Which is faster - C# unsafe code or raw C++

Asked 16 years, 1 month ago Modified 10 years ago Viewed 20k times

26

I'm writing an image processing program to perform real time processing of video frames. It's in C# using the Emgu.CV library (C#) that wraps the OpenCV library dll (unmanaged C++). Now I have to write my own special algorithm and it needs to be as fast as possible.

Which will be a faster implementation of the algorithm?

1. Writing an 'unsafe' function in C#
2. Adding the function to the OpenCV library and calling it through Emgu.CV

I'm guessing C# unsafe is slower because it goes throught the JIT compiler, but would the difference be significant?

Edit:

Compiled for .NET 3.5 under VS2008

c# c++ performance image-processing unsafe

edited Dec 15, 2014 at 7:44

Share

Improve this question

Follow

Mystical
471k ● 46 ● 339 ● 336

asked Nov 11, 2008 at 11:05

geometrikal
3,294 ● 2 ● 33 ● 45

10 Answers

Sorted by: Highest score (default)

72

it needs to be as fast as possible

Then you're asking the wrong question.

Code it in assembler, with different versions for each significant architecture variant you support.

Use as a guide the output from a good C++ compiler with optimisation, because it probably knows some tricks that you don't. But you'll probably be able to think of some improvements, because C++ doesn't necessarily convey to the compiler all information that might be useful for optimisation. For example, C++ doesn't have the C99 keyword restrict. Although in that particular case many C++ compilers (including MSVC) do now support it, so use it where possible.

Of course if you mean, "I want it to be fast, but not to the extent of going outside C# or C++", then the answer's different ;-)

I would expect C# to at least approach the performance of similar-looking C++ in a lot of cases. I assume of course that the program will be running long enough that the time the JIT itself takes is irrelevant, but if you're processing much video then that seems likely. But I'd also expect there to be certain things which if you do them in unsafe C#, will be far slower than the equivalent thing in C++. I don't know what they are, because all my experience of JITs is in Java rather than CLR. There might also be things which are slower in C++, for instance if your algorithm makes any calls back into C# code.

Unfortunately the only way to be sure how close it is is to write both and test them, which kind of misses the point that writing the C++ version is a bunch of extra effort. However, you might be able to get a rough idea by hacking some quick code which approximates the processing you want to do, without necessarily doing all of it or getting it right. If you algorithm is going to loop over all the pixels and do a few FP ops per pixel, then hacking together a rough benchmark should take all of half an hour.

Usually I would advise against starting out thinking "this needs to be as fast as possible". Requirements should be achievable, and by definition "as X as possible" is only borderline achievable. Requirements should also be testable, and "as X as possible" isn't testable unless you somehow know a theoretical maximum. A more friendly requirement is "this needs to process video frames of such-and-such resolution in real time on such-and-such a

speed CPU", or "this needs to be faster than our main competitor's product". If the C# version does that, with a bit to spare to account for unexpected minor issues in the user's setup, then job done.

Share Improve this answer Follow

edited Oct 12, 2009 at 13:34

answered Nov 11, 2008 at 11:36

Steve Jessop
279k ● 40 ● 469 ● 709

3 That was a very well-written, carefully thought out and informative post. Thank you very much. – endian Nov 11, 2008 at 13:01

9 Don't do it in assember - unless you're a genius, VS 2008 is going to do a better job at optimizing than you will. Use intrinsics to indicate where the compiler should use special operations (SSE, SSE2, etc) and compile separately for each target platform. – Eclipse Nov 11, 2008 at 16:12

4 Video codecs and media streaming are areas where someone who knows the CPU can beat C compilers. Or at least, that's what they think, and I'm not going to tell them they're wrong. Maybe my answer doesn't make clear that I don't think it's a good option for J. Random C programmer to try it casually. – Steve Jessop Nov 11, 2008 at 21:52

2 ... if only because the time taken to get into assembler programming if you aren't already is a significant investment without a guarantee of results. I don't think you need to be a genius, though, just persistent. – Steve Jessop Nov 11, 2008 at 21:53

15 When i worked with embedded hardware, a common task was to build it in C (it was a C/C++ shop), build it, then if a piece needed further optimization, we disassembled it and took the compiled asm and used that as a baseline for the 'new' assembly version of the function. Sometimes we could improve on it - sometimes we couldn't. But its a great baseline to start with. – cyberconte May 12, 2009 at 14:48

6

It depends on the algorithm, the implementation, the C++ compiler and the JIT compiler. I guess in most cases the C++ implementation will be faster. But this may change.

The JIT compiler can optimize your code for the platform your code is running on instead of an average for all the platforms your code might run on as the C++ compiler does. This is something newer versions of the JIT compiler are increasingly good at and may in some cases give JITted code an advantage. So the answer is not as clear as you might expect. The new Java hotspot compiler does this very well for example.

Other situations where managed code may do better than C++ is where you need to allocate and deallocate lots of small objects. The .net runtime preallocates large chunks of memory that can be reused so it doesn't need to call into the os every time you need to allocate memory.

I'm not sure unsafe C# runs much faster than normal C#. You'll have to try this too.

If you want to know what's the best solution for your situation you'll have to try both and measure the

difference. I dont think there will be more than

Share Improve this answer

Follow

answered Nov 11, 2008 at 11:20

Mendelt

37.5k ● 6 ● 75 ● 97

<blockquote>I'm not sure unsafe C# runs much faster than normal C#. You'll have to try this too.</blockquote> Try to rotate an image, first in c# and then using unsafe c# on a Arm4vi it takes 15 minutes against 4 seconds ;)
– kentaromiura Nov 11, 2008 at 12:20

@kentaromiura: Sounds like something else is going wrong in your safe code example. Something that can be done in 4 seconds should never take 15 minutes in managed code.
– Mendelt Nov 12, 2008 at 9:27

In video processing you are not allocating and deallocating lots of small objects. That's the way to miss frames.
– Stephan Eggermont Nov 28, 2008 at 21:36

@Stephan Eggermont: Yeah. That was probably the wrong example for this question. @kentaromiura: By the way. You were probably using the .net micro framework right? This doesn't include a JIT compiler to save memory. The bytecode interpreter is very slow. C# vs. C++ on the normal framework is closer. – Mendelt Dec 1, 2008 at 8:32

6

Languages don't have a "speed". It depends on the compiler and the code. It's possible to write inefficient code in any language, and a clever compiler will generate near-optimal code no matter the language of the source.

The only really unavoidable factor in performance between C# and C++ is that C# apps have to do more at startup (load the .NET framework and perhaps JIT some code), so all things being equal, they will launch a bit slower. After that, it depends, and there's no fundamental reason why one language must always be faster than another.

I'm also not aware of any reasons why unsafe C# should be faster than safe. In general, safe is good because it allows the compiler to make some much stronger assumptions, and so safe *might* be faster. But again, it depends on the code you're compiling, the compiler you're using and a dozen other factors.

In short, give up on the idea that you can measure the performance of a language. You can't. A language is never "fast" or slow". It doesn't have a speed.

Share Improve this answer

Follow

answered Nov 11, 2008 at 11:26

Stack Overflow is garbage

247k ● 52 ● 353 ● 555

What you can (and must in these cases) do is to measure the performance of a particular algorithm/compiler/machine triplet – Vinko Vrsalovic Nov 11, 2008 at 11:32

Actually, languages do have characteristics that allow or dissallow the compilers to make optimizations. A "perfect" Fortran compiler will always beat a "perfect" C# compiler. – Nemanja Trifunovic Nov 11, 2008 at 14:51

onebyone.livejournal.com: Yes, but that only eliminates two of the three variables. It still depends on the particular code as well. Nemanja Trifunovic: I'm not aware of any optimizations that C# forbids. The general compiler rule is that the code must work "as if" the language spec was followed.
– Stack Overflow is garbage Nov 11, 2008 at 19:03

In a perfect world, the JIT should be able to make your safe code as fast as your unsafe code. But the reality is that some algorithms run a lot faster when you write them in terms of pointers for the compiler. – Eloff Dec 6, 2009 at 11:53

@Eloff: Certainly. But the reality is also that **other** algorithms run faster when you write them *without* using pointers, as the compiler dodges all the nasty, performance-crippling aliasing. Or why do you think Fortran is used instead of C/C++ for high-performance scientific computing? ;)
– Stack Overflow is garbage Dec 6, 2009 at 16:35

6

C# is typically slower than C++. There are runtime checks in managed code. These are what make it managed, after all. C++ doesn't have to check whether the bounds of an array have been exceeded for example.

From my experience, using fixed memory helps a lot. There is a new System.IO.UnmanagedMemoryAccessor class in .NET 4.0 which may help in the future.

Share Improve this answer

Follow

answered Nov 11, 2008 at 12:43

Thomas Bratt

51.6k ● 36 ● 126 ● 142

Using fixed memory gets around the runtime checks and enables writing similar code to C++ for better and for worst.

– Danny Varod Oct 24, 2009 at 13:27

Using fixed memory does help but does not remove all of the runtime checks. – Thomas Bratt Oct 27, 2009 at 10:57

3 In many cases the JIT can remove the array bounds checking on every access, if it can determine you will not exceed those bounds (like a for loop where the condition is i < array.Length.) Even the Mono JIT does this optimization. – Eloff Dec 6, 2009 at 11:57

4

If you are going to implement your algorithm in a standard way I think it's irrelevant. But some languages have bindings to apis or libraries that can give you a non standart boost.

1. Consider if you can use GPU processing - nvidia and ati provide the CUDA and CTM frameworks and there is an ongoing standarization effort from the khronos group (openGL). A hunch tells me also that amd will add at least one streaming processor core in their future chips. So I think there is quite a promise in that area.
2. Try to see if you can exploit SSE instructions, there are libraries around -most in C++ or C- that provide handy apis, check Intel's site for handy optimized libraries I do recall "Intel Performance Primitives" and a "Math Kernel".

But on the politics side, do incorporate your algorithm in OpenCV so others may benefit too.

Share Improve this answer

Follow

answered Nov 11, 2008 at 13:45

thAAAnos

156 • 3

I used IPP back when they were free - really nice, although the method names were quite a mouthful. Apparently OpenCV can make use of IPP if you have it. – geometrikal Nov 11, 2008 at 23:43

4

It's a battle that will rage on forever. C versus C++ versus C# versus whatever. In C#, the notion of unsafe is to unlock "dangerous" operations. ie, the use of pointers, and being able to cast to void pointers etc, as you can in C and C++. Very dangerous, and very powerful! But defeating what C# was based upon.

You'll find that nowadays, Microsoft has made strides in the direction of performance, especially since the release of .NET, and the next version of .NET will actually support inline methods, as you can with C++. This will increase performance for very specific situations. I hate that it's not going to be a c# feature, but a nasty attribute the compiler picks up on - but you can't have it all.

Personally, I'm writing a game with C# and managed DirectX (why not XNA?? beyond the scope of this post). I'm using unsafe code in graphical situations, which brings about a nod in the direction of what others have said.

It's only because pixel access is rediculously slow with GDI++ that I was driven to look for alternatives. But on the whole, the c# compiler is pretty damned good, and for code comparisons (you can find articles) you'll find the performance is very comparable to c++. That's not to say there isn't a better way to write the code.

At the end of the day, I personally see C, C++, and C# as about the same speed when executing. It's just that in some painful situations where you want to work really closely with the underlying hardware or very close to those pixels, that you'll find noticeable advantage to the C/C++ crowd.

But for business, and most things nowadays, C# is a real contender, and staying within the "safe" environment is definitely a bonus.
When stepping outside, you can get most things done with unsafe code, as I have - and boy, have I gone to some extremes! But was it worth it? Probably not. I personally wonder if I should have thought more along the lines of time-critical code in C++, and all the Object Oriented safe stuff in C#. But I have better performance than I thought I'd get!

So long as you're careful with the amount of interop calls you're making, you can get the best of both worlds. I've personally avoided that, but I don't know to what cost.

So an approach I've not tried, but would love to hear adventures in, in actually using C++.NET to develop a library in - would that be any faster than c#'s unsafe for

these special graphical situations? How would that compare to native C++ compiled code? Now there's a question!

Hmm..

Share Improve this answer

Follow

answered Jul 18, 2013 at 13:30

Simon Miller

77 ● 2

3

If you know your environment and you use a good compiler (for video processing on windows, Intel C++ Compiler is probably the best choice), C++ will beat C# hands-down for several reasons:

- The C++ runtime environment has no intrinsic runtime checks (the downside being that you have free reign to blow yourself up). The C# runtime environment is going to have some sanity checking going on, at least initially.
- C++ compilers are built for optimizing code. While it's theoretically possible to implement a C# JIT compiler using all of the optimizing voodo that ICC (or GCC) uses, it's doubtful that Microsoft's JIT will reliably do better. Even if the JIT compiler has runtime statistics, that's still not as good as profile-guided optimization in either ICC or GCC.
- A C++ environment allows you to control your memory model much better. If your application gets to the point of thrashing the data cache or

fragmenting the heap, you'll really appreciate the extra control over allocation. Heck, if you can avoid dynamic allocations, you're already much better off (hint: the running time of `malloc()` or any other dynamic allocator is nondeterministic, and almost all non-native languages force heavier heap usage, and thus heavier allocation).

If you use a poor compiler, or if you can't target a good chipset, **all bets are off**.

Share Improve this answer

Follow

answered Nov 26, 2008 at 4:11

Tom Barta

1,284 ● 7 ● 3

1 Allocation in the CLR is O(1). Next pointer go. The expense is in collection, but using exactly the same principles as C++ optimisation you can do things quite smartly. You will miss pointers, which give you the performance though. Evil evil pointers... – Spence Feb 23, 2009 at 4:57

The C++ STL has many checks in `_DEBUG` mode now. So it's a good idea to run in a debug version while testing before running a release with full optimizations. Now, I've worked a bit with Ada and they have a similar spec. about things such as overflow of an array index. In many cases, though, that test can be 100% optimized out, even in a debug version. I would imagine that C# can do the same. – Alexis Wilke Jan 10, 2022 at 0:58

To be honest, what language you write it in is not nearly as important as what algorithms you use (IMO, anyway). Maybe by going to native code you *might* make your

2

application faster, but it might also make it slower--it'd depend on the compiler, how the programs are written, what sort of interop costs you'd incur if you're using a mixed environment, etc. You can't really say without profiling it. (and, for that matter, *have you profiled your application? Do you actually know where it's spending time?*)

A better algorithm is completely independent of the language you choose.

Share Improve this answer

Follow

answered Jun 10, 2009 at 0:40

user25967

371 ● 2 ● 3

1

I'm a little late in responding but I can give you some anecdotal experience. We had some matrix multiplication routines that were originally coded in C# using pointers and unsafe code. This proved to be a bottleneck in our application and we then used pinning+P/Invoke to call into a C++ version of the Matrix multiplication routine and got a factor of 2 improvement. This was a while ago with .NET 1.1, so things might be better now. As others point out, this *proves* nothing, but it was an interesting exercise.

I also agree with thAAAnos, if you algorithm really has to be "as fast as possible" leverage IPL or, if you must, consider a GPU implementation.

Share Improve this answer

Follow

answered Nov 28, 2008 at 20:56

Peter Tate

2,158 ● 1 ● 21 ● 33

-8

Running on the CPU is always going to be faster than running on a VM on the CPU. I can't believe people are trying to argue otherwise.

For example, we have some fairly heavy image processing work on our web server that's queued up. Initially to get it working, we used PHP's GD functions.

They were slow as hell. We rewrote the functionality we needed in C++.

Share Improve this answer

Follow

answered Nov 11, 2008 at 11:37

Dave

155 ● 2 ● 5

C# doesn't run on a VM either. – pipTheGeek Nov 11, 2008 at 12:39

JIT compilers are also called VMs. Although I also consider this answer unhelpful, VM *is* an accepted synonym here. – Lilith River Jul 18, 2012 at 19:43