Midway Report:
OpenMP Debugger

Jack Wei Luca Borletti

1 URL

The url for our project is: https://github.com/JackWei33/OpenMP-Debugger

2 Work Completed

We have done extensive work and experimenting with OMPT — OpenMP's own API for tool construction. After reading several papers to understand how OMPT works and the different features it implements, we created a logging tool that uses callbacks to log events. When linking our tool to the runtime, running an OpenMP program will output several log files describing events like task creation, thread creation, parallel regions starting and ending, mutex acquistion, etc.

Our current work is to use these logs to output three graphs.

1. Graph of nodes and edges representing the task structure of the program. Every OpenMP program will create implicit tasks and users can create explicit tasks. These tasks define the workflow of the program. Our graph will also include synchronization primitives that affect how the different tasks interact with one another.
2. Bar graph of threads and tasks. This graph will output what threads worked on which tasks and for how long. We will also implement custom callbacks that allow a user to track the runtime length of their own regions. In combination with the first graph, a user will be able to see which tasks are eating up their runtime and what code regions that task corresponds to.
3. Bar graph of threads and synchronization. This graph will output how long each thread spends working vs idling in synchronization regions. This will allow users to track how synchronization and work distribution is affecting their runtime.

These graphs are in progress and we expect to finish implementing them within the next two to three days. In relation to our goals and deliverables, we are only slightly behind schedule. The three graphs described above are roughly the deliverables we intended to have completed by the midway report. The delay in schedule is largely due to OMPT being more difficult to understand than we initially anticipated. We still believe we will be able to hit all our deliverables and implement the two remaining features of race condition and deadlock detection. The "nice to haves" of the project proposal were a debugger like gdb, a memory leak detector, and a performance optimization suggestor. These seem largely out of reach as we will likely only have time to complete our initial deliverables and goals.

3 Poster Session

We plan to have information on our poster describing how we implemented all the different features of our debugger. Then, we will also have a demo demonstrating the debugger's capabilities on our own Project 3. If possible, we may also try to debug some parallel problems and solutions we find online to further showcase our debugger.

4 Preliminary Results

We currently do not have strong results, but will soon have the core features of our project implemented in the three graphs mentioned above.

5 Concerns

We have no concerns in implementing the three graphs mentioned above as we have already fleshed out the algorithm and just need to write the code. However, we are still not fully sure how to implement race condition checking. This will likely require tracking data movement and might be outside the scope of our project. We will look further into implementing this feature and then assess if we want to continue or not. Implementing the deadlock feature is much more feasible and we have a strong understanding of how it will be done.

6 Schedule

1. **December 3 - December 6**
   - Luca: Complete algorithm for parsing logs to extract necessary information for the 3 graphs
   - Jack: Build UI for graphs using Plotly
2. **December 6 - December 9**
   - Luca: Research and implement race condition detection
   - Jack: Research and implement deadlock detection
3. **December 9 - December 12**
   - Luca: Design and build our poster
   - Jack: Use our debugger on example programs and our Assignment 3
4. **December 12 - December 15**
   - Jack and Luca: Finish up any remaining work and write the final report