Quantifying NUMA and Contention Effects in Multi-GPU Systems

Kyle Spafford Jeremy S. Meredith Jeffrey S. Vetter

Future Technologies Group
Computer Science and Mathematics Division
Oak Ridge National Laboratory
1 Bethel Valley Road-MS6173
Oak Ridge, TN 37831
{*spaffordkl,jsmeredith,vetter*}*@ornl.gov*

ABSTRACT

As system architects strive for increased density and power efficiency, the traditional compute node is being augmented with an increasing number of graphics processing units (GPUs). The integration of multiple GPUs per node introduces complex performance phenomena including non-uniform memory access (NUMA) and contention for shared system resources. Utilizing the Keeneland system, this paper quantifies these effects and presents some guidance on programming strategies to maximize performance in multi-GPU environments.

Categories and Subject Descriptors

B.8.0 [**Hardware**]: Performance and Reliability General; C.1.3 [**Computer Systems Organization**]: Heterogeneous Systems

General Terms

Performance, Measurement

Keywords

Benchmarking, Performance, Graphics Processors, GPGPU

1. INTRODUCTION

For certain computations, architectures utilizing graphics processing units (GPUs) as accelerators offer increased performance, power efficiency, and spatial density. Because of these advantages, system designs featuring multiple GPUs per node are beginning to emerge.

In these systems, the GPUs are normally connected to the host processor via the PCIe bus. As more and more GPUs are added to the system, additional PCIe lanes are required to maintain the available bandwidth to each GPU. This bandwidth is critical–all data processed by a GPU must traverse this connection, and it is already relatively slow to compared to other system pathways such as Quick Path Interconnect (QPI) links (8 GB/s vs. 12.8 GB/s unidirectional peak bandwidth) or main memory.

GPGPU '11 March 5, Newport Beach, CA

Dual I/O Hub Designs.

Figures 1 and 2 show two block diagrams of multi-GPU systems. These systems illustrate different strategies for increasing the available number of PCIe connections. The first system, the Hewlett-Packard DL160, utilizes a single I/O hub (IOH) connected to a *PCIe switch*. All GPU traffic traverses a single x16 PCIe link to the switch and can then be routed to multiple NVIDIA Tesla T10 GPUs in the S1070 1U server. Using a PCIe switch is a straightforward method for adding additional GPUs, but the single link between the IOH and the switch becomes a bottleneck when all GPUs in the system are used simultaneously.

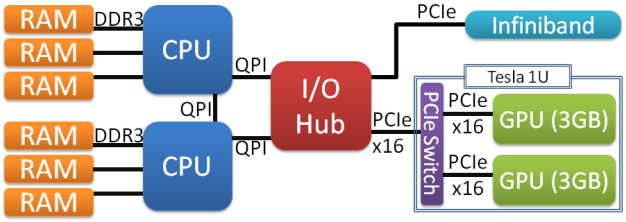

Figure 1: Block Diagram of HP DL160–An example of a PCIe switch-based design

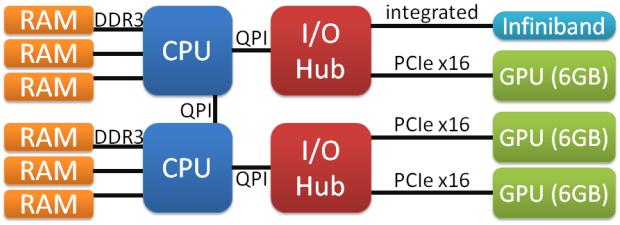

Figure 2: Block Diagram of HP SL390–An example of a dual-IOH system

An alternative approach, illustrated by the second system, is to add an *additional I/O hub*. This approach maximizes available bandwidth to the GPUs, but introduces non-uniform memory access (NUMA). Notice, in the figure, an additional QPI link must be traversed from CPU #0 to GPU #1 or #2 (and similarly for CPU #1 to GPU #0). This is the HP SL390, codenamed Ariston, and is preferable over the switch-based approach for workloads which utilize more than one GPU simultaneously. This study aims to explore the complexity added to the system by the introduction of the second IOH.

Contention.
In previous work, PCIe contention has been observed to have a significant impact on performance [4], up to a 30% degradation in bandwidth. This work furthers that analysis in a dual I/O hub environment and endeavors to determine best practices for sharing GPUs among multiple processes and threading design for simultaneous MPI and GPU use.

2. EXPERIMENTAL PLATFORM

Performance experiments presented in this paper were conducted on Keeneland, the National Science Foundation Track2D Experimental System based on the HP SL390 server accelerated with NVIDIA Tesla M2070 GPUs. Keeneland has 120 compute nodes, each with dual socket, hex-core Intel X5660 2.8 Ghz Westmere processors and 3 GPUs per node, with 24GB of DDR3 RAM. Nodes are interconnected with single rail, QDR Infiniband. Unless otherwise specified, results were measured using the following software stack: Intel C/C++ Compiler version 11.1, Open-MPI 1.4.3, and NVIDIA CUDA 3.2RC2. The results presented in the following sections refer to specific components in the system, as numbered in Figure 2.

3. MICROBENCHMARKS

3.1 PCIe Bandwidth

In order to understand NUMA at the most basic level, microbenchmarks were used to measure the latency and bandwidth of MPI and GPU data transfers. Bandwidth was measured using the transfer of blocks of data ranging in size from 1KB to 64MB. CPU and memory affinity were set using `numactl`, a standard utility for process and memory pinning. A direct comparison of PCIe bandwidth is presented in Figure 3 for sending data to the GPUs and in Figure 4 for retrieving data from the GPUs. These graphs show the bandwidth for the "short" and "long" paths in the system. Bandwidth results starting from CPU #1 are identical, but inverted.

PCIe bandwidth was observed to be assymetric for both correct and incorrect NUMA mappings, with slightly higher bandwidth observed when retrieving data from the GPU with correct NUMA pinning. The penalty of incorrect NUMA assignment is also assymetric, as well as substantial, with a measured 31% reduction in readback bandwidth and only a 13% reduction in download bandwidth for the largest size transferred. Furthermore, the assumption that the extra hop required in an incorrect NUMA mapping would mostly impact latency is inaccurate; as seen here, a significant
penalty on the bandwidth of large data transfers exists.

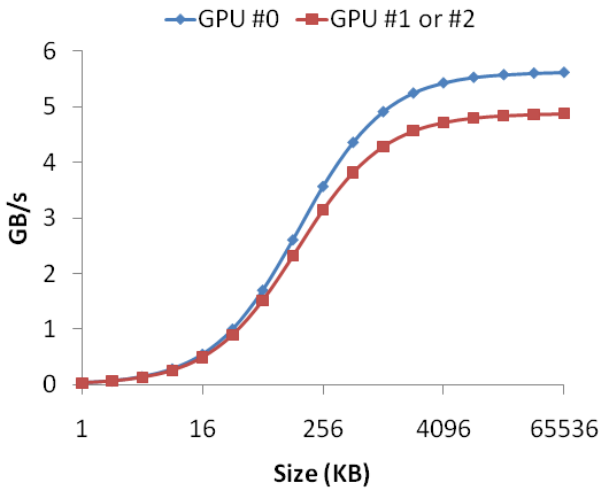

Figure 3: PCIe Bandwidth CPU #0-to-GPU

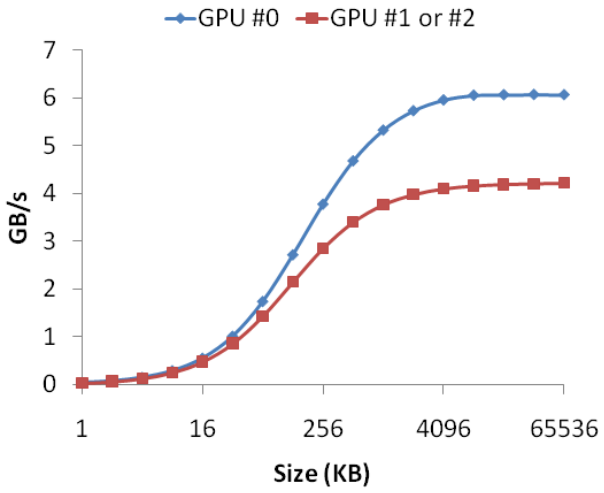

Figure 4: PCIe Bandwidth GPU-to-CPU #0

3.2 MPI Latency

Table 1 shows the average latency of MPI communication between two nodes of the Keeneland system for small and large messages. The correct mapping uses an MPI task pinned to the same NUMA node as the Inifiband HCA (NUMA node #0), and the the incorrect mapping has it mapped to other (NUMA node #1). A 12% penalty for small messages results from an incorrect mapping, and a 26% penalty for large messages is observed. Similar to the PCIe bandwith results above, this latter penalty also implies a bandwidth reduction, not simply a latency reduction.

4. BENCHMARKS

4.1 SHOC

The impact of NUMA assignment on basic algorithms and parallel computing primitives was measured using the Scal-

	NUMA Mapping	
	Correct	Incorrect
MPI Latency (0B)	1.877 μs	2.136 μs
MPI Latency (16MB)	13.955 μs	18.886 μs

Table 1: Average MPI Latency

able HeterOgeneous Computing Benchmark Suite (SHOC) version 1.0. The results in Table 2 use the large problem size on a single NVIDIA Tesla M2070 GPU and include data transfer time over the PCI express bus.

The raw performance here may seem lower than expected. This is because PCIe transfer time is typically not included in these measurements (it is often assumed to be amortized over many kernel executions, which is not always possible). One can also observe that the severity of the performance penalty depends on the computational density of the benchmark kernel. As density increases, the PCIe transfer time represents a smaller fraction of total runtime, and the NUMA penalty, as a percentage of overall execution time, decreases. For example, compare the relatively low penalty for matrix multiplication (SGEMM or DGEMM) to the higher penalty for the sum reduction or the nine-point stencil computation, which have a much lower flop-to-byte ratio.

4.2 HPL

Moving beyond single kernels, High Performance Linpack (HPL) provides excellent insight into NUMA effects because it is fairly simple and well-known, and has sustained MPI transfers and GPU operations (primarily the DGEMM and DTRSM kernels). It is much more representative of the behavior of full applications than single kernels. While no substantial performance difference due to NUMA was measured using a single MPI task, a clearly discernable trend emerged as the problem size was increased.

Figure 5 shows HPL performance under three different pinning schemes, with three MPI ranks per node. Work is dynamically split between the CPU and GPU, and each MPI rank uses four CPU cores for multithreaded Intel MKL calls. These results are from NVIDIA's version 9 of HPL, using a problem size scaled based on N=50,000 for a single node. A complete listing of parameters is provided in the appendix.

The pinning scheme is described using three numbers. These are the CPUs to which the three MPI ranks on each node are bound. Note that the three MPI ranks on each node choose GPUs in a sequential manner. Unlike the previous performance comparisons, where some NUMA assignments are clearly incorrect, it becomes more difficult to predict the optimal mapping.

At first glance, it is surprising that the 0-1-1 assignment performs the worst, since it should result in the best bandwidth to the GPUs, based on microbenchmark results. In this case, the bandwidth advantage is mitigated by other factors, including reduced MPI performance (two MPI ranks have the "long path" to the HCA) and lower CPU core utilization. Core utilization is lower since eight of the MKL threads inherit the pinning to CPU #1, leaving two cores idle on CPU #0. The best performance is obtained by pinning the first two MPI ranks and allowing the third to be placed as needed.

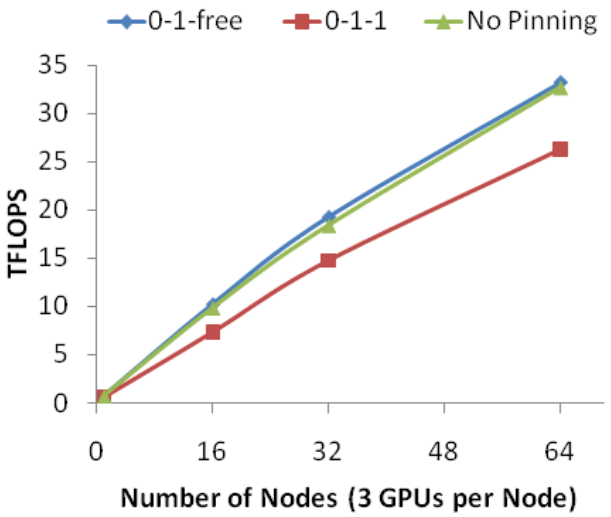

Figure 5: HPL Performance ranging from one to sixty-four nodes. Series names indicate which CPU the MPI ranks were pinned to, respecitvely (free indicates no pinning for the third rank).

5. CONTENTION

While the PCIe link between the IOH and PCIe switch was the obvious bottleneck in the HP DL160, the limiting factor of the dual-IOH design is less apparent. Analysis of theoretical bandwidths shows Ariston to be a fairly balanced design. The following experiments explore performance effects arising due to contention of shared system resources, and identify whether or not actual execution is as balanced as the theoretical bandwidths would predict. Results are obtained using the contention benchmark from the SHOC suite. In this benchmark, the user creates two types of tasks–MPI communicators and tasks that repeatedly transfer data to a GPU. These tasks are first executed sequentially, in which no contention should occur, then simulataneously. The contention penalty is defined as the difference between the simultaneous and the sequential runs.

With this capability, we attempt to anwer the following three questions:

- Does the dual-IOH design deliver on its purported bandwidth advantages?
- How many MPI tasks can share a GPU and still achieve acceptable bandwidth?
- Should MPI tasks always spawn a separate thread to control a GPU, so that the new thread can be appropriately pinned for NUMA effects?

5.1 Bus Saturation

The main advantage of the dual-IOH design over a switch-based approach is increased bandwidth when the PCIe bus is fully saturated. For an elementary understanding of contention effects, and for verification of the claimed advantage, actual bandwidth is measured for two scenarios, which correspond to usage patterns observed in accelerated applications.

Test	Units	Correct NUMA	Incorrect NUMA	% Penalty
SGEMM	GFLOPS	535.640	519.581	3%
DGEMM	GFLOPS	239.962	230.809	4%
FFT	GFLOPS	30.501	26.843	12%
FFT-DP	GFLOPS	15.181	13.352	12%
MD	GB/s	12.519	11.450	9%
MD-DP	GB/s	19.063	17.654	7%
Reduction	GB/s	5.631	4.942	12%
Scan	GB/s	0.007	0.005	31%
Sort	GB/s	1.081	0.983	9%
Stencil	seconds	8.749	11.895	36%

Table 2: SHOC Benchmark Results

This first scenario involves one GPU control task and one MPI task for each GPU in the system–for a total of six tasks on a Keeneland node. This scenario is a close approximation of the behavior of HPL and real applications, where one MPI rank is used per GPU, and issues both MPI and GPU traffic. Scenario two involves spawning three GPU control tasks and then filling the remaining nine cores with MPI tasks. This approximates a worst-case behavior for MPI applications that divide work between GPU ranks and CPU ranks–when all twelve ranks generate PCIe traffic at once.

Figure 6 shows the performance penalty under contention for MPI latency on small and large messages, and for PCIe bandwidth to the GPUs. In the first scenario, we see penalties ranging from approximately 6% to 9%. These penalties grow when more MPI commmunication tasks are added to fully saturate the cores in the node, increasing to a range of 7% to 14%. The MPI latency on small messages incurs the least contention penalty in both scenarios, whereas the other two, more bandwidth-dependent tests, incur larger penalties.

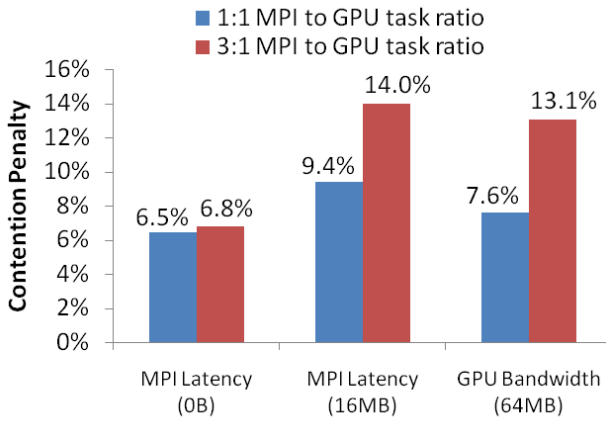

Figure 6: Contention penalty – Using one MPI communication task per GPU control task, and using one task on all cores. In both cases there are three GPU control tasks.

5.2 Sharing A Fermi Among MPI Ranks

NVIDIA's Fermi introduced the ability to simultaneously execute two kernels. An important part of obtaining high performance from this throughput-oriented architecture is to ensure the GPU remains supplied with an adequate amount of work. As such, one proposed use case is to share the GPU among multiple MPI ranks in order to obtain high utilization. This is a reasonable approach, but it can easily be bottlenecked by contention on the PCIe bus.

Figure 7 shows the bandwidth of large transfers from host to GPU memory for 64MB messages when all three GPUs are being accessed simulataneously from one to four threads per GPU. The chart shows the minimum, mean, and maximum speeds achieved by the most bottlenecked task. When more than one task accesses each GPU, we see immediate drops in bandwidth, but these are commensurate with linear expectations for two and three tasks per GPU (i.e. one-half and one-third of the bandwidth, respectively). However, when we reach four tasks per GPU, bottlenecks and imbalances become severe, with some tasks seeing as low as 0.07 GB/sec, almost two orders of magnitude below the maximum.

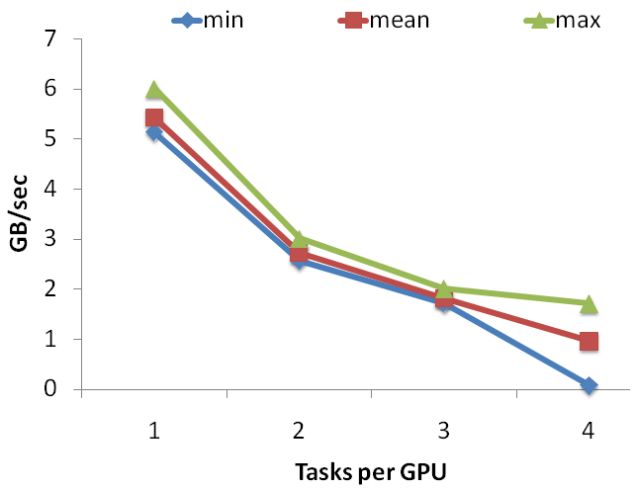

Figure 7: Sharing GPUs – This chart shows the decrease in bandwidth as GPUs are shared among multiple processes. Shown here are the minimum, mean, and maximum of the transfer rate to the most-bottlenecked GPU. All three GPUs on the node were accessed concurrently in all cases.

5.3 Splitting MPI/GPU Control into Different Threads

In most accelerated applications that use MPI, the same rank controls a GPU and issues MPI traffic. This presents a problem for architectures like the SL390. Consider the MPI rank assigned to GPU #1. When pinned to CPU #0, it has the correct mapping for the HCA and the "long path" to GPU #1, and vice versa when pinned to CPU #1.

Given the performance penalties observed from incorrect NUMA mapping, we hypothesize that higher performance may be achieved by spawning a separate GPU control thread. This way, both threads can be pinned to avoid any "long paths" in the system. For a Keeneland node, this involves pinning MPI threads and GPU threads differently.

Figure 8 shows the MPI latency and GPU transfer rate under contention, in a scenario in which MPI communication and GPU control tasks are pinned together (to the NUMA node closest to that GPU), and in the case where the MPI and GPU tasks are separated and pinned to the appropriate NUMA nodes. Results are shown for one, two, and three sets of GPU and MPI tasks per node. One can observe that splitting these actions results in approximately a 5μs improvement in latency for large MPI messages, regardless of the number of tasks per node. For PCIe bandwidth, a single GPU control task had enough dedicated bandwidth to be unaffected by the presence of the MPI communication task. However, at two and three GPU control tasks, moving the MPI communication to the correct NUMA node freed enough bandwidth to allow a modest increase in performance.

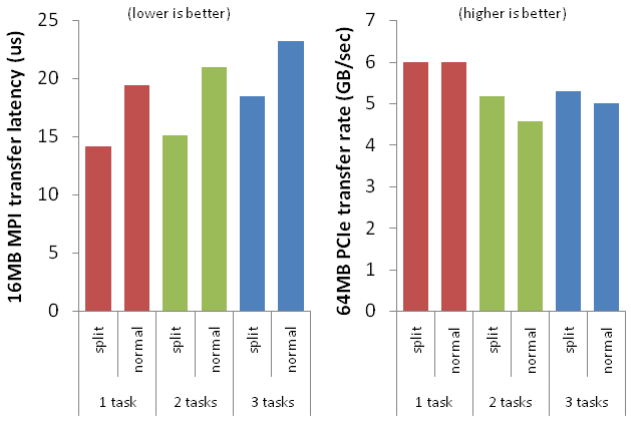

Figure 8: Performance under contention, with and without splitting – Left: MPI Latency, Right: PCIe Bandwidth.

5.4 Summary

The dual-IOH design exhibited robustness under contention, limiting penalties under saturation and allowing satisfactory concurrent bandwidth to all GPUs in the system. Performance limits were reached when a disproportionate dropoff in bandwidth was observed at four tasks per GPU, and the contention penalties for both bandwidth and latency increased notably with a large number of simultaneous MPI communicators on the node. A strong decrease in contention penalties was observed when MPI communication was moved to a different thread and relocated to the appropriate NUMA node for the system interconnect.

6. APPLICATIONS

A number of high performance computing applications are beginning to utilize GPUs, and here were present results under different NUMA mappings. These applications are playing critical roles in leadership science in the U.S. Department of Energy and National Science Foundation.

6.1 Application Overviews

Several scientific computing applications were selected to be investigated in detail, testing their performance with both correct and incorrect NUMA mappings in varying scenarios:

- **LAMMPS** is the Large-scale Atomic/Molecular Massively Parallel Simulator and can simulate biomolecular, solid-state, and coarse-grained systems [11]. A set of modules called "gpulammps" is available for a set of pair potential calculations [2]. Results are shown using the `lj-cut-dyn` test.
- **DCA++** is a Quantum Monte Carlo simulation code, used for materials science studies such as high-temperature superconductivity [8]. Its Hirsch-Fye solver has undergone GPU acceleration, largely utilizing dense matrix computation [10]. For this solver we tested its performance on a 24-site cluster with 80 time slices.
- **GROMACS** is a free molecular dynamics package designed primarily for biomolecular systems [6]. It uses OpenMM for GPU acceleration of a subset of its algorithms [5], but as of this writing does not support parallel runs. To test its performance we used the `dhfr-impl-2nm` benchmark.

6.2 Application Results

The series of tests measured single tasks utilizing GPU #0. The task was mapped to NUMA node #0 (correct) or node #1 (incorrect). Average performance is compared in each mode to determine the penalty of an incorrect mapping.

For DCA++ and LAMMPS, which support multiple tasks per node, a two-task, two-GPU test was also performed in two modes. In the first mode the tasks utilizing GPUs #0 and #1 were mapped to their closest NUMA nodes (#0 and #1, respectively), and in the second case they were mismapped to farther NUMA nodes, (#1 and #0, respectively).

Each of the applications uses GPUs differently and has different sensitivity to poor PCIe performance, and as seen in Figure 9, this results in varying performance penalties for a NUMA mismatch. LAMMPS appears to have the lowest penalty from an incorrect mapping, and GROMACS the highest, ranging from 0.5% to almost 4%. Both two-task runs show a greater penalty from mis-mapping than the single-task runs, which was likely caused by the added contention on QPI links between the NUMA nodes.

7. CONCLUSION

Microbenchmarks confirm that significant NUMA effects are present in the studied dual-IOH architecture. Quantified at the most basic level, the penalty for bandwidth to the

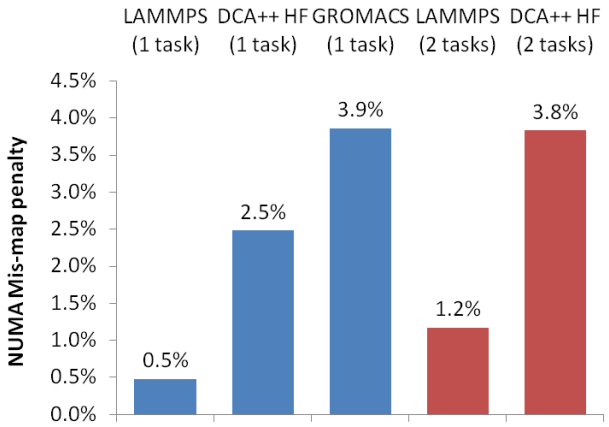

Figure 9: Application performance penalties–Performance drop when using an incorrect NUMA mapping

GPU is roughly 22% on average and for MPI latency can be as large as five microseconds. The severity of this penalty's manifestation in more complex benchmarks or real applications varies based on several factors including computational density, number of kernel executions per PCIe transfer, and the total fraction of the application which actually utilizes the GPU.

An average performance penalty of 13.5% was observed in the SHOC benchmark suite for single-run kernels. HPL results were more dramatic, showing a 21% gap between the best and worst pinning scheme at sixty-four nodes. More interestingly, the optimal NUMA mapping was different than what microbenchmarks suggest as the best approach.

When testing three accelerated applications, the penalty became relatively small compared to the total runtime, diminishing the perceived importance of the NUMA mapping. However, these results may not be conclusive–there is strong dependence and variance on how much of the application actually uses the GPU. Furthermore, if HPL (where most functions run on the GPU) is indicative of where production applications are headed, the relative importance of NUMA will increase at scale.

We also arrive at three major conclusions about the contention for shared system resources. First, the dual-IOH design delivers on its claim of better bandwidth–a total simultaneous bandwidth of 16.2 GB/s was observed to the GPUs, which is more than twice the theoretical bandwidth of the switch-based approach. Second, sharing Fermis among a small number of MPI tasks or threads is a feasible approach to increase GPU utilization. Sublinear bandwidth scaling only occured when a GPU was shared by more than three tasks. Third, splitting MPI communication and GPU traffic into different threads can be an effective method to cope with contention penalties in simple cases–saving about five microseconds of MPI latency and preserving the maximum amount of bandwidth to GPUs. However, our results may not be definitive for more complex applications–thread creation overhead, thread lifespan, and decreased productivity due to programming complexity must all be evaluated when deciding on a threading strategy.

8. RELATED WORK

Most direcly related to our work are the differences in GPU bandwidth due to NUMA effects which have been briefly documented by Kidnratenko et al. [7], including an explicit call for further study. Our results augment these early findings and add significant detail, hopefully providing a better understanding of the details of NUMA and the role it plays in the accelerated application ecosystem.

Also related to our contribution are the ongoing GPU benchmarking efforts including SHOC [4], Parboil [1], and Rodinia [3]. These software suites provide valuable test kernels and the basic codes necessary for performance measurement in heterogeneous environments.

NUMA analysis and approaches for the correction of NUMA performance problems have existed for quite some time in non-accelerated systems. Current projects such as Memphis [9] seek to address NUMA at a lower level in the system hierarchy–across CPU sockets.

9. ACKNOWLEDGMENTS

This research is sponsored in part by the Office of Advanced Computing Research; U.S. Department of Energy and the National Science Foundation award OCI-0910735. The work was performed at Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725.

10. REFERENCES

[1] *Parboil Benchmark Suite*. `http://impact.crhc.illinois.edu/parboil.php`.

[2] *GPULAMMPS*, 2010. `http://code.google.com/p/gpulammps/`.

[3] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC '09)*, pages 44–54, Austin, TX, USA, 2009. IEEE.

[4] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. The Scalable Heterogeneous Computing (SHOC) Benchmark Suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU '10, pages 63–74, New York, NY, USA, 2010. ACM.

[5] M. S. Friedrichs, P. Eastman, V. Vaidyanathan, M. Houston, S. Legrand, A. L. Beberg, D. L. Ensign, C. M. Bruns, and V. S. Pande. Accelerating Molecular Dynamic Simulation on Graphics Processing Units. *Journal of Computational Chemistry*, 30(6):864–872, April 2009.

[6] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *Journal of Chemical Theory and Computation*, 4(3):435–447, March 2008.

[7] V. Kindratenko, J. Enos, G. Shi, M. Showerman, G. Arnold, J. Stone, J. Phillips, and W. mei Hwu. GPU Clusters for High-Performance Computing. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pages 1 –8, 31 2009-sept. 4 2009.

[8] T. A. Maier, M. S. Jarrell, and D. J. Scalapino. Structure of the pairing interaction in the two-dimensional hubbard model. *Physical Review Letters*, 96(4):47005, 2006.

[9] C. McCurdy and J. Vetter. Memphis: Finding and Fixing NUMA-Related Performance Problems on Multi-Core Platforms. In *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, pages 87 –96, 2010.

[10] J. S. Meredith, G. Alvarez, T. A. Maier, T. C. Schulthess, and J. S. Vetter. Accuracy and Performance of Graphics Processors: A Quantum Monte Carlo Application Case Study. *Parallel Comput.*, 35(3):151–163, 2009. 1513319.

[11] S. Plimpton. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics*, 117:1–19(0), March 1995.

11. APPENDIX

SHOC benchmarks executed with the large problem size (-s 4).

HPL results are the average of five runs. The following parameters were used in HPL measurements.

```
N - 50,000 192,000 271,000 384,000
NB - 768
PxQ - 1x3, 6x8, 8x12, 16x12
PFACT - 0
NBMIN - 2
NDIVs - 2
RFACT - 0
BCAST - 0
DEPTH - 1
SWAP - 1
S THRESHOLD - 192
L1 - 1
U - 1
EQUILIBRIATION - 1
MEM ALIGNMENT - 8
```