GPU Memory Harvesting for Global Memory Cache in Deep Learning Clusters

Technical Report - CS 8803-SMR

Anmol Agarwal
Masters in Computer Science
Georgia Institute of Technology
`aagarwal622@gatech.edu`

Hersh Dhillon
Masters in Computer Science
Georgia Institute of Technology
`hdhillon30@gatech.edu`

Prakhar Jagwani
Masters in Computer Science
Georgia Institute of Technology
`pjagwani3@gatech.edu`

December 5, 2023

Abstract

The computational capacity of GPUs has been increasing over the past recent years, however, the memory capacity has remained almost the same. On the other hand, both the compute and memory requirements of models has been increasing steadily over over the years. Hence there is need for Memory Harvesting of GPUs to allow training large models despite memory contraints. We currenlty focus on harvesting memory on a single GPU and then can extend to multi-GPU scenario. We also demonstrate how we perform eviction, prefetching of tensors and measure the overhead of Ravenstore when training models.

Keywords Memory Harvesting · Caching Policies · Prefetching Policies

1 Introduction

1.1 Motivation

Our Memory Harvesting technique is meant to help large models train on GPUs with limited capacity. We leverage the following key insights in order to help define our design goals:

- **Limited Memory Growth**: <TODO: Paste figure as on slides> As shown in figure <REF>, for state of art GPUs, there is drastic increase in TeraFLOPS across different GPUs, but memory increase is negligible. This implies that we might be able to meet compute requirements of the models better than their memory requirements. Hence, there is a need for a way to address the issue of limited memory without hampering model's training and performance.
- **Highly Skewed Memory Utilization**: <TODO: Paste figure as on report> Even though workloads suffer from memory bottlenecks, several studies have shown that, in production clusters, memory utilization is suboptimal. For example, as shown in figure <REF>, in Alibaba's production cluster, only 20% of GPUs have applications utilizing over 80% of their memory. There is need to harvest underutilized GPUs to support applications requiring higher amount of GPU memory than is available.
- **Naive Eviction/Prefetching Policies**: In other to swap tensors out to GPU memory to give space for other tensors needed by application, policies like LRU can be used to decide what tensors to evict. However, these policies are naive because they don't take into account the following factors:
  - **Heterogeneous Size of Tensors**: LRU policy might only evict tensor which was least recently used. However, if the size of this tensor was in KBs, it could have done better by evicting a larger tensor (whose size is in MBs/GBs) instead. Similarly, LRU might swap out a very large tensor (running into GBs) to make space a small tensor whose size maybe in bytes.
  - **Latency Awareness**: The pattern of usage of tensors is quite predictive when training models, and the pattern is repetitive. For example, when training a deep neural network, we can offload activations

generated at each layer to CPU because they are not going to be required before backward propagation phase. Now, since we know that activations which were generated most recently (i.e. from last layer), would be needed first. A smart prefetching policy will know this pattern, and decide to prefetch activations starting from the last layer.

1.2 Design Decisions

With above issues in mind, we propose the following design decisions to mitigate these issues and maximize the utility of the system. For the sake of interpretability, we will refer to tensors not managed by Ravenstore as those lying in Userspace and those managed by Ravenstore as those lying in Ravenspace. Applications can only use tensors present in user space.

- **APIs**: Clients/Applications are provided with the following four APIs to help manage tensors for memory harvesting purposes:
  - **Create**: Whenever clients want new tensors, they can call create with specified size and Ravenstore will create space for those tensors.
  - **Get**: Whenever clients need to use tensors, they can call get API specified with tensor id, so that those tensors move to Userspace (if they were present in Ravenspace) and clients can use them.
  - **Put**: When clients don't need tensors, but might need them later (for example, activation tensors might be needed later during backward propagation phase), they can call put API to allow moving tensors back to Ravenspace and Ravenstore can do whatever it wants to ensure maximum GPU utilization and low latency.
  - **Free**: When clients don't need tensors at all, they can call free API to delete the tensor.
- **Abstraction**: Whatever Ravenstore with Memory Harvesting does underneath with the tensors created by applications is completely unknown to them. Neither the applications should care about that. All they need is the following:
  - Tensors must be available to them when they call get() API.
  - All other APIs must work as expected without interfering with clients' normal program run.
- **Low Latency in Get API**: When clients call get() when they need tensors for further computations on their GPU, Memory Harvesting implementation does the following:
  - Check if tensor is already present on GPU device where the client is running.
  - If tensor is not already present, then Ravenstore will fetch the tensor from CPU memory to GPU device.
  - And the tensor would be moved to Userspace and handed over to the user.

  The second step, i.e., when tensor is not already present on GPU, incurs high latency when moving the tensor to GPU. This will hugely increase training/inference time of models on the clients. In other to avoid that, we have prefetching mechanisms. Clients will need to specify the time they will need the tensors (through Put API) again in future, and Memory Harvesting framework will fetch those tensors beforehand if possible to avoid delays in Get() API call.

2 Related Work

3 High Level Design

Ravenstore consists of three components, Client, Daemon and Interceptor. Client makes API calls, these calls go through Shared IPC and Daemon (or Server) acts according to the calls like allocating tensors, bringing them to GPU, deleting them, performing eviction, etc. Ravenstore manages two ways:

- **Interception**: Ravenstore intercepts CUDAMalloc and CUDAFree calls from clients and allocates/deallocates tensors. Here Ravenstore Interceptor comes into play. But tensors are not managed by ravenstore for memory harvesting purposes.
- **Client APIs**: Here clients make API calls and memory harvesting is done based on calls received by the Daemon.

For more in-depth overview of architecture of Ravenstore, please refer Ravenstore Final Report document <REF>. Now we discuss high level design of memory harvesting technique implemented alongside Ravenstore.

3.1 Memory Harvesting Design Overview

4 Low Level Design

5 Implementation

6 Evaluation and Benchmarks

7 Future Work

7.1 Multi GPU Design

8 Conclusion

9 Headings: first level

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

See Section 9.

9.1 Headings: second level

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

$$\xi_{ij}(t) = P(x_t = i, x_{t+1} = j | y, v, w; \theta) = \frac{\alpha_i(t) a_{ij}^{w_t} \beta_j(t+1) b_j^{v_{t+1}}(y_{t+1})}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i(t) a_{ij}^{w_t} \beta_j(t+1) b_j^{v_{t+1}}(y_{t+1})} \tag{1}$$

9.1.1 Headings: third level

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Paragraph Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

10 Examples of citations, figures, tables, references

10.1 Citations

Citations use `natbib`. The documentation may be found at

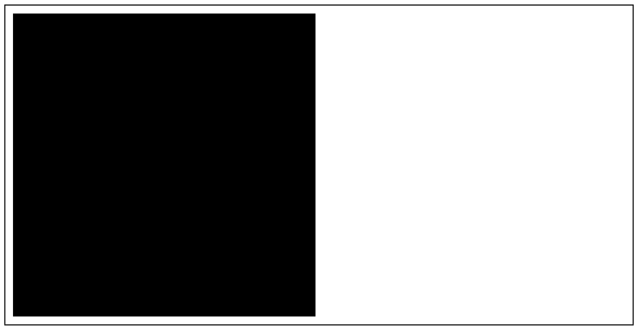

Figure 1: Sample figure caption.

Table 1: Sample table title

Part		
Name	Description	Size (μm)
Dendrite	Input terminal	$\sim$100
Axon	Output terminal	$\sim$10
Soma	Cell body	up to 10^6

`http://mirrors.ctan.org/macros/latex/contrib/natbib/natnotes.pdf`

Here is an example usage of the two main commands (`citet` and `citep`): Some people thought a thing [**??**] but other people thought something else [**?**]. Many people have speculated that if we knew exactly why **?** thought this...

10.2 Figures

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

See Figure 1. Here is how you add footnotes. [1] Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

10.3 Tables

See awesome Table 1.

The documentation for `booktabs` ('Publication quality tables in LaTeX') is available from:

`https://www.ctan.org/pkg/booktabs`

10.4 Lists

- Lorem ipsum dolor sit amet
- consectetur adipiscing elit.
- Aliquam dignissim blandit est, in dictum tortor gravida eget. In ac rutrum magna.

References

[1] Sample of the first footnote.