C++ Smart Pointer performance

Asked 16 years, 1 month ago Modified 14 years, 3 months ago

Viewed 8k times

18

How much do using smart pointers, particularly boost::shared_ptr cost more compared to bare pointers in terms of time and memory? Is using bare pointers better for performance intensive parts of gaming/embedded systems? Would you recommend using bare pointers or smart pointers for performance intensive components?

c++ boost smart-pointers

Share

Improve this question

Follow

asked Nov 21, 2008 at 11:24

amit kumar

21k • 24 • 93 • 128

6 Answers

Sorted by: Highest score (default)

21

Dereferencing smart pointers is typically trivial, certainly for boost in release mode. All boost checks are at compile-time. (Smart pointers could in theory do smart stuff across threads). This still leaves a lot of other operations. Nicola mentioned construction, copying and destruction. This is not the complete set, though. Other important operations are swapping, assignment and

resetting to NULL. Basically, any operation that requires smartness.

Note that some of these operations are excluded by some smart pointers. E.g. `boost::scoped_ptr` cannot even be copied, let alone be assigned. As this leaves less operations, the implementation can be optimized for these fewer methods.

In fact, with TR1 coming up, it's quite likely that compilers can do better with smart pointers than raw pointers. E.g. it's possible that a compiler can prove that a smart non-copyable pointer is not aliased in some situations, merely because it's non-copyable. Think about it: aliasing occurs when two pointers are created pointing to the same object. If the first pointer cannot be copied, how would a second pointer end up pointing to the same object? (There are ways around that, too - operator* has to return an lvalue)

Share Improve this answer Follow

edited Sep 7, 2010 at 8:23

answered Nov 21, 2008 at 12:54

MSalters

179k ● 11 ● 164 ● 368

Unfortunately, your idea for optimised smart pointers won't work, at least in C++. You might have stored the pointer elsewhere before putting it into the smart pointer. Further, it's easy (but not advised) to get the raw C pointer back out of a

smart pointer, by doing &*smart_ptr; – Chris Jefferson Nov 21, 2008 at 14:29

i agree with chris jefferson. nobody stops you from storing it elsewhere before putting it into the smart pointer – Johannes Schaub - litb Nov 28, 2008 at 17:31

I've written a little answer about aliasing here: stackoverflow.com/questions/270408/… – Johannes Schaub - litb Nov 28, 2008 at 17:32

You could indeed have initialized the scoped_ptr from a raw pointer, in which case the compiler could not optimize. My point was that this is the exception. The compiler would need to exclude these uncommon cases when optimizing, just like it needs to exclude the &*MySmartPtr cases. – MSalters Dec 2, 2008 at 15:20

7

Boost provide different smart pointers. Generally both the memory occupation, which varies in accordance to the kind of smart pointer, and the performance should not be an issue. For a performance comparison you can check this http://www.boost.org/doc/libs/1_37_0/libs/smart_ptr/smart tests.htm.

As you can see only construction, copy and destruction are taken into account for the performance comparison, which means that dereferencing a smart pointer has supposedly the same cost as that of a raw pointer.

The following snippet demonstrates that there's no performance loss by using a `shared_ptr<>` in place of a raw pointer:

```
#include <iostream>
#include <tr1/memory>

int main()
{
#ifdef USE_SHARED_PTR
    std::tr1::shared_ptr<volatile int> i(new int(1));
#else
    volatile int * i = new int(1);
#endif

    long long int h = 0;

    for(long long int j=0;j < 10000000000LL; j++)
    {
        h += *i;
    }

    std::cout << h << std::endl;
    return 0;
}
```

Share Improve this answer

edited Nov 21, 2008 at 20:02

Follow

answered Nov 21, 2008 at 11:38

Nicola Bonelli
8,247 ● 4 ● 29 ● 35

6

The only way to deal with performance problems is to profile your code. The largest part of performance problems is imagined anyway; only profiling will point out to you where your bottlenecks lie.

If it turns out that using smart pointers produces a bottleneck where raw pointers don't, use raw pointers! Until then, I wouldn't worry too much about it; most operations on smart pointers are reasonably fast. You're likely comparing strings too often (or something like that) for them to matter.

Share Improve this answer

Follow

answered Nov 26, 2008 at 18:17

unwesen

2

Reference-counted smart pointers (the most common type) only cost more when you copy, create and delete them. This extra cost can be substantial if you are copying a lot, because most of them are thread-safe.

If you just want an "auto-deleting" pointer, there is the much maligned auto_ptr, or the new and shiny (but not much supported yet) unique_ptr from C++0x.

Share Improve this answer

Follow

answered Nov 21, 2008 at 14:32

Chris Jefferson

7,157 ● 11 ● 46 ● 69

When I last tested, with VC6, the compiler wasn't able to optimize the code with a smart pointer as well as it could with a raw pointer. Things might have changed since then.

Share Improve this answer

Follow

answered Nov 21, 2008 at 15:02

Mark Ransom

308k ● 44 ● 416 ● 647

2 When I last tested an older version of boost (1.34 I think) with VC6, the compiler optimized out the atomic increment of the refcount for a weak_ptr. That made things run considerably faster, although caused quite a few crashes in the threaded libs. – Michel Nov 25, 2008 at 20:17

VC6 I realize is still used heavily by large projects that can't easily switch, but let's be fair here, it was not a mature compiler until VC2003, the 2nd release within the VS.NET product life cycle. – ApplePieIsGood Dec 12, 2008 at 20:10

There's an often overlooked halfway-house between a "manually" managed `std::vector<T*>` (ie raw pointers) and a `std::vector<boost::shared_ptr<T> >`, in the form of the `boost::ptr_container` classes.

These combine the performance of a raw pointer container with the convenience of a container of smart pointers (ie they provide the functionality people would like STL containers of `std::auto_ptr` to provide, if that worked).

Share Improve this answer

Follow

edited May 23, 2017 at 10:27

Community Bot

1 • 1

answered Mar 19, 2009 at 14:01

timday

24.9k • 12 • 87 • 136