

Теория и практика .NET-бенчмаркинга

Андрей Акиньшин, JetBrains

Санкт-Петербург, 14.09.2017

Часть 1

Почему мы об этом говорим?

Люди любят бенчмаркать



Search

35,747 results

Типичный вопрос

Почему мой бенчмарк работает криво?



Я тут написал бенчмарк, замерял через Stopwatch:

69

<Тут бы мог быть код вашего бенчмарка>



c#

.net



share edit close delete flag

5



Дядя Фёдор

3 Answers

active

oldest

votes



Неправильно ты, дядя Фёдор, код бенчмаркаешь...

71

*<А тут могло бы быть подробное объяснение,
почему же дядя Фёдор криво бенчмаркает>*



share edit flag



Матроскин

Некоторые делают выводы и пишут статьи

Хабрахабр

Публикации

Хабы

Компании

Пользователи

Песочница

18 апреля в 17:52

Разработка → Почему JavaScript работает быстрее, чем C++?

6 сентября 2015 в 19:50

Разработка → Сравнение производительности C++ и C#

8 августа 2009 в 15:47

Разработка → Производительность C++ vs. Java vs. PHP vs. Python. Тест «в лоб»

14 января 2012 в 05:30

Разработка → Почему C быстрее Java (с точки зрения Java-разработчика) перевод

NodeJS outperforming .NET Core? (self.dotnet)

C#/.NET Core

```
var sw = new Stopwatch();
sw.Start();
var n = 0;
for (var j = 0; j < 10; j++)
{
    for (var i = 0; i < 1000000000; i++)
    {
        n += i;
    }
}
sw.Stop();
Console.WriteLine("Elapsed Time: " + sw.ElapsedMilliseconds);
```

.NET Core Result: 19,035 milliseconds.

TypeScript/NodeJS

```
console.log("Elapsed Time:", Stopwatch.measure(()=>{
    let n = 0;
    for (let j = 0; j < 10; j++)
    {
        for (let i = 0; i < 1000000000; i++)
        {
            n += i;
        }
    }
}).total.milliseconds);
// Latest: 10M loops per second.
```

NodeJS Result: 9,918 milliseconds

NodeJS outperforming .NET Core? (self.dotnet)

C#/.NET Core

```
var sw = new Stopwatch();
sw.Start();
var n = 0;
for (var j = 0; j < 10; j++)
{
    for (var i = 0; i < 1000000000; i++)
    {
        n += i;
    }
}
sw.Stop();
Console.WriteLine("Elapsed Time: " + sw.ElapsedMilliseconds);
```

.NET Core Result: 19,035 milliseconds.

TypeScript/NodeJS

```
console.log("Elapsed Time:", Stopwatch.measure(() => {
    let n = 0;
    for (let j = 0; j < 10; j++)
    {
        for (let i = 0; i < 1000000000; i++)
        {
            n += i;
        }
    }
})).total.milliseconds);
// Latest: 10M loops per second.
```

NodeJS Result: 9,918 milliseconds

 **SikhGamer** 101 points 1 month ago

You are running the NET Core in Debug mode. If I take your example and run it in Debug mode I get:-

Elapsed Time: 19,185.8573 milliseconds

If I run it in Release mode, I get:-

Elapsed Time: 2,514.0627 milliseconds

[permalink](#) [embed](#) [save](#) [report](#) [give gold](#) [reply](#)

Применения бенчмарков

- Performance analysis

- Performance analysis
 - Сравнение алгоритмов

- Performance analysis
 - Сравнение алгоритмов
 - Оценка улучшений производительности

- Performance analysis
 - Сравнение алгоритмов
 - Оценка улучшений производительности
 - Анализ регрессии

- Performance analysis
 - Сравнение алгоритмов
 - Оценка улучшений производительности
 - Анализ регрессии
 - ...

- Performance analysis
 - Сравнение алгоритмов
 - Оценка улучшений производительности
 - Анализ регрессии
 - ...
- Научный интерес

- Performance analysis
 - Сравнение алгоритмов
 - Оценка улучшений производительности
 - Анализ регрессии
 - ...
- Научный интерес
- Маркетинг

- Performance analysis
 - Сравнение алгоритмов
 - Оценка улучшений производительности
 - Анализ регрессии
 - ...
- Научный интерес
- Маркетинг
- Весёлое времяпрепровождение 😊

Часть 2

Общая методология

План performance-работ

1 Поставить задачу

- 1 Поставить задачу
- 2 Выбрать метрики

- 1 Поставить задачу
- 2 Выбрать метрики
- 3 Выбрать инструмент

- 1 Поставить задачу
- 2 Выбрать метрики
- 3 Выбрать инструмент
- 4 Провести эксперимент

- 1 Поставить задачу
- 2 Выбрать метрики
- 3 Выбрать инструмент
- 4 Провести эксперимент
- 5 Получить результаты

- 1 Поставить задачу
- 2 Выбрать метрики
- 3 Выбрать инструмент
- 4 Провести эксперимент
- 5 Получить результаты
- 6 **Выполнить анализ и сделать выводы**

- 1 Поставить задачу
- 2 Выбрать метрики
- 3 Выбрать инструмент
- 4 Провести эксперимент
- 5 Получить результаты
- 6 **Выполнить анализ и сделать выводы**

Анализ полученных данных — самый важный этап

Виды performance-работ

- Profiling

- Profiling
- Monitoring

- Profiling
- Monitoring
- Performance tests

- Profiling
- Monitoring
- Performance tests
- Benchmarking (micro/macro)

- Profiling
- Monitoring
- Performance tests
- Benchmarking (micro/macro)
- ...

C# compiler	старый csc / Roslyn
CLR	CLR2 / CLR4 / CoreCLR / Mono
OS	Windows / Linux / MacOS / FreeBSD
JIT	LegacyJIT-x86 / LegacyJIT-x64 / RyuJIT-x64
GC	MS (разные CLR) / Mono (Boehm/Sgen)
Toolchain	JIT / NGen / .NET Native
Hardware	тысячи его
...	...

C# compiler	старый csc / Roslyn
CLR	CLR2 / CLR4 / CoreCLR / Mono
OS	Windows / Linux / MacOS / FreeBSD
JIT	LegacyJIT-x86 / LegacyJIT-x64 / RyuJIT-x64
GC	MS (разные CLR) / Mono (Boehm/Sgen)
Toolchain	JIT / NGen / .NET Native
Hardware	тысячи его
...	...

И не забываем про версии, много-много версий

Советы по запуску бенчмарков:

Советы по запуску бенчмарков:

- Release build (*Top 1 бенчмарк-ошибок*)

Советы по запуску бенчмарков:

- Release build (*Top 1 бенчмарк-ошибок*)
- Без дебаггера

Советы по запуску бенчмарков:

- Release build (*Top 1 бенчмарк-ошибок*)
- Без дебаггера
- Выключите другие приложения

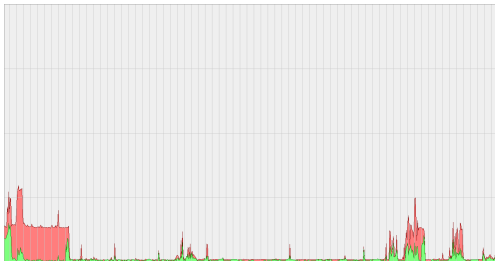
Советы по запуску бенчмарков:

- Release build (*Top 1 бенчмарк-ошибок*)
- Без дебаггера
- Выключите другие приложения
- Используйте максимальную производительность

Советы по запуску бенчмарков:

- Release build (*Top 1 бенчмарк-ошибок*)
- Без дебаггера
- Выключите другие приложения
- Используйте максимальную производительность

Загруженность CPU:



dotnet / **BenchmarkDotNet**

Unwatch ▾

171

★ Unstar

2,250

🔗 Fork

222

📄 1,125 commits

🌿 5 branches

📦 32 releases

👤 41 contributors

🏢 MIT

- Standard benchmarking routine: generating an isolated project per each benchmark method; auto-selection of iteration amount; warmup; overhead evaluation; statistics calculation; and so on.
- Supported runtimes: Full .NET Framework (4.6+), .NET Core (1.1+), Mono
- Supported languages: C#, F#, and Visual Basic
- Supported OS: Windows, Linux, MacOS
- Easy way to compare different environments (`x86` vs `x64` , `LegacyJit` vs `RyuJit` , and so on; see: [Jobs](#))
- Reports: markdown, csv, html, plain text, png plots.
- Advanced features: [Baseline](#), [Params](#)
- Powerful diagnostics based on ETW events (see [BenchmarkDotNet.Diagnostics.Windows](#))

.NET Foundation

This project is supported by the [.NET Foundation](#).

Часть 3

Таймеры

DateTime vs Stopwatch

```
var start = DateTime.Now;  
Foo();  
var finish = DateTime.Now;  
var time = (finish - start).TotalMilliseconds;
```

VS

```
var sw = Stopwatch.StartNew();  
Foo();  
sw.Stop();  
var time = sw.ElapsedMilliseconds;
```

Характеристики таймеров

- **Монотонность**
замеры должны не уменьшаться

- **Монотонность**

замеры должны не уменьшаться

- **Resolution**

минимальное положительное время между замерами

- **Монотонность**

замеры должны неуменьшаться

- **Resolution**

минимальное положительное время между замерами

- **Latency**

время на получение замера

OS	Implementation
Windows	GetSystemTimeAsFileTime
Linux	gettimeofday

OS	Implementation
Windows	GetSystemTimeAsFileTime
Linux	gettimeofday

	OS	Runtime	Time*
Latency	Windows	Full/Core	$\approx 7-8\text{ns}$
Latency	Windows	Mono	$\approx 30-31\text{ns}$
Resolution	Windows	Any	$\approx 0.5..15.625\text{ms}$

OS	Implementation
Windows	GetSystemTimeAsFileTime
Linux	gettimeofday

	OS	Runtime	Time*
Latency	Windows	Full/Core	$\approx 7-8\text{ns}$
Latency	Windows	Mono	$\approx 30-31\text{ns}$
Resolution	Windows	Any	$\approx 0.5..15.625\text{ms}$
Latency	Linux	Mono	$\approx 26-30\text{ns}$
Resolution	Linux	Mono	$\approx 1\mu\text{s}$

OS	Implementation
Windows	GetSystemTimeAsFileTime
Linux	gettimeofday

	OS	Runtime	Time*
Latency	Windows	Full/Core	$\approx 7-8\text{ns}$
Latency	Windows	Mono	$\approx 30-31\text{ns}$
Resolution	Windows	Any	$\approx 0.5..15.625\text{ms}$
Latency	Linux	Mono	$\approx 26-30\text{ns}$
Resolution	Linux	Mono	$\approx 1\mu\text{s}$

* Intel i7-4702MQ CPU 2.20GHz

См. также: <http://aakinshin.net/en/blog/dotnet/datetime/>

Stopwatch.GetTimestamp()

Hardware timers

- NA
- TSC (Variant / Constant / Invariant)
- ACPI PM (Freq = 3.579545 MHz)
- HPET (Freq = 14.31818 MHz)

Hardware timers

- NA
- TSC (Variant / Constant / Invariant)
- ACPI PM (Freq = 3.579545 MHz)
- HPET (Freq = 14.31818 MHz)

Implementation

- Windows: QueryPerformanceCounter
- Linux: clock_gettime / mach_absolute_time /
gettimeofday

Stopwatch.GetTimestamp()

Runtime	OS	Timer	1 tick	Latency	Resolution
Full	Win	TSC	300-400ns	15-18ns	300-400ns
Full	Win	HPET	69.8ns	500-800ns	≈Latency
Full	Win	NA	100ns	7-10ns	0.5-55ms
Mono	Win	TSC	100ns	35-45ns	300-400ns
Mono	Win	HPET	100ns	500-800ns	≈Latency
Mono	Win	NA	100ns	30-40ns	0.5-55ms
Core	Linux	TSC	1ns	30-35ns	≈Latency
Core	Linux	HPET/ACPI	1ns	500-800ns	≈Latency
Mono	Linux	TSC	100ns	20-25ns	100ns
Mono	Linux	HPET/ACPI	100ns	500-800ns	≈Latency

Intel i7-4702MQ CPU 2.20GHz

См. также: <http://aakinshin.net/en/blog/dotnet/stopwatch/>

Важно помнить про таймеры

Важно помнить про таймеры

- Важно понимать значения Latency и Resolution

Важно помнить про таймеры

- Важно понимать значения Latency и Resolution
- 1 tick \neq Resolution

Важно помнить про таймеры

- Важно понимать значения Latency и Resolution
- 1 tick \neq Resolution
- Время может идти назад ☹

Важно помнить про таймеры

- Важно понимать значения Latency и Resolution
- 1 tick \neq Resolution
- Время может идти назад ☹
- Два последовательных замера могут быть равны

Важно помнить про таймеры

- Важно понимать значения Latency и Resolution
- 1 tick \neq Resolution
- Время может идти назад ☹
- Два последовательных замера могут быть равны
- Два последовательных замера могут различаться на миллисекунды

Важно помнить про таймеры

- Важно понимать значения Latency и Resolution
- 1 tick \neq Resolution
- Время может идти назад ☹
- Два последовательных замера могут быть равны
- Два последовательных замера могут различаться на миллисекунды

Тем временем...

Search

[c#] datetime.now

search

19,597 results

relevance

newest

votes

active

Часть 4

Количество итераций

Плохой бенчмарк

```
// Resolution(Stopwatch) = 466 ns  
// Latency(Stopwatch) = 18 ns  
var sw = Stopwatch.StartNew();  
Foo(); // 100 ns  
sw.Stop();  
WriteLine(sw.ElapsedMilliseconds);
```

Плохой бенчмарк

```
// Resolution(Stopwatch) = 466 ns  
// Latency(Stopwatch) = 18 ns  
var sw = Stopwatch.StartNew();  
Foo(); // 100 ns  
sw.Stop();  
WriteLine(sw.ElapsedMilliseconds);
```

Небольшое улучшение

```
var sw = Stopwatch.StartNew();  
for (int i = 0; i < N; i++) // (N * 100 + eps) ns  
    Foo();  
sw.Stop();  
var total = sw.ElapsedTicks / Stopwatch.Frequency;  
WriteLine(total / N);
```

Запустим бенчмарк несколько раз:

```
int[] x = new int[128 * 1024 * 1024];  
for (int iter = 0; iter < 5; iter++)  
{  
    var sw = Stopwatch.StartNew();  
    for (int i = 0; i < x.Length; i += 16)  
        x[i]++;  
    sw.Stop();  
    Console.WriteLine(sw.ElapsedMilliseconds);  
}
```

Запустим бенчмарк несколько раз:

```
int[] x = new int[128 * 1024 * 1024];  
for (int iter = 0; iter < 5; iter++)  
{  
    var sw = Stopwatch.StartNew();  
    for (int i = 0; i < x.Length; i += 16)  
        x[i]++;  
    sw.Stop();  
    Console.WriteLine(sw.ElapsedMilliseconds);  
}
```

Результат:

```
176  
81  
62  
62  
62
```

Несколько запусков метода

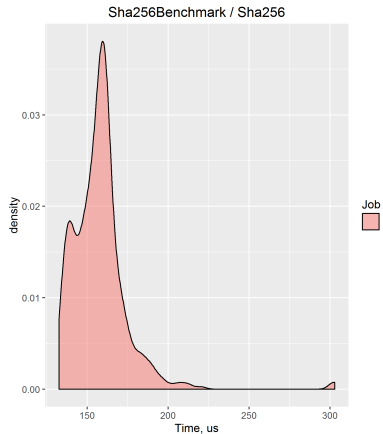
```
Run 01 : 529.8674 ns/op
Run 02 : 532.7541 ns/op
Run 03 : 558.7448 ns/op
Run 04 : 555.6647 ns/op
Run 05 : 539.6401 ns/op
Run 06 : 539.3494 ns/op
Run 07 : 564.3222 ns/op
Run 08 : 551.9544 ns/op
Run 09 : 550.1608 ns/op
Run 10 : 533.0634 ns/op
```

Несколько запусков бенчмарка



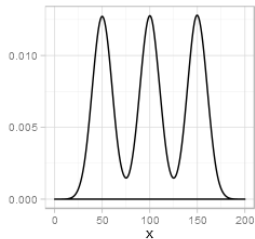
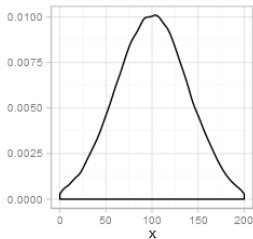
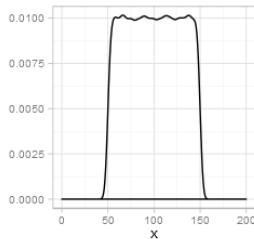
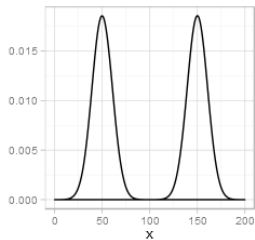
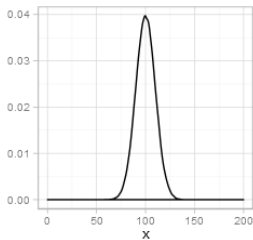
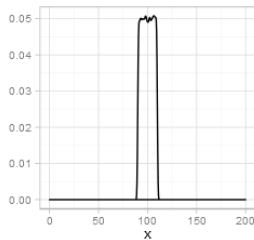
BenchmarkDotNet v0.9.9

Центральная предельная теорема спешит на помощь!



BenchmarkDotNet v0.9.9

Но есть и сложные случаи



Часть 5

Работаем с памятью

Сумма элементов массива

```
const int N = 1024;  
int[,] a = new int[N, N];
```

[Benchmark]

```
public double SumIj()  
{  
    var sum = 0;  
    for (int i = 0; i < N; i++)  
        for (int j = 0; j < N; j++)  
            sum += a[i, j];  
    return sum;  
}
```

[Benchmark]

```
public double SumJi()  
{  
    var sum = 0;  
    for (int j = 0; j < N; j++)  
        for (int i = 0; i < N; i++)  
            sum += a[i, j];  
    return sum;  
}
```

Сумма элементов массива

```
const int N = 1024;  
int[,] a = new int[N, N];
```

[Benchmark]

```
public double SumIj()  
{  
    var sum = 0;  
    for (int i = 0; i < N; i++)  
        for (int j = 0; j < N; j++)  
            sum += a[i, j];  
    return sum;  
}
```

[Benchmark]

```
public double SumJi()  
{  
    var sum = 0;  
    for (int j = 0; j < N; j++)  
        for (int i = 0; i < N; i++)  
            sum += a[i, j];  
    return sum;  
}
```

	SumIj	SumJi
LegacyJIT-x86	≈1.3ms	≈4.0ms

Часть 6

Работаем с условными переходами

Branch prediction

```
const int N = 32767;
int[] sorted, unsorted; // random numbers [0..255]
private static int Sum(int[] data)
{
    int sum = 0;
    for (int i = 0; i < N; i++)
        if (data[i] >= 128)
            sum += data[i];
    return sum;
}
```

[Benchmark]

```
public int Sorted()
{
    return Sum(sorted);
}
```

[Benchmark]

```
public int Unsorted()
{
    return Sum(unsorted);
}
```

Branch prediction

```
const int N = 32767;
int[] sorted, unsorted; // random numbers [0..255]
private static int Sum(int[] data)
{
    int sum = 0;
    for (int i = 0; i < N; i++)
        if (data[i] >= 128)
            sum += data[i];
    return sum;
}
```

[Benchmark]

```
public int Sorted()
{
    return Sum(sorted);
}
```

[Benchmark]


```
public int Unsorted()
{
    return Sum(unsorted);
}
```

	Sorted	Unsorted
LegacyJIT-x86	$\approx 20\mu s$	$\approx 139\mu s$

Часть 7

Inlining

Исходники .NET Framework



Microsoft Reference Source .NET Framework 4.5.2 [Download](#) [Feedback](#) [License](#) [Help](#)

- currency.cs
- currenttimezone.cs
- datamismatchexception.cs
- datetime.cs
- datetimekind.cs
- datetimeoffset.cs
- dayofweek.cs
- dbnull.cs
- decimal.cs**
- defaultbinder.cs
- delegate.cs
- delegateserializationholder.cs
- dividebyzeroexception.cs
- dllnotfoundexception.cs
- double.cs
- duplicatewaitobjectexception.cs
- empty.cs
- entrypointnotfoundexception.cs
- enum.cs
- environment.cs

```
158 // Constructs a Decimal from an integer value.
159 //
160 public Decimal(int value) {
161     // JIT today can't inline methods that contains "starg" opcode.
162     // For more details, see DevDiv Bugs 81184: x86 JIT CQ: Removing the inline striction of "starg".
163     int value_copy = value;
164     if (value_copy >= 0) {
165         flags = 0;
166     }
167     else {
168         flags = SignMask;
169         value_copy = -value_copy;
170     }
171     lo = value_copy;
172     mid = 0;
173     hi = 0;
174 }
175
176 // Constructs a Decimal from an unsigned integer value.
177 //
178 [CLSCompliant(false)]
179 public Decimal(uint value) {
180     flags = 0;
181     lo = (int) value;
182     mid = 0;
183     hi = 0;
184 }
```


Inlining — это сложно

```
// mscorlib/system/decimal.cs,158
// Constructs a Decimal from an integer value.
public Decimal(int value) {
    // JIT today can't inline methods that contains "starg"
    // opcode. For more details, see DevDiv Bugs 81184:
    // x86 JIT CQ: Removing the inline striction of "starg".
    int value_copy = value;
    if (value_copy >= 0) {
        flags = 0;
    } else {
        flags = SignMask;
        value_copy = -value_copy;
    }
    lo = value_copy;
    mid = 0;
    hi = 0;
}
```

```
[Benchmark]
int Calc() => WithoutStarg(0x11) + WithStarg(0x12);

int WithoutStarg(int value) => value;

int WithStarg(int value)
{
    if (value < 0)
        value = -value;
    return value;
}
```

```
[Benchmark]
int Calc() => WithoutStarg(0x11) + WithStarg(0x12);

int WithoutStarg(int value) => value;

int WithStarg(int value)
{
    if (value < 0)
        value = -value;
    return value;
}
```

LegacyJIT-x86	LegacyJIT-x64	RyuJIT-x64
≈1.7ns	0	≈1.7ns

LegacyJIT-x64

```
; LegacyJIT-x64  
mov     ecx, 23h  
ret
```

LegacyJIT-x64

```
; LegacyJIT-x64  
mov     ecx, 23h  
ret
```

RyuJIT-x64

```
// Inline expansion aborted due to opcode  
// [06] OP_starg.s in method  
// Program:WithStarg(int):int:this
```

Часть 8

Constant folding

Учимся извлекать корни

```
double Sqrt13() =>  
    Math.Sqrt(1) + Math.Sqrt(2) + Math.Sqrt(3) + /* ... */  
    + Math.Sqrt(13);
```

VS

```
double Sqrt14() =>  
    Math.Sqrt(1) + Math.Sqrt(2) + Math.Sqrt(3) + /* ... */  
    + Math.Sqrt(13) + Math.Sqrt(14);
```

Учимся извлекать корни

```
double Sqrt13() =>  
    Math.Sqrt(1) + Math.Sqrt(2) + Math.Sqrt(3) + /* ... */  
    + Math.Sqrt(13);
```

VS

```
double Sqrt14() =>  
    Math.Sqrt(1) + Math.Sqrt(2) + Math.Sqrt(3) + /* ... */  
    + Math.Sqrt(13) + Math.Sqrt(14);
```

	RyuJIT-x64
Sqrt13	≈91ns
Sqrt14	0 ns

RyuJIT-x64, Sqrt13

```

vsqrtsd    xmm0,xmm0,mmword ptr [7FF94F9E4D28h]
vsqrtsd    xmm1,xmm0,mmword ptr [7FF94F9E4D30h]
vaddsd     xmm0,xmm0,xmm1
vsqrtsd    xmm1,xmm0,mmword ptr [7FF94F9E4D38h]
vaddsd     xmm0,xmm0,xmm1
vsqrtsd    xmm1,xmm0,mmword ptr [7FF94F9E4D40h]
vaddsd     xmm0,xmm0,xmm1
vsqrtsd    xmm1,xmm0,mmword ptr [7FF94F9E4D48h]
vaddsd     xmm0,xmm0,xmm1
vsqrtsd    xmm1,xmm0,mmword ptr [7FF94F9E4D50h]
vaddsd     xmm0,xmm0,xmm1
vsqrtsd    xmm1,xmm0,mmword ptr [7FF94F9E4D58h]
vaddsd     xmm0,xmm0,xmm1
vsqrtsd    xmm1,xmm0,mmword ptr [7FF94F9E4D60h]
vaddsd     xmm0,xmm0,xmm1
vsqrtsd    xmm1,xmm0,mmword ptr [7FF94F9E4D68h]
vaddsd     xmm0,xmm0,xmm1
vsqrtsd    xmm1,xmm0,mmword ptr [7FF94F9E4D70h]
vaddsd     xmm0,xmm0,xmm1
vsqrtsd    xmm1,xmm0,mmword ptr [7FF94F9E4D78h]
vaddsd     xmm0,xmm0,xmm1
vsqrtsd    xmm1,xmm0,mmword ptr [7FF94F9E4D80h]
vaddsd     xmm0,xmm0,xmm1
vsqrtsd    xmm1,xmm0,mmword ptr [7FF94F9E4D88h]
vaddsd     xmm0,xmm0,xmm1
ret

```

RyuJIT-x64, Sqrt14

```
vmovsd    xmm0, qword ptr [7FF94F9C4C80h]  
ret
```

Большое дерево выражения

```
* stmtExpr void (top level) (IL 0x000... ???)
|   /--* mathFN    double sqrt
|   | \--* dconst   double 13.0000000000000000
|   /--* +         double
|   | | /--* mathFN    double sqrt
|   | | | \--* dconst   double 12.0000000000000000
|   | | \--* +         double
|   | | | /--* mathFN    double sqrt
|   | | | | \--* dconst   double 11.0000000000000000
|   | | \--* +         double
|   | | | /--* mathFN    double sqrt
|   | | | | \--* dconst   double 10.0000000000000000
|   | | \--* +         double
|   | | | /--* mathFN    double sqrt
|   | | | | \--* dconst   double 9.0000000000000000
|   | | \--* +         double
|   | | | /--* mathFN    double sqrt
|   | | | | \--* dconst   double 8.0000000000000000
|   | | \--* +         double
|   | | | /--* mathFN    double sqrt
|   | | | | \--* dconst   double 7.0000000000000000
|   | | \--* +         double
|   | | | /--* mathFN    double sqrt
|   | | | | \--* dconst   double 6.0000000000000000
|   | | \--* +         double
|   | | | /--* mathFN    double sqrt
|   | | | | \--* dconst   double 5.0000000000000000
|   |
|   // ...
```

Constant folding в действии

```

N001 [000001]  dconst    1.0000000000000000 => $c0 {DblCns[1.000000]}
N002 [000002]  mathFN     => $c0 {DblCns[1.000000]}
N003 [000003]  dconst    2.0000000000000000 => $c1 {DblCns[2.000000]}
N004 [000004]  mathFN     => $c2 {DblCns[1.414214]}
N005 [000005]  +          => $c3 {DblCns[2.414214]}
N006 [000006]  dconst    3.0000000000000000 => $c4 {DblCns[3.000000]}
N007 [000007]  mathFN     => $c5 {DblCns[1.732051]}
N008 [000008]  +          => $c6 {DblCns[4.146264]}
N009 [000009]  dconst    4.0000000000000000 => $c7 {DblCns[4.000000]}
N010 [000010]  mathFN     => $c1 {DblCns[2.000000]}
N011 [000011]  +          => $c8 {DblCns[6.146264]}
N012 [000012]  dconst    5.0000000000000000 => $c9 {DblCns[5.000000]}
N013 [000013]  mathFN     => $ca {DblCns[2.236068]}
N014 [000014]  +          => $cb {DblCns[8.382332]}
N015 [000015]  dconst    6.0000000000000000 => $cc {DblCns[6.000000]}
N016 [000016]  mathFN     => $cd {DblCns[2.449490]}
N017 [000017]  +          => $ce {DblCns[10.831822]}
N018 [000018]  dconst    7.0000000000000000 => $cf {DblCns[7.000000]}
N019 [000019]  mathFN     => $d0 {DblCns[2.645751]}
N020 [000020]  +          => $d1 {DblCns[13.477573]}
...

```

Часть 9

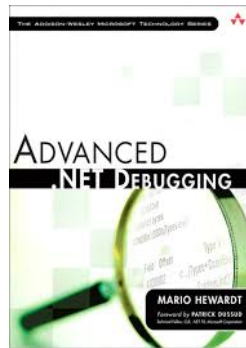
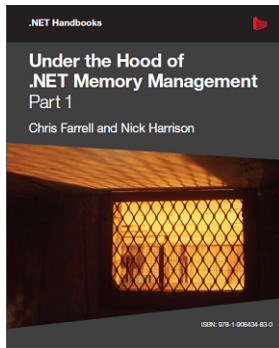
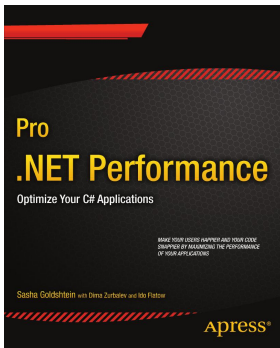
Заключение

- Все представленные выводы и бенчмарки могут быть враньём
- На вашем железе цифры могут быть другие, это нормально
- Использование BenchmarkDotNet не делает ваш бенчмарк правильным
- Использование самописных бенчмарков не делает выводы ложными

- Бенчмаркинг и прочие замеры производительности — это сложно
- Бенчмаркинг требует очень много сил, знаний, времени и нервов
- Бенчмарк без анализа — плохой бенчмарк

Методическая литература

Для успешных микробенчмарков нужно очень много знать:



+ 6292 - [::|||:] Поделиться

2014-12-22 12:45

#431616

xxx: Вот заводят люди себе семьи, находят девушек, обзаводятся хобби, а потом удивляются, почему они так плохо знают архитектуру x86_64.

Андрей Акиншин

<http://aakinshin.net>

<https://github.com/AndreyAkinshin>

https://twitter.com/andrey_akinshin

andrey.akinshin@gmail.com