Parallel Acceleration with GPUs for High Performance Applications

K.A. Hawick and D.P. Playne

Computer Science, Massey University, Albany, North Shore 102-904, Auckland, New Zealand

http://complexity.massey.ac.nz

Graphical Processing Units (GPUs)

Graphical Processing Units (GPUs) and other devices have proved a valuable mechanism to speed up many applications using a data-parallel programming model that can accelerate a conventional CPU. It is becoming economically viable to host multiple GPUs on a single CPU host node, and clusters using this multi-GPU assisted node model are becoming prevalent. We are experimenting with the hosting of up to eight GPUs on a single CPU using PCI bus extension technology and report on the attainable performance of a range of simulation and complex systems applications.

We are exploring a range of combinations of multi-core CPUs, running various thread management software systems to manage their multiple GPU accelerators. We anticipate significant flexibility and scalability achievable using this approach and believe it has major implications for future generation HPC systems including clusters and supercomputer facilities.

Many of the supercomputers in the present Top 500 list[1] now incorporate graphical processing unit devices as accelerators for the individual nodes. Such systems use fast interconnection technologies such as InfiniBand or a proprietary technology instead of regular Ethernet switching technology. These interconnects aim at providing fast and low latency interconnections between individual nodes, each of which will typically run a full operating system software stack. Applications programmers will typically use a software model such as message passing - implemented with for example MPI or a variant - to communicate between participating nodes in a single MIMD program.

Individual nodes however can be accelerated by one or more devices. Although other accelerators such as customised field programmable gate arrays are sometimes used, at the time of writing GPUs are becoming a very popular acceleration devices. Typically a hosting CPU node is accelerated by one or more GPU devices which are connected to the CPU using PCI or PCI Express bus technology.

Applications programmers invoke special codes on the GPU that are called from the normal CPU program. The GPU typically offers a very large number of simpler cores than those found on a CPU. The parallelism model most likely used on GPUs is that of fine grained data parallelism whereby very many threads are used to manage data-parallel operations. Modern GPUs often are structured with some shared facilities such as floating-point units(FPUs) – not all the individual cores will have an FPU and they are typically grouped together so that for example some sort of multi-processing unit with floating point capability acts to group together individual simpler cores.

The CPU hosting program can in fact be multiple threaded, and this approach is often used to make use of the multiple cores that a modern CPU will typically incorporate. It is also possible for these cores to run more than one thread and this approach is often favoured if there are slow memory or data access operations required. A single hardware CPU core will quite often be given two or more applications threads to run to allow data accesses to be suitably interleaved with computations. Modern CPUs will often have special extra hardware capabilities to explicitly support this approach.

We therefore find that there are several hierarchical layers of parallelism available to exploit on a multi-noded, multi-cored, GPU-accelerated supercomputer system. Different applications and user schedules can make use of these in many different manners to exploit the overall hardware resource to best effect. The following terminology is useful to describe this hierarchy of levels of parallelism, where we give some references to our recent work against each relevant category:

Grid: Distributed and often separately owned supercomputing resources can be used [2]

Node: Individual Nodes connected with InfiniBand or Ethernet switching technology [3, 4]

Multi-CPU: Multiple CPUs on multi-socket motherboard share node main memory [5, 6]

CPU: Individual CPU has multiple cores of high individual capability including their own floating point unit [7, 8]

CPU-Core: CPU core can run more than one application thread [9, 10]

GPU-Device: CPU thread can be accelerated by a GPU device [11, 12]

GPU-MP: GPU device has some number of individual multi processors, usually with their own floating point unit and controlling some group of lower-level simple cores [13, 14]

GPU-Core: Individual fine-grained accelerator cores carry out data-parallelism operations [15, 16]

There is potential scope for work at all levels of this hierarchy. Practicably, the financial resources required to work at the high end of supercomputing are now restricted to a very few institutions. Nevertheless it is quite feasible to progress systems software, library development, applications development and other parallel computing software research anywhere.

A particularly interesting area is that of automated parallel code generation. The notion of expressing the algorithm and application in a high level domain-specific language and subsequently using tools to generate MPI code, threading code, GPU CUDA code or some other platform specific implementation is an attractive one that can leverage the increasing cost of programmer effort in exploiting parallel computer systems.

Complex systems simulations often require very high performance computing resources either to enable the simulation of large enough models to exhibit multi-length scale complexity and emergence, or just to obtain adequate statistical averaging to support quantitative conclusions and comparisons with theoretical predictions. Simulation of large scale models that exhibit phase transitions and other complex phenomena is the main driver behind our group's work on parallel computing.

Multiple GPUs (mGPU)

While GPU devices provide a very powerful computing architecture, the performance of a single GPU is still far too limited for most high-performance computing applications. The logical solution for these applications is to distribute the computational task between multiple GPU devices. Computing on multiple GPU devices or mGPU involves solving a computational task on more one GPU that can be hosted on the same host machine or different hosts connected by a network.

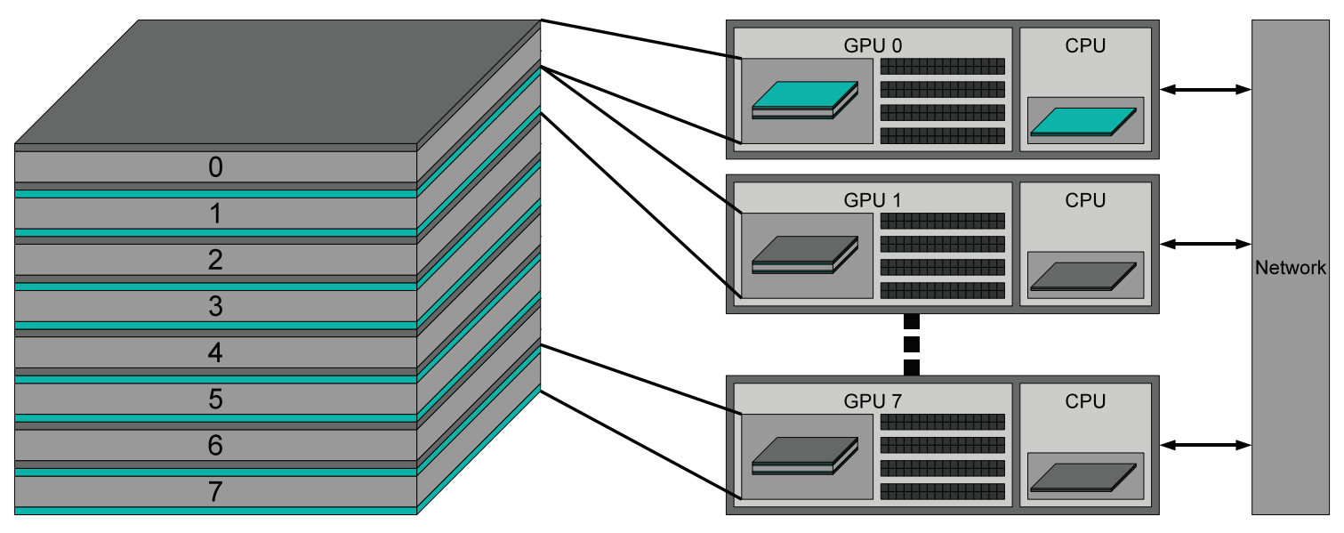

Figure 1: A lattice split between eight GPU devices hosted by separate nodes connected by a network.

The advantages of this approach is two-fold - firstly the number of GPU cores available to process the computation is increased and secondly the amount of available memory is increased. Many GPU applications are limited by the relatively small amount of memory available on each device. By splitting a problem between different devices, the total amount of device memory is increased. The down-side is that, for most applications, some amount of communication between devices is necessary. The amount of communication will depend on the nature of the application, for example the lattice shown in Figure 1 is split between eight GPUs with the boundaries of each lattice segment highlighted. Each time-step these boundaries must be communicated to the neighbouring devices.

mGPU Systems

Graphics cards are connected to their host by the PCIe bus, GPUs will operate best when connected to a PCIe x16 slot. Some motherboards have four such PCIe slots and can host four cards. As there are dual-GPU graphics cards such as the GTX295, GTX590 or GTX690 a single host is capable of hosting up to eight GPUs. Figure 2 shows a single host connected to four single-GPU devices.

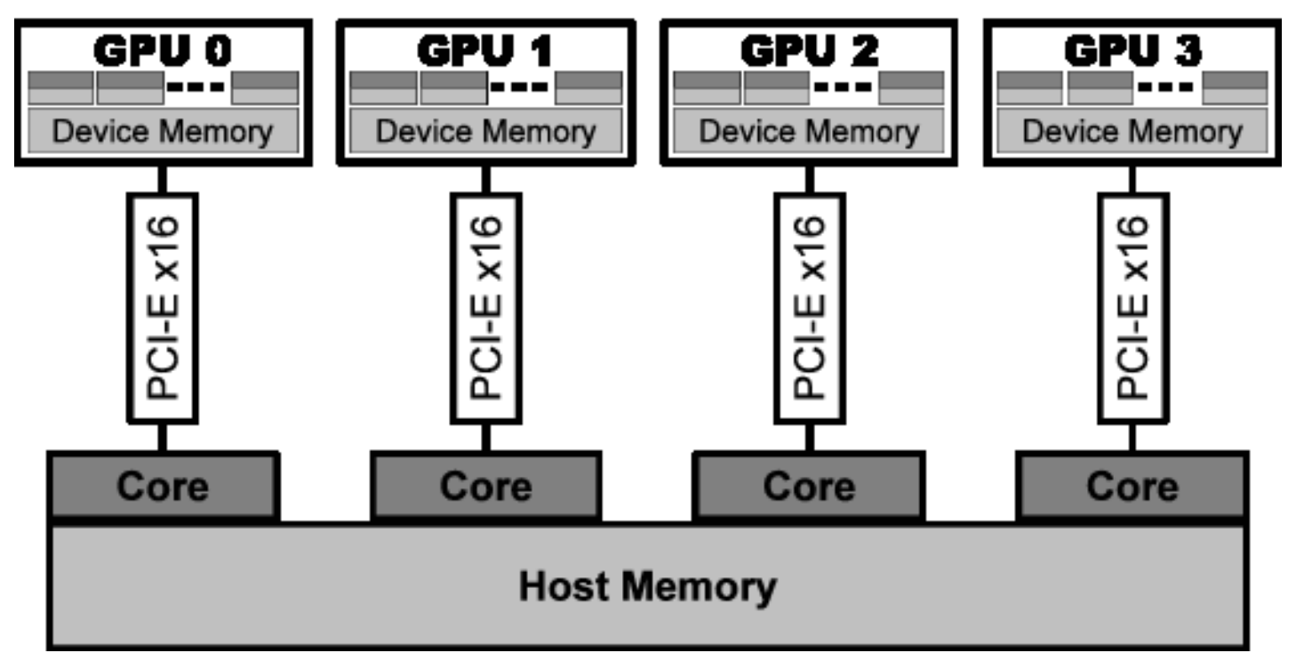

Figure 2: Four GPU Devices connected to a single host.

Another way to host multiple devices from a single machine is to host them in a PCIe chassis and connect them to the host through a PCIe extender card. The Dell C410x is one such PCIe extender chassis which is capable of hosting sixteen Tesla computing cards (See Figure 3). A host equipped with a PCIe Host Interface Card (HIC) can connect to this chassis and be connected to 1-8 of the device in the chassis. While it is theoretically possible for a host with multiple HIC cards to connect to all sixteen GPU devices, in practice this has proved problematic due to unidentified hardware/driver limitations.

Figure 3: Dell C410x PCIe chassis.

The final configuration for mGPU systems is to host the devices in distributed nodes connected via a network. This type of mGPU system is more extensible as additional nodes can easily be added to the network. However, the downside of such a system is the increased communication time between devices. GPUs hosted by the same machine are connected to the same PCIe bus and can communicate with the host CPU or other devices through this bus. However, when GPU devices are hosted by separate machines this communication must go through the relatively slow network. This architecture is shown in Figure 4.

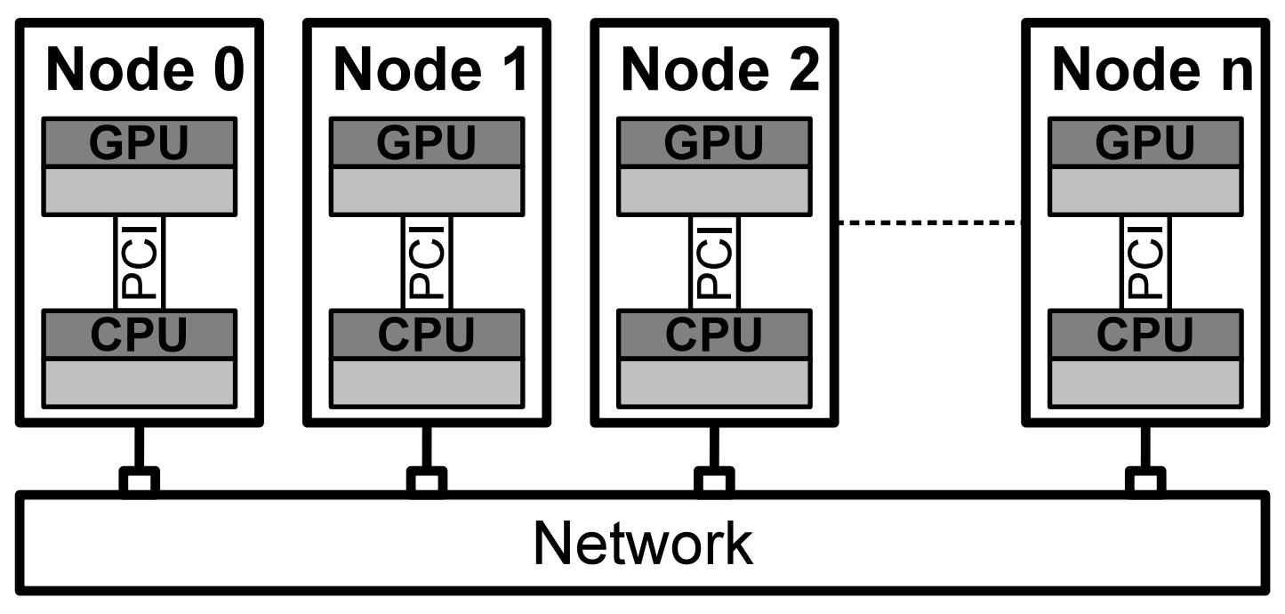

Figure 4: A cluster of GPU accelerated nodes connected by a network.

mGPU Communication

Any interesting mGPU application inevitably requires communication between the devices. If the mGPU system is a distributed architecture there will be an additional transfer cost if the two devices involved in the transfer are hosted on different nodes. The impact of this communication will depend on the nature of the application, the computation to communication ratio and the type, architecture and speed of the network.
Prior to the release of the Fermi architecture GPUs all communication between devices took place through the host memory. In this communication method, any data transfer requires two separate memory transactions. Initially copying from the first device to an area of host memory and the second copying to the second device. This transfer method is shown in Figure 5.

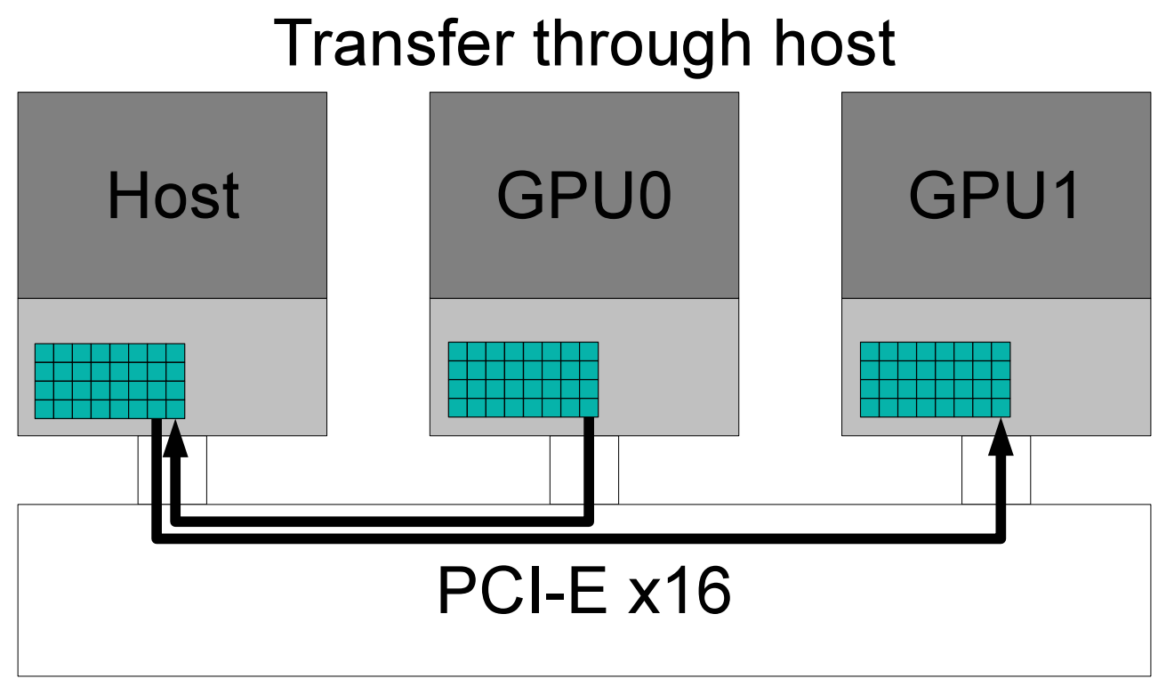

Figure 5: Data transfer between two GPU devices through host memory.

The new GPUDirect 2.0 technology supported by Fermi architecture GPUs allows the unnecessary extra transfer through host memory to be eliminated. It allows data to be transferred directly from one GPU device to another through the PCIe bus. This data transfer method is shown in Figure 6. It should be noted that although this data transfer does not involve the host memory it is still initiated by a function call from the host.

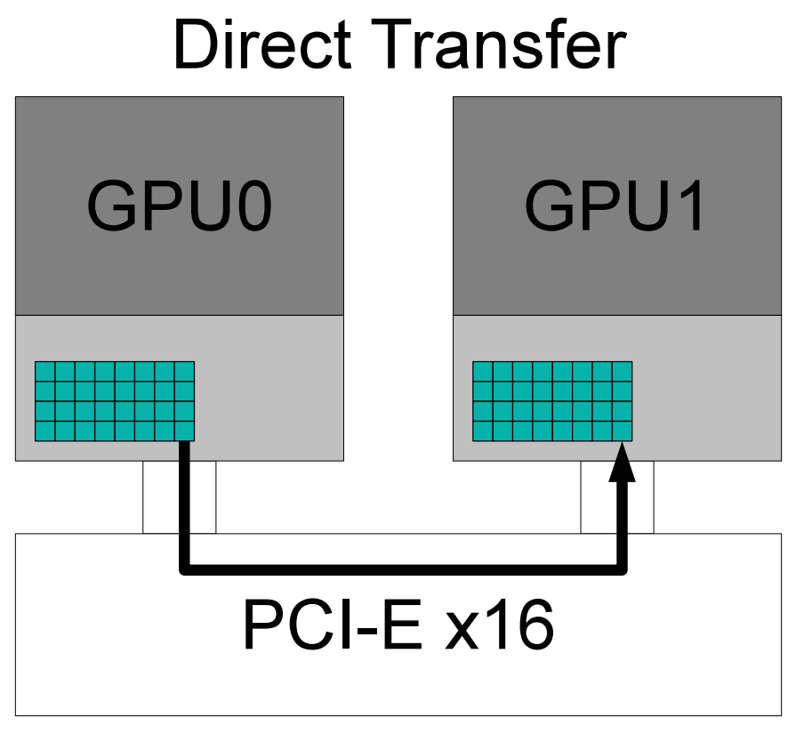

Figure 6: Direct Transfer between two GPU devices hosted on the same PCIe bus.

The other option for inter-device communication that GPUDirect 2.0 provides is called Direct Access. With this communication method the host is not involved in the data transfer at all. Rather than a memory copy being initiated by the host to copy a segment of data from one device to the other, Direct Access allows the threads on one device to access data stored in the memory of another device. This memory access method with a thread on GPU1 reading a single memory location from GPU0 is shown in Figure 7.

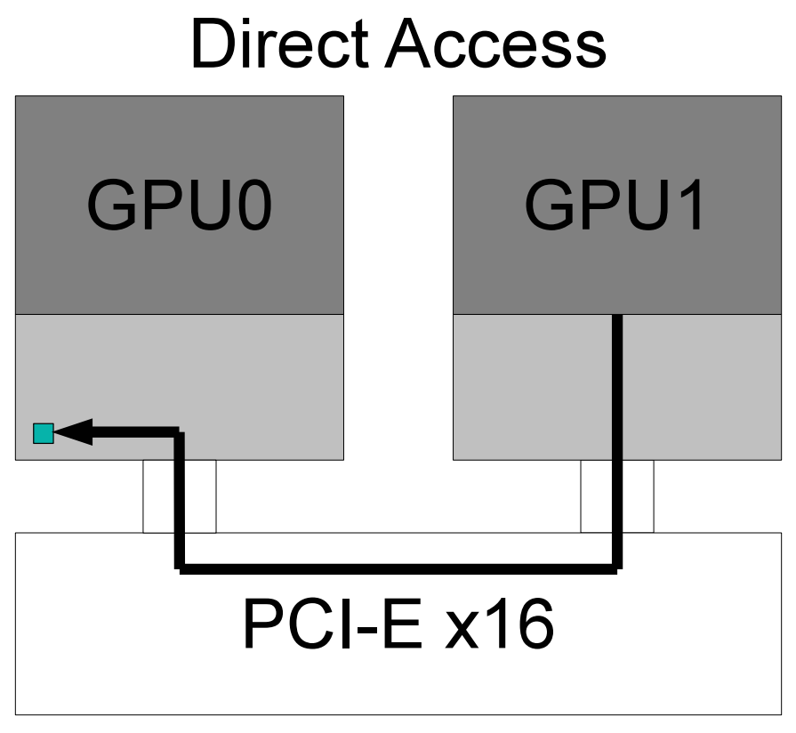

Figure 7: Direct memory access between two GPUs on the same PCIe bus.

The Direct Transfer and Direct Access methods provided by GPUDirect 2.0 can also be used by distributed mGPU systems if they are connected together by InfiniBand connections from certain providers. The communication between GPU device on different nodes will obviously still be slower than two devices on the same node but it helps to eliminate non-essential data copying through the host memory.

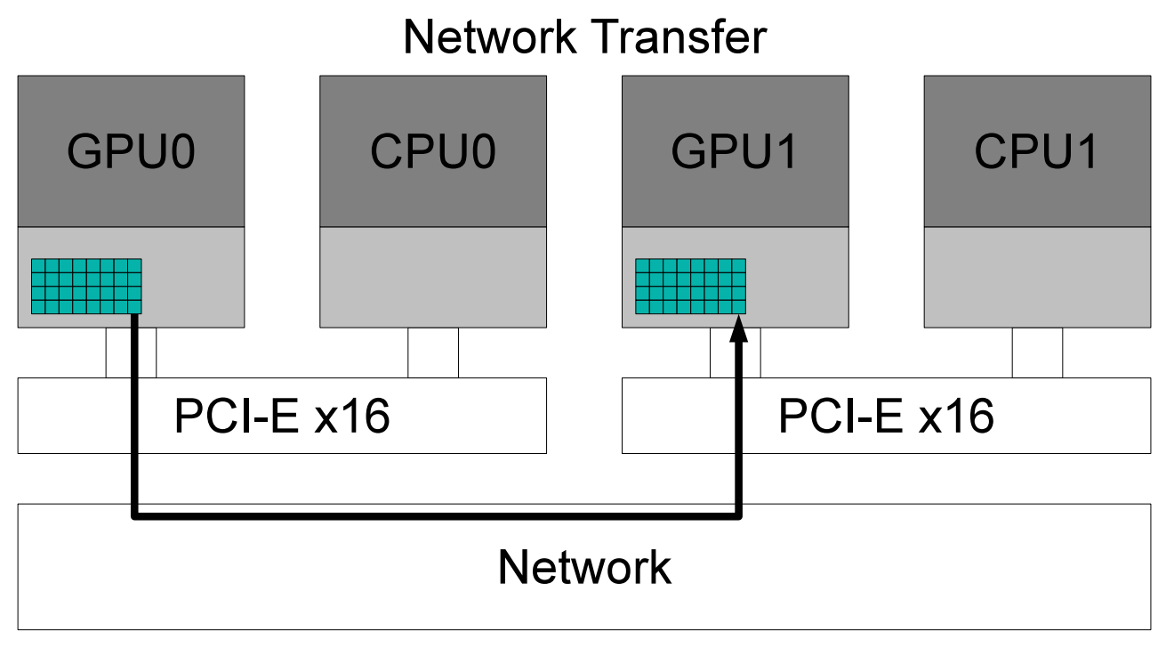

Figure 8: Direct data transfer between two GPU devices across a network.

The mGPU communication methods provided by GPUDirect 2.0 have significantly reduced the effort required to develop an efficient mGPU application. Prior to this direct communication, such applications required on more complicated communication schemes that overlap communication and computation in order to hide latencies. For some applications this may still be required for optimal performance but will also benefit from direction data transfer. Over the last several years the number of applications using GPUs for computation has been adopted by a wide-range of application developers. From computer games to major scientific applications, GPUs have been employed to provide major performance improvements.

Discussion

The recent major improvements in mGPU communication will help significantly in the transition of these applications to make use of multiple GPUs. Given that most modern motherboards are capable of supporting multiple graphics cards or a single graphics card with multiple GPUs and the increasing number of supercomputers equipped with GPUs, it can be expected that the number of mGPU applications will continue to grow in number.

Of particular note is the performance of the direct access method of mGPU communication which can provide the best performance for small transfers and for large transfers it cause a drop in performance of approximately 20% drop compared to a more complex asynchronous Direct Transfer. This is a remarkable result for this type of Non-Uniform Memory Access or NUMA style communication.

This result can be seen in Figure 9 which shows six different methods of data transfer. The plot shows a data exchange where two devices each send the other a block of data. This transfer can be synchronous (s) where the two transfers are performed one after the other, or asynchronous where both transfers take place at the same time. The first method is the transfer through host (DHD) using paged (pag) and pinned host memory (pin), the second is the Direct Transfer (DD) and finally Direct Access (DA). Asynchronous Direct Transfer and Direct Access methods provide the best performance with Direct Access providing the best performance for small transfers and Asynchronous Direct Transfer providing the best performance for large transfers.

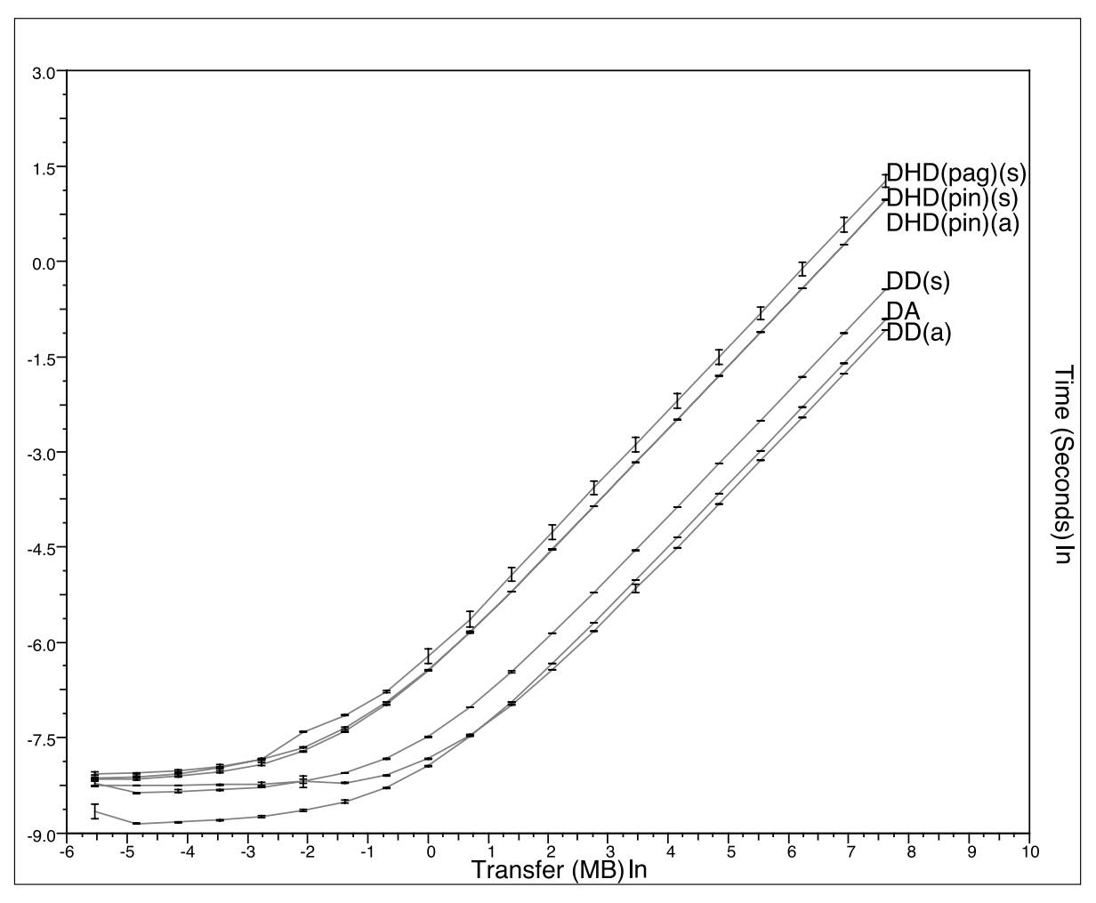

Figure 9: Two devices with Transfer through Host (DHD), Direct Transfer (DD) and Direct Access (DA). Data for synchronous (s) and asynchronous (a).

One phenomena that may develop is the number of users, especially computer games enthusiasts, holding onto older graphics cards and using them to accelerate computation. Old graphics cards that are not capable of running the latest games with the best graphics may be recycled as a secondary computation card used to simulate physics engines or other computational tasks in a computer game. This recycling of GPUs would extend the usable lifespan of the device and improve the performance of modern applications without impacting graphics performance.

Further Information

Further information on our work in these areas is available at: http://complexity.massey.ac.nz.
You can learn more about parallel computing and modern parallel programming languages and other technologies such as: CUDA; MPI; OpenMP; OpenCL; TBB; pThreads;... in Massey's Computer Science courses:

159.335 Concurrent Programming & Operating Systems

159.735 Parallel & Distributed Systems

159.732 Advanced programming & Simulations

References

[1] TOP500.org, "TOP 500 Supercomputer Sites." http://www.top500.org/. Last accessed November 2010.

[2] K. A. Hawick and H. A. James, "Simulating a computational grid with networked animat agents," in *Proc. Fourth Australasian Symposium on Grid Computing and e-Research (AusGrid 2006)* (R.Buyya and T.Ma, eds.), CSTN-028, (Hobart, Australia), pp. 63–70, January 2006. ACSW Frontiers 2006, ISBN 1-920-68236-8, ISSN 1445-1336.

[3] K. A. Hawick, H. A. James, and C. J. Scogings, "64-bit architectures and compute clusters for high performance simulations," tech. rep., Information and Mathematical Sciences, Massey University, Albany, North Shore 102-904, Auckland, New Zealand, April 2006.

[4] D. Playne and K. Hawick, "Auto-generation of parallel finite-differencing code for mpi, tbb and cuda," in *Proc. International Parallel and Distributed Processing Symposium (IPDPS); Workshop on High-Level Parallel Programming Models and Supportive - HIPS 2011*, (Anchorage, Alaska, USA), pp. 1163–1170, IEEE, 16-20 May 2011. in conjunction with IPDPS 2011, the 25th IEEE International Parallel & Distributed Processing Symposium.

[5] M. Gan-El and K. A. Hawick, "Parallel containers - a tool for applying parallel computing applications on clusters," Research Letters in the Information and Mathematical Sciences ISSN 1175-2777, Information and Mathematical Sciences, Massey University, Albany, North Shore 102-904, Auckland, New Zealand, May 2004.

[6] K. Hawick and D. Playne, "Spectral and real-space solid representations and visualisation of real and complex field equations," Tech. Rep. CSTN-101, Computer Science, Massey University, August 2009.

[7] K. A. Hawick, A. Leist, and D. P. Playne, "Mixing Multi-Core CPUs and GPUs for Scientific Simulation Software," Tech. Rep. CSTN-091, Computer Science, Massey University, September 2009.

[8] A. Leist, K.A.Hawick, and D.P.Playne, "Gp-gpu and multi-core architectures for computing clustering coefficients of irregular graphs," in *Proc. International Conference on Scientific Computing (CSC'11)*, no. CSC2720, (Las Vegas, USA), July 2011.

[9] K. A. Hawick and D. P. Playne, "Automated and parallel code generation for finite-differencing stencils with arbitrary data types," in *Proc. Int. Conf. Computational Science, (ICCS), Workshop on Automated Program Generation for Computational Science, Amsterdam June 2010.*, no. CSTN-106, December 2010.

[10] D. P. Playne and K. A. Hawick, "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.

[11] 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.

[12] K. A. Hawick and D. P. Playne, "Numerical Simulation of the Complex Ginzburg-Landau Equation on GPUs with CUDA," in *Proc. IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN)*, no. CSTN-070, (Innsbruck, Austria), pp. 39–45, IASTED, 15-17 February 2011.

[13] K. Hawick, D. Playne, and M. Johnson, "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.

[14] D. Playne and K. Hawick, "Job parallelism using graphical processing unit individual multiprocessors and highly localised memory," Tech. Rep. CSTN-159, Computer Science, Massey University, June 2012.

[15] K. A. Hawick and D. P. Playne, "Hypercubic Storage Layout and Transforms in Arbitrary Dimensions using GPUs and CUDA," *Concurrency and Computation: Practice and Experience*, vol. 23, pp. 1027–1050, July 2011.

[16] A.Gilman and K. Hawick, "Field programmable gate arrays for computational acceleration of lattice-oriented simulation models," Tech. Rep. CSTN-151, Computer Science, Massey University, 2012. accepted for and to appear in Proc. International Conference on Computer Design (CDES'12), 16-19 July, 2012, Las Vegas, USA.