Part III: Academic research

List three things that surprised you, about the work itself or things the authors mentioned.

1. I was surprised when beginning reading this article that I had not thought about the possibility of implementing a scheduler in hardware. For something that is so crucial to an OS it would make sense that it is as fast as possible and therefore made in hardware.

2. I was initially surprised by the variability in how all the different parallel algorithms performs in the three different schedulers. Primarily I was surprised by how comparatively poorly the software scheduler performed.

3. In figure 6 there is the time breakdown for schedulers time spent on the algorithms tested. I was surprised with some of the algorithms in how much time was sent not running. For example in `maxflow`, at large thread counts a majority of time isn't spent running but is spent starved

List three questions the paper raised in your mind.

1. As one of the simulated parallel workloads the study uses `gtfold`. Is this the algorithm that softwares like Folding@home would use?

2. One thing that I am a bit confused about is when the author talks about tailoring algorithms for specific application characteristics. Is this something that normal schedulers do, or is it talking about specializes schedulers designed to be optimized for a single application/process?

3. The paper describes scenarios/algorithms in which Carbon outperforms ADM but does not include much about algorithms in which software outperforms both Carbon and ADM. Do these exist?

List one thing this paper made you want to learn more about.

1. This study used the M5 simulator for modeling the memory hierarchy of a system. After some brief research I have learned that the M5 simulator was initially created for network simulation but since then has increased to in including full-system simulation capability, a detailed I/O subsystem, and the ability to simulate multiple networked systems deterministically. I would be interested in learning more about this simulation software.

What kind of workloads particularly benefit from ADM? Why?

It seems like processes that have great variability in the task size benefit most from ADM. `ced` has very small tasks mixed with long phases with deep queues and a mild imbalance, with this algorithm ADM outperforms both Carbon and software implementations at all thread counts. Similarly `cg` has long tasks followed by imbalanced phases and short tasks. For this algorithm as well ADM outperform the other schedulers, and stretches its advantage at higher thread counts. Finally, `gtfold` has both large tasks many short, imbalanced parallel phases: this is another algorithm where there is great variability in task size and again one where ADM dominates. ADM is best suited for these tasks because unlike Carbon where the local task unit (LTU) "prefetches a long task while executing another long task, leading to imbalance since tasks

cannot be reclaimed from LTUs. The software and ADM schedulers do not have this problem, but ADM scales better than software."

Similarly, ADM greatly beats out normal software schedulers due to its ability to implement asynchronous scheduling and bypassing the cache hierarchy depending on the application.

What kind of workloads don't see particularly dramatic improvements with ADM? Why?

Out of all the parallel tasks, ADM performs worst as compared to Carbon in the `mergesort` algorithm. The paper claims that ADM struggles due to the tree-style parallelization of the algorithm. Where only a few threads are generating new tasks and those are needed to be redistributed ASAP. These characteristics best suite the speed of encoding the scheduler in hardware: ie. the Carbon scheduler.

How important is caching though?!

Caching is crucial. As the paper puts it "The latency of a cache line transfer in CMPs with 64 or 128 cores is close to a hundred cycles, so a few such transfers can negate the benefits of parallel execution of fine-grain tasks." Meaning, all of these schedulers are working to be as efficient as possible to manage threads while maintaining effective cache hierarchies. As, if there are even as few as a couple cache misses then all of the scheduler benefits may be undone.