่ 22

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

```
(1)        if(special_flag[1]==0)
(2)            return-1;
(3)        else if(special_flag[1] == 1)
(4)            return 0;
(5)        else
(6)            return 1;
```

ภาพที่ 20 แสดงถึงคำสั่งกระโดดต่อเนื่องหลายคำสั่ง

```
(1)    x = special_flag[1];
(2)    cond1 = (x == 0);
(3)    cond2 = (x == 1);
(4)    cond3 = !cond1 && !cond2;
(5)    shift = sizeof(int)*8 - 1;
(6)    mask = (cond3 << shift) >> shift;
(7)    return ((x-1) & ~mask) | 1 & mask;
```

ภาพที่ 21 แสดงถึงคำสั่งโปรแกรมหลังจากคำสั่งกระโดดได้ถูกจำกัดไป

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

<table>
<tr><th>ค่าของตัวแปร x</th><th>Value</th></tr>
<tr><td>x=0</td><td>x=0
cond1=1
cond2=0
cond3=0
shift=31
mask=0
# ((x-1) & ~mask) | 1 & mask
# ((0-1) & ~(0)) | 1 & 0
# return -1</td></tr>
<tr><td>x=1</td><td>x=1
cond1=0
cond2=1
cond3=0
shift=31
mask=0
# ((x-1) & ~mask) | 1 & mask
# ((1-1) & ~(0)) | 1 & 0
# return 0</td></tr>
<tr><td>x=2</td><td>x=2
cond1=0
cond2=1
cond3=0
shift=31
mask=fffffff
# ((x-1) & ~mask) | 1 & mask
# ((2-1) & ~(-1)) | 1 & (-1)
# return 1</td></tr>
</table>

ภาพที่ 22 แสดงถึงการทำงานของคำสั่งโปรแกรมแบบละเอียดหลังจากคำสั่งกระโดดได้ถูกจำกัดออกไป

1.5. "Lazy" evaluation

วิธีการปรับปรุงสมรรถนะด้วยวิธีนี้คือการพยายามทำให้ชุดคำสั่งที่จำเป็นต่อการทำงานของโปรแกรมจริงๆเท่านั่ ได้ประมวลผล คำสั่ง ใดที่ไม่ได้ใช้ในการทำงาน ของโปรแกรมจะไม่ให้คำสั่งนั้นได้ประมวลผล

จากตัวอย่างของโค้ดในภาพที่ 23 เป็นชุดคำสั่งที่ยังไม่ได้ทำการปรับปรุงสมรรถนะจะเห็นว่า new_pair จะทำการประมวลผลทุกๆ ครั้งที่ชุดคำสั่งเหล่านี้มีการ ประมวลผล แต่ว่าค่าของ newton_pair จะถูกใช้ก็ต่อเมื่อ ev_flag เป็นจริงเพียงเท่านั้น ดังนั้นเมื่อเราทำการย้าย ตำแหน่งของ new_pair เข้าไปอยู่ภายในกรอบการทำงานของ ev_flag ดังภาพที่ 24 จะทำให้ newton_pair ถูกประมวลผลเมื่อค่าของ newton_pair มีความจำเป็นต่อการทำงานของโปรแกรมจริงๆเท่านั้น

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

```
(1)    int newton_pair=force->newton_pair;
(2)    if (evflag) {
(3)        ev_tally(i,j,nlocal,newton_pair,
           evdwl,ecoul,fpair,delx,dely,delz);
(4)    }
```

ภาพที่ 23 แสดงถึงคำสั่งโปรแกรมก่อนทำการแก้ไขปัญหา Lazy evaluation

```
(1)    if (evflag) {
(2)        int newton_pair = force->newton_pair;
(3)        ev_tally(i,j,nlocal,newton_pair,
           evdwl,ecoul,fpair,delx,dely,delz);
(4)    }
```

ภาพที่ 24 แสดงถึงคำสั่งโปรแกรมกหลังทำการแก้ไขปัญหา Lazy evaluation

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ผลและวิจารณ์

ผล

1. วิธีวัดผลการทดลอง

ผู้วิจัยได้ใช้เครื่องคอมพิวเตอร์เพื่อวัดผลจำนวนสาม แพลตฟอร์มด้วยกันคือ Rock Cluster, Mac OSX และ Ubuntu ซึ่งข้อมูลโดยละเอี ยดของแต่ละแพลตฟอร์มได้แสดงไว้ในตารางที่ 2 ส่วนข้อมูลที่นำมาใช้ในการวัดผลคือ 5-mer peptide ซึ่งมีการเขย่าจำนวน 100,000 ครั้ง โดยผลการทดลองที่วัดเวลาในการประมวลผลเป็นวินาทีแสดงไว้ในตารางที่ 3

ตารางที่ 2 แสดงข้อมูลของแต่ละแพลตฟอร์มที่นำมาใช้ในการทดลองครั้งนี้

Platform	Hardware	Software
Rocks Cluster	2xCPU Intel(R) Xeon(R) CPU E5620 Quad Core @ 2.40GHz	OS: Rocks Cluster 5.4 (x86-64)
	RAM 16 GB	Compiler: gcc version 4.1.2 20080704 (Red Hat 4.1.2-48)
		MPI: OpenMPI 1.4.3
Mac OSX	CPU Intel(R) 3520M Dual-core @ 2.9GHz	OS: Mac OS X 10.8.5
	RAM 8 GB	Compiler: gcc version 4.2.1 (Based on Apple Inc. build 5658)
		MPI: OpenMPI 1.6.5
Ubuntu	CPU Intel(R) 3520M Dual-core @ 2.9GHz	OS: Ubuntu 64 12.04.3 LTS
	RAM 2 GB	Compiler: gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5)
		MPI: OpenMPI 1.4.1

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ดำเนินการคอมไพล์โปรแกรม LAMMPS ที่ได้ดำเนินการปรับปรุงสมรรถนะแล้วลงยังแพลตฟอร์มที่ต้องการจะทำการวัดผล ซึ่งในงานวิจัยชิ้ นนี้มี 3 แพลตฟอร์มคือ IBM Cluster ที่ทำงานร่วมกับ CUDA, Mac OSX และ Ubuntu Linux คำสั่งที่ใช้ในการแปล LAMMPS สำหรับการทำงานแบบขนานคือ make openmpi และแบบลำดับคือ make serial ซึ่งตัวอย่างของคำสั่งสำหรับการทำงานแบบลำดับแสดงไว้ในภาพที่ 25

```
compute_msd_molecule.cpp                  fix_press_berendsen.cpp
compute_msd_molecule.h                    fix_press_berendsen.h
compute_pair.cpp                          fix_print.cpp
compute_pair.h                            fix_print.h
veerapongs-MacBook-Pro:src veerapong$ make serial
g++ -O -DLAMMPS_GZIP  -I../STUBS     -M pair_comb.cpp > pair_comb.d
g++ -O  -DLAMMPS_GZIP  -I../STUBS     -c pair_comb.cpp
```

ภาพที่ 25 แสดงถึงคำสั่งแปลภาษาของโปรแกรม LAMMPS แบบลำดับ

หลังจากทำการแปลโปรแกรม LAMMPS เป็นที่เรียบร้อยแล้วจะทำการประมวลผลโปรแกรม LAMMPS ร่วมกับไฟล์ตั้งค่าที่เราได้ทำการตั้งเอาไว้ดังรูปที่ 26 ว่าจะประมวลผล Solvated 5-mer peptide เป็นจำนวนการเขย่า 100,000 ครั้ง โดยคำสั่งที่ใช้ประมวลผลแสดงดังภาพที่ 27 ซึ่งเป็นคำสั่งที่ทำให้โปรแกรมประมวลผลแบบเบื้องหลัง ไม่แสดงผลลัพธ์ออกมาทางหน้าจอแสดงผล

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

```
# Solvated 5-mer peptide
units           real
atom_style      full
pair_style      lj/charmm/coul/long 8.0 10.0 10.0
bond_style      harmonic
angle_style     charmm
dihedral_style  charmm
improper_style  harmonic
kspace_style    pppm 0.0001
read_data       data.peptide
neighbor        2.0 bin
neigh_modify    delay 5
timestep2.0
thermo_style    multi
thermo          50
fix             1 all nvt temp 275.0 275.0 100.0 tchain 1
fix             2 all shake 0.0001 10 100 b 4 6 8 10 12 14 18 a 31
group           peptide type <= 12
#dump           1 peptide atom 10 dump.peptide
#dump           1 peptide image 25 image.*.jpg type type &
#               axes yes 0.8 0.02 view 60 -30 bond atom 0.5
#dump_modify    1 pad 3
#compute        bnd all property/local btype batom1 batom2
#dump           2 peptide local 300 dump.bond index c_bnd[1] c_bnd[2] c_bnd[3]
run             100000
```

ภาพที่ 26 แสดงถึงไฟล์ตั้งค่าของ Solvated 5-mer peptide

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

```
veerapongs-MacBook-Pro:peptide veerapong$ nohup ./lmp_serial < in.peptide &
[1] 4858
veerapongs-MacBook-Pro:peptide veerapong$ appending output to nohup.out
```

ภาพที่ 27 แสดงถึงคำสั่งที่สั่งให้โปรแกรม LAMMPS ทำงานแบบลำดับ

หลังจากที่ได้สั่งทำประมวลผลโปรแกรมเป็นที่เรียบร้อยแล้วให้ลองทำการตรวจสอบดูก่อนว่าโปรแกรมทำงานอยู่หรือไม่โดยตรวจสอบจาก Logging ของโปรแกรมดังแสดงดังภาพที่ 28 ซึ่งถ้าหากมีการเปลี่ยนแปลงของข้อมูลใน Logging แสดงว่าโปรแกรมทำงานอยู่ และหลังจากสิ้นสุดการทำงานของโปรแกรม Logging ของ LAMMPS จะเป็นดังภาพที่ 29

```
---------------- Step      400 ----- CPU =      6.6319 (sec) ----------------
TotEng   =     -5242.7116 KinEng   =      1098.4817 Temp     =       273.0436
PotEng   =     -6341.1932 E_bond   =        16.3397 E_angle  =        37.0700
E_dihed  =        16.2130 E_impro  =         2.9105 E_vdwl   =       698.2117
E_coul   =     26791.5873 E_long   =    -33903.5253 Press    =      -276.4768
---------------- Step      450 ----- CPU =      7.4783 (sec) ----------------
TotEng   =     -5289.3114 KinEng   =      1107.8454 Temp     =       275.3711
PotEng   =     -6397.1568 E_bond   =        22.0946 E_angle  =        31.6180
E_dihed  =        20.7305 E_impro  =         1.9198 E_vdwl   =       690.5844
E_coul   =     26744.2913 E_long   =    -33908.3954 Press    =     -1552.8455
SHAKE stats (type/ave/delta) on step 500
  4 1.111 2.69807e-07
  6 0.997 2.69538e-07
  8 1.08 2.5317e-07
  10 1.111 3.17036e-07
  12 1.08 1.38902e-07
  14 0.96 0
  18 0.957201 3.83275e-06
  31 104.52 0.000412004
---------------- Step      500 ----- CPU =      8.3308 (sec) ----------------
TotEng   =     -5320.0981 KinEng   =      1090.9940 Temp     =       271.1824
PotEng   =     -6411.0920 E_bond   =        13.1791 E_angle  =        33.2942
E_dihed  =        17.9521 E_impro  =         1.2339 E_vdwl   =       667.9731
E_coul   =     26766.8138 E_long   =    -33911.5381 Press    =     -1584.9569
---------------- Step      550 ----- CPU =      9.1991 (sec) ----------------
TotEng   =     -5302.0011 KinEng   =      1110.6866 Temp     =       276.0773
PotEng   =     -6412.6878 E_bond   =        19.9748 E_angle  =        30.0551
E_dihed  =        18.7455 E_impro  =         1.4394 E_vdwl   =       654.7827
E_coul   =     26771.1212 E_long   =    -33908.8065 Press    =     -1877.8923
SHAKE stats (type/ave/delta) on step 600
  4 1.111 2.92417e-06
  6 0.997 8.88639e-08
```

ภาพที่ 28 แสดงถึง Logging ของโปรแกรม LAMMPS ขณะที่ทำงาน

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

```
---------------- Step   100000 ----- CPU =   1984.6090 (sec) ----------------
TotEng   =     -5270.6700 KinEng   =      1110.1940 Temp     =       275.9549
PotEng   =     -6380.8640 E_bond   =        20.7139 E_angle  =        33.1665
E_dihed  =        19.5743 E_impro  =         1.2858 E_vdwl   =       687.2401
E_coul   =     26764.3882 E_long   =    -33907.2329 Press    =      -717.4548
Loop time of 1984.61 on 1 procs for 100000 steps with 2004 atoms

Pair  time (%) = 1533.36 (77.2626)
Bond  time (%) = 4.43877 (0.22366)
Kspce time (%) = 178.286 (8.98345)
Neigh time (%) = 236.342 (11.9087)
Comm  time (%) = 9.39642 (0.473465)
Outpt time (%) = 0.0700452 (0.00352942)
Other time (%) = 22.7161 (1.14461)

FFT time (% of Kspce) = 52.3184 (29.3451)
FFT Gflps 3d (1d only) = 1.04816 1.50173

Nlocal:    2004 ave 2004 max 2004 min
Histogram: 1 0 0 0 0 0 0 0 0 0
Nghost:    11276 ave 11276 max 11276 min
Histogram: 1 0 0 0 0 0 0 0 0 0
Neighs:    707575 ave 707575 max 707575 min
Histogram: 1 0 0 0 0 0 0 0 0 0

Total # of neighbors = 707575
Ave neighs/atom = 353.081
Ave special neighs/atom = 2.34032
Neighbor list builds = 8440
Dangerous builds = 0
```

ภาพที่ 29 แสดงถึง Logging ของโปรแกรม LAMMPS หลังจากสิ้นสุดการทำงาน

ทำการทดลองวัดผลแต่ละวิธีของแต่ละแพลตฟอร์ม โดยทำการวัดผลจำนวน 3 ครั้งด้วยกันเพื่อหาค่าเฉลี่ย ข้อมูลในตารางที่ 3 ไปจนถึงตารางที่ 8 แสดงถึงข้อมูลของแต่ละแพลตฟอร์มโดยมีข้อมูลแบ่งเป็น 3 ชุดด้วยกันคือ Unoptimized, Floating-point operation และ Best

ตารางที่ 3 แสดงถึงข้อมูลของ Rock Cluster ที่มีการทำงานแบบขนานโดยใช้ MPI จำนวน 16 โหนด

Rock Cluser – MPI			
ลำดับที่	Unoptimized (sec)	Floating-point reordering (sec)	Best (sec)
1	791.99	787.59	738.82
2	792.8	792.19	748.54
3	796.02	781.62	741.36
Average	793.6	787.13	742.91

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ตารางที่ 4 แสดงถึงข้อมูลของ Rock Cluster ที่มีการทำงานแบบลำดับ

Rock Cluser – Serial			
Technique	Unoptimized (sec)	Floating-point reordering (sec)	Best (sec)
1	3912.01	3817.36	2824.92
2	3901.71	3825.06	2815.65
3	3897.32	3840.18	2832.53
Average	3903.68	3827.53	2824.37

ตารางที่ 5 แสดงถึงข้อมูลของ Mac OSX ที่มีการทำงานแบบขนานโดยใช้ MPI จำนวน 4 โหนด

Mac OSX – MPI			
Technique	Unoptimized	Floating-point reordering	Best
1	937.23	929.94	774.78
2	946.82	917.41	773.73
3	1113.85	914.39	780.14
Average	3903.68	920.58	776.22

ตารางที่ 6 แสดงถึงข้อมูลของ Mac OSX ที่มีการทำงานแบบลำดับ

Mac OSX – Serial			
Technique	Unoptimized	Floating-point reordering	Best
1	1984.61	1856.89	1456.32
2	2043.8	1858.97	1455.54
3	1893.42	1792.92	1455.96
Average	1973.94	1836.26	1455.94

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ตารางที่ 7 แสดงถึงข้อมูลของ Ubuntu ที่มีการทำงานแบบขนานโดยใช้ MPI จำนวน 4 โหนด

Ubuntu – MPI			
Technique	Unoptimized	Floating-point reordering	Best
1	964.64	928.2	813.64
2	937.3	922.58	831.41
3	1037.4	923.76	801.4
Average	979.78	924.85	815.48

ตารางที่ 8 แสดงถึงข้อมูลของ Ubuntu ที่มีการทำงานแบบลำดับ

Ubuntu – Serial			
Technique	Unoptimized	Floating-point reordering	Best
1	2015.43	1772.66	1545.86
2	2012.91	1759.55	1555.93
3	1969.33	1828.49	1539.67
Average	1999.22	1786.9	1547.15

หลังจากที่ทำการวัดผลเพื่อหาค่าเฉลี่ยของแต่ ละวิธีและแต่ละแพลตฟอร์มมาได้แล้ว เราจะทำการนำข้อมูลทั้งห มดมาวิเคราะห์ เพื่อดูว่าผลลัพธ์ เป็นอย่างไรบ้าง ซึ่งในกรณีการวิจัยนี้เรา คาดหวังว่าโปรแกรมจะทำงานเร็วขึ้นอย่างน่าพอใจ ข้อมูลในตารางที่ 9 แสดงถึงผลลัพธ์ของค่าเฉลี่ยรวมทุกๆวิธีของแต่ละแพลตฟอร์ม

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ตารางที่ 9 แสดงถึงเวลาในการประมวลผลแบบเฉลี่ยของทุกแพลตฟอร์ม

Criteria	**Rocks Cluster (sec)**		**Mac OSX (sec)**		**Ubuntu (sec)**	
	MPI	**Serial**	**MPI**	**Serial**	**MPI**	**Serial**
Unoptimized	793.60	3903.68	999.30	1973.94	979.78	1999.22
Floating-point reordering	787.13	3807.53	920.58	1836.26	924.85	1786.90
Best	742.91	2824.37	776.22	1455.94	815.48	1547.15

หลังจากที่ได้ค่าเฉลี่ยมาแล้วเราจะนำมาวิเคาะห์ถึงปริมาณการเพิ่มขึ้นของสมรร ถนะของโปรแกรม ซึ่งข้อมูลในตารางที่ 10 ได้แสดงให้เห็นถึงปริมาณการเพิ่มขึ้นของสมรรถนะเป็นเปอร์เซ็นต์ ซึ่งทุกๆแพลตฟอร์มก็มีสมรรถนะที่ดีขึ้นแตกต่างกันไป แต่ก็ถือว่าดีขึ้นทุกๆแพลตฟอร์ม

ตารางที่ 10 แสดงถึงเปอร์เซ็นต์ที่ดีขึ้นสำหรับการประมวลผลในแต่ละแพลตฟอร์มเมื่อเทียบกับก่อนการเพิ่มสมรรถนะ

Platform	Rocks Cluster		Mac OSX		Ubuntu	
Parallel/Serial	MPI	Serial	MPI	Serial	MPI	Serial
Floating-point reordering	0.82%	2.46%	7.88%	6.97%	5.61%	10.62%
Best	6.39%	27.65%	22.32%	26.24%	16.77%	22.61%

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ตารางที่ 11 แสดงถึง Line of Code ของโปรแกรมก่อนถูกแก้ไข

Original	
File	**Line of Code**
math_const.h	32
neight_half_bin.cpp	487
neighbor.h	392
pair_ij_charmm_coul_long.cpp	1014
pppm.h	3014

ตารางที่ 12 แสดงถึง Line of Code ของโปรแกรมที่ถูกแก้ไขไปเพื่อให้ได้ Best performance

Best performance	
File	**Line of Code**
math_const.h	1
neight_half_bin.cpp	20
neighbor.h	8
pair_ij_charmm_coul_long.cpp	67
pppm.h	38

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ตารางที่ 13 แสดงถึง Line of Code ของโปรแกรมที่ถูกแก้ไขไปด้วยวิธี Floating-point reordering

Floating-point reordering	
File	**Line of Code**
math_const.h	1
pppm.h	38

ตารางที่ 14 แสดงถึง Line of Code ของโปรแกรมที่ถูกแก้ไขไปแต่ละวิธีในรูปแบบเปอร์เซ็นต์

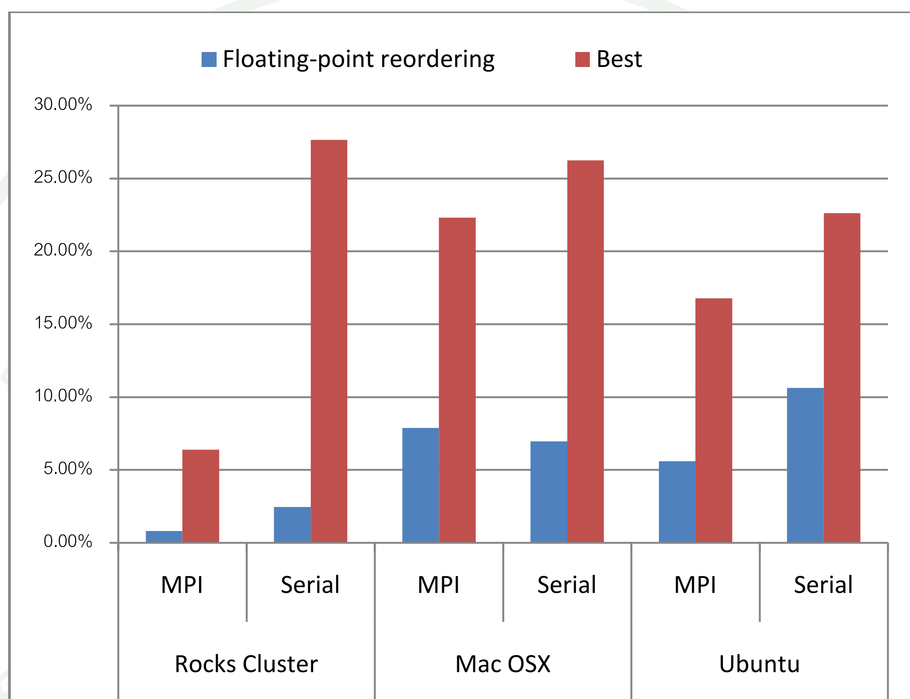
Optimization method	**Line of Code (%)**
Best	2.69
Floating-point reordering	1.38

จากข้อมูลในตารางที่ 11-14 แสดงให้เห็นว่าการทำการเพิ่มสมรรถนะให้กับ LAMMPS ในครั้งนี้ได้ทำการแก้ไขโปรแกรมไปน้อยมากเพียงไม่กี่เปอร์เซ็นต์เท่านั้น

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

2. ผลการทดลอง

ผลการทดลองแสดงให้เห็นว่าการทดลองนี้สามารถเพิ่มสมรรถนะได้สูงสุดถึง 27.65 เปอร์เซ็นบน Rock Cluster ที่ประมวลผลแบบขนาน และน้อยที่สุดคือ 0.82 เปอร์เซ็นบน Rock Cluster ที่ประมวลผลแบบลำดับ ส่วนผลการทดลองบนแพลตฟอร์มอื่นๆนั้นแสดงดังรูปที่ 30



ภาพที่ 30 แผนภาพแสดงการเปรียบเทียบผลลัพธ์ของแพลตฟอร์มแบบต่างๆ

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

วิจารณ์

1. เทียบผลการทดลองกับงานวิจัยก่อนหน้าที่ใช้วิธีการเดียวกัน

หากเปรียบเทียบผลการทดลองที่ได้กับงานวิจัยของ McKenna ที่ถือว่ามีความใกล้เคียงกันนั้น จะเห็นได้ว่างานวิจัยของ McKenna สามารถเพิ่มสมรรถนะได้สูงสุดประมาณ 2 เปอร์เซ็นต์เมื่อเทียบกับของเดิม แต่งานชิ้นนี้มีสมรรถนะที่เพิ่มมากขึ้นสูงสุดถึง 28 เปอร์เซ็นต์ ซึ่งดีกว่ าของ McKenna เป็นอย่างมาก ทำให้เห็นว่ายังคงมีพื้นที่ให้ทำการเพิ่มสมรรถนะได้อีก รวมไปถึงข้อดีของการใช้เทคนิคอย่างง่าย แบบทั่วๆ ไป ที่โปรแกรมยังคงสามารถทำงานบนแพลตฟอร์มชนิดต่างๆ ได้เหมือนเดิม

2. เทียบผลการทดลองกับงานวิจัยก่อนหน้าที่ใช้วิธีการต่างกัน

งานก่อนหน้าที่ใช้วิธีการที่แตกต่างกันคืองานของ Fisher ซึ่งงานของ Fisher นั้นจะใช้ชุดคำสั่งพิเศษ ที่จะมีเฉพาะเครื่องๆนั้นเพียงเท่านั้นเข้ามาช่วยในการเพิ่มสมรรถนะ ซึ่งจะส่งผลให้ผลการทดลองสำหรับงานของ Fisher เพิ่มสมรรถนะได้เป็นอย่างมากซึ่งมากสุดถึง 3.5 เท่า แต่ว่าผลเสียที่ตามมาคือโปรแกรมจะสามารถทำงานได้บนเครื่องที่การปรับปรุงได้เพียงเครื่องเดียวเท่านั้น ไม่สามารถนำไปทำงานยังเครื่องอื่นๆได้

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

สรุปและข้อเสนอแนะ

สรุป

ในวิทยานิพนธ์เล่มนี้ ผู้วิจัยได้นำเสนอแนวทางการปรับปรุง สมรรถนะของโปรแกรม LAMMPS โดยใช้วิธีอย่างง่ายที่มีใช้งานทั่วไปซึ่งสามารถแบ่งสรุปประเด็นต่างๆได้ดังนี้

การปรับปรุงสมรรถนะโดยวิธีอย่างง่ายจะทำให้โปรแกรมสามารถทำงานบนเครื่องอื่นๆได้เหมือนเดิมไม่เฉพาะเครื่องใดเครื่องหนึ่ง

สามารถอธิบายได้ว่าเหตุใดการปรับปรุงสมรรถนะดังที่กล่าวมานั้น จึงได้ผลดี และเหตุใดตัวแปลภาษาและสถาปัตยกรรมแบบซุปเปอร์สเกลาร์จึงไม่สามารถทำแบบนี้ได้

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ข้อเสนอแนะ

เราสามารถเพิ่มสมรรถนะให้กับโปรแกรมได้โดยใช้วิธีอย่างง่ายตามที่ได้กล่าวมาด้วยวิธีการต่างๆดังต่อไปนี้

1. ศึกษาพฤติกรรมของโปรแกรมว่าส่วนไหนที่ทำงานหนักและเป็นจุดด้อยของโปรแกรม

2. ทำการเลือกวิธีที่เหมาะสมสำหรับส่วนของโปรแกรมที่ต้องการจะปรับปรุงสมรรถนะ

3. ทำการวัดผลกับหลายๆแพลตฟอร์ม เพื่อให้แน่ใจว่าโปรแกรมมีสมรรถนะที่เพิ่มขึ้นกับทุกๆแพลตฟอร์ม ทำให้แน่ใจว่าสมรรถนะของ โปรแกรมไม่ได้เพิ่มขึ้นบนแพลตฟอร์มใดแพลตฟอร์มหนึ่งเท่านั้น

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

เอกสารและสิ่งอ้างอิง

S. Plimpton, A. Thompson, P. Crozier and A. Kohlmeyer. 1990. **LAMMPS Molecular Dynamics Simulator**. Available Source: http://lammps.sandia.gov, November 20, 2013.

J. E. Smith and G. S. Sohi. 1995. The Microarchitecture of Superscalar Processors. **Proceedings of the IEEE, December 1995**.

R.B. Jones and V.H. Allan. 1990. Software pipelining: a comparison and improvement. **Microprogramming and Microarchitecture. Micro 23. Proceedings of the 23rd Annual Workshop and Symposium:** 46-56.

M.G. Stoodley and C.G. Lee. 1996. Software pipelining loops with conditional branches. **Microarchitecture, 1996. MICRO-29.Proceedings of the 29th Annual IEEE/ACM International Symposium:** 262-273.

H. Wei, J. Yu and H. Yu. 2012. **Software Pipelining for Stream Programs on Resource Constrained Multicore Architectures, Parallel and Distributed Systems, IEEE Transactions on vol. 23, no. 12:** 2338-2350

J.P. Grossman. 2000. Cheap out-of-order execution using delayed issue. **Proceedings. 2000 International Conference on Sep 2000:** 549-551.

J.D. Collins, P.H. Wang and H. Wang. 2002. Memory latency-tolerance approaches for Itanium processors: out-of- order execution vs speculative precomputation. **High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium:** 187-196.

O. Mutlu, H. Kim, D.N. Armstrong and Y.N. Patt. 2005. An analysis of the performance impact of wrong-path memory references on out-of-order and runahead execution processors. **Computers, IEEE Transactions on vol. 54, no. 12, Dec 2005:** 1556-1571.

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

S.K. Srinivasan. 2010. Automatic Refinement Checking of Pipelines with Out-of-Order Execution. **Computers, IEEE Transactions on vol. 59, no. 8, Aug 2010**: 1138-1144.

R. Silvera, J. Wang, G.R. Gao and R. Govindarajan. 1997. A register pressure sensitive instruction scheduler for dynamic issue processors. **Parallel Architectures and Compilation Techniques., 1997. Proceedings., 1997 International Conference on Nov 1997:** 78-89.

P. Ratanaworabhan and M. Burtscher. 2006. Load Instruction Characterization and Acceleration of the BioPerf Programs. **Workload Characterization, 2006 IEEE International Symposium on Oct 2006:** 71-79.

M. Booshehri, A. Malekpour and P. Luksch. 2013. An Improving Method for Loop Unrolling. **(IJCSIS) International Journal of Computer Science and Information Security, Vol. 11, No. 5, May 2013.**

H. S. Warren. 2012. **Hacker's Delight, Second Edition.** Addison Wesley, USA.

M.B. Kraeling. 1996. Optimization of C code in a real-time environment. **WESCON/96, Oct 1996:** 574-580.

J. Fisher, V. Natoli and D. Richie. 2006. Optimization of LAMMPS. **HPCMP Users Group Conference, June 2006:** 374-377.

G. McKenna. 2007. Performance analysis and optimisation of LAMMPS on XCmaster. **HPCx and Blue Gene. A dissertations of University of Edinburgh, 2007.**

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์

ประวัติการศึกษาและการทำงาน

ชื่อ-นามสกุล	นายวีระพง แก้วเทศ
เกิดวันที่	11 พฤษภาคม 2529
สถานที่เกิด	กรุงเทพมหานคร
ประวัติการศึกษา	วศ.บ. (วิศวกรรมคอมพิวเตอร์) คณะวิศวกรรมศาสตร์มหาวิทยาลัยมหิดล (2552)
ตำแหน่งปัจจุบัน	Specialist Software Engineer
สถานที่ทำงานปัจจุบัน	บริษัทฟรีวิวโซลูชั่น ชั้น 29 อาคารลุมพินีทาวเวอร์ แขวงทุ่งมหาเมฆ เขตสาธร กรุงเทพมหานคร 10200
ทุนการศึกษาที่ได้รับ	-

ลิขสิทธิ์ มหาวิทยาลัยเกษตรศาสตร์