Parallel Processing, Concurrency, and Async Programming in .NET

.NET provides several ways for you to write asynchronous code to make your application more responsive to a user and write parallel code that uses multiple threads of execution to maximize the performance of your user's computer.

In This Section

Asynchronous Programming
Describes mechanisms for asynchronous programming provided by .NET.

Parallel Programming
Describes a task-based programming model that simplifies parallel development, enabling you to write efficient, fine-grained, and scalable parallel code in a natural idiom without having to work directly with threads or the thread pool.

Threading
Describes the basic concurrency and synchronization mechanisms provided by .NET.

Async Overview

- 06/20/2016

Not so long ago, apps got faster simply by buying a newer PC or server and then that trend stopped. In fact, it reversed. Mobile phones appeared with 1ghz single core ARM chips and server workloads transitioned to VMs. Users still want responsive UI and business owners want servers that scale with their business. The transition to mobile and cloud and an internet-connected population of >3B users has resulted in a new set of software patterns.

- Client applications are expected to be always-on, always-connected and constantly responsive to user interaction (for example, touch) with high app store ratings!
- Services are expected to handle spikes in traffic by gracefully scaling up and down.

Async programming is a key technique that makes it straightforward to handle blocking I/O and concurrent operations on multiple cores. .NET provides the capability for apps and services to be responsive and elastic with easy-to-use, language-level asynchronous programming models in C#, Visual Basic, and F#.

Why Write Async Code?

Modern apps make extensive use of file and networking I/O. I/O APIs traditionally block by default, resulting in poor user experiences and hardware utilization unless you want to learn and use challenging patterns. Task-based async APIs and the language-level asynchronous programming model invert this model, making async execution the default with few new concepts to learn.

Async code has the following characteristics:

- Handles more server requests by yielding threads to handle more requests while waiting for I/O requests to return.
- Enables UIs to be more responsive by yielding threads to UI interaction while waiting for I/O requests and by transitioning long-running work to other CPU cores.
- Many of the newer .NET APIs are asynchronous.
- It's easy to write async code in .NET!

Asynchronous programming patterns

- 10/16/2018

.NET provides three patterns for performing asynchronous operations:

- **Task-based Asynchronous Pattern (TAP)**, which uses a single method to represent the initiation and completion of an asynchronous operation. TAP was introduced in .NET Framework 4. **It's the recommended approach to asynchronous programming in .NET.** The async and await keywords in C# and the Async and Await operators in Visual Basic add language support for TAP. For more information, see Task-based Asynchronous Pattern (TAP).
- **Event-based Asynchronous Pattern (EAP)**, which is the event-based legacy model for providing asynchronous behavior. It requires a method that has the `Async` suffix and one or more events, event handler delegate types, and `EventArg`-derived types. EAP was introduced in .NET Framework 2.0. It's no longer recommended for new development. For more information, see Event-based Asynchronous Pattern (EAP).
- **Asynchronous Programming Model (APM)** pattern (also called the IAsyncResult pattern), which is the legacy model that uses the IAsyncResult interface to provide asynchronous behavior. In this pattern, synchronous operations require `Begin` and `End` methods (for example, `BeginWrite` and `EndWrite` to implement an asynchronous write operation). This pattern is no longer recommended for new development. For more information, see Asynchronous Programming Model (APM).

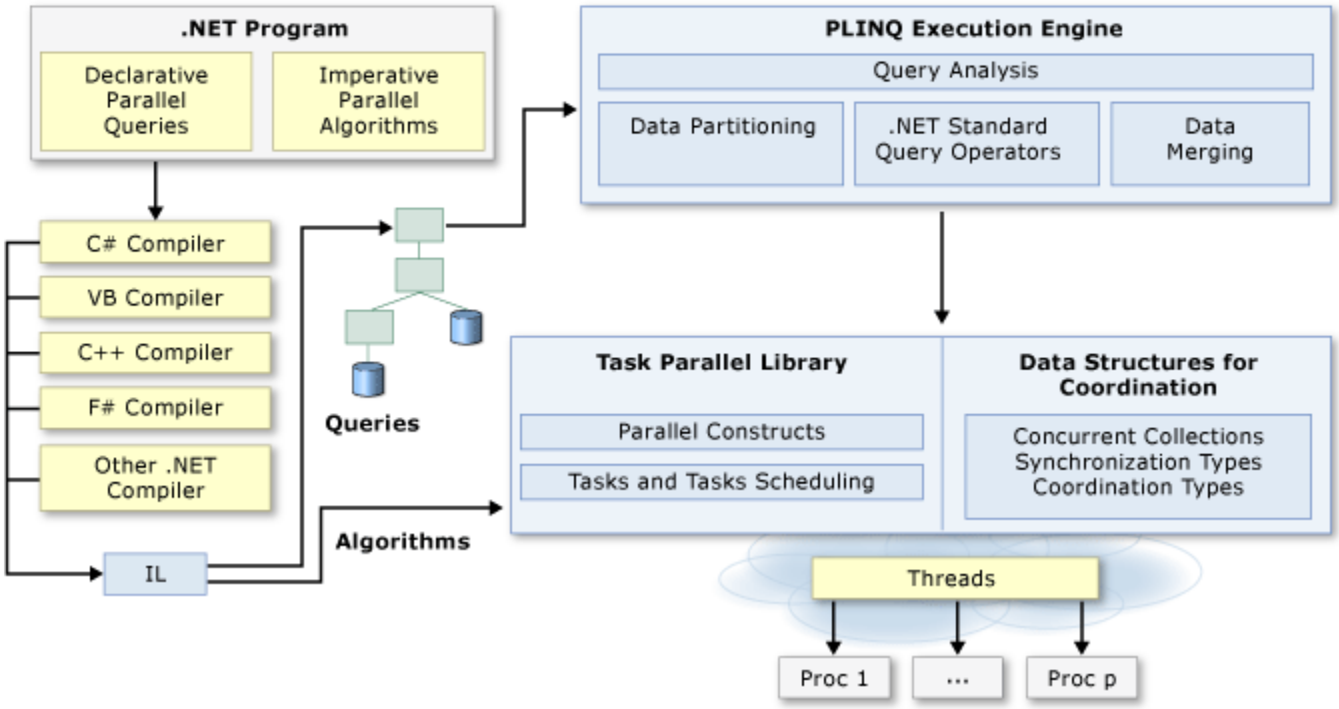
Parallel Programming in .NET

- 09/12/2018

Many personal computers and workstations have multiple CPU cores that enable multiple threads to be executed simultaneously. To take advantage of the hardware, you can parallelize your code to distribute work across multiple processors.

In the past, parallelization required low-level manipulation of threads and locks. Visual Studio and .NET enhance support for parallel programming by providing a runtime, class library types, and diagnostic tools. These features, which were introduced in .NET Framework 4, simplify parallel development. You can write efficient, fine-grained, and scalable parallel code in a natural idiom without having to work directly with threads or the thread pool.

The following illustration provides a high-level overview of the parallel programming architecture in .NET.



Managed threading

- 03/30/2017

Whether you're developing for computers with one processor or several, you want your application to provide the most responsive interaction with the user, even if the application is currently doing other work. Using multiple threads of execution is one of the most powerful ways to keep your application responsive to the user and at the same time make use of the processor in between or even during user events. While this section introduces the basic concepts of threading, it focuses on managed threading concepts and using managed threading.

Note

Starting with .NET Framework 4, multithreaded programming is greatly simplified with the **System.Threading.Tasks.Parallel** and **System.Threading.Tasks.Task** classes, **Parallel LINQ (PLINQ)**, concurrent collection classes in the **System.Collections.Concurrent** namespace, and a programming model that is based on the concept of tasks rather than threads.