การประมวลผลแบบขนานของลิงค์เดต้าบนสถาปัตยกรรมแบบหลายแกน

โดย

นางสาวชิดชนก โชคสุชาติ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาปรัชญาดุษฎีบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์และสารสนเทศ

ภาควิชาคอมพิวเตอร์

บัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร

ปีการศึกษา 2558

การประมวลผลแบบขนานของลิงค์เดต้าบนสถาปัตยกรรมแบบหลายแกน

โดย

นางสาวชิดชนก โชคสุชาติ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาปรัชญาดุษฎีบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์และสารสนเทศ

ภาควิชาคอมพิวเตอร์

บัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร

ปีการศึกษา 2558

PARALLEL PROCESSING OF LINKED DATA ON MANY-CORE ARCHITECTURE

By

Miss Chidchanok Choksuchat

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree

Doctor of Philosophy Program in Computer and Information Science

Department of Computing

Graduate School, Silpakorn University

Academic Year 2015

บัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร อนุมัติให้วิทยานิพนธ์เรื่อง “การประมวลผลแบบขนานของลิงค์เดต้าบนสถาปัตยกรรมแบบหลายแกน” เสนอโดย นางสาวชิดชนก โชคสุชาติ เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาปรัชญาดุษฎีบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์และสารสนเทศ

……..
(รองศาสตราจารย์ ดร.ปานใจ ธารทัศนวงศ์)
คณบดีบัณฑิตวิทยาลัย
วันที่.....เดือน..... พ.ศ.....

อาจารย์ที่ปรึกษาวิทยานิพนธ์
รองศาสตราจารย์ ดร.จันทนา จันทราพรชัย

คณะกรรมการตรวจสอบวิทยานิพนธ์

..... ประธานกรรมการ
(อาจารย์ ดร.ทัศนวรรณ ศูนย์กลาง)
...../...../.....

..... กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.ภุชงค์ อุทโยภาศ)
...../...../.....

..... กรรมการ
(ดร.เทพชัย ทรัพย์นิธิ)
...../...../.....

..... กรรมการ
(รองศาสตราจารย์ ดร.ปานใจ ธารทัศนวงศ์)
...../...../.....

..... กรรมการ
(รองศาสตราจารย์ ดร.จันทนา จันทราพรชัย)
...../...../.....

54307801: สาขาวิชาวิทยาการคอมพิวเตอร์และสารสนเทศ
คำสำคัญ : การประมวลผลแบบขนาน/ลิงค์เดต้า/สถาปัตยกรรมแบบหลายแกน

ชิดชนก โชคสุชาติ: การประมวลผลแบบขนานของลิงค์เดต้าบนสถาปัตยกรรมแบบหลายแกน. อาจารย์ที่ปรึกษาวิทยานิพนธ์ : รศ.ดร.จันทนา จันทราพรชัย. 138 หน้า.

ในปัจจุบันเว็บเชิงความหมายได้ใช้รูปแบบข้อมูลอาร์ดีเอฟ (Resource Description Framework) ที่กำหนดโดยองค์กร W3C เป็นเมตาเดต้าสำหรับเก็บ เผยแพร่และเชื่อมข้อมูลระหว่างกันด้วยเทคโนโลยีลิงค์เดต้า ลักษณะของอาร์ดีเอฟมีหลายแบบตามการใช้งาน ในงานวิจัยนี้สนใจการค้นหาข้อมูลเชิงความหมายโดยใช้อาร์ดีเอฟรูปแบบทริพเพิลซึ่งเป็นเซตของประธาน กริยา กรรม มีการเก็บข้อมูลจำนวนหลายล้านทริพเพิลในรูปแบบไฟล์ข้อความทำให้ไฟล์มีขนาดใหญ่ขึ้น ในการคิวรี่ข้อมูลจากไฟล์ขนาดใหญ่นี้ด้วยโปรแกรมแบบเทรดเดียวบนเครื่องคอมพิวเตอร์ส่วนบุคคลจะเร็วไม่พอ จึงมักนำเซิร์ฟเวอร์ขนาดใหญ่และมีหลายแกนมาใช้ประมวลผลแบบขนาน ซึ่งมีค่าใช้จ่ายสูง งานวิจัยนี้จึงนำเสนอการประมวลผลคิวรี่ข้อมูลแบบอาร์ดีเอฟโดยใช้สถาปัตยกรรมแบบหลายแกนด้วยหน่วยประมวลกราฟิกส์บนเครื่องคอมพิวเตอร์ส่วนบุคคล ที่มีราคาไม่แพงและให้คำตอบได้ในเวลารวดเร็ว

เนื่องจากสถาปัตยกรรมของหน่วยประมวลผลกราฟิกส์นั้นมีหน่วยความจำภายในขนาดเล็ก จึงจำเป็นต้องแปลงข้อมูลอาร์ดีเอฟเพื่อลดขนาดข้อมูล ให้นำเข้าสู่หน่วยความจำได้จำนวนมาก เพื่อให้ทำงานค้นหาคุ้มค่าที่สุดในการนำข้อมูลเข้าแต่ละครั้ง ผู้วิจัยจึงนำเสนอการแปลงข้อมูลรูปแบบซีบีเอ็ม (CBM, Combined BitMap representation) ที่ลดขนาดข้อมูล 1.7 กิกะไบต์ จากรูปแบบทริพเพิลได้ถึง 93% เมื่อนำไปค้นหาตามคิวรี่บนหน่วยประมวลผลกราฟิกส์รุ่น Tesla K40c พบว่าสามารถเร่งความเร็วได้มากกว่าการประมวลผลแบบลำดับถึง 13,000-27,000 เท่า เนื่องจากลักษณะเฉพาะของข้อมูลอาร์ดีเอฟเพื่อการค้นหา มีการเปลี่ยนแปลงน้อย เน้นที่การอ่านอย่างเดียว จากประเด็นการใช้เวลาแปลงข้อมูลเพื่อลดขนาดหลายชั่วโมง สามารถพัฒนาไปสู่การตั้งเวลาแปลงข้อมูลเพื่อเตรียมไว้ใช้ในการค้นหาได้ ทั้งนี้ยังมีการทดลองค้นหาโดยตรงด้วยวิธีเปรียบเทียบสตริงจากข้อมูลไฟล์ทริพเพิลแบบบรูซฟอร์ทแบบลำดับและแบบมัลติเทรดบนหน่วยประมวลผลกลางและบนหน่วยประมวลผลกราฟิกส์ ด้วยการใช้ประโยชน์ของหน่วยความจำพิน (โฮสต์) หน่วยความจำโกลบอล แชร์ และโอเปอเรชันสตรีม บนหน่วยประมวลผลกราฟิกส์หลายใบ ทั้งนี้ไฟล์ขนาดใหญ่สุดที่ใช้ทดสอบมีขนาด 400 กิกะไบต์หรือ 2.86 X 10^9 ทริพเพิล นอกจากนี้ยังมีการประยุกต์ใช้การประมวลผลแบบขนานกับฐานข้อมูลแบบคีย์-ค่า ภาษาจาวา และในส่วนการดึงข้อมูลจากเว็บด้วยวิธีแมพรีดิวซ์

ภาควิชาคอมพิวเตอร์ บัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร
ลายมือชื่อนักศึกษา..... ปีการศึกษา 2558
ลายมือชื่ออาจารย์ที่ปรึกษาวิทยานิพนธ์ .....

54307801: MAJOR: COMPUTER AND INFORMATION SCIENCE
KEYWORD: PARALLEL COMPUTING/LINKED DATA/MANYCORE ARCHITECTURE
CHIDCHANK CHOKSUCHAT: PARALLEL PROCESSING OF LINKED DATA ON MANY-CORE ARCHITECTURE. THESIS ADVISOR: ASSOC.PROF. CHANTANA CHANTRAPONCHAI, Ph.D. 138 pp.

Resource Description Framework (RDF) is the commonly used format for Semantic Web data. Nowadays, huge amounts of data on the Internet in the RDF format are used by search engines for providing answers to the queries of users. Querying through big data needs suitable searching methods supported by a very high processing power, because the traditional, sequential keyword matching on a semantic web server may take a prohibitively long time. In this research, we aim at accelerating the search in big RDF data by exploiting modern many-core architectures based on Graphics Processing Units (GPUs). While GPUs have become inexpensive processors that can be used for general purpose computing, the restrictions of the GPU memory hierarchy prevent a direct use of GPUs for processing big data: 1) data transfers between the GPU (device) memory and CPU (host) may incur a high run time overhead, 2) the size of GPU memory is not large enough to hold big amounts of data. Hence, in this research, we aim to propose a software engineering solution for the search and management of the related data transfers, taking into account the constrained memory of GPU architectures. Our method is general enough and can be applied to search any large text data, but we specifically focus on the parallel search for the RDF data set, because it is currently the most common format for semantic web applications.

We present two representation frameworks. First, method with preprocessing transforms RDF data set to Combined BitMap representation (CBM) for compact data and convenient search. Since GPUs have limited memory size, without compaction, the RDF data may not be entirely stored in the GPU memory; thus, using the CBM structure enables us to put more RDF data in the GPU memory. Since GPUs contain many processing elements, utilizing them concurrently will speed up the RDF query processing. The experimental results show that the proposed representation can reduce the size of original RDF data from 1.7 GB by 93 percent. While query on NVIDIA Tesla K40c, the accelerating of search around 13,000-27,000 times compare to the serial search version. According the long time consuming of preprocessing method, the RDF data set rarely updated, and, hence, we develop to schedule batch file for preparation the compact data before searching through GPUs in the future. Second, we also develop several implementations of the RDF search for many-core architectures using two programming approaches: OpenMP for systems with CPUs and CUDA for systems comprising CPUs and GPUs. We implement the directed search on multiple GPUs by brute-force string matching method. We use the global memory (device), shared memory (device), pinned memory (host) and stream operation (device) on multiple GPUs. The maximum size of RDF data set is 400 GB or 2.68 x 10^9triples.

In addition, we experiment to search the key-value storage with parallel engine, applying Java library to parallelize with multithread version; Parallel Java, and extract web sites of Hua Hin Health Tourism domain by Java Concurrency and MapReduce.

Department of Computing
Student's signature .....
Thesis Advisor's signature .....

Graduate School, Silpakorn University
Academic Year 2015

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลงด้วยความกรุณาเป็นอย่างสูงของรองศาสตราจารย์ ดร.จันทนา จันทราพรชัย อาจารย์ที่ปรึกษาวิทยานิพนธ์ของข้าพเจ้า ที่กรุณาให้โอกาสและความรู้ทั้งหลายมากกว่าด้านการเรียน ให้คำปรึกษาที่นำมาประยุกต์ใช้ได้ในชีวิตจริง ให้กำลังใจและแก้ปัญหาให้ข้าพเจ้ามาโดยตลอด สิ่งต่าง ๆ ที่ข้าพเจ้าทำผิดพลาดไป ข้าพเจ้ากราบขออภัยและกราบขอบพระคุณอาจารย์เป็นอย่างสูงไว้ ณ ที่นี้ด้วย

ขอขอบพระคุณ ผู้ช่วยศาสตรจารย์ ดร. ภุชงค์ อุทโยภาส ผู้ทรงคุณวุฒิ ที่กรุณาให้คำแนะนำในการวิจัย และดำรงชีวิตอย่างมีคุณธรรมและกรุณาให้ใช้ห้องปฏิบัติการศูนย์วิจัยระบบคอมพิวเตอร์สมรรถนะสูงและเครือข่ายคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ ม. เกษตรศาตร์

ขอขอบพระคุณทุนปริญญาเอกกาญจนาภิเษก ทุน DAAD (Deutscher Akademischer Austausch Dienst) กองทุนสนับสนุนการวิจัย ด้านการท่องเที่ยว และทุนจากบัณฑิตวิทยาลัย ม.ศิลปากร

Danke Schön an Prof. Sergei Gorlatch, Michael Haidl und Julia Kaiser-Mariani für die Zeit die sie sich genommen haben um meine Veröffentlichungen zu kommentieren.

Vielen Dank an alle Menschen in Arbeitsgruppe Parallele und Verteilte Systeme die mir an der Universität Münster geholfen haben.

ขอขอบพระคุณ ดร.เทพชัย ทรัพย์นิธิ ผู้ทรงคุณวุฒิ และ ดร.ทัศนวรรณ ศูนย์กลาง ประธานกรรมการสอบที่ช่วยแนะนำแนวทางการทำวิทยานิพนธ์ให้มีความถูกต้องและสมบูรณ์มากขึ้น

ขอขอบพระคุณรองศาสตราจารย์ ดร.ปานใจ ธารทัศนวงศ์ ผู้ช่วยศาสตราจารย์สุจิตรา อดุลย์เกษม และคณาจารย์ประจำภาควิชาคอมพิวเตอร์ มหาวิทยาลัยศิลปากรทุกท่านที่กรุณาประสิทธิ์ประสาทความรู้และประสบการณ์อันมีค่ายิ่งแก่ศิษย์

ขอขอบพระคุณ รศ. ดร. บุษบา ฤกษ์อำนวยโชค และคณาจารย์จากห้องปฏิบัติการมนุษย์พันธุศาสตร์ โรงพยาบาลรามาธิบดี มหาวิทยาลัยมหิดล ที่ให้โอกาสและประสบการณ์ในการวิจัย

ขอกราบขอบพระคุณ คุณพ่อ คุณแม่และครอบครัวที่เป็นกำลังใจให้เสมอมา

ขอขอบพระคุณอาจารย์ทุกท่านที่พบในงานประชุมวิชาการนานาชาติ ท่านกรุณาให้คำแนะนำเพื่อนำมาปรับปรุงวิทยานิพนธ์ให้ถูกต้องมากขึ้นระหว่างการทำวิจัย

ขอขอบพระคุณคณะกรรมการจากวารสารระดับชาติและระดับนานาชาติ ที่กรุณาสละเวลาอ่านงาน และให้คำแนะนำที่เป็นประโยชน์ เพื่อพัฒนางานวิจัย

ขอขอบคุณพี่ประวิม เหลืองสมานกุลและพี่กัลยา ตาทองที่ให้คำแนะนำและช่วยเหลือในการประสานงานตลอดการทำวิทยานิพนธ์นี้

ขอขอบคุณพี่ เพื่อนและน้องนักศึกษาจากมหาวิทยาลัยเกษตรศาสตร์และมหาวิทยาลัยศิลปา-กรทุกท่านที่คอยเป็นกำลังใจและช่วยเหลือข้าพเจ้าในการทำวิจัย

สารบัญ

หน้า

สารบัญตาราง

ตารางผนวก หน้า

สารบัญรูป

รูปที่ หน้า

รูปที่ หน้า

รูปที่ หน้า

รูปผนวก หน้า

รูปผนวก หน้า

บทที่ 1

บทนำ

ในปัจจุบันเว็บไซต์จำนวนมากได้เผยแพร่ข้อมูลอย่างอิสระในรูปแบบเว็บข้อมูล (web of data) ที่ถือว่าเป็นฐานข้อมูลเปิดประเภทหนึ่งที่มีองค์กรเวิลด์ไวด์เว็บ (W3C) เป็นผู้กำหนดมาตรฐานของข้อมูลภายใน ส่งผลให้ข้อมูลเว็บที่เผยแพร่มีรูปแบบที่สามารถแปลงให้เครื่องคอมพิวเตอร์สามารถประมวลผลความหมายและเข้าใจได้ในระดับหนึ่ง ถือว่าเป็นประโยชน์จากเว็บเชิงความหมาย (Semantic Web) ซึ่งสามารถช่วยงานมนุษย์ได้มากขึ้น ดังเห็นได้อย่างชัดเจนในการทำงานของเครื่องมือค้นหาเว็บไซต์ (search engine tool) โดยคอมพิวเตอร์จะช่วยกรองข้อมูลให้ตรงความต้องการของมนุษย์มากขึ้น ส่งผลให้ผลลัพธ์ของการค้นหาเว็บไซต์ที่เกี่ยวข้องลดลง ตรงกับความต้องการของมนุษย์มากขึ้น โดยพิจารณาเซตข้อมูลที่มีลักษณะของออนโทโลยี (Ontology) ที่มีลักษณะเด่นในการสื่อความหมายหลายระดับ ข้อมูลเหล่านี้จะได้รับการเผยแพร่จากเทคโนโลยีลิงค์เดต้า (Linked Data) ที่ภายในยอมให้มีการเชื่อมต่อด้วยความสัมพันธ์ระหว่างเซตข้อมูลเพื่อขยายขอบเขตไปยังเซตข้อมูลที่เกี่ยวข้องกัน ทำให้เซตข้อมูลเหล่านี้มีขนาดใหญ่ (Big Data) มากขึ้น ระดับหลายร้อยถึงหมื่นกิกะไบต์ เมื่อผู้ใช้ค้นหาข้อมูลผ่านเครื่องมือช่วยค้นหาสำหรับเว็บเชิงความหมาย (semantic web search engine) จะสามารถหาผลลัพธ์ที่สัมพันธ์กันจากเซตข้อมูลที่เกี่ยวข้องได้ ในกรณีนี้ประเด็นที่เกิดขึ้นคือการสร้างโปรแกรมแบบลำดับเพื่อค้นหาจากเซตข้อมูลขนาดใหญ่เหล่านี้ จะใช้เวลาในการค้นหาผลลัพธ์นาน ดังนั้นจึงนำเทคโนโลยีการประมวลผลแบบขนานเข้ามาใช้ทำให้การค้นหารวดเร็วขึ้น ซึ่งเริ่มแรกโปรแกรมแบบขนานนี้ใช้ทำงานบนเครื่องซูเปอร์คอมพิวเตอร์ที่เป็นเครื่องที่มีหลายหน่วยประมวลผลภายใน ต่อมาจึงมีการประมวลผลระหว่างเครื่องคอมพิวเตอร์ ซึ่งแต่ละครั้งต้องใช้ทรัพยากรมากมายและมีราคาสูง ผู้วิจัยจึงสนใจการโปรแกรมแบบขนานบนเครื่องคอมพิวเตอร์ส่วนบุคคล ซึ่งเป็นแนวคิดในการประหยัดทรัพยากรและผู้ใช้ทั่วไปสามารถเข้าถึงการใช้งานค้นหาข้อมูลขนาดใหญ่ได้

สำหรับการโปรแกรมแบบขนานในช่วงแรกมีการพัฒนาแบบมัลติเทรดบนหน่วยประมวลผลกลางแบบหลายแกน (multicore CPU) ที่สามารถทำการประมวลผลได้ทั้งแบบลำดับและแบบขนาน ต่อมามีสถาปัตยกรรมแบบหลายแกนบนหน่วยประมวลผลกราฟิกส์ (many-core GPU) ที่เหมาะสมกับการทำงานแบบขนานที่มีประสิทธิภาพสูงที่สนับสนุนการทำงานด้านกราฟิกส์

โดยเฉพาะและได้มีการนำมาใช้เพื่อประมวลผลงานทั่วไปด้านวิทยาศาสตร์อย่างกว้างขวาง สถาปัตยกรรมดังกล่าวได้รับการพัฒนาหน่วยคำนวณมากขึ้นจนมีหลายพันแกน นั่นหมายถึงความสามารถในการประมวลผลหลายพันงานพร้อมกัน ทั้งยังมีราคาถูกลงจนกระทั่งผู้ใช้ทั่วไปสามารถเป็นเจ้าของได้ ดังนั้นผู้วิจัยเชื่อว่าผู้ใช้สามารถประมวลผลแบบขนานและสร้างผลลัพธ์ที่มากขึ้นได้ด้วยการใช้งานหน่วยประมวลผลกราฟิกส์ร่วมบนคอมพิวเตอร์ส่วนบุคคล อย่างไรก็ตามงานดังกล่าวมีประเด็นด้านการจัดการ เพื่อประสานการทำงานระหว่างหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกส์ ในด้านการรับส่งข้อมูลระหว่างกันอย่างเหมาะสมทำให้สามารถทำการค้นหาข้อมูลขนาดใหญ่บนพื้นที่หน่วยความจำแบบหลายระดับชั้นของหน่วยประมวลกราฟิกส์ที่ล้วนมีขนาดเล็กกว่าหน่วยความจำของหน่วยประมวลผลกลางและขนาดเล็กกว่าเซตข้อมูลได้ ดังนั้นผลของงานวิจัยนี้จะได้ตัวแบบเพื่อแก้ปัญหาการค้นหาข้อมูลเชิงความหมายขนาดใหญ่บนสถาปัตยกรรมแบบขนานดังกล่าว

ความสำคัญของงานวิจัย

การประมวลผลเพื่อคิวรี่ฐานความรู้บนหน่วยประมวลผลกราฟิกส์ที่มีประสิทธิภาพนั้น มีประเด็นที่ต้องพิจารณาได้แก่วิธีนำข้อมูลขนาดใหญ่เข้าประมวลผลในหน่วยประมวลผลกราฟิกส์ เนื่องจากพื้นที่ในหน่วยความจำของหน่วยประมวลผลกราฟิกส์นั้นมีขนาดเล็กกว่าหน่วยประมวลผลกลางมาก ทั้งยังแบ่งระดับชั้นของการเข้าถึงได้แตกต่างกัน นอกจากนั้นการประมวลผลต้องอาศัยการสั่งงานจากหน่วยประมวลผลกลางหรือโฮสต์ โดยส่งเซตข้อมูล คำสำคัญและคำสั่งผ่านพีซีไอ (PCI express) ที่เปรียบเสมือนท่อที่มีขนาดเล็กเชื่อมต่อระหว่างหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกส์หรือดีไวส์ ถือเป็นคอขวดในการรับส่งข้อมูล ดังนั้นเพื่อให้เป้าหมายคือการนำเซตข้อมูลขนาดใหญ่ไปประมวลผลบนหน่วยประมวลผลกราฟิกส์ได้อย่างมีประสิทธิภาพ จึงต้องมีการแปลงโครงสร้างข้อมูลให้เหมาะสมเพื่อนำข้อมูลเข้าสู่หน่วยความจำของหน่วยประมวลผลให้มากที่สุดเท่าที่หน่วยความจำนั้นจะรับได้ และมีการวนใช้ข้อมูลในหน่วยประมวลผลกราฟิกส์อย่างคุ้มค่า ต้องคำนึงถึงการรับส่งคำสั่งระหว่างโฮสต์และดีไวส์ การจัดการปริมาณพื้นที่ที่ใช้งาน ผลลัพธ์ที่ได้จากการค้นหาถูกส่งออกจากดีไวส์และนำส่วนต่อไปมาใช้ค้นหาในรอบใหม่จนกว่าจะหมดเซตข้อมูล การศึกษาดังกล่าวเป็นโอกาสที่สำคัญยิ่งในการศึกษาการประมวลผลแบบขนานกับข้อมูลประเภทออนโทโลยีเพื่อประยุกต์ใช้กับขอบเขตข้อมูลเฉพาะด้านในอนาคตได้อีกมากมาย

ที่มาของงานวิจัย

งานวิจัยนี้มีที่มาจากเซตข้อมูลเปิดสำหรับเว็บเชิงความหมายได้รับการเผยแพร่ออกมามากขึ้นจนมีขนาดใหญ่มาก โดยเซตข้อมูลนี้มีลักษณะเป็นไฟล์อักขระหลายชนิด เรียกหน่วยย่อยว่า

ทริพเพิล (triple) ประกอบด้วยประธาน (subject) ภาคการกระทำ (predicate) และกรรม (object) ซึ่งเนื้อหาสามารถสัมพันธ์กับเซตข้อมูลอื่นในไฟล์เดียวกันได้ โดยเซตของทริพเพิลที่มากที่สุดมีมากกว่าพันล้านทริพเพิล ซึ่งปริมาณข้อมูลดังกล่าวยังมีการเพิ่มขึ้นอยู่เสมอ และมีการขยายการใช้งานอย่างรวดเร็วจนอยู่ในรูปแบบของกลุ่มข้อมูลเปิดขนาดใหญ่ที่เชื่อมถึงกัน โดย W3C ได้ประกาศว่าเว็บเชิงความหมายเป็นเทคโนโลยีสำหรับอนาคต ที่ทำให้ใช้ข้อมูลเชื่อมโยงระหว่างกัน ทั้งยังช่วยมนุษย์วิเคราะห์ข้อมูลที่ค้นหาได้ โดยพิจารณาจากลักษณะสำคัญประการหนึ่งของเว็บเหล่านี้ นั่นคือภายในโครงสร้างแต่ละเว็บนั้นมีการกำกับความหมาย (annotation) เพื่อใช้ประกอบการค้นหาข้อมูลให้ได้ผลลัพธ์ที่เกี่ยวข้องกับขอบเขตที่ผู้ใช้สนใจมากที่สุด ด้วยความน่าเชื่อถือในระดับสูง เมื่อใช้ประกอบกับข้อมูลที่เชื่อมต่อกันระหว่างเว็บก็จะทำให้สามารถค้นหาข้อมูลที่สัมพันธ์ระหว่างกัน ส่งผลให้ขยายขอบเขตที่สนใจได้มากขึ้น นอกจากนี้ทิม เบิร์นเนอร์ส-ลี ผู้ก่อตั้งเวิลด์ไวด์เว็บ (WWW) ได้แบ่งข้อมูลเปิดที่เผยแพร่ได้ตามระดับความสามารถในการสื่อความหมายตามบริบทของเว็บ ที่นำมาใช้วิเคราะห์ตามสาขาหรือขอบเขตข้อมูล ซึ่งแหล่งข้อมูลดังกล่าวมีการเก็บและใช้งานกับออนโทโลยีอย่างแพร่หลาย ตัวอย่างเช่น รูปแบบการบรรยายทรัพยากร (Resource Description Framework: RDF) ในหลายสาขา เช่น การเก็บข้อมูลจากกลุ่มเครือข่ายสังคมออนไลน์ ข้อมูลด้านการท่องเที่ยว ศัพท์ทางการแพทย์ ข้อมูลจากวิกิพีเดีย ข้อมูลภูมิสารสนเทศ พจนานุกรมภาษาอังกฤษ และประยุกต์ใช้เก็บข้อมูลที่ได้จากเทคโนโลยีลิงค์เดต้าจนกลายเป็นชุดข้อมูลเปิดที่เชื่อมต่อถึงกัน (Linked Open Data) แล้ว เช่น ข้อมูลการเชื่อมวิกิพีเดียและพจนานุกรม เป็นต้น จากที่กล่าวมาข้างต้นนั้นส่วนใหญ่มีหลายงานวิจัยที่นำข้อมูลเหล่านี้มาใช้ในการค้นหาข้อมูลบนเครื่องขนาดใหญ่ระหว่างเครือข่ายซึ่งต้องควบคุมปัจจัยด้านเครือข่ายและเสียค่าใช้จ่ายจำนวนมาก ดังนั้นผู้วิจัยจึงมีแนวคิดนำเซตข้อมูลขนาดใหญ่นี้มาประมวลผลแบบขนานบนหน่วยประมวลผลแบบหลายแกนทั้งระดับหน่วยประมวลผลกลาง และหน่วยประมวลผลกราฟิกส์บนเครื่องคอมพิวเตอร์เครื่องเดียว ที่มีค่าใช้จ่ายถูกกว่าและสามารถทำงานบนเครื่องคอมพิวเตอร์ส่วนบุคคลที่มีหน่วยประมวลผลกราฟิกส์ได้

วัตถุประสงค์ของงานวิจัย

วัตถุประสงค์ของงานวิจัยสามารถแบ่งตามกระบวนการที่เกิดขึ้นได้ดังนี้

1. สามารถออกแบบโครงสร้างข้อมูลที่ช่วยลดขนาดของเซตข้อมูล พัฒนาอัลกอริทึมและพัฒนาโปรแกรมเพื่อนำข้อมูลฐานความรู้ ไฟล์ออนโทโลยี ไปประมวลผลบนสถาปัตยกรรมแบบขนานได้ สามารถประยุกต์มาใช้กับเครื่องคอมพิวเตอร์ส่วนบุคคลที่มีหน่วยประมวลผลกราฟิกส์ได้

2. การวัดประสิทธิภาพ ประกอบด้วย ความเร็วของการประมวลผลข้อมูลออนโทโลยีเทียบกับการประมวลผลแบบลำดับและมัลติเทรด วัดเวลาในการสื่อสารข้อมูลระหว่างโฮสต์และดีไวส์ ธรูพุตของการค้นหาคำสำคัญ

ประเด็นของงานวิจัย

ข้อมูลเปิดขนาดใหญ่ที่เชื่อมต่อกันถูกนำมาใช้ประมวลผลแบบขนาน พิจารณาได้ว่าคล้ายคลึงกับระบบคลังข้อมูล ฉะนั้นการทำงานกับข้อมูลที่ไม่ค่อยมีการเปลี่ยนแปลงแต่เน้นที่การอ่านข้อมูลเพื่อค้นหา สำหรับการเขียนข้อมูลจะเขียนคำตอบบนไฟล์ใหม่ ซึ่งภายในหน่วยประมวลผลกราฟิกส์อาจใช้หน่วยความจำเดียวกับข้อมูลเข้าหรือคนละส่วนขึ้นอยู่กับปัจจัยในการทดลอง ในบางกรณีอาจมีขั้นตอนการเตรียมข้อมูลก่อนประมวลผลเพื่อความเหมาะสมกับการเก็บข้อมูลลงหน่วยความจำที่มีพื้นที่ขนาดเล็กด้วยโครงสร้างข้อมูลที่พัฒนาขึ้น ประเด็นคือการจัดการเซตข้อมูลที่นำมาประมวลผลบนหน่วยประมวลผลกราฟิกส์อย่างเหมาะสมและสามารถประยุกต์ใช้กับเซตข้อมูลหลายโดเมนได้อย่างไร ประเด็นหลักนี้ก่อให้เกิดคำถามที่ต้องมีการศึกษาและรายงานผลในแต่ละขั้นตอนดังต่อไปนี้

1. สามารถประมวลผลแบบขนานกับฐานความรู้บนสถาปัตยกรรมแบบหลายแกนแบบใดบ้าง
2. สามารถใช้โครงสร้างข้อมูลแบบใด ในการค้นหาคำตอบจากฐานความรู้ขนาดใหญ่ที่นำมาประมวลผลบนหน่วยประมวลผลหลายแกนคือ หน่วยประมวลผลกราฟิกส์ที่มีพื้นที่ของหน่วยความจำขนาดเล็กกว่าหน่วยประมวลผลกลาง ด้วยความเร็วที่เพิ่มขึ้นได้อย่างไร

ขั้นตอนการดำเนินงาน

1. ศึกษาปัญหาเฉพาะด้านของการทำงานของออนโทโลยี ที่นำมาใช้ประมวลผลแบบขนานบนหน่วยประมวลผลแบบหลายแกน และศึกษาเทคโนโลยีแบบขนานที่ใช้ในปัจจุบันเพื่อแก้ปัญหาตามสภาพแวดล้อมที่เอื้ออำนวย
2. ศึกษาทฤษฎีและงานวิจัยที่เกี่ยวข้อง
3. ออกแบบ และพัฒนาอัลกอริทึม สำหรับเทคโนโลยีการประมวลแบบขนาน โดยใช้หน่วยประมวลผลกราฟิกส์
4. เปรียบเทียบวิธีการกับงานวิจัยอื่น หาจุดเด่นจุดด้อย พัฒนาเครื่องมือเป็นชุดพัฒนาเพื่อนำไปเผยแพร่ และทดสอบกับกรณีศึกษา รวมทั้งเขียนรายงาน

เค้าโครงงานวิจัย

บทที่ 1 บทนำ ภาพรวมของงานวิจัย ความสำคัญของงานวิจัย ที่มาของงานวิจัย วัตถุประสงค์ ประเด็นของงานวิจัย คำถามที่มีต่องานวิจัย ขั้นตอนการดำเนินงานและเค้าโครงวิทยานิพนธ์

บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง ประกอบด้วยทฤษฎีการประมวลผลแบบขนานข้อมูลที่ได้รับการเผยแพร่จากเทคโนโลยีลิงค์เดต้า โครงสร้างข้อมูลและอัลกอริทึมที่สนับสนุนการประมวลฐานออนโทโลยีแบบขนาน และงานวิจัยที่เกี่ยวข้อง

บทที่ 3 ขั้นตอนการดำเนินงาน ประกอบด้วยกรอบการทำงาน ขั้นตอนการพัฒนาโปรแกรมและแผนการศึกษาวิจัย

บทที่ 4-5 นำเสนอการพัฒนางานวิจัยได้แก่ การประมวลคิวรี่แบบขนานบนหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกส์ที่มีการแปลงข้อมูลก่อนการประมวลผล (บทที่ 4) การนำเข้าประมวลผลในหน่วยประมวลผลกราฟิกส์โดยตรง (บทที่ 5)

บทที่ 6 สรุปผลการวิจัยและการดำเนินงานในอนาคต

ภาคผนวก ตัวอย่างการประยุกต์ใช้กับโดเมนอื่น ตัวอย่างการทำงานกับภาษาจาวา ตัวอย่างการทำงานกับฐานข้อมูลแบบคีย์-ค่า

คำศัพท์ที่เกี่ยวข้อง

1. เว็บเชิงความหมายหรือเว็บข้อมูล (web of data) เป็นส่วนขยายของเว็บแบบเดิมที่องค์กรเวิลด์ไวด์เว็บสากล (W3C) กำหนดมาตรฐานว่าเป็นฐานข้อมูลเปิดแบบหนึ่งที่สามารถค้นหาได้เช่นเดียวกับฐานข้อมูล เทคโนโลยีนี้ช่วยให้มนุษย์สร้างที่เก็บข้อมูลบนเว็บได้โดยตรง โดยเพิ่มคำบรรยายและเขียนกฎจัดการความหมาย โดยสนับสนุนคอมพิวเตอร์ให้เข้าใจข้อมูลโดยวิเคราะห์ความหมายข้อมูลจากหน้าเว็บได้ ต่อมาข้อมูลที่สัมพันธ์กันนี้ได้รับการเผยแพร่ด้วยเทคโนโลยีลิงค์เดต้าเชื่อมต่อข้อมูลระหว่างฐานความรู้ได้ โดยข้อมูลมักอยู่บนเทคโนโลยีข้อมูลอาร์ดีเอฟ

2. เทคโนโลยีลิงค์เดต้าหรือการเผยแพร่ข้อมูลที่เชื่อมต่อกัน (Linked Data) เป็นการผลิตเซตข้อมูลที่มีความสัมพันธ์ระหว่างกัน (relationships among data) คือ Linked Open Data โดยข้อมูลที่เก็บอย่างน้อยต้องมีความหมายและรูปแบบอยู่ระดับของอาร์ดีเอฟที่แปลงกลับเป็นข้อมูลเชิงโครงสร้างหรือภาษามาร์กอัพอื่นได้ ในปัจจุบันนี้เทคโนโลยีลิงค์เดต้าถือเป็นหัวใจของเว็บเชิงความหมาย เนื่องจากเป็นการบูรณาการข้อมูลร่วมกับหลายชุดข้อมูล และสื่อความหมายตามตรรกศาสตร์เชิงบรรยาย (Description Logic) นับเป็นประโยชน์อย่างยิ่งต่อผู้พัฒนาแอปพลิเคชันด้านเว็บเชิงความหมายสามารถเลือกใช้เซตข้อมูลตามระดับการเปิดเผยข้อมูลได้

3. ออนโทโลยี (ontology) เป็นชุดคำศัพท์ที่นิยามแนวคิดและความสัมพันธ์ เพื่อบรรยายและแสดงขอบเขตของสิ่งที่สนใจ ถูกสร้างเพื่อกำหนดกลุ่มเฉพาะตามแอปพลิเคชันที่ใช้งาน กำหนดคุณลักษณะของความสัมพันธ์และกฎในข้อมูลได้ โดยคำศัพท์ใช้บรรยายหนึ่งหรือสองคลาสหรือซับซ้อนมากในระดับหลายพันคลาสและกำหนดความซับซ้อนของคลาสตามเป้าหมายที่นำไปใช้

4. การประมวลแบบขนาน เป็นการเพิ่มประสิทธิภาพจากการประมวลผลแบบลำดับทั้งความเร็วและปริมาณผลลัพธ์ให้มากขึ้น ซึ่งสามารถทำได้หลายวิธีทั้งแบบการใส่ไดเรคทีฟเพื่อทำมัลติเทรดในโปรแกรมแบบลำดับ ตลอดจนใช้กรอบการทำงานแบบขนานที่ได้รับการพัฒนาสำหรับใช้งานบนสถาปัตยกรรมแบบขนานโดยเฉพาะ

5. สถาปัตยกรรมแบบหลายแกน คือ หน่วยประมวลผลที่ประกอบด้วยหน่วยความจำและหน่วยคำนวณที่เชื่อมต่อกัน โดยสามารถคำนวณพร้อมกันตั้งแต่สองแกนขึ้นไป ในลักษณะการใช้หน่วยความจำร่วมกัน (shared memory) ในปัจจุบันหน่วยประมวลผลเหล่านี้เช่น หน่วยประมวลผลกราฟิกส์ สามารถทำงานพร้อมกันได้ระดับหลายพันแกนขึ้นไป หน่วยประมวลผลกราฟิกส์แบบหลายแกน ต่างจากหน่วยประมวลผลกลางแบบหลายแกนโดย หน่วยประมวลผลกราฟิกส์เน้นจำนวนแกนของหน่วยคำนวณบนพื้นที่หน่วยความจำที่น้อยลง และปัจจุบันมีชุดพัฒนาโปรแกรมแบบขนานที่ควบคุมการทำงานด้วยภาษาโปรแกรมระดับบนช่วยให้สร้างงานที่ต้องใช้การประมวลผลที่มีประสิทธิภาพสูงได้สะดวกมากขึ้น

6. โฮสต์ (Host) เครื่องที่มีหน่วยประมวลผลกลาง เป็นหน่วยคำนวณและควบคุมหลักของระบบ สามารถส่งคำสั่งไปควบคุมการทำงานของดีไวส์ได้ ส่วนดีไวส์ (Device) ส่วนที่มีหน่วยประมวลผลกราฟิกส์เป็นหน่วยประมวลผล สามารถรับคำสั่งจากโฮสต์เพื่อทำงานแบบขนานได้

7. ธรูพุต (Throughput) คือค่าประสิทธิภาพการทำงานที่วัดได้ เช่น จำนวนศัพท์ที่ค้นหาได้ต่อเวลาหนึ่งนาที

8. URL (Uniform Resource Locator) ที่อยู่ของเว็บเพจ ส่วน URI (Uniform Resource Identifier) คือ การกำหนดที่อยู่ของทรัพยากรหน้าเว็บอย่างละเอียด ลึกลงไปถึงส่วนต่าง ๆ ในเว็บเพจตาม URL ทั้ง URL และ URI เป็นไปตาม RFC 2396ส่วนมากมีรูปแบบ scheme:[//[user:password@]host[:port]][/]path[?query][#fragment] รูปแบบที่เห็นทั่วไปคือ URL#fragment

9. IRI (Internationalized Resource Identifier) การกำหนดทรัพยากรเว็บระดับ URI แต่มีการกำหนดภาษาหลายชนิดมากขึ้นตาม RFC 3987

10. TBox (terminological component) คือการพิจารณาโครงสร้างหรือคำศัพท์ของฐานความรู้

11. ABox (assertion component) คือ การยืนยันอินสแตนซ์หรือสมาชิกของคลาสหรือความรู้ที่ต้องการหาคำตอบ

บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะนำเสนอการศึกษาทฤษฎีที่เกี่ยวข้องกับการประมวลผลแบบขนาน รูปแบบและหลักการการประมวลแบบขนาน การจัดการข้อมูล ประกอบด้วย การทำดัชนีข้อมูลและการแทนค่าข้อมูล เว็บเชิงความหมาย ภาษาที่ใช้พัฒนา การจัดเก็บออนโทโลยี ตรรกศาสตร์ ภาษาคิวรี่และการคิวรี่เพื่อหาคำตอบ จากนั้นจึงนำเสนอผลของการศึกษางานวิจัยที่เกี่ยวข้อง ดังรายละเอียดต่อไปนี้

ทฤษฎีที่เกี่ยวข้อง

การประมวลผลแบบขนาน

ในหัวข้อนี้นำเสนอทฤษฎีที่เกี่ยวข้องกับการประมวลผลแบบขนาน

การประมวลผลแบบขนาน เป็นการใช้เครื่องหรือหน่วยประมวลผลจำนวนมากกว่าหนึ่งในการคำนวณหรือแก้ปัญหาจากโจทย์เดียวกัน เพื่อให้ได้คำตอบเร็วกว่าคำนวณบนเครื่องประมวลผลเครื่องเดียวหรือหน่วยเดียว [1]

การประมวลผลพร้อมกัน (concurrent processing) หมายถึง การประมวลผลพร้อมกันเป็นการทำงานแบบขนานในด้านแนวคิด นั่นคือ การประมวลผลงานตั้งแต่สองงานขึ้นไปพร้อมกัน ซึ่งในความจริงเครื่องอาจจะปฏิบัติงานขณะใด ๆ ได้ครั้งละหนึ่งโปรแกรมอย่างเร็ว จนดูเหมือนทำหลายโปรแกรมพร้อมกัน (multitasking) หรือทำแยกงานในแต่ละหน่วยประมวลผล [2]

การออกแบบสถาปัตยกรรมที่ทำงานแบบขนานของหน่วยประมวลผลกลาง

ฟลินน์ ได้แบ่งสถาปัตยกรรมคอมพิวเตอร์ออกเป็นสี่กลุ่มตามจำนวนของคำสั่งหรือสัญญาณควบคุมที่ทำงานพร้อมกันและตามการไหลของข้อมูล [3] ได้ดังต่อไปนี้

SIMD (Single-instruction stream, multiple-data stream) หน่วยประมวลผลหลายหน่วยทำงานคำสั่งเดียวกัน แต่ทำกับข้อมูลที่ต่างกัน เช่นทำกับแถวลำดับ (array) ข้อมูลตำแหน่งต่างกัน

SISD (Single-instruction stream, single-data stream) เป็นหน่วยประมวลผลที่มีการทำงานแบบลำดับ ทำงานแต่ละคำสั่งกับข้อมูลหนึ่งส่วน

MIMD (Multiple-instruction streams, multiple-data streams) หน่วยประมวลผลหลายหน่วยทำงานหลายคำสั่งพร้อมกันกับข้อมูลที่แตกต่างกัน

MISD (Multiple-instruction streams, single-data stream) คือการทำงานหลายคำสั่งกับข้อมูลกระแสเดี่ยว หน่วยประมวลผลหลายหน่วยใช้ข้อมูลเดียวกัน ซึ่งไม่เป็นที่นิยม

ในหน่วยประมวลผลกลางแบบหลายแกนมีการออกแบบฮาร์ดแวร์เพื่อสนับสนุนการประมวลแบบขนาน และเพื่อเพิ่มประสิทธิภาพของหน่วยประมวลผลกลางดังต่อไปนี้

สถาปัตยกรรมแบบไปป์ไลน์ เป็นการประมวลผลหลายคำสั่งในขณะเดียวกันแต่ต่างขั้นตอนในไปป์ไลน์

สถาปัตยกรรมแบบซูเปอร์สเกลาร์ เป็นการเพิ่มจำนวนและประเภทของหน่วยคำนวณให้มีจำนวนมากขึ้น คำสั่งจะถูกแบ่งไปยังแต่ละหน่วย แต่ละประเภทเพื่อคำนวณอย่างอิสระได้ เมื่อหน่วยประมวลผลกลางทำคำสั่งทั้งหมดในเวลาเดียวกันถือเป็นการเพิ่มความเร็วในการประมวลผล

ในบางครั้งเมื่อสถาปัตยกรรมเปลี่ยน จำนวนชั้นและขนาดของหน่วยความจำแคชจะเปลี่ยนไป ด้วยเหตุนี้การเขียนโปรแกรมบนสถาปัตยกรรมแบบขนานที่ต่างกันต้องคำนึงถึงระดับชั้นของแคชและความสมดุลของหน่วยประมวลผล สำหรับในเชิงพาณิชย์จะแบ่งฮาร์ดแวร์เป็น 3 ประเภท

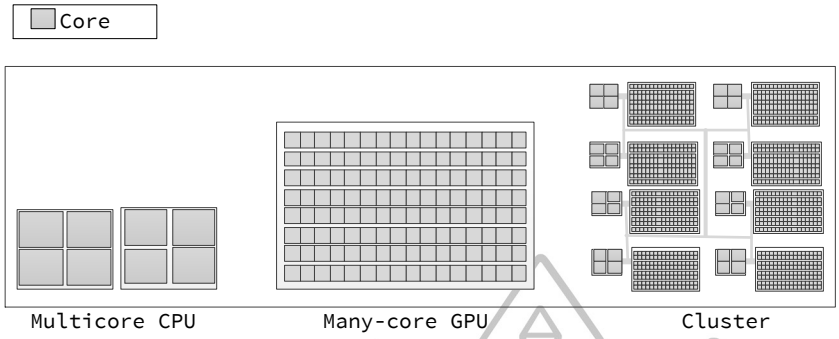
ประเภท SMP (Shared Memory Multiprocessor) แต่ละหน่วยประมวลผลจะมีหน่วยประมวลผลกลางและแคช ทำให้ต้องใช้หน่วยความจำหลักและหน่วยควบคุมร่วมกัน

ประเภทคลัสเตอร์ เป็นการนำเครื่องคอมพิวเตอร์มาเชื่อมต่อกันด้วยเครือข่ายความเร็วสูงเพื่อสื่อสารระหว่างกัน แต่ละเครื่องมีหน่วยความจำแยกจากกัน ต่างจาก SMP คือไม่มีหน่วยความจำที่ใช้ร่วมกัน (shared memory) บางครั้งเรียกว่าการประมวลผลแบบกระจาย (distributed computing)

ประเภทไฮบริด การใช้คลัสเตอร์ที่มีหน่วยประมวลแต่ละโหนดเป็น SMP จึงมีหน่วยความจำที่ใช้ร่วม (shared memory)

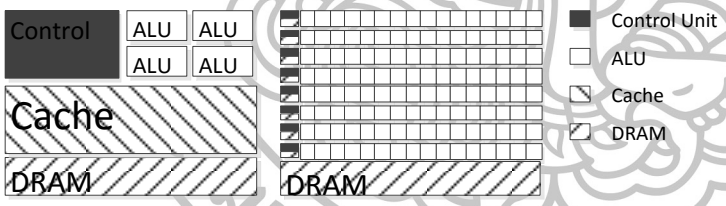
หน่วยประมวลผลกราฟิกส์ (Graphics Processing Unit: GPU) มีหน้าที่ประมวลผลร่วมกับหน่วยประมวลผลกลาง มีลักษณะเป็นชิพที่ใช้จัดการการประมวลผลกราฟิกส์เช่นภาพ โมเดลสามมิติ เป็นต้น มาจากแนวคิดการคำนวณสีแต่ละพิกเซลที่เป็นอิสระต่อกันซึ่งเหมาะกับการทำงานแบบขนาน ต่อมานักพัฒนาได้นำมาใช้งานด้านอื่น ๆ เรียก GPGPU (General Purposed Graphics Processing Unit) ส่งผลให้ผู้ผลิตใส่หน่วยความจำนี้บนบอร์ดหลักพร้อมกับโปรแกรมช่วยให้นักพัฒนาเขียนโปรแกรมควบคุมหน่วยประมวลนี้จากบนโฮสต์ได้ ทำให้เครื่องคอมพิวเตอร์ส่วนบุคคลทำงานได้เหมือนกับซูเปอร์คอมพิวเตอร์ขนาดย่อม ขณะเดียวกัน GPU มีราคาถูกลง จึงเป็นทางเลือก

ให้นำมาทดแทนหน่วยประมวลผลกลางแบบหลายแกนและคลัสเตอร์ได้ งานวิจัยนี้เน้นการทำงานบนสถาปัตยกรรมดังกล่าว



รูปที่ 2.1 การเปรียบเทียบจำนวนแกนของหน่วยประมวลผลระหว่าง CPU GPU และคลัสเตอร์

จากรูปที่ 2.1 เมื่อเปรียบเทียบแกนของการประมวลผลแบบขนานของหน่วยประมวลผลกลางแบบหลายแกน (Multicore CPU) หน่วยประมวลผลกราฟิกส์แบบหลายแกน (Many-core GPU) [4] และแบบคลัสเตอร์ จะเห็นได้ว่าจำนวนแกนของหน่วยประมวลผลกลางแบบหลายแกนมีน้อยกว่าหน่วยประมวลผลกราฟิกส์แบบหลายแกน แต่มีพื้นที่ของหน่วยความจำที่มากกว่า ในขณะที่คลัสเตอร์มีจำนวนแกนมากที่สุดและมีค่าใช้จ่ายในการประมวลผลมากที่สุด ผู้วิจัยจึงสนใจหน่วยประมวลผลกราฟิกส์ที่มีจำนวนหลายแกนและมีราคาถูกกว่า



รูปที่ 2.2 ภาพสถาปัตยกรรมภายในระหว่าง CPU และ GPU

ที่มา: David B. Kirk, Wen-mei W. Hwu, **Programming Massively Parallel Processors: A Hands-on Approach**, (Elsevier, 2010), 4.

เมื่อพิจารณาลักษณะงานที่นำมาประมวลผล หน่วยประมวลผลกลางแบบหลายแกนเหมาะกับการประมวลผลจำนวนงานที่ไม่มากนักพร้อมกัน โดยแต่ละแกนรันบนเทรดเดียว จุดเด่นคือแต่ละแกนนั้นทำงานร่วมกันอย่างรวดเร็ว บนหน่วยความจำหลักขนาดใหญ่ ส่วนหน่วยประมวลผลกราฟิกส์แบบหลายพันแกนเหมาะกับการประมวลผลแบบขนานที่ต้องการจำนวนเทรดมาก เหมาะกับปัญหาขนาดใหญ่ที่แบ่งได้เป็นงานขนาดเล็กหลายพันงาน ลักษณะเด่นคือเทรดทั้งหมดทำงานเพื่อเพิ่มธรูพุตหรือจำนวนงานต่อวินาที เนื่องจากมี ALU (Arithmetic Logic Unit) จำนวนมาก ที่ถูกควบคุมการทำงานได้ง่ายด้วยคำสั่ง SIMD ผ่าน ALU ดังรูปที่ 2.2 (ขวา) การจัดการพื้นที่บนหน่วย

ความจำของหน่วยประมวลผลกราฟิกส์จึงมีความสำคัญเนื่องจากขนาดของหน่วยความจำส่งผลต่อการปรับปรุงประสิทธิภาพการทำงาน แกนของหน่วยประมวลผลกราฟิกส์ต้องทำงานต่อเนื่องกันด้วยคำสั่งเดียวกันแต่กับข้อมูลต่างกันได้ แต่ละแกนทำงานด้วยข้อมูลแบบอะเรย์หรือเมตริกซ์แบบขนาน

ตารางที่ 2.1 เปรียบเทียบลักษณะทั่วไปของหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกส์

ประเภท / ชื่อ	หน่วยประมวลผลกลาง/ Intel ® Core™ i7 -6700HQ (Physical)	หน่วยประมวลผลกราฟิกส์/ NVIDIA Tesla K40c
จำนวนแกน	4 แกน (8 เทรด)	2,880 แกน
หน่วยความจำสูงสุด	64 กิกะไบต์	12 กิกะไบต์
นาฬิกากราฟิกส์	1.05 GHz (Intel® HD Graphics 530)	745 MHz
บัสกราฟิกส์	PCI-Express 3.0 (16 lanes)	PCI-Express 3.0 (16 lanes)
นาฬิกาหน่วยความจำ	3.5 GHz	3.0 GHz
ขนาดของแคช	6MB	

จากตารางที่ 2.1 แสดงการเปรียบเทียบลักษณะทั่วไปจากผลิตภัณฑ์ในช่วงไตรมาสที่ 3 ของ ค.ศ. 2015 ของหน่วยประมวลผลกลางและกราฟิกส์ พบว่าจำนวนแกนที่มีมากกว่าของ GPU ทำหน้าที่ประมวลผลร่วมกับCPU โดย GPU จะมีหน่วยความจำและความเร็วนาฬิกาน้อยกว่า CPU และใช้แคชของ CPU ในการทำงาน ซึ่ง GPU สามารถทำงานพร้อมกันหลายพันแกนเพื่อให้มีธรูพุตของงานที่มากกว่าผลจากการทำงานจาก CPU ได้

รูปแบบชั้นโปรแกรม

ในการรันโปรแกรม ประกอบด้วยส่วนที่สำคัญดังนี้

โพรเซส (process) งานของโปรแกรมที่กำลังได้รับการประมวลผลในเวลาหนึ่ง

เทรด (thread) เป็นโพรเซสขนาดเล็ก ปกติหนึ่งโพรเซสประกอบด้วยหลายเทรดที่ได้รับการประมวลผลให้ทำงานพร้อมเพรียงกันโดยแบ่งปันพื้นที่หน่วยความจำร่วมกัน

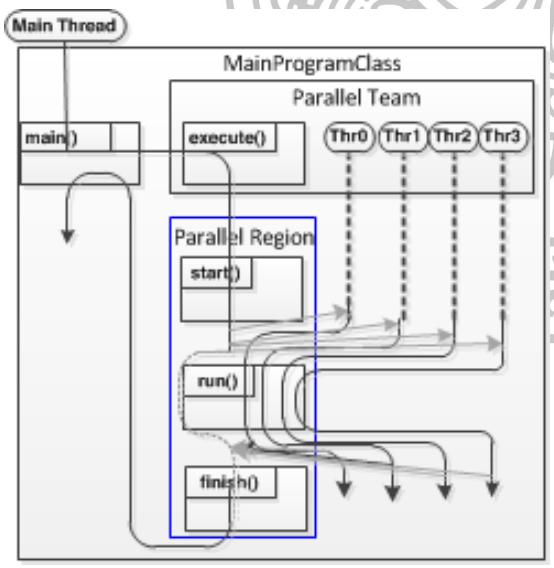
ในระบบปฏิบัติการหนึ่ง จะประกอบด้วยหลายแอปพลิเคชัน (หลายโพรเซส) ที่ทำงานพร้อมกัน และในแต่ละโพรเซส มีเทรดจำนวนมากทำงานอยู่ โดยระบบปฏิบัติการจะใช้วิธีการสลับงานระหว่างโพรเซส หรือระหว่างเทรดภายในโพรเซส การรันโปรแกรมบน GPU จะใช้เทรดจำนวนมากช่วยกันรันภายใต้โพรเซสที่ควบคุมโดย CPU

การเขียนโปรแกรมแบบขนานบน CPU มีเป้าหมายที่กำหนดงานไปให้แต่ละหน่วยประมวลผลทำงาน มีไลบารีและแพลตฟอร์มต่าง ๆ ดังต่อไปนี้

OpenMP (Open Multi-Processing) [5] OpenMP เป็นไลบารี่ที่เกิดขึ้นในช่วงการพัฒนาหน่วยประมวลผลด้วยเทคโนโลยีไฮเปอร์เทรด ช่วยให้เขียนแอปพลิเคชันสะดวกในการ

พัฒนาการทำงานแบบหลายเทรดให้ทำงานแบบขนาน โดยกระจายงานไปให้แต่ละเทรดทำงาน โดยโพรเซสเซอร์ทั้งหมดต้องใช้หน่วยความจำหลักร่วมกัน รองรับภาษาซีและฟอร์แทรน

ต่อมาพบว่าได้มีเริ่มมีนำหลักการของ OpenMP คือเซตของ directive และฟังก์ชันไลบราลีตอนรันไทม์มาพัฒนา โดยนำมารวมกับภาษาจาวา ปี ค.ศ. 2000 ไลบราลี่ JOMP (Java OpenMP) [6] เป็นไลบราลี่แรกที่ได้นำภาษาจาวามาพัฒนาตามหลัก OpenMP ไลบราลี่นี้ถูกใช้งานถึงจาวา 1.4 เนื่องจากในจาวา 1.5 ภาษาจาวามาตรฐานได้รวมหลักการแบบขนานไว้ด้วย ประกอบกับขาดการพัฒนาที่ต่อเนื่อง ต่อมาค.ศ. 2007 มีการพัฒนาไลบราลี่ JaMP (Java/OpenMP) [7] แบบปลั๊กอินที่อยู่ในแถบเมนูของโปรแกรมอีคลิปส์ (Eclipse) จึงได้รับความนิยม และพัฒนาต่อเนื่องสำหรับใช้กับหน่วยประมวลผลกราฟิกส์ ผู้พัฒนายังได้สร้างโปรแกรมจัดการพื้นที่ (Garbage Collector) ด้วย อย่างไรก็ตามทั้ง JOMP และ JaMP นั้นล้วนมีจุดเด่นในการทำงานแบบขนานในเวลารันไทม์ทั้งสิ้น ในปี ค.ศ. 2009 ได้มีการพัฒนาไลบราลี่จาวาแบบขนาน PJ (Parallel Java) [8] ที่ทำงานด้วยภาษาจาวาเต็มรูปแบบ สำหรับการทำงานของไลบราลี่คล้ายกับ OpenMP ของภาษาซี กังรูปที่ 2.3 ซึ่งมีลักษณะสำคัญ 5 กลุ่มคือกลุ่ม 1 สร้างขอบเขตของการทำงานแบบขนาน กลุ่ม 2 ใช้แบ่งการทำงาน แจกงานแก่ worker กลุ่ม 3 ประมวลผลแบบขนาน กลุ่ม 4 รวมงานที่ได้จากทุกเทรด และกลุ่ม 5 คือตัวแปรและชนิดข้อมูลซึ่งไลบราลีนี้ยังพัฒนาอยู่



รูปที่ 2. 3 การทำงานของเทรดแบบขนานของ PJ ที่เหมือนกับส่วนขนานของ OpenMP

บริษัทอินเทลและไมโครซอฟต์ได้ร่วมกันให้ทุนมหาวิทยาลัยอิลินอยด์ในการพัฒนาไลบราลี่ DPJ (Deterministic Parallel Java) [9] โดยใช้ jsr166y ที่มีคุณสมบัติทำงานแบบพร้อมเพียงและ

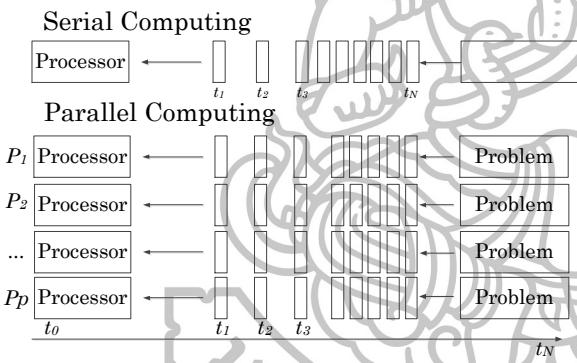
ปรับส่วนคอลเลคชัน (concurrency and collection update) ซึ่งปัจจุบันรวมไว้ในจาวา 1.7 และ JDK 7 สำหรับเวอร์ชัน 8 นั้น JVM (Java Virtual Machine) ยอมให้แอปพลิเคชันหนึ่งมีหลายเทรดและรันพร้อมกันได้ [10]

หลักการและรูปแบบการประมวลแบบขนาน

กฎของอัมดาฮ์ล (Amdahl's law) หรือข้อตกลงของอัมดาฮ์ล (Amdahl's argument) เป็นทฤษฎีการเร่งความเร็วของการพัฒนาโปรแกรมแบบขนาน กฎนี้กล่าวว่าความเร็วที่เพิ่มขึ้น จะขึ้นอยู่กับสัดส่วนของงานในโปรแกรมที่ทำแบบขนานเทียบกับงานทั้งหมด [11] กฎนี้ก่อให้เกิดหลักการทำงานและรูปแบบ แล้วจึงขยายไปสู่การออกแบบสถาปัตยกรรมแบบขนาน

อัลกอริทึมแบบขนาน (Parallel Algorithm) เป็นขั้นตอนวิธีทางคอมพิวเตอร์ที่ออกแบบในลักษณะให้หลายเทรดทำงานพร้อมกันเพื่อการประมวลผลแบบขนาน สำหรับแต่ละปัญหาต้องได้รับการออกแบบแบ่งงานเป็นงานย่อย โดยให้ทุกหน่วยประมวลผลทำงานได้อย่างเต็มประสิทธิภาพ

การออกแบบอัลกอริทึมแบบขนาน



รูปที่ 2.4 การเปรียบเทียบการแบ่งปัญหาเพื่อประมวลผลแบบลำดับและแบบขนาน

โมเดลแบบขนาน (Parallel Models) นั้นเกิดจากแนวคิดการใช้หน่วยประมวลผล (Processing Elements: PEs) จำนวน P หน่วยดังรูปที่ 2.5 ทำงานขนาดใหญ่ชิ้นเดียวกันพร้อม ๆ กัน จะทำให้งานเสร็จเร็วขึ้นเป็นสัดส่วนคือ 1/P ในทางปฏิบัติระหว่างหน่วยประมวลผลจะมีการติดต่อสื่อสารระหว่างกันทำให้เวลาของการประมวลผลแบบขนานมากกว่า 1/P หากกล่าวถึงโมเดลแบบ RAM (Random Access Machine) เป็นโมเดลคอมพิวเตอร์ที่มี 1 หน่วยประมวลผล (Uniprocessor) ส่วนโมเดลแบบ PRAM (Parallel RAM) เป็นโมเดลที่มีหลายหน่วยประมวลผล ประกอบด้วยหน่วยควบคุม (Control Unit) หน่วยความจำร่วม และหน่วยประมวลผล n หน่วย สำหรับหน่วยความจำร่วม อาจเป็นแบบรวมศูนย์ (Centralized Shared Memory) หรือกระจายในแต่ละหน่วยประมวลผล (Distributed Shared Memory) สมมติให้เวลาที่โมเดล PRAM ใช้ในการ

ติดต่อสื่อสารหน่วยประมวลผล (Synchronization Overhead) ในแต่ละครั้งเป็น 0 และเวลาเข้าถึงข้อมูลในหน่วยความจำ (Memory Access Overhead) มีค่าเป็น 0 ด้วย แม้ว่าในทางปฏิบัติเวลาที่ใช้ติดต่อสื่อสารมากกว่า 0 เพื่อความสะดวกในการวิเคราะห์ความซับซ้อนด้านเวลาของขั้นตอนวิธีเท่านั้น เนื่องจากโมเดลแบบ PRAM จะช่วยลดข้อจำกัด ในเรื่องเวลาที่ใช้ในการติดต่อระหว่างหน่วยความจำ ซึ่งแตกต่างกันตามชนิดของคอมพิวเตอร์แบบขนาน ดังนั้นโมเดล PRAM จึงเป็นระบบคอมพิวเตอร์แบบขนานในอุดมคติหรือตามทฤษฎี

การแบ่งภาระงาน

การแบ่งภาระงาน [12] สำหรับระบบประมวลแบบขนาน จะพิจารณาเฉพาะส่วนงานที่สามารถประมวลแบบขนานได้ โดยแบ่งส่วนย่อยนั้นกระจายไปให้แต่ละโพรเซสเซอร์หรือแต่ละเทรดตามลักษณะสถาปัตยกรรมแบบหลายแกนที่รองรับ ควรแบ่งงานให้เท่ากัน เพื่อให้แต่ละเทรดทำงานเสร็จพร้อมกันได้ ไม่เสียเวลารอระหว่างเทรด เมื่องานแบบขนานเสร็จสิ้น อาจต้องทำงานแบบลำดับต่อ ส่วนของการทำงานแบบขนานในหนึ่งโปรแกรม ควรมีมากพอเพื่อเวลารวมของโปรแกรมประยุกต์นั้นเร็วขึ้นเมื่อเทียบกับทำงานแบบลำดับ [13]

การส่งงานไปยังหน่วยประมวลผล

เมื่อแบ่งงานแล้วจะส่งงานย่อยไปยังหน่วยประมวลผล โดยทั่วไปขึ้นอยู่กับทรัพยากรของระบบอาทิ แบนด์วิธ I/O ซึ่งอาจเกิดปัญหาคอขวดในการส่งข้อมูลไปประมวลผลในยุคแรก แต่ในสมัยต่อมาได้รับการแก้ไขด้วยการลดขนาดข้อมูลที่ส่ง จึงมีปัญหาด้านความสามารถของหน่วยความจำแทน เพราะหน่วยความจำมีขนาดจำกัด การย่อข้อมูลจะทำให้บรรจุข้อมูลได้มากขึ้นและเพิ่มแบนวิดธ์ในการรับส่งข้อมูล ในกรณีของงานประมวลคิวรี่จากข้อมูลจำนวนมาก ต้องลดขนาดข้อมูลและลดโอเวอร์เฮดให้เหลือน้อยที่สุด [14][15] เพื่อให้การรับส่งข้อมูลทำได้เร็ว และเอาข้อมูลเข้าไปในหน่วยความจำได้มาก เพื่อค้นหาข้อมูลให้มากที่สุดในแต่ละรอบการทำงาน

การรวมคำตอบ

หลังจากการประมวลผลในหน่วยประมวลแบบขนานแล้ว จะต้องรวมคำตอบที่ได้จากการประมวลผลทั้งหมดเข้าด้วยกัน ซึ่งในโมเดลการโปรแกรมทั่วไป อาจจะใช้วิธีการ Master-Slave โดยกำหนดให้หน่วยประมวลผลหรือเทรดตัวหนึ่งทำหน้าที่เป็น Master เพื่อกระจายงานและรวบรวมข้อมูล กลับจากหน่วยอื่น (Slave) การรวมนั้นใช้ตัวดำเนินการบางอย่างเข้าช่วยระหว่างการรวมเพื่อให้ได้ข้อมูลสรุปแบบที่ต้องการ บางกรณี อาจใช้หน่วยความจำร่วมช่วยในการรวมข้อมูล [16]

ดังนั้นจึงต้องวางแผนเพื่อส่งคำสั่งหรือข้อมูลอย่างมีประสิทธิภาพจากทุกหน่วยประมวลผลหรือเทรดมาสู่หน่วยความจำร่วมในโปรแกรม [17]

การคำนวณประสิทธิภาพในการทำงานของโปรแกรมแบบขนาน

การออกแบบและพัฒนาโปรแกรมต้องเขียนแอปพลิเคชันที่ทำงานแบบขนานโดยใช้เทรด มีผู้วิจัยเรื่องรูปแบบการคิดเวลาเพื่อประเมินการทำงานบนหน่วยประมวลผลกราฟิกส์ เช่นการคิวรี่ [18] เวลาทั้งหมดที่ใช้ไปสำหรับการประเมินคิวรี่ของทุกหน่วยประมวลผลที่เกี่ยวข้องคือ $T_{overall}=T_{mm\_dm(I)}+T_{GPU}+T_{dm\_mm(O)}$ เวลาทั้งหมดที่ใช้ มาจาก $T_{mm\_dm(I)}$ เป็นเวลาที่ใช้คัดลอกข้อมูลเข้าจากหน่วยความจำหลักไปที่หน่วยความจำหน่วยประมวลผลกราฟิกส์ โดย I แทนเซตของข้อมูลเข้า T_{GPU} เป็นเวลาการประเมินคิวรี่ในหน่วยประมวลผลกราฟิกส์ใช้บนหน่วยความจำหน่วยประมวลผลกราฟิกส์และ $T_{dm\_mm(O)}$ เป็นเวลาที่ใช้คัดลอกผลลัพธ์การคิวรี่จากหน่วยความจำหน่วยประมวลผลกราฟิกส์กลับมาที่หน่วยความจำหลัก โดยกำหนดให้ O เป็นเซตของข้อมูลออก ปัจจัยที่สนับสนุนการประมวลผลแบบขนานบนหน่วยประมวลผลกราฟิกส์คือ นำข้อมูลเข้าสู่หน่วยประมวลผลกราฟิกส์แล้วกำหนดให้หน่วยประมวลผลทำงานอย่างคุ้มค่า การนำข้อมูลออกหรือนำกลับมาใช้ใหม่เพื่อหลีกเลี่ยงข้อจำกัดเรื่องแบนด์วิธของหน่วยความจำและเวลานำข้อมูลเข้าออก

Compute Unified Device Architecture (CUDA)

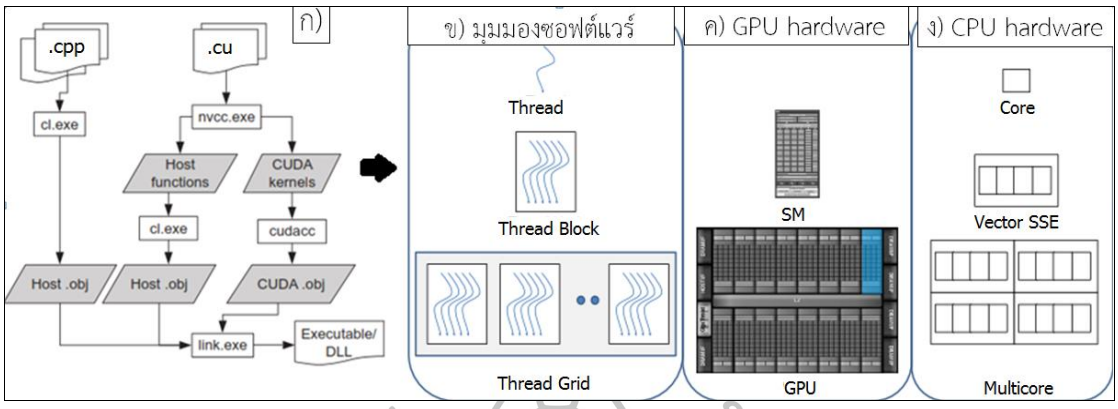
Compute Unified Device Architecture (CUDA) หรือคูด้าเป็นกรอบการทำงานที่ช่วยผู้พัฒนาโปรแกรมแบบขนานโดยใช้หน่วยประมวลผลกราฟิกส์ โดยมีการควบคุมการทำงานจากโปรแกรมโฮสต์ ในเบื้องต้นรองรับภาษาฟอร์แทรนและซี บริษัทเอ็นวิเดีย (NVIDIA) สร้างและพัฒนาในแต่ละรุ่น การออกแบบสัมพันธ์กับกฎของอัมดาฮ์ล เมื่อใช้งานหน่วยประมวลผล *n* หน่วย มีส่วนที่เป็นการทำงานขนาน *P* และส่วนที่ไม่ใช่งานแบบขนาน *1-P* ดังนั้น ความเร็ว $S(n) = 1/[(1-P)+P/n]$ สำหรับคูด้าต้องคำนึงถึงขอบเขตของโค้ดที่ทำงานแบบขนานเพื่อให้ทำงานได้เร็วที่สุดเท่าที่จะทำได้ ฉะนั้นความเร็ว *N* เท่า จะเป็นไปได้เมื่อใช้หน่วยประมวลผล *n* หน่วยและงานแบบลำดับในรูป *1-P* จะต้องลดลงให้เป็น 0 เมื่อพิจารณาโปรแกรมในหน่วยประมวลกราฟิกส์พบว่าสามารถทำความเร็วในส่วนของการทำงานแบบขนานเป็น $P >> 1-P$ [19] ตัวอย่างการแบ่งงานแบบขนานระดับลูป โดยแต่ละลูปทำงานอิสระต่อกัน การสร้างโปรแกรมคูด้ามีขั้นตอนคือ 1) สร้างอะเรย์ฝั่งดีไวส์ 2) คัดลอกข้อมูลจากหน่วยความจำหลักไปยังอะเรย์นั้น 3) รวมผลจากอะเรย์ทั้งหมดของดีไวส์ 4) ส่งผลลัพธ์กลับมาที่โฮสต์

ตารางที่ 2.2 ประเภทและคุณลักษณะของหน่วยความจำบนหน่วยประมวลผลกราฟิกส์จากมุมมองของคูด้า

หน่วยความจำ	สถานที่	แคช	การเข้าถึง	ขอบเขต
รีจิสเตอร์	ส่วนหนึ่งของชิพ	ไม่	อ่าน/เขียน	1 เทรด
โลคอล	ส่วนหนึ่งของชิพ	ใช้	อ่าน/เขียน	1 เทรด
แชร์	ส่วนหนึ่งของชิพ	-	อ่าน/เขียน	เทรดทั้งหมดในหนึ่งบล็อก
โกลบอล	ไม่ใช่ส่วนหนึ่งของชิพ (ถ้าไม่ใช้แคช)	ใช้	อ่าน/เขียน	เทรดทั้งหมด + โฮสต์
คอนสแตนซ์	ไม่ใช่ส่วนหนึ่งของชิพ (ถ้าไม่ใช้แคช)	ใช้	อ่าน	เทรดทั้งหมด+ โฮสต์
เท็กซ์เจอร์	ไม่ใช่ส่วนหนึ่งของชิพ (ถ้าไม่ใช้แคช)	ใช้	อ่าน/เขียน	เทรดทั้งหมด+ โฮสต์

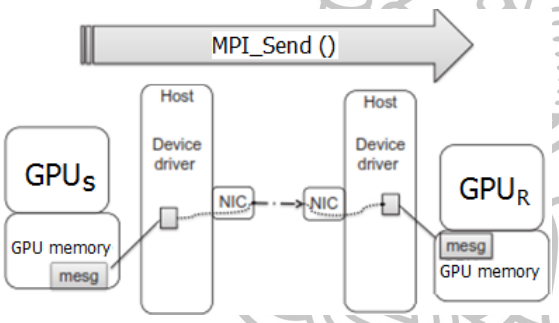
สำหรับพื้นที่หน่วยความจำที่ใช้ประมวลผลของหน่วยประมวลผลกราฟิกส์คือพื้นที่หน่วยความจำที่แยกส่วนจากหน่วยประมวลของโฮสต์ หน่วยความจำโกลบอลคือหน่วยความจำที่มีพื้นที่ใหญ่ที่สุดบนหน่วยประมวลผลกราฟิกส์เป็นส่วนที่ถูกใช้ทำกิจกรรมร่วมกันทุกเทรด มีอัตราการส่งข้อมูลเข้าและออกสูงสุด นอกจากนี้หน่วยความจำอื่นแต่ละระดับที่เกี่ยวข้องมีคุณลักษณะดังตารางที่ 2.2 การเข้าถึงหน่วยความจำโกลบอลอาจใช้เวลาถึง 600 เท่าของการเข้าถึงรีจิสเตอร์ ผู้พัฒนาโปรแกรมจึงควรตระหนักว่าถึงแม้แบนด์วิธของหน่วยความจำโกลบอลจะมีค่าสูงประมาณ 160-200 กิกะไบต์/วินาที แต่ยังเป็นค่าที่น้อยมากเมื่อเทียบกับความสามารถระดับเทระฟลอป (teraflop) ที่หน่วยประมวลผลกราฟิกส์สามารถทำได้ ด้วยเหตุนี้การนำข้อมูลกลับมาใช้ใหม่จึงจำเป็นเพื่อให้ได้การประมวลที่มีสมรรถนะสูงขึ้น

ในการคอมไพล์โปรแกรมบนกรอบของคูด้าจะใช้ตัวแปรภาษา nvcc สำหรับขั้นตอนการทำงานของคอมไพเลอร์ nvcc สำหรับระบบปฏิบัติการวินโดว์เป็นดัง รูปที่ 2.6 ก) การเขียนโปรแกรมเป็น .cu ที่มีภาษาซีเป็นพื้นฐาน นอกจากการทำงานบนโฮสต์แล้ว nvcc คอมไพล์งานบนเคอร์เนลโดยคูด้าเป็นแบบแถวลำดับของเทรด โดยเทรดทั้งหมดทำงานเดียวกันเป็นลักษณะ SIMT (single instruction multiple threads) แต่ละเทรดจะมีลำดับกำกับเพื่อใช้คำนวณที่อยู่ของข้อมูลและควบคุมการทำงาน ภายในเคอร์เนลมีการประมวลตามกริดที่มีบล็อกอยู่ภายใน กริดคือกลุ่มของบล็อกที่สามารถแยกการทำงานอิสระต่อกันได้ บล็อกคือการรวมกลุ่มของแถวลำดับของเทรดที่ทำงานร่วมกัน (Cooperative Thread Array, CTA) ภายในบล็อกมีเทรดจำนวนมากทำงานและใช้ข้อมูลร่วมกันบนหน่วยความจำร่วม แล้วจึงเชื่อมการประมวลให้ตรงกัน โดยเทรดจากบล็อกต่างกันทำงานเป็นอิสระต่อกัน เมื่อเป็นการทำงานระหว่างคูด้ากับแกนของโฮสต์จะถูกมองเป็น SM (stream multiprocessor) จากรูปที่ 2.6 ข) ถึง ง) คือบล็อกของเทรดเทียบกับแกนประมวลผล เมื่อมองไปที่การทำงานแบบขนานระดับเทรดก็จะเทียบได้กับ SSE (streaming SIMD extensions)



รูปที่ 2.6 การคอมไพล์บนระบบปฏิบัติการวินโดว์เทียบซอฟต์แวร์และฮาร์ดแวร์ของ GPU และ CPU

ที่มา: Farber, R., **CUDA Application Design and Development**, (Elsevier, 2011), หน้า 103, 186



รูปที่ 2.7 เส้นทางข้อมูลของ MPI GPUDirect

ที่มา: Farber, R., **CUDA Application Design and Development**, 2011, หน้า 247

สำหรับการใช้คู่ค้าในระบบกลุ่มเมฆและภายในคลัสเตอร์ของคอมพิวเตอร์ที่ประกอบด้วยหน่วยประมวลผลกราฟิกส์ต้องอาศัยไลบารี่ของ MPI (Message Passing Interface) และ GPUDirect ซึ่งอาศัยการทำงานของไดรเวอร์เอ็นวิเดียเป็นดังรูปที่ 2.7 จะทำการจองบัฟเฟอร์ คัดลอกข้อมูลจากหน่วยความจำดีไวส์ลงในบัฟเฟอร์ฝั่งต้นทางของ GPU_S และปลายทางจะนำข้อมูลลงบัฟเฟอร์และคัดลอกขึ้นหน่วยความจำของGPU_R เพื่อส่งเมสเสจระหว่างคลัสเตอร์หรือระหว่างคอมพิวเตอร์

ไลบรารี่และฟังก์ชันสำหรับการทำงานแบบขนานด้วยภาษาจาวา

JaMP (Java/OpenMP) เป็นปลั๊กอินที่อยู่ในแถบเมนูของโปรแกรมอีคลิปส์ (Eclipse) เป็นไลบารี่จาวาที่ใช้ร่วมกับ OpenMP เพื่องานแบบหลายแกน และได้รับการพัฒนาต่อเนื่องสำหรับใช้กับหน่วยประมวลผลกราฟิกส์ โดย JaMP ทำงานแบบรันไทม์

เจคูด้า (Java bindings for CUDA, jCUDA) เป็นไลบรารี่ภาษาจาวาสำหรับนักพัฒนาที่ใช้ภาษาจาวา ที่เชื่อมระหว่างคูด้ารันไทม์และ API ของไดรเวอร์ [20] มีลักษณะเด่นคือสร้างโมดูลผ่าน API ได้ มีการสร้างไลบรารีตามคูด้า เพื่อแก้ปัญหาโจทย์ได้หลายด้านเช่น CUBLAS CUSPARSE รวมถึงรองรับ OpenGL และจัดการข้อผิดพลาดได้อย่างรวดเร็ว ส่วนงานวิจัย JCUDA [21] หมายถึงการพอร์ตบางส่วนของคูด้าไปขยายไวยากรณ์จาวา โดยมีอินเตอร์เฟส (JNI) เชื่อมกับภาษาระดับต่ำเพื่อใช้เคอร์เนลและคัดลอกข้อมูลระหว่างกัน

MPJExpress [22] เป็นไลบรารี่ MPI ภาษาจาวาที่ให้นักพัฒนาจาวาประมวลแอปพลิเคชันแบบขนานบนคลัสเตอร์และบนกลุ่มเมฆในลักษณะการส่งเมสเสจได้

การทำงานแบบฟอร์กจอยน์ (fork/join) มีแนวคิดเรียบง่ายโดยมีที่มาจากอัลกอริทึมแบบแยกแล้วเอาชนะ (divide-and-conquer) ดังนี้

```
Result solve(Problem problem)
{ if (problem is small)
        directly solve problem
  else {
        split problem into independent parts
        fork new subtasks to solve each part
  join all subtasks compose result from subresults
  }
}
```

จากขั้นตอนวิธีดังกล่าวอธิบายได้ว่า ขั้นฟอร์กเป็นจุดเริ่มแบ่งงานย่อยโดยแตกเทรดทำงานย่อยแต่ละงานแบบขนาน ขั้นจอยน์เป็นจุดรวมของทุกเทรด จะรอจนกว่างานย่อยของแต่ละเทรดทั้งหมดเสร็จสมบูรณ์แล้วนำผลลัพธ์มารวมกัน อัลกอริทึมนี้จึงทำงานแบบวนรอบตลอดเวลา ตั้งแต่แตกงานย่อยซ้ำ ๆ กันรอจนกว่าจะแก้ปัญหาแต่ละส่วนจบในลักษณะแบ่งแยกและเอาชนะ เป็นเช่นนี้เรื่อย ๆ การนำวิธีการเช่นนี้มาใช้กับเทรดของจาวาโดยตรงถือเป็นงานที่ยาก เนื่องจากไม่สามารถลดโอเวอร์เฮดของเทรดปกติเพื่องานนี้งานเดียวได้จึงนำแนวคิดของ Cilk มาช่วยในการปรับ นั่นคือกำหนดให้สร้างเทรดที่ทำงานตามจำนวนของหน่วยประมวลผลกลางในแพลตฟอร์ม และตั้งค่าพารามิเตอร์การฟอร์กให้ตามแพลตฟอร์มมากกว่าตามอัลกอริทึมและกำหนดให้ข้อมูลอยู่บนหน่วยความจำภายใน [23]

คอนเคอร์เรนต์จาวา แยกแนวคิดของงานออกจากเทรดโดยงานคือส่วนที่ผู้ใช้กำหนดและงานจะถูกเทรดกำหนดให้ทำงานโดยมี *executor* เป็นผู้จัดคิวการทำงานของเทรด ลักษณะการทำงานแบบผู้ผลิต-ผู้บริโภค (Producer-Consumer) โดยผู้ผลิตป้อนงานเข้าไป และเทรดที่รับงานไป

ทำคือผู้บริโภค สิ่งที่บริโภคคืองานนั่นเอง นอกจากนั้นมีการปรับแต่ง *ThreadFactory* เพื่อลดการแทรกแซงจากไคลเอนต์หรืองานอื่น ประโยชน์ของการใช้ทั้งสองอย่างร่วมกันคือสามารถกำหนด thread pool หรือ thread group ได้โดยอัตโนมัติสำหรับกลุ่มเทรดที่ทำงานเฉพาะอย่างหนึ่ง และยังสร้างเทรดขึ้นใหม่ได้ตลอดเวลา [24]

การทำงานแบบแมพรีดิวซ์ การแบ่งแยกข้อมูลกันประมวลผลผ่านระบบเครือข่ายเป็นทางเลือกของการประมวลแบบขนานประเภทหนึ่ง ที่นิยมกันมากคือแมพรีดิวซ์ [25] การทำงานโดย worker สองประเภทได้แก่ Mapper และ Reducer โดยผู้ใช้กำหนดให้ฟังก์ชันแมพประมวลผล ทำการแมพอินพุตกับแต่ละ Mapper และให้ผลลัพธ์ในลักษณะคู่ลำดับคีย์-ค่า เซตคู่ลำดับนี้ขึ้นมาชุดหนึ่งหรือหลายชุด จากนั้น Reducer จะใช้ฟังก์ชันรีดิวซ์ เพื่อนำค่าที่สร้างขึ้นเหล่านี้มารวมกันเป็นคำตอบของปัญหาจากคีย์ที่กำหนด การประมวลงานนี้มักทำบนคลัสเตอร์ขนาดใหญ่ดังงานของบริษัทกูเกิลใน ค.ศ. 2004 ที่ใช้แมพรีดิวซ์หลายร้อยโปรแกรมทำงานมากกว่า 1,000 งานบนคลัสเตอร์ขนาดใหญ่ทุกวัน แบ่งการทำงานเป็น 7 ขั้นตอนดังนี้

1. ไลบรารีแมพรีดิวซ์ในโปรแกรมของผู้ใช้แบ่งไฟล์นำเข้าครั้งแรก ให้มีขนาดชิ้นละ 16 - 64 เมกะไบต์ โดยกำหนดเป็นพารามิเตอร์นำเข้าได้

2. มาสเตอร์ทำหน้าที่มอบหมายงาน เมื่อกำหนดให้ทำงานแมพ หรืองานรีดิวซ์ มาส-เตอร์จะคอยเก็บตัวรับงานที่ว่างแล้วมอบหมายให้ทำต่อไป

3. ตัวรับงานถูกกำหนดให้ทำแมพจะอ่านข้อมูลแล้วแยกไว้ เพื่อสร้างคู่ลำดับคีย์/ค่าออกจากข้อมูลเข้าตามที่ผู้ใช้กำหนดฟังก์ชันแมพ ส่วนคู่ลำดับคีย์/ค่าที่ถูกสร้างขึ้นมาโดยฟังก์ชันแมพจะถูกนำไปเก็บเป็นบัฟเฟอร์ในหน่วยความจำ

4. ในขณะเดียวกันคู่ลำดับในบัฟเฟอร์ถูกเขียนลงภายในดิสก์ แล้วส่งค่าที่อยู่บัฟเฟอร์นี้ให้แก่มาสเตอร์ ซึ่งจะส่งต่อไปให้หน่วยที่ทำรีดิวซ์ต่อไป

5. เมื่อหน่วยรีดิวซ์ทราบที่อยู่ข้อมูลนี้จากมาสเตอร์แล้วเพื่ออ่านข้อมูลบัฟเฟอร์ภายในเมื่ออ่านข้อมูลทั้งหมดแล้ว จะเรียงข้อมูลตามคีย์ชั่วคราวที่ติดอยู่ดังนั้นในขั้นนี้คีย์เดียวกันจะถูกนำมาอยู่กลุ่มเดียวกัน ในกรณีที่จำนวนของข้อมูลชั่วคราวที่สร้างขึ้นมาโดยอัตโนมัตินี้มีขนาดใหญ่มาก ต้องใช้การเรียงลำดับภายนอก

6. เมื่อหน่วยรีดิวซ์เรียงลำดับข้อมูลชั่วคราว คีย์ที่เหมือนกันจะถูกนับรวมกันจากนั้นจึงส่งค่าไปยังฟังก์ชันรีดิวซ์ของผู้ใช้ นำเอาต์พุตของรีดิวซ์มาต่อกับเอาต์พุตที่ได้จากการรีดิวซ์ก่อนหน้า

7. เมื่องานแมพและรีดิวซ์ทั้งหมดเสร็จสิ้นลง มาสเตอร์จะแจ้งผู้ใช้ผ่านโปรแกรม โดยเรียกฟังก์ชันแมพรีดิวซ์ในโปรแกรมกลับมาที่โค้ดของผู้ใช้นั่นเอง

ประโยชน์สำหรับงานแมพรีดิวซ์คือการทำดัชนีที่ทำให้จัดการได้ง่าย และขจัดปัญหาเครื่องหรือโหนดทำงานล้มเหลว เครื่องทำงานช้าหรือเครือข่ายมีปัญหาออกไป เนื่องจากมีกลไกที่ทนต่อความพร่องอยู่ เมื่อใช้แมพรีดิวซ์แล้วจะลดการการแทรกแซงจากงานอื่น นอกจากนี้ยังสะดวกในการเพิ่มเครื่องเข้ามาในคลัสเตอร์ด้วยโดยไม่ต้องแก้ไขโปรแกรม

การจัดการข้อมูลอาร์ดีเอฟบนหน่วยประมวลผลแบบขนาน

ข้อมูลที่ใช้ในงานวิทยานิพนธ์นี้มีลักษณะเป็นอักขระปริมาณมาก มีทั้งเชิงโครงสร้างและลักษณะไร้โครงสร้าง มีประเภทเป็นสตริง อาทิ ที่อยู่เว็บไซต์ ตัวอักษรบรรยาย เป็นต้นและวันที่ที่เกี่ยวข้องกับข้อมูลส่วนนั้น จึงต้องมีขั้นเตรียมข้อมูลก่อน โดยทำงานบนหน่วยประมวลผลกลาง แล้วในเคอร์เนลของหน่วยประมวลผลกราฟิกส์จึงเป็นการคิวรี่ข้อมูล สำหรับงานวิจัยที่คล้ายคลึงกันนั้นพบว่ามีจะการนำข้อมูลอักขระมาแปลงโดยการใช้ดัชนี ซึ่งมีโครงสร้างข้อมูลเช่น โครงสร้างต้นไม้ของตัวอักษร [26] กราฟ [27] แล้วแปลงข้อมูลเพื่อลดขนาด ข้อมูลนำเข้า ให้สามารถนำเข้าสู่หน่วยประมวลผลกราฟิกส์ได้ปริมาณมากที่สุด และใช้หมุนเวียนกันให้เกิดประโยชน์สูงสุด ดังนั้นในหัวข้อนี้จึงนำเสนอลักษณะข้อมูลที่ใช้ และการแทนค่าข้อมูลดังรายละเอียดต่อไปนี้

การลดขั้นตอนการทำดัชนีข้อมูล การนำเซตข้อมูลขนาดใหญ่มาใช้ในการประมวลผลแบบขนานมักทำดัชนีก่อนเพื่อให้เหมาะกับการประมวลผลแบบขนาน เพื่อประสิทธิภาพในการทำงานและลดพื้นที่ให้หน่วยความจำ แต่ขั้นตอนเตรียมข้อมูลนี้มักใช้เวลามาก จึงมีการวิจัยลดเวลาในส่วนการเตรียมข้อมูลนี้เช่น การทำแมพรีดิวซ์ [25] เพื่อสร้างต้นแบบประมวลเซตข้อมูลบนอินเตอร์เน็ตแบบกระจายบนคลัสเตอร์ขนาดใหญ่ โดยดัชนีมีคีย์ (key) และเลขเอกสารเป็นค่า (value) แต่มีประเด็นว่าการเตรียมข้อมูลต้องทำหลายขั้นตอนซึ่งใช้เวลานานและรับค่าที่ยังไม่ได้เรียงลำดับมาโดยตรงไม่ได้ ไม่มีการตั้งค่าให้หน่วยแมพสื่อสารระหว่างกันได้ ดังนั้นจึงสร้างพจนานุกรมทั้งหมดในขั้นตอนเดียวไม่ได้ จึงมีงานวิจัยนำแมพรีดิวซ์มาประยุกต์เหลือการส่งผ่านเพียงครั้งเดียว (single-pass) [28] โดยให้หน่วยแมพส่งคู่ลำดับ *<เทอม, รายการที่ส่งต่อ>* เพื่อลดจำนวนครั้งในการส่งและขนาดข้อมูลที่ส่งทั้งหมดระหว่างขั้นแมพและรีดิวซ์ ส่งผลให้ความเร็วของการประมวลเพิ่มขึ้นสัมพันธ์โดยตรงกับจำนวนของแกนและหน่วยประมวลผล แต่ยังต้องใช้ทรัพยากรจำนวนมากคือ 8 เครื่อง ซึ่งอย่างไรก็ตามยังมีการแมพรีดิวซ์แบบทำดัชนีจากข้อมูลขนาดใหญ่ [29] ด้วยการสลับคู่ลำดับ *<เทอม, ส่ง{เลขเอกสาร, ความถี่ของเทอม}>* เป็น *<ทัพเพิล{เทอม, เลขที่เอกสาร}, ความถี่ของเทอม>* ทำให้เหลือเพียงค่าเดียวในแต่ละคีย์ ส่งผลให้ทุกคีย์เป็นเอกลักษณ์และการันตีได้ว่าแมพส่งค่าไปขั้นรีดิวซ์ตามลำดับ ส่งผล

ให้ลิสต์ที่ส่งไปเรียงกันโดยไม่เหลือผลลัพธ์ไปยังรอบต่อไปอย่างไรก็ตามงานวิจัยนี้จัดการข้อมูลด้วยคลัสเตอร์ขนาด 99 โหนดจัดการกับข้อมูล ClueWeb09 ที่มีปริมาณเพจประมาณ 50 ล้านเพจนับว่าเป็นการใช้ทรัพยากรมากเช่นกัน

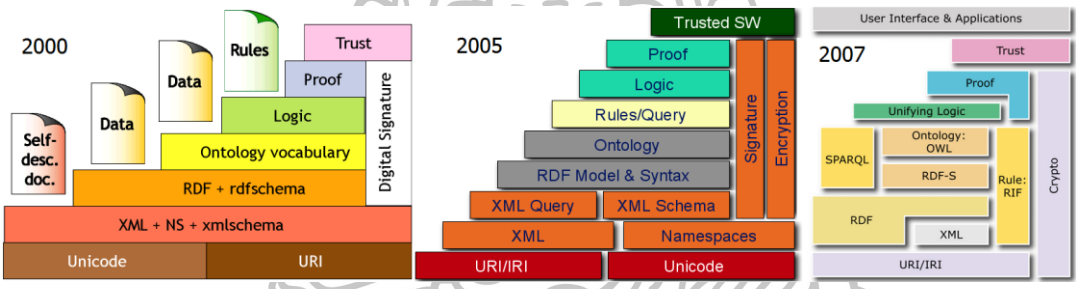
การแทนค่าข้อมูลด้วยโครงสร้างต้นไม้ของอักขระ การทำดัชนีข้อมูลและค้นหากับเซตข้อมูล ClueWeb09 ด้วยโครงสร้างต้นไม้ตัวอักษร ซึ่งต่างจากการเก็บข้อมูลแบบคีย์-ค่า คือใช้พื้นที่น้อยกว่า แต่ต้องสร้างไฟล์พจนานุกรมเพิ่มโดยทำงานบนหน่วยประมวลผลกลาง แล้วทำดัชนีแบบขนานร่วมกับหน่วยประมวลผลกราฟิกส์ ใน [26] สรุปผลการเลือกจำนวนหน่วยประมวลผลกลาง เพื่อใช้คำนวณแบบขนานเหมาะสุดคือ 8 แกน แบ่งเป็นแปลงข้อมูล 6 แกน ทำดัชนี 2 แกนและใช้หน่วยประมวลผลกราฟิกส์ 2 ตัวช่วยทำดัชนี ผลจากการทดสอบ พบว่ามีค่าธรูพุต 262.76 MB/s และเมื่อนำเทียบกับวิธีที่ใช้ 99 โหนดบนคลัสเตอร์นั้นธรูพุตมีค่า 167.37 MB/s ดังนั้นงานที่อาศัยหน่วยประมวลผลกราฟิกส์สามารถประมวลผลดีกว่าคลัสเตอร์ได้ ซึ่งเป็นการลงทุนที่น้อยกว่า

การสร้างดัชนีโครงสร้างต้นไม้ โดยนำแมพรีดิวซ์มาประยุกต์สร้างดัชนีต้นไม้บนหน่วยประมวลผลกราฟิกส์ ข้อมูลเข้าคือหน้าเว็บเพจจากเซตข้อมูล Clueweb09 โดยกำหนดขนาดบล็อกที่มีรายละเอียดของเอกสาร แล้วจึงอ่านข้อมูลเพื่อแปลงรหัส แล้วเขียนผลลัพธ์ชั่วคราวลงบนบัฟเฟอร์ ในขั้นนี้สามารถทำบนหน่วยประมวลผลกลางและกราฟิกส์และเก็บดัชนีไว้ การสร้างต้นไม้แบบไบนารีเหมาะกับข้อมูลที่ดึงจากหน้าเว็บพร้อมกับสร้างสมดุลแก่ต้นไม้โดยจับที่ n-key ใด ๆ ค่าความสูงคือ $\log_t \frac{n+1}{2}$ เมื่อ t เป็นดีกรีของต้นไม้ สำหรับโครงสร้างโดยรวมคือการนำต้นไม้ตัวอักษรมาเก็บต้นไม้ไบนารีที่สร้างไว้ ส่วนของต้นไม้อักษรนั้น มีการสร้างตารางแทนค่าอักขระตามจำนวนอักษร 3 ตัวแรกความเป็นไปได้ =26*26*26 = 17,576 กรณี จากนั้นกำหนดโครงสร้างต้นไม้ไบนารีภายในต้นไม้อักขระที่มีขนาดข้อมูลทั้งหมด 512 ไบต์ สรุปว่าการสร้างดัชนีบนหน่วยประมวลผลกลางนั้นสร้างต้นไม้อักขระไปถึงเทรดที่ 17,612 เมื่อทำงานบนหน่วยประมวลผลกราฟิกส์ สามารถเรียงสายอักขระโดยมีความยาวเทอมนำหน้าตามด้วยสายอักขระของเทอมนั้นแบบต่อเนื่องกันชุดละ 512 ไบต์สำหรับการขยับเทอมเพื่อทำการค้นหานั้นสามารถกำหนดให้เทอมเป้าหมายเป็นเทรด *i+1* ถ้าค้นพบเทอมเป้าหมายให้ปรับค่า ถ้าไม่มีเทอมนี้อยู่ทั้งรากและใบให้สร้างโหนดใหม่ แต่ถ้าพบที่ใบแล้วเมื่อต้องการทำกิจกรรมต่อให้ทำที่ใบของโหนดนั้น

การสร้างดัชนีโครงสร้างกราฟ จำเป็นต้องกำหนดโครงสร้างของแถวลำดับเช่นกัน (structure of array) [30] กรณีที่มีกราฟและทาบโครงสร้างนั้นไปยังหน่วยประมวลผลกราฟิกส์ 3 ตัว ต้องแบ่งข้อมูลออกเป็นต้นไม้ 3 ส่วน คัดลอกข้อมูลส่วนที่สัมพันธ์กันขึ้นมาอีกชุดหนึ่ง ส่วนมากคือหางของต้นไม้จากการ์ดตัวแรก ซึ่งหางนั้นจะอยู่ทุกหน่วยประมวลผลกราฟิกส์ที่มีข้อมูลสัมพันธ์กัน

ข้อมูลจากลิงค์เดต้า

เว็บเชิงความหมาย หรือ เว็บของข้อมูล (web of data) เป็นส่วนขยายของเว็บดั้งเดิมที่องค์กรเวิลด์ไวด์เว็บสากล (W3C) กำหนดมาตรฐานไว้ ให้เป็นข้อมูลเปิดแบบหนึ่งที่จัดการได้เหมือนฐานข้อมูล เทคโนโลยีนี้ช่วยให้มนุษย์สร้างที่เก็บข้อมูลบนเว็บได้โดยตรง โดยเพิ่มคำบรรยายและเขียนกฎจัดการความหมาย โดยสนับสนุนคอมพิวเตอร์ให้วิเคราะห์ความหมายข้อมูลในเว็บได้มากขึ้นและมีความน่าเชื่อถือตามระดับความหมายและความสัมพันธ์ระหว่างแต่ละเซตข้อมูล ต่อมาข้อมูลที่สัมพันธ์กันนี้ถือเป็นลิงค์เดต้า ที่อยู่บนพื้นฐานเทคโนโลยีข้อมูลอาร์ดีเอฟมากขึ้นเช่น OWL (web ontology language) และ SKOS (Simple Knowledge Organization System)



รูปที่ 2.8 ระดับชั้นของเว็บเชิงความหมาย ปีค.ศ. 2000 ค.ศ. 2005 และ ค.ศ. 2007

จากรูปที่ 2.8 แสดงระดับชั้นของเว็บเชิงความหมาย ปีค.ศ. 2000 ค.ศ. 2005 และ ค.ศ. 2007 [31] [32] [33] รายละเอียดของชั้นการทำงานของเว็บเชิงความหมายนั้นมาจากกิจกรรมของ W3C [32] มาประยุกต์เป็นระดับชั้น โดยเพิ่มกฎหรือตรรกศาสตร์มากขึ้น และได้มีการปรับปรุงระดับชั้นปี ค.ศ. 2000 และ ค.ศ. 2005 ไว้ในงานวิจัย [34] โดยมีวิวัฒนาการของชั้นจากแบบเริ่มต้น [31] ที่เพิ่มส่วนของการเข้ารหัส [35] ต่อมาได้มีการปรับอาร์ดีเอฟให้เป็นข้อมูลพื้นฐานมากขึ้นและมีการรวมกฎและหลักตรรกศาสตร์ให้เป็นส่วนที่ชัดเจนยิ่งขึ้น [33] ในปัจจุบันแนวโน้มการนำชั้นเหล่านี้ไปใช้งานจะสอดคล้องกับภาพของแนวคิด “เว็บเป็นหนึ่งเดียว” ของ W3C [36] สามารถสรุปตามหน้าที่ของแต่ละชั้นและเพิ่มเติมในส่วนของการนำไปใช้เป็นแอปพลิเคชันเว็บได้ดังต่อไปนี้

ชั้นการเข้ารหัส จัดการเรื่องความปลอดภัยที่ครอบคลุมข้อมูลทุกระดับชั้น เน้นควบคุมลิขสิทธิ์ของข้อมูลเปิด ทั้งข้อมูลที่สร้างขึ้นภายใต้ลิขสิทธิ์เปิดและข้อมูลที่เปิดเผย จากผู้ให้ข้อมูลที่น่าเชื่อถือ ด้วยเหตุนี้ผู้ใช้หรือข้อมูลอื่น ๆ ที่มาเชื่อมต่อจึงสามารถนำข้อมูลไปใช้ตามต้องการอย่างถูกกฎหมาย เริ่มตั้งแต่การใช้งานระหว่างไฟล์ข้อมูล จนถึงการแลกเปลี่ยนข้อมูลข้ามเครือข่าย

ชั้นกำหนดตัวตน คือการระบุที่อยู่ของแหล่งข้อมูลเพื่อแสดงว่าหาข้อมูลนั้นได้จากที่ใด ภายในระบบอินเตอร์เน็ต และตัดสินว่าเนมสเปซของข้อมูลนั้นหาได้จากชื่อโดเมนใด

ชั้นคำศัพท์ [37] กำหนดรายการและคำจำกัดความของเทอมที่อธิบายเนื้อหาเซตข้อมูล

ชั้นออนโทโลยีเป็นคำศัพท์ที่เก็บความสัมพันธ์เชิงตรรกะระหว่างเทอม [37]

ชั้นแลกเปลี่ยนข้อมูลสามารถครอบคลุมชั้นกำหนดตัวตน ชั้นกำหนดศัพท์ และชั้นออนโทโลยีได้ มีรูปแบบอาร์ดีเอฟเป็นสำคัญ สื่อสารข้อมูลในรูปแบบทริพเพิล สามารถเรียงต่อเนื่องกันได้ แม้ว่าอยู่ในไฟล์ต่างชนิดกัน

นิยามที่ 2.1 ทริพเพิล เทอม และกราฟของอาร์ดีเอฟ

กำหนดให้ $\mathcal{U}$ เป็นเซตของ URI, B เป็นเซตของโหนดว่าง (blank node) และ L เป็นเซตของข้อความ อาร์ดีเอฟหนึ่งทริพเพิลประกอบด้วยเซตของ (Subject, Predicate, Object) $\in \mathcal{U}\times(\mathcal{U}\cup\mathrm{B})\times(\mathcal{U}\cup B\cup L)$ เทอมของอาร์ดีเอฟคือเอเลเมนต์ของ $\mathcal{U}\cup B\cup L$ กราฟอาร์ดีเอฟคือเซตที่ประกอบด้วยอาร์ดีเอฟจำนวนมาก

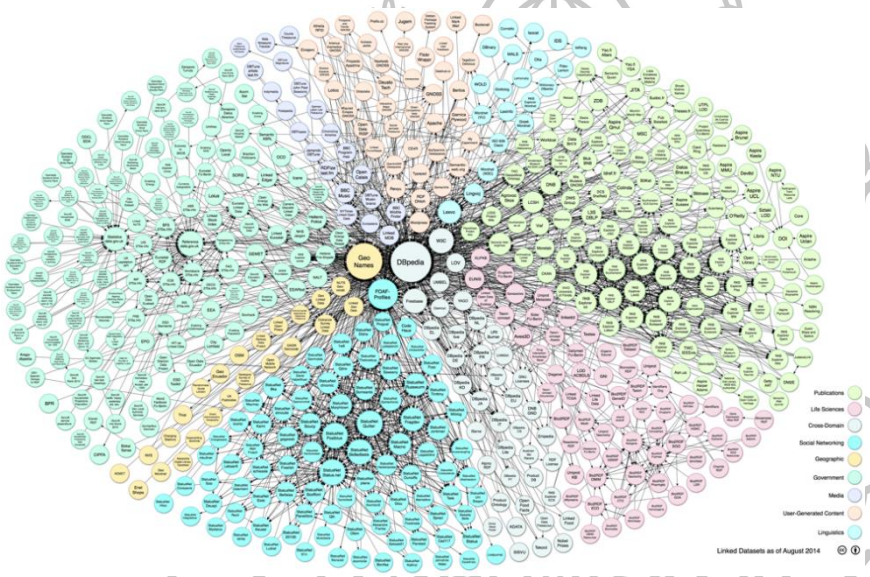
ชั้นคิวรี่ใช้ภาษาสปาร์เคิล (SPARQL) เป็นภาษาสำหรับคิวรี่และเป็นโปรโตคอลของอาร์ดีเอฟ [38] ทำหน้าที่หาเส้นทางเพื่อประมวลคิวรี่ผ่านเซตข้อมูลแบบลิงค์เดต้า และผ่านการพัฒนามาถึงรุ่น 1.1 ที่มีการเพิ่มเรื่องกราฟอาร์ดีเอฟ (Graph Store) มากขึ้น [39]

นิยามที่ 2.2 เทอมของการคิวรี่

เทอมที่ต้องการคิวรี่อาร์ดีเอฟ อยู่ที่ตำแหน่งใดให้แทนค่าด้วยพรีฟิกส์ ‘?’ ก่อนตัวแปร

ลิงค์เดต้า หรือเทคโนโลยีการเผยแพร่ข้อมูลที่เชื่อมต่อกัน [40] เป็นข้อมูลที่มีความสัมพันธ์กันและเชื่อมกันระหว่างเซตข้อมูล เป็นสิ่งตรงข้ามกับการเก็บฐานความรู้เดียว จึงเรียกว่าลิงค์เดต้า สำหรับข้อมูลที่เก็บอย่างน้อยต้องอยู่ในรูปแบบทริพเพิลสื่อความหมายระดับอาร์ดีเอฟ ที่เป็นภาษามาร์กอัพอื่นเช่น เอ็กซ์เอ็มแอล (XML) ได้ ในปัจจุบันนี้ลิงค์เดต้าถือเป็นหัวใจของเว็บเชิงความหมาย เนื่องจากเป็นการบูรณาการข้อมูลขนาดใหญ่ร่วมกันหลายชุดข้อมูล มีการเพิ่มขึ้นอย่างรวดเร็วและแสดงความหมายทางตรรกศาสตร์ได้ เป็นประโยชน์ต่อผู้พัฒนาแอปพลิเคชันด้านเว็บเชิงความหมายสามารถเลือกใช้ตามระดับความซับซ้อนที่ต้องการ ทำให้การทำงานของเว็บเชิงความหมายไร้พรมแดน มีกลุ่มผู้สนับสนุนโครงการเชื่อมต่อข้อมูลเปิดมากมายทั้งบุคคลทั่วไปและองค์กร ตามนิยามของทิมเบิร์นเนอร์ส-ลี ได้แบ่งระ-ดับข้อมูลเปิดออกเป็น 5 ระดับ ซึ่งข้อมูลประเภทอาร์ดีเอฟถือเป็นข้อมูลเปิดที่ใช้เชื่อมระหว่างกันอยู่ในระดับที่ 5 หมายความว่าเซตข้อมูลที่สนใจนั้นเชื่อมโยงข้อมูลที่มีอยู่กับเซตข้อมูลอื่นเพื่อบอกบริบท

จากรูปที่ 2.9 พบว่า DBpedia [41] คือศูนย์กลางของลิงค์เดต้า โดยเป็นการสร้างอาร์ดีเอฟจากเนื้อหาของวิกิพีเดียจึงทำให้มีรายละเอียดภายในมากมาย สำหรับลักษณะของ DBPedia ที่ใช้ในการทดลองคือเซตข้อมูลที่เชื่อมต่อกับลิงค์ภายนอกด้วย การเชื่อมต่อลิงค์ภายนอกมี 2 แบบได้แก่ แบบที่ 1 ลิงค์ของ HTML ที่ออกสู่เว็บเพจภายนอก ได้แก่ dbpedia:reference ชี้ไปที่เว็บใด ๆ เพื่ออ้างถึงสิ่งต่าง ๆ โดยละเอียด และ foaf:homepage ที่ชี้ไปที่หน้าเว็บที่เป็นทางการ ส่วนแบบที่ 2 ลิงค์ของอาร์ดีเอฟที่ชี้ไปยังแหล่งของข้อมูล ตามคุณสมบัติของ OWL คือ owl:sameAs สามารถอ้างอิงไปถึงเซตข้อมูลที่เชื่อมกันได้ ตัวอย่างลิงค์ของอาร์ดีเอฟที่ชี้ไปยังแหล่งของข้อมูลอื่นเช่น GeoNames Yago DBLP Freebase และ U.S. Census เป็นต้น



รูปที่ 2.9 แผนภาพ Linking Open Data (LOD) Project Cloud ที่ได้รับการรวบรวมจนถึงก.ย. 2011

ที่มา: Cyganiak, R., **Linking Open Data (LOD) Project Cloud** [ออนไลน์], เข้าถึง 25 พ.ค. 2012, http://richard.cyganiak.de/2007/10/lod/lod-datasets_2011-09-19.png

เรื่องขนาดและความสัมพันธ์ของลิงค์เดต้า [42] จากรูปที่ 2.9 ขนาดของวงกลมสัมพันธ์กับจำนวนเซตของทริพเพิล โดยวงกลมขนาดใหญ่มากจะมีจำนวนมากกว่า 1 พันล้านทริพเพิล ส่วนวงกลมที่ขนาดใหญ่จะมี 10 ล้านถึง 1 พันล้านทริพเพิล ส่วนวงกลมขนาดกลางมี 1 แสนถึง 10 ล้านทริพเพิล วงกลมขนาดเล็กอยู่ที่ระหว่าง 10,000 – 500,000 ทริพเพิลและวงกลมข้อมูลเปิดที่ขนาดเล็กมากจะมีจำนวนน้อยกว่า 10,000 ทริพเพิล ซึ่งขนาดเหล่านี้วัดโดยผู้จัดทำและจากการประมาณการในบางครั้ง นอกจากนี้อาจขยายขนาดของข้อมูลได้เสมอเพราะเป็นรูปแบบเดียวกัน ดังนั้นจึงสามารถนำความสัมพันธ์มาเชื่อมต่อกันได้ ความสัมพันธ์ระหว่างเซตข้อมูลพิจารณาที่ทิศทางของลูกศร ทิศทางเดียวใช้แทนความสัมพันธ์ทางเดียวเช่นลูกศรจาก A ถึง B หมายถึงเซตข้อมูล A มีอาร์ดีเอฟที่

ใช้ใน B ได้ด้วย ลูกศรสองทิศทางใช้เชื่อมต่อระหว่างกันของ 2 เซตข้อมูล ส่วนความหนาของลูกศรสัมพันธ์กับจำนวนลิงค์ ถ้าหนามากคือลิงค์สัมพันธ์กันมากกว่า 100,000 ทริพเพิล หนาปานกลางแทนลิงค์ที่สัมพันธ์กันระหว่าง 1,000 ถึง 100,000 ทริพเพิลและลูกศรบางแทนจำนวนที่สัมพันธ์กันน้อยกว่า 1,000 ทริพเพิล [40] การวัดค่าของจำนวนเซตข้อมูลแบบที่สองนี้ไม่สามารถวัดจากโปรแกรมจัดการสร้างออนโทโลยีเช่น โพรทีเจ [43] ได้ดังนั้นจากความรู้ข้างต้นจึงใช้การรวบรวมข้อมูลการวัดค่าจากแหล่งเผย แพร่ฐานความรู้แทนซึ่งสามารถประมาณการขนาดเซตของข้อมูลได้

การนำพื้นฐานของเว็บเชิงความหมายมาประยุกต์เพื่อสร้างแอปพลิเคชันเว็บนั้นมีหลายด้าน ซึ่งสรุปตามแนวทางเว็บเป็นหนึ่งเดียวของ W3C ได้ดังนี้

ชั้นเก็บข้อมูลเพื่อเผยแพร่ โดยทั่วไปมักเก็บอยู่ในรูปของไฟล์บนเซิร์ฟเวอร์ของเว็บ หรือในฐานข้อมูลผ่านชั้นการแปลงข้อมูลตัวอย่างเช่น การแปลงจากทริพเพิลเป็นการเก็บแบบทัพเพิล [41] สามารถแบ่งปันสิทธิ์การใช้งานลิงค์เดต้าได้

ชั้นส่งผ่านข้อมูลคือการส่งผ่านโปรโตคอล HTTP เนื่องจากข้อมูลเก็บอยู่ที่เซิร์ฟเวอร์จึงต้องเรียกและดูข้อมูลระหว่างกันผ่านระบบอินเตอร์เน็ตได้

ชั้นแอปพลิเคชัน เน้นเว็บแอปพลิเคชันสามแบบคือ แบบที่หนึ่ง การสร้างเครื่องมือค้นหาข้อมูลของลิงค์เดต้าผ่านเว็บเชิงความหมาย เพื่อให้ค้นหาได้โดยสะดวกและแสดงข้อมูลที่ลิงค์เดต้าเก็บไว้ได้ แบบที่สองเป็นการผลิตลิงค์เดต้า เป็นการอำนวยความสะดวกเพื่อการบูรณาการ (integration) ข้อมูลสำหรับระบบธุรกิจและงานวิจัย แบบที่สามคือแมชอัพ (mashups) เป็นการรวมหลายเซตข้อมูลเข้าด้วยกันเพื่อสร้างเซอร์วิสแบบใหม่ เพื่อดูภาพรวมหรือได้ข้อมูลขอบเขตใหม่เพิ่มขึ้น

รูปแบบข้อมูลประเภทออนโทโลยี เป็นคำศัพท์ที่ใช้กำหนด นิยามแนวคิดและความสัมพันธ์ เรียกได้ว่าเทอม ใช้เพื่อบรรยายและแสดงขอบเขตของสิ่งที่เกี่ยวข้อง ชุดคำศัพท์นี้ถูกนำมาใช้เพื่อแบ่งกลุ่มของเทอมตามแอปพลิเคชันเฉพาะ ตามคุณลักษณะของความสัมพันธ์และนิยามกฎที่เป็นไปได้สำหรับเทอมที่นำมาใช้ รายละเอียดคำศัพท์มีตั้งแต่ใช้บรรยายเพียงหนึ่งหรือสองแนวคิดหรือมีความซับซ้อนมากในระดับหลายพันแนวคิดก็เป็นได้

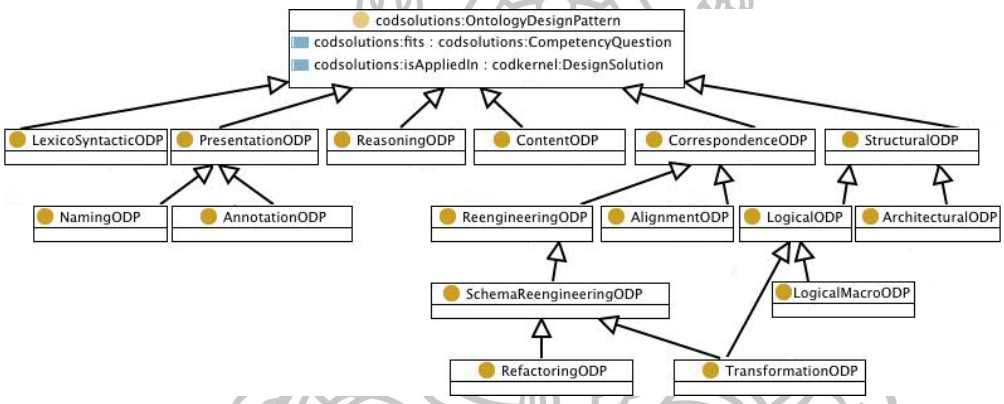
การออกแบบออนโทโลยีมีหลายประเภท ซึ่งมีการสรุปได้ 6 รูปแบบตามงานวิจัยด้านการออกแบบ [44] และต่อมาได้ปรับปรุงรายละเอียดเป็นดังรูปที่ 2.10 จึงอธิบายได้ดังต่อไปนี้

การออกแบบโครงสร้าง (Structure ODPs) ประกอบด้วย*การออกแบบเชิงตรรกะ*ที่สอดคล้องทั้งเนื้อหาและตรรกศาสตร์และ*การออกแบบสถาปัตยกรรม*ที่รักษาสมดุลรูปแบบภายนอกและในออนโทโลยีว่าแต่ละเทอมมีความสมเหตุสมผลตามหลักตรรกศาสตร์เชิงบรรยายหรือไม่

การออกแบบความสอดคล้องกัน (Correspondence ODPs) ประกอบด้วย การสร้างแนวคิดใหม่ เพื่อปรับแนวความคิดและการออกแบบจากการจับคู่เพื่อสร้างความหมายใหม่หรือรวมความ หมายร่วมกันจากออนโทโลยีอย่างน้อยสองฐานที่มีอยู่

การออกแบบเชิงเนื้อหา (Content ODPs) เป็นการแปลงรหัสจากเชิงความคิดมากกว่าด้านตรรกะ ตัวอย่างคือการแก้ปัญหาการคลาสโดเมนและคุณสมบัติขึ้นใหม่ จึงมีการสร้างบต้นแบบ - (building blocks) เพื่อให้นำกลับมาใช้ใหม่ได้

การออกแบบเชิงเหตุผล (Reasoning ODPs) เป็นแอปพลิเคชันเชิงตรรกะที่แสดงผลลัพธ์ที่เป็นเหตุผลต่อกัน โดยใช้เครื่องมือตรวจความเป็นเหตุผล (Reasoning engine)



รูปที่ 2.10 ภาพของประเภทการออกแบบออนโทโลยี (Ontology Design Patterns)

ที่มา: EvaBlomqvist, **Ontology Design Patterns** [ออนไลน์], http://ontologydesignpatterns.org/wiki/Image:Odptypes.jpg, เข้าถึง 31 ต.ค. 2013

การออกแบบไวยากรณ์ (Lexico-Syntactic ODPs) เป็นการออกแบบตามโครงสร้างทางภาษา ที่ประกอบด้วยคำตามลำดับ เริ่มจากคำที่มีความหมายครอบคลุม และการดึงความหมายเพื่อสรุปสิ่งที่ต้องการสื่อสารมักใช้งานร่วมกับ*การออกแบบเชิงตรรกะและเนื้อหา*พร้อมกับภาษาธรรมชาติเพื่อประโยชน์ด้านสื่อการสอน

การออกแบบเพื่อนำเสนอ (Presentation ODPs) งานวิจัย [44] ให้ความหมายเพิ่มเติมจากโครงการนีออนว่าเป็นส่วนจัดการการอ่านเพื่อผู้ใช้งานจริง จึงมีการออกแบบเพิ่มเติมอีก 2 ประเภทคือ การตั้งชื่อ และการใส่รายละเอียด (Annotation ODPs)

การเก็บข้อมูลและลักษณะข้อมูล

การพิจารณารูปแบบของข้อมูลที่ต้องการใช้งานก่อนนำเข้าเป็นเซตข้อมูลเพื่อการคิวรี่ในหน่วยประมวลผลกราฟิกส์นั้น ต้องทำการศึกษาเพื่อต่อยอดและทดสอบจากงานวิจัยที่มีการพิสูจน์

เรื่องการลดขนาดข้อมูลประเภทออนโทโลยีโดยเฉพาะ เนื่องจากต้องคงความถูกต้องของข้อมูลแบบออนโทโลยี อย่างน้อยต้องเป็นระดับอาร์ดีเอฟ ที่ใช้แลกเปลี่ยนกันของลิงค์เดต้า สำหรับรายละเอียดของการเก็บข้อมูลแบ่งเป็นการเข้ารหัส สร้างดัชนี แล้วเก็บลงฐานข้อมูลแบบทัพเพิลการแปลงค่าเป็นบิตตามดัชนีแล้วโหลดทั้งหมดเข้าหน่วยความจำ และการเก็บข้อมูลลงไฟล์ย่อแบบไบนารี

การจัดเก็บฐานออนโทโลยีลงบนดิสก์ รายละเอียดและความหมายแต่ละฐานออนโทโลยีที่ศึกษาแบ่งตามชื่อของฐานได้ดังต่อไปนี้

ทีดีบี (TDB) เป็นส่วนประกอบของอาปาเช่ เจนา (Apache Jena) มีหน้าที่เก็บข้อมูลทริพเพิลลงสู่ดิสก์โดยตรงและรองรับการคิวรี่ [45] ปัจจุบันได้มีการนำทีดีบีมาใช้ในการเก็บอาร์ดีเอฟด้วยประสิทธิ ภาพสูงบนเครื่องเดี่ยวมากขึ้น รองรับทั้งภาษาสปาร์เคิลเดิมและรุ่นปรับปรุง ในการทำงานแอปพลิเคชันจะได้รับโมเดลหรือเซตข้อมูลอาร์ดีเอฟจากทีดีบี แล้วจึงแปลงเป็นโมเดลหรือเซตข้อมูลอื่นต่อไป ทีดีบียังรองรับการทำงานแบบธุรกรรม (transaction) เนื่องจากมีคุณสมบัติที่ทำงานพร้อมเพรียงกัน สำ หรับตารางเป็นลักษณะของทัพเพิลที่เรียกว่าตารางคุณสมบัติ เก็บดัชนีในรูปต้นไม้ B+ ได้3 แบบคือ *SPO, POS* และ *OSP* ซึ่งอักษร *S* คือ subject อักษร *P* คือ predicate และอักษร *O* คือ object นอกจากนี้ยังมีการพัฒนาทีดีบีเพิ่มให้ทำงานแบบขนานบนคลัสเตอร์ได้ [46]

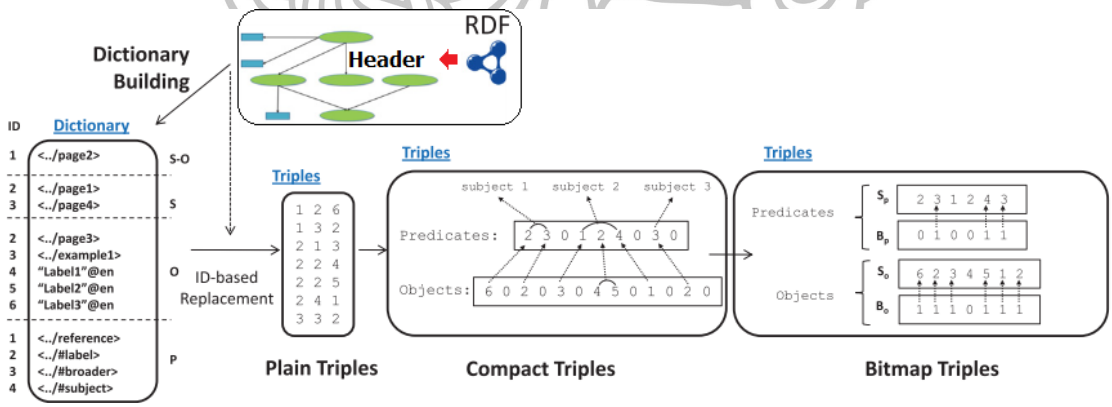
คิวมิวลัสอาร์ดีเอฟ (cumulusRDF) [47] เป็นฐานข้อมูลสำหรับอาร์ดีเอฟบนกลุ่มเมฆโดยเก็บดัชนีแบบคีย์-ค่า (key-value) ตามคุณสมบัติของฐานข้อมูลคาสซานดรา (Apache Cassandra) คุณสมบัติที่สำคัญคือแปลงข้อมูลจากทริพเพิลเอ็นธรี (N3) มาเป็นทัพเพิล ผ่านไลบรารีเฮคเตอร์จากไคลเอนต์ (Hector) มีการเก็บดัชนีหลายแบบซึ่งผลการทดสอบที่มีประสิทธิภาพมากที่สุดได้แก่ดัชนี 3 ชุดคือ *SPO* (*S* เป็นคีย์ *PO* เป็นค่า) *OSP* (*O* เป็นคีย์ *SP* เป็นค่า) และ *POS* (*PO* รวมกันเป็นคีย์หลัก *P* และ *O* เดี่ยวเป็นคีย์รอง และ *S* เป็นค่า) เมื่อผู้ใช้ต้องการแบ่งเก็บฐานอาร์ดีเอฟทำได้โดยระบุโทเคน (token) ของข้อมูลที่ต้องการแบ่งและระบุที่อยู่ของเครื่องถัดไป การประมวลเพื่อเก็บข้อมูลจะเป็นแบบหมุนวน (round robin) ตามลำดับ

การจัดการอาร์ดีเอฟบนหน่วยความจำ บิตแมท (BitMat) [48] เป็นรูปแบบการประมวลผลโดยใช้คลัสเตอร์ขนาดใหญ่ มีการเตรียมโดยแปลงข้อมูลให้มีขนาดเล็กลงเป็นเมตริกซ์ของบิต การประมวลผลทั้งหมดต้องอาศัยหน่วยความจำที่อยู่บนเครื่องนั้น แต่ละทริพเพิลอยู่ในรูปสามมิติแบบลูกบาศก์บิตได้ ซึ่งแต่ละเซลคือ 1 ทริพเพิลเมื่อนำเข้าสู่หน่วยความจำหลัก ลูกบาศก์เหล่านี้จะถูกสไลด์เป็นสองมิติ ด้วยเหตุนี้จึงทำบิตแมทไว้ 6 รูปแบบ โดยที่แต่ละโครงสร้างสามารถทำการจอยน์ 1 ครั้ง จึงเป็นเซตของทริพเพิล ตามหลักการสปาร์สเมตริกซ์ จะเขียนเป็นแถวลำดับของแถวบิตได้ ซึ่งแต่ละแถวจะเก็บชุดของทริพเพิลไว้ด้วย *S* เดียวกัน ขั้นตอนในอนาคตคือลดขนาดของข้อมูลที่เก็บได้และทำ

คิวรี่แบบมัลติเพิลจอยน์ได้ คิวรี่เหล่านี้ใช้โอเปอเรชันของบิตคือ AND และ OR บนแถวของบิตแมท แล้วผลลัพธ์เป็นบิตแมทอีกชุดหนึ่ง การเก็บข้อมูลบนหน่วยความจำนี้ถูกออกแบบให้อ่านอย่างเดียว

รูปแบบอาร์ดีเอฟแบบไบนารี เพื่อลดขนาดการเก็บข้อมูลบนไฟล์ สำหรับเผยแพร่และแลกเปลี่ยนข้อมูลแบบเอชดีที (HDT: Header-Dictionary-Triples) เป็นการย่อข้อมูลได้เล็กกว่าต้นฉบับประ มาณร้อยละ 15 และย่อให้ลดลงอีกร้อยละ 3.5 การย่อแบ่งเป็นสามส่วน ส่วนแรกคือ หัว (Header) จะเก็บขนาดข้อมูลเดิมและกราฟอาร์ดีเอฟในต้นไม้สำหรับโหนดระบุตัวตน (ID) แล้วจะถูกส่งผ่านระบบอินเตอร์เน็ตไปที่ไคลเอนต์ด้วยเมื่อผู้ใช้เรียกดู และคำนวณดีกรีจากกราฟ Subject $\xrightarrow{\text{Predicate}}$ Object ส่วนที่สอง คือ พจนานุกรม จะจัดการ URI ที่ซ้ำกันแล้วใช้จัดลำดับตามอักขระ ส่วนที่สามคือทริพเพิลเป็นเซตของบิตแมพ (Bitmap) ที่ย่อลงอีกด้วยรหัสฮัฟแมน [49] จาก

รูปที่ 2.11 แสดงขั้นตอนการสร้างเอชดีที เริ่มจากการนำอาร์ดีเอฟมาสรุปข้อมูลเก็บไว้ที่หัว (Header) แล้วสร้างไฟล์พจนานุกรมเพื่อเตรียมเข้ารหัสทริพเพิล จากนั้นสร้างทริพเพิลสามรูปแบบคือแทนค่าจากเลขระบุตำแหน่งโดยตรง (plain triples) จากนั้นรวมกลุ่มย่อของ predicate และ object ตามเลขระบุ subject เป็นทริพเพิลที่ย่อแล้ว (compact triples) และสร้างต้นไม้ตามดัชนีบิตแมพ แบ่งเป็นส่วนของ *Subject* ที่ถือว่าเป็นตัวนับจำนวนต้นไม้ที่เกิดขึ้นทั้งหมด จากนั้นจึงเป็นคู่ลำดับของ (*Predicate, Object*)



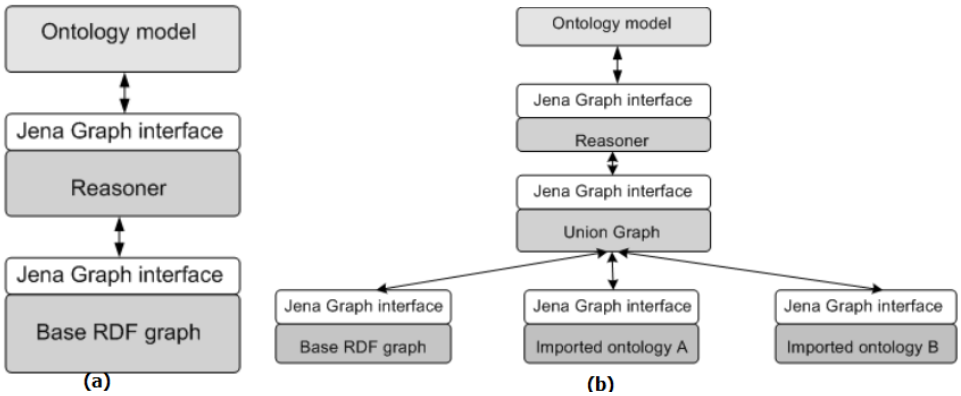
รูปที่ 2.11 ขั้นตอนการสร้างรูปแบบอาร์ดีเอฟให้เป็นรูปแบบย่อเอชดีที

ที่มา: J. D. FernáNdez, M. A. MartíNez-Prieto, C. GutiéRrez, A. Polleres, and M. Arias. (2013). **Binary RDF representation for publication and exchange (HDT).** Web Semant. vol. 19. pp. 30.

มิดเดิลแวร์

หลังจากผู้ใช้งานได้นำไฟล์อาร์ดีเอฟมาจากแหล่งที่น่าเชื่อถือแล้ว อาจสร้างเพิ่มโดยใช้เครื่องมือออกแบบและพัฒนาออนโทโลยีด้วยโพรทีเจ หรือโฮโสะ (Hozo) [50] หรือสร้างเองผ่านเครื่องมือดังกล่าวโดยตรง แล้วจึงนำมาเชื่อมต่อกับโปรแกรมที่พัฒนาขึ้น ในกรณีที่พัฒนาร่วมกับโปรแกรมภาษาจาวา มีอีกแนวทางหนึ่งที่สามารถใช้พัฒนาออนโทโลยีโมเดลผ่านโปรแกรมที่พัฒนาขึ้นได้ ด้วยการใช้ชุดโปรแกรมของเจนา ที่ได้รับการพัฒนาจากแลบของบริษัท เอชพี (HP) [51] ในภายหลังกลายเป็นชุดรหัสเปิดที่พัฒนาภายใต้อาปาเช่ กลายเป็นอาปาเช่ เจนา (Apache Jena) [52] ซึ่งใช้ในงานวิจัยนี้ ซึ่งการทำงานของชุดพัฒนารูปที่ 2. 12 (a) ประกอบด้วยส่วนสร้างโมเดลออนโทโลยีเชื่อมต่อกับส่วนตรวจสอบเหตุผลและกราฟอาร์ดีเอฟด้วยอินเตอร์เฟสกราฟของเจนา ส่วน (b) เป็นกรณีที่ใช้ชุดพัฒนาทำงานกับออนโทโลยีหลายชุด ซึ่งผู้พัฒนาอาจนำเข้ามาพัฒนาร่วมกันได้ โดยเจนาได้เปลี่ยนจากกราฟอาร์ดีเอฟที่เป็นฐานเดี่ยวไปสู่การรวม (union) กับฐานกราฟออนโทโลยีอื่น ๆ ที่มีการนำมาพัฒนาร่วมกันในโปรแกรม

ชุดพัฒนาของเจนา ประกอบด้วยสามส่วนคือ ส่วนพัฒนาเชื่อมกับอาร์ดีเอฟ คือ API ของอาร์ดีเอฟและเออาร์คิว (ARQ) โดยที่ API เป็นแกนหลักเชื่อมต่อ สร้าง และอ่านกราฟอาร์ดีเอฟ รวมถึงรองรับรูปแบบเอ็กซ์เอ็มแอลของอาร์ดีเอฟ ในส่วนเออาร์คิว ทำหน้าที่คิวรี่ภาษาสปาร์เคิล รวมถึงรองรับการคิวรี่ผ่านฐานระยะไกลและค้นหาอักขระ ส่วนเชื่อมต่อกับฐานทริพเพิล ประกอบด้วยส่วนของทีดีบี (TDB) ที่จัดการโมเดลตลอดช่วงการทำงานของ API ของเจนา และส่วนของฟูเซกิ (Fuseki) ที่มีหน้าจอสำหรับคิวรี่สปาร์เคิลผ่าน HTTP ได้เลยนอกจากนี้ยังรองรับบริการแบบ REST



รูปที่ 2. 12 อินเตอร์เฟสกราฟของเจนาที่เชื่อมระหว่างฐานอาร์ดีเอฟส่วนตรวจสอบและโมเดลที่พัฒนา

ที่มา: Apache Jena, **Jena Ontology API** [ออนไลน์], https://jena.apache.org/documentation/ontology/, เข้าถึงเมื่อ 20 ก.ค. 2557

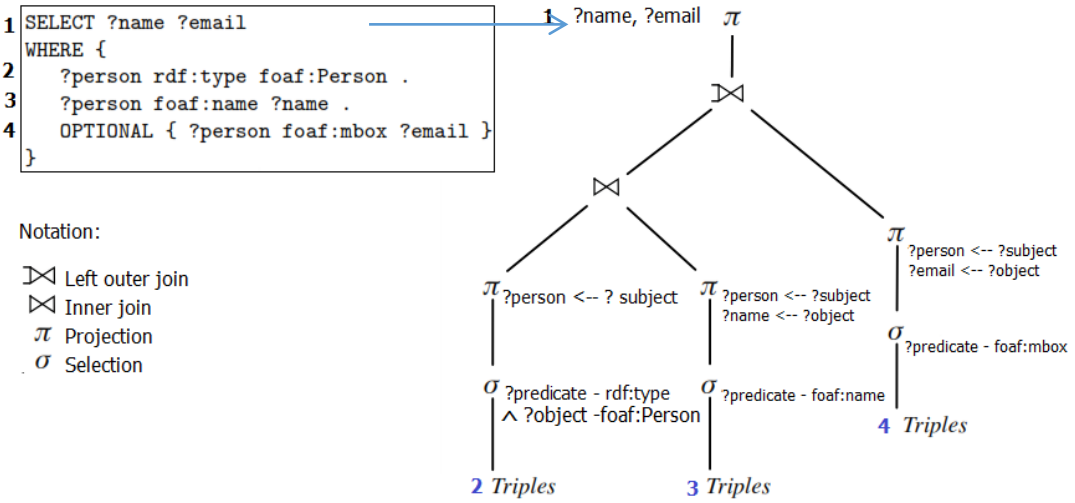
ส่วนที่เชื่อมต่อภาษาออนโทโลยีของเว็บ (OWL) ประกอบด้วยส่วน API ของออนโทโลยีที่ทำงานกับโมเดลเป็นหลักทั้งอาร์ดีเอฟเอส (RDFS : Resource Description Framework Schema) และ OWL เพื่อเพิ่มส่วนความหมายแก่ฐานอาร์ดีเอฟเดิม และส่วน API ของการอนุมานเป็นส่วนตรวจสอบข้อมูลในฐานทริพเพิล ที่ผู้ใช้ตั้งกฎเองหรือใช้กฎที่มีอยู่แล้วใน API เพื่อตรวจสอบ RDFS

ภาษาและโปรโตคอลที่ใช้ในการคิวรี่อาร์ดีเอฟ

ภาษาที่ใช้ในการคิวรี่ภาษาอาร์ดีเอฟและโปรโตคอลได้แก่สปาร์เคิล (SPARQL) [38] ซึ่งสามารถแปลงค่าเพื่อการคิวรี่ด้วยพีชคณิตสำหรับสปาร์เคิล [53] ดังรูปที่ 2.13 เป็นการแสดงคิวรี่ SELECT แล้วแปลงค่าเป็นโครงสร้างต้นไม้ความสัมพันธ์ ประโยคคิวรี่ด้านซ้ายหลัง WHERE คือลูกของต้นไม้ความสัมพันธ์ การแปลงในรูปแบบนี้มีข้อดีที่สามารถเชื่อมต่อกับฐานข้อมูลแบบที่ใช้ SQL ได้

การใช้งานภาษาสปาร์เคิลที่มีอยู่นั้นมีแบบเดิม [38] และแบบปรับการ aggregate ปรับการคิวรี่ระหว่างโลคอลและฐานความรู้กราฟจากเว็บและปรับปรุงคุณสมบัติเพิ่มเติม [39] โดยทั่วไปการคิวรี่มี 4 รูปแบบคือ SELECT, CONSTRUCT, ASK และ DESCRIBE ซึ่งแต่ละแบบมีหน้าที่ดังนี้

คำสั่ง SELECT มีหน้าที่คืนค่าทั้งหมด ใช้เมื่อต้องการหาค่าใด ๆ รวมทั้งสับเซตและตัวแปรตามขอบเขตที่รูปแบบคิวรี่ไปจับคู่ได้ คำสั่งนี้เทียบเท่ากับ SELECT ของ SQL ที่คิวรี่ฐานข้อมูลแล้วคืนค่าในรูปแบบตารางซึ่งต่างจากสปาร์เคิลที่คิวรี่ค่าที่เก็บอยู่ตามที่ระบุในฟรีฟิกส์ ส่วนหลัง WHERE คือเงื่อนไขที่ใช้ในการค้นหา



รูปที่ 2.13 คิวรี่ภาษาสปาร์เคิลและต้นไม้ความสัมพันธ์

ที่มา: R. Cyganiak, **SPARQL query and its translation into a relational operator tree,** 2005, หน้า 3

คำสั่ง ASK คืนค่าเป็นบูลีน เหมาะกับประโยคคำถามที่ต้องการคำตอบว่าใช่หรือไม่ใช่

คำสั่ง CONSTRUCT มีหน้าที่ดึงค่าสับเซตของคิวรี่ แล้วคืนค่าออกมาเป็นกราฟ เซตของทริพเพิล เหมาะสำหรับการนำมาประยุกต์ใช้เพื่อแปลงให้อยู่ในรูปอาร์ดีเอฟ และการอนุมานอย่างง่ายด้วย แต่ยังคงมีข้อจำกัดเรื่องการใช้รูปประโยคของทริพเพิล

คำสั่ง DESCRIBE มีหน้าที่คืนค่าออกมาเป็นกราฟเช่นกันโดยถามว่าของสิ่งที่ดูคล้ายกันมีความเกี่ยวข้องกันหรือไม่ เหมาะสมกับการนำมาสร้างงานต้นแบบ

การคิวรี่เพื่อหาคำตอบ

รูปแบบการคิวรี่ด้วยวิธีแบบลำดับ การคิวรี่ด้วยสปาร์เคิลแบบลำดับพิจารณาปัจจัยดังนี้

เทอมของอาร์ดีเอฟ มีค่าที่เป็นไปได้สามแบบ คือ 1) *IRI* (Internationalized Resource Identifiers) เป็นโปรโตคอลที่เป็นส่วนเติมเต็มของ URI (Uniform Resource Identifiers) [54] โดย IRI เป็นอักษรหมวดยูนิโค้ด (ISO10646) ซึ่งเวลาใช้งานจะจับคู่กับ URI เมื่อขณะนั้นทรัพยากรต้องการความสามารถที่มากขึ้นจะใช้คุณสมบัติของ IRI แทน 2) *ข้อความ* 3) *โหนดว่าง* (blank node) ทั้งสามแบบถูกกำหนดในไวยากรณ์ของอาร์ดีเอฟ กรณี Subject ก็เป็นได้ทั้งสามแบบข้างต้น ส่วน Object เป็นข้อความและ IRI ส่วน Predicate เป็นได้เฉพาะ IRI

ตัวแปร หมายถึงตัวแปรของคิวรี่ เขียนอยู่ในรูป *?ตัวแปร*

ทัพเพิลอาร์ดีเอฟ เป็นฟังก์ชันจากตัวแปรไปที่เทอมอาร์ดีเอฟ โดยต้องแปลงจากทริพเพิลมาเขียนเป็นทัพเพิล สำหรับความแตกต่างคือ ทริพเพิล เขียนอยู่ในรูปของ subject predicate และ object ซึ่งทริพเพิลใด ๆ สามารถแปลงเป็นทัพเพิลได้ดังด้านล่าง

?subject⟶ http://dbpedia.org/page/Thailand
?predicate⟶ http://www.w3.org/2003/01/geo/wgs84_pos#geometry
?object ⟶ http://www.openlinksw.com/schemas/virtrdf#Geometry

เมื่อได้เอกสารอาร์ดีเอฟ หรืออาร์ดีเอฟเอสหรืออาวล์ แล้วขั้นต่อไปเป็นการแปลงข้อมูลเหล่านี้ให้อยู่ในรูปพจนานุกรมที่ประกอบด้วยส่วนของความสัมพันธ์ทั้งหมดของออนโทโลยี สถิติของแต่ละรายการทั้งคลาสและพร็อพเพอร์ตี้ การเข้ารหัสของคลาสและพร็อพเพอร์ตีเป็นรูปแบบตารางแฮช ทั้งหมดนี้สามารถเชื่อมต่อกับขั้นตอนการคิวรี่ด้วยอินเตอร์เฟสของพจนานุกรม อีกส่วนที่สำคัญคือโครงสร้างของทริพเพิลของออนโทโลยีอาจอยู่ในรูปต้นไม้ [55] หรือกราฟ ในเวลาเดียวกันหลังจาก

รับประโยคคิวรี่รูปแบบสปาร์เคิลเข้าสู่ขั้นประมวลผลคิวรี่ และแบ่งเป็น 5 ขั้นตามลำดับที่สัมพันธ์กัน [56] ดังนี้

1. การตรวจสอบความสอดคล้องกัน (satisfiability checks) คือตรวจความเป็นเหตุผลกันทั้งผลจากการอนุมาน TBox (terminological component) และข้อมูลจริงที่ใส่ ABox (assertion component) เช่นคลาส A มีอินสแตนซ์ *a1* หรือไม่ โดยตรวจสอบจากพจนานุกรม

2. การเข้ารหัสคิวรี่ (query encoding) หลังจากตรวจข้อมูลและความเป็นเหตุผลแล้วต้องเข้ารหัส เพื่อคำนวณตามพรีฟิกส์ (prefix) ที่ใช้ และเลือกตัวดำเนินการต่อไป ผลลัพธ์ที่ได้จากขั้นตอนนี้คือลำดับชั้นของคลาสหรือคอนเซปต์ และลำดับชั้นของพร็อพเพอร์ตี้ ส่วนนี้ต้องใช้พจนานุกรมเช่นกัน

3. การวางแผนคิวรี่ (query plan generation) ขั้นตอนนี้จะได้รูปของต้นไม้จากจอยน์แล้วสร้างลำดับการทำงานไว้ ทรัพยากรที่ส่วนนี้ต้องการคือพจนานุกรม

4. การประมวลคิวรี่ (query execution) ทำหน้าที่คิวรี่ตามลำดับที่จัดมาจาก 3) ส่วนนี้ต้องการทริพเพิลเพื่อนำมาจับคู่กับคิวรี่ที่รับมา

5. การแปลผลคิวรี่ (query translation) คือขั้นตอนการแปลงข้อมูลจากขั้นคิวรี่เป็นรูปแบบผลลัพธ์ที่ต้องการ ทรัพยากรที่ส่วนนี้ต้องการคือพจนานุกรม

รูปแบบการคิวรี่แบบขนานของฐานออนโทโลยี

เมื่อได้ขั้นตอนของการแต่ละคิวรี่เพื่อการประมวลผลแบบขนานแล้วต้องเตรียมและจัดข้อมูลให้ทำงานบนหน่วยประมวลผลแบบขนาน โดยแบ่งงานและแบ่งข้อมูล ให้แยกกันทำงาน การหาคำตอบหรืออนุมานฐานความรู้ออนโทโลยี ด้วยการประมวลผลแบบขนาน [56] จะทำได้เร็วขึ้นและได้ปริมาณมากขึ้น สิ่งสำคัญคือ การแบ่งปัญหาให้เล็กลงและให้มีขนาดสมดุลและให้มีส่งข้อมูลไปยังหน่วยประมวลผลแบบขนานให้น้อยที่สุด ในส่วนประสิทธิภาพนั้น ควรมีการใช้ทริพเพิลผลลัพธ์ที่ได้จากหน่วยประมวลผลเดียวกันให้มากและลดการอนุมานที่ซ้ำซ้อนกันในหน่วยประมวลผล

รูปแบบโครงสร้างข้อมูลที่นิยมใช้ได้แก่กราฟ ข้อดีของรูปแบบนี้ได้แก่การค้นหาความสัมพันธ์ทำได้ง่ายและรวดเร็ว ลดความซ้ำซ้อนของข้อมูล ทำให้รองรับข้อมูลปริมาณมาก ข้อเสียคือการสร้าง กราฟใช้เวลา และอาจจะใช้โครงสร้างภายในที่ซับซ้อน เช่นลิสต์ รวมทั้งการแบ่งกราฟเป็นส่วนๆ ไปยังโหนดต่าง ๆ ในกรณีการประมวลผลแบบกระจาย ในบางงานวิจัย ใช้อัลกอริทึมแบบแฮชฟังก์ชัน เพื่อลดความซ้ำซ้อนของข้อมูล และเป็นโครงสร้างข้อมูลแบบอะเรย์ที่ไม่ซับซ้อน ประสิทธิภาพของแฮชจึงขึ้นอยู่กับฟังก์ชันทางคณิตศาสตร์ ข้อเสียคือไม่สามารถแบ่งส่วนอย่างเด็ดขาด

เพื่อแก้ปัญหา ทำให้ต้องเก็บแฮชทั้งตารางในหน่วยความจำที่เดียว จะมีปัญหาเรื่องการจองหน่วยความจำเมื่อขนาดของแฮชโตขึ้นจนระบบปฏิบัติการไม่สามารถจัดสรรได้ การแก้ไขได้แก่ใช้อัลกอริทึมเฉพาะโดเมน (domain) ตัวอย่างได้แก่ออนโทโลยี LUBM ซึ่งเป็นชุดตรวจสอบ OWL ในโดเมนเรื่องมหาวิทยาลัย ใช้ฟังก์ชันแฮชทำงานแบบสตรีมมิ่งเพื่อไม่ต้องโหลดกราฟทั้งหมดเข้าสู่หน่วยความจำ จากนั้นประมวลผลกราฟเพื่อให้นำเซตข้อมูลอื่นกลับมาใช้ใหม่ได้ การวัดประสิทธิภาพการจำลองตัวของผลลัพธ์คือ *OR* = ∑(จำนวนของเซตทัพเพิลในผลลัพธ์ของแต่ละหน่วยประมวลผล)/ผลรวมของทัพเพิลในกราฟผลลัพธ์ที่ยูเนี่ยนกัน ส่วนการวัดผลการจำลองตัวของอินพุตคือ *IR* = ∑(จำนวนของโหนดแต่ละหน่วยประมวลผล)/ผลรวมของโหนดในกราฟอินพุตที่รับเข้ามา

การแบ่งกฎในออนโทโลยีนั้น จะสร้างกราฟที่มี predicate ให้เป็นด้านของแต่ละแนวคิด (concept) และให้แบ่งจำนวนของคอนเซปต์ในแต่ละส่วนให้เท่ากัน อัลกอริทึมแบบขนานสำหรับแต่ละโหนดคือการทำงานแบบวนรอบ งานแต่ละรอบของทุกหน่วยประมวลผลคือกฎมาตรวจเซตข้อมูลเพื่อให้ได้ทัพเพิลที่ผ่านการอนุมานแล้ว แล้วจึงตรวจว่ามีทัพเพิลถูกส่งผ่านหน่วยประมวลผลอีกหรือไม่ ถ้ามีให้บันทึกข้อมูลไว้ กรณีที่กฎถูกส่งผ่านมาให้สร้างกฎทั้งหมดใหม่อีกครั้งแล้วส่งข้อความบอกหน่วยประมวลผล การเริ่มรอบใหม่ให้นำผลลัพธ์จากรอบที่แล้วรวมกับข้อมูลที่ถูกสร้างใหม่มาเป็นอินพุตแล้วทำวิธีข้างต้นซ้ำอีกแล้วหยุดเมื่อทัพเพิลที่ใช้ส่งต่อไม่ถูกสร้างขึ้นมาอีก [57]

การประเมินผลงานตามความหลักการวิเคราะห์ผลลัพธ์

การประเมินประสิทธิภาพของอัลกอริทึมหลังจากแบ่งกฎและแบ่งข้อมูลแล้ว มีรูปแบบ - ตารางที่ 2.3 การประเมินผลงานตามหลักวิเคราะห์ผและสรุปงานของระบบ

ความต้องการของระบบ	**คำอธิบาย**
การแบ่งชิ้นงานต้องสมดุล	จำนวนการทำงานที่ให้แต่ละโพรเซสเซอร์ต้องใกล้เคียงกัน หรือในกรณีที่เสร็จมีงานเสร็จก่อนงานชิ้นอื่นต้องรอจนกว่างานอื่นแล้วเสร็จ จึงวัดเวลาที่ใช้ทั้งหมด
การสื่อสารกันน้อยที่สุด	ข้อมูลที่มีการส่งต่อกันระหว่างโพรเซสเซอร์ควรมีน้อยเพื่อให้งานเป็นอิสระแก่กันมากที่สุด
ประสิทธิภาพ	แต่ละทริพเพิลที่เป็นผลลัพธ์ต้องได้จากโพรเซสเซอร์เดียวเท่านั้น จึงใช้วิธีลดจำนวนครั้งของการอนุมานให้ซ้ำซ้อนกันน้อยที่สุดในแต่ละโพรเซสเซอร์
ความเร็วและขนาดข้อมูล	การแบ่งงานควรทำงานอย่างรวดเร็วและแบ่งขนาดข้อมูลให้ใหญ่ที่สุดเท่าที่จะเป็นไปได้
ความต้องการของผู้ใช้งาน	**คำอธิบาย**
เวลาในการตอบสนอง	ไม่ควรใช้เวลานานในการหาคำตอบให้ผู้ใช้
การได้คำตอบที่เกี่ยวข้อง	ระบบควรคืนค่าคำตอบของสิ่งที่ผู้ใช้ถามไม่ว่าคำตอบนั้นจะถูกต้องที่สุดหรือไม่ก็ตาม
การแสดงผลทางจอ	ควรแสดงผลด้วยว่ามีคำตอบหรือไม่มีคำตอบ ไม่ควรเป็นจอว่าง

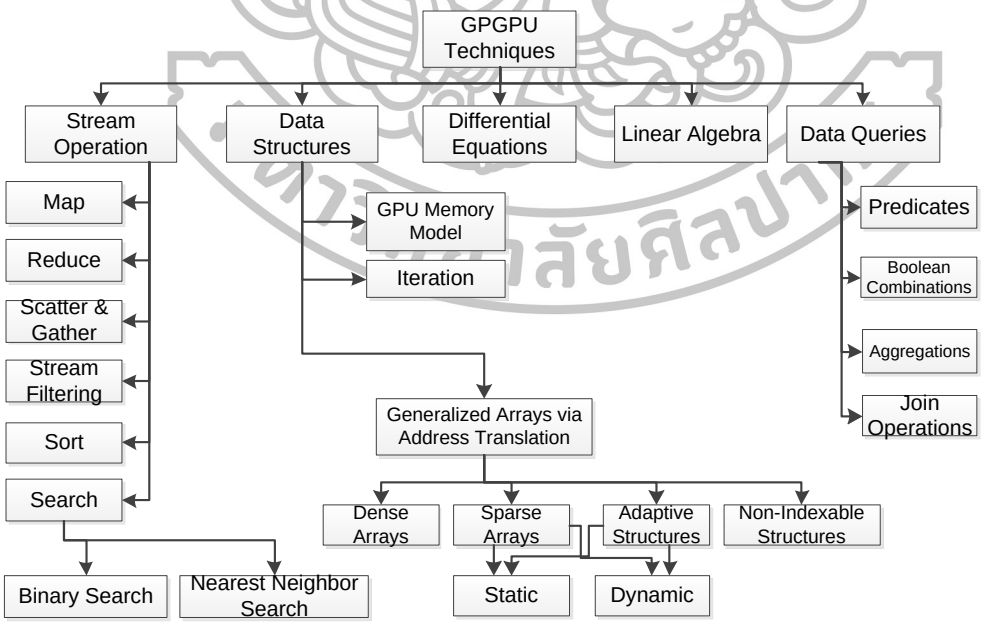
การประเมินโดยวิเคราะห์ความต้องการระบบและผู้ใช้ คำนวณเวลา ภาระงานขึ้นกับขนาดเซตของกฎและข้อมูล วิธีแบ่งงานนี้จึงเหมาะเฉพาะบางงาน เป้าหมายการประเมินผลเป็นดังตารางที่ 2.3 [57]

การประเมินเวลาการทำงาน

การประเมินผลข้อมูลเมื่อนำเข้าสู่หน่วยประมวลผลกราฟิกส์แบ่งเป็น 3 ด้าน ดังนี้ ได้แก่ เวลาการแบ่งเป็นงานย่อยจำนวนมาก เวลาในการประมวลงานย่อยและเวลาการรวมงานก่อนกลับส่งไปหน่วยประมวลผลกลาง พิจารณาสมการเพื่อคำนวณการแบ่งงานคือ $T_{overall}=T_{mm\_dm(I)}+T_{GPU}+T_{dm\_mm(O)}$ มีพารามิเตอร์ของแบนด์วิธ คือ *Band* เป็นการส่งข้อมูลมีหน่วยเป็นไบต์ต่อวินาที โดยเวลาแต่ละส่วนย่อยในการส่งข้อมูลเป็น x ไบต์ เวลาส่งข้อมูลเริ่มต้น T_0 วินาที ถัดมาข้อมูลที่แบ่งเป็นชิ้นจะถูกส่งด้วยความเร็วตามแบนด์วิธ เวลาคือ $T_{mm\_dm}(x) = T_0 + \frac{x}{Band}$ เวลาของงานย่อยคือ $T_{computation}$ งานย่อยของการจอยน์โดยมีพารามิเตอร์เวลาโดยประมาณคือ T^0_{GPU} เวลาต่อหน่วยประมวลผลคือ μ และให้จุดสมดุลของลำดับงานมีเท่ากับขนาดของคิวรี่ คือ $(n_0, \ldots, n_q)$ ดังนั้นเวลาต่อหน่วยคือ $\mu = \frac{T^0_{GPU}}{O(n_0,\ldots,n_q)}$ เมื่อนำมารวมกันจะได้เวลาของงานย่อย $T_{computation} = \mu \cdot o(n_0,\ldots,n_q)$

งานวิจัยที่เกี่ยวข้อง

งานวิจัยที่เกี่ยวข้องในด้านการประมวลผลแบบขนานบนหน่วยประมวลผลกราฟิกส์



รูปที่ 2.14 งานวิจัยด้านต่าง ๆ ที่ใช้หน่วยประมวลผลกราฟิกส์ ในการเพิ่มประสิทธิภาพการประมวลผล

จอห์น ดี โอเวนส์ และคณะ [58] ได้รวบรวมงานวิจัยด้านต่าง ๆ ที่ใช้หน่วยประมวลผลกราฟิกส์ ในการเพิ่มประสิทธิภาพการประมวลผลดังรูปที่ 2.14 งานวิจัยฉบับนี้ [58] แบ่งการใช้หน่วยประมวลผลกราฟิกส์ออกเป็น 5 ด้าน คือการประมวลสตรีม โครงสร้างข้อมูล สมการเชิงอนุพันธ์ย่อย พีชคณิตเชิงเส้นและการคิวรี่ข้อมูล

อซาโนวิคและคณะ [59] นำเสนอมุมมองแบบเบิร์กเลย์ การคำนวณ 13 แบบเรียก ว่าดวาร์ฟส (Dwarfs) ซึ่งได้รับความเชื่อถือว่ามีผลต่อวิทยาการคอมพิวเตอร์ที่ใช้เพื่อการสื่อสารและการคำนวณ เพื่อออกแบบ ประเมินรูปแบบและสถาปัตยกรรมของการโปรแกรมแบบขนานจุดประสงค์คือเพื่อให้เขียนโปรแกรมได้สะดวกสำหรับระบบที่มีการประมวลแบบขนานระดับสูงอย่างมีประสิทธิภาพเหมาะสมกับระบบเป้าหมายที่มีได้แก่ 1,000 แกนต่อชิพ

ตารางที่ 2.4 ประเด็นสำคัญสำหรับการพัฒนาระบบประมวลแบบขนาน

ประเด็น	คำถาม 7 ประเด็น
ด้านแอปพลิเคชัน	แอปพลิเคชันที่สร้างเกี่ยวข้องกับสิ่งใด
	เคอร์เนลสำคัญของแอปพลิเคชันใช้ทำงานอะไร
ด้านเซิร์ฟเวอร์ (ในกรณีที่เป็นการคำนวณในระบบฝังตัวและเซิร์ฟเวอร์นั้นก็มีข้อจำกัดที่แตกต่างกัน)	แต่ละบล็อกของฮาร์ดแวร์ทำงานอย่างไร
	สามารถเชื่อมต่อแต่ละบล็อกได้อย่างไร
การเขียนโปรแกรม	อธิบายแอพลิเคชันและเคอร์เนลให้ชัดเจนได้อย่างไร
	เขียนโปรแกรมเพื่อจัดสรรฮาร์ดแวร์ให้ลงตัวได้อย่างไร
การประเมินผล	วัดความสำเร็จได้อย่างไร

สำหรับคอมไพเลอร์ที่ใช้ควรปรับแต่งได้อัตโนมัติ (autotuner) สำหรับประเด็นการพัฒนาแบ่งได้ 7 ประเด็น ดังตารางที่ 2.4 พิจารณาประเภทของดวาร์ฟสว่าเป็นงานชนิดใด และในเคอร์เนลของแอปพลิเคชันนั้นทำสิ่งใดบ้าง

เบื้องต้นดังตารางที่ 2.5 สำหรับคอลัมน์ที่ 1 เป็นการแบ่งประเภทของดวาร์ฟส 2 ประเภทได้แก่คือวิธีที่มีอยู่ก่อนและส่วนที่เพิ่มมาภายหลัง ซึ่งพิจารณาตามชนิดได้ในคอลัมน์ 2 และตัวอย่างขอบเขตงานตามดวาร์ฟสคือคอลัมน์ 3 และคอลัมน์สุดท้ายเป็นการตอบคำถามด้านแอปพลิเคชันในตารางที่ 2.4 ว่าเคอร์เนลของหน่วยประมวลผลกราฟิกส์ทำงานอะไร เดิมนั้นพิจารณาจากชุดเครื่องมือที่ใช้ทดสอบเคอร์เนลบนหน่วยประมวลผลกราฟิกส์และติดต่อกับหน่วยประมวลผลกลางคือ Rodinia [60] โดยเครื่องมือชุดนี้แบ่งส่วนของแอปพลิเคชันตามด้านของดวาร์ฟส ยกตัวอย่าง Monte Carlo ใช้ในช่วงแรก ต่อมาได้นำแมพรีดิวซ์มาไว้ในกลุ่ม ตัวอย่างงานการทำเหมืองเว็บ เมื่อนำมาประมวลบนหน่วยประมวลผลกราฟิกส์ในเคอร์เนลจะจับคู่ (คะแนนความคล้ายคลึง, หมายเลขเอกสาร) ในส่วนนี้ Rodinia ได้นำโครงงาน Mars [61] ซึ่งเป็นการพัฒนาแมพรีดิวซ์บนหน่วย

ประมวลผลกราฟิกส์มาทดสอบ สำหรับชนิดที่ 8 -13 ผู้วิจัยได้รวบรวมงานที่เกี่ยวข้องกับดวาร์ฟสด้านนั้นมาเป็นตัวอย่าง

ตารางที่ 2.5 ตัวอย่างงานที่สัมพันธ์กับชนิดของดวาร์ฟสและอัลกอริทึมที่ใช้ในเคอร์เนลของแอปพลิเคชัน

	ชนิดของดวาร์ฟส [59]	ตัวอย่างขอบเขตงาน [60]	อัลกอริทึม/แอปพลิเคชัน/เคอร์เนล
ดั้งเดิม	1) Dense Linear Algebra	Data Mining, Strip Mining [62]	K-means[63], Stream Cluster
	2) Sparse Linear Algebra	Block Compress Sparse Row (BCSR)	Conjugate Gradient Vectorcomputers withgather/scatter
	3) Spectral Methods	3D FFT	FourierTransform
	4) N-Body Methods	Particle Methods	MD-GRAPE (IBM)
	5) Structured Grids	Physics Simulation Image Processing Medical Imaging	HotSpot SRAD Leukocyte Tracking
	6) Unstrudctured Grids	Pattern Recognition	Back Propagation
	7) Monte Carlo/ MapReduce	Web Mining	Similarity Scores (Mars project)
เพิ่มเติม	8) Combinational Logic	Near-Minimal Logic Minimization	Expansion kernel algorithm [64]
	9) Graph Traversal	Graph Algorithms	Breadth-First Search [65]
	10)Dynamic Programming	Bioinformatics	Needleman-Wunsch
	11)Back-track and Branch & Bound	Flow-Shop	Branch-and-Bound Algorithm [66]
	12)Graphical Models	สร้างโมเดลกราฟจากโหนดและด้านแบบสุ่ม	Bayesian networks Hidden Markov Models
	13)Finite State Machine	สร้างระบบที่มีการกำหนดสถานะ การถ่ายทอดเหตุการณ์ไปสถานะถัดไป	Fragment shaders for agent anima- tion using finite state machines [67]

งานวิจัยที่เกี่ยวข้องในด้านการประมวลผลข้อมูลอักขระแบบขนานบนหน่วยประมวลผลกราฟิกส์

ปัจจุบันการเขียนโปรแกรมเชื่อมกับคูด้าเพื่อบังคับการทำงานบนหน่วยประมวลผลกราฟิกส์เพื่อนำข้อมูลแบบอักขระเข้าไปประมวลผลแบบขนานได้ โดยข้อมูลต้องมีความสัมพันธ์กันเชิงโครงสร้าง จากการศึกษาผลงานที่เกี่ยวข้องสามารถแบ่งการทำงานกับข้อมูลอักขระได้ 3 ระดับดังนี้

ข้อมูลประเภทฐานข้อมูล การประมวลผลทำได้โดยการจัดข้อมูลให้เป็นไฟล์ข้อความที่คั่นด้วยเครื่องหมายเช่น “|” จากนั้นจึงแปลงเข้าฐานข้อมูลและใช้ภาษา SQL เรียกค้น ตัวอย่างระบบ alenka [68] ที่มีชนิดข้อมูล 2 ชนิด ได้แก่สตริงและวันที่ นำเข้าข้อมูลในหน่วยประมวล

กราฟิกส์ และทำการจอยน์บนหน่วยประมวลผลกราฟิกส์ [18] และคิวรี่ผ่านหน่วยประมวลผลกราฟิกส์ แล้วส่งมาแสดงผลบนหน่วยประมวลผลกลาง

ข้อมูลประเภท XML สามารถนำมาประมวลผลบนหน่วยประมวลผลกราฟฟิกส์ ในรูปแบบข้อมูลชนิดต้นไม้ ซึ่งใช้ในหลายงานวิจัย หน่วยประมวลผลกลางรับไฟล์ XML และคิวรี่ XPath จากนั้นแปลงไฟล์ให้อยู่ในรูปแถวลำดับแล้วนำเข้าสู่หน่วยประมวลผลกราฟฟิกส์ แล้วจึงใช้อัลกอริทึมเช่น Twigjoin [69] มาช่วยในการประมวลผล หรือประยุกต์การคำนวณเป็นแบบขนานเช่น Parallel Twigstack [70] จากนั้นย้ายข้อมูลกลับสู่หน่วยประมวลผลกลางแล้วแปลงเป็นคำตอบ

ข้อมูลประเภทอาร์ดีเอฟ/อาวล์ (RDF/OWL) นำมาประมวลบนหน่วยประมวลผลกราฟฟิกส์ ด้วยเครื่องแบบกระจาย แบบคลัสเตอร์ ข้อมูลอาวล์มีกฎของความสัมพันธ์จำนวนมาก สามารถใช้หลักการทำงานของแมพรีดิวซ์แบ่งไฟล์แล้วจับคู่ความหมายจากนั้นจึงย่อไฟล์ตามคำที่จับคู่ไว้ แล้วจึงนำไปประมวลผลแบบขนานบนเครื่องประมวลผลแบบกระจาย [57] และยังสามารถนำแมพรีดิวซ์มาประยุกต์ใช้ในการประมวลบนหน่วยประมวลผลกราฟิกส์ได้ [61]

งานวิจัยที่เกี่ยวข้องกับการประมวลผลคิวรี่บนหน่วยประมวลกราฟิกส์

การประมวลออนโทโลยี ที่มีความสัมพันธ์เชิงตรรกะต่างจากการประมวลผลไฟล์อักขระทั่วไปคือต้องมีการประมวลผลกฎที่ของออนโทโลยีมีอยู่กับข้อมูลที่มีอยู่ เพื่อให้ได้ผลลัพธ์ในเชิงความหมาย ดังเช่นงานวิจัยต่อไปนี้

การประมวลผลการจอยน์ข้อมูลอาร์ดีเอฟแบบขนานบนหน่วยประมวลกราฟิกส์ การศึกษาเกี่ยวกับการประมวลผลเซตทริพเพิลบนหน่วยประมวลผลกราฟิกส์พบว่ามีการทำอาร์ดีเอฟเป็นตารางแฮชแล้วจึงนำไปทำงานแบบขนาน มีขั้นตอนคือนำไฟล์อาร์ดีเอฟมาใช้เป็นข้อมูลเข้า ส่วนของอัลกอริทึมได้แก่เรียงและจอยน์บนหน่วยประมวลผลกราฟิกส์ทั้ง Sort/Merge แบบ Radix และแบบ Nested Loop ประเด็นที่น่าสนใจคือการนำข้อมูลเข้าหน่วยประมวลผลกราฟิกส์โดยจำลองตารางแฮชที่มีดัชนีค้นหา ซึ่งใช้พื้นที่น้อยกว่าการนำข้อมูลเข้าแบบไร้โครงสร้างหรือแบบข้อความ ซึ่งต้องมีการกำหนดการแบ่งขนาดข้อมูลเป็นชิ้น (chunk) และค่าจากแฮชฟังก์ชันที่ได้จะต้องถูกปัดให้เป็นจำนวนเต็ม แล้วนำตารางที่มีพารามิเตอร์ของสปาร์เคิลมาจอยน์กันแบบสองคอลัมน์หรือตามรายการเชื่อมตัวแปรดังคู่ลำดับคีย์/ค่า ซึ่งผลของการจอยน์จะมีค่าที่เกิดจากการวนซ้ำจำนวนมากและทำให้ต้องคำนึงถึงเวลาในการรับส่งข้อมูลบนหน่วยประมวลผลกราฟิกส์ [71]

การคิวรี่ของสปาร์เคิลจะคล้ายคลึงกับการคิวรี่ด้วย SQL พบว่ามีการคิวรี่ฐานข้อมูล [18] แต่แตกต่างกันในบางเงื่อนไขและความสามารถของฟังก์ชันการ aggregate อย่างไรก็ตามเมื่อศึกษาเฉพาะการคิวรี่ข้อมูลแบบขนานพบว่ามีการแบ่งพัฒนา คือ การคิวรี่บนหน่วยประมวลผลกราฟิกส์ ใน

หน่วยความจำของหน่วยประมวลผลกลางเพื่อเพิ่มประสิทธิภาพของการจอยน์ และการพัฒนาวิธีจอยน์แบบขนาน จุดมุ่งหมายคือ 1) เรื่องประสิทธิภาพที่เน้นสร้างบล็อกเพื่อปรับการประมวลผลเฉพาะแบบขนาน 2) โดยทั่วไปจะใช้ฐานข้อมูลเก็บข้อมูลขนาดใหญ่ จะนำข้อมูลนี้มาแบ่งให้ทำงานบนดีไวส์ ในวิทยานิพนธ์ฉบับนี้สนใจประเด็นการอ่านข้อมูล มีการสร้างข้อมูลพรีมิทีฟเช่นเดียวกับ CUDPP (CUDA data parallel processing Library) [72] ประกอบด้วย การแมพ การอ่าน/เขียน การสแกนพรีฟิกซ์ โดยการแปลงเป็นตัวดำเนินการแบบไบนารี การแยก (split) และเรียงลำดับ ตัวดำเนินการดังกล่าวช่วยในการเข้าถึงข้อมูลระหว่างฐานข้อมูลและโครงสร้างข้อมูลในหน่วยความจำ

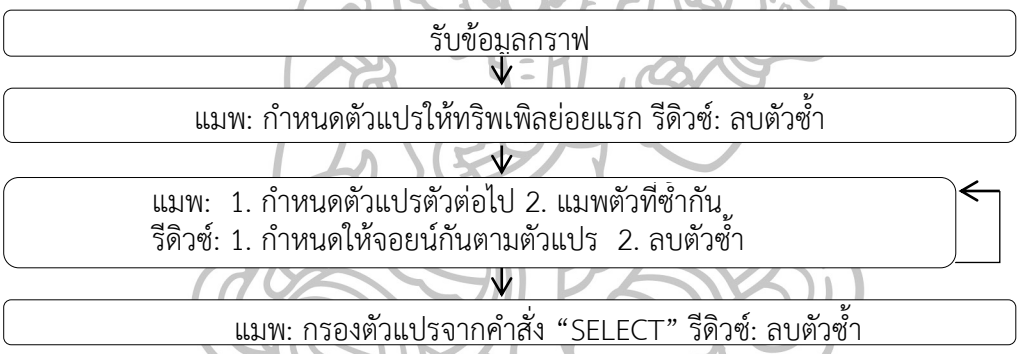
การเตรียมข้อมูลเพื่อการค้นหาเชิงความหมายบนหน่วยประมวลผลกราฟิกส์

การค้นหาแบบเชิงความหมายบนหน่วยประมวลผลกราฟิกส์ [73] ได้กล่าวถึงการค้นบนเว็บแบบเดิม ที่ต้องการขยายความเป็นการค้นแบบเชิงความหมาย ด้วยการเปรียบเทียบคลาสแบบต้นไม้ขนาดใหญ่อย่างรวดเร็ว โดยมากเป็นข้อมูลบนหน้าเว็บโดยตรง และใช้พลังงานต่ำ การเปรียบเทียบแบ่งเป็นการคำนวณแฮชอย่างรวดเร็ว วิธีบลูมฟิลเตอร์และการลดลงแบบขนาน การทดสอบมีช่วงรันไทม์ วัดพลังงานไฟฟ้าที่ใช้ไป ระหว่าง 2 แพลตฟอร์ม ได้แก่ เซิร์ฟเวอร์ อินเทล เอ็กซ์ 86 โพรเซสเซอร์ และ เครื่องที่มี GPU ผลการทดลองพบว่าการเร่งความเร็วเป็น 4 เท่า พร้อมกับการลดพลังงานได้ 78% เมื่อเทียบกับการทำงานแบบลำดับ การวิจัยนี้จึงสรุปว่าการใช้วิธีนี้ในศูนย์ข้อมูลที่สนับสนุนการค้นหาเชิงความหมายแบบกระจาย จะสามารถลดการใช้พลังงาน ค่าใช้จ่ายในการดำเนินงานต่อพื้นที่ที่เก็บเซิร์ฟเวอร์ สำหรับระบบประมวลผลแบบขนานขนาดใหญ่หลายงานวิจัยได้เก็บข้อมูลบนอาปาเช่ คาสซานดรา [74] ในคิวมิวลัสอาร์ดีเอฟ [47] ก็ใช้วิธีนี้ ซึ่งการค้นหาบนหน่วยประมวลผลกราฟิกส์ช่วยลดการใช้พลังงาน

งานวิจัยที่เกี่ยวข้องกับการประมวลออนโทโลยีบนคลัสเตอร์และบนกลุ่มเมฆ

การประมวลผลแบบขนานกับข้อมูลปริมาณมากใช้แมพรีดิวซ์ [25] ใช้กลไกสำคัญคือตัวแมพและตัวรีดิวซ์ เริ่มจากแบ่งข้อมูลเป็นหลายบล็อก แล้วจับคู่ไฟล์ที่มีปัญหาเดียวกันเข้าจัดสรรบล็อกเหล่านี้ให้กับตัวแมพ และแต่ละตัวแมพให้ผลลัพธ์ชั่วคราว ส่งต่อให้ตัวรีดิวซ์ทำการรวมผลที่ได้ [75] เป็นการแปลงข้อมูลออนโทโลยีชนิดอาร์ดีเอฟเอส/อาวล์ด้วยแมพรีดิวซ์เพื่อประมวลผลแบบขนาน แก้ปัญหาคิวรี่ข้อมูลทริพเพิลขนาดใหญ่ โดยขั้นตอนคล้ายกับ [76] คือเริ่มจากการลดขนาดข้อมูลก่อนนำไปประมวลผล แต่งาน [76] ใช้การแปลงข้อมูลให้อยู่ในรูปเมตริกซ์ของบิต ส่วน [75] มุ่งประเด็นของการทำงานเพิ่มขึ้นของปริมาณข้อมูลเว็บเชิงความหมายในเวลาอันรวดเร็ว และทำการกระจายการประมวลผลเพื่อเพิ่มความเร็ว

ตามกฎอาร์ดีเอฟเอสและกฎของ ter Horst [77] ของอาวล์ด้วยการใช้รูปแบบแมพรีดิวซ์ เทคนิคนี้ช่วยกระจายการหาเหตุผลผ่านเซตของอัลกอริทึมที่นำมาใช้ร่วมกันอย่างมีประสิทธิภาพ โดยขั้นตอนประกอบด้วยการสร้างส่วนอนุมานเพื่อเว็บขนาดใหญ่ขึ้น (Web-scale Inference Engine) บนคลัสเตอร์ 64 โหนด และประเมินผลโดยใช้เซตข้อมูลขนาดใหญ่ตามจริงประกอบด้วย Bio2RDF (ข้อมูลด้านชีววิทยา), LLD (Linked Life Data), LDSR (Linked Data Semantic Repository) และ LUBM (Lehigh University Benchmark) ปริมาณมากถึงขนาดหนึ่งแสนล้านทริพเพิล ผลการทดลองแสดงว่าในกรณีที่ขนาดข้อมูลเพิ่มขึ้นแบบเชิงเส้น ระบบนี้จัดการได้ทั้งกรณีข้อมูลที่มีปริมาณมากและกรณีของความเร็วในการหาคำตอบมีการแนะนำแนวคิดเพื่อนำมาใช้กับอาวล์รุ่นที่ 2 [78] ได้คือขั้นตอนการนับคำเพื่อแยกงาน ค่าแมพมาจากทุก ๆ ทัพเพิลที่มีรูปแบบ *<คำ, คำ>* ให้เปลี่ยนรูปแบบเป็น *<คำ,1>* และกำหนดให้คีย์ทัพเพิลนี้คือ*คำ*โดยไม่สนใจค่า 1 ส่วนรีดิวซ์คือการนำคีย์ที่เหมือนกันมารวมกัน

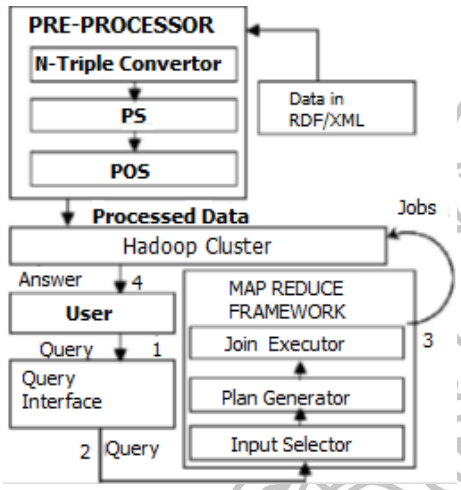


รูปที่ 2.15 *ภาพรวมของอัลกอริทึมทำซ้ำประยุกต์สู่การประมวลผลภาษาสปาร์เคิลของข้อมูลทริพเพิล*

แนวคิดที่สอง [79] คือส่วนตรวจสอบเหตุผลของอาวล์ประกอบด้วย 2 ขั้นตอนคือการเตรียมและการตรวจที่แบ่งงานเป็น 8 บล็อกตามประเภทของคุณสมบัติในอาวล์ ด้วยการเข้ารหัสพจนานุกรมโดยแบ่งเซตของทริพเพิลเป็นสองส่วนคืออินสแตนซ์และโมเดล แล้วส่งข้อมูลเข้าทำงานต่อในส่วนของการตรวจเหตุผลที่มีความสัมพันธ์กันได้แก่ 1 Inheritance properties 2 Transitive properties 3 Same as statement 4 Equivalence 5 Derived Equivalence 6 Same as inheritance 7 hasValue statement และ 8 someValues และ allValues statement โดยทำกับทุกอินสแตนซ์ ทำซ้ำเงื่อนไขที่ต้องการ จากนั้นลบส่วนที่ซ้ำซ้อนและส่งทริพเพิลคำตอบกลับออกมา

นอกจากการนี้ยังมีการประมวลผลข้อมูลอาร์ดีเอฟบนกลุ่มเมฆด้วยวิธีฮิวริสติกบนฮาดูป [80] ซึ่งเน้นแก้ปัญหาการคิวรี่ข้อมูลแบบทริพเพิลขนาดใหญ่และคิวรี่อาร์ดีเอฟผ่านระบบกลุ่มเมฆ

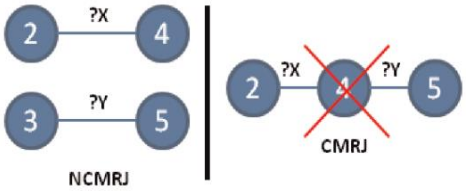
ขั้นตอนเริ่มจากการเก็บข้อมูลแบบอาร์ดีเอฟในไฟล์แบบกระจายของฮาดูป โดยกำหนดให้มีงานมากกว่า 1 งานที่ใช้ในตอบคำถามของคิวรี่ ดังนั้นจึงต้องวางแผนการคิวรี่โดยภาษาสปาร์เคิล (SPARQL) ขั้นตอนการหาคำตอบดังรูปที่ 2.16 [80] พบว่าผู้ใช้คิวรี่ข้อมูลผ่านหน้าจอและแมพรีดิวซ์ส่งงานให้กับคลัสเตอร์ของฮาดูปที่ผ่านการแยก predicate-subject และแยก predicate-subject-object แล้วโดยใช้หลักความน่าจะเป็นจัดหมู่ (combination) ค้นเซตข้อมูลทริพเพิลในรูปกราฟเพื่อหาคำตอบแล้วให้คลัสเตอร์ของฮาดูปส่งคำตอบให้แก่ผู้ใช้ต่อไป



รูปที่ 2.16 สถาปัตยกรรมของระบบการประมวลผลข้อมูลอาร์ดีเอฟบนกลุ่มเมฆด้วยวิธีฮิวริสติก

ส่วนรายละเอียดรูปที่ 2.17 [80] ที่น่าสนใจคือ การจอยน์ด้วยแมพรีดิวซ์มีการกำหนด 1) TP: รูปแบบทริพเพิลในประโยค SELECT ของสปาร์เคิล 2) TPJ: คือการจอยน์ระหว่าง 2 ทริพเพิล 3) MRJ: การจอยน์ในแมพรีดิวซ์ที่ประกอบด้วยการจอยน์ของตั้งแต่ 2 ทริพเพิลขึ้นไป 4) ส่วนการจอยน์ในแมพรีดิวซ์ (NCMRJ) ก็รับค่าไปใช้งานต่อในส่วนแมพรีดิวซ์เดิมและ 5) แบบมีโหนดซ้ำ (CMRJ) หรือโหนดที่ใช้งานร่วมกันดังรูปที่ 2.17 (ขวา) เป็นส่วนที่มีปัญหาเนื่องจากกำหนดให้ 1 ทริพเพิล

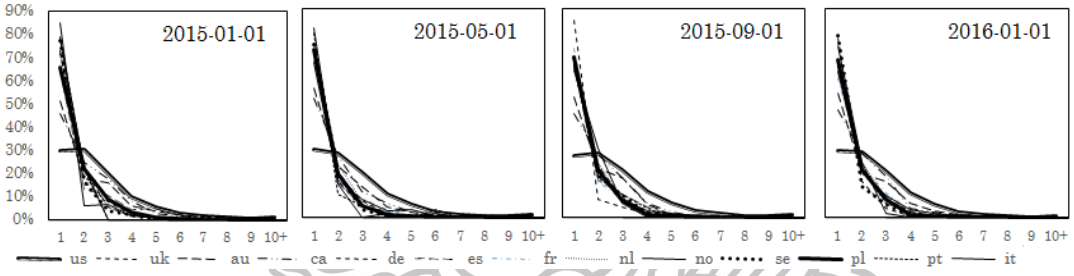
จุดที่นำมาประเมินเวลาคืองานตอนแมพและรีดิวซ์ สำหรับโมเดลฮิวริสติกคือการใช้กฎการจัดหมู่คำนวณความเป็นไปได้ของพร็อพเพอร์ตี้ที่ใช้งาน เริ่มจากการอ่านข้อมูลจำนวน 1 พันล้านทริพเพิลแล้วเรียงข้อมูลจึงต้องพิจารณาว่าแผนการใดทำงานต่อชิ้นงานน้อยใช้ขั้นตอนที่สุด จุดนี้จึงต้องวางแผนข้อมูลแล้วจึงนำไปจอยน์แบบนับค่าคีย์ด้วยแมพรีดิวซ์ต่อไป



รูปที่ 2.17 การจอยน์ในแมพรีดิวซ์ที่ไม่มีโหนดซ้ำ (NCMRJ) และแบบมีโหนดซ้ำ (CMRJ)

การสำรวจพฤติกรรมของผู้ใช้งานอินเตอร์เน็ตในการค้นหาโดยใช้คำสำคัญ

ตามพฤติกรรมของผู้ใช้งานอินเตอร์เน็ต การค้นหาคำสำคัญของเว็บเชิงความหมายในยุคแรกมีการค้นหาคำสำคัญแบบเดี่ยวและแบบหลายคำผ่านเซตข้อมูลโครงสร้างแบบอาร์ดีเอฟ ตัวอย่างเครื่องมือค้นหาเว็บไซต์เชิงความหมายที่มีโครงสร้างอาร์ดีเอฟได้แก่ Swoogle [81] Watson [82] และ Sindice [83] เป็นต้น



รูปที่ 2.18 สถิติการค้นหาคำสำคัญตามพฤติกรรมผู้ใช้ตามช่วงเวลา 1 ม.ค. 2015 – 1 ม.ค. 2016

มีงานวิจัยที่กล่าวว่าการค้นหาคำสำคัญอยู่ในเนื้อหาไม่สนใจว่าเป็นโครงสร้างใด ไม่ว่าจะเป็นเว็บเพจ เอกสารอาร์ดีเอฟหรือเอกสารกึ่งโครงสร้าง [84] ในงานวิจัยนี้นักวิจัยทำการค้นหาคำสำคัญหลายคำบนเนื้อหาเดียวกันเพื่อเพิ่มความเร็วในการค้นหา เริ่มจากการกำหนดจำนวนคำในการค้น จากรูปที่ 2.17 แสดงสถิติของการศึกษา [85] จำนวนคำที่ใช้ในการคิวรี่ ของแต่ละประเทศ วัดเปอร์เซ็นต์ของการค้นหาคำเดี่ยวและการค้นหาหลายคำในภาษาต่าง ๆ เช่น ผู้ใช้ภาษาอังกฤษจากประเทศอังกฤษและสหรัฐอเมริกามีแนวโน้มใช้คำในการค้นหา 2 – 3 คำ ในขณะที่ประเทศอื่นมักใช้ 1 คำในการค้นหา จากแนวโน้มทั้งหมดพบว่าการค้นหามากกว่า 6 คำมีแนวโน้มลดลงจนกระทั่งเหลือ 0% ในการค้นหาตั้งแต่ 10 คำขึ้นไป จากการศึกษาในบทที่ 2 นี้ทำให้ผู้วิจัยใช้จำนวนคำสำคัญ 1 – 8 คำ โดยเพิ่มเป็นจำนวนเลขยกกำลังของ 2 คำสำคัญที่ใช้มี 8 คำคือ thai spa huahin tourism health wellness massage budget แบ่งทดสอบ 4 ครั้งย่อยคือ 1, 2, 4, และ 8 คำ ในบทที่ 4 - 5

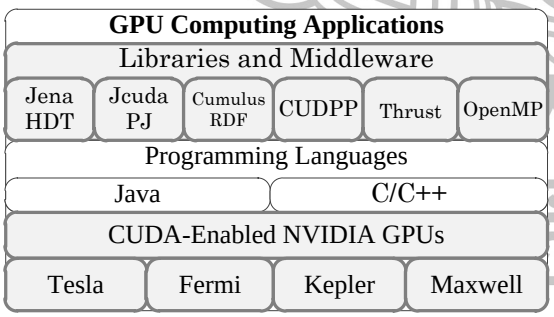
บทที่ 3

ขั้นตอนการดำเนินงาน

ขั้นตอนการดำเนินงานของการประมวลผลแบบขนานของฐานความรู้ที่ได้จากลิงค์เดต้าบนสถาปัตยกรรมแบบหลายแกน แบ่งออกเป็น กรอบการทำงานของระบบ ประกอบด้วย กระบวนการทำงานของระบบ การพัฒนาโปรแกรมและขั้นตอนการค้นหาข้อมูลแบบอาร์ดีเอฟ

กรอบการทำงาน

กรอบการทำงานซอฟต์แวร์ เป็นการนำแนวคิดที่มีเฟรมเวิร์กของคูด้าเป็นพื้นฐานมาปรับปรุงใหม่เพื่อให้เหมาะสมกับงานที่ต้องการใช้ โดยสร้างเป็นโปรแกรมเปิด (open source) ที่นำกลับมาใช้ใหม่ได้ผ่าน API (Application programming interface) ที่มีการใช้งานที่ต่างจากไลบรารี่เฉพาะด้านอื่น ๆ นั่นคือมีส่วนที่ปรับและเขียนโปรแกรมเพิ่มได้จากภายในกรอบการทำงาน มีฟังก์ชันที่ทำงานเพื่อประโยชน์เฉพาะด้าน มีส่วนให้นักพัฒนาสามารถเพิ่มฟังก์ชันการทำงานได้เองและโปรแกรมแกนหลักที่ตรงกับวัตถุประสงค์การวิจัย กรอบการทำงานในงานวิจัยนี้เป็นดังรูปที่ 3.1 ประกอบ ด้วยสามส่วนหลักคือไลบรารี่และมิดเดิลแวร์ของโปรแกรมที่พัฒนาด้วยภาษาจาวาและซี และกรอบการทำงานโปรแกรมคูด้าที่ควบคุมการทำงานแบบขนานบนหน่วยประมวลผลกราฟิกส์



รูปที่ 3.1 ส่วนประกอบของแอปพลิเคชัน เพื่อควบคุมการทำงานบนหน่วยประมวลผลกราฟิกส์

อธิบายรูปที่ 3.1 ที่ประกอบด้วยชั้นดังต่อไปนี้

ชั้นการเขียนโปรแกรมประกอบด้วยการเขียนโปรแกรมภาษาซีและภาษาจาวา ซึ่งจะกล่าวถึงในขั้นตอนการพัฒนาโปรแกรม

ชั้นไลบรารี่และมิดเดิลแวร์ ใช้เชื่อมต่อกับการคิวรี่และเซตข้อมูลของเว็บเชิงความหมาย ใช้เชื่อมต่อกับคูด้า ใช้ทำงานแบบมัลติเทรด แบ่งออกเป็นการใช้งานร่วมกับภาษาจาวาและภาษาซี

สำหรับไลบรารี่และมิดเดิลแวร์ที่ใช้ร่วมกับภาษาจาวามีดังต่อไปนี้

มิดเดิลแวร์อาปาเช เจนา (Apache Jena) [52] ที่ใช้จัดการเซตข้อมูลของเว็บเชิงความหมาย ลิงค์เดต้า สปาร์เคิล

เอชดีที (HDT: Header-Dictionary-Triples) [86] เป็นไลบรารี่ที่ใช้ย่อเซตข้อมูลอาร์ดีเอฟด้วยโครงสร้างต้นไม้ แบบบิต แบบไบนารี ตามแต่ผู้ใช้เลือกใช้งาน มีทั้งภาษาซีและจาวา ภายในมีมิดเดิลแวร์ Raptor [87] จากไลบรารี่ Redland [88] เชื่อมกับเว็บเชิงความหมาย ส่วนรูปแบบย่อใช้ zlib [89]

เจคูด้า [90] เป็นไลบรารี่ที่ใช้เชื่อมระหว่างภาษาจาวาและคูด้า ช่วยให้สามารถใช้อินเตอร์-เฟสภาษาจาวา ที่เป็นที่นิยมของมิดเดิลแวร์เว็บเชิงความหมายได้

PJ (Parallel Java) [91] เป็นไลบรารี่ที่ทำหน้าที่ไดเรคทีฟแบบมัลติเทรดสำหรับภาษาจาวาเทียบได้กับ OpenMP ของภาษาซี

CumulusRDF [92] จากงานวิจัย [47] และการแนะนำจาก W3C [93] ทำหน้าที่ควบคุมการคิวรี่ฐานข้อมูลแบบคีย์-ค่า เชื่อมกับฐานความรู้ทริพเพิล อาร์ดีเอฟและลิงค์เดต้า

สำหรับไลบรารี่และมิดเดิลแวร์ที่ใช้ร่วมกับภาษาซีมีดังต่อไปนี้

CUDPP (CUDA Data Parallel Primitives Library) [72] เป็นไลบรารี่ที่เชื่อมต่อระหว่างการนำข้อมูลเข้าประมวลในหน่วยประมวลผลกราฟิกส์ด้วยพริมิทีฟทั้งหลาย มี Thrust เป็นพื้นฐาน

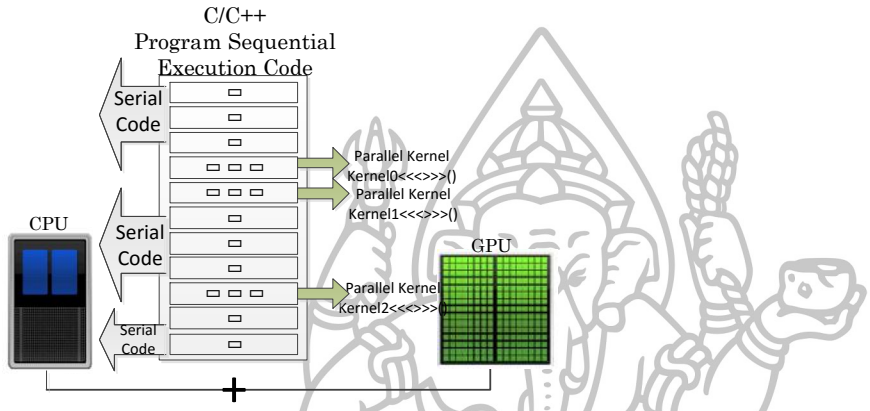
Thrust [94] เป็นไลบรารี่ที่เชื่อมต่อระหว่างการนำข้อมูลใด ๆ เข้าประมวลในหน่วยประ-มวลผลกราฟิกส์ด้วยพริมิทีฟ ที่รวมอยู่ในเครื่องมือของคูด้า

CUSP [95] เป็นไลบรารี่ที่เชื่อมกับคูด้าใช้จัดการพีชคณิตเชิงเส้นและกราฟ ทำงานบนพื้นฐานและเพิ่มเติมจาก Thrust

OpenMP [96] เป็นไดเรคทีฟที่ทำให้โปรแกรมทำงานมัลติเทรดกับภาษาซีและฟอร์แทรน ประมวลผลบนหน่วยประมวลผลกลางหรือโฮสต์เป็นหลัก

ชั้นกรอบการทำงานคูด้า ทำหน้าที่ควบคุมการทำงานฮาร์ดแวร์ของหน่วยประมวลผลกราฟิกส์ โดยเปิดให้นักพัฒนาโปรแกรมสามารถสร้างโปรแกรมจากฝั่งโฮสต์ คำสั่งของคูด้าจะถูกส่งไปมาระหว่างหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกส์ ภาพรวมการโปรแกรมเป็นดังรูปที่ 3.2 การเขียนโปรแกรมผสมระหว่างส่วนที่ต้องการประมวลผลแบบลำดับและแบบขนานคือเมื่อเขียน

โค้ดของแอปพลิเคชันจะมีส่วนที่ทำงานแบบลำดับ ที่ทำงานบนหน่วยประมวลผลกลาง ใช้หน่วยความจำของระบบหรือโฮสต์ และโค้ดส่วนที่ต้องการให้ทำงานแบบขนานบนหน่วยประมวลผลกราฟิกส์หรือบนหน่วยประมวลผลหลายแกนที่ต้องคำนึงถึงการจัดการงานแบบขนานบนหน่วยความจำของหน่วยประมวลผลกราฟิกส์ที่มีขนาดเล็กกว่าหน่วยความจำระบบ สำหรับรายละเอียดภายในโปรแกรมนั้นจะกล่าวถึงในขั้นตอนการพัฒนาโปรแกรม



รูปที่ 3.2 การเขียนโปรแกรมผสมระหว่างส่วนที่ต้องการประมวลผลแบบลำดับและแบบขนาน

ตารางที่ 3.1 การพัฒนาทางด้านฮาร์ดแวร์ของหน่วยประมวลผลกราฟิกส์จากบริษัทเอ็นวิเดีย

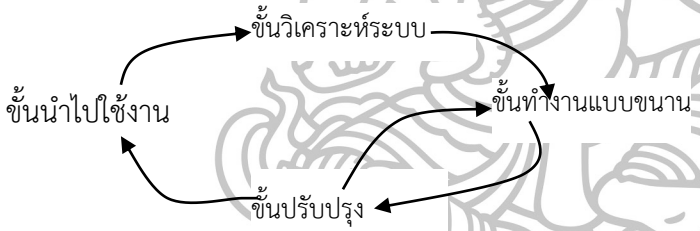
คุณสมบัติ	รุ่นเทสลา	รุ่นเฟอร์มิ	รุ่นเคปเลอร์			แมกซ์เวลล์
ตัวอย่างการ์ด	Geforce310M	GTX560 Ti	GTX 660	TeslaK20	TeslaK40	GTX 980
ชิพ	GT218	GF114	GK106	GK110	GK110B	GM204
ทรานซิสเตอร์	1.40 พันล้าน	1.95 พันล้าน	2.54 พันล้าน	7.1 พันล้าน	7.1 พันล้าน	5.20 พันล้าน
จำนวนแกน	16	384	960	2496	2880	2048
Stream Multiprocessors	2	8	5	13	15	16
หน่วยความจำสูงสุด	512 MB	1024 MB	2048 MB	5120 MB	12288 MB	4096 MB
นาฬิกากราฟิกส์	625MHz	822MHz	980Hz.	5.2 GHz	6 GHz	1024 MHz
บัสหน่วยความจำ	64 บิต	256 บิต	192 บิต	320 บิต	384 บิต	256 บิต
นาฬิกาหน่วยความจำ (MHz)	800 MHz	4008 MHz	6Gbps	2.6 GHz	3.0 GHz	7.0 Gbps
แคช L2	0	512 KB	384 KB	1.5 MB	1.5 MB	2 MB
Compute Capability	1.2	2.1	3.0	3.5	3.5	5.2

กรอบการทำงานคู่ด้าสามารถส่งคำสั่งควบคุมระหว่างหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกส์ โดยเขียนโปรแกรมขยายภาษาซีใช้สำหรับการโปรแกรมฝั่งของหน่วยประมวลกราฟิกส์ มีการพัฒนาการใช้งานบนหน่วยประมวลกราฟิกส์ที่มีคุณสมบัติตามตารางที่ 3.1 แสดงรายละเอียดของรุ่นของหน่วยประมวลผลกราฟิกส์ที่มีคุณสมบัติการคำนวณและจำนวนทรานซิสเตอร์ที่มากขึ้น หน่วยความจำที่เพิ่มขึ้น จากรุ่นเทสลาจนถึงรุ่นเคปเลอร์มีการเพิ่มจำนวนแกน ประสิทธิภาพที่ได้ต่อวัตต์จากรุ่นเฟอร์มิมาเคปเลอร์เพิ่มขึ้นแบบทวีคูณ ในรุ่นแมกซ์เวลล์จะเห็นได้ชัดว่าจำนวนแกนและทรานซิสเตอร์มีมากเป็นทวีคูณ เพิ่มประสิทธิภาพการคำนวณด้วยการเพิ่มหน่วยควบคุมทางตรรกศาสตร์ (control logic)

ขั้นตอนการพัฒนาโปรแกรม

การวิเคราะห์ ออกแบบและปรับปรุงระบบ

การวิเคราะห์และออกแบบระบบแบบ APOD [97] สรุปได้สี่ขั้นตอนดังรูปที่ 3.3



รูปที่ 3.3 Systematic optimization แบบ APOD

ขั้นวิเคราะห์ (Assess หรือ Analyze [98]**)** คือมองภาพรวมของแอปพลิเคชันโดยวิเคราะห์ว่าส่วนใดของงานควรทำงานแบบขนาน กำหนดขนาดของปัญหา (weak vs. strong scaling) เมื่อให้ปัญหาขนาด $\mathcal{P}$ ประมวลแบบลำดับ ในเวลา $\mathcal{T}$ เช่นประมวลผลเซตข้อมูลอาร์ดีเอฟแบบลำดับใช้เวลา 1 ชั่วโมง ปัญหามีขนาดใหญ่เกินไปทำให้ประสิทธิภาพการทำงานช้า เมื่อแบ่งปัญหาให้เล็กลงให้ทำงานพร้อมกันจะช่วยเพิ่มประสิทธิภาพ เนื่องจากฮาร์ดแวร์แต่ละชนิดมีลักษณะไม่เหมือนกัน จึงต้องทดลองแบ่งปัญหาหลายขนาดแล้วทดสอบหาขนาดปัญหาที่เหมาะกับประมวลผลต่อแกนที่สุด ทำการประมวลผลเพื่อทดสอบหาเวลาที่เร็วที่สุด ดังนั้นการวัดประสิทธิภาพจะเทียบกับขนาดปัญหาที่เท่ากัน การใช้หน่วยประมวลกราฟิกส์ ควรทำงานได้เร็วกว่าการใช้เพียง CPU ทำงานแบบลำดับ

ขั้นทำงานแบบขนาน (Parallelize) พิจารณาว่าในการประมวลผลแบบขนานต้องแก้ปัญหาโดยวิธีใด ใช้ไลบรารี ใช้ไดเรคทีฟ หรืออัลกอริทึมที่หมาะสม ซึ่งเป็นขั้นตอนที่สำคัญที่สุด

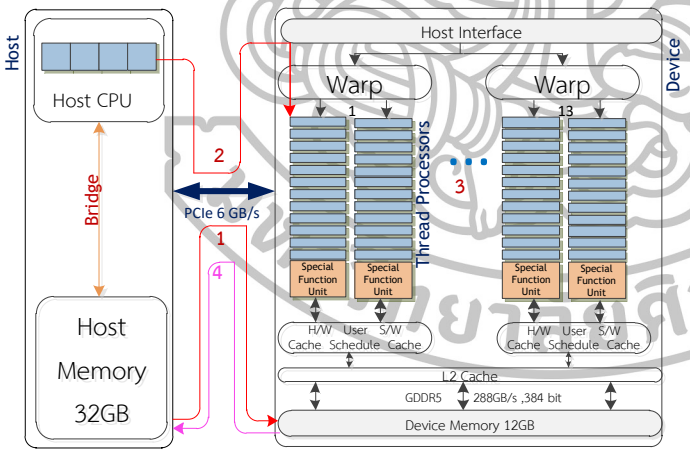
ขั้นปรับโปรแกรม (Optimize) ขั้นตรวจสอบ หรือวัดประสิทธิภาพของโปรแกรมว่าเป็นไปตามภาพรวมที่ตั้งไว้หรือไม่ หากยังใช้ไม่ได้ให้กลับไปแก้ไขขั้นการทำงานแบบขนานอีกครั้ง ดังนั้นในขั้นการทำงานแบบขนานและขั้นปรับโปรแกรมทำงานไป-กลับระหว่างกันได้ คือแก้ไขการเขียนโปรแกรมแบบขนานและปรับปรุงจนกว่าจะได้โปรแกรมที่เหมาะสม

ระดับของการปรับโปรแกรม พิจารณาจากอัลกอริทึม วิธีการที่เรียบง่ายน่าจะมีประสิทธิภาพที่สุด และอาจจะทำลองเปลี่ยนวิธีอื่น ๆ ทั้งนี้การปรับโปรแกรมขึ้นอยู่กับรุ่น และไลบารี่ของสถาปัตยกรรมหน่วยประมวลผลกราฟิกส์ด้วย

ขั้นนำไปใช้งาน (Deploy) ในกรณีที่โปรแกรมประสบความสำเร็จตามต้องการ

รูปแบบการส่งคำสั่งของคูด้า

การส่งคำสั่งจากโปรแกรมภาษาซีที่พัฒนาจากหน่วยประมวลผลกลางและไปทำงานบนหน่วยประมวลผลกราฟิกส์ เพื่อประมวลผลแบบขนานมีการไหลของคำสั่งดังรูปที่ 3.4



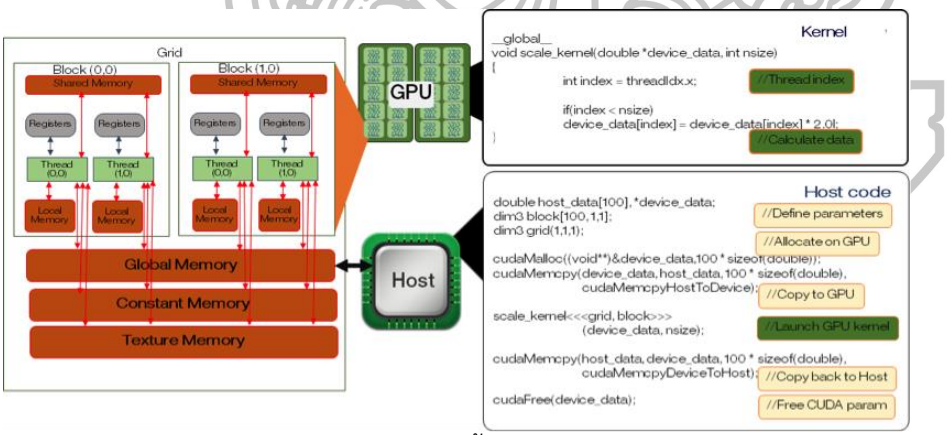
รูปที่ 3.4 การไหลของคำสั่งจากโปรแกรมที่เขียนบนคูด้า

ที่มา ประยุกต์จาก https://www.pgroup.com/images/insider/pgi-fermi-block-diagram.png

จากรูปที่ 3.4 หมายเลข 1 คัดลอกข้อมูลจากหน่วยความจำของหน่วยประมวลผลกลางไปหน่วยความจำของหน่วยประมวลกราฟิกส์ หมายเลข 2. ส่งคำสั่งจากโฮสต์ ไปหน่วยประมวลผล

กราฟิกส์ หมายเลข 3 ประมวลผลแบบขนานบนแกน และ 4. คัดลอกผลลัพธ์จากหน่วยความจำของหน่วยประมวลกราฟิกส์ไปหน่วยความจำของโฮสต์ถือว่าสิ้นสุดการทำงานหนึ่งรอบ นอกจากนี้ในหน่วยประมวลกราฟิกส์ประกอบด้วยหน่วยความจำหลายชั้น และหลายพันแกนจำนวนตามสถาปัตยกรรมที่มี ในแกนมีหน่วยคำนวณพีชคณิต (ALU) คำนวณทศนิยม (FPU) และหน่วยคำนวณฟังก์ชันพิเศษ (SFU) เช่น ล็อก ตรีโกณมิติ เลขยกกำลัง แคช การตั้งค่าทำงานจากผู้ใช้ ความเร็วการเข้าถึงหน่วยความจำจากชั้นที่ช้าสุดไปเร็วสุดได้แก่หน่วยความจำโกล- บอล โลคอล เท็กซ์เจอร์ << คอนสแตนซ์ << แชร์ รีจิสเตอร์

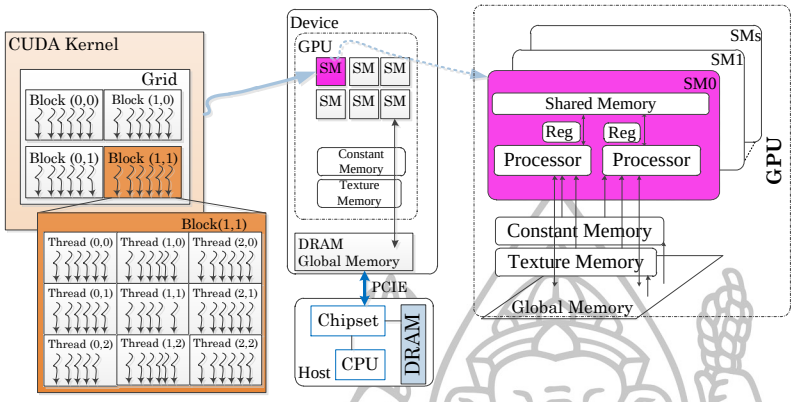
ความสัมพันธ์ระหว่างโปรแกรมและชั้นของหน่วยความจำตามมุมมองของคูด้าเป็นดังรูปที่ 3.5 ได้แก่โค้ดภายในเคอร์เนลซึ่งแทรกตัวอยู่ในโค้ดทั้งหมดที่อยู่ฝั่งโฮสต์ รูปแบบการทำงานภายในเคอร์เนล แบ่งข้อมูลให้ทำงานแบบขนานด้วยคำสั่งจากโปรแกรมเดียวกัน คำสั่งที่อยู่ในแต่ละเคอร์เนลล์จะจัดกลุ่มเทรดในคูด้าให้เป็นบล็อก (Thread-blocks) แล้วจัดเรียงหลายบล็อกลงสู่กริด สำหรับการใช้งานหน่วยความจำ แต่ละเทรดจะใช้หน่วยความจำระดับโลคอลได้ ภายในเทรดที่อยู่ในบล็อกเดียวกันจะใช้หน่วยความจำแชร์ สำหรับเทรดในกริดเดียวกันจะใช้หน่วยความจำโกลบอลร่วมกันได้ สามารถดูรายละเอียดร่วมกับตารางที่ 2.2 เรื่องระยะเวลาของแต่ละหน่วยความจำและคุณสมบัติอ่านเขียน เพื่อประยุกต์ให้เหมาะกับการใช้งาน



รูปที่ 3.5 ความสัมพันธ์ระหว่างโปรแกรมและชั้นของหน่วยความจำตามมุมมองของคูด้า

เมื่อเทียบมุมมองของคูด้ากับฮาร์ดแวร์ของหน่วยประมวลผลกราฟิกส์ได้ดังรูปที่ 3.6 ในรูปด้านซ้าย เคอร์เนลล์ของคูด้าทำงานภายใน SM (Stream Multiprocessor) ของหน่วยประมวลผลกราฟิกส์ (รูปกลาง) ที่ทุก SM เข้าถึงหน่วยความจำโกลบอล คอนสแตนซ์และเท็กเจอร์ ซึ่งเป็นส่วนที่

โฮสต์และดีไวส์เข้าถึงร่วมกันได้ และภายใน SM (รูปขวา) จะมีหน่วยความจำโลคอล รีจิสเตอร์และแชร์ที่เทรดในบล็อกสามารถใช้ได้รวมอยู่

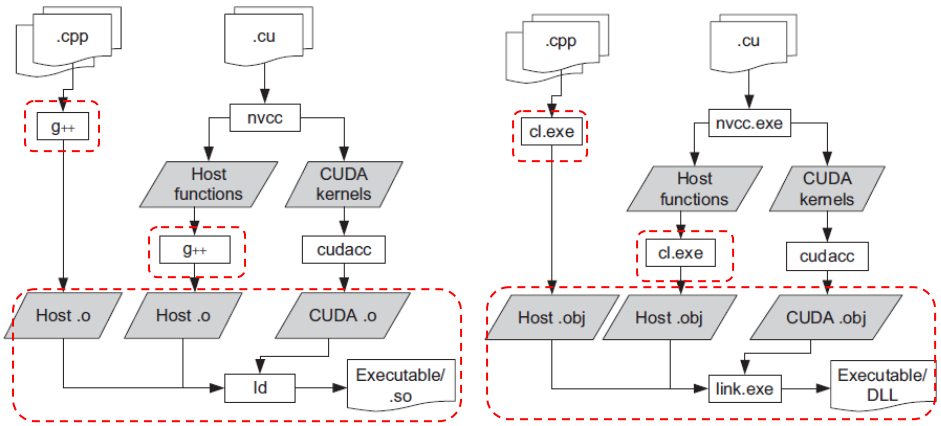


รูปที่ 3.6 มุมมองของกริด บล็อก เทรดของคูด้าและหน่วยประมวลผลกราฟิกส์

เครื่องมือที่ใช้ในการเขียนโปรแกรมประกอบด้วยโปรแกรมแก้ไขไฟล์อักขระ ที่บันทึกข้อมูลเป็นไฟล์นามสกุล c, cpp ตามภาษาซี ไฟล์เฮดเดอร์นามสกุล h ไฟล์ที่ใช้คอมไพล์โดยคูด้าคือ .cu และไฟล์เฮดเดอร์นามสกุล cuh เมื่อเป็นจาวาใช้คลาสจาวา เชื่อมต่อกับไฟล์ภาษาซีที่ผ่านการคอมไพล์แล้วเป็น ptx หรือไฟล์คูด้าแบบไบนารี นามสกุล cubin การคอมไพล์โปรแกรมด้วย nvcc [99] ระหว่างสองระบบปฏิบัติการคือลินุกซ์และวินโดว์ มีความแตกต่างกันดังรูปที่ 3.7 นั่นคือ ระบบปฏิบัติการลินุกซ์ (รูปที่ 3.7 ซ้าย) ใช้ g++ ในการคอมไพล์ทำให้ได้ไฟล์ .o สุดท้ายเมื่อนำมารวมกับไฟล์ของคูด้า .cu ที่ผ่านการคอมไพล์ด้วย nvcc จะได้ไฟล์ปฏิบัติการ .so ขณะที่ระบบปฏิบัติการวินโดว์ (รูปที่ 3.7 ขวา) ใช้ cl.exe ในการคอมไพล์ทำให้ได้ไฟล์ออบเจ็กต์ (.obj) สุดท้ายเมื่อนำมารวมกับไฟล์ของคูด้า .cu ที่ผ่านการคอมไพล์ด้วย nvcc จะได้ไฟล์ปฏิบัติการ DLL นอกจากนี้ยังมีบางคำสั่งในโปรแกรมที่ทำหน้าที่เดียวกันแต่ใช้คำสั่งต่างกันของทั้งสองระบบปฏิบัติการ อาจทำให้เกิดปัญหาต่อผู้ที่จะนำโปรแกรมไปใช้งานต่างระบบปฏิบัติการจากโปรแกรมที่ผู้พัฒนาได้สร้างขึ้นได้ ดังนั้นหากต้องการเขียนโปรแกรมแล้วผู้ใช้สามารถนำไปใช้งานได้ทั้งสองระบบ ต้องเขียนเงื่อนไขของการคอมไพล์สำหรับแต่ละระบบปฏิบัติการ โดยมีทั้งแบบฝังไว้ในโค้ดหลักหรือไฟล์เฮดเดอร์ [100] เพื่อให้คอมไพล์แล้วนำไปใช้งานได้ทั้งสองระบบปฏิบัติการ

การตรวจสอบโปรแกรมอาจใช้โปรแกรมดีบัก (debug) เช่น cuda-gdb [101] สำหรับระบบปฏิบัติการลินุกซ์ และ Parallel Nsight สำหรับระบบปฏิบัติการวินโดว์ [102] และสำหรับระบบปฏิบัติการลินุกซ์เมื่อใช้ร่วมกับเครื่องมือเขียนโปรแกรมอีคลิปต์ (Eclipse) [103] ที่ใช้เป็น

เครื่องมือพัฒนาภาษาซีและภาษาจาวา การประเมินการทำงานแบบขนานนั้นสามารถทำได้โดยใช้ Visual Profiler [104] ร่วมกับคูด้า ที่เป็นผลงานจากบริษัทเอ็นวิเดีย



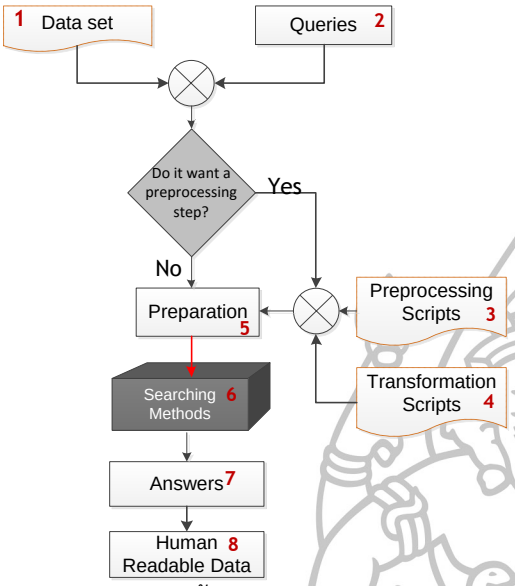
รูปที่ 3.7 การคอมไพล์โปรแกรมของระบบปฏิบัติการลินุกซ์ (ซ้าย) และระบบปฏิบัติการวินโดว์ (ขวา)

การวัดค่าที่ได้จากการทดสอบด้วยการคำนวณประกอบด้วยค่าเวลาที่ใช้ทั้งหมด ประกอบด้วยเวลาที่ใช้ในการคัดลอกข้อมูล เวลาที่ใช้ในการค้นหาข้อมูลบนหน่วยประมวลผลกราฟิกส์ เทียบกับเวลาที่ใช้ในการประมวลผลแบบลำดับและเวลาที่ใช้ในการค้นหาแบบ มัลติเทรด ค่าแบนด์วิธของการส่งไฟลข้อมูลไปยังหน่วยประมวลผลกราฟิกส์ ค่าธรูพุตของจำนวนการค้นหาต่อหน่วยเวลา

ขั้นตอนการค้นหาข้อมูลแบบลิงค์เดต้า

ภาพรวมขั้นตอนการค้นหาข้อมูลแบบลิงค์เดต้าดังรูปที่ 3.8 เริ่มจากนำเซตข้อมูลแบบอาร์ดีเอฟจากหลากหลายโดเมน จากหลายแหล่ง รวมถึงเซตข้อมูลที่สร้างขึ้นเองที่เป็นไปตามมาตรฐานของลิงค์เดต้า (1) มาใช้ในการทดสอบคิวรี่แบบสปาร์เคิล (2) จากนั้นพิจารณาว่าต้องมีขั้นตอนก่อนประมวลผลหรือไม่เช่น การแปลงไฟล์ประเภทอาร์ดีเอฟเป็นแบบ NT การแปลงโครงสร้างข้อมูลจากอักขระเป็นบิต การตัดต่อข้อมูล หากมีให้ผ่านขั้นตอนก่อนประมวลผล (3) และขั้นตอนการแปลงข้อมูล (4) จากนั้นจึงไปสู่ขั้นตอนการเตรียมข้อมูล เช่น การเรียงข้อมูล การสร้างดัชนี (5) จากนั้นจึงเข้าสู่ขั้นตอนการค้นหา ในที่นี้ให้เป็นกล่องดำ เนื่องจากการค้นหามีหลายแบบ เช่น การเปรียบเทียบสตริง การค้นหาจากโครงสร้างแบบบิต เป็นต้นซึ่งจะกล่าวถึงในแต่ละการทดลอง และสถานที่ทำการประมวลผลมีตั้งแต่โฮสต์ที่เป็นการประ มวลแบบลำดับ การประมวลขนานแบบมัลติ - เทรด เพื่อทำการเปรียบเทียบและการประมวลแบบขนานบนหน่วยประมวลผลกราฟิกส์ จากนั้นจึง

รวบรวมคำตอบออกมา (7) และสรุปผลออกมาแก่ผู้ใช้ เช่น จำนวนที่ค้นพบ ข้อความที่ค้นพบ ตำแหน่งที่ค้นพบ เป็นต้น



รูปที่ 3.8 ภาพรวมขั้นตอนการค้นหาข้อมูลแบบลิงค์เดต้า

รายละเอียดการทำงานที่เกิดขึ้นทั้งสองฝั่ง โดยฝั่งโฮสต์มีขั้นตอนการจัดการข้อมูลดังนี้

การเก็บข้อมูลและการเตรียมข้อมูล ประกอบด้วยเซตข้อมูลเชิงความหมายระดับอาร์ดีเอฟหลายแบบ มีการเก็บข้อมูลแบบไฟล์อักขระ ฐานข้อมูลดัชนีและค่า ข้อมูลแบบไบนารี่ตามโครงสร้างต้นไม้ที่มีดัชนีแบบบิต ซึ่งข้อมูลอาจจะดาวน์โหลดได้โดยตรงหรือต้องนำมาแปลงด้วยเครื่องมือที่เกี่ยวข้อง เป็นการแปลงข้อมูลให้เหมาะกับการนำข้อมูลเข้าสู่หน่วยประมวลผลกราฟิกส์เพื่อประกอบการวิจัย ได้แก่ การนำข้อมูลรับเข้ามาแปลงเป็นทริพเพิล แปลงเป็นฐานข้อมูลคีย์-ค่า แปลงเป็นข้อมูลไบนารี่ของอาร์ดีเอฟ และแปลงเป็นข้อมูลรูปแบบต้นไม้เพื่อนำไปใช้งาน หน้าที่หลักของกระบวนการนี้เก็บข้อมูลเพื่อนำเข้าประมวลบนสถาปัตยกรรมแบบหลายแกนเพื่อการทดสอบ

การพัฒนาโครงสร้างข้อมูล แบบแถวลำดับเพื่อประมวลบนหน่วยประมวลผลกราฟิกส์

การพัฒนาอัลกอริทึม เพื่อการเตรียมข้อมูล เช่นการเรียงบิตตามดัชนี เช่น subject-predicate-object หรือ object-subject-predicate

การประมวลแบบลำดับและแบบขนานแบบมัลติเทรด บนโฮสต์ ได้แก่ การประมวลโดยใช้ไลบรารี่แบบขนานด้วยภาษาซีและ OpenMP ภาษาจาวา (Parallel Java, เจคูด้า) การทำงานโดยพร้อมเพรียงกันของภาษาจาวา (Concurrent Java)

วิธีการค้นหาข้อมูลรองรับการเปรียบเทียบสตริง รองรับการคิวรี่ในคำสั่ง SELECT

การประยุกต์ใช้งาน ในขอบเขตออนโทโลยีการท่องเที่ยว ข้อมูลเปิดในลิงค์เดต้า

การแสดงผลแก่ผู้ใช้ การแสดงผลลัพธ์แก่ผู้ใช้

รายละเอียดการทำงานที่เกิดขึ้นที่ฝั่งดีไวส์ มีขั้นตอนการจัดการข้อมูลดังนี้

การทำงานพื้นฐาน ประมวลผลข้อมูลแบบขนานของการทำงานพื้นฐานที่ประกอบด้วย การอ่าน การเรียง การจับคู่และการลดรูป

การประมวลข้อมูล การจัดการเข้าถึงข้อมูลประกอบด้วยการเข้าถึงข้อมูล โดยแบ่งการเข้าถึงแคชและหน่วยความจำ

การคิวรี่ข้อมูล การประมวลผลด้วยภาษาคิวรี่ ด้วยการทำงานของคำสั่ง SELECT

คุณลักษณะของเครื่องที่ใช้ทดสอบ

คุณสมบัติเครื่องที่ใช้ทดสอบต่างกันไปในแต่ละการทดลอง รายละเอียดต่อไปนี้เป็นคุณสมบัติของเครื่องล่าสุด ฮาร์ดแวร์: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz, 6 แกน, RAM 32 กิกะไบต์, Tesla K40c จำนวน 2 หน่วยประมวลผลกราฟิกส์ 2,880 แกนคูด้า และหน่วยความจำโกลบอล 12 กิกะไบต์ ซอฟต์แวร์: CUDA compilation tools, release 7.0, V7.0.27, NVIDIA-SMI 352.79 Driver Version: 352.79, ระบบปฏิบัติการ: CentOS Linux release 7.2.1511 (Core)

แหล่งข้อมูล

สร้างออนโทโลยีเกี่ยวกับโดเมนการท่องเที่ยวเชิงสุขภาพ แหล่งข้อมูลอาร์ดีเอฟตามมาตรฐาน W3C ได้แก่ Datahub, Google Lab, BTC 2012, DBpedia, Geonames, Linkedata, และฐานความรู้ทางการแพทย์และชีววิทยา MeSH, Karyotype ดังรายละเอียดในบทที่ 4

แผนการศึกษาวิจัย

แผนการวิจัยในระยะเวลา 5 ปี ประกอบด้วยการวิจัย เขียนผลการวิจัยและวิเคราะห์ผลการทดลองเผยแพร่ในงานประชุมวิชาการนานาชาติและวารสารนานาชาติ ดังตารางที่ 3.2 และผลงานท้ายเล่มวิทยานิพนธ์

ตารางที่ 3. 2 แผนการศึกษาวิจัย

ปีที่	รายละเอียดของแผนงาน	ตั้งแต่	ถึง	ระยะเวลา (เดือน)	คิดเป็น % ของงานวิจัย
1	ศึกษาปัญหาเฉพาะด้านของการทำงานของออนโทโลยี ที่นำมาใช้ประมวลผลแบบขนานบนหน่วยประมวลผลแบบหลายแกน รวม ถึงศึกษาเทคโนโลยีแบบขนานที่ใช้ในปัจจุบันเพื่อแก้ปัญหาตามสภาพแวดล้อมที่เอื้ออำนวย	มิ.ย. 2011	พ.ค. 2012	12	15
2	ศึกษาทฤษฎีที่เกี่ยวข้องกับออนโทโลยี ทำความเข้าใจข้อมูลประ กอบด้วย รายละเอียดและองค์ประกอบของฐานความรู้สำหรับเว็บเชิงความหมาย ระดับการสื่อความหมาย การเลือกกลุ่มข้อมูลมาใช้งาน ขั้นตอนการแปลงข้อมูล ส่วนของโครง สร้างข้อมูลและอัลกอริทึมของการค้นหา	มิ.ย. 2012	พ.ค. 2013	12	15
3	ออกแบบ และพัฒนาอัลกอริทึม วิธีการแก้ปัญหาของเทคโนโลยีการประมวลแบบขนาน โดยใช้หน่วยประมวลผลกราฟิกส์	มิ.ย. 2013	พ.ค. 2014	12	20
4	เปรียบเทียบวิธีการกับงานวิจัยอื่น หาจุดเด่นจุดด้อย พัฒนาเครื่อง มือเป็นชุดพัฒนาเพื่อนำไปเผยแพร่ และทดสอบกับกรณีศึกษา รวมทั้งเขียนรายงาน	มิ.ย. 2014	พ.ค. 2015	12	20
5	ประยุกต์ใช้งานที่พัฒนากับเซตข้อมูลหลายโดเมน เขียนบทความ รายงานเพื่อเผยแพร่ในงานประชุมวิชาการและวารสาร	มิ.ย. 2015	พ.ค. 2016	12	30
	รวม			60	100

บทที่ 4

การประมวลผลแบบขนานโดยผ่านขั้นตอนก่อนการประมวลผล

ข้อมูลเพื่อนำมาใช้ในงานวิจัยอยู่ในกลุ่มของลิงค์เดต้าเป็นไปตามมาตรฐานของ W3C ในการเผยแพร่ข้อมูลของเว็บเชิงความหมาย งานวิจัยนี้นำแหล่งข้อมูลที่น่าเชื่อถือแปลงข้อมูลเพื่อนำเข้าประมวลผล [105] และการสร้างออนโทโลยี [106] ตามข้อกำหนดของ W3C

แหล่งข้อมูล

จากตารางที่ 4.1 แสดงชื่อของเซตข้อมูล แหล่งที่มา ขนาดไฟล์ จำนวนเรคคอร์ดและลักษณะไฟล์ ที่ใช้ในงานวิจัย (*คือขนาดที่ย่อ (gz) จากหน้าเว็บ เอชดีที [107])

ตารางที่ 4.1 รายละเอียดของข้อมูล ขนาดและลักษณะไฟล์

ชื่อเซตข้อมูล/แหล่ง	ขนาด	เรคคอร์ด	ลักษณะไฟล์
Freebase-weekly/ [108]	400GB	2,863x10^6 ทริพเพิล	อาร์ดีเอฟแบบ N-Triples [109]
Wikidata/ [110] [111]	55GB	339 x10^6 ทริพเพิล	อาร์ดีเอฟแบบ N-Triples
DBpedia/ [41]	45GB	13x10^6 ควอด	อาร์ดีเอฟแบบ N-Quad [112]
DBpedia/BTC2012 [113]	47GB	198x10^6 ควอด	อาร์ดีเอฟแบบ N-Quad
Freebase/BTC2012	17GB	101x10^6 ควอด	อาร์ดีเอฟแบบ N-Quad
Geonames [114] [115]	10GB	N/A	อาร์ดีเอฟแบบ TTL [116]
Freebase/เอชดีที	2.6GB	770x10^6 ทริพเพิล	ไบนารีเอชดีที [49]
DBpedia/เอชดีที	6GB	431x10^6 ทริพเพิล	ไบนารีเอชดีที
Yago/เอชดีที	903MB	159x10^6 ทริพเพิล	ไบนารีเอชดีที
Geonames [117]/เอชดีที	344MB	123x10^6 ทริพเพิล	ไบนารีเอชดีที
WordNet31/เอชดีที	62MB	5.5x10^6 ทริพเพิล	ไบนารีเอชดีที
DBLP/เอชดีที	286MB	55x10^6 ทริพเพิล	ไบนารีเอชดีที
SWDF /เอชดีที	5.64MB	242,256 ทริพเพิล	ไบนารีเอชดีที
MeSH [118]	709MB	N/A	อาร์ดีเอฟแบบ TTL
KaryotypeOntology [119] [120]	1.43MB	N/A	OWL [78]
HuaHinHealthTour	5.6MB	23,654 ทริพเพิล	อาร์ดีเอฟแบบ N-Triples

รายละเอียดจากตารางที่ 4.1 ประกอบด้วยข้อมูล Freebase-weekly [108] ที่มาจากการเก็บข้อมูลทุกสองสัปดาห์ด้วยการครอว์ลจาก เมตาเว็บ [121] และบริษัทกูเกิล ผู้วิจัยเก็บข้อมูลเมื่อ 14 ก.ย. 2014 ต่อมาทางแลบกูเกิลยุติการให้ข้อมูลในเดือนมิถุนายน 2015 และปิดเซิร์ฟเวอร์สิงหาคม 2016 ผู้วิจัยจึงหาข้อมูลที่มีลักษณะคล้ายกันจาก Wikidata ที่เริ่มให้บริการในปี 2014 มาแทน มีการเก็บเอ็นติตี้ในโดเมนทั่วไปของอินเตอร์เน็ตเช่นเดียวกัน [110] [111] ซึ่งข้อมูลส่วนใหญ่ยังคล้ายคลึงกับวิกิพีเดีย สำหรับ DBpedia เป็นข้อมูลศูนย์กลางของลิงค์เดต้าที่มีที่มาจากการแปลงข้อมูลวิกิพีเดียเป็นรูปแบบอาร์ดีเอฟ ฐานความรู้ Wordnet3.1 [122] คือข้อมูลคำศัพท์จากพจนานุกรมมหาวิทยาลัยพรินซ์ตัน ฐานความรู้ Yago [123] เป็นข้อมูลเชื่อมระหว่าง Wordnet และ DBpedia สำหรับ DBLP [124] เป็นอาร์ดีเอฟของฐานข้อมูลการตีพิมพ์งานวิจัยนานาชาติ ส่วน SWDF [125] คือฐานความรู้จากข้อมูลงานประชุมวิชาการนานาชาติของเว็บเชิงความหมาย ฐานความรู้ Geonames เป็นข้อมูลที่เก็บพิกัดทางภูมิศาสตร์โดยมี 10,951,423 คุณลักษณะ หรือประมาณ 162 ล้านทริพเพิล ณ วันที่ 26 ก.พ. 2016 ฐานความรู้ทางการแพทย์คือ Medical Subject Headings (MeSH) และทางชีววิทยาเกี่ยวกับเซลล์คือ Karyotype Ontology ส่วนฐานความรู้ที่สร้างขึ้นเองคือ HuaHin-HealthTour [126] เกี่ยวกับโดเมนการท่องเที่ยวเชิงสุขภาพด้านสปา

ประเภทของข้อมูลนั้นมีพื้นฐานจากอาร์ดีเอฟที่มีลักษณะสำคัญคือความสัมพันธ์ระหว่าง Subject $\xrightarrow{\text{Predicate}}$ Object เมื่ออยู่ในรูปแบบ N-Triple จะเรียงลำดับเทอมของ URI หรือ IRI ในรูป Subject Predicate Object . ในแต่ละทริพเพิล นั่นคือสามเทอมเรียงกันมีช่องว่างหนึ่งช่องระหว่างเทอมและปิดท้ายบรรทัดด้วยจุด (.) ในกรณีที่เป็น N-Quad จะเพิ่มชื่อกราฟมากับทริพเพิลเดิมรวมเป็นสี่คอลัมน์ ในกรณีที่เป็นข้อมูลอาร์ดีเอฟแบบ Turtle (TTL) ที่มีพรีฟิกส์ของเนมสเปซของเบส ของอาร์ดีเอฟ ของอาร์ดีเอฟเอส ของอาล์ว และเนมสเปซของความสัมพันธ์อื่นที่เกี่ยวข้อง จากนั้นเป็นข้อมูลของแต่ละ Subject ระหว่าง Predicate Object; จะมีช่องว่างและปิดท้ายด้วยเครื่องหมายอัฒภาค (;) จนกว่าจะหมดของ Subject นั้นแล้วปิดท้ายด้วยจุด (.) จึงเริ่ม Subject ใหม่ไปเรื่อย ๆ และสามารถแทรกรหัส 303 [127] ให้เซิร์ฟเวอร์ตอบรีไดเรคอัตโนมัติลงในไฟล์ได้ การเพิ่มระดับเชิงความหมายและกฎทางตรรกศาสตร์เป็นไฟล์อาล์ว (.OWL) [78] เป็นอาร์ดีเอฟ/เอ็กซ์เอ็มแอล ดังนั้นรูปแบบทริพเพิลของอาร์ดีเอฟจึงเหมาะกับการตรวจสอบความเป็นเหตุผล การเชื่อมโยงความสัมพันธ์ระหว่างทริพเพิล ส่วนเอ็กซ์เอ็มแอลเหมาะกับการสื่อสารระหว่างเครื่องคอมพิวเตอร์ และการใช้รูปแบบกราฟเหมาะสมกับการจัดภาพรวมแสดงความสัมพันธ์เพื่อให้ผู้ใช้ดูแล้วเข้าใจความสัมพันธ์ได้ง่าย

การแปลงข้อมูลระหว่างอาร์ดีเอฟ

ในกรณีที่ไฟล์ที่ได้มาแต่ยังไม่ใช่รูปแบบที่ต้องการ จำเป็นต้องแปลงข้อมูล เช่นข้อมูลที่ได้เป็นรูปแบบ N-Quad แต่ในงานวิจัยไม่ต้องการใช้คอลัมน์ชื่อกราฟ ต้องการใช้เพียงแบบ N-Triple แบบ N3 หรือแบบ RDF/XML วิธีหนึ่งคือการใช้เครื่องมือที่ใช้ป้อนคำสั่ง (command-line tool) ดังเช่น rdf-convert-0.4 [128] ภายในโปรแกรมนี้ใช้งานไลบรารี่ OpenRDF Rio parser ใช้แปลงข้อมูลอาร์ดีเอฟหลายแบบ อาทิ RDF/XML Tri [129] Trix N-Triple RDF/JSON JSON-LD Sesame-BinaryRDF และ N-Quad ซึ่งผู้สนใจสามารถอ่านรายละเอียดของประเภทไฟล์เพิ่มได้จากเว็บไซต์ของ W3C

อีกประเด็นที่เกิดจากข้อมูลต้นฉบับคือการใช้อักขระพิเศษในข้อมูลประเภท URI และข้อความใน Object ทำให้บางครั้งตัวแปลงไม่สามารถแปลอักขระนั้นหรือสับสนระหว่างอักขระนั้นกับรูปแบบไฟล์เช่น เครื่องหมายมากกว่า น้อยกว่า ช่องว่าง อัญประกาศ ภาษาที่ผิดรูปแบบจาก UTF ที่ใช้ เป็นต้น ทำให้ต้องตัดข้อมูลบางส่วนออกไป ดังในงานวิจัยเรื่องกรอบการทำงานแบบ TripleID ที่นำเซตข้อมูลแบบอาร์ดีเอฟขนาดใหญ่เข้าไปคิวรี่ในหน่วยประมวลผลกราฟิกส์ [130] ผู้วิจัยได้ทำการเตรียมข้อมูลเพื่อทดสอบโดยแปลงข้อมูลจาก BTC-2009 [131] เซตที่ดาวน์โหลดมาคือ btc-2009-small.nq.gz [132] จากนั้นใช้คำสั่ง bin/rdfconvert --input ‘ประเภทอาร์ดีเอฟ’ <ชื่อไฟล์ต้นทาง> --output 'ประเภทอาร์ดีเอฟ' <ชื่อไฟล์ปลายทาง> ผลลัพธ์เป็นไปดังตารางที่ 4.2 ไฟล์ต้นฉบับขนาด ไฟล์ปลายทาง และจำนวนเทอมของ Subject Predicate และ Object ทั้งหมด พบว่าไฟล์ที่นำมาใช้วิจัยมีขนาดลดลงจากต้นฉบับ ดังนั้นหากต้องการใช้ไฟล์และจำนวนทรัพเพิลในการวิจัยเท่าไรต้องหาไฟล์ต้นฉบับที่มีขนาดใหญ่กว่า จากนั้นจึงแบ่งไฟล์เพื่อทำการแปลง เนื่องจากข้อจำกัดของเครื่องมือที่ใช้ป้อนคำสั่งคือสามารถแปลงใหญ่ได้ขนาดหนึ่งเช่น 500MB การแปลงและผลลัพธ์จะทำในหน่วยความจำจึงต้องเผื่อพื้นที่ไว้ขณะประมวล เมื่อแปลงแล้วจะได้ไฟล์เล็กลง และหยุดแปลงทันทีที่พบอักขระผิดปกติ ดังนั้นการแปลงไฟล์เพื่อนำมาทดลองถือเป็นขั้นตอนที่มีความสำคัญช่วยในการตรวจสอบข้อมูลด้วย

ตารางที่ 4.2 การแปลงไฟล์ต้นฉบับมาใช้ในงานวิจัย

เซตข้อมูล	ชื่อไฟล์ (รูปแบบ)	ขนาด	#triples	#subject	#predicate	#objects
ไฟล์ต้นฉบับ	btc-2009-small.nq	2.172GB	9,627,877	1,383,542	8,205	2,260,819
ไฟล์ที่ใช้วิจัย	012347.nt	1.611GB	7,083,790	1,113,824	7,542	1,674,407
ไฟล์ที่ใช้วิจัย	012347.rdf	1.556GB				

รายละเอียดของเทอม Subject Predicate และ Object ในหนึ่งไฟล์อาร์ดีเอฟจะมีปริมาณที่มีนัยสำคัญคือจำนวนของ Predicate จะมีน้อยที่สุด ในขณะที่จำนวน Subject ส่วนใหญ่แล้วมักจะมีน้อยกว่า Object โดยมีส่วนที่ซ้ำกันได้ในกรณีที่ข้อมูลเป็นเซตของ URI หรือ IRI

จากการที่ Predicate เป็นจำนวนที่น้อยที่สุดจึงมีการนำ Predicate มาใช้เป็นตัวแบ่งกลุ่มเพื่อการคิวรี่ จากสถิติของเซตข้อมูลอาร์ดีเอฟ BTC-2009 พบว่า Predicate ที่ใช้มากที่สุดคือ dbp ของ http://dbpedia.org/property โดยมีรายละเอียดดังนี้ dbp:wikilink 156,434,900 ทริพเพิลรองลงมาคือ rdf:type จำนวน 143,479,200 ทริพเพิลและ rdfs:seeAlso 53,852,300 ทริพเพิล ส่วนของ owl:sameAs จำนวน 6,539,300 ทริพเพิลและ dbp:redirect จำนวน 6,451,500 ทริพเพิล ต่อมาใน BTC-2012 ได้เน้นการคลอร์วข้อมูลจากเซตข้อมูลอื่นนอกจาก DBpedia ให้มี owl:sameAs ที่เชื่อมโยงมาที่ DBpedia ส่วน Predicate rdf:type ใช้มากขึ้นคือจำนวน 152 ล้านทริพเพิล [133]

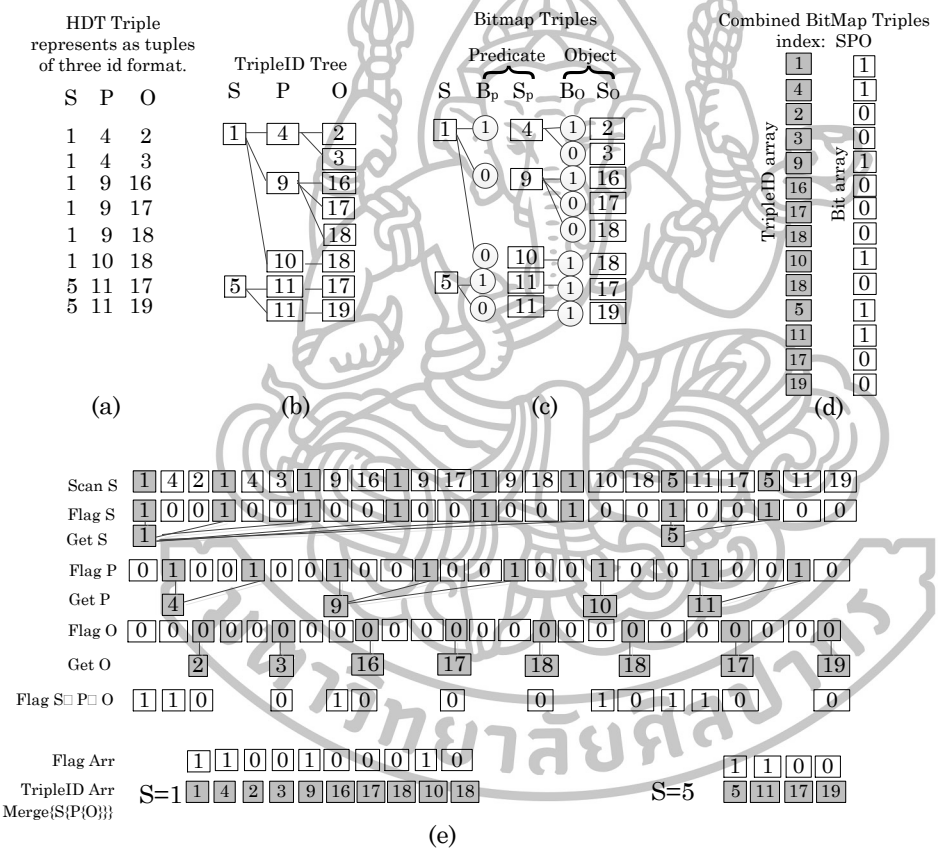
การดึงข้อมูล การแปลงข้อมูล และนำข้อมูลไปประมวลผล

เนื่องจากข้อมูลอาร์ดีเอฟจากลิงค์เดต้าส่วนใหญ่มีขนาดใหญ่กว่าหน่วยความจำของหน่วยประมวลผลกลางและมากกว่าหน่วยประมวลผลกราฟิกส์หลายเท่า จึงต้องทำการย่อข้อมูลเพื่อลดขนาดของข้อมูลที่นำเข้าไปคิวรี่ในหน่วยประมวลผลกราฟิกส์ ผู้วิจัยได้ศึกษา Header-Dictionary-Triples (HDT) ที่เป็นโครงสร้างข้อมูลต้นไม้ไบนารีแบบย่อดังรายละเอียดในบทที่ 2 หัวข้อย่อยรูปแบบอาร์ดีเอฟแบบไบนารี จาก*ขั้นตอนการสร้างรูปแบบอาร์ดีเอฟให้เป็นรูปแบบย่อเอชดีที* แล้วนำมาต่อยอดเพื่อให้เหมาะกับวัตถุประสงค์การนำเซตข้อมูลลิงค์เดต้าเข้าไปค้นหาในหน่วยประมวลผลกราฟิกส์ อยู่ในขั้นตอนก่อนประมวลผล (3) ดัง*รูปที่ 3.8 ภาพรวมขั้นตอนการค้นหาข้อมูลแบบลิงค์เดต้า*โดยแปลงเป็นแถวลำดับบิตแมพ (Combined BitMap Array) ซึ่งสร้างดัชนีสองแบบคือ Subject Predicate Object (SPO) และ Predicate Object Subject (POS)

ตารางที่ 4.3 รูปแบบการคิวรี่จากรูปแบบดัชนีของทริพเพิล

ลำดับที่	รูปแบบการคิวรี่ (? แทน เทอมที่ต้องการหาคำตอบ)	ดัชนี
1	(spo) (sp?) (s?o) (s??) (???)	SPO
2	(spo) (?po) (?p?) (??o) (???)	POS

การสร้างดัชนีแต่ละแบบส่งผลต่อรูปแบบการคิวรี่ดังตารางที่ 4.3 จากดัชนีสองแบบที่ผู้วิจัยสร้างให้ซีบีเอ็มคือ SPOและ POS สามารถคิวรี่ได้ 8 รูปแบบ ซึ่งแตกต่างจากงาน cumulusRDF [47] ที่มีการสร้างดัชนีแบบที่ 3 เพิ่มคือ ((Object Subject) Predicate) OSP ด้วย เพื่อคิวรี่ใน 8 รูปแบบดังกล่าว ธิบายตารางที่ 4.3 ตามรูปแบบการคิวรี่ โดย ? แทน เทอมที่ต้องการหาคำตอบ ไฟล์ทริพเพิลต้นฉบับจะเรียงตัวแบบ SPO ไม่มีการใช้ดัชนี แต่สามารถใช้มิดเดิลแวร์และภาษาสปาร์เคิลในการคิวรี่ได้ สำหรับไฟล์เอชดีทีนั้นมีเพียงดัชนี SPO (2013) ที่ต้องใช้โปรแกรม hdt-it หรือคำสั่งเพื่อคิวรี่ได้ทั้ง 8 แบบ คือ (spo) (sp?) (s?o) (s??) (?po) (?p?) (??o) (???) แต่จากการทดสอบโปรแกรมพบว่าอาจมีการเลื่อนของดัชนีได้เล็กน้อยหลังจากการค้นหา



รูปที่ 4.1 ขั้นตอนการสร้างแถวลำดับเพื่อเตรียมข้อมูลก่อนนำไปประมวลผลในหน่วยประมวลผลกราฟิกส์

ขั้นตอนการสร้างแถวลำดับเพื่อเตรียมข้อมูลนำไปประมวลผลในหน่วยประมวลผลกราฟิกส์ จากรูปที่ 4.1 (a) ทัพเพิลสามคอลัมน์ของคีย์ SPO ถูกเก็บเป็นทริพเพิลของเอชดีที รูปที่ 4.1 (b) คือโครงสร้างต้นไม้ภายในเอชดีทีที่มีรากคือ S ความลึกชั้นที่ 1 คือ P และ ชั้นที่ 2 คือ O รูปที่ 4.1 (c) คือโครงสร้างต้นไม้ภายในเอชดีทีที่มีรากคือ S แบ่งเป็นสองต้น ต้นแรกมีความลึกชั้นที่ 1 คือ บิตแมพของ P และมีความลึกชั้นที่ 2 คือ บิตแมพของ O โดยให้ตัวแรกที่มีโหนดแม่เดียวกันเป็น 1

นอกนั้นเป็น 0 อีกต้นหนึ่งคือต้นไม้จาก (b) สำหรับรูปที่ 4.1 (e) ผู้วิจัยทำต่อจากเอชดีทีภาพ (a) จากบนลงล่าง เริ่มจากขั้น Scan S คือสแกนหา Subject จากลำดับทริพเพิลที่ผ่านการเรียงลำดับแล้ว ทำขั้น Flag S คือทำเครื่องหมาย 1 ไว้ทุกตำแหน่งของ Subject จากนั้นขั้น Get S ลดรูปให้เหลือเพียงค่าเดียวตั้งเป็นค่าแรกของแต่ละ Subject ลำดับถัดมา Flag P คือทำเครื่องหมาย 1 ไว้ทุกตำแหน่งของ Predicate จากนั้นลดรูปให้เหลือเพียงค่าเดียวตั้งเป็นค่าต่อมาของ Subject ที่ขั้น Get P ลำดับสุดท้าย Flag O คือทำเครื่องหมาย 0 ไว้ทุกตำแหน่งของ Object ตั้งเป็นลำดับถัดมาจาก Predicate ผลลัพธ์มีสองแถวลำดับคือแถวลำดับทริพเพิล (TripleID array) มาจากการรวม {S{P{O}}} และแถวลำดับบิตที่มาจากการยูเนี่ยนต่างบิตตามตำแหน่ง S=1, P=1 และกลุ่มของ O ใน P และ S เดียวกันเข้าด้วยกัน

อัลกอริทึมที่ 4.1 เป็นตัวอย่างการสร้างแถวลำดับ SPO บรรทัด 8-11 ดึงลำดับทุกทริพเพิลจากเอชดีที บรรทัด 12-20 ทำเครื่องหมายตำแหน่ง S บรรทัด 21-42 กำจัดข้อมูลซ้ำ ดึงลำดับทริพเพิลที่เป็นเอกลักษณ์ออกมาสร้างเป็นแถวลำดับทริพเพิลและแถวลำดับบิตจากบิตตำแหน่งเดียวกัน บรรทัด 43-46 บรรทัด 47-50 คือใช้ชุดคำสั่งเดียวกันแต่เป็นตัวแปรสำหรับ P และ O ตามลำดับเพื่อสร้างแถวลำดับทริพเพิลและบิต สำหรับ P และ O ตามลำดับ

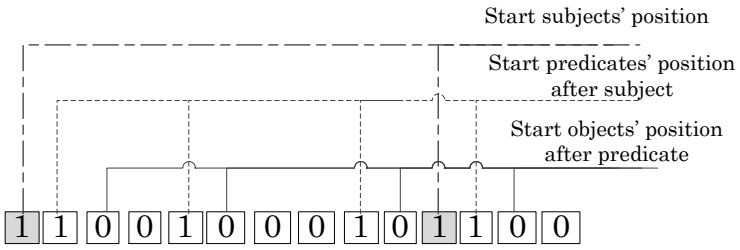
ตารางที่ 4.4 การคาดคะเนพื้นที่และเวลาที่ใช้เก็บและค้นหาสำหรับโครงสร้างเอชดีทีและซีบีเอ็ม

ลำดับ	พื้นที่ในการเก็บ	เวลาในการค้นหาโดยเฉลี่ย	เวลาในการเข้าถึงเฉลี่ย
A	$O(3n)$	$O(3n)$	$O(1)$
B	$O(n)$ และ < a	$O(2\log(n))$	N/A
C	2b	$O(4\log(n))$	$O(4\log(n))$
D	< c < b < a	$O(2n)$	$O(1)$

ถ้าให้จำนวนแต่ละหลักของ SPO เป็น *n* แล้วเวลา พื้นที่และการเข้าถึงได้ตามตารางที่ 4.4 วัตถุประสงค์ในการแปลงคือต้องการข้อมูลที่มีขนาดเล็กที่สุดเพื่อที่จะได้ขนย้ายเข้าไปประมวลผลในหน่วยประมวลผลกราฟิกส์ได้มากที่สุดในการคิวรี่แต่ละรอบ สรุปคือ การนำข้อมูลโครงสร้าง (a) มาแปลงเป็นแถวลำดับแบบ (d) พบว่าพื้นที่ในการเก็บข้อมูล (d) น้อยกว่า (a-c) ดังผลการทดลองในตารางที่ 4.6

อัลกอริทึม 4.1 การแปลงต้นไม้ HDT เป็นอาเรย์ Subject Predicate Object

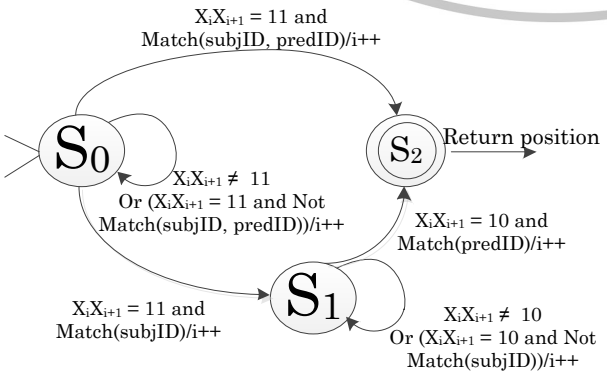
```
1  declare integer h_subject[rangeS], h_predicate[rangeP], h_object[rangeO]
2  declare integer flagS[rangeS+1], flagP[rangeP+1], flagO[rangeO+1]
3  declare integer h_TripleID[rangeElement], flagE[rangeElement+1]
4  declare integer countS=0, countP=0, countO=0, countE=0, s=0, p=0, o=0
5  declare HDT hdt_structure
6  declare IteratorTripleID hdt_structure.getTriples
7  declare TripleID tripleIDset

8  //Extract all triple ID from HDT factory
9  tripleIDset ← null
10 while (hdt_structure.getTriples.hasNext == true)
11   tripleIDset ← get next Triples of hdt_structure
12   //Fill in flag of subject array
13     for (s =0, s < rangeS, s ++)
14        if(s%3 isEqual 0)
15        {
16         flagS[s] = 1
17         h_sdata[countS] = get subject of tripleIDset
18        }else {
19         flagS[s] = 0
20        }

21   //Fill in array of subject ID
22     if ((countS isGreaterThan or equal to 1))
23     {
24       if (h_subject[countS] == h_subject[countS - 1])
25       {
26       // First subject case: checking duplicate Subject ID
27         h_TripleID[countE] = h_subject[countS - 1]
28         countE+1
29        } else {
30         h_TripleID[countE] = h_subject[countS]
31           append Subject ID array of tripleIDset
32           append Bit array of tripleIDset = '1'
33           countE+1
34           }
35     } else {
36            h_TripleID[countE] = h_subject[countS]
37            append Subject ID array of tripleIDset
38             //append Bit array of tripleIDset = '1'
39             flagE[countE]='1'
40             countE+1
41     }
42    countS+1

43       //Check Predicate position: if(p%3 isEqual 1)
44          // First predicate case: checking duplicate Predicate ID
45          // append Predicate ID array of tripleIDset
46          // append Bit array of tripleIDset = '1'

47       //Check Predicate position: if(o%3 isEqual 2)
48          // First predicate case: checking duplicate Object ID
49          // append Object ID array of tripleIDset
50          // append Bit array of tripleIDset = '0'
```



รูปที่ 4.2 ตำแหน่งของบิตแมพผสมเพื่อการค้นหาบิตตามตำแหน่งของทริพเพิล

รูปแบบที่นักวิจัยนำเสนอเพื่อประหยัดพื้นที่เก็บข้อมูลนี้ ใช้การดึงหมายเลขทริพเพิลแบบเลขจำนวนเต็มจากพจนานุกรมของเอชดีที โดยเรียกว่า Combined BitMap (CBM) ซึ่งซีบีเอ็ม [134] นี้คือการทำโครงสร้างบิตแมพผสม เมื่อค้นหาบิต จะได้ผลลัพธ์เป็นตำแหน่งของบิตเริ่มต้นจากรูปที่ 4.2 แสดงว่าเลข 1 เป็นตำแหน่งของ subject และ predicate และ 0 เป็นตำแหน่งของ object ข้อสังเกตจากรูปแบบของอาร์ดีเอฟทั่วไปที่มีจำนวน object ได้ไม่จำกัดส่งผลให้รูปแบบนี้มี 0 สำหรับ predicate เดียวกันอย่างไม่จำกัดเช่นกัน สำหรับในอนาคตจะลดรูป 0 ได้ N ตัวในกรณีที่ N เป็นจำนวนของ 0 ทำให้สามารถย่อพื้นที่เก็บข้อมูลได้อีก

ผู้วิจัยนำรูปแบบนี้มาใช้ในการเขียนสถานะการค้นหา (searching state) ในรูปแบบซีบีเอ็ม เมื่อค้นหาบิตโดยให้ส่งค่าตำแหน่งที่เจอกลับมา จากรูปที่ 4.3 สถานะ S_0 ค้นหาค่า '11' ที่เป็นตำแหน่งเริ่มต้นของลำดับ subject (subject ID) จากนั้นตรวจสอบว่า ถ้าลำดับนั้นตรงกัน สถานะจะเปลี่ยนเป็น S_1 การตรวจสอบลำดับของ predicate (predicate ID) โดยหาตำแหน่งของ 10 เมื่อลำดับของ predicate ที่ค้นหาตรงกัน ได้ค่าแรกของลำดับ object (object ID) สถานะเปลี่ยนเป็น S_2 โดยการค้นหาดังกล่าวนี้ต้องค้น subject แล้วตามด้วย predicate



รูปที่ 4.3 สถานะของการค้นหา (searching state)

เมื่อพิจารณาการพัฒนาโปรแกรมบนหน่วยประมวลผลกราฟิกส์แล้ว พบว่าทั้งแถวลำดับของเลขลำดับทริพเพิลและซีบีเอ็มต้องถูกนำเข้าไปที่หน่วยความจำของหน่วยประมวลผลกราฟิกส์ทั้งหมด ดังนั้นพื้นที่ของหน่วยความจำนี้ขึ้นอยู่กับขนาดของแถวลำดับของเลขทริพเพิล เทรดจำนวนมากถูกนำมาใช้ค้นหา “11” และ “10” ซึ่งลำดับของการค้นหาตามข้อมูลที่เรียงตาม subject และ predicate ตามลำดับ เรียกดัชนีแบบนี้ว่า SPO ในการทดลองจะนำมาเทียบกับการค้นหาบนหน่วยประมวลผลกลาง

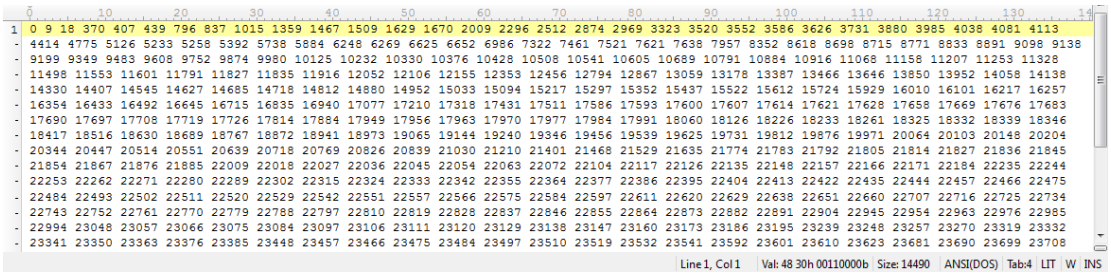
ตารางที่ 4.5 กรอบการทำงานระหว่างโฮสต์และดีไวส์ของโครงสร้างซีบีเอ็ม

ประมวลผล (เครื่อง)			กระบวนการ	ข้อมูลเข้า	ข้อมูลออก
ก่อน (โฮสต์)	ค้นหา(ดีไวส์)	หลัง (โฮสต์)			
√			แปลงข้อมูลทริพเพิลเป็นเอชดีที	ทริพเพิล	เอชดีที
√			แปลงเซตข้อมูลเอชดีทีเป็นซีบีเอ็ม	เอชดีที	ซีบีเอ็ม
√			แบ่งคิวรี่เป็นคิวรี่ย่อย	คิวรี่	คิวรี่ย่อย
√			แปลงคิวรี่เป็นลำดับทริพเพิล (ID)	คิวรี่ย่อย	ลำดับทริพเพิล
	√		ค้นหาซีบีเอ็มแบบขนานโดยคู่ค้า	ซีบีเอ็ม	ตำแหน่งที่พบ
		√	นำตำแหน่งที่พบมาเทียบลำดับทริพเพิล	ตำแหน่งที่พบ	ลำดับทริพเพิล
		√	แปลงลำดับทริพเพิลเป็นทริพเพิล	ลำดับทริพเพิล	ทริพเพิล
		√	รวมผลลัพธ์คืนผู้ใช้	ทริพเพิล	ข้อความ

ตารางที่ 4.5 สรุปกรอบการทำงานระหว่างโฮสต์และดีไวส์ของโครงสร้างซีบีเอ็มตามลำดับของกระบวนการที่เกิดขึ้นในขั้นตอนก่อนการประมวลผลบนหน่วยประมวลผลกลาง ขั้นประมวลผลหรือค้นหาบนหน่วยประมวลผลกราฟิกส์ และขั้นหลังการประมวลผลบนหน่วยประมวลผลกลาง รวมทั้งข้อมูลเข้าและออกจากระบบ ต่อไปเป็นการแสดงรายละเอียดในขั้นตอนการค้นหารูปแบบซีบีเอ็มบนหน่วยประมวลผลกราฟิกส์ดังอัลกอริทึมที่ 4.2 ให้ทุกเทรด (บรรทัด 1) เปรียบเทียบบิตซึ่งแทนค่าเป็น ‘11’ เมื่อต้องการค้นหา subject และ ‘10’ เมื่อต้องการค้นหา predicate (บรรทัด 2) เมื่อค้นพบในตำแหน่งใดให้ทำค่าถูกที่ตำแหน่งนั้น (บรรทัด 3)

อัลกอริทึม 4.2 การค้นหารูปแบบซีบีเอ็มบนหน่วยประมวลผลกราฟิกส์

```
Input:   dataCBM, Bits, where Bits are "11" or "10"
Output:  positionArray
1        for i= threadIdx; i<n; i+=blockDim * gridDim do
2            if dataArray[i] == Bits then
3                positionArray[i] = true
4            end
5        end
```

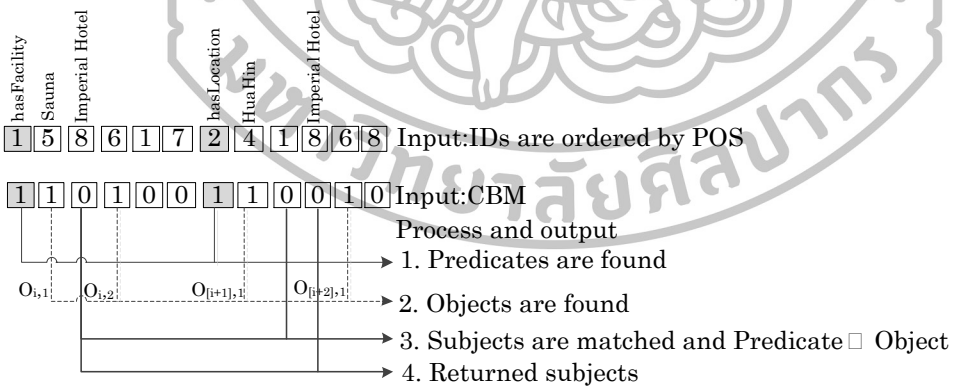


รูปที่ 4.4 ผลการค้นหาทุก subject จากซีบีเอ็ม

การจัดการคิวรี่ย่อยจากดัชนี POS เป็นการแบ่งคิวรี่ด้วยดัชนีเดียวกับที่ใช้จัดการกับเซตข้อมูลอาร์ดีเอฟ กำหนดให้มีคิวรี่ “select spa which have some sauna facility and are located in Hua Hin” ภาษาสปาร์เคิลเป็นดังคิวรี่ด้านล่าง เป็นการถามแบบ (?PO)

```
PREFIX ont: <http://tourinthailand.org/ontology/healthtour.owl#>
SELECT ?spaName
WHERE {
  ?spaName ont:hasFacility ont:Sauna  . (1)
  ?spaName ont:hasLocation ont:HuaHin . (2)
}
```

คิวรี่ย่อยที่ (1) และ (2) มีรูปแบบ (?po) จึงต้องใช้ดัชนี POS ซึ่งจัดการทำนองเดียวกับดัชนี SPO ข้างต้น เพียงแต่สลับความหมายของบิตโดย ‘11’ คือ predicate และ ‘10’ คือ object



รูปที่ 4.5 การค้นซีบีเอ็มตามดัชนี POS เพื่อหาคำตอบของคิวรี่ย่อยบนหน่วยประมวลผลกราฟิกส์

จากรูปที่ 4.5 กระบวนการเริ่มจากค้นหาลำดับทริพเพิลที่มี ID ตรงกับ predicate จากคิวรี่ย่อย (1) และ (2) คือ hasFacility (ID= {1}) และ hasLocation ตามลำดับ (ID= {2}) นั่นคือค้น

รูปแบบซีบีเอ็มเพื่อหาบิต ‘11’ จากนั้นหา object ด้วยบิต ‘10’ ตามลำดับด้วยลำดับทริพเพิลที่มี ID ตรงกับ object จากคิวรี่ย่อย (1) และ (2) คือ Sauna (ID= {5}) และ Huahin (ID= {4}) ตามลำดับ ต่อมานำ Predicate และ Object มาทำอินเตอร์เซคกัน เพื่อหา subject ที่อยู่ในกลุ่มของทั้ง predicate และ object ที่กำหนด คือ {{8} ∩ {1,8}} สุดท้ายคืนค่า subject ID = {8}

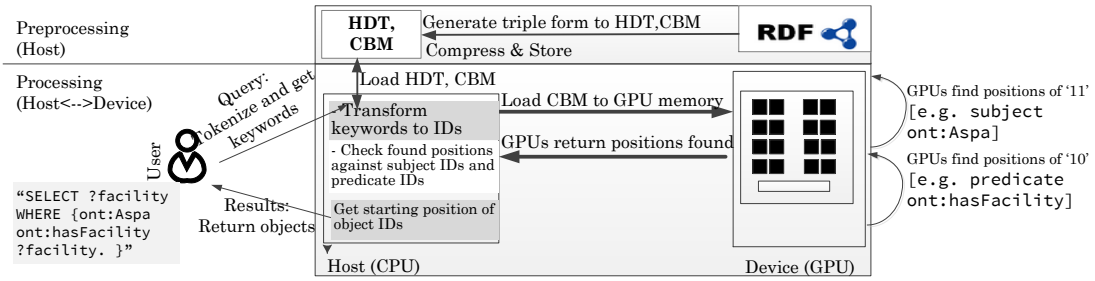
ในทำนองเดียวกันการตอบคำถามแบบ ?P? ใช้ดัชนี POS เช่นเดียวกัน เมื่อคิวรี่ “find spas with facilities and their locations.” จากรูปที่ 4.5 จะตอบคำถาม (3) และ (4) ได้ดังนี้

```
PREFIX ont: <http://tourinthailand.org/ontology/healthtour.owl#>
SELECT ?spaName
WHERE {
  ?spaName ont:hasFacility ?facility  . (3)
  ?spaName ont:hasLocation ?place     . (4)
}
```

คำตอบของ (3) คือ {1 {5 {8}, 6 {1,7}}} คำตอบ (4) คือ {2 {4 {1,8}, 6{8}}} คำตอบที่ได้ คือ subject ID = {8}

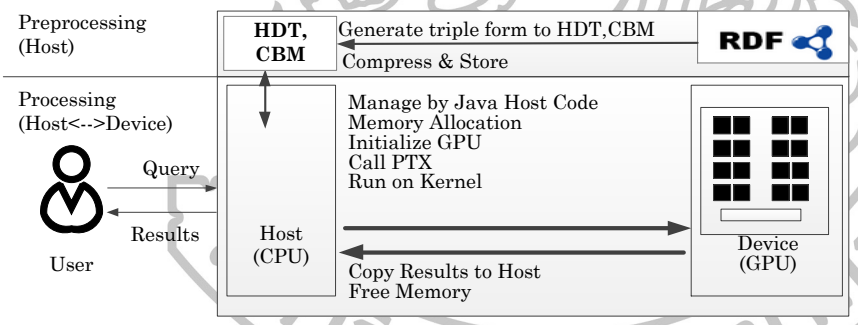
วิธีทดสอบ มีสองแบบแบ่งตามภาษาที่ใช้พัฒนา

การทดสอบด้วยโปรแกรมภาษาซีประกอบด้วยการหาคำตอบว่าการเก็บรูปแบบซีบีเอ็มนั้นลดพื้นที่เก็บข้อมูลจากรูปแบบเอชดีทีเท่าไร และเมื่อนำเข้าไปประมวลผลบนหน่วยประมวลผลกราฟิกส์แล้วมีความเร็วกว่าการประมวลผลการค้นหาบนหน่วยประมวลผลกลางเท่าไร และหาจำนวนของเทรดต่อบล็อกที่เหมาะสม รวมทั้งเวลาที่ใช้รับส่งข้อมูล สถาปัตยกรรมระบบเป็นไปดังรูปที่ 4.6 สถาปัตยกรรมระบบการแปลงอาร์ดีเอฟด้วยการประยุกต์โครงสร้างเอชดีทีมาเป็นแบบซีบีเอ็มจากรูปที่ 4.6 (ตัวอย่างเป็นดัชนี SPO) ดังที่อธิบายรูปที่ 4.3 เริ่มจากขั้นตอนการเตรียมข้อมูลที่ทำงานบนโฮสต์คือโหลดและเก็บโครงสร้างเอชดีที และซีบีเอ็ม นั้นคือแปลงคีย์เวิร์ดเป็นลำดับเลขและบิตทั้งเซตข้อมูลและคิวรี่ จองพื้นที่หน่วยความจำ โหลดข้อมูลเข้าสู่หน่วยความจำ จากนั้นย้ายไปทำการประมวลผลบนดีไวส์ หาบิต ‘11’ และ ‘10’ สำหรับ subject และ predicate ตามลำดับในกรณี SPO และ predicate และ object ตามลำดับในกรณี POS เมื่อการทำงานเสร็จสิ้นนำผลลัพธ์กลับสู่โฮสต์ จากรูปที่ 4.6 เป็นการประมวลผลคำถามด้านซ้ายล่างจากผู้ใช้คำตอบที่คืนมาถึงเป็น object ของคิวรี่สปาร์เคิล เมื่อเสร็จงานก็คืนหน่วยความจำของหน่วยประมวลผลกราฟิกส์ จากนั้นมีกระบวนการแปลงข้อมูลต่อเพื่อส่งผลลัพธ์แก่ผู้ใช้งาน ดังสรุปในตารางที่ 4.5



รูปที่ 4.6 สถาปัตยกรรมระบบการนำซีบีเอ็มมาประมวลผลบนหน่วยประมวลผลกราฟิกส์

การทดสอบด้วยภาษาจาวา สถาปัตยกรรมระบบแบบใช้ภาษาจาวาเพื่ออำนวยความสะดวกต่อผู้ใช้โปรแกรมเอชดีทีภาษาจาวาทำการแปลงการแปลงอาร์ดีเอฟด้วยการประยุกต์โครงสร้างเอชดีทีมาเป็นโครงสร้างแบบซีบีเอ็มด้วยภาษาจาวาอย่างต่อเนื่อง หน่วยประมวลผลกราฟิกส์ด้วยดัชนี SPO ในขั้นต้นนี้ เริ่มจากขั้นตอนการเตรียมข้อมูลที่ทำงานบนโฮสต์ จองพื้นที่หน่วยความจำ เริ่มการทำงานจากฝั่งดีไวส์ เรียกใช้ไฟล์ PTX ทำงานในเคอร์เนล เมื่อการทำงานเสร็จสิ้นนำผลลัพธ์กลับสู่โฮสต์ แล้วล้างหน่วยความจำ จากนั้นมีกระบวนการแปลงข้อมูลกลับบนโฮสต์ต่อเพื่อส่งผลลัพธ์แก่ผู้ใช้งาน ดังสรุปในตารางที่ 4.5



รูปที่ 4.7 สถาปัตยกรรมระบบการนำซีบีเอ็มมาประมวลผลบนหน่วยประมวลผลกราฟิกส์ด้วยเจคูด้า

คุณสมบัติของเครื่องที่ใช้ทดสอบ ประกอบด้วย

เครื่องที่ 1 เครื่องเซิร์ฟเวอร์เมฆาของคณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์ ฮาร์ดแวร์ ประกอบด้วย 2 x CPU Intel ® Xeon ® CPU E5620 @ 2.40GHz, RAM 16 กิกะไบต์ ซอฟต์แวร์ระบบปฏิบัติการ Red Hat 4:4:7 - 11 (x86-64) และโปรแกรม Java 1:7.79 เครื่องนี้ทำหน้าที่แปลงทริพเพิลจากเอชดีทีมาเป็นซีบีเอ็ม

เครื่องที่ 2 เครื่องเทสลา K40c ประกอบด้วย 2x หน่วยประมวลผลกราฟิกส์รุ่น Tesla K40c แต่ละการ์ดมีมัลติโพรเซสเซอร์ 15 หน่วย 192 แกนคูด้า/มัลติโพรเซสเซอร์ สัญญาณนาฬิกาอัตรา 745 MHz พื้นที่หน่วยความจำโกลบอล 12,288 MBytes สำหรับหน่วยประมวลผลกลางคือ Intel ® Core™ i7-5820K CPU @ 3.30GHz มี 6 แกน หน่วยความจำขนาด 16 GB เครื่องนี้ทำหน้าที่ประมวลผลการค้นหาบนหน่วยประมวลผลกราฟิกส์ โปรแกรม NVIDIA CUDA 7.0

เครื่องที่ 3 การทดสอบกับไลบรารี่เจคูด้า และแปลงข้อมูลจากทริพเพิลเป็นเอชดีทีฮาร์ดแวร์ Intel ® Core ™ i5 CPU 4 แกน 4 เทรด ส่วน GPU คือ GeForce GTX 560 Ti มี 384 แกน ซอฟต์แวร์ ระบบปฏิบัติการ Windows ® 7 มิดเดิลแวร์อาปาเช่ เจนา และ Jcuda 0.5a กับ NVIDIA CUDA 5.0

ผลการทดลอง

ผลการทดลองของการค้นหาข้อมูลแบบอาร์ดีเอฟแบบผ่านการแปลงข้อมูลก่อนนำเข้าหน่วยประมวลผลกราฟิกส์มีดังต่อไปนี้ [105]

สำหรับการแปลงไฟล์เอชดีทีมาเป็นไฟล์ซีบีเอ็มด้วยดัชนี SPO และ POS เป็นการเรียงลำดับตาม subject predicate object และ predicate object subject ตามลำดับ กรณี SPO เป็นการจัดตามข้อมูลเอชดีที ต้นฉบับทั่วไป ที่มักเรียงตาม subject ในขณะที่ POS เป็นการจัดกลุ่มจาก predicate ที่ปกติจะมีจำนวนน้อยกว่า subject และ object เสมอ จากผลงานวิจัยเช่น [47] เชื่อว่าทำให้ลำดับการเข้าถึงลดลง และเชื่อมการค้นหาได้หลายแบบมากขึ้น เพื่อรองรับการคิวรีด้วย SPARQL ในอนาคต (ดูตาราง 4.3 ประกอบ) ดังนั้นการแปลงข้อมูลรูปแบบซีบีเอ็มที่ใช้รูปแบบเอชดีทีเป็นฐานจะช่วยลดขนาดเซตข้อมูลลง ผู้วิจัยจึงทำการเพื่อให้เห็นความชัดเจน โดยเปรียบเทียบการแปลงข้อมูลลงฐานข้อมูล โครงสร้างไฟล์เอชดีทีและไฟล์ซีบีเอ็มดัง

ตารางที่ 4.6 แสดงชื่อเซตข้อมูลสามารถดูประกอบกับตารางที่ 4.1 ประกอบด้วยสามชุดหลักคือ เซตข้อมูล DBpedia1/20-BTC2012 หมายถึงชุดข้อมูล DBpedia/BTC2012 ที่แบ่งใช้ 1 ใน 20 ส่วนของไฟล์เดิม เซตข้อมูล Freebase1/10- BTC2012 หมายถึงชุดข้อมูล Freebase/BTC2012 ที่แบ่งใช้ 1 ใน 10 ส่วนของไฟล์ Healthtour-2014-04-08 [135] เป็นไฟล์ที่สร้างขึ้นเพื่อใช้ในงานวิจัยเกี่ยวข้องกับการท่องเที่ยวเชิงสุขภาพในเขตเทศบาลเมืองหัวหิน [106] ถัดมาคือคอลัมน์ Index เป็นดัชนี SPO หรือ POS ที่เป็นรูปแบบของไฟล์นั้น คอลัมน์ Original Size คือขนาดของไฟล์แบบทริพเพิลในหน่วยเมกะไบต์และกิกะไบต์ คอลัมน์ Triples คือจำนวนของทริพเพิลในไฟล์นั้น คอลัมน์ Red (%) หมายถึงเปอร์เซ็นต์การลดลงจากไฟล์ทริพเพิลแบบ NT สำหรับแต่ละรูปแบบ คอลัมน์ MySQL Database คือใช้ฐานข้อมูล MySQL ที่ใช้เก็บเซตข้อมูลทริพเพิลที่แปลงตามดัชนี SPO และ POS

แบ่งเป็นคอลัมน์ โดย Size ที่แสดงขนาดของตารางนั้น คอลัมน์ HDT คือไฟล์เอชดีทีที่แปลงมาจากไฟล์ทริพเพิลตามดัชนี SPO และ POS แบ่งเป็นคอลัมน์ Size ที่แสดงขนาดของไฟล์เอชดีทีนั้น คอลัมน์ CBM คือไฟล์ซีบีเอ็มที่แปลงมาจากไฟล์เอชดีทีตามดัชนี SPO และ POS แบ่งเป็นคอลัมน์ Size ที่แสดงขนาดของไฟล์ซีบีเอ็มนั้น

ตารางที่ 4.6 เปรียบเทียบการแปลงข้อมูลลงฐานข้อมูล ไฟล์เอชดีทีและไฟล์ซีบีเอ็ม

Dataset	Index	Original Size	Triples	MySQL Database		HDT		CBM	
				Size	Red (%)	Size	Red (%)	Size	Red (%)
DBpedia1/20-BTC2012	SPO	1.78GB	9.9×10^6	1.29GB	28	507.4MB	71	135MB	92
DBpedia1/20- BTC2012	POS	1.7GB	9.9×10^6	1.21GB	29	565.0MB	66	108MB	93
Freebase1/10- BTC2012	SPO	1.2GB	9.9×10^6	1.37GB	-14	109.4MB	91	107MB	91
Freebase1/10- BTC2012	POS	1.2GB	9.9×10^6	1.03GB	14	140.5MB	88	112MB	90
Healthtour-2014-04-08	SPO	5.6MB	23,654	5.13MB	8	844.3KB	85	220KB	96
Healthtour-2014-04-08	POS	5.6MB	23,654	3.21MB	43	553.4KB	90	190KB	96

เครื่องมือที่ใช้วัดค่าในการทดลองนี้คือ HDT-it [136] ในส่วนการแปลงไฟล์ใช้คำสั่ง rdf2hdt บนเซิร์ฟเวอร์เมฆา ซึ่งเป็นหนึ่งในโปรแกรมของงานวิจัยเอชดีที การแปลงไฟล์ก่อนนำไปประมวลผลพบว่าการแปลงไฟล์จากไฟล์ทริพเพิลเป็นฐานข้อมูลนั้นสามารถลดขนาดลงได้ 14-43 % ในกรณีที่เป็นดัชนี POS แต่กรณีที่เป็นดัชนี SPO พบว่ามีกรณีที่ไฟล์ใหญ่กว่าเดิม ในกรณีนี้คาดว่าเกิดจากการแปลงโดยแยกเทอมของ subject predicate และ object จากกันด้วยช่องว่างแล้วเกิดการตัดบางอักขระประเภทสตริงใน object กลายเป็นเรคคอร์ดใหม่ในตาราง การแปลงไฟล์จากไฟล์แบบทริพเพิลเป็นแบบเอชดีทีนั้นสามารถลดขนาดลงได้ 71-91 % ในกรณีที่เป็นดัชนี SPO และลดลง 66-90 % ในกรณีที่เป็นดัชนี POS การแปลงไฟล์จากไฟล์เอชดีทีเป็นซีบีเอ็มนั้นสามารถลดขนาดลงจากไฟล์แบบทริพเพิลได้ 91-96 % ในกรณีที่เป็นดัชนี SPO และลดลงจากไฟล์ทริพเพิลได้ 90-96 % ในกรณีที่เป็นดัชนี POS ดังนั้นซีบีเอ็มจึงช่วยลดขนาดไฟล์ทริพเพิลได้ดีที่สุดในการทดลองนี้ เนื่องจากข้อมูลทริพเพิลนั้น มีการใช้ predicate ร่วมกันมาก

สำหรับเวลาที่ใช้ในการแปลงไฟล์ในแต่ละขั้นเป็นไปดังตารางที่ 4.7 โดยใช้เครื่องเมฆา พบว่าเวลาที่ใช้ในการแปลงจากเอชดีทีเป็นซีบีเอ็มนั้นมากกว่าเวลาที่ใช้ในการแปลงจากทริพเพิลเป็นเอชดีทีถึงแปดพันเท่า

ตารางที่ 4.7 เวลาที่ใช้ในการแปลงไฟล์ระหว่างทริพเพิลเป็นเอชดีทีและเอชดีทีเป็นซีบีเอ็ม

Dataset name (raw triples)	NT-> HDT Generated Time (minutes)	HDT-> CBM Generated Time (minutes)
Freebase 10M	1.02	8,888
DBpedia0 10M	24.45	3,769

ตารางที่ 4.8 การค้นหา subject จากรูปแบบต้นไม้ไบนารีย่อแบบเอชดีทีบนหน่วยประมวลผลกราฟิกส์

Dataset	data size(Bytes)	time-consuming (s)	blocks/grid	threads/block
Freebase	1,793,413	7.16	1588	1024
DBpedia	1,626,803	6.54	1751	1024

จากการทดสอบโดยใช้เครื่องที่ 3 เพื่อค้นหารูปแบบไบนารีย่อแบบเอชดีทีบนหน่วยประมวลผลกราฟิกส์ GTX560Ti ดัง

ตารางที่ 4.8 แสดง ชื่อของเซตข้อมูล ขนาดข้อมูล เวลาที่ใช้ ขนาดบล็อกต่อกริด และขนาดของเทรดในหนึ่งบล็อก พบว่า เมื่อทดสอบการค้นหา subject แบบอักขระเดียวพบว่าใช้เวลา 7.2 วินาทีสำหรับข้อมูลขนาด 1.7 MB และใช้เวลา 6.54 วินาที สำหรับข้อมูลขนาด 1.6 เมกะไบต์

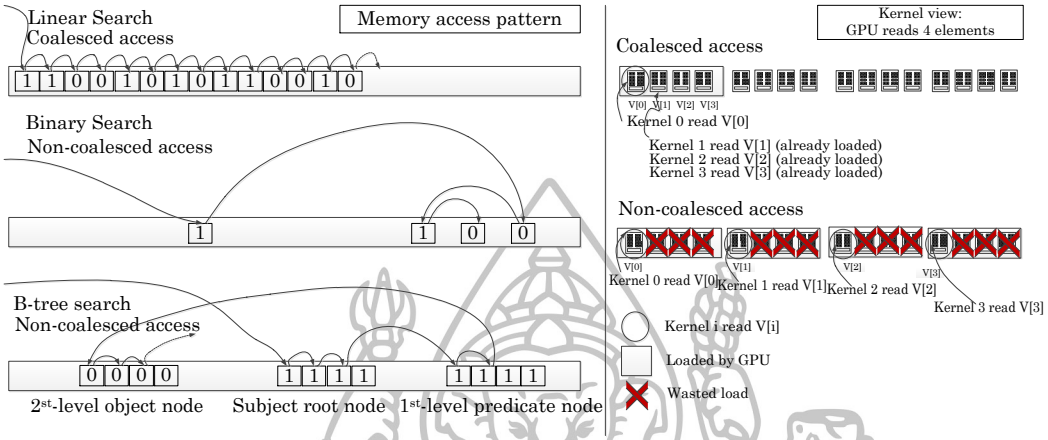
ตารางที่ 4.9 การค้นหาโดยใช้ดัชนี SPO จากรูปแบบอะเรย์ซีบีเอ็มบนหน่วยประมวลผลกราฟิกส์

HDT	Data size (Bytes)	Data size (bit)	S(11) (positions)	P(10) (positions)	O(00) (positions)	O(01) (positions)	Times (ms)	blocks/ grid	threads/ block
Freebase	1,787,921	14,303,370	386,361	3,932,948	6,037,265	3,932,827	28.32	13969	1024
DBpedia	2,291,421	18,331,369	1,580,553	6,845,230	6,845,180	3,042,504	34.17	17902	1024

การค้นหาโดยใช้ดัชนี SPO จากรูปแบบอะเรย์ซีบีเอ็มบนหน่วยประมวลผลกราฟิกส์ดังตารางที่ 4.9 พบว่าข้อมูล 1.7 เมกะไบต์ นั้นใช้เวลาในการหา subject (11) predicate (10) object {00, 01} ด้วยรูปแบบดังในวงเล็บ ทั้งหมดเป็น 28.32 มิลลิวินาที ส่วนข้อมูล 2.3 เมกะไบต์ ใช้เวลา 34.17 มิลลิวินาที

การที่รูปแบบซีบีเอ็มทำการค้นหาได้เร็วกว่านั้นเป็นเพราะใช้การค้นหาแบบเชิงเส้น ที่ส่งผลให้การเข้าถึงหน่วยความจำเป็นไปแบบเรียงกัน ส่วนของการค้นหาข้อมูลที่มีการทำดัชนี B-trees เพื่อให้การค้นหาเป็นไปอย่างรวดเร็ว หรือการใช้ลิสต์ที่เรียงแล้วด้วย Binary search ต้องใช้เวลาในการเข้าถึงหน่วยความจำ $O(\log_2(n))$ และลักษณะการเข้าถึงหน่วยความจำเป็นลักษณะ

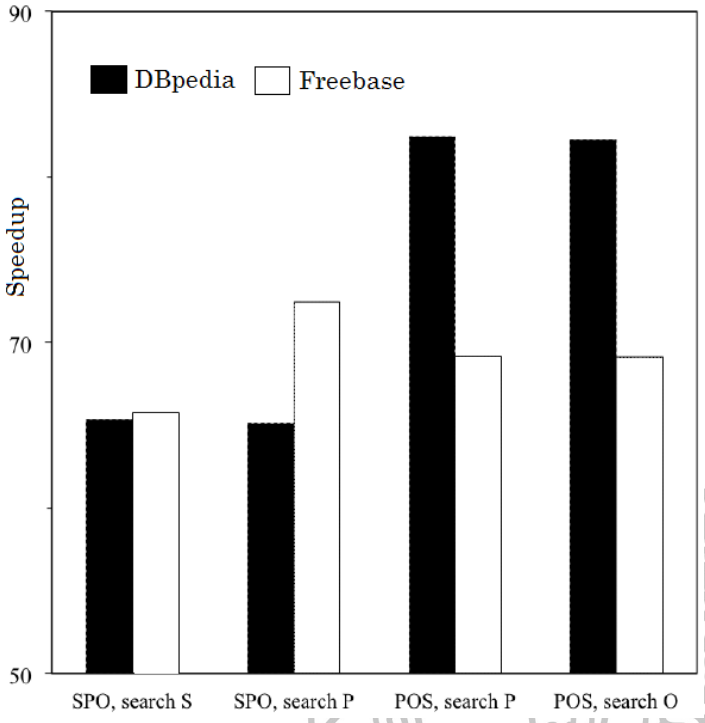
กระโดดที่ไม่ต่อเนื่องกัน ขณะที่ถ้าใช้การเข้าถึงแบบเชิงเส้นจะใช้เวลา O(n) สำหรับข้อมูลขนาด n [137]



รูปที่ 4.8 รูปแบบการเข้าถึงหน่วยความจำของการค้นหาแบบเชิงเส้น แบบไบนารีและแบบบีทรี

การค้นแบบไบนารีไม่ได้โหลดข้อมูลที่อยู่ไกลกันขึ้นมารอไว้ ดังรูปที่ 4.8 ทำให้เวลาอ่านข้อมูลโดนหน่วงออกไปอีก เช่นให้หน่วยประมวลผลกราฟิกส์ GTX 560 Ti มีความหน่วง 468 รอบนาฬิกา [138] ค้นหาข้อมูลขนาด 1,746 KB โดยข้อมูลเป็นเลขจำนวนเต็มขนาด 32 บิต ในกรณีที่ใช้เวลามากที่สุดจะเป็น $\log_2(436.5K) \times 468$ รอบนาฬิกา = 8,784 รอบนาฬิกา ซึ่งเมื่อมีการอ่านข้อมูลหลายรอบจะทำงานเป็นล้านรอบนาฬิกา จึงขัดขวางการอ่านข้อมูลทั้งหมด กรณีเป็นบีทรี คุณลักษณะเด่นอย่าง pivot จะถูกเก็บรวมกลุ่มเป็นโหนดแบบเชิงเส้น ทำให้แคชได้ง่ายกว่า ซึ่งเหมาะกับการทำงานคิวรี่เดียว เทรดเดียวบนหน่วยประมวลผลกลาง จึงเกิดผลลัพธ์ที่เทรดกระจายตัวกันเข้าถึงหน่วยความจำจึงทำให้อ่านค่าต่อเนื่องกันได้ยาก จากตัวอย่างที่คล้ายกับการจัดเรียงรูปแบบเอชดีทีนี้ช่วยอธิบายได้ว่าเหตุใดการประมวลผลเอชดีทีบนหน่วยประมวลผลกราฟิกส์จึงช้ากว่าแบบซีบีเอ็ม

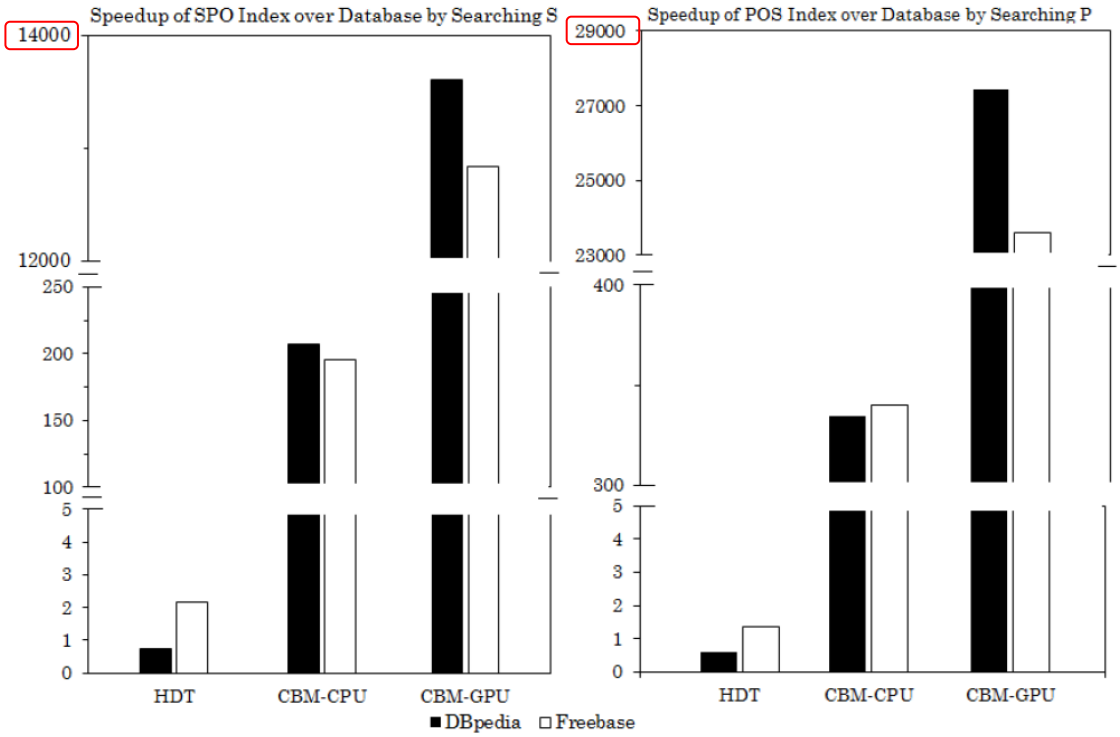
จากอัลกอริทึม 4.2 เพื่อค้นหาตามดัชนี SPO และ POS ผลของการเร่งความเร็วในเซตข้อมูล DBpedia และ Freebase ถูกทดสอบเพื่อเปรียบเทียบกับการประมวลผลแบบลำดับ โดยดัชนี SPO ค้นหาตาม subject หรือ ‘11’ และ predicate หรือ ‘10’ และดัชนี POS ค้นหาตาม predicate หรือ ‘11’ และ object หรือ ‘10’ พบ ว่าสามารถเร่งความเร็วมากกว่าการประมวลผลแบบลำดับได้มากกว่า 60% ดังรูปที่ 4.9



รูปที่ 4.9 ความเร็วในค้นหาข้อมูลจากโครงสร้างซีบีเอ็ม

การเปรียบเทียบการค้นหาระหว่างฐานข้อมูล รูปแบบเอชดีทีและซีบีเอ็ม ทั้งดัชนีแบบ SPO และดัชนีแบบ POS เป็นดังรูปที่ 4.10 พบว่าการค้นหาข้อมูลของ HDT บนหน่วยประมวลผลกลางนั้นเร็วกว่าฐานข้อมูล 0.5-2 เท่า ส่วนซีบีเอ็มนั้นเมื่อค้นหาบนหน่วยประมวลผลกลางแล้วเร็วกว่าระบบฐานข้อมูล 150-375 เท่า สำหรับบนหน่วยประมวลผลกราฟิกส์แล้วซีบีเอ็มเร็วกว่า 13,000-27,000 เท่า ของการค้นหาผ่านฐานข้อมูล (รูปที่ 4.9 - 4.10 โดยใช้เครื่องที่ 2 ทดสอบ)

กล่าวโดยสรุป เมื่อต้องการประมวลผลคิวรี่แบบขนาน โดยไฟล์ทริพเพิลที่ใช้ในการเชื่อมต่อกันบนลิงค์เดต้า ที่เป็นไฟล์ขนาดใหญ่กว่าขนาดของหน่วยความจำบนหน่วยประมวลผลกราฟิกส์ ต้องนำไฟล์ต้นฉบับมาผ่านการแบ่งและแปลงขนาดให้เล็กลง โดยต้องมีโครงสร้างที่สามารถคิวรี่โดยสปาร์เคิลได้อยู่อย่างน้อยในแปดรูปแบบความเป็นไปได้ของ (spo), ..., (???) การแปลงไฟล์ด้วยวิธีเอชดีทีทำได้เพียงดัชนี SPO และแม้ว่าทำให้ไฟล์ต้นฉบับมีขนาดเล็กลงเหลือขนาดเพียง 10% ของไฟล์ต้นฉบับ แต่เมื่อนำไฟล์โครงสร้างต้นไม้เข้าประมวลผลบนหน่วยประมวลผลกราฟิกส์พบว่ามีการเรียงตัวของข้อมูลบนหน่วยความจำแบบไม่ต่อเนื่องทำให้เป็นอุปสรรคต่อการค้นหาข้อมูล



รูปที่ 4.10 การเปรียบเทียบการเร่งความเร็วระหว่างเอชดีทีและซีบีเอ็ม เหนือกว่าฐานข้อมูล

ในประเด็นดังกล่าวผู้วิจัยทำการตรวจซ้ำด้วยการใช้เครื่องมือของบริษัทเอ็นวิเดียและวิชวลสตูดิโอ 2010 ของบริษัทไมโครซอฟต์ที่ชื่อ Parallel Nsight พบว่าเป็นดังตารางที่ 4.10 ขนาดของการส่งผ่านข้อมูลขณะประมวลผลเอชดีทีบนหน่วยความจำคู่ค้าที่ส่งข้อมูลระหว่างสองบรานช์ คือ เคอร์เนลและหน่วยความจำโกลบอลและ เคอร์เนลและหน่วยความจำแชร์ (ทำงานประโยคคิวรี่) ในคอลัมน์ HDT คือการใช้เซตข้อมูลเอชดีทีที่มีขนาดตามตาราง 4.1 ในการประมวลผลบนหน่วยประมวลผลกราฟิกส์ ส่วนคอลัมน์ถัดมาคือต้นทางและปลายทางที่ข้อมูลถูกส่งเข้าและออกตามลำดับดังนี้ ระหว่างเคอร์เนลและหน่วยความจำแชร์ (Kernel-Shared) มีมีการร้องขอข้อมูลมากเป็นจำนวนแสนถึงล้าน เพื่อให้คิวรี่ที่ผู้ใช้ต้องการจับคู่กับเซตข้อมูลได้ คอลัมน์ต่อมาระหว่างเคอร์เนลล์และหน่วยความจำแชร์ (Kernel-ShMem) เช่นกันแต่เป็นขนาดของข้อมูลที่ถูกส่งผ่านหลายรอบ สังเกตว่าในข้อมูลขนาดเล็ก SWDF มีการส่งข้อมูลรวมแล้วขนาดใหญ่กว่าต้นฉบับประมาณ 3.25 เท่า (18.21 เมกะไบต์/5.6 เมกะไบต์) เมื่อเปรียบการส่งผ่านระหว่างเคอร์เนล์และหน่วยความจำโกลบอล (Kernel-Global) พบว่ามีจำนวนการร้องขอที่มากกว่าระหว่างเคอร์เนลล์และหน่วยความจำแชร์ 2.23 เท่า เนื่องจากเป็นการส่งไฟล์ตามขนาดทั้งหมดของไฟล์เอชดีที เมื่อเปิดใช้แคช L1-L2 ตามคุณสมบัติที่ใช้ได้ของ GTX 560Ti รุ่นเฟอร์มิ พบว่าการเปิด L1 มีความหน่วงของสัญญาณนาฬิกามากขึ้นอีก ส่วนการเปิด L2 ทำให้ความหน่วงของสัญญาณนาฬิกาลดลงจาก 468 รอบนาฬิกา [138] (ที่ใช้

วิเคราะห์ตารางที่ 4.8 หน้า 62-63) การส่งผ่านข้อมูลระหว่าง L1-L2 และ L2-Device จึงแทบไม่ต่างจากไฟล์ต้นฉบับในไฟล์ขนาดเล็ก แต่สำหรับไฟล์ขนาดใหญ่กว่าที่แม้แบ่งส่วนเป็น 1/10 และ 1/20 แล้วเมื่อส่งผ่าน Global-L1 Cache กลับมีขนาดโดยรวมมากกว่าหน่วยความจำ 1 กิกะไบต์ ของหน่วยประมวลผลกราฟิกส์ GTX 560 Ti เนื่องจากการส่งข้อมูลทั้งหมดประกอบด้วยข้อมูลเข้าและออกที่ผ่านอุปกรณ์ PCIE ที่มีค่าความเร็วไม่เกิน 6GB/s หากทำการทดสอบข้อมูลขนาด 400 กิกะไบต์นั้นต้องมีอุปสรรคมากขึ้น ด้วยเหตุนี้ผู้วิจัยจึงทำการทดสอบเพิ่มเติมด้วยเครื่องที่สองดังรูป 4.9 – 4.10 ที่มีหน่วยความจำขนาด 12 กิกะไบต์

ตารางที่ 4.10 ขนาดข้อมูลที่ถูกส่งผ่านหน่วยความจำของคู่ด้าวัดด้วย Parallel Nsight จาก VS2010

HDT	Kernel-Shared	Shared-ShMem	Kernel-Global	Global-L1 Cache	L1-L2	L2-Device
SWDF	149.31kReq	18.21MB	334.42kReq	40.84MB	5.11MB	5.11MB
Wordnet	4.61MReq	562.40MB	10.27MReq	1.22GB	157.67MB	157.67MB
Freebase	4.89MReq	597.55MB	10.89MReq	1.30GB	155.98MB	155.98MB
DBpedia	6.32MReq	771.14MB	13.92MReq	1.66GB	209.64MB	209.64MB

ผู้วิจัยจึงนำเสนอ การนำข้อมูลเข้าค้นหาบนหน่วยประมวลผลกราฟิกส์แบบเปรียบเทียบสตริงเพื่อลดเวลาจากขั้นตอนการเตรียมข้อมูลก่อนการประมวลผล ใช้ขนาดเซตข้อมูลตั้งแต่ 1 – 400 กิกะไบต์ ลดเซตคำตอบด้วยการลดขนาดการส่งข้อมูลออก โดยทดสอบทั้งหน่วยความจำแชร์ เท็กเจอร์ และคุณสมบัติใหม่ของคู่ด้าคือหน่วยความจำยูนิฟาย และเพิ่มเครื่องที่ใช้ทดสอบ รวมทั้งทดสอบบนระบบปฏิบัติการวินโดว์ ดังบทที่ 5

บทที่ 5

การประมวลผลแบบขนานด้วยการเปรียบเทียบสตริง

การประมวลคิวรี่ แบบขนานบนหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกส์โดยตรง ไม่ผ่านขั้นตอนแปลงข้อมูลก่อนการประมวลผล โดยใช้การเปรียบเทียบสตริง จากงาน [139] ผู้วิจัยได้พัฒนาโปรแกรม [140] เพื่อแสดงว่าเซตข้อมูลอาร์ดีเอฟ สามารถทำการประมวลคิวรี่แบบขนานด้วยรูปแบบสตริงด้วยหน่วยประมวลผลระดับชั้นบนหน่วยประมวลผลกราฟิกส์และมีการวัดประสิทธิภาพการทำงาน ให้ผู้สนใจสามารถดาวน์โหลดและพัฒนาต่อได้ผ่านทาง Github

ลักษณะเซตข้อมูลอาร์ดีเอฟ

ตารางที่ 5.1 ลักษณะและตัวอย่างรูปแบบไฟล์อาร์ดีเอฟที่ใช้ในงานวิจัย

ลักษณะไฟล์	คำอธิบาย	ตัวอย่าง

TTL [116]	รูปแบบย่อเริ่มด้วยเนมสเปซจัดกลุ่มตาม subject	`@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.`
		`@prefix dc: <http://purl.org/dc/elements/1.1/>.`
		`@prefix base:<http://tourinthailand.org/healthtour#>`
		`@prefix bh: < http://tourinthailand.org/hht/0.1/>.`
		`<http://tourinthailand.org/healthtourism#publish>`
		`dc:title "Hua Hin Health Tourism" ;`
		`bh:editor [`
		`bh:fullname "Chantana Chantrapornchai";`
		`bh:homePage <http://tourinthailand.org/>`
		`] .`

N-Triples [109]	ทริพเพิล	`base:publish dc:title "Hua Hin Health Tourism" .`
		`base:publish bh:editor _:bnode .`
		`_:bnode bh:fullname "Chantana Chantrapornchai" .`
		`_:bnode bh:homePage <http://tourinthailand.org/> .`

N-Quad [112]	ทริพเพิลและชื่อกราฟ	`base:subject base:predicate base:object base:graph1.`
		`base:publish dc:title "HuaHinHealthTourism" base:g2.`
		`base:publish bh:editor _:node base:g2.`
		`_:node bh:fullname "Chantana Chantrapornchai" base:g2.`
		`_:node bh:homePage <http://tourinthailand.org/> base:g2.`

OWL [78]	เอ็กซ์เอ็มแอล/อาร์ดีเอฟ	`<rdf:RDF xmlns="http://xmlns.com/foaf/0.1/"`
		`xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"`
		`xmlns:dc="http://purl.org/dc/terms/"`
		`xml:base="http://tourinthailand.org/healthtour#"`
		`xml:bh="http://tourinthailand.org/hht/0.1/">`
		`<Document`
		`rdf:about="http://tourinthailand.org/healthtourism#publish">`
		`<dc:title xml:lang="en-US">Hua Hin Health Tourism</dc:title>`
		`< base:publish>`
		`< bh:editor rdf:nodeID="1">`
		`<bh:fullname>Chantana Chantrapornchai</bh:fullname >`
		`<bh:homePage><http://tourinthailand.org/></bh:homePage>`
		`</bh:editor>`
		`</base:publish >`
		`</Document>`
		`</rdf:RDF>`

ลักษณะเซตข้อมูลอาร์ดีเอฟที่ใช้ในการวิจัยเรื่องการประมวลผลคิวรี่นี้มีสี่ลักษณะ โดยต่างก็มีพื้นฐานจาก Subject $\xrightarrow{\text{Predicate}}$ Object แต่ละลักษณะขึ้นอยู่กับการใช้งาน รูปแบบทริพเพิลเหมาะกับการนำมาวิเคราะห์คำและใช้ในการคิวรี่ รูปแบบเอ็กซ์เอ็มแอลเหมาะกับการประมวลเมตาเดต้าระหว่างเครื่องเซิร์ฟเวอร์ และการมองรูปแบบกราฟเหมาะกับการทำวิชวลไลเซชันเพื่อให้ผู้ใช้ดูภาพรวมความสัมพันธ์ได้โดยง่าย นำชนิดเซตข้อมูลในตารางที่ 4.1 มาแสดงตัวอย่างได้ดังตารางที่ 5.1 เพื่อแสดงว่าประเภทไฟล์ทุกแบบ คำอธิบาย และตัวอย่าง สามารถนำมาอ่านไฟล์แบบสตริงบนคูด้าได้ทั้งหมด

การทดลองเร่งความเร็วในการค้นหาเซตข้อมูลอาร์ดีเอฟขนาดใหญ่บนหน่วยประมวลผลกราฟิกส์

เครื่องที่ใช้ในการทดสอบเป็นดังต่อไปนี้ โดยแต่ละเครื่องมีการลงโปรแกรมคูด้า 7.5

1. เครื่องเซิร์ฟเวอร์อกาเมมนอน (Agamemnon) จากมหาวิทยาลัยมึนสเตอร์ ประเทศเยอรมนี [139] มีฮาร์ดแวร์ Intel(R) Xeon(R) CPU E5-1620 v2 @ 3.70GHz 64 บิต หน่วยประมวลผลกลาง 4 แกน หน่วยประมวลผลกราฟิกส์จำนวนสองดีไวส์ รุ่น Tesla K20c มีหน่วยความจำโกลบอล 4 กิกะไบต์ มัลติโพรเซสเซอร์จำนวน 13 หน่วยแต่ละหน่วยมี 192 แกนคูด้า แต่ละหน่วยประมวลผลกราฟิกส์ จึงมี 2,496 แกนแต่ละโพรเซส เซอร์มีความถี่ 706MHz ระบบปฏิบัติการลีนุกซ์

2. เครื่องเซิร์ฟเวอร์ TeslaK40 จากบริษัทเอ็นวิเดีย ที่ตั้งคือห้องปฏิบัติการศูนย์วิจัยระบบคอมพิวเตอร์สมรรถนะสูงและเครือข่ายคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ ม. เกษตรศาตร์ [130] ฮาร์ดแวร์ Intel(R) Core(TM) i7 - 5820K CPU@ 3.3GHz 6 แกนและหน่วยความจำ 16 GB ใช้ทดสอบทั้งแบบการ์ด 1 และ 2 ใบ แต่ละใบมีมัลติโพรเซสเซอร์จำนวน 15 หน่วย แกนคูด้า 192 แกนในแต่ละหน่วย ดังนั้นมีแกนคูด้าทั้งหมด 2,880 แกน สัญญาณนาฬิกาสูงสุด 745 MHz และหน่วยความจำโกลบอลขนาด 12 กิกะไบต์ใช้ขนาดของบล็อกในกริด = 480 จำนวนเทรดในบล็อก = 1,024

3. เครื่องคอมพิวเตอร์ส่วนบุคคลระบบปฏิบัติการวินโดว์ หน่วยประมวลผลกลาง i5-2400 ความถี่ 3.10GHz หน่วยความจำ 8 GB PCI-Express 2.0 (16 lanes) แคช 6MB การ์ด Ge-force GTX 660 จำนวน 2 ใบ โดยแต่ละใบมีจำนวนแกน 960 แกน สัญญาณนาฬิกา 980 MHz แบนด์วิธหน่วยความจำ 144.2 GB/s

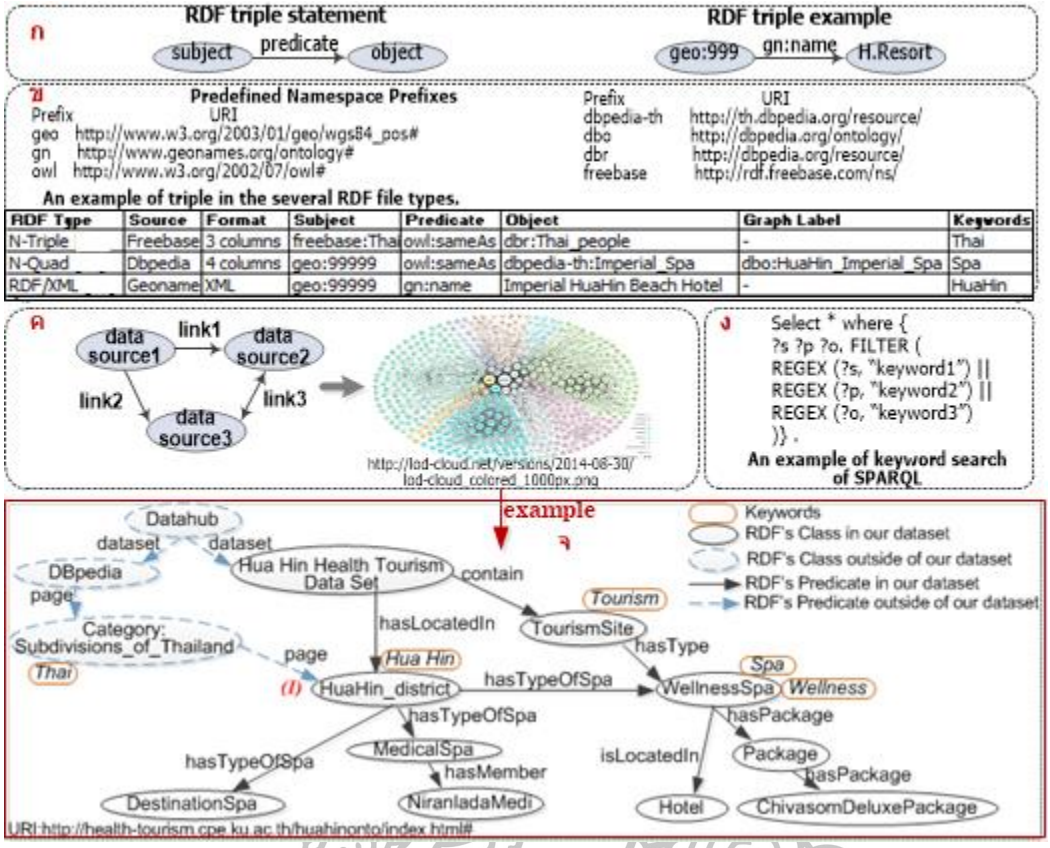
ขั้นตอนการทดลอง

การทดลองที่ 1 ประมวลผลการค้นหาแบบลำดับของข้อมูลอาร์ดีเอฟ ถือเป็นพื้นฐานการเปรียบเทียบกับการค้นหาแบบขนานด้วยวิธีแบบมัลติเทรดและการค้นข้อมูลอาร์ดีเอฟบนหน่วยประมวลผลกราฟิกส์ทั้งแบบที่ใช้หน่วยความจำโกลบอลเพียงอย่างเดียว ดังรูปที่ 5.2 และแบบปรับปรุงที่ใช้หน่วยความจำแชร์ช่วยในการเก็บคำสำคัญทีละคำ และเก็บคำสำคัญที่ใช้ในการค้นหาทั้งหมด [139] โดยชิ้นส่วนไฟล์ (chunk) มีขนาด 1MB เป็นเซตข้อมูล เพื่อนำเข้าไปเป็น Text ในหน่วยประมวลผลกราฟิกส์ ในส่วนของสตริงย่อยคือคำสำคัญที่ใช้ค้นหา สำหรับเครื่องที่ใช้ทดสอบได้แก่ เครื่องเซิร์ฟเวอร์อกาเมมนอน

ตารางที่ 5.2 ขนาดเซตข้อมูลอาร์ดีเอฟที่ใช้ทดสอบเรียงตามขนาดไฟล์

ลำดับ	ขนาดไฟล์	เซตข้อมูล	จำนวนทริพเพิล
1	1.3 GB	1/10Freebase/BTC2012	9,999,999
2	1.6 GB	1/20DBpedia/BTC2012	10,000,000
3	2.6 GB	2/10Freebase/BTC2012	20,000,001
4	3.6 GB	2/20DBpedia/BTC2012	17,449,051
5	3.9 GB	3/10Freebase/BTC2012	30,000,000
6	5.1 GB	4/10Freebase/BTC2012	41,241,557
7	5.3 GB	3/20DBpedia/BTC2012	30,000,007
8	7.5 GB	4/20DBpedia/BTC2012	40,000,006
9	10.0 GB	Geonames-rdf-all.txt	17,028,402
10	45 GB	DBpedia.nq	13,144,657
11	80 GB	Freebase-Weekly80	572,600,000
12	400 GB	Freebase-Weekly400	2,863,000,000

ขนาดของข้อมูลที่ใช้ทดสอบเป็นดังตารางที่ 5.2 ขนาดไฟล์มีตั้งแต่ 1.3 – 400 กิกะไบต์ จำนวนทริพเพิลเริ่มจาก 9,999,999 - 2,863,000,000 ทริพเพิล สำหรับเซตข้อมูลได้นำ DBpedia และ Freebase จาก BTC2012 มาแบ่งเป็น 20 และ 10 ส่วน ตามลำดับ เช่น 1/10Freebase, 2/10Freebase หมายถึงข้อมูล Freebase ที่มีขนาด 1 ส่วนใน 10 ส่วนและขนาด 2 ส่วนใน 10 ส่วนของไฟล์ต้นฉบับ

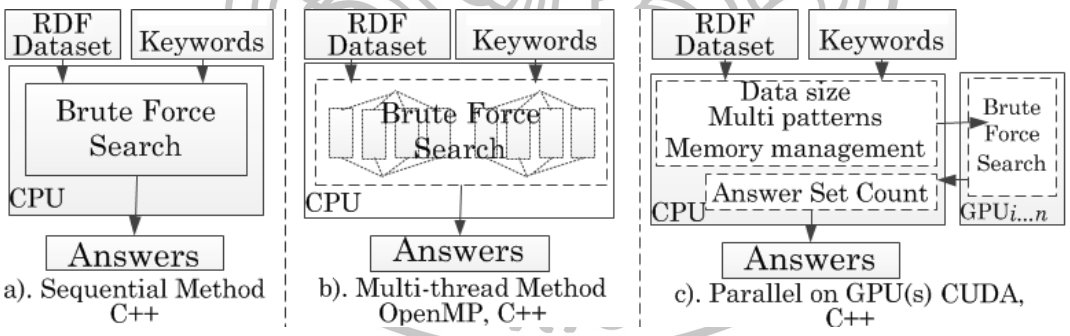


รูปที่ 5.1 การใช้คำสำคัญค้นหาในไฟล์อาร์ดีเอฟ

จากการศึกษาในบทที่ 2 เรื่องการสำรวจพฤติกรรมของผู้ใช้งานอินเตอร์เน็ตในการค้นหาโดยใช้คำสำคัญ ทำให้ผู้วิจัยใช้จำนวนคำสำคัญ 1 – 8 คำ โดยเพิ่มเป็นจำนวนเลขยกกำลังของ 2 คำสำคัญที่ใช้มี 8 คำคือ thai spa huahin tourism health wellness massage budget แบ่งทดสอบ 4 ครั้งย่อยคือ 1, 2, 4, และ 8 คำ

จากรูปที่ 5.2 การใช้คำสำคัญค้นหาในไฟล์อาร์ดีเอฟ อธิบายจากบนลงล่าง ส่วน ก ภาพซ้าย อธิบายว่าประโยคทริพเพิลของอาร์ดีเอฟทั่วไปประกอบด้วย subject predicate และ object โดยมี subject และ object เป็นโหนดที่มี predicate เชื่อมความสัมพันธ์ระหว่างกัน ภาพด้านขวาคือตัวอย่างเช่น พิกัดทางภูมิศาสตร์ชื่อโหนด geo:999 เป็น subject ของทริพเพิล ส่วน gn:name เป็น predicate เชื่อมความสัมพันธ์กับ object ชื่อ H:resort ดังนั้นทั้งทริพเพิลนี้แสดงความหมายว่า geo:999 คือพิกัดที่ชื่อว่า H:resort ส่วน ข เป็นรูปแสดงภายในไฟล์อาร์ดีเอฟที่ประกอบด้วยการประกาศเนมสเปซที่ประกอบด้วยสองส่วนคือพรีฟิกซ์และ URI เต็มของพรีฟิกซ์นั้น และตัวอย่างของทริพเพิลในตารางทั้งแบบ N-Triple N-Quad และ RDF/XML ที่มีชื่อของเซตข้อมูล ลักษณะภายในไฟล์แบบ สามคอลัมน์ สี่คอลัมน์และเอ็กเอ็มแอล ตามลำดับ จากนั้นแสดงตัวอย่างชื่อของ subject predicate object โดยแบบ N-Quad มีช่องชื่อกราฟ เพิ่มขึ้นมา ที่กำหนดการใช้งาน “พรีฟิกซ์:ชื่อ

เฉพาะ” เพื่อย่อ URI ของพรีฟิกซ์ของทั้งไฟล์และตัวอย่างคำสำคัญที่ใช้ค้นหาในคอลัมน์สุดท้าย ส่วน ค แสดงถึงว่าแต่ละไฟล์ของอาร์ดีเอฟสามารถเชื่อมถึงกันได้ จากการเผยแพร่ด้วยเทคโนโลยีลิงค์เดต้า จนกลุ่มวิจัยนี้ มีขนาดของเซตข้อมูลเปิดที่ใหญ่มากขึ้นเรื่อย ๆ โดยระหว่างเซตหรือไฟล์ข้อมูลจะมีความสัมพันธ์เฉพาะเชื่อมต่อกัน ส่วน ง แสดงถึงประโยค สปาร์เคิล (SPARQL) ที่เป็นภาษาคิวรีของไฟล์อาร์ดีเอฟ ในกรณีที่เป็นการค้นหาด้วยคำสำคัญ จะใช้คำสั่ง FILTER และฟังก์ชัน REGEX (Regular Expression) มาช่วยกำหนดการค้นหาเฉพาะคำสำคัญ ในตำแหน่ง subject (?s) ตำแหน่ง predicate (?p) และตำแหน่ง object (?o) ส่วน จ เป็นส่วนขยายของ ค โดยแสดงว่า Datahub ที่เก็บเซตข้อมูลจำนวนมากประกอบด้วยเซตข้อมูล DBpedia และ Hua Hin Health Tourism Data Set ซึ่งใน DBpedia มีเพจชื่อ Subdivisions of Thailand ที่ใช้คำสำคัญว่า ‘thai’ ในการค้นหาได้จากรายละเอียดในเพจ ที่เก็บไว้ใน subject predicate หรือ object ในเพจดังกล่าวยังเชื่อมต่อกับ Hua Hin Health Tourism Data Set ด้วยเพจ (page) ชื่อ Hua Hin district ซึ่งเชื่อมด้วยความสัมพันธ์ hasLocatedIn ภายในเซตข้อมูล Hua Hin Health Tourism ซึ่งในส่วนนี้สามารถใช้คำว่า ‘Hua Hin’ ค้นหาได้ นอกจากนั้นยังแสดงว่าคอนเซปต์อื่น ๆ ได้แก่ TourismSite ที่ใช้คำว่า ‘Tourism’ ค้นหาได้ หรือคอนเซปต์ WellnessSpa ที่ใช้คำว่า ‘Spa’ หรือ ‘Wellness’ ค้นหาได้ เป็นต้น



รูปที่ 5.2 อัลกอริทึมที่ใช้ค้นหา (a) การใช้หน่วยประมวลผลกลางแบบลำดับ (b) การค้นหาแบบมัลติเทรดด้วย OpenMP (c) การค้นหาแบบมัลติเทรดด้วยการประมวลผลบนหน่วยประมวลผลกราฟิกส์

รูปที่ 5.2 แสดงภาพ (a) การทำงานของอัลกอริทึมแบบบรูซฟอร์ซที่ใช้ค้นหาดังอัลกอริทึมที่ 5.1 (b) การใช้หน่วยประมวลผลกลางแบบลำดับดังอัลกอริทึมที่ 5.2 (c) การค้นหาแบบมัลติเทรดด้วย OpenMP การค้นหาแบบมัลติเทรดด้วยการประมวลผลบนหน่วยประมวลผลกราฟิกส์

อัลกอริทึม 5.1 อัลกอริทึมการเปรียบเทียบสตริงแบบลำดับ (BFSEQ)

Sequential Algorithm (BFSEQ)

```
Input:  dataArray and keywordArray
Output: Resulting positions in positionArray
1 while not EOF do
2  Read the RDF data file chunk in dataArray.
3  /*compare the data to keyword*/
4  for i =0; i<n; i++ do
5    j=0
6    while i<n and j<m do
7       if dataArray[i+j]!= keyword[j] then
8          break
9       if j == m then
10         positionArray[i] = true
```

การทำงานของอัลกอริทึมที่ 5.1 BFSEQ นี้คือค้นหาคำสำคัญแบบลำดับ ในส่วนของข้อมูลจะอ่าน ชิ้นส่วนข้อมูลขนาด 1 เมกะไบต์ เข้าไปวนลูปซ้ำจนกว่าข้อมูลจะหมดในขณะเดียวกันก็เปรียบเทียบสตริงไปพร้อมกันทุกเทรด

การทำงานแบบขนานของ OpenMP เป็นการปรับแบบบรูซฟอร์ซมาใช้กับการค้นหาแบบมัลติเทรดแบบ OpenMP ด้วยการเติม *#preagma omp parallel for private (j)* ลงในอัลกอริทึมที่ 5.1 ตั้งชื่อว่า BFOMP

อัลกอริทึม 5.2 การค้นหาด้วยวิธีบรูซฟอร์ซแบบขนานบนหน่วยประมวลผลกราฟิกส์

Parallel search on GPU (BFG)

```
Input: dataArray, keywordArray
Output: positionArray
1. Allocate device memory for dataArray, keywordArray, positionArray
2. While not EOF do
3.   Read the RDF datafile chunk in dataArray
4.   Copy dataArray to GPU
5.   for each keyword in keywordArray do
6.     positionArray (initialized to false and copy keyword to GPU
7.     Launch the following kernel code for threads for each keyword
8.     Kernel Code Begin (Search)
9.      j=0
10.     for i = threadIdx.dx; i<n; i+ = blockDim * gridDim do
11.         if dataArray[i] == keyword[0] then
12.            j=1
13.            while i + j < n and j < m do
14.              if j == m then
15.                 break
16.         if j == m then
17.            positionArray[1]= true
18.    Kernel Code End (End)
19.    Copy positionArray back to CPU and sum total positions found.
20. Free the memory.
```

การสร้างโปรแกรมบนคูด้า โดยการนำอัลกอริทึม BFSEQ มาปรับเป็นอัลกอริทึมที่ 5.2 คือ BFG เพื่อเป็นการลดโอเวอร์เฮดของการคัดลอกข้อมูลเข้าและออกจากหน่วยความจำโกลบอลของหน่วยประมวลผลกราฟิกส์ เคอร์เนลของหน่วยประมวลผลกราฟิกส์จำทำงานวนซ้ำภายในเทรดของหน่วยประมวลผลกราฟิกส์ให้ทำงานคุ้มค่าที่สุด รวมทั้งเรียงตามหน่วยความจำไปเรื่อย ๆ เพื่อให้เกิดความต่อเนื่อง จากบรรทัดที่ 11 แต่ละเทรดทำการเปรียบเทียบสตริงจากตำแหน่งที่ 1 เพื่อทำการค้นหาหลายคำสำคัญจะเริ่มวนซ้ำแต่ละคำสำคัญในบรรทัดที่ 5 คัดลอกคำสำคัญลงสู่หน่วยความจำของหน่วยประมวลผลกราฟิกส์ในบรรทัดที่ 6 และเริ่มทำงานบนเทรดของหน่วยประมวลผลกราฟิกส์ในบรรทัดที่ 7

การประยุกต์ใช้หน่วยความจำแชร์

การปรับปรุงอัลกอริทึมที่ 5.2 ด้วยการใช้หน่วยความจำแชร์บนหน่วยประมวลผลกราฟิกส์ ในเวอร์ชันนี้เรียกว่า BFGSH ดังอัลกอริทึมที่ 5.3 เนื่องจากหน่วยความจำแชร์มีขนาดที่จำกัดมากเมื่อเทียบกับหน่วยความจำโกลบอลแต่อยู่ใกล้เทรดที่ทำงานมากที่สุด จึงช่วยให้การประมวลผลเป็นไปอย่างรวดเร็วมากกว่า ด้วยความที่หน่วยความจำแชร์มีขนาดเล็กมากนี้จึงใส่เซตข้อมูลทั้งหมดไม่ได้ แต่สามารถใส่คำสำคัญไปทีละคำ ในงานวิจัยนี้ใส่ได้ไม่เกินขนาดทั้งหมด 48 KB โดยกำหนดให้ขนาดคำสำคัญเท่ากับเลขจำนวนเต็มขนาด 16 ไบต์แล้วกำหนดให้ 16 เทรดแรกคัดลอกคำสำคัญลงสู่หน่วยความจำแชร์ ดังตัวแปร *s_keyword* ในบรร-ทัดที่ 10 และ 11 ของอัลกอริทึมที่ 5.3 ดังนั้นจำนวนของขนาดข้อมูลที่ส่งผ่านในหน่วยไบต์เป็นดังสมการที่ 1

อัลกอริทึม 5.3 การค้นหาด้วยวิธีบรูซฟอร์ซบนหน่วยประมวลผลกราฟิกส์ร่วมกับหน่วยความจำแชร์ (BFGSH)

Brute-force search on a GPU with shared memory (BFGSH)

```
Input: dataArray, keywordArray
Output: positionArray
1. Allocate device memory for dataArray, keywordArray, positionArray
2. while not EOF do
3.    Read the RDF data file chunk in dataArray.
4.    Copy dataArray to GPU
5.    for each keyword in keywordArray do
6.       positionArray (initialized to false) and copy keyword to GPU
7.       Launch the following Kernel code for threads for each keyword
8.       Kernel Code Begin (Search)
9.         j=0
10.        Allocate shared memory of size[int], call s_keyword.
11.        if thread_index < 16 then
12.           s_keyword[thread_index] = keyword[thread_index]
13.        syncthreads
14.        /* The same as Lines 8-17 of Algorithm 2 except that we use
              s_keyword for comparison*/
15.    Kernel Code End (End Search)
16.    Copy positionArray back to CPU and sum total positions found.
17. Free the memory.
```

การปรับปรุงการค้นหาด้วยการใช้คำสำคัญหลายคำ ในกรณีนี้ทดลองใส่คำสำคัญทั้งหมดลงหน่วยความจำแชร์ของหน่วยประมวลผลกราฟิกส์ ดังบรรทัดที่ 4 กำหนด *dataArray* ภายในเทรดของหน่วยประมวลผลกราฟิกส์ ชื่อของอัลกอริทึม 5.4 คือการค้นหาด้วยวิธีบรูซฟอร์ซบนหน่วยประมวลผลกราฟิกส์ร่วมกับหน่วยความจำแชร์ สำหรับคำสำคัญหลายคำ (BFGSHM) จากโค้ดตัวอย่างกำหนดจำนวนของคำสำคัญให้เป็น 8 คำ แต่ละคำมีขนาดเท่ากับเลขจำนวนเต็มขนาด 16 ไบต์โดยขนาดของคำสำคัญทั้งหมดถูกคัดลอกลงหน่วยประมวลผลกราฟิกส์ภายในครั้งเดียวดังบรรทัดที่ 10 – 12 มีตัวแปรที่ใช้ทำเครื่องหมายคือ *match* เพื่อนับจำนวนของคำสำคัญที่พบใน *positionArray* ในบรรทัดที่ 13

การวิเคราะห์ข้อมูลที่ส่งระหว่างหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกส์ถือเป็นการคำนวณโอเวอร์เฮดในอัลกอริทึมที่ทำงานบนหน่วยประมวลผลกราฟิกส์ แนวคิดของสำคัญต้องนำข้อมูลและคำสำคัญที่เป็นส่วนประกอบสำคัญ เข้าไปทำการค้นหาบนหน่วยประมวลผลกราฟิกส์โดยต้องนำข้อมูลเข้าไปทำงานให้นานที่สุด เพื่อความคุ้มค่า ชิ้นส่วนที่ใหญ่ที่สุด และจำนวนครั้งในการขนย้ายต้องน้อยที่สุด

อัลกอริทึม 5.4 การค้นหาด้วยวิธีบรูซฟอร์ซบนหน่วยประมวลผลกราฟิกส์ร่วมกับหน่วยความจำแชร์ สำหรับคำสำคัญหลายคำ (BFGSHM)

Brute-force search on a GPU with shred memory for multiple keywords (BFGSM)

```
Input: dataArray, keywordArray
Output: positionArray
1. Allocate device memory for dataArray, keywordArray, positionArray.
2. Copy keywordArray to GPU.
3. while not EOF do
4.    Read the RDF data file chunk in dataArray.
5.    Copy dataArray, and positionArray to GPU
6.    Launch the following kernel code for threads
7.    Kernel Code Begin (Search)
8.    j = 0
9.    Allocate shared memory of int [16*8] called s_keyword
10.   If (thread_index) < 16*8) then
11.       s_keyword[thread_index] = keywordArray[thread_index]
12.   syncthreads
13.   match =  -total_keyword
14.   for p = 0 to total_keyword do
15.     /*replace the global memory pointer with the shared memory pointer.*/
16.     keyword = s_keyword + 16 * p
17.    /* Here is the same as Lines 9-17, from Algorithm 2 except that
18.    when j==m, we update match++ and set positionArray[i] =
19.    match */
20.   Kernel Code End (End Search)
21.   Copy positionArray back to CPU and sum total positions found.
22. Free the memory.
```

กำหนดให้ N_{base} เป็นจำนวนการส่งข้อมูลทั้งหมดระหว่างหน่วยความจำของหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกส์ เทียบจากอัลกอริทึม 5.2 – 5.4 จะประกอบด้วยการย้ายเซตข้อมูลอาร์ดีเอฟทั้งหมดด้วยตัวแปร *dataArray* การย้ายคำสำคัญทั้งหมดด้วย *keywordArray* และการย้ายคำตอบกลับหน่วยประมวลผลกลาง *positionArray* ให้ *n* เป็นความยาวของข้อมูลอาร์ดีเอฟทั้งหมด *c* เป็นขนาดชิ้นส่วนของหน่วยประมวลผลกลาง *k* เป็นจำนวนของคำสำคัญ ขอบเขตล่างของขนาดข้อมูลที่ถูกย้ายคือ

$$N_{base} = \left\lceil \frac{n}{c} \right\rceil \times |c| + \sum_{i=1}^{P} \left(|k_i| + \left\lceil \frac{n}{c} \right\rceil \times |c| \right) \quad (1)$$

จากสมการที่ 1 เทอมแรกขึ้นอยู่กับจำนวนครั้งของการคัดลอกข้อมูลจากหน่วยประมวลผลกลางไปยังหน่วยประมวลผลกราฟิกส์ $\left\lceil \frac{n}{c} \right\rceil$ และขนาดของข้อมูลแต่ละชิ้น $|c|$ ในเทอมที่สองผลรวมของการย้ายคำสำคัญทั้งหมดคือ $|k_i|$ และ $\left\lceil \frac{n}{c} \right\rceil \times |c|$ เป็นขนาดของการคัดลอกผลลัพธ์ของการค้นหาแต่ละคำสำคัญกลับ โดยมีข้อสังเกตคือต้องจองพื้นที่ของอะเรย์ เพื่อส่งข้อมูลกลับด้วย ตัวอย่างเช่น จากระบบที่นักวิจัยทดสอบจองหน่วยความจำโกลบอลขนาด 1.8 กิกะไบต์ สำหรับ *dataArray* และ *positionArray* จะถูกแบ่งเป็นพื้นที่ของข้อมูลเข้า 0.9 กิกะไบต์ ในอัลกอริทึม 5.2- 5.3 การค้นหาข้อมูลอาร์ดีเอฟแต่ละชิ้นส่วนนั้นการย้ายข้อมูลส่วนนี้จะมีข้อมูลคำสำคัญติดไปด้วย

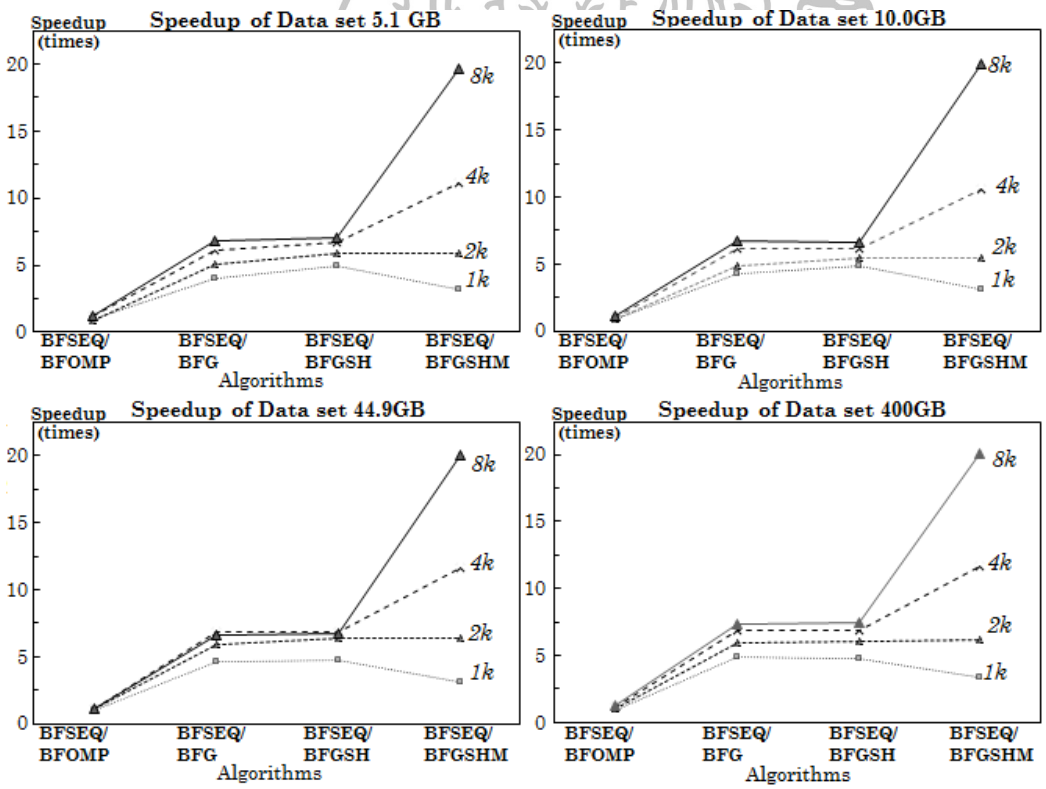
สำหรับอัลกอริทึมที่ 5.4 เมื่อคัดลอก *keywordArray* ลงหน่วยประมวลผลกราฟิกส์ดังบรรทัดที่ 2 และคัดลอกผลลัพธ์ดังบรรทัดที่ 19 แล้ว จำนวนการย้ายข้อมูลทั้งหมดลดลงจากสมการที่ 1 เมื่อกำหนดให้ N_{opt} เป็นจำนวนครั้งทั้งหมดของการคัดลอกข้อมูลระหว่างหน่วยประมวลผลกราฟิกส์และหน่วยประมวลผลกลางของคำสำคัญจำนวน *p* คำ พิจารณาสมการที่ 2 จากผลการทดลองจะพบว่าเวลาที่ใช้คัดลอกข้อมูลลดลงจากอัลกอริทึม 5.2 และ 5.3

$$N_{opt} = \left\lceil \frac{n}{c} \right\rceil \times |c| + \sum_{i=1}^{p} |k_i| + \left\lceil \frac{n}{c} \right\rceil \times c \times p \quad (2)$$

ผลลัพธ์จากทั้ง 2 สมการนี้เห็นได้ชัดเจนจากรูปที่ 5.6 เวลาที่ใช้ในการค้นหาและการย้ายข้อมูลขณะประมวลผลบนหน่วยประมวลผลกราฟิกส์ของข้อมูลขนาด 80 กิกะไบต์และ 400 กิกะไบต์

การทดลองนี้ใช้เครื่องเซิร์ฟเวอร์กาเมมนอน เซตข้อมูลที่ใช้คือ 4/10Freebase/BTC2012 ขนาด 5.1 กิกะไบต์ จำนวน 41,241,557 ทริพเพิล ข้อมูล Geonames-rdf-all.txt ขนาด 10.0 กิกะไบต์ จำนวน 17,028,402 ทริพเพิล ข้อมูล DBpedia.nq จำนวน 45 กิกะไบต์ 13,144,657 ทริพเพิล เซตข้อมูล Freebase-Weekly80 ขนาด 80 กิกะไบต์ จำนวน 572,600,000 ทริพเพิล และ Freebase-Weekly400 ขนาด 400 กิกะไบต์ จำนวน 2,863,000,000 ทริพเพิล

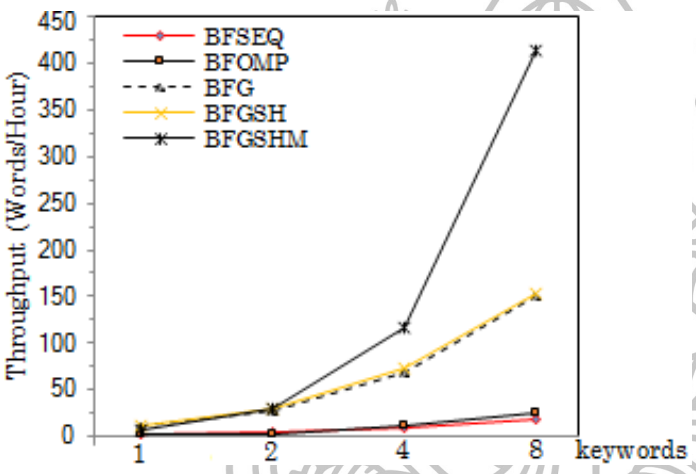
ในการทดลองนี้ใช้คำสำคัญจากออนโทโลยีการท่องเที่ยวเชิงสุขภาพ [106] จำนวนคีย์เวิร์ดอยู่ในความยาว 48 อักษร 1 คำคือ “thai" 2 คำ คือ “thai spa" 4 คำคือ “thai spa huahin tourism" และ 8 คำคือ “thai spa huahin tourism health wellness massage budget" โค้ดของโปรแกรมเผยแพร่ที่ https://github.com/chidcha/Parallel_Search



รูปที่ 5.3 การเร่งความเร็วของเซตข้อมูลขนาด 5.1 GB, 10 GB, 45 GB และ 400 GB

สำหรับการเร่งความเร็วแสดงดังรูปที่ 5.3 การเร่งความเร็วเปรียบเทียบจากการค้นหาด้วยคำสำคัญ 1k คือ ขนาด 1 คำสำคัญ, 2k คือ ขนาด 2 คำสำคัญ, 4 k คือ ขนาด 4 คำสำคัญ, และ 8k คือ ขนาด 8 คำสำคัญ กับขนาดข้อมูลรูปบนซ้ายถึงล่างขวาได้แก่ 5.1 กิกะไบต์ 10 กิกะไบต์ 44.9

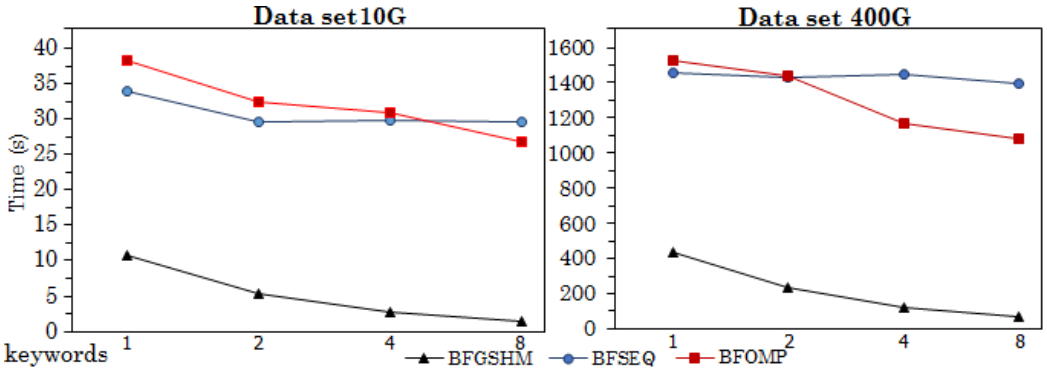
กิกะไบต์ และ 400 กิกะไบต์ ตามลำดับ เมื่อใช้ขนาดชิ้นส่วนข้อมูลที่ 1 เมกะไบต์ สังเกตได้ว่าอัลกอริทึม สามารถเร่งความเร็วได้มากกว่า BFSEQ เมื่อนำไปทำการค้นหาด้วยหน่วยประมวลผลกราฟิกส์พบว่าทั้ง BFG และ BFGSH ต่างก็เร่งความเร็วได้มากกว่า BFSEQ แต่ทั้ง BFG และ BFGSH กลับมีความเร็วที่ต่างกันเล็กน้อย เนื่องจากในการทดลองนี้ใช้ 128 เทรดเท่ากัน เมื่อทดสอบกับ BFGSHM พบว่า เมื่อใช้หน่วยประมวลผลแชร์ช่วยเก็บคำสำคัญหลายคำทำให้ช่วยเร่งความเร็วในการค้นหามากกว่าหน่วยประมวลผลกลาง BFSEQ อยู่ 20.48 เท่าสำหรับข้อมูลขนาด 400 กิกะไบต์ เนื่องจากข้อมูลที่มีชิ้นเล็กที่สุด 1 เมกะไบต์ ในการทดลองนี้ มีความถี่ของการส่งข้อมูลเข้าออกเป็นจำนวนมาก



รูปที่ 5.4 ธรูพุตการค้นหาจำนวนคำสำคัญต่อชั่วโมงของอัลกอริทึม 5 แบบ

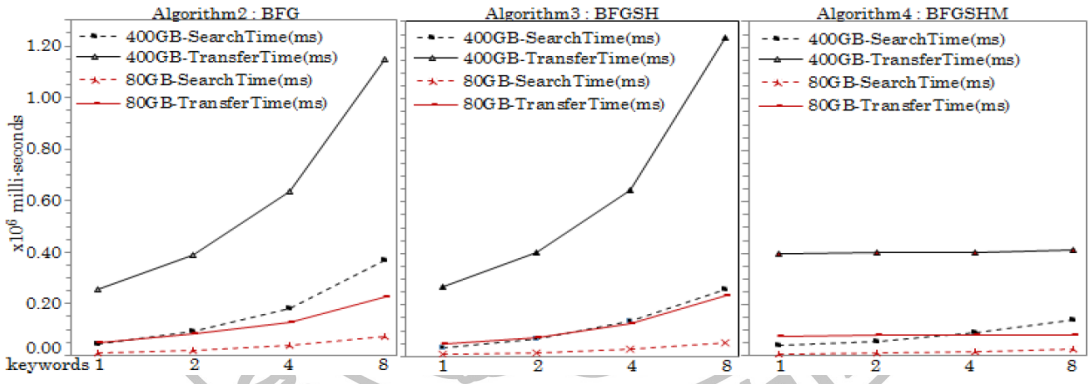
สำหรับการวัดธรูพุตตามจำนวนการค้นคำสำคัญต่อชั่วโมง พบว่าเป็นดังรูปที่ 5.4 นั้นคือเส้นกราฟบนสุดแสดงการทำงานของอัลกอริทึมที่ 5.4 BFGSHM ที่มีการใช้หน่วยประมวลผลแชร์ช่วยเก็บคำสำคัญแบบหลายคำเพื่อใช้ในการค้นหาบนหน่วยประมวลผลกราฟิกส์สามารถค้นคำได้ 414 คำสำคัญต่อชั่วโมงเมื่อใช้คำสำคัญแบบ 8 คำค้น ในขณะที่บนหน่วยประมวลผลกลาง BFSEQ ค้นด้วย 8 คำค้นเช่นกันมีธรูพุตที่ 20 คำต่อชั่วโมง บนเซตข้อมูล 400 กิกะไบต์

เมื่อลงรายละเอียดในเวลาของการประมวลผลบนหน่วยประมวลผลกราฟิกส์ที่ใช้ในการค้นหาคำสำคัญ ผู้วิจัยได้ทำการทดสอบด้วยข้อมูลขนาด 80 กิกะไบต์และ 400 กิกะไบต์ ข้อมูลทั้งหมดจะผ่านการย้ายข้อมูลเข้าและออก เพื่อทำการค้นหา เมื่อแยกรายละเอียดของเวลาที่ใช้เคลื่อนย้ายข้อมูลระหว่างโฮสต์และดีไวส์ จะได้ว่า เวลาการเคลื่อนย้ายข้อมูลประกอบด้วยผลรวมของเวลาที่ใช้เคลื่อนย้ายข้อมูล เวลาที่ใช้เคลื่อนย้ายคำสำคัญ และเวลาที่ใช้ในการเคลื่อนย้ายคำตอบ



รูปที่ 5.5 เวลาการค้นหาโดยเฉลี่ย

เมื่อพิจารณาเวลาการค้น หาโดยเฉลี่ยดังรูปที่ 5.5 แสดงเวลาโดยเฉลี่ยเมื่อใช้ค้นหาคำสำคัญ 1, 2, 4, และ 8 คำ สำหรับเซตข้อมูลขนาด 10 กิกะไบต์ และ 400 กิกะไบต์ ระหว่าง BGGSHM BFSEQ และ BFOMP สรุปได้ว่า BFGSHM สามารถค้นคำสำคัญทั้ง 8 คำได้ด้วยเวลาเฉลี่ยน้อยที่สุดจากเซตข้อมูลทั้งสองขนาด สังเกตขนาดหน่วยของเซตข้อมูลทั้งสองพบว่ามากขึ้นตามขนาดของเซตข้อมูล

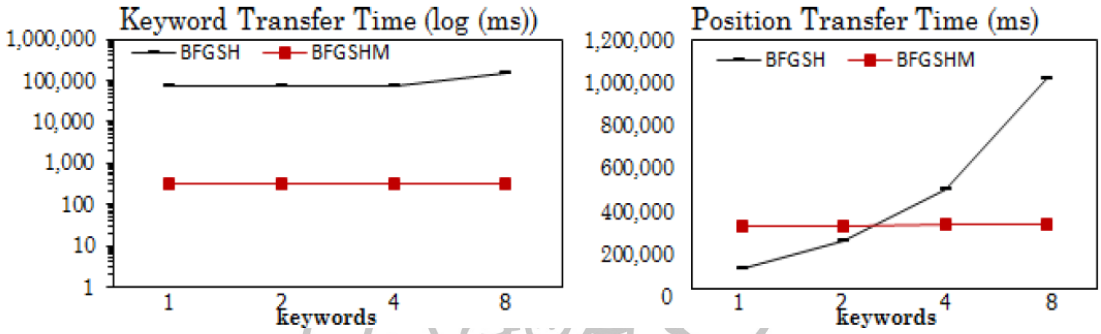


รูปที่ 5.6 เวลาที่ใช้ในการค้นหาและการย้ายข้อมูลขณะประมวลผลบนหน่วยประมวลผลกราฟิกส์ของข้อมูลขนาด 80 กิกะไบต์และ 400 กิกะไบต์

รูปที่ 5.6 แสดงเวลาที่ใช้ค้นหาและเวลาที่ใช้ในการย้ายข้อมูลในเซตข้อมูล 2 ขนาด คือ 80 กิกะไบต์ และ 400 กิกะไบต์ เส้นประแสดงถึงเวลาที่ใช้ในการค้นหา และเส้นทึบแสดงถึงเวลาที่ใช้ในการเคลื่อนย้ายข้อมูล สำหรับ BFG และ BFGSH ตามอัลกอริทึม 5.2 และ 5.3 ตามลำดับ พบว่าเวลาที่ใช้ในการย้ายข้อมูลสำหรับการค้นหาคำสำคัญ 1 คำ 2 คำ 4 คำและ 8 คำ มีการเพิ่มขึ้นอย่างรวดเร็ว ซึ่งเป็นไปตามสมการที่ 1 ดังที่ผู้วิจัยได้ทำการคำนวณไว้จากทั้งสองอัลกอริทึม ขณะที่เวลาที่ใช้ในการเคลื่อนย้ายข้อมูลของ BFGSHM ใช้เวลาเกือบคงที่สำหรับค้นหา 1-8 คำ ในขณะที่เส้นประที่

แสดงการค้นหาในเซตข้อมูลขนาด 400 กิกะไบต์เป็นขนาดเดียวที่มีการเติบโตบ้าง ส่วน 80 กิกะไบต์มีการเติบโตเล็กน้อย ซึ่งเป็นไปตามสมการที่ 2 ที่ได้แสดงไว้

จาก $T_{\text{transfer}} = T_{data} + T_{keywords} + T_{answer}$ คือเวลาที่ใช้ในการเคลื่อนย้ายข้อมูลที่สัมพันธ์กับตัวแปรในอัลกอริทึมที่ใช้ประมวลผลการค้นหาบนหน่วยประมวลผลกราฟิกส์ นั่นคือ เวลาที่ใช้เคลื่อนย้ายข้อมูล (T_{data}) สัมพันธ์กับ *dataArray* เวลาที่ใช้เคลื่อนย้ายคำสำคัญ ($T_{keywords}$) สัมพันธ์กับ *keywordArray* และเวลาที่ใช้ในการเคลื่อนย้ายคำตอบกลับหน่วยประมวลผลกลาง (T_{answer}) สัมพันธ์กับ *positionArray*



รูปที่ 5.7 แสดงเวลาที่ใช้ในการเคลื่อนย้ายคำสำคัญ (ซ้าย) และย้ายคำตอบ (ขวา)

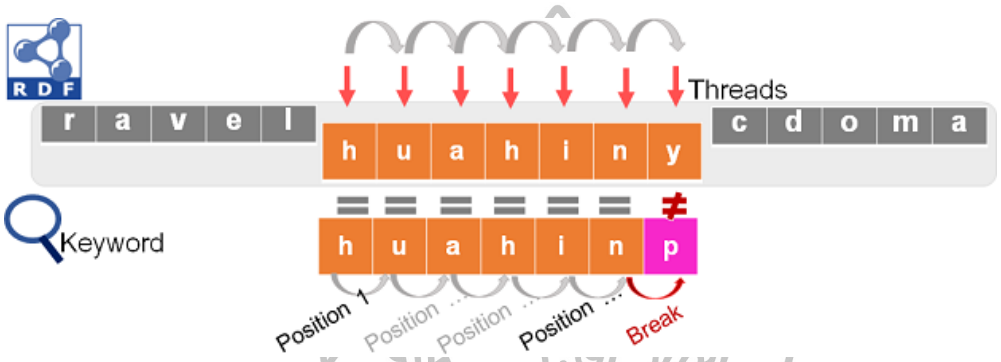
รูปที่ 5.7 แสดงเวลาที่ใช้ในการเคลื่อนย้ายคำสำคัญ หน่วยลอการิทึม (มิลลิวินาที) (ซ้าย) และเวลาที่ใช้ย้ายคำตอบ (ขวา) หน่วยมิลลิวินาที ที่สัมพันธ์กับตัวแปรในอัลกอริทึม BFGSH และ BFGSHM ที่ใช้ประมวลผลการค้นหาคำสำคัญ 1 คำ 2 คำ 4 คำและ 8 คำบนหน่วยประมวลผลกราฟิกส์ สำหรับ BFGSHM (เส้นทึบมีสี่เหลี่ยมคั่น) สังเกตได้ว่าเวลาที่ใช้เคลื่อนย้ายค่อนข้างคงที่ เนื่องจากผู้วิจัยได้ใส่คำสำคัญทั้งหมดเพียงครั้งเดียวลงในหน่วยความจำแชร์ เมื่อพิจารณาการคืนค่าของคำตอบที่ค้นพบก็พบว่าเวลาที่ใช้ก็ค่อนข้างคงที่เช่นกัน สำหรับทุกจำนวนคำสำคัญ ขณะที่ BFGSH นั้นใช้เวลาในการเคลื่อนย้ายคำตอบเติบโตแบบเชิงเส้นตามจำนวนคำสำคัญ ดังที่คำนวณไว้ในสมการที่ 1

สรุปได้ว่าการนำข้อมูลเข้าไปค้นหาโดยตรงในหน่วยประมวลผลกราฟิกส์ โดยแบ่งข้อมูลออกเป็นชิ้นส่วนขนาดเล็ก 1 เมกะไบต์ ข้อดีคือช่วยลดเวลาที่ใช้ในการแปลงข้อมูลก่อนการประมวลผลดังบทที่ 4 แต่ข้อเสียคือทำให้การเร่งความเร็วในการค้นหาข้อมูลบนหน่วยประมวลผลกราฟิกส์ลดลงอย่างมากจากหลักหมื่นเท่า เหลือหลักสิบเท่าดังนั้นผู้วิจัยจึงได้ปรับปรุงการค้นหาโดยตรงนี้โดยเพิ่มขนาดสิ้นส่วนข้อมูลให้มีหลายขนาด และประยุกต์การใช้งานหน่วยความจำหลายแบบในหน่วยประมวลผลกราฟิกส์ในหัวข้อถัดไป *การปรับปรุงการค้นหาแบบขนานของข้อมูลอาร์ดีเอฟในงานต่าง ๆ*

การปรับปรุงการค้นหาแบบขนานของข้อมูลอาร์ดีเอฟในงานต่าง ๆ

การแบ่งข้อมูลขนาดใหญ่ออกเป็นส่วนย่อยหลายขนาด

การค้นหาด้วยวิธีบรูทฟอร์ซ (Brute-force) แบบขนาน ได้รับการพัฒนาให้ทำการค้นหาบนแกนของหน่วยประมวลผลกราฟิกส์ได้ ข้อมูลเข้าได้แก่เซตข้อมูลของออนโทโลยีในรูปแบบอาร์ดีเอฟด้วยคำสำคัญตามที่ผู้ใช้กำหนด ผลลัพธ์คือจำนวนของคำตอบที่ได้จากการค้นหาดังรูปที่ 5.8



รูปที่ 5.8 การค้นหาบรูทฟอร์ซแบบขนาน

ปัญหาของโปรแกรมวิธีบรูทฟอร์ซแบบลำดับคือการทำงานอย่างล่าช้า เนื่องจากข้อมูลเข้าแบบอักขระที่มีขนาดใหญ่และทำการค้นหาเทียบกับคำสำคัญทีละอักขระแบบลำดับไปเรื่อย ๆ จนหมดชุดข้อมูล สำหรับการทำงานแบบขนานที่โปรแกรมด้วยวิธีมัลติเทรด สามารถแสดงได้ด้วยรูปที่ 5.8 ทุกเทรดจะทำการค้นหาอักขระและเทียบกับคำสำคัญพร้อม ๆ กัน วิธีนี้สามารถเร่งการทำงานโดยใช้การทำงานแบบขนานได้ระดับหนึ่ง แต่ความเร็วที่ได้จะไม่เกินจำนวนเท่าของจำนวนแกนหน่วยประมวลผลกลางที่มีอยู่ แต่ในหน่วยประมวลผลกราฟิกส์ ไม่สามารถใส่ข้อมูลทั้งหมดในครั้งเดียวได้ จึงทดสอบโดยการแบ่งขนาดข้อมูลเป็นส่วนย่อย ขนาดตั้งแต่ 1 เมกะไบต์ ถึง 1024 เมกะไบต์

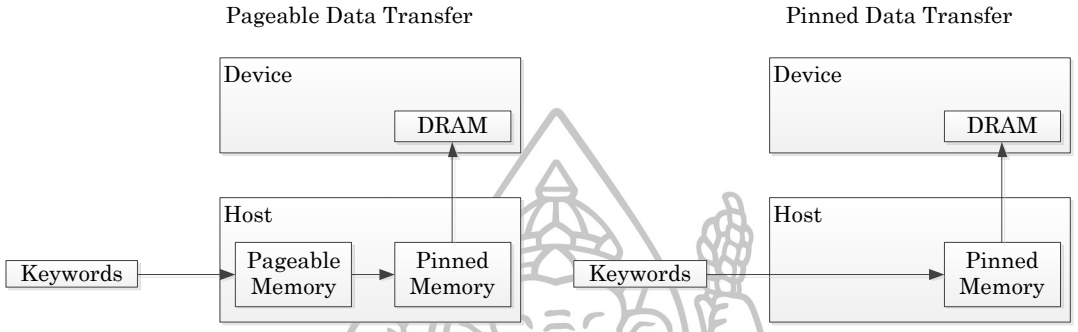
การสร้างอัลกอริทึมหลายแบบเพื่อค้นหาอาร์ดีเอฟบนหน่วยประมวลผลกราฟิกส์ โดยยังเปรียบเทียบกับการประมวลแบบลำดับและแบบมัลติเทรดด้วย OpenMP การทำงานแบบขนานด้วยคู่ด้าประกอบด้วยการใช้หน่วยประมวลผลกราฟิกส์หนึ่งใบ และหลายใบ

การใช้หน่วยประมวลผลกราฟิกส์หลายใบ

เนื่องจากการทำงานของหน่วยประมวลผลกราฟิกส์มีจุดเด่นที่จำนวนแกนและการเพิ่มขนาดของหน่วยความจำภายในหน่วยประมวลผลกราฟิกส์ให้มีขนาดใหญ่ขึ้น เมื่อรันงานจึงต้องมีการ

ตรวจสอบว่า cuda_device มากกว่าจำนวนดีไวส์ที่มีในเครื่องหรือไม่ จากนั้นจึงใช้ฟังก์ชัน cudaSet Device(cuda_device) เพื่อกำหนดว่าต้องการใช้หน่วยประมวลผลกราฟิกส์ลำดับที่เท่าไรจากทั้งหมด

การใช้หน่วยความจำพิน



รูปที่ 5.9 เปรียบเทียบขั้นตอนการย้ายคำสำคัญระหว่างโฮสต์และดีไวส์ แบบ Pageable Data Transfer (ซ้าย) และ Pinned Data Transfer

การทำงานบนดีไวส์จะต้องคัดลอกคำสำคัญ หรือไฟล์อาร์ดีเอฟ จากโฮสต์และไปยังดีไวส์ การใช้งานหน่วยความจำพิน (pinned memory) โดยตรง (รูปที่ 5.9 ขวา) ช่วยลดขั้นตอนการย้ายข้อมูลเหลือเพียง pinned memory (host) ไปยัง device (host) เมื่อเปรียบเทียบกับการใช้หน่วย ความจำแบบเพจ (pageable memory) (รูปที่ 5.9 ซ้าย) ที่ข้อมูลต้องผ่าน pageable memory (host) ไปยัง pinned memory (host) ไปยัง device (host) โดยต้องจัดการเตรียมหน่วยความจำสำหรับข้อมูลขนาดใหญ่มาก

นักวิจัยได้ทำการทดสอบการส่งข้อมูลระหว่างโฮสต์และดีไวส์ เพื่อเปรียบเทียบความแตกต่างของอัตราการย้ายข้อมูล (data transfer rate) ระหว่างโหมดของ pinned memory transfer และ pageable memory transfer ดังตารางที่ 5.3 โค้ดที่นำมาประยุกต์ใช้ในการทดสอบคือ *Testbandwidth* จากตัวอย่างในคูด้า เพื่อใช้วัดอัตราแบนด์วิธของการส่งข้อมูลทั้งสองแบบทดสอบบนเครื่องอกาเมมนอนที่ใช้หน่วยประมวลผลกราฟิกส์ Tesla K20c โดยกำหนดให้ใช้หน่วยประมวลผลกราฟิกส์เพียง 1 ใบ

ตารางที่ 5.3 แสดงแบนด์วิธของการย้ายข้อมูลระหว่างโอสต์และดีไวส์บน Tesla K20c ที่ใช้หน่วยประมวลผลกราฟิกส์ 1 ใบนั้น การทำงานคือ การย้ายข้อมูลจากโฮสต์ไปดีไวส์ พบว่าการใช้หน่วยความจำพินโดยตรง สามารถใช้แบนด์วิธได้มากกว่าการย้ายข้อมูลผ่าน pageable memory

โดยข้อมูลขนาดเล็กจะได้รับประโยชน์สูงสุด เนื่องจากสามารถใช้แบนด์วิธได้มากกว่า 10 เท่า เมื่อข้อมูลมีขนาดมากขึ้นความได้เปรียบนี้จะลดลงจนความแตกต่างของแบนด์วิธเหลือเพียง 1.1 เท่า จากที่ทราบการกำหนดหน่วยความจำพินเกิดขึ้นในฝั่งโฮสต์แต่เมื่อทดสอบเพิ่มเติมภายในดีไวส์ ด้วยการย้ายข้อมูลภายในดีไวส์ พบว่ามีแนวโน้มเหมือนการย้ายข้อมูลจากโฮสต์ไปดีไวส์ นั่นคือ การใช้หน่วยความจำพินโดยตรง สามารถใช้แบนด์วิธได้มากกว่าการย้ายข้อมูลผ่าน pageable memory โดยข้อมูลขนาดขนาดเล็กจะได้รับประโยชน์สูงสุดเช่นกัน เนื่องจากสามารถใช้แบนด์วิธประมาณ 9 เท่า เมื่อข้อมูลมีขนาดมากขึ้นความได้เปรียบนี้จะลดลงจนความแตกต่างของแบนด์วิธเหลือเพียง 1.3 เท่า ส่วนสุดท้ายการทดสอบเพิ่มเติมเรื่องการย้ายข้อมูลกลับจากดีไวส์ไปโฮสต์ พบว่าการย้ายข้อมูลของการใช้งานต่อเนื่องจาก pageable memory ก่อนหน้าหรือหน่วยความจำพินนั้นแทบไม่แตกต่างกัน ประโยชน์สูงสุดในการใช้หน่วยความจำพินจึงอยู่ที่การย้ายข้อมูลจากโฮสต์ไปดีไวส์

ตารางที่ 5.3 แบนด์วิธของการย้ายข้อมูลระหว่างโอสต์และดีไวส์บน Tesla K20c

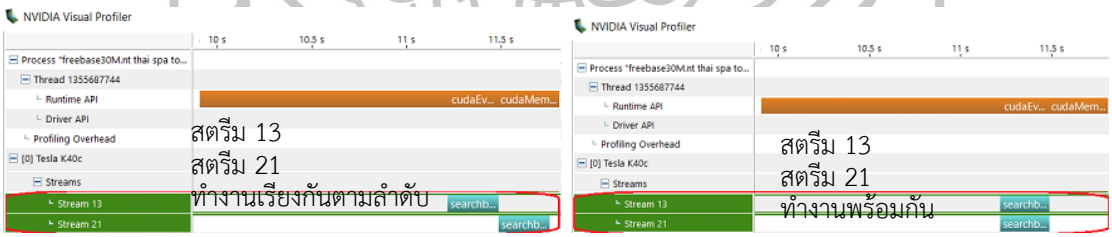
การทดลอง: การย้ายข้อมูลจากโฮสต์ไปดีไวส์ (ประโยชน์หลัก)		
ขนาดข้อมูล (ไบต์)	ใช้หน่วยความจำพินโดยตรง	ย้ายผ่าน pageable memory
	แบนด์วิธ (เมกะไบต์ต่อวินาที)	แบนด์วิธ (เมกะไบต์ต่อวินาที)
1024	407.4	32.2
32768	3906.2	625.0
33554432	6135.2	5535.4
การทดลอง: การย้ายข้อมูลภายในดีไวส์ (เพิ่มเติมด้วยโหมดต่อเนื่อง)		
ขนาดข้อมูล (ไบต์)	ใช้หน่วยความจำพินโดยตรง	ย้ายผ่าน pageable memory
	แบนด์วิธ (เมกะไบต์ต่อวินาที)	แบนด์วิธ (เมกะไบต์ต่อวินาที)
1024	337.5	35.3
32768	4391.7	765.6
33554432	6554.3	4927.3
การทดลอง: การย้ายข้อมูลจากดีไวส์ไปโฮสต์ (เพิ่มเติมด้วยโหมดต่อเนื่อง)		
ขนาดข้อมูล (ไบต์)	ใช้หน่วยความจำพินโดยตรง	ย้ายผ่าน pageable memory
	แบนด์วิธ (เมกะไบต์ต่อวินาที)	แบนด์วิธ (เมกะไบต์ต่อวินาที)
1024	407.4	407.7
32768	12804.1	13106.2
33554432	146943.8	147401.8

การใช้งานโอเปอเรชันสตรีม

การทำงานของโอเปอเรชันสตรีมของคูด้ามีจุดเด่นที่ แอปพลิเคชันสามารถประมวลผลหลายฟังก์ชันพร้อมกันบนหลายโพรเซสเซอร์ได้ การสั่งงานแบบ asynchronous ในคูด้า จะทำการควบคุมการสั่งงานของเทรดจากโฮสต์โดยที่ไม่ต้องรอให้ทุกเทรดของดีไวส์ทำงานเสร็จพร้อมกัน ทำให้ฟังก์ชันในเคอร์เนลล์สามารถทำงานพร้อมกันในคนละสตรีมได้ ตัวอย่างสถานการณ์เหล่านี้

“เคอร์เนลล์แรกทำงานในสตรีมที่ 1 ขณะเดียวกันหน่วยความจำคัดลอกระหว่างที่อยู่สองที่ไปที่หน่วยความจำดีไวส์เดียวกันในสตรีมที่ 2 หน่วยความจำคัดลอกจากโฮสต์ไปดีไวส์ด้วยบล็อกขนาด 64 กิโลไบต์หรือขนาดอื่น ๆ ที่น้อยกว่าในสตรีมที่ 3 ขณะเดียวกันหน่วยความจำคัดลอกฟังก์ชันด้วยคำสั่งที่ลงท้ายด้วย Async อยู่ในสตรีมที่ 4 ขณะเดียวกันหน่วยความจำเริ่มตั้งค่าเรียกฟังก์ชันในสตรีมที่ 5”

การกำหนดให้ว่าสตรีมใดทำงานอะไรนั้นมีสองแบบคือกำหนดโดยคูด้า ถือเป็น default stream เช่น `kernel<<< blocks, threads, bytes >>>();` อีกแบบคือ กำหนดเลขที่ของสตรีมลงไปเช่นต้องการให้สตรีมที่ i ทำงาน ใช้ว่า `kernel<<< blocks, threads, bytes, stream[i] >>>();` โดย `stream[i]` มาจาก `cudaStream_t streams[num_streams];` ซึ่ง `num_streams` เป็นค่าคงที่จำนวนเต็ม และเริ่มสร้างสตรีมด้วย `cudaStreamCreate (&streams[i]);`



รูปที่ 5.10 ตัวอย่างการใช้หลายสตรีมที่ทำงานเป็นลำดับ (ซ้าย) และทำงานพร้อมกัน (ขวา) ตรวจสอบด้วย NVIDIA Visual Profiler

จากรูปที่ 5.10 ตัวอย่างการใช้หลายสตรีมที่ทำงานเป็นลำดับ (ซ้าย) คืองานแรกเสร็จจึงทำงานที่สองและทำงานพร้อมกัน (ขวา) นั่นคือทำงานแรกและทำงานที่สองพร้อมกันบนสตรีมที่ต่างกัน นักพัฒนาโปรแกรมสามารถใช้ NVIDIA VISUAL PROFILER [104] ตรวจสอบ

การทดลองการประยุกต์ใช้การประมวลผลแบบขนานแบบการเปรียบเทียบสตริง

ผู้วิจัยพัฒนาโปรแกรมด้วยอัลกอริทึมค้นหาในด้านต่อไปนี้

1. การค้นหาด้วยวิธีบรูทฟอร์ซ บนหน่วยความจำโกลบอล บนหน่วยความจำโกลบอล โดยใช้หน่วยประมวลผลกราฟิกส์ 1 ใบ และหลายใบ

2. การค้นหาด้วยวิธีบรูทฟอร์ซ โดยใช้การเคลื่อนย้ายข้อมูลผ่านหน่วยความจำพิน โดยตรงบนการ์ดหลายใบ

3. การค้นหาด้วยวิธีบรูทฟอร์ซ บนหน่วยความจำโกลบอล ใช้โอเปอเรชันสตรีม โดยใช้หน่วยประมวลผลกราฟิกส์ 1 ใบ และหลายใบ

การลดขนาดข้อมูลของคำตอบ ผู้วิจัยได้พัฒนาเพิ่มเติมให้ทุกโปรแกรมที่ทำงานบนหน่วยประมวลผลกราฟิกส์ มีการเปรียบเทียบกับเวอร์ชันที่มีการลดขนาดของผลลัพธ์ที่ได้รับจากดีไวส์กลับมาที่โฮสต์ เพื่อลดขนาดของข้อมูลที่นำออก นั่นคือนับเพียงผลลัพธ์ที่ค้นพบแทนการส่งข้อมูลผลลัพธ์ทั้งหมดออกมา

ตารางที่ 5.4 เซตข้อมูล ขนาด จำนวนทริพเพิลและและคำสำคัญที่ประยุกต์ใช้

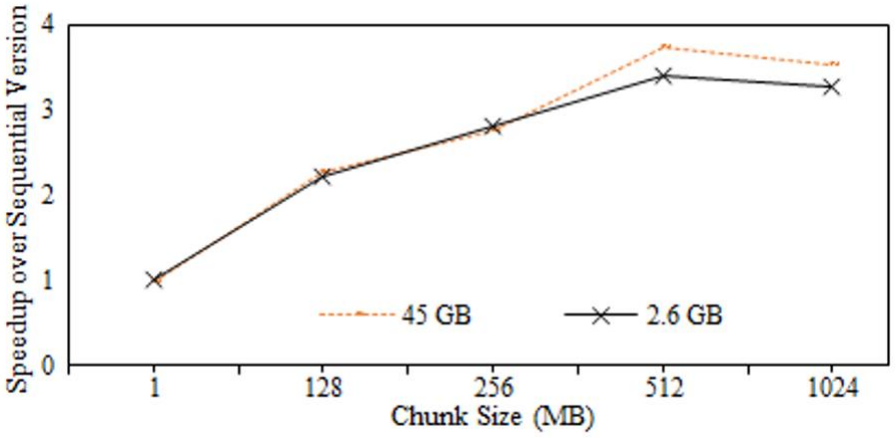
Data Set (Sources)	Size (MB) / # of Triples	Keywords Search
Freebase 2/10 (BTC -2012)	2,600 / 20,000,000	Thailand Health-Tourism Domain: thai spa huahin tourism health wellness massage budget
DBpedia	45,000 / 198,090,024	
Freebase-weekly (07/09/2014)	80,000 / 572, 600	
Karyotype Ontology (JD Warrender, 2013)	1.43 / N/A	Cytogenetic Domain (Random 16 keywords of Finding Kerndrup & Kjeldsen 2001, Cancer Genet Cytogenet), case No. 14 Acute myeloid leukemia, NOS 47, XY, del (5) (q15q31), +8, t (10;12) (q22;q22),t(15;17)(q22;q11),ider(17)(q10)t (15;17) (http://cgap.nci.nih.gov)
MeSH	709 / N/A	
Freebase 1/10 (BTC -2012)	1,320 / 9,999,999	
DBpedia 1/20 (BTC-2012)	1,829 / 9,999,999	
Wikidata 1/10 – (27/07/2015)	5,120 / 32,142,127	
Freebase-weekly (07/09/2014)	400,000 / 2,863, 000	Including 2 groups of keyword searching

การแบ่งขนาดของข้อมูลที่ต้องการนำเข้าหน่วยประมวลผลกราฟิกส์ การแบ่งขนาดของข้อมูล (chunk) ที่ต้องการนำเข้าหน่วยประมวลผลกราฟิกส์ แบ่งเป็น 1 เมกะไบต์ 32 เมกะไบต์ 64 เมกะไบต์ 128 เมกะไบต์ 256 เมกะไบต์ 512 เมกะไบต์ และ1024 เมกะไบต์

นักวิจัยใช้คำสำคัญ 8 คำในโดเมนการท่องเที่ยวเชิงสุขภาพ ทดสอบบนเครื่องคอมพิวเตอร์ส่วนบุคคลระบบปฏิบัติการวินโดว์ และใช้คำในโดเมนด้านเซลล์พันธุศาสตร์ 16 คำ ค้นหาในส่วนข้อมูล (chunk) ขนาด 1 GB ทำงานบน เครื่องเซิร์ฟเวอร์ TeslaK40 สำหรับข้อมูล Freebase-weekly ที่มีขนาด 400 GB นั้นใช้คำค้นหาจากทั้งสองกลุ่มโดเมน โดยโปรแกรมที่พัฒนาขึ้นนี้สามารถทำงานได้ทั้งสองระบบปฏิบัติการ ส่งผลลัพธ์มาเก็บไว้ในหน่วยความจำฝั่งโฮสต์

ผลการทดลองการประยุกต์ใช้การประมวลผลแบบขนานแบบการเปรียบเทียบสตริง

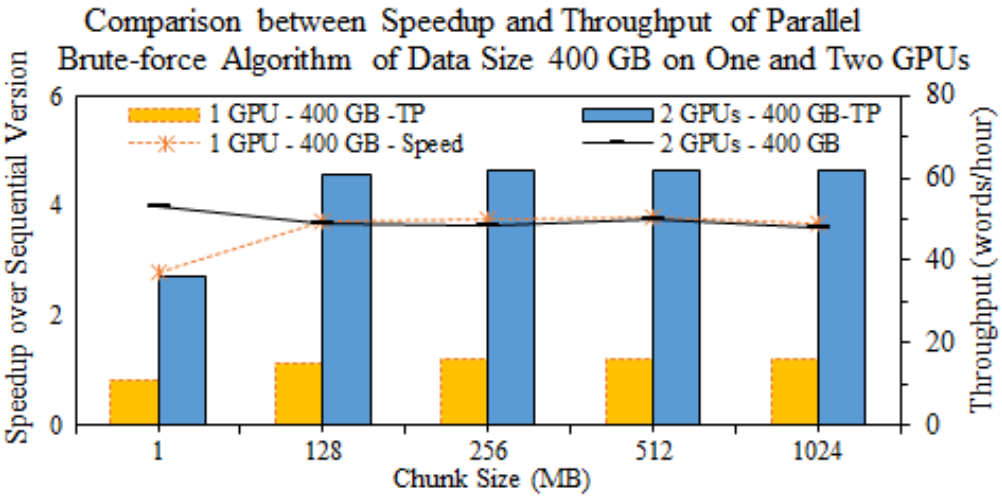
ผลการทดลองที่ได้จากการค้นหาด้วยวิธีบรูทฟอร์ซแบบขนาน และทดสอบกับข้อมูลหลายขนาดแล้วได้ผลการทดลองดังนี้



รูปที่ 5.11 การเปรียบเทียบการเร่งความเร็วระหว่างการค้นหาแบบลำดับและมัลติเทรดบนข้อมูลขนาด 2.6 และ 45 กิกะไบต์

รูปที่ 5.11 แสดงผลการเร่งความเร็ว (แกน y) ระหว่างการค้นหาบรูทฟอร์ซ แบบมัลติเทรดด้วย OpenMP และเร่งเหนือกว่าแบบลำดับ บนข้อมูลขนาด 2.6 และ 45 กิกะไบต์ ด้วยการแบ่งขนาดข้อมูลเป็นส่วนละ 1 เมกะไบต์ 128 เมกะไบต์ 256 เมกะไบต์ 512 เมกะไบต์ และ 1024 เมกะไบต์ ตามลำดับ (แกน x) พบว่าการเร่งความเร็วของ OpenMP เท่ากับการทำงานแบบลำดับสำหรับข้อมูลขนาด 1 เมกะไบต์ 128 เมกะไบต์ และ 256 เมกะไบต์ คือ 1 เท่า 2 เท่าและ 3 เท่า ตามลำดับ การเร่งความเร็วของ OpenMP ดีกว่าการทำงานแบบลำดับเมื่อแบ่งเป็นส่วนย่อยขนาด 512 เมกะ

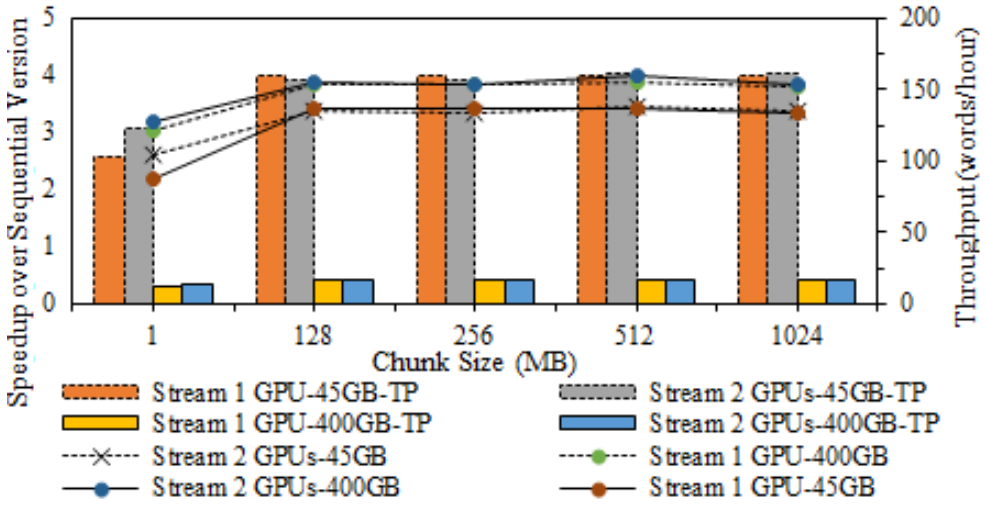
ไบต์ ของกรณีข้อมูลขนาด 45 กิกะไบต์มีค่า 3.5 เท่ามากกว่าส่วนย่อยขนาดเดียวกันของข้อมูลขนาด 2.6 กิกะไบต์ และการเร่งความเร็วลดลงเล็กน้อยเมื่อใช้ส่วนย่อยขนาด 1024 เมกะไบต์ แต่ความเร็วของกรณีข้อมูลขนาด 45 กิกะไบต์ยังมากกว่ากรณีข้อมูลขนาด 2.6 กิกะไบต์



รูปที่ 5.12 การเปรียบเทียบระหว่างการเร่งความเร็ว (กราฟเส้น) และธรูพุต (กราฟแท่ง) ของอัลกอริทึมบรูทฟอร์ซแบบขนานของข้อมูลขนาด 400 กิกะไบต์ บนหน่วยประมวลกราฟิกส์จำนวน 1 และ 2 ใบ

จากรูปที่ 5.12 การทดสอบการค้นหาคำสำคัญจำนวน 8 คำในข้อมูลขนาดใหญ่ที่สุดในการทดลองนี้คือ 400 GB ซึ่งมีขนาดใหญ่กว่าหน่วยความจำของหน่วยประมวลผลกราฟิกส์หลายสิบเท่า บนหน่วยประมวลผลกราฟิกส์จำนวน 1 และ 2 ใบ พบว่า การเร่งความเร็ว (กราฟเส้น) บนหน่วยประมวลผลกราฟิกส์จำนวน 2 ใบ สามารถเร่งความเร็วได้มากกว่าหน่วยประมวลผลกราฟิกส์จำนวน 1 ใบ เมื่อแบ่งข้อมูลออกเป็นชิ้นละ 1 เมกะไบต์ (ดูแกน y ซ้ายประกอบกราฟเส้น)

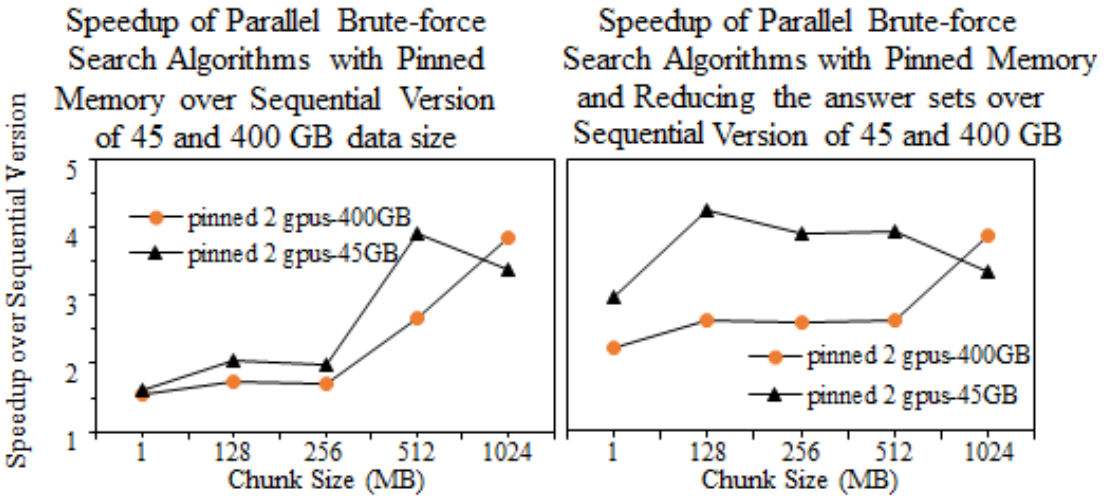
ส่วนธรูพุตของการค้นหาคำสำคัญเฉลี่ยภายใน 1 ชั่วโมง (กราฟแท่ง) เมื่อแบ่งข้อมูลออกเป็นส่วนๆ ละ 1 เมกะไบต์ (ดูตามแกน y ด้านขวา) พบว่าการแบ่งส่วนขนาดเล็ก 1 เมกะไบต์เพื่อนำเข้าไปประมวลผลบนหน่วยประมวลผลกราฟิกส์ 1 ใบ จะมีธรูพุต 10 คำต่อ 1 ชั่วโมงซึ่งน้อยกว่าประมวลผลบนหน่วยประมวลผลกราฟิกส์ 2 ใบ ที่อยู่ที่อัตรา 35 คำต่อ 1 ชั่วโมง และเป็นอัตราที่น้อยที่สุดในรูป เมื่อพิจารณาการแบ่งส่วนข้อมูลขนาด 128 เมกะไบต์ 256 เมกะไบต์ 512 เมกะไบต์และ 1024 เมกะไบต์เพื่อนำเข้าไปประมวลผลบนหน่วยประมวลผลกราฟิกส์ 1 ใบ มีค่าธรูพุตเฉลี่ยอยู่ที่อัตรา 15 คำต่อชั่วโมงซึ่งเพิ่มขึ้นจากข้อมูลขนาด 1 เมกะไบต์เพียงเล็กน้อย แต่ยังน้อยกว่ากรณีการนำเข้าไปประมวลผลบนหน่วยประมวลผลกราฟิกส์ 2 ใบเมื่อแบ่งข้อมูลเป็นขนาด 128 เมกะไบต์ 256 เมกะไบต์ 512 เมกะไบต์และ 1024 เมกะไบต์ ที่มีธรูพุตเฉลี่ยอยู่ที่ 60 คำต่อชั่วโมง



รูปที่ 5.13 การเร่งความเร็วของโอเปอเรชันสตรีมและธรูพุตบนหน่วยประมวลผลกราฟิกส์ 1-2 ใบ เมื่อค้นหาด้วยคำสำคัญ 8 คำ เทียบกับการทำงานแบบลำดับของข้อมูลขนาด 45 และ 400 กิกะไบต์

จากรูปที่ 5.13 การเร่งความเร็วของโอเปอเรชันสตรีมและธรูพุตบนหน่วยประมวลผลกราฟิกส์ 1-2 ใบ เมื่อค้นหาด้วยคำสำคัญ 8 คำ เทียบกับการทำงานแบบลำดับของข้อมูลขนาด 45 และ 400 กิกะไบต์ เมื่อลงในรายละเอียดแล้ว การเร่งความเร็วของเซตข้อมูล (แกน y ซ้าย) ดูข้อมูลตามกราฟเส้น พบว่าการเร่งความเร็วด้วยหน่วยประมวลผลกราฟิกส์ 1 และ 2 ใบ และสตรีมโอเปเรชันของเซตข้อมูลขนาด 400 กิกะไบต์ พบว่าทำได้ดีกว่าเวอร์ชันการค้นหาแบบลำดับ เมื่อแบ่งข้อมูลเป็นขนาดเล็ก ส่วนละ 1 เมกะไบต์ ทำได้ 3 เท่า และได้ถึง 4 เท่า เมื่อข้อมูลที่นำเข้าประมวลผลมีขนาดตั้งแต่ 128 เมกะไบต์ถึง 1,024 เมกะไบต์ ซึ่งเป็นการเร่งความเร็วที่มากกว่า สตรีมโอเปเรชันของเซตข้อมูลขนาด 45 กิกะไบต์ เมื่อแบ่งข้อมูลเป็นขนาดเล็กส่วนละ 1 เมกะไบต์ ทำได้ 2 เท่า และทำได้เป็น 3 เท่าเมื่อข้อมูลที่นำเข้าประมวลผลมีขนาดตั้งแต่ 128 เมกะไบต์ถึง 1,024 เมกะไบต์ ตามกราฟเส้นที่ขนานกัน

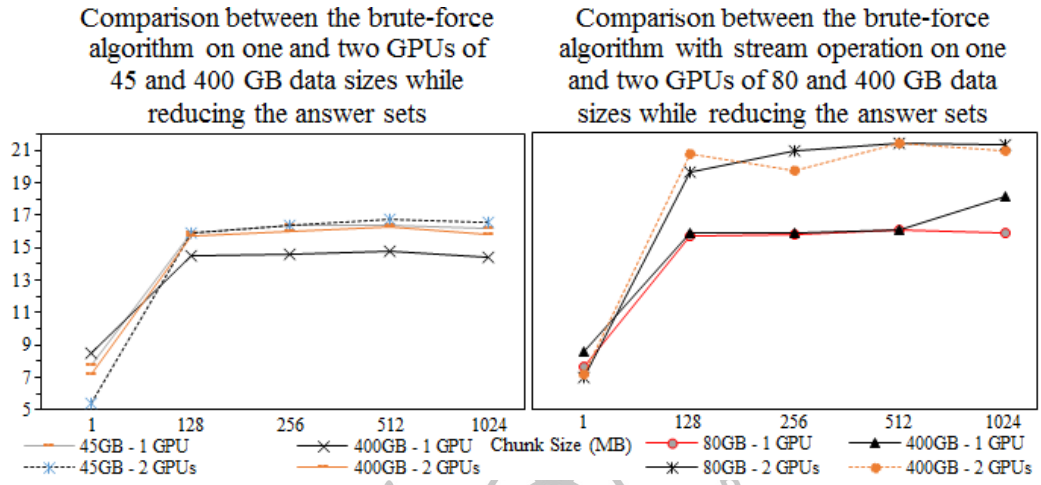
สำหรับด้านธรูพุต (แกน y ขวา และข้อมูลแบบกราฟแท่ง) ของการค้นหาคำสำคัญต่อหนึ่งชั่วโมง พบว่า เซตข้อมูลขนาด 45 กิกะไบต์ (กราฟแท่งขอบเส้นประ) เมื่อแบ่งเป็นไฟล์ขนาดเล็ก 1 เมกะไบต์ บนหน่วยประมวลผลกราฟิกส์ 1 ใบ สามารถค้นหาได้ 100 คำต่อ 1 ชั่วโมง เมื่อเป็นหน่วยประมวลผลกราฟิกส์จำนวน 2 ใบสามารถค้นหาได้มากขึ้นเป็น 125 คำต่อชั่วโมง เมื่อแบ่งเป็นไฟล์ขนาด 128 – 1024 เมกะไบต์ บนหน่วยประมวลผลกราฟิกส์ 1 และ 2 ใบ สามารถค้นหาได้ที่อัตรา 160 คำต่อ 1 ชั่วโมง ขณะที่เซตข้อมูลขนาด 400 กิกะไบต์ (กราฟแท่งขอบเส้นทึบ) เมื่อแบ่งเป็นไฟล์ขนาด 1 – 1,024 เมกะไบต์ บนหน่วยประมวลผลกราฟิกส์ 1 และ 2 ใบ สามารถค้นหาได้ 16 คำต่อ 1 ชั่วโมง



รูปที่ 5.14 การเร่งความเร็วของอัลกอริทึมแบบขนานของการค้นหาแบบบรูทฟอร์ซ โดยการย้ายข้อมูลด้วยหน่วยความจำพินโดยตรง (ซ้าย) และแบบลดเซตคำตอบ (ขวา) เทียบกับโปรแกรมค้นหาแบบลำดับของเซตข้อมูลขนาด 45 และ 400 กิกะไบต์

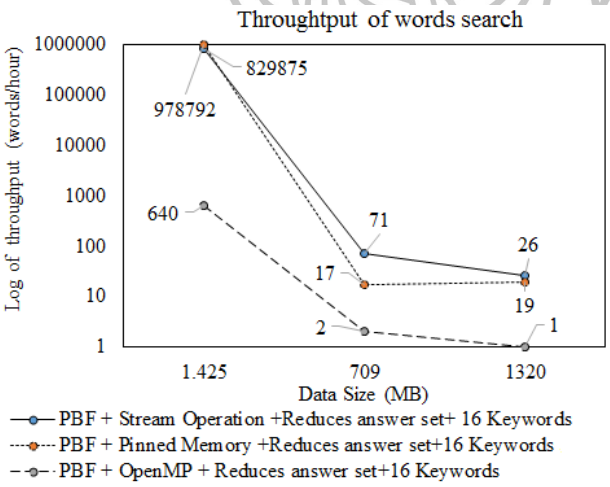
จากผลการทดลองที่ผ่านมา (รูปที่ 5.12 และ 5.13) นักวิจัยพบว่าการใช้งานหน่วยประมวลผลกราฟิกส์จำนวน 2 ใบ มีประสิทธิภาพในการทำงานที่ดีกว่าหน่วยประมวลผลกราฟิกส์เพียงใบเดียว จากรูปที่ 5.14 ได้นำการเร่งความเร็วของอัลกอริทึมแบบขนานของการค้นหาแบบบรูทฟอร์ซ โดยการย้ายข้อมูลด้วยหน่วยความจำพินโดยตรง (ซ้าย) และแบบลดเซตคำตอบ (ขวา) เทียบกับการค้นหาแบบลำดับของเซตข้อมูลขนาด 45 และ 400 กิกะไบต์

เซตข้อมูลขนาด 45 กิกะไบต์ จากภาพซ้ายการเร่งความเร็วของข้อมูลตั้งแต่ 1 – 512 เมกะไบต์ มีแนวโน้มเพิ่มขึ้นครึ่งเท่าจากข้อมูลขนาด 1 เมกะไบต์เมื่อเป็นข้อมูลขนาด 128 – 256 เมกะไบต์ และเพิ่มขึ้น 2 เท่า (จาก 2 เป็น 4 เท่า) เมื่อข้อมูลมีขนาด 512 เมกะไบต์ แล้วความเร็วตกลงเมื่อแบ่งข้อมูลขนาด 1,024 เมกะไบต์สำหรับเซตข้อมูลขนาด 45 เมกะไบต์เช่นกัน เมื่อพิจารณาการลดขนาดของคำตอบพบว่า การเร่งความเร็วเพิ่มขึ้น 1.5 เท่าจากสำหรับข้อมูลขนาด 1 เมกะไบต์ เมื่อเปรียบเทียบกับภาพซ้ายที่ไม่มีการลดขนาดคำตอบ เมื่อเป็นการแบ่งข้อมูลขนาด 128 เมกะไบต์พบว่ามีการเร่งความเร็วสูงสุดคือประมาณ 4.25 เท่ามากกว่าการแบ่งเป็นข้อมูลขนาด 256 ไฟล์ที่มีการเร่งความเร็วเป็น 4 เท่า (มากกว่าภาพซ้าย) แต่เมื่อข้อมูล 512 – 1,024 เมกะไบต์ การเร่งความเร็วไม่เปลี่ยนแปลงจากกรณีที่ไม่ลดขนาดคำตอบ (ภาพซ้าย) สำหรับเซตข้อมูลขนาด 400 กิกะไบต์ มีแนวโน้มเร่งความเร็วได้มากขึ้นเช่นกันในกรณีลดขนาดคำตอบ แต่มีผลกับการแบ่งไฟล์ย่อยขนาด 1 – 256 เมกะไบต์เท่านั้น ซึ่งจากการตรวจสอบร่วมกับตารางที่ 5.3 พบว่าการเคลื่อนย้ายข้อมูลมาที่หน่วยความจำพินโดยตรงมีผลดีกับข้อมูลขนาดเล็กและการย้ายจากโฮสต์มาดีไวส์เท่านั้น



รูปที่ 5.15 การเปรียบเทียบระหว่างอัลกอริทึมบรูทฟอร์ซบนหน่วยประมวลผลกราฟิกส์ 1 และ 2 ใบ สำหรับข้อมูลขนาด 45 และ 400 กิกะไบต์ ที่มีการลดขนาดคำตอบ (ซ้าย) และของข้อมูลขนาด 80 และ 400 กิกะไบต์ ที่เพิ่มโอเปอเรชันสตรีม (ขวา)

จากรูปที่ 5.15 การเปรียบเทียบระหว่างอัลกอริทึมบรูทฟอร์ซบนหน่วยประมวลผลกราฟิกส์ 1 และ 2 ใบ สำหรับข้อมูลขนาด 45 และ 400 กิกะไบต์ ที่มีการลดขนาดคำตอบ (ซ้าย) และของข้อมูลขนาด 80 และ 400 กิกะไบต์ ที่เพิ่มโอเปอเรชันสตรีม (ขวา) พิจารณาที่ข้อมูลขนาด 400 กิกะไบต์ พบว่าในกรณีหน่วยประมวลผลกราฟิกส์ 2 ใบโดยใช้โอเปอเรชันสตรีม สามารถเร่งความเร็วได้มากถึง 21 เท่าในกรณีที่แบ่งข้อมูลขนาด 512 เมกะไบต์

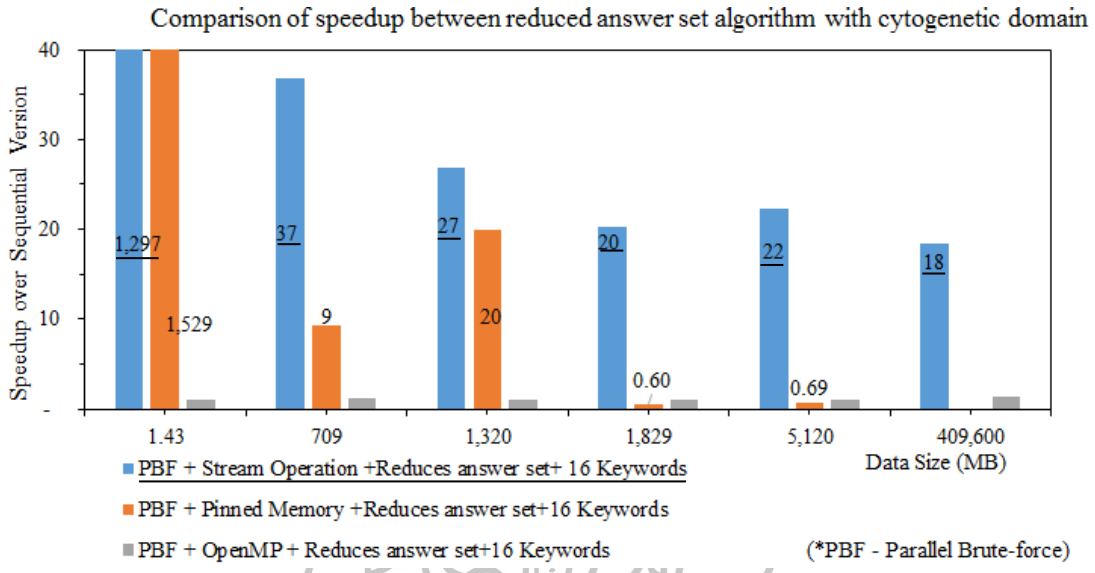


*PBF = Parallel Brute-Force

รูปที่ 5.16 ธรูพุตของการค้นหาคำสำคัญในโดเมนเซล์พันธุศาสตร์

จากรูปที่ 5.16 และรูปที่ 5.17 เป็นการแสดงการค้นหาคำสำคัญจำนวน 16 คำ และแบ่งข้อมูลย่อยขนาด 1,024 เมกะไบต์ พิจารณารูปที่ 5.16 ธรูพุตของการค้นหาคำสำคัญนั้นมีการค้นหาจำ นวนคำได้ในอัตราแสนคำต่อชั่วโมง ถือว่ามากที่สุดในไฟล์ขนาดเล็กที่สุดคือ 1.425 เมกะไบต์ เมื่อ

ไฟล์มีขนาดใหญ่ขึ้นธรูพุตจะลดลง อย่างไรก็ตามการประมวลผลแบบขนานบนหน่วยประมวลผลกราฟิกส์มีธรูพุตมากกว่าการประมวลผลแบบขนานมัลติเทรด OpenMP บนหน่วยประมวลผลกลาง ตั้งแต่ 26 – 1,529 เท่า เรียงตามลำดับขนาดไฟล์ใหญ่ที่สุดไปยังไฟล์ขนาดเล็กสุด



รูปที่ 5.17 การเปรียบเทียบการเร่งความเร็วระหว่างอัลกอริทึมบรูทฟอร์ซแบบขนาน ที่มีการลดขนาดเซตคำตอบ ของเวอร์ชันที่ใช้โอเปอเรชันสตรีม การย้ายข้อมูลเข้าหน่วยความจำพินและมัลติเธรด OpenMP ของข้อมูลที่มีขนาดตั้งแต่ 1.43 – 409,600 เมกะไบต์

เมื่อพิจารณารูปที่ 5.17 การเปรียบเทียบการเร่งความเร็วระหว่างอัลกอริทึมบรูท - ฟอร์ซแบบขนาน ที่มีการลดขนาดเซตคำตอบของเวอร์ชันที่ใช้โอเปอเรชันสตรีม (กราฟแท่งแรกสัญลักษณ์มีขีดเส้นใต้ข้อมูล) การย้ายข้อมูลเข้าหน่วยความจำพินและมัลติเธรด OpenMP ของข้อมูลที่มีขนาดตั้งแต่ 1.43 – 409,600 เมกะไบต์ พบว่า การเร่งความเร็วของหน่วยประมวลผลกราฟิกส์หลายใบ ของไฟล์ที่มีขนาดเล็ก 1.43 เมกะไบต์ จะมีการเร่งความเร็วได้มากที่สุด เกิน 1,500 เท่าเมื่อย้ายข้อมูลเข้าหน่วย ความจำพินโดยตรงซึ่งมากกว่าเมื่อใช้สตรีมโอเปอเรชันที่เร่งความเร็วได้เกิน 1,200 เท่า เมื่อข้อมูลมีขนาดมากขึ้นนั่นคือตั้งแต่ 1,829 เมกะไบต์ขึ้นไป การย้ายข้อมูลเข้าหน่วยความจำพินโดยตรงไม่มีผลในการเร่งความเร็ว ซึ่งแตกต่างจากโอเปอเรชันสตรีมที่ยังคงเร่งความเร็วได้ 20 เท่าและ ลดลงเหลือ 18 เท่าเมื่อข้อมูลมีขนาด 409,600 เมกะไบต์ ในขณะที่การทำงานแบบขนานด้วยมัลติเทรด OpenMP นั้นสามารถเร่งความเร็วได้ไม่เกิน 2 เท่าของไฟล์ทุกขนาด

บทที่ 6

สรุปผลการวิจัย

การประมวลผลแบบขนานของลิงค์เดต้าบนสถาปัตยกรรมแบบหลายแกนนั้น จำเป็นต้องให้เทรดหลายเทรดช่วยกันทำงานให้มากที่สุด ซึ่งเทรดเหล่านี้ต้องทำงานกับข้อมูลขนาดใหญ่ซึ่งไม่อาจจะนำเข้าไปยังหน่วยประมวลผลกราฟิกส์ทั้งหมดไม่ได้ งานวิจัยนี้จึงต้องพยายามย่อขนาดข้อมูลให้มีขนาดเล็ก เพื่อนำเข้าหน่วยประมวลกราฟิกส์ และทำการค้นหาด้วยการเร่งความเร็วให้มีธรูพุตการค้นหาคำต่อชั่วโมงที่มากที่สุด

เมื่อเริ่มแรกทำการทดสอบกับฐานข้อมูลแบบคีย์ค่าแล้วพบว่าขนาดข้อมูลไม่เป็นเล็กที่น่าพอใจ เมื่อเทียบกับการย่อไฟล์ทริพเพิลเป็นไบนารีของรูปแบบเอชดีที จึงได้นำรูปแบบเอชดีทีมาเป็นโครงสร้างฐานเพื่อต่อยอดในการย่อข้อมูล ดังนั้นจึงเป็นขั้นตอนการทำงานก่อนการคิวรี่บนหน่วยประมวลผลกราฟิกส์ตามวัตถุประสงค์ของการวิจัย นั่นคือการออกแบบโครงสร้างข้อมูลแบบซีบีเอ็ม (CBM, Combined BitMap representation) ที่ลดขนาดข้อมูล 1.7 กิกะไบต์ จากรูปแบบทริพเพิลได้ถึง 93% พัฒนาอัลกอริทึมการค้นหาตามดัชนี subject predicate object และ predicate object subject ด้วยการประยุกต์ใช้การค้นหาบรูทฟอร์ซแบบขนานโดยการใช้หน่วยความจำโกลบอลเพียงอย่างเดียว ซีบีเอ็มของไฟล์ขนาด 108-135 เมกะไบต์ นำเข้า 18,331,369 บิต ธรูพุต $5.6x10^{10}$ บิต/ชั่วโมง โดย 1 บิตคือ 1 ตำแหน่งของ subject, predicate หรือ object ในงานวิจัยได้ใช้ความรู้ด้านวิศวกรรมซอฟต์แวร์มาพัฒนาโปรแกรมเพื่อนำข้อมูลฐานความรู้ ไฟล์ออนโทโลยี ไปประมวลผลบนสถาปัตยกรรมแบบขนานได้ สามารถค้นหาข้อมูลที่ถูกย่อลงด้วยเวลาอันรวดเร็ว สามารถประยุกต์มาใช้กับเครื่องคอมพิวเตอร์ส่วนบุคคลที่มีหน่วยประมวลผลกราฟิกส์ตั้งแต่รุ่นเทสลาถึงแมกซ์เวลล์ได้ แต่เมื่อพิจารณาในการทดสอบที่มีการใช้โอเปอเรชันสตรีมของคูด้า สามารถทำงานบนหน่วยประมวลผลกราฟิกส์ที่มีสถาปัตยกรรมรองรับให้ใช้ตั้งแต่รุ่นเฟอร์มิเป็นต้นไป สำหรับการใช้เวลาหลายร้อยชั่วโมงเพื่อทำการแปลงและย่อข้อมูลก่อนการประมวลผลนั้นสามารถแก้ไขได้โดยการหาเซิร์ฟเวอร์ที่มีศักยภาพ อาจเป็นการทำงานบนคลาวด์ ซึ่งสามารถลงทุนทำการแปลงข้อมูลหนึ่งครั้งแล้วใช้งานได้ระยะเวลาหนึ่ง เนื่องจากเซตข้อมูลที่ได้รับการเผยแพร่ด้วยเทคโนโลยีลิงค์เดต้านี้มีการอัพเดตไม่บ่อยครั้ง เช่น รายปักษ์ รายเดือน รายไตรมาส หรือครึ่งปีถึงหนึ่งปี เป็นต้น ดังนั้นข้อมูลที่ได้มาจึงเป็นข้อมูลขนาดใหญ่ที่เน้นการอ่านข้อมูล

ในกรณีที่ต้องการลดขั้นตอนก่อนและหลังการประมวลผล การประมวลผลโดยตรงเป็นวิธีที่เรียบง่ายและได้ผลดีสำหรับข้อมูลสตริง เพื่อการคิวรี่ข้อมูลอาร์ดีเอฟโดยไม่คิดถึงตำแหน่งของคอนเซปต์ (subject) มองไฟล์เป็นอักขระธรรมดาที่ทำการเปรียบเทียบสตริงกับคำสำคัญได้ ไม่เสียเวลาในการแปลงและย่อข้อมูล สำหรับระดับการเร่งความเร็วผ่านการปรับปรุงการค้นหาด้วยวิธีบรูทฟอร์ซแบบขนานพบว่าการทำงานร่วมกับหน่วยความจำแชร์ ซึ่งทำหน้าที่เก็บคำสำคัญหลายคำช่วยลดเวลาในการย้ายคำสำคัญเข้าสู่หน่วยประมวลผลกราฟิกส์ได้ ส่งผลให้ลดเวลาย้ายข้อมูลทำให้สามารถเร่งความเร็วในการค้นหาสำหรับชิ้นส่วนไฟล์ขนาด 1 เมกะไบต์ การปรับปรุงการค้นหาด้วยวิธีบรูทฟอร์ซแบบขนานร่วมกับการเคลื่อนย้ายข้อมูลเข้าสู่หน่วยความจำพินโดยตรงช่วยลดเวลาย้ายข้อมูลจากโฮสต์สู่ดีไวส์ได้ และส่งผลให้เร่งความเร็วได้ดีเมื่อชิ้นไฟล์มีขนาดเล็ก แต่ไม่มีการเร่งความเร็วเพิ่มเติมเมื่อชิ้นไฟล์มีขนาดใหญ่ การใช้หน่วยความจำโกลบอลเพียงอย่างเดียวสามารถเร่งความเร็วมากกว่าการค้นหาแบบลำดับและแบบมัลติเทรดได้ ในกรณีที่ใช้หน่วยประมวลผลกราฟิกส์หลายใบจะทำให้ความเร็วเพิ่มขึ้นได้อีก แต่ต้องมีการรองานก่อนหน้าเสร็จก่อนแล้วจึงเริ่มทำงานในเคอร์เนลล์คูด้าจึงสนับสนุนการทำงานแบบสตรีมที่ยอมให้เคอร์เนลล์หลายเคอร์เนลล์ทำงานในเวลาเดียวกัน คนละช่องสตรีมได้ ทำให้เพิ่มประสิทธิภาพการทำงานได้อีก จะเห็นการทำงานได้ชัดเจนมากขึ้นเมื่อดูผ่านเครื่องมือเช่น NVIDIA Visual Profiler หรือ Parallel Nsight ซึ่งการใช้สตรีมนี้สามารถเร่งความเร็วได้มากที่สุดเมื่อทำงานร่วมกับการใช้หน่วยประมวลผลกราฟิกส์หลายใบ ขนาดไฟล์อักขระ < 1 เมกะไบต์ ธรูพุต > 978,792 คำต่อชั่วโมง การเร่งความเร็ว > 10,000 เท่าของแบบลำดับ ขนาดไฟล์อักขระ 400 กิกะไบต์ ธรูพุตลดลงเหลือ 414 คำต่อชั่วโมง ความเร็ว 14-24 เท่าของแบบลำดับ

การย้ายข้อมูลขนาดใหญ่เข้าสู่หน่วยประมวลผลกราฟิกส์ยังถือเป็นปัญหาใหม่ แม้ว่าจะมีการแปลงข้อมูลให้มีขนาดเล็กลงแล้วก็ตาม จากการทดสอบ [139] และ [109] พบว่าการนำข้อมูลขนาดเล็กเข้าไปประมวลผลบนหน่วยประมวลผลกราฟิกส์มักได้ผลการเร่งความเร็ว ธรูพุตของการค้นหาที่ดีกว่าเนื่องจากใช้เวลาในการขนย้ายข้อมูลน้อยกว่า เมื่อมีการเพิ่มจำนวนและขนาดของเซตข้อมูลให้มากขึ้น สามารถใช้ระดับของหน่วยความจำที่ใกล้กับเทรดมาช่วยในการประมวลผลได้ แต่ขนาดของหน่วยความจำเหล่านี้มีขนาดเล็กลงไปอีก โดยมีขนาดเล็กกว่าหน่วยประมวลผลโกลบอลทั้งสิ้น และมีข้อจำกัดในเรื่องการอ่านอย่างเดียวเช่น เท็กเซอร์ ไม่สามารถเปลี่ยนข้อมูลได้ เช่น คอนสแตนซ์ มีขนาดเล็กและทำงานเฉพาะบล็อกเช่นหน่วยความจำแชร์ทำให้ต้องมองที่วัตถุประสงค์ที่ต้องการในการทำคิวรี่ใดก่อนเสมอ แล้วจึงเลือกใช้ทรัพยากรที่มีให้เหมาะสม สำหรับหน่วยความจำยูนิฟายที่มีเพื่ออำนวยความสะดวก โดยลดการคัดลอกข้อมูลระหว่างหน่วยความจำของระบบ และหน่วยความจำของดีไวส์ พบว่ายังมีจุดบกพร่องอยู่ นั่นคือการนำข้อมูลเข้าก็ยังต้องทำผ่าน PCIexpress ที่มีขนาดไม่เกิน 6GBps ที่ถือเป็นคอขวดในการเคลื่อนย้ายข้อมูล จากการศึกษาพบว่าใน

อนาคตหน่วยประมวลผลกราฟิกส์ในรุ่นพาสคาลได้มีการนำบอร์ดที่ประกอบด้วยสถาปัตยกรรม NVLink [141] ที่ถือได้ว่าเป็นเลนเคลื่อนย้ายข้อมูลความเร็วสูงมาประกอบกับหน่วยความจำยูนิฟายที่ปรับปรุงพร้อมกับคูด้า 8 จะช่วยอำนวยความสะดวกแก่ผู้พัฒนาระบบ

ในอนาคตสามารถพัฒนางานวิจัยนี้ต่อไปได้ โดยการนำวิธีการไปต่อยอดรองรับคิวรี่ที่ซับซ้อน และการดึงข้อมูลแบบสตรีมผ่านเครือข่าย ไม่ต้องโหลดทั้งหมดแบบออฟไลน์ การใช้เทคนิค NVgraph ช่วยค้นหาข้อมูลรูปแบบกราฟ รวมทั้งการใช้ Deep learning ในการช่วยรวบรวมข้อมูลในรูปแบบลิงค์ดาต้า เพื่อเพิ่มฐานความรู้แบบอัตโนมัต

รายการอ้างอิง

[1] ธีรณี อจลากุล. (2549). **การประมวลผลแบบขนานและแบบกระจาย**. โครงการหนังสืออิเล็กทรอนิกส์ สำหรับระดับอุดมศึกษา โดย สำนักงานคณะกรรมการการอุดมศึกษา กระทรวงศึกษาธิการ. เข้าถึงเมื่อ 17 สิงหาคม. จากฐานข้อมูลวิทยานิพนธ์ไทย http://www.thailis.or.th/ebook/ebook.php?RecId=2

[2] Lea, Doug. (1999). **Concurrent Programming in Java: Design Principles and Patterns.** 2nd ed. Boston: Addison-Wesley Longman Publishing.

[3] Flynn, M. J. (1972). "Some Computer Organizations and Their Effectiveness." **IEEE Trans. Comput** 21, 9: 948–960.

[4] Catanzaro, Bryan. (2012). **An Introduction to CUDA/OpenCL and Manycore Graphics Processors**. Accessed February 11, 2015. Available from http://parlab.eecs.berkeley.edu/sites/all/parlab/files/CatanzaroIntroToCUDAOpenCL.pdf

[5] Dagum, L., and R. Menon. (1998). "OpenMP: an industry standard API for shared-memory programming." **IEEE Comput. Sci. Eng**. 5: 46–55.

[6] Bull, J. M., and M. E. Kambites. (2000). "JOMP-an OpenMP-like interface for Java." In **Proceedings of the ACM 2000 conference on Java Grande**, 44-53. San Francisco, California, June 3-5, 2000.

[7] Michael, K. et al. (2007). "JaMP: an implementation of OpenMP for a Java DSM: Research Articles." **Concurr Comput Pr. Exper**. 19. 2333–2352.

[8] Kaminsky, A. (2009). **Building Parallel Programs: SMPs, Clusters & Java**. Cengage Course Technology, 1st ed.

[9] Robert, J. and others. (2009). "A type and effect system for deterministic parallel Java." **SIGPLAN** 44: 97–116.

[10] Oracle. (2015). **Thread (Java Platform SE8)**. Accessed August 02,2016. Available from https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html.

[11] Amdahl, Gene. M. (1967). "Validity of the single processor approach to achieving large scale computing capabilities." In **Proceedings of Spring Joint Computer Conference**, 483–485. April 18-20, 1967.

[12] Haring, G., and G. Kotsis. (1995). "Workload modeling for parallel processing systems." In **Proceedings of the Third International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems**, 8–12.

[13] Lublin, Uri, and D. G. Feitelson. (2003). "The workload on parallel super-computers: modeling the characteristics of rigid jobs." **Journal of Parallel and Distributed Computing** 63, 11: 1105–1122.

[14] Holloway, A. L. and others. (2007). "How to barter bits for chronons: compression and bandwidth tradeoffs for database scans." In **Proceedings of the 2007 ACM SIGMOD international conference on Management of data**, 389–400.

[15] Zukowski, M. and others. (2006). "Super-Scalar RAM-CPU Cache Compression." In **Proc. 22nd Int. Conf Data Eng. 2006 ICDE 06,** 59–59.

[16] He, B. and others. (2007). "Efficient gather and scatter operations on graphics processors." In **Proceedings of the 2007 ACM/IEEE conference on Supercomputing**, 1–12.

[17] Kim, C. and others. (2011). "Designing fast architecture-sensitive tree search on modern multicore/many-core processors." **ACM Trans Database Syst**. 36, 4. 1–34.

[18] He, Bingsheng. and others. (2009). "Relational query coprocessing on graphics processors." **ACM Trans Database Syst**. 34, 4 (December 2009): 1–39.

[19] Farber, Rob. (2011). **CUDA Application Design and Development**. Morgan Kaufmann Publishers Inc.

[20] **jCUDA - Java bindings for CUDA.** Accessed November 13, 2013. Available from http://jcuda.org/jcuda/JCuda.html.

[21] Yonghong, Yan, Grossman Max, and Sarkar Vivek. (2009). "JCUDA: A Programmer-Friendly Interface for Accelerating Java Programs with CUDA." In **Proceedings of the 15th International Euro-Par Conference on Parallel Processing**, 887-899. Berlin: Springer-Verlag.

[22] **MPJ Express Project**. Accessed November 13,2013. Available from http://mpj-express.org/.

[23] Lea, Doug. (2000). "A Java fork/join framework." In **Proceedings of the ACM 2000 conference on Java Grande,** 36-43. San Francisco, California, June 3-5, 2000.

[24] Goetz, Brian, and Tim Peierls. (2006). **Java concurrency in practice**. Pearson Education

[25] Dean, Jeffrey, and Ghemawat Sanjay. (2004). "MapReduce: simplified data processing on large clusters." In **Proceedings of the 6th conference on**

Symposium on Opearting Systems Design & Implementation.6. 10.

[26] Zheng Wei and Joseph JaJa. (2012) "A fast algorithm for constructing inverted files on heterogeneous platforms." **J. Parallel Distrib. Comput**. 72, 5 (May): 728–738.

[27] U. Kang, Hanghang Tong, Jimeng Sun, Ching-Yung Lin, and Christos Faloutsos. (2012). "GBASE: an efficient analysis platform for large graphs." **The VLDB Journal — The International Journal on Very Large Data Bases** 21, 5 (October):637–650.

[28] Mccreadie, Richard M. C., Craig Macdonald, and Iadh Ounis. (2009). "Comparing Distributed Indexing: To MapReduce or Not?" In **the 7th Workshop on Large-Scale Distributed Systems for Information Retrieval**.

[29] Lin, Jimmy, et al. (2009). "Of Ivory and Smurfs: Loxodontan MapReduce Experiments for Web Search." in **Text Retrieval Conference (TREC)**.

[30] Jianlong Zhong and Bingsheng He. (2013). "Medusa: Simplified Graph Processing on GPUs." **IEEE Transactions on Parallel and Distributed Systems** 99. 1: 1.

[31] W3C. (2000). **Semantic Web - XML2000 - slide 'Architecture.'** Accessed November 02,2013 Available from http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html.

[32] W3C. (2001). **Semantic Web Activity.** Accessed November 01, 2013 Available from http://www.w3.org/2001/12/semweb-fin/w3csw.

[33] W3C. (2007). **Semantic Web, and Other Technologies to Watch: January 2007 (24).** Accessed November 03, 2013 Available from http://www.w3.org/ 2007/Talks/ 0130-sb-W3CTechSemWeb/#(24).

[34] Horrocks, Ian. et al. (2005). "Semantic Web Architecture: Stack or Two Towers?" In **Principles and Practice of Semantic Web Reasoning 3703**, 37–41. Springer Berlin Heidelberg.

[35] W3C. (2005). **Representing Knowledge in the Semantic Web - slide 'The Semantic Web stack.'** Accessed November 02, 2013 Available from http://www.w 3c.it/talks/2005/openCulture/slide7-0.html.

[36] W3C. **Description of W3C Technology Stack.** Accessed November 02, 2013 Available from http://www.w3.org/Consortium/techstack-desc.html.

[37] W3C. **Semantic Web.** Accessed November 04, 2013 Available from

http://www.w3.org/standards/semanticweb/.

[38] **SPARQL Query Language for RDF.** Accessed November 05, 2013 Available from http://www.w3.org/TR/rdf-sparql-query/.

[39] **SPARQL 1.1 Update.** Accessed November 05, 2013 Available from http://www.w3.org/TR/2013/REC-sparql11-update-20130321/.

[40] Berners-Lee, Tim. (2009) **Linked Data.** Accessed June 18, 2011 Available from www.w3.org/DesignIssues/LinkedData.html.

[41] Auer, S. et al. (2007). "DBpedia: A Nucleus for a Web of Open Data." In **The Semantic Web 4825**. Busan, Korea: Springer Berlin Heidelberg. 722–735.

[42] Heath, Tom. (2007). **Linked Data - Connect Distributed Data across the Web.** Accessed November 01, 2013 Available from http://linkeddata.org/.

[43] **The Protégé Ontology Editor and Knowledge Acquisition System.** Accessed November 03, 2013 Available from http://protege.stanford.edu/.

[44] Mikel, Egaña, et al. (2008). "Applying Ontology Design Patterns in Bio-ontologies." Acitrezza, Italy. 7–16.

[45] The Apache Software Foundation. (2013). **Apache Jena - TDB**. Accessed November 10,2013. Available from http://jena.apache.org/documentation/tdb/.

[46] Owens, A., et al. (2009). "Clustered TDB: A Clustered Triple Store for Jena- ECS EPrints Repository." In **the WWW2009**.

[47] Ladwig, Günter, and Andreas Harth. (2011). "CumulusRDF: Linked data management on nested key-value stores." In **The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2011)**, 30.

[48] Atre, Medha, Jagannathan Srinivasan and James A. Hendler. (2009). **BitMat: A main memory RDF triple store**. Tetherless World Constellation, Rensselar Plytehcnic Institute, Troy NY.

[49] FernáNdez. Javier D., et al. (2013). "Binary RDF representation for publication and exchange (HDT)." **Web Semant.** 19. 22–41.

[50] Hozo. (2003). **Hozo-Ontology Editor-.** Accessed November 03, 2013 Available from http://www.hozo.jp/.

[51] Carroll, Jeremy J., et al. (2004). "Jena: implementing the semantic web recommendations." **In Proceedings of the 13th international World**

Wide Web conference on Alternate track papers & posters, 74–83.

[52] The Apache Software Foundation. (2011). **Apache Jena - Home**. Accessed June 17,2016. Available from https://jena.apache.org/.

[53] Cyganiak, Richard. (2005). **A relational algebra for SPARQL.** Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170: 35.

[54] Berners-Lee, Tim, et. al. (1998). "Uniform Resource Identifiers [RFC2396]." Accessed November 01,2013 Available from http://www.ietf.org/rfc /rfc2396.txt.

[55] Curé, Olivier, et al. (2014). "Waterfowl: A compact, self-indexed and inference-enabled immutable RDF store." In **European Semantic Web Conference**, 302-316. Springer International Publishing.

[56] Bock, Jürgen. "Parallel Computation Techniques for Ontology Reasoning." (2008). In **Proceedings of the 7th International Conference on The Semantic Web**, Karlsruhe, Germany. 901–906.

[57] Ramakrishna, Soma, and V.K. Prasanna. (2008). "Parallel Inferencing for OWL Knowledge Bases." In **the Proceedings of the 2008 37th International Conference on Parallel Processing,** 75-78.

[58] Owens, John D., et al. (2007). "A Survey of General-Purpose Computation on Graphics Hardware." **Computer Graphics Forum**. 26. 1. 80–113.

[59] Asanovic, Krste, et al. (2009). "A view of the parallel computing landscape." **Commun. ACM**. 52, 10: 56–67.

[60] Shuai Che et al. (2009). "Rodinia: A benchmark suite for heterogeneous computing." In **the IEEE International Symposium on Workload Characterization,** 44–54.

[61] Bingsheng He et al. (2008). "Mars: a MapReduce framework on graphics processors." In **Proceedings of the 17th international conference on Parallel architectures and compilation techniques**, 260-269. Toronto, Ontario, Canada.

[62] Volkov, Vasily, and James W. Demmel. (2008) "Benchmarking GPUs to tune dense linear algebra." In **International Conference for High Performance Computing, Networking, Storage and Analysis**, 1-11. Austin, Texas.

[63] Zechner, Mario, and Michael Granitzer. (2009). "Accelerating K-Means on the Graphics Processor via CUDA." In **First International Conference on**

Intensive Applications and Services, 7–15. Valencia, 20-25 April 2009.

[64] Savran, Ibrahim, and Jason D. Bakos. (2010). “GPU Acceleration of Near-Minimal Logic Minimization.” In **Proceedings of Symposium on Application Accelerators in High-Performance Computing.**

[65] Harish, Pawan, and P. J. Narayanan. (2007). "Accelerating large graph algorithms on the GPU using CUDA." In **International Conference on High-Performance Computing**, 197–208. Goa, India. Berlin Heidelberg:Springer.

[66] Melab, Nouredine, et al. (2012). “A GPU-accelerated Branch-and-Bound Algorithm for the Flow-Shop Scheduling Problem.” In **2012 IEEE International Conference on Cluster Computing**, 10-17.

[67] Rudomín, Isaac, Erik Millán and Benjamín Hernández. (2005). “Fragment shaders for agent animation using finite state machines.” **Simulation Modelling Practice and Theory**. 13. 8 (November): 741–751.

[68] **Alenka – SQL for CUDA.** Accessed June 26, 2011. Available from http://source forge.net/projects/alenka/files/.

[69] Bruno, Nicolas, Nick Koudas and Divesh Srivastava. (2002). “Holistic twig joins: optimal XML pattern matching.” In **Proceedings of the 2002 ACM SIGMOD international conference on Management of data**, 310–321. Madison, Wisconsin.

[70] Shnaiderman, L., and O. Shmueli. (2012). “A Parallel Twig Join Algorithm for XML Processing using a GPGPU.”

[71] Senn, Juerg. (2010). “Parallel Join Processing on Graphics Processors for the Resource Description Framework.” In **23rd International Conference on Architecture of Computing Systems (ARCS),** 1–8.

[72] **cudpp - CUDA Data Parallel Primitives Library - Google Project Hosting**. Accessed November 07, 2013. Available from http://code.google.com /p/cudpp/.

[73] Tripathy, A., S. Mohan and R. Mahapatra. (2011). “Optimizing a Semantic Comparator Using CUDA-enabled Graphics Hardware.” In **Fifth IEEE International Conference on the Semantic Computing (ICSC)**, 125–132.

[74] **Architecture Overview - Cassandra Wiki.** Accessed November 02, 2013 Available from http://wiki.apache.org/cassandra/ArchitectureOverview.

[75] Urbani, Jacopo, et al. (2012). "WebPIE: A web-scale parallel inference engine using MapReduce." **Web Semantics: Science, Services and Agents on**

the World Wide Web 10, (January): 59-75.

[76] Williams, Gregory Todd, et al. (2010). “Scalable reduction of large datasets to interesting subsets.” **Web Semantics: Science, Services and Agents on the World Wide Web** 8, 4 (November):365–373.

[77] ter Horst, Herman J. (2005). “Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary.” **Web Semantics: Science, Services and Agents on the World Wide Web** 3, 2 (October): 79–115.

[78] **OWL 2 Web Ontology Language Primer (Second Edition).** Accessed November 11, 2013 Available from http://www.w3.org/TR/owl2-primer/.

[79] Rohloff, Kurt, and Richard E. Schantz. (2010). “High-performance, massively scalable distributed systems using the MapReduce software framework: the SHARD triple-store.” In **Programming Support Innovations for Emerging Distributed Applications**, 1–5. Reno, Nevada.

[80] Husain, M., et al. (2009). “Heuristics Based Query Processing for Large RDF Graphs Using Cloud Computing.” **Knowledge and Data Engineering, IEEE Transactions on Knowledge and Data Engineering**. 1.

[81] Ding Li et al. (2004). "Swoogle: a search and metadata engine for the semantic web." In **Proceedings of the thirteenth ACM international conference on Information and knowledge management**.

[82] D’Aquin, M., and Motta, E. (2011). “Watson, more than a semantic web search engine.” **Semantic Web** 2. 1: 55–63.

[83] Oren, E., et al. (2008). “Sindice. com: a document-oriented lookup index for open linked data.” **Int. J. Metadata Semant. Ontol.** 3. 1: 37–52.

[84] Assel, M. et al. (2011). “MPI realization of high performance search for querying large RDF graphs using statistical semantics.” In **Proceedings of the 1st Workshop on High-Performance Computing for the Semantic Web (HPCSW2011)**, Greece.

[85] A Trellian Company. (2016). **Keyword Usage Statistics on the average number of keywords per search phrase by Country**. Accessed January 28,2016. Available from http://www.keyworddiscovery.com/keyword-stats.html?date=2016-01-01.

[86] Mario Arias Gallego et al. (2016). **HDT - Your binary format for RDF**. Accessed July 27,2015. Available from http://www.rdfhdt.org.

[87] Dave Beckett. (2016). **Raptor RDF Syntax Library**. Accessed July 27,2015. Available from http://librdf.org/raptor/.

[88] Dave Beckett. (2015). **Redland RDF Libraries**. Accessed July 27,2015. Available from http://librdf.org/.

[89] Jean-loup Gailly, and Mark Adler. (2015). **zlib Home Site**. Accessed July 27,2014. Available from http://www.zlib.net/.

[90] JCUDA. (2013). **Java bindings for CUDA**. Accessed January 27,2013. Available from http://www.jcuda.org/.

[91] Alan Kaminsky. (2012). **Parallel Java Library**. Accessed February 27,2012. Available from https://www.cs.rit.edu/~ark/pj.shtml.

[92] Andreas Wagner et al. (2013). **RDF store on a cloud-based architecture**. Accessed Available from https://github.com/cumulusrdf/cumulusrdf.

[93] W3C Semantic Web. (2014). **CumulusRDF - Semantic Web Standards**. Accessed May 19,2016. Available from https://www.w3.org/2001/sw/wiki/CumulusRDF.

[94] NVIDIA Corporation. (2013). **Thrust**. Accessed July 27,2016. Available from http://docs.nvidia.com/cuda/thrust/#axzz4FcCHo56b.

[95] NVIDIA Corporation. (2014). **CUSP: Main Page**. Accessed July 27,2014. Available from http://cusplibrary.github.io/.

[96] OpenMP. (2012). **The OpenMP® API specification for parallel programming**. Accessed July 27,2012. Available from http://openmp.org/wp/.

[97] NVIDIA Corporation. (2012). **Assess, Parallelize, Optimize, Deploy: Best Practices Guide :: CUDA Toolkit Documentation:** Accessed July 27,2012. Available from http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/#assess-parallelize-optimize-deploy.

[98] Udacity. (2014). **Introduction to Parallel Programming: Optimizing GPU Programs**. Accessed September 27,2015. Available from https://github.com/udacity/cs344.

[99] NVIDIA Corporation. (2012). **NVIDIA CUDA Compiler Driver NVCC**. Accessed July 27,2016. Available from http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/#axzz498vqA6Py.

[100] Kardam, Rahul Kumar. (2007). **Windows to UNIX porting, Part 1: Porting C/C++ sources**. Accessed July 27,2016. Available from http://www.ibm.com

/developerworks/aix/library/au-porting/.

[101] NVIDIA Corporation. (2015). **CUDA-GDB :: CUDA Toolkit Documentation**. Accessed July 27,2016. Available from http://docs.nvidia.com/cuda/cuda-gdb/#axzz498vqA6Py.

[102] NVIDIA Corporation. (2013). **NVIDIA Nsight Visual Studio Edition**. Accessed July 27,2016. Available from https://developer.nvidia.com/nvidia-nsight-visual-studio-edition.

[103] NVIDIA Corporation. (2012). **Nsight Eclipse Edition**. Accessed July 27,2016. Available from https://developer.nvidia.com/nsight-eclipse-edition.

[104] NVIDIA Corporation. (2013). **Profiler User's Guide**. Accessed July 27,2016. Available from http://docs.nvidia.com/cuda/profiler-users-guide/#axzz498vqA6Py.

[105] Choksuchat, C., and C. Chantrapornchai. (2013). "On the HDT with the Tree Representation for Large RDFs on GPU." In **Proceedings of the 2013 International Conference on Parallel and Distributed Systems**. 651–656.

[106] Choksuchat, C. et al. (2014). "Parallel health tourism information extraction and ontology storage." **11th Int. Jt. Conf. on Comput. Sci. Softw. Eng. JCSSE 2014**. 236–241.

[107] RDFHDT Team. (2013). **RDF HDT | Datasets**. Accessed July 28,2016. Available from http://www.rdfhdt.org/datasets/.

[108] Google. (2014). **Data Dumps - Freebase API — Google Developers**. Accessed September 14,2014. Available from https://developers.google.com/freebase/data.

[109] David Beckett. (2014). **RDF 1.1 N-Triples**. Accessed August 19,2014. Available from http://www.w3.org/TR/n-triples/.

[110] Erxleben, F. et al. (2014). "Introducing Wikidata to the Linked Data Web." In **The Semantic Web – ISWC 2014: 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I**. Springer International Publishing. 50–65.

[111] WMFLabs. (2015). **Wikidata RDF dumps 2015-07-27**. Accessed May 23,2016. Available from http://tools.wmflabs.org/wikidata-exports/rdf/exports/20150727/.

[112] Gavin Carothers. (2014). **RDF 1.1 N-Quads**. Accessed October 19,2014. Available

from http://www.w3.org/TR/n-quads/.

[113] Andreas Harth. (2012). **Billion Triples Challenge 2012 Data Set**. Accessed November 23,2014. Available from http://km.aifb.kit.edu/projects/btc-2012/.

[114] **GeoNames Ontology - Geo Semantic Web**. Accessed May 23,2014. Available from http://www.geonames.org/ontology/documentation.html.

[115] Unxos GmbH. (2012). **GeoNames**. Accessed April 27,2012. Available from http://download.geonames.org/.

[116] David Beckett et al. (2014). **RDF 1.1 Turtle**. Accessed July 12,2015. Available from http://www.w3.org/TR/turtle/.

[117] Mario Arias Gallego. (2013). **Geonames HDT Data Set**. Accessed January 28,2014. Available from http://gaia.infor.uva.es/hdt/geonames-11-11-2012.hdt.gz.

[118] Unified Medical Language System. (2012). **Medical Subject Headings - Summary | NCBO BioPortal**. Accessed July 28,2015. Available from https://bioportal.bioontology.org/ontologies/MESH.

[119] Jennifer Warrender. (2015). **Latest Index of Jennifer Warrender's karyotype**. Accessed July 28,2016. Available from http://homepages.cs.ncl.ac.uk /jennifer.warrender/karyotype/latest/.

[120] Warrender, J. D., and P. Lord. (2013). "The Karyotype Ontology: a computational representation for human cytogenetic patterns." **ArXiv Prepr. ArXiv1305 3758**.

[121] Bollacker, K. et al. (2008)."Freebase: a collaboratively created graph database for structuring human knowledge." In **Proceedings of the 2008 ACM SIGMOD international conference on Management of data**. 1247–1250.

[122] RDFHDT Team. (2013). **Wordnet3.1| HDT Data Set**. Accessed July 28,2013. Available from http://gaia.infor.uva.es/hdt/wordnet31.hdt.gz.

[123] RDFHDT Team. (2013). **Yago | HDT Data Set**. Accessed July 28,2013. Available from http://gaia.infor.uva.es/hdt/yago2s-2013-05-08.hdt.gz.

[124] RDFHDT Team. (2013). **DBLP | HDT Data Set**. Accessed February 14,2013. Available from http://gaia.infor.uva.es/hdt/dblp-2012-11-28.hdt.gz.

[125] RDFHDT Team. (2013). **SWDF | HDT Data Set**. Accessed February 14,2013. Available from http://gaia.infor.uva.es/hdt/swdf-2012-11-28.hdt.gz.

[126] Chantana Chantrapornchai, and Chidchanok Choksuchat. (2014). **Hua Hin Health Tourism Data Set**. Accessed May 24,2014. Available from https://datahub.io/dataset/hua-hin-health-tourism-data-set.

[127] Bizer, Chris, Richard Cyganiak, and Tom Heath. **How to publish Linked Data on the Web**. Accessed May 23,2016. Available from http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/LinkedDataTutorial/.

[128] Jeen. (2014). **RDFConvert**. Accessed July 28,2014. Available from https://sourceforge.net/projects/rdfconvert/.

[129] Gavin Carothers, and Andy Seaborne. (2014). **RDF 1.1 TriG**. Accessed April 20,2016. Available from https://www.w3.org/TR/trig/.

[130] Chantrapornchai, C. et al. (2016). "TripleID: A Low-Overhead Representation and Querying Using GPU for Large RDFs." In **Beyond Databases, Architectures and Structures. Advanced Technologies for Data Mining and Knowledge Discovery: 12th International Conference, BDAS 2016, Ustroń, Poland, May 31 - June 3, 2016, Proceedings**.Springer International Publishing. 400–415.

[131] Andreas Harth. (2009). **Billion Triple Challenge 2009 Dataset and Statistics**. Accessed March 22,2014. Available from https://km.aifb.kit.edu/projects/btc-2009/.

[132] Andreas Harth. (2009). **Billion Triples Challenge 2009 Data Set**. Accessed November 23,2012. Available from https://km.aifb.kit.edu/projects/btc-2009/btc-2009-small.nq.gz.

[133] Gromgull. (2012). **Some basic BTC2012 Stats**. Accessed July 28,2016. Available from http://gromgull.net/blog/2012/07/some-basic-btc2012-stats/.

[134]Chidchanok Choksuchat. (2015). **CBM_Combined-BitMap-Representation**. Accessed July 28,2016. Available from https://github.com/chidcha/CBM_Combined-BitMap-Representation.

[135] Chantana Chantrapornchai, Chidchanok Choksuchat, and Benjaporn Maneesang. (2014). **Hua Hin Health Tourism Data Set| OWL**. Accessed April 30,2014. Available from http://tourinthailand.org/ontology/healthtour.owl.

[136] RDFHDT Team. (2010). **Header-Dictionary-Triples Representation of RDF Program**. Accessed May 01,2013. Available from https://code.google.com/archive/p/hdt-it/downloads.

[137] Hwu, W. W. (2011). **GPU Computing Gems Jade Edition**. Morgan Kaufmann

Publishers Inc.

[138] Mei, X., and X. Chu. (2015). “Dissecting GPU Memory Hierarchy through Microbenchmarking.” **ArXiv Prepr. ArXiv150902308**.

[139] Chidchanok Choksuchat et al. (2015). "Accelerating Keyword Search for Big RDF Web Data on Many-Core Systems." In **14th International Conference on Intelligent Software Methodologies, Tools and Techniques**, Italy.

[140] Chidchanok Choksuchat. (2015). **Parallel_Search**. Accessed April 20,2015. Available from https://github.com/chidcha/Parallel_Search.

[141] NVIDIA® NVLink TM High-Speed. (2014). **NVIDIA® NVLink TM High-Speed Interconnect: Application Performance**. Accessed May 20,2015. Available from http://info.nvidianews.com/rs/nvidia/images/NVIDIA%20NVLink%20High-Speed%20Interconnect%20Application%20Performance%20Brief.pdf.

[142] Choksuchat, C., and C. Chantrapornchai. (2013). "Experimental framework for searching large RDF on GPUs based on key-value storage." In **10th International Joint Conference on Computer Science and Software Engineering (JCSSE 2013).** IEEE. 171–176.

[143] Choksuchat, C., and C. Chantrapornchai. (2013). “Large RDF representation framework for GPUs case study key-value storage and binary triple pattern.” In **Int Comput. Sci. Eng. Conf. ICSEC 2013.** 13–18.

[144] Choksuchat, C., and C. Chantrapornchai. (2012). “ON THE IMPROVEMENT OF SEARCH ENGINE USING SEMANTIC WEB: CASE STUDY OF HUA HIN TOURISM INFORMATION SYSTEM.” **ECTI TRANSACTIONS ON COMPUTER AND INFORMATION TECHNOLOGY**. 6. 2. 186–204.

[145] Chidchanok Choksuchat, and Chantana Chantrapornchai. (2016). “On the Development of Health Tourism Semantic Web with its Parallel Engine.” **Int. J. Metadata Semant. Ontol.** 11. 1. 16–28.

ภาคผนวก

ภาคผนวก ก

การประยุกต์ใช้การประมวลผลแบบขนาน

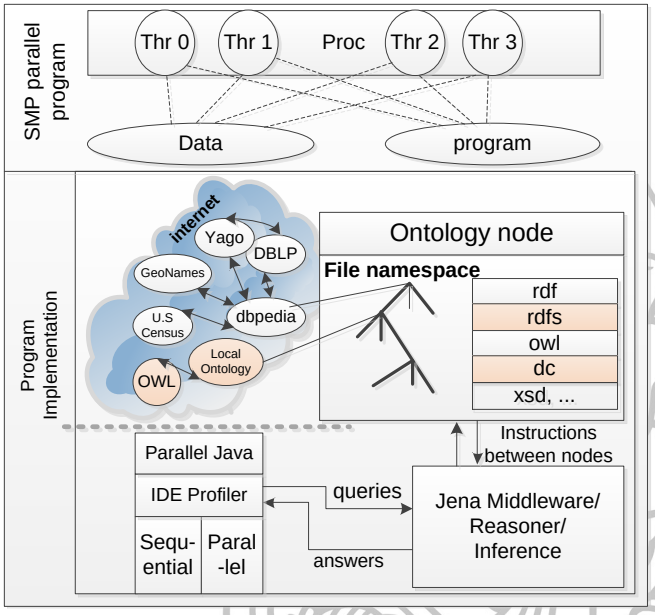
บทนี้นำเสนอการประยุกต์ใช้การประมวลแบบขนานในกรณีศึกษาที่หลากหลายตามด้านต่อไปนี้ การใช้งานจาวาแบบขนานเพื่อการคิวรี SPARQL บนเซตข้อมูล Linkded Open Data อย่าง DBpedia การทดสอบการค้นหาข้อมูลลอาร์ดีเอฟขนาดใหญ่บนฐานข้อมูลแบบคีย์-ค่า (key-value storage) บนหน่วยประมวลผลกราฟิกส์ และการดึงข้อมูลด้านการท่องเที่ยวเชิงสุขภาพและเก็บลงฐานความรู้ออนโทโลยีแบบขนาน

การใช้งานจาวาแบบขนานเพื่อการคิวรี SPARQL บนเซตข้อมูล DBpedia

วัตถุประสงค์การใช้ไลบรารี่ จาวาแบบขนาน (PJ: Parallel Java) เพื่อการคิวรี SPARQL บนเซตข้อมูล DBpedia คือการสร้างกระบวนการค้นหาข้อมูลเชิงความหมายแบบขนานบนหน่วยประมวลผลกลางเหมือนการใช้ OpenMP แต่เนื่องจากการคิวรีเว็บเชิงความหมายจำนวนมากมีมิดเดิลแวร์เป็นภาษาจาวา ผู้วิจัยจึงนำเสนอการประยุกต์ใช้ไลบราลี่ Parallel Java หรือ PJ ในการทำมัลติเทรดเพื่อคิวรีภาษา SPARQL แบบขนาน เพื่อความสะดวกในการเชื่อมต่อกับมิดเดิลแวร์ที่นักพัฒนานิยมเว็บเชิงความหมายใช้กัน [141] อยู่แล้วและสามารถนำไปประยุกต์ใช้งานกับการคิวรีทั่วไปได้ นอกจากนี้ยังถือโอกาสใช้การทดลองนี้เพื่อพิจารณาความสัมพันธ์ระหว่างไลบราลี่ PJ และ Concurrency Java และสุดท้ายเพื่อเปรียบเทียบเวลาที่ใช้ประมวลผลระหว่างการคิวรีแบบลำดับและแบบขนาน

จากรูปผนวก ก.1 แสดงสถาปัตยกรรมของระบบที่เริ่มจากส่วนของการพัฒนาโปรแกรม ผู้วิจัยใช้ Netbeans IDE ในการพัฒนาโค้ดและโพรไฟเลอร์ เพื่อตรวจสอบผลการทดลอง โดยแบ่งการทำงานของโปรแกรมออกเป็นสามส่วนหลัก ดังนี้ ส่วนการทำงานแบบลำดับ ส่วนการทำงานแบบขนานด้วย fixed schedule และ dynamic schedule ซึ่งคลาสแบบขนานเหล่านี้ทำงานบนไลบรารี่ PJ ทั้งสิ้น หลังจากเตรียมความพร้อมของเครื่องมือแล้ว ใช้แพกเกจคิวรี SPARQL ของอาปาเช่ เจนา เพื่อแปลงคำสั่งคิวรีระหว่างโหนดออนโทโลยี โดยสร้างเซอร์วิสของภาษา SPARQL เชื่อมกับฐานความรู้ DBpedia ที่มีการเชื่อมต่อไปอีกกับฐานความรู้อื่น ได้แก่ Yago DBLP Geonames U.S. Censes และออนโทโลยีที่เครื่องของนักวิจัย โดยอ้างถึงฐานความรู้เหล่านี้ด้วย URI ของเนมสเปซ ต่อมาเทรดและระบบจัดการงานหลายงานที่สร้างขึ้น แล้วส่งคำตอบในรูปแบบ XML ผ่าน API ของ

อาปาเช่ เจนา อีกครั้ง หลังจากนั้นจึงเก็บ คำตอบที่ได้รับมาในไฟล์เพื่อเตรียมจัดรูปแบบแสดงแก่ผู้ใช้งาน จากรูปผนวก ก.1 ส่วนล่างสุดคือเตรียมเครื่องมือเขียน ถัดขึ้นมาเป็นการจัดการโหนดออนโทโลยีด้วยการเชื่อมต่อหลายฐานความรู้ดังที่กล่าวมา และบนสุด เป็นการจัดการแตกเทรดด้วย PJ ให้ทำงานในส่วนของข้อมูลและโปรแกรมแบบขนานใน SMP (Shared Memory Multiprocessor)



รูปผนวก ก.1 สถาปัตยกรรมของระบบการใช้งานจาวาแบบขนานเพื่อการคิวรี SPARQL บนเซตข้อมูล DBpedia

การโปรแกรมด้วย PJ ประกอบด้วย งานหลักคือการประมวลผลคิวรี SPARQL ด้วยลูปของ PJ ซึ่งใช้พารามิเตอร์คือ จำนวนคิวรีที่ต้องการใช้ประมวลผล จำนวนชิ้นส่วนข้อมูลย่อย จำนวนของโพรเซสเซอร์ การใช้งานฟังก์ชันสำคัญมีดังนี้

ส่วนที่ 1 การใช้งานเทรดแบบขนาน (parallel thread team)

```
new ParallelTeam().execute (new ParallelRegion()..)
```

อธิบายได้ว่า `ParallelTeam()` ใช้หนึ่งเทรด ถ้าทดสอบด้วยการทำงานหลายเทรดให้ใช้ `ParallelTeam(n)` โดยที่ *n* เป็นจำนวนเทรดที่ต้องการแยก โดยทั่วไปมักไม่เกินจำนวนโพรเซสเซอร์ มากที่สุดควรเป็น จำนวนโพรเซสเซอร์ - 1

การใช้งาน `ParallelRegion()` เป็นการประกาศว่าโค้ดในส่วนนนี้เป็นการทำงานแบบขนาน

ส่วนที่ 2 การใช้งานเมธอด `run()`เพื่อประมวลงานแบบขนานเช่น `execute(0, n-1, new IntegerLoop())`

ส่วนที่ 3 การทำงานของ schedule นั้น ผู้วิจัยใช้ dynamic loop เพื่อทำ load balancing แบบอัตโนมัติ โดยลูปที่ PJ มีให้เรียกใช้จะเหมือนกับ OpenMP ที่มีสามแบบคือ แบบ fixed แบบdynamic และแบบ guided

กรณี *fixed schedule* ไลบรารี่ PJ จัดการแบ่งเซตของลูป *n* ทำงานแบบ iteration กับชิ้นส่วนไฟล์ย่อยจำนวน *k* ชิ้น โดยกำหนดให้งานย่อยหนึ่งชิ้นงานรันแต่ละเทรด กรณี *dynamic schedule* ไลบรารี่ PJ จัดการแบ่งเซตของลูป iteration แก่ชิ้นงาน โดยพิจารณาตามขนาดของชิ้นงานด้วย กรณี *guided schedule* ไลบรารี่ PJ จัดการแบ่งเซตของลูป iteration แก่ชิ้นงาน โดยพิจารณาชิ้นงานแรกใหญ่สุดแล้วเรียงลำดับไปหาชิ้นงานที่เล็กกว่าไปเรื่อย ๆ

ส่วนที่ 4 การใช้งานแบบขนานของลูป *for* ในส่วนงานหลักนั้น ลำดับดัชนีของการคิวรี่จะถูกใช้เป็นค่าเริ่มต้นของการทำงานแบบขนาน สำหรับตัวแปรที่ใช้ร่วมกัน ซึ่งประกาศใช้งานนอกเขตของการทำงานแบบขนานในเวลารันงาน

ส่วนที่ 5 ตั้งค่าในแต่ละเทรดด้วย Pseudorandom number generator (PRNG) และตัวนับ

ส่วนที่ 6 ตั้งค่าขยายระยะในแคช เพื่อให้แต่ละเทรดสามารถเข้าถึงตัวแปรในเทรดของตัวเองได้ ตัวอย่างการตั้งค่าเช่น

Long pad0,pad1,pad2,pad3,pad4,pad5,pad6,pad7;

Long pad8,pad9,pada,padb,padc,padd,pade,padf;

เพื่อเปรียบเทียบประสิทธิภาพของการทำงานแบบลำดับการคิวรีแบบขนานด้วย PJ ในการค้นหาของเว็บเชิงความหมาย ในกรณีของการคิวรีแบบลำดับ ผู้วิจัยใช้เวอร์วิส SPARQL จากแพกเกจของอาปาเช่ เจนา ในการเชื่อมต่อกับ DBpedia เพื่อคิวรี่ 6 คิวรี่ตามลำดับ ในการทดสอบ จะแยกออกเป็น สองกลุ่มต่อไปนี้ และสามารถดูที่มาและรายละเอียดการเชื่อมต่อในตารางที่ 6.2

กลุ่มแรก เซตข้อมูล DBpedia ประกอบด้วยลิงค์เชื่อมต่อไปยังเว็บอื่น ๆ และมีการเชื่อมต่อไปยังเซตข้อมูลอาร์ดีเอฟอื่น ด้วยความสัมพันธ์ “owl:sameAs” ที่เป็นการแสดงความหมายที่เซตข้อมูล DBpedia สื่อถึงเซตข้อมูลภายนอกได้แก่ การเชื่อมต่อกับข้อมูลจากภูมิศาสตร์กับ geonames.org การเชื่อมต่อกับผู้เขียนและหนังสือจากฐานอาร์ดีเอฟในโครงการกูเตนเบิร์ก การ

เชื่อมต่อกับข้อมูลการตีพิมพ์ด้านวิทยาศาสตร์ DBLP การเชื่อมต่อกับข้อมูลแต่ละรัฐของสหรัฐอเมริกาในฐานข้อมูลสถิติของ U.S. census

กลุ่มที่ 2 เป็นข้อมูลพิกัดทางภูมิศาสตร์ ภายใน DBpedia ที่มีสถานที่จำนวน 986,000 โดยการใช้เซตคำศัพท์ W3C Basic Geo ตัวอย่างการค้นหาคือ “show me all things next to the places like Eiffel Tower and Brandenburg Gate”

ในการพัฒนาโปรแกรมแบบขนาน ได้ทำการทดสอบกับ schedule สองแบบคือ fixed สำหรับทดสอบแบบไม่มีการ load balance และแบบ dynamic เพื่อทดสอบการ load balance งานจำนวน *n* ชิ้น

ประเด็นสำคัญคือจำนวนของเทรดและขนาดของไฟล์ย่อยที่ใช้ในเวลารันงาน จำนวนคิวรีที่มีคือ 6 คิวรี จำนวนงานและจำนวนเทรด มีค่า ตั้งแต่ {2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096} แบนด์วิธของเครือข่ายคือ 256 กิโลไบต์ต่อวินาที รันงานจนผลลัพธ์คือเวลาค่อนข้างคงที่ โดยรันครบรอบการทำงานครั้งหนึ่งคือ 12*6*12 = 864 ครั้ง เมื่อได้เวลามาของแต่ละคิวรี t_1,t_2, ..., t_6 เก็บไว้ เนื่องจากมีเวลาอื่นด้วย (เช่นการเชื่อมต่อเครือข่าย) ตารางผนวก ก.1 แสดงรายละเอียดการทดลองการใช้งานจาวาแบบขนานเพื่อการคิวรี SPARQL บน DBpedia

ตารางผนวก ก.1 รายละเอียดการทดลองการใช้งานจาวาแบบขนานเพื่อการคิวรี SPARQL บน DBpedia

Input Parameters:
Query Parameter: 6 ; {Q1,Q2,Q3,Q4,Q5,Q6}
N chunk size: {2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096}
Process: Dynamic Scheduler by 12 value Threads.
Output Parameters:

Threads	Q1	Q2	Q3	Q4	Q5	Q6	$\sum$Query Time	Total Time	Other Time
{2, 4, 8, 16, 32, 64, 128, 256,512, 1024,2048, 4096}	t_1	t_2	..	..	..	t_6	A from $\sum_{i=1 \text{to } 6} Qt_i$	B	B-A

คิวรีทั้ง 6 คิวรีที่กล่าวถึงมีดังนี้

พรีฟิกซ์ที่ใช้

```
owl: <http://www.w3.org/2002/07/owl#>
```

```
xsd:<http://www.w3.org/2001/XMLSchema#>
rdfs: <http://www.w3.org/2000/01/rdf-schema#>
rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
foaf: <http://xmlns.com/foaf/0.1/>
dc: <http://purl.org/dc/elements/1.1/>
: <http://dbpedia.org/resource/>
dbpedia2: <http://dbpedia.org/property/>
dbpedia: <http://dbpedia.org/>
skos:<http://www.w3.org/2004/02/skos/core#>
geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
```

คิวรีที่ใช้ในการทดลอง

```
1) SELECT * WHERE
{?subject owl:sameAs ?link.
 ?subject rdf:type <http://dbpedia.org/ontology/Country>} LIMIT 1000

2) SELECT * WHERE
{?subject owl:sameAs ?link.
 ?subject rdf:type <http://dbpedia.org/ontology/Writer> } LIMIT 1000

3) SELECT * WHERE
{?subject owl:sameAs ?link.
 ?subject rdf:type <http://dbpedia.org/class/yago/Programmer110481268>
.} LIMIT 1000


4) SELECT * WHERE
{?subject owl:sameAs ?link.
 ?subject <http://purl.org/dc/terms/subject>
<http://dbpedia.org/resource/Category:States_of_the_United_States>
.} LIMIT 1000


5) SELECT ?subject ?label ?lat ?long WHERE
{<http://dbpedia.org/resource/Eiffel_Tower>  geo:lat ?eiffelLat.
 <http://dbpedia.org/resource/Eiffel_Tower>  geo:long ?eiffelLong.
 ?subject geo:lat ?lat.
 ?subject geo:long ?long.
 ?subject rdfs:label ?label.
  FILTER(?lat - ?eiffelLat <= 0.05 && ?eiffelLat - ?lat <= 0.05 &&
  ?long - ?eiffelLong <= 0.05 && ?eiffelLong - ?long <= 0.05 &&
  lang(?label) = "en" ). } LIMIT 20


6) SELECT ?subject ?label ?lat ?long WHERE
{<http://dbpedia.org/resource/Brandenburg_Gate> geo:lat
      ?brandenburgLat.
 <http://dbpedia.org/resource/Brandenburg_Gate> geo:long
      ?brandenburgLong.
 ?subject geo:lat ?lat.
 ?subject geo:long ?long.
 ?subject rdfs:label ?label.
 FILTER(?lat - ?brandenburgLat <= 0.05 &&
```

```
?brandenburgLat - ?lat <=  0.05 &&
?long - ?brandenburgLong <= 0.05 &&
?brandenburgLong - ?long <= 0.05 &&
lang(?label) = "en" ). } LIMIT 20
```

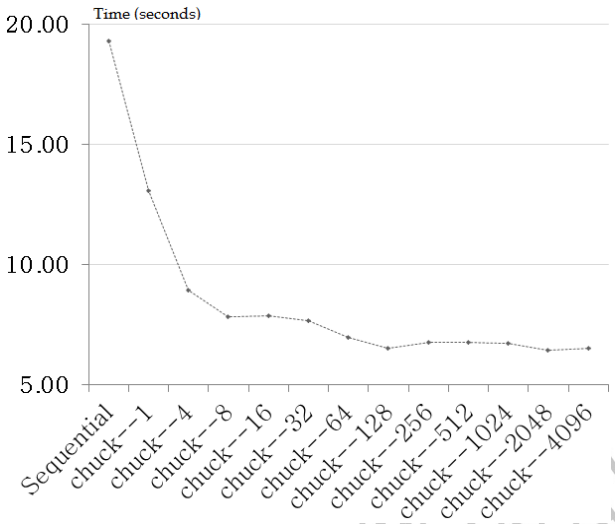
เครื่องที่ใช้ Intel® Core™ i52400 CPU @ 3.10GHz 3.40 GHz, 4 แกน หน่วยความจำหลัก 8 กิกะไบต์และใช้ JDK 1.7 ความเร็วของเครือข่ายไวไฟไร้สายไม่ต่ำกว่า 256กิโลบิตต่อวินาที (ปี 2012) ข้อมูลและรายละเอียดการเชื่อมต่อของ DBpedia กับเซตข้อมูลอื่น ๆ และคำตอบที่ได้จากการทดสอบเป็นไปดังตารางผนวก ก.2

ตารางผนวก ก.2 เซตข้อมูลและรายละเอียดการเชื่อมต่อกับ DBpedia

เซตข้อมูลที่ได้จากเทคโนโลยีลิงค์เดต้า/แหล่ง	จำนวนของทริพเพิล	ประเภทการเชื่อมหรือแท็กที่เชื่อม	คำตอบ
GeoNames (https://datahub.io/dataset/geonames)	93,896,732	Owl:sameAs	1,000
DBLP (https://datahub.io/dataset/fu-berlin-dblp)	28,000,000	Owl:sameAs	1,000
Rdfabout (https://datahub.io/dataset/sec-rdfabout)	1,800,000	Owl:sameAs	1,000
Geo-Coordinate ใน DBpedia	986,000 (Locations)	ทุกสิ่งที่อยู่ใกล้เคียง	20
project-gutenberg (https://datahub.io/dataset/fu-berlin-project-gutenberg)	10,000	Owl:sameAs	1,000

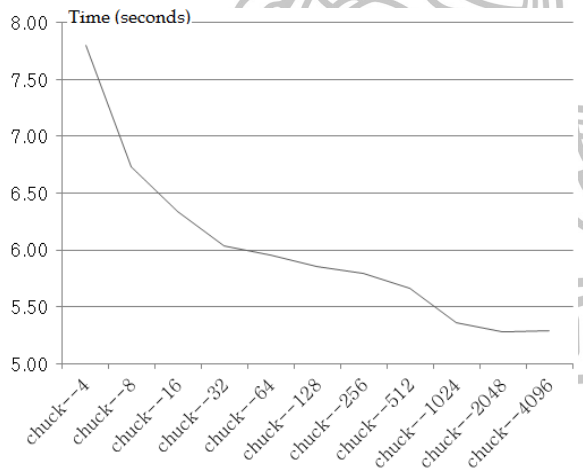
จากตารางผนวก ก.2 คอลัมน์แรกคือเซตข้อมูลที่ได้จากเทคโนโลยีลิงค์เดต้าและแหล่งที่ใช้ดาวน์โหลด คอลัมน์ที่ 2 คือจำนวนของทริพเพิลในเซตข้อมูลนั้น ๆ คอลัมน์ที่ 3 คือพร็อพเพอร์ตี้ของ owl ที่ใช้บอกความสัมพันธ์ที่คล้ายคลึงกัน ใช้เชื่อมต่อระหว่างเซตข้อมูลได้ ยกเว้นกลุ่มที่ 2 คือ Geo-Coordinate ใน DBpedia ที่ใช้การค้นหาพิกัดด้วยการ FILTER พิกัดที่ใกล้เคียงสถานที่สำคัญ และคอลัมน์สุดท้ายคือจำนวนคำตอบ ซึ่งผู้วิจัยกำหนดจำนวนไว้ในคิวรีข้างต้น

การทดสอบนี้ได้ผลลัพธ์เป็นเวลาที่ใช้ทำงานคิวรีแบบลำดับและแบบขนาน *fixed schedule* ดังรูปผนวก ก.2



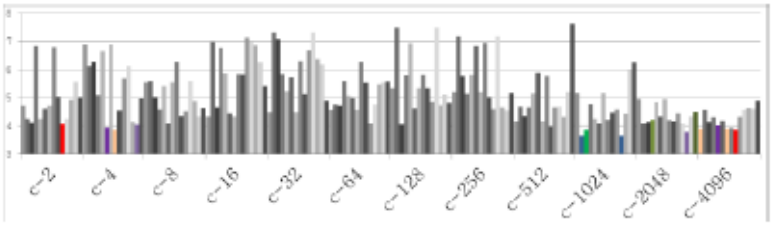
รูปผนวก ก.2 ผลลัพธ์ของเวลาที่ใช้ในการทำงาน (แกน y) ระหว่างคิวรีแบบลำดับและแบบขนาน *fixed schedule*

จากรูปผนวก ก.2 แบ่งออกเป็น 2 กรณี ได้แก่ กรณีแรก การเปรียบเทียบระหว่างการคิวรีแบบลำดับและการคิวรีแบบขนานที่ชิ้นส่วนไฟล์ 1 ส่วน ซึ่งวิธีคิวรีแบบลำดับมีค่า 19 วินาที ขณะที่คิวรีแบบขนานมีค่า 13 วินาที กรณีที่ 2 เมื่อแยกไฟล์เป็นส่วนย่อย 4-4096 ชิ้นส่วน เวลาลดลงเหลือ 8.9 วินาที

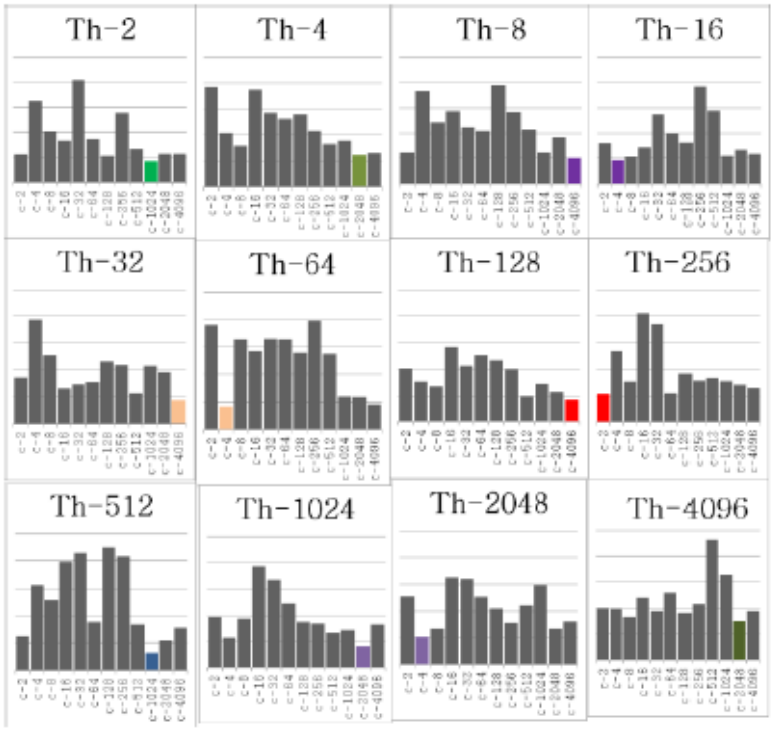


รูปผนวก ก.3 ผลลัพธ์ที่ได้จากการรันงาน *dynamic schedule*

จากรูปผนวก ก.3 ผลลัพธ์ของการทำงานแบบขนาน แบบ *dynamic schedule* ที่เริ่มจาก *dynamic (4) ถึง dynamic (4096)* ผลลัพธ์ที่ได้ช่วงแรกดูคล้ายกับ *fixed schedule* แต่ในส่วนตั้งแต่ *dynamic (16)* เริ่มมี load balance ที่ดี เวลาที่ใช้จึงเริ่มลดลงกว่า *fixed(16)*



(a).



(b).

รูปผนวก ก. 4 ผลลัพธ์จากการคิวรีทั้งหมดแยกตามจำนวนเทรดและจำนวนชิ้นไฟล์ย่อย (a) ภาพรวมของเวลาที่ใช้ในการคิวรี ที่แยกตามกลุ่มของชิ้นส่วนไฟล์ (b) ส่วนนี้แยกตามเทรด ทั้งสองภาพแกน y มีเวลาระหว่าง 3-7.6 วินาที

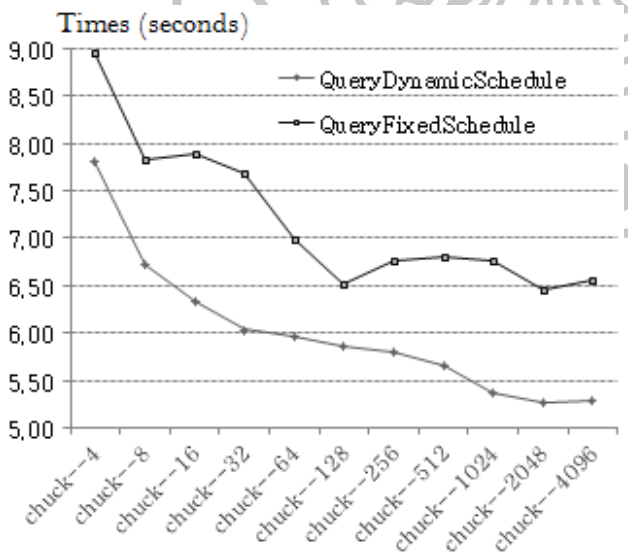
จากรูปผนวก ก.4 แสดงผลลัพธ์จากการคิวรีทั้งหมดแยกตามจำนวนเทรดและจำนวนชิ้นไฟล์ย่อย (a) ภาพรวมของเวลาที่ใช้ในการคิวรี ที่แยกตามกลุ่มของชิ้นส่วนไฟล์ (b) ส่วนนี้แยกตามเทรด ทั้งสองภาพแกน y มีเวลาระหว่าง 3-7.6 วินาที จากทั้งสองส่วนสามารถสรุปได้ว่าเวลาที่ดีที่สุดของทุกเทรดจะมีค่าไม่ตรงกัน ยกตัวอย่าง จากสมรรถนะเครื่องนั้นการแบ่งงานเป็น 2 เทรดสามารถหาเวลาที่ดีที่สุดได้จากการแบ่งเป็น 1024 ชิ้นไฟล์ย่อย แต่อย่างไรก็ตามการทดสอบผ่านเครือข่ายมีปัจจัยรบกวนนั่นคือเวลาที่ใช้ในการเชื่อมต่อเครือข่ายที่เราต้องกำจัดออกตามตารางผนวกที่ ก.1 ได้ผลลัพธ์ดังตารางผนวกที่ ก.3 พบว่าในความจริงการแบ่งงานเป็น 2 เทรดสามารถหาเวลาที่ดีที่สุดได้

จาก 128 ชิ้นไฟล์ย่อย ในกรณีที่มีเครือข่ายความเร็วสูงผู้วิจัยคาดว่าจะทำให้เวลาที่ใช้ในการคิวรีแบบขนานผ่านเครือข่ายสามารถทำงานได้ดียิ่งขึ้น

ตารางผนวก ก.3 สรุปเปอร์เซนต์ของเวลาที่ใช้ในการคิวรี

จำนวนเทรด	จำนวนชิ้นไฟล์ย่อย	%เวลาที่ใช้ในการคิวรี	%เวลาอื่น ๆ
2	128	3.00	97.00
4	4	2.84	97.16
8	2	3.81	96.19
16	4	2.60	97.40
32	32	3.48	96.52
64	4	2.52	97.48
128	8	2.47	97.53
256	256	2.94	97.06
512	2	2.97	97.03
1024	2048	4.65	95.35
2048	256	3.34	96.66
4096	8	3.24	96.76

การอภิปรายผลการทดลองที่ผ่านมานั้นจากส่วนแรก การทดสอบ load balance ระหว่างกรณี *fixed* และ *dynamic schedule* พบว่าเป็นดังรูปผนวกที่ ก.5 *dynamic schedule* ลดเวลาคิวรีให้อยู่ในระหว่าง 5-8 วินาที เป็นช่วงที่ลดลงจาก *fixed schedule* ที่อยู่ที่ 6.5-9 วินาที



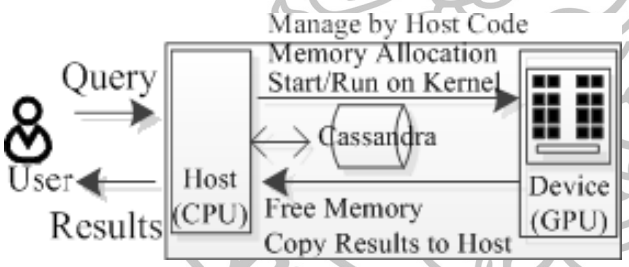
รูปผนวก ก. 5 เปรียบเทียบเวลารันคิวรีระหว่าง *fixed* และ *dynamic schedule*

ภาคผนวก ข

การประยุกต์ใช้งานการค้นหาอาร์ดีเอฟขนาดใหญ่ใน ฐานข้อมูลแบบคีย์-ค่าบนหน่วยประมวลผลกราฟิกส์

การนำข้อมูลอาร์ดีเอฟใส่ในฐานข้อมูลคีย์-ค่าแบบอปาเช คาสซานดรา แล้วนำคีย์-ค่าจากฐานข้อมูลไปประมวลผลคิวรี่บนหน่วยประมวลผลกราฟิกส์ การประยุกต์นี้เกิดขึ้นด้วยวัตถุประสงค์ ความต้องการใช้ฐานข้อมูลที่เหมาะสมกับการทำงานแบบขนานและต้องสามารถกระจายตัวทำงานแบบขนานได้ด้วย นอกจากนั้นผู้วิจัยตั้งสมมุติฐานไว้ว่าการเก็บข้อมูลลงฐานข้อมูลนั้นสามารถลดขนาดข้อมูลได้ส่วนหนึ่ง ซึ่งการใช้งานฐานข้อมูลคาสซานดราที่มีใช้ในการเก็บข้อมูลขนาดใหญ่แบบกระจาย อาจเป็นคำตอบในการนำข้อมูลแบบอาร์ดีเอฟมาประมวลผลคิวรีแบบขนานบนหน่วยประมวลผลกราฟิกส์ได้

ไลบรารี่ที่เกี่ยวข้องประกอบด้วย CumulusRDF เป็นแพลตฟอร์มสำหรับเก็บอาร์ดีเอฟแบบคีย์-ค่าที่สามารถประจายตัวบนคลาวด์มีภาษาคิวรีคือ CQL อาปาเช่ เจนา มิดเดิลแวร์เพื่อการคิวรี่อาร์ดีเอฟด้วยภาษา SPARQL และเจคูด้า สามารถเรียกใช้งานคูด้าโดยนักพัฒนาภาษาจาวาได้

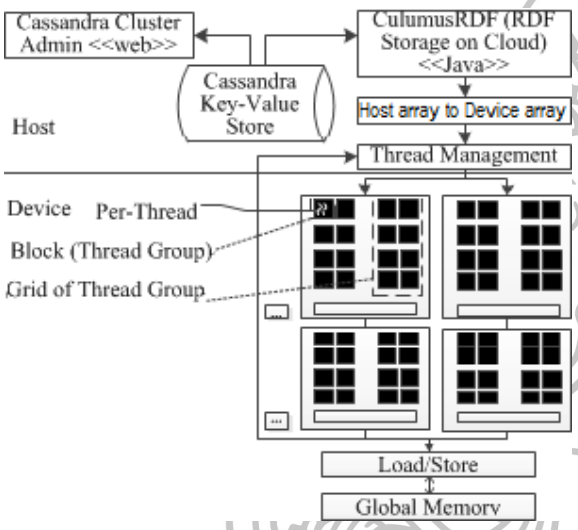


รูปผนวก ข.1 ภาพรวมของการคิวรีอาร์ดีเอฟในฐานข้อมูลคาสซานดราบนหน่วยประมวลผลกราฟิกส์

จากรูปผนวก ข.1 ภาพรวมของการคิวรีอาร์ดีเอฟในฐานข้อมูลคาสซานดราบนหน่วยประมวล ผลกราฟิกส์ เริ่มจากระบบรับคิวรีจากผู้ใช้งาน แล้วเริ่มจองหน่วยความจำในโฮสต์และดีไวส์ จากนั้นเริ่มเรียกใช้งานเคอร์เนลล์จากฝั่งโฮสต์ ขอ้มูลเข้าได้แก่คีย์จากคาสซานดรา แล้วจึงถูกแบ่งเป็นชิ้นย่อยเพื่อจับคูในการคิวรี่ภายในหน่วยประมวลผลกราฟิกส์ หลังจากหาคำตอบของคิวรีได้แล้ว เก็บค่าไว้แล้วส่งกลับมาที่โฮสต์ด้วยการหาค่าที่ผูกกับคีย์นั้นอยู่ แล้วจึงแสดงค่านั้นออกมาแก่ผู้ใช้

รูปผนวก ข.2 เป็นรายละเอียดการติดตั้งและใช้งานโปรแกรม สังเกตว่าการใช้งานอาปาเช่ คาสซานดรานี้ใช้ผ่านเว็บเบราเซอร์สำหรับการนำเข้าอาร์ดีเอฟลงสู่ฐานข้อมูลแบบคีย์-ค่า แต่ละทริพเพิลจะได้รับคีย์หนึ่ง ตัวอย่างจากงานวิจัยนี้ข้อมูลรับเข้าเป็น N-Quad ที่มี <s p o c> โดย c เป็น context ขยายทริพเพิลเมื่อนำเข้าสู่อาปาเช่คาสซานดรา จะถูกตัดออกแล้วแบ่งเป็นสามรูปแบบได้แก่

{subject predicate object หรือ SPO, predicate object subject หรือ POS, object subject predicate หรือ OSP} โดยแต่ละแบบสามารถคิวรีได้ 8 แบบ ดังนี้ (? ? ?) ใช้ได้ทั้งหมด, (s p o) (s p ?) (s ? ?) ใช้ SPO, (? p o) (? p ?) ใช้ POS และ (s ? o) และ (? ? o) ใช้ OSP โดยผู้วิจัยได้เลือกใช้งาน *flat layout* เนื่องจากเป็นรูปแบบมาตรฐานของฐานข้อมูลแบบคีย์ค่าและมีประสิทธิภาพที่ดีกว่า *hierarchy layout* ข้อดีคือทุกค่าในตารางผ่านการเรียงตามคีย์แล้ว พรีฟิกซ์จะเรียกได้ตามคีย์ของคอลัมน์



รูปผนวก ข.2 รายละเอียดการพัฒนาโปรแกรมและชั้นของหน่วยความจำในหน่วยประมวลผลกราฟิกส์

กรณีทริพเพิล (s p o) เก็บเป็น {s: {po:-} } แปลว่า s มีค่าเป็นคีย์ของแถว p เป็นคีย์ของคอลัมน์และ o เป็นค่า ถ้าเก็บแบบ {s:{p:{o}}} จะทำให้คีย์ของคอลัมน์มีค่าซ้ำซ้อน เพราะหมายถึง predicate เดียวกัน ถูกเพิ่มลงใน subject หลายต่อหลายครั้ง ดังนั้นคาสซานดราจะทำหน้าที่จัดการดัชนีในระดับ low-level ทั้งหมด ดังนั้นในกรณี OSP จึงถูกเก็บแบบ {o:{sp:-}}

การแปลงข้อมูลสตริงของ URI ในกรณีของ (SPO, OSP) สามารถใช้คุณสมบัติของ HTTP redirect ที่มีต่อ URI ได้ มีผลกับการจอยน์ข้อมูล เช่น (?s p1 <uri>)(<uri> p2 ?o) เหตุการณ์นี้เกิดขึ้นเมื่อ object เป็น URI เนื่องจากการโปรเจคชัน ของ CQL แตกต่างจาก SQL ที่ไม่มีการรับรองว่าผลลัพธ์จะประกอบด้วยคีย์ของคอลัมน์ที่ไม่มีดัชนีเทียบกับตาราง OSP ในคอลัมน์ O-S เมื่อข้อมูลมีปริมาณมาก จะใช้เวลานานในการจอยน์ S-O ดังนั้นในการประมวลผลแบบขนาน ผู้วิจัยดึงคีย์ขึ้นมาหนึ่งคู่ เพื่อจับคู่กันบนหน่วยประมวลผลกราฟิกส์เท่านั้น

ตารางผนวก ข.1 เซตข้อมูลที่ผ่านการนับค่าจากวิธี external sorting

Distinct name	Freebase	DBpedia0	DBpedia1
Triples	101,241,280	100,000,000	98,090,024
Subject	22,826,967	12,031,393	12,158,885
Predicates	11,118	30,582	30,759
rdf:type	8,436,113	8,599,542	8,141,595
Objects	44,671,587	16,835,432	17,976,396

การเรียงข้อมูลที่มีขนาดใหญ่เกินกว่าขนาดของหน่วยความจำ ในกรณีที่เครื่องมีประสิทธิภาพด้อยการใช้วิธีเรียงแล้วเก็บข้อมูลลงดิสก์ทีละส่วน (external sorting) แทนการเก็บลงหน่วยความจำสามารถช่วยได้ดังตารางผนวก ข.1 อีกคีย์หนึ่งที่มีความสำคัญคือ POS จากตารางแสดงว่า predicate เป็นข้อมูลที่มีการแชร์กับส่วนอื่น ๆ มากที่สุด จึงมีจำนวนน้อยที่สุด หนึ่งในแท็กที่มีการใช้งานกับ predicate มากที่สุดคือ *rdf:type* แพกเกจ CumulusRDF ใช้ PO เป็นคีย์ของแถว ในดัชนี POS ปกติเก็บเป็น *{po:{s:-}}* นั่นคือการที่ผูก PO ไว้ด้วยกันเสมือนมี S ผูกด้วยค่า P เดียว แต่จะทำให้ข้อมูลมีการแตกค่าตามแถวน้องลง แต่เพื่อความสะดวกในการค้นหา ทำให้ต้องมีการตั้งดัชนีรองขึ้นมาเก็บเป็น *{po: {'p': p}}* ข้อดีคือสามารถคิวรีได้ทั้ง (? p o) (? p ?)

เมื่อเปรียบเทียบระหว่าง CQL และฐานข้อมูลเชิงสัมพันธ์ที่ใช้กันอย่างแพร่หลาย column-family-key เทียบได้กับตาราง และคู่ลำดับคีย์-ค่า ที่ใช้เทียบได้กับแถว ส่วนคีย์ของแถวเทียบได้กับไพรมารีคีย์นั่นเอง

เริ่มการแปลงข้อมูลโดยการใช้ CumulusRDF โดยแพกเกจนี้ทำการสร้างคีย์แฮชเพื่อให้ค้นหา ผู้วิจัยใช้เวลาในส่วนนี้จัดการ 300 ล้านทริพเพิล ในเวลา 70 ชั่วโมง เพื่อสร้างคีย์-ค่าขึ้นมา

ผู้วิจัยทำการแปลงข้อมูล 3 คีย์สเปซ คือ Freebase 1 ชุด และ DBpedia จำนวน 2 ชุด ข้อจำกัดของอาปาเช่ คาสซานดราคือทั้งคีย์และชื่อคอลัมน์ต้องมีขนาดไม่เกิน 64 กิโลไบต์ เมื่อวัดค่าการกำจัดออกของคีย์เกินขณะสร้างจำนวน 40 ทริพเพิลจาก 300 ล้านทริพเพิลนับว่าเป็นจำนวนที่น้อยมาก

การดึงข้อมูลจากอาปาเช่ คาสซานดรา โดยใช้ CQL สำหรับคลัสเตอร์เดียวกันที่ใช้คาสซานดรา แต่ละตารางที่สร้างจะเป็นไปตามคุณสมบัติของการเก็บคีย์อย่างเดียวลงสู่แคช (http://cassandra.apache.org/doc/cql3/CQL.html) ตั้งขนาด heap คือ 1037959168/1037959168 ส่วนหน่วยความจำโกลบอลคือ 329 เมกะไบต์ คีย์แคชเริ่มที่ 49 เมกะไบต์ เก็บครั้งละ 14,400 วินาที

การสร้างบล็อกในหน่วยประมวลผลกราฟิกส์ ด้วยการส่งคีย์เข้าไป มีสองแบบคือ แบบแรก การส่งค่าอะเรย์ 1 มิติ (static) ที่ประกอบด้วย ขนาดของข้อมูล ความยาวคีย์ อะเรย์ของคีย์เข้า ค่าผลลัพธ์ ดัชนี และออฟเซต แบบที่สองคือการส่งพอยน์เตอร์-พอยน์เตอร์ (dynamic) เข้าสู่หน่วยประมวลผลกราฟิกส์ ดังนั้นข้อมูลที่ส่งเข้าไปมี ขนาดของคีย์ พอยน์เตอร์ของอินพุต และเอาต์พุต และความยาวคีย์ อ่านรายละเอียดการส่งค่าในโปรแกรมได้ที่ [142]

การสร้างโปรแกรมจาวา เชื่อมคาสซานดราและทริฟต์ (Thrift) เจคูด้าจะทำหน้าที่เชื่อม CQL จากคาสซานดรากับคูด้าโดยเรียกใช้ไฟล์ PTX การจับคู่ของคีย์เป็นดังอัลกอริทึมผนวก ข.1 การจับคู่คีย์มีสองแบบคือ แบบสตริงและ UUID ซึ่งเป็นค่านี้เป็นคีย์แถวของ POS ซึ่งผู้วิจัยไม่ได้ใช้ในขั้นตอนนี้ งานวิจัยนี้สนใจการจอยน์ของ SPO และ OSP ดังนั้น จึงใช้แทนค่าของสับสตริงและข้อมูลสตริงตามลำดับ การใช้อัลกอริทึมผนวก ข.1 นั้นใช้หน่วยความจำคอนสแตนซ์และหน่วยความจำแชร์ร่วมด้วย ขนาดของบล็อกมีค่าเท่า *N* เทรด แต่ละ *N* เทรดสามารถสแกนหาเพื่อจับคู่คีย์แบบขนานแล้วเก็บขึ้นเพื่อปรับค่าดัชนี อัลกอริทึมผนวก ข.2 แบบ Horspool (อ่านเพิ่มเติมเรื่องสาเหตุการใช้งาน งานวิจัยที่เกี่ยวข้องได้ที่ [142] และ https://code.google.com/p/exactstrmatchgpu/) ใช้วินโดว์ขนาด *m* ในการค้นหาข้อมูลจากขวาไปซ้ายและ pre-compute array ของคาแรคเตอร์ขวาสุดของวินโดว์เพื่อข้ามตำแหน่งที่หาไม่เจอไป ที่เลือกใช้วิธีนี้เพราะข้อมูลส่วนใหญ่เป็น URI ที่ด้านซ้ายมีค่า URL ที่ซ้ำกันจำนวนมากมาย ดังนั้นด้านขวามือจึงเป็นชื่อเว็บเพจ ซึ่งทำการกระโดดข้ามการซ้ำกันของ URL แล้วเปรียบเทียบเฉพาะชื่อส่วนนี้ด้วย Horspool ได้ เครื่องที่ใช้ทดสอบดังตารางผนวก ข.2

อัลกอริทึมผนวก ข.1 การค้นหาคีย์-ค่าด้วยบรูซฟอร์ทบนคูด้า

Brute-force searching key-value on GPU

```
Constant memory is stored the S keys pattern
Shared memory is stored the segment of O keys data
1.Tid = blockIdx.x * blockDim.x + threadIdx.x;
2.Copy text to shared memory
3.Scan O-keys data in device memory, try to match S-keys
4.CUDA thread found, update indices
```

อัลกอริทึมผนวก ข.2 การค้นหาคีย์-ค่าด้วย Horspool บนคูด้า

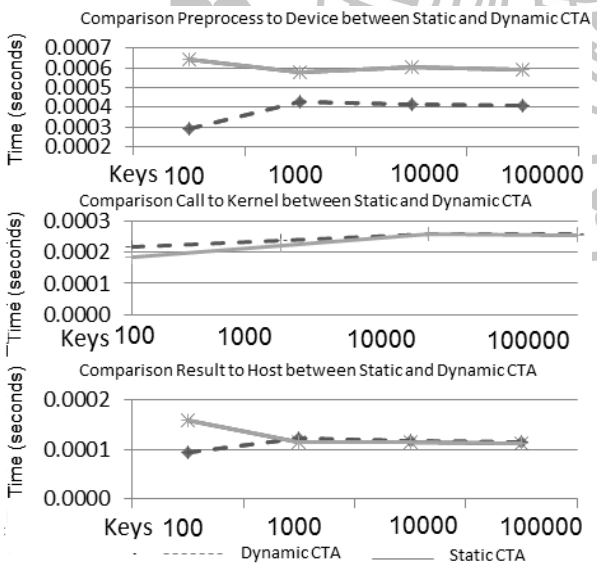
Horspool searching key-value on GPU

```
1.Define pre-computed array condition
2.CUDA kernel scan data for matching the keys
    Skipping distance is an 1D array nevertheless of a match/mismatch
and each valid index element of thread Id
```

ตารางผนวก ข.2 เครื่องที่ใช้ทดสอบการค้นหาอาร์ดีเอฟในฐานข้อมูลแบบคีย์-ค่าบนหน่วยประมวลผลกราฟิกส์

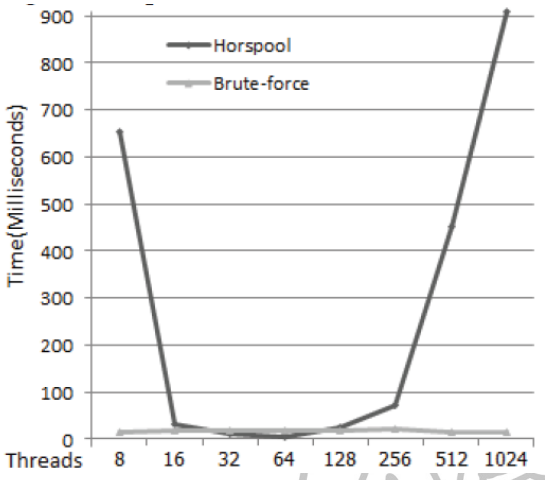
Resources	CPU: Intel ® Core™ i5	GPU: GTX 560 Ti
Cores	4	384
Graphics Clock	850 MHz	822 MHz
Processor Clock	3.1 GHz	1645 MHz
Memory Clock	21 GBps	4008 Gbps (501GBps)
Standard Memory	1024 x 8	1024
Level 2 cache size	4 x 256 KB	512 KB
Level 3 cache size	6 MB shared cache	

ผลการทดลองการค้นหาอาร์ดีเอฟขนาดใหญ่ในฐานข้อมูลแบบคีย์-ค่าบนหน่วยประมวลผลกราฟิกส์ ที่เกี่ยวข้องกับงานวิจัยในวิทยานิพนธ์นี้เป็นไปดังรูปผนวก ข.3 เมื่อทดสอบบนเซตข้อมูล Freebase ด้วยการทดสอบคิวรีที่มีคีย์ของแถวขนาด 100, 1000, 10000 และ 100000 เวลาที่ใช้รันวัดจากสามส่วนดังนี้ ส่วนก่อนการประมวลผลจากโฮสต์จนถึงดีไวส์ (H2D) ส่วนการส่งพารามิเตอร์และเรียกใช้เคอร์เนลล์ (ภายในดีไวส์ หรือ D2D) และการย้ายคำตอบกลับจากดีไวส์สู่โฮสต์ (D2H) โดยรวมแล้วการทำงานแบบ dynamic ช่วยลดเวลาในการทำงานได้มากสำหรับข้อมูลขนาดเล็ก (100 คีย์) โดยเฉพาะในขั้นตอน H2D และช่วยลดเวลาเพียงเล็กน้อยเมื่อข้อมูลมีขนาดใหญ่ขึ้น (1000-100000 คีย์)



รูปผนวก ข.3 การเปรียบเทียบเวลาสามส่วน H2D, D2D และ D2H ระหว่าง บล็อกแบบ static และ dynamic

การจอยน์ในคูด้าโดยจับคู่ S-O บนหน่วยประมวลผลกราฟิกส์ ตามรูปแบบของ CQL เป็นการจับคู่แบบ <??o><?o??> ซึ่งเปรียบได้กับการจอยน์ในฐานข้อมูลเชิงสัมพันธ์ จำนวนเทรดต่อบล็อกที่ใช้ทดสอบคือ 8, 16, 32, 64, 128, 256, 512, และ 1024 พบว่าการทำงานแบบ Horspool ดีกว่าการทำงานของบรูทฟอร์ซที่ขนาด 32 และ 64 เทรดบน GTX 560 Ti ดังรูปผนวก ข.4



รูปผนวก ข.4 เปรียบเทียบการจอยน์ด้วยการเปรียบเทียบสตริงแบบบรูทฟอร์ซและ Horspool บน GPU ด้วยคีย์ขนาด 100 บนฐานข้อมูลแบบคีย์-ค่า ของอาปาเช คาสซานดรา

ภาคผนวก ค

กรณีศึกษาการเปรียบเทียบการประมวลผลอาร์ดีเอฟบนหน่วยประมวลผลกราฟิกส์ระหว่างโครงสร้างแบบฐานข้อมูลคีย์-ค่า และรูปแบบทริพเพิลแบบไบนารี่

จากการทดสอบของผู้วิจัยพบว่า แม้ฐานข้อมูลคีย์-ค่าบนฐานข้อมูลอาปาเช่ คาสซานดรา จะเหมาะสมกับการทำงานแบบขนานบนคลาวด์แต่พบว่าการย่อข้อมูลยังทำได้ไม่ดีพอ ดังตารางผนวก ค.1 ในกรณีที่พิจารณาเรื่องการนำข้อมูลเข้าสู่หน่วยประมวลผลกราฟิกส์ให้มากที่สุด เมื่อนำมาเปรียบเทียบกับการย่อข้อมูลแบบเอชดีที ที่สามารถย่อข้อมูลได้มากกว่าดังกล่าวไว้ในบทที่ 4 และยังสามารถคิวรีข้อมูลได้เช่นกัน

ตารางผนวก ค.1 เปรียบเทียบขนาดบนดิสก์ของไฟล์อาร์ดีเอฟและดัชนีของคาสซานดรา

Keyspaces	Original (GB)	SPO (GB)	OSP (GB)	POS (GB)
Freebase	17.38	4.58	6.32	6.81
DBpedia0	23.76	5.52	12.56	9.82
DBpedia1	23.33	6.07	11.84	8.79

โดยทั่วไปแล้วคาสซานดรายังมีการย่อข้อมูลลงอีกด้วย *NODETOOL* ที่มีอัตราการย่อ minimum/maximum อยู่ที่ 4/32 ผู้วิจัยได้นำมาทดสอบกับเซตข้อมูลที่ใช้ได้ผลดังตารางผนวก ค.2

ตารางผนวก ค.2 การย่อแถวด้วยเครื่องมือ *NODETOOL* ของอาปาเช่ คาสซานดรา

	SPO (Bytes)			OSP (Bytes)			POS (Bytes)		
Key-spaces	min	Max (MB)	mean	min	Max (MB)	mean	min	Max (MB)	mean
Freebase	180	0.08	622	125	177.92	337	180	256.21	331
DBpedia0	150	0.30	1437	125	102.96	536	180	34.48	449
DBpedia1	125	0.25	1244	125	148.27	541	180	102.96	507

เมื่อเปรียบเทียบกับเอชดีทีใน ตารางผนวก ค.3 แล้วพบว่าเอชดีทีกำจัดแถวที่ไม่สามารถนำเข้ามาแปลงข้อมูลออกไปมากกว่าคาสซานดรา การที่มีคีย์ SPO อย่างเดียวเอชดีทีจึงมีผลการย่อโดยรวมที่มากกว่า คาสซานดราที่แยกคีย์ออกเป็นสามแบบเมื่อนำมารวมกันจึงมีขนาดมากกว่าที่เอชดีทีได้

ตารางผนวก ค.3 เปรียบเทียบขนาดข้อมูลอาร์ดีเอฟและโครงสร้างเอชดีที

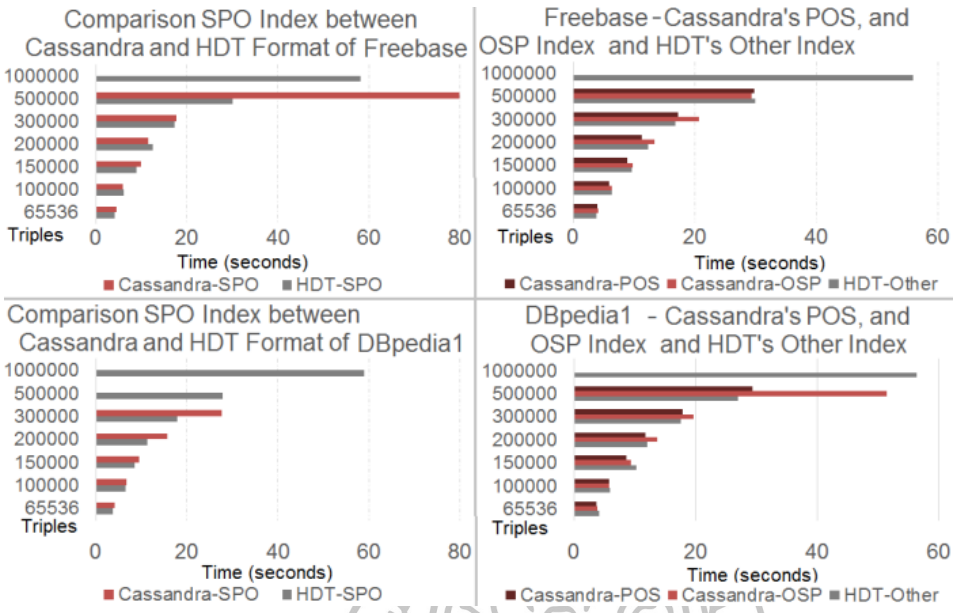
Dataset Name	Chunk in Disk		HDT- SPO		
	Triples	Size (GB)	Triples	Size (GB)	Times(s)
Freebase	10 M	1.31	9,994,362	0.1	62
Freebase	20 M	2.62	19,980,084	0.2	219
Freebase	30 M	3.94	29,944,158	0.3	373
DBpedia0	10 M	1.87	9,897,356	0.5	95
DBpedia1	10 M	1.87	9,966,001	0.5	108

การนำข้อมูลที่อยู่ในอาปาเช่ คาสซานดราและเอชดีทีมาประมวลผลการค้นหาแบบบรูทฟอร์ซเปรียบเทียบกันบนหน่วยประมวลผลกราฟิกส์ ดังขั้นตอนในตารางผนวกค.4 ตารางแบ่งเป็นสามคอลัมน์ บอกคุณลักษณะของการเก็บข้อมูลทั้งสองแบบ ได้แก่ เซตข้อมูลเข้า ที่เหมือนกันคือ Freebase และ DBpedia ถัดไป ลักษณะการเก็บข้อมูล คาสซานดราเก็บเป็นคีย์สเปซ ขณะที่เอชดีทีเก็บข้อมูลเป็น เฮดเดอร์ ดิกชันนารีและทริพเพิล ดัชนีที่ใช้ คาสซานดราใช้ คอลัมน์แฟมิลี่ ขณะที่เอชดีทีใช้แท็ก hdt#triplesOrder การเก้บดัชนีเพื่อการคิวรี คาสซานดราใช้สามแบบคือ SPO, OSP, POS ขณะที่เอชดีทีใช้ SPO การเก็บแคช คาสซานดราเก็บ key cache แต่เอชดีทีเก็บลงหน่วยความจำ รูปแบบการเก็บข้อมูลมีการเรียงแล้วทั้งสองฐานและเรียงตามดัชนีที่สร้างไว้ ภาษาคิวรีใช้แตกต่างกัน คาสซานดราใช้ CQL ส่วนเอชดีทีใช้ ไลบรารี่ของเอชดีทีหาแบบไบนารี่

การพัฒนาโปรแกรมเป็นไปในทำนองเดียวกันคือใช้เจคูด้า เป็นไลบรารี่เชื่อมต่อระหว่างอินเตอร์เฟสการค้นหาของแต่ละฐานและคูด้า ใช้ 128 เทรด จากนั้นทำเป็นไฟล์ PTX การวัดค่าคือจำนวนคีย์ที่ประมวลผลได้และเวลาที่ใช้

ตารางผนวก ค.4 การเรียบเทียบรายละเอียดการทดสอบบน GPU ระหว่างคาสซานดราและเอชดีที [143]

Exchanging Storage	Apache Cassandra	HDT binary RDF
Inputs	Freebase, DBpedia	Freebase, DBpedia
Storage Format	Keyspace	Header,Dictionary,Triples
Grouped Index	Column Family	hdt#triplesOrder
Query Indices	SPO, OSP, POS	SPO, and other apply based on SPO
Cache	Key cache	In-memory Index
Format	Based on SPO, OSP, POS [143] Sorted Fashion	Triple ID based on SPO Sorted Fashion
Query Language	CQL	HDT-java binary search
CPU activity	Query from cache to GPU by set of Coorperative Thread Array	
Java's GPU binding	Jcuda	Jcuda
Cooperative Thread Array	Structure of Array: number of words, pointer of input/output, word length/offset Generate to PTX file	
Threads	128	128
CUDA kernel	Retrieve data from cache to java array, Send parameter to CUDA by Jcuda Connect to PTX file, send data to process Results back from device to host	
Measure	Quantity and time consumed (see more in [143])	



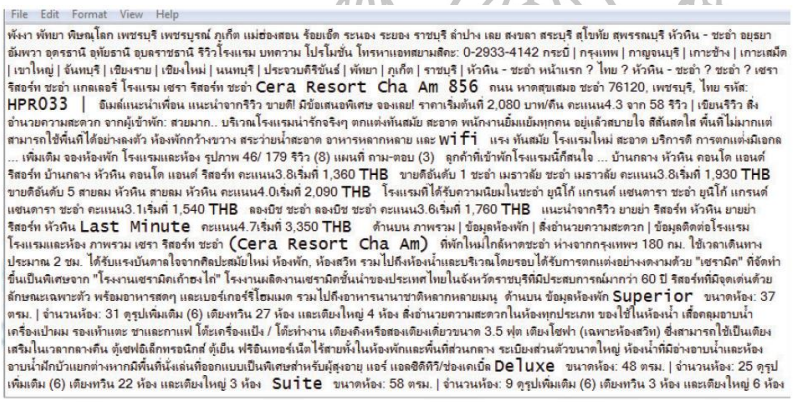
รูปผนวก ค.1 การเปรียบเทียบเวลาที่ใช้ในการนำข้อมูลอาร์ดีเอฟเข้าสู่หน่วยประมวลผลกราฟิกส์ระหว่างเอชดีทีและอาปาเช คาสซานดรา

ผลลัพธ์ดังรูปผนวก ค.1 เนื่องด้วยโครงสร้างของเอชดีทีแตกต่างจากโครงสร้างของคาสซานดรา เมื่อใช้งานเอชดีทีโครงสร้างนี้จะเก็บดัชนีที่ใช้ในหน่วยความจำทั้งหมด แล้วอ้างอิงด้วยเลขทริพเพิลเช่น เลขเอเลเม้นต์ที่ 568 หมายความถึงกลุ่มของ subject ID, predicated ID และ object ID ได้เป็นเลข 95, 6032, 2757077 ซึ่งแตกต่างจากการหาค่า SPO ในคาสซานดราที่อยู่ในรูป {s: {p : {o}}} และสามารถค้นหาได้บางรูปแบบคือ (s p o) (? ? ?) (s ? ?) (s p ?) เมื่อพิจารณาถึงข้อมูลที่นำเข้ามาทดสอบจากแกน y พบว่า ในขณะที่เอชดีทีส่งเลขทริพเพิลได้มากที่สุดจำนวนหนึ่งล้านชุดแต่คาสซานดราส่งเข้าไปได้ 5 แสนคีย์ของแถว บางชุดข้อมูลได้แก่ DBpedia 1 คลาสซานดราส่งค่าได้มากที่สุดเพียง 300000 คีย์เท่านั้น เมื่อพิจารณาเรื่องเวลาที่ใช้ในการโหลดข้อมูลส่วนใหญ่แล้วเวลาที่ใช้ในการโหลดข้อมูลสู่หน่วยประมวลผลกราฟิกส์โครงสร้างเอชดีทีใช้เวลาน้อยกว่า

ภาคผนวก ง

การดึงข้อมูลแบบขนานเพื่อสร้างออนโลยี ในโดเมนการท่องเที่ยวเชิงสุขภาพ

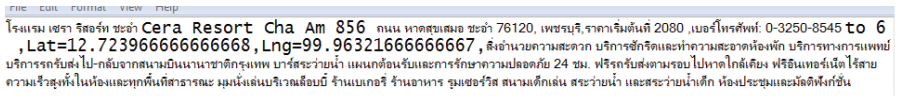
จากปัญหาการเก็บข้อมูลจากเว็บไซต์ต่าง ๆ เพื่อนำมาสร้างออนโทโลยีที่เรียงตามคอนเซปต์ของ subject predicate object นั้นต้องใช้ข้อมูลจากเว็บไซต์หลายแหล่ง ภายใต้เว็บไซต์นั้นมีข้อมูลที่เป็นอักขระอยู่ เมื่อต้องการรวบรวมข้อมูลเหล่านั้นมาใส่ตามแนวคิดที่ได้ออกแบบไว้ก่อนเป็นโครงสร้างออนโทโลยีต้องใช้ระยะเวลานาน ดังนั้นผู้วิจัยจึงนำการทำงานแบบขนานมาประยุกต์ใช้ในการดึงข้อมูลของเว็บไซต์และเรียงข้อมูลเพื่อเตรียมใส่ตามแนวคิดของออนโทโลยี [106]


File Edit Format View Help

พังงา พัทยา พิษณุโลก เพชรบุรี เพชรบูรณ์ ภูเก็ต แม่ฮ่องสอน ร้อยเอ็ด ระนอง ระยอง ราชบุรี ลำปาง เลย สงขลา สระบุรี สุโขทัย สุพรรณบุรี หัวหิน - ชะอำ อยุธยา อัมพวา อุดรธานี อุทัยธานี อุบลราชธานี รีวิวโรงแรม บทความ โปรโมชั่น โทรหาแอทสยามสิตะ: 0-2933-4142 กระบี่ | กรุงเทพ | กาญจนบุรี | เกาะช้าง | เกาะเสม็ด | เขาใหญ่ | จันทบุรี | เชียงราย | เชียงใหม่ | นนทบุรี | ประจวบคีรีขันธ์ | พัทยา | ภูเก็ต | ราชบุรี | หัวหิน - ชะอำ หน้าแรก ? ไทย ? หัวหิน - ชะอำ ? ชะอำ ? เซรา รีสอร์ท ชะอำ แกลเลอรี่ โรงแรม เซรา รีสอร์ท ชะอำ **Cera Resort Cha Am 856** ถนน หาดสุขเสมอ ชะอำ 76120, เพชรบุรี, ไทย รหัส: **HPR033** | อีเมล์แนะนำเพื่อน แนะนำจากรีวิว ขายดี! มีข้อเสนอพิเศษ จองเลย! ราคาเริ่มต้นที่ 2,080 บาท/คืน คะแนน4.3 จาก 58 รีวิว | เขียนรีวิว สิ่งอำนวยความสะดวก จากผู้เข้าพัก: สวยมาก.. บริเวณโรงแรมน่ารักจริงๆ ตกแต่งทันสมัย สะอาด พนักงานยิ้มแย้มทุกคน อยู่แล้วสบายใจ สิ้นสุดใส พื้นที่ไม่มากแต่สามารถใช้พื้นที่ได้อย่างลงตัว ห้องพักกว้างขวาง สระว่ายน้ำสะอาด อาหารหลากหลาย และ **wifi** แรง ทันสมัย โรงแรมใหม่ สะอาด บริการดี การตกแต่งมีเอกล ... เพิ่มเติม จองห้องพัก โรงแรมและห้อง รูปภาพ 46/ 179 รีวิว (8) แผนที่ กลม-ตอบ (3) ลูกค้าที่เข้าพักโรงแรมนี้ก็สนใจ ... บ้านกลาง หัวหิน คอนโด แอนด์ รีสอร์ท บ้านกลาง หัวหิน คอนโด แอนด์ รีสอร์ท คะแนน3.8เริ่มที่ 1,360 **THB** ขายดีอันดับ 1 ชะอำ เมราลัย ชะอำ เมราลัย คะแนน3.8เริ่มที่ 1,930 **THB** ขายดีอันดับ 5 สายลม หัวหิน สายลม หัวหิน คะแนน4.0เริ่มที่ 2,090 **THB** โรงแรมที่ได้รับความนิยมในชะอำ ยูนิโก้ แกรนด์ แซนตารา ชะอำ ยูนิโก้ แกรนด์ แซนตารา ชะอำ คะแนน3.1เริ่มที่ 1,540 **THB** ลองบีช ชะอำ ลองบีช ชะอำ คะแนน3.6เริ่มที่ 1,760 **THB** แนะนำจากรีวิว ยายย่า รีสอร์ท หัวหิน ยายย่า รีสอร์ท หัวหิน **Last Minute** คะแนน4.7เริ่มที่ 3,350 **THB** ตำบล ภาพรวม | ข้อมูลห้องพัก | สิ่งอำนวยความสะดวก | ข้อมูลติดต่อโรงแรม โรงแรมและห้อง ภาพรวม เซรา รีสอร์ท ชะอำ **(Cera Resort Cha Am)** ที่พักใหม่ใกล้หาดชะอำ ห่างจากกรุงเทพฯ 180 กม. ใช้เวลาเดินทาง ประมาณ 2 ชม. ได้รับแรงบันดาลใจจากศิลปะสมัยใหม่ ห้องพัก, ห้องสวิท รวมไปถึงห้องน้ำและบริเวณโดยรอบได้รับการตกแต่งอย่างลงตัวด้วย "เซรามิค" ที่จัดทำขึ้นเป็นพิเศษจาก "โรงงานเซรามิคเก้าอ้อไก่" โรงงานผลิตงานเซรามิคชั้นนำของประเทศไทยในจังหวัดราชบุรีที่มีประสบการณ์มากกว่า 60 ปี รีสอร์ทที่มีจุดเด่นด้วยลักษณะเฉพาะตัว พร้อมอาหารสดๆ และเบอร์เกอร์รีโฮมเมด รวมไปถึงอาหารนานาชาติหลากหลายเมนู ตำบล ข้อมูลห้องพัก **Superior** ขนาดห้อง: 37 ตรม. | จำนวนห้อง: 31 ดูรูปเพิ่มเติม (6) เตียงทวิน 27 ห้อง และเตียงใหญ่ 4 ห้อง สิ่งอำนวยความสะดวกในห้องทุกประเภท ของใช้ในห้องน้ำ เสื้อคลุมอาบน้ำ เครื่องเป่าผม รองเท้าแตะ ชาและกาแฟ โต๊ะเครื่องแป้ง / โต๊ะทำงาน เตียงดึงหรือสองเตียงเดี่ยวขนาด 3.5 ฟุต เตียงโซฟา (เฉพาะห้องสวิท) ซึ่งสามารถใช้เป็นเตียงเสริมในเวลากลางคืน ตู้เซฟอิเล็กทรอนิกส์ ตู้เย็น ฟรีอินเทอร์เน็ตไร้สายทั้งในห้องพักและพื้นที่ส่วนกลาง ระเบียงส่วนตัวขนาดใหญ่ ห้องน้ำที่มีอ่างอาบน้ำและห้องอาบน้ำฝักบัวแยกต่างหากมีพื้นที่นั่งเล่นที่ออกแบบเป็นพิเศษสำหรับผู้สูงอายุ แอร์ แอลซีดีทีวี/ช่องเคเบิ้ล **Deluxe** ขนาดห้อง: 48 ตรม. | จำนวนห้อง: 25 ดูรูปเพิ่มเติม (6) เตียงทวิน 22 ห้อง และเตียงใหญ่ 3 ห้อง **Suite** ขนาดห้อง: 58 ตรม. | จำนวนห้อง: 9 ดูรูปเพิ่มเติม (6) เตียงทวิน 3 ห้อง และเตียงใหญ่ 6 ห้อง

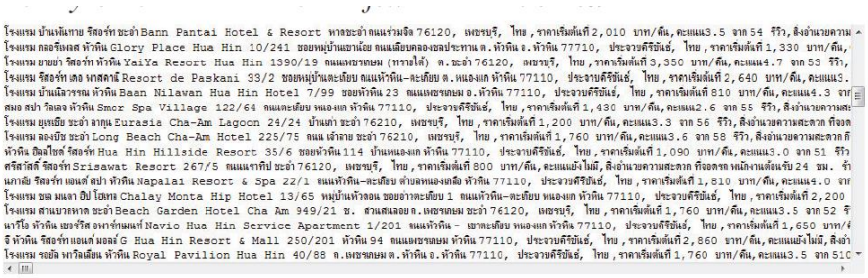
รูปผนวก ง.1 ข้อมูลจากเว็บไซต์ AtSiam

จากรูปผนวก ง.1 เป็นข้อมูลจากเว็บพอร์ทัลของโรงแรมที่ชื่อ AtSiam ผู้วิจัยพัฒนาโปรแกรมดึงข้อมูลออกจากเว็บไซต์ ได้ข้อมูลในรูปผนวก ง.2 เมื่อพิจารณาว่ามีหลายเว็บไซต์จึงดึงข้อมูลแบบขนานด้วย Java Concurrency ได้ดังรูปผนวก ง.3 จะเห็นได้ว่าข้อมูลถูกเรียงในรูปแบบคอลัมน์ เพื่อให้สามารถนำไปแมพได้กับไฟล์ที่ดึงมาจากเว็บไซต์อื่น

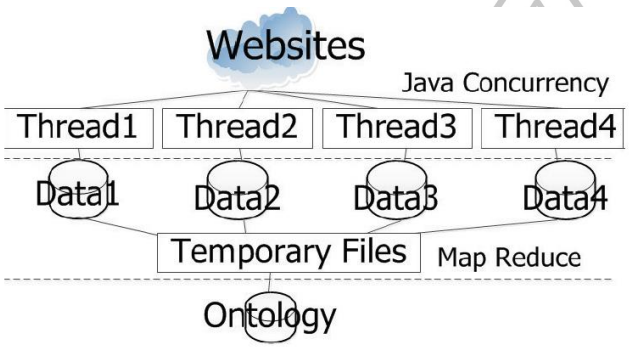

File Edit Format View Help

โรงแรม เซรา รีสอร์ท ชะอำ **Cera Resort Cha Am 856** ถนน หาดสุขเสมอ ชะอำ 76120, เพชรบุรี, ราคาเริ่มต้นที่ 2080 ,เบอร์โทรศัพท์: 0-3250-8545 **to 6** **,Lat=12.723966666666668,Lng=99.96321666666667,** สิ่งอำนวยความสะดวก บริการซักรีดและทำความสะอาดห้องพัก บริการทางการแพทย์ บริการรถรับส่งไป-กลับจากสนามบินนานาชาติกรุงเทพ บาร์สระว่ายน้ำ แผนกต้อนรับและการรักษาความปลอดภัย 24 ชม. ฟรีรถรับส่งตามรอบไปหาดใกล้เคียง ฟรีอินเทอร์เน็ตไร้สายความเร็วสูงทั่วในห้องและทุกพื้นที่สาธารณะ มุมนั่งเล่นบริเวณล็อบบี้ ร้านบเกอรี ร้านอาหาร รูมเซอร์วิส สนามเด็กเล่น สระว่ายน้ำ และสระว่ายน้ำเด็ก ห้องประชุมและมัลติฟังก์ชั่น

รูปผนวก ง.2 ข้อมูลที่ได้หลังจากดึงเฉพาะสิ่งที่สนใจออกมา



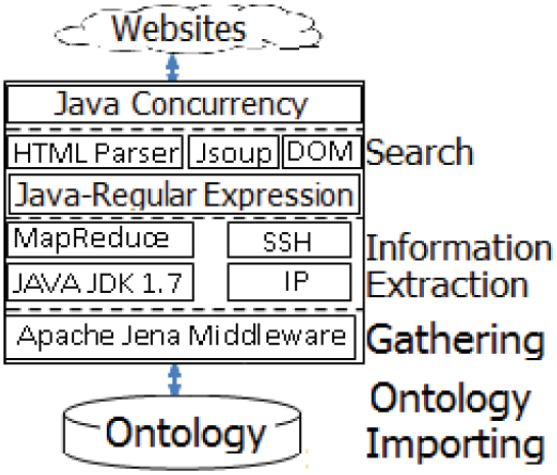
รูปผนวก ง.3 ข้อมูลที่ได้หลังจากผ่านการดึงข้อมูลแบบขนาน



รูปผนวก ง.4 องค์ประกอบของระบบ

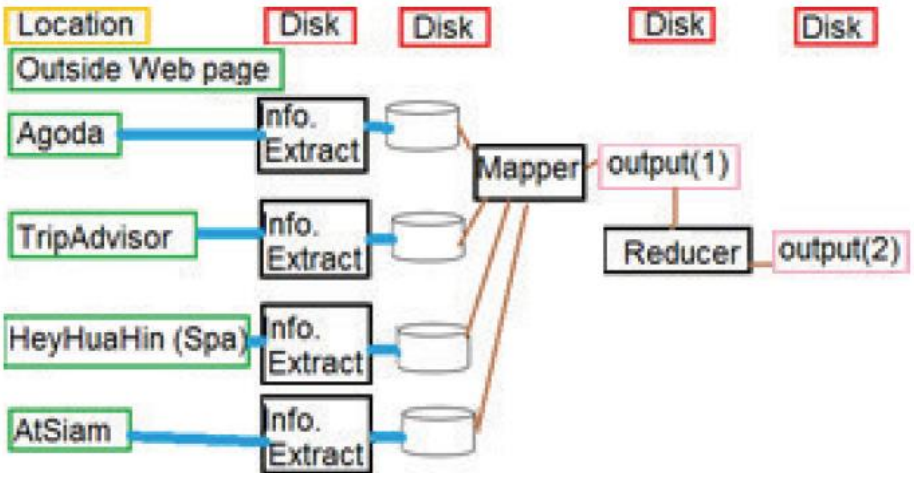
จากรูปผนวก ง.4 แสดงองค์ประกอบของระบบการดึงข้อมูลแบบขนานซึ่งแบ่งออกเป็นสามขั้นตอนคือ การค้นหา การดึงข้อมูล และการรวมข้อมูล ผู้วิจัยใช้ Java Concurrency ในขั้นตอนการค้นหาและการดึงข้อมูล ขณะที่ใช้แมพรีดิวซ์ในการรวมไฟล์เพื่อใส่ลงออนโทโลยี

เริ่มจากใช้ Java Concurrency คลอว์ลข้อมูลจากแต่ละเว็บไซต์ ไลบรารี่ HTML parser และ Jsoup ถูกใช้เพื่อดึงข้อมูลจากเว็บ การหาคำสำคัญสำหรับแต่ละแอททริบิวต์ใช้ Regular Expression สำหรับแมพรีดิวซ์ใช้รวมข้อมูลไฟล์เท็กซ์ที่ได้จากขั้นตอนก่อนหน้า ในส่วนนี้ใช้งาน Java JDK 1.7 SSH IP และ Hadoop ส่วนมิดเดิลแวร์ใช้อาปาเช่เจนา สำหรับทำตรวจสอบความเป็นเหตุผลและตรวจสอบความถูกต้องของออนโทโลยี



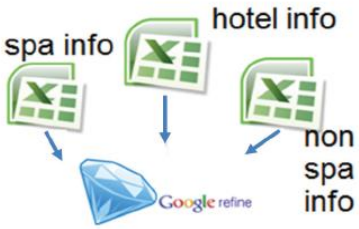
รูปผนวก ง.5 สถาปัตยกรรมของระบบ

จากรูปผนวก ง.5 จากซ้ายมาขวา โปรแกรมอ่านไฟล์เท็กซ์ที่ได้จากขั้นตอนก่อนหน้าลงสู่ดิสก์ ส่วนของงานแมพ ทำหน้าที่แมพงานที่มีแต่ละไฟล์ แล้วมารวมกัน ได้ไฟล์ทั้งหมดนี้เป็นเอาต์พุต (1) จากนั้นขั้นรีดิวซ์ตัวรีดิวเซอร์ในขั้นตอนนี้ นำ เอาต์พุต (1) ไปรวมกับเรคคอร์ดที่มีคีย์เหมือนกัน จึงได้เอาต์พุต (2) นำเอาต์พุต (2) นี้ไปสร้างออนโทโลยีในขั้นตอนต่อไป



รูปผนวก ง.6 เฟสของแมพรีดิวซ์

จากรูปผนวก ง.6 ไฟล์เท็กซ์ที่ได้จากเฟสของแมพรีดิวซ์ (เอาต์พุต (2)) และข้อมูลจากการเก็บภาคสนาม (field-trip) ได้รับการรวมโดยใช้ Gogole Refine แล้วจึงส่งออกไปในรูปแบบออนโทโลยี เป็นดังรูปผนวก ง.7



รูปผนวก ง.7 การรวมไฟล์ที่ได้จากแมพรีดิวซ์

รายละเอียดรูปผนวก ง.7 การรวมไฟล์ Spa information, Hotel information และ Medical tourism information (non-spa info) ที่ได้จากแมพรีดิวซ์และข้อมูลจากการเก็บภาคสนามไปยังออนโทโลยีโดยใช้เครื่องมือ Google Refine หลังจากนำเข้าโครงสร้างของออนโทโลยีแล้วอาจมีบางพร็อพเพอร์ตี้หรือชนิดข้อมูลที่ต้องทำการแก้ไข สามารถแก้ด้วยมือต่อได้โดยใช้ Protégé 4.3 (2014)

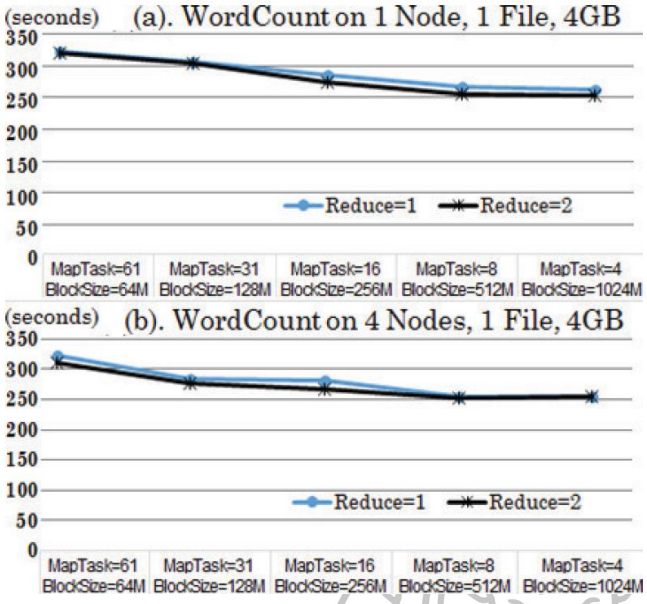
เครื่องที่ใช้ในการทดสอบคือคอมพิวเตอร์ 4 แกน Intel ® Core™ i3 CPU M 350 @ 2.27GHz (64-bit) และหน่วยความจำ 3.7 GB RAM ผู้วิจัยใช้มัลติทรดในเครื่องนี้เพื่อดึงข้อมูลเว็บไซต์

ตารางผนวก ง.1 รายชื่อ เทรด และเวลาของข้อมูลที่ดึงออกมาด้วย Java Concurrency

Website/Thread	*1 Thread(ms)*	*2 Thread (ms)*	*4 Thread (ms)*
Heyhuahin (82 webs)	4602	2529	1994
Atsiam (146 webs)	100012	64728	45763
Tripadvisor (247 webs)	270121	135600	73981
Agoda (237 webs)	194443	143590	74935

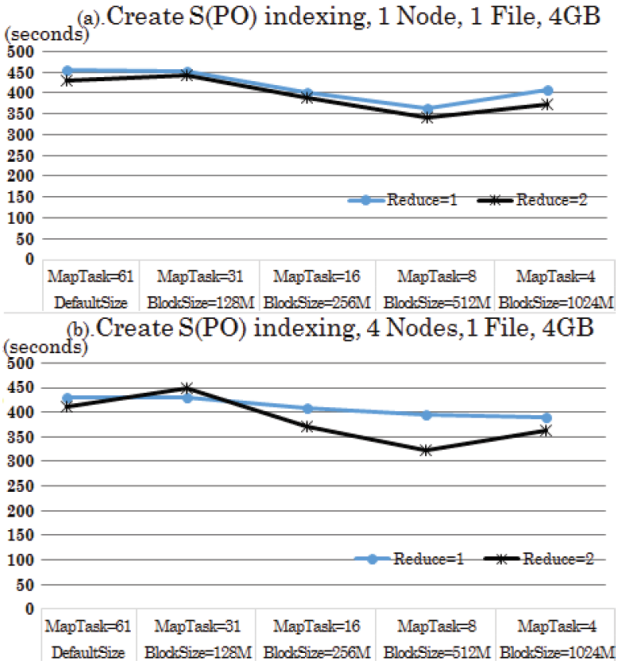
จากตารางผนวก ง.1 แสดงรายชื่อของเว็บไซต์จำนวน 4 เว็บไซต์ได้แก่ Heyhuahin (82 หน้า), Atsiam (146 หน้า), Tripadvisor (247 หน้า) และ Agoda (237 หน้า) ใช้เวลาในการดึงข้อมูลมากที่สุดเมื่อใช้เพียง 1 เทรด เมื่อเริ่มทำงานแบบขนานโดยการใช้ 2 เทรด เวลาในการทำงานลดลง และเมื่อใช้ 4 เทรดเวลาจะลดลงมากที่สุด โดยเป็นการเร่งความเร็ว 2.30 เท่า 2.18 เท่า 3.65 เท่าและ 2.59 เท่าตามลำดับ

โหนด master มีคุณสมบัติดังนี้ Intel ® Core™ i5-3330 CPU@ 3.00GHz (64-bit), หน่วยความจำ 7.7 GB โหนด slave มีคุณสมบัติดังนี้ Intel ® Core™ 2 Duo CPU E8400 @ 3.00GHz (64-bit), 1.9 GB RAM และ Intel ® Core™ i5-3330 CPU@ 3.00GHz (64-bit), 7.7 GB RAM 2 เครื่อง



รูปผนวก ง.8 การเปรียบเทียบระหว่างการประมวล WordCount ระหว่าง 1 โหนดและ 4 โหนด พร้อมกับพารามิเตอร์ งานแมพ (Map Task), ขนาดบล็อก (Block Size) และ งานรีดิวซ์ (Reduce Task)

ในขั้นตอนแรกของการทดสอบประสิทธิภาพ รูปผนวก ง.8 - ง.9 แสดงตัวอย่างการรันงานแมพรีดิวซ์ เซตข้อมูลได้รับการประบุกต์มาจาก BTC2012 ไฟล์ที่นำเข้าเป็น N-Triple ได้รับการนำมาใช้เปรียบเทียบระหว่างการทำงาน 1 โหนและ 4 โหนด ขนาดของไฟล์คือ 4 กิกะไบต์ จำนวน 21,398,608 ทริพเพิล ผู้วิจัยประยุกต์ใช้โปรแกรม WordCount ทดสอบกับข้อมูลบนคลัสเตอร์ที่ติดตั้งไว้ จากรูปผนวก ง.8 การประมวลผล WordCount ระหว่าง 1 โหนด รูป (a) และ 4 โหนด รูป (b) พิจารณา งานแมพ (Map Task) ถูกแบ่งออกเป็น 4 – 61 ด้วยบล็อกขนาด 64, 128,256,512, และ 1024 เมกะไบต์ วัดเวลาที่ใช้ระหว่างทำ 1-2 รีดิวเซอร์ ผลลัพธ์ที่ได้คือลดเวลาการทำงานเริ่มจาก 61 งานที่บล็อกขนาด 64 เมกะไบต์ เวลาลดลงเมื่อ จาก 31 งานที่บล็อกขนาด 128 เมกะไบต์ แล้วลดลงต่อเนื่องผ่าน 16 งานที่บล็อกขนาด 256 เมกะไบต์ 8 งานที่บล็อกขนาด 512 เมกะไบต์ จนถึง 4 งานที่บล็อกขนาด 1024 เมกะไบต์ หมายความว่า งานแมพที่น้อยที่สุดแปรผันตามขนาดของบล็อกที่โตขึ้น และสามารถเร่งความเร็ว ของงานได้ทั้งกรณี 1 และ 4 โหนด สำหรับงานรีดิวซ์ทั้งสองงานสามารถปรับปรุงให้เร่งความเร็วได้ใน 1 โหนดด้วย งานแมพที่น้อยที่สุดแล้วเพิ่มขนาดบล็อก อย่างไรก็ตามในสี่โหนดนั้น ขนาดบล็อกที่เล็กที่สุดได้ประโยชน์จากการใช้งานรีดิวซ์สองครั้งมากกว่า ผู้วิจัยจัดทำการทดสอบด้วยคลัสเตอร์ขนาดเล็ก จึงยังไม่สามารถเห็นผลลัพธ์ของงานที่ลดลงอย่างรวดเร็วได้ ซึ่งหวังว่านักพัฒนาโปรแกรมนำโมเดลเช่นนี้ไปใช้กับคลัสเตอร์ขนาดใหญ่มากกว่า 64 โหนดจะเห็นผลลัพธ์ที่เป็นที่น่าพอใจ



รูปผนวก ง.9 การเปรียบเทียบระหว่างดัชนี S(PO) บนเครื่อง 1 และ 4 โหนด พร้อมกับ พารามิเตอร์ งานแมพ (Map Task), ขนาดบล็อก (Block Size) และ งานรีดิวซ์ (Reduce Task)

จากรูปผนวก ง.9 แสดงถึงดัชนี S(PO) สำหรับการจับกลุ่มค่า predicate และ object ไปที่แต่ละคีย์ของ subject ด้วยการโปรแกรมแมพรีดิวซ์ เป้าหมายของโปรแกรมนี้ คือการรวมสตริงบนค่า subject เดียวกัน ในงานนี้ใช้การรวมข้อมูลที่ได้จากการคลอว์ลในตารางผนวก ง.1 จำนวน 712 ไฟล์ การตรวจสอบด้วยการวัดประสิทธิภาพระหว่าง 1 (รูปผนวก ง.9 a) และ 4 โหนด (รูปผนวก ง.9 b) โดยสร้างพารามิเตอร์เหมือนโปรแกรม WordCount ได้แก่ งานแมพ (Map Task), ขนาดบล็อก (Block Size) และ งานรีดิวซ์ (Reduce Task) ผลลัพธ์ที่มีประสิทธิภาพดีที่สุดคือการใช้ งานแมพ = 8 และ งานรีดิวซ์ = 2 สำหรับทั้งสองกรณี ขณะที่ 4 โหนดทำเวลาได้ดีกว่าเล็กน้อย สำหรับคลัสเตอร์ขนาดเล็กทำงานกับไฟล์ขนาดใหญ่ พบว่าการปรับปรุงงานนับว่ายังไม่เป็นที่น่าพอใจ ผู้วิจัยเชื่อว่าการมีคลัสเตอร์ขนาดใหญ่จะทำงานได้มีประสิทธิภาพมากกว่านี้ แต่อย่างไรก็ตามหากไม่มีแมพรีดิวซ์การรวมไฟล์ขนาด 4 กิกะไบต์จะใช้เวลาเกือบชั่วโมง ขณะที่แมพรีดิวซ์ช่วยให้เวลาทำงานลดลงเหลือ 5-7 นาที ประเด็นที่เหลือคือการตั้งค่าที่เหมาะสมแก่งานของแมพรีดิวซ์เพื่อให้ผลการเร่งความเร็วออกมาเป็นที่น่าพอใจ

สำหรับผลลัพธ์ที่ได้นำไปใส่ในคอนเซ็ปต์ของออนโทโลยีต่อไป มีเป้าหมายเพื่อสร้างเว็บเชิงความหมายในงานที่ต่อเนื่องกันใน [144] และ [145]

ประวัติผู้วิจัย

ประวัติการศึกษา

พ.ศ. 2547	วท.บ. วิทยาการคอมพิวเตอร์ มหาวิทยาลัยศิลปากร
พ.ศ. 2554	วท.ม. เทคโนโลยีสารสนเทศ มหาวิทยาลัยศิลปากร

ทุนการศึกษา

พ.ศ. 2554	ทุนกาญจนาภิเษก รุ่นที่ 14
พ.ศ. 2557	- ทุน TRF-DAAD Project Based Personnel Exchange Programme - ทุนอุดหนุนการทำวิทยานิพนธ์จากงบประมาณแผ่นดิน (หมวดเงินอุดหนุนทั่วไป) ของบัณฑิตวิทยาลัย มหาวิทยาลัยศิลปากร

ประวัติการทำงาน

พ.ศ. 2547 – 49	บริษัทโดล ไทยแลนด์จำกัด, ERP (JDE), BI (I2), ETL (Infomatica)

โครงการวิจัย

พ.ศ. 2556	หัวหน้าโครงการย่อย: ออนโทโลยีโมเดลและการคิวรี่สำหรับการท่องเที่ยวเชิงสุขภาพ กรณีศึกษาการท่องเที่ยวเชิงสุขภาพ อำเภอหัวหิน ในแผนงาน RDG5650033 ระบบบริหารจัดการการท่องเที่ยวเชิงสุขภาพ กรณีศึกษาอำเภอหัวหิน โดยอาศัยออนโทโลยีและเทคโนโลยีการประมวลผลแบบขนานบนกลุ่มเมฆ (แหล่งทุน: สกว)
พ.ศ. 2558	หัวหน้าโครงการย่อย: บริการเว็บสำหรับปรับเปลี่ยนข้อมูลการท่องเที่ยวเชิงสุขภาพ ในแผนงาน RDG5850042 ระบบบริการข้อมูลเพื่อประชาสัมพันธ์การท่องเที่ยวเชิงสุขภาพผ่านเว็บเซอร์วิส แบบผสานโลกเสมือน กรณีศึกษาอำเภอหัวหิน (แหล่งทุน: สกว)

พิจารณาบทความวารสาร

พ.ศ. 2558	Computer & Electrical Engineering (CAEE), Elsevier, Mar 15.

งานประชุม

พ.ศ. 2554	WICT 2011: The World Congress on Information and Communication Technologies, India.
พ.ศ. 2556	ICSEC 2013: The 2013 International Computer Science and Engineering Conference, Thailand.
พ.ศ. 2559	IEA/AIE 2016: The 29th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems, 2~4 August 2016, Morioka, Japan.

รางวัล

พ.ศ. 2558	Merit Award - NVIDIA GTC 2015 POSTER CONTEST (GPU TECHNOLOGY CONFERENCE ASIA SOUTH 2015)–Aug 2015

ผลงานที่เกี่ยวข้องกับวิทยานิพนธ์

ผลงานวารสาร

- **Chidchanok Choksuchat**, Chantana Chantrapornchai. (2012). On The Improvement of Search Engine Using Semantic Web: Case Study of Hua Hin Tourism Information System. ECTI Transactions on Computer and Information Technology. 6 (2):186-204. (Index: TCI)

- **Chidchanok Choksuchat**, Chantana Chantrapornchai. (2016). On the Development of Health Tourism Semantic Web with its Parallel Engine. International Journal of Metadata Semantics and Ontologies, 11(1): 16–28. (IE Compendex)

ผลงานโปสเตอร์

- **Chidchanok Choksuchat**, Chantana Chantrapornchai, (Aug 2015) Optimization Parallel Search for Resource Description Framework Tasks, ASIA SOUTH POSTER CONTEST at GTC Asia South 2015, At NTU, Singapore. (Merit Award)

งานประชุมวิชาการนานาชาติ

- Chidchanok Choksuchat, Chantana Chantrapornchai, Michael Haidl, & Sergei Gorlatch. (2015). Accelerating Keyword Search for Big RDF Web Data on Many-Core Systems. Presented at the The 1 4 th International Conference on Intelligent Software Methodologies, Tools and Techniques, Italy: Springer-Verlag Communications in Computer and Information Science (CCIS)

- Chidchanok Choksuchat, Suphaksa Ngamphak, Benjaporn Maneesaeng , Yuwathida Chiwpreechar, Chantana Chantrapornchai. (2014). Parallel health tourism information extraction and ontology storage. 2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE), 236 - 241.

- Chidchanok Choksuchat, Chantana Chantrapornchai (2014). Query Processing for Large RDF with Tree Representation on GPU, RGJ-Ph.D. Congress XV, Pattaya, Chonburi, Thailand

- Chidchanok Choksuchat, Chantana Chantrapornchai (2013). Experimental framework for searching large RDF on GPUs based on key-value storage. 10th International Joint Conference on Computer Science and Software Engineering (JCSSE), 171-176.

- Chidchanok Choksuchat, Chantana Chantrapornchai.(2013). Large RDF Representation Framework for GPUs Case Study Key-Value Storage and Binary Triple Pattern. 2013 International Computer Science and Engineering Conference (ICSEC) (IEEE) , 13-18.

- Chidchanok Choksuchat, Chantana Chantrapornchai (2013). On the HDT with the Tree Representation for Large RDFs on GPU. The second International Workshop on Scalable Computing for Big Data Analytics (SC-BDA) of The 19th IEEE International Conference on Parallel and Distributed Systems (ICPADS), 651 - 656.

- Chidchanok Choksuchat, “The case study of Parallel Java for SPARQL queries on semantic web”, Ph.D. Forum, KST -2013, Faculty of Informatics, Burapha University

- Chidchanok Choksuchat, Chantana Chantrapornchai. (2012). The Case Study of Parallel Java for SPARQL Queries on DBpedia. The 2012 International Computer Science and Engineering Conference, 167-172.

- Chidchanok Choksuchat, Chantana Chantrapornchai. (2012). On the Comparison of Top-Down and Bottom-Up Searching in OWL DL. Proceedings of the Joint International Symposium on Natural Language Processing and Agriculture Ontology Service, 85-89.