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

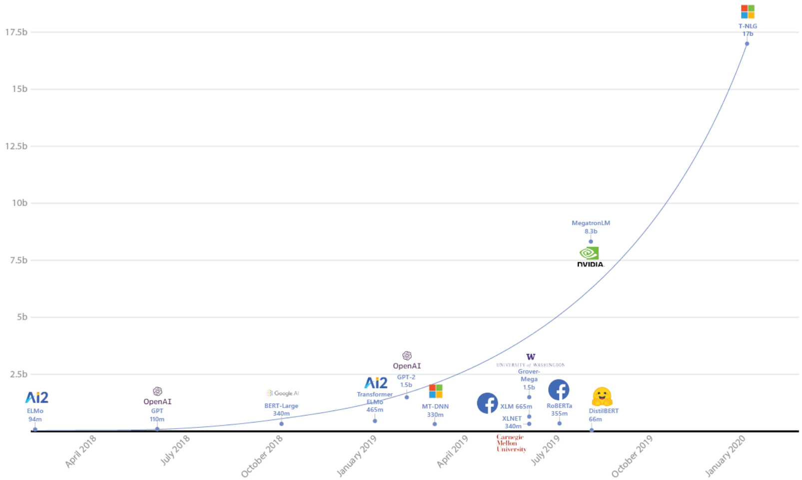
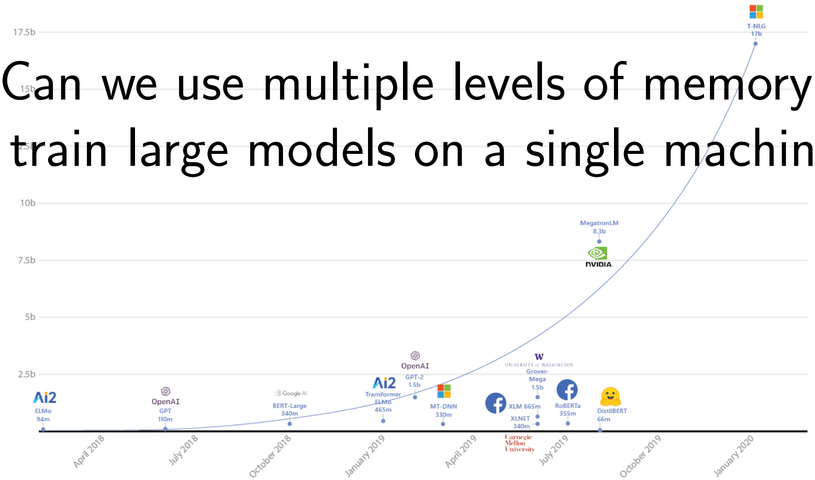


Image: https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/

Why Deep Neural Networks



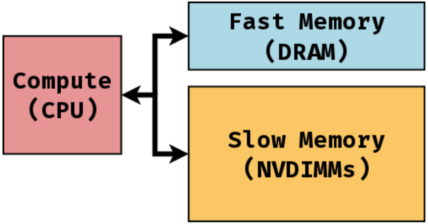
Can we use multiple levels of memory to train large models on a single machine?

Image: `https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/`

Heterogeneous Memory Systems

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

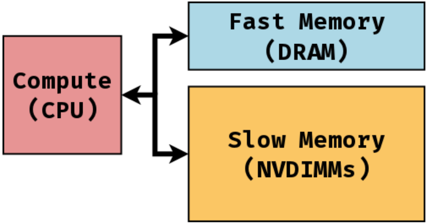
NVDIMM Style



Heterogeneous Memory Systems

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

NVDIMM Style



Challenges

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

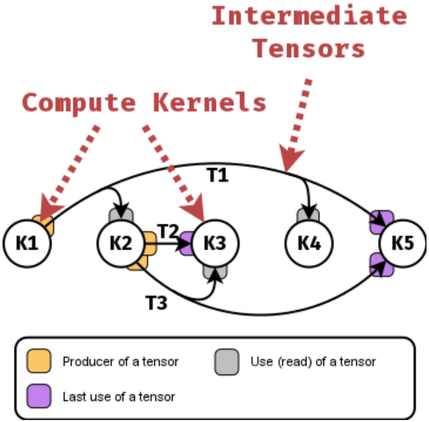
Outline

AutoTM

Goal

Minimize execution time

- Arbitrary computation graph
- Size constraint on **fast** memory



AutoTM

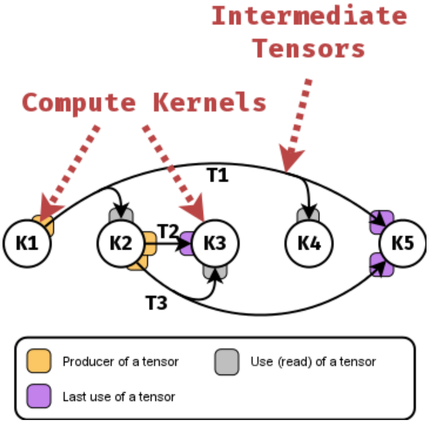
Goal

Minimize execution time

- Arbitrary computation graph
- Size constraint on **fast** memory

How

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



AutoTM

Goal

Minimize execution time

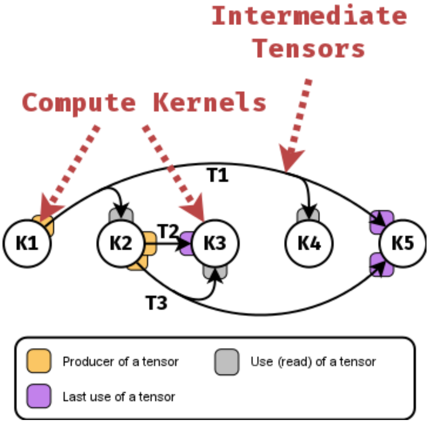
- Arbitrary computation graph
- Size constraint on **fast** memory

How

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

Strategy

- Profile kernel performance.
- Model tensor assignment as ILP.

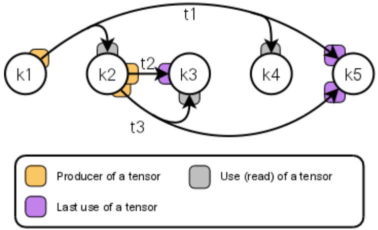


Kernel Profiling

Profile performance of kernels for all tensor IO locations.

Kernel	IO Tensor Locations		
	T1	T2	T3
K2	DRAM	DRAM	DRAM
	DRAM	DRAM	PMM
	DRAM	PMM	DRAM
	DRAM	PMM	PMM
	PMM	DRAM	DRAM
	PMM	DRAM	PMM
	PMM	PMM	DRAM
	PMM	PMM	PMM

Table: Profile space for kernel K2.

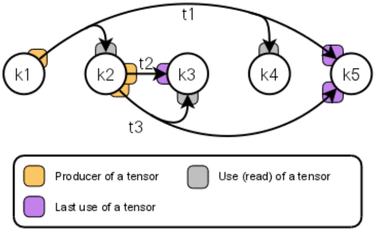


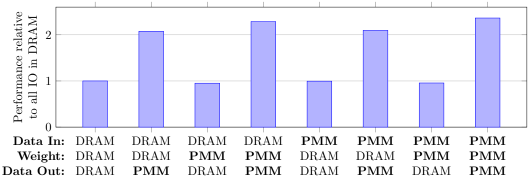
Kernel Profiling

Profile performance of kernels for all tensor IO locations.

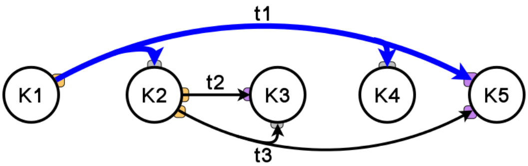
Kernel	IO Tensor Locations		
	T1	T2	T3
K2	DRAM	DRAM	DRAM
	DRAM	DRAM	PMM
	DRAM	PMM	DRAM
	DRAM	PMM	PMM
	PMM	DRAM	DRAM
	PMM	DRAM	PMM
	PMM	PMM	DRAM
	PMM	PMM	PMM

Table: Profile space for kernel K2.

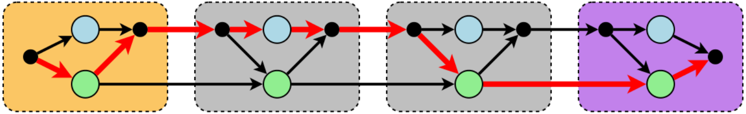




Tensor Lifetime Flow Network



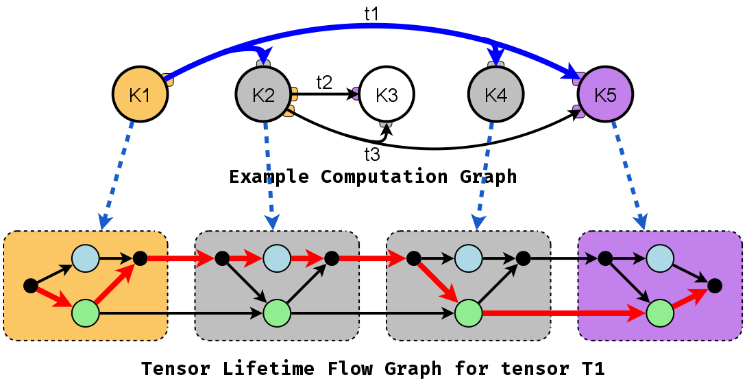
Example Computation Graph



Tensor Lifetime Flow Graph for tensor T1

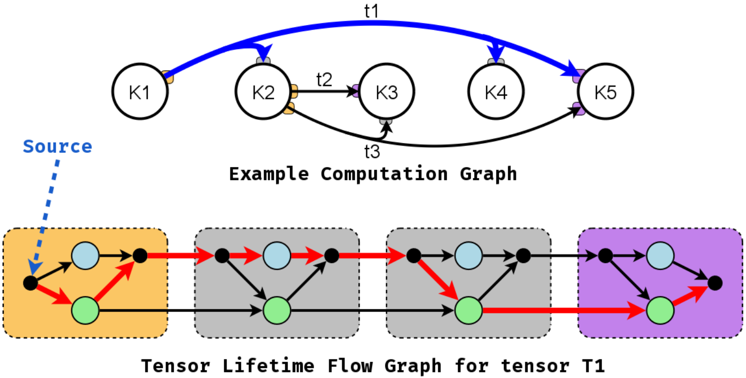
Path of flow through the graph describes where a tensor's memory location throughout its lifetime.

Tensor Lifetime Flow Network



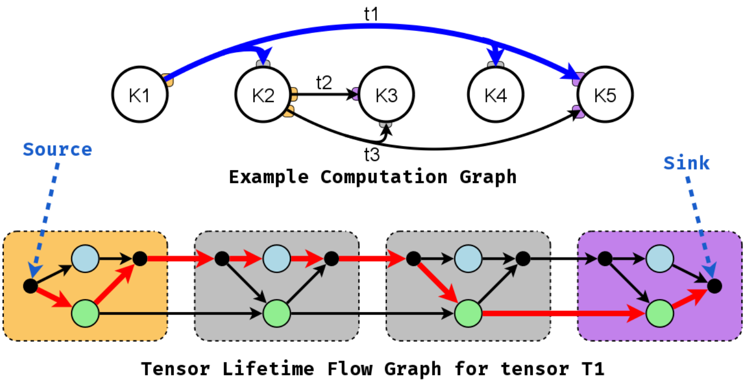
Path of flow through the graph describes where a tensor's memory location throughout its lifetime.

Tensor Lifetime Flow Network



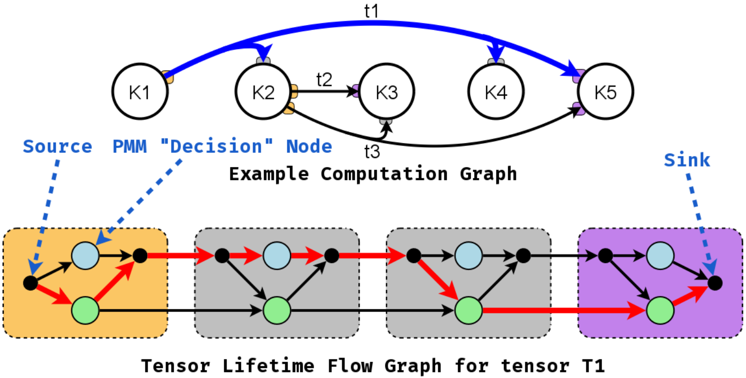
Path of flow through the graph describes where a tensor's memory location throughout its lifetime.

Tensor Lifetime Flow Network



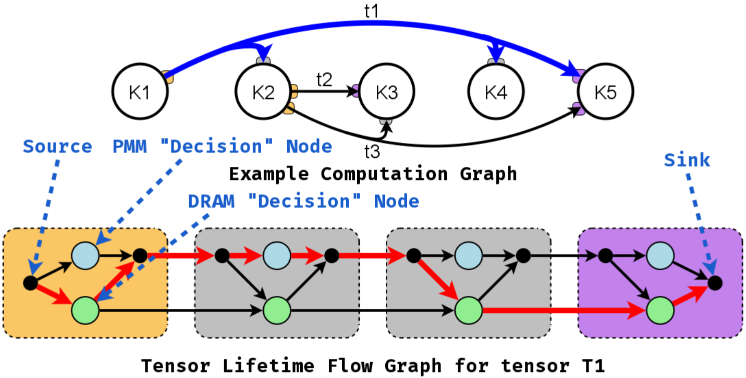
Path of flow through the graph describes where a tensor's memory location throughout its lifetime.

Tensor Lifetime Flow Network



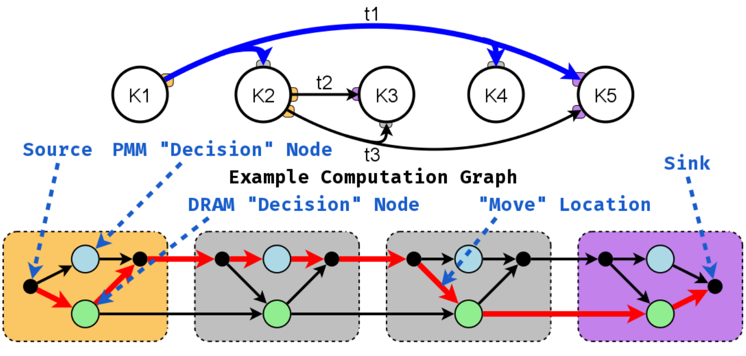
Path of flow through the graph describes where a tensor's memory location throughout its lifetime.

Tensor Lifetime Flow Network



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

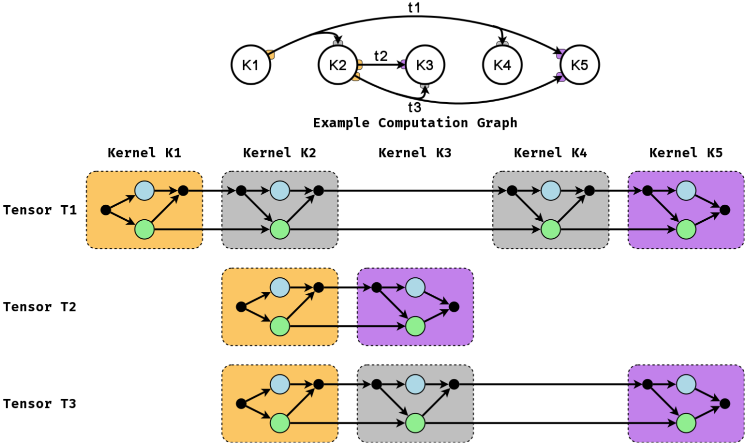
Tensor Lifetime Flow Network



Tensor Lifetime Flow Graph for tensor T1

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

ILP Modeling

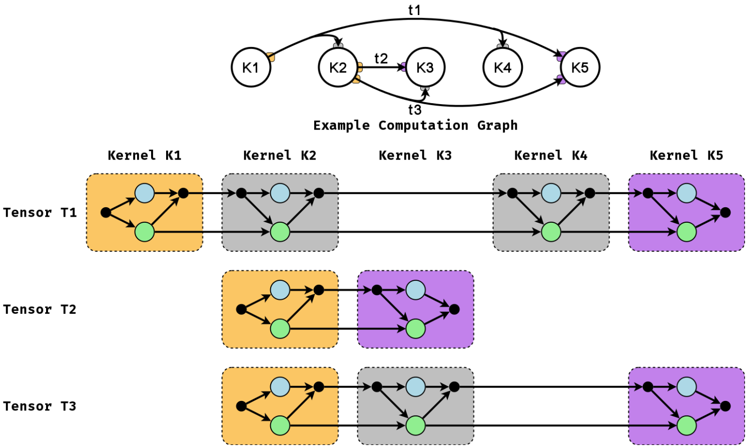


ILP Modeling

Objective Function

Computation time

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



ILP Modeling

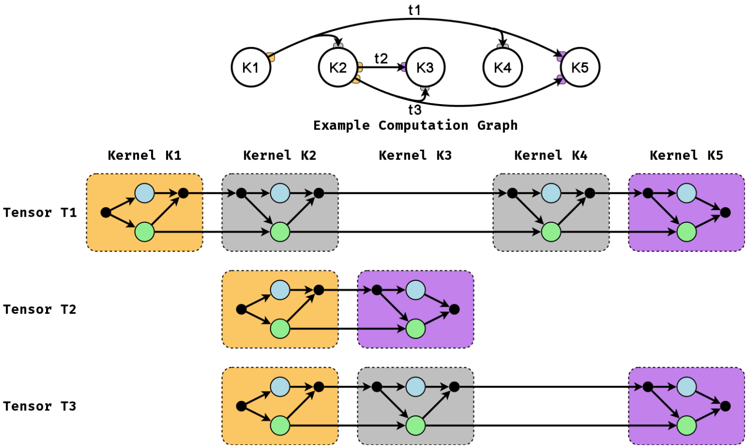
Objective Function

Computation time

$$\min \underbrace{\sum_{k \in \mathcal{K}} \rho_k}_{\substack{\text{Kernel} \\ \text{Execution} \\ \text{Time}}} + \sum_{t \in \mathcal{T}} M_t$$

$\mathcal{K}$: Set of Kernels

ρ_k : Run time of kernel k



ILP Modeling

Objective Function

Computation time

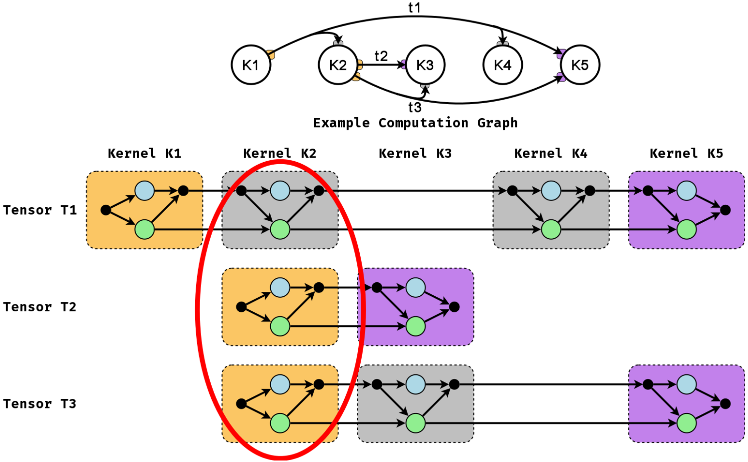
$$\min \underbrace{\sum_{k \in \mathcal{K}} \rho_k}_{\text{Kernel Execution Time}} + \sum_{t \in \mathcal{T}} M_t$$

$\mathcal{K}$: Set of Kernels

ρ_k : Run time of kernel k

Example

Run time of kernel k_2



ILP Modeling

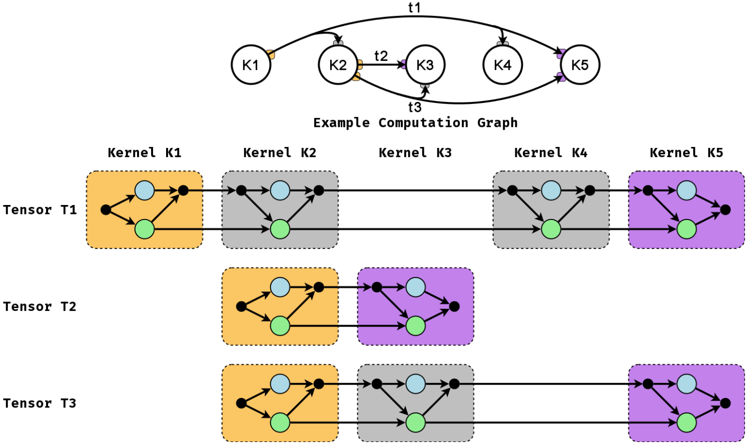
Objective Function

Computation time

$$\min \underbrace{\sum_{k \in \mathcal{K}} \rho_k}_{\substack{\text{Kernel} \\ \text{Execution} \\ \text{Time}}} + \underbrace{\sum_{t \in \mathcal{T}} M_t}_{\substack{\text{Tensor} \\ \text{Movement} \\ \text{Time}}}$$

$\mathcal{T}$: Set of Tensors

M_t : Time moving tensor t



ILP Modeling

Objective Function

Computation time

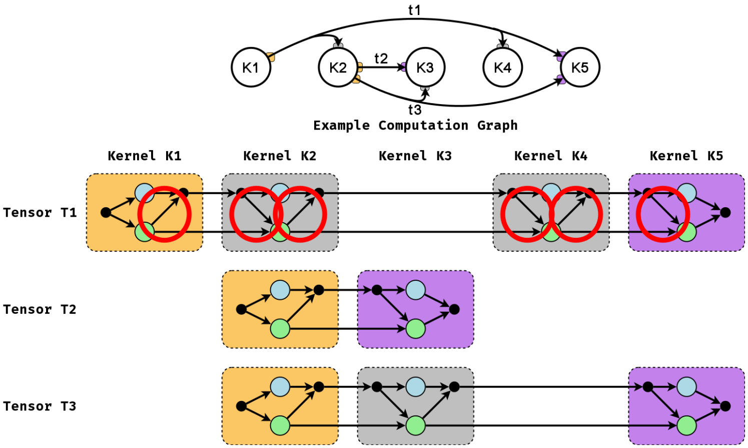
$$\min \underbrace{\sum_{k \in \mathcal{K}} \rho_k}_{\substack{\text{Kernel}\\\text{Execution}\\\text{Time}}} + \underbrace{\sum_{t \in \mathcal{T}} M_t}_{\substack{\text{Tensor}\\\text{Movement}\\\text{Time}}}$$

$\mathcal{T}$: Set of Tensors

M_t : Time moving tensor t

Example

Time moving tensor t_1



ILP Modeling

Objective Function

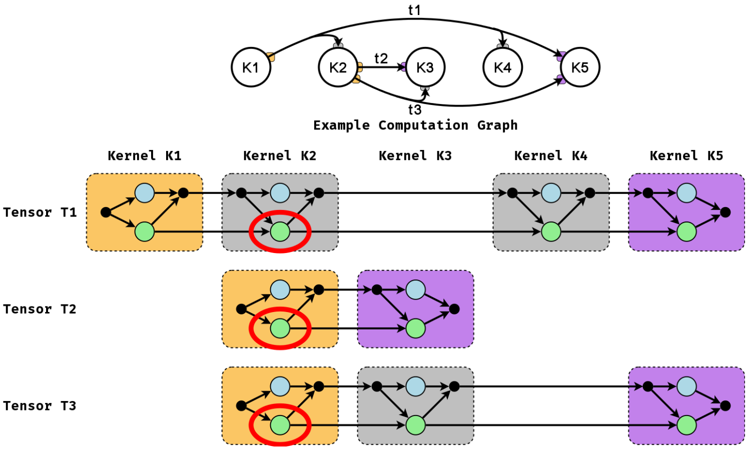
Computation time

$$\min \underbrace{\sum_{k \in \mathcal{K}} \rho_k}_{\text{Kernel Execution Time}} + \underbrace{\sum_{t \in \mathcal{T}} M_t}_{\text{Tensor Movement Time}}$$

Constraints

Limit DRAM at each kernel

$$\sum_{t \in \mathcal{L}(k)} \|t\| I_{t,k}^{\text{DRAM}} \leq \text{Limit} \quad \forall k$$



Variations of AutoTM

Name	Description	PMM System	GPU System
Static	Tensor's can't move	✓	✗
Synchronous	Tensor's move but block computation	✓	✓
Asynchronous	Tensor movement concurrent with computation	●	✓

Outline

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™ 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™ DC PMM
- 384 GiB DRAM

Workloads ⟶

Conventional	Batchsize	Memory (GB)
Inception v4	1024	111
Vgg 19	2048	143
Resnet 200	512	132
DenseNet 264	512	115

[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™ DC PMM
- 384 GiB DRAM

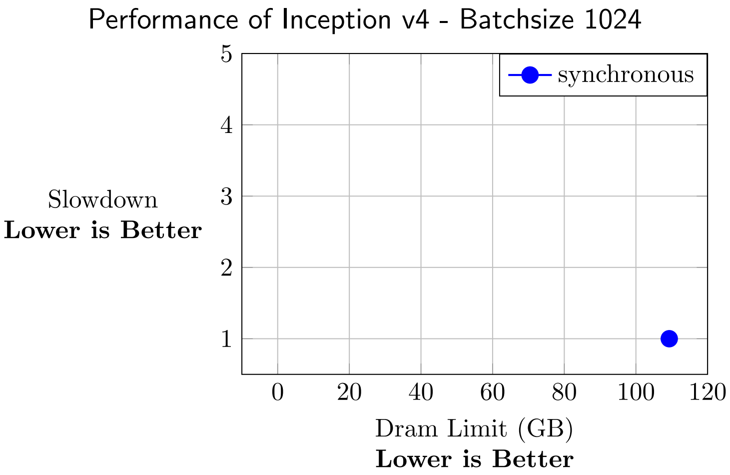
Workloads ⟶

Conventional	Batchsize	Memory (GB)
Inception v4	1024	111
Vgg 19	2048	143
Resnet 200	512	132
DenseNet 264	512	115

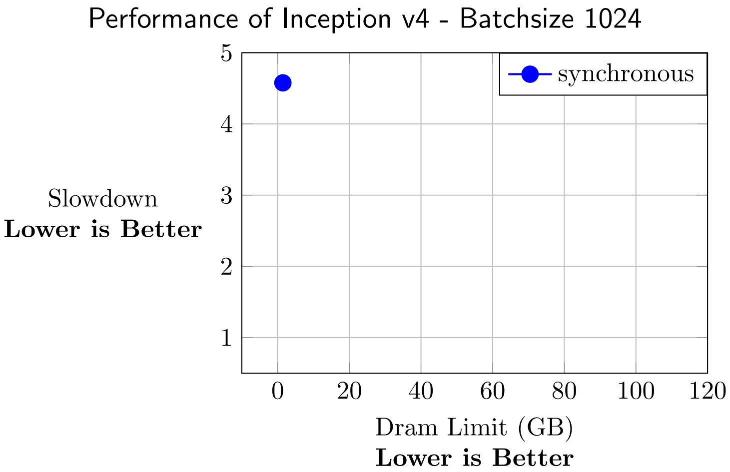
Large	Batchsize	Memory (GB)
Inception v4	6144	659
Vgg 416	128	658
Resnet 200	2560	651
DenseNet 264	3072	688

[1] https://github.com/NervanaSystems/ngraph
[2] https://github.com/JuliaOpt/JuMP.jl
[3] gurobi.com

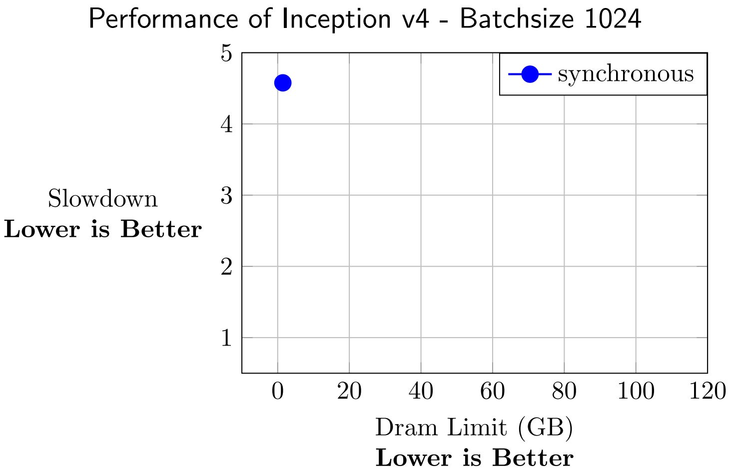
Scaling Performance - Inception V4



Scaling Performance - Inception V4

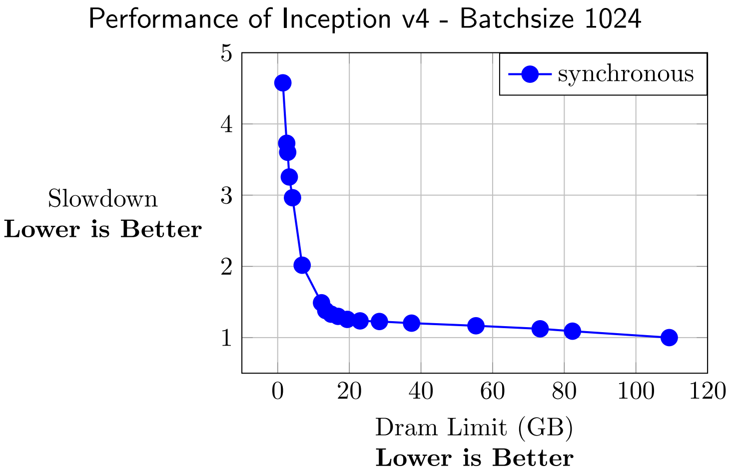


Scaling Performance - Inception V4

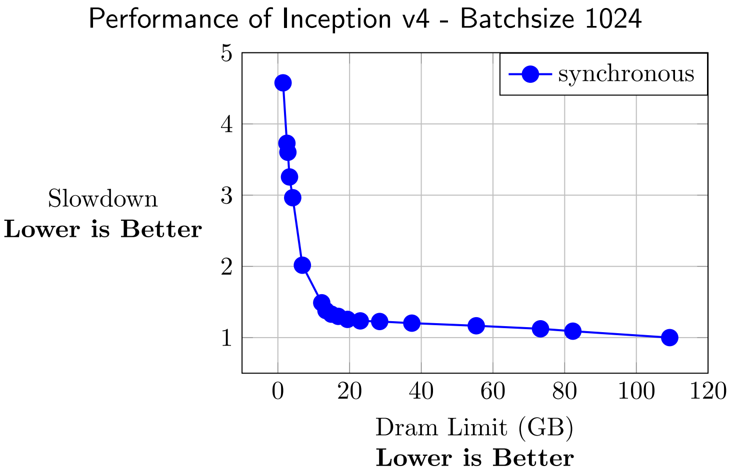


- Just using PMM is **too slow**.

Scaling Performance - Inception V4



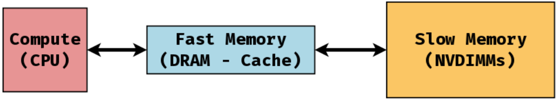
Scaling Performance - Inception V4



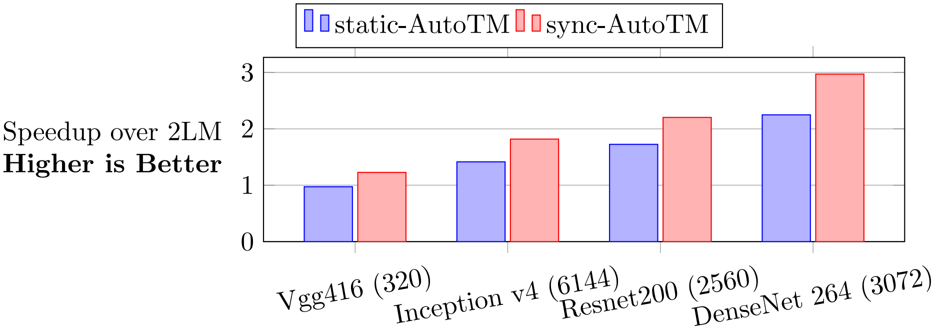
◦ Best performance when **working-set fits in memory**.

Comparison Against 2LM

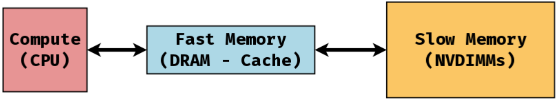
2LM DRAM Cache



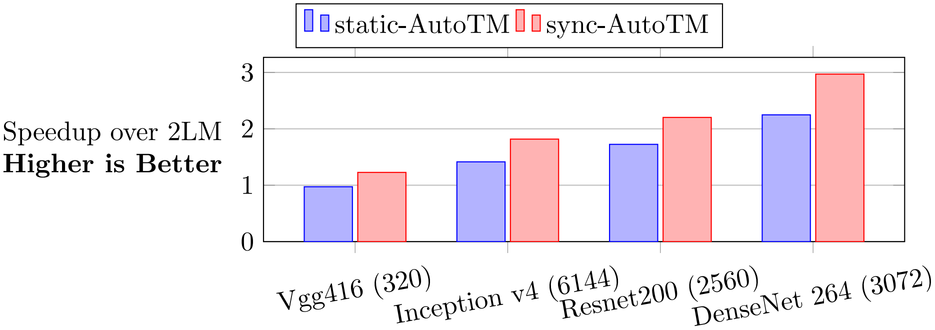
Comparison Against 2LM



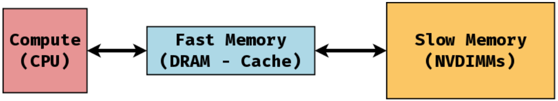
2LM DRAM Cache



Comparison Against 2LM

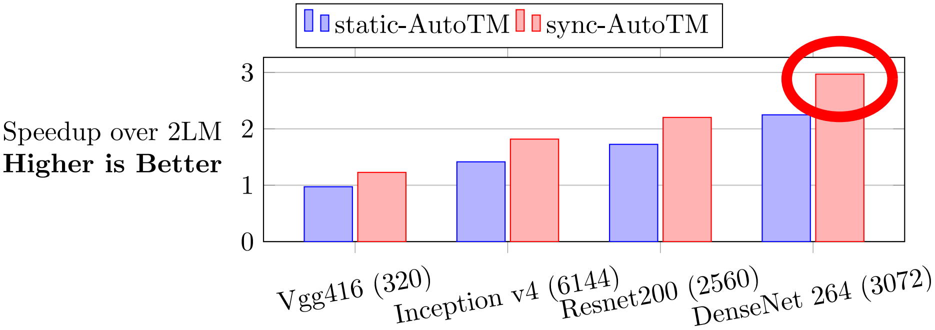


2LM DRAM Cache

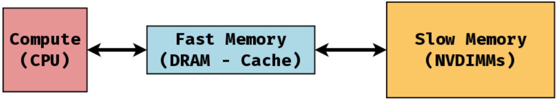


- **Avoid Dirty Writebacks**
- **Lower Memory Contention**

Comparison Against 2LM



2LM DRAM Cache



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

Outline

Background

AutoTM

- Profiling
- ILP Modeling

Results

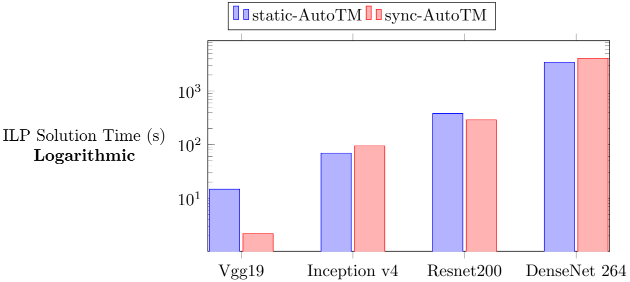
Wrap Up

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.

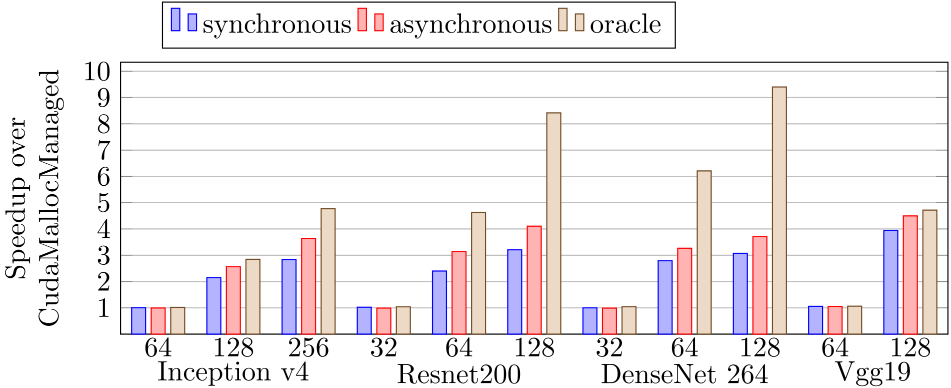
Code Available: `https://github.com/darchr/AutoTM`

Common Questions

Asynchronous Movement on PMMs

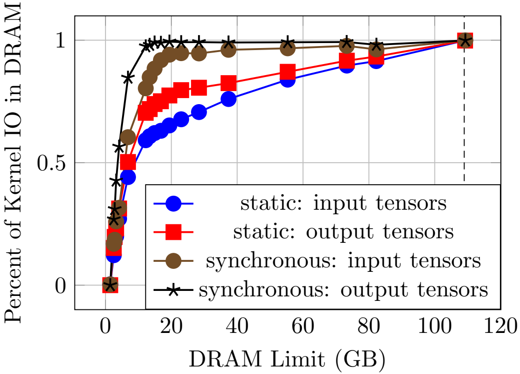
- Interference between DRAM and PMM.
- Low bandwidth and difficulty of DMA.
- Performance of kernels greatly impacted due to copy kernels.

GPU


synchronous
asynchronous
oracle
Speedup over CudaMallocManaged
10
9
8
7
6
5
4
3
2
1
64
128
256
32
64
128
32
64
128
64
128
Inception v4
Resnet200
DenseNet 264
Vgg19

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`