Performance Evaluation of CPU-GPU communication Depending on the Characteristic of Co-Located Workloads

Dongyou Seo
School of Computer Science and Engineering
Seoul National University
Seoul, Korea
dyseo@dcslab.snu.ac.kr

Shin-gyu Kim
School of Computer Science and Engineering
Seoul National University
Seoul, Korea
sgkim@dcslab.snu.ac.kr

Hyeonsang Eom
School of Computer Science and Engineering
Seoul National University
Seoul, Korea
hseom@dcslab.snu.ac.kr

Heon Y. Yeom
School of Computer Science and Engineering
Seoul National University
Seoul, Korea
yeom@dcslab.snu.ac.kr

***Abstract*—Todays, there are many studies in complicated computation and big data processing by using the high performance computability of GPU. Tesla K20X recently announced by NVIDIA provides 3.95 TFLOPS in precision floating point performance [1]. The performance of K20X is 10 times higher than Intel's high-end CPUs. Due to the high performance computability of GPU, K20X was adapted to Titan, the first super computer in the world [2][3]. However, additional steps are needed in GPU computing process, which aren't needed in the computation using only CPU. The data required to execute on GPU has to move from main memory to global memory of GPU before GPU computation. The results created on GPU also have to write back to main memory. The data movement is called as CPU-GPU communication. The communication between CPU and GPU is a big part of the computation using GPU. So, many studies tried to optimize CPU-GPU communication [4][5]. In this paper, we evaluated the performance of CPU-GPU communication depending on co-located workloads and presented which workload severely degraded the performance of CPU-GPU communication.**

Keywords-component; Performance Evaluation; General Purpose GPU; Workload Consolidation; CPU-GPU communication; CUDA

I. INTRODUCTION

At the initiatory stage, GPU is simply an accelator for graphic processing. But, after CUDA SDK is announced by NVIDIA in 2007, GPU is no longer an accelator, but a first-class computing unit in all fields which use parallel algorithm. CUDA is used for image and video processing, fluid mechanics simulation, CT image restoration, earthquake analysis and ray tracing in a way that is faster and more convenient [6]. Also, the hardware performance of GPU has made great advances as well as the software platform for GPU. Figure 1 represents the comparison of GFLOPS between Intel's CPU and NVIDIA's GPU. After July 2008, NVIDIA's GPU got ahead of Intel's CPU in both single and double precision performance. Since both development environment and hardware performance have made great advance, GPU receives attention from the fields where high performance computing is needed. To use the high performance computability, CPU-GPU communication is inevitable, not in the computation using only CPU. Depending on the characteristic of workloads, the time of CPU-GPU communication may take longer than GPU computation, especially for the workloads which have

cyclic pattern in CPU-GPU communication. In this paper, we evaluate the performance of CPU-GPU communication when consolidated with various workloads in multi-core platform and demonstrate which workload can degrade the performance a little more.

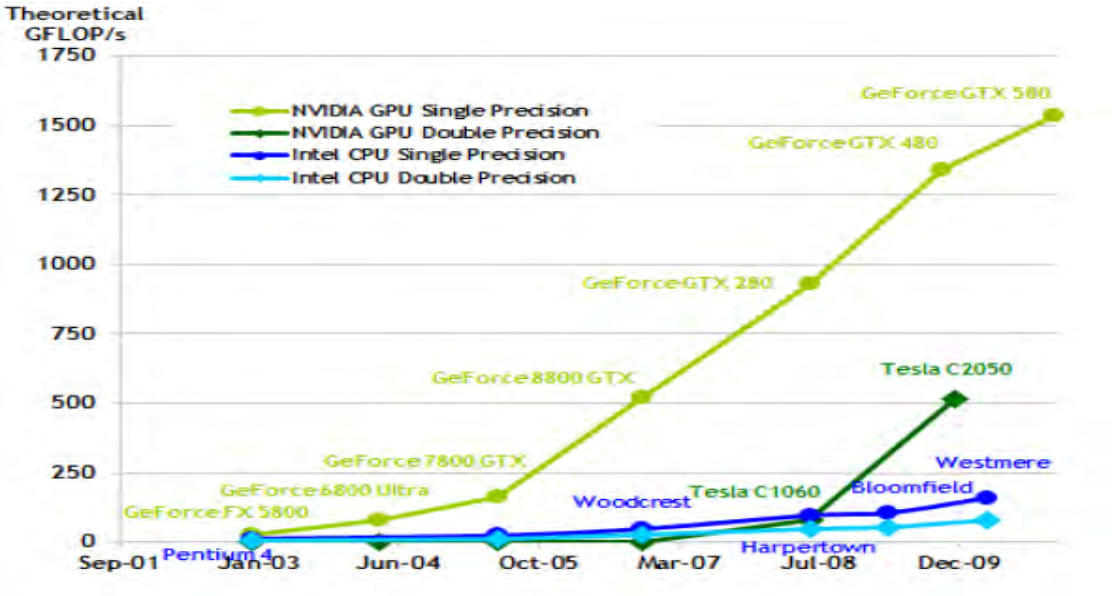

Figure 1. GFLOPS Comparision between Nvidia's GPUs and Intel's CPUs

II. CPU-GPU COMMUNICATION

A. *Connection Structure between CPU and GPU*

CPU-GPU communication happens through PCI express. The device which plays a role as a messenger is Direct Memory Access (DMA) controller, not CPU. When CPU has to deal with the situation where CPU-GPU communication is needed, CPU requests DMA controller and then DMA controller moves the data from host / device to device / host. The data movement is completed and DMA controller interrupts CPU to notify the completion of data movement. This process is composed of CPU-GPU communication. Figure 2 shows the connection diagram between CPU and DMA controller. Both address and data buses are shared by CPU and DMA controller [7]. Although DMA controller is moving the data (during CPU-GPU communication), other cores of same CPU can access to host memory at the same time due to Symmetric Multi-Processing (SMP). In short, both CPU and DMA controller can concurrently send requests to host memory so that CPU and DMA controller affect each other.

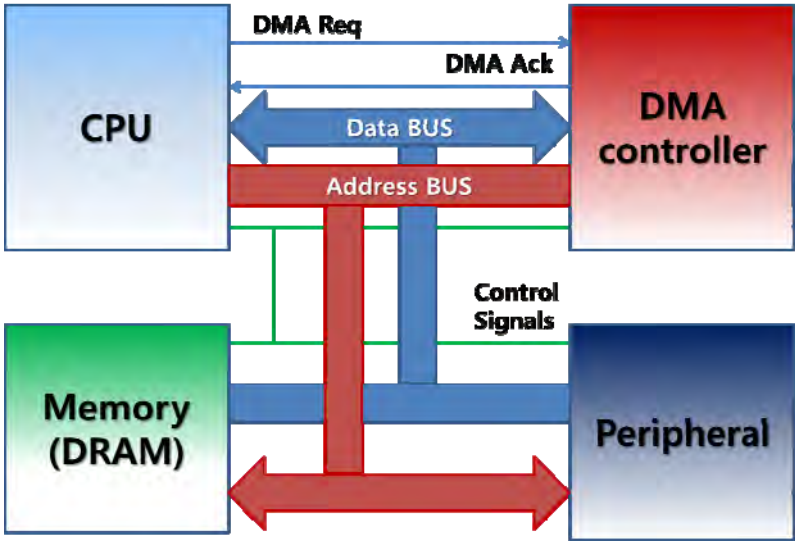

Figure 2. Connection diagram between CPU and DMA controller

B. Cyclic Communication Pattern

In general, the GPU computation using CUDA is composed of 3 steps. First, the data which CPU prepares for GPU computation is moved to GPU's global memory. Second, the transferred data is executed on GPU (For example, __kernel__ function). After the end of GPU computation, the results created by GPU write back to host memory. But, the 3 steps may not just happen at once. Some workloads iterate the steps according to the characteristic.

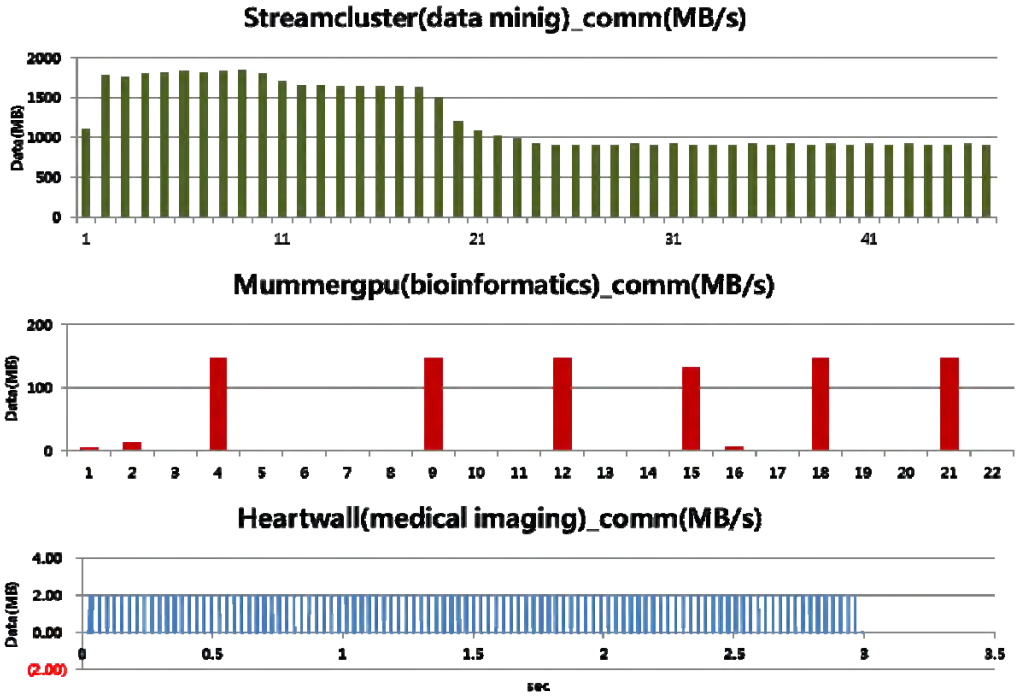

Figure 3. The workloads representing cyclic communication pattern

Figure 3 represents that streamcluster, mummergpu and heartwall, rodinia 2.2 benchmark suit, have cyclic pattern in CPU-GPU communication. X axis is timeline and Y axis is the transferred data (Mega Byte) for 1second. In case of streamcluster, the amount of the transferred data is bigger than 1GB for 1second. If high-memory traffic workloads are consolidated, CPU-GPU communication of streamcluster will severely be degraded. In next section, we'll show the numerical value.

III. EXPERIMENTS

In this section, we evaluated the performance of CPU-GPU communication and GPU computation time when same 3 SPEC CPU 2006 benchmarks (only CPU) and 1 rodinia 2.2 benchmark (both CPU & GPU) are consolidated in same CPU socket.

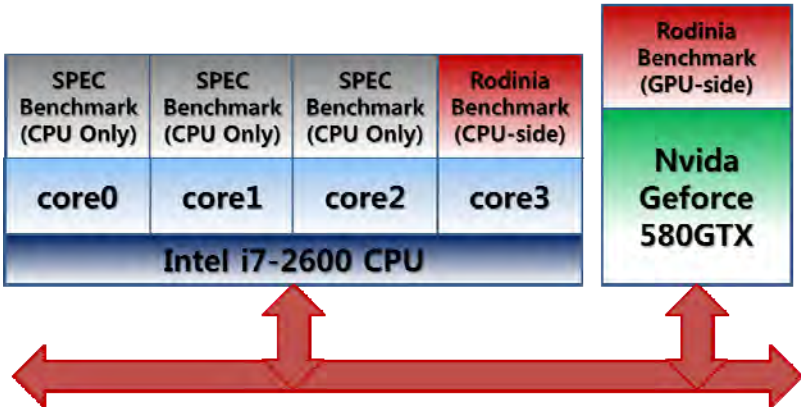

Figure 4. The diagram of GPU workload consolidation

Figure 4 is the diagram of our experiment. A Rodinia benchmark occupies a dedicated core of CPU and GPU during the runtime. Each SPEC workload also has a dedicated core. We measured the runtime of both CPU-GPU communication and GPU computation time by changing SPEC workloads. Table 1 shows our experimental setup.

TABLE I. EXPERIMENTAL SETUP

Components	Descriptions
CPU	**Inter i7-2600 CPU @ 3.40GHz 4 cores, 8GB main memory**
GPU	**NVIDIA Geforce 580GTX, 1.5GB global memory**
Operating System	**Ubuntu 12.04 LTS**
Benchmarks	**SPEC CPU 2006 benchmark suite [8], rodinia 2.2 benchmark suite [9]**

A. The characterization of SPEC CPU 2006 workloads

We selected SPEC CPU 2006 benchmark as an interrupter, the most widely known CPU benchmark. Since we pay attention to degrade the performance of CPU-GPU communication due to the memory bus contention, we evenly selected Memory-intensive workloads and CPU-intensive workloads as the interrupter. CPU-intensive workloads are enough to cache their data in Last Level Cache (LLC) so that the workloads don't frequently access to main memory. But, Memory-intensive workloads often access to main memory during the runtime. Figure 5 shows the measurement of average read and write bandwidth. We measured the bandwidth by configuring Intel's offcore events into model specific register (MSR) [10]. povray, sjeng, gobmk and hmmer are CPU-intensive workloads. soplex, mcf, lbm, milc and libquantum are Memory-intensive workloads.

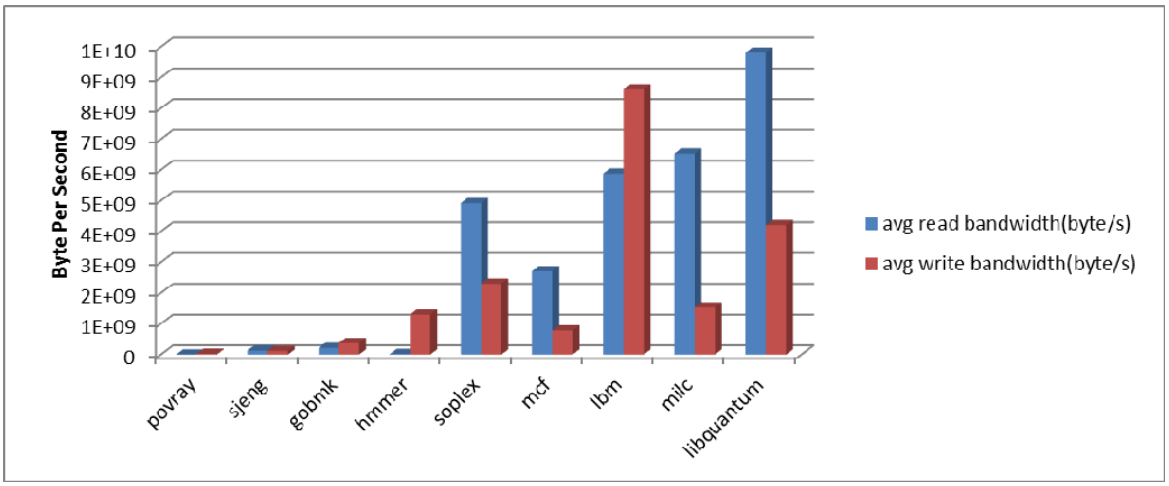

Figure 5. The memory bandwidth measurment of SPEC CPU 2006 workloads

B. The performance variation of GPU workloads

As you see Figure 4, we measured both CPU-GPU communication and GPU computation time by consolidating a rodinia benchmark with same 3 SPEC workloads and using nvprof [11], included in CUDA SDK 5.0. There is the difference at the ratio of CPU-GPU communication time among rodina benchmarks. In case of streamcluster and mummergpu, the communication time takes up a considerable part of total GPU computing. As the results of Figure 3 and Figure 6 are put together, we think that the communication time of the workloads which have short communication period and big transferred data takes the great part of GPU computing. Specially, the communication time of sphyraena occupies most of GPU computing since sphyraena queries SELECT statement for very large databases by exploiting GPU [12]. Also, Figure 6 demonstrates that GPU computation time isn't influenced by co-located workloads. In contrast to GPU computation time, CPU-GPU communication time is susceptible to the characteristic of co-located workloads. Since memory-intensive workloads (lbm, milc, libquantum) take the great part of memory bus traffic, all rodinia workloads are degraded in CPU-GPU communication time. The workload whose communication time takes the great part of total GPU computing is vulnerable to the memory bus contention. Although lbm don't have higher read bandwidth than libquantum, CPU-GPU communication time was more degraded in all cases when GPU workload was consolidated with lbm, not libquantum. Considering the greatest interference of lbm, we can think that write bandwidth exerts more influence over CPU-GPU communication than read bandwidth. In short, the GPU workloads which have high communication time is better when memory-intensive workloads aren't executed at the same time.

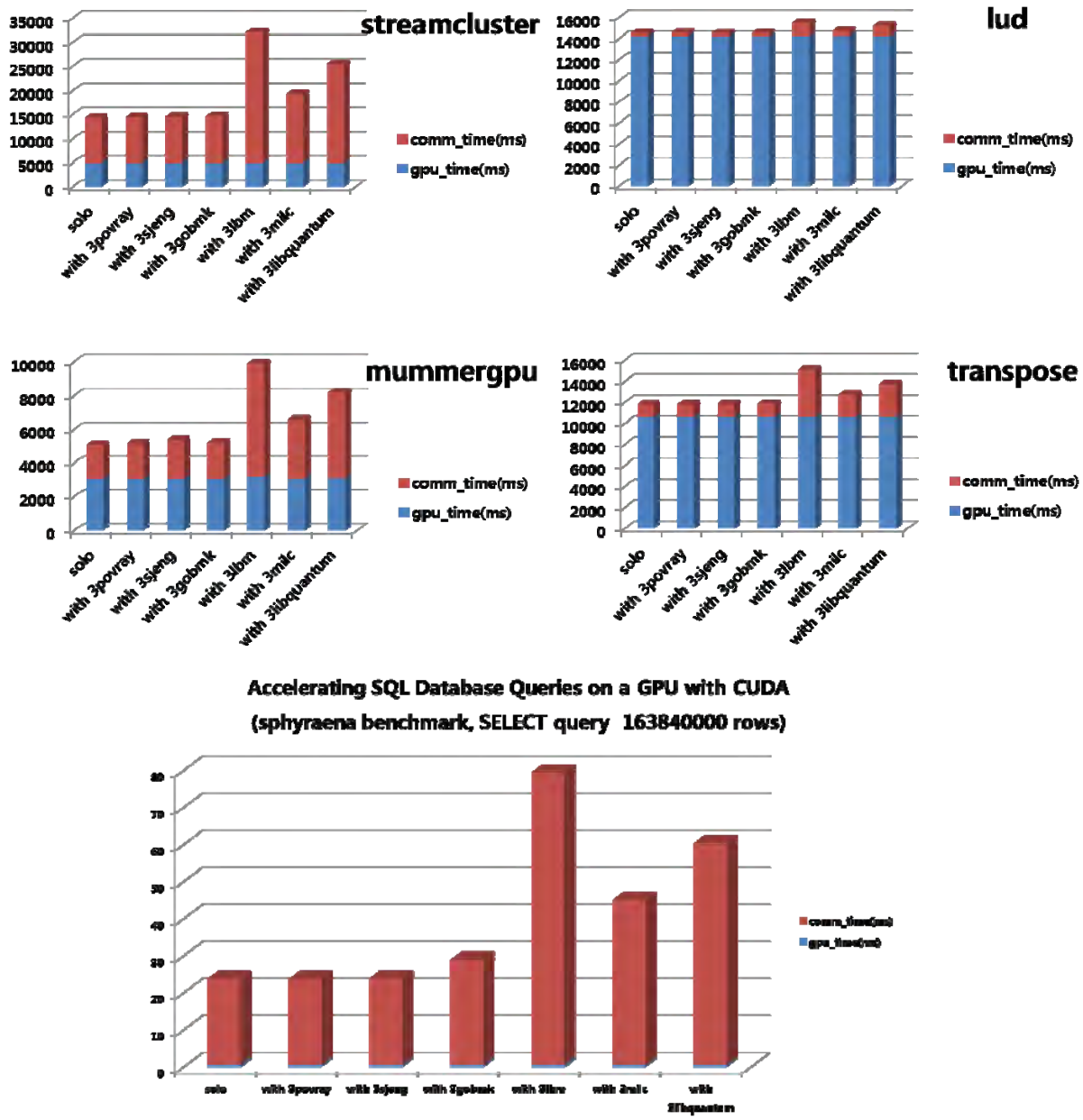

Figure 6. CPU-GPU communication time and GPU compution time of rodinia 2.2 benchmark suite depending on co-located SPEC workloads

IV. FUTURE WORKS

In previous section, we represented that CPU-GPU communication time is susceptible to the characteristic of co-located workloads. Memory-intensive workloads like lbm, milc and libquantum considerably degrade the performance of the communication time. In cloud platform which has to guarantee performance isolation, the huge degradation of GPU computation will be a serious problem because GPU instance is more expensive than normal instance. Also, we found that high write bandwidth workloads more severely affect the communication time than read bandwidth. If we adapt the profiling method using Intel's offcore events into linux kernel scheduler, the scheduler can distinguish high write bandwidth process without huge overhead because reading and writing model specific register (MSR) is approximate to assembly language [10]. As a result, when CPU-GPU communication happens, we can take the necessary bandwidth since kernel scheduler knows which processes interfere in the communication and can force the processes not to occupy memory bus during the communication time. In future, we'll implement the system which can adjust memory bus traffic for CPU-GPU communication.

V. CONCLUSION

In this paper, we show that CPU-GPU communication is very severely degraded when GPU workload is consolidated with high bandwidth workloads in the same socket, although GPU computation time is not affected. High write bandwidth workloads exert more influence rather than high read bandwidth workloads. In conclusion, we insist that system needs to know the importance of the interference and should try to retain the needed bandwidth for CPU-GPU communication.

ACKNOWLEDGMENT

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology. (No. 2010-0020731) The ICT at Seoul National University provided research facilities for this study.

REFERENCES

[1] Tesla Kepler GPU Accelerators, NVIDIA Technical Report
[2] http://www.nvidia.co.kr/object/tesla-k20x-worlds-most-efficient-gpus-powers-titan-20121112-kr.html#2
[3] http://www.top500.org/lists/2012/06
[4] Thomas B. Jablin, Prakash Prabhu, James A. Jablin, Nick P. Johnson, Stephen R. Beard, David I. August, "Automatic CPU-GPU Communication Management and Optimization", PLDI'11, June 4–8, 2011, San Jose, California, USA
[5] Haicheng Wu, Gregory Diamos, Srihari Cadambi, Sudhakar Yalamanchili, "Kernel Weaver: Automatically Fusing Database Primitives for Efficient GPU Computation", MICRO'12
[6] http://www.nvidia.co.kr/object/cuda-kr.html
[7] http://www.embedded.com/electronics-blogs/beginner-s-corner/4024879/Introduction-to-direct-memory-access
[8] John L. Henning, "SPEC CPU2006 Benchmark Descriptions"
[9] https://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Main_Page
[10] Intel 64 and IA-32 aArchitectures Software Developer's Manual: Volume 3B: System Programming Guide, Part2
[11] http://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvprof-overview
[12] *Peter Bakkum and Kevin Skadron, "Accelerating SQL Database Operations on a GPU with CUDA: Extended Results", In GPGPU '10: Proceedings of the Third Workshop on Genral-Purpose Computation on Graphics Processing Units, pages 94-103, New York, NY, USA, 2010,ACM*