AutoTM: Automatic Tensor Movement in Heterogeneous Memory Systems using Integer Linear Programming

Mark Hildebrand\(^1\),
Jawad Khan\(^2\), Sanjeev Trika\(^2\),
Jason Lowe-Power\(^1\), Venkatesh Akella\(^1\)

\(^1\) University of California, Davis
\(^2\) Intel Corporation

https://github.com/darchr/AutoTM
March 12, 2020
Executive Summary

Problem
Automatic two-level memory management for Deep Neural Networks

Idea
- Profile Guided Optimization
- Model as an Integer Linear Program (ILP)

Results
- Replace 50-80% DRAM with NVDIMMs with geometric mean 27.1% performance loss.
- 3x better performance than real hardware cache.
Outline

Background

AutoTM
  Profiling
  ILP Modeling

Results

Wrap Up
Why Deep Neural Networks
Why Deep Neural Networks

Why Deep Neural Networks

Can we use multiple levels of memory to train large models on a single machine?

Heterogeneous Memory Systems

- Two types of memory.
- Same memory controller.
- Both are byte addressable.
- NVDIMMs for high capacity and low cost

NVDIMM Style

Compute (CPU) → Fast Memory (DRAM) → Slow Memory (NVDIMMs)
Heterogeneous Memory Systems

- Two types of memory.
- Same memory controller.
- Both are byte addressable.
- NVDIMMs for high *capacity* and low *cost*

**Challenges**
- All tensors in NVDIMMs memory is too slow.
- DRAM as a cache for NVDIMMs also too slow.
- Intelligent memory management required.
Outline

Background

AutoTM
  Profiling
  ILP Modeling

Results

Wrap Up
Goal

Minimize execution time
- Arbitrary computation graph
- Size constraint on fast memory
AutoTM

Goal
Minimize execution time
- Arbitrary computation graph
- Size constraint on fast memory

How
- Place tensors in fast or slow memory.
- Optimal tensor movement

Intermediate Tensors
Compute Kernels

- Producer of a tensor
- Use (read) of a tensor
- Last use of a tensor
Goal

Minimize execution time
• Arbitrary computation graph
• Size constraint on fast memory

How
• Place tensors in fast or slow memory.
• Optimal tensor movement

Strategy
• Profile kernel performance.
• Model tensor assignment as ILP.
Kernel Profiling

Profile performance of kernels for all tensor IO locations.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>IO Tensor Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>T2</td>
</tr>
<tr>
<td>DRAM</td>
<td>DRAM DRAM DRAM</td>
</tr>
<tr>
<td>DRAM</td>
<td>DRAM DRAM PMM</td>
</tr>
<tr>
<td>DRAM</td>
<td>PMM PMM PMM</td>
</tr>
<tr>
<td>PMM</td>
<td>DRAM DRAM DRAM</td>
</tr>
<tr>
<td>PMM</td>
<td>DRAM PMM PMM</td>
</tr>
<tr>
<td>PMM</td>
<td>PMM PMM DRAM</td>
</tr>
<tr>
<td>PMM</td>
<td>PMM PMM PMM</td>
</tr>
</tbody>
</table>

Table: Profile space for kernel K2.
Kernel Profiling

Profile performance of kernels for all tensor IO locations.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>IO Tensor Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1</td>
</tr>
<tr>
<td>K2</td>
<td>DRAM</td>
</tr>
<tr>
<td></td>
<td>DRAM</td>
</tr>
<tr>
<td></td>
<td>DRAM</td>
</tr>
<tr>
<td></td>
<td>PMM</td>
</tr>
<tr>
<td></td>
<td>PMM</td>
</tr>
<tr>
<td></td>
<td>PMM</td>
</tr>
<tr>
<td></td>
<td>PMM</td>
</tr>
</tbody>
</table>

Table: Profile space for kernel K2.
Path of flow through the graph describes where a tensor’s memory location throughout its lifetime.
Path of flow through the graph describes where a tensor’s memory location throughout its lifetime.
Path of flow through the graph describes where a tensor’s memory location throughout its lifetime.
Tensor Lifetime Flow Network

Path of flow through the graph describes where a tensor’s memory location throughout its lifetime.
Path of flow through the graph describes where a tensor’s memory location throughout its lifetime.
Path of flow through the graph describes where a tensor’s memory location throughout its lifetime.
Path of flow through the graph describes where a tensor’s memory location throughout its lifetime.
ILP Modeling

Objective Function

\[ \min \sum_{k \in K} \rho_k + \sum_{t \in T} M_t \]

Example Computation Graph
ILP Modeling

Objective Function

Computation time

$$\min \sum_{k \in K} \rho_k + \sum_{t \in T} M_t$$
Objective Function

Computation time

\[
\min \sum_{k \in \mathcal{K}} \rho_k + \sum_{t \in \mathcal{T}} M_t
\]

\( \mathcal{K} \): Set of Kernels

\( \rho_k \): Run time of kernel \( k \)
ILP Modeling

Objective Function

Computation time

$$\min \sum_{k \in K} \rho_k + \sum_{t \in T} M_t$$

**Kernel**

**Execution Time**

$K$: Set of Kernels

$\rho_k$: Run time of kernel $k$

**Example**

Run time of kernel $k_2$
ILP Modeling

Objective Function

Computation time

\[
\min \sum_{k \in K} \rho_k + \sum_{t \in T} M_t
\]

\( K \): Set of Kernels

\( T \): Set of Tensors

\( M_t \): Time moving tensor \( t \)
ILP Modeling

Objective Function

Computation time

\[
\min \sum_{k \in K} \rho_k + \sum_{t \in T} M_t
\]

- Kernel Execution Time
- Tensor Movement Time

\( T \): Set of Tensors

\( M_t \): Time moving tensor \( t \)

Example

Time moving tensor \( t_1 \)
ILP Modeling

Objective Function
Computation time

\[
\min \sum_{k \in K} \rho_k + \sum_{t \in T} M_t
\]

Constraints
Limit DRAM at each kernel

\[
\sum_{t \in \mathcal{L}(k)} \|t\|^{\text{DRAM}}_{t,k} \leq \text{Limit} \quad \forall k
\]
# Variations of AutoTM

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>PMM System</th>
<th>GPU System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>Tensor’s can’t move</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Synchronous</strong></td>
<td>Tensor’s move but block computation</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>Tensor movement concurrent with computation</td>
<td><img src="green" alt="Green" /></td>
<td>✓</td>
</tr>
</tbody>
</table>
Outline

Background

AutoTM
   Profiling
   ILP Modeling

Results

Wrap Up
Experiments!

Software
- Modified the *ngraph*\(^1\) compiler.
- Julia’s *JuMP*\(^2\) package for ILP modeling.
- Gurobi\(^3\) as the ILP solver.

Hardware
- 1.5 TB Optane\(^TM\) DC PMM
- 384 GiB DRAM

---

\(^1\)https://github.com/NervanaSystems/ngraph
\(^2\)https://github.com/JuliaOpt/JuMP.jl
\(^3\)gurobi.com
Experiments!

Software
• Modified the ngraph\(^1\) compiler.
• Julia’s JuMP\(^2\) package for ILP modeling.
• Gurobi\(^3\) as the ILP solver.

Hardware
• 1.5 TB Optane\(^{\text{TM}}\) DC PMM
• 384 GiB DRAM

Workloads

<table>
<thead>
<tr>
<th></th>
<th>Conventional</th>
<th>Batchsize</th>
<th>Memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception v4</td>
<td>1024</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>Vgg 19</td>
<td>2048</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>Resnet 200</td>
<td>512</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>DenseNet 264</td>
<td>512</td>
<td>115</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\)https://github.com/NervanaSystems/ngraph
\(^2\)https://github.com/JuliaOpt/JuMP.jl
\(^3\)gurobi.com
Experiments!

Software

- Modified the *ngraph*\(^1\) compiler.
- Julia’s *JuMP*\(^2\) package for ILP modeling.
- Gurobi\(^3\) as the ILP solver.

Hardware

- 1.5 TB Optane\(^{\text{TM}}\) DC PMM
- 384 GiB DRAM

Workloads

<table>
<thead>
<tr>
<th>Conventional</th>
<th>Batchsize</th>
<th>Memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception v4</td>
<td>1024</td>
<td>111</td>
</tr>
<tr>
<td>Vgg 19</td>
<td>2048</td>
<td>143</td>
</tr>
<tr>
<td>Resnet 200</td>
<td>512</td>
<td>132</td>
</tr>
<tr>
<td>DenseNet 264</td>
<td>512</td>
<td>115</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Large</th>
<th>Batchsize</th>
<th>Memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception v4</td>
<td>6144</td>
<td>659</td>
</tr>
<tr>
<td>Vgg 416</td>
<td>128</td>
<td>658</td>
</tr>
<tr>
<td>Resnet 200</td>
<td>2560</td>
<td>651</td>
</tr>
<tr>
<td>DenseNet 264</td>
<td>3072</td>
<td>688</td>
</tr>
</tbody>
</table>

---

\(^1\)https://github.com/NervanaSystems/ngraph
\(^2\)https://github.com/JuliaOpt/JuMP.jl
\(^3\)gurobi.com
Scaling Performance - Inception V4

Performance of Inception v4 - Batchsize 1024

Slowdown
Lower is Better

Dram Limit (GB)
Lower is Better

synchronous
Scaling Performance - Inception V4

Performance of Inception v4 - Batchsize 1024

- Slowdown
  - Lower is Better

Just using PMM is too slow.
```
Scaling Performance - Inception V4

Performance of Inception v4 - Batchsize 1024

- Just using PMM is too slow.
```
Scaling Performance - Inception V4

Performance of Inception v4 - Batchsize 1024

Dram Limit (GB)

Slowdown
Lower is Better

synchronous

Best performance when working-set fits in memory.
○ Best performance when **working-set fits in memory**.
Comparison Against 2LM

2LM DRAM Cache

Compute (CPU) → Fast Memory (DRAM - Cache) → Slow Memory (NVDIMMs)
Comparison Against 2LM

Speedup over 2LM
Higher is Better

2LM DRAM Cache
Comparison Against 2LM

Speedup over 2LM
Higher is Better

Vgg416 (320) Inception v4 (6144) Resnet200 (2560) DenseNet 264 (3072)

2LM DRAM Cache

- Avoid Dirty Writebacks
- Lower Memory Contention
Comparison Against 2LM

Speedup over 2LM
Higher is Better

Vgg416 (320) Inception v4 (6144) Resnet200 (2560) DenseNet 264 (3072)

0
1
2
3

2LM DRAM Cache

Software management outperforms hardware management by up to 3x.
Outline

Background

AutoTM
  Profiling
  ILP Modeling

Results

Wrap Up
Limitations

- Static computation graphs.
- Kernel profiling overhead.
- ILP solution times.
- ILP solution may be hard to interpret.
Limitations

- Static computation graphs.
- Kernel profiling overhead.
- ILP solution times.
- ILP solution may be hard to interpret.
Conclusion

AutoTM: A technique for managing tensors in heterogeneous memory systems.

• Profiling for Kernel Performance.
• Use ILP to optimally assign tensor location and movement.
• Three formulations: Static, Synchronous, Asynchronous.

We show

• Reduce DRAM requirement.
• Significant performance improvement over hardware solutions.

Code Available: https://github.com/darchr/AutoTM
Common Questions
Asynchronous Movement on PMMs

- Interference between DRAM and PMM.
- Low bandwidth and difficulty of DMA.
- Performance of kernels greatly impacted due to copy kernels.
GPU

Inception v4  Resnet200  DenseNet 264  Vgg19

Speedup over CudaMallocManaged

- blue: synchronous
- red: asynchronous
- brown: oracle
RNNs - more complex models

- AutoTM is limited to static computation graphs.
- RNNs have dynamic behavior (i.e. unrolling based on sequence length).
- RNNs can be implemented statically.
- Key ideas from AutoTM can be used for dynamic workloads.
Concluding Conclusion

AutoTM: A technique for managing tensors in heterogeneous memory systems.

- Profiling for Kernel Performance.
- Use ILP to optimally assign tensor location and movement.
- Three formulations: Static, Synchronous, Asynchronous.

We show

- Reduce DRAM requirement.
- Significant performance improvement over hardware solutions.

Code Available: https://github.com/darchr/AutoTM