How to profile the performance overhead of exception handling

Asked 16 years, 3 months ago Modified 1 year, 2 months ago

Viewed 14k times

15

What is the best way to measure exception handling overhead/performance in C++?

Please give standalone code samples.

I'm targeting Microsoft Visual C++ 2008 and gcc.

I need to get results from the following cases:

1. Overhead when there are no try/catch blocks
2. Overhead when there are try/catch blocks but exceptions are not thrown
3. Overhead when exceptions are thrown

c++ exception gcc visual-c++ profiling

Share

Improve this question

Follow

edited Sep 24, 2023 at 7:50

Jan Schultke
38.3k ● 8 ● 87 ● 168

asked Sep 4, 2008 at 6:38

Antti Kissaniemi
19.2k ● 13 ● 56 ● 47

8 Answers

Sorted by: Highest score (default)

34

Section 5.4 of the draft Technical Report on C++ Performance is entirely devoted to the overhead of exceptions.

Share Improve this answer Follow

answered Sep 4, 2008 at 9:30

Xavier Nodet
5,085 ● 2 ● 38 ● 49

3 This could be improved by quoting the relevant content in the answer, in case the link goes dead. – OMGtechy Jun 9, 2015 at 14:15

If by "entirely devoted" you mean lots of text and zero data then yes. – Stephan Dollberg Aug 15, 2021 at 14:43

10

As a suggestion: don't bother too much with the overhead when exceptions are thrown. Exception handling implementations usually make not throwing fast and catching slow. That's ok since those cases are, well, exceptional.

Carl

Share Improve this answer Follow

answered Sep 4, 2008 at 7:32

Carl Seleborg
13.3k ● 11 ● 59 ● 70

8

Here's the measuring code I came up with. Do you see any problems with it?

Works on Linux and Windows so far, compile with:

```
g++ exception_handling.cpp -o exception_handling [ -O2
```

or for example Visual C++ Express.

To get the base case ("exception support removed from the language altogether"), use:

```
g++ exception_handling.cpp -o exception_handling [ -O2
DNO_EXCEPTIONS
```

or similar settings in MSVC.

Some preliminary results here. They are probably all hokey because of varying machine load, but they do give some idea about the relative exception handling overhead. (Executive summary: none or little when no exceptions are thrown, huge when they actually are thrown.)

```
#include <stdio.h>

// Timer code

#if defined(__linux__)
#include <sys/time.h>
#include <time.h>
```

```
double time()
{
    timeval tv;
    gettimeofday(&tv, 0);
    return 1.0 * tv.tv_sec + 0.000001 * tv.tv_usec;
}
#elif defined(_WIN32)
#include <windows.h>

double get_performance_frequency()
{
    unsigned _int64 frequency;
    QueryPerformanceFrequency((LARGE_INTEGER*) &freque
works
    return double(frequency);
}

double performance_frequency = get_performance_frequen

double time()
{
    unsigned _int64 counter;
    QueryPerformanceCounter((LARGE_INTEGER*) &counter)
    return double(counter) / performance_frequency;
}
#else
# error time() not implemented for your platform
#endif

// How many times to repeat the whole test
const int repeats = 10;

// How many times to iterate one case
const int times = 1000000;

// Trick optimizer to not remove code
int result = 0;



// Case 1. No exception thrown nor handled.

void do_something()
```

```
{
    ++result;
}

void case1()
{
    do_something();
}



// Case 2. No exception thrown, but handler installed

#ifndef NO_EXCEPTIONS
void do_something_else()
{
    --result;
}

void case2()
{
    try
    {
        do_something();
    }
    catch (int exception)
    {
        do_something_else();
    }
}



// Case 3. Exception thrown and caught

void do_something_and_throw()
{
    throw ++result;
}

void case3()
{
    try
    {
        do_something_and_throw();
    }
    catch (int exception)
    {
        result = exception;
    }
}
#endif // !NO_EXCEPTIONS

void (*tests[])() =
{
    case1,
#ifndef NO_EXCEPTIONS
    case2,
    case3
#endif // !NO_EXCEPTIONS
};

int main()
{
#ifdef NO_EXCEPTIONS
    printf("case0\n");
#else
    printf("case1\tcase2\tcase3\n");
#endif
    for (int repeat = 0; repeat < repeats; ++repeat)
    {
        for (int test = 0; test < sizeof(tests)/sizeof
        {
            double start = time();

            for (int i = 0; i < times; ++i)
                tests[test]();

            double end = time();

            printf("%f\t", (end - start) * 1000000.0 /
        }
        printf("\n");
    }

    return result; // optimizer is happy - we produce
}
```

Share Improve this answer Follow

edited Sep 4, 2008 at 9:13

answered Sep 4, 2008 at 7:38

Antti Kissaniemi
19.2k ● 13 ● 56 ● 47

1 Did you try to do a test where you'd catch any exception (catch (...)). It may be different than just catching a single exception class... – Xavier Nodet Sep 23, 2008 at 20:48

3

[Kevin Frei](#) talks about exception handling performance cost in his speech "[The Cost of C++ Exception Handling on Windows](#)". (Under "Summary & Conclusions" there is one list item that says "[Exception handling performance cost is] not always measurable".)

Share Improve this answer Follow

answered Sep 5, 2008 at 15:55

andreas
7,393 ● 3 ● 25 ● 37

2

There's no really good way to meassure that in code. You would need to use a a profiler.

This won't show you directly how much time is spent with exception handling but with a little bit of research you will find out which of the runtime methods deal with exceptions (for example for VC++.NET it's __cxx_exc[...]).

Add their times up and you have the overhead. In our project we used vTunes from Intel which works with both Visual C++ and gcc.

Edit: Well, if you just need a generic number that might work. Thought you had an actual application to profile where you can't just turn off exceptions.

Share Improve this answer Follow

edited Sep 4, 2008 at 11:41

answered Sep 4, 2008 at 7:42

Maximilian
4,828 ● 6 ● 30 ● 28

2

Another note on exception handling performance: simple tests don't take caching into account. The try-code and the catch-code are both so small that everything fits in the instruction and data caches. But compilers may try to move the catch-code far away from the try-code, which reduces the amount of code to keep in cache normally, thus enhancing performance.

If you compare exception handling to traditional C-style return-value checking, this caching effect should be taken into account as well (the question is usually ignored in discussions).

Carl

Share Improve this answer

Follow

answered Sep 4, 2008 at 14:53

Carl Seleborg
13.3k ● 11 ● 59 ● 70

0

Won't the answer depend on what cleanup has to happen as a result of a throw? If an excpetion is thrown that causes a whole load of objects to go out of scope up the stack, then that will add to the overhead.

In other words, I'm not sure if there is a an answer to the 3rd question that is independent of the specifics of the code.

Share Improve this answer

Follow

answered Sep 4, 2008 at 20:58

Greg Whitfield
5,719 ● 2 ● 31 ● 32

Good point, and I wonder how many tests don't account for the time spent destroying those objects in the version coded without exceptions, where it is spread out over more routines. – Andrew Lazarus Mar 7, 2014 at 5:31

-1

Full details on how g++ handles exceptions is shown here. It describes it as being for the Itanium architecture, however the general techniques used are the same. It won't tell you exact overhead in terms of time, however you can glean what the rough code overhead will be.

Share Improve this answer

Follow

answered Feb 17, 2009 at 17:32

Arcane
1,228 ● 1 ● 8 ● 15