School of Physics and Astronomy

EPSRC Scholarship
Summer Project

An Investigation Into Particle Tracking and Simulation Algorithms Using GPUs

James Henderson
September 2010

Abstract

Over the last few years Graphical Processing Units (GPUs) have demonstrated their use in scientific calculations. They have the ability to carry out tasks in parallel, enabling great speed-ups of calculations. This project investigated any potential speed-up from GPUs when calculating a particle's trajectory through a magnetic field. The results show that a speed-up of 32x can be achieved on the GPU versus a serial processor, when considering many particles.

Declaration

I declare that this project and report is my own work.

Signature: Date:

Supervisor: Prof. Philip Clark 10 Weeks

Contents

1 Introduction

In recent years Graphical Processing Units (GPUs) have proven extremely useful in complex scientific calculations. They have been found to speed-up program times by huge amounts and improve what we can achieve in the same time. This project investigated potential speed-up of a section of the particle tracking toolkit Geant4, used to simulate particle trajectories in medical, space and high-energy physics. The section investigated was the Runge-Kutta algorithm: a numerical differential equation solver, which was ported (translated) into GPU code and any speed-up compared to a serial version of the code.

2 Background

2.1 The Runge-Kutta Method

The Runge-Kutta 4th-order (RK4) method is used to step a particle through a differential equation of motion. In this case the Lorentz equation was used to move a particle through a magnetic field.

$$\mathbf{F} = m\mathbf{a} \quad = \quad q \cdot (\mathbf{E} + \mathbf{v} \times \mathbf{B})$$

$$\mathbf{a} = \frac{d\mathbf{v}}{dt} \quad = \quad \frac{q}{m} \cdot (\mathbf{E} + \mathbf{v} \times \mathbf{B}) \tag{1}$$

Where $\mathbf{F}$ denotes the force on the particle, m - mass, q - charge, $\mathbf{E}$ - the electric field, $\mathbf{B}$ - the magnetic field and $\mathbf{v}$ - the velocity. It is worth noting this project found the trace of the particle's velocity, but it is simple to find position if we know both initial position and velocity at each step.

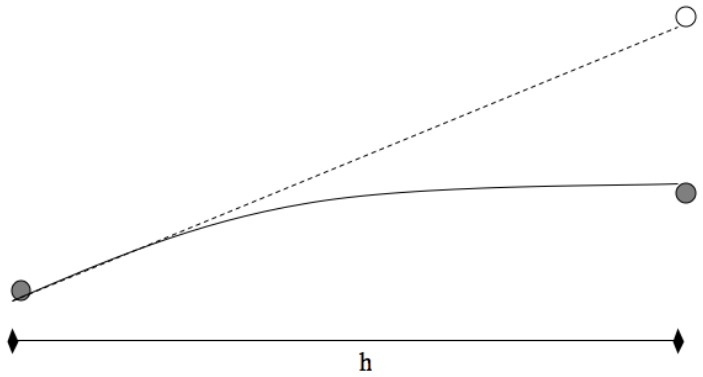

Figure 1: The simple Euler method of integration, the gradient is found at the start point and the next point is found by the equation of a straight line. The step is denoted h and the solid line shows the true trajectory, the dashed line shows the flawed calculated trajectory.

The most basic integration method is simple Euler integration, where the gradient ($\mathbf{a}$ in equation 1) is found at a point and the trajectory is modified to the next step by the

equation of a straight line. Figure 1 shows one step of the Euler method, it can be seen that the main problem is that the gradient can change over the distance of one step. To solve this problem we can simply reduce the step-size but this means we must do more calculations, which takes longer.

The RK4 method is accepted as giving the most accurate results for the least calculations. Figure 2 demonstrates how the RK4 method works.

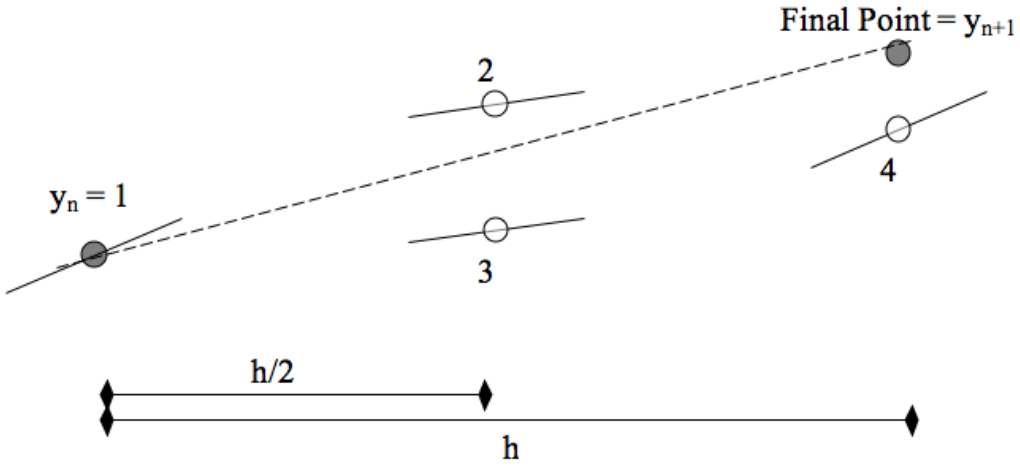

Figure 2: The Runge-Kutta 4th order method of integration. Gradients are found at 4 points and averaged over to take us to the next step. The full step is denoted h.

- The first gradient (m_1) is found at the initial point (point 1).
- Point 2 is found using simple Euler and a half step (h/2) from point 1 and gradient m_1.
- The second gradient (m_2) is now found at point 2.
- Point 3 is found using simple Euler and a half step (h/2) from point 1 and gradient m_2.
- m_3 is found at point 3.
- Point 4 is found using simple Euler and a full step (h) from point 1 and gradient m_3.
- m_4 is found at point 4.
- A weighted averaged of the gradients is taken and the final point is found by Euler from point 1, see equation 2 where the gradients are denoted as ms.

$$y_{n+1} = y_n + h \cdot (m_1 + 2m_2 + 2m_3 + m_4) \tag{2}$$

2.1.1 Error Analysis

However, the problem remains from the Euler method where the step size may be too large and miss some change in the gradient. To cover that problem, the RK4 method does error analysis that calculates the y_{n+1} value in one RK4 evaluation and then again in two evaluations at half the step size, see Figure 3. The two results for y_{n+1} are compared and if similar enough the step is accepted, if not the step-size is decreased and the step re-evaluated.

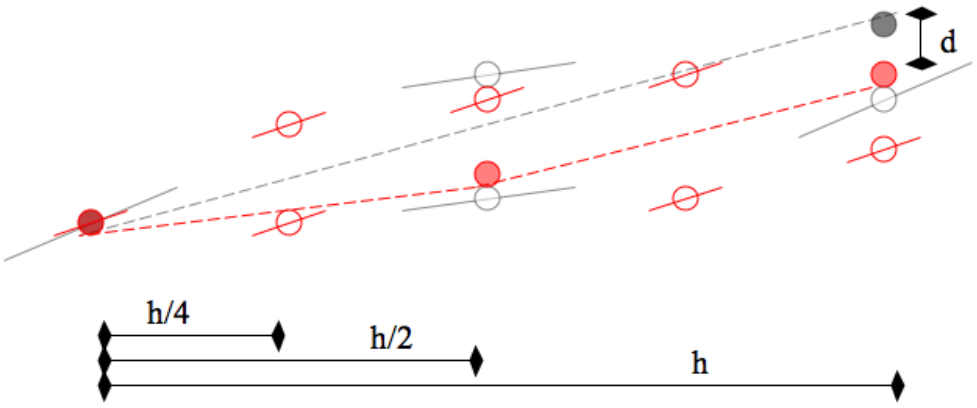

Figure 3: The error analysis of RK4. The full step is done as one full evaluation (faded black) and then again as two half evaluations (red). The difference in the y direction is shown as d.

2.2 Parallel Computing

Graphical Processing Units (GPUs) were designed to process data for many display pixels at the same time. They are an array of many thousands of simple processors all designed to carry out simple calculations at the same time (in parallel). This contrasts with the Central Processing Unit (CPU) who's job it is to carry out complex tasks in serial. Until recently CPU manufacturers have been able to increase clock speeds on CPUs and hence instructions are executed faster and programs run faster. CPU development has decelerated due to limits with clock speed: overheating and data transport speed, so program developers are looking towards carrying out tasks in parallel to speed them up. This is more complex than simply increasing clock speed as the programs must be re-written to work on the GPUs.

3 Method

The RK4 algorithm is an inherently serial process; the calculation of a step requires us to know where we are starting from so we must have done the preceding step first. Hence it was not possible to simply calculate all the steps independently on the GPU and fix them all together later. Therefore, it was decided that the best way to speed-up the

trajectory tracking process was to port the error analysis section to GPU as well as doing each component (x, y and z) in parallel.

To port the error analysis, the one full evaluation was carried out in parallel with one of the half evaluations. Then the other half evaluation was done, meaning that where the serial code took the time of 3 evaluations, the GPU code took 2. Evaluating the components in parallel meant the GPU was 3 times faster in theory, than the CPU. Programs were written in CUDA - a GPU language designed by nVidia, and C++ - to run in serial and compare with the CUDA run times.

4 Results & Discussion

As the GPU is physically located away from the CPU, data must be transferred between the two devices. This meant that the data transfer overheads made the CPU anywhere up to 70x faster, depending on the particle setup, than the GPU when tracking a single particle. It was however clear that the GPU was significantly faster at calculating the trajectory when it did have access to the data. This suggested that for multiple particles the GPU would be faster which was indeed the case.

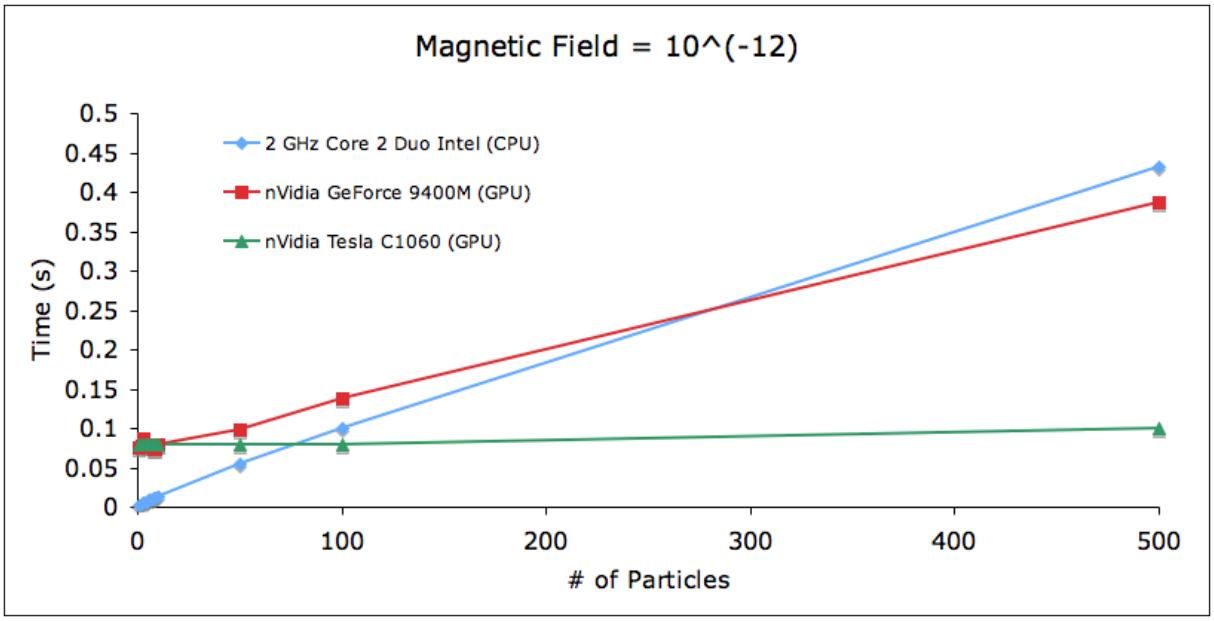

Figure 4: The timing results from magnetic field = 1 $\times 10^{-12}$. The CPU time increases linearly whereas the GPUs become relatively faster with the number of particles.

In both Figures 4, 5 it can be seen that the CPU times increase linearly from zero as they have no transfer overhead, whereas the GPU times have a minimum value due to this transfer. The nVidia GeForce 9400M appears to increase linearly, especially in Figure 4, this is probably as it is a very small GPU, so has a limited number of processors that will be saturated early on.

The maximum number of particles investigated was 50,000, at which point the GPUs showed a speed-up of 32x over the CPU, under certain particle conditions.

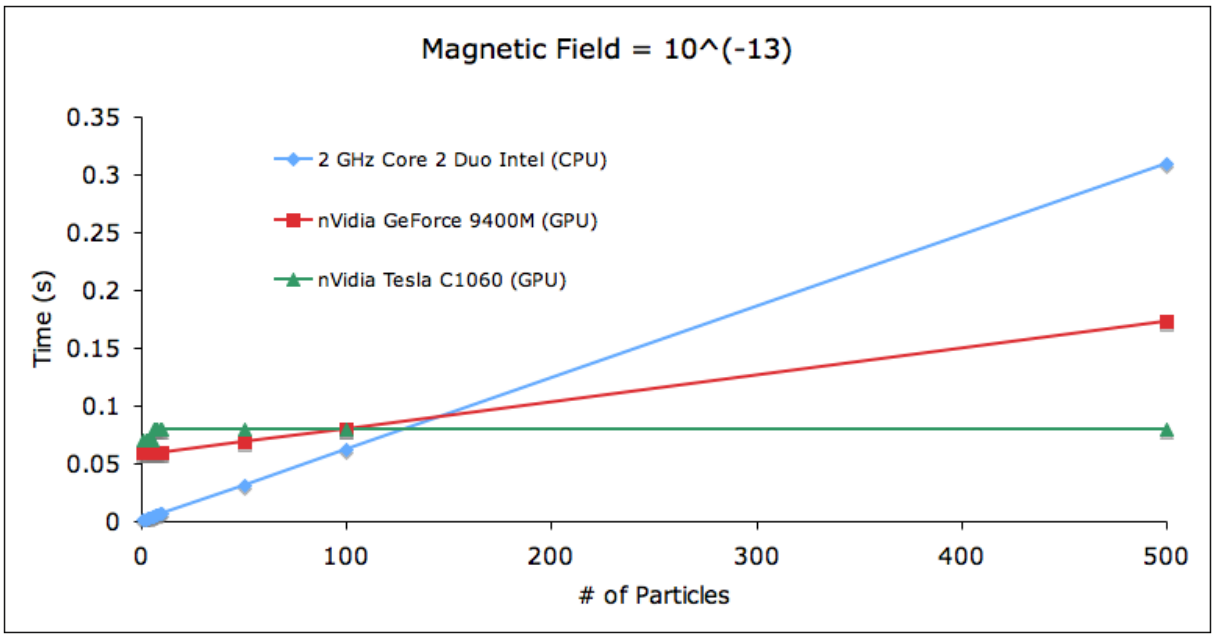

Figure 5: The timing results from magnetic field = 1 $\times 10^{-13}$. Again the GPUs become relatively faster with particle number

5 Conclusion

The GPU has again proven its worth in detailed scientific tasks and has provided a speed-up of 32 times under certain conditions, over the serial CPU. This could be improved by an experienced programmer as the author has very limited experience in the field of GPU programming. Yet even so, this report has shown that with a little research and time even an inexperienced programmer can achieve 32x speed-up of a very relevant piece of code.

6 Acknowledgements

Thanks to Professor Philip Clark with orientation of the theory and a great deal of assistance through out the course of the project. Thanks also to Andy Washbrook for help with many struggles with the computers and to John Apostolakis for guidance on the project direction.