In .Net, when if ever should I pass structs by reference for performance reasons?

Asked 15 years, 11 months ago Modified 15 years, 5 months ago

Viewed 5k times

19

In my C# application, I have a large struct (176 bytes) that is passed potentially a hundred thousand times per second to a function. This function then simply takes a pointer to the struct and passes the pointer to unmanaged code. Neither the function nor the unmanaged code will make any modifications to the struct.

My question is, should I pass the struct to the function by value or by reference? In this particular case, my guess is that passing by reference would be much faster than pushing 176 bytes onto the call stack, unless the JIT happens to recognize that the struct is never modified (my guess is it can't recognize this since the struct's address is passed to unmanaged code) and optimizes the code.

Since we're at it, let's also answer the more general case where the function does *not* pass the struct's pointer to unmanaged code, but instead performs some read-only operation on the contents of the struct. Would it be faster to pass the struct by reference? Would in this case the JIT recognize that the struct is never modified and thus optimize? Presumably it is *not* more efficient to pass a 1-byte struct by reference, but at what struct size does it become better to pass a struct by reference, if ever?

Thanks.

EDIT:

As pointed out below, it's also possible to create an "equivalent" class for regular use, and then use a struct when passing to unmanaged code. I see two options here:

1) Create a "wrapper" class that simply contains the struct, and then pin and pass a pointer to the struct to the unmanaged code when necessary. A potential issue I see is that pinning has its own performance consequences.

2) Create an equivalent class whose fields are copied to the struct when the struct is needed. But copying would take a lot of time and seems to me to defeat the point of passing by reference in the first place.

EDIT:

As mentioned a couple times below, I could certainly just measure the performance of each of these methods. I *will* do this and post the results. However, I am still interested in seeing people's answers and reasonings from an intellectual perspective.

c# .net optimization

Share

Improve this question

Follow

edited Jan 26, 2009 at 22:19

asked Jan 26, 2009 at 20:58

Walt D

4,681 ● 8 ● 36 ● 46

1 Interesting question. Of course the obvious response is that everyone's situation is different, you shouldn't optimize prematurely, and you should profile your application if you're going to try to boost performance. That said, I'm curious about some real answers. – George Mauer Jan 26, 2009 at 21:05

That's a rather large struct, why not convert it to a class? If you're gonna be passing it by ref, it defeats the purpose of making it a struct anyway. – Ricardo Villamil Jan 26, 2009 at 21:11

A pointer to this struct is being passed directly to a hardware device driver, so yes, it does need to be a struct. Keeping it as a struct and simply getting a pointer to it is, I assume, far faster than marshaling an equivalent object. – Walt D Jan 26, 2009 at 21:20

If it's not being modified, I'd have to expect that a single pinned instance of the object would be much faster. Not to mention the badness that structs inflict on the x64 JITter. – Greg D Jan 26, 2009 at 21:37

4 Answers

Sorted by: Highest score (default)

13

I did some very informal profiling, and the results indicate that, for my particular application, there is a modest performance gain for passing by reference. For by-value I got about 10,050,000 calls per second, whereas for by-reference I got about 11,200,000 calls per second.

Your mileage may vary.

Share Improve this answer

Follow

answered Jan 27, 2009 at 1:09

Walt D

4,681 ● 8 ● 36 ● 46

1 What was the size of your structs? – Timo Oct 31, 2016 at 15:20

176 bytes each. (They get passed by-pointer to the graphics driver, which is why they need to be structs and not classes.) – Walt D Nov 4, 2016 at 21:40

9

Before you ask whether or not you should pass the struct by reference, you should ask yourself why you've got such an enormous struct in the first place. Does it *really* need to be a struct? If you need to use a struct at some point for P/Invoke, would it be worth having a struct *just* for that, and then the equivalent class elsewhere?

A struct that big is very, very unusual...

See the Design Guidelines for Developing Class Libraries section on Choosing Between Classes and Structures for more guidance on this.

Share Improve this answer Follow

answered Jan 26, 2009 at 21:10

Jon Skeet
1.5m ● 889 ● 9.3k ● 9.3k

A struct period is quite unusual – George Mauer Jan 26, 2009 at 21:12

4 This particular struct is being passed directly to a hardware device driver, so yes, in my case it *does* need to be a struct. It also now occurs to me that I could wrap the struct in a class and then pin the struct when I need the pointer, but pinning may introduce other performance issues. – Walt D Jan 26, 2009 at 21:18

1 @Justice: In that case, it should either be just for P/Invoke, or should be redesigned. A "very large" struct is a big design smell. – Jon Skeet Jul 9, 2009 at 19:55

1 I know changing every old comment on the topic is probably not feasible, but with proper usage of C# 7.2's `in`, would you consider the smell to have dissipated? Should the design guidelines be rewritten? – fuglede Mar 4, 2018 at 16:56

1 @JonSkeet: Thanks for elaborating. This is probably sufficient material for a new question, but if one is careful about never boxing and always passing (and returning and assigning) by reference, is there any downside to having large (immutable) structs at all? – fuglede Mar 5, 2018 at 7:44

2

The only way you can get an answer to this question is to code up both and measure the performance.

You mention unmanaged/managed interop. My experience is that it takes a surprisingly long time to to the interop. You could try changing your code from:

```
void ManagedMethod(MyStruct[] items) {
  foreach (var item in items) {
    unmanagedHandle.ProcessOne(item);
  }
}
```

To:

```
void ManagedMethod(MyStruct[] items) {
  unmanagedHandle.ProcessMany(items, items.Count);
}
```

This technique helped me in a similar case, but only measurements will tell you if it works for your case.

Share Improve this answer Follow

answered Jan 26, 2009 at 22:14

Hallgrim
15.5k ● 11 ● 48 ● 54

I think you have misunderstood my particular scenario. I am only passing a single item, not an array of items, albeit many many times per second. – Walt D Jan 26, 2009 at 22:17

-2

Why not just use a class instead, and pass your class to your P/Invoke function?

Using class will pass nicely around in your managed code, and will work the same as passing a struct by reference to a P/Invoke function.

e.g.

```
// What you have
public struct X
{
   public int data;
}
[DllImport("mylib.dll")]
static extern void Foo( ref X arg);

// What you could do
[StructLayout(LayoutKind.Sequential)]
public class Y
{
    public int data;
}
[DllImport("mylib.dll")]
static extern void Bar( Y arg );
```

Share Improve this answer Follow

edited Jul 9, 2009 at 19:17

answered Jul 9, 2009 at 19:04

Wil S
345 ● 1 ● 2 ● 12

Why the downvote. Is this wrong? Can we help the contributor to fix the answer? – speciesUnknown Mar 26, 2018 at 19:14