

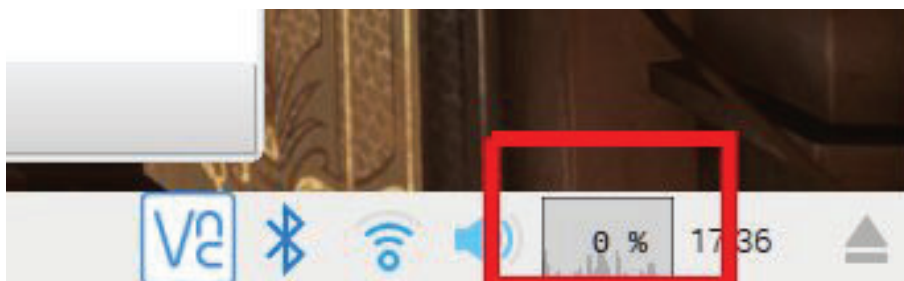
ภาคผนวก M

การทดลองที่ 13 การพัฒนาอัลกอริทึมแบบขนานด้วย OpenMP

การพัฒนาอัลกอริทึมแบบขนานบนซีพียูในปัจจุบันจำเป็นต้องอาศัยภาษาคอมพิวเตอร์ระดับสูง เช่น ภาษา C/C++ ภาษา Java เป็นต้น เพื่อช่วยลดเวลารัน (Run Time) ซึ่งเท่ากับเร่งความเร็ว (Speedup) ให้อัลกอริทึมหรือโปรแกรม โดยการสร้างเธรดผู้ช่วย (Worker Thread) และมอบหมายงานให้ไปรันบนซีพียูคอร์ที่ยังว่างอยู่ ผู้อ่านสามารถประยุกต์ใช้หลักการนี้บนเครื่องคอมพิวเตอร์ทั่วไปจนถึงเครื่องซูเปอร์คอมพิวเตอร์ตามเนื้อหาในบทที่ 8 ดังนั้น การทดลองมีวัตถุประสงค์ดังนี้

- เพื่อพัฒนาโปรแกรมภาษา C ด้วยไลบรารี OpenMP ให้สามารถทำงานแบบมัลติเธรดและใช้งานซีพียูมัลติคอร์ได้เต็มที่
- เพื่อเรียนรู้การวัด CPU Utilization (%CPU) เวลาจริง (T_{real}) เวลาผู้ใช้ (T_{user}) และเวลาระบบ (T_{sys}) ในซีพียูมัลติคอร์
- เพื่อทำความเข้าใจการวัดประสิทธิภาพของอัลกอริทึมแบบขนานด้านความซับซ้อนเชิงเวลาด้วยพีชคณิต BigO และตัวชี้วัด Speedup จากเวลาที่วัดได้

M.1 การวัด CPU Utilization



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

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

1. เลื่อนเมาส์ไปบนตำแหน่งว่างของ Task Bar
2. คลิกขวา เพื่อให้เมนูต่อไปนี้ปรากฏขึ้นแล้วคลิกซ้ายเลือก Add/Remove Panel Items
3. คลิกที่แท็บ Panel Applets
4. เลื่อนรายการขึ้นลงเพื่อหารายการชื่อ CPU Usage Monitor แล้วคลิก Add
5. กดปุ่ม Up และ Down เพื่อวางตำแหน่งของ CPU Usage Monitor ในตำแหน่งที่ต้องการ โปรดสังเกต รายชื่อ เมื่อได้ตำแหน่งที่ต้องการแล้วกด Close หมายเหตุ **Spacer** หมายถึง ช่องว่างที่คั่นระหว่าง Applet ที่อยู่บน Task Bar
6. สังเกตด้านขวาของ Task Bar จะมีจอสีเทาขนาดเล็กแสดงเป็นกราฟแท่ง โดยแท่งขวาสุดคือ วินาทีล่าสุด
7. เลื่อนเมาส์ไปบนกราฟแล้วคลิกขวาเพื่อเพิ่มการแสดงผลเป็นตัวเลขหน่วยเป็นเปอร์เซ็นต์ (%)
8. ทดสอบการทำงานโดยการเปิดคลิปเดียวกันบน [YouTube.com](https://www.youtube.com) ที่ความละเอียดแตกต่างกัน เช่น 240p, 480p และ 720p ทีละค่าเพื่อให้เห็นค่า $\%CPU_{max}$ ที่แตกต่างกัน

M.2 การคูณเมทริกซ์แบบขนาน

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

การคูณเมทริกซ์เป็นพื้นฐานของการคำนวณพื้นฐานทางวิทยาศาสตร์ และวิศวกรรมศาสตร์ กำหนดให้เมทริกซ์จัตุรัส A ขนาด $N \times N$ สามารถเขียนในรูปแบบของอะเรย์ 2 มิติในภาษา C/C++ ได้ดังนี้

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

โดยดัชนีตัวแรก i คือ หมายเลขแถว มีค่าตั้งแต่ 0 ถึง $N-1$ ดัชนีตัวที่สอง j คือ หมายเลขคอลัมน์ มีค่าตั้งแต่ 0 ถึง $N-1$ ดังนี้

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

เมื่อทำความเข้าใจพื้นฐานของเมทริกซ์ในรูปแบบของอะเรย์ 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>
#include <omp.h>

#define N 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
    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());
    c_1=time(NULL); // time measure: start time
    #pragma omp parallel for private(k, j)
        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];
                }
            }
        }
    c_2=time(NULL); // time measure: stop time
    t_mul = (float)(c_2-c_1); // Multiply time
```

```
printf("Mutiply Time: %f \n",t_1);

/* TERMINATE PROGRAM */
return 0;
}
```

3. exit ออกจากโปรแกรม nano เพื่อคอมไพล์โปรแกรมด้วยคำสั่งต่อไปนี้

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

แก้ไขหากมีข้อผิดพลาดจนกว่าจะคอมไพล์โปรแกรมสำเร็จและมีไฟล์ชื่อ mulMultrix

4. ตั้งค่าจำนวนเธรด $n=1$ ของโปรแกรมด้วยคำสั่งต่อไปนี้

```
$ export OMP_NUM_THREADS=1
```

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

```
$ time ./mulMatrix
```

Tmul = 1.000
Treal = 0.671
Tuser =0.661
Tsys = 0.021

ซึ่งจะรายงานผลการจับเวลาการทำงานของทั้งโปรแกรมในแง่มุมต่างๆ

6. จดบันทึกค่า CPU Utilization สูงสุดหรือ $\%CPU_{max}$ ที่สังเกตได้ หาค่าเฉลี่ยของ $T_{mul,n}$ T_{real} T_{user} และ T_{sys} ที่ได้ลงในตารางที่ [M.1](#)

ตารางที่ M.1: เวลาและ $\%CPU_{max}$ ของการคูณแมทริกซ์ที่ขนาด N และจำนวนเทรตเท่ากับ 1, 2, 4, 8 เทรต

เวลาเฉลี่ย	$N=200$ (วินาที)	$N=400$ (วินาที)	$N=800$ (วินาที)	$N=1000$ (วินาที)
$n=1$ เทรต				
$T_{mul,1}$	1.000000	5.0000	49.0000	99.0000
T_{real}	0.671	5.306	49.289	99.815
T_{user}	0.661	5.300	49.235	99.712
T_{sys}	0.021	0.001	0.022	0.041
$\%CPU_{max}$	10.7%	27.9%	35%	30.2%
$n=2$ เทรต				
$T_{mul,2}$	0.00000	2.0000	25.0000	50.000
T_{real}	0.371	2.739	24.936	50.526
T_{user}	0.697	5.301	48.972	100.449
T_{sys}	0.002	0.010	0.101	0.031
$\%CPU_{max}$	14.2%	28.8%	55.6%	54.8%
$n=4$ เทรต				
$T_{mul,4}$	0.0000	2.00000	13.000	27.000
T_{real}	0.281	1.427	13.697	27.439
T_{user}	0.704	5.145	49.558	100.458
T_{sys}	0.011	0.011	0.050	0.118
$\%CPU_{max}$	10.4%	45.1%	97.5%	100%
$n=8$ เทรต				
$T_{mul,8}$	0.0000	1.000	14.000	29.0000
T_{real}	0.257	1.529	14.671	28.900
T_{user}	0.719	5.124	52.792	105.917
T_{sys}	0.011	0.019	0.088	0.155
$\%CPU_{max}$	10.7%	47.6%	98.0%	100%

7. เปลี่ยนจำนวนเทรตเท่ากับ $n=2$ เทรต ด้วยคำสั่งต่อไปนี้

```
$ export OMP_NUM_THREADS=2
```

แล้ววนกลับไปทำข้อ 5 เพื่อรอกค่าเฉลี่ยเวลาในตารางที่ M.1 จนครบ แล้วจึงเปลี่ยนจำนวนเทรต $n=4$ และ 8 เทรต

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

จากตารางที่ M.1 ผู้อ่านสามารถใช้ประกอบการคำนวณประสิทธิภาพการคำนวณแบบขนานในหัวข้อถัดไป

M.3 ความซับซ้อนของการคำนวณ (Complexity)

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

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

สำหรับการคูณเมทริกซ์ $T_{N,n}$ คือ $T_{mul,n}$ เป็นระยะเวลาเฉลี่ยของการคูณเมทริกซ์ขนาด $N \times N$ ด้วยจำนวน n เทวดจากหัวข้อที่ผ่านมา ผู้อ่านสามารถคำนวณค่าอัตราส่วนของเวลาในตารางที่ M.2 เพื่อใช้วิเคราะห์ต่อไป

ตารางที่ M.2: อัตราส่วนเวลาการคูณเมทริกซ์ที่ขนาด N และเวลาที่ขนาด 200 ที่จำนวนเทวดเท่ากับ 1, 2, 4, 8 เทวด จากสมการที่ (M.2)

	$N=200$	$N=400$	$N=800$	$N=1000$
$n=1$ เทวด				
$T_{N,1}/T_{200,1}$	1.00	5.000	49.00	99.000
$\sqrt[2]{T_{N,1}/T_{200,1}}$	1.00	2.236	7.00	9.95
$\sqrt[3]{T_{N,1}/T_{200,1}}$	1.00	1.71	3.659	4.626
$n=2$ เทวด				
$T_{N,2}/T_{200,2}$	1.00	2.000	25.000	50.000
$\sqrt[2]{T_{N,2}/T_{200,2}}$	1.00	1.414	5.00	7.071
$\sqrt[3]{T_{N,2}/T_{200,2}}$	1.00	1.26	2.924	3.684
$n=4$ เทวด				
$T_{N,4}/T_{200,4}$	1.00	2.000	13.000	27.000
$\sqrt[2]{T_{N,4}/T_{200,4}}$	1.00	1.414	3.61	5.19
$\sqrt[3]{T_{N,4}/T_{200,4}}$	1.00	1.26	2.351	3
$n=8$ เทวด				
$T_{N,8}/T_{200,8}$	1.00	1.000	14.000	29.000
$\sqrt[2]{T_{N,8}/T_{200,8}}$	1.00	1.00	3.742	5.385
$\sqrt[3]{T_{N,8}/T_{200,8}}$	1.00	1.000	2.41	3.072

จงเปรียบเทียบค่าผลการคำนวณของ $\sqrt[2]{T_{N_2,n}/T_{200,1}}$ 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

เพราะ $N_2 = 400, 800, 1000$

$N_2/200$ จึงเท่ากับ 2, 4, 5

M.4 ประสิทธิภาพ (Performance) ของการคำนวณแบบขนาน

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

$$\frac{Perf_{new}}{Perf_{old}} = \frac{T_{old}}{T_{new}} \quad (M.3)$$

ดังนั้น ประสิทธิภาพของการคำนวณแบบขนานสามารถวัดได้จากอัตราส่วนระหว่างระยะเวลา $T_{alg,1}$ ของ 1 เทรดและ $T_{alg,n}$ ของ n เทรด 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 ซึ่งไม่ได้เกี่ยวข้องกับการอัลกอริทึมแบบขนาน ผู้อ่านสามารถประยุกต์ตัวชี้วัดนี้กับอัลกอริทึมการคูณเมทริกซ์ ดังนี้

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

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

ตารางที่ M.3: ผลการคำนวณ $Speedup(n)$ ของการคูณเมทริกซ์ที่ขนาด N และจำนวนเทรดเท่ากับ 1, 2, 4, 8 เทรด 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 $Speedup(2)$	1/0	2.5	1.96	1.98
$n=4$ เทรด $Speedup(4)$	1/0	2.5	3.77	3.67
$n=8$ เทรด $Speedup(8)$	1/0	5.0	3.5	3.41

จำนวนเทรด และ จำนวนซีพียูคอร์ มีผลต่อค่า Speedup อย่างไร วิเคราะห์ทั้งหมด 3 กรณีดังนี้

- จำนวนเทรด < จำนวนซีพียูคอร์
speedup น้อย
- จำนวนเทรด = จำนวนซีพียูคอร์
speedup ปานกลาง
- จำนวนเทรด > จำนวนซีพียูคอร์
speedup สูง

M.5 กิจกรรมท้ายการทดลอง

1. เหตุใดการทดลองจึงต้องใช้การหาค่าเฉลี่ยเวลาต่างๆ

เพื่อให้สามารถได้ค่าที่น่าเชื่อถือ เพราะถ้าวัดครั้งเดียวอาจจะมีปัจจัยอื่นที่ทำให้ค่าที่วัดได้ผิดพลาด

2. T_{sys} หมายถึง เวลาซีพียูทำงานประเภทไหน เวลาที่ซีพียูใช้ในการประมวลผลบน kernel space

3. T_{user} หมายถึง เวลาซีพียูทำงานประเภทไหน เวลาที่ซีพียูใช้ในการประมวลผลบน user space

4. T_{real} มีความสัมพันธ์กับ T_{mul} อย่างไร

5. T_{user} มีความสัมพันธ์กับ T_{mul} และจำนวนเทรต n อย่างไร

6. เหตุใดค่าเฉลี่ยเวลา T_{user} จึงไม่แตกต่างกัน ที่ N คงที่

7. เวลาส่วนใหญ่ของการรัน T_{real} T_{user} และ T_{sys} สัมพันธ์กันอย่างไร จงสร้างสมการ

8. จำนวนเทรตที่เพิ่มขึ้นทำให้การคำนวณเร็วขึ้นอย่างไร มีข้อจำกัดหรือไม่

9. ที่ขนาดข้อมูล $N=1000$ จำนวนเทรตที่เพิ่มขึ้นทำให้ T_{user} เปลี่ยนแปลงอย่างไร มีข้อจำกัดหรือไม่

10. ที่ขนาดข้อมูล N ต่างๆ ค่า $\%CPU_{max}$ มีการเปลี่ยนแปลงและมีความสัมพันธ์กับจำนวนเทรต n อย่างไร

11. ขนาดข้อมูล N ที่เพิ่มขึ้นมีผลต่อ $Speedup(n)$ ที่ $n=1, 2, 4$ และ 8 หรือไม่ อย่างไร