Revised Schedule

Week	Planned Work	Assignee	Status
Apr 1 - Apr 7	Understanding the cache simulator, begin work on extending cache simulator as needed for multiple cores of a specific cache hierarchy	Joshua	Done
	Survey the different scheduling algorithms from research papers and select a few promising ones.	Rae	Done
Apr 8 - Apr 14	Finish up initial implementation of cache simulator, ready for testing. Begin work on front end of the program evaluation framework	Joshua	Done
	Draft scheduling algorithms that takes in tasks with data sharing annotated / extract data sharing information from memory trace	Rae	Done
By Apr 19	Finish up the front end of the evaluator. Stamp out bugs in the evaluation platform found in testing.	Joshua	WIP
	Run the selected algorithms to schedule tasks that represent standard programs based on theoretical understanding of the data usage.	Rae	WIP
By Apr 22	Carry out trace-driven testing of the cache simulator, to check the correctness and performance of the implementation (are the results what we expect from the system?)	Joshua	Todo
	Run the selected algorithms to schedule tasks based on profiling collected.	Rae	Todo
By Apr 26	Begin work on implementation of the actual scheduling algorithm in parallel code for real world testing, making improvements to cache simulation platform as necessary.	Joshua	Todo
	Evaluate selected algorithms based on performance we obtain from testing. Refine testing procedure and algorithms where necessary.	Rae	Todo
By Apr 29	Run the scheduling algorithm on parallel code on local multi-core machines, as well as the PSC machines if time permits. Collect data for analysis.	Both	Todo
By May 3	Prepare deliverables: poster, demo and report	Both	Todo
May 5	Presentation Day	Both	Todo

Summary of Progress

We have made good progress in the understanding and modeling of the problem. On a high level, we model each program task T_i as a job with an expected computation time C_i and a set of input data $D_i = \{ d_{i\ j} \}$. Our goal has two axes of optimization, first to minimize the maximum computation time on any device, and second to minimize the data movement. We make some idealization in this model and our scheduling algorithms. One assumption is that input data dominates data movement which is valid for many common tasks such as matrix multiplication. Another assumption is that each unit of data $d_{i\ j}$ has the same size and is easy to recognize from the profiling of programs. While this may not be true for all programs, it holds for data parallelism.

We have selected three candidate algorithms to evaluate, each having their own priorities of the two axes of optimization, and have drafted them for testing. We will first proceed with standardized tasks such as matrix multiplication and its variants. The reason why matmul is chosen is because it offers a neat breakdown of sub tasks (blocked matmul) that have data sharing patterns. We aim to do some preliminary testing on this before moving on to more dynamic programs.

The three selected algorithms came from the paper “Memory-Aware Scheduling of Tasks Sharing Data on Multiple GPUs with Dynamic Runtime Systems” (M. Gonthier, L. Marchal and S. Thibault).

On the evaluation platform side, we’ve made good progress on implementing a cache coherence simulator based on the Intel i7-9700 CPU present on the GHC machines. This includes matching the number of cores and sizes of individual caches. Exact details on the interconnect and cache coherence protocol of the exact intel CPU is hard to determine, but as far as we can tell, the CPU utilizes a ring interconnect with a MESIF (Modified, Exclusive, Shared, Invalid, Forward) cache coherence protocol.

So far, we have implemented a seemingly working version of the multicore cache coherence simulator based on the parameters above as well as a MESI cache coherence protocol. Further work still needs to be done in tracing the behavior and testing the performance of this cache coherence protocol, in order to ensure that the simulator is working as we expect.

New list of Goals

- Test scheduling algorithm on standardized tasks such as matrix multiplication before moving on to dynamic tasks.
    - Added for a theoretical fallback and to make sure things are working.
- Decompose benchmark programs into parallelizable tasks. Identify a fair suite of programs and annotate the task units and synchronization points.
- Profile computation time and memory access at rescheduling
- Iteratively access our algorithm and improve on profiling.
- Analyze effectiveness of estimating the memory access time using our own cache simulator.
- Evaluate the total run time (computation time + memory access time) of our program with and without dynamic scheduling.
- Finish up the front-end of the simulator, in order to take in arbitrary traces.
- Fully trace behavior and characterize performance of cache simulator (with the help of memory traces), in order to ensure the correctness of our simulator
- Build up system to translate scheduling algorithm and multithreaded programs into memory traces for the cache simulator
- (nice to have) Implementation of the full MESIF cache coherence protocol in the cache simulator
    - This is a lower priority since we should have a working MESI multi-core cache simulator by the end of the project, which should be sufficient for testing our scheduling algorithm.
- (nice to have) Run the actual program on local multicore and PSC machines to evaluate how the scheduling algorithm performs outside of our simulated environment.
    - This might not be achievable in time depending on how difficult it is to integrate the scheduler and get it to allocate tasks dynamically.

Presentation

We will first have a graphic decomposition of the program tasks and data sharing, while running through a trace of the scheduler. The diagrams will be shown on the poster, with possibly some animation to further understanding of our project.

We will report on the different scheduling algorithms considered, as well as the performance difference in terms of memory activity as well as total runtime.

Depending on the runtime of the cache simulator, as well as whether the results of the scheduling can be displayed interactively/in an interesting manner, we might have a demo showing the actual process and results of the scheduling algorithm. However, this demo would require time to be spent on the visualization for the cache simulator, which would be a low priority for us given the other goals of the project.

Main ToDo/ concerns

Mainly have to allocate more time to code and integrate the profiling and scheduling algorithm. The integration is important for us to be able to measure scheduling overhead.

The accuracy of the cache simulator still needs to be verified. If there are major bugs found in the cache simulator implementation, it might result in delays to the timeline of the project.