faster Math.exp() via JNI?

Asked 16 years, 3 months ago Modified 9 years, 6 months ago

Viewed 5k times

10

I need to calculate `Math.exp()` from java very frequently, is it possible to get a native version to run faster than **java**'s `Math.exp()`??

I tried just jni + C, but it's slower than just plain **java**.

java c optimization java-native-interface

Share

Improve this question

Follow

edited Jun 2, 2015 at 19:25

Noam M

3,164 ● 5 ● 27 ● 41

asked Sep 15, 2008 at 20:06

Dan

1,721 ● 2 ● 22 ● 36

have you done any performance testing to get exact numbers of the time it takes Math.exp() versus the JNI version? How about after being called 10k times to see the effect of the JIT? – martinatime Sep 15, 2008 at 20:36

This depends on your JVM, but usually `Math.exp` *is implemented in C*. You may want to use a faster (less precise) algorithm though. – Joni Feb 17, 2012 at 8:01

15 Answers

Sorted by: Highest score (default)

16

This has already been requested several times (see e.g. here). Here is an approximation to Math.exp(), copied from this blog posting:

```
public static double exp(double val) {
    final long tmp = (long) (1512775 * val + (10726932
    return Double.longBitsToDouble(tmp << 32);
}
```

It is basically the same as a lookup table with 2048 entries and linear interpolation between the entries, but all this with IEEE floating point tricks. Its 5 times faster than Math.exp() on my machine, but this can vary drastically if you compile with -server.

Share Improve this answer Follow

edited May 23, 2017 at 12:00
Community Bot
1 ●1

answered Jan 8, 2009 at 16:43
martinus
18k ●16 ●74 ●92

1 See also this gist.github.com/Alrecenk/55be1682fe46cdd89663 – tobi delbruck Jun 30, 2021 at 9:31

+1 to writing your own exp() implementation. That is, if this is *really* a bottle-neck in your application. If you can

12

✓

deal with a little inaccuracy, there are a number of extremely efficient exponent estimation algorithms out there, some of them dating back centuries. As I understand it, Java's exp() implementation is fairly slow, even for algorithms which must return "exact" results.

Oh, and don't be afraid to write that exp() implementation in pure-Java. JNI has a lot of overhead, and the JVM is able to optimize bytecode at runtime sometimes even beyond what C/C++ is able to achieve.

Share Improve this answer Follow

answered Sep 15, 2008 at 20:22

Daniel Spiewak
55.1k ● 14 ● 111 ● 120

Two important points here: (1) JNI overhead often outweighs all other considerations; (2) JVM JIT is surprisingly good (sometimes equal or faster than C/C++) at optimizing small methods as long as the machine is "warmed" sufficiently.
– kevinarpe Mar 2, 2016 at 9:55

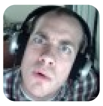
6

Use Java's.

Also, cache results of the exp and then you can look up the answer faster than calculating them again.

Share Improve this answer Follow

answered Sep 15, 2008 at 20:07



jjnguy
139k ● 53 ● 297 ● 326

How would you cache the results? Caching can be quite costly: I tried with a HashMap and it was twice slower than simply computing the exp. In my test I compute 71M exp, but with "only" 1.8M different arguments. – Juh_ Oct 8, 2015 at 7:29

5

You'd want to wrap whatever loop's calling `Math.exp()` in C as well. Otherwise, the overhead of marshalling between Java and C will overwhelm any performance advantage.

Share Improve this answer

Follow

answered Sep 15, 2008 at 20:08

John Millikin

201k ● 41 ● 215 ● 227

3

You might be able to get it to run faster if you do them in batches. Making a JNI call adds overhead, so you don't want to do it for each exp() you need to calculate. I'd try passing an array of 100 values and getting the results to see if it helps performance.

Share Improve this answer

Follow

answered Sep 15, 2008 at 20:17

Bill the Lizard

405k ● 211 ● 572 ● 889

2

The real question is, has this become a bottle neck for you? Have you profiled your application and found this to be a major cause of slow down?

If not, I would recommend using Java's version. Try not to pre-optimize as this will just cause development slow down. You may spend an extended amount of time on a problem that may not be a problem.

That being said, I think your test gave you your answer. If jni + C is slower, use java's version.

Share Improve this answer

Follow

answered Sep 15, 2008 at 20:11

scubabbl

12.8k ● 7 ● 38 ● 37

2

Commons Math3 ships with an optimized version: `FastMath.exp(double x)`. It did speed up my code significantly.

Fabien ran some tests and found out that it was almost twice as fast as `Math.exp()`:

```
0.75s for Math.exp     sum=1.7182816693332244E7
0.40s for FastMath.exp sum=1.7182816693332244E7
```

Here is the javadoc:

Computes exp(x), function result is nearly rounded. It will be correctly rounded to the theoretical value for 99.9% of input values, otherwise it will have a 1 UPL error.

Method:

```
    Lookup intVal = exp(int(x))
    Lookup fracVal = exp(int(x-int(x) / 1024.0) * 1024
    Compute z as the exponential of the remaining bits
one
    exp(x) = intVal * fracVal * (1 + z)
```

Accuracy: Calculation is done with 63 bits of precision, so result should be correctly rounded for 99.9% of input values, with less than 1 ULP error otherwise.

Share Improve this answer

Follow

answered May 30, 2014 at 21:21

Renaud

16.5k ● 7 ● 82 ● 80

best solution for me! – Juh_ Oct 8, 2015 at 7:54

0

Since the Java code will get compiled to native code with the just-in-time (JIT) compiler, there's really no reason to use JNI to call native code.

Also, you shouldn't cache the results of a method where the input parameters are floating-point real numbers. The gains obtained in time will be very much lost in amount of space used.

Share Improve this answer

Follow

answered Sep 15, 2008 at 20:09

Alan Krueger

4,786 ● 4 ● 37 ● 50

0

The problem with using JNI is the overhead involved in making the call to JNI. The Java virtual machine is pretty optimized these days, and calls to the built-in Math.exp() are automatically optimized to call straight through to the C exp() function, and they might even be optimized into straight x87 floating-point assembly instructions.

Share Improve this answer Follow

answered Sep 15, 2008 at 20:10

Adam Rosenfield
399k ● 101 ● 522 ● 597

0

There's simply an overhead associated with using the JNI, see also: http://java.sun.com/docs/books/performance/1st_edition/html/JPNativeCode.fm.html

So as others have suggested try to collate operations that would involve using the JNI.

Share Improve this answer Follow

answered Sep 15, 2008 at 20:13

tonylo
3,352 ● 3 ● 29 ● 27

0

Write your own, tailored to your needs.

For instance, if all your exponents are of the power of two, you can use bit-shifting. If you work with a limited range or set of values, you can use look-up tables. If you don't need pin-point precision, you use an imprecise, but faster, algorithm.

Share Improve this answer

Follow

answered Sep 15, 2008 at 20:18

user9116

597 ● 3 ● 7

0

There is a cost associated with calling across the JNI boundary.

If you could move the loop that calls exp() into the native code as well, so that there is just one native call, then you might get better results, but I doubt it will be significantly faster than the pure Java solution.

I don't know the details of your application, but if you have a fairly limited set of possible arguments for the call, you could use a pre-computed look-up table to make your Java code faster.

Share Improve this answer

Follow

answered Sep 15, 2008 at 20:20

Dan Dyer

54.4k ● 19 ● 135 ● 166

0

There are faster algorithms for exp depending on what your'e trying to accomplish. Is the problem space restricted to a certain range, do you only need a certain resolution, precision, or accuracy, etc.

If you define your problem very well, you may find that you can use a table with interpolation, for instance, which will blow nearly any other algorithm out of the water.

What constraints can you apply to exp to gain that performance trade-off?

-Adam

Share Improve this answer Follow

answered Sep 15, 2008 at 20:21

Adam Davis

93.5k ● 60 ● 271 ● 333

0

> I run a fitting algorithm and the minimum error of the fitting result is way larger than the precision of the Math.exp().

Transcendental functions are always much more slower than addition or multiplication and a well-known bottleneck. If you know that your values are in a narrow range, you can simply build a lookup-table (Two sorted array ; one input, one output). Use Arrays.binarySearch to find the correct index and interpolate value with the elements at [index] and [index+1].

Another method is to split the number. Lets take e.g. 3.81 and split that in 3+0.81. Now you multiply e = 2.718 three times and get 20.08.

Now to 0.81. All values between 0 and 1 converge fast with the well-known exponential series

1+x+x^2/2+x^3/6+x^4/24.... etc.

Take as much terms as you need for precision; unfortunately it's slower if x approaches 1. Lets say you go to x^4, then you get 2.2445 instead of the correct 2.2448

Then multiply the result 2.781^3 = 20.08 with 2.781^0.81 = 2.2445 and you have the result 45.07 with an error of one part of two thousand (correct: 45.15).

Share Improve this answer Follow

edited Sep 15, 2008 at 21:59

answered Sep 15, 2008 at 21:45

TSK
1,144 ● 1 ● 7 ● 3

▲

0

▼

It might not be relevant any more, but just so you know, in the newest releases of the OpenJDK (see here), Math.exp should be made an intrinsic (if you don't know what that is, check here).

This will make performance unbeatable on most architectures, because it means the Hotspot VM will replace the call to Math.exp by a processor-specific implementation of exp at runtime. You can never beat these calls, as they are optimized for the architecture...

Share Improve this answer

Follow

answered Mar 1, 2013 at 10:44

reverse_engineer

4,269 ● 4 ● 19 ● 27