PROCEEDINGS OF SCIENCE

Performance of Kepler GTX Titan GPUs and Xeon Phi System

Hwancheol Jeong*, **Weonjong Lee, and Jeonghwan Pak**
Lattice Gauge Theory Research Center, CTP, and FPRD,
Department of Physics and Astronomy,
Seoul National University, Seoul, 151-747, South Korea
E-mail: wlee@snu.ac.kr

Kwang-jong Choi, Sang-Hyun Park, and Jun-sik Yoo
Department of Physics and Astronomy,
Seoul National University, Seoul, 151-747, South Korea

Joo Hwan Kim, Joungjin Lee, and Young Woo Lee
Seoul Science High School, Seoul, 110-530, South Korea

NVIDIA's new architecture, Kepler improves GPU's performance significantly with the new streaming multiprocessor SMX. Along with the performance, NVIDIA has also introduced many new technologies such as direct parallelism, hyper-Q and GPU Direct with RDMA. Apart from other usual GPUs, NVIDIA also released another Kepler 'GeForce' GPU named GTX Titan. GeForce GTX Titan is not only good for gaming but also good for high performance computing with CUDA. Nevertheless, it is remarkably cheaper than Kepler Tesla GPUs. We investigate the performance of GTX Titan and find out how to optimize a CUDA code appropriately for it. Meanwhile, Intel has launched its new many integrated core (MIC) system, Xeon Phi. A Xeon Phi coprocessor could provide similar performance with NVIDIA Kepler GPUs theoretically but, in reality, it turns out that its performance is significantly inferior to GTX Titan.

31st International Symposium on Lattice Field Theory - LATTICE 2013
July 29 - August 3, 2013
Mainz, Germany

*Speaker.

PoS(LATTICE 2013)423

1. Introduction

On 2013, NVIDIA launched a new Kepler GPU, GTX Titan, named after the fastest supercomputer, a GPU cluster of NVIDIA Tesla K20X at Oak Ridge National Laboratory [1]. GeForce GPUs are designed for gaming. However, GTX Titan is good for parallel computing with CUDA, too. From the standpoint of computing, GTX Titan is as great as Tesla K20X. Nevertheless, the price of the former is about 3 times cheaper than of the latter.

Meanwhile, in 2012, Intel announced the Xeon Phi system with Intel many integrated core architecture (MIC) [2]. A Xeon Phi coprocessor integrates many CPU cores on a PCI express card like GPU, so that it could, in principle, provide similar theoretical performance with GTX Titan. The merit is that most of usual C codes which runs on CPUs can run on Xeon Phi system without much modification, because it is a CPU-based platform. However, it turns out that its performance is so low that it is very hard to obtain the high performance from Xeon Phi.

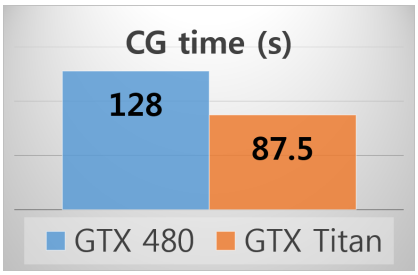
2. GTX Titan & Kepler Architecture

Architecture	Fermi	Kepler (GK104)	Kepler (GK110)		
GPU Device	GTX 580	GTX 680	Tesla K20X	GTX TITAN	GTX 780
# of CUDA Cores	512	1536	2688	2688	2304
Core Clock (MHz)	772	1006	732	837	863
SP GFLOPS	1581	3090	3950	4500	3977
DP GFLOPS	197	128	1312	1300	166
Memory Size (GB)	1.5	2	6	6.1	3
Memory Bandwidth (GB/sec)	192.4	192.26	250	288.4	288.4
L1 cache + shared memory (KB)	64	64	64	64	64
read-only data cache (KB) (#)	0	0	48	48	48
L2 cache (KB)	768	512	1536	1536	1536

Table 1: chip and memory specifications of recent NVIDIA GPUs

Table 1 presents chip and memory specification of NVIDIA Kepler GPUs compared with Fermi GTX 580. GTX Titan inherits most of important features of the Kepler architecture such as new streaming multiprocessor SMX, increased memory bandwidth and dynamic parallelism. In addition, it supports the features provided only for Tesla or Quadro GPUs such as large memory size and high performance in double precision floating point calculation.

Figure 1 shows the performance of conjugate gradient (CG) solver by Fermi GTX 480 and Kepler GTX Titan without applying any optimization to Kepler GPUs. There are also other studies presenting the performances of Kepler GPUs on Lattice QCD codes [3] [4]. Although GTX Titan's



Device	CG time (ratio)
GTX 480	128 s (1)
GTX Titan	87.5 s (0.68)

Figure 1: CG performance by GTX 480 and GTX Titan. Here, CG time means the time (in the unit of second) which it takes to run the CG code 10 times in two GPUs.

performance are, in principle, much better than that of GTX 480, our CG code is optimized only for Fermi GPUs and not for Kepler GPUs. Hence, it is necessary to tune the code such that it achieves the highest performance for GTX Titan.

	Fermi	Kepler
simultaneous blocks / SM(X)	8	16
warp schedulers / SM(X)	2	4
registers / thread	63	255
bandwidth (GB/s)	192 (GTX 580)	288 (GTX Titan)

Table 2: changed properties related with thread and block scheduling

There are several optimization schemes possible for GTX Titan. Table 2 shows those changes in GPUs regarding thread and block scheduling. A SMX (Kepler) has 6 times more cores than SM (Fermi). To deal with these cores, the SMX has twice number of blocks run simultaneously and twice of warp schedulers than SM. The number of registers per thread is also increased to 255, so that a thread can store more variables to registers and reuse them quickly. Therefore, we might obtain better performance by simply adjusting thread and block numbers.

The change of the memory bandwidth is also very important. Unfortunately in general, the main bottle neck in GPUs is the limitation in data transfer speed between GPU registers and memories. The performance of a CUDA program is usually determined by the product of CGMA (compute to global memory access) ratio and the amount of data transfer per time [5]. Here, CGMA ratio means the number of floating point operations per single data transfer.

Kepler architecture has new features to improve the memory usage as follows [6].

- 8 bytes shared memory bank mode is added. Fermi GPUs provide only 4 bytes (32 bits) mode. But Kepler GPUs provide 8 bytes (64 bits) mode, too. When this mode is turned on, one gets about twice the effective bandwidth for double precision floating point numbers.
- Fermi GPUs only support 16 Kbytes (shared memory) + 48 Kbytes (L1 cache) and 48 + 16 modes. Kepler GPUs can allocate 32 K to shared mem and 32 K to L1.
- 48 KB Read-only data cache is added. The texture memory can be used as an additional read-only cache memory for Kepler GPUs.

- Warp shuffle is introduced. By warp shuffle, data between threads in a warp can be exchanged without using shared memory. Thus we can reduce redundant use of the shared memory. Moreover, its latency is lower than shared memory access.

There is one more important new technology: direct parallelism. If we use it, new threads can be spawned directly from GPU kernel, so that one can reduce communications with CPU [7]. We are implementing the above new technologies to our GPU code.

3. GPU Direct

Recently NVIDIA introduced an advanced version of GPU Direct, called as GPU Direct RDMA (remote direct memory access). GPU Direct is a technology by which one can improve communication between GPUs, between a GPU and other network devices, and between a GPU and storage devices. There are three kinds of GPU Direct [8]. GPU Direct version 1 is designed for communication between GPU to other network or storage devices. GPU Direct version 2 provides peer-to-peer communication between GPUs on the same PCIe bus. They were already available in Fermi architecture. The new one, GPU Direct RDMA extends this to infiniband network communication between GPUs using RDMA. Unfortunately, GPU Direct RDMA is only available for Tesla and Quadro GPUs of Kepler architecture. GTX Titan supports only GPU Direct v2.

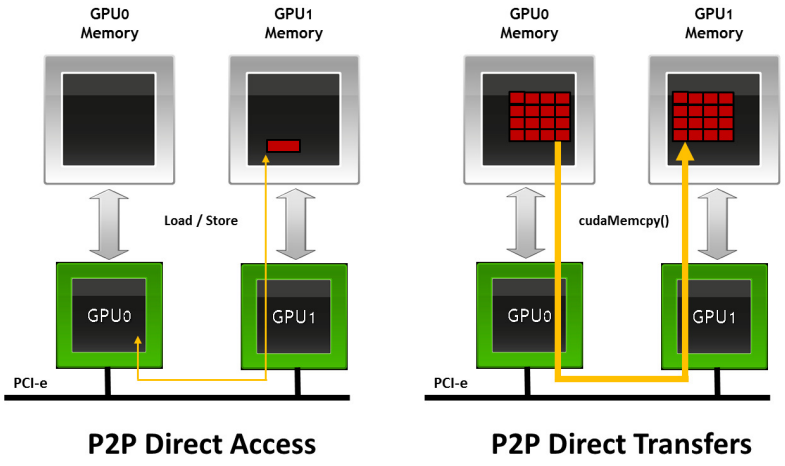


Figure 2: GPU Direct on the same PCIe bus

Fig. 2 is the schematic diagram of GPU Direct v2, by which a GPU can access the memory of another GPU on the same PCIe bus or transfer data to the another without using the CPU memory.

A usual MPI + CUDA program assigns one GPU per one MPI process node. For simplicity, consider a cluster node with 2 GPUs (GPU0 and GPU1) and we run a MPI job of 2 processes. The first MPI process (MPI0) is assigned with CPU0 and GPU0, and the second one with CPU1 and GPU1. Then we want to transfer some data stored in GPU0's memory to GPU1's memory. Without GPU Direct, we should follow a cumbersome procedure.

1. First copy the data from GPU0's memory to CPU0's memory by using CUDA.
2. Send the data in CPU0's memory to CPU1's memory through the memory of the infiniband network adapter by using MPI.

3. Copy the data to GPU1's memory by using CUDA.

Figure 3 shows the code doing this data transfer. Whereas, this 3-step data transfer can be reduced

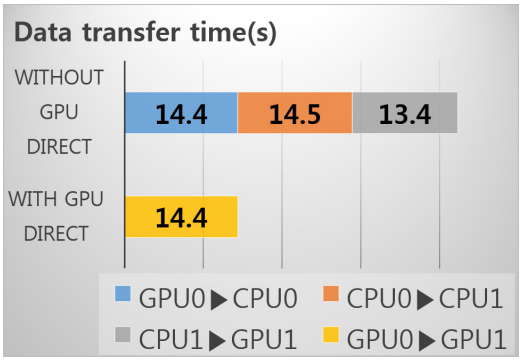
```
// GPU0 to CPU0
if( rank == 0 ) cudaMemcpyGtoC_host( a1, d_a, size );
// CPU0 to CPU1
if( rank == 0 ) MPI_Send( a1, N, MPI_FLOAT, 1, 0, MPI_COMM_WORLD );
else MPI_Recv( a1, N, MPI_FLOAT, 0, 0, MPI_COMM_WORLD, &status );
// CPU1 to GPU1
if( rank == 1 ) cudaMemcpyCtoG_host( d_a, a1, size );
```

Figure 3: data transfer without GPU Direct : GPU0 → CPU0 → CPU1 → GPU1

to a single step by GPU Direct (Figure 4). With GPU Direct, the data in GPU0 are transferred to GPU1 at once by a CUDA function: `cudaMemcpyPeer()`.

```
// GPU0 to GPU1
cudaMemcpyPeer( d_a1, 1, d_a0, 0, size );
```

Figure 4: data transfer with GPU Direct : GPU0 → GPU1



GPU Direct	data transfer time (ratio)
off	42.3 s (1)
on	14.4 s (0.34)

Figure 5: data transfer time for 10^7 single precision numbers (40MB) from GPU0 to GPU1. Both GPUs are GTX Titan.

Figure 5 presents the transfer times for 40MB data between 2 GTX Titan on the same PCIe bus with and without GPU Direct v.2. GPU Direct v.2 reduces the transfer time down to 1/3.

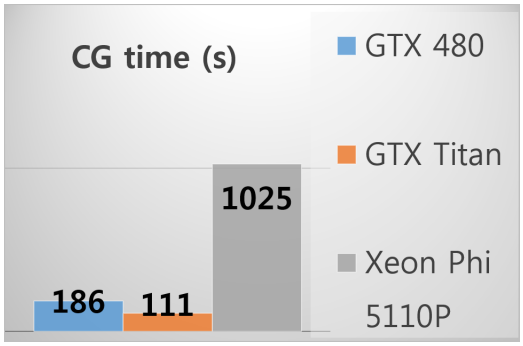
4. Xeon Phi

Table 3 presents the specification of Xeon Phi coprocessors compared with NVIDIA Kepler GPUs. A Xeon Phi coprocessor gives, in principle, similar performance with GTX Titan theoretically. However, the theoretical performance of the Xeon Phi coprocessor includes a factor 16 comes from vectorization. The vectorization means a parallelized calculation using SIMD instructions. A Xeon Phi coprocessor supports 512 bit SIMD operations, so that 16 single precision calculations can be computed simultaneously. On the other hand, this means that the actual performance of Xeon Phi system is highly dependent on the vectorization of the code.

	Intel Xeon Phi		NVIDIA GPU	
	7110X	5110P	Tesla K20X	GTX Titan
# of Cores	61	60	2688	2688
Core Clock (MHz)	1333	1053	732	837
SP TFLOPS	2.44	2.02	3.95	4.5
DP TFLOPS	1.22	1.01	1.31	1.27
Memory Size (GB)	16	8	6	6.1
Mem. Bandwidth (GB/s)	352	320	250	288
Price (USD)	4130	2650	3800	1100

Table 3: specification comparison between Intel Xeon Phi coprocessors and NVIDIA Kepler GPUs

The problem is that, it is **not always** possible to convert the code into a vectorized one. The first difficulty is that one must program the code in the level of the assembly language to control the array structure of the SIMD registers. Unfortunately, the C level compiler cannot do this job automatically to our satisfaction [9]. The second difficulty is that our QCD code is not, in general, designed to fit it into the structure format of specific SIMD registers required by the vectorization. Hence, in practice, the gain of 16 in vectorization is useless to us. Therefore, in the end of day we find out that the real performance of Xeon Phi systems is inferior to that of GTX Titan by a factor of about 10.



Device	CG time (ratio)
GTX 480	186 s (1)
GTX Titan	111 s (0.60)
Xeon Phi 5110P	1025 s (5.5)

Figure 6: CG performance by GTX 480, GTX Titan, and Xeon Phi. Here, CG time means the time (in the unit of second) which it takes to run the CG code 10 times in a single GPU or a single Xeon Phi.

Figure 6 shows the performance of the CG solver by Xeon Phi 5110P compared with that by GTX 480 and GTX Titan. Even though the code is vectorized by Intel compiler, Xeon Phi 5110P solves CG about 5 times and 10 times slower than GTX 480 and GTX Titan respectively.

In addition to the vectorization, the scalability of the code also could be an important factor of the performance of the Xeon Phi system. Each core of a Xeon Phi coprocessor supports 4 hardware threads, thus about 240 parallel processes can run on it at the same time. However, if the code has bad scalability, its performance with those 240 parallel MPI processes would be bad, too.

To resolve this issue, one can consider OpenMP. By virtue of its memory sharing functionality, this multi-threading method usually works better with a badly scaling code than the MPI of multi-

processing method. There is a result where they obtained a reasonable performance by Xeon Phi system [10]. Indeed, they not only optimized their code properly by considering the vectorization, but also adopted the OpenMP as a parallelization method.

5. Conclusion

GTX Titan provides 1.15 GFLOPS per USD of double precision performance, which is much better than 0.35 of Tesla K20X and 0.38 of Xeon Phi 5110P. Besides the theoretical performance, there are many other improvements on Kepler GPUs, such as direct parallelism, Hyper-Q and GPU Direct RDMA. By applying GPU Direct v2 to two GPUs on the same PCIe bus, we achieved about 3 times gain in data transfer. We also investigated the Xeon Phi system. However, the performance of Xeon Phi is so low (by a factor of 10) that we do not recommend using Xeon Phi systems for the lattice QCD simulation yet.

6. Acknowledgement

The research of W. Lee is supported by the Creative Research Initiatives program (2013-003454) of the NRF grant funded by the Korean government (MSIP). This work was supported by SNU Undergraduate Research Program. This work was supported by Seoul Science High School R&E program. W. Lee acknowledges support from the KISTI supercomputing center through the strategic support program [No. KSC-2012-G3-08].

References

[1] "NVIDIA GTX Titan." http://nvidianews.nvidia.com/Releases/NVIDIA-Introduces-GeForce-GTX-TITAN-DNA-of-the-World-s-Fastest-Supercomputer-Powered-by-World-s-Fa-925.aspx.

[2] "Wikipedia - Intel MIC." http://en.wikipedia.org/wiki/Intel_MIC.

[3] M. Clark, "Gpu computing with quda." https://www.olcf.ornl.gov/wp-content/uploads/2013/02/Clark_M_LQCD.pdf.

[4] H.-J. Kim, "Gpu computations for lattice qcd." http://thy.phy.bnl.gov/www/lunchtalks/20130523_Hyung-Jin_Kim.pdf.

[5] D. B. Kirk and W. mi W. Hwu, *Programming Massively Parallel Processors : A Hands-on Approach*. Elsevier Science, 2010. ISBN:0123814723.

[6] "Kepler tuning guide." http://docs.nvidia.com/cuda/kepler-tuning-guide/.

[7] "Kepler architecture." http://www.nvidia.com/object/nvidia-kepler.html.

[8] "NVIDIA GPUDirect." http://developer.nvidia.com/gpudirect.

[9] H. Jeong, S. Kim, W. Lee, and S.-H. Myung, *Performance of sse and avx instruction sets*, *PoS (Lattice 2012)* (2012), no. 249. arXiv:1211.0820[hep-lat].

[10] B. Joo, D. D. Kalamkar, K. Vaidyanathan, M. Smelyanskiy, K. Pamnany, V. W. Lee, P. Dubey, and W. W. III, "Lattice qcd on intel xeon phi." http://software.intel.com/en-us/articles/optimizing-lattice-qcd-on-intelr-xeon-phitm-coprocessor.