ooc_cuDNN: A Deep Learning Library Supporting CNNs over GPU Memory capacity

Extended Abstract

Yuki Ito
Tokyo Institute of Technology
Tokyo, Japan
itou.y.aj@m.titech.ac.jp

Ryo Matsumiya
Tokyo Institute of Technology
Tokyo, Japan
matsumiya.r.aa@m.titech.ac.jp

Toshio Endo
Tokyo Institute of Technology
Tokyo, Japan
endo@is.titech.ac.jp

1 INTRODUCTION

Convolutional Neural Networks (CNNs) have achieved notable success particularly in the field of image recognition and image processing. Although there are many libraries which accelerate computation of CNNs with a GPU, few works can compute in case over GPU memory capacity. For example, learning of VGG16 network [1] with a 1024-images batch consumes about 60 GB memory. Unfortunately, a Tesla P100, which is a NVIDIA GPU recently released, has only 16 GB memory. Furthermore, such workloads with existing CNN libraries take significant large overhead due to CPU-GPU communication. In other words, swiftly computing CNNs over GPU memory capacity is an important issue.

We developed ooc_cuDNN[2] library, which enables a NVIDIA GPU to support CNNs consuming memory over GPU memory capacity. ooc_cuDNN's interface is compatible with cuDNN [3], a deep neural network library for NVIDIA GPUs. Our methodology of ooc_cuDNN is dividing the data into the parts and swapping them between the GPU memory and the host memory.

In this poster, we present design and implementation of ooc_cuDNN. ooc_cuDNN is implemented an optimization technique based on performance models we built. Additionally, ooc_cuDNN provides fused functions as combination of some computation to reduce extra communication. Thanks to ooc_cuDNN, a Tesla P100 can process a learning task which consumes larger than GPU memory capacity with 10-13 % overhead.

2 OOC_CUDNN

ooc_cuDNN enables a GPU to infer and/or learn in cases the batch, channels, and/or feature map are larger than the GPU memory capacity. Generally, interface of ooc_cuDNN is compatible with one of cuDNN. This section shows design and implementation of ooc_cuDNN.

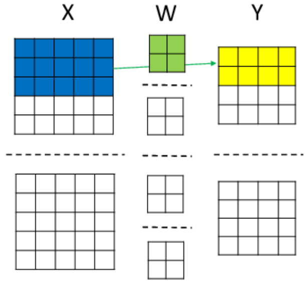

Figure 1: How to divide the data in ooc_cuDNN

2.1 Design

To process larger data than GPU memory capacity, we designed computation functions of ooc_cuDNN as follows:

- Input data and output data can be stored into either the host memory and the GPU memory.
- Batches, channels, and feature maps are divided. Then, the GPU computes each part.
- If the data needed for the computation are stored into the host memory, swapping between the host memory and the GPU memory is caused.
- Computation is executed with the GPU using cuDNN.
- To reduce the overhead, the communication is overlapped with the computation.

Fig. 1 shows how to divide the data in convolution computation. X denotes an input layer, W denotes an weight filter, and Y denotes an output layer. ooc_cuDNN divides X, Y, and W. Each part of Y (yellow cells) is computed based on the part of X (blue cells) and W (green cells).

2.2 Optimizing division sizes

In ooc_cuDNN, all layers and weight filters can be divided. This means that there are various ways to divide them. The division sizes are affected by ooc_cuDNN's performance. To optimize division sizes, we modeled ooc_cuDNN's performance. Based on this model, ooc_cuDNN decides division sizes.

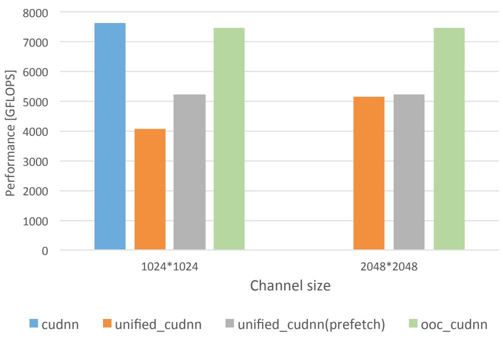

Figure 2: Result of micro benchmark

Our performance model is built considering with the performance of cuDNN and the communication between the CPU and the GPU. The division sizes are chosen to maximize the performance based on our model.

2.3 Fused functions

Our method described above enables a GPU to compute convolution fast even if the data are larger than the GPU memory capacity. On the other hand, complexity of some computation in CNNs (bias, activating, and pooling) are so small that the communication time is much larger than the computation time. Such small computation causes large overhead because they cannot hide the communication.

ooc_cuDNN provides fused functions. A fused function conducts a convolution computation and some of the small computation at once. Communication for all computation in a fused function are aggregated. Therefore, the communication can be hidden mainly by convolution computation.

3 PERFORMANCE EVALUATION

We measured ooc_cuDNN's performance. Our measurement used a computer with a NVIDIA Tesla P100, which has 16 GB memory.

3.1 Micro benchmark

We measured ooc_cuDNN's performance with a program computing a convolution. The program using ooc_cuDNN is compared to three programs as follows:

- cuDNN: using original cuDNN.
- unified_cuDNN: using original cuDNN but the memory region is allocated as Unified Memory, an automatic swapping mechanism introduced to recent NVIDIA GPUs.

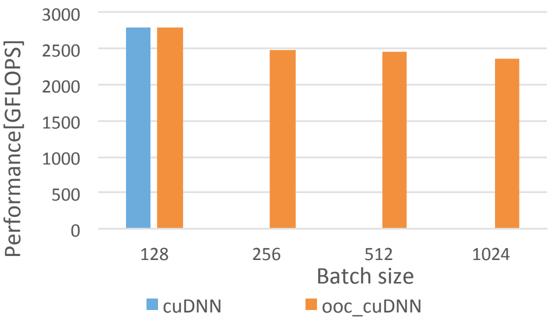

Figure 3: Performance of VGG16 with our C++ program

- unified_cuDNN (prefetch): same as unified_cuDNN except the memory region is prefetched to GPU memory by `cudaMemPrefetchAsync()`.

The input data size was 1024×1024, the batch size was 1, and the filters sizes were 3×3.

The result is shown as Fig. 2. cuDNN could not compute when the channel size was 2048×2048. This is because the data are larger than the GPU memory capacity.

The other programs could compute even the larger channel case. ooc_cuDNN computes 1.45x faster than unified_cuDNN and 1.42x faster than unified_cuDNN (prefetch). There are two reasons:

- The communication could not overlap with the computation.
- Page faults caused in GPU were occurred.

As we described above, in ooc_cuDNN, the communication overlaps the computation. In addition, ooc_cuDNN does not cause such page faults.

3.2 Performance of VGG16

We measured ooc_cuDNN's forwarding and back-propagating performance using a practically used CNN, VGG16.

Our measurement executed forwarding once and back-propagation once with ooc_cuDNN and cuDNN. In this experiment, we used a C++ program that calls ooc_cuDNN or cuDNN functions without any frameworks. Also, we used fused functions when using ooc_cuDNN.

The result is shown as Fig. 3. When the batch size is greater than or equal to 256, the data are larger than the GPU memory capacity. Thus, cuDNN could not compute in those cases.

ooc_cuDNN could compute in such cases. Compared to cuDNN with a 128-images batch, ooc_cuDNN computes larger batches only with 10-13% overhead.

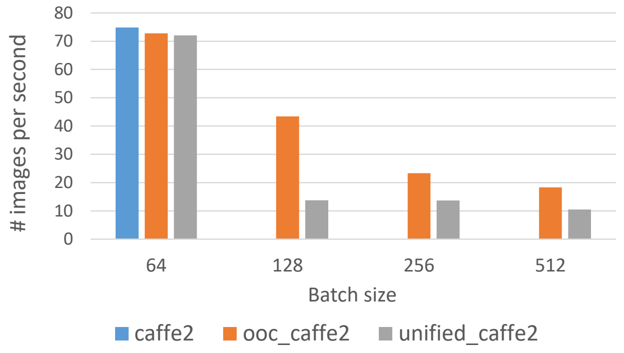

Figure 4: The number of images computed per second of VGG16 with Caffe2

3.3 Integrating with deep learning frameworks

We implemented Caffe2 with ooc_cuDNN and evaluated the performance. Caffe2[4] is an existing deep learning framework developed by Facebook.

Our measurement executed 100 traning iterations of VGG16 with original Caffe2, ooc_Caffe2 and unified_Caffe2 as follows:

- ooc_Caffe2 : using ooc_cuDNN functions in Caffe2 (We modified Caffe2's source code to support ooc_cuDNN). In current design, fused functions is not supported.
- unified_Caffe2 : using original cuDNN in Caffe2, and the memory region is allocated as Unified Memory.

The result is shown as Fig. 4. When the batch size is greater than or equal to 128, original Caffe2 could not compute because of GPU memory shortage. In contrast, ooc_Caffe2 and unified_Caffe2 could compute in such cases. When exceeding GPU memory capacity, ooc_Caffe2 could computes 1.7x-3.1x faster than unified_Caffe2.

In the current ooc_Caffe2, a large amount of CPU-GPU communication occurs and the overhead leads to performance degradation. Therefore, we will implement ooc_Caffe2 supporting fused functions to reduce the number of communication.

4 CONCLUSION

Most GPU libraries cannot compute CNNs which consume larger memory than GPU memory capacity. ooc_cuDNN enables such computation with GPUs.

Thanks to optimization based on ooc_cuDNN's performance model and aggregating communication between CPU and GPU, ooc_cuDNN can compute forwarding and back-propagating 10-13% overhead even if the data are larger than GPU memory capacity.

ACKNOWLEDGMENTS

This work is supported by JST-CREST.

REFERENCES

[1] K. Simonyan, et al. Convolutional Networks for Large-Scale Image Recognition. In *ICLR 2015*.

[2] Y. Ito, et al. ooc_cuDNN: Accommodating Convolutional Neural Networks over GPU Memory Capacity. In *IEEE Big Data 2017*.

[3] S. Chetlur, et al. cuDNN: Efficient Primitives for Deep Learning. https://arxiv.org/abs/1410.0759, 2014.

[4] Caffe2: A New Lightweight, Modular, and Scalabele Deep Learning Framework. https://caffe2.ai/, 2017.