การพัฒนาโปรแกรมแบบขนานพร้อมแบบจำลองด้านเวลาบนหน่วยประมวลผลกราฟิก กรณีศึกษาการจำลองสึนามิ

นายพงษ์พัฒน์ เป้าเพ็ชร์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2555

DEVELOPING PARALLEL PROGRAMS WITH TIMING MODEL ON A GRAPHICS PROCESSING UNIT: A CASE STUDY OF TSUNAMI SIMULATION

Mr. Pongpat Poapetch

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Engineering Program in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2012

หัวข้อวิทยานิพนธ์	การพัฒนาโปรแกรมแบบขนานพร้อมแบบจำลองด้านเวลาบนหน่วยประมวลผลกราฟิก กรณีศึกษาการจำลองสึนามิ
โดย	นายพงษ์พัฒน์ เป้าเพ็ชร์
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร.วีระ เหมืองสิน

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(รองศาสตราจารย์ ดร.บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(ผู้ช่วยศาสตราจารย์ ดร.วีระ เหมืองสิน)

.....กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.เกริก ภิรมย์โสภา)

.....กรรมการภายนอกมหาวิทยาลัย

(ผู้ช่วยศาสตราจารย์ ดร.ภุชงค์ อุทโยภาศ)

พงษ์พัฒน์ เป้าเพ็ชร์ : การพัฒนาโปรแกรมแบบขนานพร้อมแบบจำลองด้านเวลาบนหน่วยประมวลผลกราฟิก กรณีศึกษาการจำลองสึนามิ. (DEVELOPING PARALLEL PROGRAMS WITH TIMING MODEL ON A GRAPHICS PROCESSING UNIT: A CASE STUDY OF TSUNAMI SIMULATION) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ. ดร.วีระ เหมืองสิน, 11 หน้า.

ในปัจจุบันการคำนวณแบบขนานมีความแพร่หลายมากขึ้น คอมพิวเตอร์รุ่นใหม่ๆแทบทุกเครื่องต่างก็เป็นคอมพิวเตอร์แบบขนาน นอกจากนี้ยังมีการนำเอาหน่วยประมวลผลกราฟิกที่มีอยู่บนการ์ดควบคุมจอภาพมาใช้ในการคำนวณอย่างแพร่หลาย

งานวิจัยนี้มุ่งเน้นไปที่ปัญหาการจำลองสึนามิ ซึ่งเป็นปัญหาที่มีเวลาในการคำนวณที่จำกัดขึ้นอยู่กับเวลาที่คลื่นใช้ในการเดินทางจากแหล่งกำเนิดคลื่นมาจนถึงชายฝั่ง ลบด้วยเวลาที่ต้องใช้ในการแจ้งเตือนภัยและอพยพ ตัวอย่างเช่น สมมติว่ารอยเลื่อนมีพลังที่อยู่ใกล้ประเทศไทยและมีโอกาสทำให้เกิดคลื่นสึนามิ มีระยะทางที่คลื่นสึนามิจะต้องใช้เวลาอย่างน้อย 90 นาทีจึงจะมาถึงชายฝั่งประเทศไทย หากกำหนดให้เวลาที่ใช้ในการแจ้งเตือนภัยและอพยพ 30 นาที โปรแกรมจำลองสึนามิก็ต้องทำงานให้เสร็จภายใน 60 นาที โดยจะทำการพัฒนาโปรแกรมจำลองสึนามิ จากเดิมที่มีการทำงานแบบลำดับ ให้สามารถทำงานแบบขนานบนหน่วยประมวลกราฟฟิกได้ และทำการปรับปรุงโปรแกรมให้ทำงานได้อย่างมีประสิทธิภาพบนเครื่องที่แตกต่างกัน ด้วยเทคนิคและวิธีปรับปรุงโปรแกรมบนหน่วยประมวลผลกราฟิก และหาค่าปัจจัยที่ส่งผลต่อเวลาในการทำงานของโปรแกรม เพื่อหาวิธีลดเวลาในการทำงานโดยไม่ส่งผลกระทบต่อผลลัพธ์ที่จะนำไปใช้ในการการประเมินสถาณการณ์

ภาควิชา	วิศวกรรมคอมพิวเตอร์	ลายมือชื่อนิสิต	.....
สาขาวิชา	วิศวกรรมคอมพิวเตอร์	ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก	.....
ปีการศึกษา	2555		

5470284121 : MAJOR COMPUTER ENGINEERING

KEYWORDS: CUDA / CPU / TIME CONSTRAINT / TSUMANI SIMULATION / TUNAMI

PONGPAT POAPETCH: DEVELOPING PARALLEL PROGRAMS WITH TIMING MODEL ON A GRAPHICS PROCESSING UNIT: A CASE STUDY OF TSUNAMI SIMULATION. ADVISOR : ASST. PROF.VEERA MUANGSIN, Ph.D., 11 pp.

Nowadays, high-performance personal computers can be constructed with a wide range of multicore CPUs and GPGPUs (general-purpose graphics processing units) that vary in the number and performance of processor cores. The variation makes it even more difficult to develop a parallel program that performs well on different computers and to estimate its performance on a particular computer. Also, given performance requirements for a parallel program, it is hard to determine whether a specific computer will be able to satisfy the requirements. The latter problem is the case for applications with real time constraints that guarantee to produce results within specified time. Some of these applications solve optimization algorithms that give improved results over time. Therefore, within fixed time, the speed of computation of such an application on a particular machine also determines the quality of the results.

Our research focuses how to develop parallel programs on GPGPUs that solve optimization problems with real time constraints and give the best possible results. In particular, we are interested in real time tsunami simulation. The time constraints of this problem are determined by the travelling time of the first wave from the origin to the shore and the time allowed for warning and evacuation. Our parallel tsunami simulation program is based on TUNAMI program. The algorithm uses finite difference methods with nested multi-resolution regions. The original Fortran code is rewritten in C language with CUDA API. The results show that the C+CUDA version achieves good speedup and can satisfy the time constraints for real time tsunami warning for Thailand.

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

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

Academic Year: 2012 .....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุลวงด้วยดีด้วยความช่วยเหลือของบุคคลหลายท่านโดยบุคคลท่านแรกที่ขอขอบพระคุณเป็นอย่างยิ่งคือ ผู้ช่วยศาสตราจารย์ ดร.วีระ เหมืองสิน อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่คอยให้คำแนะนำต่างๆตลอดทุกขั้นตอนของการทำวิทยานิพนธ์ ให้ความช่วยเหลือในการแก้ปัญหาระหว่างการทำวิทยานิพนธ์ รวมทั้งชี้แนะแนวทางและทัศนคติที่เป็นประโยชน์ในการทำวิทยานิพนธ์

ลำดับต่อมาขอขอบพระคุณผู้ช่วยศาสตราจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์ ประธานกรรมการสอบวิทยานิพนธ์ อาจารย์ ผู้ช่วยศาสตราจารย์ ดร.เกริก ภิรมย์โสภา กรรมการสอบวิทยานิพนธ์ และผู้ช่วยศาสตราจารย์ ดร.ภชุงค์ อุทโยภาศ กรรมการภายนอกมหาวิทยาลัย ที่ได้กรุณาให้คำแนะนำและชี้แนะแนวทางที่เป็นประโยชน์ต่อการทำวิทยานิพนธ์ในครั้งนี้ อีกทั้งขอขอบพระคุณผู้ดูแลระบบคอมพิวเตอร์ของคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ที่ให้คำปรึกษาและอำนวยความสะดวกในการใช้งานเครื่องระบบคลัสเตอร์เพกาซัส

นอกจากนี้ขอขอบพระคุณบุคคลดังต่อไปนี้ นายอลงกต บุรุษอาชาไนย ภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย และ นายธีรยุทธ โกสินทร์ ที่คอยให้คำปรึกษาเกี่ยวกับการทำวิทยานิพนธ์นี้เป็นอย่างมาก ขอขอบคุณนายธารนิธิ เศรษฐพานิชผล และเพื่อนๆพี่ๆร่วมห้องปฏิบัติการทุกคนที่คอยรับฟัง และให้คำปรึกษาทั้งเรื่องวิทยานิพนธ์และการดำเนินชีวิตในมหาวิทยาลัยทำให้มีความสุขและสนุกสนานกับการศึกษาในภาควิชาตลอด 2 ปีที่ผ่านมา

สุดท้ายนี้ขอขอบพระคุณ บิดา มารดา และสมาชิกในครอบครัว ที่ได้ให้กำลังใจและให้การอุปการะในทุกด้านอยู่เสมอมา รวมทั้งขอขอบพระคุณคณาจารย์ทุกท่านที่ได้ประสิทธิ์ประสาทวิชาตลอดช่วงชีวิตที่ผ่านมาทำให้มีความรู้ติดตัวมาจนถึงทุกวันนี้

สารบัญ

หน้า

หน้า

หน้า

หน้า

สารบัญตาราง

หน้า

สารบัญรูป

หน้า

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันการคำนวณแบบขนาน (parallel computing) มีความแพร่หลายมากขึ้น คอมพิวเตอร์รุ่นใหม่ๆแทบทุกเครื่อง แม้แต่สมาร์ทโฟนต่างก็เป็นคอมพิวเตอร์แบบขนาน เนื่องจากหน่วยประมวลผลที่ผลิตออกมาใหม่ส่วนใหญ่เป็นหน่วยประมวลผลแบบหลายคอร์ (Multicore processor) คือในหน่วยประมวลผลหนึ่งตัวมีหน่วยประมวลผลย่อยที่เรียกว่าคอร์ (core) อยู่หลายตัว นอกจากนี้ยังมีการนำเอาหน่วยประมวลผลกราฟิก (Graphics Processing Unit) ที่มีอยู่บนการ์ดควบคุมจอภาพ (Video card) มาใช้ในการคำนวณทั่วๆ ไป เรียกว่า General Purpose Graphics Processing Unit (GPGPU หรือ GPU) ซึ่งก็เป็นหน่วยประมวลผลแบบขนานชนิดหนึ่งเช่นกันแต่หน่วยประมวลผลกราฟิกจะมีจำนวนคอร์มากกว่าหน่วยประมวลผลมัลติคอร์อยู่หลายเท่าและยังมีความสามารถในการประมวลผลสูงกว่าหน่วยประมวลผลทั่วไปสำหรับการคำนวณบางประเภท เช่น การประมวลผลข้อมูลแบบอาร์เรย์ด้วย เหตุนี้การสร้างคอมพิวเตอร์สมรรถนะสูงที่มีขนาดเล็กในรูปแบบของคอมพิวเตอร์ส่วนบุคคล ที่มีทั้งหน่วยประมวลผลมัลติคอร์และหน่วยประมวลผลกราฟิกจึงกลายเป็นที่นิยมมากขึ้นเรื่อยๆ

อย่างไรก็ตาม คอมพิวเตอร์สมรรถนะสูงมีความต้องการหรือปัญหาหลักอยู่สองประการ คือ การเขียนโปรแกรมและประสิทธิภาพการเขียนโปรแกรมสำหรับหน่วยประมวลผลกราฟิก จะต้องใช้ภาษาหรือไลบรารีเพื่อควบคุมการคำนวณบนหน่วยประมวลผลกราฟิกตัวอย่างเช่น CUDA [1] และ OpenCL [2] เป็นต้นซึ่งมีรูปแบบการโปรแกรมที่ให้การประมวลผลหลักทำงานโดยหน่วยประมวลผลหลัก และแบ่งการคำนวณบางส่วนไปให้หน่วยประมวลผลกราฟิก โดยการส่งส่วนของโปรแกรมซึ่งประกอบด้วยคำสั่ง (Instruction code) และข้อมูลที่เกี่ยวข้องไปที่หน่วยความจำของหน่วยประมวลผลกราฟิกเมื่อหน่วยประมวลผลกราฟิกคำนวณเสร็จก็จะทำการคัดลอกผลลัพธ์มาจากหน่วยความจำของหน่วยประมวลผลกราฟิกกลับไปที่หน่วยประมวลผลหลักต่อไปส่วนของโปรแกรมที่ทำงานบนหน่วยประมวลผลกราฟิกจะมีลักษณะเป็น multithread โดยที่แต่ละคอร์ทำงานทีละหนึ่งเธรดและแต่ละเธรดทำงานตามคำสั่งชุดเดียวกัน

บนข้อมูลที่ไม่เหมือนกัน (Single instruction, multiple data: SIMD) [3] แต่อาจมีการใช้ข้อมูลร่วมกันผ่านหน่วยความจำร่วม (Shared memory) การโปรแกรมจึงเน้นไปที่การแบ่งงาน, การแบ่งข้อมูลให้แก่เธรด และการจัดการเรื่องการใช้ข้อมูลร่วมกันระหว่างเธรด เพื่อให้ทำงานได้ถูกต้องตามอัลกอริทึม

สำหรับเรื่องของสมรรถนะ (Performance) หรือประสิทธิภาพ (Efficiency) นั้น โปรแกรมคำนวณแบบขนานมีความซับซ้อนกว่าโปรแกรมแบบคำนวณลำดับ (Sequential program) โปรแกรมแต่ละโปรแกรมมีสมรรถนะหรือประสิทธิภาพในการทำงานบนคอมพิวเตอร์แต่ละเครื่องไม่เหมือนกัน ขึ้นอยู่กับปัจจัยหลายอย่าง เช่น

- ประเภท, จำนวนและสมรรถนะของหน่วยประมวลผลหรือคอร์ซึ่งมีความหลากหลายมาก โดยเฉพาะกรณีของจำนวนคอร์ของหน่วยประมวลผลกราฟิก
- ความสามารถในการคำนวณแบบขนาน (Parallelism) ซึ่งขึ้นอยู่กับความเป็นอิสระต่อกันของของข้อมูลที่ใช้ในแต่ละเธรด หากมีความขึ้นต่อกันของข้อมูล (data dependency) มาก ก็จะทำให้ความสามารถในการคำนวณแบบขนานลดลง
- การแบ่งงานเพื่อให้แต่ละหรือคอร์ได้รับงานในปริมาณงานที่ใกล้เคียงกัน (load balancing) และมีขนาดของชิ้นงานที่เหมาะสม ไม่ใหญ่ไม่เล็กเกินไป (granularity)
- หน่วยประมวลผลกราฟิกอาจมีคอร์จำนวนมากถึงหลายร้อยคอร์ ซึ่งสามารถใช้ข้อมูลร่วมกัน และอาจมีการเข้าถึงข้อมูลปริมาณมากพร้อม ๆ กันได้หน่วยประมวลผลกราฟิกจึงมักมีประเภทและโครงสร้างของหน่วยความจำที่ซับซ้อน และการเลือกใช้งานอย่างเหมาะสมมีผลต่อสมรรถนะและประสิทธิภาพของโปรแกรม
- หน่วยประมวลผลกราฟิกอยู่บนการ์ดควบคุมจอภาพซึ่งเป็นอุปกรณ์อินพุตเอาต์พุตประเภทหนึ่งจึงต้องมีงานส่วนเกิน (Overhead) ที่เกี่ยวกับการติดต่อระหว่างหน่วยประมวลผลหลักและ หน่วยประมวลผลกราฟิกโดยเฉพาะอย่างยิ่งการโอนย้ายข้อมูลระหว่างกัน (Transfer data)

เนื่องจากปัจจัยเหล่านี้ มีความแตกต่างกันมากใน GPU แต่ละรุ่นการพัฒนาโปรแกรมบน GPU ให้มีประสิทธิภาพสูงบน GPU ใด ๆจึงเป็นปัญหาที่น่าสนใจตัวอย่างเช่นเมื่อนำโปรแกรมเดียวกันไปรันบนคอมพิวเตอร์ที่มี GPU ที่มีจำนวนคอร์มากขึ้น ก็สามารถใช้คอร์ทั้งหมดที่มีอย่างเต็มที่ และได้ความเร็วเพิ่มขึ้นอย่างสูงสุดเท่าที่เป็นไปได้ เรียกว่ามีความสามารถในการขยายระบบ (scalability)

กรณีเฉพาะของปัญหาเกี่ยวกับประสิทธิภาพของโปรแกรมกรณีหนึ่งคือ การคำนวณที่มีเวลาจำกัด (Time constraints) โดยเวลาในที่นี้หมายถึงเวลาจริง (Real-time) ตัวอย่างของปัญหาประเภทนี้ ได้แก่ การพยากรณ์เหตุการณ์ใดๆ ก็ตามที่จะเกิดขึ้นจริงในอนาคต จึงต้องกำหนดเวลาที่ต้องการผลการพยากรณ์ไว้ก่อนที่เหตุการณ์จริงจะเกิดขึ้น หากได้ผลหลังจากนั้นจะไม่เกิดประโยชน์ เช่น การพยากรณ์อากาศ ภัยพิบัติ ตลาดหุ้น เป็นต้น

ตัวอย่างอีกประเภทคือ การทำงานโต้ตอบหรือสื่อสารกับมนุษย์ ซึ่งต้องการความเร็วในการโต้ตอบ แต่ก็มีความทนต่อความผิดพลาด (Error tolerance) เช่น ในเกมซึ่งเน้นที่การทำให้ผู้เล่นรู้สึกว่าวัตถุต่างๆ มีลักษณะและการเคลื่อนที่เป็นธรรมชาติตามกฎทางฟิสิกส์ แต่ที่จริงแล้วใช้วิธีคำนวณที่ได้ผลลัพธ์เร็วแต่อาจไม่ตรงกับในธรรมชาติแต่ผู้เล่นยังยอมรับได้ อีกตัวอย่างหนึ่งคือการคำนวณหาเส้นทางในขณะที่พาหนะเคลื่อนที่ไปเรื่อยๆ ผลลัพธ์ที่ได้อาจไม่ใช่เส้นทางที่ดีที่สุดจริงๆ ปัญหาเหล่านี้มักมีลักษณะที่ หากมีเวลามากขึ้น หรือมีเวลาเท่าเดิมแต่มีคอมพิวเตอร์ที่เร็วขึ้นก็น่าจะได้ผลลัพธ์ที่แม่นยำขึ้น ดังนั้น การพัฒนาโปรแกรมที่สามารถทำงานได้เสร็จทันเวลาที่กำหนดโดยได้ความแม่นยำสูงสุดตามเวลาที่มี จึงเป็นปัญหาที่ท้าทายปัญหาหนึ่งนั่นคือ หากนำโปรแกรมไปรันบนคอมพิวเตอร์ที่เร็วขึ้นก็สามารถใช้ประโยชน์จากเวลาและหน่วยประมวลผลที่มีให้มากที่สุดและได้ผลลัพธ์ที่แม่นยำที่สุด หากนำโปรแกรมไปรันบนคอมพิวเตอร์ที่ช้าลง ก็ยังสามารถทำงานให้เสร็จตามเวลาที่กำหนด โดยอาจต้องแลกมาด้วยความละเอียดหรือความแม่นยำที่ลดลง

สำหรับงานวิจัยนี้ มุ่งเน้นไปที่ปัญหาการจำลองสึนามิ ซึ่งหากทำงานได้เร็วพอ ก็จะสามารถใช้ในการแจ้งเตือนภัยแบบเรียลไทม์ได้ และเป็นปัญหาที่มีเวลาในการคำนวณที่จำกัดขึ้นอยู่กับเวลาที่คลื่นใช้ในการเดินทางจากแหล่งกำเนิดคลื่นมาจนถึงชายฝั่ง ลบด้วยเวลาที่ต้องใช้ในการแจ้งเตือนภัยและอพยพ ตัวอย่างเช่น สมมติว่ารอยเลื่อนมีพลังที่อยู่ใกล้ประเทศไทยและมี

โอกาสทำให้เกิดคลื่นสึนามิ มีระยะทางที่คลื่นสึนามิจะต้องใช้เวลาอย่างน้อย 90 นาทีจึงจะมาถึงชายฝั่งประเทศไทย หากกำหนดให้เวลาที่ใช้ในการแจ้งเตือนภัยและอพยพ 30 นาที โปรแกรมจำลองสึนามิก็ต้องทำงานให้เสร็จภายใน 60 นาทีอย่างไรก็ตาม ในขณะที่รอเป็นเวลา 60 นาทีให้ได้ผลลัพธ์จากคอมพิวเตอร์เครื่องหนึ่ง ก็ยังสามารถใช้คอมพิวเตอร์เครื่องอื่นโปรแกรมจำลองสึนามิเช่นเดียวกันแต่ต้องการผลลัพธ์ภายในเวลาที่สั้นกว่าเพื่อใช้ในการประเมินสถานการณ์ก่อน โดยที่ผลลัพธ์อาจมีความแม่นยำน้อยกว่า หรืออาจใช้คอมพิวเตอร์หลายเครื่อง ที่ถูกกำหนดให้คำนวณเสร็จในเวลาที่ต่างกัน เพื่อให้ได้ผลลัพธ์ออกมาเป็นระยะ ๆ โดยมีความแม่นยำเพิ่มขึ้นเรื่อยๆ

1.2 วัตถุประสงค์ของงานวิจัย

งานวิจัยนี้มีวัตถุประสงค์เพื่อนำเสนอระเบียบวิธี (Methodology) สำหรับการพัฒนาโปรแกรมคำนวณแบบขนานบนหน่วยประมวลผลกราฟิก สำหรับการคำนวณปัญหาที่มีเวลาจำกัด โดยใช้โปรแกรมจำลองสึนามิเป็นกรณีศึกษา เพื่อทำให้โปรแกรมคำนวณแบบขนานบนหน่วยประมวลผลกราฟิกสามารถทำงานได้อย่างมีประสิทธิภาพ ด้วยการปรับปรุงรูปแบบการเข้าถึงหน่วยความจำ (Memory Access Pattern) การจัดการเธรด (Thread Management) และปรับความละเอียดในการทำงานของโปรแกรม (Resolution) เพื่อเพิ่มประสิทธิภาพการคำนวณตามขีดความสามารถของหน่วยประมวลผลกราฟิกที่ใช้ ให้สามารถทำงานได้ตามข้อจำกัดด้านเวลาได้

1.3 ขอบเขตของงานวิจัย

1. ใช้โปรแกรม TUNAMI ที่มีการคำนวณแบบลำดับเป็นกรณีศึกษา
2. ใช้คอมไพล์เลอร์ CUDA compilation tools, release 4.1, V0.2.1221
3. กราฟิกการ์ดที่นำมาทดสอบมี 2 รุ่น คือ NVIDIA Quadro FX 3800 และ GeForce GT 430 โดยมีรายละเอียดดังนี้

NVIDIA Quadro FX 3800	
CUDA Capability version	1.3
Global memory	1 GB DDR3
CUDA Cores	192 Cores
GPU Clock Speed	1.20 GHz
Memory Clock rate	800 MHz
Memory Bus Width	256-bit

GeForce GT 430	
CUDA Capability version	2.1
Global memory	1 GB DDR3
CUDA Cores	96 Cores
GPU Clock Speed	700 MHz
Memory Clock rate	800 MHz
Memory Bus Width	128-bit

4. ข้อมูลที่ใช้ในการจำลองสึนามิ จะใช้ข้อมูลที่สำรวจไว้ โดย ผศ.ดร.อาณัติ เรืองรัศมี และนิสิตภาควิชาวิศวกรรมโยธา จุฬาลงกรณ์มหาวิทยาลัย เป็นผู้จัดทำข้อมูลส่วนนี้เป็นแฟ้มข้อมูล ซึ่งประกอบด้วยแฟ้มข้อมูลของระดับความสูงคลื่นผิวน้ำและแฟ้มข้อมูลของระดับความลึกของน้ำทะเลรวมไปถึงพิกัดและขนาดของพื้นที่ในแต่ละบริเวณ ความละเอียดที่ใช้ในการคำนวณในแต่ละบริเวณ ตำแหน่งของพื้นที่ภายในของแต่ละบริเวณ
5. ข้อมูลการเกิดสึนามิที่จะนำมาเปรียบเทียบ จะใช้ผลลัพธ์ที่ได้จากการคำนวณด้วยโปรแกรม TUNAMI แบบลำดับ ด้วยแฟ้มข้อมูลที่ได้จากภาควิชาวิศวกรรมโยธา ในการเปรียบเทียบความถูกต้องของผลลัพธ์

1.4 ขั้นตอนและวิธีดำเนินการวิจัย

1. ศึกษาการทำงานของโปรแกรม TUNAMI ที่ทำงานแบบลำดับ
2. พัฒนาโปรแกรม TUNAMI ให้ทำงานบนหน่วยประมวลผลกราฟิกได้ โดยใช้ CUDA C
3. ทำการปรับปรุงประสิทธิภาพของโปรแกรมในด้านการใช้หน่วยความจำและการจัดการเธรด ให้สามารถทำงานได้ตามขีดข้อจำกัดของอุปกรณ์ที่ใช้ทดลอง
4. ทำการจับเวลาในการทำงานในส่วนต่างๆของโปรแกรม
5. ทดสอบโปรแกรมโดยปรับปัจจัยต่าง ๆ เช่นขนาดของปัญหา ระยะเวลา ฯลฯ
6. ทำการวัดผลลัพธ์ที่ได้ เช่น ความคลาดเคลื่อนของเวลาที่ใช้กับเวลาที่กำหนดความคลาดเคลื่อนของผลลัพธ์โดยเปรียบเทียบกับข้อมูลที่ได้จากเหตุการณ์สึนามิที่เกิดขึ้นจริง

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. ได้โปรแกรม TUNAMI แบบขนานที่สามารถทำการคำนวณโดยมีข้อจำกัดด้านเวลา
2. ได้ระเบียบวิธีและสมการของการคำนวณแบบขนานบนหน่วยประมวลผลกราฟิกที่สามารถคำนวณระเบียบผลต่างสืบเนื่องแบบมีข้อจำกัดด้านเวลา
3. เป็นแนวทางการวิจัยด้านการคำนวณแบบขนานบนหน่วยประมวลผลกราฟิกที่มีการคำนวณระเบียบผลต่างสืบเนื่องซึ่งสามารถนำไปประยุกต์ให้ทำงานกับการคำนวณแบบอื่นๆ ได้

1.6 ลำดับการจัดเรียงเนื้อหาในวิทยานิพนธ์

วิทยานิพนธ์นี้แบ่งเนื้อหาออกเป็น 6 บทดังต่อไปนี้ บทที่ 1 เป็นบทนำซึ่งกล่าวถึงความเป็นมาและความสำคัญของปัญหา รวมถึงวัตถุประสงค์ของการวิจัย บทที่ 2 กล่าวถึงทฤษฎีพื้นฐานและงานวิจัยที่เกี่ยวข้องในงานวิจัยนี้ บทที่ 3 กล่าวถึงแนวคิดของงานวิจัย บทที่ 4 กล่าวถึงการทดลองและวิเคราะห์ผลการวิจัย บทที่ 5 กล่าวถึงการประเมินผลงานวิจัย และบทที่ 6 กล่าวถึงสรุปผลการวิจัยและข้อเสนอแนะ

1.7 ผลงานที่ได้รับการตีพิมพ์

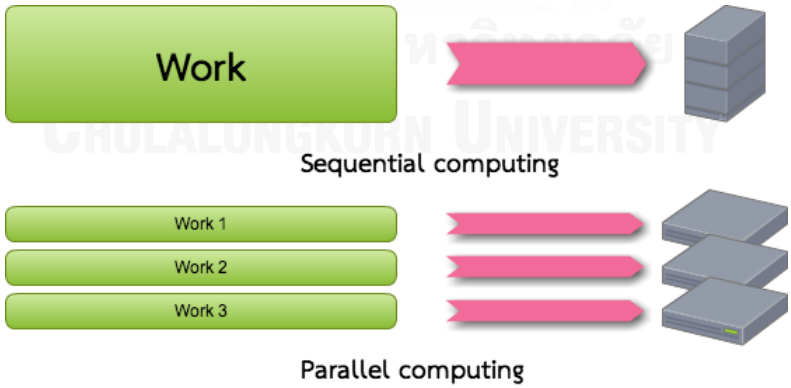
ส่วนหนึ่งของวิทยานิพนธ์นี้ได้ตีพิมพ์เป็นบทความทางวิชาการในหัวข้อเรื่อง “Parallel Computing with Real-Time Constraints on Graphics Processing Unit: A Case Study of Tsunami Simulation” โดย พงษ์พัฒน์ เป้าเพ็ชร์ และวีระ เหมืองสิน นำเสนอในงานประชุมวิชาการ “The 17th International Annual Symposium on Computational Science and Engineering (ANSCSE 17)” ณ มหาวิทยาลัยขอนแก่น จังหวัดขอนแก่น ระหว่างวันที่ 27 - 29 มีนาคม พ.ศ. 2556

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

งานวิจัยนี้เป็นการศึกษาการทำงานการคำนวณแบบขนานบนหน่วยประมวลผลกราฟิก โดยสนใจโปรแกรมจำลองสึนามิเป็นกรณีศึกษา เพื่อปรับปรุงการทำงานของโปรแกรมให้สามารถทำงานได้รวดเร็วและมีประสิทธิภาพในการคำนวณที่ดีขึ้นเป็นหลัก โดยประกอบด้วยรายละเอียดดังนี้

2.1 การคำนวณแบบขนาน (Parallel computing)

การคำนวณแบบขนานคือ การใช้หน่วยประมวลผลหลายตัวมาช่วยกันประมวลผล โดยมีเป้าหมายหลักเพื่อที่จะลดเวลาที่ใช้ในการคำนวณลง โดยจะทำการแบ่งปัญหาที่มีขนาดใหญ่ออกเป็นส่วนย่อยๆ แล้วกระจายออกไปให้แต่ละหน่วยประมวลผลช่วยกันทำงาน ตามรูปที่ 1 ซึ่งในการเขียนโปรแกรมเพื่อให้สามารถคำนวณแบบขนานนั้นจะต่างจากการเขียนโปรแกรมคำนวณแบบลำดับ โดยจะต้องคำนึงประสิทธิภาพของการทำงานของโปรแกรมด้วย เพื่อให้ได้ประโยชน์จากการคำนวณแบบขนานอย่างมีประสิทธิภาพ ซึ่งปัญหาที่จะนำมาแก้ปัญหาแบบขนานได้นั้นข้อมูลควรมีความเป็นอิสระต่อกัน (Data independent) เพื่อให้ได้รับประโยชน์จากการแบ่งงาน ส่วนปัญหาที่ไม่มีความเป็นอิสระของข้อมูลอย่างเช่น การหา Fibonacci Number ก็จะไม่ได้รับประโยชน์

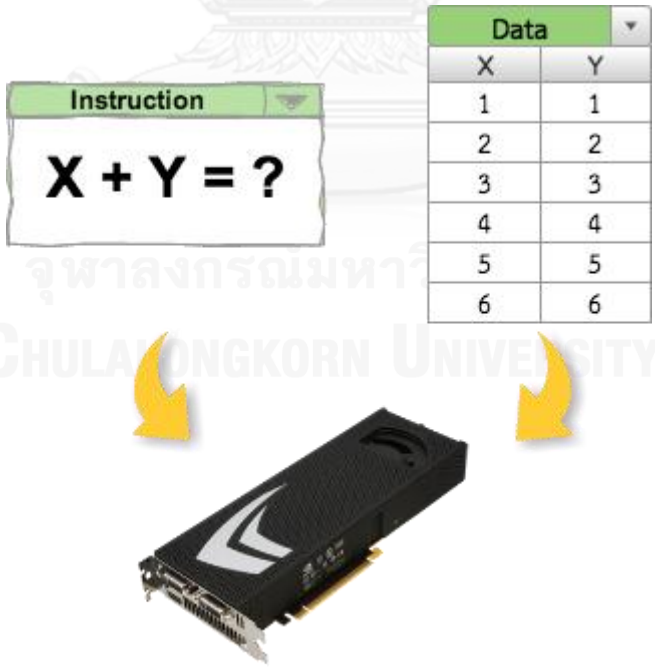


รูปที่ 1 การคำนวณแบบลำดับและการคำนวณแบบขนาน

2.2 CUDA (Compute Unified Device Architecture)

CUDA คือ แพลตฟอร์มของการคำนวณแบบขนาน และโปรแกรมมิ่งโมเดล ทำงานอยู่บนหน่วยประมวลผลกราฟิกของบริษัท NVIDIA โดยการทำงานของ CUDA นั้นจะมีไลบรารี่และคอมไพล์เลอร์ในการแปลงโค้ดที่เขียนมาให้สามารถนำไปประมวลผลบนหน่วยประมวลผลกราฟิกได้ โดยชุดคำสั่งที่สามารถใช้ได้ในปัจจุบันประกอบไปด้วย ชุดคำสั่งภาษาซี/ซีพลัสพลัส (C/C++) ภาษาฟอร์แทรน (FORTRAN) ภาษาไพทอน (Python) ภาษาจาวา (Java) และชุดคำสั่งภาษาอื่นๆที่ได้รับความนิยม โดยทาง NVIDIA เองได้จัดเตรียมให้ผู้พัฒนาโปรแกรมสามารถดาวน์โหลดไปติดตั้งได้โดยมีการจัดเตรียมเฉพาะชุดคำสั่งภาษาซี/ซีพลัสพลัส ส่วนภาษาอื่นๆนั่นจะต้องทำการติดตั้งคอมไพล์เลอร์อื่นๆเพิ่มเติมเองซึ่งมีทั้งแบบ Open Source และ Commercial

โดยในสถาปัตยกรรมของ CUDA นั่นจะเป็นการทำงานแบบ SIMD (Single instruction multiple data) คือ ในหนึ่งชุดคำสั่งจะแบ่งงานออกไปให้เธรดหลายๆเธรดทำงาน ทำให้สามารถทำงานได้กับข้อมูลจำนวนมากที่ทำงานเหมือนๆกัน ซึ่งจะเป็นการเพิ่ม throughput ให้กับโปรแกรม

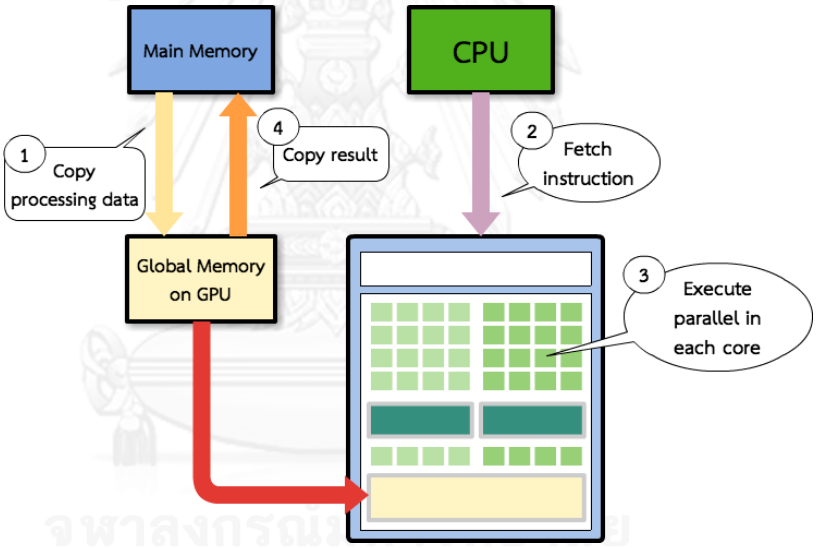


รูปที่ 2 การทำงานแบบ Single instruction multiple data

ในการเขียนโปรแกรม CUDA นั้น ผู้เขียนสามารถเขียนเพื่อสั่งงานให้หน่วยประมวลผลกราฟิกทำงานตามที่ต้องการได้ รวมไปถึงเข้าใช้งานหน่วยความจำ (Memory accesses) ชนิดต่างๆที่อยู่บนหน่วยประมวลผลกราฟิกได้อย่างอิสระ ซึ่งในแต่ละหน่วยความจำนั้นจะมีคุณสมบัติที่แตกต่างกันไป ซึ่งต้องใช้ความเข้าใจเพื่อที่จะสามารถใช้งานหน่วยความจำได้อย่างมีประสิทธิภาพสูงสุด

2.2.1 การทำงานของ CUDA

สำหรับการทำงานของสถาปัตยกรรมของ CUDA นั้นจะแบ่งออกได้ 2 ส่วน คือ การทำงานบนหน่วยประมวลผลหลัก (CPU) และการทำงานบนหน่วยประมวลผลกราฟิก (GPU) โดยมีขั้นตอนการทำงานดังนี้



รูปที่ 3 Processing flow on CUDA [4]

ขั้นตอนแรกสุดของการทำงานบนหน่วยประมวลผลกราฟิกคือ เริ่มจากการคัดลอกข้อมูลในอาร์เรย์ที่จำเป็นในการคำนวณที่อยู่ในหน่วยความจำของหน่วยประมวลผลหลักไปสู่ หน่วยความจำหลักของหน่วยประมวลผลกราฟิก ขั้นตอนต่อมาจะทำการส่งชุดคำสั่ง (Instruction set) จากหน่วยประมวลผลหลักไปเก็บไว้ในหน่วยความจำเฉพาะของหน่วยประมวลผลกราฟิก ในขั้นตอนถัดไปจะทำการประมวลผลตามชุดคำสั่งและข้อมูลที่ถูกคัดลอกมา และขั้นตอนสุดท้ายจะทำการคัดลอกข้อมูลผลลัพธ์จากการคำนวณกลับไปสู่หน่วยความจำหลัก ตามรูปที่ 3

2.2.2 สถาปัตยกรรมของหน่วยความจำ CUDA

สำหรับหน่วยความจำของ CUDA นั้นจะมีความแตกต่างจากหน่วยความจำของหน่วยประมวลผลหลักอยู่หลายอย่าง ซึ่งมีการใช้งานที่ต่างกันซึ่งทำจะทำให้ส่งผลต่อประสิทธิภาพของโปรแกรม ประกอบด้วยหน่วยความจำหลายชนิด ซึ่งแต่ละชนิดมีรายละเอียดดังนี้

2.2.2.1 Global Memory

Global memory คือ หน่วยความจำหลักของหน่วยประมวลผลกราฟิก ซึ่งจะมีขนาดใหญ่ที่สุดของหน่วยความจำทั้งหมดเป็นที่เก็บข้อมูลที่คัดลอกระหว่างหน่วยประมวลผลหลักกับหน่วยประมวลผลกราฟิก ผู้ใช้สามารถเขียนและอ่านข้อมูลได้ แต่มีเวลาในการเข้าถึงหน่วยความจำ (Memory access time) ช้าที่สุด

2.2.2.2 Share Memory

Share memory คือ หน่วยความจำที่ใช้ร่วมกันในแต่ละ Thread block โดย Thread ที่อยู่ในบล๊อคเดียวกันจะแชร์ข้อมูลร่วมกัน สามารถเขียนและอ่านได้ มีความเร็วในการเขียนและอ่านสูงเกือบเทียบเท่ารีจิสเตอร์ แต่มีขนาดเล็กซึ่งขึ้นอยู่กับสถาปัตยกรรมของการ์ดจอในแต่ละรุ่นเพราะฉนั้นการใช้หน่วยความจำชนิดนี้จำเป็นต้องคำนึงถึงการใช้งานข้อมูลซ้ำ (Data reuse) ในการใช้การด้วย

2.2.2.3 Local Memory

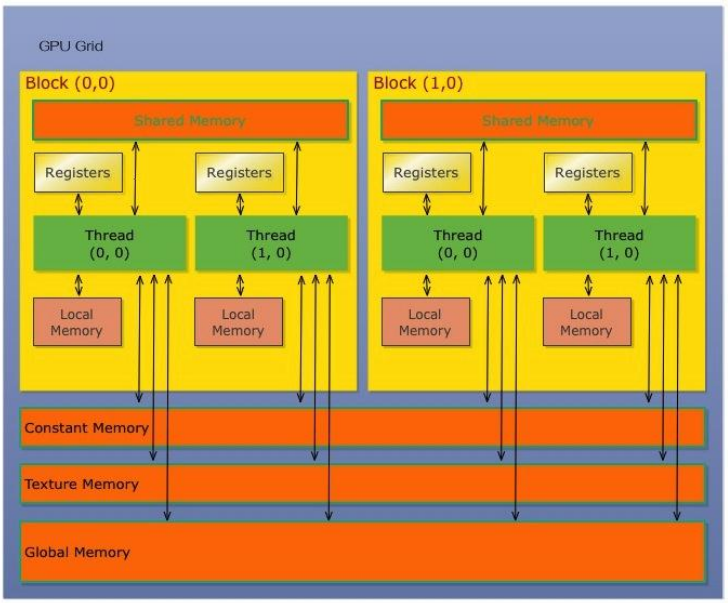
Local memory คือ หน่วยความเฉพาะของแต่ละเทรด สามารถเขียนและอ่านได้ แต่ใช้เวลาในการเข้าถึงข้อมูลช้า จะถูกใช้ในกรณีที่เทรดนั้นมีการใช้รีจิสเตอร์เกินจากปริมาณที่มีอยู่ จึงจะนำ local memory มาช่วยในการเก็บข้อมูล

2.2.2.4 Texture Memory

Texture memory เป็นหน่วยความจำชนิด Read only ผู้ใช้ไม่สามารถเรียกใช้งานได้ตรงๆ ใช้กับงานประเภท Texture ต่างๆ

2.2.2.5 Constant Memory

Constant memory เป็นหน่วยความจำชนิด Read only ทำหน้าที่ในการเก็บค่า Constants และ Arguments ของ kernel



รูปที่ 4 สถาปัตยกรรมของหน่วยความจำบนหน่วยประมวลผลกราฟิก

2.3 ประสิทธิภาพการคำนวณแบบขนานบนหน่วยประมวลผลกราฟิก (Efficiency of Parallel Computing on GPU)

การคำนวณแบบขนานบนหน่วยประมวลผลกราฟิกนั้นมีปัจจัยหลายอย่างที่ส่งผลให้การคำนวณมีประสิทธิภาพลดลงซึ่งมีงานวิจัยต่างๆที่ได้ทำการปรับปรุงชุดคำสั่งของโปรแกรมเพื่อเพิ่มประสิทธิภาพให้ดีขึ้น และจะช่วยเพิ่มอัตราเร็วในการคำนวณขึ้นอีกด้วยซึ่งในงานวิจัยจำนวนหนึ่งที่ได้เสนอออกมาจะขอกล่าวในลักษณะสรุปสั้นๆ ดังนี้

ผลงานวิจัยของ M. Boyer และคณะ [5] ได้นำเสนอการแก้ไขปัญหาที่เกี่ยวกับการคำนวณบนหน่วยประมวลผลกราฟิกแยกเป็น 2 ชนิดคือ ปัญหาเกี่ยวกับความถูกต้องของข้อมูล (Correctness) โดยแก้ปัญหา สถานะการแข่งขันของข้อมูล (Data race condition) และปัญหาเกี่ยวกับประสิทธิภาพการคำนวณ โดยการแก้ปัญหา พื้นที่เก็บข้อมูลซ้อน (Bank conflict) ในงานวิจัยของเขาได้แสดงตัวอย่างของปัญหาที่เกิดขึ้น และวิธีการแก้ไขปัญหาดังกล่าว และยังได้นำเสนอเครื่องมือที่ช่วยแก้ไขปัญหา แต่เครื่องมือที่สร้างขึ้นยังไม่สามารถนำมาใช้กับการคำนวณบนหน่วยประมวลผลกราฟิกจริงได้ ยังทำงานได้บนเครื่องมือจำลอง (Emulator) เท่านั้น

ผลงานวิจัยของ S. Ueng และคณะ [6] ได้นำเสนอเครื่องมือที่ช่วยในการปรับปรุงประสิทธิภาพการคำนวณแบบอัตโนมัติชื่อ CUDA-lite โดยผู้ใช้เขียนชุดคำสั่ง CUDA-C แบบปรกติมาแล้วคำการวิเคราะห์อย่างคราวๆว่าส่วนใดสามารถทำการปรับปรุงได้ หลังการนั้นให้ทำการเพิ่มหมายเหตุ (Annotation) ข้างบนชุดคำสั่งนั้น เมื่อทำการคอมไพล์ชุดคำสั่งผ่านเครื่องมือแล้ว เครื่องมือจะทำการแก้ไขให้เอง แต่เมื่อนำประสิทธิภาพที่ได้มาเทียบกับการแก้ไขด้วยผู้ใช้เองยังได้ประสิทธิภาพไม่เทียบเท่า

ผลงานวิจัยของ Y. Yang และคณะ [7] ได้นำเสนอเครื่องมือที่ช่วยในการปรับปรุงประสิทธิภาพการคำนวณแบบอัตโนมัติ โดยเขาได้บอกว่ามีความสามารถมากกว่า CUDA-lite โดยเขาได้กล่าวว่าเครื่องมือนี้สามารถปรับปรุงได้ในระดับการจัดการจำนวนเทรด และผสานเทรดได้ (Merge threads) ซึ่งได้ประสิทธิที่ดี แต่ก็ยังมีข้อจำกัดอยู่คือ ยังไม่สามารถใช้กับอัลกอรึทึมระดับสูงได้เช่นกัน

ผลงานวิจัยของ D Eschweiler และคณะ [8] ได้นำเสนอถึงปัจจัยที่มีผลต่อประสิทธิภาพในการคำนวนแบบขนานบนหน่วยประมวลผลกราฟิก โดยได้แบ่งปัจจัยออกเป็น 2 ประเภทคือ ปัจจัยด้านการจัดการเทรด (Thread management) และปัจจัยด้านการเข้าถึงข้อมูลในหน่วยความจำ (Memory accesses) ซึ่งในงานวิจัยได้อธิบายถึงปัจจัยพร้อมยกตัวอย่างรวมถึงเสนอวิธีแก้ไขด้วย ซึ่งเป็นประโยชน์ต่องานวิจัยอื่นๆสามารถนำไปวิเคราะห์และปรับปรุงประสิทธิภาพได้

2.4 การปรับปรุงประสิทธิภาพของโปรแกรมบนหน่วยประมวลผลกราฟิก

การคำนวณแบบขนานบนหน่วยประมวลผลกราฟิก คือ การนำหน่วยประมวลผลกราฟิกมาช่วยในการคำนวณปัญหาต่างๆ เนื่องจากหน่วยประมวลผลกราฟิกประกอบด้วยแกนประมวลผล (Core) จำนวนมาก และมีการคำนวณแบบขนานแบบข้อมูล (Data parallelism) จึงสามารถทำการคำนวณงานชนิดเดียวกับได้พร้อมกันที่ละมากๆ โดยการคำนวณแบบนี้จะต้องคำนึงถึงความเกี่ยวเนื่องกันของข้อมูล (Data dependency) โดยถ้าข้อมูลที่นำมาเป็นข้อมูลขาเข้านั้นมีความเป็นอิสระต่อกันก็จะได้รับประโยชน์จากการทำงานบนหน่วยประมวลผลกราฟิกได้อย่างเต็มที่ อย่างไรก็ตาม การคำนวณแบบขนานบนหน่วยประมวลผลกราฟิกนั้น มีปัจจัยที่ทำให้ได้ประสิทธิภาพต่ำมากกว่าการคำนวณแบบขนานบนหน่วยประมวลผลหลัก เนื่องจากมีสภาปัตยกรรมที่แตกต่างกันอยู่หลายอย่าง ปัจจัยที่ทำให้ประสิทธิภาพในการคำนวณแบบขนานบนหน่วยประมวลผลกราฟิกลดลง ตัวอย่างเช่น การทับซ้อนที่เก็บข้อมูล (Bank Conflict) การรวมตัวของหน่วยความจำ (Memory Coalescing) การทำสำเนาการคัดลอก (Replication Copy) เป็นต้น

การปรับปรุงโปรแกรมคำนวณแบบขนานนั้นอาจมีข้อจำกัดอยู่ที่ฮาร์ดแวร์ซึ่งความสามารถสูงสุดที่ทำได้ในแต่ละอุปกรณ์นั่นไม่เท่ากัน หากต้อง การความรวดเร็วของการคำนวณเกินกว่าความสามารถของอุปกรณ์เพื่อที่จะสามารถทำตามข้อจำกัดด้านเวลาอาจต้องใช้วิธีอื่นเข้ามาช่วย ยกตัวอย่างเช่น การปรับลดความละเอียดของข้อมูล (Resolution decrease) การปรับลดจำนวนรอบของการคำนวณ (Time step decrease) ซึ่งจะสามารถช่วยลดเวลาในการทำงานลงได้ตามความต้องการ แต่จะต้องแลกกับความแม่นยำที่ลดลง (Precision decrease)

2.5 การเก็บข้อมูลการทำงานของโปรแกรมคำนวณแบบขนานบนหน่วยประมวลผลกราฟิก

การเก็บข้อมูลการทำงานของโปรแกรมคำนวณแบบขนานบนหน่วยประมวลผลกราฟิกจะเป็นการเก็บข้อมูลเวลาที่ใช้ในการทำงานของโปรแกรมในเชิงสถิติ เพื่อนำมาวิเคราห์ประสิทธิภาพในการทำงานของแต่ละฟังก์ชั่นการทำงานของโปรแกรม โดยจะทำการจับเวลาที่ใช้ในการทำงานแยกออกเป็นแต่ละฟังก์ชั่น ซึ่งจะทำให้ทราบว่าส่วนใดของโปรแกรมที่คำนวณนั้นใช้เวลาในการคำนวณมากน้อยเพียงใด ซึ่งในการรวบรวมข้อมูลแยกออกเป็นฟังก์ชั่นจะทำให้เรา

ทราบว่า ส่วนใดของโปรแกรมที่นำมาทดลองที่จะสามารถนำมาปรับแต่งการทำงานเพื่อเพิ่มประสิทธิภาพได้

2.6 การวัดประสิทธิภาพของโปรแกรมคำนวณแบบขนาน

การวัดประสิทธิภาพของโปรแกรมคำนวณแบบขนานจะใช้วิธีการวัดด้วยการหาอัตราเร็วที่เพิ่มขึ้น (Speedup) โดยจะทำการเปรียบเทียบเวลาที่ใช้ในการคำนวณแบบขนานเปรียบเทียบกับเวลาที่ใช้ในการคำนวณแบบลำดับ โดยมีสมการของการหาอัตราเร็วที่เพิ่มขึ้น ดังนี้

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

โดยที่

S_p คือ อัตราเร็วที่เพิ่มขึ้น

p คือ จำนวนโปรเซสเซอร์ที่ใช้

T_s คือ เวลาที่ใช้ในการคำนวณแบบลำดับ

T_p คือ เวลาที่ใช้ในการคำนวณแบบขนาน

2.7 การคำนวณแบบทันถ่วงที (Real-time computing)

การคำนวณแบบทันถ่วงที คือ การคำนวณที่มีข้อจำกัดในเรื่องของเวลามาเป็นกำหนดผลลัพธ์ (Time constrain) หรือขีดจำกัดของเวลา (Deadlines) โดยในการคำนวณแต่ละครั้งนั้นจะต้องได้ผลลัพธ์ออกมาให้ทันเวลาที่กำหนดไว้ล่วงหน้าหรือเสร็จก่อนเวลาที่กำหนด ถ้าการคำนวณครั้งใดๆไม่สามารถให้ผลลัพธ์ได้ทันเวลาจะถือว่าเป็นการคำนวณที่ล้มเหลว (Computing failure) ซึ่งทำให้การคำนวณในครั้งนั้นไม่สามารถคำนวณผลลัพธ์มาใช้งานได้ หรือใช้ได้แต่ไม่ได้ประโยชน์จากการคำนวณในครั้งนั้น การคำนวณแบบทันถ่วงทีนั้นยังสามารถแบ่งได้เป็น 2 แบบด้วยกันคือ

Hard real-time คือ การคำนวณแบบทันถ่วงทีแบบที่มีข้อจำกัดสูง ซึ่งจะเข็มงวดในเรื่องของขีดจำกัดด้านเวลาสูงมาก ซึ่งจะใช้ในระบบที่มีความเสี่ยงที่จะเกิดความเสียหายแก่ชีวิต และ

ทรัพย์สิน ตัวอย่างเช่น ในระบบของรถยนต์การควบคุมการทำงานของระบบเครื่องยนต์นั้นมีความสำคัญหากทำงานช้าเกินไปอาจทำให้เกิดความเสียหายขึ้นได้ หรือในระบบรักษาความปลอดภัยในโรงงานซึ่งถ้าเกิดอุบัติเหตุขึ้นแต่ระบบรักษาความปลอดภัยได้ล่าช้า อาจทำให้เกิดความเสียหายรุนแรงขึ้นได้

Soft real-time คือ การคำนวณแบบทันถ่วงทีที่มีข้อจำกัดไม่เข็มงวดมากนัก ซึ่งจะเป็นระบบที่ต้องการความรวดเร็วในการคำนวณแต่อาจเกิดความล่าช้าได้บ้าง ไม่เกิดความเสียหายต่อชีวิต หรือทรัพย์สิน ตัวอย่างเช่น การแสดงผลเวลาเข้าจอดของรถไฟฟ้า ซึ่งสามารถคาดเคลื่อนจากเวลาจริงได้ แต่ไม่สร้างความเสียหายต่อชีวิตผู้ใช้บริการ เป็นต้น

นอกจากนี้ การคำนวณแบบที่มีข้อจำกัดด้านเวลานั้นยังมีรูปแบบอื่นๆอีกอย่างเช่น การคำนวณแบบที่มีข้อจำกัดด้านเวลาแต่สามารถใช้วิธีอื่นเพื่อให้สามารถทำงานได้ทันเวลา โดยใช้วิธีการลดความละเอียดในการคำนวณลง (Resolution decrease) ซึ่งทำให้ได้ผลลัพธ์ออกมาทันเวลา แต่อาจต้องสูญเสียความแม่นยำของข้อมูลลงไปบ้าง

2.8 การจำลองสึนามิ (Tsunami simulation)

สึนามิ (Tsunami) คือ การเคลื่อนตัวของปริมาตรน้ำขนาดใหญ่ที่เกิดจากการเคลื่อนตัวของแผ่นเปลือกโลกในมหาสมุทรหรือ เกิดจากการประทุของภูเขาไฟในทะเล ซึ่งเมื่อคลื่นน้ำลึกเคลื่อนที่เข้าสู่บริเวณชายฝั่งจะทำให้คลื่นกระแทกและเกิดมวลน้ำขนาดใหญ่ซึ่งก่อให้เกิดความเสียหายทั้งต่อชีวิตและทรัพย์สินอย่างมหาสาร ตัวอย่างเช่น เหตุการณ์สึนามิที่เกิดขึ้นใน พ.ศ. 2547 [9] ที่เกิดแผ่นดินไหวจากการยุบตัวของเปลือกโลกบริเวณใต้มหาสมุทรอินเดีย ทำให้เกิดความเสียหายเป็นวงกว้างมีผู้เสียชีวิตถึง 230,000 คน และมูลค่าทรัพย์สินที่ความเสียหายถึง 2,800 ล้านดอลลาร์สหรัฐ สาเหตุหนึ่งที่ทำให้เกิดความเสียหายสูง เนื่องจากในบางประเทศยังไม่เคยประสพเกิดเหตุการณ์ ดังกล่าวมาก่อนจึงไม่มีการแจ้งเตือนภัยล่วงหน้า รวมไปถึงยังไม่มีการติดตั้งระบบการเตือนภัยที่บริเวณชายฝั่ง ซึ่งหากมีการติดตั้งระบบเตือนภัยรวมถึง การเตรียมการรับมือล่วงหน้าจะสามารถช่วยลดความเสียหายที่เกิดขึ้นจากเหตุการณ์ดังกล่าวได้ จากเหตุการณ์นั้นเองจึงได้มีการพัฒนาโปรแกรมเพื่อจำลองเหตุการณ์เกิดสึนามิ (Tsunami simulation) ขึ้น

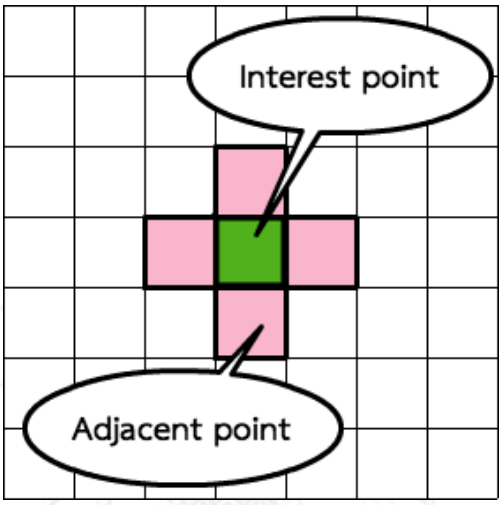
เพื่อคำนวณเหตุการณ์จำลองว่าถ้าเกิดเหตุการณ์ขึ้นจะเกิดผลกระทบที่บริเวณใดบ้างและจะทำความเสียหายประมาณไหน เพื่อที่จะสามารถเตือนภัยและลดความเสียหายลงได้

หนึ่งในโปรแกรมจำลองเหตุการณ์เกิดสึนามิคือ โปรแกรมที่ชื่อว่า TUNAMI (Tohoku University's Numerical Analysis Model for Investigation of Near-field tsunamis) เป็นผลงานวิจัยของ Dr.Fimihiko Imamura จาก Tohoku University โดยมี ผศ.ดร.อาณัติ เรื่องรัศมี อาจารย์ประจำภาควิชาวิศวกรรมโยธา จุฬาลงกรณ์มหาวิทยาลัย ได้นำมาปรับปรุงให้สามารถใช้กับการเกิดเหตุในมหาสมุทรอินเดีย และได้ผลลัพธ์ที่มีความแม่นยำเป็นที่น่าพอใจ แต่ใช้เวลาในการคำนวณในแต่ละพื้นที่การทดลองนาน 4-6 ชั่วโมง [10] ต่อมาได้ถูกนำมาพัฒนาต่อโดย นายกิตติพัฒน์ วิโรจน์ศิริ นักศึกษาภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย ให้สามารถคำนวณแบบขนานในระบบคลัสเตอร์ได้ [11] และได้ถูกนำปรับปรุงประสิทธิภาพโดย นายสิทธิกร ถาวรรัตนวณิช นักศึกษาภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย ให้สามารถคำนวณแบบหลายระดับความละเอียดและปรับปรุงประสิทธิภาพได้ดีขึ้น [12] และได้ถูกนำมาพัฒนาต่อโดยให้สามารถคำนวณบนหน่วยประมวลผลกราฟิกโดย นายเขม อำนวยลาภ และนายฉัตรชัย พีรพัฒน์ดิษฐ์ นักศึกษาภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย [13] ซึ่งจากโปรแกรมที่ถูกพัฒนามาใช้เวลาในการประมวลผลประมาณ 1.30 – 2 ชม. ซึ่งยังไม่สามารถทำในใช้เตือนภัยจริงๆได้ในประเทศไทย เนื่องจากประเทศไทยอยู่บนแผ่นเปลือกโลกยูเรเซียซึ่งใกล้กับแผ่นเปลือกโลกออสเตรเลีย ซึ่งมีระยะห่างจากรอยต่อของแผ่นเปลือกโลกซึ่งขนานไปกับบริเวณพื้นที่ชายฝั่งของประเทศไทยในระยะทางที่เท่าๆกัน ถ้าทำการคำนวณจากระยะทางที่ห่างจากแนวรอยเลื่อนที่อยู่ใกล้ที่สุด กับความเร็วคลื่นใต้น้ำจะพบว่าคลื่นจะใช้เวลาประมาณ 1.30 – 2 ชม. ที่จะเดินทางจากรอยต่อมาถึงบริเวณชายฝั่งของประเทศไทย ดังนั้นความเร็วในการประมวลผลที่ได้ยังไม่สามารถนำมาเตือนภัยแบบเรียลไทม์ได้ สามารถนำมาใช้ได้เพียงเก็บรวบรวมผลลัพธ์ที่ได้มาไว้ใช้ในการเทียบเคียงเท่านั้น

2.9 โปรแกรมจำลองสึนามิแบบลำดับ (TUNAMI program)

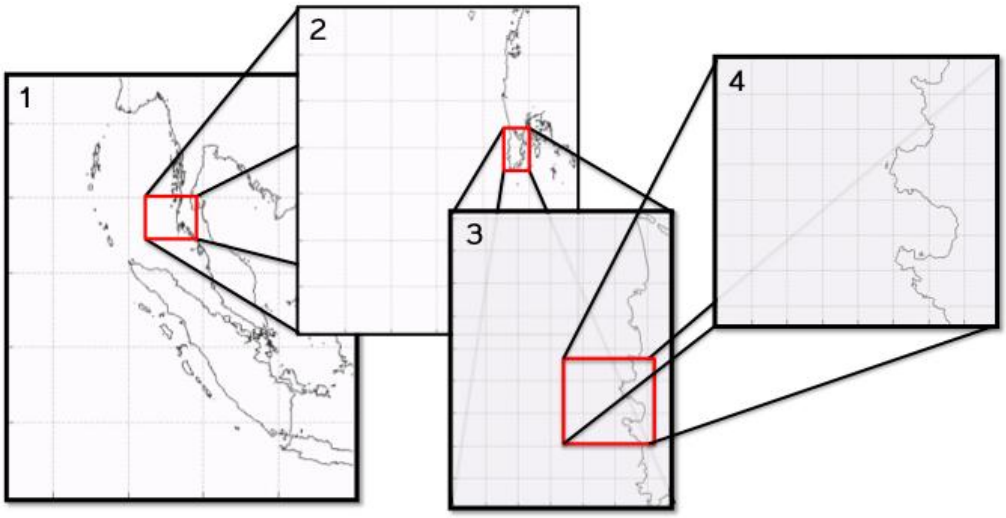
โปรแกรม TUNAMI คือโปรแกรมจำลองเหตุการณ์เกิดสึนามิโดยมีลักษณะการคำนวณโดยแบ่งการทำงานออกเป็น 2 ส่วน คือ ส่วนแรกจะใช้การคำนวณคลื่นสึนามิในระยะไกลโดยใช้การคำนวณแบบพิกัดทรงกลม (Far-field Tsunami Simulation in Spherical Coordinate

System) และส่วนที่สองจะใช้การคำนวณคลื่นสึนามิในระยะใกล้โดยใช้การคำนวณแบบพิกัดคาร์ทีเซียน (Near-field Tsunami Simulation in Cartesian Coordinate System) โดยโปรแกรมจะใช้การคำนวณแบบ Finite Difference Method คือ การหาคำตอบในตำแหน่งที่สนใจจากคำตอบของบริเวณที่อยู่ใกล้เคียง ณ เวลาก่อนหน้า ดังรูปที่ 5



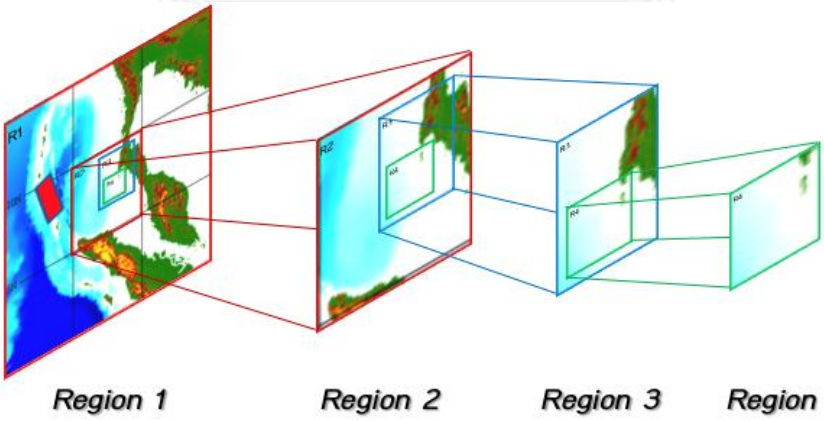
รูปที่ 5 การคำนวณแบบ Finite Difference Method

เนื่องจากในบริเวณที่สนใจในแต่ละตำแหน่งของโปรแกรมนั้น มีความต้องการต่างกัน คือ บริเวณที่ใกล้กับชายฝั่งจะต้องการความละเอียดที่สูงกว่า ส่วนบริเวณที่อยู่ใกล้จากชายฝั่งออกไปต้องการความละเอียดน้อย ดังรูปที่ 6 ดังนั้นโปรแกรม TUNAMI จึงมีวิธีการคำนวณแบบหลายระดับความละเอียด (Multi-scale) โดยในแต่ละระดับความละเอียดจะมีการแลกเปลี่ยนกันของข้อมูลบริเวณขอบของแต่ละพื้นที่



รูปที่ 6 การคำนวณแบบหลายระดับความละเอียด (Multi-scale)

โดยในแต่ละระดับความละเอียดนั้นจะถูกเรียกว่า Region ซึ่งในโปรแกรมจะแบ่งเป็น 4 ระดับเรียงจาก 1 ถึง 4 โดยที่พื้นที่ขนาดใหญ่สุดจะเป็น Region 1 และไล่ระดับความละเอียดไปเรื่อยๆจนถึง Region 4 ซึ่งจะเป็นพื้นที่บริเวณชายฝั่ง ดังรูปที่ 7 ซึ่งในบริเวณ Region 4 จะถูกนำมาใช้ในการแสดงผล



รูปที่ 7 ตัวอย่างระดับความละเอียดในแต่ละระดับของโปรแกรมจำลองสึนามิ

2.10 โปรแกรมจำลองสึนามิด้วยการคำนวณแบบขนาน (Parallel Tsunami Simulation Program)

เนื่องจากโปรแกรมจำลองสึนามิด้วยการคำนวณแบบลำดับ ใช้เวลาประมาณ 4 – 6 ชั่วโมงต่อการคำนวณหนึ่งโซน ซึ่งใช้เวลานานเกินที่จะสามารถนำไปเตือนภัยแบบทันถ่วงทีได้ จึงได้นำโปรแกรมจำลองสึนามิด้วยการคำนวณแบบลำดับนี้มาพัฒนาให้เป็นการคำนวณแบบขนานเพื่อที่จะช่วยเพิ่มความเร็วในการจำลองสึนามิขึ้น

การคำนวณการเกิดคลื่นสึนามิของโปรแกรมจำลองสึนามิด้วยการคำนวณแบบขนานนี้ได้ทำการทดสอบบนระบบคลัสเตอร์ TERA ของ Thai National Grid Center ในมหาวิทยาลัยเกษตรศาสตร์ ประเทศไทย และระบบคลัสเตอร์ TSUBAME ของ GSIC Center ของสถาบันเทคโนโลยีแห่งโตเกียว (Tokyo Institute of Technology หรือ TiTech) ประเทศญี่ปุ่น โปรแกรมจำลองสึนามิด้วยการคำนวณแบบขนานจะใช้ส่วนการคำนวณจากโปรแกรมจำลองสึนามิด้วยการคำนวณแบบลำดับ ที่เขียนด้วยชุดคำสั่งภาษาฟอร์แทรน แต่การเรียกใช้งานส่วนการคำนวณและการควบคุมการทำงานแบบขนานด้วย MPI จะเขียนด้วยชุดคำสั่งภาษาซีพลัสพลัส (C++)

แนวทางในการแก้ปัญหาด้วยการคำนวณแบบขนานนี้ จะประกอบด้วยสองขั้นตอน โดยขั้นแรกจะเป็นการแบ่งการคำนวณออกตามพื้นที่ที่ระดับความละเอียดต่าง ๆ และให้แต่ละหน่วยประมวลผลรับผิดชอบการคำนวณในหนึ่งระดับความละเอียด ซึ่งการแบ่งการคำนวณแบบขนานจำเป็นต้องคำนึงถึงการเชื่อมต่อของข้อมูล หรือการใช้งานข้อมูลร่วมกันระหว่างระดับความละเอียดส่วนขั้นที่สองจะเป็นการแบ่งการคำนวณในพื้นที่เดียวกันออกเป็นหลาย ๆ งาน เพื่อให้แต่ละหน่วยประมวลผลรับผิดชอบในการคำนวณ ซึ่งการแบ่งการคำนวณแบบนี้ จะช่วยลดขนาดของพื้นที่ในการคำนวณต่อหนึ่งหน่วยประมวลผลที่ทำให้ระยะเวลาการคำนวณลดลงด้วย แต่ด้วยความยากในการแบ่งงานให้ได้ประสิทธิภาพและความเร็วอันเนื่องมาจากความซับซ้อนของปัญหาประเภทที่มีหลายระดับความละเอียด

2.11 โปรแกรมจำลองสึนามิด้วยการคำนวณแบบขนานบนหน่วยประมวลผลกราฟิก (Parallel Tsunami Simulation Program on Graphics Processing Unit)

โปรแกรมจำลองสึนามิด้วยการคำนวณแบบขนานบนหน่วยประมวลผลกราฟิกนั่นได้ถูกพัฒนาด้วยชุดคำสั่งภาษาคูด้าฟอร์แทรน (CUDA Fortran) โดยจะทำการส่งฟังก์ชั่นในการคำนวณในบางส่วนเข้าไปในหน่วยประมวลผลกราฟิกทำงานแต่การใช้หน่วยประมวลผลหลักซึ่งมีจำนวนคอร์มากกว่าอยู่หลายเท่า สำหรับโปรแกรมนี้สามารถคำนวณได้เร็วขึ้นโดยรวมทั้งระบบแล้วประมาณ 3 เท่า ซึ่งยังไม่เร็วพอที่จะสามารถนำไปใช้ในระบบเตือนภัยสึนามิได้ เนื่องจากยังต้องใช้เวลาในการทำงานราวๆ 1-2 ชม.

บทที่ 3
แนวคิดของงานวิจัย

แนวคิดของงานวิจัยนี้คือ ทำการศึกษาขั้นตอนในการทำงานของโปรแกรม TUNAMI ที่ใช้ในเป็นกรณีศึกษา และทำการออกแบบระเบียบวิธีการในการพัฒนาโปรแกรมแบบขนานบนหน่วยประมวลผลกราฟิก โดยเริ่มจากการแปลงโปรแกรมจากเดิมที่มีการคำนวณแบบลำดับให้สามารถคำนวณแบบขนานโดยใช้ไลบรารี CUDA C ที่ทำงานบนหน่วยประมวลผลกราฟิกของ NVIDIA แล้วทำการปรับแต่งการทำงานของโปรแกรมเพื่อให้ได้ประสิทธิภาพที่ดี (Optimization) โดยใช้เทคนิคต่าง ๆ ที่เพิ่มประสิทธิภาพในการคำนวณให้ดีขึ้น เช่น การจัดการเธรด การจัดการหน่วยความจำ เป็นต้น แล้วจึงทำการทดลองคำนวณโปรแกรมด้วยการปรับค่าปัจจัยอื่นๆที่ทำให้โปรแกรมทำงานได้ความเร็วเพิ่มขึ้น โดยทดสอบการปรับปัจจัยด้วยวิธีต่างๆและทำการสรุปผลการทดสอบ

โดยในงานวิจัยนี้จะเป็นการคำนวณโดยทำงานบนเครื่องคอมพิวเตอร์เพียงเครื่องเดียวโดยใช้วีดีโอการ์ดที่มีหน่วยประมวลผลกราฟิกเพียงตัวเดียว ซึ่งวีดีโอการ์ดที่นำมาใช้ในงานวิจัยนี้ประกอบด้วยวีดีโอการ์ดที่หาได้ทั่วไปและมีราคาถูก และอีกการ์ดเป็นวีดีโอการ์ดสำหรับการ คำนวณโดยเฉพาะซึ่งจะมีราคาสูงกว่า และเมื่อเปรียบเทียบการคำนวณแบบขนานบนหน่วยประมวลผลกราฟิกกับการคำนวณแบบขนานที่ทำงานบนระบบคลัสเตอร์ จะเห็นว่าการใช้การคำนวณแบบขนานบนหน่วยประมวลผลกราฟิกจะได้เปรียบในเรื่องมูลค่าของอุปกรณ์ที่ใช้ซึ่งจะมีราคาถูกกว่าการใช้งานระบบคลัสเตอร์อย่างมาก และปริมาณการใช้ไฟฟ้าซึ่งการคำนวณบนหน่วยประมวลผลกราฟิกจะใช้เครื่องคอมพิวเตอร์เพียงเครื่องเดียวซึ่งน้อยกว่าการใช้ระบบคลัสเตอร์อย่างมาก รวมไปถึงไม่ต้องเสียเวลาในการติดต่อกันระหว่างโหนดของคลัสเตอร์ ทั้งนี้ขึ้นอยู่กับปริมาณที่นำมาใช้ในการคำนวณด้วย ส่วนโปรแกรมจำลองสึนามิที่นำมาใช้ในงานวิจัยนี้ มีปริมาณที่สามารถทำงานบนหน่วยประมวลผลกราฟิกชุดเดียวได้

3.1 แนวคิดในการแก้ปัญหา

แนวคิดในการดำเนินงานวิจัย มีขั้นตอนดังนี้

1. ศึกษาลักษณะการทำงานของโปรแกรมจำลองสึนามิ (TUNAMI) ที่ใช้การคำนวณแบบลำดับ และใช้การคำนวณแบบขนาน เพื่อดูโครงสร้างของโปรแกรมว่ามีลักษณะในการทำงานอย่างไร และหาเวลาที่ใช้ในการคำนวณในแต่ฟังก์ชั่นของโปรแกรม
2. ทำการพัฒนาโปรแกรมจำลองสึนามิ (TUNAMI) ให้ทำงานบนหน่วยประมวลผลกราฟิกได้โดยใช้ไลบราลี่ของ CUDA C และทำการปรับปรุงประสิทธิภาพของโปรแกรมในด้านการใช้หน่วยความจำ (Memory access) และการจัดการเธรด (Thread management) ให้สามารถทำงานได้ประสิทธิภาพที่ดีขึ้น
3. ทำการทดลองโปรแกรมและเก็บผลการทดลองของโปรแกรมโดยจับเวลาของการทำงานฟังก์ชั่นต่างๆของโปรแกรม
4. ทดสอบหาปรับค่าปัจจัยต่างๆที่จะส่งผลต่อเวลาที่ใช้ในการทำงานของโปรแกรมทั้งปัจจัยที่ส่งผลต่อความแม่นยำของโปรแกรม และไม่ส่งผลต่อความแม่นยำของโปรแกรม
5. ทำการวัดผลลัพธ์ที่ได้ เช่น ความคลาดเคลื่อนของเวลาที่ใช้กับเวลาที่กำหนดความคลาดเคลื่อนของผลลัพธ์โดยเปรียบเทียบกับข้อมูลที่ได้จากเหตุการณ์สึนามิที่เกิดขึ้นจริง

3.2 แนวคิดในการทดลอง

สำหรับแนวคิดในการทดลองของงานวิจัยนี้ จะใช้ข้อกำหนดในเรื่องเวลาที่ใช้ในการทำงานของโปรแกรมมาเป็นตัวทดสอบ ในทั้งระบบคลัสเตอร์คอมพิวเตอร์และเวิร์คสเตชั่น โดยได้มีการออกแบบการทดสอบทั้งหมด 5 รูปแบบดังต่อไปนี้

1. ทดสอบด้วยการคำนวณโปรแกรมจำลองสึนามิแบบลำดับ
2. ทดสอบด้วยการคำนวณโปรแกรมจำลองสึนามิแบบขนานบนหน่วยประมวลผลกราฟิก
3. ทดสอบด้วยการคำนวณโปรแกรมจำลองสึนามิแบบขนานบนหน่วยประมวลผลกราฟิก โดยปรับปรุงโปรแกรมให้ทำการได้มีประสิทธิภาพสูงขึ้นตามระบบคอมพิวเตอร์ที่ใช้

4. ทดสอบด้วยการคำนวณโปรแกรมจำลองสึนามิแบบขนานบนหน่วยประมวลผลกราฟิกที่ได้ปรับปรุงประสิทธิภาพแล้ว โดยปรับค่าปัจจัยที่เกี่ยวข้องด้านเวลาแต่ไม่มีผลต่อความแม่นยำ
5. ทดสอบด้วยการคำนวณโปรแกรมจำลองสึนามิแบบขนานบนหน่วยประมวลผลกราฟิกที่ได้ปรับปรุงประสิทธิภาพแล้ว โดยปรับค่าปัจจัยที่เกี่ยวข้องด้านเวลาและส่งผลกระทบต่อความแม่นยำ

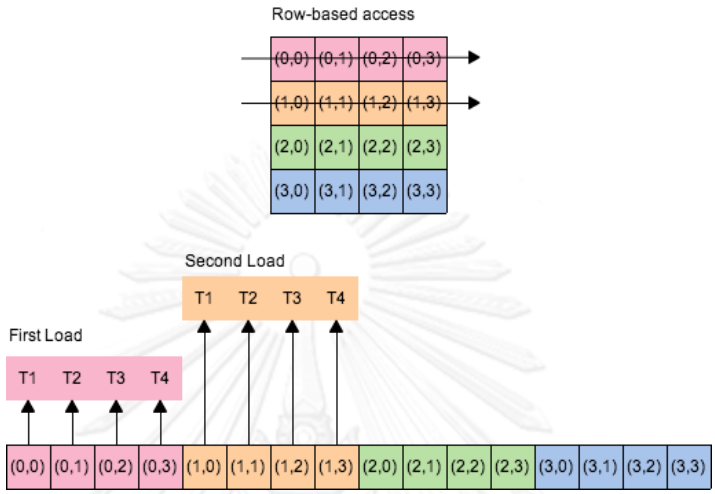
3.3 แนวคิดในการปรับปรุงประสิทธิภาพของโปรแกรมจำลองสึนามิบนหน่วยประมวลผลกราฟิก

ในโปรแกรมจำลองสึนามิเดิมนั้น มีการเขียนในแบบลำดับโดยมีลักษณะการทำงานแบบลำดับโดยคำนวณด้วย Finite difference method และทำงานไล่ลำดับไปตามเวลาในการคำนวณ ทำให้ไม่สามารถใช้เทคนิคของ Shared memory เข้ามาช่วยได้เพิ่มความเร็วในการเข้าถึงหน่วยความจำได้ เนื่องจากการใช้เทคนิค Shared memory จะได้ประโยชน์จากการทำงานแบบ data reuse แต่การทำงานของโปรแกรม TUNAMI ไม่มีการใช้งาน data reuse เพราะจะคัดลอกข้อมูลใหม่มาทับของเก่าทุกรอบของการทำงาน และนอกจากนี้โปรแกรมจำลองสึนามิแบบลำดับแต่เดิม ไม่ได้มีการลำดับของคำสั่งไว้อย่างมีประสิทธิภาพเท่าที่ควร หากทำการปรับปรุงแล้วอาจจะได้รับความเร็วในการคำนวณที่เพิ่มขึ้น ในงานวิจัยนี้มีแนวคิดในการปรับปรุงประสิทธิภาพของโปรแกรมดังนี้

3.3.1 Memory accessing

สำหรับโปรแกรมแบบขนานบนหน่วยประมวลผลกราฟิกนั่น การใช้หน่วยความจำเป็นสิ่งที่ส่งผลต่อประสิทธิภาพของโปรแกรมเป็นอย่างมาก เนื่องจากหน่วยความจำของหน่วยประมวลผลกราฟิกมีเวลาในการเข้าถึงนาน โดยที่หากมีการใช้หน่วยความจำที่ซ้ำๆกันหรือไม่เรียงต่อกันจะทำให้เสียเวลาในการรอคอย (Latency) เนื่องจากการเข้าถึงหน่วยความจำที่ซ้ำๆกันของแต่ละเธรดนั้น จะเข้าถึงได้ในรูปแบบลำดับ ซึ่งทำให้เธรดบางตัวเกิดการรอคอย ซึ่งสามารถลดปัญหาประเภทนี้ได้โดยการเข้าถึงแบบ Memory coalescing ตามรูปที่ 8 โดยที่ให้เข้าถึงหน่วยความจำในตำแหน่งที่ต่อเนื่องกัน โดยในการอ่านหน่วยความจำหนึ่งครั้ง จะได้ข้อมูลที่อยู่ในบล็อค

เดียวกันออกมาด้วยทำให้เธรดอยู่อยู่ติดกันสามารถอ่านข้อมูลได้ทันที จะช่วยลดเวลาในการอ่านหน่วยความจำได้



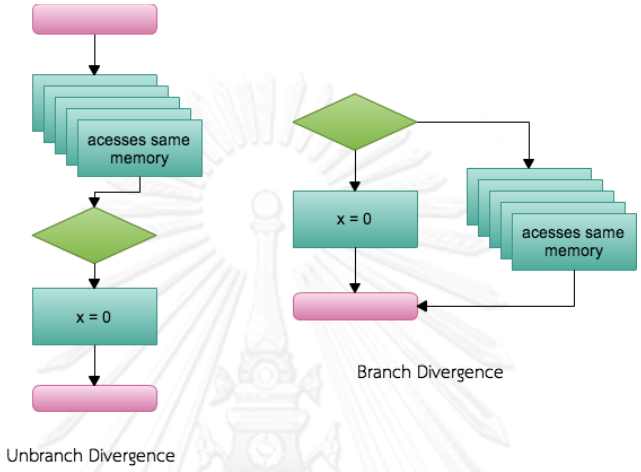
รูปที่ 8 การเข้าถึงหน่วยความจำแบบ Memory Coalescing

นอกจากนี้ยังได้ทำการปรับรูปแบบของอาร์เรย์เดิมซึ่งอยู่ในรูปของ อาร์เรย์ 2 มิติ และอาร์เรย์ 3 มิติ ให้อยู่ในรูปของอาร์เรย์ 1 มิติ โดยแปลงในรูปแบบ row-based array เนื่องจากการเข้าถึงหน่วยความจำแบบ 1 มิตินั้นจะใช้เวลาในการเข้าถึงที่น้อยกว่า ถึงแม้ว่าจะต้องเสียเวลาในการคำนวณหา Index ก็ตาม แต่เมื่อคิดเวลาที่ใช้โดยรวมแล้วจะคุ้มค่ากว่าการใช้อาร์เรย์แบบ 2 มิติ และ 3 มิติ

3.3.2 Branch Divergence

การปรับ branch divergence หรือลำดับการทำงานของโปรแกรมนั้น อาจจะส่งผลต่อประสิทธิภาพของโปรแกรมเพียงเล็กน้อย แต่ถ้าหากมีการลำดับการทำงานของโปรแกรมที่ไม่ดี เช่น มีเงื่อนไขที่มีโอกาสเกิดน้อยมาก แต่เธรดทุกตัวต้องเข้าถึงที่หน่วยความจำเดียวกัน จึงจำเป็นต้องรอคอยที่จะเข้าถึงหน่วยจำนั้นๆเพื่อมาทำการเช็คเงื่อนไข ซึ่งถ้าหากเราสามารถคาดการได้ว่า เงื่อนไข หรือ ชุดคำสั่งไหนที่ไม่จำเป็นต้องทำทั้งหมด เมื่อจัดลำดับการทำงานใหม่ก็จะสามารถได้รับประสิทธิภาพที่เพิ่มขึ้นได้ ตามรูปที่ 9

เนื่องจากการเข้าถึงหน่วยความจำของหน่วยประมวลผลกราฟิกเป็นสิ่งสำคัญของการเขียนโปรแกรมคำนวณแบบขนาน ซึ่งการเข้าถึงหน่วยความจำหลักหนึ่งครั้งจำเป็นต้องใช้สัญญาณนาฬิกา (Clock) ประมาณ 400 – 600 ครั้ง ดังนั้นถ้าสามารถลดเวลาในส่วนนี้ได้ จะเป็นการเพิ่มประสิทธิภาพของการทำงานรวมทั้งโปรแกรม

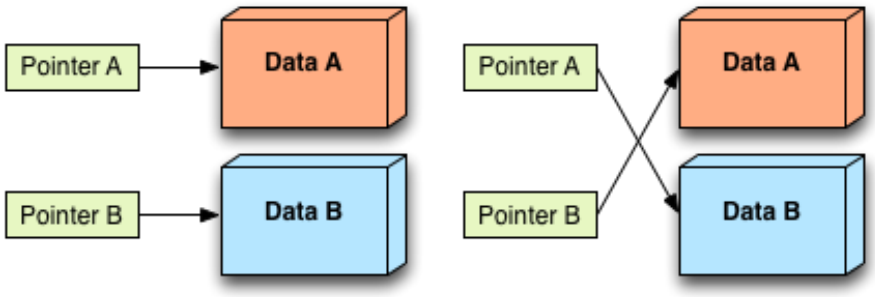


รูปที่ 9 การทำ Branch Divergence

3.3.3 Page flipper

ในโปรแกรมจำลองสึนามินั้น มีฟังก์ชั่นที่ทำการแลกเปลี่ยนข้อมูลระหว่างรอบการทำงานของโปรแกรมในทุก Region คือ ฟังก์ชั่น CHANGE 1-4 โดยโปรแกรมเดิมจะทำการคัดลอกข้อมูลทุกตัวจากรอบการทำงานใหม่กลับไปเป็นข้อมูลตั้งต้นของในรอบถัดไป ซึ่งสามารถใช้เทคนิค Page flipper เพื่อทำการสลับ pointer ที่ชี้อาร์เรย์ โดยจะลดเวลาที่ใช้ในการคัดลอกเหลือเพียงการสลับ pointer เท่านั้น ดังรูปที่ 10

เวลาที่ใช้ในการทำงานของฟังก์ชั่น CHANGE แต่เดิมจะมีการคัดลอกด้วยการใช้ nested loop 2 ชั้น ซึ่งจะใช้เวลาเป็น O(mn) เมื่อใช้เทคนิค Page flipper จะสามารถลดเวลาให้อยู่ในรูปของ O(n) ได้



รูปที่ 10 ตัวอย่างการทำ Page flipper

3.4 วิเคราะห์การทำงานของโปรแกรม

3.4.1 โปรแกรมจำลองสึนามิที่มีการคำนวณแบบลำดับ (TUNAMI program)

ในโปรแกรมจำลองสึนามิเดิมนั้นเขียนขึ้นด้วยการคำนวณแบบลำดับ และมีแบ่งการทำงานอย่างชัดเจนโดยที่ การทำงานของของโปรแกรมจะถูกแบ่งออกเป็นฟังก์ชั่นย่อยๆ แยกส่วนของการทำงานของจากกัน ดังรูปที่ 11 โดยโปรแกรมจะคำนวณต่อเนื่องไปตามเวลา (Time step) และมีเก็บข้อมูลในรูปแบบอาร์เรย์ (Array) ก่อนหน้าไว้เพื่อใช้ในการคำนวณใน Time step ถัดไปของโปรแกรม โดยลักษณะของโปรแกรมจะทำการคำนวณตามกฎการอนุรักษ์มวลก่อน (Conservation of Mass and Momentum) แล้วตามด้วยการคำนวณตามกฎอนุรักษ์โมเมนตัม แล้วทำการบันทึกผลจากการคำนวณต่างๆ โดยสามารถแบ่งการทำงานของโปรแกรม ออกได้เป็น 4 ส่วนหลักๆ ดังนี้

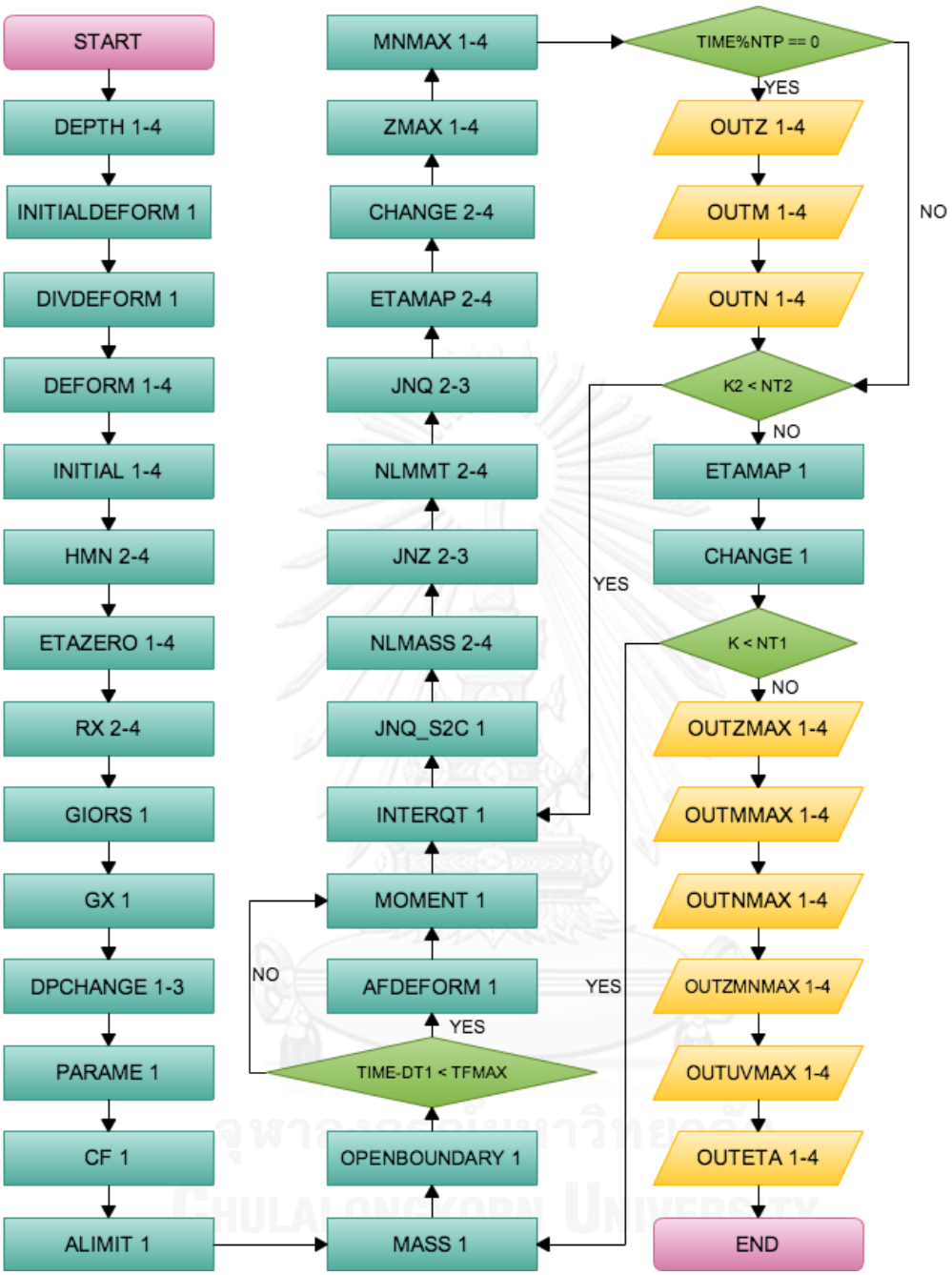
1. Initial section – เป็นส่วนการอ่านค่าต่างๆจากแฟ้มข้อมูลมาเก็บไว้ในอาร์เรย์เริ่มต้น ทำการกำหนดค่าคงที่และตัวแปรเริ่มต้น เพื่อเตรียมข้อมูลไว้ใช้สำหรับการคำนวณต่อไป ประกอบด้วยฟังก์ชั่น ดังต่อไปนี้
   - DEPTH 1-4
   - INITIALDEFORM 1
   - DIVEFORM 1
   - DEFORM 1-4

- INITIAL 1-4
- HMN 2-4
- ETAZERO 1-4
- RX 2-4
- GIORS 1
- GX 1
- DPCHANGE 1-3
- PARAME 1
- CF1
- ALIMIT 1

2. Computation section – เป็นส่วนที่ทำการคำนวณตามกฎการอนุรักษ์มวลและกฎการอนุรักษ์โมเมนตัม และมีการคำนวณเวลาที่คลื่นมาถึง ณ เวลาใดๆ ซึ่งแบ่งออกเป็น 2 ส่วนย่อยดังนี้
   - ที่ Region 1 จะเป็นการคำนวณด้วยสมการเชิงเส้น (Linear Equation) ประกอบด้วยฟังก์ชั่นดังต่อไปนี้
     - MASS 1
     - OPENBOUNDARY 1
     - AFDEFORM 1
     - MOMENT 1
     - INTERQT 1
   - ที่ Region 2-4 จะเป็นการคำนวณด้วยสมการเชิงไม่เส้น (Non-linear Equation) ประกอบด้วยฟังก์ชั่นดังต่อไปนี้
     - NLMASS 2-4
     - NLMMT 2-4
   - นอกจากนี้ยังมีฟังก์ชั่นที่เกี่ยวกับการคำนวณอื่นอีก คือ การหาค่าสูงสุดในจุดต่างของพื้นที่ที่สนใจ ประกอบด้วยฟังก์ชั่นดังต่อไปนี้

- ZMAX 1-4
- MNMAX 1-4
- ETAMAP 1-4

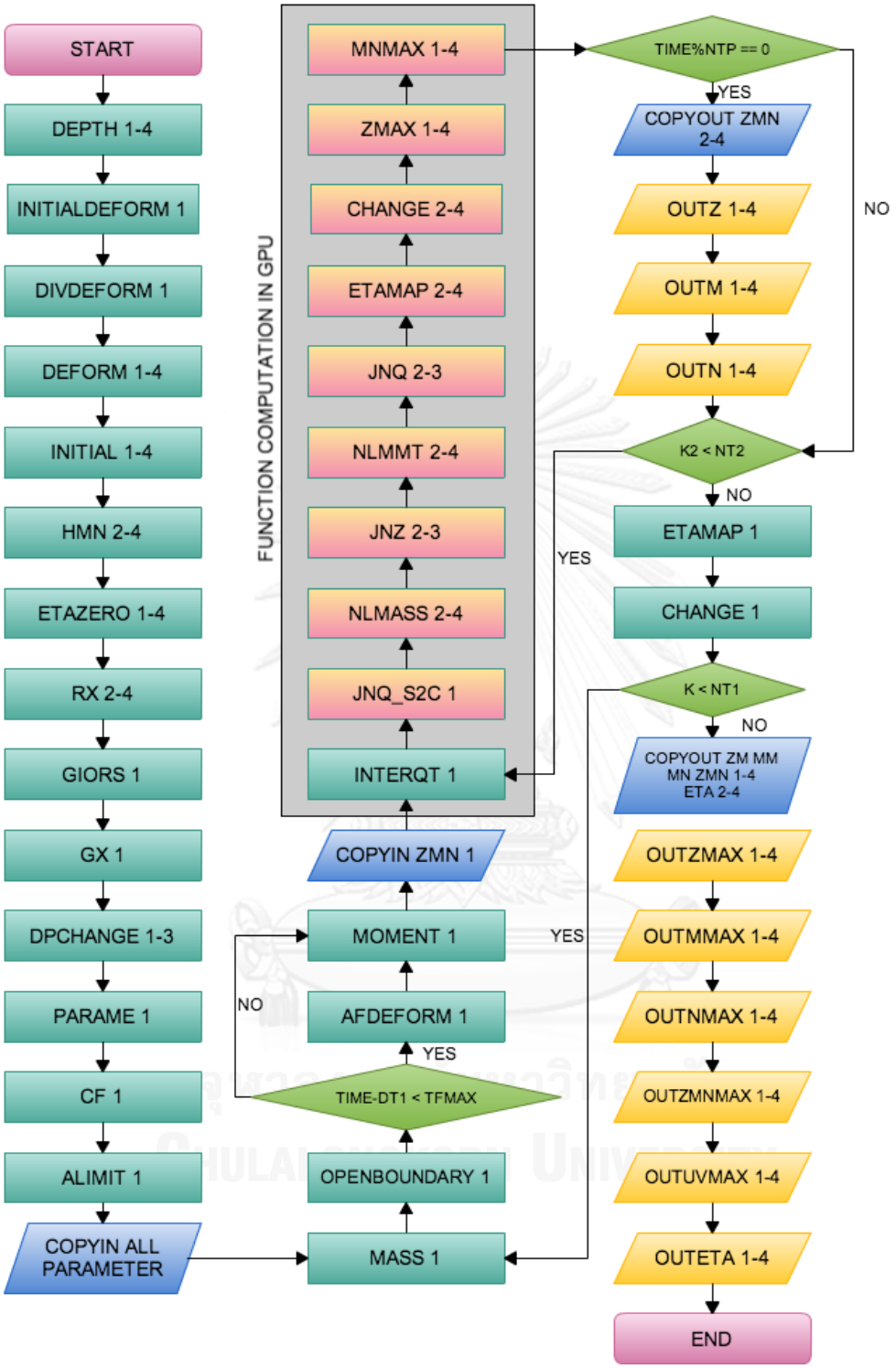
3. Exchange section – เป็นส่วนที่ทำงานเกี่ยวกับการแลกเปลี่ยนข้อมูลในอาร์เรย์ระหว่าง Region ประกอบด้วยฟังก์ชั่นดังต่อไปนี้
   - JNQ_S2C 1
   - JNZ 2-3
   - JNQ 2-3
   - CHANGE 1-4
4. Output section – เป็นส่วนที่ทำงานเกี่ยวกับการเขียนข้อมูลลงไฟล์เอาต์พุต ประกอบด้วยฟังก์ชั่นดังต่อไปนี้
   - OUTZ 1-4
   - OUTM 1-4
   - OUTN 1-4
   - OUTZMAX 1-4
   - OUTMMAX 1-4
   - OUTZMNMAX 1-4
   - OUTUV 1-4
   - OUTETA 1-4



รูปที่ 11 รูปแสดงลำดับการทำงานของโปรแกรมจำลองสึนามิที่มีการคำนวณแบบลำดับ

3.4.2 โปรแกรมจำลองสึนามิที่มีการคำนวณแบบขนานบนหน่วยประมวลผลกราฟิก (TUNAMI program on GPGPU)

เมื่อทำการวิเคราะห์การทำงานของโปรแกรมจำลองสึนามิที่มีการคำนวณแบบลำดับแล้วจะพบว่า โปรแกรมมีส่วนที่ทำงานคือการคำนวณในส่วนของการคำนวณใน R2 – R4 ดังนั้นจะทำการพัฒนาโปรแกรมโดยให้ส่วนที่ทำงานบนหน่วยประมวลผลกราฟิกดังรูปที่ 12 เนื่องจากเป็นส่วนที่มีการทำงานเยอะที่สุด และไม่มีส่วนที่เกี่ยวข้องกับการอ่านและเขียนแฟ้มข้อมูล (Read-Write I/O) ซึ่งหน่วยประมวลผลกราฟิกไม่สามารถทำงานได้โดยตรง จึงเลือกส่วนของการทำงานนี้เข้าไปคำนวณบนหน่วยประมวลผลกราฟิกทั้งหมด ถึงแม้ว่าจะมีบางฟังก์ชั่นในส่วนการทำงานชุดนี้ที่มีการทำงานเป็นแบบลำดับประกอบด้วย JNQ_S2C, JNZ 2-3 และ JNQ 2-4 ซึ่งฟังก์ชั่นเหล่านี้ ไม่สามารถปรับปรุงให้คำนวณแบบขนานได้ แต่ก็ได้ถูกนำเข้าไปคำนวณในหน่วยประมวลผลกราฟิกด้วย เนื่องจากการทำการคัดลอกข้อมูลเข้าออกที่มากจะเสียเวลา และไม่คุ้มค่ากับการยอมเสียเวลาในการคำนวณแบบลำดับ และสาเหตุที่ไม่นำฟังก์ชั่นบางฟังก์ชั่นที่เกี่ยวกับการคำนวณเข้าไปทำงานบนหน่วยประมวลผลกราฟิก เนื่องจากมีการทำงานเพียงเล็กน้อยการคัดลอกข้อมูลเพื่อนำไปคำนวณข้างบนนั้น จะทำให้เสียเวลามากกว่าการทำงานบนหน่วยประมวลผลหลัก



รูปที่ 12 รูปแสดงลำดับการทำงานของโปรแกรมจำลองสึนามิที่มีการคำนวณแบบขนานบนหน่วยประมวลผลกราฟิก

บทที่ 4
การทดลองและวิเคราะห์ผลการทดลอง

4.1 การทดลอง

4.1.1 รูปแบบของการทดลอง

จากแนวคิดในการทดลองที่ได้ออกแบบไว้ ได้นำมาทดสอบเพื่อเก็บข้อมูลของการทำงานของโปรแกรม ซึ่งมีรายละเอียดดังต่อไปนี้

4.1.1.1 ข้อมูลที่ใช้ในการทดลอง

ข้อมูลที่นำมาใช้เป็นข้อมูลขาเข้าของโปรแกรม จะใช้ข้อมูลของพื้นที่และข้อมูลระดับน้ำจริงที่ได้สำรวจโดย ผศ.ดร.อาณัติ เรืองรัศมี และนิสิตภาควิชาวิศวกรรมโยธา จุฬาลงกรณ์มหาวิทยาลัยจุฬาลงกรณ์มหาวิทยาลัย ที่ใช้สำหรับการจำลองสึนามิบางส่วนมาใช้ในการทดสอบการทำงานของโปรแกรม โดยทั้งการทดลองของโปรแกรมจำลองสึนามิแบบลำดับและ การทดลองของโปรแกรมจำลองสึนามิแบบขนานจะใช้ข้อมูลตามตารางที่ 1

จากตารางจะแสดงค่าขนาดของแต่ละ Region โดยที่ No. of Grid จะแสดงถึงขนาดในแกน X และ Y ของแต่ละ Region และขนาดที่ใช้ในการคำนวณของแต่ละ Region จะแสดงอยู่ในคอลัมน์ Grid Size ส่วนจากเปรียบเทียบค่าใน Region ที่เล็กลงไป ส่วนในคอลัมน์ The coordinate of grid on overlapped region จะเป็นการแสดงค่าขอบเขต (boundary) ของ Region ที่อยู่ด้านบน ตัวอย่างเช่น ที่ Region R2(21) จะถูกขยายมาจากตำแหน่ง Region ที่อยู่ในด้านบนที่พิกัด (271, 481) ถึงพิกัด (360, 585) ใน R1(1) เป็นต้น

ตารางที่ 1 พิกัดและขนาดของพื้นที่ในแต่ละระดับความละเอียดที่ใช้ในการทดลองโปรแกรมจำลองสึนามิ

Region levels	Region ID	No. of Grid		Grid Size	The coordinate of grid on overlapped region			
					From		To	
		X	Y		X	Y	X	Y
R1	1	690	840	579,600	-	-	-	-
R2	21	721	841	606,361	271	481	360	585
R3	211	241	325	78,325	482	424	561	531
R4	2111	487	325	158,275	68	68	229	175

4.1.1.2 เครื่องมือที่ใช้ในการทดลอง

เครื่องมือที่ใช้ในการวิจัยนี้ประกอบด้วย

1. โปรแกรมจำลองสึนามิด้วยการคำนวณแบบลำดับ (TUNAMI)
2. โปรแกรมจำลองสึนามิด้วยการคำนวนแบบขนานบนหน่วยประมวลผลกราฟิก (TUNAMI on GPGPU)
3. GCC compiler version 4.4.6 20110731
4. Gfortran based on GCC 4.4.6 20110731
5. CUDA compilation tools, release 4.1, V0.2.1221
6. ระบบคลัสเตอร์คอมพิวเตอร์ Pegasus – เป็นคลัสเตอร์คอมพิวเตอร์ที่มีขนาด 8 เครื่องประมวลผล และ 1 ฟรอนเอ็นต์ แต่ละเครื่องประมวลผลมีคุณสมบัติดังนี้
   - หน่วยประมวลผล Intel Xeon CPU L5520 @ 2.27GHz (4 cores)
   - หน่วยความจำหลัก 4 Gigabytes
   - วีดีโอการ์ด NVIDIA Quadro FX 3800
   - พื้นที่เก็บข้อมูล 248 Gigabytes SAS per Nodes

- เชื่อมต่อภายในแบบ Gigabyte Ethernet

7. เครื่องเวิร์คสเตชั่น SPACE LAB – มีคุณสมบัติดังนี้
   - หน่วยประมวลผล Intel Core i5 CPU 650 @ 3.20GHz (2 cores)
   - หน่วยความจำหลัก 8 Gigabyte
   - วีดีโอการ์ด NVIDIA GeForce GT 430
   - พื้นที่เก็บข้อมูล 1 Terabytes S-ATA II

NVIDIA Quadro FX 3800	
CUDA Capability version	1.3
Global memory	1 GB DDR3
CUDA Cores	192 Cores
GPU Clock Speed	1.20 GHz
Memory Clock rate	800 MHz
Memory Bus Width	256-bit

GeForce GT 430	
CUDA Capability version	2.1
Global memory	1 GB DDR3
CUDA Cores	96 Cores
GPU Clock Speed	700 MHz
Memory Clock rate	800 MHz
Memory Bus Width	128-bit

เมื่อเปรียบเทียบคุณสมบัติของหน่วยประมวลผลกราฟิกบนทั้ง 2 วีดีโอการ์ด พบว่าทั้ง 2 การ์ดมีหน่วยความจำเท่ากันคือ 1 GB (DDR3) ถึงแม้ว่า Quadro FX 3800 จะมีสถาปัตยกรรมที่เก่ากว่าคือรุ่น 1.3 แต่เนื่องจากเป็นวีดีโอการ์ดสำหรับการทำงานประเภทงานคำนวณโดยเฉพาะ จึงมีจำนวน Cores มีมากกว่า รวมไปถึง GPU Clock speed, Memory Clock rate และ Memory Bus width มีมากกว่าเช่นกัน

4.1.2 ผลการทดลอง

4.1.2.1 การทดลองด้วยโปรแกรมจำลองสึนามิด้วยการคำนวณแบบลำดับ

การทดลองนี้ใช้โปรแกรมจำลองสึนามิด้วยการคำนวณแบบลำดับที่คอมไพล์ด้วย Gfortran โดยที่ใช้การคำนวณด้วยจำนวนหน่วยประมวลผลหนึ่งหน่วย (หนึ่งคอร์) ด้วยการทดลองบนระบบคลัสเตอร์เพกาซัส และเวิร์คสเตชั่นสเปซแลบ โดยเก็บผลของการคำนวณมาเพื่อใช้เป็นฐานเวลาในการเปรียบเทียบตามตารางที่ 2 โดยจะใช่วิธีในการจับเวลาในการทำงานด้วยฟังก์ชั่น Clock ของภาษา C ดังรูปที่ 13 สำหรับการจับเวลาฟังก์ชั่นการทำงานบนหน่วยประมวลผลกราฟิกหรือเรียกว่า เคอร์แนล (Kernel) จะเพิ่มคำสั่ง cudaThreadSynchronize() หลังเคอร์แนลทุกครั้ง เพื่อรอให้การทำงานทุกเธรดทำงานให้เสร็จพร้อมกันก่อนจึงค่อยวัดเวลาหลังการทำงานเคอร์แนล ดังรูปที่ 14

```
fstart = clock();
moment(IF1, JF1, is, js, ie, je, z1, z1next, m1,
    m1next, n1, n1next, r2, r3, r4, r5, hz1, GX);
fend = clock();
tmoment += (float)(fend - fstart)/(float)(CLOCKS_PER_SEC);
```

รูปที่ 13 การจับเวลาการทำงานของฟังก์ชั่นภาษา C

```
fstart = clock();
cuda_interqt<<<dimGrid1, dimBlock>>>(IF1, JF1, k2, NT2, m1_d,
    m1_dnext, n1_d, n1_dnext, m1t_d, m1t_dnext, n1t_d, n1t_dnext);
cudaThreadSynchronize();
fend = clock();
tinterqt += (float)(fend - fstart)/(float)(CLOCKS_PER_SEC);
```

รูปที่ 14 การจับเวลาการทำงานของเคอร์แนลบนหน่วยประมวลผลกราฟิก

ในตารางที่ 2 จะแสดงฟังก์ชั่นที่ทำงานของแต่ละ Region และเวลาที่ใช้ในการทำงานแยกออกมาในแต่ละฟังกชั่นทั้งในระบบคลัสเตอร์เพกาซัส และ เวิร์คสเตชั่นสเปซแลบ โดยในโปรแกรมจะมีเปอร์เซ็นต์ของการทำงานของฟังก์ชั่นในโปรแกรมที่น่าสนใจอยู่ 2 ส่วนหลักด้วยกัน ประกอบด้วย

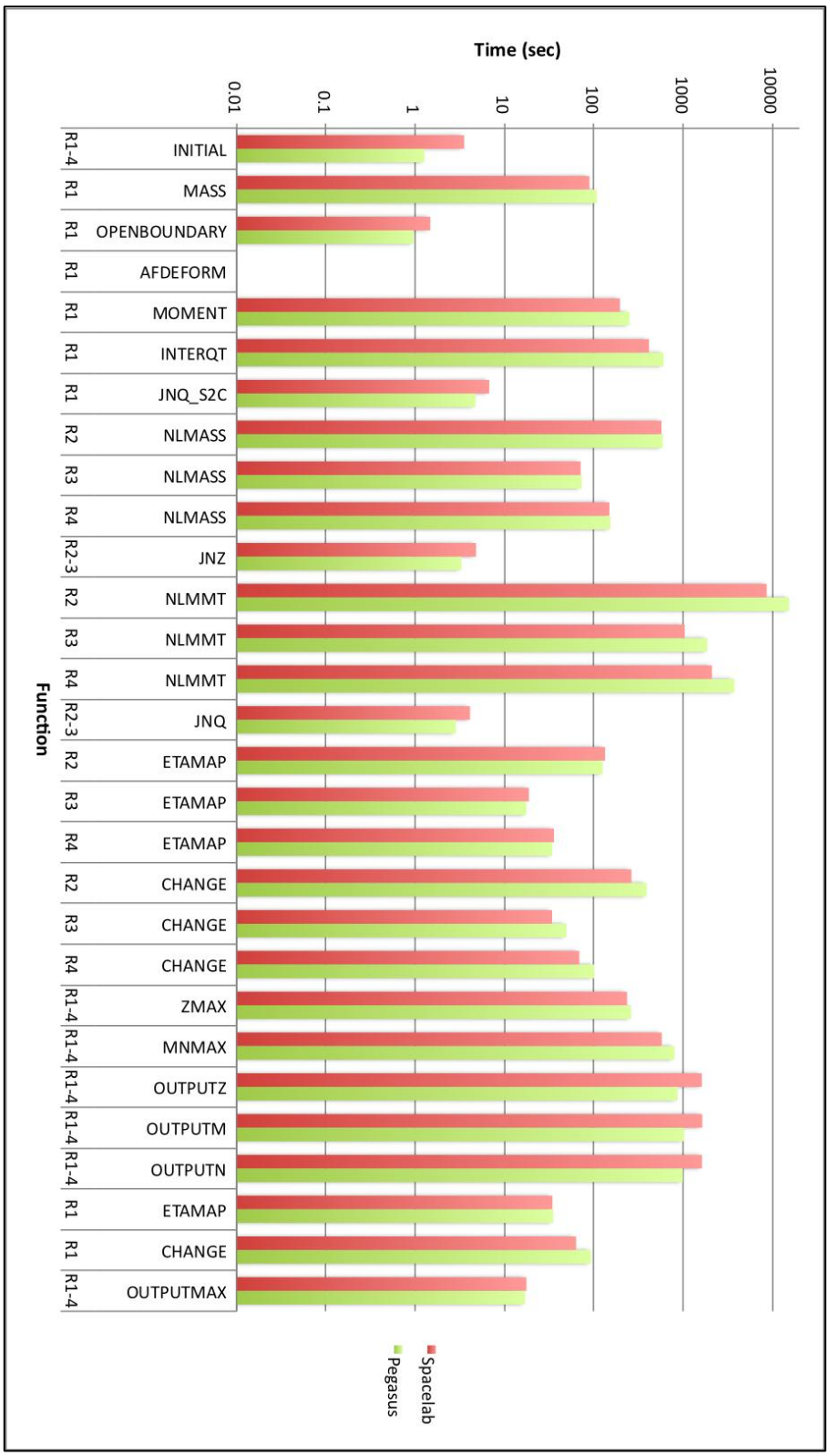
ส่วนแรกการทำงานในส่วนของ Computation section มีฟังก์ชั่นที่มีเวลาในการคำนวณมากที่สุดคือ NLMMT R2 – R4 ฟังก์ชั่นนี้เป็นฟังก์ชั่นที่คำนวณตามกฎอนุรักษ์โมเมนตันของ Region 2 ถึง Region 4 โดยที่เครื่องเวิร์คสเตชั่นสเปซแลบ จะใช้เวลาในการทำงาน 8,583 วินาที, 1,037 วินาที และ 2,094 วินาที ตามลำดับ และที่ระบบคลัสเตอร์เพกาซัส จะใช้เวลาในการทำงาน 15,316 วินาที, 1,859 วินาที และ 3,729 วินาทีตามลำดับ โดยเวลาที่ใช้ในการทำงานจะขึ้นอยู่กับขนาดของ Gridsize ที่ระบุในตารางที่ 1 สามารภคำนวณหาเวลาที่ใช้ในการคำนวณต่อหนึ่งจุด ได้ประมาณ 0.014 วินาที, 0.013 วินาที และ 0.013 วินาที ตามลำดับ Region 2-4 บนเครื่องสเปซแลบ และจะใช้เวลาต่อจุดประมาณ 0.025 วินาที, 0.023 วินาที และ 0.024 วินาที ตามลำดับ Region 2-4 บนเครื่องสเปซแลบ โดยไม่ขึ้นอยู่กับ Region เนื่องจากมีการคำนวณที่เหมือนกัน โดยสำหรับฟังก์ชั่น NLMMT ที่ Region 2 จะใช้เวลาในการทำงานมากที่สุด คิดเป็นสัดส่วนเปอร์เซ็นต์ของการทำงานได้ร้อยละ 43 และ 55 โดยถ้ารวมการทำงานของ 3 Region เข้าด้วยกัน จะพบว่ามีสัดส่วนที่เกินครึ่งของเวลาที่ใช้ในการคำนวณทั้งหมดในทั้งสองเครื่องที่ใช้ทดลอง ซึ่งหากลดเวลาในการทำงานของฟังก์ชั่น NLMMT ได้มากก็จะส่งผลให้โปรแกรมทำงานได้รวดเร็วขึ้นมาก ส่วนฟังก์ชั่นอื่นใน Computation section ที่น่าสนใจประกอบด้วย NLMASS 2-4 เป็นฟังก์ชั่นที่คำนวณกฎอนุรักษ์มวล เมื่อรวมทุก Region จะอยู่ที่ประมาณร้อยละ 3 – 4 ซึ่งเมื่อเทียบกับฟังก์ชั่นอื่นๆของโปรแกรมถือว่ามีนัยสำคัญ และอีกส่วนที่น่าสนใจคือ ฟังก์ชั่น ZMAX 1-4 และ MNMAX 1-4 คือฟังก์ชั่นสำหรับการคำนวณหาค่าสูงสุดในแต่ละจุดพิกัดคิดเป็นร้อยละ 4 ของทั้งหมดของโปรแกรมบนทั้งสองเครื่องการทดลอง ซึ่ง หากสามารถลดเวลาในการทำงานในส่วนต่างๆที่กล่าวมาได้ จะสามารถเพิ่มประสิทธิภาพของการทำงานโดยรวมของโปรแกรมได้มาก

ตารางที่ 2 ตารางผลการคำนวณโปรแกรมจำลองสึนามิแบบลำดับ

Region	Function	Machine		Percentage	
		Spacelab (sec)	Pegasus (sec)	% Spacelab	% Pegasus
R1-4	INITIAL	3.533	1.272	0.018	0.005
R1	MASS	89.144	107.517	0.455	0.391
R1	OPENBOUNDARY	1.478	0.961	0.008	0.003
R1	AFDEFORM	0.004	0.005	0.000	0.000
R1	MOMENT	195.855	249.952	1.001	0.909
R1	INTERQT	413.691	602.047	2.113	2.190
R1	JNQ_S2C	6.747	4.739	0.034	0.017
R2	NLMASS	570.455	588.043	2.914	2.139
R3	NLMASS	71.076	72.975	0.363	0.265
R4	NLMASS	149.066	152.787	0.762	0.556
R2-3	JNZ	4.804	3.286	0.025	0.012
R2	**NLMMT**	**8583.38**	**15316.14**	**43.849**	**55.717**
R3	**NLMMT**	**1037.775**	**1859.098**	**5.302**	**6.763**
R4	**NLMMT**	**2094.493**	**3729.907**	**10.700**	**13.569**
R2-3	JNQ	4.114	2.853	0.021	0.010
R2	ETAMAP	133.377	125.602	0.681	0.457
R3	ETAMAP	18.738	17.519	0.096	0.064
R4	ETAMAP	35.734	34.184	0.183	0.124
R2	CHANGE	263.214	388.475	1.345	1.413
R3	CHANGE	34.053	49.273	0.174	0.179
R4	CHANGE	68.749	100.874	0.351	0.367
R1-4	ZMAX	235.012	260.85	1.201	0.949
R1-4	MNMAX	575.819	797.73	2.942	2.902
R1-4	**OUTPUTZ**	**1616.185**	**863.396**	**8.256**	**3.141**
R1-4	**OUTPUTM**	**1631.888**	**1024.749**	**8.337**	**3.728**
R1-4	**OUTPUTN**	**1621.501**	**990.008**	**8.284**	**3.601**
R1	ETAMAP	34.352	35.026	0.175	0.127
R1	CHANGE	63.17	92.724	0.323	0.337
R1-4	OUTPUTMAX	17.586	16.939	0.090	0.062
TOTAL		19574.993	27488.931	100.000	100.000

ส่วนที่สองที่น่าสนใจได้แก่ส่วนการทำงานของ Output section โดยฟังก์ชั่นที่น่าสนใจประกอบด้วย OUTPUTZ 1-4, OUTPUTM 1-4 และ OUTPUTN 1-4 โดยที่ฟังก์ชั่น OUTPUTZ ทำหน้าที่เขียนระดับน้ำบนจุดพิกัดต่างๆในพื้นที่ขอบเขตที่สนใจทั้ง 4 Regions ส่วน OUTPUTM จะทำหน้าที่เขียนค่าของแรงในแนวแกน X บนจุดพิกัดต่างๆในพื้นที่ขอบเขตที่สนใจ และส่วนสุดท้าย OUTPUTN ทำหน้าที่ในการเขียนค่าแรงในแนวแกน Y บนจุดพิกัดต่างๆในพื้นที่ขอบเขตที่สนใจ ซึ่งเวลาที่ใช้ในการคำนวณฟังก์ชั่นต่างๆ ประมาณ 1,600 วินาที บนเครื่องเวิร์คสเตชั่นสเปซแลบ และใช้เวลาต่อฟังก์ชั่นประมาณ 800 – 1,000 วินาที บนระบบคลัสเตอร์เพกาซัส ซึ่งหากรวมของทุกฟังก์ชั่นใน Output section จะคิดเป็นร้อยละ 24 ของโปรแกรมบนเครื่องเวิร์คสเตชั่นสเปซแลบ และคิดเป็นร้อยละ 10 ของโปรแกรมบนระบบคลัสเตอร์เพกาซัส จะเห็นได้ว่าเวลาที่ใช้ในส่วนของ Output บนระบบคลัสเตอร์เพกาซัสมีสัดส่วนของเวลาที่น้อยกว่าบนเครื่องสเปซแลบ เนื่องจากบนระบบคลัสเตอร์เพกาซัสที่การใช้ฮาร์ดดิสก์แบบ SAS ซึ่งมีแบนวิธในการเขียนและอ่านข้อมูลที่สูงกว่าฮาร์ดดิสก์ทั่วๆไป ในขณะที่เครื่องสเปซแลบมีฮาร์ดดิสก์แบบ SATA-II ซึ่งเป็นฮาร์ดดิสก์ที่ใช้กันอยู่ทั่วๆไปในปัจจุบัน

จากรูปที่ 15 กราฟแสดงเวลาในการทำงานโปรแกรมจำลองสึนามิแบบลำดับ ซึ่งเป็นกราฟแบบ logarithm โดยจะทำให้เห็นเวลาที่ใช้ในการทำงานในฟังก์ชั่นต่างๆของโปรแกรมได้อย่างชัดเจน โดยจากกราฟจะเห็นว่าการทำงานของฟังก์ชั่น NLMMT และฟังก์ชั่น OUTPUT ต่างๆจะใช้เวลามากกว่า 1,000 วินาที ส่วนฟังก์ชั่นอื่นๆ มีเวลาในการทำงานอยู่ที่ประมาณ 0 – 100 วินาที เป็นส่วนใหญ่



รูปที่ 15 กราฟแสดงเวลาในการทำงานของฟังก์ชันในโปรแกรมจำลองสึนามิแบบลำดับ

4.1.2.2 การทดลองด้วยโปรแกรมจำลองสึนามิด้วยการคำนวณแบบขนาน

หลังจากที่ทราบผลของเวลาที่ใช้ในการคำนวณโปรแกรมจำลองสึนามิแบบลำดับแล้ว จึงได้ทำการพัฒนาโปรแกรมจำลองสึนามิด้วยการคำนวณแบบขนานโดยคำนึงถึงการทำงานของโปรแกรมตามแนวคิดของการพัฒนาโปรแกรมที่กล่าวในบทที่ 3 และทำการทดลองเพื่อเก็บค่าเวลาที่ใช้ในการคำนวณของโปรแกรมโดยแบ่งการเก็บค่าของเวลาออกเป็นฟังก์ชั่นย่อยๆ โดยแบ่งออกเป็น 3 ตาราง โดยที่ตารางที่ 3 จะเป็นเปรียบเทียบเวลาในการทำงานของโปรแกรมจำลองสึนามิแบบลำดับและ โปรแกรมจำลองสึนามิบนหน่วยประมวลผลกราฟิกทั้งสองเครื่องการทดลอง ตารางที่ 4 เป็นผลจากการทดลองบนระบบคลัสเตอร์เพกาซัส ส่วนในตารางที่ 5 เป็นผลการทดลองจากเวิร์คสเตชั่นสเปซแลบ โดยในตารางจะแยกเวลาของแต่ละฟังก์ชั่นของการทำงาน ซึ่งจะมีฟังก์ชั่นที่เพิ่มจากการคำนวณแบบลำดับมาคือ เวลาของการคัดลอกข้อมูลเข้าและออกจากหน่วยความจำของหน่วยประมวลผลกราฟิกรวมทั้งโปรแกรม โดยที่ทั้งสองตารางจะแสดงผลการทดลองของโปรแกรมจำลองสึนามิแบบขนานทั้งแบบที่ยังไม่ได้วิเคราะห์และปรับปรุงการทำงานและรูปแบบการปรับปรุงประสิทธิภาพของโปรแกรม โดยประกอบด้วยคอลัมน์ 5 คอลัมน์ คือ Naïve (ยังไม่ได้รับการปรับปรุง 32 T/B), 32 T/B (32 Threads per block), 64 T/B (64 Threads per block), 128 T/B (128 Threads per block) และ 256 T/B (256 Threads per block)

ตารางที่ 3 ตารางผลการคำนวณโปรแกรมจำลองสึนามิแบบขนานบนหน่วยประมวลผลกราฟิกเปรียบเทียบกับโปรแกรมจำลองสึนามิแบบลำดับ

Region	Function	Pegasus			Spacelab		
		Sequential	Naïve	speedup	Sequential	Naïve	speedup
R1-4	INITIAL	1.272	1.05	1.21	3.533	1.59	2.22
R1	MASS	107.517	105.59	1.02	89.144	148.19	0.60
R1	OPENBOUNDARY	0.961	1.41	0.68	1.478	3.07	0.48
R1	AFDEFORM	0.005	0.01	0.50	0.004	0.01	0.44
R1	MOMENT	249.952	542.81	0.46	195.855	495.90	0.39

R1	INTERQT	602.047	32.22	18.68	413.691	279.44	1.48
R1	JNQ_S2C	4.739	169.95	0.03	6.747	78.45	0.09
R2	NLMASS	588.043	33.14	17.74	570.455	172.42	3.31
R3	NLMASS	72.975	5.35	13.64	71.076	28.88	2.46
R4	NLMASS	152.787	9.23	16.55	149.066	44.79	3.33
R2-3	JNZ	3.286	214.12	0.02	4.804	118.41	0.04
R2	**NLMMT**	**15316.14**	**622.96**	**24.59**	**8583.38**	**1725.73**	**4.97**
R3	**NLMMT**	**1859.098**	**82.96**	**22.41**	**1037.775**	**223.61**	**4.64**
R4	**NLMMT**	**3729.907**	**154.30**	**24.17**	**2094.493**	**440.47**	**4.76**
R2-3	JNQ	2.853	259.98	0.01	4.114	110.53	0.04
R2	ETAMAP	125.602	16.20	7.75	133.377	124.55	1.07
R3	ETAMAP	17.519	2.78	6.30	18.738	19.53	0.96
R4	ETAMAP	34.184	4.33	7.89	35.734	34.58	1.03
R2	CHANGE	388.475	27.61	14.07	263.214	207.11	1.27
R3	CHANGE	49.273	3.75	13.14	34.053	30.03	1.13
R4	CHANGE	100.874	7.72	13.07	68.749	56.82	1.21
R1-4	ZMAX	260.85	25.71	10.15	235.012	198.88	1.18
R1-4	MNMAX	797.73	62.76	12.71	575.819	392.07	1.47
R1-4	OUTPUTZ	863.396	323.67	2.67	1616.185	518.45	3.12
R1-4	OUTPUTM	1024.749	365.22	2.81	1631.888	521.66	3.13
R1-4	OUTPUTN	990.008	356.61	2.78	1621.501	519.32	3.12
R1	ETAMAP	35.026	37.38	0.94	34.352	29.34	1.17
R1	CHANGE	92.724	80.36	1.15	63.17	68.43	0.92
R1-4	OUTPUTMAX	16.939	5.03	3.37	17.586	4.86	3.62
Memory copy Host to Device		0	195.79	0.00	0	208.56	0.00
Memory copy Device to Host		0	2.64	0.00	0	2.74	0.00
TOTAL		27488.931	3752.63	7.33	19574.993	6808.38	2.88

จากตารางที่ 3 ผลลัพธ์จากการทำงานแบบดั้งเดิมเมื่อเปรียบเทียบกับโปรแกรมจำลองสีนามิที่คำนวณแบบลำดับนั้นจะเห็นว่าสามารถลดเวลาในการทำงานได้จาก 27,488 วินาที เหลือ 3,752 วินาที คิดเป็น 7.33 เท่า ที่ฟังก์ชั่น NLMMT ซึ่งโปรแกรมจำลองสีนามิแบบลำดับใช้เวลามากกว่า 20,000 วินาที หรือร้อยละ 70 ของโปรแกรม เมื่อทำงานแบบขนานบนหน่วยประมวลผลกราฟิกจะใช้เวลาในการคำนวณฟังก์ชั่น NLMMT ประมาณ 850 วินาที คิดเป็นร้อยละ 24 ของโปรแกรม ซึ่งลดเวลาในการทำงานลงไปได้มาก และในส่วนของฟังก์ชั่น OUTPUTZ, OUTPUTM, OUTPUTN ซึ่งในการคำนวณแบบลำดับจะใช้เวลาในการเขียนประมาณ 2,900 วินาที หรือร้อยละ 10 ของโปรแกรม เมื่อทำงานแบบขนานบนหน่วยประมวลผลกราฟิกจะใช้เวลาประมาณ 1,000 วินาที คิดเป็นร้อยละ 29 ของโปรแกรม เนื่องจากว่าโปรแกรมจำลองสีนามิแบบขนานบนหน่วยประมวลผลกราฟิกมีสัดส่วนของเวลาที่เปลี่ยนได้จากเดิมมากและเวลาทั้งหมดของโปรแกรมอยู่ที่ 3,752 วินาที เพราะฉะนั้นเวลาในการทำงานฟังก์ชั่น OUTPUT จึงมีสัดส่วนของเวลาที่เพิ่มขึ้น สาเหตุเนื่องจากความแตกต่างของคอมไพล์เลอร์ที่ใช้กับโปรแกรม โดยที่โปรแกรมจำลองสีนามิแบบลำดับเขียนโดยใช้ภาษา FORTRAN โดยใช้ compiler gfortran แต่โปรแกรมจำลองสีนามิแบบขนานบนหน่วยประมวลผลกราฟิกจะใช้ compiler CUDA-C (nvcc) ทำให้เวลาที่ใช้ในการเขียน OUTPUT ต่างกัน ส่วนฟังก์ชั่น JNQ_S2C, JNZ, JNQ ซึ่งเป็นฟังก์ชั่นสำหรับการแลกเปลี่ยนข้อมูลระหว่างบริเวณขอบเขตของ Region ใช้เวลาในการคำนวณเพิ่มขึ้นจากโปรแกรมดั้งเดิมมาก อันเนื่องมาจากฟังก์ชั่นเหล่านี้จะมีการทำงานเป็นลำดับแบบที่เกี่ยวเนื่องกัน (data dependency) ในแต่ละรอบของการทำงานของฟังก์ชั่น ทำให้ไม่สามารถปรับปรุงให้ทำงานแบบขนานได้ และการทำงานแบบลำดับบนหน่วยประมวลผลกราฟิกจะทำงานได้ช้ากว่าหน่วยประมวลผลหลัก เนื่องจากแต่ละ Thread ของหน่วยประมวลผลกราฟิกมี Clock speed ที่ช้ากว่า Clock speed ของหน่วยประมวลผลหลัก

ตารางที่ 4 ตารางผลการคำนวณโปรแกรมจำลองสึนามิแบบขนานบนหน่วยประมวลผลกราฟิกที่ได้รับการปรับปรุงบนระบบคลัสเตอร์เพกาซัส

Region	Function	Pegasus				
		Naïve	32 T/B	64 T/B	128 T/B	256 T/B
R1-4	INITIAL	1.05	1.03	1.03	1.05	1.03
R1	MASS	105.59	105.50	105.95	105.95	106.11
R1	OPENBOUNDARY	1.41	1.37	1.51	1.38	1.21
R1	AFDEFORM	0.01	0.01	0.01	0.02	0.01
R1	MOMENT	542.81	438.95	480.91	462.71	347.10
R1	INTERQT	32.22	26.46	23.74	24.22	23.32
R1	JNQ_S2C	169.95	172.07	169.75	149.58	171.69
R2	NLMASS	33.14	25.38	23.86	25.29	23.79
R3	NLMASS	5.35	4.02	3.36	3.98	4.12
R4	NLMASS	9.23	6.98	6.48	6.59	7.46
R2-3	JNZ	214.12	211.89	214.05	213.39	210.72
R2	NLMMT	622.96	377.87	329.11	329.18	303.94
R3	NLMMT	82.96	51.93	35.36	47.71	46.00
R4	NLMMT	154.30	97.26	67.23	88.73	82.80
R2-3	JNQ	259.98	259.43	260.78	248.81	258.85
R2	ETAMAP	16.20	8.76	5.23	4.91	4.81
R3	ETAMAP	2.78	1.44	1.16	1.48	1.41
R4	ETAMAP	4.33	2.88	1.88	2.06	1.62
R2	CHANGE	27.61	1.23	0.96	1.15	0.49
R3	CHANGE	3.75	0.95	0.88	1.01	0.41
R4	CHANGE	7.72	1.16	1.12	1.53	0.55
R1-4	ZMAX	25.71	14.35	11.88	12.47	12.11
R1-4	MNMAX	62.76	34.47	23.92	23.55	24.04
R1-4	OUTPUTZ	323.67	325.18	324.75	326.77	322.59
R1-4	OUTPUTM	365.22	366.81	368.77	372.06	362.05
R1-4	OUTPUTN	356.61	357.93	360.36	363.04	353.24
R1	ETAMAP	37.38	38.07	38.15	37.66	38.60
R1	CHANGE	80.36	0.00	0.00	0.00	0.01

R1-4	OUTPUTMAX	5.03	3.61	3.67	3.63	3.65
Memory copy Host to Device		195.79	195.95	198.79	198.03	193.31
Memory copy Device to Host		2.64	2.78	2.84	2.77	2.83
TOTAL		3752.63	3135.72	3067.49	3060.72	2909.86

ด้วยการปรับปรุงประสิทธิภาพด้วยวิธีปรับการเข้าถึงหน่วยความจำการใช้ Page flipper และจัดลำดับการทำงานของโปรแกรม (Branch divergence) จะเห็นว่าโปรแกรมได้ประสิทธิภาพที่ดีขึ้นจากโปรแกรมที่ไม่ได้ปรับปรุงอยู่พอสมควร โดยที่โปรแกรมที่ยังไม่ได้รับการปรับปรุงจะเลือกใช้ขนาด Threads per block เท่ากับ 32 ซึ่งเป็นค่ามาตรฐานและน้อยสุดที่สามารถใช้กับขนาดของข้อมูลที่ใช้ในการทดลองได้ เมื่อทำการเปรียบเทียบประสิทธิภาพของโปรแกรมกับรูปแบบของ 32 T/B จะเห็นว่าในฟังก์ชั่น NLMMT ที่ Region 2 จากเดิมใช้เวลา 622.96 วินาที ลดเวลาในการทำงานเหลือเพียง 377.87 วินาที โดยฟังก์ชั่น NLMMT ทั้งหมดใช้เวลาประมาณ 527 วินาที คิดเป็นร้อยละ 16 ของโปรแกรม ซึ่งลดจากโปรแกรมที่ยังไม่ได้ปรับปรุงเดิมร้อยละ 8 ของโปรแกรม โดยที่เวลาทั้งหมดของโปรแกรมที่ได้รับการปรับปรุงประสิทธิภาพเมื่อเปรียบเทียบกับโปรแกรมดั้งเดิมพบว่าสามารถลดเวลาที่ใช้ในการทำงานลดได้คิดเป็นร้อยละ 11 ของโปรแกรมที่ยังไม่ได้รับการปรับปรุง

เมื่อทำการทดลองเพิ่มเติมโปรแกรมปรับค่าของ Threads per block ตามค่ามาตรฐานซึ่งเหมาะสมกับการทำงานพบว่า ขนาด Threads per block ที่ดีที่สุดคือ 256 ซึ่งจะใช้เวลาน้อยที่สุดเมื่อเปรียบเทียบกับเวลาที่ใช้ในการทำงานในทุกรูปแบบของ Threads per block ที่ใช้ในการทดลอง โดยสามารถลดเวลาในการทำงานจากฟังก์ชั่นที่ใช้เวลามากที่สุดอย่าง NLMMT จากเดิม 850 วินาที เหลือเพียงประมาณ 420 วินาที ซึ่งลดลงจากเดิมประมาณ 1 เท่าตัว ส่วนอีกฟังก์ชั่นที่มีการลดลงของเวลามากเมื่อเทียบกับโปรแกรมที่ยังไม่ได้ปรับปรุงประสิทธิภาพคือ ฟังก์ชั่น MOMENT(R1) คือ

ฟังก์ชั่นที่คำนวณกฎอนุรักษ์โมเมนตัมของพื้นที่ Region 1 โดยจะใช้เวลา 347 วินาที เมื่อเปรียบเทียบกับโปรแกรมที่ยังไม่ได้ปรับปรุงประสิทธิภาพที่ใช้เวลา 542 วินาที พบว่าเมื่อเปรียบเทียบเป็นสัดส่วนสามารถลดเวลาของฟังก์ชั่นลงไปร้อยละ 35 ของฟังก์ชั่น ซึ่งเมื่อเปรียบเทียบโปรแกรมโดยรวมแล้วสามารถลดเวลาที่ใช้ในการทำงานเหลือเพียง 2,909 วินาที จากเดิม 3,556 วินาที หรือประมาณ 10 นาที

ตารางที่ 5 ตารางผลการคำนวณโปรแกรมจำลองสึนามิแบบขนานบนหน่วยประมวลผลกราฟิกที่ได้รับการปรับปรุงบนเวิร์คสเตชั่นสเปซแลบ

Region	Function	Spacelab				
		Naïve	32 T/B	64 T/B	128 T/B	256 T/B
R1-4	INITIAL	1.59	1.57	1.61	1.69	1.60
R1	MASS	148.19	148.02	148.66	151.20	150.34
R1	OPENBOUNDARY	3.07	3.08	3.26	3.23	3.26
R1	AFDEFORM	0.01	0.01	0.01	0.01	0.01
R1	MOMENT	495.90	493.58	501.69	505.80	507.23
R1	INTERQT	279.44	148.20	89.34	56.81	52.81
R1	JNQ_S2C	78.45	81.89	76.58	75.17	76.63
R2	NLMASS	172.42	85.32	46.28	34.67	30.13
R3	NLMASS	28.88	16.66	13.37	5.58	5.57
R4	NLMASS	44.79	24.88	14.13	14.49	5.93
R2-3	JNZ	118.41	120.27	117.41	120.77	121.42
R2	NLMMT	1725.73	999.56	566.24	400.78	480.02
R3	NLMMT	223.61	129.76	75.49	61.74	64.36
R4	NLMMT	440.47	253.62	145.05	105.17	121.29
R2-3	JNQ	110.53	110.49	111.36	112.53	110.72
R2	ETAMAP	124.55	65.37	35.48	20.42	18.93
R3	ETAMAP	19.53	11.51	10.37	7.36	6.99
R4	ETAMAP	34.58	20.46	12.19	9.50	5.92
R2	CHANGE	207.11	2.26	2.77	2.54	2.62
R3	CHANGE	30.03	1.63	1.88	1.86	1.70
R4	CHANGE	56.82	2.25	1.94	2.16	2.40
R1-4	ZMAX	198.88	106.71	63.77	42.92	37.37

R1-4	MNMAX	392.07	207.22	117.37	73.98	64.32
R1-4	OUTPUTZ	518.45	516.82	523.68	526.19	531.03
R1-4	OUTPUTM	521.66	522.91	531.03	540.03	531.29
R1-4	OUTPUTN	519.32	520.14	529.56	537.37	528.34
R1	ETAMAP	29.34	29.56	29.68	29.66	29.79
R1	CHANGE	68.43	0.00	0.00	0.00	0.00
R1-4	OUTPUTMAX	4.86	4.85	4.97	4.96	4.92
Memory copy Host to Device		208.56	211.69	213.61	214.67	212.29
Memory copy Device to Host		2.74	2.74	2.80	2.84	2.85
TOTAL		6808.38	4843.00	3991.55	3666.10	3712.05

จากตารางที่ 5 ซึ่งเป็นผลการทดลองโปรแกรมจำลองสึนามิแบบขนานบนหน่วยประมวลผลกราฟิกบนเครื่องเวิร์คสเตชั่นสเปซแลบ เมื่อดูที่โปรแกรมที่ยังไม่ได้รับการปรับปรุงประสิทธิภาพจะใช้เวลาประมาณ 6,800 วินาที ซึ่งมากกว่าเวลาที่ใช้บนระบบคลัสเตอร์เพกาซัส เนื่องจาก 2 ปัจจัยคือ ปัจจัยแรกเนื่องจากกราฟิกการ์ดที่ใช้บนเครื่องเวิร์คสเตชั่นสเปซแลบ เป็นกราฟิกการ์ดที่ขนาดเล็ก ถึงแม้ว่าจะเป็น สตาปัตยกรรมที่ใหม่กว่าแต่มีคุณสมบัติที่ด้อยกว่ากราฟิกการ์ดบนระบบเพกาซัส ไม่ว่าจะเป็น Clock speed, CUDA cores, Memory bandwidth ที่น้อยกว่าทั้งหมด ทำให้ประสิทธิภาพในการคำนวณแบบขนานบนหน่วยประมวลกราฟิกนั้นทำงานได้ช้ากว่า อีกปัจจัยหนึ่งที่เห็นได้ชัดจากตารางผลการทดสอบคือ เวลาที่ใช้ในการทำงานฟังก์ชั่น OUTPUT ซึ่งต่างจากการทำงานบนระบบเพกาซัสคลัสเตอร์อยู่ประมาณ 200 วินาทีในทุกฟังก์ชั่น ซึ่งเมื่อรวมทั้งหมดจะได้ประมาณ 500 วินาทีทำให้ส่งผลต่อเวลาโดยรวมของโปรแกรมอยู่พอสมควร เมื่อดูที่ฟังก์ชั่น NLMMT ที่ Region 2 จะพบว่าใช้เวลาในการทำงานประมาณ 1,700 วินาที คิดเป็นร้อยละ 25 ของโปรแกรม เมื่อเปรียบเทียบกับโปรแกรมที่คำนวณแบบลำดับซึ่งใช้เวลาประมาณ 8,500 วินาที ซึ่งลดลงไป 5 เท่า และเมื่อคิดเวลารวมทุก Region ของฟังก์ชั่น NLMMT จะใช้เวลาประมาณ 2,400 วินาที ต่างจากโปรแกรมที่คำนวณแบบลำดับซึ่งใชเวลาประมาณ 11,700 วินาที อยู่ 4.9

เท่า ส่งผลให้โปรแกรมทำงานได้เร็วขึ้นมาก ส่วนฟังก์ชั่นที่ใช้เวลาในการทำงานมากอีกฟังก์ชั่นคือ OUTPUT ซึ่งจากของเดิมจะใช้เวลาประมาณ 1,500 วินาทีต่อ OUTPUT หนึ่งชนิด โดยโปรแกรมคำนวณแบบขนานบนหน่วยประมวลผลกราฟิกจะใช้เวลาเหลือเพียง 500 วินาที เมื่อคิดรวมทั้ง 3 ชนิดของ OUTPUT จะใช้เวลาประมาณ 1,500 วินาที ซึ่งลดจากของเดิมที่ใช้เวลาประมาณ 4,500 วินาที ได้ถึง 3 เท่า ซึ่งถือเป็นส่วนสำคัญอีก 1 ส่วน

ต่อมาเมื่อทำงานปรับปรุงการประสิทธิภาพการทำงานของโปรแกรม พบว่า เวลาที่ใช้ในการคำนวณลดลงอย่างมาก เมื่อทำการเปรียบเทียบกับโปรแกรมที่ปรับปรุงประสิทธิภาพที่ใช้จำนวน Threads per block เท่ากัน คือ 32 จะเห็นว่าเวลาที่ใช้ในการทำงานฟังก์ชั่น NLMMT ลดลงไปอย่างมาก จากเดิมใช้เวลาประมาณ 2,300 วินาที เหลือเพียง 1,400 วินาที หรือเกือบ 1 เท่าตัว โดยเมื่อทำการเปรียบเทียบเวลากับโปรแกรมที่ยังไม่ได้รับการปรับปรุงประสิทธิภาพ สามารถลดเวลาได้ถึงร้อยละ 28 ของโปรแกรมเดิม และเมื่อทำงานปรับขนาดของ Threads per block เพื่อหาค่าที่ดีที่สุดของกราฟิกการ์ดที่เหมาะสมกับปัญญาที่ใช้ในการทดลองพบว่า ขนาดที่ดีที่สุดคือ 128 Threads per block ซึ่งไม่เหมือนกับระบบคลัสเตอร์เพกาซัสที่ใช้เป็น 256 Threads per block โดยเวลาที่ใช้น้อยที่สุดบนเครื่องเวิร์คสเตชั่นสเปซแลบใช้เวลาประมาณ 3,600 วินาที เมื่อทำการวิเคราะห์ในโปรแกรมพบว่าฟังก์ชั่น NLMMT ใช้เวลาลดลงไปอย่างมากเมื่อเพียบกับโปรแกรมที่ยังไม่ได้ปรับปรุงประสิทธิภาพคิดเป็น 4.2 เท่าจากฟังก์ชั่น NLMMT รวมทุก Region และเมื่อทำการเปลี่ยนเทียบประสิทธิโดยรวมทั้งโปรแกรมจากเดิมใช้เวลาประมาณ 6,800 ลดเหลือเพียง 3,600 วินาที ซึ่งคิดเป็นร้อยละ 47 ของโปรแกรมที่ยังไม่ได้ปรับปรุงประสิทธิภาพ

4.1.2.3 การทดลองด้วยโปรแกรมจำลองสึนามิแบบขนานโดยใช้ขนาดข้อมูลที่แตกต่างกัน

เมื่อได้ทำการทดลองเพื่อหาค่า Threads per block ที่ดีที่สุดของแต่ละเครื่องการทดลองแล้ว จึงได้ทำการตั้งค่าที่ดีที่สุดมาทดสอบบนชุดข้อมูลที่มีขนาดที่แตกต่างกัน เพื่อหาค่าความสัมพันธ์ของเวลาบนเครื่องการทดลองที่แต่ต่างกัน โดยใช้ข้อมูลที่มีขนาดต่างกันทั้งหมด 5 ชุด ซึ่งตัวโปรแกรมจะทำการคำนวณในบริเวณพื้นที่ที่เป็นพื้นน้ำ โดยชุดของข้อมูลที่นำมาทดสอบจะมีขนาดแตกต่างกันตามตารางที่ 6

ตารางที่ 6 ข้อมูลที่ใช้ในการทดลองที่มีขนาดพื้นที่แตกต่างกัน

Region Level	Region ID	X	Y	Grid size	Water size
1	1	690	840	579600	179871
2	21	721	841	606361	547045
2	23	721	841	606361	299396
3	211	241	325	78325	67620
3	212	637	469	298753	275475
3	213	397	211	83767	60977
3	231	655	466	305230	90912
3	232	637	466	296842	210574
4	2111	487	325	158275	131692
4	2112	487	421	205027	179123
4	2131	595	601	357595	246310
4	2311	667	457	304819	107779
4	2323	631	493	311083	250013

ในตารางจะแสดงข้อมูล Region ID แบ่งออกตามระดับของข้อมูล 1-4 โดยมีการบอกถึงขนาดตามแกน X และ Y และขนาดของ Grid size คือผลคูณของค่า X และ Y ของ Region นั้น ส่วนค่า Water size เป็นบริเวณพื้นน้ำของ Region ซึ่งจะเป็นจุดที่มีการคำนวณของโปรแกรม โดยข้อมูลทั้ง 5 ชุดที่ใช้ในการทดลองประกอบด้วย 2111, 2112, 2131, 2311 และ 2323 ซึ่งขนาดของ

บริเวณที่เป็นพื้นน้ำจะแตกต่างกันเพื่อสามารถหาสมการความสัมพันธ์ด้านเวลาได้ โดยขนาดของบริเวณที่เป็นพื้นน้ำโดยรวมมีขนาดดังต่อไปนี้

- 2111 – มีขนาดของพื้นน้ำ 926,228 จุด
- 2112 – มีขนาดของพื้นน้ำ 1,442,506 จุด
- 2131 – มีขนาดของพื้นน้ำ 1,034,203 จุด
- 2311 – มีขนาดของพื้นน้ำ 677,958 จุด
- 2323 – มีขนาดของพื้นน้ำ 939,854 จุด

ตารางที่ 7 ตารางผลการคำนวณโปรแกรมจำลองสึนามิแบบขนานบนหน่วยประมวลผลกราฟิกด้วยขนาดของข้อมูลที่แตกต่างกันบนระบบคลัสเตอร์เพกาซัส

Region	Function	Pegasus				
	Region ID	2311	2111	2323	2131	2122
	Water size	677958	926228	939854	1034203	1442506
R1	Initial	1.3	1.03	1.31	1.13	1.47
R1	Compute1	479.37	454.43	476.222	469.192	460.61
R2-4	NLMMT	456.804	432.74	546.283	545.116	755.468
R2-4	Compute2	146.101	141.28	157.741	148.241	176.091
R1-4	Exchange	581.469	642.72	742.977	620.488	902.509
R1-4	Output	1205.686	1041.53	1119.088	1162.13	1498.152
R1-4	Mem copy	201.81	196.14	200.961	198.712	200.302
Total		3275.561	2909.87	3304.871	3145.009	3994.602

ในตารางที่ 7 แสดงข้อมูลของเวลาในการคำนวณโดยแบ่งตามฟังก์ชั่นที่ทำงานใน Region ที่แตกต่างกันที่ใช้ขนาดของข้อมูลที่แตกต่างกันประกอบด้วยฟังก์ชั่น ซึ่งแยกเป็น 4 ส่วน และได้มีการขยายส่วนของ Compute ออกเป็น 3 ส่วนย่อยประกอบด้วย Compute1 คือ ส่วนการคำนวณที่ทำงานบนหน่วยประมวลผลหลัก NLMMT คือ เวลาที่ใช้ในการคำนวณฟังก์ชั่นรวมทั้ง 3 Regions และส่วนสุดท้าย Compute2 คือ เวลาที่ใช้ในการทำงานฟังก์ชั่นอื่นๆบนหน่วยประมวลผลกราฟิก

เมื่อทำการวิเคราะห์เวลาที่ใช้ในการทำงานของโปรแกรมโดยรวม จะเห็นว่ามีแนวโน้มของเวลาเพิ่มขึ้นตามขนาดของพื้นน้ำ แต่จะมีปัจจัยอื่นเข้ามาเกี่ยวข้อง อย่างเช่นที่บริเวณ 2311 ที่มีบริเวณพื้นน้ำจะใช้เวลาในการทำงานมากกว่า 2111 เนื่องจากบริเวณ 2311 มีขนาดของ Grid size รวมเป็น 1,796,010 จุดซึ่งใหญ่กว่าบริเวณ 2111 ที่มีขนาด 1,422,561 จุด ทำให้จะเสียเวลาในส่วนของการเขียน OUTPUT มากกว่า และอีกส่วนหนึ่งที่เป็นปัจจัยที่ส่งผลต่อเวลาในการคำนวณคือ EXCHANGE โดยที่บริเวณ 2323 จะใช้เวลาในการคำนวณมากกว่าบริเวณ 2131 เนื่องจากที่บริเวณ 2323 มีพื้นที่ใน Region 3 ที่มากกว่าทำให้เกินการส่งข้อมูลระหว่างระดับชั้นมากกว่าบริเวณ 2131

ตารางที่ 8 ตารางผลการคำนวณโปรแกรมจำลองสึนามิแบบขนานบนหน่วยประมวลผลกราฟิกด้วยขนาดของข้อมูลที่แตกต่างกันบนเวิร์คสเตชั่นสเปซแลบ

Region	Function	Spacelab				
	Region ID	2311	2111	2323	2131	2122
	Water size	677958	926228	939854	1034203	1442506
R1	Initial	2.116	1.692	2.09	1.783	2.354
R1	Compute1	649.392	660.236	656.37	656.061	650.049
R2-4	NLMMT	576.739	567.696	668.691	856.111	1182.826
R2-4	Compute2	328.912	295.391	347.786	291.828	329.884
R1-4	Exchange	338.643	315.025	434.612	365.296	584.553
R1-4	Output	1543.034	1608.548	1486.523	1779.127	2219.118
R1-4	Mem copy	212.935	217.511	214.652	214.489	215.142
Total		3651.771	3666.099	3810.724	4164.695	5183.926

ในตารางที่ 8 แสดงข้อมูลของเวลาในการคำนวณโดยแบ่งตามฟังก์ชั่นที่ทำงานใน Region ที่แตกต่างกันที่ใช้ขนาดของข้อมูลที่แตกต่างกันประกอบด้วยฟังก์ชั่น ซึ่งแยกเป็น 4 ส่วนเช่นเดียวกับตารางที่ 7 แต่เป็นการทดลองบนเครื่องเวิร์คสเตชั่นสเปซแลบ ซึ่งที่ได้จากการทำงานของโปรแกรมจะมีความใกล้เกี่ยวกับทั้งสองเครื่องการทดลอง คือเวลามีการแปรผันตามปัจจัยอื่นนอกจากขนาดของบริเวณพื้นที่น้ำ เช่นที่บริเวณ 2323 จะใช้เวลาในการ EXCHANGE มากกว่าบริเวณ เนื่องจากที่บริเวณ 2323 มีพื้นที่ใน Region 3 ที่มากกว่าทำให้เกินการส่งข้อมูลระหว่างระดับชั้นมากกว่าบริเวณ 2131 แต่สิ่ง

ที่แตกต่างจากระบบคลัสเตอร์เพกาซัสคือ เวลาที่ใช้ในการเขียนข้อมูล OUTPUT ซึ่งจะไม่เป็นไปตามขนาดของ Grid size ทั้งนี้เนื่องจากว่าที่เครื่องเวิร์คสเตชั่นสเปซแลบในฮาร์ดดิสแบบ SATA-II ซึ่งไม่เสถียรเท่ากับฮาร์ดดิสแบบ SAS ทำให้เวลาที่ใช้อาจผิดไปจากที่คาดการณ์ได้รวมไปถึงอาจที่การคัดลอกข้อมูลอื่นๆขณะทดลองอยู่ ทำให้เวลาที่ใช้ในการคัดลอกข้อมูลของการทดลองใช้เวลามากขึ้นด้วย

4.1.2.4 การทดลองด้วยโปรแกรมจำลองสึนามิแบบขนานโดยปรับค่าปัจจัยต่างๆ เพื่อเพิ่มความเร็วในการทำงาน

เมื่อเราทำการปรับปรุงชุดคำสั่งให้สามารถทำงานได้มีประสิทธิภาพที่ดีขึ้นแล้ว แต่ยังต้องการความเร็วที่เพิ่มขึ้นอีกในการทำงานของโปรแกรม ดังนั่นจึงต้องทำการทดลองปรับปรุงค่าปัจจัยต่างๆที่ส่งผลต่อเวลาในการคำนวณโดยที่ไม่ส่งผลกระทบต่อความแม่นยำของผลลัพธ์ของโปรแกรม

สำหรับการปรับค่าปัจจัยที่ไม่ส่งผลต่อความแม่นยำของโปรแกรมซึ่งเมื่อวิเคราะห์จากการทำงานของโปรแกรมแล้ว จะพบว่ามีบางฟังก์ชั่นที่ไม่จำเป็นต่อการนำผลลัพธ์ของการคำนวณมาใช้ในระบบเตือนภัย ซึ่งประกอบด้วยฟังก์ชั่นดังต่อไปนี้

- ZMAX 1-4 – เป็นฟังก์ชั่นสำหรับการคำนวณหาค่าสูงสุดของระดับน้ำในแต่ละจุดในขอบเขตของการทดลอง
- MNMAX 1-4 – เป็นฟังก์ชั่นสำหรับการคำนวณหาแรงในแนวแกน X และ Y ที่เป็นค่าสูงสุดในแต่ละจุดในขอบเขตของการทดลอง
- OUTM 1-4 – เป็นฟังก์ชั่นสำหรับการเขียนผลลัพธ์ในการคำนวณของแรงในแนวแกน X
- OUTN 1-4 - เป็นฟังก์ชั่นสำหรับการเขียนผลลัพธ์ในการคำนวณของแรงในแนวแกน Y
- OUTZMAX 1-4, OUTMMAX 1-4, OUTNMAX 1-4, OUTZMNMAX 1-4, OUTUVMAX 1-4 – ฟังก์ชั่นเหล่านี้เป็นการ

เขียนผลลัพธ์ในการคำนวณหาค่ามากสุดในแต่ละจุดในขอบเขตของการทดลอง

ซึ่งในระบบที่ต้องการนำผลลัพธ์ไปใช้ในเวลาที่จำกัด อาจจะสามารถตัดการทำงานของฟังก์ชั่นพวกนี้ออกไปได้ (Function decrease) เพื่อลดภาระในการคำนวณส่วนที่ไม่จำเป็นต่อการนำไปใช้ในระบบเตือนภัย ซึ่งเมื่อทำการตัดฟังก์ชั่นเหล่านี้ออกไป เวลาที่ใช้ในการคำนวณจะเป็นไปตาม ตารางที่ 9

ในตารางที่ 9 จะแสดงเวลาในการทำงานแยกแต่ละฟังก์ชั่น โดยจะเลือกใช้ขนาด Thread per block ที่ได้ประสิทธิภาพในการคำนวณดีสุดของแต่ละเครื่องการทดลอง ซึ่งเมื่อทำการตัดฟังก์ชั่นที่ไม่จำเป็นออกออกไปจะเห็นว่าเวลาที่ใช้ในการทำงานลดลงอย่างมากบนทั้งสองเครื่องการทดลอง โดยในระบบบคลัสเตอร์เพกาซัสจะใช้เวลาเหลือเพียง 2,154 วินาที จากเดิมใช้เวลา 2,909 วินาที ลดลงไป 750 วินาที หรือประมาณ 12 นาที และคิดเป็นร้อยละ 25 ของโปรแกรมที่ยังไม่ได้ตัดฟังก์ชั่นที่ไม่จำเป็นออกซึ่งเป็นค่าที่สูง ส่วนในเครื่องเวิร์คสเตชั่นสเปซแลบจะใช้เวลาเหลือเพียง 2,466 วินาที จากเดิมใช้เวลา 3,666 วินาที ลดลงไป 1,200 วินาที หรือ 20 นาที โดยคิดเป็นร้อยละ 32 ของโปรแกรมที่ยังไม่ได้ตัดฟังก์ชั่นที่ไม่จำเป็นออก ซึ่งสูงกว่าบนระบบคลัสเตอร์เพกาซัส อันเนื่องมาจากเวลาที่ใช้ในการทำงานฟังก์ชั่น OUTPUT บนเวิร์คสเตชั่นสเปซแลบนั้นใช้เวลาสูงกว่า เมื่อทำการตัดในส่วนนี้ออกส่งผลให้ประสิทธิภาพโดยรวมของโปรแกรมดีขึ้นมาก

ตารางที่ 9 ตารางผลการคำนวณโปรแกรมจำลองสึนามิแบบลดฟังก์ชั่นที่ไม่จำเป็น

Region	Function	Pegasus		Spacelab	
		256 T/B	% pegasus	128 T/B	% spacelab
R1-4	INITIAL	1.03	0.05	1.69	0.07
R1	MASS	106.11	4.92	151.20	6.13
R1	OPENBOUNDARY	1.21	0.06	3.23	0.13
R1	AFDEFORM	0.01	0.00	0.01	0.00
R1	MOMENT	347.10	16.11	505.80	20.50

R1	INTERQT	23.32	1.08	56.81	2.30
R1	JNQ_S2C	171.69	7.97	75.17	3.05
R2	NLMASS	23.79	1.10	34.67	1.41
R3	NLMASS	4.12	0.19	5.58	0.23
R4	NLMASS	7.46	0.35	14.49	0.59
R2-3	JNZ	210.72	9.78	120.77	4.90
R2	NLMMT	303.94	14.11	400.78	16.25
R3	NLMMT	46.00	2.13	61.74	2.50
R4	NLMMT	82.80	3.84	105.17	4.26
R2-3	JNQ	258.85	12.01	112.53	4.56
R2	ETAMAP	4.81	0.22	20.42	0.83
R3	ETAMAP	1.41	0.07	7.36	0.30
R4	ETAMAP	1.62	0.08	9.50	0.38
R2	CHANGE	0.49	0.02	2.54	0.10
R3	CHANGE	0.41	0.02	1.86	0.08
R4	CHANGE	0.55	0.03	2.16	0.09
R1-4	OUTPUTZ	322.59	14.97	526.19	21.33
R1	ETAMAP	38.60	1.79	29.66	1.20
R1	CHANGE	0.01	0.00	0.00	0.00
Memory copy Host to Device		193.31	8.97	214.67	8.70
Memory copy Device to Host		2.83	0.13	2.84	0.12
TOTAL		2154.77	100.00	2466.84	100.00

นอกจากการปรับลดฟังก์ชั่นที่ไม่จำเป็นออกจะเป็นการลดเวลาในการทำงานลงได้มากแล้วอีกวิธีหนึ่งที่จะช่วยในการลดเวลาในการทำงานได้พอสมควร คือ การปรับลดรอบเวลาของการเขียนผลลัพธ์ (decrease write output steps) เนื่องจากโปรแกรมแต่เดิมนั้น มีรอบของการเขียนผลลัพธ์ทุกๆ 60 วินาทีของการคำนวณ ในการจำลองสึนามิเวลาในการคำนวณล่วงหน้าคือ 10 ชม. ดังนั้นจะได้ผลลัพธ์ทั้งหมด 600 ชุด ซึ่งเป็นความละเอียดสูง เหมาะแก่การนำผลลัพธ์ที่ได้นำไปสร้างซิมมูเลชั่นหรือทำการ

วิเคราะห์โดยละเอียด แต่หากต้องการนำผลลัพธ์ที่ได้ไปใช้ในระบบเตือนภัย ซึ่งไม่จำเป็นต้องมีความละเอียดสูงมาก ใช้เพียงข้อมูลในบางช่วงเวลามาวิเคราะห์และสามารถประเมินสถานการณ์ได้อย่างรวดเร็ว ซึ่งได้ทำการทดลองปรับลดเวลาในการเขียนผลลัพธ์ตามตารางที่ 10

ในตารางจะแสดงการเวลาที่ใช้ในการเขียนผลลัพธ์ของโปรแกรมบนทั้งสองเครื่องการทดลอง ซึ่งจะแสดงค่าความถี่ที่ใช้ในการเขียนผลลัพธ์ประกอบด้วย 60, 120, 240, 300, 600, 1200 และ 1800 วินาที และแสดงถึงเปอร์เซ็นต์ในการเขียนผลลัพธ์ในแต่ละค่าความถี่ จากข้อมูลในตารางจะเห็นว่ายิ่งปรับลดเวลาในการเขียนผลลัพธ์ลงมาก เวลาที่ใช้ก็จะลดลงอย่างมากเช่นกัน โดยเมื่อปรับลดความถี่รอบการเขียนผลลัพธ์เป็น 300 วินาที หรือ 5 นาทีต่อรอบ จะได้ผลการทดลอง 120 ชุดใน 10 ชม.ของการจำลอง ซึ่งถือว่ายังมีความละเอียดสูงในระดับหนึ่งและยังสามารถนำไปวิเคราะห์ผลอย่างละเอียดได้ โดยเวลาที่ใช้เขียนผลลัพธ์จะเหลือเพียง 234 วินาที จากเดิม 1,590 วินาที ซึ่งเมื่อคิดเป็นร้อยละจะสามารถลดได้สูงถึงร้อยละ 85 และคิดเป็นร้อยละ 9.8 ของโปรแกรมโดยรวม บนเครื่องเวิร์คสเตชั่นสเปซแลบ ส่วนบนระบบคลัสเตอร์เพกาซัสจากเดิมใช้เวลาในการเขียนผลลัพธ์รวม 1,037 วินาที เหลือเพียง 209 วินาที ซึ่งคิดเป็นร้อยละ 10 ของโปรแกรมโดยรวม และเมื่อต้องการปรับลดรอบความถี่ของการเขียนผลลัพธ์ลงเป็น 1,200 วินาที (20 นาที) หรือ 1,800 วินาที (30 วินาที) ต่อรอบ จะเห็นว่าอัตราร้อยละของการเขียนผลลัพธ์ลดลงไปเหลือเพียง 1 – 2 บนทั้งสองเครื่องการทดสอบ ซึ่งคิดเป็นสัดส่วนที่น้อยมากกับเวลารวมทั้งหมด ซึ่งจากเดิมคิดเป็นร้อยละ 42 บนเครื่องเวิร์คสเตชั่นสเปซแลบ และคิดเป็นร้อยละ 35 บนระบบคลัสเตอร์เพกาซัส

ตารางที่ 10 ตารางการคำนวณโปรแกรมจำลองสึนามิแบบปรับลดรอบการเขียนผลลัพธ์

Time step		60	120	240	300	600	1200	1800
Spacelab	output Z,M,N	1590.66	1169.02	587.23	234.63	116.68	58.74	39.51

	% program	42.85	35.04	21.40	9.80	5.12	2.64	1.79
Pegasus	output Z,M,N	1037.88	524.50	262.92	209.65	105.15	52.94	35.19
	% program	35.67	21.86	11.95	10.09	5.24	2.75	1.84

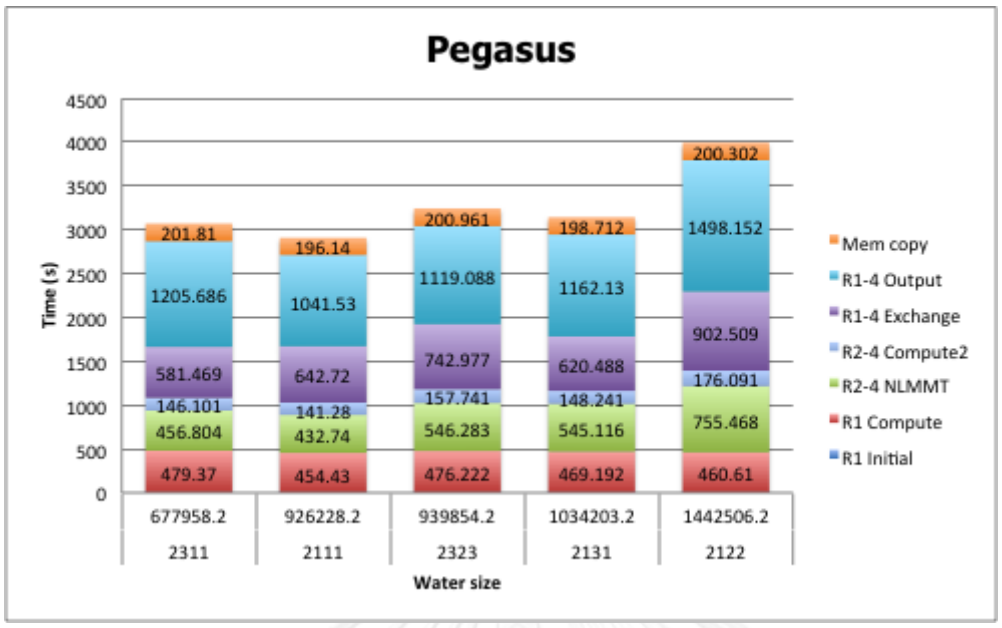
4.1.3 วิเคราะห์งานวิจัย

จากผลการทดลองในการคำนวณแบบขนานด้วยหน่วยประมวลผลกราฟิกจะเห็นได้ว่าเวลาที่ใช้ในการคำนวณลดลงไปอย่างมากทั้งบนระบบคลัสเตอร์เพกาซัส และเครื่องเวิร์คเสตชั่นสเปซแลบเมื่อนำมาเปรียบเทียบกับโปรแกรมจำลองสึนามิแบบลำดับ ซึ่งเมื่อนำเวลาที่ใช้ในการจำนวนมาคำนวณหาค่าความเร็วที่เพิ่มขึ้นจะได้ผลดังตารางที่ 11 และตารางที่ 12 จะเห็นว่าทั้งสองระบบสามารถลดเวลาในการทำงานได้เป็นอย่างมากเมื่อเปรียบเทียบกับการทำงานแบบลำดับ โดยได้ค่า speedup 7.33 และ 2.88 เท่า สำหรับโปรแกรมจำลองสึนามิที่ทำงานแบบขนานในการทำงานบนระบบคลัสเตอร์เพกาซัส และเวิร์คสเตชั่นสเปซแลบตามลำดับ และเมื่อทำการปรับปรุงประสิทธิภาพของโปรแกรมด้วยเทคนิคต่างๆพบว่าโปรแกรมสามารถทำงานได้ดีขึ้น โดยที่บนระบบคลัสเตอร์เพกาซัสสามารถลดเวลาในการทำงานลงได้ประมาณ 850 วินาที หรือประมาณ 14 นาที โดยได้รับ speedup เพิ่มจากเดิมเป็น 9.45 เท่า ซึ่งเช่นเดียวกันบนเวิร์คสเตชั่นเปสแลปเมื่อได้นำโปรแกรมที่ได้รับการปรับปรุงแล้วมาคำนวณแล้ว ผลปรากฏว่าได้รับประสิทธิภาพที่ดีขึ้นมากโดยสามารถลดเวลาในการคำนวณได้ประมาณ 3000 นาที หรือประมาณ 50 นาที และได้รับ speedup เพิ่มขึ้นจากเดิมเป็น 5.43 สาเหตุหนึ่งที่ได้รับประสิทธิภาพในการคำนวณที่ดีขึ้น เนื่องจากว่าการปรับขนาดของเธรดในแต่ละเธรดบล๊อคนั้นให้ได้รับค่า Occupancy ที่เหมาะสม ซึ่งการปรับขนาดของเธรดบล๊อคให้ใหญ่ขึ้นเป็น 2 เท่า หรือ 4 เท่า นั้นประสิทธิภาพของโปรแกรมจะไม่ได้เพิ่มตามจำนวนเท่าที่เพิ่มขึ้นเนื่องจากในการทำงานจริงของหน่วยประมวลผลกราฟิกจะสามารถทำงานได้ครั้งละ 32 เธรด ซึ่งเธรดที่เหลือจะทำการรอให้ตัวจัดการเธรดนำเข้าไปคำนวณในรอบถัดไป และถ้าจัดขนาดของเธรดบล๊อคให้เหมาะสม จะสามารถลดเวลาในการทำงานของตัวจัดการเธรด นอกจากนี้ในการ

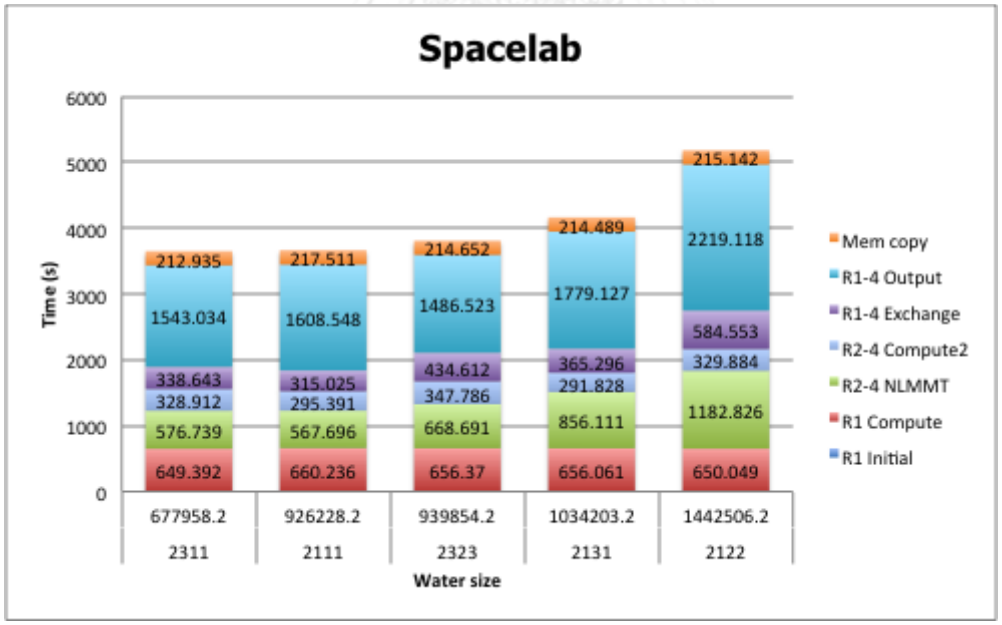
ทำงานของโปรแกรมยังมีส่วนของการทำงานที่ไม่ขึ้นอยู่กับขนาดของเธรดบล๊อคอยู่ ด้วยทำงานเวลาของการทำงานโดยรวมไม่ลดลงมากเท่าที่ควร และอีกสาเหตุหนึ่งคือ เนื่องจากเวิร์คสเตชั่นสเปซแลบมีหน่วยประมวลผลกราฟิกที่มีความเร็วของสัญญาณนาฬิกา (GPU Clock) ที่น้อย ดังนั้นในการจัดสรรขนาดของเธรดบล๊อคให้เหมาะสมกับปริมาณงานเพื่อให้แต่ละคอร์สามารถทำงานได้อย่างเต็มที่ จึงเป็นสิ่งที่สำคัญเช่นกัน

ในส่วนของการทดลองด้วยโปรแกรมจำลองสึนามิแบบขนานโดยปรับขนาดของพื้นที่ของข้อมูลที่ใช้ในการทดลอง เราสามารถนำผลของการทดลองมาสร้างกราฟได้ดังรูปที่ 16 และรูปที่ 17

โดยรูปที่ 16 จะเป็นกราฟที่แสดงเวลาที่ใช้ในการทำงานบนระบบคลัสเตอร์เพกาซัส โดยข้อมูลในแกน Y คือเวลาที่ใช้ในการทำงาน และแกน X จะแบ่งออกตามขนาดของข้อมูลที่ใช้ในการคำนวณ ซึ่งเรียงตามขนาดของบริเวณพื้นที่น้ำของแต่ละบริเวณที่นำมาทดลอง เมื่อดูที่เวลาในการทำงานจะเห็นได้ชัดเจนในส่วนของฟังก์ชั่น NLMMT ที่ขึ้นอยู่กับขนาดของบริเวณพื้นน้ำคือ ถ้าบริเวณพื้นน้ำมีขนาดมากก็จะใช้เวลาในการคำนวณมาก และในส่วน EXCHANGE ที่บริเวณ 2323 ที่มีเวลาในการทำงานมากกว่า 2131 ก็เนื่องมาจากที่ Region 3 ที่ขนาดใหญ่กว่า ทำให้เกิดการส่งต่อข้อมูลที่มากเพราะต้องมีการแลกเปลี่ยนข้อมูลกับ Region 2 และ Region 4



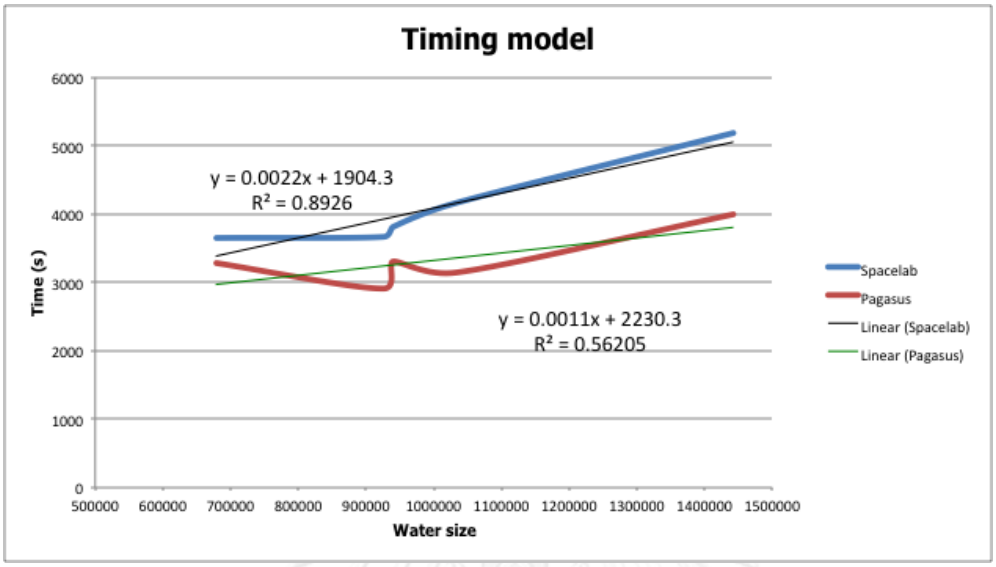
รูปที่ 16 กราฟแสดงความสัมพันธ์ระหว่างเวลากับขนาดของพื้นที่น้ำบนระบบคลัสเตอร์เพกาซัส



รูปที่ 17 กราฟแสดงความสัมพันธ์ระหว่างเวลากับขนาดของพื้นที่น้ำบนเวิร์คสเตชั่นสเปซแลบ

ในรูปที่ 17 จะเป็นกราฟที่แสดงเวลาที่ใช้ในการทำงานบนเวิร์คสเตชั่นสเปซแลบ ซึ่งมีแนวโน้มของเวลาใกล้เคียงกับเวลาที่ใช้บนระบบคลัสเตอร์เพกาซัสคือฟังก์ชั่น

NLMMT ที่ใช้เวลาในการทำงานมากขึ้นเมื่อบริเวณของพื้นน้ำมากขึ้น และมีส่วนของ EXCHANGE ที่บริเวณ 2323 ที่ใช้เวลาในการคำนวณมากกว่าโซน 2131



รูปที่ 18 กราฟแสดงความสัมพันธ์ระหว่างเวลากับความถี่รอบการเขียนผลลัพธ์บนเวิร์คสเตชั่นสเปซแลบ

เมื่อนำเวลาที่ใช้ในการคำนวณของทั้ง 2 เครื่องการทดสอบนำมาสร้างกราฟเพื่อหาสมการความสัมพันธ์ของเวลาจะได้ออกมาดังรูปที่ 18 โดยข้อมูลในแกน X คือขนาดของบริเวณพื้นน้ำ ส่วนแกน Y จะแสดงเวลาในหน่วยของวินาที โดยในกราฟจะประกอบด้วยเส้น 4 เส้น คือ เส้นสีน้ำเงินเป็นเวลาที่ใช้ในการทำงานบนเครื่องเวิร์คสเตชั่นสเปซแลบ เส้นสีดำคือ Trendline แบบสมการเชิงเส้นของเวลาในการทำงานบนเครื่องเวิร์คสเตชั่นสเปซแลบ เส้นสีแดงเป็นเวลาที่ใช้ในการทำงานบนระบบคลัสเตอร์เพกาซัส และเส้นที่เขียวคือ Trendline แบบสมการเชิงเส้นของเวลาในการทำงานบนระบบคลัสเตอร์เพกาซัส โดยที่ความสัมพันธ์ระหว่างเวลากับขนาดของพื้นน้ำบนระบบคลัสเตอร์เพกาซัสจะอยู่ในรูปสมการ

$$y = 0.0011x + 2230.3$$

โดยที่ y เป็นค่าของขนาดบริเวณพื้นน้ำ

x เป็นค่าเวลาที่ใช้ในการทำงานของโปรแกรม

ส่วนความสัมพันธ์ระหว่างเวลากับความถี่รอบการเขียนผลลัพธ์บนเวิร์คสเตชั่นสเปซแลบจะอยู่ในรูปสมการ

$$y = 0.0022x + 1904.3$$

โดยที่ y เป็นค่าของขนาดบริเวณพื้นน้ำ

x เป็นค่าเวลาที่ใช้ในการทำงานของโปรแกรม

สำหรับสมการเวลาบนระบบคลัสเตอร์เพกาซัสจะมีค่า R^2 เท่ากับ 0.56 ซึ่งเป็นค่าที่อยู่ในระดับที่พอรับได้ ส่วนบนเวิร์คสเตชั่นเปซแลบ จะมีค่า R^2 เท่ากับ 0.89 ซึ่งเป็นค่าที่สูงสามารถประเมินเวลาในการคำนวณได้ค่อนข้างแม่นยำ

ในส่วนของการทดลองด้วยโปรแกรมจำลองสึนามิแบบขนานโดยปรับค่าปัจจัยต่างๆเพื่อเพิ่มความเร็วในการทำงานนั้นจะทำการวิเคราะห์แต่ละปัจจัยดังต่อไปนี้

1. การปรับลดเวลาในการคำนวณโปรแกรมจำลองสึนามิด้วยการตัดฟังก์ชั่นที่ไม่จำเป็นออก เมื่อทำการวิเคราะห์โปรแกรมโดยรวมแล้วจะเห็นว่าในโปรแกรมมีส่วนการคำนวณและเขียนผลลัพธ์ที่ไม่จำเป็นต่อระบบเตือนภัย ซึ่งใช้เวลาในการทำงานประมาณร้อยละ 25 และ 32 บนระบบคลัสเตอร์เพกาซัสกับเวิร์คสเตชั่นสเปซแลบตามลำดับ ซึ่งมีค่ามาก หากต้องการความเร็วในการคำนวณเพื่อนำไปใช้ในระบบเตือนภัยเมื่อตัดส่วนเหล่านี้ออกไปจะลดเวลาในการคำนวณได้ประมาณ 800 วินาที (13 นาที) และ 1,200 วินาที (20 นาที) บนระบบคลัสเตอร์เพกาซัสกับเวิร์คสเตชั่นสเปซแลบตามลำดับ ซึ่งทำให้สัดส่วนของการทำงานของโปรแกรมโดยรวมเหลือเพียงร้อยละ 14 และ 21 บนระบบคลัสเตอร์เพกาซัสกับเวิร์คสเตชั่นสเปซแลบตามลำดับ
2. การคำนวณโปรแกรมจำลองสึนามิแบบปรับลดรอบการเขียนผลลัพธ์ สำหรับโปรแกรมจำลองสึนามิเดิมนั้น มีรอบการเขียนที่ถี่มาก คือเขียนผลลัพธ์ทุกๆ 60

วินาที หรือ 1 นาที ซึ่งเป็นประโยชน์ต่อการใช้ซิมมูเลชั่น แต่ไม่เป็นประโยชน์ต่อการใช้ในระบบเตือนภัย เนื่องจากการเขียนผลลัพธ์ใช้เวลามาก ดังนั้นเมื่อทำการลดเวลาที่ใช้ในการเขียนให้ห่างขึ้นจะเห็นว่าโปรแกรมโดยรวมจะทำงานได้ไวขึ้น เนื่องจากเวลาที่ใช้ในการเขียนผลลัพธ์นั้นสูงคิดเป็นร้อยละ 42 บนเครื่องเวิร์คสเตชั่นสเปซแลบ และร้อยละ 35 บนระบบคลัสเตอร์เพกาซัส เมื่อนำเอาเวลาที่ใช้ในการเขียนผลลัพธ์ที่มีความถี่ต่อรอบต่างกันมาสร้างกราฟเพื่อหาค่าความสัมพันธ์ จะได้กราฟซึ่งอยู่ในรูปของสมการเลขยกกำลัง ตามรูปที่ 19 และรูปที่ 20 โดยที่ความสัมพันธ์ระหว่างเวลากับความถี่รอบการเขียนผลลัพธ์บนระบบคลัสเตอร์เพกาซัสจะอยู่ในรูปสมการ

$$y = 64552x^{-1.005}$$

โดยที่ y เป็นค่าของความถี่ในการเขียนผลลัพธ์

x เป็นค่าเวลาที่ต้องการใช้ในการเขียนผลลัพธ์

ส่วนความสัมพันธ์ระหว่างเวลากับความถี่รอบการเขียนผลลัพธ์บนเวิร์คสเตชั่นสเปซแลบจะอยู่ในรูปสมการ

$$y = 34554x^{-0.828}$$

โดยที่ y เป็นค่าของความถี่ในการเขียนผลลัพธ์

x เป็นค่าเวลาที่ต้องการใช้ในการเขียนผลลัพธ์

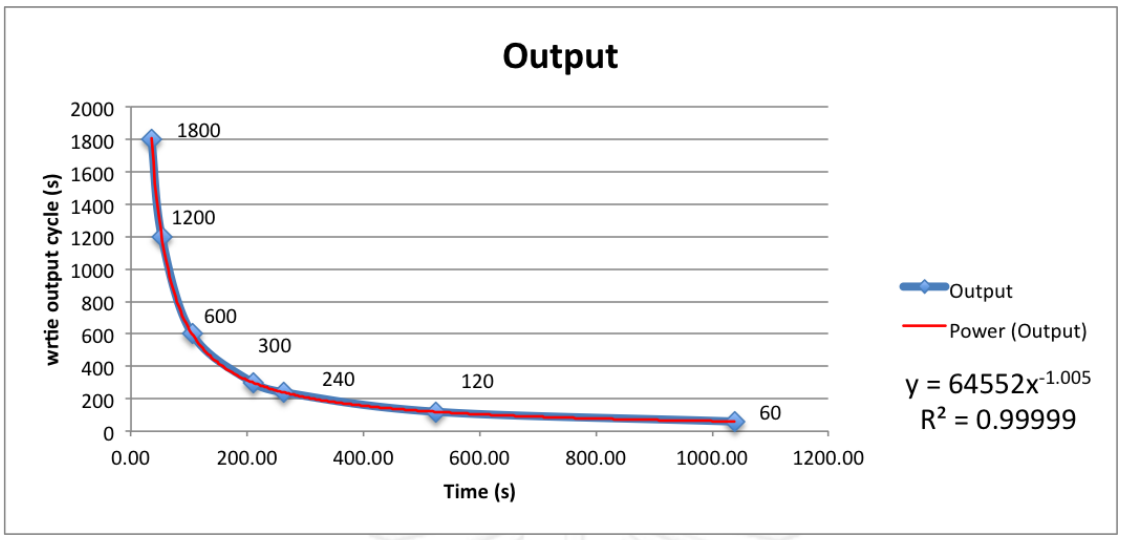
จากการทดลองด้วยการปรับค่าปัจจัยต่างๆนั้นพบว่าสามารถช่วยลดเวลาในการทำงานลงได้อย่างมากโดยทั้ง 2 วิธีสามารถนำมาใช้ร่วมกันได้โดยไม่ส่งผลกระทบต่อโปรแกรมด้านความแม่นยำ และยังสามารถสร้างสมการของเวลาในการเขียนผลลัพธ์ได้เช่นกัน และอีกหนึ่งผลลัพธ์ที่ได้จากการปรับค่าปัจจัยต่างๆนั้นคือ ขนาดของผลลัพธ์ซึ่งในโปรแกรมเดิมจะมีขนาดประมาณ 24 GB ต่อ 1 ชุดการทดลอง เมื่อลดปัจจัยที่มีผลต่อปริมาณผลลัพธ์แล้ว จะทำให้ขนาดของผลลัพธ์นั้นลดลงตามไปด้วยซึ่งทำให้สามารถคัดลอกและส่งข้อมูลออกไปใช้งานได้รวดเร็วยิ่งขึ้น

ตารางที่ 11 ตารางเปรียบเทียบเวลาระหว่างการคำนวณแบบลำดับกับการคำนวณแบบขนานบนระบบคลัสเตอร์เพกาซัส

Section	Pegasus				
	Sequential	CUDA (naive)	% naive	CUDA (opt)	% opt
Compute + transfer time	24593.84	2702.11	72.01	1868.33	64.21
Output time	2895.09	1050.53	27.99	1041.53	35.79
Total time	27488.93	3752.63	100.00	2909.86	100.00
Speedup	-	7.32	-	9.44	-

ตารางที่ 12 ตารางเปรียบเทียบเวลาระหว่างการคำนวณแบบลำดับกับการคำนวณแบบขนานบนเวิร์คสเตชั่นสเปซแลบ

Section	Spacelab				
	Sequential	CUDA (naive)	% naive	CUDA (opt)	% opt
Compute + transfer time	14687.83	5244.09	77.02	2057.55	56.12
Output time	4887.16	1564.29	22.98	1608.55	43.88
Total time	19574.99	6808.38	100.00	3666.10	100.00
Speedup	-	2.88	-	5.34	-

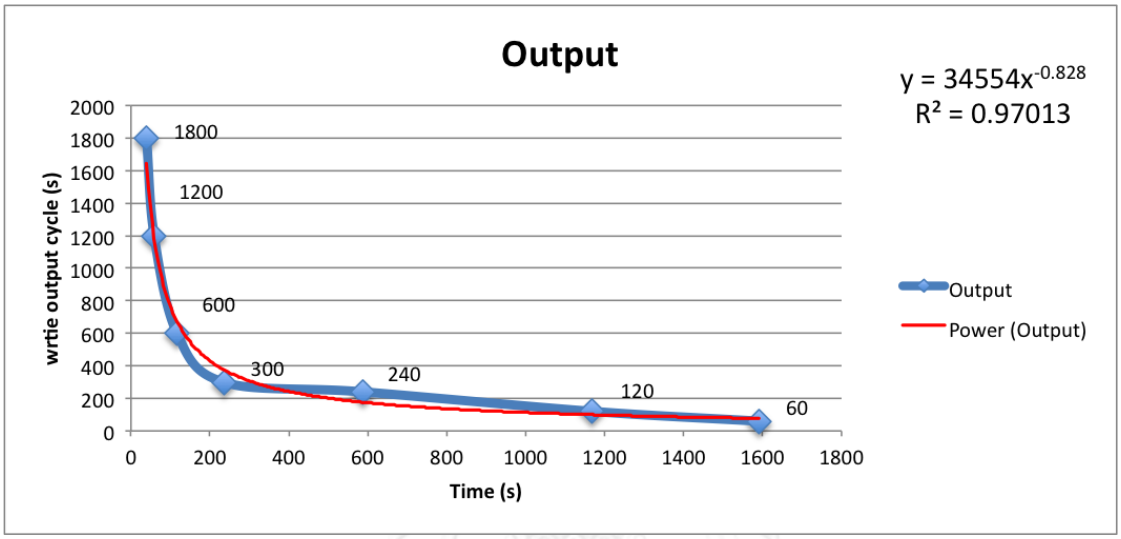


รูปที่ 19 กราฟแสดงความสัมพันธ์ระหว่างเวลากับความถี่รอบการเขียนผลลัพธ์บนระบบคลัสเตอร์เพกาซัส

จากรูปที่ 19 แสดงถึงกราฟความสัมพันธ์ระหว่างเวลากับความถี่รอบการเขียนผลลัพธ์บนระบบคลัสเตอร์เพกาซัส โดยในแนวแกน X จะเป็นค่าเวลาที่ใช้ในการเขียนผลลัพธ์ ส่วนในแนวแกน Y เป็นความถี่รอบการเขียนผลลัพธ์ ซึ่งเมื่อแทนค่าของเวลาที่ต้องการใช้ในการเขียนผลลัพธ์จะสามารถคำนวณหารอบของการเขียนผลลัพธ์ออกมาได้ โดยที่กราฟแสดงความสัมพันธ์ระหว่างเวลากับความถี่รอบการเขียนผลลัพธ์บนระบบคลัสเตอร์เพกาซัส จะมีค่า R^2 คือค่าความคลาดเคลื่อนของผลลัพธ์เป็น 0.99 ซึ่งมีความคลาดเคลื่อนจากความเป็นจริงต่ำมาก เมื่อทดลองแทน X เป็น 1037.88 วินาที จะได้ค่า Y ออกมาเท่ากับ 60.07 ซึ่งมีความใกล้เคียงกับความเป็นจริงมาก

ส่วนในรูปที่ 20 แสดงถึงกราฟแสดงความสัมพันธ์ระหว่างเวลากับความถี่รอบการเขียนผลลัพธ์บนเวิร์คสเตชั่นสเปซแลบ โดยมีค่า R^2 เท่ากับ 0.97 ซึ่งถือว่าต่ำมากเช่นกัน เมื่อทดลองแทน X เป็น 1590.66 วินาที จะได้ค่า Y ออกมาเท่ากับ 77.19 ซึ่งใกล้เคียงกับค่าจริงคือ 60

ดังนั้นหากต้องการระบุเวลาที่ใช้ในการเขียนผลลัพธ์ สามารถทำการคำนวณหาค่าจากสมการเพื่อหาความถี่ที่ต้องใช้ในโปรแกรม เมื่อนำเวลาที่ระบุไปบวกกับส่วนของเวลาที่ใช้ในการคำนวณทั้งหมดก็จะได้เวลาของโปรแกรมโดยรวมออกมา



รูปที่ 20 กราฟแสดงความสัมพันธ์ระหว่างเวลากับความถี่รอบการเขียนผลลัพธ์บนเวิร์คสเตชั่นสเปซแลบ

บทที่ 5
สรุปผลงานวิจัย

ในบทนี้จะกล่าวถึงผลสรุปของงานวิจัย ปัญหาที่พบ และข้อเสนอแนะที่จะเป็นแนวทางในการพัฒนาโปรแกรมจำลองสึนามิแบบขนานผลหน่วยประมวลผลกราฟิกต่อไป

5.1 บทสรุปงานวิจัย

วิทยานิพนธ์ฉบับนี้ได้นำเสนอโปรแกรมจำลองสึนามิแบบขนานบนหน่วยประมวลผลกราฟิก โดยพัฒนาโปรแกรมจำลองสึนามิจากเดิมที่มีการคำนวณแบบลำดับด้วยภาษา FORTRAN เป็นภาษา CUDA C นอกการนั้นแล้วยังทำการปรับปรุงประสิทธิการทำงานของโปรแกรมให้สามารถทำงานได้สอดคล้องกันเครื่อง หรือระบบที่นำมาใช้ในการคำนวณ เมื่อได้ทำการพัฒนาและปรับปรุงประสิทธิภาพของโปรแกรมโดยรวมแล้ว จึงทำการหาปัจจัยอื่นๆมีผลต่อเวลาที่ใช้ในการคำนวณของโปรแกรมที่สามารถปรับลดหรือแก้ไขได้โดยไม่ส่งผลต่อความแม่นยำของผลลัพธ์ของโปรแกรม และทำการวัดเวลาในการทำงานของโปรแกรมเพื่อวิเคราะห์การทำงาน

จากการทดลองและวัดประสิทธิภาพในการทำงานของโปรแกรมจำลองสึนามิที่ได้ปรับปรุงจากโปรแกรมเดิมพบว่าทั้งบนระบบเพกาซัสและเวิร์คสเตชั่นสเปซแลบได้ผลการทำงานที่น่าพอใจโดยใช้เวลาในการคำนวณประมาณ 50 และ 60 นาที ตามลำดับ และได้ค่าความเร็วที่เพิ่มขึ้นเป็น 9.45 และ 5.34 ซึ่งเวลาที่ใช้ดังกล่าวมีความเร็วพอที่จะสามารถนำไปประเมินสถานการณ์เหตุการณ์ล่วงหน้าได้ ถ้าหากต้องการความเร็วที่เพิ่มขึ้นสามารถทำได้โดยการเลือกใช้เครื่องที่นำมาประมวลผลที่มีประสิทธิภาพที่สูงขึ้นเพื่อให้โปรแกรมทำงานได้รวดเร็วขึ้น เนื่องจากระบบที่ใช้ในการทดสอบของงานวิจัยนี้เป็นระบบที่มีขนาดเล็ก และเครื่องเวิร์คสเตชั่นใช้กราฟิกการ์ดที่มีหน่วยประมวลผลกราฟิกขนาดเล็ก นอกจากนี้ปัจจัยอื่นๆอย่างเช่น ชนิดของที่เก็บข้อมูลที่ใช้ก็มีส่วนเกี่ยวข้องกับเวลาที่ใช้ในการเขียนผลลัพธ์ด้วยเช่นกัน ซึ่งในการทดลองปรับปรุงประสิทธิภาพพบว่าส่วนที่ส่งผลต่อประสิทธิภาพของโปรแกรมมากที่สุดคือ การปรับขนาดของ Threads per block และการใช้ Page flipper เข้ามาช่วยในการสลับค่าข้อมูลใน

Region และการปรับขนาดของ Threads per block จะส่งผลต่อการทำงานของโปรแกรม โดยรวมโดยขึ้นอยู่กับสถาปัตยกรรมของฮาร์ดแวร์และตัวโปรแกรมเอง ถ้าโปรแกรมมีการใช้งานหน่วยความจำภายในปริมาณมาก การแบ่ง Threads ต้องคำนึงคิดขนาดของหน่วยความจำด้วย เพราะเป็นการใช้หน่วยความจำร่วม ถ้ามีการใช้หน่วยความจำเกินปริมาณที่มี จะทำให้เกิดการรอคอยเข้าถึงข้อมูล ซึ่งทำให้ส่งผลต่อประสิทธิภาพของโปรแกรม นอกจากนี้ยังมีเทคนิคอื่นๆที่สามารถนำมาปรับปรุงเพื่อเพิ่มประสิทธิภาพได้ เช่นถ้าต้องการใช้ Shared memory สำหรับโปรแกรมแบบ Finite different method อาจต้องมีการปรับรูปแบบของการเก็บข้อมูลในหน่วยความจำ และเพิ่มปริมาณงานต่อเธรดเพื่อให้เกิดการใช้งานหน่วยความจำแบบ data reuse ก็จะสามารถเพิ่มประสิทธิของโปรแกรมได้อีก แต่ต้องคำนึงถึงขนาดของหน่วยความจำแบบ Shared memory เพราะมีขนาดเล็กมากเมื่อเทียบกับหน่วยความจำหลักของหน่วยประมวลผลกราฟิก

ในการทดลองถัดมาได้ใช้ข้อมูลที่มีขนาดแตกต่างกันมาใช้ในการทดลอง เพื่อที่จะหาความสัมพันธ์ระหว่างเวลาและขนาดของข้อมูลที่ใช้ ซึ่งผลการทดลองพบว่าสมการที่สร้างขึ้นสามารถนำมาประเมินเวลาที่ใช้ในการทำงานของโปรแกรมได้ มีความแม่นยำอยู่ในระดับหนึ่ง ซึ่งพอที่จะสามารถคาดเดาเวลาที่จะใช้หากที่การกำหนดขนาดของข้อมูลที่ใช้

ส่วนการทดลองที่เหลือคือ ปรับค่าปัจจัยภายนอกอื่นๆ ซึ่งในการทดลอง ได้ทำการลดฟังก์ชั่นที่ไม่จำเป็นต่อระบบเตือนภัยและ ปรับความถี่รอบการเขียนผลลัพธ์ ซึ่งทั้งสองวิธีสามารถลดเวลาที่ใช้ได้พอสมควร โดยการปรับลดปัจจัยเหล่านี้จะขึ้นอยู่กับผู้เชี่ยวชาญที่ต้องการนำไปใช้ว่าต้องการการใช้งานในรูปแบบอย่างไร โดยปัจจัยที่เกี่ยวข้องกับความถี่รอบการเขียนผลลัพธ์ผู้ใช้สามารถคำนวณได้จากเวลาที่ต้องการ เพื่อหาค่าความถี่ที่ต้องใช้ในโปรแกรมและยังสามารถนำไปประยุกต์เข้ากับวิธีลดการทำงานบางฟังก์ชั่นออกเพื่อให้ได้ความเร็วที่เพิ่มขึ้นได้

5.2 ข้อเสนอแนะ

จากผลการทดลอง พบว่าโปรแกรมที่ได้พัฒนาขึ้นมีข้อจำกัดบางประการ ซึ่งขึ้นอยู่กับกรณีศึกษาที่นำมาใช้ เช่นในโปรแกรมจำลองสึนามิที่นำมาใช้ มีบางส่วนของโปรแกรมที่มีความเกี่ยวเนื่องกันของข้อมูลโดยไม่สามารถทำให้เป็นการคำนวณแบบขนานได้ ซึ่งมีปริมาณการ

ทำงานอยู่พอสมควร โดยการปรับปรุงเพื่อให้ได้ประสิทธิภาพที่ดีขึ้นต้องคำนวณถึงโครงสร้างของโปรแกรมเป็นหลัก ในบางโปรแกรมนั้นอาจจะไม่ได้รับประสิทธิภาพจากการปรับปรุงเท่าที่ควร อีกปัจจัยหนึ่งคือ หน่วยประมวลผลกราฟิกนั้นได้มีการปรับปรุงเพิ่มประสิทธิภาพและสถาปัตยกรรมให้ดีขึ้นเรื่อย ซึ่งการแก้ไขด้วยเทคนิคต่างๆในปัจจุบัน อาจจะถูกปรับปรุงแก้ไขด้วยสถาปัตยกรรมที่เปลี่ยนแปลงไปได้ ซึ่งในการเขียนโปรแกรมคำนวณแบบขนานบนหน่วยประมวลผลกราฟิกต้องเข้าใจสถาปัตยกรรมของรุ่นนั้นๆ เพื่อสามารถพัฒนาโปรแกรมให้ได้ประสิทธิภาพสูงสุด นอกจากนี้การพัฒนาโปรแกรมที่ได้ทำการแปลงภาษาของโปรแกรมเพื่อให้สามารถทำงานบนหน่วยประมวลผลกราฟิกนั้น จะส่งผลต่อความแม่นยำของผลลัพธ์ซึ่งในวีดีโอการ์ดรุ่นใหม่ๆได้เพิ่มความสามารถในด้านนี้อยู่เรื่อยๆ ในอนาคตการพัฒนาโปรแกรมบนหน่วยประมวลผลกราฟิกอาจจะไม่ส่งผลต่อความแม่นยำของโปรแกรม

รายการอ้างอิง

1. NVIDIA. *NVIDIA CUDA C Programming Guide*. 2012 [cited 2012 14 December]; Available from: http://docs.nvidia.com/cuda-c-programming-guide/index.html.
2. KHRONOS. *Online manual pages OpenCL 1.2* [cited 2012 14 December]; Available from: http://www.kh-ronos.org/registry/cl/sdk/1.2/docs/man/xhtml/.
3. Wikipedia. *Single instruction, multiple data (SIMD)*. [cited 2012 14 December]; Available from: http://en.wikipedia.org/wiki/SIMD.
4. Wikipedia. *Processing flow on CUDA*. [cited 2012 14 December]; Available from: http://en.wikipedia.org/wiki/CUDA.
5. Boyer, M., K. Skadron, and W. Weimer, *Automated Dynamic Analysis of CUDA Programs*, in *Third Workshop on Software Tools for MultiCore Systems*2008.
6. Ueng, S.-Z., et al., *CUDA-Lite: Reducing GPU Programming Complexity*, in *Languages and Compilers for Parallel Computing*, Jos and A. Nelson, Editors. 2008, Springer-Verlag. p. 1-15.
7. Yang, Y., et al., *A GPGPU compiler for memory optimization and parallelism management*, in *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation*2010, ACM: Toronto, Ontario, Canada. p. 86-97.
8. Eschweiler, D., D. Becker, and F. Wolf, *Patterns of Inefficient Performance Behavior in GPU Applications*, in *Proceedings of the 2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing*2011, IEEE Computer Society. p. 262-266.
9. Wikipedia. แผ่นดินไหวและคลื่นสึนามิในมหาสมุทรอินเดีย พ.ศ. *2547*. [cited 2012 14 December]; Available from: http://th.wikipedia.org/wiki/แผ่นดินไหวและคลื่นสึนามิในมหาสมุทรอินเดีย_พ.ศ._2547.
10. Imamura, F., A.C. Yalciner, and A.G. Ozyurt. *Tsunami Modeling Manual (TUNAMI – Tohoku University's Numerical Analysis Model for Investigation of Near-field Tsunamis)*. [cited 2012 8 December]; Available from: http://www.tsunami.civil.tohoku.ac.jp/hokusai3/J/projects/manual-ver-3.1.pdf.

11. นายกิตติพัฒน์ วิโรจน์ศิริ และ ผศ. ดร. วีระ เหมืองสิน, โปรแกรมแบบขนานเพื่อจำลองการเกิดคลื่นสึนามิ (*Parallel Tsunami Simulation Program)*, 2006, จุฬาลงกรณ์มหาวิทยาลัย: วิศวกรรมคอมพิวเตอร์ วิศวกรรมศาสตร์.
12. นายสิทธิกร ถาวรรัตนวณิช และ ผศ.ดร.วีระ เหมืองสิน, การพัฒนาโปรแกรมคำนวณแบบขนานที่ปรับปรุงประสิทธิภาพได้สำหรับปัญหาที่มีหลายระดับความละเอียด: กรณีศึกษาการจำลองสึนามิ, *(Parallel Program Development with Adaptive Performance Tuning For Multi-scale Problem: A Case Study Of Tsunami Simulation)*, วิศวกรรมคอมพิวเตอร์ วิศวกรรมศาสตร์2011, จุฬาลงกรณ์มหาวิทยาลัย.
13. นายเขม อำนวยลาภ, นายฉัตรชัย พีรพัฒน์ดิษฐ์ และ ผศ.ดร.วีระ เหมืองสิน, ระบบเตือนภัยสึนามิด้วยการจำลองแบบทันกาลบนหน่วยประมวลผลกราฟิกส์, *(Tsunami Warning System with Real-Time Simulation on Graphics Processing Unit)*, 2011, จุฬาลงกรณ์มหาวิทยาลัย: วิศวกรรมคอมพิวเตอร์ วิศวกรรมศาสตร์.

ภาคผนวก

ภาคผนวก

ฟังก์ชั่นบนหน่วยประมวลผลกราฟิก

function.cu

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

__global__ void cuda_interqt(int ifn, int jfn, int kk, int nt, float *ma,
    float *mb, float *na, float *nb, float *m_ta, float *m_tb, float *n_ta,
    float *n_tb) {

    int x = threadIdx.x + blockIdx.x * blockDim.x;
    int y = threadIdx.y + blockIdx.y * blockDim.y;
    int index = x + y * blockDim.x * gridDim.x;

    if (index > (ifn*jfn)-1) return;

    m_ta[index] = ma[index];
    n_ta[index] = na[index];

    m_tb[index] = ma[index]+((mb[index]-ma[index])*kk)/nt;
    n_tb[index] = na[index]+((nb[index]-na[index])*kk)/nt;
}

__global__ void cuda_jnq_s2c(int ix, int jx, int iy, int jy, float *mx,
    float *mxb, float *nx, float *nxb, float *my, float *myb, float *ny,
    float *nyb, float *hy, int* l0, int bchk) {

    int x = threadIdx.x + blockIdx.x * blockDim.x;
    int y = threadIdx.y + blockIdx.y * blockDim.y;
    int index = x + y * blockDim.x * gridDim.x;

    if (index != 0) return;

    int chk, iss, jss, ies, jes, isl, jsl, iel, jel;
    int i, j, ii, jj, is, js, kb, index_x, index_y;
    float si, sj, di, dj;

    iss = 1;
    jss = 1;
    ies = iy - 1;
    jes = jy - 1;
    isl = l0[0] - 1;
    jsl = l0[1] - 1;
    iel = l0[2] - 1;
    jel = l0[3] - 1;
    chk = bchk;
    kb = chk/1000;

    if(kb == 1) {
        chk = chk-1000;
        i = iss;
        j = jss -1;
        ii = isl -1;
        jj = jsl -1;
        do {
            index_x = (ii*jx)+jj;
            index_y = (i*jy)+j;
```

function.cu

```
56              if(hy[index_y+1] < 0.0) {
57                  nyb[index_y] = 0.0;
58              } else {
59                  si = (i-iss+4.5)/8.0;
60                  is = (int)si;
61                  di = si - is;
62                  ii = is + isl - 1;
63                  nyb[index_y] = (1-di)*nxb[index_x]+di*nxb[index_x+jx];
64              }
65              ny[index_y] = nyb[index_y];
66              i = i+1;
67          } while(i <= ies);
68      }
69
70      kb = chk/100;
71      if(kb == 1) {
72          chk = chk-100;
73          i = ies;
74          j = jss;
75          ii = iel;
76          jj = jsl - 1;
77          do {
78              index_x = (ii*jx)+jj;
79              index_y = (i*jy)+j;
80              if(hy[index_y] < 0.0) {
81                  myb[index_y] = 0.0;
82              } else {
83                  sj = (j-jss+4.5)/8.0;
84                  js = (int)sj;
85                  dj = sj - js;
86                  jj = js + jsl - 1;
87                  myb[index_y] = (1-dj)*mxb[index_x]+dj*mxb[index_x+1];
88              }
89              my[index_y] = myb[index_y];
90              j = j + 1;
91          } while(j <= jes);
92      }
93
94      kb = chk/10;
95      if(kb == 1) {
96          chk = chk - 10;
97          i = iss;
98          j = jes;
99          ii = isl - 1;
100         jj = jel;
101         do {
102             index_x = (ii*jx)+jj;
103             index_y = (i*jy)+j;
104             if(hy[index_y] < 0.0) {
105                 nyb[index_y] = 0.0;
106             } else {
107                 si = (i-iss+4.5)/8.0;
108                 is = (int)si;
109                 di = si - is;
110                 ii = is + isl - 1;
```

function.cu

```
111                 nyb[index_y] = (1-di)*nxb[index_x]+di*nxb[index_x+jx];
112             }
113             ny[index_y] = nyb[index_y];
114             i = i+1;
115         } while(i <= ies);
116     }
117
118     if(chk == 1) {
119         i = iss - 1;
120         j = jss;
121         ii = isl - 1;
122         jj = jsl - 1;
123         do {
124             index_x = (ii*jx)+jj;
125             index_y = (i*jy)+j;
126             if(hy[index_y+jy] < 0.0) {
127                 myb[index_y] = 0.0;
128             } else {
129                 sj = (j-jss+4.5)/8.0;
130                 js = (int)sj;
131                 dj = sj - js;
132                 jj = js + jsl - 1;
133                 myb[index_y] = (1-dj)*mxb[index_x]+dj*mxb[index_x+1];
134             }
135             my[index_y] = myb[index_y];
136             j = j+1;
137         } while(j <= jes);
138     }
139 }
140
141 __global__ void cuda_nlmass(int ifn, int jfn, float *ma, float *na, float *za,
142     float *zb, float *hz, float *dzb, float th, float rx) {
143
144     int x = threadIdx.x + blockIdx.x * blockDim.x;
145     int y = threadIdx.y + blockIdx.y * blockDim.y;
146     int index = x + y * blockDim.x * gridDim.x;
147
148     int i = index / jfn;
149     int j = index % jfn;
150
151     if (index > (ifn*jfn)-1 || i < 1 || j < 1) return;
152
153     float GX = 1.0e-5;
154     float zz, dd;
155     zz = 0;
156     dd = 0;
157
158     if(hz[index] >= th) {
159         zz = za[index] - rx*(ma[index]-ma[index-jfn])-rx*(na[index]-na[index-1]);
160         if(fabs(zz) < GX) zz = 0.0;
161         dd = zz + hz[index];
162         if(dd < GX) dd = 0.0;
163         dzb[index] = dd;
164         zb[index] = dd - hz[index];
165     }
```

function.cu

```
166  }
167
168  __global__ void cuda_jnz(int ix, int jx, int iy, int jy, float *zx,
169      float *zxb, float *zy, float *zyb, float *dzx, float *dzxb,
170      float *dzy, float *dzyb, float *hy, int l0[4], int bchk) {
171
172      int x = threadIdx.x + blockIdx.x * blockDim.x;
173      int y = threadIdx.y + blockIdx.y * blockDim.y;
174      int index = x + y * blockDim.x * gridDim.x;
175
176      if (index != 0) return;
177
178      int chk, iss, jss, ies, jes, isl, jsl, iel, jel;
179      int i, j, ii, jj, i1, j1, l, kb, index_x, index_y;
180      float s, sd;
181
182      iss = 1;
183      jss = 1;
184      ies = iy - 1;
185      jes = jy - 1;
186      isl = l0[0] - 1;
187      jsl = l0[1] - 1;
188      iel = l0[2] - 1;
189      jel = l0[3] - 1;
190      chk = bchk;
191      kb = chk/1000;
192
193      if(kb == 1) {
194          chk = chk - 1000;
195          ii = isl;
196          jj = jsl;
197          i1 = iss;
198          j1 = jss;
199          do {
200              s = 0.0;
201              l = 0;
202              sd = 0.0;
203              for(j = j1; j < j1+2; ++j) {
204                  for(i = i1; i < i1+2; ++i) {
205                      index_y = (i*jy)+j;
206                      if(hy[index_y] > 0.0) {
207                          s = s + zyb[index_y];
208                          sd = sd + dzyb[index_y];
209                          l = l + 1;
210                      }
211                  }
212              }
213              index_x = (ii*jx)+jj;
214              if(l >= 5) {
215                  zxb[index_x] = s/l;
216                  dzxb[index_x] = sd/l;
217              } else {
218                  zxb[index_x] = 0.0;
219              }
220              ii = ii + 1;
```

function.cu

```
221             i1 = i1 + 3;
222         } while(i1+2 <= ies);
223     }
224
225     kb = chk/100;
226     if(kb == 1)
227     {
228         chk = chk - 100;
229         ii = iel;
230         jj = jsl;
231         i1 = ies - 2;
232         j1 = jss;
233         do {
234             s = 0.0;
235             l = 0;
236             sd = 0.0;
237             for(j = j1; j < j1+2; ++j) {
238                 for(i = i1; i < i1+2; ++i) {
239                     index_y = (i*jy)+j;
240                     if(hy[index_y] > 0.0) {
241                         s = s + zyb[index_y];
242                         sd = sd + dzyb[index_y];
243                         l = l + 1;
244                     }
245                 }
246             }
247             index_x = (ii*jx)+jj;
248             if(l >= 5) {
249                 zxb[index_x] = s/l;
250                 dzxb[index_x] = sd/l;
251             } else {
252                 zxb[index_x] = 0.0;
253             }
254             jj = jj + 1;
255             j1 = j1 + 3;
256         } while(j1+2 <= jes);
257     }
258
259     kb = chk/10;
260     if(kb == 1) {
261         chk = chk - 10;
262         ii = isl;
263         jj = jel;
264         i1 = iss;
265         j1 = jes - 2;
266         do {
267             s = 0.0;
268             l = 0;
269             sd = 0.0;
270             for(j = j1; j <= j1+2; ++j) {
271                 for(i = i1; i <= i1+2; ++i) {
272                     index_y = (i*jy)+j;
273                     if(hy[index_y] > 0.0) {
274                         s = s + zyb[index_y];
275                         sd = sd + dzyb[index_y];
```

function.cu

```
276                         l = l + 1;
277                     }
278                 }
279             }
280             index_x = (ii*jx)+jj;
281             if(l >= 5) {
282                 zxb[index_x] = s/l;
283                 dzxb[index_x] = sd/l;
284             } else {
285                 zxb[index_x] = 0.0;
286             }
287             ii = ii + 1;
288             i1 = i1 + 3;
289         } while(i1+2 <= ies);
290     }
291
292     if(chk == 1) {
293         ii = isl;
294         jj = jsl;
295         i1 = iss;
296         j1 = jss;
297         do {
298             s = 0.0;
299             l = 0;
300             sd = 0.0;
301             for(i = i1; i <= i1+2; ++i) {
302                 for(j = j1; j <= j1+2; ++j) {
303                     index_y = (i*jy)+j;
304                     if(hy[index_y] > 0.0) {
305                         s = s + zyb[index_y];
306                         sd = sd + dzyb[index_y];
307                         l = l + 1;
308                     }
309                 }
310             }
311             index_x = (ii*jx)+jj;
312             if(l >= 5) {
313                 zxb[index_x] = s/l;
314                 dzxb[index_x] = sd/l;
315             } else {
316                 zxb[index_x] = 0.0;
317             }
318             jj = jj + 1;
319             j1 = j1 + 3;
320         } while(j1+2 <= jes);
321     }
322 }
323
324 __device__ void cuda_xmmt(float gg, int i, int j, int ifn, int jfn,
325     float *hz, float *za, float *zb, float *dza, float *dzb,
326     float *dma, float *dmb, float *ma, float *mb, float *na,
327     float *nb, float rx, float fn) {
328
329     float dd, xnn, ff, xm, xdm, xne, xmm0;
330     float gx = 1.0E-5;
```

function.cu

```
331         int index = (i*jfn)+j;
332 
333         dd = 0.0;
334         xnn = 0.0;
335         ff = 0.0;
336         xm = 0.0;
337         xdm = 0.0;
338         xne = 0.0;
339         xmm0 = 0.0;
340 
341         if (i == ifn-1) {
342             mb[index] = 0.0;
343             return;
344         }
345 
346         if (dzb[index] > gx) {
347             if (dzb[index+jfn] > gx) {
348                 dd = dmb[index];
349             } else {
350                 if (zb[index]+hz[index+jfn] > gx) {
351                     dd = zb[index] + hz[index+jfn];
352                 } else {
353                     mb[index] = 0.0;
354                     return;
355                 }
356             }
357         } else {
358             if (dzb[index+jfn] > gx) {
359                 if (zb[index+jfn] + hz[index] > gx) {
360                     dd = zb[index+jfn] + hz[index];
361                 } else {
362                     mb[index] = 0.0;
363                     return;
364                 }
365             } else {
366                 mb[index] = 0.0;
367                 return;
368             }
369         }
370         // ------  LINEAR TERM  ------
371         if (j != 0) {
372             if (dd >= gx) {
373                 xnn = 0.25*(na[index]+na[index+jfn]+na[index-1]+na[index+jfn-1]);
374                 ff = fn*sqrt(pow(ma[index],2)+pow(xnn,2.0f))/pow(dd,(float)(7.0/3.0));
375                 xm = (1.0-ff)*ma[index]-gg*rx*dd*(zb[index+jfn]-zb[index]);
376             } else {
377                 mb[index] = 0.0;
378                 return;
379             }
380         } else {
381             mb[index] = 0.0;
382             return;
383         }
384         // ------  NON-LINEAR TERM  ------
385         if (i > 2 && i < ifn-4 && j > 2 && j < jfn-4) {
```

function.cu

```
386         if (dma[index] >= gx) {
387             if (ma[index] > 0.0) {
388                 if (i != 0) {
389                     if (dma[index-jfn] >= gx) {
390                         if (dzb[index] < gx)
391                             xdm = 0.0;
392                         else if (dzb[index-jfn] < gx)
393                             xdm = 0.0;
394                         else
395                             xdm = pow(ma[index-jfn],2)/dma[index-jfn];
396                     } else {
397                         xdm = 0.0;
398                     }
399                     xm = xm-rx*(pow(ma[index],2)/dma[index]-xdm);
400                 }
401             } else {
402                 if (dma[index+jfn] >= gx) {
403                     if (dzb[index+jfn] < gx)
404                         xdm = 0.0;
405                     else if (dzb[index+jfn+jfn] < gx)
406                         xdm = 0.0;
407                     else
408                         xdm = pow(ma[index+jfn],2)/dma[index+jfn];
409                 } else {
410                     xdm = 0.0;
411                 }
412                 xm = xm-rx*(xdm-pow(ma[index],2)/dma[index]);
413             }
414
415             if (xnn > 0.0) {
416                 if (j != 1) {
417                     if (dma[index-1] >= gx) {
418                         if (dzb[index-2] < gx) {
419                             xdm = 0.0;
420                             xmm0 = xm-rx*(ma[index]*xnn/dma[index]-xdm);
421                         } else {
422                             if (dzb[index-1] < gx) {
423                                 xmm0 = xm;
424                             } else if (dzb[index+jfn-1] < gx) {
425                                 xmm0 = xm;
426                             } else if (dzb[index+jfn-2] < gx) {
427                                 xmm0 = xm;
428                             } else {
429                                 xne = 0.25*(na[index-1]+na[index+jfn-1]
430                                     +na[index-2]+na[index+jfn-2]);
431                                 xdm = ma[index-1]*xne/dma[index-1];
432                                 xmm0 = xm-rx*(ma[index]*xnn/dma[index]-xdm);
433                             }
434                         }
435                         xm = xmm0/(1.0+ff);
436                     } else {
437                         xdm = 0.0;
438                         xm = xm-rx*(ma[index]*xnn/dma[index]-xdm);
439                         xm = xm/(1.0+ff);
440                     }
```

function.cu

```
441                 }
442             } else {
443                 if (dma[index+1] >= gx) {
444                     if (dzb[index+1] < gx) {
445                         xdm = 0.0;
446                         xmm0 = xm-rx*(xdm-ma[index]*xnn/dma[index]);
447                     } else {
448                         if (dzb[index+2] < gx) {
449                             xmm0 = xm;
450                         } else if (dzb[index+jfn+1] < gx) {
451                             xmm0 = xm;
452                         } else if (dzb[index+jfn+2] < gx) {
453                             xmm0 = xm;
454                         } else {
455                             xne = 0.25*(na[index+1]+na[index+jfn+1]
456                                 +na[index]+na[index+jfn]);
457                             xdm = ma[index+1]*xne/dma[index+1];
458                             xmm0 = xm-rx*(xdm-ma[index]*xnn/dma[index]);
459                         }
460                     }
461                     xm = xmm0/(1.0+ff);
462                 } else {
463                     xdm = 0.0;
464                     xm = xm-rx*(xdm-ma[index]*xnn/dma[index]);
465                     xm = xm/(1.0+ff);
466                 }
467             }
468 
469         } else {
470             xm = xm/(1.0+ff);
471         }
472     } else {
473         xm = xm/(1.0+ff);
474     }
475     // ---- LIMITING OF DISCHARGE ----
476     if (fabs(xm) < gx) xm = 0.0;
477     if (xm > 10.0*dd) xm = 10.0*dd;
478     if (xm < -10.0*dd) xm = -10.0*dd;
479     mb[index] = xm;
480 
481 }
482 
483 __device__ void cuda_ymmt(float gg, int i, int j, int ifn, int jfn,
484     float *hz, float *za, float *zb, float *dza, float *dzb,
485     float *dna, float *dnb, float *ma, float *mb, float *na,
486     float *nb, float rx, float fn) {
487 
488     float dd, xmm, ff, xn, xdn, xme, xnn0;
489     float gx = 1.0E-5;
490     int index = (i*jfn)+j;
491 
492     dd = 0.0;
493     xmm = 0.0;
494     ff = 0.0;
495     xn = 0.0;
```

function.cu

```
496         xdn = 0.0;
497         xme = 0.0;
498         xnn0 = 0.0;
499 
500         if (j == jfn-1) {
501             nb[index] = 0.0;
502             return;
503         }
504 
505         if (dzb[index] > gx) {
506             if (dzb[index+1] > gx) {
507                 dd = dnb[index];
508             } else {
509                 if (zb[index]+hz[index+1] > gx) {
510                     dd = zb[index]+hz[index+1];
511                 } else {
512                     nb[index] = 0.0;
513                     return;
514                 }
515             }
516         } else {
517             if (dzb[index+1] > gx) {
518                 if (zb[index+1]+hz[index] > gx) {
519                     dd = zb[index+1]+hz[index];
520                 } else {
521                     nb[index] = 0.0;
522                     return;
523                 }
524             } else {
525                 nb[index] = 0.0;
526                 return;
527             }
528         }
529         // ------  LINEAR TERM  ------
530         if (i != 0) {
531             if (dd >= gx) {
532                 xmm = 0.25*(ma[index]+ma[index+1]+ma[index-jfn]+ma[index-jfn+1]);
533                 ff = fn*sqrt(pow(na[index],2)+pow(xmm,2.0f))/pow(dd,(float)(7.0/3.0));
534                 xn = (1.0-ff)*na[index]-gg*rx*dd*(zb[index+1]-zb[index]);
535             } else {
536                 nb[index] = 0.0;
537                 return;
538             }
539         } else {
540             nb[index] = 0.0;
541             return;
542         }
543         // ------  NON-LINEAR TERM  ------
544         if (i > 2 && i < ifn - 4 && j > 2 && j < jfn - 4) {
545             if (dna[index] >= gx) {
546                 if (na[index] > 0.0) {
547                     if (j != 0) {
548                         if (dna[index-1] >= gx) {
549                             if (dzb[index] < gx) {
550                                 xdn = 0.0;
```

function.cu

```
551                             } else if (dzb[index-1] < gx) {
552                                 xdn = 0.0;
553                             } else {
554                                 xdn = pow(na[index-1],2)/dna[index-1];
555                             }
556                         } else {
557                             xdn = 0.0;
558                         }
559                         xn = xn-rx*(pow(na[index],2)/dna[index]-xdn);
560                     }
561                 } else {
562                     if (dna[index+1] >= gx) {
563                         if (dzb[index+1] < gx) {
564                             xdn = 0.0;
565                         } else if (dzb[index+2] < gx) {
566                             xdn = 0.0;
567                         } else {
568                             xdn = pow(na[index+1],2)/dna[index+1];
569                         }
570                     } else {
571                         xdn = 0.0;
572                     }
573                     xn = xn-rx*(xdn-pow(na[index],2)/dna[index]);
574                 }
575
576                 if (xmm > 0.0) {
577                     if (i != 1) {
578                         if (dna[index-jfn] >= gx) {
579                             if (dzb[index-jfn-jfn] < gx) {
580                                 xdn = 0.0;
581                                 xnn0 = xn-rx*(na[index]*xmm/dna[index]-xdn);
582                             } else {
583                                 if (dzb[index-jfn-jfn+1] < gx) {
584                                     xnn0 = xn;
585                                 } else if (dzb[index-jfn] < gx) {
586                                     xnn0 = xn;
587                                 } else if (dzb[index-jfn+1] < gx) {
588                                     xnn0 = xn;
589                                 } else {
590                                     xme = 0.25*(ma[index-jfn]+ma[index-jfn+1]
591                                         +ma[index-jfn-jfn]+ma[index-jfn-jfn+1]);
592                                     xdn = na[index-jfn]*xme/dna[index-jfn];
593                                     xnn0 = xn-rx*(na[index]*xmm/dna[index]-xdn);
594                                 }
595                             }
596                             xn = xnn0/(1.0+ff);
597                         } else {
598                             xdn = 0.0;
599                             xn = xn-rx*(na[index]*xmm/dna[index]-xdn);
600                             xn = xn/(1.0+ff);
601                         }
602                     }
603                 } else {
604                     if ( dna[index+jfn] >= gx) {
605                         if (dzb[index+jfn] < gx) {
```

function.cu

```
606                         xdn = 0.0;
607                         xnn0 = xn-rx*(xdn-na[index]*xmm/dna[index]);
608                     } else {
609                         if (dzb[index+jfn+jfn] < gx) {
610                             xnn0 = xn;
611                         } else if (dzb[index+jfn+1] < gx) {
612                             xnn0 = xn;
613                         } else if (dzb[index+jfn+jfn+1] < gx) {
614                             xnn0 = xn;
615                         } else {
616                             xme = 0.25*(ma[index+jfn]+ma[index+jfn+1]
617                                 +ma[index]+ma[index+1]);
618                             xdn = na[index+jfn]*xme/dna[index+jfn];
619                             xnn0 = xn-rx*(xdn-na[index]*xmm/dna[index]);
620                         }
621                     }
622                     xn = xnn0/(1.0+ff);
623                 } else {
624                     xdn = 0.0;
625                     xn = xn-rx*(xdn-na[index]*xmm/dna[index]);
626                     xn = xn/(1.0+ff);
627                 }
628             }
629         } else {
630             xn = xn/(1.0+ff);
631         }
632     } else {
633         xn = xn/(1.0+ff);
634     }
635     // ---- LIMITING OF DISCHARGE ----
636     if (fabs(xn) < gx) xn = 0.0;
637     if (xn > 10.0*dd) xn = 10.0*dd;
638     if (xn < -10.0*dd) xn = -10.0*dd;
639     nb[index] = xn;
640 }
641
642 __global__ void cuda_nlmmt(int ifn, int jfn, float th, float *za,
643     float *zb, float *ma, float *mb, float *na, float *nb,
644     float *dza, float *dzb, float *dma, float *dmb, float *dna,
645     float *dnb, float *hz, float *hm, float *hn, float rx, float dt, float fm) {
646     float gx=1.0E-5;
647     float dm1, dm2, dn1, dn2;
648     int x = threadIdx.x + blockIdx.x * blockDim.x;
649     int y = threadIdx.y + blockIdx.y * blockDim.y;
650     int index = x + y * blockDim.x * gridDim.x;
651
652     int i = index / jfn;
653     int j = index % jfn;
654
655     if (index > (ifn*jfn)-1) return;
656
657     dma[index] = 0.0;
658     dmb[index] = 0.0;
659     dna[index] = 0.0;
660     dnb[index] = 0.0;
```

function.cu

```
661
662         if (i < ifn-1) {
663             dm1 = 0.25*(za[index]+zb[index]+za[index+jfn]+zb[index+jfn])
664                 + 0.5*(hz[index]+hz[index+jfn]);
665             dm2 = 0.5*(zb[index]+zb[index+jfn]+hz[index]+hz[index+jfn]);
666         } else {
667             dm1 = zb[index]+hz[index];
668             dm2 = zb[index]+hz[index];
669         }
670         if (dm1 >= gx) dma[index] = dm1;
671         if (dm2 >= gx) dmb[index] = dm2;
672
673         if (j < jfn-1) {
674             dn1 = 0.25*(za[index]+zb[index]+za[index+1]+zb[index+1])
675                 + 0.5*(hz[index]+hz[index+1]);
676             dn2 = 0.5*(zb[index]+zb[index+1]+hz[index]+hz[index+1]);
677         } else {
678             dn1 = zb[index]+hz[index];
679             dn2 = zb[index]+hz[index];
680         }
681         if (dn1 >= gx) dna[index] = dn1;
682         if (dn2 >= gx) dnb[index] = dn2;
683
684     }
685
686     __global__ void cuda_nlmmt2(int ifn, int jfn, float th, float *za,
687         float *zb, float *ma, float *mb, float *na, float *nb, float *dza,
688         float *dzb, float *dma, float *dmb, float *dna, float *dnb,
689         float *hz, float *hm, float *hn, float rx, float dt, float fm) {
690
691         float gg = 9.81;
692         int x = threadIdx.x + blockIdx.x * blockDim.x;
693         int y = threadIdx.y + blockIdx.y * blockDim.y;
694         int index = x + y * blockDim.x * gridDim.x;
695         float fn = 0.5*dt*gg*pow(fm,2);
696
697         int i = index / jfn;
698         int j = index % jfn;
699
700         if (index > (ifn*jfn)-1) return;
701
702         if (hz[index] > th && hm[index] > th) {
703             cuda_xmmt(gg, i, j, ifn, jfn, hz, za, zb, dza, dzb, dma, dmb,
704                 ma, mb, na, nb, rx, fn);
705         }
706         __syncthreads();
707         if (hz[index] > th && hn[index] > th) {
708             cuda_ymmt(gg, i, j, ifn, jfn, hz, za, zb, dza, dzb, dna, dnb,
709                 ma, mb, na, nb, rx, fn);
710         }
711
712     }
713
714     __global__ void cuda_jnq(int ix, int jx, int iy, int jy, float *mx,
715         float *mxb, float *nx, float *nxb, float *my, float *myb, float *ny,
```

function.cu

```
716         float *nyb, float *hy, int* l0, int bchk) {
717
718         int x = threadIdx.x + blockIdx.x * blockDim.x;
719         int y = threadIdx.y + blockIdx.y * blockDim.y;
720         int index = x + y * blockDim.x * gridDim.x;
721
722         if (index != 0) return;
723
724         int chk, iss, jss, ies, jes, isl, jsl, iel, jel;
725         int i, j, ii, jj, is, js, kb, index_x, index_y;
726         float si, sj, di, dj;
727
728         iss = 1;
729         jss = 1;
730         ies = iy - 1;
731         jes = jy - 1;
732         isl = l0[0] - 1;
733         jsl = l0[1] - 1;
734         iel = l0[2] - 1;
735         jel = l0[3] - 1;
736         chk = bchk;
737         kb = chk/1000;
738         if(kb == 1) {
739             chk = chk-1000;
740             i = iss;
741             j = jss -1;
742             ii = isl -1;
743             jj = jsl -1;
744             do {
745                 index_x = (ii*jx)+jj;
746                 index_y = (i*jy)+j;
747                 if(hy[index_y+1] < 0.0) {
748                     nyb[index_y] = 0.0;
749                 } else {
750                     si = (i-iss+2)/3.0;
751                     is = (int)si;
752                     di = si - is;
753                     ii = is + isl - 1;
754                     nyb[index_y] = (1-di)*nxb[index_x]+di*nxb[index_x+jx];
755                 }
756                 i = i + 1;
757             } while(i <= ies);
758         }
759
760         kb = chk/100;
761         if(kb == 1) {
762             chk = chk - 100;
763             i = ies;
764             j = jss;
765             ii = iel;
766             jj = jsl - 1;
767             do {
768                 index_x = (ii*jx)+jj;
769                 index_y = (i*jy)+j;
770                 if(hy[index_y] < 0.0) {
```

function.cu

```
771                 myb[index_y] = 0.0;
772             } else {
773                 sj = (j-jss+2)/3.0;
774                 js = (int)sj;
775                 dj = sj - js;
776                 jj = js + jsl - 1;
777                 myb[index_y] = (1-dj)*mxb[index_x]+dj*mxb[index_x+1];
778             }
779             j = j + 1;
780         } while(j <= jes);
781     }
782
783     kb = chk/10;
784     if(kb == 1) {
785         chk = chk - 10;
786         i = iss;
787         j = jes;
788         ii = isl - 1;
789         jj = jel;
790         do {
791             index_x = (ii*jx)+jj;
792             index_y = (i*jy)+j;
793             if(hy[index_y] < 0.0) {
794                 nyb[index_y] = 0.0;
795             } else {
796             si = (i - iss + 2)/3.0f;
797                 is = (int)si;
798                 di = si - is;
799                 ii = is + isl - 1;
800                 nyb[index_y] = (1-di)*nxb[index_x]+di*nxb[index_x+jx];
801             }
802             i = i + 1;
803         } while(i <= ies);
804     }
805
806     if(chk == 1) {
807         i = iss - 1;
808         j = jss;
809         ii = isl - 1;
810         jj = jsl - 1;
811         do {
812             index_x = (ii*jx)+jj;
813             index_y = (i*jy)+j;
814             if(hy[index_y+jy] < 0.0) {
815                 myb[index_y] = 0.0;
816             } else {
817                 sj = (j - jss + 2)/3.0;
818                 js = (int)sj;
819                 dj = sj - js;
820                 jj = js + jsl - 1;
821                 myb[index_y] = (1-dj) * mxb[index_x] + dj * mxb[index_x+1];
822             }
823             j = j + 1;
824         } while(j <= jes);
825     }
```

function.cu

```
826 }
827
828 __global__ void cuda_etamap(int ifn, int jfn, float *h, float *za, float *zb,
829     int kk, float *eta, float tempf, float *inetaf) {
830
831     int x = threadIdx.x + blockIdx.x * blockDim.x;
832     int y = threadIdx.y + blockIdx.y * blockDim.y;
833     int index = x + y * blockDim.x * gridDim.x;
834
835     if (index > (ifn*jfn)-1) return;
836
837     float diff;
838     if(h[index] > 0) {
839         if(eta[index] == 0) {
840             diff = zb[index] - inetaf[index];
841             if(diff > tempf || diff < -tempf)
842                 eta[index] = kk;
843         }
844     }
845 }
846
847 __global__ void cuda_change(int ifn, int jfn, float *za, float *zb, float *ma,
848     float *mb, float *na, float *nb, float *dza, float *dzb) {
849
850     int x = threadIdx.x + blockIdx.x * blockDim.x;
851     int y = threadIdx.y + blockIdx.y * blockDim.y;
852     int index = x + y * blockDim.x * gridDim.x;
853
854     if (index > (ifn*jfn)-1) return;
855
856     za[index] = zb[index];
857     ma[index] = mb[index];
858     na[index] = nb[index];
859     dza[index] = dzb[index];
860 }
861
862 __global__ void cuda_change_flipper(int ifn, int jfn, float *za, float *zb,
863     float *ma, float *mb, float *na, float *nb, float *dza, float *dzb) {
864
865     int x = threadIdx.x + blockIdx.x * blockDim.x;
866     int y = threadIdx.y + blockIdx.y * blockDim.y;
867     int index = x + y * blockDim.x * gridDim.x;
868
869     if (index != 0) return;
870
871     float* swap;
872     swap = za;
873     za = zb;
874     zb = swap;
875
876     swap = ma;
877     ma = mb;
878     mb = swap;
879
880     swap = na;
```

function.cu

```
881         na = nb;
882         nb = swap;
883 
884         swap = dza;
885         dza = dzb;
886         dzb = swap;
887 
888     }
889 
890     __global__ void cuda_zmax(int ifn, int jfn, float *za, float *zb, float *zm) {
891 
892         int x = threadIdx.x + blockIdx.x * blockDim.x;
893         int y = threadIdx.y + blockIdx.y * blockDim.y;
894         int index = x + y * blockDim.x * gridDim.x;
895 
896         if (index > (ifn*jfn)-1) return;
897 
898         if(zm[index] < zb[index]) {
899             zm[index] = zb[index];
900         }
901     }
902 
903     __global__ void cuda_mnmax(int ifn, int jfn, float *ma, float *mb, float *na,
904         float *nb, float *za, float *zb, float *mm, float *nm, float *zmn) {
905 
906         int x = threadIdx.x + blockIdx.x * blockDim.x;
907         int y = threadIdx.y + blockIdx.y * blockDim.y;
908         int index = x + y * blockDim.x * gridDim.x;
909 
910         if (index > (ifn*jfn)-1) return;
911 
912         float tempmn, tempmnmax;
913         tempmn = sqrt(pow(mb[index],2) + pow(nb[index],2));
914         tempmnmax = sqrt(pow(mm[index],2) + pow(nm[index],2));
915         if(tempmnmax < tempmn) {
916             mm[index] = mb[index];
917             nm[index] = nb[index];
918             zmn[index] = zb[index];
919         }
920     }
```

ประวัติผู้เขียนวิทยานิพนธ์

นายพงษ์พัฒน์ เป้าเพ็ชร์ เกิดวันที่ 11 กรกฎาคม พ.ศ. 2531 ที่จังหวัดกรุงเทพมหานคร เริ่มศึกษาชั้น ประถมศึกษาที่โรงเรียนสันติวัน ชั้นมัธยมศึกษาตอนต้นและตอนปลายที่โรงเรียนโยธินบูรณะ สำเร็จการศึกษาในหลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ ปีการศึกษา 2553 และเข้ารับการศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2554