

Native AOT vs JIT в .NET 10:

производительность, оптимизации
и подводные камни

DOT
NEXT

Юрий Малич
NP4 GmbH
yurymalich@yandex.ru

Сегодня в программе

- Виды компиляций JIT и Native AOT и их отличия
- Достоинства и недостатки каждого подхода
- Выбор поддерживаемых наборов команд при компиляции Native AOT в машинный код
- Особенности тестирования Native AOT в BenchmarkDotNet
- Сравнение производительности JIT vs Native AOT в BenchmarkDotNet в разных алгоритмах с разными наборами команд
- Выводы

О себе

- Закончил ПГУПС (Автоматика и телемеханика на ж/д транспорте)
- Программирую примерно с 1996 года, с MS DOS на Intel i386 (Pascal, C++, ASM)
- На C# .NET с первых версий. Microsoft Certified Professional 2016
- Писал статьи по архитектуре микропроцессоров для сайтов iXBT.com, fcenter.ru, 3dnews.ru
- Принимал участие в разработке системных утилит для Windows: в Nero AG (Nero Burning Rom) и в TuneUp (TuneUp Utilities)
- Разрабатываю Desktop и Backend приложения на C# .NET

Андрей Порожняков «Native AOT. Возможности и ограничения» DotNext 2024

Native AOT. Возможности и ограничения

Иногда .NET-приложения запускаются дольше, чем нужно. Это связано с особенностями JIT-компиляции — при каждом старте IL-код компилируется в машинные инструкции. Проблема становится заметной, когда приложения запускаются часто, например, при реализации FaaS.

Решением может стать ahead-of-time, или AOT-компиляция. Это одна из функций, над которой сейчас работает группа ASP.NET Microsoft. Она генерирует машинные инструкции на этапе сборки приложения, и можно обойтись без IL-кода. Но такой подход предъявляет ряд требований к приложению и разработке.

Расскажу, как ускорить запуск приложения за счет AOT-компиляции. Рассмотрим требования и ограничения публикации Native AOT. Отдельно скажу о поддержке Minimal API и генераторов исходного кода, о плюсах, минусах и практической пользе технологии.

Спикеры



Андрей Порожняков
SKAI

<https://dotnext.ru/archive/2024/talks/a24ece9645844bbea7fc1bf313b359ee/>

https://vk.com/video-144195266_456239144

JIT (Just-In-Time) компиляция

- Исходный код (C#) компилируется в промежуточный язык (IL), который сохраняется в сборке
- При запуске приложения JIT-компилятор транслирует IL в машинный код в момент первого исполнения каждого метода

JIT (Just-In-Time) компиляция

- + Кроссплатформенность
- + Оптимизация под конкретный процессор
- + PGO-оптимизации горячих методов
- + Полная поддержка рефлексии и динамики
- + Поддерживаются все типы C#-проектов
- Медленный "холодный" старт (first-hit latency)
- Требуется установка .NET Runtime на целевой машине (или self-contained)

NativeAOT (Ahead-Of-Time) компиляция

- Компиляция в машинный код происходит один раз во время публикации проекта
- Весь код (**включая рантайм**) преобразуется в двоичный машинный код для **заданных архитектуры процессора (x64, ARM64, RISC etc) и ОС (WinAPI, Linux API etc)**

NativeAOT (Ahead-Of-Time) компиляция

- + Обеспечивает очень быстрый запуск
- + Не требуется рантайм (всё включено и откомпилировано)
- + Оптимизации не ограничены по глубине и времени компиляции
- Сборка привязана к архитектуре процессора и ОС на уровне API
- PGO очень сложны в применении
- Поддержка рефлексии ограничена триммингом
- Не работают или работают с ограничениями многие типы проектов

NativeAOT (Ahead-Of-Time) компиляция

Полностью поддерживают Native AOT

Тип проекта / Приложение	Уровень поддержки	Примечания
Console tools / CLI-утилиты	Отличная	Лучший сценарий для AOT
Minimal APIs (ASP.NET Core)	Отличная	Специальный шаблон webapi.aot
gRPC services	Хорошая	Полностью поддерживается
Worker Service / Background Service	Хорошая	Отлично подходит
Blazor WebAssembly	Хорошая	Значительный прирост производительности
Dapper + ADO.NET / SQLite	Хорошая	Рекомендуется вместо EF Core

NativeAOT (Ahead-Of-Time) компиляция

Ограниченно поддерживают (работает с оговорками)

Тип проекта / Приложение	Уровень поддержки	Основные ограничения
.NET MAUI (Windows)	Экспериментальная / Частичная	Поддержка улучшается, но пока не стабильна на всех платформах
Windows App SDK / WinUI 3	Хорошая	Поддерживается лучше, чем классический WPF/WinForms
Blazor Hybrid (в MAUI)	Ограниченная	Работает нестабильно

NativeAOT (Ahead-Of-Time) компиляция

Не поддерживают или поддерживают очень плохо

Тип проекта / Приложение	Статус	Причина
ASP.NET Core MVC / Razor Pages	Не поддерживается	Сильная зависимость от runtime reflection
SignalR	Не поддерживается	—
Blazor Server	Не поддерживается	—
Entity Framework Core (EF Core)	Не рекомендуется	Microsoft официально не советует использовать в production с AOT
WPF	Плохо / Не поддерживается	Сильная зависимость от reflection
Windows Forms (WinForms)	Плохо / Работает с ограничениями	Сильная зависимость от reflection при загрузке ресурсов
C++/CLI проекты	Не поддерживается	Полностью запрещено
Runtime code generation (System.Reflection.Emit)	Не поддерживается	Запрещено в AOT
Динамическая загрузка сборок (Assembly.Load)	Не поддерживается	Запрещено

ReadyToRun компиляция

- Во время публикации проекта C# компилируется в промежуточный язык (IL) и прекомпилируется в машинный код (**не все методы**)
- В сборке сохраняется как IL, так и **машинный код для заданных архитектуры (x64, ARM64, RISC etc) и ОС (WinAPI, Linux API etc)**

ReadyToRun компиляция

- + Обеспечивает более быстрый запуск
- + JIT может оптимизировать код под конкретный процессор во время исполнения
- + Полная поддержка рефлексии и динамики
- + Поддерживаются все типы C# проектов
- + Кроссплатформенность
- Требуется установка .NET Runtime на целевой машине (или self-contained)
- Размер пакета сборки увеличивается в 2-3 раза

NativeAOT компиляция

```
<PropertyGroup>  
  <OutputType>Exe</OutputType>  
  <TargetFramework>net10.0</TargetFramework>
```

```
  <PublishAot>true</PublishAot>  
  <IsAotCompatible>true</IsAotCompatible>
```

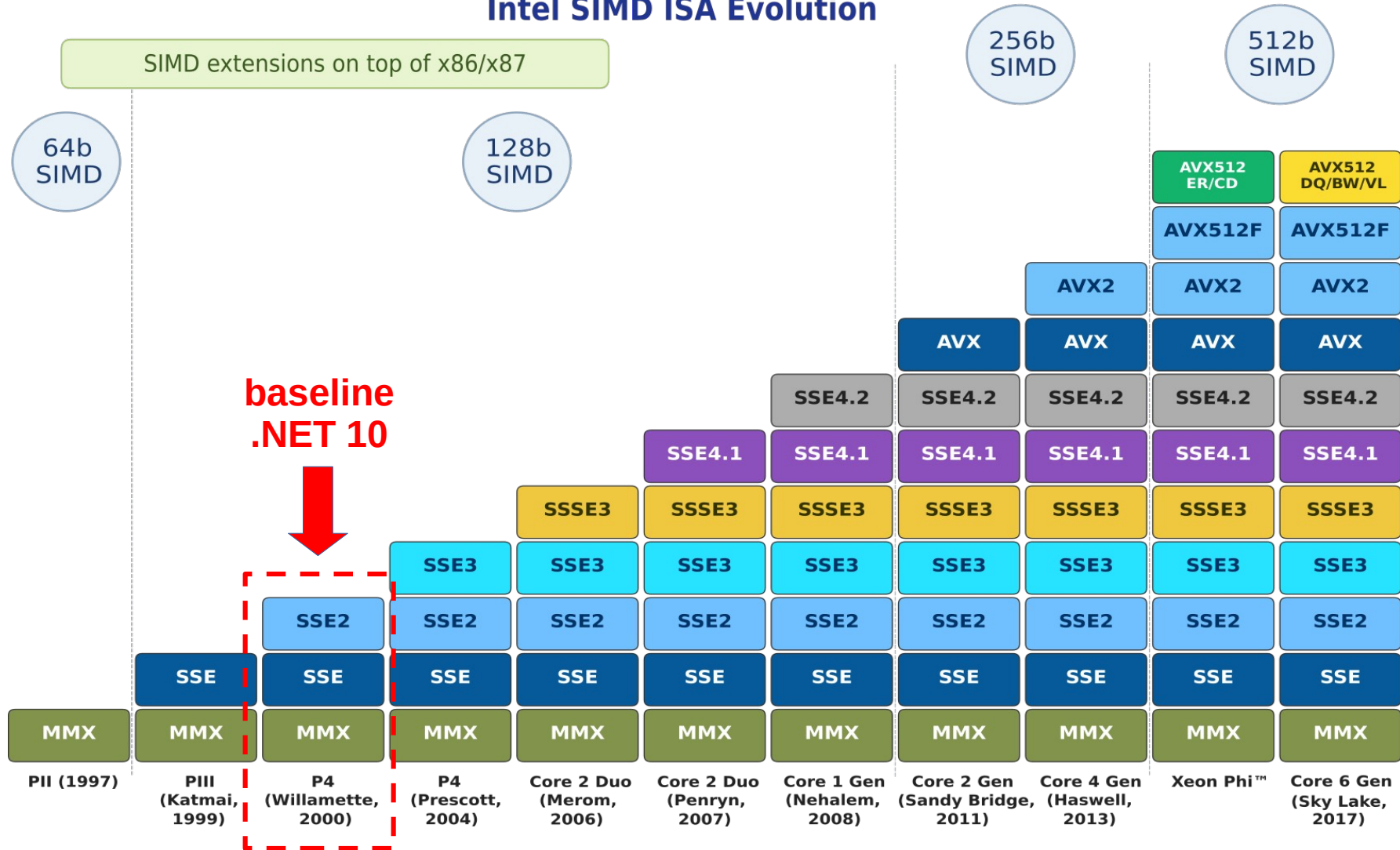
```
</PropertyGroup>
```

```
> dotnet publish -c Release -r win-x64 --self-contained /p:PublishAot=true
```

NativeAOT компиляция

А какой набор команд использует компилятор для x86-64?

Intel SIMD ISA Evolution



NativeAOT компиляция <IlcInstructionSet>

```
<PropertyGroup>
```

```
<PublishAot>true</PublishAot>
```

```
<IlcInstructionSet>avx2</IlcInstructionSet>
```

```
<!--<IlcInstructionSet>native</IlcInstructionSet>-->
```

```
</PropertyGroup>
```

Disclaimer:

-instruction-set options were explicitly specified to be undocumented and subject to change without notice

<https://github.com/dotnet/runtime/issues/119353>

<InstructionSet> в .NET 10 для x86-64

Значение	Наборы команд	Уровень микро-архитектуры	Векторные регистры	Поколение процессоров
base (by default)	Базовый x64 (SSE2)	x86-64-v1	128 bit	Базовый для всех x86-64 CPUs (2000)
sse4.2	SSE (1, 2, 3, 4.1, 4.2) + POPCNT	x86-64-v2	128 bit	Intel Core i* 1 Gen: Intel Nehalem (2008) , AMD Bulldozer (2011)
avx	AVX + предыдущие		128 bit по умол. *256 bit с vectort256	Intel Core i* 2 Gen: Intel Sandy Bridge (2011) AMD Bulldozer (2011)
avx2	AVX2 + BMI1 + BMI2 + FMA + LZCNT + MOVBE + F16C	x86-64-v3	256 bit	Intel Core i* 4 Gen (Intel Haswell 2013) AMD Excavator (2015)
avx512	AVX512 (F, CD, BW, DQ, VL)	x86-64-v4	256 bit по умол. *512 bit с vectort512	Intel Core i* 6 Gen (Skylake-X 2017) server , Core i* 10 Gen (Rocket Lake 2021) desktop AMD Zen 4 (2022)
native	Инструкции текущего CPU (как у JIT)			Данный CPU

А что, если задать неверно?

```
<PropertyGroup>
```

```
  <PublishAot>true</PublishAot>
```

```
  <IlcInstructionSet>avx512</IlcInstructionSet>
```

```
</PropertyGroup>
```

```
public static void Main(string[] args)
{
    Console.WriteLine("Hello, World!");
}
```

А что, если задать неверно?

```
The current CPU is missing one or more of the required instruction sets.
```

.NET 11 breaking change

Minimum hardware requirements updated

<https://learn.microsoft.com/en-us/dotnet/core/compatibility/jit/11/minimum-hardware-requirements>

OS	Previous JIT/AOT minimum	New JIT/AOT minimum (.NET 11)	Previous R2R target	New R2R target (.NET 11)
Apple	x86-64-v1	x86-64-v2 (SSE4.2)	x86-64-v2 (SSE4.2)	(No change)
Linux	x86-64-v1	x86-64-v2 (SSE4.2)	x86-64-v2 (SSE4.2)	x86-64-v3 (AVX2)
Windows	x86-64-v1	x86-64-v2 (SSE4.2)	x86-64-v2 (SSE4.2)	x86-64-v3 (AVX2)

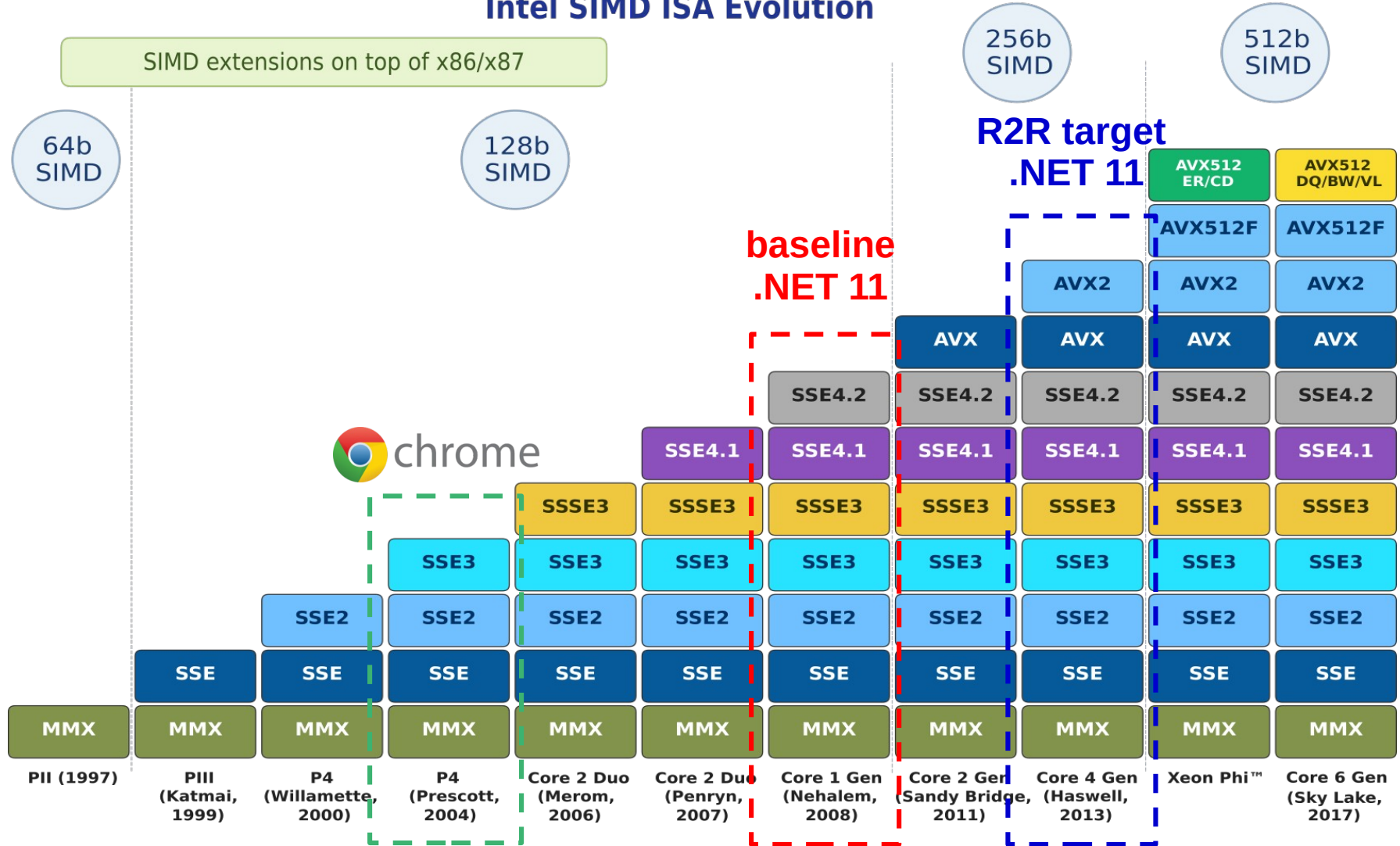


.NET 11 breaking change

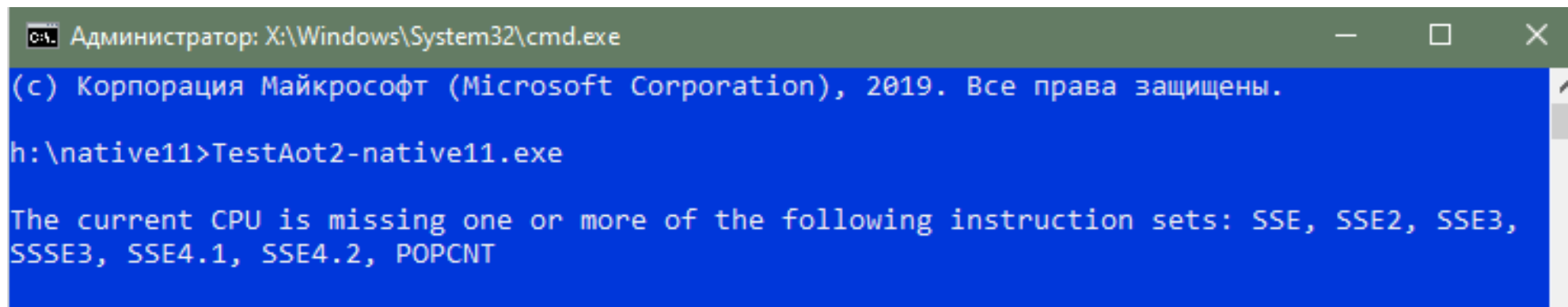
- Лучшая производительность Native AOT по умолчанию
- Упрощение кодовой базы, снижение сложности поддержки
- «Выравнивание» с требованиями ОС (Windows 11)



Intel SIMD ISA Evolution



.NET 11 breaking change



A screenshot of a Windows command prompt window. The title bar is green and contains the text "Администратор: X:\Windows\System32\cmd.exe". The main area has a blue background with white text. The text in the window is as follows:

```
(c) Корпорация Майкрософт (Microsoft Corporation), 2019. Все права защищены.  
h:\native11>TestAot2-native11.exe  
  
The current CPU is missing one or more of the following instruction sets: SSE, SSE2, SSE3,  
SSSE3, SSE4.1, SSE4.2, POPCNT
```

<InstructionSet> в .NET 11 для x86-64

Значение	Наборы команд	Уровень микро-архитектуры	Векторные регистры	Поколение процессоров
base (by default)	SSE (1, 2, 3, 4.1, 4.2) + POPCNT	x86-64-v2	128 bit	Intel Core i* 1 Gen: Intel Nehalem (2008) , AMD Bulldozer (2011)
avx	AVX + предыдущие		128 bit по умол. *256 bit с vectort256	Intel Core i* 2 Gen (Intel Sandy Bridge 2011) AMD Bulldozer+ (2011)
avx2	AVX2 + BMI1 + BMI2 + FMA + LZCNT + MOVBE + F16C	x86-64-v3	256 bit	Intel Core i* 4 Gen (Intel Haswell 2013) AMD Excavator+ (2015)
avx512	AVX512 (F, CD, BW, DQ, VL)	x86-64-v4	256 bit по умол. *512 bit с vectort512	Intel Core i* 6 Gen (Skylake-X 2017) server /Core i* 10 Gen (Rocket Lake 2021) desktop AMD Zen 4 (2022)
native	Инструкции текущего CPU (как у JIT)			Данный CPU

<IcInstructionSet> .NET 11 для x86-64

Значение	Наборы команд	Векторные регистры	Поколение процессоров
avx512v2	avx512 +IFMA+VBMI	256 bit по умол. *512 bit с vectort512	Intel Core i* 9 Gen (Ice Lake 2019) server /Core i* 10 Gen (Rocket Lake 2021) desktop AMD Zen 4 (2022)
avx512v3	avx512v2 + BITALG+VBMI2+VNNI+VPOPCNTDQ	256 bit по умол. *512 bit с vectort512	Intel Core i* 10 Gen (Ice Lake 2019) AMD Zen 4 (2022)
avx10v1		?	?
avx10v2		?	?

<IlcInstructionSet>

<https://github.com/dotnet/runtime/blob/main/src/coreclr/nativeaot/docs/optimizing.md>

Optimizing programs targeting Native AOT



<IlcInstructionSet>

```
> cd %USERPROFILE%\nuget\packages\runtime.win-x64.microsoft.dotnet.ilcompiler\10.0.10\tools  
> ilc --help
```

x64: base, sse4.2, avx, avx2, avx512, avx512v2, avx512v3, avx10v1, avx10v2, apx, aes, aes_v256, aes_v512, avx512vp2intersect, avxifma, avxvnni, gfni, gfni_v256, gfni_v512, sha, waitpkg, x86serialize, vectort128, vectort256, vectort512, avxvnniint, avxvnniint_v51

arm64: base, neon, aes, crc, dotprod, rdma, sha1, sha2, lse, rcpc, vectort128, rcpc2, sve, sve2

The following CPU names are predefined groups of instruction sets and can be used in --instruction-set too:

x86-64-v2, x86-64-v3, x86-64-v4, armv8-a, armv8.1-a, armv8.2-a, armv8.3-a, armv8.4-a, armv8.5-a, armv8.6-a, **apple-m1**

Тесты в BenchmarkDotNet JIT vs AOT

BenchmarkDotNet

- JIT vs AOT (x64):
 - **base** (NET10 SSE2)
 - **sse4.2** (NET11 **base**)
 - **avx2**
 - **avx512** (выборочно)
- JIT vs AOT (ARM64):
 - **base**
 - **native**

BenchmarkDotNet

- AMD Ryzen 5 5600U (**AVX2**), турбо-буст отключен, частота ограничена 2300 МГц
- AMD RYZEN AI MAX+ PRO 395 (**AVX-512**) турбо-буст 5,1 ГГц
- Apple M2 (**ARM64**)

BenchmarkDotNet

- Поиск в строках
- Regex
- Linq
- Сериализация и десериализация Json и XML
- Скорость запуска приложений с разными настройками

BenchmarkDotNet

```
<PropertyGroup>  
  <OutputType>Exe</OutputType>  
  <TargetFrameworks>net10.0</TargetFrameworks>  
  <AllowUnsafeBlocks>>true</AllowUnsafeBlocks>  
  <PublishAot>>true</PublishAot>  
  
</PropertyGroup>
```

BenchmarkDotNet

```
[SimpleJob(RuntimeMoniker.Net10_0)]  
[SimpleJob(RuntimeMoniker.NativeAot10_0)]  
public class AOT  
{  
    [Benchmark]  
    public int DoWork()  
    {  
    }  
}
```

Под какой набор команд компилируется этот тест?

IlcInstructionSet = native (!)

BenchmarkDotNet

```
public static void Main(string[] args)

// JIT job
var jobNet10Jit = NewJob
    .WithRuntime(CoreRuntime.Core10_0)
    .WithId("NET 10.0 RyuJIT")
    .AsBaseline();

var config = DefaultConfig.Instance.AddJob(jobNet10Jit);

// AOT «base» toolchain
var aotToolchainBase = NativeAotToolchain.CreateBuilder()
    .UseNuGet()
    .IlcInstructionSet("base")
    .DisplayName("AOT base")
    .TargetFrameworkMoniker("net10.0")
    .ToToolchain();

// AOT «base» toolchain
var jobAotBase = NewJob.WithToolchain(aotToolchainBase)
    .WithId("AOT base")
    .WithRuntime(NativeAotRuntime.Net10_0);

config.AddJob(jobAotBase);
```

BenchmarkDotNet

```
// AOT «avx2» toolchain
var aotToolchainAvx2 = NativeAotToolchain.CreateBuilder()
    .UseNuGet()
    .IlcInstructionSet("avx2")
    .DisplayName("AOT avx2")
    .TargetFrameworkMoniker("net10.0")
    .ToToolchain();

// AOT «avx2» toolchain
var jobAotNative = NewJob.WithToolchain(aotToolchainAvx2)
    .WithId("AOT avx2")
    .WithRuntime(NativeAotRuntime.Net10_0);

config.AddJob(jobAotAvx2);

// Run tests
BenchmarkRunner.Run<TestClass>(config);
```

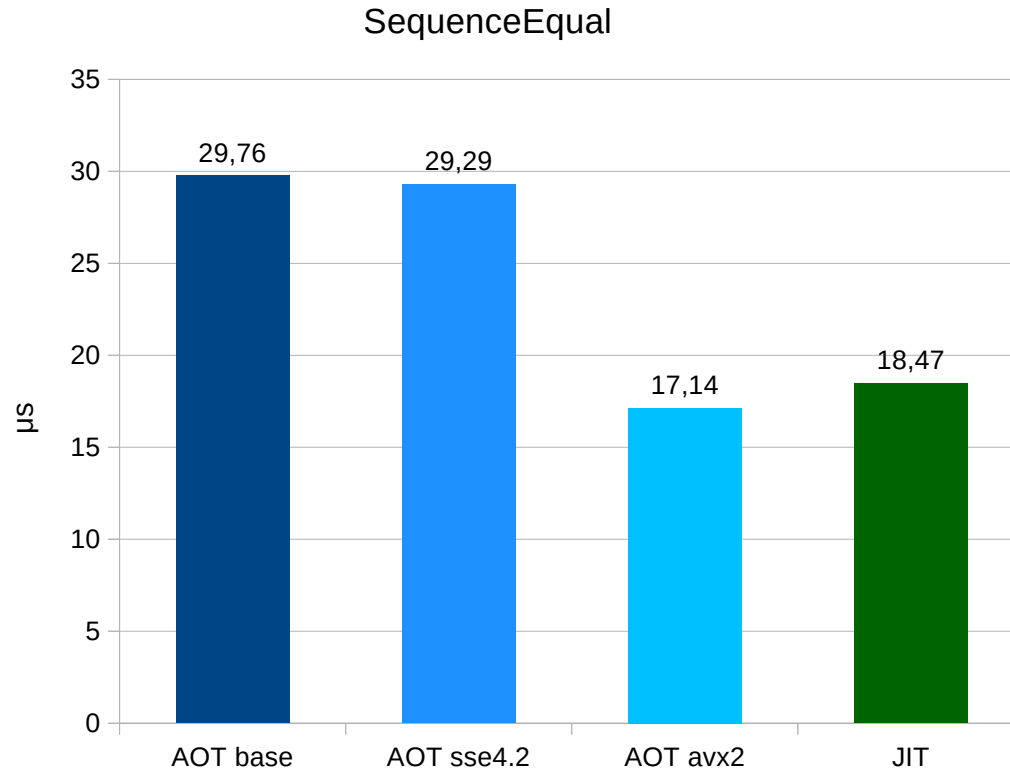
MemoryExtensions.SequenceEqual()

```
[Params(500 * 1024)]
public int N;

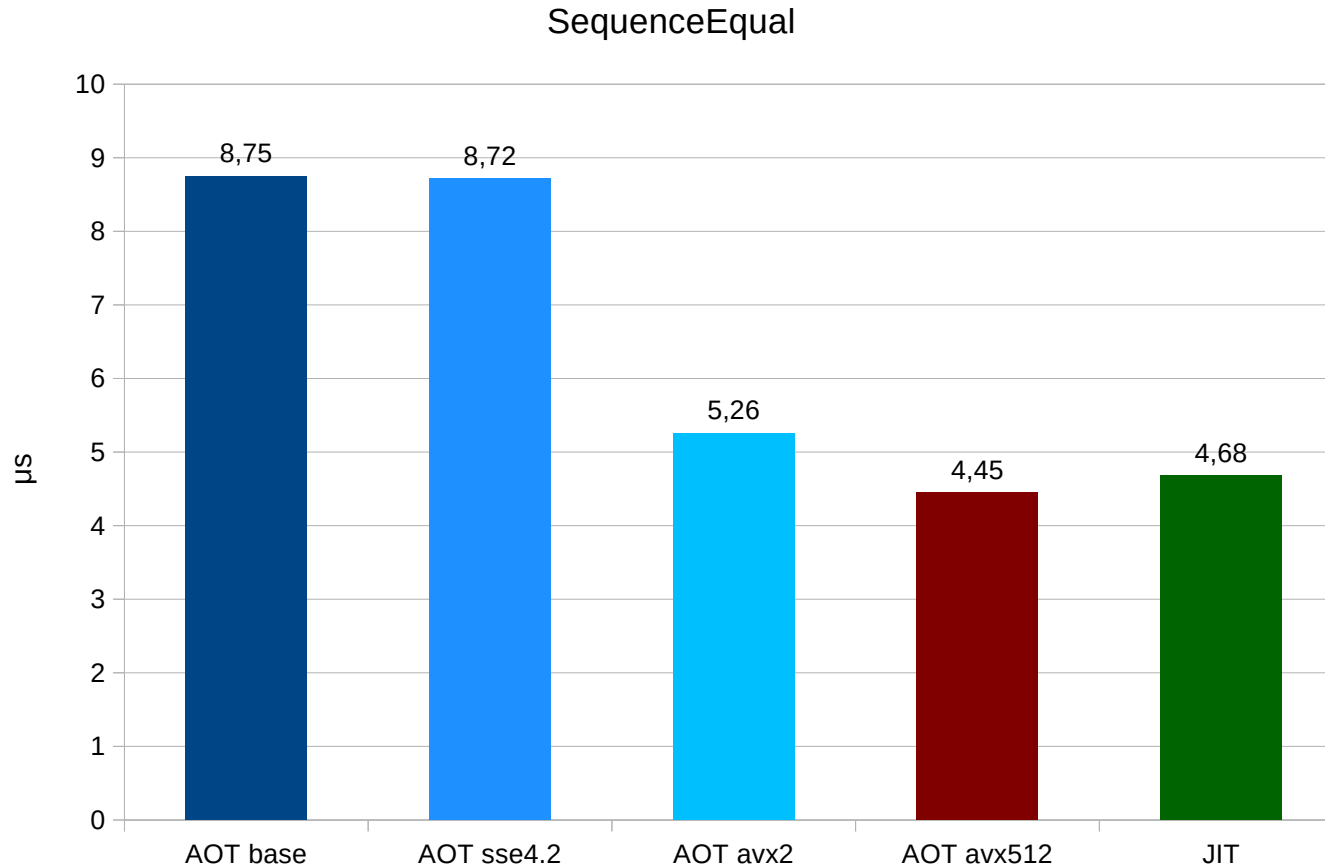
[GlobalSetup]
public void Setup()
{
    _buffer1 = new byte[N];
    _buffer2 = new byte[N];
}

[Benchmark]
public void SequenceEqual()
{
    var buf1Span = _buffer1.AsSpan();
    var buf2Span = _buffer2.AsSpan();
    B = buf1Span.SequenceEqual(buf2Span);
}
```

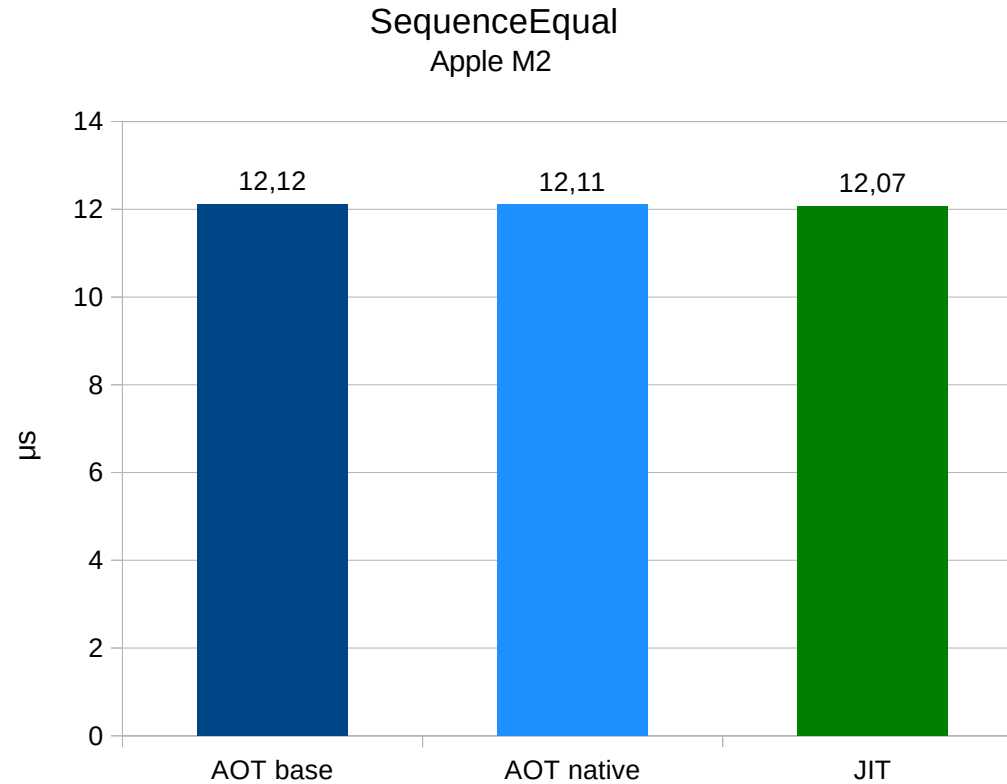
MemoryExtensions.SequenceEqual()



MemoryExtensions.SequenceEqual()



MemoryExtensions.SequenceEqual()



string.IndexOf()

```
[Params("unboxed")]
public string StringToSearch = string.Empty;

[GlobalSetup]
public void Setup()
{
    // 558 bytes string
    StringToTest = "The object type is an alias for System.Object in .NET. ... unboxed."
}

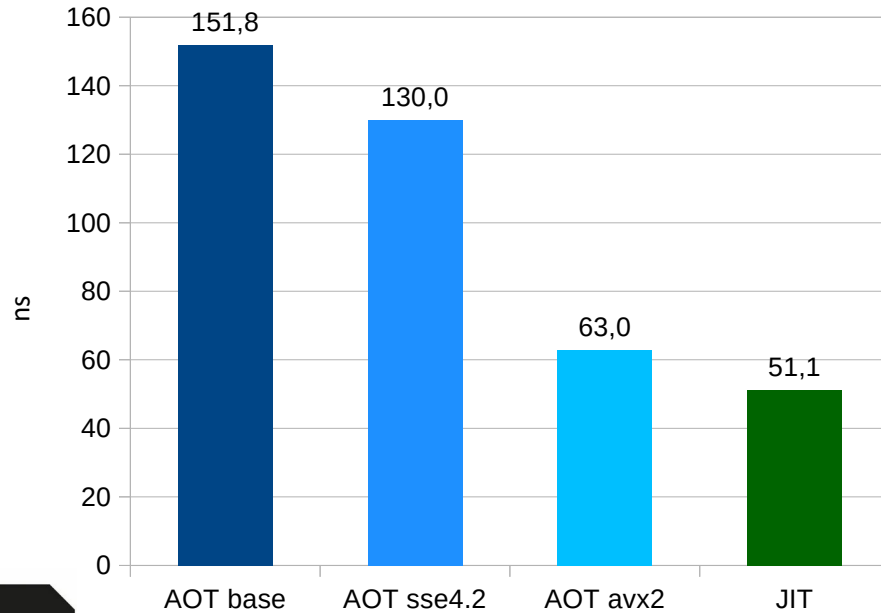
[Benchmark]
public void StringIndexOfOrdinal()
{
    res = StringToTest.IndexOf(StringToSearch, StringComparison.Ordinal);
}

[Benchmark]
public void StringIndexOfOrdinalIgnoreCase()
{
    res = StringToTest.IndexOf(StringToSearch, StringComparison.OrdinalIgnoreCase);
}
```

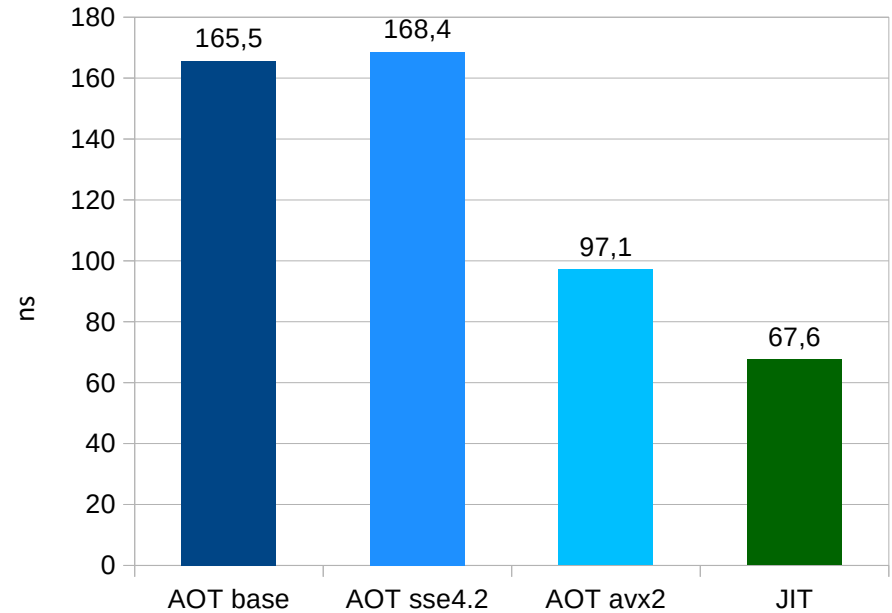
String.IndexOf()

[Ordinal, OrdinalIgnoreCase]

IndexOf(Ordinal)

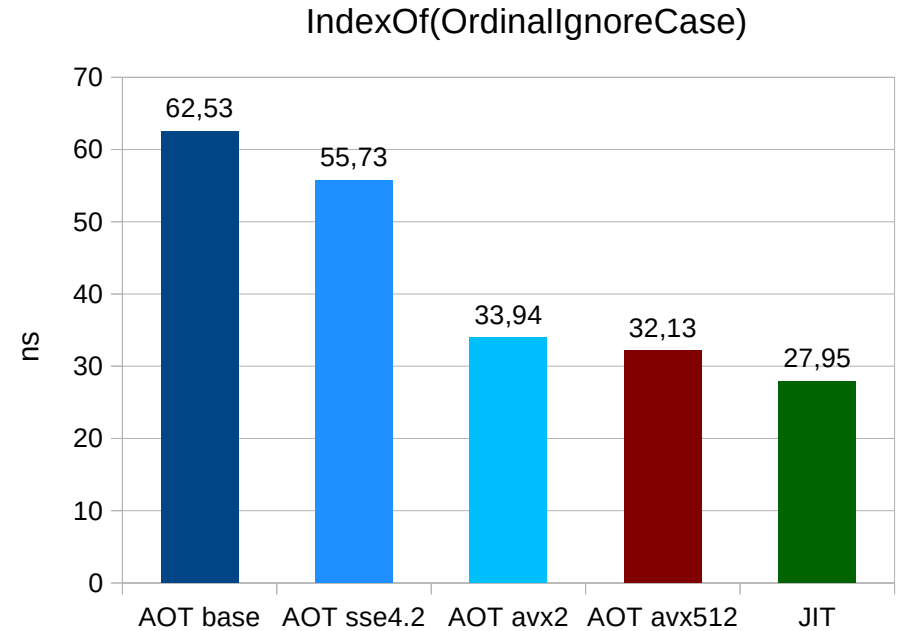
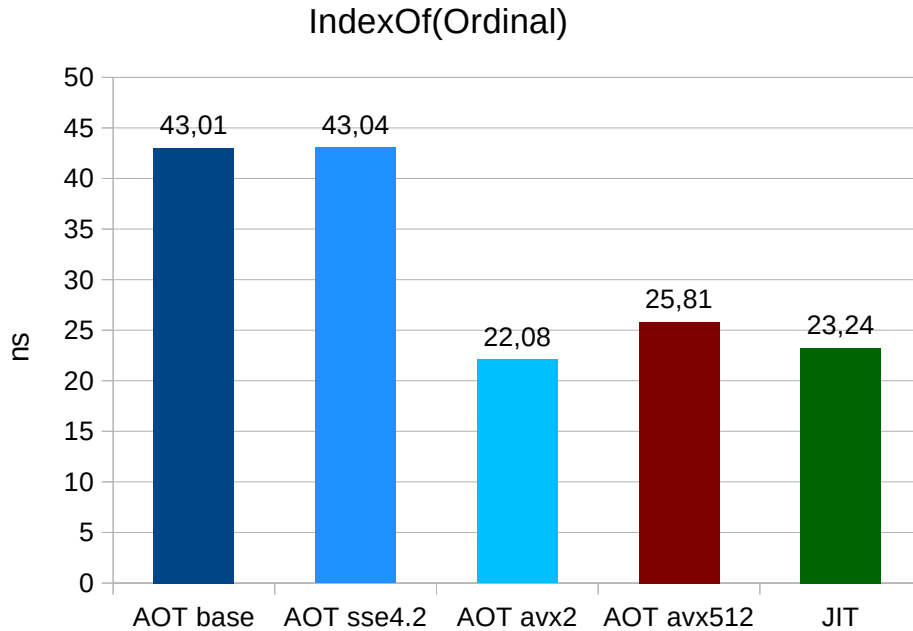


IndexOf(OrdinalIgnoreCase)



String.IndexOf()

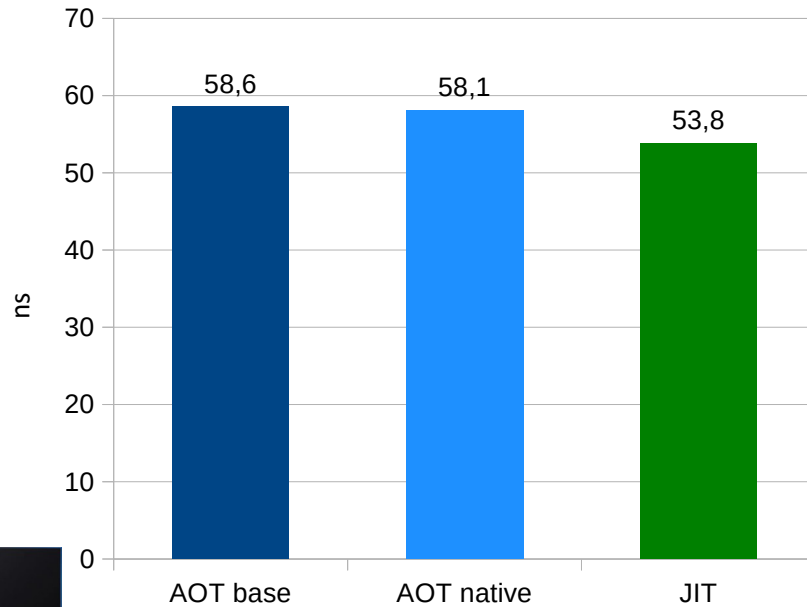
[Ordinal, OrdinalIgnoreCase]



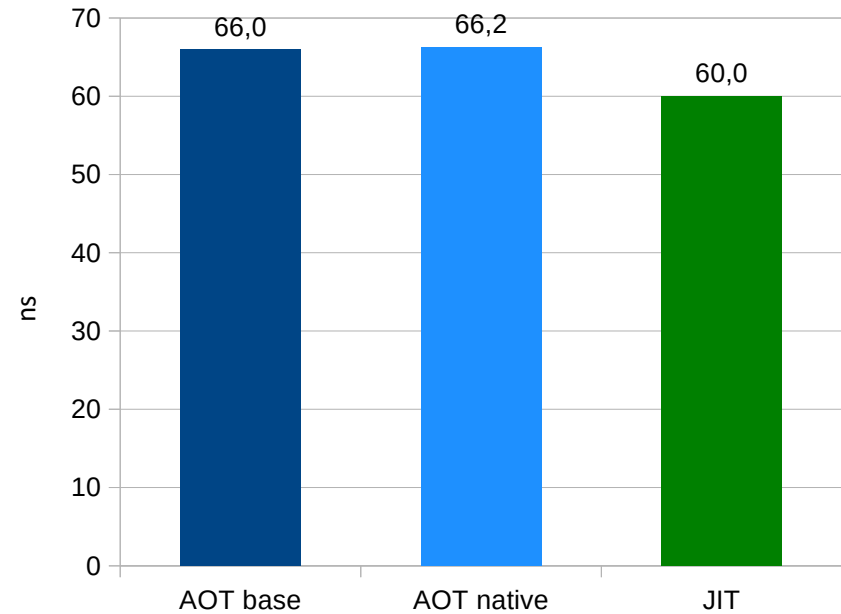
String.IndexOf()

[Ordinal, OrdinalIgnoreCase]

IndexOf(Ordinal)
Apple M2



IndexOf(OrdinalIgnoreCase)
Apple M2



String.IndexOf()

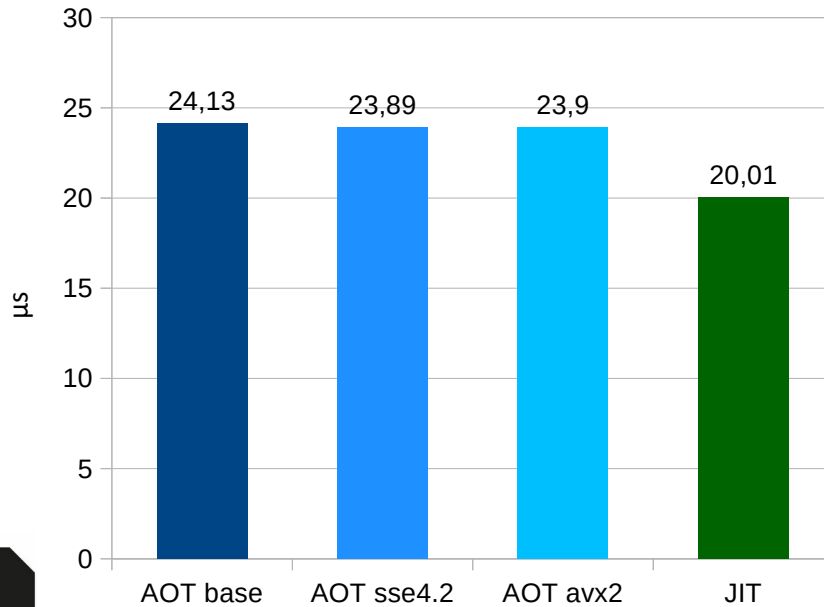
```
[Benchmark]
public void StringIndexOfCurrentCulture()
{
    res = StringToTest.IndexOf(StringToSearch, StringComparison.CurrentCulture);
}

[Benchmark]
public void StringIndexOfCurrentCultureIgnoreCase()
{
    res = StringToTest.IndexOf(StringToSearch, StringComparison.CurrentCultureIgnoreCase);
}
```

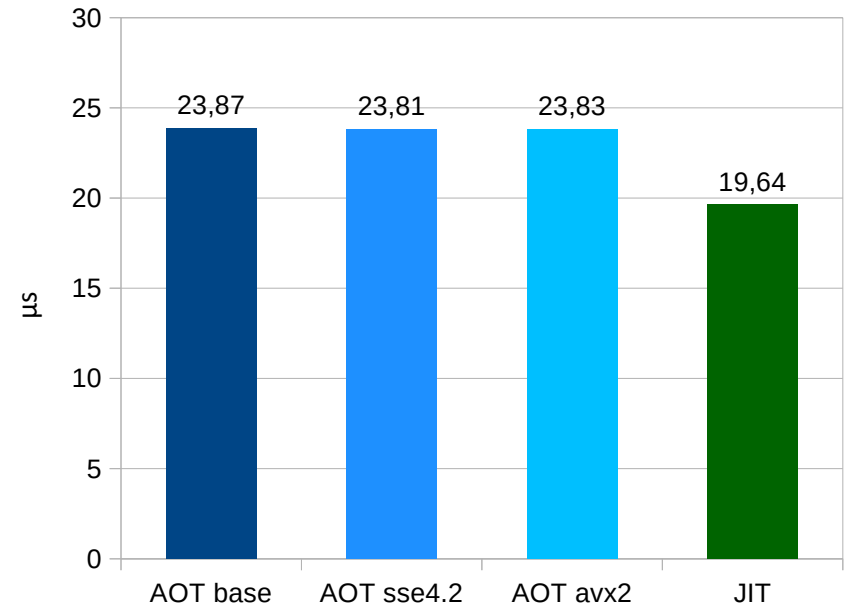
String.IndexOf()

[CurrentCulture, CurrentCultureIgnoreCase]

IndexOf(CurrentCulture)

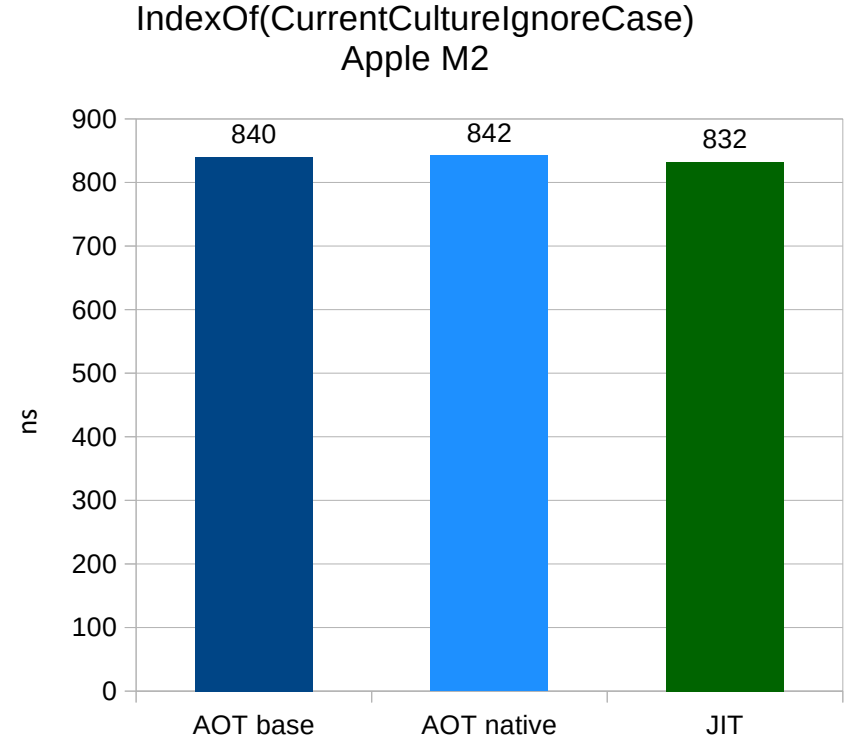
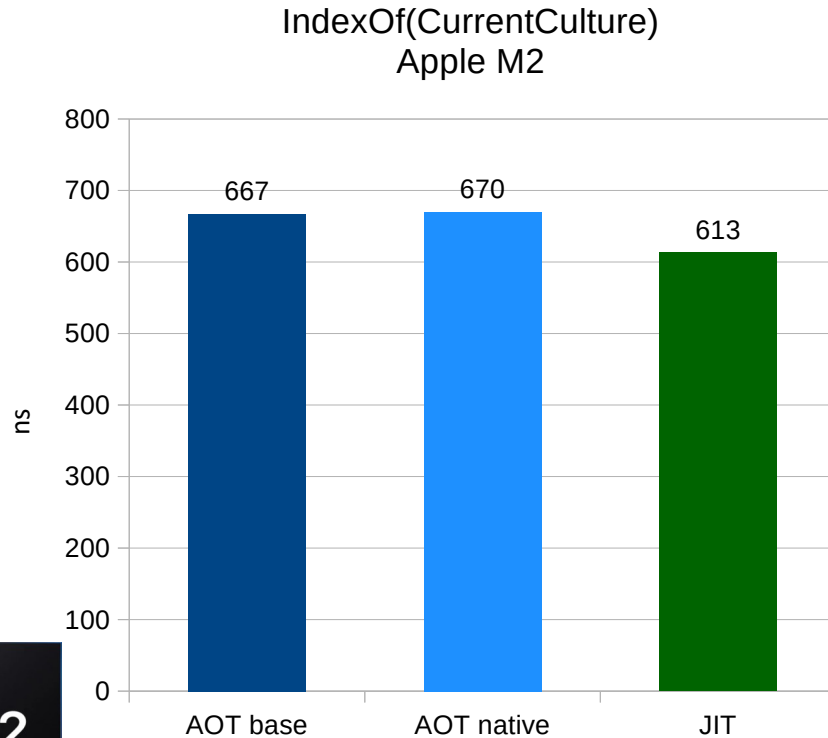


IndexOf(CurrentCultureIgnoreCase)



String.IndexOf()

[CurrentCulture, CurrentCultureIgnoreCase]



Regex.Count() #1

```
// The Adventures of Sherlock Holmes by Arthur Conan Doyle
private static readonly string book = new HttpClient()
    .GetStringAsync("https://www.gutenberg.org/files/1661/1661-0.txt").Result;

private const string pattern = @"\byear\b";

// Interpreted
private Regex _regexInterpreted = new Regex(pattern, RegexOptions.IgnoreCase);

[Benchmark]
public int Count1Interpreted() => _regexInterpreted.Count(book);

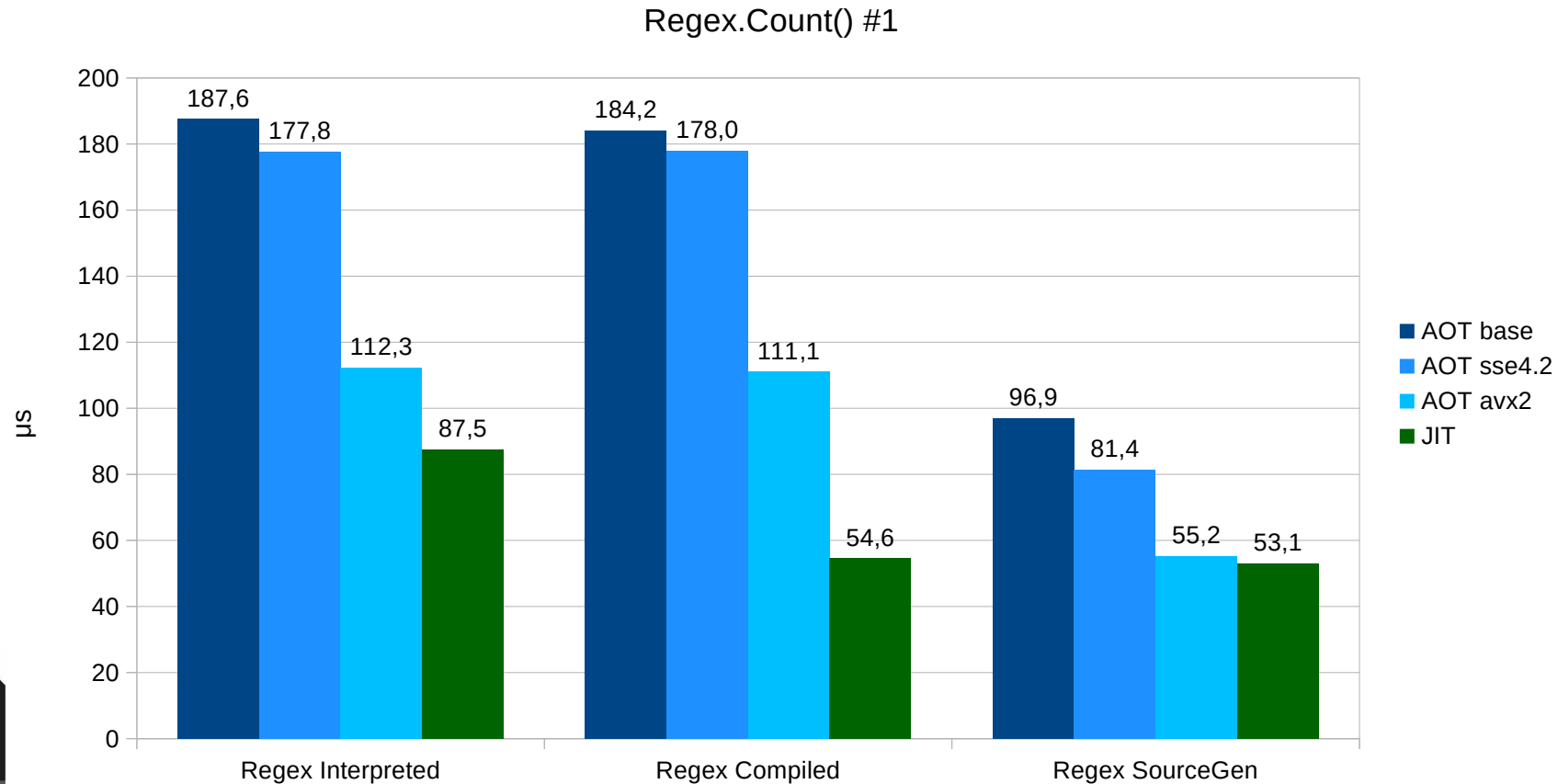
// Compiled
private Regex _regexCompiled = new Regex(pattern, RegexOptions.Compiled | IgnoreCase);

[Benchmark]
public int Count1Compiled() => _regexCompiled.Count(book);

// SourceGenerated
[GeneratedRegex(pattern, RegexOptions.IgnoreCase, "en-US")]
private static partial Regex Regex1Sourcegenerated();

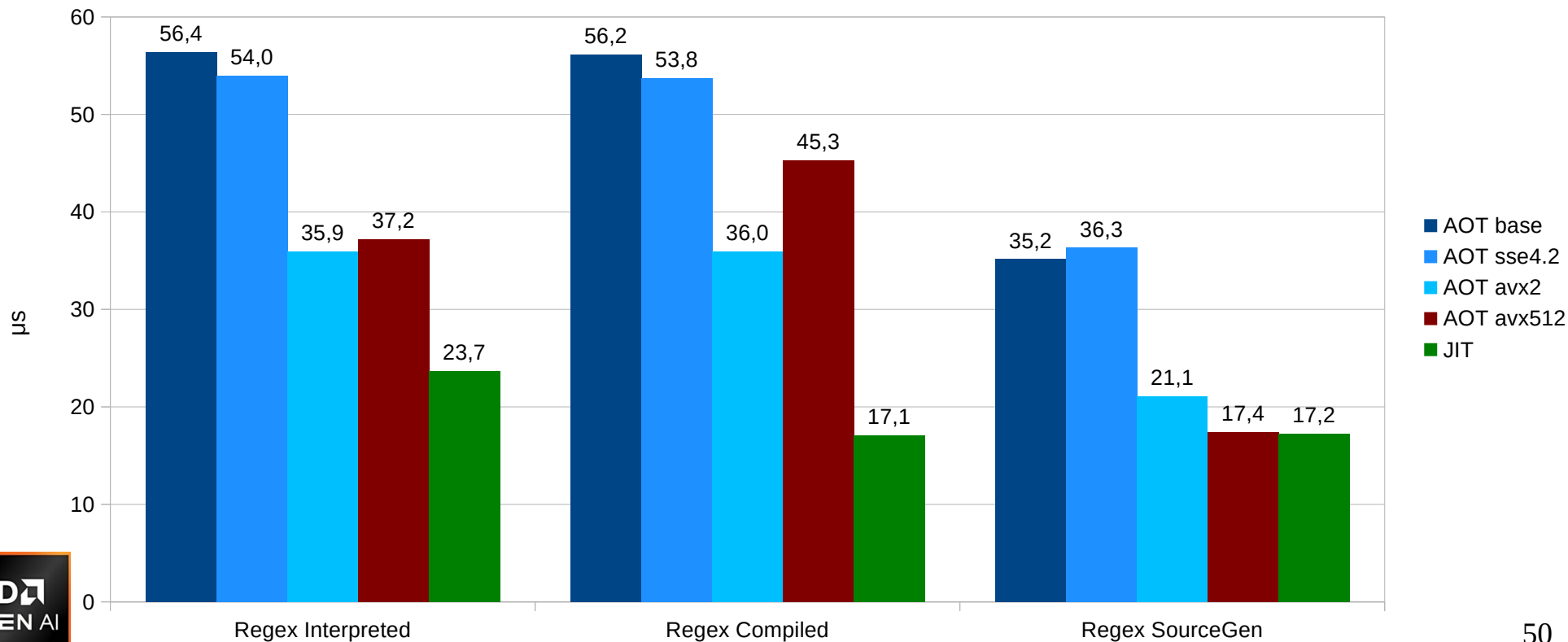
[Benchmark]
public int Count1SourceGenerated() => Regex1Sourcegenerated().Count(book);
```

Regex.Count() #1

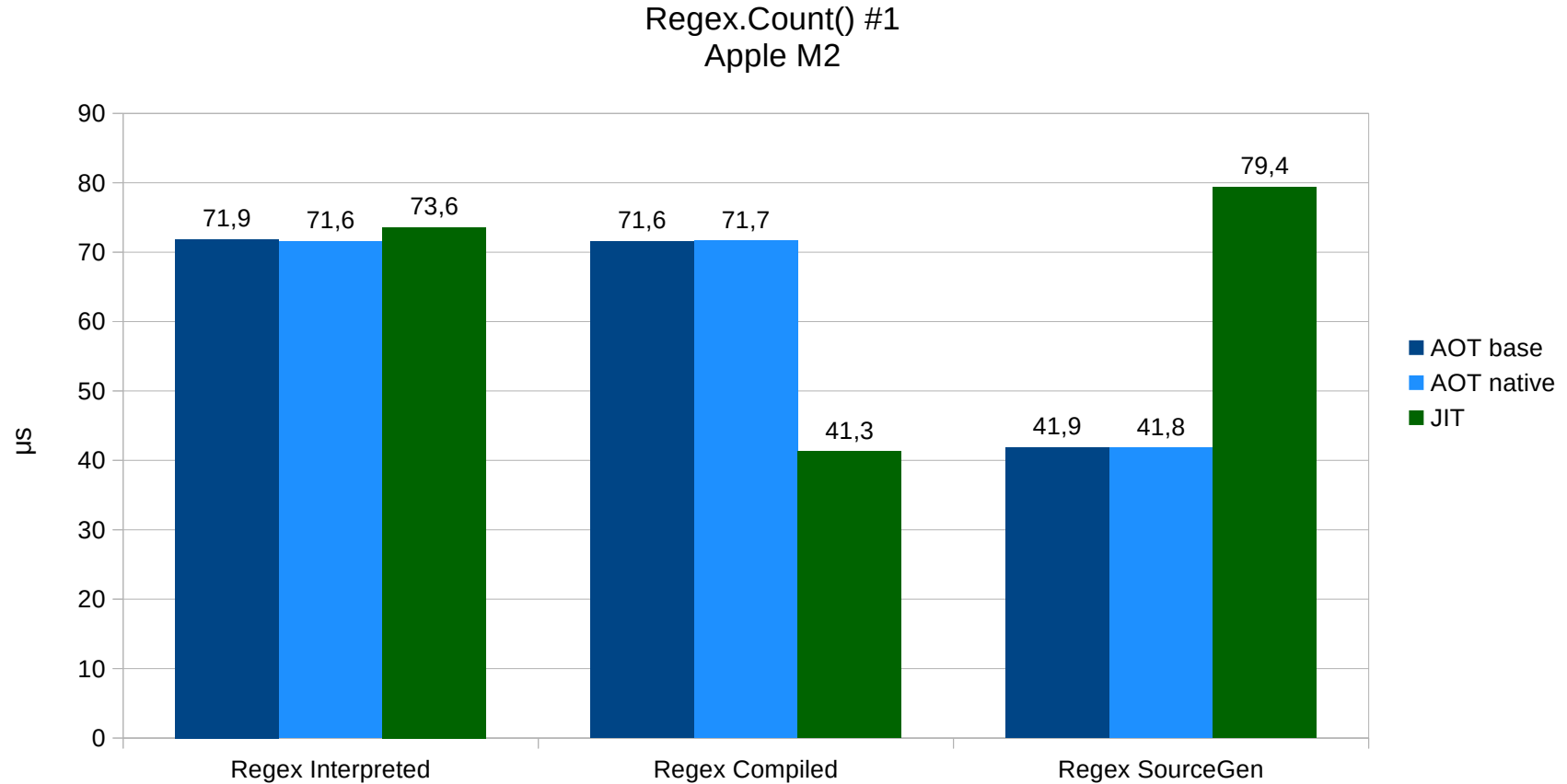


Regex.Count() #1

Regex.Count() #1



Regex.Count() #1



Regex.Count() #2

```
// The Adventures of Sherlock Holmes by Arthur Conan Doyle
private static readonly string book = new HttpClient()
    .GetStringAsync("https://www.gutenberg.org/files/1661/1661-0.txt").Result;

private string pattern = "Holmes|Watson|Lestrade|Hudson|Moriarty|Adler|Moran|Morstan|Gregson";

// Interpreted
private Regex _regexInterpreted = new Regex(pattern, RegexOptions.IgnoreCase);

[Benchmark]
public int Count2Interpreted() => _regexInterpreted.Count(book);

// Compiled
private Regex _regexCompiled = new Regex(pattern, RegexOptions.Compiled | RegexOptions.IgnoreCase);

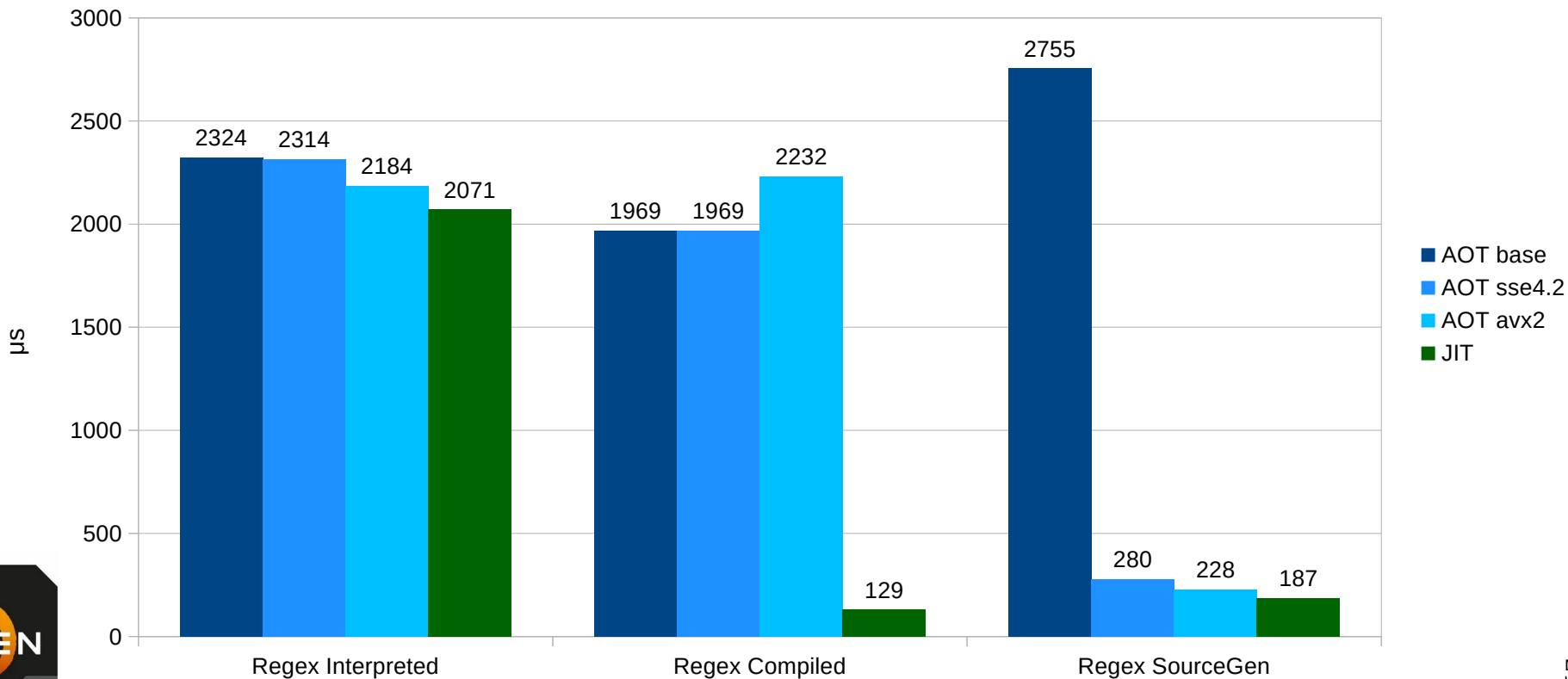
[Benchmark]
public int Count2Compiled() => _regexCompiled.Count(book);

// SourceGenerated
[GeneratedRegex(pattern, RegexOptions.IgnoreCase, "en-US")]
private static partial Regex Regex1Sourcegenerated();

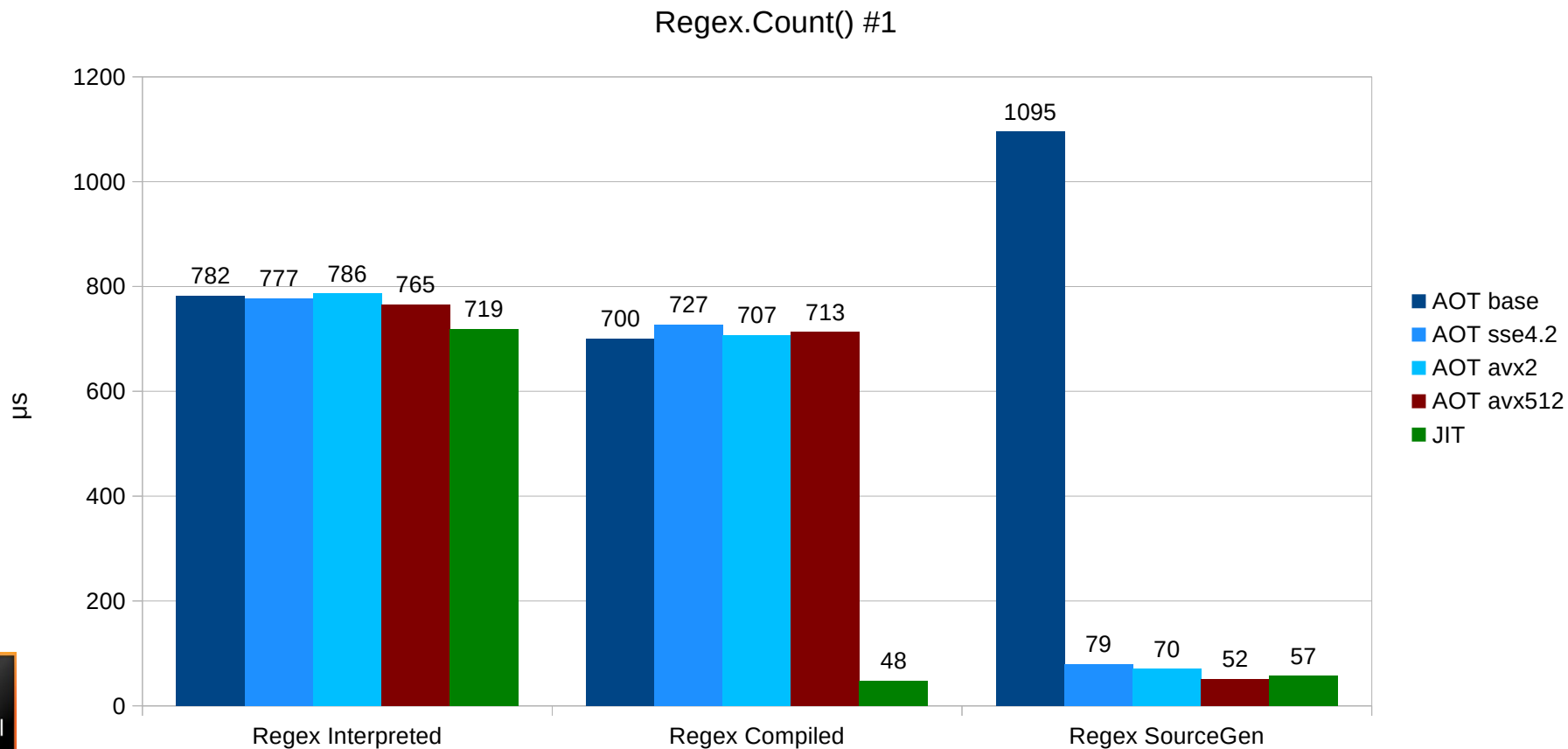
[Benchmark]
public int Count2SourceGenerated() => Regex1Sourcegenerated().Count(book);
```

Regex.Count() #2

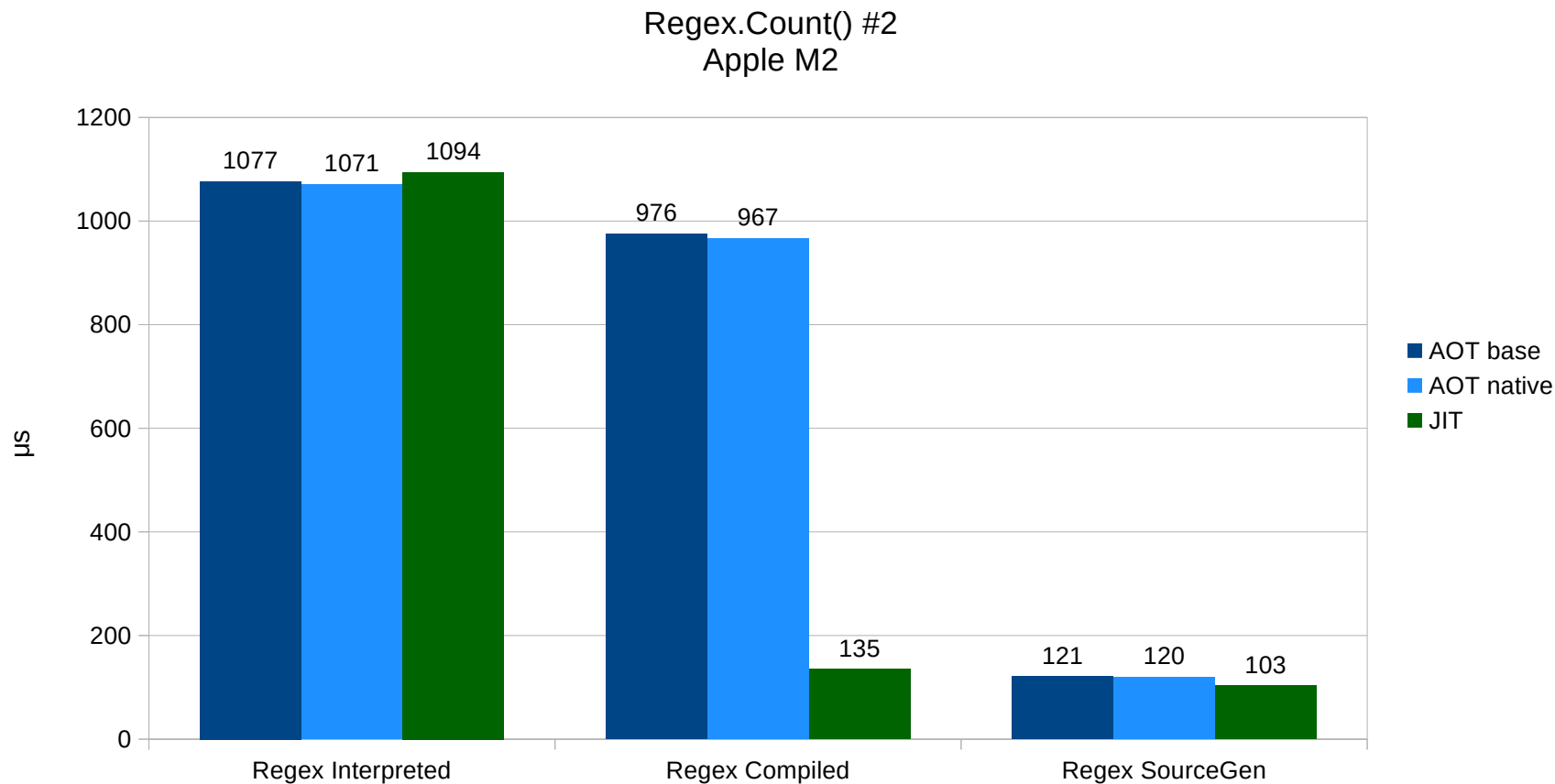
Regex.Count() #1



Regex.Count() #2



Regex.Count() #2



base (SSE2)

```

movaps    xmmword ptr [rsp+30h], xmm15
lea       rdx,[rsp+40h]
lea       r8,[rsp+30h]
lea       rcx,[rsp+0C0h]
call      S_P_CoreLib_System_Runtime_Intrinsics_Vector128_Shuffle
jmp       (07FF60CE7E804h)
movaps    xmm0, xmm6
pshufb    xmm0, xmm15
movaps    xmmword ptr [rsp+0C0h], xmm0
movaps    xmmword ptr [rsp+40h], xmm7
movaps    xmmword ptr [rsp+30h], xmm14
lea       rdx,[rsp+40h]
lea       r8,[rsp+30h]
lea       rcx,[rsp+0B0h]
call      S_P_CoreLib_System_Runtime_Intrinsics_Vector128_Shuffle
movaps    xmm0, xmmword ptr [rsp+0C0h]
movaps    xmm1, xmmword ptr [rsp+0B0h]
pand      xmm0, xmm1
movaps    xmmword ptr [rsp+0E0h], xmm0
test      byte ptr [g_cpuFeatures (07FF60D418260h)],1 // check SSSE3
jne       (07FF60CE7E870h)
movaps    xmmword ptr [rsp+40h], xmm8
movaps    xmmword ptr [rsp+30h], xmm15
.....
call      S_P_CoreLib_System_Runtime_Intrinsics_Vector128_Shuffle
jmp       (07FF60CE7E882h)
movaps    xmm1,xmm8
pshufb    xmm1,xmm15

```



SSE 4.2

```

movaps    xmm2, xmm6
pshufb    xmm2, xmm0
movaps    xmm3, xmm7
pshufb    xmm3, xmm1 // SSSE3
pand      xmm2, xmm3
movaps    xmm3, xmm8
pshufb    xmm3, xmm0
movaps    xmm4, xmm9
pshufb    xmm4, xmm1
pand      xmm3, xmm4
movaps    xmm4, xmm2
palignr   xmm4, xmm12,0Eh
movaps    xmm5, xmm3
palignr   xmm5, xmm13,0Fh
pand      xmm4, xmm5
movaps    xmm5, xmm10
pshufb    xmm5, xmm0
movaps    xmm0, xmm11
pshufb    xmm0, xmm1
pand      xmm0, xmm5
pand      xmm0, xmm4
movaps    xmm12, xmm2
movaps    xmm13, xmm3

```

```

[Intrinsic]
public static Vector128<byte> Shuffle(Vector128<byte> vector, Vector128<byte> indices)
{
    Unsafe.SkipInit(out Vector128<byte> result);

    for (int index = 0; index < Vector128<byte>.Count; index++)
    {
        byte selectedIndex = indices.GetElementUnsafe(index);
        byte selectedValue = 0;

        if (selectedIndex < Vector128<byte>.Count)
        {
            selectedValue = vector.GetElementUnsafe(selectedIndex);
        }
        result.SetElementUnsafe(index, selectedValue);
    }

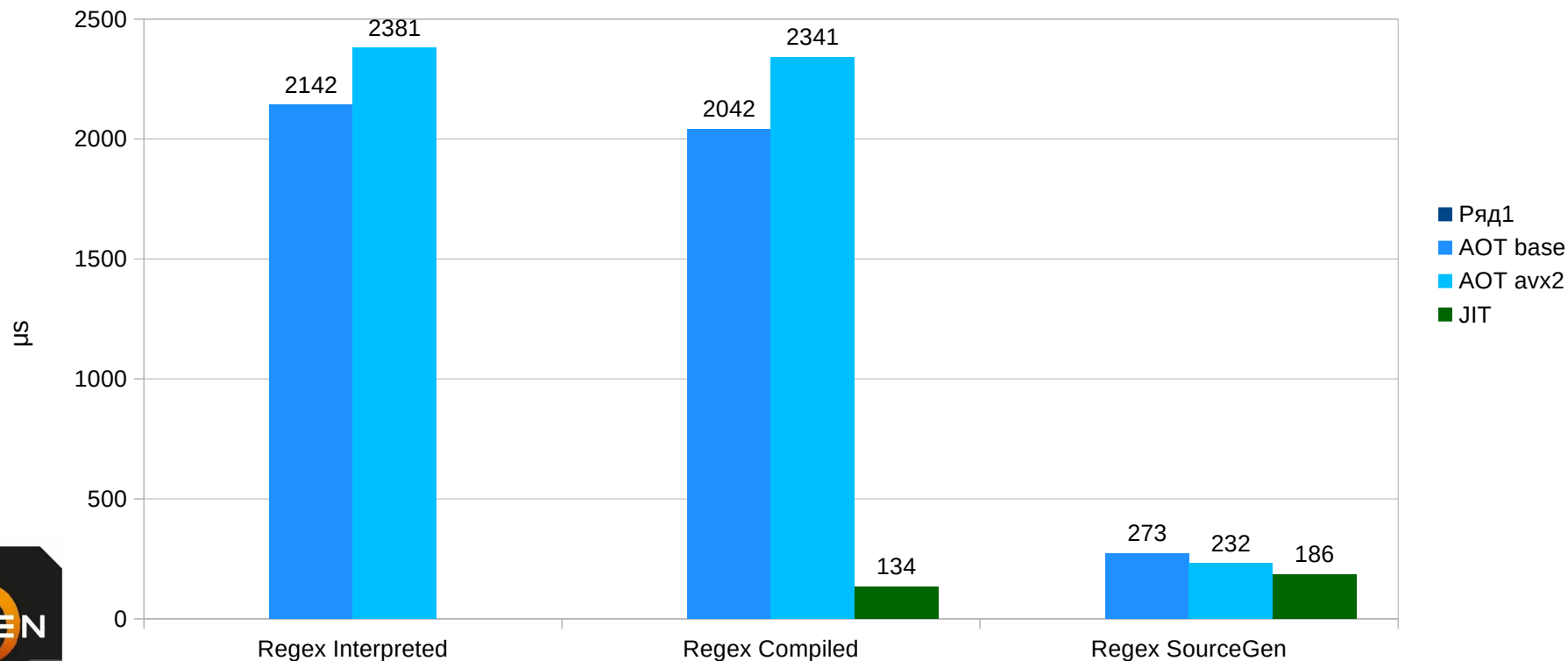
    return result;
}

```

Regex.Count() #2

Regex.Count() #2

.NET 11



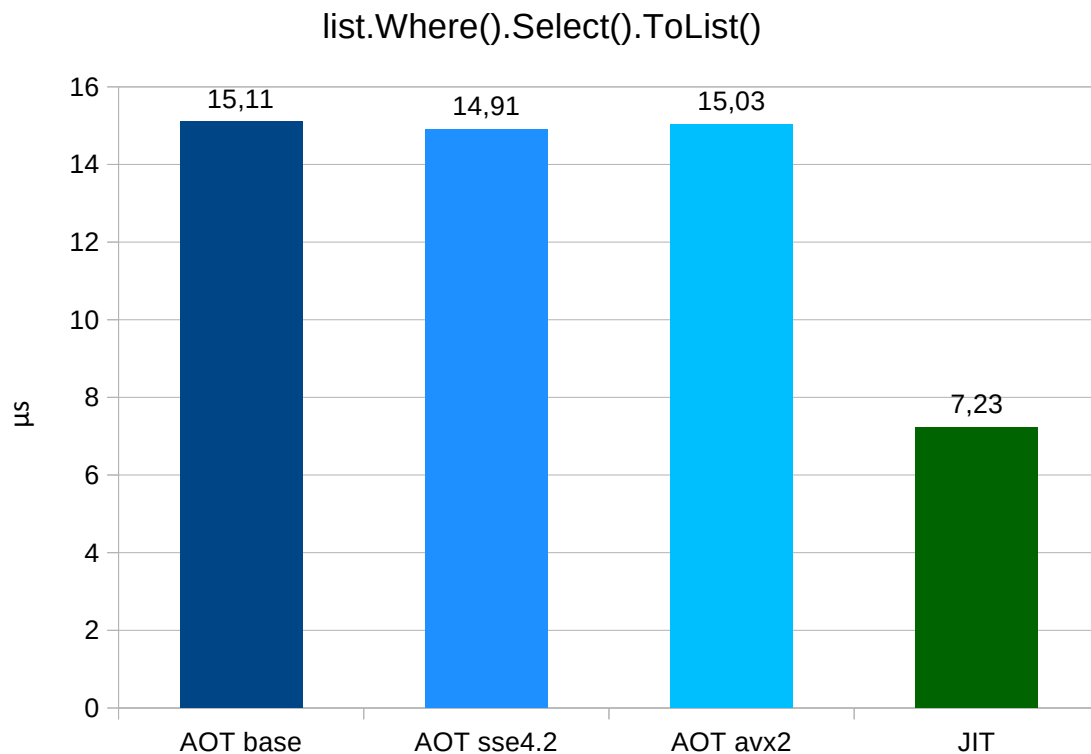
BenchmarkDotNet

Тесты Linq

#1 Linq: Where + Select + ToUpper + ToList

```
// Simple filtering + projection + materialization
[Benchmark]
public List<string> WhereSelectToList()
{
    return _peopleList
        .Where(p => p.Age >= 30 && p.IsActive)
        .Select(p => p.Name.ToUpper())
        .ToList();
}
```

#1 Linq: Where + Select + ToUpper + ToList

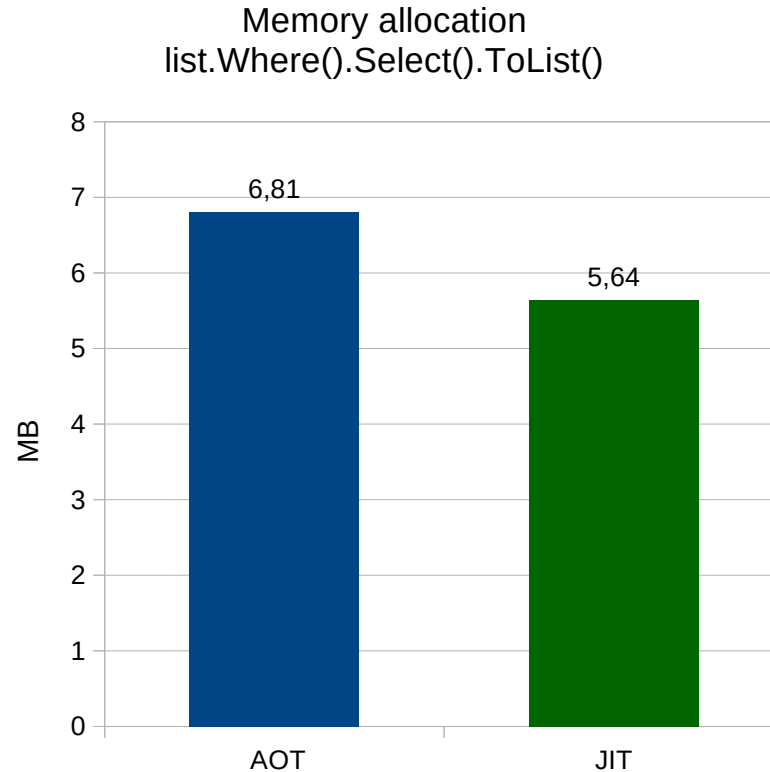


#1 Linq: Where + Select + ToUpper + ToList



#1 Linq: Where + Select + ToUpper + ToList

Выделение памяти



#2 Linq: Inner Join List

```
// Join collections
[Benchmark]
public object List_InnerJoin()
{
    return (from o in _orders
            join e in _employees on o.CustomerId equals e.Id
            where o.Amount > 1000
            select new OrderInfo
            {
                OrderId = o.Id,
                EmployeeName = e.Name,
                Amount = o.Amount,
                Department = e.Department
            }).ToList();
}
```

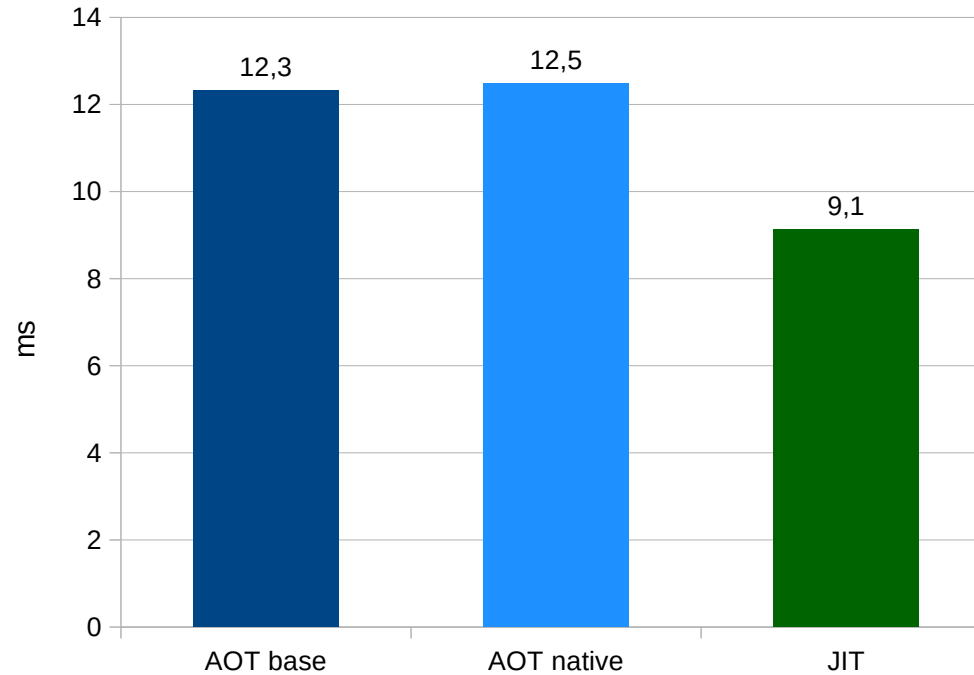
#2 Linq: Inner Join List



#2 Linq: Inner Join List

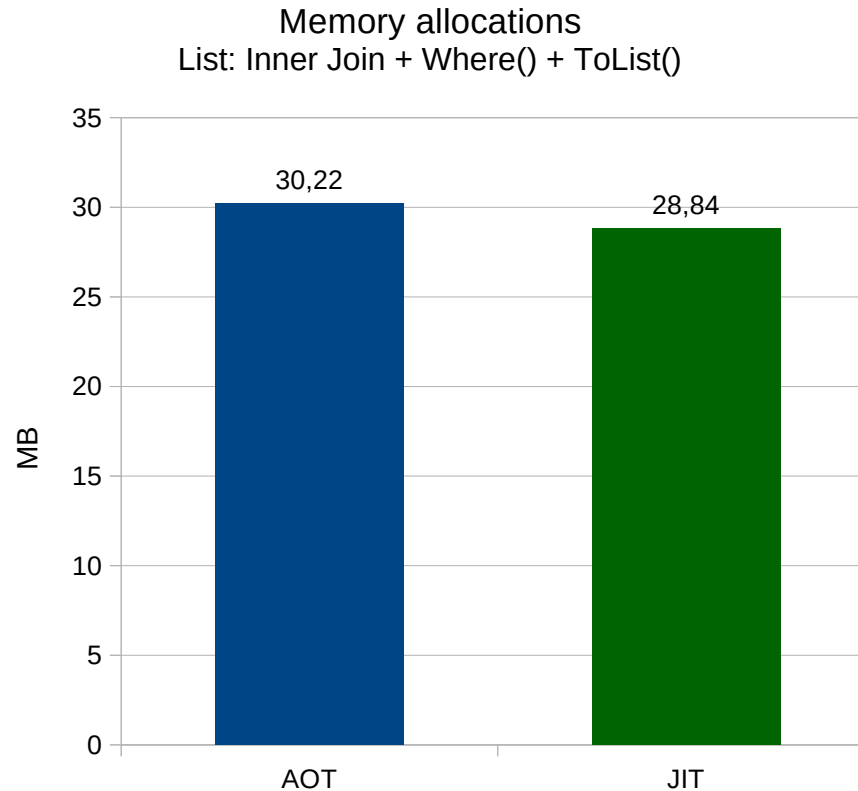
List: Inner Join + Where() + ToList()

Apple M2



#2 Linq: Inner Join List

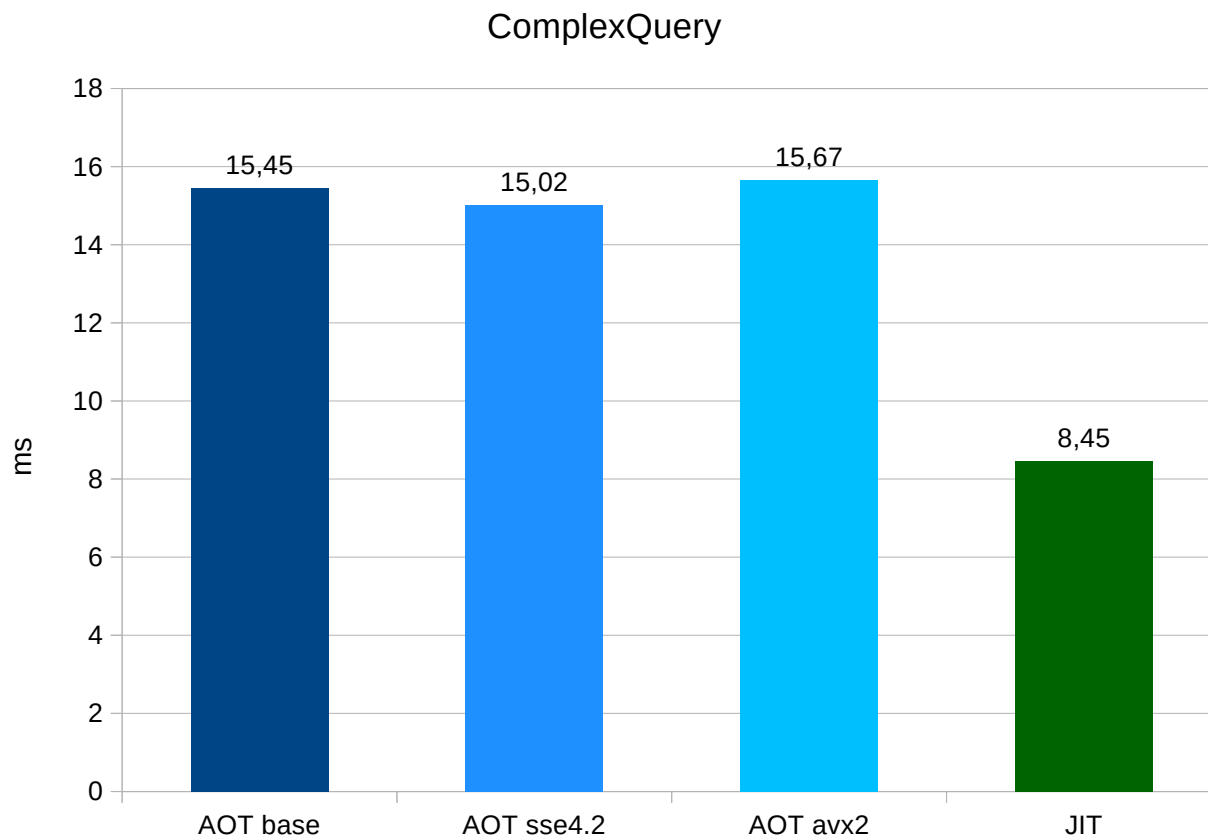
Выделение памяти



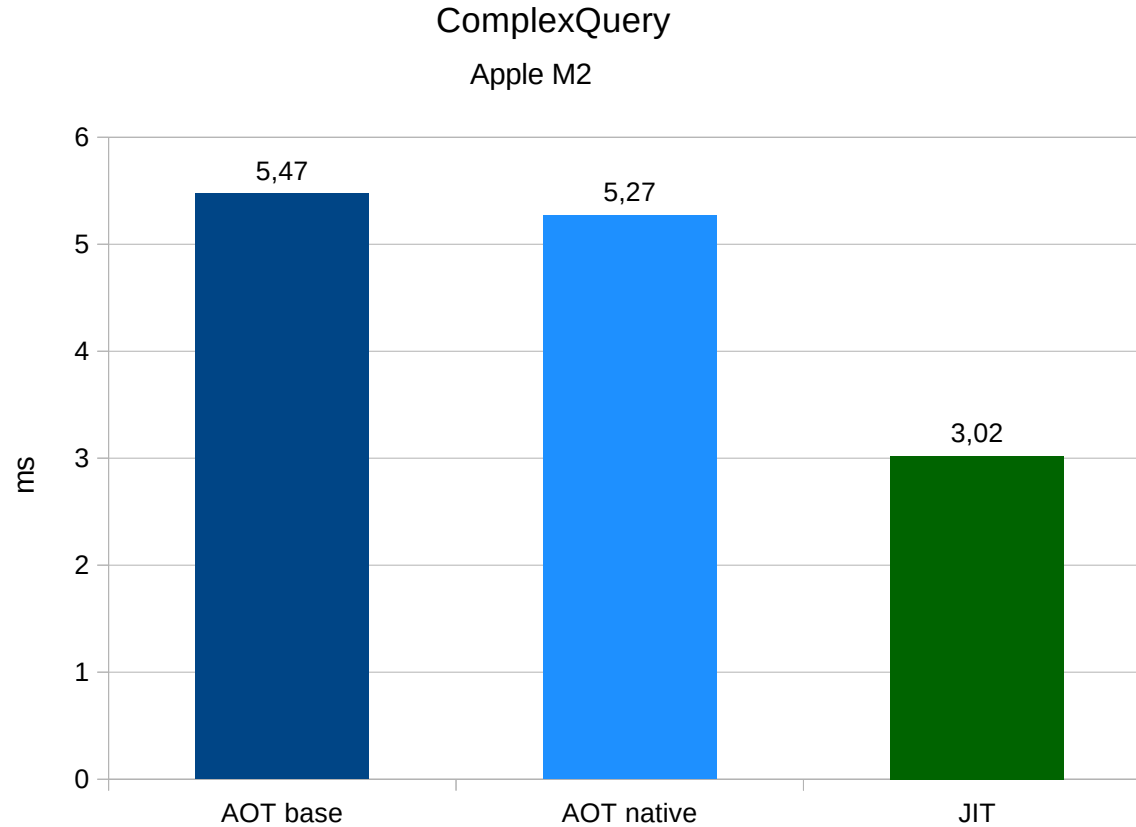
#3 Linq: Complex chained query

```
// Complex chained query
[Benchmark]
public Dictionary<string, int> ComplexQuery()
{
    return _peopleList
        .Where(p => p.IsActive && p.Age >= 25)
        .GroupBy(p => p.City)
        .Select(g => new
        {
            City = g.Key,
            Count = g.Count(),
            AvgAge = (int)g.Average(p => p.Age)
        })
        .OrderByDescending(x => x.Count)
        .Take(10)
        .ToDictionary(x => x.City, x => x.Count);
}
```

#3 Linq: Complex chained query



#3 Linq: Complex chained query

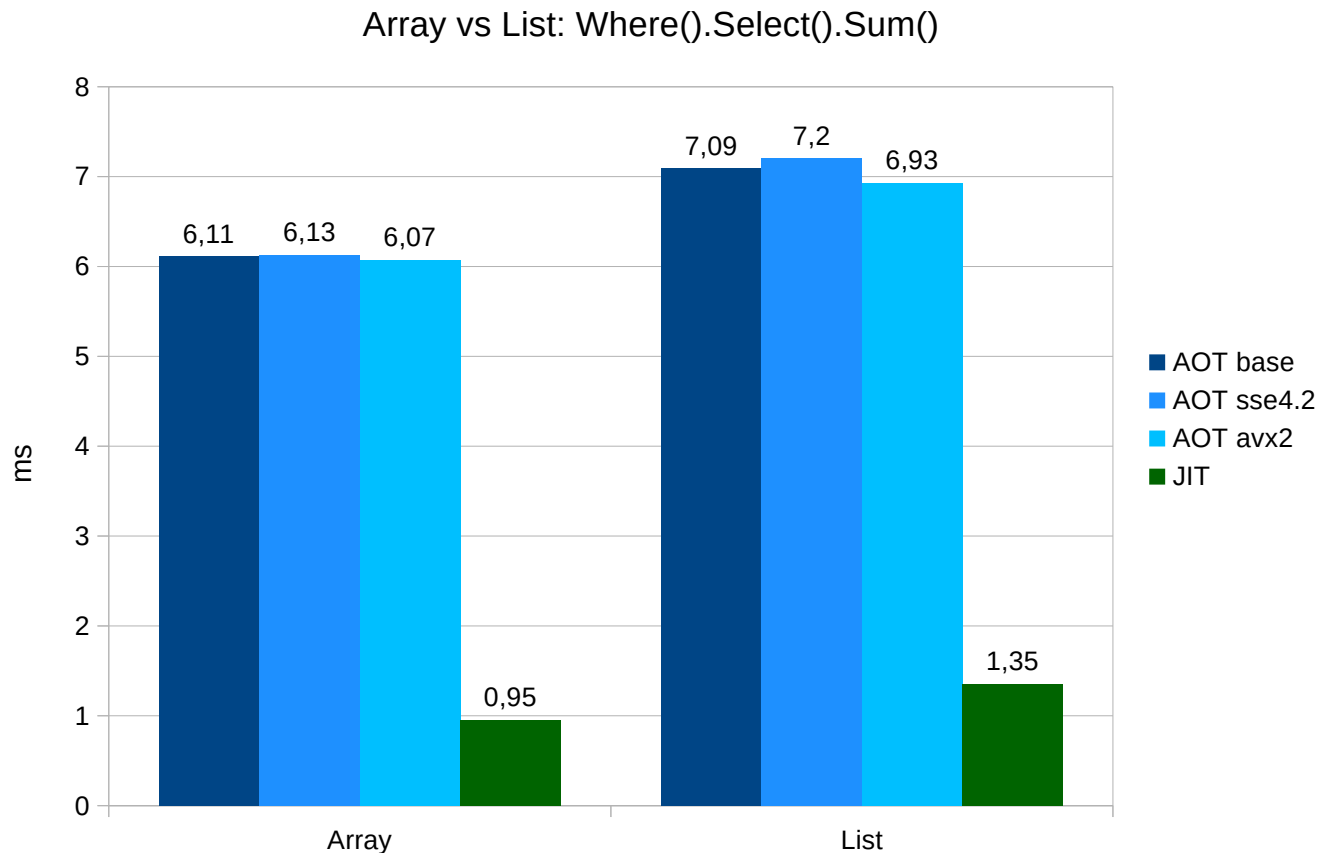


#4 Linq: Array vs List

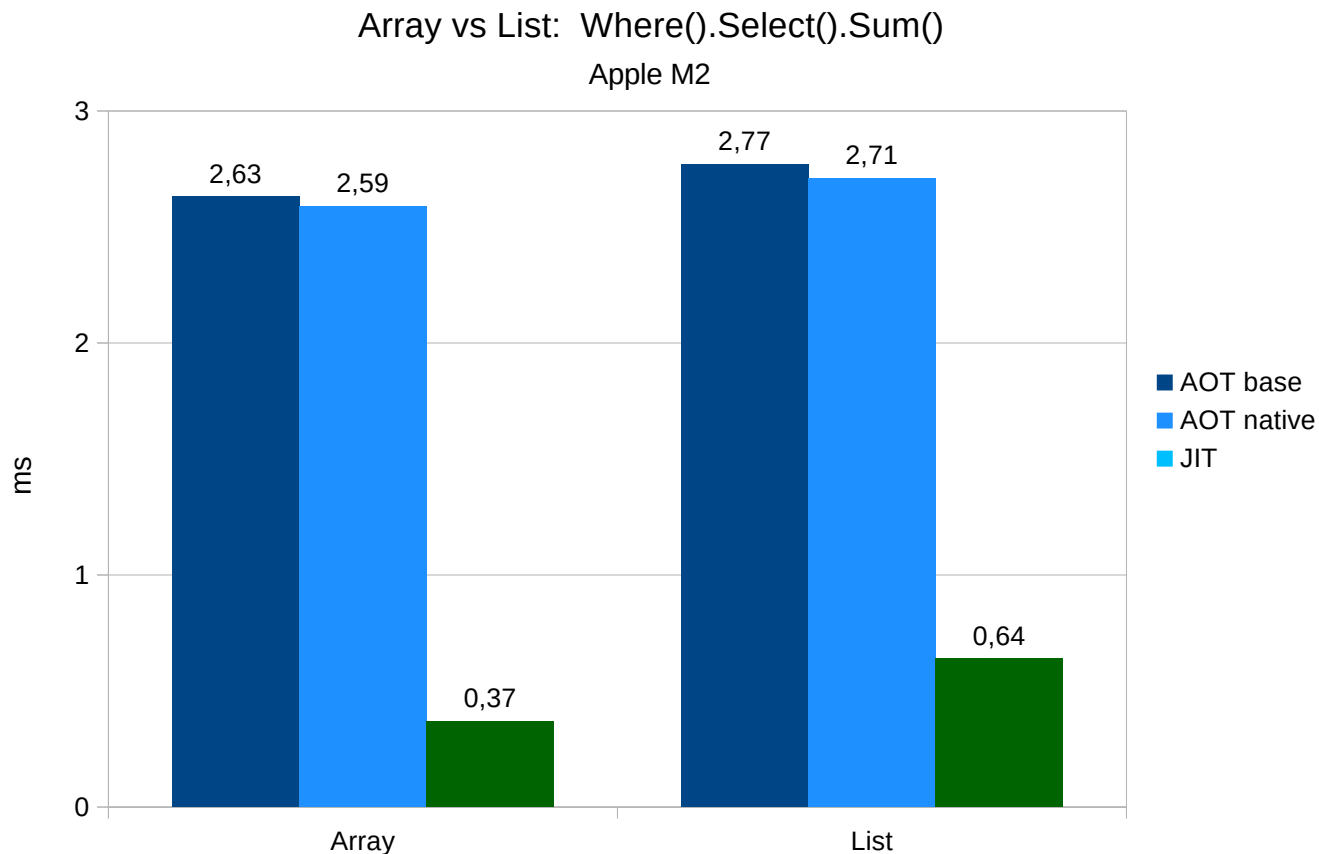
```
[Benchmark(Description = "Array: Where + Select + Sum")]  
public long Array_WhereSelectSum()  
{  
    return _peopleArray  
        .Where(p => p.Age > 40)  
        .Select(p => (long)p.Id)  
        .Sum();  
}
```

```
[Benchmark(Description = "List: Where + Select + Sum")]  
public long List_WhereSelectSum()  
{  
    return _peopleList  
        .Where(p => p.Age > 40)  
        .Select(p => (long)p.Id)  
        .Sum();  
}
```

#4 Linq: Array vs List (девиртуализация)



#4 Linq: Array vs List (девиртуализация)

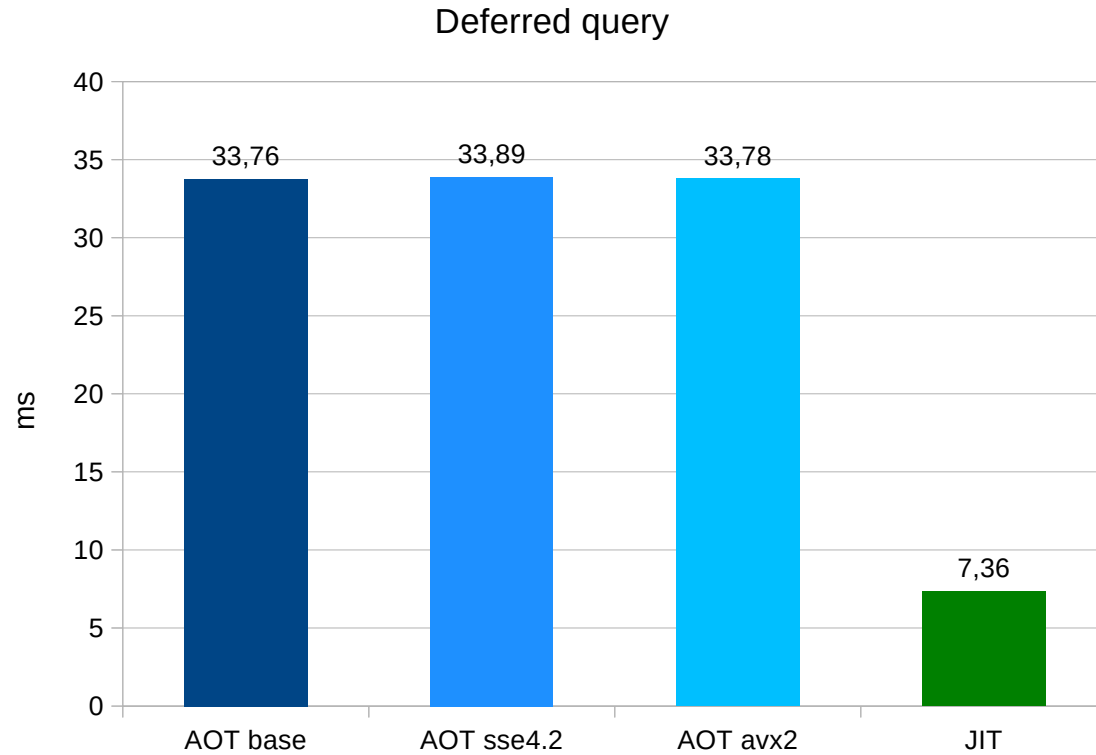


#5 Linq: Deferred query

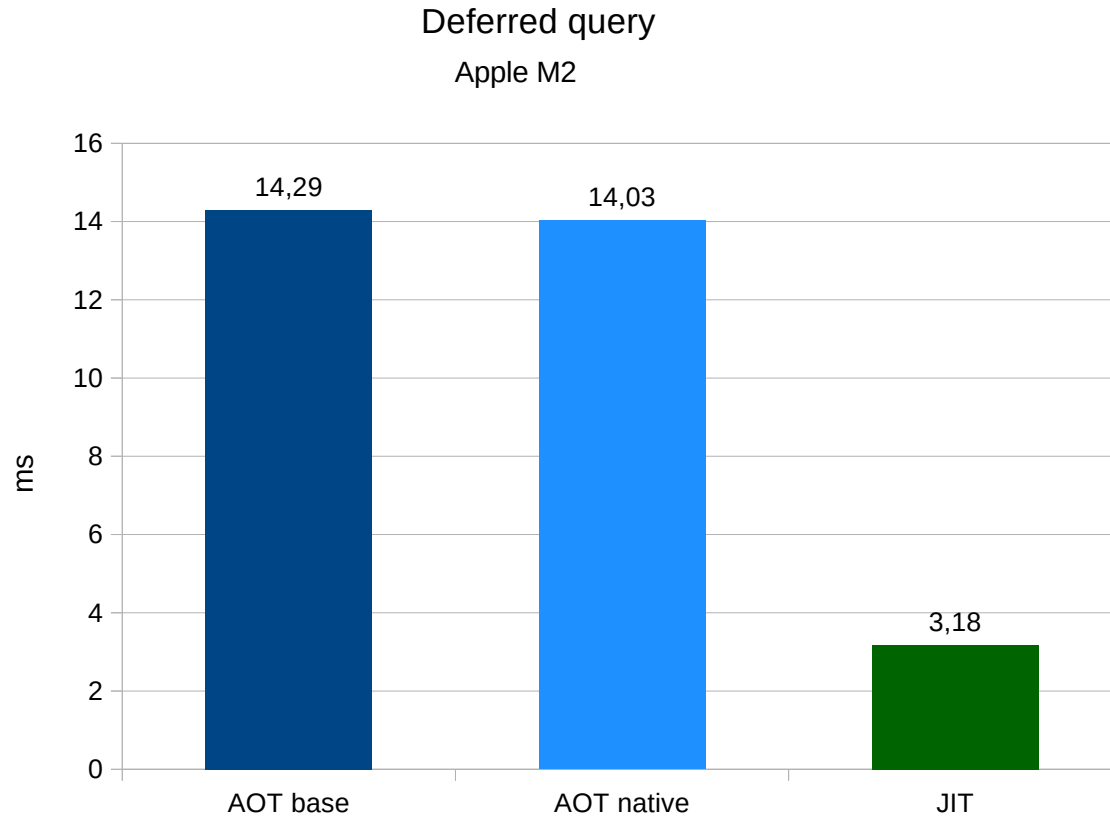
```
[Benchmark(Description = "Deferred: Same query enumerated 5 times with foreach")]
public int Deferred_EnumerateFiveTimes()
{
    var query = _peopleList
        .Where(p => p.Age >= 30 && p.IsActive)
        .Select(p => p.Name.Length);

    // N = 5
    int total = 0;
    for (int i = 0; i < N; i++)
    {
        foreach (var length in query)
        {
            total += length;
        }
    }
    return total;
}
```

#5 Linq: Deferred query



#5 Linq: Deferred query



Сериализация JSON через System.Text.JsonSerializer: source generated

```
[Benchmark]
public void SerializeJsonUsingSourceGen()
{
    Json2 = JsonSerializer.Serialize(PnrList, SourceGenerationContext.Default.PnrList);
}
```

```
[Benchmark]
public void DeserializeJsonUsingSourceGen()
{
    PnrList = JsonSerializer
        .Deserialize<PnrList>(Json, SourceGenerationContext.Default.PnrList);
}
```

Сериализация JSON через System.Text.JsonSerializer: reflection

```
[Benchmark]
[RequiresUnreferencedCode("...")]
[RequiresDynamicCode("...")]
public void SerializeJsonUsingReflection()
{
    Json2 = JsonSerializer.Serialize(PnrList, options);
}
```

```
[Benchmark]
[RequiresUnreferencedCode("...")]
[RequiresDynamicCode("...")]
public void DeserializeJsonUsingReflection()
{
    PnrList = JsonSerializer.Deserialize<PnrList>(Json, options);
}
```

Сериализация JSON через System.Text.JsonSerializer: reflection

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>

    <PublishAot>true</PublishAot>

  </PropertyGroup>

  <ItemGroup>
    <TrimmerRootAssembly Include="ClassLibraryForDeserialization" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\ClassLibrary1\ClassLibraryForDeserialization.csproj" />
  </ItemGroup>

</Project>
```

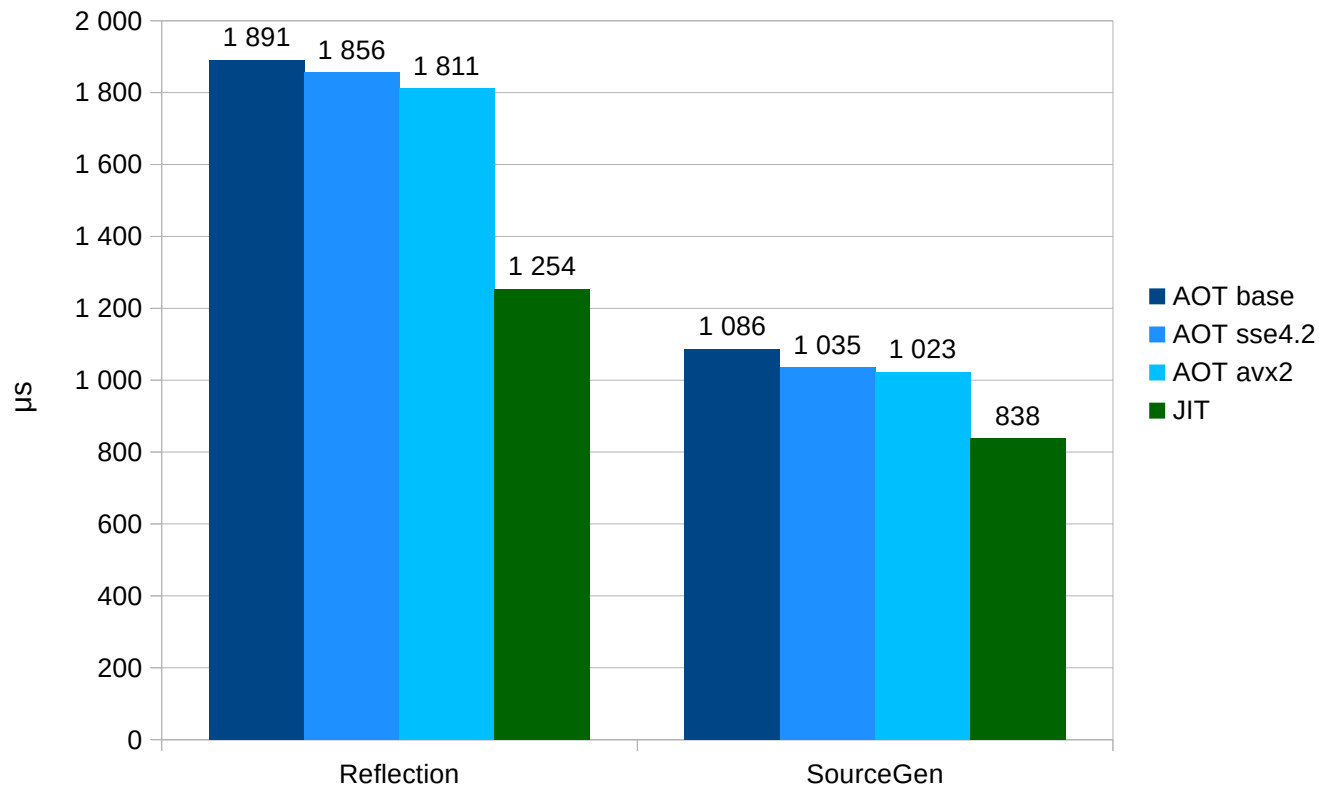
Сериализация JSON через System.Text.JsonSerializer: reflection

```
[GlobalSetup]
[DynamicDependency(DynamicallyAccessedMemberTypes.All, typeof(List<FlightSegment>))]
[DynamicDependency(DynamicallyAccessedMemberTypes.All, typeof(List<HotelStay>))]
[DynamicDependency(DynamicallyAccessedMemberTypes.All, typeof(List<CarRental>))]
[RequiresUnreferencedCode(".")]
[RequiresDynamicCode(".")]
public void Setup()
{
    options = new JsonSerializerOptions()
    {
        TypeInfoResolver = new DefaultJsonTypeInfoResolver(),
    };

    PnrList = new PnrList { Pnrs = PnrFactory.CreateMany(100).ToArray() };
    Json = JsonSerializer.Serialize(PnrList, SourceGenerationContext.Default.PnrList);
}
```

Сериализация JSON через System.Text.JsonSerializer

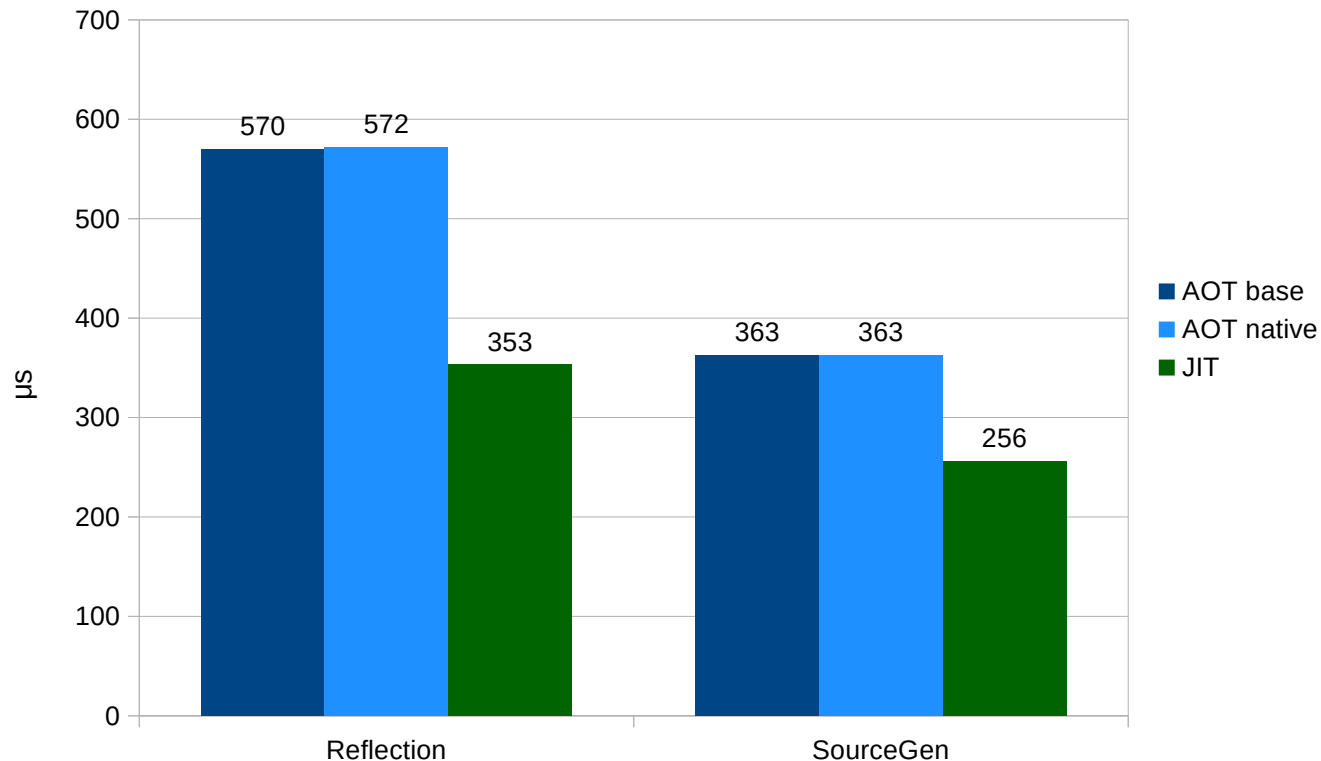
System.Text.JsonSerializer.Serialize()



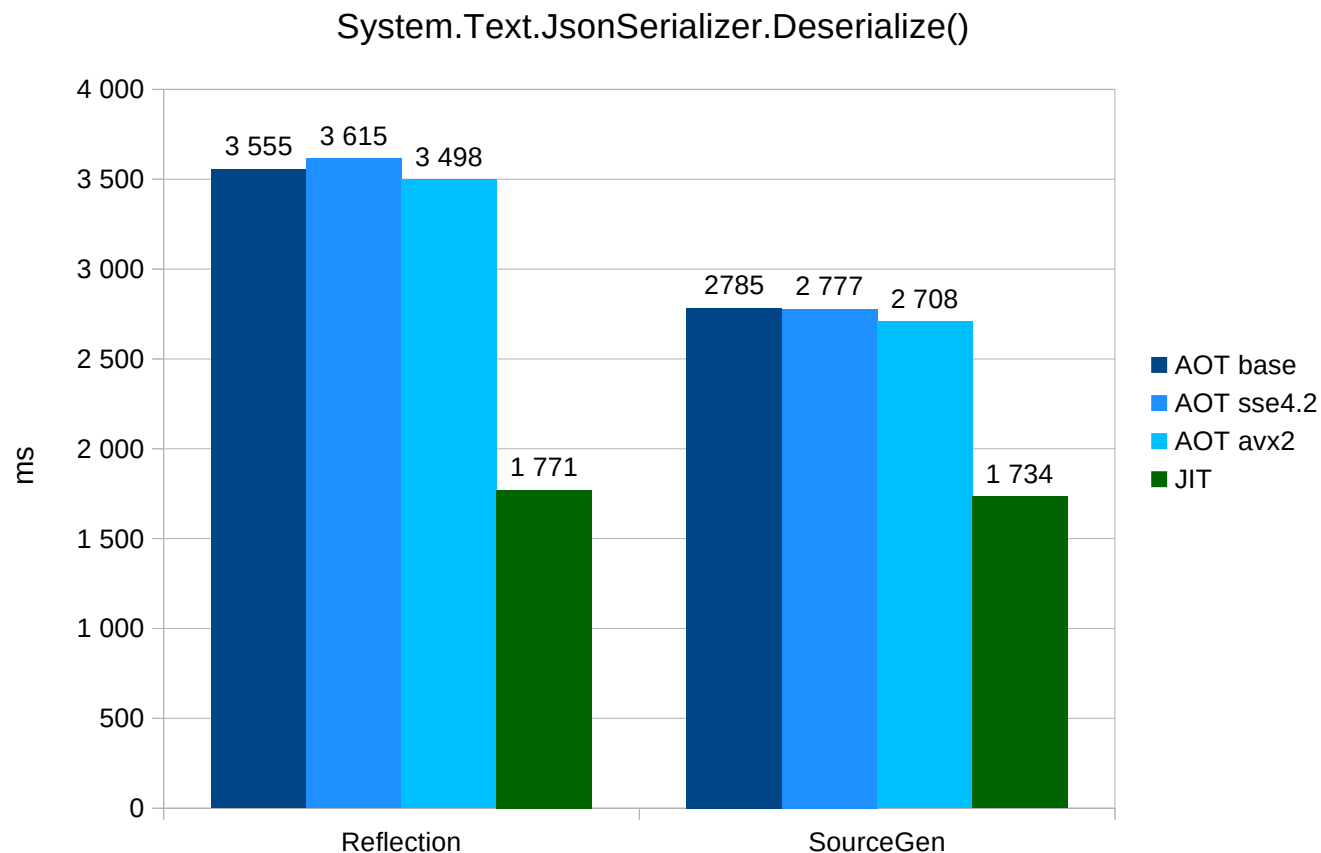
Сериализация JSON через System.Text.JsonSerializer

System.Text.JsonSerializer.Serialize()

Apple M2



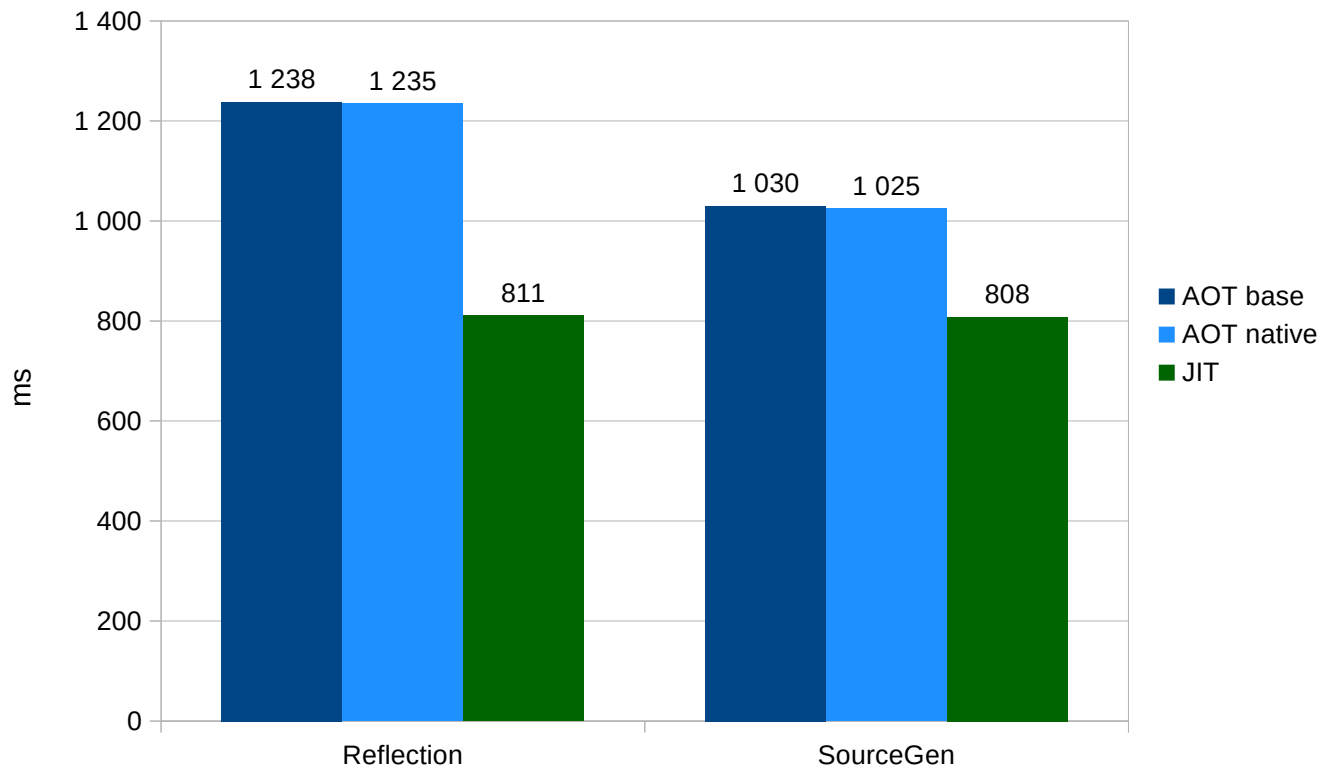
Десериализация JSON через System.Text.JsonSerializer



Десериализация JSON через System.Text.JsonSerializer

System.Text.JsonSerializer.Deserialize()

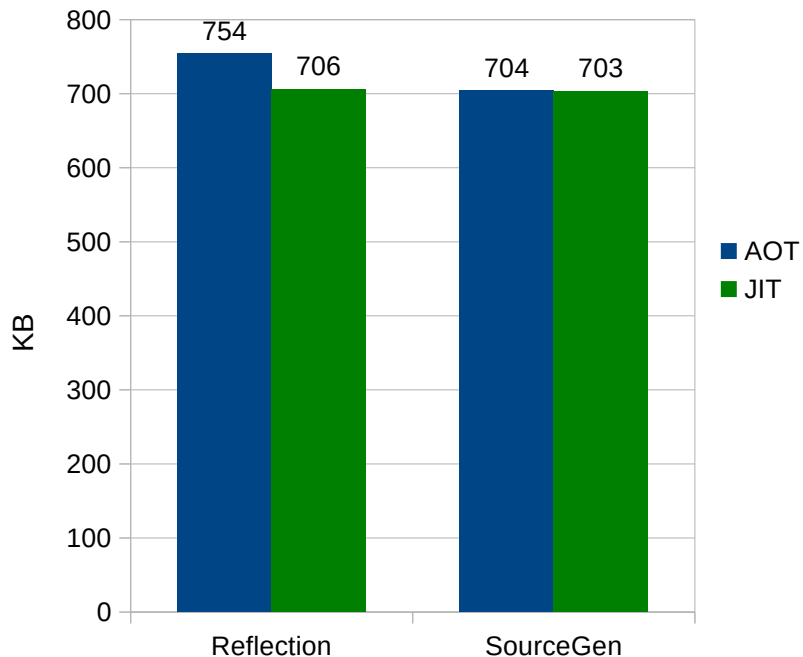
Apple M2



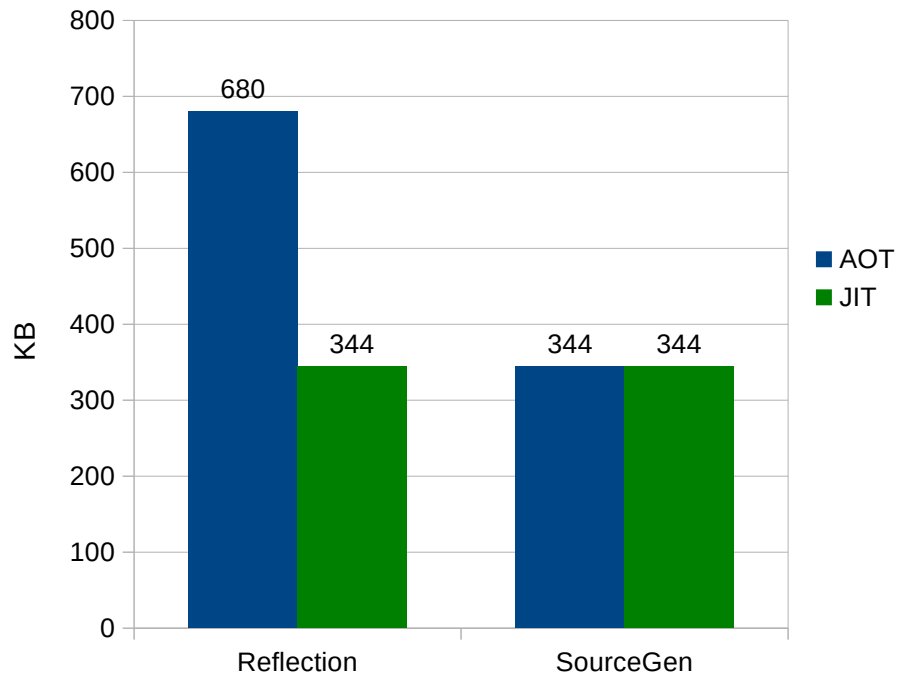
Сериализация и десериализация JSON

Выделение памяти

System.Text.JsonSerializer.Serialize():
Memory Allocations



System.Text.JsonSerializer.Deserialize():
Memory Allocations



Сериализация и десериализация XML (reflection only)

```
[GlobalSetup]
[DynamicDependency(DynamicallyAccessedMemberTypes.All, typeof(List<FlightSegment>))]
[DynamicDependency(DynamicallyAccessedMemberTypes.All, typeof(List<HotelStay>))]
[DynamicDependency(DynamicallyAccessedMemberTypes.All, typeof(List<CarRental>))]
[RequiresUnreferencedCode("...")]
[RequiresDynamicCode("...")]
public void Setup()
{
    PnrList = new PnrList { Pnrs = PnrFactory.CreateMany(100).ToArray() };
    XmlSerializer = new XmlSerializer(typeof(PnrList));

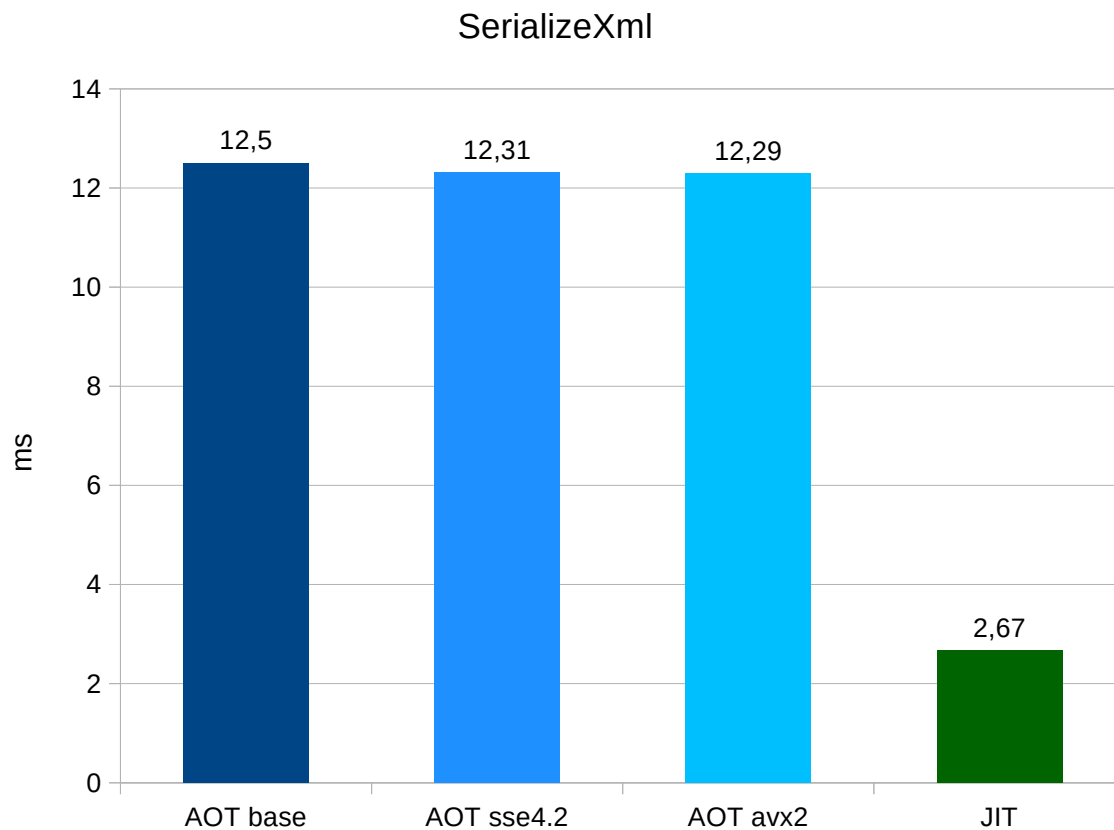
    var stringWriter = new StringWriter();
    XmlSerializer?.Serialize(stringWriter, PnrList);
    Xml = stringWriter.ToString();
}
```

Сериализация и десериализация XML (reflection only)

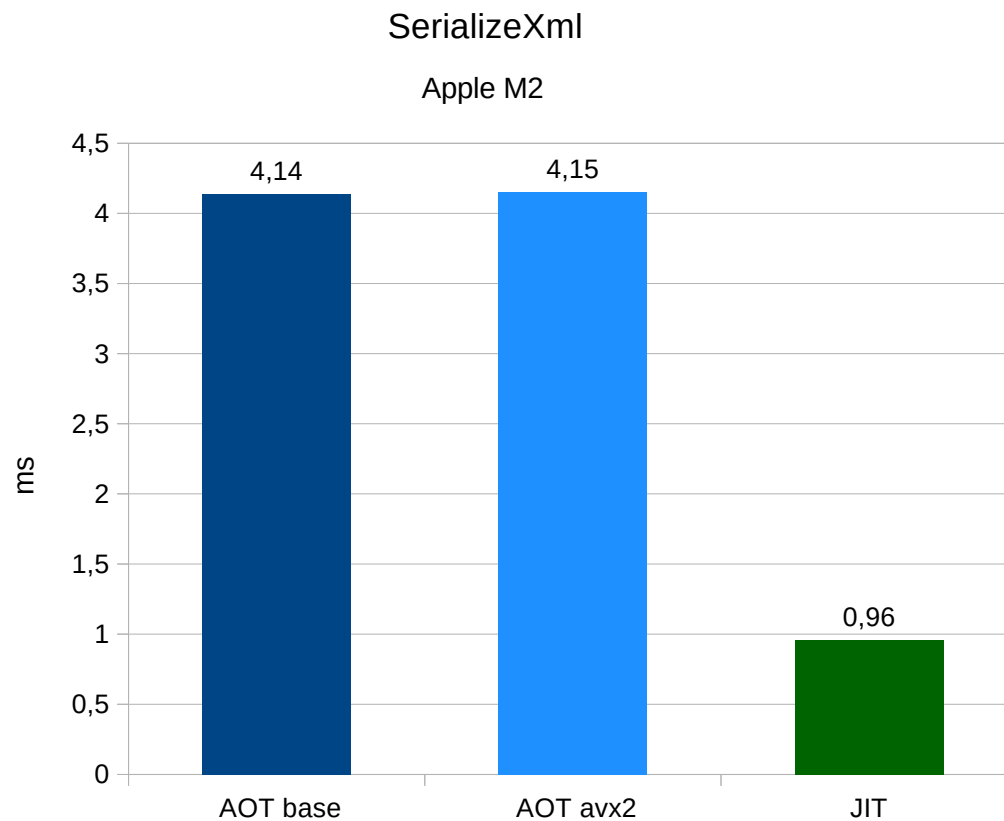
```
[Benchmark]
[RequiresUnreferencedCode("...")]
[RequiresDynamicCode("...")]
public void SerializeXmlInStringWriter()
{
    var stringWriter = new StringWriter();
    XmlSerializer?.Serialize(stringWriter, PnrList);
    Res = stringWriter.ToString();
}

[Benchmark]
[RequiresUnreferencedCode("...")]
[RequiresDynamicCode("...")]
public void DeserializeXml()
{
    PnrList2 = XmlSerializer?.Deserialize(new StringReader(Xml)) as PnrList;
}
```

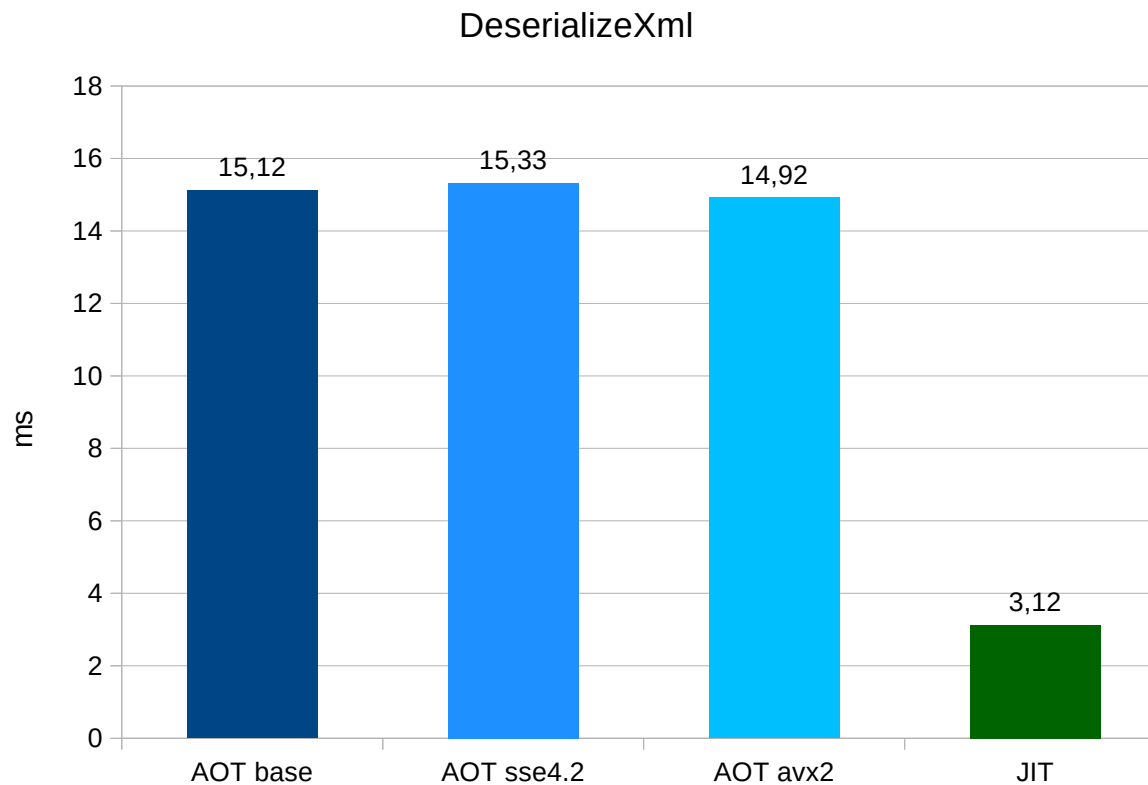
Сериализация XML (reflection)



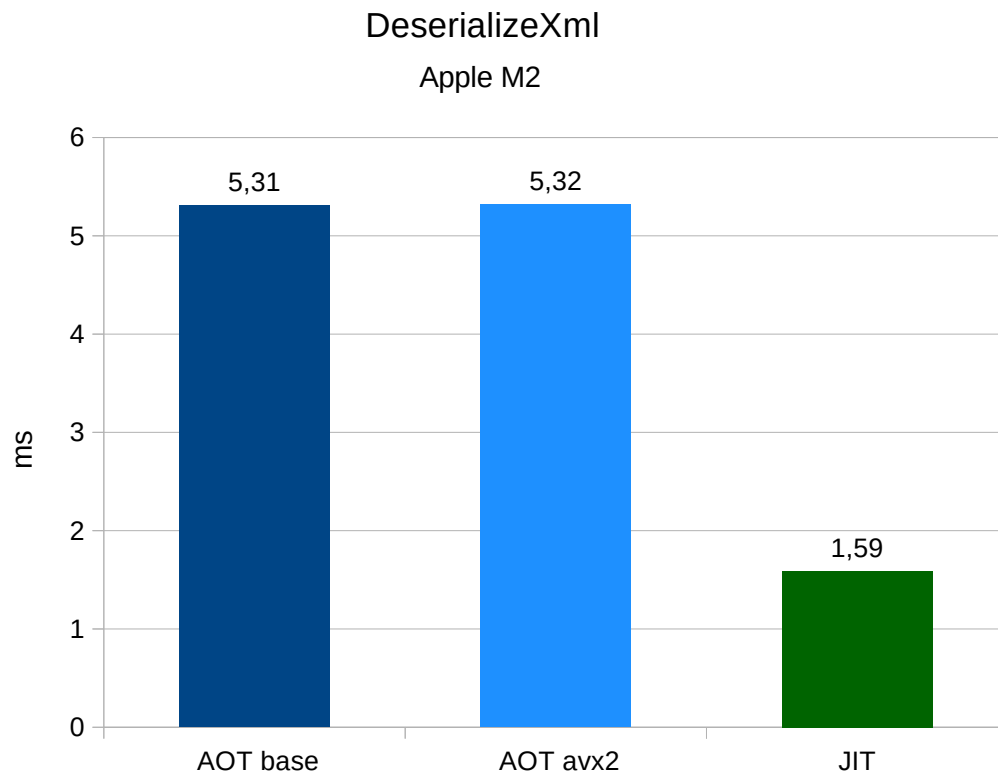
Сериализация XML (reflection)



Десериализация XML (reflection)

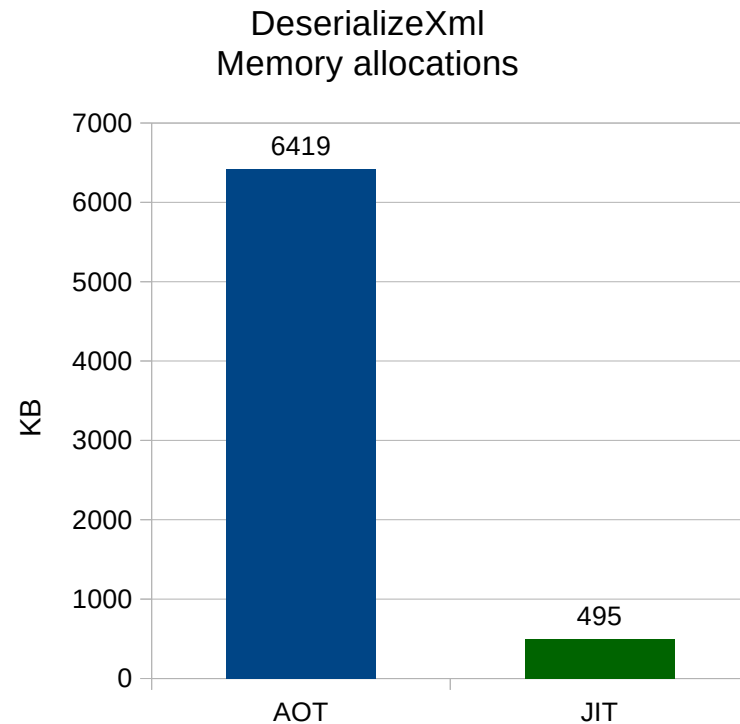
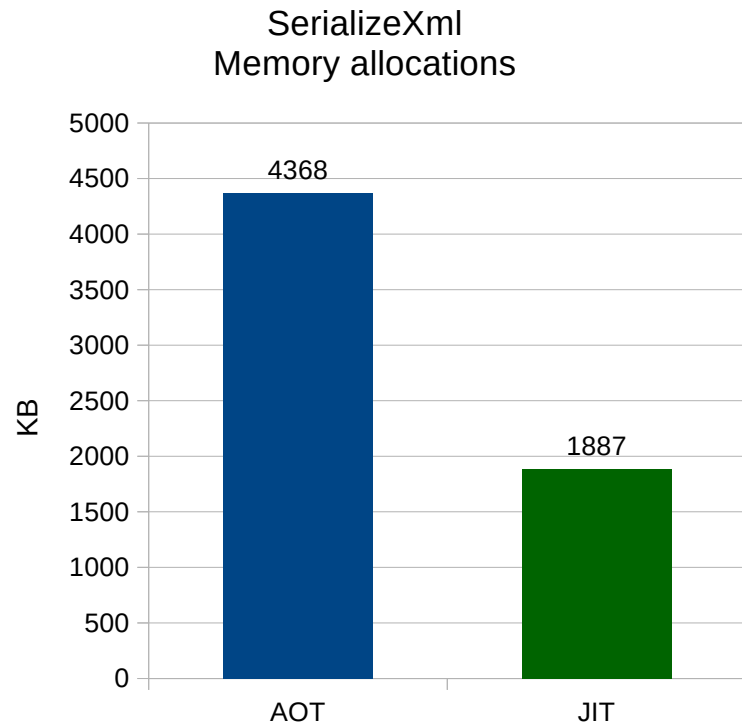


Десериализация XML (reflection)



Сериализация и десериализация XML

Выделение памяти



Native AOT: Скорость простого консольного приложения

```
public static void Main(string[] args)
{
    var json = File.ReadAllText(args[0]);

    var pnrList = JsonSerializer.Deserialize<PnrList>(json, PnrListContext.Default.PnrList);

    var flighsByFlightNumber = pnrList!.Pnrs
        .SelectMany(x => x.Flights.Select(f => new { f.FlightNumber, Pnr = x }))
        .GroupBy(x => x.FlightNumber)
        .Where(x => x.Count() > 1)
        .OrderBy(x => x.Key)
        .Select(x => x.Key)
        .ToArray();

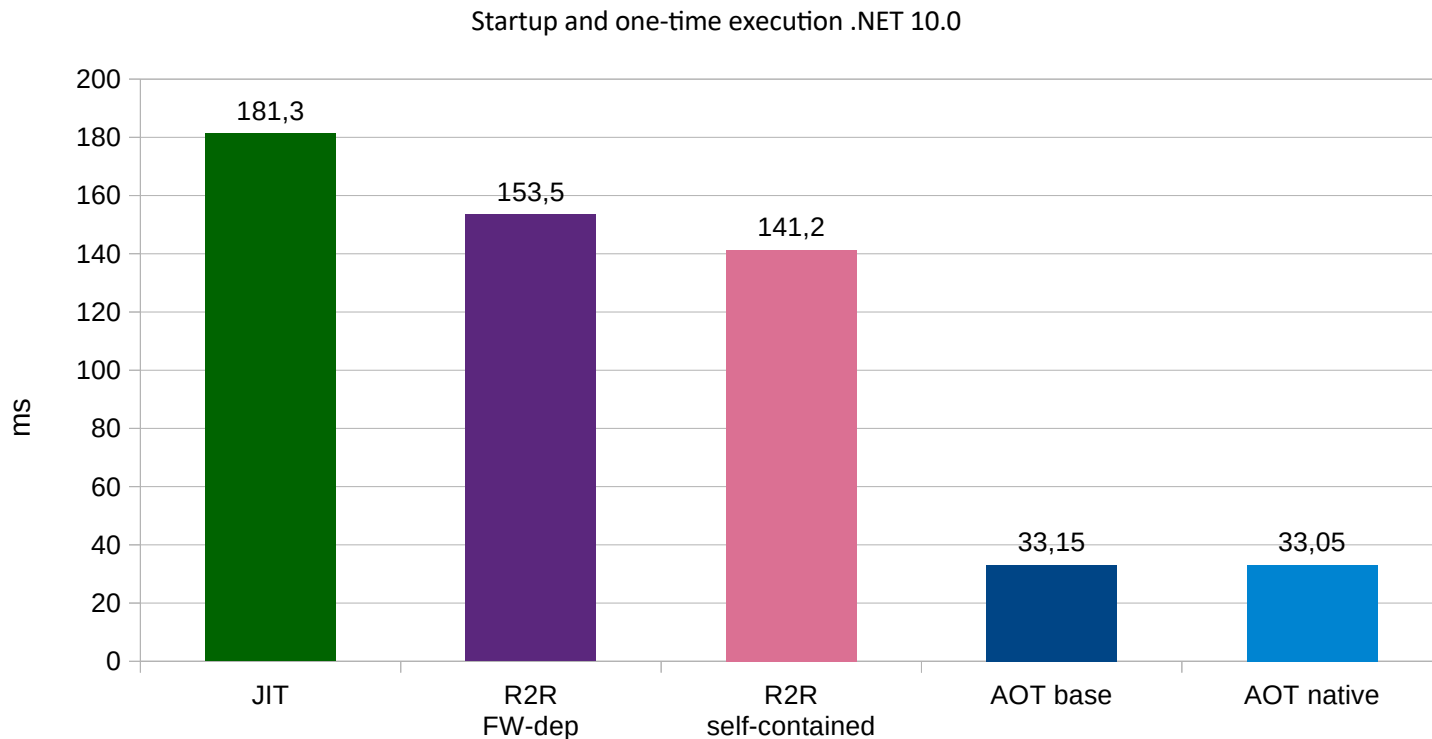
    var str2 = JsonSerializer.Serialize(flighsByFlightNumber, StringArrayContext.Default.StringArray);

    Console.WriteLine(str2.Substring(1, 7));
}
```

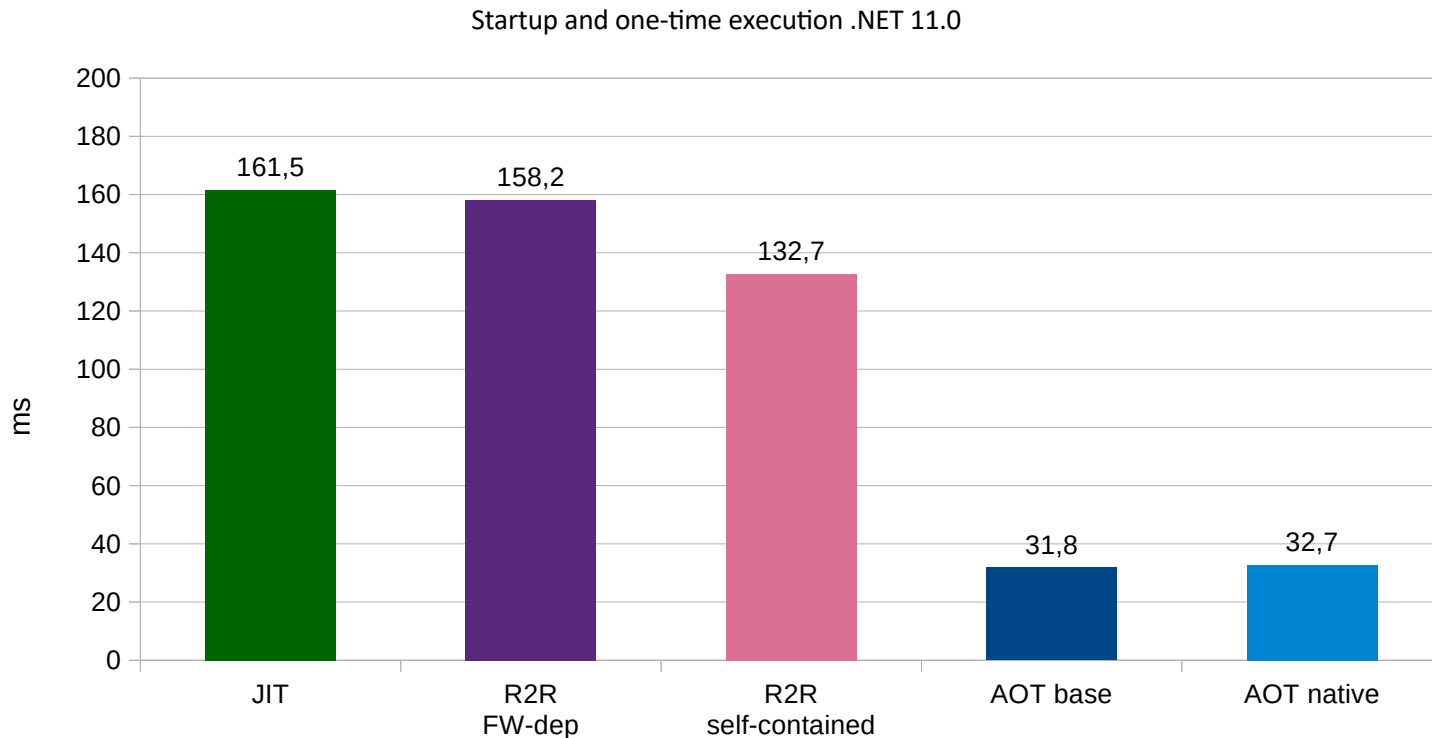
Native AOT: Скорость простого консольного приложения

- JIT
- ReadyToRun framework-dependent
- ReadyToRun self-contained
- Native AOT **base**
- Native AOT **native** (лучший набор команд для данного процессора - AVX2+)

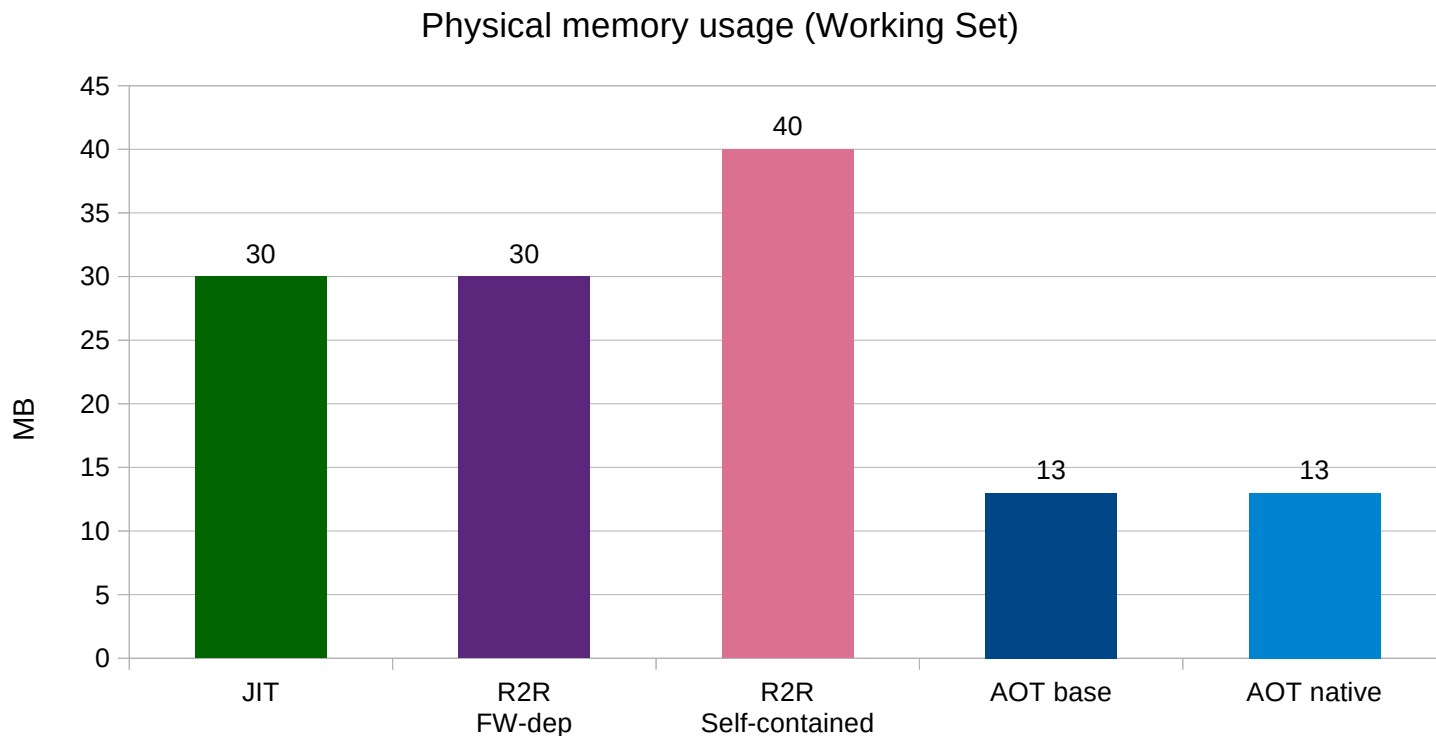
АОТ: Скорость выполнения простого консольного приложения .NET 10



АОТ: Скорость выполнения простого консольного приложения .NET 11



АОТ: Физическая память процесса .NET 10, .NET 11



АОТ: Скорость простого консольного приложения

- Размер .exe-файла — 4 MB
- Приложение АОТ (.NET10) работает даже под Windows 7 x64!

Выводы

- JIT генерирует более производительный код **на длинных дистанциях** в большинстве сценариев
- Адаптирует производительность под конкретный процессор с использованием его расширенных наборов команд (AVX2 и др.)
- IL-бинарники портируемые и универсальные
- Но более медленные при запуске
- R2R ускоряет запуск, но не очень существенно

Выводы

- Native AOT-бинарники выигрывают в несколько раз в скорости запуска и исполнения простых, «короткоживущих» утилит
- Аллоцируют на 15 МБ меньше физической памяти при запуске
- Привязаны к платформе (процессор и ОС)
- С помощью свойства **<IlcInstructionSet>** можно выбрать набор команд и сгенерировать более быстрый код для современных процессоров
- Каждое решение (JIT или Native AOT) имеет свою сферу применения

JIT vs NativeAOT

Характеристика	.NET JIT	Native AOT
Скорость запуска (Cold Start)	Медленнее (нужен прогрев, время на компиляцию)	Очень быстро
Оптимизация под CPU	Динамическая (на целевой машине) по мере необходимости	Статическая (задаётся при сборке)
Потребление памяти	Выше при старте (+runtime), ниже в работе	Ниже при старте , выше в работе
Рантайм	Нужен .NET рантайм (или self-contained)	Не нужен (автономный)
Поддержка рефлексии и динамики	Полная	Сильно ограничена
Портируемость	Высокая: один IL для любой архитектуры CPU и ОС	Привязан к архитектуре CPU и ОС на уровне API
Применение	Десктоп, Web API, разработка	Serverless, CLI, High-perf

ССЫЛКИ

- **dotnet runtime 11.0 with restored SSE2 baseline**
<https://github.com/ymalich/runtime-sse2>
- **TestAOT benchmarks**
<https://github.com/ymalich/TestAOT-for-DotNext2026>

ССЫЛКИ

- **Optimizing programs targeting Native AOT**
<https://github.com/dotnet/runtime/blob/main/src/coreclr/nativeaot/docs/optimizing.md>
- **Minimum hardware requirements updated**
<https://learn.microsoft.com/en-us/dotnet/core/compatibility/jit/11/minimum-hardware-requirements>
- **Updating the managed baselines to x86-64-v2 and armv8-a + lse**
<https://github.com/dotnet/runtime/pull/118101>
- **Not possible to specify `IlcInstructionSet < sse4.2`**
<https://github.com/dotnet/runtime/issues/119353>

Спасибо за внимание!

Вопросы?

Контакты:

email: yurymalich@yandex.ru



[@Yury_Malich](https://t.me/Yury_Malich)



<https://github.com/ymalich>