Milestone Report - 15618 Spring 2025
Meghna Jain (meghnaj) and Raveena Gupta (raveenag)

TITLE: Parallel Static Time Analysis

INITIAL SCHEDULE: The following is the initial schedule we had planned to follow for our project.

Week	Dates	Tasks
0	3/24 - 3/30	● Plan the outline of the project. ● Set up repository and gather data
1	3/31 - 4/6	● Get sequential code running ● Benchmark the sequential algorithm
2	4/7 - 4/13	● Implement parallel static time analysis using OpenMP - part 1 from paper ● Run experiments to analyse performance ● Complete Milestone report
3	4/14 - 4/20	● Implement the timing updates using task graphs - overcoming synchronization challenges posed by OpenMP - part 2 from paper
4	4/21 - 4/27	● Stretch goal: Implement the CPU-parallel partitioning algorithm - algorithm from paper ● Realistic goal: Complete the task graph implementation
5	4/28 - 5/2	● Final Poster Session

We have stayed on track with the progress of our project as per the initial schedule. We faced roadblocks in getting the sequential code to work because we could not find open source libraries with sequential implementations of the algorithm. Thus, we had to write the sequential code from scratch and validate it, which took longer than expected.

Currently, we are in the process of adding task loop-based parallelism using OpenMP (part 1 from the paper). As for performance, we are not yet seeing a speedup. While this is consistent with the findings of the research paper, which states that the synchronization costs of the task loop-based parallelism are high, we suspect two additional 2 reasons for the lack of speedup in our implementation:

1. We need to optimize the way we are creating tasks and synchronizing between them, and reduce the overhead of doing so.
2. Our input circuits are very small, which means that there aren't enough independent tasks for all the threads, leading to low thread utilization and high overhead. Our plan for this is to generate more circuits using a synthesizer and confirm this.

NEW SCHEDULE:
Based on our current progress, we have prepared a more detailed schedule for the project.

Week	Dates	Tasks
3.2	4/16 - 4/20	● Raveena: create more test circuits ● Meghna: optimize OpenMP implementation, and complete performance analysis
4.1	4/21 - 4/23	● Raveena: Create the task graph ● Meghna: Work on task graph dependencies
4.2	4/24 - 4/27	● Raveena: Parallelize forward pass using the task graph ● Meghna: Parallelize backward pass using the task graph
5.1	4/28 - 4/30	● Raveena: Optimize the load balancing of the task graph ● Meghna: Optimize the synchronization of the task graph, find similar path dependencies and bundle them together
5.2	4/30 - 5/2	● Performance analysis, factor analysis ● Write final report

COMPLETED: The following paragraphs explain what we have completed for Static Timing Analysis so far.

So far, the project has implemented a sequential and a naive parallel algorithm for our static timing analysis tool. Our team has processed the dataset, which is a netlist that represents the connections between circuit components, their delay, and the data path of signals. We have parsed the JSON data and extracted the details to observe the individual paths in our circuitry. Our first step was creating a directed adjacency graph (DAG) to represent connections between gates. For instance, if Gate 1's signal went into Gate 2 and Gate 3, we denote a 1 in the columns place of our adjacency matrix. Then we use topological ordering to ensure that independent nodes are processed first within a queue, which removes any dependencies in our path. Using our topological ordering, we traverse through all the paths and accumulate the delay. This is our "arrival delay". Following the forward pass, we do a backward pass to identify the required time for the signal, thus finding potential timing issues and calculating the slack.

To enhance efficiency and time, we implemented OpenMP parallelization. To do this, we used our topological sort to find paths that have their dependencies finished. (These are nodes with no in-degrees.) Each node with no dependencies is a task taken by a thread which calculates the delay for that specific path. We have thus enabled concurrent execution of the forward pass.

DEMO DELIVERABLES:

We will be able to reach our goals as we have a robust sequential code and we're working on parallelizing our static timing tool. Our "nice to haves" is having our last algorithm - Task Graph Partition, which is a graph partitioning scheme by making smaller clusters of graphs in adjacent levels. This may still remain a stretch goal, however, we should be able to calculate speedup and have task graphs parallelism done by our final due date.

We will create graphs to visualize the performance of all 3 algorithms. We will include a factor analysis which will explain how each of the algorithms responds to different inputs (large vs small circuit, balanced vs unbalanced circuit, etc.).

CHALLENGES:

Our biggest unknown is that our parallel code is taking longer than our sequential code, this needs to be observed more as it can be due to the overhead/critical sections/small problem size.