Journal of Physics Special Topics

A2_1 The fast inverse square root in scientific computing

T.O. Hands, I. Griffiths, D.A. Marshall, G. Douglas

Department of Physics and Astronomy, University of Leicester. Leicester, LE1 7RH.

Oct 20, 2011.

Abstract
Scientific computing often requires the calculation of square roots - a process which is very expensive in terms of processing time compared to most other floating point operations. In this paper we examine the use of a faster but less accurate square root algorithm in scientific computing using N-body simulations as an example. The results show that the the speed increase afforded by the faster algorithm is not offset by the reduction in accuracy, and hence an alternate approach is suggested.

Introduction
Finding the square root of a floating point number is often the most computationally expensive part of a scientific code, with commonly implemented algorithms often requiring several iterations of a solver to perform the operation. Pushes to improve the speed of such algorithms have largely come from the computer games industry, where square root operations are required to unitise vectors for lighting. This operation requires dividing each component of a vector by its magnitude - the calculation of which requires a square root to be computed. Hence computer game developers have created novel algorithms to compute this inverse square root [1].

N-body simulations face a similar challenge to computer games. The most commonly used method of finding the acceleration of each body i in such a simulation is to compute each component of the vector given by

$$\boldsymbol{a}_i = -G\sum_{j\neq i}\frac{m_j\boldsymbol{r_{ij}}}{|\boldsymbol{r_{ij}}|^3}. \tag{1}$$

Here G is the gravitational constant, m_j is the mass of body j and $\boldsymbol{r_{ij}} = \boldsymbol{r_i} - \boldsymbol{r_j}$ is a vector connecting the two bodies i and j (where $\boldsymbol{r}_i$ denotes the position vector of body i). The vector magnitude in the denominator of the fraction is where this force computation incurs most of its computing cost. Hence the aim of this paper is to determine whether or not a faster but approximate inverse square root would be appropriate in such simulations. Fortunately, it is no longer necessary to rely on custom algorithms for such optimisations. With the advent of the SSE (Streaming Single instrutction, multiple data Extensions) instruction set, an instruction called `rsqrtss` became available on nearly all modern x86 and x64 processors which performs such a square root [2]. Previous testing indicates that this method may be upto 10x faster than a standard square root [3]. It is this instruction which we test against the standard and more accurate `sqrtss` in our N-body code.

Method
To test the efficiency gains made by using the fast inverse square root, a simple N-body integrator was created. This program used a leapfrog integrator for accuracy, unsmoothed forces and the simulation units were chosen such that $G = 1$.

A test problem was set-up using 2 bodies of equal mass $m = 0.5$, each at a distance $r = 0.5$ from the origin with both bodies starting on the x axis. The two body circular orbit equation used to set-up the velocities of the bodies is as follows

$$G(m_i + m_j) = \frac{4\pi^2 a^3}{T^2}, \tag{2}$$

where a is the distance between the two bodies and T is the period of the bodies [4]. Using this equation with the values relevant to our set-up, along with

$$v = \frac{2\pi r}{T}, \tag{3}$$

(where r is the distance of the body in question from the barycentre of the two bodies) yields an initial velocity for the first body of $v_y = 0.5$ and $v_y = -0.5$ for the second. With these conditions, 25 simulations were run using both the normal square root and the fast reciprocal square root in order to measure the speed increase. The implementation was such that the only difference in the two methods was the calculation of the square root - every other operation remained identical. The simulations were run over a time period of 0 - 200 in the time units of our simulation using a timestep $\tau = 0.001$. A further simulation was run until $t = 10000$ using $\tau = 0.01$ to observe the difference in the accuracy of each simulation.

Method	Mean time to complete run (ms)
`sqrtss`	38.78
`rsqrtss`	37.05

Table 1: Mean times to perform the run of simulations from 20 data points

Method	Mean distance ($\|\bar{\boldsymbol{r}}\|$) from origin to body 1	Magnitude of $0.5 - \|\bar{\boldsymbol{r}}\|$
`sqrtss`	0.5000264	0.0000264
`rsqrtss`	0.4998505	0.0001495

Table 2: Mean distance from the origin to body 1 over the whole simulation time

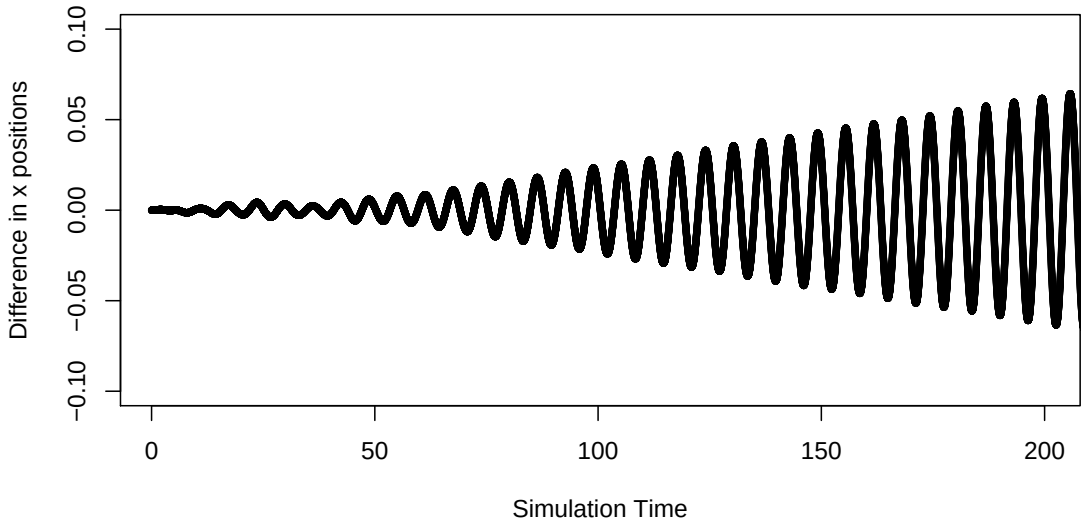

Fig. 1: Demonstration of the discrepancy in position caused by using the fast inverse square root. The graph shows the difference in the x axis position of one of the two bodies between the two simulations over time.

Results

Table 1 demonstrates the mean results of the 25 timing runs. The `rsqrtss` method is seen to be 4.7% faster. This is more in line with the figures presented in [5] and is perhaps to be expected since there are many operations other than the square root happening during each iteration of the simulation. Table 2 shows the mean position of one of the two bodies in the simulation relative to the origin. Since the orbit of the bodies is circular we expect $|\boldsymbol{r}|$ to be 0.5 throughout the simulation. In reality we must remember that the relatively large time step of the simulation as well as rounding errors in calculations do result in the orbits becoming disturbed over time. It is, however, still clear that the inverse square root method is not as effective at conserving the orbital radius - the normal square root method is seen to conserve this quantity to an extra decimal place. Figure 1 shows the difference between the x positions of body 1 in the two simulations over time. If the simulations were equally accurate, we would expect the position of body 1 to be integrated in an identical manner, but clearly this is not the case. There is a very large error in position caused by the use of the `rsqrtss` instruction which grows with time. Indeed, over time the orbit in this simulation precesses to the extent that it is on the opposite side of its orbit to what we might expect.

Conclusion

It is clear that the `rsqrtss` method is not suitable for scientific use. The errors caused by using this method over time are very large and mean that the N-body code no longer provides an accurate simulation of the actual physics. Aside from this, the 4.7% speed increase that we observed when using this method in the context of an entire N-body code is insignificant enough that even the smallest loss of accuracy would be enough to stop us using the method in a scientific code. A promising alternative to such a method which may improve on this speed increase and loss of accuracy is proposed in [6], wherein using a look-up table to seed an iterative solver results in a highly accurate and fast square root operation.

References

[1] C. Lomont, `www.lomont.org/Math/Papers/2003/InvSqrt.pdf`, (unpublished).

[2] `http://siyobik.info/main/reference/instruction/RSQRTSS` (Retrieved 20/10/2011).

[3] `http://assemblyrequired.crashworks.org/2009/10/16/timing-square-root/` (Retrieved 20/10/2011).

[4] `http://www.ottisoft.com/Activities/Two-body\%20problem.htm` (Retrieved 20/10/2011).

[5] `http://assemblyrequired.crashworks.org/2009/10/20/square-roots-in-vivo-normalizing-vectors/` (Retrieved 20/10/2011).

[6] J.A. Pineiro, J.D. Bruguera, IEEE Transactions on Computers **51**, 1377 (2002).