Milestone Report

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

Summary of Current Progress:
We have completed the implementation for the shared address space model via OpenMP. We performed benchmarking for the shared address space model on 1, 2, 4, 8, 16, 32, 64, and 128 threads in terms of execution time speedup compared to the original static assignment approach, as well as workload balancing. We discovered that dynamic task assignment via OpenMP improved workload balancing across threads and thereby improved speedup by 1.1x - 1.5x across the 4 examples that were tested when compared to the original static assignment approach using the same number of threads. At higher thread counts, the speedup is no longer linear due to resource contention in the Z3 solver. Using Nvidia Nsight, we found that at higher thread counts, the execution time is dominated by calls to pthread_mutex_lock(), which takes up 94% of the total execution time with 64 threads and 97.9% with 128 threads.

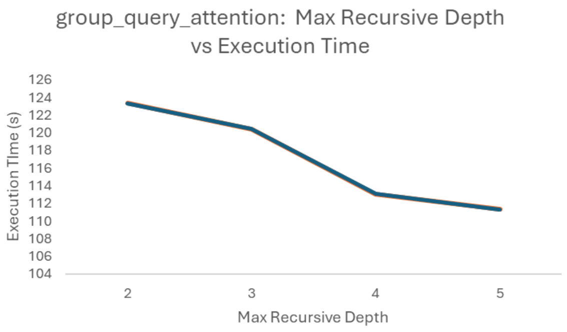
A message passing model has the potential to significantly reduce the contention for the Z3 solver at higher thread counts since each process has its own copy of the Z3 solver and therefore does not need to contend for resources on the same Z3 solver. We currently have also implemented a message-passing parallelization strategy via MPI, however, our current approach suffers from several issues. First, since the device memory manager pre-allocates device memory for μGraph fingerprinting, each process requires significant GPU device memory. On the catalyst-cluster with 4 A5000 GPUs on each node, each process requires its own A5000 GPU. Additionally, we are having issues with inter-node communication that prevents us from scaling to more than 1 node. Therefore, we have only tested the message passing model on 1, 2, and 4 processes. We suspect that this is an issue with the catalyst-cluster, however, we have not been able to utilize the GPU partition on PSC.

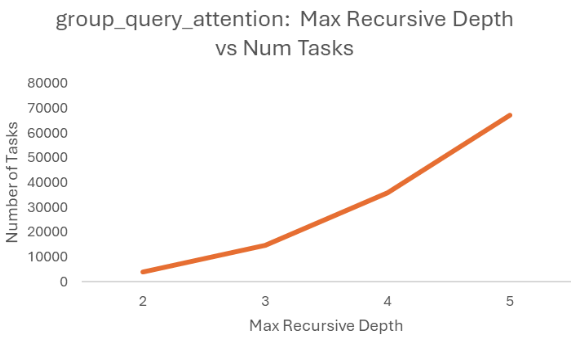
Goals and Deliverables:
Our plan is to implement the shared address space and the message passing versions of the Mirage DFS search algorithm and benchmark the performance of the two versions in terms of performance speedup and workload distribution. We have implemented both versions and also completed the benchmarking for the shared address space version. However, due to aforementioned issues we are still working to complete the benchmarking on the message passion version. We plan to complete our benchmarking of the message passing version for our final presentation. Therefore, we aim to achieve all our deliverables.

One of our “nice-to-have” goals was to investigate how to reduce contention on the Z3 solver. Since the message passing model theoretically would significantly reduce Z3 solver contention by having a separate Z3 solver for each process, by implementing the message passing model we hope to achieve this. Our other “nice-to-have” goal was to implement a BFS-based search process. However, after further discussion, we realized that a BFS-based implementation would alter the search algorithm significantly without offering much benefit in parallelization or workload balancing. Therefore, we will not be pursuing this goal.

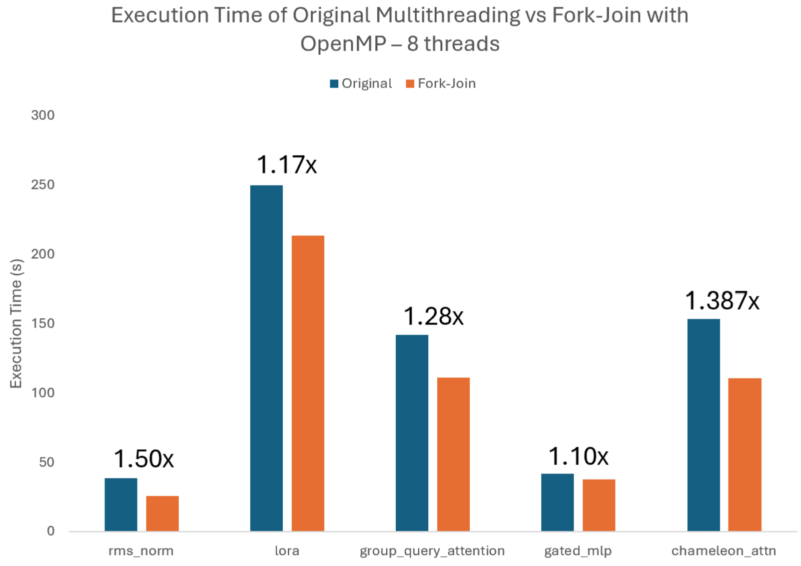
Poster Session Plans:
As mentioned in the proposal, we plan to show graphs that demonstrates the performance speedup that our approaches are able to achieve, as well as graphs that demonstrate improved workload balancing with these approaches.

Preliminary Results:
Shared Address Space (OpenMP):

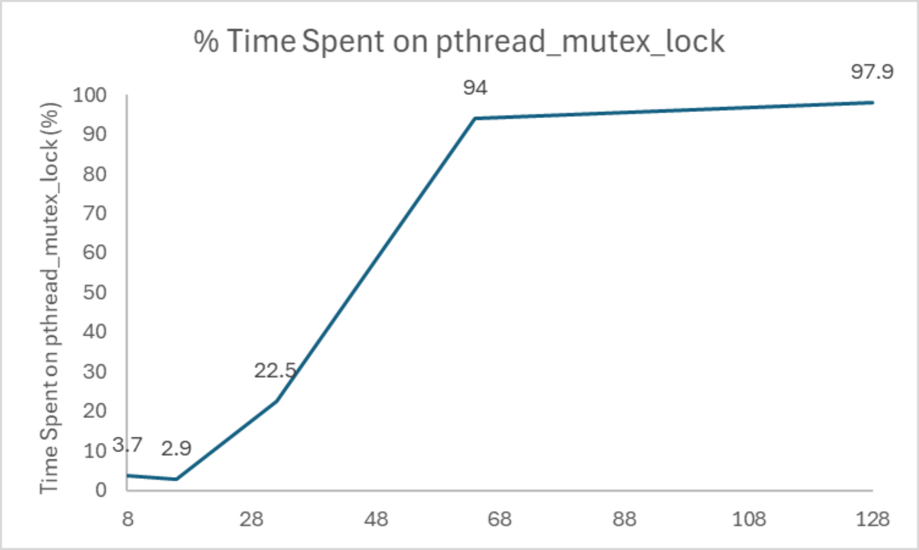




A tunable parameter in the shared address space model is the max recursive depth. Given that we choose to perform dynamic scheduling during the recursive search, our strategy involves launching a separate task at each recursive call up until a certain maximum recursive depth. Beyond this recursive depth, the program executes serially. The above figures depict the search performance at different maximum recursive depths, for the Group Query Attention example. The left graph depicts the execution time at different max recursive depth, while the right graph depicts the number of tasks generated during the recursive search. As can be seen, as the number of maximum recursive depth increases, the number of tasks generated to be distributed across different threads increases, and the execution time decreases.

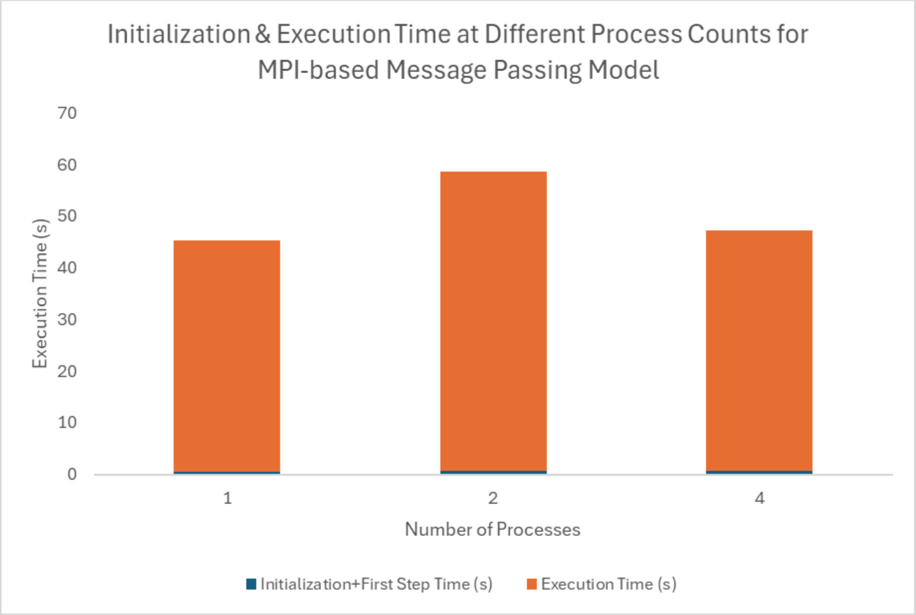


This graph depicts the speedup the OpenMP-based shared address space model was able to achieve over the original approach with pthread-based static assignment. As can be seen, across different examples, the OpenMP-based approach was able to achieve a speedup of between 1.10x - 1.50x over the original approach.



This graph depicts the time spent on pthread_mutex_lock() inside the Z3 solver at different thread counts. As can be seen, at higher thread counts, the execution time is dominated by calls to pthread_mutex_lock(), indicating significant resource contention within the Z3 solver.

Message Passing Model



This graph depicts the initialization and execution time at different process counts for the MPI-based message passing model of the search, measured on the catalyst-cluster. With the current implementation, it doesn’t appear that increasing the number of processes improves the execution time.

Issues:

The most pressing concern is with scaling the MPI-based message passing approach to a higher number of processes, as well as to verify why exactly increasing the number of processes does not improve performance. We suspect that this might be an issue with MPI’s compatibility with the catalyst-cluster. Therefore, we hope to test our message passing approach on PSC instead. However, we are unable to access the GPU partition on PSC, potentially due to permission issues. After this is resolved, we hopefully will be able to scale to higher process counts with the MPI-based approach.