CSC 543
MULTIPROCESSING AND CONCURRENT PROGRAMMING

SPRING 2021
FINAL PROJECT

Bob Elward
relwa136@live.Kutztown.edu

Project Description

- **Microsoft C# has numerous Concurrent and Parallel processing options.**
- **The focus here is on a highly productive feature: Task Parallel Library (TPL).**
- **The samples will use .NET Core 5, C# 9.0 and Visual Studio Community 2019.**
- Keep in mind that these tools can be used to develop applications for:
  - Windows
  - Mac OS
  - Linux
  - Android
  - iOS
- GitHub Repository: https://github.com/belward1/CSC543FinalProject
- Microsoft Tools for KU Students: Student Resources - Computer Science and Information Technology - Kutztown University
  - Then click on Software -> Azure Dev Tools

Threads, Tasks and Thread Pools

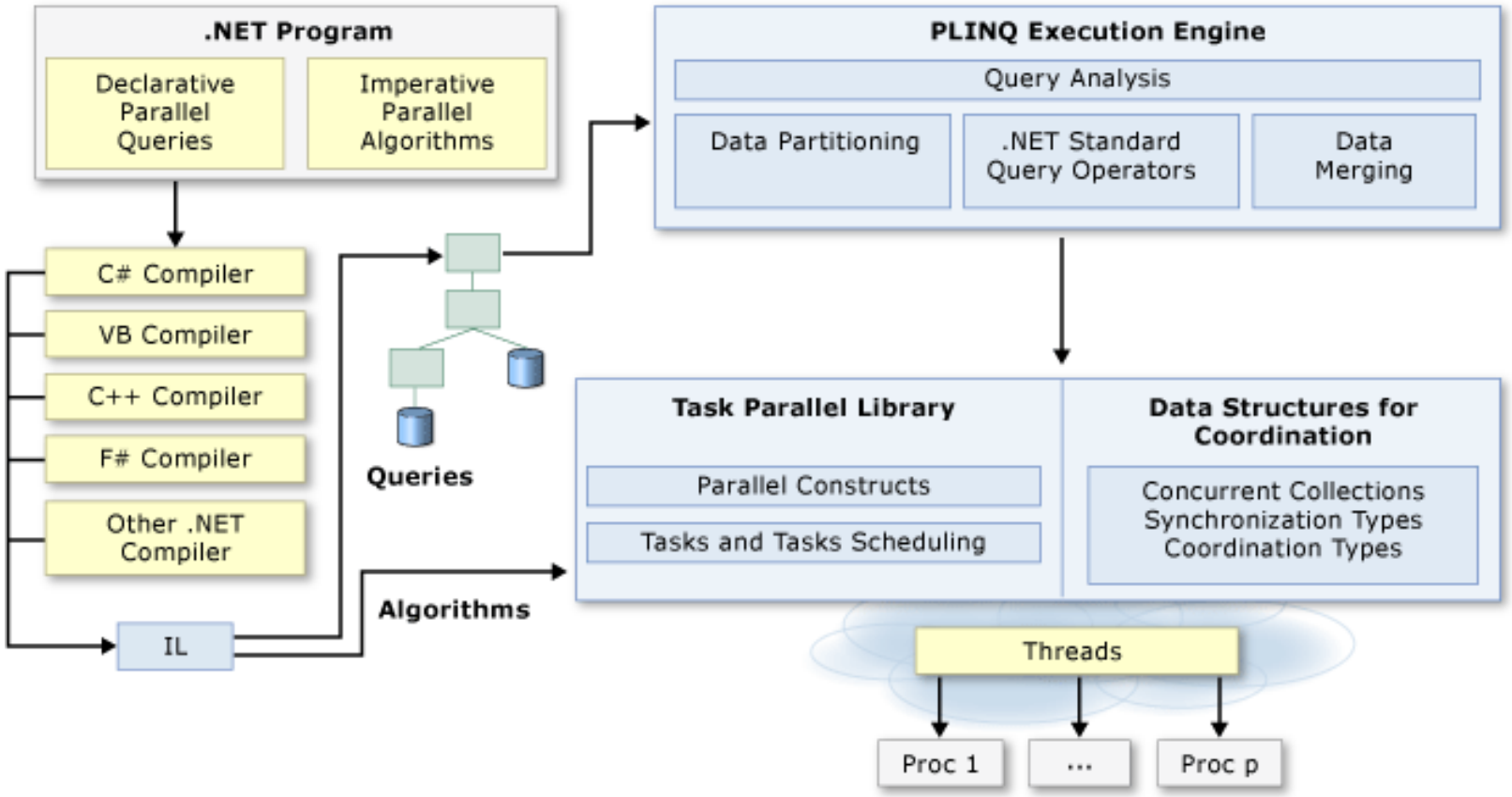
- **Tasks are a higher-level concept:** tasks are basically a promise to run a method and return when it is done.
- **Threads are a lower-level concept:** Threads are a part of your operating system, and the thread class is a way to manage them.
- **Leveraging the thread pool:** tasks use the thread pool, which is a "pool" of threads that can be used and reused. Creating threads can be expensive, which is why we have the thread pool.
- **Threads do not naturally return anything:** Tasks are able to return an object when they are completed. Which makes them great for executing a method and returning the result asynchronously.
- **Cancellation tokens:** Tasks can use cancellation tokens so that they can be requested to be canceled. This token can be passed along to other tasks which will be canceled as well.
- **Tasks support async/await:** async/await is a simple way to wait for an asynchronous method to finish without blocking the parent method.
- **Exceptions:** Threads cannot cascade an Exception to its parent method. A Task does cascade exceptions to the parent method and are caught via a AggregateException.

Task Parallel Library (TPL)

- **Starting with .NET Framework 4, the TPL is the preferred way to write multithreaded and parallel code.**
- **According to Microsoft:**

  “The purpose of the TPL is to make developers more productive by simplifying the process of adding parallelism and concurrency to applications. The TPL scales the degree of concurrency dynamically to most efficiently use all the processors that are available. In addition, the TPL handles the partitioning of the work, the scheduling of threads on the ThreadPool, cancellation support, state management, and other low-level details.

  By using TPL, you can maximize the performance of your code while focusing on the work that your program is designed to accomplish.”



Parallel Programming in .NET

- **TPL Parallel Class Methods:**
  - **Invoke(…)**
  - **For(…)**
  - **ForEach(…)**

- Source: [Parallel Programming in .NET | Microsoft Docs]

JCiP Annotations

- **Coded C# .NET versions of the Java annotations:**
  - **GuardedBy**
  - **Immutable**
  - **NotThreadSafe**
  - **ThreadSafe**

- See:
  - Java Concurrency in Practice
    Brian Goetz, et al
    Addison-Wesley

    ISBN-10: 0-321-34960-1

Parallel.Invoke(…)

- **The Parallel.Invoke(…) method has a single signature.**
- **Parallel.Invoke(Action[] // params Action[] actions to execute**
  );
- **This example will demonstrate invoking three separate actions.**
- See: How to: Use Parallel.Invoke to Execute Parallel Operations | Microsoft Docs

Loops (for and foreach)

Sequential Loop:

```
int n = 10;
for (int i = 0; i < n; i++)
{
        // … do work
}
```

Parallel Parallel.For(…) Loop:

```
int n = 10;
Parallel.For (0, n, i =>
                {
                        // … do work
                }
        );
```

Delegate, Anonymous Method and Lambda Expression

- **Parallel.For third parameter: Action<int>**
  - **Named Delegate:**
    ```
    Parallel.For(0, n, DoWork);

    public void DoWork(int i)
    {
        // … do work
    }
    ```
  - **Anonymous Method:**
    ```
    Parallel.For(0, n, delegate(int i)
                        {
                            // … do work
                        }
                );
    ```
  - **Lambda Expression:**
    ```
    Parallel.For(0, n, (i) =>
                        {
                            // … do work
                        }
                );
    ```
- See: Lambda Expressions in PLINQ and TPL | Microsoft Docs

Parallel.For(…)

- **The Parallel.For(…) method has 12 overloads.**

- ```
  Parallel.For(Int32                  // From inclusive
               , Int32                // To exclusive
               , Action<Int32>        // Body delegate
               );
  ```

- **This example will demonstrate tasks running on separate threads.**

- Delegates, Anonymous functions, Lambda expressions
  See: Delegates, Lambda Expressions & Closures in C# - Alan Zucconi
  ```

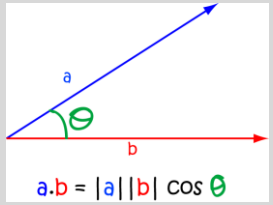
Parallel.ForEach(…) with ThreadLocal

- **The Parallel.ForEach(…) method has 20 overloads.**
- **We'll look at one that allows for ThreadLocal state:**
- **This example will sum the elements in an array.**

```
Parallel.ForEach<TSource,TLocal>(IEnumerable<TSource>                          // Source
                                , Func<TLocal>                                 // Initialize the local state
                                , Func<TSource,ParallelLoopState,TLocal,TLocal> // Body – once per iteration
                                , Action<TLocal>                               // Finalize the local state
                                 );
```

Parallel Dot Product

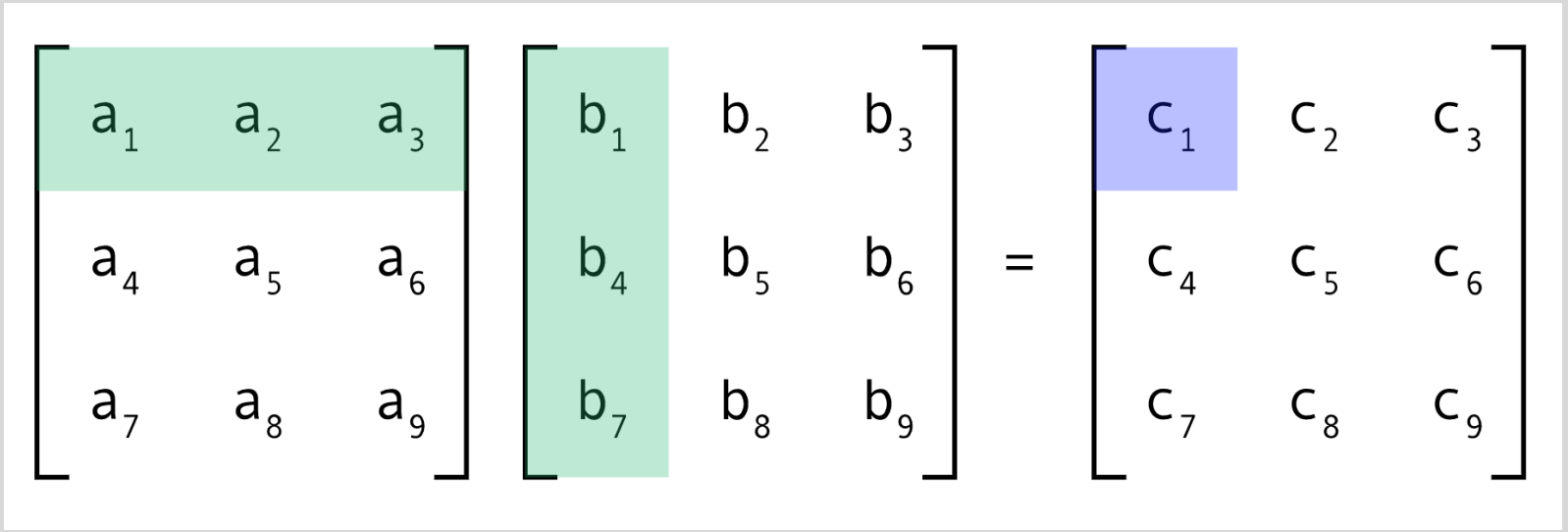
- **A real-world calculation found in engineering and mathematics**
- **Dot Product:**
  - **Geometric: A • B = |A| * |B| * cos(θ)**
  - **Mathematical: Σ (a_i * b_i)**



- **Example shows three (3) alternatives:**
  - **Sequential**
  - **Parallel.For**
  - **Parallel.ForEach w/ range Partitioner**
- **Partitioner:**
  - **Range partitioning**
  - **Chunk partitioning – can dynamically adjust the chunk sizes**
  - **See:** Custom Partitioners for PLINQ and TPL | Microsoft Docs

Parallel Matrix Multiplication

- **A real-world calculation found in engineering and mathematics**



- Image Reference:
  More efficient matrix multiplication (fastai PartII-Lesson08) | by bigablecat | AI³ | Theory, Practice, Business | Medium
- See: Matrix multiplication - Wikipedia

Parallel Pi Calculation

- **Definition of Pi (See: Pi – Wikipedia): Π = Circumference / Diameter**
- **The value can be calculated using infinite series:**
  (See: 5 Ways to Calculate Pi - wikiHow)
  - **Gregory-Leibniz series:**
    Π = (4/1) – (4/3) + (4/5) – (4/7) + (4/9) – (4/11) + (4/13) – (4/15) …
  - **Nilakantha series:**
    Π = 3 + 4/(2*3*4) – 4/(4*5*6) + 4/(6*7*8) – 4/(8*9*10) + 4/(10*11*12) – 4/(12*13*14) …
- **More complex series:**
  (See: Pi (Ramanujan's formula) Calculator - High accuracy calculation (casio.com))
  -

(1) *Ramanujan* 1, 1914

$$\frac{1}{\pi} = \frac{\sqrt{8}}{99^2} \sum_{n=0}^{\infty} \frac{(4n)!}{(4^n n!)^4} \frac{1103 + 26390n}{99^{4n}}$$

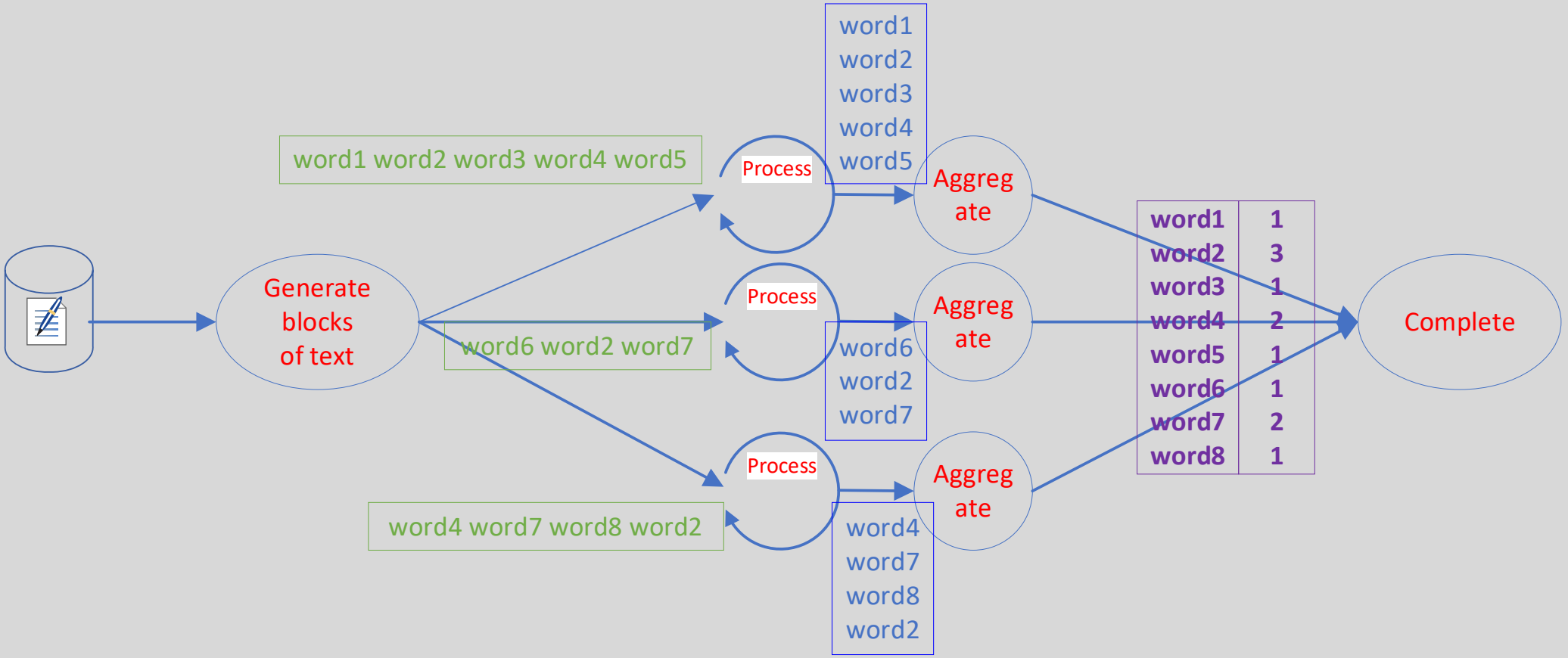
(2) *Ramanujan* 2, 1914

$$\frac{4}{\pi} = \frac{1}{882} \sum_{n=0}^{\infty} \frac{(-1)^n (4n)!}{(4^n n!)^4} \frac{1123 + 21460n}{882^{2n}}$$

(3) *Chudonovsky*, 1987

$$\frac{1}{\pi} = 12 \sum_{n=0}^{\infty} \frac{(-1)^n (6n)!}{(3n)!(n!)^3} \frac{13591409 + 545140134n}{(640320^3)^{n+\frac{1}{2}}}$$

Parallel Map Reduce



Books

- **Concurrency in C# Cookbook**
  Asynchronous, Parallel, and Multithreaded Programming
  Second Edition, 2019
  Stephen Cleary
  ISBN-13: 978-1-492-05450-4
  O'Reilly Media, Sebastopol, CA 95472, USA
  Copyright Stephen Cleary
- **Pro C# 9 With .NET 5**
  Foundational Principles and Practices in Programming
  Tenth Edition, 2021 (Release May 12)
  Andrew Troelsen and Philip Japikse
  ISBN-13: 978-1484269381
  Apress Media, LLC, California, USA
  Springer Science+Business Media New York, USA
  Copyright 2017 by Andrew Troelsen and Philip Japikse
- **C# 9 and .NET 5**
  Fifth Edition, 2020
  Mark J. Price
  ISBN-13: 978-1-80056-810-5
  Packt Publishing Ltd., Birmingham B3 2PB, UK
  Copyright 2020 Packt Publishing