204324 ปฏิบัติการระบบคอมพิวเตอร์

ปฏิบัติการที่ 6: วิเคราะห์พฤติกรรมการใช้งาน Cache ของโปรแกรม

1. วิเคราะห์โปรแกรมต่อไปนี้

```
# 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;
}
```

จากโค้ดด้านบน ถ้าให้ว่า CPU มี cache ขนาด 4 Kbytes แบบ direct-mapped โดยมีขนาดของ block เท่ากับ 16 bytes จงคำนวณหา miss rate (โดยประมาณ) เมื่อรันฟังก์ชั่น sum1 sum2 และ sum3 เมื่อ N มีค่า 64 และ 60 โดยให้ว่าการติดต่อไปที่ memory ของฟังก์ชั่นทั้งสามเกิดขึ้นเฉพาะส่วน array เท่านั้น ตัวแปรตัวอื่นๆถูกเก็บอยู่ใน register ทั้งหมด ค่า address เริ่มต้นของ array a อยู่ที่ 0x08000000

แสดงวิธีทำและเติมตารางต่อไปนี้

	Miss Rate		
	sum1	sum2	sum3
N = 64			
N = 60			

ไม่มีวิธีทำ = ไม่มีคะแนน

2. วิเคราะห์การทำงานของโปรแกรมการคูณเมตริกซ์

ดาวน์โหลด zip ไฟล์ matmult.zip (โปรแกรมนี้ดัดแปลงมาจากโปรแกรมชื่อเดียวกันที่ใช้ใน csapp.cs.cmu.edu) จากนั้น unzip เพื่อเข้าไปใน directory ชื่อ matmult และทำการ make เพื่อให้ได้ executable สองตัวคือ mm และ bmm ซึ่งเป็นโปรแกรมการคูณเมตริกซ์ที่ไม่ใช้ blocking และที่ใช้ blocking ตามลำดับ ใช้คำสั่งต่อไปนี้

```
make clean
make
```

จากนั้นรัน mm

```
./mm
```

ตอบคำถามต่อไปนี้

- ลำดับลูปแบบใดที่มีค่า cycles / loop iteration สูงที่สุด
- ลำดับลูปแบบใดที่มีสมรรถนะสูงที่สุด
- อธิบายเปรียบเทียบลำดับลูปที่มีสมรรถนะสูงสุดกับลำดับลูปที่มีสมรรถนะต่ำสุดว่ามีความแตกต่างกันอย่างไร
- ทำไมเมื่อ n มีค่ามากขึ้น ค่า cycles / loop iteration จึงมีค่าสูงขึ้น

รัน bmm (ใช้เวลานานระดับหนึ่งสำหรับ CPU รุ่นเก่าที่ไม่ค่อย “แรง”)

```
./bmm
```

ตอบคำถามต่อไปนี้

- ทำไมเมื่อใช้ blocking โดยให้ block size มีขนาดคงที่ (50) ค่า cycles / loop iteration สำหรับลำดับ ijk จึงไม่แปรปรวนและเพิ่มขึ้นเมื่อ n เพิ่มขึ้นเหมือนในกรณีที่ไม่ใช้ blocking
- จากที่เราได้เรียนรู้มาว่าจำนวน miss เมื่อทำ blocking จะมีค่าประมาณ $1/(4B) * n^3$ นั่นคือเมื่อขนาดของ block B มีค่ามากขึ้น จำนวน miss น่าจะมีค่าลดลง แต่ทำไมผลจากการรันจึงดูเหมือนไม่เป็นไปตามสูตรนี้

3. ดัดแปลงโปรแกรมโดยใช้เทคนิค Blocking

จากโค้ด transpose.c ที่ให้มา คอมไพล์และรันโปรแกรมนี้

```
gcc -O3 -o tp transpose.c
./tp
```

- ดัดแปลงโปรแกรมนี้เพื่อวัดเวลาที่ใช้ในการ transpose เมตริกซ์ขนาด 100x100 200x200 500x500 1000x1000 2000x2000 และ 5000x5000
- ดัดแปลงโปรแกรมนี้โดยใช้เทคนิค blocking ให้ชื่อโปรแกรมใหม่ว่า transposeB.c แล้ววัดเวลาเปรียบเทียบกับ transpose.c (ใช้ขนาดของเมตริกซ์ตามที่ใช้วัดเวลาในข้อแรก) พร้อมสรุปและวิจารณ์ผลการทดลอง

สิ่งที่ต้องส่ง

- ไฟล์ lab6_answer.pdf ที่ตอบคำถามและรายงานผลการทดลอง

- ไฟล์ transposeB.c ที่ดัดแปลง transpose.c โดยใช้เทคนิค blocking

การส่งงาน:

- **นำงานที่ต้องส่งใส่ไว้ในโฟลเดอร์ชื่อ**

studentID1_firstname1_studentID2_firstname2_lab6

โดย studentID และ firstname คือเลขประจำตัวและชื่อแรกของสมาชิกที่ทำปฏิบัติการร่วมกัน จากนั้น zip โฟลเดอร์นี้แล้วส่ง zip ไฟล์มาที่ Google Classroom ของวิชาก่อนกำหนดส่ง

ถ้าถึงกำหนดส่งแล้วนิสิตยังทำปฏิบัติการไม่สมบูรณ์ ขอให้ชี้แจงอุปสรรค บั๊กต่างๆที่พบมาในไฟล์ README.pdf แล้วใส่ไฟล์นี้มาในโฟลเดอร์เดียวกับงานอื่นๆที่ต้องส่ง