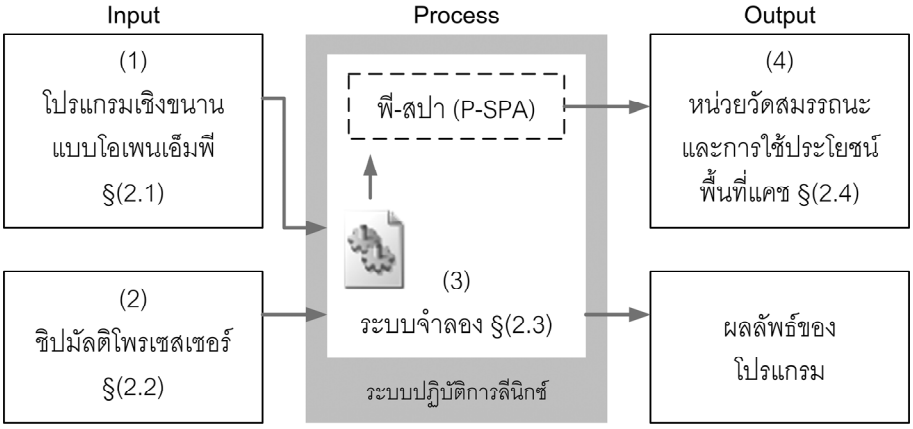
บทที่ 2

ความรู้พื้นฐานและทฤษฎีที่เกี่ยวข้อง

วิทยานิพนธ์ฉบับนี้นำเสนอเครื่องมือพี-สปา (P-SPA ย่อมาจาก Parallel program analysis using a Simics-base, Performance Analyzer) ซึ่งเป็นเครื่องมือวิเคราะห์สมรรถนะโปรแกรมเชิงขนานแบบโอเพนเอ็มพี เพื่อนำเสนอการใช้ประโยชน์พื้นที่แคช ซึ่งเป็นผลจากคำสั่งการแจกตัวของลูปเพื่อประมวลผลแบบขนาน และผู้ใช้สามารถวิเคราะห์สมรรถนะของโปรแกรมบนแคชหรือชิปมัลติโพรเซสเซอร์ ที่มีสถาปัตยกรรมแตกต่างกันได้

การวิจัยเพื่อพัฒนาเครื่องมือวิเคราะห์สมรรถนะพี-สปาตั้งอยู่บนความรู้พื้นฐานและทฤษฎีที่เกี่ยวข้องกับ ข้อมูลที่รับเข้าสู่เครื่องมือ (Input) แนวทางการพัฒนาเครื่องมือ (Process) และผลลัพธ์ของการวิเคราะห์สมรรถนะที่เครื่องมือควรนำเสนอได้ (Output) ซึ่งสรุปได้เป็นสี่ส่วน (ภาพที่ 2.1) ประกอบด้วย (1) โปรแกรมเชิงขนานแบบโอเพนเอ็มพี และ (2) คุณลักษณะของชิปมัลติโพรเซสเซอร์ที่เป็นระบบคอมพิวเตอร์เป้าหมาย (3) ทฤษฎีที่เกี่ยวข้องกับการประเมินสมรรถนะของระบบคอมพิวเตอร์ด้วยระบบจำลอง และ (4) ทฤษฎีที่เกี่ยวข้องกับการนำเสนอข้อมูลของเครื่องมือพี-สปา ได้แก่ หน่วยวัดสมรรถนะและการใช้ประโยชน์พื้นที่แคช ซึ่งหัวข้อถัดไปของบทนี้ จะได้กล่าวถึงรายละเอียดของทฤษฎีที่เกี่ยวข้องทั้งสี่ส่วน

ภาพที่ 2.1 ขอบเขตการทำงานของเครื่องมือวิเคราะห์สมรรถนะพี-สปา



2.1 โปรแกรมเชิงขนานแบบโอเพนเอ็มพี

โมเดลการเขียนโปรแกรมเชิงขนานแตกต่างจากโมเดลการเขียนโปรแกรมเชิงลำดับตรงที่ โมเดลการเขียนโปรแกรมจะต้องมีความเหมาะสมกับสถาปัตยกรรมของระบบคอมพิวเตอร์และมีการแบ่งงานที่สอดคล้องกับอัลกอริทึมของโปรแกรม โมเดลการเขียนโปรแกรมเชิงขนานแบ่งออกเป็นสามโมเดลหลัก ประกอบด้วย *โมเดลการเขียนโปรแกรมแบบแบ่งข้อมูลเชิงขนาน* (Data parallel) *แบบใช้หน่วยความจำร่วม* (Shared memory) และ*แบบรับส่งข้อความ* (Message passing) ซึ่งมีสถาปัตยกรรมของระบบคอมพิวเตอร์เชิงขนาน*แบบใช้หน่วยความจำร่วม* (Shared memory) *แบบใช้หน่วยความจำกระจาย* (Distributed memory) และ*แบบใช้หน่วยความจำกระจายร่วมกัน* (Distributed shared-memory system) โมเดลการเขียนโปรแกรมจะช่วยให้โปรแกรมเมอร์เข้าใจถึงลักษณะการแบ่งงานและกระบวนการที่เกิดขึ้นระหว่างประมวลผล ทำให้คาดหวังผลลัพธ์และตรวจสอบความถูกต้องของโปรแกรมเมื่อนำไปประมวลผลบนสถาปัตยกรรมเป้าหมายได้

โมเดลการเขียนโปรแกรมเชิงขนานแบบใช้หน่วยความจำร่วมนั้น มีคุณลักษณะของโมเดลคือ การประมวลผลของโปรแกรมจะถูกแบ่งออกเป็นส่วนงานย่อย (Task) หลายส่วน ในแต่ละส่วนสามารถเข้าถึงข้อมูลหรือพื้นที่ในหน่วยความจำร่วมกันได้ ไม่มีส่วนงานย่อยใดเป็นเจ้าของข้อมูลโดยตรง ทำให้โปรแกรมเมอร์ไม่จำเป็นต้องเขียนคำสั่งในโปรแกรม เพื่อระบุวิธีการสื่อสารและแลกเปลี่ยนข้อมูล งานย่อยแต่ละส่วนสื่อสารกันโดยใช้ตัวแปรร่วมกัน ซึ่งถือเป็นการสื่อสารและแลกเปลี่ยนข้อมูลไปโดยปริยาย (Implicit communication) ลักษณะการใช้ตัวแปรเพื่อสื่อสารนี้มีความใกล้เคียงกับโปรแกรมแบบลำดับ จึงทำให้การเขียนโปรแกรมแบบขนานด้วยโมเดลนี้ทำได้รวดเร็ว และง่ายต่อการทำความเข้าใจลักษณะการประมวลผลของโปรแกรม

ด้วยเหตุนี้โมเดลการเขียนโปรแกรมเชิงขนานแบบใช้หน่วยความจำร่วม จึงได้รับความสนใจจากกลุ่มบริษัทผู้ผลิตฮาร์ดแวร์และซอฟต์แวร์คอมพิวเตอร์[1] ใช้เป็นโมเดลการเขียนโปรแกรมมาตรฐานบนสถาปัตยกรรมคอมพิวเตอร์แบบใช้หน่วยความจำร่วม กลุ่มบริษัทเหล่านี้ได้ก่อตั้งกลุ่มที่ชื่อว่า เออาร์บี (ARB ย่อมาจาก Architecture review board) ขึ้นมา เพื่อร่วมมือกันวิจัยและพัฒนาเครื่องมือสำหรับพัฒนาโปรแกรมเชิงขนาน โดยใช้โมเดลการเขียนโปรแกรมแบบใช้หน่วยความจำร่วม ในปี พ.ศ.2540 กลุ่มบริษัทเหล่านี้ได้นำเสนอ โอเพนเอ็มพีซึ่งเป็นส่วนต่อ

---

[1] AMD, Cray, Fujisu/Lahey, HP, IBM, Intel, Microsoft, NEC, The Portland Group, SGI, Sun, Totalview Technologies and Vast from Crescent Bay Software

ประสานโปรแกรมประยุกต์หรือเอพีไอ สำหรับการพัฒนาโปรแกรมเชิงขนาน ที่ใช้โมเดลการเขียนโปรแกรมแบบใช้หน่วยความจำร่วม (Chapman *et al*., 2008)

ดังที่กล่าวในบทนำแล้วว่า เอพีไอของโอเพนเอ็มพีประกอบด้วย คำสั่งกำกับคอมไพเลอร์ ไลบรารีร่วมฟังก์ชันสำเร็จ และตัวแปรกำหนดสภาพแวดล้อมในการประมวลผล ซึ่งเอพีไอเหล่านี้ถูกพัฒนาเพื่อใช้บนภาษาซี ซีพลัสพลัส และฟอร์แทรน โปรแกรมที่แทรกด้วยเอพีไอของโอเพนเอ็มพี เมื่อนำไปประมวลผลจะแบ่งส่วนงานย่อยหนึ่งออกเป็นหนึ่งเทรด (Thread paradigm) ในหนึ่งโปรแกรมสามารถแบ่งส่วนงานย่อยออกเป็นหลายเทรด ตามจำนวนหน่วยประมวลผลที่มีหรือตาม ที่โปรแกรมเมอร์กำหนด กล่าวคือเป็นโปรแกรมเชิงขนานแบบหลายเทรดนั่นเอง ซึ่งในหัวข้อถัดไปจะกล่าวถึง รูปแบบหรือโมเดลการประมวลผลของโปรแกรมแบบโอเพนเอ็มพีโดยสรุป รายละเอียดเพิ่มเติมของเนื้อหาในส่วนดังกล่าวได้เรียบเรียงและสรุปมาจากคู่มือการเขียนโปรแกรมด้วยโอเพนเอ็มพี (OpenMP, 2005; Chapman *et al*., 2008)

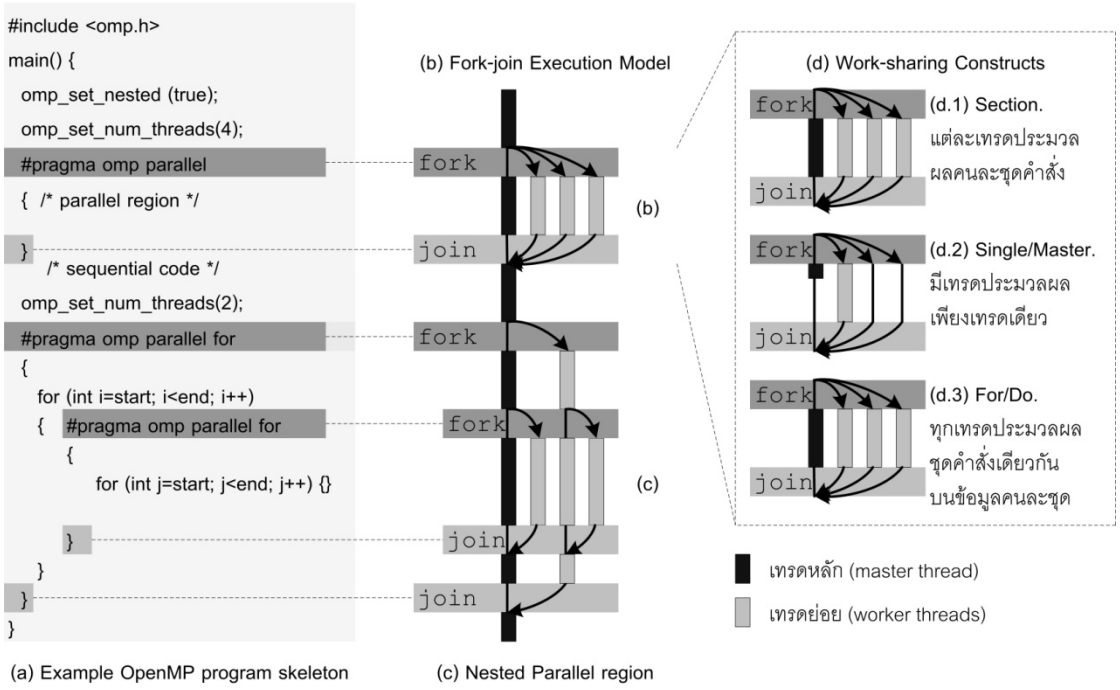
2.1.1 โมเดลการประมวลผล

โปรแกรมเชิงขนานที่พัฒนาด้วยเอพีไอของโอเพนเอ็มพีนั้น จะมีโมเดลการประมวลผลแบบ*ฟอร์ก-จอยน์* (Fork-join execution model) โดยเมื่อเริ่มต้นประมวลผล โปรแกรมจะมีเทรดเดียว ซึ่งเรียกว่าเทรดหลัก (Master thread) เมื่อโปรแกรมประมวลผลมาถึงส่วนที่โปรแกรมเมอร์แทรกคำสั่งกำกับไว้ เทรดหลักจะสร้างทีมของเทรดย่อย (Worker thread) ขึ้นมา (ดังภาพที่ 2.2 (a) ตัวอย่างโปรแกรมแบบโอเพนเอ็มพีอย่างง่ายโปรแกรมหนึ่ง และภาพที่ 2.2 (b) แสดงตัวอย่างการสร้างเทรดย่อยตามลักษณะของโมเดลฟอร์ก-จอยน์) ซึ่งโปรแกรมเมอร์สามารถกำหนดจำนวนของเทรดย่อยได้ตามต้องการผ่านเอพีไอของโอเพนเอ็มพี จากนั้นเทรดหลักจะแบ่งงานประมวลผลไปยังเทรดย่อย เมื่อเทรดย่อยประมวลผลเสร็จเรียบร้อย ก็จะส่งผลลัพธ์กลับมายังเทรดหลัก ส่วนของโปรแกรมตั้งแต่มีการสร้างเทรดย่อยเพื่อประมวลผลพร้อมกัน จนกระทั่งเทรดย่อยประมวลผลเสร็จ เรียกส่วนดังกล่าวว่า *พื้นที่ส่วนประมวลผลแบบขนาน* (Parallel region) ซึ่งเมื่อสิ้นสุดพื้นที่ส่วนประมวลผลแบบขนาน เทรดย่อยที่ถูกสร้างขึ้นจะถูกทำลายออกไปจากโปรแกรม ซึ่งโปรแกรมเมอร์สามารถใช้เอพีไอของโอเพนเอ็มพี เพื่อกำหนดพื้นที่ส่วนประมวลผลแบบขนานซ้อนกันได้อีกด้วย ดังแสดงในภาพที่ 2.2 (c)

เอพีไอของโอเพนเอ็มพีมีคำสั่งกำกับคอมไพเลอร์เพื่อแบ่งงานประมวลผล (Work-sharing constructs) สามลักษณะ ดังแสดงในภาพที่ 2.2 (d) ซึ่งประกอบด้วย (1) คำสั่งกำกับ

section ดังภาพที่ 2.2 (d1) เป็นการแบ่งงานหรือส่วนของโปรแกรมที่แตกต่างกันให้แต่ละเทรด (2) คำสั่งกำกับ **single** และ **master** ดังภาพที่ 2.2 (d2) เป็นการกำหนดให้มีเพียงหนึ่งเทรดเท่านั้นที่มีสิทธิ์เข้าไปประมวลผลในส่วนของโปรแกรมดังกล่าว โดยคำสั่ง **single** นั้นเทรดที่เข้าไปประมวลผลอาจจะเป็นเทรดหลักหรือเทรดย่อยก็ได้ ในขณะที่คำสั่ง **master** เทรดหลักเท่านั้นที่จะเข้าไปประมวลผลได้ และ (3) คำสั่งกำกับ **for**, **parallel for**, **do** และ **parallel do** ดังภาพที่ 2.2 (d3) เป็นการแบ่งงานประมวลผลที่ระดับลูป โดยการแจกตัวนับลูปไปให้ทุกเทรดในโปรแกรม ซึ่งจะประมวลผลด้วยชุดคำสั่งเดียวกัน แต่ประมวลผลบนข้อมูลคนละส่วน ซึ่งการแบ่งงานประมวลผลนี้อาจมองในภาพรวมได้เป็นสองระดับ คือ การแบ่งที่ระดับงาน (Task level) โดยการใช้คำสั่งกำกับ **section**, **single** และ **master** กับการแบ่งที่ระดับลูป (Loop level) ด้วยคำสั่งกำกับ **for**

ภาพที่ 2.2 ตัวอย่างโปรแกรมแบบโอเพนเอ็มพี (a) โครงสร้างของโปรแกรมที่เขียนด้วยเอพีไอของโอเพนเอ็มพี (b) โมเดลการประมวลผลแบบฟอร์ก-จอยน์ (c) การประมวลผลแบบขนานในลูปซ้อนกันหลายระดับ และ (d) วิธีการแบ่งงานประมวลผลตามคำสั่งกำกับสามลักษณะ



โปรแกรมเมอร์ใช้โมเดลการประมวลผลเป็นแนวทางการพัฒนาโปรแกรมเชิงขนานเพื่อกำหนดให้โปรแกรมมีลำดับการประมวลผลและได้ผลลัพธ์ตรงกับผลลัพธ์จากโปรแกรมเชิงลำดับ ไม่ว่าแบ่งการประมวลผลเป็นกี่เทรด หรือที่เรียกว่าโปรแกรมมีคุณสมบัติเทรดเซฟ (Thread

safe) แต่อย่างไรก็ตาม เนื่องจากโอเพนเอ็มพีใช้โมเดลการเขียนโปรแกรมแบบใช้หน่วยความจำร่วม แต่ละเทรดสามารถเข้าถึงข้อมูลหรือตัวแปรเดียวกัน เพื่ออ่านและเปลี่ยนแปลงค่าของตัวแปรร่วมกันได้ ซึ่งการเข้าถึงข้อมูลของแต่ละเทรดในระหว่างการประมวลผลอาจมีลำดับที่ไม่แน่นอน ส่งผลให้การประมวลผลของโปรแกรมในแต่ละครั้งอาจได้ผลลัพธ์ที่ไม่ตรงกัน ทำให้โปรแกรมเชิงขนานที่ได้ขาดคุณลักษณะเทรดเซฟ ด้วยเหตุนี้ การประมวลผลแบบขนานจำเป็นต้องมีโมเดลหรือรูปแบบของลำดับการเข้าถึงข้อมูลในหน่วยความจำ เพื่อให้โปรแกรมเมอร์ทราบถึงลำดับการเข้าถึงข้อมูลในระหว่างที่โปรแกรมประมวลผล ในหัวข้อถัดไปจะได้กล่าวถึงโมเดลที่ใช้ข้อกำหนดลำดับการเข้าถึงข้อมูลในหน่วยความจำของโปรแกรมแบบโอเพนเอ็มพี

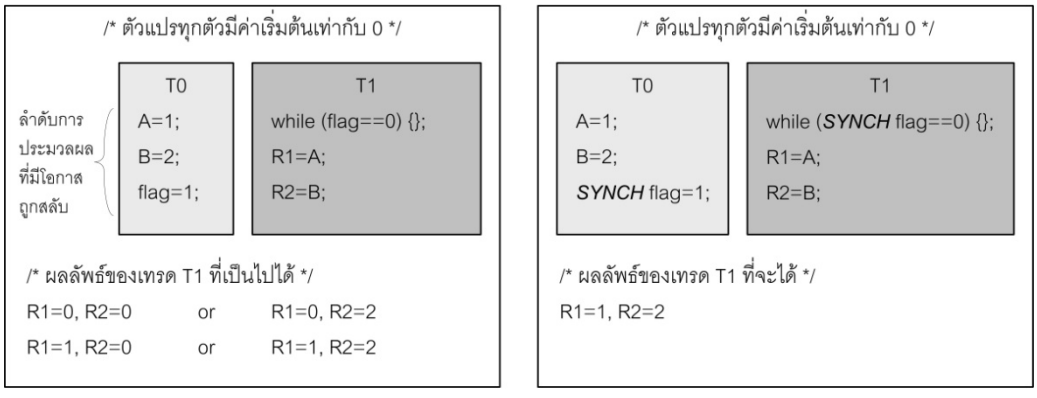
2.1.2 โมเดลความต้องกันของหน่วยความจำ

โปรแกรมแบบโอเพนเอ็มพีกำหนดรูปแบบของหน่วยความจำ หรือ *โมเดลของหน่วยความจำ* (Memory model) ออกเป็นสองส่วนตามขอบเขตการเข้าถึงตัวแปร ได้แก่ ส่วนพื้นที่ที่ใช้ร่วมกันและส่วนพื้นที่ชั่วคราว ส่วนพื้นที่ใช้ร่วมกันเป็นพื้นที่ที่ใช้เก็บตัวแปรซึ่งทุกเทรดในโปรแกรมสามารถใช้ค่าของตัวแปรเหล่านั้นร่วมกันได้ จึงเรียกว่าเป็น *พื้นที่สำหรับเก็บตัวแปรใช้ร่วมกัน* (Shared variable) ส่วนที่สองคือส่วนพื้นที่หน่วยความจำชั่วคราวของแต่ละเทรด (Temporary view) เป็นพื้นที่ที่ใช้เก็บตัวแปรที่ทำไว้ใช้เฉพาะกับคำสั่งในเทรดของตน กล่าวคือเป็น *พื้นที่สำหรับเก็บตัวแปรส่วนตัว* (Private variable) ซึ่งแต่ละเทรดไม่สามารถแลกเปลี่ยนค่าของตัวแปรส่วนตัวกันได้โดยตรง ต้องส่งผ่านพื้นที่หน่วยความจำของโปรแกรมอีกที นอกจากนี้ พื้นที่หน่วยความจำชั่วคราวของแต่ละเทรด ไม่จำเป็นต้องอยู่ในหน่วยความจำอาจอยู่ในรีจิสเตอร์หรือแคชก็ได้ ซึ่งแต่ละเทรดจะไม่สามารถเข้าถึงพื้นที่หน่วยความจำชั่วคราวของเทรดอื่นได้ คุณสมบัติดังกล่าวนี้เรียกว่า *พื้นที่ส่วนตัวของเทรด* (Threadprivate memory) (OpenMP, 2005)

โปรแกรมเชิงขนานมี *โมเดลความต้องกันของหน่วยความจำ* (Memory consistency model) เพื่อใช้เป็นข้อกำหนดกลางที่เข้าใจร่วมกันระหว่างผู้ผลิตหน่วยประมวลผล ผู้ผลิตคอมไพเลอร์และโปรแกรมเมอร์ ว่า ณ จุดใดในโปรแกรมที่ทุกเทรดซึ่งกำลังประมวลผลอยู่ในหลายหน่วยประมวลผล ควรจะมองเห็นตัวแปรใช้ร่วมกันมีค่าตรงกันหรือสอดคล้องต้องกัน โปรแกรมแบบโอเพนเอ็มพีก็เช่นกัน เอพีไอของโอเพนเอ็มพีจะระบุโมเดลความต้องกันของหน่วยความจำเพื่อให้คอมไพเลอร์บังคับให้เกิดการดำเนินการที่ทำให้ค่าของตัวแปรในพื้นที่หน่วยความจำ และพื้นที่หน่วยความจำชั่วคราวของทุกเทรดมีค่าต้องกัน ตามที่ถูกระบุไว้ในคำสั่งกำกับ

เอพีไอของโอเพนเอ็มพีใช้โมเดลกำหนดความต้องกัน ซึ่งจัดอยู่ในกลุ่ม *โมเดลแบบผ่อนคลาย* (Relaxed consistency models) ที่มีลักษณะใกล้เคียงกับ *โมเดลแบบผ่อนปรนลำดับ* (Weak-ordering consistency model) โดยลักษณะกำหนดของโมเดลกำหนดความต้องกันในกลุ่มผ่อนคลายมีข้อกำหนดว่า ในพื้นที่ส่วนประมวลผลแบบขนานนั้น โอเพนเอ็มพีไม่รับประกันว่าทุกเทรดจะเห็นค่าของตัวแปรสอดคล้องกัน โปรแกรมเมอร์จะต้องวางจุดบังคับ ซึ่งเมื่อประมวลผลถึง ณ จุดบังคับนั้น ทุกเทรดจึงจะเห็นค่าตัวแปรในโปรแกรมมีค่าตรงกัน จากภาพที่ 2.3 (a) แสดงถึงลักษณะของโปรแกรมที่มีปัญหาความต้องกันของหน่วยความจำ การประมวลผลของเทรด T0 นั้น หน่วยประมวลผลหรือคอมไพเลอร์สามารถสลับลำดับการประมวลผลของแต่ละคำสั่งได้ โดยที่ไม่มีผล กระทบต่อผลลัพธ์ภายในเทรด T0 แต่การสลับลำดับดังกล่าว จะส่งผลต่อการประมวลผลของเทรด T1 ซึ่งผลลัพธ์ของเทรด T1 ขึ้นอยู่กับลำดับการประมวลผลคำสั่ง **`flag=1`** ของเทรด T0 โมเดลแบบผ่อนปรนลำดับนี้จะอนุญาตให้แต่ละเทรดสลับลำดับการประมวลผลได้ ดังแสดงในภาพที่ 2.3 (b) เทรด T0 สามารถประมวลผลคำสั่ง **`A=1`** และ **`B=2`** สลับกันได้ แต่ทั้งนี้เทรด T0 ต้องประมวลผลทั้งสองคำสั่งก่อนที่จะประมวลผลคำสั่ง ***`SYNCH`*** **`flag=1`** ซึ่งเป็นจุดที่ใช้บังคับให้ทั้งสองเทรดต้องทำงานประสานจังหวะเวลา (Synchronize) เพื่อกำหนดให้ตัวแปรในโปรแกรมมีค่าตรงกัน ซึ่งจะทำให้โปรแกรมได้ผลลัพธ์เดียวกันทุกครั้ง

ภาพที่ 2.3 ลักษณะของโปรแกรมที่ (a) มีปัญหาความต้องกันของหน่วยความจำ และ (b) แก้ปัญหาความต้องกันด้วยการวางจุดบังคับ ตามข้อกำหนดของโมเดลกำหนดความต้องกันแบบผ่อนปรนลำดับ



เอพีไอของโอเพนเอ็มพีมีคำสั่งกำกับคอมไพเลอร์ที่ใช้เป็นจุดบังคับให้ทุกเทรดมองเห็นตัวแปรมีค่าตรงกันนั้น เมื่อถึงจุดบังคับโปรแกรมแบบโอเพนเอ็มพีจะดำเนินการกระจายค่าของทุกตัวแปรในโปรแกรม (Flush operation) ซึ่งทำให้ตัวแปรที่อยู่ในพื้นที่หน่วยความจำ และพื้นที่หน่วยความจำชั่วคราวของทุกเทรดมีค่าตรงกัน คำสั่งกำกับคอมไพเลอร์ที่บังคับให้ดำเนินการ

กระจายค่า ตัวแปรนี้แบ่งออกเป็นสี่กลุ่ม (Hoeflinger & Supinski, 2005) ประกอบด้วย (1) กลุ่มคำสั่งกำกับที่กำหนดส่วนพื้นที่กีดกั้น (Barrier region) ได้แก่คำสั่งกำกับ **`barrier`** (2) กลุ่มคำสั่งกำกับที่กำหนดการเข้าและออก พื้นที่ส่วนประมวลผลแบบขนาน ได้แก่คำสั่งกำกับ **`parallel`**, พื้นที่ส่วนวิกฤต (Critical region) ได้แก่คำสั่งกำกับ **`critical`** และ **`atomic`**, พื้นที่กำหนดลำดับ (Ordered region) ได้แก่คำสั่งกำกับ **`ordered`** (3) กลุ่มคำสั่งกำกับที่กำหนดการแบ่งงานประมวลผล ได้แก่คำสั่งกำกับ **`section`**, **`for`** และ **`do`** และ (4) กลุ่มคำสั่งกำกับที่กำหนดให้ประมวลผลด้วยเทรดเพียงเทรดเดียว ได้แก่คำสั่งกำกับ **`single`** และ **`master`**

โมเดลความต้องกันของหน่วยความจำนั้น ทำให้โปรแกรมเมอร์ทราบว่า ณ จุดใดในโปรแกรมจำเป็นต้องใช้จุดบังคับเพื่อให้มองเห็นตัวแปรใช้ร่วมกันทุกตัวมีค่าตรงกัน ซึ่งการวางจุดบังคับนี้ช่วยรับประกันว่าโปรแกรมจะประมวลผลได้ถูกต้องเสมอไม่ว่าจะแบ่งงานออกกี่เทรด กล่าวคือ โปรแกรมมีคุณสมบัติเทรดเซฟ สามารถประมวลผลแบบขนานได้ผลลัพธ์ถูกต้องเหมือนการประมวลผลแบบลำดับทุกประการ การประมวลผลแบบขนานมีเทคนิคการประมวลผลหลายเทคนิค ซึ่งหัวข้อถัดไปจะกล่าวถึงเทคนิคการประมวลผลแบบขนานที่ระดับลูป

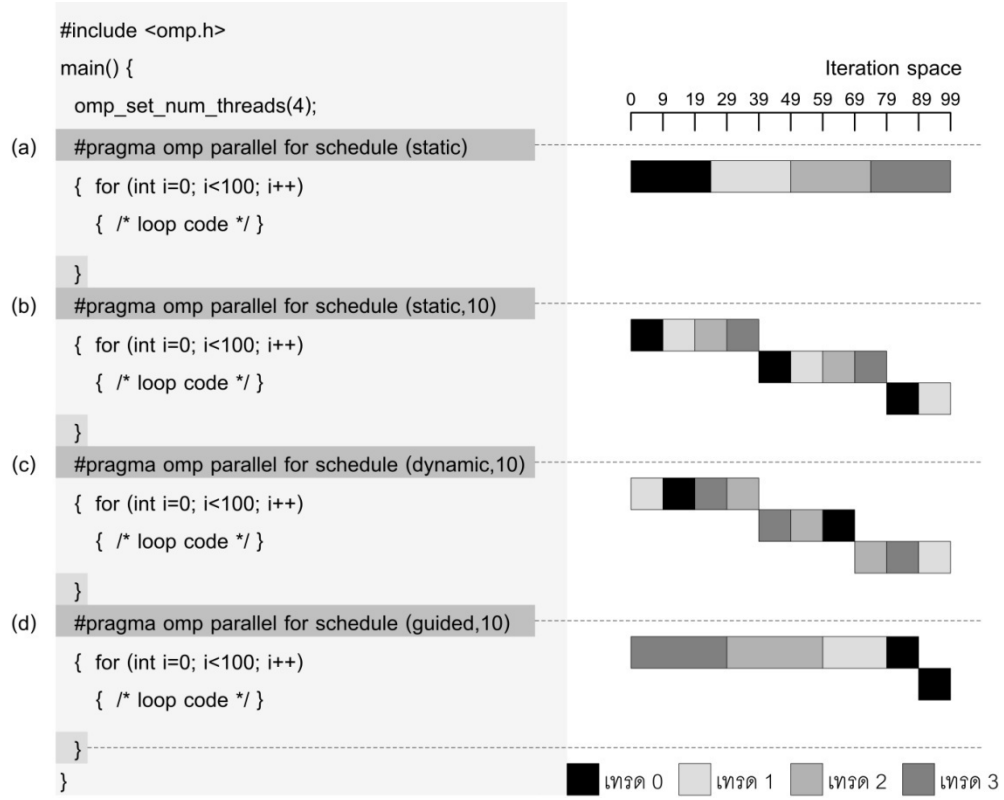
2.1.3 การแจกตัวนับลูปเพื่อประมวลผลแบบขนาน

ดังที่กล่าวในบทที่ 1 ว่าโอเพนเอ็มพีมีคำสั่งกำกับคอมไพเลอร์ที่ใช้กำหนดการแจกตัวนับลูปอยู่สองรูปแบบ ประกอบด้วย การแจกตัวนับลูปโดยไม่สร้างเทรดเพิ่ม และการแจกตัวนับลูปพร้อมทั้งสร้างเทรดเพิ่ม โปรแกรมเมอร์สามารถกำหนดการประมวลผลแบบขนานที่ระดับลูปโดยเพิ่มคำสั่งกำกับทั้งสองรูปแบบเหล่านี้ไว้เหนือส่วนงานประมวลผลในลูป เพื่อระบุให้คอมไพเลอร์ทราบว่า ส่วนงานประมวลผลในลูปที่อยู่ถัดไปสามารถประมวลผลแบบขนานได้

อย่างไรก็ตาม การแบ่งช่วงตัวนับลูปไปยังแต่ละเทรดควรมีจำนวนลูปที่สมดุลกัน (Load balancing) เพื่อให้การประมวลผลในลูปมีสมรรถนะดีที่สุด ด้วยเหตุนี้ โอเพนเอ็มพีจึงมีข้อกำหนด เพื่อใช้ระบุวิธีการแจกตัวนับลูปไปยังแต่ละเทรด (**`schedule`**) ซึ่งมีอยู่สามวิธีคือ การแจกส่วนแบบกำหนดตายตัว (**`static`**) การแจกส่วนเมื่อโปรแกรมรัน (**`dynamic`**) และการแจกส่วนแบบปรับขนาดตามข้อมูลเมื่อโปรแกรมรัน (**`guided`**) ในภาพที่ 2.4 แสดงลักษณะการแบ่งลูปจำนวน 100 ลูปไปยัง 4 เทรดของทั้งสามคำสั่ง ลูปประมวลผลจะถูกแบ่งออกเป็นหลายส่วนหรือบล็อก โปรแกรมเมอร์สามารถกำหนดจำนวนลูปในแต่ละบล็อกหรือขนาดของบล็อก (Chunk size) ไว้ในข้อกำหนดได้ แต่ถ้าหากไม่ได้ระบุไว้ ขนาดบล็อกเริ่มต้นของคำสั่ง **`static`** จะเท่ากับจำนวน

ลูปทั้งหมดหารด้วยจำนวนเทรด ส่วนในคำสั่ง **`dynamic`** และ **`guided`** ขนาดบล็อกเริ่มต้นเท่ากับหนึ่งลูป

ภาพที่ 2.4 รูปแบบการแบ่งช่วงตัวนับลูปของโปรแกรมแบบโอเพนเอ็มพี (a) คำสั่ง static ไม่กำหนด chunk size (b) คำสั่ง static chunk size = 10 (c) คำสั่ง dynamic chunk size = 10 และ (d) คำสั่ง guided chunk size = 10



ภาพที่ 2.4 (a) และ (b) แสดงวิธีการแจกบล็อกของลูปแบบกำหนดตายตัว ซึ่งบล็อกของลูปจะถูกส่งไปให้ทุกเทรดด้วยวิธีวนรอบ (Round robin) ตามลำดับของบล็อกและเทรด ในขณะที่การแจกส่วนเมื่อโปรแกรมรัน การแจกบล็อกของลูปจะไม่เป็นลำดับที่แน่นอน ขึ้นอยู่ที่ว่าเทรดใดว่างก็สามารถรับบล็อกของลูปไปประมวลผลได้ทันที ดังภาพที่ 2.4 (c) และวิธีการสุดท้ายการแจกส่วนแบบปรับขนาดตามข้อมูลเมื่อโปรแกรมรัน มีลักษณะการแจกบล็อกของลูปเช่นเดียวกับการแจกส่วนเมื่อโปรแกรมรัน แต่ขนาดของบล็อกในขณะที่ประมวลผลจะมีขนาดใหญ่กว่าที่กำหนดไว้ และจะลดขนาดลงจนกระทั่งเท่ากับขนาดของบล็อกที่กำหนดไว้ ดังภาพที่ 2.4 (d)

วิธีการแจกส่วนงานประมวลผลในลูปนั้น แต่ละวิธีจะมีความเหมาะสมกับคุณลักษณะของโปรแกรม และสถาปัตยกรรมของระบบคอมพิวเตอร์ที่แตกต่างกัน โปรแกรมเมอร์

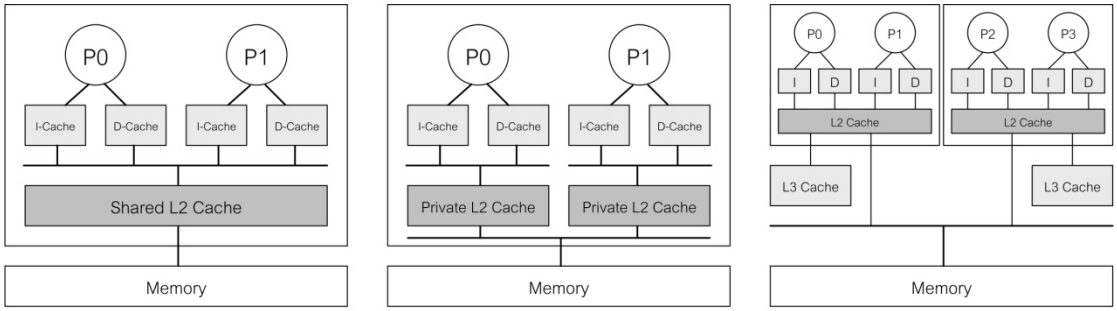
จึงต้องมีความเข้าใจในคุณลักษณะของโปรแกรมและสถาปัตยกรรมเป้าหมายด้วย โอเพนเอ็มพีซึ่งเป็นเอพีไอสำหรับการเขียนโปรแกรมเชิงขนาน บนสถาปัตยกรรมแบบใช้หน่วยความจำร่วม และรวมถึงชิปมัลติโพรเซสเซอร์ซึ่งเป็นหน่วยประมวลผลที่กำลังได้รับความนิยม ที่ถูกนำมาใช้บนคอมพิวเตอร์ส่วนบุคคลเพิ่มมากขึ้น ดังนั้นการศึกษาสถาปัตยกรรมของชิปมัลติโพรเซสเซอร์จะช่วยให้ โปรแกรมเมอร์รู้แนวทางการพัฒนาโปรแกรมแบบโอเพนเอ็มพี บนหน่วยประมวลผลดังกล่าวให้มีสมรรถนะการประมวลผลที่ดียิ่งขึ้น ในหัวข้อถัดไปจะกล่าวถึง รายละเอียดของสถาปัตยกรรมของหน่วยประมวลผลแบบชิปมัลติโพรเซสเซอร์

2.2 ชิปมัลติโพรเซสเซอร์

ปี พ.ศ.2508 กอร์ดอน มัวร์ได้ทำนายไว้ว่า “ในทุกสองปี ทรานซิสเตอร์ที่ถูกบรรจุลงหน่วยประมวลผลจะมีจำนวนเพิ่มขึ้นเป็นสองเท่า” (Grama *et al.*, 2003) มัวร์กล่าวคำทำนายดังกล่าวไว้ โดยพิจารณาจากแนวโน้มของเทคโนโลยีการผลิตทรานซิสเตอร์ทำให้มีขนาดเล็กลง จากจำนวนทรานซิสเตอร์ที่เพิ่มขึ้น ทำให้บริษัทผู้ผลิตหน่วยประมวลผลสามารถเพิ่มจำนวนหน่วยประมวลผลย่อยหลายหน่วย ลงบนแผงวงจรชิปเดียวกันได้ ซึ่งเรียกหน่วยประมวลผลชนิดนี้ว่า ชิปมัลติโพรเซสเซอร์ (CMPs ย่อมาจาก Chip multiprocessors) ปัจจุบันบริษัทผู้ผลิตหน่วยประมวลผลชั้นนำสามารถผลิตชิปมัลติโพรเซสเซอร์ ที่มีจำนวนหน่วยประมวลผลย่อยตั้งแต่ 2, 4, 8 และ 9 หน่วย และคาดว่าจะสามารถผลิตชิปมัลติโพรเซสเซอร์ที่มีหน่วยประมวลย่อย 16 หน่วย ได้ในปลายปี พ.ศ. 2552

ชิปมัลติโพรเซสเซอร์มีลักษณะสถาปัตยกรรมแบบใช้หน่วยความร่วม หน่วยประมวลผลย่อยทุกหน่วยจะอ้างถึงพื้นที่หน่วยความจำร่วมกันเพียงพื้นที่เดียว (Global address space) โดยทั่วไปชิปมัลติโพรเซสเซอร์มีแคชบนแผงวงจรชิป (On-chip cache) สองระดับคือ แคชระดับที่ 1 และแคชระดับที่ 2 ซึ่งหน่วยประมวลผลแต่ละหน่วยจะมีแคชระดับที่ 1 เป็นส่วนตัวแยกออกจากกัน โดยที่แคชระดับที่ 1 นี้จะแบ่งออกสองส่วน คือ แคชคำสั่ง (Instruction cache) และแคชข้อมูล (Data cache) ส่วนแคชระดับที่ 2 สามารถออกแบบให้ทุกหน่วยประมวลผลใช้ร่วมกัน ดังแสดงในภาพที่ 2.5 (a) หรือแต่ละหน่วยประมวลผลใช้แยกออกจากกันก็ได้ ดังภาพที่ 2.5 (b) นอกจากนี้ ชิปมัลติโพรเซสเซอร์บางรุ่น เช่น ไอบีเอ็ม พาวเวอร์ 5 ออกแบบให้มีแคชระดับที่ 3 วางอยู่นอกแผงวงจรชิป (Off-chip cache) เพื่อทำหน้าที่รับข้อมูลที่ถูกแทนที่ (Replacement) จากแคชระดับที่ 2 มาเก็บไว้ (Sinharoy *et al.*, 2005) ดังแสดงในภาพที่ 2.5 (c)

ภาพที่ 2.5 สถาปัตยกรรมของชิปมัลติโพรเซสเซอร์สามรูปแบบ (a) แบบมี 2 หน่วยประมวลผลใช้แคชระดับที่ 2 ร่วมกัน (b) แบบมี 2 หน่วยประมวลผลใช้แคชระดับที่ 2 แยกกันเป็นส่วนตัว และ (c) มี 4 หน่วยประมวลผลและใช้แคชระดับที่ 3 นอกแผงวงจรชิป



(a) CMPs 2 cores with shared L2 cache (b) CMPs 2 cores with private L2 cache (c) CMPs 4 cores with off-chip L3 cache

ตารางที่ 2.1 สถาปัตยกรรมของแคชบนชิปมัลติโพรเซสเซอร์ 5 รุ่น

Characteristics Of CMPs	AMD Athlon X2	IBM Power5	Intel Core 2 Quad	Sun UltraSPARC T1	Cell Multiprocessors[2]
Number of Cores	2	2	4	8	9
Inst.-L1 - size per core	64 KB	64 KB	32 KB	16 KB	32 KB
block size	64 bytes	128 bytes	64 bytes	32 bytes	128 bytes
set associative	8-way	8-way	8-way	8-way	8-way
Data-L1 - size per core	64 KB	32 KB	32 KB	8 KB	32 KB
block size	64 bytes	128 bytes	64 bytes	16 bytes	128 bytes
set associative	8-way	8-way	8-way	8-way	8-way
L2 - size / shared	1MB / Private	2 MB / Shared	8 MB/Shared	3 MB / Shared	512 KB / Shared
block size	64 bytes	128 bytes	64 bytes	64 bytes	128 bytes
set associative	16-way	8-way	8-way	8-way	8-way
Topology	Bus	Bus	Bus	Bus	Ring
Cache Coherence	MOESI	Bus-based	-	Bus-based	-
Level	L2	L1		L1	

จากตารางที่ 2.1 แสดงถึงความหลากหลายของสถาปัตยกรรมของแคชบนชิปมัลติโพรเซสเซอร์ที่จำหน่ายในท้องตลาดทั้งห้ารุ่น จะเห็นได้ว่าโดยส่วนใหญ่แคชระดับที่ 1 มีขนาด 32 กิโลไบต์ มีขนาดของแถวที่ 64 ไบต์ และมีจำนวนเซตที่ 8 เซต ส่วนแคชระดับที่ 2 มีขนาดตั้งแต่

[2] Cell มีหน่วยประมวลผลสองชนิด คือ PPE (Power Processor Element) จำนวน 1 หน่วย มีสถาปัตยกรรมของแคชดังที่ระบุไว้ในตาราง และ SPE (Synergistic Processor Element) จำนวน 8 หน่วย ซึ่งไม่มีแคช

512 กิโลไบต์ ไปจนถึง 8 เมกะไบต์ ซึ่งโดยส่วนใหญ่จะถูกออกแบบให้ทุกหน่วยประมวลผลใช้ร่วมกัน โดยมีขนาดของแถวที่ 64 ไบต์ และมีจำนวนเซต 8 เซตเช่นเดียวกับแคชระดับที่ 1

การออกแบบสถาปัตยกรรมและการทำงานของแคชนั้น มีองค์ประกอบหลายส่วนที่ส่งผลต่อสมรรถนะการประมวลผลของโปรแกรม ซึ่งในหัวข้อถัดไปจะกล่าวถึงองค์ประกอบของแคชเหล่านั้น

2.2.1 ลักษณะพื้นฐานของแคช

แคชเป็นหน่วยความจำขนาดเล็ก ซึ่งจะถูกวางให้อยู่ใกล้หน่วยประมวลผลมากที่สุด ทำหน้าที่พักข้อมูลจากหน่วยความจำหลักก่อนที่จะนำส่งต่อไปยังหน่วยประมวลผล เพื่อลดเวลาการเข้าถึงข้อมูลบนหน่วยความจำหลักของหน่วยประมวลผล แคชจะถูกแบ่งออกเป็นแถวหรือบล็อก (Cache block) ซึ่งขนาดข้อมูลที่ถูกเคลื่อนย้ายระหว่างแคชกับหน่วยความจำหลักนั้น จะเท่ากับขนาดของบล็อกบนแคช (Block size) แคชจะคอยรับคำร้องขอข้อมูลของหน่วยประมวลผลที่ต้อง การอ่านหรือเขียนข้อมูลเข้ามา และตรวจสอบว่าในแคชแต่ละบล็อกมีข้อมูลที่หน่วยประมวลผลต้องการหรือไม่ หากบนแคชมีข้อมูลที่หน่วยประมวลผลต้องการ ทำให้เกิดการพบข้อมูลในแคช (Cache hit) จากนั้นแคชจะส่งหรือเปลี่ยนแปลงค่าของข้อมูลตามที่หน่วยประมวลผลร้องขอ ในขณะที่หากบนแคชไม่มีข้อมูลดังกล่าว จะทำให้เกิดการไม่พบข้อมูลในแคช (Cache misses) แคชจะส่งคำร้องขอไปยังหน่วยความจำหลักเพื่อให้ดำเนินการตามที่หน่วยประมวลผลร้องขอมา การเข้าถึงข้อมูลบนหน่วยความจำหลักต้องใช้เวลานาน ทำให้ระบบคอมพิวเตอร์สูญเสียสมรรถนะการประมวลผล ดังนั้นประสิทธิภาพการทำงานของแคชจึงขึ้นอยู่กับอัตราการพบหรือไม่พบข้อมูลในแคช

อัตราการพบหรือไม่พบข้อมูลในแคชนั้น นอกจากจะใช้เป็นข้อมูลสะท้อนสมรรถนะการประมวลผลของโปรแกรมแล้ว สาเหตุหรือชนิดของการไม่พบข้อมูลยังบ่งบอกถึงคุณลักษณะของโปรแกรมได้อีกด้วย ซึ่งสาเหตุหรือชนิดการไม่พบข้อมูลในแคช สามารถจำแนกได้ออกเป็นสี่ชนิด คือ *การเข้าถึงข้อมูลครั้งแรก* (Compulsory miss) *การใช้พื้นที่แคชเต็ม* (Capacity miss) *การขัดแย้งของข้อมูล* (Conflict miss) และ *การใช้ข้อมูลร่วมกัน* (Coherence miss)

การไม่พบข้อมูลเนื่องจากการเข้าถึงข้อมูลครั้งแรก เกิดขึ้นเมื่อหน่วยประมวลผลต้องการเข้าถึงข้อมูลในแคชบล็อกเป็นครั้งแรก ซึ่งแคชบล็อกดังกล่าวยังว่างอยู่ไม่มีข้อมูลเก็บไว้ จึงทำให้ไม่พบข้อมูลที่ต้องการ ส่วนการไม่พบข้อมูลชนิดใช้พื้นที่แคชเต็ม เกิดขึ้นเนื่องจากโปรแกรมที่

กำลังประมวลผลในขณะนั้น ต้องการพื้นที่สำหรับจัดเก็บข้อมูลในการประมวลผล (Working set) มีขนาดใหญ่กว่าพื้นที่แคช ทำให้ไม่สามารถนำข้อมูลมาเก็บไว้บนแคชได้พร้อมกันทั้งหมด จึงทำให้ไม่พบข้อมูลในส่วนที่ไม่ได้จัดเก็บไว้ และการไม่พบข้อมูลเนื่องจากการขัดแย้งของข้อมูล เกิดขึ้นเนื่องจาก การส่งข้อมูลมายังแคชนั้นข้อมูลจะถูกเก็บไว้บนแคชบล็อกที่กำหนดไว้ ตามวิธีการส่งข้อมูล (Mapping) ซึ่งแต่ละแคชบล็อกสามารถเก็บข้อมูลจากหน่วยความจำหลักได้หลายตำแหน่ง (Address) ดังนั้นเมื่อมีข้อมูลใหม่เข้ามาเก็บไว้ในแคชบล็อกที่มีข้อมูลเก็บอยู่ ข้อมูลเดิมจะถูกแทนที่ด้วยข้อมูลใหม่ ต่อมาหากหน่วยประมวลผลต้องการใช้ข้อมูลเดิมประมวลผล จะทำให้ไม่พบข้อมูลในแคชเนื่องจากการขัดแย้งของข้อมูล

การไม่พบข้อมูลในแคชที่กล่าวมาทั้งสามชนิด เป็นการไม่พบข้อมูลที่สามารถเกิดขึ้นได้ ทั้งในระบบคอมพิวเตอร์หน่วยประมวลผลเดียว และระบบคอมพิวเตอร์เชิงขนาน ส่วนการไม่พบข้อมูลในแคชเนื่องจากการใช้ข้อมูลร่วมกัน เป็นการไม่พบข้อมูลที่จะเกิดขึ้นบนระบบคอมพิวเตอร์เชิงขนานเท่านั้น ซึ่งเป็นผลอันเนื่องมาจากกลไกรักษาความสอดคล้องของข้อมูลในแคช โดยจะกล่าวถึงรายละเอียดในหัวข้อถัดไป

2.2.2 ความสอดคล้องของข้อมูลในแคช

บนระบบคอมพิวเตอร์เชิงขนานที่แต่ละหน่วยประมวลผลมีแคชแยกเป็นส่วนตัวนั้น หน่วยประมวลผลแต่ละหน่วยสามารถดึงข้อมูลหรือตัวแปรในหน่วยความจำ มาเก็บไว้ที่แคชส่วนตัวได้ ทำให้มีโอกาสที่ตัวแปรเดียวกันถูกทำสำเนาหลายชุด (Replicas) กระจายอยู่บนแคชของหลายหน่วยประมวลผล เมื่อมีหน่วยประมวลผลหน่วยใดหน่วยหนึ่งเปลี่ยนแปลงค่าของตัวแปรบนแคชของตัวเอง ส่งผลให้เกิดปัญหาค่าของตัวแปรดังกล่าวมีค่าไม่ตรงกับตัวแปรเดียวกัน ที่อยู่บนหน่วยความจำ และบนแคชของหน่วยประมวลผลอื่น ปัญหาดังกล่าวเรียกว่า *ปัญหาเกี่ยวกับความสอดคล้องของข้อมูลในแคช* (Cache coherence problem) แคชจึงมีกลไกทางฮาร์ดแวร์ภายในแคชคอนโทรลเลอร์ ที่ทำหน้าที่ควบคุมและรักษาความสอดคล้องของข้อมูลในแคชให้มีค่าตรงกัน (Cache coherence protocol) ซึ่งกลไกดังกล่าวนี้สามารถแบ่งออกได้สองกลุ่ม คือ *โพรโทคอลแบบบัส* (Bus-based protocol) และ*โพรโทคอลแบบสารบบ* (Directory-based protocol)

โพรโทคอลแบบบัสเป็นโพรโทคอลที่ใช้บนระบบคอมพิวเตอร์เชิงขนาน ที่หน่วยประมวลผลเชื่อมต่อกันแบบบัส (Bus topology) การเชื่อมต่อแบบบัสนั้น ทุกหน่วยประมวลผลจะคอยตรวจสอบเหตุการณ์ที่เกิดบนบัสอยู่ตลอดเวลา หรือเรียกว่าการสนูป (Snoop) ดูข้อมูล ทำให้

บางครั้งเรียกโพรโทคอลนี้ว่า สนูปี (Snoopy protocol) เมื่อมีหน่วยประมวลผลเปลี่ยนแปลงค่าของตัวแปรบนแคชของตัวเอง หน่วยประมวลผลดังกล่าวจะแจ้งการเปลี่ยนแปลงนั้นไปยังบัส หน่วยประมวลผลอื่นจะทราบการเปลี่ยนแปลงผ่านทางบัสอีกที อย่างไรก็ตาม โพรโทคอลแบบบัสนี้ ระบบบัสต้องสื่อสารกับหน่วยประมวลผลทุกหน่วยอยู่ตลอดเวลา ทำให้บัสต้องมีช่องทางหรือแบนด์วิดท์ (Bandwidth) ของการสื่อสารที่สามารถรองรับกับจำนวนหน่วยประมวลผลในระบบได้

ส่วนโพรโทคอลแบบสารบบนั้น แคชจะมีส่วนที่ใช้เก็บข้อมูลเพื่อให้ทราบว่า หน่วยประมวลผลใดบ้างที่เก็บสำเนาของตัวแปรชุดเดียวกันไว้บ้าง เมื่อมีหน่วยประมวลผลเปลี่ยนแปลงค่าของตัวแปรบนแคชของตนเอง หน่วยประมวลผลดังกล่าวจะแจ้งการเปลี่ยนแปลงไปยังหน่วยประมวลผลอื่นที่เก็บสำเนาของตัวแปรที่ถูกเปลี่ยนแปลงโดยตรง ซึ่งหน่วยประมวลผลที่ไม่ได้เก็บตัวแปรดังกล่าวไม่จำเป็นต้องรับรู้การเปลี่ยนแปลงที่เกิดขึ้น โพรโทคอลแบบสารบบเหมาะสำหรับระบบคอมพิวเตอร์เชิงขนานขนาดใหญ่ ที่มีหน่วยประมวลผลตั้งแต่ 16 หน่วยขึ้นไป เพื่อช่วยลดความคับคั่งของการสื่อสารภายในเครือข่ายได้

ในปัจจุบันชิปมัลติโพรเซสเซอร์ที่มีหน่วยประมวลผลน้อยกว่า 16 หน่วยนั้น มักจะใช้รูปแบบการเชื่อมโยงแบบบัส และใช้กลไกรักษาความสอดคล้องของข้อมูลด้วยโพรโทคอลแบบบัสเป็นส่วนใหญ่ดังแสดงไว้ในตารางที่ 2.1 เมื่อเกิดการเปลี่ยนแปลงค่าของตัวแปร โพรโทคอลแบบบัสจะมีวิธีจัดการเพื่อให้สำเนาตัวแปรทุกสำเนามีค่าสอดคล้องกัน หรือที่เรียกว่า *นโยบายความสอดคล้องข้อมูล* (Coherence policy) แบ่งออกเป็นสองวิธี คือ *การปรับข้อมูล* (Update) และ *การยกเลิกข้อมูล* (Invalidate) การทำงานของนโยบายการปรับข้อมูลนั้น เมื่อหน่วยประมวลผลตรวจสอบบนบัสแล้วพบว่าเกิดการเปลี่ยนแปลงค่าของตัวแปรที่ตนเองเก็บไว้ ก็จะปรับค่าให้ตรงกับค่าใหม่ที่เปลี่ยนแปลงทันที ในขณะที่นโยบายการยกเลิกข้อมูลนั้น หน่วยประมวลผลจะไม่ปรับค่าของตัวแปรที่ถูกเปลี่ยนแปลง แต่จะยกเลิกสำเนาของตัวแปรที่ตนเก็บไว้ไม่ให้ถูกนำมาใช้

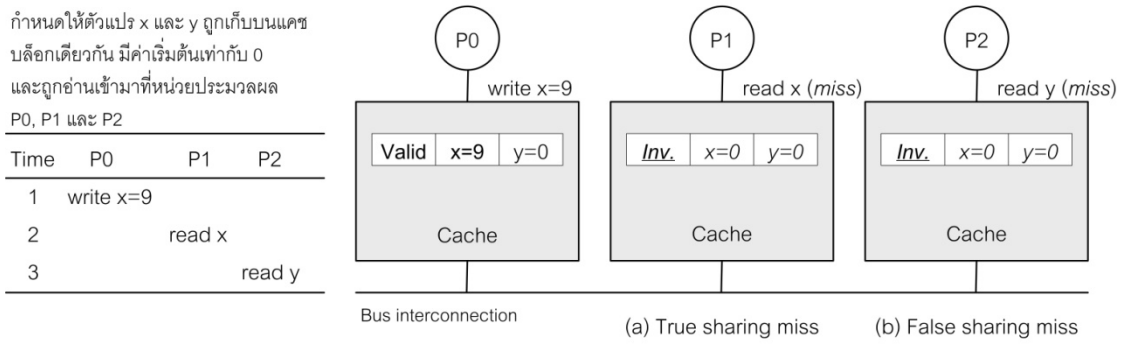
เมื่อมีหน่วยประมวลผลใดเขียนค่าของตัวแปร หน่วยประมวลผลอื่นที่ถือสำเนาของตัวแปรนั้นจะต้องยกเลิกสำเนาที่ตนเก็บไว้ ในการนี้วงจรควบคุมแคชจะกำหนดให้สถานะของแคชบล็อกที่เก็บสำเนาตัวแปรนั้นมีสถานะเป็นใช้งานไม่ได้ (Invalid) ส่งผลให้ตัวแปรหรือข้อมูลทั้งหมดที่ถูกเก็บไว้ในแคชบล็อกนั้นถูกยกเลิกไป ซึ่งหากต่อมาหน่วยประมวลผลที่เพิ่งยกเลิกสำเนาต้องการอ่านหรือเขียนข้อมูลใดในแคชบล็อกดังกล่าว ก็จะไม่พบข้อมูลในแคช สถานการณ์นี้จัดเป็น *การไม่พบข้อมูลเนื่องจากใช้ข้อมูลร่วมกัน* (Cache sharing miss)

การไม่พบข้อมูลเนื่องจากใช้ข้อมูลร่วมกัน เกิดจากการที่หน่วยประมวลผลหนึ่งไม่พบตัวแปรที่ต้องการ เพราะแคชบล็อกที่เก็บตัวแปรนั้นเพิ่งถูกยกเลิกไป การไม่พบข้อมูลลักษณะนี้

จำแนกเป็นสองกรณี ตามสาเหตุของการยกเลิกแคชบล็อกที่เก็บตัวแปรที่ต้องการ โดยหากการยกเลิกนั้น เกิดเพราะหน่วยประมวลผลอื่นเขียนค่าลงบนตัวแปรที่ต้องการเช่นกัน จัดเป็น *กรณีร่วมใช้จริง* (True sharing miss) แต่หากการยกเลิกแคชบล็อกเป็นการพลอยถูกยกเลิก เพราะหน่วยประมวลผลอื่นเขียนค่าลงบนตัวแปรอื่นที่เก็บอยู่บนแคชบล็อกเดียวกันกับตัวแปรที่ต้องการ ทำให้ตัวแปรนี้พลอยถูกยกเลิกไปด้วยเช่นนี้ จัดเป็น *กรณีร่วมใช้ไม่จริง* (False sharing miss)

ภาพที่ 2.6 แสดงตัวอย่างของชิปมัลติโพรเซสเซอร์ที่มีสามหน่วยประมวลผล (P0, P1 และ P2) โดยสมมติให้ทุกหน่วยประมวลผลเริ่มต้นมีสำเนาของตัวแปร x และ y เก็บที่แคช และทั้งสองตัวแปรเก็บอยู่ในแคชบล็อกเดียวกัน ภาพที่ 2.6 (a) แสดงตัวอย่างกรณีร่วมใช้จริงระหว่างหน่วยประมวลผล P0 และ P1 ในการเข้าถึงตัวแปร x ในลำดับเวลาที่ 1 หน่วยประมวลผล P0 จะเขียนค่าตัวแปร x เป็น 9 จึงประกาศลงบัส แคชของ P1 และ P2 ต้องยกเลิกสำเนาของแคชบล็อกที่เก็บตัวแปร x ที่ตนถืออยู่ ในลำดับเวลาที่ 2 เมื่อ P1 ต้องการอ่านค่าของตัวแปร x ก็จะไม่พบข้อมูลที่แคช เพราะบล็อกของตนเพิ่งยกเลิกไป กรณีนี้ P1 ไม่พบข้อมูล x เพราะ P0 เพิ่งเขียน ตัวแปร x มีการร่วมใช้ จึงจัดเป็นการไม่พบข้อมูลกรณีร่วมใช้จริง ในภาพที่ 2.6 (b) แสดงตัวอย่างกรณีร่วมใช้ไม่จริง ระหว่าง P0 และ P2 กล่าวคือ ในลำดับเวลาถัดมา หาก P2 ต้องการอ่านค่าตัวแปร y จะไม่พบข้อมูลที่แคช เพราะบล็อกที่เก็บตัวแปร y เพิ่งถูกยกเลิกไป แต่การยกเลิกนี้เกิดจาก P0 เขียน x ไม่ได้เกี่ยวข้องกับ y แต่เพราะ x และ y อยู่ในแคชบล็อกเดียวกันจึงทำให้ y พลอยถูกยกเลิกไปด้วย ดังนั้นการที่ P2 ไม่พบ y ที่แคชของตนนี้ เรียกว่าเป็นการไม่พบเนื่องจากกรณีร่วมใช้ไม่จริง

ภาพที่ 2.6 การไม่พบข้อมูลในแคชเนื่องจาก (a) ข้อมูลร่วมใช้จริง และ (b) ข้อมูลร่วมใช้ไม่จริง



การไม่พบข้อมูลในแคชเนื่องจากใช้ข้อมูลร่วมกันทั้งสองกรณี เกิดขึ้นเนื่องจากกลไกรักษาความสอดคล้องของข้อมูลในแคช ซึ่งส่งผลต่อสมรรถนะการประมวลผลของโปรแกรม หากลดการไม่พบข้อมูลทั้งสองชนิดนี้ได้ โดยเฉพาะอย่างยิ่งกรณีการร่วมใช้ไม่จริง จะทำให้ลดจำนวนการไม่พบข้อมูล ลดเวลาการรอข้อมูลจากหน่วยความจำและเวลาในการควบคุมความสอดคล้อง

ลง โปรแกรมจะประมวลผลได้เร็วขึ้น นอกจากนี้ ลำดับและความถี่ในการเข้าถึงพื้นที่เก็บข้อมูลของหน่วยประมวลผล เป็นคุณสมบัติอีกประการหนึ่งที่ส่งผลต่อการใช้ประโยชน์พื้นที่แคช ซึ่งจะได้กล่าวถึงรายละเอียดในหัวข้อถัดไป

2.2.3 **คุณสมบัติของพื้นที่เก็บข้อมูล**

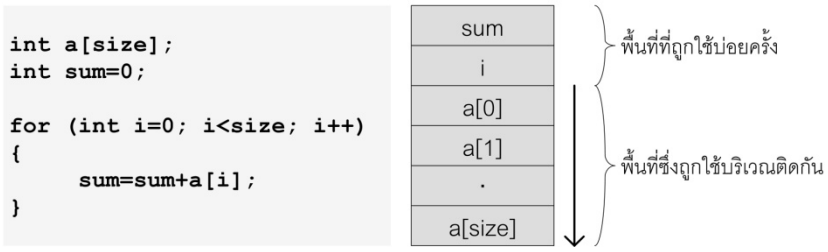
จากเป้าหมายการใช้งานแคชที่ใช้เป็นหน่วยพักข้อมูลก่อนที่จะถูกนำไปประมวลผล หากแคชมีพื้นที่ที่สามารถรองรับข้อมูลทั้งหมดที่ใช้ในโปรแกรม หรือหากแคชไม่สามารถจัดเก็บข้อมูลได้ทั้งหมด แต่สามารถจัดเตรียมข้อมูลให้ได้ตรงตามลำดับการเรียกใช้ข้อมูลของโปรแกรมได้ จะทำให้โปรแกรมมีสมรรถนะการประมวลผลที่เร็วขึ้น เพราะหน่วยประมวลผลสามารถดึงข้อมูลจากแคชไปใช้ได้ทันที การวิเคราะห์ถึงความต้องการพื้นที่สำหรับจัดเก็บข้อมูลในการประมวลผล และลำดับการเข้าถึงพื้นที่เก็บข้อมูลของโปรแกรม รวมถึงความถี่ในการเข้าถึงพื้นที่เก็บข้อมูลในแต่ละพื้นที่ จะช่วยให้เข้าใจถึงความสัมพันธ์ระหว่างการเรียกใช้ข้อมูลของโปรแกรมกับพื้นที่เก็บข้อมูลได้ดียิ่งขึ้น

คุณสมบัติของพื้นที่เก็บข้อมูล (Data locality) ซึ่งเป็นคุณสมบัติที่แสดงถึงการใช้พื้นที่เก็บข้อมูล ลำดับการเข้าถึงและความถี่การเข้าถึงพื้นที่เก็บข้อมูลของโปรแกรมบนหน่วยความจำ คุณสมบัติดังกล่าวนี้แบ่งออกเป็นสองลักษณะ คือ *พื้นที่ที่ถูกใช้บ่อยครั้ง* (Temporal locality) และ *พื้นที่ซึ่งถูกใช้บริเวณติดกัน* (Spatial locality) โดยพื้นที่เก็บข้อมูลที่ถูกใช้บ่อยครั้ง หมายถึง พื้นที่หรือตำแหน่งในหน่วยความจำ ซึ่งใช้เก็บข้อมูลที่ถูกหน่วยประมวลผลนำมาประมวลผลบ่อยครั้ง หากข้อมูลที่อยู่ในพื้นที่ดังกล่าวนี้ถูกนำมาเก็บไว้บนแคช จะใช้ให้อัตราการพบข้อมูลในแคชเพิ่มขึ้นด้วย ส่วนพื้นที่เก็บข้อมูลซึ่งถูกใช้บริเวณติดกัน หมายถึง พื้นที่ที่ใช้เก็บข้อมูลหรือตัวแปรหลายตัว ซึ่งตัวแปรเหล่านั้นอาจถูกจัดเก็บไว้เป็นโครงสร้าง เช่น อาร์เรย์ และถูกหน่วยประมวลผลเข้าถึงเป็นลำดับติดกันอย่างต่อเนื่อง คุณสมบัติดังกล่าวนี้หน่วยประมวลผลจะเข้าถึงพื้นที่ในหน่วย ความจำที่อยู่ติดกันและขยายพื้นที่ออกไป ซึ่งหากแคชสามารถบรรจุข้อมูลบนพื้นที่นี้ได้ทั้งหมด หรือสามารถจัดเตรียมข้อมูลเข้ามาให้แคชให้สอดคล้องกับลำดับการเข้าถึงของหน่วยประมวลผล จะทำให้โปรแกรมประมวลผลได้เร็วขึ้น

การประมวลผลในลูปเป็นการประมวลผลที่แสดงให้เห็นถึงคุณสมบัติของพื้นที่เก็บข้อมูลได้อย่างชัดเจน จากตัวอย่างการประมวลผลในลูป (ภาพที่ 2.7) ตัวแปร sum และ i เป็นตัวแปรที่ถูกหน่วยประมวลผลต้องเรียกใช้ทุกครั้งในการประมวลผลแต่ละรอบ จากลักษณะดังกล่าวนี้

พื้นที่ที่ใช้เก็บตัวแปร sum และ i แสดงถึงคุณสมบัติของพื้นที่ที่ถูกใช้บ่อยครั้ง ส่วนตัวแปรอาร์เรย์ a จะถูกหน่วยประมวลผลเข้าถึงข้อมูลในพื้นที่ติดกันอย่างเป็นลำดับ ตามดรรชนีของอาร์เรย์ ลักษณะดังกล่าวนี้ พื้นที่ที่ใช้เก็บตัวแปร a มีคุณสมบัติของพื้นที่ซึ่งถูกใช้บริเวณติดกัน ซึ่งทั้งตัวแปร sum, i และ a นี้ หากสามารถนำเข้าบรรจุไว้ในแคช หรือจัดเตรียมให้สอดคล้องกับลำดับการเรียกใช้ จะช่วยเพิ่มสมรรถนะการประมวลผลของโปรแกรมได้

ภาพที่ 2.7 การประมวลผลในลูป ซึ่งพื้นที่เก็บตัวแปร sum และ i มีคุณสมบัติถูกใช้บ่อยครั้ง ส่วนตัวแปร a มีคุณสมบัติพื้นที่ซึ่งถูกใช้บริเวณติดกัน



การประมวลผลในลูปบนระบบคอมพิวเตอร์หน่วยประมวลผลเดี่ยวนั้น ทั้งตัวแปรที่ใช้ตรวจสอบเงื่อนไขและตัวแปรที่ถูกจัดเก็บเป็นโครงสร้าง สามารถถูกดึงเข้าไปสู่แคชได้ตามลำดับการเข้าถึงของหน่วยประมวลผลอย่างตรงไปตรงมา ซึ่งแตกต่างกับการประมวลผลลูปบนระบบคอมพิวเตอร์เชิงขนาน ที่ลูปประมวลผลจะถูกแบ่งและกระจายไปประมวลผลบนหลายหน่วยประมวลผลหรือหลายเทรด ตัวแปรที่ถูกจัดเก็บเป็นโครงสร้างจะถูกแบ่งและกระจายไปประมวลผลด้วยเช่นกัน ซึ่งการแบ่งตัวแปรเหล่านี้จะส่งผลต่อคุณสมบัติของพื้นที่เก็บข้อมูล และส่งสมรรถนะการประมวลผลของโปรแกรมด้วย ดังนั้นการวิเคราะห์ถึงคุณสมบัติของพื้นที่เก็บข้อมูล จะช่วยให้เห็นการใช้ประโยชน์พื้นที่แคชที่มีผลต่อสมรรถนะของโปรแกรมได้ดียิ่งขึ้น

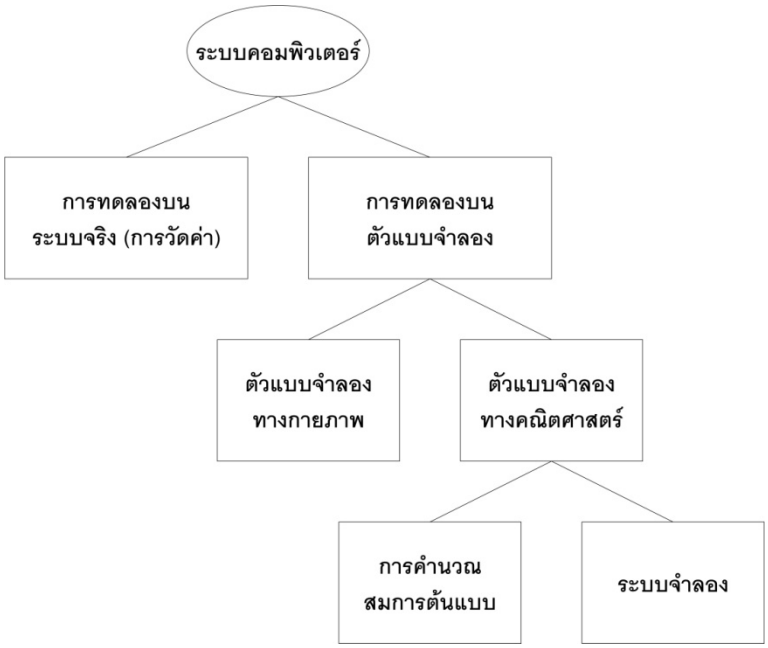
จากที่กล่าวมาข้างต้น การใช้ประโยชน์พื้นที่แคชและประสิทธิภาพการทำงานของแคช สามารถสะท้อนข้อมูลเชิงสมรรถนะได้หลายลักษณะ เช่น อัตราการพบหรือไม่พบข้อมูลของแคช ชนิดการไม่พบข้อมูล และคุณสมบัติของพื้นที่เก็บข้อมูล ซึ่งเทคนิคการวิเคราะห์สมรรถนะบนระบบคอมพิวเตอร์ เพื่อให้ได้มาซึ่งข้อมูลเหล่านี้มีอยู่ด้วยกันหลายเทคนิค โดยในหัวข้อถัดไปจะกล่าวถึง เทคนิคการวิเคราะห์สมรรถนะด้วยระบบจำลอง

2.3 เทคนิคการวิเคราะห์ด้วยระบบจำลอง

การพัฒนาระบบคอมพิวเตอร์เพื่อให้มีสมรรถนะการประมวลผลที่สูงขึ้นนั้น การวิเคราะห์สมรรถนะเป็นกระบวนการที่สำคัญอย่างหนึ่ง เพื่อให้ได้ข้อมูลและคุณลักษณะที่สามารถสะท้อนสมรรถนะของระบบคอมพิวเตอร์ และนำข้อมูลเหล่านั้นไปปรับปรุงสมรรถนะของระบบคอมพิวเตอร์ ด้วยเหตุนี้จึงมีการพัฒนาและนำเสนอเทคนิคการวิเคราะห์สมรรถนะที่เหมาะสมกับระบบคอมพิวเตอร์เป้าหมายแต่ละระบบ

เทคนิคการวิเคราะห์สมรรถนะบนระบบคอมพิวเตอร์ รวมถึงระบบคอมพิวเตอร์เชิงขนาน สามารถแบ่งออกเป็นสามเทคนิค ประกอบด้วย *การคำนวณสมการต้นแบบ* (Analytical modeling) *ระบบ*จำลอง (Simulation) และ*การวัดค่า* (Measurement) (Jain, 1991) จากภาพที่ 2.8 แสดงถึงความแตกต่างของเทคนิควิเคราะห์สมรรถนะทั้งสามเทคนิค ซึ่งจะเห็นได้ว่าการคำนวณสมการต้นแบบและระบบจำลองเป็นเทคนิคที่ใช้ตัวแบบจำลอง (Model) เข้ามาเกี่ยวข้อง ในขณะที่เทคนิคการวัดค่าเป็นการวิเคราะห์สมรรถนะบนระบบคอมพิวเตอร์จริง

ภาพที่ 2.8 แนวทางการวิเคราะห์สมรรถนะระบบคอมพิวเตอร์ (Averill & David, 1999)



จากตารางที่ 2.2 แสดงเกณฑ์ในการเปรียบเทียบให้เห็นถึงข้อดีและข้อด้อยของทั้งสามเทคนิค ซึ่งจะเห็นได้ว่าเทคนิคการคำนวณสมการต้นแบบนั้น เป็นเทคนิคที่ทำได้ง่าย ต้นทุนต่ำ ใช้เวลาในการวิเคราะห์น้อย สามารถใช้ได้ทุกช่วงระยะเวลาการพัฒนาระบบคอมพิวเตอร์ แต่ทว่า

มีความถูกต้องของข้อมูลต่ำ ส่วนเทคนิคการวัดค่า เป็นเทคนิคที่ทำได้ยาก ต้นทุนสูง อาจใช้เวลาในการวิเคราะห์มากน้อยต่างกัน ซึ่งเช่นเดียวกับความถูกต้องของข้อมูลขึ้นอยู่กับสภาพแวดล้อมของระบบคอมพิวเตอร์ในการวัดค่าแต่ละครั้งด้วย นอกจากนี้ยังใช้ได้เฉพาะระบบคอมพิวเตอร์ที่ถูกพัฒนาขึ้นมาแล้วเท่านั้น ในขณะที่การวิเคราะห์สมรรถนะด้วยระบบจำลอง เป็นเทคนิคที่มีความยุ่งยากปานกลาง ใช้ต้นทุนและเวลาในการวิเคราะห์ปานกลาง สามารถใช้ได้ทุกช่วงระยะเช่นเดียวกับการคำนวณสมการต้นแบบ แต่ให้ความถูกต้องของข้อมูลที่สูงกว่า

จากข้อได้เปรียบของการวิเคราะห์สมรรถนะด้วยระบบจำลอง ทำให้ถูกนำมาใช้ในงานวิจัยเพิ่มมากขึ้น สกาดอร์นและคณะได้สำรวจงานวิจัยที่นำเสนอในงานประชุมที่เกี่ยวกับสถาปัตยกรรมระบบคอมพิวเตอร์[3] พบว่าในปี พ.ศ.2544 มีงานวิจัยที่นำโปรแกรมระบบจำลองมาใช้ในงานวิจัยเพิ่มขึ้นจากปี พ.ศ.2528 ถึงร้อยละ 62 และยังชี้ให้เห็นว่ามีแนวโน้มที่จะเพิ่มขึ้นจนถึงปัจจุบัน (Skadron *et al*., 2003)

ตารางที่ 2.2 ข้อดีและข้อด้อยของเทคนิคการวิเคราะห์สมรรถนะสามเทคนิค (Jain, 1991)

เกณฑ์การเปรียบเทียบ	การคำนวณสมการต้นแบบ	ระบบจำลอง	การวัดค่า
1. ช่วงเวลาการพัฒนาระบบคอมพิวเตอร์	ทุกช่วงเวลา	ทุกช่วงเวลา	พัฒนาเสร็จแล้ว
2. เวลาที่ใช้ในการวิเคราะห์	น้อย	ปานกลาง	หลากหลาย
3. เครื่องมือการวิเคราะห์	นักวิเคราะห์	ภาษาโปรแกรม	การแทรกคำสั่ง
4. ความถูกต้องของข้อมูล	ต่ำ	ปานกลาง	หลากหลาย
5. ระดับความยาก	ง่าย	ปานกลาง	ยาก
6. ค่าใช้จ่าย	น้อย	ปานกลาง	สูง
7. การปรับขนาดของการวิเคราะห์	ต่ำ	ปานกลาง	สูง

ระบบจำลองหรือโปรแกรมระบบจำลอง (Simulator) ที่ใช้ในงานวิจัยทางคอมพิวเตอร์นั้น เป็นโปรแกรมที่ถูกพัฒนาขึ้นมาเพื่อจำลองลักษณะการทำงาน เหตุการณ์ และพฤติกรรมของระบบคอมพิวเตอร์เป้าหมาย การทำงานของโปรแกรมระบบจำลองจะขึ้นอยู่กับตัวแบบจำลอง

ตัวแบบจำลองต้องอธิบายคุณลักษณะการทำงานของระบบเป้าหมาย ในสี่คุณลักษณะ ประกอบด้วย *เวลา* (Time), *สถานะ* (State), *หน้าที่* (Functionality) และ *ความไม่แน่นอน* (Randomness) (Marurngsith, 2006) ซึ่งคุณลักษณะประการแรกอธิบายว่า ระบบ

[3] งานประชุม the International Symposium on Computer Architecture (ISCA)

เป้าหมายมีการทำงานตามจังหวะช่วงเวลาที่แน่นอนหรือไม่ (Time-varying และ Time invariant) หรือเหตุการณ์ที่เกิดขึ้นบนระบบเป้าหมายมีช่วงเวลาเกิดขึ้นที่แน่นอนหรือไม่ (Static model และ Dynamic model) คุณลักษณะประการที่สองอธิบายว่า เมื่อค่าของตัวแปรสถานะถูกเปลี่ยนแปลง ค่าที่เปลี่ยนไปมีความต่อเนื่องกับค่าเดิมหรือไม่ (Continuous state และ Discrete state) กล่าวคือ ค่าของตัวแปรสถานะในแต่ละสถานะนั้น มีลักษณะต่อเนื่องหรือแยกออกจากกัน คุณลักษณะประการที่สามอธิบายว่า ระบบเป้าหมายทำหน้าที่ตอบสนองต่อองค์ประกอบใด เช่น ทำหน้าที่ตอบสนองต่อเวลาที่เปลี่ยนไป (Time-driven) หรือตอบสนองต่อเหตุการณ์ที่เกิดขึ้น (Event-driven) และคุณลักษณะประการสุดท้ายอธิบายว่า เมื่อมีข้อมูลนำเข้าชุดเดิมหรือเหตุการณ์เกิดขึ้นในลักษณะเดิม มาเปลี่ยนแปลงค่าของตัวแปรสถานะของระบบเป้าหมายแล้ว ค่าของตัวแปรสถานะจะเปลี่ยนแปลงในลักษณะเดิมทุกครั้งหรือไม่ (Deterministic และ Stochastic) กล่าวคือ การเปลี่ยนแปลงค่าของตัวแปรสถานะสามารถทำนายได้ล่วงหน้าหรือไม่

ตัวแบบจำลองที่ใช้จำลองระบบคอมพิวเตอร์นั้น จัดเป็น *ระบบจำลองชนิดเหตุการณ์ไม่ต่อเนื่อง* (Discrete-event simulation) ซึ่งคาสแซนดราสและเลอฟอร์จูน (Cassandras & Lafortune, 2008) สรุปคุณลักษณะการทำงานของระบบจำลองชนิดเหตุการณ์ไม่ต่อเนื่องว่า สถานะของระบบ เป็นตัวแปรที่มีค่าไม่ต่อเนื่อง (Discrete state) ตัวระบบจะทำงานเมื่อมีเหตุการณ์มากระตุ้นและจะทำงานตอบสนองต่อเหตุการณ์นั้น (Event-driven) เหตุการณ์ที่เข้ามากระตุ้นระบบจะเกิดขึ้นในช่วงเวลาใดก็ได้ (Dynamic model) และพฤติกรรมที่ตอบสนองต่อเหตุการณ์นั้นไม่ได้ขึ้นกับเวลา กล่าวคือพฤติกรรมไม่ได้เปลี่ยนแปลงไปตามเวลา (Time invariant)

แบบจำลองระบบคอมพิวเตอร์เป็นเครื่องมือที่ใช้เก็บข้อมูลเกี่ยวกับพฤติกรรมของระบบคอมพิวเตอร์ในขณะที่ประมวลผลโปรแกรมใดโปรแกรมหนึ่ง ผลลัพธ์จากแบบจำลองมักจะเป็นข้อมูลที่อยู่ในรูปแบบของหน่วยวัด (Metrics) ซึ่งค่าของหน่วยวัดนั้นสามารถสะท้อนให้เห็นสมรรถนะการประมวลผลของโปรแกรมที่ศึกษาได้ ในหัวถัดไปจะกล่าวถึงหน่วยวัดที่พบว่าใช้เป็นค่าสะท้อนสมรรถนะของระบบคอมพิวเตอร์ได้

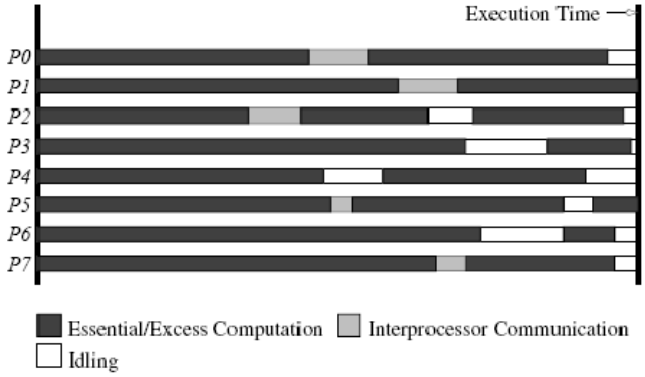
2.4 หน่วยวัดสมรรถนะและการใช้ประโยชน์พื้นที่แคช

หน่วยวัดสมรรถนะ (Performance metric) ที่แสดงสมรรถนะของ*ระบบคอมพิวเตอร์เชิงขนาน (Parallel System)* จะรวมทั้งคุณลักษณะของสถาปัตยกรรมของเครื่อง และการออกแบบอัลกอริทึม ในภาพรวมหน่วยวัดสมรรถนะของระบบเชิงขนาน แบ่งเป็นการวัดเชิงเวลาใน

รูปแบบของ*เวลาประมวลผล (Execution Time)* และการวัดเชิง*อัตราการผลิตผลลัพธ์* หรือปริมาณงานที่ผลิตได้ในหนึ่งหน่วยเวลา *(Throughput)* ซึ่งภายหลังจากการจับค่าดังกล่าวบนหน่วยประมวลผลจำนวนหลายหน่วยแล้ว มักจะนำค่าเหล่านั้นมาเปรียบเทียบกับระบบเชิงลำดับในรูปแบบของ อัตราการเร็วขึ้น หรือ *สปีดอัพ (Speedup)* สปีดอัพจะสะท้อนให้เห็นว่าเมื่อใช้หน่วยประมวลผลหลายหน่วยแล้วโปรแกรมประมวลผลเร็วขึ้นหรือมีอัตราการผลิตผลลัพธ์มากขึ้นจากระบบเชิงลำดับกี่เท่า เมื่อทราบค่าสปีดอัพแล้ว สามารถนำสปีดอัพมาหาค่าประสิทธิภาพ (Efficiency) และเวลาส่วนเกิน (Overhead) ได้ว่าขณะประมวลผลเชิงขนานแต่ละหน่วยประมวลผลได้เพิ่มสมรรถนะการประมวลผลไปกี่เท่า และเกิดเวลาส่วนเกินโดยรวมเท่าไร

เนื้อหาต่อไปนี้สรุปประเด็นสำคัญของหน่วยวัดสมรรถนะของระบบเชิงขนาน และกล่าวถึงการใช้ประโยชน์พื้นที่แคชที่ส่งผลต่อสมรรถนะของโปรแกรมไว้ในส่วนท้าย เนื้อหาในส่วนหน่วยวัดสมรรถนะนี้เรียบเรียงและสรุปจาก (Grama *et al.*, 2003)

ภาพที่ 2.9 เวลาประมวลผลเชิงขนานบนระบบคอมพิวเตอร์ 8 หน่วยประมวลผล ประกอบด้วยเวลาการคำนวณ เวลาสื่อสาร และเวลาเดินเครื่องสูญเปล่า (Grama *et al.*, 2003)



2.4.1 เวลาประมวลผล

เวลาประมวลผล (Execution time) เป็นข้อมูลพื้นฐานที่ใช้สะท้อนสมรรถนะของระบบคอมพิวเตอร์ ซึ่งหมายถึงเวลาทั้งหมดที่โปรแกรมใช้ในการประมวลผล เวลาประมวลผลของระบบคอมพิวเตอร์หน่วยประมวลผลเดี่ยวหรือ เวลาประมวลผลเชิงลำดับ (Sequential runtime - T_s) นั้น เวลาประมวลผลจะขึ้นอยู่กับขนาดของปัญหา (Problem size - W) เพียงอย่างเดียว ซึ่งแตกต่างกับเวลาประมวลผลของระบบเชิงขนานหรือ เวลาประมวลผลเชิงขนาน (Parallel runtime

- T_p) ที่นอกจากจะขึ้นอยู่กับขนาดของปัญหาแล้ว เวลาประมวลผลยังขึ้นอยู่กับจำนวนหน่วยประมวลผล (Number of processors - p) ด้วย

เวลาประมวลผลเชิงขนานสามารถแบ่งออกได้เป็นสามส่วน (ภาพที่ 2.9) ประกอบด้วย (1) เวลาการคำนวณ (Computation time) ซึ่งเป็นเวลาที่แต่ละหน่วยประมวลผลใช้คำนวณเพื่อแก้ปัญหา (2) เวลาการสื่อสาร (Communication time) เป็นเวลาที่แต่ละหน่วยประมวลผลใช้สื่อสารเพื่อแลก เปลี่ยนข้อมูลหรือประสานจังหวะเวลาร่วมกัน และ (3) เวลาเดินเครื่องเปล่า (Idling time) ซึ่งเป็นเวลาที่ไม่มีการประมวลผลเกิดขึ้น เนื่องจากอาจต้องรอผลลัพธ์จากหน่วยประมวลผลอื่น หรือรอประสานเวลาทำงานกับหน่วยประมวลผลอื่น

เวลาประมวลผลเป็นหน่วยวัดสมรรถนะที่ใช้แสดงเวลาที่สูญเสียไปในการประมวลผล ซึ่งเป็นคุณลักษณะของโปรแกรมทั่วไป แต่บางโปรแกรมที่ต้องการหน่วยวัดสมรรถนะสะท้อนคุณลักษณะที่จำเพาะเจาะจงลงไป เช่น โปรแกรมการคำนวณทางวิทยาศาสตร์ ที่ต้องการทราบถึงอัตราการคำนวณตัวเลขต่อหนึ่งหน่วยเวลา หรือโปรแกรมระบบเครือข่าย ที่ต้องการทราบถึงอัตราการรับส่งข้อมูลต่อหนึ่งหน่วยเวลา โปรแกรมเหล่านี้จะใช้ข้อมูลสะท้อนสมรรถนะที่แตกต่างกัน ซึ่งจะได้กล่าวถึงรายละเอียดในหัวข้อถัดไป

2.4.2 ปริมาณงาน

จากที่กล่าวไว้หัวข้อก่อนหน้านี้ แต่ละโปรแกรมมีคุณลักษณะการทำงานที่แตกต่างกัน ทำให้ต้องการข้อมูลที่ใช้สะท้อนสมรรถนะการทำงานในลักษณะที่แตกต่างกัน แต่การทำงานของทุกโปรแกรมนั้นก็ต้องการปริมาณงานที่ทำได้ต่อหนึ่งหน่วยเวลาให้มากที่สุด *ปริมาณงาน* (Throughput) ซึ่งหมายถึงอัตราการผลิตผลลัพธ์หรือปริมาณงานที่ได้จากการทำงานหรือการประมวลผลของโปรแกรมต่อหนึ่งหน่วยเวลา โดยหน่วยของปริมาณงานของแต่ละโปรแกรมอาจมีความแตกต่างกัน ขึ้นอยู่กับคุณลักษณะหรือวัตถุประสงค์ของการทำงานของโปรแกรม

ตัวอย่างหน่วยของปริมาณงานต่อหนึ่งหน่วยเวลา ได้แก่ มิปส์ (MIPS ย่อมาจาก Million instruction per second) ซึ่งหมายถึงจำนวนคำสั่งในหน่วยหนึ่งล้านคำสั่ง ที่ระบบคอมพิวเตอร์ประมวลผลได้ภายในเวลาหนึ่งวินาที ฟล็อบส์ (FLOPS ย่อมาจาก Floating point operation per second) หมายถึงจำนวนคำสั่งที่เกี่ยวกับเลขจำนวนจริงที่ถูกประมวลผลภายในเวลาหนึ่งวินาที ซึ่งฟล็อบส์แตกต่างจากมิปส์ตรงที่ว่า ฟล็อบส์จะนับเฉพาะคำสั่งที่เกี่ยวกับการประมวลผลเลขจำนวนจริงเท่านั้น ซึ่งมักนำมาใช้เป็นหน่วยวัดสมรรถนะของโปรแกรมการคำนวณ

ทางวิทยาศาสตร์ ในขณะที่มิปส์จะนับทุกคำสั่งที่โปรแกรมใช้ประมวลผล นอกจากนี้ ยังมีหน่วยของปริมาณงานชนิดอื่น เช่น จำนวนการทำรายการต่อนาที (Transaction per minute) ของโปรแกรมระบบฐานข้อมูล

ทั้งเวลาประมวลผลและปริมาณงานนั้น เป็นหน่วยวัดสมรรถนะของการประมวลผลโปรแกรมหนึ่งโปรแกรม แต่ยังมีหน่วยวัดสมรรถนะอีกลักษณะหนึ่งที่ใช้เปรียบเทียบสมรรถนะระหว่างโปรแกรมสองโปรแกรม ซึ่งจะได้กล่าวถึงรายละเอียดในหัวข้อถัดไป

2.4.3 สปีดอัพ

เป้าหมายของการพัฒนาระบบคอมพิวเตอร์เชิงขนานคือ ต้องการสมรรถนะที่เพิ่มขึ้นเมื่อเทียบกับระบบคอมพิวเตอร์หน่วยประมวลผลเดี่ยว ดังนั้น การวิเคราะห์สมรรถนะของระบบขนานมักจะนำข้อมูลสมรรถนะที่ได้ไปเปรียบเทียบกับ ระบบคอมพิวเตอร์หน่วยประมวลผลเดี่ยว เพื่อดูความแตกต่างเชิงสมรรถนะของทั้งสองระบบ *สปีดอัพ* (Speedup - S) เป็นหน่วยวัดสมรรถนะเชิงเปรียบเทียบที่แสดงความสัมพันธ์ของระบบสองระบบ (Relative performance metric) ซึ่งการคำนวณหาสปีดอัพสามารถคำนวณได้โดยใช้เวลาประมวลผล หรือปริมาณงานของสองระบบมาเปรียบเทียบกัน ดังสมการที่ 2.1

$$Speedup = \frac{Serial\ runtime(T_s)}{Parallel\ runtime(T_p)} \text{ หรือ } = \frac{Parallel\ Throughput}{Serial\ Throughput} \qquad \text{สมการที่ 2.1}$$

นอกจากนี้ กฎของแอมดาห์ได้นิยามสปีดอัพของโปรแกรมหรืออัลกอริทึมเชิงขนานโดยพิจารณาว่า หากเพิ่มปัจจัยที่ช่วยเพิ่มสมรรถนะการประมวลผลเข้าไปในระบบเชิงขนาน เช่น หน่วยประมวลผล ปัจจัยดังกล่าวนี้จะไม่มีผลกระทบต่อโปรแกรมทั้งหมด เนื่องจากโปรแกรมสามารถแบ่งออกเป็นสองส่วน คือ ส่วนโปรแกรมแบบลำดับ (Sequential section) ซึ่งเป็นส่วนของโปรแกรมที่ไม่สามารถเพิ่มสมรรถนะได้ และส่วนโปรแกรมแบบขนาน (Parallelizable section) ซึ่งเป็นส่วนของโปรแกรมที่สามารถเพิ่มสมรรถนะได้ ทำให้ปัจจัยมีผลกระทบต่อโปรแกรมแค่บางส่วนเท่านั้น (Patterson & Hennessy, 1990) ผลกระทบหรือสปีดอัพที่ได้ตามกฎของแอมดาห์สามารถคำนวณได้จากสมการที่ 2.2

$$S(n) = \frac{t_s}{ft_s + (1-f)t_s / n} = \frac{n}{1+(n-1)f} \qquad \text{สมการที่ 2.2}$$

เมื่อ n = จำนวนหน่วยประมวลผล จำนวนโพรเซส หรือจำนวนเทรด

$S(n)$ = สปีดอัพของปัจจัยช่วยเพิ่มสมรรถนะ n หน่วย เช่น จำนวนหน่วยประมวลผลหรือเทรด

f = สัดส่วนของโปรแกรมที่ต้องทำงานเป็นลำดับ ไม่สามารถทำงานเชิงขนานได้

t_s = เวลาในการประมวลผลเชิงลำดับ (Serial runtime)

จากที่กล่าวไว้ข้างต้น สปีดอัพเป็นหน่วยวัดสมรรถนะที่ใช้เปรียบเทียบระบบคอมพิวเตอร์ เมื่อระบบคอมพิวเตอร์มีการเพิ่มปัจจัยช่วยเพิ่มสมรรถนะเข้าไปในระบบ เช่น เพิ่มจำนวนหน่วยประมวลผลเป็นสองเท่า สปีดอัพที่ได้ก็ควรจะเพิ่มขึ้นเป็นสองเท่าด้วยเช่นกัน กล่าวคือมีลักษณะการเพิ่มขึ้นเท่ากับจำนวนปัจจัยที่ใส่เข้าไปในระบบ (Linear speedup) แต่จากกฎของแอมดาห์แสดงให้เห็นว่า สปีดอัพที่ได้ขึ้นอยู่กับสัดส่วนของโปรแกรมเชิงขนาน ที่ตอบสนองต่อปัจจัยช่วยเพิ่มสมรรถนะ ซึ่งค่าสปีดอัพนี้ สามารถนำไปคำนวณหาประสิทธิภาพของระบบเชิงขนาน โดยจะกล่าวถึงในหัวข้อถัดไป

2.4.4 ค่าประสิทธิภาพ

ค่าประสิทธิภาพ (Efficiency - E) เป็นข้อมูลที่แสดงถึงประสิทธิภาพการใช้ประโยชน์หน่วยประมวลผลบนระบบเชิงขนาน ซึ่งไม่ได้ใช้เวลาเพื่อการคำนวณเพื่อแก้ปัญหาเพียงอย่างเดียว หน่วยประมวลผลบนระบบเชิงขนานยังต้องสูญเสียเวลาไปกับการสื่อสารและเวลาเดินเครื่องเปล่าดังที่กล่าวถึงในเวลาประมวลผลเชิงขนาน ทำให้แต่ละหน่วยประมวลผลประมวลผลได้ไม่เต็มประสิทธิภาพดังเช่นระบบคอมพิวเตอร์หน่วยประมวลผลเดี่ยว

ค่าประสิทธิภาพนี้ จะขึ้นอยู่กับจำนวนหน่วยประมวลผลในระบบเชิงขนานและสปีดอัพที่ได้ ซึ่งจะมีค่าอยู่ช่วงระหว่าง 0 ถึง 1 และสามารถคำนวณได้จากสมการที่ 2.3

$$Efficiency\ \ (E) = \frac{Speedup\ \ (S)}{Number\ \ of\ \ processors\ \ (p)} \qquad \text{สมการที่ 2.3}$$

ค่าประสิทธิภาพของระบบเชิงขนานนั้น แสดงถึงการใช้ประโยชน์หน่วยประมวลผล ซึ่งหากระบบเชิงขนานใดมีค่าประสิทธิภาพเท่ากับ 1 หมายความว่าระบบเชิงขนานดังกล่าวมีการใช้ประโยชน์หน่วยประมวลผลเต็มประสิทธิภาพมากที่สุด อย่างไรก็ตาม ดังที่กล่าวไว้ในเวลาการประมวลผลเชิงขนาน การประมวลผลแบบขนานนอกจากจะสูญเสียเวลาไปกับการคำนวณแล้ว ยังต้องสูญเสียเวลาไปกับการสื่อสารและเวลาเดินเครื่องเปล่า จึงทำให้ระบบเชิงขนานมีค่าประสิทธิภาพหรือใช้ประโยชน์หน่วยไม่เต็มประสิทธิภาพ เวลาในสองส่วนหลังนี้เป็นเวลาที่ไม่

เกิดขึ้นการประมวลผลแบบลำดับ จึงเรียกว่าเวลาส่วนเกิน ซึ่งจะกล่าวถึงรายละเอียดในหัวข้อถัดไป

2.4.5 เวลาส่วนเกิน

เวลาส่วนเกิน (Overhead time - T_o) หมายถึงเวลาส่วนที่เกิดขึ้นเฉพาะการประมวลผลบนโปรแกรมเชิงขนาน แต่จะไม่เกิดขึ้นในการประมวลผลบนโปรแกรมเชิงลำดับ ได้แก่ เวลาการสื่อสาร และเวลาเดินเครื่องเปล่า เวลาส่วนเกินบนระบบเชิงขนานนี้ สามารถคำนวณได้จากผลต่างของผลรวมของเวลาส่วนเกินทั้งหมดของทุกหน่วยประมวลผล กับเวลาประมวลผลเชิงลำดับ ซึ่งสามารถสรุปได้ดังสมการที่ 2.4

$$Overhead\ time(T_o) = pT_p - T_s \qquad \text{สมการที่ 2.4}$$

เวลาส่วนเกินสามารถนำมาคำนวณหาเวลาประมวลผลเชิงขนานได้จากสมการที่ 2.5 และคำนวณหาสปีดอัพได้จากสมการที่ 2.6

$$T_p = \frac{T_s + T_o}{p} \quad \text{สมการที่ 2.5,} \qquad S = \frac{T_s}{T_p} = \frac{pT_s}{T_s + T_o} \qquad \text{สมการที่ 2.6}$$

จากสมการที่ 3 และ 6 สามารถนำมาสรุปได้เป็นสมการที่ 2.7

$$E = \frac{S}{p} = \frac{T_s}{T_s + T_o} = \frac{1}{1 + (T_o / T_s)} \qquad \text{สมการที่ 2.7}$$

จากสมการที่ 2.4 จะเห็นได้ว่าหากระบบเชิงขนานมีจำนวนหน่วยประมวลผลเพิ่มขึ้นจะส่งผลให้เวลาส่วนเกินเพิ่มขึ้นด้วย และจากสมการที่ 2.7 แสดงให้เห็นว่าหากเวลาส่วนเกินเพิ่มขึ้น จะทำให้ค่าประสิทธิภาพลดลง ดังนั้นสามารถสรุปได้ว่า หากระบบเชิงขนานมีจำนวนหน่วยประมวลผลเพิ่มขึ้น จะส่งผลให้ค่าประสิทธิภาพของระบบเชิงขนานลดลง อย่างไรก็ตามระบบเชิงขนานเป็นระบบคอมพิวเตอร์ที่พัฒนาเพื่อให้สามารถแก้ปัญหาที่มีขนาดใหญ่ หากขนาดของปัญหามีขนาดใหญ่สอดคล้องกับจำนวนหน่วยประมวลผลก็จะทำให้ค่าประสิทธิภาพอยู่ในเกณฑ์ที่ยอมรับได้

หน่วยวัดสมรรถนะที่กล่าวมาทั้งหมด เป็นข้อมูลที่สะท้อนสมรรถะเชิงสรุปโดยรวมของทั้งโปรแกรม นอกจากนี้ หากต้องการวิเคราะห์สมรรถนะไปยังแต่ละส่วนของโปรแกรมหรือระบบคอมพิวเตอร์ ดังนั้นข้อมูลที่นำมาใช้สะท้อนสมรรถนะจะขึ้นอยู่กับคุณลักษณะของเป้าหมายที่ต้องการศึกษาด้วย ซึ่งการใช้ประโยชน์พื้นที่แคชเป็นเป้าหมายของวิทยานิพนธ์นี้ โดยจะได้กล่าวรายละเอียดในหัวข้อถัดไป

2.4.6 การใช้ประโยชน์พื้นที่แคช

จากที่กล่าวไว้ในหัวข้อที่ 2.2.1 แคชเป็นองค์ประกอบหนึ่งที่สำคัญ ที่ส่งผลต่อสมรรถนะการประมวลผลของระบบคอมพิวเตอร์ การนำเสนอข้อมูลการใช้ประโยชน์พื้นที่แคช จะทำให้ผู้วิเคราะห์เข้าใจถึงอิทธิพลของแคชที่มีผลต่อสมรรถนะการประมวลผล เพื่อนำข้อมูลไปปรับปรุงโปรแกรมหรือการออกแบบสถาปัตยกรรมของแคช ให้เหมาะสมกับคุณลักษณะของงาน การนำเสนอข้อมูลการใช้ประโยชน์พื้นที่แคชมีอยู่ด้วยกันหลายลักษณะ ได้แก่ ประสิทธิภาพการทำงานของแคช และคุณสมบัติของพื้นที่เก็บข้อมูล

ข้อมูลประสิทธิภาพการทำงานของแคชนั้น มักจะนำเสนอด้วยอัตราการพบหรือไม่พบข้อมูลในแคช และชนิดของการไม่พบข้อมูล อัตราการพบหรือไม่พบข้อมูลในแคชจะถูกนำเสนอในลักษณะ สัดส่วนร้อยละของการพบหรือไม่พบข้อมูลต่อการเข้าถึงข้อมูลบนหน่วยความจำทั้งหมดของโปรแกรม เป้าหมายการทำงานของแคชที่มีประสิทธิภาพ ย่อมต้องการอัตราการพบข้อมูลที่สูง และอัตราการไม่พบข้อมูลที่ต่ำ จึงจะทำให้โปรแกรมมีสมรรถนะการประมวลผลที่สูงขึ้น ซึ่งสามารถคำนวณได้จากสมการที่ 2.8 และ 2.9 ตามลำดับ

$$Miss\ rate = \frac{Memory\ access\ misses}{Total\ memory\ access} \qquad \text{สมการที่ 2.8}$$

$$Hit\ \ rate = 1 - Miss\ \ rate \qquad \text{สมการที่ 2.9}$$

ข้อมูลอัตราการพบหรือไม่พบข้อมูลในแคชนั้น สามารถนำไปคำนวณหาเวลาเฉลี่ยของการเข้าถึงข้อมูลบนหน่วยความจำ ที่เรียกว่า เอแมต (AMAT ย่อมาจาก Average memory access time) ได้อีกด้วย ค่าเอแมตเป็นเวลาเฉลี่ยของทั้งโปรแกรมที่ใช้ในการเข้าถึงข้อมูลบนหน่วยความจำ โปรแกรมที่มีค่าเอแมตต่ำจะบ่งบอกถึงแคชมีประสิทธิภาพการทำงานที่ดีกว่าโปรแกรมที่มีค่าเอแมตสูง ซึ่งค่าเอแมตสามารถคำนวณได้จากสมการที่ 2.10

$$AMAT = Hit\ time + (Miss\ rate \times Miss\ penalty)$$ สมการที่ 2.10

เมื่อ $Hit\ \ time$ = เวลาที่ใช้ในการเข้าถึงข้อมูลในกรณีที่พบข้อมูลในแคช

$Miss\ \ penalty$ = เวลาที่ใช้ในการเข้าถึงข้อมูลในกรณีที่ไม่พบข้อมูลแคช

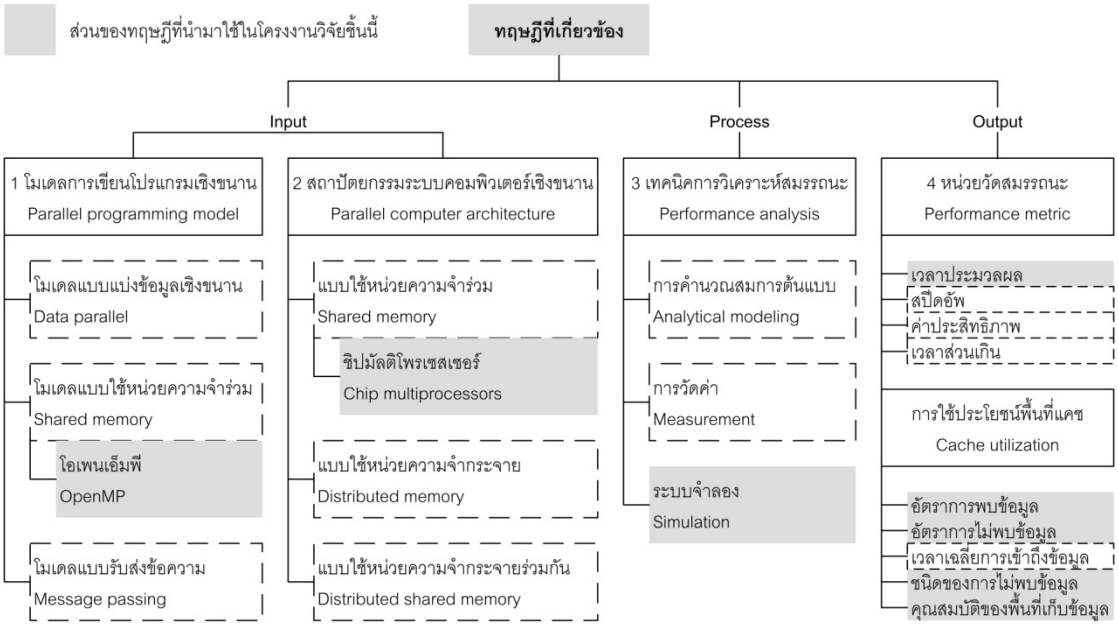
การไม่พบข้อมูลในแคชนั้น เป็นปัญหาที่มีผลต่อสมรรถนะการประมวลผลของโปรแกรม การจำแนกชนิดการไม่พบข้อมูลจะทำให้โปรแกรมเมอร์เข้าใจคุณลักษณะของโปรแกรมที่ส่งผลต่อสมรรถนะการประมวลผลได้ดีขึ้น จากที่กล่าวไว้ในหัวข้อที่ 2.2.1 และ 2.2.2 การไม่พบข้อมูลสามารถจำแนกได้สี่ชนิด คือ การเข้าถึงข้อมูลครั้งแรก การใช้พื้นที่แคชเต็ม การขัดแย้งของข้อมูล การใช้ข้อมูลร่วมกัน (การร่วมใช้จริงและใช้ไม่จริง) การจำแนกชนิดการไม่พบข้อมูลในแคชเหล่านี้ จะทำให้โปรแกรมเมอร์สามารถหลีกเลี่ยงการไม่พบข้อมูลในบางกรณีได้ เช่น การไม่พบข้อมูลเนื่องจากการใช้ข้อมูลร่วมใช้ไม่จริง ซึ่งเป็นสาเหตุที่สามารถพบบนโปรแกรมเชิงขนานแบบใช้หน่วยความจำร่วม และสามารถปรับปรุงโปรแกรมเพื่อหลีกเลี่ยงการไม่พบข้อมูลชนิดนี้ได้ ซึ่งการนำเสนอข้อมูลชนิดของการไม่พบข้อมูลในแคช อาจนำเสนอสัดส่วนเป็นร้อยละของการไม่พบข้อมูลแต่ละชนิด จากการประมวลผลของโปรแกรมทั้งหมด

นอกจากนี้ คุณสมบัติของพื้นที่เก็บข้อมูล เป็นข้อมูลอีกลักษณะหนึ่งที่แสดงถึงการใช้ประโยชน์พื้นที่แคช ดังที่กล่าวถึงไว้ในหัวข้อที่ 2.2.3 คุณสมบัติของพื้นที่เก็บข้อมูลแบ่งออกเป็นสองลักษณะ คือ พื้นที่ที่ถูกใช้บ่อยครั้ง และพื้นที่ซึ่งถูกใช้บริเวณติดกัน ซึ่งสามารถใช้เป็นข้อมูลที่แสดงถึงความต้องการพื้นที่สำหรับจัดเก็บข้อมูลระหว่างการประมวลผลของโปรแกรม หากแคชมีพื้นที่เพียงพอที่รองรับพื้นที่ดังกล่าวได้โปรแกรมจะมีสมรรถนะการประมวลผลที่สูงขึ้น นอกจากนี้คุณสมบัติดังกล่าวนี้ยังแสดงถึงลำดับและความถี่ของการเข้าถึงพื้นที่เก็บข้อมูลอีกด้วย ซึ่งคุณลักษณะดังกล่าวนี้มีประโยชน์ต่อการจัดการของแคช ที่จะดึงข้อมูลจากหน่วยความจำเข้าไว้บนแคชให้สอดคล้องกับการประมวลผลของโปรแกรมอีกด้วย

2.5 สรุปความรู้พื้นฐานและทฤษฎีที่เกี่ยวข้อง

ในบทความรู้พื้นฐานและทฤษฎีที่เกี่ยวข้องนี้ ได้กล่าวถึง องค์ความรู้พื้นฐานที่เกี่ยวข้องกับเครื่องมือวิเคราะห์สมรรถนะพี-สปา ซึ่งแบ่งออกเป็นสี่ส่วน ได้แก่ โปรแกรมที่พัฒนาด้วยโอเพนเอ็มพี ชิปมัลติโพรเซสเซอร์ ระบบจำลอง และหน่วยวัดสมรรถนะและการใช้ประโยชน์พื้นที่แคช จากภาพที่ 2.10 แสดงให้เห็นภาพรวมของทฤษฎีที่เกี่ยวข้องทั้งหมด และแสดงให้เห็นถึงความสัมพันธ์ของทฤษฎีที่นำมาใช้ในโครงงานวิจัยชิ้นนี้ทั้งสี่ส่วน

ภาพที่ 2.10 ภาพรวมของทฤษฎีที่เกี่ยวข้องและส่วนของทฤษฎีที่นำมาใช้ในวิทยานิพนธ์ชิ้นนี้



ในส่วน *โปรแกรมแบบโอเพนเอ็มพี* ได้กล่าวถึงวิธีการแบ่งช่วงตัวนับลูปของคำสั่งกำกับคอมไพเลอร์ ซึ่งการประมวลผลในลูปเป็นพื้นที่ส่วนประมวลผลแบบขนานเป้าหมายสำหรับการวิเคราะห์ของพี-สปา ในหัวข้อ *ชิปมัลติโพรเซสเซอร์* ได้กล่าวถึงลักษณะทางสถาปัตยกรรมของหน่วยประมวลผลชนิดนี้ รวมถึงสถาปัตยกรรมของแคชบนชิปมัลติโพรเซสเซอร์และคุณลักษณะของแคชที่ส่งผลต่อสมรรถนะการประมวลผลของโปรแกรม ซึ่งคุณลักษณะเหล่านี้เป็นส่วนที่จะนำไปใช้เป็นข้อมูลสะท้อนสมรรถนะสำหรับการวิเคราะห์ของพี-สปา ในหัวข้อ *ระบบจำลอง* เนื่องจากโครงงานวิจัยชิ้นนี้จะใช้เทคนิคการวิเคราะห์ด้วยระบบจำลอง ซึ่งเป็นเทคนิคที่สามารถใช้วิเคราะห์บนระบบคอมพิวเตอร์ที่อยู่ในช่วงของการออกแบบได้ และให้ความถูกต้องของข้อมูลที่เป็นที่ยอมรับได้ และในหัวข้อสุดท้ายกล่าวถึงทฤษฎีของ *หน่วยวัดสมรรถนะของระบบเชิงขนาน* และ*การใช้ประโยชน์พื้นที่แคช* ซึ่งจะใช้เป็นข้อมูลสะท้อนสมรรถนะสำหรับการวิเคราะห์ของพี-สปา