ระบบการประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์

นายวรินทร์ วัฒนพรพรหม

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2545

ISBN 974-17-1195-6

A PARALLEL PROCESSING SYSTEM FOR A BLAST PROGRAM

Mr. Warin Wattanapornprom

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computer Science

Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University

Academic Year 2002

ISBN 974-17-1195-6

หัวข้อวิทยานิพนธ์	ระบบการประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์
โดย	นายวรินทร์ วัฒนพรพรหม
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษา	รองศาสตราจารย์ ดร.ประภาส จงสถิตย์วัฒนา
อาจารย์ที่ปรึกษาร่วม	อาจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สมศักดิ์ ปัญญาแก้ว)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.บุญเสริม กิจศิริกุล)

..... อาจารย์ที่ปรึกษา
(รองศาสตราจารย์ ดร.ประภาส จงสถิตย์วัฒนา)

..... อาจารย์ที่ปรึกษาร่วม
(อาจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์)

..... กรรมการ
(อาจารย์ ดร.วีระ เหมืองสิน)

วรินทร์ วัฒนพรพรหม : ระบบการประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์. (A PARALLEL PROCESSING SYSTEM FOR A BLAST PROGRAM) อ. ที่ปรึกษา : รศ.ดร. ประภาส จงสถิตย์วัฒนา, อ.ที่ปรึกษาร่วม : อ.ดร.ณัฐวุฒิ หนูไพโรจน์, จำนวนหน้า 91 หน้า. ISBN 974-17-1195-6.

บลาสท์เป็นโปรแกรมเปรียบเทียบลำดับของโปรตีนที่ใช้กันอย่างแพร่หลายที่สุดโปรแกรมหนึ่ง ซึ่งใช้หาลำดับข้อมูลพันธุกรรมหรือโปรตีนที่ค้นพบใหม่กับนิวคลีโอไทด์หรือกรดอะมิโนที่อยู่ในฐานข้อมูลที่เหมือนกันอย่างมีนัยสำคัญทางสถิติ โดยการเปรียบเทียบดังกล่าวช่วยให้นักวิทยาศาสตร์สามารถอนุมานเกี่ยวกับโครงสร้างและหน้าที่ของโปรตีนที่ถูกค้นพบ หรือเพื่อเลือกลำดับของโปรตีนใหม่ๆ เพื่อใช้ในการค้นคว้าต่อไป ถึงแม้ว่าโปรแกรมบลาสท์ได้ถูกออกแบบเพื่อความเร็วแล้วก็ตาม จุดอ่อนที่สำคัญที่สุดของโปรแกรมบลาสท์คือโปรแกรมใช้ทรัพยากรระบบอย่างมากไม่ว่าจะเป็นเวลาของหน่วยประมวลผลกลาง หน่วยความจำ หรือการรับเข้า/ส่งออกของข้อมูล โดยโปรแกรมบลาสท์ใช้เวลานานมากในการเปรียบเทียบข้อคำถามจำนวนมากกับฐานข้อมูลขนาดใหญ่ การศึกษาเบื้องต้นเกี่ยวกับโปรแกรมบลาสท์ชี้ชัดว่าเวลาในการทำงานเป็นสัดส่วนกับขนาดของฐานข้อมูลที่ใช้ในการเปรียบเทียบ โดยโปรแกรมบลาสท์จะมีประสิทธิภาพสูงสุดเมื่อฐานข้อมูลทั้งหมดอยู่ในหน่วยความจำหลักขณะประมวลผล เนื่องจากฐานข้อมูลทางพันธุศาสตร์มีขนาดมหาศาลและจะเพิ่มขนาดขึ้นเป็นสองเท่าในทุกๆ 1.3 ปีเพราะฉะนั้นการปรับปรุงความสามารถของโปรแกรมในหน่วยความจำที่จำกัดจึงเป็นสิ่งที่สำคัญ

วิทยานิพนธ์นี้เสนอการตัดแบ่งฐานข้อมูลออกเป็นส่วนเล็กๆ จนอยู่ในหน่วยความจำหลักได้ และทำการเปรียบเทียบลำดับของโปรตีนกับฐานข้อมูลที่ถูกแบ่งด้วยหน่วยย่อยของระบบประมวลผลเชิงขนาน จากนั้นจึงทำการเชื่อมต่อผลลัพธ์ที่ได้จากเปรียบเทียบเข้าด้วยกัน พบว่าระบบที่ได้ทำการพัฒนาขึ้นสามารถขจัดการสับเปลี่ยนค่าข้อมูล จะเห็นได้จากอัตราการเร่งความเร็วในระดับซูเปอร์ลีเนียร์

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต.....
สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษา.....
ปีการศึกษา 2545 ลายมือชื่ออาจารย์ที่ปรึกษาร่วม.....

4370480421 : MAJOR COMPUTER SCIENCE

KEY WORD: PARALLEL SYSTEM / MEMORY LIMITED / BLAST / DISTRIBUTED DATABASE / CLUSTERING

WARIN WATTANAPORNPROM : THESIS TITLE. (A PARALLEL SYSTEM FOR A BLAST PROGRAM) THESIS ADVISOR : ASSOC.PROF PRABHAS CHONGSTITVATANA, THESIS COADVISOR : DR.NATAWUT NUPAIROJ, 91 pp. ISBN 974-17-1195-6.

BLAST (Basic Local Alignment Search Tool) is one of the most widely used search tools, which identifies statistically significant matches between newly sequenced segments of genetic material or proteins and databases of known nucleotide or amino acid sequences. Such searches allow scientists to make inferences about the structures and functions of their discoveries or to screen new sequences for further investigation. Although BLAST has been designed and optimized for speed, the major drawback of BLAST is that it consumes large amount of CPU-time, memory, and I/O. It takes a very long time to search a large number of queries in a large database. Preliminary study of BLAST indicates that BLAST's running time is proportional to the size of the database. BLAST shows the highest efficiency if the whole database can be fitted in the memory. As the genome databases are enormous and doubling in size every 1.3 years, it is important to improve the performance in the limited main memory environment.

This thesis proposes to separate the database into smaller parts, which each part fits the available memory and then search each part separately in each node of a parallel system. The implemented system can eliminate swapping time as shown in the result of superlinear speedup.

Department Computer Engineering Student's signature.....
Field of study Computer Science Advisor's signature…….....
Academic year 2002 Co-advisor's signature.....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี ด้วยความช่วยเหลืออย่างดียิ่งจาก รศ.ดร.ประภาส จงสถิตวัฒนา และ อ.ดร.ณัฐวุฒิ หนูไพโรจน์ ซึ่งได้สละเวลาในการให้คำปรึกษาตลอดจนช่วยตรวจแก้ไขวิทยานิพนธ์ด้วยความเอาใจใส่อย่างดียิ่ง รวมทั้ง ศ.น.พ.สิริฤกษ์ ทรงศิวิไล และ อ.ดร.นพ.บดินทร์ ทรัพย์สมบูรณ์ จากคณะแพทย์ศิริราชที่ได้เอื้อเฟื้อข้อมูลที่ใช้ในการทดสอบตลอดจนเพื่อนๆ ทุกคนที่เอื้อเฟื้อคอมพิวเตอร์สำหรับใช้ในการทดสอบโปรแกรม วิทยานิพนธ์ฉบับนี้จะไม่สามารถสำเร็จลุล่วงไปได้เลยหากจะขาดเสียซึ่งบุคคลใดบุคคลหนึ่ง

ท้ายนี้ผู้วิจัยใคร่ขอกราบขอบพระคุณ บิดา มารดา รวมถึงทุกคนในครอบครัวซึ่งให้การสนับสนุนและกำลังใจแก่ผู้วิจัยเสมอมาจนสำเร็จการศึกษา

สารบัญ

หน้า

สารบัญตาราง

สารบัญรูปภาพ(ต่อ)

ภาพประกอบ หน้า

ภาพประกอบ หน้า

บทที่ 1

บทนำ

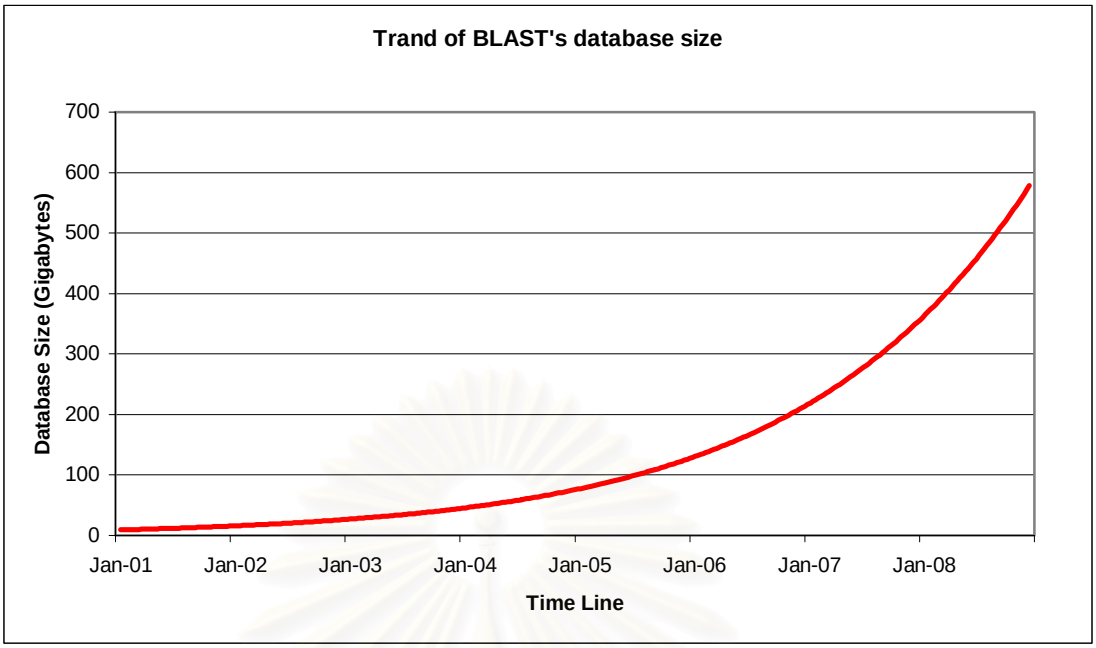
1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบันพัฒนาการทางคอมพิวเตอร์และอินเตอร์เน็ตได้รับการพัฒนาจนก้าวหน้าไปไกลมาก ทำให้มีขีดความสามารถในการประมวลผลข้อมูลสูงขึ้นกว่าเดิม จากประสิทธิภาพที่ดีขึ้นมากของคอมพิวเตอร์ ประจวบกับการสร้างเครื่องมือเพื่อศึกษาสิ่งมีชีวิตในระดับโมเลกุลได้มากขึ้น การผสมผสานศาสตร์ทางด้านคอมพิวเตอร์กับชีววิทยาจึงเกิดขึ้น โดยเฉพาะอย่างยิ่งการถอดรหัสพันธุกรรม ซึ่งมีข้อมูลทางพันธุกรรมมากมายที่จะต้องถอดรหัส และต้องใช้ขีดความสามารถในการคำนวณและเทคนิคที่สูงมาก

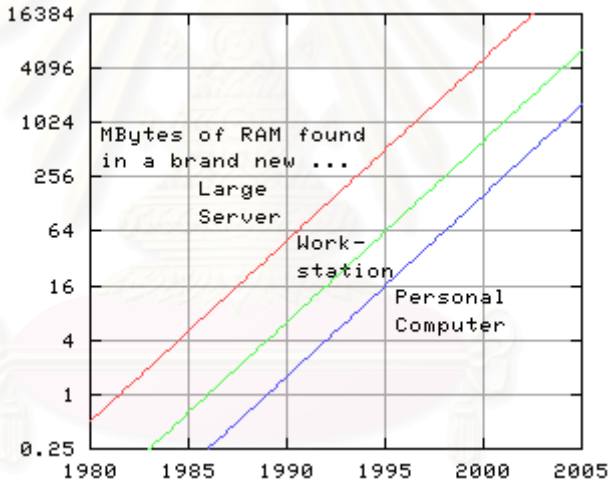
นักวิทยาศาสตร์ในหลาย ๆ ประเทศให้ความสำคัญกับการทำวิจัยที่เกี่ยวข้องกับโครงการทางด้านการถอดรหัสพันธุกรรมของสิ่งมีชีวิตต่าง ๆ ตัวอย่างที่เห็นได้ชัด เช่น โครงการฮิวแมนจีโนม (Human Genome Project - HGP) เป็นโครงการระหว่างประเทศที่ร่วมมือกันทำงานหลายองค์กร โดยใช้เงินลงทุนจำนวนมาก ใช้นักวิทยาศาสตร์ นักชีววิทยา นักคอมพิวเตอร์ มาร่วมกันศึกษาและวิเคราะห์ โดยช่วยกันทำการจัดทำข้อมูลสายรหัสพันธุกรรมและรวบรวมเก็บเป็นฐานข้อมูลกลาง ที่ถูกบรรจุลง (download) เพื่อใช้ร่วมกัน การถอดรหัสพันธุกรรมนั้น เป็นเรื่องที่สำคัญและจะให้ประโยชน์แก่มวลมนุษย์ชาติอย่างมหาศาล เพราะรหัสพันธุกรรมที่ได้จะเกี่ยวข้องกับการรู้ถึงการเกิดโรคต่าง ๆ เกี่ยวพันกับการผลิตยารักษาโรค การปรับปรุงพันธุ์ การโคลนนิ่ง ตลอดจนการใช้ประโยชน์ในเรื่องการสร้างอาหาร ตลอดจนถึงปัจจัยสี่อื่นๆ

บลาสท์ (BLAST - Basic Local Alignment Search Tool) เป็นโปรแกรมที่ใช้กันอย่างแพร่หลายที่สุดโปรแกรมหนึ่งในวงการแพทย์ ถูกพัฒนาโดยศูนย์สารสนเทศเทคโนโลยีชีวภาพแห่งชาติ (National Center for Biotechnology Information) หรือ เอ็นซีบีไอ (NCBI) ประเทศสหรัฐอเมริกา มีความสามารถในการแยกแยะ และเปรียบเทียบลำดับ (sequence) ของโปรตีนที่เกี่ยวกับพันธุศาสตร์เช่น ยีน (gene) หรือโครโมโซม (chromosome) ต่างๆ โดยทำการเปรียบเทียบทีละคู่ (pairwise sequence alignment) กับฐานข้อมูลของนิวคลีโอไทด์ (nucleotide) หรือกรดอะมิโน (Amino Acid) ต่างๆ เพื่อช่วยในการอนุมานโครงสร้างและหน้าที่ของลำดับโปรตีนดังกล่าวว่าเป็นยีนอะไรและทำหน้าที่อะไร หรือเพื่อกรองข้อมูลลำดับของโปรตีนใหม่ๆ เพื่อการค้นคว้าทางการแพทย์ต่อไป

จากการศึกษาเบื้องต้นพบว่าประสิทธิภาพของโปรแกรมบลาสท์ขึ้นอยู่กับขนาดของฐานข้อมูลทางพันธุศาสตร์ ทั้งนี้การทำงานของโปรแกรมบลาสท์นั้นช้าลงทุกวันเนื่องจากขนาดของฐานข้อมูลทางพันธุศาสตร์มีอัตราการโตขึ้นเป็นเท่าตัวในทุก ๆ 1.3 ปี [3] โดยจากรูปที่ 1.1 ขนาดของฐานข้อมูลพันธุศาสตร์จะมีขนาดสูงกว่า 1 เทราไบต์ภายในสิ้นปี ค.ศ. 2005 การใช้ข้อคำถาม (query) จำนวนมากทำการเปรียบเทียบกับฐานข้อมูลทางพันธุศาสตร์จะทำได้ยากเนื่องจากใช้เวลานานและมีความเสี่ยงสูงที่จะไม่สามารถทำการเปรียบเทียบข้อคำถามทั้งหมดให้เสร็จได้ ตลอดจนไม่สามารถติดตามผลของการเปรียบเทียบ ว่าข้อคำถามได้ทั้งหมดได้ถูกเปรียบเทียบกับฐานข้อมูลเสร็จไปแล้วเป็นจำนวนเท่าไร การเปรียบเทียบลำดับจากฐานข้อมูลที่มีขนาดเล็กบางตัวเช่นการเปรียบเทียบลำดับจากฐานข้อมูล NR ขนาด 234 เมกะไบต์ในเครื่องที่ใช้ตัวประมวลผลกลาง เอเอ็มดี ดูรอน (AMD Duron) ความเร็ว 650 เมกะเฮิร์ทซ์ และมีหน่วยความจำติดตั้ง 128 เมกะไบต์ใช้เวลาการเปรียบเทียบนานถึง 7 วัน 13 ชั่วโมง นอกจากนั้นการประเมินการเปรียบเทียบในฐานข้อมูลพันธุศาสตร์ที่ใหญ่กว่าเช่นฐานข้อมูล NT ขนาด 2.1 กิกะไบต์ด้วยจำนวนข้อคำถามที่เท่ากันและใช้เครื่องประสิทธิภาพเดียวกันจะใช้เวลาไม่น้อยกว่า 60 วัน จึงได้มีความพยายามมากมายในการเพิ่มประสิทธิภาพในการเปรียบเทียบของโปรแกรมบลาสท์ โดยการใช้เครื่องที่มีความเร็วของหน่วยประมวลผลที่สูงขึ้น [3][4][7] อย่างไรก็ดีความพยายามเหล่านี้ไม่สามารถเพิ่มประสิทธิภาพในการประมวลผลได้มากในระดับที่น่าพอใจนัก เนื่องมาจากการวิจัยต่างๆ ไม่ได้แก้ปัญหาที่ถูกจุด ซึ่งปัญหาที่แท้จริงเกิดจากขนาดของฐานข้อมูลไม่สามารถที่จะเก็บอยู่ในหน่วยความจำหลักในขณะประมวลผลได้ทั้งหมดทำให้การทำงานของโปรแกรมกินเวลานานเนื่องจากปัญหาคอขวดจากการสับค่า (Swap) ข้อมูลระหว่างฮาร์ดดิสก์และหน่วยความจำหลักซึ่งในปัจจุบันระบบที่มีหน่วยความจำมากขนาดที่สามารถจุฐานข้อมูลได้มีน้อยมากและมีราคาสูง โดยจากรูปที่ 1.2 ในปี ค.ศ. 2005 แนวโน้มของขนาดของหน่วยความจำของเครื่องคอมพิวเตอร์ส่วนบุคคลจะมีขนาดเฉลี่ยที่ 1.5 กิกะไบต์ ในขณะที่ขนาดของฐานข้อมูลจากรูปที่ 1.1 ในปีเดียวกันจะมีขนาดเพิ่มขึ้นถึง 100 กิกะไบต์



รูปที่ 1.1 แนวโน้มของขนาดของฐานข้อมูลของโปรแกรม บลาสท์ ภายใน 8 ปี [3]



รูปที่ 1.2 แนวโน้มของขนาดหน่วยความจำที่จะมีใช้ในเครื่องคอมพิวเตอร์ภายในปี ค.ศ. 2005 [17]

จากปัญหาของความล่าช้าที่เกิดขึ้น จึงมีหลายหน่วยงานริเริ่มที่จะนำระบบการประมวลผลแบบขนาน (Parallel System) เข้ามาใช้ในการเพิ่มประสิทธิภาพในการประมวลผลของโปรแกรม โดยการตัดแบ่งข้อคำถามออกเป็นข้อคำถามย่อยๆเพื่อทำการเปรียบเทียบโดยการส่งข้อคำถามขนาดเล็กที่ตัดได้กระจายไปตามหน่วยประมวลผลต่างๆ ของระบบเพื่อทำการเปรียบเทียบ ทั้งนี้ผลของความเร็วที่ได้จากการตัดแบ่งข้อคำถามไม่ต่างจากผลที่ได้การเปรียบเทียบข้อคำถามของโปรแกรมบลาสท์ในหลายๆเครื่อง ซึ่งไม่อาจเพิ่มประสิทธิภาพของโปรแกรมได้เท่าที่ควร

จากการเพิ่มขนาดของหน่วยความจำหลักพบว่า สามารถทำการเพิ่มประสิทธิภาพของการเปรียบเทียบ โดยการศึกษาเบื้องต้นพบว่าการเปรียบเทียบลำดับจากฐาน ข้อมูลที่ชื่อ NR ในเครื่องที่มีหน่วยความจำติดตั้ง 256 เมกะไบต์ เราสามารถใช้เวลาน้อยลง 2 เท่า ในขณะที่เราเพิ่มขนาดของหน่วยความจำขึ้นเป็น 512 เมกะไบต์ เรากลับไม่สามารถลดเวลาในการเปรียบเทียบลำดับจากเครื่องที่ใช้หน่วยความจำขนาด 256 เมกะไบต์ได้เลย

งานวิจัยนี้มีแนวคิดที่จะพัฒนาสภาพแวดล้อมการประมวลผลแนวขนาน โดยการตัดแบ่งฐานข้อมูลออกเป็นส่วยย่อยๆ เพื่อทำการเปรียบเทียบตามหน่วยประมวลผลต่าง ๆ ของระบบ เพื่อลดปัญหาความซ้ำซ้อนของการสับเปลี่ยนข้อมูลระหว่างหน่วยความจำหลักและฮาร์ดดิสก์โดยจะเน้นทำการออกแบบระบบให้เหมาะสม กับการประมวลผลของระบบที่จำนวนหน่วยประมวลผลต่าง ๆ กันเพื่อเพิ่มประสิทธิภาพสูงสุด

1.2 วัตถุประสงค์ของการวิจัย

เพื่อออกแบบและพัฒนาระบบประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์

1.3 ขอบเขตของการวิจัย

1. งานวิจัยนี้จะทำการทดลองกับโปรแกรมบลาสท์เวอร์ชัน 2.0 ของเอ็นซีบีไอเท่านั้น
2. งานวิจัยนี้จะทำการทดลองกับฐานข้อมูลเพียงสองตัวคือ NT และ NR ซึ่งหาได้ในประเทศไทย
3. งานวิจัยนี้จะทำการทดลองกับข้อคำถามที่อยู่ในรูปแบบฟาสตา เท่านั้น
4. งานวิจัยนี้จะเน้นทำการออกแบบระบบตัวอย่างที่เหมาะสมกับการประมวลผลในแนวขนานที่จำนวนหน่วยประมวลผลต่างๆกันไป

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. เพิ่มประสิทธิภาพ ในการใช้งานโปรแกรมบลาสท์
2. ลดความซ้ำซ้อนในการสับเปลี่ยนข้อมูลระหว่างหน่วยความจำ ตลอดจนช่วยยืดอายุการใช้งานของอุปกรณ์คอมพิวเตอร์
3. ช่วยเพิ่มประสิทธิภาพและประสิทธิผลที่เร็วขึ้นในการทำวิจัยในด้านเทคโนโลยีชีวภาพ

1.5 ขั้นตอนในการดำเนินการวิจัย

1. ศึกษาวิธีการใช้งานโปรแกรมบลาสท์
2. จำแนกปัญหา วิเคราะห์ความรู้ และตั้งสมมุติฐาน
3. ทดสอบสมมุติฐาน
4. วิเคราะห์และออกแบบระบบ
5. พัฒนาโปรแกรมต้นแบบ
6. ประเมินผลและปรับปรุงระบบ
7. สรุปผลการวิจัยและข้อเสนอแนะ พร้อมจัดทำวิทยานิพนธ์

บทที่ 2

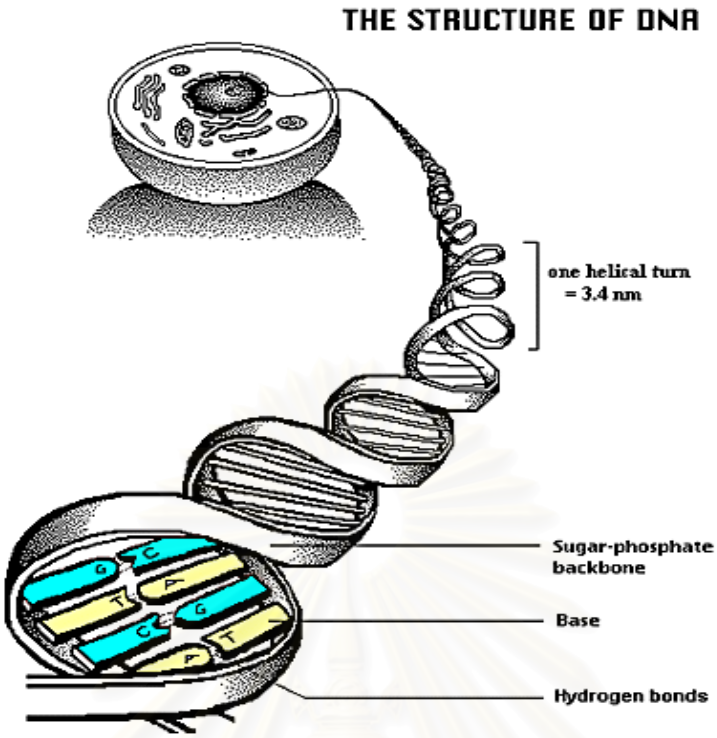
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 พันธุศาสตร์โมเลกุล[18]

โครโมโซม (Chromosome) คือส่วนประกอบในนิวเคลียส (nucleus) ของเซลล์ ซึ่งมียีน (Gene) เป็นจำนวนมากกระจายอยู่ โครโมโซมนั้นประกอบด้วยเส้นใยบางๆ ของสารนิวคลีโอโปรตีน หรือที่เรียกว่า เส้นใยโครมาติน โครมาตินนี้ประกอบด้วยโมเลกุลของดีเอ็นเอและโปรตีน ดีเอ็นเอเป็นกรดนิวคลีอิกชนิดหนึ่งพบมากในนิวเคลียส ต่างกับอาร์เอ็นเอซึ่งเป็นกรดนิวคลีอิกอีกชนิดที่ใกล้เคียงดีเอ็นเอมาก แต่จะพบมากในไซโตพลาสซึม

2.1.1 โมเลกุลดีเอ็นเอ (DNA)

โมเลกุลของดีเอ็นเอที่อยู่บนโครโมโซมมีความยาวมาก เมื่ออยู่ในโครโมโซมจึงอยู่ในลักษณะที่ขมวดเข้าด้วยกัน แต่ละโมเลกุลของดีเอ็นเอแบ่งออกเป็นหน่วยย่อย ๆ เรียกว่า นิวคลีโอไทด์ยูนิต (nucleotide unit) ซึ่งประกอบไปด้วยหมู่ฟอสเฟต น้ำตาลดีออกซีไรโบส (deoxyribose) และสารประกอบไนโตรเจนที่เรียกว่านิวคลีโอไทด์ (nucleotides) นิวคลีโอไทด์ที่จับตัวกันอยู่บนดีเอ็นเอนี้เรียกว่า เบส (base) เบสในดีเอ็นเอมีเพียง 4 ชนิดได้แก่ กัวนีน (guanine) ไซโตซีน (cytosine) ไทมีน (thymine) และอะดินีน (adenine) นิยมเรียกย่อ ๆ ว่า G C T และ A ตามลำดับ ดีเอ็นเอมีลักษณะเป็นเส้นนิวคลีโอไทด์ 2 เส้นพันกันแบบเชือกฟั่น บางครั้งจึงนิยมเรียกโมเลกุลของดีเอ็นเอว่า เชือกดีเอ็นเอ (DNA strand) เส้นนิวคลีโอไทด์เกิดจากการจับตัวกันของเบสกับน้ำตาลดีออกซีไรโบส น้ำตาลดีออกซีไรโบสจับกับหมู่ฟอสเฟต หมู่ฟอสเฟตจับกับน้ำตาลดีออกซีไรโบส แล้วน้ำตาลดีออกซีไรโบสก็จับกับเบสอีก เป็นเช่นนี้เรื่อยไปแบบลูกโซ่ ส่วนการเกิดเป็นเส้นคู่นั้น เนื่องจากเบสของเส้นหนึ่งจับกับเบสของอีกเส้นหนึ่งด้วยไฮโดรเจนบอนด์ โดยที่กัวนีน (หรือ G) จับคู่กับ ไซโตซีน (หรือ C) และ อะดินีน (หรือ A) จับคู่กับ ไทมีน (หรือ T) แต่ละคู่นี้เรียกว่าคู่เบส (base pair) เป็นคู่ที่แน่นอน ไม่มีการสับคู่ ดังแสดงในรูปที่ 2.1



รูปที่ 2.1 โครงสร้างของดีเอ็นเอ

นอกจากที่ดีเอ็นเอจะมีการเก็บข้อมูลที่จำเป็นในการสร้างและการดำรงชีวิตแล้ว ดีเอ็นเอยังมีประโยชน์ในการใช้เป็นแม่แบบในการสร้างโมเลกุลอาร์เอ็นเออีกด้วย โดยการคลายเกลียวสายโพลีนิวคลีโอไทด์ของดีเอ็นเอออก เพื่อเป็นแม่แบบให้นิวคลีโอไทด์โมเลกุลเดี่ยวเข้ามาจับคู่กับเบสบนสายโพลีนิวคลีโอไทด์เดิม แต่ในขบวนการนี้ อะดินีน (A) จะจับคู่กับ ยูราซีล (U) และกัวนีน (G) จับคู่กับ ไซโตซีน (C) และจะไม่ใช้สายนิวคลีโอไทด์ของดีเอ็นเอทั้งสองเส้นพร้อมกัน แต่จะใช้สายดีเอ็นเอต้นแบบเพียงเส้นเดียวเป็นช่วงๆ ตามความจำเป็นของการใช้อาร์เอ็นเอชนิดนั้น ๆ และเนื่องจากโมเลกุลของอาร์เอ็นเอถูกสร้างขึ้นมาจากสายดีเอ็นเอเพียงข้างเดียว อาร์เอ็นเอจึงเป็นสายโพลีนิวคลีโอไทด์สายเดียว ซึ่งอาร์เอ็นเอนี้จะนำเอาไปใช้ประโยชน์ในการสังเคราะห์โปรตีนต่อไป

2.1.2 อาร์เอ็นเอ

อาร์เอ็นเอ แบ่งออกได้เป็น 3 ชนิด ตามลักษณะโครงสร้างและหน้าที่การทำงาน ดังนี้ คือ tRNA rRNA และ mRNA

2.1.2.1 tRNA (transfer RNA) มีขนาดเล็กมาก ถึงแม้ tRNA จะถูกสร้างขึ้นมาโดยลอกลำดับเบสจากดีเอ็นเอเช่นเดียวกับอาร์เอ็นเอชนิดอื่นๆ แต่เบสของมันมีการเปลี่ยนแปลงเพิ่มเติมมากมาย เพื่อให้มีหน้าที่เฉพาะเจาะจงในขบวนการสังเคราะห์โปรตีน กล่าวคือ ปลายด้านหนึ่งของ tRNA จะมีแอนติโคดอน (anticodon) ซึ่งเป็นเบสที่จับคู่กันได้กับเบสที่เป็นรหัส หรือโคดอน (Codon) บนสาย mRNA ส่วนปลายอีกด้านของ tRNA ทำหน้าที่พากรดอะมิโนเฉพาะสำหรับรหัสดังกล่าวเพื่อนำมาเชื่อมต่อเป็นสายโพลีเปปไทด์ ในการสังเคราะห์โปรตีนนั่นเอง

2.1.2.2 rRNA (ribosomal RNA) รวมกันเข้ากับโปรตีนประกอบเป็นไรโบโซม (ribosome)

2.1.2.3 mRNA (messenger RNA) เป็นอาร์เอ็นเอ ที่มีบทบาทสำคัญที่สุดในการกำหนดลำดับของกรดอะมิโนที่มาเรียงตัวกันเป็นโพลีเปปไทด์ ทั้งนี้เพราะ mRNA เป็นตัวรับคำสั่งจากดีเอ็นเอ โดยถอดรหัสในรูปของลำดับเบสที่เรียงตัวกันอยู่ในดีเอ็นเอ ขบวนการถ่ายทอดรหัสนี้เรียกว่า ทรานสคริปชัน (transcription)

2.2 **วิธีที่ใช้ในการค้นหายีน** [18]

หลังจากได้ข้อมูลที่จะใช้สอนและทดสอบโปรแกรมแล้วจะต้องเลือกวิธีที่ใช้ในการค้นหายีน สำหรับวิธีโดยทั่วไปที่ใช้ในการค้นหายีนมีทั้งหมด 3 วิธี คือ

1. วิธีการเปรียบเทียบลำดับสองเส้นระหว่าง สายนิวคลีโอไทด์ใหม่ กับสายนิวคลีโอไทด์ที่รู้จักแล้วหรือสายนิวคลีโอไทด์ที่หาโปรตีนไว้แล้วซึ่งจะมาจากฐานข้อมูลโปรตีน (Search by Sequence Similarity) โปรแกรม BLAST ใช้วิธีการเปรียบเทียบลำดับสองเส้นในการค้นหายีน
2. วิธีหาบริเวณที่เกี่ยวข้องกับการควบคุมการแสดงออกของยีน (Search by Signal) วิธีนี้จะหาส่วนต่าง ๆ ที่สำคัญในยีน เช่น ส่วนโปรโมเตอร์ (Promoter) บริเวณที่มีไรโบโซมมาจับ(Ribosome Binding site) บริเวณที่ควบคุมการสร้างโปรตีน (regulatory elements) จุดเริ่มต้นของยีน (Start Codon) ลำดับเบสซ้ำ (repetitive sequence) บริเวณที่เป็น CpG Islands บริเวณที่มีการเติม poly-A เป็นต้น เพื่อที่จะใช้ลักษณะพิเศษเหล่านี้ช่วยในการค้นหายีน
3. วิธีหาคุณสมบัติทางสถิติของลำดับเบสในสายดีเอ็นเอ (Search by Content) วิธีนี้จะใช้ค่าสถิติเกี่ยวกับคุณสมบัติของดีเอ็นเอ เช่น หาความถี่ในการเกิดเบสที่ตำแหน่งของโคดอนต่างๆ ความแตกต่างเกี่ยวกับความถี่ในการเกิดกรดอะมิโนชนิดต่างๆ สัดส่วนการใช้โคดอน (Codon preference) เป็นต้น ซึ่งค่าเหล่านี้จะแตกต่างกันไปในสิ่งมีชีวิตที่ต่างชนิดกัน

2.2.1 **หลักการเปรียบเทียบลำดับสองเส้น**

ความคล้ายคลึงกัน (Similarity) ของลำดับของสายดีเอ็นเอหรือโปรตีนสองสาย มีสาเหตุที่เป็นไปได้ 2 ประการคือ

1. ดีเอ็นเอหรือโปรตีนของสิ่งมีชีวิตทั้งสองชนิดมีต้นกำเนิดมาจากแหล่งเดียวกัน จากนั้นดีเอ็นเอหรือโปรตีนแต่ละสายเกิดการเปลี่ยนแปลงในระหว่างที่สิ่งมีชีวิตแต่ละชนิดมีวิวัฒนาการ ซึ่งการเปลี่ยนแปลงที่เกิดขึ้นในลำดับของสายดีเอ็นเอหรือโปรตีนมีไม่มากนัก เมื่อทำการวิเคราะห์จะยังพบว่าดีเอ็นเอหรือโปรตีนทั้งสองสายมีความคล้ายคลึงกัน
2. บริเวณของดีเอ็นเอหรือโปรตีนที่คล้ายคลึงกันนั้น มีหน้าที่ทางชีววิทยาเหมือนกัน วิวัฒนาการจึงทำให้ลำดับในสิ่งมีชีวิตแต่ละชนิดมีความคล้ายคลึงกัน (convergent evolution)

ในการพิจารณาว่าลำดับดีเอ็นเอหรือโปรตีน 2 สายสามารถเทียบเคียงกันได้หรือไม่ (homologous)นอกจากความคล้ายคลึงกันของลำดับทั้งสองสายแล้ว จะต้องพิจารณาว่าลำดับทั้งสองสายมีต้นกำเนิดมาจากแหล่งเดียวกันหรือไม่ ลำดับที่มีความคล้ายคลึงกันอาจไม่ สามารถเทียบเคียงกันได้หรือไม่ ลำดับสายสั้นๆอาจมีความคล้ายคลึงกันโดยบังเอิญ ตัวอย่างเช่นถ้าเบสของสายดีเอ็นเอมีการกระจายตัวอย่างอิสระไม่ขึ้นแก่กัน โอกาสที่จะพบชิ้นดีเอ็นเอขนาด 10 คู่เบสที่มีลำดับเบสที่จำเพาะมีค่าความน่าจะเป็นเท่ากับ 1 ใน 4^{10} หรือประมาณ 10^{-6} ซึ่งนับเป็นค่าที่สูงเพียงพอที่จะพบลำดับเบสดังกล่าวในจีโนมของแบคทีเรียส่วนใหญ่ (ซึ่งมีขนาดประมาณ10^{6} คู่เบสขึ้นไป) ในขณะที่โอกาสที่จะพบชิ้นดีเอ็นเอขนาด 1000 คู่เบสที่มีลำดับเบสที่จำเพาะมีค่าความน่าจะเป็นเท่ากับ 1 ใน 4^{1000} หรือประมาณ 10^{-600} ซึ่งเป็นค่าที่ต่ำมาก ดังนั้นถ้าดีเอ็นเอขนาด 1000 คู่เบสสองสายมีลำดับเบสที่เหมือนกัน หรือคล้ายคลึงกัน มีโอกาสสูงมากๆที่ดีเอ็นเอสองสายนี้มีต้นกำเนิดมาจากแหล่งเดียวกัน

2.2.2 **หลักการเปรียบเทียบลำดับสองเส้นโดยใช้วิธีการเขียนโปรแกรมแบบพลวัต** (dynamic programming)

วิธีการตรวจสอบความคล้ายคลึงกันของลำดับเบสหรือกรดอะมิโนอีกวิธีหนึ่งคือ นำลำดับที่ต้องการเปรียบเทียบมาเรียงขนานคู่กัน (align) ไปโดยพยายามจัดให้คล้ายกันมากที่สุดเท่าที่จะเป็นไปได้ ในการนี้บางครั้งอาจต้องยอมให้ลำดับเบสที่ไม่ตรงกันอยู่คู่กันบ้าง หรือยอมให้เกิดช่องว่างขึ้นบ้างเพื่อให้ส่วนที่คล้ายกันได้อยู่คู่กัน เมื่อเปรียบเทียบลำดับเบส 2 สาย ผลที่เป็นไปได้ในแต่ละตำแหน่งคือ 1) มีเบสหรือกรดอะมิโนเหมือนกัน (match) 2) เบสหรือกรดอะมิโนที่ตำแหน่งนั้นต่างกัน (mismatch) และ 3) เกิดช่องว่าง (gap) ในลำดับสายหนึ่ง ณ ตำแหน่งนั้น การเปรียบเทียบลำดับ 2 สายจะมีการเรียงตัวในรูปแบบที่เป็นไปได้เป็นจำนวนมาก ต้องทำการเปรียบเทียบในการเรียงตัวทุกรูปแบบที่เป็นไปได้ แล้วคัดเลือกการเรียงตัวรูปแบบที่ดีที่สุด โดยทั่วไปก็จะมีปัญหาแรกว่า จะตัดสินอย่างไรว่ารูปแบบการเรียงแบบใดที่มีลักษณะใกล้เคียงกันมากที่สุด การจะตัดสินใจได้ก็จะต้องมีการให้คะแนนผลการเรียงทั้งสามชนิดที่กล่าวข้างต้น แล้วรวมเข้าด้วยกันตลอดทั้งเส้นหรือบริเวณที่สนใจจึงจะบอกได้ เช่น อาจให้คะแนนสำหรับเบสที่ตรงกันมีค่าเป็นบวก คะแนนสำหรับเบสที่มีไม่ตรงกันมีค่าเป็นศูนย์หรือลบ และคะแนนค่าปรับสำหรับช่องว่าง (gap penalty) ดังนั้นการเรียงตัวแต่ละรูปแบบจะมีคะแนนความคล้าย (similarity score) ที่แตกต่างกัน การเรียงตัวใน รูปแบบที่มีคะแนนสูงสุดจะถือว่าเป็น การเรียงตัวที่ดีที่สุด

อย่างไรก็ตามการคัดเลือกการเรียงตัวที่ดีที่สุด ด้วยการหาคะแนนไปเรื่อยๆทีละคู่นั้นทำได้ในกรณีที่สายดีเอ็นเอมีขนาดค่อนข้างสั้นเท่านั้น ตัวอย่างเช่นถ้าต้องการเปรียบเทียบดีเอ็นเอที่มีขนาด 1000 คู่เบส รูปแบบการเรียงตัวที่เป็นไปได้มีประมาณ 10^{600} แบบ ซึ่งมากกว่าจำนวนอะตอมทั้งหมดในเอกภพเสียอีก การคิดคำนวณคะแนนของรูปแบบของการเรียงตัวที่เป็นไปได้ทั้งหมดโดยการไล่ทำไปเรื่อยๆจึงใช้เวลานานมากเกินไป ดังนั้นจึงจำเป็นจะต้องใช้เครื่องมือและวิธีการทางคอมพิวเตอร์ช่วย อัลกอริทึมการเขียนโปรแกรมแบบพลวัต ซึ่งถูก Needleman และ Wunsch นำมาใช้ เป็นวิธีการทางคอมพิวเตอร์ที่สามารถช่วยหารูปแบบการเรียงตัวที่ดีที่สุด โดยไม่ต้องคิดคำนวณคะแนนความคล้ายของการเรียงตัวทุกแบบ โดยที่สามารถพิสูจน์ได้ว่า วิธีนี้ให้ผลการเรียงตัวที่ดีที่สุดถูกต้องเสมอ

2.2.3 การเปรียบเทียบดีเอ็นเอ 2 สายด้วยวิธี Needleman และ Wunsch [8]

ทำโดยการเรียงลำดับเบสหรือกรดอะมิโนสายที่หนึ่งในแนวนอนแถวแรกของตาราง และเรียงลำดับของสายที่สองในแนวตั้งแถวแรกของตาราง จากนั้นจะให้คะแนนในแต่ละช่องตาราง (cell) จากแถวบนสุดด้านซ้าย โดยคะแนนในแต่ละช่องจะเป็นคะแนนที่สูงที่สุดของคะแนนดังต่อไปนี้

1. คะแนนของช่องด้านบนบวกกับค่าปรับที่เกิดจากช่องว่าง
2. คะแนนของช่องด้านซ้ายบวกกับค่าปรับที่เกิดจากช่องว่าง
3. คะแนนของช่องด้านบนซ้ายที่ติดกันในแนวทแยง รวมกับคะแนนเบสที่เหมือนกัน (match score) ถ้าเบสในแนวตั้งและแนวนอนเป็นเบสชนิดเดียวกัน หรือรวมกับคะแนนเบสที่ไม่เหมือนกัน (mismatch score) ถ้าเบสในแนวตั้งและแนวนอนเป็นเบสต่างชนิดกัน

คะแนนจะถูกคิดคำนวณในทุกช่องบนตาราง โดยคะแนนที่สูงที่สุดของแถวสุดท้ายในแนวตั้งหรือแนวนอนจะเป็นคะแนนของการเรียงตัวที่ดีที่สุด จากตัวอย่างในตารางที่ 2.1 คะแนนที่สูงที่สุดคือ 10 ส่วนรูปแบบการเรียงตัวที่ดีที่สุดจะเป็นไปตามแนวของช่องที่มีคะแนนที่สูงที่สุด ดังที่แสดงในตารางที่ 2.1 ในช่องที่ขีดเส้นใต้

ในการกำหนดคะแนนนั้น จะกำหนดให้ค่าปรับที่เกิดจากช่องว่าง มีค่าสัมบูรณ์มากกว่าคะแนนที่เบสตรงกัน ทั้งนี้เนื่องจากโอกาสที่จะเกิด การสอดแทรก (insertion) หรือการลบ (deletion) ของเบสในสายดีเอ็นเอ (ซึ่งจะทำให้มี ช่องว่างในการทำ การเรียงตัว) มีน้อยกว่าค่าของการแปลงรูป (point mutation) การกำหนดค่าคะแนนจะมีผลต่อรูปแบบการจัดเรียงในการเรียงตัวที่ดีที่สุด เช่นถ้ากำหนดค่าค่าปรับที่เกิดจากช่องว่าง สูงมาก จะทำให้ การเรียงตัวที่ดีที่สุดมีช่องว่างจำนวนน้อย แต่อาจมี ข้อแตกต่างจำนวนมาก หากกำหนดค่าปรับที่เกิดจากช่องว่าง ต่ำมาก จะทำให้การเรียงตัวที่ดีที่สุดมีช่องว่างจำนวนมากแทน

จากตัวอย่างในตารางที่ 2.1 พบการเรียงตัวที่ดีที่สุด 3 แบบ แบบที่ 2 ซึ่งมีลักษณะเป็นการลบเพียง 1 เบส 2 ตำแหน่งที่ใกล้กัน น่าจะมีโอกาสเกิดขึ้นได้น้อยกว่าแบบที่ 1 และแบบที่ 3 ในกรณีเช่นนี้การกำหนดค่าค่าปรับที่เกิดจากช่องว่างให้เป็นฟังก์ชั่นที่มีความสัมพันธ์กับความยาวของช่องว่างจะทำให้ช่องว่างเล็ก ๆ รวมกันมีแนวโน้มจะรวมกันเป็นช่องว่างที่ยาวขึ้น

วิธีการหนึ่งที่นิยมใช้กันมากในการกำหนดฟังก์ชั่นของค่าปรับที่เกิดจากช่องว่างคือ การกำหนดให้มีคะแนนเป็น ฟังก์ชันเชิงเส้น (linear function) ของความยาวของช่องว่าง โปรแกรมที่ทำการเรียงตัวส่วนมากจะอนุญาตให้ทำการกำหนด ค่าปรับที่เกิดจากช่องว่าง ซึ่งเป็นคะแนนลบที่เกิดขึ้นทุกครั้งที่เกิดช่องว่างและ ค่าปรับที่เกิดจากความกว้างของช่องว่าง (gap extension penalty) ซึ่งเป็นคะแนนที่จะคูณกับความกว้างก่อนจะนำไปลบจาก คะแนนความคล้าย

ตารางที่ 2.1 การเปรียบเทียบดีเอ็นเอ ATGCCCTGACCGGAATGCC กับดีเอ็นเอ ATGCTGACCCCAATGCC โดยวิธี Needleman-Wunsch algorithm

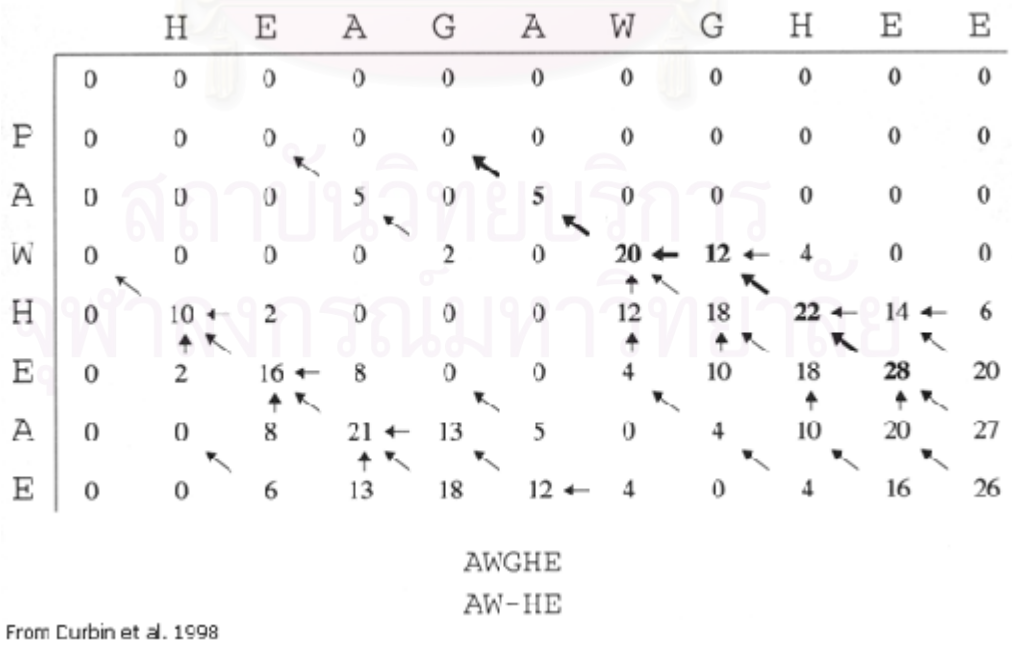
	A	T	G	C	C	C	T	G	A	C	C	G	G	A	A	T	G	C	C
A	1	-1	-3	-5	-7	-9	-11	-13	-15	-17	-19	-21	-23	-25	-27	-29	-31	-33	-35
T	-1	2	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22	-24	-26	-28	-30	-32
G	-3	0	3	1	-1	-3	-5	-7	-9	-11	-13	-15	-17	-19	-21	-23	-25	-27	-29
C	-5	-2	1	4	2	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22	-24	-26
T	-7	-4	-1	2	4	2	1	-1	-3	-5	-7	-9	-11	-13	-15	-17	-19	-21	-23
G	-9	-6	-3	0	2	4	2	2	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20
A	-11	-8	-5	-2	0	2	4	2	3	1	-1	-3	-5	-7	-9	-11	-13	-15	-17
C	-13	-10	-7	-4	-2	0	2	4	2	4	2	0	-2	-4	-6	-8	-10	-12	-14
C	-15	-12	-9	-6	-4	-2	0	2	4	3	5	3	1	-1	-3	-5	-7	-9	-11
C	-17	-14	-11	-8	-6	-4	-2	0	2	5	4	5	3	1	-1	-3	-5	-7	-9
C	-19	-16	-13	-10	-8	-6	-4	-2	0	3	6	4	5	3	1	-1	-3	-5	-7
A	-21	-18	-15	-12	-10	-8	-6	-4	-2	1	4	6	4	6	4	2	0	-2	4
A	-23	-20	-17	-14	-12	-10	-8	-6	-4	-1	2	4	6	5	7	5	3	1	-1
T	-25	-22	-19	-16	-14	-12	-10	-8	-6	-3	0	2	4	6	5	8	6	4	2
G	-27	-24	-21	-18	-16	-14	-12	-10	-8	-5	-2	1	3	4	4	5	9	7	5
C	-29	-26	-23	-20	-18	-16	-14	-12	-10	-7	-4	-1	1	3	4	6	7	10	8

จากตารางที่ 2.1 ให้คะแนนเบสที่เหมือนกัน (match) เป็น +1 คะแนนเบสที่ต่างกัน (mismatch) เป็น 0 และถ้ามีการเติมช่องว่าง (gap) จะให้คะแนนเป็น –2 ในช่องที่ขีดเส้นใต้จากช่องด้านบนซ้ายถึงเซลที่มีคะแนนสูงที่สุดเป็นแนวขอการเรียงตัวที่ดีที่สุด ส่วนคะแนนที่เป็นตัวเอียงเป็นอีกแนวหนึ่งที่เป็นไปได้สำหรับการเรียงตัวที่ดีที่สุดจากตารางนี้แสดงผล alignment ที่ดีที่สุดคือ

ATGCCCTGACCGGAATGCC หรือ ATGCCCTGACCGGAATGCC หรือ ATGCCCTGACCGGAATGCC
ATGC—-TGACCCCAATGCC ATG-C-TGACCCCAATGCC ATG--CTGACCCCAATGCC

2.2.4 การเปรียบเทียบสายลำดับสองเส้นด้วยวิธี Smith และ Waterman [9]

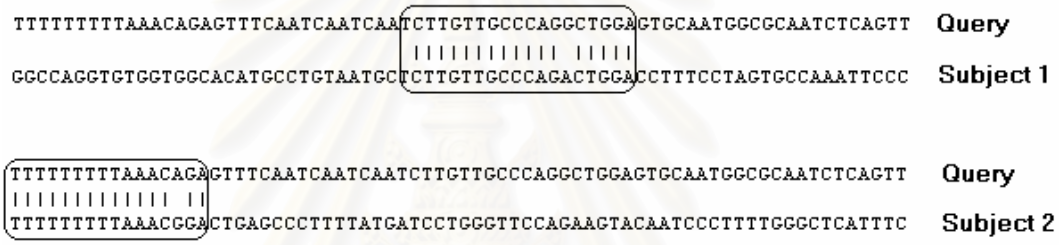
อัลกอริทึมการเขียนโปรแกรมแบบพลวัตซึ่งถูกคิดค้นโดย Needleman และ Wunsch จะใช้สำหรับการหาการเรียงตัวทั้งสาย (global alignment) ซึ่งเป็นการเปรียบเทียบลำดับทั้งเส้นสองสาย แต่เนื่องจากดีเอ็นเอและโปรตีนอาจมีส่วนที่เปรียบเทียบกันได้เพียงส่วนใดส่วนหนึ่งภายในโมเลกุล ดังนั้นในเวลาต่อมา Smith และ Waterman ได้คิดค้น อัลกอริทึมสำหรับการหาการเรียงตัวที่จำกัดในสาย (local alignment algorithm) ซึ่งเป็นวิธีการเปรียบเทียบลำดับเบสของดีเอ็นเอ 2 สายหรือลำดับกรดอะมิโนของโพลีเปปไทด์ 2 สาย โดยหาความคล้ายคลึงในส่วนใดส่วนหนึ่งในสายชีวโมเลกุลดังกล่าวจากการดัดแปลงอัลกอริทึมของ Needleman และ Wunsch โดยกำหนดให้คะแนนของช่วงที่ติดลบเป็นศูนย์และทำการให้คะแนนต่อไปเรื่อยๆจนกระทั่งคะแนนตกลงจนเท่ากันศูนย์ดังรูปที่ 2.2 แล้วจึงนำสายของการเรียงตัวที่มีความยาวมากกว่าค่าที่กำหนดมาเป็นผลลัพธ์ซึ่งในการเปรียบเทียบการเรียงตัวอาจให้ผลลัพธ์ของการเรียงตัวภายในมากกว่าหนึ่ง



รูปที่ 2.2 การเปรียบเทียบโดยใช้อัลกอริทึมของ Smith และ Waterman

2.2.5 Basic Local Alignment Search Tools [1] [11]

เนื่องจากการเปรียบเทียบโปรตีนจากฐานข้อมูลไม่สามารถถูกทำได้โดยการทำการเปรียบเทียบโปรตีนสองตัวว่าเหมือนกันตรง ๆ หรือไม่เนื่องจากโปรตีนที่มีลักษณะที่ใกล้เคียงกันบางรูปแบบอาจมีโครงสร้างที่ต่างกันออกไปแต่สามารถนำมาใช้แทนกันได้ และฐานข้อมูลของโปรตีนเหล่านี้ยังมี ลักษณะจำเพาะเนื่องจากโปรตีนที่มีหน้าที่บางอย่างอาจซ่อนอยู่ในโปรตีนอีกชนิดหนึ่งได้อย่างลงตัว ดังนั้นเราจึงไม่สามารถใช้ระบบจัดการฐานข้อมูลทั่วไปเพื่อที่จะใช้ในการจัดการกับฐานข้อมูลของโปรตีนได้ เพราะฉะนั้นระบบจัดการฐานข้อมูลของโปรตีนหรือรหัสทางพันธุกรรมจึงต้องมีการออกแบบเป็นพิเศษ โดยใช้หลักการเปรียบเทียบทางปัญญาประดิษฐ์เพื่อใช้ในการแยกแยะหน้าที่การทำงานของโปรตีนที่มีลักษณะเหมือนกันได้



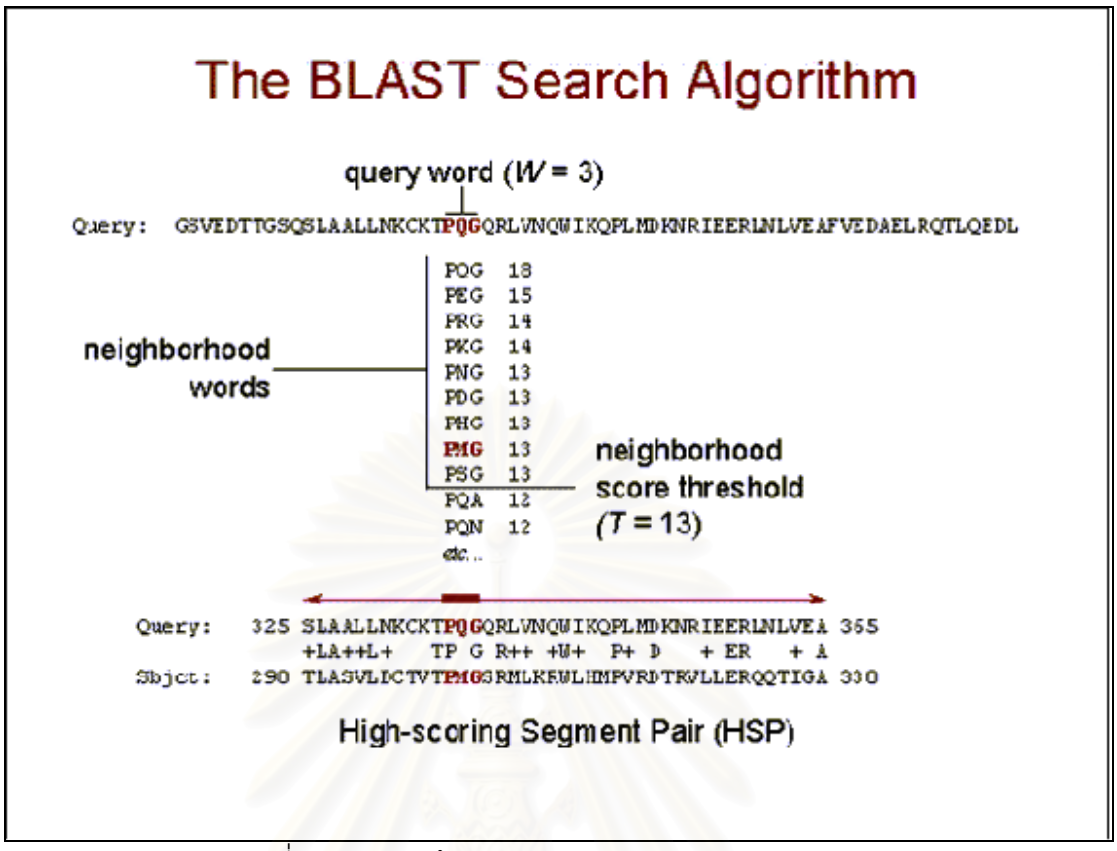
รูปที่ 2.3 ความสามารถในการจับคู่ของโปรแกรมบลาสท์

บลาสท์ เป็นโปรแกรมที่ใช้ในการวิเคราะห์ลำดับนิวคลีโอไทด์ของดีเอ็นเอ (DNA) หรือ อาร์เอ็นเอ (RNA) หรือ ลำดับของกรดอะมิโนของเปปไทด์ (peptide) หรือโปรตีนที่ได้จากการทดลองกับลำดับของนิวคลีโอไทด์หรือกรดอะมิโนที่มีอยู่ในฐานข้อมูลของบลาสท์จะทำการเปรียบเทียบทีละคู่ลำดับของนิวคลีโอไทด์ เพื่อหาบริเวณของลำดับที่มีการคล้ายกันมากที่สุดบลาสท์จะต้องทำงานเปรียบเทียบลำดับตั้งแต่ 1 ลำดับ จนกระทั่งเปรียบเทียบทั้งฐานข้อมูลเพื่อหาลำดับที่มีส่วนคล้ายกันทั้งหมดโดยผลลัพธ์ที่ได้จะเป็นส่วนเหมือนหรือส่วนคล้ายของลำดับย่อยๆ ของโปรตีนทั้งหมดดังรูปที่ 2.3 แสดงให้เห็นถึงส่วนที่มีความคล้ายกันจากการเปรียบเทียบ

ข้อมูลที่ได้จากการทำบลาสท์อาจช่วยให้เราทราบข้อมูลเกี่ยวกับหน้าที่ (Functional information) หรือข้อมูลเกี่ยวกับวิวัฒนาการ (Evolutionary information) ที่เกี่ยวข้องกับสายลำดับที่กำลังศึกษา

2.2.5.1 **อัลกอริทึมของบลาสท์ [11] [13]** อัลกอริทึมของบลาสท์คิดค้นโดย Altschul, Gish, Miller, Myers และ Lipman ในปีคศ.1990 เป็นทฤษฎีในการเปรียบเทียบความเหมือนหรือใกล้เคียงกันของโปรตีนที่เกี่ยวกับพันธุศาสตร์ โดยตัวโปรแกรมจะทำการเปรียบเทียบลำดับของโปรตีนกับทุก ๆ ข้อมูลที่มีอยู่ในฐานข้อมูลโดยการใช้ ฟังก์ชันวิทยาการศึกษาสำนึก (Heuristic Function) ช่วยในการเปรียบเทียบโดยคิดจากตารางแทนค่าจากค่าความน่าจะเป็นที่มีชื่อว่า สับซทิทยูชันแมทริกซ์ (Substitution Matrix) โดยการกำหนดคะแนน (Score) ให้กับการจัดเรียงตัวของโปรตีนแล้วทำการรวมคะแนนแล้วนำมาเปรียบเทียบคะแนนที่ได้ระหว่างข้อคำถามและระเบียนต่าง ๆ ในฐานข้อมูล (database records) ถ้าหากมีคะแนนเท่ากันหรือใกล้เคียงกันโดยไม่เกินกว่าค่าคงตัวที่กำหนดไว้ (กำหนดค่าตั้งต้นเท่ากับ 3) ก็จะทำการเลือกชิ้นส่วนของโปรตีนที่พบจากฐานข้อมูลนั้นเพื่อแสดงผล

ในการค้นหาของบลาสท์จะพยายามเปรียบเทียบลำดับในส่วนย่อย (Local) มากกว่าที่จะเป็นทั้งลำดับ (Global) โดยในการทำการเปรียบเทียบลำดับในส่วนย่อยนั้นจะใช้อัลกอริทึมที่ชื่อว่า สแตนดาร์ด สมิธ วอเตอร์-แมน (Standard Smith-Waterman) [9] ซึ่งเป็นกระบวนการค้นหาที่ทำงานช้ามาก แต่อัลกอริทึมของบลาสท์จะปรับปรุงอัลกอริทึมนี้เพื่อเพิ่มความเร็ว โดยมีขั้นตอนพื้นฐาน 3 ขั้นตอน คือ เริ่มจากขั้นตอนแรกบลาสท์จะสร้างลิสต์ (list) ของทุกลำดับที่ถูกแบ่งเป็นลำดับสั้น ๆ แล้ว ที่เรียกว่าเวิร์ดส์ (Words) ซึ่งจะใช้เก็บค่าคะแนนก่อนเริ่มทำการเปรียบเทียบ ในขั้นตอนต่อมาบลาสท์จะค้นหาการเกิดขึ้นซ้ำกันเวิร์ดส์ ซึ่งในกระบวนการค้นหานี้เนื่องจากเวิร์ดส์เป็นลำดับสั้น ๆ ดังนั้นจึงสามารถค้นหาได้เร็ว และสามารถค้นหาได้ละเอียดขึ้น ต่อมาหลังจากเปรียบเทียบกับทุก ๆ เบส (Base – A G T C) ของเวิร์ดส์แล้วจะต้องมีการให้คะแนน (Scored) ถ้าเบสตรงกัน และอยู่ในตำแหน่งที่ตรงกันด้วยจะให้คะแนนสูงสุด ในขณะที่เบสตรงกันแต่ตำแหน่งต่างกันก็จะให้คะแนนต่ำลงมา แต่ถ้าไม่เหมือนกันเลยจะให้คะแนนติดลบ ค่าผลรวมของคะแนนที่ของแต่ละเวิร์ดส์จะนำไปพิจารณาระดับความคล้ายกัน สำหรับเวิร์ดส์ที่มีคะแนนสูง ๆ จะเรียกว่าไฮสกอริ่งเซ็กเมนท์แพรส์ (High-scoring segment pairs) หรือ HSPs บลาสท์จะพยายามหา HSP ที่ดีที่สุด (มีคะแนนสูงที่สุด) โดยพยายามเพิ่มการเปรียบเทียบใน 2 ทิศทาง จนกระทั่งได้ ลำดับที่มีคะแนนสูงที่สุด ที่เรียกว่า แม็กซิมั่มสอกริ่งแซ็กเมนท์แพร์ (Maximal- scoring segment pair) หรือ MSP



รูปที่ 2.4 อัลกอริทึมของโปรแกรมบลาสท์ในการหา HSP

จากรูปที่ 2.4 แสดงให้เห็นถึงการเทียบความคล้ายคลึงโดยการเปรียบกับเวิร์ดส์ โดยในที่นี้ค่าคะแนนความคล้ายคลึงของ PQG และ PMG จะมีค่าเท่ากับ 13 ซึ่งเป็นค่าที่ยอมรับได้

2.3 โปรแกรมบลาสท์

2.3.1 NCBI BLAST [11] และ WU-BLAST [12]

เอ็นซีบีไอบลาสท์ (NCBI BLAST) และ วู-บลาสท์ (WU-BLAST) เป็นเครื่องมือ (tool) ที่นำอัลกอริทึมบลาสท์มาใช้ โดย เอ็นซีบีไอบลาสท์ ถูกพัฒนาขึ้นโดยพัฒนาโดยศูนย์สารสนเทศเทคโนโลยีชีวภาพแห่งชาติ ประเทศสหรัฐอเมริกา หรือ เอ็นซีบีไอ (NCBI) ขณะที่ วู-บลาสท์ เป็นอีกรุ่น (version) ของบลาสท์ที่พัฒนาขึ้นโดย Dr. Worren Gish และกลุ่มผู้พัฒนาจาก มหาวิทยาลัยวอชิงตัน (Washington University)

เอ็นซีบีไอบลาสท์มักเน้นไปในการพัฒนาเพื่อหาวิธีเปรียบเทียบเป็นกลุ่ม (Multiple-sequence profile) ตรงกันข้ามกับวู-บลาสท์ที่พัฒนาขึ้นเพื่อหาวิธีจัดการกับช่องว่างและการจัดการกับลำดับที่ซ้ำกัน โดยงานวิจัยนี้จะทำการวิจัยอยู่บนพื้นฐานของเอ็นซีบีไอบลาสท์ซึ่งเป็นมาตรฐานที่ใช้งานกันทั่วไป

โปรแกรมที่อยู่ในตระกูลบลาสท์ซึ่งใช้ อัลกอริทึมบลาสท์ในการเปรียบเทียบลำดับที่ต้องการรู้กับฐานข้อมูลโดยแบ่งออกเป็นอัลกอริทึมย่อยต่าง ๆ ซึ่งจะถูกเรียกใช้ผ่านแฟ้มกระทำการ (Executable File) ที่ชื่อว่า blastall โดยแต่ละอัลกอริทึมต่างๆ สามารถถูกเรียกใช้โดยเป็นพารามิเตอร์ (Parameter) ของโปรแกรมดังต่อไปนี้

1. blastn เป็นโปรแกรมสำหรับเปรียบเทียบลำดับเบสบนสายดีเอ็นเอกับลำดับเบสของดีเอ็นเอที่อยู่ในฐานข้อมูล
2. blastp เป็นโปรแกรมสำหรับเปรียบเทียบลำดับกรดอะมิโนของโพลีเปปไทด์กับลำดับกรดอะมิโนของโพลีเปปไทด์ที่อยู่ในฐานข้อมูล
3. blastx เป็นโปรแกรมสำหรับเปรียบเทียบลำดับกรดอะมิโนทั้ง 6 กรอบการอ่าน (6 reading-frame) ที่ได้จากการแปลรหัสลำดับเบสของสายดีเอ็นเอที่สนใจ (จากลำดับเบสของดีเอ็นเอทั้ง 2 สาย) โดยเปรียบเทียบกับลำดับกรดอะมิโนของโพลีเปปไทด์ที่อยู่ในฐานข้อมูล
4. tblastn เป็นโปรแกรมสำหรับเปรียบเทียบลำดับกรดอะมิโนของโพลีเปปไทด์กับลำดับกรดอะมิโน ที่ได้จากการแปลรหัสลำดับเบสของสายดีเอ็นเอที่อยู่ในฐานข้อมูลครบทั้ง 6 กรอบการอ่านโดยอ่านทั้งไปและในทางกลับสำหรับทุกความเป็นไปได้ของการพบโปรตีน
5. tblastx เป็นโปรแกรมสำหรับเปรียบเทียบลำดับกรดอะมิโน ที่ได้จากการแปลรหัสลำดับเบสของสายดีเอ็นเอที่สนใจ (จากลำดับเบสของดีเอ็นเอทั้ง 2 สาย) โดยเปรียบเทียบกับลำดับกรดอะมิโนที่ได้จากการแปลรหัสลำดับเบสของสายดีเอ็นเอที่อยู่ในฐานข้อมูล

โปรแกรมเอ็นซีบีไอบลาสท์รุ่นแรกจะไม่เติมช่องว่างในการเรียงตัวของลำดับโปรตีนแต่ในเอ็นซีบีไอบลาสท์เวอร์ชันที่ 2 หรือเรียกอีกอย่างหนึ่งว่า แก็พท์บลาสท์ (Gapped BLAST) ได้ถูกพัฒนาต่อมาให้มีการเติมช่องว่างได้ทำให้ผลที่ได้ง่ายต่อการเข้าใจมากขึ้นในแง่ความหมายทางชีววิทยา

การเปรียบเทียบจากโปรแกรมเอ็นซีบีไอบลาสท์สามารถทำได้ 4 วิธีคือ

1. เวิลด์ไวด์เว็บบลาสท์ (www BLAST) เป็นการให้บริการผ่านทางเว็บไซต์ของเอ็นซีบีไอโดยไปยัง http://www.ncbi.nlm.nih.gov/BLAST
2. สแตนด์อะโลนบลาสท์ (Standalone BLAST) สามารถใช้โปรแกรมบลาสท์บนเครื่องคอมพิวเตอร์ของเราเอง โดยเป็นการเปรียบเทียบกับฐานข้อมูลของตนเองที่มีอยู่ หรือฐานข้อมูลที่ดาวน์โหลดมาจากที่อื่น เช่นจาก NCBI โดยต้องเลือกโปรแกรมให้ตรงกับระบบปฏิบัติการที่ใช้อยู่
3. เครือข่ายบลาสท์ (Network BLAST) ซึ่งปัจจุบันเอ็นซีบีไอให้บริการ Blastcl3 อันเป็นผู้รับบริการเครือข่าย (BLAST network client) โดยผู้ใช้บริการสามารถเชื่อมต่อกับเครื่องบริการของเอ็นซีบีไอโดยใช้เกณฑ์วิธี ทีซีพี/ไอพี (TCP/PI protocol) เพื่อทำการเปรียบเทียบ
4. อีเมล์ (E-mail server) ใช้กรณีที่ผู้ใช้บริการไม่สะดวกที่จะเชื่อมต่ออินเทอร์เน็ตโดยผ่านเวิลด์ไวด์เว็บ ผู้ใช้สามารถทำได้โดยการส่งอีเมล์ตามรูปแบบ (format) ที่กำหนด โดยแนบไฟล์ซึ่งมีข้อมูลลำดับเบสหรือกรดอะมิโนไปยังอีเมล์แอดเดรส blast@ncbi.nlm.nih.gov หลังจากนั้นจะได้รับอีเมล์กลับมาพร้อมกับผลการเปรียบเทียบ

2.3.2 **รูปแบบของการส่งข้อคำถาม**

สามารถส่งลำดับของข้อคำถามเพื่อให้บลาสท์ดำเนินการวิเคราะห์ได้หลายรูปแบบ โดยอาจส่งในรูปแบบที่เรียกว่ารูปแบบฟาสตา (FASTA format) หรือ ยีนแบงก์แฟลตไฟล์ (GenBank flatfile) หรือใช้ ตัวระบุรูปแบบ (Identifiers) ก็ได้โดยมีรายละเอียดดังต่อไปนี้

1. **รูปแบบฟาสตา** เริ่มต้นโดยบรรทัดแรกจะมีเครื่องหมายมากกว่า (>) ตามด้วยคำบรรยายเป็นจำนวน 1 บรรทัด เรียกว่าบรรทัดนิยามฟาสตา (FASTA definition line) ส่วนบรรทัดต่อไปเป็นลำดับซึ่งมีความกว้างไม่เกิน 80 คอลัมน์ (คือรวมตัวอักษรและช่องว่าง) ดังตัวอย่างต่อไปนี้

```
>gi|532319|pir|TVFV2E|TVFV2E envelope protein
ELRLRYCAPAGFALLKCNDADYDGFKTNCSNVSVVHCTNLMNTTVTTGLLLNGSYSENRT
APTEVRRYTGGHERQKRVPFVXXXXXXXXXXXXXXXXXXXXXVQSQHLLAGILQQQKNL
LAAVEAQQQMLKLTIWGVK
```

2. **ลำดับเปลือย** (Bare sequence) สามารถส่งลำดับเปล่า ๆ โดยไม่ต้องมีบรรทัดนิยามฟาสตาเลยก็ได้ หรืออาจเป็นลำดับที่มีตัวเลขนำหน้าในแต่ละบรรทัด และมี

ช่องว่างคั่นทุก ๆ 10 ตำแหน่ง ซึ่งเรียกว่า ยีนแบงก์แฟลตไฟล์ อันเป็นส่วนหนึ่งของไฟล์ที่ได้จากยีนแบงค์ดังตัวอย่างต่อไปนี้

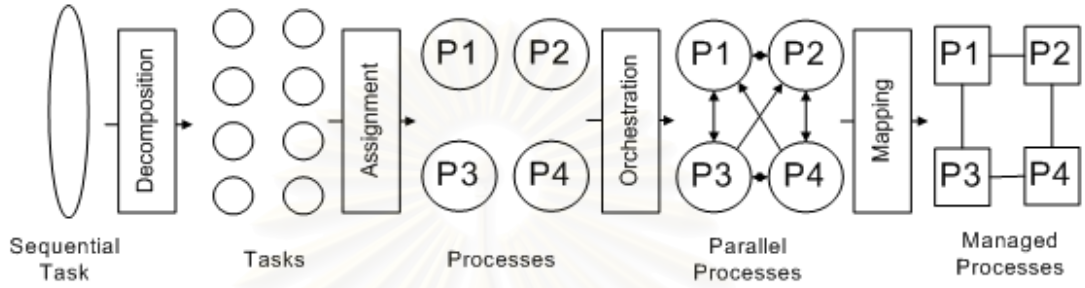
```
1 qikdllvsss tdldttlvlv naiyfkgmwk tafnaedtre mpfhvtkqes kpvqmmcmnn
61 sfnvatlpae kmkilelpfa sgdlsmlvll pdevsdleri ektinfeklt ewtnpntmek
121 rrvkvylpqm kieekynlts vlmalgmtdl fipsanltgi ssaeslkisq avhgafmels
181 edgiemagst gviedikhsp eseqfradhp flflikhnpt ntivyfgryw sp
```

3. **ตัวระบุรูปแบบ** ที่ใช้มี 3 แบบคือหมายเลขภาคี (Accession number), รุ่นภาคี (Accession version) หรือ ตัวระบุรูปแบบยีนแบงก์ (GenBank Identifier) เช่น p01013, AAA68881.1, 129295 นอกจากนี้ยังสามารถส่งตัวระบุรูปแบบลับดับของเอ็นซีบีไอ (NCBI sequence identifier) ซึ่งมีเส้นตรงแนวดิ่ง (|) คั่น (เช่น gi|129295) ก็ได้

4. **รูปแบบการบันทึกวากยสัมพันธ์ลำดับอย่างย่อ** (Abstract Syntax Notation Sequence – ASN.1) เป็นรูปแบบข้อมูลที่ถูกพัฒนาขึ้นโดยกลุ่มคนที่อยู่ในอุตสาหกรรมคอมพิวเตอร์ จากนั้น NCBI จึงนำมาพัฒนาต่อเพื่อใช้กับข้อมูลต่าง ๆ ด้านชีววิทยา เช่น ข้อมูลลำดับนิวคลีโอไทด์หรือกรดอะมิโน แผนที่ของยีน ข้อมูลเกี่ยวกับอนุกรมวิธาน โครงสร้างของโมเลกุล ตลอดจนข้อมูลเกี่ยวกับบรรณานุกรม ทำให้โปรแกรมต่าง ๆ รู้จักข้อมูลเหล่านี้ และสะดวกแก่การนำไปวิเคราะห์ต่อไป อย่างไรก็ตามรูปแบบนี้ไม่เหมาะกับการอ่านหรือแปลผลโดยคน เนื่องจากมีรายละเอียดมากและสลับซับซ้อน จึงมักใช้รูปแบบนี้เป็นข้อมูลนำเข้าสำหรับโปรแกรมต่าง ๆ มากกว่า

2.4 หลักการเขียนโปรแกรมแนวขนาน

แนวคิดเบื้องต้นของการเขียนโปรแกรมแนวขนานมาจากความต้องการเขียนโปรแกรมเพื่อทำงานงานหนึ่งโดยใช้เครื่องคอมพิวเตอร์หรือหน่วยประมวลผลหลาย ๆ หน่วยเราควรที่จะสามารถแบ่งปัญหาออกเป็นส่วน ย่อย ๆ เพื่อแบ่งการทำงานจากงาน ๆ เดียวในหน่วยประมวลผลใด ๆ ไปยังหน่วยประมวลผลอื่น ๆ ในระบบโดยมีขั้นตอนเป็นลำดับดังรูปที่ 2.5 โดยมีรายละเอียดดังนี้



รูปที่ 2.5 ขั้นตอนในการเขียนโปรแกรมแนวขนาน

1. **ดีคอมโปสิชัน** (Decomposition) - เป็นการจำแนกปัญหาออกเป็นชิ้น ๆ สามมารถทำได้หลายแบบแตกต่างกันไปตามแต่ละปัญหา ยกตัวอย่างเช่นในกรณีของ BLAST ฐานข้อมูลของระบบมีขนาดใหญ่มากและการเปรียบเทียบสามารถแบ่งออกไปทำงานเป็นส่วนๆ ได้ดังนั้นเราจึงทำการแบ่งฐานข้อมูลออกเป็นส่วนๆ เพื่อทำการเปรียบเทียบแล้วจึงนำคำตอบที่ได้ในแต่ละหน่วยประมวลผลมารวมกัน ในขณะเดียวกันจำนวนข้อคำถามต่อการเปรียบเทียบครั้งหนึ่งมีจำนวนมาก เราอาจสามารถแบ่งข้อคำถามออกเป็นส่วนๆ เพื่อแยกกันไปเปรียบเทียบในหน่วยประมวลผลต่างๆ กันได้แล้วจึงนำผลลัพธ์มารวมกันในภายหลัง ทั้งนี้เราควรมีการเลือกใช้วิธีการแยกปัญหาที่ดีเพื่อที่จะสามารถแบ่งงานไปยังหน่วยประมวลผลต่างๆ ได้อย่างเท่าเทียมและเพื่อลดขั้นตอนในการรอกันเองระหว่างหน่วยประมวลผลและเพื่อลดข้อมูลที่หน่วยประมวลผลต่างๆ ส่งหากัน
2. **แอสไซน์เมนต์** (Assignment) - การแบ่งงานไปยังหน่วยประมวลผลต่าง ๆ ควรจะสามารถแบ่งงานให้หน่วยประมวลผลต่าง ๆ ได้อย่างเท่ากัน
3. **ออเครสเทรชัน** (Orchestration) - เป็นการออกแบบข้อมูลที่หน่วยประมวลผลจะส่งหากันโดยจะทำการออกแบบให้ข้อมูลที่ส่งหากันมีขนาดเล็กที่สุดทั้งนี้ทั้งเนื่องจากระบบที่มีการแบ่งงานออกเป็นส่วนต่างๆ ดังนั้นการทำงานของระบบอาจมีการส่งงานต่อให้หน่วยประมวลผลอื่นทำหรือแลกเปลี่ยนข้อมูลกัน

4. **แม็พปิ้ง** (Mapping) – เป็นการเชื่อมโยงระบบเข้าด้วยกันโดยอาจคำนึงถึงสถาปัตยกรรมของระบบที่ทำการออกแบบ (System Architecture)

2.5 วิธีวัดประสิทธิภาพของระบบการประมวลผลแนวขนาน[]

การวัดประสิทธิภาพของระบบประมวลผลแนวขนานจะวัดจากอัตราการเร่งความเร็ว (Speedup) หรือความเร็วที่เพิ่มขึ้นของระบบเป็นอัตราส่วนของความเร็วที่วัดได้จากระบบการประมวลผลที่ใช้หน่วยประมวลผลเพียงหน่วยเดียวกับความเร็วที่วัดได้จากระบบประมวลผลแนวขนาน

$$SpeedUp\ (P) = Speed(P)/Speed(1)$$

หากหน่วยที่วัดเป็นเวลา อัตราการเร่งความเร็วจะเป็นอัตราส่วนของเวลาที่วัดได้จากระบบประมวลผลแนวขนาดกับเวลาที่วัดได้จากระบบการประมวลผลที่ใช้หน่วยประมวลผลเพียงหน่วยเดียว

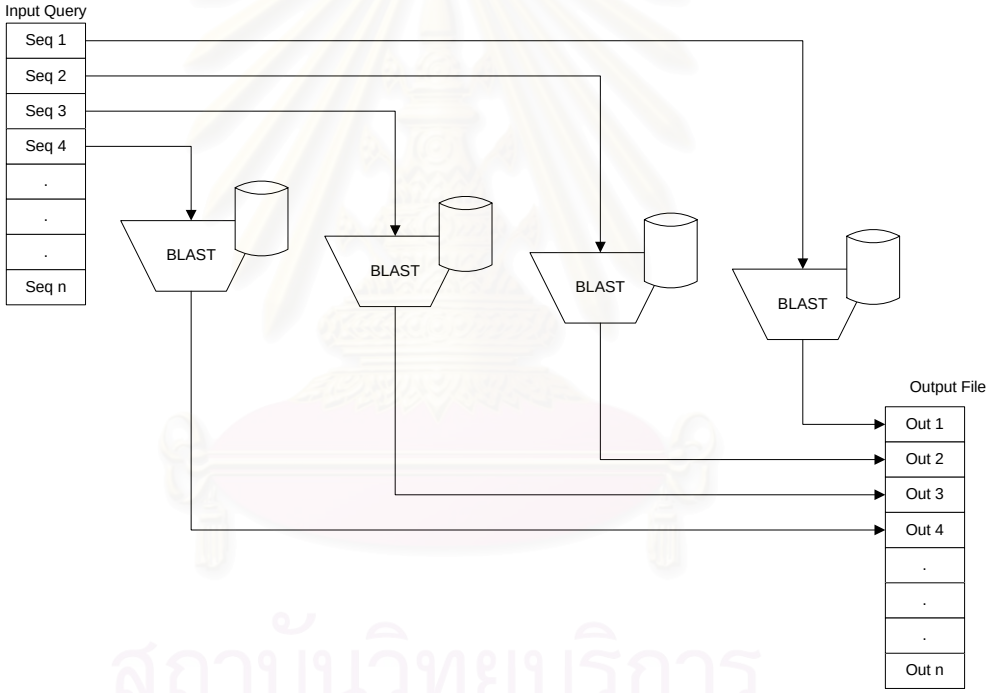
$$SpeedUp\ (P) = Time(1)/Time(P)$$

2.6 งานวิจัยที่เกี่ยวข้อง

มีความพยายามมากมายที่จะพัฒนาประสิทธิภาพการทำงานของโปรแกรมบลาสท์อยู่มากมายและงานวิจัยที่ประสพความสำเร็จในการเพิ่มประสิทธิภาพในการเปรียบเทียบข้อมูลของโปรแกรมบลาสท์มีดังต่อไปนี้

2.6.1 การทำข้อคำถามขนาน (Parallel Query) โดยใช้ลาสท์สคริปต์ [4] (BLAST Script)

จะแบ่งกลุ่มก้อนของข้อคำถามออกไปตามหน่วยประมวลผลต่างๆเพื่อทำการเปรียบเทียบ ซึ่งความเร็วของระบบโดยประมาณจะเท่ากับความเร็วของการเปรียบเทียบที่จำนวนข้อคำถามที่ต้องการเปรียบเทียบทั้งหมดหารด้วยจำนวนหน่วยประมวลผล เปรียบได้กับการเพิ่มจำนวนเครื่องเพื่อแบ่งงานออกเป็นหลาย ๆ ส่วนเพื่อทำการเปรียบเทียบ



รูปที่ 2.6 การทำข้อคำถามขนานที่มีใช้งานทั่วไป

จากรูปที่ 2.6 ข้อคำถามจะถูกตัดแบ่งออกไปตามหน่วยประมวลผลต่างๆ ของระบบ โดยแต่ละหน่วยประมวลผลจะมีฐานข้อมูลของโปรแกรมบลาสท์ที่เหมือนกันอยู่ในทุกๆ หน่วย หลังจากทำการประมวลผลแล้วจะนำผลลัพธ์ที่ได้กลับมาต่อที่แฟ้มข้อมูลสำหรับเก็บผลลัพธ์ แล้วหน่วยประมวลผลจะทำการขอข้อคำถามใหม่จนกระทั่งทำเสร็จสิ้นทุกข้อคำถาม

2.6.2 Efficiency of Shared-Memory Multiprocessors for a Genetic Sequence Similarity Search Algorithm [3]

โดย Ed Huai-hsin Chi , Elizabeth Shoop , John Carlis , Ernest Retzel , John Riedl งานวิจัยนี้ทำการพัฒนาระบบสภาวะแวดล้อมสำหรับโปรแกรมบลาสท์ โดยการนำโปรแกรมบลาสท์ไปทำงานบนเครื่อง SMPs (Shared-Memory Multiprocessors) และทำการวัดประสิทธิภาพการทำงานของโปรแกรม ผลการทดลองสามารถสรุปได้ว่าสามารถขจัดปัญหาคอขวดจากการสับเปลี่ยนข้อมูล ระหว่างฮาร์ดดิสก์และหน่วยความจำได้แต่มีข้อแม้ว่าหน่วยความจำรวมของระบบยังต้องมีขนาดใหญ่พอที่จะจุฐานข้อมูลทั้งหมดได้

2.6.3 High-Throughput BLAST [4] [16]

โดย Nick Camp, Haruna Cofer, and Roberto Gomperts ถูกพัฒนาโดยบริษัท Silicon Graphics Co.,Ltd. (SGI) งานวิจัยนี้ทำการปรับปรุงและพัฒนารหัสต้นฉบับ (source code) ของโปรแกรมบลาสท์ใหม่และให้ชื่อว่า HT-BLAST โดยโปรแกรมนี้ถูกปรับปรุงเพื่อทำการเปรียบเทียบข้อคำถามจำนวนมากกับฐานข้อมูลหลาย ๆ ฐานข้อมูลพร้อมกันได้อย่างรวดเร็วโดยไม่จำเป็นต้องรอให้ข้อคำถามใด ๆ ทำงานเสร็จไปก่อน

2.6.4 TurboBLAST [14]

โดยบริษัท TurboGenomics, Inc ทำการพัฒนาระบบการเปรียบเทียบแบบขนานบนโปรแกรมบลาสท์ในเชิงพาณิชย์ งานวิจัยนี้ถูกพัฒนาโดยการตัดแบ่งข้อคำถามและฐานข้อมูลออกเป็นส่วน ๆ รวมถึงความสามารถในการถ่ายเทภาระของระบบ (load balancing) โดยการทำการเปรียบเทียบข้อคำถามจำนวนมากเมื่อเครื่องใดเครื่องหนึ่งในระบบมีทรัพยากรมากพอที่จะทำการเปรียบเทียบได้ กระบวนการของผู้จัดการระบบ (system manager process) จะทำการกระจายข้อคำถามไปยังเครื่องนั้น ๆ และเนื่องจากงานวิจัยนี้ถูกพัฒนาโดยใช้ภาษาจาวา (JAVA) จึงสามารถที่จะทำการประมวลผลแบบขนานข้ามไปมาได้ระหว่างฐานงาน (Platform) งานวิจัยนี้สามารถแสดงความสามารถของระบบได้ถึงระดับซูเปอร์ลิเนียร์สปีดอัพ (superlinear speedup) ซึ่งเป็นความสามารถของระบบประมวลผลแบบขนานที่สามารถข้ามขีดจำกัดของกฎของแอมดาห์ล (Amdahl's Law)

2.6.5 BlastMachine [15]

โดยบริษัท Paracel เป็นโปรแกรมเชิงพาณิชย์ ทำการพัฒนาระบบเปรียบเทียบแบบขนานบนโปรแกรมบลาสท์ที่ถูกยกเครื่องใหม่ (reengineer) และมีการออกแบบฮาร์ดแวร์ (hardware) เพื่อใช้ในการเปรียบเทียบฐานข้อมูลโปรตีนโดยเฉพาะโดย ระบบของ BlastMachine อยู่บนพื้นฐานของระบบการประมวลผลแบบขนานที่เรียกว่าสถาปัตยกรรมแบบคลัสเตอร์ (clustered architecture) โดยใช้เพนเทียมโปรเซสเซอร์ของอินเทล (Intel Pentium processors) และระบบปฏิบัติการลินุกซ์ (Linux operating system) และใช้โปรแกรมที่ทาง Paracel ได้ทำการพัฒนาใหม่โดยมีพื้นฐานจากโปรแกรมบลาสท์เดิมของเอ็นซีบีไอ

2.6.6 BioScan [7]

โดย Singh R., ทำการพัฒนาฮาร์ดแวร์สำหรับการเปรียบเทียบข้อมูลพันธุศาสตร์โดยเฉพาะด้วยวงจรรวมความจุสูงมาก (VLSI - Very Large Scale Integration) จากอัลกอริทึมของบลาสท์โดยชิพของ BioScan ประกอบด้วยซีมอส (CMOS - complementary metal-oxide semiconductor) ขนาด 1.2 ไมโครเมตร, และวิ่งที่ความเร็ว 32 เมกะเฮิร์ทซ์ โดยในระบบจะถูกโหลดเวิร์ดส์โดยคิดคำนวณจากความยาวของลำดับที่ต้องการเปรียบเทียบก่อนที่จะทำการเปรียบเทียบลำดับของโปรตีนระบบที่ทำสำเร็จในรุ่นแรกไม่ได้มีความเร็วมากกว่าระบบปัจจุบัน และทางผู้วิจัยกำลังพัฒนาฮาร์ดแวร์ในรุ่นที่เร็วขึ้นและมีความสามารถในการทำการเชื่อมโยงเป็นระบบการประมวลผลแนวขนาน

2.6.7 Improving the Performance of BLAST in a Memory Limited Environment [10]

โดย Warin Wattanapornprom, Natawut Nupairoj และ Prabhas Chongstitvatana งานวิจัยนี้ทำการประปรุงประสิทธิภาพของโปรแกรมบลาสท์ในการเปรียบเทียบข้อคำถามจำนวนมากๆ โดยใช้หน่วยประมวลผลเพียงหน่วยเดียว ทั้งนี้งานวิจัยนี้ได้เสนอวิธีปรับปรุงประสิทธิภาพของโปรแกรมบลาสท์โดยการตัดแบ่งฐานข้อมูลออกเป็นส่วนย่อยๆ เพื่อทำการเปรียบเทียบเนื่องจากงานวิจัยนี้พบว่าโปรแกรมบลาสท์จะมีประสิทธิภาพสูงสุด เมื่อฐานข้อมูลของระบบสามารถจัดเก็บไว้ในหน่วยความจำได้ทั้งหมดขณะทำการเปรียบเทียบ โดยผลของงานวิจัยนี้สามารถเพิ่มประสิทธิภาพของการเปรียบเทียบได้เท่าตัว ทั้งนี้ขึ้นอยู่กับขนาดของฐานข้อมูลและสภาพแวดล้อมของระบบ

บทที่ 3

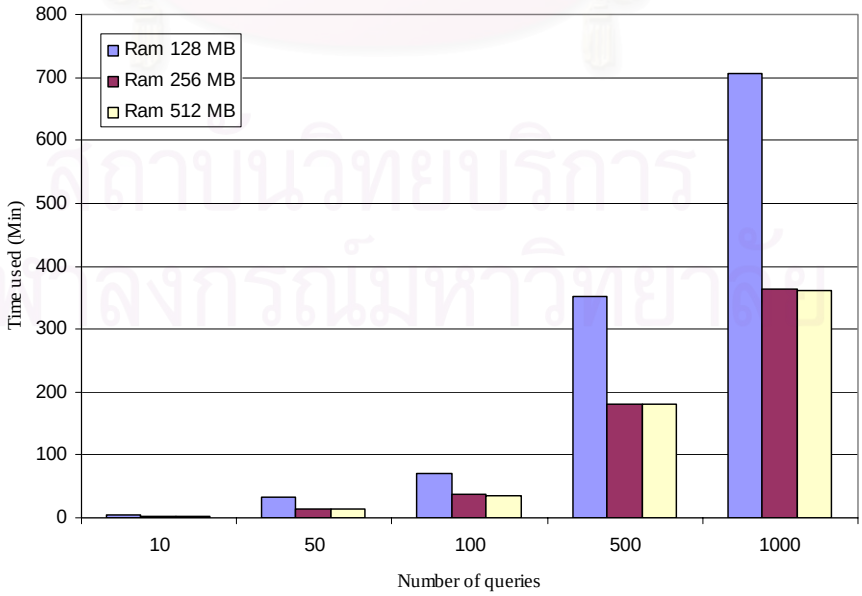
การวิเคราะห์และปรับปรุงประสิทธิภาพเบื้องต้นของโปรแกรมบลาสท์

3.1 การวิเคราะห์ประสิทธิภาพของโปรแกรมบลาสท์

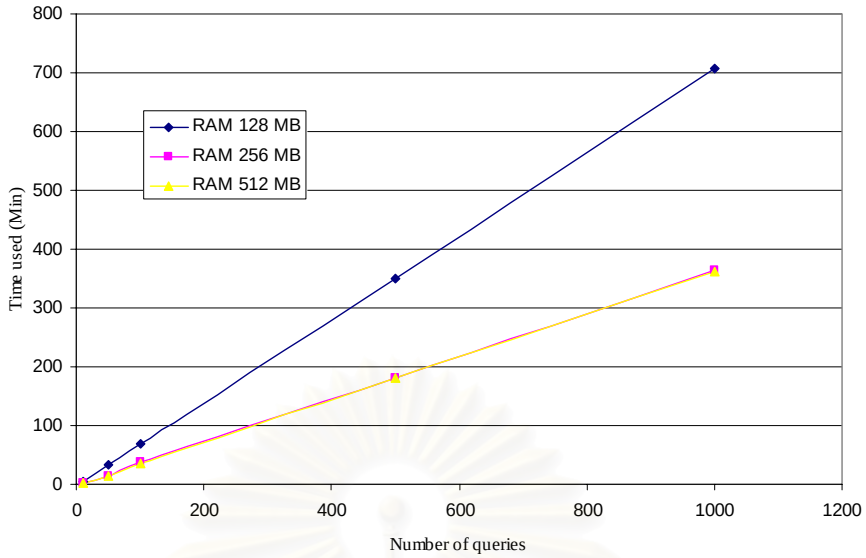
จากการทดลองทำการวัดประสิทธิภาพของโปรแกรมบลาสท์ที่หน่วยความจำหลักขนาดต่างกัน เราพบว่าบลาสท์เป็นโปรแกรมที่มีความต้องการใช้ หน่วยความจำของระบบสูงและโปรแกรมบลาสท์จะมีประสิทธิภาพสูงสุดก็ต่อเมื่อหน่วยความจำหลักสามารถบรรจุฐานข้อมูลขณะทำการสืบค้นข้อมูลได้ทั้งหมด จากผลการทดลองโดยการจับเวลาที่ได้จากการสืบค้นข้อคำถามจำนวนเท่ากันจากฐานข้อมูลเดียวกันที่สภาพแวดล้อมการทำงานเดียวกันโดยเปลี่ยนหน่วยความจำหลักของระบบต่าง ๆ ดังนี้

ตารางที่ 3.1 การวัดประสิทธิภาพของโปรแกรมบลาสท์
ในการสืบค้นข้อมูลที่หน่วยความจำขนาดต่างๆกัน

หน่วยความจำที่ใช้ (เมกะไบต์)	จำนวนข้อคำถามที่ใช้				
	10	50	100	500	1000
	เวลาที่ใช้ในการประมวลผล (นาที)				
128	5	33	70	351	706
256	2	15	37	180	364
512	2	14	37	180	363



รูปที่ 3.1 ประสิทธิภาพของโปรแกรมบลาสท์ในการสืบค้นข้อมูลที่หน่วยความจำขนาดต่างๆกัน



รูปที่ 3.2 ประสิทธิภาพของโปรแกรมบลาสท์ในการสืบค้นข้อมูลที่หน่วยความจำขนาดต่างๆกัน

จากการทดลองจะเห็นได้ว่าความเร็วของโปรแกรมบลาสท์ขึ้นอยู่กับขนาดของหน่วยความจำหลักของระบบ หากระบบมีหน่วยความจำหลักมากก็จะสามารถทำงานได้เร็วขึ้นเนื่องจากไม่ต้องเสียเวลาในการสับเปลี่ยนข้อมูลระหว่างหน่วยความจำหลักและจานบันทึกข้อมูลหรือฮาร์ดดิสก์ซึ่งเป็นหน่วยเก็บข้อมูลทุติยภูมิ (secondary storage) ซึ่งมีอัตราการแลกเปลี่ยนข้อมูลช้ากว่าหน่วยความจำหลักอย่างมาก ดังนั้นหากขนาดของหน่วยความ จำหลักมีขนาดใหญ่เพียงพอที่จะสามารถจัดเก็บฐานข้อมูลพันธุศาสตร์ของโปรแกรมบลาสท์ได้ทั้งหมดในขณะที่ทำการเปรียบเทียบลำดับจะทำให้โปรแกรมบลาสท์สามารถทำงานได้ประสิทธิภาพสูงสุดเท่าที่หน่วยประมวลผลจะทำได้ จะเห็นได้จากการเพิ่มขนาดของหน่วยความจำจาก 256 เมกะไบต์ซึ่งใหญ่เพียงพอที่จะใส่ฐานข้อมูลทั้งหมดขึ้นเป็น 512 เมกะไบต์แทบจะไม่ได้ช่วยในการสืบค้นของระบบได้เลย ปัญหานี้เรียกว่า ปัญหาหน่วยความจำขนาดจำกัด (memory-bounded problem)

3.2 การปรับปรุงประสิทธิภาพของโปรแกรมบลาสท์ในสภาวะหน่วยความจำจำกัด [10]

เนื่องจากหน่วยความจำที่จะสามารถเก็บฐานข้อมูลพันธุศาสตร์ในขณะประมวลผลนั้นมีความจำเป็นที่จะต้องมีขนาดใหญ่มาก ทั้งนี้คอมพิวเตอร์และระบบปฏิบัติการที่สามารถรองรับหน่วยความจำมหาศาลนั้นหายากและมีราคาแพง ดังนั้นเราจึงเสนอวิธีการที่จะสามารถทำให้หน่วยความจำหลักสามารถที่จะจุขนาดของฐานข้อมูลในทั้งหมดในขณะทำการสืบค้นข้อมูลและเราพบว่ารูปแบบของฐานข้อมูลทางพันธุศาสตร์สามารถที่จะตัดต่อได้โดยแต่ละระเบียนของฐานข้อมูลทางพันธุศาสตร์จะเริ่มต้นด้วยเครื่องหมาย “>” ดังรูปที่ 3.3

```
>gi|6|emb|CAA42669.1| (X60065) beta-2-glycoprotein  I [Bos taurus]
PALVLLLGFLCHVAIAGRTCPKPDELPFSTVVPLKRTYEPGEQIVFSCQPGYVSRGGIRRFTCPLTGLWPINTLKCMPRV
CPFAGILENGTVRYTTFEYPNTISFSCHTGFYLKGASSAKCTEEGKWSPDLPVCAPITCPPPPIPKFASLSVYKPLAGNN
SFYGSKAVFKCLPHHAMFGNDTVTCTEHGNWTQLPECREVRCPFPSRPDNGFVNHPANPVLYYKDTATFGCHETYSLDGP
EEVECSKFGNWSAQPSCKASCKLSIKRATVIYEGERVAIQNKFKNGMLHGQKVSFFCKHKEKKCSYTEDAQCIDGTIEIP
KCFKEHSSLAFWKTDASDVKPC

>gi|129249|sp|P02820|OSTC_BOVIN OSTEOCALCIN PRECURSOR (GAMMA-CARBOXYGLUTAMIC ACID-
CONTAINING PROTEIN) (BONE GLA-PROTEIN) (BGP)_gi|538590|pir|GEBO osteocalcin precursor -
bovine_gi|8|emb|CAA35997.1| (X51700) bone Gla precursor (100 AA) [Bos taurus]_gi|720|emb|CAA37737.1| (X53699)
Gla protein precusor [Bos taurus]
MRTPMLLALLALATLCLAGRADAKPGDAESGKGAAFVSKQEGSEVVKRLRRYLDHWLGAPAPYPDPLEPKREVCELNPDC
DELADHIGFQEAYRRFYGPV
```

รูปที่ 3.3 แสดงโครงสร้างของ ฐานข้อมูลพันธุศาสตร์ของโปรแกรมบลาสท์

จากสมมุติฐานที่ว่าโปรแกรมบลาสท์จะทำงานได้ดีที่สุดเมื่อฐานข้อมูลทั้งหมดสามารถที่จะถูกจัดเก็บได้ในหน่วยความจำ จึงได้ทำการทดลองเพื่อพิสูจน์ว่าหากทำการตัดแบ่งฐานข้อมูลออกเป็นสองส่วนเพื่อทำการแยกสืบค้นข้อมูลสองครั้งจะสามารถเพิ่มความเร็วจากการทำงานของโปรแกรมได้ เนื่องจากระบบไม่จำเป็นต้องทำการสับเปลี่ยนข้อมูลระหว่างหน่วยความจำหลักและฮาร์ดดิสก์ การทดลองทำการตัดแบ่งฐานข้อมูลจากจุดเริ่มต้นของระเบียน แต่มีเงื่อนไขว่าขนาดของฐานข้อมูลที่ทำการตัดมีขนาดเล็กกว่าหน่วยความจำหลักแล้วทำการวัดประสิทธิภาพของโปรแกรมบลาสท์ด้วยการจับเวลาที่ได้จากการสืบค้นข้อคำถามเดียวกันจากฐานข้อมูลที่ถูกตัดแบ่งที่ขนาดต่างๆกันที่สภาพแวดล้อมการทำงานเดียวกัน จากนั้นจึงตรวจสอบผลลัพธ์ที่ได้จากการประมวลผลของโปรแกรมว่าถูกต้อง

เราสามารถหาเวลารวมได้จากสูตร

$$t_{total} = \sum_{i=1}^{n} t_i + t_{post}$$

โดย

t_{total} = เวลาที่ใช้ทั้งหมด

t_{post} = เวลาที่ใช้ในการเชื่อมต่อผลลัพธ์

t_i = เวลาที่ใช้ในการสืบค้นบนฐานข้อมูลส่วนที่ i

n = จำนวนฐานข้อมูลที่ถูกตัด

ตารางที่ 3.2 ประสิทธิภาพของโปรแกรมบลาสท์ที่ขนาดของฐานข้อมูลต่างๆกัน

ฐานข้อมูลชุดที่	จำนวนที่ตัดแบ่ง (ส่วน)	ขนาด (เมกะไบต์)	เวลาที่ใช้ (นาที)	เวลาที่ใช้ในการต่อผลลัพธ์ (นาที)	เวลารวม (นาที)
1	1	234	70	0	70
2	1	109	17	1	38
	2	125	20		
3	1	100	16	1.5	38.5
	2	100	15		
	3	34	6		

ผลลัพธ์ของโปรแกรมบลาสท์ตามปกติแล้วสามารถที่จะแบ่งออกได้เป็นสี่ส่วนคือ

1. ส่วนหัวเรื่องของผลลัพธ์จะแสดงวันที่และเวลาที่บันทึกผลและมีส่วนอ้างอิงเพื่อให้เกียรติแก่ผู้คิดค้นและพัฒนาโปรแกรมตลอดจนถึงรายละเอียดของฐานข้อมูลที่นำมาใช้เปรียบเทียบ
2. ผลลัพธ์โดยสรุปที่ได้จากการเปรียบเทียบซึ่งจะถูกนำมาเรียงกันตามคะแนนที่ได้จากการเปรียบเทียบ
3. ผลลัพธ์ที่ได้จากการเปรียบเทียบโดยละเอียด ทั้งนี้จะแสดงส่วนของลำดับที่ได้คะแนนจากการเปรียบเทียบ และรายละเอียดของช่วงที่ได้คะแนน ตลอดจนถึงช่องว่างที่ถูกสร้างขึ้นเพื่อให้การเปรียบเทียบสมบูรณ์
4. ส่วนสรุปผลที่ได้จากการเปรียบเทียบนำมาเก็บไว้เป็นสถิติต่างๆ

จากผลลัพธ์ที่ได้จากการประมวลผลสามารถแบ่งออกเป็นกรณีต่างๆได้ดังนี้

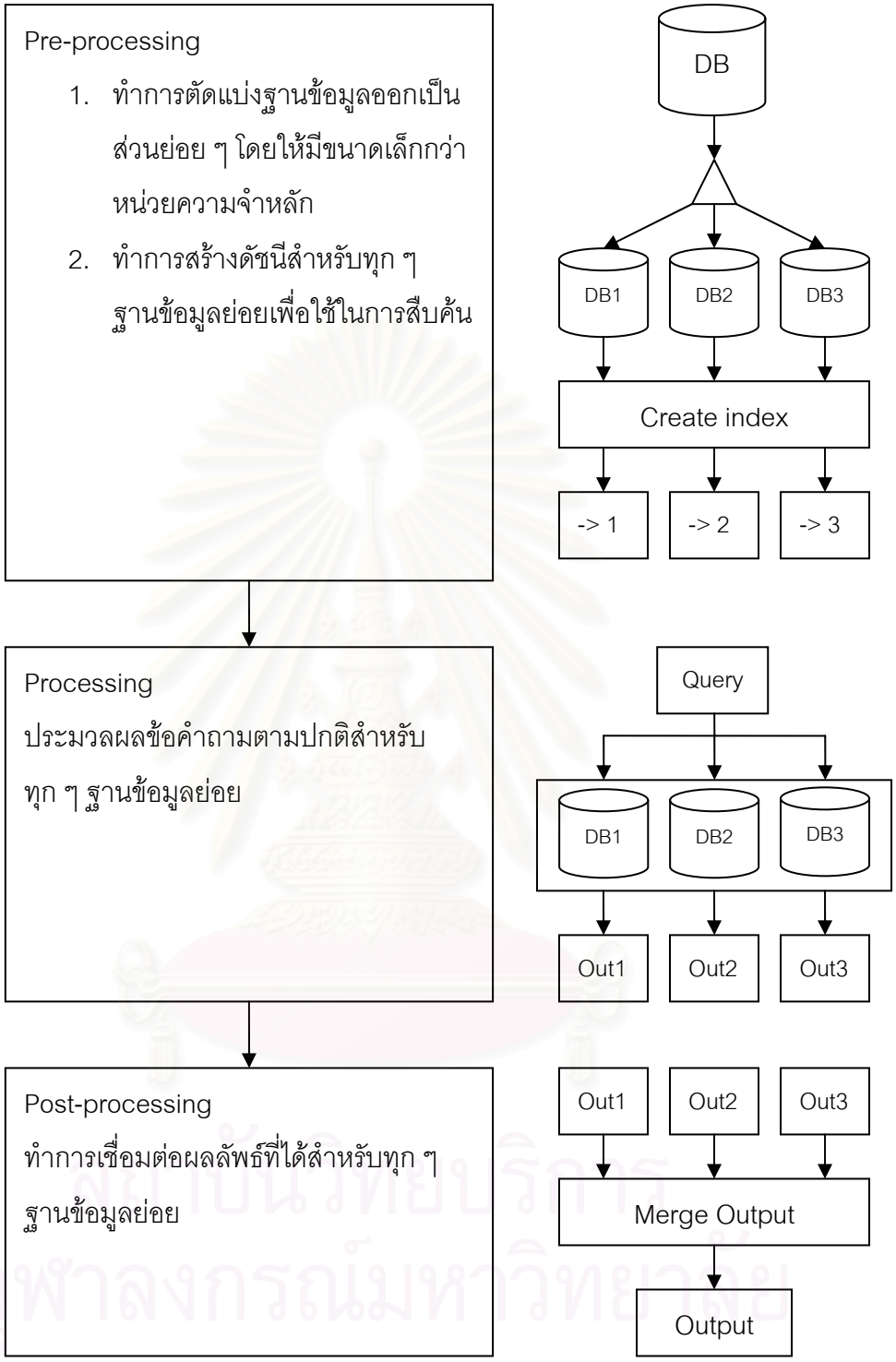
1. กรณีที่พบคำตอบจากการสืบค้นจากฐานข้อมูลทั้งหมดคำตอบที่ได้สามารถนำมาต่อกันได้ทันที
2. กรณีที่ไม่พบคำตอบเลยคำตอบที่ได้จากการสืบค้นให้เลือกเอาตำตอบใดคำตอบหนึ่งจากผลลัพธ์ทั้งหมด (คำตอบเหมือนกัน)
3. กรณีที่พบคำตอบจากบางส่วนสามารถนำคำตอบที่ได้มาต่อกันตามลำดับ

ทั้งนี้การเชื่อมต่อผลลัพธ์ในส่วนของหัวเรื่องและส่วนสรุปผลจำเป็นจะต้องนำผลที่ได้มารวมกันเพื่อให้ผลจากการเปรียบเทียบเกิดจากการค้นหาจากฐานข้อมูลเสมือนจากทุกๆ ฐานข้อมูลย่อย โดยผลลัพธ์ที่ได้จากจากทั้งสามกรณีที่ได้กล่าวมาแล้วคำตอบที่ได้จากการประมวลผลหลังทำการตัดต่อฐานข้อมูลตรงกันกับข้อมูลที่ได้จากการประมวลผลโดยไม่ตัดแบ่งฐานข้อมูล

ผลลัพธ์ที่ได้จากการฐานข้อมูลย่อยในบางกรณีหากมีมากเกินไปโปรแกรมบลาสท์จะทำการเลือกเอาผลลัพธ์เพียงบางส่วนที่มีคะแนนความน่าเชื่อถือสูงออกมาแสดง โดยจะทำให้ผลลัพธ์ที่ได้จากการแยกเปรียบเทียบในแต่ละฐานข้อมูลย่อยมีจำนวนมากกว่าผลลัพธ์ที่ได้จากการเปรียบเทียบกับฐานข้อมูลก่อนทำการแบ่ง ทั้งนี้เนื่องจากตัวกรองผลลัพธ์ไม่ได้ถูกนำมาใช้งานกับการเปรียบเทียบโดยรวม

ขั้นตอนการปรับปรุงประสิทธิภาพของโปรแกรมบลาสท์สามารถสรุปได้โดยแบ่งออกเป็นสามขั้นตอนหลักๆ ในรูปที่ 3.4 ดังนี้

1. **การประมวลผลเบื้องต้น** - เริ่มจากการทำการตัดแบ่งฐานข้อมูลออกเป็นส่วนย่อย ๆ โดยให้มีขนาดเล็กกว่าหน่วยความจำหลัก หลังจากนั้นทำการสร้างดัชนีสำหรับทุกๆ ฐานข้อมูลย่อยเพื่อใช้ในการสืบค้น
2. **การทำการประมวลผลตามปกติ** - ประมวลผลข้อคำถามตามปกติสำหรับทุกๆ ฐานข้อมูลย่อย
3. **การประมวลผลช่วงปลาย** - ทำการเชื่อมต่อผลลัพธ์ที่ได้สำหรับทุก ๆ ฐานข้อมูลย่อย



รูปที่ 3.4 แสดงขั้นตอนการปรับปรุงประสิทธิภาพของโปรแกรมบลาสท์ในสภาวะหน่วยความจำจำกัด

จากการทดลองจะเห็นได้ว่าการตัดต่อฐานข้อมูลให้เล็กลงก่อนทำการแยกประมวลผลสามารถเพิ่มประสิทธิภาพในการประมวลผลได้ดี โดยเฉพาะอย่างยิ่งจะสามารถสังเกตได้จากการที่เวลาที่ใช้ในการประมวลผลหลังการตัดต่อจะเท่ากันกับเวลาที่ใช้ในการประมวลผลจากฐานข้อมูลที่ไม่ได้ทำการตัดต่อที่หน่วยความจำใหญ่เพียงพอที่จะจัดเก็บฐานข้อมูลในขณะประมวลผลได้ทั้งหมด ทั้งนี้ยังมิได้รวมเวลาที่ใช้ในการต่อผลลัพธ์ซึ่งน้อยกว่าหนึ่งนาที

ในการตัดแบ่งฐานข้อมูลออกเป็นส่วยย่อย ๆ เพื่อประมวลผลจากการทดลองโดยใช้ข้อคำถามจำนวน 500 ข้อคำถามโดยใช้เวลา 180 นาที และใช้เวลาในการต่อผลลัพธ์ประมาณ 5นาที เราจะสามารถเพิ่มอัตราการเร่งความเร็วได้มากกว่าระบบที่ยังไม่ได้รับการปรับปรุง

โดยในที่นี้อัตราการเร่งความเร็วจะเท่ากับอัตราส่วนของเวลาที่วัดได้จากระบบที่ปรับปรุงแล้วกับเวลาที่วัดได้จากระบบเดิมดังสมการ

$$\text{SpeedUp} = \frac{\text{Time(Original)}}{\text{Time(Improve)}}$$

$$= 351/(180+5)$$

$$= 1.9 \text{ เท่า}$$

ในขณะที่ความเร็วของระบบที่ใช้วิธีการแบบ Parallel Query โดยทำการแบ่งข้อคำถามจำนวนเท่า ๆ กันไปประมวลผลตามหน่วยประมวลผลต่าง ๆ ของระบบจะใช้เวลาดังสมการ

$$\text{TimeUsed} = \text{NodeMen/Node} + \text{NetworkOverhead}$$

โดย

NodeMem = ความเร็วสูงสุดของระบบที่บลาสท์สามารถทำได้ที่หน่วยความจำของหน่วยประมวลผลของระบบโดยอยู่บนสมมุติฐานที่แต่ละหน่วยมีหน่วยความจำเท่ากัน

NetworkOverhead = เวลาที่เกิดจากค่าใช้จ่ายรายหัว (overhead) ของระบบเครือข่าย เกิดจากการส่งข้อมูลของข้อคำถามระหว่างหน่วยประมวลผล

หากทำการเปรียบเทียบประสิทธิภาพของระบบสภาพแวดล้อมการประมวลผลแนวขนานทั้งสองแบบโดยใช้สูตร

$$\text{SpeedUp} = \frac{\text{Time(1 node)}}{\text{Time(P}}$$

จะเห็นได้ว่าอัตราการเร่งความเร็วของ Parallel Query จากการทดลอง ข้างต้นที่จำนวน 2 node โดยใช้ข้อคำถามจำนวน 500ข้อคำถามโดยแต่ละ node ใช้หน่วยความจำขนาด 128 เมกะไบต์ โดยอยู่บนสมมุติฐานว่าค่าของ NetworkOverhead มีค่าน้อยมาก และสามารถทำการประมวลผลเชิงขนานได้ 100 เปอร์เซ็นต์

= 351/(351/2)

= 2 เท่า

บทที่ 4

ระบบการประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์

4.1 การออกแบบระบบการประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์

4.1.1 แนวคิดในการพัฒนา

จากการวิเคราะห์ประสิทธิภาพของโปรแกรมบลาสท์เบื้องต้นทำให้ทราบว่า โปรแกรมบลาสท์จะทำงานได้ประสิทธิภาพสูงสุดเมื่อหน่วยความจำหลักสามารถเก็บฐานข้อมูลได้ทั้งหมดขณะทำการเปรียบเทียบลำดับได้ ระบบประมวลผลเชิงขนานจึงควรจะสามารถขจัดปัญหาหน่วยความจำขนาดจำกัดได้โดยการตัดแบ่งฐานข้อมูลออกเป็นส่วนย่อยๆ ดังที่ได้เสนอไว้แล้วในการปรับปรุงประสิทธิภาพของโปรแกรมในสภาวะหน่วยความจำขนาดจำกัด โดยฐานข้อมูลของโปรแกรมจะถูกกระจายไปยังหน่วยประมวลผลต่างๆ หลังจากนั้นแต่ละหน่วยประมวลผลจะทำการเปรียบเทียบข้อคำถามในแต่ละฐานข้อมูลที่ได้รับมอบหมาย เมื่อได้ทำการเปรียบเทียบเสร็จแล้วจึงทำการรวมผลลัพธ์ที่ได้จากฐานข้อมูลต่างๆ เป็นผลลัพธ์ที่ได้จากฐานข้อมูลเดียวกัน

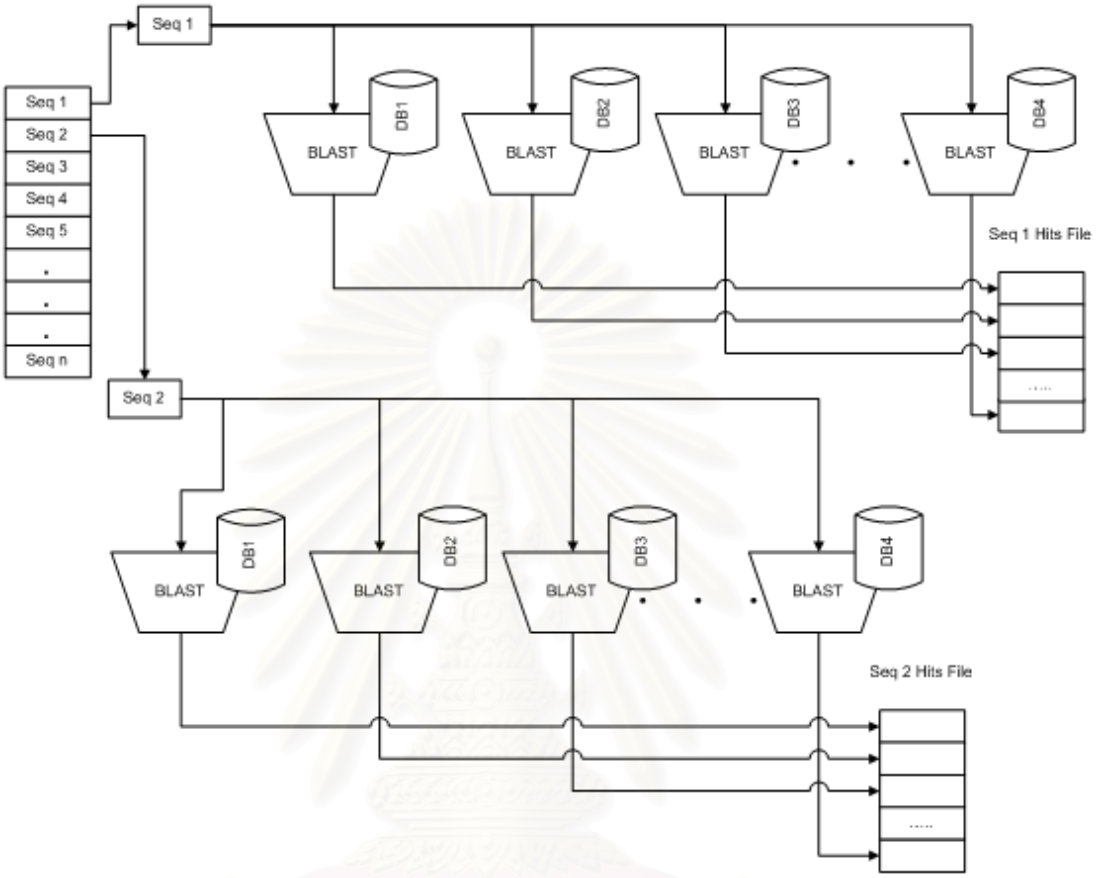
ทั้งนี้ฐานข้อมูลไม่ควรจะถูกแบ่งออกเป็นส่วนย่อยๆ ตามจำนวนของหน่วยประมวลผลที่มีในระบบ เนื่องจากในการทดลองตัดแบ่งฐานข้อมูลออกเป็นส่วนย่อยที่เล็กกว่าขนาดของหน่วยความจำหลักแล้ว ไม่สามารถช่วยให้เวลารวมของการเปรียบเทียบสูงขึ้นแต่อย่างใด ดังนั้นจึงควรตัดแบ่งฐานข้อมูลให้มีขนาดเล็กกว่าขนาดของหน่วยความจำหลักเพียงเล็กน้อย เพื่อไม่ให้เกิดค่าใช้จ่ายรายหัวที่เพิ่มขึ้นจากการรวมผลลัพธ์

เหตุผลหลายประการทำให้เวลาที่ได้จากการเปรียบเทียบชุดของข้อคำถามกับฐานข้อมูลต่าง ๆ มีขนาดไม่เท่ากันมีดังนี้

1. ขนาดของฐานข้อมูลไม่สามารถถูกแบ่งให้มีขนาดเท่ากัน
2. จำนวนและขนาดของระเบียนในฐานข้อมูลไม่เท่ากัน
3. ผลลัพธ์ที่ได้จากการเปรียบเทียบมีขนาดไม่เท่ากัน

จากสาเหตุดังกล่าวหากทำการเปรียบเทียบข้อคำถามชุดเดียวกันในแต่ละหน่วยประมวลผลของระบบจะทำให้เกิดภาวะของการรอผลลัพธ์ ผลลัพธ์ที่ได้จากหน่วยประมวลผลที่ใช้เวลาน้อยกว่าจะยังไม่สามารถนำมารวมกันได้ จนกว่าผลลัพธ์ที่ได้จากหน่วยประมวลผลที่ใช้เวลาเปรียบเทียบมากที่สุดจะทำงานเสร็จ แนวคิดของการประยุกต์ใช้ระบบประมวลผลแนวขนานแบบ ข้อคำถามขนาน จึงได้ถูกนำมาใช้งาน โดยระบบจะทำการกระจายข้อคำถามย่อยไปยังหน่วยประมวลผลต่าง ๆ ที่ได้รับมอบหมายฐานข้อมูลให้ทำการเปรียบเทียบไว้แล้วจนกระทั่งฐานข้อมูลที่

ได้รับมอบหมายไม่มีข้อคำถามที่ต้องทำการเปรียบเทียบเหลืออยู่อีก แล้วรับมอบหมายฐานข้อมูลที่ยังทำการเปรียบเทียบไม่เสร็จมาประมวลผล



รูปที่ 4.1 ระบบการประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์

จากรูปที่ 4.1 ระบบการประมวลผลเชิงขนานจะทำการส่งข้อคำถามเดียวกันไปยังหน่วยประมวลผลต่าง ๆ ของระบบ โดยทั้งนี้ฐานข้อมูลอาจสามารถจัดเก็บในหน่วยความจำของหน่วยประมวลผลเพียงจำนวนหนึ่งโดยไม่จำเป็นต้องใช้ครบทุกหน่วยประมวลผล เราอาจทำการแบ่งการทำงานออกเป็นสองชุด หรือมากกว่าสองชุดขึ้นไป เพื่อความรวดเร็วในการประมวลผล ในทางกลับกันหากขนาดของฐานข้อมูลโดยรวมมีขนาดใหญ่กว่าหน่วยความจำของระบบ เราอาจทำการแบ่งการประมวลผลออกเป็นหลาย ๆ รอบ แล้วทำการประมวลผลเฉพาะในส่วนที่สามารถจัดเก็บในหน่วยความจำได้ก่อน หลังจากนั้นจึงกลับมาประมวลผลในส่วนที่เหลือ โดยทั้งนี้จะต้องคำนึงถึงค่าใช้จ่ายรายหัวที่เกิดขึ้นจากการกระจายข้อคำถามและรวบรวมผลลัพธ์ที่ได้

ระบบการประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์นอกจากถูกออกแบบเพื่อเพิ่มประสิทธิภาพของระบบแล้ว ยังคำนึงถึงปัจจัยอื่นๆดังต่อไปนี้

1. ความเข้ากันได้ของระบบ (compatibility) เนื่องจากโปรแกรมบลาสท์มีหลายฐานงานซึ่งสนับสนุนระบบปฏิบัติการส่วนใหญ่ ระบบจึงควรจะสามารถสนับสนุนการทำงานของโปรแกรมบลาสท์ในฐานงานอื่นๆ และสามารถที่จะทำงานพร้อมกันข้ามฐานงานได้ ระบบจึงถูกออกแบบให้ถูกพัฒนาขึ้นโดยใช้ภาษาจาวาและใช้โปรโตคอลแบบ TCP/IP ในการแลกเปลี่ยนข้อมูลระหว่างหน่วยประมวลผล
2. ความสามารถปรับเปลี่ยนให้เหมาะสมกับระบบขนาดต่างๆ (scalability) ระบบควรมีความสามารถในการรองรับจำนวนของหน่วยประมวลผลต่างๆ กันได้ โดยสามารถใช้ประสิทธิภาพได้สูงสุดเท่าที่ระบบจะทำได้
3. ความทนทานต่อการผิดพลาดของระบบ (fault-tolerance) ระบบควรมีความสามารถที่จะสามารถรองรับข้อผิดพลาดที่อาจเกิดขึ้นระหว่างการใช้งานโดยทั้งนี้หากหน่วยประมวลผลใดหน่วยประมวลผลหนึ่งของระบบไม่สามารถใช้งานได้ ทำงานผิดพลาด หรือมีเหตุขัดข้องให้หลุดจากระบบ ระบบควรจะสามารถแก้ไขข้อผิดพลาดที่เกิดขึ้นได้
4. ความคล่องตัวของระบบ (flexibility) ระบบควรจะมีความคล่องตัวสูงโดยทั้งนี้เราสามารถเพิ่มหรือลดจำนวนหน่วยประมวลผลเพื่อเพิ่มประสิทธิภาพของระบบขณะใช้งานได้

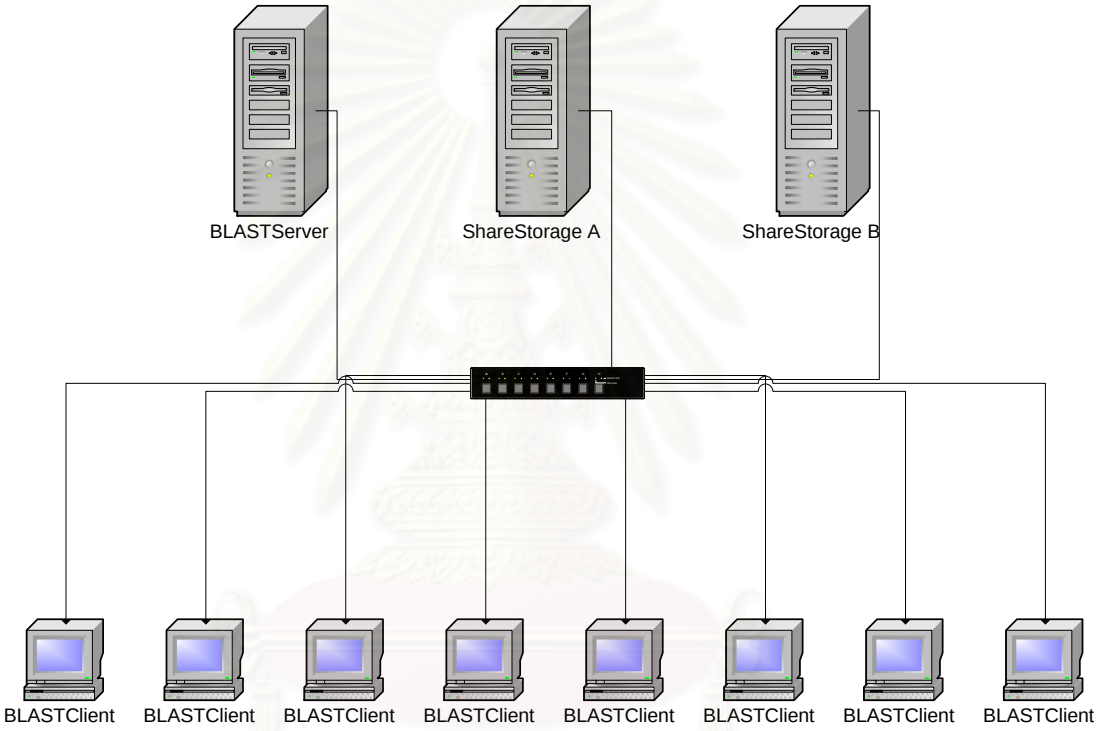
4.1.2 **โครงสร้างของระบบ**

โครงสร้างของระบบประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์ถูกออกแบบให้อยู่ในรูปแบบของระบบรับ-ให้บริการในเครือข่ายโดยจะประกอบด้วยสามส่วนหลักๆ คือ BlastServer BlastClient และ SharedStorage โดยในแต่ละส่วนจะมีหน้าที่ต่างๆ กันดังนี้คือ

1. BLASTServer จะทำหน้าที่ตัดแบ่งข้อคำถามออกเป็นลำดับย่อยๆ โดยจะคอยรับการลงทะเบียนของ BLASTClient ต่างๆ และทำการกำหนดฐานข้อมูลรวมทั้งคอยจัดการส่งคำสั่งในการประมวลผลของโปรแกรมบลาสท์ให้ BLASTClient แต่ละหน่วยนำไปประมวลผลตลอดจนถึงการรวบรวมและเชื่อมต่อผลลัพธ์ที่ได้จากการประมวลผล

2. BLASTClient มีหน้าที่ในการรับคำสั่งและตัวแปรเสริมสำหรับแฟ้มกระทำการ blastall ตลอดจนข้อคำถามและฐานข้อมูลที่ได้รับมอบหมายเพื่อใช้ในการเปรียบเทียบ
3. ShareStorage มีหน้าที่เก็บแฟ้มข้อมูลรับเข้าซึ่งเป็นข้อคำถามที่ถูกตัดแบ่งแล้ว และเก็บแฟ้มข้อมูลผลลัพธ์ที่ได้เพื่อใช้ในการรวบรวมผลลัพธ์เข้าด้วยกัน

ทั้งนี้ส่วนประกอบต่างๆ อาจรวมอยู่ในหน่วยประมวลผลเดียวหรือแยกออกไปต่างหากเพื่อเพิ่มประสิทธิภาพของระบบ



รูปที่ 4.2 โครงสร้างของระบบประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์

จากรูปที่ 4.2 โครงสร้างของระบบจะประกอบไปด้วย BLASTServer หนึ่งตัวและ ShareStorage กี่ตัวก็ได้ขึ้นอยู่กับความสามารถของระบบปฏิบัติการของเครื่องบริการแฟ้ม ทั้งนี้อาจทำการปรับเปลี่ยนให้เหมาะสมโดยอาจใช้ ShareStorage สองตัวเพื่อเก็บข้อคำถามและผลลัพธ์ หรืออาจทำการเก็บข้อคำถามและผลลัพธ์ใน ShareStorage เดียวกันก็ได้

BLASTClient ซึ่งเป็นหน่วยประมวลผลย่อยของระบบก็เช่นเดียวกันสามารถที่จะรวมอยู่ในเครื่องเดียวกันกับ BLASTServer หรือ ShareStorage ก็ได้หรืออาจแยกออกมาเพื่อลดภาระในการทำงานของ BLASTServer ก็ได้ ทั้งนี้ BLASTClient แต่ละตัวจะต้องมีฐานข้อมูลที่ตรงกันและได้ทำการสร้างดัชนีในการเปรียบเทียบไว้แล้ว

4.1.3 หลักการกระจายข้อคำถาม

การกระจายข้อคำถามของ BLASTServer จะต้องพิจารณาจากภาระของแต่ละหน่วยประมวลผลของระบบ โดยพยายามที่จะทำให้แต่ละหน่วยประมวลผลมีความจำเป็นต้องสับเปลี่ยนค่าข้อมูลระหว่างหน่วยความจำหลักและจานบันทึกข้อมูลให้น้อยที่สุดเท่าที่จะเป็นไปได้ และ เนื่องด้วยระบบอาจเกิดข้อผิดพลาดที่เกิดจากหน่วยประมวลผลใดๆ ซึ่งมีความจำเป็นที่จะต้องกลับไปประมวลผลซ้ำ รวมถึงการหลีกเลี่ยงปัญหาที่อาจเกิดจากการรอกันระหว่างหน่วยประมวล ผลระบบจึงถูกออกแบบให้ทำการประมวลผลเบื้องต้น โดยการตัดแบ่งข้อคำถามออกเป็นลำดับเดี่ยว จัดเก็บใส่แฟ้มชั่วคราวเพื่อส่งไปประมวลผล โดย BLASTServer จะทำการกำหนดฐานข้อมูลให้แต่ละ BLASTClient รับผิดชอบตามโครงสร้างข้อมูลดังรูปที่ 4.3

DB1	DB2	...	DB4
0	0	…	0
0	0	…	0
:	:	.:.	:
0	0	…	0

รูปที่ 4.3 โครงสร้างข้อมูลแบบแถวลำดับสำหรับจัดเก็บหมายเลขของ BLASTClient ที่ได้รับจากการเข้าลงทะเบียนในระบบ

จากรูป 4.3 ระบบจะทำการสร้างโครงสร้างข้อมูลแบบแถวลำดับ (Array) โดยมีจำนวนสดมภ์ขนาดเท่ากับจำนวนฐานข้อมูลที่ถูกแบ่งแยก และมีจำนวนแถวเท่ากับจำนวน BLASTClient สูงสุดที่ระบบจะรับได้ เพื่อกำหนดฐานข้อมูล (DB) ที่แต่ละ BLASTClient จะต้องทำการรับผิดชอบ โดยจะกำหนดให้ค่าตั้งต้นของโครงสร้างแถวลำดับเป็น 0

ในกรณีที่มี BLASTClient ทำการลงทะเบียนเข้าสู่ระบบ BLASTServer จะทำการกำหนดฐานข้อมูลให้โดยการเปลี่ยนค่าในโครงสร้างแถวลำดับเป็นค่ารหัสของ BLASTClient ที่ทำการลงทะเบียนดังรูปที่ 4.4

DB1	DB2	DB3	DB4
CID01	CID02	CID03	CID04
CID05	0	0	0
:	:	:	:
0	0	0	0

รูปที่ 4.4 ตัวอย่างการเข้าลงทะเบียนของ BLASTClients

จากรูป 4.4 กำหนดให้ฐานข้อมูลถูกแบ่งออกเป็น 4 ส่วน BLASTServer จะทำการลงทะเบียน BLASTClient ตามลำดับก่อนหลังโดยเริ่มจาก CID01, CID02, CID03, CID04 และ CID05 ตามลำดับ ทั้งนี้เพื่อทำการกระจายภาระของการเปรียบเทียบลำดับกับฐานข้อมูลให้มีความเท่าเทียมมากที่สุด จากตัวอย่างในที่นี้ BLASTClient รหัส CID01 ได้รับมอบหมายให้ใช้ฐานข้อมูล DB1 ในการเปรียบเทียบ

ในระหว่างการประมวลผล BLASTServer จะทำการกระจายข้อคำถามย่อยในรูปแบบของลำดับ (Sequence) ไปตาม BLASTClient โดยจะใช้โครงสร้างข้อมูลแบบแถวลำดับในการบอกสถานะของการประมวล ผลของแต่ละหน่วยประมวลผลดังรูปที่ 4.5

	DB1	DB2	…	DB4
Sequence1	0	0	…	0
Sequence2	0	0	…	0
:	:	:	…	:
Sequencen	0	0	…	0

รูปที่ 4.5 โครงสร้างข้อมูลแบบแถวลำดับสำหรับจัดเก็บสถานะของลำดับย่อยต่างๆ

จากรูป 4.5 ระบบจะทำการสร้างโครงสร้างข้อมูลแบบแถวลำดับ โดยมีจำนวนสดมภ์ขนาดเท่ากับจำนวนฐานข้อมูลที่ถูกแบ่งแยก และมีจำนวนแถวเท่ากับจำนวนของลำดับย่อยทั้งหมดที่ถูกแบ่ง โดยจะกำหนดให้ค่าตั้งต้นของโครงสร้างแถวลำดับเป็น 0

ในระหว่างการประมวลผล BLASTServer จะทำการกระจายข้อคำถามย่อยไปตาม BLASTClient ต่างๆ โดยจะคำนึงถึงฐานข้อมูลที่แต่ละ BLASTClient ได้รับมอบหมายดังตัวอย่างในรูปที่ 4.6

	DB1	DB2	DB3	DB4
Sequence1	CID01	CID02	CID03	CID04
Sequence2	CID05	0	0	0
:	:	:	:	:
Sequencen	0	0	0	0

รูปที่ 4.6 ตัวอย่างการกำหนดสถานะของการเปรียบเทียบลำดับ

จากรูป 4.6 กำหนดให้ฐานข้อมูลถูกแบ่งออกเป็น 4 ส่วนและข้อคำถามถูกแบ่งออกเป็นจำนวน n ลำดับ BLASTServer จะทำการส่งคำสั่งและตัวแปรเสริมในการประมวลผลของโปรแกรมบลาสท์ไปยังแต่ละ BLASTClient ตามลำดับก่อนหลังโดยเริ่มจาก CID01, CID02, CID03, CID04 และ CID05 ตามลำดับ ทั้งนี้จะดูจากโครงสร้างข้อมูลแบบแถวลำดับที่ใช้บอกฐานข้อมูลที่ BLASTClient รับผิดชอบอยู่ โดยค่าของสถานะลำดับที่กำลังถูกเปรียบเทียบในแต่ละฐานข้อมูลจะถูกแทนที่ด้วยค่ารหัสของ BLASTClient ที่ส่งไปเปรียบเทียบ

เมื่อ BLASTClient ใดๆ ทำการเปรียบเทียบลำดับและฐานข้อมูลที่กำหนดให้เสร็จสิ้นจะทำการเขียนแฟ้มข้อมูลผลลัพธ์ลงบน Share Storage แล้วทำการแจ้งกลับไปยัง BLASTServer ว่าได้ทำการเปรียบเทียบเสร็จสิ้นแล้ว หลังจากนั้น BLASTServer จะทำการบันทึกสถานะแล้วกำหนดแฟ้มข้อมูลนำเข้าใหม่ให้กับ BLASTClient เพื่อทำการเปรียบเทียบต่อไปดังรูปที่ 4.7

	DB1	DB2	DB3	DB4
Sequence1	-1	-1	CID03	-1
Sequence2	CID05	CID02	0	CID04
Sequence3	CID01	0	0	0
:	:	:	:	:
Sequencen	0	0	0	0

รูปที่ 4.7 ตัวอย่างการกำหนดสถานะของการเปรียบเทียบให้กับลำดับใหม่เมื่อมีการแจ้งจาก BLASTClient ว่าการเปรียบเทียบลำดับที่ได้รับมอบหมายสิ้นสุดลง

จากรูป 4.7 กำหนดให้ BLASTClient รหัส CID01, CID02, และ CID04 ทำการเปรียบเทียบลำดับที่ได้รับมอบหมายเสร็จสิ้น BLASTServer จะทำการเปลี่ยนสถานะเดิมของลำดับที่ถูกเปรียบเทียบเป็นสถานะ -1 เพื่อบอกว่าลำดับนั้นๆ ได้ถูกทำการเปรียบเทียบเสร็จเรียบร้อยแล้ว หลังจากนั้น BLASTServer จะกำหนดแฟ้มข้อมูลนำเข้าใหม่ให้ BLASTClient นำไปประมวลผลโดยการแทนที่สถานะ 0 ด้วยรหัสของ BLASTClient เพื่อบอกว่าลำดับใดกำลังถูกประมวลผล

และหาก BLASTClient หน่วยใดหน่วยหนึ่งหลุดไปจากระบบ BLASTServer จะทำการทำการถอนการลงทะเบียนออกจากระบบให้กับ BLASTClient รหัสนั้นๆ โดยการเปลี่ยนค่ารหัสของ BLASTClient ด้วยค่าตั้งต้นคือ 0 ลงในแถวลำดับที่ใช้สำหรับจัดเก็บและบอกฐานข้อมูลที่ต้องทำการรับผิดชอบให้กับ BLASTClient ต่างๆ ดังรูปที่ 4.8 และทำการเปลี่ยนสถานะของลำดับที่ BLASTClient นั้นๆ กำลังทำงานอยู่ให้เป็นค่าตั้งต้นเช่นกันดังรูป 4.9

DB1	DB2	DB3	DB4
0	CID02	CID03	CID04
CID05	0	0	0
:	:	:	:
0	0	0	0

รูปที่ 4.8 ตัวอย่างการถอนการลงทะเบียนของ BLASTClient

จากรูปที่ 4.8 ต่อเนื่องมาจากรูปที่ 4.6 BLASTClient รหัส CID01 ทำงานผิดพลาดหรือหลุดออกไปจากระบบ BLASTServer จึงทำการปรับค่ารหัสของ BLASTClient ให้เป็นค่าตั้งต้นเหมือนเดิม

	DB1	DB2	DB3	DB4
Sequence1	-1	-1	CID03	-1
Sequence2	CID05	CID02	0	CID04
Sequence3	0	0	0	0
:	:	:	:	:

Sequencen	0	0	0	0

รูปที่ 4.9 ตัวอย่างการปรับเปลี่ยนสถานะของลำดับเมื่อ BLASTClient ถูกถอนออกจากการลงทะเบียน

จากรูปที่ 4.9 ต่อเนื่องมาจากรูปที่ 4.7 รหัสของ BLASTClient หมายเลขที่ CID01 ถูกแก้ให้กลับคืนสู่ค่าตั้งต้นเพื่ออนุญาตให้ทำการเปรียบเทียบซ้ำโดย BLASTClient อื่นได้ในภายหลัง

หากฐานข้อมูลใดได้รับการเปรียบเทียบจนเสร็จแล้วดังรูปที่ 4.10 ระบบจะทำการกำหนดฐานข้อมูลให้ BLASTClient ที่รับผิดชอบฐานข้อมูลนั้นไปรับผิดชอบเปรียบเทียบฐานข้อมูลอื่นต่อไปดังรูปที่ 4.11

	DB1	DB2	DB3	DB4
Sequence1	-1	-1	CID03	-1
Sequence2	-1	CID02	0	CID04
Sequence3	-1	0	0	0
Sequence4	-1	0	0	0
Sequence5	-1	0	0	0
Sequence6	-1	0	0	0

รูปที่ 4.10 ตัวอย่างการกำหนดฐานข้อมูลใหม่ให้กับ BLASTClient

จากรูปที่ 4.10 กำหนดให้จำนวนลำดับมีทั้งหมด 6 ลำดับ และ จำนวนลำดับทั้ง 6 ได้ถูกนำไปเปรียบเทียบกับฐานข้อมูล DB1 จนเสร็จสิ้นแล้วดังจะเห็นได้ว่าสถานะของลำดับทั้งหมดถูกเปลี่ยนจาก 0 เป็น -1 โดย BLASTClient รหัส CID05 ทำงานจนเสร็จสิ้นแล้วและไม่มีงานเหลืออีก BLASTServer จะทำการกำหนดฐานข้อมูลให้ BLASTClient รหัส CID05 รวมถึง BLASTClient รหัสอื่นๆ ที่รับผิดชอบฐานข้อมูล DB1 อยู่ โดยการเปลี่ยนค่าข้อมูลในแถวลำดับของฐานข้อมูลเป็น -1 เพื่อไม่อนุญาตให้ BLASTClient ทำการลงทะเบียนได้อีกแล้วจึงกำหนดฐานข้อมูลใหม่ให้กับ BLASTClient ที่เคยรับผิดชอบฐานข้อมูล DB1 อยู่ดังรูปที่ 4.11

DB1	DB2	DB3	DB4
-1	CID02	CID03	CID04
-1	CID05	0	0
-1	0	0	0
-1	0	0	0
-1	0	0	0

รูปที่ 4.11 ตัวอย่างการกระจายข้อคำถามหลังจากกำหนดฐานข้อมูลให้กับ BLASTClient

จากรูปที่ 4.11 กำหนดให้จำนวน BLASTClient ที่ระบบรับได้คือ 5 หน่วย เนื่องจาก BLASTClient รหัส CID05 ได้ทำการเปรียบเทียบเสร็จสิ้นแล้ว BLASTServer จึงไม่อนุญาตให้ BLASTClient สามารถเข้าลงทะเบียนและรับผิดชอบฐานข้อมูล DB1 ได้อีกจึงเปลี่ยนค่าข้อมูลในแถวลำดับให้เป็น -1 และกำหนดให้ BLASTClient ทำการรับผิดชอบฐานข้อมูลอื่นแทนโดยในที่นี้คือ DB2

การกำหนดฐานข้อมูลให้ BLASTClient ตายตัวจนกว่าจะได้รับฐานข้อมูลใหม่ช่วยให้ BLASTClient ซึ่งมีหน้าที่ในการเปรียบเทียบข้อมูลไม่ต้องทำการสับเปลี่ยนค่าข้อมูลของหน่วยความจำหลักกับจานบันทึกข้อมูลได้ นอกจากนั้นแล้วการตัดแบ่งข้อคำถามออกเป็นส่วนย่อยๆ ยังสามารถแบ่งภาระงานของแต่ละ BLASTClient ได้อย่างเท่าเทียมกัน รวมถึงความสามารถในการกลับไปทำการประมวลผลซ้ำเมื่อเกิดข้อผิดพลาดซึ่งอาจเกิดจาก BLASTClient ไม่สามารถทำงานต่อได้ทั้งยังเพิ่มความคล่องตัวของระบบ โดยหากเกิดความต้องการเปรียบเทียบลำดับขนาดมากๆ ในเวลาที่น้อยลงเราสามารถเพิ่มหรือลดจำนวน BLASTClient ขณะใช้งานระบบได้อีกด้วย

ทั้งนี้ ในการพัฒนาระบบ จำเป็นจะต้องคำนึงถึงปัญหาการประสานเวลา ที่อาจเกิดขึ้นได้ (synchronization problem) เนื่องจากการอ่านและเขียนลงบนหน่วยความจำเดียวกันในเวลาเดียวกัน เพราะฉะนั้นก่อนที่จะอนุญาตให้ทำการอ่านหรือเขียนลงบนแถวลำดับ จำเป็นจะต้องทำการปิดตาย (lock) หน่วยความจำที่จะทำการอ่านและเขียนไว้ก่อนเพื่อหลีกเลี่ยงความผิดพลาดที่อาจเกิดขึ้นได้

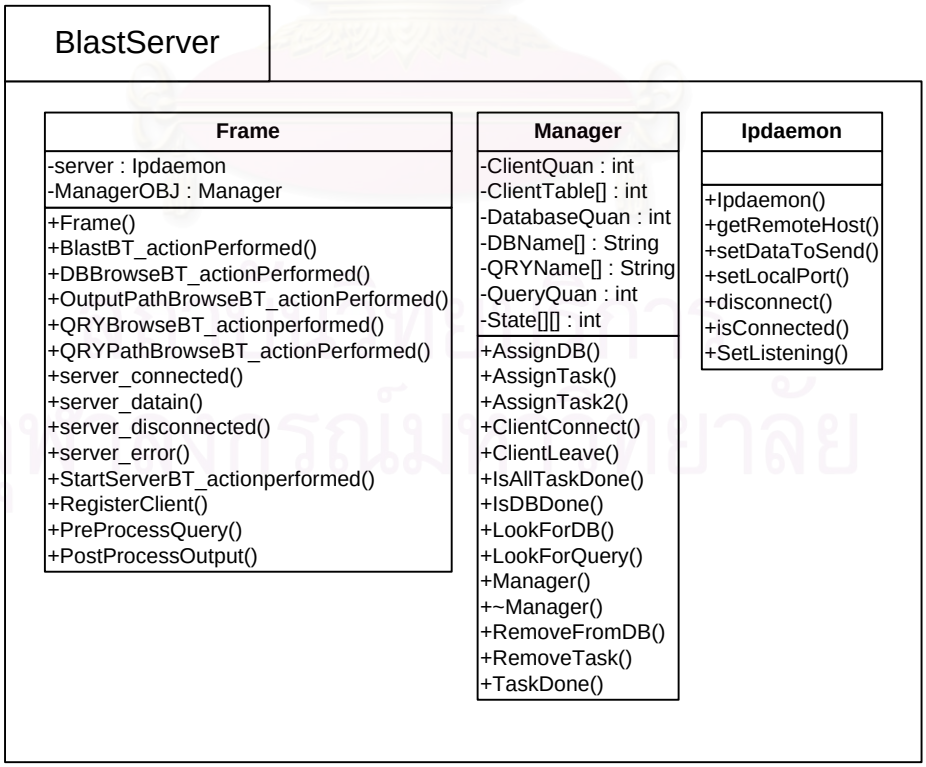
4.2 ต้นแบบระบบการประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์

ต้นแบบระบบการประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์ที่ได้ทำการพัฒนาขึ้นถูกตั้งชื่อเพื่อ ให้เกียรติแก่สถาบันการศึกษาว่า ระบบประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์แห่งจุฬาลงกรณ์มหาวิทยาลัย (Chulalongkorn University's Parallel System for BLAST – CUP BLAST) โดยส่วนที่ถูกพัฒนามีรายละเอียดแบ่งได้เป็นส่องส่วนคือ BLASTServer และ BLASTClient ซึ่งถูกออกแบบโดยใช้การเขียนโปรแกรมเชิงวัตถุ (object-oriented programming) แบ่งออกเป็นสองแพคเกจ (package) ดังนี้

1. BLASTServer
2. BLASTClient

4.2.1 แพคเกจ BLASTServer

แพคเกจ BLASTServer มีหน้าที่ตัดแบ่งข้อคำถามออกเป็นส่วนย่อยๆ โดยจะคอยรับการลงทะเบียนของ BLASTClient ต่างๆ และทำการกำหนดฐานข้อมูลรวมทั้งคอยจัดการส่งคำสั่งในการประมวลผลของโปรแกรมบลาสท์ให้แต่ละ BLASTClient นำไปประมวลผลตลอดจนถึงการรวบรวมและเชื่อมต่อผลลัพธ์ที่ได้จากการประมวลผล โดยจะมีคลาสที่สำคัญดังแบบจำลองคลาสในรูปที่ 4.12 ดังต่อไปนี้



รูปที่ 4.12 แบบจำลองคลาสสำหรับแพคเกจ BLASTServer

4.2.1.1 **คลาส** Frame เป็นคลาสที่เป็นส่วนต่อประสานกับผู้ใช้งาน และทำหน้าที่ในการประมวลผลเบื้องต้นโดยการตัดแบ่งข้อคำถามออกเป็นแฟ้มชั่วคราวย่อยๆ ตลอดจนถึงการรวมผลลัพธ์เข้าด้วยกัน ประกอบด้วยตัวกระทำการที่สำคัญดังต่อไปนี้

1. StartServerBT_actionperformed() เป็นตัวกระทำการที่ใช้สำหรับเรียกให้ server ทำหน้าที่ให้บริการข้อมูลและเริ่มการรับการลงทะเบียนจาก BLASTClient ด้วยการสร้างอ๊อบเจกต์ server ขึ้นโดยตัวกระทำการนี้จะถูกเรียกใช้โดยตรงจากผู้ใช้งานระบบ
2. BlastBT_actionPerformed() เป็นตัวกระทำการที่ผู้ใช้จะทำการเรียกให้โปรแกรมบลาสท์ถูกเรียกใช้งาน โดยการสร้างอ๊อบเจกต์ ManagerOBJ ขึ้นจากจำนวนฐานข้อมูลและจำนวนข้อคำถามที่ถูกแบ่ง ทั้งนี้ตัวกระทำการนี้จะไม่สามารถใช้งานได้จนกว่าจะเรียกใช้ตัวกระทำการ StartServerBT_actionperformed() ก่อน
3. PreProcessQuery() เป็นตัวกระทำการที่ใช้สำหรับตัดแบ่งข้อคำถามออกเป็นแฟ้มชั่วคราวย่อยๆ โดยจะตัดข้อคำถามให้เป็นลำดับเดี่ยว
4. PostProcessOutput() เป็นตัวกระทำการที่จะทำการรวบรวมผลลัพธ์ที่ได้จากการเปรียบเทียบเข้าด้วยกันจะถูกเรียกใช้เองเมื่อการเปรียบเทียบสิ้นสุดลง
5. RegisterClient() เป็นตัวกระทำการที่จะทำการลงทะเบียนเบื้องต้นและจัดเก็บรหัสของ BLASTClient เพื่อบอกให้ระบบรู้ว่าขณะนี้มี BLASTClient ใดบ้างที่สามารถส่งข้อคำถามไปประมวลผลได้
6. server_connected() เป็นตัวกระทำการที่ถูกเรียกใช้งานเมื่อเกิดเหตุการณ์ที่มี BLASTClient ใหม่ขอทำการลงทะเบียนเข้าสู่ระบบ
7. server_datain() เป็นตัวกระทำการที่ถูกเรียกใช้เมื่อได้รับการส่งข้อมูลจาก BLASTClient โดยจะทำการบอกอ๊อบเจกต์ ManagerOBJ ว่า BLASTClient ได้ทำการเปรียบเทียบลำดับที่ได้รับมอบหมายเสร็จสิ้นแล้ว

8. server_disconnect() เป็นตัวกระทำการที่ถูกเรียกใช้งานเมื่อเกิดเหตุการณ์ที่เกิดจาก BLASTClient หายไปจากระบบหรือร้องขอทำการถอนการลงทะเบียนจากระบบ
9. server_error() เป็นตัวกระทำการที่จะทำการบันทึกข้อมูล เมื่ออ๊อบเจกต์ server ใน BLASTServer เกิดความผิดพลาดหรือไม่สามารถใช้งานได้อีกต่อไป

4.2.1.2 **คลาส Manager** เป็นคลาสที่มีหน้าที่จัดการรับการลงทะเบียนของ BLASTClient ต่างๆ และทำการกำหนดฐานข้อมูลรวมทั้งคอยจัดการส่งคำสั่งในการประมวลผลของโปรแกรมบลาสท์ให้แต่ละ BLASTClient นำไปประมวลผล โดยมีตัวกระทำการที่สำคัญดังต่อไปนี้

1. AssignDB() เป็นตัวกระทำการที่จะทำการกำหนดฐานข้อมูลรับผิดชอบให้กับ BLASTClient โดยจะกระจายภาระให้เกิดการสับเปลี่ยนค่าข้อมูลในแต่ละ BLASTClient ให้น้อยที่สุด
2. AssignTask() เป็นตัวกระทำการที่จะทำการเลือกแฟ้มข้อมูลนำเข้าและกำหนดคำสั่งในรูปแบบการเรียกใช้แฟ้มกระทำการพร้อมด้วยตัวแปรเสริมต่างๆ เพื่อส่งให้ BLASTClient นำไปใช้ประมวลผล
3. AssignTask2() เป็นตัวกระทำการที่จะคอยตรวจสอบว่าจะฐานข้อมูลใดๆ หรือฐานข้อมูลทั้งหมด ถูกเปรียบเทียบเสร็จแล้ว และพิจารณาว่าจะกำหนดให้ BLASTClient ไปรับผิดชอบฐานข้อมูลอื่นต่ออีกหรือไม่โดยทำการเลือกเรียกใช้ตัวกระทำการ AssignDB() และ AssignTask() ตามเงื่อนไขที่เกิดขึ้น
4. ClientConnect() เป็นตัวกระทำการที่คอยกำหนดฐานข้อมูลรับผิดชอบแรกให้กับ BLASTClient ที่ได้ทำการลงทะเบียนเข้ามาในระบบ
5. ClientLeave() เป็นตัวกระทำการที่จะทำการยกเลิกการลงทะเบียนของ BLASTClient และทำการปรับเปลี่ยนสถานะของลำดับที่กำลังถูก BLASTClient ีที่ถูกแจ้งให้กลายเป็นสถานะตั้งต้นเพื่อทำการประมวลผลใหม่
6. RemoveFromDB() เป็นตัวกระทำการที่จะถอนการลงทะเบียบของ BLASTClient ออกโดยจะถูกเรียกใช้โดยตัวกระทำการ ClientLeave()

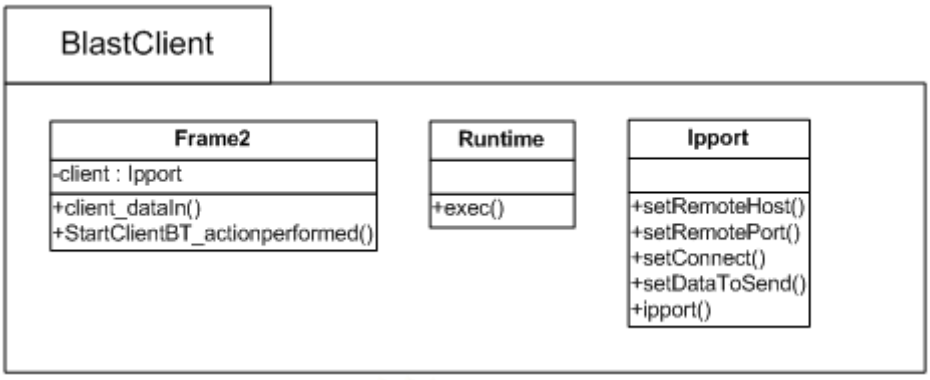
7. RemoveTask() เป็นตัวกระทำการที่คอยปรับเปลี่ยนสถานะของลำดับที่ถูกยกเลิกเมื่อเกิดการถอนการลงทะเบียนโดยจะถูกเรียกใช้โดยตัวกระทำการ ClientLeave() เช่นกัน

4.2.1.3 คลาส lpdaemon เป็นคลาสที่มีหน้าที่เป็นส่วนต่อประสานงานกับ BLASTClient โดยจะทำหน้าที่เป็นผู้ให้บริการเครือข่ายโดยผ่านโปรโตคอลแบบ TCP/IP มีตัวกระทำการที่สำคัญดังต่อไปนี้

1. getRemoteHost() เป็นตัวกระทำการที่จะทำให้ทราบถึงรหัสของ BLASTClient ที่ถูกกำหนดขึ้นโดยอัตโนมัติเมื่อมีการเชื่อมต่อจาก BLASTClient
2. setDataToSend() เป็นตัวกระทำการที่จะทำการส่งข้อมูลไปยัง BLASTClient
3. setLocalPort() เป็นตัวกระทำการที่ใช้ในการบอกหมายเลขของช่องทางเข้า/ออกของข้อมูล
4. disconnect() เป็นตัวกระทำการที่ใช้สำหรับการตัดการเชื่อมต่อกับ BLASTClient
5. isConnected() เป็นตัวกระทำการที่ใช้สำหรับตรวจสอบว่า BLASTClient ที่ทำการเชื่อต่ออยู่ยังทำงานได้อยู่หรือไม่
6. SetListening() เป็นตัวกระทำการเพื่อเริ่มยอมรับการให้บริการระบบเครือข่ายแบบ TCP/IP เพื่อให้ BLASTClient สามารถเข้าเชื่อมต่อได้

4.2.2 แพคเกจ BLASTClient

แพคเกจ BLASTClient มีหน้าที่ในการรับคำสั่งและตัวแปรเสริมสำหรับแฟ้มกระทำการ blastall ตลอดจนข้อคำถามและฐานข้อมูลที่ได้รับมอบหมายเพื่อใช้ในการเปรียบเทียบโดยจะมีคลาสที่สำคัญดังแบบจำลองคลาสในรูปที่ 4.13 ดังต่อไปนี้



รูปที่ 4.13 แบบจำลองคลาสสำหรับแพคเกจ BLASTClient

4.2.2.1 **คลาส Frame2** เป็นคลาสที่เป็นส่วนต่อประสานกับผู้ใช้งาน และทำหน้าที่ในการเริ่มการใช้งาน BLASTClient โดยจะมีตัวกระทำการที่สำคัญดังนี้

1. StartClientBT_actionperformed() เป็นตัวกระทำการที่ใช้เพื่อเริ่มการทำงานของระบบจะถูกเรียกโดยผู้ใช้งานระบบโดยกระตุ้นให้อ๊อบเจกต์ client เริ่มการทำงานโดยลงทะเบียนเข้าสู่ระบบ
2. client_dataIn() เป็นตัวกระทำการที่ถูกเรียกเมื่อมีเหตุการณ์ได้รับข้อมูลจาก BLASTServer

4.2.2.2 **คลาส Runtime** เป็นคลาสที่เป็นส่วนเชื่อมประสานกับระบบปฏิบัติการโดยเรียกใช้งานแฟ้มกระทำการ blastall มีตัวกระทำการที่สำคัญคือ

1. exec() เป็นตัวกระทำการที่ใช้เรียกแฟ้มกระทำการ blastall ถูกเรียกใช้งานอัตโนมัติเมื่อมีคำสั่งให้ประมวลผล

4.2.2.3 **คลาส Ipport** เป็นคลาสที่เป็นส่วนเชื่อมประสานกับ BLASTServer ผ่านโพรโตคอลแบบ TCP/IP โดยจะมีตัวกระทำการที่สำคัญดังนี้

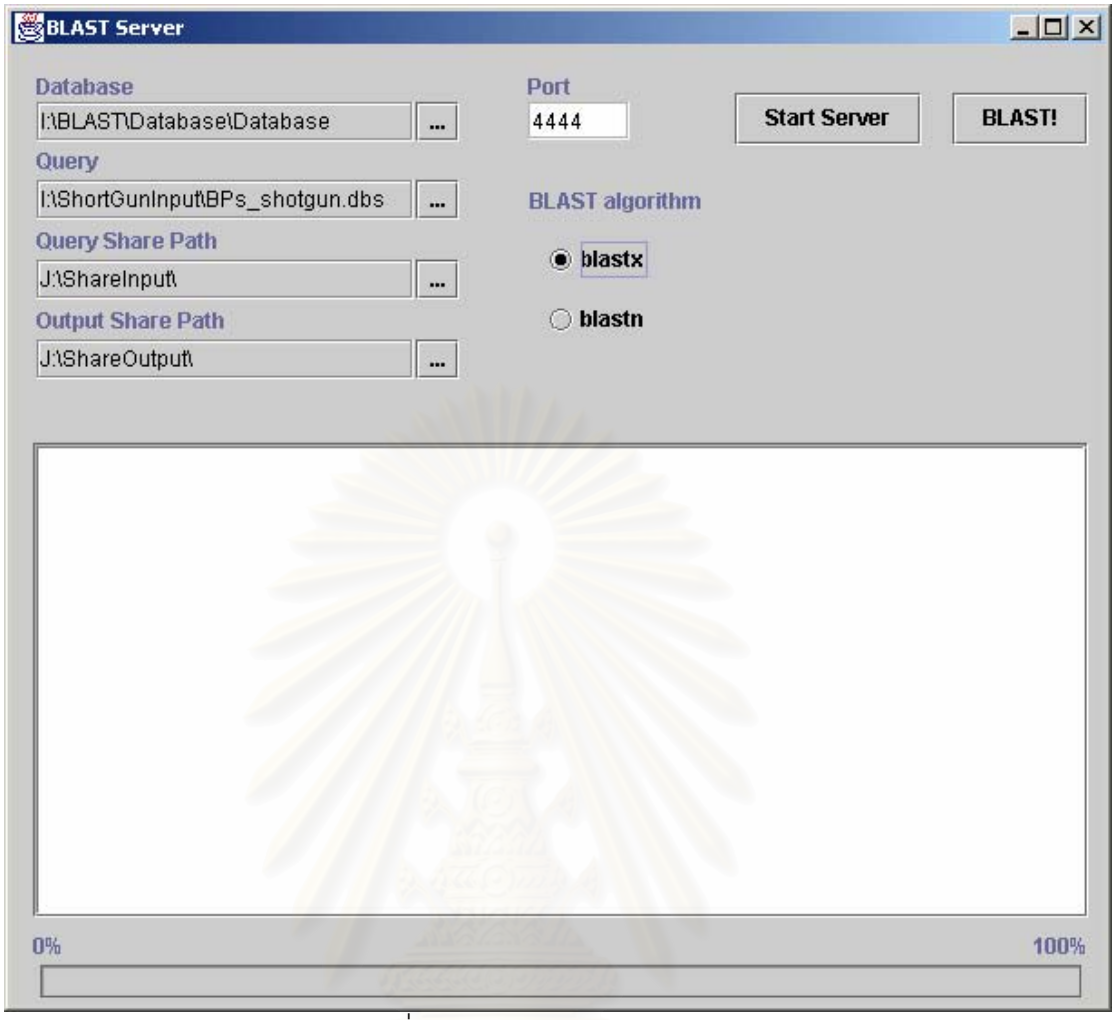
1. setRemoteHost() เป็นตัวกระทำการที่ใช้สำหรับตั้งค่าไอพีแอดเดรส (IP Address) ของ BLASTServer ที่จะเข้าลงทะเบียน
2. setRemotePort() เป็นตัวกระทำการที่ใช้สำหรับตั้งค่าช่องทางเข้า/ออกของข้อมูลระหว่าง BLASTServer และ BLASTClient
3. setConnected() เป็นตัวกระทำการที่ใช้สำหรับเริ่มการเชื่อมต่อกับ BLASTServer โดยทั้งนี้จะต้องทำการตั้งค่าไอพีแอดเดรสและค่าช่องทางเข้า/ออกของข้อมูลด้วยตัวกระทำการ setRemoteHost() และ setRemotePort() เสียก่อน

4. isConnected() เป็นตัวกระทำการที่ใช้ทดสอบว่า BLASTClient ยังเชื่อมต่อกับ BLASTServer อยู่หรือไม่
5. setDataToSend() เป็นตัวกระทำการที่ใช้ในการส่งข้อมูลไปยัง BLASTServer

4.2.3 ส่วนต่อประสานกับผู้ใช้

ส่วนต่อประสานกับผู้ใช้ของระบบประมวลผลแนวขนานก็แบ่งออกเป็นสองส่วนเช่นเดียวกับแพคเกจทั้งสองโดยจะแบ่งออกเป็นหน้าต่างสำหรับ BLASTServer และหน้าต่างของ BLASTClient โดยจะออกแบบให้ใช้งานได้ง่ายกว่าการใช้งาน blastall ตามปกติ

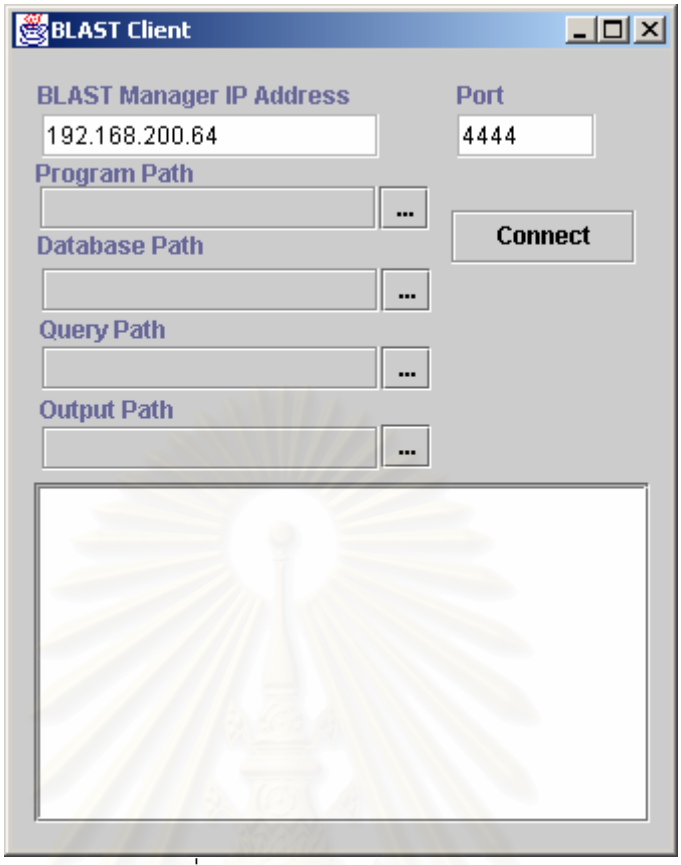
4.2.3.1 หน้าต่าง BLASTServer จะทำการอนุญาตให้ผู้ใช้งานทำการเลือกค้นดูแฟ้มฐานข้อมูลและแฟ้มข้อคำถามซึ่งเป็นข้อมูลนำเข้าของแฟ้มกระทำการ blastall ตลอดจนถึงอนุญาตให้ผู้ใช้เลือกขั้นตอนวิธีในการเปรียบเทียบที่ได้เสนอไปสองแบบคือ blastx และ blastn ทั้งนี้ผู้ใช้งานจะต้องทำการเลือกเส้นทาง (path) สำหรับเข้าใช้ระบบแฟ้มที่ถูกแบ่ง (shared file system) บนเครื่องบริการแฟ้ม (file server) ผู้ใช้งานจะต้องทำการกดปุ่ม Start Server เพื่อทำการเริ่มการทำงานของระบบและกดปุ่ม BLAST! เพื่อเริ่มการประมวลผลโดยจะมีช่องแสดงผลซึ่งจะบอกถึงสถานะต่างๆ ของการทำงานและมีแถบแสดงความก้าวหน้าของการทำงาน โดยจะบอกเป็นอัตราส่วนของจำนวนงานทั้งหมดที่ได้ทำไปแล้วดังรูปที่ 4.14



รูปที่ 4.14 หน้าต่าง BLASTServer

จากรูปที่ 4.14 หน้าต่าง BLASTServer จะมีปุ่มที่อนุญาตให้ผู้ใช้งานเรียกค้นดูแฟ้มข้อมูลต่างๆ เช่นแฟ้มคุณสมบัติของฐานข้อมูลและแฟ้มข้อมูลข้อคำถามตลอดจนถึงการเลือกเส้นทางสำหรับเข้าใช้ระบบแฟ้มที่ถูกแบ่ง มีปุ่มเลือกใช้งานขั้นตอนวิธีในการเปรียบเทียบสองแบบ พื้นที่แสดงผลและสถานะต่างๆ ของระบบ และแถบแสดงความก้าวหน้าของการทำงาน ทั้งนี้ผู้ใช้จำเป็นจะต้องกำหนดหมายเลขของช่องทางเข้า/ออกของข้อมูลให้ตรงกันกับค่าของ BLASTClient

4.2.3.2 **หน้าต่าง** BLASTClient ผู้ใช้มีความจำเป็นจะต้องกำหนดเส้นทางของแฟ้มกระทำการ เส้นทางของฐานข้อมูลที่ถูกตัดแบ่ง ตลอดจนเส้นทางเข้าใช้ระบบแฟ้มที่ถูกแบ่งตลอดจนไอพีแอดเดรส และ หมายเลขของช่องทางเข้า/ออกของข้อมูลของ BLASTServer ดังรูป 4.15

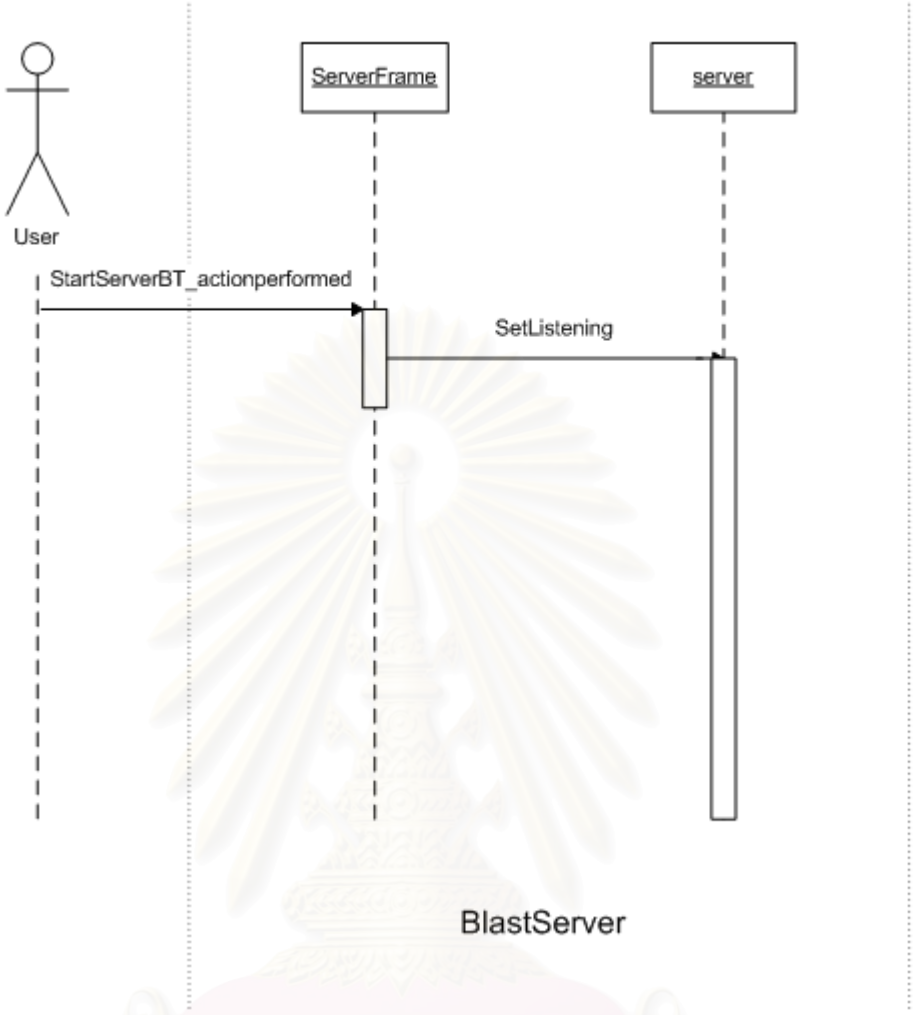


รูปที่ 4.15 หน้าต่าง BLASTClient

จากรูปที่ 4.15 หน้าต่าง BLASTClient จะมีปุ่มให้ผู้ใช้งานเรียกค้นดูเส้นทางเก็บแฟ้มข้อมูลต่างๆ ช่องกำหนดไอพีแอดเดรส และ ช่องกำหนดหมายเลขของช่องทางเข้า/ออกของข้อมูลของ BLASTServer ตลอดจนถึงพื้นที่แสดงสถานะของระบบว่ากำลังทำการเปรียบเทียบลำดับและฐานข้อมูลใดอยู่ โดย BLASTClient จะทำงานได้ก็ต่อเมื่อได้ทำการเชื่อมต่อไปยัง BLASTServer จากการกดปุ่ม Connect ทั้งนี้ข้อมูลเส้นทางต่างๆ ตลอดจนถึงข้อมูลของ BLASTServer จะต้องถูกกำหนดไว้ก่อนที่จะสามารถใช้งานระบบได้

4.2.4 การทำงานของระบบ

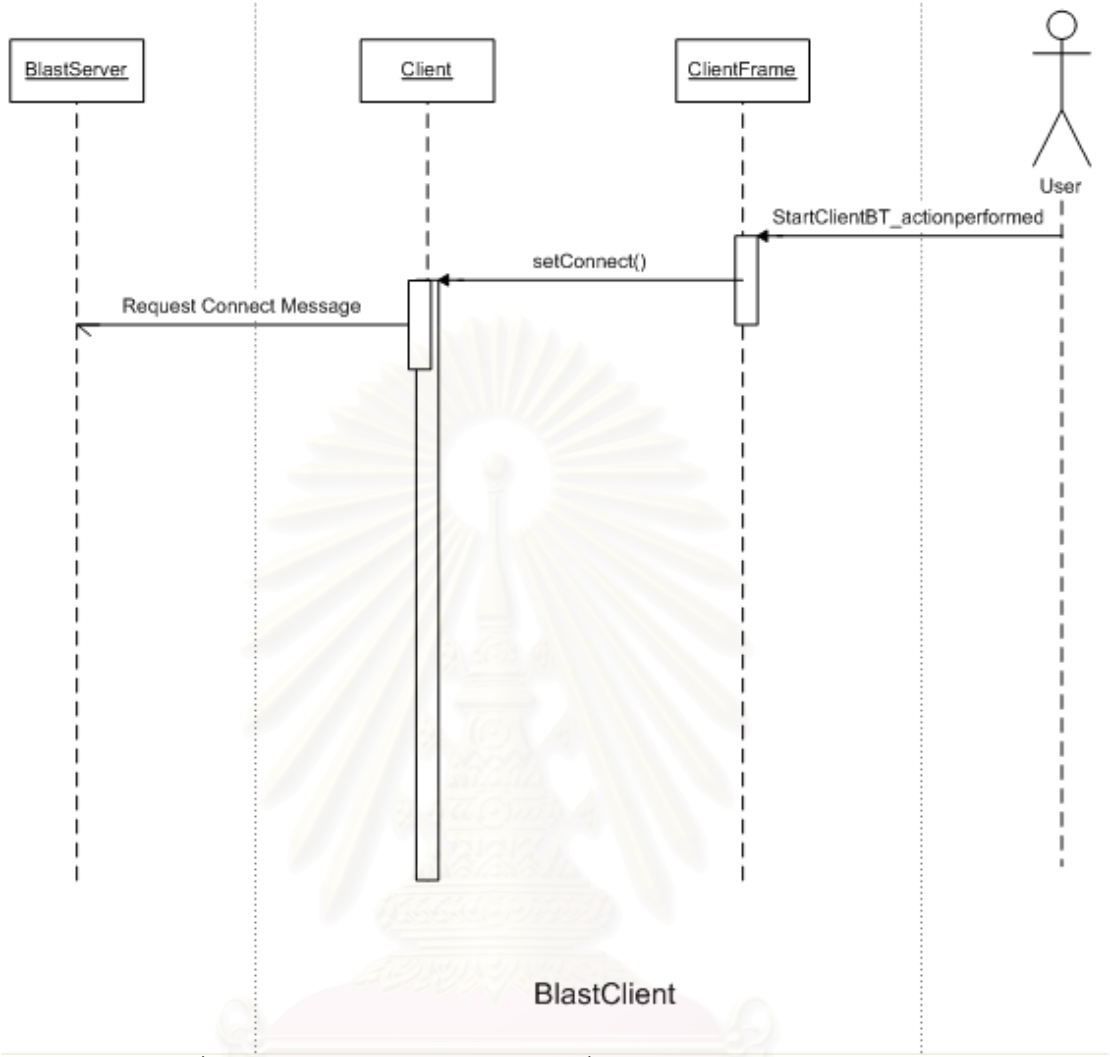
การทำงานของระบบประมวลผลเชิงขนานของโปรแกรมบลาสท์เริ่มจากการติดตั้งเครื่องบริการแฟ้ม หรือ SharedServer โดยการแบ่งเนื้อที่ให้สำหรับเขียน/อ่านแฟ้มข้อมูลแบบเต็มอ่าน/เขียนเต็มรูปแบบ และทำการใส่ระบบแฟ้ม (mount file system) ให้กับ BLASTServer และ BLASTClient ทุกหน่วยที่ต้องการใช้งานระบบ แล้วทำการจัดเตรียมเส้นทางสำหรับการเข้าใช้ระบบแฟ้มให้กับ BLASTServer และ BLASTClient แต่ละหน่วยโดยทั้งนี้เส้นทางของระบบแฟ้มที่ใช้เขียน/อ่านของทั้ง BLASTServer และ BLASTClient จะต้องตรงกัน



รูปที่ 4.16 แผนภาพลำดับเหตุการณ์เมื่อผู้ใช้งานระบบกดปุ่ม Start Server

ผู้ใช้งานระบบเรียกใช้แอปเพล็ต (applet) BLASTServer เพื่อเริ่มใช้งานระบบ โดยเริ่มจากการกดปุ่ม Start Server บนหน้าต่าง BLASTServer โดยรูปที่ 4.16 จะแสดงลำดับขั้นตอนการทำงานเมื่อผู้ใช้งานระบบทำการกดปุ่ม Start Server จะเป็นการเรียกใช้ตัวกระทำการ StartServerBT_actionperformed() จากนั้นตัวกระทำการนี้จะไปเรียกใช้ตัวกระทำการ SetListening() ในคลาส Ipdaemon เพื่อเริ่มให้บริการเครือข่ายแบบ TCP/IP และรอรับการเชื่อมต่อจาก BLASTClient เพื่อลงทะเบียนเข้าสู่ระบบ

เมื่อระบบอนุญาตให้ทำการลงทะเบียนได้แล้วผู้ใช้จึงจะสามารถทำการเชื่อมต่อ BLASTClient โดยการกดที่ปุ่ม Connect เพื่อทำการลงทะเบียนเข้าสู่ระบบ ทั้งนี้ผู้ใช้จะต้องกำหนดค่าไอพีแอดเดรส ช่องกำหนดหมายเลขของช่องทางเข้า/ออกของข้อมูล ตลอดจนถึงเส้นทางการเข้าใช้ระบบแฟ้มก่อนที่จะทำงานก่อนที่จะทำการเชื่อมต่อกับ BLASTServer

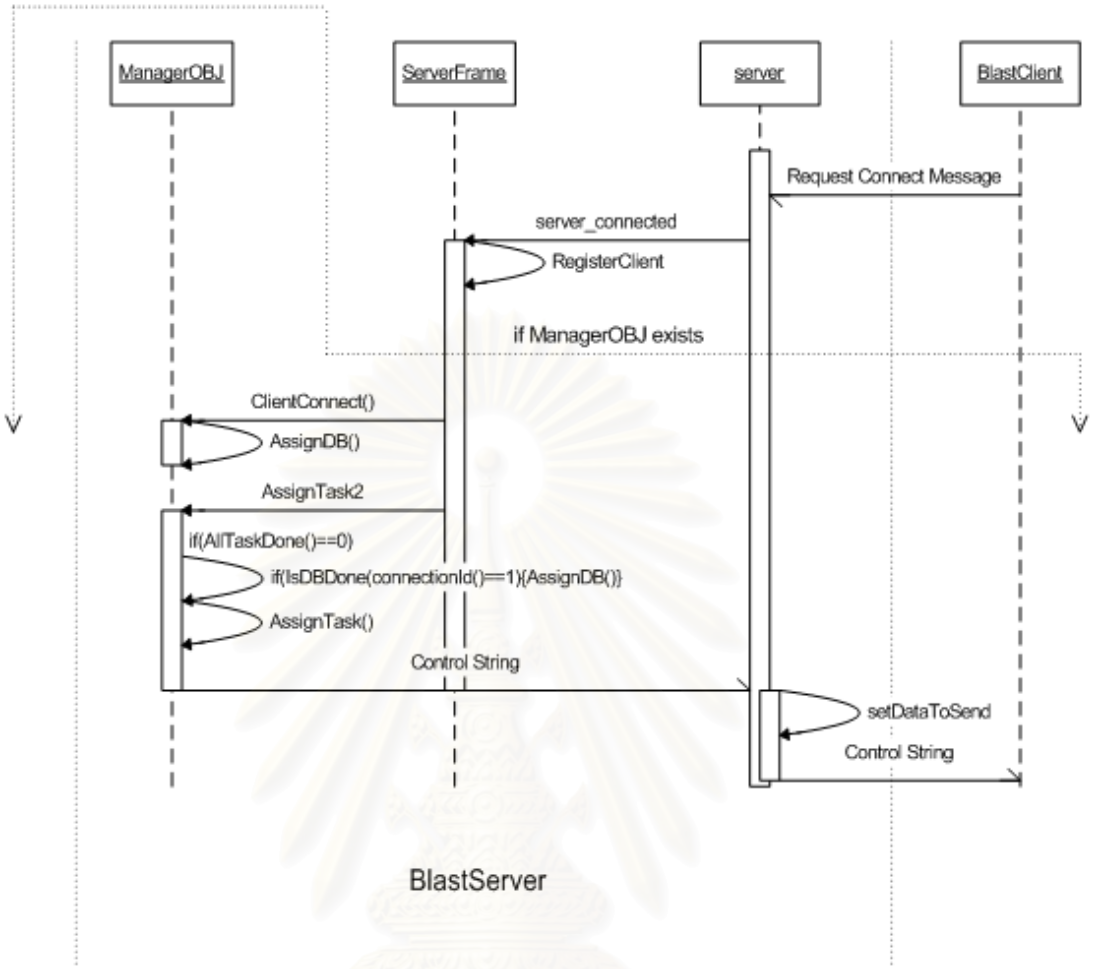


รูปที่ 4.17 แผนภาพลำดับเหตุการณ์เมื่อผู้ใช้งานระบบกดปุ่ม Connect

จากรูปที่ 4.17 เมื่อผู้ใช้ทำการกดปุ่ม Connect บนหน้าต่าง BLASTClient จะเป็นการเริ่มเรียกใช้ตัวกระทำการ StartClientBT_actionperformed ซึ่งจะทำการเรียกใช้ตัวกระทำการ setConnect บนอ๊อบเจต client เพื่อทำการเชื่อมต่อโดยการส่งคำร้องขอไปยัง BLASTServer

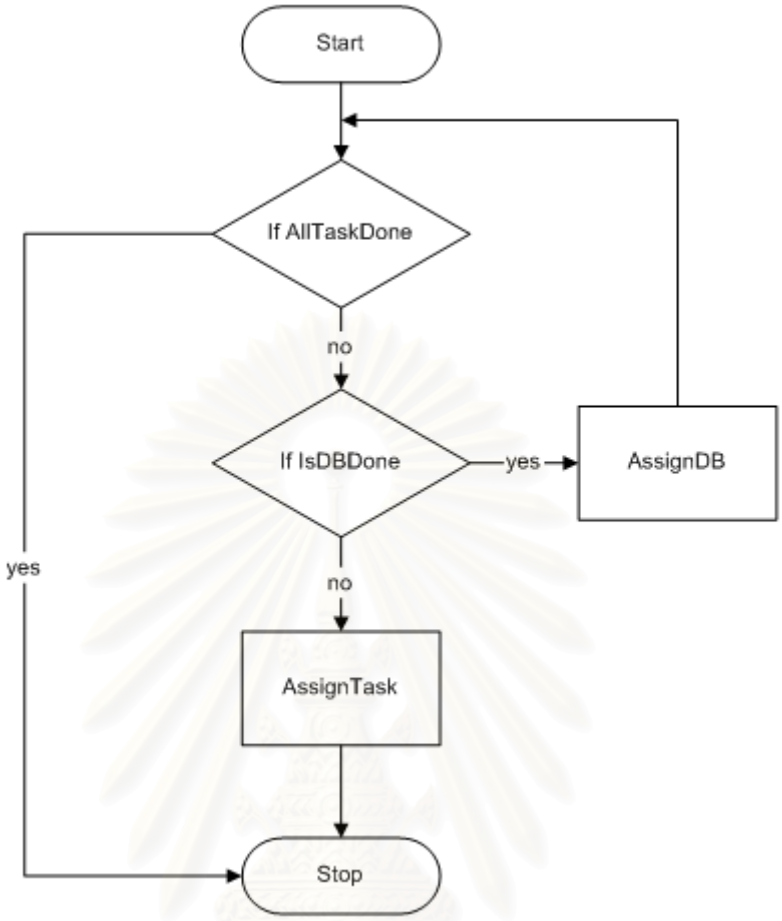
และเมื่อมีคำร้องถูกส่งเข้ามาจาก BLASTClient จะแบ่งได้เป็นสองกรณีคือในกรณีที่ผู้ใช้ได้เริ่มทำการสั่งให้ระบบเริ่มทำการเปรียบเทียบแล้วและในกรณีที่ผู้ใช้ยังไม่ได้สั่งให้ระบบทำการเปรียบเทียบ

หากผู้ใช้ยังไม่ได้สั่งให้ระบบเริ่มทำการเปรียบเทียบ BLASTServer จะทำการลงทะเบียนเบื้องต้นให้กับ BLASTClient โดยการบันทึกว่ามี BLASTClient รหัสใดบ้างเพื่อบอกให้ระบบรู้ว่าขณะนี้มี BLASTClient ใดบ้างที่สามารถส่งข้อคำถามไปประมวลผลได้ หลังจากนั้นจะนำ BLASTClient ที่ได้ลงทะเบียนเบื้องต้นไปลงทะเบียนขั้นต่อไปเพื่อรับผิดชอบฐานข้อมูลหลังจากที่ผู้ใช้ทำการสั่งให้เริ่มเปรียบเทียบลำดับดังรูปที่ 4.18



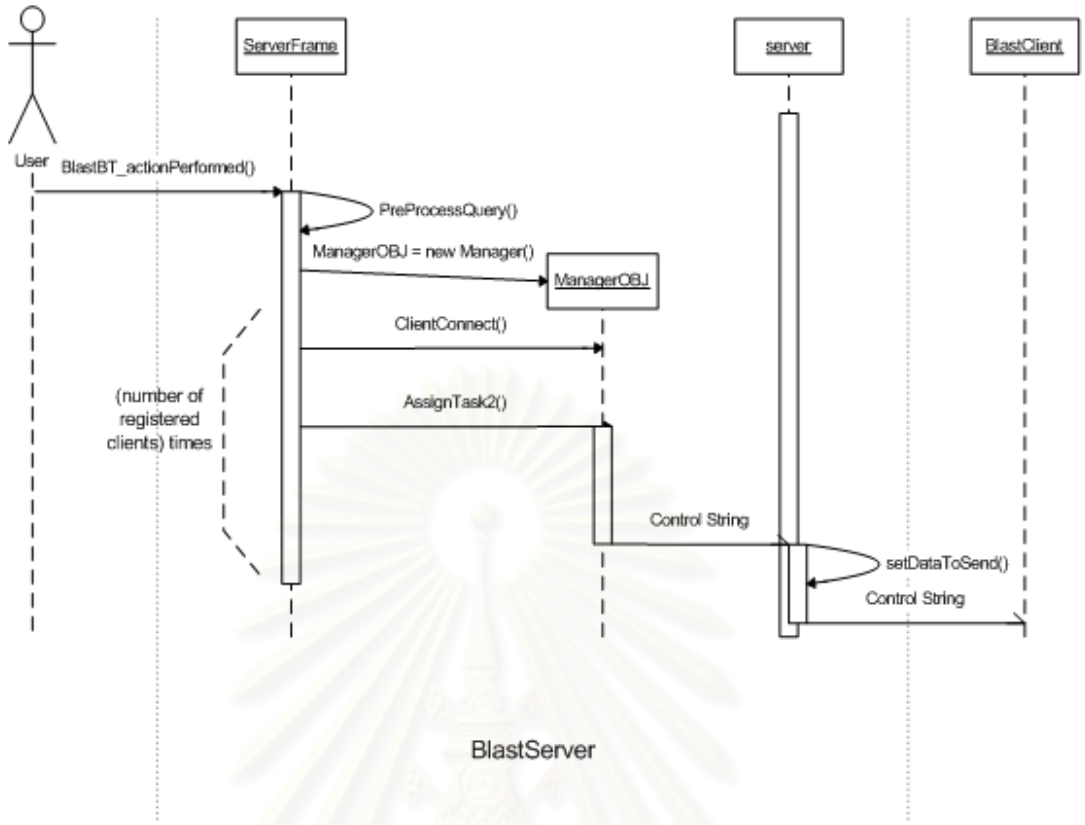
รูปที่ 4.18 แผนภาพลำดับเหตุการณ์ของ BLASTServer เมื่อมีคำร้องขอลงทะเบียนจาก BLASTClient

จากรูปที่ 4.18 เมื่อเกิดเหตุการณ์ BLASTServer ได้รับคำร้องขอลงทะเบียนจาก BLASTClient ตัวกระทำการ server_connect() ของอ๊อบเจกต์ server จะถูกเรียกใช้ หลังจากนั้นหากเป็นการลงทะเบียนครั้งแรกตัวกระทำการ server_connect() จะทำการเรียกใช้ตัวกระทำการ RegisterClient() เพื่อทำการละทะเบียนเบื้องต้นโดยจะเก็บรหัสของ BLASTClient ทั้งหมดไว้ที่ ServerFrame และหากผู้ใช้ได้สั่งให้ระบบเริ่มทำการเปรียบเทียบแล้วโดยหากมีอ๊อบเจกต์ ManagerOBJ แล้ว ServerFrame จะทำการเรียกตัวกระทำการ ClientConnect() โดย ClientConnect() จะทำหน้าที่กำหนดฐานข้อมูลรับผิดชอบให้กับ BLASTClient() จากนั้น ServerFrame() จะทำการขอสายอักขระที่ใช้ในการเรียกแฟ้มกระทำการพร้อมด้วยตัวแปรเสริมต่างๆ โดยการเรียกตัวกระทำการ AssignTask2() จากอ๊อบเจกต์ ManagerOBJ โดยการทำงานของตัวกระทำการจะเป็นดังรูปที่ 4.19 จากนั้นจะส่งผ่านสายอักขระไปยังอ๊อบเจกต์ server เพื่อส่งข้อมูลไปยัง BLASTClient โดยตัวกระทำการ setDataToSend()



รูปที่ 4.19 ผังงานของตัวกระทำการ AssignTask2()

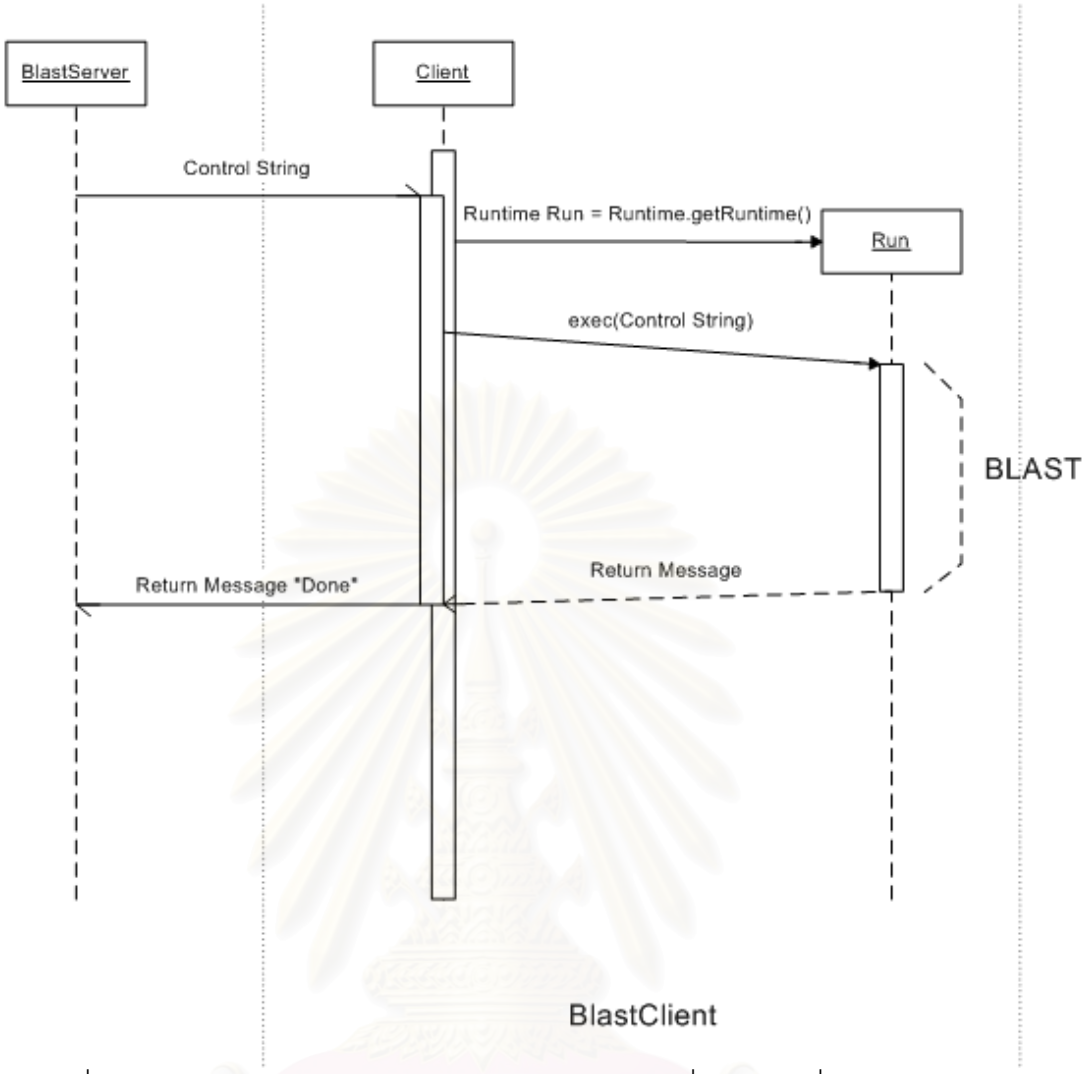
จากรูปที่ 4.19 การทำงานของตัวกระทำการ AssignTask2() จะเริ่มจากการตรวจสอบว่างานทั้งหมดถูกเปรียบเทียบเสร็จเรียบร้อยแล้วหรือยัง หากงานทั้งหมดถูกเปรียบเทียบแล้วก็จะหยุดการทำงาน หากงานทั้งหมดยังไม่เสร็จจะทำการตรวจสอบว่าฐานข้อมูลที่รับผิดชอบอยู่ถูกเปรียบเทียบเสร็จแล้วหรือยัง ถ้าฐานข้อมูลที่รับผิดชอบอยู่ได้รับการเปรียบเทียบเสร็จแล้วจะทำการเลือกฐานข้อมูลใหม่แล้วเริ่มการทำงานทั้งหมดอีกครั้ง หากยังเปรียบเทียบไม่เสร็จก็จะเลือกลำดับที่ยังไม่ได้ทำการเปรียบ เทียบมาให้โดยตัวกระทำการ AssignTask()



รูปที่ 4.20 แผนภาพลำดับเหตุการณ์ของ BLASTServer เมื่อมีเหตุการณ์ที่ผู้ใช้งานเริ่มสั่งให้ทำการเปรียบเทียบลำดับ

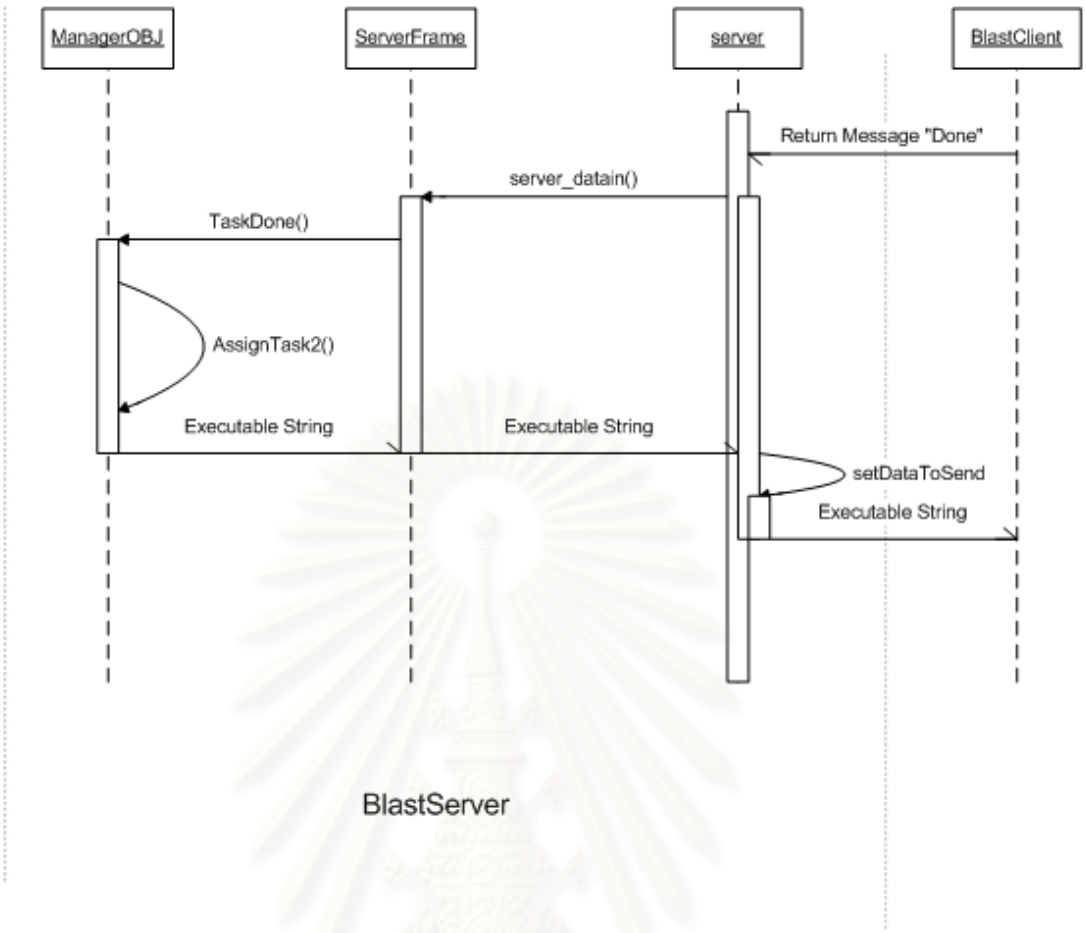
เมื่อผู้ใช้ทำการสั่งให้ระบบเริ่มทำการเปรียบเทียบโดยการกดปุ่ม BLAST! ตัวกระทำการ BLASTBT_actionPerformde() จะถูกเรียกใช้จากนั้นจะเริ่มทำการประมวลผลเบื้องต้นโดยการแบ่งข้อคำถามออกเป็นแฟ้มชั่วคราวย่อยๆ แล้วจึงสร้างอ๊อบเจกต์ ManagerOBJ ขึ้นโดยสร้างโครงสร้างข้อมูลขนาดให้เหมาะสมกับจำนวนข้อคำถามย่อยและจำนวนฐานข้อมูลดังที่ได้กล่าวถึงแล้วในหลักการกระจายข้อคำถาม หลังจากนั้นหากมี BLASTClient ที่ได้ทำการลงทะเบียนเบื้องต้นระบบจะเริ่มทำการกำหนดฐานข้อมูลและกระจายข้อคำถามให้กับ BLASTClient ทีละหน่วยเช่นเดียวกับในรูปที่ 4.18

เมื่อมีการส่งข้อมูลกลับมายัง BLASTClient อ๊อบเจกต์เมื่อได้รับข้อมูลแล้วจะทำการประมวลผลสายอักขระแล้วทำการสร้างอ๊อบเจกต์ Run ขึ้นโดยมีต้นแบบจากคลาส Runtime แล้วใช้ตัวกระทำการ exec() เพื่อทำการเปรียบเทียบลำดับด้วยแฟ้มกระทำการ blastall และจะรอจนกว่าแฟ้มกระทำการ blastall จะทำงานเสร็จจะทำการส่งสารกลับเพื่อแสดงออกบนพื้นที่แสดงสถานะของระบบแล้วจึงส่งข้อมูลกลับไปยัง BLASTServer ว่าได้ทำการเปรียบเทียบลำดับที่ให้เสร็จสิ้นแล้ว



รูปที่ 4.21 แผนภาพลำดับเหตุการณ์ของ BLASTClient เมื่อได้รับคำสั่งในการประมวลผล

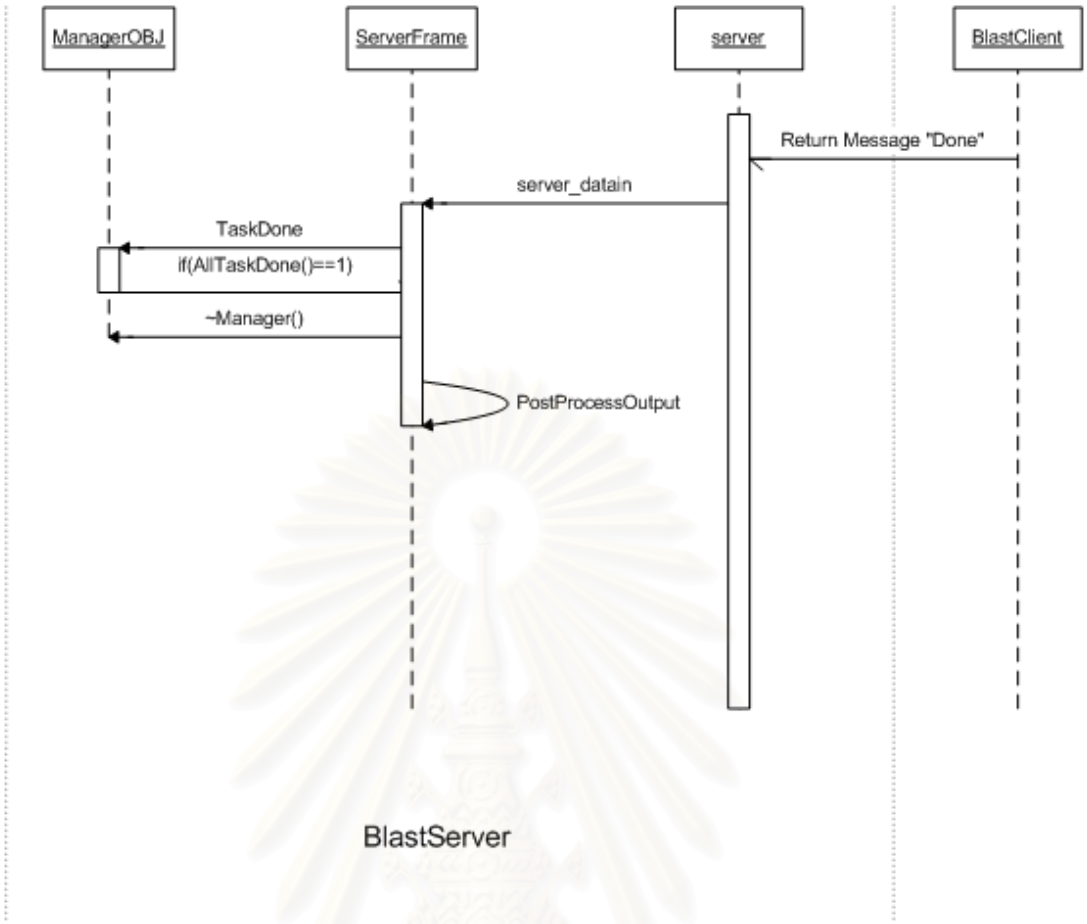
เมื่อ BLASTServer ได้รับคำตอบจาก BLASTClient ว่าทำการเปรียบเทียบลำดับเสร็จเรียบร้อยแล้วผ่านตัวกระทำการ server_datain() จะทำการเรียกตัวกระทำการ TaskDone() บนอ๊อบเจกต์ ManagerOBJ เพื่อปรับสถานะของการเปรียบเทียบลำดับแล้วจึง เรียกใช้ตัวกระทำการ AssignTask2 อีกครั้งเพื่อกำหนดแฟ้มข้อมูลนำเข้าให้ BLASTClient อีกครั้งดังรูปที่ 4.22



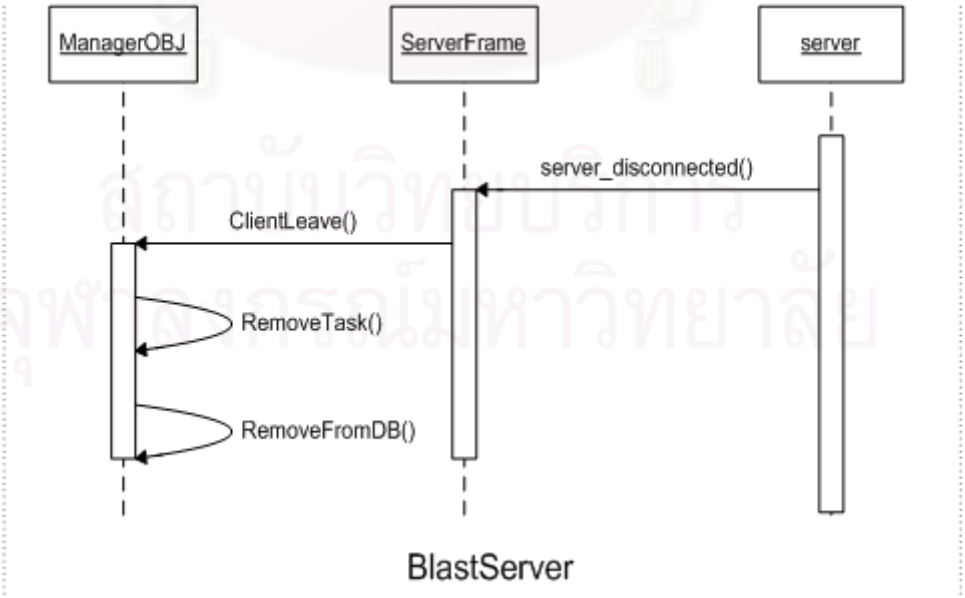
รูปที่ 4.22 แผนภาพลำดับเหตุการณ์ของ BLASTServer เมื่อได้รับคำตอบจาก BLASTClient

เมื่อลำดับทั้งหมดถูกเปรียบเทียบเสร็จสิ้นแล้วทราบจากตัวกระทำการ AllTaskDone() BLASTServer จะทำลายอ๊อบเจกต์ ManagerOBJ หลังจากนั้นก็จะทำการเชื่อมต่อผลลัพธ์เข้าด้วยกันโดยตัวกระทำการ PostProcessOutput() ดังรูปที่ 4.23 จึงเสร็จการทำงานของระบบทั้งหมด หลังจากนั้นระบบจึงพร้อมสำหรับการประมวลผลข้อคำถามชุดใหม่ต่อไป

เมื่อ BLASTServer พบว่ามี BLASTClient ใดไม่สามารถทำงานต่อหรือหลุดจากระบบโดยการตรวจสอบด้วยตัวกระทำการ isConnected() หรือว่า BLASTClient ใดต้องการถอนการลงทะเบียนจากระบบโดยการส่งคำร้องมาให้ BLASTServer อ๊อบเจกต์ server จะเรียกตัวกระทำการ server_disconnected() ซึ่งจะไปเรียกแฟ้มกระทำการ ClientLeave() บนอ๊อบเจกต์ ManagerOBJ โดยจะทำการเปลี่ยนแปลงสถานะของการเปรียบเทียบของ BLASTClient นั้นๆ ให้เป็นสถานะตั้งต้นด้วยตัวกระทำการ RemoveTask() และทำการถอนการลงทะเบียนรับผิดชอบฐานข้อมูลด้วยตัวกระทำการ RemoveFromeDB() ตามลำดับดังรูปที่ 4.24



รูปที่ 4.23 แผนภาพลำดับเหตุการณ์ของ BLASTServer เมื่อลำดับทั้งหมดถูกเปรียบเทียบเสร็จเรียบร้อยแล้ว



รูปที่ 4.24 แผนภาพลำดับเหตุการณ์ของ BLASTServer เมื่อ BLASTClient ใดๆ หลุดจากระบบ

บทที่ 5

การทดสอบระบบการประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์

5.1 การวัดประสิทธิภาพของระบบที่มีจำนวนหน่วยประมวลผลสูง

เนื่องจากระบบประมวลผลเชิงขนาดส่วนใหญ่จะมีประสิทธิภาพลดลงเมื่อจำนวนหน่วยประมวลผลมีมากขึ้นทั้งนี้สาเหตุสามารถเกิดขึ้นได้หลายประการ จุดประสงค์ของการทดลองนี้เพื่อทำให้ทราบถึงแนวโน้มของระบบเมื่อมีหน่วยประมวลผลมากขึ้น

5.1.1 วิธีการทดลอง

1. ทำการเปรียบเทียบโดยการจับเวลาการทำงานของโปรแกรมบลาสท์ตามปกติ แล้วนำไปเปรียบเทียบกับเวลาที่ได้จากการทำงานของระบบที่ได้ทำการพัฒนาขึ้นที่จำนวนหน่วยประมวลผลต่างๆ กัน
2. บันทึกผลและคำนวณอัตราการเร่งความเร็ว
3. นำผลที่ได้มาวาดกราฟและวิเคราะห์แนวโน้มของระบบเมื่อมีหน่วยประมวลผลมากขึ้น

5.1.2 สภาวะที่ใช้ในการทดลอง

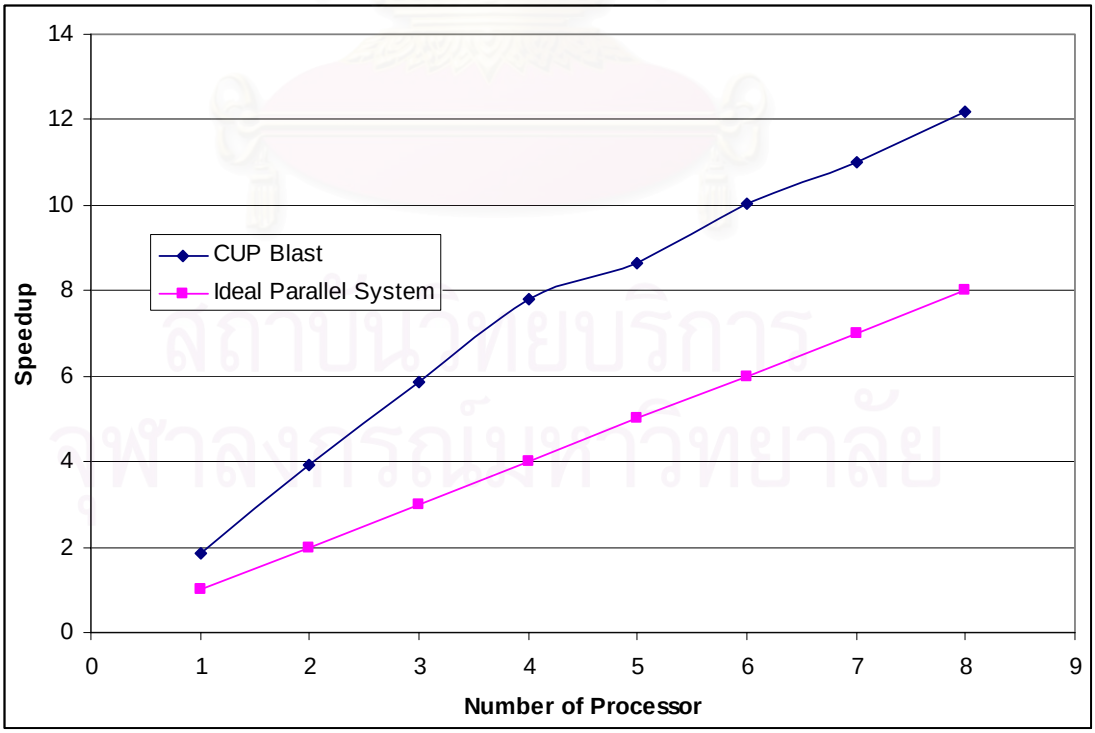
1. ตัวประมวลผลกลาง Intel Pentium III 533 เมกะเฮิร์ทซ์
2. หน่วยความจำขนาด 128 เมกะไบต์
3. ระบบปฏิบัติการวินโดวส์ มิลเลนเนียม เอดิชัน
4. ระบบเครื่องบริการแฟ้ม คือ ระบบเครือข่ายของไมโครซอฟท์
5. ฐานข้อมูล NR ขนาด 234 เมกะไบต์ แบ่งออกเป็น 2 ส่วนคือ
   I. NR1 ขนาด 118 เมกะไบต์ และ
   II. NR2 ขนาด 116 เมกะไบต์
6. ขั้นตอนวิธีที่ใช้ในการเปรียบเทียบคือ blastx
7. ข้อคำถามจำนวน 1000 ลำดับ

5.1.3 ผลการทดลอง

เวลาที่ใช้ในการทดสอบจากการทำงานตามปกติใช้เวลาทั้งสิ้น 804 นาที ทั้งนี้เวลาที่ใช้ในการทดสอบการทำงานของประสิทธิภาพของระบบจะเห็นได้จากตารางที่ 5.1 และรูปที่ 5.1

ตารางที่ 5.1 ผลการทดสอบประสิทธิภาพของระบบในการทดลองวัดประสิทธิภาพของระบบที่มีจำนวนหน่วยประมวลผลสูง

จำนวน BLASTClient	เวลาในการประมวลผลเบื้องต้น (นาที)	เวลาที่ใช้ในการเปรียบเทียบ (นาที)	เวลาที่ใช้ในการรวมผลลัพธ์ (นาที)	เวลาที่ใช้ทั้งสิ้น (นาที)	อัตราการเร่งความเร็ว (เท่า)
1	5	421	10	436	1.84
2	5	191	10	206	3.9
3	5	122	10	137	5.8
4	5	88	10	103	7.8
5	5	78	10	93	8.6
6	5	65	10	80	10
7	5	58	10	73	11
8	5	51	10	66	12



รูปที่ 5.1 ผลการวัดประสิทธิภาพของระบบในการทดลองวัดประสิทธิภาพของระบบที่มีจำนวนหน่วยประมวลผลสูง

5.1.3 การวิเคราะห์ผลการทดลอง

จากการทดลองที่ได้จะเห็นได้ว่า อัตราการเร่งความเร็วของระบบสูงเกินกว่า อัตราการเร่งความเร็วของระบบประมวลผลเชิงขนานในอุดมคติ ทั้งนี้ปรากฏการณ์นี้เรียกว่าปรากฏการณ์อัตราการเร่งความเร็วแบบซูเปอร์ลีเนียร์ (superlinear speedup) เป็นปรากฏการณ์ที่อัตราการเร่งความเร็วของระบบประมวลผลที่จำนวนหน่วยประมวลผลใดๆ มีค่าสูงกว่าจำนวนของหน่วยประมวลผลที่ทำการวัด ในกรณีนี้ความเร็วที่สูงขึ้นเนื่องมาจาก การกำจัดการสับเปลี่ยนค่าข้อมูลระหว่างหน่วยความจำหลักและจานบันทึกข้อมูล ดังที่กล่าวไว้แล้วในบทที่ 3

การวิเคราะห์แนวโน้มที่ได้เมื่อเพิ่มจำนวนหน่วยประมวลผลให้กับระบบจะพิจารณาจากค่าความชันของกราฟ โดยค่าความชันของกราฟในช่วงต้น (จำนวนหน่วยประมวลผลตั้งแต่ 1 ถึง 4 หน่วย) มีค่าเท่ากับ 2 ในขณะที่ค่าความชันของกราฟในช่วงปลาย (จำนวนหน่วยประมวลผลตั้งแต่ 6 ถึง 8 หน่วย) มีค่าเหลือเพียงแค่ 1 ซึ่งมีค่าลดลงเนื่องมาจากอัตราการเร่งความเร็วของระบบนั้นลดลง ในขณะที่จำนวนของหน่วยประมวลผลคงที่ สาเหตุหลักก็คือ ถึงแม้ว่าเวลาที่ใช้ในการเปรียบเทียบลดลง แต่ว่าเวลาที่ใช้ในการประมวลผลเบื้องต้นรวมถึงเวลาที่ใช้ในการประมวลผลในช่วงปลายของระบบคงที่ เนื่องมาจากการประมวลผลเบื้องต้นและช่วงปลายของระบบเป็นการทำงานเป็นขั้นตอนโดยที่ไม่ได้ถูกแบ่งออกไปประมวลผลตามหน่วยต่างๆ และสาเหตุรองก็คือ ภาระการทำงานของ BLASTServer และ SharedStorage มีมากขึ้นเมื่อจำนวนของหน่วยประมวลผลมากขึ้นตามลำดับรวมถึง ภาระของระบบเครือข่ายที่ต้องทำงานหนักขึ้นเมื่อมีการส่งข้อมูลมากขึ้นอีกด้วย

5.2 การวัดประสิทธิภาพของระบบเมื่อขนาดของฐานข้อมูลรวมใหญ่กว่าขนาดของหน่วยความจำรวมมาก

จากสมมุติฐานเบื้องต้นที่ว่า ระบบจะทำงานช้าลงเมื่อเกิดอัตราการสับเปลี่ยนค่าข้อมูลระหว่างหน่วยความจำและจานบันทึกสูง การเปรียบเทียบข้อคำถามกับฐานข้อมูลขนาดยิ่งใหญ่ขึ้นเท่าไรยิ่งทำให้ใช้เวลาในการเปรียบเทียบนานมากขึ้น การทดลองนี้มีจุดประสงค์เพื่อศึกษาถึงผลที่ได้รับจากการประมวลผลเชิงขนาน ซึ่งกำจัดปัญหาที่เกิดขึ้นจากการสับเปลี่ยนค่าข้อมูลระหว่างหน่วยความจำหลักและจานบันทึกข้อมูล

5.2.1 **วิธีการทดลอง**

1. ทำการเปรียบเทียบโดยการจับเวลาทำงานของโปรแกรมบลาสท์ตามปกติ แล้วนำไปเปรียบเทียบกับเวลาที่ได้จากการทำงานของระบบที่ได้ทำการพัฒนาขึ้นที่จำนวนหน่วยประมวลผลต่างๆ กัน
2. บันทึกผลและคำนวณอัตราการเร่งความเร็ว
3. นำผลที่ได้มาวาดกราฟและวิเคราะห์แนวโน้มของระบบ

5.2.2 **สภาวะที่ใช้ในการทดลอง**

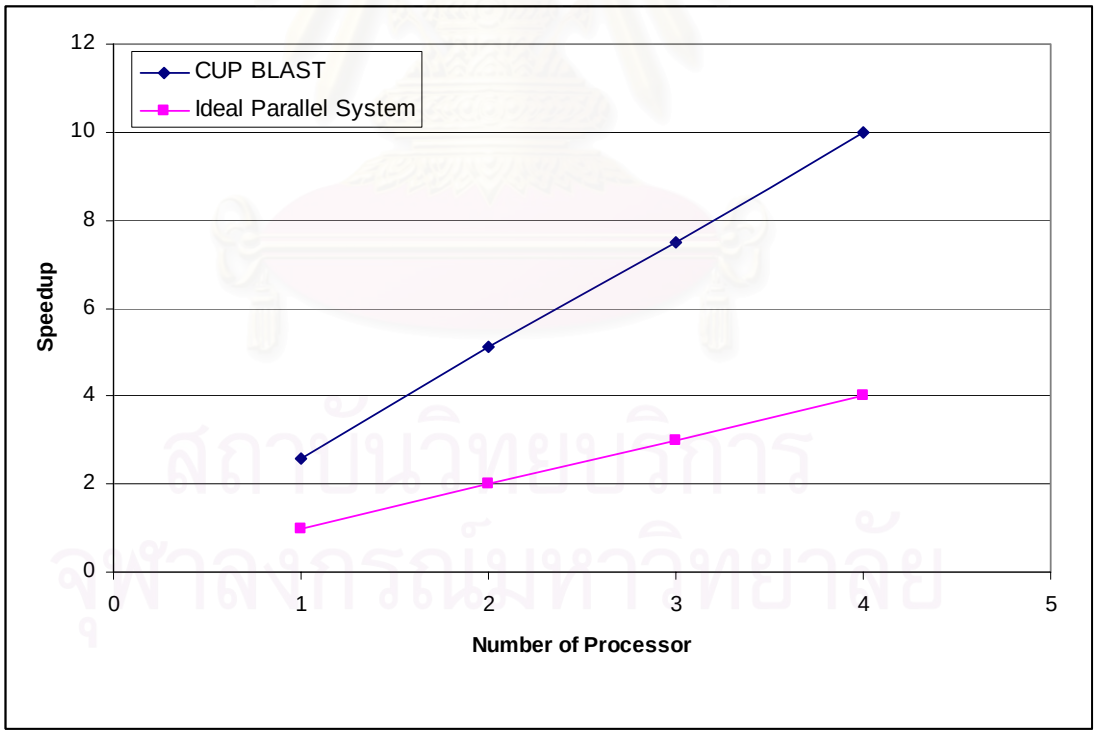
1. เครื่อง Intel Pentium II 500E MHz
2. หน่วยความจำขนาด 256 เมกะไบต์
3. ระบบปฏิบัติการ ซูซี ลินุกซ์ 8.0
4. ระบบเครื่องบริการแฟ้ม คือ ระบบปฏิบัติการแซมบ้า
5. ฐานข้อมูล NT ขนาด 2392 เมกะไบต์ แบ่งออกเป็น 12 ส่วนคือ
   I. NT1 ขนาด 209 เมกะไบต์
   II. NT2 ขนาด 209 เมกะไบต์
   III. NT3 ขนาด 209 เมกะไบต์
   IV. NT4 ขนาด 209 เมกะไบต์
   V. NT5 ขนาด 209 เมกะไบต์
   VI. NT6 ขนาด 209 เมกะไบต์
   VII. NT7 ขนาด 209 เมกะไบต์
   VIII. NT8 ขนาด 209 เมกะไบต์
   IX. NT9 ขนาด 209 เมกะไบต์
   X. NT10 ขนาด 209 เมกะไบต์
   XI. NT11 ขนาด 209 เมกะไบต์ และ
   XII. NT12 ขนาด 85 เมกะไบต์
6. ขั้นตอนวิธีที่ใช้ในการเปรียบเทียบคือ blastn
7. ข้อคำถามจำนวน 200 ลำดับ

5.2.3 ผลการทดลอง

เวลาที่ใช้ในการทดสอบโดยการทำงานตามปกติใช้เวลาทั้งสิ้น 1250 นาที และเวลาที่ได้จากการพัฒนาระบบที่ถูกพัฒนาขึ้นจะเห็นได้จากตารางที่ 5.2 และกราฟในรูปที่ 5.2

ตารางที่ 5.2 ผลการทดสอบประสิทธิภาพของระบบในการทดลองวัดประสิทธิภาพของระบบเมื่อขนาดของฐานข้อมูลรวมใหญ่กว่าขนาดของหน่วย ความจำรวมมาก

จำนวน BLASTClient	เวลาในการประมวลผลเบื้องต้น (นาที)	เวลาที่ใช้ในการเปรียบเทียบ (นาที)	เวลาที่ใช้ในการรวมผลลัพธ์ (นาที)	เวลาที่ใช้ทั้งสิ้น (นาที)	อัตราการเร่งความเร็ว (เท่า)
1	2	481	5	488	2.6
2	2	238	5	245	5.1
3	2	159	5	166	7.5
4	2	118	5	125	10



รูปที่ 5.2 ผลการวัดประสิทธิภาพของระบบในการทดลองวัดประสิทธิภาพของระบบเมื่อขนาดของฐานข้อมูลรวมใหญ่กว่าขนาดของหน่วย ความจำรวมมาก

5.2.4 การวิเคราะห์ผลการทดลอง

จากรูปที่ 5.2 จะเห็นได้ว่าอัตราการเร่งความเร็วของระบบมีค่าสูงขึ้นในระดับ ซูเปอร์ลีเนียร์ เช่นเดียวกับผลการทดลองที่ได้ในการทดลองที่ 5.1 แต่ค่าความชันเฉลี่ยที่ได้มีค่าสูงถึง 2.46 ซึ่งสูงกว่าค่าความชันในการทดลองที่ 5.1 มาก ทั้งนี้เนื่องมาจาก การทดลองที่ 5.2 สามารถกำจัดเวลาที่ใช้ในการสับเปลี่ยนข้อมูลได้มากกว่า โดยจะสังเกตได้จากอัตราส่วนของขนาดของฐาน ข้อมูลที่ใช้ต่อขนาดของหน่วยความจำหลักของแต่ละหน่วยประมวลผลย่อยของระบบ ในที่นี้อัตราส่วนดังกล่าวของการทดลองที่ 5.2 มีค่าเท่ากับ 9.3 เท่า ในขณะที่อัตราส่วนของการทดลองที่ 5.1 มีค่าเพียง 1.8 เท่า เพราะฉะนั้นระบบที่พัฒนาขึ้นเหมาะสำหรับระบบที่มีปัญหาหน่วยความจำขนาดจำกัด โดยประโยชน์ที่ได้รับซึ่งก็คืออัตราการเร่งความเร็วของระบบจะยิ่งสูงขึ้น เมื่ออัตราส่วนของฐานข้อมูลต่อขนาดของหน่วยความจำมีค่าสูงขึ้นตามลำดับ

5.3 การวัดประสิทธิภาพของระบบเทียบกับระบบประมวลผลเชิงขนานแบบข้อคำถามขนาน

จุดประสงค์ของการทดลองเพื่อที่จะเปรียบเทียบประสิทธิภาพของระบบที่ได้พัฒนาขึ้นเทียบกับระบบที่พัฒนาขึ้นและใช้งานกันอย่างแพร่หลายอย่างกับระบบประมวลผลเชิงขนานแบบข้อคำถามขนาน

5.3.1 วิธีการทดลอง

1. ทำการเปรียบเทียบโดยการจับเวลาทำงานของระบบที่ได้ทำการพัฒนาโดยไม่ทำการตัดแบ่งฐานข้อมูลออกเป็นส่วนย่อยๆ เทียบกับการทำงานของระบบที่ถูกตัดแบ่งฐานข้อมูลในผลการทดลองที่ 5.2
2. บันทึกผลและคำนวณอัตราการเร่งความเร็วของระบบทั้งสอง
3. นำผลที่ได้มาวาดกราฟและวิเคราะห์เปรียบเทียบกับการทดลองที่ 5.2

5.3.2 สภาวะที่ใช้ในการทดลอง

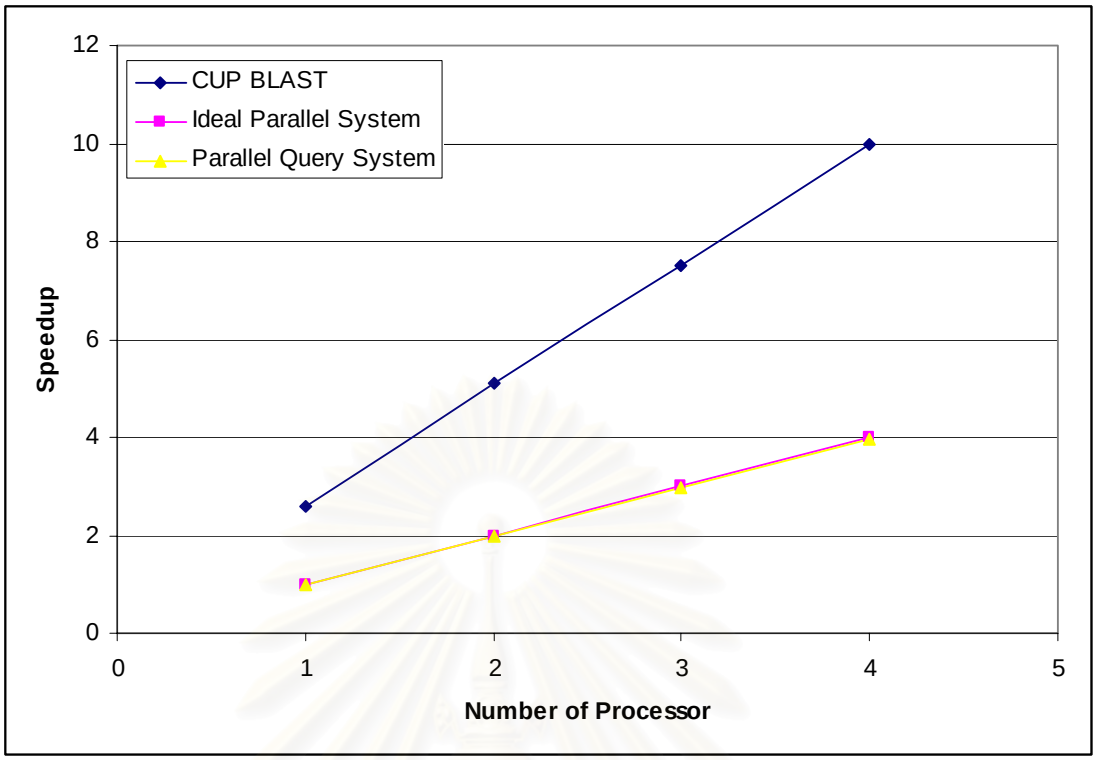
1. เครื่อง Intel Pentium II 500E MHz
2. หน่วยความจำขนาด 256 เมกะไบต์
3. ระบบปฏิบัติการ ซูซี่ ลินุกซ์ 8.0
4. ระบบเครื่องบริการแฟ้ม คือ ระบบปฏิบัติการแซมบ้า
5. ฐานข้อมูล NT ขนาด 2392 เมกะไบต์
6. ขั้นตอนวิธีที่ใช้ในการเปรียบเทียบคือ blastn
7. ข้อคำถามจำนวน 200 ลำดับ

5.3.3 **ผลการทดลอง**

เวลาที่ใช้ในการทดลองระบบการประมวลผลแบบข้อคำถามขนานซึ่งได้จากระบบที่พัฒนาขึ้นโดยไม่ทำการตัดแบ่งฐานข้อมูล จะเห็นได้จากตารางที่ 5.3 และกราฟในรูปที่ 5.3

ตารางที่ 5.3 ผลการทดสอบประสิทธิภาพของระบบในการทดลองวัดประสิทธิภาพของระบบเทียบกับระบบประมวลผลเชิงขนานแบบข้อคำถามขนาน

จำนวน BLASTClient	เวลาในการประมวลผลเบื้องต้น (นาที)	เวลาที่ใช้ในการเปรียบเทียบ (นาที)	เวลาที่ใช้ในการรวมผลลัพธ์ (นาที)	เวลาที่ใช้ทั้งสิ้น (นาที)	อัตราการเร่งความเร็ว (เท่า)
1	2	1256	1	1259	0.99
2	2	628	1	631	1.98
3	2	419	1	422	2.96
4	2	314	1	317	3.94



รูปที่ 5.3 ผลการเปรียบเทียบประสิทธิภาพระหว่างระบบการประมวลผลเชิงขนานที่พัฒนาขึ้น และระบบการประมวลผลเชิงขนานแบบข้อคำถามขนาน

5.3.3 การวิเคราะห์ผลการทดลอง

จากรูปที่ 5.3 จะเห็นว่าประสิทธิภาพของระบบที่พัฒนาขึ้นสูงกว่าระบบประมวลผลเชิงขนานแบบข้อคำถามขนานอย่างเห็นได้ชัดโดย ทั้งนี้ระบบแบบข้อคำถามขนานมีความสามารถใกล้เคียงกับระบบประมวลผลเชิงขนานในอุดมคติ หากแต่ว่าระบบการประมวลผลแบบข้อคำถามขนานมีความได้เปรียบกว่าระบบที่ได้ทำการพัฒนาขึ้น เนื่องจากแฟ้มผลลัพธ์มีจำนวนน้อยกว่าและสามารถเชื่อมต่อได้ทันที

5.4 การวัดประสิทธิภาพของระบบที่จำนวนลำดับในข้อคำถามต่าง ๆกัน

เนื่องจากระบบที่ทำการพัฒนาถูกออกแบบให้รองรับกับฐานข้อมูลขนาดใหญ่ เนื่องมาจากการสับเปลี่ยนค่าข้อมูลระหว่างหน่วยความจำในการเปรียบเทียบลำดับจำนวนมากๆ ดังนั้นการประมวลผลด้วยจำนวนลำดับที่ต่างกันย่อมจะมีผลต่ออัตราการเร่งความเร็วของระบบดังจะสังเกตได้จากการทดลองนี้

5.4.1 วิธีการทดลอง

1. ทำการเปรียบเทียบโดยการจับเวลาทำงานของระบบที่ได้ทำการพัฒนาโดยเลือกใช้ลำดับในข้อคำถามจำนวนต่างๆ กัน ต่อจากการทดลองที่ 5.2
2. บันทึกผลและคำนวณอัตราการเร่งความเร็วของระบบที่จำนวนข้อคำถามต่างๆ
3. นำผลที่ได้มาวาดกราฟและวิเคราะห์เปรียบเทียบกันกับผลการทดลองที่ 5.2

5.4.2 สภาวะที่ใช้ในการทดสอบ

1. เครื่อง Intel Pentium II 500E MHz
2. หน่วยความจำขนาด 128 เมกะไบต์
3. ระบบปฏิบัติการ ซูซี่ ลินุกซ์ รุ่น 8.0
4. ฐานข้อมูล NT ขนาด 2392 เมกะไบต์ แบ่งออกเป็น 12 ส่วนคือ
    I. NT1 ขนาด 209 เมกะไบต์
    II. NT2 ขนาด 209 เมกะไบต์
    III. NT3 ขนาด 209 เมกะไบต์
    IV. NT4 ขนาด 209 เมกะไบต์
    V. NT5 ขนาด 209 เมกะไบต์
    VI. NT6 ขนาด 209 เมกะไบต์
    VII. NT7 ขนาด 209 เมกะไบต์
    VIII. NT8 ขนาด 209 เมกะไบต์
    IX. NT9 ขนาด 209 เมกะไบต์
    X. NT10 ขนาด 209 เมกะไบต์
    XI. NT11 ขนาด 209 เมกะไบต์ และ
    XII. NT12 ขนาด 85 เมกะไบต์
5. ขั้นตอนวิธีที่ใช้ในการเปรียบเทียบคือ blastx

5.4.3 ผลการทดลอง

ข้อคำถามที่ใช้ประกอบด้วยลำดับทั้งสิ้น 1 ลำดับ เวลาที่ใช้ในการทำงานตามปกติคือ 375 วินาที ผลที่ได้จากการทดลองระบบเป็นดังนี้

ตารางที่ 5.4 ผลการทดสอบประสิทธิภาพของระบบในการทดลองวัดประสิทธิภาพของระบบ โดยใช้ลำดับจำนวน 1 ลำดับในข้อคำถาม

จำนวน BLASTClient	เวลาในการประมวลผลเบื้องต้น (วินาที)	เวลาที่ใช้ในการเปรียบเทียบ (วินาที)	เวลาที่ใช้ในการรวมผลลัพธ์ (วินาที)	เวลาที่ใช้ทั้งสิ้น (วินาที)	อัตราการเร่งความเร็ว (เท่า)
1	0	492	3	495	0.76
2	0	276	3	279	1.34
3	0	168	3	171	2.19
4	0	114	3	117	3.21

ข้อคำถามที่ใช้ประกอบด้วยลำดับทั้งสิ้น 2 ลำดับ เวลาที่ใช้ในการทำงานตามปกติคือ 742 วินาที ผลที่ได้จากการทดลองระบบเป็นดังนี้

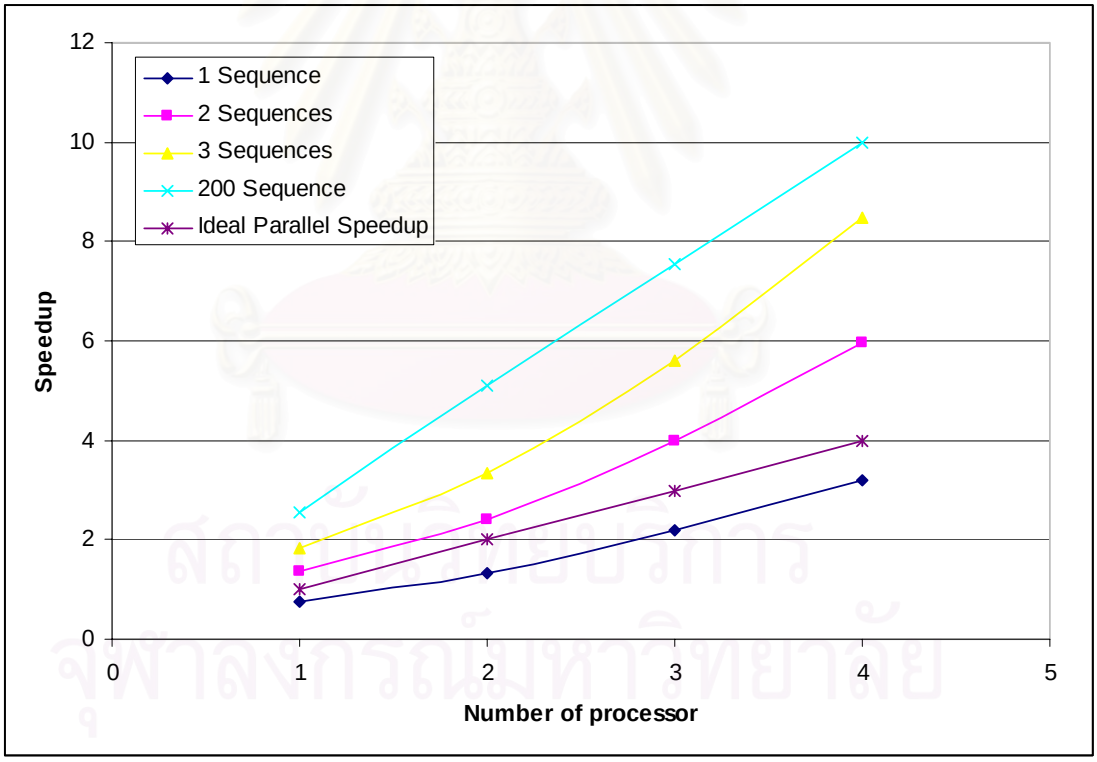
ตารางที่ 5.5 ผลการทดสอบประสิทธิภาพของระบบในการทดลองวัดประสิทธิภาพของระบบ โดยใช้ลำดับจำนวน 2 ลำดับในข้อคำถาม

จำนวน BLASTClient	เวลาในการประมวลผลเบื้องต้น (วินาที)	เวลาที่ใช้ในการเปรียบเทียบ (วินาที)	เวลาที่ใช้ในการรวมผลลัพธ์ (วินาที)	เวลาที่ใช้ทั้งสิ้น (วินาที)	อัตราการเร่งความเร็ว (เท่า)
1	1	546	3	550	1.34
2	1	303	3	307	2.41
3	1	181	3	185	4
4	1	121	3	125	5.94

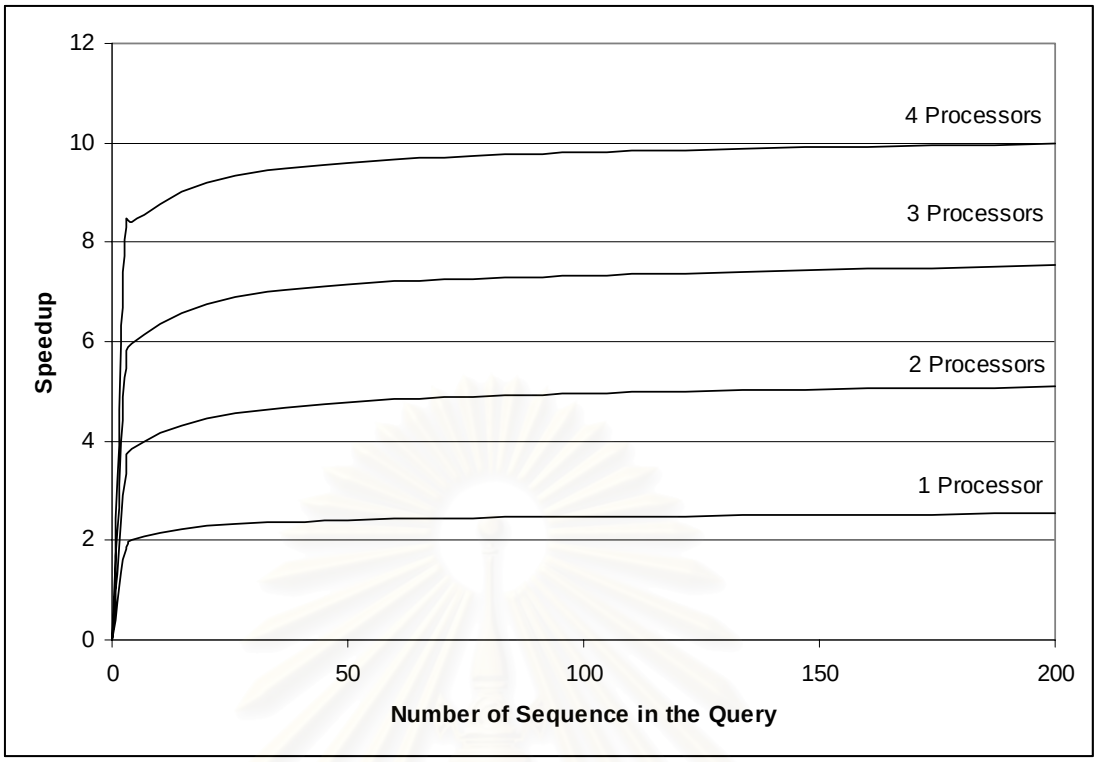
ข้อคำถามที่ใช้ประกอบด้วยลำดับทั้งสิ้น 3 ลำดับ เวลาที่ใช้ในการทำงานตามปกติคือ 1113 วินาที ผลที่ได้จากการทดลองระบบเป็นดังนี้

ตารางที่ 5.6 ผลการทดสอบประสิทธิภาพของระบบในการทดลองวัดประสิทธิภาพของระบบ โดยใช้ลำดับจำนวน 3 ลำดับในข้อคำถาม

จำนวน BLASTClient	เวลาในการประมวลผลเบื้องต้น (วินาที)	เวลาที่ใช้ในการเปรียบเทียบ (วินาที)	เวลาที่ใช้ในการรวมผลลัพธ์ (วินาที)	เวลาที่ใช้ทั้งสิ้น (วินาที)	อัตราการเร่งความเร็ว (เท่า)
1	1	546	3	550	1.34
2	1	303	3	307	2.41
3	1	181	3	185	4
4	1	121	3	125	5.94



รูปที่ 5.4 ผลการเปรียบเทียบประสิทธิภาพของระบบที่จำนวนลำดับในข้อคำถามต่างๆ กัน



รูปที่ 5.5 ผลการวัดประสิทธิภาพเปรียบเทียบของระบบที่จำนวนลำดับในข้อคำถามจำนวนต่างๆ กันเปรียบเทียบตามจำนวนของหน่วยประมวลผล

5.4.4 การวิเคราะห์ผลการทดลอง

จากการทดลองที่ได้จะเห็นว่าประสิทธิภาพของระบบเมื่อใช้ลำดับในข้อคำถามเพียงลำดับเดียวทำงานได้ช้ากว่าการทำงานตามปกติ เนื่องจากนอกจากจะต้องทำการสับเปลี่ยนค่าข้อมูลเมื่อฐานข้อมูลเปลี่ยนไปแล้ว ยังต้องเสียเวลาในการเชื่อมต่อผลลัพธ์อีกด้วย ระบบจึงไม่เหมาะสมที่จะทำการประมวลผลข้อคำถามที่มีลำดับเพียงลำดับเดียว และเมื่อพิจารณาการเปรียบเทียบข้อคำถามที่มีลำดับตั้งแต่ 2 ลำดับขึ้นไปจะพบว่าการประมวลผลทำได้เร็วขึ้นและเกินขีดจำกัดของระบบประมวลผลเชิงขนานในอุดมคติเข้าสู่ขอบเขตของซูเปอร์ลีเนียร์ โดยจะเห็นได้ว่าอัตราการเร่งความเร็วจะยิ่งสูงขึ้นมากเมื่อจำนวนลำดับในข้อคำถามเพิ่มขึ้น ทั้งนี้ความคุ้มของระบบเกิดจากการที่ฐานข้อมูลที่ต้องการเปรียบเทียบฝังอยู่ในหน่วยความจำได้นานที่สุด กล่าวคือจะทำการเปรียบเทียบจนกระทั่งไม่มีความจำเป็นต้องใช้ฐานข้อมูลนั้นๆ อีกจึงเริ่มทำการเปรียบเทียบฐานข้อมูลใหม่

5.5 การวัดประสิทธิภาพของระบบในฐานงานอื่นๆ

เนื่องจากระบบถูกออกแบบมาโดยใช้เทคโนโลยีจาวาทำให้สามารถทำงานได้บนฐานงานอื่นๆ ที่รองรับการทำงานของเครื่องเสมือนจาวา (Java virtual machine) โดยเฉพาะระบบปฏิบัติการโซลาร์รีสซึ่งถูกพัฒนาโดยบริษัท ซันไมโครซิสเท็ม เช่นเดียวกับภาษาจาวา การทดลองนี้เพียงเพื่อทดสอบความสามารถในการทำงานในฐานงานอื่นๆ นอกจากระบบปฏิบัติการวินโดว์และลินุกซ์

5.5.1 วิธีการทดลอง

1. ทำการเปรียบเทียบโดยการจับเวลาทำงานของระบบที่จำนวนหน่วยประมวลผลต่างๆ กัน
2. บันทึกผลและคำนวณอัตราการเร่งความเร็วของระบบ
3. นำผลที่ได้มาวาดกราฟและวิเคราะห์ผลการทดลอง

5.5.2 สภาวะที่ใช้ในการทดสอบ

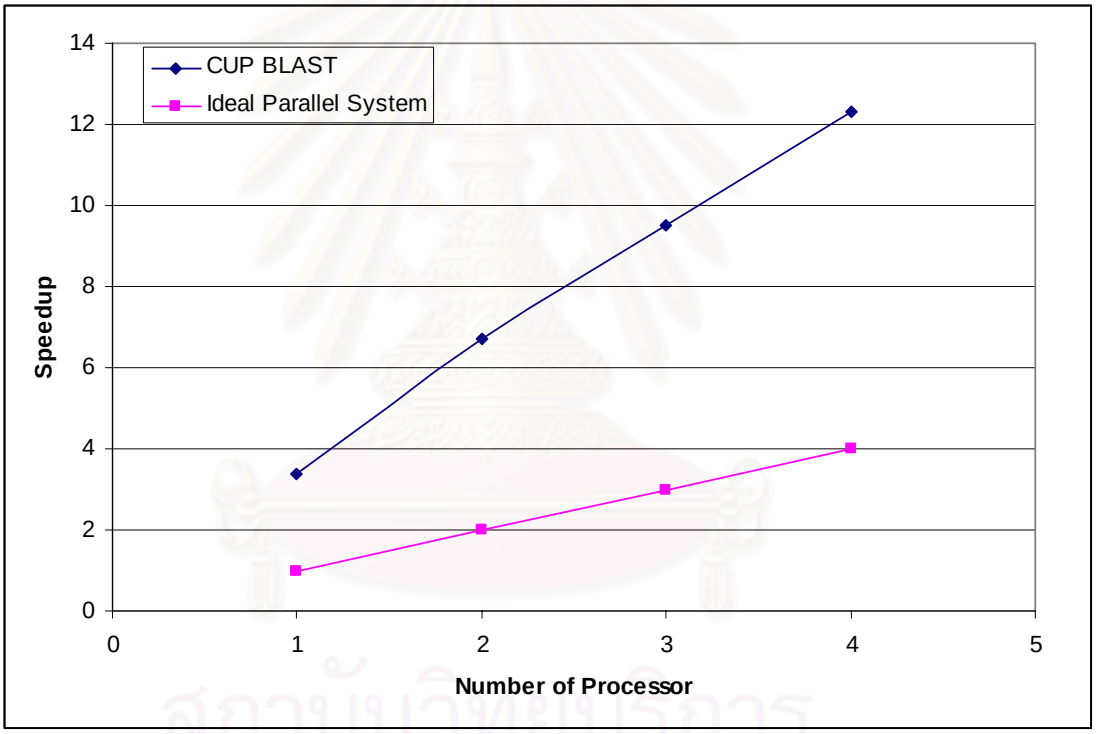
1. เครื่อง Intel Pentium II 233 เมกะเฮิร์ทซ์
2. หน่วยความจำขนาด 64 เมกะไบต์
3. ระบบปฏิบัติการ โซลาร์รีส รุ่นที่ 5.8 สำหรับสถาปัตยกรรมแบบ x86
4. ฐานข้อมูล NR ขนาด 234 เมกะไบต์ แบ่งออกเป็น 2 ส่วนคือ
   I. NR1 ขนาด 59 เมกะไบต์
   II. NR2 ขนาด 58 เมกะไบต์
   III. NR3 ขนาด 58 เมกะไบต์ และ
   IV. NR4 ขนาด 59 เมกะไบต์
5. ขั้นตอนวิธีที่ใช้ในการเปรียบเทียบคือ blastx
6. ข้อคำถามจำนวน 100 ลำดับ

5.5.3 ผลการทดลอง

เวลาที่ใช้ในการทดสอบโดยการทำงานตามปกติใช้เวลาทั้งสิ้น 1778 นาที และเวลาที่ได้จากการพัฒนาระบบที่ถูกพัฒนาขึ้นจะเห็นได้จากตารางที่ 5.2 และกราฟในรูปที่ 5.2

ตารางที่ 5.7 ผลการทดสอบประสิทธิภาพของระบบบนฐานงานระบบปฏิบัติการ โซลาริส สำหรับสถาปัตยกรรมแบบ x86

จำนวน BLASTClient	เวลาในการประมวลผลเบื้องต้น (นาที)	เวลาที่ใช้ในการเปรียบเทียบ (นาที)	เวลาที่ใช้ในการรวมผลลัพธ์ (นาที)	เวลาที่ใช้ทั้งสิ้น (นาที)	อัตราการเร่งความเร็ว (เท่า)
1	4	508	11	524	3.4
2	4	248	11	265	6.7
3	4	169	11	187	9.5
4	4	125	11	144	12.3



รูปที่ 5.6 ผลการทดสอบประสิทธิภาพของระบบบนฐานงานระบบปฏิบัติการ โซลาริส สำหรับสถาปัตยกรรมแบบ x86

5.5.4 การวิเคราะห์ผลการทดลอง

จากการทดลองที่ได้จะเห็นว่าประสิทธิภาพของระบบเมื่อในฐานงานอื่นๆ สามารถเพิ่มประสิทธิภาพของอัตราการเร่งความเร็วของระบบได้ถึงขอบเขตของซูเปอร์ลีเนียร์เช่นกัน ทั้งนี้เนื่องมาจากการตัดแบ่งฐานข้อมูลให้มีขนาดเล็กเพียงพอที่จะฝังลงในหน่วยความจำหลักในขณะประมวลผลได้นั่นเอง

บทที่ 6

สรุปผลการวิจัย และข้อเสนอแนะ

6.1 สรุปผลการวิจัย

งานวิจัยนี้ได้ทำการออกแบบและพัฒนาระบบประมวลผลเชิงขนานสำหรับโปรแกรมบลาสท์ เพื่อแก้ปัญหาการทำงานที่กินเวลานานของโปรแกรมบลาสท์ในการเปรียบเทียบข้อคำถามจำนวนมาก กับฐานข้อมูลพันธุวิศวกรรมที่มีขนาดมหาศาล โดยใช้แนวคิดในการกำจัดการสับเปลี่ยนค่าข้อมูลของหน่วยความจำหลักและจานบันทึกข้อมูลให้เหลือน้อยที่สุด จากการตัดแบ่งฐานข้อมูลออกเป็นฐานข้อมูลเล็กๆ เพื่อให้สามารถฝังตัวอยู่ในหน่วยความจำได้ทั้งหมดในขณะทำการประมวลผล โดยสามารถเพิ่มประสิทธิภาพของอัตราการเร่งความเร็วของระบบได้ถึงขอบเขตซูเปอร์ลีเนียร์เมื่อระบบทำการประมวลผลข้อคำถามจำนวนมากกว่าหนึ่งข้อคำถามขึ้นไป ทั้งนี้อัตราการเร่งความเร็วของระบบที่ได้ ขึ้นอยู่กับอัตราส่วนของฐานข้อมูลต่อขนาดของหน่วยความจำหลักของระบบที่ใช้ในการอ้างอิง แต่ถึงกระนั้นการเพิ่มจำนวนหน่วยประมวลผลขึ้นไปมากๆ มีแนวโน้มที่จะทำให้อัตราการเร่งความเร็วของระบบลดลง เนื่องมากจาก ระบบจำเป็นต้องทำการประมวลผลเบื้องต้น และทำการประมวลผลช่วงปลายเมื่อเสร็จสิ้นการเปรียบเทียบข้อคำถามทั้งหมดแล้ว ซึ่งจะกินเวลาคงที่ไม่ว่าจะใช้จำนวนหน่วยประมวลผลมากขนาดไหนก็ตาม

ทั้งนี้งานเบื้องต้นได้รับการทดสอบและทำการนำไปประยุกต์ใช้งานจริง[19] แล้วที่ ศูนย์จีโนม สถาบัน ชีววิทยา มหาวิทยาลัยพุทรา ประเทศมาเลเซีย (Genome Centre, Institute of Bioscience, Universiti Putra Malaysia) โดยปัจจุบันได้ทำการประมวลผลบนฐานงานระบบปฏิบัติการโซลารีส 8 บนเครื่องโดโนแฝน แซลเวอร์ (SOLARIS 8 on Donovan Salvor machine) ซึ่งมีตัวประมวลผลสองตัว ความเร็วตัวละ 450 เมกะเฮิร์ทซ์ โดยมีหน่วยความจำหลัก 128 เมกะไบต์ สำหรับแต่ละตัวประมวลผล ฐานข้อมูลที่ใช้คือฐานข้อมูล NR สำหรับอัลกอริทึม blastn ขนาด 1.5 กิกะไบต์ จากการประยุกต์ใช้ระบบที่ได้ทำการออกแบบทำให้สามารถหาผลลัพธ์จากข้อคำถามขนาด 1 กิโลไบต์ ภายในเวลาเพียง 40-55 วินาที จากเดิมใช้เวลา 60-75 วินาที ซึ่งมีอัตราการเร่งความเร็วสูงขึ้นจากเดิมถึง 1.36-1.5 เท่า

6.2 ปัญหาและข้อจำกัดที่ได้พบจากการวิจัย

1. ฐานข้อมูลที่ได้ทำการทดสอบเป็นฐานข้อมูลที่ไม่เป็นปัจจุบันเท่าที่ควรเมื่อเทียบกับฐานข้อมูลที่ใช้ในประเทศอื่นๆ
2. ระบบยังไม่สามารถรองรับการทำงานอัลกอริทึมและตัวแปรเสริมของแฟ้มกระทำการ blastall ได้ครบ ทั้งนี้ระบบได้ทำการทดสอบเฉพาะอัลกอริทึม blastx และ blastn เท่านั้น
3. ระบบที่ถูกพัฒนาขึ้นมามีข้อจำกัดโดยอยู่บนสมมุติฐานที่ว่า ฐานข้อมูลทั้งหมดถูกกระจายออกไปตามหน่วยประมวลผลต่างๆ และมีขนาดเท่ากันในทุกๆ หน่วยประมวลผล ทั้งนี้จะต้องทำการสร้างดัชนีเพื่อใช้งานไว้ก่อนล่วงหน้าก่อนเริ่มการประมวลผล
4. ระบบยังไม่มีความสามารถในการกู้คืนในกรณีที่ BLASTServer หรือ SharedStorage ไม่สามารถทำงานต่อได้หรือหลุดจากระบบ

6.3 ข้อเสนอแนะ

1. ควรปรับปรุงการทำงานของตัวโปรแกรมบลาสท์เพื่อรองรับปัญหาหน่วยความจำขนาดจำกัด
2. ควรปรับปรุงและพัฒนาความสามารถของระบบเพื่อรองรับการใช้งานเต็มรูปแบบ ทั้งนี้ควรจะสามารถใช้งานอัลกอริทึมอื่นๆ ที่แฟ้มกระทำการ blastall สามารถทำได้ทุกๆ อัลกอริทึม ตลอดจนสามารถรองรับการใช้งานตัวแปรเสริมต่างๆ ที่เหลือได้
3. ควรเพิ่มความสามารถในการกู้คืนให้กับระบบก่อนทำการใช้งานจริง โดยการบันทึกสถานะของการเปรียบเทียบของระบบเพื่อเริ่มการใช้งานใหม่หลังจากเสียหายของระบบ
4. ควรเพิ่มความสามารถในการคาดคะเนเวลาที่ระบบจะใช้ในการเปรียบเทียบ เพื่อให้ผู้ใช้สามารถทราบได้ว่าจะต้องรอผลลัพธ์เป็นเวลานานเท่าไร
5. ควรศึกษาเพิ่มเติมเพื่อปรับปรุงการประมวลผลเบื้องต้นของระบบ ตลอดจนถึงการรวมผลลัพธ์เข้าด้วยกันเมื่อเสร็จสิ้นการเปรียบเทียบข้อคำถามทั้งหมดให้สามารถกระจายไปทำงานในเครื่องอื่นๆ เพื่อเพิ่มประสิทธิภาพในการประมวลผลโดยรวม
6. ควรศึกษาเพิ่มถึงความเป็นไปได้ในการกระจายฐานข้อมูลและสร้างดรรชนีให้กับฐานข้อมูลเหล่านั้นในระหว่างทำการเปรียบเทียบเพื่อเพิ่มความคล่องตัวของระบบ

รายการอ้างอิง

[1] Altschul, S.F., Gish W., Miller W., Myers E.W., and Lipman D.J. Basic local alignment search tool. Journal of Molecular Biology. 215 (1990):403-410.

[2] Smith, T. F. and Waterman, M. S. Identication of common molecular subsequences. Journal of Molecular Theory. 147 (1981):195-197.

[3] Chi, E.H., Shoop, E., Carlis, J., Retzel, E. and Riedl, J. Efficiency of Shared-Memory Multiprocessors for a Genetic Sequence Similarity Search Algorithm. (1997)

[4] Camp, N., Cofer, H., and Gomperts, R. High-Throughput BLAST. SGI. 1998.

[5] Xian-He, S. Scalable Problems and Memory-Bounded Speedup. Journal of Parallel and Distributed Computing (1993)

[6] Amdahl, G.M. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities AFIPS Conference Proceedings. (1967)483-485.

[7] Singh, R. A scalable systolic multiprocessor system for analysis of biological sequences. Proceedings of Symposium on Integrated Systems. (1993)

[8] Needleman, S.B., and Wunsch, C.D. A general method applicable to the search for similarities in the amino acid sequences of two proteins Journal of Molecular Biology. 489 (1970):443-453.

[9] Smith, T.F. and Waterman, M.S. Identification of common molecular subsequences. Journal of Molecular Biology 147 (1981):195-197.

[10] Wattanapornprom W., Nupairoj N. and Chongstitvatana P. Improving the Performance of BLAST in a Memory Limited Environment. International Conference on Bioinformatics proceedings CD-Rom. (2002)

[11] Basic local alignment search tools. [Online] Available from: http://www.ncbi.nlm.nih.gov/blast/ April, 2001

[12] Washington University BLAST. [Online] Available from: http://blast.wustl.edu/ April, 2001

[13] BLAST Algorithm. [Online] Available from: http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/BLAST_algorithm.html April, 2001

[14] R.D. Bjornson, A.H. Sherman, S.B. Weston, N. Willard and J. Wing. A parallel implementation of BLAST based on the TurboHub process integration architecture. [Online]Available from: http://www.turbogenomics.com/products/turboblast.pdf April, 2001

[15] Optimized BLAST system for sequence similarity searching. [Online] Available from:http://www.paracel.com/products/blastmachine.html April, 2001

[16] Camp, N., Cofer, H., and Gomperts, R. High-Throughput BLAST. [Online] Available from:http://www.sgi.com/chembio/resources/papers/HTBlast/ April, 2001

[17] Aronsson L. Trends in Information Technology. [Online] Available from:http://www.lysator.liu.se/~aronsson November, 2001

[18] ปวีณา เลิศอำไพพร. การรู้จำยีนของข้าวโดยแบบจำลองมาร์คอฟ. วิทยานิพนธ์ปริญญามหาบัณฑิต. ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย. 2545

[19] W, Lee, researcher of Genome Centre,Institute of Bioscience, Universiti Putra Malaysia, 43300 UPM Serdang, Malaysia. [Personal communication], February 28, 2002.

ภาคผนวก

ภาคผนวก ก
รูปแบบฟาสตา (FASTA format)

รูปแบบฟาสตามีรูปแบบดังนี้

```
>gi|14090356|dbj|AP003233.3|AP003233 Oryza sativa genomic DNA, chromosome 1, PAC clone:P0037C04
TTACTTTTGTTTTCCTTTCGCCAATACTCTTCTGGACACAGGGCAAGTTTAGACCCTATTTAGATGGGAC
TAAATCCCTATCATATCGAATGTTTGGATACTAATTATAAATATTAAACGTGGACTATTAATAAACCCAT
TCTATAACCCAGAACTAATTCGCGAGACGAATCTATTGAGCCTAATTAATCAATGATTAGCCCATGTGAT
GCTACAGTAAACATACGCTAGTTATGGATTAATTAGGCTTAAAAAATTTATCACGCGAATTAGCTTCTTA
TTTATGTAATTAGTTTTATAAATAGTCTATGTTTAATACTCCAAATTCATCCGTTATGACATGGACTAAA
GTTTAGTCTTTGGATCCAAACTTATTCTTCAAACTTTCAACTTTTCCATCACATCAAAACTATCCTACAC
ACACAAACTTTCAATTTTTCCTTTCATATCGTTCTAACTTCAACCAAACTTCTAATTTTAACGTGAACTA
AACACACCCACAGTACACATGGGAAAGAAGCTGGTTCCAGGGTTAAAGGTCCTGCACTGCAGGCGTTGGC
TGCGCTGGACAGAAAAAAAAATGACCAAACCAAAAACCGAACTGAATTAACAGAGACCGAGAAATTCGAT
CATCAGTTGTGACTAATTGAATTTAACACTGTCTTTTAAAACGAAATAACTGAGCTGACCGAATCGATGT
```

รูปที่ ก.1 ตัวอย่าง FASTA format

รูปแบบฟาสตาเริ่มต้นโดยบรรทัดแรกจะมีเครื่องหมายมากกว่า (>) ตามด้วยคำบรรยายจำนวน 1 บรรทัด เรียกว่า FASTA definition line ซึ่งจะบอกเกี่ยวกับ Accession number หรือ GenBank Identifier และ Locus number ส่วนบรรทัดต่อไปเป็นลำดับซึ่งมีความกว้างไม่เกิน 80 คอลัมน์ (รวมตัวอักษรและช่องว่าง)

สำหรับอักษรย่อที่ใช้แทนนิวคลีโอไทด์หรือกรดอะมิโนต้องเป็นไปตามมาตรฐาน IUB/IUPAC ดังนี้

นิวคลีโอไทด์ :

A แทน adenosine	M แทน A C (amino)
C แทน cytidine	S แทน G C (strong)
G แทน guanine	W แทน A T (weak)
T แทน thymidine	B แทน G T C
U แทน uridine	D แทน G A T
R แทน purine	H แทน A C T
Y แทน pyrimidine	V แทน G C A

K แทน GT (keto)

N แทน A G C T (any)

- แทน gap

กรดอะมิโน :

A แทน alanine

B แทน aspartate or asparagine

C แทน cystine

D แทน aspartate

E แทน glutamate

F แทน phenylalanine

G แทน glycine

H แทน histidine

I แทน isoleucine

K แทน lysine

L แทน leucine

M แทน methionine

N แทน asparagine

P แทน proline

Q แทน glutamine

R แทน arginine

S แทน serine

T แทน threonine

U แทน selenocysteine

V แทน valine

W แทน tryptophan

Y แทน tyrosine

Z แทน glutamate or glutamine

X แทน any

* แทน translation stop

- แทน gap of indeterminate

ภาคผนวก ข
การใช้งานโปรแกรมบลาสท์

ในการเปรียบเทียบของโปรแกรมบลาสท์จำเป็นจะต้องมีฐานข้อมูลที่ได้ทำการสร้างดัชนี (index) ในการค้นหาไว้แล้วโดยการใช้งานโปรแกรมที่มีชื่อว่า formatdb ที่อยู่ในชุดของโปรแกรมบลาสท์ และ ข้อคำถามที่ใช้ในการเปรียบเทียบลำดับสองเส้นโดยจะเรียกจากแฟ้มกระทำการที่ชื่อ blastall

การใช้งานแฟ้มกระทำการ formatdb

formatd เป็นโปรแกรมที่ใช้เพื่อจัดรูปแบบของฐานข้อมูลโปรตีน หรือฐานข้อมูลของสายนิวคลีโอไทด์ ก่อนที่ฐานข้อมูลเหล่านี้จะสามารถถูกค้นหาโดยโปรแกรม blastall โปรแกรม blastpgp หรือ โปรแกรม MegaBLAST สำหรับข้อมูลภายในฐานข้อมูลนั้นสามารถเป็นไปได้สองรูปแบบ คือ รูปแบบฟาสตา หรือ รูปแบบ ASN.1 ถึงแม้ว่าโดยทั่วรูปแบบฟาสตา จะเป็นที่นิยมใช้เป็นข้อมูลนำเข้ามากกว่า แต่รูปแบบ ASN.1 ก็เป็นรูปแบบที่มักถูกใช้ในรายงานของ GenBank

การใช้งานโปรแกรม formatdb จะเรียกแฟ้มกระทำการ formatdb ตามด้วยทางเลือกในการประมวลผลของโปรแกรมดังนี้

-t ตามด้วยชื่อที่ต้องการตั้งให้แฟ้มฐานข้อมูลที่ต้องการทำดัชนี [สายอักขระ] (ไม่บังคับใช้)

-i ตามด้วยชื่อแฟ้มของฐานข้อมูลที่ใช้เป็นข้อมูลนำเข้า [สายอักขระ] (บังคับใช้)

-l ชื่อแฟ้มข้อมูลแสดงผลลัพธ์ [สายอักขระ] (ไม่บังคับใช้) ค่าโดยปริยาย (default value) จะให้เป็น formatdb.log

-p ประเภทของแฟ้มฐานข้อมูลนำเข้า ซึ่งเป็นไปได้ 2 แบบ คือ T แทนข้อมูลโปรตีน และ F แทนข้อมูลสายนิวคลีโอไทด์ [อักขระ] (ไม่บังคับใช้) ถ้าไม่ใส่ ค่าโดยปริยายจะเป็น T

-o ทางเลือกกระจาย มีให้เลือก 2 แบบ คือ T กระจายตามรหัสลำดับ (SeqId) และสร้างดัชนี (Index) ให้ด้วย F ไม่กระจายตามรหัสซีเควน (SeqId) ไม่ต้องสร้างดัชนี [ตัวเลือกตรรกะแบบบูล T/F] (ไม่บังคับใช้)

-a ข้อมูลนำเข้าเป็นข้อมูลที่อยู่ในรูปแบบ ASN.1 T ข้อมูลอยู่ในรูปแบบ ASN.1 F ข้อมูลอยู่ในรูปแบบ FASTA [ตัวเลือกตรรกะแบบบูล T/F] (ไม่บังคับใช้)

-b ฐานข้อมูล ASN.1 อยู่ในรูปแบบภาวะทวิภาค (Binary Mode) T ASN.1 รูปแบบภาวะทวิภาค F ASN.1 รูปแบบแอสกี (ASCII) โดยทั่วไปแล้ว ข้อมูลในรูปแบบ ASN.1 จะแสดงได้ 2 รูปแบบ คือ แบบแอสกี และ แบบภาวะทวิภาค ในกรณีที่ข้อมูลนำเข้าเป็นรูปแบบ ฟาสตาตัวเลือก -b นี้จะไม่ต้องใช้ [ตัวเลือกตรรกะแบบบูล T/F] (ไม่บังคับใช้)

-e ข้อมูลนำเข้าเป็นแบบ Seq-entry ตามปกติรูปแบบ ASN.1 จะประกอบด้วยลับดับเดียวหรือชุดของลำดับหากข้อมูลเป็นชุดของลำดับให้ใช้ทางเลือกนี้ [ตัวเลือกตรรกะแบบบูล T/F] (ไม่บังคับใช้)

-n ชื่อเบส สำหรับแฟ้มข้อมูลบลาสท์ ใช้เมื่อต้องการให้ชื่อกับฐานข้อมูลบลาสท์ให้แตกต่างจากแฟ้มข้อมูลฟาสตาเดิม [สายอักขระ] (ไม่บังคับใช้)

-v ใช้เมื่อต้องการแตกแฟ้มข้อมูลฟาสตาออกเป็นแฟ้มข้อมูลย่อยๆ หลายๆ แฟ้มข้อมูล [เลขจำนวนเต็ม] (ไม่บังคับใช้)

-s ใช้เมื่อต้องการสร้างดัชนีสำหรับข้อมูลที่มีเลขภาคีเหมือนกันเนื่องมาจากฐานข้อมูลบางชนิด เช่น ฐานข้อมูล EST STS GSS หรือ HTGS จะใช้เลขภาคี เป็นเลขดัชนีไม่ได้เพราะข้อมูลบางตัวในฐานข้อมูลสามารถมีเลขภาคีเหมือนกันได้ [ตัวเลือกตรรกะแบบบูล T/F] (ไม่บังคับใช้) ค่าโดยปริยายคือ F

-A สร้างฐานข้อมูลแบบ ASN.1 [ตัวเลือกตรรกะแบบบูล T/F] (ไม่บังคับใช้) ค่าโดยปริยายคือ F

-L สร้างแฟ้มสมนามด้วยชื่อนี้ [ชื่อแฟ้มข้อมูลผลลัพธ์] (ไม่บังคับใช้)

การใช้งานแฟ้มกระทำการ blastall

โปรแกรมบลาสท์เป็นชุดของโปรแกรมย่อยของโปรแกรมบลาสท์ทั้งห้าโปรแกรมคือ blastp, blastn, blastx , tblastn, และ tblastx โดยเวลาใช้งานจะเรียกผ่านแฟ้มกระทำการที่ชื่อ blastall ตามด้วยทางเลือกในการประมวลผลของโปรแกรมดังนี้

-p ตามด้วยชื่อของโปรแกรมย่อย [ตัวเลือกสายอักขระ blastp/ blastn/ blastx/ tblastn/ tblastx] (บังคับใช้)

-d ชื่อแฟ้มฐานข้อมูลที่ได้ทำการสร้างดัชนีในการค้นหาไว้แล้ว [สายอักขระ] (บังคับใช้) สามารถมีได้หลายชื่อโดยจะต้องอยู่ภายในเครื่องหมายอัญประกาศ ["กลุ่มของสายอักขระคั่นด้วยช่องว่าง"] ดังตัวอย่าง –d "nr est" ทั้งนี้ผลลัพธ์ที่ได้จะเกิดจากการเปรียบเทียบลำดับทั้งในฐานข้อมูล nr และ est โดยจะเป็นผลลัพธ์จากฐานข้อมูลเสมือนที่เกิดจากฐานข้อมูลสองตัวต่อกัน

-i ชื่อแฟ้มข้อคำถาม [ชื่อแฟ้มข้อมูลนำเข้า] (บังคับใช้) โดยข้อคำถามควรจะอยู่ในรูปแบบฟาสตา และหากมีข้อคำถามหลายๆ ลำดับในแฟ้ม ข้อคำถามทั้งหมดจะถูกนำมาเปรียบเทียบ

-e ค่าความเชื่อถือได้ของผลลัพธ์ใช้สำหรับกำหนดค่าขั้นต่ำของความน่าเชื่อถือของผลลัพธ์ที่ได้ [จำนวนจริง] (ไม่บังคับใช้)

-o ชื่อแฟ้มผลลัพธ์ [ชื่อแฟ้มข้อมูลผลลัพธ์] (ไม่บังคับใช้) ค่าโดยปริยายคือ stdout

-F ชื่อตัวกรองลำดับ (ตัวกรอง DUST สำหรับโปรแกรม blastn และตัวกรอง SEQ สำหรับโปรแกรมอื่นๆ) โดยตัวกรองดังกล่าวเป็นตัวกรองมาตรฐานของ NCBI [ชื่อตัวกรอง] (ไม่บังคับใช้) ค่าโดยปริยายคือ T (หากใช้ค่าปริยายโปรแกรมจะกำหนดค่าตัวกรองเอง หากกำหนดค่าตัวกรองเป็น F โปรแกรมจะไม่ใช้ค่าตัวกรองไม่เช่นนั้นจะใช้ชื่อของตัวกรองที่กำหนด)

-S สายของลำดับที่จะใช้เปรียบเทียบกับฐานข้อมูล (สำหรับโปรแกรม blastn, blastx, และ tblastx) [จำนวนเต็ม] (ไม่บังคับใช้) ค่าโดยปริยายคือ 3 ([1] เพื่อเปรียบเทียบจากด้านบน [2] เพื่อเปรียบเทียบกับสายตรงข้ามจากด้านล่างและ [3] เพื่อใช้เปรียบเทียบทั้งสองทาง)

-T เพื่อสร้างแฟ้มผลลัพธ์ในรูปแฟ้มเอชทีเอ็มแอล (HTML file) [ตัวเลือกตรรกะแบบบูล T/F] (ไม่บังคับใช้) ค่าโดยปริยายคือ F

-U ใช้ตัวกรองตัวพิมพ์เล็ก [ตัวเลือกตรรกะแบบบูล T/F] (ไม่บังคับใช้) ค่าโดยปริยายคือ F

ภาคผนวก ค
ผลงานตีพิมพ์

การประชุมทางวิชาการชีวสารสนเทศศาสตร์นานาชาติ ครั้งที่ 1 (The 1st International Conference on Bioinformatics (InCoB2002)) เมื่อวันที่ 6-8 กุมภาพันธ์ 2545 ในบทความเรื่อง Improving the Performance of BLAST in a Memory Limited Environment โดยผู้แต่งคือ Warin Wattanapornprom, Natawut Nupairoj and Prabhas Chongstitvatana

Improving the Performance of BLAST in a Memory Limited Environment

Warin Wattanapornprom[1], Natawut Nupairoj[2] and Prabhas Chongstitvatana[3]
Department of Computer Engineering, Chulalongkorn University
Phayathai Rd., Phathumwan Bangkok 10330, THAILAND
E-Mail: warin@chula.com[1], natawut@cp.eng.chula.ac.th[2] and prabhas@chula.ac.th[3]

Abstract: BLAST has become an important tool for the research in bioinformatics areas, since it can help scientists to make inferences about the functions of proteins. The BLAST's database is enormous and has been growing every day, and this causes the lower performance of the program. This paper presents an alternative way to improve the performance of BLAST in a single machine by reducing the overhead of disk swapping.

Key words: BLAST, Distributed database, Parallel search, Clustering, Memory Limited

1. Introduction

BLAST (Basic Local Alignment Search Tool) [1] is one of the most widely used search tools, which identifies statistically significant matches between newly sequenced segments of genetic material or proteins and databases of known nucleotide or amino acid sequences. Such searches allow scientists to make inferences about the structures and functions of their discoveries or to screen new sequences for further investigation.

Although BLAST have been designed and optimized for speed, the major drawback of BLAST is that it consumes large amount of CPU-time, memory, and I/O. It takes a very long time to search a large number of queries in a large database. There were attempts to reduce the searching time using many approaches such as upgrading computer hardware, using some parallel approach such as parallel queries search or using multiple processor machines. Our preliminary study of BLAST indicates that BLAST's running time is proportional to the size of the database. BLAST shows the highest efficiency if the whole database can be fitted in the memory. As the genome databases are enormous and doubling in size every 1.3 years [2], it is important to recognize the performance limitation due to the limited main memory.

To overcome this problem, we propose to separate the database into smaller parts, which each part fits the available memory and then search each part separately. We find that the time used for searching all separated parts is almost equal to the time used to search the whole database with the memory large enough to hold it. This provides us the maximum efficiency for a given memory size.

This paper reports on our progress to design and develop the parallel system environment for BLAST. The paper is organized as follows. In Section 2, we begin with a preliminary experiment to the better understanding of the behavior of BLAST and the essential knowledge to improve the performance of the program. Then we describe the process to improve the performance and the result of the improved system in Section 3. Section 4 concludes the paper with status of the current prototype implementation and discussions of some future work.

2. Background

From our preliminary study of BLAST, we find that BLAST needs high bandwidth of the main memory. BLAST will show the highest efficiency if the whole database can be fitted in the main memory while searching. In the experiment, we vary the memory size while maintaining the other system environment such as CPU or hard disk; we find that the system with memory less than the size of the database will take longer time to process large amount of queries while the system with excess memory will not increase the performance of the search. This problem is called “memory-bounded problem”, a problem in parallel processing [3].

Memory Size	Number of Queries				
(Megabyte)	10	50	100	500	1000
128	5	33	70	351	706
256	2	15	37	180	364
512	2	14	37	180	363

Table 1 shows the performance of BLAST at the different system environments

From the sample experiment, we use database named NR size 234 megabytes to represent the performance of the program because this size of database can fit into 256 megabytes of memory or higher. We can see that the performance is improved when we use larger size of memory. But if the memory size is too large, the system can no longer be improved. As we can see in the table 1, the performance of BLAST at 512 megabytes of memory cannot improve the performance from the system with 256 megabytes of memory at all.

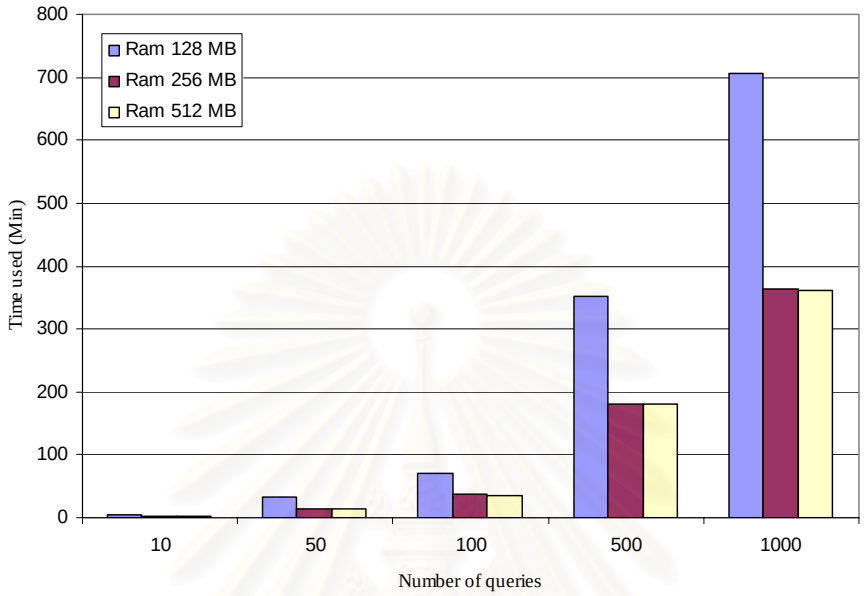


Figure 1 shows the performance of BLAST at the different system environments

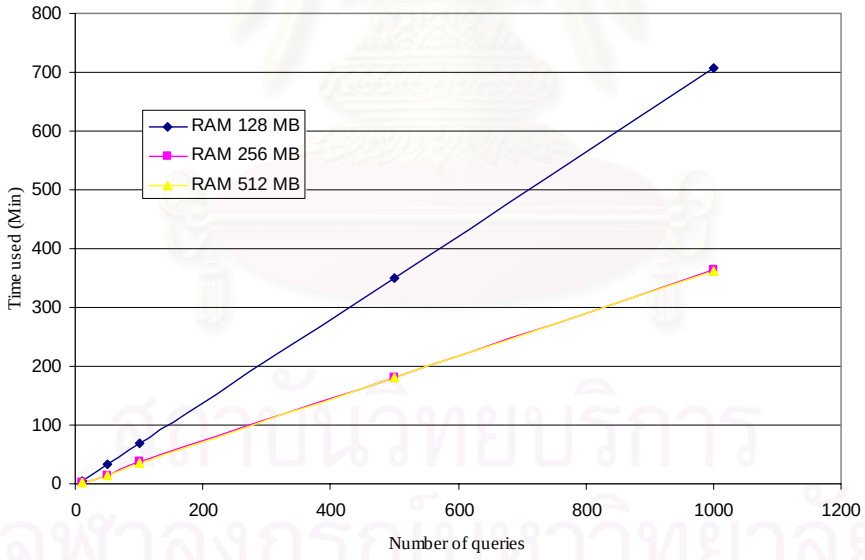


Figure 2 shows the linear performance of BLAST at the different system environments

The behavior of the searching process can be described. BLAST tries to copy the whole database into the main memory while searching every single query in the database. If the database is too large to fit into the main memory, the operating system will automatically allocate the memory space by swapping

the memory contents. The disk throughput is normally much less than the memory throughput, so the time is wasted by swapping between disk and memory. Since the memory which can hold the whole database at running time has to be enormous and expensive, moreover the systems that can install large size of database are scarce, the system should be improved to solve this problem.

We find that the database can be divided into smaller pieces. BLAST's database structure is simply a text file. Each database record is started with character ">" followed by necessary fields separated by "|" and ended up with protein or DNA sequence. The database file can be cut at the end of any record which is just before the ">".Output from the separated database can also be merged by just concatenate the result of each query together.

```
CKLSIKRATVIYEGERVAIQNKFKNGMLHGQKVSFFCKHKEKKCSYTEDAQCIDGTIEIPKCFKEHSSLAFWKTDASDVKPC
> gi|129249|sp|P02820|OSTC_BOVIN OSTEOCALCIN PRECURSOR (GAMMA-CARBOXYGLUTAMIC ACID-
CONTAINING PROTEIN) (BONE GLA-PROTEIN) (BGP)_gi|538590|pir||GEBO osteocalcin precursor -
bovine_gi|8|emb|CAA35997.1| (X51700) bone Gla precursor (100 AA) [Bos taurus]_gi|720|emb|CAA37737.1|
(X53699) Gla protein precusor [Bos taurus]
```

Figure 2 shows an example of the BLAST's database records

3. Performance Improvement

The system performance is hypothetically better if the database size is fitted the memory size while searching. To avoid the overhead caused by swapping, we propose steps to reduce the time to search a large number of sequence queries as follows.

1. Preprocessing – separate the database into smaller parts to be able to fit in the memory (the size should be almost similar to the available memory) then create index files for each of the.
2. Processing – Search all queries in each of the splited database parts.
3. Postprocessing – Concatenate the output.

We proof this hypothesis by separate the database into different sizes which are small enough to fit in the main memory then process all queries and concatenate the output together. The experiment is carried out in one machine to eliminate the overhead of communication between machines. The total time is the sum of all processing time in each separated databases and postprocessing time. The preprocessing time is not counted because the preprocessing is done only once before processing any query.

$$t_{total} = \sum_{i=1}^{n} t_i + t_{post}$$

Where

t_{total} = Total time used.

t_{post} = Time to concatenate all output.

t_i = Time to process database part i.

n = Number of separated databases.

No. of separated Database	Database Part No.	Database size (Megabyte)	Time used (min)	Time used to concatenate (min)	Total time used (min)
1	1	234	70	0	70
2	1	109	17	1	38
	2	125	20		
3	1	100	16	1.5	38.5
	2	100	15		
	3	34	6		

Table 2 shows the performance of BLAST at various numbers of separated databases.

The experiment is done with the same database in the section 2. We separate the database into different sizes and process the same 100 of queries at the fixed memory size at 128 megabytes. We can see that no matter how many parts the database is separated, if each part of the database fits the memory, the time used to process all of them will equal to the time used to process the original database in the environment which memory can contain the whole database at runtime. The time used to concatenate the output depends on number of separated databases because the more output file will need more time to concatenate. The performance of the proposed method can be measured using “speedup”. Speedup is the ratio of the normal execution time to the improved execution time. In our experiment, the speedup of the improved system is the total time used in an improved system divided by the time used in a single database search which is 70/38 = 1.84 times faster.

The results show that the time used to search through the separated databases at the limited memory size is equal to the time used to search in the environment that memory size is large enough to hold the whole database. In fact, there are some overhead in merging the outputs which depend on the search result and the number of separated databases. The overhead is less than a minute in this case and will take longer time depends on the pieces of the separated database.

4. Conclusions and Future work

In this paper, we have studied the main factor which slowdowns the searching process of BLAST and have found the way to solve this problem. Since BLAST's databases are enormous and have been growing larger every year and main memory cannot hold the whole database at running time, so the overheads are generated by swapping of the memory contents. We propose to separate the database to be able to fit the available memory and search through each database part separately. This provides us the maximum efficiency for a given memory size. Form the results of our experiments, BLAST shows that the performance can be doubled when we apply our technique. However, the improvement can be varied depended on the database size, the disk speed and the size of memory. Our proposed steps are not only enabling BLAST to search through the whole database at the peak performance, but also reducing the access time. The access time of the disk is reduced due to the smaller database fits in the given memory and requires no disk swapping.

We plan to expand our studies to apply our techniques to improve the performance of the searching on the clustered computers by distributing each database part to each of the clustering node and performing the search in parallel, and we expect even greater performance improvement. We are in the process of designing an optimized system at different number of processors to maximize the performance of the system.

5. References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. & Lipman, D.J. "Basic local alignment search tool" *Journal of Molecular Biolog.* 215:403-410. (1990)

[2] Chi E.H., Shoop E., Carlis J., Retzel E., Riedl J,"Efficiency of Shared-Memory Multiprocessors for a Genetic Sequence Similarity Search Algorithm" (1997)

[3] Xian-He S, "Scalable Problems and Memory-Bounded Speedup" *Journal of Parallel and Distributed Computing* (1993)

ประวัติผู้เขียนวิทยานิพนธ์

นายวรินทร์ วัฒนพรพรหม เกิดวันที่ 24 กันยายน พ.ศ. 2521 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาระดับปริญญาตรีวิทยาศาสตรบัณฑิต สาขาเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีนานาชาติสิรินธร มหาวิทยาลัยธรรมศาสตร์ ในปีการศึกษา 2542 และ เข้าศึกษาต่อในหลักสูตรวิทยาศาสตร์มหาบัณฑิต สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อ พ.ศ. 2543