

Теория и практика .NET-бенчмаркинга

Андрей Акиньшин, JetBrains

Москва, 25.01.2017

Часть 1

Теория

Часть 1.1

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

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 1.2

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)
- ...

Исходный код

Исходный код

X

Окружение

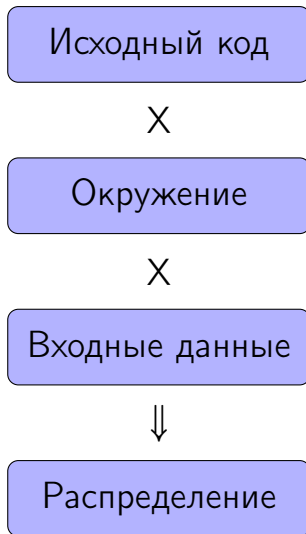
Исходный код

X

Окружение

X

Входные данные



Непонимание проблематики влечёт за собой следующие проблемы:

Непонимание проблематики влечёт за собой следующие проблемы:

- Легко обмануть себя, сделать неправильные выводы, принять вредные бизнес-решения

Непонимание проблематики влечёт за собой следующие проблемы:

- Легко обмануть себя, сделать неправильные выводы, принять вредные бизнес-решения
- Легко пропустить важную конфигурацию, которая испортит жизнь в продакшене

Непонимание проблематики влечёт за собой следующие проблемы:

- Легко обмануть себя, сделать неправильные выводы, принять вредные бизнес-решения
- Легко пропустить важную конфигурацию, которая испортит жизнь в продакшене
- Легко повестись на кривые бенчмарки или чёрный маркетинг

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

Почитаем Intel® 64 and IA-32 Architectures Optimization Reference Manual:

2.1.3 THE SKYLAKE MICROARCHITECTURE: Cache and Memory Subsystem

- Simultaneous handling of more loads and stores enabled by enlarged buffers.
- Page split load penalty down from 100 cycles in previous generation to 5 cycles.
- L3 write bandwidth increased from 4 cycles per line in previous generation to 2 per line.
- L2 associativity changed from 8 ways to 4 ways.

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

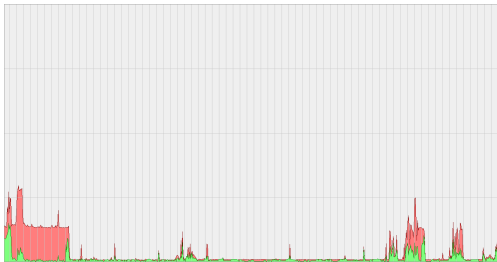
Советы по запуску бенчмарков:

- 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:



Требования к бенчмарку

- Надёжность

- Надёжность
- Точность

- Надёжность
- Точность
- Воспроизводимость

- Надёжность
- Точность
- Воспроизводимость
- Изолированность

- Надёжность
- Точность
- Воспроизводимость
- Изолированность
- Переносимость

- Надёжность
- Точность
- Воспроизводимость
- Изолированность
- Переносимость
- Простота

- Надёжность
- Точность
- Воспроизводимость
- Изолированность
- Переносимость
- Простота
- Честность

dotnet / **BenchmarkDotNet**

Unwatch ▾

128

★ Unstar

1,577

Fork

153

781 commits

3 branches

25 releases

26 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, .NET Core (RTM), 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 1.3

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 неуменьшаться

- **Resolution**

минимальное положительное время между замерами

- **Монотонность**

замеры должны не уменьшаться

- **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

search

19,597 results

relevance

newest

votes

active

Часть 1.4

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

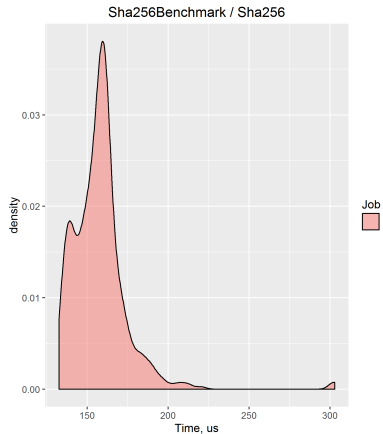
```
Run 01 : 529.8674 ns/op
Run 02 : 532.7541 ns/op
Run 03 : 558.7448 ns/op
Run 04 : 555.6647 ns/op
Run 05 : 539.6401 ns/op
Run 06 : 539.3494 ns/op
Run 07 : 564.3222 ns/op
Run 08 : 551.9544 ns/op
Run 09 : 550.1608 ns/op
Run 10 : 533.0634 ns/op
```

Несколько запусков бенчмарка



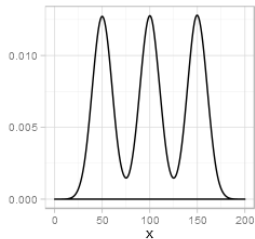
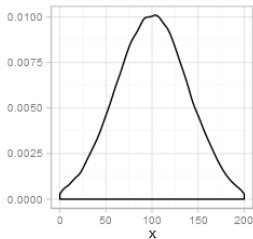
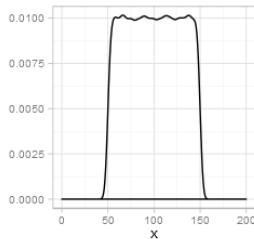
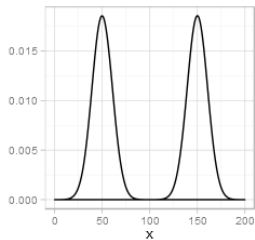
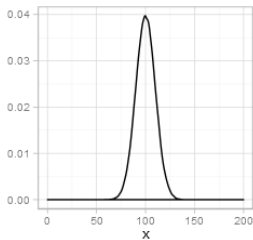
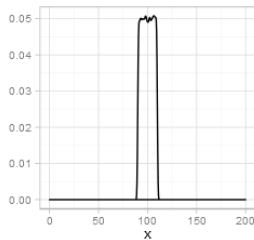
BenchmarkDotNet v0.9.9

Центральная предельная теорема спешит на помощь!



BenchmarkDotNet v0.9.9

Но есть и сложные случаи



Часть 1.5

Различные сложности

```
var sw = Stopwatch.StartNew();  
int x = 0;  
for (int i = 0; i < N; i++) // overhead  
    x++; // target operation  
sw.Stop();
```

Плохой бенчмарк

```
var sw1 = Stopwatch.StartNew();  
Foo();  
sw1.Stop();  
var sw2 = Stopwatch.StartNew();  
Bar();  
sw2.Stop();
```

Плохой бенчмарк

```
var sw1 = Stopwatch.StartNew();  
Foo();  
sw1.Stop();  
var sw2 = Stopwatch.StartNew();  
Bar();  
sw2.Stop();
```

Вспомним про:

- Interface method dispatch
- Garbage collector and autotuning
- Conditional jitting

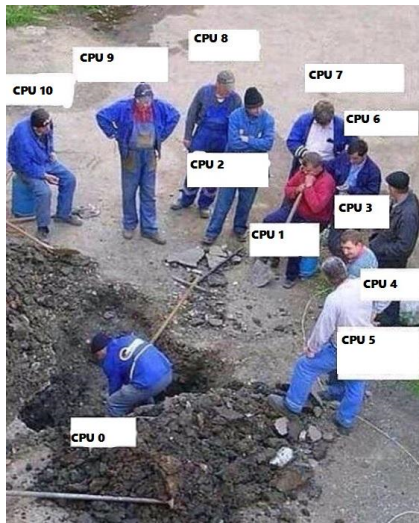
- Dead code elimination
- Inlining
- Constant folding
- Instruction Level Parallelism
- Branch prediction
- ...

Знай Latency операций!

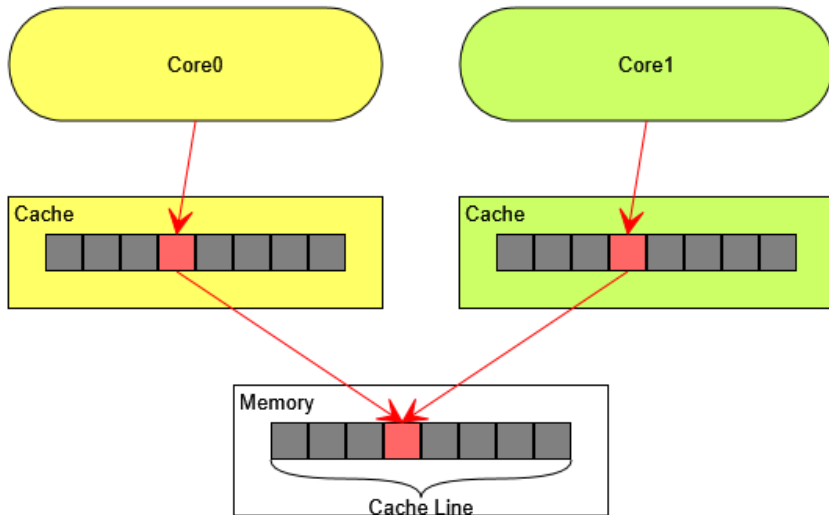
Event	Latency
1 CPU cycle	0.3 ns
Level 1 cache access	0.9 ns
Level 2 cache access	2.8 ns
Level 3 cache access	12.9 ns
Main memory access	120 ns
Solid-state disk I/O	50-150 μ s
Rotational disk I/O	1-10 ms
Hardware virtualization reboot	40 sec
Physical system reboot	5 min

© Systems Performance: Enterprise and the Cloud

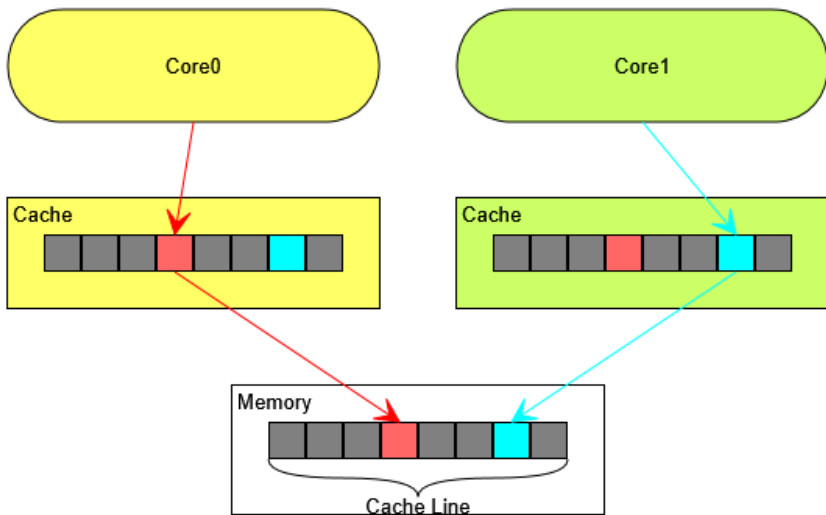
Processor affinity



True sharing



False sharing



False sharing в действии

```
private static int[] x = new int[1024];

private void Inc(int p)
{
    for (int i = 0; i < 10000001; i++)
        x[p]++;
}

private void Run(int step)
{
    var sw = Stopwatch.StartNew();
    Task.WaitAll(
        Task.Factory.StartNew(() => Inc(0 * step)),
        Task.Factory.StartNew(() => Inc(1 * step)),
        Task.Factory.StartNew(() => Inc(2 * step)),
        Task.Factory.StartNew(() => Inc(3 * step)));
    Console.WriteLine(sw.ElapsedMilliseconds);
}
```

False sharing в действии

```
private static int[] x = new int[1024];

private void Inc(int p)
{
    for (int i = 0; i < 10000001; i++)
        x[p]++;
}

private void Run(int step)
{
    var sw = Stopwatch.StartNew();
    Task.WaitAll(
        Task.Factory.StartNew(() => Inc(0 * step)),
        Task.Factory.StartNew(() => Inc(1 * step)),
        Task.Factory.StartNew(() => Inc(2 * step)),
        Task.Factory.StartNew(() => Inc(3 * step)));
    Console.WriteLine(sw.ElapsedMilliseconds);
}
```

Run(1)	Run(256)
≈400ms	≈150ms

Бенчмаркинг — это сложно

Anon et al., “A Measure of Transaction Processing Power”

There are lies, damn lies and then there are performance measures.

Часть 2

Практика

Часть 2.1

Сложности нанобенчмаркинга

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](#)

Какой из методов работает быстрее?

```
[MethodImpl(MethodImplOptions.NoInlining)]  
public void Empty0() {}
```

```
[MethodImpl(MethodImplOptions.NoInlining)]  
public void Empty1() {}
```

```
[MethodImpl(MethodImplOptions.NoInlining)]  
public void Empty2() {}
```

```
[MethodImpl(MethodImplOptions.NoInlining)]  
public void Empty3() {}
```

Давайте забенчмаркаем!

```
private void MeasureX() // X = 0, 1, 2, 3
{
    for (int i = 0; i < Rep; i++)
    {
        var sw = Stopwatch.StartNew();
        for (int j = 0; j < N; j++)
            EmptyX(); // X = 0, 1, 2, 3
        sw.Stop();
        Write(sw.ElapsedMilliseconds + " ");
    }
}
```

Давайте забенчмаркаем!

```
private void MeasureX() // X = 0, 1, 2, 3
{
    for (int i = 0; i < Rep; i++)
    {
        var sw = Stopwatch.StartNew();
        for (int j = 0; j < N; j++)
            EmptyX(); // X = 0, 1, 2, 3
        sw.Stop();
        Write(sw.ElapsedMilliseconds + " ");
    }
}
```

Empty0:	242	253	245	253	242	244	245	255	245	245	// Slow
Empty1:	241	240	237	244	242	241	238	245	239	239	// Slow
Empty2:	224	228	229	224	223	224	227	222	228	222	// Fast
Empty3:	229	222	226	222	224	226	227	229	225	230	// Fast

Рекомендуемая литература: *Agner Fog*,
“*The microarchitecture of Intel, AMD and VIA CPUs.*
An optimization guide for assembly programmers and compiler makers.”

Рекомендуемая литература: *Agner Fog*,
“*The microarchitecture of Intel, AMD and VIA CPUs.*
An optimization guide for assembly programmers and compiler makers.”

3.8 Branch prediction in Intel Haswell, Broadwell and Skylake

Pattern recognition for indirect jumps and calls.

Indirect jumps and indirect calls are predicted well.

Рекомендуемая литература: *Agner Fog*,
“*The microarchitecture of Intel, AMD and VIA CPUs.*
An optimization guide for assembly programmers and compiler makers.”

3.8 Branch prediction in Intel Haswell, Broadwell and Skylake

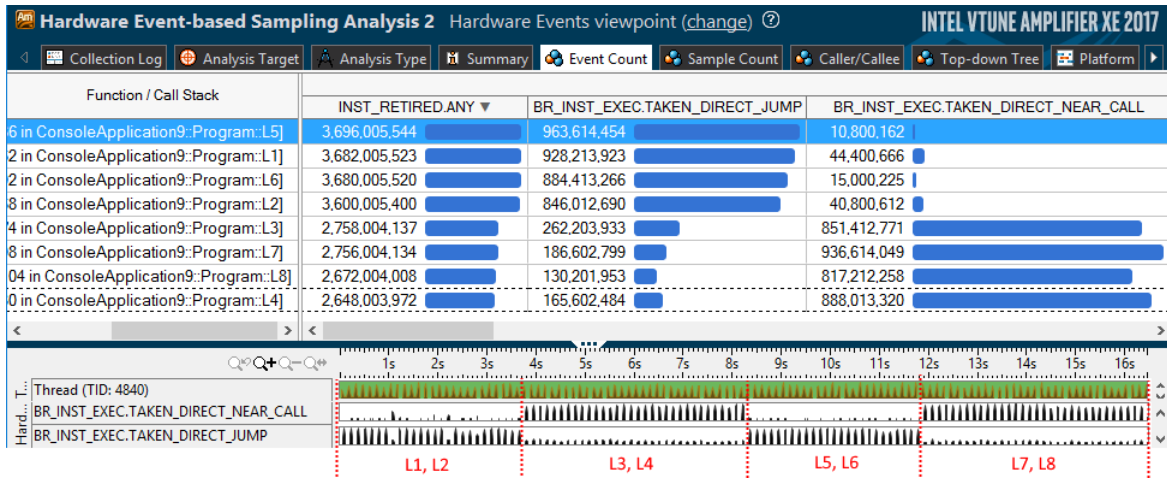
Pattern recognition for indirect jumps and calls.

Indirect jumps and indirect calls are predicted well.

...

These observations may indicate that there are two branch prediction methods: a fast method tied to the μop cache and the instruction cache, and a slower method using a branch target buffer.

Воспользуемся правильным инструментом



Раскрутка цикла спешит на помощь!

```
var sw = Stopwatch.StartNew();  
for (int j = 0; j < N; j++)  
{  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
    Empty0();  
}  
sw.Stop();
```

Часть 2.2

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 2.3

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 2.4

Interface method dispatch


```
private interface IInc {  
    double Inc(double x);  
}  
  
private class Foo : IInc {  
    double Inc(double x) => x + 1;  
}  
  
private class Bar : IInc {  
    double Inc(double x) => x + 1;  
}  
  
private double Run(IInc inc) {  
    double sum = 0;  
    for (int i = 0; i < 1001; i++)  
        sum += inc.Inc(0);  
    return sum;  
}
```

```
// Which method is faster?
```

```
[Benchmark]  
public double FooFoo() {  
    var foo1 = new Foo();  
    var foo2 = new Foo();  
    return Run(foo1) + Run(foo2);  
}
```

```
[Benchmark]  
public double FooBar() {  
    var foo = new Foo();  
    var bar = new Bar();  
    return Run(foo) + Run(bar);  
}
```

```
private interface IInc {
    double Inc(double x);
}
private class Foo : IInc {
    double Inc(double x) => x + 1;
}
private class Bar : IInc {
    double Inc(double x) => x + 1;
}
private double Run(IInc inc) {
    double sum = 0;
    for (int i = 0; i < 1001; i++)
        sum += inc.Inc(0);
    return sum;
}
```

// Which method is faster?

```
[Benchmark]
public double FooFoo() {
    var foo1 = new Foo();
    var foo2 = new Foo();
    return Run(foo1) + Run(foo2);
}
```


```
[Benchmark]
public double FooBar() {
    var foo = new Foo();
    var bar = new Bar();
    return Run(foo) + Run(bar);
}
```

	FooFoo	FooBar
LegacyJIT-x64	$\approx 5.4\mu s$	$\approx 7.1\mu s$

Часть 2.5

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
```

Ещё одна загадка



Alexandre Mutel @xoofx · 9h

Checking JIT inline of a custom List<T> this[index] accessor, the one passing string to throw helper method is not inlining oO
Why but why?



```
public T this[int index]
{
    // not inlining! oO
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    get
    {
        if ((uint)index ≥ (uint)count) ThrowArgumentOutOfRangeException(nameof(index));
        return Items[index];
    }
}

public T this[int index]
{
    // inlining ok!
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    get
    {
        if ((uint)index ≥ (uint)count) ThrowArgumentOutOfRangeException();
        return Items[index];
    }
}
```



Alexandre Mutel @xoofx · 9h

Checking JIT inline of a custom List<T> this[index] accessor, the one passing string to throw helper method is not inlining oO Why but why?



Adam Sitnik @SitnikAdam · 9h

@xoofx Have you tried InliningDiagnoser? you should get the answer
#BenchmarkDotNet



Alexandre Mutel

@xoofx

@SitnikAdam interesting it says "Fail Reason:
Cross assembly inline failed due to
NoStringInterning"

Часть 2.6

SIMD

Поговорим про SIMD

```
private struct MyVector {
    public float X, Y, Z, W;
    public MyVector(float x, float y, float z, float w) {
        X = x; Y = y; Z = z; W = w;
    }
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public static MyVector operator *(MyVector left, MyVector right) {
        return new MyVector(left.X * right.X, left.Y * right.Y,
                             left.Z * right.Z, left.W * right.W);
    }
}

private Vector4    vector1,    vector2,    vector3;
private MyVector myVector1, myVector2, myVector3;
[Benchmark] public void MyMul() => myVector3 = myVector1 * myVector2;
[Benchmark] public void BclMul() => vector3 = vector1 * vector2;
```

Поговорим про SIMD

```
private struct MyVector {  
    public float X, Y, Z, W;  
    public MyVector(float x, float y, float z, float w) {  
        X = x; Y = y; Z = z; W = w;  
    }  
    [MethodImpl(MethodImplOptions.AggressiveInlining)]  
    public static MyVector operator *(MyVector left, MyVector right) {  
        return new MyVector(left.X * right.X, left.Y * right.Y,  
                             left.Z * right.Z, left.W * right.W);  
    }  
}  
  
private Vector4    vector1,    vector2,    vector3;  
private MyVector myVector1, myVector2, myVector3;  
[Benchmark] public void MyMul() => myVector3 = myVector1 * myVector2;  
[Benchmark] public void BclMul() => vector3 = vector1 * vector2;
```

	LegacyJIT-x64	RyuJIT-x64
MyMul	12.9ns	2.5ns
BclMul	12.9ns	0.2ns

	LegacyJIT-x64	RyuJIT-x64
MyMul	≈12.9ns	≈2.5ns
BclMul	≈12.9ns	≈0.2ns

```
; LegacyJIT-x64
; MyMul, BclMul: Naïve SSE
; ...
movss    xmm3,dword ptr [rsp+40h]
mulss    xmm3,dword ptr [rsp+30h]
movss    xmm2,dword ptr [rsp+44h]
mulss    xmm2,dword ptr [rsp+34h]
movss    xmm1,dword ptr [rsp+48h]
mulss    xmm1,dword ptr [rsp+38h]
movss    xmm0,dword ptr [rsp+4Ch]
mulss    xmm0,dword ptr [rsp+3Ch]
xor      eax,eax
mov      qword ptr [rsp],rax
mov      qword ptr [rsp+8],rax
lea      rax,[rsp]
movss    dword ptr [rax],xmm3
movss    dword ptr [rax+4],xmm2
; ...
```

```
; RyuJIT-x64
; MyMul: Naïve AVX
; ...
vmulss    xmm0,xmm0,xmm4
vmulss    xmm1,xmm1,xmm5
vmulss    xmm2,xmm2,xmm6
vmulss    xmm3,xmm3,xmm7
; ...

; BclMul: Smart AVX intrinsic
vmovupd    xmm0,xmmword ptr [rcx+8]
vmovupd    xmm1,xmmword ptr [rcx+18h]
vmulps     xmm0,xmm0,xmm1
vmovupd    xmmword ptr [rcx+28h],xmm0
```

Часть 2.7

Constant folding

Интересный issue на dotnet/coreclr

 dotnet / coreclr

[Watch](#) 994 [Unstar](#) 7,001 [Fork](#) 1,697

[Code](#) [Issues 956](#) [Pull requests 39](#) [Projects 1](#) [Wiki](#) [Pulse](#) [Graphs](#)

JIT optimization - Perform additional constant propagation for expressions. #987

[New issue](#)

[Open](#) **briansull** opened this issue on May 12, 2015 · 0 comments



briansull commented on May 12, 2015

coreclr contributor +

Thee following two cases should generate the same code:

Projects
None yet

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
```


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 2.8

Instruction level parallelism

Неожиданные perf-эффекты

dotnet / coreclr

Watch 994 Unstar 7,001 Fork 1,697

<> Code ① Issues 956 Pull requests 40 Projects 1 Wiki Pulse Graphs

.Net 4.6 RC x64 is twice as slow as x86 (release version)

#993

Closed BijanVan opened this issue on May 13, 2015 · 10 comments

 BijanVan commented on May 13, 2015

Net 4.6 RC x64 is twice as slow as x86 (release version):

Consider this piece of code:

```
class SpectralNorm
{
    public static void Main(String[] args)
    {
        int n = 5500;
        if (args.Length > 0) n = Int32.Parse(args[0]);

        var spec = new SpectralNorm();
        var watch = Stopwatch.StartNew();
        var res = spec.Approximate(n);

        Console.WriteLine("{0:f9} -- {1}", res, watch.Elapsed.TotalMilliseconds);
    }

    double Approximate(int n)
    {
        // create unit vector
```

Projects
None yet

Labels
CodeGen
optimization

Milestone
No milestone

Assignees
 sivarv

9 participants


¹Справедливо для RyuJIT RC, ныне пофикшено

```
private double[] x = new double[11];

[Benchmark]
public double Calc()
{
    double sum = 0.0;
    for (int i = 1; i < x.Length; i++)
        sum += 1.0 / (i * i) * x[i];
    return sum;
}
```

```
private double[] x = new double[11];

[Benchmark]
public double Calc()
{
    double sum = 0.0;
    for (int i = 1; i < x.Length; i++)
        sum += 1.0 / (i * i) * x[i];
    return sum;
}
```

	LegacyJIT-x64	RyuJIT-x64 ¹
Calc	1 попугай	2 попугая

¹RyuJIT RC

```
; LegacyJIT-x64
; eax = i
mov     eax,r8d
; eax = i*i
imul    eax,r8d
; xmm0=i*i
cvtsi2sd    xmm0,eax
; xmm1=1
movsd    xmm1,
        mmword ptr [7FF9141145E0h]
; xmm1=1/(i*i)
divsd    xmm1,xmm0

; xmm1=1/(i*i)*x[i]
mulsd    xmm1,
        mmword ptr [rdx+r9+10h]
; xmm1 = sum + 1/(i*i)*x[i]
addsd    xmm1,xmm2
; sum = sum + 1/(i*i)*x[i]
movapd   xmm2,xmm1
```

```
; RyuJIT-x64
; r8d = i
mov     r8d,eax
; r8d = i*i
imul    r8d,eax
; xmm1=i*i
vcvtsi2sd    xmm1,xmm1,r8d
; xmm2=1
vmovsd    xmm2,
        qword ptr [7FF9140E4398h]
; xmm2=1/(i*i)
vdivsd    xmm2,xmm2,xmm1
mov     r8,rdx
movsxd   r9,eax
; xmm1 = 1/(i*i)
vmovaps   xmm1,xmm2
; xmm1 = 1/(i*i)*x[i]
vmulsd    xmm1,xmm1,
        mmword ptr [r8+r9*8+10h]
; sum += 1/(i*i)*x[i]
vaddsd    xmm0,xmm0,xmm1
```

Часть 3

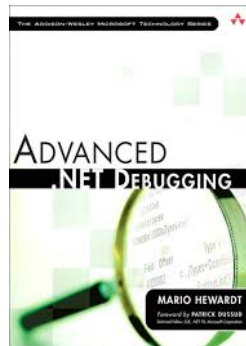
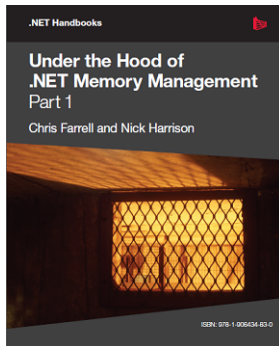
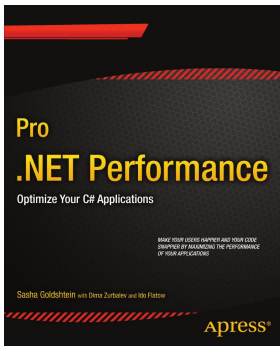
Заключение

- 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