Parallelizing a DFS-based hyperparameter search algorithm using Shared Address Space and Message Passing models

Joshua Hong (jjhong), Kit Ao (kitao)

Based on the work of Wu *et al*

Webpage URL: https://joshua-j-hong.github.io/15618-Parallel-DFS-Hyperparam-Search/

1. Summary

We are going to parallelize the DFS-based hyperparameter search algorithm of the Mirage tensor optimizer using a Shared Address Space model (via OpenMP) and a Message Passing model (via MPI). We are going to compare these two methods (in addition to static assignment via pthread-based multithreading) in terms of workload balancing, resource contention, as well as overall runtime improvements.

2. Background

Effective search for GPU kernel parameters enables the generation of fast GPU kernels that significantly improve the performance of tensor programs. Mirage is a tensor optimizer that performs an exhaustive search over the kernel, thread block, and thread levels in order to find configurations (known as μGraphs) that are equivalent to the input tensor program. The search process can be viewed as a DFS. Pruning is performed by invoking a probabilistic equivalence verification process that verifies whether a given μGraphs is equivalent to or a subset of the input program. Overall, Mirage divides the input tensor program into LAX subprograms, consisting of multi-linear operations (matrix multiplication, convolution), division, as well as exponentiation. It then performs the search on each of these LAX subprograms. The entire optimization process is bounded by the search process. Therefore, a good parallelism strategy is needed in order to improve the optimization process. Currently, a multithreaded approach based on interleaved static workload assignment is used as the parallelization strategy. However, as we will show, this current approach suffers from workload imbalance as well as resource contention.

3. Challenges

There are several challenges to implementing parallelism on the Mirage search process.

1. The DFS search algorithm is not an intuitive algorithm to parallelize. This is because in a DFS, consecutive nodes are explored sequentially. This is unlike a BFS where the nodes of the same level of the search are independent of each other and can be processed in parallel.
2. During the search process, if a node's configuration fails a verification step, the node gets pruned. Due to this, there is significant divergent execution. The search times of nodes at the same level can differ significantly since some nodes could be pruned very early on. This means that a static assignment of work based on partitioning the higher-level nodes would result in workload imbalance.
3. The search time is bound by a compute-intensive verification step that invokes the z3 solver. The black-box nature of the z3 solver makes it very difficult to optimize for the Mirage search process. Additionally, contention on the z3 solver might explain the slowdown in speedup seen at higher thread counts.

4. Resources

We plan on developing our Shared Address Space and Message Passing approaches on locally machines as well as the GHC machines that have GPUs. Additionally, we hope to have access to machines with more resources, such as the PSC machines. The code base we are starting from is the public Mirage repository, available here Mirage. Additionally, we reference the documentation page, Mirage Documentation for more information as well as the original paper.

Wu, M., Cheng, X., Padon, O., & Jia, Z. (2024). A Multi-Level Superoptimizer for Tensor Programs. arXiv preprint arXiv:2405.05751.

As mentioned before, we will utilize the OpenMP framework for the Shared Address Space implementation, and the MPI framework for the Message Passing implementation.

5. Goals and Deliverables

5.1 Plan to Achieve

1. Implement the Shared Address Space (OpenMP) and Message Passing (MPI) versions of the DFS search algorithm.
2. Benchmark the performance of the two versions in terms of speedup performance and workload distribution.
3. **Poster Session:** we will show graphs that demonstrate that our approaches provide significant speedup and better workload balancing when compared to the existing multithreaded implementation.

5.2 Hope to achieve

1. Investigate how to reduce contention on the z3 solver. Possibly ideas are caching to reduce unnecessary calls to the z3 solver, reducing calls to the z3 solver by limiting the number of verifications.
2. Implement a BFS-based search process to see if that maps better to a parallel implementation.

6. Platform Choice

As mentioned previously, we plan on developing our Shared Address Space and Message Passing approaches on locally machines as well as the GHC machines that have GPUs. We choose to use these machines for development as a GPU is required to use Mirage. We also hope to benchmark with higher thread counts to fully capture the scaling properties of our approaches, which may require the use of machines with more allocated resources. Our development will be done in C++, as the original codebase we are adding on to and the parallelization interfaces we plan on using are both in C++.

7. Tentative Schedule

Week	Goals
November 18 - November 24	Shared Address Space approach
November 25 - December 1	Message Passing approach
December 2 - December 8	Additional optimizations and approaches involving the z3 solver and different search algorithms
December 9 - December 15	Final Report