

Re: Why is C# 450% slower than C++ on nested loops ??

Re: Why is C# 450% slower than C++ on nested loops ??

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2005-12/msg05841.html>

- *From:* Stefan Simek <simek.nospam@xxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Sat, 31 Dec 2005 11:36:46 +0100
-

Peter Olcott wrote:

"Stefan Simek" <simek.nospam@xxxxxxxxxxxxxxxxxxxxx> wrote in message news:e2rqYVXDGHA.2912@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Peter Olcott wrote:

"Anders Borum" <anders@xxxxxxxxxxxxxxxxxxxxx>
wrote in message
news:edHCjaSDGHA.3528@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Hello!

It must have been
after the JIT,
because C# beat C++
on some things,
just not the nested
loops. The author
of the link would
know relatively
obvious things such
as this.

Unless somebody documents what

Re: Why is C# 450% slower than C++ on nested loops ??

methods have been used to measure the benchmarks, I am disposing the data altogether. Benchmarks should be verifiable ..

I searched high and low and finally found some stuff on unverified code. The MSDN documentation mentions it yet does not discuss what it means. It could be anything from security checks to tests for memory leaks. It doesn't say what its purpose is.

--
With regards
Anders Borum / SphereWorks
Microsoft Certified Professional (.NET MCP)

I guess you should google for 'define:verify'. Verifiable benchmarks have nothing to do with verifiable code. Anders just wanted the benchmarks to be verifiably *valid*, which they aren't by any means. A nested loop written the way it is in the benchmark is measuring nothing but a compiler's ability to optimize nested loops that do more or less nothing. Verifiable code means (in the .NET vocabulary) that the CLR can statically verify the code to ensure it will do nothing it is not expected to do. But enough of that, let's see what native assembly the

Yet I just found a Microsoft article that says that this sort of technology is between 20 and 50 years away. Code is not expected to be incorrect. Proving that code is not incorrect is technology that does not yet exist. I don't know what it is verifying, but, correct code is not it. The posted benchmark

Re: Why is C# 450% slower than C++ on nested loops ??

was crucial to my needs because my critical 100-line function is mostly loops.

That's why the code must meet certain criteria in order to be verifiable today. C# compiler generates such code, and the C++/CLI compiler is able to do so.

compilers generate for the loop and get over with it.

<...>

ha! I won't post the complete disassemblies (the C++ one is terribly cryptic so it would do no help), but I've found the following. The C++ compiler (8.0 in my case, but I suspect 6.0 is doing the same) manages to pre-cache the additions in the outer loops, which the C# compiler doesn't. Thus, in the C# code all the additions that happen at

```
x+=a+b+c+d+e+f;
```

are evaluated for every single innermost loop, while c++ does something like the following:

```
for (a = 0; a < n; a++)
{
    for (b = 0, tmp_ab = a; b < n; b++, tmp_ab++)
    {
        for (c = 0, tmp_abc = tmp_ab; c < n; c++, tmp_abc++)
        {
            for (d = 0, tmp_abcd = tmp_abc; d < n; d++, tmp_abcd++)
            {
                for (e = 0, tmp_abcde = tmp_abcd; e < n; e++, tmp_abcde++)
                {
                    for (f = 0, tmp = tmp_abcde; f < n; f++, tmp++)
                        x += tmp;
                }
            }
        }
    }
}
```

Re: Why is C# 450% slower than C++ on nested loops ??

If you compile this code in C#, the execution time is 6734 ms for .NET 2.0 and 8593 ms for .NET 1.1 versus 3656 ms for unmanaged C++ (8.0) on my machine. But note that the innermost loop in the C++ is only four instructions. This can hardly be matched to any real-life algorithm, and I can assure you that if there was anything more than 'nothing' in the inner loop, the performance difference would be much less than 80% (I expect it to drop far below 10% for most real-life algorithms).

Since this is not the same code that was posted on the benchmark, it is not a valid test. If you are worried about arithmetic overflow then define X as a double. I want to know why there was a 450% difference in the original benchmark. When you change the code, this avoids rather than answers the question. It is reported that 2005 does a much better job of optimization of .NET code, yet, only with the C++, not the C# compiler.

No. I gave you an exact answer **why** there is the difference. The C++ translates the original code logic to something comparable to the snippet I have posted. C# compiler fails to do this optimization. But it has nothing to do with nested loops per se. It is an optimization of the addition performed in the innermost loop.

That said, I wouldn't expect C# (or .NET) to match pure native code performance for purely computational tasks like the one you describe is. C# can win easily in cases where a lot of dynamic allocation is involved, etc., but it will probably never outperform optimized native C++ (or handwritten assembly) for computational tasks. If you need to squeeze every clock of your CPU, you will probably get best results using an optimizing compiler targeted for exactly the CPU the code will run on.

Re: Why is C# 450% slower than C++ on nested loops ??

There is no reason (in theory) why .NET code would need to be any slower than native code. The reason in practice seems to be that Microsoft just hasn't gotten around to implementing every possible optimization in every language, yet. The one tricky thing about optimization in .NET is that it is divided into two different processes. Some of the optimizations at the compile step can screw up some of the optimizations at the JIT step. The problem is that the compile step does not know how many registers the target architecture has.

That's why I used the word **probably**. The JIT is much more time constrained, so it **might** never be able to match the optimizations of traditional compilers. It **should** certainly give better **average** results across platforms in the future, because it makes use of any available hardware present.

I didn't want to write all this, but after reading through the long discussions on this matter without touching the important points, I just decided to do so :)

Just my 2c.

Stefan

If you really want to get answers to your questions, not just troll around, you really can get them here. If you don't...

Stefan

.