

## บทที่ 1

### บทนำ

#### ความเป็นมาและความสำคัญของปัญหา

ในการแก้ปัญหาอย่างการสลับกันในพีชคณิตด้วยขั้นตอนวิธีหาคำตอบที่เหมาะสมที่สุด ส่วนใหญ่นั้น ถึงแม้ว่าจะใช้การค้นหาที่น้อยกว่า  $n!$  แต่ก็ยังต้องใช้การค้นหาเข้าไปยังต้นไม้ หรือ กลุ่มของคำตอบจำนวนหนึ่ง ไม่เว้นแม้แต่ขั้นตอนวิธีเชิงพันธุกรรมที่สืบค้นคำตอบจากประชากรใน แต่ละรุ่น รวมถึงงานจากขั้นตอนอย่างการกลายพันธุ์ และการผสม สำหรับสร้างประชากรในรุ่นต่อไป ทำให้เมื่อนักวิจัยต้องการศึกษาเพื่อดูผลกระทบของพารามิเตอร์และหาพารามิเตอร์ที่เหมาะสมสำหรับปัญหานั้นๆ พวกเขาต้องทำการทดลองด้วยพารามิเตอร์ที่ต่างกันหลายชุดเพื่อจะได้พารามิเตอร์ที่เหมาะสมสำหรับปัญหานั้น มีการงานวิจัยของ Juan Li [1] ที่ศึกษาผลของ พารามิเตอร์ของขั้นตอนวิธีเชิงพันธุกรรมในปัญหา การวางแผนโครงข่ายการให้บริการสื่อสารส่วนบุคคลเพื่อหาค่าพารามิเตอร์ที่เหมาะสมสำหรับปัญหานี้ พบว่าคุณภาพของคำตอบดีขึ้นเมื่อเพิ่ม ขนาดของประชากร งานวิจัยของ Sofiene และ Jaleddine [2] ศึกษาผลของพารามิเตอร์ของ ขั้นตอนวิธีเชิงพันธุกรรม สำหรับปัญหาการลู่เข้าของสนามแม่เหล็กไฟฟ้า พบว่าการเพิ่มขนาดของ ประชากรทำให้เข้าใกล้คำตอบที่ดีกว่า แต่ประชากรที่ใหญ่ทำให้เวลาที่ใช้ในการลู่เข้าของ สนามแม่เหล็กไฟฟ้า มากขึ้นตามมา งานวิจัยของ Mohd Zakimi [3] ศึกษาผลของพารามิเตอร์ ของขั้นตอนวิธีเชิงพันธุกรรม ในปัญหา Multi objective Optimization (MMO) ซึ่งพบว่า พารามิเตอร์ที่เหมาะสมของปัญหานี้ขนาดประชากรอาจไม่ใช่ขนาดที่มากขึ้นเรื่อยๆ แต่เป็น ขนาดที่เหมาะสมค่าหนึ่ง แน่แน่นอนว่าหากขั้นตอนวิธีที่ต้องการศึกษาความเกี่ยวข้องของพารามิเตอร์ เหล่านี้มีความซับซ้อน การทดลองแต่ละครั้งต้องใช้ระยะเวลาสำหรับการคำนวณแบบปกติ ตัวประมวลผลกราฟิก (GPU) เป็นหนึ่งในตัวเลือกที่สามารถนำมาเพิ่มความเร็วได้ เนื่องจาก GPU มีราคาที่ถูกและพลังในการคำนวณที่มากกว่าเมื่อเปรียบเทียบกับ CPU[4] ซึ่งจะกล่าวถึงตัวอย่าง การทดลองที่นำ GPU ไปเพิ่มความเร็วให้กับขั้นตอนวิธีเชิงพันธุกรรมในหัวข้อต่อไป

ขั้นตอนวิธีอุปติการณ์ร่วมกัน (COIN) เป็นหนึ่งในขั้นตอนวิธีหาคำตอบที่ เหมาะสมที่สุด ที่แสดงให้เห็นความสามารถเมื่อเปรียบเทียบกับขั้นตอนวิธีที่นิยมหลายตัว [5] พารามิเตอร์ของ COIN เองก็เป็นที่น่าสนใจว่าการเปลี่ยนแปลงค่าพารามิเตอร์บางตัว อย่างเช่น จำนวนการเลือกประชากรของกลุ่มที่ดีและไม่ดีนั้นจำเป็นต้องเท่ากันหรือไม่ จำนวนการเลือก ประชากรที่เหมาะสมค่าหนึ่งอาจทำให้สามารถหาคำตอบได้ด้วยจำนวนรุ่นที่น้อยลงหรือไม่ แน่แน่นอนว่า

บทคัดย่อและแฟ้มข้อมูลฉบับเต็มของวิทยานิพนธ์ตั้งแต่ปีการศึกษา 2554 ที่ให้บริการในคลังปัญญาจุฬาฯ (CUIR)

เป็นแฟ้มข้อมูลของนิสิตเจ้าของวิทยานิพนธ์ที่ส่งผ่านทางบัณฑิตวิทยาลัย

การทดลองในแต่ละรอบนั้นหากว่าพารามิเตอร์บางตัวซึ่งมีผลทำให้ปริมาณข้อมูลในการคำนวณมีมากขึ้น อย่างจำนวนประชากร ก็จะต้องใช้เวลามากขึ้นในการคำนวณในแต่ละรอบ ดังนั้นเมื่อนำขั้นตอนวิธีปฏิบัติการร่วมกัน มาทำงานบน GPU แล้ว ด้วยการทำงานแบบขนานเราสามารถใช้เวลาในการทำการทดลองในแต่ละรอบน้อยลง ส่งผลให้เวลาที่ต้องใช้ในการศึกษาน้อยลงเช่นกัน กระบวนการวิเคราะห์ ขั้นตอนวิธีปฏิบัติการร่วมกัน เพื่อออกแบบการทำงานแบบขนาน ,การจัดแบ่งลักษณะของโครงสร้าง เทวด และ บล็อก เพื่อให้ทำงานบน GPU จะถูกกล่าวถึงต่อไป

### วัตถุประสงค์ของการวิจัย

นำขั้นตอนวิธีปฏิบัติการร่วมกัน ไปทำงานแบบขนานบนหน่วยประมวลผลกราฟิก เพื่อให้ ขั้นตอนวิธีปฏิบัติการร่วมกันทำงานเร็วกว่าการประมวลผลบนหน่วยประมวลผลกลางแบบปกติ

### ขอบเขตของการวิจัย

1. เปรียบเทียบการทำงานของ ขั้นตอนวิธีปฏิบัติการร่วมกัน ดั้งเดิมที่ทำงานบนหน่วยประมวลผลกลาง intel core i3 2.1 GHz กับ ขั้นตอนวิธีปฏิบัติการร่วมกัน ที่ถูกเขียนด้วย CUDA C เพื่อการประมวลผลแบบขนานบน GPU NVIDIA Geforce GT 540M ในหน่วยของเวลาไมโครเซคัน โดยทำการทดลอง 10 ครั้งและนำค่าเฉลี่ยมาเปรียบเทียบ
2. ใช้ปัญหาการเดินทางของนกชาย(TSP) เพื่อคำนวณค่าความเหมาะสมของประชากรแต่ละตัวในแต่ละรุ่น โดยใช้จำนวนประชากร 500 และ 1000 ตัวต่อรุ่น

### ประโยชน์ที่คาดว่าจะได้รับ

สามารถนำขั้นตอนวิธีปฏิบัติการร่วมกันในรูปแบบขนานนี้ไปทำงานวิจัยอื่นที่ต้องทำการประมวลผลซ้ำๆ ด้วยชุดข้อมูลที่มีปริมาณมากโดยใช้เวลาน้อยกว่าการทำงานปกติบนหน่วยประมวลผลกลาง เช่น การทดลองหาพารามิเตอร์ที่เหมาะสมสำหรับชุดข้อมูลหนึ่ง

### วิธีดำเนินการวิจัย

1. ศึกษาทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2. วิเคราะห์ลักษณะการทำงานของขั้นตอนวิธีคู่ปฏิบัติการร่วมกันเพื่อหา ลักษณะการทำงานแบบขนาน
3. ออกแบบโปรแกรมเพื่อให้ขั้นตอนวิธีคู่ปฏิบัติการร่วมกันทำงานแบบขนาน บนอุปกรณ์ประมวลผลกราฟิก
4. เขียนโปรแกรมทั้งแบบประมวลผลลำดับบนหน่วยประมวลผลกลางและ แบบขนานบนหน่วยประมวลผลกราฟิก
5. ทดลองประมวลผลทั้งสองแบบและจับเวลา
6. วิเคราะห์ผลการทดลอง
7. สรุปผลการวิจัยและเรียบเรียงวิทยานิพนธ์

### โครงสร้างของวิทยานิพนธ์

วิทยานิพนธ์ฉบับนี้ประกอบไปด้วย 5 บทหลัก คือ บทนำ เอกสารและทฤษฎีที่เกี่ยวข้อง วิธีดำเนินการวิจัย การทดลองและผลการทดลอง และสรุปงานวิจัยกับแนวทางพัฒนาต่อ

ในบทแรกจะกล่าวถึง ความเป็นมา ปัญหา วัตถุประสงค์ ขอบเขตของงานวิจัย ประโยชน์ที่จะได้รับ ขั้นตอนการทำวิจัย โครงสร้างของวิทยานิพนธ์ และผลงานที่ตีพิมพ์ จากวิทยานิพนธ์ บทที่สอง จะกล่าวถึง ทฤษฎีของวิธีคู่ปฏิบัติการร่วมกันและ ประวัติและการทำงานเบื้องต้นของ NVIDIA CUDA C บทที่สามกล่าวถึงวิธีดำเนินการวิจัย ซึ่งจะอธิบายขั้นตอนและหัวข้อที่ต้องศึกษารวมถึงการออกแบบโปรแกรมเพื่อประยุกต์ขั้นตอนวิธีคู่ปฏิบัติการร่วมกันบนหน่วยประมวลผลกราฟิก บทที่สี่ อธิบายวิธีการทดลองและแสดงผลการทดลอง ตลอดจนวิเคราะห์ผลการทดลอง และบทสุดท้ายจะสรุปงานวิจัยพร้อมแนะนำแนวทางเพื่อพัฒนาต่อยอดงานวิจัยนี้ให้ดีขึ้นต่อไป

### ผลงานที่ตีพิมพ์จากวิทยานิพนธ์

2012 Ninth International Joint Conference on Computer Science and Software Engineering (JCSSE) pp 126-130

## บทที่ 2

### เอกสารและงานวิจัยที่เกี่ยวข้อง

#### 2.1 ขั้นตอนวิธีปฏิบัติการร่วมกัน

เป็นหนึ่งในขั้นตอนวิธีเชิงพันธุกรรมที่มีแนวคิดสำคัญที่แตกต่างจาก ขั้นตอนวิธีเชิงพันธุกรรม แบบดั้งเดิม คือ การนำคำตอบที่ไม่ดีมาร่วมพิจารณาและเรียนรู้เพื่อหลีกเลี่ยงโอกาสของการได้มาซึ่งคำตอบที่ไม่ดีนั้นในรุ่นถัดไป จากการทดลอง [5] ที่แสดงให้เห็นว่าการเรียนรู้จากคำตอบที่ดีและไม่ดีพร้อมกัน ดีกว่าการเรียนรู้จากคำตอบที่ดี หรือไม่ดีเพียงอย่างเดียว และยังเปรียบเทียบกับขั้นตอนวิธีที่นิยมหลายตัว [5]

วิธีปฏิบัติการร่วมกัน แบ่งการทำงานออกเป็น 5 ขั้นตอนหลัก คือ กำหนดค่าเริ่มต้น, สร้างกลุ่มของประชากร, ประเมินค่าความเหมาะสมของประชากรแต่ละตัว, เลือกประชากรจำนวนหนึ่งออกมาเป็นกลุ่มที่ดีและไม่ดี, ปรับปรุงค่าความน่าจะเป็นร่วม เราสามารถพิจารณาการทำงานภายในของ 5 ขั้นตอนหลักได้ดังนี้

##### 2.1.1 กำหนดค่าเริ่มต้น

ตัวกำเนิด คือ ตารางขนาดเท่ากับ  $n \times n$  ซึ่ง  $n$  คือ ขนาดของคำตอบ ในตารางนี้บรรจุต้นไม้ที่ขึ้นต่อกัน (dependency tree) ซึ่งแต่ละแถวกำหนดให้เป็น ต้นไม้ที่ขึ้นต่อกัน หนึ่งต้น และแต่ละคอลัมน์คือความน่าจะเป็นร่วม  $h(X_i | X_j)$  ซึ่งเปรียบได้กับกิ่งของต้นไม้,  $X_i$  แทนแถวของตารางขณะที่  $X_j$  แสดงคอลัมน์ และกำหนดว่าเมื่อเกิดเหตุการณ์  $X_i$  และตามด้วย  $X_j$  ตำแหน่งที่  $X_{ij}$  คือความน่าจะเป็นร่วม  $h(X_i | X_j)$

การกำหนดค่าเริ่มต้นให้กับตารางจะแบ่งความน่าจะเป็นร่วม ไปยังทุกคอลัมน์เท่าๆกัน คือ  $\frac{1}{(n-1)}$  ยกเว้นตำแหน่งที่  $i$  เท่ากับ  $j$  จะได้ตารางที่เติมเต็มด้วยค่าความน่าจะเป็นร่วมที่เท่ากันทุกตำแหน่ง สำหรับตำแหน่ง  $X_{ij}$  ซึ่ง  $i$  เท่ากับ  $j$  จะได้ ความน่าจะเป็นร่วมเป็น 0

	A	B	C	D	E
A	0	0.25	0.25	0.25	0.25
B	0.25	0	0.25	0.25	0.25
C	0.25	0.25	0	0.25	0.25
D	0.25	0.25	0.25	0	0.25
E	0.25	0.25	0.25	0.25	0.25

(a)

	A	B	C	D	E
A	0	0.40	0.20	0.20	0.20
B	0.25	0	0.25	0.25	0.25
C	0.25	0.25	0	0.25	0.25
D	0.25	0.25	0.25	0	0.25
E	0.25	0.25	0.25	0.25	0.25

(b)

	A	B	C	D	E
A	0	0.10	0.30	0.30	0.30
B	0.25	0	0.25	0.25	0.25
C	0.25	0.25	0	0.25	0.25
D	0.25	0.25	0.25	0	0.25
E	0.25	0.25	0.25	0.25	0.25

(c)

รูปที่ 2-1 ตัวกำเนิด ขนาด 5x5 หลังจากผ่านการกำหนดค่าเริ่มต้น ทุกช่องจะถูกเติมด้วยความ

น่าจะเป็นร่วมเท่ากับ  $\frac{1}{(n-1)}$  ยกเว้นช่องที่  $i$  เท่ากับ  $j$  ความน่าจะเป็นร่วมเท่ากับ 0

### 2.1.2 การสร้างกลุ่มของประชากร

ประชากรแต่ละตัวคือผลจากการเดินทางใน dependency tree ในแต่ละแถวของตาราง ตัวกำเนิด โดย เริ่มจากตำแหน่ง  $X_i$  ใดๆ ตามความน่าจะเป็นที่เหมาะสม เช่น หากว่าไม่มีข้อสังเกตที่ชัดเจนว่าตำแหน่งใดน่าจะถูกเลือกเป็นพิเศษแล้ว อาจให้ความน่าจะเป็นของการเลือกตำแหน่งเริ่มต้นเท่ากันที่  $\frac{1}{n}$  ตำแหน่งนี้จะกลายเป็นรากของต้นไม้ เมื่อได้รากของต้นไม้มาแล้ว สมาชิกของคำตอบที่เหลือได้มาจากการค้นเข้าไปยังต้นไม้ด้วยวิธีลงลึกก่อน สำหรับการเลือกแต่ละ  $X_j$  จะเลือกจากความน่าจะเป็นร่วม  $h(X_i | X_j)$  ของแต่ละกิ่ง

### 2.1.3 การประเมินค่าความเหมาะสมของประชากร

สิ่งที่ได้มาจากขั้นตอนการสร้างกลุ่มประชากรคือคำตอบที่ประกอบไปด้วยกลุ่มของคำตอบที่ยังไม่สามารถบอกได้ถึงความดีหรือไม่ดีของประชากรแต่ละตัว ในขั้นตอนนี้จะทำการประเมินค่าความเหมาะสมจากฟังก์ชันความเหมาะสม ซึ่งแตกต่างกันไปในแต่ละปัญหา เช่น ปัญหา TSP ค่าความเหมาะสม คือ ค่าระยะทางรวมของทุกเมืองบนเส้นทางที่สร้างขึ้นมา

### 2.1.4 เลือกประชากรเป็นกลุ่มที่ดีและไม่ดี

เมื่อได้ค่าความเหมาะสมของประชากรแต่ละตัวแล้ว เราสามารถแบ่งเป็นกลุ่มที่ดีและไม่ดีได้จากค่าความเหมาะสม วิธีในการแบ่งที่มี 2 วิธี [5] วิธีแรกคือการแบ่งแบบเท่ากัน (uniform) และแบบปรับตัว ในงานวิจัยนี้เราใช้วิธีแบ่งแบบเท่ากันคือการเลือกจากประชากรที่มีค่าความเหมาะสมมากที่สุดลงไปหาหน่อยเป็นจำนวน  $c$  เปอร์เซนต์ เป็นกลุ่มที่ดี และเลือกจากประชากรที่มีค่าความเหมาะสมน้อยที่สุดไปหาหมาก  $c$  เปอร์เซนต์เท่ากัน เพื่อเป็นตัวแทนของกลุ่มที่ไม่ดี

### 2.1.5 การปรับปรุงค่าความน่าจะเป็นร่วม

กลุ่มของคำตอบที่ดีและไม่ดีที่ได้จากขั้นตอนการเลือกประชากรจะกลายเป็นตัวแทนของกลุ่มคำตอบทั้งหมด เมื่อพิจารณาคู่ลำดับ  $X_i, X_j$  ของแต่ละตัวแทน เขียนได้ว่า  $X_{i,j}$  มีความน่าจะเป็นร่วม  $h(X_i | X_j)$  เมื่อนับคู่ลำดับนี้ในกลุ่มประชากรที่ดี มีจำนวน  $r_{i,j}$  สมการการให้รางวัลคือ

$$X_{i,j}^{t+1} = X_{i,j}^t + \frac{k}{(n-1)} r_{i,j} - \frac{k}{(n-1)^2} \left( \sum_{m=1}^n r_{i,m} \right) \quad , m \neq j \quad (1)$$

$r_{i,m}$  คือช่องอื่นที่ไม่ใช่  $j$

นิพจน์  $\frac{k}{(n-1)^2} \left( \sum_{m=1}^n r_{i,m} \right)$  คือการลดค่าช่องอื่นๆลง

เช่นเดียวกัน สมการการลงโทษ เมื่อ  $p_{i,j}$  คือจำนวน  $X_i, X_j$  ที่พบในกลุ่มที่ไม่ดี คือ

$$X_{i,j}^{t+1} = X_{i,j}^t - \frac{k}{(n-1)} p_{i,j} + \frac{k}{(n-1)^2} \left( \sum_{m=1}^n p_{i,m} \right) \quad , m \neq j \quad (2)$$

เมื่อรวมสมการ (1) และ (2) เข้าไว้ด้วยกัน

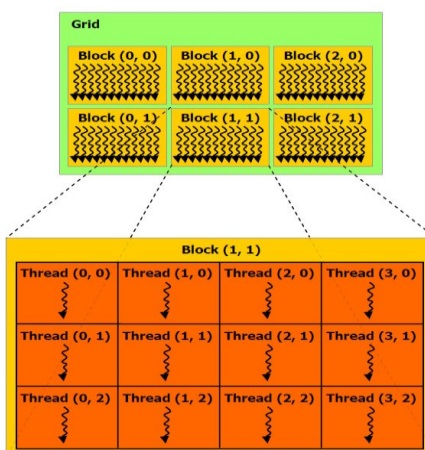
$$X_{i,j}^{t+1} = X_{i,j}^t + \frac{k}{(n-1)} (r_{i,j} - p_{i,j}) - \frac{k}{(n-1)^2} \sum_{m=1}^n (r_{i,m} - p_{i,m}) \quad , m \neq j \quad (3)$$

มีกฎบางข้อเพิ่มเติมในการ สำหรับการปรับปรุงค่าความน่าจะเป็นร่วมนี้ เนื่องด้วยการปรับปรุงค่าความน่าจะเป็นร่วมนั้นคือการเพิ่ม หรือ ลด ด้วยอัตราส่วนคงที่ แต่ความน่าจะเป็นต้องไม่เป็นค่าติดลบ มากกว่านั้นถ้าความน่าจะเป็นร่วมได้ถูกเพิ่มค่าไปถึงจุดๆหนึ่งแล้วต้องไม่เพิ่มค่าขึ้นอีก เพื่อให้เหลือความน่าจะเป็นในการเลือกสมาชิกตัวอื่นด้วย เพื่อให้สามารถสร้างคำตอบที่หลากหลาย

## 2.2 หน่วยประมวลผลกราฟิกและ CUDA C

นับตั้งแต่ชุดพัฒนา CUDA (Compute Unified Device Architecture) เปิดตัวเมื่อเดือนพฤศจิกายน ปี 2006 ด้วยสถาปัตยกรรมที่รวม CPU และ อุปกรณ์ประมวลผล GPU มาร่วมกันประมวลผล ใช้ฟังก์ชันที่สามารถเรียกได้จาก CPU และทำงานที่อุปกรณ์ประมวลผล GPU จะถูกเรียกพร้อมกันเป็นจำนวนตามที่กำหนดด้วยตัวเลขของ บล็อก และ เทรด ซึ่งจำนวนที่ถูกประมวลผลจริงในช่วงเวลาหนึ่งจะเป็นไปตามจำนวนแกนของ GPU ที่มี

เทรด คือ หน่วยย่อยที่สุด ซึ่งกลุ่มของ เทรด จะอยู่ภายใต้ บล็อก หนึ่งและทำงานภายในแกนประมวลผลเดียวกัน ส่วน บล็อก กลุ่มของ บล็อก เรียกว่า กริด แต่ละบล็อกสามารถถูกประมวลผลแยกกันไปตามแกนที่ว่างและพร้อมประมวลผลได้ ดังนั้นงานที่ถูกประมวลผลในแต่ละบล็อก นั้นควรจะเป็นงานที่ไม่เกี่ยวเนื่องกัน ภาพที่ 2 แสดงให้เห็นถึงภาพของการแบ่ง บล็อก และ เทรด โดยในภาพ บล็อก ถูกแบ่งออกเป็น 2 มิติ และในแต่ละบล็อกนั้นบรรจุ เทรด ที่สามารถอ้างถึงด้วยตัวเลข 2 มิติเช่นกัน



รูปที่ 2-2 การแบ่ง บล็อกและเทรด โดยกลุ่มของบล็อกที่อ้างถึงโดยตัวเลข 2 มิติอยู่ภายใต้กริดเดียวกัน และในแต่ละ บล็อก บรรจุ เทรด ที่อ้างถึงด้วยเลข 2 มิติ

### สถาปัตยกรรม SIMT (Single-Instruction, Multiple-Thread)

ชุดพัฒนา CUDA นั้นได้ถูกสร้างขึ้นจากชุดของสตรีมมิ่งมัลติโพรเซสเซอร์ที่ทำงานพร้อมกัน เมื่อชุดคำสั่งจาก CPU เรียกไปยัง GPU จำนวนของเทรดต่อบล็อกที่กำหนดโดยโปรแกรมเมอร์ เทรดจะถูกแบ่งออกเป็นกลุ่มตามจำนวนวาร์ป ( วาร์ป คือ กลุ่มของเทรดที่ทำงานพร้อมกันหนึ่งชุด จำนวนของเทรดที่อยู่ในวาร์ปนี้แตกต่างกันไปตามรุ่นของตัวประมวลผลกราฟิก ตัวอย่างเช่นใน รุ่นของความสามารถการคำนวณ 2.x หนึ่งวาร์ปจะมี 48 เทรด และในรุ่นที่ต่ำกว่าจะมี 32 เทรด) จากนั้นมัลติโพรเซสเซอร์จะสร้าง จัดการวางแผนเส้นทาง และสั่งให้กลุ่มของเทรด

ทำงาน ในเทรด์แต่ละตัวจะมีตัวนับโปรแกรมและรีจิสเตอร์เก็บสถานะของตัวเองดังนั้นเทรด์แต่ละตัวจะสามารถทำงานแตกต่างจากเทรด์อื่นที่อยู่ในการ์ปเดียวกันได้

การทำงานของแต่ละเทรด์ในการ์ปนั้นมีความเกี่ยวข้องทางด้านประสิทธิภาพการทำงาน เนื่องจากการทำงานของเทรด์ทุกตัวในการ์ปจะเริ่มต้นจากตำแหน่งโปรแกรมเดียวกันและทำงานตามชุดคำสั่งที่เหมือนกันไปเรื่อยๆ แต่ถ้ามีเทรด์บางตัวที่ต้องทำงานแตกต่างจากเทรด์ตัวอื่นเนื่องจากเงื่อนไขบางอย่างเฉพาะตัว มัลติโพรเซสเซอร์จะให้เทรด์ตัวอื่นหยุดรอ และปล่อยให้เทรด์ตัวที่ทำงานแตกต่างกันนั้นได้ทำงานของตัวเองเสร็จสิ้น และเมื่อชุดคำสั่งของเทรด์ตัวนั้นเป็นชุดเดียวกับเทรด์ตัวอื่น มัลติโพรเซสเซอร์จะสั่งให้เทรด์ทุกตัวทำงานพร้อมกันอีกครั้ง นี่คือความหมายของ Single-Instruction , Multiple-Thread ที่หนึ่งคำสั่งจะสั่งให้เทรด์ทำงานพร้อมกันหลายตัวดังที่อธิบาย หากว่าการทำงานของโปรแกรมมีเงื่อนไขที่ทำให้บางเทรด์ในการ์ปต้องทำงานแตกต่างจากเทรด์ตัวอื่น ประสิทธิภาพการทำงานจะด้อยลงเนื่องจากต้องมีการหยุดรอกัน จากสถาปัตยกรรม SIMT นั้น โปรแกรมเมอร์สามารถเขียนโปรแกรมในระดับของเทรด์เพื่อให้เทรด์ทำงานแบบขนานให้ได้ประสิทธิภาพการทำงานสูงสุดได้ แต่สำหรับการนำตัวประมวลผลไปประยุกต์ใช้กับปัญหาที่เน้นทางด้านความถูกต้องมากกว่า การทำงานภายในของ SIMT นั้นอาจไม่ต้องนำมาพิจารณา

จำนวนของรีจิสเตอร์และความจำรวมเป็นทรัพยากรที่จำกัด ดังนั้นจำนวนของบล็อกและเวิร์ปที่ถูกประมวลผลโดยมัลติโพรเซสเซอร์ ณ เวลาหนึ่งเองก็ถูกจำกัดโดยจำนวนของรีจิสเตอร์และความจำรวมที่มีอยู่บนมัลติโพรเซสเซอร์ ถ้าทรัพยากรเหล่านี้มีไม่เพียงพอที่จะสำหรับเทรด์ทั้งหมดที่อยู่ในหนึ่งบล็อก การทำงานนั้นจะล้มเหลว สมการ (4) ,(5) ,(6) แสดงการคำนวณจำนวนเวิร์ป จำนวนรีจิสเตอร์ และจำนวนความจำรวมที่ใช้ในหนึ่งบล็อก

$$W_{block} = \text{ceil}\left(\frac{T}{W_{size}}, 1\right) \quad (4)$$

- $T$  คือ จำนวนเทรด์ต่อบล็อก
- $W_{size}$  คือ ขนาดของเวิร์ป ซึ่งในรุ่นความสามารถการคำนวณ 2.x เท่ากับ 48
- $\text{ceil}(x, y)$  คือ ทำให้  $x$  มีค่าใกล้เคียงจำนวนเท่าของ  $y$  มากที่สุด ในสมการคือการทำให้  $\frac{T}{W_{size}}$  มีค่าเป็นจำนวนเท่าของ 1

จากสมการที่ (4) จะได้จำนวนเวิร์ปในหนึ่งบล็อกมาซึ่งจะนำไปหาจำนวนรีจิสเตอร์และจำนวนความจำรวมต่อไปในสมการที่ (5) และ (6) ตามลำดับ



$$R_{block} = \text{ceil}(R_k \times W_{size}, G_T) \times W_{block} \quad (5)$$

- $G_T$  คือ จำนวนเทรตน้อยสุดที่สามารถจองได้ เท่ากับ 64 สำหรับรุ่นของความสามารถการคำนวณ
- $R_k$  คือ จำนวนวีจีเอสเตอร์ที่ถูกต้องใช้โดยเคอร์เนล

จากสมการที่ (5) ถ้าจำนวนวีจีเอสเตอร์ที่ต้องใช้ในหนึ่งบล็อกมากกว่าจำนวนวีจีเอสเตอร์ที่มีอยู่ในมัดติโพรเซสเซอร์ จำนวนของวาร์ปต่อบล็อกจะถูกลดลงในตอนคำนวณจริง เพื่อให้โปรแกรมสามารถทำงานได้

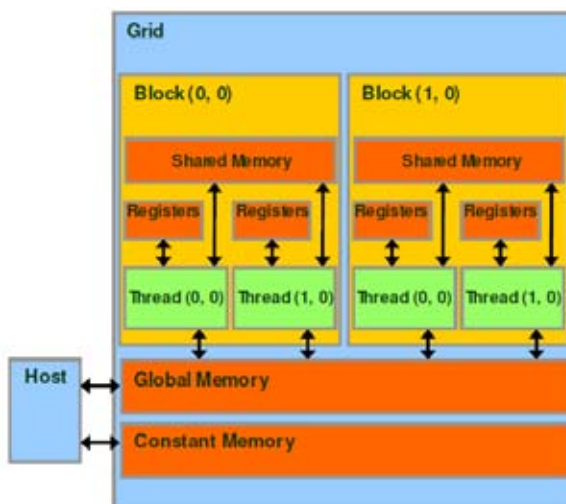
$$S_{block} = \text{ceil}(S_k, G_S) \quad (6)$$

- $S_k$  คือ จำนวนของความจำร่วมที่ถูกต้องใช้โดยเคอร์เนลมีหน่วยเป็นไบท์
- $G_S$  คือ จำนวนความจำร่วมที่น้อยที่สุดที่สามารถจองได้ มีขนาด 128 ไบท์สำหรับรุ่นของความสามารถการคำนวณ 2.x

สังเกตได้ว่าหากจำนวนเทรตต่อบล็อกที่โปรแกรมเมอร์กำหนดให้ในกับเคอร์เนลนั้นไม่เป็นจำนวนเท่าของจำนวนวาร์ปสูงสุดตามรุ่นของความสามารถการคำนวณของตัวประมวลผลกราฟิก เช่น รุ่นของความสามารถการคำนวณ 2.x มีจำนวนวาร์ปสูงสุด 48 จะทำให้เหลือเศษและต้องปัดเป็นจำนวนวาร์ปต่อบล็อกขึ้นอีกหนึ่งค่า ตัวอย่างเช่น จำนวนเทรตต่อบล็อกคือ 49 จากสมการ (4) เราจะได้จำนวนวาร์ปต่อบล็อกเป็น 2 ทำให้การจองวีจีเอสเตอร์และความจำร่วมเพิ่มขึ้นอย่างมาก ซึ่งจะมีผลให้ความสามารถในการคำนวณของมัดติโพรเซสเซอร์ลดลง

*ชนิดของความจำในอุปกรณ์ประมวลผล*

ในอุปกรณ์ประมวลผลกราฟิกนั้นมีชนิดของความจำอยู่จำนวนหนึ่ง ผู้เขียนโปรแกรมสามารถเลือกใช้เพื่อเพิ่มอัตราส่วน CGMA (Compute to global memory access) ซึ่งเป็นจำนวนตัวเลขทศนิยมที่บอกถึงประสิทธิภาพของการส่งผ่านข้อมูลในความจำทั่วไป ( global memory )



รูปที่ 2-3 ความจำในอุปกรณ์ประมวลผลกราฟิก GeForce 8800 GTX

จากรูปที่ 2-3 ด้านล่างสุดของรูปจะเห็นได้ว่าโฮสจะอ่านและเขียนข้อมูลไปที่ความจำทั่วไป (Global Memory) หรือ ความจำคงที่ (Constant Memory) โดยการเรียกจาก API ฟังก์ชัน ความจำแบบคงที่ จะถูกอ่านอย่างเดียวซึ่งจะถ่ายโอนข้อมูลได้เร็วกว่าในบางกรณีเนื่องจากมีการแคชข้อมูลและสามารถถูกเข้าถึงแบบขนานได้ดีกว่าความจำทั่วไป

เหนือเทรดขึ้นไปเป็นรีจิสเตอร์และความจำร่วม ตัวแปรที่อยู่ในหน่วยความจำนี้จะสามารถอ่านและเขียนแบบขนานได้อย่างรวดเร็ว รีจิสเตอร์จะถูกจองไว้ให้กับเทรดเป็นรายตัว เทรดแต่ละตัวสามารถเข้าถึงรีจิสเตอร์ของตัวเองได้เท่านั้น โดยปกติแล้วในการประมวลผลเคอร์เนลในแต่ละครั้ง รีจิสเตอร์จะถูกใช้เก็บตัวแปรที่มีการใช้บ่อยครั้งและเข้าถึงได้จากเทรดที่เป็นเจ้าของเท่านั้น

ความจำร่วมจะถูกจองไว้สำหรับกลุ่มของเทรด เทรดทุกตัวที่อยู่ในกลุ่มเดียวกันจะสามารถเข้าถึงความจำร่วมของกลุ่มตัวเองได้เท่านั้น ซึ่งความจำร่วมนี้มีประโยชน์ในการแลกเปลี่ยนข้อมูลระหว่างเทรดในกลุ่มเดียวกันได้

#### การใช้งานความจำให้เกิดประสิทธิภาพสูงสุด

ในภาพรวมของการใช้งานความจำให้เกิดประสิทธิภาพสูงสุดนั้นคือการพยายามลดการส่งผ่านข้อมูลในเส้นทางที่มีความกว้างของช่องทางในการรับส่งข้อมูลต่ำ (low bandwidth) เช่น การส่งผ่านข้อมูลระหว่างอุปกรณ์ประมวลผลกราฟิกกับโฮส ซึ่งมีความกว้างของช่องทางในการรับส่งข้อมูลต่ำกว่าช่องทางในการส่งผ่านข้อมูลระหว่างความจำทั่วไปภายในอุปกรณ์ประมวลผลกราฟิกเอง

และนอกจากการลดการส่งผ่านข้อมูลระหว่างอุปกรณ์และโฮสแล้ว การใช้ความจำอย่างมีประสิทธิภาพยังสามารถทำได้โดยการเพิ่มการส่งผ่านข้อมูลระหว่างความจำทั่วไปและความจำ

ภายในหน่วยประมวลผลกราฟิก คือ ความจำร่วม ( Shared memory ) ซึ่งเป็นหน่วยความจำที่อยู่ติดกับหน่วยประมวลผลและมีความเร็วในการส่งผ่านข้อมูลที่รวดเร็ว เราสามารถพิจารณาการเข้าถึงของความจำแต่ละชนิดในเชิงลึกเพื่อให้เข้าใจการทำงานและสามารถออกแบบการอ่านเขียนหน่วยความจำแต่ละชนิดเพื่อให้เกิดประสิทธิภาพสูงสุดได้ดังนี้

#### ความจำทั่วไป ( Global Memory )

ดังกล่าวข้างต้นว่าการส่งข้อมูลที่มีประสิทธิภาพคือการส่งผ่านที่เป็นกลุ่มก้อน (Coalesce) ถ้าขนาดของข้อมูลที่ส่งผ่านแต่ละครั้งมีขนาดได้ใกล้เคียงกับขนาดสูงสุดของอัตราการส่งผ่านข้อมูลสูงสุดได้ ยิ่งจะทำให้เกิดประสิทธิภาพมากขึ้นเท่านั้น ในการเข้าถึงความจำสามารถส่งข้อมูลด้วยขนาด 32, 64 หรือ 128 ไบท์ โดยปกติแล้วความจำจะเรียงต่อกันเป็นจำนวนเท่าของ 32, 64, 128 ไบท์

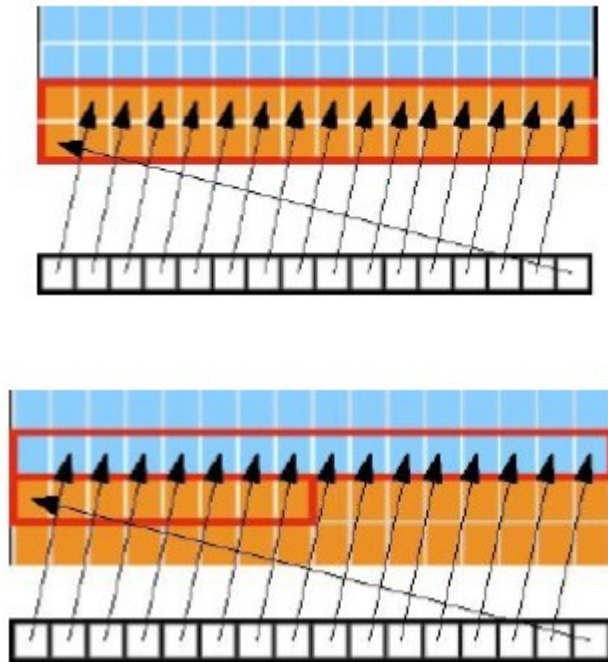
เมื่ออาร์ปหรือกลุ่มของเทรตได้ประมวลผลคำสั่งหนึ่งคำสั่งเพื่อเข้าถึงความจำทั่วไป การส่งผ่านข้อมูลแบบกลุ่มก้อนจะเกิดขึ้นได้หรือไม่ขึ้นอยู่กับจำนวนไบท์ที่เทรตแต่ละตัวในอาร์ปอ่านหรือเขียนข้อมูลว่าเป็นจำนวนเท่าของ 32, 64 หรือ 128 ไบท์หรือไม่ และอยู่กระจายกันในความจำทั่วไปมากน้อยแค่ไหนยิ่งข้อมูลถูกกระจายออกเป็นส่วนๆ มาก ทำให้จำนวนครั้งในการร้องขอการอ่านเขียนข้อมูลจะถูกแบ่งออกเป็นหลายครั้งมากขึ้น ทำให้การส่งผ่านข้อมูลไม่มีประสิทธิภาพ เช่น ถ้าขนาดของการส่งผ่านเป็น 32 ไบท์ และแต่ละเทรตถึงข้อมูลครั้งละ 4 ไบท์ ดังนั้นปริมาณงานที่ทำได้คือ คือ 8

จำนวนครั้งในการส่งผ่านข้อมูลและปริมาณงานที่ทำได้ขึ้นอยู่กับรุ่นของความสามารถในการคำนวณของอุปกรณ์ประมวลผลกราฟิก สำหรับรุ่นของความสามารถในการคำนวณ 1.0 และ 1.1 นั้น หากว่าข้อมูลที่แต่ละเทรตต้องการนั้นอยู่กันอย่างกระจัดกระจายในหน่วยความจำจะทำให้การส่งผ่านข้อมูลแบบกลุ่มก้อนนั้นเกิดขึ้นได้ยาก แต่สำหรับรุ่นของความสามารถในการประมวลผล 2.x จะมีการเก็บข้อมูลที่ได้อ่านหรือเขียนมาก่อนหน้านี้หรือ cache ทำให้ลดผลกระทบจากการกระจายตัวของข้อมูลในหน่วยความจำและเพิ่มปริมาณงานที่ทำได้

เพื่อเพิ่มปริมาณงานที่ทำได้ของความจำทั่วไป จะต้องเพิ่มการส่งผ่านข้อมูลแบบกลุ่มก้อนให้มากที่สุดโดยวิธีการดังนี้

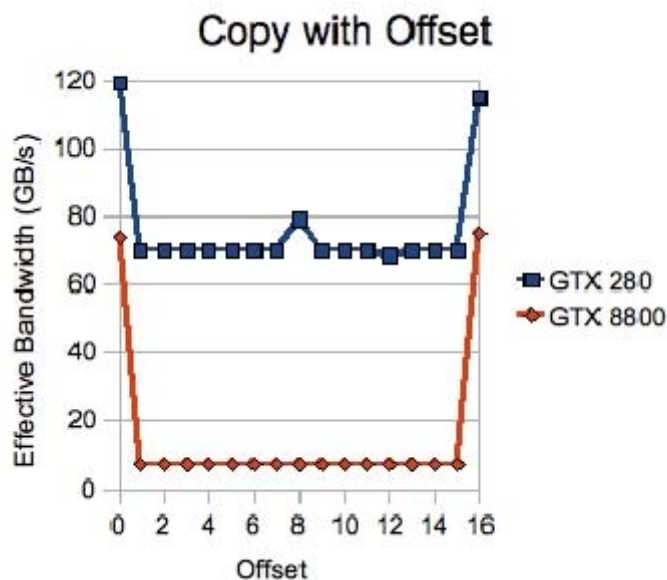
- ปรับแต่งการเข้าถึงข้อมูลให้มีรูปแบบที่แน่นอนให้มากที่สุด เช่น ในกรณีรุ่นความสามารถของการประมวลผล 1.2 ขึ้นไป การเรียกอ่านข้อมูลของแต่ละเทรต ควร

มีขนาด 32 ไบต์สำหรับการเข้าถึงข้อมูลที่ละ 8 บิต ,64 ไบต์สำหรับ 16 บิต และ 128 ไบต์ สำหรับ 32,64,128 บิต



รูปที่ 2-4 รูปแบบการเข้าถึงตำแหน่งข้อมูลแบบเรียงตัวที่อยู่ภายใน 128 ไบต์เดียวกันและไม่อยู่ภายใน 128 ไบต์เดียวกัน

จากรูปที่ 2-4 บนจะเห็นได้ว่าการส่งผ่านข้อมูลจะเกิดขึ้นครั้งเดียวและมีขนาด 128 ไบต์ ถ้าตำแหน่งข้อมูลที่ร้องขออยู่ภายใน 128 ไบต์เดียวกัน และรูปล่างถ้าตำแหน่งข้อมูลที่ร้องไม่อยู่ภายใน 128 ไบต์ เดียวกัน จะเกิดการร้องขอข้อมูลสองครั้งแทนที่จะเป็นครั้งเดียวเหมือนกรณีรูปบน



รูปที่ 2-5 ประสิทธิภาพของการคัดลอกข้อมูลตามระยะห่างจากตำแหน่งเริ่มต้น GTX 280 และ GTX 8800

จากรูปที่ 2-5 จะเห็นได้ว่าที่ตำแหน่งที่ระยะห่างเป็น 0 หรือ ที่ 16 ไบท์ การส่งผ่านข้อมูลจะมีประสิทธิภาพที่สุดเนื่องจากการส่งผ่านครั้งเดียวทำให้อัตราการส่งผ่านข้อมูลสูงถึง 74 GBps ขณะที่ระยะห่างอื่นซึ่งไม่เป็นจำนวนเท่าของ 16 ไบท์อัตราการส่งผ่านข้อมูลลดลงเหลือเพียง 7 GBps หรือ ประสิทธิภาพลดลงประมาณ 8 เท่า

จากกราฟที่แสดงให้เราเห็นได้ว่าถึงแม้จะมีรูปแบบในการเข้าถึงข้อมูลแล้วแต่ถ้าตำแหน่งการเข้าถึงข้อมูลไม่เป็นจำนวนเท่าที่เหมาะสมแล้ว อัตราการส่งผ่านข้อมูลจะลดลงอย่างมาก ในรุ่นความสามารถการคำนวณ 2.x นั้นปัญหานี้จะน้อยลงเนื่องจากการแคชข้อมูลที่เคยอ่านมาก่อนไว้ ทำให้ต้องมีการส่งผ่านข้อมูลเฉพาะที่ไม่มีอยู่ในแคช

- ใช้ชนิดของตัวแปรที่สามารถกำหนดการเรียงตัวในหน่วยความจำได้

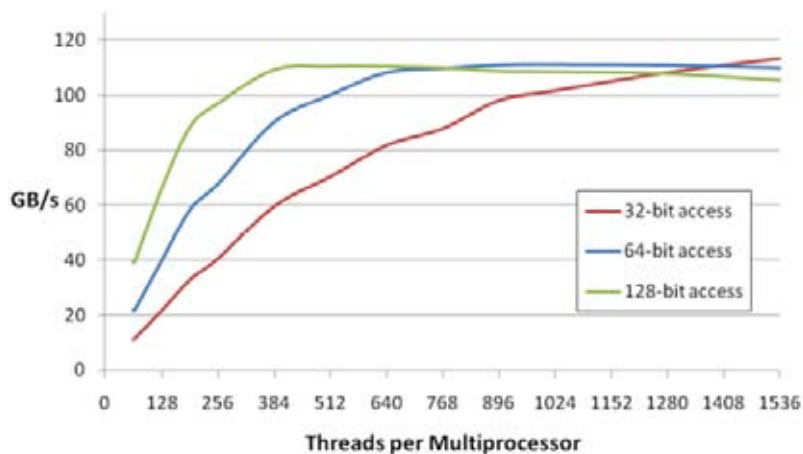
คำสั่งในการเข้าถึงข้อมูลทั้งอ่านและเขียนข้อมูลบนความจำทั่วไปนั้นสนับสนุนการร้องขอข้อมูลที่ขนาด 4, 8, 16 ไบท์ ถ้าข้อมูลของตัวแปรตัวหนึ่งมีตำแหน่งบนหน่วยความจำไม่เรียงกัน คำสั่งในการเข้าถึงจะถูกแยกออกเป็นหลายคำสั่งแทนที่จะเป็นคำสั่งเดียว

ดังนั้นเพื่อให้แน่ใจว่าตัวแปรเรียงตัวในหน่วยความจำเป็นขนาด 2, 4, 8 หรือ 16 ไบต์ตามที่ต้องการ การใช้ตัวแปรที่ให้มากับชุดพัฒนา เช่น `__align__(8)`, `__align__(16)` ตัวอย่างการใช้ คือ

```
struct __align__(8) {
    float x;
    float y;
};
```

- เติมข้อมูลส่วนที่ขาดสำหรับการเข้าถึงในบางกรณี

ในกรณีที่ความจำทั่วไปถูกใช้เพื่อเก็บข้อมูลแบบอะเรย์สองมิติ และความกว้างของอะเรย์ไม่เป็นจำนวนเท่าของขนาดวาร์ป การเข้าถึงในการอ่านเขียนข้อมูลจำไม่มีประสิทธิภาพ แต่ยังมีวิธีที่เข้าถึงหน่วยความจำได้อย่างมีประสิทธิภาพโดยการใช้ฟังก์ชัน `cudaMallocPitch()` และ `cuMemAllocPitch()` และ ฟังก์ชันสำหรับความจำที่เกี่ยวข้อง ผู้เขียนโปรแกรมก็สามารถจองหน่วยความจำด้วยวิธีนี้เพื่อเพิ่มความสามารถในการอ่านเขียนหน่วยความจำให้มีประสิทธิภาพได้



รูปที่ 2-6 การเข้าถึงหน่วยความจำทั่วไปด้วยขนาด 4, 8 และ 16 ไบต์

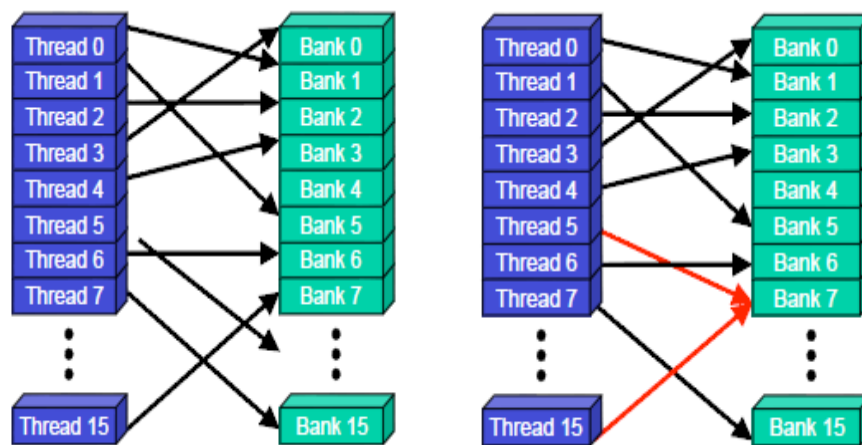
จากรูปที่ 2-6 แสดงการเข้าถึงความจำทั่วไปด้วยขนาด 4, 8, และ 16 ไบต์ การเข้าถึงข้อมูลที่อยู่ในความจำทั่วไปด้วยขนาดไบต์ที่มากขึ้นจะทำให้ปริมาณข้อมูลที่ส่งผ่านได้ต่อวินาที

มากขึ้นตามแกนนอน ส่วนแกนนอนแสดงจำนวนเทดที่ร้องขอข้อมูลซึ่งจำนวนเทดที่มากขึ้นจะทำให้ปริมาณการส่งผ่านข้อมูลมีมากขึ้นเช่นกัน

ความจำร่วม ( Shared Memory )

ความจำร่วมเป็นหน่วยความจำซึ่งอยู่บนชิพ หรืออยู่ใกล้กับตัวประมวลผลดังนั้นมันจึงทำงานได้เร็วกว่าความจำภายใน ( local memory ) และความจำทั่วไป ( global memory )  
ความเร็ว

เพื่อให้สามารถใช้ความเร็วในการส่งผ่านข้อมูลสูงสุดและการเข้าถึงแบบพร้อมกัน ความจำร่วมจะถูกแบ่งออกเป็นโมดูลขนาดเท่าๆกันเรียกว่าแบงค์ ซึ่งทำให้สามารถเข้าถึงได้พร้อมๆกันได้ ดังนั้นถ้ามีการอ่านหรือเขียนตำแหน่งความจำที่ n ซึ่งอยู่ต่างแบงค์กันจะสามารถเข้าถึงได้พร้อมๆกันทำให้อัตราการส่งผ่านข้อมูลสูง



รูปที่ 2-7 การเข้าถึงแบงค์ของความจำร่วม ด้านซ้ายแสดงการเข้าถึงแบงค์ปกติ รูปด้านขวาแสดงการเข้าถึงแบงค์ที่ขัดแย้ง

อย่างไรก็ตาม ถ้ามีการอ่านหรือเขียนข้อมูลบนแบงค์เดียวกัน จากหลายเทดพร้อมๆกัน เรียกเหตุการณ์นี้ว่าการเข้าถึงแบงค์ที่ขัดแย้ง ดังรูปที่ 2-7 เมื่อมีการเข้าถึงแบงค์เดียวกัน การร้องขอจะถูกแบ่งออกเป็นหลายคำสั่ง เพื่อรอคิวกันในการใช้ความจำ ดังนั้นปัจจัยที่มีผลในการลดอัตราการส่งผ่านข้อมูลในกรณีความจำร่วมคือจำนวนของการร้องขอที่ถูกแบ่งออกมา ซ้อยกเว้นอย่างเดียว คือ ถ้าเทดที่ร้องขอไปยังที่แบงค์เดียวกันเป็นเทดที่อยู่ในวาร์ปเดียวกัน จะไม่เกิดการขัดแย้งแต่จะมีการกระจายข้อมูลตำแหน่งที่ร้องขอไปยังทุกเทดพร้อมกันแทน

เพื่อลดเหตุการณ์ของการเข้าถึงที่ขัดแย้งของแบงค์ ต้องเข้าใจลักษณะการเข้าถึงแบงค์จากเทรดในวาร์ป

แบงค์ของความจำรวมจะถูกแบ่งออกให้มีขนาด 32 บิตและแต่ละแบงค์จะมีอัตราการส่งผ่านข้อมูล 32 บิตต่อหนึ่งรอบนาฬิกาหน่วยประมวลผล สำหรับรุ่นความสามารถในการประมวลผล 1. ที่ขนาดวาร์ป 32 เทรดและจำนวนของแบงค์เป็น 16 การร้องขอในการใช้ความจำจะถูกแบ่งออกเป็นสองคำสั่ง อันแรกเป็นของครึ่งแรกของวาร์ปและอีกคำสั่งสำหรับครึ่งหลังของวาร์ปและจะไม่เกิดการขัดแย้งในกรณีนี้ที่แต่ละครึ่งวาร์ปเข้าถึงความจำที่ตำแหน่งเดียวกัน

สำหรับรุ่นความสามารถการประมวลผล 2.x ขนาดของวาร์ปเป็น 32 เหมือนกันแต่ขนาดของแบงค์เป็น 32 บิต ดังนั้นคำสั่งในการเข้าถึงแบงค์จะไม่ถูกแบ่งออกเป็นสองคำสั่งเหมือนในรุ่นความสามารถการประมวลผล 1.x ดังนั้นกรณีที่เกิดการขัดแย้งได้จะสามารถมาจากทั้งสองครึ่งวาร์ป

#### ความจำภายใน ( local memory )

ความจำภายในนั้นสามารถถูกเข้าถึงได้เฉพาะเทรดที่จองเท่านั้น ที่เป็นเช่นนี้ไม่ได้เป็นเพราะมันอยู่ใกล้หน่วยประมวลผล อันที่จริงแล้วมันอยู่ภายนอกตัวชิพ ( off-chip ) ดังนั้นการใช้ความจำภายในนั้นค่อนข้างช้า และเปรียบเสมือนการเข้าถึงความจำทั่วไป ที่ไม่มีการแคชบนรุ่นความสามารถการประมวลผล 1.x ดังนั้นความจำภายในนั้นไม่ได้มีความเร็วในการเข้าถึงดังเช่นชื่อของมันในรุ่นความสามารถการประมวลผล 1.x

หน้าที่ของความจำภายในนั้นถูกใช้เก็บค่าตัวแปรที่เกิดขึ้นโดยอัตโนมัติซึ่งเป็นหน้าที่ของ nvcc ที่จะตัดสินใจว่ามีรีจิสเตอร์เพียงพอที่จะเก็บตัวแปรใหม่หรือไม่ โดยปกติแล้ว nvcc จะเก็บตัวแปรที่มีขนาดใหญ่ เช่น ตัวแปรโครงสร้าง หรืออะเรย์ ซึ่งพื้นที่ของรีจิสเตอร์อาจมีไม่เพียงพอต่อการเก็บค่าของตัวแปร และตัวแปรเองอาจมีการขยายขนาดภายหลัง

ในการดูว่า nvcc ตัดสินใจให้ตัวแปรใดใช้ความจำภายในใด สามารถดูได้จากผลลัพธ์หลังการคอมไพล์โปรแกรมที่ออกมาเป็น PTX assembly (โดยเติมตัวเลือกการคอมไพล์ -ptx ให้กับ nvcc) หลังจากการคอมไพล์ส่วนแรก ถ้ามีการใช้ความจำภายในเกิดขึ้นจะถูกแสดงด้วย .local ถ้าพบว่าไม่มีการจองความจำภายในในการคอมไพล์ส่วนแรก อาจจะมีการจองเป็นความจำภายในในภายหลังได้เนื่องจากขนาดของข้อมูลที่เพิ่มขึ้นจนใหญ่เกินกว่ารีจิสเตอร์จะเก็บได้ การตรวจสอบว่าตัวแปรนั้นๆจะถูกจองในหน่วยความจำภายในหรือไม่นั้นไม่สามารถทำได้ แต่สามารถดูได้ในภาพรวมของด้วยการใส่ตัวเลือกในการคอมไพล์ -ptxas-options=-v



### ความจำเท็กเทอร์ (Texture Memory)

ความจำเท็กเทอร์เป็นความจำชนิดอ่านอย่างเดียวและมีแคช ดังนั้นตัวประมวลผลจะใช้การอ่านเสียเวลาในการอ่านข้อมูลจากความจำเท็กเทอร์เพียงครั้งเดียวในตอนแรก หรือตอนที่หาข้อมูลไม่พบในแคช หลังจากนั้นตัวประมวลผลจะอ่านข้อมูลจากแคชเพียงอย่างเดียว ทำให้ใช้เวลาในการอ่านน้อยเมื่อข้อมูลไม่มีการเปลี่ยนแปลงบ่อย ตัวเท็กเทอร์แคชนั้นถูกปรับแต่งเพื่อให้เข้ากับภาพสองมิติ ดังนั้นเทร็ดที่อยู่ในวาร์ปเดียวกันและอ่านความจำเท็กเทอร์ที่ตำแหน่งเดียวกันจะเข้าถึงหน่วยความจำอย่างมีประสิทธิภาพ

### ความจำคงที่ (Constant Memory)

มีขนาด 64 กิโลบิตและไม่ได้อยู่บนชิพ (off-chip) ลักษณะของความจำคงที่คือมีการแคชข้อมูลเหมือนกับความจำเท็กเทอร์ที่จะมีการอ่านข้อมูลมาใหม่เฉพาะข้อมูลที่หาได้จากแคชของมันทำให้ทำงานได้เร็วในกรณีที่ไม่มีการเปลี่ยนแปลงข้อมูลบ่อย

การอ่านข้อมูลจากแคชของความจำคงที่มีความเร็วพอๆกับการอ่านข้อมูลจากรีจิสเตอร์ที่เดียวกันเนื่องจากทุกเทร็ดที่อยู่ในครั้งวาร์ปอ่านข้อมูลที่ตำแหน่งเดียวกัน สำหรับการอ่านข้อมูลที่อยู่ต่างตำแหน่งกันการเข้าถึงจะเป็นไปในลักษณะอนุกรม ดังนั้นความเร็วในการอ่านเขียนข้อมูลของเทร็ดจะแปรผันกับจำนวนตำแหน่งที่แตกต่างของความจำที่เข้าถึงจากเทร็ด

### รีจิสเตอร์ (Register)

โดยทั่วไปการเข้าถึงรีจิสเตอร์จะใช้เวลาเป็นศูนย์ต่อการทำงานหนึ่งคำสั่ง แต่บางครั้งอาจมีความช้าได้เนื่องจาก รีจิสเตอร์อ่าน หลังจากมีการเขียนข้อมูลหรืออาจเป็นการขัดแย้งของการเข้าถึงแเบงค์ของรีจิสเตอร์

การค้างของการอ่านหลังการเขียนจะมีระยะเวลาประมาณ 24 รอบนาฬิกา แต่การค้างนี้แทบจะมองไม่เห็นเนื่องจากในกลุ่มของตัวประมวลผลจะมีอย่างน้อย 192 เทร็ดที่ทำงานอยู่ (เท่ากับ 6 วาร์ป) สำหรับรุ่นความสามารถการประมวลผล 1.x สำหรับรุ่นความสามารถการประมวลผล 2.0 นั้นมีเทร็ด 768 ซึ่งยิ่งทำให้การค้างแทบจะไม่มีเลย ตัวคอมไพล์และจัดการตารางของเทร็ดจะจัดการตารางขอคำสั่งเพื่อหลีกเลี่ยงการเกิดการขัดแย้งของแเบงค์ มันจะทำได้ดีที่สูงสุดถ้าจำนวนของเทร็ดต่อบล็อกเป็นจำนวนเท่าของ 64

ภาวะกีดกันของรีจิสเตอร์ เป็นชื่อเรียกของเหตุการณ์ที่ไม่มีรีจิสเตอร์เพียงพอต่องานทำป้อนเข้าไปยังตัวประมวลผลถึงแม้ว่าตัวประมวลผลนั้นจะมีรีจิสเตอร์ 32 บิต อยู่จำนวนหลักพัน เพราะว่ารี้จิสเตอร์เหล่านี้ต้องถูกแบ่งโดยเทร็ดจำนวนมากเช่นกันที่อยู่บนตัว

ประมวลผลนั้น เพื่อป้องกันเหตุการณ์นี้ สามารถใส่ตัวจำกัดจำนวนรีจิสเตอร์ที่ใช้ได้โดยใส่  $\text{maxrregcount}=N$  โดยที่  $N$  คือจำนวนรีจิสเตอร์สูงสุดที่ให้ใช้ได้

ใน GPU นั้นมีหน่วยความจำหลายระดับแต่ที่ GPU ใช้ติดต่อกับ CPU นั้นจะทำผ่านความจำทั่วไปซึ่งการเข้าแลกเปลี่ยนข้อมูลจะช้ากว่าหน่วยความจำที่อยู่กับหน่วยประมวลผลย่อยแต่ละตัว ซึ่งถูกเรียกว่า ความจำร่วมซึ่งหน่วยความจำนี้จะอยู่ติดกับหน่วยประมวลผลย่อยแต่ละตัวทำให้การแลกเปลี่ยนข้อมูลสามารถทำได้ด้วยความเร็วกว่าความจำทั่วไปและทุก เทรด ภายในหน่วยประมวลผลย่อยนั้นสามารถเห็นและแลกเปลี่ยนข้อมูลระหว่างเทรด ด้วยความจำร่วมนี้ได้ การเขียนโปรแกรมให้ย้ายข้อมูลที่ต้องเรียกประมวลผลมากหรือบ่อยให้มาอยู่ที่ความจำร่วมแทนก็เป็นการเพิ่มความเร็วให้กับการประมวลผลด้วยเช่นกัน

การนำ GPU ประยุกต์เข้ากับงานทั่วไปหรือ General Purpose computing on Graphics Processing Units (GPGPU) ชุดพัฒนาของ NVidia นี้ ทำให้เราสามารถใช้อยู่ประโยชน์ของสถาปัตยกรรมที่ออกแบบมาสำหรับการประมวลผลแบบขนาน มาช่วยเพิ่มความเร็วในการคำนวณงานที่สามารถออกแบบให้อยู่ในรูปแบบขนานได้ ด้วยการใช้ภาษา C ในการเขียนโปรแกรมทำให้นักพัฒนาเรียนรู้เพิ่มเติมอีกเล็กน้อยก็สามารถเขียนโปรแกรมได้ด้วยโมเดลการเขียนโปรแกรมแบบขนานที่ถูกออกแบบมาสำหรับการแก้ปัญหาโดยใช้หลักสามอย่าง คือ การแบ่งงานออกเป็นกลุ่มของ เทรด , ความจำร่วม และ การ synchronization การแบ่งงานที่ไม่เกี่ยวเนื่องหรือไม่ขึ้นต่อกันไปยังแต่ละบล็อกและการแบ่งปัญหาย่อยที่ต้องใช้การติดต่อแลกเปลี่ยนข้อมูลกันให้อยู่ในรูปแบบเทรด ในแต่ละบล็อกอีกที ซึ่ง synchronization จะเข้ามาเพื่อทำให้แน่ใจว่างานที่กำลังดำเนินอยู่ในแต่ละ เทรด จะมาหยุดพร้อมกันที่จุดที่เรากำหนด เราจึงได้เห็นงานวิจัยที่นำ GPU มาใช้งานกับขั้นตอนวิธี ที่เป็นที่รู้จักและสามารถเพิ่มความเร็วให้กับวิธีขั้นตอน เหล่านั้นได้ เช่น งานวิจัยดังต่อไปนี้

Jun Li [6] ที่นำความสามารถในการคำนวณที่ทรงพลังของ GPU มาประยุกต์ใช้กับขั้นตอนวิธีเชิงคาดการณ์ล่วงหน้า-ย้อนหลัง (forward-backward) ซึ่งมีความซับซ้อนเชิงเวลาเท่ากับ  $N^2T$  โดยขั้นตอนแรกคือออกแบบการทำงานแบบขนานให้ขั้นตอนการหาความน่าจะเป็นให้กับการคาดการณ์ล่วงหน้าแต่ละเส้นทาง และขั้นตอนที่สองคือการรวมค่าความน่าจะเป็นทั้งหมดด้วยวิธีแบบขนานอีกครั้ง ในงานวิจัยยังได้ใช้ประโยชน์จากความเร็วของหน่วยความจำที่เร็วกว่าหน่วยความจำบน CPU เพื่อเพิ่มความเร็วขึ้นอีก งานวิจัยนี้สามารถเร่งความเร็วให้กับ Hidden Markov Models ได้ 4-25 เท่า

งานของ Stefano [7] ที่นำ GPU มาเร่งความเร็วของโครงข่ายประสาทเทียม (Artificial Neural Network) ในงานการเรียนรู้จำเสียงโดยใช้กระบวนการฝึกฝนแบบส่งค่าย้อนกลับ (back-propagation) งานวิจัยได้ทำการเรียนรู้โดยใช้แบบจำลองของเสียงที่มีคำศัพท์ขนาดใหญ่ ในงานวิจัยได้แสดงการเปรียบเทียบความเร็วที่ได้จากการใช้มัลติเทรคบนหน่วยประมวลผลหลายแกนกับความเร็วที่ได้จากตัวประมวลผลกราฟิก ซึ่งแสดงให้เห็นว่าสามารถลดเวลาที่ต้องใช้ในการเรียนรู้ชุดคำศัพท์ขนาดใหญ่นี้ลงได้ 6 เท่า

งานของ Hongwei [8] ที่นำ GPU มาเร่งความเร็วของ Message Digest algorithm 5 (MD5) ด้วยการออกแบบเป็นสองขั้นตอน ขั้นตอนแรกคือการประมวลผลเบื้องต้นของข้อความเช่นการสำรวจเข้าไปในข้อความเพื่อหาชุดของตัวอักษรที่จะเข้ารหัสและหาความยาวของข้อความ ขั้นตอนที่สองคือการเข้ารหัสพร้อมๆกันหลายข้อความ โดยหนึ่งเทรคแทนการทำงานของเข้ารหัสให้กับหนึ่งข้อความ งานวิจัยได้แสดงให้เห็นว่าสามารถเพิ่มความเร็วได้มากกว่า 10 เท่าของการทำงานบน CPU

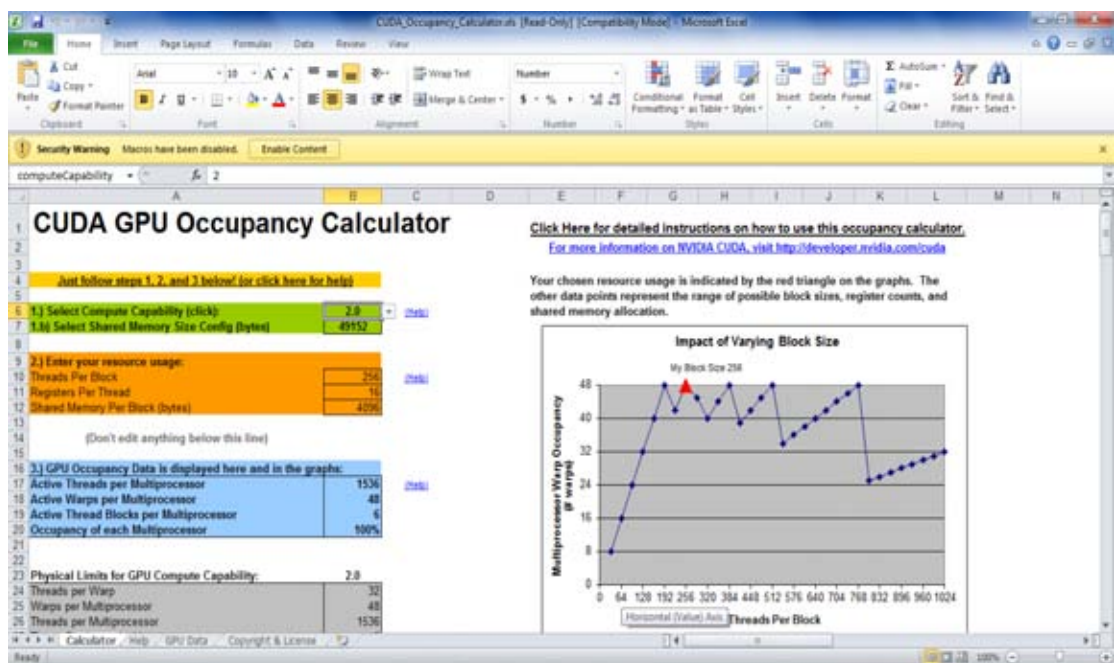
## บทที่ 3

### วิธีดำเนินการวิจัย

ความสามารถในการประมวลผลแบบขนานของหน่วยประมวลผลกราฟิก สามารถนำมาประยุกต์ใช้กับขั้นตอนวิธีปฏิบัติการร่วมกันเพื่อเพิ่มความเร็วได้ด้วยการศึกษาและปฏิบัติตามแนวทางดังต่อไปนี้

#### 3.1 การคำนวณประสิทธิภาพการใช้งานหน่วยประมวลผลกราฟิก

หนึ่งในความสามารถของหน่วยประมวลผลกราฟิก คือ การที่หน่วยประมวลผลกราฟิกออกแบบมาเพื่อการคำนวณปริมาณมาก โดยการจัดวางหน่วยความจำที่ติดอยู่กับแต่ละหน่วยประมวลผล ดังนั้นการใช้หน่วยประมวลผลได้อย่างเต็มความสามารถ จำนวนเทรดต่อบล็อกเป็นปัจจัยหนึ่ง จำนวนเทรดต่อบล็อกที่สอดคล้องกับจำนวนของหน่วยประมวลผลจะแตกต่างกันไปตามรุ่นของความสามารถในการคำนวณของอุปกรณ์ประมวลผลกราฟิก



รูปที่ 3-1 ตารางการคำนวณการครอบครองหน่วยประมวลผลกราฟิกของ NVIDIA

สำหรับการคำนวณจำนวนเทรตต่อบล็อกนั้น ชุดพัฒนาของ NVIDIA ได้ให้เครื่องมือซึ่งเป็นตารางคำนวณการครอบครองหน่วยประมวลผลกราฟิกมาดังรูปที่ 3-1 โดยการใช้เบื้องต้นคือ

1. เลือกรุ่นของความสามารถในการคำนวณของอุปกรณ์ประมวลผลกราฟิกให้ตรงกับรุ่นของอุปกรณ์ประมวลผลกราฟิกที่ใช้ ซึ่งรุ่นของความสามารถในการประมวลผลที่สูงกว่าหรือเท่ากับ 2 นั้น จะต้องเลือกหน่วยความจำร่วมกันที่ตรงกับรุ่นของอุปกรณ์ประมวลผล

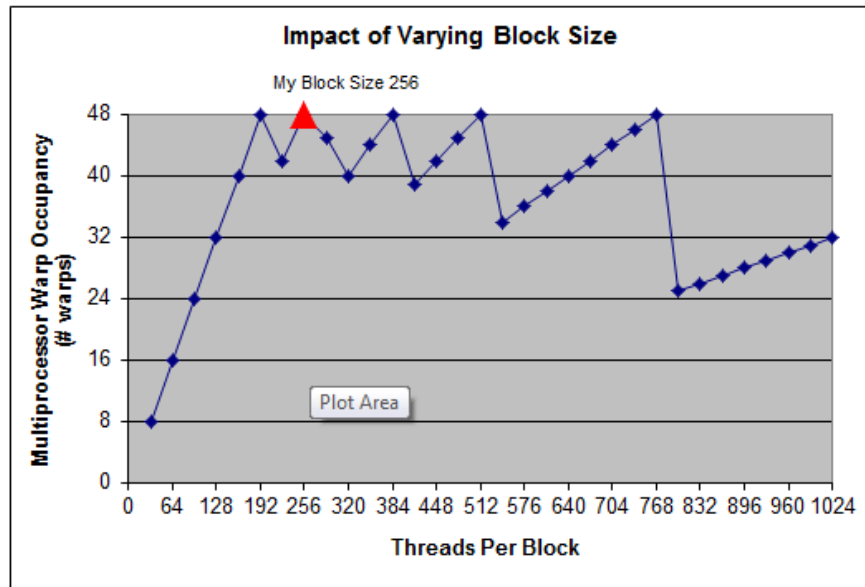
```

1>ptxas info      : Compiling entry function
'_Z14SortFitnessGPUiiP9stFitnessPiS1_S1_' for 'sm_20'
1>ptxas info      : Function properties for
'_Z14SortFitnessGPUiiP9stFitnessPiS1_S1_'
1>      0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
1>ptxas info      : Used 15 registers, 16+0 bytes smem, 72 bytes cmem[0],
16 bytes cmem[14]

```

รูปที่ 3-2 ตัวอย่างข้อความที่ได้จากการคอมไพล์เคอร์เนล SortFitness ด้วยตัวเลือกการคอมไพล์ --ptxas-options=-v จะได้จำนวนรีจิสเตอร์ที่ใช้ไป 15 รีจิสเตอร์และหน่วยความจำร่วมกันที่ใช้ไป 16 ไบท์

- 2 ใส่จำนวนรีจิสเตอร์และหน่วยความจำร่วมกันสำหรับแต่ละเคอร์เนลที่ได้จากขั้นตอนการคอมไพล์โปรแกรมด้วยตัวเลือกการคอมไพล์ --ptxas-options=-v เพื่อให้ตัวคอมไพล์ nvcc แสดงรายละเอียดของรีจิสเตอร์และหน่วยความจำร่วมกันออกมาดังรูปที่ 3-2 เดิมตัวเลขรีจิสเตอร์ในช่อง Registers Per Thread สำหรับตัวเลขของหน่วยความจำร่วมกันที่แสดงออกมาในขั้นตอนคอมไพล์โปรแกรมนั้นเป็นจำนวนหน่วยความจำร่วมกันที่ระบุจำนวนที่แน่นอนที่ยังไม่นับรวมหน่วยความจำร่วมกันแบบภายนอกซึ่งใช้ตัวกำหนดชนิดของตัวแปรแบบ extern \_\_shared\_\_ ซึ่งผู้ใช้ต้องนับเองว่าใช้หน่วยความจำร่วมกันในแต่ละเคอร์เนลไปเป็นจำนวนเท่าไรลงไปยังช่อง Shared Memory Per Block (bytes)



รูปที่ 3-3 กราฟแสดงการครอบครองของหน่วยประมวลผล

- ปรับจำนวนเทรตต่อบล็อกในช่อง Threads Per Block เพื่อให้ได้ค่าการครอบครองมากที่สุด เพื่อให้การใช้หน่วยประมวลผลมีประสิทธิภาพมากที่สุดจากรูปที่ 3-3 จำนวนเทรตต่อบล็อกที่ทำให้จำนวนเหมาะสมที่สุด 48 วาร์ป (จะแตกต่างกันตามรุ่นของอุปกรณ์ประมวลผลกราฟิก) คือที่จำนวนเทรตต่อบล็อกเท่ากับ 192 ,256 ,384 ,512 และ 768

สำหรับการคำนวณการครอบครองนั้นเป็นเพียงวิธีหนึ่งที่ทำให้ประสิทธิภาพการประมวลผลมากขึ้นแต่ความเร็วในการประมวลผลก็ยังขึ้นอยู่กับปัจจัยอื่นๆ อย่างเช่น การจัดเก็บข้อมูลในหน่วยความจำ ซึ่งถ้าสามารถจัดวางให้เรียงต่อกันจะทำให้อัตราการอ่านหรือเขียนข้อมูลมีประสิทธิภาพสูงขึ้น

### 3.2 การกำหนดตัวแปรและค่าเริ่มต้น

การกำหนดตัวแปรที่เป็นแบบทั่วไปซึ่งต้องใช้แลกเปลี่ยนข้อมูลทั้งอ่านและเขียนในการทำงานของเคอร์เนลที่เกี่ยวข้อง เมื่อเราพิจารณาขั้นตอนทั้ง 5 ขั้นตอนแล้วเราจะพบว่าจะต้องมีตัวแปรทั่วไปที่สำคัญดังนี้

- ตัวแปรสำหรับเก็บตาราง จะถูกอ่านในขั้นตอนการกำเนิดประชากรและจะถูกเขียนเมื่ออยู่ในขั้นตอนปรับปรุงตัวกำเนิด

2. ตัวแปรสำหรับเก็บประชากร ในขั้นตอนกำเนิดประชากรจะสร้างประชากรซึ่งเป็นลำดับของเมืองที่ต่อเนื่องกันและนำลำดับนี้ไปเก็บในตัวแปรสำหรับเก็บประชากร ตัวแปรนี้จะถูกนำไปใช้อีกครั้งในขั้นตอนการปรับปรุงประชากร
3. ตัวแปรสำหรับเก็บระยะทางระหว่างเมือง ตัวแปรนี้จะถูกอ่านโดยเคอร์เนลที่ทำหน้าที่รวมระยะทางของลำดับเมืองที่ถูกสร้างจากขั้นตอนกำเนิดประชากร

### 3.3 การประยุกต์ขั้นตอนวิธีปฏิบัติการร่วมกันบนหน่วยประมวลผลกราฟิก

ขั้นตอนวิธีปฏิบัติการร่วมกันแบ่งออกเป็น 5 ขั้นตอนหลักเช่นกันและนำมาออกแบบการทำงานแบบขนานได้ดังนี้

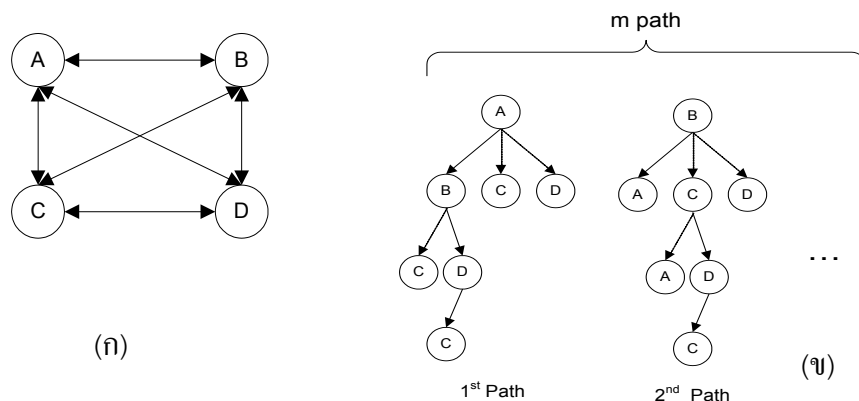
#### 3.3.1 การกำหนดค่าเริ่มต้น

ในขั้นตอนนี้การทำงานไม่ต่างจากที่กล่าวถึงในทฤษฎีซึ่งจะทำงานครั้งเดียวตอนเริ่มต้น จึงไม่มีการออกแบบใดๆ เพื่อเพิ่มความเร็วและขั้นตอนนี้จะประมวลผลบนหน่วยประมวลผลกลางตามปกติ

#### 3.3.2 การสร้างกลุ่มของประชากร

ก. ลักษณะงาน

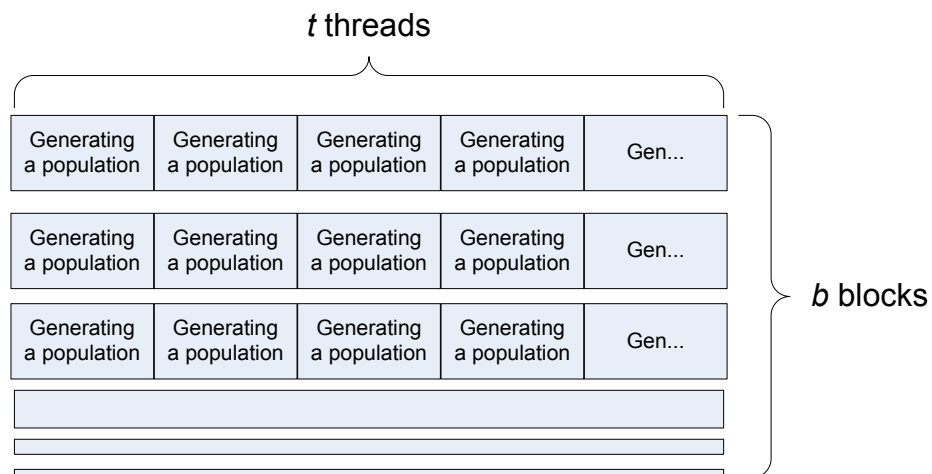
ขั้นตอนวิธีจะสร้างคำตอบซึ่งเป็นลำดับงานย่อยที่เรียงต่อกันออกมาจำนวนหนึ่งจากการสุ่มเลือกแต่ละโหนดตามความน่าจะเป็นที่อยู่ใน ตัวกำเนิด เมื่อเปรียบเทียบกับปัญหา TSP คำตอบคือ ลำดับของเส้นทางที่สามารถผ่านทุกเมืองได้โดยไม่ย้อนกลับมายังเมืองที่ไปมาแล้ว



ภาพที่ 3-4 (ก) แสดงเส้นทางระหว่างเมืองในปัญหา TSP (ข) แสดงการสร้างคำตอบซึ่งเป็นเส้นทางจากเมืองสู่เมืองจำนวน m เส้นทาง

ข. การแบ่งงาน

ในขั้นตอนนี้ งานที่เกี่ยวข้องคือการค้นหาเส้นทางตามความน่าจะเป็นที่อยู่ในตัวกำเนิด เมื่อพิจารณาความเกี่ยวข้องของงานในการหาเส้นทางแต่ละเส้นทางแล้ว สามารถพิจารณาได้ว่าไม่มีความเกี่ยวข้องกัน โดยการหาเส้นทางแต่ละเส้นจะใช้เฉพาะข้อมูลความน่าจะเป็นจากตัวกำเนิด ร่วมกันและไม่มีการเขียนค่าใดๆ กลับลงไปยังตัวกำเนิด ดังนั้นเราสามารถแบ่งงานออกให้อยู่ในรูปของบล็อกและเทรต โดยแต่ละเทรตจะทำการกำเนิดเส้นทางเป็นอิสระโดยไม่เกี่ยวข้องกันกับเทรตอื่น



รูปที่ 3-5 การแบ่งงานของการกำเนิดเส้นทางออกเป็น  $b$  บล็อก และ  $t$  เทรต โดยงานในแต่ละเทรตนั้นทำงานเป็นอิสระไม่เกี่ยวข้องกันทั้งในบล็อกเดียวกันและต่างบล็อก

สำหรับการหาจำนวนเทรตที่เหมาะสมในแต่ละบล็อกนั้นยังคงใช้ตารางคำนวณ “CUDA Occupancy Calculator” โดยป้อนข้อมูลรีจิสเตอร์และหน่วยความจำรวมกันที่ได้ออกมาจากการคอมไพล์รหัสต้นฉบับของโปรแกรม จากนั้นตารางจะแสดงจำนวนเทรตต่อบล็อกที่เหมาะสมที่สุดออกมาบนกราฟ ให้เราเลือกจำนวนเทรตต่อบล็อกที่มีการครอบครองหน่วยประมวลผลมากที่สุดมา สำหรับจำนวนบล็อกคือการนำจำนวนประชากร  $m$  ประชากรมาหารด้วยจำนวนเทรต  $t$  ดังนั้นจะได้  $b = \frac{m}{t}$  เมื่อเรากำหนดจำนวนบล็อกและเทรตแล้วจะสร้างเทรตออกมาทั้งหมด  $b \times t$  ตัว ดังนั้นคอร์เนลการกำเนิดประชากรจะถูกเรียกเป็นจำนวนเท่ากัน

ขั้นตอนการทำงานของคอร์เนลสามารถเขียนเป็นลำดับได้ดังนี้

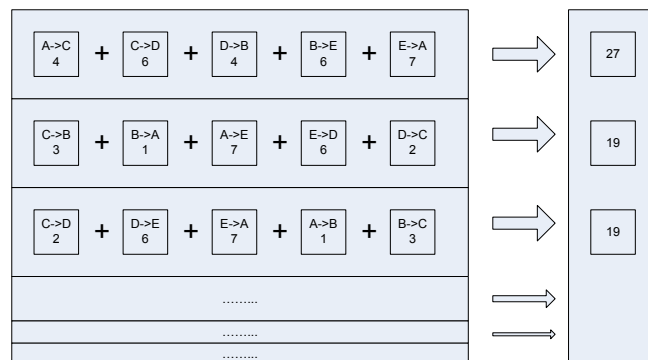


1. สุ่มตัวเลขเพื่อหาเมืองแรก  $X_1$  โดยสุ่มจากความน่าจะเป็น  $\frac{1}{(n-1)}$  ที่ซึ่ง  $n$  คือจำนวนเมือง
2. วนรอบตามจำนวนเมืองที่เหลือ  $n - 2$  รอบ
  - ก. สุ่มตัวเลขเพื่อหาเมืองถัดไป  $X_j$  โดยดูว่าตัวเลขที่ได้มาตกอยู่ในช่วงความน่าจะเป็นร่วม  $h(X_i|X_j)$  ของ  $X_i, X_j$  ใดๆ
  - ข. ลบเมืองที่เลือกแล้วออกจากเมืองที่สามารถเลือกได้
  - ค. วนซ้ำจนครบตามจำนวนรอบ
3. ส่งคำตอบซึ่งอยู่ในรูปของลำดับเมืองกลับไปเก็บที่ตัวแปรแบบทั่วไป

### 3.3.3 การประเมินค่าความเหมาะสมของประชากรแต่ละตัว

#### ก. ลักษณะงาน

การประเมินค่าความเหมาะสมของประชากรคือการหาค่าความเหมาะสม (Fitness) ของแต่ละคำตอบ ซึ่งประเมินได้จากฟังก์ชันค่าความเหมาะสมที่จะแตกต่างกันไปตามลักษณะปัญหา หากเปรียบเทียบกับปัญหา TSP ในขั้นตอนนี้คือการหาระยะทางรวมของลำดับเมืองที่สร้างจากขั้นตอนกำเนิดประชากร โดยนำเอาระยะทางระหว่างเมืองมารวมกันออกมาเป็นค่าความเหมาะสมของแต่ละเส้นทาง

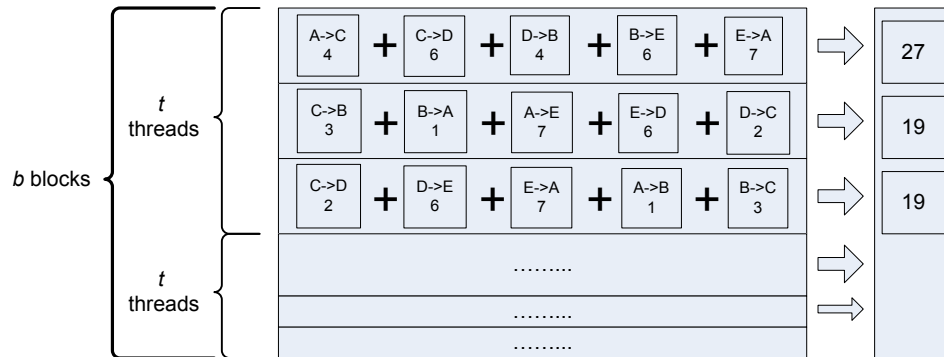


รูปที่ 3-6 การคำนวณระยะทางรวมของแต่ละเส้นทางที่สร้างขึ้นมาจากขั้นตอนสร้างประชากร

#### ข. การแบ่งงาน

การกระจายงานที่สามารถทำไปพร้อมกันเป็นวิธีที่ทำให้การประมวลผลรวมนั้นมีความเร็วที่สุด สามารถแบ่งงานของการรวมระยะทางของแต่ละเส้นทางให้ไปอยู่ใน บล็อกและเทร็ด โดยการคำนวณหาจำนวนเทร็ดต่อบล็อกที่เหมาะสม ใช้การบ่อนค่ารีจิสเตอร์และ

หน่วยความจำร่วมที่ถูกใช้ไปในคอร์เนลนี้ซึ่งหาได้จากการขั้นตอนการคอมไพล์โปรแกรมลงไปยังตาราง “CUDA Occupancy Calculator” ทำที่ผู้ศูกรภาพจะแสดงค่าเทรกดต่อบล็อกที่เหมาะสม



รูปที่ 3-7 การแบ่งงานของการประเมินค่าความเหมาะสมของ  $m$  เส้นทางไปยัง  $m$  บล็อก และภายในแต่ละบล็อกแบ่งเป็น  $n+1$  เทรด

สามารถแบ่งงานของการประเมินค่าความเหมาะสมออกได้เป็น  $b$  บล็อก และ  $t$  เทรด โดยจำนวนของบล็อกหาได้จากการนำจำนวนประชากรทั้งหมดหารด้วยจำนวนเทรกดต่อบล็อก  $b = \frac{m}{t}$  โดย  $m$  จำนวนประชากร สำหรับขั้นตอนการทำงานย่อยในแต่ละเทรกดหรือแต่ละคอร์เนลของการประเมินค่าความเหมาะสมมีดังนี้

1. วนรอบเป็นจำนวน  $n - 1$  เมือง
2. อ่านคู่ลำดับของเมือง  $X_i, X_{i+1}$  จากตัวแปรทั่วไปที่เก็บประชากร
3. อ่านค่าระยะทางระหว่างเมืองของ  $X_i, X_{i+1}$  จากตัวแปรทั่วไปที่เก็บระยะทางระหว่างเมือง
4. บวกระยะทาง
5. เริ่มขั้นตอนสาม โดยเลื่อนไปยังคู่ลำดับถัดไป  $i = i + 1$  จน  $n - 1$  รอบ

### 3.3.4 การเรียงค่าความเหมาะสม

ในการเรียงลำดับค่าความเหมาะสมนี้ เราได้เลือกการเรียงลำดับแบบประสาร เพราะมีความเร็วในการเรียงลำดับเร็วที่สุดแบบหนึ่ง คือ  $O(n \log n)$  สำหรับข้อมูล  $n$  ตัว

ก. ลักษณะงาน

จากหัวข้อที่ 3.3.3 หลังจากที่เราได้ ข้อมูลชุดหนึ่งที่เก็บค่าความเหมาะสมของทุกตัวอย่างประชากรแล้วนำมาเรียงลำดับจากน้อยไปหามาก เมื่อประยุกต์ใช้กับปัญหา TSP การ



0.05 และนำไปเพิ่มให้กับ (A,B) (ค) แสดงการกระจายค่าความน่าจะเป็นให้กับ คอลัมน์อื่น ในกรณี (A,B) เป็นส่วนหนึ่งของเส้นทางกลุ่มไม่ดี

งานต่อมาที่ต้องทำ คือ การพิจารณาหน่วยย่อยของคำตอบที่ดีและไม่ดีมาเพิ่มหรือลดความน่าจะเป็นของ ตัวกำเนิด หากยกตัวอย่างปัญหา TSP กลุ่มตัวอย่างที่ดีคือ กลุ่มเส้นทางที่ค่าความเหมาะสมมากที่สุด หรือเป็นกลุ่มเส้นทางที่สั้นที่สุดจำนวน C เปอร์เซนต์ของเส้นทางทั้งหมด เส้นทางระหว่างเมือง A->B ซึ่งเป็นสมาชิกของกลุ่มเส้นทางที่ดี A->B->C->D->E->A แล้ว ค่าความน่าจะเป็นของ (A,B) , (A,C) ,(A,D) ,(A,E) จะถูกลดลงและรวบรวมไปเพิ่มให้กับ (A,B) ด้วยอัตราส่วนที่เท่ากัน ภาพที่ 3-8 (ก) แสดงค่าเริ่มต้น ซึ่งความน่าจะเป็นของการเลือกทุกเส้นทางจะมีค่าเท่ากัน คือ 0.25 ภาพที่ 3-8 (ข) เป็นการปรับปรุงค่าความน่าจะเป็นในกรณี (A,B) เป็นส่วนหนึ่งของเส้นทางกลุ่มดี ตำแหน่งที่ไม่ใช่ (A,B) จะถูกดึงค่าความน่าจะเป็นออกไป 0.05 และนำไปเพิ่มให้กับ (A,B) ในทางตรงกันข้าม ถ้า (A,B) เป็นส่วนหนึ่งของเส้นทางกลุ่มไม่ดี ค่าจาก คอลัมน์ (A,B) จะถูกกระจายไปให้กับ คอลัมน์อื่นที่ไม่ใช่ (A,B) ดังภาพที่ 3-8 (ค)

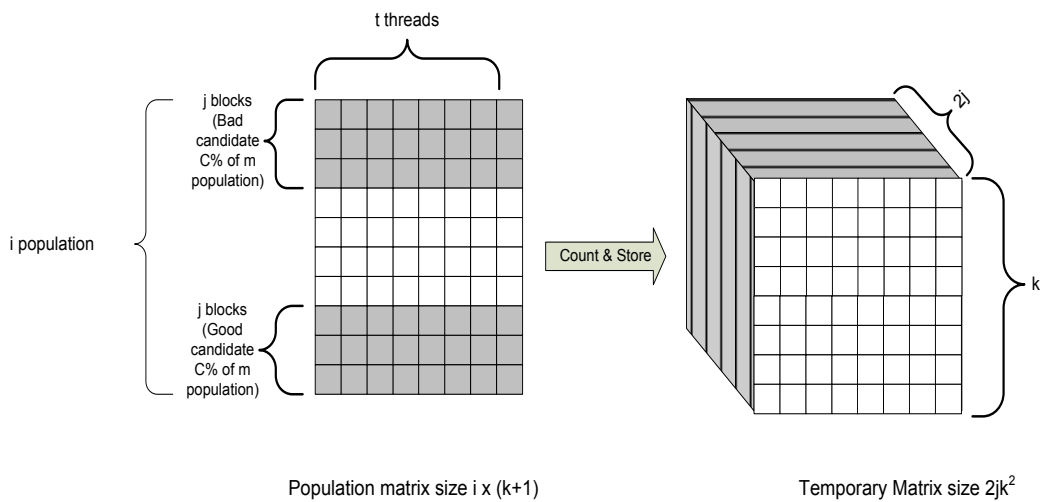
#### ข. การแบ่งงาน

จากทุกขั้นตอนด้านบน จะได้ตารางของประชากรที่เรียงลำดับตามค่าความเหมาะสมแล้ว ในขั้นตอนนี้จะดึงเอากลุ่มของประชากรที่ดีจำนวน  $j$  ตัวหรือ  $C\%$  ของประชากรทั้งหมดและกลุ่มของประชากรที่ไม่ดีจำนวนอีก  $j$  ตัวหรือ  $C\%$  เช่นกันรวมแล้ว  $2j$  ตัวหรือ  $2C\%$  ของประชากรทั้งหมดมา ปรับปรุง ตัวกำเนิด จากลักษณะงานที่ได้อธิบายไป เมื่อนำประชากร 1 ตัวขนาด  $k+1$  ไปปรับปรุง ตัวกำเนิด แล้วจะเกิดงานขึ้น  $k^2$  สำหรับการปรับปรุงตัวเลขความน่าจะเป็นรวมทั้งหมดใน ตัวกำเนิด ขนาด  $k \times k$  ตัว ดังนั้นเมื่อนำประชากรจำนวน  $2j$  ตัว ไปปรับปรุง ตัวกำเนิด จะเกิดงานขึ้น  $2jk^2$

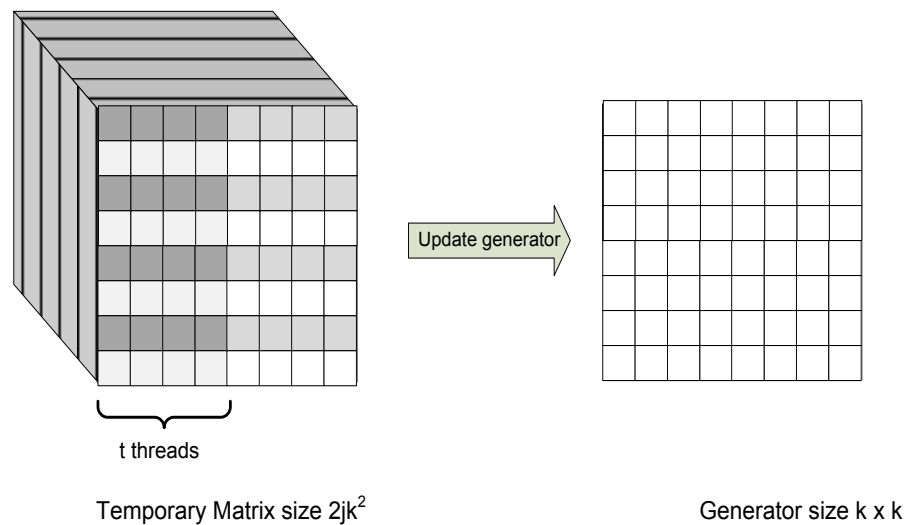
เมื่อมองลึกลงไปอีกสำหรับงานนี้ จุดประสงค์หลักในงาน คือ การค้นเข้าไปใน population matrix ขนาด  $i \times (k+1)$  (เมื่อ  $i$  คือจำนวนประชากรที่สร้างขึ้นในแต่ละรุ่น และ  $k$  คือขนาดของปัญหา) แล้วนับจำนวนคู่ลำดับ  $X_c, X_r$  ที่พบในกลุ่มดีได้ผลรวมออกมาเป็น  $r_{c,r}$  และคู่ลำดับ  $X_c, X_p$  ที่พบในกลุ่มไม่ดีได้ผลรวมออกมาเป็น  $p_{c,p}$  จากนั้นจึงนำเข้าสมการ (1),(2) เพื่อปรับปรุง ตัวกำเนิด

แบ่งงานออกเป็นสองขั้นตอนใหญ่สำหรับการประมวลผลแบบขนานคือ แบ่งงานการตรวจสอบว่ามีคู่ลำดับ  $X_c, X_r$  และ  $X_c, X_p$  ใน population matrix ออกเป็น  $2j \times k$  บล็อก

และนำผลไปเก็บไว้ในเมทริก 3 มิติ ซึ่งผลลัพธ์ชั่วคราวมีขนาด  $2j \times k \times k$  ดังภาพ 3-9 (ก) ขั้นตอนที่สองรวมค่าในมิติ  $2j$  ให้เหลือคำตอบเป็นเมทริก 2 มิติ ขนาด  $k \times k$  งานจะถูกแบ่งออกให้อยู่ในรูปเทร็ด เพื่อใช้การรวมแบบขนานเช่นเดียวกับที่ใช้ในขั้นตอนประเมินค่าความเหมาะสมของประชากรเพื่อเพิ่มความเร็ว ส่วนงานในมิติ  $k \times k$  นั้นเป็นงานที่ไม่เกี่ยวเนื่องกันจึงสามารถแบ่งงานให้อยู่ในรูปแบบบล็อก  $k$  ได้ ดังเช่นภาพที่ 3-9 (ข)



(ก) Generate temporary matrix



(ข) Update generator matrix

รูปที่ 3-9 (ก) การปรับปรุงค่าในตัวกำหนด เริ่มจากการนับจำนวนคู่ลำดับ  $X_c, X_r$  และ  $X_c, X_p$  และนำไปเก็บไว้ในตาราง (ข) แสดงการรวมค่าในมิติ  $2j$  เพื่อนำไปปรับปรุงค่าในตัวกำหนด

## บทที่ 4

### การทดลองและผลการทดลอง

#### เครื่องมือที่ใช้ในการวิจัย

ในการทดลองใช้หน่วยประมวลผลกลาง Intel core i3 ความเร็วนาฬิกา 2.1 GHz หน่วยความจำ 8 GB ซึ่งจะใช้ประมวลผล วิธีปฏิบัติการร่วมกันแบบประมวลผลแบบลำดับ ต่อเนื่องและหน่วยประมวลผลกราฟิก NVIDIA Geforce GT 540 M ซึ่งมีความเร็วของสัญญาณนาฬิกา 1344 MHz ขนาดของหน่วยความจำภายใน 2 GB โดยจะใช้ทำสอบประมวลผลวิธีปฏิบัติการร่วมกันแบบขนาน

#### การทดลอง

การทดลองจะทำการประมวลผลวิธีปฏิบัติการร่วมกันทั้งแบบประมวลผลแบบลำดับต่อเนื่องและแบบประมวลผลแบบขนาน ข้อมูลที่ได้จากการทดลองจะอยู่ในรูปของเวลา และค่าเหมาะสมที่สุดของปัญหา การเดินทางของนักขาย (TSP) ที่คำนวณออกมาได้จากวิธีปฏิบัติการร่วมกันทั้งสองแบบ

#### การวัดเวลา

ฟังก์ชันที่ใช้เพื่อวัดเวลานั้นใช้ฟังก์ชันที่วัดเวลาของภาษา C ได้แก่ฟังก์ชัน QueryPerformanceFrequency [11] และฟังก์ชัน QueryPerformanceCounter [12] ทำงานร่วมกันเพื่อเริ่มต้นการวัดเวลา

```
void startWatch()  
{  
    QueryPerformanceFrequency( (LARGE_INTEGER *)&pf );  
    freq_ = 1.0 / (double)pf;  
    QueryPerformanceCounter( (LARGE_INTEGER *)&baseTime_ );  
}
```

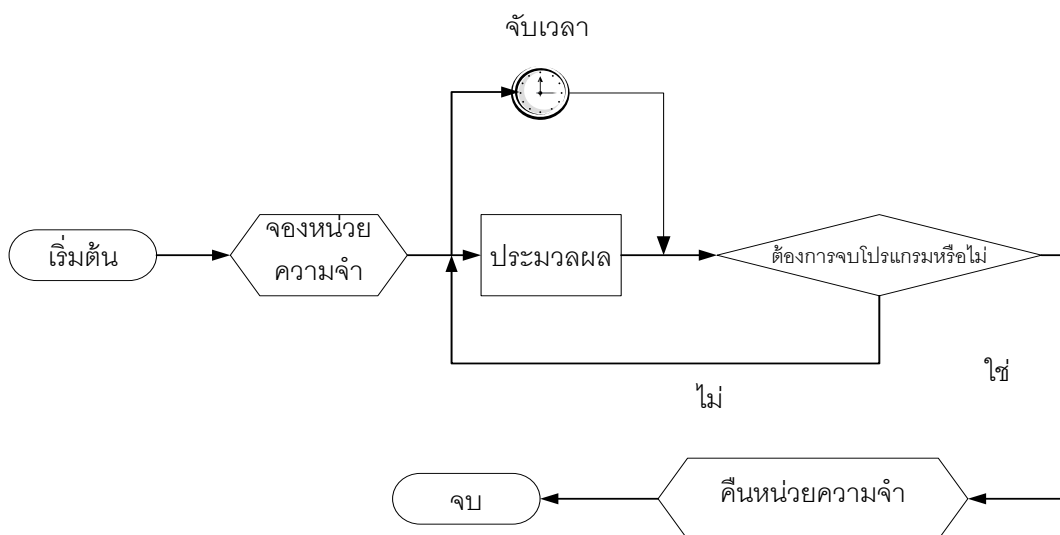
รูปที่ 4-1 ฟังก์ชันที่ใช้ในการเริ่มต้นจับเวลา ฟังก์ชัน startWatch ซึ่งภายในใช้ ฟังก์ชัน QueryPerformanceFrequency ใช้เพื่อหาความถี่ของของตัววัดประสิทธิภาพที่กำลังใช้อยู่ และฟังก์ชัน QueryPerformanceCounter ใช้เพื่อเริ่มต้นนับเวลา

ฟังก์ชัน QueryPerformanceFrequency นั้นใช้เพื่อหาจำนวนหน่วยเวลาที่วัดได้ในหนึ่งวินาทีของตัววัดประสิทธิภาพความละเอียดสูงที่ใช้อยู่ ซึ่งถ้าหน่วยประมวลผลกลางที่ใช้อยู่นั้นไม่สนับสนุนตัววัดประสิทธิภาพความละเอียดสูงนี้ค่าที่ได้กลับมาจะเป็น 0 ค่าได้มาจากฟังก์ชันนี้เป็นจำนวนเวลาที่นับได้ในหนึ่งวินาที ดังนั้นความถี่ต้องนำมาหารด้วย หนึ่งเพื่อกลายเป็นความถี่ หลังจากนั้นฟังก์ชัน QueryPerformanceCounter จำทำการเริ่มจับเวลาโดยค่าที่คืนมาจะเป็นเวลา ณ จุดเริ่มต้น

```
void stopWatch ()
{
    QueryPerformanceCounter( (LARGE_INTEGER *)&val );
    dif = (val - baseTime_) * freq_;
    printf("Execute time %g", dif);
}
```

รูปที่ 4-2 ฟังก์ชันที่ใช้ในการหยุดจับเวลา ฟังก์ชัน startWatch ซึ่งภายในใช้ ฟังก์ชัน QueryPerformanceFrequency ใช้เพื่อหาความถี่ของของตัววัดประสิทธิภาพที่กำลังใช้อยู่ และ ฟังก์ชัน QueryPerformanceCounter ใช้เพื่อเริ่มต้นนับเวลา

หลังจากประมวลผลเสร็จ ฟังก์ชัน QueryPerformanceCounter จะถูกเรียกอีกครั้งและค่าที่ได้จากฟังก์ชันนี้ คือ เวลา ณ เวลาปัจจุบัน ดังนั้น เวลาที่ใช้ไปคือการนำค่าเวลาที่เรารับได้ตอนเริ่มต้นการจับเวลา มาลบออกจากค่าเวลาที่ได้ ณ เวลาปัจจุบัน



รูปที่ 4-3 ตำแหน่งการจับเวลาโดยเริ่มจากการจองหน่วยความจำและหยุดหลังจากประมวลผลเสร็จ

การวัดเวลาจะเริ่มนับวัดจากการส่งข้อมูลนำเข้าที่อยู่ในรูปตารางของความยาวระหว่างเมืองของแต่ละเมือง ไปให้แก่อันตอนการประมวลผลเพื่อให้ได้คำตอบหลังจากประมวลและได้คำตอบเป็นค่าที่เหมาะสมสำหรับปัญหาออกมา จะหยุดจับเวลา เวลาที่ได้จะอยู่ในรูปของวินาที

การวัดเวลาที่บนหน่วยประมวลผลกลางและบนหน่วยประมวลผลกราฟิกนั้นไม่แตกต่างกัน แต่เนื่องจากหลักการทำงานของคำสั่งที่เรียกไปยังคอร์เนลของหน่วยประมวลผลกราฟิกนั้น คอร์เนลจะคืนค่ากลับมายังฝั่งของหน่วยประมวลผลกลางทันทีโดยมิได้ต้องประมวลผลให้เสร็จสิ้นก่อน ดังนั้นหน่วยประมวลผลกลางจะเรียกไปยังคอร์เนลถัดไปที่ เป็นเช่นนี้การวัดเวลาสำหรับหน่วยประมวลผลกราฟิกโดยใช้ฟังก์ชัน `QueryPerformanceFrequency` และ `QueryPerformanceCounter` จะผิดพลาด เวลาที่ได้จะเป็นแค่เวลาที่หน่วยประมวลผลกลางใช้เรียกไปยังคอร์เนลเท่านั้น ดังนั้น ก่อนที่จะเริ่มเรียกฟังก์ชันวัดเวลาต้องทำการทำให้ตรงกัน (synchronization) โดยใช้ฟังก์ชันของ CUDA คือ `cudaThreadSynchronize()` [3-1] สำหรับ CUDA เวอร์ชันต่ำกว่า 2.0 และสำหรับเวอร์ชันที่สูงกว่าหรือเท่ากับ 2.0 จะใช้ฟังก์ชัน `cudaDeviceSynchronize()` [3-2] เมื่อเรียกฟังก์ชันเหล่านี้ จะทำให้ ณ จุดที่เรียก เทตของหน่วยประมวลผลกลางและตัวควบคุมของหน่วยประมวลผลกราฟิกมีการทำให้ตรงกันด้านเวลา หลังจากเรียกฟังก์ชัน `cudaDeviceSynchronize()` แล้วจึงค่อยเรียกฟังก์ชันวัดเวลา และทำเช่นเดียวกันหลังจากเรียกคอร์เนลเพื่อประมวลผลเสร็จสิ้นแล้ว เมื่อต้องการวัดเวลาจะต้องเรียกฟังก์ชัน `cudaDeviceSynchronize()` อีกครั้งแล้วจึงเรียกฟังก์ชันวัดเวลา

เวลาที่เกิดจากการจองหน่วยความจำเริ่มต้นและเวลาจากการคืนหน่วยความจำนั้นจะไม่ได้ถูกนำมาวัดด้วยเนื่องจากวัตถุประสงค์ในการใช้งานที่ต้องการให้สามารถทำการทดลองซ้ำ ได้เรื่อยอีกๆ เพื่อปรับเปลี่ยนพารามิเตอร์ และเก็บผลการทดลองโดยที่ไม่ต้องออกจากโปรแกรม ดังนั้นเราจึงวัดเฉพาะเวลาที่เกี่ยวข้องในแต่ละรอบจริง

### การเก็บข้อมูล

ข้อมูลเวลาที่ได้จากการทดลอง 10 ครั้ง ทั้งการทดลองบนหน่วยประมวลผลกับหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกจะถูกนำมาหาค่าเฉลี่ยเพื่อเปรียบเทียบความเร็วของทั้งสองหน่วยประมวลผล โดยจะนำเวลาเฉลี่ยที่ได้จากหน่วยประมวลผลกลางมาหารด้วยเวลาเฉลี่ยที่ได้จากหน่วยประมวลผลกราฟิก ดังนั้น ผลท้ายสุดของการเปรียบเทียบเวลาจะ



ออกมาอยู่ในรูปจำนวนเท่า มากกว่านั้นการเปรียบเทียบ จะรวมถึงการเปรียบเทียบความเร็วในแต่ละระดับประชากร ในการทดลองนี้เราได้ใช้ตัวอย่างประชากรที่ 500 และ 1000 ซึ่งเพียงพอสำหรับการแสดงความแตกต่างของความเร็วในการประมวลผลของทั้งสองหน่วยประมวล

การนำวิธีการแก้ปัญหาใดๆ ไปประยุกต์จากการประมวลผลแบบลำดับต่อเนื่องไปเป็นการประมวลผลแบบขนาน สิ่งที่ไม่เปลี่ยนแปลงคือลักษณะของการหาคำตอบ คำตอบที่ได้ควรมีค่าที่ใกล้เคียงกัน สำหรับ วิธีการปฏิบัติการร่วมกัน ค่าที่เหมาะสมที่ได้มาจากการประมวลผลจากการประมวลผลทั้งสองแบบต้องมีค่าใกล้เคียงกัน และไม่ห่างจากค่าที่ดีที่สุดของปัญหา

### ผลการทดลอง

จากการทดลองโปรแกรมทั้งบนหน่วยประมวลผลกลางและบนหน่วยประมวลผลกราฟิกโดยใช้ฟังก์ชันวัดเวลาอันเดียวกัน ผลการทดลองสามารถแสดงดังตาราง 4-1

ปัญหา	ประชากร	หน่วยประมวลผล	เวลาเฉลี่ย (วินาที)	ความเร็ว(เท่า)
grostel24	500	CPU	2.360	7.08
		GPU	0.333	
	1000	CPU	4.672	8.74
		GPU	0.534	
grostel48	500	CPU	6.017	6.85
		GPU	0.878	
	1000	CPU	12.815	9.50
		GPU	1.349	
pr76	500	CPU	14.121	6.48
		GPU	2.179	
	1000	CPU	29.901	7.79
		GPU	3.835	
kroA100	500	CPU	18.776	5.31
		GPU	3.536	
	1000	CPU	41.662	6.57
		GPU	6.338	

ตารางที่ 4-1 เวลาที่ใช้ในการประมวลผลหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกและจำนวนเท่าของความเร็วของหน่วยประมวลผลกราฟิกต่อหน่วยประมวลผลกลาง

ตารางที่ 4-1 เปรียบเทียบเวลาที่ใช้ไปในการทดลองโปรแกรมระหว่างหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกโดยใช้พารามิเตอร์เหมือนกัน โปรแกรมทำงานเป็นจำนวน 200 รอบหรือ 200 รุ่น จำนวนรุ่นซึ่งเป็นพารามิเตอร์ตัวหนึ่งของขั้นตอนวิธีพันธุกรรม ตัวเลข 200 ที่ใช้มิได้มีผลต่อการเพิ่มขึ้นหรือลดลงของอัตราส่วนความเร็วในการทำงานของโปรแกรมเมื่อเปรียบเทียบทั้งสองหน่วยประมวลผล เวลาที่ใช้จะเพิ่มขึ้นแปรผันตรงต่อจำนวนรอบ ตัวเลขที่นำมาใช้นี้เป็นจำนวนรอบที่ขั้นตอนวิธีปฏิบัติการร่วมกันสามารถให้ค่าที่ดีที่สุดสำหรับคำตอบของปัญหาการเดินทางของนักศึกษาที่ 24 เมืองคือ 1272

ความเร็วที่หน่วยประมวลผลกราฟิกสามารถทำได้เร็วกว่าหน่วยประมวลผลกลางนั้นจะเพิ่มขึ้นเมื่อเพิ่มจำนวนประชากรซึ่งเป็นพารามิเตอร์หนึ่งของขั้นตอนวิธีปฏิบัติการร่วมกัน

สำหรับการประเมินว่าขั้นตอนวิธีปฏิบัติการร่วมกันสามารถประยุกต์ไปทำงานแบบขนานบนหน่วยประมวลผลกราฟิกได้หรือไม่นั้น สำหรับคำตอบที่ได้มีการสุ่มเข้ามาเกี่ยวข้องด้วย ดังนั้นคุณผลของคำตอบที่ได้จากขั้นตอนวิธีปฏิบัติการร่วมกันว่าใกล้เคียงกับผลที่ได้จากการทำงานบนหน่วยประมวลผลกลางหรือไม่ โดยค่าเฉลี่ยนั้นไม่ควรห่างกันมาก

ปัญหา	หน่วยประมวลผล	จำนวนประชากร			
		500		1000	
		ดีที่สุด	เฉลี่ย	ดีที่สุด	เฉลี่ย
Grostel24	CPU	1272	1318	1272	1291
	GPU	1272	1283	1272	1275
Grostel48	CPU	5648	5975	5606	5909
	GPU	5414	5529	5170	5379
Padberg/Rinaldi 76	CPU	135218	142694	124292	134268
	GPU	137041	143889	131592	135548
kroA100	CPU	38698	39950	35616	38417
	GPU	36127	37309	33065	34172

ตารางที่ 4-2 คำตอบที่ได้จากขั้นตอนวิธีปฏิบัติการร่วมกันบนหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิก

จากตารางที่ 4-2 สังเกตได้ว่าคำตอบที่ได้จากขั้นตอนวิธีอุปติการณ์ร่วมกันบนหน่วยประมวลผลกราฟิกนั้นมีค่าที่ไม่แตกต่างจากหน่วยประมวลผลกลางอย่างมีนัยยะอีกทั้งที่จำนวนเมือง 24 เมืองยังให้ค่าที่เป็นคำตอบที่ดีที่สุดคือ 1272

จำนวนประชากร ขั้นตอน	500				1000			
	CPU		GPU		CPU		GPU	
กำเนิดประชากร	0.0207	67.94%	0.0025	54.19%	0.0432	67.94%	0.0034	45.88%
ประเมินค่าความเหมาะสม	0.0003	0.93%	0.0001	2.94%	0.0005	0.93%	0.0003	3.72%
จัดเรียงประชากร	0.0007	2.19%	0.0007	13.44%	0.0011	2.19%	0.0011	14.41%
ปรับปรุงตัวกำเนิด	0.0088	28.94%	0.0013	29.43%	0.0177	28.94%	0.0027	36.00%
รวมเวลา	0.0305		0.0045		0.0624		0.0074	

ตารางที่ 4-3 เวลาที่ใช้ไปในแต่ละขั้นตอน เปรียบเทียบบนหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิก ที่ 48 เมืองและจำนวนประชากร 500 และ 1000

เมื่อพิจารณาตารางที่ 4-3 พบว่าขั้นตอนที่ใช้เวลาทำงานมากที่สุดคือการกำเนิดประชากร ซึ่งใช้เวลา 67.94%, 54.19% และ 67.94% และ 53.51% ของเวลาที่ใช้ในประมวลผลทั้งหมดที่หน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกที่ประชากร 500 และ 1000 ตามลำดับ และงานที่ใช้เวลาเป็นอันดับสอง คือ งานปรับปรุงตัวกำเนิด ใช้เวลาเป็น 28.94%, 29.43% และ 29.43% ที่หน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิก ที่จำนวนเมือง 48 เมือง ซึ่งหากพิจารณาแล้วถ้าสามารถลดเวลาของการกำเนิดประชากรและปรับปรุงตัวกำเนิดได้มากเท่าไรจะเพิ่มความเร็วได้เท่านั้น

ขั้นตอน	ตามทฤษฎี	ประชากร 500	ประชากร 1000	ประชากร 1500	ประชากร 2000
กำเนิดประชากร	66.7%	16%	32.4%	47.9%	61.2%
ประเมินค่าความเหมาะสม	100%	16.3%	31.9%	46%	61%
ปรับปรุงตัวกำเนิด (1)	33.3%	26.8%	26.8%	26.8%	26.8%
ปรับปรุงตัวกำเนิด (2)	100%	74.1%	74.3%	74.4%	74.5%

ตารางที่ 4-4 เปรอ์เซ็นต์การใช้งานหน่วยประมวลผลแต่ละขั้นตอน ที่ประชากร 500, 1000, 1500 และ 2000

จากการวัดเปอร์เซ็นต์การใช้งานหน่วยประมวลผลของอุปกรณ์ประมวลผลกราฟิกซึ่งได้จากการทดสอบบน NVIADIA visual profiler ซึ่งเป็นเครื่องมือที่อยู่ในชุดพัฒนาโปรแกรม CUDA เมื่อทำการประมวลผลโดยให้แสดงผลลัพธ์จากการวิเคราะห์การใช้งานหน่วยประมวลผล สังเกตได้ว่าเปอร์เซ็นต์เพิ่มขึ้นเรื่อยๆ เมื่อเพิ่มจำนวนประชากรทั้งในขั้นตอนกำเนิดประชากรและประเมินค่าความเหมาะสม ในขณะที่ในขั้นตอนปรับปรุงตัวกำเนิดที่แบ่งเป็นขั้นตอนย่อยสองขั้นตอนไม่มีการใช้งานหน่วยประมวลผลเพิ่มขึ้นเลย ถือได้ว่าค่อนข้างคงที่

จากตารางที่ 4-4 จึงสอดคล้องกับตารางที่ 4-3 ที่ซึ่งความเร็วที่เพิ่มขึ้นเมื่อเพิ่มจำนวนประชากรนั้นได้มาจากขั้นตอนกำเนิดประชากรซึ่งใช้เวลาในการประมวลผลมากที่สุด ดังที่ได้กล่าวไว้ว่าหากลดเวลาในส่วนของการกำเนิดประชากรได้มากเท่าไรยิ่งมีส่วนในการลดเวลารวมในการประมวลผลทั้งหมดได้มากขึ้นเท่านั้น การลดของเวลาในขั้นตอนกำเนิดประชานั้นเป็นเพราะยิ่งเพิ่มจำนวนประชากรมากขึ้นการใช้หน่วยความจำยิ่งมากขึ้นตามลำดับ ทำให้เวลาที่หน่วยประมวลผลหยุดทำงาน (idle) น้อยลง หน่วยประมวลผลทำงานเต็มเวลามากขึ้นได้งานที่มากขึ้น

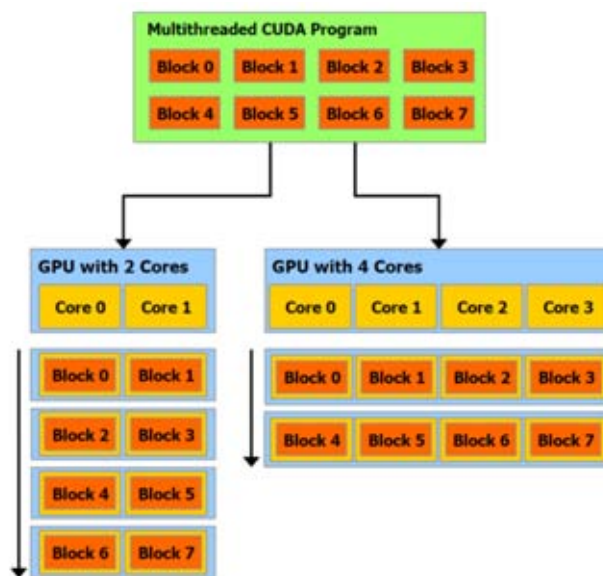
การเพิ่มขึ้นของเปอร์เซ็นต์การใช้งานหน่วยประมวลผลในขั้นตอนประเมินค่าความเหมาะสมนั้นเพิ่มขึ้นดังเช่นขั้นตอนการกำเนิดประชากร แต่ไม่ได้ทำให้เวลารวมลดลงมากนัก เนื่องจาก การประมวลผลในขั้นตอนประเมินค่าความเหมาะสมนั้นถือว่าใช้น้อยมากเมื่อเทียบกับขั้นตอนอื่น เช่น ขั้นตอนการกำเนิดประชากร

## วิเคราะห์การทดลอง

เพื่อเพิ่มความเข้าใจเกี่ยวกับความเร็วที่เพิ่มขึ้นบนหน่วยประมวลผลกราฟิก จะวิเคราะห์หลักการทำงานและเวลาที่ใช้ไปในแต่ละขั้นตอนเปรียบเทียบระหว่างหน่วยประมวลผลกราฟิกและหน่วยประมวลผลกลาง

### การวิเคราะห์จากหลักการทำงาน

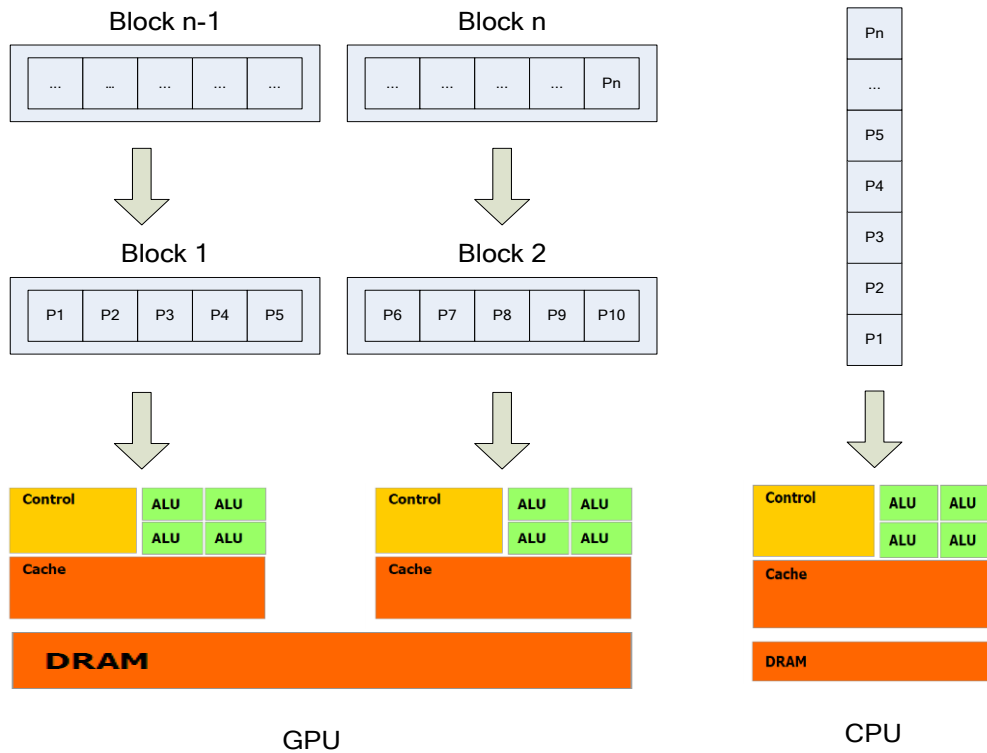
จากผลการทดลองเห็นได้ว่าจำนวนประชากรมีผลกับความเร็วในการประมวลผลของทั้งหน่วยประมวลผลกลางและหน่วยประมวลผลกราฟิกที่แตกต่างกัน



รูปที่ 4-4 แสดงการจัดแบ่งบล็อกเข้าไปประมวลผลในแกนประมวลผลของหน่วยประมวลผลกราฟิก

เพื่อให้เห็นความเกี่ยวข้องของความเร็วที่เพิ่มขึ้นเมื่อจำนวนประชากรเพิ่มขึ้น พิจารณาการทำงานของหน่วยประมวลผลกราฟิกซึ่งมีการจัดเรียงการทำงานของแต่ละบล็อกที่แบ่งไว้ก่อนแล้วเข้าไปยังแกนประมวลผลแต่ละแกนและทำการประมวลผลพร้อมๆกันเมื่อเปรียบเทียบการทำงานของหน่วยประมวลผลกลางที่ทำงานของแต่ละขั้นตอนเป็นลำดับ ดังนั้น เวลาที่ใช้ในการกำเนิดประชากรแต่ละตัวบนหน่วยประมวลผลกลางจะแปรผันต่อจำนวนประชากร ขณะที่บนหน่วยประมวลผลกราฟิกนั้นจะไม่แปรผันตรงต่อจำนวนประชากรแต่จะขึ้นอยู่กับจำนวนแกนประมวลผลของอุปกรณ์ประมวลผลแต่ละรุ่น รูปที่ 4-4 การทำงานของหน่วยประมวลผล

กราฟิกจะจัดเรียงบล็อกเข้าไปประมวลผลตามจำนวนแกนประมวลผลที่มี ดังนั้นหากว่าแกนประมวลผลในอุปกรณ์ประมวลผลกราฟิกมีมากจะเกิดการประมวลผลแบบขนานพร้อมกันเป็นการเพิ่มความเร็ในการประมวลผลใช้เวลาไม่แปรผันกับจำนวนบล็อก



รูปที่ 4-5 ตัวอย่างเปรียบเทียบงานของการกำเนิดประชากร P1 ถึง Pn ของขั้นตอนการกำเนิดประชากรระหว่างหน่วยประมวลผลกราฟิกและหน่วยประมวลผลกลาง

เราสามารถอธิบายโดยยกตัวอย่างงานในการกำเนิดประชากร จากรูปที่ 4-5 งานในการกำเนิดประชากร P1 ถึง P5 อยู่ภายในบล็อกเดียวกัน และงาน P6 ถึง P10 ซึ่งอยู่ในบล็อกที่ 2 สามารถถูกประมวลผลพร้อมกันโดยใช้แกนประมวลผลแยกกันเป็นอิสระ ในขณะที่หน่วยประมวลผลกลางงาน P2 สามารถใช้หน่วยประมวลผลเมื่อเสร็จงาน P1 แล้ว จากรูปที่ 4-5 เป็นเพียงตัวอย่างเมื่อพิจารณาว่าหน่วยประมวลผลกลางมีแกนประมวลผลเดียว ในขณะที่ในปัจจุบันหน่วยประมวลผลกลางมีจำนวนแกนประมวลผลเพิ่มขึ้น แต่เช่นกันจำนวนแกนหน่วยประมวลผลกราฟิกก็เพิ่มมากกว่าและมากขึ้นเรื่อยๆ

จากการอธิบายเกี่ยวกับการประมวลผลแบบขนานด้วยแกนประมวลผลกราฟิกซึ่งมีจำนวนมากกว่าจำนวนแกนประมวลผลของหน่วยประมวลผลกลาง เช่น ในงานวิจัย จำนวนแกน

ประมวลผลของ NVIDIA GeForce 540M นั้นมีจำนวน 96 แกนประมวลผล ที่ความเร็วนาฬิกา 1344 MHz ดังนั้น จากการออกแบบเช่นรูปที่ 4-5 เมื่อมีจำนวนประชากรเพิ่มมากขึ้นจะทำให้เวลาที่ถูกใช้ไม่แปรผันกับจำนวนประชากรเช่นในหน่วยประมวลผลกลาง แต่แปรผันกับจำนวนแกนประมวลผล

## บทที่ 5

### สรุปผลการวิจัย และแนวทางในการพัฒนาต่อ

#### สรุปผลการวิจัย

งานวิจัยนี้นำเสนอการพัฒนาวิธีขั้นตอนปฏิบัติการร่วมกันให้ประมวลผลแบบขนานบนหน่วยประมวลผลกราฟิก และทำการเปรียบเทียบความเร็วหน่วยประมวลผลทั้งสองแบบ จากผลการทดลองทั้งหมดเราได้ทราบว่าการทำงานแบบขนานบนหน่วยประมวลผลกราฟิกของขั้นตอนวิธีปฏิบัติการร่วมกันนั้นทำงานได้เร็วกว่าบนหน่วยประมวลผลกลาง ซึ่งจำนวนเท่าของความเร็วนั้นเพิ่มขึ้นเมื่อจำนวนประชากรที่เพิ่มขึ้น เป็นผลมาจากสถาปัตยกรรมของหน่วยประมวลผลกราฟิกที่ทำมาเพื่อการประมวลผลแบบขนานและการประมวลผลปริมาณมาก อีกทั้งการออกแบบการโปรแกรมของขั้นตอนวิธีปฏิบัติการร่วมกันเสียใหม่เพื่อให้ทำงานแบบขนานบนหน่วยประมวลผลกราฟิก

#### แนวทางในการพัฒนาต่อ

อย่างที่ได้อธิบายในการวิเคราะห์ผลการทดลอง เวลาส่วนใหญ่ได้ถูกใช้ไปในขั้นตอนการกำเนิดประชากร ซึ่งถ้าเราสามารถลดเวลาในส่วนนี้ได้มากเท่าไรก็หมายถึงสามารถเพิ่มความเร็วของขั้นตอนวิธีปฏิบัติการร่วมกันบนหน่วยประมวลผลกราฟิกให้ยิ่งมากขึ้น การออกแบบการทำงานแบบขนานโดยพิจารณาการใช้หน่วยความจำที่นำมาใช้ไม่ว่าจะเป็นหน่วยความจำคงที่ หน่วยความจำร่วมกันซึ่งให้ความเร็วในการเข้าถึงที่แตกต่างกันมาร่วมด้วยในการออกแบบเพิ่มเติมอาจสามารถเพิ่มความเร็วมากยิ่งขึ้น