

Programação Assíncrona e Paralelismo

Johnathan Fercher

January 27, 2018

Table of contents

1. Introdução
2. Programação Síncrona e Assíncrona em C#
3. Paralelismo em C#
4. Programação Assíncrona e Paralela em C#
5. Conclusões

Introdução

Programação Assíncrona tem o propósito de possibilitar a execução de tarefas demoradas sem o bloqueio da execução.

Programação Paralela tem o propósito de possibilitar a execução de tarefas ao mesmo tempo.

Problema assíncrono

- Consumir uma API;
- Operações de CRUD em banco de dados;
- Quaisquer outras operações que demorem: **Treinamento de Inteligências Artificiais, Processamento de Imagens e Etc;**

Problema paralelo

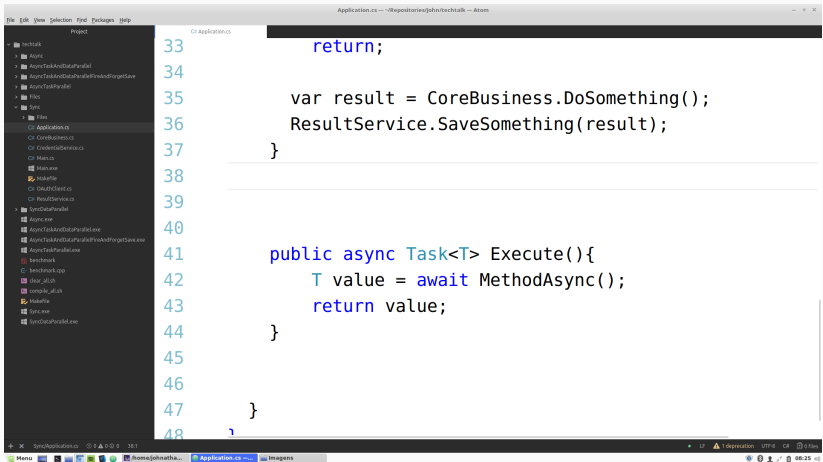
- **Em um jogo:** Uma thread é responsável por obter os comandos do Joystick e outras N são responsáveis por controlar a renderização de objetos, comandos de adversários e etc;
- **Em um robô:** Uma thread é responsável pela leitura de sensores e outras N são responsáveis por controlar motores, realizar comunicação com outros robôs e etc;
- **Em um algoritmo de reconhecimento facial:** Pode-se dividir uma imagem em quadrantes, onde N threads serão responsáveis por aplicar filtros nas secções;

Programação Síncrona e Assíncrona em C#

Código síncrono

```
33         return;  
34  
35         var result = CoreBusiness.DoSomething();  
36         ResultService.SaveSomething(result);  
37     }  
38  
39  
40  
41     public void Execute(){  
42         var value = Method();  
43     }  
44  
45  
46     }  
47 }  
48
```


Código assíncrono

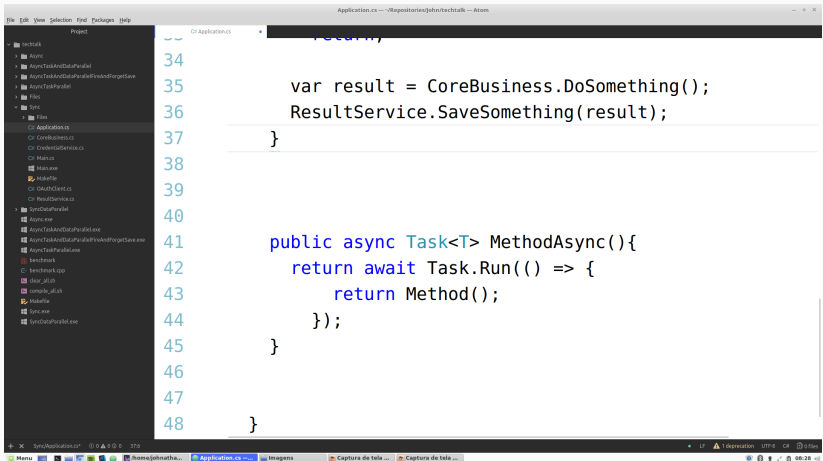


```
Application.cs -- V:\Repositories\john\techtalk -- Atom

Project
  Application.cs
  Async
  AsyncTaskAndDataParallel
  AsyncTaskAndDataParallelForbiddenForgetSave
  AsyncTaskParallel
  Files
  Sync
  Files
  Application.cs
  CoreBusiness.cs
  CredentialsServices
  Maths.cs
  Main.exe
  MainFile
  OAuthClient.cs
  ResultService.cs
  SyncDataParallel
  Async.exe
  AsyncTaskAndDataParallel.exe
  AsyncTaskAndDataParallelForbiddenForgetSave.exe
  AsyncTaskParallel.exe
  benchmark
  benchmarkApp
  diag_utility
  compile_utility
  MainFile
  Sync.exe
  SyncDataParallel.exe

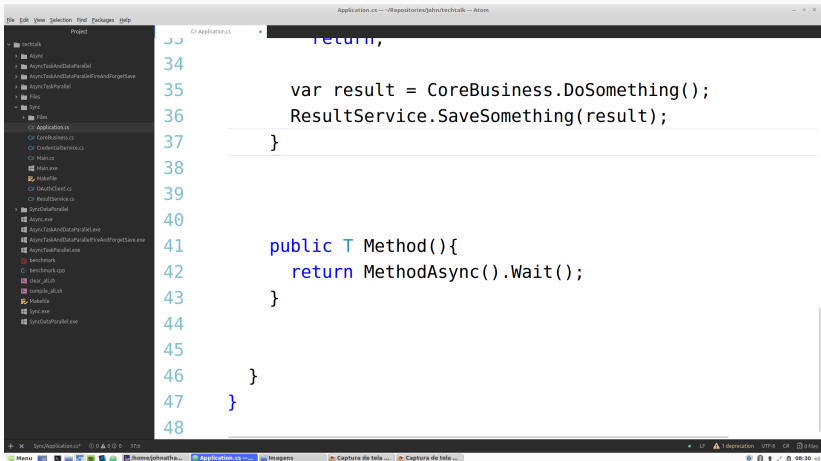
33      return;
34
35      var result = CoreBusiness.DoSomething();
36      ResultService.SaveSomething(result);
37  }
38
39
40
41  public async Task<T> Execute(){
42      T value = await MethodAsync();
43      return value;
44  }
45
46
47  }
48  }
```

Transformando código síncrono em assíncrono



```
34  
35  
36 var result = CoreBusiness.DoSomething();  
37 ResultService.SaveSomething(result);  
38 }  
39  
40  
41 public async Task<T> MethodAsync(){  
42     return await Task.Run(() => {  
43         return Method();  
44     });  
45 }  
46  
47  
48 }
```

Transformando código assíncrono em síncrono



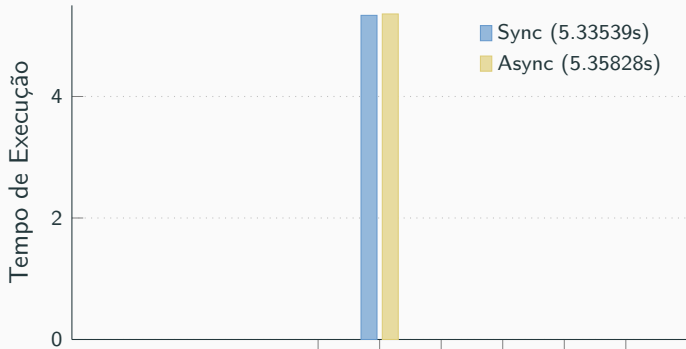
The screenshot shows a Visual Studio code editor window titled 'Application.cs - V:\Repositories\john\techtalk - Atom'. The editor displays a C# file named 'Application.cs' with the following code:

```
33 return,  
34  
35 var result = CoreBusiness.DoSomething();  
36 ResultService.SaveSomething(result);  
37 }  
38  
39  
40  
41 public T Method(){  
42     return MethodAsync().Wait();  
43 }  
44  
45  
46 }  
47 }  
48
```

The code is written in C# and shows a transformation from asynchronous to synchronous code. The first part (lines 33-37) shows a return statement followed by a call to `CoreBusiness.DoSomething()` and `ResultService.SaveSomething(result)`. The second part (lines 41-43) shows a synchronous method `Method()` that calls `MethodAsync().Wait()`. The code is formatted with line numbers on the left side of the editor window. The Visual Studio interface includes a menu bar at the top with 'File', 'Edit', 'View', 'Selection', 'Find', 'Packages', and 'Help'. A sidebar on the left shows a project tree with folders like 'Project', 'bin', 'obj', 'Files', and 'Sync'. The bottom status bar shows the file path 'V:\Repositories\john\techtalk - Atom' and the file name 'Application.cs'.

Sync x Async

Benchmark



Fui tapeado?

Por que não houve ganhos de performance?

Programação Assíncrona apenas é responsável por não bloquear a execução.

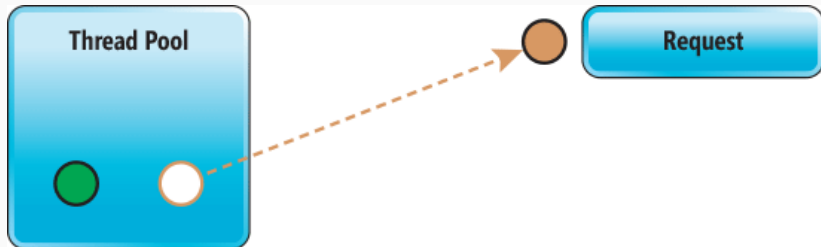
Então para que vou usar isso?

Depende da aplicação.

Vantagens da Programação Assíncrona

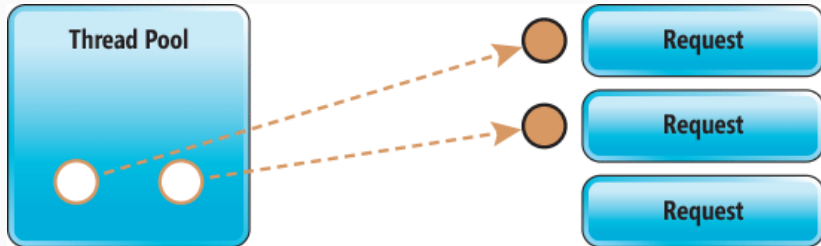
- **Em aplicações que possuem front-end:** Aprimoramento da interação do usuário com a aplicação;
- **Em um servidor:** Aprimoramento da escalabilidade;

Uma requisição síncrona no WebAPI



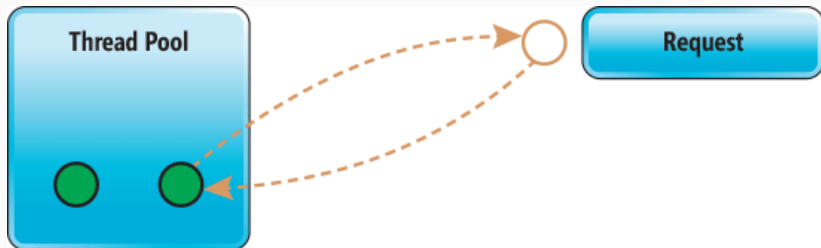
- Uma thread livre e outra thread bloqueada realizando **nada**;

Três requisições síncronas no WebAPI



- Duas threads bloqueadas realizando **nada** e uma requisição em espera;

Uma requisição assíncrona no WebAPI



- Thread realiza uma requisição bloqueante e retorna para a espera;

- Redução de custos em relação à memória;
- Redução do tempo de CPU desperdiçado;

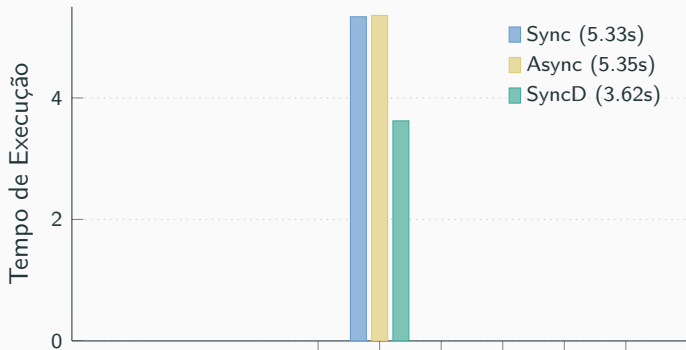
Paralelismo em C#

- **Paralelismo:** Duas tarefas são executadas simultaneamente.
 - Executadas em diferentes processadores;
 - Executadas em diferentes núcleos de um mesmo processador;
- **Concorrência:** Duas tarefas estão em progresso ao mesmo tempo;
 - Executadas de forma alternada em um mesmo processador;

- **Paralelismo de Dados:** A mesma instrução é executada em diferentes threads.
 - Ex: Enquanto uma thread itera sobre a parte inicial de uma lista, outra itera sobre a parte final;
- **Paralelismo de Tarefas:** Diferentes instruções são executadas em várias threads;
 - Ex: Enquanto uma thread valida um token, outra busca credenciais no banco;

SyncDataParallel

Benchmark



Problemas com Async

Posso aplicar isso em qualquer problema?

Depende da quantidade de transformação de dados envolvida.

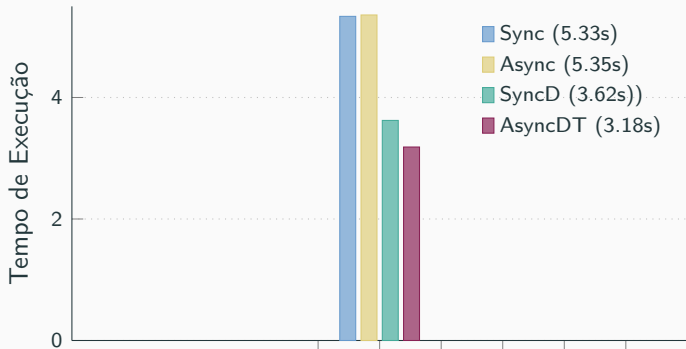
Quanto maior o número de threads melhor?

Existe um limiar onde a aplicação gasta mais tempo na comunicação entre threads do que o ganho de performance na execução paralela.

Programação Assíncrona e Paralela em C#

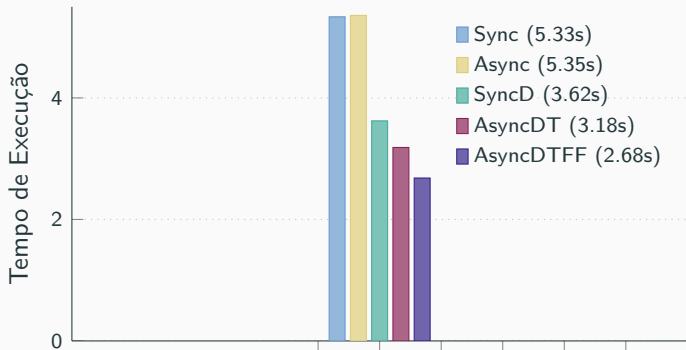
AsyncTaskAndDataParallel

Benchmark



AsyncTaskAndDataParallelFireAndForget

Benchmark



QueueWork != Async sem Await

Conclusões

- É possível criar métodos assíncronos e paralelos no WebAPI de forma simples e conveniente;
- Métodos assíncronos devem ser utilizados em APIs com finalidade de otimizar a utilização de recursos;
- Paralelismo nunca fornece um ganho linear de performance e deve ser utilizado em problemas onde a quantidade de transformações de dados é grande;

Perguntas?