Resource Allocation of Non-linear Deep Neural Networks
(for presentation)

XX/XX/18
Behnam Pourghassemi, Joo Hwan Lee, Yang Seok Ki
MSL
outline

- Background
- Problem with existing frameworks
- Proposed solution
  - Allocating resources more efficient among layers
    - Approach
    - Motivation
    - Performance analysis using static causal resource profiling
  - Cost model for performance analysis
  - Implementation
    - Reducing complexity
    - Algorithm
  - Model architecture
Background

- Neural network
  - Contains sequence of dependent layers.
  - In each iteration, layers are executed on device in order.
Background

• Multi-path neural network
  – Multiple paths for different neural networks
    • E.g. multiple models in multi-armed bandit algorithms
  – Multiple paths in single neural network
    • E.g. multiple feature extractor and subsequent training model

• Critical path in multi-path neural network limits the network throughput
outline

- Background
- **Problem with existing frameworks**
- **Proposed solution**
  - Allocating resources more efficient among layers
    - Approach
    - Motivation
    - Performance analysis using static causal resource profiling
  - **Cost model for performance analysis**
  - **Implementation**
    - Reducing complexity
    - Algorithm
  - **Model architecture**
Problem statement

• Current deep learning frameworks are not optimized for multi-path neural network

• Inefficient resource allocation among layers is the primary source of performance overhead

• E.g. Tensorflow does not consider critical path in optimization
  – Maximum utilization of resources (cores, memory, etc.) on individual layer to improve execution time of that layer.
  – Optimizing layer’s execution time affects network performance only if the layer is on the critical path.
  – Inefficient resource utilization prevents concurrent execution of independent layers in multi-path neural network
Inefficient resource allocation cases

- Inefficient device core utilization in Tensorflow:
  - Utilize 100% of device (e.g. GPU) cores and device memory bandwidth for individual layer (kernel)
  - Prevent concurrent execution of independent kernels

Figure: Serial execution of layers on GPU from two different networks (VGG and AlexNet) running in one tensorflow session. The timeline snapshot is extracted from generated google traces via tensorflow.
Inefficient resource allocation cases

• Inefficient memory allocation in Tensorflow:
  – Lazy memory allocation in convolution layer.
    • Multiple implementations for convolution. E.g. Winograd, FFT, GEMM.
    • Different implementations require different amount of workspace memory.
    • For each convolution, pick the implementation with minimum execution time on target device based on convolution parameters.
    • Selected algorithm might require large amount of workspace memory.

  – Not a suitable approach for multi-path convolutional neural network specially on DRAM-limited devices.
Inefficient resource allocation

- Inefficient memory allocation drawback
  - Prevent concurrent execution of kernels if the total requested workspace memory is larger than available memory on device.
  - Example: two independent convolutions (conv1 and conv2) with following execution time and workspace running on GPU with 1.5GB available memory

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Workspace size</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPLICIT_GEMM</td>
<td>0</td>
<td>160ms</td>
</tr>
<tr>
<td>IMPLICIT_PRECOMP_GEMM</td>
<td>19KB</td>
<td>110ms</td>
</tr>
<tr>
<td>GEMM</td>
<td>1.9GB</td>
<td>142ms</td>
</tr>
<tr>
<td>WINOGRAD</td>
<td>2.1MB</td>
<td>72ms</td>
</tr>
<tr>
<td>WINOGRAD_NONFUSED</td>
<td>1.4GB</td>
<td>46ms</td>
</tr>
<tr>
<td>FFT</td>
<td>1.6GB</td>
<td>71ms</td>
</tr>
<tr>
<td>FFT_TILING</td>
<td>570MB</td>
<td>47ms</td>
</tr>
</tbody>
</table>

Tensorflow choice
- Cannot fit two convolutions in GPU
- Total exec time = 92 ms (sequential)

Optimum choice
- Can fit two convolutions in GPU
- Total exec time = 47 ms (parallel)

Input size: 56*56*128  filter size: 3*3*256  batch size: 128
Inefficient resource allocation

- Inefficient memory allocation drawback (ctd.)
  - Unbalanced memory utilization limits the network throughput
    - Lazy memory utilization for individual layer limits the batch size in training and/or inference which has an effect on the network throughput and application’s performance.
    - Example: VGG and Alexnet memory consumption snapshot on Quadro M400 GPU. Inefficient memory usage of few convolution layers limits the network batch size to 64.
Outline

- Background
- Problem with existing frameworks
- Proposed solution
  - Allocating resources more efficient among layers
    - Approach
    - Motivation
    - Performance analysis using static causal resource profiling
  - Cost model for performance analysis
  - Implementation
    - Reducing complexity
    - Algorithm
  - Model architecture
Approach

• Allocate resources more efficiently between layers to improve performance goal in multi-path neural network

• Performance goal examples
  – Network throughput (our focus)
  – Network delay
  – Application’s total execution time

• Resource allocation
  – Static: assign resources to individual layers before application starts (our focus)
  – Dynamic: resources are assigned during network iterations
    • Flexible with network changes (e.g. changes in topology, inputs, batch size, etc.)
    • Generally, more difficult to implement
Approach + Motivation

• Performance analysis approach
  – Analyze network performance with modification in resource utilization of layers → update resource utilization of individual layer(s) if performance analysis shows throughput improvement

– Causal Profiling for time domain (Motivation)
  • Suitable for performance profiling of parallel applications
  • Highlight most influential functions on total execution time
  • Quantifies effect of time speedup an individual function on total execution time
    – E.g. 20% speedup in function A can potentially improves application runtime by 5%
    – It virtually speedup selected function via slowing down concurrent execution paths
    – Compares execution time after virtual speedup with baseline to calculate effect of speedup of selected function on program.
Static causal profiling for resource utilization

- Apply causal profiling on resource utilization for multi-path neural network
  - Show effect of improving resource utilization for individual layer on total execution time.
  - Example: If convolution layer uses 10% and/or 20% less memory, how much network trains faster?
outline

- Background
- Problem with existing frameworks
- Proposed solution
  - Allocating resources more efficient among layers
    - Approach
    - Motivation
    - Performance analysis using static causal resource profiling
  - Cost model for performance analysis
- Implementation
  - Reducing complexity
  - Algorithm
- Model architecture
Cost model for causal resource profiling

• Causal resource profiling requires mapping between resource and time
  – Execution time of layers depends on resource utilization
    • E.g. using 20% less device cores in Maxpool layer will change execution time of that layer

• Solution
  – Cost model template of resources for common layers in neural networks
  – Typically contains discrete values in resource utilization (e.g. number of cores, workspace for different implementations of that layer)
Cost model template for resources

• Relates execution time of each layer to resource utilization
  – Resources
    • Device compute unit (CPU -> # core, GPU -> # SM, FPGA -> area)
    • Memory space
    • Memory bandwidth

– Example of memory space cost model template for convolution layer
outline

- Background
- Problem with existing frameworks
- Proposed solution
  - Allocating resources more efficient among layers
    - Approach
    - Motivation
    - Performance analysis using static causal resource profiling
  - Cost model for performance analysis
  - Implementation
    - Complexity
    - Algorithm
  - Model architecture
Resource allocation with cost model template

• Performance analysis of network via simulation of executing layers concurrently based on resource constrains
  – Inputs
    • \( k \) streams of execution.
    • For each stream \( s \) there are \( n_s \) sequentially dependent layers.
    • For each layer \( l \) there are \( m_l \) different implementations. Each of them has execution time \( t \) and set of resource utilization \( \{ w_r | r \in Resources \} \).
    • Parallel execution of layers changes the execution time of that layer:
      \[ t_{l,parallel} = f(t_{l,serial}, ...) \]
      (Function \( f \) depends on all concurrent layers)
  – Condition
    • \( \forall t \in \mathbb{R}, \forall r \in Resources, \sum_{l=1}^{K} w_{r,l} \cdot \alpha_{l,t} < M_r \) which \( K \) is the total number of layers, \( w_{r,l} \) is utilization of resource \( r \) by layer \( l \), \( M_r \) is maximum available resource \( r \) on device and \( \alpha_{l,t} \) is 1 if layer \( l \) is executing at time \( t \) and 0 if not.
  – Goal
    • Find the parallel scheduling an implementation of all layers that minimize makespan
Performance analysis with cost model template

• Finding the optimal parallel scheduling is NP-complete
  – Proof: It is NP-complete for simpler assumption

• Naïve solution
  – Find the optimum resource allocation with series of performance analysis assuming different resource utilization for each layer coming from cost model template for that layer.

  – Complexity
    • Requires $O(k^n)$ iterations for each resource which $n$ is depth of network and $k$ is average number of different resource utilizations per layer
    • E.g. for Resnet50 with 48 convolution layer and 8 different implementation for each convolution we need almost $2.2 \times 10^{43}$ performance analysis!
Reducing performance analysis complexity

• Reducing complexity of problem
  – Heuristic on function $f$
    • Parallel executing of layers doesn’t change the execution time of layer
    • Parallel executing of layers extends time of parallel execution homogenously with constant factor $\mu$ (e.g. $\mu = 1.5$, layer1 = 1 ms, layer2 = 2ms $\rightarrow$ layer1 = 1.5 ms, layer2 = $1.5 + 1 = 2.5$ ms)
  – Prune the search space
    • Prune implementation $i$ for layer $l$ if there exists another implementation $j$ for this layer with $t_i > t_j$ and $\forall r \in Resources$, $w_{r,i} > w_{r,j}$
    • Implementations with relatively long execution time are unlikely to be used in optimal solution even though they have small resource utilization.
      – E.g. prune implementation $k$ for layer L if $t_k > \beta \times \min\{t_i \mid i \in layer L implementations\}$
  – Window-based resource allocation of layers
Reducing performance analysis complexity

- Window-based resource allocation of layers
  - Create window based on average time of layers in each path (gathered after first iteration) and depth of the network
  - Typically between 5 to 10 layers on each stream
  - Performance analysis of layers inside the window
  - Find the optimal resource allocation and slide the window
Resource allocation inside window

- Scheduling simulation for all possible combination of implementation of layers inside the window.

- Scheduler
  - Parallel scheduling algorithm for one case
    1. Make a queue of layer for each stream based on their dependency
    2. Pick the stream with maximum idle time (the time that hasn’t scheduled any layer)
    3. If there are multiple streams with similar idle time, pick the stream that has shortest layer for scheduling
    4. Check if stream can schedule the first layer from its queue based on resource constrains. If yes schedule it and reset idle time for stream
    5. If stream cannot schedule layer, proceed its time to the first moment the layer meets the resource condition and increase its idle time by this amount
    6. Repeat from step 2 until all layers are scheduled

  - Report the window makespan
outline

- Background
- Problem with existing frameworks
- Proposed solution
  - Allocating resources more efficient among layers
    - Approach
    - Motivation
    - Performance analysis using static causal resource profiling
  - Cost model for performance analysis
- Implementation
  - Complexity
  - Algorithm
- Model architecture
Model architecture

- Model overview
  - May apply performance analysis for few iterations to improve efficiency of resource allocation