Модель памяти .NET

Валерий
Петров

SIDENIS
passion to deliver

Обо мне

- Окончил МатМех СПбГУ
- Сейчас работаю в компании Sidenis
- Увлекаюсь concurrency и всякими «кишочками»
- Нравится узнавать новое
- Неравнодушен к качественному коду

SIDENIS
passion to deliver

WARNING!

- Доклад **НЕ** про GC, Stack, Heap
- **НЕ** будем писать свои lock-free структуры
- Доклад про разные процессорные архитектуры, компиляторы и concurrency
- **Here be dragons!**
- Верить нельзя никому (кроме спецификаций)

SIDENIS
passion to deliver

О чём доклад

- Зачем нужна модель памяти
- Что такое модель памяти
- Какая модель памяти в .NET
- Что такое volatile в .NET
- Простые примеры для разогрева
- Ошибки из реальных проектов (да, одна из них в самом .NET)

SIDENIS
passion to deliver

DotNext 2016 Piter

The C++ and CLR Memory Models

Sasha Goldshtein

CTO, Sela Group

@goldshtn

SIDENIS
passion to deliver

Shipilëv JMM

Historical (e.g. outdated) materials

(ENG) JVMLS 2014: Java Memory Model Pragmatics
Workshop collaterals.

(RUS) JEEConf 2014: Java Memory Model Pragmatics
Video: JEEConf, Direct: part 1, 364 MB, Direct: part 2, 365 MB

(RUS) CodeFest 2014: Java Memory Model Pragmatics

(RUS) JUG.Ru 2014: Java Memory Model Pragmatics
Video: Youtube, Direct: 1090 MB

(RUS) JavaOne Russia 2013: Java Memory Model
(hosted for Sergey Kuksenko)

(RUS) JavaOne Moscow 2012: Java Memory Model
(hosted for Sergey Kuksenko)
Video: Youtube

(RUS) JavaDay Kiev 2011: Java Memory Model
(hosted for Sergey Kuksenko)

SIDENIS
passion to deliver

SPB .NET Community

Модель памяти .NET

Валерий Петров
Sidenis

17 встреча SPB .NET Community
09.02.2017

SIDENIS
passion to deliver

Вопрос

Q: Компьютер выполняет программу, которую Вы написали?

SIDENIS
passion to deliver

Вопрос

Q: Компьютер выполняет программу, которую Вы написали?

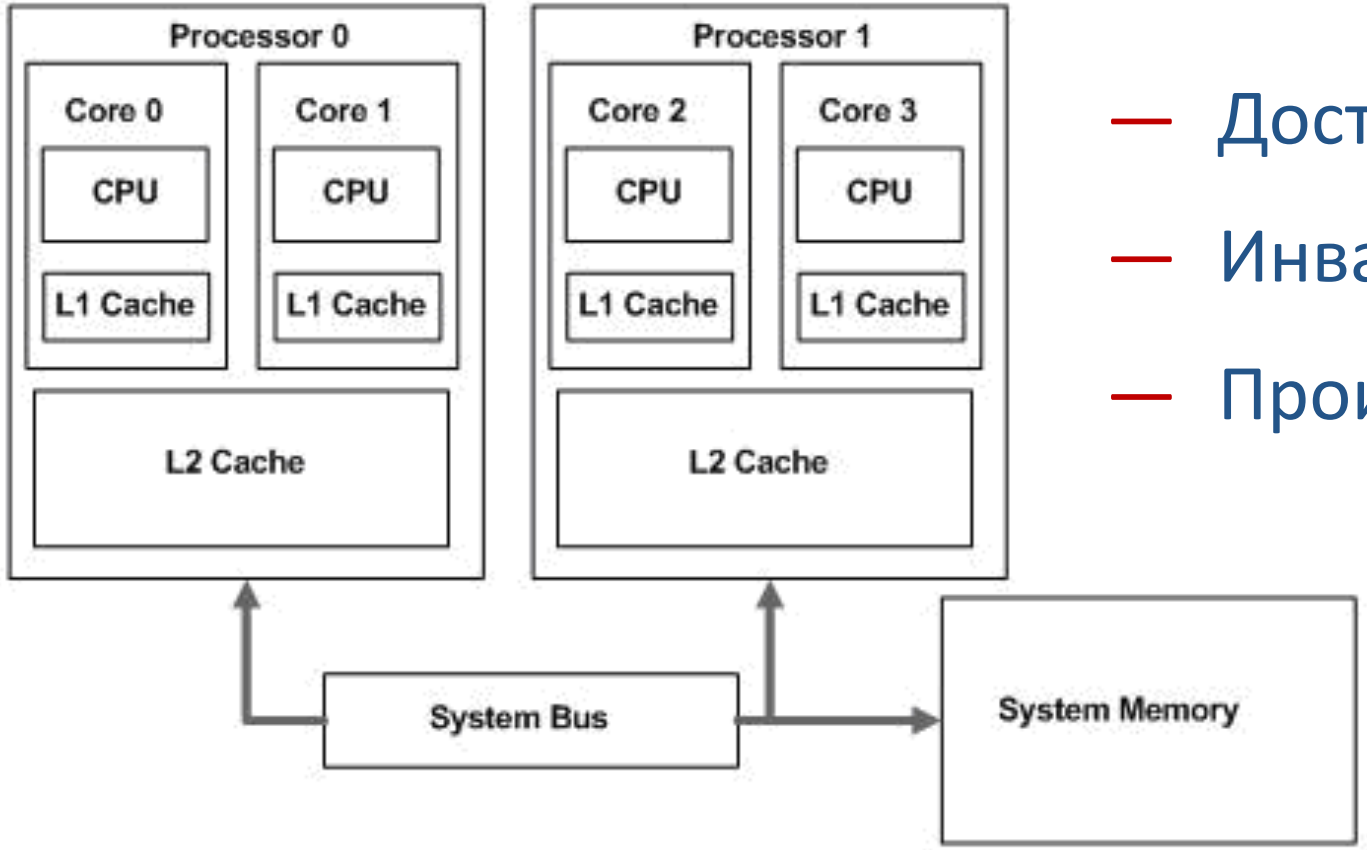
A: Нет.
Компилятор, JIT и CPU умеют оптимизировать!

SIDENIS
passion to deliver

Почему они это делают?

SIDENIS
passion to deliver

Почему они это делают?



- Доступ к памяти (долго)
- Инвалидация кэша (дорого)
- Производительность (IPC)

https://software.intel.com/sites/default/files/m/d/4/1/d/8/286501_286501.gif

SIDENIS
passion to deliver

Почему они это делают?

Раньше инструкции исполнялись последовательно

Cycle	1	2	3	4	5	6	7	8	9
$Instr_1$	Fetch	Decode	Execute	Write					
$Instr_2$					Fetch	Decode	Execute	Write	
$Instr_3$									Fetch

http://www.slideshare.net/nithilgeorge/2010-1002-intro-to-microprocessors1

SIDENIS
passion to deliver

Почему они это делают?

Затем появился конвейер

Cycle	1	2	3	4	5	6	7	8	9
$Instr_1$	Fetch	Decode	Execute	Write					
$Instr_2$		Fetch	Decode	Execute	Write				
$Instr_3$			Fetch	Decode	Execute	Write			
$Instr_4$				Fetch	Decode	Execute	Write		
$Instr_5$					Fetch	Decode	Execute	Write	
$Instr_6$						Fetch	Decode	Execute	Write

http://www.slideshare.net/nithilgeorge/2010-1002-intro-to-microprocessors1

SIDENIS
passion to deliver

Почему они это делают?

Но инструкции всё равно исполнялись в порядке следования

Cycle	1	2	3	4	5	6	7	8	9
$Instr_1$	Fetch	Decode	Execute			Write			
$Instr_2$		Fetch	Decode	Wait		Execute	Write		
$Instr_3$			Fetch	Decode	Wait		Execute	Write	
$Instr_4$				Fetch	Decode	Wait		Execute	Write
$Instr_5$					Fetch	Decode	Wait		Execute
$Instr_6$						Fetch	Decode	Wait	

http://www.slideshare.net/nithilgeorge/2010-1002-intro-to-microprocessors1

SIDENIS
passion to deliver

Почему они это делают?

Затем произошло массовое внедрение out-of-order execution

Cycle	1	2	3	4	5	6	7	8	9
$Instr_1$	Fetch	Decode	Execute			Write			
$Instr_2$		Fetch	Decode	Wait		Execute	Write		
$Instr_3$			Fetch	Decode	Execute	Write			
$Instr_4$				Fetch	Decode	Wait	Execute	Write	
$Instr_5$					Fetch	Decode	Execute	Write	
$Instr_6$						Fetch	Decode	Execute	Write

http://ww.slideshare.net/nithilgeorge/2010-1002-intro-to-microprocessors1

SIDENIS
passion to deliver

Почему они это делают?

- Компиляторщики хотят делать клёвые оптимизации
- Производители CPU хотят делать больше за те же такты

Разные архитектуры
Hardware memory model

Type	Alpha	ARMv7	PA-RISC	POWER	SPARC RMO	SPARC PSO	SPARC TSO	x86	x86 oostore	AMD64	IA-64	z/Architecture
Loads reordered after loads	Y	Y	Y	Y	Y				Y		Y	
Loads reordered after stores	Y	Y	Y	Y	Y				Y		Y	
Stores reordered after stores	Y	Y	Y	Y	Y	Y			Y		Y	
Stores reordered after loads	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Atomic reordered with loads	Y	Y		Y	Y						Y	
Atomic reordered with stores	Y	Y		Y	Y	Y					Y	
Dependent loads reordered	Y											
Incoherent instruction cache pipeline	Y	Y		Y	Y	Y	Y	Y	Y		Y	

https://en.wikipedia.org/wiki/Memory_ordering

SIDENIS
passion to deliver

Разные архитектуры

Type	ARM	x86, AMD64
Loads reordered after loads	YES	NO
Loads reordered after stores	YES	NO
Stores reordered after stores	YES	NO
Stores reordered after loads	YES	YES

SIDENIS
passion to deliver

Как быть?

- Пишем код для абстрактной машины (Software memory model)
- Абстрактную машину специфицируем (Software memory model)
- Пусть компиляторщики их стыкуют (Hardware <-> Software memory model)

SIDENIS
passion to deliver

Memory ordering semantics

SIDENIS
passion to deliver

Sequential Consistency

— «... результат любого выполнения такой же, как в случае если бы операции всех процессоров были выполнены в некотором последовательном порядке, и операции каждого отдельного процессора появлялись в этой последовательности в порядке определенном его программой.»

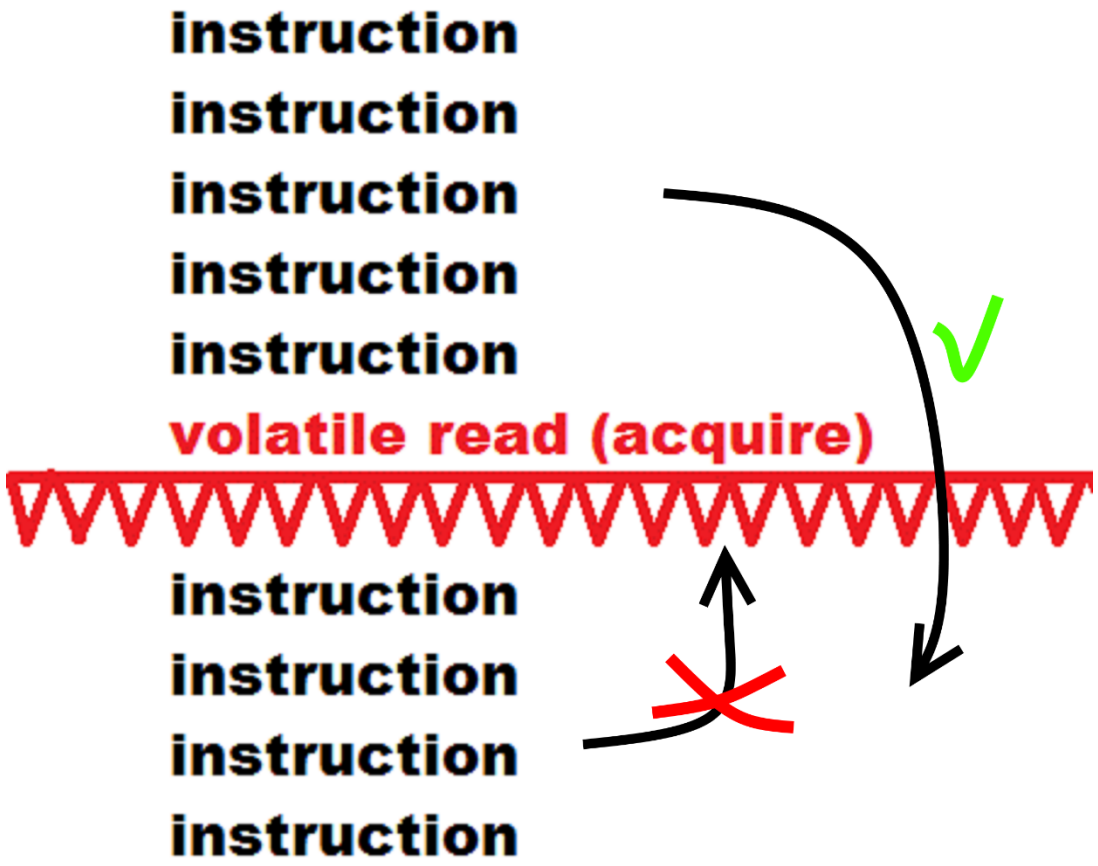
```
processor 1:      <-- A1 run --> <-- B1 run -->          <-- C1 run -->
processor 2:        <-- A2 run --> <-- B2 run -->
Time ------------------------------------------------------------------>
```

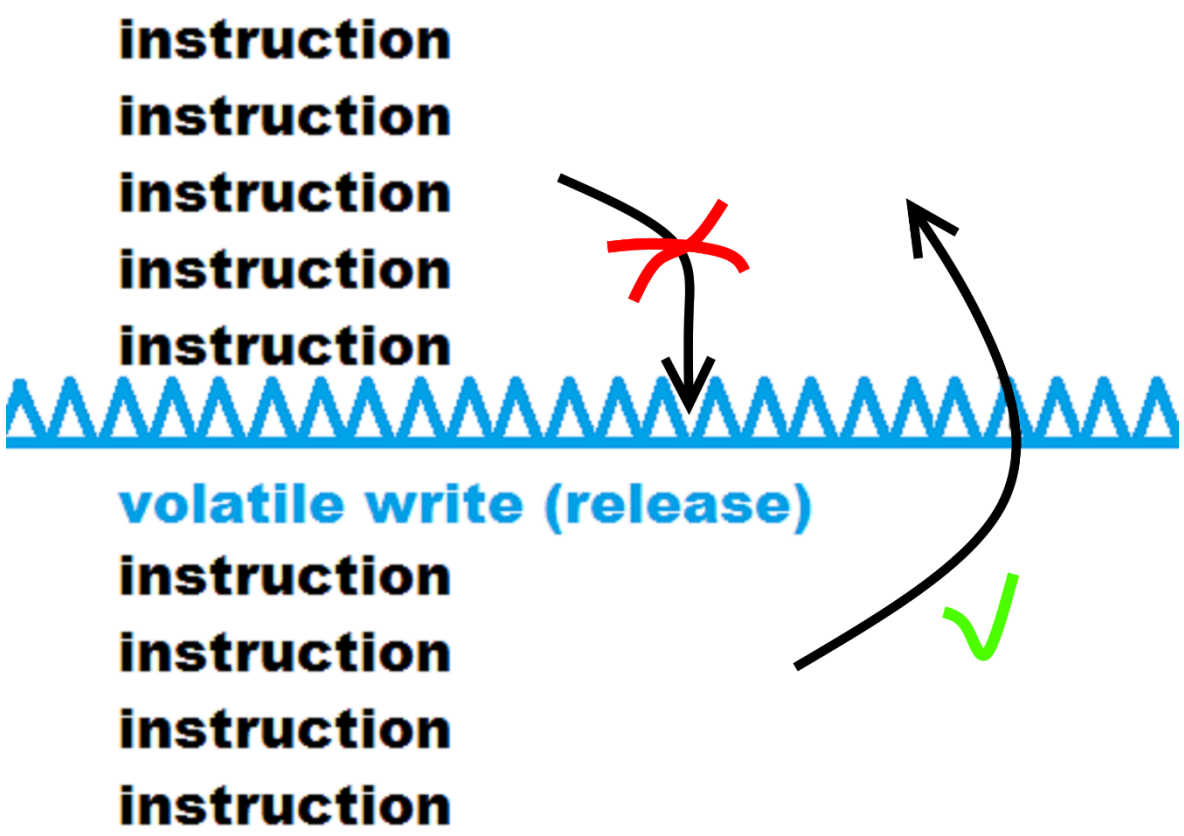
— Медленно, сложно и дорого

SIDENIS
passion to deliver

Release…

Release consistency
Acquire and Release semantics





.NET Standards!

SIDENIS
passion to deliver

ECMA 334 - C#

The C# language was standardized as ECMA 334 in 2002 and approved as ISO/IEC 23270 in 2003.

ECMA 334 Resources

- ECMA 334 Standard Overview
- ECMA 334 Standard (PDF)

ECMA 335 - CLI

Common Language Infrastructure - the formalized basis of .NET -- was standardized as ECMA 335 in 2001 and approved as ISO/IEC 23271 in 2003. The standards have been since updated, to reflect changes in .NET, such as generics.

ECMA 335 Resources

- ECMA 335 Standard Overview
- ECMA 335 Standard (PDF)
- Wikipedia entry on CLI

ECMA 335 Partitions with added Microsoft Specific Implementation Notes

- Partition I: Concepts and Architecture
- Partition II: Meta Data Definition and Semantics
- Partition III: CIL Instruction Set
- Partition IV: Profiles and Libraries
- Partition V: Debug Interchange Format
- Partition VI: Annexes

ECMA Technical Report 084: Information Derived from Partition IV XML File

- ECMA TR/84 Report (PDF)
- ECMA TR/84 Tools and Source Code

SIDENIS
passion to deliver

ECMA-335

I.12.6 Memory model and optimizations

- I.12.6.1 The memory store
- I.12.6.2 Alignment
- I.12.6.3 Byte ordering
- I.12.6.4 Optimization
- I.12.6.5 Locks and threads
- I.12.6.6 Atomic reads and writes
- I.12.6.7 Volatile reads and writes
- I.12.6.8 Other memory model issues

SIDENIS
passion to deliver

ECMA-335 I.12.6.4 Optimization

Conforming implementations of the CLI are **free to execute** programs **using any technology** that guarantees, **within a single thread of execution**, that **side-effects** and exceptions generated by a thread **are visible in the order specified by the CIL**. For this purpose **only volatile** operations (including volatile reads) **constitute visible side-effects**. (Note that while only volatile operations constitute visible side-effects, volatile operations also affect the visibility of non-volatile references.) Volatile operations are specified in §I.12.6.7.

ECMA-335

I.12.6.5 Locks and threads

- System.Threading.Thread.VolatileRead/VolatileWrite/MemoryBarrier
- System.Threading.Volatile.Read/Write
- System.Threading.Interlocked
- System.Threading.Monitor.Enter/Exit

SIDENIS
passion to deliver

SIDENIS
passion to deliver

System.Threading.Volatile

```
15  namespace System.Threading
16  {
17      //
18      // Methods for accessing memory with volatile semantics.  These are preferred over Thread.VolatileRead
19      // and Thread.VolatileWrite, as these are implemented more efficiently.
20      //
21      // (We cannot change the implementations of Thread.VolatileRead/VolatileWrite without breaking code
22      // that relies on their overly-strong ordering guarantees.)
23      //
24      // The actual implementations of these methods are typically supplied by the VM at JIT-time, because C# does
25      // not allow us to express a volatile read/write from/to a byref arg.
26      // See getILIntrinsicImplementationForVolatile() in jitinterface.cpp.
27      //
28      public static class Volatile
```

SIDENIS
passion to deliver

System.Threading.Volatile

```
public static bool Read(ref bool location)
{
    //
    // The VM will replace this with a more efficient implementation.
    //
    var value = location;
    Interlocked.MemoryBarrier();
    return value;
}
```

SIDENIS
passion to deliver

System.Threading.Volatile

```
7107  bool getILIntrinsicImplementationForVolatile(MethodDesc * ftn,
7108                                                CORINFO_METHOD_INFO * methInfo)
7109  {
7110      STANDARD_VM_CONTRACT;

7123      // However, C# does not yet provide a way to declare a byref as "volatile." So instead,
7124      // we substitute raw IL bodies for these methods that use the correct volatile instructions.
7125      //
7126
7127      // Precondition: ftn is a method in mscorlib in the System.Threading.Volatile class
7128      _ASSERTE(ftn->GetModule()->IsSystem());
7129      _ASSERTE(MscorlibBinder::IsClass(ftn->GetMethodTable(), CLASS__VOLATILE));
7130      _ASSERTE(strcmp(ftn->GetMethodTable()->GetClass()->GetDebugClassName(), "System.Threading.Volatile") == 0);
```

ECMA-335

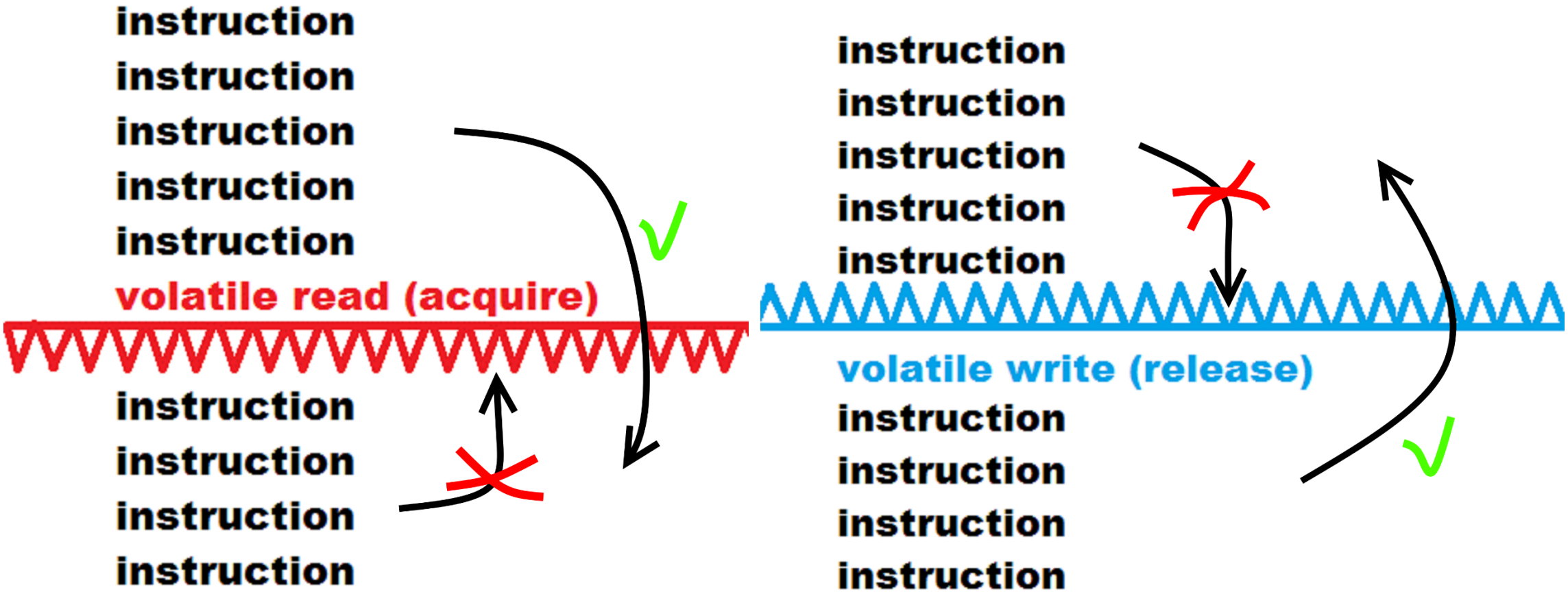
I.12.6.6 Atomic reads and writes

SIDENIS
passion to deliver

A conforming CLI shall guarantee that **read and write** access to **properly aligned** memory locations **no larger than the native word size** (the size of type native int) **is atomic when all the write accesses to a location are the same size**. Atomic writes shall alter no bits other than those written. Unless explicit layout control is used to alter the default behavior, data elements no larger than the natural word size (the size of a native int) shall be properly aligned. **Object references shall be treated as though they are stored in the native word size.**

ECMA-335

I.12.6.7 Volatile reads and writes



ECMA-335

I.12.6.7 Volatile reads and writes

CIL -> native
компилятор

ECMA-335

I.12.6.8 Other memory model issues

SIDENIS
passion to deliver

It is **explicitly not a requirement** that a conforming implementation of the CLI guarantee **that all state updates performed within a constructor be uniformly visible before the constructor completes.** CIL generators can ensure this requirement themselves by inserting appropriate calls to the memory barrier or volatile write instructions.

ECMA-335

I.12.6.8 Other memory model issues

SIDENIS
passion to deliver

It is **explicitly not a requirement** that a conforming implementation of the CLI guarantee **that all state updates performed within a constructor be uniformly visible before the constructor completes.**

CIL generators can ensure this requirement themselves by inserting appropriate calls to the memory barrier or volatile write instructions.

SIDENIS
passion to deliver

ECMA-334
10.5.3 volatile fields

- Чтение – acquire
- Запись – release

SIDENIS
passion to deliver

3

Note that this is off the cuff, without studying your code closely. I don't *think* Set performs a memory barrier, but I don't see how that's relevant in your code? Seems like more important would be if Wait performs one, which it does. So unless I missed something in the 10 seconds I devoted to looking at your code, I don't believe you need the volatiles.

Edit: Comments are too restrictive. I'm now referring to Matt's edit.

Matt did a good job with his evaluation, but he's missing a detail. First, let's provide some definitions of things thrown around, but not clarified here.

A volatile read reads a value and then invalidates the CPU cache. A volatile write flushes the cache, and then writes the value. A memory barrier flushes the cache and then invalidates it.

Простите, что?

The .NET memory model ensures that all writes are volatile. Reads, by default, are not, unless an explicit VolatileRead is made, or the volatile keyword is specified on the field. Further, interlocked methods force cache coherency, and all of the synchronization concepts (Monitor, ReaderWriterLock, Mutex, Semaphore, AutoResetEvent, ManualResetEvent, etc.) call interlocked methods internally, and thus ensure cache coherency.

Again, all of this is from Jeffrey Richter's book, "CLR via C#".

I said, initially, that I didn't *think* Set performed a memory barrier. However, upon further reflection about what Mr. Richter said, Set would be performing an interlocked operation, and would thus also ensure cache coherency.

I stand by my original assertion that volatile is not needed here.

Edit 2: It looks as if you're building a "future". I'd suggest you look into PFX, rather than rolling your own.

share edit flag

edited Mar 27 '09 at 13:20

answered Mar 25 '09 at 18:09

wekempf
2,398 • 9 • 14

SIDENIS
passion to deliver

С чем можно столкнуться?

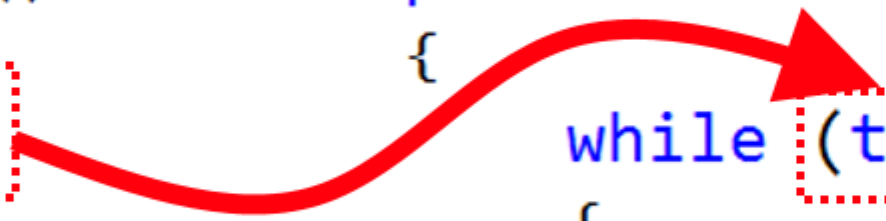
- **Loop Read Hoisting** - while (true)
- **Read Elimination**
- **Copy Propagation** – одна из фаз работы RyuJIT
- **Partially Constructed Object** –ссылка на объект доступна до завершения работы конструктора
- И ещё много чего, что не запрещено спецификацией

SIDENIS
passion to deliver

Loop Read Hoisting

```
private bool flag = true;

public void Loop()
{
    while (flag)
    {
        // Do something
    }
}
```



```
private bool flag = true;

public void Loop()
{
    while (true)
    {
        // Do something
    }
}
```

SIDENIS
passion to deliver

Read Elimination

```
public int ReadElimination()
{
    if (x == 0)
        throw new Exception();

    var result = y + 42;
    result += x;

    return result;
}
```

```
public int ReadElimination()
{
    var tmp = x;
    if (tmp == 0)
        throw new Exception();

    var result = y + 42;
    result += tmp;

    return result;
}
```

SIDENIS
passion to deliver

Copy Propagation

```
public void CopyPropagation()
{
    c.S = date;
    c.S2 = c.S;
    CheckStrings(c);
}

public void CheckStrings(Container a)
{
    if (a.S.Length == 0 || a.S2.Length == 0)
    { /* Do something */ }
}
```

```
public void CopyPropagation()
{
    c.S = date;
    c.S2 = date;
    if (date.Length == 0 || date.Length == 0)
    { /* Do something */ }
}
```

Optimization Phases of RyuJIT

SIDENIS
passion to deliver

Демо

- Сейчас демо про partially constructed object на ARM
- Далее будут примеры из **CoreFX, RavenDb**

SIDENIS
passion to deliver

ConcurrentDictionary

C# Volatile read behavior

4

In the reference source code of the C#.net ConcurrentDictionary (C# reference source), I don't understand why a volatile read is required in the following code snippet:

```
public bool TryGetValue(TKey key, out TValue value)
{
    if (key == null) throw new ArgumentNullException("key");
      int bucketNo, lockNoUnused;

    // We must capture the m_buckets field in a local variable.
    It is set to a new table on each table resize.
    Tables tables = m_tables;
    IEqualityComparer<TKey> comparer = tables.m_comparer;
    GetBucketAndLockNo(comparer.GetHashCode(key),
                       out bucketNo,
                       out lockNoUnused,
                       tables.m_buckets.Length,
                       tables.m_locks.Length);

    // We can get away w/out a lock here.
    // The Volatile.Read ensures that the load of the fields of 'n'
    //doesn't move before the load from buckets[i].
    Node n = Volatile.Read<Node>(ref tables.m_buckets[bucketNo]);
```

SIDENIS
passion to deliver

ConcurrentDictionary

```
485         /// <summary>
486         /// Attempts to get the value associated with the specified key from the <see
487         /// cref="ConcurrentDictionary{TKey,TValue}"/>.
488         /// </summary>
489         /// <param name="key">The key of the value to get.</param>
490         /// <param name="value">When this method returns, <paramref name="value"/> contains the object from
491         /// the
492         /// <see cref="ConcurrentDictionary{TKey,TValue}"/> with the specified key or the default value of
493         /// <typeparamref name="TValue"/>, if the operation failed.</param>
494         /// <returns>true if the key was found in the <see cref="ConcurrentDictionary{TKey,TValue}"/>;
495         /// otherwise, false.</returns>
496         /// <exception cref="T:System.ArgumentNullException"><paramref name="key"/> is a null reference
497         /// (Nothing in Visual Basic).</exception>
498         [SuppressMessage("Microsoft.Concurrency", "CA8001", Justification = "Reviewed for thread safety")]
499         public bool TryGetValue(TKey key, out TValue value)
500         {
```

SIDENIS
passion to deliver

ConcurrentDictionary

```
private bool TryGetValueInternal(TKey key, int hashcode, out TValue value)
{
    Debug.Assert(_comparer.GetHashCode(key) == hashcode);

    // We must capture the _buckets field in a local variable. It is set to a new table on each table resize.
    Tables tables = _tables;

    int bucketNo = GetBucket(hashcode, tables._buckets.Length);

    // We can get away w/out a lock here.
    // The Volatile.Read ensures that the load of the fields of 'n' doesn't move before the load from buckets[i].
    Node n = Volatile.Read<Node>(ref tables._buckets[bucketNo]);

    while (n != null)
    {
        if (hashcode == n._hashcode && _comparer.Equals(n._key, key))
        {
            value = n._value;
            return true;
        }
        n = n._next;
    }

    value = default(TValue);
    return false;
}
```

SIDENIS
passion to deliver

ConcurrentDictionary

```
private bool TryGetValueInternal(TKey key, int hashcode, out TValue value)
{
    Debug.Assert(_comparer.GetHashCode(key) == hashcode);

    // We must capture the _buckets field in a local variable. It is set to a new table on each table resize.
    Tables tables = _tables;

    int bucketNo = GetBucket(hashcode, tables._buckets.Length);

    // We can get away w/out a lock here.
    // The Volatile.Read ensures that the load of the fields of 'n' doesn't move before the load from buckets[i].
    Node n = Volatile.Read<Node>(ref tables._buckets[bucketNo]);

    while (n != null)
    {
        if (hashcode == n._hashcode && _comparer.Equals(n._key, key))
        {
            value = n._value;
            return true;
        }
        n = n._next;
    }

    value = default(TValue);
    return false;
}
```

SIDENIS
passion to deliver

ConcurrentDictionary

```
Node newNode = new Node(node._key, newValue, hashcode, node._next);

if (prev == null)
{
    tables._buckets[bucketNo] = newNode;
}
else
{
    prev._next = newNode;
}
```

```
Node newNode = new Node(node._key, value, hashcode, node._next);
if (prev == null)
{
    tables._buckets[bucketNo] = newNode;
}
else
{
    prev._next = newNode;
}
```

SIDENIS
passion to deliver

ConcurrentDictionary

- TryAddInternal:

```
Monitor.Enter(tables._locks[lockNo], ref lockTaken)
```

- TryUpdateInternal:

```
lock (tables._locks[lockNo])
```

SIDENIS
passion to deliver

ConcurrentDictionary

```
/// <summary>
/// A node in a singly-linked list representing a particular hash table bucket.
/// </summary>
private sealed class Node
{
    internal readonly TKey _key;
    internal TValue _value;
    internal volatile Node _next;
    internal readonly int _hashcode;

    internal Node(TKey key, TValue value, int hashcode, Node next)
    {
        _key = key;
        _value = value;
        _next = next;
        _hashcode = hashcode;
    }
}
```

SIDENIS
passion to deliver

ConcurrentDictionary

```
/// <summary>
/// A node in a singly-linked list representing a particular hash table bucket.
/// </summary>
private sealed class Node
{
    internal readonly TKey _key;
    internal TValue _value;
    internal volatile Node _next;
    internal readonly int _hashcode;

    internal Node(TKey key, TValue value, int hashcode, Node next)
    {
        _key = key;
        _value = value;
        _next = next;
        _hashcode = hashcode;
    }
}
```

SIDENIS
passion to deliver

ConcurrentDictionary

```
private bool TryGetValueInternal(TKey key, int hashcode, out TValue value)
{
    Debug.Assert(_comparer.GetHashCode(key) == hashcode);

    // We must capture the _buckets field in a local variable. It is set to a new table on each table resize.
    Tables tables = _tables;

    int bucketNo = GetBucket(hashcode, tables._buckets.Length);

    // We can get away w/out a lock here.
    // The Volatile.Read ensures that the load of the fields of 'n' doesn't move before the load from buckets[i].
    Node n = Volatile.Read<Node>(ref tables._buckets[bucketNo]);

    while (n != null)
    {
        if (hashcode == n._hashcode && _comparer.Equals(n._key, key))
        {
            value = n._value;
            return true;
        }
        n = n._next;
    }

    value = default(TValue);
    return false;
}
```

SIDENIS
passion to deliver

ConcurrentDictionary

```
if (prev == null)
{
    tables._buckets[bucketNo] = newNode;
    Volatile.Write(ref tables._buckets[bucketNo], newNode);
}
else
{
```

https://github.com/dotnet/corefx/pull/22382

ConcurrentDictionary

stephentoub approved these changes on Jul 18

stephentoub merged commit **e3e1322** into `dotnet:master` on Jul 18

11 checks passed

SIDENIS
passion to deliver

Статический анализ, Roslyn, бага в RavenDB

- github.com/m08pvv/CSharpPartiallyConstructedObjectAnalyzer
- https://github.com/ravendb/ravendb/pull/3376

```
if (keyGeneratorsByTag.TryGetValue(tag, out value))
    return value.GenerateDocumentKey(databaseCommands, conventions, entity);

lock (generatorLock)
{
    if (keyGeneratorsByTag.TryGetValue(tag, out value))
        return value.GenerateDocumentKey(databaseCommands, conventions, entity);

    value = new HiLoKeyGenerator(tag, capacity);
    // doing it this way for thread safety
    keyGeneratorsByTag = new Dictionary<string, HiLoKeyGenerator>(keyGeneratorsByTag)
    {
        {tag, value}
    };
}
```

SIDENIS
passion to deliver

RavenDB

```
-       private IDictionary<string, HiLoKeyGenerator> keyGeneratorsByTag = new Dictionary<string, HiLoKeyGenerator>();
+       private volatile IDictionary<string, HiLoKeyGenerator> keyGeneratorsByTag = new Dictionary<string, HiLoKeyGenerator>();
```

SIDENIS
passion to deliver

Вывод

- Стараться использовать более высокоуровневые конструкции
- Читать спецификации

SIDENIS
passion to deliver

Что почитать

- ECMA-335 и ECMA-334
- http://www.albahari.com/threading/
- Andrew Tanenbaum “Structured Computer Organization”
- Jeffrey Richter “CLR via C#”
- Eugene Agafonov, Andrew Koryavchenko "Mastering C# Concurrency"
- https://habrahabr.ru/company/intel/blog/

SIDENIS
passion to deliver

Вопросы?

m08pvv

SIDENIS
passion to deliver

Бонус

While Microsoft's CLR team was building a JIT compiler for the **IA64** architecture, they realized that **many developers (including themselves) had written code that would not work correctly if executed with non-volatile (unordered) read and write memory accesses**. So they thought about making **the IA64 JIT** compiler always produce read instructions that include acquire semantics and write instructions that always include release semantics. This would allow already-written applications that work on the x86 to continue to work just fine on the IA64. Unfortunately, this would hurt performance significantly, and so a compromise was struck. **Microsoft's IA64 JIT compiler** ensures that all writes are always performed with release semantics, but reads are allowed to be unordered; programmers still must call Thread's VolatileRead method or apply C#'s volatile keyword to a field so that it is read with acquire semantics.

SIDENIS
passion to deliver

Бонус

This produces a much more sane memory model for programmers to rationalize in their heads. In fact, **Microsoft now promises** that **all of its JIT compilers that exist today or are created in the future will adhere to this memory model** in which reads can be unordered but **writes always occur with release semantics**. To be more precise, the memory access rules are a bit more complicated than what I discussed in this section. The other memory access rules would be expected by most programmers anyway.

SIDENIS
passion to deliver

Список литературы:

- **The C# Memory Model in Theory and Practice** [https://msdn.microsoft.com/en-us/magazine/jj883956.aspx]
- **ECMA-335 specification** [http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-335.pdf]
- **ECMA-334 specification** [%VSINSTALLDIR%\VC#\Specifications\1033\CSharp Language Specification.docx]
- **Jeffrey Richter “CLR via C#, Second Edition”**