Abstract

Simulating particle collisions is an important application for physics calculations as well as for various effects in computer games and movie animations. Increasing demand for physical correctness and hence visual realism demands higher order time-integration methods and more sophisticated collision management algorithms. We report on the use of single and multiple Graphical Processing Units (GPUs) to accelerate these calculations. We explore the performance of multiple GPUs (m-GPUs) housed on a single PCIe bus as well as the use of special purpose PCIe bus extender technology using GPU housing chassis systems such as Dell’s C410x PowerEdge. We describe how a hard sphere collision system with gravitational interactions was developed as a benchmark. We compare the performance of various GPU models and show how algorithms that use GPU-GPU communications with NVIDIA’s Compute Device Unified Architecture (CUDA 4) can considerably aid communications amongst multiple GPUs working on a single simulated particle system.

Keywords: hard-sphere collisions; GPU; GPU-GPU communication; CUDA 4; cluster; PCIe bus.

1 Introduction

Particle simulation is a technique used heavily in the computer games industry and also for constructing animation of sophisticated computer generated scene effects in the movie industry. Traditionally some rather poor approximations to the physics have been used in these applications to save on computational requirements and in many circumstances these are not noticed by the viewer or player.

We are interested in software for “physics engines” (Bourg 2002, Conger & LaMothe 2004) that make better approximations to the point of being able to explore the statistical mechanical behaviours or numerical experiments based on particles. We are therefore interested in high quality physics engines that might ultimately be used as games engines (Thorn 2011, Millington 2007, Gregory 2009, Menard 2011) as well – providing sufficient real time performance can be achieved.

Graphical Processing Units (GPUs) have found many widespread recent uses in accelerating the performance of many scientific and simulation calculations. GPU commodity pricing and ubiquity means they are widely available in many games computers, but they are also finding sophisticated uses in supercomputers and indeed many of the world current top supercomputers (Meuer, Strohmaier, Simon & Dongarra 2010) employ GPU technology as general purpose “GPGPU” accelerators - to speed up calculations and not just graphical rendering (Wright, Haenel, Sellers & Lipchak 2011).

GPUs come in a number of different models with different numbers of cores; floating point capabilities and often very importantly different levels and amounts of memory (Leist, Playne & Hawick 2009, Playne & Hawick 2011). We have experimented with a number of individual GPU devices ranging from low priced game rendering models to top end gamer de-
vices such as the GTX 580/590 series and to blade quality devices such as the C2050/C2070 series.

GPUs provide a powerful data-parallel architecture for classical N-body particle dynamics simulations which have in fact been used as a benchmark application for the hardware devices (Hawick, Playne & Johnson 2011, Playne, Johnson & Hawick 2009, Nyland, Harris & Prins 2007, Stock & Garraghan 2008, Kavanagh, Lewis & Massingill 2008). We extend this idea to the simulation of hard spheres under the influence of a gravitational field. The use of hard-sphere collisions (Allen & Tildesley 1987) provides a communications challenge involving correct book keeping management of collisions as well as the computational challenge of performing accurate numerical integration (Hawick et al. 2011) of the classical mechanical equations of motion for an N-body system. We are interested in large and complex systems where there are many particles and interactions, but the system density gives us a parameter to vary along with the number of processing cores to use this application as a benchmark for exploring performance tradeoffs of modern GPUs.

An attractive and relatively recent development involves the use of multiple GPUs that all cooperate to support and accelerate the performance of a single CPU. Nvidia’s Compute Unified Device Architecture (CUDA) – version 4 – offers software capabilities to manage direct communications between such cooperating GPUs and without passing data via the controlling CPU.

Graphical Processing Units (GPUs) are generally connected to their hosting processors via a Peripheral Component Interconnect Express (PCIe) bus (PCI-SIG 2010) and usually motherboards will support at most four PCIe devices. Although known as a bus, the PCIe standard is actually structured as point-to-point serial links with dynamic bandwidth negotiation. We construct a synthetic benchmark application to measure the bandwidth, latency and PCIe bus contention issues that arise as a multi-threaded CPU host program delegates work to m-GPU accelerator devices. We explore performance properties of different models of GPUs as well as that of PCIe extender cards and device chassis that support operation of more than four GPU devices per CPU. In addition to benchmark data we discuss applications and appropriate software and threading architectures to make good use of GPUs and GPU-accelerated clusters configured in this manner.

We use this application as a benchmark for a m-GPU system built using Dell’s C410x PowerEdge chassis (Dell Inc. 2010) for extending the PCIe bus and compare the performance of various GPU/cluster combinations. We discuss how the compute to communications ratio of an application that involves hard inelastic collisions differs from the simple N-body particle dynamics as a data-parallel benchmark.

Figure 1 shows a rendering of a simulation of several thousand colliding hard spherical particles with a density of around 0.025. This is in fact quite a high value that gives rise to many collisions in each simulated time unit. In this paper we explore how the number of particles in the system can be increased using multiple GPUs, but also how the performance changes as we increase the particle system density - and hence change the communications to computation ratio.

This paper is organised as follows. We describe the algorithms for approximating the mechanics of interacting hard core spherical particles in Section 2.

In Section 3 we describe the GPU configurations we employed and our GPU code implementations using CUDA. We present some performance results for various GPUs at different numbers and densities of simulated particles in Section 5 and discuss the scalability and implications for combining GPUs together in Section 6. We offer some conclusions and areas for further work in Section 7.

2 Hard Core Collisions & Interactions

In addition to game physics engine applications (Eberly 2004), a number of problems in chemistry and physics can be formulated in terms of interacting hard-core bodies. A hard-core body in this context simply means a rigid body that cannot be deformed beyond a certain point. Spheres are particularly useful for many models since they are very easily parameterised in terms of a position (of the centre) and a radius. Spheres can be rendered in a 3-D space with shading or false colour or texture maps and can be used to approximate planetary dynamical systems or simple molecular models.

The packing density is essentially a measure of the wasted space when you pack a number of solids into a (large) box. A limiting fraction (between 0 and 1) gives a universal measure of this for different shapes. There are some important physical and chemical properties of various materials that relate to the packing density of their component molecules. The volume of a single sphere of radius \(a\) is just \(\frac{4}{3}\pi a^3\) so if there are \(N\) non-overlapping spheres in a rectilinear box of Volume \(V\), it is straightforward to relate the radius to the density \(\rho\) of the simulated system.

A number of authors have reported on well-known approaches to parallelising the N-body particle dynamical problem. Approaches include use of space dividing oct-trees (Warren & Salmon 1993, Barnes & Hut 1986) to allow the \(O(N^2)\) computation to be reduced to \(O(N \log N)\) and data parallel computers have been applied successfully to this sort of problem for some years (Brunet, Mesirov & Edelman 1990, G.Fox, M.Johnson, G.Lyzenga, S.Otto, J.Salomon & D.Walker 1988, Playne 2008). A common approach is to divide the particle up amongst processing units and employ a one-dimensional communicating ring approach to allow all processors to access all (possibly reduced) information on the particles they do not have direct responsibility for updating.

Vector parallel techniques have also been successfully employed for updating collision lists of hard disks and hard spheres (Allen & Tildesley 1987, Donev, Torquato & Stillinger 2005). In this present paper we employ a hybrid approach since we want hard core particles that cannot intersect but also wish to apply high-order numerical integration methods to accurately track the changing trajectories of particles under the influence of gravity.

Our benchmark code is aimed at simulating the behaviour of interacting polydisperse hard-core spheres. The code computes collision dynamics for a system of \(N\) hard spheres, which are contained in a periodically repeating cell of unit edge length. The spheres, labelled by index \(i\) have diameters \(a_i\), and masses \(m_i\). Their positions \(\mathbf{r}_i\), velocities \(\mathbf{v}_i = \frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t}\) and accelerations \(\frac{\mathrm{d}^2\mathbf{r}_i}{\mathrm{d}t^2}\) are tracked during the simulation. Hard spheres that only interact via inelastic collisions do not need accelerations to be recorded. Accelerations are however required and computed when a global
gravitational field is applied. To make this physically meaningful we apply fixed boundaries for the roof and floor of the simulated box, but retain periodicity in the horizontal dimensions.

We consider a set of $N$ particles, labelled $i = 0, 1, 2, ..., N - 1$ that interact via pair-wise interactions that are dependent on various properties of the particles. Central forces that depend solely upon the relative distance between particles $i$ and $j$. For example the Newtonian gravitational potential arising on the $i$th particle from the $j$th particle, $V(r_{ij})$, can be written as:

$$V_{G,i,j}(r_{ij}) = -\frac{Gm_i m_j}{r_{ij}}$$  

(1)

An alternative pair-wise force law that might model chemical van der Waal’s force or some other form of long range attraction between particles $(i,j)$ which can be approximated using a Lennard-Jones potential as:

$$V_{L-j}(r_{ij}) = 4\epsilon \left( \left( \frac{\sigma}{r_{ij}} \right)^{12} - \left( \frac{\sigma}{r_{ij}} \right)^6 \right)$$  

(2)

We can also add in a fixed extra term that depends solely on height $V_g(y) = m_i g |y|$ to model an overall gravitational field so that all particles tend to drift downwards within the simulated box – this can be incorporated into the total $V(r)$. The classical (Newtonian mechanical) force can then be written as the gradient of the potential:

$$F = \nabla V(r)$$  

(3)

For centralised forces like gravitational systems, we can simply sum pair-wise forces along a vector connecting the particle centres, and for a single such axis the gradient is just a single-variable derivative and hence:

$$F_i = \sum_j G m_i m_j r_{i,j}$$  

(4)

Given Newton’s third law: $F_i = m_i a_i = \Rightarrow a_i = F_i/m_i$ and we can employ the separate $x, y, z$ components of the acceleration in the Newtonian classical rigid body equations of motion so that we compute changes in particle $i$’s velocity and position.

We consider the possibility that particles interact over long ranges via such a force law, but that they also have some minimum distance of separation $2a$ so that they are hard-cored particles and cannot overlap.

Figure 2 shows the problem of two hard spheres that collide. The particles are hard spheres, and collide inelastically - they do not interact except for an impulse applied at the point of collision. However both do experience the externally applied gravitational field force and particle A which has a horizontal initial velocity follows a parabolic trajectory. The time and point of collision with particle B must be determined to correctly apply the collision behaviour.

Our model is summarised in Algorithm 1.

### Algorithm 1

1. initialise $N$ particles in 3D
2. for $t \leftarrow 0$ to $T_{max}$ do
   1. compute gravitational and pair-wise force sums on each particle
   2. compute particle accelerations from forces
   3. time integrate all particles by $h$
   4. check for penetrating core collisions
   while collision occurred do
      1. warp time back to earliest collision, undoing core penetrations
      2. compute impulses for collision, correcting velocities
      3. check for penetrating core collisions
   end while
3. resume time integration

In effect then, the inelastic hard-core collisions are impulsive corrections to the time integration of the usual N-body Newtonian mechanics. We can use any time-integration algorithm we wish, but higher-order will yield better energy conservation. The inelastic collisions with box boundaries would change the overall energy and the simulated system would approach an equilibrium temperature given the particle core radii. In this present paper we focus on the GPU performance and benchmarking aspects and do not explore these statistical physics effects further.

## 3 GPU Implementation

Understanding the scaling and communications performance of multi-GPU systems (Spampinato 2009) is becoming very important as large scale supercomputers that use them become more prevalent – as evidenced by those on the present Top 500 world list of supercomputers (TOP500.org n.d., Meuer et al. 2010).

A key aspect to understanding their behaviour is the scalability of the Peripheral Component Interconnect Express (PCIe) bus (PCI-SIG 2010) used to communicate between GPUs and CPU. PCIe is a sophisticated technology – it is a point-to-point serial structure with lanes, dynamic negotiation and has developed through a number of versions. It is implemented by a number of different vendors on different GPU models and boards.

Typically a PC motherboard is limited to having at most four PCIe slots into which GPU cards can be located. In some cases this is further limited by power requirements and physical geometry of the slots. Dell and other manufacturers are now making available a number of PCIe bus extenders in the form of an integrated chassis that allows various software controlled configurations of devices. The Dell C410x PowerEdge chassis (Dell Inc. 2010) we discuss in this present paper houses up to 16 GPUs each in its own bay, and provides power and cooling suitable for a machine room environment. This is a good deployment platform to experiment with blade-level GPUs, although much of our earlier work was successfully carried out using the very much cheaper gamer-quality GPUs that do not have error-corrected memory.
3.1 Single-GPU Implementation

This benchmark makes use of the well-known tiling algorithm to compute the all-pairs gravitational forces between particles (Nyland et al. 2007, Playne et al. 2009). This algorithm processing the particles in tiles stored in shared memory. Each block of threads will load one tile of particles at a time into shared memory and each thread computes the force of gravity exerted by each of those particles on the single particle that threads is responsible for.

A similar algorithm can be used to detect collisions between particles. Each pair of particles must be compared to determine if they have collided during the previous time step. If a collision has occurred, the time at which it happened is calculated and saved. If a thread’s particle is involved in multiple collisions, it will record the time of the first collision and the index of the particle it collided with. The threads use atomic operations to compute the time at which first collision in the entire system occurred. The kernel used to detect these collisions is shown in Algorithm 2. This algorithm will determine the first collision particle $i$ was involved in and will be computed for each particle by a separate thread. Each thread will compute $\text{max}\_\text{times}[i]$ which is how long ago the first collision of that particle occurred.

Algorithm 2

\begin{verbatim}
collision detect kernel
  p1 ← particles[i]
  for j ← 0 to N do
    p2 ← particles[j]
    d ← distance from p1 to p2
    if d < p1.R + p2.R then
      t ← required step back
      if t > max\_times[i] then
        max\_times[i] = t
        collision\_index[i] = j
      end if
    end if
  end for
end if
end if
\end{verbatim}

To ensure all collisions are processed correctly, they are resolved one at a time. Once the time of the first collision is determined, a kernel is launched which will test to see if the particle was involved in the first collision. If the thread’s particle was not involved in the collision, the kernel will immediately return, this can be determined by comparing that particle’s collision time with the time of the first collision. If the thread’s particle was involved in the collision it will catch the index of the other particle involved in the collision. The thread with the lowest index will step both particles back in time to the point of the collision, perform the collision and step both particles back to the current system time. The kernel to perform this collision process is shown in Algorithm 3.

Algorithm 3

\begin{verbatim}
collide particles kernel
  if max\_times[i] == system\_max\_time then
    j ← collision\_index[i]
    if i < j then
      p1 ← particles[i]
      p2 ← particles[j]
      step particles p1 and p2 back by system\_max\_time
      collide particles p1 and p2
      step particles p1 and p2 forward by system\_max\_time
    end if
  end if
\end{verbatim}

This process of detecting the first collision and resolving it must be performed iteratively until no more collisions occur. The collision detection kernel will return 0 when no collisions have occurred. These kernels, along with the force calculation and integration kernels can be used to compute an N-body simulation with collisions on a single GPU. However, utilising multiple GPUs is somewhat more difficult.

Algorithm 4 shows the basic algorithm for computing the total force on each particle. Initially a kernel is called to compute the forces the particles on the device exert on each other. Once this has been completed the device will loop through all the other device. For each iteration of the loop, the device will copy the particle data out of the other device into its memory. It then computes the total force those particles exert on its particles.

Figure 3: A diagram comparing GPUDirect communication and memory transfer through the host. The dotted lines show the transfer of the data through host memory.

3.2 M-GPU Implementation

The main challenge of computing an N-body simulation on an m-GPU system is managing the communication between them. This m-GPU implementation evenly distributes $N$ particles between $P$ devices such that each GPU device is responsible for $\frac{N}{P}$ particles. For each device to compute the total force on each of its particles, it must copy the particle data out of the other GPUs. Likewise to detect if any collision have occurred, each device must compare each of its particle with each other as well as the particles stored on other GPUs. Including m-GPU communication in the benchmark allows the test system to be evaluated in terms of computation as well as communication throughput.

This implementation makes use of the CUDA 4.0 functionality of peer-to-peer memory transfer. GPUDirect 2.0 allows data to be copied directly from one GPU device into another across the PCI-e bus. Without GPUDirect 2.0, data had to be first copied out of the device to host memory and then into the second device. This peer-to-peer communication can significantly improve the performance of m-GPU applications.

Algorithm 4 shows the basic algorithm for computing the total force on each particle. Initially a kernel is called to compute the forces the particles on the device exert on each other. Once this has been completed the device will loop through all the other device. For each iteration of the loop, the device will copy the particle data out of the other device into its memory. It then computes the total force those particles exert on its particles.
Algorithm 4

call compute force kernel
for d ← 0 to num_devices do
if i ≠ d then
  copy particles from device_d to device_i
end if
end for

time integrate all particles by h

A similar algorithm is used to detect particle collisions. Initially a kernel is called to detect collisions of particles on the same device. Once this is completed the particles on other devices are checked for collisions. Like the force calculation, the devices are iterated through and their particles copied to the current device. These particles are then checked for collisions with this device’s particles. This process is presented in pseudo code in Algorithm 5

Algorithm 5

max_time_i ← 0

copy max_time_i into device_i
call collision detect kernel
for d ← 0 to num_devices do
if i ≠ d then
  copy particles from device_d to device_i
call collision detect kernel
end if
end for
copy max_time_i out of device_i

To determine when the first overall collision occurred, the first collision times for each device must be compared. The time of the first collision is found and the index of the device is recorded. If the collision occurred between two particles on the same device, the same kernel as in Algorithm 6 will be launched. If the collision occurred between particles on different devices, the appropriate data will be exchanged and one kernel on each device will be launched. This kernel will update the particle on the device that was involved in the collision. The process for determining the time of the first collision and computing the collision is shown in Algorithm 6.

This basic algorithm can be used to implement an N-body simulation with hard sphere collision on a m-GPU system. The main point of difference between the implementations is how the data transfer is performed. This includes both removing redundant communication as well as different methods of implementing communication between devices.

When collisions are detected and computed, the devices must exchange particle data. If multiple collisions occur within a single time step, the devices must exchange this data multiple times. However, most of the particle data will remain unchanged. Only the data about the particles that were involved in the collision needs to be propagated to the other devices. This is not a change in the fundamental algorithm, merely a reduction in the data that is communicated.

The more important difference in implementations is the method of CUDA communication. Fermi architecture GPUs have support for GPUDirect 2.0 which allows data to be directly communicated between devices. This is the preferred communication method used by the benchmark. However, Tesla architecture GPUs do not support this functionality and any data transfer must be communicated through the host memory.

4 Benchmarking

The benchmark has been used to evaluate a number of Fermi architecture GPUs in a variety of configurations. The testing has been focused on high-performance graphics cards and compares the gamer level GTX480 and GTX580 with the professional C2050 and C2070 GPUs. Both the single-GPU and m-GPU implementations have been executed on these cards to compare their performance. The C2050 and...
C2070 cards have been tested both hosted on a traditional motherboard and on the Dell C410x chassis.

To compare the performance of the different GPU configurations, the simulations have been run with three different system configurations which varies the computation between the highly parallel force computation and the more restricted collision detection. A particle configuration with very low density is very computationally similar to the benchmark without particle collisions as the computation is almost entirely parallel. Whereas particle configuration with a high density will have a great deal more collisions and requires more communication between devices to resolve these collisions.

To test the difference in performance based on particle density, three different initialisation configurations have been compared. The first initialises the particle positions and velocities randomly with a very low density. This configuration results in almost no collisions and purely tests the parallel processing performance. The second initialises the particle positions in a two-dimensional lattice with random velocities, this results in a medium density simulation with a collision rate of ≈0.03 collisions per particle per step. The final configuration initialises the system with the particles laid out in three-dimensional lattice with random velocities. In this configuration there are a higher number of collisions, the rate is ≈0.06 per particle per step. The number of collisions vs system size is shown in Figure 6.

5 Results

We present some selected performance timing data for the hard core gravitational particle simulation for a number of GPU devices in both single GPU and m-GPU configurations. These configurations are evaluated for a number of different system sizes but also with the three different particle densities discussed in the previous section - low, medium and high.

The first case is the dilute limit of density where no collisions occur during the period of the benchmark, effectively these systems have a collision per particle per step ratio of 0.00. This test will show the best possible performance as the computation can be executed entirely in parallel. Collisions are still detected but as they never occur then there is no serial process required to resolve them. The single-GPU implementation has been tested on the GTX480, GTX580, C2050 and C2070. The multi-GPU implementation has been tested on 2xGTX480, 2xGTX580, 2xC2050, 2xC2070, 4xC2050 and 4xC2070 configurations. The performance plot of this benchmark is shown in Figure 7.

From this plot it can be seen that the GTX580, C2050 and C2070 GPUs offer almost indistinguishable performance while the GTX480 performs significantly slower. However, for the m-GPU implementation the 2xC2050 and 2xC2070 configurations perform faster than the 2xGTX480 and 2xGTX580 systems.

The medium particle density systems (≈0.03 collisions per particle per step) shows an interesting change in single-GPU performance. For this test the gamer level GTX480 and GTX580 both performed faster than the C2050 and C2070 cards. However, once again the 2xC2050 and 2xC2070 both showed higher performance for the m-GPU implementations. The performance plots the medium density configurations (≈0.03) on the different devices can be seen in Figure 8.

The high particle density (≈0.06) configuration shows very similar results to the medium density (≈0.03) systems. The GTX480 and GTX580 cards both offer the best single-GPU performance but the Tesla compute cards provide the best m-GPU performance. These results suggest that the C2050 and C2070 Tesla compute cards have faster communication than the GTX480 and GTX580 graphics cards. NVidia datasheet specifies that Tesla computing GPUs support faster PCIe communication, evidenced by these findings (?). The performance results for the high-density configurations are shown in Figure 9.

The performance results for the C2050 and C2070 configuration shown in this section have been hosted on a Dell C410x chassis. This configuration has been compared to a configuration hosting the same GPUs on a traditional motherboard. The results show no measurable difference between the two configurations.
Figure 7: Comparison of GPU configurations computing N-body simulations initialised at the dilute limit of density. Results are shown for system sizes in the range $N = \{1024, 2048 \cdots 65536\}$ in normal scale (left) and ln-ln scale (right).

Figure 8: Timing data comparing GPU devices computing medium density configuration initialised in a two-dimensional lattice with random velocities. Results are shown for system sizes in the range $N = \{1024, 2048 \cdots 65536\}$ in normal scale (left) and ln-ln scale (right).

Figure 9: Timing data for single and m-GPU implementations on various devices with high density configurations initialised in a three-dimensional lattice with random velocities. Results are shown for system sizes in the range $N = \{1024, 2048 \cdots 65536\}$ in normal scale (left) and ln-ln scale (right).
for single or m-GPU implementations and thus have not been presented separately. For this benchmark, hosting the devices on a C410x does not degrade performance.

6 Discussion

Including hard-sphere particle collisions into the N-body benchmark allows greater insights into the performance of different GPU configurations. The gamer-level GeForce cards provided similar or slower performance compared to the Tesla compute cards for low density configurations where collisions almost never occurred. However, for medium and higher density systems the GeForce cards both provided significant performance benefits over the Tesla compute cards.

However, for m-GPU implementations where device-device communication is required, the results showed the opposite. The Tesla compute cards provide a significant performance benefit as compared to the GeForce graphics cards, this performance difference is attributed to the Tesla compute cards’ improved PCIe communication capabilities (7). This difference in performance was more pronounced for configuration medium and high particle densities as more communication is required to resolve collisions.

The Tesla compute cards are significantly more expensive than the consumer-level GeForce graphics cards yet for single-GPU applications where ECC is not required, the GeForce cards provide comparable or improved performance. However, the Tesla compute cards are the only GPUs that support ECC and for m-GPU implementations they provide significantly higher performance. We believe this is due to the better DMA transfer capabilities of the Tesla over GTX cards.

These Tesla compute cards can be hosted in PCIe extender chassis’s such as Dell’s C410x. For the single-GPU and m-GPU implementations tested in this research, devices hosted on this chassis provided performance that was indistinguishable from the same cards hosted on a traditional motherboard. This shows that the chassis does not degrade performance in any way, as was initially feared.

The m-GPU implementation was only tested with four devices on this chassis. Host machines are now available which can host multiple HCI cards to allow a single host to connect to up to eight devices hosted on the C410x. However, we do not currently own such a host machines and cannot currently test the performance of such a configuration.

7 Conclusions

We have described how features of NVidia’s latest CUDA release aid the performance of this benchmark – in particular those that support direct GPU-to-GPU communications without passing through CPU code. We have also explored the performance capabilities of commodity priced gamer level GPU cards as well as significantly more expensive blade-quality production cards. We found that the gamer-level cards were better for an m-GPU approach, and that this appears to be due to their enhanced ability to communicate rather than their floating point performance.

We have shown that the PCIe extender bus approach works well and without significant loss of performance for the regimes we have been able to explore. We expect performance to degrade with bus contention as more GPUs are added and we plan to explore this further as more hardware becomes available. We also anticipate availability of further improved GPU models that may have even better floating point performance and communications abilities than those models available to us.

There are open areas of computational physics such as the phase separation of polydisperse particles that can potentially be explored through fast simulations such as we describe. We have experimented with simple spherical particle collisions but there is scope for important work parallelising other and more general rigid body collision algorithms on data parallel architectures such as GPUs. Collision detection and particle dynamics continue to be important algorithms deployed in computer games and we anticipate applications such as we have described as becoming even more important as it becomes standard practice for “gamer computers” to have multiple GPUs available for performance acceleration.

8 Acknowledgments

It is with great pleasure the authors note their thanks to Arno Leist for technical assistance in configuring the C410x GPU chassis.

References
