MCTE 4327

Software Engineering

Week 09 Parallel Software Engineering

Outline

- Parallel class
- Parallel for loop
- Parallel for each

Multi-core CPU

- Over the past 10 years, CPU manufacturers have shifted from single- to multicore processors.
- This is problematic for us as programmers because single-threaded code does not automatically run faster as a result of those extra cores.
- Leveraging multiple cores is easy for most server applications, where each thread can independently handle a separate client request, but is harder on the desktop

Parallel software engineering on desktop app

- Parallel software engineering on a desktop application typically requires that you take your computationally intensive code and do the following:

  1. *Partition* it into small chunks.
  2. Execute those chunks in parallel via multithreading.
  3. *Collate* the results as they become available, in a thread-safe and performant manner.

Amdahl’s law

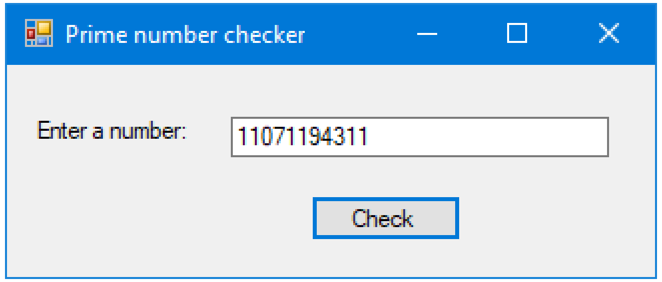
- A challenge in leveraging multicores is Amdahl’s law, which states that the maximum performance improvement from parallelization is governed by the portion of the code that must execute sequentially.
- For instance, if only two-thirds of an algorithm’s execution time is parallelizable, you can never exceed a threefold performance gain even with an infinite number of cores.

High parallelizable problems

- The easiest gains come with what’s called *embarrassingly parallel* problems—where a job can be divided easily into tasks that execute efficiently on their own (structured parallelism is very well suited to such problems).
- Examples include many image processing tasks, ray tracing, and brute force approaches in mathematics or cryptography.

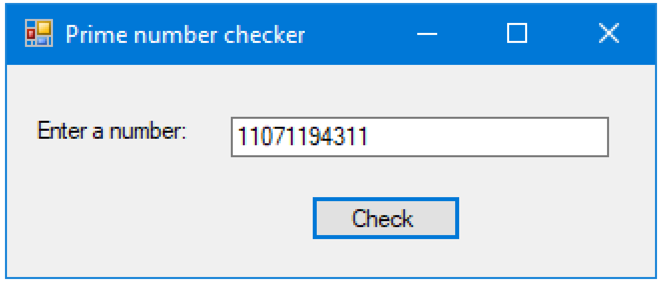
Example – non-parallel processing

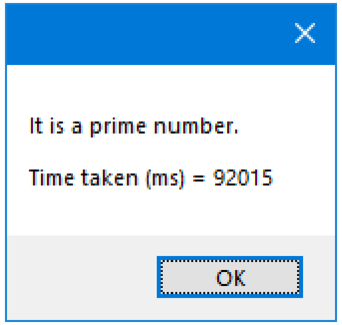
```
using System;
using System.Windows.Forms;

namespace Week9
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            button1.Click += button1_Click;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            long number = long.Parse(textBox1.Text); //Without input validity check

            var sw = new System.Diagnostics.Stopwatch();
            sw.Start();
            bool is_prime = Prime(number);

            sw.Stop();

            if (is_prime)
            {
                MessageBox.Show("It is a prime number.\n\nTime taken (ms) = " + sw.ElapsedMilliseconds.ToString());
            }
            else
            {
                MessageBox.Show("It is not prime number.\n\nTime taken (ms) = " + sw.ElapsedMilliseconds.ToString());
            }
        }
```



```
        private bool Prime(long number)
        {
            //Not the most efficient algorithm
            for (long i = 2; i <= number / 2; i++)
            {
                if (number % i == 0)
                {
                    return false;
                }
            }
            return true;
        }
    }
}
```

Results





Time taken = 92s

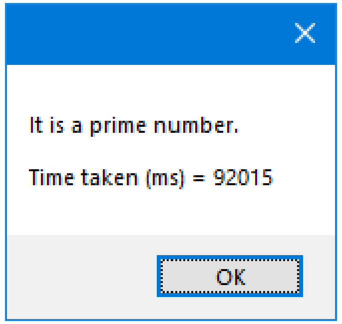
Example – parallel processing

The main loop can be broken down into 4 threads.

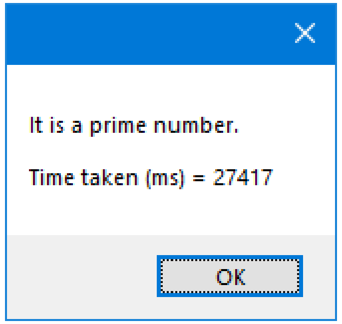
```
private bool PrimeParallel(long number)
{
    Task<bool> Task1 = Task.Run(() =>
    {
        for (long i = 2; i < number / 8; i++)
        {
            if (number % i == 0)
            {
                return false;
            }
        }
        return true;
    });

    Task<bool> Task2 = Task.Run(() =>
    {
        for (long i = number / 8; i < number / 4; i++)
        {
            if (number % i == 0)
            {
                return false;
            }
        }
        return true;
    });

    Task<bool> Task3 = Task.Run(() =>
    {
        for (long i = number / 4; i < 3 * number / 8; i++)
        {
            if (number % i == 0)
            {
                return false;
            }
        }
        return true;
    });

    Task<bool> Task4 = Task.Run(() =>
    {
        for (long i = 3 * number / 8; i <= number / 2; i++)
        {
            if (number % i == 0)
            {
                return false;
            }
        }
        return true;
    });

    return Task1.Result && Task2.Result && Task3.Result && Task4.Result;
}
```

Results

Serial processing



4 parallel threads



Parallel class

- The parallel class under *System.Threading.Tasks* provides the ability to easily parallelize code without multi-threading.

Parallel.For

Performs the parallel equivalent of a C# for loop

Parallel.ForEach

Performs the parallel equivalent of a C# foreach loop

Parallel.For

```
for (int i = 0; i < 100; i++)
{
    Console.WriteLine(i);
}
```

```
Parallel.For(0, 100, i =>
{
    Console.WriteLine(i);
});
```

Parallel.ForEach

```
foreach (string str in list)
 {
     Console.WriteLine(str);
 }
```

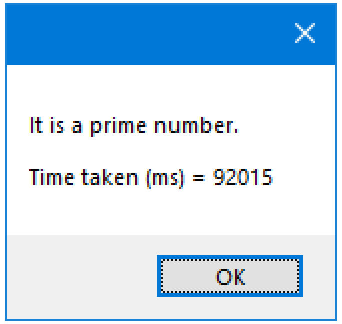
```
Parallel.ForEach(list, str =>
{
    Console.WriteLine(str);
});
```

Example – prime number checker using Parallel class

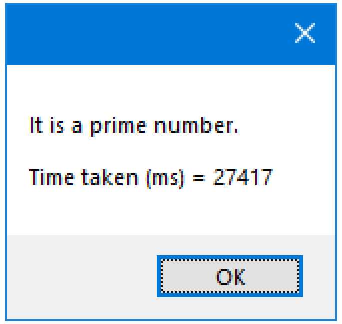
```
private bool Prime_Paralel_Class(long number)
{
    bool prime = true;

    Parallel.For(2, number / 2 + 1, i =>
    {
        if (number % i == 0)
        {
            prime = false; //The loop can be exited as well
        }
    });
    return prime;
}
```

Results

Serial processing



4 parallel threads



Using parallel class

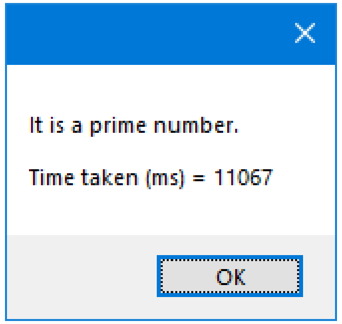
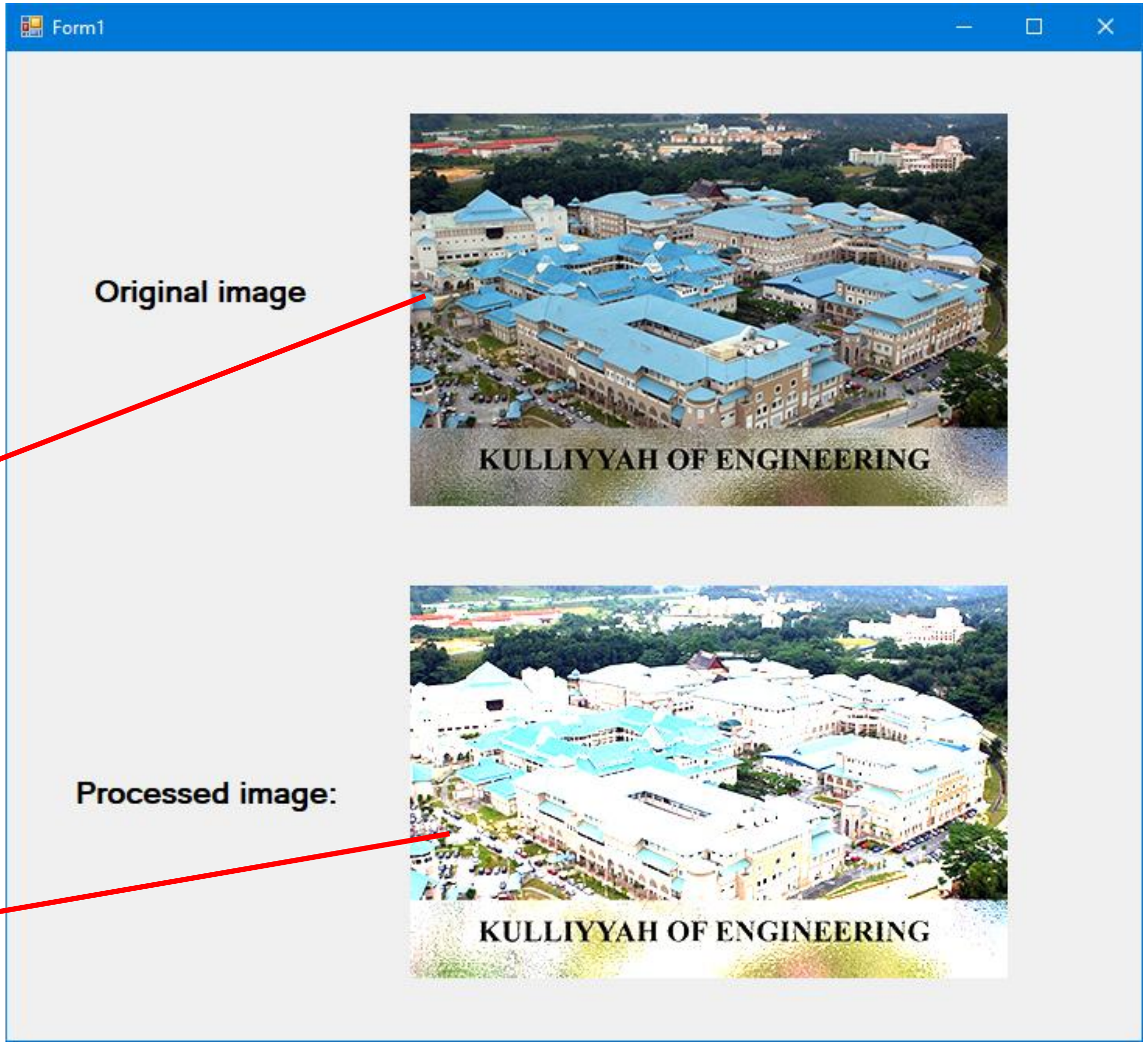


Image processing revisited

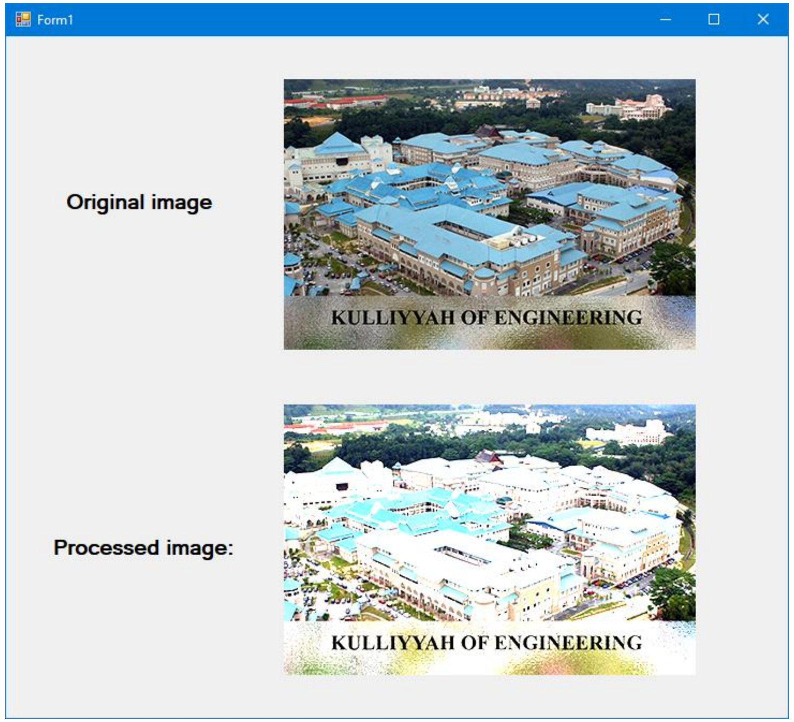
Develop a program that increases the brightness of an image.



PictuerBox1

PictuerBox2

```csharp
public Form1()
{
    InitializeComponent();

    Bitmap image = new Bitmap(@"C:\uiam.jpg");
    pictureBox1.Image = image;
    pictureBox2.Image = Process(image);
}

private Bitmap Process(Bitmap image)
{
    Bitmap processed_image = new Bitmap(image.Width, image.Height);
    for (int row = 0; row < processed_image.Width; row++)
    {
        for (int col = 0; col < processed_image.Height; col++)
        {
            Color color = image.GetPixel(row, col);

            int new_red = color.R * 2;
            int new_green = color.G * 2;
            int new_blue = color.B * 2;

            new_red = (new_red > 255) ? 255 : new_red;
            new_green = (new_green > 255) ? 255 : new_green;
            new_blue = (new_blue > 255) ? 255 : new_blue;
            processed_image.SetPixel(row, col, Color.FromArgb(new_red, new_green, new_blue));
        }
    }
    return processed_image;
}
```



- The image’s dimension is 284x252 only.
- The previous code uses the GetPixel method, which is very slow. The processing time is around 150ms on Core i7 8700 CPU.
- For video, the expected frame rate is only 6 frames per second. It cannot be used for a real-time video stream.
- The performance can be improved by using pointer operations like C++ by turning on unsafe mode.

```
private unsafe Bitmap UnsafeProcess(Bitmap image)
{
    Bitmap processed_image = new Bitmap(image);
    BitmapData imageData = processed_image.LockBits(new Rectangle(0, 0, processed_image.Width, processed_image.Height), ImageLockMode.ReadWrite,
                PixelFormat.Format24bppRgb);
    int bytesPerPixel = 3;

    byte* scan0 = (byte*)imageData.Scan0.ToPointer(); //Pointer that points to the base of the image
    int stride = imageData.Stride;

    for (int col = 0; col < imageData.Height; col++)
    {
        byte* rowdata = scan0 + (col * stride); //Pointer that points to the base row

        for (int row = 0; row < imageData.Width; row++)
        {
            int red = rowdata[row * bytesPerPixel];
            int green = rowdata[row * bytesPerPixel + 1];
            int blue = rowdata[row * bytesPerPixel + 2];

            red *= 2;
            green *= 2;
            blue *= 2;

            rowdata[row * bytesPerPixel] = (red > 255) ? (byte)255 : (byte)red;
            rowdata[row * bytesPerPixel + 1] = (green > 255) ? (byte)255 : (byte)green;
            rowdata[row * bytesPerPixel + 2] = (blue > 255) ? (byte)255 : (byte)blue;
        }
    }

    processed_image.UnlockBits(imageData);

    return processed_image;
}
```

- Processing time is now around 2.1ms per frame or 476 frames per second.

```
private unsafe Bitmap ParallelProcess(Bitmap image)
{
    Bitmap processed_image = new Bitmap(image);
    BitmapData imageData = processed_image.LockBits(new Rectangle(0, 0, processed_image.Width, processed_image.Height), ImageLockMode.ReadWrite,
                PixelFormat.Format24bppRgb);
    int bytesPerPixel = 3;

    byte* scan0 = (byte*)imageData.Scan0.ToPointer();
    int stride = imageData.Stride;

    Parallel.For(0, imageData.Height, col =>
    {
        byte* rowdata = scan0 + (col * stride);

        for (int row = 0; row < imageData.Width; row++)
        {
            int red = rowdata[row * bytesPerPixel];
            int green = rowdata[row * bytesPerPixel + 1];
            int blue = rowdata[row * bytesPerPixel + 2];

            red *= 2;
            green *= 2;
            blue *= 2;

            rowdata[row * bytesPerPixel] = (red > 255) ? (byte)255 : (byte)red;
            rowdata[row * bytesPerPixel + 1] = (green > 255) ? (byte)255 : (byte)green;
            rowdata[row * bytesPerPixel + 2] = (blue > 255) ? (byte)255 : (byte)blue;
        }
    });

    processed_image.UnlockBits(imageData);

    return processed_image;
}
```