Modelling Multi-GPU Systems [1]

Daniele G. SPAMPINATO [a] , Anne C. ELSTER [a] and Thorvald NATVIG [a]
[a] *Norwegian University of Science and Technology (NTNU), Trondheim, Norway*

Abstract. Due to the power and frequency walls, the trend is now to use multiple GPUs on a given system, much like you will find multiple cores on CPU-based systems. However, increasing the hierarchy of resource widens the spectrum of factors that may impact on the performance of the system. The goal of this paper is to analyze such factors by investigating and benchmarking the NVIDIA Tesla S1070. This system combines four T10 GPUs, making available up to 4 TFLOPS of computational power. As a case study, we develop a red-black, SOR PDE solver for Laplace equations with Dirichlet boundaries, well known for requiring constant communication in order to exchange neighboring data. To aid both design and analysis, we propose a model for multi-GPU systems targeting communication between the several GPUs.
The main variables exposed by our benchmark application are: domain size and shape, kind of data partitioning, number of GPUs, width of the borders to exchange, kernels to use, and kind of synchronization between the GPU contexts. We show that the multi-GPU system greatly benefits from using all its four GPUs on very large data volumes. Four GPUs were almost four times faster than a single GPU. The results also allow us to refine our static communication model.

Keywords. GPU computing, multi-gpu, performance modelling, NVIDIA s1070

1. Introduction

GPU computing for high performance computing is generating a lot of interest. In this paper, we investigate multi-GPU systems' performance factors, such as data volume dimensions, data partitioning techniques, number of GPUs, inter-GPU communication methods, and kernel design. In particular, we focus on NVIDIA's S1070 multi-GPU solutions, a system recently deployed at HPC centers world wide. These include Tokyo Technology University's Tsubame supercomputer which was ranked 29^{th} in the world when installed, and the new multi-GPU-based system at GENCI in France. Our methodology and general results should, however, be applicable to most modern multi-GPU systems.

The increasing hierarchy of resources issues new challenges to the developers, widening the spectrum of factors which may impact the performance of a multi-GPU system. The aim of this work is to investigate such factors and consider some important models of parallel systems in order to identify some common properties that can help us in our study. Communication is always a relevant aspect when dealing with distributed resources. By designing a benchmark framework around the SOR PDE solver, an application that constantly requires inter-GPU communication, we are able to develop better a multi-GPU model.

[1] A big thank you to NVIDIA for sponsoring our HPC-Lab with cutting-edge GPUs.

2. Current NVIDIA Architecture and Programming Model

The compute unified device architecture (CUDA) environment presents a cutting-edge programming model well-suited for modern GPU architectures [1]. NVIDIA developed this programming environment to fit to the processing model of their Tesla architecture and expose the parallel capabilities of GPUs to the developers.

CUDA maintains a separated view of the two main actors involved in the computation, namely the host (CPU system) and the device (GPU card/system). The host executes the main program, while the device acts like a coprocessor.

Our test bench, the **NVIDIA Tesla S1070 Computing System** is a 1U rack-mount system equipped with four Tesla T10 GPUs (Figure 1).

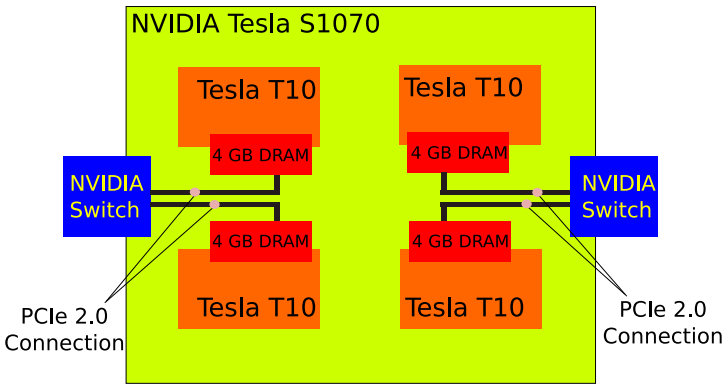


Figure 1. NVIDIA Tesla S1070 Computing System Architecture.

The Tesla T10 has 240 processing cores working either at 1.296 GHz (-400 configuration) or at 1.44 GHz (-500 configuration). The cores are grouped in 30 Streaming Multiprocessors (SMs), each with 8 single-precision cores and a single double-precision core. Each single-precision core is able to issue up to 3 FLOP per cycle, i.e. a *multiply* concurrently to a *multiply-add*, while the double-precision core is able to issue up to 2 FLOP per cycle. This gives a peak theoretical performance of $4\ \text{GPUs} \cdot 1.44\ \text{GHz} \cdot 3\ \text{FLOP/cycle} \cdot 240\ \text{cores} = 4.147\ \text{TFLOP/s}$ in single precision and $4\ \text{GPUs} \cdot 1.44\ \text{GHz} \cdot 2\ \text{FLOP/cycle} \cdot 30\ \text{cores} = 345\ \text{GFLOP/s}$ in double precision.

From a memory point of view, every GPU is connected to 4 GB high speed DRAM, with a bandwidth of 102 GB/s. This gives to the system a total of 16 GB. The connection to the host passes through NVIDIA Switches and PCIe Host Interconnection Cards (HIC). A single PCIe 2.0 16x (or 8x) slot on the host is connected to two GPUs using an NVIDIA Switch and a PCIe HIC. This connection provides a transfer rate of up to 12.8 GB/s between the host node and the computing system.

3. Programming Multiple GPUs

The NVIDIA CUDA Runtime API gives the programmer the possibility to select which device to execute the kernels on. By default device 0 is used, and the devices are enumerated progressively.

To use multiple CUDA contexts, we can associate them to different CPU threads, one for each GPU. For optimal performance, the number of CPU cores should not be less than the number of GPUs in the system. Managing the threads could be done by implementing an ad-hoc communication layer through system libraries, such as NPTL. Otherwise, existing libraries could be used, such as message-passing libraries adapted to perform shared memory communication [2].

4. Parallel Models

As argued in [3], parallel models often lack of connection to the real world, becoming powerless tools in terms of prediction capabilities. Nonetheless, they do not loose their relevance when analyzing new architectures, because they help in focusing on the main characteristics exposed by the systems at issue.

SMP clusters are one such example where the computing platform combines elements from both message-passing multicomputers and shared memory multiprocessors.

4.1. Similarities between SMP Clusters and Multi-GPU Systems

A multi-GPU computing architecture has several characteristics that make it similar to the computing model of an SMP cluster. E.g. Eicker and Lippert [9] sheds some ligtht on the JULI cluster architecture. As shown in Figure 2, every GPU, which in turn is a highly multithreaded system, is linked to every other through the host system. This kind of interconnection, requires communication to setup cooperation in solving a common problem. Because of the difference between a general processor and a graphics processor, some of the concepts need a proper contextualization, posing sometimes new problems to solve.

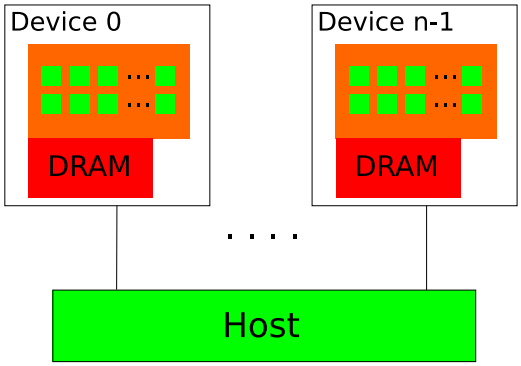


Figure 2. Multi-GPU system as a network of GPUs.

4.2. Synchronization and Consistency

Synchronization is suggested exclusively at thread level. Threadblocks' independence is an important requirements for granting scalability and speed. Also within a threadblock, however, the use of synchronization routines must be carefully controlled and not mis-

used. Mutual exclusions within a threadblock, can be implemented on recent hardware[2] using a combination of barriers and atomic operations, but this goes against the requirement of massive data parallelism required by GPUs to be efficients. Not accidentally, CUDA does not provide any native solution to this kind of approach.

4.3. Memory Performance

Global memory is not cached, and accessing it is an expensive operation, so it is important to design an appropriate access pattern. The datatype must be 4, 8 or 16 bytes large, and must be aligned to a multiple of its size.

There is a separate read-only constant memory, and access to this is cached. Optimal performance is achieved when all the threads of a half-warp read from the same address. Texture memory, which uses the 2D texture unit of the graphics hardware, provide 2D space locality for cache.

4.4. Inter-GPU Communication

We propose a model for a multi-GPU system that extends the Hockney model [5]. Taking into account the description in Section 4, a multi-GPU system is the combination of an interconnection node and a set of GPUs,

$$T_{multi-GPU} = T_{node} + \max_{g \in G} \left(T^{g}_{kernel} + T^{g}_{GPU-GPU} \right) , \tag{1}$$

where T_{node} models the specific node (e.g. a node of a multicomputer), and the max operator is motivated by the fact that several GPU contexts are executed in parallel. $T_{GPU-GPU}$ expresses the communication between two GPUs, and, supposing identical connections to the host for both the devices, we can define it as

$$\begin{aligned} T_{GPU-GPU} &= T_{GPU-host} + T_{host-host} + T_{host-GPU} \\ &= 2 \cdot T_{GPU-host} + T_{host-host} \, . \end{aligned} \tag{2}$$

The host-to-host communication time $T_{host-host}$, summarizes the time required by all the data movements and synchronization mechanisms part of the communication flow between two GPUs.

The transfer bandwidth between host and device must be carefully considered. Compacting many small transfers into a large one is often much more efficient. A way to increase the bandwidth is to use page-locked memory. This would prevent paging mechanisms from being used on those memory spaces where data have been allocated. Page-locked memory, however, must be used with care. Reducing memory resources may produce system slowdown side effects. Pinned memory introduces a higher startup time relevant for small transfers [4].

Also the $T_{host-host}$ term should be minimized. This can be done exploiting the real parallelism exposed by modern multicore processors. As reported in [2], the use of MPI to communicate between processes on the same node can result in improvable communication overhead. The paper mentions that 2/3 of the communication is spent in buffered

[2] Atomic operations are just implemented on devices of compute capability 1.1 or higher.

MPI_Sendrecv. MPI libraries usually are based on inter-process communication. Using threads that share the same address space could turn out more beneficial. Thus, as an alternative to the message-passing approach, multithreading with an appropriate synchronization can be used to implement on-node communication among threads associated to different GPU contexts.

5. Benchmarks

The effect of using more than one GPU is shown in Figure 5 just for the texture based case. The graph underline that involving the most of the GPUs available is always an appropriate decision since from small dimensions of the domain ($N \times N > 3000 \times 3000 \approx 36$ MB). The curve shows that using four GPUs can be up to 3.4X faster than using only one. We have also found that it is 1.8X faster than using two [6].

After $N = 16000$ however, the curves start exhibiting a drastic reduction in performance. We think that this effect can be produced by resource contentions. As described in Section 2, GPUs on the S1070 share pairwise the two PCIe channels. It is possible that, with a relevant traffic ($> 2GB$), the PCIe contention lowers the performance of two parallel transfers almost down to the performance of a single one. As a result, the execution time required by N GPUs gets closer to the time required by a subset of N.

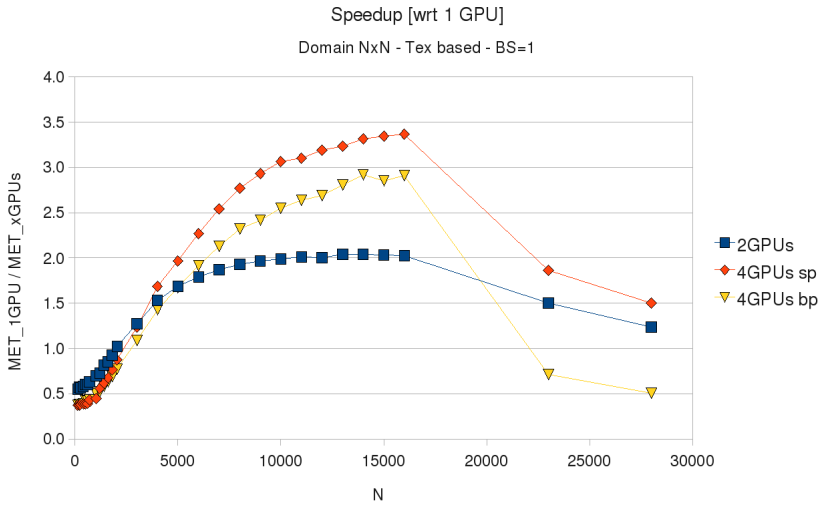


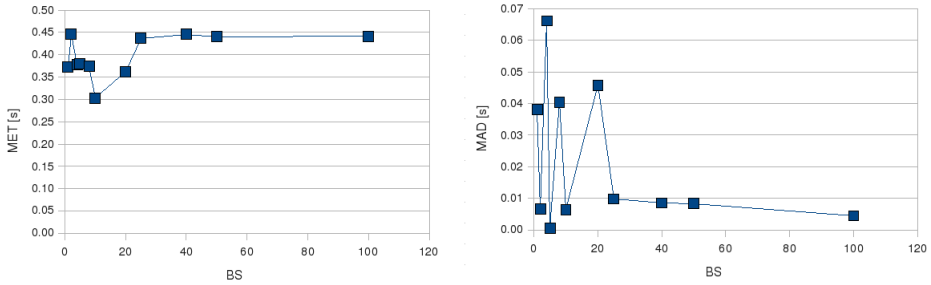
Figure 3. Speedups with respect to one GPU varying number of GPUs.

5.1. Border Size

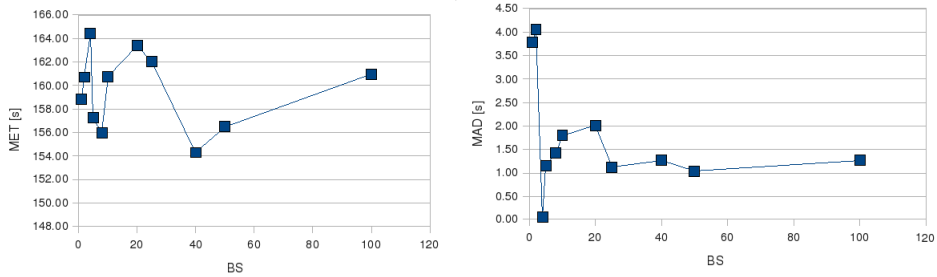
Elster and Holtet [7,8] have shown that increasing the border size (BS) and doing redundant computation is an effective technique to overcome the latency factor on SMP clusters when solving PDEs numerically. We ran the solver on some representative domains exchanging different borders with width in the range [1-100]. It is reported that, for supercomputers with Infiniband interconnection, some minimal performance improve-

ments were found using values of BS close to one. Since PCIe links are closer in latency to Infiniband interconnections than to Ethernet, we decided to have more test points in the neighborhood of $BS = 1$.

In Figure 4 we report results from the benchmark tests run using strip-partitioned, squared domains and texture-based kernels. Analyzing the graphs, we can deduce that the technique does not boost enough performance. The whole set of results can hardly approach a 10% of improvement (1.1X) with respect to the version based on unitary border width.



(a) Domain 1024x1024, strip partitioning, texture-based kernels, Median of 3 runs.



(b) Domain 52000x52000, strip partitioning, texture-based kernels, Median of 3 runs.

Figure 4. Border size influence on performance using four GPUs.

Thus, the empirical results bring us to a similar conclusion as in the supercomputers' case reported in [8], confirming our assumption that PCIe interconnections are fast enough to make the effect of the discussed method vanish.

5.2. Threads Synchronization

During the design phase, we decided to base our communication on POSIX threads synchronization, in order to assess its impact on communication and compare it with the alternative option of message-passing libraries. The latter option was seen to be responsible of around 70% of the communication overhead [2].

Figure 5 shows the impact of both synchronization and data transferring on communication per iteration.

Contrarily to what expected, the impact of synchronization is quite relevant, practically dominating the overall communication. Even though this impact is almost negligible on large scale (the core computation is kernel bound), such an effect must be ana-

lyzed and controlled as it may become more relevant in the perspective of more powerful hardware and shorter distances between host and devices.

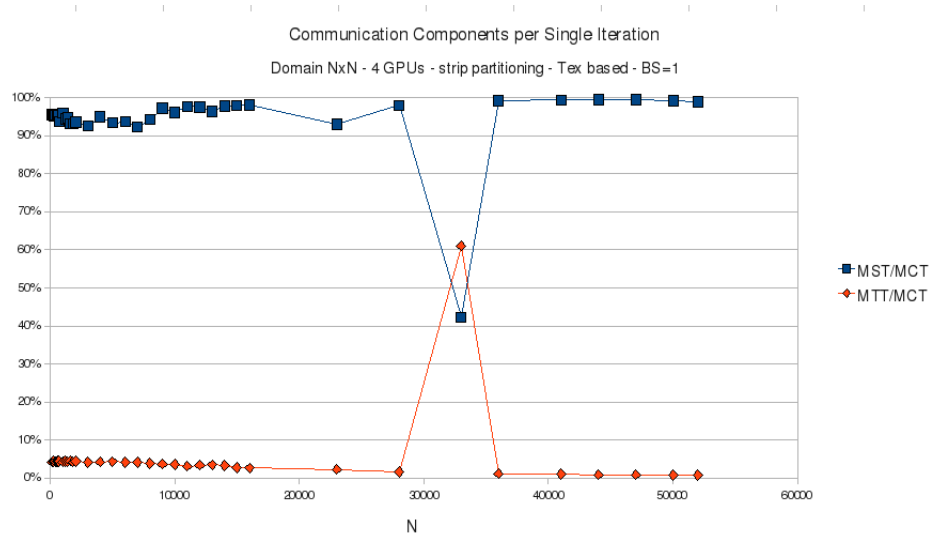


Figure 5. Sinchronization and transfer time over communication time during one iteration.

Profiling, it seems that the delay is due to lock contentions on neighbor domain areas. We plan a deeper investigation in our future work.

6. Conclusions and Future Work

The NVIDIA S1070 multi-GPU system was analyzed based on its specific hardware features and on possible analogies to fundamental parallel models, such as shared memory multiprocessors and multicomputers. The Poisson problem with Dirichlet boundary conditions was picked as our model problem since it does boarder exchanges of information common to a large class of application problems. Based on these analyzes, we defined a test space for the benchmark PDE solver tool we developed.

Varying the domains' dimensions up to 11 GB, we found the application I/O bound. Transferring the domain from host to device memory took always the largest percentage of time, requiring up to three times the kernel time. Excluding the first and the last necessary data transfers, the core computation was instead kernel bound. Exchanging only the essential data during the computation, i.e. the subdomains' borders, required no more than 10% of the total elapsed time.

Since our test system only consisted of four GPUs, vertical strips were more effective than blocked subdomains. However, simple communication models, normally used in parallel computing, were not found suitable for performance prediction on multi-GPU systems. Statistical approaches were found to be more stable and adaptable to slight technological alterations. We found a linear relation between the size of contiguous data expressed in gigabytes and the time required to transfer such data between the host and the device.

Using all four GPUs on the S1070, was always beneficial, performing up to 3.5X and 1.8X speedup on one and two GPUs, respectively. However, performance decreased working with large data volumes possibly due to resource contention.

Synchronizing GPUs through Pthreads condition variables took a relatively large percentage of the communication during the core computation. This is similar to what was previously documented for shared-memory based, message-passing libraries [2].

The framework we developed for our tests can be considered a premature stage of what could become a framework devoted to platform-independent, multi-GPU benchmarking. More attention must be paid to decouple the kernel logic from the synchronization logic, so to allow an easier and more independent design and analysis of both. As described in [4], the use of different precision standards, can also have a certain impact on performance. In a PDE solver context, introducing exit conditions based on proper approximations of the sought solutions can be a possible way to investigate eventual delays introduced by graphics hardware's precision. Our framework could be extended to the third dimension. For example, the requirement of exchanging not only borders but also surfaces introduces asymmetries in communication that would be important to examine.

The next goal should be to analysis GPU clusters composed by several multi-GPU nodes. In such systems, different GPUs may be interconnected through a multi-level communication network. Finally, with the introduction of the Green500 list[3], vendors are challenged to optimize the ratio performance/Watt instead of the only speed factor. This will no doubt lead to new interesting models.

References

[1] *NVIDIA CUDA 2.1 Programming Guide*. NVIDIA Corporation. http://www.nvidia.com/object/cuda_develop.html.

[2] P. Micikevicius. 3D Finite Difference Computation on GPUs Using CUDA, in *Proceedings of the 2nd Workshop on General Purpose Processing on Graphics Processing Units*, 383, pages 79-84, March 2009.

[3] T. Natvig and A. C. Elster. Using Context-Sensitive Transmission Statistics to Predict Communication Time, in *PARA 2008, LNCS 2010*, A. C. Elster *et al*. editors, Springer, to be published.

[4] D. G. Spampinato and A. C. Elster. Linear Optimization on Modern GPUs, in *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium*, Rome, Italy, May 2009, (CDROM) ISSN: 1530-2075, ISBN: 978-1-4244-3750-4.

[5] R. W. Hockney. The Communication Challange for MPP: Intel Paragon and Meiko CS-2, in *Parallel Computing*, volume 20, issue 3, pages 389-398, March 1994.

[6] D. G. Spampinato. *Modeling Communication on Multi-GPU Systems*. Master thesis, Norwegian University of Science and Technology, July 2009. http://www.idi.ntnu.no/ elster/master-studs/spampinato/spampinato-master-ntnu.pdf

[7] Anne C. Elster and R. Holtet. *Benchmarking Clusters vs. SMP Systems by Analyzing the Trade-off Between Extra Calculations vs. Computations* SC'02 Poster

[8] R. Holtet. *Communications-reducing Stencil-based Algorithms and Methods* NTNU MS thesis, July 2003. http://www.idi.ntnu.no/ elster/hpc-group/ms-theses/holtet-msthesis.pdf

[9] N. Eicker and T. Lippert. "'Low-level Benchmarking of a New Cluster Architecture"' in emphParallel Computing: Architectures, Algorithms and Applications, Vol. 38, PARCO 2007 Proceedings, Eds. C. Bischof et al., pp 381-388, IOS Press.

[3] http://www.green500.org