Performance Tradeoff Spectrum of Integer and Floating Point Applications Kernels on Various GPUs

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

April 2013

ABSTRACT

Floating point precision and performance and the ratio of floating point units to integer processing elements on a graphics processing unit accelerator all continue to present complex tradeoffs for optimising core utilisation on modern devices. We investigate various hybrid CPU and GPU combinations using a range of different GPU models occupying different points in this tradeoff space. We analyse some performance data for a range of numerical simulation kernels and discuss their use as benchmark problems for characterising such devices.

KEY WORDS
MIPS vs FLOPS; computational performance; accelerator; benchmark; GPU.

1 Introduction

Graphical Processing Units [10, 11] have become almost mainstream accelerator devices in many applications areas. They remain non-trivial to program even with the advent of highly developed software libraries and tools such as NVidia's Compute Unified Device Architecture (CUDA) [12] and Open Compute Language (OpenCL) [16]. There are still some applications and parts of applications for which GPUs provide very good speedups and others for which they are less suitable. To further complicate the users decision on which platform to deploy upon there have been many different GPU models released over the last five years each of which has different design features and performance characteristics.

In this paper we use some very simple synthetic benchmarks to experiment with a range of different GPU devices and benchmark performance across them. Our particular focus of interest in integer versus floating point performance. Other factors such as memory transfer bandwidth and the exact ratio of floating point, double precision and special function evaluation units also play a part.

Our long term goal is to develop a set of GPU related benchmarks appropriate for computational fluid dynamics (CFD) [1, 2, 18] and comparing conventional CFD calculations [9] that are formulated in terms of partial differential equations that require raw floating point performance [6] with those formulated in terms of integer calculations and a lattice gas model approach [8].

There are a number of well known benchmarks for floating point performance in the context of linear algebra and matrix calculations [4, 17]. The NAS parallel benchmarks [3] also exercise some features that are specific to CFD problems as well. Other benchmarks have considered asynchronous coupling effects between the GPU and its hosting CPU [13], or have been tailored to specific applications areas such as particle dynamics [7, 15] or graph and network problems [5].

There is topical scope to consider multiple GPUs [14, 19] attached to the same CPU and driven or serviced by different cores. In this present paper however we focus on rather low-level capabilities of the various GPU accelerators we study and the main contribution is that we have been able to study quite a large collection of different vintage devices and can comment on which particular features contribute to which low level performance trend.

Our article is structured as follows: We review some of the key architectural features of graphical processing units in Section 2. We benchmark the GPUs by separating and emphasising the individual elements that make GPGPU and specifically NVidia GPUs both powerful and limiting. The areas we identified were: integer and double precision computation and global memory access. These elements represent the greatest divide between the various devices and we see in Section 3 both the strengths and weaknesses of each

device relative to the other generations of NVidia hardware. We present a selection of benchmarks for low level operations in Section 4. We discuss their implications in Section 5 and offer some tentative conclusions, directions for the future and other areas for further investigation in Section 6.

2 GPU Architecture

Since the initial release of CUDA and the rise of GPGPU computing, NVIDIA has released several GPU architectures. Each of these subsequent GPU architecture releases have brought with them higher performance and additional chip capabilities that make GPGPU programs faster and easier to develop.

The GT200 released in 2008 saw the introduction of double precision processing and a reduction on the performance penalties on non-coalesced memory accesses. The GF100 Fermi architecture GPUs released in 2010 in the GeForce 400 series and saw an increase in the number of cores per multiprocessor to 32 and most significantly for the GPGPU community the introduction of an L1/L2 cache structure. The GF110 was released later in 2010 in the GeForce 500 series which brought with it performance improvements over the GeForce 400 series. The general architecture of the Fermi architecture multiprocessor can be seen in Figure 1.

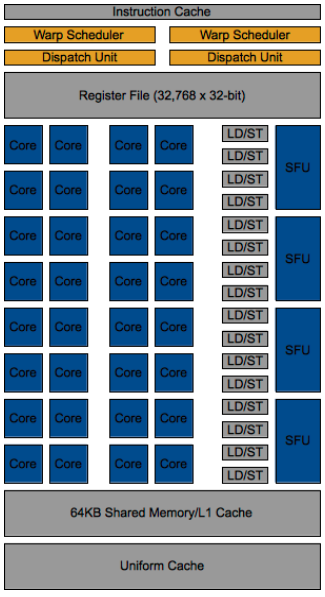

Figure 1: The architecture of a Fermi GPU multiprocessor with 32 cores, 16 Load/Store units and 4 special function units.

In 2012 NVIDIA released the new Kepler architecture GPU featuring the new generation Streaming Multiprocess architecture (SMX). These multiprocessors contain 192 cores which has allowed the maximum number of cores in a single GPU to be increased to 1,536. These GPUs provide higher performance than the previous generation Fermi devices while using significantly less power. The architecture of these GPUs is shown in Figure 2.

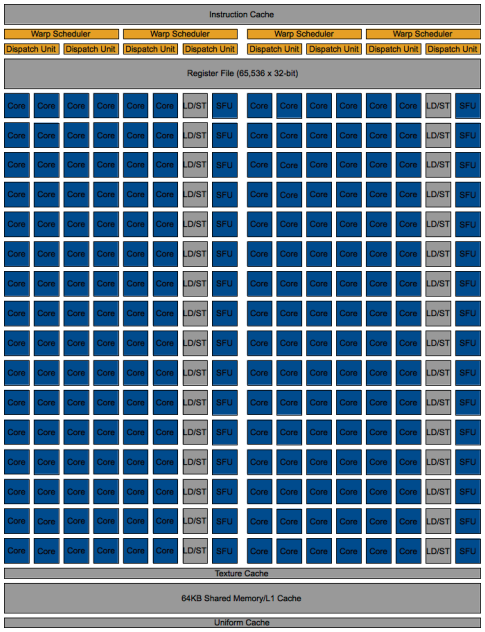

Figure 2: A Kepler architecture multiprocessor containing 192 cores, 32 Load/Store units and 32 special function units.

While these Kepler GPU have significantly higher performance than the previous generation GPUs (theoretical peak of 3090.4 GFlops for a GeForce GTX680 as compared to 1581.1 GFlops for a GeForce GTX580) the overall memory bandwidth has remained almost the same 192.2 GB/sec (GTX680) compared to 192.4 GB/sec (GTX580). This presents a problem for some GPGPU applications which maybe limited by memory speed and not computational performance. We evaluate the practical performance of these two GPU architectures and compare them in terms of computational throughput and memory access.

3 Implementation Method

We decided to reduce the benchmarks (micro benchmarks) to their simplest possible state. For computation we chose a basic Linear Congruential random number generator as shown in Listing 1. While, not the best random number generator for high quality randomness it only uses integer calculations and no memory access. Each thread has

one kernel which generates one thousand random numbers. Only kernel execution time is recorded and averaged over multiple separate runs.

```
__global__ void int_compute_benchmark()
 {
    int ix = blockDim.x *blockIdx.x +threadIdx.x;
    int M = 8;
    int a = ix;
    int c = 3;
    int X = 1;
    int i;
    for(i=0; i<1000; i++)
    {
        X = (a * X + c) % M;
    }
}
```

Listing 1: Device kernel generating one thousand random numbers using a Linear Congruential generator for the integer computation benchmark.

To evaluate the double precision speed of the GPUs we use the same idea as the integer but change the algorithm to a simple quadratic equation solver as shown in Listing 2. This uses both double precision computation as well as the special function units in each of the multi processors.

```
__global__ void double_compute_benchmark()
 {
    int ix = blockDim.x *blockIdx.x +threadIdx.x;
    double linear = ix;
    double cons = blockIdx.x*blockIdx.y;
    double num1=0,num2=0;
    double power=0;
    for(int i=0; i<1000; i++){
        double quadratic = i+1;
        power=pow (linear / 2 , 2.0);
        num1= ( - linear + sqrt(power - ( 4 *
            quadratic * cons )) ) / (2 *
            quadratic);
        num2= ( - linear - sqrt(power - ( 4 *
            quadratic * cons )) ) / (2 *
            quadratic);
    }
}
```

Listing 2: Device kernel to solve one thousand different quadratic equations for the double precision computation benchmark.

We examine how memory reading and writing speed differ between the devices. Again we use the most simple example to exasperate memory transfer cost. Examining both random and coalesced reads and writes exposes the advantages and flaws for differing architecture. We expect that the random reads will perform much worse on all devices but will affect the 600 series cards the most, as the number of multi-processes have been reduced.

Listing 3 shows the algorithm we use to test the coalesced memory reads and writes. Firstly we allocate two integer arrays and populate them with random integers. The memory allocation and population is not included in the benchmark timing. Each element is then copied from array A to array B, incremented and written back to A. This allows for large coalesced reads and writes with very little other computation.

```
__global__ void
 coalesced_memory_benchmark(int *A, int *B,int *
     rStore)
 {
    unsigned int i = ((((blockIdx.y * gridDim.x)
        + blockIdx.x) * blockDim.x) +
        threadIdx.x);
        A[i] = B[i];
        B[i]++;
        B[i] = A[i];
}
```

Listing 3: Device kernel to benchmark the coalesced memory read and write speed of the devices.

```
__global__ void
 random_memory_benchmark(int *A, int *B,int *
     rStore)
 {
        unsigned int i = ((((blockIdx.y *
            gridDim.x) + blockIdx.x) *
            blockDim.x) + threadIdx.x);
        int rnd1 = rStore[i];
        int rnd2 = rStore[rnd1];
        int rnd3 = rStore[rnd2];
        int rnd4 = rStore[rnd3];
        A[rnd1] = B[rnd2];
        B[rnd3] = A[rnd4];
}
```

Listing 4: Device kernel to benchmark random memory read and writes to device global memory.

Listing 4 shows the algorithm we have used to benchmark the random memory access time for the various GPUs. Each thread must perform two random reads and two random writes to global memory. Using the same random number for multiple threads may result in some collisions. However as this is consistent across all of the benchmark it does not present any advantage to a specific device.

4 Performance Results

Figure 3 shows the plot of kernel execution time for the integer computation benchmark vs the number of thread blocks, which contain 32 threads each. We see generally predictable results. With the GTX 680 the fastest followed but the: 2090, 580, 590 and so on. The order is representative of the number of cores per GPU and for devices with

	260	480	580	590	660m	680	M2050	M2070	M2075	M2090
Compute Version	1.3	2.0	2.0	2.0	3.0	3.0	2.0	2.0	2.0	2.0
Total Global Memory(MB)	896	1536	1536	1536	512	2048	2687	5375	5375	5375
Number of Compute Cores	216	480	512	512	384	1536	448	448	448	512
Number of Multi Procs	27	15	16	16	2	8	14	14	14	16
GPU Clock Rate(MHz)	1400	1400	1590	1225	950	706	1150	1150	1150	1301
Memory Clock (MHz)	1000	1848	2004	1710	256	3004	1546	1494	1556	1848
Memory Bus(Bit)	448	384	384	384	256	256	384	384	384	384
L2 Cache(KBytes)	0	768	768	768	512	512	768	768	768	768
Const Memory Size(KB)	64	64	64	64	64	64	64	64	64	64
Shared Memory Size(KB)	16	48	48	48	48	48	48	48	48	48
Registers Per Block	16384	32768	32768	32768	65536	65536	32768	32768	32768	32768
Has ECC	No	No	No	No	No	No	Yes	Yes	Yes	Yes

Table 1: Table comparing the various NVidia GPU models that we benchmark.

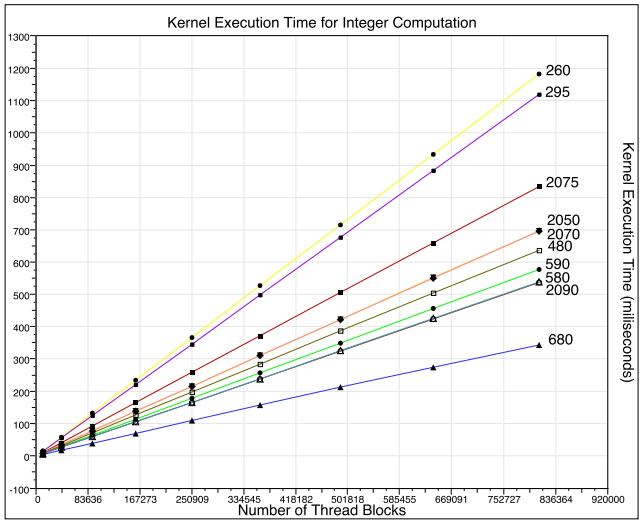

Figure 3: Integer computation test

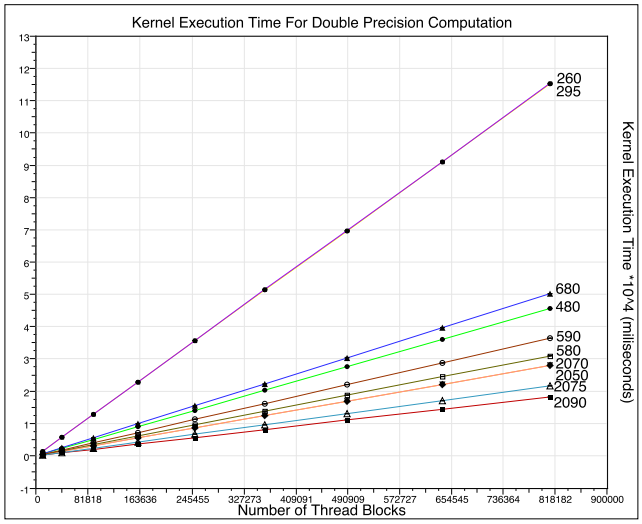

Figure 4: Double computation test

the same number of cores the clock speed separates them.

Figure 4 shows the kernel execution time for the double precision computation benchmark vs the number of thread blocks, again containing 32 threads each. Unlike the integer computation benchmark we see some unexpected results. We see that the most recent GPU tested the GTX 680 is the slowest aside from the 200 series GPUs. As expected the Tesla compute cards are the best performing cards in this test. With the 2050 and 2070 again showing nearly identical results as the main difference between them is the memory size and minute GPU clock rate difference. The 2075 shows an improvement over the 2070 and 2050 which is then followed by the 580 and 480.

We see in Figure 5 the results of the random access memory benchmark. Again the results of this test are unusual as the most recently released GPU the GTX 680 is not the fastest as it is beaten by the 480, 580 and 590. We believe this is due to the smaller number of multiprocessors in the 680 with eight compared with sixteen in the 580 and 590 and 14 in the 480. Because the multiprocessors handle the memory operations for the cores within each one, randomly accessing the memory will significantly affect the devices with lower numbers of multiprocessors.

Figure 6 shows us the results of the coalesced memory access bench mark. We see that unlike the random access benchmark the 680 performs very well. The 580 also performs well and comes in a close second. The GPUs seem to be grouped into three distinct groups with the 260 and the 295 performing surprisingly well compared to the Tesla GPUs.

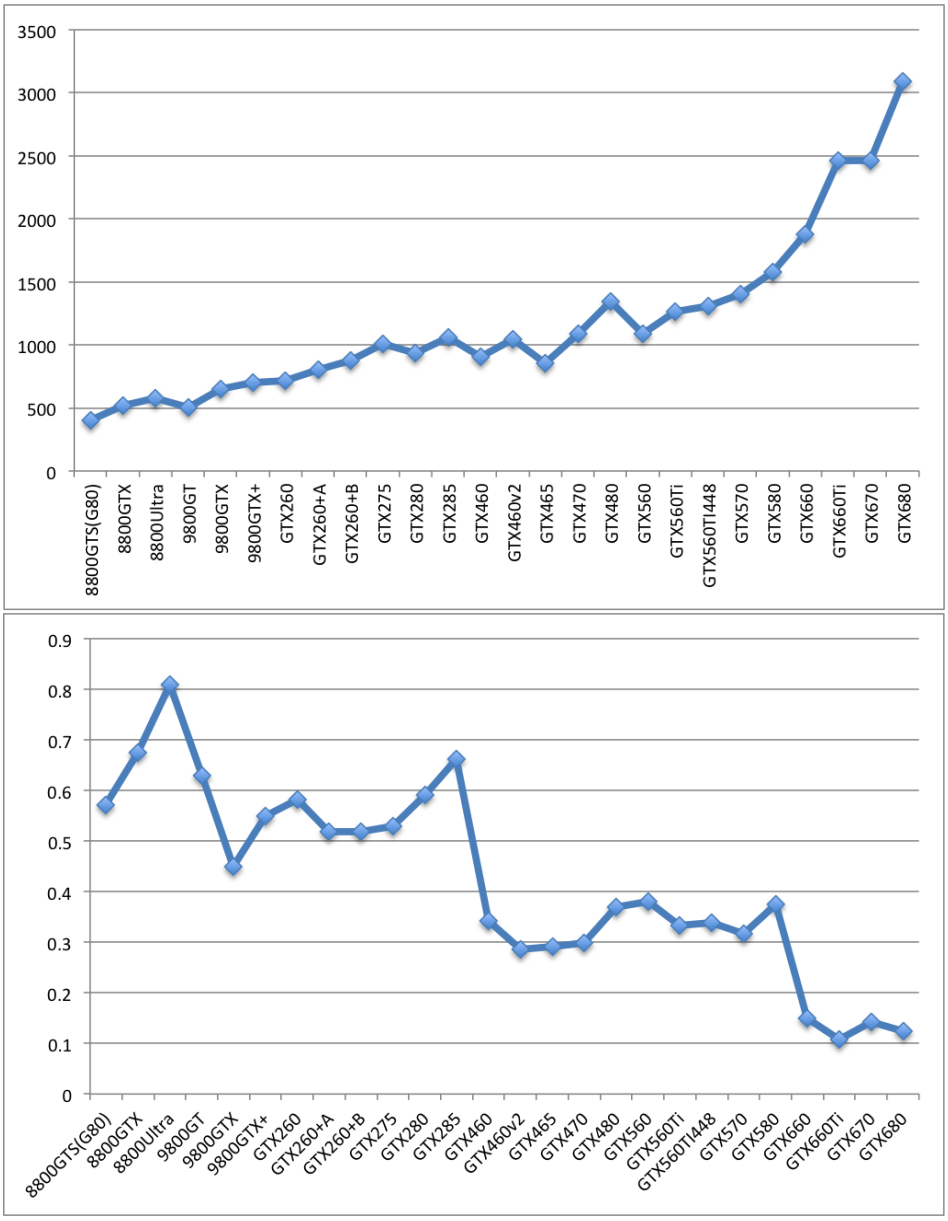

Figure 7: GFlops (above) and GFlops per core (below)

5 Discussion

GPUs are primarily designed to render computer graphics and only recently have begun to be used for general purpose computing (GPGPU). Producing graphics requires primarily integer calculations and we see the result of this in Figure 3 where each generation of NVidia GPU performs better than the previous. The main factor in the integer computation performance seems to be the number of cores followed by the clock speed. We see evidence of the impact clock speed makes in the difference between the 580 and 590 which have almost identical specifications aside from a lower GPU clock speed and lower memory clock speed. The 2075 is also significantly slower than the 2070 and the 2050 as with the memory benchmark this cannot be explained by the specifications.

Double precision has historically been a weak point for GPGPU and specifically the NVidia GeForce consumer GPUs. In the Tesla series they have concentrated on bridg-

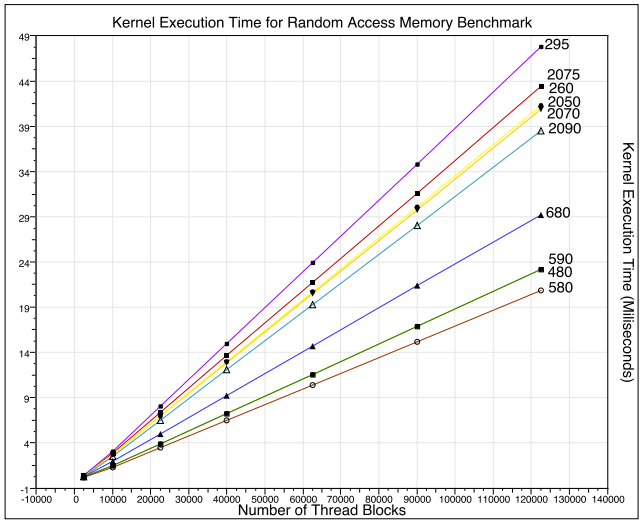

Figure 5: Integer memory test random access

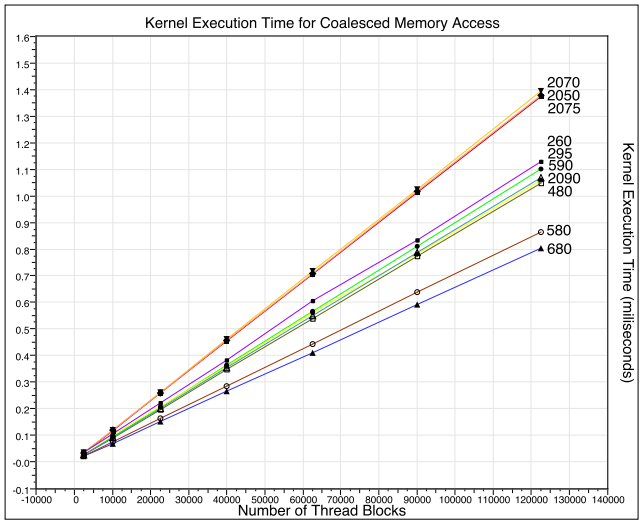

Figure 6: Integer memory test with contiguous access

ing this divide and we see the results of this in Figure 4. The Tesla cards: 2050, 2070, 2075 and the 2090 all perform much better than their GeForce counterparts. More surprisingly the best of these cards was the oldest Fermi architecture card, the 480. The 680 being the slowest card despite being one of the new generation Kepler is reflective of the growing divide between the consumer graphics cards and the professional level GPGPU devices such as the 2090 and consumer graphics focused GeForce cards such as the 680. Although the 680 has many more cores that all of the other GPUs, the ratio of special function units to compute cores is much lower.

The random access memory benchmark shows some surprising results as explained in Section 4. Again we see the 680 being out performed by the previous generation of GPUs. As with the Double precision benchmark the evolving architecture prioritising the number of cores over the number of multiprocessors and special function units. The relatively large improvement of the 2075 over the 2050 and the 2070 cannot be fully explained by the specifications shown in table 4 we can only assume that the change in architecture from GF100 to GF110 in the 2070 and 2075 respectively has some unseen performance benefit in accessing random memory.

The coalesced memory access benchmark is similar to the integer computation benchmark as it reflects the NVidia ideal where all memory access is coalesced. The 680 is not massively faster than the 580 it represents an evolutionary improvement over the previous generation. The biggest surprise is the speed of the Tesla cards, which are mostly much slower than their equivalent GeForce cards. The 2050, 2070 and 2075 are all beaten by both of the 200 series cards. We believe this is due to the higher memory clock and core speed.

Figure 7 (lower) illustrates the overall trend of the NVidia GPGPU architecture. We see that while the overall GFlops per GPU has been increasing as shown in Figure 7(above), the computational power per core has been decreasing. This clearly shows NVidia's plan for GPU architectures moving forward. It may reduce the effectiveness of NVidia GPGPU for memory intensive simulations and possibly more importantly simulations which rely on special function units as shown in Figure 4

6 Conclusion

We have shown that by creating simple micro benchmarks we can easily identify and compare specific functions of GPUs. We see that although some of the latest NVidia GPU architectures have raw performance in certain areas they do not perform as well in fifty percent of our benchmarks. While we do not propose buying older generation GPUs, it may give insight into why simulations are not performing as well on some GPUs and not others. We also show that there is a growing divide between the GeForce consumer cards and the professional GPGPU Tesla GPUs. The next generation of GPUs that have been announced are the K20x and its GeForce cousin the Titan show the continuing trend towards the many core less multiprocessors architecture.

References

[1] Abbott, M., Basco, D.: Computational Fluid Dynamics: an introduction for engineers. Longman (1989)

[2] Acheson, D.: Elementary Fluid Dynamics. Oxford Applied Mathematics and Computing Science Series, Clarendon Press (1990)

[3] Bailey, D., Barscz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., Weeratunga, S.: The nas parallel benchmarks. Tech. Rep. RNR-94-007, NASA Ames Research Center, Moffett Field, CA, USA. (1994)

[4] Dongarra, J.J., Luszczek, P., Petitet, A.: The LINPACK Benchmark: past, present and future. Concurrency and Computation: Practice and Experience 15(9), 803–820 (2003)

[5] Graph500.org: The Graph 500 List. http://www.graph500.org/, last accessed November 2010

[6] Griebel, M., Dornseifer, T.: Numerical Simulation in Fluid Dynamics A Practical Introduction. No. ISBN 0-89871-398-6, SIAM (1998)

[7] Hawick, K.A., Playne, D.P., Johnson, M.G.B.: Numerical precision and benchmarking very-high-order integration of particle dynamics on gpu accelerators. In: Proc. International Conference on Computer Design (CDES'11). pp. 83–89. No. CDE4469, CSREA, Las Vegas, USA (18-21 July 2011)

[8] Johnson, M.G.B., Playne, D.P., Hawick, K.A.: Data-parallelism and gpus for lattice gas fluid simulations. In: Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'10). pp. 210–216. CSREA, Las Vegas, USA (12-15 July 2010), pDP4521

[9] K.Srinivas, C.A.J.Fletcher: Computational Techniques for Fluid Dynamics. Springer Series in Computational Physics, Springer-Verlag (1992), a Solutions Manual

[10] Leist, A., Playne, D.P., Hawick, K.A.: Exploiting Graphical Processing Units for Data-Parallel Scientific Applications. Concurrency and Computation: Practice and Experience 21(18), 2400–2437 (25 December 2009), CSTN-065

[11] Luebke, D., Humphreys, G.: How gpus work. Computer pp. 96–100 (February 2007)

[12] NVIDIA® Corporation: NVIDIA CUDA C Programming Guide Version 4.1 (2011), http://www.nvidia.com/ (last accessed April 2012)

[13] Playne, D.P., Hawick, K.A.: Comparison of GPU Architectures for Asynchronous Communication with Finite-Differencing Applications. Concurrency and Computation: Practice and Experience (CCPE) Online, 1–11 (7 April 2011), http://onlinelibrary.wiley.com/doi/10.1002/cpe.1726/abstract

[14] Playne, D.P., Hawick, K.A.: Classical mechanical hard-core particles simulated in a rigid enclosure using multi-gpu systems. In: Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'12). pp. 76–82. CSREA, Las Vegas, USA (16-19 July 2012)

[15] 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. pp. 150–156. WorldComp, Las Vegas, USA (13-16 July 2009)

[16] Stone, J.E., Gohara, D., Guochun, S.: Opencl: A parallel programming standard for heterogeneous computing systems. Computing in Science & Engineering 12(3), 66–73 (May-June 2010)

[17] TOP500.org: TOP 500 Supercomputer Sites. http://www.top500.org/, last accessed November 2010

[18] Tritton, D.: Physical Fluid Dynamics. Clarendon Press, 2 edn. (1988)

[19] Zaspel, P., Griebel, M.: Solving incompressible two-phase flows on multi-gpu clusters. Computers & Fluids In press, 1–9 (2012)