

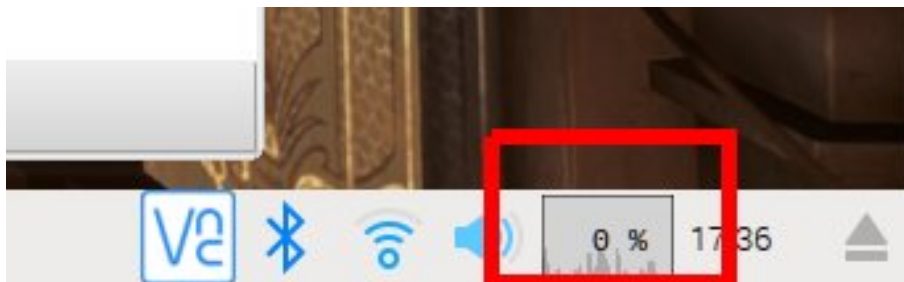
ภาคผนวก M

การทดลองที่ 13 การพัฒนาอัลกอริธึมแบบขนานด้วยไลบรารี OpenMP

การพัฒนาอัลกอริธึมแบบขนานบนซีพียูชนิดมัลติคอร์ในปัจจุบันจำเป็นต้องอาศัยภาษาคอมพิวเตอร์ระดับสูง เช่น ภาษา C/C++ ภาษา Java เป็นต้น เพื่อช่วยลดเวลารัน (Run Time) ซึ่งเท่ากับเร่งความเร็ว (Speedup) ให้ อัลกอริธึมหรือโปรแกรม โดยการสร้างเธรดผู้ช่วย (Worker Thread) และมอบหมายงานให้ไปรันบนแกนประมวลผลที่ยังว่างอยู่ ผู้อ่านสามารถประยุกต์ใช้หลักการนี้บนเครื่องคอมพิวเตอร์ทั่วไปจนถึงเครื่องซูเปอร์คอมพิวเตอร์ตามเนื้อหาในบทที่ 8 ดังนั้น การทดลองมีวัตถุประสงค์ดังนี้

- เพื่อพัฒนาโปรแกรมภาษา C ด้วยไลบรารี OpenMP ให้สามารถทำงานแบบมัลติเธรดและใช้งานซีพียูชนิดมัลติคอร์ได้เต็มที่
- เพื่อเรียนรู้การวัด CPU Utilization (%CPU) เวลาจริง (T_{real}) เวลาผู้ใช้ (T_{user}) และเวลาระบบ (T_{sys}) ในซีพียูชนิดมัลติคอร์
- เพื่อทำความเข้าใจการวัดประสิทธิภาพของอัลกอริธึมแบบขนานจากการประเมินความซับซ้อนเชิงเวลาด้วยพีชคณิต BigO ที่มา: [Rosen \(2002\)](#) และตัวชี้วัด Speedup ที่มา: [Patterson and Hennessy \(2016\)](#) จากเวลาที่วัดได้

M.1 การวัด CPU Utilization



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

ผู้อ่านสามารถติดตั้งเครื่องมือและกราฟในรูปที่ M.1 แสดงการใช้งานซีพียู (CPU Usage Monitor) ย้อนหลังและค่าสรุป ณ เวลาปัจจุบันของบอร์ด Pi 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 โปรดสังเกตรายชื่อ 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 ที่ละค่าเพื่อให้เห็นค่า $\%CPU_{max}$ ที่แตกต่าง

M.2 การคูณเมทริกซ์แบบขนาน

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

การคูณเมทริกซ์เป็นพื้นฐานของการคำนวณพื้นฐานทางวิทยาศาสตร์ และวิศวกรรมศาสตร์ กำหนดให้เมทริกซ์จัตุรัส A ขนาด $N \times N$ สามารถเขียนในรูปแบบของอาร์เรย์ 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 $N-1$ ดังนั้น

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

เมื่อทำความเข้าใจพื้นฐานของเมทริกซ์ในรูปแบบของอาร์เรย์ 2 มิติแล้ว ผู้อ่านสามารถทำการทดลองตามขั้นตอนต่อไปนี้

1. สร้างไดเรกทอรี `/home/pi/asm/Lab13` บนโปรแกรม Terminal ด้วยคำสั่งต่อไปนี้ตามลำดับ

```
$ cd /home/pi/asm
$ 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
double t_mul; // Multiply time
double start, stop; // 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();
        B[i][j]= rand();
    }
}

/* 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());
    start=omp_get_wtime(); // 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];
                }
            }
        }
}
```

```

    stop=omp_get_wtime(); // time measure: stop time
    t_mul = stop-start; // Multiply time
    printf("Mutiply Time: %2.4f \n",t_mul);

    /* TERMINATE PROGRAM */
    return 0;
}

```

3. exit ออกจากโปรแกรม nano เพื่อคอมไพล์โปรแกรมด้วยคำสั่งต่อไปนี้

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

แก้ไขหากมีข้อผิดพลาดจนกว่าจะคอมไพล์โปรแกรมสำเร็จและมีไฟล์ชื่อ mulMatrix

4. ตั้งค่าจำนวนเธรด $n=1$ ของโปรแกรมด้วยคำสั่งต่อไปนี้

```
$ export OMP_NUM_THREADS=1
```

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

```
$ time ./mulMatrix
```

ซึ่งจะรายงานผลการจับเวลาการทำงานของทั้งโปรแกรมในแง่มุมต่างๆ

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

ตารางที่ M.1: เวลาและ %CPU_{max} ของการคูณเมทริกซ์ที่ขนาด N และจำนวนเธรดเท่ากับ 1, 2, 4, 8 เธรด

เวลาเฉลี่ย (วินาที)	$N=200$ (วินาที)	$N=400$ (วินาที)	$N=800$ (วินาที)	$N=1000$ (วินาที)
$n=1$ เธรด				
$T_{mul,1}$	0.0349	0.3146	2.8271	3.8651
T_{real}	0.037	0.316	2.840	3.824
T_{user}	0.037	0.319	2.840	3.814
T_{sys}	0.000	0.000	0.000	0.010
%CPU _{max}	9%	23%	25%	25%
$n=2$ เธรด				
$T_{mul,2}$	0.0190	0.2033	1.3250	2.2626
T_{real}	0.021	0.214	1.338	2.251
T_{user}	0.031	0.406	2.663	4.519
T_{sys}	0.000	0.010	0.000	0.030
%CPU _{max}	9%	14%	35%	37%
$n=4$ เธรด				
$T_{mul,4}$	0.0113	0.1564	0.938	1.9639
T_{real}	0.014	0.191	0.976	1.383
T_{user}	0.027	0.406	3.709	5.070
T_{sys}	0.000	0.048	0.008	0.000
%CPU _{max}	9%	13%	52%	61%
$n=8$ เธรด				
$T_{mul,8}$	0.0090	0.0777	0.5748	1.0279
T_{real}	0.012	0.083	0.587	1.049
T_{user}	0.084	0.527	4.375	7.870
T_{sys}	0.000	0.000	0.010	0.000
%CPU _{max}	6%	15%	64%	82%

7. เปลี่ยนจำนวนเธรดเท่ากับ $n=2$ เธรด ด้วยคำสั่งต่อไปนี้

```
$ export OMP_NUM_THREADS=2
```

แล้ววนกลับไปทำข้อ 5 เพื่อรอกค่าเฉลี่ยเวลาในตารางที่ M.1 จนครบ แล้วจึงเปลี่ยนจำนวนเธรด $n=4$ และ 8 เธรด

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

จากตารางที่ M.1 ผู้อ่านสามารถใช้ประกอบการคำนวณประสิทธิภาพการคำนวณแบบขนานในหัวข้อถัดไป

M.3 ความซับซ้อน (Complexity) ของการคำนวณ

ความซับซ้อนเชิงเวลา (Run Time Complexity) ของการคูณเมทริกซ์เท่ากับ $O(N^3)$ ในทางทฤษฎี ผู้อ่านสามารถประยุกต์ใช้อัตราส่วนระหว่างระหว่าง $O(N_2^3)$ และ $O(N_1^3)$ ที่ภาระงานขนาด $N_2:N_1$ และจำนวน n เรดเหมือนกัน เพื่อวัดความซับซ้อนของอัลกอริธึมได้ดังสมการต่อไปนี้

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

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

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

	$N=200$	$N=400$	$N=800$	$N=1000$
$n=1$ เรด				
$T_{N,1}/T_{200,1}$	1.00	9.01	81.02	109.03
$\sqrt[2]{T_{N,1}/T_{200,1}}$	1.00	3.00	9.00	10.44
$\sqrt[3]{T_{N,1}/T_{200,1}}$	1.00	2.08	4.33	4.78
$n=2$ เรด				
$T_{mul,N}/T_{mul,200}$	1.00	11.02	69.74	119.08
$\sqrt[2]{T_{mul,N}/T_{mul,200}}$	1.00	3.32	8.35	10.91
$\sqrt[3]{T_{mul,N}/T_{mul,200}}$	1.00	2.23	4.12	4.92
$n=4$ เรด				
$T_{mul,N}/T_{mul,200}$	1.00	12.07	85.29	120.70
$\sqrt[2]{T_{mul,N}/T_{mul,200}}$	1.00	3.47	9.24	10.99
$\sqrt[3]{T_{mul,N}/T_{mul,200}}$	1.00	2.29	4.40	4.92
$n=8$ เรด				
$T_{mul,N}/T_{mul,200}$	1.00	8.63	63.97	114.21
$\sqrt[2]{T_{mul,N}/T_{mul,200}}$	1.00	2.94	7.99	10.69
$\sqrt[3]{T_{mul,N}/T_{mul,200}}$	1.00	2.05	4.00	4.85

M.4 ประสิทธิภาพ (Performance) ของการคำนวณแบบขนาน

ผู้อ่านสามารถวัดประสิทธิภาพ (Performance) ของอัลกอริธึมใดๆ ได้จากอัตราส่วนของเวลาเดิม (T_{old}) และเวลาใหม่ (T_{new}) ที่ได้ทำการปรับปรุงอัลกอริธึมนั้นๆ 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 เธรดและ $T_{alg,n}$ ของ n เธรด และตั้งชื่อเรียกว่า $Speedup(n)$ ด้วยสมการต่อไปนี้

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

โดย $T_{alg,n}$ คือ ช่วงการรันโปรแกรมอัลกอริธึมด้วยจำนวน n เธรด โดยไม่รวมช่วงเวลาอื่นๆ ซึ่งไม่ได้เกี่ยวข้องกับ การอัลกอริธึมแบบขนาน ผู้อ่านสามารถประยุกต์ตัวชี้วัดนี้กับอัลกอริธึมการคูณเมทริกซ์ ดังนี้

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

โดย $T_{mul,n}$ คือ ช่วงการรันโปรแกรมคำนวณเมทริกซ์จริงๆ ด้วยจำนวน n เธรด ที่ขนาด N เท่ากันโดยไม่รวมช่วง เวลาสุ่มค่าตั้งต้น และการแสดงผลอื่นๆ ผู้อ่านคำนวณค่า $Speedup(n)$ และกรอกในตารางที่ [M.3](#) เพื่อวิเคราะห์ ผลการคำนวณที่ได้โดยตอบคำถามในกิจกรรมท้ายการทดลอง

ตารางที่ M.3: ผลการคำนวณ $Speedup(n)$ ของการคูณเมทริกซ์ที่ขนาด N และจำนวนเธรดเท่ากับ 2, 4, 8 เธรด เทียบกับ 1 เธรดด้วยสมการที่ (M.5)

Speedup	$N=200$	$N=400$	$N=800$	$N=1000$
$n=2$ เธรด $Speedup(2) = T_{mul,1}/T_{mul,2}$	1.84	1.50	2.13	1.68
$n=4$ เธรด $Speedup(4) = T_{mul,1}/T_{mul,4}$	3.09	2.31	2.93	2.73
$n=8$ เธรด $Speedup(8) = T_{mul,1}/T_{mul,8}$	3.88	4.05	4.92	3.70

M.5 กิจกรรมท้ายการทดลอง

1. เหตุใดการทดลองจึงต้องใช้การหาค่าเฉลี่ยเวลาต่างๆ
2. T_{sys} หมายถึง เวลาซีพียูทำงานประเภทไหน
3. T_{user} หมายถึง เวลาซีพียูทำงานประเภทไหน
4. T_{real} มีความสัมพันธ์กับ T_{mul} อย่างไร 6) $\sqrt[3]{T_{mul, N_2}/T_{mul, 200}}$ บ่งชี้ถึง $\frac{N_2}{200}$ หรือ
5. T_{user} มีความสัมพันธ์กับ T_{mul} และจำนวนเธรด n อย่างไร | ใน N_2 ไม่ห้ฐาน ๒๒ N_1
6. เมื่อ $N_1=200$ จงเปรียบเทียบค่าผลการคำนวณของ $\sqrt[2]{T_{mul, N_2}/T_{mul, 200}}$ และ $\sqrt[3]{T_{mul, N_2}/T_{mul, 200}}$ ที่ได้ในตารางที่ M.2 เมื่อ $N_2= 400, 800$ และ 1000 และ $n= 1, 2, 4$ และ 8 เธรดตามลำดับ ว่ามีค่าใกล้เคียงกับ $N_2/200= 2, 4, 5$ ตามลำดับอย่างไร เพราะเหตุใด
7. จำนวนเธรด และ จำนวนแกนประมวลผล มีผลต่อค่า Speedup อย่างไร วิเคราะห์ทั้งหมด 3 กรณีดังนี้
 - จำนวนเธรด < จำนวนแกนประมวลผล ถ้า Speedup < 1
 - จำนวนเธรด = จำนวนแกนประมวลผล ถ้า Speedup ใกล้กับ 1 / 1 เท่ากับ 1
 - จำนวนเธรด > จำนวนแกนประมวลผล ถ้า Speedup > 1
8. เหตุใดค่าเฉลี่ยเวลา T_{user} จึงไม่แตกต่างกัน ที่ N คงที่
9. เวลาส่วนใหญ่ของการรัน T_{real} T_{user} และ T_{sys} สัมพันธ์กันอย่างไร จงสร้างสมการ
10. จำนวนเธรดที่เพิ่มขึ้นทำให้การคำนวณเร็วขึ้นอย่างไร มีข้อจำกัดหรือไม่
11. ที่ขนาดข้อมูล $N=1000$ จำนวนเธรดที่เพิ่มขึ้นทำให้ T_{user} เปลี่ยนแปลงอย่างไร มีข้อจำกัดหรือไม่
12. ที่ขนาดข้อมูล N ต่างๆ ค่า $\%CPU_{max}$ มีการเปลี่ยนแปลงและมีความสัมพันธ์กับจำนวนเธรด n อย่างไร
13. ขนาดข้อมูล N ที่เพิ่มขึ้นมีผลต่อ $Speedup(n)$ ที่ $n=1, 2, 4$ และ 8 หรือไม่ อย่างไร

Cpu

Processor

Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.21 GHz