Sentinel: Efficient Tensor Migration and Allocation on Heterogeneous Memory Systems for Deep Learning

Jie Ren, Jiaolin Luo, Kai Wu, Minjia Zhang, Hyerin Jeon, and Dong Li

1 PASA Lab, University of California, Merced
2 Microsoft

The full paper is accepted in the 27th IEEE International Symposium on High-Performance Computer Architecture (HPCA'2021)
The Size of Deep Learning Model is Increasing Quickly

- ResNet50, 26 M
- OpenAI, GPT, 110 M
- Bert Large, 340 M
- OpenAI, GPT-2, 1.5 B
- NVIDIA, MegatronLM, 8.5 B
- OpenAI, GPT-3, 170 B


Model size
Heterogenous Memory (HM) is Promising to Address Memory Capacity Limitation

Optane-based HM

GPU-based HM
Deep Learning Training with HM Faces Challenges

- Which tensor to swap in/out of fast memory?
- When should tensor swapping happen?
Current Memory Management for DL training Has Problems

- Heavily relying on domain knowledge limits the solution generality
Current Memory Management for DL training Has Problems

- Heavily relying on domain knowledge limits the solution generality

Tensor swapping within HM

Feedforward CNN only

Suboptimal usage of fast memory

Parameter

Feature maps

Forward

Weight $w$

N

Workspace (conv)

Gradients maps

Backward

Loss Function
Current Memory Management for DL training Has Problems

• The current profiling methods limit the feasibility of the solutions
  – Static profiling

  ✔ Tensor swapping within HM
  ✔ Graph agnostic
  ✗ model-dependent or compiler-supported offline profiling
  ✗ Inaccurate profiling results

Input size and workspace are only known at runtime
Current Memory Management for DL training Has Problems

- The current profiling methods limit the effectiveness of the solutions
  - Dynamic profiling at the application level

- Tensor swapping within HM
- Graph agnostic
- Inaccurate profiling results
- Unnecessary tensor movement
Characterization of Deep Learning Training Workloads: Method

**Application level**

TensorFlow

- Track tensor lifetime
- Intercept tensor (de)allocation

**OS level**

- Track tensor mem access
- Intercept protection faults

**Diagram**

- Each memory page has only one tensor during profiling.
- Combine techniques for profiling.
Workload Characterization: Tensor Lifetime and Its Distribution

- Observation 1: There are a large number of small tensors (smaller than 4KB) with short lifetime
  - 92% of its tensors have lifetime no longer than one layer. Among them, 98% is small tensors
Workload Characterization: Memory Access Distribution

- Observation 2: The uneven distribution of hot and cold tensors in DNN provides opportunities for data management.
Workload Characterization: Page-Level False Sharing

• Observation 3. Small tensors commonly share pages with other tensors
  – The page-level profiling (not tensor-level) for data management can be misleading because of page-level false sharing
Design Choices

- Choose layers as the basic granularity for tensor management
  - Lifetime and memory access patterns of tensors are associated with layers

- Treat tensors differently based on their lifetime
Sentinel: A Software Runtime System For Tensor Management on HM

1. Dynamic Profiling
   - Coordinate between OS and TF runtime

2. Data Organization
   - Mem alloc/dealloc

3. No Migration
   - Reserve space in fast memory

4. Adaptive Migration
   - Determine migration intervals
   - Test & trial

HM
- Fast memory
- Slow memory
- reserved

DNN model
- $layer_{(1)}$
- $layer_{(2)}$
- $layer_{(3)}$
- $layer_{(4)}$
- $layer_{(5)}$

short-lived tensor
long-lived tensor
Sentinel: A Software Runtime System For Tensor Management on HM

1. Dynamic Profiling
   Coordinate between OS and TF runtime

2. Data Organization
   Mem alloc/dealloc

3. Profiling at the tensor level with high accuracy and ignorable overhead

4. Adaptive Migration
   Determine migration intervals
   Test & trial

- Sentinel model:
  - layer(1)
  - layer(2)
  - layer(3)
  - layer(4)
  - layer(5)

- Short-lived tensors:
  - coordinate between OS and TF runtime

- Long-lived tensors:
  - DNN model
  - coordinate between OS and TF runtime

- Profiling at the tensor level with high accuracy and ignorable overhead

- Adaptive Migration:
  - Determine migration intervals
  - Test & trial
Dynamic Profiling

• Profile at the tensor level and ensure each page has at most one tensor

• Profiling overhead is easily amortized over training steps
  
  Domain knowledge: exploiting workload repeatability of deep learning training

• Memory overhead is ignorable
Sentinel: A Software Runtime System For Tensor Management on HM

1. Dynamic Profiling
   Coordinate between OS and TF runtime

2. Data Organization
   Mem alloc/dealloc

3. No Migration
   Reserve space in fast memory

4. Adaptive Migration
   Determine migration intervals
   Test & trial

DNN model

- layer\(_{(1)}\)
- layer\(_{(2)}\)
- layer\(_{(3)}\)
- layer\(_{(4)}\)
- layer\(_{(5)}\)

Sentinel
Data Reorganization

- After profiling, reorganize memory allocation for tensors to avoid page-level false sharing

![Diagram showing tensor reorganization](image-url)
Sentinel: A Software Runtime System For Tensor Management on HM

1. Dynamic Profiling
   Coordinate between OS and TF runtime

2. Data Organization
   Mem alloc/dealloc

3. No Migration
   Reserve space in fast memory

4. Adaptive Migration
   Determine migration intervals
   Test & trial

DNN model

layer\(_{(1)}\)
layer\(_{(2)}\)
layer\(_{(3)}\)
layer\(_{(4)}\)
layer\(_{(5)}\)
Lazy (no) Migration for Short-lived Tensors

- Avoid unnecessary data movement
- Reuse the same memory space to save fast memory space
Sentinel: A Software Runtime System For Tensor Management on HM

1. Dynamic Profiling
   Coordinate between OS and TF runtime

2. Data Organization
   Mem alloc/dealloc

3. No Migration
   Reserve space in fast memory

4. Adaptive Migration
   Determine migration intervals
   Test & trial

- layer_{(1)}
- layer_{(2)}
- layer_{(3)}
- layer_{(4)}
- layer_{(5)}

- short-lived tensor
- long-lived tensor

PASA Lab
Proactive Migration for Long-lived Tensors

Domain knowledge: The deep learning model topology (i.e., layers) and its depth to decide the optimal migration interval and trigger data migration.
Proactive Migration for Long-lived Tensors

Trigger migration for B (from slow to fast mem)

- Tensor used in migration interval A
- Tensor used in migration interval B
- Tensor used in migration interval after B

Diagram:
- Fast Mem
- Slow Mem
- Migration Interval A (n layers)
- Migration Interval B (n layers)
Proactive Migration for Long-lived Tensors

Trigger migration for B (from fast to slow mem)

- Tensor used in migration interval A
- Tensor used in migration interval B
- Tensor used in migration interval after B

Diagram:
- Fast Mem
- Slow Mem
- Migration Interval A (n layers)
- Migration Interval B (n layers)
Proactive Migration for Long-lived Tensors

Trigger migration for after-B (from slow to fast mem)

- Tensor used in migration interval A
- Tensor used in migration interval B
- Tensor used in migration interval after B

Migration Interval A (n layers)  Migration Interval B (n layers)
Determining Migration Interval

• It cannot be too large
  – Too many data to migrate \( \rightarrow \) fast memory space constraint

• It cannot be too small
  – No time to migration \( \rightarrow \) time constraint

• Our solution:
  – Prune the search space of the migration interval based on profiling results
  – Use a few more training steps to determine the optimal one
Sentinel: A Software Runtime System For Tensor Management on HM

- Dynamic profiling with ignorable runtime overhead
- Graph agnostic
- Bridge the semantic gap between OS and DL training framework
- Minimize fast memory usage
- Avoid page-level false sharing
Evaluation

• Training models and data set:
  - (1) ResNet_v1-32 (CIFAR-10), (2) LSTM (PTB), (3) DCGAN (MNIST), (4) MobileNet (CIFAR-10), (5) BERT-large (Cola)

Evaluation platform 1

Software platform:
  - Linux v4.9
  - TensorFlow v1.14

Hardware platform:
  - 2-socket Intel Xeon Gold 6252 CPU@2.3GHz
  - DRAM: 96GB x 2; Optane: 756GB x2
  - The size of DRAM is equal to 20% of peak memory consumption of DNN models

Evaluation platform 2

Software platform:
  - HostOS: Linux v5.6.0
  - CUDA v10.1
  - TensorFlow v1.14

Hardware platform:
  - GPU: Nvidia V100 with 15.75 GB of GDDR6
  - CPU Intel(R) Xeon(R) E5-2670 with 128 GB of DDR4

The size of DRAM is equal to 20% of peak memory consumption of DNN models
Evaluation – Overall Performance

• Performance difference between Sentinel and the DRAM-only system is very small
  – No difference in DCGAN and 9% difference on average
  – We consistently outperform a state-of-the-art solution IAL (an application-agnostic solution) by 37%
Evaluation - Effectiveness of Sentinel

The fast memory size increases in a much slower rate than the peak memory consumption.

Comparison between peak memory consumption of DNN models and fast memory size for ResNet_v1 variants.

- Peak memory consumption in ResNet
- Peak memory consumption of fast memory with Sentinel
Evaluation - Overall Performance on GPU

- Sentinel consistently outperforms five state-of-the-art solutions on GPU for models with various training batch size.
Conclusion

• Leveraging limited domain knowledge, we can greatly avoid fundamental tradeoff and achieve higher performance
  – We consistently outperform five state-of-the-art solutions (on both Optane-based and GPU based training platform)

• We need innovation in profiling methods for small tensors

• We need innovation to determine when to trigger data migration
  – There is a tradeoff