Using Gaming GPUs for Deep Learning

Extended Abstract

Gangwon Jo
ManyCoreSoft Co., Ltd.
Seoul, Korea
gangwon@manycoresoft.co.kr

Jungho Park
ManyCoreSoft Co., Ltd.
Seoul, Korea
jungho@manycoresoft.co.kr

Jaejin Lee
Dept. of Computer Science and Engineering
Seoul National University
Seoul, Korea
jaejin@snu.ac.kr

ABSTRACT

We propose two techniques to use cost-effective gaming GPUs as accelerators for deep learning, instead of expensive HPC-dedicated GPUs. A closed-circuit direct water cooling system keeps gaming GPUs at a low temperature. It ensures the long lifetime and reliability of gaming GPUs, rids the GPUs of memory errors, and reduces cooling fan noise. A VMDNN (virtual GPU memory for deep neural networks) library virtually expands the GPU memory space during the training period of a deep neural network model, by automatically swapping out unnecessary GPU memory object to the main memory. We present a gaming-GPU-based deep learning system that adopts these two techniques. The experimental result shows that the proposed system achieves 2–2.5x cost effectiveness compared to a system containing HPC-dedicated, high-end GPUs.

CCS CONCEPTS

• **Computer systems organization** → **Heterogeneous (hybrid) systems**; • **Hardware** → *Temperature optimization*; • **Software and its engineering** → *Virtual memory*; • **Computing methodologies** → Neural networks;

KEYWORDS

Deep learning, gaming GPUs, memory management, virtual memory, water cooling

ACM Reference Format:
Gangwon Jo, Jungho Park, and Jaejin Lee. 2018. Using Gaming GPUs for Deep Learning: Extended Abstract. In *Proceedings of International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2018).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

This work was supported by National Research Foundation of Korea grants funded by the Ministry of Science and ICT and the Ministry of Education: PF Class Heterogeneous High Performance Computer Development (No. 2016M3C4A7952587), Center for Manycore Programming (No. 2013R1A3A2003664), and BK21 Plus for Pioneers in Innovative Computing (No. 21A20151113068, Dept. of Computer Science and Engineering, SNU). ICT at Seoul National University provided research facilities for this study.

HPC Asia 2018, January 2018, Tokyo, Japan

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: Memory size of the state-of-the-art GPUs.

Category	GPU	Memory Size
Gaming GPUs	NVIDIA GeForce GTX 1080	8 GB
	NVIDIA GeForce GTX 1080 Ti	11 GB
	NVIDIA GeForce GTX Titan Xp	12 GB
HPC-dedicated GPUs	NVIDIA Tesla P100	12–16 GB
	NVIDIA Tesla P40	24 GB

1 INTRODUCTION

Deep learning is now one of the important applications of high-performance computing (HPC) systems. Especially, the massively parallel processing capability of the state-of-the-art GPUs makes deep neural networks (DNNs) to be trained in a reasonable time period and to be practically used in many areas. Unlike traditional HPC applications coming from computational science, training a DNN does not require double-precision floating-point operations. Thus, it can achieve good performance not only on expensive HPC-dedicated GPUs (*e.g.*, NVIDIA Tesla GPUs) but also on inexpensive gaming GPUs (*e.g.*, NVIDIA GeForce GPUs) that contain only a few double-precision units. For example, a popular DNN model called VGGNet [4] was originally trained on four NVIDIA GeForce GTX Titan Black gaming GPUs. Caffe [3] and TensorFlow [2] are two of the widely-used deep learning frameworks, and they were evaluated on an NVIDIA GeForce GTX 1080 gaming GPU and an NVIDIA GeForce GTX Titan X gaming GPU, respectively, at the time of their publication.

Although gaming GPUs performs comparably to or better than HPC-dedicated GPUs in terms of single-precision FLOPS, they also have some limitations for use in HPC systems and deep learning. First, gaming GPUs and their coolers are not designed for high-density systems. When multiple gaming GPUs are installed in a single system, it is hard to remove the large amount of heat from the GPUs. This may cause erroneous computations because gaming GPUs do not have ECC memory. This also shortens the lifetime of the GPUs. Moreover, such a system requires fast but very noisy cooling fans. Note that multi-GPU systems are very common in deep learning.

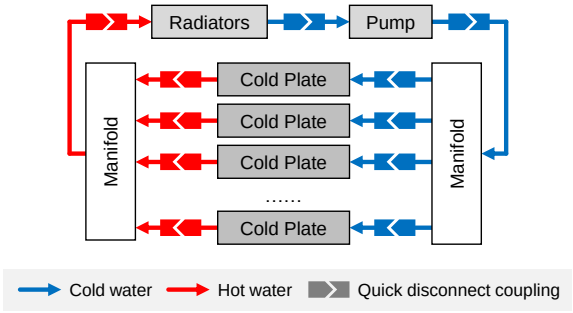

Figure 1: The structure of the direct water cooling system.

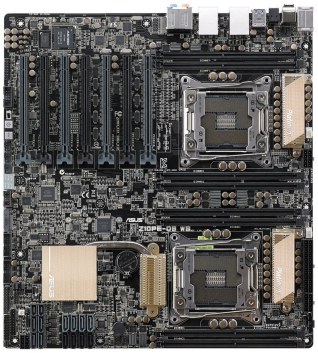

Figure 2: A commodity motherboard that provides seven PCIe x16 slots.

Figure 3: Seven water-cooled gaming GPUs installed on the motherboard in Figure 2.

Second, gaming GPUs have less memory than HPC-dedicated GPUs as shown in Table 1. On the other hand, since DNNs are becoming deeper and wider, a larger GPU memory capacity is strongly required. For example, training the VGG-16 network [4] with a batch size of 64 (*i.e.*, updating network parameters after feeding 64 input data to the network) requires about 10 GB of GPU memory. Thus, VGG-16 can be trained on an NVIDIA Tesla P100 HPC-dedicated GPU, but not on an NVIDIA GeForce GTX 1080 gaming GPU.

In this poster, we propose two techniques to overcome aforementioned problems and to use cost-effective gaming GPUs as accelerators for deep learning. A *closed-circuit direct water cooling system* keeps gaming GPUs at a low temperature and ensures their reliability. A *VMDNN (virtual GPU memory for deep neural networks) library* virtually expands the GPU memory space on demand during the training period of a deep neural network. Finally, we present *DEEP Gadget*, a gaming-GPU-based HPC system for deep learning that adopts these techniques. It achieves 2–2.5x cost effectiveness with seven gaming GPUs compared to a system containing four HPC-dedicated GPUs.

2 WATER COOLING

In these days, liquid cooling has been emerged as a promising cooling solution for high-density HPC systems and datacenters. Direct liquid cooling (also known as direct contact liquid cooling or direct-to-chip liquid cooling, but different from immersion cooling) is one of the major liquid cooling methods. It directly attaches a cold plate to a server component (*e.g.*, a GPU) and brings liquid into the plate. It usually uses water (chilled water or warm water) as coolant because water has a high thermal conductivity and is suitable for absorbing the heat through a limited liquid-contacting area within a small and thin cold plate.

Especially, liquid cooling is essential for gaming GPUs because it can provide much higher cooling performance than conventional air cooling. It ensures the long lifetime and reliability of gaming GPUs as with HPC-dedicated GPUs. It rids the GPUs of memory errors caused by high temperature. In addition, it significantly reduces cooling fan noise and makes a multi-GPU server work quietly even at full load.

Figure 1 shows the structure of the proposed direct water cooling system that can be used for gaming-GPU-based servers. A cold plate is attached to every GPU and (optionally) every CPU. All water cooling components including a pump, manifolds, cold plates, and radiators are installed in a server chassis and form a closed loop. Non-spill quick disconnect couplings help every component to be easily disconnected from the remaining system and replaced with a new one.

We have designed GPU cold plates that are much thinner than the previous ones available in the market. This makes a high-end gaming GPU to fit into a single PCIe slot, while the existing air-cooling and water-cooling products require a dual-slot space for a single GPU. Some of the commodity motherboards provide seven PCIe x16 slots as shown in Figure 2. Up to seven water-cooled gaming GPUs can be installed on such a motherboard as shown in Figure 3.

3 VMDNN: VIRTUAL GPU MEMORY FOR DEEP NEURAL NETWORKS

To overcome the limitation of GPU memory capacity, we propose a pure software-based solution called *VMDNN (virtual GPU memory for DNNs)*. It implements a GPU memory swapping mechanism similar to the paging mechanism of conventional operating systems.

When a deep learning framework such as Caffe and TensorFlow is running on a GPU, it allocates many GPU memory

objects to store all the parameters of a DNN model, such as input data, feature maps, weights, and gradients. Then, it runs various CUDA kernels on the GPU one by one. However, each kernel does not access all the GPU memory objects. It usually accesses only the memory objects related to a single layer of the neural network. VMDNN automatically swaps out some of the unnecessary GPU memory objects to the main memory for the current kernel. It also swaps in the GPU memory objects from the main memory for the next kernel. In addition, if the total size of GPU memory objects accessed by a single kernel is bigger than the GPU memory size, VMDNN divides the GPU memory objects into smaller pieces and executes the kernel multiple times with each piece at a time. This is possible because CUDA kernels executed during the training period usually process tens or hundreds of independent input data at the same time.

VMDNN minimizes the swapping overhead based on an observation that the deep learning framework repeats a set of CUDA kernel calls millions of times in a fixed order during the training period of a DNN model. GPU memory objects are allocated before the training period and reused whenever the CUDA kernels are executed again. VMDNN automatically detects a GPU memory-object access pattern that is repeated multiple times, and measures the execution time and the memory object usage of each kernel. Then, VMDNN generates an optimal swapping schedule to maximally hide memory transfers (swap-outs and swap-ins) with GPU computations.

VMDNN is implemented as a shared library and transparent to the target deep learning framework. We do not need to modify or recompile the source code of the deep learning framework. The shared library is executed with the framework by setting the environment variable *LD_PRELOAD*. It intercepts all CUDA kernel calls from the framework and performs the memory management. As a result, VMDNN is compatible with different versions of deep learning frameworks and backend libraries such as cuDNN [1]. In addition, VMDNN is available for users who download a deep learning framework as binary with a package manager or a Docker image.

4 DEEP GADGET

DEEP Gadget is a gaming-GPU-based HPC system for deep learning that adopts two techniques in Section 2 and 3. Table 2 shows an example of the system configuration. It contains two Intel Xeon E5-2630 v4 CPUs, seven NVIDIA GeForce GTX 1080 Ti gaming GPUs, and 256 GB main memory. Water cooling keeps the temperature of the GPUs about 70°C at full load. The VGG-16 network with a batch size of 512 requires about 60 GB of GPU memory, but this can be trained on a single GPU with the help of the VMDNN library. Note that the proposed techniques of this poster are not restricted to any specific model of CPUs or GPUs.

5 EVALUATION

We compare the performance of the DEEP Gadget system in Table 2 with a system containg four NVIDIA Tesla P100 HPC-dedicated GPUs. Table 3 describes the specification of the 4x P100 system. We train two popular DNN models, VGG-16 [4] and Inception-v4 [5], using Caffe [3] and measure the number of trained images per second. We use the same version of cuDNN in two systems because it largely affects the training performance.

Figure 4 shows the result. The performance of the DEEP Gadget system is slightly better than that of the 4x P100 system for both VGG-16 and Inception-v4 as shown in Figure 4 (a). However, the price of the DEEP Gadget system ($25,000) is much lower than that of the 4x P100 system

Table 2: A possible configuration of the DEEP Gadget system.

Component	Specification
CPU	2x Intel Xeon E5-2630 v4 (water-cooled)
GPU	7x NVIDIA GeForce GTX 1080 Ti (water-cooled)
Main memory	128 GB DDR4 2,400 MHz
Motherboard	ASUS Z10PE-D8 WS
(PCIe slots)	2x PCIe 3.0 @ x16, 5x PCIe 3.0 @ x8
Storage	250 GB M.2 NVMe SSD + 32 TB RAID 6 HDD storage (6x 8 TB SATA3 HDD)
Power supply	2x 1,300 W
OS	Ubuntu 16.04 LTS
Software	CUDA Toolkit 9.0, cuBLAS 9.0, cuDNN 6.0, NCCL, VMDNN library

Table 3: The specification of the 4x P100 system.

Component	Specification
CPU	2x Intel Xeon E5-2683 v4
GPU	4x NVIDIA Tesla P100 SXM2
Main memory	512 GB DDR4 2,133 MHz
OS	CentOS 7.2
Software	CUDA Toolkit 8.0, cuBLAS 8.0, cuDNN 6.0, NCCL

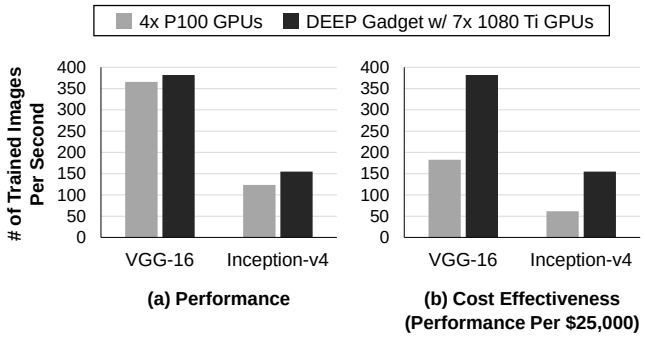

Figure 4: The training performance and the cost effectiveness (*i.e.*, performance per $25,000) of the 4x P100 system and the DEEP Gadget system with 7x 1080 Ti GPUs.

($50,000). Thus, the DEEP Gadget system achieves 2x and 2.5x cost effectiveness for VGG-16 and Inception-v4, respectively as shown in Figure 4 (b). This result shows that the DEEP Gadget system is appropriate and beneficial for deep learning research.

REFERENCES

[1] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient Primitives for Deep Learning. (2014). arXiv:1410.0759

[2] Martin Abadi et al. 2016. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*. 265–283.

[3] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. (2014). arXiv:1408.5093

[4] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. (2014). arXiv:1409.1556

[5] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. 2016. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. (2016). arXiv:1602.07261