ปฏิบัติการที่ 6: วิเคราะห์พฤติกรรมการใช้งาน Cache ของโปรแกรม

1. วิเคราะห์โปรแกรม

```
# define N 64
typedef int array_t[N][N];

int sum1(array_t a) {
      int i, j;
      int sum = 0;
      for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                  sum += a[i][j];
      return sum;
}

int sum2(array_t a) {
      int i, j;
      int sum = 0;
      for (j = 0; j < N; j++)
            for (i = 0; i < N; i++)
                  sum += a[i][j];
      return sum;
}

int sum3(array_t a) {
      int i, j;
      int sum = 0;
      for (i = 0; i < N; i+=2)
            for (j = 0; j < N; j+=2)
                  sum += (a[j][i] + a[j][i+1] + a[j+1][i] +
            a[j+1][i+1]);
      return sum;
}
```

ขนาด block เท่ากับ 16 byte สามารถเก็บตัวเลขได้ 16 / 4 = 4 ตัว
Cache มีขนาด 4 kbyte มีจำนวน block เท่ากับ 4000 / 16 = 250 block

Sum 1 :

จากโค้ดการเข้าถึงข้อมูลจะไปในทิศ ซ้ายไปขวาแล้วขึ้นแถวใหม่ ซึ่งหากการเก็บข้อมูลใน memory เป็นแบบ row-major จะมี spatial locality ทำให้ miss ทุกๆ 4 ช่องการคำนวน
N หาร 4 ลงพอดี ดังนั้น
Miss rate ของ sum1 = 0.25

Sum 2:

จากโค้ดการเข้าถึงข้อมูลจะเป็นในรูปแบบ บนลงล้างแล้วขึ้นหลักใหม่ในทางขวา
ดังนั้น N ช่องแรกนั้น miss แต่ 3 x N ช่อง ต่อมาไม่ miss เพราะข้อมูลจะอยู่ใน cache แล้ว
ซึ่ง Sum2 จะเข้าถึง cache N x N ครั้ง
ดังนั้น Miss Rate = (N x N / 4) / (N x N) = 0.25

Sum 3:

จากโค้ดการเข้าถึงข้อมูลจะเป็นในรูปแบบคล้าย Sum2 แต่เข้าถึงทีละ 4 ช่องพร้อมกัน ดังนั้นการเข้าถึงจะ miss N / 2 ก้อนแรก จะ miss ครั้งละ 2 แต่ N / 2 ก้อนถัดไปจะไม่ miss เพราะเก็บข้อมูลลง cache แล้ว
โดยก้อนหนึ่งจะมี 4 ช่อง
ดังนั้น Miss Rate = 2 * (N / 2) / (N * 4) = 0.25

	Miss Rate		
	sum1	sum2	sum3
N = 64	0.25	0.25	0.25
N = 60	0.25	0.25	0.25

2. วิเคราะห์การทำงานของโปรแกรมการคูณเมตริกซ์

1. ตอบคำถามต่อไปนี้

1. ลำดับลูปแบบใดที่มีค่า cycles / loop iteration สูงที่สุด
   ตอบ jki
2. ลำดับลูปแบบใดที่มีสมรรถนะสูงที่สุด
   ตอบ ikj
3. อธิบายเปรียบเทียบลำดับลูปที่มีสมรรถนะสูงสุดกับลำดับลูปที่มีสมรรถนะต่ำสุดว่ามีความแตกต่างกันอย่างไร
   ตอบ จากลักษณะการเข้าถึงช่องของแต่ละข้อมูลแต่ละแบบได้ว่า

	jki	ikj
A	บนลงล่างแล้วเปลี่ยนหลักไปทางขวา	ซ้ายไปขวาแล้วเปลี่ยนแถวลงล่าง
B	บนลงล่างแล้วเปลี่ยนหลักไปทางขวา	ซ้ายไปขวาแล้วเปลี่ยนแถวลงล่าง
C	บนลงล่างแล้วเปลี่ยนหลักไปทางขวา	ซ้ายไปขวาแล้วเปลี่ยนแถวลงล่าง

จากลักษณะการเข้าถึงข้อมูลจะเห็นได้ว่า ikj จะมี spatial locality ที่สูงทำให้เกิด miss น้อยกว่ามากในขณะที่ jki เดินทางในแนวตั้งทำให้เกิด miss ตลอดจึงช้ากว่า

4. ทำไมเมื่อ n มีค่ามากขึ้น ค่า cycles / loop iteration จึงมีค่าสูงขึ้น
   ตอบ เพราะจำนวนช่องหากมากขึ้นในขณะที่ cache มีขนาดคงที่จะทำให้ในกรณีที่มีการอ่านซ่ำจะทำให้เกิด miss ได้หากพื้นที่ไม่พอ อย่าง ikj จะมีการอ่าน array B ซ้ำทั้ง array ป่อยมากถ้า array B ใหญ่เกินไป ก็จะเก็บ cach ได้ไม่พอ

2. ตอบคำถามต่อไปนี้

1. ทำไมเมื่อใช้ blocking โดยให้ block size มีขนาดคงที่ (50) ค่า cycles / loop iteration สำหรับลำดับ ijk จึงไม่แปรปรวนและเพิ่มขึ้นเมื่อ n เพิ่มขึ้นเหมือนในกรณีที่ไม่ใช้ blocking
   ตอบ จากเดิมเหตุผลตอนที่ยังไม่ได้ทำการ blocking ที่ว่าทำไมเวลาขนาดของ matrix ใหญ่ขึ้นทำไม cycles / loop iteration ถึงมากคื้นนั้น การ blocking จะมาแก้ปัญหาโดยเพิ่ม temporal locality โดยแทนที่จะคูณ matrix ขนาดใหญ่ที่เสี่ยงการที่ cach จะไม่พอแล้ว miss แค่ทำการ แยกส่วนของ matrix ให้เล็กลงแล้วคูณให้เสร็จไปการทำแบบนี้จะทำให้ เกิดการใช้ข้อมูลซ้ำในปริมาณที่ cache พอรับไหวทำให้ลดการ miss จากการที่ cache ไม่พอได้
2. จากที่เราได้เรียนรู้มาว่าจำนวน miss เมื่อทำ blocking จะมีค่าประมาณ 1/(4B) * n3 นั่นคือเมื่อขนาดของ block B มีค่ามากขึ้น จำนวน miss น่าจะมีค่าลดลง แต่ทำไมผลจากการรันจึงดูเหมือนไม่เป็นไปตามสูตรนี้
   ตอบ การเพิ่มขนาด block นั้นเสี่ยงที่จะลด temporal locality เพราะ แต่ละ block ก็คือการคูณ matrix เหมือนกันยิ่งใหญ่ยิ่งเปลือง cache