DotNet 2021

ONLINE TECH CONFERENCE

#DotNet2021

22nd June 2021

.NET 6: Un salto más en rendimiento

www.dotnet2021.com

DotNet 2021

ORGANIZATION

plain concepts

IN COOPERATION WITH

FUNDACIÓN GOMAESPUMA

"Educando con una sonrisa."

SPONSORS

Microsoft

intelequia

Devs DNA™

My Public Inbox®

#DotNet2021

Daniel Cáceres

Technical Lead at Plain Concepts

Actualmente formo parte del equipo de Research en Plain Concepts, donde llevo los últimos años trabajando en el desarrollo de todo tipo de aplicaciones usando .NET y específicamente en nuestro motor 3D, Wave Engine. Desde siempre he disfrutado mucho comprendiendo como funcionan las cosas y el mundo del desarrollo software me da la oportunidad de no dejar nunca de aprender.

@danielcaceresm

dcaceres@plainconcepts.com

.NET: Muchos saltos de rendimiento

Desde que Microsoft inició el desarrollo de .Net Core, el equipo ha realizado un esfuerzo muy grande en mejorar el rendimiento del framework y en proporcionar herramientas para que los desarrolladores podamos mejorar significativamente el de nuestras aplicaciones.

Composite Framework Scores

Each framework's peak performance in each test type (shown in the colored columns below) is multiplied by the weights shown above. The results are then summed to yield a weighted score. Only frameworks that implement all test types are included. 122 total frameworks ranked, 119 visible, 3 hidden by filters. See filter panel above.

Rnk	Framework	JSON	1-query	20-query	Fortunes	Updates	Plaintext	Weighted score
1	lithium	1,599,157	833,811	63,415	659,850	34,887	6,998,356	**10,642** 100.0%
2	just	1,616,908	668,190	65,399	467,321	35,858	6,992,170	**9,696** 91.1%
3	drogon	1,070,061	650,753	58,922	666,737	33,377	6,519,621	**9,544** 89.7%
4	ntex	1,603,310	636,561	34,610	655,964	24,222	7,006,384	**8,933** 83.9%
5	actix	1,563,586	635,091	34,955	653,529	24,301	7,004,195	**8,894** 83.6%
6	may-minihttp	1,593,818	635,419	34,617	489,691	21,036	6,991,256	**8,024** 75.4%
7	wizzardo-http	1,548,467	631,584	31,936	290,654	16,678	7,016,349	**6,808** 64.0%
8	asp.net core	1,242,834	397,081	22,348	411,986	17,839	7,022,212	**6,462** 60.7%
9	jooby	1,297,219	548,113	31,840	423,234	16,348	4,031,131	**6,441** 60.5%
10	beetlex	1,148,005	405,715	23,308	371,228	18,213	4,947,208	**5,924** 55.7%

https://www.techempower.com/

.NET 5: Garbage Collector

Múltiples optimizaciones en la trazabilidad de los objetos.

Mejoras en el soporte de varios núcleos.

Implementaciones “GC pause friendly”.

Type	Runtime	Mean	Ratio
DoubleSorting	.NET FW 4.8	88.88 ns	1.00
DoubleSorting	.NET Core 3.1	73.29 ns	0.83
DoubleSorting	.NET 5.0	35.83 ns	0.40
Int32Sorting	.NET FW 4.8	66.34 ns	1.00
Int32Sorting	.NET Core 3.1	48.47 ns	0.73
Int32Sorting	.NET 5.0	31.07 ns	0.47
StringSorting	.NET FW 4.8	2,193.86 ns	1.00
StringSorting	.NET Core 3.1	1,713.11 ns	0.78
StringSorting	.NET 5.0	1,400.96 ns	0.64

https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-5/

.NET 5: JIT

Vectorización de “zeroing”.

SkipLocalsInit & Unsafe.SkipInit

Mejoras en la verificación de límites.

Method	Runtime	Mean	Ratio
Zeroing	.NET FW 4.8	22.85 ns	1.00
Zeroing	.NET Core 3.1	18.60 ns	0.81
Zeroing	.NET 5.0	15.07 ns	0.66

Method	Runtime	Mean	Ratio	Code Size
IsSorted	.NET FW 4.8	1,083.8 ns	1.00	236 B
IsSorted	.NET Core 3.1	581.2 ns	0.54	136 B
IsSorted	.NET 5.0	463.0 ns	0.43	105 B

Method	Runtime	Mean	Ratio	Code Size
BoundsChecking	.NET FW 4.8	14.466 ns	1.00	830 B
BoundsChecking	.NET Core 3.1	4.264 ns	0.29	320 B
BoundsChecking	.NET 5.0	3.641 ns	0.25	249 B

https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-5/

.NET 5: Text

Optimizaciones en char y string.

Mejoras muy notables en Regex.

Method	Runtime	Mean	Ratio
Email	.NET FW 4.8	1,036.729 ms	1.00
Email	.NET Core 3.1	930.238 ms	0.90
Email	.NET 5.0	50.911 ms	0.05
Uri	.NET FW 4.8	870.114 ms	1.00
Uri	.NET Core 3.1	759.079 ms	0.87
Uri	.NET 5.0	50.022 ms	0.06
IP	.NET FW 4.8	75.718 ms	1.00
IP	.NET Core 3.1	61.818 ms	0.82
IP	.NET 5.0	6.837 ms	0.09

Method	Runtime	Mean	Ratio
IsMatch	.NET FW 4.8	2,558.1 ns	1.00
IsMatch	.NET Core 3.1	789.3 ns	0.31
IsMatch	.NET 5.0	129.0 ns	0.05

Method	Runtime	Mean	Ratio	Code Size
Trim	.NET FW 4.8	21.694 ns	1.00	569 B
Trim	.NET Core 3.1	8.079 ns	0.37	377 B
Trim	.NET 5.0	6.556 ns	0.30	365 B

Method	Runtime	Mean	Ratio	Code Size
ToUpperInvariant	.NET FW 4.8	208.34 ns	1.00	171 B
ToUpperInvariant	.NET Core 3.1	166.10 ns	0.80	164 B
ToUpperInvariant	.NET 5.0	69.15 ns	0.33	105 B

https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-5/

.NET 5: Varios

Sockets

Intrinsics: Mejor soporte ARM64

Async ValueTask

Colleciones y LINQ

Method	Runtime	Mean	Ratio	Allocated
SendReceive	.NET Core 3.1	5.924 us	1.00	624 B
SendReceive	.NET 5.0	5.230 us	0.88	144 B

Method	Runtime	Mean	Ratio	Allocated
Get	.NET Core 3.1	1,267.4 ms	1.00	122.76 MB
Get	.NET 5.0	681.7 ms	0.54	74.01 MB
Post	.NET Core 3.1	1,464.7 ms	1.00	280.51 MB
Post	.NET 5.0	735.6 ms	0.50	132.52 MB

Method	Runtime	Mean	Ratio	Allocated
ValueTaskCost	.NET FW 4.8	1,635.6 us	1.00	294010 B
ValueTaskCost	.NET Core 3.1	842.7 us	0.51	120184 B
ValueTaskCost	.NET 5.0	812.3 us	0.50	186 B

Method	Runtime	Mean	Ratio	Allocated
SkipLast	.NET Core 3.1	1,641.0 ns	1.00	248 B
SkipLast	.NET 5.0	684.8 ns	0.42	48 B

https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-5/

.NET 6

Múltiples mejoras en arm64.

Optimizaciones al usar interfaces.

Muchas mejoras en JIT.

DateTime.UtcNow 2.5x más rápido que en .Net Core 3.1

Method	Implementation	Mean	Version	Ratio	Improvement
IsInterface1	Interfaces1	2.166 ns	.NET 6 Preview 2	1.000	
IsInterface1	Interfaces1	1.629 ns	.NET 6 Preview 3	**0.752**	x1.3
IsNotImplemented	Interfaces1	2.166 ns	.NET 6 Preview 2	1.000	
IsNotImplemented	Interfaces1	1.950 ns	.NET 6 Preview 3	**0.900**	x1.1

https://github.com/dotnet/core/issues/5853

.NET 6: FileStream

Implementación de Windows reescrita.

La mayoría de métodos Async acababan realizando llamadas síncronas.

Uso de ValueTask.

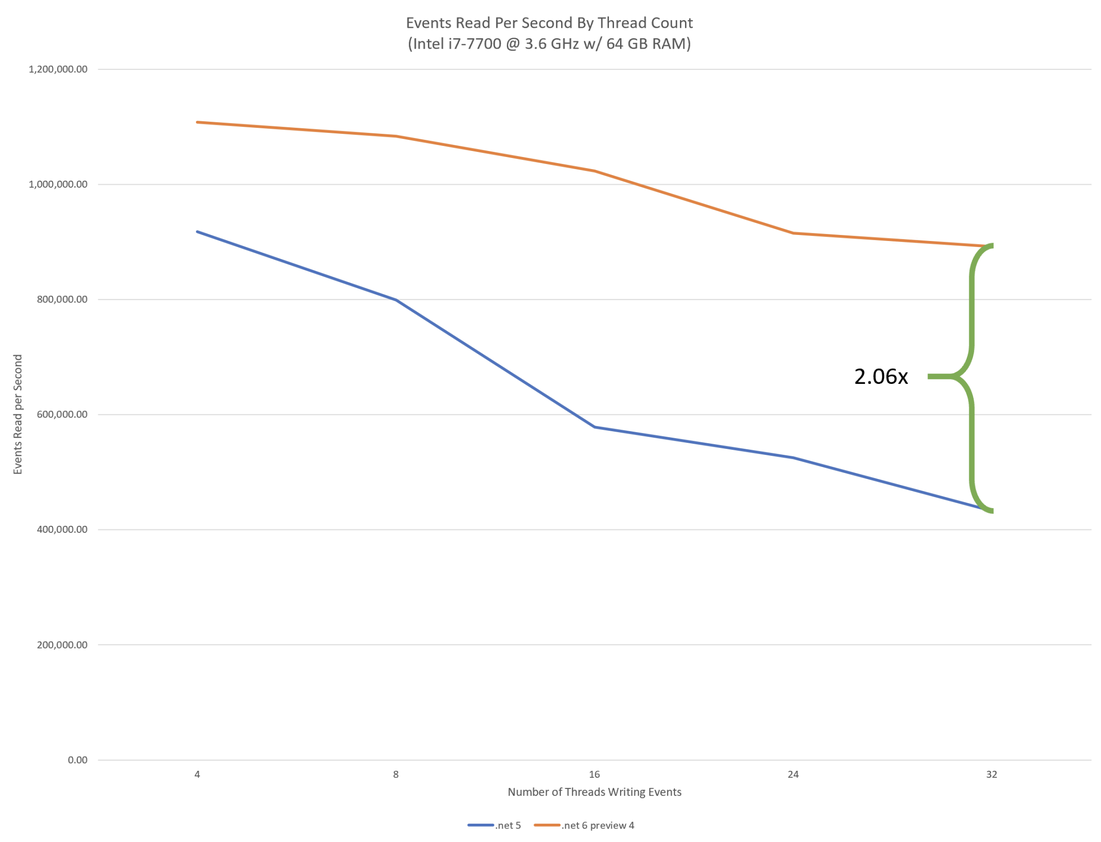
Method	Runtime	Mean	Ratio	Allocated
ReadAsync	.NET 5.0	3.419 ms	1.00	39,504 B
ReadAsync	.NET 6.0	1.445 ms	0.42	192 B
WriteAsync	.NET 5.0	12.181 ms	1.00	39,192 B
WriteAsync	.NET 6.0	2.193 ms	0.18	192 B

https://github.com/dotnet/core/issues/6098

.NET 6: EventPipe

CoreCLR y Mono usan la misma implementación.

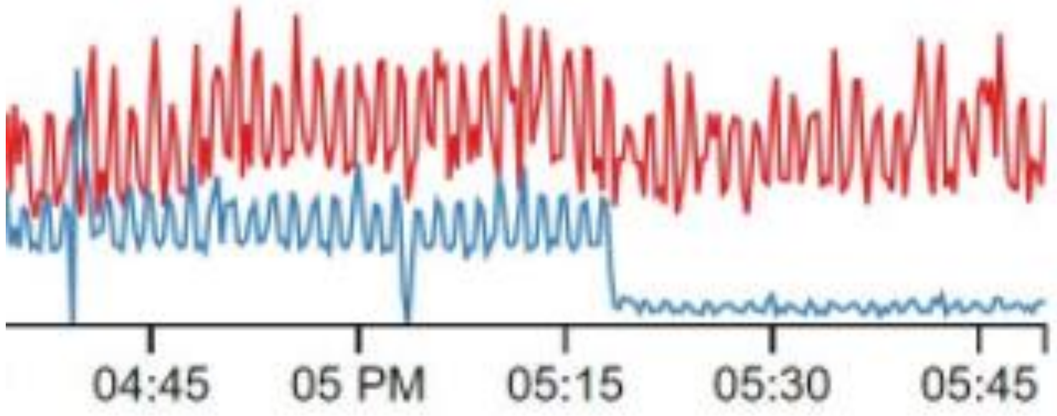
Múltiples optimizaciones.



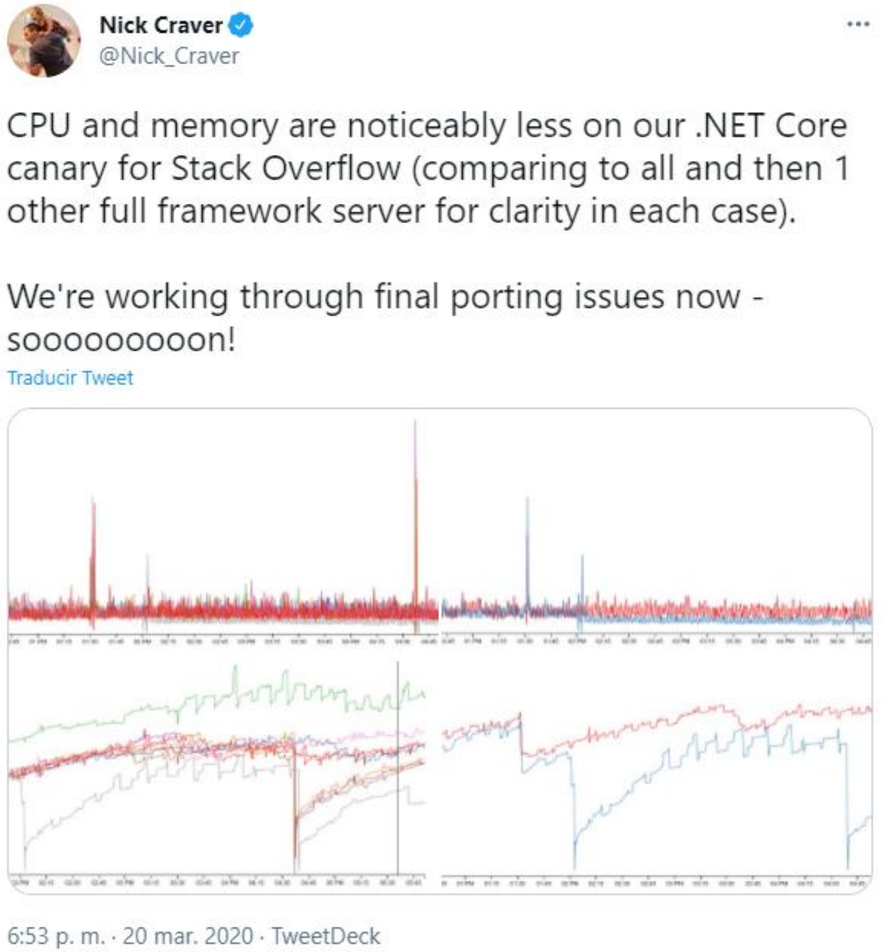
https://github.com/dotnet/core/issues/6098

Framework vs herramientas

Normalmente esperamos que actualizar la versión del framework veamos:



https://stackexchange.com/performance

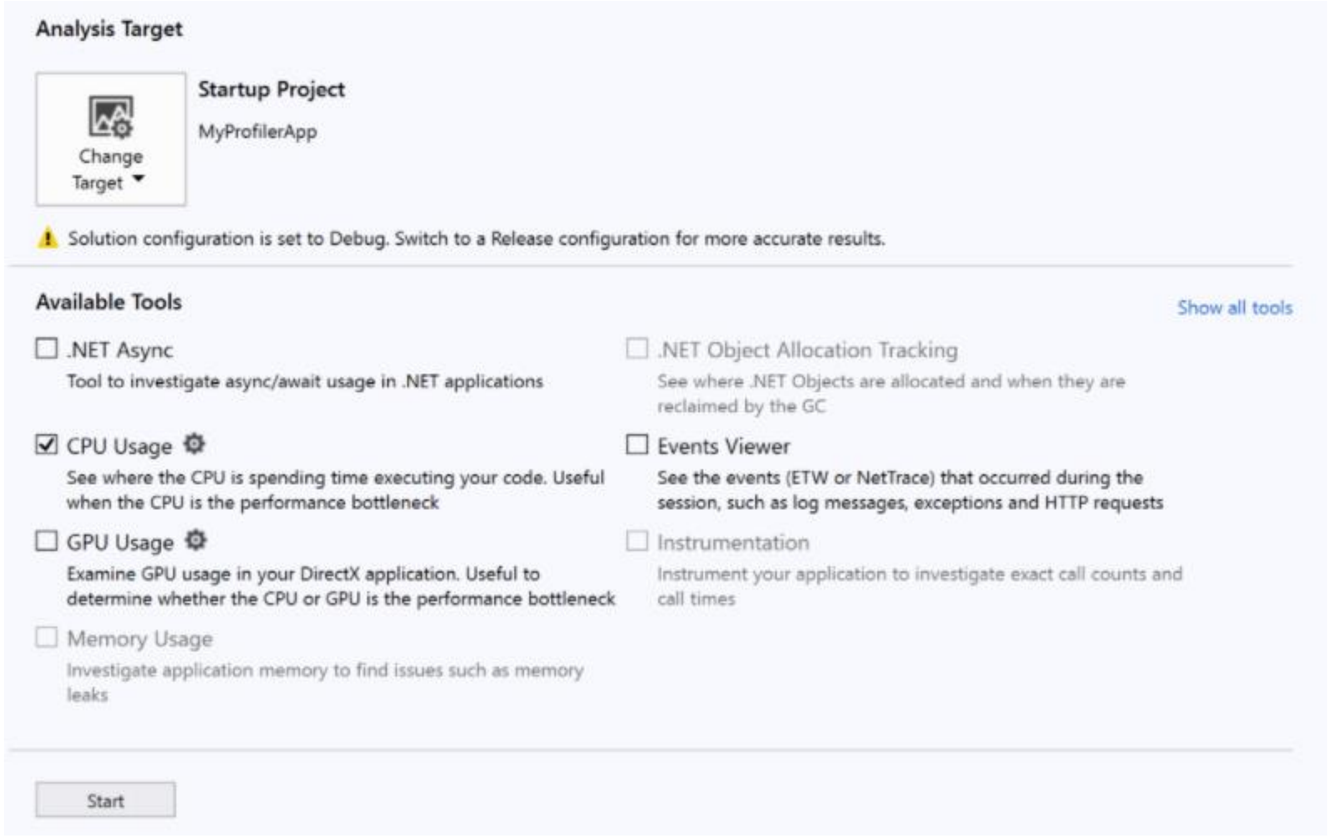


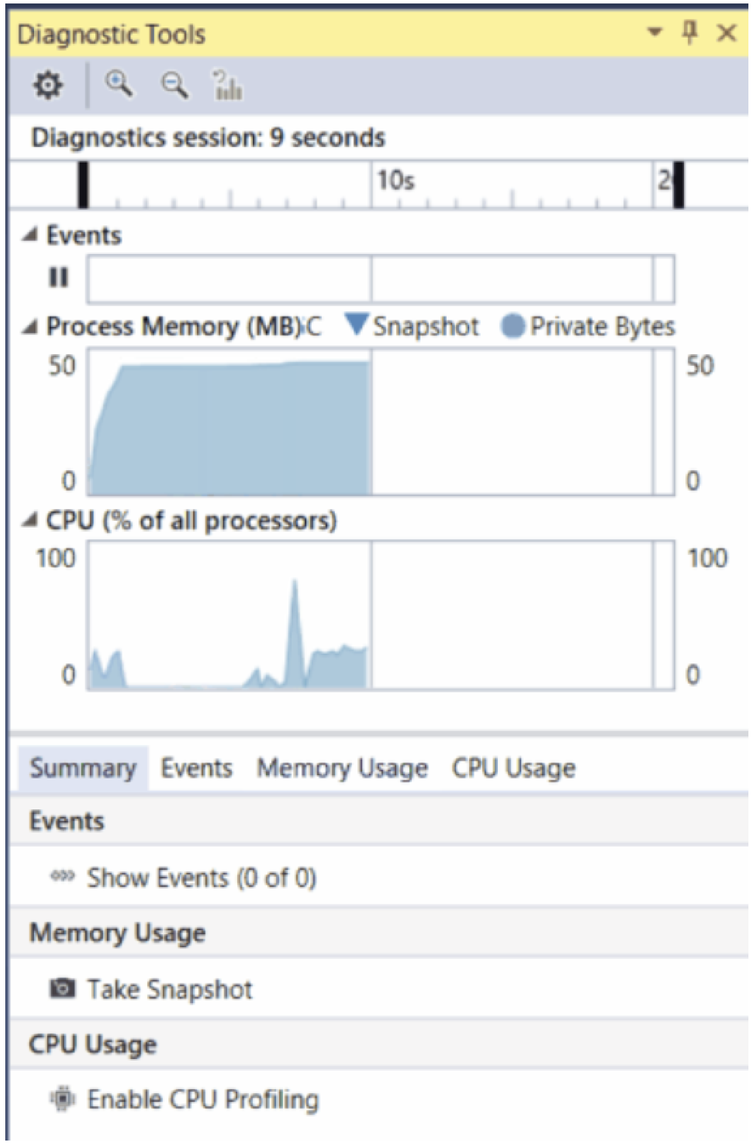
Framework vs herramientas

Herramientas que nos permitan:

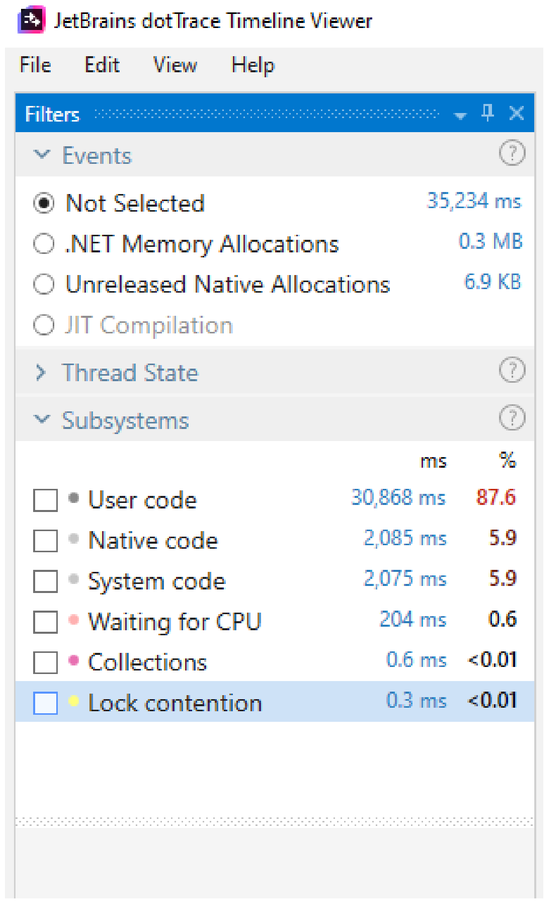
- Analizar problemas de rendimiento
- Implementar soluciones
  - Nuevas features como: Span, Source Generators
  - Nuevas APIS como: Intrinsics, Unsafe.SkipInit, CollectionsMarshal.GetValueRefOrNullRef
- Medir la mejora

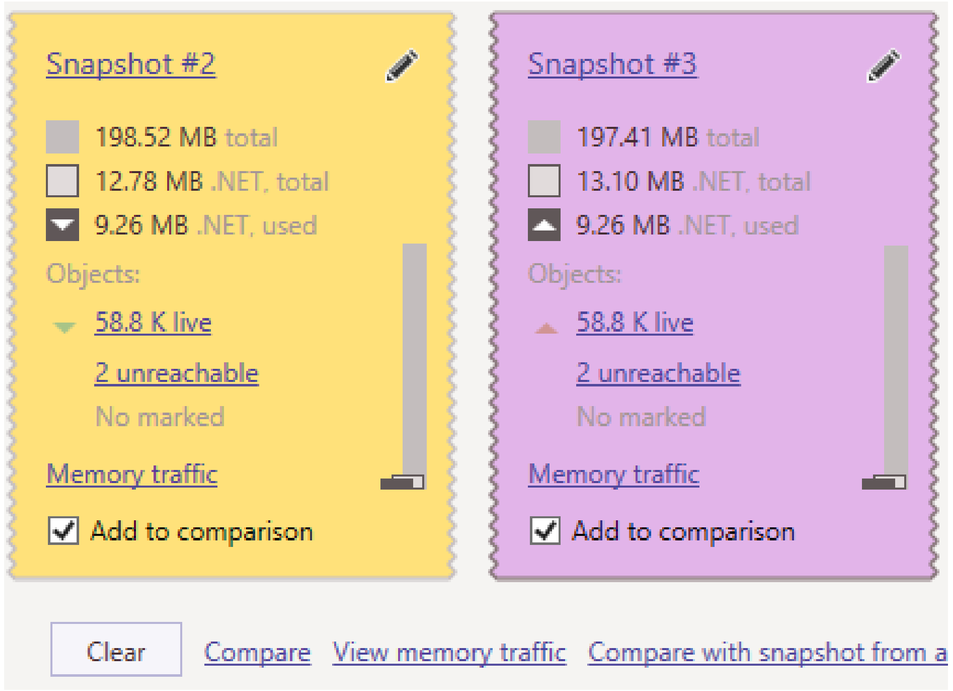
Visual Studio 2019



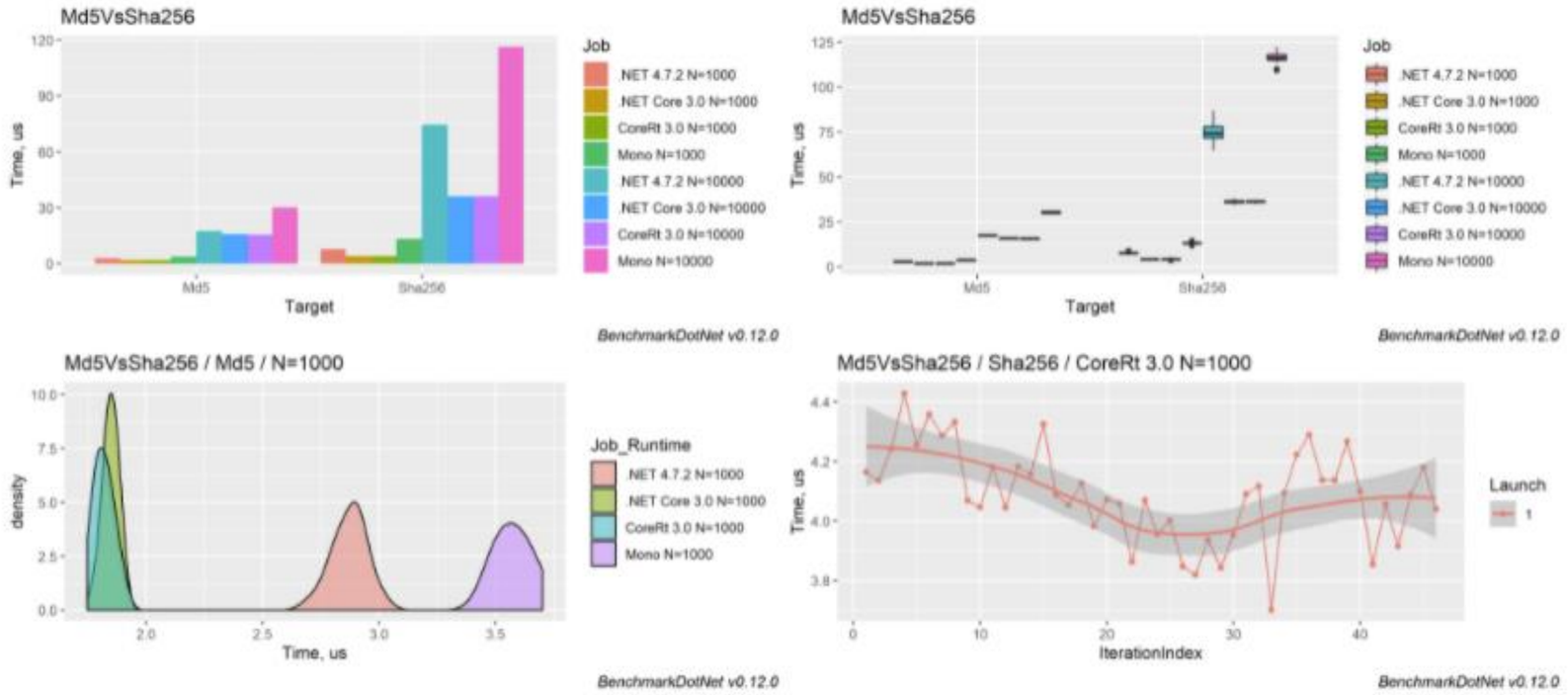


dotTrace y dotMemory





BenchmarkDotNet

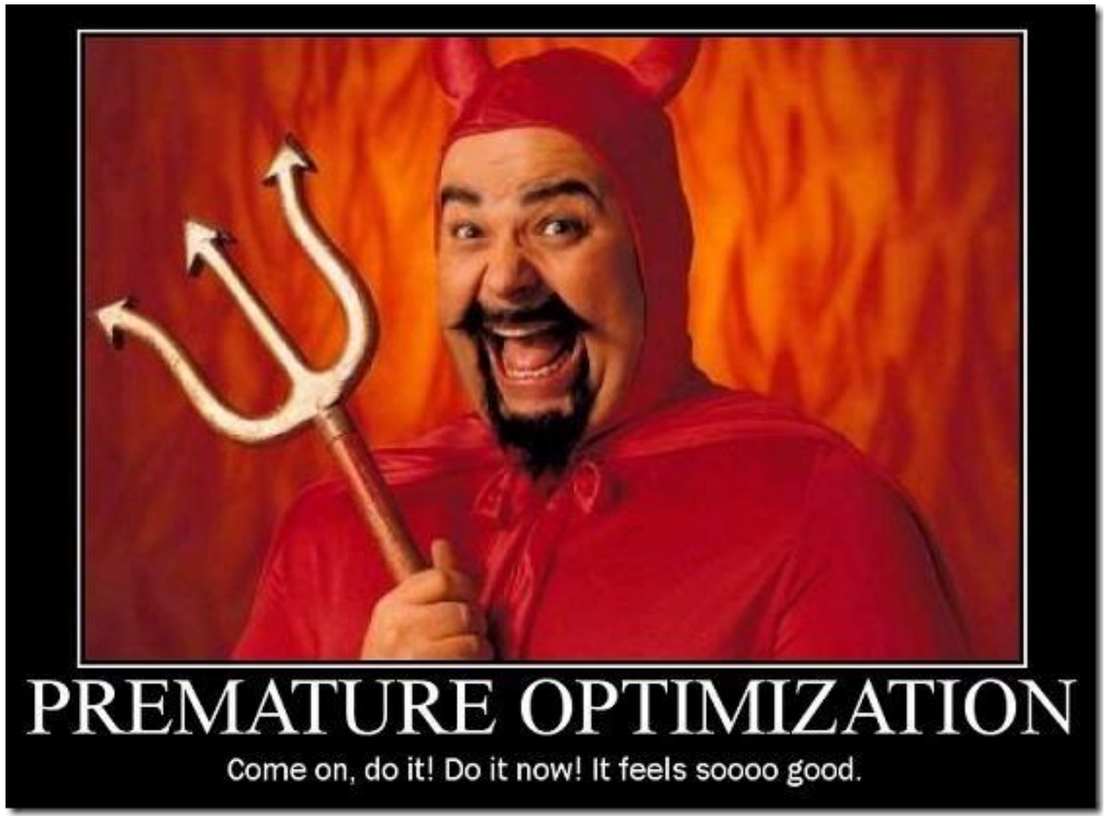


https://github.com/dotnet/BenchmarkDotNet

Wave Engine

- Framework
  - Minimizar uso de recursos -> Maximizar código de usuario
- Motor 3D
  - 60FPS -> 16,6ms
  - 300FPS -> 3,3ms

- Optimización prematura vs código “eficiente”
  - Conocer como funciona la plataforma.
  - Mantenibilidad del código.
  - ¿Merece la pena?



Wave Engine

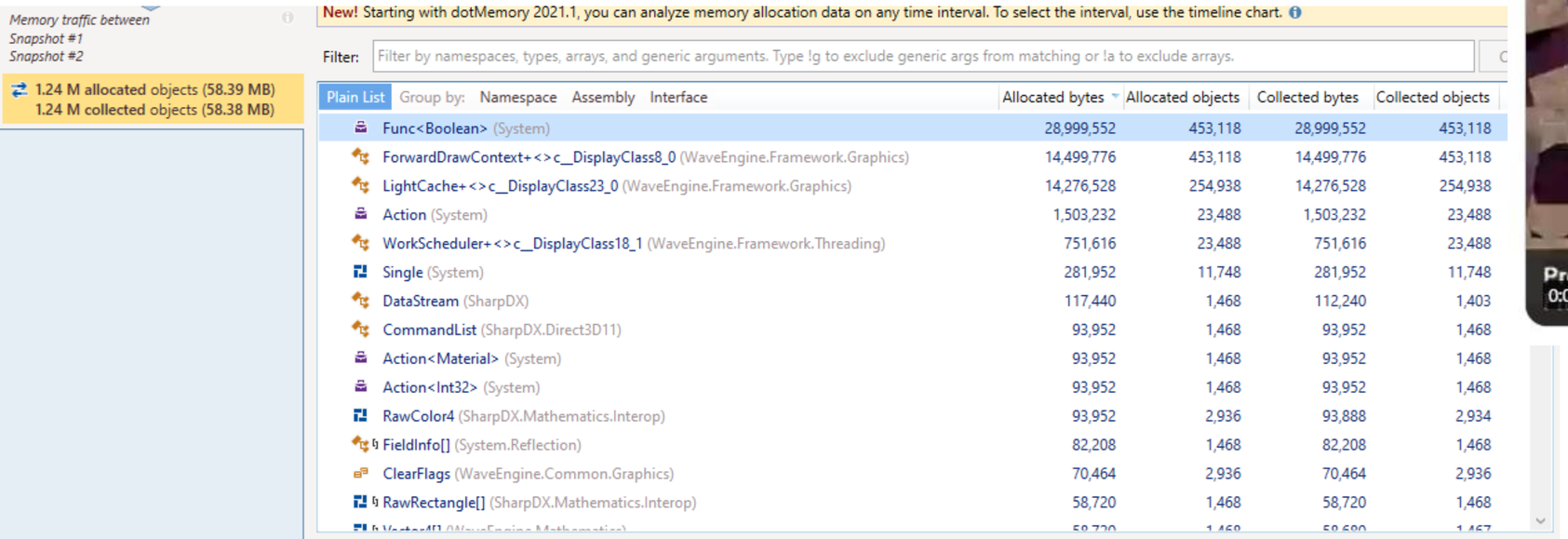
- Antes de cada release o merge de una feature importante
  - Tests donde probamos diferentes características.
  - Revisar memory leaks.
  - Tráfico de memoria.
  - Uso de CPU y uso de memoria con respecto a la release anterior.
- Al encontrar un problema específico
  - En un proyecto tenemos un problema de rendimiento.
  - Hacemos profile del proyecto.
  - Analizamos los datos, e intentamos corregirlo.

Wave Engine: Performance review

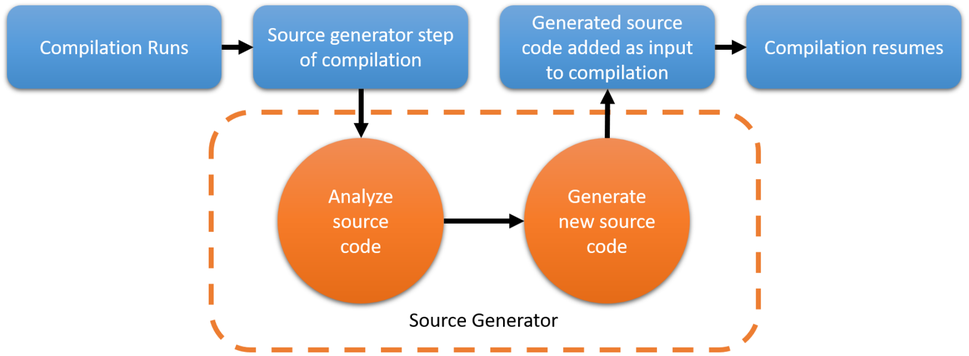
Mejora de FPS 1.3x

Problemas típicos:

- Tráfico de memoria:
  - Boxing structs, enums… (Structs como campos, Enum.HasFlag, etc.)
  - Scope de lambdas



Wave Engine: Source Generators




```
SceneEvent lightsScene LOAD: 158ms
SceneEvent digitalScene LOAD: 298ms
SceneEvent droneScene LOAD: 222ms
SceneEvent weatherScene LOAD: 167ms
YAML LOAD: 845ms
```

```
SceneEvent lightsScene LOAD: 1072ms
SceneEvent droneScene LOAD: 3162ms
SceneEvent weatherScene LOAD: 1726ms
SceneEvent digitalScene LOAD: 6429ms
YAML LOAD: 12389ms
```

```
SceneEvent lightsScene LOAD: 104ms
SceneEvent digitalScene LOAD: 240ms
SceneEvent droneScene LOAD: 103ms
SceneEvent weatherScene LOAD: 75ms
CODE LOAD: 522ms
```

```
SceneEvent lightsScene LOAD: 174ms
SceneEvent droneScene LOAD: 280ms
SceneEvent weatherScene LOAD: 199ms
SceneEvent digitalScene LOAD: 576ms
CODE LOAD: 1229ms
```

https://devblogs.microsoft.com/dotnet/introducing-c-source-generators/

Questions & Answers

@danielcaceresm

dcaceres@plainconcepts.com

DotNet 2021

ONLINE TECH CONFERENCE

www.dotnet2021.com

#DotNet2021

Thanks and ...
See you soon!

Thanks also to the sponsors. Without whom this would not have been posible.