

62010336 นายนรุ่งราษฎร์
62010713 นายวุฒิ จันทร์สมน

ภาคผนวก M

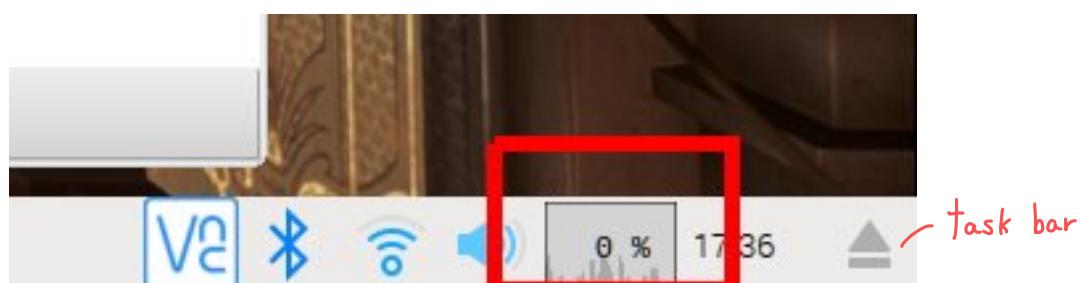
การทดลองที่ 13 การพัฒนาอัลกอริทึมแบบขนานด้วย OpenMP

การพัฒนาอัลกอริทึมแบบขนานชนิดนี้พิจารณาเป็นต้องอาศัยภาษาคอมพิวเตอร์ระดับสูง เช่น ภาษา C/C++ ภาษา Java เป็นต้น เพื่อช่วยลดเวลา.ran (Run Time) ซึ่งเท่ากับเร่งความเร็ว (Speedup) ให้อัลกอริทึมหรือโปรแกรม โดยการสร้างthreadผู้ช่วย (Worker Thread) และมอบหมายงานให้ไปรันบนชนิดปิ๊คคอร์ที่ยังว่างอยู่ ผู้อ่านสามารถประยุกต์ใช้หลักการนี้บนเครื่องคอมพิวเตอร์ทั่วไปจนถึงเครื่องซูเปอร์คอมพิวเตอร์ตามเนื้อหาในบทที่ 8 ดังนั้น การทดลองมีวัตถุประสงค์ดังนี้

- เพื่อพัฒนาโปรแกรมภาษา C ด้วยไลบรารี OpenMP ให้สามารถทำงานแบบมัลติ-thread และใช้งานชนิดปิ๊คคอร์ได้เต็มที่
- เพื่อเรียนรู้การวัด CPU Utilization (%CPU) เวลาจริง (T_{real}) เวลาผู้ใช้ (T_{user}) และเวลาระบบ (T_{sys}) ในชนิดปิ๊คคอร์
- เพื่อทำความเข้าใจการวัดประสิทธิภาพของอัลกอริทึมแบบขนานด้านความซับซ้อนเชิงเวลาด้วยพีซคณิต BigO และตัวชี้วัด Speedup จากเวลาที่วัดได้

Elapsed
Wallclock
OS

M.1 การวัด CPU Utilization



รูปที่ M.1: กราฟแสดงการใช้งานชนิดปิ๊ค (CPU Usage Monitor) ย้อนหลังและค่าสรุป ณ เวลาปัจจุบัน ที่มา: abload.de

ผู้อ่านสามารถติดตั้งเครื่องมือและการกราฟแสดงการใช้งานชนิดปิ๊ค (CPU Usage Monitor) ย้อนหลังและค่าสรุป ณ เวลาปัจจุบันของบอร์ด Pi3 ประกอบการทดลองที่ 13 ตามขั้นตอนเหล่านี้

1. เลื่อนมาส์ปีบันตำแหน่งว่างของ Task Bar
 2. คลิกขวา เพื่อให้เมนูต่อไปนี้ปรากฏขึ้นแล้วคลิกซ้ายเลือก Add/Remove Panel Items
 3. คลิกที่แท็บ Panel Applets
 4. เลื่อนรายการขึ้นลงเพื่อหารายการชื่อ CPU Usage Monitor และคลิก Add
 5. กดปุ่ม Up และ Down เพื่อวางตำแหน่งของ CPU Usage Monitor ในตำแหน่งที่ต้องการ โปรดสังเกตรายชื่อ เมื่อได้ตำแหน่งที่ต้องการแล้วกด Close หมายเหตุ Spacer หมายถึง ช่องว่างที่คั่นระหว่าง Application ที่อยู่บน Task Bar
 6. สังเกตด้านขวาของ Task Bar จะมีจอสีเทาขนาดเล็กแสดงเป็นกราฟแท่ง โดยแท่งขาวสุดคือ วินาทีล่าสุด
 7. เลื่อนมาส์ปีบันกราฟแล้วคลิกขวาเพื่อเพิ่มการแสดงผลเป็นตัวเลขหน่วยเป็นเปอร์เซ็นต์ (%)
 8. ทดสอบการทำงานโดยการเปิดคลิปเดียวกันบน YouTube.com ที่ความละเอียดแตกต่างกัน เช่น 240p, 480p และ 720p ทีลักษณะเพื่อให้เห็นค่า $\%CPU_{max}$ ที่แตกต่าง
- ตัว
รับข้อมูลจาก
CPUทำงานมาก

M.2 การคูณแมทริกซ์แบบขนาด

$$C = A \times B \quad (\text{M.1})$$

การคูณแมทริกซ์เป็นพื้นฐานของการคำนวนพื้นฐานทางวิทยาศาสตร์ และวิศวกรรมศาสตร์ กำหนดให้แมทริกซ์จตุรัส A ขนาด $N \times N$ สามารถเขียนในรูปแบบของอะเรย์ 2 มิติในภาษา C/C++ ได้ดังนี้

$$A = (A[i][j])$$

โดยตัวนี้ตัวแรก i คือ หมายเลขแถว มีค่าตั้งแต่ 0 ถึง $N-1$ ตัวนี้ตัวที่สอง j คือ หมายเลขคอลัมน์ มีค่าตั้งแต่ 0 ถึง $N-1$ ดังนั้น

๖๖๗

$$A = \begin{pmatrix} A[0][0] & A[0][1] & \dots & A[0][N-1] \\ A[1][0] & A[1][1] & \dots & A[1][N-1] \\ \vdots & \vdots & \ddots & \vdots \\ A[N-1][0] & A[N-1][1] & \dots & A[N-1][N-1] \end{pmatrix}$$

เมื่อทำความเข้าใจพื้นฐานของแมทริกซ์ในรูปแบบของอะเรย์ 2 มิติแล้ว ผู้อ่านสามารถทำการทดลองตามขั้นตอนต่อไปนี้

1. ย้ายและสร้างไดเรกทอรี /home/Pi/Lab13 บนโปรแกรม Terminal ด้วยคำสั่งต่อไปนี้ตามลำดับ

```
$ cd /home/Pi/
$ mkdir Lab13
$ cd Lab13
$ nano multMatrix.c
```

2. กรอกโปรแกรมต่อไปนี้ด้วยโปรแกรม nano และบันทึกในไฟล์ชื่อ **multMatrix.c** ในไดเรกทอรีที่สร้างไว้

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h> } Header เหล่า
#include <omp.h>

#define N 200 ค่าคงที่ 200 x 200
float A[N][N], B[N][N], C[N][N]; // matrices of NxN elements

int main () {
    /* DECLARING VARIABLES */
    int i, j, k; // indices for matrix multiplication
    float t_mul; // Multiply time เวลา C-2 - C-1 (stop-start)
    clock_t c_1, c_2; // start time and stop time

    /* FILLING MATRICES WITH RANDOM NUMBERS */
    srand ( time(NULL) );
    for(i=0;i<N;i++) {
        for(j=0;j<N;j++) {
            A[i][j] = (rand()%100);
            B[i][j] = (rand()%100);
        }
    }
    /* MATRIX MULTIPLICATION */
    printf("Max number of threads: %i \n",omp_get_max_threads());
    #pragma omp parallel
        printf("Number of threads: %i \n",omp_get_num_threads());
    จับเวลา เริ่มต้น → c_1=time(NULL); // time measure: start time
    #pragma omp parallel for private(k, j)
    {  

        loop 3 รอบ
        for(i=0;i<N;i++) {
            for(j=0;j<N;j++) {
                C[i][j]=0.; // set initial value of resulting matrix C = 0
                for(k=0;k<N;k++) {
                    C[i][j]=C[i][j]+A[i][k]*B[k][j];
                }
            }
        }
    }  

    จับเวลา ล็อกสุดท้าย → c_2=time(NULL); // time measure: stop time
    t_mul = (float)(c_2-c_1); // Multiply time
```

```

printf("Mutiply Time: %f \n", t_mul);
}

/* TERMINATE PROGRAM */
return 0;
}

```

3. exit ออกจากโปรแกรม nano เพื่อคอมไพล์โปรแกรมด้วยคำสั่งต่อไปนี้

```
$ gcc -fopenmp multMatrix.c -o mulMatrix
```

 compiler flag

แก้ไขหากมีข้อผิดพลาดจนกว่าจะคอมไпал์โปรแกรมสำเร็จและมีไฟล์ชื่อ mulMatrix

4. ตั้งค่าจำนวน-thread n=1 ของโปรแกรมด้วยคำสั่งต่อไปนี้

```
$ export OMP_NUM_THREADS=1
```

5. รันโปรแกรมจับเวลาด้วยคำสั่ง time ดังนี้จำนวน 5 ครั้งเพื่อหาค่าเฉลี่ย ขณะทำการทดลองขอให้ผู้อ่านใช้นาฬิกาข้อมือจับเวลาไปพร้อมๆ กัน เพื่อเปรียบเทียบกับค่าของ $T_{mul,n}$ และ T_{real}

```
$ time ./mulMatrix
```

ซึ่งจะรายงานผลการจับเวลาการทำงานของทั้งโปรแกรมในแต่ละมุ่งต่างๆ

6. จดบันทึกค่า CPU Utilization สูงสุดหรือ $\%CPU_{max}$ ที่สังเกตได้ หากค่าเฉลี่ยของ $T_{mul,n}$ T_{real} T_{user} และ T_{sys} ที่ได้ลงในตารางที่ M.1

ตารางที่ M.1: เวลาและ $\%CPU_{max}$ ของการคุณแม่ทริกซ์ที่ขนาด N และจำนวนเหตุเด่ากับ 1, 2, 4, 8 เหตุ

เวลาเฉลี่ย	$N=200$ (วินาที)	$N=400$ (วินาที)	$N=800$ (วินาที)	$N=1000$ (วินาที)
$n=1$ เหตุ				
$T_{mul,1}$	0.000	1.000	11.200	29.600
T_{real}	0.231	1.382	11.359	29.998
T_{user}	0.219	1.363	11.313	29.948
T_{sys}	0.012	0.019	0.045	0.043
$\%CPU_{max}$	5%	16%	25%	25%
$n=2$ เหตุ				
$T_{mul,2}$	0.000	0.600	5.800	15.000
T_{real}	0.142	0.753	5.881	15.169
T_{user}	0.228	1.417	11.543	30.034
T_{sys}	0.011	0.016	0.039	0.054
$\%CPU_{max}$	7%	22%	48%	50%
$n=4$ เหตุ				
$T_{mul,4}$	0.000	0.600	3.000	7.800
T_{real}	0.143	0.443	3.248	8.047
T_{user}	0.337	1.418	12.352	31.280
T_{sys}	0.015	0.024	0.057	0.045
$\%CPU_{max}$	8%	24%	100%	100%
$n=8$ เหตุ				
$T_{mul,8}$	0.000	0.200	3.000	7.400
T_{real}	0.132	0.447	3.295	7.909
T_{user}	0.378	1.400	12.499	30.705
T_{sys}	0.013	0.020	0.037	0.045
$\%CPU_{max}$	9%	25%	100%	100%

7. เปลี่ยนจำนวนเหตุเด่ากับ $n=2$ เหตุ ด้วยคำสั่งต่อไปนี้

```
$ export OMP_NUM_THREADS=2
```

แล้ววนกลับไปทำข้อ 5 เพื่อกรอกค่าเฉลี่ยเวลาในตารางที่ M.1 จนครบ แล้วจึงเปลี่ยนจำนวนเหตุ $n=4$ และ 8 เหตุ

8. เปลี่ยนขนาดข้อมูล $N=400$ แล้วกลับไปเริ่มทำข้อ 3 จนถึงข้อ 8 จนครบ $N=800$ และ 1000

จากตารางที่ M.1 ผู้อ่านสามารถใช้ประกอบการคำนวณประสิทธิภาพการคำนวณแบบบานานในหัวข้อถัดไป

M.3 ความซับซ้อนของการคำนวณ (Complexity)

ผู้อ่านสามารถประยุกต์ใช้อัตราส่วนระหว่างความซับซ้อนเชิงเวลา (Run Time Complexity) $O(N_2)$ และ $O(N_1)$ ที่จำนวน n เทrod เมื่อกัน เพื่อวัดความซับซ้อนของอัลกอริทึมได้ดังสมการต่อไปนี้

$$\frac{O(N_2^3)}{O(N_1^3)} = \frac{T_{N_2,n}}{T_{N_1,n}} \quad (\text{M.2})$$

สำหรับการคูณแมทริกซ์ $T_{N,n}$ คือ $T_{mul,n}$ เป็นระยะเวลาเฉลี่ยของการคูณแมทริกซ์ขนาด $N \times N$ ด้วยจำนวน n เทrod จากหัวข้อที่ผ่านมา ผู้อ่านสามารถคำนวณค่าอัตราส่วนของเวลาในตารางที่ M.2 เพื่อใช้วิเคราะห์ต่อไป

ตารางที่ M.2: อัตราส่วนเวลาการคูณแมทริกซ์ที่ขนาด N และเวลาที่ขนาด $\sqrt[2]{\frac{400}{200}}$ ที่จำนวนเทrod เท่ากับ 1, 2, 4, 8 เทrod จากสมการที่ (M.2)

		$N=200$	$N=400$	$N=800$	$N=1000$
$\frac{11,200}{1.000}$	$\frac{29,600}{1.000}$	$n=1$ เทrod $T_{N,1}/T_{200,1}$	1.00	1.00	11.200
		$\sqrt[2]{T_{N,1}/T_{200,1}}$	1.00	1.00	3.347
		$\sqrt[3]{T_{N,1}/T_{200,1}}$	1.00	1.00	2.237
$\frac{5.800}{0.600}$	$\frac{15.000}{0.600}$	$n=2$ เทrod $T_{N,2}/T_{200,2}$	1.00	1.00	9.667
		$\sqrt[2]{T_{N,2}/T_{200,2}}$	1.00	1.00	3.109
		$\sqrt[3]{T_{N,2}/T_{200,2}}$	1.00	1.00	2.130
$\frac{3,000}{0.600}$	$\frac{7,800}{0.600}$	$n=4$ เทrod $T_{N,4}/T_{200,4}$	1.00	1.00	5
		$\sqrt[2]{T_{N,4}/T_{200,4}}$	1.00	1.00	2.236
		$\sqrt[3]{T_{N,4}/T_{200,4}}$	1.00	1.00	1.709
$\cancel{\frac{2,700}{3,000}}$	$\frac{7,400}{0.200}$	$n=8$ เทrod $T_{N,8}/T_{200,8}$	1.00	1.00	15
		$\sqrt[2]{T_{N,8}/T_{200,8}}$	1.00	1.00	3.873
		$\sqrt[3]{T_{N,8}/T_{200,8}}$	1.00	1.00	2.466

จะเปรียบเทียบค่าผลการคำนวณของ $\sqrt[2]{T_{N_2,n}/T_{200,1}}$ และ $\sqrt[3]{T_{N_2,n}/T_{200,1}}$ ที่ได้ในตารางที่ M.2 เมื่อ $N_2 = 400, 800$ และ 1000 และ $n = 1, 2, 4$ และ 8 ตามลำดับ ว่ามีค่าใกล้เคียงกับ $N_2/200 = 2, 4, 5$ อย่างไร เพราะเหตุใด

2, 2.5

M.4 ประสิทธิภาพ (Performance) ของการคำนวณแบบขنان

ผู้อ่านสามารถวัดประสิทธิภาพ (Performance) ของอัลกอริทึมใดๆ ได้จากอัตราส่วนของเวลาเดิม (T_{old}) และเวลาใหม่ (T_{new}) ที่ได้ทำการปรับปรุงอัลกอริทึมนั้นๆ ที่มา: [Patterson and Hennessy \(2016\)](#)

$$\frac{Perf_{new}}{Perf_{old}} = \frac{T_{old}}{T_{new}} \quad \begin{matrix} \text{เวลาต่ำ} \\ T_{old} > T_{new} \end{matrix} \quad (M.3)$$

ดังนั้น ประสิทธิภาพของการคำนวณแบบเบบบานสามารถวัดได้จากอัตราส่วนระหว่างระยะเวลา $T_{alg,1}$ ของ 1 เทrod และ $T_{alg,n}$ ของ n เทrod และตั้งชื่อเรียกว่า $Speedup(n)$ ด้วยสมการต่อไปนี้

$$Speedup(n) = \frac{T_{alg,1}}{T_{alg,n}} \quad (M.4)$$

โดย $T_{alg,n}$ คือ ช่วงการรันโปรแกรมอัลกอริทึมด้วยจำนวน n เทรด โดยไม่รวมช่วงเวลาอื่นๆ ซึ่งไม่ได้เกี่ยวข้องกับการอัลกอริทึมแบบขนาด ผู้อ่านสามารถประยุกต์ตัวชี้วัดนี้กับอัลกอริทึมการคุณแมทริกซ์ ดังนี้

$$Speedup(n) = \frac{T_{mul,1}}{T_{mul,n}} \quad (M.5)$$

โดย $T_{mul,n}$ คือ ช่วงการรันโปรแกรมคำนวณแมทริกซ์จริงๆ ด้วยจำนวน n เทrod ที่ขนาด N เท่ากันโดยไม่รวมช่วงเวลาสู่มค่าตั้งต้น และการแสดงผลอื่นๆ ผู้อ่านคำนวณค่า $Speedup(n)$ และกรอกในตารางที่ [M.3](#) เพื่อวิเคราะห์ผลการคำนวณที่ได้โดยตอบคำถามในกิจกรรมท้ายการทดลอง

ตารางที่ M.3: ผลการคำนวณ $Speedup(n)$ ของการคูณแมทริซที่ขนาด N และจำนวน-thread เท่ากับ 1, 2, 4, 8 thread จากสมการที่ [\(M.5\)](#)

Speedup	$N=200$	$N=400$	$N=800$	$N=1000$
$n=1$ เทรด $Speedup(1)$	1.00	1.00	1.00	1.00
$n=2$ เทรด $Speedup(2)$		1.67	1.93	1.97
$n=4$ เทรด $Speedup(4)$		1.67	3.73	3.79
$n=8$ เทรด $Speedup(8)$		5.00	3.73	4.00

จำนวนเทรด และ จำนวนซีพียูคอร์ มีผลต่อค่า Speedup อย่างไร วิเคราะห์ทั้งหมด 3 กรณีดังนี้

- จำนวนเต็ม < จำนวนซีพียคอร์

- จำนวนเทرد = จำนวนซีพียุคอร์

- จำนวนทรด > จำนวนซีพียูคอร์

M.5 กิจกรรมท้ายการทดลอง

เลือก 2 ข้อ

1. เหตุใดการทดลองจึงต้องใช้การหาค่าเฉลี่ยเวลาต่างๆ
2. T_{sys} หมายถึง เวลาซึ่งทำงานประเภทไหน
3. T_{user} หมายถึง เวลาซึ่งทำงานประเภทไหน
4. T_{real} มีความสัมพันธ์กับ T_{mul} อย่างไร
5. T_{user} มีความสัมพันธ์กับ T_{mul} และจำนวนเทรด n อย่างไร
6. เหตุใดค่าเฉลี่ยเวลา T_{user} จึงไม่แตกต่างกัน ที่ N คงที่
7. เวลาส่วนใหญ่ของการรัน T_{real} T_{user} และ T_{sys} สัมพันธ์กันอย่างไร จงสร้างสมการ
8. จำนวนเทรดที่เพิ่มขึ้นทำให้การคำนวณเร็วขึ้นอย่างไร มีข้อจำกัดหรือไม่
9. ที่ขนาดข้อมูล $N=1000$ จำนวนเทรดที่เพิ่มขึ้นทำให้ T_{user} เปลี่ยนแปลงอย่างไร มีข้อจำกัดหรือไม่
10. ที่ขนาดข้อมูล N ต่างๆ ค่า $\%CPU_{max}$ มีการเปลี่ยนแปลงและมีความสัมพันธ์กับจำนวนเทรด n อย่างไร
11. ขนาดข้อมูล N ที่เพิ่มขึ้นมีผลต่อ $Speedup(n)$ ที่ $n=1, 2, 4$ และ 8 หรือไม่ อย่างไร

