Technical Report CSTN-139

Classical Mechanical Hard-Core Particles Simulated in a Rigid Enclosure using Multi-GPU Systems

D.P.Playne and K.A. Hawick
Computer Science, Institute for Information and Mathematical Sciences,
Massey University, North Shore 102-904, Auckland, New Zealand
email: { d.p.playne, k.a.hawick }@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181

March 2012

ABSTRACT
Hard-core interacting particle methods are of increasing importance for simulations and game applications as well as a tool supporting animations. We develop a high accuracy numerical integration technique for managing hard-core colliding particles of various physical properties such as differing interaction species and hard-core radii using multiple Graphical Processing Unit (m-GPU) computing techniques. We report on the performance tradeoffs between communications and computations for various model parameters and for a range of individual GPU models and multiple-GPU combinations. We explore uses of the GPU Direct communications mechanisms between multiple GPUs accelerating the same CPU host and show that m-GPU multi-level parallelism is a powerful approach for complex N-Body simulations that will deploy well on commodity systems.

KEY WORDS
m-GPU; GPUDirect; N-Body; hard-core collisions; polydisperse radii; multi-species.

1 Introduction

Models based upon classical N-Body particle systems [2] are commonly used at various levels of approximation in game simulations [5,11,12] but still play an important role in understanding physical phenomena such as diffusion, phase mixing and separation [4], and other behaviours that arise from specific geometric distributions.

Considerable work has been reported in the literature on uses of molecular dynamics whereby approximate potential models are used to simulate atomic and molecular systems [14, 30, 36]. A recent review by Larsson and co-workers [26] points out that there is still considerable scope for improved algorithms and for hybrid solutions to the molecular dynamics N-Body applications problem.

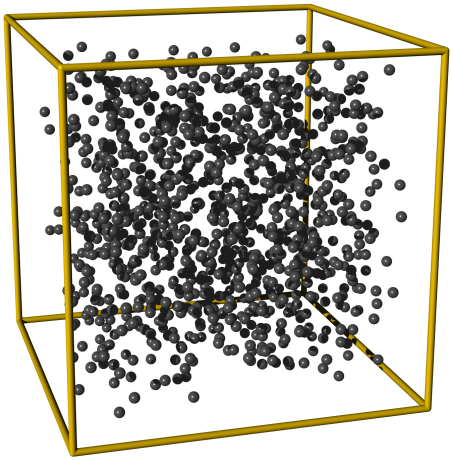

Figure 1: Three-dimensional particle system with rigid walls.

Another N-Body application area of significance is in simulating astrophysical phenomena including: general relativity systems [44]; cosmological simulations [37]; simulations related to dark matter theories [25]; and other modifications to Newtonian gravity [39]. A body of recent important work is progressing in this field where simulation provides an important means of exploring the implications of various theories of dark matter.

It is therefore of continuing importance to understand the computational performance for N-Body particle system simulations and indeed such simulations have

found use as benchmark kernels for high-performance computing systems [8, 19, 20, 23, 29, 33, 38].

In the limit of a large number of particles and a thermodynamically equilibrated system a simple benchmark rating such as number of particle updates per second suffices, but in practice, many gaming situations in which particle models are employed the system is quite definitely not in equilibrium. Much present research into understanding growth and transient behaviour in physical, chemical and biological systems is also concerned with models and systems that are far from equilibrium.

It is therefore interesting to add to the parametric space of a particle integration benchmark by considering how the load balance and computational organisation can be changed by varying a manageable parameter such as the temperature of a simulated system.

We consider a system of hard spherical particles that interact at long and medium range through an attractive potential with a soft repulsive core, but with a hard impenetrable core at very short range. This is a realistic model for many scenarios and allows us to experiment with a range of computational intensities in the resulting particle benchmark. The system is also usefully realistic – it can be modelled with a definite walled container and not with the unrealistic periodic boundaries often used in physics model simulations used for studying statistical mechanical properties.

Two typical paradigms for deploying N-Body simulation codes [34] are as either near teal-time interactive codes that have a built in visualisation capability, or as batch-oriented non-interactive codes, that may offer a frame dumping capability but which are not intended nor capable of rendering in real time.

There are a number of sophisticated data organisational methods that approximate the $\mathcal{O}(N^2)$ interactions with appropriate potential cut-offs which can be managed as spatial trees and neighbour lists [3, 41]. Some of these approaches can be optimised for particular architectures such as hypercubic systems [6, 15]. Collision dynamics can also be optimised using appropriate collision lists and associated techniques [10].

The size of systems that are feasible to simulate obviously depends on the desired simulated time period as well as most critically – upon the number of particles involved. At the time of writing, work has been reported on up to $N \approx 10^10$ possible particles in a gravitational simulation involving collisional and collisionless systems [7]. Articles such as that by Aarseth report work with more complex models such as post Newtonian corrections [1] with $N \approx 10^5$ particles simulated on on GRAPE [22] type computer architectures.

Various architectural approaches have been applied to this important class of application problems, including clusters and volunteer distributed computing systems [9] and conventional multi-core CPU systems modern multicore CPU systems [40]. Work by Groen and co-workers [16] indicates the typical number of cores (60-750) that have been recently feasibly employed in multicore cluster systems to tackle a single N-Body application simulation cold dark matter.

In this present paper we are interested in the capabilities of graphical processing units and systems deploying single and multiple GPU systems and several authors and research groups have reported work on N-Body calculations on GPUs – although nearly always on a very specific algorithm [14, 17, 21, 42, 43].

A considerable range of sophisticated physics quality simulation codes for specific and special purpose N-Body applications are available and have been compared in the review by Fortin and co-workers [13].

We are interested in the computational structure of a hybrid N-Body application that involves different sorts of short or long range potentials, hard core collisions and also interactions with enclosing rigid walls of a container. This covers the principal algorithms needed for both games quality physics simulations as well as more sophisticated energy conserving statistical mechanics studies. We are also interested in being able to deploy very accurate high-order numerical integration methods such as the Hairer 10th order time stepping algorithm [18].

In this paper we explore how a sophisticated N-Body simulation can be developed for multiple GPU systems with the option of choosing collisional or collisionless systems, and parameterising the model using temperature T as well as the number of individual particles N.

2 Distribution of Speeds

It becomes feasible and realistic to ascribe a temperature to an ensemble of simulated particles once the system size N is large enough. In practice a few thousand particles yields thermal measurements that can be compared with the theoretical definition discussed below.

The scalar speed is defined in terms of the velocity

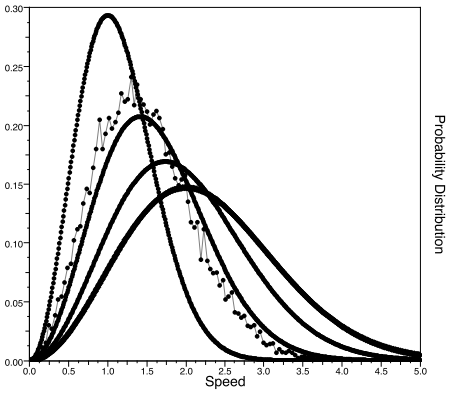

Figure 2: Theoretical Maxwell-Boltzmann distribution computed for Temperatures 1,2,3,4 with $k_B = m \equiv 1$ from equation 2 showing increasing most-probable speed and distribution width with increasing temperature. Experimental data from a sample simulation is included for reference.

components for each particle as:

$$v = \sqrt{(}v_x^2 + v_y^2 + v_z^2) \tag{1}$$

and taking the Boltzmann probability energy factor e^{-E/k_BT} alongwith kinetic energy expression for a point particle $\frac{1}{2}mv^2$ we obtain the probability density distribution of speeds as:

$$f(v) = 4\pi \left(\frac{m}{2\pi k_B T}\right)^{3/2} v^2 \exp(-mv^2/k_B T) \tag{2}$$

which is normalised so that:

$$\int_0^\infty f(v)dv \equiv 1 \tag{3}$$

and is known as the Maxwell-Boltzmann distribution with the form as shown in Figure 2. Differentiating, we locate the most probably speed at the peak of the distribution:

$$\frac{df(v_p)}{dv} = 0 => v_p = \sqrt{(}2k_BT/m) \tag{4}$$

where v_P is the most probable occurring speed and thus temperature T can be obtained from:

$$T = \frac{mv_p^2}{2k_B} \tag{5}$$

Temperature is also related to the width of the Maxwell-Boltzmann distribution functions and can also be (more reliably) fitted to that, than from the peak position which is more prone to experimental uncertainty.

3 Molecular Dynamics

The molecular dynamics (MD) technique involves a numerical integration of the classical equations of motion for a number of particles, or molecules, given a model Hamiltonian [2, 24, 27]. The resulting numerical trajectory through phase space, can be used to make a number of useful measurements of the dynamical properties of a system [31].

There are several difficulties involved in applying this technique to the simulation of an alloy, even a purely binary one. Primarily, the first requirement is for a suitable model Hamiltonian for the system. While this is not easy to obtain exactly, it *is* possible to construct an approximate one with the basic properties, using pair-wise potentials. For example one very simple model would be to assume a system of soft spherical atoms in a box, with two atomic species present, and atoms of each species preferentially attracting atoms of its own species.

A model Hamiltonian of the Lennard-Jones form [27] can be constructed using:

$$U(R) = 4\epsilon \left[\left(\frac{\sigma}{R}\right)^{12} - x^{i,j} \left(\frac{\sigma}{R}\right)^{6} \right] \tag{6}$$

This gives the potential energy U due to the pair-wise interaction of two atoms of species i, j, separated by some radial distance $R = |\mathbf{r}_i - \mathbf{r}_j|$, given two atom specific parameters in the form of an energy ϵ and a length scale σ, and a cross term coupling fraction $x^{i,j}$ which is greater for interactions between like species i, j than for opposite species.

4 Model Implementation

This molecular dynamics simulation has been implemented for multi-GPU systems using CUDA. The simulation computes the potential between particles using the all-pairs algorithm and computes hard-sphere collisions using a posteriori method. All hard-sphere collisions are inelastic as are the collisions of particles hitting the walls of an enclosing box. A diagram of the inelastic collisions are shown in Figure 3.

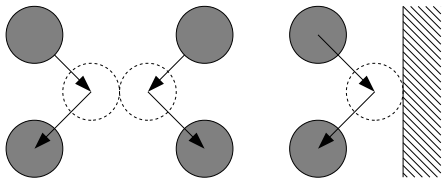

Figure 3: Inelastic Collisions

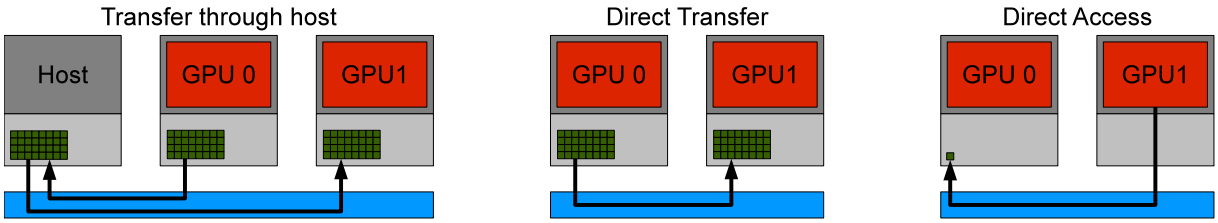

Figure 4: GPU transfer methods.

This simulation has been implemented and tested for m-GPU systems containing up to four GPU devices. The high-level algorithms are the same as the ones discussed in [19]. The particles are evenly distributed between the GPU devices and each device is responsible for updating its particles. To calculate the total force on a particle, each device will calculate the force each of it's own particles exert on each other as well as the force exerted on its particles by the particles stored in the other devices. This requires communication between the different devices, there are a number of ways this communication can be performed and they are discussed and compared in section 5. All force calculations make use of the tiling algorithm [29] as it still provides the best performance.

Various methods can be used to integrate the motion of the particles based on the laws of motion and the potential between them. The different methods have various tradeoffs in terms of memory usage, computation time, stability and accuracy. Selecting the optimal method is not always simple. It has been shown that high-order methods may be computationally more expensive per step, they are stable and accurate with larger step sizes resulting in an overall improvement in performance. The implementations in this work have been tested using the following methods - Euler, Runge-Kutta 2^{nd}, Runge-Kutta 4^{th}, Dormand-Prince 5^{th} and Hairer 10^{th}.

The fundamental collision model used in this research is the same as the method discussed in [19]. After each time-step, each particle is checked against every other particle to determine if a collision has occurred. The collision detection phase is similar to the all-pairs force calculation. If a collision has occurred, the device will calculate the time at which the collision occurred and record it. Once all the particles have been checked, the collision that occurred first will be corrected. Both particles are stepped backwards in time to the point of collision, an inelastic collision is computed at this point and the particles are then moved forward in time to same time as the rest of the system. This process is repeated until there are no more collisions.

5 M-GPU Communication

The focus of this research was to compare the different methods of communication CUDA provides for transferring data between devices. In [19] we evaluated the performance of PCIe extender chassis such as the Dell C410x. For that comparison the simple peer-to-peer memory copy made available by NVIDIA's GPUDirect 2.0. In this research we compare the performance of all the peer-to-peer communication methods.

Before GPUDirect, all communication in an m-GPU system had to go through the host memory. GPUDirect 2.0 allows the direct transfer of data between devices and eliminates the need for extra system memory and transfer overhead. There are two types of direct communication supported by GPUDirect 2.0 - Direct Transfer and Direct Access.

Direct Transfer is a host controlled memcpy from one device directly to another. This data transfer does not need to go through the host and thus avoids unnecessary transfer overheads. CUDA 4.0 provides two methods for Direct Transfer - `cudaMemcpyPeer` and `cudaMemcpyPeerAsync`. `cudaMemcpyPeer` is a blocking memory call that will not return until the transfer is complete. `cudaMemcpyPeerAsync` is non-blocking and the method will return immediately even though the transfer has not yet completed.

Direct Access allows a thread executing on one device to access a value stored in the memory of another device. This feature makes used of the Unified Virtual Addressing (UVA) supported by CUDA 4.0. UVA gives a single address space to the host and device memory, previously the host and each device had a separate memory address space [28]. Direct Access is designed for Non-Uniform Memory Access (NUMA) patterns. Using this method of communication the kernels on each device can simply access the values from the other devices and there is no need

for any memcpy calls.

6 CUDA Implementations

To use the CUDA peer-to-peer communication methods, the devices must have Peer Access enabled. This must be set on each device for every other device peer-to-peer communication will be used. Peer-to-peer communication with device `d` can be enabled by simply calling the function `cudaDeviceEnablePeerAccess(d,0)`.

Listings 1 and 2 show the two methods of Direct Transfer. Listing 1 blocking copy method `cudaMemcpyPeer`. The thread connects two each device and uses `cudaMemcpyPeer` to copy the particle data out of another device into memory on the current device. Once all devices have performed this copy, kernels will be launched on each device to compute something with this data. This computation can either be a force calculation or a collision detection kernel.

Listing 1: Direct Transfer Method - cudaMemcpyPeer.

```
for(int t = 1; t < P; t++) {
  for(int id = 0; id < P; id++) {
    int id2 = (id + t)%P;
    cudaSetDevice(id);
    cudaMemcpyPeer(p2[id],id,p[id2], id2,size);
  }
  for(int id = 0; id < P; id++) {
    cudaSetDevice(id);
    //Compute something using p2[id]
  }
}
```

The non-blocking copy method is very similar but will use `cudaMemcpyPeerAsync` instead. In this implementation the host will launch all the memory copy calls at once instead of waiting for each one to finish before launching the next. The kernels are also launched immediately but will not execute until the data transfer is completed. The code snippet for this data transfer method is shown in Listing 2.

Listing 2: Direct Transfer Method - cudaMemcpyPeerAsync.

```
for(int t = 1; t < P; t++) {
  for(int id = 0; id < P; id++) {
    int id2 = (id + t)%P;
    cudaSetDevice(id);
    cudaMemcpyPeerAsync(p2[id],id,p[id2], id2,
                        size, stream[id]);
  }
  for(int id = 0; id < P; id++) {
    cudaSetDevice(id);
    //Compute something using p2[id]
  }
}
```

The Direct Access method shown in Listing 3 does not use any copy functions. Instead the computation kernels can simply access the particles in the other devices. Each device simply calculates the device id of all the other devices and get the address of their particle data. The computation kernels can then access the particles at these addresses. This requires Unified Virtual Addressing otherwise each device would access its own memory and not the memory on other devices.

Listing 3: Direct Access Method.

```
for(int id = 0; id < P; id++) {
  cudaSetDevice(id);
  id1 = (id + 1)%P;
  id2 = (id + 2)%P;
  ...
  //Compute something using p[id1], p[id2]...
}
```

These three implementations can all be used to transfer data between multiple devices for molecular dynamics simulations on m-GPU systems. The performance of these three methods are compared in Section 7.

7 Performance Results

The implementations discussed in this work have been evaluated using two different experiments. The first shows the time required to compute a molecular dynamics simulation with potentials and hard-sphere collisions at different system densities. The second compares the performance of the different m-GPU implementations. Both of these experiments have been computed with four NVIDIA C2070 compute cards connected to a Dell C410x PCIe extender chassis and use the Runge-Kutta 4^{th} order integration method. The performance of this system is compared to other GeForce and Tesla card configurations in [19].

The first experiment was to compare the three peer-to-peer transfer methods - Synchronous Direct Transfer (DT-S), Asynchronous Direct Transfer (DT-A) and Direct Access (DA). These three methods have been tested on the Dell C410x and the results for a range of transfer sizes are shown in Figure 5. This shows that the Asynchronous Direct Transfer method provides the best performance and Direct Access outperforms Synchronous Direct Transfer for small transfers but

is slightly slower for larger transfers (> 256 KB). It is important to note that the Direct Transfer methods communicate the information between devices but it must still be read by the kernels from global memory, whereas the kernels using Direct Access will have already read the required values.

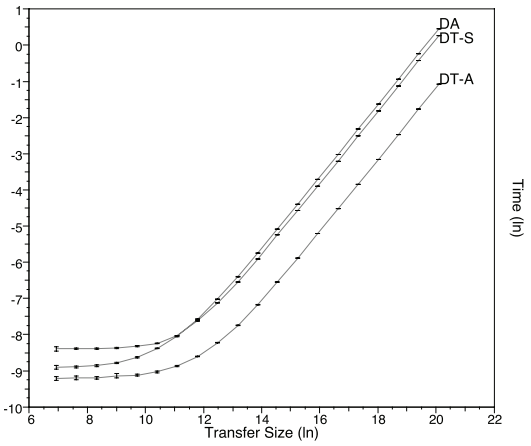

Figure 5: Transfer time for three peer-to-peer communication methods for 4KB-2GB shown in ln-ln scale.

The second experiment is designed to compare the performance of the three m-GPU implementations in the actual simulation. The methods have been compared across a range of system sizes and densities. The performance results (in milliseconds per time step) are presented for one fixed density. These results are shown in Table 1.

Table 1: Performance comparision of GPUDirect simulation implementations. Times acurate to ± 0.005 millisecs per timestep.)

Size N	Direct Transfer	Direct Transfer (Async)	Direct Access
1024	3.46	2.66	3.52
2048	5.55	4.76	5.58
4096	9.81	9.03	9.80
8192	18.78	18.12	18.73
16384	53.38	52.26	53.18
32768	194.56	193.50	193.78
65536	729.27	727.75	727.01

8 Discussion

The almost indistinguishable performance results of the three implementations presented in Section 7 are surprising to say the least. The close performance of the two Direct Transfer methods is not entirely unusual as the data is transferred as a block in by both implementations. In previous work [35], with asynchronous data transfer through host showed a significant performance improvement over synchronous. However, this work focused on lattice models where only the borders had to be transferred and thus all communication could be completely hidden.

However, it was highly unexpected that the Direct Access method would provide almost the same performance as the other two implementations. It was expected that the NUMA style of access to another device's memory would significantly reduce performance. However, all performance benchmarks and tests for both all-pairs force calculation and collision detection show almost no difference in performance. The high performance of Direct Access represents a significant step forward for m-GPU programming. Previous m-GPU research [32, 35] which used communication through the host found the performance of m-GPU programs to be extremely sensitive to communication methods used. Getting the best performance out of such applications required a great deal of fine-tuning.

For this reason, the high-performance of the Direct Access functionality of GPUDirect 2.0 was unexpected. However, it has performed well in every test and significantly reduces programmer effort. Kernels can simply access data from other devices without the need for explicit data transfer and extra memory to store duplicated data. For this reason we believe the Direct Access functionality of GPUDirect 2.0 to be extremely valuable for m-GPU systems.

9 Conclusions

The molecular dynamics simulation discussed in this work is capable of collisional and collision less simulations, with or without elastic wall collisions and a range of possible force/potential models including simple gravitational attraction and multiple species Lennard Jones style systems. These simulations have been implemented for m-GPU systems with multiple GPUs to accelerate a single CPU. Such systems are becoming an increasingly important node architecture for future supercomputers and clusters.

These implementations make use of the GPUDirect 2.0 device-device communication methods that allow data to be directly transferred from one device to another without going through the host. The three implementations show almost no difference in performance despite the fact that the Direct Access method

uses a Non-Uniform Memory Access pattern. This surprising result makes it significantly easier to develop efficient m-GPU applications and represent a major step forward in m-GPU technology.

References

[1] Aarseth, S.J.: Post-newtonian n-body simulations. Mon. Not. R. Astron. Soc. 378, 285–292 (2007)

[2] Allen, M., Tildesley, D.: Computer simulation of liquids. Clarendon Press (1987)

[3] Barnes, J., Hut, P.: A hierarchical o(n log n) force-calculation algorithm. Nature 324(4), 446–449 (1986)

[4] Biferale, L., Coveney, P.V., Ubertini, S., Succi, S.: Discrete simulation of fluid dynamics: Applications. Phil. Trans. Roy. Soc. A 329, 2384–2386 (2011)

[5] Bourg, D.M.: Physics for Game Developers. No. ISBN 978-0596000066, O'Reilly (2002)

[6] Brunet, J.P., Mesirov, J.P., Edelman, A.: An optimal hypercube direct n-body solver on the connection machine. In: Proc. Supercomputing 90. pp. 748–752. IEEE Computer Society, 10662 Los Vaqueros Circle, CA 90720-1264 (nov 1990)

[7] Dehnen, W., Read, J.I.: N-body simulations of gravitational dynamics. arXiv 1105.1082v1, University of Leicester (May 2011)

[8] Demiroz, B., Topcuoglu, H.R., Kandemir, M., Tosun, O.: Particle simulation on the cell be architecture. Cluster Comput. July, 1–14 (2011)

[9] Desell, T., Magdon-Ismail, M., Szymanski, B., Varela, C.A., Willett, B.A., Arsenault, M., Newberg, H.: Evolutionary n-body simulations to determine the origin and structure of the milky way galaxy's halo using volunteer computing. In: Proc. IEEE Int. Symp. on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW). pp. 1888–1895 (16-20 May 2011)

[10] Donev, A., Torquato, S., Stillinger, F.H.: Neighbor list collision-driven molecular dynamics simulation for nonspherical hard particles: I. algorithmic details. J. Comput. Phys. 202(2), 737–764 (2005)

[11] Eberly, D.H.: 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics. No. ISBN: 978-0122290633, Morgan Kaufmann (2006)

[12] Eberly, D.H., Shoemake, K.: Game Physics. No. ISBN 978-1558607408, Morgan Kaufmann (2003)

[13] Fortin, P., Athanassoula, E., Lambert, J.C.: Comparisons of different codes for galactic n-body simulations. Astronomy and Astrophysics 531, A120–1–11 (2011)

[14] Ganesan, N., Bauer, B.A., Lucas, T.R., Patel, S., Taufer, M.: Structural, dynamic, and electrostatic properties of fully hydrated dmpc bilayers from molecular dynamics simulations accelerated with graphical processing units (gpus). J. Computational Chemistry July, 2958–2973 (2011)

[15] G.Fox, M.Johnson, G.Lyzenga, S.Otto, J.Salmon, D.Walker: Solving problems on concurrent processors, vol. 1. Prentice Hall (1988)

[16] Groen, D., Zwart, S.P., Ishiyama, T., Makino, J.: High performance gravitational n-body simulations on a planet-wide distributed supercomputer. arXiv 1101.0605v1, Leiden Observatory, Leiden University, The Netherlands (2011)

[17] Hagan, R., Cao, Y.: Multi-gpu load balancing for in-situ visualization. In: Int. Conf. on Parallel and Distributed Processing Techniques and Applications (2011)

[18] Hairer, E.: A Runge-Kutta Method of Order 10. J. Inst. Maths. Applics. 21, 47–59 (1978)

[19] Hawick, K.A., Playne, D.P.: Hard-sphere collision simulations with multiple gpus, pcie extension buses and gpu-gpu communications. Tech. Rep. CSTN-138, Computer Science, Massey University, Albany, Auckland, New Zealand (August 2011)

[20] Hawick, K., Playne, D., Johnson, M.: Numerical precision and benchmarking very-high-order integration of particle dynamics on gpu accelerators. In: Proc. International Conference on Computer Design (CDES'11). No. CDE4469, Las Vegas, USA (July 2011)

[21] Hu, Q., Syal, M., Gumerov, N.A., Duraiswami, R., Leishman, J.G.: Toward improved aeromechanics simulations using recent advancements in scientific computing. In: Proc. 67th Annual Forum of the American Helicopter Society. Virginia Beach, USA (3-5 May 2011)

[22] Hut, P., Makino, J.: Astrophysics on the GRAPE Family of Special-Purpose Computers. Science 283, 501–505 (1999)

[23] Kavanagh, G.D., Lewis, M.C., Massingill, B.L.: GPGPU planetary simulations with CUDA. In: Proceedings of the 2008 International Conference on Scientific Computing (2008)

[24] Kittel, C.: Introduction to Solid State Physics. Wiley (2004), iSBN 978-0-471-41526-8

[25] Koda, J., Shapiro, P.R.: Gravothermal collapse of isolated self-interacting dark matter haloes: N-body simulation versus the fuid model. Mon. Not. R. Astron. Soc. March, 1–14 (2011)

[26] Larsson, P., Hess, B., Lindahl, E.: Algorithm improvements for molecular dynamics simulations. Wiley Interdisciplinary Reviews: Computational Molecular Science 1, 93–108 (January 2011)

[27] Lennard-Jones, J.: Cohesion. Proc. Royal Soc. 43, 461–482 (1931)

[28] NVIDIA® Corporation: CUDA™ 4.0 Programming Guide (2011), `http://www.nvidia.com/`, last accessed November 2011

[29] Nyland, L., Harris, M., Prins, J.: Fast n-body simulation with cuda. In: Nguyen, H. (ed.) GPU Gems 3, chap. 31. Addison Wesley Professional (August 2007)

[30] Pawley, G.S.: Molecular dynamics simulation of the plastic phase; a model for SF6. Molecular Physics 43(6), 1321–1330 (1981)

[31] Pawley, G.S.: Molecuar dynamics and spectroscopy. In: Price, D.L., Skold, K. (eds.) Methods in experimental physics. pp. 441–519. Academic Press (1986), volume 23 Neutron Scattering Part A

[32] Playne, D.P., Hawick, K.A.: Comparison of GPU Architectures for Asynchronous Communication with Finite-Differencing Applications. Tech. Rep. CSTN-111, Massey University (2010), submitted to: Concurrency and Computation: Practice and Experience

[33] Playne, D.P., Johnson, M.G.B., Hawick, K.A.: Benchmarking GPU Devices with N-Body Simulations. In: Proc. 2009 International Conference on Computer Design (CDES 09) July, Las Vegas, USA. No. CSTN-077 (July 2009)

[34] Playne, D.P.: Notes on Particle Simulation and Visualisation. Hons. thesis, Computer Science, Massey University (June 2008)

[35] Playne, D., Hawick, K.: Hierarchical and Multi-level Schemes for Finite Difference Methods on GPUs. In: Proc. CCGrid 2010, Melbourne, Australia. No. CSTN-099 (May 2010)

[36] Sarman, S., Evans, D.J.: Heat flow and mass diffusion in binary lennard-jones mixtures. Phys.Rev.A 45(4), 2370–2379 (feb 1992)

[37] Schneider, M.D., Cole, S., Frenk, C.S., Szapudi, I.: Fast generation of ensembles of cosmological n-body simulations via mode-resampling. arXiv 1103.2767v3, Lawrence Livermore Nat. Lab., USA. (May 2011), lLNL-JRNL-471523

[38] Stock, M.J., Gharakhani, A.: Toward efficient GPU-accelerated N-body simulations. In: in 46th AIAA Aerospace Sciences Meeting and Exhibit, AIAA 2008-608 (January 2008)

[39] Suzuki, T.: Method of N-body simulation on the MOdified Gravity. arXiv 1110.5420, Yamaguchi University, Japan (October 2011)

[40] Tanikawa, A., Yoshikawa, K., Okamoto, T., Nitadori, K.: N-body simulation for self-gravitating collisional systems with a new SIMD instruction set extension to the x86 architecture, Advanced Vector eXtensions. New Astronomy 17, 82–92 (2012)

[41] Warren, M.S., Salmon, J.K.: A parallel hashed oct-tree n-body algorithm. In: Supercomputing. pp. 12–21 (1993), citeseer.ist.psu.edu/warren93parallel.html

[42] Yokota, R., Barba, L.A.: Fast multipole method vs. spectral methods for the simulation of isotropic turbulence on gpus. arXiv 1110.2921v1, Boston University, USA (October 2011)

[43] Yokota, R., Barba, L.A.: Fast n-body simulations on gpus. arXiv 1108.5815, Boston University, USA (August 2011)

[44] Zhao, G.B., Li, B., Koyama, K.: N-body Simulations for f(R) Gravity using a Self-adaptive Particle-Mesh Code. Phys. Rev. D 83, 044007–1–19 (2011)