.NET Core

Intrinsics and other micro-optimizations

Egor Bogatov

Engineer at Microsoft

Agenda

Useful micro-optimizations

Pitfalls for external contributors

Intrinsics & SIMD with examples

.NET Core 3.0-x features

Prefer Spans API where possible

```
var str = "EGOR 3.14 1234 7/3/2018";

string name   = str.Substring(0, 4);
float pi      = float.Parse(str.Substring(5, 4));
int number    = int.Parse(str.Substring(10, 4));
DateTime date = DateTime.Parse(str.Substring(15, 8));
```

Allocated on heap: 168 bytes

```
var str = "EGOR 3.14 1234 7/3/2018".AsSpan();

var name      = str.Slice(0, 4);
float pi      = float.Parse(str.Slice(5, 4));
int number    = int.Parse(str.Slice(10, 4));
DateTime date = DateTime.Parse(str.Slice(15, 8));
```

Allocated on heap: 0 bytes

Allocating a temp array

```
char[] buffer =
    new char[count];
```

Allocating a temp array

```
Span<char> span =
    new char[count];
```

Allocating a temp array

```
Span<char> span =
    count <= 512 ?
    stackalloc char[512] :
    new char[count];
```

Allocating a temp array

```
Span<char> span =
    count <= 512 ?
    stackalloc char[512] :
    ArrayPool<char>.Shared.Rent(count);
```

Allocating a temp array

```
char[] pool = null;
Span<char> span =
    count <= 512 ?
    stackalloc char[512] :
    (pool = ArrayPool<char>.Shared.Rent(count));

 if (pool != null)
    ArrayPool<char>.Shared.Return(pool);
```

Allocating a temp array - final pattern

```
char[] pool = null;
Span<char> span =
    count <= 512 ?
    stackalloc char[512] :
    (pool = ArrayPool<char>.Shared.Rent(count));

 if (pool != null)
    ArrayPool<char>.Shared.Return(pool);
```

Allocating a temp array – without ArrayPool

```
Span<char> span = count <= 512 ?
                  stackalloc char[512] :
                  new char[count];
```

Optimizing .NET Core: pitfalls

Check for IReadOnlyCollection #28472

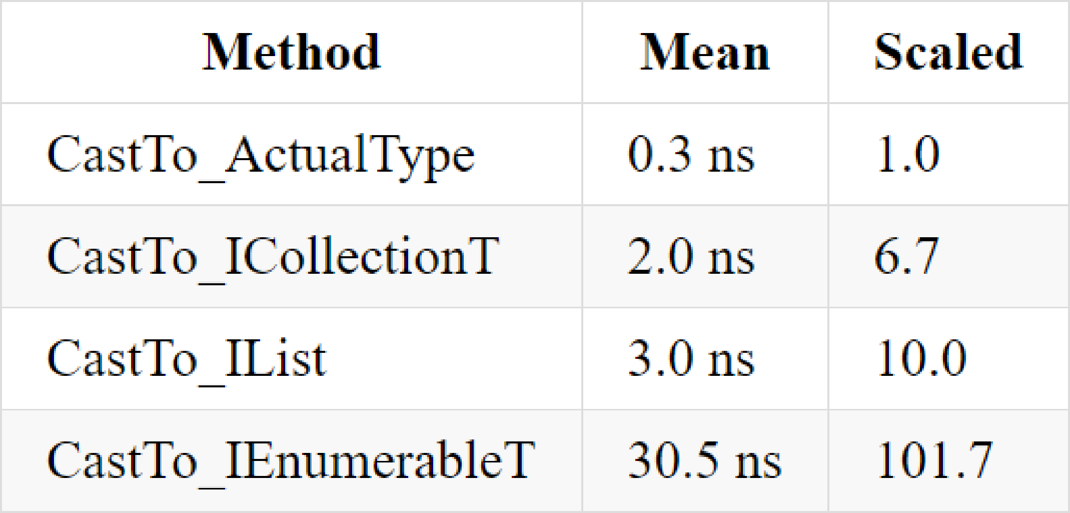
Closed danielearwicker wants to merge 2 commits into `dotnet:master` from `danielearwicker:readonlyCount`

```
           public static int Count<TSource>(this IEnumerable<TSource> source)
           {
   ~ 3 ns      if (source is ICollection<TSource> collectionoft)
                   return collectionoft.Count;

   ~ 3 ns      if (source is IIListProvider<TSource> listProv)
                   return listProv.GetCount(onlyIfCheap: false);

   ~ 3 ns      if (source is ICollection collection)
                   return collection.Count;

  ~ 30 ns      if (source is IReadOnlyCollection<TSource> rocollectionoft)
                   return rocollectionoft.Count;

               int count = 0;
~ 10-... ns    using (IEnumerator<TSource> e = source.GetEnumerator())
                   while (e.MoveNext())
                       count++;

               return count;
           }
```

Casts are not cheap

```
object value = new List<string> { };

var t0 = (List<string>)value;
var t1 = (ICollection<string>)value
var t2 = (IList)value
var t3 = (IEnumerable<string>)value
```

```
// Covariant interfaces:
public interface IEnumerable<out T>
public interface IReadOnlyCollection<out T>

IEnumerable<object> a = new List<string> {..}
```



Method	Mean	Scaled
CastTo_ActualType	0.3 ns	1.0
CastTo_ICollectionT	2.0 ns	6.7
CastTo_IList	3.0 ns	10.0
CastTo_IEnumerableT	30.5 ns	101.7

Cast to covariant interface – different runtimes

```
return ((IReadOnlyCollection<string>)_smallArray).Count;
```

```
 Method             |    Runtime    |    Mean | Scaled |
-------------------:|----------------:|------------:|---------:|
 CastAndCount |       .NET 4.7 |  78.1 ns |      6.7 |
 CastAndCount | .NET Core 3 |  42.9 ns |      3.7 |
 CastAndCount |       CoreRT  |  11.6 ns |      1.0 |
 CastAndCount |         Mono  |    6.7 ns |      0.6 |
```

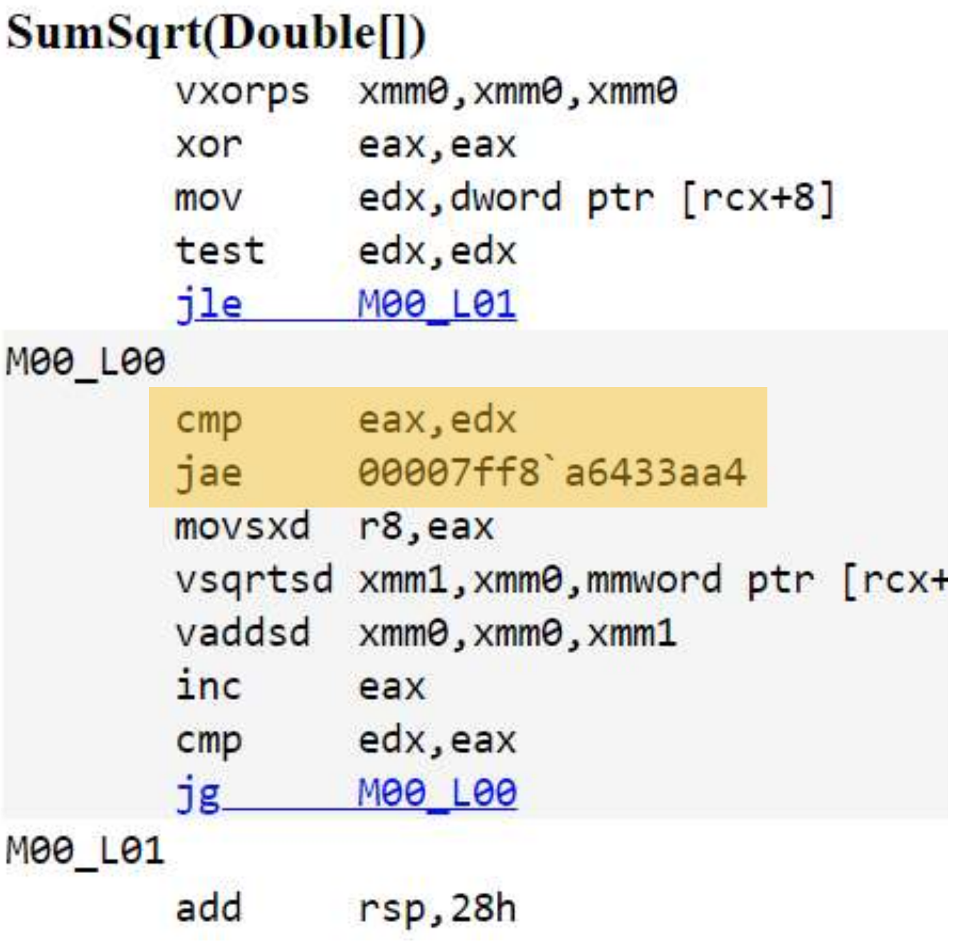
.NET Core: bounds check

Bounds check

```
public static double SumSqrt(double[] array)
{
    double result = 0;
    for (int i = 0; i < array.Length; i++)
    {


        result += Math.Sqrt(array[i]);
    }

    return result;
}
```

Bounds check

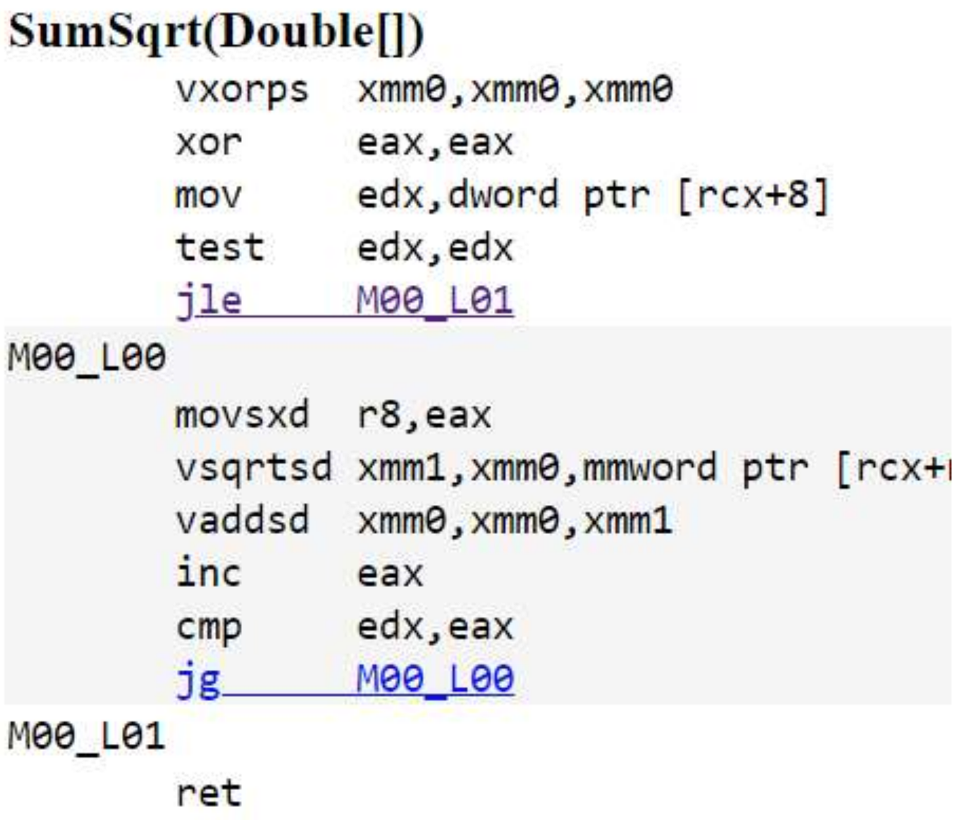
```
public static double SumSqrt(double[] array)
{
    double result = 0;
    for (int i = 0; i < array.Length; i++)
    {
        if (i >= array.Length)
            throw new ArgumentOutOfRangeException();
        result += Math.Sqrt(array[i]);
    }

    return result;
}
```



```
SumSqrt(Double[])
          vxorps   xmm0,xmm0,xmm0
          xor      eax,eax
          mov      edx,dword ptr [rcx+8]
          test     edx,edx
          jle      M00_L01
M00_L00
          cmp      eax,edx
          jae      00007ff8`a6433aa4
          movsxd   r8,eax
          vsqrtsd  xmm1,xmm0,mmword ptr [rcx+
          vaddsd   xmm0,xmm0,xmm1
          inc      eax
          cmp      edx,eax
          jg       M00_L00
M00_L01
          add      rsp,28h
```

Bounds check eliminated!

```
public static double SumSqrt(double[] array)
{
    double result = 0;
    for (int i = 0; i < array.Length; i++)
    {


        result += Math.Sqrt(array[i]);
    }

    return result;
}
```

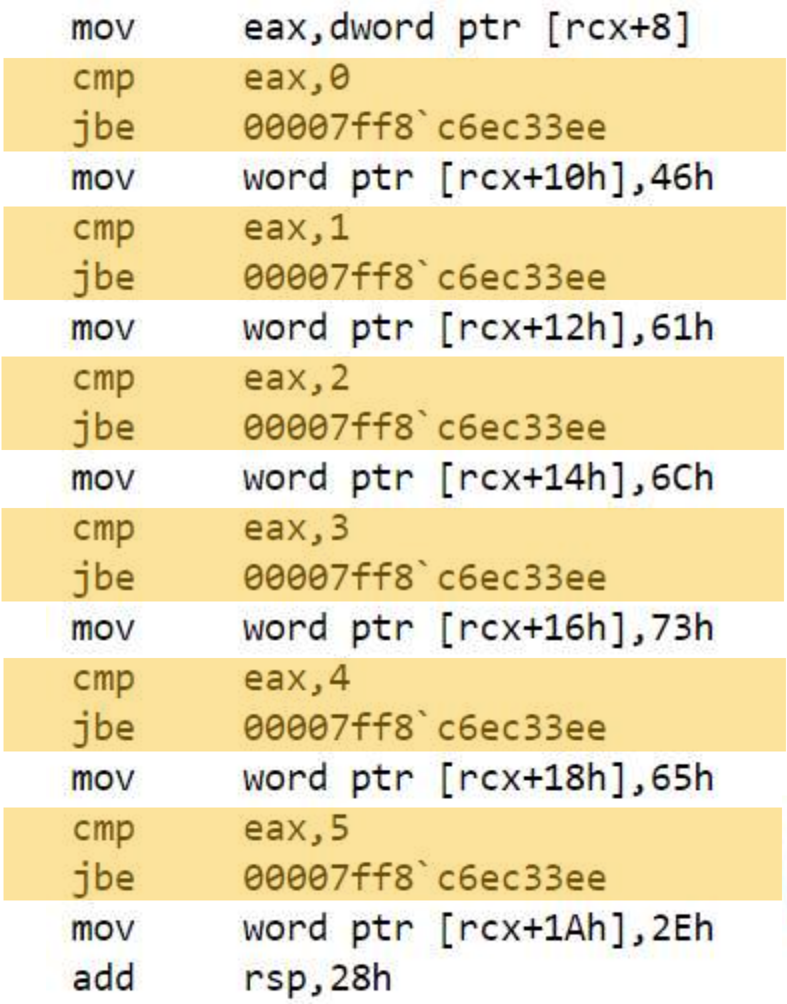


SumSqrt(Double[])

```
        vxorps  xmm0,xmm0,xmm0
        xor     eax,eax
        mov     edx,dword ptr [rcx+8]
        test    edx,edx
        jle     M00_L01
M00_L00
        movsxd  r8,eax
        vsqrtsd xmm1,xmm0,mmword ptr [rcx+
        vaddsd  xmm0,xmm0,xmm1
        inc     eax
        cmp     edx,eax
        jg      M00_L00
M00_L01
        ret
```

Bounds check: tricks

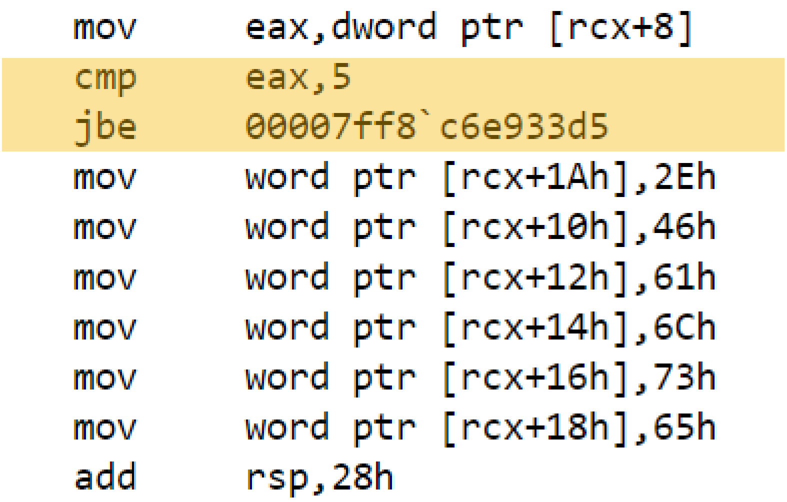
```
public static void Test1(char[] array)
{
    array[0] = 'F';
    array[1] = 'a';
    array[2] = 'l';
    array[3] = 's';
    array[4] = 'e';
    array[5] = '.';
}
```



```
mov         eax,dword ptr [rcx+8]
cmp         eax,0
jbe         00007ff8`c6ec33ee
mov         word ptr [rcx+10h],46h
cmp         eax,1
jbe         00007ff8`c6ec33ee
mov         word ptr [rcx+12h],61h
cmp         eax,2
jbe         00007ff8`c6ec33ee
mov         word ptr [rcx+14h],6Ch
cmp         eax,3
jbe         00007ff8`c6ec33ee
mov         word ptr [rcx+16h],73h
cmp         eax,4
jbe         00007ff8`c6ec33ee
mov         word ptr [rcx+18h],65h
cmp         eax,5
jbe         00007ff8`c6ec33ee
mov         word ptr [rcx+1Ah],2Eh
add         rsp,28h
```

Bounds check: tricks

```
public static void Test1(char[] array)
{
    array[5] = '.';
    array[0] = 'F';
    array[1] = 'a';
    array[2] = 'l';
    array[3] = 's';
    array[4] = 'e';
}
```

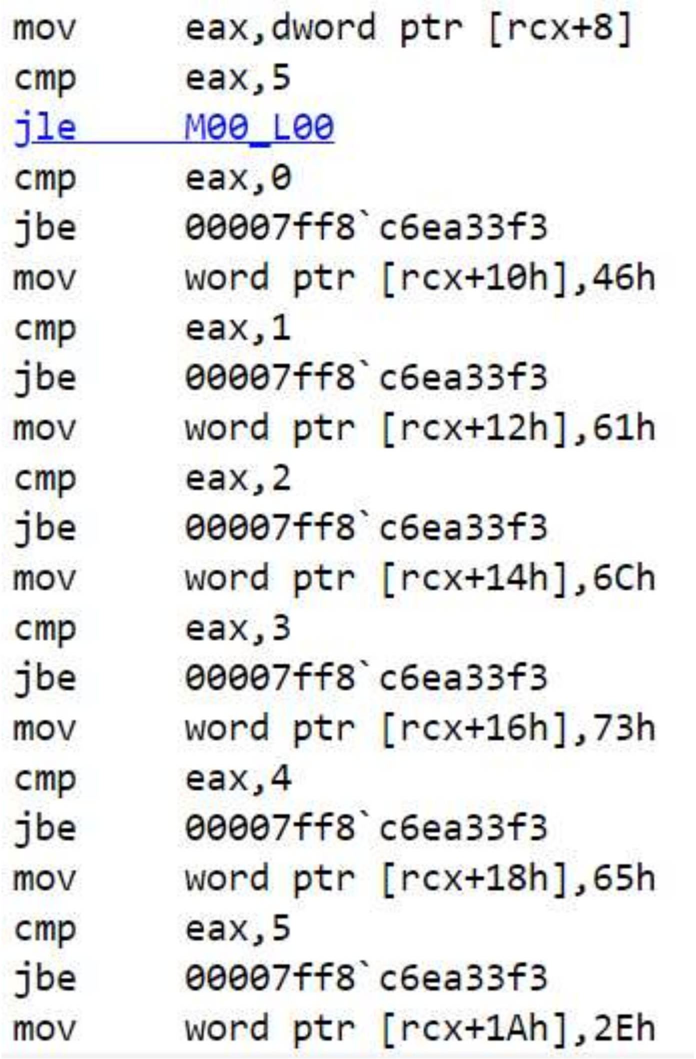


```
mov     eax,dword ptr [rcx+8]
cmp     eax,5
jbe     00007ff8`c6e933d5
mov     word ptr [rcx+1Ah],2Eh
mov     word ptr [rcx+10h],46h
mov     word ptr [rcx+12h],61h
mov     word ptr [rcx+14h],6Ch
mov     word ptr [rcx+16h],73h
mov     word ptr [rcx+18h],65h
add     rsp,28h
```

👍

Bounds check: tricks

```
public static void Test1(char[] array)
{
    if (array.Length > 5)
    {
        array[0] = 'F';
        array[1] = 'a';
        array[2] = 'l';
        array[3] = 's';
        array[4] = 'e';
        array[5] = '.';
    }
}
```

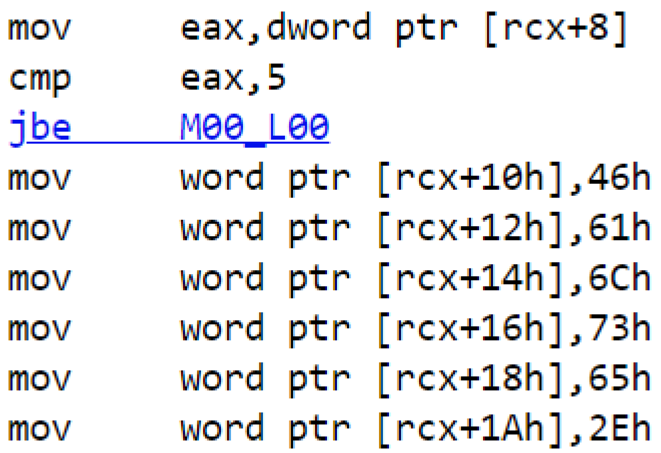


```
mov     eax,dword ptr [rcx+8]
cmp     eax,5
jle     M00_L00
cmp     eax,0
jbe     00007ff8`c6ea33f3
mov     word ptr [rcx+10h],46h
cmp     eax,1
jbe     00007ff8`c6ea33f3
mov     word ptr [rcx+12h],61h
cmp     eax,2
jbe     00007ff8`c6ea33f3
mov     word ptr [rcx+14h],6Ch
cmp     eax,3
jbe     00007ff8`c6ea33f3
mov     word ptr [rcx+16h],73h
cmp     eax,4
jbe     00007ff8`c6ea33f3
mov     word ptr [rcx+18h],65h
cmp     eax,5
jbe     00007ff8`c6ea33f3
mov     word ptr [rcx+1Ah],2Eh
```

👎

Bounds check: tricks

```
public static void Test1(char[] array)
{
    if ((uint)array.Length > 5)
    {
        array[0] = 'F';
        array[1] = 'a';
        array[2] = 'l';
        array[3] = 's';
        array[4] = 'e';
        array[5] = '.';
    }
}
```



```
mov         eax,dword ptr [rcx+8]
cmp         eax,5
jbe         M00_L00
mov         word ptr [rcx+10h],46h
mov         word ptr [rcx+12h],61h
mov         word ptr [rcx+14h],6Ch
mov         word ptr [rcx+16h],73h
mov         word ptr [rcx+18h],65h
mov         word ptr [rcx+1Ah],2Eh
```

👍

Bounds check: tricks – CoreCLR sources:

```
// Boolean.cs

public bool TryFormat(Span<char> destination, out int charsWritten)
{
    if (m_value)
    {
        if ((uint)destination.Length > 3)
        {
            destination[0] = 'T';
            destination[1] = 'r';
            destination[2] = 'u';
            destination[3] = 'e';
            charsWritten = 4;
            return true;
        }
    }
```

.NET Core: Intrinsics & SIMD

Intrinsics

- Recognize patterns

```
private static uint Rotl(uint value, int shift)
{
    return (value << shift) | (value >> (32 - shift));
}
```

```
mov   eax,dword ptr [rcx+8]
mov   ecx,dword ptr [rcx+0Ch]
rol   eax,cl
ret
```

- Replace methods (usually marked with [Intrinsic])

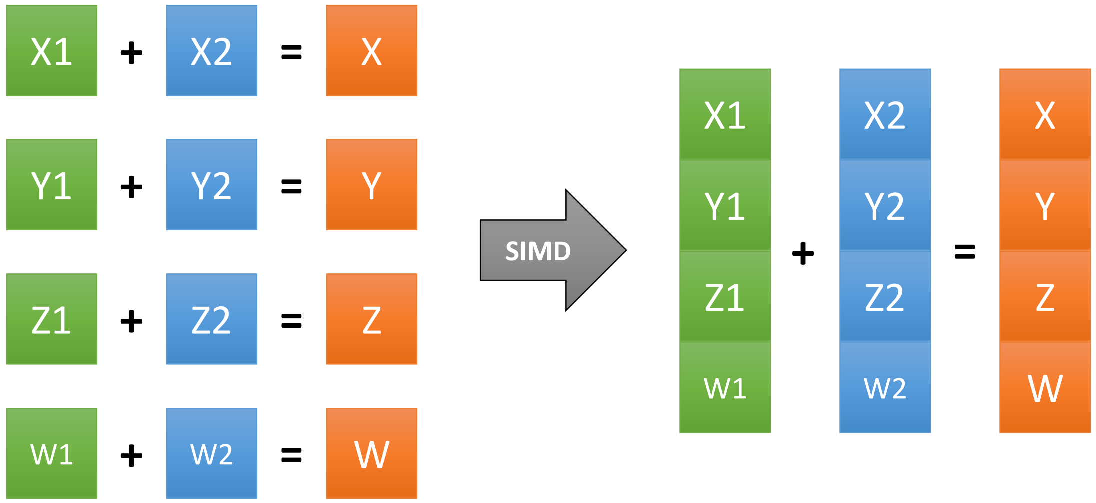
```
[Intrinsic]
public static double Round(double a)
{
    double flrTempVal = Floor(a + 0.5);
    if ((a == (Floor(a) + 0.5)) && (FMod(flrTempVal, 2.0) != 0))
        flrTempVal -= 1.0;
    return copysign(flrTempVal, a);
}
```

```
cmp       dword ptr [rcx+48h] …
jne       M00_L00
vroundsd  xmm0,xmm0,mmword ptr …
ret
```

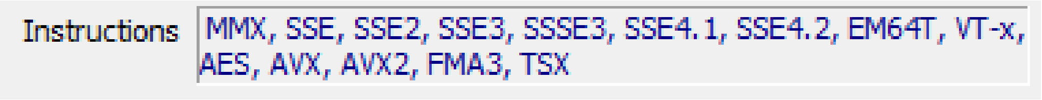
- System.Runtime.Intrinsics

SIMD

```
Vector4 result =
        new Vector4(1f, 2f, 3f, 4f) +
        new Vector4(5f, 6f, 7f, 8f);
```



```
vmovups    xmm0,xmmword ptr [rdx]
vmovups    xmm1,xmmword ptr [rdx+16]
vaddps     xmm0,xmm0,xmm1
```



Instructions MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3, TSX

Meet **System.Runtime.Intrinsics**

```
var v1 =     new Vector4(1, 2, 3, 4);
var v2 =     new Vector4(5, 6, 7, 8);
var result = new Vector4(v1.X + v2.X, v1.Y + v2.Y, ...);
```

```
var left =   Sse.LoadVector128(&v1.X); // Vector128<float>
var right =  Sse.LoadVector128(&v2.X);
var sum =    Sse.Add(left, right);
Sse.Store(&result.X, sum);

var mulPi = Sse.Multiply(sum, Sse.SetAllVector128(3.14f));
```

System.Runtime.Intrinsics

- System.Runtime.Intrinsics

  Vector64<T>

  Vector128<T>

  Vector256<T>

- System.Runtime.Intrinsics.X86

  Sse (Sse, Sse2…Sse42)

  Avx, Avx2

  Fma

  …

- System.Runtime.Intrinsics.Arm.Arm64

  Simd

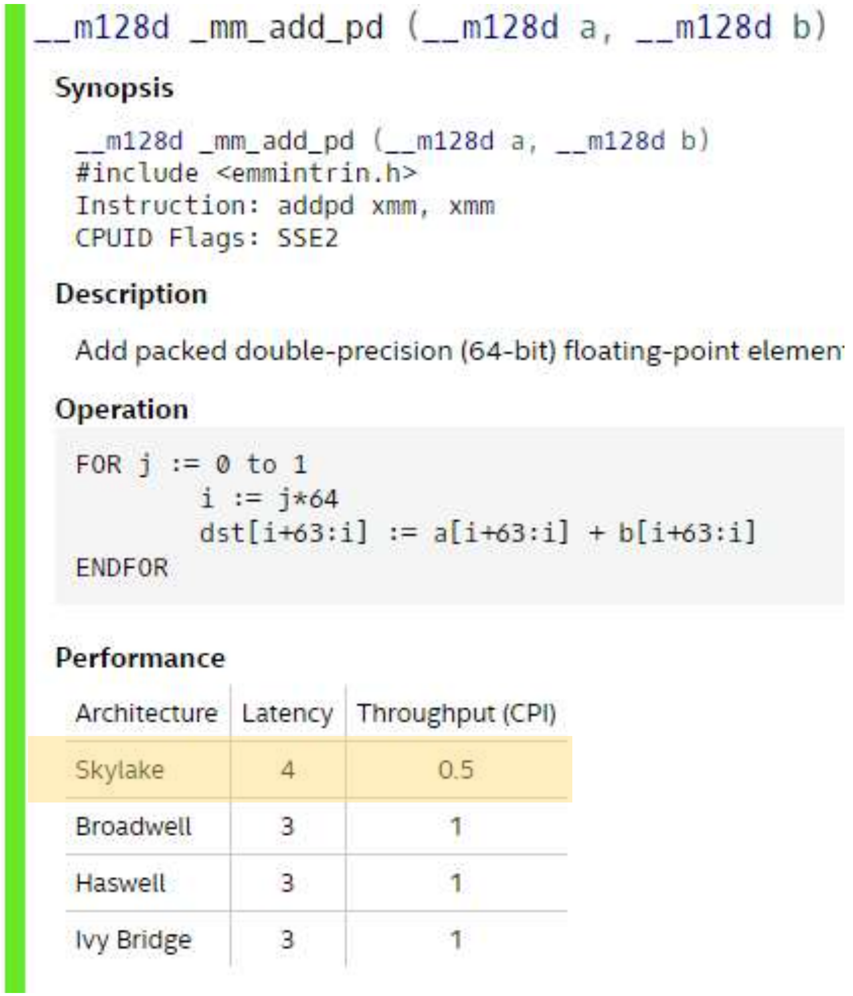
  …

System.Runtime.Intrinsics

```
public class Sse2 : Sse
{
    public static bool IsSupported => true;

    /// <summary>
    /// __m128i _mm_add_epi8 (__m128i a,  __m128i b)
    ///   PADDB xmm, xmm/m128
    /// </summary>
    public static Vector128<byte> Add(Vector128<byte> left, Vector128<byte> right);

    /// <summary>
    /// __m128i _mm_add_epi8 (__m128i a,  __m128i b)
    ///   PADDB xmm, xmm/m128
    /// </summary>
    public static Vector128<sbyte> Add(Vector128<sbyte> left, Vector128<sbyte> right);
```

S.R.I.: Documentation

```
/// <summary>
/// __m128d _mm_add_pd (__m128d a,  __m128d b)
///   ADDPD xmm, xmm/m128
/// </summary>
public static Vector128<double> Add(
   Vector128<double> left,
   Vector128<double> right);
```



```
__m128d _mm_add_pd (__m128d a, __m128d b)
```

Synopsis

```
__m128d _mm_add_pd (__m128d a, __m128d b)
#include <emmintrin.h>
Instruction: addpd xmm, xmm
CPUID Flags: SSE2
```

Description

Add packed double-precision (64-bit) floating-point elemen

Operation

```
FOR j := 0 to 1
        i := j*64
        dst[i+63:i] := a[i+63:i] + b[i+63:i]
ENDFOR
```

Performance

Architecture	Latency	Throughput (CPI)
Skylake	4	0.5
Broadwell	3	1
Haswell	3	1
Ivy Bridge	3	1

S.R.I.: Usage pattern

```
if (Arm.Simd.IsSupported)
    DoWorkusingNeon();
else if (Avx2.IsSupported)
    DoWorkUsingAvx2();
else if (Sse2.IsSupported)
    DoWorkUsingSse2();
else
    DoWorkSlowly();
```

JIT

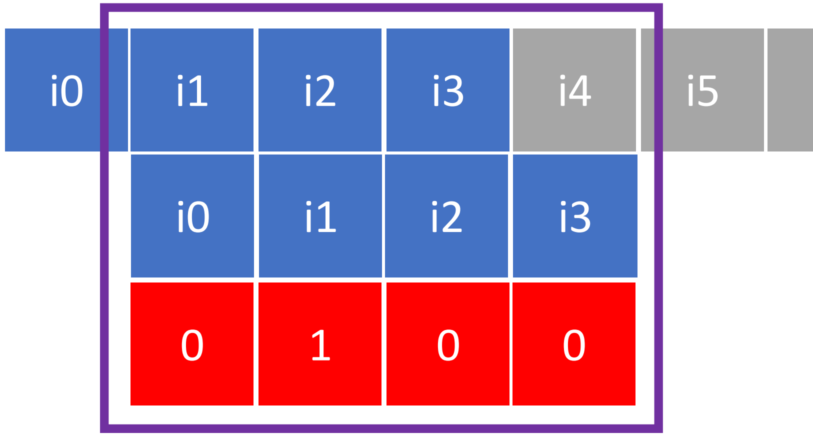
```
if (Arm.Simd.IsSupported)
    DoWorkusingNeon();
else if (x86.Avx2.IsSupported)
    DoWorkUsingAvx2();
else if (x86.Sse2.IsSupported)
    DoWorkUsingSse2();
else
    DoWorkSlowly();
```

IsSorted(int[]) – simple implementation

```
bool IsSorted(int[] array)
{
    if (array.Length < 2)
        return true;

    for (int i = 0; i < array.Length - 1; i++)
    {
        if (array[i] > array[i + 1])
            return false;
    }
    return true;
}
```

IsSorted(int[]) – optimized with SSE41

```
bool IsSorted_Sse41(int[] array)
{
    fixed (int* ptr = &array[0])
    {
        for (int i = 0; i < array.Length - 4; i += 4)
         {
             var curr = Sse2.LoadVector128(ptr + i);
             var next = Sse2.LoadVector128(ptr + i + 1);
             var mask = Sse2.CompareGreaterThan(curr, next);
             if (!Sse41.TestAllZeros(mask, mask))
                 return false;
         }

    }
    return true;
}
```



_mm_test_all_zeros

```
          Method |      Mean |
---------------- |---------:|
        IsSorted | 35.07 us |
 IsSorted_unsafe | 21.19 us |
  IsSorted_Sse41 | 13.79 us |
```

Reverse<T>(T[] array), level: student

```
void Reverse<T>(T[] array)
{
    for (int i = 0; i < array.Length / 2; i++)
    {
        T tmp = array[i];
        array[i] = array[array.Length - i - 1];
        array[array.Length - i - 1] = tmp;
    }
}
```

“1 2 3 4 5 6” => “6 5 4 3 2 1”

Reverse<T>(T[] array), level: CoreCLR developer

```
void Reverse<T>(T[] array)
{
    ref T p = ref Unsafe.As<byte, T>(ref array.GetRawSzArrayData());
    int i = 0;
    int j = array.Length - 1;
    while (i < j)
    {
        T temp = Unsafe.Add(ref p, i);
        Unsafe.Add(ref p, i) = Unsafe.Add(ref p, j);
        Unsafe.Add(ref p, j) = temp;
        i++;
        j--;
    }
}
```

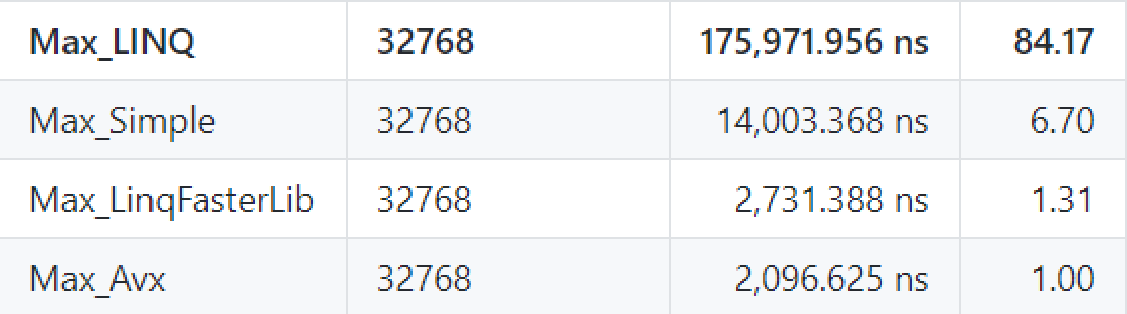
No bounds/covariance checks

Reverse<T>(T[] array), level: SSE-maniac

```
int* leftPtr = ptr + i;
int* rightPtr = ptr + len - vectorSize - i;

var left = Sse2.LoadVector128(leftPtr);
var right = Sse2.LoadVector128(rightPtr);

var reversedLeft =  Sse2.Shuffle(left, 0x1b); //0x1b =_MM_SHUFFLE(0,1,2,3)
var reversedRight = Sse2.Shuffle(right, 0x1b);

Sse2.Store(rightPtr, reversedLeft);
Sse2.Store(leftPtr, reversedRight);
```

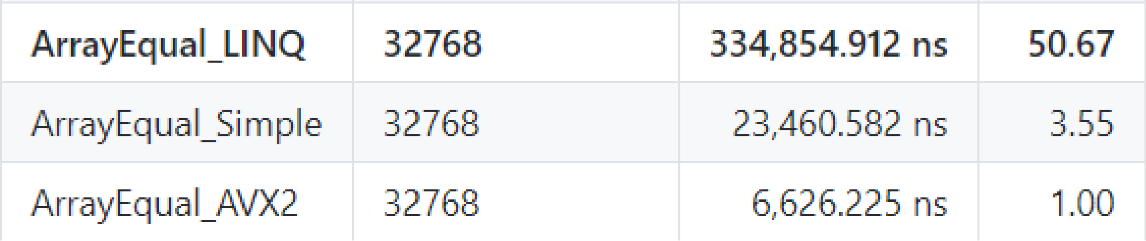
LINQ vs SIMD

```
int max = arrayOfInts.Max();
```



Max_LINQ	**32768**	**175,971.956 ns**	**84.17**
Max_Simple	32768	14,003.368 ns	6.70
Max_LinqFasterLib	32768	2,731.388 ns	1.31
Max_Avx	32768	2,096.625 ns	1.00

```
bool equal = Enumerable.SequenceEqual(arrayOfFloats1, arrayOfFloats2);
```

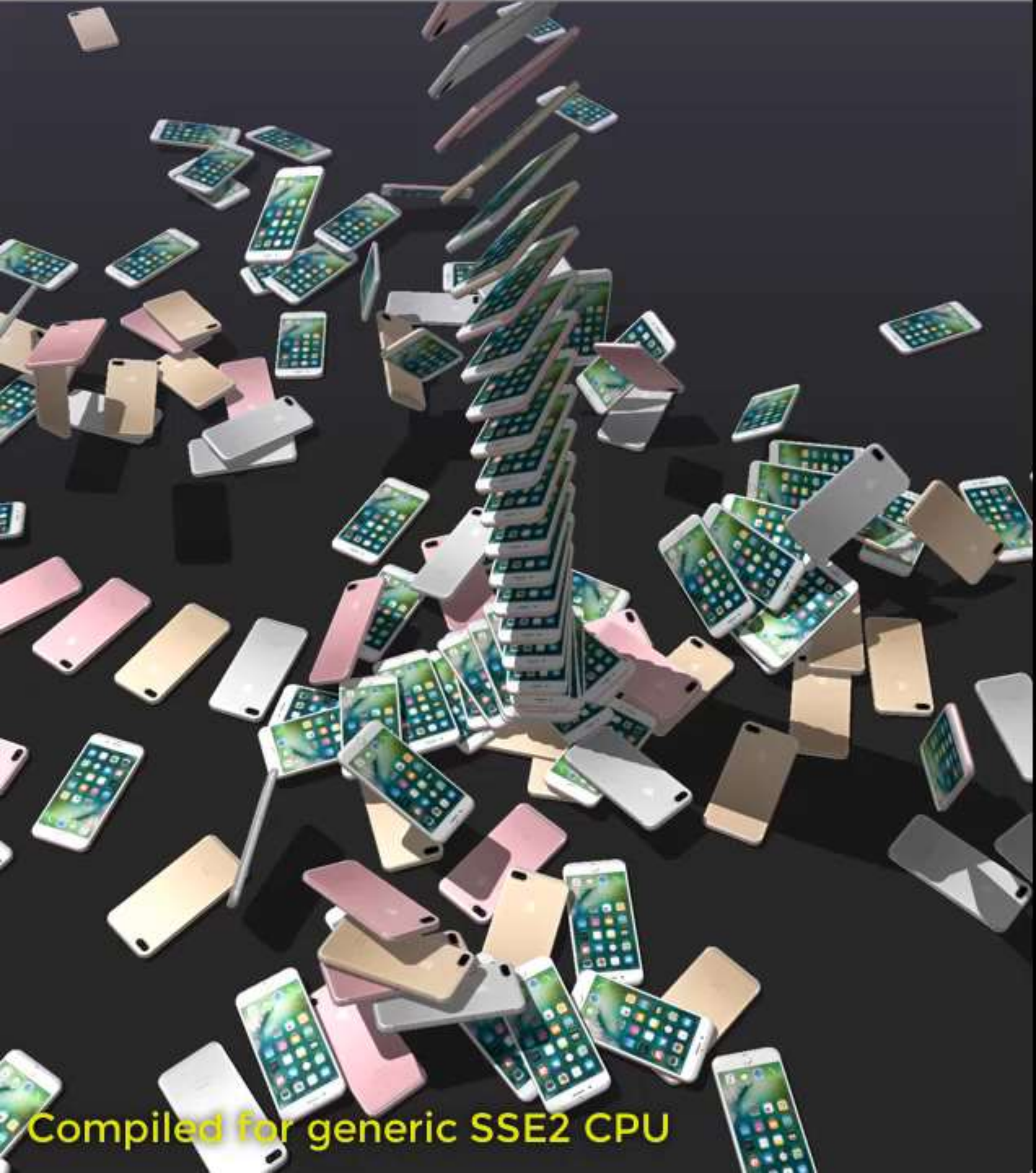


ArrayEqual_LINQ	**32768**	**334,854.912 ns**	**50.67**
ArrayEqual_Simple	32768	23,460.582 ns	3.55
ArrayEqual_AVX2	32768	6,626.225 ns	1.00

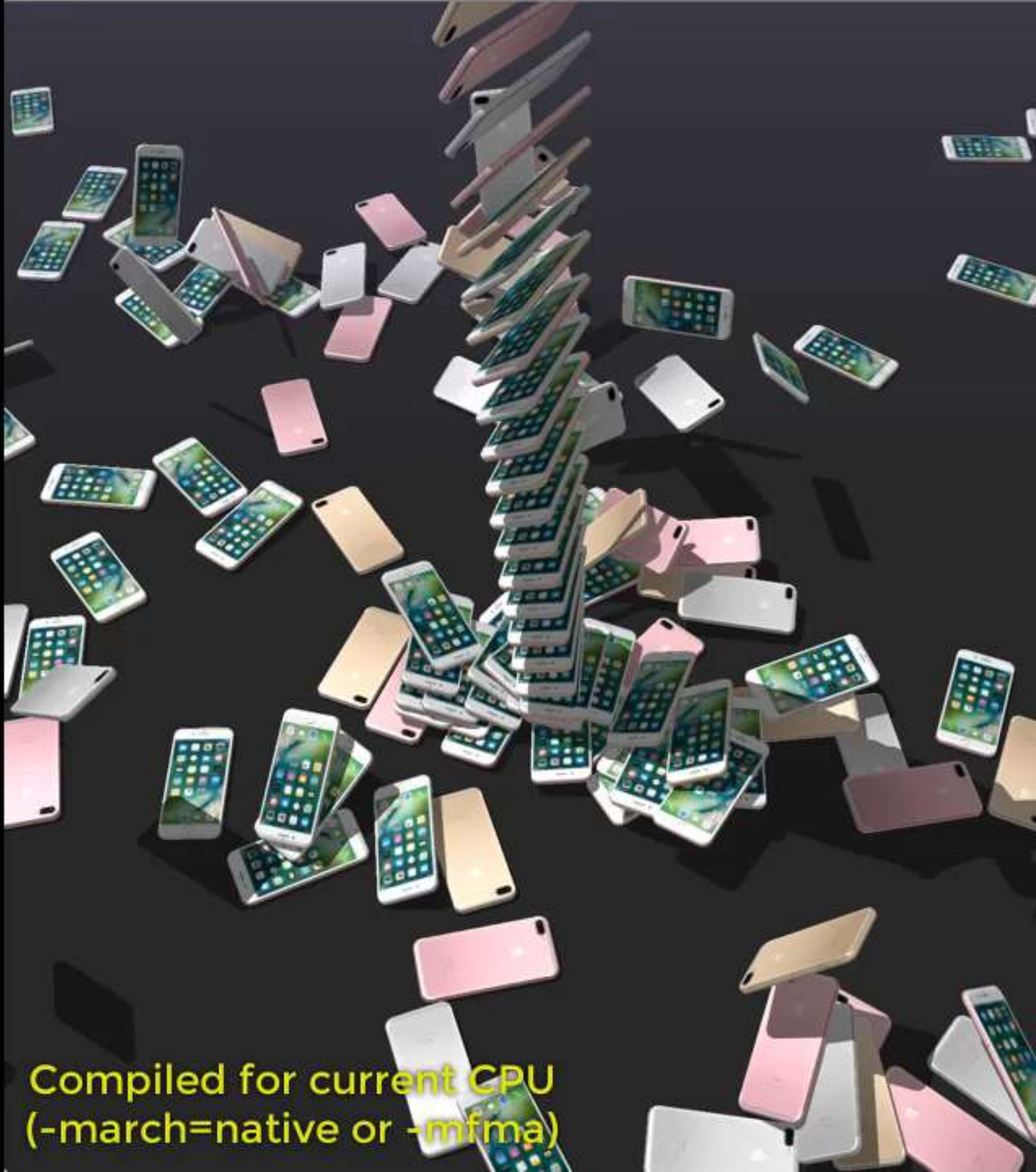
Be careful with floats and intrinsics

```
Fma.MultiplyAdd(x, y, z); // x*y+z
Sse3.HorizontalAdd(x, x);
```

```
a (39.33427f) * b (245.2255f) + c (150.424f) =
fmadd:      9796.190
fmul,fadd: 9796.189
```



Compiled for generic SSE2 CPU



Compiled for current CPU
(-march=native or -mfma)

61453.ToString("X"): "0xFOOD"

```
public static int CountHexDigits(ulong value)
{
    int digits = 1;
    if (value > 0xFFFFFFFF)
    {
        digits += 8;
        value >>= 0x20;
    }
    if (value > 0xFFFF)
    {
        digits += 4;
        value >>= 0x10;
    }
    if (value > 0xFF)
    {
        digits += 2;
        value >>= 0x8;
    }
    if (value > 0xF)
        digits++;

    return digits;
}
```

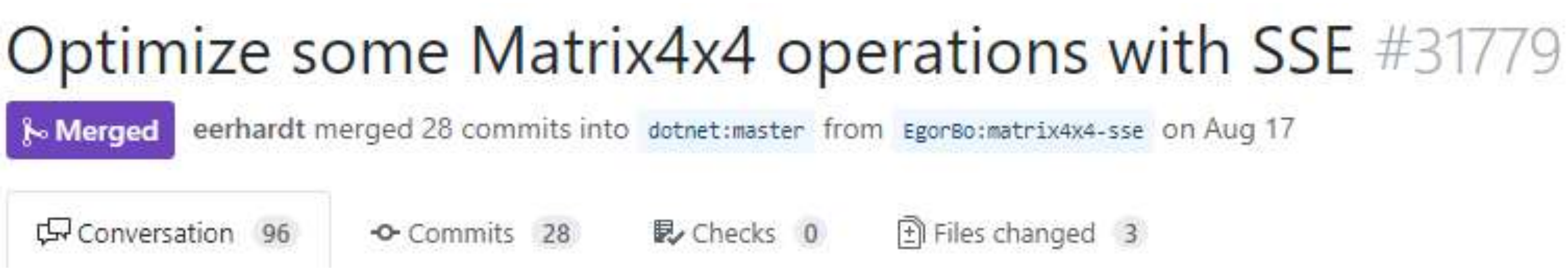
0xFOOD = 0000 0000 ... 0000 0000 1111 0000 0000 1101

```
Lzcnt.LeadingZeroCount(0xFOOD): 42
```

```
return (67-(int)Lzcnt.LeadingZeroCount(value | 1)) >> 2;
```

Optimize FormattingHelpers.CountHexDigits using Lzcnt.LeadingZeroCount #19006

Open EgorBo wants to merge 5 commits into dotnet:master from EgorBo:CountHexDigits-lzcnt



Optimize some Matrix4x4 operations with SSE #31779

Merged eerhardt merged 28 commits into dotnet:master from EgorBo:matrix4x4-sse on Aug 17

```
public static unsafe Matrix4x4 operator *(Matrix4x4 value1, Matrix4x4 value2)
{
    // OLD
    m.M11 = value1.M11 * value2.M11 + value1.M12 * value2.M21 + value1.M13 * value2.M31 + value1.M14 * value2.M41;
    m.M12 = value1.M11 * value2.M12 + value1.M12 * value2.M22 + value1.M13 * value2.M32 + value1.M14 * value2.M42;
    m.M13 = value1.M11 * value2.M13 + value1.M12 * value2.M23 + value1.M13 * value2.M33 + value1.M14 * value2.M43;
    m.M14 = value1.M11 * value2.M14 + value1.M12 * value2.M24 + value1.M13 * value2.M34 + value1.M14 * value2.M44;


    // NEW
    var row = Sse.LoadVector128(&value1.M11);
    Sse.Store(&value1.M11,
        Sse.Add(Sse.Add(Sse.Multiply(Sse.Shuffle(row, row, 0x00), Sse.LoadVector128(&value2.M11)),
                        Sse.Multiply(Sse.Shuffle(row, row, 0x55), Sse.LoadVector128(&value2.M21))),
                Sse.Add(Sse.Multiply(Sse.Shuffle(row, row, 0xAA), Sse.LoadVector128(&value2.M31)),
                        Sse.Multiply(Sse.Shuffle(row, row, 0xFF), Sse.LoadVector128(&value2.M41)))));
```

Matrix4x4.Add (Matrix4x4, Matrix4x4)

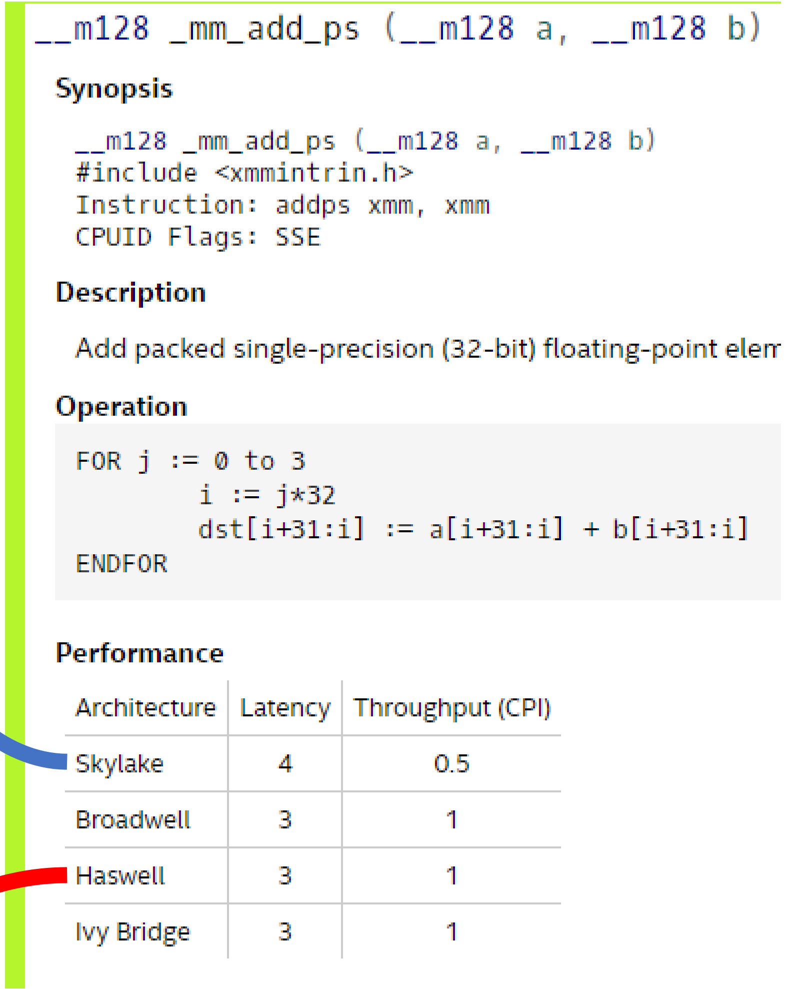
```
Matrix4x4 result = matrix1 + matrix2;
```

Windows (Coffee Lake):

Method	Mean	Scaled
Add_old	13.353 ns	1.00
Add_new	4.486 ns	0.34

macOS (Haswell):

Method	Mean	Scaled
Add_old	15.347 ns	1.00
Add_new	7.473 ns	0.49



```
__m128 _mm_add_ps (__m128 a, __m128 b)
```

Synopsis

```
__m128 _mm_add_ps (__m128 a, __m128 b)
#include <xmmintrin.h>
Instruction: addps xmm, xmm
CPUID Flags: SSE
```

Description

Add packed single-precision (32-bit) floating-point elem

Operation

```
FOR j := 0 to 3
        i := j*32
        dst[i+31:i] := a[i+31:i] + b[i+31:i]
ENDFOR
```

Performance

Architecture	Latency	Throughput (CPI)
Skylake	4	0.5
Broadwell	3	1
Haswell	3	1
Ivy Bridge	3	1

Better Matrix4x4 layout:

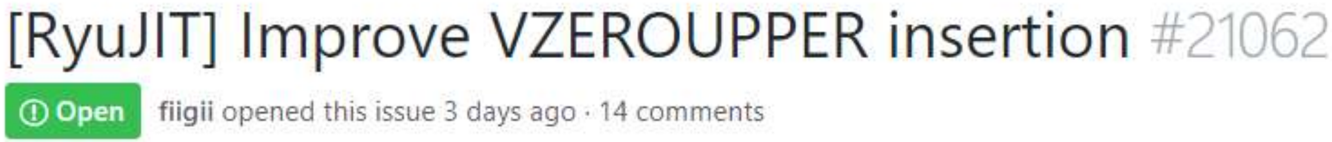
```
public struct Matrix4x4
{
    public float M11;
    public float M12;
    public float M13;
    //... 16 float fields
}
```

```
public struct Matrix4x4
{
    public Vector128<float> Row1;
    public Vector128<float> Row2;
    public Vector128<float> Row3;
    public Vector128<float> Row4;
}
```

AVX problems

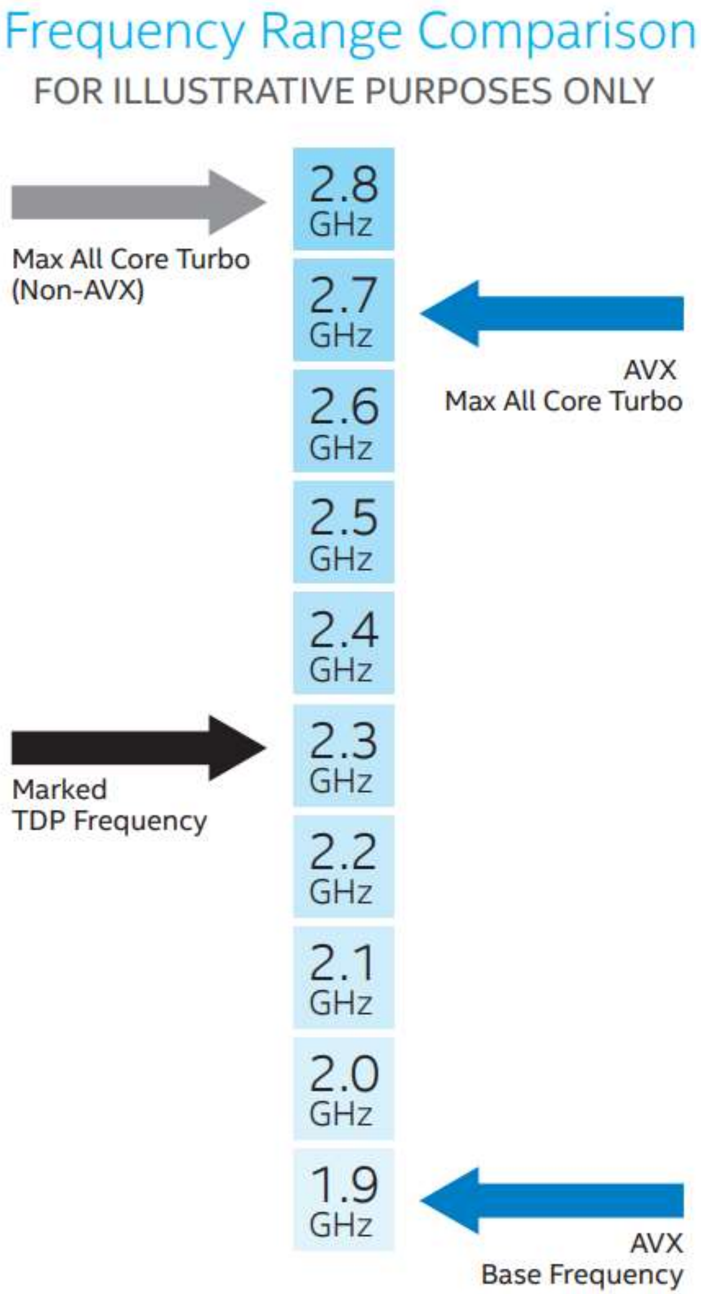
```
var v1 = Avx.LoadVector256(&m1.M11);
var v2 = Avx.LoadVector256(&m2.M11);
var v3 = Avx.Add(v1, v2);
```

```
SSE <-> AVX
```



[RyuJIT] Improve VZEROUPPER insertion #21062

ⓘ Open fiigii opened this issue 3 days ago · 14 comments



Frequency Range Comparison

FOR ILLUSTRATIVE PURPOSES ONLY

Alignment

```
// Prologue: iterate until data is aligned
for (…)

// Main loop: 100% optimized SIMD operations
for (…) LoadAlignedVector256(i)

// Epilogue: do regular `for` for the rest
for (…)
```

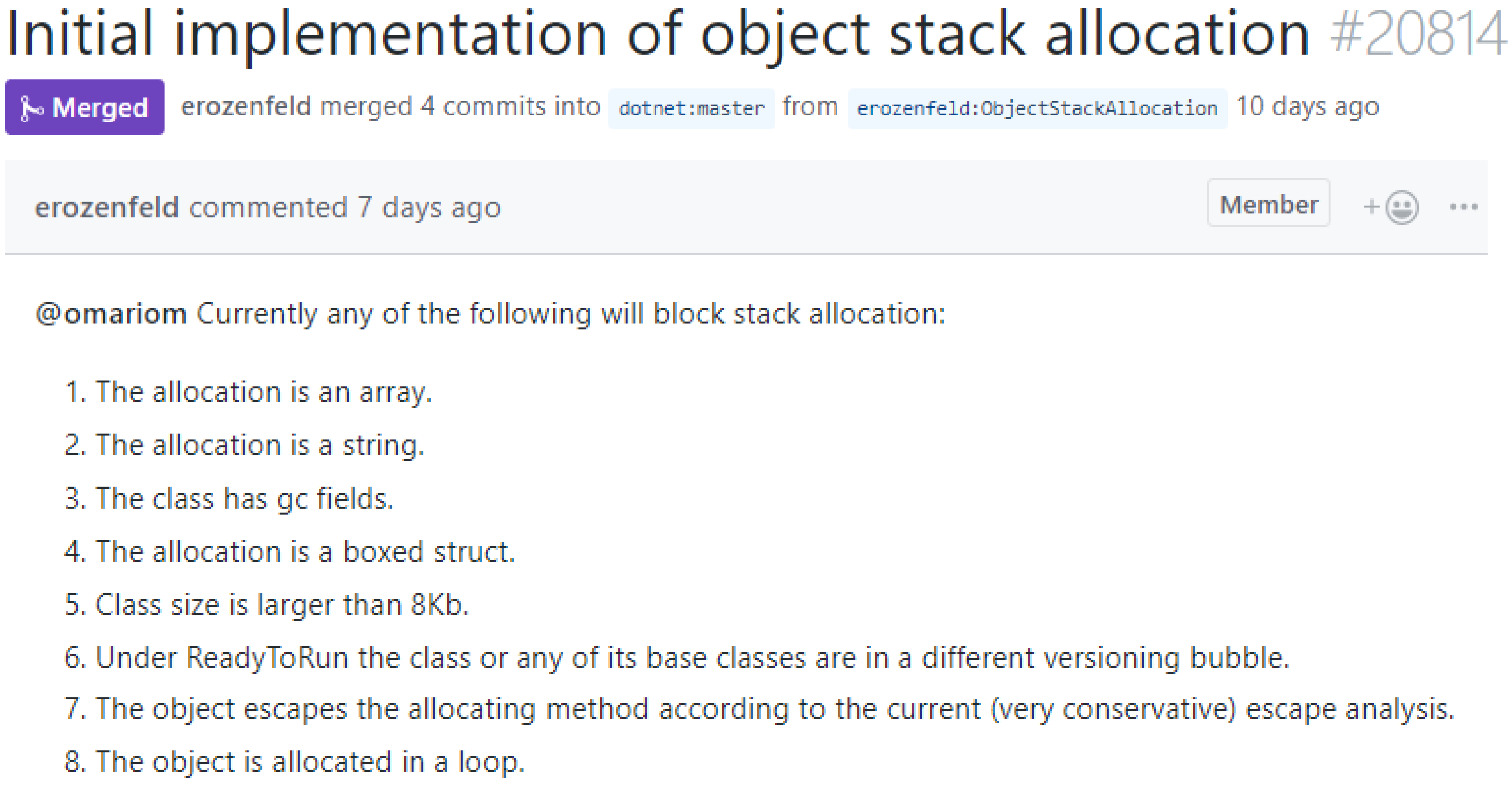
.NET Core: future

Objects on stack (escape analysis)

```
public string DoSomething()
{
    var builder = new StringBuilder();
    builder.Append(…);
    builder.Append(…);
    return builder.ToString();
    // builder never escapes the method
}
```

For Java folks: we have user-defined value-types ;-)

Objects on stack – merged!



Initial implementation of object stack allocation #20814

Merged **erozenfeld** merged 4 commits into `dotnet:master` from `erozenfeld:ObjectStackAllocation` 10 days ago

erozenfeld commented 7 days ago

@omariom Currently any of the following will block stack allocation:

1. The allocation is an array.
2. The allocation is a string.
3. The class has gc fields.
4. The allocation is a boxed struct.
5. Class size is larger than 8Kb.
6. Under ReadyToRun the class or any of its base classes are in a different versioning bubble.
7. The object escapes the allocating method according to the current (very conservative) escape analysis.
8. The object is allocated in a loop.

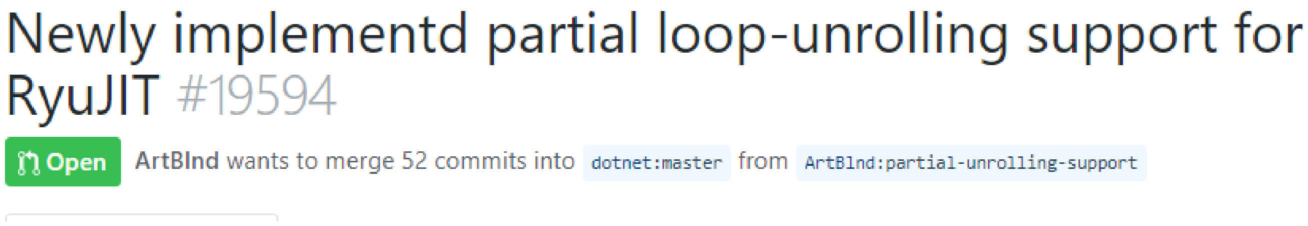
Tiered JIT Compilation – enabled by default

- **COMPlus_TieredCompilation=1**
- **COMPlus_TieredCompilation_Tier1CallCountThreshold=30**
- Cold methods with hot loops problem
- [MethodImpl(MethodImplOptions.AggressiveOptimization)]

Loop unrolling (auto-vectorization)

```
for (uint i = 0; i < 256; ++i)
{
    total += array[i];
}
```

```
for (uint i = 0; i < 64; ++i)
{
    total += array[i + 0];
    total += array[i + 1];
    total += array[i + 2];
    total += array[i + 3];
}
```

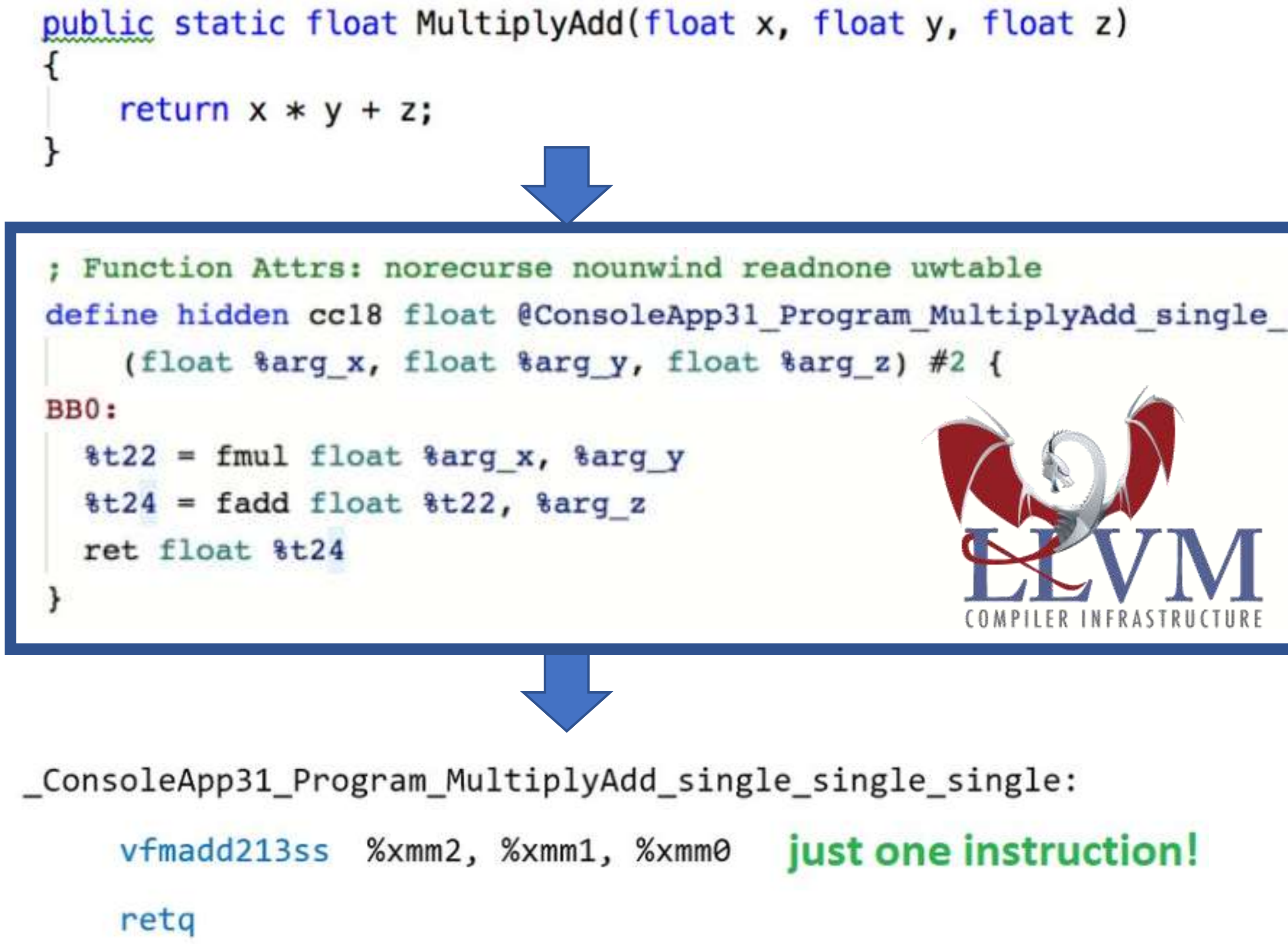


Newly implementd partial loop-unrolling support for RyuJIT #19594

Open ArtBlnd wants to merge 52 commits into `dotnet:master` from `ArtBlnd:partial-unrolling-support`

And don’t forget - C# has other backends!

- .NET 4.x CLR
- CoreRT
- Mono
    - JIT
    - AOT
    - LLVM (AOT/JIT)
    - Interpreter
- **IL2CPP**
- **Burst**



```
public static float MultiplyAdd(float x, float y, float z)
{
    return x * y + z;
}
```

```
; Function Attrs: norecurse nounwind readnone uwtable
define hidden cc18 float @ConsoleApp31_Program_MultiplyAdd_single_
    (float %arg_x, float %arg_y, float %arg_z) #2 {
BB0:
  %t22 = fmul float %arg_x, %arg_y
  %t24 = fadd float %t22, %arg_z
  ret float %t24
}
```

```
_ConsoleApp31_Program_MultiplyAdd_single_single_single:
    vfmadd213ss  %xmm2, %xmm1, %xmm0
    retq
```

just one instruction!

Micro-optimizations are for

- BCL and Runtime
  - Because you expect it to be fast
- Game Dev – 16ms per frame
  - Don’t be CPU-bound ☺
- High-load related libs and apps
- Image/Video processing, DL/ML frameworks
- Silly benchmarks (Go vs C#, Java vs C#)

Thanks!

Egor Bogatov

EgorBo