Technical Report CSTN-125

Asynchronous Communication for Finite-Difference Simulations on GPU Clusters using CUDA and MPI

D.P. Playne and K.A. Hawick
Computer Science, Massey University
Albany, North Shore 102-904, Auckland, New Zealand
{ *d.p.playne, k.a.hawick* } *@massey.ac.nz*

Abstract

Graphical processing Units (GPUs) are finding widespread use as accelerators in computer clusters. It is not yet trivial to program applications that use multiple GPU-enabled cluster nodes efficiently. A key aspect of this is managing effective communication between GPU memory on separate devices on separate nodes. We develop a algorithmic framework for Finite-Difference numerical simulations that would normally require highly synchronous data-parallelism so they can effectively use loosely coupled GPU-enabled cluster nodes. We employ asynchronous communications and appropriate memory overlay of computations and communications to hide latency.

Index Terms

GPU; asynchronous communications; clusters; CUDA; MPI

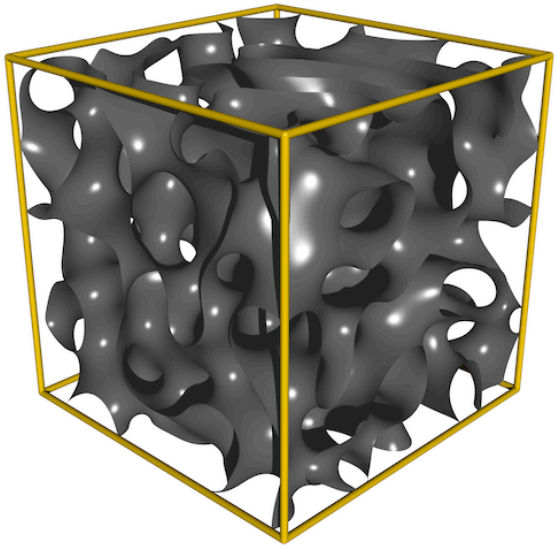

Figure 1. Ray-traced rendering of a Cahn-Hilliard system simulated on a GPU cluster.

1. Introduction

Accelerators such as Graphical Processing Units (GPUs) have steadily been finding a role in super-computer systems in recent years and at the time of writing are particularly prominent in major international systems featuring in the Top500 list of Super-computers [1]. The Tianhe-1A(top), Nebulae(second) and Tsubame(fourth) all employ GPU accelerators and apparently seventeen major systems out of the Top 500 in November 2010 all use NVIDIA GPUs to accelerate node performance.

There is therefore some importance in understanding how GPU accelerators behave when combined in a multi-processor system for various applications. The Linpack benchmark[2] used by the compilers of the Top 500 list tests capabilities in dense linear algebra. This is certainly an important application paradigm but there are others and we are interested in simulation models on recti-linear in hyper-dimensional systems. A good test application for this paradigm is that of solving finite-difference field equations in one, two, three and higher hyper-dimensional meshes.

In previous work[3] we considered dual GPU acceleration of a single compute node and we have since been able to experiment with triple and quadruple[4] GPUs-per-node as well as a range of different GPU models with varying numbers of cores and memory configurations. In this present paper we study the tradeoff space that arises from decomposing a hyper-dimensional rectilinear problems such as solving a high-order partial differential equation (PDE) using

finite-difference methods [5] on a GPU-accelerated cluster.

Finite-difference methods are still used extensively in computational simulations – particularly in wave-based seismic exploration applications[6] and generally are straightforward to parallelise using geometric stencil methods of decomposition which attain good computational speedup[3], [7] and especially on GPUs [8], [9], [10], [11].

There are various ways to "slice and dice" the data set – and in fact for a typical simulation problem of a physical system one can often choose the system size to best fit the memory configuration and layout of the parallel system. In this paper we report on results for a periodic mesh simulation of the Cahn-Hilliard equation for materials science using a second-order space/second-order-time finite-differencing approach. The technique (and our code) extends to other systems such as the Time-Dependent Ginzburg-Landau equation[12] as well, and it explores a slightly different tradeoff space in communications versus computations, since the TDGL uses complex number arithmetic whereas the Cahn-Hilliard model field is wholly real.

There are a number of useful parallel programming technologies that could be employed for these simulations. In this paper we focus on the combination of the open standard Message Passing Interface (MPI)[13], [14] to program inter-node communications and NVIDIA's Compute Unified Device Architecture (CUDA) [15], [16], [17] programming language for programming the calculations on the GPUs themselves.

It is not especially difficult to now build a GPU-accelerated cluster and we report some performance data on a cluster comprising sixteen GPU-accelerated desktop processors connected with cheap 1G Ethernet, which suffices for the work we report here. Commodity-priced switchgear using 10G Ethernet is becoming available although 100G Ethernet or InfiniBand switching systems[18] would be more desirable to achieve good bandwidth that is compatible with present generation CPU clock frequencies.

Halo problems of the nature we describe are not new. A great deal of work has been done on data-parallelism as it pertains to such applications[19]. However managing the computation to communications ratio of regular mesh problems on parallel platforms remains a challenge particularly in the case of hybrid architectures. The tradeoff space shifts around with the coming of each new parallel platform and we present a study of parameters and (well-known) issues as they pertain to a hybrid system where a multi-dimensional simulation model can be geometrically decomposed across cluster nodes in one of the dimensions, and across the memory and data-parallel thread structure of the accelerating GPU in the other two dimensions.

The use of asynchronous communications across multiple GPU systems is still not yet a widely-known approach and we build on our prior work[3] with multiple GPUs hosted from a single GPU and show the importance of asynchronous communications for a GPU cluster with multiple CPUs accelerated with GPUs. We also include some reference performance data from our example simulation model on combinations of single and multiple GPUs by way of comparison.

The application problem we use as a benchmark for our study is a second-order-space/second-order-time partial differential equation – the Cahn-Hilliard field equation [20], [21], [22], which is used for simulations of phase separation in materials science. Figure 1 shows an example of a simulated Cahn-Hilliard model system, ray-traced to show the spinodally decomposing interfaces between two phases after a long period of computational thermal quenching. The numerical methods we describe also apply to other partial differential equations such as the Ginzburg-Landau model of super-conductivity[12], [23], [24] and many other application areas, although we focus on the parallel data distribution and MPI/CUDA hybrid management aspects in this present paper..

In summary therefore, we explore combinations of Message Passing Interface (MPI) and Compute Unified Device Architecture (CUDA) to program a very large computationally simulated partial differential equation model system across a cluster of NVIDIA GTX470 GPU-accelerated Intel Core 2 Quad Q9400 2.66 GHz processor nodes. Our article is structured as follows:

In Section 2 we discuss the challenges faced when working with multiple GPU devices. In Section 4 we present several methods of decomposing a finite-differencing simulation across multiple GPUs on both a single host and distributed nodes. We present and discuss performance data in Section 5 and offer some conclusions in Section 6.

2. Multi-GPU Systems

In previous work [3], [4] we have discussed how the use of CUDA asynchronous memory copies can be used to improve performance when decomposing a finite-difference application across multiple GPUs – connected via PCIe Express bus[25] to a single host CPU. In this present work however we investigate how this method performs when the GPUs are distributed across a compute cluster – when they are hosted by different CPUs.

When GPUs are mounted on a single host, communication between them is relatively simple and very fast. The necessary communication data must be copied from one device into the host memory and then copied to the appropriate device. When the GPUs are distributed throughout a cluster, this data must be communicated via a network. These networks have significantly higher latency and lower bandwidth, thus we expect a drop in performance. The main advantage of GPUs distributed over a cluster is improved scalability which would otherwise be strictly limited in the case of multiple GPUs contending with one another on a single PCIe bus.

In practice, GPUs hosted on a single machine are currently limited to hosting up to a maximum of four GPUs due to motherboard physical constraints - and sometime also by power-supply and cooling limitations. GPU clusters in contrast have only those physical limitations that arise from network infrastructure. However, the speed of the cluster will obviously not scale infinitely due to communication latency limitations. One advantage of increased GPU numbers – as in a cluster over a bus-based arrangement – is the resulting increased upper bound on feasible model system size.

GPUs still have very limited device memory which restricts the maximum system size that can be simulated. By decomposing the simulation across many GPUs, this maximum simulated system size can be increased considerably. It is an important result for GPU clusters to explore the scalability and locate the higher limitations for a cluster system.

GPUs come in a number of models and variations. In this article, we have focused exclusively on NVIDIA GPUs running the Compute Unified Device Architecture (CUDA) software. Other software systems such as the open compute language standard OpenCL are feasible and promise functionality on other vendor's platforms. Our experience has been that CUDA still delivers considerably more performance than OpenCL and we focus solely on CUDA and NVIDIA devices in this paper. CUDA is not totally trivial to port applications source code to, but we employ a well-optimised and tested source code we developed for solving the Cahn-Hilliard equation.

3. Cahn-Hilliard Equation

We and other authors have described this equation in detail elsewhere[20], [21], [22], but for completeness we give a brief summary here. The equation is usually formulated as:

$$\frac{\partial u}{\partial t} = \mathrm{M}\nabla^2\left(-\mathrm{B}u + \mathrm{U}u^3 - \mathrm{K}\nabla^2 u\right) \qquad (1)$$

The field $u(x, y)$ or $u(x, y, z)$ is a multi-dimensional scalar field taking values between ± 1 which represent the two extreme materials phases. So these might represent different atomic species in an alloy or a two separate sorts of fluids. The field is initialised randomly then "numerically quenched" by stepping it forwards in time. The parameters: M, U, K specify the detailed material properties and for our benchmarking purposes here can be set to unity. The model then has a single remaining parameter B which controls the temperature of the simulated quench experiment and thus the rate at which the field separates out into domains. The spatial calculus in the equation employs a double Laplacian operator and thus has a larger halo boundary – or number of neighbours – than simpler common finite-difference equations featuring in mathematics textbooks. The Laplacians are approximated by finite-difference stencils to second-order accuracy and the time-stepping must generally be second-order for domain-growth without introducing artifacts from numerical instability[26]. Figure 1 shows a three dimensional model system that has been time-stepped for many iterations after its random quench and exhibits a complex pattern of interleaving spatial component clusters. The figure was rendered as hyper-surfaces that represent the interfaces between physical domains and are shown rendered using ray-tracing.

4. Finite Difference Decomposition

To split a finite-differencing simulation of a field equation like the Cahn-Hilliard system across a cluster of GPUs, the field must be decomposed into sections that can be stored and processed by each GPU. There are many options in terms of field decomposition - blocks, layers and so forth[27]. We have found that decomposing the field into layers in the highest dimension has proved the most efficient. The field is split into equal layers and spread across the GPUs in each node of the GPU cluster. This method of decomposition is shown in Figure 2.

The main challenge of decomposing an application across a compute cluster is the relatively high latency of communication across the network. This communication is especially important as the GPU accelerated nodes have a higher computational throughput. In our previous work [3], [4] on multiple GPUs on a single host, our update algorithm communicated the bordering information using asynchronous memory

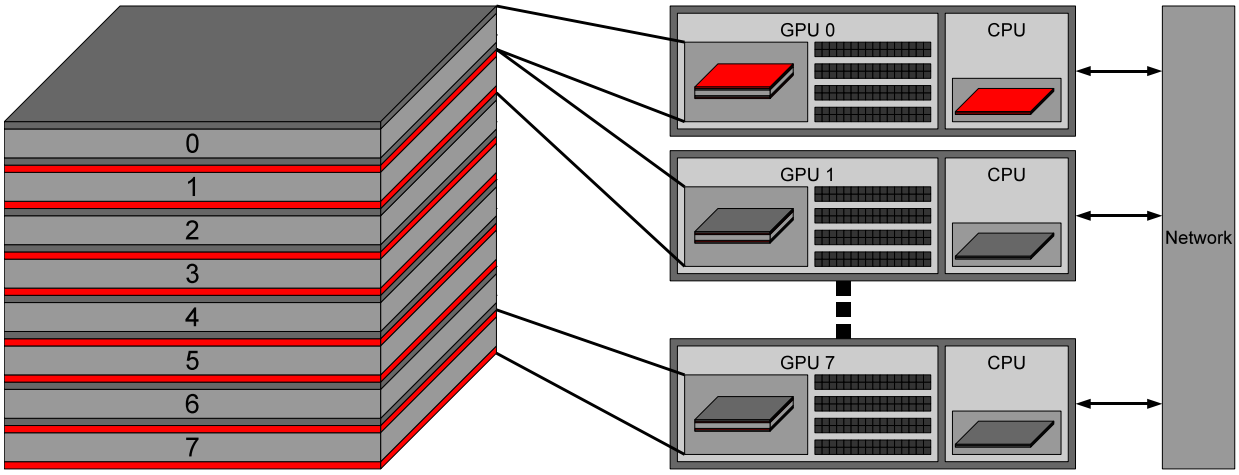

Figure 2. Layer decomposition of a three-dimensional field split across eight nodes in a GPU cluster.

copies to hide latency. This algorithm can be seen in Algorithm 1.

Algorithm 1 Multi-GPU update algorithm using asynchronous memory communication to hide latency.

```
for all steps do
    Compute border cells (stream 1)
    Compute remaining cells in field (stream 2)
    Copy borders from GPU to host (stream 1)
    Exchange borders with neighbours (CPU)
    Copy new borders into GPU (stream 1)
    Synchronize streams
end for
```

The main advantage of this algorithm for multiple GPUs is that the communication can be performed while the GPU is still working. The main computation time is taken up computing the simulation for the main body of the field. By computing the borders first and using asynchronous memory copies, the communication can be performed during this main computation time.

4.1. Algorithm A

This algorithm can be adapted for use with distributed GPUs. Instead of exchanging borders with another thread through the host memory, this data must be sent via MPI. This algorithm still allows information to be communicated while the GPU is still computing, however the latency of sending data across a network is much higher than simple memory copies and we expect a drop in performance. The adapted algorithm is shown in Algorithm 2.

Algorithm 2 GPU cluster update algorithm using asynchronous memory communication and MPI.

```
for all steps do
    Compute border cells (stream 1)
    Compute remaining cells in field (stream 2)
    Copy borders from GPU to host (stream 1)
    Send data to each neighbour (MPI)
    Wait for data from neighbours (MPI)
    Copy new borders into GPU (stream 1)
    Synchronize streams
end for
```

While this algorithm does work, we found that the performance was unpredictable when the number of hosts was increased. This performance data is presented in Section 5. However, a simple modification to this algorithm has provided faster and more reliable performance.

4.2. Algorithm B

This method uses a uni-directional communication method to exchange borders between GPU nodes. Rather than sending an receiving one border from each neighbour, this method sends data to only one neighbour and only receives data from the other. The amount of data sent is remains the same (each send is twice the size of Algorithm A) but the communication is simpler as the nodes are not trying to send and receive data from the same neighbour. The two methods of communication can be seen in Figure 3. The steps of this algorithm can be seen in Algorithm 3.

This method of uni-directional communication ef-

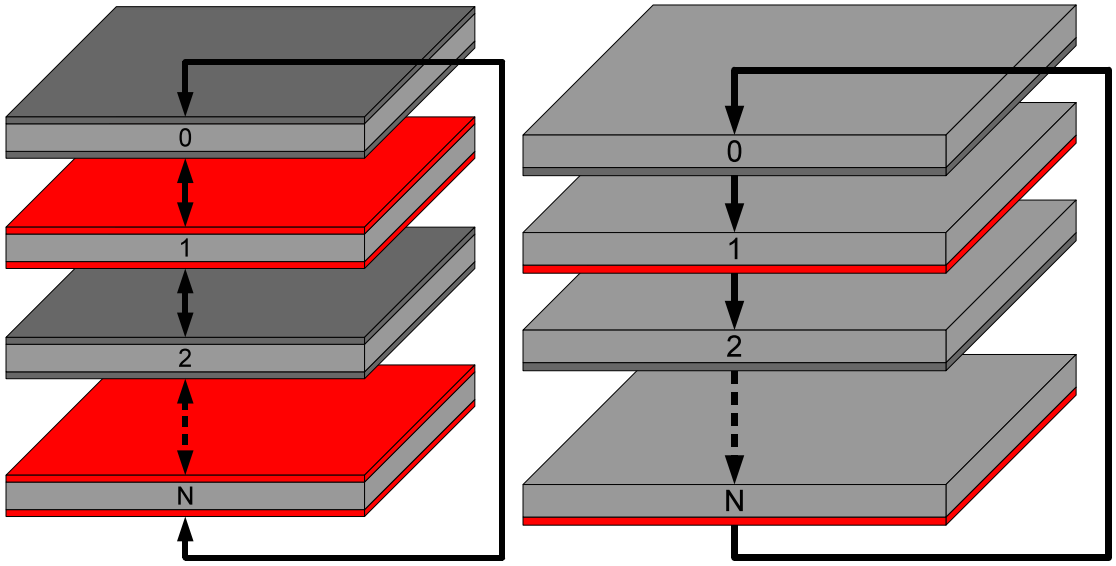

Figure 3. Border communication schemes between GPUs. Bi-directional communication as used by Algorithm A (left) and uni-directional communication as used by Algorithm B (right).

Algorithm 3 GPU cluster update algorithm using asynchronous memory communication and MPI.

```
for all steps do
    Compute border cells (stream 1)
    Compute remaining cells in field (stream 2)
    Copy border from GPU to host (stream 1)
    Send data to one neighbour (MPI)
    Wait for data other neighbour (MPI)
    Copy new border into GPU (stream 1)
    Synchronize streams
end for
```

fectively means that the field section each node is responsible is continuously moving through the field. By shifting the data each time-step to accommodate the incoming data, the field is still correct and valid. This method provides more reliable performance and better scalability as the number of cluster nodes increases. These results are presented in Section 5.

5. Results and Discussion

To compare these different architectures and algorithms we have measured their performance across a range of system sizes and configurations and compared these to single GPU implementation. First of all we have tested the performance of our multiple GPU, single host machine (as presented in [4]). This machine is a Intel i7 980X with 12GB of system memory hosting four GTX480s. The performance data of this machine (compared to single GPU implementations) are shown in Figure 4.

This implementation makes efficient use of all the GPUs and provides an almost linear speedup. However, it is limited to a maximum of four GPUs. Next we investigate how our bi-directional communication algorithm (algorithm A) performs on the GPU cluster. This data is shown in Figure 5.

As can be seen from the plot, this algorithm shows varying results and unreliable performance. The algorithm also does not scale well and some systems actually take longer with more nodes. However, this algorithm does allow a larger number of nodes to be utilised (our cluster is limited to 16) and thus allows larger systems to be simulated. Finally we compare our uni-directional communication algorithm (algorithm B) with single GPU data (See Figure 6).

It can be clearly seen that this algorithm provides much more reliable and scalable performance. While it does not quite reach almost linear speed-up attained by the single-host algorithm, it does give a consistent performance improvement and can be scaled to many more GPUs. Considering the increased latency due to network communication, this algorithm provides an efficient method of decomposing a finite-differencing equation across a GPU cluster.

Another point of importance is the maximum system size each architecture is able to support. Finite-differencing systems have a somewhat limited maxi-

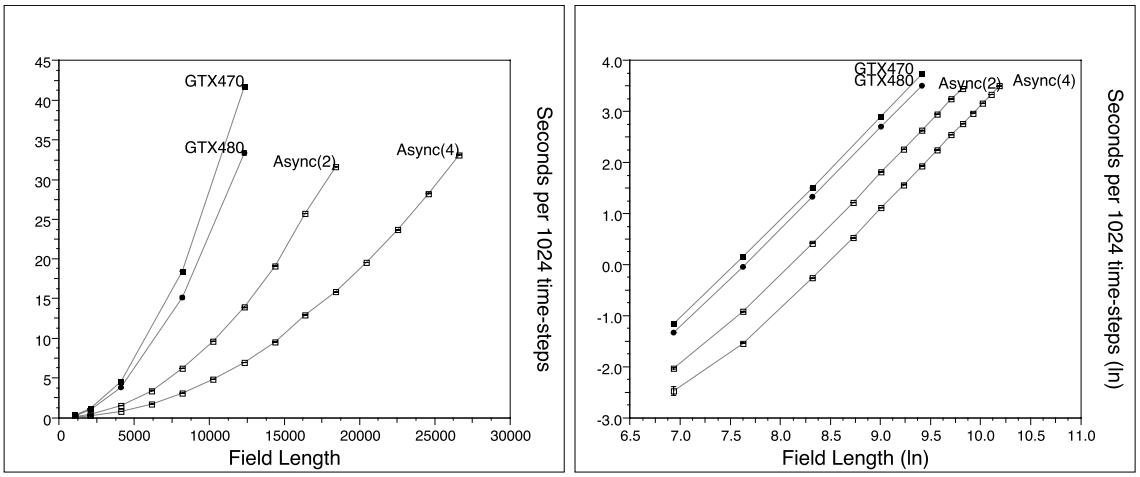

Figure 4. Comparison of single GPU data and multiple GTX480s hosted on a single node. Results shown on a linear scale (left) and ln-ln scale (right).

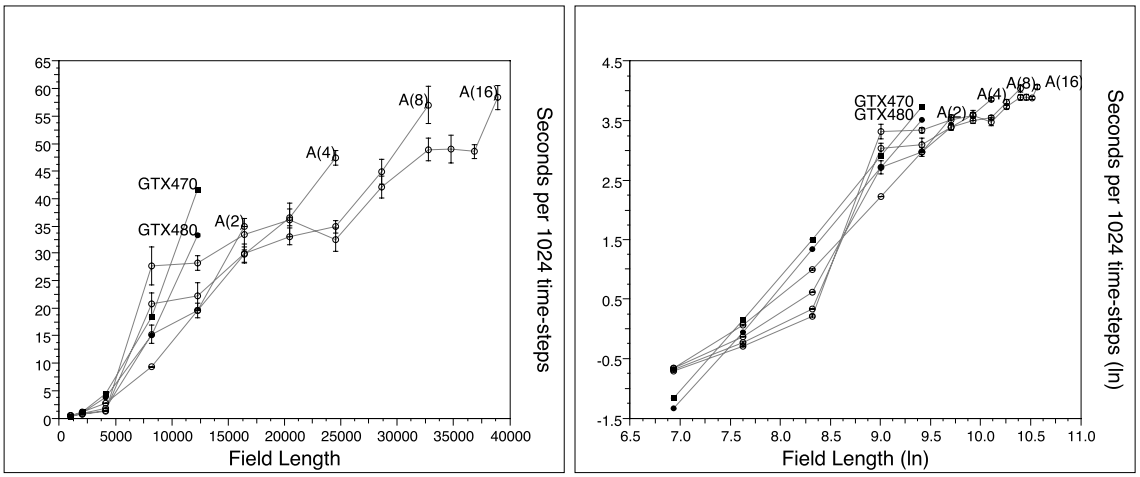

Figure 5. Comparison of single GPU data and 2, 4, 8 and 16 GTX470s on a distributed cluster using the bi-directional update algorithm (A). Results shown on a linear scale (left) and ln-ln scale (right).

mum system size due to the relatively small amount of device memory available on GPUs. The most GPUs that the field can be split between, the more device memory available and the larger the maximum system size is. Table 1 shows the maximum system size each test architecture is able to compute.

We have also investigated how these update algorithms perform for three-dimensional systems. The single-host algorithm provides the almost linear speedup as seen previously in the two-dimensional version. However, the GPU cluster algorithms are somewhat disappointing. The bi-directional communication algorithm showed the same unreliable results as experienced in two-dimensions but did provide a small

Table 1. Maximum system size that can be simulated on each machine configuration.

Architecture	Max System Size
GTX470	12288^2
GTX480	12288^2
GTX470x2	16384^2
GTX480x2	18432^2
GTX470x4	24576^2
GTX480x4	26624^2
GTX470x8	32768^2
GTX470x16	47104^2

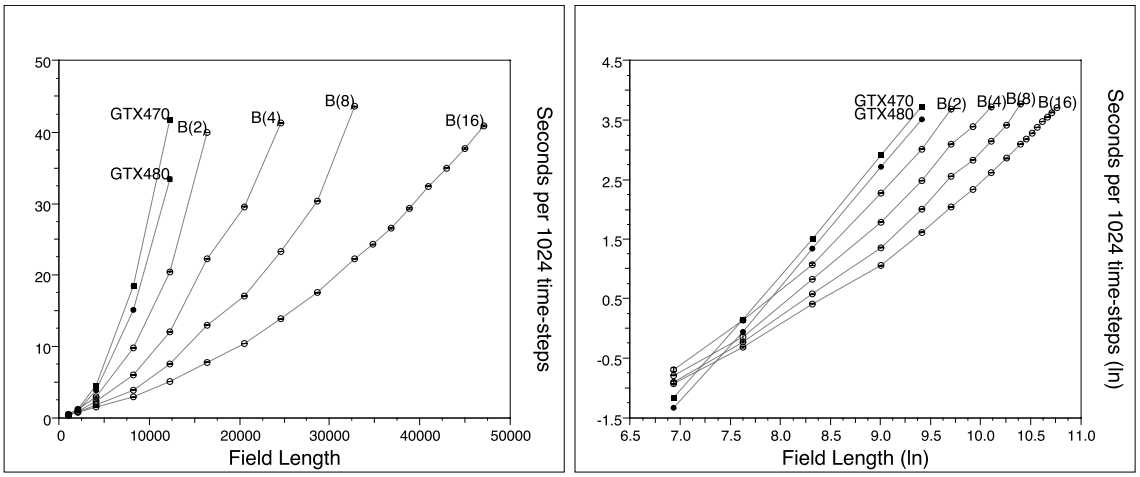

Figure 6. Comparison of single GPU data and 2, 4, 8 and 16 GTX470s on a distributed cluster using the uni-directional update algorithm (B). Results shown on a linear scale (left) and ln-ln scale (right).

speedup over a single GPU.

The uni-directional algorithm close to a 2x speedup over a single GPU with two nodes, however this performance gain does not scale. Simulations decomposed over 4, 8 and 16 nodes showing no further performance gain (and in some cases a slight performance loss). In three-dimensions the communication time outweighs the computational gain of more processing nodes.

The only real advantage decomposing a simulation over more cluster nodes is the maximum system size that can be computed. The single GPU (GTX480) implementation was limited to 448^3, four GPUs on a single host was capable of simulating a system of size 896^3 while 16 cluster nodes were capable of computing up to a system size of 1280^3.

6. Summary and Conclusions

We have reported on geometric domain decomposition results of a finite difference application problem (the Cahn-Hilliard partial differential equation) on a cluster of NVIDIA GPU-accelerated Intel CPUs. We have explored combinations of multiple GPUs on a single node as well as a cluster of single-GPU nodes. We have presented and compared two algorithms for managing communication between these nodes and compared their performance.

We have shown that the increased complexity involved when communicating across a network interconnect can cause unexpected and unreliable results as exhibited by the bi-directional communication update algorithm. Even minor changes such as the adoption of a uni-directional communication method can have drastic impact on the reliability and overall performance of the simulation.

The algorithm we have presented makes efficient use of distributed GPU nodes showing good scalability and improvements in both maximum system size and performance for two-dimensional simulations. It also shows a limited but tangible performance gain in three-dimensions but more importantly allows larger simulated systems to be computed than would be otherwise feasible in the memory of a single CPU/GPU combination.

GP-GPU computing has already offered a new lease of life to data-parallel computing as an accelerator for individual CPUs. We anticipate it will now continue to offer good means of accelerating cluster systems at the small to medium commodity priced range using the "gamer" grade GPUs we have discussed here as well as blade grade systems used in supercomputers.

Acknowledgments

Thanks to A.Leist for valuable assistance with configuring MPI.

References

[1] H. Meuer, E. Strohmaier, H. Simon, and J. Dongarra, "36th list of top 500 supercomputer sites," www.top500.org/lists/2010/11/press-release, November 2010.

[2] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK Benchmark: past, present and future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, 2003.

[3] D. Playne and K. Hawick, "Hierarchical and Multi-level Schemes for Finite Difference Methods on GPUs," in *Proc. CCGrid 2010, Melbourne, Australia*, no. CSTN-099, May 2010.

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

[5] A. Mitchell and D. Griffiths, *The Finite Difference Method in Partial Differential Equations*. Wiley, 1980, no. ISBN 0-471-27641-3.

[6] B. Alessandrini and V. Raganelli, "The propagation of acoustic and elastic waves in an heterogeneous discrete medium," *Eur.J.Mech., A/ Solids*, vol. 11, no. 4, pp. 519–538, 1992.

[7] R. F. Barrett, P. C. Roth, and S. W. Poole, "Finite difference stencils implemented using chapel," Oak Ridge National Laboratory, Tech. Rep. ORNL Technical Report TM-2007/122, 2007.

[8] P. Micikevicius, "3D finite difference computation on GPUs using CUDA," in *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, no. ISBN:978-1-60558-517-8, 2009.

[9] D. Egloff, "High Performance Finite Difference PDE Solvers on GPUs," QuantAlea Gmbh, Tech. Rep., 2010.

[10] S. E. Krakiwsky, L. E. Turner, and M. M. Okoniewski, "Acceleration of Finite-Difference Time-Domain (FDTD) Using Graphics Processor Units (GPU)," *IEEE MIT-S Digest*, vol. WEIF-2, pp. 1033–1036, 2004.

[11] S. Zainud-Deen, E. El-Deen, M. Ibrahim, K. Awadalla, and A. Botros, "Electromagnetic Scattering Using GPU-Based Finite Difference Frequency Domain Method," *Prog. in Electromagnetics Res. B*, vol. 16, pp. 351–369, 2009.

[12] V. L. Ginzburg and L. D. Landau, "(Published in English in Collected papers of L.D.Landau, Oxford Press, 1965, pp138-167)," *Zh. Eksp. Teor. Fiz.*, vol. 20, p. 1064, 1950, edited I.D. ter Haar.

[13] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1994, ISBN 0-262-57104-8.

[14] W. Gropp, E. Lusk, N. Doss, and A. Sjkellum, *A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard*. Argonne National Laboratories, 1996.

[15] *CUDA™ 3.1 Programming Guide*, NVIDIA® Corporation, 2010, last accessed August 2010. [Online]. Available: http://www.nvidia.com/

[16] W.-M. Hwu, C. Rodrigues, S. Ryoo, and J. Stratton, "Compute Unified Device Architecture Application Suitability," *Computing in Science and Engineering*, vol. 11, pp. 16–26, 2009.

[17] A. Leist, D. Playne, and K. Hawick, "Exploiting Graphical Processing Units for Data-Parallel Scientific Applications," *Concurrency and Computation: Practice and Experience*, vol. 21, pp. 2400–2437, December 2009, CSTN-065.

[18] O. Pentakalos, "An Introduction to the InfiniBand Architecture," http://www.oreillynet.com/lpt/a/1535, April 2002, last visited 2 Nov 2010.

[19] K. S. Perumalla and B. G. Aaby, "Data parallel execution challenges and runtime performance of agent simulations on gpus," in *SpringSim '08: Proceedings of the 2008 Spring simulation multiconference*. New York, NY, USA: ACM, 2008, pp. 116–123.

[20] J. W. Cahn and J. E. Hilliard, "Free Energy of a Nonuniform System. I. Interfacial Free Energy," *The Journal of Chemical Physics*, vol. 28, no. 2, pp. 258–267, 1958.

[21] K. A. Hawick and D. P. Playne, "Modelling and visualizing the Cahn-Hilliard-Cook equation," in *Proceedings of 2008 International Conference on Modeling, Simulation and Visualization Methods (MSV'08)*, Las Vegas, Nevada, July 2008.

[22] D. Playne and K. Hawick, "Data Parallel Three-Dimensional Cahn-Hilliard Field Equation Simulation on GPUs with CUDA," in *Proc. 2009 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'09) Las Vegas, USA.*, no. CSTN-073, 13-16 July 2009.

[23] A. A. Abrikosov, "Ginzburg-Landau equations for the extended saddle-point model," *Phys. Rev. B*, vol. 56, no. 1, pp. 446–452, Jul 1997.

[24] K. A. Hawick and D. P. Playne, "Numerical Simulation of the Complex Ginzburg-Landau Equation on GPUs with CUDA," Massey University, Tech. Rep. CSTN-070, January 2010, to appear in Proc. IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN), 14-16 Feb. 2011, in Innsbruck, Austria.

[25] PCI-SIG, "PCIe Express Base Specification 1.1," http://www.pcisig.com/specifications/pciexpress/base, November 2010.

[26] K. A. Hawick, "Domain Growth in Alloys," 1991, Edinburgh University, Ph.D. Thesis.

[27] D. A. Reed, L. M. Adams, and M. L. Patrick, "Stencils and problem partitionings: Their influence on the performance of multiple processor systems," *IEEE Transactions on Computers C*, vol. 36, no. 7, pp. 845–858, jul 1987.