ใบรับรองวิทยานิพนธ์

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์

วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)

ปริญญา

วิทยาการคอมพิวเตอร์ วิทยาการคอมพิวเตอร์

สาขา **ภาควิชา**

เรื่อง ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิก

Parallel Compact Genetic Algorithm On GPU

นามผู้วิจัย นางสาวนุกูล สถาพร

ได้พิจารณาเห็นชอบโดย

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก .....

(ผู้ช่วยศาสตราจารย์วรเศรษฐ สุวรรณิก, วศ.ด.)

หัวหน้าภาควิชา .....

(ผู้ช่วยศาสตราจารย์ศิริกร จันทร์นวล, M.Sc.)

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์รับรองแล้ว

.....

(รองศาสตราจารย์กัญจนา ธีระกุล, D.Agr.)

คณบดีบัณฑิตวิทยาลัย

วันที่ ..... เดือน ..... พ.ศ. .....

วิทยานิพนธ์

เรื่อง

ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิก

Parallel Compact Genetic Algorithm on GPU

โดย

นางสาวนุกูล สถาพร

เสนอ

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์
เพื่อความสมบูรณ์แห่งปริญญาวิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)
พ.ศ. 2555

นุกูล สถาพร 2555: ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิก ปริญญาวิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์) สาขาวิทยาการคอมพิวเตอร์ ภาควิชาวิทยาการคอมพิวเตอร์ อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก: ผู้ช่วยศาสตราจารย์วรเศรษฐ สุวรรณิก, วศ.ด. 81 หน้า

งานวิจัยนี้มีวัตถุประสงค์หลักเพื่อเพิ่มความเร็วในการทำงานให้กับขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ ผู้วิจัยได้นำเสนอการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิก โดยศึกษาผลการทำงานเมื่อมีวัดค่าความเหมาะสม 2 แบบคือการวัดค่าความเหมาะสมโดยใช้เธรดเดียว และการวัดค่าความเหมาะสมโดยการลดทอน และศึกษาผลการทำงานเมื่อมีจำนวนเธรดในบล็อกแตกต่างกัน

การทดลองวัดประสิทธิภาพทำกับปัญหาทดสอบ 3 ปัญหา คือ ปัญหา OneMax, ปัญหา OneMax Noisy และปัญหา Royal Road โดยทดลองกับปัญหาขนาด 1 x 512 บิต จนถึง 512 x 512 บิต เปรียบเทียบผลการทดลองกับขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบอนุกรมบนหน่วยประมวลผลกลาง ผลการทดลองพบว่า ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานสามารถทำงานได้เร็วกว่าขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบอนุกรมในปัญหาขนาด 8 x 512 บิตขึ้นไปทุกปัญหาทดสอบ ไม่ว่าจะใช้การวัดค่าความเหมาะสมแบบเธรดเดียวหรือการลดทอน โดยการวัดค่าความเหมาะสมแบบการลดทอนสามารถทำงานได้เร็วกว่าการวัดค่าความเหมาะสมแบบเธรดเดียว และการลดจำนวนเธรดในบล็อกจะส่งผลให้ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานทำงานได้เร็วขึ้น

_____ _____ ___ / ____ / ___

ลายมือชื่อนิสิต ลายมือชื่ออาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

Nugool Sataporn 2012: Parallel Compact Genetic Algorithm on GPU. Master of Science (Computer Science), Major Field: Computer Science, Department of Computer Science. Thesis Advisor: Assistant Professor Worasait Suwannik, Ph.D. 81 pages.

The objective of this research is to accelerate the compact genetic algorithm (cGA). We propose implementations of parallel compact genetic algorithm (pcGA) on Graphic Processing Unit (GPU) by using two different fitness functions: one-thread method and reduction method as well as the different number of threads in block

For the experiment, we compare the performance of pcGA in 3 problems: OneMax, OneMax Noisy, and Royal Road. The results show that for a large problem (from 8x512 to 512x512 bits), the performance of the proposed algorithm is better than the performance of serial compact genetic algorithm either one-thread method or reduction method in all problems. By pcGA using one-thread fitness function perform better than pcGA using reduction fitness function. Further, reducing number of threads in block can accelerate pcGA.

Student's signature Thesis Advisor's signature

กิตติกรรมประกาศ

ผู้วิจัยขอกราบขอบพระคุณ ผศ.ดร. วรเศษฐ สุวรรณิก ประธานกรรมการที่ปรึกษา ที่ได้ให้คำปรึกษาในด้านการเรียน การค้นคว้าวิจัย ชี้แนะแนวทางแก้ไขปัญหาในงานวิจัย จนกระทั่งงานวิจัยฉบับนี้เสร็จสมบูรณ์

ขอกราบขอบพระคุณสถาบันส่งเสริมการสอนวิทยาศาสตร์และเทคโนโลยี (สสวท.) ที่ได้ให้การสนับสนุนด้านเงินทุนในการศึกษาในระดับปริญญาโทในครั้งนี้

ขอขอบคุณ คุณอณิมา รอตเสียงล้ำ สำหรับคำแนะนำทั้งในด้านการเรียนและการทำวิจัย พร้อมทั้งให้กำลังใจในการทำงานเสมอมาตลอดระยะเวลาในการทำวิจัย

ขอขอบคุณ ภาควิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยเกษตรศาสตร์ที่ได้สนับสนุนด้านอุปกรณ์ และสถานที่ตลอดระยะเวลาในการศึกษาในระดับปริญญาโทในครั้งนี้

ขอขอบพระคุณ คณะวิทยาศาสตร์ มหาวิทยาลัยเกษตรศาสตร์ ที่ได้สนับสนุนทุนการศึกษาในการไปนำเสนองานวิจัย ณ เกาะเชจู ประเทศเกาหลี

ขอขอบพระคุณ บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์ ที่ได้สนับสนุนทุนการศึกษาในการไปนำเสนองานวิจัย ณ เกาะเชจู ประเทศเกาหลี

ด้วยความดีหรือประโยชน์อันใดเนื่องจากวิทยานิพนธ์เล่มนี้ ขอมอบแด่คุณพ่อทองพูน สถาพร และคุณแม่สมควร สถาพร ที่ได้กรุณาอบรมและให้กำลังใจผู้วิจัยมาตลอดทุกเรื่อง

นุกูล สถาพร
เมษายน 2555

สารบัญ

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญตาราง (ต่อ)

ตารางที่ **หน้า**

สารบัญตาราง (ต่อ)

ตารางที่ **หน้า**

สารบัญภาพ

ภาพที่ **หน้า**

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

คำอธิบายสัญลักษณ์และคำย่อ

GA	=	Genetic Algorithm
cGA	=	Compact Genetic Algorithm
$scGA$	=	Serial Compact Genetic Algorithm ที่ใช้ในการทดลอง
$pcGA$	=	Parallel Compact Genetic Algorithm ที่ใช้ในการทดลอง
S_{exec}	=	ความเร็วที่เพิ่มขึ้นเมื่อเปรียบเทียบเวลาในการทำงานทั้งหมดของ pcGA กับ scGA
S_{Iter}	=	ความเร็วที่เพิ่มขึ้นเมื่อเปรียบเทียบเวลาในการทำงานต่อรอบของ pcGA กับ scGA

ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิก

Parallel Compact genetic algorithm on GPU

คำนำ

ขั้นตอนวิธีเชิงพันธุกรรม (Genetic Algorithm - GA) เป็นกระบวนการในการค้นหาคำตอบที่เลียนแบบมาจากการคัดเลือกทางพันธุกรรมของสิ่งมีชีวิต (Glodberge, 1989; Mitchell, 1998) การจัดเก็บโครโมโซมของ GA จะถูกเข้ารหัสเป็นเลขฐานสอง (binary string) การจัดเก็บประชากรทั้งหมดต้องใช้หน่วยความจำขนาด $n \times l$ เมื่อ n คือจำนวนประชากร และ l คือความยาวของโครโมโซม ดังนั้นในปัญหาขนาดใหญ่ GA จะต้องการพื้นที่หน่วยความจำในการจัดเก็บโครโมโซมมากขึ้น เช่นการแก้ปัญหาขนาด 1,000 บิตและมีจำนวนประชากร 1 ล้านตัว การจัดเก็บประชากรทั้งหมดใน GA จะต้องการพื้นที่หน่วยความจำขนาด 1 GB

ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ (Compact genetic algorithm - cGA) เป็นขั้นตอนวิธีที่ใช้แบบจำลองของความน่าจะเป็นในการค้นหาคำตอบ (Harik et al., 1998) โดยแทนที่คำตอบในรูปของเวกเตอร์ความน่าจะเป็นและจัดเก็บขนาดของประชากรเป็นค่าคงที่แทนที่จะสร้างประชากรทั้งหมดและเก็บลงบนหน่วยความจำ โครโมโซมความยาว l จะต้องการพื้นที่ในการจัดเก็บ $l \times log_2(n+1)$ บิต เมื่อ n คือขนาดของประชากร ดังนั้นในปัญหาขนาด 1,000 บิตและมีจำนวนประชากร 1 ล้านตัว cGA จึงต้องการพื้นที่หน่วยความจำเพียง 20 MB ในการจัดเก็บประชากรทั้งหมด จะเห็นว่า cGA ต้องการพื้นที่หน่วยความจำในการจัดเก็บโครโมโซมน้อยกว่า GA แบบดั้งเดิม การทำงานของ cGA จะทำงานโดยปรับเปลี่ยนค่าเวกเตอร์ความน่าจะเป็นให้เข้าใกล้คำตอบ เวลาที่ใช้ในการหาคำตอบขึ้นอยู่กับขนาดของเวกเตอร์ความน่าจะเป็นซึ่งเท่ากับความยาวโครโมโซม ดั้งนั้นถ้าโครโมโซมมีความยาวมากจะส่งผลให้ใช้เวลาในการแก้ปัญหามากขึ้น

หน่วยประมวลผลกราฟิก (Graphics Processors Units : GPU) เป็นหน่วยประมวลผลที่ช่วยลดงานด้านการแสดงผลของหน่วยประมวลผลกลาง (Central Processing Unit : CPU) ในปัจจุบันมีการนำเทคโนโลยีที่เรียกว่า GPGPU (General-Purpose computation on GPUs) คือการนำหน่วยประมวลผลของ GPU ที่เดิมใช้ในการประมวลผลทางด้านกราฟิกเพียงอย่างเดียวมาใช้ในการประมวลผลข้อมูล ซึ่งบน GPU นี้มีเธรดจำนวนมากที่สามารถทำงานแบบขนานทำให้สามารถประมวลผลได้เร็วกว่า CPU เมื่อมีอัลกอริทึมที่เหมาะสม

งานวิจัยนี้นำเสนอการพัฒนาขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน (Parallel Compact Genetic Algorithm : pcGA) บนหน่วยประมวลผลกราฟิก เพื่อช่วยลดเวลาในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์

วัตถุประสงค์

1. เพื่อเพิ่มความเร็วในการหาคำตอบของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์

2. เพื่อศึกษาความเร็วในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิกเมื่อมีการวัดค่าความเหมาะสมที่แตกต่างกัน

3. เพื่อศึกษาความเร็วในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิกเมื่อจำนวนเธรดในบล็อกแตกต่างกัน

ประโยชน์ที่คาดว่าจะได้รับ

1. ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์สามารถทำงานได้เร็วขึ้นเมื่อเปรียบเทียบกับการทำงานบนหน่วยประมวลผลกลางเพียงอย่างเดียว

2. ทราบถึงประสิทธิภาพในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์บนหน่วยประมวลผลกราฟิกเมื่อมีการวัดค่าความเหมาะสมที่แตกต่างกัน

3. ทราบถึงประสิทธิภาพในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์บนหน่วยประมวลผลกราฟิกเมื่อจำนวนเธรดในบล็อกแตกต่างกัน

ขอบเขตและข้อจำกัด

1. ทดสอบกับปัญหา OneMax, OneMax Noisy, และ Royal Road

2. ทดสอบกับปัญหาขนาด 1 x 512 บิต จนถึง 512 x 512 บิต โดยเพิ่มขนาดปัญหาครั้งละ 2 เท่า

3. ใช้แบบจำลองการเขียนโปรแกรม CUDA ในการทำงานบนหน่วยประมวลผลกราฟิก

การตรวจเอกสาร

ทฤษฎีที่เกี่ยวข้อง

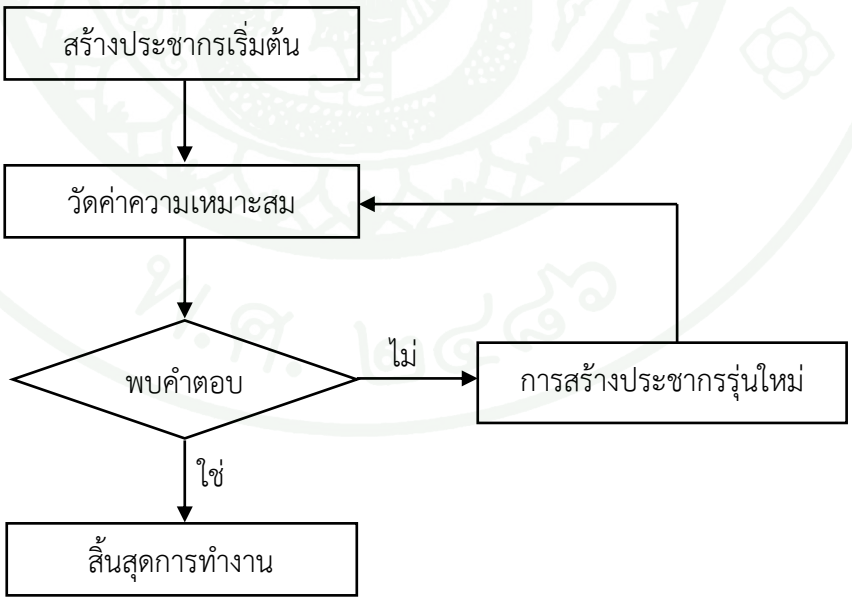
1. ขั้นตอนวิธีเชิงพันธุกรรม (Genetic Algorithm)

1.1 ความหมายของขั้นตอนวิธีเชิงพันธุกรรม

ขั้นตอนวิธีเชิงพันธุกรรม (Glodberge, 1989; Mitchell, 1998) เป็นขั้นตอนวิธีที่ใช้กระบวนการทางพันธุกรรมในการแก้ปัญหา การทำงานเริ่มต้นด้วยการสร้างประชากรซึ่งเป็นตัวแทนของคำตอบที่ต้องการ จากนั้นคัดเลือกสิ่งมีชีวิตที่เหมาะสมมาดำเนินการทางพันธุกรรมเพื่อสร้างประชากรรุ่นใหม่ที่ดีขึ้นกว่าเดิม ขั้นตอนวิธีเชิงพันธุกรรมถูกนำมาประยุกต์ใช้กับงานด้านต่าง ๆ อย่างแพร่หลาย โดยเฉพาะในปัญหาการค้นหาคำตอบที่เหมาะสม

1.2 การทำงานของขั้นตอนวิธีเชิงพันธุกรรม

การทำงานของขั้นตอนวิธีเชิงพันธุกรรมแสดงดังภาพที่ 1 โดยมีการทำงานดังนี้



ภาพที่ 1 ลำดับการทำงานของขั้นตอนวิธีเชิงพันธุกรรม

1.2.1 การเข้ารหัสโครโมโซม (Encode mechanism)

การเข้ารหัสโครโมโซมเป็นการเข้ารหัสตัวแทนของคำตอบ ซึ่งจะขึ้นอยู่กับลักษณะของปัญหา คำตอบที่ได้อาจจะอยู่ในรูปของเลขฐานสอง เรียกว่า โครโมโซม (ตัวอย่างโครโมโซม เช่น 010011011) ชุดของโครโมโซมจะเรียกว่าประชากร

1.2.2 การวัดค่าความเหมาะสม (Fitness function)

การวัดค่าความเหมาะสมเป็นฟังก์ชันที่ใช้ในการประเมินค่าความเหมาะสมของโครโมโซมแต่ละตัว ค่าความเหมาะสมนี้จะขึ้นกับลักษณะของปัญหา

1.2.3 การดำเนินการทางพันธุกรรม (Genetic Operation)

การสร้างประชากรรุ่นใหม่ในขั้นตอนวิธีเชิงพันธุกรรม ต้องใช้ตัวดำเนินการทางพันธุกรรมเพื่อให้ได้ประชากรรุ่นใหม่ที่มีโอกาสมีค่าความเหมาะสมสูงกว่าประชากรรุ่นเดิม ตัวดำเนินการทางพันธุกรรมประกอบด้วย

(ก) การคัดเลือก (Selection) เป็นการคัดเลือกประชากรในรุ่นก่อนหน้าเพื่อนำมาสร้างประชากรรุ่นต่อไป ซึ่งกระบวนการคัดเลือกมีหลายวิธี แต่ส่วนใหญ่จะเลือกประชากรที่มีค่าความเหมาะสมสูง ๆ เพื่อให้สามารถสร้างประชากรรุ่นต่อไปที่มีประสิทธิภาพ

(ข) การไขว้เปลี่ยน (Crossover) เป็นการนำประชากรรุ่นก่อนหน้าจำนวน 2 ตัว เรียกว่าประชากรรุ่นพ่อ แม่ มาผสมกันเพื่อให้เกิดประชากรรุ่นใหม่

(ค) การกลายพันธุ์ (Mutation) เป็นการกลายพันธุ์ของประชากร โดยจะเปลี่ยนบิตในทางตรงข้าม เช่น จากบิต 0 เป็นบิต 1 และจากบิต 1 เป็นบิต 0 ซึ่งการกลายพันธุ์นี้อาจเกิดขึ้นหรือไม่ก็ได้ โดยดูจากค่าความน่าจะเป็นของโอกาสที่จะเกิดการกลายพันธุ์ ซึ่งค่าความน่าจะเป็นนี้ต้องกำหนดให้เหมาะสม

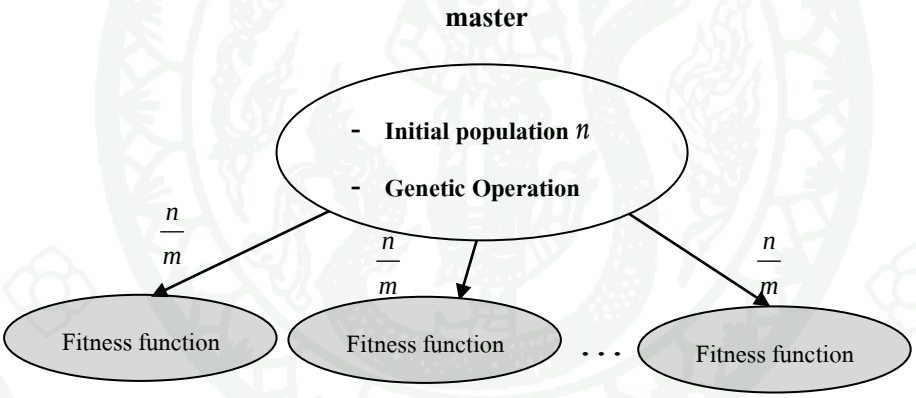
2. ขั้นตอนวิธีเชิงพันธุกรรมแบบขนาน (Parallel Genetic Algorithm)

ขั้นตอนวิธีเชิงพันธุกรรมแบบขนานเกิดจากแนวคิดในการใช้ประโยชน์จากโครงสร้างประชากรในขั้นตอนวิธีเชิงพันธุกรรมให้สอดคล้องกับการทำงานของฮาร์ดแวร์ที่มีหน่วยประมวลผลหลายตัว ซึ่งขั้นตอนวิธีเชิงพันธุกรรมเหมาะสมกับการทำงานแบบขนานเนื่องจากการดำเนินการของขั้นตอนวิธีเชิงพันธุกรรม เช่น การวัดค่าความเหมาะสม การคัดเลือก, การไขว้เปลี่ยน

และการกลายพันธุ์ สามารถทำงานได้อย่างอิสระ สามารถจัดกลุ่มประเภทของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานได้ 4 ประเภทดังนี้ (Cantú-Paz, 1998)

2.1 ขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบ master-slave (Global Single Population Master-Slave)

การทำงานของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบ master-slave ทำได้โดยสร้างประชากรทั้งหมดเป็นกลุ่มเดียวรวมทั้งจัดเก็บประชากรทั้งหมดและดำเนินการทางพันธุกรรมในส่วนของ master จากนั้น master จะแบ่ง individuals จำนวน $\frac{n}{m}$ ตัว (เมื่อ n คือจำนวนประชากร และ m คือจำนวน slave) ไปวัดค่าความเหมาะสมใน slave ซึ่ง slave นี้เป็นหน่วยประมวลผลที่สามารถทำงานได้พร้อมกัน การทำงานแบบขนานแบบนี้จะช่วยให้สามารถวัดค่าความเหมาะสมได้เร็วขึ้น ดังแสดงในภาพที่ 2

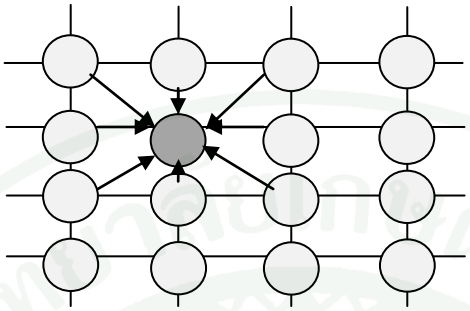


ภาพที่ 2 การแบ่งประชากรในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบ master-slave เมื่อ n คือจำนวนประชากร m คือจำนวน slave ที่ใช้

2.2 ขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบละเอียด (Single-population Fine-Grained)

ขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบละเอียดจะมองประชากรทั้งหมดเป็นกลุ่มเดียวเหมือนกับแบบ master-slave แต่จะกระจาย individuals แต่ละตัวไปทำงานในแต่ละหน่วยประมวลผลในรูปแบบของตารางหรือแถวลำดับ 2 มิติ การวัดค่าความเหมาะสมของ individuals แต่ละตัวสามารถทำได้โดยอิสระ การดำเนินการทางพันธุกรรมเพื่อสร้างประชากรรุ่นใหม่ทำได้โดยคัดเลือกจาก individuals ตัวที่อยู่ในตำแหน่งใกล้เคียงกับตำแหน่งที่ต้องการสร้างเท่านั้น ดังภาพที่ 3 วงกลมหนึ่งวงเป็นตัวแทน individuals หนึ่งตัวเมื่อต้องการสร้าง individuals ในรุ่นถัดไปในตำแหน่ง

วงกลมเข้มจะคัดเลือกจาก individuals ที่อยู่รอบๆ ทั้ง 8 individuals ตามลูกศรโดยเลือกเพียง 2 individuals เพื่อดำเนินการทางพันธุกรรมให้ได้ประชากรรุ่นใหม่แทนที่ในตำแหน่งวงกลมเข้ม



ภาพที่ 3 รูปแบบการคัดเลือก individuals ที่อยู่ใกล้เคียงเพื่อสร้างประชากรรุ่นใหม่ในขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบ fine-grained

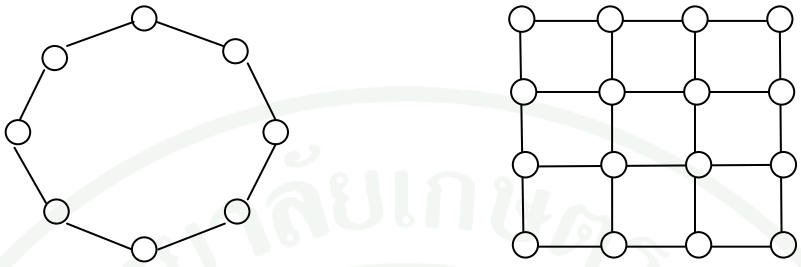
2.3 ขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบหยาบ (Multiple-population Coarse-Grained)

ขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบหยาบ อาจเรียกว่าการทำงานแบบขนานแบบเกาะไอซ์แลนด์ (island model) วิธีนี้จะแบ่งประชากรออกเป็นกลุ่มย่อย (subpopulation) โดยวงกลมหนึ่งวงแทนประชากรกลุ่มย่อยหรืออาจเรียกว่า “เกาะ” ซึ่งประชากรในแต่ละกลุ่มย่อยสามารถสร้างประชากร วัดค่าความเหมาะสมและดำเนินการทางพันธุกรรมอย่างเป็นอิสระ ซึ่งทำให้สามารถดำเนินการกับประชากรในแต่ละกลุ่มย่อยได้พร้อม ๆ กัน เมื่อถึงจำนวนรุ่นที่เหมาะสมจึงมีการกระจายประชากรระหว่างกลุ่มย่อยนี้โดยการอพยพประชากรจากเกาะหนึ่งไปยังเกาะอื่น ๆ การอพยพนี้ขึ้นกับ topology ที่เลือกใช้ ซึ่งมีหลายแบบ เช่น การเชื่อมต่อแบบต่อเนื่อง (chain), การเชื่อมต่อแบบทุกเกาะ (fully connected), การเชื่อมต่อแบบลูกบาศก์ (hypercube), การเชื่อมต่อแบบวงแหวน (ring), การเชื่อมต่อแบบวงล้อ (cartwheel) และการเชื่อมต่อแบบตาข่าย (lattice) เป็นต้น ดังแสดงตัวอย่างในภาพที่ 4

2.4 ขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบลำดับชั้น (Hierarchical Genetic Algorithm)

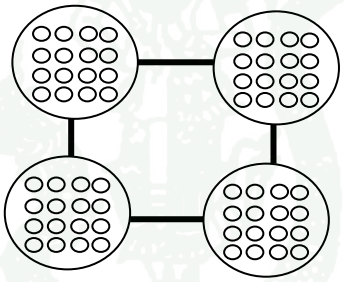
ขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบลำดับชั้นเป็นการนำวิธีการทำงานของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานมาทำงานทำงานร่วมกัน ซึ่งการทำงานแบบลำดับชั้นนี้จะมีการแบ่งประชากรออกเป็นกลุ่มย่อย การวัดค่าความเหมาะสมและดำเนินการทางพันธุกรรมจะทำภายในกลุ่มย่อยตามโครงสร้างของการทำงานแบบขนานแบบเกาะไอซ์แลนด์ แต่ภายในแต่ละเกาะ individuals จะเชื่อมต่อกันในรูปของตาราง ซึ่งการคัดเลือกประชากรเพื่อสร้างประชากรรุ่นใหม่ต้องดำเนินการ

ตามโครงสร้างของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบละเอียด การกระจาย individuals ไปยังเกาะต่าง ๆ และการเชื่อมต่อระหว่าง individuals ภายในเกาะแสดงดังภาพที่ 5



(1) การเชื่อมต่อระหว่างเกาะแบบวงแหวน (2) การเชื่อมต่อระหว่างเกาะแบบตาข่าย

ภาพที่ 4 รูปแบบการเชื่อมต่อระหว่างเกาะ (topology) ของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบ Coarse-Grained



ภาพที่ 5 การเชื่อมต่อระหว่างเกาะแบบวงแหวนและการกระจาย individuals ในแต่ละเกาะ ของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบลำดับชั้น

3. ขั้นตอนวิธีเชิงพันธุกรรมแบบคอมแพคท์ (Compact Genetic Algorithm)

3.1 ความหมายของขั้นตอนวิธีเชิงพันธุกรรมแบบคอมแพคท์

ขั้นตอนวิธีเชิงพันธุกรรมแบบคอมแพคท์ (Harik et al., 1998) แทนที่ประชากรของ GA ด้วยเวกเตอร์ความน่าจะเป็นและยกเลิกการดำเนินการทางพันธุกรรมของ GA แบบดั้งเดิม เช่น การไขว้เปลี่ยน, การกลายพันธุ์ การทำงานจะปรับเปลี่ยนค่าเวกเตอร์ความน่าจะเป็นเพื่อให้เข้าใกล้คำตอบ โดยเวกเตอร์ความน่าจะเป็นนี้จะกำหนดให้มีขนาด l เมื่อ l คือความยาวของโครโมโซมหรือขนาดของปัญหา เวกเตอร์ความน่าจะเป็นนี้จะเป็นสัดส่วนของค่า 1 ในโครโมโซม สามารถเขียนสัดส่วนนี้ด้วยเซตจำกัดขนาด $n + 1$ เมื่อ n คือจำนวนประชากรทั้งหมดดังนี้ $(0, \frac{1}{n}, \frac{2}{n}, \frac{3}{n}, ..., \frac{n}{n})$

ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์จึงสามารถจัดเก็บประชากรทั้งหมดด้วยพื้นที่หน่วยความจำเพียง $l \times log_2 (n+1)$ แทนที่จะเก็บประชากรทั้งหมดดังเช่น GA แบบดั้งเดิม (Harik et al., 1999)

3.2 การทำงานของขั้นตอนวิธีเชิงพันธุกรรมแบบคอมแพคท์

การทำงานของขั้นตอนวิธีเชิงพันธุกรรมแบบคอมแพคท์ ดังแสดงในภาพที่ 6 เริ่มต้นจะกำหนดค่าเวกเตอร์ความน่าจะเป็นของยีนทุกตำแหน่งเป็น 0.5 แล้วสร้างโครโมโซม 2 ตัวจากเวกเตอร์ความน่าจะเป็นนี้วัดค่าความเหมาะสมเพื่อหาโครโมโซมที่มีค่าความเหมาะสมสูงสุด เปรียบเทียบโครโมโซมทั้ง 2 ตัวถ้าเหมือนกันไม่ต้องแก้ไขค่าของเวกเตอร์ความน่าจะเป็น แต่ถ้าโครโมโซมทั้ง 2 ตัวต่างกันให้เพิ่มค่าเวกเตอร์ความน่าจะเป็นอีก $\frac{1}{n}$ เมื่อโครโมโซมตัวที่ชนะมีค่าเป็น 1 และลดความน่าจะเป็นลง $\frac{1}{n}$ เมื่อโครโมโซมตัวที่ชนะมีค่าเป็น 0 ทำงานไปเรื่อย ๆ จนกว่าค่าเวกเตอร์ความน่าจะเป็นมีค่าเป็น 0 หรือ 1 ทุกบิต

1) กำหนดค่าเริ่มต้นของเวกเตอร์ความน่าจะเป็นเท่ากับ 0.5

```
for i ← 1 to l do
        p[i] ← 0.5
```

2) สร้าง individuals a และ b จากเวกเตอร์ความน่าจะเป็น

```
a ← generate(p)
b ← generate(p)
```

3) วัดค่าความเหมาะสมของ individuals a และ b

```
winner, loser ← evaluate(a, b)
```

4) ปรับค่าเวกเตอร์ความน่าจะเป็นตาม individuals ที่มีค่าความเหมาะสมสูงกว่า

```
for i ← 1 to l do
    if winner[i] != loser[i] then
        if winner[i] = 1
            p[i] += 1/n
        else
            p[i] -= 1/n
```

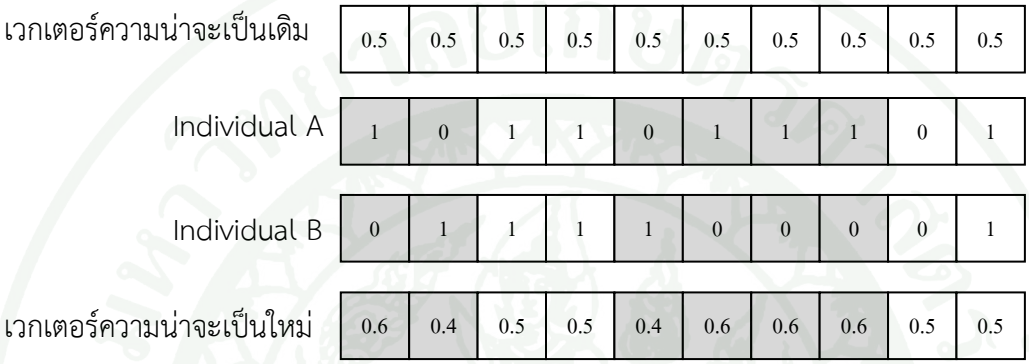
5) ตรวจสอบค่าเวกเตอร์ความน่าจะเป็นว่ามีค่าเป็น 0 หรือ 1 ทุกบิตหรือไม่

```
for i ← 1 to l do
    if p[i] > 0 and p[i] < 1 then
        return to step 2
```

6) เวกเตอร์ความน่าจะเป็น p จะแสดงคำตอบสุดท้าย

ภาพที่ 6 ขั้นตอนวิธีเชิงพันธุกรรมแบบคอมแพคท์

การปรับเปลี่ยนค่าเวกเตอร์ความน่าจะเป็นของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ดังแสดงในภาพที่ 7 เมื่อ n = 10 ค่าความเหมาะสมของ individuals A เท่ากับ 7 ค่าความเหมาะสมของ individuals B เท่ากับ 5 ดังนั้นจึงปรับเปลี่ยนเวกเตอร์ความน่าจะเป็นตาม individuals A รอบละ $\frac{1}{10}$ = 0.1 โดยแถบสีเทาแสดงการปรับเวกเตอร์ความน่าจะเป็นซึ่งการปรับเปลี่ยนค่าเวกเตอร์ความน่าจะเป็นจะทำไปจนกระทั่งทุก ๆ บิตของเวกเตอร์ความน่าจะเป็นนี้มีค่าเท่ากับ 0 หรือ 1



ภาพที่ 7 การปรับเปลี่ยนค่าเวกเตอร์ความน่าจะเป็นของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์

4. แบบจำลองการเขียนโปรแกรม CUDA (CUDA Programming model)

4.1 Host และ Kernel

แบบจำลองการเขียนโปรแกรม CUDA คือรูปแบบการเขียนโปรแกรมที่ขยายมาจากภาษา C/C++ โปรแกรมเมอร์สามารถเขียนโปรแกรมด้วยภาษา C/C++ (NVIDIA, 2010a) โดยแบ่งการทำงานเป็น 2 ส่วน ดังนี้

4.1.1 การทำงานบนหน่วยประมวลผลกลาง (CPU)

การทำงานในส่วนนี้เรียกว่า host ซึ่งจะทำงานแบบอนุกรม และเมื่อต้องการทำงานแบบขนานจะเรียกใช้ kernel ซึ่ง kernel นี้ดำเนินการในส่วนของ device บนหน่วยประมวลผลกราฟิก ในหนึ่งโปรแกรมอาจมี kernel สำหรับการทำงานแบบขนานมากกว่าหนึ่ง kernel ได้

4.1.2 การทำงานบนบนหน่วยประมวลผลกราฟิก (GPU)

การทำงานในส่วนนี้เรียกว่า device จะดำเนินการเมื่อโปรแกรมเรียกใช้ kernel บน host ซึ่ง kernel นี้จะถูกสร้างบนหน่วยประมวลผลกราฟิกในส่วนของ device ฟังก์ชันที่ทำงานบน device จะขึ้นต้นด้วย __global__ การเรียกใช้ kernel บนหน่วยประมวลผลกลางต้องกำหนดจำนวนบล็อกและจำนวนของเธรดในแต่ละบล็อกเพื่อให้ทราบว่าในการทำงานจะมีการดำเนินงานของเธรดทั้งหมดกี่เธรดดังแสดงในภาพที่ 8

1. ฟังก์ชันที่ทำงานบนหน่วยประมวลผลกราฟิก

```
__global__ void Vec_Add (float* A, float* B,
float* C)
{
      int  i  = threadIdx.x;
      C[i]   = A[i]  +   B[i];
}
```

2. การเรียกใช้ kernel บนหน่วยประมวลผลต้องกำหนดจำนวนบล็อกและจำนวนเธรดในแต่ละบล็อก

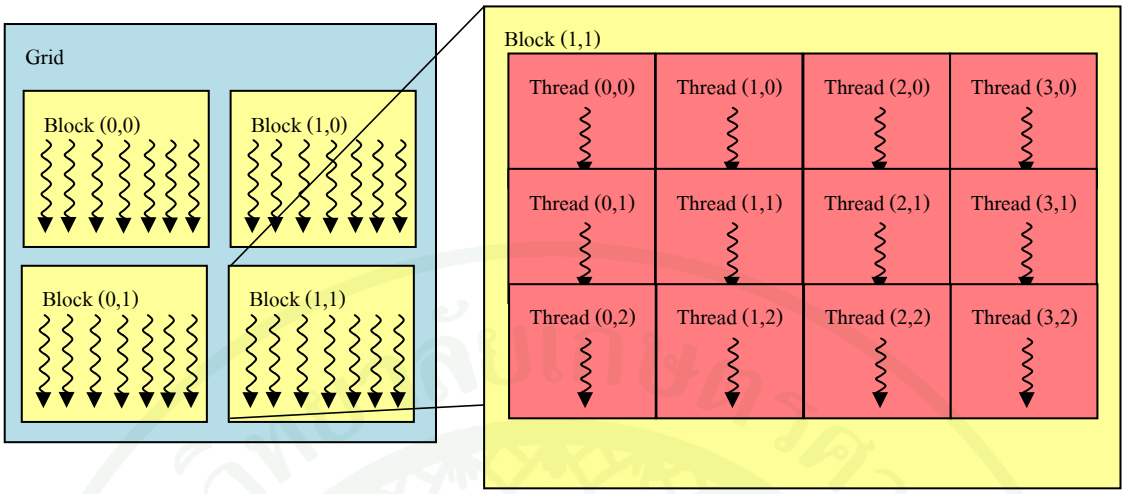
```
int main()
{
    float* A;
    float* B;
    float* C;

    Vec_Add<<1,N>>> (A, B, C);
}
```

ภาพที่ 8 การเรียกใช้ kernel บนหน่วยประมวลผลกลางและฟังก์ชันของ kernel ที่ทำงานบนหน่วยประมวลผลกราฟิก

4.2 เธรดและเธรดบล็อก (Thread and Thread Block)

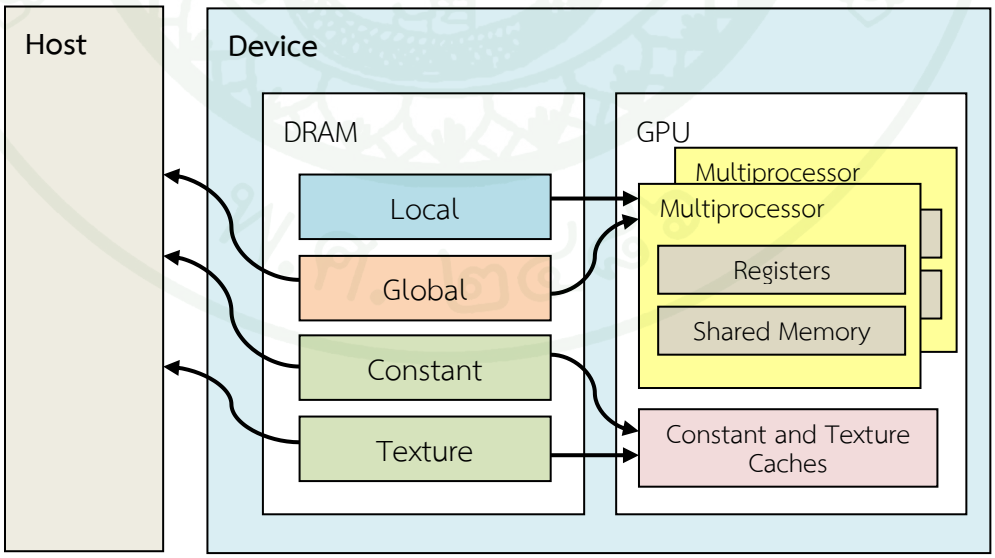
การทำงานแบบขนานของแบบจำลองการเขียนโปรแกรม CUDA จะใช้เธรดจำนวนมากดำเนินการพร้อม ๆ กัน โดยแบ่งเธรดเหล่านี้เป็นบล็อก แต่ละบล็อกมีเธรดได้ถึง 512 เธรด กลุ่มของเธรดในบล็อกจะทำงานร่วมกัน จนกระทั่งมีการขัดจังหวะเพื่อหยุดรอการทำงานภายในบล็อก โดยใช้คำสั่ง __syncthreads() แบบจำลองการเขียนโปรแกรม CUDA จะแบ่งบล็อกของเธรดเหล่านี้ลงบนตาราง (grid) ของหน่วยประมวลผลกราฟิก ส่วนของตารางนี้จะสามารถทำงานร่วมกันแบบขนาน แต่ละเธรดภายในบล็อกจะมีหมายเลขเธรด ซึ่งเรียกว่า threadIdx และแต่ละบล็อกจะมีหมายเลขบล็อกเรียกว่า blockIdx ดังแสดงในภาพที่ 9



ภาพที่ 9 ลำดับชั้นของเธรดในแบบจำลองการเขียนโปรแกรม CUDA

4.3 ประสิทธิภาพของหน่วยความจำประเภทต่าง ๆ (Memory Performance)

หน่วยความจำบนหน่วยประมวลผลกราฟิกมีหลายชนิด ซึ่งเธรดที่อยู่ลำดับชั้นต่างกันสามารถเข้าถึงหน่วยความจำเหล่านี้ได้แตกต่างกัน การทำงานบนหน่วยประมวลผลกราฟิกจึงต้องรู้จักการจัดการหน่วยความจำเหล่านี้เพื่อให้เกิดประโยชน์สูงสุดในการใช้งาน (NVIDIA, 2012a) ประเภทหน่วยความจำของหน่วยประมวลผลกราฟิกดังแสดงในภาพที่ 10 ได้แก่



ภาพที่ 10 หน่วยความจำประเภทต่าง ๆ บนหน่วยประมวลผลกราฟิก

4.3.1 รีจิสเตอร์ (Register)

หน่วยความจำประเภทรีจิสเตอร์จัดเก็บตัวแปรท้องถิ่นของแต่ละเธรด ซึ่งแต่ละเธรดสามารถเข้าถึงข้อมูลในรีจิสเตอร์ได้อย่างรวดเร็ว เธรดใด ๆ จะไม่สามารถเข้าถึงตัวแปรที่อยู่ในรีจิสเตอร์ของเธรดอื่นได้

4.3.2 หน่วยความจำแบบแบ่งส่วน (Shared memory)

หน่วยความจำแบบแบ่งส่วนใช้ในการเข้าถึงข้อมูลร่วมกันระหว่างเธรดภายในบล็อกเดียวกัน การเข้าถึงสามารถทำได้รวดเร็ว และมีศักยภาพสูงกว่าหน่วยความจำแบบครอบคลุมประมาณ 100 เท่า การประกาศใช้ตัวแปรแบบครอบคลุมจะขึ้นต้นด้วย __shared__

4.3.3 หน่วยความจำแบบท้องถิ่น (Local memory)

ใช้จัดเก็บตัวแปรท้องถิ่นของแต่ละเธรด แต่จัดเก็บลงบนหน่วยความจำนอกชิป ดังนั้นเวลาที่ใช้ในการเข้าถึงจึงเหมือนกับการเข้าถึงตัวแปรในหน่วยความจำแบบครอบคลุม

4.3.4 หน่วยความจำแบบครอบคลุม (Global memory)

เป็นหน่วยความจำบนหน่วยประมวลผลกราฟิก และเป็นหน่วยความจำส่วนที่ใหญ่ที่สุดของหน่วยประมวลผลกราฟิก สามารถเชื่อมต่อและส่งข้อมูลระหว่างหน่วยประมวลผลและหน่วยประมวลผลกราฟิกได้ การเข้าถึงทำได้ช้าเนื่องจากเป็นหน่วยความจำที่อยู่ภายนอกชิป

4.3.5 หน่วยความจำแบบพื้นผิว (Texture memory)

เป็นหน่วยความจำประเภทอ่านอย่างเดียว จัดเก็บลงบนหน่วยประมวลผลกราฟิก ซึ่งอยู่นอกชิป แต่มีแคชบนชิปทำให้เข้าถึงได้รวดเร็วเมื่อเข้าถึงผ่านแคช

4.3.6 หน่วยความจำแบบคงที่ (Constant memory)

เป็นหน่วยความจำแบบอ่านอย่างเดียว จัดเก็บในหน่วยความจำของหน่วยประมวลผลกราฟิกที่อยู่นอกชิปการทำงานจะทำได้ช้า แต่ถ้าทุกเธรดอ่านค่าที่เหมือนกันจากหน่วยความจำชนิดนี้ผลการทำงานจะเหมือนการอ่านจากรีจิสเตอร์

5. ปัญหาที่ใช้ในการทดลอง

5.1 ปัญหา OneMax

ปัญหา OneMax (Goldberg et al., 1992) เป็นปัญหาในการนับเลขหนึ่งในโครโมโซม ค่าความเหมาะสมจะขึ้นกับจำนวนเลขหนึ่งในโครโมโซมนั้น ดังแสดงในสมการ (1) ดังนั้นโครโมโซมที่ดีที่สุดคือโครโมโซมที่ทุกบิตมีค่าเป็นหนึ่งปัญหา OneMax เป็นการจำลองปัญหาที่ตัวแปรทุกตัวเป็นอิสระต่อกัน

$$f(x) = \sum_{i=1}^{l} x_i \tag{1}$$

เมื่อ l คือขนาดของโครโมโซม และ $x_i \in \{0,1\}$

5.2 ปัญหา OneMax Noisy

ปัญหา OneMax noisy (Goldberg et al., 1992) มีลักษณะคล้ายกับปัญหา OneMax แต่ปัญหาจะมีความยากเพิ่มขึ้น โดยเพิ่มการรบกวนจากภายนอกหลังจากวัดค่าความเหมาะสมของทุกบิตในโครโมโซมเรียบร้อยแล้ว การวัดค่าความเหมาะสมของปัญหา OneMax Noisy แสดงในสมการ (2)

$$f(x) = \sum_{i=1}^{l} x_i + \sigma_N^2 \tag{2}$$

โดย Philippe Giguere และ David E. Goldberg ได้เสนอแนวคิดในการหาค่าความแปรปรวนจากการรบกวนแบบเกาซ์เซียนในปัญหา OneMax (Giguere and Goldberg, 1998) ดังสมการ (3)

$$\sigma_N^2 = \frac{l^2 p(t)(1-p(t))}{n}\left[\frac{l-n}{l-1}\right] \tag{3}$$

เมื่อ l คือขนาดของโครโมโซม

$p(t)$ คือสัดส่วนของค่าความเหมาะสมของบิตที่สุ่มได้ต่อค่าความเหมาะสมของโครโมโซม ในรุ่นที่ t

n คือจำนวนบิตที่สุ่ม

5.3 ปัญหา Royal Road

ปัญหา RoyalRoad (Melanie Mitchell *et al.*, 1992) เป็นปัญหาที่จำลองความสัมพันธ์ของบิต จะแบ่งโครโมโซมออกเป็นบล็อกจากนั้นวัดค่าความเหมาะสมภายในบล็อก ถ้าทุกบิตในบล็อกมีค่าเท่ากับหนึ่ง บล็อกนั้นจะได้ค่าเท่ากับ 1 ถ้ามีอย่างน้อยหนึ่งบิตที่มีค่าเท่ากับ 0 บล็อกนั้นจะมีค่าความเหมาะสมเป็น 0 การวัดค่าความเหมาะสมแสดงในสมการ (4)

$$f(x) = \left(\sum_{i=1}^{k} \delta_i(x) \right) O(s), \quad \text{เมื่อ } \delta_i(x) = \begin{cases} 1 \text{ ถ้าทุกบิตในบล็อกเป็น } 1 \\ 0 \text{ มีอย่างน้อย } 1 \text{ บิตเป็น } 0 \end{cases} \tag{4}$$

เมื่อ k คือจำนวนบล็อก

และ $O(1) = O(2) = O(3) = \ldots = O(k)$ = จำนวนบิตในบล็อก

ยกตัวอย่างโครโมโซมขนาด 64 บิต แบ่งเป็น 8 บล็อก (k = 8) จะได้จำนวนบิตในบล็อกเท่ากับ 8 ($O(s)$ = 8) ถ้าโครโมโซมนี้มี 2 บล็อกที่มีค่าเป็นหนึ่งทุกบิต ค่าความเหมาะสมของโครโมโซมนี้คือ 2 × 8 = 16

งานวิจัยที่เกี่ยวข้อง

งานวิจัยนี้มุ่งศึกษาการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิก ซึ่งจากการศึกษาพบว่ายังไม่มีงานวิจัยที่นำขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์มาทำงานแบบขนานบนหน่วยประมวลผลกราฟิก ดังนั้นส่วนของการศึกษางานวิจัยที่เกี่ยวข้องจึงศึกษางานในส่วนที่เกี่ยวข้อง 2 ส่วนคือ ศึกษาการพัฒนาขั้นตอนวิธีเชิงพันธุกรรมบนหน่วยประมวลผลกราฟิก และศึกษาการพัฒนาขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน ซึ่งมีรายละเอียดดังนี้

1. งานวิจัยที่เกี่ยวกับการพัฒนาขั้นตอนวิธีเชิงพันธุกรรมบนหน่วยประมวลผลกราฟิก

งานวิจัยของ Qizhi Yu และคณะ (Yu *et al*., 2005) นำเสนอการทำงานของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบละเอียดบนหน่วยประมวลผลกราฟิก โดยใช้หน่วยความจำประเภท texture แบ่งการวัดค่าความเหมาะสมออกเป็นส่วนย่อย ๆ เรียกว่า fragment program นำค่าที่ได้หลังจากการทำงานของ fragment program ไปเก็บไว้ใน fitness texture การคัดเลือก จะพิจารณาจากค่าความเหมาะสมใน fitness texture โดยพิจารณาเฉพาะโครโมโซมที่อยู่ใกล้เคียงกัน ครั้งละ 5 โครโมโซมเท่านั้น ผลการทดลองพบว่าเมื่อขนาดของประชากรเท่ากับ 32 × 32 บิต การทำงานบนหน่วยประมวลผลกราฟิกทำงานได้ช้ากว่าการทำงานบนหน่วยประมวลผลกลางเนื่องมาจากสูญเสียในการทำงานแบบสายท่อ (pipeline) ของหน่วยประมวลผลกราฟิก แต่เมื่อเพิ่มขนาดของประชากรพบว่าหน่วยประมวลผลกราฟิกสามารถทำงานได้เร็วกว่าถึง 20 เท่าเมื่อทดสอบบนประชากรขนาด 512 × 512 บิต

งานวิจัยของ Jian-Ming Li และคณะ (Li *et al*., 2007) นำเสนอการทำงานของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบละเอียดที่มีตัวดำเนินการในระดับบิต (bit-operator) ที่เหมาะสมกับโครโมโซมของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบละเอียดบนหน่วยประมวลผลกราฟิกมากขึ้น โดยจัดเก็บโครโมโซมลงบนหน่วยประมวลผลแบบ texture ทดลองกับปัญหา 3 ปัญหาเพื่อเปรียบเทียบประสิทธิภาพได้แก่ ปัญหา Schwefel function, Shaffer’s function, และ Camel function พบว่าการทำงานบนหน่วยประมวลผลกราฟิกมีโอกาสพบคำตอบสูงกว่าทำงานบนหน่วยประมวลผลกลางและสามารถทำงานได้เร็วขึ้น โดยเฉพาะกับประชากรขนาด 1000 พบว่าสามารถทำงานได้เร็วขึ้นถึง 21, 50.8, และ 73.6 เท่าในการวัดบน Schwefel function, Shaffer’s function, และ Camel function ตามลำดับ

งานวิจัยของ Ogier Maitre และ Laurent A. Baumes (Maitre and Baumes, 2009) นำเสนอการพัฒนาขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบเกาะไอซ์แลนด์บนหน่วยประมวลผลกราฟิกซึ่งการดำเนินการทางพันธุกรรม ได้แก่ การคัดเลือก, การไขว้เปลี่ยน และการกลายพันธ์จะทำบนหน่วยประมวลผลหลักแต่ส่งประชากรไปวัดค่าความเหมาะสมบนหน่วยประมวลผลกราฟิกทดสอบ

บนฟังก์ชัน Weierstrass-Mandelbrot และปัญหาในการจับคู่โครงสร้างของอะตอม พบว่าการทำงานที่วัดค่าความเหมาะสมบนหน่วยประมวลผลกราฟิกนั้นทำงานได้เร็วกว่า ทั้งการทำงานบนหน่วยประมวลผลกราฟิกรุ่น GTX 260 และรุ่น 8800GTX และสามารถนำโปรแกรมที่พัฒนาบน 8800GTX ไปทำงานบน GTX 260 ได้แม้ว่าสถาปัตยกรรมของ GTX 260 และ 8800GTX จะแตกต่างกันโดยใช้แบบจำลองการเขียนโปรแกรม CUDA และพบว่าขนาดของโครโมโซมมีผลต่อการส่งข้อมูลไปวัดค่าความเหมาะสมบนหน่วยประมวลผลกราฟิก

งานวิจัยของ Thé Van Luong และคณะ (Luong *et al*., 2010) ได้ทดลองเปรียบเทียบประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานแบบเกาะไอซ์แลนด์ที่มีโครงสร้างแตกต่างกัน 3 แบบคือโครงสร้างที่ดำเนินการทางพันธุกรรมแบบอนุกรมบนหน่วยประมวลผลกลางแต่วัดค่าความเหมาะสมแบบขนานบนหน่วยประมวลผลกราฟิก, โครงสร้างที่ดำเนินการทางพันธุกรรมทั้งหมดบนหน่วยประมวลผลกราฟิก โดยจัดเก็บประชากรทั้งหมดลงบนหน่วยความจำแบบครอบคลุม, และโครงสร้างที่ดำเนินการทางพันธุกรรมทั้งหมดบนหน่วยประมวลผลกราฟิกแต่จัดเก็บประชากรทั้งหมดลงบนหน่วยความจำแบบแบ่งส่วนทดสอบกับฟังก์ชัน Weierstrass-Mandelbrot พบว่าการทำงานบนหน่วยประมวลผลกราฟิกสามารถทำงานได้เร็วขึ้นถึง 2000 เท่า เมื่อเทียบกับการทำงานบนหน่วยประมวลผลกลาง และการใช้หน่วยความจำแบบแบ่งส่วนใช้เวลาในการทำงานน้อยกว่าการใช้หน่วยความจำแบบครอบคลุม

2. งานวิจัยที่เกี่ยวกับการพัฒนาขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน

งานวิจัยของ Fernando G. Lobo และคณะ (Lobo *et al*., 2004) เสนอสถาปัตยกรรมของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานแบบ manager-worker การทำงานมีลักษณะคล้ายกับการทำงานแบบขนานแบบ master-slave ของขั้นตอนวิธีเชิงพันธุกรรม แต่การทำงานแบบขนานของ manager-worker จะสามารถมีจำนวน worker ได้ไม่จำกัด โดยการทำงานของแต่ละ worker จะเริ่มต้นและสิ้นสุดใน worker จะไม่มีการเชื่อมต่อระหว่าง worker การเชื่อมต่อจะเกิดขึ้นระหว่าง manager กับ worker เท่านั้น ซึ่งวิธีนี้จะช่วยลดการขัดจังหวะในระหว่างการทำงาน แต่ละ worker จะสร้างประชากร m ตัว และวัดค่าความเหมาะสมและปรับเปลี่ยนเวกเตอร์ความน่าจะเป็นใน worker จากนั้นจึงส่งค่าไปยัง manager การทดลองทดสอบกับปัญหา 3-bit trap function พบว่าค่า *m* มีผลต่อประสิทธิภาพในการทำงานโดยค่า *m* น้อยจะทำให้การเชื่อมต่อระหว่าง manager-worker มากขึ้น แต่ถ้าค่า m สูงจะทำให้ประสิทธิภาพลดลงเนื่องจากแต่ละ worker ทำงานมากขึ้น นอกจากนั้นยังพบว่าค่า m มีความสัมพันธ์กับจำนวนของหน่วยประมวลผลที่ใช้ และเมื่อเพิ่มจำนวนหน่วยประมวลผลขึ้นสองเท่า จะลดค่าเฉลี่ยของเวลาในการวัดค่าความเหมาะสมลงครึ่งหนึ่ง

งานวิจัยของ Kumara Sastry และคณะ (Sastry *et al*., 2007) เสนอการพัฒนาขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานแบบ master-slave โดยการกระจายความน่าจะเป็นไปยังหน่วยประมวลผลย่อยที่เป็น slave แต่ละหน่วยประมวลผลย่อยจะดำเนินการกับเวกเตอร์

ความน่าจะเป็นขนาด $\frac{1}{n_p}$ เพื่อสร้าง individuals จากค่าเวกเตอร์ความน่าจะเป็นนี้ (เมื่อ l คือขนาดของโครโมโซมและ n_p คือจำนวนหน่วยประมวลผลย่อยที่ใช้) จากนั้นรวบรวม individuals จากแต่ละหน่วยประมวลผลย่อยและวัดค่าความเหมาะสมแบบขนานโดยให้แต่ละหน่วยประมวลผลย่อยวัดค่าความเหมาะสมในแต่ละส่วนและรวบรวมจากค่าความเหมาะสมจากทุก ๆ หน่วยประมวลผล จากนั้นกระจายค่าความเหมาะสมที่ได้ไปยังหน่วยประมวลผลย่อยทุกตัวเพื่อเลือกตัวที่เหมาะสมและเปลี่ยนแปลงแก้ไขค่าเวกเตอร์ความน่าจะเป็นตาม individuals ตัวที่เหมาะสมมากกว่า ทดลองกับปัญหา OneMax noisy โดยใช้การรบกวนแบบ Gaussian พบว่าสามารถแก้ปัญหาขนาด 33 ล้านบิตได้ถ้ามีการผ่อนปรนการบรรจบกันของคำตอบ และสามารถแก้ปัญหาขนาด 1,100 ล้านบิตได้ในปัญหาทดสอบ

สรุปงานวิจัยที่เกี่ยวข้องเรียงตามประเภทงานวิจัย (ไทย-ต่างประเทศ) และเรียงตามปีที่พิมพ์

ที่	ปี	ชื่อบทความ	สิ่งที่นำเสนอ	ผลการทดลอง
1	2004	An Architecture for Massive Parallelization of the Compact Genetic Algorithm	การทำงานขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน ที่เรียกว่า manager-worker	เมื่อเพิ่มจำนวนหน่วยประมวลผลขึ้นสองเท่า จะทำให้เวลาในการวัดค่าความเหมาะสมลดลงครึ่งหนึ่ง
2	2005	Parallel Genetic Algorithms on Programmable Graphics Hardware	การทำงานของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานประเภท fine-grained บนหน่วยประมวลผลกราฟิกโดยใช้หน่วยความจำประเภท texture	หน่วยประมวลผลกราฟิกสามารถทำงานได้เร็วกว่าถึง 20 เท่าเมื่อทดสอบบนประชากรขนาด 512 × 512 บิต
3	2007	Towards billion bit optimization via parallel estimation of distribution algorithm	การพัฒนาขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานแบบ master-slave โดยการกระจายความน่าจะเป็นไปยังหน่วยประมวลผลย่อย	สามารถแก้ปัญหาขนาด 33 ล้านบิตได้ ถ้ามีการผ่อนปรนการบรรจบกันของคำตอบ และสามารถแก้ปัญหาขนาด 1,100 ล้านบิตได้ในปัญหาทดสอบ

ที่	ปี	ชื่อบทความ	สิ่งที่นำเสนอ	ผลการทดลอง
4	2007	An efficient fine-grained parallel genetic algorithm based on GPU-accelerated	การทำงานของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานประเภท fine-grained ที่มีตัวดำเนินการบิต (bit-operator) ที่เหมาะสมกับโครโมโซมของ ขั้นตอนวิธีเชิงพันธุกรรมแบบขนานประเภท fine-grained บนหน่วยประมวลผลกราฟิก	การทำงานบนหน่วยประมวลผลกราฟิกมีโอกาสพบคำตอบสูงกว่าทำงานบนหน่วยประมวลผลและให้ประสิทธิภาพในด้านเวลาในการทำงานสูงกว่า
5	2009	Coarse grain parallelization of evolutionary algorithms on GPGPU cards with EASEA	การพัฒนาขั้นตอนวิธีเชิงพันธุกรรมแบบขนานประเภทเกาะไอซ์แลนด์บนหน่วยประมวลผลกราฟิกและทดสอบการทำงานของหน่วยประมวลผลกราฟิกที่มีสถาปัตยกรรมแตกต่างกันโดยใช้ CUDA	การทำงานที่วัดค่าความเหมาะสมบนหน่วยประมวลผลกราฟิกนั้นให้ประสิทธิภาพที่ดีกว่า ทั้งการทำงานบนหน่วยประมวลผลกราฟิกรุ่น GTX 260 และรุ่น 8800GTX ซึ่งทั้งสองรุ่นสามารถใช้โปรแกรมเดียวกันที่พัฒนาโดย CUDA ได้
6	2010	GPU-based island model for evolutionary algorithms	เปรียบเทียบประสิทธิภาพเกาะไอซ์แลนด์ที่มีโครงสร้างแตกต่างกัน 3 โครงสร้าง	การทำงานบนหน่วยประมวลผลกราฟิกสามารถทำงานได้เร็วขึ้นถึง 2000 เท่า และการใช้หน่วยความจำแบบแบ่งส่วนใช้เวลาในการทำงานน้อยกว่าการใช้หน่วยความจำแบบครอบคลุม

อุปกรณ์และวิธีการ

อุปกรณ์

1. Hardware:

1.1 Intel® Core 2 Quad CPU Q8200 2.33 GHz, 2.75 GB of RAM ซึ่งเป็นหน่วยประมวลผลที่ออกวางจำหน่ายในปี ค.ศ. 2008

1.2 NVIDIA GeForce GT 240 เป็นหน่วยประมวลผลกราฟิกที่ออกวางจำหน่ายในปี ค.ศ. 2008 เช่นเดียวกัน มีคุณสมบัติดังตารางที่ 1

ตารางที่ 1 คุณสมบัติของ NVIDIA GeForce GT 240 (NVIDIA, 2012b)

รายการ	คุณสมบัติ
จำนวนหน่วยประมวลผล	96 core
Clock Rate ของหน่วยประมวลผล	1340 MHz
แบนด์วิดท์ของหน่วยความจำ	54.4 GB/sec
จำนวนเธรดสูงสุดต่อบล็อก	512 เธรด
จำนวนบล็อกสูงสุดต่อตาราง(grid)	65,535 บล็อก

2. Software:

2.1 CUDA Toolkit 4.1
2.2 Microsoft Visual C++ 2008 Express Edition
2.3 Operating System: Microsoft Windows XP

วิธีการ

จุดมุ่งหมายของงานวิจัยนี้คือ การเพิ่มความเร็วในการทำงานให้กับขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ โดยการพัฒนาขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิก ที่มาและแนวคิดเพื่อพัฒนาประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน แสดงดังต่อไปนี้

1. ที่มาของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานโดยใช้แบบจำลองการเขียนโปรแกรม CUDA

การทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบอนุกรมจะทำงานโดยสร้าง individuals *a, b* จากเวกเตอร์ความน่าจะเป็น *p* จากนั้นวัดค่าความเหมาะสมของ individuals *a, b* และปรับเปลี่ยนค่าเวกเตอร์ความน่าจะเป็นตาม individuals ตัวที่ชนะ ซึ่งแต่ละขั้นตอนจะใช้เวลาในการทำงานแบบอนุกรมขั้นตอนละ $O(l)$ เมื่อ l คือความยาวของโครโมโซมซึ่งเท่ากับขนาดของเวกเตอร์ความน่าจะเป็น จะเห็นว่าเวลาที่ใช้ในการทำงานจะเพิ่มขึ้นตามขนาดของเวกเตอร์ความน่าจะเป็น ดังนั้นในปัญหามีขนาดใหญ่ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบอนุกรมจะใช้เวลาในการทำงานมาก

การศึกษาการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานโดยใช้แบบจำลองการเขียนโปรแกรม CUDA ซึ่งเป็นแบบจำลองที่สามารถทำงานบนหน่วยประมวลผลกราฟิก โดยแต่ละเธรดสามารถดำเนินการกับคำสั่งเดียวกันได้พร้อมกัน (single-instruction, multiple-thread-SIMT) จึงเหมาะสมกับการนำมาพัฒนาประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ เนื่องจากแต่ละบิตของเวกเตอร์ความน่าจะเป็นสามารถทำงานได้อย่างอิสระ

2. การทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิก

2.1 การหยุดรอการทำงานระหว่างบล็อก (Synchronization)

การทำงานของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานคอมแพคท์โดยใช้แบบจำลองการเขียนโปรแกรม CUDA บนหน่วยประมวลผลกราฟิกมีแนวคิดโดยให้แต่ละเธรดดำเนินการกับแต่ละบิตของเวกเตอร์ความน่าจะเป็น ตั้งแต่การนำมาสร้าง individuals *a, b* การวัดค่าความเหมาะสม การปรับเปลี่ยนค่าเวกเตอร์ความน่าจะเป็น จนกระทั่งพบว่าทุกบิตของเวกเตอร์ความน่าจะเป็นถูกแปรสภาพเป็น 0 หรือ 1 การประมวลผลปัญหาขนาดใหญ่ที่มีจำนวนบิตของเวกเตอร์ความน่าจะเป็นมากจึงต้องใช้เธรดจำนวนมากเช่นเดียวกัน แต่ CUDA จำกัดจำนวนของเธรดในแต่ละบล็อกไว้เพียง 512 เธรดต่อบล็อกเท่านั้น ดังนั้นในปัญหาขนาดใหญ่จึงต้องใช้จำนวนบล็อกมากกว่า 1 บล็อก

ผู้วิจัยพบว่าเกิดปัญหาในการหยุดรอเพื่อทำงานพร้อมกันระหว่างบล็อก เนื่องจาก CUDA มีคำสั่ง __syncthreads() สำหรับการหยุดรอภายในบล็อกแต่ไม่มีคำสั่งในการหยุดรอระหว่างบล็อก ซึ่งการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานต้องการการหยุดรอก่อนการทำงานในขั้นตอนการหา individuals ตัวที่ชนะและการตรวจสอบการแปรสภาพของเวกเตอร์ความน่าจะเป็นว่าเป็น 0 หรือ 1 ครบทุกบิตหรือไม่ ซึ่งการหยุดรอเพื่อให้มั่นใจว่าทุกบล็อกทำงานเสร็จอาจทำได้โดยการส่งค่า individual *a, b* มาวัดค่าความเหมาะสมและตรวจสอบการแปรสภาพของเวกเตอร์ความน่าจะเป็นบนหน่วยประมวลผลกลาง จากนั้นจึงให้หน่วยประผลกราฟิกทำงานในขั้นตอนต่อไป ซึ่งการทำงานในลักษณะนี้ต้องใช้ kernel มากกว่า 1 kernel และการกลับมาทำงานบนหน่วยประมวลผลกลางจะส่งผลเสียต่อเวลาในการทำงาน ทั้งในส่วนของการหยุดรอและการส่งข้อมูลกลับไปมาระหว่างหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิก (Zhang Dan *et al*., 2010)

ผู้วิจัยจึงเสนอแนวคิดในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานที่สามารถแก้ปัญหาการหยุดรอการทำงานระหว่างบล็อกบนหน่วยประมวลผลกลาง ซึ่งใช้ kernel เพียงหนึ่ง kernel โดยเพิ่มเวกเตอร์สำหรับเก็บค่าความน่าจะเป็นในกรณีที่ individuals *a* เป็นผู้ชนะและเวกเตอร์สำหรับเก็บความน่าจะเป็นในกรณีที่ individuals *b* เป็นผู้ชนะ ซึ่งการปรับค่าเวกเตอร์นี้สามารถปรับได้ภายในบล็อกที่วัดค่าความเหมาะสมเรียบร้อยแล้วโดยไม่จำเป็นต้องรอให้ทุกบล็อกทำงานเสร็จสิ้น

2.2 การสร้างเลขสุ่มบนหน่วยประมวลผลกราฟิก

การทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์มีการสุ่มค่าเพื่อสร้าง individuals a และ b แต่แบบจำลองการเขียนโปรแกรม CUDA ในรุ่นแรก ๆ (CUDA Tookit 1.1 – 3.1) ไม่มีฟังก์ชันในการสร้างเลขสุ่มดังนั้นจึงต้องสร้างฟังก์ชันในการสุ่มค่าบน device เอง โดยใช้ฟังก์ชัน Linear Congruential Generator (LCG) ดังในสมการ (5) ซึ่งมีการกำหนดค่าเริ่มต้น x_n ให้กับฟังก์ชันการสุ่ม เรียกค่าเริ่มต้นนี้ว่า seed การกำหนดค่าจะทำบน host ดังขั้นตอนที่ 2 ในภาพที่ 11 การส่ง seed นี้จะทำเพียงครั้งเดียวหลังจากนั้นจะเก็บค่า seed ใหม่ลงบนหน่วยความจำแบบครอบคลุมของหน่วยประมวลผลกราฟิก

$$X_{n+1} = (ax_n + c) \bmod m \qquad (5)$$

จากสมการ (5) กำหนดค่า a ในการทดลอง เท่ากับ 214013 ค่า c เท่ากับ 2531011 และ m เท่ากับ 1 กับทุกปัญหาการทดลอง

จนกระทั่งประมาณเดือนสิงหาคม ปี ค.ศ. 2010 แบบจำลองการเขียนโปรแกรม CUDA รุ่น CUDA Tookit 3.2 ได้บรรจุชุดคำสั่งสำหรับการสร้างเลขสุ่ม ที่เรียกว่า CURAND Library ขึ้น (NVIDIA, 2010b) ดังนั้นจึงเพิ่มการทดลองเพื่อศึกษาประสิทธิภาพด้านเวลาในการทำงานของ

ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานทั้งรูปแบบที่ทำงานด้วยฟังก์ชัน CURAND() และรูปแบบที่ใช้ฟังก์ชันการสุ่มที่สร้างขึ้นเองตามสมการ (5)

2.3 ขั้นตอนการทำงาน

การทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิกจะทำงานทั้งในส่วนของ host และ device ขั้นตอนที่ทำงานบน host จะใช้คำว่า Host นำหน้า และขั้นตอนที่ทำงานบน device จะใช้ คำว่า Device นำหน้า การทำงานแสดงดังนี้

(1) Host: กำหนดค่าเวกเตอร์ความน่าจะเป็นเริ่มต้น โดยให้เวกเตอร์ความน่าจะเป็น $p[i] = 0.5$ เมื่อ $i \in 1 \ldots l$ โดย l คือขนาดของปัญหา (Problem size)

(2) Host: กำหนดค่า seed ในการสร้างเลขสุ่มบนหน่วยประมวลผลกลางซึ่งต้องกำหนดเป็นแถวลำดับของ seed ที่มีขนาดเท่ากับขนาดของปัญหาโดยให้ seed ทุก ๆ ตำแหน่งมีค่าเท่ากับดรรชนีของเวกเตอร์ความน่าจะเป็นในตำแหน่งนั้น

(3) Host: ส่งเวกเตอร์ของความน่าจะเป็นและ seed ไปยัง device ซึ่ง device จะจัดเก็บข้อมูลทั้งหมดลงบนหน่วยความจำแบบครอบคลุม

(4) Device: คัดลอกเวกเตอร์ของความน่าจะเป็นลงบน *s_prob* ซึ่งจัดเก็บในหน่วยความจำแบบแบ่งส่วนเพื่อให้เข้าถึงได้เร็วขึ้นในครั้งต่อไป และสร้าง individuals 2 ตัวคือ s_a, *s_b* โดยใช้ค่าจาก *d_seed* และเวกเตอร์ความน่าจะเป็น *s_prob*

(5) Device: วัดค่าความเหมาะสมของ *s_a* และ *s_b* คำนวณค่า *fitnessA* และ *fitnessB* ซึ่งจัดเก็บผลรวมของค่าความเหมาะสมในบล็อก และเก็บค่าความเหมาะสมของ *a* และ *b* ในแต่ละบล็อกลงบน *d_part_fitnessA* และ *d_part_fitnessB* ตามดรรชนีของหมายเลขบล็อก

(6) Device: คำนวณและจัดเก็บความน่าจะเป็นในกรณีที่ *a* เป็นผู้ชนะลงใน *s_probA* และจัดเก็บความน่าจะเป็นถ้า *b* เป็นผู้ชนะลงใน *s_probB*

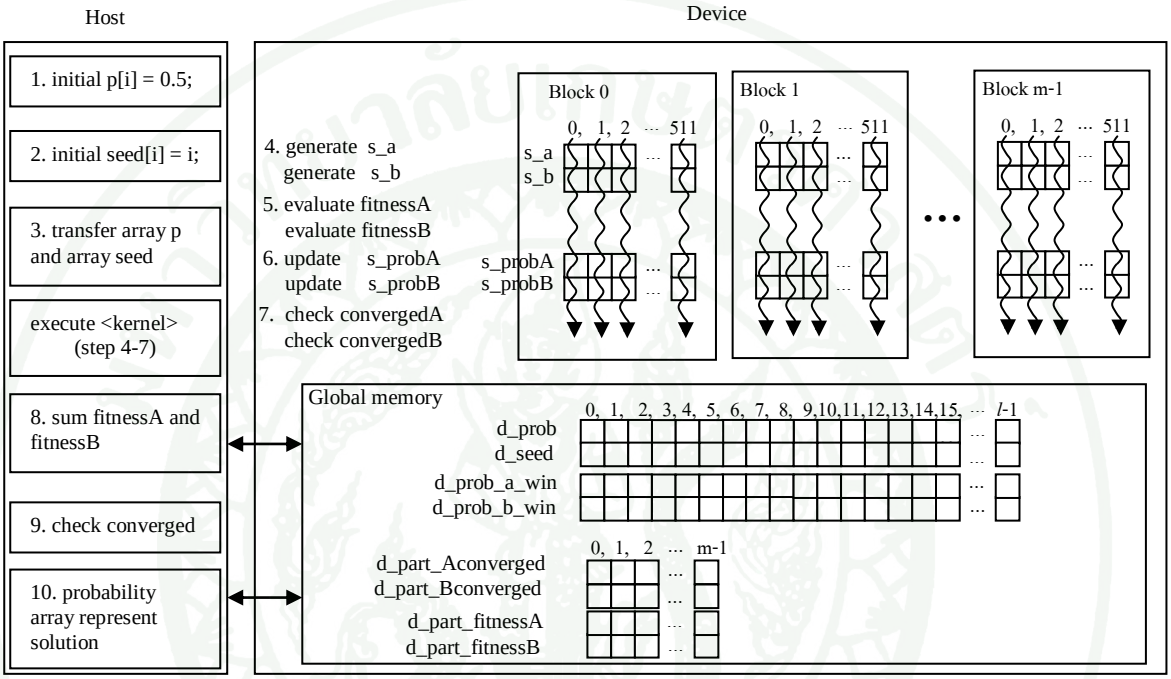
(7) Device: ตรวจสอบการถูกแปรสภาพของ *s_probA* และ *s_probB* ในบล็อกจากนั้นเก็บค่าของแต่ละบล็อกลงบน *d_part_convergedA* และ *d_part_convergedB* ตามดรรชนีของหมายเลขบล็อก

(8) Host: ส่ง *d_part_fitnessA* และ *d_part_fitnessB* จาก device ไปยัง host และคำนวณค่าความเหมาะสมรวมของ individuals *a* และ *b* เปรียบเทียบค่าความเหมาะสมของ *a* และ *b* เพื่อหาผู้ชนะ

(9) Host: ส่ง *d_part_convergedA* เมื่อ a เป็นผู้ชนะ หรือ *d_part_convergedB* เมื่อ *b* เป็นผู้ชนะไปยังหน่วยประมวลผลกลาง ตรวจสอบการแปรสภาพของทุกบล็อกถ้าพบว่ามีบล็อกใด ที่ค่าของเวกเตอร์ความน่าจะเป็นไม่ใช่ 0 หรือ 1 ให้กลับไปสู่ขั้นตอนที่ 4 โดยเปลี่ยนเวกเตอร์ความน่าจะเป็นให้เป็นเวกเตอร์ความน่าจะเป็นของตัวที่ชนะ

(10) Host: ถ้าทุก blocks ถูกแปรสภาพให้เป็น 0 หรือ 1 ให้ส่งเวกเตอร์ความน่าจะเป็นจาก device ไปยัง host ซึ่งเวกเตอร์ความน่าจะเป็นจะแสดงคำตอบที่ต้องการ

แต่ละรอบจะสลับการทำงานระหว่าง host บนหน่วยประมวลผลกลางและ device บนหน่วยประมวลผลกราฟิกเพียงครั้งเดียว (เรียกใช้ kernel เพียงครั้งเดียว) การทำงานบน host เป็นแบบอนุกรม การทำงานแบบขนานจะเกิดขึ้นบน device เท่านั้น ซึ่งสามารถแยกการทำงานของ host และ device รวมทั้งการจัดเก็บตัวแปรในหน่วยความจำ ได้ดังภาพที่ 11



ภาพที่ 11 การเชื่อมต่อระหว่าง host และ device (เมื่อ l คือขนาดของปัญหาและ m คือจำนวนบล็อกที่ใช้)

3. แนวทางการทดลอง

การทดลองแบ่งออกเป็น 3 กลุ่ม คือการทดลองเพื่อเปรียบเทียบประสิทธิภาพการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิก (parallel compact Genetic Algorithm: pcGA) และขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบอนุกรมที่ทำงานบนหน่วยประมวลผลกลางเพียงอย่างเดียว (serial compact Genetic Algorithm: scGA), การทดลองเพื่อเปรียบเทียบประสิทธิภาพเมื่อมีการวัดค่าความเหมาะสมที่แตกต่างกัน และการทดลองเพื่อเปรียบเทียบประสิทธิภาพเมื่อจำนวนเธรดในบล็อกมีขนาดแตกต่างกัน ทั้ง 3 กลุ่มการทดลอง ทดสอบกับปัญหาทั้ง 3 แบบคือ ปัญหา OneMax, ปัญหา OneMax Noisy และปัญหา Royal Road

3.1 การทดลองเพื่อเปรียบเทียบประสิทธิภาพการทำงานแบบขนานและแบบอนุกรม

3.1.1 การทดลองเพื่อเปรียบเทียบเวลาในการทำงานเมื่อใช้ฟังก์ชันการสุ่มที่แตกต่างกัน

การทดลองเพื่อเปรียบเทียบเวลาในการทำงานเมื่อใช้ฟังก์ชันการสุ่มที่แตกต่างกัน เป็นการเปรียบเทียบเวลาในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ที่ทำงานแบบขนานบนหน่วยประมวลผลกราฟิกที่ใช้ฟังก์ชันการสุ่มที่สร้างขึ้นดังสมการ (5) และเวลาในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ที่ทำงานแบบอนุกรมบนหน่วยประมวลผลกลางเพียงอย่างเดียว โดยใช้ฟังก์ชันการสุ่ม rand() ของ Visual C++

ทุกการทดลองทดสอบกับปัญหาทั้ง 3 แบบคือ ปัญหา OneMax, ปัญหา OneMax Noisy และปัญหา Royal Road แต่ละการทดลองทำซ้ำ 20 ครั้ง พารามิเตอร์ในแต่ละปัญหาทดสอบกำหนดดังตารางที่ 2 และ 3

ตารางที่ 2 พารามิเตอร์ที่ใช้ทดลองแก้ปัญหา OneMax และปัญหา OneMax Noisy

พารามิเตอร์	**OneMax และ OneMax Noisy**
จำนวนประชากร *(n)*	512 ตัว
ขนาดของปัญหา *(l)*	1 × 512, 2 × 512, 4 × 512, 8 × 512, 16 × 512, 32 × 512, 64 × 512,128 × 512, 256 × 512, 512 × 512 บิต
จำนวนเธรดในแต่ละบล็อก	512 เธรด
จำนวนครั้งในการวัดค่าความเหมาะสม	1,000,000 ครั้ง

3.1.2 เปรียบเทียบเวลาในการทำงานเมื่อใช้ฟังก์ชันการสุ่มเหมือนกัน

เพื่อลดความลำเอียงที่อาจเกิดจากการใช้ฟังก์ชันการสุ่มจึงออกแบบการทดลองเพื่อเปรียบเทียบเวลาในการทำงานเมื่อใช้ฟังก์ชันการสุ่มดังสมการ (5) เหมือนกันของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ที่ทำงานแบบขนานบนหน่วยประมวลผลกราฟิก และขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ที่ทำงานแบบอนุกรมบนหน่วยประมวลผลกลางเพียงอย่างเดียว

ทุกการทดลองทดสอบกับปัญหาทั้ง 3 แบบคือ ปัญหา OneMax, ปัญหา OneMax Noisy และ ปัญหา Royal Road แต่ละการทดลองทำซ้ำ 10 ครั้ง พารามิเตอร์ในแต่ละปัญหาทดสอบกำหนดดังตารางที่ 2 และ 3

ตารางที่ 3 พารามิเตอร์ที่ใช้ทดลองกับแก้ปัญหา Royal Road

พารามิเตอร์	Royal Road
จำนวนประชากร *(n)*	512 ตัว
ขนาดของปัญหา *(l)*	1 × 512, 2 × 512, 4 × 512, 8 × 512, 16 × 512, 32 × 512, 64 × 512,128 × 512, 256 × 512, 512 × 512 บิต
จำนวนเธรดในแต่ละบล็อก	512 เธรด
ขนาดของบล็อกในปัญหา Royal Road	512 บิต
จำนวนครั้งในการวัดค่าความเหมาะสม	3,000,000 ครั้ง (สำหรับปัญหาขนาด 1 x 512 ถึง 32 × 512 บิต) 6,000,000 ครั้ง (สำหรับปัญหาขนาด 64 × 512 บิตขึ้นไป)

3.1.3 เปรียบเทียบเวลาในการทำงานโดยใช้ฟังก์ชันการสุ่ม rand() และ CURAND()

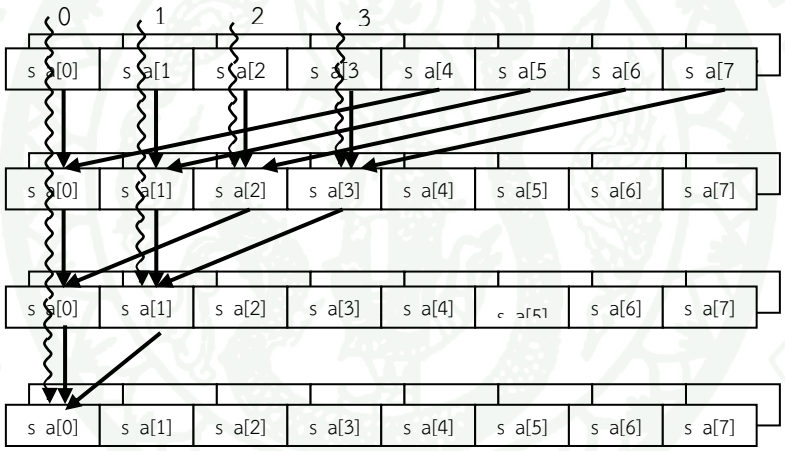
การทดลองโดยใช้ฟังก์ชันการสุ่ม rand() และ CURAND() เป็นการทดลองเพื่อเปรียบเทียบประสิทธิภาพของการสุ่มในชุดคำสั่ง CURAND และประสิทธิภาพของการสุ่มโดยใช้ฟังก์ชัน rand() บน visual C++ โดยการเปรียบเทียบการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ที่ทำงานแบบขนานบนหน่วยประมวลผลกราฟิก ซึ่งจะใช้การสุ่มด้วยชุดคำสั่ง CURAND และเวลาในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ที่ทำงานแบบอนุกรมบนหน่วยประมวลผลกลางเพียงอย่างเดียวที่ใช้การสุ่มด้วยฟังก์ชันการสุ่ม rand() ของ visual C++

ทุกการทดลองทดสอบกับปัญหาทั้ง 3 แบบคือ ปัญหา OneMax, ปัญหา OneMax Noisy และ ปัญหา Royal Road แต่ละการทดลองทำซ้ำ 20 ครั้ง พารามิเตอร์ในแต่ละปัญหาทดสอบกำหนดดังตารางที่ 2 และ 3

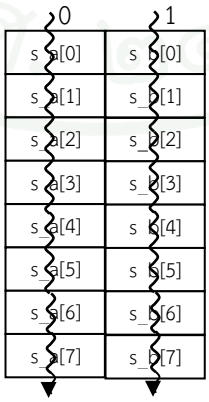
3.2 การทดลองเพื่อเปรียบเทียบประสิทธิภาพเมื่อมีการวัดค่าความเหมาะสมที่แตกต่างกัน

การทดลองเพื่อเปรียบเทียบประสิทธิภาพเมื่อมีการวัดค่าความเหมาะสมที่แตกต่างกัน เป็นการทดลองเพื่อเปรียบการทำงานของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานบนหน่วยประมวลผล 2 รูปแบบ คือ แบบที่มีการวัดค่าความเหมาะสมด้วยวิธีลดทอน (reduction) (John Nickolls *et al*., 2010) มีรูปแบบการทำงานดังภาพที่ 12 และขั้นตอนวิธีเชิงพันธุกรรมแบบขนานบนหน่วยประมวลผลกราฟิกแบบที่มีการวัดค่าความเหมาะสมโดยใช้เธรดเดียววัดค่าความเหมาะสมดังภาพที่ 13 และทั้งสองแบบใช้ฟังก์ชันการสุ่มตามสมการ (5)

ทุกการทดลองทดสอบกับปัญหาทั้ง 3 แบบคือ ปัญหา OneMax, ปัญหา OneMax Noisy และปัญหา Royal Road แต่ละการทดลองทำซ้ำ 20 ครั้ง พารามิเตอร์ในแต่ละปัญหาทดสอบกำหนดดังตารางที่ 2 และ 3



ภาพที่ 12 การวัดค่าความเหมาะสมด้วยวิธีลดทอน (reduction)



ภาพที่ 13 การวัดค่าความเหมาะสมโดยใช้เธรดเดียววัดค่าความเหมาะสมรวมของบล็อก

3.3 การทดลองเพื่อเปรียบเทียบประสิทธิภาพเมื่อจำนวนเธรดในบล็อกแตกต่างกัน

การทดลองเพื่อเปรียบเทียบประสิทธิภาพเมื่อจำนวนเธรดในบล็อกแตกต่างกันเป็นการทดลองเพื่อเปรียบเทียบประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมแบบขนานบนหน่วยประมวลผลกราฟิกแบบที่ใช้การวัดค่าความเหมาะสมแบบเธรดเดียวกับแบบที่ใช้การวัดค่าความเหมาะสมแบบลดทอนที่มีจำนวนเธรดในบล็อกแตกต่างกัน เพื่อศึกษาว่าจำนวนเธรดในบล็อกส่งผลต่อประสิทธิภาพในการทำงานอย่างไร

การทดลองทดสอบกับปัญหาทั้ง 3 แบบคือ ปัญหา OneMax, ปัญหา OneMax Noisy และ ปัญหา Royal Road แต่ละการทดลองทำซ้ำ 20 ครั้ง พารามิเตอร์ในแต่ละปัญหาทดสอบกำหนดดังนี้

ตารางที่ 4 พารามิเตอร์ที่ใช้ทดลองแก้ปัญหา OneMax และปัญหา OneMax Noisy

พารามิเตอร์	**OneMax และ OneMax Noisy**
จำนวนประชากร *(n)*	512 ตัว
ขนาดของปัญหา *(l)*	16 × 512 บิต
จำนวนเธรดในบล็อกในปัญหา Royal Road	32, 64, 128, 256, 512 เธรด
จำนวนครั้งในการวัดค่าความเหมาะสม	100,000 ครั้ง

ตารางที่ 5 พารามิเตอร์ที่ใช้ทดลองกับแก้ปัญหา Royal Road

พารามิเตอร์	**Royal Road**
จำนวนประชากร *(n)*	512 ตัว
ขนาดของปัญหา *(l)*	16 × 512 บิต
จำนวนเธรดในแต่ละบล็อก	32, 64, 128, 256, 512 เธรด
จำนวนเธรดในบล็อกในปัญหา Royal Road	512 บิต
จำนวนครั้งในการวัดค่าความเหมาะสม	300,000 ครั้ง

4. การวัดประสิทธิภาพด้านเวลาในการทำงาน

4.1 วัดจากเวลาในการทำงานทั้งหมด (Execution Time)

เวลาในการทำงานทั้งหมด เรียกว่า Execution time คือเวลาที่ใช้ในการวิวัฒนาการจนกว่าจะพบการแปรสภาพเป็น 0 หรือ 1 ทุกบิต การวัดประสิทธิภาพด้านเวลาของ pcGA จะวัดประสิทธิภาพตั้งแต่การส่งข้อมูลจากหน่วยประมวลผลกลางไปยังหน่วยประมวลผลกราฟิก จนกระทั่งมีการส่งข้อมูลกลับหลังจากทุกบิตของเวกเตอร์ความน่าจะเป็นเป็น 0 หรือ 1 หรือถึงจำนวนรุ่นที่จำกัดไว้ การวัดประสิทธิภาพด้านเวลาของ scGA จะวัดเวลาในการทำงานตั้งแต่เริ่มต้นจนพบว่าทุกบิตของเวกเตอร์ความน่าจะเป็นเป็น 0 หรือ 1 หรือถึงจำนวนรุ่นที่จำกัดไว้เช่นกัน เปรียบเทียบประสิทธิภาพของ Execution time ของ scGA และ pcGA ดังสมการ (6)

$$S_{exec} = \frac{T_{execution\_scGA}}{T_{execution\_pcGA}} \tag{6}$$

เมื่อ $T_{execution\_scGA}$ คือเวลาที่ใช้ในการวิวัฒนาการจนพบคำตอบบน CPU (ms)

$T_{execution\_pcGA}$ คือเวลาที่ใช้ในการวิวัฒนาการจนพบคำตอบในการทำงานบน CUDA (ms)

4.2 วัดจากเวลาในการทำงานต่อรอบ (Iteration Time)

ความแตกต่างของฟังก์ชันการสุ่มและการทำงานของของ floation point บนหน่วยประมวลผลกราฟิกและหน่วยประมวลผลกลางอาจส่งผลให้จำนวนรอบในการทำงานต่างกัน ทำให้เวลาที่ใช้ในการทำงานทั้งหมดแตกต่างกันด้วย เหตุผลดังกล่าวอาจก่อให้เกิดความลำเอียงเมื่อใช้การวิเคราะห์เวลาในการทำงานทั้งหมดเพียงอย่างเดียว จึงวัดประสิทธิภาพโดยใช้เวลาที่อัลกอริทึมสามารถดำเนินการในแต่ละรอบ เรียกว่า iteration time โดยการวัด iteration time ของ scGA จะวัดเวลาในการทำงานในตั้งแต่ขั้นตอนที่ 2 – 5 ในภาพที่ 6 และ iteration time ของ pcGA จะวัดเวลาในการทำงานตั้งแต่ขั้นตอนที่ 4 – 9 ในภาพที่ 11 การวัดประสิทธิภาพทำได้ดังสมการ (7)

$$S_{Iter} = \frac{T_{Iteration\_scGA}}{T_{Iteration\_pcGA}} \tag{7}$$

เมื่อ $T_{Iteration\_scGA}$ คือเวลาที่ใช้ในการทำงานต่อรอบบน CPU (ms)

$T_{Iteration\_pcGA}$ คือเวลาที่ใช้ในการทำงานต่อรอบบน CUDA (ms)

ผลและวิจารณ์

ผล

1. การทดลองเพื่อเปรียบเทียบประสิทธิภาพการทำงานแบบขนานและแบบอนุกรม

ผู้วิจัยได้ศึกษาความเร็วในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิก (pcGA) เปรียบเทียบกับขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบอนุกรมบนหน่วยประมวลผลกลาง (scGA) ผลการทดลองจำแนกตามการเปรียบเทียบได้ 3 กรณี คือ

1.1 การทดลองเพื่อเปรียบเทียบเวลาในการทำงานเมื่อใช้ฟังก์ชันการสุ่มที่แตกต่างกัน

การทดลองเปรียบเทียบเวลาในการทำงานเมื่อใช้ฟังก์ชันการสุ่มที่แตกต่างกัน โดย scGA ใช้ฟังก์ชันการสุ่ม rand() ของ Visual C++ และ pcGA ใช้ฟังก์ชันการสุ่มตามสมการ (5) จำแนกตามปัญหาทั้ง 3 แบบคือปัญหา OneMax, ปัญหา OneMax Noisy และปัญหา Royal Road ได้ผลดังนี้

1.1.1 ปัญหา OneMax

ผลการทดลองกับปัญหา OneMax เมื่อปัญหามีขนาดแตกต่างกัน โดยใช้แบบจำลองการเขียนโปรแกรม CUDA (Nugool and Worasait, 2010) พบว่า

ในปัญหาขนาด 1×512, 2×512 บิต เวลาในการทำงานทั้งหมดของ scGA ใช้เวลาน้อยกว่า pcGA แต่ในปัญหาขนาด 4×512 บิต ขึ้นไป pcGA จะใช้เวลาในการทำงานทั้งหมดน้อยกว่า โดยในปัญหาขนาด 512×512 บิต การใช้ pcGA จะช่วยให้สามารถทำงานได้เร็วกว่า scGA 21.24 เท่า และเมื่อพิจารณาจำนวนรอบในการทำงานพบว่า pcGA ที่ใช้ฟังก์ชันการสุ่มตามสมการ (5) จะใช้จำนวนรอบมากกว่า scGA ที่ใช้ฟังก์ชันการสุ่มของ Visual C++ เสมอ ดังตารางที่ 6

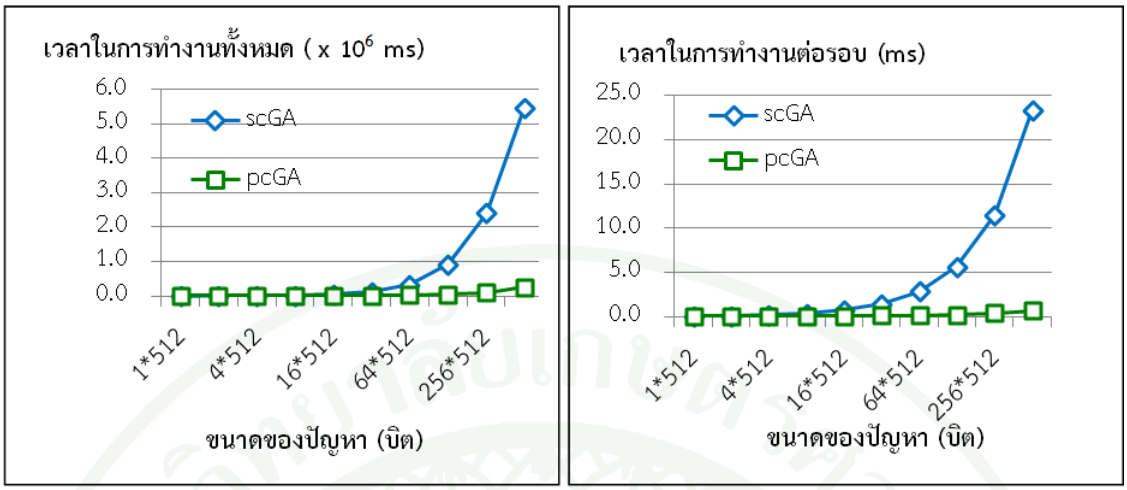
ผลการทดลองด้านเวลาในการทำงานต่อรอบของ scGA และ pcGA พบว่า scGA ใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเป็น 2 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า ในขณะที่ pcGA ใช้เวลาเพิ่มขึ้นเล็กน้อยเมื่อปัญหามีขนาดใหญ่ขึ้น ดังแสดงในตารางที่ 7 เมื่อเปรียบเทียบความเร็วที่เพิ่มขึ้นของเวลาในการทำงานต่อรอบพบว่าในปัญหาขนาดใหญ่ 512×512 บิต pcGA สามารถทำงานได้เร็วกว่า scGA 34.65 เท่า

ตารางที่ 6 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ scGA ที่ใช้ฟังก์ชันการสุ่ม rand() และpcGA ที่ใช้ฟังก์ชันการสุ่ม LCG ในปัญหา OneMax

ขนาดปัญหา (บิต)	เวลา (ms)		จำนวนครั้งในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	scGA	pcGA	scGA	pcGA	
1 × 512	582	1,693	35,037	47,586	0.3438
2 × 512	1,639	2,449	48,794	68,468	0.6691
4 × 512	4,580	3,538	68,537	96,144	1.2944
8 × 512	13,158	4,629	97,637	132,884	2.8423
16 × 512	38,465	8,044	143,281	185,944	4.7816
32 × 512	111,235	13,146	207,275	258,252	8.4616
64 × 512	327,398	24,534	305,657	352,052	13.3444
128 × 512	928,805	49,491	435,632	469,516	18.7671
256 × 512	2,405,555	105,333	536,967	600,264	22.8375
512 × 512	5,454,168	256,727	618,901.40	747,650.00	21.2450

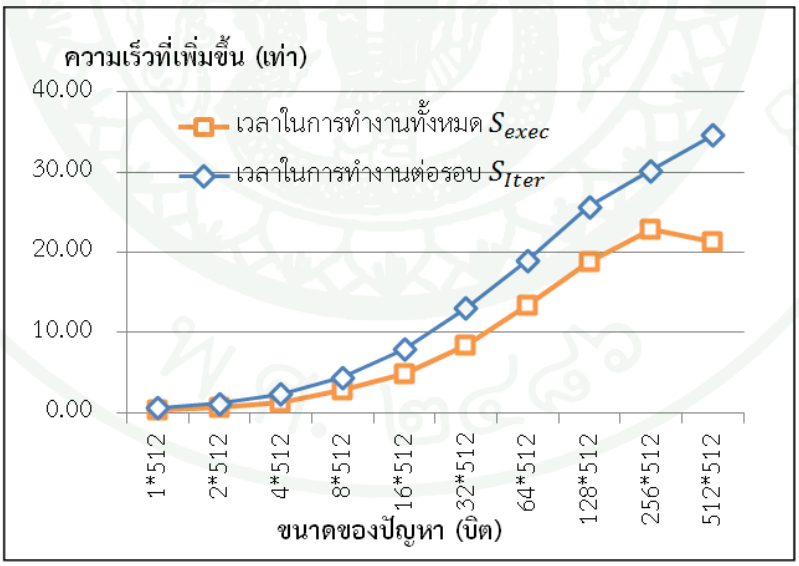
ตารางที่ 7 วลาในการทำงานต่อรอบ (ms) ของ scGA ที่ใช้ฟังก์ชันการสุ่ม rand() และ pcGA ที่ใช้ฟังก์ชันการสุ่ม LCG ในปัญหา OneMax

ขนาดปัญหา	เวลาในการทำงานต่อรอบ(ms)		ความเร็วที่เพิ่มขึ้น S_{Iter}
	scGA	pcGA	
1 × 512	0.0459	0.0771	0.5952
2 × 512	0.0894	0.0769	1.1628
4 × 512	0.1763	0.0772	2.2831
8 × 512	0.3510	0.0794	4.4192
16 × 512	0.7361	0.0931	7.9058
32 × 512	1.4085	0.1079	13.0501
64 × 512	2.8244	0.1488	18.9844
128 × 512	5.6263	0.2193	25.6608
256 × 512	11.4829	0.3805	30.1820
512 × 512	23.3138	0.6728	34.6513



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 14 การเปรียบเทียบประสิทธิภาพของเวลาในการทำงานของ scGA ที่ใช้ฟังก์ชัน rand() และ pcGA ที่ใช้ฟังก์ชัน LCG (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ ในปัญหา OneMax



ภาพที่ 15 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA ที่ใช้ฟังก์ชัน LCG เปรียบเทียบกับการทำงานของ scGA ที่ใช้ฟังก์ชัน rand() ในปัญหา OneMax

1.1.2 ปัญหา OneMax Noisy

ผลการทดลองกับปัญหา OneMax Noisy เมื่อปัญหามีขนาดแตกต่างกัน ได้ผลใกล้เคียงกับปัญหา OneMax นั่นคือ ในปัญหาขนาด 1 × 512, 2 × 512 และ 4 × 512 บิต เวลาในการทำงานทั้งหมดของ scGA น้อยกว่า pcGA แต่ในปัญหาขนาด 8 × 512 บิต ขึ้นไป pcGA จะใช้เวลาในการทำงานทั้งหมดน้อยกว่า โดยในปัญหาขนาด 512 × 512 บิต การใช้ pcGA จะช่วยให้สามารถทำงานได้เร็วกว่า scGA 21.03 เท่า และเมื่อพิจารณาจำนวนรอบในการทำงานพบว่า pcGA จะใช้จำนวนรอบในการทำงานมากกว่า scGA เสมอ โดยเฉพาะในปัญหาขนาด 4 × 512 pcGA ที่ใช้ฟังก์ชันการสุ่มในสมการที่ (5) ใช้จำนวนรอบในการทำงานถึง 1,246,612 รอบ ดังตารางที่ 8

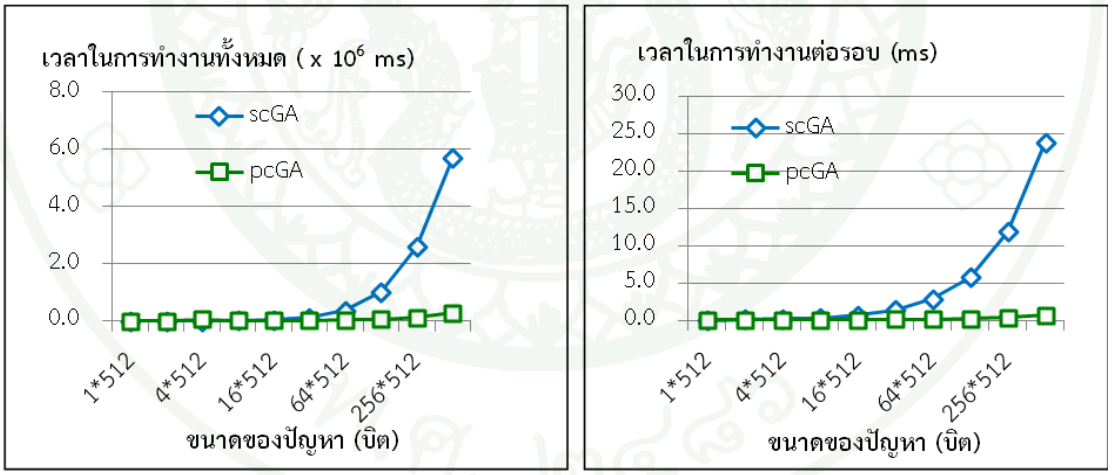
ผลการทดลองด้านเวลาในการทำงานต่อรอบของ scGA และ pcGA พบว่า scGA ใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเป็น 2 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า ในขณะที่ pcGA ใช้เวลาเพิ่มขึ้นเล็กน้อยเมื่อปัญหามีขนาดใหญ่ขึ้นดังแสดงในตารางที่ 9 เมื่อเปรียบเทียบความเร็วที่เพิ่มขึ้นของเวลาในการทำงานต่อรอบพบว่าในปัญหาขนาดใหญ่ 512 × 512 บิต pcGA สามารถทำงานได้เร็วกว่า scGA 33.15 เท่า

ตารางที่ 8 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ scGA ที่ใช้และ pcGA เมื่อขนาดของปัญหาแตกต่างกันในปัญหา OneMax Noisy

ขนาดปัญหา (บิต)	เวลา (ms)		จำนวนครั้งในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	scGA	pcGA	scGA	pcGA	
1 × 512	711	2,713	40,294	53,110	0.2622
2 × 512	2,004	4,097	55,875	76,636	0.4891
4 × 512	5,575	59,469	79,627	1,246,612	0.0937
8 × 512	19,394	7,811	143,061	151,468	2.4828
16 × 512	45,279	13,223	161,134	228,516	3.4242
32 × 512	127,883	21,062	229,568	321,390	6.0717
64 × 512	365,921	32,359	330,080	383,608	11.3081
128 × 512	1,006,338	58,739	447,112	494,156	17.1323
256 × 512	2,594,774	120,883	561,680	625,772	21.4652
512 × 512	5,721,156	272,086	637,843	857,060	21.0270

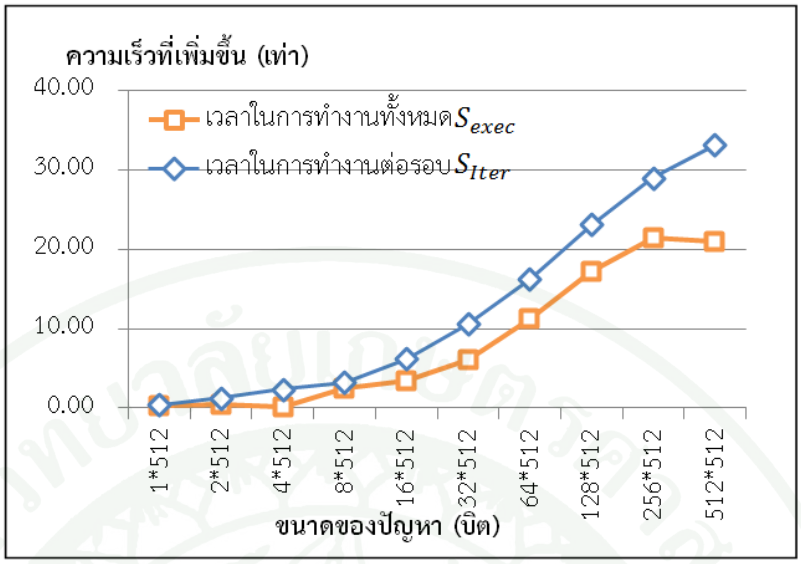
ตารางที่ 9 เวลาในการทำงานต่อรอบ (ms) ของ scGA ที่ใช้ฟังก์ชันการสุ่ม rand() และpcGA ที่ใช้ฟังก์ชันการสุ่ม LCG ในปัญหา OneMax Noisy

ขนาดปัญหา	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น
	scGA	pcGA	s_{Iter}
1 × 512	0.0475	0.1084	0.4375
2 × 512	0.1394	0.1078	1.2931
4 × 512	0.2482	0.1031	2.4086
8 × 512	0.3392	0.1036	3.2737
16 × 512	0.7452	0.1201	6.2067
32 × 512	1.4440	0.1359	10.6221
64 × 512	2.8928	0.1784	16.2190
128 × 512	5.7936	0.2506	23.1164
256 × 512	11.9010	0.4106	28.9823
512 × 512	23.9041	0.7210	33.1537



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 16 การเปรียบเทียบประสิทธิภาพของเวลาในการทำงานของ scGA ที่ใช้ฟังก์ชัน rand() และ pcGA ที่ใช้ฟังก์ชัน LCG (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ ในปัญหา OneMax Noisy



ภาพที่ 17 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA ที่ใช้ฟังก์ชัน LCG เปรียบเทียบกับการทำงานของ scGA ที่ใช้ฟังก์ชัน rand() ในปัญหา OneMax Noisy

1.1.3 ปัญหา Royal Road

ผลการทดลองกับปัญหา Royal Road พบว่าการทำงานของ pcGA ที่ใช้ฟังก์ชันการสุ่มตามสมการ (5) ใช้เวลาในการทำงานทั้งหมดและจำนวนรอบในการหาคำตอบน้อยกว่า scGA ที่ใช้ฟังก์ชันในการสุ่ม rand() ของ visual C++ ทุกขนาดของปัญหา และ scGA ไม่สามารถพบการแปรสภาพเป็น 0 หรือ 1 ครบทุกบิตได้ภายในจำนวนครั้งในการวัดค่าความเหมาะสมที่จำกัดไว้ จึงทำให้การทำงานของ pcGA เร็วขึ้นเมื่อเปรียบเทียบกับ scGA ถึง 70.34 เท่าในปัญหาขนาด 512×512 บิต ดังแสดงในตาราง 10

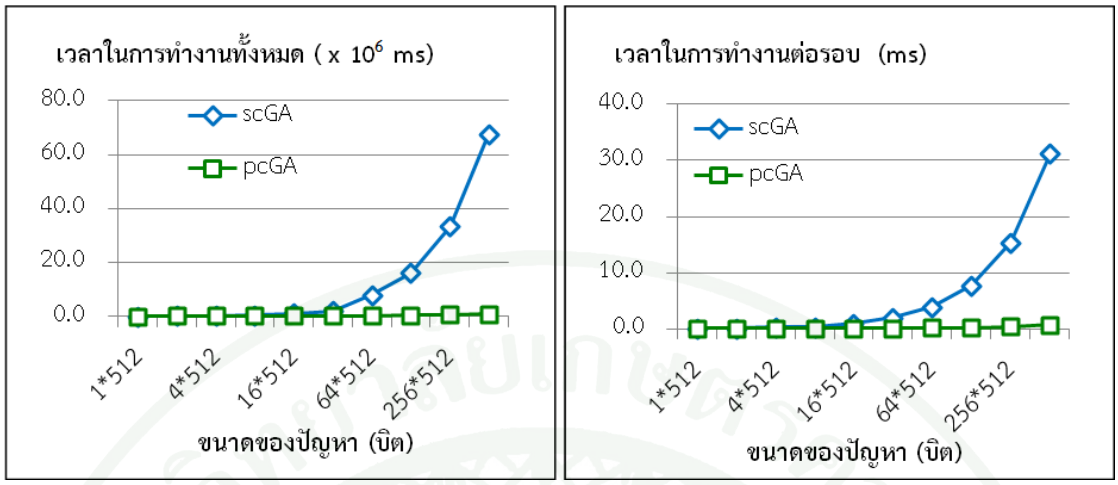
ผู้วิจัยพิจารณาเวลาในการทำงานต่อรอบของ scGA และ pcGA พบว่า scGA ใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเป็น 2 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า ในขณะที่ pcGA ใช้เวลาเพิ่มขึ้นเล็กน้อยเมื่อปัญหามีขนาดใหญ่ขึ้น ดังแสดงในตารางที่ 11 เมื่อเปรียบเทียบความเร็วที่เพิ่มขึ้นของเวลาในการทำงานต่อรอบพบว่าในปัญหาขนาดใหญ่ 512×512 บิต pcGA สามารถทำงานได้เร็วกว่า scGA 46.29 เท่า

ตารางที่ 10 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ scGA ที่ใช้ฟังก์ชันการสุ่ม rand() และ pcGA ที่ใช้ฟังก์ชันการสุ่ม LCG ในปัญหา Royal Road

ขนาดปัญหา (บิต)	เวลา (ms)		จำนวนครั้งในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	scGA	pcGA	scGA	pcGA	
1 × 512	57,963	56,749	3,000,000	1,631,070	1.0214
2 × 512	117,994	87,776	3,000,000	2,596,828	1.3443
4 × 512	239,894	89,994	3,000,000	2,596,828	2.6657
8 × 512	489,461	91,223	3,000,000	2,596,828	5.3656
16 × 512	1,002,085	133,622	3,000,000	3,234,570	7.4994
32 × 512	2,004,781	155,705	3,000,000	3,234,570	12.8755
64 × 512	8,033,001	213,718	6,000,000	3,241,148	37.5870
128 × 512	16,085,867	315,976	6,000,000	3,241,708	50.9086
256 × 512	33,315,498	536,427	6,000,000	3,280,830	62.1063
512 × 512	67,480,592	959,350	6,000,000	3,282,546	70.3399

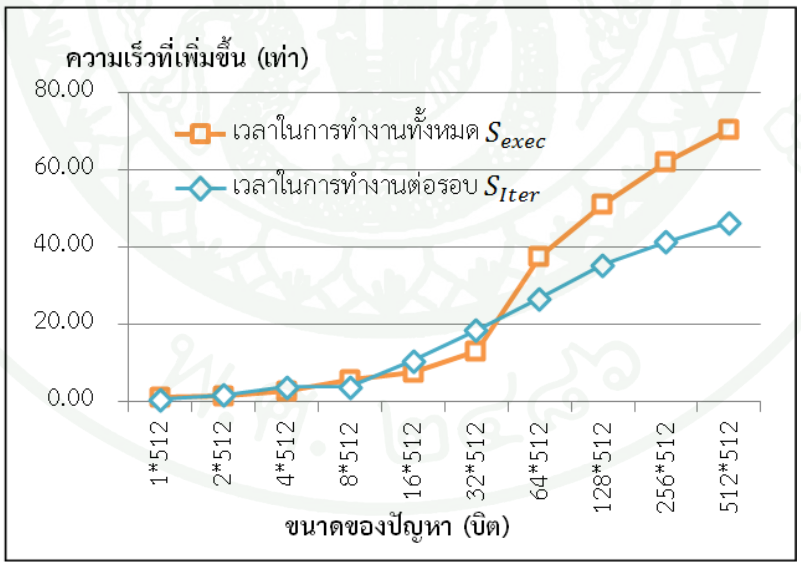
ตารางที่ 11 เวลาในการทำงานต่อรอบ (ms) ของ scGA ที่ใช้ฟังก์ชันการสุ่ม rand() และ pcGA ที่ใช้ฟังก์ชันการสุ่ม LCG ในปัญหา Royal Road

ขนาดปัญหา	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น S_{Iter}
	scGA	pcGA	
1 × 512	0.0366	0.0759	0.4822
2 × 512	0.1212	0.0756	1.6026
4 × 512	0.2775	0.0757	3.6637
8 × 512	0.2775	0.0767	3.6197
16 × 512	0.9675	0.0924	10.4723
32 × 512	1.9566	0.1068	18.3224
64 × 512	3.9253	0.1475	26.6170
128 × 512	7.7433	0.2190	35.3547
256 × 512	15.4100	0.3736	41.2431
512 × 512	31.1679	0.6733	46.2937



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 18 การเปรียบเทียบประสิทธิภาพของเวลาในการทำงานของ scGA ที่ใช้ฟังก์ชัน rand() และ pcGA ที่ใช้ฟังก์ชัน LCG (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ ในปัญหา Royal Road



ภาพที่ 19 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA ที่ใช้ฟังก์ชัน LCG เปรียบเทียบกับการทำงานของ scGA ที่ใช้ฟังก์ชัน rand() ในปัญหา Royal Road

1.2 เปรียบเทียบเวลาในการทำงานเมื่อใช้ฟังก์ชันการสุ่มเหมือนกัน

การทดลองเพื่อเปรียบเทียบเวลาในการทำงานของ pcGA และ scGA เมื่อใช้ฟังก์ชันการสุ่มตามสมการ (5) จำแนกตามปัญหาทั้ง 3 แบบคือปัญหา OneMax, ปัญหา OneMax Noisy และปัญหา Royal Road ได้ผลการทดลองดังนี้

1.2.1 ปัญหา OneMax

การทดลองกับปัญหา OneMax พบว่าเมื่อใช้ฟังก์ชันการสุ่มที่เหมือนกันจะให้จำนวนครั้งในการวัดค่าความเหมาะสมของปัญหาเท่ากัน ยกเว้นในปัญหาขนาด 64 × 512, 256 × 512 และ 512 × 512 บิต ด้านเวลาในการทำงานพบว่าเวลาในการทำงานของ pcGA จะน้อยกว่า scGA เมื่อปัญหามีขนาด 8 × 512 บิตขึ้นไป โดยการทำงานแบบ pcGA เร็วกว่า scGA 18.34 เท่าในปัญหาขนาด 512 × 512 บิต ดังแสดงในตารางที่ 12

ตารางที่ 12 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ scGA และ pcGA เมื่อใช้ฟังก์ชันการสุ่ม LCG ในปัญหา OneMax

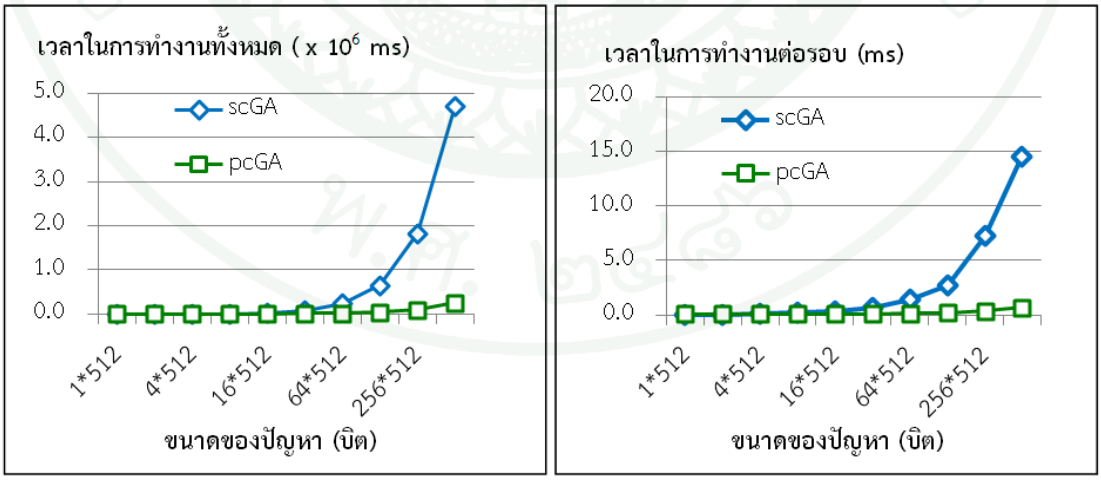
ขนาดปัญหา (บิต)	เวลา (ms)		จำนวนครั้งในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	scGA	pcGA	scGA	pcGA	
1 × 512	496.32	1,693.12	47,586.00	47,586.00	0.2931
2 × 512	1,393.83	2,449.75	68,468.00	68,468.00	0.5690
4 × 512	3,903.60	3,538.41	96,144.00	96,144.00	1.1032
8 × 512	10,940.89	4,629.43	132,884.00	132,884.00	2.3633
16 × 512	30,985.28	8,044.34	185,944.00	185,944.00	3.8518
32 × 512	87,145.11	13,146.00	258,252.00	258,252.00	6.6290
64 × 512	243,999.40	24,534.48	356,340.00	352,052.00	9.9452
128 × 512	662,187.59	49,491.17	469,516.00	469,516.00	13.3799
256 × 512	1,833,649.48	105,333.70	602,668.00	600,264.00	17.4080
512 × 512	4,710,771.95	256,727.60	746,952.00	747,650.00	18.3493

ผลการทดลองด้านเวลาในการทำงานต่อรอบของ scGA และ pcGA พบว่า scGA ใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเป็น 2 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า ในขณะที่ pcGA ใช้เวลาเพิ่มขึ้นเล็กน้อยเมื่อปัญหามีขนาดใหญ่ขึ้น ดังแสดงในตารางที่ 13 เมื่อเปรียบเทียบความเร็วที่

เพิ่มขึ้นของเวลาในการทำงานต่อรอบพบว่าในปัญหาขนาดใหญ่ 512 × 512 บิต pcGA สามารถทำงานได้เร็วกว่า scGA 21.58 เท่า

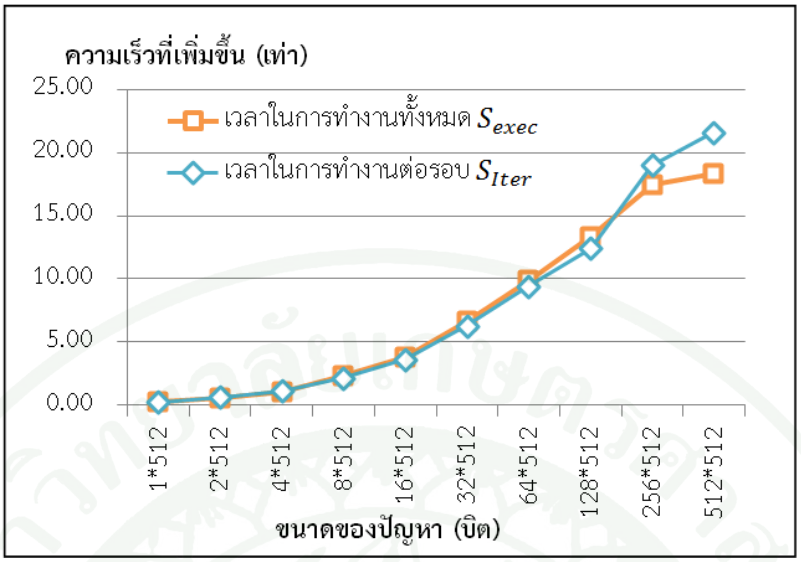
ตารางที่ 13 เวลาในการทำงานต่อรอบ (ms) ของ scGA และ pcGA เมื่อใช้ฟังก์ชันการสุ่ม LCG ในปัญหา OneMax

ขนาดปัญหา	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น
	scGA	pcGA	S_{Iter}
1 × 512	0.0187	0.0771	0.2432
2 × 512	0.0504	0.0769	0.6559
4 × 512	0.0867	0.0772	1.1232
8 × 512	0.1718	0.0794	2.1632
16 × 512	0.3413	0.0931	3.6660
32 × 512	0.6785	0.1079	6.2861
64 × 512	1.4051	0.1488	9.4448
128 × 512	2.7231	0.2193	12.4195
256 × 512	7.2370	0.3805	19.0219
512 × 512	14.5204	0.6728	21.5816



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 20 เวลาในการทำงานของ scGA และ pcGA เมื่อใช้ฟังก์ชันการสุ่ม LCG (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ ในปัญหา OneMax



ภาพที่ 21 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA เปรียบเทียบกับการทำงานแบบ scGA เมื่อใช้ฟังก์ชันการสุ่ม LCG ในปัญหา OneMax

1.2.2 ปัญหา OneMax Noisy

ผลการทดลองกับปัญหา OneMax Noisy เมื่อปัญหามีขนาดแตกต่างกัน ได้ผลใกล้เคียงกับปัญหา OneMax นั่นคือ ในปัญหาขนาด 1 × 512, 2 × 512 และ 4 × 512 บิต scGA ใช้เวลาน้อยกว่า pcGA แต่ในปัญหาขนาด 8 × 512 บิต ขึ้นไป pcGA จะใช้เวลาในการทำงานทั้งหมดน้อยกว่า โดยในปัญหาขนาด 512 × 512 บิต pcGA สามารถทำงานได้เร็วกว่า scGA 24.26 เท่า จำนวนรอบในการทำงานเมื่อใช้ฟังก์ชันการสุ่มที่เหมือนกัน พบว่า pcGA จะใช้จำนวนรอบในการทำงานไม่เท่ากับ scGA ยกเว้นในปัญหาขนาด 1 × 512, 4 × 512 และ 64 × 512 ดังตารางที่ 14

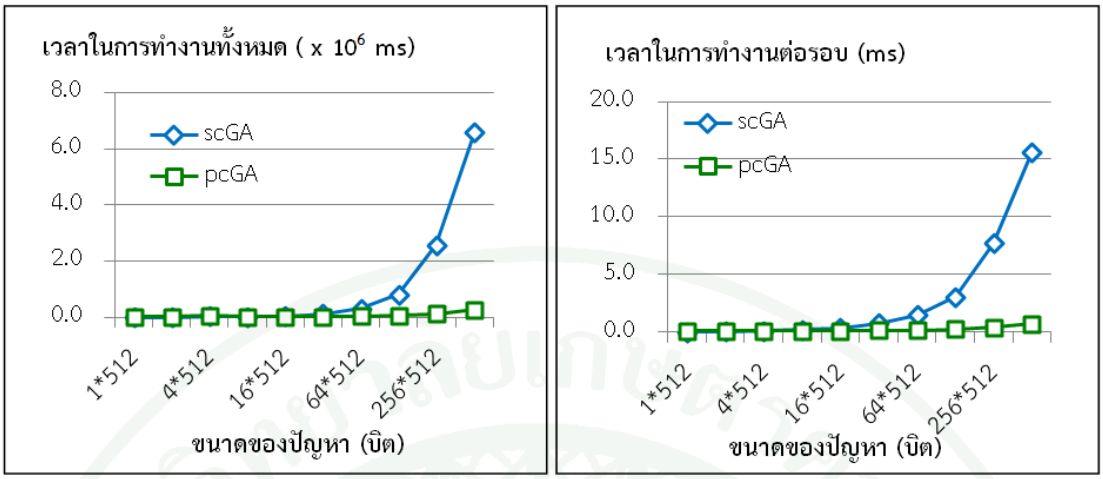
ผลการทดลองด้านเวลาในการทำงานต่อรอบของ scGA และ pcGA พบว่า scGA ใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเป็น 2 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า ในขณะที่ pcGA ใช้เวลาเพิ่มขึ้นเล็กน้อยเมื่อปัญหามีขนาดใหญ่ขึ้น ดังแสดงในตารางที่ 15 เมื่อเปรียบเทียบความเร็วที่เพิ่มขึ้นของเวลาในการทำงานต่อรอบพบว่าในปัญหาขนาดใหญ่ 512 × 512 บิต pcGA สามารถทำงานได้เร็วกว่า scGA 21.71 เท่า

ตารางที่ 14 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ scGA และ pcGA เมื่อใช้ฟังก์ชันการสุ่ม LCG ในปัญหา OneMax Noisy

ขนาดปัญหา (บิต)	เวลา (ms)		จำนวนรอบในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	scGA	pcGA	scGA	pcGA	
1 × 512	605	2,713	53,110	53,110	0.2229
2 × 512	1,741	4,097	76,794	76,636	0.4249
4 × 512	44,384	59,469	1,246,612	1,246,612	0.7463
8 × 512	16,418	7,811	182,294	151,468	2.1018
16 × 512	40,112	13,223	211,486	228,516	3.0335
32 × 512	110,726	21,062	286,832	321,390	5.2571
64 × 512	306,489	32,359	383,608	383,608	9.4715
128 × 512	811,988	58,739	494,634	494,156	13.8236
256 × 512	2,574,526	120,883	627,396	625,772	21.2977
512 × 512	6,600,377	272,086	772,704	857,060	24.2584

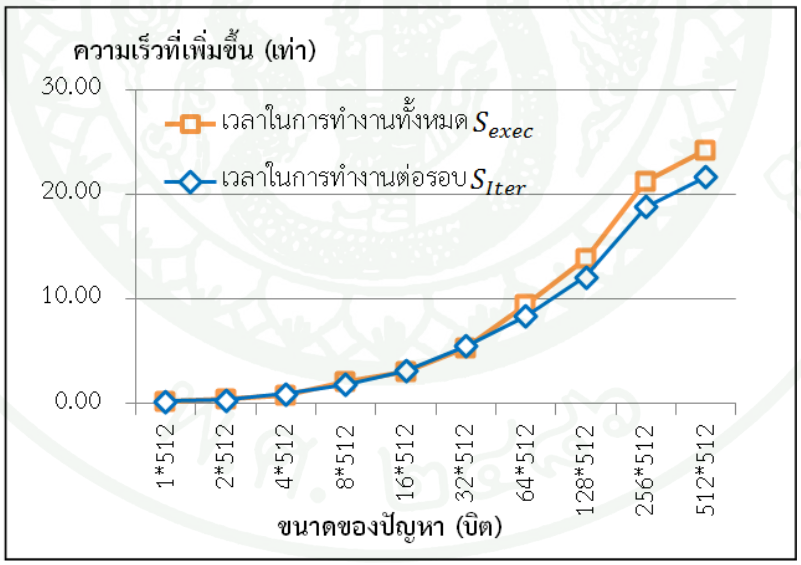
ตารางที่ 15 เวลาในการทำงานต่อรอบ (ms) ของ scGA และ pcGA เมื่อใช้ฟังก์ชันการสุ่ม LCG ในปัญหา OneMax Noisy

ขนาดปัญหา	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น S_{Iter}
	scGA	pcGA	
1 × 512	0.0261	0.1084	0.2402
2 × 512	0.0380	0.1078	0.3528
4 × 512	0.0953	0.1031	0.9245
8 × 512	0.1892	0.1036	1.8260
16 × 512	0.3754	0.1201	3.1266
32 × 512	0.7481	0.1359	5.5031
64 × 512	1.4928	0.1784	8.3697
128 × 512	3.0439	0.2506	12.1450
256 × 512	7.7422	0.4106	18.8544
512 × 512	15.6562	0.7210	21.7143



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 22 เวลาในการทำงานของ scGA และ pscGA เมื่อใช้ฟังก์ชันการสุ่ม LCG
(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบในปัญหา OneMax Noisy



ภาพที่ 23 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA เปรียบเทียบกับการทำงานแบบ scGA
เมื่อใช้ฟังก์ชันการสุ่ม LCG ในปัญหา OneMax Noisy

1.2.3 ปัญหา Royal Road

ผู้วิจัยได้ทดลองกับปัญหา Royal Road เมื่อปัญหามีขนาดแตกต่างกัน พบว่าในปัญหาขนาด 1 × 512, 2 × 512 และ 4 × 512 บิต scGA ใช้เวลาน้อยกว่า pcGA แต่ในปัญหาขนาด 8 × 512 บิต ขึ้นไป pcGA จะใช้เวลาในการทำงานทั้งหมดน้อยกว่า โดยในปัญหาขนาด 512 × 512 บิต pcGA สามารถทำงานได้เร็วกว่า scGA 23.96 เท่า และเมื่อพิจารณาจำนวนรอบในการทำงาน พบว่า pcGA และ scGA ใช้จำนวนครั้งในการวัดค่าความเหมาะสมเท่ากันเสมอ โดยปัญหาที่มีขนาดใกล้เคียงกันจะใช้จำนวนรอบในการทำงานเท่ากันแต่ใช้เวลาในการทำงานต่างกัน ดังตารางที่ 16

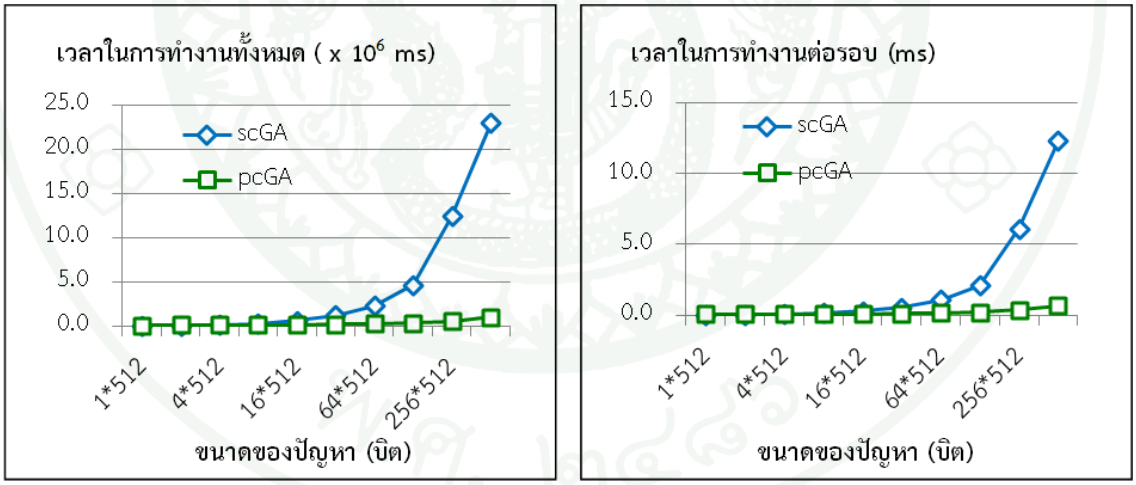
ตารางที่ 16 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ scGA และ pcGA เมื่อใช้ฟังก์ชันการสุ่ม LCG ในปัญหา Royal Road

ขนาดปัญหา (บิต)	เวลา (ms)		จำนวนครั้งในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	scGA	pcGA	scGA	pcGA	
1 × 512	18,963	56,749	1,631,070	1,631,070	0.3342
2 × 512	62,415	87,776	2,596,828	2,596,828	0.7111
4 × 512	118,701	89,994	2,596,828	2,596,828	1.3190
8 × 512	234,020	91,223	2,596,828	2,596,828	2.5654
16 × 512	593,711	133,622	3,234,570	3,234,570	4.4432
32 × 512	1,167,247	155,705	3,234,570	3,234,570	7.4965
64 × 512	2,305,027	213,718	3,241,148	3,241,148	10.7854
128 × 512	4,631,332	315,976	3,241,708	3,241,708	14.6572
256 × 512	12,464,081	536,427	3,280,830	3,280,830	23.2354
512 × 512	22,982,715	959,350	3,282,546	3,282,546	23.9566

ผลการทดลองด้านเวลาในการทำงานต่อรอบของ scGA และ pcGA พบว่า scGA ใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเป็น 2 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า ในขณะที่ pcGA ใช้เวลาเพิ่มขึ้นเล็กน้อยเมื่อปัญหามีขนาดใหญ่ขึ้น ดังแสดงในตารางที่ 17 เมื่อเปรียบเทียบความเร็วที่เพิ่มขึ้นของเวลาในการทำงานต่อรอบพบว่าในปัญหาขนาดใหญ่ 512 x 512 บิต pcGA สามารถทำงานได้เร็วกว่า scGA 18.27 เท่า

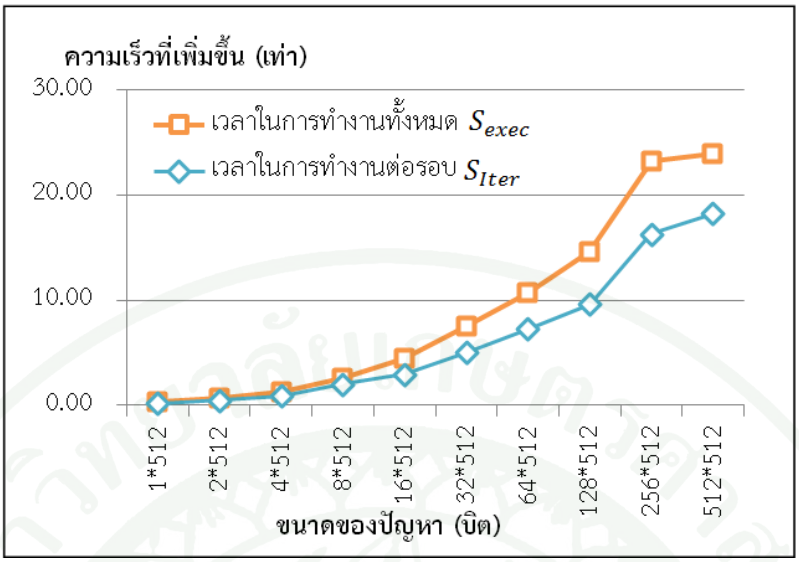
ตารางที่ 17 เวลาในการทำงานต่อรอบ (ms) ของ scGA และ pcGA เมื่อใช้ฟังก์ชันการสุ่ม LCG ในปัญหา Royal Road

ขนาดปัญหา	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น
	scGA	pcGA	S_{Iter}
1 × 512	0.0137	0.0759	0.1803
2 × 512	0.0363	0.0756	0.4800
4 × 512	0.0695	0.0757	0.9183
8 × 512	0.1533	0.0767	2.0004
16 × 512	0.2741	0.0924	2.9667
32 × 512	0.5391	0.1068	5.0480
64 × 512	1.0747	0.1475	7.2874
128 × 512	2.1050	0.2190	9.6109
256 × 512	6.1023	0.3736	16.3322
512 × 512	12.3018	0.6733	18.2718



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 24 เวลาในการทำงานของ scGA และ pscGA เมื่อใช้ฟังก์ชันการสุ่ม LCG (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ ในปัญหา Royal Road



ภาพที่ 25 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA เปรียบเทียบกับการทำงานแบบ scGA เมื่อใช้ฟังก์ชันการสุ่ม LCG ในปัญหา Royal Road

1.3 เปรียบเทียบเวลาในการทำงานเมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND()

การทดลองเปรียบเทียบเวลาในการทำงานเมื่อใช้ scGA ใช้ฟังก์ชันการสุ่ม rand() และ pcGA ใช้ฟังก์ชันการสุ่ม CURAND() จำแนกตามปัญหาทั้ง 3 แบบคือ ปัญหา OneMax, ปัญหา OneMax Noisy และ ปัญหา Royal Road ได้ผลดังนี้

1.3.1 ปัญหา OneMax

ผลการทดลองกับปัญหา OneMax เมื่อปัญหามีขนาดแตกต่างกัน พบว่าจำนวนรอบในการทำงานของ scGA และ pcGA ใกล้เคียงกันในทุกขนาดปัญหาทดสอบ และในปัญหาขนาด 1×512, 2×512, 4×512 และ 8×512 บิต เวลาในการทำงานทั้งหมดของ scGA ใช้เวลาน้อยกว่า pcGA แต่ในปัญหาขนาด 16×512 บิต ขึ้นไป pcGA จะใช้เวลาในการทำงานทั้งหมดน้อยกว่า scGA เล็กน้อย โดยในปัญหาขนาด 512×512 บิต การใช้ pcGA จะช่วยให้สามารถทำงานได้เร็วกว่า scGA เพียง 1.25 เท่า ดังตารางที่ 18

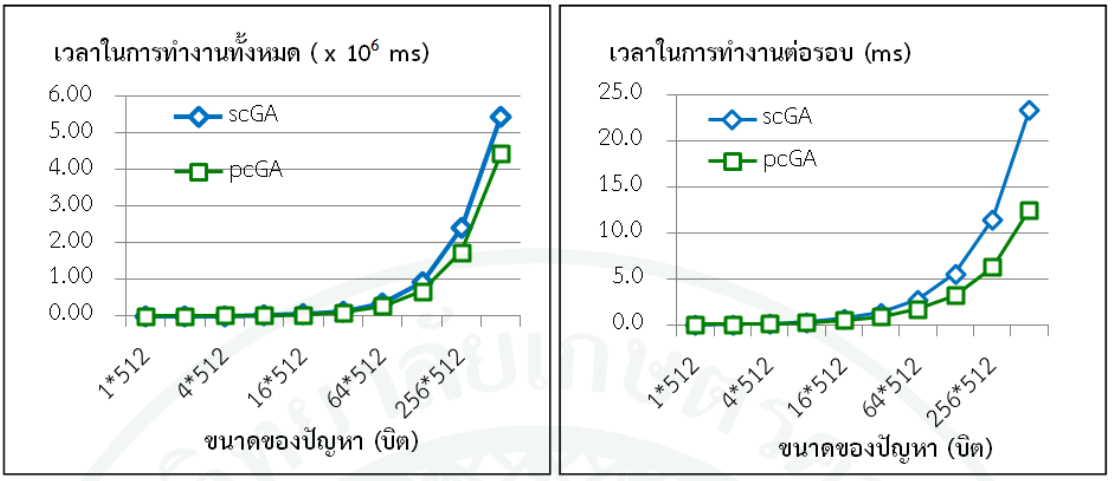
ผลการทดลองด้านเวลาในการทำงานต่อรอบของ scGA และ pcGA พบว่า scGA ใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเป็น 2 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า ในขณะที่ pcGA ที่ทำงานโดยใช้ฟังก์ชัน CURAND() ใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเป็น 1.8 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า ดังแสดงในตารางที่ 19 เมื่อเปรียบเทียบความเร็วที่เพิ่มขึ้นของเวลาในการทำงานต่อรอบพบว่าในปัญหาขนาดใหญ่ 512×512 บิต pcGA สามารถทำงานได้เร็วกว่า scGA 1.86 เท่า

ตารางที่ 18 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ scGA และ pcGA เมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND() ในปัญหา OneMax

ขนาดปัญหา (บิต)	เวลา (ms)		จำนวนรอบในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	scGA	pcGA	scGA	pcGA	
1 × 512	582	2,157	35,037	34,909	0.2698
2 × 512	1,639	3,031	48,794	48,916	0.5408
4 × 512	4,580	5,855	68,537	68,600	0.7823
8 × 512	13,158	13,492	97,637	97,191	0.9752
16 × 512	38,465	36,583	143,281	139,163	1.0515
32 × 512	111,236	94,681	207,275	201,661	1.1748
64 × 512	327,399	268,055	305,657	290,504	1.2214
128 × 512	928,805	666,107	435,632	407,729	1.3944
256 × 512	2,405,555	1,745,844	536,967	549,631	1.3779
512 × 512	5,454,169	4,454,348	618,901	711,323	1.2245

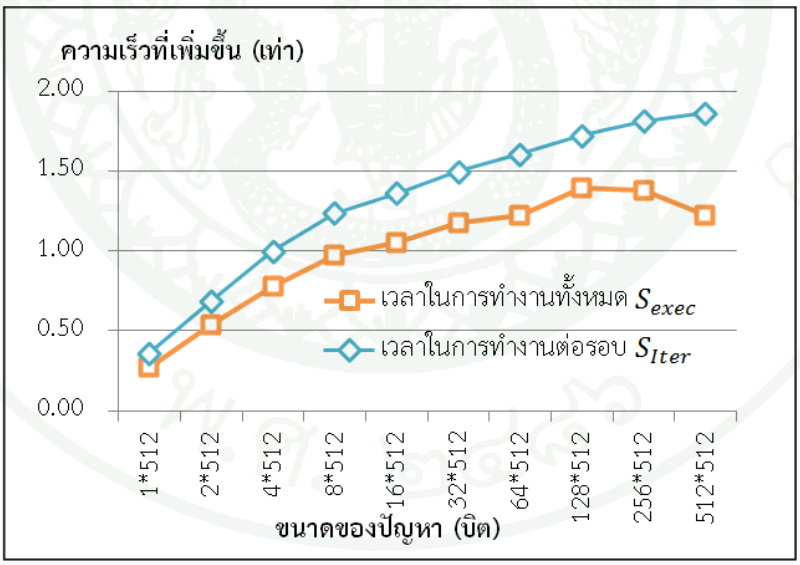
ตารางที่ 19 เวลาในการทำงานต่อรอบ (ms) ของ scGA และ pcGA เมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND() ในปัญหา OneMax

ขนาดปัญหา	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น S_{Iter}
	scGA	pcGA	
1 × 512	0.0459	0.1269	0.3617
2 × 512	0.0894	0.1296	0.6896
4 × 512	0.1763	0.1763	0.9999
8 × 512	0.3510	0.2831	1.2397
16 × 512	0.7361	0.5403	1.3623
32 × 512	1.4085	0.9387	1.5004
64 × 512	2.8244	1.7570	1.6075
128 × 512	5.6263	3.2591	1.7263
256 × 512	11.4829	6.3249	1.8155
512 × 512	23.3138	12.4965	1.8656



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 26 การเปรียบเทียบประสิทธิภาพของเวลาในการทำงานของ scGA และ pcGA (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ เมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND() ในปัญหา OneMax



ภาพที่ 27 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA เปรียบเทียบกับการทำงานแบบ scGA เมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND() ในปัญหา OneMax

1.3.2 ปัญหา OneMax Noisy

ผลการทดลองกับปัญหา OneMax Noisy เมื่อปัญหามีขนาดแตกต่างกัน พบว่า จำนวนรอบในการทำงานของ scGA และ pcGA ใกล้เคียงกันในทุกขนาดปัญหาทดสอบ และในปัญหาขนาด 1 × 512, 2 × 512 และ 4 × 512 บิต เวลาในการทำงานทั้งหมดของ scGA ใช้เวลาน้อยกว่า pcGA ในปัญหาขนาด 8 × 512 บิต scGA และ pcGA ใช้เวลาในการทำงานใกล้เคียงกัน และในปัญหาขนาด 16 × 512 บิต ขึ้นไป pcGA จะใช้เวลาในการทำงานทั้งหมดน้อยกว่า scGA เล็กน้อย โดยในปัญหาขนาด 512 × 512 บิต การใช้ pcGA จะช่วยให้สามารถทำงานได้เร็วกว่า scGA เพียง 1.25 เท่า ดังตารางที่ 20

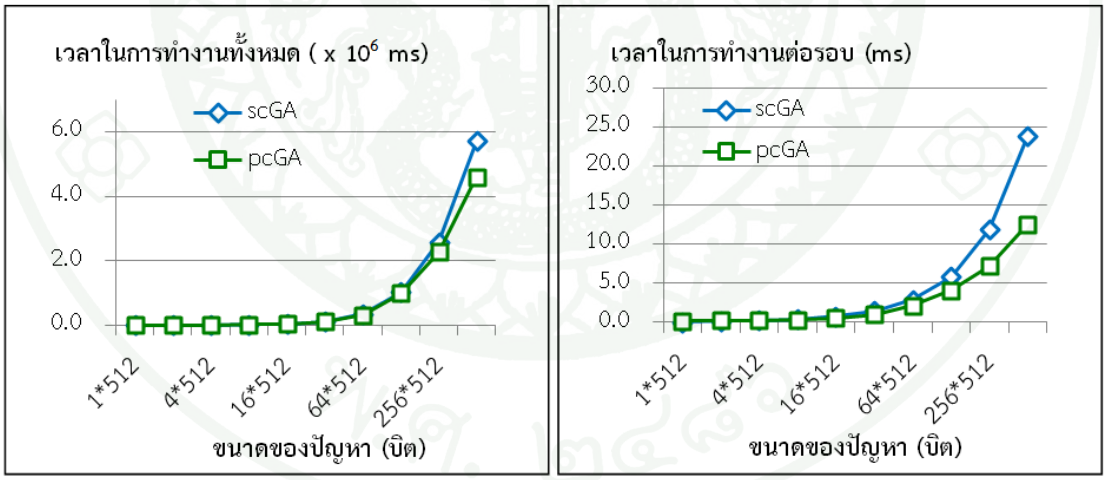
ผลการทดลองด้านเวลาในการทำงานต่อรอบของ scGA และ pcGA พบว่า scGA ใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเป็น 2 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า ในขณะที่ pcGA ที่ทำงานโดยใช้ฟังก์ชัน CURAND() ใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเป็น 1.8 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า ดังแสดงในตารางที่ 21 เมื่อเปรียบเทียบความเร็วที่เพิ่มขึ้นของเวลาในการทำงานต่อรอบพบว่าในปัญหาขนาดใหญ่ 512 × 512 บิต pcGA สามารถทำงานได้เร็วกว่า scGA 1.91 เท่า

ตารางที่ 20 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมทำงานของ scGA และ pcGA เมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND() ในปัญหา OneMax Noisy

ขนาดปัญหา (บิต)	เวลา (ms)		จำนวนรอบในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	scGA	pcGA	scGA	pcGA	
1 × 512	711	3,141	40,294	39,762	0.2265
2 × 512	2,004	4,615	55,875	56,625	0.4342
4 × 512	5,575	7,740	79,627	78,361	0.7203
8 × 512	19,394	18,906	143,061	124,557	1.0258
16 × 512	45,279	43,508	161,134	161,842	1.0407
32 × 512	127,883	111,306	229,568	238,069	1.1489
64 × 512	365,921	313,838	330,080	363,962	1.1660
128 × 512	1,039,420	1,007,163	444,596	621,041	1.0320
256 × 512	2,594,774	2,273,026	561,680	714,262	1.1416
512 × 512	5,721,156	4,596,941	637,843	732,226	1.2446

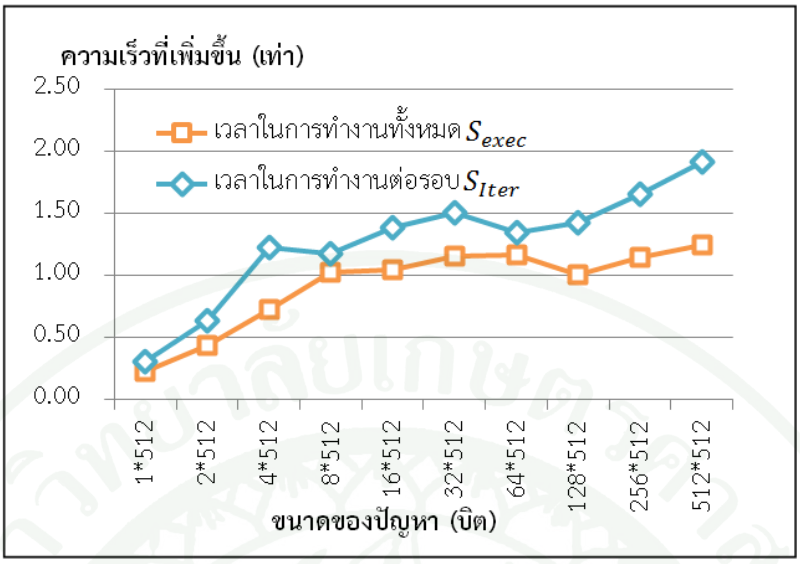
ตารางที่ 21 เวลาในการทำงานต่อรอบ (ms) ของ scGA และแบบ pcGA เมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND() ในปัญหา OneMax Noisy

ขนาดปัญหา	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น
	scGA	pcGA	S_{Iter}
1 × 512	0.0475	0.1579	0.3005
2 × 512	0.1394	0.2196	0.6349
4 × 512	0.2482	0.2027	1.2247
8 × 512	0.3621	0.3090	1.1717
16 × 512	0.7452	0.5404	1.3790
32 × 512	1.4440	0.9611	1.5024
64 × 512	2.8928	2.1618	1.3381
128 × 512	5.7936	4.0850	1.4183
256 × 512	11.9010	7.2141	1.6497
512 × 512	23.9041	12.5185	1.9095



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 28 เวลาในการทำงานของ scGA และ pcGA (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ เมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND() ในปัญหา OneMax Noisy



ภาพที่ 29 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA เปรียบเทียบกับการทำงานแบบ scGA เมื่อใช้การสุ่ม rand() และ CURAND() ในปัญหา OneMax Noisy

1.3.3 ปัญหา Royal Road

ผลการทดลองกับปัญหา Roal Road เมื่อปัญหามีขนาดแตกต่างกัน พบว่า ในปัญหาขนาด 1×512 และ 2×512 บิต เวลาในการทำงานทั้งหมดของ scGA ใช้เวลาน้อยกว่า pcGA แต่ในปัญหาขนาด 4×512 บิต ขึ้นไป pcGA จะใช้เวลาในการทำงานทั้งหมดน้อยกว่าเพียงเล็กน้อย โดยในปัญหาขนาด 512×512 บิต การใช้ pcGA จะช่วยให้สามารถทำงานได้เร็วกว่า scGA 1.80 เท่า และเมื่อพิจารณาจำนวนรอบในการทำงานพบว่า scGA ไม่สามารถพบคำตอบเมื่อถึงจำนวนครั้งในการวัดค่าความเหมาะสมที่จำกัดไว้ได้ ในขณะที่ pcGA สามารถพบคำตอบได้ ยกเว้นในปัญหาขนาด 256×512 และ 512×512 บิต ทั้ง scGA และ pcGA ไม่สามารถค้นหาคำตอบได้ ดังตารางที่ 22

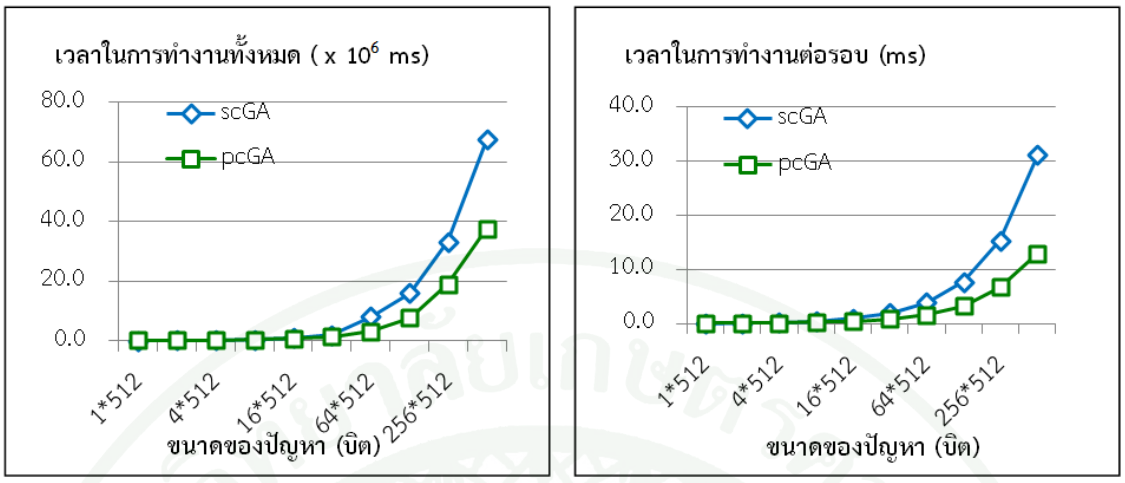
ผลการทดลองด้านเวลาในการทำงานต่อรอบของ scGA และ pcGA พบว่า scGA ใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเป็น 2 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า ในขณะที่ pcGA ใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเป็น 1.7 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า ดังแสดงในตารางที่ 23 เมื่อเปรียบเทียบความเร็วที่เพิ่มขึ้นของเวลาในการทำงานต่อรอบพบว่าในปัญหาขนาดใหญ่ 512×512 บิต pcGA สามารถทำงานได้เร็วกว่า scGA 2.40 เท่า

ตารางที่ 22 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ scGA และ pcGA เมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND() ในปัญหา Royal Road

ขนาดปัญหา (บิต)	เวลา (ms)		จำนวนครั้งในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	scGA	pcGA	scGA	pcGA	
1 × 512	57,963	113,712	3,000,000	1,877,304	0.5097
2 × 512	117,994	128,481	3,000,000	2,123,791	0.9184
4 × 512	239,894	184,234	3,000,000	2,191,126	1.3021
8 × 512	489,461	308,300	3,000,000	2,241,953	1.5876
16 × 512	1,002,085	725,548	3,000,000	2,770,794	1.3811
32 × 512	2,004,781	1,299,491	3,000,000	2,770,794	1.5427
64 × 512	8,033,001	3,131,755	6,000,000	3,669,207	2.5650
128 × 512	16,085,867	7,807,033	6,000,000	4,752,790	2.0604
256 × 512	33,315,498	19,010,206	6,000,000	6,000,000	1.7525
512 × 512	67,480,592	37,552,568	6,000,000	6,000,000	1.7970

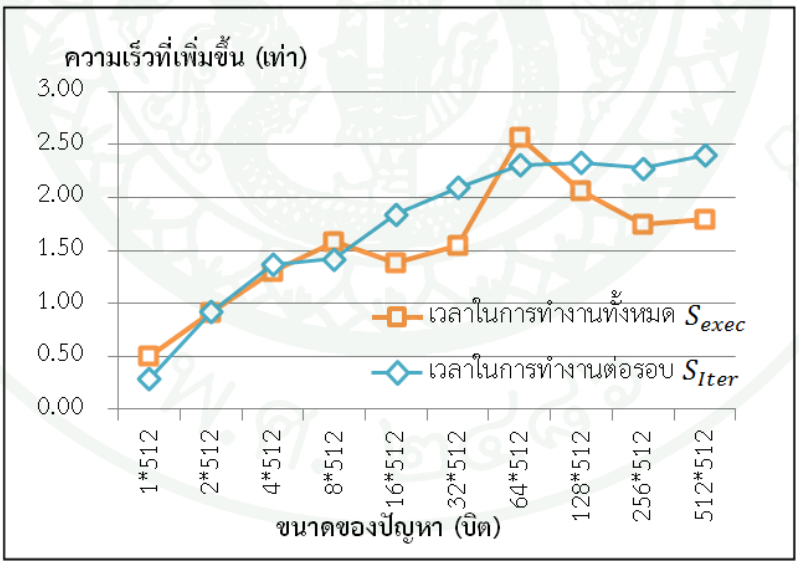
ตารางที่ 23 เวลาในการทำงานต่อรอบ (ms) ของscGA และแบบขนาน pcGA เมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND() ในปัญหา Royal Road

ขนาดปัญหา	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น S_{Iter}
	scGA	pcGA	
1 × 512	0.0366	0.1265	0.2894
2 × 512	0.1212	0.1312	0.9241
4 × 512	0.2407	0.1755	1.3715
8 × 512	0.4837	0.3395	1.4249
16 × 512	0.9675	0.5244	1.8450
32 × 512	1.9566	0.9328	2.0976
64 × 512	3.9253	1.6995	2.3097
128 × 512	7.7433	3.3249	2.3289
256 × 512	15.4100	6.7547	2.2814
512 × 512	31.1679	12.9669	2.4036



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 30 เวลาในการทำงานของ scGA และ pcGA (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ เมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND() ในปัญหา Royal Road



ภาพที่ 31 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA เปรียบเทียบกับ scGA เมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND() ในปัญหา Royal Road

2. การทดลองเพื่อเปรียบเทียบประสิทธิภาพเมื่อมีการวัดค่าความเหมาะสมที่แตกต่างกัน

การทดลองเพื่อเปรียบเทียบประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน เมื่อมีการวัดค่าความเหมาะสมที่แตกต่าง โดยใช้ฟังก์ชันการสุ่มดังสมการ (5) ทดสอบกับปัญหาทั้ง 3 แบบคือ ปัญหา OneMax, ปัญหา OneMax Noisy และปัญหา Royal Road ได้ผลการทดลองดังนี้

2.1 ปัญหา OneMax

ผลการทดลองกับปัญหา OneMax เมื่อปัญหามีขนาดแตกต่างกัน พบว่าเมื่อใช้ฟังก์ชันการสุ่มที่เหมือนกัน การวัดค่าความเหมาะสมแบบเธรดเดียว และการวัดค่าความเหมาะสมแบบการลดทอนใช้จำนวนครั้งในการทำงานเท่ากัน โดยการวัดค่าความเหมาะสมแบบลดทอนใช้เวลาในการทำงานน้อยกว่าการวัดค่าความเหมาะสมแบบเธรดเดียวในทุกขนาดปัญหา และในปัญหาขนาด 512 × 512 บิต การวัดค่าความเหมาะสมโดยการลดทอนจะช่วยให้สามารถทำงานได้เร็วกว่าการวัดค่าความเหมาะสมแบบเธรดเดียว 7.57 เท่า ดังตารางที่ 24

ตารางที่ 24 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ pcGA เมื่อใช้วิธีการวัดค่าความเหมาะสมแตกต่างกัน ในปัญหา OneMax

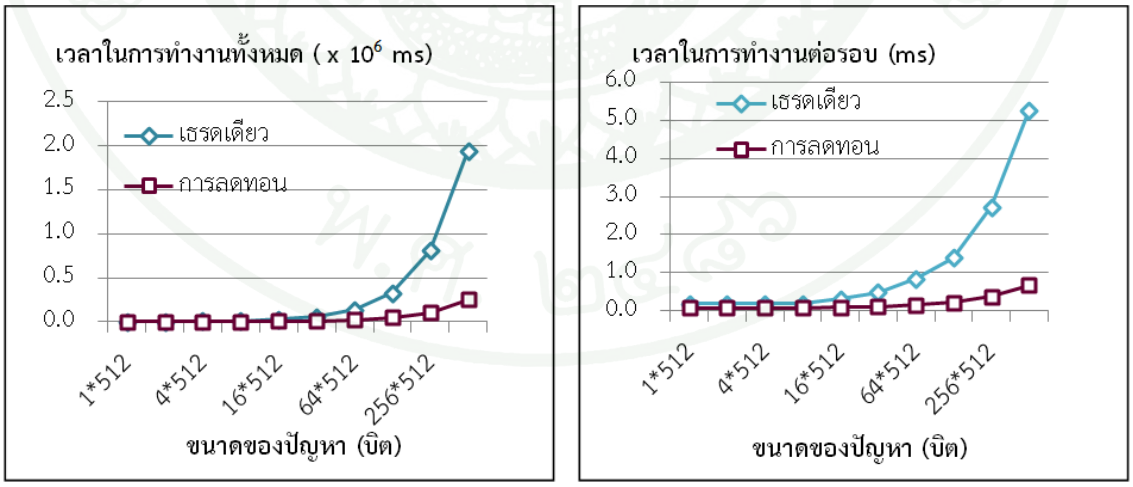
ขนาดปัญหา (บิต)	เวลา (ms)		จำนวนครั้งในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	เธรดเดียว	การลดทอน	เธรดเดียว	การลดทอน	
1 × 512	4,254	1,693	47,586	47,586	2.5126
2 × 512	6,133	2,450	68,468	68,468	2.5036
4 × 512	8,842	3,538	96,144	96,144	2.4987
8 × 512	11,785	4,629	132,884	132,884	2.5457
16 × 512	28,172	8,044	185,944	185,944	3.5021
32 × 512	54,085	13,146	258,252	258,252	4.1142
64 × 512	137,232	24,534	352,052	352,052	5.5934
128 × 512	326,144	49,491	469,516	469,516	6.5899
256 × 512	810,554	105,334	600,264	600,264	7.6951
512 × 512	1,944,842	256,728	747,650	747,650	7.5755

ผลการทดลองด้านเวลาในการทำงานต่อรอบของ pcGA พบว่า pcGA ที่วัดค่าความเหมาะสมโดยการลดทอนใช้เวลาในการทำงานน้อยกว่า pcGA ที่วัดค่าความเหมาะสมโดยใช้เธรดเดียว

ดังแสดงในตารางที่ 25 และเมื่อเปรียบเทียบความเร็วที่เพิ่มขึ้นของเวลาในการทำงานต่อรอบพบว่าในปัญหาขนาดใหญ่ 512 × 512 บิต pcGA แบบการลดทอนสามารถทำงานได้เร็วกว่า pcGA แบบการเธรดเดียวถึง 7.82 เท่า

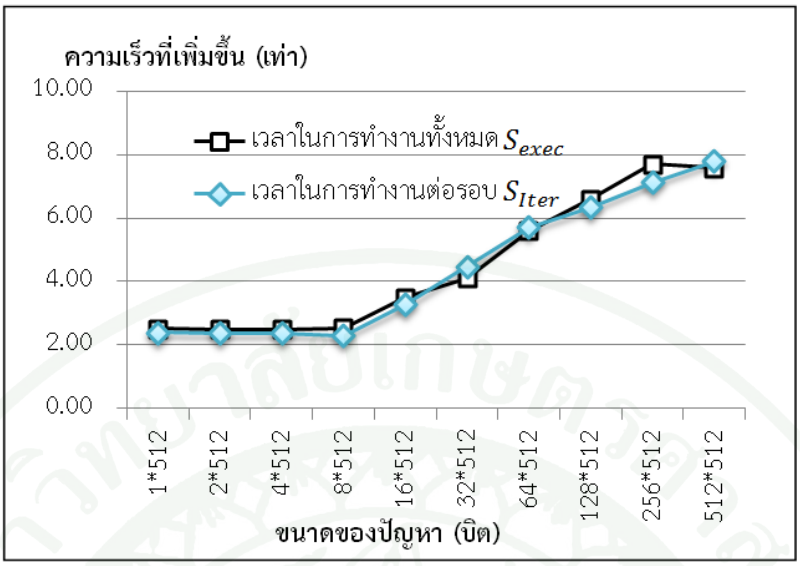
ตารางที่ 25 เวลาในการทำงานต่อรอบ (ms) ของ pcGA เมื่อใช้วิธีการวัดค่าความเหมาะสมแตกต่างกัน ในปัญหา OneMax

ขนาดปัญหา	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น
	เธรดเดียว	การลดทอน	S_{Iter}
1 × 512	0.1857	0.0771	2.4091
2 × 512	0.1834	0.0769	2.3866
4 × 512	0.1848	0.0772	2.3934
8 × 512	0.1847	0.0794	2.3257
16 × 512	0.3051	0.0931	3.2770
32 × 512	0.4841	0.1079	4.4855
64 × 512	0.8508	0.1488	5.7190
128 × 512	1.3907	0.2193	6.3429
256 × 512	2.7201	0.3805	7.1496
512 × 512	5.2603	0.6728	7.8184



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 32 เวลาในการทำงานของ pcGA แบบเธรดเดียว และ pcGA แบบการลดทอน (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ ในปัญหา OneMax



ภาพที่ 33 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA แบบการลดทอน เปรียบเทียบกับ pcGA แบบเธรดเดียว ในปัญหา OneMax

2.2 ปัญหา OneMax Noisy

ผลการทดลองกับปัญหา OneMax Noisy เมื่อปัญหามีขนาดแตกต่างกัน พบว่าเมื่อใช้ฟังก์ชันการสุ่มที่เหมือนกัน การวัดค่าความเหมาะสมแบบเธรดเดียว และการวัดค่าความเหมาะสมแบบการลดทอนใช้จำนวนครั้งในการทำงานเท่ากัน โดยการวัดค่าความเหมาะสมแบบการลดทอนใช้เวลาในการทำงานน้อยกว่าการวัดค่าความเหมาะสมแบบเธรดเดียว ในทุกขนาดปัญหา และในปัญหาขนาด 512×512 บิต การใช้การลดทอนจะช่วยให้สามารถทำงานได้เร็วกว่าการวัดค่าความเหมาะสมแบบเธรดเดียว 8.35 เท่า ดังตารางที่ 26

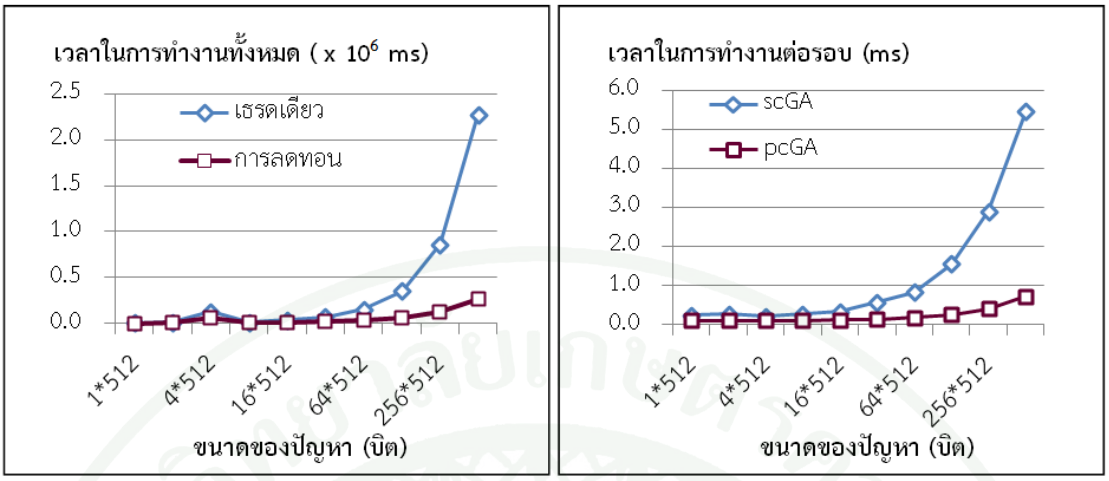
ผลการทดลองด้านเวลาในการทำงานต่อรอบของ pcGA พบว่า pcGA แบบที่ใช้การลดทอนวัดค่าความเหมาะสมใช้เวลาในการทำงานน้อยกว่า pcGA ที่วัดค่าความเหมาะสมโดยใช้เธรดเดียว และทั้งสองวิธีใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเล็กน้อยเมื่อปัญหามีขนาดใหญ่ขึ้น ดังแสดงในตารางที่ 27 เมื่อเปรียบเทียบความเร็วที่เพิ่มขึ้นของเวลาในการทำงานต่อรอบพบว่าในปัญหาขนาดใหญ่ 512×512 บิต pcGA แบบการลดทอนสามารถทำงานได้เร็วกว่า pcGA แบบเธรดเดียวถึง 7.59 เท่า

ตารางที่ 26 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ pcGA เมื่อใช้วิธีการวัดค่าความเหมาะสมแตกต่างกัน ในปัญหา OneMax Noisy

ขนาดปัญหา (บิต)	เวลา (ms)		จำนวนครั้งในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	เธรดเดียว	การลดทอน	เธรดเดียว	การลดทอน	
1 × 512	6,558	2,713	53,110	53,110	2.4179
2 × 512	10,512	4,097	76,636	76,636	2.5660
4 × 512	126,594	59,469	1,246,612	1,246,612	2.1287
8 × 512	15,978	7,811	151,468	151,468	2.0455
16 × 512	38,664	13,223	228,516	228,516	2.9240
32 × 512	72,325	21,062	321,390	321,390	3.4339
64 × 512	156,600	32,359	383,608	383,608	4.8394
128 × 512	351,796	58,739	494,156	494,156	5.9891
256 × 512	863,571	120,883	625,772	625,772	7.1439
512 × 512	2,272,595	272,086	857,060	857,060	8.3525

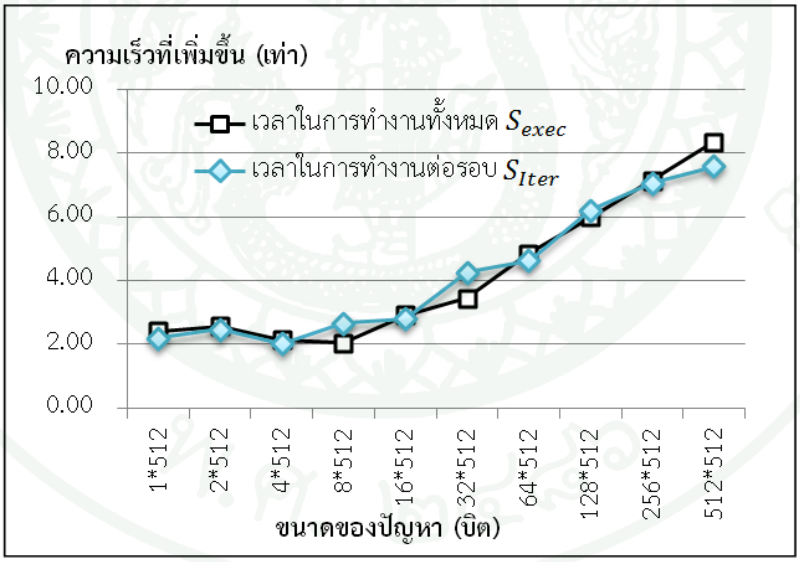
ตารางที่ 27 เวลาในการทำงานต่อรอบ (ms) ของ pcGA เมื่อใช้วิธีการวัดค่าความเหมาะสมแตกต่างกัน ในปัญหา OneMax

ขนาดปัญหา	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น S_{Iter}
	เธรดเดียว	การลดทอน	
1 × 512	0.2377	0.1084	2.1920
2 × 512	0.2667	0.1078	2.4736
4 × 512	0.2111	0.1031	2.0479
8 × 512	0.2739	0.1036	2.6437
16 × 512	0.3355	0.1201	2.7941
32 × 512	0.5797	0.1359	4.2643
64 × 512	0.8251	0.1784	4.6261
128 × 512	1.5546	0.2506	6.2029
256 × 512	2.8995	0.4106	7.0612
512 × 512	5.4725	0.7210	7.5900



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 34 เวลาในการทำงานของ pcGA แบบเธรดเดียว และ pcGA แบบการลดทอน (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ ในปัญหา OneMax Noisy



ภาพที่ 35 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA แบบการลดทอน กับ pcGA แบบเธรดเดียว ในปัญหา OneMax Noisy

2.3 ปัญหา Royal Road

ผลการทดลองกับปัญหา Royal Road เมื่อปัญหามีขนาดแตกต่างกัน พบว่าเมื่อใช้ฟังก์ชันการสุ่มที่เหมือนกัน การวัดค่าความเหมาะสมแบบเธรดเดียวและการวัดค่าความเหมาะสมแบบการลดทอนใช้จำนวนครั้งในการทำงานเท่ากัน และในปัญหาขนาดใกล้เคียงกันจะใช้จำนวนรอบในการทำงานเท่ากันแต่ใช้เวลาในการทำงานต่างกัน การวัดค่าความเหมาะสมแบบลดทอนใช้เวลาในการทำงานน้อยกว่าการวัดค่าความเหมาะสมแบบเธรดเดียวในทุกขนาดปัญหา และในปัญหาขนาด 512 × 512 บิต การใช้การลดทอนจะช่วยให้สามารถทำงานได้เร็วกว่าการวัดค่าความเหมาะสมแบบเธรดเดียว 8.84 เท่า ดังตารางที่ 28

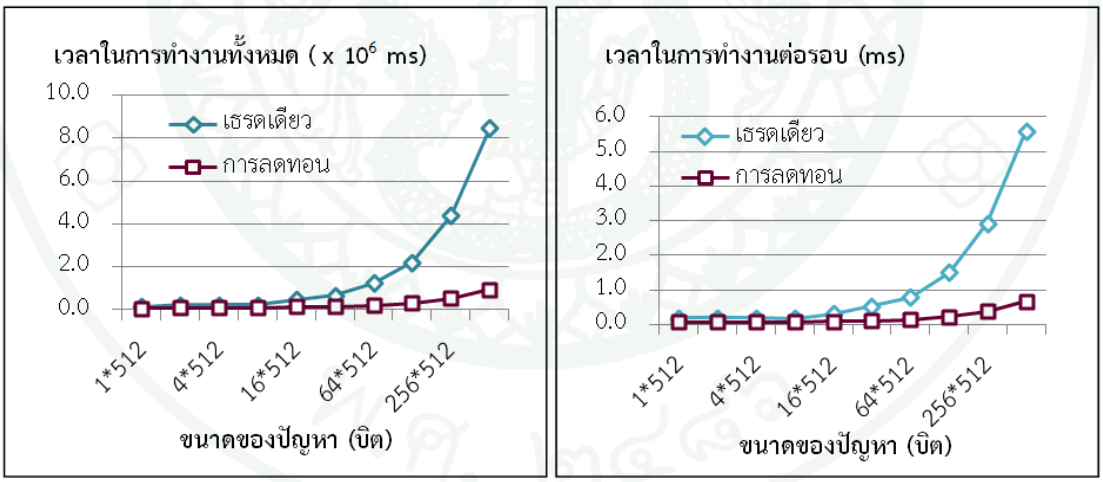
ผลการทดลองด้านเวลาในการทำงานต่อรอบของ pcGA พบว่า pcGA แบบที่ใช้การลดทอนวัดค่าความเหมาะสมใช้เวลาในการทำงานน้อยกว่า pcGA ที่ใช้เธรดดียวัดค่าความเหมาะสม และทั้งสองวิธีใช้เวลาในการทำงานต่อรอบเพิ่มขึ้นเล็กน้อยเมื่อปัญหามีขนาดใหญ่ขึ้น ดังแสดงในตารางที่ 29 เมื่อเปรียบเทียบความเร็วที่เพิ่มขึ้นของเวลาในการทำงานต่อรอบพบว่าในปัญหาขนาด 512 × 512 บิต pcGA แบบการลดทอนสามารถทำงานได้เร็วกว่า pcGA แบบที่ใช้เธรดเดียววัดค่าความเหมาะสมถึง 8.30 เท่า

ตารางที่ 28 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ pcGA เมื่อใช้วิธีการวัดค่าความเหมาะสมแตกต่างกัน ในปัญหา Royal Road

ขนาดปัญหา (บิต)	เวลา (ms)		จำนวนครั้งในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	เธรดเดียว	การลดทอน	เธรดเดียว	การลดทอน	
1 × 512	146,050	56,749	1,631,070	1,631,070	2.5736
2 × 512	226,943	87,776	2,596,828	2,596,828	2.5855
4 × 512	227,148	89,994	2,596,828	2,596,828	2.5240
8 × 512	228,022	91,223	2,596,828	2,596,828	2.4996
16 × 512	474,942	133,622	3,234,570	3,234,570	3.5544
32 × 512	667,094	155,705	3,234,570	3,234,570	4.2844
64 × 512	1,246,193	213,718	3,241,148	3,241,148	5.8310
128 × 512	2,210,810	315,976	3,241,708	3,241,708	6.9968
256 × 512	4,380,514	536,427	3,280,830	3,280,830	8.1661
512 × 512	8,482,515	959,350	3,282,546	3,282,546	8.8419

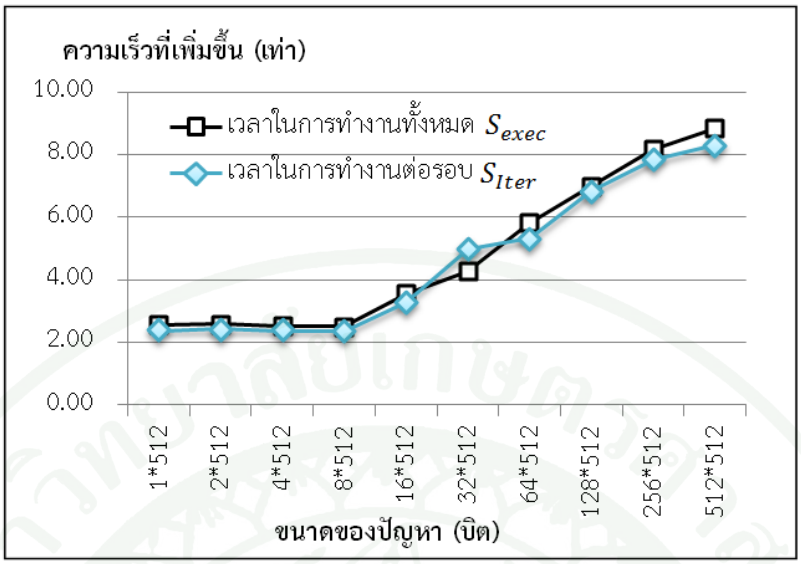
ตารางที่ 29 เวลา (ms) ในการทำงานต่อรอบของ pcGA เมื่อใช้วิธีการวัดค่าความเหมาะสมแตกต่างกัน ในปัญหา Royal Road

ขนาดปัญหา	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น
	เธรดเดียว	การลดทอน	s_{Iter}
1 × 512	0.1829	0.0759	2.4099
2 × 512	0.1830	0.0756	2.4194
4 × 512	0.1826	0.0757	2.4112
8 × 512	0.1816	0.0767	2.3691
16 × 512	0.3041	0.0924	3.2919
32 × 512	0.5332	0.1068	4.9935
64 × 512	0.7866	0.1475	5.3338
128 × 512	1.5004	0.2190	6.8505
256 × 512	2.9352	0.3736	7.8558
512 × 512	5.5893	0.6733	8.3017



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 36 เวลาในการทำงานของ pcGA แบบเธรดเดียว และ pcGA แบบการลดทอน (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ ในปัญหา Royal Road



ภาพที่ 37 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA แบบการลดทอน เปรียบเทียบกับ pcGA แบบเธรดเดียว ในปัญหา Royal Road

3. การทดลองเพื่อเปรียบเทียบประสิทธิภาพเมื่อจำนวนของเธรดในบล็อกแตกต่างกัน

การทดลองเพื่อเปรียบเทียบประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานทั้งแบบการลดทอน และแบบเธรดเดียวเมื่อจำนวนของเธรดในบล็อกแตกต่างกัน โดยใช้ฟังก์ชันการสุ่มดังสมการ (5) ทดสอบกับปัญหาทั้ง 3 แบบคือ ปัญหา OneMax, ปัญหา OneMax Noisy และปัญหา Royal Road ได้ผลดังนี้

3.1 ปัญหา OneMax

ผลการทดลองกับปัญหา OneMax เมื่อจำนวนของเธรดในบล็อกแตกต่างกัน พบว่า pcGA ที่ใช้การวัดค่าความเหมาะสมแบบเธรดเดียวจะใช้เวลาในการทำงานเพิ่มขึ้นเมื่อขนาดของเธรดในบล็อกเพิ่มขึ้น ในขณะที่ pcGA ที่ใช้การวัดค่าความเหมาะสมแบบการลดทอนใช้เวลาในการทำงานใกล้เคียงกันแม้จำนวนเธรดในบล็อกแตกต่างกัน ตารางที่ 30 แสดงให้เห็นว่าเมื่อจำนวนของเธรดในบล็อกมีขนาด 32 เธรด pcGA แบบที่วัดค่าความเหมาะสมโดยใช้เธรดเดียวจะใช้เวลาในการทำงานใกล้เคียงกับแบบที่ใช้การลดทอน แต่เมื่อขนาดของเธรดในบล็อกมากขึ้น pcGA ที่ใช้การวัดค่าความเหมาะสมแบบการลดทอน ยังคงเร็วกว่า pcGA แบบเธรดเดียว โดยในบล็อกขนาด 512 เธรด pcGA แบบการลดทอนสามารถทำงานได้เร็วกว่า pcGA แบบเธรดเดียว 3.50 เท่า

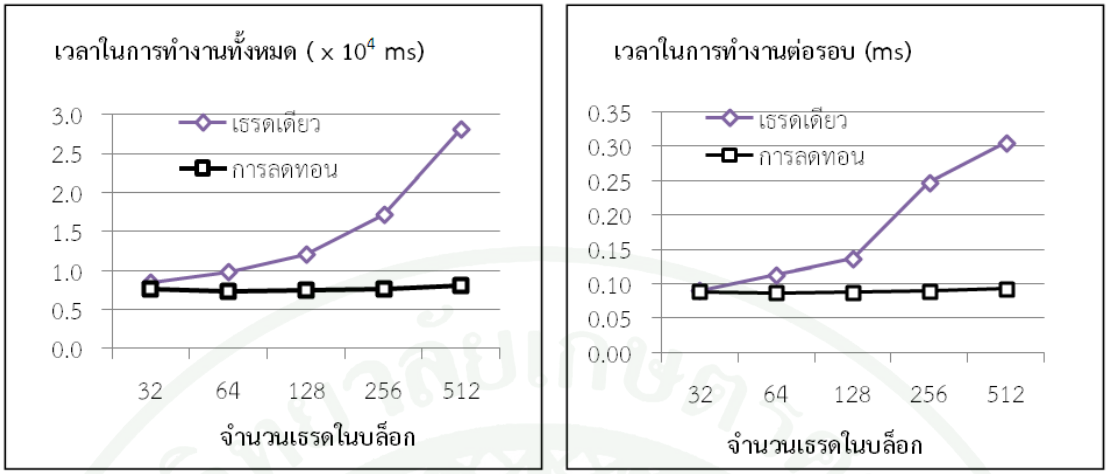
ตารางที่ 30 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ pcGA แบบการลดทอน และแบบเธรดเดียวเมื่อจำนวนเธรดในบล็อกแตกต่างกัน ในปัญหา OneMax

จำนวนเธรดในบล็อก	เวลาในการทำงาน (ms)		จำนวนครั้งในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	เธรดเดียว	การลดทอน	เธรดเดียว	การลดทอน	
32	8,501	7,681	185,944	185,944	1.1067
64	9,845	7,317	185,944	185,944	1.3456
128	12,078	7,532	185,944	185,944	1.6035
256	17,154	7,604	185,944	185,944	2.2559
512	28,172	8,044	185,944	185,944	3.5021

ผลการทดลองด้านเวลาในการทำงานต่อรอบ เมื่อจำนวนของเธรดในบล็อกแตกต่างกัน พบว่าเมื่อจำนวนเธรดในบล็อกมากขึ้น pcGA ที่ใช้การวัดค่าความเหมาะสมแบบเธรดเดียวจะใช้เวลาในการทำงานทำงานต่อรอบเพิ่มขึ้น ในขณะที่ pcGA ที่ใช้การวัดค่าความเหมาะสมแบบการลดทอนใช้เวลาในการทำงานเกือบจะคงที่ โดยในบล็อกที่มีเธรดขนาด 32 เธรด pcGA แบบที่วัดค่าความเหมาะสมโดยใช้เธรดเดียวจะใช้เวลาในการทำงานต่อรอบใกล้เคียงกับแบบที่ใช้การลดทอน และเมื่อขนาดของเธรดในบล็อกมากขึ้น pcGA ที่ใช้การวัดค่าความเหมาะสมแบบการลดทอนยังคงเร็วกว่า pcGA แบบเธรดเดียว โดยในบล็อกขนาด 512 เธรด pcGA แบบการลดทอนสามารถทำงานได้เร็วกว่า pcGA แบบเธรดเดียว 3.50 เท่า ดังแสดงในตารางที่ 31

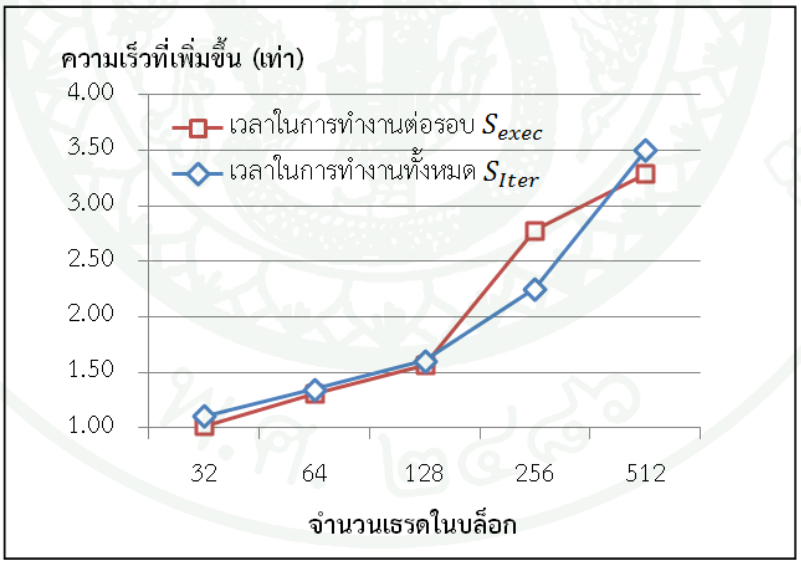
ตารางที่ 31 เวลาในการทำงานต่อรอบ (ms) ของ pcGA แบบการลดทอนและแบบเธรดเดียว เมื่อจำนวนเธรดในบล็อกแตกต่างกัน ในปัญหา OneMax

จำนวนเธรดในบล็อก	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น S_{Iter}
	เธรดเดียว	การลดทอน	
32	0.0903	0.0888	1.0172
64	0.1130	0.0865	1.3059
128	0.1370	0.0876	1.5640
256	0.2480	0.0892	2.7800
512	0.3054	0.0928	3.2889



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 38 เวลาในการทำงานของ pcGA แบบเธรดเดียว และ pcGA แบบการลดทอน เมื่อจำนวนเธรดในบล็อกแตกต่างกัน (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ ในปัญหา OneMax



ภาพที่ 39 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA แบบการลดทอน เปรียบเทียบกับ pcGA แบบเธรดเดียว เมื่อจำนวนเธรดในบล็อกแตกต่างกันในปัญหา OneMax

3.2 ปัญหา OneMax Noisy

ผลการทดลองกับปัญหา OneMax Noisy เมื่อจำนวนของเธรดในบล็อกแตกต่างกัน ได้ผลเช่นเดียวกับปัญหา OneMax นั่นคือ pcGA ที่ใช้การวัดค่าความเหมาะสมแบบเธรดเดียวจะใช้เวลาในการทำงานเพิ่มขึ้นเมื่อจำนวนเธรดในบล็อกเพิ่มขึ้น ในขณะที่ pcGA ที่ใช้การวัดค่าความเหมาะสมแบบการลดทอนใช้เวลาในการทำงานใกล้เคียงกันแม้จำนวนเธรดในบล็อกแตกต่างกัน ตารางที่ 32 แสดงให้เห็นว่าเมื่อจำนวนของเธรดในบล็อกมีขนาด 32 เธรด pcGA แบบที่วัดค่าความเหมาะสมโดยใช้เธรดเดียวจะใช้เวลาในการทำงานใกล้เคียงกับแบบที่ใช้การลดทอน แต่เมื่อขนาดของเธรดในบล็อกมากขึ้น pcGA ที่ใช้การวัดค่าความเหมาะสมแบบการลดทอน ยังคงเร็วกว่า pcGA แบบเธรดเดียว โดยในบล็อกขนาด 512 เธรด pcGA แบบการลดทอนสามารถทำงานได้เร็วกว่า pcGA แบบเธรดเดียว 2.92 เท่า

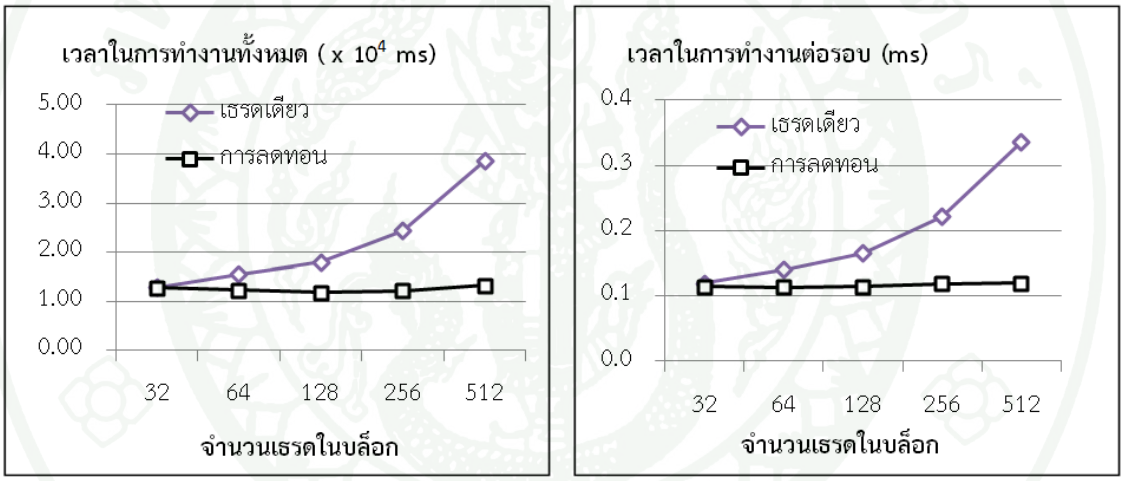
ผลการทดลองด้านเวลาในการทำงานต่อรอบดังแสดงในตารางที่ 33 พบว่าเมื่อขนาดของเธรดในบล็อกแตกต่างกันได้ผลเช่นเดียวกับปัญหา OneMax นั่นคือเมื่อจำนวนเธรดในบล็อกมากขึ้น pcGA ที่ใช้การวัดค่าความเหมาะสมแบบเธรดเดียวจะใช้เวลาในการทำงานทำงานต่อรอบเพิ่มขึ้นเล็กน้อย ในขณะที่ pcGA ที่ใช้การวัดค่าความเหมาะสมแบบการลดทอนใช้เวลาในการทำงานเกือบจะคงที่ โดยในบล็อกที่มีเธรดขนาด 32 เธรด pcGA แบบที่วัดค่าความเหมาะสมโดยใช้เธรดเดียวจะใช้เวลาในการทำงานต่อรอบใกล้เคียงกับแบบที่ใช้การลดทอน แต่เมื่อขนาดของเธรดในบล็อกมากขึ้น pcGA ที่ใช้การวัดค่าความเหมาะสมแบบการลดทอน ยังคงเร็วกว่า pcGA แบบเธรดเดียว โดยในบล็อกขนาด 512 เธรด pcGA แบบการลดทอนสามารถทำงานได้เร็วกว่า pcGA แบบเธรดเดียว 2.80 เท่า

ตารางที่ 32 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ pcGA แบบการลดทอนและแบบเธรดเดียวเมื่อจำนวนเธรดในบล็อกแตกต่างกัน ในปัญหา OneMax Noisy

จำนวนเธรดในบล็อก	เวลาในการทำงาน (ms)		จำนวนครั้งในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	เธรดเดียว	การลดทอน	เธรดเดียว	การลดทอน	
32	12,854	12,695	228,516	228,516	1.0125
64	15,462	12,221	228,516	228,516	1.2652
128	18,062	11,812	228,516	228,516	1.5291
256	24,399	12,118	228,516	228,516	2.0134
512	38,664	13,223	228,516	228,516	2.9240

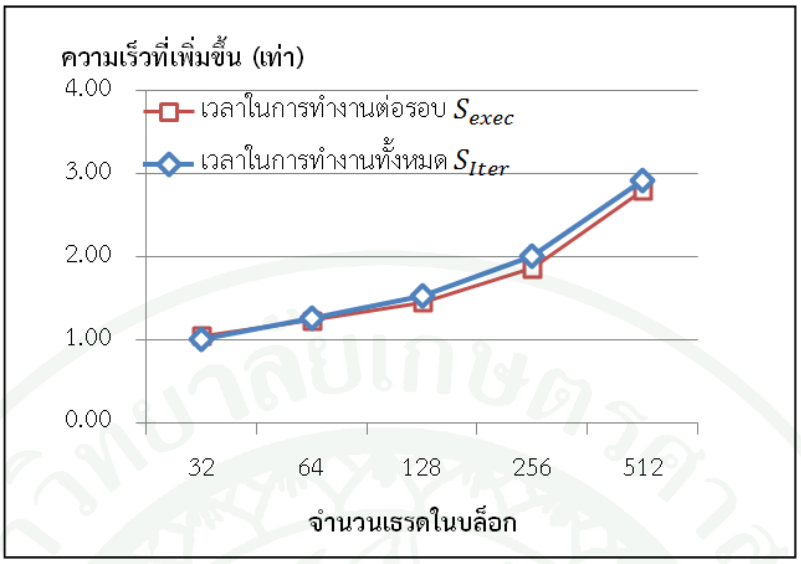
ตารางที่ 33 เวลาในการทำงานต่อรอบ (ms) ของ pcGA แบบการลดทอนและแบบเธรดเดียว เมื่อจำนวนเธรดในบล็อกแตกต่างกัน ในปัญหา OneMax Noisy

จำนวนเธรดในบล็อก	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น
	เธรดเดียว	การลดทอน	S_{Iter}
32	0.1194	0.1138	1.0488
64	0.1406	0.1136	1.2372
128	0.1657	0.1141	1.4522
256	0.2218	0.1187	1.8683
512	0.3359	0.1199	2.8011



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 40 เวลาในการทำงานของ pcGA แบบเธรดเดียว และ pcGA แบบการลดทอน เมื่อจำนวนเธรดในบล็อกแตกต่างกัน (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ ในปัญหา OneMax Noisy



ภาพที่ 41 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA แบบการลดทอน เปรียบเทียบกับ pcGA แบบเธรดเดียว เมื่อจำนวนเธรดในบล็อกแตกต่างกันในปัญหา OneMax Noisy

3.3 ปัญหา Royal Road

ผลการทดลองกับปัญหา Royal Road เมื่อจำนวนของเธรดในบล็อกแตกต่างกัน ได้ผลเช่นเดียวกับปัญหา OneMax และปัญหา OneMax Noisy นั่นคือ pcGA ที่ใช้การวัดค่าความเหมาะสมแบบเธรดเดียวจะใช้เวลาในการทำงานเพิ่มขึ้นเมื่อขนาดของเธรดในบล็อกเพิ่มขึ้น ในขณะที่ pcGA ที่ใช้การวัดค่าความเหมาะสมแบบการลดทอนใช้เวลาในการทำงานใกล้เคียงกันแม้จำนวนเธรดในบล็อกแตกต่างกัน ตารางที่ 34 แสดงให้เห็นว่าเมื่อจำนวนของเธรดในบล็อกมีขนาด 32 เธรด pcGA แบบที่วัดค่าความเหมาะสมโดยใช้เธรดเดียวจะใช้เวลาในการทำงานใกล้เคียงกับแบบที่ใช้การลดทอน แต่เมื่อขนาดของเธรดในบล็อกมากขึ้น pcGA ที่ใช้การวัดค่าความเหมาะสมแบบการลดทอน ยังคงเร็วกว่า pcGA แบบเธรดเดียว โดยในบล็อกขนาด 512 เธรด pcGA แบบการลดทอนสามารถทำงานได้เร็วกว่า pcGA แบบเธรดเดียว 3.64 เท่า

ผลการทดลองด้านเวลาในการทำงานต่อรอบ ดังแสดงในตารางที่ 35 พบว่าเมื่อขนาดของเธรดในบล็อกแตกต่างกันได้ผลเช่นเดียวกับปัญหา OneMax และปัญหา OneMax Noisy นั่นคือ เมื่อจำนวนเธรดในบล็อกมากขึ้น pcGA ที่ใช้การวัดค่าความเหมาะสมแบบเธรดเดียวจะใช้เวลาในการทำงานทำงานต่อรอบเพิ่มขึ้นเล็กน้อย ในขณะที่ pcGA ที่ใช้การวัดค่าความเหมาะสมแบบการลดทอนใช้เวลาในการทำงานเกือบจะคงที่ โดยในบล็อกที่มีเธรดขนาด 32 เธรด pcGA แบบที่วัดค่าความเหมาะสมโดยใช้เธรดเดียวจะใช้เวลาในการทำงานต่อรอบใกล้เคียงกับแบบที่ใช้การลดทอน แต่เมื่อขนาดของเธรดในบล็อกมากขึ้น pcGA ที่ใช้การวัดค่าความเหมาะสมแบบการลดทอน ยังคง

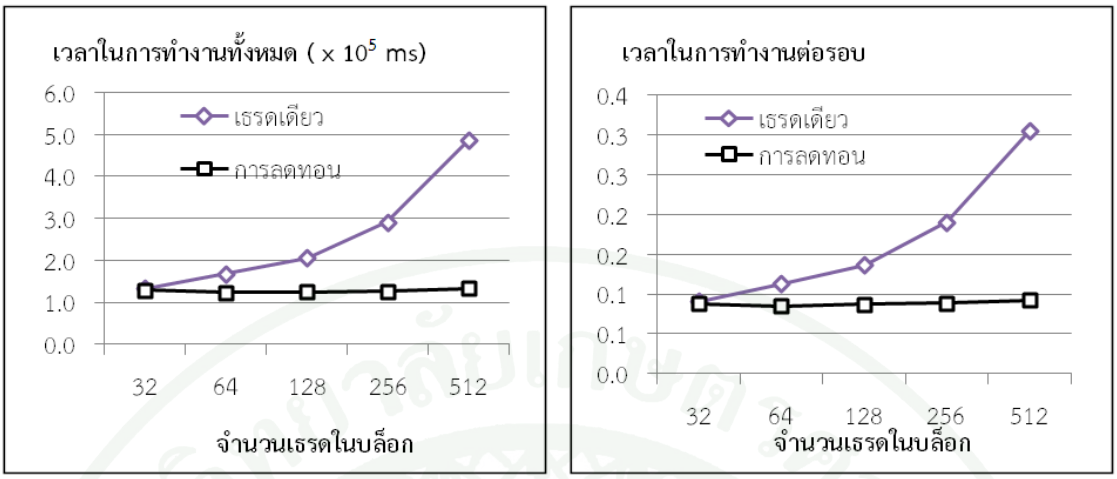
เร็วกว่า pcGA แบบเธรดเดียว โดยในบล็อกขนาด 512 เธรด pcGA แบบการลดทอนสามารถทำงานได้เร็วกว่า pcGA แบบเธรดเดียว 3.32 เท่า

ตารางที่ 34 เวลา (ms) และจำนวนครั้งในการวัดค่าความเหมาะสมของ pcGA แบบการลดทอนและแบบเธรดเดียวเมื่อจำนวนเธรดในบล็อกแตกต่างกัน ในปัญหา Royal Road

จำนวนเธรดในบล็อก	เวลาในการทำงาน (ms)		จำนวนครั้งในการวัดค่าความเหมาะสม		ความเร็วที่เพิ่มขึ้น S_{exec}
	เธรดเดียว	การลดทอน	เธรดเดียว	การลดทอน	
32	132,554	129,114	3,234,570	3,234,570	1.0266
64	167,859	123,645	3,234,570	3,234,570	1.3576
128	205,501	124,565	3,234,570	3,234,570	1.6498
256	291,483	126,703	3,234,570	3,234,570	2.3005
512	485,728	133,622	3,234,570	3,234,570	3.6351

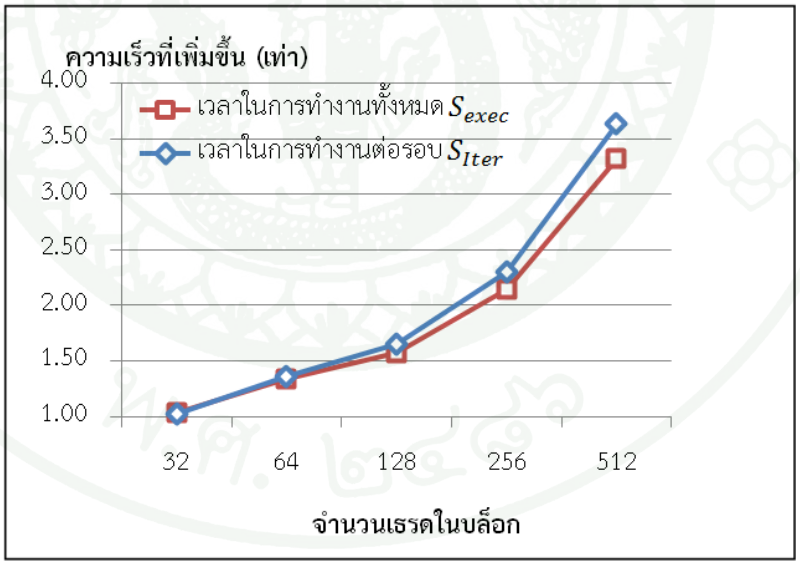
ตารางที่ 35 เวลาในการทำงานต่อรอบ (ms) ของ pcGA แบบการลดทอนและแบบเธรดเดียวเมื่อจำนวนเธรดในบล็อกแตกต่างกัน ในปัญหา Royal Road

จำนวนเธรดในบล็อก	เวลาในการทำงานต่อรอบ (ms)		ความเร็วที่เพิ่มขึ้น S_{Iter}
	เธรดเดียว	การลดทอน	
32	0.0908	0.0879	1.0332
64	0.1130	0.0847	1.3331
128	0.1368	0.0872	1.5681
256	0.1907	0.0888	2.1468
512	0.3056	0.0920	3.3205



(ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ

ภาพที่ 42 เวลาในการทำงานของ pcGA แบบเธรดเดียว และ pcGA แบบการลดทอน เมื่อจำนวนเธรดในบล็อกแตกต่างกัน (ก) เวลาในการทำงานทั้งหมด (ข) เวลาในการทำงานต่อรอบ ในปัญหา Royal Road



ภาพที่ 43 ความเร็วที่เพิ่มขึ้นในการทำงานของ pcGA แบบการลดทอน เปรียบเทียบกับ pcGA แบบเธรดเดียว เมื่อจำนวนเธรดในบล็อกแตกต่างกันในปัญหา Royal Road

วิจารณ์

การวิจารณ์ผลการทดลองแบ่งออกเป็น 3 กรณีคือ วิจารณ์ประสิทธิภาพในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน (pcGA) เปรียบเทียบกับขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบอนุกรม (scGA), วิจารณ์ประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน (pcGA) แบบที่ใช้เธรดเดียววัดค่าความเหมาะสมและแบบที่ใช้การวัดค่าความเหมาะสมแบบการลดทอน และวิจารณ์ประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน (pcGA) เมื่อจำนวนเธรดในบล็อกที่ใช้แตกต่างกัน

1. วิจารณ์ประสิทธิภาพในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน (pcGA) เปรียบเทียบกับขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบอนุกรม (scGA)

ผลการทดลองวัดประสิทธิภาพในการทำงานของ pcGA เปรียบเทียบกับ scGA เป็นไปตามสมมติฐาน คือ pcGA สามารถทำงานได้เร็วกว่า scGA เมื่อปัญหามีขนาดใหญ่ (8 x 512 บิตขึ้นไป) ทั้งในปัญหา OneMax, ปัญหา OneMax Noisy และปัญหา Royal Road สำหรับปัญหาขนาดเล็ก scGA สามารถทำงานได้เร็วกว่า pcGA เนื่องจากในปัญหาขนาดเล็กเวลาที่สูญเสียในการส่งข้อมูลระหว่างหน่วยประมวลผลกลางกับหน่วยประมวลผลกราฟิก และการสูญเสียที่เกิดการทำงานแบบสายท่อ (pipeline) ของหน่วยประมวลผลกราฟิก (Yu *et al*., 2005; Dan *et al*., 2010) ไม่คุ้มค่ากับการทำงานแบบขนาน ดังนั้นในปัญหาขนาดเล็กการทำงานบนหน่วยประมวลผลกลางยังคงให้ประสิทธิภาพที่ดีกว่า

อย่างไรก็ตามเมื่อปัญหามีขนาดใหญ่ขึ้นเวลาในการทำงานของ scGA จะเพิ่มขึ้นแบบแปรผันตรงกับขนาดของปัญหา แต่การทำงานของ pcGA ในหน่วยประมวลผลกราฟิก ทุกเธรดสามารถทำงานได้พร้อมกัน จึงทำให้ใช้เวลาในการทำงานเพิ่มขึ้นเพียงเล็กน้อย เห็นได้จากเวลาในการทำงานต่อรอบของ scGA จะเพิ่มขึ้น 2 เท่าเมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่าเสมอ ในขณะที่ pcGA จะใช้เวลาเพิ่มขึ้น 1.00–1.8 เท่า เมื่อปัญหามีขนาดใหญ่ขึ้น 2 เท่า

จำนวนรอบในการทำงานบนหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกเมื่อใช้ฟังก์ชันการสุ่ม rand() และฟังก์ชันการสุ่มตามสมการ (5) แตกต่างกันเนื่องจากลำดับการสุ่มมีความแตกต่างกันโดยบนหน่วยประมวลผลกลางลำดับการสุ่มจะใช้ลำดับเดียวกันในการทำงาน ในขณะที่ฟังก์ชันการสุ่มตามสมการ (5) บนหน่วยประมวลผลกราฟิกแต่ละเธรดจะใช้ลำดับการสุ่มที่แยกจากกัน

ผลการทดลองเมื่อใช้ฟังก์ชันการสุ่มตามสมการ (5) เหมือนกันทั้งบนหน่วยประมวลผลกราฟิกและหน่วยประมวลผลกลางมีความแตกต่างกันในปัญหาบางขนาด ซึ่งไม่เป็นไปตามสมมติฐานจึงได้ศึกษาเพิ่มเติมพบว่าเกิดความคลาดเคลื่อนจากหลักการทำงานของเลขทศนิยมบนหน่วยประมวลผล

กลางและหน่วยประมวลผลกราฟิกที่มีความแตกต่างกัน (Whitehead and Fit-Florea, 2011) ส่งผลต่อการแปรสภาพเป็น 0 หรือ 1 ของเวกเตอร์ความน่าจะเป็นทำให้จำนวนรอบในการทำงานไม่เท่ากัน

ผลการทดลองเมื่อใช้ฟังก์ชันการสุ่ม rand() และ CURAND() พบว่าการใช้ฟังก์ชันการสุ่ม CURAND() ส่งผลให้ประสิทธิภาพของ pcGA แย่ลง เนื่องจากฟังก์ชัน CURAND() ใช้เวลาในการทำงานมากกว่าฟังก์ชัน LCG ที่สร้างขึ้นตามสมการ (5) ซึ่งได้ทดสอบโดยการจับเวลาในการเรียกใช้ kernel สำหรับการสร้างเลขสุ่มทั้งสองแบบบนแถวลำดับขนาดต่าง ๆ และตัดโปรแกรมในส่วนที่อาจก่อให้เกิดการสูญเสียจากค่าใช้จ่ายอื่นออกไป ผลปรากฏว่าฟังก์ชันการสุ่มตามสมการ (5) ใช้เวลาเพิ่มขึ้นเล็กน้อยในขณะที่ฟังก์ชันการสุ่ม CURAND() ใช้เวลาเพิ่มขึ้นเมื่อปัญหามีขนาดใหญ่ขึ้นโดยในปัญหาขนาด 512 x 512 บิต ฟังก์ชันการสุ่ม CURAND() ใช้เวลาในการทำงานมากกว่าฟังก์ชันการสุ่ม LCG ตามสมการ (5) ถึง 83.24 เท่า ดังแสดงในตารางที่ 36 และจากการศึกษาเพิ่มเติมพบว่าประสิทธิภาพด้านเวลาในการทำงานของฟังก์ชันการสุ่ม CURAND() ขึ้นกับสถาปัตยกรรมโดยฟังก์ชันการสุ่ม CURAND() จะให้ประสิทธิภาพที่ดีในหน่วยประมวลผลกราฟิกที่มีสถาปัตยกรรมใหม่ ๆ เท่านั้น (Nandapalan et al., 2011)

ตารางที่ 36 แสดงเวลาในการทำงาน (ms) ของฟังก์ชันการสุ่ม CURAND() และฟังก์ชันการสุ่มตามสมการ (5) เมื่อขนาดของแถวลำดับแตกต่างกัน

ขนาดของแถวลำดับ	เวลาในการทำงาน (ms)		เวลาที่เพิ่มขึ้น (เท่า)
	LCG	CURAND()	
1 × 512	0.017	0.043	2.56
2 × 512	0.017	0.046	2.71
4 × 512	0.017	0.075	4.37
8 × 512	0.018	0.138	7.81
16 × 512	0.019	0.303	16.31
32 × 512	0.022	0.521	24.15
64 × 512	0.026	1.033	39.30
128 × 512	0.026	1.909	71.89
256 × 512	0.046	3.589	77.88
512 × 512	0.083	6.927	83.03

2. วิจารณ์ประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน (pcGA) แบบที่ใช้เธรดเดียววัดค่าความเหมาะสมและใช้การวัดค่าความเหมาะสมแบบลดทอน

จากการทดลองเพื่อเปรียบเทียบประสิทธิภาพของ pcGA ที่วัดค่าความเหมาะสมโดยใช้เธรดเดียวและแบบที่วัดค่าความเหมาะสมโดยวิธีลดทอน พบว่าการวัดค่าความเหมาะสมโดยใช้การลดทอนสามารถทำงานได้เร็วกว่า ทั้งในปัญหา OneMax, ปัญหา OneMax Noisy และ ปัญหา Royal Road เนื่องจากในการลดทอน ทุกเธรดในบล็อกเดียวกันสามารถทำงานได้พร้อมกัน ในขณะที่การวัดค่าความเหมาะสมโดยใช้เธรดเดียว เธรดอื่นในบล็อกต้องรอการทำงานของเธรดที่ใช้วัดค่าความเหมาะสม

แต่อย่างไรก็ดี pcGA แบบที่ใช้เธรดเดียววัดค่าความเหมาะสมยังคงให้ประสิทธิภาพที่ดีกว่า scGA เมื่อปัญหามีขนาด 8×512 บิตขึ้นไป เนื่องจาก pcGA แบบเธรดเดียวจะวัดค่าความเหมาะสมภายในบล็อกเท่านั้น แล้วนำผลที่ได้แต่ละบล็อกมาหาค่าความเหมาะสมรวมของโครโมโซม เวลาที่ใช้ในการวัดค่าความเหมาะสมจะเท่ากับ $O(k)+ O(n)$ เมื่อ k คือจำนวนเธรดในบล็อกและ n คือจำนวนบล็อก แต่ scGA จะวัดค่าความเหมาะสมทั้งหมดของโครโมโซม จึงใช้เวลาในการวัดค่าความเหมาะสมเท่ากับ $O(l)$ เมื่อ l คือความยาวของโครโมโซม

3. วิจารณ์ประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน (pcGA) เมื่อจำนวนเธรดในบล็อกแตกต่างกัน

จากการทดลองเพื่อเปรียบเทียบประสิทธิภาพการทำงานของ pcGA แบบการลดทอนและแบบเธรดเดียว เมื่อจำนวนเธรดในบล็อกแตกต่างกัน พบว่าบล็อกขนาดเล็กให้ประสิทธิภาพด้านเวลาในการทำงานที่ดีกว่า โดยเฉพาะ pcGA แบบเธรดเดียวซึ่งในบล็อกขนาดเล็ก pcGA แบบเธรดเดียวสามารถทำงานได้เร็วเทียบเท่ากับ pcGA แบบการลดทอน ทั้งในปัญหา OneMax, ปัญหา OneMax Noisy และปัญหา Royal Road เนื่องจากเวลาในการวัดค่าความเหมาะสมของ pcGA แบบเธรดเดียวเท่ากับ $O(n)+ O(k)$ เมื่อ n คือจำนวนเธรดในบล็อก และ k คือจำนวนบล็อก ดังนั้นเมื่อจำนวนของเธรดในบล็อกลดลง จะใช้จำนวนบล็อกในการทำงานมากขึ้นซึ่งบล็อกเหล่านี้สามารถทำงานพร้อมกันแบบขนานได้ ส่งผลให้เวลาในการวัดค่าความเหมาะสมของ pcGA แบบเธรดเดียวลดน้อยลง ในขณะที่การลดจำนวนเธรดในบล็อกจะส่งผลต่อเวลาในการทำงานของ pcGA แบบการลดทอนเพียงเล็กน้อย เนื่องจากเวลาในการทำงานของ pcGA แบบการลดทอนเท่ากับ $O(\log n)$ เมื่อ n คือจำนวนเธรดในบล็อก โดยในแบบจำลองการเขียนโปรแกรม CUDA จำกัดจำนวนเธรดในบล็อกได้ไม่เกิน 512 เธรด ในการทดลองจึงสามารถวัดประสิทธิภาพบนจำนวนเธรดในบล็อกขนาด 32, 64, 128, 512 เท่านั้น จำนวนเธรดในบล็อกจึงส่งผลต่อเวลาในการวัดค่าความเหมาะสมของ pcGA แบบการลดทอนเพียงเล็กน้อย แต่หากจำนวนเธรดในบล็อกมีขนาดแตกต่างกันมากขึ้นจะส่งผลต่อเวลาในการทำงานของ pcGA แบบการลดทอน

สรุปและข้อเสนอแนะ

สรุป

วัตถุประสงค์หลักของงานวิจัยนี้คือการเพิ่มความเร็วในการทำงานให้กับขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ ซึ่งผู้วิจัยได้นำเสนอการใช้แบบจำลองการเขียนโปรแกรม CUDA ในการพัฒนาการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน (pcGA) ที่มีการทำงานส่วนใหญ่บนหน่วยประมวลผลกราฟิก

1. ประสิทธิภาพในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน (pcGA) เปรียบเทียบกับขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบอนุกรม (scGA)

การทดลองเมื่อเปรียบเทียบประสิทธิภาพกับขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบอนุกรม (scGA) ที่ทำงานบนหน่วยประมวลผลกลางเพียงอย่างเดียวบนฟังก์ชันการสุ่มแบบต่างๆ พบว่าในปัญหาขนาด 1 x 512 ถึง 4 x 512 ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบอนุกรมจะให้ประสิทธิภาพที่ดีกว่า แต่เมื่อปัญหามีขนาด 8 x 512 บิตขึ้นไป ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานจะทำงานได้เร็วกว่าในทุกปัญหาทดสอบ

2. ประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน (pcGA) แบบที่ใช้เธรดเดียววัดค่าความเหมาะสมและใช้การวัดค่าความเหมาะสมแบบลดทอน

เวลาในการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์ส่วนใหญ่ใช้ในการวัดค่าความเหมาะสม ดังนั้นวิธีวัดค่าความเหมาะสมที่ใช้เวลาในการทำงานน้อยจะยิ่งส่งผลให้ขั้นตอนวิธีเชิงพันธุกรรมแบบขนานทำงานได้เร็วขึ้น ซึ่งผู้วิจัยพบว่าในทุกปัญหาทดสอบขั้นตอนวิธีเชิงพันธุกรรมแบบขนานที่ใช้การวัดค่าความเหมาะสมแบบการลดทอนสามารถทำงานได้เร็วกว่าขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบที่ใช้เธรดเดียววัดค่าความเหมาะสม

3. ประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาน (pcGA) เมื่อจำนวนเธรดในบล็อกแตกต่างกัน

ประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนาดทั้งแบบที่ใช้การวัดค่าความเหมาะสมแบบเธรดเดียวและแบบที่ใช้การวัดค่าความเหมาะสมแบบการลดทอนขึ้นอยู่กับขนาดของบล็อก โดยในบล็อกขนาดเล็ก pcGA จะสามารถทำงานได้เร็วกว่าบล็อกที่มีเธรดจำนวนมาก

ข้อเสนอแนะ

งานวิจัยนี้ศึกษาเกี่ยวกับการทำงานของขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์แบบขนานบนหน่วยประมวลผลกราฟิก ซึ่งช่วยให้ขั้นตอนวิธีเชิงพันธุกรรมคอมแพคท์สามารถทำงานได้เร็วขึ้น แสดงให้เห็นว่าหน่วยประมวลกราฟิกสามารถทำงานแบบขนานได้อย่างมีประสิทธิภาพเมื่อมีอัลกอริทึมที่เหมาะสม ดังนั้นควรศึกษาการนำหน่วยประมวลกราฟิกไปใช้กับงานด้านอื่น ๆ ต่อไป

เอกสารและสิ่งอ้างอิง

Cantú-Paz, E. 1998. A Survey of Parallel Genetic Algorithms. **Calculateurs Paralléles: Réseaux et Systémes répartis** 10(2).

Goldberg, D. E. 1989. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Addison-Wesley Professional, USA.

_____, K. Deb and J. H. Clark. 1992. Genetic algorithms, noise, and the sizing of populations. **Complex Systems** 6.

Giguére, P. and D. E. Goldberg. 1998. Population Sizing for Optimum Sampling with Genetic Algorithms: A Case Study of the Onemax Problem. pp. 855–862. In **Proceedings of the Third Annual Genetic Programming Conference** Madison, WI. July 22–25.

Harik, G.R., F.G. Lobo and D.E. Goldberg. 1998. The compact genetic algorithm. pp. 523–528. In **Proceedings of the International Conference on Evolutionary Computation (ICEC'98)**, Piscataway.

Nickolls, J., I. Buck, M. Garland, and K. Skadron. 2008. Scalable parallel programming with CUDA. **Queue 6(2)**.

Li, J.-M., X.-J. Wang, R.-S. He and Z.-X. Chi. 2007. An effient fine-grained parallel genetic algorithm based on gpu-accelerated**.** pp. 855–862. In **Network and Parallel Computing Workshops 2007 NPC Workshops IFIP International Conference**.

Lobo, F.G., C.F. Lima and H. Mártires. 2004. An Architecture for Massive Parallelization of the Compact Genetic Algorithm**. Genetic and Evolutionary Computation – GECCO 2004 Lecture Notes in Computer Science** 3103/2004.

Luong,T.V., Melab, N. and Talbi ,E.-G.. 2010. GPU-based island model for evolutionary algorithms. pp. 1089-1096. In **Genetic And Evolutionary Computation Conference, Proceedings of the 12th annual conference on Genetic and evolutionary computation**.

Maitre, O., L.A., Baumes, N., Lachiche, A., Corma and P., Collet. 2009. Coarse grain parallelization of evolutionary algorithms on GPGPU cards with EASEA. pp. 1403-1410. In **Proceedings of the 11th Annual Conference on Genetic and evolutionary computation table of contents**, Montreal, Québec, Canada.

Mitchell, M. 1998. An Introduction to Genetic Algorithms (Complex Adaptive Systems)**. MIT Press**. Cambridge, Massachusetts.

_____., and S. Forrest and J.H. Holland. 1992. The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance. pp. 245-254. In **Proceedings of the First European Conference on Artificial Life Cambridge**, MA:MIT Press.

Nandapalan, N., R. P. Brent, L.M. Murray and A. Rendell. 2011. High-performance pseudo-random number generation on graphics processing units. In **Proceeding of the 9th International Conference Parallel Processing and Applied Mathematics**, Torun, Poland, September 11-14, 2011.

NVIDIA Corporation. 2010a. **NVIDIA CUDA C Programming Guide**. CUDA Developer zone: http://developer.download.nvidia.com/compute/cuda/31/toolkit/docs/CUDA_C_Programming_ Guide_3.1.pdf. December 20, 2010.

_____. 2010b. **CUDA CURAND Library.** CUDA Developer zone**:** http://developer.nvidia.com/curand. October 22, 2010.

_____. 2012a. **CUDA C Best Practices Guide.** CUDA Developer zone**:** http://developer.download.nvidia.com/compute/cuda/41/toolkit/docs/CUDA_C_Best_Practices_Guide_4.1.pdf. January 9, 2012.

_____. 2012b. **Geforce GT 240**. NVIDIA Product**:** http://www.nvidia.com/object/ product_geforce_gt_240_us.html. May 1, 2012.

Qizhi, Y., C. Chen. and Z. Pan. 2005. Parallel Genetic Algorithms on Programmable Graphics Hardware. pp. 1051–1059. In **Advances in Natural Computation ICNC 2005, Proceedings** 3612.

Sastry, K., D.E. GlodBerg and X. Llorá. 2007. Towards billion bit optimization via parallel estimation of distribution algorithm. pp. 577-584. In **Genetic and Evolutionary Computation Conference-GECCO 2007.**

Sataporn, N. and W. Suwannik. 2010. Compact Genetic Algorithm with CUDA**.** pp. 129-132. In **conference on Computational Intelligence and Vehicular System,** CIVS 2010, Korea (South) , November 22-23.

Whitehead, N. and A. Fit-Florea. 2010. Precision & Performane: Floating Point and IEEE 754 Compliance for NVIDIA GPUs. In **NVIDIA GPU Technology Conference 2011**.

D. Zhang, Z. Rongcai, H. Lin, W. Tao and Q. Jin. 2010. A Parallel Cost Model for GPU**.** In **2010 International Conference on Computer and Communication Technologies in Agriculture Engineering**.

ภาคผนวก

2010 International Conference on Computational Intelligence and Vehicular System (CIVS)

Compact Genetic Algorithm with CUDA

Nugool Sataporn
Department of Computer Science
Kasetsart University
Bangkok, Thailand
nugool.sataporn@gmail.com,

Worasait Suwannik
Department of Computer Science
Kasetsart University
Bangkok, Thailand
worasait.suwannik@gmail.com

***Abstract* - CUDA is a parallel computing architecture invented by NVIDIA. Developers can run a general purpose program on a GPU. This papers presents an implementation of parallel compact genetic algorithm (cGA) using CUDA. The experimental result shows that for a large problem (i.e., 512×512 bits OneMax problem), CUDA implementation of cGA achieves 23.58 times speedup.**

Index Terms – CUDA, GPU, compact genetic algorithm.

I. Introduction

Genetic Algorithm (GA) [1] is a search method inspired by natural evolution. GA chromosome is encoded as binary string. To store the whole population of GA chromosome required the order of $n \times l$ memory space, where n is a population size and l is a chromosome length.

Compact Genetic Algorithm (cGA) [2] uses a probability vector to represent the whole GA population. The length of the probability vector is equal to the chromosome length. Therefore, it requires the order of l memory space, which is less than GA.

There are several implementations of parallel cGA. Fernando proposed a manager-worker architecture for parallelizing cGA [3]. They conducted the experiment in simulation and found that when the number of processor are doubled, the evaluation time is cut by half because each processor reduce bit of probability vector for execution. Kumara implements a master-slave parallelizing cGA on a cluster that consists of 1280 computers. Their implementation has linear speedup with the number of processors [4].

CUDA (compute unified device architecture) is a parallel computing architecture invented by NVIDIA. It enabled a general purpose program to run on a graphics card. This paper presents an implementation of cGA on using CUDA.

The organization of this paper is as follows. The next section describes cGA. Section 3 gives a brief explanation about CUDA. Section 4 describes an implementation of cGA with CUDA. Section 5 and 6 gives an experimental details and result. Section 7 concludes the paper.

II. Compact Genetic Algorithm

In cGA, a whole population of GA chromosome is represented by a single probability vector. Therefore, the algorithm requires less memory than GA (hence the name compact). The initial value of each element in the probability vector is 0.5. This means that, initially, each bit has an equal chance to be 0 or 1. The probability vector is used to generate two individuals a and b. The fitter individual is called the winner. The probability vector is updated according to a value of the winner. The pseudocode of cGA is shown in Figure 1.

1) initialize probability vector
   $for\ i \leftarrow 1\ to\ l\ do$
   $p[i] \leftarrow 0.5$

2) generate two individuals from the vector
   $a \leftarrow generate(p)$
   $b \leftarrow generate(p)$

3) let them compete
   $winner, loser \leftarrow evaluate(a, b)$

4) update the probability vector towards the better one
   $for\ i \leftarrow 1\ to\ l\ do$
   $if\ winner[i] loser[i]\ then$
   $if\ winner[i] = 1\ then$
   $p[i] \mathrel{+}= 1/n$
   $else$
   $p[i] \mathrel{-}= 1/n$

5) check if the vector has converged
   $for\ i \leftarrow 1\ to\ l\ do$
   $if\ p[i] > 0\ and\ p[i] < 1\ then$
   $return\ to\ step\ 2$

6) p represents the final solution

Fig.1 Pseudocode of compact genetic algorithm

III. CUDA

A. Threads and Thread Blocks

CUDA programming model extends C and C++ languages [6,7]. There are two type of code: host and device code. Host code runs on CPU while device code runs on a graphics processing unit (GPU). A function that can run parallelly on GPU is called a *kernel*. Several threads can execute the kernel code simultaneously. A CUDA thread block consists of up to 512 threads. Threads in the same block can be synchronized. However, thread blocks cannot be synchronized.

978-1-4244-8717-2/10/$26.00 ©2010 IEEE

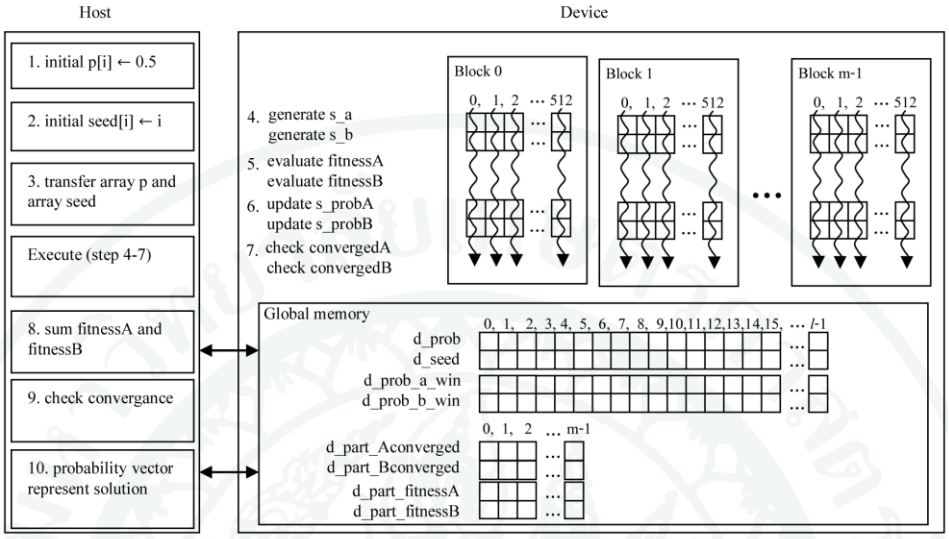


Fig 2. Diagram of our CUDA implementation. *l* is the problem size. *m* is the number of thread block.

B. *Memory Performance*

A CUDA thread can access several types of memory but our implementation concerns only three of them.

- Registers store local variables of each thread. Accessing register memory is very fast.
- Global memory or device memory is a largest memory space that a CUDA thread can access. A program can transfer data from device memory to host memory. Accessing device memory is very slow because it is off the GPU.
- Shared memory is used to share data between threads in the same thread block. Accessing shared memory is very fast. It is 100-150 times faster than global memory.

IV. cGA WITH CUDA

cGA models the GA population in a single probability vector. Each thread process each element in the vector. Each thread has the following tasks.

- Generating one bit in individual *a* and *b* using an element in the probability vector.
- Update the element in the probability vector.
- Check the convergence of the element.

A thread block can consists of at most 512 threads. For problems larger than 512 bits, more than one thread block will be launched. Each block has 512 threads. Unfortunately, CUDA thread block cannot be synchronized except when execution goes back to host and the program call a particular CUDA function. A naïve CUDA implementation of cGA requires two switches per iteration: one in step 4 and the other is in step 5 of Figure 1. Moreover, each of those switches takes time because data should be sent to a host.

We propose a technique that can avoid data transfer between host and device by adding two more probability vectors, our implementation of cGA with CUDA has one switch per iteration. One vector is for the case that an individual *a* wins.
The other is for the case that *b* wins.The overview of our implementation is shown in Figure 2. The pseudocode of our implementation is shows in Figure 3.

cGA requires random numbers to generate individuals. However, there is no random function in CUDA. Therefore, we write Linear Congruential Generator (equation 1). The value of *a*, *c*, and *m* is equal to 214013, 2531011, and 2^{32} respectively [5]. Each thread has its own pseudo random sequence. The seed for each thread is the index of the element in the probability vector that the thread processes.

$$x_{n+1} = (ax_n + c) \bmod m \quad (1)$$

In each iteration, data transfer from device to host is minimal (step 7). The probability vectors is transferred only twice (i.e., when the algorithm starts and before it stops).

V. EXPERIMENT

Our implementation of serial and CUDA cGA runs on NVIDIA GeForce GT 240 (12 processor, 96 cores) and Intel Core 2 Quad CPU Q8200 2.33GHz. The problem is OneMax or bit counting problem. The problem sizes start from 512 bits and are increased by double until 512×512 bits. We repeat each experiment for 50 times. The number of thread per block is 512 threads.

```
1. Host: Initialize probability vector
   d_prob. Each element is set to 0.5.
2. Host: Initialize seed array d_seed.
   The length of the array is equal to
   the problem size. Each seed is set
   to its index in the array in order to
   make every seed different from each
   other.
3. Device: Generate 2 individuals
     a. Copy the probability vector
        d_prob to a shared memory s_prob
        in order to make the latter
        access faster.
     b. Generate two individuals a and b
        from s_prob.
4. Device: Evaluate fitness
     a. Evaluate partial fitness of a and
        b using reduction. Reduction is
        a technique to parallelly sum all
        values in an array.
     b. Stored the partial fitness of
        each block in an element in the
        array d_part_fit_a and
        d_part_fit_b.
5. Device: Calculate the probability in
   the case that a wins in d_prob_a and
   b wins in d_prob_b.
6. Device: Check convergence
     a. Check the convergence in each
        block.
     b. Store the convergence in
        d_part_conv_a and d_part_conv_b
7. Host: Find the winner
a. Transfer d_part_fit_a, d_part_fit_a,
   d_part_conv_a, and d_part_conv_b from
   device to host. The length of these
   arrays are short (i.e., equal to the
   number of thread blocks)
b. Sum the value of d_part_fit_a and
   d_part_fit_b to find the winner
   between a and b.
8. Host: Check convergence of the
   winner.
     a. If the winner is not converge,
```

Fig. 3 Pseudo code of CUDA cGA. The step that begins with "Host:" runs on the host. The one that begins with "Device:" runs on the device. The variable with a prefix d_ and s_ stores in global and shared memory respectively.

VI. Result

Figure 4 shows execution time speedup and evaluation time speedup. Execution time speedup is defined by Equation 2. Evaluation time speedup is defined by Equation 3. As can be seen from Figure 3, when the problem size is greater than 8×512 bits, CUDA cGA is faster than serial cGA. When the problem size is 512×512 bits, the fitness evaluation time speedup is 23.59. The evaluation time speedup is greater than execution time speedup because our CUDA implementation requires more fitness evaluations than our serial implementation.

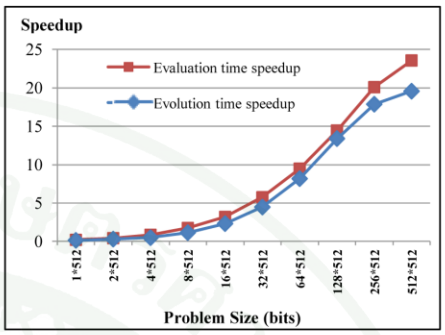


Fig. 4 CUDA speedup

$$S_{exec} = \frac{T_{evo\_cpu}}{T_{evo\_cuda}} \quad (2)$$

where:

T_{evo_cpu} is the execution time of serial implementation
T_{evo_cuda} is the execution time of CUDA implementation

$$S_{eval} = \frac{T_{iteration\_cpu}}{T_{iteration\_cuda}} \quad (3)$$

where:

$T_{iteration\_cpu}$ is the execution time each iteration of one individual in serial implementation
$T_{iteration\_cuda}$ is the execution time each iteration of one individual in CUDA implementation

VI. Conclusion and Future Work

We present a CUDA implementation of cGA. Our implementation required only one switch from device to host per iteration. For small problem, serial cGA is faster. However, when the problem becomes bigger, our CUDA cGA is faster.

Our CUDA cGA has more fitness evaluations than serial cGA. This is possibly because of the quality of the pseudo random generator. Therefore, further work to improve the generator which can reduce the number of fitness evaluation in CUDA cGA would increase speedup even more.

Although our test problem is OneMax, our approach can be used to solve various problems which the fitness can be partially evaluated. Our approach can be used to solve OneMax Noisy by modifying the step 7.b in Figure 3. Future work will address a CUDA implementation that can effectively solve a problem that the fitness cannot be partially evaluated.

ACKNOWLEDGMENT

This research was partly supported by Faculty of Science and The Graduate School of Kasetsart University , Thailand.

REFERENCES

[1] D. E. GoldBerg, "Genetic Algorithm in search optimization and Machine Learning," *Addison-Wesley Professional*, USA,1989

[2] G.R. Harik, , F.G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 287-309, November 1999.

[3] F.G. Lobo, C.F. Lima, and H. Mártires, "An Architecture for Massive Parallelization of the Compact Genetic Algorithm," *Genetic and Evolutionary Computation – GECCO 2004 Lecture Notes in Computer Science,* Volume 3103/2004, pp. 412-413, 2004.

[4] K. Sastry, D.E. GlodBerg, and X. Llorá, "Towards billion bit optimization via parallel estimation of distribution algorithm," *Genetic and Evolutionary Computation Comference-GECCO 2007*, pp. 577-584, 2007.

[5] S. Aupetit1, P. Liardet, and M. Slimane, "Evolutionary Search for Binary Strings with Low Aperiodic Auto-correlations," S*pringer-Verlag Berlin Heidelberg 2004*, pp. 39–50, 2004.

[6] NVIDIA Corporation, "NIVIDA CUDA C Programing Guide," *http://developer.nvidia.com/object/gpu_programming_guide.html*, version 3.1.1, 2010.

[7] NVIDIA Corporation, "NVIDIA GPU Computing Developer," *http://developer.nvidia.com/object/gpucomputing.htm*, 2010.

DEVICE CODE

```
__global__ void evaluateCga(float* prob,int*
d_seed,float* d_probAwin,float* d_probBwin,int*
d_fitnessA,int* d_fitnessB,int* d_Aconverge,int*
d_Bconverge)
{
    __shared__ float s_prob[threadsPerBlock];
    __shared__ float s_prob_a_win[threadsPerBlock];
    __shared__ float s_prob_b_win[threadsPerBlock];
    __shared__ int s_a[threadsPerBlock];
    __shared__ int s_b[threadsPerBlock];
    __shared__ int s_aa[threadsPerBlock];
    __shared__ int s_bb[threadsPerBlock];
    __shared__ int s_Aconverge;
    __shared__ int s_Bconverge;
    int id = blockDim.x * blockIdx.x + threadIdx.x;
    int tid = threadIdx.x;
    unsigned int x = d_seed[id];
    float r;
    const float updatevalue = 1.0f/ POPULATION_SIZE;
    s_prob[tid] = prob[id];
    s_prob_a_win[tid] =    s_prob[tid];
    s_prob_b_win[tid] =    s_prob[tid];
    if(tid ==0){
        s_Aconverge = 1;
        s_Aconverge = 1;
    }
    x = myRand(x);
    r = x / ((float) (UINT_MAX)+1.0);
    s_a[tid] = (r >= s_prob[tid]) ? 0 : 1;
    x = myRand(x);
    r = x / ((float) (UINT_MAX)+1.0);
    s_b[tid] = (r >= s_prob[tid]) ? 0 : 1;
    d_seed[id] = x;
    __syncthreads();
    s_aa[tid] = s_a[tid];
    s_bb[tid] = s_b[tid];
    for (int s = blockDim.x / 2; s > 0; s /= 2) {
        if (tid < s) {
            s_aa[tid] += s_aa[tid + s];
            s_bb[tid] += s_bb[tid + s];
        }
        __syncthreads();
    }
    if(s_a[tid] != s_b[tid]){
        if(s_a[tid] == 1) {
            s_prob_a_win[tid] += updatevalue;
            if(s_prob_a_win[tid] > 1){
                s_prob_a_win[tid] = 1;
            }
        }else{
            s_prob_a_win[tid] -= updatevalue;
            if(s_prob_a_win[tid] < 0){
                s_prob_a_win[tid] = 0;
            }
        }
        if(s_b[tid] == 1) {
            s_prob_b_win[tid] += updatevalue;
            if(s_prob_b_win[tid] > 1){
                s_prob_b_win[tid] = 1;
            }
        }else{
            s_prob_b_win[tid] -= updatevalue;
            if(s_prob_b_win[tid] < 0){
                s_prob_b_win[tid] = 0;
            }
        }    }
    d_probAwin[id] = s_prob_a_win[tid];
    d_probBwin[id] = s_prob_b_win[tid];
    __syncthreads();
    if(0< s_prob_a_win[tid] &&s_prob_a_win[tid]< 1){
            s_Aconverge = 0;
    }
    if(0< s_prob_b_win[tid] &&s_prob_b_win[tid]<1){
            s_Bconverge = 0;
    }
    __syncthreads();
    if(tid == 0){
        d_fitnessA[blockIdx.x]= s_aa[0];
        d_fitnessB[blockIdx.x]= s_bb[0];
        d_Aconverge[blockIdx.x] = s_Aconverge;
        d_Bconverge[blockIdx.x] = s_Bconverge;
    }   }
```

ประวัติการศึกษาและการทำงาน

ชื่อ	นางสาวนุกูล สถาพร
เกิดวันที่	27 พฤษภาคม 2525
สถานที่เกิด	อำเภอบ้านหมี่ จังหวัดลพบุรี
ประวิติการศึกษา	วท.บ. (วิทยาการคอมพิวเตอร์) มหาวิทยาลัยราชภัฏพระนครศรีอยุธยา (เกียรตินิยมอันดับสอง)
ตำแหน่งปัจจุบัน	ครู คศ. 1
สถานที่ทำงานปัจจุบัน	โรงเรียนเทพศาลาประชาสรรค์ เขตพื้นที่การศึกษามัธยมศึกษา เขต 42
ทุนการศึกษาที่ได้รับ	ได้รับทุนโครงการส่งเสริมการผลิตครูที่มีความสามารถพิเศษทางวิทยาศาสตร์และคณิตศาสตร์ (สควค.) จากสถาบันส่งเสริมการสอนวิทยาศาสตร์และเทคโนโลยี (สสวท.)