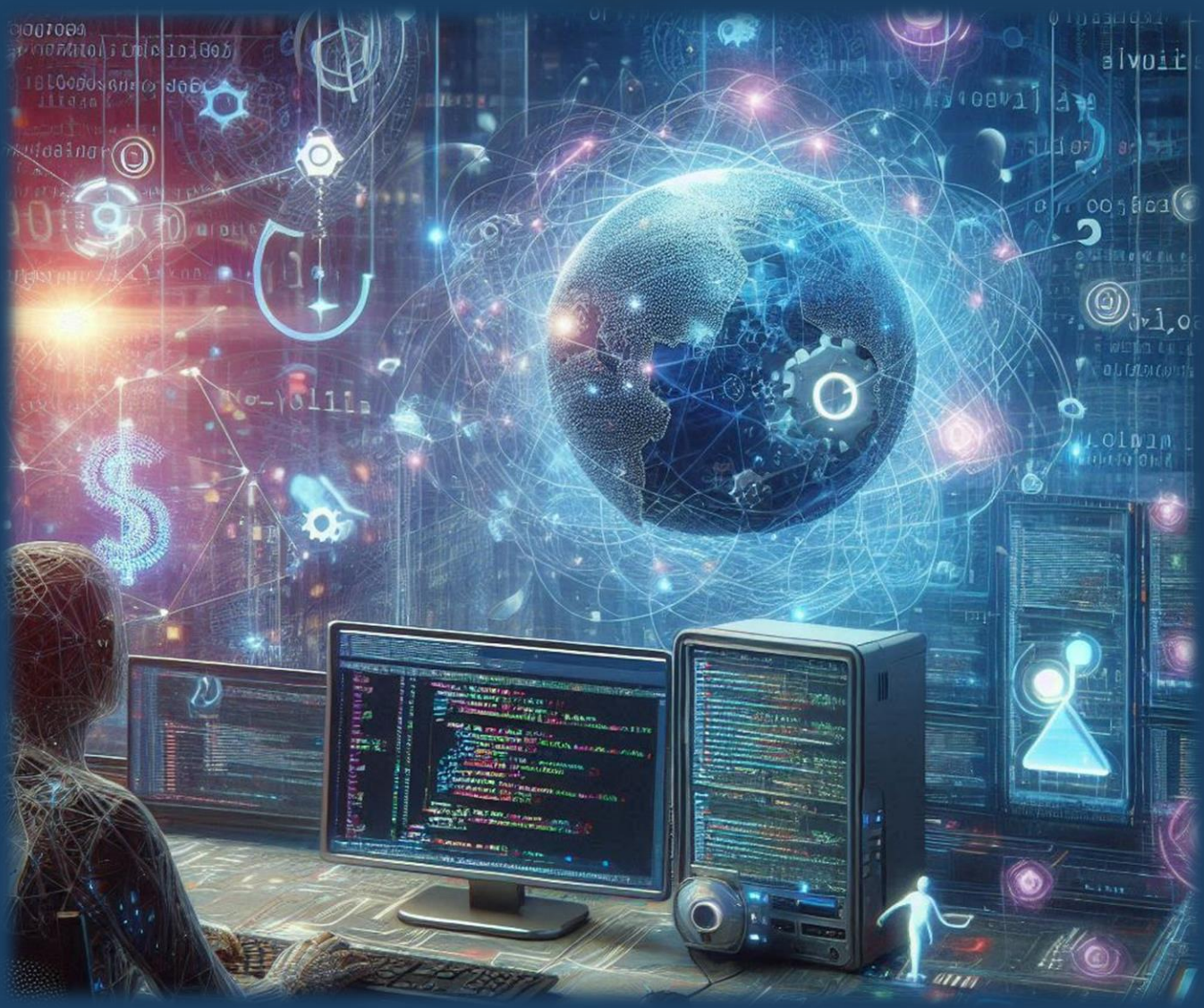
C# Turbocharged

Dominando os princípios da Computação Paralela



Emerson de Oliveira Batista

1

Princípios de Threads

Princípios de Programação Paralela

Threads Simples

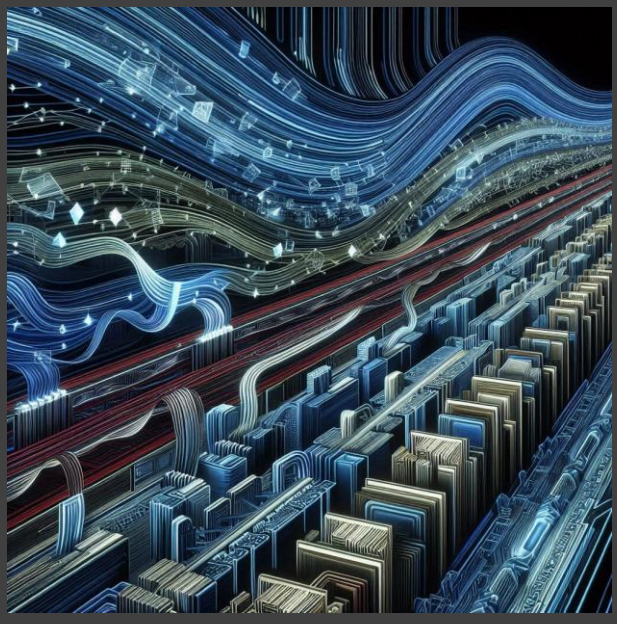
Threads são unidades básicas de execução que permitem que um programa execute várias tarefas simultaneamente, conhecida como fluxos de processo ou processos leves. Em C#, você pode criar threads de forma simples usando a classe Thread. Abaixo está um exemplo de como iniciar e executar uma thread:

Thread Simples

```
using System;
using System.Threading;

class Program
{
    static void Main()
    {
        Thread thread = new Thread(() =>
        {
            Console.WriteLine("Thread em execução.");
        });

        thread.Start();
        thread.Join(); // Aguarda a thread terminar antes de continuar

        Console.WriteLine("Programa principal continuando.");
    }
}
```

2

Fila de Threads



Princípios de Programação Paralela

ThreadPool

O ThreadPool em C# é um pool (fila) de threads gerenciado pelo sistema operacional que permite a reutilização eficiente de threads para executar tarefas. Ele é adequado para cenários onde você precisa controlar o número de threads ativas. Abaixo tem-se um exemplo básico de como usar o ThreadPool:

Thread Pool

```
using System;
using System.Threading;

class Program
{
    static void Main()
    {
        ThreadPool.QueueUserWorkItem((state) ⇒
        {
            Console.WriteLine("Tarefa no ThreadPool em execução.");
        });

        // Espera um pouco para permitir que a tarefa do ThreadPool execute
        Thread.Sleep(1000);

        Console.WriteLine("Programa principal continuando.");
    }
}
```

3

Tarefas Paralelas

Princípios de Programação Paralela

Tasks com Task Parallel Library (TPL)

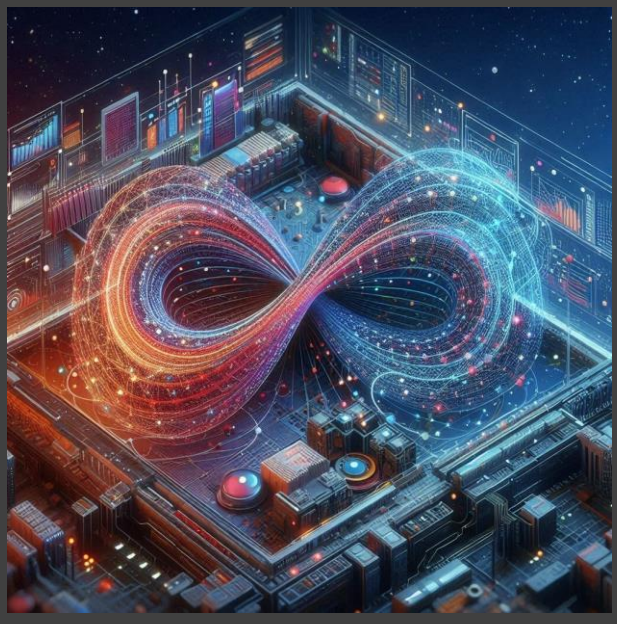
As tarefas (task) são métodos de simplificar o processo de incluir paralelismo e concorrência em aplicações. Neste contexto, a TPL é uma biblioteca poderosa para programação paralela em C#. Ela simplifica a criação e o gerenciamento de tarefas assíncronas. Veja um exemplo de utilização:

TPL

```
using System;
using System.Threading.Tasks;

class Program
{
    static void Main()
    {
        Task<int> task = Task.Run(() ⇒
        {
            return 42;
        });

        Console.WriteLine("Resultado da tarefa: " + task.Result);
    }
}
```

4

Laços Paralelos Finitos



Princípios de Programação Paralela

Paralelismo com Parallel.For

A execução de tarefas repetitivas tem um fluxo, normalmente, linear. Para algumas situações de aplicações pode ser importante usufruir de paralelismo em tarefas repetitivas (laços ou loops).

Para isso o método Parallel.For permite iterar em um intervalo de números de forma paralela, dividindo automaticamente o trabalho entre múltiplos threads. Veja um exemplo prático:

Parallel For

```
using System;
using System.Threading.Tasks;

class Program
{
    static void Main()
    {
        Parallel.For(0, 10, i ⇒
        {
            Console.WriteLine("Iteração " + i);
        });
    }
}
```

5

Laços Paralelos de Coleções

Princípios de Programação Paralela

Paralelismo com Parallel.ForEach

Coleções são muito comuns para diversos tipos de sistemas e aplicações. Neste caso, a aplicação de tarefas repetitivas para cada item da coleção pode ter um desempenho melhor se for processado paralelamente.

Neste cenário, o Parallel.ForEach é ideal para iterar sobre uma coleção de dados de forma paralela, executando operações em múltiplos threads. Aqui está um exemplo de uso:

Parallel ForEach

```
using System;
using System.Threading.Tasks;

class Program
{
    static void Main()
    {
        int[] numeros = { 1, 2, 3, 4, 5 };

        Parallel.ForEach(numeros, numero ⇒
        {
            Console.WriteLine(numero * numero);
        });
    }
}
```

6

Sincronização de fluxo

Princípios de Programação Paralela

Uso de Locks para Sincronização

Em programação concorrente, é essencial garantir que recursos compartilhados sejam acessados de maneira segura. O uso de locks ajuda a evitar condições de corrida. Veja um exemplo simples:

Lock Sync

```
using System;
using System.Threading;

class Program
{
    static int contador = 0;
    static object lockObj = new object();

    static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            Thread thread = new Thread(() =>
            {
                lock (lockObj)
                {
                    contador++;
                    Console.WriteLine("Contador: " + contador);
                }
            });
            thread.Start();
        }
    }
}
```

7

Código Profissional com PLINQ

Princípios de Programação Paralela

PLINQ (Parallel LINQ)

PLINQ é uma extensão do LINQ que permite processamento de consultas de forma paralela, aproveitando automaticamente vários núcleos de CPU. O LINQ atua de forma semelhante a cláusulas SQL para objetos. É ideal para operações em grandes conjuntos de dados. Aqui está um exemplo de uso:

PLINQ

```
using System;
using System.Linq;

class Program
{
    static void Main()
    {
        int[] numeros = { 1, 2, 3, 4, 5 };

        var quadrados = numeros.AsParallel().Select(numero => numero * numero);

        foreach (var resultado in quadrados)
        {
            Console.WriteLine(resultado);
        }
    }
}
```

8

Conclusões

Princípios de Programação Paralela

Conclusões

Dominar técnicas de programação paralela e concorrente em C# pode transformar a performance e a eficiência de suas aplicações. Com esses exemplos simples e eficazes, você estará pronto para explorar e implementar essas técnicas em seus projetos, aproveitando ao máximo o potencial da linguagem e seus recursos para processamento paralelo.

Com técnicas avançadas apresentadas neste documento e o uso correto de locks, você pode criar aplicações mais responsivas e eficientes em C#. Cada técnica oferece maneiras distintas de aproveitar o poder do processamento paralelo e concorrente, adequando-se às necessidades específicas de cada projeto. Explore e experimente essas técnicas para maximizar o desempenho de suas aplicações.

Finalização

Finalizando

Agradecimento e fechamento

Este E-Book foi construído com um objetivo didático de mostrar a aplicação orientada de Inteligências Artificiais (IA) de forma a aumentar a produtividade e geração de aprendizado e conhecimento.

As imagens foram geradas por IA, bem como a estruturação inicial dos tópicos. O texto foi criado e compilado pelo autor, bem como a diagramação deste documento.

Ele é um documento de princípios e tem caráter resumido, na busca por estimulação ao aprofundamento.

https://github.com/mersobap

Autor:
Emerson de Oliveira Batista

GitHub | Linkedin