Поговорим про память

Андрей Акиньшин, JetBrains

DotNext 2017 SPb

Часть 1
О чём будем разговаривать?

Хотим, чтобы программы работали
очень быстро и кушали мало памяти!

Знай свой bottleneck

bottleneck

['bɔtlnek]

ABBYY Lingvo Live

1. сущ.
   1) горлышко бутылки
   2) узкий проход
   3) пробка (в уличном движении)
   4) узкое место
   5) воен. дефиле
   6) муз.

      слайд-игра (техника игры на гитаре, при которой к струнам прижимают металлическую пластинку или приспособление в форме бутылочного горлышка), ботлнек, слайд (приспособление для слайд-игры на гитаре)
2. гл.
   1) создавать затор, пробку
   2) протолкнуть (что-л.) через затор

Знай свой bottleneck

- CPU
- Database
- I/O
- Network
- Concurrency
- ...
- **Main memory (вот про неё и будем разговаривать)**

История одной оптимизации

Делаем замер времени (один) до и после:

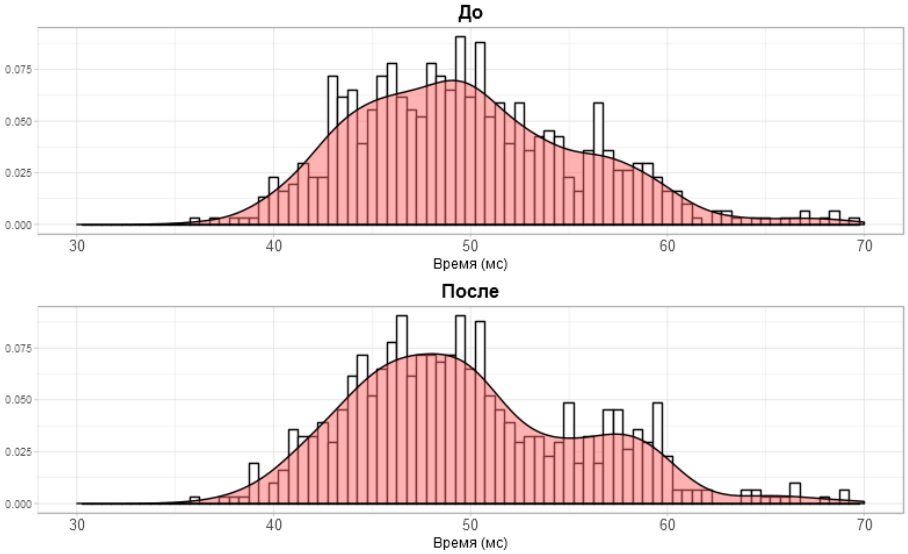
До	50ms
После	45ms

Делаем замер времени (один) до и после:

До	50ms
После	45ms

Вопрос: а стало ли лучше?

Посмотрим на распределения



Краткое содержание доклада

Сегодня мы узнаем:

- Из чего складывается производительность памяти?
- 10 способов написать *некорректный* memory benchmark

Краткое содержание доклада

Сегодня мы узнаем:

- Из чего складывается производительность памяти?
- 10 способов написать *некорректный* memory benchmark

Обсуждаемые темы:

- CPU Cache (cache line size, associativity, . . .)
- Cache Bank Conflicts
- Alignment
- Struct Promotion
- Prefetching
- Store forwarding; 4K aliasing
- Трудности работы с GC и pinned objects
- Large Object Heap; Full Framework vs Mono

Часть 2
Введение

Отказ от ответственности

Нужно понимать

- Все примеры академические, нужны для иллюстрации идей
- Погрешности малы, для целей доклада ими можно пренебречь
- На вашем железе цифры могут быть другие, это нормально

Подопытное железо

- `Ivy Bridge  Core i7-3632QM CPU 2.20GHz`
- `Haswell  Core i7-4702MQ CPU 2.20GHz`
- `Skylake  Core i7-6700HQ CPU 2.60GHz`

Информация о бенчмарках

- BenchmarkDotNet v0.10.6 (supported by the .NET Foundation)
- Gist с исходниками: https://git.io/v9dV8

Давайте позамеряем время!

```
// Инициализируем собственную хешмапу
var dict = new MyAwesomeDictionary<int, int>();
dict.Put(0, 0);
```

Давайте позамеряем время!

```
// Инициализируем собственную хешмапу
var dict = new MyAwesomeDictionary<int, int>();
dict.Put(0, 0);
```

```
// А теперь немного побенчмаркаем:
var sw = Stopwatch.StartNew();
for (int i = 0; i < 1000000; i++)
  dict.Get(0);
sw.Stop();
WriteLine(sw.Elapsed);
```

Давайте позамеряем время!

```
// Инициализируем собственную хешмапу
var dict = new MyAwesomeDictionary<int, int>();
dict.Put(0, 0);
```

```
// А теперь немного побенчмаркаем:
var sw = Stopwatch.StartNew();
for (int i = 0; i < 1000000; i++)
  dict.Get(0);
sw.Stop();
WriteLine(sw.Elapsed);
```

Есть ли проблемы с этим бенчмарком?

Нужно правильно дизайнить memory-бенчмарки

Нужно правильно дизайнить memory-бенчмарки

Пробуйте разные паттерны доступа к памяти

- Последовательный доступ
- Случайный доступ
- Пользовательский сценарий
- “Хитрые” случаи

Нужно правильно дизайнить memory-бенчмарки

Пробуйте разные паттерны доступа к памяти

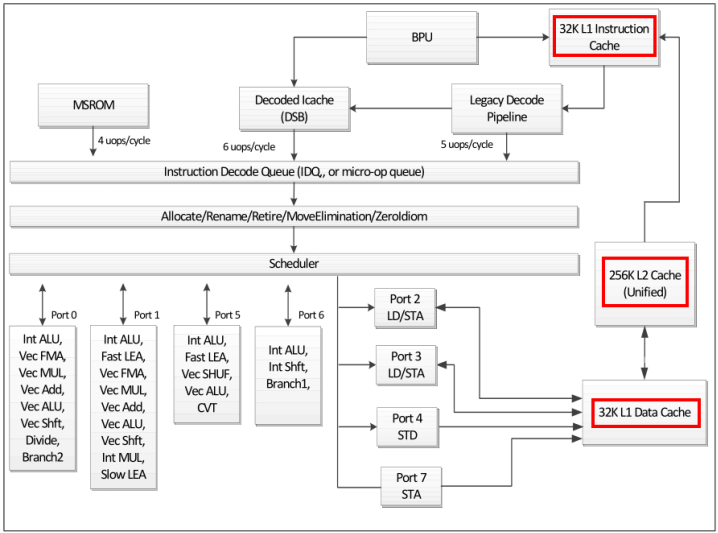
- Последовательный доступ
- Случайный доступ
- Пользовательский сценарий
- “Хитрые” случаи

Пробуйте разные размеры рабочей памяти

- Очень маленький
- Очень большой
- Значения, близкие к размеру L1, L2, L3

Часть 3
Поговорим про CPU Cache

Доступ к памяти — это вам не константа!



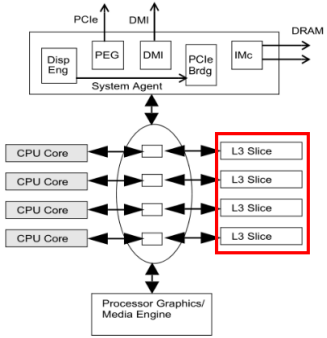


Figure 2-1. CPU Core Pipeline Functionality of the Skylake Microarchitecture

(Бенчмарк) CPU Cache

```
                /* Сколько стоит сделать N инкрементов? */
private const int N = 16 * 1024 * 1024;
private byte[] data;

[Params(1, 2, 4, 8, 16, 32, 64, 128, 256, 512,
        1024, 2048, 4096, 6144, 8192, 16384, 32768)]
public int SizeKb;

[Setup] public void Setup() => data = new byte[SizeKb * 1024];

[Benchmark]
public void Calc() {
    var mask = data.Length - 1;
    for (int i = 0; i < N; i++)     // Бегаем по массиву с шагом в 64B
        data[(i * 64) & mask]++;    // И тыкаемся в элементы
}
```

Windows 10, .NET 4.6.2, LegacyJIT-x86

(Результаты) CPU Cache

Сколько стоит сделать N инкрементов?

	Cache Level	Ivy Bridge	Skylake

(Результаты) CPU Cache

Сколько стоит сделать N инкрементов?

	Cache Level	Ivy Bridge	Skylake
4 KB	L1	≈20ms	≈18ms
8 KB		≈20ms	≈18ms
16 KB		≈20ms	≈18ms
32 KB		≈20ms	≈18ms

(Результаты) CPU Cache

Сколько стоит сделать N инкрементов?

	Cache Level	Ivy Bridge	Skylake
4 KB	L1	≈20ms	≈18ms
8 KB		≈20ms	≈18ms
16 KB		≈20ms	≈18ms
32 KB		≈20ms	≈18ms
64 KB	L2	≈36ms	≈22ms
128 KB		≈36ms	≈22ms
256 KB		≈36ms	≈22ms

(Результаты) CPU Cache

Сколько стоит сделать N инкрементов?

	Cache Level	Ivy Bridge	Skylake
4 KB	L1	≈20ms	≈18ms
8 KB		≈20ms	≈18ms
16 KB		≈20ms	≈18ms
32 KB		≈20ms	≈18ms
64 KB	L2	≈36ms	≈22ms
128 KB		≈36ms	≈22ms
256 KB		≈36ms	≈22ms
512 KB	L3	≈50ms	≈36ms
1 MB		≈50ms	≈36ms
2 MB		≈50ms	≈36ms
4 MB		≈54ms	≈40ms
6 MB		≈54ms	≈40ms

(Результаты) CPU Cache

Сколько стоит сделать N инкрементов?

	Cache Level	Ivy Bridge	Skylake
4 KB	L1	≈20ms	≈18ms
8 KB		≈20ms	≈18ms
16 KB		≈20ms	≈18ms
32 KB		≈20ms	≈18ms
64 KB	L2	≈36ms	≈22ms
128 KB		≈36ms	≈22ms
256 KB		≈36ms	≈22ms
512 KB	L3	≈50ms	≈36ms
1 MB		≈50ms	≈36ms
2 MB		≈50ms	≈36ms
4 MB		≈54ms	≈40ms
6 MB		≈54ms	≈40ms
8 MB	RAM	≈80ms	≈70ms
16 MB		≈102ms	≈80ms
32 MB		≈108ms	≈83ms

Часть 4
Поговорим про CPU Cache Levels

(Схематическое устройство) CPU Cache Levels

CPU0	CPU1	CPU2	CPU3
L1	L1	L1	L1
L2	L2	L2	L2
L3			

(Бенчмарк) CPU Cache Levels

```
private const int N = 16 * 1024 * 1024;
private byte[] data;

[Params(2048, 4096, 6144)] // Нагрузка на L3
public int SizeKb;

[Setup] public void Setup() => data = new byte[SizeKb * 1024];

[Benchmark]
public void Calc() {
    var mask = data.Length - 1;
    for (int i = 0; i < N; i++)
        data[(i * 64) & mask]++;
}
```

Windows 10, .NET 4.6.2, LegacyJIT-x86, **Skylake**

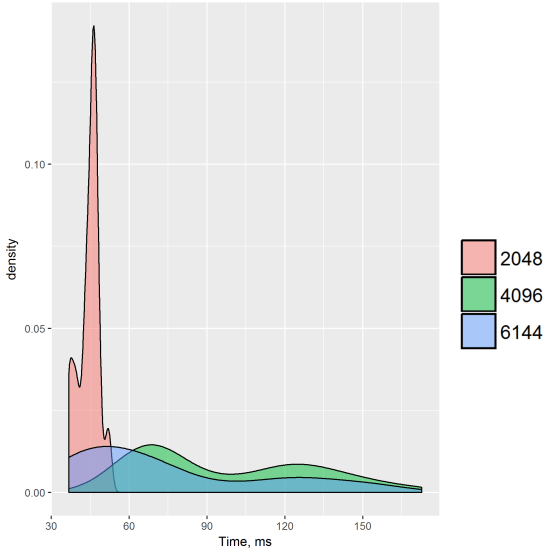
(Результаты) CPU Cache Levels

	Условия	Mean	Min	Max
2 MB	Стерильные	**35.38ms**	34.92ms	36.08ms
4 MB		**39.49ms**	39.01ms	40.06ms
6 MB		**40.44ms**	39.87ms	40.92ms

(Результаты) CPU Cache Levels

	Условия	Mean	Min	Max
2 MB	Стерильные	**35.38ms**	34.92ms	36.08ms
4 MB		**39.49ms**	39.01ms	40.06ms
6 MB		**40.44ms**	39.87ms	40.92ms
2 MB	Firefox, Rider, IDEA, TEXstudio, Dropbox, GoogleDrive, ...	**44.50ms**	36.75ms	52.34ms
4 MB		**97.64ms**	63.14ms	172.74ms
6 MB		**75.02ms**	42.07ms	154.55ms

(Распределение) CPU Cache Levels



Часть 5
Поговорим про CPU Cache Associativity

(Бенчмарк) CPU Cache Associativity

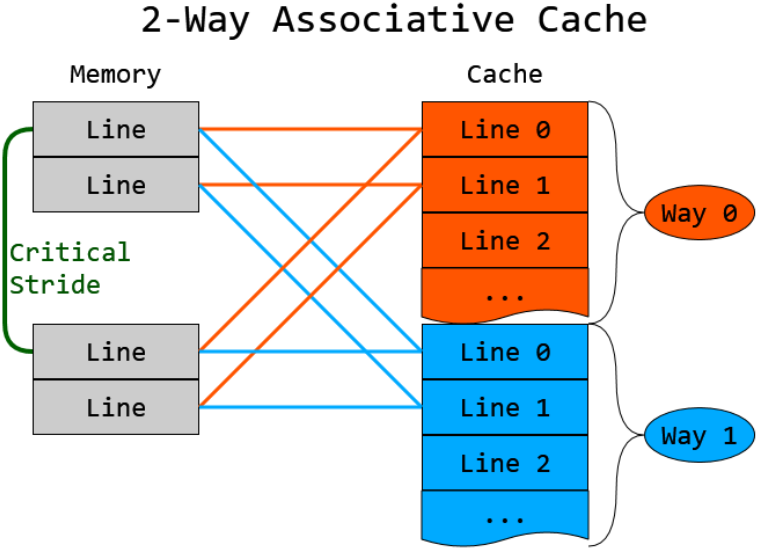
```
private int[,] a;
[Params(511, 512, 513)] public int N;
[Setup] public void Setup() => a = new int[N, N];

// Ищем максимальный элемент в колонке
[Benchmark] public int Max() {
    int max = 0;
    for (int i = 0; i < N; i++)
        max = Math.Max(max, a[i, 0]);
    return max;
}
```

Windows 10, .NET 4.6.2, LegacyJIT-x86, Skylake

(Результаты) CPU Cache Associativity

N	Max
511	≈844ns
512	≈1330ns
513	≈844ns

(Теория) CPU Cache Associativity



(Теория) CPU Cache Associativity

```
var criticalStride = cacheSize / associativity;
```

(Теория) CPU Cache Associativity

```
var criticalStride = cacheSize / associativity;
```

Level	Size	Associativity	Critical Stide
L1	32KB	8-way	4KB
L2	256KB	4-way	64KB
L2	256KB	8-way	32KB
L3	6MB	12-way	512KB

Минутка занимательного про Skylake

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.

Часть 6
Поговорим про Cache Bank Conflicts

(Бенчмарк) Cache Bank Conflicts

```
int[] data = new int[2 * 1024 * 1024];
[Params(15, 16, 17)] public int Delta;
// Есть ли у нас пары элементов на расстоянии Delta,
// в которых первое число меньше второго?
[Benchmark] public unsafe bool Calc() {
    fixed (int* dataPtr = data) {
        int* ptr = dataPtr;
        int d = Delta;
        bool res = false;
        for (int i = 0; i < 1024 * 1024; i++) {
            res |= (ptr[0] < ptr[d]);
            ptr++;
        }
        return res;
    }
}
```

Windows 10, .NET 4.6.2, LegacyJIT-x86

(Результаты) Cache Bank Conflicts

Delta	15	16	17

Skylake: выпущен 2015-08-05, микроархитектура Skylake
Ivy Bridge: выпущен 2012-04-29, микроархитектура Sandy Bridge

(Результаты) Cache Bank Conflicts

Delta	15	16	17
Skylake	≈1.05ms	≈1.05ms	≈1.05ms

Skylake: выпущен 2015-08-05, микроархитектура Skylake
Ivy Bridge: выпущен 2012-04-29, микроархитектура Sandy Bridge

(Результаты) Cache Bank Conflicts

Delta	15	16	17
Skylake	≈1.05ms	≈1.05ms	≈1.05ms
Ivy Bridge	≈1.07ms	≈1.30ms	≈1.07ms

Skylake: выпущен 2015-08-05, микроархитектура Skylake

Ivy Bridge: выпущен 2012-04-29, микроархитектура Sandy Bridge

(Мануал) Cache Bank Conflicts

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

3.6.1.3. Handling L1D Cache Bank Conflict

In Intel microarchitecture code name Sandy Bridge, the internal organization of the L1D cache may manifest a situation when two load micro-ops whose addresses have a bank conflict. When a bank conflict is present between two load operations, the more recent one will be delayed until the conflict isresolved. A bank conflict happens when two simultaneous load operations have the same bit 2–5 of their linear address but they are not from the same set in the cache (bits 6–12).

(Мануал) Cache Bank Conflicts

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

3.6.1.3. Handling L1D Cache Bank Conflict

In Intel microarchitecture code name Sandy Bridge, the internal organization of the L1D cache may manifest a situation when two load micro-ops whose addresses have a bank conflict. When a bank conflict is present between two load operations, the more recent one will be delayed until the conflict isresolved. A bank conflict happens when two simultaneous load operations have the same bit 2–5 of their linear address but they are not from the same set in the cache (bits 6–12).

With the Haswell microarchitecture, the L1 DCache bank conflict issue does not apply.

Часть 7
Поговорим про Alignment

(Бенчмарк) Alignment

```
public struct Struct3 {
  public byte X1, X2, X3;
}
public struct Struct8 {
  public byte X1, X2, X3, X4, X5, X6, X7, X8;
}
private Struct3[] struct3 = new Struct3[256];
private Struct8[] struct8 = new Struct8[256];
```

(Бенчмарк) Alignment

```
[Benchmark] public int S3() {
    int res = 0;
    for (int i = 0; i < struct3.Length; i++) {
        var s = struct3[i]; res += s.X1 + s.X2;
    }
    return res;
}
[Benchmark] public int S8() {
    int res = 0;
    for (int i = 0; i < struct8.Length; i++) {
        var s = struct8[i]; res += s.X1 + s.X2;
    }
    return res;
}
```

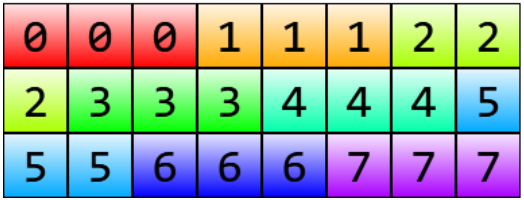
(Результаты) Alignment

	LegacyJIT-x64	Mono
S3	≈1276ns	≈1146ns
S8	≈241ns	≈2232ns

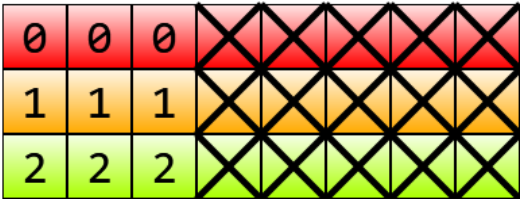
(Результаты) Alignment

	LegacyJIT-x64	Mono
S3	≈1276ns	≈1146ns
S8	≈241ns	≈2232ns
`Marshal.SizeOf(typeof(S3))`	3	8

Full .NET Framework



Mono



Часть 8
Поговорим про Struct promotion

(Бенчмарк) Struct promotion

```
// А что будет под RyuJIT-x64?
[Benchmark] public int S3() {
    int res = 0;
    for (int i = 0; i < struct3.Length; i++) {
        var s = struct3[i]; res += s.X1 + s.X2;
    }
    return res;
}
[Benchmark] public int S8() {
    int res = 0;
    for (int i = 0; i < struct8.Length; i++) {
        var s = struct8[i]; res += s.X1 + s.X2;
    }
    return res;
}
```

(Результаты) Struct promotion

	RyuJIT-x64
S3	≈280ns
S8	≈280ns

(Смотрим ASM) Struct promotion

```
; *** S3 ***
; res += s.X1 + s.X2;
add         eax, r10d
add         eax, r9d

; *** S8 ***
; res += s.X1 + s.X2;
movzx       r9d, byte ptr [rsp+20h]
add         eax, r9d
movzx       r9d, byte ptr [rsp+21h]
add         eax, r9d
```

(Текущая реализация) Struct promotion

Иногда RyuJIT[1] может аллоцировать структуру в регистрах, а не на стеке

- The struct must not be address-taken.
- It must have no overlapping fields.
- It must have only primitive fields.
- It must not be an argument or a return value that is passed in registers.
- It can't be larger than 32 bytes.
- It can't have more than 4 fields.

[1] Справедливо для v4.6.1637.0, поведение может поменяться, см. coreclr#6733

Часть 9
Поговорим про Prefetching

(Бенчмарк) Prefetching

```
// Подготавливаем данные
int[] data = new int[32 * 1024 * 1024];
int[] next = new int[32 * 1024 * 1024];
int n = 2000;
```

(Бенчмарк) Prefetching

```
// Подготавливаем данные
int[] data = new int[32 * 1024 * 1024];
int[] next = new int[32 * 1024 * 1024];
int n = 2000;
```

```
var random = new Random(42);
for (int i = 0; i < data.Length; i++)
    data[i] = random.Next();
for (int i = 0; i < next.Length; i++)
    next[i] = random.Next(data.Length);
```

(Бенчмарк) Prefetching

```
// CPU-bound method
[MethodImpl(MethodImplOptions.NoInlining)]
private static int IsNice(int x) {
    if (x % 13 == 0  | x % 17 == 0  |
        x % 19 == 0  | x % 43 == 0  |
        x % 71 == 0  | x % 101 == 0 |
        x % 233 == 0 | x % 383 == 0 |
        x % 479 == 0 | x % 541 == 0)
        return 1;
    return 0;
}
```

(Бенчмарк) Prefetching

```
[Benchmark]
public int Simple() {
  var res = 0;
  var index = 0;
  for (int i = 0; i < n; i++) {
    index = next[index];
    var x = data[index];

    res += IsNice(x);
  }
  return res;
}
```

(Бенчмарк) Prefetching

```
[Benchmark]
public int Simple() Prefetch() {
  var res = 0;
  var index = 0; next[0];   var y = data[index];
  for (int i = 0; i < n; i++) {
    index = next[index];  var x = y;
    var x = data[index];  index = next[index]; y = data[index];

    res += IsNice(x);
  }
  return res;
}
```

(Результаты) Prefetching

Simple	$\approx 161\mu s$
Prefetch	$\approx 96\mu s$

Windows 10, .NET 4.6.2, LegacyJIT-x86, Haswell

(Минутка занимательного) Prefetching

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

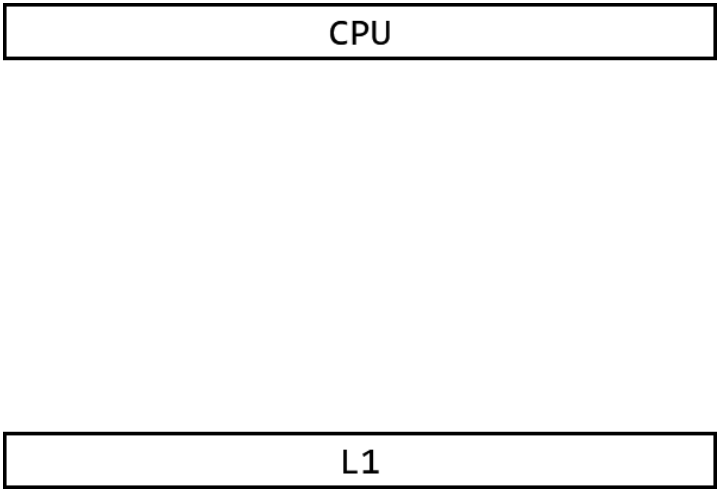
> 3.7.2. Hardware Prefetching for First-Level Data Cache
>
> The additional instructions to load data from one member in the modified sequence can trigger the DCU hardware prefetch mechanisms to prefetch data in the next cache line, enabling the work on the second member to complete sooner.

```
; Without prefetching
mov eax, [ebx+4]
```

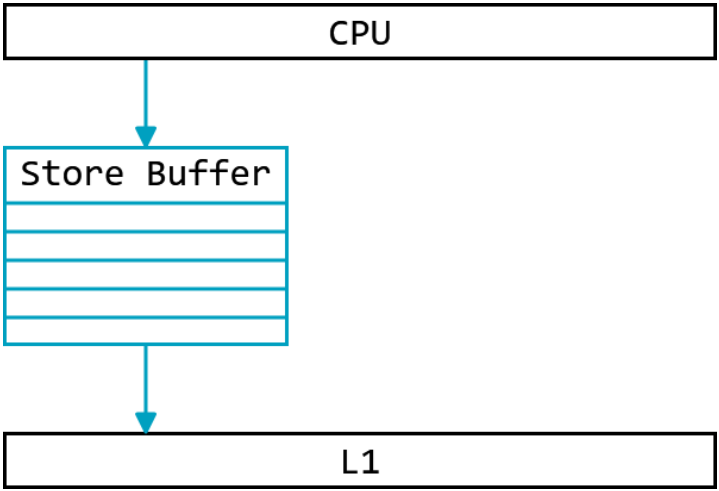
```
; With prefetching
mov eax, [ebx+4]
mov eax, [ebx+4]
mov eax, [ebx+4]
```

Часть 10
Поговорим про 4K aliasing

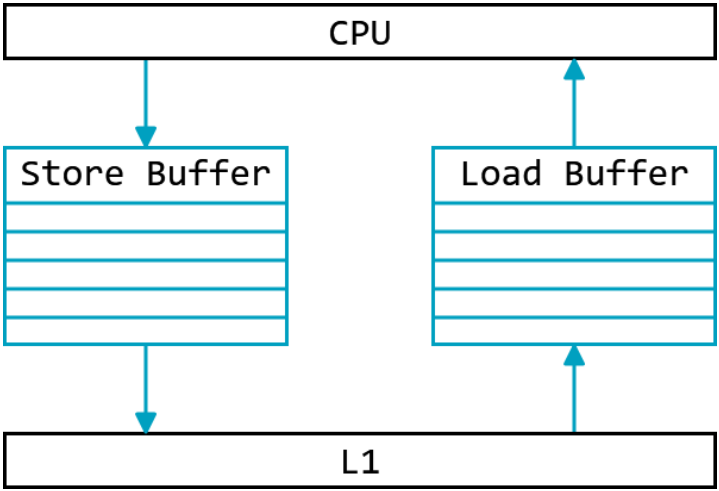
Store forwarding



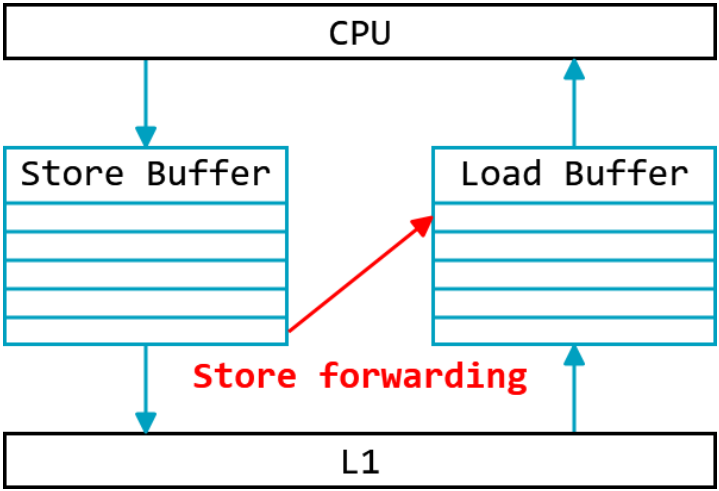
Store forwarding



Store forwarding



Store forwarding



Выравниванием доступ к памяти на 4KB

```
byte[] data = new byte[32 * 1024 * 1024];
int baseOffset; // Хотим выровнять на 4KB
```

Выравниванием доступ к памяти на 4KB

```
byte[] data = new byte[32 * 1024 * 1024];
int baseOffset; // Хотим выровнять на 4KB
```

```
GCHandle handle = GCHandle.Alloc(data,
                                 GCHandleType.Pinned);
IntPtr addrOfObject = handle.AddrOfPinnedObject();
```

Выравниванием доступ к памяти на 4KB

```
byte[] data = new byte[32 * 1024 * 1024];
int baseOffset; // Хотим выровнять на 4KB
```

```
GCHandle handle = GCHandle.Alloc(data,
                                 GCHandleType.Pinned);
IntPtr addrOfObject = handle.AddrOfPinnedObject();
```

```
long address = addrOfObject.ToInt64();
const int align = 4 * 1024; // 4KB
baseOffset = ((int) (align - (address % (align))));
```

(Бенчмарк) 4K aliasing

```
public int StrideOffset; // [Params]
[Benchmark]
public void ArrayCopy() => Array.Copy(
  sourceArray:      data,
  sourceIndex:      baseOffset,
  destinationArray: data,
  destinationIndex: baseOffset +
                    24 * 1024 + StrideOffset, //24KB
  length:           16 * 1024);               //16KB
```

(Результаты) 4K aliasing

StrideOffset	Haswell	Comment

(Результаты) 4K aliasing

StrideOffset	Haswell	Comment
-33	≈390ns	Unaligned

(Результаты) 4K aliasing

StrideOffset	Haswell	Comment
-33	≈390ns	Unaligned
-32	≈330ns	Aligned

(Результаты) 4K aliasing

StrideOffset	Haswell	Comment
-33	≈390ns	Unaligned
-32	≈330ns	Aligned
-31	≈390ns	Unaligned
-1	≈390ns	Unaligned

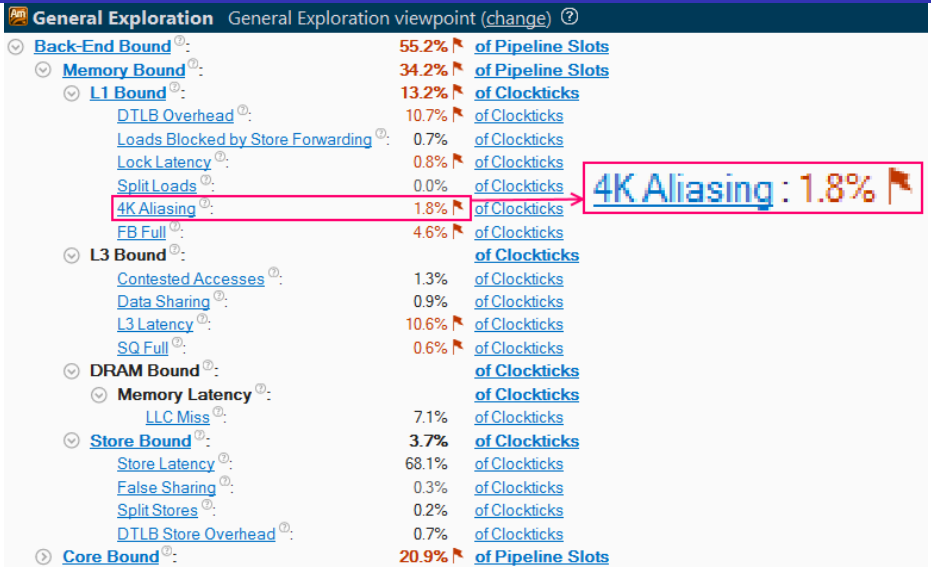
(Результаты) 4K aliasing

StrideOffset	Haswell	Comment
-33	≈390ns	Unaligned
-32	≈330ns	Aligned
-31	≈390ns	Unaligned
-1	≈390ns	Unaligned
0	≈290ns	Aligned

(Результаты) 4K aliasing

StrideOffset	Haswell	Comment
-33	≈390ns	Unaligned
-32	≈330ns	Aligned
-31	≈390ns	Unaligned
-1	≈390ns	Unaligned
0	≈290ns	Aligned
1	≈1250ns	4K aliasing
2	≈1250ns	4K aliasing
31	≈1250ns	4K aliasing

Rider: взгляд из VTune Amplifier



(Мануал) 4K aliasing

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

11.8. 4K ALIASING

To resolve 4K aliasing, try the following methods in the following order:

- Align data to 32 Bytes.
- Change offsets between input and output buffers if possible.
- Use 16-Byte memory accesses on memory which is not 32-Byte aligned.

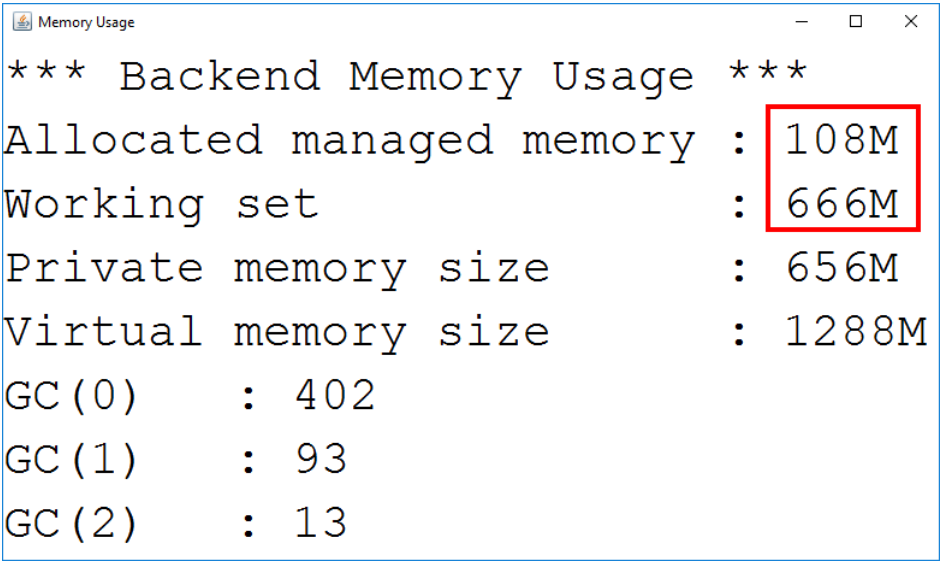
Часть 11
Поговорим про GC

Possible politically incorrect language

~~a garbage collector~~ → a sanitation worker

The term ***garbage collector*** may be considered outdated, disrespectful, or offensive. Consider changing the word or phrase.

Секретная Rider-фича



Факты об одном эксперименте

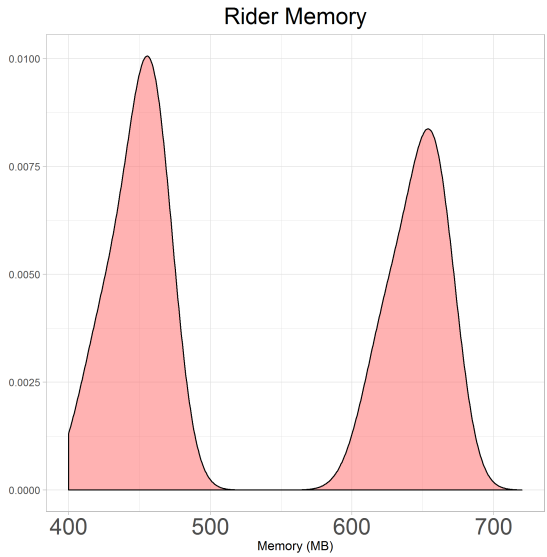
Факты об одном эксперименте

- Среднестатистический working set после загрузки: 566MB.

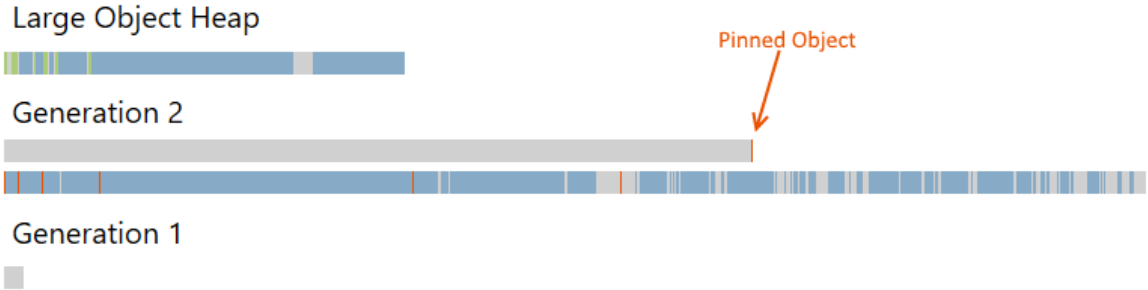
Факты об одном эксперименте

- Среднестатистический working set после загрузки: 566MB.
- Большинство людей имеют больше среднестатистического количества ног.

Посмотрим на распределение



Посмотрим на снэпшот



Посмотрим на код

```
var myListener = new HttpListener();
// ...
while (myListener.IsListening)
{
  HttpListenerContext ctx = null;
  try
  {
    ctx = myListener.GetContext(); // А что внутри?
    string rstr = ProcessRequest(ctx.Request);
    byte[] buf = Encoding.UTF8.GetBytes(rstr);
    ctx.Response.ContentLength64 = buf.Length;
    ctx.Response.OutputStream.Write(buf, 0, buf.Length);
  }
  // ...
}
```

Продолжаем смотреть на код

```
public HttpListenerContext GetContext() {
    if(Logging.On)Logging.Enter(Logging.HttpListener, this, "GetContext", "");

    SyncRequestContext memoryBlob = null;
    HttpListenerContext httpContext = null;
    bool stoleBlob = false;

    try {
        CheckDisposed();
        if (m_State==State.Stopped) {
            throw new InvalidOperationException(SR.GetString(SR.net_listener_mustcall, "Start()"));
        }
        if (m_UriPrefixes.Count==0) {
            throw new InvalidOperationException(SR.GetString(SR.net_listener_mustcall, "AddPrefix()"));
        }
        uint statusCode = UnsafeNclNativeMethods.ErrorCodes.ERROR_SUCCESS;
        uint size = 4096;
        ulong requestId = 0;
        memoryBlob = new SyncRequestContext((int) size);
```

Продолжаем смотреть на код

```
internal unsafe class SyncRequestContext : RequestContextBase
{
    private GCHandle m_PinnedHandle;
    internal SyncRequestContext(int size)
    {
        BaseConstruction(Allocate(size));
    }
    private UnsafeNclNativeMethods.HttpApi.HTTP_REQUEST* Allocate(int size)
    {
        if (m_PinnedHandle.IsAllocated)
        {
            if (RequestBuffer.Length == size)
            {
                return RequestBlob;
            }
            m_PinnedHandle.Free();
        }
        SetBuffer(size);
        if (RequestBuffer == null)
        {
            return null;
        }
        m_PinnedHandle = GCHandle.Alloc(RequestBuffer, GCHandleType.Pinned);
        return (UnsafeNclNativeMethods.HttpApi.HTTP_REQUEST*) Marshal.UnsafeAddrOfPinnedArrayElement(RequestBuffer, 0);
    }
```

Из-за нескольких pinned-байт имеем лишних 200MB

```
GCHandle.Alloc(RequestBuffer, GCHandleType.Pinned);
```

Часть 12
Поговорим про LOH

Поговорим про LOH

Large Object Heap

(Больше 85000 байт)

Generation 2

Generation 1

Справедливо для Full .NET Framework, .NET Core

(Бенчмарк) LOH

```
public class ChunkedArray // Чудо-класс, который
{                         // спасает от LOH
    private readonly List<object[]> arrays;

    public ChunkedArray(int n, int chunkSize)
    {
        arrays = new List<object[]>();
        while (n > 0)
        {
            var size = Math.Min(n, chunkSize / IntPtr.Size);
            arrays.Add(new object[size]);
            n -= size;
        }
    }
}
```

(Бенчмарк) LOH

```
public class Node  // Вспомогательный класс,
{                  // который поможет нам бенчмаркать
    public ChunkedArray Array { get; }
    public Node Next { get; }
    public int X;

    public Node(ChunkedArray array, Node next)
    {
        Array = array;
        Next = next;
    }

    // Очень полезный финализатор
    ~Node() => X++;
}
```

(Бенчмарк) LOH

```
// *** Allocate ***
var node = new Node(new ChunkedArray(N, ChunkSize),
                    null);
for (int i = 0; i < 1000; i++)
  node = new Node(new ChunkedArray(N, ChunkSize),
                  (i % 50 == 0) ? null : node);
```

(Бенчмарк) LOH

```
// *** Allocate ***
var node = new Node(new ChunkedArray(N, ChunkSize),
                    null);
for (int i = 0; i < 1000; i++)
  node = new Node(new ChunkedArray(N, ChunkSize),
                  (i % 50 == 0) ? null : node);
```

```
// *** Force Collect (.NET 4.5.1+) ***
GCSettings.LargeObjectHeapCompactionMode =
    GCLargeObjectHeapCompactionMode.CompactOnce;
GC.Collect();
GC.WaitForPendingFinalizers();
```

Аллоцируем много больших объектов, собираем мусор и замеряем время:

ChunkSize	Full Framework	Mono
80000	≈3.3s	≈2.0s

Windows 10, .NET 4.6.2, RyuJIT-x64, Mono 4.6.2, Skylake

Аллоцируем много больших объектов, собираем мусор и замеряем время:

ChunkSize	Full Framework	Mono
80000	≈3.3s	≈2.0s
∞	≈1.9s	≈1.5s

Windows 10, .NET 4.6.2, RyuJIT-x64, Mono 4.6.2, Skylake

По данному бенчмарку:

По данному бенчмарку:

- Нельзя делать выводы о рантаймах

По данному бенчмарку:

- Нельзя делать выводы о рантаймах
- Нельзя делать выводы о том, что же использовать в продакшене

По данному бенчмарку:

- Нельзя делать выводы о рантаймах
- Нельзя делать выводы о том, что же использовать в продакшене
- Вообще нельзя так бенчмаркать GC, пример слишком *академический*

Почему Rider всё ещё использует чанки?

Нельзя просто так взять и начать рефакторить legacy

Нужно подумать про:

Нельзя просто так взять и начать рефакторить legacy

Нужно подумать про:

- Разные ОС и рантаймы

Нельзя просто так взять и начать рефакторить legacy

Нужно подумать про:

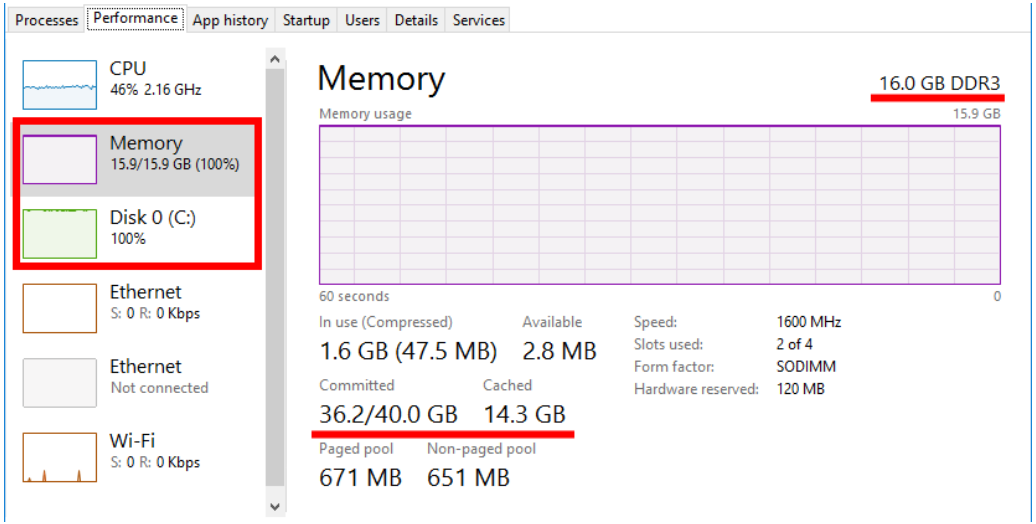
- Разные ОС и рантаймы
- Разные нагрузки и сценарии

Нельзя просто так взять и начать рефакторить legacy

Нужно подумать про:

- Разные ОС и рантаймы
- Разные нагрузки и сценарии
- Целевые метрики: latency, footprint

По уши в свопе



Часть 13
Заключение

Для memory-bound бенчмаркинга нужно много понимать

- Общая методология
  (размер рабочей памяти, паттерны доступа и всё такое)
- Устройство железа и рантаймов
- Пользовательские сценарии

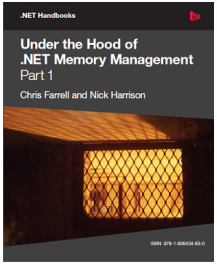
Для memory-bound бенчмаркинга нужно много понимать

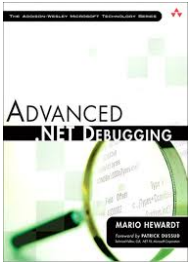
- Общая методология
  (размер рабочей памяти, паттерны доступа и всё такое)
- Устройство железа и рантаймов
- Пользовательские сценарии

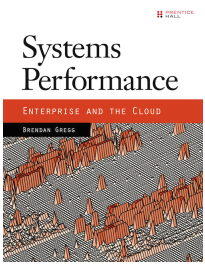
Напоминания

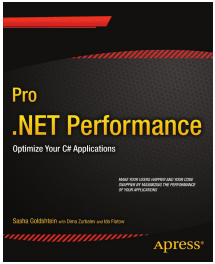
- Это был доклад о *замерах*, а не об *оптимизациях*
- В большинстве случаев всё не так уж и плохо

Методическая литература









- Intel® 64 and IA-32 Architectures Optimization Reference Manual
- An optimization guide for assembly programmers and compiler makers
- What Every Programmer Should Know About Memory

\+ 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.
```

Вопросы?

Андрей Акиньшин
http://aakinshin.net
https://github.com/AndreyAkinshin
https://twitter.com/andrey_akinshin
andrey.akinshin@gmail.com