โครงการ

การเรียนการสอนเพื่อเสริมประสบการณ์

ชื่อโครงการ ขั้นตอนวิธีแบบขนานที่ใช้หน่วยความจำอย่างมีประสิทธิภาพสำหรับหาคำตอบที่ดีที่สุดในปริศนาที่มีการเลื่อนเป็นลำดับ
Memory-Efficient Parallel Algorithm for Finding Optimal Solution to Sequential Move Puzzle

ชื่อนิสิต นายวรวุธ วรวิชญวงศา 583 36563 23

ภาควิชา คณิตศาสตร์และวิทยาการคอมพิวเตอร์
สาขาวิชาวิทยาการคอมพิวเตอร์

ปีการศึกษา 2561

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ขั้นตอนวิธีแบบขนานที่ใช้หน่วยความจำอย่างมีประสิทธิภาพสำหรับหาคำตอบที่ดีที่สุดในปริศนาที่มีการเลื่อนเป็นลำดับ

นายวรวุธ วรวิชญวงศา

โครงงานนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์ ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2561

Memory-Efficient Parallel Algorithm for Finding Optimal Solution to Sequential Move Puzzle

Worawut Worawichwongsa

A Project Submitted in Partial Fulfillment of the Requirements

for the Degree of Bachelor of Science Program in Computer Science

Department of Mathematics and Computer Science

Faculty of Science

Chulalongkorn University

Academic Year 2018

หัวข้อโครงงาน	ขั้นตอนวิธีแบบขนานที่ใช้หน่วยความจำอย่างมีประสิทธิภาพสำหรับหาคำตอบที่ดีที่สุดในปริศนาที่มีการเลื่อนเป็นลำดับ
โดย	นายวรวุธ วรวิชญวงศา
สาขาวิชา	วิทยาการคอมพิวเตอร์
อาจารย์ที่ปรึกษาโครงงานหลัก	ผู้ช่วยศาสตราจารย์ ดร.ศุภกานต์ พิมลธเรศ

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับโครงงานฉบับนี้เป็นส่วนหนึ่ง ของการศึกษาตามหลักสูตรปริญญาบัณฑิต ในรายวิชา 2301499 โครงงานวิทยาศาสตร์ (Senior Project)

.....
(ศาสตราจารย์ ดร.กฤษณะ เนียมมณี) หัวหน้าภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

คณะกรรมการสอบโครงงาน

.....
(ผู้ช่วยศาสตราจารย์ ดร.ศุภกานต์ พิมลธเรศ) อาจารย์ที่ปรึกษาโครงงานหลัก

.....
(ผู้ช่วยศาสตราจารย์ สุรชัย สมบัติบริบูรณ์) กรรมการ

.....
(อาจารย์ ดร.วุฒิชัย จงจิตเมตต์) กรรมการ

วรวุธ วรวิชญวงศา: ขั้นตอนวิธีแบบขนานที่ใช้หน่วยความจำอย่างมีประสิทธิภาพสำหรับหาคำตอบที่ดีที่สุดในปริศนาที่มีการเลื่อนเป็นลำดับ. (Memory-Efficient Parallel Algorithm for Finding Optimal Solution to Sequential Move Puzzle) อ.ที่ปรึกษาโครงงานหลัก : ผศ.ดร.ศุภกานต์ พิมลธเรศ, 47 หน้า.

ปริศนาที่มีการเลื่อนเป็นลำดับเป็นกลุ่มปัญหาสำหรับทดสอบขั้นตอนวิธีในการค้นหาในกราฟ เนื่องจากปริศนาที่มีการเลื่อนเป็นลำดับสามารถแทนได้ด้วยกราฟ จึงสามารถค้นหาเส้นทางที่สั้นที่สุดระหว่างสองจุดยอดได้ โดยมักจะให้จุดยอดหนึ่งจุดแทนรูปแบบเริ่มต้นของปัญหา และเส้นทางที่สั้นที่สุดไปสู่จุดยอดนั้นจะเป็นคำตอบที่ดีที่สุดของปริศนา งานวิจัยนี้จึงออกแบบให้จำนวนเต็มแทนรูปแบบต่างๆของปัญหา และใช้หน่วยประมวลผลกราฟิกเพื่อเพิ่มความเร็วของขั้นตอนวิธีการค้นหาในกราฟ ผลการทดสอบแสดงให้เห็นว่าขั้นตอนวิธีในงานวิจัยนี้สามารถเพิ่มความเร็วได้ถึงสี่เท่าและแปดเท่าสำหรับปริศนาเรียงแปดเลขและปริศนาเรียงสิบห้าเลขตามลำดับ และลดปริมาณหน่วยความจำที่จำเป็นต้องใช้ลงได้

ภาควิชา คณิตศาสตร์และวิทยาการคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิทยาการคอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาโครงงานหลัก .....

ปีการศึกษา 2561

5833656323: MAJOR COMPUTER SCIENCE

KEYWORDS : COMBINATION PUZZLE / PARALLEL PROGRAMMING / GRAPH SEARCHING

WORAWUT WORAWICHWONGSA: MEMORY-EFFICIENT PARALLEL ALGORITHM FOR FINDING OPTIMAL SOLUTION TO SEQUENTIAL MOVE PUZZLE. ADVISOR : ASST. PROF. SUPHAKANT PHIMOLTARES, Ph.D., 47 pp.

Sequential move puzzle is a group of problems that is usually used for testing graph searching algorithms. Since sequential move puzzle can be represented by a graph, the shortest path between any vertices can be found easily. For such puzzle, one of vertices will be an initial configuration and the shortest path to that vertex will be an optimal solution to the puzzle. In this research, each configuration will be assigned by distinct integer and GPU is used to accelerate the graph searching algorithm. The result shows that the proposed algorithm can speed up by x4 and x8 for 8-puzzle and 15-puzzle respectively and reduce the memory used during the calculation.

Department : Mathematics and Computer Science Student's Signature

Field of Study : Computer Science Advisor's Signature

Academic Year : 2018

กิตติกรรมประกาศ

การวิจัยขั้นตอนวิธีแบบขนานที่ใช้หน่วยความจำอย่างมีประสิทธิภาพสำหรับหาคำตอบที่ดีที่สุดในปริศนาที่มีการเลื่อนเป็นลำดับ ได้รับการสนับสนุนจากจุฬาลงกรณ์มหาวิทยาลัย ทั้งในด้านองค์ความรู้ สิทธิในการเข้าถึงงานวิจัยต่างๆ และสถานที่สำหรับค้นหาความรู้เพิ่มเติม ซึ่งทำให้งานวิจัยสามารถสำเร็จลุล่วงไปด้วยดี รวมถึง ผศ.ดร.ศุภกานต์ พิมลธเรศ ที่ได้ให้คำปรึกษาสำหรับปัญหาต่างๆ ระหว่างการทำงาน ข้าพเจ้าจึงใคร่ขอขอบพระคุณเป็นอย่างยิ่งสำหรับความช่วยเหลือในทุกๆ ด้าน และหวังว่าผลการวิจัยนี้จะเป็นประโยชน์แก่การแก้ปัญหาและพัฒนาวิธีการค้นหาต่างๆต่อไป

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและเหตุผลการวิจัย

ปริศนาที่มีการเลื่อนเป็นลำดับหรือที่เรียกอีกอย่างว่าปริศนาการเรียงสับเปลี่ยน (Combination Puzzle / Sequential Move Puzzle) คือกลุ่มของปริศนาที่มีส่วนเคลื่อนตำแหน่งได้ ซึ่งอาจจะเลื่อนหรือหมุนเพื่อให้ตำแหน่งของชิ้นส่วนนั้นๆเปลี่ยนแปลงไป โดยปัญหาคือต้องเปลี่ยนรูปแบบใดๆของปริศนานั้นให้กลับมายังรูปแบบเริ่มต้น ซึ่งปริศนาส่วนมากสามารถแก้ไขได้แล้ว (Scherphuis, 2015) แต่คำตอบที่ได้มามักยังไม่ใช่คำตอบที่ดีที่สุด

การแก้ปริศนาที่มีการเลื่อนเป็นลำดับโดยใช้จำนวนการเลื่อนให้น้อยที่สุดนั้น เป็นหนึ่งในปัญหาที่ท้าทายมากในทางวิทยาการคอมพิวเตอร์ ปัญหาดังกล่าวหลายอย่างถูกระบุให้อยู่กลุ่มปัญหาเอ็นพีบริบูรณ์ (NP-Complete) เช่น ปัญหาเรียงเลข (Ratner & Warmuth, 1990) และลูกบาศก์ของรูบิก (Demaine, Eisenstat, & Rudoy, 2018) ซึ่งการใช้ขั้นตอนวิธีการแก้ปกติแบบลำดับนั้นไม่สามารถแก้ปัญหากลุ่มได้ในเวลาที่ยอมรับได้ ในขณะที่ขั้นตอนวิธีแบบขนานกำลังเป็นที่นิยมใช้ในคอมพิวเตอร์ปัจจุบัน และอุปกรณ์สำหรับใช้ร่วมกับขั้นตอนวิธีแบบขนานก็กำลังถูกพัฒนาอย่างรวดเร็วเช่นกัน อย่างไรก็ตามการใช้ขั้นตอนวิธีแบบขนานสามารถลดได้แค่เวลาที่ใช้ในการประมวลผล แต่กลุ่มปัญหาดังกล่าวต้องการทั้งเวลาและหน่วยความจำปริมาณมากในการแก้ปัญหา คอมพิวเตอร์ปัจจุบันยังมีหน่วยความจำไม่พอเก็บคำตอบดังกล่าวได้ทั้งระหว่างคำนวณและภายหลังจากการคำนวณจึงจำเป็นที่จะต้องมีโครงสร้างการเก็บข้อมูลที่มีประสิทธิภาพร่วมด้วย

อุปกรณ์สำหรับการประมวลผลแบบขนานที่มีประสิทธิภาพสูงสำหรับคอมพิวเตอร์ส่วนตัวในปัจจุบันคือหน่วยประมวลผลกราฟิก (GPU) (Brodtkorb, Hagen, & Sætra, 2013) โดยมีชุดคำสั่งจำนวนมากที่ใช้หน่วยประมวลผลกราฟิกเป็นตัวช่วยเพิ่มประสิทธิภาพการประมวลผล เนื่องจากการพัฒนาหน่วยประมวลผลกราฟิกให้สามารถประมวลผลข้อมูลทั่วไปได้ และการปฏิบัติการจุดลอยตัวต่อวินาที (FLOPS) ที่สูงกว่าหน่วยประมวลผลกลางมากในราคาที่เท่ากัน ทำให้หน่วยประมวลผลกราฟิกมักใช้เป็นหน่วยประมวลผลแบบหลายแกนที่ประมวลผลขั้นตอนวิธีแบบขนานในคอมพิวเตอร์ส่วนตัว

รูปแบบของการแก้ปัญหาดังกล่าวมักจะถูกสร้างเป็นตัวแบบทางคณิตศาสตร์ในรูปของกราฟโดยมีจุดยอดเป็นสถานะหรือรูปแบบที่เปลี่ยนไปของปริศนานั้น และมีเส้นเชื่อมจุดยอดแต่ละจุดเป็นการเคลื่อนตำแหน่งที่สามารถทำได้ ทำให้สามารถนำขั้นตอนวิธีค้นหาในกราฟมาแก้ปริศนานี้ได้เช่นกัน

1.2 วัตถุประสงค์ของการวิจัย

เพื่อพัฒนาขั้นตอนวิธีแบบขนาน ที่สามารถค้นหาคำตอบที่ดีที่สุด สำหรับทุกรูปแบบของปริศนาที่มีการเลื่อนเป็นลำดับ โดยใช้หน่วยความจำให้น้อยที่สุด ในเวลาที่ใช้ในการประมวลผลที่ยอมรับได้ และสร้างออกมาเป็นตารางคำตอบ

1.3 ขอบเขตการวิจัย

1. ตัวงานต้องใช้กับปัญหาที่มีความซับซ้อนของรูปแบบได้มากถึง 10^{10}
2. ตัวงานต้องใช้กับปัญหาที่มีการเลื่อนเป็นลำดับได้ทุกรูปแบบ
3. คำตอบที่ได้จากตัวงานนี้ต้องเป็นคำตอบที่ดีที่สุด
4. ตัวงานทำงานบนคอมพิวเตอร์ที่มีหน่วยประมวลผลกราฟิกของ Nvidia ซึ่งรองรับ CUDA

1.4 ขั้นตอนการวิจัย

การวิจัยขั้นตอนวิธีแบบขนานที่ใช้หน่วยความจำอย่างมีประสิทธิภาพสำหรับหาคำตอบที่ดีที่สุดในปริศนาที่มีการเลื่อนเป็นลำดับ มีขั้นตอนการดำเนินการตามตารางที่ 1.1 ดังนี้

ตารางที่ 1.1 ตารางแสดงขั้นตอนการดำเนินงาน

การดำเนินงาน	2561					2562			
	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ	มี.ค.	เม.ย.
1. ศึกษางานวิจัยที่เกี่ยวข้อง									
2. ทดสอบขั้นตอนวิธีในงานวิจัย									
3. วิเคราะห์ปัญหาและพัฒนาขั้นตอนวิธีใหม่									
4. นำขั้นตอนวิธีใหม่ไปใช้									
5. วิเคราะห์ประสิทธิภาพขั้นตอนวิธี									
6. เขียนรายงาน									

1.5 ประโยชน์ที่คาดว่าจะได้รับ

ประโยชน์ต่อตัวนิสิต

1. ได้ศึกษาเพิ่มเติมเกี่ยวกับกราฟและการเรียงสับเปลี่ยน
2. ได้ศึกษาเพิ่มเติมเกี่ยวกับสถาปัตยกรรมการประมวลผลแบบขนานสำหรับหน่วยประมวลผลกราฟิก
3. ได้ศึกษาเพิ่มเติมเกี่ยวกับโครงสร้างการเก็บข้อมูลแบบใหม่
4. ได้ประสบการณ์การใช้หน่วยประมวลผลกราฟิกในการประมวลผลแบบขนาน

ประโยชน์ของโครงงาน

1. ได้ขั้นตอนวิธีที่ใช้เวลาในการค้นหาที่รวดเร็ว และตารางคำตอบที่เล็กลง สำหรับปริศนาที่มีการเลื่อนเป็นลำดับ
2. ได้แนวคิดการค้นหาคำตอบในต้นไม้แบบใหม่ สำหรับการพัฒนาต่อไปในอนาคต

1.6 โครงสร้างของรายงาน

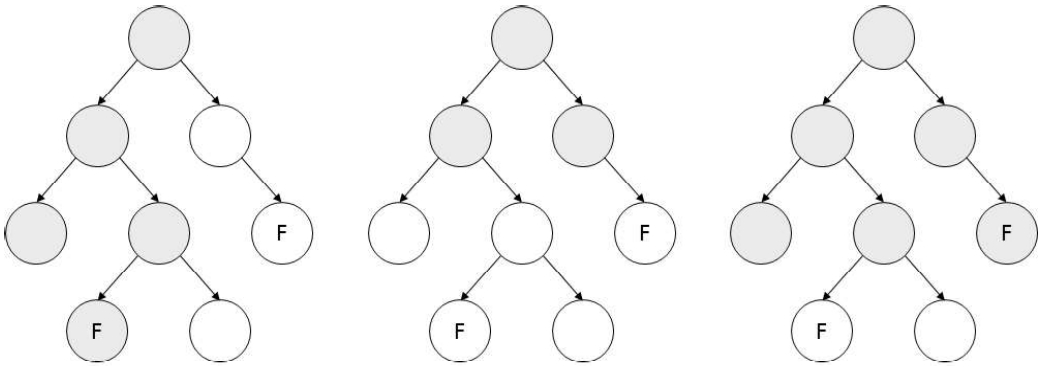
บทที่ 2 กล่าวถึงงานวิจัยที่เกี่ยวข้องกับขั้นตอนวิธี
บทที่ 3 กล่าวถึงการออกแบบขั้นตอนวิธี
บทที่ 4 กล่าวถึงผลการวิจัย
บทที่ 5 กล่าวถึงข้อสรุป และข้อเสนอแนะ

บทที่ 2
งานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงงานวิจัยที่เกี่ยวข้องกับขั้นตอนวิธีสำหรับหาคำตอบที่ดีที่สุดในปริศนาที่มีการเลื่อนเป็นลำดับ

2.1 การค้นหาในแนวลึก

การค้นหาในแนวลึก (Depth-first Search) ถูกใช้ในการแก้ไขปัญหานี้เป็นวิธีการแรกๆ โดยการค้นหาในแนวลึกจะค้นหาจากรูปแบบปัจจุบันและเน้นไปที่การค้นหารูปแบบถัดไปจนกว่าจะไม่มีเส้นต่อไปจึงจะมีการย้อนกลับไปค้นหาจากจุดก่อนหน้าจนพบจุดที่ต้องการ ซึ่งขั้นตอนวิธีนี้ใช้หน่วยความจำน้อยมาก แต่อาจไม่ให้คำตอบที่ดีที่สุดได้ จึงมีการปรับปรุงให้มีการจำกัดความลึกสูงสุดไว้ เมื่อค้นหาจนครบแล้วยังไม่พบคำตอบจึงจะเพิ่มความลึกสูงสุดอีกหนึ่งชั้น การปรับปรุงนี้จะทำให้จุดคำตอบที่พบเป็นจุดที่ห่างจากจุดเริ่มต้นค้นหาน้อยที่สุดเสมอ

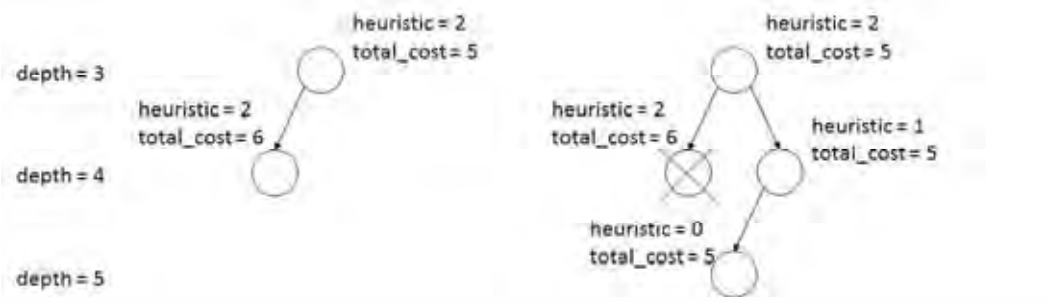


ภาพที่ 2.1 ผลการค้นหาในแนวลึก

ภาพที่ 2.1 แสดงผลการค้นหาแบบในแนวลึกในรูปแบบต่างๆ โดยต้นไม้ที่แสดงถึงลำดับการค้นหานี้เริ่มต้นหาจากรากของต้นไม้การค้นหาซึ่งจะแสดงอยู่บนสุดของต้นไม้ในรูป ค้นหาไปทางที่มีความลึกมากขึ้น หากไม่มีทางค้นหาต่อจึงย้อนกลับมาค้นหาจากจุดก่อนหน้าและจบที่จุด F ซึ่งใช้แทนจุดสิ้นสุดการค้นหา โดยหากมีหลายเส้นทางจะค้นหาจากซ้ายไปขวา ภาพซ้ายแสดงในกรณีที่ไม่มีการจำกัดความลึก จะเห็นได้ว่าจุดที่สิ้นสุดไม่ได้เป็นจุดที่ใกล้เริ่มต้นที่สุด นั่นคือไม่ได้เป็นคำตอบที่ดีที่สุด และภาพกลางแสดงในกรณีที่มีการจำกัดความลึกที่สอง ซึ่งไม่พบจุดไม่ต้องการค้นหา จึงนำมาสู่ภาพขวาซึ่งขยายความลึกสูงสุดเป็นสาม และพบจุดที่เป็นจุดสิ้นสุดซึ่งเป็นคำตอบที่ดีที่สุด

ขั้นตอนวิธีนี้จะให้คำตอบที่ดีที่สุดเสมอ และไม่ทำให้ความซับซ้อนของเวลาที่ใช้ในการคำนวณเพิ่มขึ้น แต่ขั้นตอนวิธีนี้ไม่ได้มีการออกแบบให้สามารถค้นหาคำตอบทั้งหมด จึงต้องใช้ขั้นตอนวิธีนี้ใหม่ทุกครั้งที่ต้องการหาคำตอบจากรูปแบบใดๆ และการค้นหานี้จะไม่ได้เก็บข้อมูลจุดที่เคยค้นหาไปแล้ว เนื่องจากขั้นตอนวิธีนี้สำหรับใช้กับคอมพิวเตอร์ที่ไม่มีหน่วยความจำขนาดใหญ่ จึงทำให้เกิดการต้นหาซ้ำหลายครั้ง

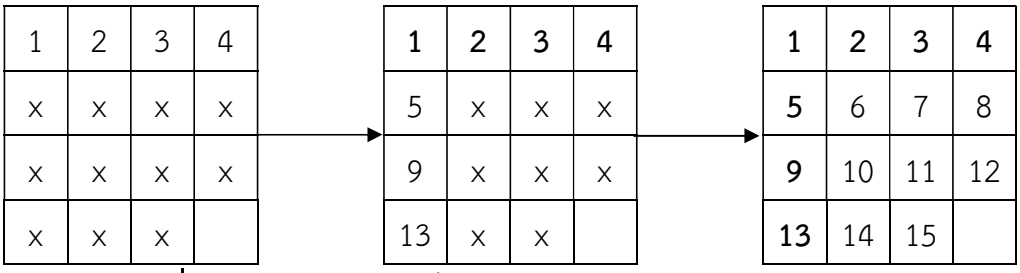
ขั้นตอนวิธีแบบเอสตาร์โดยจำกัดความลึก (Iterative-Deepening A* / IDA*) (Reinefeld, 1993) เคยถูกใช้การหาคำตอบทั้งหมดของปริศนาเรียงแปดเลข ซึ่งขั้นตอนวิธีนี้สามารถตัดกิ่งของต้นไม้ที่ไม่สามารถให้คำตอบได้ (Pruning) จึงทำให้การค้นหาคำตอบทั้งหมดนั้นมีประสิทธิภาพเพิ่มขึ้น โดยการใช้ฟังก์ชันฮิวริสติก (Heuristic) ในการประมาณความยาวเส้นทางในการไปสู่คำตอบที่ดีที่สุด หากเส้นทางที่เลือกนั้นใช้เส้นทางยาวกว่าที่จำกัดไว้ให้ยกเลิกการค้นหาในเส้นทางนั้น ดังภาพที่ 2.2



ภาพที่ 2.2 การตัดกิ่งของต้นไม้ของขั้นตอนวิธี IDA* เมื่อจำกัดความลึกที่ห้า

2.2 การแบ่งปัญหาเป็นปัญหาย่อย

กลุ่มปัญหาเอ็นพีบริบูรณ์ (NP-Complete) มีคุณสมบัติเหมือนกันอย่างหนึ่งคือความซับซ้อนของปัญหาที่เป็นฟังก์ชันเอกซ์โพเนนเชียล ทำให้เมื่อปัญหาใหญ่ขึ้นเล็กน้อยจะทำให้เวลาที่ใช้ประมวลผลมากขึ้นหลายเท่า ซึ่งทำให้การแก้ปัญหาที่ใหญ่นั้นใช้เวลานานเกินกว่าจะเป็นไปได้ในความเป็นจริง จึงมีงานวิจัยที่พยายามแก้ไขปัญหานี้ แต่อาจทำให้มีเงื่อนไขบางอย่างหรือไม่สามารถทำตามเงื่อนไขบางอย่างได้

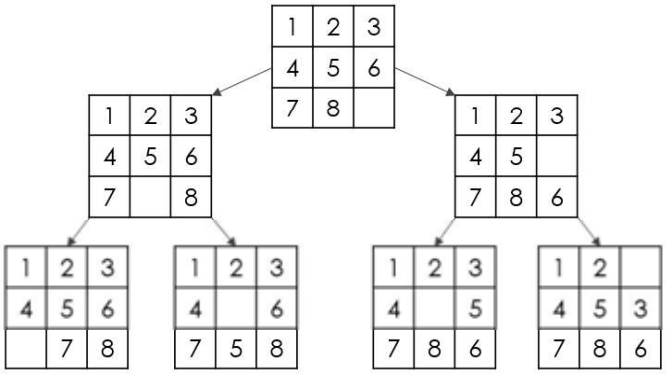


ภาพที่ 2.3 ตัวอย่างการแบ่งเป็นปัญหาย่อยของปริศนาการเรียงสิบห้าเลข

ภาพที่ 2.3 แสดงรูปแบบหนึ่งของการแบ่งเป็นปัญหาย่อย โดยการแก้ปริศนาในส่วนของเลขหนึ่งถึงสี่ให้อยู่ในตำแหน่งที่ถูกต้อง จากนั้นจึงแก้ปริศนาในส่วนของเลขห้า เก้า และสิบสามโดยไม่ย้ายตำแหน่งของเลขหนึ่งถึงสี่ สุดท้ายจึงเรียงเลขส่วนที่เหลือในตำแหน่งที่ถูกต้อง โดยมีความซับซ้อนในการแก้ปริศนาส่วนนี้เท่ากับปริศนาการเรียงเก้าเลข ซึ่งเป็นโจทย์เล็กกว่าจึงทำให้ง่ายต่อการหาคำตอบเป็นอย่างมาก แต่คำตอบที่ได้อาจจะไม่ได้เป็นคำตอบที่ดีที่สุดเสมอ

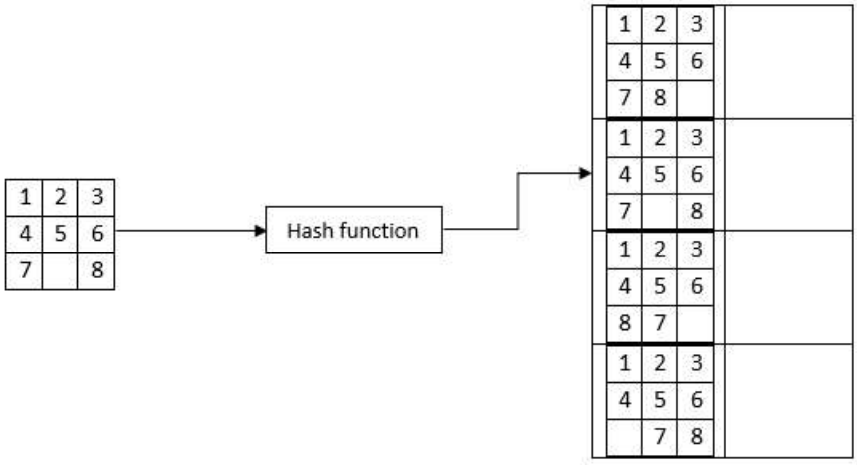
2.3 การค้นหาในแนวกว้าง

การค้นหาในแนวกว้าง (Breath-first Search) เป็นหนึ่งในการค้นหาในกราฟที่พื้นฐานที่สุดแบบหนึ่ง โดยการสร้างคิวสำหรับเก็บจุดที่จะมีค้นหาถัดไปและเก็บจุดที่จะมีการค้นหาต่อไปไว้ท้ายของคิวนั้น ขั้นตอนวิธีล่าสุดใช้หลักการการค้นหาของแนวกว้างแบบลำดับบนหน่วยประมวลผลกลาง โดยเริ่มค้นหาจากรูปแบบเริ่มต้นไปจนครบทุกรูปแบบ และเก็บข้อมูลลงในโครงสร้างข้อมูลสำหรับค้นหารูปแบบว่ามาจากตำแหน่งใด หรือมาด้วยการเคลื่อนแบบใด ซึ่งการเคลื่อนที่นั้นดูจากกรณีที่จะสามารถเคลื่อนที่กลับมาจากรูปแบบนั้นๆแทนที่จะเคลื่อนที่ไป ดังภาพที่ 2.4 และหลักการนี้ได้ถูกใช้ในงานวิจัยหลายงานดังต่อไปนี้



ภาพที่ 2.4 ตัวอย่างการค้นหาแนวกว้าง

ตารางแฮชได้นำมาใช้เพื่อเก็บคำตอบสำหรับทุกรูปแบบ (Parberry, 2015) โดยเหมาะกับการใช้เก็บคำตอบสำหรับปัญหาที่มีขนาดเล็กได้แต่สำหรับปัญหาที่มีขนาดใหญ่ขึ้น การใช้ตารางแฮชจะต้องใช้หน่วยความจำเพิ่มขึ้นในการเก็บข้อมูลรูปแบบของปริศนานั้นเพื่อแก้ปัญหาการชนกันของค่าแฮช ซึ่งใช้ในการยืนยันว่าค่าในตารางเป็นของรูปแบบนั้นๆจริง ดังภาพที่ 2.5



ภาพที่ 2.5 การเก็บข้อมูลในตารางแฮช

การเก็บแบบแถวลำดับสำหรับเก็บคำตอบ (Wang & Song, 2016) ซึ่งจำเป็นต้องมีฟังก์ชันที่สามารถกำหนดรูปแบบทั้งหมดของปริศนานั้นโดยที่มั่นใจว่าค่าที่ได้ในแต่ละรูปแบบจะต้องไม่ซ้ำกัน หรือเรียกอีกอย่างว่าฟังก์ชันหนึ่งต่อหนึ่ง โดยมีการเก็บข้อมูลรูปแบบเพิ่มเติมในแถวลำดับนั้นเพื่อใช้ดูว่าในช่องนั้นๆเก็บค่าของรูปแบบใดไว้ดังภาพที่ 2.6 แต่กลับไม่ได้ใช้คุณสมบัติของฟังก์ชันหนึ่งต่อหนึ่งซึ่งสามารถหาอินเวอร์สได้ โดยการหารูปแบบจากตำแหน่งของช่องที่ใช้เก็บข้อมูลนั้น ซึ่งจะทำให้ประหยัดหน่วยความจำในการเก็บข้อมูลได้

sequence number	hash value (state)	sequence number of previous state
1	1 (123 456 789)	-1
	.....	...
4	199 (123 586 479)	1
	.....	...
15	7543 (136 528 479)	4
	.....	...
58	87877 (326 158 479)	15
	.....	...
228	57733 (256 318 479)	58
	.....	...
885	64249 (268 351 479)	228
	.....	...
346643	362880 (987 654 321)	270670
	.....	...
362880	247116 (721 456 983)	362866

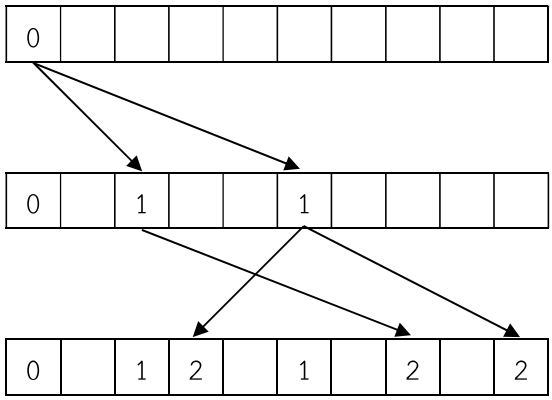
ภาพที่ 2.6 ตัวอย่างของตารางคำตอบในตัวแปรแถวลำดับ

การเก็บแบบแถวลำดับร่วมกับการแบ่งปัญหาเป็นปัญหาย่อยสำหรับปริศนาการเรียงเลขขนาดใหญ่ (Wang, 2017) โดยงานวิจัยนี้ประยุกต์ใช้หลักการที่ว่า มนุษย์คิดวิธีแก้ปัญหาได้และคอมพิวเตอร์สามารถแก้ปัญหาได้เร็ว จึงใช้หลักการแก้ปริศนาของมนุษย์ ซึ่งส่วนใหญ่จะเลือกแก้ปริศนาทีละส่วน งานวิจัยนี้จึงออกแบบให้คอมพิวเตอร์ย้ายชิ้นเลขที่เกี่ยวข้องมารวมกันในขนาด 2x3 แล้วใช้วิธีการแก้ปริศนาขนาด 2x3 ในการแก้ส่วนย่อยนี้ จากนั้นจึงค่อยๆแก้ส่วนที่เหลือด้วยหลักการนี้จนครบทั้งหมด ซึ่งงานวิจัยนี้สามารถแก้ปริศนาขนาด 20x20 ได้ในเวลาอันสั้น และคำตอบที่ได้อยู่ในระดับนำไปใช้จริงได้

2.4 การค้นหาแนวกว้างแบบขนาน

งานวิจัยในหัวข้อก่อนหน้าเป็นการค้นหาแนวกว้างแบบลำดับ ซึ่งถูกแสดงให้เห็นว่าช้ากว่าการค้นหาแบบขนานอย่างมากสำหรับปัญหาที่มีความซับซ้อนสูง (Elnaggar, Gadallah, Aziem, & Eldeeb, 2014) และการประมวลผลแบบขนานกำลังเป็นที่นิยมในปัจจุบัน จึงมีงานวิจัยที่เกี่ยวข้องกันการค้นหาแนวกว้างแบบขนานมากมาย โดยงานวิจัยนี้จะเน้นไปที่การทำงานบนหน่วยประมวลผลกราฟิกโดยใช้ CUDA เป็นชุดคำสั่งที่ใช้ในหน่วยประมวลผลกราฟิกของ Nvidia แต่การออกแบบขั้นตอนวิธีสำหรับทำงานในหน่วยประมวลผลกราฟิกนั้นมีข้อจำกัดและจุดที่ควรระวังอยู่มากมาย[1] จึงมีงานวิจัยที่พยายามออกแบบขั้นตอนวิธีเพื่อหลีกเลี่ยงข้อจำกัดดังต่อไปนี้

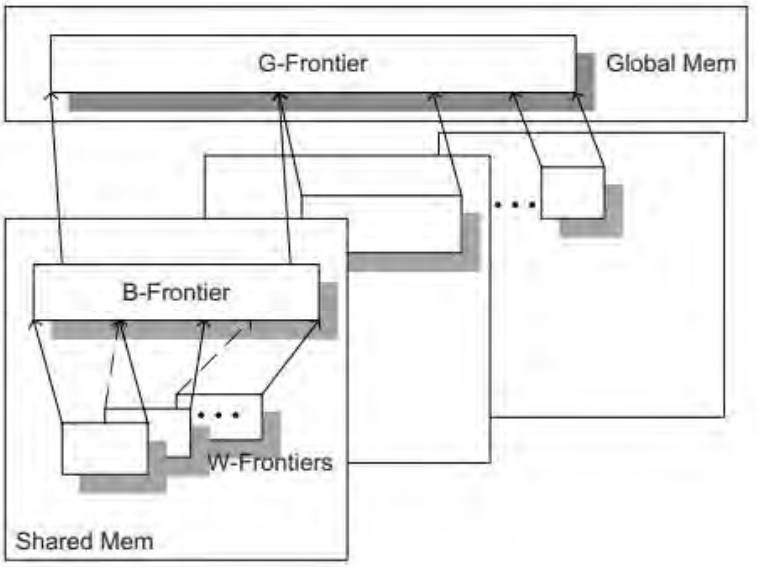
งานวิจัยแรกๆซึ่งเป็นจุดเริ่มต้นของขั้นตอนวิธีค้นหาของกราฟแบบขนานบนหน่วยประมวลผลกราฟิก (Harish & Narayanan, 2007) โดยใช้หลักการการกระจายทุกเธรดให้ประมวลผลทุกรูปแบบพร้อมกันทั้งที่คำนวณได้และไม่ได้รอบละหนึ่งความลึก โดยจะมีการคำนวณเฉพาะช่องที่มีความลึกตรงกับความลึกปัจจุบันเท่านั้น และเก็บทุกรูปแบบซึ่งจะถูกค้นหาในความลึกถัดไปในตัวแปรแบบแถวลำดับเดิมดังภาพที่ 2.7 แต่การค้นหาทุกจุดยอดในตัวแปรแถวลำดับจำนวนหลายครั้งขึ้นอยู่กับความลึกของคำตอบ ทำให้ความซับซ้อนในการประมวลผลอยู่ที่ $O(V^2+E)$ ซึ่งมากกว่าขั้นตอนวิธีแบบลำดับซึ่งคือ $O(V+E)$ และเนื่องด้วยข้อจำกัดของการประมวลผลแบบขนาน คือปัญหาเรซคอนดิชัน (Race Condition) ซึ่งทำให้การเก็บข้อมูลลงตำแหน่งเดียวกันของหลายเธรดผิดปกติได้ และข้อจำกัดของหน่วยประมวลผลกราฟิกที่มีเธรดมากแต่มีหน่วยความจำกลางเพียงหนึ่งเดียว การออกแบบขั้นตอนวิธีที่มีประสิทธิภาพจึงทำได้ค่อนข้างยาก และงานวิจัยนี้เป็นรากฐานให้กับงานวิจัยที่เกี่ยวข้องกับการค้นหาในกราฟแบบขนานอื่นๆที่มีประสิทธิภาพมากขึ้นอีกหลายงานวิจัย



ภาพที่ 2.7 ตัวอย่างข้อมูลในแถวลำดับหลังจากการคำนวณแต่ละความลึก

[1] ดู ภาคผนวก ข

ขั้นตอนวิธีค้นหาแนวกว้างแบบขนานโดยสร้างคิวสำหรับแต่ละความลึกซึ่งแก้ไขปัญหาดังกล่าวได้และมีประสิทธิภาพที่สูงมากพอ (Luo, Wong, & Hwu, 2010) โดยสร้างคิวแยกสำหรับแต่ละกลุ่มเธรดดังภาพที่ 2.8 และค่อยมารวมกันทีละมากๆ ซึ่งจะทำให้การคัดลอกข้อมูลมีประสิทธิภาพสูงขึ้น และป้องกันการเกิดเรซคอนดิชันได้ โดยมีความซับซ้อนในการประมวลผลเท่ากับแบบลำดับและให้ความเร็วเพิ่มขึ้นมากถึงแปดเท่า



ภาพที่ 2.8 คิวที่แยกสำหรับแต่ละกลุ่มของเธรด

ภาพที่ 2.8 แสดงรูปแบบการแบ่งเป็นคิวย่อยของงานวิจัยนี้ โดยจะสร้างคิวหลัก (G-Frontier) ในหน่วยความจำหลัก (Global Memory) ซึ่งเหมือนกับคิวของขั้นตอนวิธีแบบลำดับทั่วไป แต่หน่วยความจำหลักของหน่วยประมวลผลกราฟิกนั้นเข้าถึงได้ช้า เมื่อเธรดจำนวนมากต้องการเก็บข้อมูลลงท้ายคิวพร้อมกันจะทำให้การเข้าถึงที่ช้านั้นมีปัญหามากขึ้นอย่างมาก จึงออกแบบให้มีคิวย่อยสำหรับเธรดกลุ่มหนึ่งซึ่งอยู่ในบล็อกเดียวกัน ทำให้มีหน่วยความจำย่อยสำหรับแต่ละบล็อก (Shared Memory) ที่เธรดกลุ่มนี้สามารถเข้าถึงได้เหมือนกัน แล้วสร้างคิวสำหรับใช้ร่วมกันไว้ (B-Frontier) แต่ก็ยังไม่สามารถแก้ไขปัญหาเมื่อเธรดกลุ่มนี้ต้องการเข้าถึงคิวนี้พร้อมกัน จึงออกแบบให้มีคิวระดับสุดท้าย (W-Frontier) สำหรับเธรดบางส่วนของกลุ่มนั้นซึ่งไม่สามารถทำงานพร้อมกันได้ เนื่องจากการออกแบบของหน่วยประมวลผลกราฟิกที่จะให้เธรดในบล็อกมีการทำงานพร้อมกันเป็นชุดๆ (Warp) จึงให้เธรดแต่ละตัวใน Warp นั้นสามารถเข้าถึงคิวต่างกัน ดังนั้นจึงไม่เกิดปัญหาเรซคอนดิชันขึ้น แล้วนำคิวย่อยมารวมกันจนได้คิวหลัก โดยการรวมคิวจะต้องมีการคัดลอกข้อมูลจำนวนมากแต่หน่วยประมวลผลกราฟิกจะสามารถคัดลอกข้อมูลที่อยู่ในช่องเก็บข้อมูลที่ต่อเนื่องกันได้อย่างมีประสิทธิภาพมาก (Memory Coalescing) และโดยการกำหนดจุดเริ่มต้นของคิวย่อยในคิวที่ใหญ่กว่าทำให้สามารถทำการคัดลอกข้อมูลพร้อมกันได้อีกด้วย

บทที่ 3
การออกแบบขั้นตอนวิธี

3.1 บทนำ

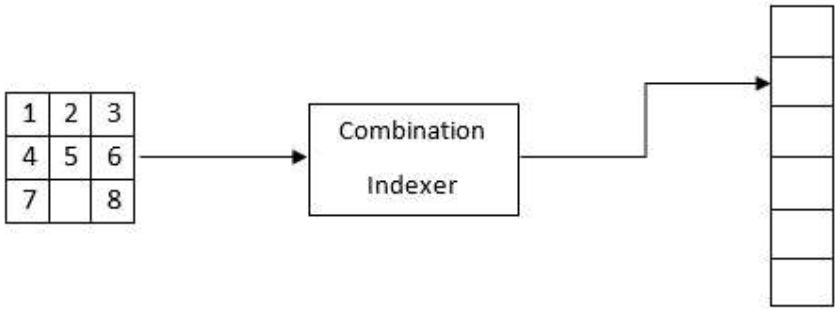
จากปัญหาที่พบในงานวิจัยก่อนๆ การออกแบบขั้นตอนวิธีที่แก้ปัญหาดังกล่าวได้ควรคุณสมบัติดังนี้

1. เป็นฟังก์ชันที่สามารถแปลงรูปแบบใดๆให้เป็นเลขชี้ตำแหน่งได้แบบหนึ่งต่อหนึ่ง และสามารถสร้างฟังก์ชันอินเวอร์สได้อย่างมีประสิทธิภาพ
2. สามารถแก้ไขปริศนาที่เป็นส่วนย่อยได้ เพื่อให้สามารถแก้ปริศนาที่มีขนาดใหญ่ขึ้นได้ โดยอาจจะไม่ได้คำตอบที่ดีที่สุด แต่ยังสามารถนำไปใช้จริงได้
3. สามารถประยุกต์ใช้ได้กับปริศนาที่มีการเลื่อนเป็นลำดับส่วนใหญ่ได้ เนื่องจากมีลักษณะพื้นฐานเหมือนกัน

ขั้นตอนวิธีที่ถูกสร้างขึ้นนี้จะประกอบด้วยสองส่วนคือ ส่วนกำหนดช่องสำหรับเก็บข้อมูล และส่วนค้นหาในกราฟแบบขนาน ซึ่งจะถูกกล่าวถึงในรายละเอียดต่อไป

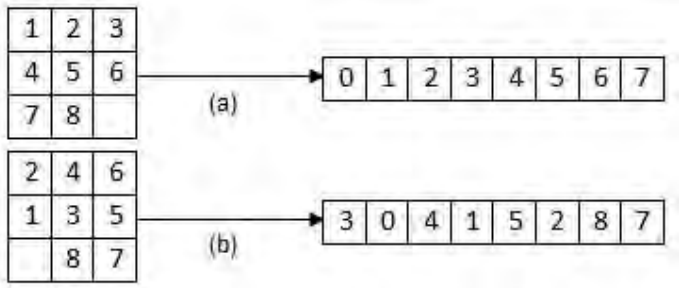
3.2 ส่วนกำหนดช่องสำหรับเก็บข้อมูล

ขั้นตอนวิธีสำหรับกำหนดช่องสำหรับเก็บข้อมูล (Combination Indexer) จะต้องสามารถแปลงรูปแบบใดๆของปริศนาที่สนใจ ให้เป็นจำนวนเต็มได้โดยไม่มีเลขซ้ำกันสำหรับรูปแบบที่แตกต่างกัน รวมถึงต้องสามารถแปลงกลับจากจำนวนเต็มนั้นให้เป็นรูปแบบเดิมได้อย่างมีประสิทธิภาพด้วย เพื่อประโยชน์ในการคาดการณ์รูปแบบถัดจากช่องเก็บข้อมูลที่สนใจ เนื่องจากไม่มีการเก็บข้อมูลรูปแบบที่เป็นเจ้าของข้อมูลที่ช่องนั้นๆ โดยการกระทำเช่นนี้จะทำให้สามารถให้หน่วยความจำให้น้อยที่สุด แต่ยังคงความสามารถในการค้นหาคำตอบได้เช่นเดิม ดังภาพที่ 3.1



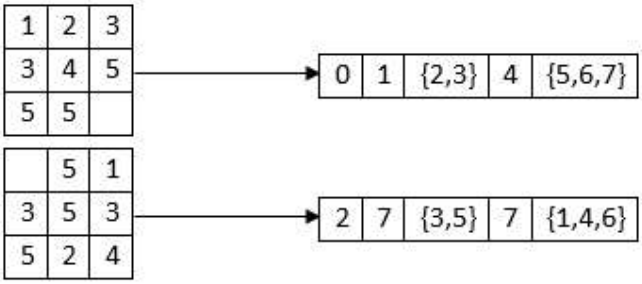
ภาพที่ 3.1 การกำหนดช่องสำหรับเก็บข้อมูล

สำหรับรูปแบบใดๆของปริศนาจะอยู่ในรูปของตำแหน่งของชิ้นส่วนทุกชิ้นเรียงอยู่ในตัวแปรแถวลำดับ โดยเริ่มนับให้ตำแหน่งแรกเป็นตำแหน่งที่ศูนย์ ในกรณีที่มีชิ้นส่วนที่เหมือนกันและสลับตำแหน่งกันได้ ให้แสดงอยู่รูปของเซต โดยการสร้างตัวเก็บข้อมูลเซต ให้สร้างเป็นตัวแปรแถวลำดับและให้ข้อมูลตำแหน่งเรียงจากน้อยไปมาก ซึ่งจะทำให้รูปแบบที่มีชิ้นส่วนเหมือนกัน มีตัวแปรตัวลำดับที่แสดงถึงรูปแบบนั้นๆเหมือนกัน



ภาพที่ 3.2 ตัวอย่างรูปแบบของปริศนาการเรียงแปดเลข

ในภาพที่ 3.2 จะแสดงตัวอย่างรูปแบบของปริศนาการเรียงแปดเลขที่ถูกแทนด้วยตัวแปรแถวลำดับ ในตัวอย่าง (a) จะแสดงรูปแบบเริ่มต้นของปริศนาการเรียงแปดเลข โดยช่องที่ศูนย์ของตัวแปรแถวลำดับแสดงตำแหน่งของชิ้นส่วนที่มีเลขหนึ่ง ซึ่งคือตำแหน่งที่ศูนย์ในปริศนาการเรียงแปดเลข และช่องถัดๆไปก็แสดงเลขที่อยู่ถัดจะนั้นตามลำดับ ในตัวอย่าง (b) จะแสดงปริศนาการเรียงแปดเลขที่ถูกสลับตำแหน่งแล้ว โดยช่องที่ศูนย์ของตัวแปรแถวลำดับแสดงตำแหน่งของชิ้นส่วนที่มีเลขหนึ่ง ซึ่งคือตำแหน่งที่สามในปริศนาการเรียงแปดเลข หรือตำแหน่งแถวกลางหลักซ้ายของปริศนานั้น



ภาพที่ 3.3 ตัวอย่างรูปแบบของปริศนาการเรียงเลขที่มีเลขซ้ำ

ในภาพที่ 3.3 แสดงตัวอย่างรูปแบบของปริศนาการเรียงเลขที่มีเลขซ้ำ ซึ่งแทนด้วยตัวแปรแถวลำดับ สังเกตว่ามีชิ้นส่วนที่มีเลขสามซ้ำกันสองชิ้นและชิ้นส่วนที่มีเลขห้าซ้ำกันสามชิ้น โดยแสดงตำแหน่งด้วยค่าในตัวแปรแถวลำดับช่องที่สองและสี่ตามลำดับ โดยตำแหน่งของชิ้นส่วนที่ซ้ำกันจะอยู่ในรูปช่องเซต ซึ่งอาจแสดงด้วยตัวแปรแถวลำดับที่มีการเรียงของเลขจากน้อยไปมาก ในการนำไปใช้จริง อาจมีสลับตำแหน่งที่ใช้แสดงตำแหน่งของชิ้นส่วนนั้นๆได้ เพียงแต่ต้องทำให้เหมือนกันระหว่างขั้นตอนวิธีที่ใช้สร้างคำตอบและส่วนที่ได้คำตอบ

ขั้นตอนวิธีที่ใช้แปลงจากตัวแปรแถวลำดับเป็นจำนวนเต็มและส่วนที่ใช้แปลงกลับจำเป็นต้องเขียนให้จำเพาะต่อปริศนาที่ต้องการหาคำตอบ เนื่องจากข้อจำกัดของการใช้ในหน่วยประมวลผลกราฟิกที่ไม่สามารถสร้างตัวแปรเพิ่มได้[2] จึงต้องมีการระบุจำนวนตัวแปรให้คงที่สำหรับปริศนานั้นๆ โดยโครงสร้างของขั้นตอนวิธีนี้จะประกอบด้วยสามส่วนได้แก่ ส่วนกำหนดตำแหน่งของชิ้นส่วนที่ไม่สัมพันธ์กัน ส่วนกำหนดตำแหน่งของชิ้นส่วนที่สัมพันธ์กันและส่วนกำหนดตำแหน่งของชิ้นส่วนที่เหมือนกัน

3.2.1 ส่วนกำหนดตำแหน่งของชิ้นส่วนที่ไม่สัมพันธ์กัน

ชิ้นส่วนที่ไม่สัมพันธ์กันคือตำแหน่งของกลุ่มชิ้นส่วนนั้นๆจะไม่เกี่ยวข้องกับตำแหน่งของกลุ่มชิ้นส่วนอื่น ตัวอย่างเช่น ในปริศนาลูกบาศก์ของรูบิกมีการหมุนของชิ้นส่วนที่เป็นมุมของลูกบาศก์ และการหมุนของชิ้นส่วนตามขอบ ซึ่งสังเกตว่าการหมุนของชิ้นส่วนเหล่านี้จะไม่เกี่ยวข้องกับชิ้นส่วนอื่นๆ เลย รวมถึงไม่เกี่ยวข้องกันเองอีกด้วย โดยชิ้นส่วนเหล่านี้จะถือว่าเป็นกลุ่มของชิ้นส่วนที่ไม่สัมพันธ์กันและมีชิ้นส่วนอยู่เพียบหนึ่งชิ้นในกลุ่มนั้นๆ

การกำหนดจำนวนเต็มให้กับรูปแบบของปริศนาใดๆนั้น จะเริ่มที่การกำหนดตำแหน่งของชิ้นส่วนที่ไม่สัมพันธ์ ซึ่งจะหาได้จากการนำจำนวนเต็มที่ใช้แทนกลุ่มชิ้นส่วนที่สัมพันธ์กัน ซึ่งจะกล่าววิธีการหาค่านั้นในหัวข้อต่อไป มารวมกันโดยใช้หลักการของเลขฐาน โดยเลขฐานที่ใช้นั้นจะมีเลขฐานไม่เท่ากันในแต่ละหลัก และให้ค่าในแต่ละหลักแทนกลุ่มชิ้นส่วนที่สัมพันธ์กันนั้นๆ โดยเลขฐานที่ใช้จะในหลักนั้นจะแทนด้วยค่าสูงสุดที่เป็นไปได้ของกลุ่มชิ้นส่วนนั้นๆ บวกกับอีกหนึ่ง สังเกตว่าค่าที่เป็นไปได้ของกลุ่มชิ้นส่วนนั้นๆจะอยู่ระหว่างศูนย์ถึงค่าสูงสุดนั้นด้วย

ให้ง่ายต่อการอธิบาย จะมีการกำหนดตัวแปรดังนี้

และสำหรับการหาค่าย้อนกลับสามารถหาได้จากความสัมพันธ์เวียนเกิดหรือรหัสเทียมต่อไปนี้

ในส่วนของขั้นตอนวิธีนี้ จะมีการใช้สัมประสิทธิ์ทวินาม (Binomial Coefficient) ในการคำนวณค่าด้วย จึงสามารถคำนวณค่านี้ไว้ก่อนได้เพื่อให้มีประสิทธิภาพของขั้นตอนวิธีนี้สูงที่สุด โดยสามารถคำนวณอย่างมีประสิทธิภาพจากเอกลักษณ์ของปาสคาล (Pascal

การกำหนดตำแหน่งของชิ้นส่วนที่เหมือนกัน จะหาได้ด้วยหลักการของเลขฐานการจัดกลุ่ม (Combination Number System) เนื่องจากเป็นการกำหนดตำแหน่งให้กับชิ้นส่วนที่เรียงอยู่ในช่องที่สามารถวางได้ โดยตำแหน่งของชิ้นส่วนเหล่านี้จะมีการเรียงลำดับแล้ว ซึ่งตรงกับหลักการของเลขฐานนี้ด้วย หลักการนี้มักใช้ในการลำดับรูปแบบการเลือกตำแหน่งให้เป็นจำนวนเต็ม แต่สำหรับการเปลี่ยนจากจำนวนเต็มให้กลับเป็นรูปแบบการเลือกตำแหน่งนั้นยังไม่มีขั้นตอนวิธีที่มีประสิทธิภาพมากพอ ในงานวิจัยนี้สามารถใช้ขั้นตอนนั้นได้ เนื่องจากขั้นตอนวิธีจากหัวข้อก่อนหน้าต้องนำมาใช้รวมกับขั้นตอนในหัวข้อนี้ ซึ่งมีความซับซ้อนในการประมวลผลไม่ต่างกัน

ให้ง่ายต่อการอธิบาย จะมีการกำหนดตัวแปรดังนี้

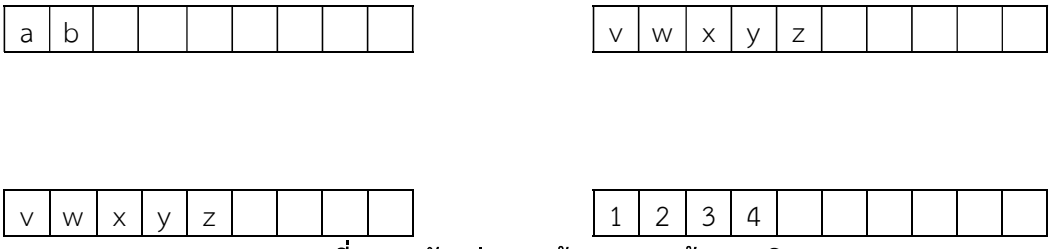
```
Let index be the current configuration for this thread
if index 's depth is current depth
        find the configuration for index
        for every next configuration
                find next index for next configuration
                if next index's depth is not set
                        set next index's depth
                                to be current depth+1
                        set next index's move
```

ภาพที่ 3.10 รหัสเทียมสำหรับการค้นหาจากจุดที่เป็นแนวหน้าของหน่วยประมวลผลกราฟิก

```
For every initial configuration
        set depth for the configuration to be 0
current depth = 0
While depth is changed
        run kernel for every vertex at current depth
        synchronize the kernel
```

ภาพที่ 3.11 รหัสเทียมสำหรับการค้นหาจากจุดที่เป็นแนวหน้าของหน่วยประมวลผลกลาง

ความซับซ้อนเชิงเวลาของขั้นตอนวิธีนี้คือ



ภาพที่ 3.12 ตัวอย่างการค้นหาแนวกว้างจากคิว

```
Let index be the current configuration for this thread
find the configuration for index
for every next configuration
        find next index for next          configuration
        if next inde    x's move  is not set
                enqueue next index with atomic instruction
                        in second queue
                set next index's move
```

ภาพที่ 3.13 รหัสเทียมสำหรับการค้นหาจากคิวของแนวหน้าของหน่วยประมวลผลกราฟิก

```
For every initial configuration
        enqueue the configuration's index  in first queue
While   first queue      is   not empty
        run kernel for every vertex          in   the   first queue
        synchronize the kernel
        first que     ue = second queue
        empty   the   second queue
```

ภาพที่ 3.14 รหัสเทียมสำหรับการค้นหาจากคิวของแนวหน้าของหน่วยประมวลผลกลาง

ความซับซ้อนเชิงเวลาของขั้นตอนวิธีนี้คือ

3.3.3 การสร้างคิวแบบหลายระดับการเข้าถึง

การสร้างคิวแบบหลายระดับการเข้าถึง (Luo, Wong, & Hwu, 2010) สามารถเพิ่มประสิทธิภาพการเข้าถึงข้อมูลและการแก้ไขข้อมูลได้ โดยการสร้างคิวในหลายระดับการเข้าถึงข้อมูลดังภาพที่ 2.8 เนื่องจากในหน่วยประมวลผลกราฟิกของ Nvidia จะมีหน่วยความจำหลายระดับซึ่งมีความเร็วการอ่านเขียนข้อมูลไม่เท่ากัน[4] แต่ด้วยขีดจำกัดของหน่วยความจำระดับบล็อกที่ต่ำจึงไม่อาจนำขั้นตอนวิธีนี้มาใช้ในงานวิจัยนี้ที่อาจจะต้องใช้หน่วยความจำระดับบล็อกที่มากได้

ตารางที่ 3.4 ตารางเปรียบเทียบประสิทธิภาพขั้นตอนวิธีค้นหาในกราฟแสดงข้อดี-ข้อเสียของขั้นตอนวิธีที่นำเสนอในงานวิจัยวิธีนี้ ซึ่งสำหรับเป้าหมายของงานวิจัยนี้ซึ่งจะออกแบบขั้นตอนวิธีที่ใช้หน่วยความจำให้น้อยที่สุด ขั้นตอนวิธีค้นหาจากจุดแนวหน้าซึ่งใช้หน่วยความจำน้อยจึงอาจจะเป็นเป้าหมายของงานวิจัยนี้ แต่ด้วยความซับซ้อนเชิงเวลาของขั้นตอนวิธีที่มากเกินไปจะทำให้การทำงานช้าเกินไปอย่างมาก งานวิจัยนี้จึงเลือกใช้ขั้นตอนวิธีค้นหาจากคิวของจุดแนวหน้าเป็นหลัก

ขั้นตอนวิธี	ความซับซ้อนเชิงเวลา	หน่วยความจำ (ไบต์ต่อรูปแบบ)
ค้นหาจากจุดแนวหน้า		

บทที่ 4
ผลการวิจัย

ในบทนี้จะกล่าวถึง ผลของการทดลองขั้นตอนวิธีที่นำเสนอในงานวิจัยนี้ และนำไปเปรียบเทียบประสิทธิภาพของขั้นตอนวิธีที่เกี่ยวข้องในบทที่ 2 โดยมีตัวชี้วัดดังนี้

1. เวลาที่ใช้หาคำตอบทั้งหมด
2. ปริมาณหน่วยความจำที่ใช้
3. จำนวนการเปลี่ยนรูปแบบเฉลี่ย

ส่วนประกอบในคอมพิวเตอร์ที่จะใช้ในการทดสอบนี้ได้แก่

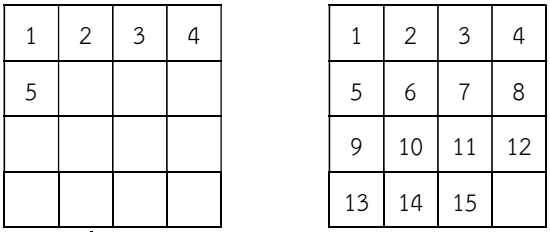
1. หน่วยประมวลผลกลาง Intel® Core

4.2 ผลของการแก้ปริศนาการเรียงสิบห้าเลข

ปริศนาการเรียงสิบหกเลขเป็นปัญหาที่ยากขึ้นกว่าปริศนาเรียงแปดเลขอย่างมาก เนื่องจากรูปแบบทั้งหมดที่มีถึง

ขั้นตอนวิธี	ขั้นตอนย่อย	เวลาที่ใช้เฉลี่ยต่อรูปแบบ	หน่วยความจำที่ใช้	จำนวนการเปลี่ยนรูปแบบสูงสุด
IDA* (Reinefeld, 1993)	1-2-3-4	757.9 ms	1,922 bytes	46
	5-9-13	7.953 ms	1,362 bytes	32
	8-Puzzle	0.029 ms	1,322 bytes	31

ตารางที่ 4.3 ตารางแสดงผลการประมาณค่าขั้นตอนวิธี IDA*



ภาพที่ 4.3 ลำดับการแก้ปัญหาย่อยสองปัญหา

ขั้นตอนวิธี	ขั้นตอนย่อย	เวลาที่ใช้	หน่วยความจำที่ใช้	จำนวนการเปลี่ยนรูปแบบเฉลี่ย
BFS (Parberry, 2015)	1-2-3-4-5	9,515 ms	100.08 MB	31.04
	Leftover	34,303 ms	692.83 MB	32.86
	Total	**43,818 ms**	**792.91 MB**	**63.99**
Proposed	1-2-3-4-5	893 ms	65.98 MB	31.04
	Leftover	4,181 ms	456.81 MB	32.86
	Total	**5,074 ms**	**522.79 MB**	**63.99**

ตารางที่ 4.4 ตารางเปรียบเทียบขั้นตอนวิธีสำหรับสองปัญหาย่อยของปริศนาเรียงสิบห้าเลข

ขั้นตอนย่อย	จำนวนรูปแบบ	จำนวนการเปลี่ยนรูปแบบเฉลี่ย
1-2-3-4	524,160	67.64
5-9-13	11,880	
8-Puzzle	362,880	
1-2-3-4-5	5,765,760	63.99
Leftover	39,916,800	
Optimal	2.09×10^{13}	52.59

ตารางที่ 4.5 ตารางเปรียบเทียบรูปแบบการแบ่งเป็นโจทย์ย่อยสำหรับปริศนาเรียงสิบห้าเลข

4.3 สรุปผลการทดลอง

จากผลการทดลองขั้นตอนวิธีนี้ในปริศนาเรียงแปดเลข พบว่าขั้นตอนวิธีนี้ใช้เวลาในการหาคำตอบน้อยกว่าถึงสี่เท่าเมื่อเทียบกันขั้นตอนวิธีก่อนหน้า และใช้หน่วยความจำน้อยลงอีกด้วย ซึ่งขั้นตอนวิธีทั้งสองแบบนี้ แตกต่างกันเพียงที่วิธีการจัดเก็บข้อมูลและรูปแบบการประมวลผล ดังตารางที่ 4.6

ขั้นตอนวิธี	โครงสร้างข้อมูล	การประมวลผล	เวลาที่ใช้	หน่วยความจำที่ใช้
IDA* (Reinefeld, 1993)	ไม่ใช้	ลำดับ	30 ms	1322 bytes
BFS (Parberry, 2015)	ตารางแฮช	ลำดับ	288 ms	6.298 MB
Proposed	แถวลำดับ	ขนาน (GPU)	61 ms	4.153 MB

ตารางที่ 4.6 ตารางเปรียบเทียบขั้นตอนวิธีในการแก้ปริศนาเรียงแปดเลข

จากตารางที่ 4.3 พบว่าสำหรับนขั้นตอนวิธี IDA* เวลาที่ใช้ในการหาคำตอบนั้นจะขึ้นอยู่กับความลึกของต้นไม้หรือก็คือจำนวนการเปลี่ยนรูปแบบของปัญหานั้นๆ โดยสำหรับปัญหาการเรียงเลขซึ่งจำนวนการแตกกิ่งก้านประมาณ 2.07 ทำให้ต่อความลึกของต้นไม้เพิ่มขึ้นหนึ่ง จะทำให้เวลาที่ใช้ในการหาคำตอบเพิ่มขึ้นประมาณ 2.07 เท่า โดยเมื่อสนใจคำตอบของรูปแบบที่ยากที่สุดซึ่งมีทำให้ความลึกสูงสุดนั้นใช้เวลาในการหาคำตอบมากเกินไป

กรณีที่แบ่งแล้วปัญหามีขนาดใหญ่ อาจจะทำให้ใช้เวลาและหน่วยความจำในการประมวลผลสูงขึ้นด้วย จึงมีการเปรียบเทียบประสิทธิภาพของขั้นตอนวิธีต่างๆสำหรับปัญหาที่ใหญ่ขึ้น ผลเป็นไปดังตารางที่ 4.7 การใช้ขั้นตอนวิธีแบบขนานในหน่วยประมวลผลกราฟิกให้ประสิทธิภาพในการประมวลผลสูงขึ้นอีกแปดเท่า และการใช้จำนวนเต็มแทนรูปแบบในการเก็บข้อมูลใช้หน่วยความจำน้อยกว่าเล็กน้อย เนื่องจากต้องใช้คิวในการเก็บข้อมูลซึ่งมีขนาดใหญ่

ขั้นตอนวิธี	โครงสร้างข้อมูล	การประมวลผล	เวลาที่ใช้	หน่วยความจำที่ใช้
BFS (Parberry, 2015)	ตารางแฮช	ลำดับ	43,818 ms	792.91 MB
Proposed	แถวลำดับ	ขนาน (GPU)	5,074 ms	522.79 MB

ตารางที่ 4.7 ตารางเปรียบเทียบขั้นตอนวิธีในการแก้ปริศนาเรียงสิบห้าเลขโดยแก้ปัญหาย่อย

บทที่ 5
ข้อสรุปและข้อเสนอแนะ

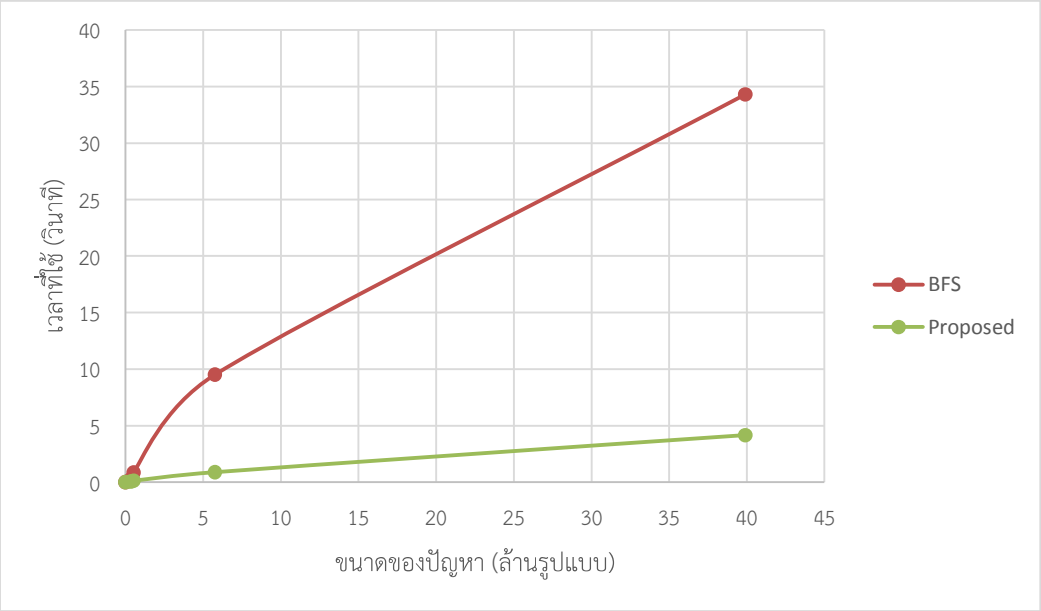
ในบทนี้จะกล่าวถึงการสรุปผลจากข้อมูลผลการวิจัยในบทที่ 4 ข้อสรุปการวิจัยขั้นตอนวิธีแบบขนานที่ใช้หน่วยความจำอย่างมีประสิทธิภาพสำหรับหาคำตอบที่ดีที่สุดในปริศนาที่มีการเลื่อนเป็นลำดับ รวมถึงข้อเสนอแนะอื่นๆ ที่ได้จากการวิจัยนี้

5.1 การสรุปผลการวิจัย

การแก้ปริศนาที่มีการเลื่อนเป็นลำดับโดยใช้จำนวนการเปลี่ยนรูปแบบให้น้อยที่สุด เพื่อเปลี่ยนจากรูปแบบใดๆ ให้กลับเป็นรูปแบบเริ่มต้นนั้น เป็นปัญหาที่ยากในทางวิทยาการคอมพิวเตอร์ เนื่องจากจำเป็นต้องใช้หน่วยความจำหรือเวลาจำนวนมากในการคำนวณ จึงมีการออกแบบขั้นตอนวิธีหลากหลายรูปแบบเพื่อให้ใช้หน่วยความจำและเวลาให้น้อยที่สุด โดยเริ่มจากวิธีการค้นหาในแนวลึกโดยจำกัดความลึกและเพิ่มความลึกหากไม่พบคำตอบ ขั้นตอนวิธีนี้ถูกออกแบบมาสำหรับหน่วยประมวลผลที่มีหน่วยความจำน้อยจึงมีประสิทธิภาพมากสำหรับหน่วยประมวลผลยุคก่อน แต่เมื่อปัญหามีความลึกมากขึ้นทำให้ใช้เวลาเพิ่มขึ้นอย่างรวดเร็วเกินไปจนไม่สามารถหาคำตอบสำหรับหลายปัญหาได้ จึงมีการแบ่งปัญหาขนาดใหญ่ให้เป็นปัญหาย่อย ซึ่งเป็นผลโดยตรงให้ความลึกและขนาดของปัญหาลดลงอย่างมาก แม้จะไม่ใช่คำตอบที่ดีที่สุดแต่คำตอบที่ได้ก็ยังอยู่ในระดับที่ใกล้เคียงคำตอบที่ดีที่สุดมากพอที่จะนำไปใช้จริงได้ โดยขั้นตอนวิธีล่าสุดสำหรับค้นหาคำตอบของปัญหานี้คือการค้นหาในแนวกว้าง เนื่องจากการพัฒนาของหน่วยความจำทำให้สามารถบันทึกข้อมูลคำตอบของรูปแบบทั้งหมดได้โดยเริ่มค้นหาจากรูปแบบเริ่มต้น และค้นหารูปแบบที่สามารถเปลี่ยนเป็นรูปแบบปัจจุบันได้และเก็บข้อมูลนั้นไว้ในโครงสร้างที่ต้องการ

งานวิจัยนี้พบว่าการใช้รูปแบบแทนจำนวนเต็มในการกำหนดตำแหน่งการเก็บข้อมูลในตัวแปรแบบแถวลำดับนั้นใช้หน่วยความจำได้อย่างมีประสิทธิภาพที่สุด เนื่องจากปัญหาที่มีการเลื่อนเป็นลำดับมักมีโครงสร้างคล้ายกับการเรียงสับเปลี่ยน จึงสามารถใช้การกำหนดจำนวนเต็มให้กับการเรียงสับเปลี่ยนในรูปแบบต่างๆได้ และใช้โครงสร้างข้อมูล Fenwick Tree มาเพิ่มประสิทธิภาพระหว่างขั้นตอนการแปลงเลขฐาน สามารถช่วยลดหน่วยความจำที่ต้องใช้ลงได้มากกว่ายี่สิบเท่า และลดเวลาในการค้นหาลงด้วยหน่วยประมวลผลกราฟิกซึ่งในปัจจุบันได้มีการพัฒนาอย่างรวดเร็วจนมีความเร็วมากกว่าหน่วยประมวลผลกลางหลายเท่า แต่การออกแบบขั้นตอนวิธีค้นหาด้วยหน่วยประมวลผลกราฟิกต้องรัดกุมมาก เนื่องจากข้อจำกัดหลายอย่างของหน่วยประมวลผลกราฟิกและการประมวลผลแบบขนาน เมื่อนำมาใช้ร่วมกับการค้นหาในกราฟที่เร่งความเร็วด้วยหน่วยประมวลผลกราฟิก อาจทำให้จำเป็นต้องใช้หน่วยความจำเพิ่มขึ้นอีกหลายเท่า

จากผลการทดลองพบว่าการออกแบบโครงสร้างใหม่และการค้นหาแนวกว้างแบบขนานในหน่วยประมวลผลกราฟิกยังคงใช้หน่วยความจำน้อยเมื่อเทียบกับขั้นตอนวิธีก่อนหน้า และสามารถลดเวลาที่ใช้ค้นหาคำตอบที่ดีที่สุดได้อีกสี่เท่าในปัญหาขนาดเล็ก และแปดเท่าในปัญหาที่มีขนาดใหญ่ขึ้น ดังภาพที่ 5.1 เป็นไปตามเป้าหมายที่ต้องการลดเวลาในการค้นหาและหน่วยความจำที่ใช้ได้ เพราะหากใช้หน่วยความจำมากเกินไปจะไม่สามารถหาคอมพิวเตอร์สำหรับประมวลผลได้ และหากใช้เวลานานเกินไป ก็ไม่สามารถรอให้ได้คำตอบสำหรับปัญหาที่มีขนาดใหญ่มากเกินไปได้ รวมถึงขั้นตอนวิธีสำหรับการแปลงรูปแบบต่างๆให้เป็นจำนวนเต็ม ยังสามารถประยุกต์ใช้กับปัญหาประเภทเดียวกันคือปัญหาที่มีการเลื่อนเป็นลำดับได้เป็นอย่างดีอีกด้วย



ภาพที่ 5.1 กราฟแสดงเวลาที่ใช้ในขั้นตอนวิธีต่างๆต่อขนาดของปัญหา

5.2 ข้อเสนอแนะ

1. การใช้ขั้นตอนวิธีแบบขนานสำหรับหน่วยประมวลผลกราฟิกที่ใช้ในงานวิจัยนี้ ยังสามารถเพิ่มประสิทธิภาพได้อีก เนื่องจากขั้นตอนวิธีที่ดีกว่ายังติดขีดจำกัดของฮาร์ดแวร์ ซึ่งอาจจะแก้ไขได้ด้วยการปรับปรุงขั้นตอนวิธีหรือการพัฒนาของหน่วยประมวลผลกราฟิกในอนาคต

2. ขนาดของคิวที่ใช้ในงานวิจัยนี้สร้างขึ้นสำหรับกรณีที่แย่ที่สุด ซึ่งอาจจะลดลงได้ขึ้นอยู่กับประเภทของปริศนาในสนใจ

รายการอ้างอิง

Brodtkorb, A. R., Hagen, T. R., & Sætra, M. L. (2013). Graphics processing unit (GPU) programming strategies and trends in GPU computing. *Journal of Parallel and Distributed Computing, 73*(1), 4-13.

Demaine, E. D., Eisenstat, S., & Rudoy, M. (2018). Solving the Rubik's Cube Optimally is NP-complete. *35th Symposium on Theoretical Aspects of Computer Science*, (pp. 24:1-24:13). Caen, France.

Elnaggar, A. A., Gadallah, M. E., Aziem, M. A., & Eldeeb, H. (2014). Enhanced parallel NegaMax tree search algorithm on GPU. *IEEE International Conference on Progress in Informatics and Computing*, (pp. 546-550). Shanghai, China.

Fenwick, P. M. (1994). A new data structure for cumulative frequency tables. *Software: Practice and Experience, 24*(3), 327-336.

Harish, P., & Narayanan, P. J. (2007). Accelerating Large Graph Algorithms on the GPU Using CUDA. *High Performance Computing 2007*, (pp. 197-208). Goa, India.

Luo, L., Wong, M., & Hwu, W.-m. (2010). An effective GPU implementation of breadth-first search. *Proceedings of the 47th Design Automation Conference*, (pp. 52-55). Anaheim, California.

Parberry, I. (2015). A Memory-Efficient Method for Fast Computation of Short 15-Puzzle Solutions. *IEEE Transactions on Computational Intelligence and AI in Games, 7*(2), 200 - 203.

ภาคผนวก

ภาคผนวก ก
แบบเสนอหัวข้อโครงงาน รายวิชา 2301399 Project Proposal
ปีการศึกษา 2561

ชื่อโครงงาน (ภาษาไทย)	ขั้นตอนวิธีแบบขนานที่ใช้หน่วยความจำอย่างมีประสิทธิภาพสำหรับหาคำตอบที่ดีที่สุดในปริศนาที่มีการเลื่อนเป็นลำดับ
ชื่อโครงงาน (ภาษาอังกฤษ)	Memory-Efficient Parallel Algorithm for Finding Optimal Solution to Sequential Move Puzzle
อาจารย์ที่ปรึกษา	ผศ.ดร.ศุภกานต์ พิมลธเรศ
ผู้ดำเนินการ	นายวรวุธ วรวิชญวงศา เลขประจำตัวนิสิต 5833656323 สาขาวิชาวิทยาการคอมพิวเตอร์ ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

หลักการและเหตุผล

การแก้ปริศนาที่มีการเลื่อนเป็นลำดับโดยใช้จำนวนการเลื่อนให้น้อยที่สุดนั้น เป็นหนึ่งในปัญหาที่ท้าทายมากในทางวิทยาการคอมพิวเตอร์ ปัญหาดังกล่าวหลายอย่างถูกระบุให้อยู่กลุ่มปัญหาเอ็นพีบริบูรณ์ เช่น ปัญหาเรียงเลข[1] และลูกบาศก์ของรูบิก[2] แต่การใช้ขั้นตอนวิธีการแก้ปกติแบบลำดับนั้นไม่สามารถแก้ปัญหากลุ่มได้ในเวลาที่ยอมรับได้ ในขณะที่ขั้นตอนวิธีแบบขนานกำลังเป็นที่นิยมใช้ในคอมพิวเตอร์ปัจจุบัน และอุปกรณ์สำหรับใช้ร่วมกับขั้นตอนวิธีแบบขนานก็กำลังถูกพัฒนาอย่างรวดเร็วเช่นกัน อย่างไรก็ตามการใช้ขั้นตอนวิธีแบบขนานสามารถลดได้แค่เวลาที่ใช้ในการประมวลผล แต่กลุ่มปัญหาดังกล่าวต้องการทั้งเวลาและหน่วยความจำปริมาณมากในการแก้ปัญหา คอมพิวเตอร์ปัจจุบันยังมีหน่วยความจำไม่พอเก็บคำตอบดังกล่าวได้ ทั้งระหว่างคำนวณและภายหลังจากการคำนวณจึงจำเป็นที่จะต้องมีโครงสร้างการเก็บข้อมูลที่มีประสิทธิภาพร่วมด้วย

อุปกรณ์สำหรับการประมวลผลแบบขนานที่มีประสิทธิภาพสูงสำหรับคอมพิวเตอร์ส่วนตัวในปัจจุบันคือหน่วยประมวลผลกราฟิก[3] โดยมีชุดคำสั่งจำนวนมากที่ใช้หน่วยประมวลผลกราฟิกเป็นตัวช่วยเพิ่มประสิทธิภาพการประมวลผล เนื่องจากการพัฒนาหน่วยประมวลผลกราฟิกให้สามารถประมวลผลข้อมูลทั่วไปได้ และการปฏิบัติการจุดลอยตัวต่อวินาทีที่สูงกว่าหน่วยประมวลผลกลางมากในราคาที่เท่ากัน ทำให้หน่วยประมวลผลกราฟิกมักใช้เป็นหน่วยประมวลผลแบบหลายแกนที่ประมวลผลขั้นตอนวิธีแบบขนานในคอมพิวเตอร์ส่วนตัว

ไม่นานนี้ได้มีงานวิจัยหลายอย่างที่เกี่ยวข้องกับการแก้ปัญหาดังกล่าว ขั้นตอนวิธีการค้นหาแบบเอสตาร์โดยเพิ่มความลึกทีละหนึ่ง[4] ถูกใช้ในการแก้ไขปัญหานี้โดยจำกัดความลึกสูงสุดและเพิ่มความลึกจนกว่าจะได้คำตอบที่ต้องการ แต่ขั้นตอนวิธีนี้จำเป็นต้องประมวลผลเส้นทางที่เคยประมวลผลแล้วซ้ำหลายครั้งและคำตอบที่สมบูรณ์หาได้เฉพาะปัญหาเรียงเลขแปดตัวเท่านั้น

การให้คะแนนรูปแบบโดยใช้ฐานข้อมูลแบบเพิ่มได้[5] แบ่งโจทย์ปัญหาเป็นส่วนย่อยที่ไม่เกี่ยวข้องกัน จากนั้นจึงแก้ปัญหาเหล่านั้นแล้วนำมารวมเป็นคำตอบที่ต้องการ วิธีการนี้ไม่ได้ให้คำตอบที่ดีที่สุดแต่สามารถประมวลผลได้อย่างรวดเร็วสำหรับปัญหาที่มีความซับซ้อนสูงขึ้นได้ เช่นปัญหาเรียงเลขสิบห้าตัวและปัญหาเรียงเลขยี่สิบสี่ตัว

ขั้นตอนวิธีก่อนหน้าใช้หลักการการค้นหาของแนวกว้างแบบลำดับบนหน่วยประมวลผลกลาง[6][7] และใช้ตารางแฮชเพื่อเก็บคำตอบสำหรับทุกรูปแบบ วิธีการนี้สามารถใช้เก็บคำตอบสำหรับปัญหาส่วนย่อยขนาดเล็กในหน่วยความจำสำรองได้แต่ไม่สามารถเก็บในหน่วยความจำหลักอย่างมีประสิทธิภาพได้เนื่องจากขนาดที่เล็กกว่ามาก และจำเป็นต้องใช้ระหว่างการประมวลผล ยิ่งไปกว่านั้นการค้นหาแบบลำดับถูกแสดงให้เห็นว่าช้ากว่าการค้นหาแบบขนานอย่างมากสำหรับปัญหาที่มีความซับซ้อนสูง[8]

การใช้ดัชนีการจัดหมู่ สามารถเก็บตารางคำตอบแบบแถวลำดับด้วยหน่วยความจำที่น้อยกว่ามาก รวมถึงการประยุกต์ใช้การค้นหาแบบขนานก็สามารถเพิ่มประสิทธิภาพของการค้นหาดังกล่าวได้อีกด้วย

วัตถุประสงค์

เพื่อพัฒนาขั้นตอนวิธีแบบขนาน ที่สามารถค้นหาคำตอบที่ดีที่สุด สำหรับทุกรูปแบบของปริศนาที่มีการเลื่อนเป็นลำดับ โดยใช้หน่วยความจำให้น้อยที่สุด ในเวลาที่ใช้ในการประมวลผลที่ยอมรับได้ และสร้างออกมาเป็นตารางคำตอบ

ขอบเขตของโครงงาน

1. ตัวงานต้องใช้กับปัญหาที่มีความซับซ้อนของรูปแบบได้มากถึง 10^{10}
2. ตัวงานต้องใช้กับปัญหาที่มีการเลื่อนเป็นลำดับได้ทุกรูปแบบ
3. คำตอบที่ได้จากตัวงานนี้ต้องเป็นคำตอบที่ดีที่สุด
4. ตัวงานทำงานบนคอมพิวเตอร์ที่มีหน่วยประมวลผลกราฟิกของ Nvidia ซึ่งรองรับ CUDA

วิธีการดำเนินงาน

การดำเนินงาน	2561					2562			
	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ	มี.ค.	เม.ย.
1. ศึกษางานวิจัยที่เกี่ยวข้อง									
2. ทดสอบขั้นตอนวิธีในงานวิจัย									
3. วิเคราะห์ปัญหาและพัฒนาขั้นตอนวิธีใหม่									
4. นำขั้นตอนวิธีใหม่ไปใช้									
5. วิเคราะห์ประสิทธิภาพขั้นตอนวิธี									
6. เขียนรายงาน									

ประโยชน์ที่คาดว่าจะได้รับ

ประโยชน์ต่อตัวนิสิต

1. ได้ศึกษาเพิ่มเติมเกี่ยวกับกราฟและการเรียงสับเปลี่ยน
2. ได้ศึกษาเพิ่มเติมเกี่ยวกับสถาปัตยกรรมการประมวลผลแบบขนานสำหรับหน่วยประมวลผลกราฟิก
3. ได้ศึกษาเพิ่มเติมเกี่ยวกับโครงสร้างการเก็บข้อมูลแบบใหม่
4. ได้ประสบการณ์การใช้หน่วยประมวลผลกราฟิกในการประมวลผลแบบขนาน

ประโยชน์ของโครงงาน

1. ได้ขั้นตอนวิธีที่ใช้เวลาในการค้นหาที่รวดเร็ว และตารางคำตอบที่เล็กลง สำหรับปริศนาที่มีการเลื่อนเป็นลำดับ
2. ได้พัฒนาแนวคิดการค้นหาคำตอบในต้นไม้แบบใหม่ สำหรับการพัฒนาต่อไปในอนาคต

อุปกรณ์และเครื่องมือที่ใช้

1. คอมพิวเตอร์ส่วนบุคคลซึ่งประกอบด้วย
    a. หน่วยประมวลผลกลาง Intel® Core

เอกสารอ้างอิง

[1] Ratner, D.; and Warmuth, M. The (n2–1)-puzzle and related relocation problems. Journal of Symbolic Computation 10 (August 1990): 111-137.

[2] Demaine, E.; D., Eisenstat, S.; and Rudoy, M. Solving the Rubik's Cube Optimally is NP-complete arXiv:1706.06708v2

[3] Brodtkorb, A.; R., Hagen, T. R.; and Sætra, M., L. Graphics processing unit (GPU) programming strategies and trends in GPU computing. Journal of Parallel and Distributed Computing 73 (January 2013): 4-13

[4] Reinefeld, A. Complete solution of the eight-puzzle and the benefit of node ordering in IDA IJCAI'93 Proceedings of the 13th International Joint Conference on Artificial intelligence pp. 248-253. Chambery. France, 1993

[5] Ariel, F.; Richard, E. K.; and Sarit, H. Additive pattern database heuristics. Journal of Artificial Intelligence Research 22 (July 2004): 279-318

[6] Parberry, I. A Memory-Efficient Method for Fast Computation of Short 15-Puzzle Solutions. IEEE Transactions on Computational Intelligence and AI in Games 7 (June 2015): 200-203

[7] GuiPing, W.; and Jun, S. BBFS-STT: An efficient algorithm for number rotation puzzle. Entertainment Computing 12 (January 2016): 1-7

[8] Elnaggar, A. A.; Gadallah, M.; Aziem, M. A.; and El-Deeb, H. Enhanced Parallel NegaMax Tree Search Algorithm on GPU 2014 IEEE International Conference on Progress in Informatics and Computing pp. 546 – 550. Shanghai. China, 2014

ภาคผนวก ข
โครงสร้างการทำงานของหน่วยประมวลผลกราฟิก

ในงานวิจัยนี้จะกล่าวถึงโครงสร้างของหน่วยประมวลผลกราฟิกของ Nvidia เท่านั้น จะอธิบายถึงลักษณะการออกแบบและขีดจำกัดต่าง ๆ ของการทำงานของหน่วยประมวลผลกราฟิกนี้

ในทางฮาร์ดแวร์จะมีการจัดหน่วยประมวลผลออกเป็นหลายกลุ่มเรียกว่า Streaming Multiprocessor (SM) โดยทุก SM จะมีหลายหน่วยประมวลผลรวมกัน แต่ทุกหน่วยประมวลผลใน SM จะต้องทำงานคำสั่งเดียวกันเสมอ โดยงานที่ได้รับมาจะเรียกว่า Block และจะมีการแบ่งงานที่ได้รับออกเป็นหลายกลุ่ม โดยขนาดของกลุ่มงานจะขึ้นอยู่กับจำนวนหน่วยประมวลผล กลุ่มของเธรดที่ทำงานคำสั่งเดียวกันเรียกว่า Warp เนื่องจากในแต่ละ Warp จำเป็นต้องทำงานคำสั่งเดียวกันเสมอ เมื่อทำงานถึงคำสั่งที่ทำให้เกิดการแบ่งกลุ่มกันทำงานและจะทำให้ทุกเธรดเกิดการรอกัน แล้วจะทำให้ประสิทธิภาพต่ำลงเรียกว่า Warp Divergence จึงต้องมีความระมัดระวังในการออกแบบขั้นตอนวิธีในเรื่องนี้ด้วย

หน่วยประมวลผลกราฟิกที่มีหน่วยประมวลผลจำนวนมาก หากมีหน่วยความจำเพียงหลักเพียงหนึ่งเดียวจะทำให้เสียเวลาในการลำดับการเข้าถึงข้อมูลได้ จึงมีการออกแบบให้มีหน่วยความจำหลายระดับ ดังตารางที่ ข.1 โดยหน่วยความจำในใหญ่ที่สุด สามารถเข้าถึงได้จากทุกหน่วยประมวลผล ว่า Global Memory หน่วยความจำย่อยลงมาจะอยู่ในแต่ละ SM เรียกว่า Shared Memory ซึ่งจะแบ่งให้สำหรับแต่ละ Block เข้าถึงได้ และมีหน่วยความจำที่เล็กที่สุดสำหรับหน่วยประมวลผลแต่ละตัวเรียกว่า Register รวมถึงมี Constant Memory ซึ่งเป็นหน่วยความจำสำหรับเก็บค่าคงทีเท่านั้น และมีการแบ่งข้อมูลไปเก็บในหน่วยความจำที่สามารถเข้าได้จากทุกที่ด้วยความเร็วที่สูงเป็นพิเศษ ดังตารางที่ ข.2

Hardware	Software	Memory
GPU	Device / Kernel	Global Memory
Streaming Multiprocessor	Grid	Global Memory
Processor Block	Block	Shared Memory
Processor Block	Warp	Shared Memory
Processor	Thread	Register

ตารางที่ ข.1 ตารางแสดงหน่วยความจำที่เข้าถึงได้

Memory	Clock cycle
Global Memory	600
Constant Memory	5
Shared Memory	5
Register	1

ตารางที่ ข.2 ตารางเปรียบเทียบการเข้าถึงหน่วยความจำ

เนื่องจากหน่วยความจำกลาง หรือ Global Memory มีประสิทธิภาพการเข้าถึงที่ต่ำ แต่จำเป็นต้องเข้าถึงจากทุกหน่วยประมวลผล จึงมีการรวมชุดคำสั่งที่ใช้เข้าถึงหน่วยความจำกลางในช่องที่ติดกัน จากหน่วยประมวลผลกลุ่มเดียวกันให้เป็นหนึ่งชุดคำสั่งเรียกว่า Memory Coalescing ซึ่งมักใช้ในการคัดลอกข้อมูลระหว่าง Global Memory และ Shared Memory เพื่อใช้ในการประมวลผลต่อไป การออกแบบขั้นตอนวิธีจึงต้องคำนึงถึงจุดนี้ด้วย

สำหรับหน่วยประมวลผลกราฟิกที่ใช้ในงานวิจัยนี้คือ Nvidia GeForce GTX 750 Ti ซึ่งมีคุณสมบัติ ดังตารางที่ ข.3 และไม่มีชุดคำสั่งสำหรับจองพื้นที่ในหน่วยความจำขณะทำงาน จึงจำเป็นต้องจองพื้นที่ในหน่วยความจำตั้งแต่การเขียนชุดคำสั่งและประเมินหน่วยความจำสูงสุดที่อาจต้องใช้ แต่ต้องระมัดระวังเรื่องขนาดหน่วยความจำสูงสุดที่มีให้ใช้ด้วยเสมอ

Specifications	Value
CUDA Cores	640
Block Size	1024
Warp Size	32
Global Memory	2048 MB
Constant Memory	64 KB
Shared Memory per block	48 KB
32-bit Register per thread	255
Compute Capability	5.0

ตารางที่ ข.3 ตารางแสดงคุณสมบัติของ Nvidia GeForce GTX 750 Ti

ภาคผนวก ค
ต้นไม้ของเฟนวิก

ต้นไม้ของเฟนวิก หรือเรียกอีกอย่างว่าต้นไม้กำหนดหลักทวิภาค (Fenwick, 1994) เป็นโครงสร้างข้อมูลรูปแบบหนึ่งถูกออกแบบมาสำหรับการหาความถี่สะสมของข้อมูลโดยต้นไม้นี้จะถูกแบ่งออกเป็นสองต้นไม้คือต้นไม้สำหรับอ่านค่าและต้นไม้สำหรับแก้ไขค่า โดยต้นไม้นี้จะถูกสร้างไว้บนตัวแปรแถวลำดับเดียวกัน และใช้บิตของแต่ละลำดับของข้อมูลเก็บข้อมูลเป็นตัวบอกตำแหน่งบนต้นไม้ โดยประโยชน์อย่างหนึ่งของโครงสร้างข้อมูลนี้คือสามารถหาบวกค่าคงที่ลงไปในช่วงของข้อมูลใดๆ ได้ด้วยความซับซ้อนเชิงเวลา

สำหรับการหาค่าในตำแหน่งใดๆในโครงสร้างข้อมูลนี้ สามารถหาได้จากรหัสเทียม ในภาพที่ ค.3 โดยสังเกตว่าทิศทางการค้นหาจะตรงกับข้ามกับการแก้ไขข้อมูล แสดงให้เห็นว่าต้นไม้สำหรับค้นหาและแก้ไขข้อมูลนั้นต่างกันเพียงเส้นเชื่อมเท่านั้น

```
Let index be the position to find the value
Let data be the Fenwick tree
value = 0
while index > 0
        value = value + data[index]
        index = index       –  ((index) & (       - index))
return value
```

ภาพที่ ค.3 รหัสเทียมสำหรับหาค่าในโครงสร้างข้อมูล

ในงานวิจัยนี้จะมีการค้นหาตำแหน่งแรกที่มีค่าในโครงสร้างข้อมูลตรงกับที่ต้องการ จึงต้องมีกขั้นตอนวิธีเพิ่มเติมจากงานวิจัยเดิม เพื่อให้สามารถหาตำแหน่งได้อย่างมีการประสิทธิภาพ ซึ่งสามารถหาได้จากรหัสเทียม ในภาพที่ ค.4

```
Let value be the value to find an index
Let data be the Fenwick          tree
depth = Most significant bit of size of data
current = depth
while depth > 0
        depth = depth / 2
        if current > size of data
                current = current          –  depth
                restart the loop
        if data[current] = value
                position = current
        if data[current] >= value
                current = current          –  depth
        else
                value = value       –  data[current]
                current = current + depth
return position
```

ภาพที่ ค.4 รหัสเทียมสำหรับค้นหาตำแหน่งในค่าในโครงสร้างข้อมูล

ความซับซ้อนเชิงเวลาของทั้งสามขั้นตอนวิธีนี้คือ

ประวัติผู้เขียน

นายวรวุธ วรวิชญวงศา

กำลังศึกษาในโครงการเกียรตินิยม สาขาวิทยาการคอมพิวเตอร์
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย