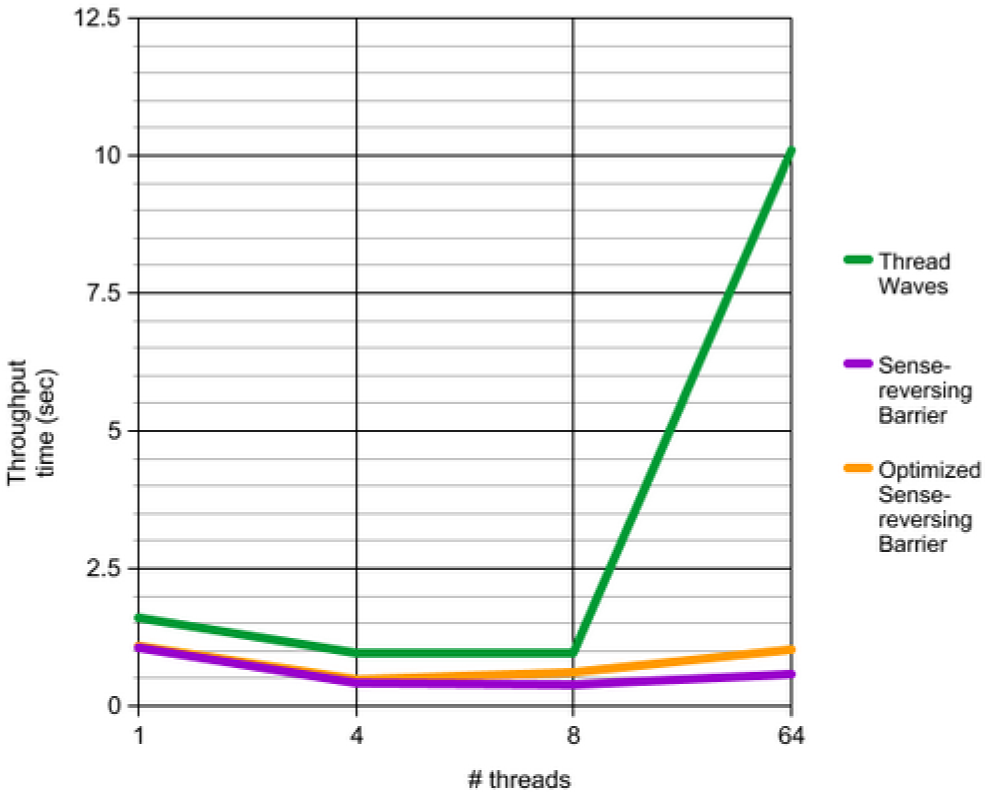
Ari 'aith' Iramanesh

In part 1 we implement phase-based concurrency on algebraic tasks in various ways; once with sequentially created threads per phase, once with barriers to reuse threads, and once with barriers with optimizations.

Here are the results:



The throughput time (sec) of each implementation of a 4-core machine

# Threads	Thread Waves	Sense-reversing Barrier	Optimized Sense-reversing Barrier
1	1.61568	1.05206	1.10015
4	0.951205	0.411107	0.49379
8	1.3547	0.399787	0.59602

64	10.0944	0.583163	1.02342

I found that the Thread Waves was consistently slower than the sense-reversing barrier. This was expected behaviour because Thread Waves creates a set of threads for every repetition of blur(), and the creation of a thread is expensive due to the allocation of memory. The sense-reversing methods allow the reusage threads by syncing each thread up in “phases.” Surprisingly, the ‘optimizations’ (Relaxed peeking and yielding on spinloop) actually consistently lowered performance. This is likely due to the internal workings of the OS-specific yield() implementation. The Relaxed Peeking did not affect performance, likely because the prior atomic_fetch_add acts as a fence, restricting the instructions’ ability to be performed faster.