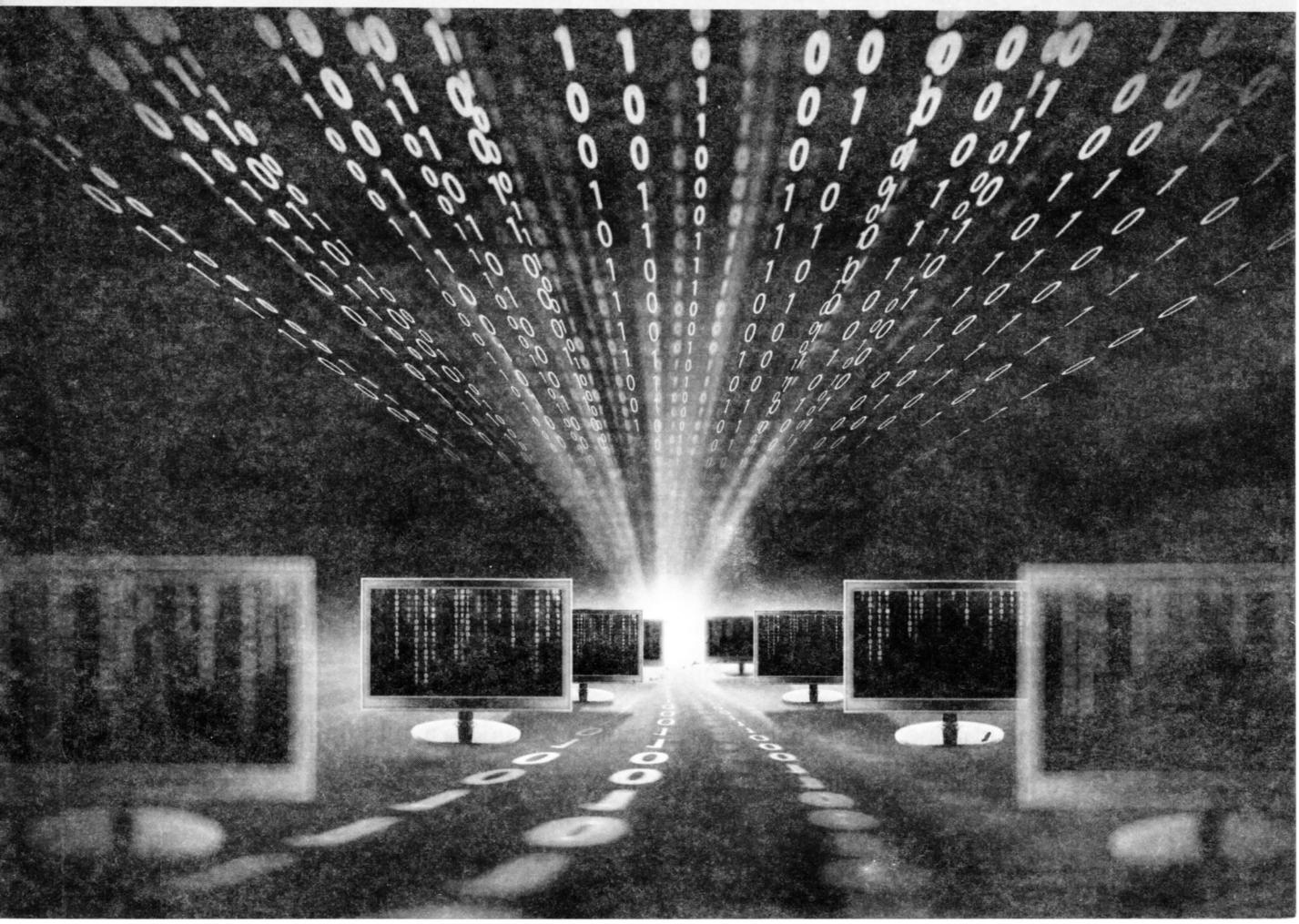


10

การเพิ่มประสิทธิภาพ การใช้หน่วยความจำ



ในบทที่แล้วได้กล่าวถึงองค์ประกอบต่างๆของหน่วยความจำ การออกแบบระบบหน่วยความจำ ปัจจัยการออกแบบหน่วยความจำ อย่างไรก็ได้ประสิทธิภาพของหน่วยความจำนั้นยังขึ้นกับตัวซอฟต์แวร์ระบบอีกด้วย ในบทนี้จะกล่าวถึงวิธีการจัดการหน่วยความจำที่เกี่ยวข้องทั้งฮาร์ดแวร์ และซอฟต์แวร์สนับสนุนเพื่อช่วยให้การทำงานเกี่ยวกับหน่วยความจำนั้นดีขึ้น และเป็นการเพิ่มประสิทธิภาพโดยรวมของระบบ โดยอ้างอิงสมการเวลาการทำงานของชีพียู (CPU Time) และเวลาการเข้าถึงหน่วยความจำ (Memory Access Time)

■ 10.1 ปัจจัยที่เกี่ยวข้องกับประสิทธิภาพของหน่วยความจำ

จากบทที่แล้วได้กล่าวถึงปัจจัยที่เกี่ยวกับประสิทธิภาพของหน่วยความจำ พิจารณาจากสมการเวลาการทำงานของชีพียู ได้แก่ $CPU\ Time = (\text{จำนวนไชเกิลที่ใช้ในการทำงาน (CPU Execution Clock Cycle)} + \text{จำนวนไชเกิลที่ต้อง Stall เนื่องจากการใช้หน่วยความจำ (Memory Stall Cycle)}) \times \text{เวลาที่ใช้ในหนึ่งรอบของสัญญาณนาฬิกา (Clock Cycle Time)}$

โดยที่จำนวนไชเกิลที่ใช้สำหรับการใช้งานของหน่วยความจำหาได้จาก

$\text{Memory Stall Cycle} = \text{จำนวนครั้งของการอ่านหน่วยความจำ} \times \text{อัตราการ Miss ที่เกิดจาก การอ่าน} \times \text{จำนวนไชเกิลสำหรับ Penalty สำหรับการอ่าน} + \text{จำนวนครั้งของการเขียนหน่วยความจำ} \times \text{อัตราการ Miss ที่เกิดจากการเขียน} \times \text{จำนวนไชเกิลสำหรับ Penalty สำหรับการเขียน}$

โดยที่

อัตราการ Miss ที่เกิดจากการอ่าน คือ จำนวนครั้งของการ Miss ในกรณีที่โปรแกรมต้องการการอ่านหรือ Load ข้อมูลและข้อมูลนั้นๆ ไม่ได้อยู่ในแคชต่อจำนวนครั้งของการอ่านหน่วยความจำทั้งหมด

อัตราการ Miss ที่เกิดจากการเขียน คือ จำนวนครั้งของการ Miss ในกรณีที่โปรแกรมต้องการการเขียนหรือ Store ข้อมูลและข้อมูลนั้นๆ ไม่ได้อยู่ในแคชต่อจำนวนครั้งของการเขียนทั้งหมด

จำนวนไชเกิลสำหรับ Penalty สำหรับการอ่าน หมายถึง จำนวนไชเกิลที่ใช้กรณีเกิดการ Miss สำหรับการอ่าน (Read Miss) หน่วยความจำ

จำนวนไชเกิลสำหรับ Penalty สำหรับการเขียน หมายถึง จำนวนไชเกิลที่ใช้กรณีเกิดการ Miss สำหรับการเขียน (Write Miss) หน่วยความจำ

ซึ่งสมการดังกล่าวจะหมายถึง การหาจำนวนไชเกิลที่ใช้โดยเฉลี่ยเมื่อเกิด Stall สำหรับกรณีการเกิด Miss ทั้งประเภทการอ่านและการเขียนหน่วยความจำนั้นเอง

สามารถเขียนเป็นสมการรวมได้ ดังนี้

$CPU\ Time = IC \times (CPI + \#Memory\ Access \text{ ต่อคำสั่ง} \times \text{อัตราการ Miss} \times \text{จำนวน Penalty ไชเกิลสำหรับการ Miss หนึ่งครั้ง} \times \text{Clock Cycle Time})$

โดยที่

IC หมายถึง จำนวนคำสั่งที่ทำงานทั้งหมด

CPI หมายถึง จำนวนไชเกิลเฉลี่ยที่ใช้ต่อหนึ่งคำสั่ง

$\#Memory\ Access \text{ ต่อคำสั่ง หมายถึง จำนวนครั้งการเข้าถึงหน่วยความจำเฉลี่ยต่อหนึ่งคำสั่ง}$

Clock Cycle Time หมายถึง ความยาวของรอบสัญญาณนาฬิกา

ตัวอย่าง สมมติว่าสำหรับเครื่องหนึ่ง มีแคชที่มี Miss Penalty เป็น 50 ไชเกิล และคำสั่งปกติใช้ 2 ไชเกิล (ไม่คิด Stall จากสาเหตุต่างๆ) สมมติให้ Miss Rate ของหน่วยความจำเป็นอัตรา 3% และจำนวนครั้งการอ้างถึงหน่วยความจำต่อคำสั่งเฉลี่ยเท่ากับ 1.33 ผลต่อประสิทธิภาพเมื่อพิจารณาผลของการ Miss จากแคชนี้เป็นอย่างไร

$CPU\ Time = IC \times (CPI + \#Memory\ Access \text{ ต่อคำสั่ง} \times \text{อัตราการ Miss} \times \text{จำนวน Penalty ไชเกิลสำหรับการ Miss หนึ่งครั้ง} \times \text{Clock Cycle Time})$

$$= IC \times (2.0 + (1.33 \times 3\% \times 50)) \times \text{Clock Cycle Time}$$

$$= 4 \times IC \text{ Clock Cycle Time}$$

ดังนั้น เวลาที่ใช้ในการทำงานของชีพียูจะเพิ่มจากปกติที่เป็น 2 ไชเกิลเป็น 4 ไชเกิล

ปกติแล้วถ้า CPI ปกติมีค่าน้อยผลของการ Miss ของแคชเมื่อคิดเป็นจำนวนไชเกิลจะมีมาก (ในเชิงสัมพัทธ์) ดังนั้น การวัด Penalty จากการ Miss ในแคชในหน่วยของไชเกิล สำหรับชีพียูที่มีความถี่หรือ Clock Rate สูง จะทำให้จำนวนไชเกิลที่ใช้ต่อการ Miss หนึ่งครั้งมาก และทำให้ CPI ที่ใช้สำหรับคำสั่งประเภทหน่วยความจำมาก

ตัวอย่าง เปรียบเทียบประสิทธิภาพของการจัดโครงสร้างแคช 2 แบบ รูปแบบแรกคือ Direct Mapped และรูปแบบที่สองคือ 2-Way Set Associative สมมติให้

- CPI ของแคชที่เป็นอุดมคติ (ไม่มีการ Miss เลย) เป็น 2 และกำหนดให้เครื่องมี Clock Cycle Time = 2 นาโนวินาที
- โปรแกรมหนึ่งๆ มีการอ้างถึงหน่วยความจำโดยเฉลี่ย 1.3 ครั้งต่อหนึ่งคำสั่ง
- แคชมีขนาด 64 กิโลไบต์ และทั้งแคชสองแบบ มีขนาดบล็อกเท่ากัน 32 ไบต์
- ให้ Miss Penalty ของแคชเป็น 70 นาโนวินาที และกำหนดให้ Hit Time ใช้เวลา 1 ไซเกล สำหรับทั้งสองแบบ
- สำหรับแบบ 2-Way Set Associative ต้องใช้ MUX ทำให้เวลาของหนึ่งรอบลัญญาณ นาฬิกา (CPU Clock Cycle Time) ต้องนานกว่าเวลาของแบบ Direct Mapped 1.1 เท่า
- กำหนดให้ Miss Rate สำหรับแบบ Direct Mapped เป็น 1.4% และ Miss Rate สำหรับแบบ 2-Way เป็น 1.0%

วิธีทำ

จาก Average Access Time = Hit Time + Miss Rate × Miss Penalty

ดังนั้น

$$\text{Average Access Time ของแบบ 1-Way} = 2 + (0.014 \times 70) = 2.98 \text{ นาโนวินาที}$$

$$\text{Average Access Time ของแบบ 2-Way} = 2 \times 1.1 + (0.010 \times 70) = 2.9 \text{ นาโนวินาที}$$

จากสมการเวลาของซีพียู

$$\text{CPU Time} = IC \times (\text{CPI} + \# \text{Memory Access} \text{ ต่อคำสั่ง} \times \text{อัตราการ Miss} \times \text{จำนวน} \text{ Penalty ไซเกล} \text{ สำหรับการ Miss} \text{ หนึ่งครั้ง} \times \text{Clock Cycle Time})$$

ได้ว่า

$$\text{CPU Time}_{\text{1-Way}} = \text{IC} \times (2.0 \times 2 + (1.3 \times 0.014 \times 70)) = 5.27 \text{ IC}$$

$$\text{CPU Time}_{\text{2-Way}} = \text{IC} (2.0 \times 2 \times 1.1 + (1.3 \times 0.010 \times 70)) = 5.31 \text{ IC}$$

$$\text{CPU Time}_{\text{2-Way}}/\text{CPU Time}_{\text{1-Way}} = 5.31/5.27 = 1.01$$

จากการคำนวณแม้ว่าเวลาการเข้าถึงหน่วยความจำของแบบ 2-Way Set Associative จะตีกว่า

แต่จะพบว่า CPU Time ของรูปแบบ Direct Mapped จะน้อยกว่า

ปกติแล้วเราจะใช้ CPU Time บ่งบอกถึงประสิทธิภาพ ดังนั้น ในแบบ Direct Mapped จึงเป็นทางเลือกที่ดี เพราะว่าใช้เวลาทำงานน้อยกว่าและฮาร์ดแวร์ก็ไม่ซับซ้อนด้วย

พิจารณาอีกด้วยในกรอบแบบแคช

ให้เปรียบเทียบ Miss Rate ระหว่างระบบที่ประกอบด้วยแคชที่เก็บคำสั่งขนาด 16 กิโลไบต์ และแคชที่เก็บข้อมูลขนาด 16 กิโลไบต์กับระบบที่ใช้แคชรวมที่บรรจุทั้งคำสั่งและข้อมูล¹ ขนาด 32 กิโลไบต์ว่าระบบใดมีอัตรา Miss ที่ต่ำกว่ากัน

- สมมติให้การ Hit ใช้เวลา 1 นาทีในการค้นหา
- Miss Penalty ใช้เวลา 50 นาทีเกล
- การ Hit สำหรับคำสั่งแบบ Load/Store ใช้เวลาเกลที่เพิ่มขึ้นอีก 1 นาทีเกล สำหรับระบบแบบแคชรวม เพราะแคชมีพอร์ตอ่านเขียนอย่างละพอร์ต
- สมมติให้ 75% ของการอ้างอิงหน่วยความจำเป็นการอ่านคำสั่งเข้ามาทำงาน (Instruction Reference) และ 25% เป็นการอ้างถึงข้อมูล (Data Reference)
- สมมติให้ใช้แคชแบบ Write Through ที่มี Write Buffer และไม่พิจารณา Stall ที่

¹ จากบทที่แล้วที่เรียกว่า Unified Cache และ Split Cache เป็นการใช้แคชแบบแยกเก็บข้อมูล และแคชที่เก็บคำสั่ง

เกิดจาก Write Buffer

ให้เวลาการเข้าถึงโดยเฉลี่ย (Average Access Time) สำหรับแต่ละกรณี โดยให้ใช้ข้อมูลต่อไปนี้

ขนาดแคช	อัตราการ Miss แคชคำสั่ง	อัตราการ Miss แคชข้อมูล	อัตราการ Miss สำหรับแคชรวม
16 KB	0.64%	6.47%	2.87%
32 KB	0.39 %	4.82%	1.89%

ให้ Miss Rate สำหรับแคชแบบแยกเป็น

$$(75\% \times 0.64\%) + (25\% \times 6.47\%) = 2.1\%$$

อัตราการ Miss สำหรับแคชแบบรวมเป็น 1.99%

เวลาในการเข้าถึงหน่วยความจำหรือ Memory Access Time สำหรับแคชแบบแยกเป็น หาได้จาก

เบอร์เชนต์การอ้างอิงหน่วยความจำที่เกี่ยวกับคำสั่ง \times (เวลาในการ Hit + อัตราการ Miss ของคำสั่ง \times เวลาที่ใช้ในการ Miss (Miss Penalty)) + เบอร์เชนต์การอ้างอิงหน่วยความจำที่เกี่ยวกับข้อมูล \times (เวลาในการ Hit + อัตราการ Miss ของข้อมูล \times เวลาที่ใช้ในการ Miss (Miss Penalty))

$$= 75\% \times (1 + 0.64\% \times 50) + 25\% \times (1 + 6.47\% \times 50) = 2.05$$

Memory Access Time สำหรับแคชแบบเดี่ยว

$$= 75\% \times (1 + 1.89\% \times 50) + 25\% \times (1 + 1 + 1.89\% \times 50)$$

$$= 1.45 + 0.73 = 2.18$$

ดังนั้น ในตัวอย่างนี้แคชแบบแยกให้เวลาในการเข้าถึงที่ดีกว่า

สรุปได้ว่า จากสมการของเวลาในการเข้าถึงหน่วยความจำ การปรับปรุงประสิทธิภาพของแคชทำได้ดังนี้

1. ลดอัตราการ Miss

2. ลด Miss Penalty

3. ลดเวลาที่ใช้เมื่อมีการ Hit เกิดขึ้น

เราสามารถแบ่งลักษณะการ Miss ได้เป็น

1. Compulsory เป็นการ Miss กรณีที่หลักเลี้ยงไม่ได้ ได้แก่ การ Miss ที่มาจากการเริ่มต้นที่โปรแกรมเริ่มทำงาน หรือเรียกว่าเป็นการเริ่มต้น Start up แคช

2. Capacity เกิดจากการที่บล็อกของข้อมูลหลายบล็อกถูกกำหนดมาไว้ที่ เพราะว่า มีความจุที่จำกัดของแต่ละบล็อก และข้อมูลจะต้องถูกนำออกไปจากแคช และถูกนำเข้ามาอีกเมื่อต้องการใช้ข้อมูลส่วนนั้น

3. Conflict เกิดจากเมื่อมีการที่มีบล็อกของข้อมูลหลายบล็อกถูกกำหนดให้มาระหว่าง Cache Line เดียวกัน ทำให้บล็อกข้อมูลที่อยู่ก่อนหน้าใน Cache Line นั้นอาจจะต้องถูกนำออกไปจากแคช และต้องนำเข้ามาอีกเมื่อต้องการใช้ แม้ว่า Cache Line อื่นจะร่วงอยู่

ในการใช้ Associativity ที่มีต่ำมาก และมีการเปลี่ยนมาเป็นต่ำกรีขนาดเล็กลงจะทำให้เกิด Conflict Miss ได้ ถ้าพิจารณาตามต่อไปนี้ของ Associativity ได้ว่า

1. แบบ 8-Way เกิดเป็น Conflict Miss อันเกิดจากการเปลี่ยนจากแบบ Fully Associative มาเป็นแบบ 8-Way

2. แบบ 4-Way เกิดเป็น Conflict Miss อันเกิดจากการเปลี่ยนจากแบบ 8-Way มาเป็นแบบ 4-Way

3. แบบ 2-Way เกิดเป็น Conflict Miss อันเกิดจากการเปลี่ยนจากแบบ 4-Way มาเป็นแบบ 2-Way

4. แบบ 1-Way เกิดเป็น Conflict Miss อันเกิดจากการเปลี่ยนจากแบบ 2-Way มาเป็นแบบ 1-Way

■ 10.2 การลดเวลาในการ Hit

การลดเวลาในการ Hit เป็นการลดเวลาในการอ่านข้อมูลในกรณีที่หาพบในแคช ซึ่งทำได้ยากกว่า เพราะว่า ปกติการเข้าถึงข้อมูลระดับที่ใกล้ๆ ชี้พื้นที่นั้น จะเร็วอยู่แล้วเนื่องจากทางกายภาพของหน่วยความจำ แต่ปัจจัยอื่นก็มีส่วนช่วย เช่น

1. การใช้แคชที่มีขนาดเล็กและรูปแบบง่าย ถ้าแคชขนาดเล็กจะประหยัดพื้นที่ชิปในชาร์ดแวร์ และทำให้ส่วนของ Tag สั้น เป็นการลดความซับซ้อนในวงจรส่วนของการเปรียบเทียบ Tag จะทำให้การ Hit เร็วขึ้น
2. หลีกเลี่ยงการใช้การแปลงแอดเดรสระหว่างการอ้างถึงส่วนของ Index ในแคช ทำให้ประหยัดเวลาในการตีความ (Address Translation) ระหว่างการค้นหา

ตัวอย่างเช่น ในกรณีการใช้แอดเดรสเสมือนสำหรับแคช ปกติแล้วการ Hit จะเกิดขึ้นบ่อยกว่าการ Miss การใช้แคชเสมือนจะช่วยเป็นการลดขั้นตอนการแปลงแอดเดรสกรณีเมื่อมีการ Hit เกิดขึ้นในแคช แต่ต้องระวังการใช้แคชเสมือนในระบบประเภท Multitasking เพราะเมื่อมีการสลับงานกัน แคชเสมือนจะอ้างถึงแอดเดรสจริง (Physical Address) ซึ่งอาจจะใช้ไม่ได้สำหรับงานนั้น อีกต่อไป ทำให้ต้องมีการนำแคชเสมือนนั้นออกไปพร้อมๆ กับเมื่อชี้พื้นที่มีการสลับงานนั้นออกจากหน่วยความจำ หรืออีกทางแก้อาจจะต้องใช้ PID (Process Id) มาประกอบกับการใช้แอดเดรสเสมือนซึ่งทำให้เพิ่มความยาวของ Tag โดยต้องระบุ PID ร่วมกับ Tag

ปัญหาของการใช้แคชเสมือนอีกปัญหานึง คือ เมื่อโปรแกรมมีการทำให้เกิด 2 แอดเดรสเสมือนที่อ้างถึงเขื่อมโยงไปยังแอดเดรสจริงเดียวกัน (Alias) ซึ่งจะทำให้แคชเสมือนเก็บ 2 แอดเดรสเสมือนต่างกันแต่อ้างถึงบล็อกข้อมูลเดียวกัน ทำให้มีสำเนาของบล็อกข้อมูลนั้นหลายแห่งในหน่วยความจำนั่นเอง

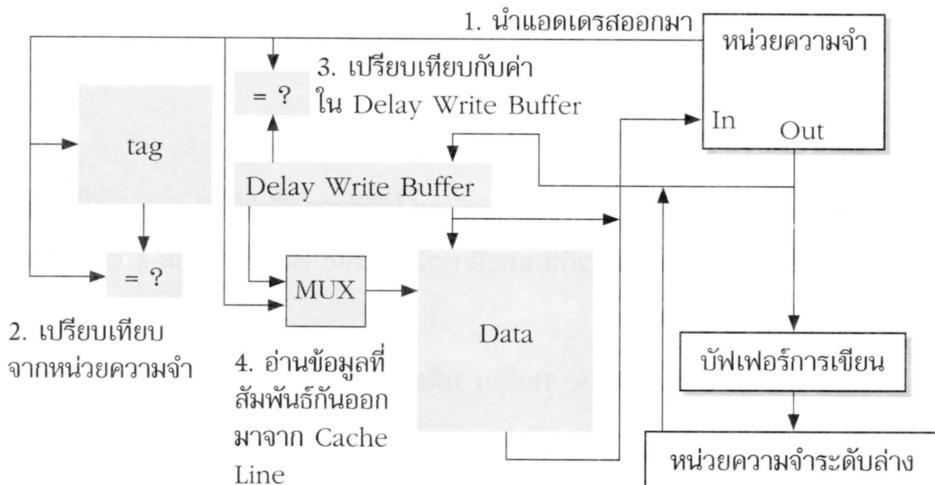
นอกจากนี้ อีกวิธีในการลดเวลาการแปลงแอดเดรส อาจจะได้แก่การแบ่งขั้นตอนในการแปลงเป็นแบบไปปีลอน์ โดยอาศัยการทับซ้อนขั้นตอนการแปลงแอดเดรสเหมือนกับการทำงานแบบไปปีลอน์ของชี้พื้นที่นั่นเอง

3. ทำให้การเขียนทำงานเป็นไปได้โดยอัตโนมัติ

โดยทั่วไปการ Hit สำหรับกรณีการเขียนนั้นใช้เวลานานกว่าการ Hit สำหรับกรณีการอ่าน ดังนั้น อาจจะทำให้การ Hit สำหรับการเขียนเร็วขึ้นโดยการแบ่งขั้นตอนดังนี้

- แบ่งส่วนของ Tag และ ส่วนข้อมูลออกจากกันเพื่อให้สามารถเข้าถึงได้โดยไม่เจ็บแก่กัน
- ทำให้มีการเขียนข้อมูลระหว่างการเปรียบเทียบ Tag

ดังนั้น ข้อมูลที่กำลังถูกเขียนเกิดจากการ Hit ครั้งก่อนหน้า ทำให้เป็นการไปป้อนการเขียน ด้วยนั้นเอง



รูปที่ 10.1 การใช้บัฟเฟอร์การเขียนและการไปป้อนการเขียน

ในรูป 10.1 แสดงโครงสร้างที่มีการเขียนและการเปรียบเทียบ Tag สามารถทำพร้อมการเขียนได้ ทำให้สามารถนำข้อมูลที่ใหม่มาจาก Write Buffer และสามารถส่งข้อมูลจาก Write Buffer ไปยังแคชข้อมูลได้

สำหรับการค้นหาจะทำการเปรียบเทียบ Tag จากทั้งในบัฟเฟอร์ที่รอการเขียน (Delay Write Buffer) และจากหน่วยความจำ Tag พร้อมกัน ถ้าตรงกันก็จะดึง Cache Line ที่สัมพันธ์กันออกมาโดย MUX

■ 10.3 การลด Miss Penalty

ในการลด Miss Penalty นั้นเป็นการลดจำนวนไข่เกลที่ใช้กรณีเกิดการ Miss ขึ้นมีหลายวิธี เช่น

10.3.1 การให้ความสำคัญกับการ Miss สำหรับการอ่าน

เป็นการให้ความสำคัญกับการ Miss สำหรับการอ่านมากกว่าการ Miss สำหรับการเขียน และการมีใช้บัฟเฟอร์การเขียนมาห่วย เมื่อมีการ Miss สำหรับการอ่านโดยไปจจะหาในบัฟเฟอร์ก่อน กรณีนี้จะช่วยลดเวลาสำหรับการ Miss สำหรับการอ่าน การใช้บัฟเฟอร์การเขียนอาจจะทำให้ค่าที่ใหม่เก็บอยู่ในบัฟเฟอร์ เพื่อมีการ Miss ในการอ่านที่แอดเดรสสนั่น

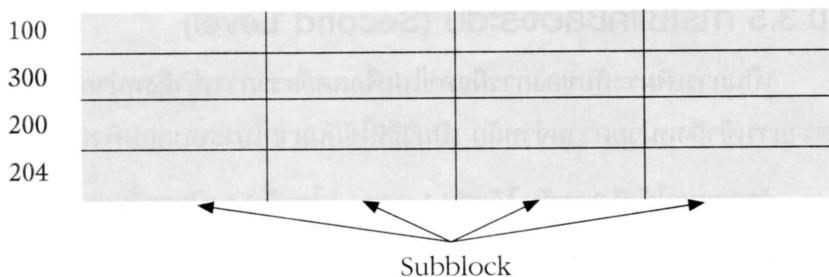
```
SW 512 (R0), R3 ; M[512] <- R3
LW R1, 1024 (R0) ; R1 <- M[1024]
LW R2, 512 (R0) ; R2 <- M[512]
```

สมมติว่าถ้าใช้แคชแบบ Direct-Mapped ใช้วิธีการ Write Through ที่โดยแอดเดรส 512 และ 1024 เป้ามไปยังบล็อกเดียวกัน และมีการใช้ Write Buffer ขนาด 4 เวิร์ดลงหาว่าค่าใน R2 จะเท่ากับค่าใน R3 เสมอหรือไม่

- ค่าใน R3 เก็บใน Write Buffer หลังจากทำการคำสั่ง SW
- คำสั่ง LW ถัดมาใช้ เกิดการ Miss จากการอ่านเพราะตำแหน่งที่เกิดจาก 1024(R0) จะเป้ามไปที่เดียวกับตำแหน่ง 512(R0)
- คำสั่ง LW ถัดไปต้องการอ่านค่ามาจากตำแหน่ง 512(R0) อีกทำให้เกิด Miss ที่เกิดจาก การอ่านอีก ถ้า Write Buffer ยังไม่ได้เปลี่ยนแปลงไปยังแอดเดรส 512 อาจจะทำให้ การ Load นี้ได้ค่าที่เก่า (Out Of Date) ซึ่งทำให้ค่าใน R2 ไม่เท่ากับค่าใน R3 ได้

10.3.2 วิธีการแทนที่บล็อกย่อย (Subblock Replacement)

เป็นการแบ่งบล็อกใหญ่เป็นบล็อกย่อย (Subblock) และทำการอ่านเข้ามาที่ละบล็อกย่อย เมื่อมีการ Miss จากการอ่าน ในรูป 10.2 จะแบ่งข้อมูลเป็นบล็อกย่อยดังนี้



รูปที่ 10.2 การแบ่งช้อมูลเป็นล็อกย่อย

ดังนั้น การค้นหา Tag ในแแคช ถ้าไม่พบข้อมูลในแแคช จึงต้องเข้ามาค้นหาในบล็อกย่อยด้วย

10.3.3 การเริ่มต้นที่เร็วและส่งเวิร์ดสำคัญก่อน (Early Restart และ Critical Word First)

วิธีนี้จะไม่ต้องรอให้ข้อมูลทั้งบล็อกถูกอ่านเข้ามาทั้งหมดก่อน จึงจะให้ปีญทำงานต่อได้ ในวิธีนี้จะทำการเริ่มการทำงานของปีญทันทีที่เริ่ดที่ต้องการมาถึง วิธีนี้เรียกว่า Early Start สำหรับในวิธี Critical Word First นี้จะทำการร้องขอเริร์ดที่ต้องการกับปีญก่อน และส่งเวิร์ดหนึ่งไปยังปีญก่อน และให้ปีญทำงานต่อระหว่างนั้นก็อ่านล่วงอื่นๆ ของบล็อกของแแคชเข้ามาให้เต็มบล็อกพิจารณาตัวอย่าง

ระบบหน่วยความจำที่มีบล็อกของแแคชนานาด 32 ไบต์และใช้ 5 ไบเกิลในการอ่านทีละไบต์เข้ามาโดยใช้บล็อกไปยังหน่วยความจำกว้างขนาด 16 ไบต์ ให้คำนวณจำนวนไบเกิลสำหรับ Miss Penalty กรณีสำหรับวิธี Critical Word First ถ้าไม่มีการใช้เริร์ดอื่นๆ ในบล็อกของแแคชระหว่างการส่งบล็อกนั้นเลย

Miss Penalty โดยเฉลี่ยจะเป็น 5 ไบเกิล สำหรับกรณี Critical Word First มีลักษณะต้องรอจนครบ 32 ไบต์ ก่อนที่จะใช้การอ่านถึง 2 ครั้ง (เท่ากับใช้ 10 ไบเกิล) สำหรับบล็อกขนาด 32 ไบต์

10.3.4 แแคชแบบ Non-Blocking

การใช้แแคชแบบ Non-Blocking ทำให้สามารถอ่านแแคชข้อมูลได้ระหว่างการรับส่งข้อมูลกรณีสำหรับการ Hit ในแแคชหรือเรียกว่า Hit Under Miss นอกจากนี้ ยังเป็นการทำงานทับซ้อนกับการ Miss หลายๆ ครั้ง โดยทำลักษณะของไปป์ไลน์ เพื่อลด Miss Penalty โดยรวม

10.3.5 การใช้แคชสองระดับ (Second Level)

เป็นการเพิ่มระดับของการมีแคชไปเพื่อลดอัตราการเข้าถึงหน่วยความจำระดับล่างสุด หรือ อัตราการเข้าถึงหน่วยความจำหลัก เป็นวิธีที่ใช้กันมากในระบบคอมพิวเตอร์ปัจจุบัน

ถ้ากำหนดให้มี 2 ระดับ ได้แก่ L1 และ L2 โดยที่ L2 เป็นระดับล่างก่อนถึงหน่วยความจำหลัก จะได้จากสูตรเวลาการเข้าถึงหน่วยความจำ

$$\text{Average Memory Access Time} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$$

และหากค่าเวลาที่ต้องใช้สำหรับ Penalty ได้จาก

$$\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$$

เราสามารถแบ่งการ Miss ในระดับ L2 เป็น 2 แบบ

- Local Miss Rate เป็นจำนวนครั้งของการ Miss ในแคชหน้าหารด้วยจำนวนครั้งในการเข้าถึงแคชนั้นๆ (เรียกว่า Miss Rate_{L2})
- Global Miss Rate เป็นจำนวน Miss ในแคชหน้าหารด้วยจำนวนครั้งในการเข้าถึงทั้งหมด กำหนดในทอม Miss Rate_{L1} × Miss Rate_{L2}

ปัจจัยที่ต้องพิจารณาในการออกแบบแคชระดับ L2 ได้แก่ ความเร็วในการเข้าถึงของแคช L1 เพราะมีผลโดยตรงกับ Hit Time หรือเวลาในการค้นหาพบร และจะมีผลกับสัญญาณนาฬิกาของชีพิญ ขณะที่ความเร็วของการเข้าถึงของแคช L2 จะมีผลต่อเวลาของ Miss Penalty

ลองพิจารณาว่าการใช้ L2 จะช่วยลดเวลาในการเข้าถึงในส่วนที่มีผลต่อ CPI หรือไม่ และต้องเพิ่มค่าใช้จ่ายเท่าไรสำหรับ L2

ขนาดของระดับ L2 ต้องใหญ่กว่าขนาดของระดับ L1 กล่าวคือ ถ้าข้อมูลอยู่ในระดับ L1 ก็ มีแนวโน้มที่จะอยู่ในแคช L2 ด้วย ดังนั้น ขนาดของระดับ L2 ควรใหญ่กว่าขนาดของ L1 ในระดับหนึ่งซึ่งทำให้อัตราการ Miss ลดลงได้

ตัวอย่าง พิจารณาผลของการใช้ระดับ L2 ถ้ามีลักษณะของระดับ L2 เป็นดังนี้

- Hit Time_{L2} สำหรับแบบ Direct Mapped = 10 นาที
- Hit Time_{L2} สำหรับแบบ Two-Way มากกว่าแบบ Direct Mapped อยู่เพียง 0.2 นาที

- Local Miss Rate_{L2} สำหรับแบบ Direct Mapped = 25%
- Local Miss Rate_{L2} สำหรับแบบ Two-Way = 20%
- Miss Penalty_{L2} = 50 ไซเกิล

สำหรับแบบ Direct Mapped จะได้ Miss Penalty_{1-Way L2} = $10 + 25\% \times 50 = 22.5$ ไซเกิล กรณี L2 ใช้แบบ 1-Way

และ Miss Penalty_{2-Way L2} = $10.2 + 20\% \times 50 = 20.2$ ไซเกิล สำหรับกรณีระดับ L2 ใช้แบบ 2-Way

ซึ่งจะเห็นว่าแบบ Miss Penalty_{2-Way L2} น้อยกว่า

ในการใช้แคช L2 นั้น ถ้าข้อมูลพบในแคช L1 ก็จะพบในระดับ L2 ด้วย คุณสมบัตินี้ เรียกว่า Multilevel Indusive ปกติแล้วคุณสมบัตินี้จะทำให้เกิดความสอดคล้องกัน (Consistency) ระหว่างข้อมูลในหน่วยความจำระดับต่างๆ ตั้งแต่ระดับของอินพุตเอาท์พุตไปจนถึงระดับแคช ทำให้ สามารถหาข้อมูลที่ใหม่สุดได้จากระดับล่างสุด ได้แก่ แคช L2 เลย ซึ่งปกติในระดับล่าง เช่น L2 นี้ จะใช้เวลาในการเข้าถึงมากกว่าจึงมักจะมีบล็อกที่ขนาดเล็กกว่าระดับบนอยู่แล้ว

■ 10.4 การลด Miss Rate

วิธีการลด Miss Rate มีหลายวิธีด้วยกันเท่านั้น

10.4.1 เพิ่มขนาด Cache Line

วิธีนี้จะทำให้ลด Compulsory Miss เพราะว่าการเพิ่มขนาดของ Cache Line เป็นการเริ่ม Locality เขิงพื้นที่แต่จะเป็นการเพิ่มไซเกิลสำหรับ Miss Penalty เพราะการลดจำนวนบล็อกที่เก็บได้ในแคช จะทำให้เพิ่ม Conflict Miss และเป็นการเพิ่ม Miss Rate ตั้งนั้น วิธีนี้อาจจะเป็นการไม่ดีถ้าเพิ่มขนาด Cache Line ให้ใหญ่จนทำให้ Miss Penalty มากเกินไป กล่าวคือเวลาในการอ่อนข้อมูลจากหน่วยความจำระดับล่างจะมากขึ้นถ้า Cache Line มีขนาดใหญ่

ตัวอย่าง จากตารางแสดงอัตราการ Miss สมมติให้ระบบหน่วยความจำใช้เวลา 40 นาทีเกลเป็น Overhead ในการโอนข้อมูลขนาด 16 ไบต์ในทุกๆ 2 นาทีเกล เช่น การรับส่งข้อมูล 16 ไบต์ จะใช้ 42 นาทีเกลและการรับส่งข้อมูลขนาด 32 ไบต์ จะใช้ 44 นาทีเกล เป็นต้น

ขนาดบล็อก (ไบต์)	อัตราการ Miss สำหรับแคชแต่ละขนาด (%)				
	1 กิกะไบต์	2 กิกะไบต์	4 กิกะไบต์	16 กิกะไบต์	256 กิกะไบต์
16	15.05	8.57	3.94	2.04	1.07
32	13.34	7.24	2.87	1.35	0.70
64	13.76	7.00	2.64	1.07	0.51
128	16.64	7.78	2.77	1.01	0.49
256	22.01	9.51	3.29	1.15	0.48

รูปที่ 10.3 กำหนดขนาดและอัตราการ Miss ของแต่ละรูปแบบของแคช

จากตารางรูปที่ 10.3 ข้างต้นขนาดบล็อกควรเป็นเท่าไร จึงจะทำให้เวลาเข้าถึงเฉลี่ย (Average Access Time) น้อยที่สุด

พิจารณาสำหรับแคชแต่ละขนาด คำนวณได้ดังนี้

$$\text{Average Access Time} = \text{Hit Time} + \text{Miss Rate} \times \text{Miss Penalty}$$

สำหรับบล็อกขนาด 16 ไบต์ และแคชขนาด 1 กิกะไบต์ จะได้ Miss Penalty = 40 + 2

เพราะว่า Hit Time ในที่นี้ใช้ 1 นาทีเกล และมีอัตราการ Miss = 15.05% และ

เพราะว่าขนาดของบล็อก = 16 ไบต์ ซึ่งใช้ 2 นาทีเกลในการโอนข้อมูลเข้ามา และจากโจทย์ มี Overhead = 40 นาทีเกล

ดังนั้น จะมี Average Access Time เป็น

$$1 + (15.05\% \times 42) = 7.321 \text{ นาทีเกล}$$

สำหรับลักษณะ 32,64,128,256 ไบต์ จะจำนวนได้ Miss Penalty เป็น 44,48,56 และ 72 ตามลำดับ

ดังนั้น สามารถคำนวณ Average Access Time ต่อไปสำหรับแต่ละกรณีของบล็อกและกรณีของขนาดแคชได้ต่อไป เพื่อหาว่าแคชขนาดเท่าไรและบล็อกขนาดเท่าไรจึงจะให้ Average Access Time ต่ำสุด

การเลือกขนาดของบล็อกนั้นขึ้นกับเวลาการทำงาน (Latency) และแบบตัวอย่างหน่วยความจำระดับล่าง

- ถ้าหน่วยความจำมี Latency น้อย และแบบตัวอย่างหน่วยความจำระดับล่างสูง ขนาดของบล็อกใหญ่ก็จะดี เพราะแคชจะบรรจุข้อมูลได้มากเมื่อเทียบกับ Penalty ที่ใช้เวลาอยู่ (เพราะใช้เวลาโอนข้อมูลเร็วเนื่องจากแบบตัวอย่างสูง)
- ในทางตรงกันข้าม อาจจะไม่ดี เพราะขนาดของบล็อกที่ใหญ่จะทำให้มีค่า Miss Penalty มาก

10.4.2 การเพิ่มตีกรีดของ Associativity

พิจารณาการเพิ่มตีกรีดของ Associativity ในตัวอย่างนี้

สมมติว่าการเพิ่ม Associativity จะทำให้เพิ่มเวลาของรอบสัญญาณนาฬิกาในหนึ่งรอบเนื่องจากฮาร์ดแวร์ซึ่งข้อนามากขึ้น ดังนี้

Clock Cycle Time ของแบบ 2-Way = 1.1 Clock Cycle Time ของแบบ 1-Way

Clock Cycle Time ของแบบ 4-Way = 1.12 Clock Cycle Time ของแบบ 1-Way

Clock Cycle Time ของแบบ 8-Way = 1.14 Clock Cycle Time ของแบบ 1-Way

ให้ Hit Time = 1 ไซเกิล และ Miss Penalty ของแคชแบบ Direct Mapped เป็น 50 ไซเกิล ถ้าใช้ตัวเลขจากตารางรูปที่ 10.3

ขนาดของแคชเท่ากับเท่าใดทำให้แต่ละประโยชน์ต่อไปนี้เป็นจริง

1. Avg Access Time ของแบบ 8-Way < Avg Access Time ของแบบ 4-Way
2. Avg Access Time ของแบบ 4-Way < Avg Access Time ของแบบ 2-Way

3. Avg Access Time ของ แบบ 2-Way < Avg Access Time ของ แบบ 1-Way

เริ่มต้นจะต้องหา

หา Average Access Time สำหรับแต่ละรูปแบบของ Associativity

Average Access Time ของแบบ 8-Way = Hit Time ของแบบ 8-Way + อัตราการ Miss ของแบบ 8-Way × Miss Penalty ของแบบ 8-Way = $1.14 + \text{อัตราการ Miss ของแบบ 8-Way} \times 50$

Average Access Time ของแบบ 4-Way = $1.12 + \text{อัตราการ Miss ของแบบ 4-Way} \times 50$

Average Access Time ของแบบ 2-Way = $1.10 + \text{อัตราการ Miss ของแบบ 2-Way} \times 50$

Average Access Time ของ แบบ 1-Way = $1 + \text{อัตราการ Miss ของแบบ 1-Way} \times 50$

ดังนั้น จากการคำนวณพบว่าเมื่อขนาดของแคช ตั้งแต่ 32,64 และ 128 ไบต์

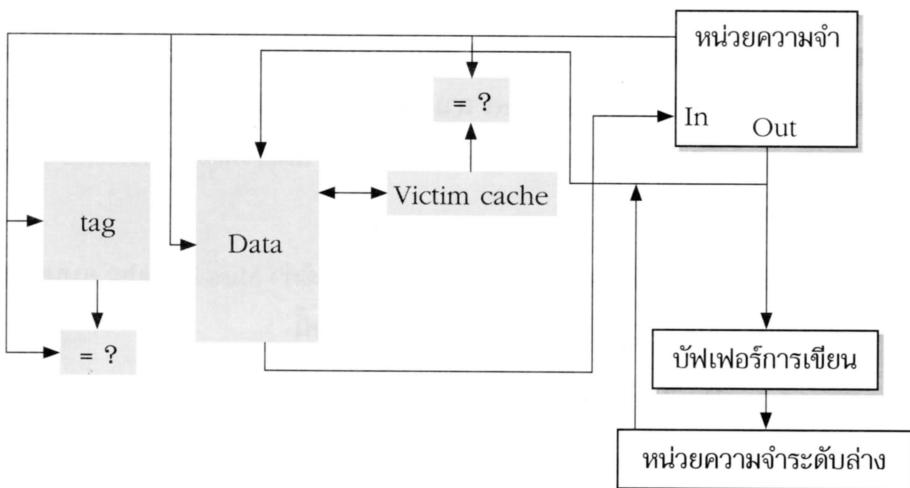
Average Access Time ของแบบ 8-Way > Access Time ของแบบ 4-Way

การเพิ่มตีกรีบของ Asscoaitivity จึงเป็นการเพิ่มเวลาการเข้าถึงหน่วยความจำโดยเฉลี่ย

10.4.3 การใช้แคชแบบ Victim

เป็นการลดจำนวน Miss แบบ Conflict Miss โดยเพิ่มแคชที่เป็นแบบ Fully Associative เข้าไปในระบบ แคชนี้เรียกว่าแคช Victim ใช้เก็บล็อกที่ถูกนำออกมาจากแคชที่ใกล้กับชิปซึ่งเพราะ ว่าจะเกิดการแทนที่ขึ้น ดังนั้น แคช Victim จะถูกนำมาตรวจสอบก่อนที่จะไปค้นหาข้อมูลในหน่วยความจำระดับล่าง ถ้าบล็อกที่ต้องการพบในแคช Victim ก็จะนำเอาบล็อกนั้นมาสับกับ Cache Line หนึ่งที่เลือกขึ้นมาด้วยวิธีการแทนที่ของระบบหน่วยความจำ (Replacement Policy) เอง

ตัวอย่างโครงสร้างเพิ่มแคช Victim ดังรูป 10.4



รูปที่ 10.4 การใช้แคช Victim

จะเห็นว่าแคช Victim จะถูกใช้เข้าไปเสริมร่วมกับแคชข้อมูล เพื่อเก็บบล็อกข้อมูลที่ถูกนำออกมาจาก Cache Line เมื่อมีการแทนที่

10.4.4 การใช้วิธี Pseudo-Associative

เป็นการทำให้ความเร็วในการ Hit เท่าๆ กับการใช้วิธีการ Direct-Mapped แต่ลดต้องการลดอัตราการ Miss ให้เหลือน้อยเหมือนกับแคชแบบ Set-Associative

สำหรับการ Hit จะมีการทำงานเหมือนกับการ Hit โดยใช้วิธี Direct-Mapped แต่เมื่อมีการ Miss ก่อนจะไปเอาข้อมูลใน หน่วยความจำระดับล่างจะต้องตรวจสอบใน Cache Line ว่า มีบล็อกที่ต้องการอยู่หรือไม่ โดยจะไปหาใน Cache Line ที่มี Pseudo Set โดย Pseudo Set จะได้จากการจะกลับค่าใน MSB บิตในพิลต์ Set (หรือ Index)

เช่น ถ้า Set Index เป็น 0110_2 ค่า Pseudo Set Index จะเป็น 1110_2 การหาใน Set นั้นจะทำพร้อมกับการหาใน Pseudo Set

ลองพิจารณาเวลาการทำงานของการค้นหาในกรณีนี้ จะมี Hit Time เท่าเดิม และจะเพิ่มเวลา Pseudo Hit Time เข้าไปในกรณีที่หาพบใน Pseudo Set และยังมี Miss Penalty เท่าเดิม

ตัวอย่าง สมมติว่าในการค้นหาล็อกใช้ 2 ไบเกิลในการหาบล็อกที่ต้องการใน Pseudo Set สำหรับกรณีไม่พบในแบบ Direct Mapped ตามวิธีใดระหว่าง Direct-Mapped, Two-Way Set Associative และ Pseudo Set-Associative จะค้นหาได้เร็วที่สุดสำหรับแคชขนาด 2 กิโลไบต์ และ 128 กิโลไบต์

ในกรณีของ Pseudo Set นั้น Miss Penalty จะใช้ค่า Miss Penalty แบบของแบบ 1-Way เพราะว่าเป็นการ Miss จากแบบ Direct-Mapped ดังนี้

$$\text{Hit Time}_{\text{pseudo}} = \text{Hit Time}_{\text{1-way}} + \text{Alternate Hit Rate}_{\text{pseudo}} \times 2$$

ส่วน Hit Time ของแบบ 1-Way เป็นลักษณะเดียวกับการค้นหาพบในรูปแบบ Direct-Mapped โดยที่ Alternate Hit Rate_{pseudo} เป็นอัตราการ Hit สำหรับกรณีที่พบใน Pseudo Set และใช้ 2 ไบเกิลสำหรับการ Hit ใน Pseudo Set แบบนี้ ดังนั้น

$$\text{Alternate Hit Rate}_{\text{pseudo}} = \text{Hit Rate}_{\text{2-Way}} - \text{Hit Rate}_{\text{1-Way}}$$

คือ ผลต่างระหว่างอัตราการ Hit ของทั้งสองแบบ

$$\begin{aligned} &= (1 - \text{Miss Rate}_{\text{2-Way}}) - (1 - \text{Miss Rate}_{\text{1-Way}}) \\ &= \text{Miss Rate}_{\text{1-Way}} - \text{Miss Rate}_{\text{2-Way}} \end{aligned}$$

สรุปสูตรได้เป็น

$$\begin{aligned} \text{Average Memory Access Time}_{\text{pseudo}} &= \text{Hit Time}_{\text{1-way}} + (\text{oัตราการ Miss}_{\text{1-way}} - \\ &\quad \text{oัตราการ Miss}_{\text{2-way}}) \times 2 + \text{oัตราการ} \\ &\quad \text{Miss}_{\text{2-way}} \times \text{Miss Penalty}_{\text{1-way}} \end{aligned}$$

โดยที่ Miss Penalty_{1-way} ได้จาก อัตราการ Miss_{pseudo} × Miss Penalty_{pseudo} และ อัตราการ Miss_{2-way} ที่คือ Hit Time_{pseudo}

ดังนี้นั่งสามารถคำนวณหา Average Memory Access Time ของแบบ Pseudo Set สำหรับแคชทั้งสองขนาด ได้แก่ 2 กิโลไบต์ และ 128 กิโลไบต์ เปรียบเทียบกับ Average Access Time สำหรับ 1-Way และ 2-Way ของทั้งสองขนาดได้

10.4.5 การใช้ฮาร์ดแวร์ทำการ Prefetch ทั้งแคชแบบคำสั่งและแคชข้อมูล

การใช้แคช Victim และแบบ Pseudo Set Associative เป็นการวิธีการทำให้อัตราการ Miss ลดลงโดยเป็นผลทำให้เวลาของรอบสัญญาณนาฬิกายาวขึ้น วิธีการ Prefetch เป็นการนำเอาข้อมูลเข้ามาไว้ก่อนใน Cache Line ก่อนจะมีการร้องขอจากชีพียู จึงเป็นการคาดคะเนว่าจะใช้บล็อกใดก่อนด้วยเทคนิคเฉพาะของตัวเปลภาราและต้องมีบัฟเฟอร์ Prefetch เพื่อไว้เก็บข้อมูลที่ Prefetch ล่วงหน้าเข้ามา รวมทั้งมีหน่วยควบคุมพิเศษให้สามารถทำ Prefetch ได้พร้อมๆ กันที่ชีพียูทำงานโดยการ Prefetch สามารถ Prefetch ได้ทั้งคำสั่งและข้อมูล

พิจารณาตัวอย่างการคำนวณผลของการใช้ Prefetch

อัตราการ Miss รวมจะเป็นเท่าไรถ้าใช้การ Prefetch คำสั่งเข้ามาช่วย สมมติให้ใช้ 1 ไบเกิลในกรณีมีการ Miss สำหรับแคชคำสั่งแต่สมมติให้มีบัฟเฟอร์ Prefetch และต้นทางพบในบัฟเฟอร์ Prefetch

สมการเวลาการเข้าถึงเฉลี่ย ได้ว่า

$$\text{Average Access Time}_{\text{Prefetch}} = \text{Hit Time} + \text{อัตราการ Miss} \times \text{อัตราการ Hit}_{\text{Prefetch}} \\ \times 1 + \text{อัตราการ Miss} \times (1 - \text{อัตราการ Hit}_{\text{Prefetch}}) \\ \times \text{Miss Penalty}$$

สมมติให้อัตราการ Hit_{Prefetch} = 25% และอัตราการ Miss สำหรับแคชคำสั่งขนาด 8 กิโลไบต์เป็น 1.10% และ Hit Time = 1 ไบเกิล และ Miss Penalty = 50 ไบเกิล

$$\text{Average Access Time}_{\text{Prefetch}} = \text{Hit Time} + \text{อัตราการ Miss} \times \text{อัตราการ Hit}_{\text{Prefetch}} \\ \times 1 + \text{อัตราการ Miss} \times (1 - \text{อัตราการ Hit}_{\text{Prefetch}}) \\ \times \text{Miss Penalty} \\ = 1 + (1.10\% \times 25\% \times 1) + (1.10\% \times (1-25\%)) \\ \times 50 \\ = 1.415$$

หาอัตราการ Miss รวมสำหรับการใช้ Prefetch จากสูตร

$$\text{Average Access Time} = \text{Hit Time} + \text{อัตราการ Miss} \times \text{Miss Penalty}$$

$$\begin{aligned}\text{อัตราการ Miss} &= (\text{Average Access Time} - \text{Hit Time}) / \text{Miss Penalty} \\ &= (1.415 - 1) / 50 = 0.83\%\end{aligned}$$

10.4.6 การใช้ Prefetch โดยอาศัยตัวแปลภาษา

การใช้วิธีการ Prefetch นั้นต้องอาศัยตัวแปลภาษาจะต้องทำการวิเคราะห์โค้ดโปรแกรม และแทรกคำสั่งเพื่อทำการ Prefetch ข้อมูลก่อนจะมีการใช้ข้อมูลนั้นๆ วิธีการมีหลายระดับ ได้แก่

- Prefetch ใส่รีจิสเตอร์ โดยทำการอ่านค่าเข้ามาใส่ในรีจิสเตอร์ก่อนมีคำสั่ง Load รีจิสเตอร์นั้นจริง
- Prefetch ใส่แคช ทำการอ่านข้อมูลเข้ามาในแคชโดยไม่ใช้รีจิสเตอร์ก่อนมีการร้องขอ ข้อมูลนั้นจากบีพียู

ในทั้งสองกรณีต้องมีการระวังปัญหาอื่นๆ ที่เกี่ยวข้องด้วย เช่นการใช้แอดเดรสที่ผิดไป เนื่องจากเป็นแอดเดรสสมมติ เพราะโปรเซสเซอร์ ได้ถูกสับออกไปแล้วรวมทั้งปัญหาการล่วงล้ำ สิทธิการใช้พื้นที่ในหน่วยความจำกรณีต่างๆ รวมทั้งกรณีการอ้างอิงแอดเดรสผิดไปด้วย

การ Prefetch จะสมมติว่าแคชสามารถให้ข้อมูลที่ร้องขอได้ระหว่างการ Prefetch ข้อมูลอยู่ หรือเป็นลักษณะของ Nonblocking (Lockup-Free) นั่นเอง

จุดมุ่งหมายของการทำงานของ Prefetch ของตัวแปลภาษา ได้แก่

- ทำให้เกิดการทำซ้ำของการทำงานของคำสั่งกับการ Prefetch ข้อมูลเข้ามาทำงาน ตัวอย่างการพิจารณาการ Prefetch ที่เห็นได้ชัด ได้แก่ ลักษณะโปรแกรมที่มีโครงสร้างการทำงานแบบลูป
- ถ้าระบบนั้นมี Miss Penalty มาก ก็ควรจะขยายสูญ (Unroll) หลายครั้งหรือใช้เทคนิคไปปีลินหางซอฟต์แวร์ (Software Pipelining) ช่วยเพื่อให้ Prefetch ข้อมูลของการทำงานในรอบอื่นๆ ข้างหน้ามาอวดไว้ในบัฟเฟอร์ของ Prefetch เพราะการอ่านข้อมูลเข้ามาในบัฟเฟอร์ใช้เวลานานจึงต้องอ่านล่วงหน้านาน

- ถ้าระบบนั้นมี Miss Penalty น้อย ก็ควรจะขยายลูปน้อยๆ ครั้งและทำการ Prefetch ไปพร้อมๆ กับการรันแต่ละคำสั่งในลูปได้

ตัวอย่าง พิจารณาโค้ดต่อไปนี้ ว่า

- การเข้าถึงข้อมูลในล่วงของโปรแกรม จะได้บ้างในโปรแกรมที่ทำให้เกิดการ Miss
- การแทรก คำสั่ง Prefetch ไปเพื่อลดการ Miss ทำได้อย่างไร
- ให้คำนวณจำนวนคำสั่ง Prefetch ที่แทรกไปและนับดูว่าจำนวน Miss ที่ลดไปเป็นอย่างไร

สมมติว่าแ吖มมีบล็อกขนาด 16 ไบต์ สมมติให้ใช้วิธีการ Write-Back สำหรับการเขียนและใช้การ Write Allocate และขนาด a และ b แต่ละตัวในอะเรย์มีขนาดเป็น 8 ไบต์เก็บศนนิยมแบบ Double และอะเรย์ a เก็บเลขศนนิยม มีขนาด 3 แถว \times 100 คอลัมน์ ส่วนอะเรย์ b เป็นอะเรย์เก็บเลขศนนิยมเข่นกันมีขนาด 101 แถว \times 3 คอลัมน์

```
for (i=0; i < 3; i++)
    for (j=0; j < 100; j++)
        a[i][j] = b[j][0] * b[j+1][0]
```

เนื่องจากขนาดของ Cache Line ขนาด 16 ไบต์ แต่ละค่าในอะเรย์มีขนาด = 8 ไบต์ ดังนั้น Cache Line 1 สถา เก็บได้อะเรย์ได้ 2 ตัว ตัวละ 8 ไบต์

ตั้งแต่ลูปแรก (เมื่อ $i=0, j=1, 2, 3\dots$) จะใช้ค่า i, j ตามลำดับดังนี้

```
i=0, j=0  b[0][0] b[1][0] a[0][0]
i=0, j=1  b[1][0] b[2][0] a[0][1]
i=0, j=2  b[2][0] b[3][0] a[0][2]
:
```

สำหรับ a จะได้ประโยชน์จาก Locality เชิงพื้นที่ เพราะอะเรย์ a จัดเก็บแบบแนวๆ ตามลักษณะของภาษา C

$i=0, j=0 \quad a[0][0]$ $i=0, j=1 \quad a[0][1]$ $i=0, j=2 \quad a[0][2]$	} จัดเก็บ a แบบแนวๆ (Rowwise) } สองตัวติดกัน } มีการ Miss เกิดขึ้น
--	--

ในการอ่านอะเรย์ a เข้ามาใน Cache Line โดยอ่านครั้งละ 2 ตัว (ขนาดแต่ละตัว = 8 ไบต์)

a [0] [0]	a [0] [1]
-----------	-----------

ดังนั้นการเขียนไปยัง $a[i][j]$ จะเกิดการ Miss จากการเขียน ทุกรอบเลขคู่ $j = 0, 2, 4, \dots$

เพราะว่าอะเรย์ a มีขนาด 3×100 จำนวน Miss = $(3 \times 100) / 2 = 150$ ครั้ง

สำหรับอะเรย์ b ไม่ได้ประโยชน์จาก Locality เชิงพื้นที่แต่ได้จาก Locality เชิงเวลา เพราะอะเรย์บางตัวของ b ถูกอ้างถึง 2 รอบติดกันของ j พิจารณาคู่ของ b ที่ปีดเล้นให้ในโคลัมข้างล่าง

i=0, j=0 b [0] [0] b [1] [0]

i=0, j=1 b [1] [0] b [2] [0]

i=0, j=2 b [2] [0] b [3] [0]

:

และสำหรับรอบ i ที่ต่างๆ กัน เพราะว่าอะเรย์ b[j][0] และ b[j+1][0] ในแต่ละรอบของ i จะมีการใช้ b[i][0]

i=0, j=0 b [0] [0] b [1] [0]

i=0, j=1 b [1] [0] b [2] [0]

i=0, j=2 b [2] [0] b [3] [0]

:

i=1, j=0 b [0] [0] b [1] [0]

i=1, j=1 b [1] [0] b [2] [0]

i=1, j=2 b [2] [0] b [3] [0]

:

ลักษณะการจัดเก็บของอะเรย์ b เป็นดังรูป สำหรับเมื่อ $j=0$ เนื่องจากมีการจัดเก็บแบบแนว

แนว

b[0][0]	b[0][1]
b[1][0]	b[1][1]

ดังนั้น การ Miss สำหรับการอ้างถึงอะเรย์ b จะเกิดในครั้งแรกเมื่อ $i=0$ คือ Miss ที่ $b[j+1][0]$ และ Miss ที่ $b[j][0]$ สำหรับ $j=0$ และหลังจาก $j=1$ ต่อไป

ภายในรอบของ i นั้นจะอ้างถึงอะเรย์ b ทั้งหมดที่ต้องการแคช ดังโคลดข้างล่างโดยตรงที่บีดเลันให้จะเป็นการ Hit และตรงกรอบสี่เหลี่ยมจะเป็นการ Miss ไปเรื่อยๆ ดังนั้น จำนวนครั้งทั้งหมดของการ Miss จะเท่ากับ 101 ครั้ง เพราะว่าสูปทำงาน 100 รอบตั้งแต่ 0-99 และเพิ่มการ Miss ครั้งแรกอีกหนึ่งครั้งเข้าไป

$i=0, j=0$	<u>b[0][0]</u>	b[1][0]
$i=0, j=1$	b[1][0]	<u>b[2][0]</u>
$i=0, j=2$	<u>b[2][0]</u>	b[3][0]
:		
$i=0, j=99$	b[99][0]	b[100][0]
$i=1, j=0$	b[0][0]	b[1][0]
$i=1, j=1$	b[1][0]	b[2][0]
$i=1, j=2$	<u>b[2][0]</u>	b[3][0]

ดังนั้น จำนวนการ Miss ทั้งหมดเป็น $150 + 101 = 251$ ครั้ง สำหรับทั้งอะเรย์ a และอะเรย์ b ถ้าทำการแยกสูปเพื่อ Prefetch อะเรย์ b และอะเรย์ a เพราะการ Prefetch อะเรย์ b ครั้งเดียวจะได้อะเรย์ b อยู่ในแคชทั้งหมด ดังนั้น สำหรับแต่ละรอบของ i และในสูปรอบอื่นๆ จะทำการ Prefetch เฉพาะอะเรย์ a ขณะที่ค่า i และ j เปลี่ยนไป

สมมติว่า Miss Penalty ใช้เวลามาก ซึ่งทำให้ต้อง Prefetch ถึง 7 รอบล่วงหน้า จะได้โคลดดังด้านล่าง

```

for( j=0; j < 100; j++) {
    Prefetch(b[j+7][0]);
    /* Prefetch b[j][0] สำหรับ 7 รอบล่วงหน้า */
    Prefetch(a[0][j+7]);
    /* Prefetch a[0][j] สำหรับ 7 รอบล่วงหน้า */
    a[0][j] = b[j][0] * b[j+1][0]
}

```

```

for (i=1; i < 3; i++)
    for (j=0; j < 100; j++) {
        Prefetch(a[i][j+7]);
        /* Prefetch a[i][j+7] สำหรับ 7 รอบล่วงหน้า */
        a[i][j] = b[j][0] * b[j+1][0]
    }
}

```

นั้นคือจะ Prefetch อะเรย์ตั้งแต่ b[7][0] ไปจน b[99][0] ในลูปแรกสำหรับอะเรย์ b และจะ Prefetch อะเรย์ตั้งแต่ a[i][7] ไปจน a[i][99] สำหรับอะเรย์ a จำนวน Miss ทั้งหมดจะเป็น

$$3 \times \left[\frac{7}{2} \right] + 7 = 12 + 7 = 19$$

เนื่องจากว่ามีการ Miss จำนวน 7 ครั้งแรกสำหรับอะเรย์ a (อะเรย์ a มี 3 แถว และจะ Miss ทุกรอบเลขคู่ คือ 0, 2, 4, ...)

<u>a[0][0]</u>	<u>a[1][0]</u>	<u>a[2][0]</u>	Miss
a[0][1]	a[1][1]	a[2][1]	
<u>a[0][2]</u>	<u>a[1][2]</u>	<u>a[2][2]</u>	Miss
a[0][3]	a[1][3]	a[2][3]	
:			

และจำนวน Miss 7 ครั้งสำหรับอะเรย์ b สำหรับอะเรย์ b[0][0]...b[6][0] และในรอบแรกจะ Miss ทั้งอะเรย์ b[0][0] และ b[1][0] และในรอบต่อๆ ไป จะ Miss ในอะเรย์ b[2][0]...b[6][0] จากนั้นตั้งแต่รอบของ b[6][0] จะไม่มีการ Miss ต่อไป

<u>b[0][0]</u>	<u>b[1][0]</u>
<u>b[1][0]</u>	<u>b[2][0]</u>
<u>b[2][0]</u>	<u>b[3][0]</u>
<u>b[3][0]</u>	<u>b[4][0]</u>
<u>b[4][0]</u>	<u>b[5][0]</u>
<u>b[5][0]</u>	<u>b[6][0]</u>
<u>b[6][0]</u>	<u>b[7][0]</u>

พิจารณาคำสั่ง Prefetch ที่เพิ่มเข้าไปเพื่อลดจำนวน Miss เป็น 232 ($251 - 19 = 232$) เท่ากับคำสั่ง Prefetch จำนวน 400 คำสั่ง

จากโดยดูข้างต้นถ้าต้องการหาว่าจะประหยัดเวลาการทำงานโดยการใช้ Prefetch ไปเท่าไร สมมติให้มีพิจารณาการ Miss ของแคชคำสั่ง และพิจารณาแต่การ Miss ของแคชข้อมูล และสมมติว่าการ Prefetch ทำพร้อมกันได้และทำพร้อมกับการ Miss ได้ (ไม่มีการ Stall เกิดขึ้น) ถ้าลูปเดิมใช้ 7 ไบเกิลต่อ 1 รอบ และการ Prefetch ในลูปแรกใช้ 9 ไบเกิลต่อ 1 รอบ และ การ Prefetch ในลูปที่สองใช้ 8 ไบเกิลต่อ 1 รอบ

ลูปเดิมทำ 3×100 รอบ ดังนั้น ใช้ $300 \times 7 = 2,100$ ไบเกิลรวม Miss ที่เกิดจากแคชแล้ว ซึ่งมีทั้งหมดมี 251 Miss แต่ละครั้งของการ Miss มีเวลา Penalty เป็น 50 ไบเกิล ดังนั้น ทั้งหมด จะใช้ $2,100 + 50 \times 251 = 14,650$ ไบเกิล

สำหรับลูปที่ใช้ Prefetch ลูปแรกทำ 100 ครั้ง Prefetch อะเรย์ b แต่ละรอบใช้ 9 ไบเกิล และมี Miss ทั้งหมด 11 ครั้งแต่ละครั้งมี Penalty เท่ากับ 50 ไบเกิลรวมได้ $900 + 11 \times 50 = 1,450$ ไบเกิล

ลูปที่สองทำทั้งหมด 2×100 ครั้ง แต่ละครั้งใช้ 8 ไบเกิลและมีจำนวน Miss = 8 ครั้ง แต่ละครั้งของการ Miss ใช้ 50 ไบเกิลรวมได้ $1,600 + 8 \times 50 = 2,000$ ไบเกิล ดังนั้น ทั้งสองลูปใช้เวลารวม $2,000 + 1,450 = 3,450$ ไบเกิล

เปรียบเทียบกับลูปเดิมใช้ 14,650 ไบเกิล ซึ่งเร็วกว่า $14,650/3,450 = 11$ เท่า

10.4.7 การใช้เทคนิคการอปตีไมซ์จากตัวแปลภาษา

การใช้เทคนิคของตัวแปลภาษาทำการอปตีไมซ์โดยได้หลายแบบ เช่น การรวมอะเรย์ (Merging Array) เพื่อเพิ่มประโยชน์จาก Locality เชิงพื้นที่เพื่อบรรบประภารมอาจจะอ้างถึงอะเรย์ที่ต่างกันแต่ใช้ตำแหน่ง Index เดียวกันในเวลาเดียวกัน ดังนั้น ถ้าทำให้พื้นที่อะเรย์เหล่านั้นติดกันจะได้ประโยชน์จาก Locality เชิงพื้นที่

พิจารณาตัวอย่างโดยรวม และหลังรวมในตัวอย่างการประกาศโครงสร้าง Struct ในภาษา C

```
int val[size];
int key[size];
```

หลังรวมจะได้

```
struct merge {
    int val;
    int key;
} merge_array[size];
```

เริ่มต้นอะเรย์ทั้งสองจากจะถูกจัดสรรหน่วยความจำในตำแหน่งที่ต่างกัน ลักษณะการทำงานของโค้ดสำหรับอะเรย์แบบนี้ ผู้เขียนอาจจะต้องการการเข้าถึงอะเรย์ทั้งสองพร้อมกัน เช่น ถ้าหากค่าในตัวแปร key พบที่จะไปนำเอาข้อมูลในตัวแปร val ดังนั้น การรวมเป็น Struct อันเดียวกันจะทำให้ได้พื้นที่ที่ติดกัน ระหว่างฟิลด์ val และ key ที่สัมพันธ์กัน และการเข้าถึงฟิลด์ key จะได้ค่าในฟิลด์ val เข้ามาในแคชด้วย

อีกเทคนิคหนึ่นการสลับลูป (Loop Interchange) เป็นการสลับการเปลี่ยนลูป Index ถ้าโปรแกรมอ้างถึงข้อมูลในอะเรย์ที่ไม่เป็นตามแนวแ夸ก์จะพยายามปรับทำให้เป็นตามแนวแ夸 เช่น

```
for (j=0; j < 100; j++)
    for (i=0; i < 100; i++)
        y[i][i] = 2 * y[i][j];
```

ลูปแรกเป็นการทำ Index จาก colum แล้วตามด้วยแ夸 ซึ่งเป็นการอ้างถึงอะเรย์ในแบบแนวคอลัมน์ซึ่งไม่เหมาะสมกับภาษาซี จะไม่ให้ประโยชน์ในแง่ Locality เชิงพื้นที่ จึงปรับลูปเป็น

```
for (i=0; i < 100; i++)
    for (j=0; j < 100; j++)
        y[i][i] = 2 * y[i][j];
```

ก็จะได้ประโยชน์จาก Locality เชิงพื้นที่

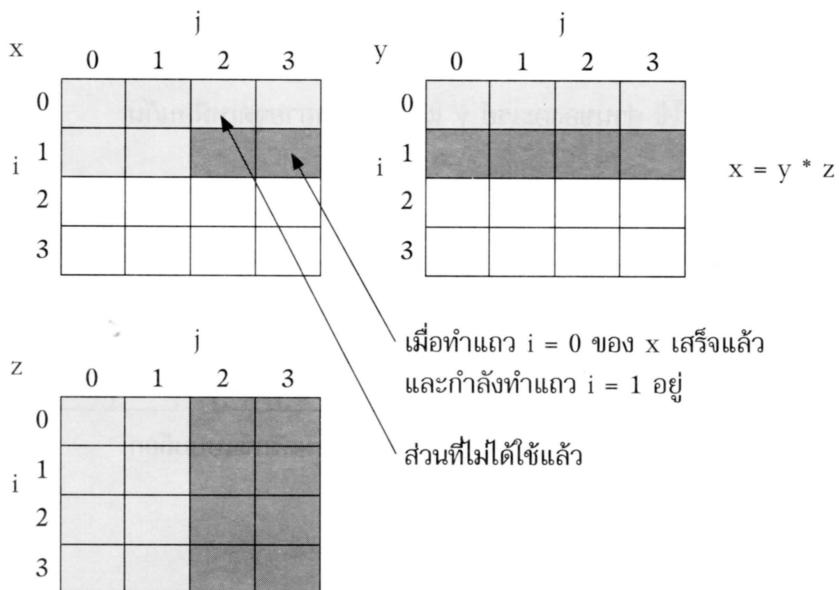
อีกวิธีเรียกว่าวิธี Blocking เป็นการลดจำนวนการ Miss โดยใช้ Locality เชิงเวลา โดยกรณีที่ใช้วิธีการปรับแนวแ夸หรือแนวคอลัมน์ให้มีได้ พิจารณาตัวอย่างการคุณเมตริกซ์

```

for (i=0; i < N ; i++)
    for (j=0; j < N; j++) {
        p =0;
        for (k=0; k < N; k++) {
            p = p +y[i][k] * z[k][j];
        }
        x[i][j] = p;
    }
}

```

มีการเข้าถึงดังรูป 10.5



รูปที่ 10.5 การเข้าถึงข้อมูลในการคูณเมตริกซ์

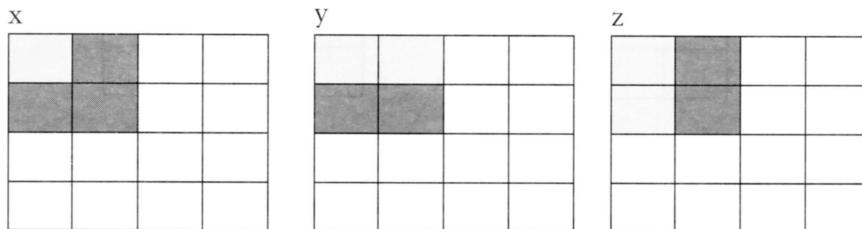
ในการคำนวณอะเรย์ $x[i][j]$ จะต้องอ่านมาจาก y และคอลัมน์จาก z จึงไม่สามารถใช้ได้รูปแบบแวดหรือคอลัมน์อย่างเดียวเท่านั้น จึงปรับเป็นแบบบล็อกซึ่งจะทำการคูณทีละ Submatrix หรือบล็อก ในโอดาดข้างล่างจะทำเป็นบล็อกขนาด $B \times B$ โดยเรียก B ว่า Block Factor เมื่อ $B < N$ ให้เลือกขนาดของ B ให้เหมาะสมที่จะเก็บทั้งบล็อกไว้ในแคชได้เพื่อให้การคำนวณของบล็อกนั้นจะไม่เกิดการ Miss

```

for (jj=0; jj < N; jj=jj+B)
for (kk=0; kk < N; kk = kk+B)
for (i=0; i < N ; i++)
    for (j=jj; j < min(jj+B-1,N); j++) {
        r =0;
        for (k=kk; k < min(kk+B-1); k++) {
            r = r +y[i][k] * z[k][j];
        }
        x[i][j] = x[i][j]+ r;
    }
}

```

จะเป็นลักษณะการคูณดังรูป 10.6 และถ้าปรับขนาด Subblock ของอะเรย์ x, y, z จะทำให้ทั้ง Subblock น้อยลงแค่ไหน ทำให้เกิด Locality เขิงเวลาได้ โดยการคำนวณของค่า $x[i][j]$ หลายจุดจะใช้ ส่วนของอะเรย์ y และ z ในหลายจุดเหมือนกัน



รูปที่ 10.6 การคูณแบบเมตริกซ์แบบล็อก

■ 10.5 สรุป

หน่วยความจำเป็นส่วนสำคัญของคอมพิวเตอร์ที่มีผลต่อประสิทธิภาพ การเพิ่มประสิทธิภาพ ของเครื่องส่วนหนึ่งต้องเพิ่มประสิทธิภาพการใช้หน่วยความจำด้วย ในบทนี้ได้กล่าวถึงวิธีการต่างๆ ทั้งด้านฮาร์ดแวร์และซอฟต์แวร์ในการเพิ่มประสิทธิภาพการใช้งานหน่วยความจำ ซึ่งพิจารณาจาก สมการเวลาของ CPU Time และเวลาการเข้าถึง โดยมองแต่ละปัจจัย ได้แก่ การ Miss ตั้งแต่ การ Miss สำหรับการอ่านและการเขียน Miss Penalty และเวลาในการ Hit นอกจากนี้ได้แนะนำ วิธีการปรับแต่ละปัจจัยและแสดงตัวอย่างการคำนวณเพิ่มแสดงให้เห็นประสิทธิภาพในเขิงตัวเลข

คำถ้ามก้าวบท

1. จากสมการ Access Time และว่าปัจจัยใดที่มีผลต่อการลดเวลาที่นี้บ้าง แต่ละวิธีทำอย่างไร
2. พิจารณาระบบหน่วยความจำที่ประกอบด้วยลำดับชั้นจากใกล้สีพิழูไปไกลสีพิழูดังนี้ แคช L1, L2, DRAM, Harddisk โดยกำหนดให้ L1 และ L2 เป็นแบบเดียวกัน บัสระหว่าง L1 และ L2 เป็นแบบ 32 บิต และบัสระหว่าง L2 กับ DRAM ขนาด 64 บิต และบัสระหว่าง DRAM และชาร์ดดิสก์กว้างเท่ากับ 64 บิตเท่านั้น

กำหนดลักษณะของแต่ละระดับ ดังนี้

- L1 มีขนาด 32 กิโลไบต์ เป็นแบบ Direct map มีอัตราการ Miss เท่ากับ 1% มีขนาดบล็อกเท่ากับ 16 ไบต์ ใช้วิธี Write Through
- L2 มีขนาด 128 กิโลไบต์ เป็นแบบ 2-Way Set Associative และมีขนาดบล็อกเท่ากับ 32 ไบต์ มี Access Time เท่ากับ 10 นาโนวินาที เชื่อมต่อกับ L1 ด้วยบัสความเร็ว 250 MHz มีอัตรารับส่งข้อมูลอยู่ที่ 128 บิตต่อ 1 ไซเกิล มีอัตราการ Miss เท่ากับ 10% และ 50% ของ จำนวนบล็อกทั้งหมดเป็น Dirty block
- หน่วยความจำหลักเป็นลักษณะ DRAM มีความกว้างเท่ากับ 64 บิต มี Access Time เท่ากับ 60 นาโนวินาที มีอัตรารับส่งข้อมูล เท่ากับ 1 เวอร์ด ต่อไซเกิล โดย 1 เวอร์ด มีขนาด 64 บิต ซึ่งเท่ากับความกว้างของบัส และความเร็วของบัสเท่ากับ 100 MHz

กำหนดให้ความถี่ของซีพิซีเป็น 1.1 GHz และ CPI ที่ไม่คิดผลของหน่วยความจำเท่ากับ 0.5 พิจารณา Benchmark ที่มีจำนวนคำสั่งประเภท Load (คิดเฉพาะข้อมูลไม่รวมคำสั่ง) 20% และคำสั่ง Store (ส่วนของข้อมูล) 5% ของจำนวนคำสั่งทั้งหมดที่ทำงาน

2.1 จงหา Average Memory Access Time ของการอ่านข้อมูลและการอ่านคำสั่ง

2.2 จงหา Average Memory Access Time ของการเขียนข้อมูล กำหนดให้การเขียนหนึ่ง

ครั้งใช้เวลามากขึ้น 50% ไม่มี บัฟเฟอร์ช่วยสำหรับการเขียนใดๆ ทั้งสิ้น

2.3 จงหา CPI รวมเมื่อผลของหน่วยความจำแล้วอธิบาย

3. พิจารณาเครื่องคอมพิวเตอร์ที่รันด้วยความถี่ 2.5 GHz มี CPI เท่ากับ 1 เวลาที่ใช้สำหรับ Miss สำหรับการอ่านเท่ากับ 100 นาโนวินาที สำหรับวิธีการเขียนเป็นแบบ Write Through กำหนดให้ การเขียนเวิร์ดขนาด 32 บิต ใช้เวลา 100 นาโนวินาที และสำหรับวิธีการเขียนแบบ Write back กำหนดให้การเขียนบล็อกขนาด 32 ไบต์ใช้เวลา 200 นาโนวินาที สมมติให้แคช มีความจุเท่ากับ 128 กิโลไบต์ (คิดเฉพาะแคชข้อมูล) พิจารณาระบบที่หน่วยความจำของเครื่องนี้รูปแบบต่างๆ ดังนี้

- A: รูปแบบ Direct-Mapped ขนาดบล็อกเท่ากับ 32 ไบต์ ใช้วิธีการ Write Back และ Write Allocate
- B: รูปแบบ 2-Way Set Associative มี ขนาดบล็อกเท่ากับ 32 ไบต์ ใช้วิธีการ Write Through และไม่มีการใช้ Write Allocate

- 3.1 จงอธิบายกรณีตัวอย่างของโปรแกรมที่ทำให้รูปแบบ A รันเร็วกว่ารูปแบบ B และเร็วกว่าเท่าไร
- 3.2 จงอธิบายกรณีตัวอย่างของโปรแกรมที่ทำให้รูปแบบ B รันเร็วกว่ารูปแบบ A และเร็วกว่าเท่าไร

4. พิจารณาเครื่องที่มีความถี่ 1 GHz ถ้าเครื่องนี้ทำงานกับโปรแกรมหนึ่งซึ่งมีอัตราการ Miss = 5% มี Access Time ของ DRAM เป็น 150 นาโนวินาที

- 4.1 จงหา CPI ของเครื่องเมื่อรันโปรแกรมนี้
- 4.2 ถ้าเพิ่มแคช L2 ลงไปซึ่งมี Access Time 10 นาโนวินาที และทำให้ Miss Rate ของ DRAM ลดลงเหลือ 3% CPI ใหม่ของโปรแกรมนี้จะดีขึ้นหรือไม่ เท่าไร
- 4.3 ถ้าโปรแกรมนี้มีทั้งการอ่านและเขียน โดยใน 5% นี้แบ่งเป็นอัตราการ Miss สำหรับการเขียน = 2% และอัตราการ Miss สำหรับการอ่าน = 3% และการเขียนเป็นแบบ Write Through โดยในการเขียนจะเพิ่มเวลาในการเขียนบัฟเฟอร์ไปอีก 20 นาโนวินาทีต่อ

บล็อกข้อมูล โดยเฉลี่ย 1 คำสั่งมีการเขียน 10% ของบล็อก จงหา CPI เฉลี่ยของโปรแกรมนี้เมื่อพิจารณาการเขียนแบบ Write Through

5. พิจารณาโค้ดต่อไปนี้ จงหาจำนวน Miss ในแคชที่เกิดขึ้นทั้งหมด กำหนดให้ สมมติว่าแคชมีบล็อกขนาด 16 ไบต์ สมมติให้ใช้การ Write-Back ซึ่งเป็นการ Write Allocate และขนาดของอะเรย์ a และ b แต่ละตัวเป็น 8 ไบต์ เก็บเลขทศนิยม และ a เป็นอะเรย์ขนาด 100 และ $\times 11$ คอลัมน์ ส่วน b เป็นอะเรย์ ขนาด 10 และ $\times 1$ คอลัมน์

```
for (i=0; i< 100; i++)
    for (j=0; j < 10; j++)
        A[i][j] = A[i][j+1] + B[j];
```

ตัวแปรได้ใช้ประโยชน์จาก Locality เขิงพื้นที่และ/หรือ Locality เขิงเวลาบ้าง

6. จากโค้ดข้างบน ถ้าจะใช้คำสั่ง Prefetch มาช่วยทำให้การทำงานเร็วขึ้น ควรทำอย่างไร
7. จงยกตัวอย่างที่เทคนิคที่ใช้ประโยชน์จาก Locality เขิงพื้นที่
8. การทำ Loop Folding ให้ประโยชน์อย่างไร อธิบาย
9. จงอธิบายวิธีการลด Miss Penalty มา 2 倍 อย่างไร
10. ให้ยกตัวอย่างวิธีการลดเวลาของ การ Hit มา 2 倍 อย่างไร
11. การเพิ่มระดับของแคชเป็นการลดค่าໄด และอาจจะมีผลต่อการเพิ่มปัจจัยใด
12. จากโค้ดการคูณเมตริกซ์ จงอธิบายการใช้ประโยชน์จาก Locality เขิงพื้นที่และ Locality เขิงเวลา

