รายงานสรุปผล Assignment
Case Study #1

สมาชิก

62010052	กันต์	มากทรัพย์สิน
62010453	นนทพันธุ์	รุจิรกาล
62010472	นวพรรษ	ศรีบุญเรือง
62010474	นวพล	กรุดพันธ์
62010494	นิติพัฒน์	บุญเกตุ
62010496	นิติภูมิ	คล้ายเนียม

1. Source Code ของ Version ที่ นศ คิดว่าเป็น Version ที่ดีที่สุด

```
2 references
static void sum(int from, int to) {
    long in_sum = 0;
    fixed(byte* data_global = Data_Global)
    for (int i = from ; i < to ; i++) {
        byte num = data_global[i];
        data_global[i] = 0;
        if (num % 2 == 0) {
            in_sum -= num;
        }
        else if (num % 3 == 0) {
            in_sum += num * 2;
        }
        else if (num % 5 == 0) {
            in_sum += num / 2;
        }
        else if (num % 7 == 0) {
            in_sum += num / 3;
        }
    }
    Sum_Global += in_sum;
}

1 reference
static bool isThreadWorking(Thread[] threads) {
    bool result = false;
    for (int i = 0 ; i < threads.Length ; i++) {
        if (threads[i].IsAlive) {
            result = true;
            break;
        }
    }
    return result;
}
```

```
0 references
static void Main(string[] args) {
    Stopwatch sw = new Stopwatch();
    int y;
    /* Read data from file */
    Console.Write("Data read...");
    y = ReadData();
    if (y == 0) {
        Console.WriteLine("Complete.");
    }
    else {
        Console.WriteLine("Read Failed!");
    }

    /* Create Thread */
    Console.WriteLine("Processer Count = " + processorCount);
    int mainTo = (int)((1 / (float)(processorCount)) * 1000000000);
    Console.WriteLine("Main Thread process from " + 0 + " to " + mainTo);

    Thread[] threads = new Thread[processorCount - 1];
    for (int i = 0 ; i < processorCount - 1 ; i++) {
        int from = (int)(((float)(i + 1) / (float)(processorCount)) * 1000000000);
        int to = (int)(((float)(i + 2) / (float)(processorCount)) * 1000000000);
        Console.WriteLine("Thread " + (i + 1) + " process from " + from + " to " + to);
        threads[i] = new Thread(() => sum(from, to));
    }

    /* Start */
    Console.Write("\n\nWorking...");
    // long initial_time = DateTimeOffset.Now.ToUnixTimeMilliseconds();

    sw.Start();

    for (int i = 0 ; i < threads.Length ; i++) {
        threads[i].Start();
    }
    sum(0, mainTo);

    while(isThreadWorking(threads)) {}

    sw.Stop();
    Console.WriteLine("Done.");
    // long used_time = DateTimeOffset.Now.ToUnixTimeMilliseconds() - initial_time;
    /* Result */
    Console.WriteLine("Summation result: {0}", Sum_Global);
    Console.WriteLine("Time used: " + sw.ElapsedMilliseconds.ToString() + "ms");

}
```

Source Code :
https://github.com/NitipoomKlaynium/Operating-System/tree/main/Problem01

2. **สมมติฐาน**
    - การเพิ่ม Thread ในการทำงาน ทำให้การทำงานเร็วขึ้นอย่างแน่นอน
    - Hardware ส่งผลต่อการทำงาน
    - การปรับปรุง Algorithm จะทำงานได้เร็วขึ้น

Version 1

```
static unsafe void sum()
{
    fixed(byte* data_list = Data_Global)
    for (int i = 0 ; i < 1000000000; i++) {
        byte num = data_list[G_index];
        data_list[G_index] = 0;
        if (num % 2 == 0)
        {
            Sum_Global -= num;
        }
        else if (num % 3 == 0)
        {
            Sum_Global += (num * 2);
        }
        else if (num % 5 == 0)
        {
            Sum_Global += (num / 2);
        }
        else if (num % 7 == 0)
        {
            Sum_Global += (num / 3);
        }
        G_index++;
    }
}
```

แนวคิด

นำ loop มาไว้ใน function sum() แล้วใช้ตัวแปร pointer ในการเก็บตัวแปร Data_Global แล้วสร้างตัวแปรมาใช้เก็บค่า element ใน array เพื่อไม่ต้องเรียกใช้ index หลายรอบ ช่วยลดความเร็วในการทำงานจาก 33 วินาที => 20 วินาที

ข้อสังเกตพบ

- การปรับปรุงแก้ไข Algorithm จะทำให้งานเร็วขึ้น สมมติฐานข้อที่ 3 เป็นจริง

ปัญหา

- ความเร็วที่เพิ่มขึ้นแต่ละเครื่องคอมพิวเตอร์ไม่เท่ากัน

วิธีแก้

- ไม่ทราบแน่ชัด

Version 2 (ยังไม่ได้นำ Version 1 มารวมเนื่องจากยังไม่ทราบแน่ชัดในการแก้ปัญหา ทดสอบเฉพาะ Thread ก่อน)

แนวคิด

การแบ่ง process การทำงานให้กับ Thread จำนวน 2 Thread ทำงานเป็นแบบ parallel แทนที่เราจะ register thread เพียงตัวเดียว

```
static void sum(int start,int stop)
{
    long sum_local = 0;
    for (int i = start; i < stop; i += 2){
        int num = Data_Global[i];
        Data_Global[i] = 0;
        if (num % 2 == 0) {
            sum_local -= num;
        }
        else if (num % 3 == 0) {
            sum_local += num * 2;
        }
        else if (num % 5 == 0) {
            sum_local += num / 2;
        }
        else if (num % 7 == 0) {
            sum_local += num / 3;
        }
    }
    Sum_Global += sum_local;
}
```

Thread ตัวแรกสร้างเพื่อที่จะ process การทำงานของ sum function ในการทำงานเป็น loop เลขคู่

```
Thread odd_ = new Thread(() => sum(0,1000000000));
```

Thread ตัวที่สองสร้างเพื่อที่จะ process การทำงานของ sum function ในการทำงานเป็น loop เลขคี่

```
Thread even_ = new Thread(() => sum(1,1000000000));
```

```
sw.Start();
odd_.Start();
even_.Start();
odd_.Join();
even_.Join();
```

การทำงานจะทำงานโดยให้ Thread ทั้ง 2 เริ่ม แล้ว รอ Thread ทั้ง 2 จบ(ผ่าน Method Join ของ Thread)ถึงจะหยุดจับเวลา

ข้อสังเกตพบ

- การเพิ่ม Thread ทำให้การทำงานเร็วขึ้นตามสมมติฐานข้อที่ 1(0->2 เทรด)
- ความเร็วเริ่มต้นจาก 32 วินาที -> 12.4 วินาที

ปัญหา

- ถ้าเพิ่มเทรดขึ้นไปอีกจะเร็วกว่าเดิมมากแค่ไหน ?

วิธีแก้

- ทดลองเพิ่มเทรด ใน Version ที่ 3

Version 3

แนวคิด

แบ่ง Scope การทำงาน แต่ Loop ยังคงเหมือน Version 2

```
Thread odd_1 = new Thread(() => sum(1,500000000));
Thread even_1 = new Thread(() => sum(0,500000000));
Thread odd_2 = new Thread(() => sum(500000001,1000000000));
Thread even_2 = new Thread(() => sum(500000000,1000000000));
```

เป็นพัฒนาต่อยอดมาจาก Ver 1. และ Ver 2. โดยมีแนวคิดคือ “การลดภาระงานลงครึ่งหนึ่ง แต่ยังคงแบ่งหน้าที่ process ที่ฟังชันก์ คู่/คี่ดังเดิม”

โดยการแบ่ง process การทำงานให้กับ Thread จำนวน 4 Thread ทำงานเป็นแบบ parallel

โดยที่ Thread odd_1,odd_2 จะ process การทำงานของ sum function ในการทำงานเป็น loop เลขคี่ดังเดิม แต่จะลดภาระการ process ประมาณครึ่งหนึ่ง โดยการที่หมอบหมายงานให้ถึงเพียง 50000000

Thread even_1,even_2 จะ process การทำงานของ sum function ในการทำงานเป็น loop เลขคี่ดังเดิม แต่จะลดภาระการ process ประมาณครึ่งหนึ่ง โดยการที่หมอบหมายงานให้ถึงเพียง 50000000

```
odd_1.Start();
even_1.Start();
odd_2.Start();
even_2.Start();
odd_1.Join();
even_1.Join();
odd_2.Join();
even_2.Join();
```

การทำงานจะทำงานโดยให้ Thread ทั้ง 4เริ่ม แล้ว รอ Thread ทั้ง 4 จบ(ผ่าน Method Join ของ Thread) ถึงจะหยุดจับเวลา

ข้อสังเกตพบ

- การเพิ่ม Thread ทำให้การทำงานเร็วขึ้นตามสมมติฐานข้อที่ 1
- เร็วขึ้นกว่า Version 2 จาก 12.4 วินาที -> 6.795 วินาที
- ใช้เทรดมากขึ้น จาก 2 -> 4 เทรด

ปัญหา

- ถ้าเพิ่มเทรดขึ้นไปอีกจะเร็วกว่าเดิมมากแค่ไหน ? (เหมือน Version 2 เพื่อประสิทธิภาพสูงสุด)

วิธีแก้

- ทดลองเพิ่มเทรด ใน Version ที่ 4

Version 4

แนวคิด

การใช้แนวคิดเดิมเหมือนกับ Version ที่ 3 แต่เปลี่ยนเป็นการเพิ่มเทรดการในทำงานให้มากขึ้น

```
Thread odd_1 = new Thread(() => sum(1,400000000));
Thread even_1 = new Thread(() => sum(0,400000000));
Thread odd_2 = new Thread(() => sum(400000001,700000000));
Thread even_2 = new Thread(() => sum(400000000,700000000));
Thread odd_3 = new Thread(() => sum(700000001,1000000000));
Thread even_3 = new Thread(() => sum(700000000,1000000000));
```

```
sw.Start();
odd_1.Start();
even_1.Start();
odd_2.Start();
even_2.Start();
odd_3.Start();
even_3.Start();
odd_1.Join();
even_1.Join();
odd_2.Join();
even_2.Join();
odd_3.Join();
even_3.Join();
sw.Stop();
```

ข้อสังเกตพบ

- การเพิ่ม Thread การทำงาน ทำให้ผลลัพธ์เร็วขึ้นกว่าเดิมเล็กน้อยซึ่งไม่เท่ากับที่ Version 3 เพิ่ม จาก Version 2 เป็นเท่าตัว
- เพิ่มการทำงานจาก 4 Thread เป็น 6 Thread
- การเพิ่ม Thread จำนวนเยอะๆ จะไม่ทำให้ความเร็วเพิ่มขึ้นเสมอไปในจุดๆนึง

ปัญหา

- ต้องการที่เรียกใช้ Thread จำนวนมากที่สุดในแต่ละเครื่องคอมพิวเตอร์ที่ต่างกันเพื่อแสดงประสิทธิภาพของการทำงานเทรด และสังเกต Hardware ที่ต่างกันด้วย

วิธีแก้

- ทดลองเพิ่มเทรดใน Version ที่ 5

Version 5 (นำVersion 1 มาประกอบด้วยเพื่อเพิ่มประสิทธิภาพให้เร็วมากที่สุด เนื่องจาก สนใจที่ประสิทธิภาพเท่านั้น)

แนวคิด

คอมพิวเตอร์แต่ละเครื่อง มีจำนวน Thread ที่ต่างกัน ดังนั้น จึงมีแนวคิดที่จะสร้าง Thread ตามจำนวนทั้งหมดที่ Hardware สามารถทำได้ เพื่อให้คอมพิวเตอร์แต่ละเครื่องสามารถใช้ Thread ที่มีได้เต็มประสิทธิภาพ

```
static readonly int processorCount = System.Environment.ProcessorCount;
```

ปรับปรุงการเรียกใช้ Thread

- ประกาศตัวแปร Thread เป็นแบบ Array และประกาศขนาดของ Array ตามจำนวน Thread ที่คอมพิวเตอร์แล้ว - 1 ซึ่ง เพราะที่ main ถือเป็น thread อีก thread หนึ่ง
- แบ่งให้ทุก Thread ทำงานในแต่ละช่วงของ Array ในจำนวนที่เท่ากัน

```
int mainTo = (int)((1 / (float)(processorCount)) * 1000000000);
Console.WriteLine("Main Thread process from " + 0 + " to " + mainTo);

Thread[] threads = new Thread[processorCount - 1];
for (int i = 0 ; i < processorCount - 1 ; i++) {
    int from = (int)(((float)(i + 1) / (float)(processorCount)) * 1000000000);
    int to = (int)(((float)(i + 2) / (float)(processorCount)) * 1000000000);
    Console.WriteLine("Thread " + (i + 1) + " process from " + from + " to " + to);
    threads[i] = new Thread(() => sum(from, to));
}
```

- สร้าง function ที่ใช้ตรวจสอบว่ายังมี Thread ใดทำงานอยู่อีกหรือเปล่าเพื่อใช้ในการรอให้ Thread ทุก Thread ทำงานเสร็จแล้วจึงค่อยทำการแสดงผล

```
static bool isThreadWorking(Thread[] threads) {
    bool result = false;
    for (int i = 0 ; i < threads.Length ; i++) {
        if (threads[i].IsAlive) {
            result = true;
            break;
        }
    }
    return result;
}
```

- Function sum จะมี parameter ที่จะเป็นตัวกำหนดว่าให้ทำงานในช่วง index ที่เท่าไหร่ ถึงเท่าไหร่เพื่อให้กำหนดช่วงที่ทำงานได้อย่างอิสระ

- ทำการเพิ่ม Index ทีละ 1 เพื่อความสะดวกในการเรียกใช้งาน

```
static void sum(int from, int to) {
    long in_sum = 0;
    fixed(byte* data_global = Data_Global)
    for (int i = from ; i < to ; i++) {
        byte num = data_global[i];
        data_global[i] = 0;
        if (num % 2 == 0) {
            in_sum -= num;
        }
        else if (num % 3 == 0) {
            in_sum += num * 2;
        }
        else if (num % 5 == 0) {
            in_sum += num / 2;
        }
        else if (num % 7 == 0) {
            in_sum += num / 3;
        }
    }
    Sum_Global += in_sum;
}
```

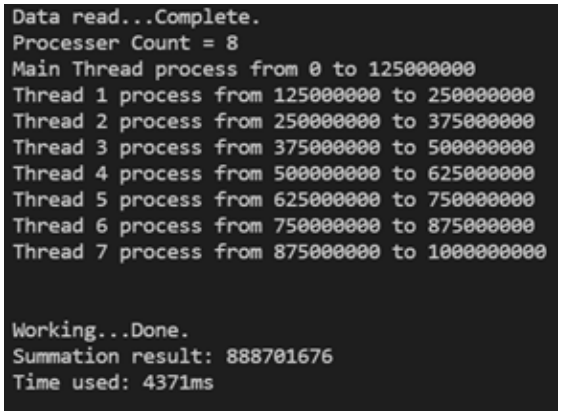
```
sw.Start();

for (int i = 0 ; i < threads.Length ; i++) {
    threads[i].Start();
}
sum(0, mainTo);

while(isThreadWorking(threads)) {}

sw.Stop();
```

ข้อสังเกตพบ

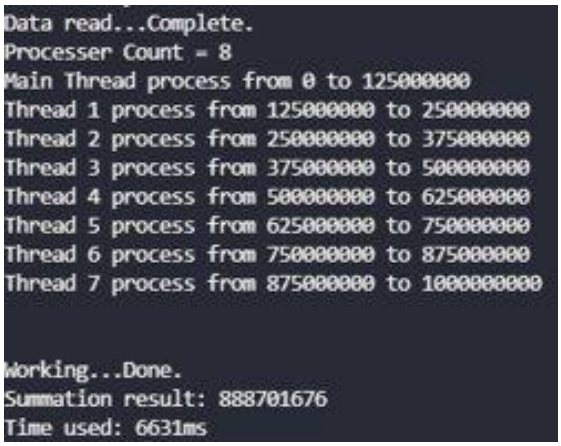
- ทำให้การทำงานเร็วขึ้น แต่จะแตกต่างกันในแต่ละ Hardware ที่ใช้ จำนวน Thread มากอาจจะทำให้เร็วขึ้นกว่า จำนวน Thread ที่น้อยกว่า ไม่เสมอไป ส่งผลให้สมมติฐานข้อที่ 2 , 3 เป็นจริง ส่วนข้อที่ 1 เป็นสมมติฐานที่เท็จ

```
Data read...Complete.
Processer Count = 6
Main Thread process from 0 to 166666672
Thread 1 process from 166666672 to 333333344
Thread 2 process from 333333344 to 500000000
Thread 3 process from 500000000 to 666666688
Thread 4 process from 666666688 to 833333312
Thread 5 process from 833333312 to 1000000000
```

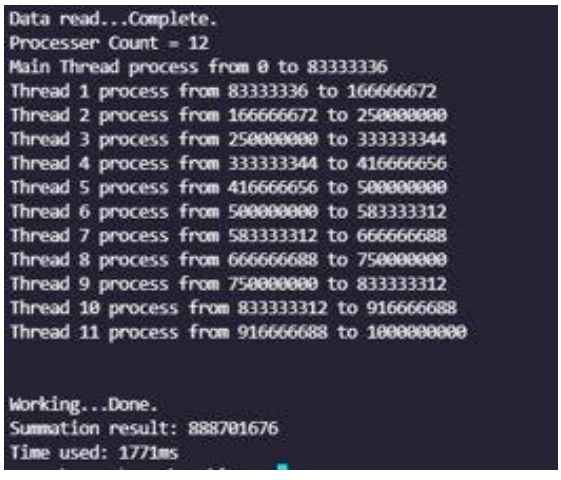
62010496 นิติภูมิ คล้ายเนียม
Intel core i5-8400 2.80GHz 6 Cores 6 Threads



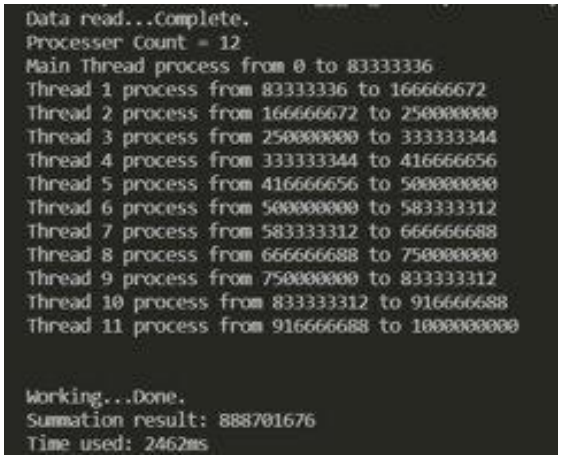
62010494 นิติพัฒน์ บุญเกตุ
Intel Core i7-7700HQ 4 Cores 8 Threads



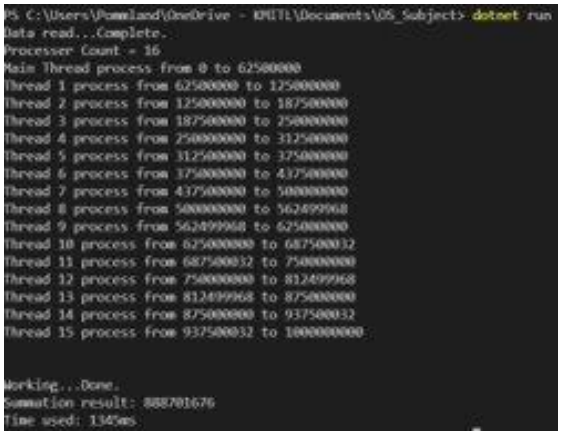
62010052 กันต์ มากทรัพย์สิน
Intel(R) Core(TM) i7-8565U 4 Cores 8 Threads



62010474 นวพล กรุดพันธ์
AMD Ryzen 5 3600 6 Cores 12 Threads



62010453 นนทพันธุ์ รุจิรกาล
Intel Core i7-9750H 6 Cores 12 Threads



62010472 นวพรรษ ศรีบุญเรือง
Intel Core i7-10700 8 Cores 16 Threads

ข้อสรุป

จากการทดลอง Version ทั้งหมดจะพบว่า

สมมติฐานข้อที่ 1 การเพิ่ม Thread จะทำให้ความเร็วเพิ่มขึ้นแน่นอน ไม่เป็นจริงทั้งหมด

- เนื่องจาก การเพิ่ม Thread ขึ้นไปมากๆจะเห็นว่าความเร็วที่เพิ่มขึ้นจะค่อยๆลดลงไปเรื่อยๆจนถึงจุดนึงที่จะไม่ทำให้ความเร็วเพิ่มมากขึ้นไปอีกได้

สมมติฐานข้อที่ 2 Hardware ส่งผลต่อการทำงาน

- จากการทดลองที่ 5 จะพบว่า Hardware แต่ละตัวทำงานได้ไม่เท่ากัน

สมมติฐานข้อที่ 3 การปรับปรุง Algorithm จะทำงานได้เร็วขึ้น

- การทดลองที่ 1 และ 5 จะพบว่าทำงานได้เร็วขึ้น ขึ้นอยู่กับรูปแบบการปรับปรุง