TDC: Towards Extremely Efficient CNNs on GPUs via Hardware-Aware Tucker Decomposition

Lizhi Xiang, Miao Yin, Chengming Zhang, Aravind Sukumaran-Rajam, P. Sadayappan, Bo Yuan, Dingwen Tao
AI is Everywhere
The development of GPU is significantly behind the expanding speed of DNN model size.

Ref: https://www.researchgate.net/publication/349044689_Freely_scalable_and_reconfigurable_optical_hardware_for_deep_learning

Ref: https://epochai.org/blog/trends-in-gpu-price-performance
Compression Techniques

Unstructured Pruning

Structured Pruning

Tensor Decomposition

\[ \mathbf{K} \rightarrow \mathbf{U}_1 \times \mathbf{C} \times \mathbf{U}_2 \]

\[ (C \times N \times R \times S) \rightarrow (C \times D_1) \times (D_1 \times D_2 \times R \times S) \times (N \times D_2) \]
Tucker Decomposition (TKD)

> Original kernel is decomposed into three kernels:

\[ K \times U_1 \times U_2 \]

- Avoid complex data structure
- Able to keep the spatial information
- Adjust D1 and D2 to control the entire computational cost under a target budget

> Tucker-format convolution (The original convolution is transformed to three small convolutions):

\[ \mathbf{x}, \mathbf{z}_1, \mathbf{z}_2, \mathbf{y} \]

- \( \mathbf{x} \) : \( H \times W \times C \)
- \( \mathbf{z}_1 \) : \( H \times W \times D_1 \)
- \( \mathbf{z}_2 \) : \( H' \times W' \times D_2 \)
- \( \mathbf{y} \) : \( H' \times W' \times N \)

| Original | TKD | \( C \times N \times R \times S \) | \( (C \times D_1) \) | \( (N \times D_2) \) |
|----------|-----|-------------------------------|-----------------|-----------------|
| 128 X 256 X 3 X 3 | \( \times \) | \( U_1 \) | \( C \) | \( U_2 \) |
| 128 X 32 X 1 X 1 | | | | |
| 32 X 32 X 3 X 3 | | | | |
| 32 X 256 X 1 X 1 | | | | |
## Discrepancy in Practice

|   | Description |
|---|-------------|
| **01** | Hard to train TKD compressed models |
| **02** | Lack of software-aware TKD convolution algorithms for CNN acceleration |
| **03** | Lack of performance-driven frameworks for highly efficient and accurate CNN inference on GPUs |
Optimized Training

> Challenges for training tucker-format models:
- Directly training Tucker-format models from scratch
  - Limited capacity -> accuracy degradation
- Initializing Tucker-format models from uncompressed models
  - Approximation error -> accuracy degradation

> Why Alternating Direction Method of Multipliers (ADMM)?
- Impose low-rankness corresponding to hardware performance
- Significantly preserve task accuracy

Accuracy comparison between directly training and our ADMM-based compression for ResNet-20 on CIFAR-10:

| Method            | Top-1 (%) | FLOPs↓ |
|-------------------|-----------|--------|
| Baseline          | 91.25     | N/A    |
| Direct Compression| 87.41     | 60%    |
| ADMM-based        | 91.02     | 60%    |
Optimized Training (ADMM-based Training)

Training objective: \[ \min_{\mathcal{K}} \ell(\mathcal{K}), \text{ s.t. } \text{rank}(\mathcal{K}) \leq \mathcal{P}_{\text{device}} \]

- \[ \min_{\mathcal{K}} \ell(\mathcal{K}), \text{ s.t. } \text{rank}(\mathcal{K}) \leq [D_1^*, D_2^*], \]
- \[ \min_{\mathcal{K}} \max_{\mathcal{K}, \mathcal{K} \in Q} \ell(\mathcal{K}) + \frac{\rho}{2} \| \mathcal{K} - \tilde{\mathcal{K}} + \mathcal{M} \|_F^2 - \frac{\rho}{2} \| \mathcal{M} \|_F^2, \]

where \( Q = \{ \tilde{\mathcal{K}} | \text{rank}(\tilde{\mathcal{K}}) \leq [D_1^*, D_2^*] \} \)

Training steps:

- \[ \mathcal{K} \leftarrow \mathcal{K} - \alpha \left( \frac{\partial \ell(\mathcal{K})}{\partial \mathcal{K}} + \rho (\mathcal{K} - \tilde{\mathcal{K}} + \mathcal{M}) \right), \]
- \[ \tilde{\mathcal{K}} \leftarrow \text{proj}(\mathcal{K} + \mathcal{M}) \]

- Hardware budget
- Selected ranks according to practical runtime of our kernel
- Truncated-HOSVD that truncates the smallest singular values
TDC: Convolution Kernel Design

Hard to translate flops reduction to actual performance improvement.

- Irregular convolution shape.
- Compute resource under-utilization.
TDC: Convolution Kernel Design

// Input: Input tensor \( \mathbf{X} \), Conv kernel \( \mathbf{K} \)
// Output: Output tensor \( \mathbf{Y} \)
shared input_tile[TC]((TH,R-1),(TW,S-1))
float temp_result[TH][TW], kernel[R][S]
unsigned int tile_tc_id = blockId/(TH*TH*W/TW)
unsigned int tile_id = blockId%(TH*TH*W/TW)
unsigned int tile_h_id = tile_id/(W/TW)
unsigned int tile_w_id = tile_id%(W/TW)
unsigned int output_n = threadIdx.x
// Copy tiled input tensor to global to shared
copy(input_tile, X)
syncthreads() // synchronize from global to shared
for c = 0 to TC:
copy(kernel, K,n,c+tile_tc_id*TC)
for (v,h,w) in (input_tile):
  for r = 0 to R
    for s = 0 to S
      y_out = h - r
      x_out = w - s
      if y_out<0 or x_out<0 or y_out>TH or x_out>TW:
        continue
      result = v * kernel[r][s]
      temp_result[y_out*TW+x_out] += result
// Write the output back to memory
for th to TH:
  for tw to TW:
    y = tile_id/(W/TW)+TH+th
    x = tile_id%(W/TW)+TW+tw
    atomicAdd(Y[HW+y,W+x,N+n], temp_result[th*TW+tw])
Analytical Modeling

- \(\text{num\_blks} = \frac{H}{TH} \times \frac{W}{TW} \times \frac{C}{TC}\)
- \(\text{waves} = \text{ceil} \left( \frac{\text{max\_ths} \times \text{occupancy}}{\text{num\_blks} \times \text{blk\_dim}} \right)\)
- \(\text{blks\_wave} = \frac{\text{max\_ths} \times \text{occupancy}}{\text{blk\_dim}}\)
- \(\text{flops\_blk} = 2 \times \frac{TH}{TW} \times TC \times N\)
- \(\text{f\_lops\_wave} = \text{blks\_wave} \times \text{f\_lops\_blk}\)
- \(\text{time\_wave} = \frac{\text{f\_lops\_wave}}{(\text{comp\_thr} \times \text{occupancy})}\)
- \(\text{estimated\_time} = \text{time\_wave} \times \text{waves}\)

Remove tiling configurations with high computation-latency

Pick the one with the lowest data movement
Hardware-aware Rank Determination

> Importance of rank D1 and D2:

• Task accuracy
• Practical speedup
• Overall computational cost

> Proposed rank search strategy:

[Diagram showing the process of rank determination with nodes labeled as 'Layer 1...N', 'Budget B', 'FLOPS reduction ratio', 'Performance Table T', 'Rank D1, D2', and additional notes on budget and FLOPS reduction ratios.]
Overview of our TDC framework for generating TKD-compressed CNN models with high-performance inference code on GPUs:

- Tucker-format Model
- ADMM-based Training
- Optimized Tucker Kernel Code Generator
- Potential Onboard Tucker Rank
- Optimal Rank Determination

| rank | latency |
|------|---------|
| 32, 32 | 0.002 ms |
Experiments

Accuracy table

| Model      | Compression Method | Top-1/Drop (%) | FLOPs↓ |
|------------|--------------------|----------------|--------|
| Original [14] | No compr.         | 69.75/0.00  | N/A    |
| FFGM [16]   | Pruning            | 68.41/-1.34  | 42%    |
| DSA [27]    | Pruning            | 68.61/-1.14  | 40%    |
| SCOP [37]   | Pruning            | 68.62/-1.13  | 45%    |
| TRP [40]    | MD                 | 65.51/-4.24  | 60%    |
| Stable [33] | CPD                | 69.06/-0.69  | 65%    |
| Opt. TT [42]| TTD                | 69.29/-0.46  | 60%    |
| Std. TKD [19]| TKD               | 66.65/-3.10  | 60%    |
| MUSCO [13]  | TKD                | 69.28/-0.47  | 58%    |
| TDC         | TKD                | 69.70/-0.05  | 63%    |

| Model      | Compression Method | Top-1/Drop (%) | FLOPs↓ |
|------------|--------------------|----------------|--------|
| Original [14] | No compr.         | 76.13/0.00  | N/A    |
| FFGM [16]   | Pruning            | 75.59/-0.54  | 42%    |
| HRank [24]  | Pruning            | 74.98/-1.15  | 44%    |
| TDC         | TKD                | 77.46/+1.33  | 40%    |
| Stable [33] | CPD                | 74.66/-1.47  | 60%    |
| TDC         | TKD                | 76.42/+0.29  | 60%    |

| Model      | Compression Method | Top-1/Drop (%) | FLOPs↓ |
|------------|--------------------|----------------|--------|
| Original [14] | No compr.         | 71.59/0.00  | N/A    |
| CC [22]     | MD                 | 68.81/-2.78  | 50%    |
| TDC         | TKD                | 71.62/+0.03  | 80%    |

| Model      | Compression Method | Top-1/Drop (%) | FLOPs↓ |
|------------|--------------------|----------------|--------|
| Original [14] | No compr.         | 74.43/0.00  | N/A    |
| TDC         | TKD                | 76.33/+1.90  | 10%    |

| Model      | Compression Method | Top-1/Drop (%) | FLOPs↓ |
|------------|--------------------|----------------|--------|
| Original [14] | No compr.         | 76.88/0.00  | N/A    |
| TDC         | TKD                | 76.92/+0.04  | 10%    |

Accuracy summary

- 0.05% accuracy loss on Resnet-18
- 0.29% accuracy increment on Resnet-50
- 0.03% accuracy increment on Vgg-16
- 1.90% accuracy increment on Densenet-121
- 0.04% accuracy increment on Densenet-201
Experiments

Layer-wise TDC kernel performance evaluation (On A100)
Experiments

End2end speedup comparison (On A100)

| Model  | TDC-oracle | TDC-modeling | TVM |
|--------|------------|--------------|-----|
| resnet18 | 3.27 | 3.14 | 2.92 |
| resnet50 | 1.7 | 1.68 | 1.67 |
| densenet121 | 2.14 | 2.11 | 2.08 |
| densenet201 | 1.7 | 1.68 | 1.63 |
| vgg16 | 2.37 | 2.33 | 2.17 |
Thank you!
Any questions are welcome

Contacts: Lizhi Xiang: u0814474@umail.utah.edu
Yinmiao: miao.yin@rutgers.edu