Transparent GPU Memory Management for DNNs

Jungho Park
Dept. of Computer Science and Engineering
Seoul National University, Seoul 08826, Korea
jungho@aces.snu.ac.kr

Wookeun Jung
Dept. of Computer Science and Engineering
Seoul National University, Seoul 08826, Korea
wookeun@aces.snu.ac.kr

Hyungmin Cho
Dept. of Computer Engineering
Hongik University, Seoul 04066, Korea
hcho@hongik.ac.kr

Jaejin Lee
Dept. of Computer Science and Engineering
Seoul National University, Seoul 08826, Korea
jaejin@snu.ac.kr

Abstract

Modern DNN frameworks exploit GPU acceleration by default to achieve high performance. The limitation of GPU memory capacity becomes a serious problem because DNNs are becoming deeper and larger. This paper proposes a purely software-based transparent solution, called tvDNN, to the GPU memory capacity problem. It is based on GPU memory swapping and memory object sectioning techniques. It also provides an efficient memory-object swapping schedule based on ILP (optimal) and heuristics (suboptimal). The experimental results show that tvDNN enables Caffe to build VGG-16 with a large batch size, such as 256 or 512, using a few GB of GPU memory without significant performance degradation.

CCS Concepts • **Software and its engineering → Memory management**; **Virtual memory**; **Allocation / deallocation strategies**;

Keywords neural network, GPU, memory management

ACM Reference Format:
Jungho Park, Wookeun Jung, Hyungmin Cho, and Jaejin Lee. 2018. Transparent GPU Memory Management for DNNs. In *Proceedings of PPoPP '18: Principles and Practice of Parallel Programming, Vienna, Austria, February 24–28, 2018 (PPoPP '18),* 2 pages.
https://doi.org/10.1145/3178487.3178531

This work was supported by the National Research Foundation of Korea grants No. 2013R1A3A2003664, No. 2016M3C4A7952587, and No. 21A20151113068 funded by the Ministry of Science and ICT and the Ministry of Education. ICT at Seoul National University provided research facilities for this study.

PPoPP '18, February 24–28, 2018, Vienna, Austria

ACM ISBN 978-1-4503-4982-6/18/02…$15.00
https://doi.org/10.1145/3178487.3178531

1 Introduction

All popular DNN frameworks, such as Caffe [4] and TensorFlow [1], exploit GPUs by default. However, studies performed on accelerating DNNs with GPUs often address the limitation of GPU memory capacity. The GPU memory size (a few gigabytes) is typically smaller than the host (CPU) main memory size (hundreds of gigabytes). VGG-16 [6], a popular DNN model for image classification, typically uses a batch size of 256, and this requires more than 30 GB of GPU memory. As a result, VGG-16 cannot be trained on NVIDIA Tesla P100 (16 GB GPU memory) or NVIDIA Titan X (12 GB GPU memory) with this batch size. It is generally known that the input batch size affects the accuracy of a DNN model. The input batch size is chosen between one and a few hundreds, and there exists a sweet spot in this range [2]. The exploration of the optimal batch size for a DNN model requires a large GPU memory space. Moreover, since DNNs are becoming deeper and larger, a much bigger GPU memory capacity will be strongly required.

In this paper, we propose a pure software-based solution, called *tvDNN*, to the GPU memory capacity problem. It provides an automatic swapping mechanism for GPU memory objects to enable a DNN framework to build a DNN model that requires a larger memory space than that of the GPU memory. To effectively hide the memory transfer overhead with GPU computation, tvDNN implements two memory-transfers scheduling algorithms for swapping: *ILP based* and *heuristic based*. It also provides a technique called memory-object sectioning to support DNN configurations with a very large batch size by dividing the GPU memory objects into smaller ones. Since tvDNN is implemented as a shared library, it is transparent to the target DNN framework. It does not need any source code modification or recompilation of the target DNN framework.

Rhu *et al.* [5] propose a run-time DNN memory management mechanism called vDNN. Even though tvDNN and vDNN share the same motivations, they have significant differences. First, tvDNN can be applied transparently to the DNN framework. Second, it achieves optimal memory-transfer schedules. Finally, it implements the memory object sectioning technique for large batch sizes.

2 Design and Implementation

The tvDNN design is based on the following four key observations on DNN frameworks. First, DNN frameworks create very few GPU memory objects for each GPU task (a GPU kernel) and reuse them during the entire training iterations. Second, only the GPU memory objects that are accessed by the currently running GPU task are required to reside in the GPU memory. Third, the execution order of the GPU tasks and their memory object access patterns are fixed during the training iterations. Finally, all GPU tasks that require a large GPU memory space are data-parallel tasks. A data-parallel task is a GPU task that has no data dependence in each GPU memory object and across GPU memory objects accessed by the task itself.

Swapping mechanism. When a DNN framework is running on a GPU, GPU memory objects are allocated to store the parameters of the DNN model, such as input data, feature maps, gradients, and weights. A GPU task currently running on the GPU may not need all the memory objects. Only the GPU memory objects that are being accessed by it are required to reside in the GPU memory. Thus, we can swap out unnecessary GPU memory objects to the host main memory. When a GPU task needs a GPU memory object that has already been swapped out to the host main memory, we swap in the GPU memory object from the host main memory before the GPU task runs.

Optimal memory-transfer scheduling. However, a problem of this approach is swapping overhead. Since the same GPU memory-object access pattern is repeated in the entire training period, tvDNN initially performs profiling in a very short period of time to obtain the repeated memory-object access pattern. Implementing the on-demand swapping mechanism in the profiling phase, tvDNN measures the execution time and the memory-transfer time (swapping time) of each GPU task. Using the profiling result, tvDNN generates an optimal swapping schedule for the GPU memory objects to maximally hide the memory-transfer time with GPU computations. We implement two memory-transfer scheduling algorithms. One is based on Integer Linear Programming (ILP) and the other is based on heuristics.

Memory-object sectioning. Another problem arises when the total size of GPU memory objects accessed by a single GPU task is bigger than the GPU memory size. The swapping mechanism cannot be a solution to this problem. Instead, tvDNN implements a technique called *memory-object sectioning*. TvDNN divides each memory object accessed by a GPU task into smaller sections and applies the swapping mechanism to these sections. Then, tvDNN executes the GPU task multiple times, each time with a section. Memory-object sectioning requires that the GPU task must be a data-parallel GPU task. In our observation, all GPU tasks that require a large GPU memory space are data parallel.

Transparency. TvDNN is implemented as a shared library and transparent to the target DNN framework. The shared library is executed with the DNN framework by setting the environment variable *LD_PRELOAD*. It intercepts every GPU tasks from the DNN framework and manages the memory objects using the swapping mechanism. Therefore, no source code modification or recompilation of the target DNN framework is necessary for tvDNN.

3 Evaluation

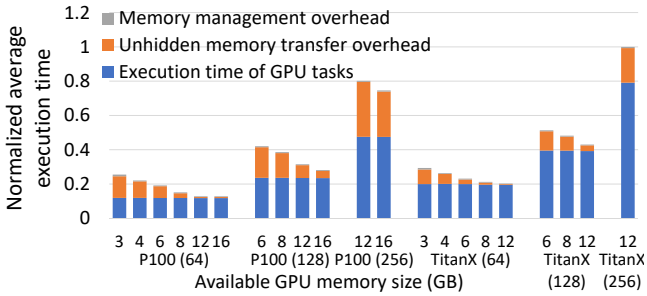

Figure 1. The normalized execution time of tvDNN with the heuristic-based scheduling algorithm on NVIDIA P100 and Titan X.

We evaluate tvDNN using Caffe with the VGG-16 model and ImageNet [3] data. Figure 1 shows the execution time of tvDNN with the heuristic scheduling algorithm in the management phase and without the memory-object sectioning technique. The numbers in parentheses show the batch size. The execution time is normalized to the execution time with a batch size of 256 on Titan X (Titan X (256)) with 12 GB of available GPU memory. The result shows that our work enables Caffe to build/run a VGG-16 that requires a larger memory space than the available GPU memory space of the target GPU.

References

[1] M. Abadi, A. Agarwal, P. Barham, and *et al.* 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. (2016). arXiv:1603.04467

[2] Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*. Springer, 437–478.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

[4] 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. *arXiv preprint arXiv:1408.5093* (2014).

[5] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler. 2016. vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–13. https://doi.org/10.1109/MICRO.2016.7783721

[6] K. Simonyan and A. Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. (2014). arXiv:1409.1556