

# Concurrency Part-II

## รายชื่อสมาชิก

64010516	ปัญญวิชญ์ วชิรเศรษฐศิริชัย
64010659	ภัทราภรณ์ จันเดชา
64010683	ภูมिरพี สิ้นศิริ
64010761	วรสพล รังษี
64010845	ศิริสิทธิ์ เทียนเจริญชัย

## Environment ที่ใช้ทดสอบ

- CPU : AMD Ryzen 7 5800H (8 core 16 thread)
- OS : EndeavourOS (Arch-based Linux)
- รันในโหมด Performance
- ไม่มีโปรแกรมอื่น ๆ รันอยู่
- ทุก ๆ ครั้งที่ทดสอบจะทำการ build เป็น Release แล้วจึงทดสอบ

## Version 0 (Original)

```
using System;
using System.Diagnostics;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Threading;

namespace Problem01
{
    class Program
    {
        static byte[] Data_Global = new byte[1000000000];
        static long Sum_Global = 0;
        static int G_index = 0;

        static int ReadData()
        {
            int returnData = 0;
            FileStream fs = new FileStream("Problem01.dat", FileMode.Open);
            BinaryFormatter bf = new BinaryFormatter();

            try
            {
                Data_Global = (byte[]) bf.Deserialize(fs);
            }
        }
    }
}
```

```

        catch (SerializationException se)
        {
            Console.WriteLine("Read Failed:" + se.Message);
            returnData = 1;
        }
        finally
        {
            fs.Close();
        }

        return returnData;
    }
    static void sum()
    {
        if (Data_Global[G_index] % 2 == 0)
        {
            Sum_Global -= Data_Global[G_index];
        }
        else if (Data_Global[G_index] % 3 == 0)
        {
            Sum_Global += (Data_Global[G_index]*2);
        }
        else if (Data_Global[G_index] % 5 == 0)
        {
            Sum_Global += (Data_Global[G_index] / 2);
        }
        else if (Data_Global[G_index] % 7 == 0)
        {
            Sum_Global += (Data_Global[G_index] / 3);
        }
        Data_Global[G_index] = 0;
        G_index++;
    }
    static void Main(string[] args)
    {
        Stopwatch sw = new Stopwatch();
        int i, y;

        /* Read data from file */
        Console.Write("Data read...");
        y = ReadData();
        if (y == 0)
        {
            Console.WriteLine("Complete.");
        }
        else
        {
            Console.WriteLine("Read Failed!");
        }

        /* Start */
        Console.Write("\n\nWorking...");
        sw.Start();
    }

```

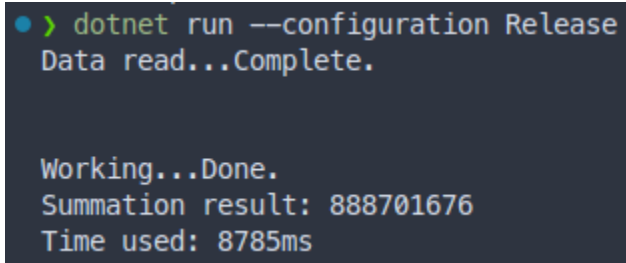
```

        for (i = 0; i < 1000000000; i++)
            sum();
        sw.Stop();
        Console.WriteLine("Done.");

        /* Result */
        Console.WriteLine("Summation result: {0}", Sum_Global);
        Console.WriteLine("Time used: " + sw.ElapsedMilliseconds.ToString() + "ms");
    }
}

```

ผลลัพธ์:



```

• > dotnet run --configuration Release
Data read...Complete.

Working...Done.
Summation result: 888701676
Time used: 8785ms

```

## Version 1

```
using System;
using System.Diagnostics;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Threading.Tasks;
using System.Collections.Concurrent;

#pragma warning disable SYSLIB0011

namespace Problem01
{
    class Program
    {
        static byte[] Data_Global = new byte[1000000000];
        static long Sum_Global = 0;

        static void ReadData()
        {
            FileStream fs = new FileStream("Problem01.dat", FileMode.Open);
            BinaryFormatter bf = new BinaryFormatter();

            try
            {
                Data_Global = (byte[])bf.Deserialize(fs);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Read Failed: " + ex.Message);
            }
            finally
            {
                fs.Close();
            }
        }

        static long Sum(int startIndex, int endIndex)
        {
            long localSum = 0;

            for (int i = startIndex; i < endIndex; i++)
            {
                byte value = Data_Global[i];

                if (value % 2 == 0)
                {
                    localSum -= value;
                }
                else if (value % 3 == 0)
                {
                    localSum += value * 2;
                }
                else if (value % 5 == 0)
                {
                    localSum += value * 2;
                }
            }
        }
    }
}
```

```

        {
            localSum += value / 2;
        }
        else if (value % 7 == 0)
        {
            localSum += value / 3;
        }

        Data_Global[i] = 0;
    }

    return localSum;
}

static void Main(string[] args)
{
    Stopwatch sw = new Stopwatch();

    /* Read data from file */
    Console.Write("Data read...");
    ReadData();
    Console.WriteLine("Complete.");

    /* Start */
    Console.Write("\n\nWorking...");
    sw.Start();

    Sum_Global = Sum(0, Data_Global.Length);

    sw.Stop();
    Console.WriteLine("Done.");

    /* Result */
    Console.WriteLine("Summation result: {0}", Sum_Global);
    Console.WriteLine("Time used: " + sw.ElapsedMilliseconds.ToString() +
"ms");
}
}

```

ผลลัพธ์:

```

> dotnet run --configuration Release
Data read...Complete.

Working...Done.
Summation result: 888701676
Time used: 7974ms

```

สิ่งที่แก้ไขจาก Version ก่อนหน้า :

- เนื่องจาก Version 0 มีการเรียก Function เป็นจำนวนหลายรอบ ซึ่งการ Call function จำเป็นต้องเลื่อน Register stack pointer, Base pointer และ Instruction pointer หลายรอบทำให้ต้องใช้เวลานานขึ้น

ปัญหาที่พบใน Version นี้ :

- โค้ดทำงานในแบบ Single-threaded ทำให้การคำนวณและประมวลผลข้อมูลในขนาดใหญ่จะช้าลง เนื่องจากทำงานบน Thread เพียงหนึ่ง Thread เท่านั้น ทำให้เวลาการทำงานยาวขึ้นและไม่เหมาะสมกับขนาดข้อมูลใหญ่ที่มีอยู่ใน Data\_Global

ข้อสังเกต/สาเหตุของปัญหา :

- มีการปรับปรุงเพื่อให้เกิด Parallelism ในการประมวลผล เพื่อให้สามารถใช้ทรัพยากรคอมพิวเตอร์หลาย Thread พร้อมกันและเพิ่มความเร็วในการคำนวณ

แก้ไข จาก ver 0 :

เปลี่ยนการเรียก function ทุก ๆ รอบ  
เป็น for loop ใน function แทน

เพราะ การเรียก function จะต้องสร้าง stack frame จำนวนรอบมาก ๆ  
โดย จำเป็นต้องเลื่อน Register stack pointer,  
Base pointer  
และ Instruction pointer

จะทำให้ใช้  
ทรัพยากรมากขึ้น

ปัญหาที่พบใน Version นี้ :

โค้ดทำงานในแบบ Single-threaded ทำให้การคำนวณ และประมวลผลข้อมูลในขนาดใหญ่จะช้าลง เนื่องจากทำงานบนแค่ Thread เดียว เท่านั้น ทำให้เวลาการทำงานยาวขึ้น และไม่เหมาะสมกับขนาดข้อมูลใหญ่ที่มีอยู่ใน Data\_Global

## Version 2

```
using System;
using System.Diagnostics;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Threading.Tasks;

#pragma warning disable SYSLIB0011
namespace Problem01
{
    class Program
    {
        static byte[] Data_Global = new byte[1000000000];
        static long[] PartialSums;
        static int BatchSize = 10000; // Adjust the batch size as needed
        static int NumThreads = Environment.ProcessorCount;

        static void ReadData()
        {
            FileStream fs = new FileStream("Problem01.dat", FileMode.Open);
            BinaryFormatter bf = new BinaryFormatter();

            try
            {
                Data_Global = (byte[])bf.Deserialize(fs);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Read Failed: " + ex.Message);
            }
            finally
            {
                fs.Close();
            }
        }

        static void Sum(int startIndex, int endIndex, int threadIndex)
        {
            long localSum = 0;

            for (int i = startIndex; i < endIndex; i++)
            {
                byte value = Data_Global[i];

                if (value % 2 == 0)
                {
                    localSum -= value;
                }
            }
        }
    }
}
```

```

        else if (value % 3 == 0)
        {
            localSum += value * 2;
        }
        else if (value % 5 == 0)
        {
            localSum += value / 2;
        }
        else if (value % 7 == 0)
        {
            localSum += value / 3;
        }

        Data_Global[i] = 0;
    }

    PartialSums[threadIndex] = localSum;
}

static void Main(string[] args)
{
    Stopwatch sw = new Stopwatch();

    /* Read data from file */
    Console.WriteLine("Data read...");
    ReadData();
    Console.WriteLine("Complete.");

    /* Start */
    Console.WriteLine("\n\nWorking...");
    sw.Start();

    PartialSums = new long[NumThreads];
    Parallel.For(0, NumThreads, threadIndex =>
    {
        int startIndex = threadIndex * BatchSize;
        int endIndex = Math.Min(startIndex + BatchSize, Data_Global.Length);

        Sum(startIndex, endIndex, threadIndex);
    });

    long totalSum = 0;
    for (int i = 0; i < NumThreads; i++)
    {
        totalSum += PartialSums[i];
    }
}

```



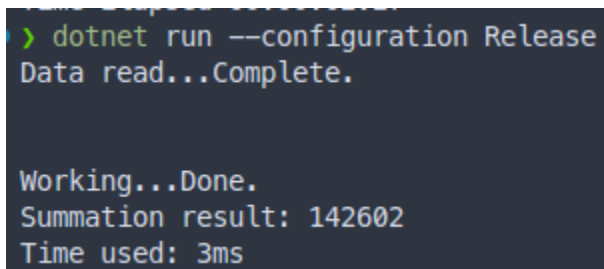
```

        sw.Stop();
        Console.WriteLine("Done.");

        /* Result */
        Console.WriteLine("Summation result: {0}", totalSum);
        Console.WriteLine("Time used: " +
            sw.ElapsedMilliseconds.ToString() + "ms");
    }
}

```

ผลลัพธ์:



```

> dotnet run --configuration Release
Data read...Complete.

Working...Done.
Summation result: 142602
Time used: 3ms

```

สิ่งที่แก้ไขจาก Version ก่อนหน้า :

- ใช้ Parallelism เพื่อแบ่งงานคำนวณในหลาย Thread เพื่อให้การประมวลผลข้อมูลมีประสิทธิภาพมากขึ้น และลดเวลาที่ใช้ในการทำงาน โดยสร้างอาร์เรย์ PartialSums เพื่อเก็บผลรวมชั่วคราวของแต่ละ Thread
- ใช้ Parallel.For เพื่อแบ่งงานให้กับ Thread ตามจำนวนที่กำหนด โดยแต่ละ Thread จะประมวลผลช่วงข้อมูลที่ได้รับและเก็บผลรวมชั่วคราวลงใน PartialSums ตาม Thread ที่เกี่ยวข้อง สุดท้ายจะทำการรวมผลรวมชั่วคราวจากทุก Thread เพื่อได้ผลรวมทั้งหมด

ปัญหาที่พบใน Version นี้ :

- เกิดการแก้ไขข้อมูลร่วมกัน ส่งผลให้เกิดปัญหาของการแก้ไขข้อมูลที่ไม่ถูกต้อง

ข้อสังเกต/สาเหตุของปัญหา :

- ส่วนของการคำนวณผลรวมข้อมูลทุกส่วนจะเก็บผลรวมในแต่ละส่วนไว้ใน PartialSums และหลังจากการคำนวณเสร็จสิ้นจะมีการรวมผลรวมของทุกส่วนเข้าด้วยกันในลำดับหลัง ๆ นี้ ทำให้เกิดการแก้ไขข้อมูลร่วมกันในเวลาเดียวกัน

### Version 3 (Final version)

```
using System;
using System.Diagnostics;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Threading.Tasks;
using System.Collections.Concurrent;

#pragma warning disable SYSLIB0011

namespace Problem01
{
    class Program
    {
        static byte[] Data_Global = new byte[1000000000];
        static long Sum_Global = 0;
        static int NumThreads = Environment.ProcessorCount;

        static void ReadData()
        {
            FileStream fs = new FileStream("Problem01.dat", FileMode.Open);
            BinaryFormatter bf = new BinaryFormatter();

            try
            {
                Data_Global = (byte[])bf.Deserialize(fs);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Read Failed: " + ex.Message);
            }
            finally
            {
                fs.Close();
            }
        }

        static void Sum (int startIndex, int endIndex)
        {
            long localSum = 0;

            for (int i = startIndex; i < endIndex; i++)
            {
                byte value = Data_Global[i];
            }
        }
    }
}
```

```

        if (value % 2 == 0)
        {
            localSum -= value;
        }
        else if (value % 3 == 0)
        {
            localSum += value * 2;
        }
        else if (value % 5 == 0)
        {
            localSum += value / 2;
        }
        else if (value % 7 == 0)
        {
            localSum += value / 3;
        }

        Data_Global[i] = 0;
    }

    lock (Data_Global)
    {
        Sum_Global += localSum;
    }
}

static void Main(string[] args)
{
    Stopwatch sw = new Stopwatch();

    /* Read data from file */
    Console.Write("Data read...");
    ReadData();
    Console.WriteLine("Complete.");

    /* Start */
    Console.Write("\n\nWorking...");
    ParallelOptions options = new ParallelOptions
    {
        MaxDegreeOfParallelism = NumThreads
    };
    var rangePartitioner = Partitioner.Create(0, Data_Global.Length);
    sw.Start();

    Parallel.ForEach(
        rangePartitioner,
        options,
        range => Sum(range.Item1, range.Item2)
    );
}

```

```

        sw.Stop();
        Console.WriteLine("Done.");

        /* Result */
        Console.WriteLine("Summation result: {0}", Sum_Global);
        Console.WriteLine("Time used: " + sw.ElapsedMilliseconds.ToString() +
"ms");
    }
}
}

```

ผลลัพธ์:

```

> dotnet run --configuration Release
Data read...Complete.

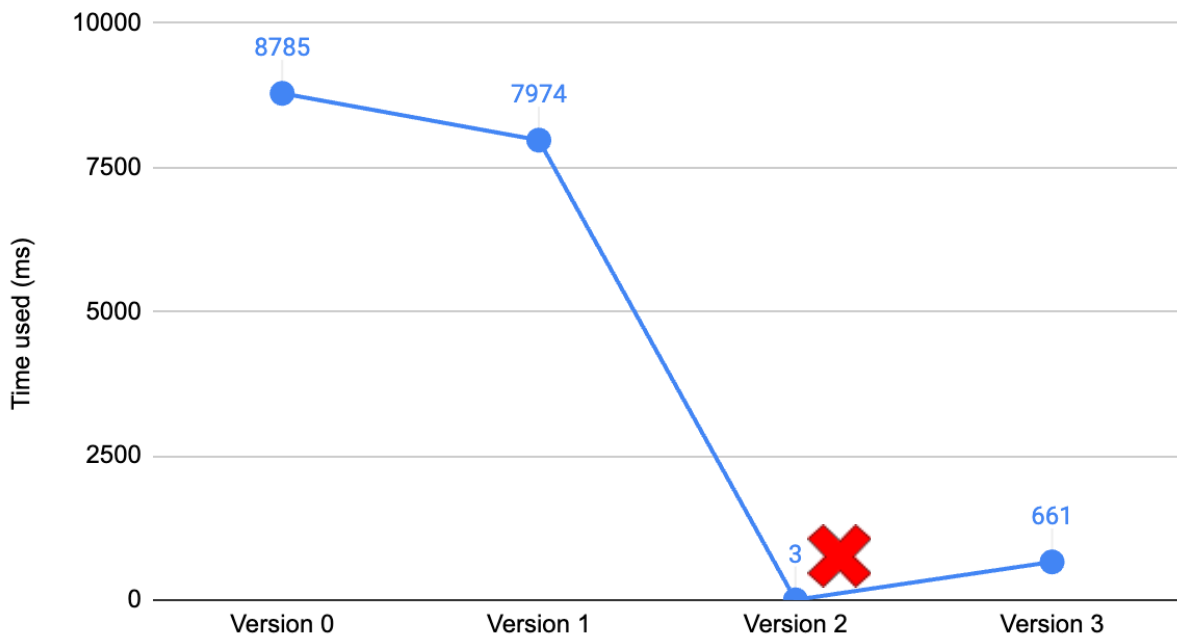
Working...Done.
Summation result: 888701676
Time used: 661ms

```

สิ่งที่แก้ไขจาก Version ก่อนหน้า :

- มีการใช้งาน lock เพื่อป้องกันการแชร์ข้อมูลร่วมกันที่สามารถเขียนได้พร้อมกันภายในส่วนของ Sum ที่ทำงานแบบพร้อมๆ นี่เป็นการรับประกันว่าการปรับปรุงค่า Sum\_Global จะไม่ถูกเข้าถึง หรือแก้ไขโดย Thread อื่น ๆ ในเวลาเดียวกัน
- ใช้ตัวแปร Sum\_Global เพื่อเก็บผลรวมที่เกิดขึ้นจากการคำนวณของ Sum โดยตรง แทนที่จะใช้ PartialSums ในการเก็บผลรวมแต่ละส่วน และรวมผลรวมทั้งหมดในภายหลัง
- ได้นำเอาผลรวม จากทุกส่วนของข้อมูลมาประมวลผลในเซตของช่วงข้อมูลที่แยกกันอย่างชัดเจน ด้วยการใช้ Parallel.ForEach และการใช้ Partitioner ในการแบ่งช่วงข้อมูลเพื่อให้แต่ละเซตของช่วงข้อมูลทำงานพร้อมกันที่จำนวน Thread ที่สูงสุดที่กำหนดไว้ (เทียบกับการใช้ Parallel.For ในโค้ด version 2 ที่แบ่งงานตามจำนวน Thread ที่เป็นค่าคงที่ NumThreads) เป้าหมายของการใช้ Parallel.ForEach และ Partitioner คือเพื่อเพิ่มประสิทธิภาพในกรณีที่ปริมาณงานที่แตกต่างกันระหว่างช่วงข้อมูลน้อยหรือมากกว่ากัน

Time used in every version



จากกราฟ แสดงให้เห็นถึงระยะเวลาในการทำงานของ program ในแต่ละ version ในสภาพแวดล้อมเดียวกัน

## สรุปผล

การทดลองได้สรุปถึงปัจจัยสำคัญที่มีผลต่อความเร็วของโปรแกรม ได้แก่ ด้าน Hardware และ Software โดย Hardware อาทิเช่น จำนวน Core และ Thread ของ CPU และความถี่ทำให้โปรแกรมสามารถประมวลผลได้เร็วขึ้น แต่ Software เป็นปัจจัยที่สำคัญมากกว่า เนื่องจาก Hardware มีข้อจำกัดในด้านราคาและการพัฒนา ในขณะที่ Software สามารถปรับปรุง แก้ไขได้ง่ายขึ้น โดยใช้วิธีการต่างๆ เพื่อเพิ่มประสิทธิภาพในการทำงานของโปรแกรม

การใช้ Multi-threading และ Parallel.For เป็นวิธีการที่ช่วยเพิ่มความเร็วในการทำงานของโปรแกรม โดยให้งานสามารถทำงานพร้อมกันหลายงาน แต่ผลการทดลองพบว่า Parallel.For มีความเร็วกว่า Multi-threading โดยจากการทดลอง อาจเป็นเพราะวิธีการเขียนและการปรับปรุงแบบขนานของ Parallel.For ช่วยให้โปรแกรมสามารถใช้ทรัพยากรของระบบได้อย่างมีประสิทธิภาพมากยิ่งขึ้น

ดังนั้น การทำงานแบบ Parallel.For และการใช้ Multi-threading เป็นวิธีที่สามารถเพิ่มความเร็วในการทำงานของโปรแกรมได้ และจากผลการทดลอง Parallel.For มีความเร็วกว่า Multi-threading แต่ผลลัพธ์เหล่านี้อาจขึ้นอยู่กับวิธีการเขียนและปรับปรุงของโปรแกรมแต่ละรูปแบบ