

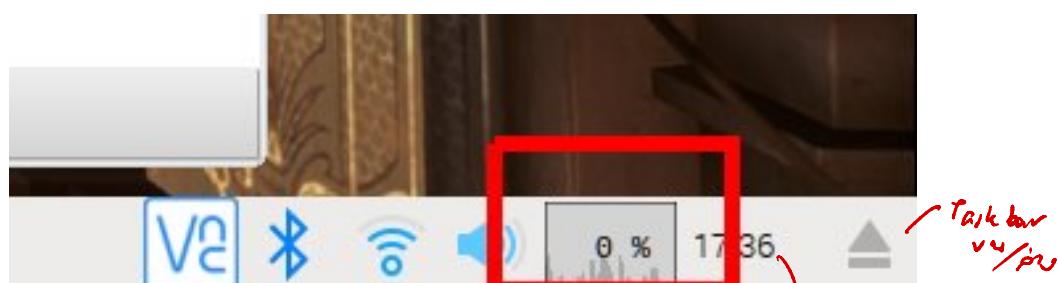
ภาคผนวก M

การทดลองที่ 13 การพัฒนาอัลกอริทึมแบบขัน ด้วย OpenMP

การพัฒนาอัลกอริทึมแบบขันชนชีพิญในปัจจุบันจำเป็นต้องอาศัยภาษาคอมพิวเตอร์ระดับสูง เช่น ภาษา C/C++/ภาษา Java เป็นต้น เพื่อช่วยลดเวลา_ran (Run Time) ซึ่งเท่ากับเร่งความเร็ว (Speedup) ให้อัลกอริทึมหรือโปรแกรม โดยการสร้างthreadผู้ช่วย (Worker Thread) และมอบหมายงานให้ไปรันบนชีพิญคอร์ที่ยังว่างอยู่ ผู้อ่านสามารถประยุกต์ใช้หลักการนี้บนเครื่องคอมพิวเตอร์ทั่วไปจนถึงเครื่องชูเบอร์คอมพิวเตอร์ตามเนื้อหาในบทที่ 8 ดังนั้น การทดลองมีวัตถุประสงค์ดังนี้

- เพื่อพัฒนาโปรแกรมภาษา C ด้วยไลบรารี OpenMP ให้สามารถทำงานแบบมัลติ-thread และใช้งานชีพิญ มัลติคอร์ได้เต็มที่
- เพื่อเรียนรู้การวัด CPU Utilization (%CPU) ในชีพิญมัลติคอร์ core num 8.
- เพื่อทำความเข้าใจการวัดประสิทธิภาพของอัลกอริทึมแบบขันด้านความซับซ้อนเชิงเวลาด้วยพีซคณิต BigO และตัวชี้วัด Speedup จากเวลาที่วัดได้

M.1 การวัด CPU Utilization



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

ผู้อ่านสามารถติดตั้งเครื่องมือและการกราฟแสดงการใช้งานชีพิญ (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 เพื่อวางแผนที่ต้องการ โปรดสังเกตรายชื่อ เมื่อได้ตำแหน่งที่ต้องการแล้วกด Close หมายเหตุ Spacer หมายถึง ช่องว่างที่คั่นระหว่าง Application ที่อยู่บน Task Bar ✓
 6. สังเกตด้านขวาของ Task Bar จะมีจอสีเทาขนาดเล็กแสดงเป็นกราฟแท่ง โดยแท่งขวาสุดคือ วินาทีล่าสุด ✓
 7. เลื่อนมาส์ปีบันกราฟแล้วคลิกขวาเพื่อเพิ่มการแสดงผลเป็นตัวเลขหน่วยเป็นเปอร์เซ็นต์ (%)
 8. ทดสอบการทำงานโดยการเปิดคลิปเดียวกันบน YouTube.com ที่ความละเอียดแตกต่างกัน เช่น 240p, 480p และ 720p ทีละค่าเพื่อให้เห็นค่า $\%CPU_{max}$ ที่แตกต่าง
- ↑ พื้นที่บันทึก
↓ จัดเรียง
- ↑ พื้นที่บันทึก
↓ จัดเรียง

M.2 การคูณแมทริกซ์แบบขบวน

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

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

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

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

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

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>
#include <omp.h>

#define N 200      // 200x200
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 set start.

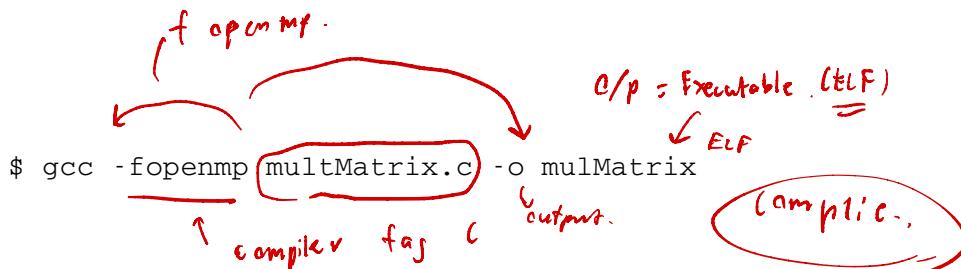
    #pragma omp parallel for private(k, j)
    for(i=0;i<N;i++) {
        for(j=0;j<N;j++) {
            C[i][j]=0.; // set initial value of resulting matrix C = 0
            for(k=0;k<N;k++) {
                C[i][j]=C[i][j]+A[i][k]*B[k][j];
            }
        }
    }

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

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

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

3. exit ออกจากโปรแกรม nano เพื่อคอมไพล์โปรแกรมด้วยคำสั่งต่อไปนี้



แก้ไขหากมีข้อผิดพลาดจนกว่าจะคอมไпал์ล์โปรแกรมสำเร็จและมีไฟล์ชื่อ mulMatrix

4. ตั้งค่าจำนวน-thread $n=1$ ของโปรแกรมด้วยคำสั่งต่อไปนี้

Terminal

```
$ export OMP_NUM_THREADS=1
```

↓ stat.

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

รัน / Exec.

```
$ time ./mulMatrix
```

ซึ่งจะรายงานผลการจับเวลาการทำงานของทั้งโปรแกรมในแต่ละมุ่งต่างๆ

กับเวลารัน.

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	5.000	5.000	5.000
T_{real}	0.616	5.067	5.040	5.020
T_{user}	0.603	5.038	4.997	4.998
T_{sys}	0.010	0.010	0.030	0.010
$\%CPU_{max}$	48%	96%	26%	26%
$n=2$ เหตุ				
$T_{mul,2}$	0.000	2.000	3.000	3.000
T_{real}	0.343	2.899	2.574	2.549
T_{user}	0.649	5.057	5.056	5.014
T_{sys}	0.010	0.020	0.011	0.010
$\%CPU_{max}$	48%	50%	51%	50%
$n=4$ เหตุ				
$T_{mul,4}$	0.000	3.000	1.000	2.000
T_{real}	0.261	6.660	1.357	1.397
T_{user}	0.936	5.193	5.040	5.055
T_{sys}	0.022	0.020	0.020	0.032
$\%CPU_{max}$	43%	58%	76%	76%
$n=8$ เหตุ				
$T_{mul,8}$	1.000	1.000	1.000	1.000
T_{real}	0.238	1.351	1.399	1.385
T_{user}	0.707	4.951	4.990	4.944
T_{sys}	0.012	0.020	0.011	0.040
$\%CPU_{max}$	43%	68%	81%	86%

ปัจจัยหนึ่งของความ



#

7. เปลี่ยนจำนวนเหตุเด่ากับ $n=2$ เหตุ ด้วยคำสั่งต่อไปนี้

\$ export OMP_NUM_THREADS=2 4 110 8 .

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

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

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

ตารางที่ M.2: อัตราส่วนเวลาการคุณแม่ทริช์ที่ขนาด N และเวลาที่ขนาด 200 ที่จำนวนเหตุเท่ากับ 1, 2, 4, 8 เหตุ จากสมการที่ (M.2) $\frac{N}{200} \neq N_{1,5205} \quad N_{1,400}$

	$N=200$	$N=400$	$N=800$	$N=1000$	$\left(\frac{T_{800,1}}{T_{200,1}} \right)$
$n=1$ เทรด					$1.01 \ln 4.00 = 5.00$
$T_{N,1}/T_{200,1}$ 0.666	1.00	1	1	1	
$\sqrt[2]{T_{N,1}/T_{200,1}}$	1.00	1	1	1	
$\sqrt[3]{T_{N,1}/T_{200,1}}$	1.00	1	1	1	
$n=2$ เทรด					
$T_{N,2}/T_{200,2}$ 0.343	1.00	N/A	1.5	1.5	
$\sqrt[2]{T_{N,2}/T_{200,2}}$	1.00	1	1.22	1.22	
$\sqrt[3]{T_{N,2}/T_{200,2}}$	1.00	1	1.14	1.14	
$n=4$ เทรด					
$T_{N,4}/T_{200,4}$ 0.261	N/A	1	1	2	
$\sqrt[2]{T_{N,4}/T_{200,4}}$	1.00	1	1	1.41	
$\sqrt[3]{T_{N,4}/T_{200,4}}$	1.00	1	1	1.259	
$n=8$ เทรด					
$T_{N,8}/T_{200,8}$ 0.238	1.00	1	1	1	
$\sqrt[2]{T_{N,8}/T_{200,8}}$	1.00	1	1	1	
$\sqrt[3]{T_{N,8}/T_{200,8}}$	1.00	1	1	1	

จะเบรียบเทียบค่าผลการคำนวณของ $\sqrt[2]{T_{N_2,n}/T_{200,1}}$ 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 $N_2/200 = 2, 4, 8$ อย่างไร เพราะเหตุใด

ခြေခံမှုနှင့် အသုတေသနများ ပေါ်စွာရှိလေ့လာတဲ့ အကြောင်းအရာများ အတွက် အမြတ်ဆုံး အသုတေသနများ ဖြစ်ပေါ်လေ့လာတယ်

M.4 ประสิทธิภาพ (Performance) ของการคำนวณแบบขนาน

ผู้อ่านสามารถวัดประสิทธิภาพ (Performance) ของอัลกอริทึมได้ ได้จากการอัตราส่วนของเวลาเดิม (T_{old}) และเวลาใหม่ (T_{new}) ที่ได้ทำการปรับปรุงอัลกอริทึมนั้นๆ ที่มา: Patterson and Hennessy (2016)

$$\frac{Perf_{new}}{Perf_{old}} = \left(\frac{T_{old}}{T_{new}} \right)^{\frac{1}{n+1}} \quad \text{--- กรณี } T_{old} > T_{new} \quad (\text{M.3})$$

ดังนั้น ประสิทธิภาพของการคำนวณแบบขนานสามารถวัดได้จากการอัตราส่วนระหว่างระยะเวลา $T_{alg,1}$ ของ 1 เทρดและ $T_{alg,n}$ ของ n เทρด และตั้งชื่อเรียกว่า $Speedup(n)$ ด้วยสมการต่อไปนี้

$$Speedup(n) = \frac{T_{alg,1}}{T_{alg,n}} \quad \text{--- } \# \text{ เทρด. } \leftarrow \text{Cmp - Set - N - } \frac{n}{n+1} \quad (\text{M.4})$$

โดย $T_{alg,n}$ คือ ช่วงการรันโปรแกรมอัลกอริทึมด้วยจำนวน n เทρด โดยไม่รวมช่วงเวลาอื่นๆ ซึ่งไม่ได้เกี่ยวข้อง กับการอัลกอริทึมแบบขนาน ผู้อ่านสามารถประยุกต์ตัวชี้วัดนี้กับอัลกอริทึมการคูณแม่ทริกซ์ ดังนี้

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

โดย $T_{mul,n}$ คือ ช่วงการรันโปรแกรมคำนวณแม่ทริกซ์จริงๆ ด้วยจำนวน n เทρด ที่ขนาด N เท่ากันโดยไม่รวมช่วงเวลาสุ่มค่าตั้งต้น และการแสดงผลอื่นๆ ผู้อ่านคำนวณค่า $Speedup(n)$ และกรอกในตารางที่ M.3 เพื่อวิเคราะห์ผลการคำนวณที่ได้โดยตอบคำถามในกิจกรรมท้ายการทดลอง

ตารางที่ M.3: ผลการคำนวณ $Speedup(n)$ ของการคูณแม่ทริกซ์ที่ขนาด N และจำนวนเทρดเท่ากับ 1, 2, 4, 8 เทρด จากสมการที่ (M.5)

Speedup	$N=200$	$N=400$	$N=800$	$N=1000$
$n=1$ เทρด $Speedup(1)$	1.00	1.00	1.00	1.00
$n=2$ เทρด $Speedup(2)$	0.5	0.5	0.5	0.5
$n=4$ เทρด $Speedup(4)$	0.25	0.25	0.25	0.25
$n=8$ เทρด $Speedup(8)$	0.125	0.125	0.125	0.125

จำนวนเทρด และ จำนวนซีพียูคอร์ มีผลต่อค่า Speedup อย่างไร วิเคราะห์ทั้งหมด 3 กรณีดังนี้

- จำนวนเทρด < จำนวนซีพียูคอร์ $\frac{1}{2} < \frac{4}{4}$ $\leftarrow 4 \text{ core } \frac{1}{2} \approx 0.5 \text{ sec.}$

speedup เมื่อมีหน่วยเวลา thread.

- จำนวนเทρด = จำนวนซีพียูคอร์ $\frac{4}{4} = \frac{4}{4}$

speedup ประมาณ 1.5 sec.

- จำนวนเทρด > จำนวนซีพียูคอร์ $\frac{8}{4} > \frac{4}{4}$

speedup ประมาณ 1.2 sec.

M.5 กิจกรรมท้ายการทดลอง

1. เหตุใดการทดลองจึงต้องใช้การหาค่าเฉลี่ยเวลาต่างๆ เมื่อในแต่ละห้องทำงานของมาสเตอร์ เรื่องนี้จะวัดเวลาที่ต่างกัน
2. T_{sys} หมายถึง เวลาซึ่งพิสูจน์ทำงานประเภทไหน Time system by OS. (วินโดวส์) จัดการห้องเรียน
3. T_{user} หมายถึง เวลาซึ่งพิสูจน์ทำงานประเภทไหน Time user by OS. (วินโดวส์) ใช้เวลาจริง
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 หากมีthreadมากกว่าหนึ่งตัว แต่ก็ต้องแบ่งงานให้กับthreadทุกตัว
9. ที่ขนาดข้อมูล $N=1000$ จำนวนเทรดที่เพิ่มขึ้นทำให้ T_{user} เปลี่ยนแปลงอย่างไร มีข้อจำกัดหรือไม่ ข้อมูล
10. ที่ขนาดข้อมูล N ต่างๆ ค่า $\%CPU_{max}$ มีการเปลี่ยนแปลงและมีความสัมพันธ์กับจำนวนเทรด n อย่างไร %CPU_max = \frac{n}{N} * CPU_{max} ต่อหน่วย (ตัวอย่าง)
11. ขนาดข้อมูล N ที่เพิ่มขึ้นมีผลต่อ $Speedup(n)$ ที่ $n=1, 2, 4$ และ 8 หรือไม่ อย่างไร %CPU_max = \frac{1}{n} * CPU_{max} ต่อหน่วย (ตัวอย่าง)