1 Progress Summary

We have completed the initial set up and testing framework for this project. We have implemented parallel versions of a global mesh operation, and we are actively working on debugging and improving them. A more detailed description of what has been done can be found in the Goals and Deliverables Update section below.

2 Goals and Deliverables Update

- **Set up.** Plan to Achieve: $\surd$
  We have successfully integrated both OpenMP and OpenMPI into the code base. This involved installing the correct compiler on our personal machines and modifying the compile process, which is done with NodeJS.

- **Implementation.** Plan to Achieve: $\surd^{\wedge}$
  We have implemented parallel versions of triangulate with OpenMP and OpenMPI. Figure 1 are the speed up tables for both methods where a NaN indicate an error with the current implementation.

```
-- OMP Speedup Table ---
 benchmark       | basic.square    | basic.poly_mesh | basic.quad_cube | edge.tri_cube   | mesh.bunny
-------------------------------------------------------------------------------------------------------
 1               | 1.0             | 1.0             | 1.0             | 1.0             | 1.0
 2               | 1.45908184      | 0.91159136      | 0.95402299      | 0.93172691      | 1.44109489
 4               | 1.40576923      | 0.98933902      | 0.97647059      | 0.94887526      | 2.2031525
 8               | 1.08779762      | 0.89748549      | 0.95402299      | 0.75324675      | 3.76396879
-- MPI Speedup Table ---
 benchmark       | basic.square    | basic.poly_mesh | basic.quad_cube | edge.tri_cube   | mesh.bunny
-------------------------------------------------------------------------------------------------------
 1               | 1.0             | 1.0             | 1.0             | 1.0             | 1.0
 2               | nan             | nan             | nan             | 0.77229801      | nan
 4               | nan             | nan             | nan             | 0.33833247      | nan
 8               | nan             | nan             | nan             | 0.17457823      | nan
```

Figure 1: current speed up table

As can been seen, there are a lot of issues with the MPI version ¯_(ツ)_/¯. This is because the mesh is implemented with list as the underlying data structure. In this set up, each mesh element (halfedge, edge, vertex, face) refer to other elements with an iterator and is identified by a global id. The mesh also has its own memory management system for emplacing and erasing elements. This makes it very cumbersome to work with. Since the primary goal of this project is to parallelize the mesh operations, our plan is to convert the mesh to a data structure that lends itself better to parallelism, implement the algorithm with that data structure and then convert back to the convert data structure. We have started to implement this new data structure, but we are still in the process of debugging it.

- **Gather Benchmarks.** Plan to Achieve: $\surd$
  We have identified 2 large benchmarks and several smaller benchmarks for testing. We have also implemented a testing script that can automatically produce speed up tables like the ones shown in Figure 1 as well as bar chart visualization for easier comparisons.

- **Testing - Load Balancing and Varying Processor Counts.** Plan to Achieve: $\surd$
  Our current OpenMP implementation for triangulate function supports load balancing testing. We are in the process of generating results for comparisons between different scheduling schemes for finding the optimal strategy for parallelization. This identically applies to varying processor counts for analysis purposes. We now need to debug OpenMPI implementation to fully support these testing plans as well.

3 Poster Session

Our poster session will focus more on the graphical visualization of parallelization improvements we were able to achieve through this project. Although our investigation is primarily on a rendering software, gauging speed improvements on the spot is expected to be difficult for most people to intuitively grasp unless they just believe whatever time measures we put up on the screen. Therefore, we believe it will be more beneficial to have graphs representing our various analysis results to facilitate the understanding of our strategies and improvements. We may still include a live demo if we manage to find a particularly effective parallelization scheme for a scene such that the performance improvement is noticeably drastic.

4 Preliminary Results

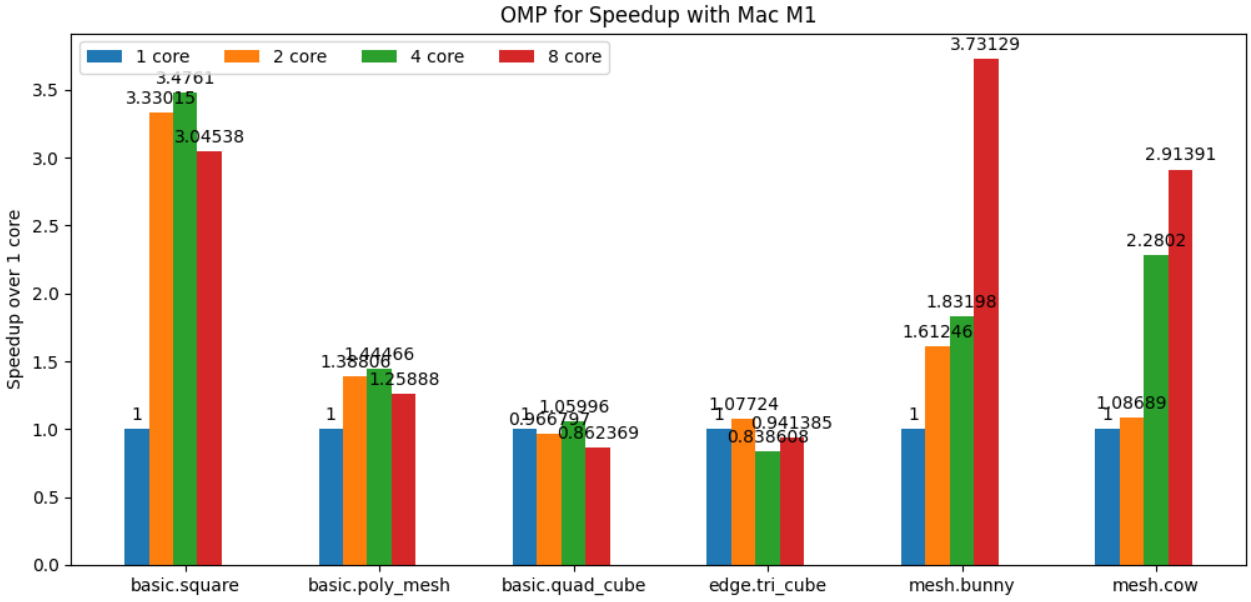

Figure 2: for

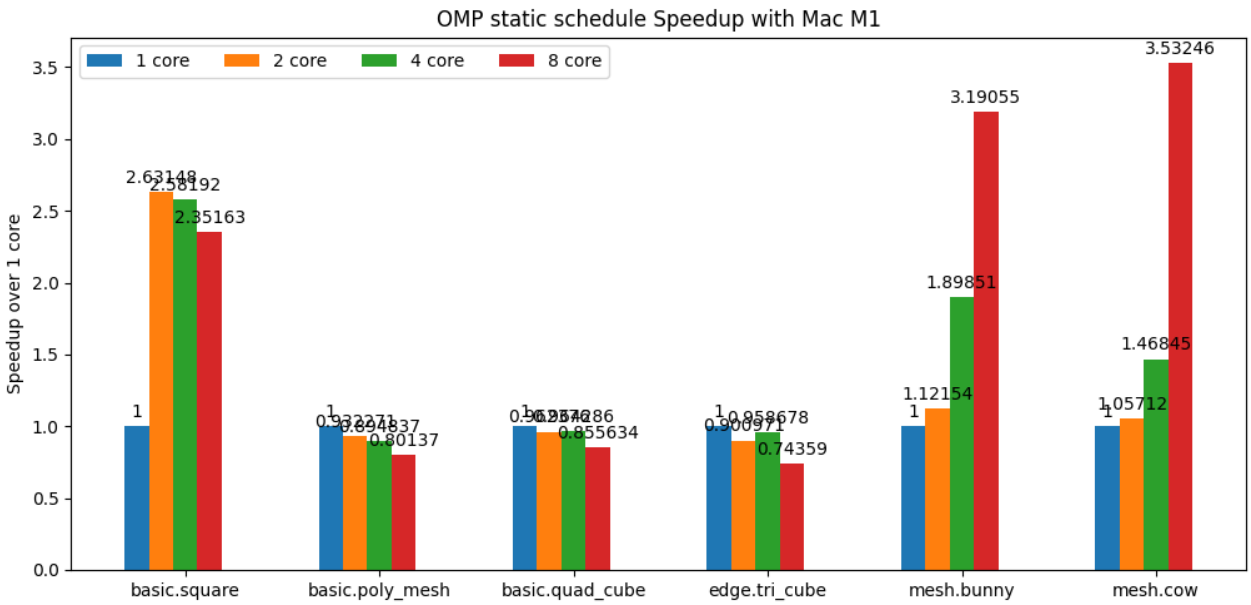

Figure 3: static schedule

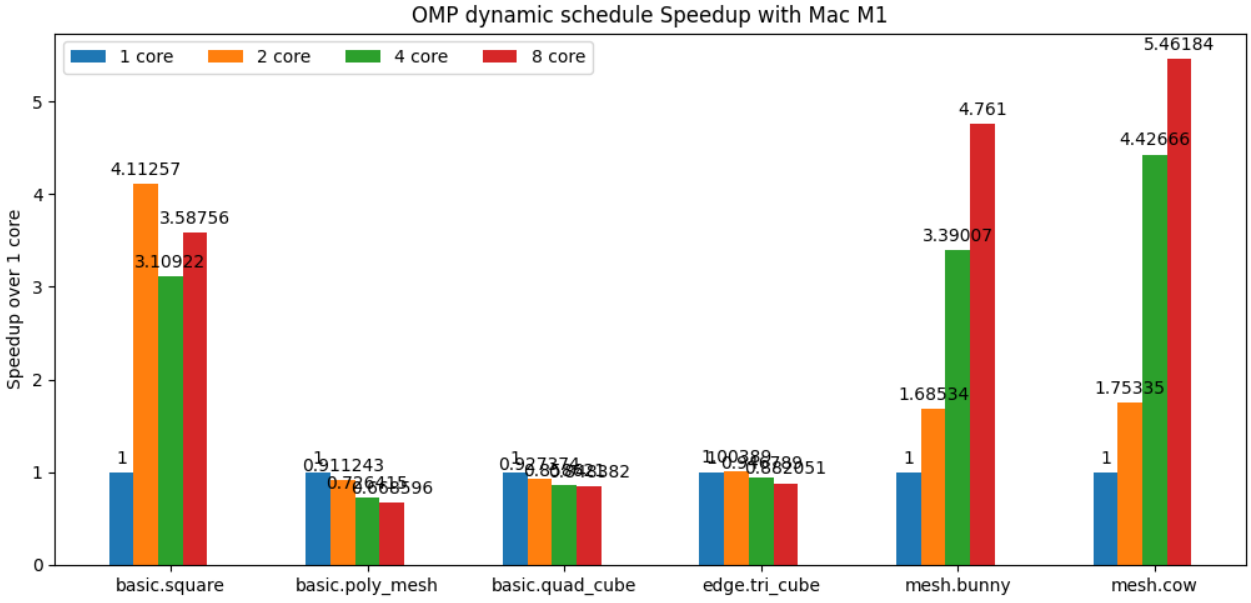

Figure 4: dynamic schedule

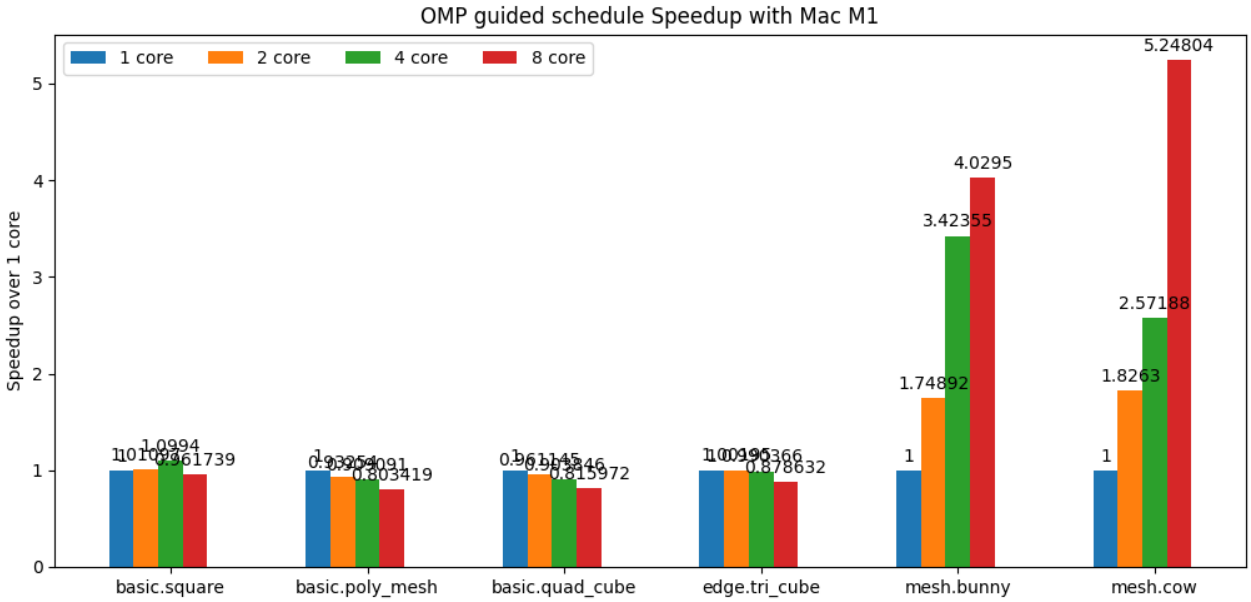

Figure 5: guided schedule

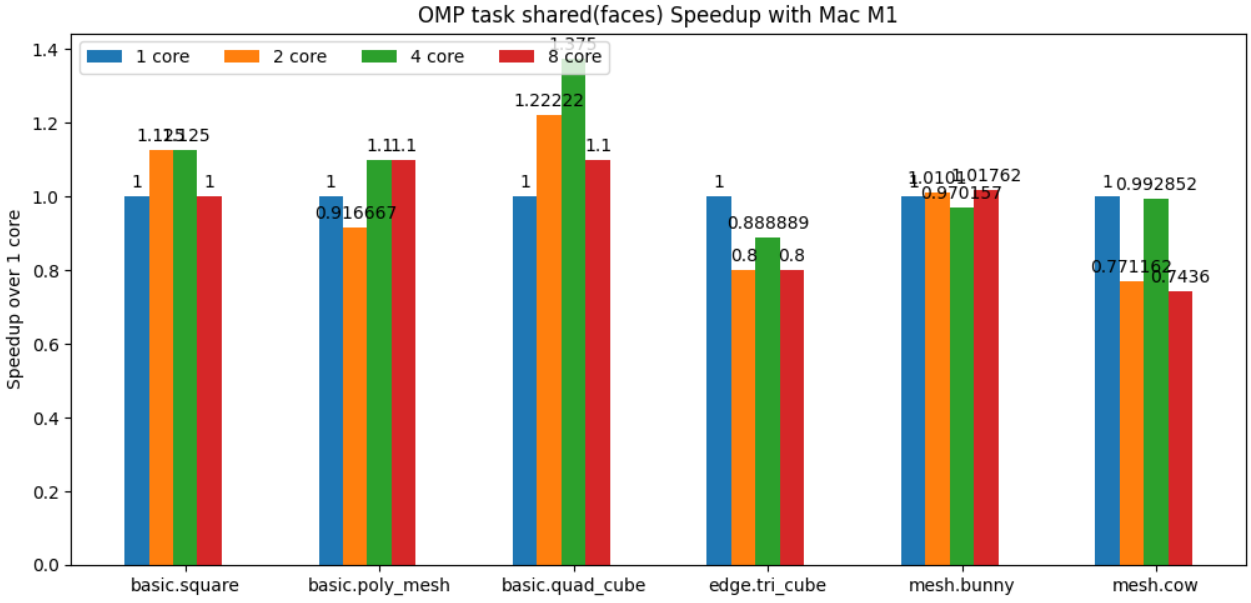

Figure 6: task shared (faces)

5 Areas of Remaining Concern

Our main source of remaining concern is fixing the OpenMPI implementation to properly work. As mentioned earlier, OpenMPI implementation is expected to require a lot more substantial code base changes regarding underlying data types and primitives in Scotty3D. This may implicitly affect performance of sequential reference implementation or the OpenMP version as well, invalidating previous results we produced already. It is also unclear how we may be able to determine if a working implementation we come up with is indeed an optimized solution or not, making it harder

to be confident in the conclusive comparison we may reach as to which approach is better for these tasks.