Parallel P2 Final

Claude Richoux

April 2019

1 Testing

For correctness testing, I made sure that the right numbers of packets were passing through each stage, checked that my queues were FIFO and weren't dropping packets or messing up their length counters, and checked that with the same seed the workers generated the same set of hashes as with serial. I think that's about all I can do to check that this is correct.

For performance testing, I wrote Python scripts in `tests/` to repeatedly run my executable with different parameters and time its performance, then calculate whatever needed to be plotted and dump that to a CSV for import into Google Sheets. I also had an Sbatch file to submit the python scripts to Slurm on the cluster.

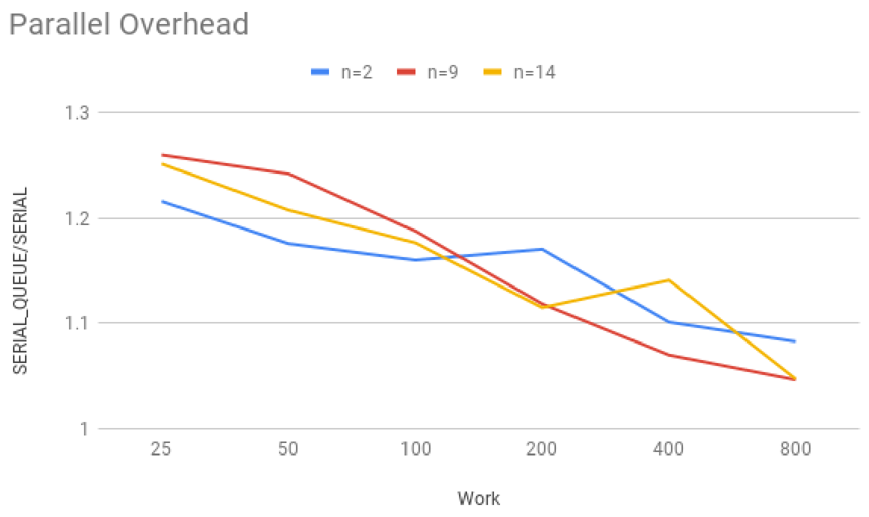
2 Parallel Overhead and Worker Rate

My parallel overhead was generally between 1 and 1.25, so not too bad. The n values didn't actually seem to correlate too strongly with more overhead, which was sort of weird, but the trend I saw with more work meaning less overhead made sense because the serial queue code would be spending proportionally more time on work than it would be on thread-related shenanigans.

I didn't really get a constant worker rate, so I think what may be happening is that the dispatcher is the bottleneck in the serial code? My worker rates were almost perfectly inverse-proportional to W (see the really well-fitting trendlines), with different constants depending on n. When $n = 2$, the worker rate is pretty constant, but for larger n, the worker rate decreases at about the same rate for more work regardless of n.

3 Dispatcher Rate

The dispatcher rate at first went up steadily as more workers were added. I think this means the queues were consistently full while it was running, for these smaller n. It peaked around 8 workers, then started going down a little when $n = 13$, and much lower with 27 workers. I think what may have happened

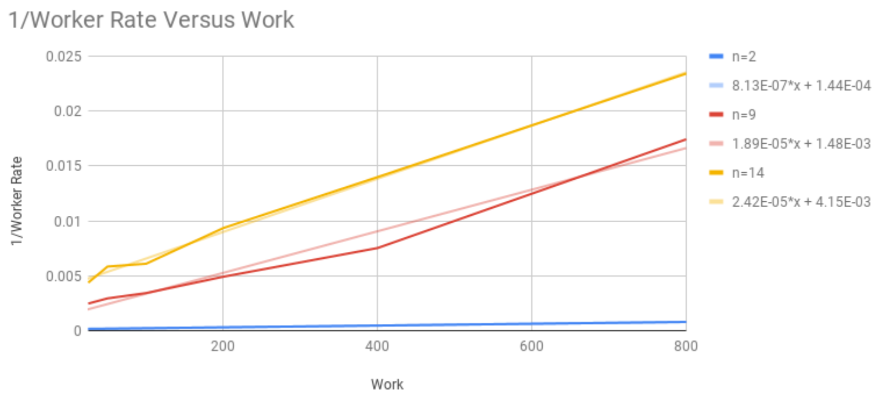


with 13 workers was that the dispatcher couldn't fill the queues as fast as the workers were emptying them, or possibly I ran out of threads and threads were having to wait around to get CPU time. I think my laptop (which is what I tested these smaller experiments on) supports 16 threads (unless I'm misreading lscpu), but some of the threads were probably busy handling my 900 Chrome tabs, so either hypothesis is plausible here.

4 Speedup with Constant Load

These last few experiments, I ran on Midway2 with 16 CPUs and 1 thread per core, to keep my performance pretty similar to the the partitions that Slurm would assign on the CS department servers. The partition I ran on had 2.40GHz clock speed, which is a bit slower than CSIL's, but I don't think that should affect ratios between serial and parallel.

For constant load, I got pretty much what I expected- speedup increases until we have more threads than cores to run them on, then threads are idling and things slow down. Overall constant load had really good speedup, because no thread was lagging behind the others because it had a huge workload. The 8000 and 4000 for W had the best speedup, because the ratio of time spent working rather than time spent doing thread shenanigans was much higher when there was more work to do overall.

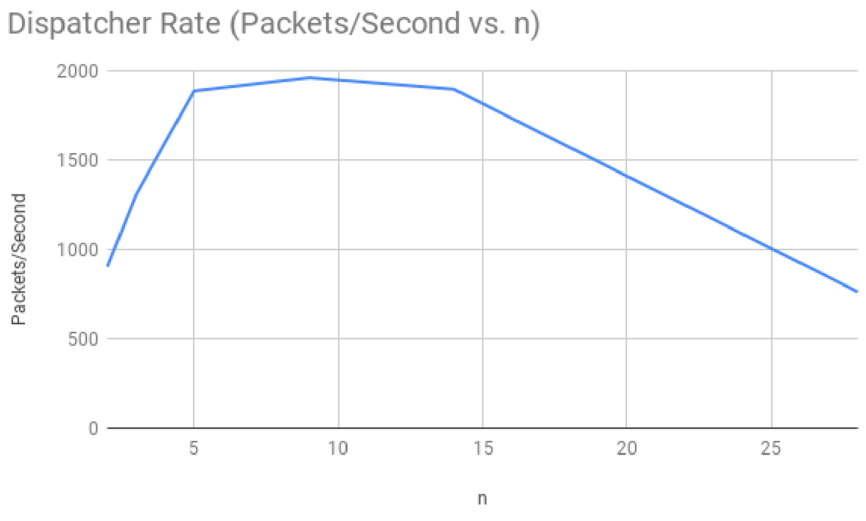


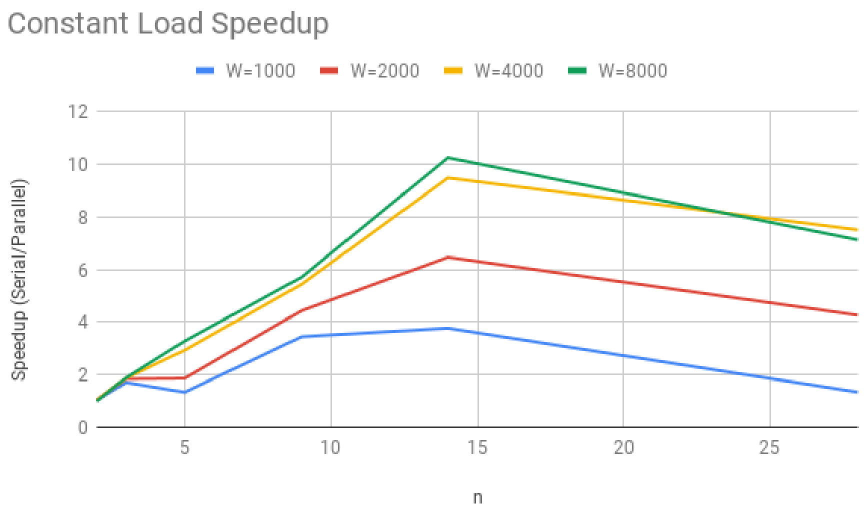
5 Speedup with Uniform Load

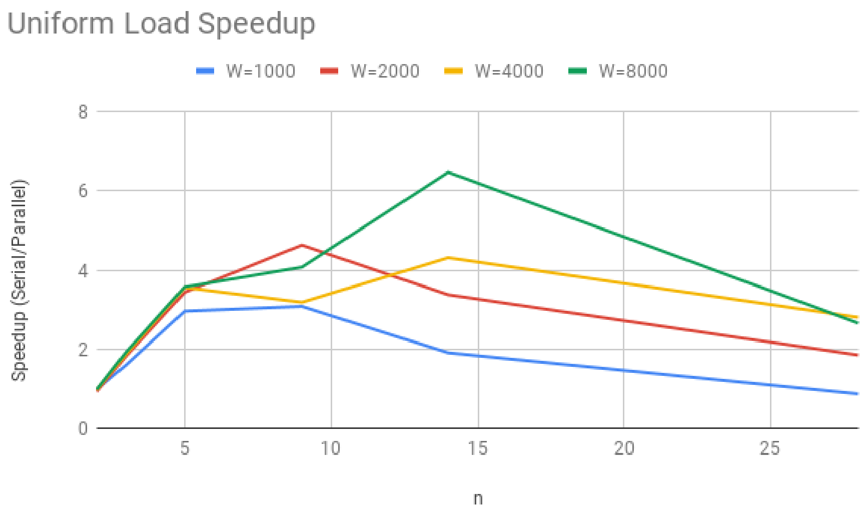
Uniform load was a bit less perfect than constant load- the work level didn't necessarily correlate one-to-one with which run had the best speedup, because some threads could be bit slower than the others due to the spread of the distributions. Speedup was also overall smaller than constant load, again because the program can't finish until the slowest thread does. The same pattern with running out of cores to run threads on shows up here as well, because that didn't change between runs.

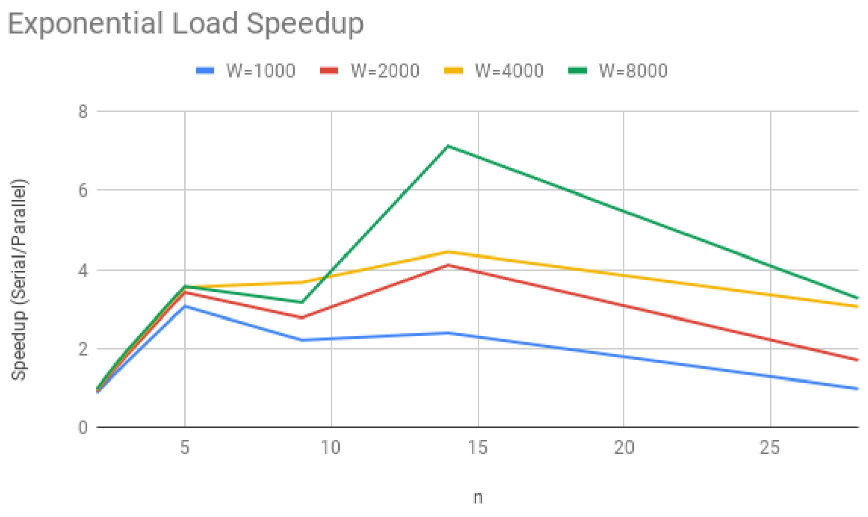
6 Speedup with Exponentially Distributed Load

This one for the most part has much the same patterns as with uniform load, except a bit more extreme because of the bigger spread of exponential load. There is a weird dip around $n = 9$ and I have no idea why that might be. $W = 8000$ is also sort of an outlier when $n = 14$, I would have thought that performance would have dipped because such a large W has the potential to have huge outliers in workload per thread with the exponentially distributed packets, but I guess not?


Dispatcher Rate (Packets/Second vs. n)
Packets/Second
n
0
500
1000
1500
2000
5
10
15
20
25


Constant Load Speedup
W=1000
W=2000
W=4000
W=8000
Speedup (Serial/Parallel)
n
0
2
4
6
8
10
12
5
10
15
20
25


Uniform Load Speedup
W=1000
W=2000
W=4000
W=8000
Speedup (Serial/Parallel)
n


Exponential Load Speedup
W=1000
W=2000
W=4000
W=8000
Speedup (Serial/Parallel)
n