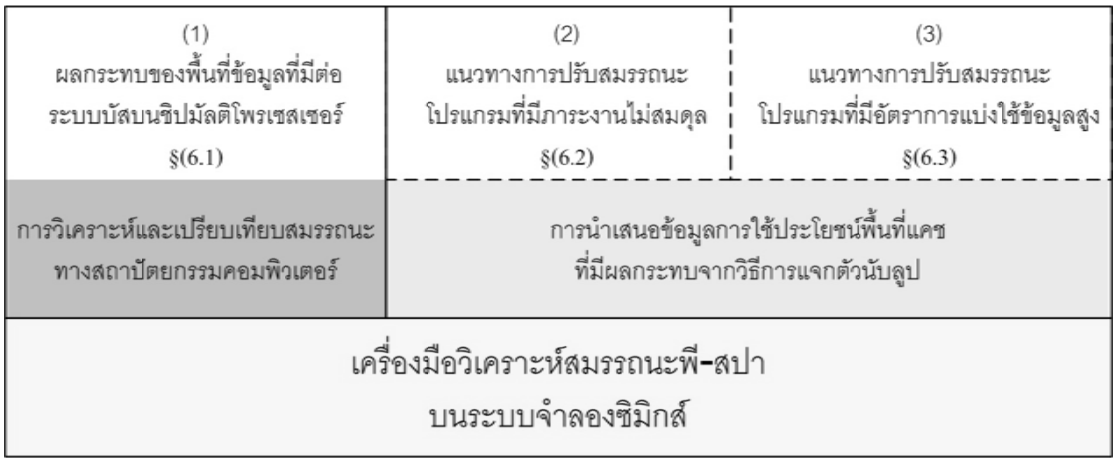
บทที่ 6

แนวทางวิเคราะห์และปรับสมรรถนะของโปรแกรมโอเพนเอ็มพีโดยใช้พี-สปา

วิทยานิพนธ์เพื่อนำเสนอเครื่องมือวิเคราะห์สมรรถนะพี-สปา ซึ่งพัฒนาบนระบบจำลองนี้ เกิดขึ้นจากแนวคิดต่อยอดของงานวิจัยที่เกี่ยวข้องที่ได้กล่าวถึงในบทที่ 3 การวิเคราะห์สมรรถนะบนระบบจำลองทำให้มีข้อได้เปรียบสองประการ คือ (1) การวิเคราะห์สมรรถนะบนระบบคอมพิวเตอร์เป้าหมายที่ยังไม่ได้พัฒนาใช้จริง หรือการเปรียบเทียบสมรรถนะระบบคอมพิวเตอร์ที่มีรายละเอียดทางสถาปัตยกรรมต่างกัน เนื่องจากระบบจำลองอนุญาตให้ผู้ใช้ปรับค่าพารามิเตอร์ทางสถาปัตยกรรมตามที่ต้องการได้ และ (2) การนำเสนอข้อมูลการใช้ประโยชน์พื้นที่แคช ที่เป็นผลกระทบจากวิธีการแจกลูปได้ชัดเจนขึ้น เนื่องจากในขณะที่โปรแกรมประมวลผลนั้น ระบบจำลองสามารถเก็บวิธีการทำงานของฮาร์ดแวร์แต่ละส่วนได้ละเอียดขึ้น

ภาพที่ 6.1 ภาพรวมของแนวทางการใช้ประโยชน์เครื่องมือวิเคราะห์พี-สปา



ในบทนี้จะกล่าวถึง การทดลองสามชิ้นเพื่อนำเสนอข้อได้เปรียบของเครื่องมือวิเคราะห์พี-สปา (ภาพที่ 6.1) ซึ่งประกอบด้วย (1) การศึกษาผลกระทบของคุณสมบัติพื้นที่เก็บข้อมูลหรือการเข้าถึงข้อมูล ที่มีต่อระบบบัสบนชิปมัลติโพรเซสเซอร์ ซึ่งมีสถาปัตยกรรมของแคชต่างกัน (2) แนวทางการปรับสมรรถนะโปรแกรมหาค่าจำนวนเฉพาะ ซึ่งมีภาระงานที่ไม่สมดุล และ (3) แนวทางการปรับสมรรถนะฟังก์ชันการคูณเมทริกซ์แบบขนาน ซึ่งมีอัตราการแบ่งใช้ข้อมูลสูง ซึ่งมีรายละเอียดของแต่ละการทดลองดังต่อไปนี้

6.1 ผลกระทบของคุณสมบัติการเข้าถึงข้อมูลที่มีต่อระบบบัสของชิปมัลติโพรเซสเซอร์

ปัญหาสำคัญที่ทำให้ชิปมัลติโพรเซสเซอร์สูญเสียสมรรถนะการประมวลผลคือปัญหาความคับคั่งของการเข้าใช้บัส (Bus saturation) เนื่องจากชิปมัลติโพรเซสเซอร์เป็นระบบคอมพิวเตอร์แบบใช้หน่วยความจำร่วมกัน ซึ่งทุกหน่วยประมวลผลย่อยต้องเข้าถึงข้อมูลบนหน่วยความจำหลักผ่านระบบบัส ปัญหาความคับคั่งของการเข้าใช้บัสทำให้ชิปมัลติโพรเซสเซอร์ถูกจำกัดให้มีจำนวนหน่วยประมวลผลย่อยที่ระบบบัสสามารถรองรับได้ไม่เกิน 16 หน่วย (Villa *et al.*, 2005)

การพัฒนาโปรแกรมให้มีคุณสมบัติการเข้าถึงข้อมูลบนพื้นที่เดิมหรือติดกัน จะช่วยให้ลดปัญหาความคับคั่งของการเข้าใช้บัสได้ เนื่องจากโปรแกรมที่มีคุณสมบัติดังกล่าวจะทำให้หน่วยประมวลผลมีอัตราการพบข้อมูลในแคชที่สูงขึ้นและมีการเข้าถึงข้อมูลบนหน่วยความจำหลักลดลง แม้ว่ามีงานวิจัยของเซา (Zhao *et al.*, 2007) แสดงให้เห็นอัตราการเข้าใช้บัสของแต่ละหน่วยประมวลผล แต่ปัจจุบันยังไม่งานวิจัยที่ชี้ให้เห็นความสัมพันธ์คุณสมบัติการเข้าถึงข้อมูลของโปรแกรมที่ส่งผลต่ออัตราการเข้าใช้บัส การทดลองนี้จึงได้นำเครื่องมือพี-สปาวิเคราะห์คุณสมบัติการเข้าถึงพื้นที่ข้อมูลของโปรแกรม เพื่อใช้สะท้อนอัตราการเข้าใช้บัสของชิปมัลติโพรเซสเซอร์

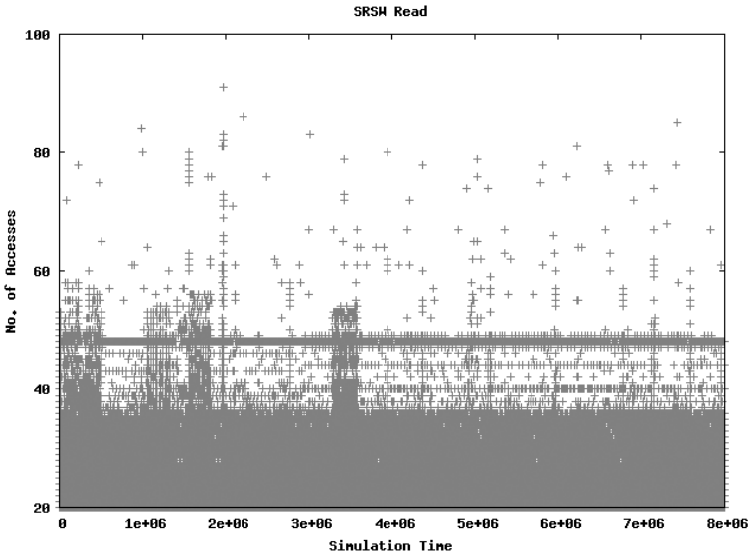
ปัจจุบันชิปมัลติโพรเซสเซอร์ที่มี 2 หน่วยประมวลผลมีการออกแบบแคชระดับที่ 2 ซึ่งเป็นแคชระดับสุดท้ายบนแผงวงจร (On-chip cache) ไว้สองสถาปัตยกรรม คือ ทุกหน่วยใช้แคชร่วมกัน และแต่ละหน่วยใช้แคชแยกกัน ซึ่งประสิทธิภาพการทำงานของแคชระดับสุดท้ายนั้นมีผลต่อการเข้าถึงข้อมูลบนหน่วยความจำผ่านระบบบัส การทดลองนี้ได้นำโปรแกรมเอฟทีหนึ่งในภาระงานหลักของโปรแกรมเอ็นพีบีมาวิเคราะห์คุณสมบัติการเข้าถึงพื้นที่ข้อมูลด้วยพี-สปา และเปรียบเทียบอัตราการไม่พบข้อมูลในแคช เวลาประมวลผล และความต้องการแบนด์วิดท์ของบัสบนแคชทั้งสองสถาปัตยกรรม

6.1.1 คุณลักษณะการเข้าถึงข้อมูลของโปรแกรมเอฟที

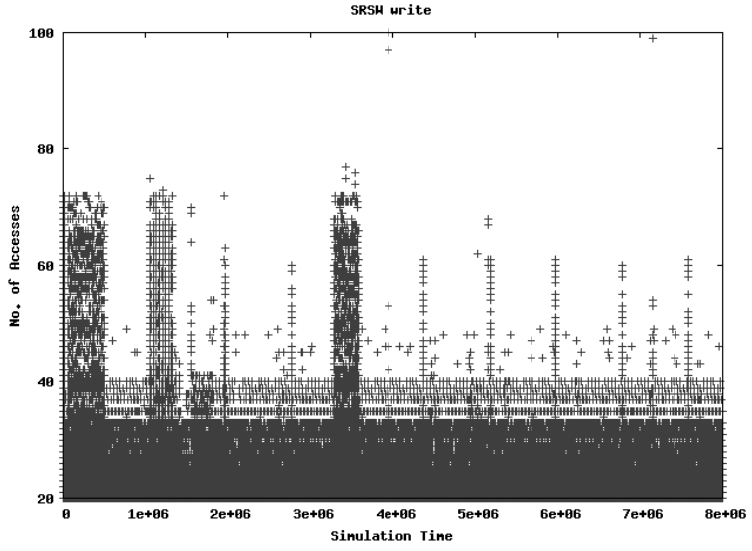
โปรแกรมเอฟทีเป็นหนึ่งภาระหลักของชุดโปรแกรมวัดเปรียบเทียบสมรรถนะเอ็นพีบี ซึ่งใช้คำนวณการแปลงฟูเรียร์แบบเร็วบนชุดข้อมูลสามมิติ (3-D Fast Fourier transform) โปรแกรมเอฟทีที่ใช้ในการทดลองนี้ถูกพัฒนาด้วยเอฟไอของโอเพนเอ็มพีบนภาษาซี ประมวลผลบนชิปมัลติโพรเซสเซอร์ที่มี 2 หน่วยประมวลผลย่อยด้วยจำนวน 2 เทรด โดยประมวลผลโปรแกรมด้วยชุดข้อมูลสามมิติขนาด 64x64x64 คำนวณการแปลงฟูเรียร์ 6 รอบ

ภาพที่ 6.2 จำนวนการเข้าถึงข้อมูลบนเซกเมนต์รูปแบบมีหนึ่งหน่วยย่อยอ่านและเขียนในแต่ละช่วงเวลา

(a) จำนวนการอ่าน



(b) จำนวนการเขียน



จากการวิเคราะห์รายละเอียดการเข้าถึงข้อมูลของหน่วยประมวลผลที่ส่งไปยังแคชด้วยพี-สปา พบว่า การเข้าถึงพื้นที่ข้อมูลในระดับเซกเมนต์ของโปรแกรมเอฟทีมีการเข้าถึงข้อมูลบนเซกเมนต์แบบ SRSW สูงถึงร้อยละ 69.61 ซึ่งจากรูปแบบดังกล่าวแสดงถึงทั้งสองหน่วยย่อยมีการเข้าถึงข้อมูลในแต่ละเซกเมนต์ในช่วงเวลาที่ไม่ตรงกัน ทำให้ไม่เกิดปัญหาแย่งใช้ข้อมูลชุดเดียวกัน อย่างไรก็ตามรูปแบบของเซกเมนต์ในอันดับรองลงมา 3 ลำดับ (MRSW, SRMW และ MRMW) ซึ่งเป็นรูปแบบของเซกเมนต์ที่ถูกเข้าถึงจากหลายหน่วย ทำให้มีโอกาสที่แต่ละหน่วยย่อย

เข้าใช้ข้อมูลร่วมกันมีสัดส่วนรวมกันร้อยละ 23.30 อาจทำให้เกิดการไม่พบข้อมูลในแคชเนื่องจากใช้ข้อมูลร่วมกันได้

เมื่อพิจารณาจำนวนการอ่าน (ภาพที่ 6.2 (a)) และการเขียน (ภาพที่ 6.2 (b)) บนเซกเมนต์แบบ SRSW นั้น จะเห็นได้ว่าในแต่ละช่วงเวลาเซกเมนต์จะถูกอ่านและเขียนมีค่าเฉลี่ยที่ใกล้เคียงกัน เนื่องจากแต่ละหน่วยย่อยเข้าถึงข้อมูลในลักษณะเพื่ออ่าน-แก้ไข-เขียน (Read-Modified-Write) คุณลักษณะดังกล่าวแสดงถึงคุณสมบัติการเข้าถึงข้อมูลพื้นที่เดิมอยู่ตลอดเวลา ซึ่งจะส่งผลให้มีอัตราการพบข้อมูลในแคชสูงขึ้น จากผลวิเคราะห์สรุปได้ว่าโปรแกรมเอฟทีมีคุณสมบัติการเข้าพื้นที่ข้อมูลพื้นที่เดิมบ่อยครั้ง และมีการเข้าถึงข้อมูลบนเซกเมนต์แบบ SRSW ทำให้ไม่เกิดปัญหาการใช้ข้อมูลชุดเดียวกัน นอกจากนี้โปรแกรมเอฟทียังแสดงมีแนวโน้มที่ทำให้เกิดอัตราพบข้อมูลบนในแคชที่สูงด้วย

ตารางที่ 6.1 สถาปัตยกรรมของชิปมัลติโพรเซสเซอร์ที่ใช้วิเคราะห์คุณสมบัติการเข้าถึงข้อมูล

สถาปัตยกรรมของชิปมัลติโพรเซสเซอร์	สถาปัตยกรรมแบบใช้แคชร่วมกัน	สถาปัตยกรรมแบบใช้แคชแยกกัน
สถาปัตยกรรมของหน่วยประมวลผล		
- จำนวนหน่วยประมวลผล	2 หน่วย	
- ชุดคำสั่ง	x86_64	
สถาปัตยกรรมของแคชระดับที่ 1		
- ขนาดของแคชระดับที่ 1 (แยกแคชคำสั่งและข้อมูล)	32 KB / private	
- ขนาดของแคชบล็อก	64 bytes	
- จำนวนเซต	8 way	
- นโยบายการเขียน	write through, non-allocate	
- เวลาการเข้าถึง	3 cycles	
สถาปัตยกรรมของแคชระดับที่ 2		
- ขนาดของแคชระดับที่ 2	**2 MB / shared**	**2 x 1 MB / private**
- ขนาดของแคชบล็อก	64 bytes	
- จำนวนเซต	8 way	
- นโยบายการเขียน	write-back, allocate	
- เวลาการเข้าถึง	10 cycles	
- เวลาการเข้าถึงในกรณีไม่พบข้อมูล	165 cycles	
กลไกรักษาความสอดคล้องของข้อมูลในแคช	**ไม่มี**	**MESI แคชระดับที่ 2**
ความจุของหน่วยความจำหลัก	2 GB	

6.1.2 วิธีทำการทดลอง

ดังที่กล่าวไว้ในหัวข้อ 6.1 ชิปมัลติโพรเซสเซอร์ที่มี 2 หน่วยประมวลผลมีการออกแบบแคชระดับที่ 2 ไว้สองสถาปัตยกรรม คือ แบบใช้แคชร่วมกันและแยกกัน การทดลองนี้จึงได้เปรียบเทียบสมรรถนะของแคชทั้งสองสถาปัตยกรรม เมื่อประมวลผลโปรแกรมเอฟทีบนระบบปฏิบัติการลินิกซ์และเก็บข้อมูลด้วยพี-สปา โดยพิจารณาเปรียบเทียบ (1) อัตราการไม่พบข้อมูลในแคชระดับที่ 1 และ 2 ที่เกิดขึ้น (2) เวลาประมวลผล และ (3) ความต้องการแบนด์วิดท์ที่ใช้ในการประมวลผลของแต่ละสถาปัตยกรรม การทดลองนี้เปรียบเทียบผลการทดลองด้วยการวิเคราะห์สถิติแบบจำแนกทางเดียว แต่เนื่องจากข้อจำกัดที่ต้องใช้เวลาการประมวลผลบนพี-สปาเป็นเวลานานกว่าสี่ชั่วโมง การทดลองนี้จึงใช้ค่าเฉลี่ยผลการทดลองจากการประมวลผลเพียงสามครั้ง ตารางที่ 6.1 แสดงคุณลักษณะของชิปมัลติโพรเซสเซอร์ของทั้งสองสถาปัตยกรรม ซึ่งถูกจำลองการทำงานบนระบบจำลองซิมิกส์

6.1.3 ผลการทดลอง

ในหัวข้อนี้กล่าวถึงผลการทดลองซึ่งจะแบ่งออกเป็นสามส่วน คือ (1) อัตราการไม่พบข้อมูลในแคชทั้งสองระดับ (2) เวลาประมวลผล และ (3) ความต้องการแบนด์วิดท์บนระบบบัสของแต่ละสถาปัตยกรรม

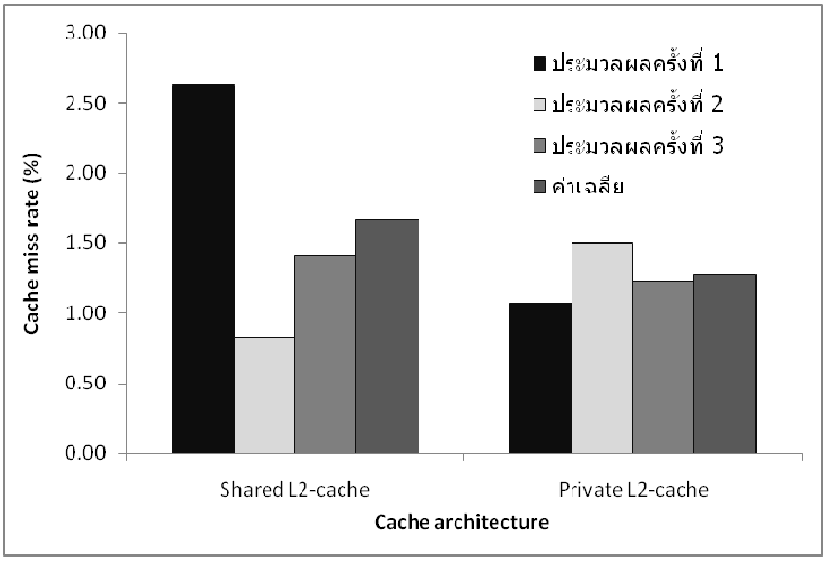
6.1.3.1 อัตราการไม่พบข้อมูลในแคชระดับที่ 1 และ 2

ผลการทดลองเปรียบเทียบอัตราการไม่พบข้อมูลในแคชระดับที่ 1 (ภาพที่ 6.3) และจากการวิเคราะห์ทางสถิติ พบว่า สถาปัตยกรรมทั้งสองแบบมีค่าเฉลี่ยของอัตราการไม่พบข้อมูลไม่แตกต่างกันอย่างมีนัยสำคัญ ซึ่งจากการวิเคราะห์โปรแกรมเอฟทีเห็นได้ว่ามีการเข้าถึงข้อมูลบนเซกเมนต์เป็นแบบ SRSW ทำให้ข้อมูลถูกใช้เพียงหน่วยย่อยเดียวบ่อยครั้ง จึงให้เกิดอัตราการไม่พบข้อมูลในแคชต่ำ

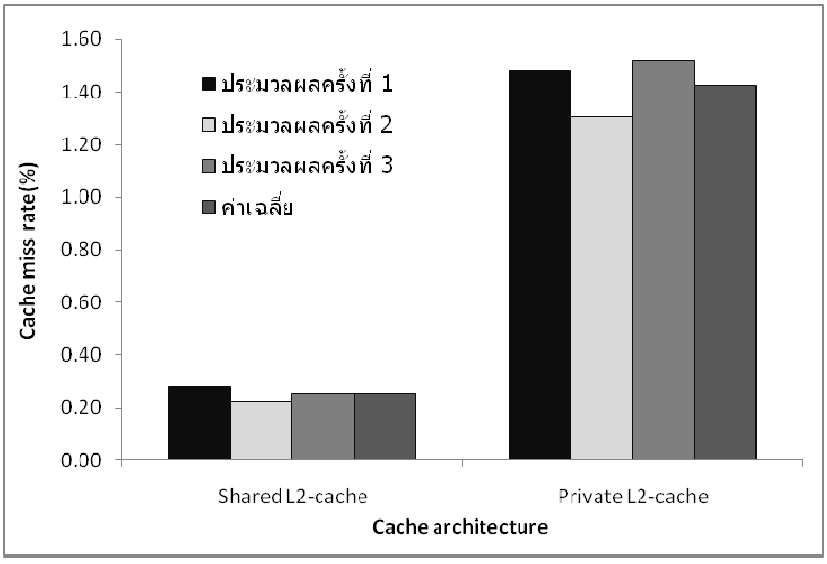
อย่างไรก็ตามเมื่อเปรียบเทียบอัตราการไม่พบข้อมูลในแคชระดับที่ 2 (ภาพที่ 6.4) และจากการวิเคราะห์ทางสถิติ พบว่า สถาปัตยกรรมแบบใช้แคชร่วมกันมีค่าเฉลี่ยอัตราการไม่พบข้อมูลน้อยกว่าสถาปัตยกรรมแบบใช้แคชแยกกันร้อยละ 1.17 อย่างมีนัยสำคัญ ที่ระดับความเชื่อมั่นร้อยละ 95 ทั้งนี้สาเหตุที่ทำให้เกิดอัตราการไม่พบข้อมูลบนแคชระดับที่ 2 ของ

สถาปัตยกรรมแบบใช้แคชแยกกันสูงกว่าอาจเกิดจากสองสาเหตุ คือ การแบ่งพื้นที่แคชระดับที่ 2 ทำให้ต้องมีการเก็บสำเนาข้อมูลซ้ำซ้อนส่งผลให้สูญเสียพื้นที่การเก็บข้อมูล และการเข้าถึงข้อมูลในเซกเมนต์แบบ MRSW, SRMW และ MRMW ของสถาปัตยกรรมแบบใช้แคชแยกกันอาจทำให้เกิดปัญหาความไม่สอดคล้องของข้อมูลในแคช ทำให้ข้อมูลที่เก็บไว้ถูกยกเลิกไป ในขณะที่ในสถาปัตยกรรมแบบใช้แคชร่วมกันจะไม่เกิดปัญหาดังกล่าว

ภาพที่ 6.3 อัตราการไม่พบข้อมูลในแคชระดับที่ 1 ของชิปมัลติโพรเซสเซอร์ทั้งสองสถาปัตยกรรม



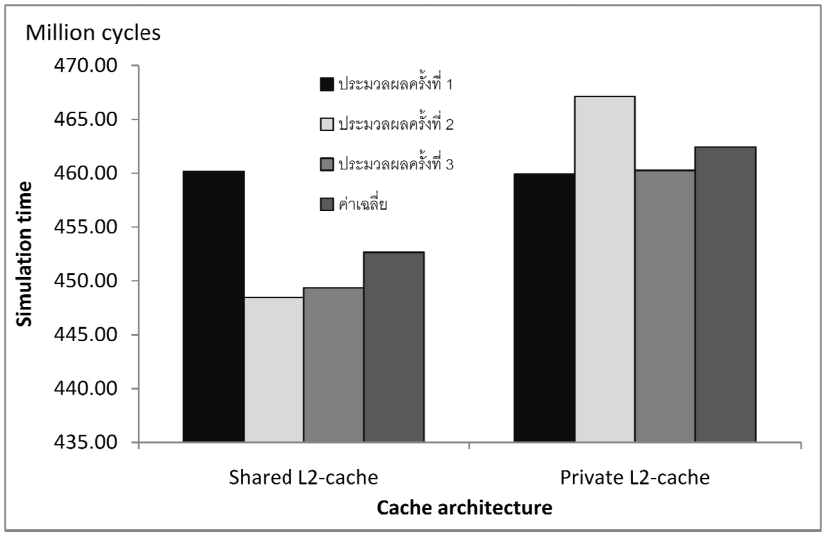
ภาพที่ 6.4 อัตราการไม่พบข้อมูลในแคชระดับที่ 2 ของชิปมัลติโพรเซสเซอร์ทั้งสองสถาปัตยกรรม



6.1.3.2 เวลาประมวลผล

ผลการทดลองเปรียบเทียบเวลาประมวลผล (ภาพที่ 6.5) และจากการวิเคราะห์ทางสถิติ พบว่า สถาปัตยกรรมทั้งสองแบบมีค่าเฉลี่ยของอัตราการไม่พบข้อมูลไม่แตกต่างกันอย่างมีนัยสำคัญ จากผลการทดลองนี้แม้ว่าผลของอัตราการไม่พบข้อมูลในแคชระดับที่ 2 ของสถาปัตยกรรมแบบใช้แคชแยกกันจะส่งให้สูญเสียเวลาประมวลผล แต่จากรูปที่ 4 การใช้แคชแยกกันก็มีแน้วโน้มจะเวลาประมวลผลที่ช้ากว่าการใช้แคชระดับสุดท้ายร่วมกัน และผลการเกิดอัตราไม่พบข้อมูลในแคชระดับสุดท้ายนั้นจะทำให้ต้องการแบนด์วิดท์บนบัสที่มีขนาดใหญ่กว่า ซึ่งจะได้กล่าวถึงในหัวข้อถัดไป

ภาพที่ 6.5 จำนวนรอบสัญญาณนาฬิกาที่ใช้ประมวลผลของซิปมัลติโพรเซสเซอร์ทั้งสองสถาปัตยกรรม



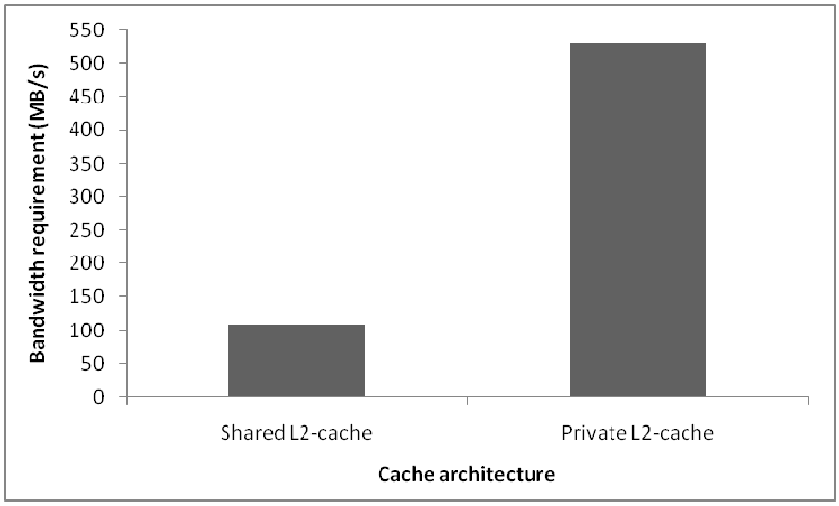
6.1.3.3 ความต้องการแบนด์วิดท์

ความต้องการแบนด์วิดท์ของสถาปัตยกรรมทั้งสองแบบในแต่ละหน่วยย่อยนั้น จะคำนวณด้วยสมการที่ 6.1 ซึ่งจำนวนรอบต่อคำสั่ง (CPI) ค่าสัดส่วนการเข้าถึงข้อมูล (Data access ratio) และอัตราการไม่พบข้อมูลในแคชระดับที่ 2 (L2 miss rate) ได้จากการวิเคราะห์โปรแกรมเอฟทีด้วยพี-สปา ขณะที่ความเร็วสัญญาณนาฬิกา (Clock rate) และขนาดของข้อมูล (Data size) เป็นคุณสมบัติของแพลตฟอร์ม

$$Bandwidth = \left(\frac{Clock\ rate}{CPI}\right) \times Data\ access\ ratio \times L2\ miss\ rate \times Data\ size \quad \text{สมการที่ 6.1}$$

จากภาพที่ 6.6 ซึ่งแสดงถึงความต้องการแบนด์วิดท์บนบัส พบว่า สถาปัตยกรรมแบบใช้แคชระดับที่ 2 แยกกันมีความต้องการแบนด์วิดท์สูงกว่าของการใช้แคชร่วมกันถึง 4.8 เท่า หากบัสมีแบนด์วิดท์ไม่เพียงต่อความต้องการจะทำให้โปรแกรมสูญเสียสมรรถนะการประมวลผลได้

ภาพที่ 6.6 ความต้องการแบนด์วิดท์ของซิปมัลติโพรเซสเซอร์ทั้งสองสถาปัตยกรรม



6.1.4 สรุปและอภิปรายผลการทดลอง

การทดลองนี้ได้แสดงให้เห็นความสัมพันธ์ระหว่างคุณสมบัติการเข้าถึงข้อมูลบนพื้นที่เดิมหรือติดกันของโปรแกรม กับความต้องการแบนด์วิดท์ของบัสบนซิปมัลติโพรเซสเซอร์ เพื่อให้เห็นถึงความสัมพันธ์ดังกล่าว การทดลองนี้จึงได้นำเครื่องมือพี-สปาวิเคราะห์โปรแกรมเอฟทีที่ใช้คำนวณการแปลงฟูเรียร์แบบเร็ว ผลจากการวิเคราะห์พบว่าโปรแกรมเอฟทีมีคุณสมบัติการเข้าถึงข้อมูลบนพื้นที่เดิม โดยจะมีเพียงหนึ่งหน่วยประมวลผลย่อยเข้าถึงข้อมูลในลักษณะอ่าน-แก้ไข-เขียนบนเซกเมนต์เดิมอยู่ตลอดเวลา ซึ่งทำให้มีโอกาสเกิดการพบข้อมูลในแคชสูงขึ้น

เพื่อให้เห็นผลกระทบการออกแบบของแคชระดับสุดท้ายบนแผงวงจรซิป การทดลองนี้ได้นำโปรแกรมเอฟทีไปประมวลผลเปรียบเทียบบนแคชสองสถาปัตยกรรม ซึ่งจากผลการทดลองพบว่า ซิปมัลติโพรเซสเซอร์แบบใช้แคชร่วมกันแสดงสรรถนะการทำงานที่ดีกว่าแบบใช้แคชแยกกันทั้งในเชิงอัตราการไม่พบข้อมูลบนแคช เวลาประมวลผล และความต้องการแบนดวิดท์บนบัส

6.2 การปรับสมรรถนะโปรแกรมหาค่าจำนวนเฉพาะ ซึ่งมีภาระงานไม่สมดุล

ปัญหาการกระจายภาระงานที่ไม่สมดุล (Load imbalance) เป็นข้อจำกัดเชิงสมรรถนะที่สำคัญของโปรแกรมเชิงขนาน ซึ่งการเลือกวิธีการกระจายภาระงานที่เหมาะสมกับคุณลักษณะของโปรแกรมจะช่วยลดข้อจำกัดดังกล่าวได้ (Ayguadé *et al.*, 2003) แต่ทว่าปัญหาการกระจายภาระงานที่ไม่สมดุลนั้น อาจเกิดจากการเลือกใช้วิธีการกระจายงานที่ไม่เหมาะสมของโปรแกรมเมอร์ หรือเป็นคุณลักษณะเฉพาะของโปรแกรม ซึ่งเป็นลักษณะสามารถปรับปรุงสมรรถนะได้ยาก โปรแกรมหาค่าจำนวนเฉพาะเป็นตัวอย่างหนึ่งที่มีการกระจายภาระงานที่ไม่สมดุลเป็นลักษณะเฉพาะตัว การทดลองนี้จึงใช้เครื่องมือวิเคราะห์พี-สปาวิเคราะห์โปรแกรมหาค่าจำนวนเฉพาะ เพื่อนำเสนอข้อมูลที่เป็นแนวทางการปรับสมรรถนะของโปรแกรมด้วยการเลือกคำสั่งการแจกลูปที่เหมาะสม ซึ่งกล่าวถึงรายละเอียดการทดลองในหัวข้อถัดไป

ภาพที่ 6.7 โปรแกรมหาค่าจำนวนเฉพาะ

```
1:  bool TestForPrime(int val)
2:  {
3:       int limit, factor = 3;
4:       limit = (long)(sqrtf((float)val)+0.5f);
5:       while((factor <= limit) && (val % factor))
6:            factor ++;
7:
8:       return (factor > limit);
9:  }
10: void FindPrimes(int start, int end)
11: {
12:     int range = end - start + 1;
13:
14:     #pragma omp parallel for
15:     for(int i = start; i <= end; i += 2)
16:     {
17:         if(TestForPrime(i))
18:             globalPrimes[gPrimesFound++] = i;
19:     }
20: }
```

6.2.1 คุณลักษณะของโปรแกรมหาค่าจำนวนเฉพาะ

การหาค่าจำนวนเฉพาะเป็นหัวใจสำคัญในการสร้างค่าตัวเลขสุ่มที่มีประสิทธิภาพ การมีจำนวนเฉพาะจำนวนมากจึงเป็นประโยชน์ต่อการพัฒนาโปรแกรมเชิงวิทยาศาสตร์ที่ใช้เทคนิคการสุ่มค่าเป็นหลัก โปรแกรมหาค่าจำนวนเฉพาะที่นำมาทดลองนั้น เป็นโปรแกรมในชุด

ฝึกอบรมการเขียนโปรแกรมบนเครื่องคอมพิวเตอร์แบบมัลติคอร์หรือหลายหน่วยประมวลผลของบริษัทอินเทล ซึ่งมีขั้นตอนการทำงานดังส่วนของโปรแกรมดังแสดงในภาพที่ 6.7

โปรแกรมหลักจะรับขอบเขตของจำนวนที่ต้องการค้นหาว่ามีจำนวนใดบ้างเป็นจำนวนเฉพาะ และนับปริมาณจำนวนเฉพาะที่พบภายในขอบเขตรับเข้ามา โปรแกรมจะวนลูปเริ่มจากจำนวนคี่ตัวเลขไปถึงจำนวนตัวเลขตัวสุดท้ายโดยเลื่อนข้ามเลขคู่ไปทุกครั้งที่วนรอบ การประมวลผลในลูปแต่ละรอบนั้น เป็นการนำตัวนับลูปหรือตัวแปร i ในภาพ ไปตรวจสอบว่าเป็นจำนวนเฉพาะหรือไม่ โดยการเรียกใช้ฟังก์ชัน TestForPrime ซึ่งฟังก์ชันนี้จะนำตัวแปร i ไปทดลองหารด้วยตัวแปร factor ซึ่งมีค่าเริ่มต้นเท่ากับ 3 หากไม่ลงตัวก็จะเพิ่มค่าตัวแปร factor ทีละหนึ่งจนกว่าจะหารลงตัว หรือหารไปจนถึงตัวหารตัวสุดท้ายซึ่งมีค่าเท่ากับตัวแปร limit หากไม่มีเลขที่หารตัวแปร i ลงตัวแสดงว่า ค่าของตัวแปร i ในการตรวจสอบครั้งนั้นเป็นจำนวนเฉพาะ

จากภาพที่ 6.7 จะเห็นว่า ในบรรทัดที่ 14 – 19 เป็นการประมวลผลในลูปที่ใช้คำสั่งแจกลูปโดยปริยาย[1] (Default) ซึ่งการแบ่งด้วยวิธีการดังกล่าวบนสองเทรด เทรดแรกจะหาค่าจำนวนเฉพาะในช่วงข้อมูลครึ่งแรกและเทรดที่สองจะรับช่วงข้อมูลครึ่งหลัง ดังนั้นในการวนลูปเพื่อตรวจสอบจำนวนเฉพาะเทรดแรกจะใช้จำนวนรอบในการวนลูปที่สั้นกว่าเทรดที่สอง จึงเห็นได้ว่าการแจกตัวนับลูปโดยปริยายนั้น ทำให้เทรดที่สองมีซึ่งรับข้อมูลช่วงครึ่งหลังนั้น ประมวลผลมากกว่าเทรดแรก โปรแกรมหาค่าจำนวนเฉพาะนี้จึงมีคุณลักษณะของการแบ่งภาระงานที่ไม่สมดุล และควรปรับปรุงสมรรถนะด้วยการเลือกใช้คำสั่งการแจกลูปที่เหมาะสม ซึ่งจากคุณลักษณะดังกล่าวของโปรแกรม คำสั่งการแจกลูปที่สามารถปรับขนาดได้น่าจะมีความเหมาะสม เนื่องจากการแจกลูปในช่วงเริ่มต้นของการประมวลผลในลูป โปรแกรมอาจจะแจกลูปประมวลผลไปยังแต่ละเทรดเป็นจำนวนมาก และค่อยปรับจำนวนลูปให้ลดลงมาของการประมวลผลช่วงหลัง

6.2.2 วิธีทำการทดลอง

การวิเคราะห์หาปัญหาเชิงสมรรถนะของโปรแกรมหาค่าจำนวนเฉพาะ ได้ดำเนินการสี่ขั้นตอนดังนี้ *ขั้นตอนที่หนึ่ง* เป็นการสร้างระบบคอมพิวเตอร์เป้าหมายบนระบบจำลองซิมิกส์ ซึ่งการทดลองนี้ได้ทดลองบนชิปมัลติโพรเซสเซอร์ขนาดสองหน่วยประมวลผล แบบคอร์ 2 ดูโอ ของ

[1] ในกรณีที่ไม่ระบุคำสั่งการแจกลูปอย่างชัดเจน การประมวลผลในลูปจะถูกด้วยคำสั่ง **static** ที่ไม่กำหนดขนาดของบล็อก

บริษัทอินเทล ที่มีสถาปัตยกรรมแคชระดับที่ 2 แบบใช้ร่วมกัน และค่าสถาปัตยกรรมของแคชดังสรุปในตารางที่ 6.2 จากนั้นประมวลผลโปรแกรมหาค่าจำนวนเฉพาะบนระบบคอมพิวเตอร์เป้าหมาย ซึ่งทำงานบนระบบปฏิบัติการลินิกซ์ โดยใช้จำนวนเทรด 2 เทรด ช่วยกันคำนวณหาค่าจำนวนเฉพาะตั้งแต่ 1 ถึง 20,000 ซึ่งในระหว่างประมวลผลนี้ มีการเก็บค่าพฤติกรรมเกี่ยวกับคุณสมบัติพื้นที่เก็บข้อมูลของเทรดทั้งสองโดยใช้เครื่องมือพี-สปา เพื่อหาโอกาสของการปรับสมรรถนะของโปรแกรม[2]

ตารางที่ 6.2 สถาปัตยกรรมของชิปมัลติโพรเซสเซอร์ที่ใช้วิเคราะห์โปรแกรมหาค่าจำนวนเฉพาะ

สถาปัตยกรรมของชิปมัลติโพรเซสเซอร์	ค่าทางสถาปัตยกรรม
สถาปัตยกรรมของหน่วยประมวลผล	
- จำนวนหน่วยประมวลผล	2 หน่วย
- ชุดคำสั่ง	x86_64
สถาปัตยกรรมของแคชระดับที่ 1	
- ขนาดของแคชระดับที่ 1 (แยกแคชคำสั่งและข้อมูล)	32 KB / private
- ขนาดของแคชบล็อก	64 bytes
- จำนวนเซต	8 way
- นโยบายการเขียน	write-back, allocate
- เวลาการเข้าถึงข้อมูล	3 cycles
สถาปัตยกรรมของแคชระดับที่ 2	
- ขนาดของแคชระดับที่ 2	4 MB / Shared
- ขนาดของแคชบล็อก	64 bytes
- จำนวนเซต	16 way
- นโยบายการเขียน	write-back, allocate
- เวลาการเข้าถึงข้อมูล	10 cycles
- เวลาการเข้าถึงข้อมูลในกรณีไม่พบข้อมูล	165 cycles
ความจุของหน่วยความจำหลัก	4 GB

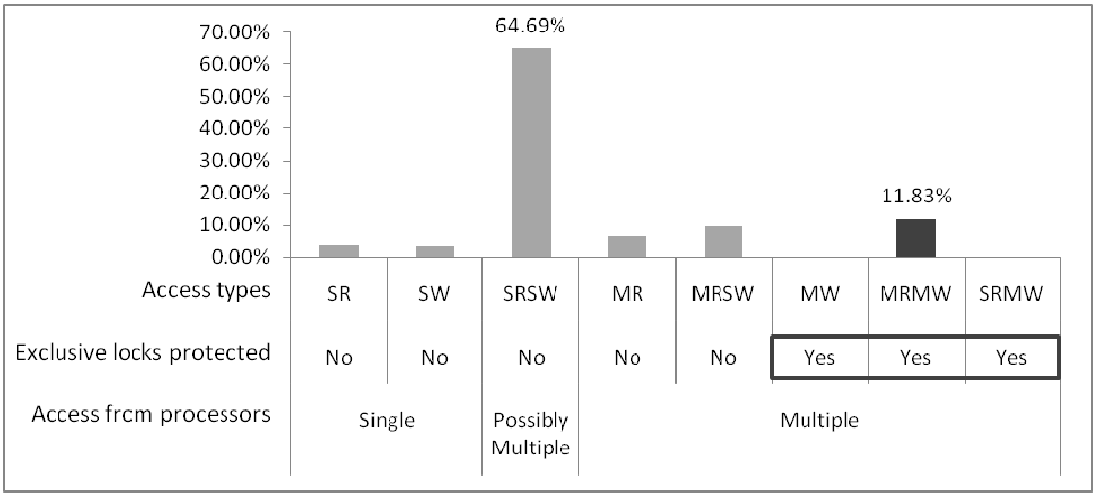
ขั้นตอนที่สอง เป็นการเปรียบเทียบสมรรถนะคำสั่งการแจกลูป โดยเก็บข้อมูลเกี่ยวกับ จำนวนการเข้าถึงข้อมูล ค่าสปีดอัพ อัตราและชนิดของการไม่พบข้อมูลในแคชระดับที่ 1 ในขั้นตอนนี้โปรแกรมหาค่าจำนวนเฉพาะถูกปรับปรุงให้ใช้คำสั่งการแจกลูป 6 รูปแบบ จำแนกตาม

[2] ในการทดลองนี้ได้วิเคราะห์พื้นที่หน่วยความจำระดับแคชบล็อกซึ่งมีขนาดเท่ากับ 64 ไบต์

คำสั่งการแจกลูปที่ต่างกันสามคำสั่ง แต่ละคำสั่งกำหนดขนาดของบล็อกสองวิธี คือ กำหนดขนาดของบล็อกโดยปริยาย และกำหนดจำนวนลูป 16 ลูปต่อบล็อก ซึ่งเป็นจำนวนของลูปที่มีขนาดเท่ากันกับแคชบล็อก โดยสรุปโปรแกรมแต่ละรูปแบบได้ถูกปรับปรุงให้ใช้คำสั่งการแจกลูป ดังนี้ **`static`**, **`static(16)`**, **`dynamic`**, **`dynamic(16)`**, **`guided`** และ **`guided(16)`**[3]

ขั้นตอนที่สาม คือขั้นตอนการนำผลลัพธ์ที่เก็บได้จากพี-สปา มาวิเคราะห์คุณสมบัติพื้นที่เก็บข้อมูลและใช้ประกอบการเปรียบเทียบสมรรถนะของโปรแกรมที่สืบเนื่องมาจากการใช้คำสั่งการแจกลูปแต่ละรูปแบบ และ*ขั้นตอนที่สี่* เป็นการนำผลวิเคราะห์จากพี-สปาไปปรับใช้จริงระบบคอมพิวเตอร์จริง เพื่อยืนยันถึงผลการวิเคราะห์จากพี-สปาที่สามารถนำไปปรับใช้จริงได้

ภาพที่ 6.8 สัดส่วนรูปแบบพื้นที่หน่วยความจำของโปรแกรมหาค่าจำนวนเฉพาะ



6.2.3 ผลการทดลอง

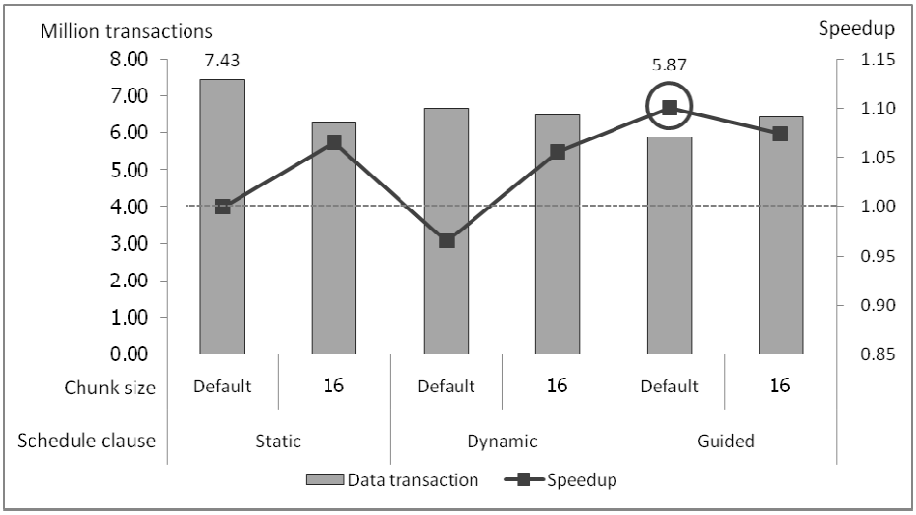
ผลการวิเคราะห์คุณสมบัติพื้นที่เก็บข้อมูลโปรแกรมหาค่าจำนวนเฉพาะของพี-สปา (ภาพที่ 6.8) พบว่า โปรแกรมหาค่าจำนวนเฉพาะมีสัดส่วนของพื้นที่แบบ SRSW สูงที่สุดเท่ากับร้อยละ 69.64 เมื่อพิจารณารูปแบบพื้นที่หน่วยความจำที่ทำให้เกิดการปิดกั้น เนื่องจากมีการแย่งกันเขียนข้อมูลจากหลายหน่วยประมวลผล (MW, MRMW และ SRMW) พื้นที่ดังกล่าวมีสัดส่วนร้อยละ 11.83 ซึ่งสัดส่วนดังกล่าวแสดงถึงโอกาสการปรับปรุงสมรรถนะของโปรแกรมให้ดีขึ้น

---

[3] ในส่วนที่เหลือของวิทยานิพนธ์ฉบับนี้ **`static`**, **`dynamic`** และ **`guided`** หมายถึง คำสั่งกำกับ **`static`**, **`dynamic`** และ **`guided`** ที่ไม่กำหนดขนาดของบล็อก ส่วน **`static(`*`n`*`)`**, **`dynamic(`*`n`*`)`** และ **`guided(`*`n`*`)`** หมายถึง คำสั่งกำกับ **`static`**, **`dynamic`** และ **`guided`** ที่กำหนดขนาดของบล็อกเท่ากับ ***`n`*** ลูป ตามลำดับ

อย่างไรก็ตาม แม้ว่าพื้นที่แบบ SRSW จะไม่ทำให้เกิดการปิดกั้นการเข้าถึงพื้นที่ แต่ก็มีโอกาสทำให้เปลี่ยนรูปแบบเป็นแบบ SRMW ได้ หากมีหน่วยประมวลผลอื่นเข้าถึงพื้นที่ดังกล่าวเพิ่มขึ้น จึงเป็นจุดเสี่ยงทำให้เกิดการปิดกั้นพื้นที่และทำให้โปรแกรมสูญเสียสมรรถนะได้

ภาพที่ 6.9 จำนวนการเข้าถึงข้อมูลและค่าสปีดอัพเมื่อกับคำสั่ง **`static`** ของโปรแกรมหาค่าจำนวนเฉพาะ

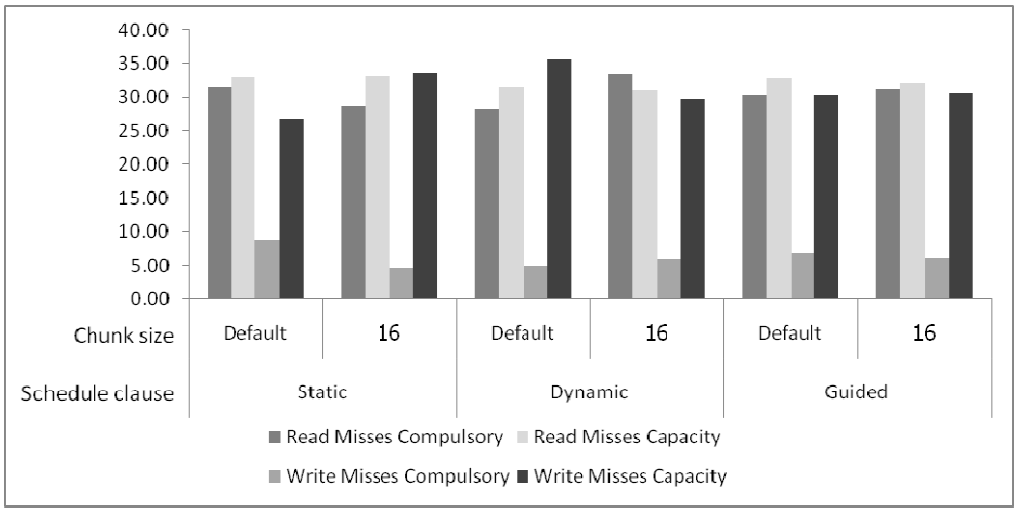


จากการวิเคราะห์คุณสมบัติพื้นที่เก็บข้อมูลของโปรแกรมหาค่าจำนวนเฉพาะ ทำให้ทราบว่าหากโปรแกรมเมอร์เลือกใช้คำสั่งการแจกลูปที่เหมาะสม ก็อาจทำให้โปรแกรมมีสมรรถนะสูงขึ้นประมาณร้อยละ 11.83 ซึ่งผลการเปรียบเทียบจำนวนการเข้าถึงข้อมูล และค่าสปีดอัพเมื่อเทียบกับคำสั่ง **`static`** (ภาพที่ 6.9) พบว่า คำสั่ง **`guided`** มีค่าสปีดอัพสูงที่สุด โดยมีค่าสปีดอัพ 1.10 เท่าหรือมีสมรรถนะเพิ่มขึ้นร้อยละ 10 เมื่อเทียบกับคำสั่ง **`static`** ซึ่งใกล้เคียงกับโอกาสการเพิ่มสมรรถนะที่ได้จากวิเคราะห์คุณสมบัติพื้นที่เก็บข้อมูล จากภาพที่ 6.9 จะเห็นได้ว่าสาเหตุสำคัญที่ทำให้คำสั่ง **`guided`** มีสมรรถนะสูงคำสั่งอื่นคือมีจำนวนการเข้าถึงข้อมูลน้อยที่สุด และจากภาพที่ 6.9 จะสังเกตได้ว่า คำสั่ง **`static(16)`**, **`dynamic(16)`** และ **`guided(16)`** มีจำนวนการเข้าถึงข้อมูลใกล้เคียงกัน แต่คำสั่ง **`guided(16)`** มีค่าปีดอัพสูงกว่า เนื่องจากมีจำนวนครั้งของการแจกลูปน้อยกว่าคำสั่ง **`static(16)`** และ **`dynamic(16)`**

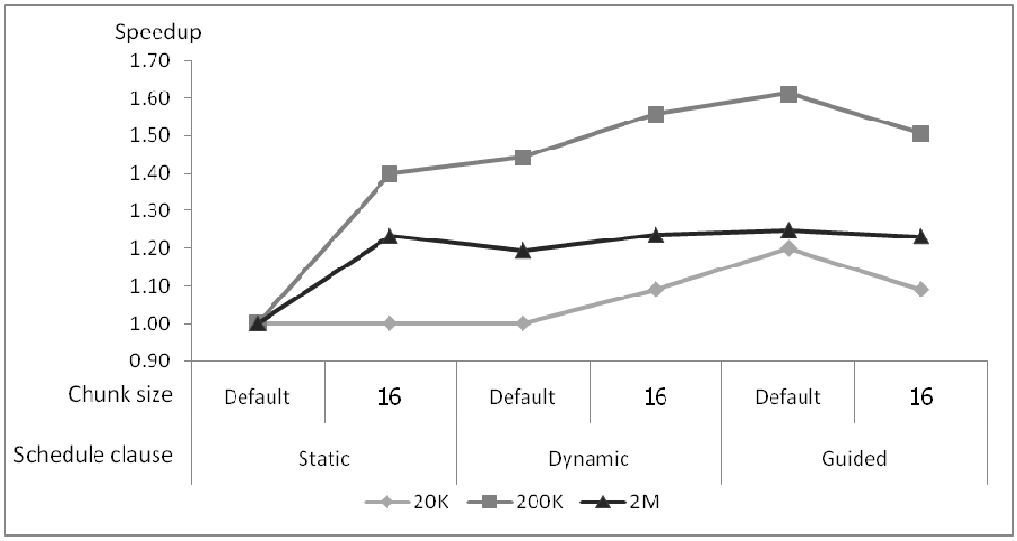
ผลการวิเคราะห์อัตราการไม่พบข้อมูลระดับที่ 1 พบว่า ทุกคำสั่งมีอัตราเฉลี่ยใกล้เคียงกันที่ร้อยละ 0.03 และผลจากการจำแนกชนิดของการไม่พบข้อมูล (ภาพที่ 6.10) พบว่าการประมวลผลด้วยทุกคำสั่งทำให้เกิดการไม่พบข้อมูลเนื่องจากพื้นที่เต็มมีค่าเฉลี่ยร้อยละ 63 ดังนั้น การไม่พบข้อมูลเนื่องจากพื้นที่เต็มจึงผลกระทบต่อสมรรถนะมากที่สุด ซึ่งผลกระทบ

ดังกล่าวนี้เป็นสาเหตุที่ทำให้คำสั่ง **dynamic** มีค่าสปีดอัพลดลง แม้ว่าจะมีจำนวนการเข้าถึงข้อมูลน้อยกว่าคำสั่ง **static**

ภาพที่ 6.10 ชนิดการไม่พบข้อมูลในแคชระดับที่ 1 ของโปรแกรมหาค่าจำนวนเฉพาะ



ภาพที่ 6.11 ค่าสปีดอัพของการประมวลผลบนระบบคอมพิวเตอร์จริง



ผลการทดลองส่วนสุดท้ายเป็นค่าสปีดอัพเมื่อเทียบกับคำสั่ง **static** ที่ได้จากการประมวลผลบนระบบคอมพิวเตอร์จริง (ภาพที่ 6.11) โดยการหาค่าจำนวนเฉพาะสามขนาด คือ ตั้งแต่ 1 ถึง 20,000 (20K) ตั้งแต่ 1 ถึง 200,000 (200K) และ ตั้งแต่ 1 ถึง 2,000,000 (2M) ผลการทดลองพบว่า คำสั่ง **guided** เป็นคำสั่งที่ให้ค่าสปีดอัพสูงที่สุดของการหาค่าจำนวนเฉพาะทั้งสามขนาด ซึ่งสอดคล้องกับผลการวิเคราะห์ของพี-สปา

อย่างไรก็ตาม การประมวลผลบนคอมพิวเตอร์จริง คำสั่ง **dynamic** ใช้เวลาประมวลผลน้อยกว่าคำสั่ง **static** ทำให้มีค่าสปีดสูงกว่าการวิเคราะห์ของพี-สปา เนื่องจากการแจกลูปไปยังแต่ละเทรดของคำสั่ง **dynamic** จะขึ้นอยู่กับความพร้อมของเทรดด้วย หากมีเทรดใดว่างก็สามารถรับลูปไปประมวลผลได้ทันที ซึ่งสภาพแวดล้อมการทำงานของแต่ละเทรดอาจจะไม่เหมือนเดิมในการประมวลผลแต่ละครั้ง ดังนั้นสมรรถนะของคำสั่ง **dynamic** จะขึ้นอยู่กับสภาพแวดล้อมระหว่างการประมวลผลด้วย แม้ว่าคำสั่ง **guided** มีลักษณะการแจกลูปเช่นเดียวกับคำสั่ง **dynamic** แต่ทว่าบนโปรแกรมที่มีภาระงานไม่สมดุลนั้น คำสั่ง **guided** ซึ่งสามารถปรับจำนวนลูปที่แจกไปยังแต่ละเทรดได้ ทำให้มีข้อได้เปรียบและมีความเหมาะสมกับคุณลักษณะโปรแกรมดังกล่าวกว่าคำสั่งอื่นที่ใช้ในการทดลอง

6.2.4 สรุปและอภิปรายผลการทดลอง

จากวิเคราะห์โปรแกรมหาค่าจำนวนเฉพาะด้วยเครื่องมือวิเคราะห์พี-สปา พบว่า แม้โปรแกรมดังกล่าวเป็นโปรแกรมที่ภาระงานที่ไม่สมดุล ซึ่งทำให้ปรับปรุงสมรรถนะได้ยาก แต่จากนำเสนอข้อมูลคุณสมบัติการเข้าพื้นที่ข้อมูลของพี-สปา ทำให้ทราบถึงโอกาสการปรับสมรรถนะของโปรแกรมได้ จากการวิเคราะห์ถึงรายละเอียดจำนวนการเข้าถึงข้อมูลและการไม่พบข้อมูลในแคช รวมถึงการทดลองประมวลผลบนระบบคอมพิวเตอร์จริงของคำสั่งการแจกลูป 6 คำสั่ง พบว่าคำสั่ง **guided** ซึ่งเป็นคำสั่งที่สามารถปรับขนาดของการแจกลูปในระหว่างการประมวลผลได้ ใช้การประมวลผลน้อยที่สุดและทำให้เกิดจำนวนเข้าถึงน้อยที่สุด นอกจากนี้ เมื่อจำแนกชนิดการไม่พบข้อมูลในแคชทำให้ทราบว่า การไม่พบข้อมูลเนื่องจากพื้นที่เต็มเป็นข้อจำกัดเชิงสมรรถนะของโปรแกรมจำนวนหาค่าเฉพาะ ซึ่งการทดลองนี้สรุปได้ว่า คำสั่ง **guided** เป็นคำสั่งที่ช่วยเพิ่มสมรรถนะของโปรแกรมได้ใกล้เคียงเป้าหมายที่กำหนดไว้

6.3 การปรับสมรรถนะฟังก์ชันการคูณเมทริกซ์ ซึ่งมีอัตราการแบ่งใช้ข้อมูลสูง

โปรแกรมที่มีอัตราที่มีการแบ่งใช้ข้อมูลสูง เป็นปัญหาอีกลักษณะหนึ่งที่ทำให้เกิดข้อจำกัดเชิงสมรรถนะของโปรแกรมเชิงขนาน ฟังก์ชันการคูณเมทริกซ์เป็นตัวอย่างที่เก่าแก่ซึ่งมีคุณลักษณะการแบ่งใช้ข้อมูลสูง ปัญหาดังกล่าวจะส่งกระทบอย่างชัดเจนบนระบบคอมพิวเตอร์ที่มีกลไกรักษาความสอดคล้องของข้อมูล การทดลองนี้จึงนำใช้เครื่องมือพี-สปาวิเคราะห์ฟังก์ชันการ

คูณเมทริกซ์ บนชิปมัลติโพรเซสเซอร์ที่ใช้แคชระดับที่ 2 แยกกัน เพื่อหาแนวทางการปรับปรุงสมรรถนะของโปรแกรมที่ลักษณะดังกล่าว ซึ่งรายละเอียดจะได้กล่าวถึงในหัวข้อถัดไป

6.3.1 คุณลักษณะของฟังก์ชันคูณเมทริกซ์

ฟังก์ชันการคูณเมทริกซ์เป็นภาระหลักของการประมวลผลโปรแกรมการคำนวณทางวิทยาศาสตร์ การประมวลผลข้อมูลมัลติมีเดีย และการประมวลผลภาพกราฟฟิก ฟังก์ชันการคูณเมทริกซ์ที่นำมาใช้การทดลองนี้ เป็นการนำฟังก์ชันการคูณเมทริกซ์อย่างง่าย ซึ่งเป็นประมวลผลในลูปซ้อนกันสามชั้น นำไปพัฒนาให้ประมวลผลแบบขนานด้วยคำสั่ง **`parallel for`** ของโอเพนเอ็มพี ดังแสดงในภาพที่ 6.12 การพัฒนาโปรแกรมเชิงขนานในลักษณะดังกล่าวนี้ เป็นวิธีการที่โปรแกรมเมอร์ผู้ไม่ชำนาญนิยมใช้กันเนื่องจากเป็นวิธีการที่ตรงไปตรงมา

ภาพที่ 6.12 ฟังก์ชันการคูณเมทริกซ์ที่ปรับปรุงให้ประมวลผลแบบขนานด้วยโอเพนเอ็มพี

```
1:  void matrices_multiplication(double a[size][size], double b[size][size],
2:                                double result[size][size])
3:  {
4:       int i, j, k;
5:
6:       #pragma omp parallel for private(i, j, k)
7:       for(i=0; i<size; i++)
8:            for(j=0; j<size; j++)
9:                 for(k=0; k<size; k++)
10:                     result[i][j] = a[i][k]*b[k][j];
11:
12: }
```

จากภาพที่ 6.12 จะเห็นได้ว่า ในส่วนการประมวลผลที่อยู่ภายใต้การวนลูปซ้อนกันสามชั้น (บรรทัดที่ 6 – 10) เป็นการประมวลผลที่ต้องเกี่ยวข้องอาเรย์ข้อมูลสามชุด ประกอบด้วยอาเรย์ข้อมูลตัวคูณสองชุด (ตัวแปร a และ b) และอาเรย์ผลลัพธ์หนึ่งชุด (ตัวแปร result) ซึ่งการประมวลผลตัวแปร result 1 ช่องนั้น ต้องใช้ตัวแปร a หนึ่งแถวและตัวแปร b หนึ่งคอลัมน์ จากลักษณะดังกล่าวทำให้ทุกเทรดมีการแบ่งใช้ข้อมูลร่วมกันสูง เนื่องจากต้องใช้ข้อมูลทั้งแถวหรือคอลัมน์ในการคำนวณ นอกจากนี้หากตัวแปร result ถูกเก็บอยู่บนแคชบล็อกเดียวกันขณะประมวลผล ก็ทำให้มีโอกาสเกิดการเขียนข้อมูลบนแคชบล็อกเดียวกันจากหลายหน่วยประมวลผล ซึ่งปัญหานี้จะทวีความรุนแรงบนชิปมัลติโพรเซสเซอร์ที่ใช้แคชแยกกัน และมีกลไกรักษาความสอดคล้องของข้อมูล เนื่องจากต้องมีการยกเลิกบนแคชบล็อกดังกล่าวบ่อยครั้ง จากคุณลักษณะ

ดังกล่าวทำให้มีโอกาสเกิดอัตราการไม่พบข้อมูลกรณีร่วมใช้ไม่จริงสูงขึ้น การทดลองนี้จึงต้องการศึกษาเพื่อหาแนวทางปรับสมรรถนะการประมวลผลฟังก์ชันการคูณเมทริกซ์ บนชิปมัลติโพรเซสเซอร์ที่ใช้แคชระดับที่ 2 แยกกัน

ตารางที่ 6.3 สถาปัตยกรรมของชิปมัลติโพรเซสเซอร์ที่ใช้วิเคราะห์ฟังก์ชันการคูณเมทริกซ์

สถาปัตยกรรมของชิปมัลติโพรเซสเซอร์	ค่าทางสถาปัตยกรรม
สถาปัตยกรรมของหน่วยประมวลผล	
- จำนวนหน่วยประมวลผล	2 หน่วย
- ชุดคำสั่ง	x86_64
สถาปัตยกรรมของแคชระดับที่ 1	
- ขนาดของแคชระดับที่ 1 (แยกแคชคำสั่งและข้อมูล)	32 KB / private
- ขนาดของแคชบล็อก	64 bytes
- จำนวนเซต	8 way
- นโยบายการเขียน	write-through non-allocate
- เวลาการเข้าถึงข้อมูล	3 cycles
สถาปัตยกรรมของแคชระดับที่ 2	
- ขนาดของแคชระดับที่ 2	4 MB / Private
- ขนาดของแคชบล็อก	64 bytes
- จำนวนเซต	16 way
- นโยบายการเขียน	write-back, allocate
- เวลาการเข้าถึงข้อมูล	10 cycles
- เวลาการเข้าถึงข้อมูลในกรณีไม่พบข้อมูล	165 cycles
กลไกรักษาความสอดคล้องของข้อมูลในแคช	MESI แคชระดับที่ 2
ความจุของหน่วยความจำหลัก	4 GB

6.3.2 **วิธีทำการทดลอง**

วิธีดำเนินการทดลองเพื่อศึกษาแนวทางการปรับสมรรถนะฟังก์ชันการคูณเมทริกซ์ นี้มีขั้นตอนเหมือนกับการวิเคราะห์โปรแกรมหาค่าจำนวนเฉพาะ โดยระบบคอมพิวเตอร์เป้าหมายได้ใช้ชิปมัลติโพรเซสเซอร์ที่ใช้แคชระดับที่ 2 แยกกันและทำงานบนระบบปฏิบัติการลินิกซ์ ค่าทางสถาปัตยกรรมดังที่สรุปไว้ในตารางที่ 6.3 ขั้นตอนที่สอง การทดลองนี้ประมวลผลฟังก์ชันการคูณ

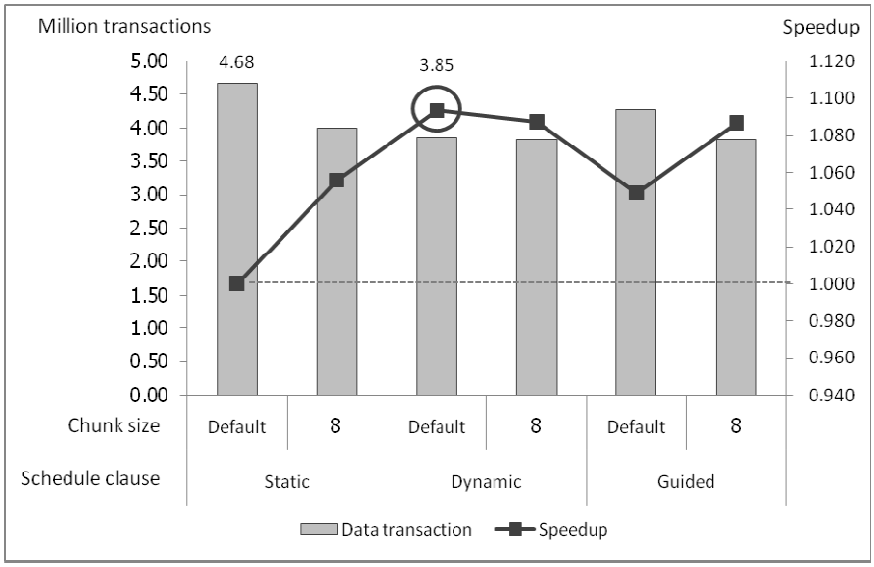
เมทริกซ์บนอาเรย์ขนาด 32x32 ช่อง เพื่อเปรียบเทียบสมรรถนะคำสั่งการแจกลูป 6 คำสั่งประกอบด้วย **`static`**, **`static(8)`**, **`dynamic`**, **`dynamic(8)`**, **`guided`** และ **`guided(8)`** ซึ่งจำนวนลูป 8 ลูปต่อบล็อกเป็นจำนวนของลูปที่มีขนาดเท่ากันกับแคชบล็อก และขั้นตอนสุดท้ายเป็น การนำผลลัพธ์มาวิเคราะห์คุณสมบัติพื้นที่เก็บข้อมูลเพื่อหาโอกาสการปรับสมรรถนะ และเปรียบเทียบอัตราและชนิดการไม่พบข้อมูลที่เป็นผลเนื่องจากการวิธีการแจกลูป

6.3.3 ผลการทดลอง

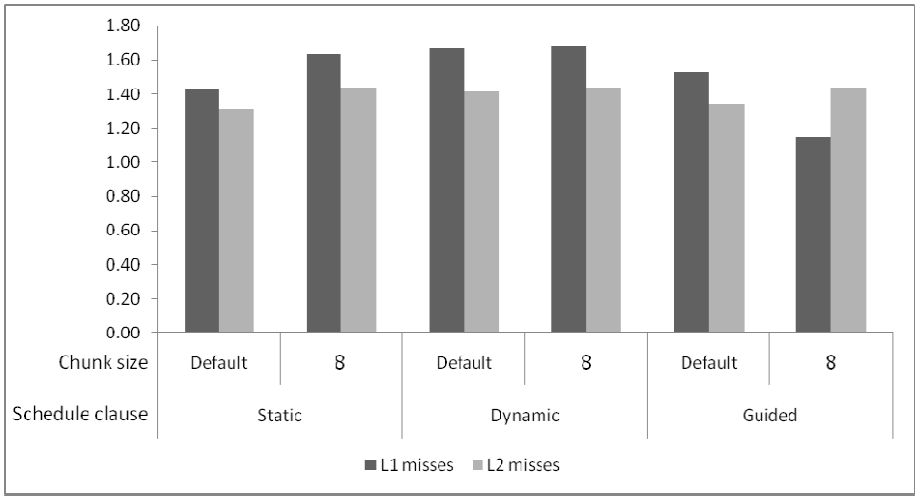
ผลการวิเคราะห์คุณสมบัติพื้นที่เก็บข้อมูลบนฟังก์ชันการคูณเมทริกซ์ พบว่า ฟังก์ชันการคูณเมทริกซ์สัดส่วนการเข้าถึงพื้นที่แบบ SRSW สูงร้อยละ 68.65 และมีพื้นที่หน่วยความจำที่ให้เกิดการปิดกั้นพื้นที่ (MW, MRMW และ SWMW) ร้อยละ 10.60 ซึ่งอาจจะเป็นสัดส่วนโอกาสที่จะปรับปรุงสมรรถนะของโปรแกรมได้

ผลการเปรียบเทียบจำนวนการเข้าถึงข้อมูลและค่าสปีดอัพเทียบกับคำสั่ง **`static`** (ภาพที่ 6.13) พบว่า คำสั่ง **`dynamic`**, **`dynamic(8)`** และ **`guided(8)`** ให้ค่าสปีดอัพสูงที่สุดประมาณ 1.09 เท่า หรือมีสมรรถนะสูงกว่าคำสั่ง **`static`** ร้อยละ 9 ซึ่งใกล้เคียงกับโอกาสการปรับสมรรถนะที่ได้จากวิเคราะห์คุณสมบัติพื้นที่เก็บข้อมูล นอกจากนี้ยังพบว่าทั้งสามคำสั่งดังกล่าวมีจำนวนการเข้าถึงข้อมูลน้อยที่สุดสามลำดับแรก โดยน้อยกว่าการเข้าถึงของคำสั่ง static ร้อยละ 20 ซึ่งเป็นสาเหตุที่ทำให้ทั้งสามคำสั่งมีสมรรถนะการประมวลผลสูงที่สุด

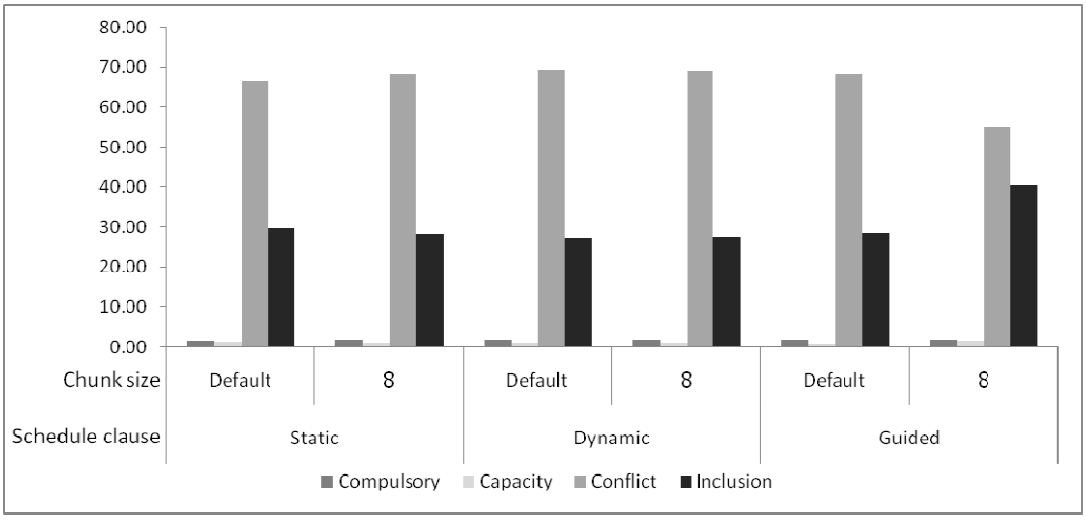
ภาพที่ 6.13 จำนวนการเข้าถึงข้อมูลและค่าสปีดอัพเมื่อกับคำสั่ง **`static`** ของฟังก์ชันการคูณเมทริกซ์



ภาพที่ 6.14 อัตราการไม่พบข้อมูลในแคชระดับที่ 1 และ 2 ของฟังก์ชันการคูณเมทริกซ์



ภาพที่ 6.15 ชนิดการไม่พบข้อมูลในแคชระดับที่ 1 ของฟังก์ชันการคูณเมทริกซ์



ส่วนการวิเคราะห์อัตราการไม่พบข้อมูลในแคชระดับที่ 1 (ภาพที่ 6.14) พบว่า คำสั่ง **`guided(8)`** มีอัตราการไม่พบข้อมูลน้อยที่สุดร้อยละ 1.15 และมีอัตราการไม่พบข้อมูลบนแคชระดับที่ 2 ใกล้เคียงกับคำสั่งอื่น ซึ่งควรจะทำให้คำสั่ง **`guided(8)`** ให้สมรรถนะประมวลผลสูงที่สุด แต่อย่างไรก็ตาม จากภาพที่ 6.15 ซึ่งเป็นแจกแจงชนิดของการไม่พบข้อมูลในแคชระดับที่ 1 พบว่า คำสั่ง **`guided(8)`** มีการไม่พบข้อมูลเนื่องจากข้อมูลดังกล่าวถูกยกเลิกจากแคชระดับที่ 2 (Inclusion miss) สูงกว่าคำสั่งอื่นถึงร้อยละ 20 จึงเป็นสาเหตุสำคัญที่ทำให้คำสั่ง **`guided(8)`** มีสมรรถนะการประมวลผลลดลง การไม่พบข้อมูลในแคชระดับที่ 1 ที่ถูกยกเลิกข้อมูลจากแคชระดับที่ 2 นั้น เกิดขึ้นเนื่องจากข้อมูลในแคชระดับที่ 2 ถูกแก้ไขจากหน่วยประมวลผลอื่น ดังนั้น แคช

ระดับที่ 2 ต้องแจ้งไปยังแคชระดับที่ 1 ทราบว่าข้อมูลที่มีอยู่ในแคชมีค่าไม่ถูกต้อง ซึ่งในกรณีนี้การไม่พบข้อมูลในแคชระดับที่ 1 เนื่องจากการยกเลิกข้อมูลระดับที่ 2 จะจัดอยู่ในชนิดการไม่พบข้อมูลเนื่องจากการใช้ข้อมูลร่วมกัน

แม้ว่าจากภาพที่ 6.15 จะชี้ให้เห็นว่าคำสั่ง **`guided(8)`** มีอัตราการไม่พบข้อมูลในระดับที่ 1 เนื่องจากการขัดแย้งของข้อมูลต่ำกว่าคำสั่งอื่น แต่ทว่าการไม่พบข้อมูลชนิดเนื่องจากการขัดแย้งของข้อมูลนั้น ยังทำให้มีโอกาสพบข้อมูลในแคชระดับที่ 2 สูง ทำให้ยังส่งข้อมูลกลับไปประมวลผลได้ทันที ในขณะที่การไม่พบข้อมูลเนื่องจากการใช้ข้อมูลร่วมกันนั้น ข้อมูลที่มีอยู่ในแคชระดับที่ 2 ก็เป็นข้อมูลที่ถูกยกเลิกด้วย ทำให้แคชระดับที่ 2 ต้องไปดึงข้อมูลจากหน่วยความจำซึ่งใช้ต้องเวลาสูงขึ้น ดังนั้น การไม่พบข้อมูลเนื่องจากการใช้ข้อมูลร่วมกัน จะส่งผลกระทบเชิงสมรรถนะสูงกว่าการไม่พบข้อมูลชนิดอื่น

6.3.4 สรุปและอภิปรายผลการทดลอง

แม้ฟังก์ชันการคูณเมทริกซ์เป็นโปรแกรมที่มีอัตราการแบ่งใช้ข้อมูลสูง และเมื่อต้องประมวลผลบนชิปมัลติโพรเซสเซอร์ที่ใช้แคชแยกกัน อาจจะทำให้เกิดการไม่พบข้อมูลเนื่องจากใช้ข้อมูลร่วมกันในอัตราที่สูง แต่จากการทดลองพบว่า กลุ่มคำสั่งที่แจกตัวนับลูปเมื่อโปรแกรมประมวลผลทั้งที่ปรับขนาดของลูปได้ (**`guided`**) และไม่ได้ (**`dynamic`**) จะช่วยปรับปรุงสมรรถนะของโปรแกรมได้สูงร้อยละ 9 และจากนำเสนอข้อมูลของพี-สปาทำให้พบว่า จุดสำคัญที่ทำให้เป็นข้อจำกัดบนชิปมัลติโพรเซสเซอร์ที่ใช้แคชร่วมกันคือ การไม่พบข้อมูลเนื่องจากใช้การใช้ข้อมูลร่วมกันในแคชระดับที่ 2 ซึ่งผลกระทบดังกล่าวจะส่งผลไปยังการทำงานของแคชระดับที่ 1 ด้วย

6.4 สรุปแนวทางวิเคราะห์และปรับสมรรถนะโปรแกรมโอเพนเอ็มพีโดยใช้พี-สปา

แนวทางการวิเคราะห์และปรับสมรรถนะโปรแกรมโอเพนเอ็มพีโดยใช้พี-สปาที่กล่าวถึงในบทนี้ งานวิจัยชิ้นนี้ได้นำเสนอแนวทางการใช้ประโยชน์พี-สปาไว้สองแนวทาง ตามข้อได้เปรียบของการวิเคราะห์บนระบบจำลองที่กล่าวในส่วนต้นของบทนี้ ซึ่งแบ่งเป็นสามการทดลอง คือ (1) การนำเสนอแนวทางการวิเคราะห์เพื่อเปรียบเทียบสถาปัตยกรรม ซึ่งได้ศึกษาผลกระทบของคุณสมบัติของพื้นที่เก็บข้อมูล ที่มีต่อระบบบัสของชิปมัลติโพรเซสเซอร์ที่มีสถาปัตยกรรมของแคชสองลักษณะ พบว่า สถาปัตยกรรมของแคชที่ใช้ร่วมกันมีสมรรถนะที่สูงกว่าการใช้แคชแยกกันทั้งในเชิงเวลาประมวลผล อัตราการไม่พบข้อมูลในแคช และความต้องการแบนด์วิดท์ (2) ได้

นำเสนอแนวทางการปรับปรุงโปรแกรมที่มีภาระงานไม่สมดุล ซึ่งได้ทดลองบนโปรแกรมหาค่าจำนวนเฉพาะ พบว่า การเลือกคำสั่งแจกลูปที่สามารถปรับขนาดทำให้โปรแกรมมีสมรรถนะสูงขึ้นร้อยละ 10 เมื่อเทียบกับคำสั่งการแจกลูปโดยปริยาย และ (3) นำเสนอแนวทางการปรับสมรรถนะของฟังก์ชันการคูณเมริกซ์ ซึ่งมีอัตราการแบ่งใช้ข้อมูลสูง พบว่า กลุ่มคำสั่งที่แจกลูปเมื่อโปรแกรมประมวลผลจะช่วยให้มีสมรรถนะสูงขึ้นร้อยละ 9 เมื่อเทียบกับคำสั่งการแจกลูปโดยปริยาย และการไม่พบข้อมูลเนื่องจากใช้ข้อมูลร่วมกัน เป็นข้อจำกัดเชิงสมรรถนะของโปรแกรมที่มีลักษณะดังกล่าวและประมวลผลบนซิปมัลติโพรเซสเซอร์ที่ใช้แคชแยกกัน

ภาพรวมของการปรับสมรรถนะโปรแกรมโอเพนเอ็มพีบนชิปมัลติโพรเซสเซอร์ จะเห็นได้ว่า โปรแกรมโอเพนเอ็มพีจะแสดงสมรรถนะการประมวลผลได้ดียิ่งขึ้น เมื่อประมวลผลบนระบบคอมพิวเตอร์ที่ใช้หน่วยความจำและแคชร่วมกัน เนื่องจากการพัฒนาโปรแกรมด้วยโอเพนเอ็มพีเป็นโมเดลการเขียนโปรแกรมแบบใช้หน่วยความจำร่วมกันดังที่กล่าวไว้ในบทที่ 2 จากการวิเคราะห์กลุ่มโปรแกรมการคำนวณทางวิทยาศาสตร์ ซึ่งพัฒนาด้วยโอเพนเอ็มพี พบว่า โดยส่วนใหญ่โปรแกรมในกลุ่มดังกล่าวมีลักษณะการเข้าถึงพื้นที่ข้อมูลที่ไม่ถูกปิดกั้น เนื่องจากพื้นที่ข้อมูลถูกอ่านและเขียนจากหน่วยประมวลผลเพียงหน่วยเดียว โอกาสการเพิ่มสมรรถนะของโปรแกรมในกลุ่มนี้จึงขึ้นอยู่กับสัดส่วนของพื้นที่ข้อมูลที่ถูกปิดกั้น ทั้งนี้การเลือกใช้คำสั่งการแจกลูปเพื่อช่วยเพิ่มสมรรถนะก็ขึ้นอยู่กับคุณลักษณะของโปรแกรมด้วย คำสั่งการแจกลูปเมื่อโปรแกรมประมวลผล (**`dynamic`** และ **`guided`**) ได้แสดงถึงโอกาสที่สามารถช่วยเพิ่มสมรรถนะได้สูงกว่าคำสั่งการแจกลูปที่กำหนดตายตัว (**`static`**) นอกจากนี้ จากคุณลักษณะเฉพาะของโปรแกรมการคำนวณทางวิทยาศาสตร์ที่มีการคำนวณในรูปแบบเดิมไม่ว่าจะประมวลผลบนชุดข้อมูลขนาดเล็กหรือใหญ่ (Temporally close) ดังนั้น คำสั่งการแจกลูปที่สามารถช่วยเพิ่มสมรรถนะได้ดีเมื่อประมวลผลบนชุดข้อมูลขนาดเล็ก ก็จะช่วยเพิ่มสมรรถนะของโปรแกรมได้ดีเมื่อประมวลผลบนชุดข้อมูลที่ใหญ่ขึ้น

เนื้อหาในบทนี้นำเสนอแนวทางการใช้ประโยชน์พี-สปาที่สามารถทำงานได้ครบถ้วนตามวัตถุประสงค์ของวิทยานิพนธ์นี้ ซึ่งเป็นการยืนยันว่าเครื่องมือพี-สปาสามารถนำไปใช้ประโยชน์ได้จริง และจะได้กล่าวสรุปและข้อเสนอแนะของวิทยานิพนธ์ฉบับนี้ในบทถัดไป