CS\textsubscript{WAP}: A Self-Tuning Compression Framework for Accelerating Tensor Swapping in GPUs

Ping Chen*, Shuibing He*, Xuechen Zhang#, Shuaiben Chen*
Peiyi Hong*, Yanlong Yin$, Xian-He Sun+, Gang Chen*
Deep Neural Network is Popular

Explosive DNN Model Size
The Shortage of GPU Memory

https://www.microway.com/hpc-tech-tips/nvidia-tesla-v100-price-analysis/

<table>
<thead>
<tr>
<th>GPU</th>
<th>Memory/GB</th>
<th>Bandwidth</th>
<th>Tensor core</th>
<th>TFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100(SXM2)</td>
<td>32 HBM2</td>
<td>900</td>
<td>640</td>
<td>15.7</td>
</tr>
<tr>
<td>TITAN RTX</td>
<td>24 GDDR6</td>
<td>672</td>
<td>576</td>
<td>16.3</td>
</tr>
<tr>
<td>P100(SXM2)</td>
<td>16 HBM2</td>
<td>732</td>
<td>NA</td>
<td>10.6</td>
</tr>
<tr>
<td>TITAN V</td>
<td>12 HBM2</td>
<td>652.8</td>
<td>640</td>
<td>15</td>
</tr>
<tr>
<td>RTX 2080Ti</td>
<td>11 GDDR6</td>
<td>616</td>
<td>544</td>
<td>13.4</td>
</tr>
<tr>
<td>RTX 2080</td>
<td>8 GDDR6</td>
<td>448</td>
<td>368</td>
<td>10.1</td>
</tr>
<tr>
<td>RTX 2070</td>
<td>8 GDDR6</td>
<td>448</td>
<td>288</td>
<td>7.5</td>
</tr>
<tr>
<td>TITAN Xp</td>
<td>12 GDDR5X</td>
<td>547.7</td>
<td>NA</td>
<td>12</td>
</tr>
<tr>
<td>RTX 1080Ti</td>
<td>11 GDDR5X</td>
<td>484</td>
<td>NA</td>
<td>11.3</td>
</tr>
<tr>
<td>TITAN X</td>
<td>12 GDDR5</td>
<td>336.5</td>
<td>NA</td>
<td>11</td>
</tr>
<tr>
<td>GTX 1080</td>
<td>8 GDDR5X</td>
<td>484</td>
<td>NA</td>
<td>8.9</td>
</tr>
<tr>
<td>RTX 1070Ti</td>
<td>8 GDDR5</td>
<td>256</td>
<td>NA</td>
<td>8.1</td>
</tr>
<tr>
<td>RTX 1070</td>
<td>8 GDDR5</td>
<td>256</td>
<td>NA</td>
<td>6.5</td>
</tr>
<tr>
<td>RTX 1060</td>
<td>6 GDDR5</td>
<td>256</td>
<td>NA</td>
<td>4.4</td>
</tr>
</tbody>
</table>

Current GPU cannot support DNN training because of GPU memory shortage.

The current GPU cannot support DNN training because of GPU memory shortage.
The Background of Deep Neural Network

During Layer-N training procedure, GPU can only visit the tensors which have dependency with Layer-N.
The GPU–CPU Swapping Solution

Training Layer by layer on GPU

Slow swapping bandwidth takes much overhead

The Swapping with Compression Solution

ReLU Layers => Tensor Sparsity

Compressing all sparse tensors (after-ReLU layers) before swapping out and decompress them after swapping in.

Not Optimal

## Related Works (Swapping and Compression)

<table>
<thead>
<tr>
<th>Technique</th>
<th>Compression</th>
<th>Compression unit/location</th>
<th>Portability</th>
<th>Compression Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>vDNN [MICRO’16]</td>
<td>❌</td>
<td>N/A</td>
<td>✓</td>
<td>N/A</td>
</tr>
<tr>
<td>cDMA [HPCA’18]</td>
<td>✓</td>
<td>GPU</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>vDNN++ [IPDPS’19]</td>
<td>✓</td>
<td>CPU</td>
<td>✓</td>
<td>❌</td>
</tr>
<tr>
<td>CSwap [CLUSTER’21]</td>
<td>✓</td>
<td>GPU</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Compression Optimization (Tensor Selection): ❌ means compressing all sparse tensors without optimization or not.

Observation 1: Changing Sparsity of Tensors

The size and sparsity of DNN tensors

Some DNN tensors sparsity changes constantly during training the tensor size changes across layers.

*Figure: We evaluate ReLU output tensors in VGG16 on ImagNet. 50 epochs.*
Observation 2: Ineffectiveness of Compressing all Tensors

Some DNN tensors are unworthy being compressed.

*Figure: We evaluate on VGG16 across Imagenet.*
Observation 2: Ineffectiveness of Compressing all Tensors
Objectives of CSWAP

Software-level framework (Portability) + Self-tuning + Selective compression

CSWAP

Policy

With compression:
• ReLU[1-6]

Without compression:
• MAX[1-4], ReLU[7-8]
Challenges of CSWAP

Challenges 1: How to determine the compression policy for a sparse tensor?

- Different sparsity ①②;
- Different sizes ①②;
- Different overlap time ③;
- Different forward and backward time ④.

These metrics influence the overall training time.
Challenges of CSWAP

Challenges 2: How to predict the (de)compression time?

- Without (de)compression time, we cannot make decisions.

Important but uneasy to know

- GPU computation
- C/D
- GPU/CPU transfer

Diagram:
- Compression / Decompression
- Forward/ Backward
- Swap out / Swap in
- Overhead
Challenges of CSWAP

Challenges 3: the compression/decompression algorithm performance varies severely with different GPU settings.

- **Super parameters**: GPU has Grid size and Block size.
- Bruce force search (Grid search) needs hours.
Overview of CSWAP

1. **The tensor profiler**: Collecting tensor sparsity, size, and execution time of layers.
2. **Execution Advisor**: Making policy, includes compression decision and GPU settings for (de)compression operations.
3. **Swapping Executor**: DNN training.

![Diagram of CSWAP components](image-url)
1. Determining Cost-Effectiveness of Tensor Compression

- We compare the swapping cost with compression $T$ with the swapping cost without compression $T'$
  - $T' > T \Rightarrow \text{compression}$
  - $T' < T \Rightarrow \text{no compression}$

\[
T' = \max\left(\frac{\text{Size}^t}{\text{BW}_{d2h}} - \text{Hidden}_f^t, 0\right) + \max\left(\frac{\text{Size}^t}{\text{BW}_{h2d}} - \text{Hidden}_b^t, 0\right)
\]

\[
T = \text{Time}^t_c + \text{Time}^t_{dc} + O_f + O_b
\]

\[
O_f = \max\left(\frac{\text{Size}^t \times (1 - \text{Sparsity}^t)}{\text{BW}_{d2h}} - \text{Hidden}_f^t, 0\right)
\]

\[
O_b = \max\left(\frac{\text{Size}^t \times (1 - \text{Sparsity}^t)}{\text{BW}_{h2d}} - \text{Hidden}_b^t, 0\right)
\]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Profiling</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Size}^t$</td>
<td>size of tensor $t$</td>
<td>one time</td>
</tr>
<tr>
<td>$\text{BW}_{h2d}$</td>
<td>effective PCIe bandwidth from CPU to GPU</td>
<td>one time</td>
</tr>
<tr>
<td>$\text{BW}_{d2h}$</td>
<td>effective PCIe bandwidth from GPU to CPU</td>
<td>one time</td>
</tr>
<tr>
<td>$\text{Hidden}_f^t$</td>
<td>overlapped swapping latency in forward propagation of tensor $t$</td>
<td>one time</td>
</tr>
<tr>
<td>$\text{Hidden}_b^t$</td>
<td>overlapped swapping latency in backward propagation of tensor $t$</td>
<td>one time</td>
</tr>
<tr>
<td>$\text{Sparsity}^t$</td>
<td>sparsity of tensor $t$</td>
<td>epoch</td>
</tr>
<tr>
<td>$\text{Time}^t_c$</td>
<td>compression time of tensor $t$</td>
<td>offline</td>
</tr>
<tr>
<td>$\text{Time}^t_{dc}$</td>
<td>decompression time of tensor $t$</td>
<td>offline</td>
</tr>
</tbody>
</table>
2. Prediction of (De)compression Time

Size
Sparsity
Compression Algorithm

Figures: The accuracy of (de)compression time prediction using LR, BR, SVM, and DT models.
3. Setting GPU Parameters for Compression Kernels

Bayesian Optimization

Explore & Exploit => Fast and jump minimum point
Hours to near 1 minutes

Algorithm 1: BO search algorithm for choosing GPU parameters for (de)compression kernels

Require: $s_1$: the number of initial samples; $s_2$: the times of attempts to find the optimal solution;

1: bayes_opt ← new bayes_opt()  // Create a CSWAP BO search engine
2: $D \leftarrow \emptyset$  // Dataset of previously observed samples
3: for $i = 1, 2, ..., s_1$ do
4: $g \leftarrow \text{random}(0.4096)$  // $g$ denotes grid size
5: $b \leftarrow \text{random}(64, 128)$  // Set block size as 64 or 128
6: $p \leftarrow \text{sample}(g, b)$
7: $y \leftarrow \text{bayes_opt}.exec(p)$  // Obtain sum of $Time_c^b$ and $Time_{de}^b$
8: $D$.append($p, y$)  // Add the new sample to $D$
9: end for
10: bayes_opt.update($D$)  // Estimate posterior distribution and acquisition function
11: for $i = 1, 2, ..., s_2$ do
12: $p \leftarrow \text{bayes_opt}.select()$  // Select the next point to search
13: $y \leftarrow \text{bayes_opt}.exec(p)$
14: $D$.append($p, y$)
15: bayes_opt.update($D$)
16: end for
17: return bayes_opt.optimize($D$)  // Return an optimal point
Experimental Setting

➢ Platform1:
  • 2.60 GHz Intel(R) Xeon(R) Gold 6126 CPU
  • NVIDIA Tesla V100 GPU with 32 GB GPU memory

➢ Workloads and datasets
  • NN: AlexNet, Plain20, VGG16, MobileNet, ResNet and SqueezeNet (6)
  • Dataset: CIFAR10, ImageNet (2)

➢ Platform2:
  • 2.10 GHz Intel(R) Xeon(R) Gold 5218R CPUs
  • RTX 2080Ti GPU, and 11 GB GPU memory

➢ Baselines
  • vDNN[1], vDNN++[2], and cDMA[3]

CSWAP outperforms vDNN and vDNN++ by 25% and 190% on average.
Eval 2: Effectiveness of Dynamic Tensor Compression

CSWAP can improve the performance by 5.5% and 5.1% on average compared to cDMA.

Performance improvement of CSwap over the static compression (SC) scheme.
Eval 3: Effectiveness of Dynamic Tensor Compression

DNN training details using CSWAP
Thanks for your attention!
Appendix

Appendix-1: ReLU layers

<table>
<thead>
<tr>
<th>Model</th>
<th>ReLU layers</th>
<th>All layers</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>7</td>
<td>21</td>
<td>33%</td>
</tr>
<tr>
<td>VGG19</td>
<td>16</td>
<td>38</td>
<td>42%</td>
</tr>
<tr>
<td>SqueezeNet</td>
<td>26</td>
<td>57</td>
<td>46%</td>
</tr>
<tr>
<td>MobileNet</td>
<td>27</td>
<td>83</td>
<td>33%</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>64</td>
<td>205</td>
<td>31%</td>
</tr>
</tbody>
</table>

Figure 12: The compression decision accuracy based on the LR model.
Figure 13: The average training time of VGG16 for one iteration. RD: random search, EP: expert knowledge, BO: CSwap BO search, and GS: grid search.
Figure 11: (a) Computation time of the compression algorithms with the tensor sparsity of 60%. (b) The compression ratio with the tensor size of 50 MB. (c) Tensor swapping time. X_CT and X_ST denote the computation time and data swapping time using the compression algorithm X.