What to do with Java BigDecimal performance?

Asked 15 years, 9 months ago Modified 2 years, 11 months ago

Viewed 48k times

66

I write currency trading applications for living, so I have to work with monetary values (it's a shame that Java still doesn't have decimal float type and has nothing to support arbitrary-precision monetary calculations). "Use BigDecimal!" — you might say. I do. But now I have some code where performance **is** an issue, and BigDecimal is more than 1000 times (!) slower than `double` primitives.

The calculations are very simple: what the system does is calculating `a = (1/b) * c` many many times (where `a`, `b` and `c` are fixed-point values). The problem, however, lies with this `(1/b)`. I can't use fixed point arithmetic because there is no fixed point. And `BigDecimal result = a.multiply(BigDecimal.ONE.divide(b).multiply(c)` is not only ugly, but sluggishly slow.

What can I use to replace BigDecimal? I need at least 10x performance increase. I found otherwise excellent JScience library which has arbitrary-precision arithmetics, but it's even slower than BigDecimal.

Any suggestions?

java optimization currency math bigdecimal

Share

Improve this question

Follow

edited Apr 24, 2009 at 9:34

Elijah
13.6k ● 10 ● 60 ● 89

asked Mar 4, 2009 at 17:58

Alexander Temerev
661 ● 1 ● 6 ● 5

---

if the values of b and c have little variation, you could memoize the values. – sfossen Mar 4, 2009 at 18:06

Oddly enough, this was something that was easier in C. Just link against a BCD library and you were done!
– Brian Knoblauch Mar 4, 2009 at 19:29

I remember attending a sales presentation from IBM for a hardware accelerated implementation of BigDecimal. So if your target platform is IBM System z, or System p, you could exploit this seamlessly. – toolkit Mar 4, 2009 at 20:27

2 not odd at all, Java makes easier common tasks, while big decimal is not so much common. – Vladimir Dyuzhev Mar 4, 2009 at 21:55

1 Don't laugh, but one solution is to use PHP. I just found this posting while investigating the reason why a small program I converted from PHP to Java was so much slower in Java than PHP. – Alex R Apr 24, 2010 at 1:08

19 Answers

Sorted by: Highest score (default)

39

May be you should start with replacing a = (1/b) * c with a = c/b ? It's not 10x, but still something.

If I were you, I'd create my own class Money, which would keep long dollars and long cents, and do math in it.

Share Improve this answer Follow

answered Mar 4, 2009 at 18:02
Vladimir Dyuzhev
18.3k ● 10 ● 52 ● 62

And implement division, rounding, exponentiation etc. myself from scratch? :) – Alexander Temerev Mar 4, 2009 at 18:06

5 Yes, I believe that's what he's suggesting. – Ryann Graham Mar 4, 2009 at 18:10

This is quite difficult task to get it right (if you doubt, take a look in Java Math classes). I don't believe no one else does high-performance monetary calculations in Java. – Alexander Temerev Mar 4, 2009 at 18:14

11 It's a hard task to do it for a general-purpose library. For specific application (which uses only a **subset**) of operations it's trivial. In fact, I have such as class in my own app, and it only need 5 or 6 common operations. – Vladimir Dyuzhev Mar 4, 2009 at 18:16

17 If you write currency trading apps for a living, these calculations are your 'core functionality'. You will need to spend time and effort getting them right to give yourself a competitive advantage. – DJClayworth Mar 4, 2009 at 18:28

Most double operations give you more than enough precision. You can represent $10 trillion with cent

20

accuracy with double which may be more than enough for you.

In all the trading systems I have worked on (four different banks), they have used double with appropriate rounding. I don't see any reason to be using BigDecimal.

Share Improve this answer Follow

edited Apr 25, 2009 at 0:59

answered Mar 4, 2009 at 19:26

Peter Lawrey
533k ● 82 ● 767 ● 1.1k

2 Yes, the precision of double is more than sufficient. I do such things too, end it works perfectly unless I forget to round and the customer sees something like -1e-13 where they expect a non-negative result. – maaartinus Aug 4, 2012 at 21:35

2 I have since designed three different trading systems for different funds and used `double` for prices or `long` cents. – Peter Lawrey Aug 5, 2012 at 6:50

15

So my original answer was just flat out wrong, because my benchmark was written badly. I guess I'm the one who should have been criticized, not OP ;) This may have been one of the first benchmarks I ever wrote... oh well, that's how you learn. Rather than deleting the answer, here are the results where I'm not measuring the wrong thing. Some notes:

- Precalculate the arrays so I don't mess with the results by generating them
- Don't *ever* call `BigDecimal.doubleValue()`, as it's extremely slow
- Don't mess with the results by adding `BigDecimal`s. Just return one value, and use an if statement to prevent compiler optimization. Make sure to have it work most of the time to allow branch prediction to eliminate that part of the code, though.

Tests:

- BigDecimal: do the math exactly as you suggested it
- BigDecNoRecip: (1/b) * c = c/b, just do c/b
- Double: do the math with doubles

Here is the output:

```
 0% Scenario{vm=java, trial=0, benchmark=Double} 0.34
33% Scenario{vm=java, trial=0, benchmark=BigDecimal} 3
trials
67% Scenario{vm=java, trial=0, benchmark=BigDecNoRecip
10 trials

    benchmark      ns linear runtime
       Double   0.335 =
   BigDecimal 356.031 ==============================
BigDecNoRecip 301.909 =========================

vm: java
trial: 0
```

Here's the code:

```java
import java.math.BigDecimal;
import java.math.MathContext;
import java.util.Random;

import com.google.caliper.Runner;
import com.google.caliper.SimpleBenchmark;

public class BigDecimalTest {
  public static class Benchmark1 extends SimpleBenchma
    private static int ARRAY_SIZE = 131072;

    private Random r;

    private BigDecimal[][] bigValues = new BigDecimal[
    private double[][] doubleValues = new double[3][];

    @Override
    protected void setUp() throws Exception {
      super.setUp();
      r = new Random();

      for(int i = 0; i < 3; i++) {
        bigValues[i] = new BigDecimal[ARRAY_SIZE];
        doubleValues[i] = new double[ARRAY_SIZE];

        for(int j = 0; j < ARRAY_SIZE; j++) {
          doubleValues[i][j] = r.nextDouble() * 100000
          bigValues[i][j] = BigDecimal.valueOf(doubleV
        }
      }
    }

    public double timeDouble(int reps) {
      double returnValue = 0;
      for (int i = 0; i < reps; i++) {
        double a = doubleValues[0][reps & 131071];
        double b = doubleValues[1][reps & 131071];
        double c = doubleValues[2][reps & 131071];
        double division = a * (1/b) * c;
        if((i & 255) == 0) returnValue = division;
      }
      return returnValue;
    }
    public BigDecimal timeBigDecimal(int reps) {
      BigDecimal returnValue = BigDecimal.ZERO;
      for (int i = 0; i < reps; i++) {
        BigDecimal a = bigValues[0][reps & 131071];
        BigDecimal b = bigValues[1][reps & 131071];
        BigDecimal c = bigValues[2][reps & 131071];
        BigDecimal division = a.multiply(BigDecimal.ON
MathContext.DECIMAL64).multiply(c));
        if((i & 255) == 0) returnValue = division;
      }
      return returnValue;
    }

    public BigDecimal timeBigDecNoRecip(int reps) {
      BigDecimal returnValue = BigDecimal.ZERO;
      for (int i = 0; i < reps; i++) {
        BigDecimal a = bigValues[0][reps & 131071];
        BigDecimal b = bigValues[1][reps & 131071];
        BigDecimal c = bigValues[2][reps & 131071];
        BigDecimal division = a.multiply(c.divide(b, M
        if((i & 255) == 0) returnValue = division;
      }
      return returnValue;
    }
  }

  public static void main(String... args) {
    Runner.main(Benchmark1.class, new String[0]);
  }
}
```

Share Improve this answer Follow

edited Apr 4, 2014 at 14:27

answered Feb 6, 2013 at 16:23

durron597
32.3k ● 18 ● 102 ● 160

1 +1 for making the benchmark but -1 for the implementation. You're mostly measuring how long it takes to create a `BigDecimal` ... or more exactly, the creation overhead is present in all benchmarks and may dominate them. Unless it's what you wanted (but why?), you'd need to pre-create the values and store in an array. – maaartinus Apr 4, 2014 at 13:50

@maaartinus Well this is embarrassing, I have gotten *so* much better at writing benchmarks in the last 14 months. I'll edit the post now – durron597 Apr 4, 2014 at 14:15

1 +1 now the values make sense! I'm not sure about what you're doing with the `if`. It probably won't get optimized away but it may. I used to do something like `result += System.identityHashCode(o)` but then I discovered the JMH `BlackHole`. – maaartinus Apr 4, 2014 at 15:25

@maaartinus Could you tell more about the JMH blackhole please? – Amrinder Arora Mar 17, 2017 at 16:04

@AmrinderArora Not really. The Blackhole is a pretty complicated thing doing something with the input, so it can't be optimized away. It's optimized for speed even in the multithreaded case. – maaartinus Mar 18, 2017 at 6:08

9

Assuming you can work to some arbitrary but known precision (say a billionth of a cent) and have a known maximum value you need handle (a trillion trillion dollars?) you can write a class which stores that value as an integer number of billionths of a cent. You'll need two longs to represent it. That should be maybe ten times as slow as using double; about a hundred times as fast as BigDecimal.

Most of the operations are just performing the operation on each part and renormalizing. Division is slightly more complicated, but not much.

EDIT:In response to the comment. You will need to implement a bitshift operation on your class (easy as along as the multiplier for the high long is a power of two). To do division shift the divisor until it's not quite bigger than the dividend; subtract shifted divisor from dividend and increment the result (with appropriate shift). Repeat.

EDIT AGAIN:You may find BigInteger does what you need here.

Share Improve this answer Follow

edited Jun 17, 2010 at 20:48

answered Mar 4, 2009 at 18:23

DJClayworth
26.8k ● 9 ● 58 ● 80

Will you suggest me an algorithm for division in this case?
– Alexander Temerev Mar 4, 2009 at 18:28

5

Store longs as the number of cents. For example, `BigDecimal money = new BigDecimal ("4.20")` becomes `long money = 420`. You just have to remember to mod by 100 to get dollars and cents for output. If you need to track, say, tenths of a cent, it'd become `long money = 4200` instead.

Share Improve this answer

Follow

answered Mar 4, 2009 at 18:06

Pesto
23.9k ● 2 ● 73 ● 76

that's adding even more operations. so that would be slower. – sfossen Mar 4, 2009 at 18:07

1 How is it slower? Math computations on long are far, far faster than those on BigDecimal. You only need convert to dollars and cents for output. – Pesto Mar 4, 2009 at 18:10

2 I need to track (in intermediate calculations) billionths of cents. Let's say we have a quote for USD/JPY: 99.223. Somewhere else I will need a JPY/USD quote, which is around 0.0100779022 (I need even more precision). – Alexander Temerev Mar 4, 2009 at 18:10

1 @Pesto: missed the long conversion, however, 2 decimal points is almost never acceptable in monetary calculations, although similar to my suggestion of fixed point math. – sfossen Mar 4, 2009 at 18:25

1 @Pesto: Ya, a single primitive won't be enough, which is why I suggested a fixed point library. – sfossen Mar 4, 2009 at 18:53

5

You might want to move to fixed point math. Just searching for some libraries right now. on sourceforge [fixed-point](#) I haven't looked at this in depth yet. [beartonics](#)

Did you test with org.jscience.economics.money? since that has assured accuracy. The fixed point will only be as accurate as the # of bits assigned to each piece, but is fast.

Share Improve this answer Follow

edited Mar 4, 2009 at 18:17

answered Mar 4, 2009 at 18:09

sfossen
4,778 ● 25 ● 18

JScience is excellent library, I must admit; however, there is no performance improvement compared to BigDecimal. – Alexander Temerev Mar 4, 2009 at 18:29

Using a fixed point library will get you speed, but you will lose some precision. You could try using BigInteger to make a fixed point library. – sfossen Mar 4, 2009 at 18:35

1 Also don't use a power of ten, if you do this, use a power of 2. power of ten easier for humans but harder for computers :P – sfossen Mar 4, 2009 at 21:08

3

Personally, I don't think BigDecimal is ideal for this.

You really want to implement your own Money class using longs internally to represent the smallest unit (i.e. cent, 10th cent). There is some work in that, implementing `add()` and `divide()` etc, but it's not really that hard.

Share Improve this answer Follow

answered Mar 4, 2009 at 19:12

SCdF
59.3k ● 24 ● 79 ● 114

What version of the JDK/JRE are you using?

2

Also you might try ArciMath BigDecimal to see if theirs speeds it up for you.

Edit:

I remember reading somewhere (I think it was Effective Java) that the BigDecmal class was changed from being JNI called to a C library to all Java at some point... and it got faster from that. So it could be that any arbitrary precision library you use is not going to get you the speed you need.

Share Improve this answer Follow

edited Mar 4, 2009 at 19:00

answered Mar 4, 2009 at 18:40

TofuBeer
61.5k ● 18 ● 119 ● 163

2

```
Only 10x performance increase desired for something th
primitive?!.
```

Throwing a bit more hardware at this might be cheaper (considering the probability of having a currency calculation error).

Share Improve this answer Follow

answered Jun 29, 2009 at 5:00

Ryan Fernandes
8,516 ● 7 ● 37 ● 54

2

On a 64bit JVM creating your BigDecimal as below makes it about 5x faster:

```
BigDecimal bd = new BigDecimal(Double.toString(d), Mat
```

Share Improve this answer

Follow

answered Apr 27, 2017 at 19:58

tsquared

119 ● 6

1

1/b is not exactly representable with BigDecimal either. See the API docs to work out how the result is rounded.

It shouldn't be *too* difficult to write your own fixed decimal class based around a long field or two. I don't know any appropriate off the shelf libraries.

Share Improve this answer

Follow

answered Mar 4, 2009 at 18:09

Tom Hawtin - tackline

147k ● 30 ● 221 ● 312

I don't need exact representation; I need knowable precision. – Alexander Temerev Mar 4, 2009 at 18:14

1

I know that I'm posting under very old topic, but this was the first topic found by google. Consider moving your calculations to the database from which you probably are taking the data for processing. Also I agree with Gareth Davis who wrote:

> . In most bog standard webapps the overhead of jdbc access and accessing other network resources swamps any benefit of having really quick math.

In most cases wrong queries have higher impact on performance than math library.

Share Improve this answer Follow

answered Jun 18, 2012 at 10:41

Marek Kowalczyk
26 ● 2

1

Commons Math - The Apache Commons Mathematics Library

http://mvnrepository.com/artifact/org.apache.commons/commons-math3/3.2

According to my own benchmarking for my specific use case it's 10 - 20x slower than double (much better than 1000x) - basically for addition / multiplication. After benchmarking another algorithm which had a sequence of additions followed by an exponentiation the performance decrease was quite a bit worse: 200x - 400x. So it seems pretty fast for + and *, but not exp and log.

> *Commons Math is a library of lightweight, self-contained mathematics and statistics components addressing the most common problems not available in the Java programming language or Commons Lang.*

Note: The API protects the constructors to force a factory pattern while naming the factory DfpField (rather than the somewhat more intuitive DfpFac or DfpFactory). So you have to use

```
new DfpField(numberOfDigits).newDfp(myNormalNumber)
```

to instantiate a Dfp, then you can call `.multiply` or whatever on this. I thought I'd mention this because it's a bit confusing.

Share Improve this answer Follow

answered Sep 12, 2013 at 15:58

samthebest
31.5k ● 25 ● 105 ● 148

0

Is JNI a possibility? You may be able to recover some speed and potentially leverage existing native fixed point libraries (maybe even some SSE* goodness too)

Perhaps http://gmplib.org/

Share Improve this answer Follow

edited Mar 4, 2009 at 19:15

answered Mar 4, 2009 at 19:08

basszero
30k ● 9 ● 58 ● 80

1 it is unlikely that JNI will help performance here, unless the calculations can be batched. JNI introduces significant overhead as you cross the JVM/native boundary.
– Kevin Day Mar 5, 2009 at 5:30

1 You are correct that the boundary does have a slowdown and I've definitely felt that pain but if BigDecimal truly has the claimed 1000x slowdown and JNI was only a fraction, it may be worth it. – basszero Mar 5, 2009 at 13:50

0

Maybe you should look into getting hardware accelerated decimal arithmetics?

http://speleotrove.com/decimal/

Share Improve this answer Follow

answered Mar 4, 2009 at 19:25

John Nilsson
17.3k ● 8 ● 35 ● 42

0

Had a similar problem to this in an equity trading system back in 99. At the very start of the design we choose to have every number in the system represented as a long multiplied by 1000000 thus 1.3423 was 1342300L. But the main driver for this was memory foot print rather than straight line performance.

One word on caution, I wouldn't do this again today unless I was *really* sure that the math performance was super critical. In most bog standard webapps the overhead of jdbc access and accessing other network resources swamps any benefit of having really quick math.

Share Improve this answer Follow

answered Mar 11, 2009 at 20:18

Gareth Davis
28k ● 12 ● 76 ● 107

0

It seems like the simplest solution is to use BigInteger instead of long to implement pesto's solution. If it seems messy it would be easy to write a class that wraps BigInteger to hide the precision adjustment.

Share Improve this answer Follow

answered Oct 30, 2010 at 5:19

ozone
311 ● 2 ● 9

0

easy... round your results often will eliminate double data type's error. if you are doing balance calculation, you have to also consider who will own the more/less penny caused by rounding.

bigdeciaml calculation produces more/less penny too, consider 100/3 case.

Share Improve this answer Follow

answered May 4, 2011 at 16:20

Chris
29 ● 1

1 Rounding results *decreases* accuracy, rather than increasing it. – Hannele Jan 18, 2012 at 20:02

1 @Hannele Most of the time yes, but sometimes it indeed *increases* it. For example, when computing sum of prices where each of them is given with two decimal places, the rounding to two decimal places *guarantees* a correct result (unless you're summing many billions of values). – maaartinus Aug 4, 2012 at 21:40

@maaartinus You have an interesting point! However, I don't believe that's directly applicable to the OP (division). – Hannele Aug 7, 2012 at 15:19

1 @Hannele: Agreed, rounding helps only if you know how many decimal places the result should have which is not the case with division. – maaartinus Aug 7, 2012 at 16:21

1 If `double` values are scaled in a manner such that any domain-required rounding is always to a whole number, then any rounded values will be "exact" unless they're really big. For example, if things that will round to the nearest $0.01 are stored as a number of pennies rather than dollars, `double` will can penny-rounded amounts precisely unless they exceeed $45,035,996,273,704.96. – supercat Oct 22, 2013 at 19:20

0

I know this is a really old thread, but i am writing an app (incidentally a trading app), in which computation of the indicators like MACD (which computes multiple exponential moving averages) over several thousand ticks of historical candlesticks was taking an unacceptable amount of time (several minutes). I was using BigDecimal.

every time i scrolled or resized the window, it would have to just iterate through the cached values to resize the Y scale, but even that would take several seconds to update. it made the app unusable. every time i would tweak the parameters for various indicators, it would take several minutes again to recompute.

then i switched it all to double and it's sooooo much faster. the problem was that i cache values using a hashmap. the solution i came up with uses a pool of wrappers for the double values. by pooling the wrappers, you don't take the performance hit of autoboxing to/from Double.

the app now calculates MACD (+MACD signal, MACD histogram) instantly with no lag. it's amazing how expensive BigDecimal object creation was. think about something like a.add(b.multiply(c)).scale(3) and how many objects that one statement creates.

```
 import java.util.HashMap;

public class FastDoubleMap<K>
{
    private static final Pool<Wrapper> POOL = new Pool
    {
        protected Wrapper newInstance()
        {
            return new Wrapper();
        }
    };

    private final HashMap<K, Wrapper> mMap;

    public FastDoubleMap()
    {
        mMap = new HashMap<>();
    }

    public void put( K pKey, double pValue )
    {
        Wrapper lWrapper = POOL.checkOut();
        lWrapper.mValue = pValue;
        mMap.put( pKey, lWrapper );
    }

    public double get( K pKey )
    {
        Wrapper lWrapper  = mMap.get( pKey );
        if( lWrapper == null )
        {
            return Double.NaN;
        }
        else
        {
            return lWrapper.mValue;
        }
    }

    public double remove( K pKey )
    {
        Wrapper lWrapper = mMap.remove( pKey );
        if( lWrapper != null )
        {
            double lDouble = lWrapper.mDouble;
            POOL.checkIn( lWrapper );
            return lDouble;
        }
        else
        {
            return Double.NaN;
        }
    }

    private static class Wrapper
        implements Pooled
    {
        private double mValue ;

        public void cleanup()
        {
            mValue = Double.NaN;
        }
    }
}
```

Share Improve this answer Follow

edited Jun 30, 2021 at 16:45

answered Jun 25, 2021 at 16:49

TheWheelMan

21 ● 2