HP ProLiant SL390 G7 Server

Evaluation Report

Thomas Schoenemeyer, Jason Temple and Gilles Fourestry
Swiss National Supercomputing Centre (CSCS), Lugano, Switzerland
schoenemeyer@cscs.ch ; jtemple@cscs.ch ; fourestey@cscs.ch

Abstract – We evaluated the HP ProLiant SL390 fully populated with eight Nvidia M2090 GPUs. For the evaluation two benchmarks provided with the Nvidia SDK as well as benchmarks from the SHOC suite were selected as well as a hybrid version of DGEMM.

We also tested a Flash storage device attached to one of the available PCI slots.

The server is an interesting option for scientists whose applications fit onto the memory of eight GPU devices and can use multiple devices simultaneously.

1 Introduction

This report summarizes the performance measurements carried out with one HP ProLiant SL390s G7 server with 4U half width trays is designed for high GPU density and supports up to 8 GPUs. Fully populated, this system offers a rather unique GPU-to-CPU density on the market and therefore it is ideally suited for applications ranging from quantum chemistry, molecular dynamics, weather forecasting, seismic processing and data analytics. For a fully-populated system the total GDDR5 memory bandwidth sums up to around 1.2 TB/s with ECC on.

2 Server description

2.1 Hardware

The server shown in Figure 1 is a two socket Intel server with 48GB DDR3 Memory (6x8GB DIMMs), two 1 Gb Ethernet ports, one CX-2 based 10 Gb Ethernet port (SFP+) and an optional on board Infiniband CX-2 port (QSFP). It has one low profile x8 PCI-e gen2 connection, and eight x16 PCI-e gen2 slots [1].

Our server had two Intel Xeon X5660 6-core processors running with 2.80GHz. Its double precision theoretical peak performance is around 5.5 Teraflops.

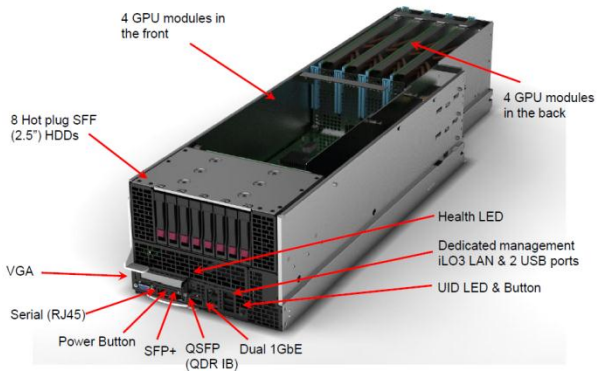

Figure 1: HP SL390 G7 server

The PLX switches PLX PEX8664 and PEX8647 as shown in Figure 2 offer exclusive x16 port flexibility to up to eight GPUs. Our server was fully populated with Nvidia M2090 GPUs (Figure 3), but also

other PCI based devices such as SSD PCIe can be installed such as the Virident FlashMax SLC storage device shown in Figure 4.

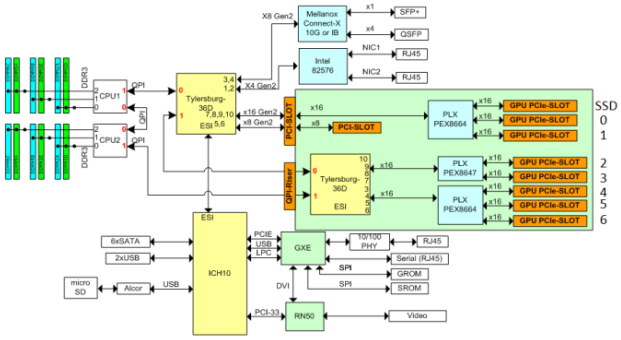

Figure 2: Board architecture incl. device numbers for GPU

Figure 3: HP SL390 G7 at CSCS

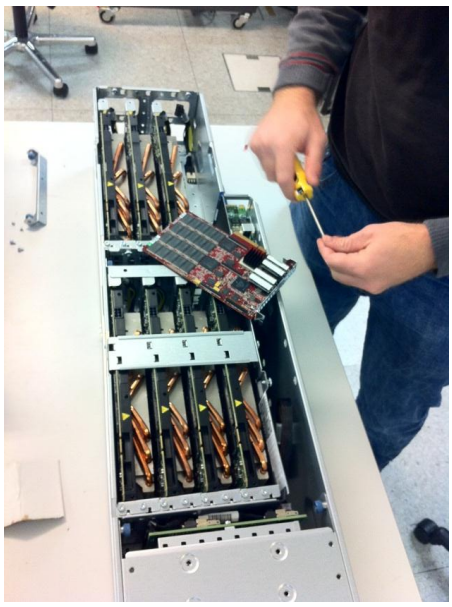

Figure 4: Prepare to install one Virident SSD SLC flash storage device at CSCS

2.2 Software

All our tests were carried out using the following software stack

- Linux kernel 2.6.32-71
- Intel Compiler Version 12.1
- Intel MPI
- nvidia gpu sdk 4.1.RC2
- mvapich2 1.8a1
- Nvidia Driver 285.05.33

We installed also the Portable Hardware Locality (hwloc) software package [2] which provides a portable abstraction of the hierarchical topology of modern architectures, including NUMA memory nodes, sockets, shared caches, cores and simultaneous multithreading. It also gathers various system attributes such as cache and memory information as well as the locality of I/O devices such as network interfaces, InfiniBand HCAs or GPUs. It primarily aims at helping applications with gathering information about modern computing hardware so as to exploit it accordingly and efficiently.

Figure 5 shows, that there is no exclusive access of sockets to GPUs which is consistent to Figure 2.

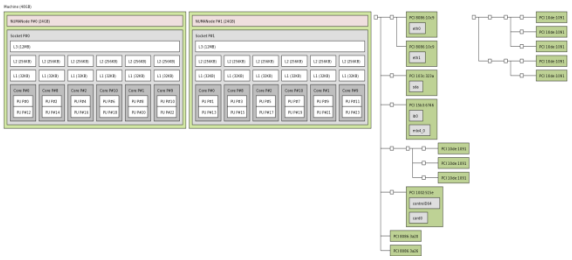

Figure 5: System topology compiled with lstopo (hwloc)

3 Attached devices

Since the server provides 8 x1 6 PCI Gen2 slots, various devices could be installed in this server. We tested the latest Nvidia GPUs as well as one Virident FlashMax Storage device.

3.1 Nvidia M2090

The M2090 is currently the fastest computing processor within the M-class GPU Computing Modules

- 665 GFlops Peak DP
- 6 GB memory size (ECC off)
- 177 GB/sec memory bandwidth (ECC off)
- 512 CUDA cores

As one of the key advantages, this HP-server allows to deploy standard M2090 modules sold by many system integrators. It does not require a special form-factor.

3.2 Virident SSD Flash Card

Before running the application test, we replaced one GPU module by SLC Flash Storage card manufactured by Virident. This device has a capacity of 800GB [5].

We run iozone using xfs and achieved excellent performance numbers of more than 1GB/s of read and write shown in Figure 6 and Figure 7.

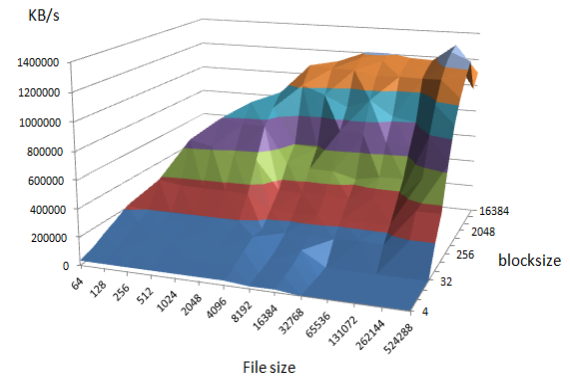

Figure 6: Iozone IO bandwidth results for Virident Flashmax (read)

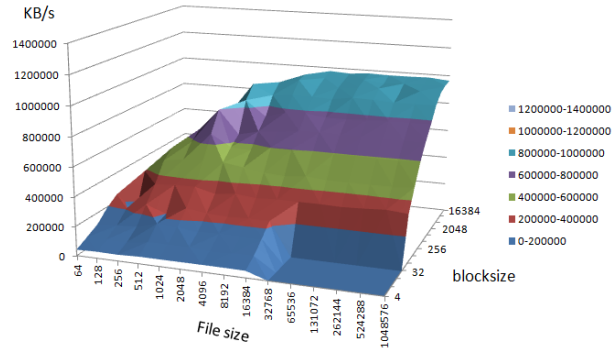

Figure 7: Iozone IO bandwidth results for Virident Flashmax (write).

Whenever an application runs on this server with challenging IO demands, the local PCI based SSD device could help as very fast local scratch storage, if the limited capacity of these devices is sufficient.

4 CUDA

The HP server offers an high GPU-CPU ratio, and there are some applications and cases that can be decomposed and run in parallel on multiple GPUs within a single host machine, achieving correspondingly higher levels of performance. One scenario could be a typical throughput scenario, where 8 copies of an application running each with one core and one dedicated GPU.

Another scenario is to distribute the workload to several GPUs running a non-MPI application. In the latter case, communication between individual GPUs might be a performance bottleneck.

To address this problem, CUDA 4.0 and CUDA4.1 introduced GPUDirect, and by this 3rd party network adapters, solid-state drives (SSDs) and other devices can directly read and write CUDA host memory, eliminating unnecessary system memory copies and CPU overhead, resulting in significant performance improvements in data transfer times [4].

GPUDirect also includes support for peer-to-peer (P2P) DMA transfers directly between GPUs and NUMA-style direct access to GPU memory from other GPUs. These capabilities lay the foundation for direct P2P communication between GPUs and other devices in a future release.

The functionality of the peer to peer access was tested with a sample code called ***simpleP2P*** , provided with the Nvidia SDK that demonstrates a combination of Peer-to-Peer (P2P) and the Unified Virtual Address Space (UVA) features new since SDK 4.0. Peer-to-Peer is currently only supported for GPUs connected to the same IO Hub, in our case devices 3, 4, 5, 6 and 7 (shown in Figure 2) allow peer-to-peer communication as well as devices 0, 1 and 2.

5 Benchmark results

The complex PCI switch design with dual IOHs connected to three PCI switches raises the question about the efficient data transfer between GPU devices and host CPUs. Therefore the first chapter 5.1 looks at the raw data bandwidth performance between host and device. In the other chapters we describe the result of various applications provided with the Nvidia SDK as well as standard benchmarks.

5.1 Bandwidth Measurements

We used the executable provided with the SDK and launched the program accordingly

./bandwidthTest --memory=pinned -device=0

In the memory mode pageable, the host-to-device (HTOD) transfer speed is around 3GB/s independent of the device number. The device-to-host bandwidth (DTOH) is slightly lower, for GPUs 0, 1 and 2 we measured 2.4 GB/s, for the GPUs 3,4,5,6 and 7 about 2.8 GB/s (Figure 8). For pinned memory, the bandwidth almost doubles as expected (Figure 9).

In order to get the best performance, the binding between GPUs and sockets must be optimized. GPUs 0, 1, 2 must be bind to socket 0. GPUs 3, 4, 5, 6 and 7 must be bind to socket 1, e.g. we issued the command as follows:

```
numactl --membind=0 --cpubind=0 ./bandwidthTest -device=0
```

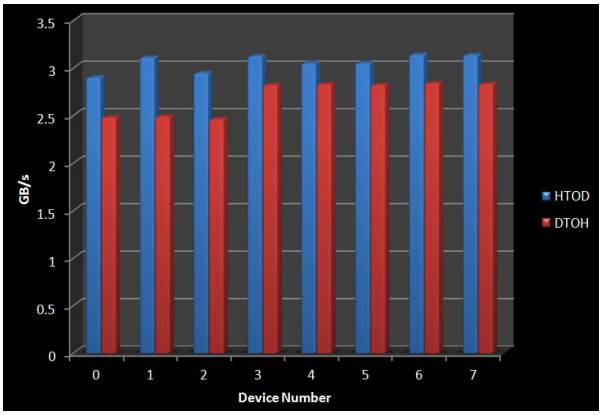

Figure 8: Host-to-device and device-to-host memory, pageable memory for individual devices

Figure 8 shows the result with optimal binding. The bandwidth is now exactly 3.1 GB/s and 2.8 GB/s for every device.

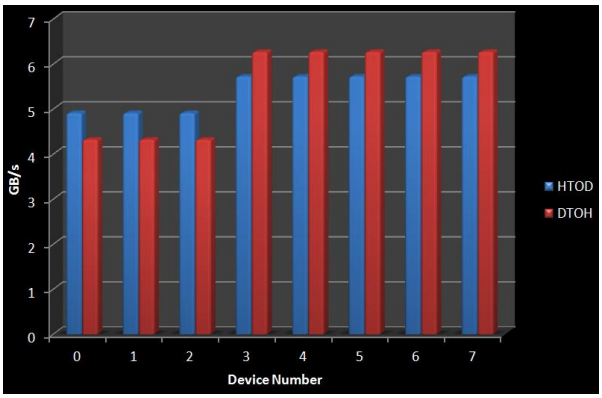

Figure 9: Host-to-device and device-to-host memory, pinned memory for individual devices.

5.2 N-body

The first one is N-body and it is described in detail in [7]. This kind of simulation technique approximates in many computational science problems such astrophysics, protein folding and turbulent fluid flows. The all-pairs approach to N-body simulation evaluates all pair-wise interactions among the N bodies requires a substantial time to compute and therefore an interesting target for acceleration. NVIDIA ported the all-pairs computational kernel using the CUDA programming model.

We launched the benchmark for three different problem sizes starting with 128K particles and up to 1024K particles with:

./nbody -benchmark -n=p -fp64 -numdevices=ng
with
p=128K, 512K, 1024K
ng= 1,2,3,4,5,6,7 and 8

For all cases we observed a good scalability to up to 7 GPUs, for the small case even to 8 GPUs (see Figure 10-12).

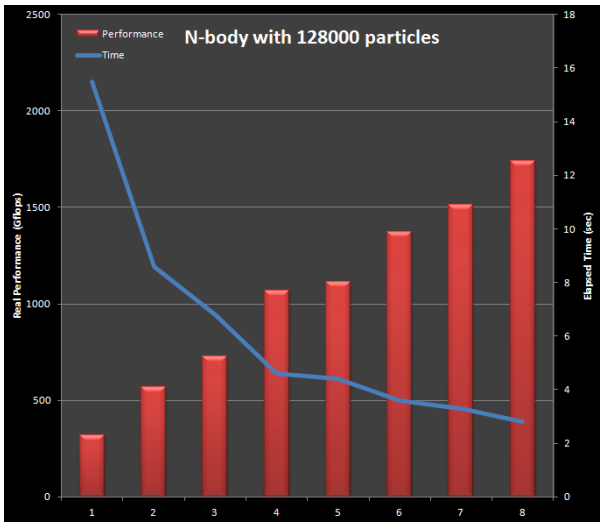

Figure 10: Nbody with 128000 particles

For the large case (see Figure 12), we measured nearly 2.2 Tflops sustained performance, which is equivalent to 40% of the theoretical peak performance (double precision) of this server.

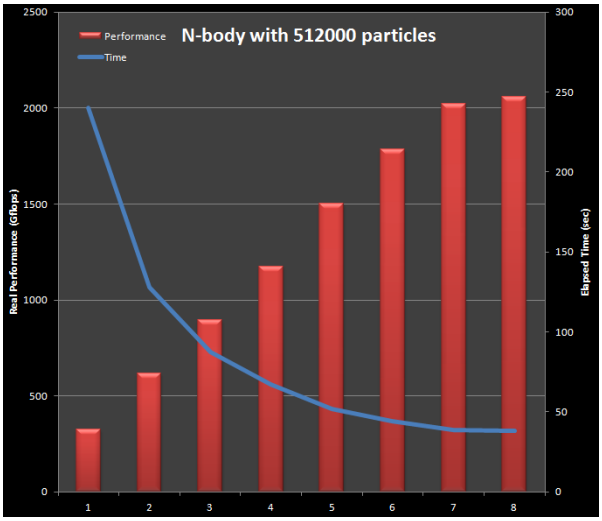

Figure 11: Nbody with 512000 particles

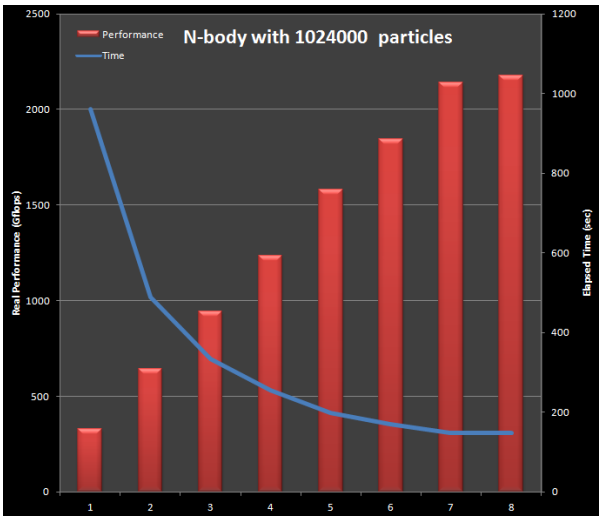

Figure 12: Nbody with 1024000 particles

5.3 MonteCarlo multi-GPU implementation

The background of this code is explained in detail in [9]. The code is the implementation of the Monte Carlo approach to option pricing written in CUDA.

Two methods are available, the threaded version uses one CPU thread for each device, the streamed version uses one CPU thread and handles all GPUs. (requires CUDA 4.0 or newer).

Because the Monte Carlo pricing of each option is independent of all others, the computation can be distributed across multiple CUDA-capable GPUs present in the system. The input options are divided into contiguous subsets (the number of subsets equals the number of CUDA-capable GPUs installed in the system).

The code uses all CUDA devices available on the server, the only way to control that is to set the environment variable accordingly with

```
export CUDA_VISIBLE_DEVICES="0,1,2,3,4,5,6,7"
```

for all GPUs on the system.

In the current SDK the number of simulation paths is fixed to 262144 due to the limited memory capacity of earlier GPU devices. In order to fully exploit the capabilities of the M2090 card we increased that number to 1048576 in MonteCarloMultiGPU.cpp.

```
./MonteCarloMultiGPU --method=streamed --type=double
```

The result is shown in Figure 13. The performance measured in options per second increases nearly linearly with the number of GPUs.

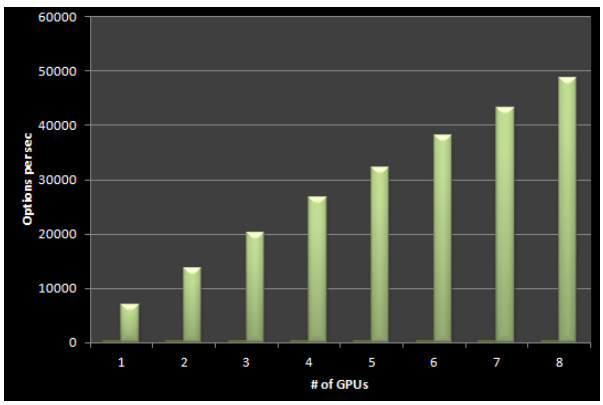

Figure 13: Options per second using 1048576 simulation paths

5.4 SHOC Benchmark for multi-GPU implementation

We used the GNU Compiler with Intel MPI

	OpenCL			*CUDA*		
Program	*S*	*EP*	*TP*	*S*	*EP*	*TP*
BusSpeedDownload	x	x		x	x	
BusSpeedReadback	x	x		x	x	
DeviceMemory	x	x		x	x	
KernelCompile	x	x				
MaxFlops	x	x		x	x	
QueueDelay	x	x				
BFS	x	x		x	x	
FFT	x	x		x	x	
MD	x	x		x	x	
Reduction	x	x	x	x	x	x
S3D	x	x		x	x	
SGEMM	x	x		x	x	
Scan	x	x	x	x	x	x
Sort	x	x		x	x	
Spmv	x	x		x	x	
Stencil2D	x		x	x		x
Triad	x	x		x	x	
BusCont		x			x	
MTBusCont		x			x	

Table 1: Available benchmarks in the SHOC suite

Due to the server design, we are most interested in the multiple GPU benchmarks. We measured the server performance and scalability with two so-called True Parallel (TP) benchmarks taken from the latest SHOC source [10]. We used Intel MPI version 4.0. For each GPU one MPI-rank is used. The server allows us to measure the performance with up to 8 devices per node, including all communication. These algorithms are decomposed among the parallel tasks and may include CPU computation time.

Reduction

This benchmark measures the performance of a sum reduction operation using the double precision floating point data [10]. The kernel performs first a partial reduction on a global input data array saving the partial results to the local shared memory. Finally, a reduction is computed over the local data array and the result is saved into a global output memory array. The result for up to 8 GPUs are shown in Figure 14 for the mean value of Allreduce-DP-Kernel.

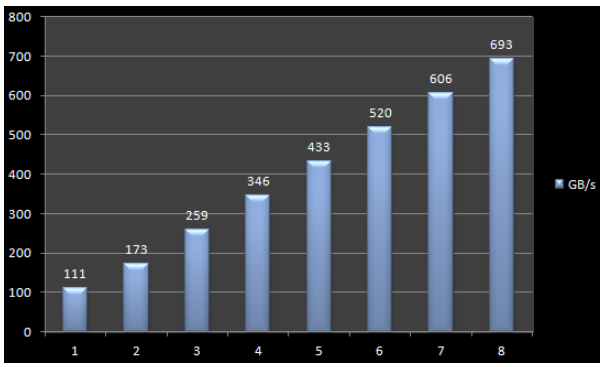

Figure 14: Bandwidth for reduction Allreduce-DP (mean)

Scan_DP

This benchmark measures the performance of the parallel prefix sum algorithm (also known as Scan) on a large array of floating point data. [10]. And based on the method decribed in [11]. The results are shown inFigure 15, we picked the mean value of TPScan-DP-Kernel.

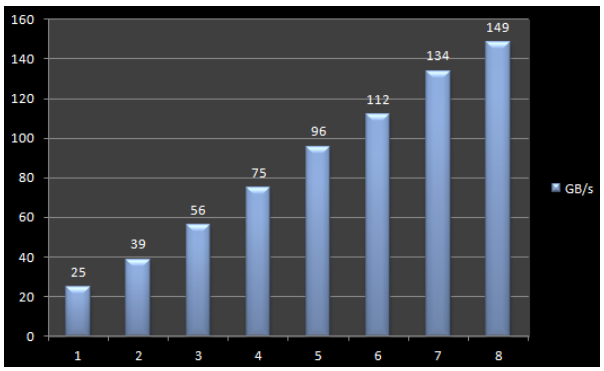

Figure 15: Bandwidth for Scan_DP

SHOC also includes benchmarks for testing multiple devices, but do not involve any communication between devices. These benchmarks are called Embarrassingly

Parallel (EP). One example is the MD benchmark.

Molecular Dynamics (EP)

This benchmark measures the speed of a simple pairwise calculation of the Lennard-Jones potential from molecular dynamics. Each thread computes the acceleration for one particle based on the potential field generated by all particles into a cutoff area. The kernel uses coalesced global memory accesses.

The benchmark is executed with mpirun and starts as many copies as MPI-ranks and uses one GPU per MPI-rank, the results are shown in Figure 16.

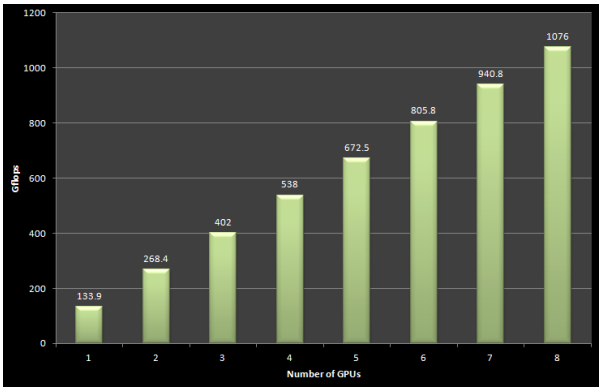

Figure 16: Throughput Performance for MD for up to 8 GPUs

This benchmark scales ideally with the number of GPUs and MPI ranks.

5.5 DGEMM

In this section, we investigate the behavior of DGEMM (i.e BLAS Double precision Generalized Matrix Matrix multiply, C = αA*B + βC), and in particular what strategy can reach maximum performance. Our first strategy is based on the hybrid dgemm developed by M. Fatica (12). In this algorithm, the matrix is split between GPUs and CPUs according to their respective performance. In order to get maximum performance, each time CUBLAS DGEMM starts to plateau, we substitute one call to the CPU dgemm with one call to cublasDgemm. Figure 18 shows the results of the hybrid multi GPU/CPU version. Maximum performance is around 1000+ Gflops and is reached using 4 GPUs with a 20K x 20K matrix. Using this algorithm, the limiting factor is the GPU memory size: one of the matrices involved in the Matrix-Matrix multiply is duplicated on all the GPUs, while the two others are split. At 20K, this matrix is taking half of the 6 GB while the 2 others, split 4 ways, take 1.5 GB each, filling the memory of each card.

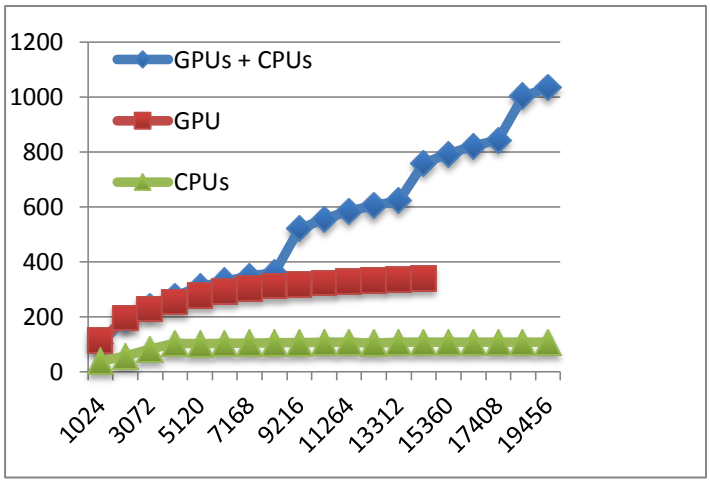

Figure 18: Hybrid multi GPU/CPU. X-axis is the matrix size (NxN), Y-axis is the performance in Gflops.

In order to circumvent this, let's split each matrix four ways:

$$\begin{pmatrix} C11 & C12 \\ C21 & C22 \end{pmatrix} = \alpha \begin{pmatrix} A11 & A12 \\ A21 & A22 \end{pmatrix} \begin{pmatrix} B11 & B12 \\ B21 & B22 \end{pmatrix} + \beta \begin{pmatrix} C11 & C12 \\ C21 & C22 \end{pmatrix}$$

Expanding the formula yields 8 matrix-matrix products and 4 matrix-matrix sums:

$$\begin{cases} C11 = \alpha(A11 * B11 + A12 * B21) + \beta C11 \\ C21 = \alpha(A21 * B11 + A22 * B21) + \beta C21 \\ C12 = \alpha(A11 * B12 + A12 * B22) + \beta C12 \\ C22 = \alpha(A21 * B12 + A22 * B22) + \beta C22 \end{cases}$$

The matrix-matrix products are sent to one GPU simultaneously, then the final sum

are computed on the CPU using a local array. Figure 19 shows the performance result of this algorithm.

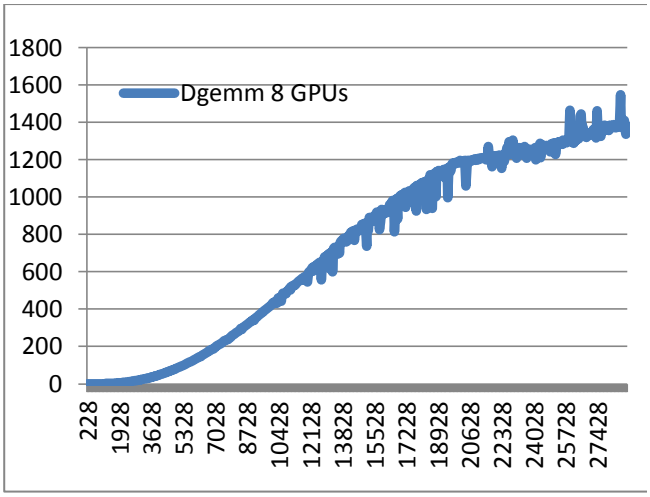

Figure 19: 8-GPU cublasDgemm. X-axis is the matrix size (NxN), Y-axis is the performance in Gflops.

Obviously, this algorithm is not suited for relatively small matrices. Comparing to the previous graph, for a matrix size below 13K, performance is below our hybrid DGEMM. Beyond that, DGEMM performance increases very fast and we are able to achieve up to 1.5 Tflops of double precision performance thanks to the fact that no matrices are replicated on the GPUs as it is done with the previous algorithm. Note that this algorithm currently does not use threads.

6 Conclusion

The HP Server offers a quite unique design in terms of GPU density and a similar approach is only provided by Tyan (13). The Tyan board is provided by various system integrators partially under their own brand.

The HP server is an interesting option for example for a non-MPI applications if the problem size fits on 40 GB of GDDR5 memory and is able to use multiple GPU devices simultaneously.

Applications such as nbody provided with the Nvidia SDK are running with impressive single-node performance to a level of 40% of the theoretical peak performance (double precision).

The system proved to be a very reliable and stable system. Since it is based on the previous Intel Xeon X5600 processor, CSCS looks forward to HPs next generation server hopefully with a similar GPU/CPU ratio, but based on the latest Intel Xeon E5 processors codenamed Sandy Bridge.

7 Literature

1. HP SL390, Technical Specifications, 2012.
2. http://www.open-mpi.org/projects/hwloc/
3. HP GPU Computing, Presentation at the HPC Advisory Council Workshop Lugano, March, 2011.
4. Nvidia GPUdirect™ Technology, Presentation, 2011.
5. Virident FlashMax PCIe Storage Class Memory, Datasheet, Virident, 2011
6. Nvidia Tesla, Datasheet, August 2011
7. Lars Nyland, Mark Harris, Jan Prins, Fast N-Body Simulation with CUDA, distributed with the SDK4.0 and SDK4.1, 2011
8. Victor Podlozhnyuk, Mark Harris, Monte Carlo Option Pricing, June 2008, distributed with SDK 3.0 and newer
9. Anthony Danalis, Scalable Heterogeneous Computing Benchmark Suite (SHOC),

Tech. Report, Oak Ridge National Laboratory, University of Tennessee, 2010.

10. Future Technologies Group, SHOC, The Scalable Heterogeneous computing benchmark suite, Manual Version 1.0.2 : March 2011

11. Shubhabrata Sengupta, Mark Harris, Yao Zhang and John D. Owens: Scan Primitives for GPU computing, Technical Paper, Graphics Hardware, 2007.

12. Massimiliano Fatica: Accelerating Linpack with CUDA on heterogenous clusters. Proceedings of 2nd workshop on General Purpose Processing on GPUs, ACM 2009.

13. Tyan GPU Platform, Specs, website, 2011