การจัดประชุมเสนอผลงานวิจัยระดับบัณฑิตศึกษา มหาวิทยาลัยสุโขทัยธรรมาธิราช ครั้งที่ 4
The 4th STOU Graduate Research Conference

คอมไพเลอร์แบบคำสั่งกำกับเพื่อเร่งความเร็วของคำสั่งซีพลัสพลัส 11 ฟอร์อีชโดยใช้ระบบประมวลผลร่วมหลายประเภท

A Directive-Based Compiler for Accelerating the C++11 for Each Instruction on Heterogeneous Computing System

จตุรภัทร สุวรรณเจริญ (Chaturapat Suwancharoen) * วรวรรณ ดีอัซ การ์บาโย (Worawan Diaz Carballo) **

บทคัดย่อ

งานวิจัยฉบับนี้มีวัตถุประสงค์เพื่อเร่งความเร็วการประมวลผลของคำสั่งฟอร์อีชในภาษาซีพลัสพลัส 11 โดยเปลี่ยนวิธีประมวลผลจากเดิมที่เป็นการประมวลผลเชิงลำดับบนซีพียู ให้เป็นการประมวลผลเชิงขนานบนระบบประมวลผลร่วมหลายประเภท ทั้งนี้เพื่อเพิ่มสมรรถนะให้กับการประมวลผลโปรแกรม และเพิ่มผลิตภาพในการพัฒนาโปรแกรมของโปรแกรมเมอร์

ในการดำเนินงานวิจัยได้มีการสร้างโครงงานต้นแบบที่ได้งานได้จริง โดยการออกแบบคำสั่งกำกับให้กับคำสั่งฟอร์อีชและสร้างคอมไพเลอร์เพื่อใช้ในการคอมไพล์โค้ดต้นฉบับ การประเมินผลใช้การทดลองเชิงเปรียบเทียบกับโปรแกรมที่เขียนด้วยภาษาซีพลัสมาตรฐาน และโปรแกรมที่เขียนด้วยระบบประมวลผลร่วมหลายประเภทโดยตรง เพื่อหาสมรรถนะและผลิตภาพของโปรแกรม

ผลการวิจัยพบว่าโครงงานวิจัยสามารถเร่งความเร็วของคำสั่งฟอร์อีชได้ถึง 203 เท่าของโปรแกรมที่เขียนด้วยภาษาซีพลัสพลัสมาตรฐาน และมีสมรรถนะเทียบเท่ากับโปรแกรมที่เขียนด้วยระบบประมวลผลร่วมหลายประเภทโดยตรง ในขณะโค้ดต้นฉบับของงานวิจัยยังคงมีขนาดสั้นและไม่ซับซ้อนเทียบเท่ากับโค้ดต้นฉบับภาษาซีพลัสพลัสมาตรฐาน

คำสำคัญ คอมไพเลอร์ ซีพลัสพลัส การประมวลผลเชิงขนาน หน่วยประมวลผลกราฟฟิก โอเพนซีแอล

* นักศึกษาหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยธรรมศาสตร์
5209035186@student.cs.tu.ac.th
** อาจารย์ประจำ ภาควิชาวิทยาการคอมพิวเตอร์คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์wdc@cs.tu.ac.th

Abstract

The objective of this research is to accelerate the C++11 for_each instruction performance by offloading the task from originally sequential exection on CPU to parallel execution on the Heterogeneous Computing System. The purposes are to increase the application computing performance and the productivity in application development.

The study method is by creating the usable project prototype including the directive instruction design to work with the for_each instruction and the compiler construction use for compiling the source code to be an executable program. The evaluation is by comparative experiment. The project program is compared with the standard C++ program and the hand-coding program on the Heterogeneous Computing System. The performance of computing and the productivity of programming are measured.

The results show that the research is able to accelerate the for_each instruction up to 203 times compared with the standard C++ and competitive with the hand-coding program on the Heterogeneous Computing System, while the source code is still short and not complicated as same as the standard C++.

Keywords: Compiler, C++, Parallel Computing, GPU, OpenCL

บทนำ

ในงานประมวลผลสมรรถนะสูง (High Performance Computing) และงานประมวลผลเชิงขนาน (Parallel Processing) มีการพัฒนาเทคโนโลยีของหน่วยประมวลผลอยู่ตลอดเวลา ทั้งนี้เพื่อเพิ่มความเร็วให้กับการประมวลผลงานขนาดใหญ่ รวมทั้งลดการใช้พลังงาน ในช่วงระยะเวลาไม่กี่ปีมานี้ เทคโนโลยีระบบประมวลผลร่วมหลายประเภท (Heterogeneous Computing System) ซึ่งเป็นการประยุกต์ใช้อุปกรณ์หลายชนิดในระบบเดียวกันมาช่วยประมวลผล เริ่มเป็นที่รู้จักและนำมาใช้กันอย่างแพร่หลายในหลายวงการและหลายอุตสาหกรรม เช่น การแพทย์ (Fabian Lecro, 2011) ดาราศาสตร์ (John M. Ford, 2010) และวิศวกรรมศาสตร์ (Hjelmervik, 2009) เนื่องจากสามารถเร่งความเร็วการคำนวณได้อย่างมหาศาล และปัจจุบันก็ได้มีผู้ออกแบบระบบประมวลผลร่วมหลายประเภทหลายระบบให้เลือกใช้ เช่น คูดา (CUDA) ของบริษัทเอ็นวิเดีย, ไดเร็คคอมพิวท์ (DirectCompute) ของบริษัทไมโครซอฟท์ และเฟรมเวิร์คโอเพนซีแอล (OpenCL) ของกลุ่มโครนอส อย่างไรก็ตาม ระบบประมวลผลร่วมหลายประเภทไม่ว่าจะเป็นการออกแบบของบริษัทใดมีจุดด้อยร่วมกันประการหนึ่ง นั่นคือ ขั้นตอนการพัฒนาโปรแกรมที่ค่อนข้างยุ่งยาก โปรแกรมเมอร์จำเป็นต้องมีความรู้ความเข้าใจในเรื่องสถาปัตยกรรมคอมพิวเตอร์ค่อนข้างมาก ส่งผลให้การพัฒนาโปรแกรมบนระบบประมวลผลร่วมหลายประเภทใช้เวลานาน ซับซ้อน เกิดข้อผิดพลาดได้ง่าย และต้นทุนสูง

จากอุปสรรคดังกล่าวของการพัฒนาโปรแกรมบนระบบประมวลผลร่วมหลายประเภท จึงได้เกิดแนวคิดที่จะนำภาษาโปรแกรมระดับสูงที่เป็นที่นิยม หรือที่โปรแกรมเมอร์คุ้นเคยดีอยู่แล้ว เช่น ภาษาซี (C), ซีพลัสพลัส (C++), ฟอร์แทรน (Fortran), ไพธอน (Python), ซีชาร์ป (C#) และจาวา (Java) มาประยุกต์ให้สามารถประมวลผลได้ด้วยระบบประมวลผลร่วมหลายประเภท นั่นคือเป็นการนำเอาข้อดีมาผสานใช้ร่วมกัน โปรแกรมเมอร์ผู้พัฒนาโปรแกรมยังคงใช้ภาษาระดับสูงที่คุ้นเคยในการเขียนโปรแกรม และจะได้โปรแกรมที่มีสมรรถนะสูงเพื่อใช้งานบนระบบประมวลผลร่วมหลายประเภท

สำหรับภาษาโปรแกรมระดับสูงนั้น ภาษาซีพลัสพลัสถือเป็นภาษาหนึ่งที่เป็นที่นิยมมาก เนื่องจากตัวภาษาเขียนโปรแกรมได้ทุกประเภท ทุกระบบ สนับสนุนการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) และมีการปรับปรุงภาษาให้ทันสมัยอยู่เสมอ ในปี ค.ศ. 2011 ภาษาซีพลัสพลัสได้มีการปรับปรุงมาตรฐานครั้งใหญ่เรียกว่าภาษาซีพลัสพลัส 11 (C++11) มีการเพิ่มไวยากรณ์ใหม่เพื่อรองรับการเขียนโปรแกรมยุคปัจจุบัน ไวยากรณ์หนึ่งที่น่าสนใจนั่นคือ นิพจน์แลมบ์ดา (Lambda Expression) ซึ่งเป็นกลุ่มของคำสั่งที่สามารถนำไปประยุกต์ใช้เป็นอากิวเมนต์, พารามิเตอร์ และวัตถุของฟังก์ชั่นได้ นิพจน์แลมบ์ดานั้นสามารถนำมาประยุกต์ใช้ร่วมกับคำสั่ง `for_each` ได้ โดยการทำงานจะเป็นการวนรอบทำงานตามที่ระบุไว้ในนิพจน์แลมบ์ดากับข้อมูลทุกตัวในชุด เช่น เวกเตอร์ (vector), คิว (queue) และลิสต์ (list) เป็นต้น จะเห็นได้ว่าการใช้คำสั่ง `for_each` ร่วมกับนิพจน์แลมบ์ดาคือรูปแบบการวนรอบ (Loop) ลักษณะหนึ่งที่สามารถเร่งความเร็วได้ด้วยการประมวลผลเชิงขนาน และความน่าสนใจของการใช้คำสั่ง `for_each` ร่วมกับนิพจน์แลมบ์ดา คือ เป็นการเขียนโปรแกรมยุคใหม่ที่โปรแกรมเมอร์สนใจงานที่กระทำกับโครงสร้างข้อมูล (Data Structure) มากกว่าการคำนึงถึงขั้นตอนการวนรอบ ดังเช่น ไวยากรณ์วนรอบแบบดั้งเดิม เช่น ไวยากรณ์ `for` และ `while`

ภาพที่ 1 ตัวอย่างการเรียกใช้คำสั่ง for_each ร่วมกับนิพจน์แลมบ์ดา

```
vector<int> vec;
for_each(vec.begin(), vec.end(), [](int &n) {
    n = n + 1;
});
```

งานวิจัยฉบับนี้จึงได้นำเสนอแนวคิดและวิธีการเร่งความเร็วการประมวลผลคำสั่ง for_each ร่วมกับนิพจน์แลมบ์ดาในโค้ดต้นฉบับภาษาซีพลัสพลัส 11 ให้ประมวลผลเชิงขนานบนระบบประมวลผลร่วมหลายประเภท

งานวิจัยและทฤษฎีที่เกี่ยวข้อง

มีงานวิจัยจำนวนมากที่เกี่ยวข้องกับการพยายามทำให้ภาษาโปรแกรมระดับสูงที่เป็นที่นิยมอย่าง ภาษาซี, ซีพลัส พลัส และไพธอน ให้มีความเร็วการในประมวลผลที่สูงขึ้น โดยการใช้ระบบประมวลผลร่วมหลายประเภท เช่น คูดา และโอเพนซีแอล มาช่วยเร่งการประมวลผล

เทียนยี เดวิด ฮาน และคณะ (2009) นำเสนอไฮคูดา (hiCUDA) เป็นคำสั่งกำกับของภาษาซีซึ่งใช้แทนการเขียนคูดาโดยตรง ทำให้โค้ดต้นฉบับสั้นลง และมีคอมไพเลอร์ต้นแบบทำหน้าที่คอมไพล์โค้ดให้เป็นโปรแกรมที่ประมวลผลโดยใช้คูดา ไฮคูดาได้รับการทดสอบโดยใช้ชุดวัดประสิทธิภาพอัลกอริทึมทางวิทยาศาสตร์ Parboil พบว่าประสิทธิภาพของโปรแกรมที่เขียนด้วยไฮคูดาเทียบเท่ากับโปรแกรมที่เขียนด้วยคูดาโดยตรง

ซียอง ลี และคณะ (2009) นำเสนอการแปลคำสั่งกำกับโอเพนเอ็มพี (OpenMP) ให้ประมวลผลอัตโนมัติบนคูดา คำสั่งเชิงขนานที่เดิมจะต้องประมวลผลเชิงขนานด้วยซีพียูหลายตัว คอมไพเลอร์ต้นแบบของงานวิจัยนี้จะย้ายงานไปประมวลผลบนคูดาแทน การทดสอบได้ใช้โปรแกรมวัดประสิทธิภาพ JACOBI, SPMUL, NAS Parallel Benchmark EP และ NAS Parallel Benchmark CG พบว่าโปรแกรมที่แปลจากโอเพนเอ็มพีเป็นคูดาประมวลผลได้เร็วขึ้น

พิสิษฐ์ มรรคไพสิฐ และคณะ (2011) นำเสนอกริฟฟอน (Griffon) เป็นคำสั่งกำกับของภาษาซีโดยจะแปลโค้ดลูป for ที่กำหนดไปประมวลผลเชิงขนานบนคูดาโดยอัตโนมัติ งานกริฟฟอนมีความคล้ายกับงานไฮคูดา แต่กริฟฟอนได้ปรับปรุงไวยากรณ์ของคำสั่งกำกับให้เขียนง่ายขึ้น และทำงานอัตโนมัติมากขึ้น

จิงกัว และคณะ (2011) นำเสนอแนวคิดให้การแปลโค้ดและการค้นหาโค้ดส่วนที่ทำงานเชิงขนานให้เป็นไปอย่างอัตโนมัติอย่างสมบูรณ์ จึงได้ออกแบบแซค (SaC: Single Assignment C) เป็นไวยากรณ์คล้ายภาษาซีและคอมไพเลอร์ช่วยแปลส่วนที่ทำงานเชิงขนานในโค้ดต้นฉบับไปประมวลผลบนคูดาโดยอัตโนมัติ

ไบรอัน คาตันซาโร และคณะ (2011) นำเสนอแนวคิดที่ว่านอกเหนือจากการทำให้โปรแกรมมีสมรรถนะสูงสุดแล้ว ความง่ายและความรวดเร็วในการเขียนโปรแกรมเพื่อใช้งานจริงก็เป็นสิ่งสำคัญ ภาษาไพธอนเป็นภาษาสคริปต์ที่เขียนได้เร็วกว่าภาษาซีหรือซีพลัสพลัสหลายเท่า จึงได้นำเสนอคอปเปอร์เฮด (Copperhead) ซึ่งเป็นโมดูลและตัวแปลภาษาไพธอน ทำหน้าที่แปลฟังก์ชั่นที่กำหนดให้ประมวลผลบนคูดาโดยอัตโนมัติ การออกแบบเป็น

แบบโมดูลมาตรฐานของภาษาไพธอนทำให้โปรแกรมเรียนรู้ง่าย และไม่ต้องมีความรู้เรื่องการเขียนโปรแกรมบนคูดา โค้ดที่เขียนด้วยคอปเปอร์เฮดเขียนได้สั้นกว่าโค้ดที่เขียนด้วยคูดาโดยตรงประมาณ 4 เท่า

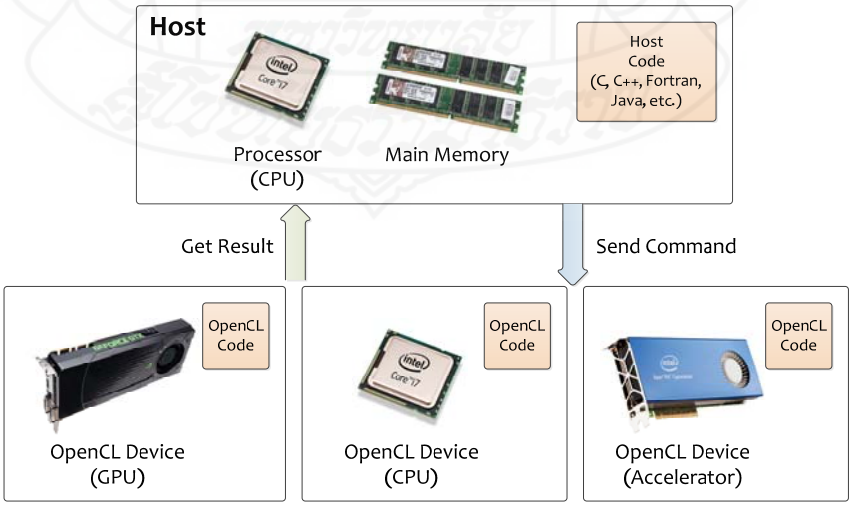
ระบบประมวลผลร่วมหลายประเภทด้วยเฟรมเวิร์คโอเพนซีแอล

ระบบประมวลผลร่วมหลายประเภท คือ ระบบคอมพิวเตอร์ที่ใช้อุปกรณ์มากกว่าหนึ่งประเภทในการประมวลผลโปรแกรมเดียว ตัวอย่างเช่น หน่วยประมวลผลกลาง (ซีพียู, CPU), หน่วยประมวลผลกราฟิก (จีพียู, GPU) และอุปกรณ์เร่งการประมวลผลเฉพาะทาง (Accelerator) จุดประสงค์เพื่อต้องการเพิ่มสมรรถนะในการประมวลผลงานโดยการใช้ทรัพยากรในระบบให้ได้คุ้มค่ามากที่สุด และใช้จุดเด่นของแต่ละอุปกรณ์ในการช่วยประมวลผลแต่ละส่วนของโปรแกรม เช่น หน่วยประมวลผลกลางสามารถคำนวณงานทั่วไปของโปรแกรมได้ดี ในขณะที่หน่วยประมวลผลกราฟิกสามารถคำนวณข้อมูลที่มีลักษณะเป็นชุด และมีปริมาณมหาศาลได้อย่างรวดเร็ว จึงสามารถนำมาประยุกต์ใช้กับงานประเภทการประมวลผลสมรรถนะสูงและการประมวลผลเชิงขนานได้เป็นอย่างดี

โอเพนซีแอลของกลุ่มโครนอส เป็นหนึ่งในเฟรมเวิร์กของระบบประมวลผลร่วมหลายประเภท ซึ่งในระยะหลังมานี้โอเพนซีแอลถือว่าโดดเด่นขึ้นมาก เนื่องด้วยประสิทธิภาพที่พัฒนาขึ้น และจุดเด่นที่เห็นได้ชัดคือการเป็นมาตรฐานเปิด สนับสนุนอุปกรณ์หลายชนิดและหลายยี่ห้อ ช่วยให้การพัฒนาโปรแกรมทำได้หลากหลายและยืดหยุ่น

การทำงานของโอเพนซีแอลเป็นไปดังภาพที่ 2 กล่าวคือ โอเพนซีแอลประกอบด้วยส่วนโฮส คือ ระบบส่วนกลางของคอมพิวเตอร์ เช่น ซีพียู, หน่วยความจำ, เมนบอร์ด และฮาร์ดดิสก์ และส่วนอุปกรณ์โอเพนซีแอล คือ อุปกรณ์ที่ใช้ประมวลผลโปรแกรมเชิงขนาน เช่น จีพียู และชิปเร่งความเร็ว การทำงานเริ่มจากโปรแกรมส่วนโฮสทำงาน เมื่อถึงจุดที่ระบุให้ทำงานเชิงขนานบนโอเพนซีแอล โฮสจะส่งคำสั่ง, โปรแกรมโอเพนซีแอล และข้อมูลที่ต้องการคำนวณ ไปยังอุปกรณ์โอเพนซีแอล เมื่ออุปกรณ์ได้รับคำสั่งก็จะประมวลผลตามโปรแกรมโอเพนซีแอลนั้น และส่งข้อมูลผลลัพธ์กลับมายังโฮสให้นำไปใช้งานได้ต่อไป

ภาพที่ 2 การทำงานของโอเพนซีแอล



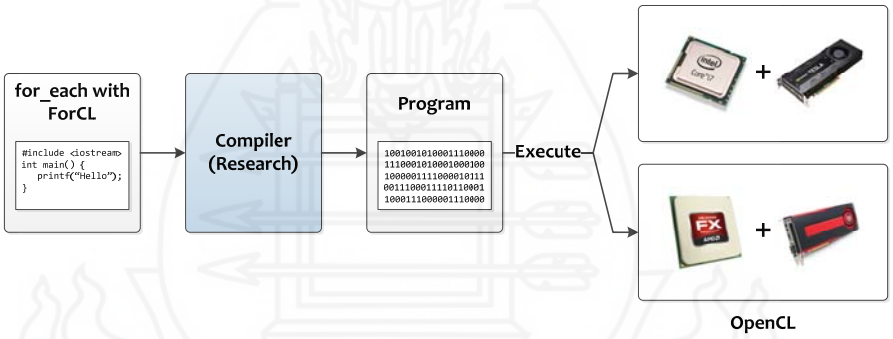
วัตถุประสงค์การวิจัย

1. เพื่อเพิ่มสมรรถนะ (Performance) ของคำสั่ง `for_each` ในภาษาซีพลัสพลัส 11 โดยการพัฒนาคอมไพเลอร์ เพื่อใช้คอมไพล์โค้ดต้นฉบับให้เป็นโปรแกรมที่ประมวลผลด้วยระบบประมวลผลร่วมหลายประเภท
2. เพื่อเพิ่มผลิตภาพ (Productivity) ของการเขียนโปรแกรมประมวลผลสมรรถสูง โดยการปรับปรุงภาษาซีพลัส พลัส 11 ให้มีคำสั่งกำกับที่คำสั่ง `for_each` ที่สามารถประมวลผลเชิงขนานได้โดยอัตโนมัติ

วิธีดำเนินการวิจัย

ในงานวิจัยฉบับนี้ผู้วิจัยได้พัฒนาโครงงานต้นแบบที่ใช้งานได้จริง โดยได้พัฒนาคอมไพเลอร์ต้นแบบเพื่อใช้ในการคอมไพล์โค้ดต้นฉบับภาษาซีพลัสพลัสที่มีการเรียกใช้คำสั่ง `for_each` ร่วมกับนิพจน์แลมบ์ดาให้ออกมาเป็นโปรแกรมที่สามารถประมวลผลเชิงขนานบนระบบประมวลผลร่วมหลายประเภทโดยใช้เฟรมเวิร์คโอเพนซีแอล

ภาพที่ 3 ภาพรวมของโครงงานวิจัย



การดำเนินงานแบ่งเป็น 3 ส่วน ได้แก่

1. การออกแบบไวยากรณ์ภาษา งานวิจัยนำเสนอคำสั่งกำกับ (Directive) ของภาษาซีพลัสพลัสชื่อ “ฟอร์ซีแอล” (ForCL) คำสั่งกำกับนี้เป็นตัวบ่งบอกให้คอมไพเลอร์ทราบว่าให้คอมไพล์การเรียกใช้คำสั่ง `for_each` ที่จุดนี้ให้ประมวลผลเชิงขนานบนโอเพนซีแอล กำหนดไวยากรณ์อยู่ในรูปแบบดังนี้

```
#pragma forcl command newline
for_each_lambda_call
```

โดยที่

- `forcl` คือ ชื่อกลุ่มของคำสั่งกำกับ (ในงานวิจัยกำหนดให้ชื่อ forcl)
- *`command`* คือ ชื่อคำสั่งกำกับ
- *`newline`* คือ อักขระขึ้นบรรทัดใหม่

- *for_each_lambda_call* คือ คำสั่งการเรียกใช้คำสั่ง for_each ร่วมกับนิพจน์แลมบ์ดาตามมาตรฐานของภาษาซีพลัสพลัส 11

โค้ดต้นฉบับตัวอย่างแสดงในภาพที่ 4 เป็นการเรียกใช้คำสั่ง for_each ในการคำนวณค่า sine ของข้อมูลทุกตัวที่อยู่ในเวกเตอร์ และมีคำสั่งกำกับ #pragma forcl parallel ระบุให้คำสั่ง for_each ดังกล่าวนี้ประมวลผลเชิงขนานบนโอเพนซีแอล

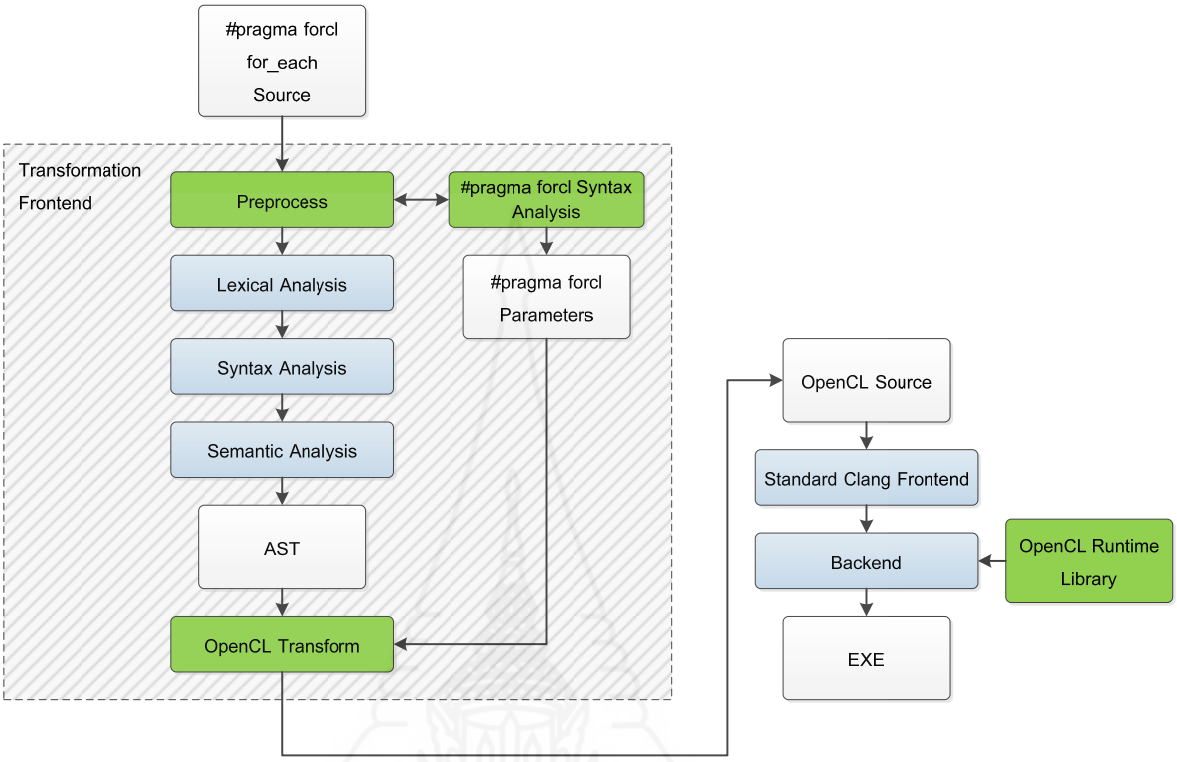
ภาพที่ 4 ตัวอย่างโค้ดต้นฉบับการเรียกใช้คำสั่ง for_each ที่มีคำสั่งกำกับฟอร์ซีแอล

```
vector<float> vec;
#pragma forcl parallel
for_each(vec.begin(), vec.end(), [](float &n) {
    n = sin(n);
});
```

2. การสร้างคอมไพเลอร์ คอมไพเลอร์มีหน้าที่แปลโค้ดต้นฉบับให้เป็นโปรแกรมที่สามารถใช้งานได้ สำหรับคอมไพเลอร์งานวิจัยฉบับนี้ได้สร้างด้วยไลบรารี่แอลแอลวีเอ็มและแคลง (LLVM & Clang) ซึ่งนอกเหนือจากการคอมไพล์โค้ดต้นฉบับภาษาซีพลัสพลัสตามมาตรฐานได้แล้ว ยังได้เพิ่มส่วนการแปลคำสั่งกำกับฟอร์ซีแอลและคำสั่ง for_each ให้เป็นโปรแกรมโอเพนซีแอลโดยอัตโนมัติ สำหรับขั้นตอนการคอมไพล์ของคอมไพเลอร์แสดงในภาพที่ 5 ซึ่งอธิบายได้ดังนี้

1) คอมไพเลอร์ส่วนหน้าค้นหาคำสั่งกำกับฟอร์ซีแอลที่มีอยู่ในโค้ดต้นฉบับ
2) วิเคราะห์ไวยากรณ์ของคำสั่งกำกับฟอร์ซีแอล เช่น ชื่อคำสั่ง, ตำแหน่งของคำสั่ง for_each
3) วิเคราะห์ไวยากรณ์ของคำสั่ง for_each จากโครงสร้างต้นไม้ (AST: Abstract Syntax Tree) เช่น ชื่อตัวแปร, ชนิดของตัวแปร, พารามิเตอร์ของนิพจน์แลมบ์ดา และแคปเจอร์ของนิพจน์แลมบ์ดา เป็นต้น
4) แปลงรูปคำสั่ง for_each ให้เป็นโค้ดต้นฉบับโอเพนซีแอลที่ยังคงทำงานได้เหมือนเดิม โดยอาศัยข้อมูลที่ได้จากการวิเคราะห์ไวยากรณ์ของคำสั่งกำกับและคำสั่ง for_each
5) นำโค้ดต้นฉบับโอเพนซีแอลที่แปลงรูปแล้วไปคอมไพล์ตามวิธีคอมไพล์โค้ดต้นฉบับภาษาซีพลัสพลัสมาตรฐาน ทำการลิงก์เข้ากับไลบรารี่โอเพนซีแอล ผลลัพธ์จะได้โปรแกรมที่สามารถใช้งานได้

ภาพที่ 5 ขั้นตอนการคอมไพล์



3. การทดลองและประเมินผล การประเมินผลงานวิจัยใช้การทดลองเชิงเปรียบเทียบระหว่าง 3 กลุ่มงาน ได้แก่

1) โปรแกรมที่ใช้คำสั่งกำกับฟอร์ซีแอลและฟังก์ชั่น `for_each` ที่คอมไพล์ด้วยคอมไพล์เลอร์งานวิจัย งานกลุ่มนี้โปรแกรมที่ได้จะประมวลผลด้วยระบบประมวลผลร่วมหลายประเภทโดยใช้เฟรมเวิร์คโอเพนซีแอล ซึ่งได้ทดลองทั้งโดยใช้อุปกรณ์ประเภทซีพียูและจีพียู

2) โปรแกรม `for_each` มาตรฐานของภาษาซีพลัสพลัส 11 คอมไพล์ด้วยคอมไพล์เลอร์จีซีซี (GCC) มาตรฐาน งานกลุ่มนี้โปรแกรมที่ได้จะประมวลผลเชิงลำดับบนซีพียู นอกจากนี้งานกลุ่มนี้ให้ถือเป็นฐานของการคำนวณค่าสปีดอัพ (Speedup) เปรียบเทียบกับการทดลองกลุ่มอื่นๆ

3) โปรแกรมที่เขียนด้วยโอเพนซีแอลโดยตรง งานกลุ่มนี้โปรแกรมที่ได้จะประมวลผลด้วยระบบประมวลผลร่วมหลายประเภทโดยใช้เฟรมเวิร์คโอเพนซีแอล ซึ่งได้ทดลองโดยใช้อุปกรณ์ประเภทซีพียูและจีพียู

เกณฑ์การประเมินกำหนด 2 เกณฑ์ ได้แก่

1) สมรรถนะ (Performance) ของโปรแกรมที่ได้ การประเมินสมรรถนะใช้การวัดเวลาในการประมวลผลข้อมูลทดสอบชุดเดียวกัน อัลกอรึทึมเดียวกัน และบนระบบคอมพิวเตอร์เดียวกัน

2) ผลิตภาพ (Productivity) ของการเขียนโปรแกรม บ่งบอกถึงความรวดเร็วและความง่ายในการเขียนโปรแกรม ในขณะที่ได้โปรแกรมที่มีสมรรถนะสูงและมีข้อผิดพลาดน้อย การประเมินใช้จำนวนบรรทัดของโค้ดต้นฉบับ โดยกำหนดให้ใช้อัลกอรึทึมเดียวกัน และรูปแบบการเขียนโปรแกรม (Code Convention) แบบเดียวกัน

ขั้นตอนการทดลอง ในด้านสมรรถนะกำหนดให้งานทั้ง 3 กลุ่ม เขียนโปรแกรมเพื่อคำนวณค่า sine ของจำนวนทุกตัวที่อยู่ในเวกเตอร์ และเขียนโปรแกรมจับเวลาการคำนวณ แล้วคอมไพล์ด้วยคอมไพเลอร์ของแต่ละงานได้เป็นโปรแกรมที่รันได้จริง จากนั้นรันโปรแกรม 10 ครั้ง บันทึกผลการวัดเวลา คำนวนหาค่าเฉลี่ย และค่าสปีดอัพ

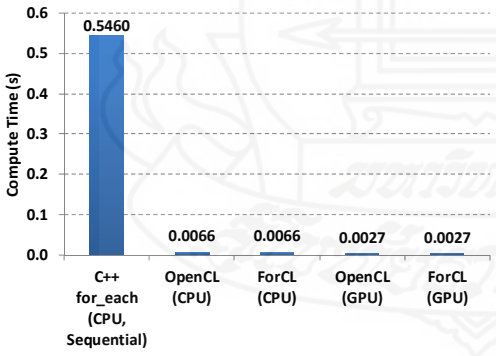
ในด้านผลิตภาพ ใช้โปรแกรมนับบรรทัด (CLOC: Count Lines of Code) นับจำนวนบรรทัดของโค้ดต้นฉบับในงานทั้ง 3 กลุ่ม และนำมาเปรียบเทียบ

ผลการวิจัย

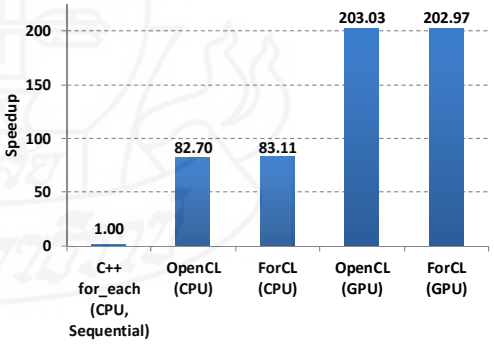
การทดลองเพื่อเปรียบเทียบสมรรถนะ ได้ทดลองบนระบบคอมพิวเตอร์ที่มีคุณสมบัติ คือ ซีพียู Intel Core i7 860 ความถี่ 2.80 GHz (ตัวประมวลผล 4 ตัว/8 เธรด), หน่วยความจำหลักขนาด 4 GB, จีพียู NVIDIA GeForce GTX 295 ความถี่ 576 MHz หน่วยความจำกราฟิก 896 MB (ตัวประมวลผล 240 ตัว) และใช้ระบบปฏิบัติการ Microsoft Windows 7 Service Pack 1 ผลการทดลองโดยใช้ข้อมูลเวกเตอร์ขนาด 10 ล้านตัว พบว่า โปรแกรมที่เขียนด้วยคำสั่งกำกับฟอร์ซีแอล ประมวลผลได้เร็วกว่าโปรแกรมที่เขียนด้วยคำสั่ง `for_each` ในภาษาซีพลัสพลัสมาตรฐานประมาณ 203 เท่าเมื่อเลือกอุปกรณ์โอเพนซีแอลประเภทจีพียู และเร็วกว่าประมาณ 83 เท่าเมื่อเลือกอุปกรณ์ประเภทซีพียู และเมื่อเปรียบเทียบระหว่างโปรแกรมที่เขียนด้วยคำสั่งกำกับฟอร์ซีแอลกับโปรแกรมที่เขียนด้วยโอเพนซีแอลโดยตรงพบว่าประมวลผลได้เร็วเท่าเทียมกัน

นอกจากนี้ยังแสดงให้เห็นว่าในระบบประมวลผลร่วมหลายประเภท หากเลือกใช้อุปกรณ์ประเภทจีพียูจะให้สมรรถนะที่ดีกว่าใช้ซีพียูอย่างเห็นได้ชัด

ภาพที่ 6 เปรียบเทียบเวลาประมวลผล (น้อยกว่าดีกว่า)

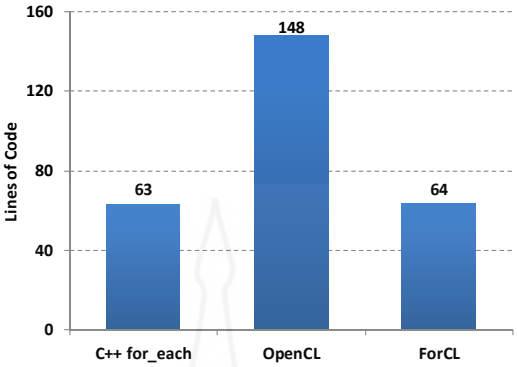


ภาพที่ 7 เปรียบเทียบ Speedup (มากกว่าดีกว่า)



ในด้านผลิตภาพพบว่าจำนวนบรรทัดของโค้ดต้นฉบับที่เขียนด้วยคำสั่งกำกับฟอร์ซีแอลมีจำนวนใกล้เคียงกับโค้ดต้นฉบับที่เขียนด้วยคำสั่ง `for_each` ภาษาซีพลัสพลัสมาตรฐาน และมีจำนวนน้อยกว่าครึ่งเมื่อเปรียบเทียบกับโค้ดต้นฉบับที่เขียนด้วยโอเพนซีแอลโดยตรง

ภาพที่ 8 เปรียบเทียบจำนวนบรรทัดของโค้ดต้นฉบับ (น้อยกว่าดีกว่า)



อภิปรายผลการวิจัย

จากผลการทดลองแสดงให้เห็นว่า กลุ่มงานวิจัยซึ่งก็คือ โค้ดต้นฉบับที่ใช้คำสั่ง `for_each` ร่วมกับการใช้คำสั่งกำกับฟอร์ซีแอล และคอมไพล์ด้วยคอมไพเลอร์ของงานวิจัย ให้ผลโปรแกรมที่มีสมรรถนะสูงกว่าการเขียนโปรแกรม `for_each` ในภาษาซีพลัสพลัสมาตรฐานอย่างเห็นได้ชัด เนื่องจากคอมไพเลอร์ในงานวิจัยได้ช่วยย้ายงานและการประมวลผลคำสั่ง `for_each` ไปประมวลผลเชิงขนานบนระบบประมวลผลร่วมหลายประเภทแทนการประมวลผลเชิงลำดับบนซีพียูโดยอัตโนมัติ ในขณะที่สมรรถนะนั้นดีเทียบเท่าโปรแกรมโอเพนซีแอลโดยตรง

ด้านมุมมองของโปรแกรมเมอร์ โปรแกรมเมอร์ยังคงเขียนโปรแกรมที่เรียกใช้คำสั่ง `for_each` ได้ง่ายและไม่ต่างจากภาษาซีพลัสพลัสมาตรฐานมากนัก โปรแกรมเมอร์ไม่จำเป็นต้องเขียนโปรแกรมโอเพนซีแอลโดยตรงซึ่งโค้ดมีขนาดใหญ่ ซับซ้อน และเกิดข้อผิดพลาดได้ง่ายกว่า และยังคงได้โปรแกรมที่มีสมรรถนะสูง

จากผลงานวิจัยฉบับนี้ก่อให้เกิดประโยชน์ดังนี้

1. เพิ่มสมรรถนะให้กับโปรแกรมภาษาซีพลัสพลัส ทำให้โปรแกรมประมวลผลได้รวดเร็วและคุ้มค่าทรัพยากรที่มีอยู่ในระบบมากขึ้น

2. ทำให้การเขียนโปรแกรมที่ต้องการสมรรถนะในด้านการประมวลผลสูง เขียนได้ง่ายขึ้น ส่งผลให้ลดต้นทุน และประหยัดเวลาในการพัฒนาโปรแกรม

3. สามารถนำงานไปต่อยอด เช่น ปรับปรุงไวยากรณ์ส่วนอื่นของภาษาซีพลัสพลัสหรือภาษาอื่นได้

ข้อเสนอแนะ

ผู้วิจัยยังคงดำเนินงานวิจัยต่อยอดจากงานวิจัยฉบับนี้ จากข้อจำกัดของงานวิจัยฉบับนี้ ได้แก่

1. คอมไพเลอร์สามารถแปลโค้ดที่อยู่ในนิพจน์แลมบ์ดาภายในคำสั่ง `for_each` ได้เฉพาะไวยากรณ์ที่ไม่ซับซ้อน เช่น ใช้ตัวดำเนินการพื้นฐาน (`+`, `-`, `*`, `/`, `%`, `=`, `+=`), เรียกใช้ฟังก์ชั่นพื้นฐาน (`sin`, `log`, `min`, `max`, `printf`), ไม่มีการเรียกใช้คำสั่ง for_each ซ้อนกันหลายชั้น และการคำนวนข้อมูลแต่ละตัวในเวกเตอร์ต้องไม่มีผลกระทบซึ่งกันและกัน (Loop Dependency)

2. คอมไพเลอร์ยังไม่สนับสนุนการแปลแคปเจอร์ของนิพจน์แลมบ์ดา

ในงานวิจัยฉบับต่อไปผู้วิจัยจะเพิ่มเติมความสามารถในส่วนนี้ และจะทำให้สามารถแปลโค้ดอัลกอริทึมทางวิทยาศสาตร์ที่ซับซ้อนขึ้นได้ เช่น Search, Sum, Matrix Multiplication, Fast Fourier transform และอื่นๆ

เอกสารอ้างอิง

Catanzaro, B., Garland, M., and Keutzer, K. (2011). *Copperhead: Compiling an Embedded Data Parallel Language*. PPoPP '11 Proceedings of the 16th ACM symposium on Principles and practice of parallel programming, February 12-16, 2011, San Antonio, Texas, USA, 47-56.

Danial, A. (2014). *CLOC: Count Lines of Code* [โปรแกรมคอมพิวเตอร์]. Retrieved September 9, 2014, from http://cloc.sourceforge.net

Ford, J. M., Demorest, P., and Ransom, S. (2010). *Heterogeneous real-time computing in radio astronomy*. Proc. SPIE 7740, Software and Cyberinfrastructure for Astronomy, 77400A.

Guo, J., Thiyagalingam, J., and Scholz. S. B. (2011). *Breaking the GPU Programming Barrier with the Auto-Parallelising SAC Compiler*. DAMP '11 Proceedings of the sixth workshop on Declarative aspects of multicore programming, January 23, 2011, Austin, Texas, USA, 15-24.

Han, T. D., and Abdelrahman, T. S. (2009). *hiCUDA: A High-level Directive-based Language for GPU Programming.* GPGPU-2 Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, March 8, 2009, Washington, D.C., USA, 52-61.

Hjelmervik, and Mikkelsen, J. (2009). *Heterogeneous Computing with Focus on Mechanical Engineering*. (Doctoral thesis). University of Oslo, Faculty of Mathematics and Natural Sciences.

ISO/IEC. (2012). *Working Draft, Standard for Programming Language C++, 5.1.2 Lambda expressions*, N3337, 88-92.

ISO/IEC. (2012). *Working Draft, Standard for Programming Language C++, 25.2.4 For each*, N3337, 847.

Khronos OpenCL Working Group. (2011). *The OpenCL Specification Version 1.1 Document Revision 44*. Retrieved June 5, 2014, from http://www.khronos.org/opencl/

Lecron, F., Mahmoudi, S. A., Benjelloun, M., Mahmoudi, S., and Manneback, P. (2011). *Heterogeneous Computing for Vertebra Detection and Segmentation in X-Ray Images*. International Journal of Biomedical Imaging, 2011, Article ID 640208, 12 pages.

Lee, S., Min, S., and Eigenmann, R. (2009). *OpenMP to GPGPU: A Compiler Framework for Automatic Translation and Optimization*. PPoPP '09 Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming, February 14–18, 2009, Raleigh, North Carolina, USA, 101-110.

llvm.org. (2014). *Clang Documentation*. Retrieved June 5, 2014, from http://clang.llvm.org/docs/index.html

Makpaisit, P., and Marurngsith, W. (2011). *Griffon - GPU Programming APIs for Scientific and General Purpose Computing.* Advances in Intelligent and Soft Computing, 91, 175-182.