Parallel Computing

A1 Report

Anton Chen a11911337

Description atomic implementation:

In the atomic implementation I have set all the statistical counters that are used in the worker threads to atomic, since these are shared variables. Due to the sum operation being non-atomic, I had to find a workaround that involved compare_exchange_weak() to guarantee the right result.
The producer has been initialized as a future and launched as its own thread with launch::async.
The push and pop operation of the SafeQ class have both been locked with the same spin_lock as they would otherwise create a datarace. The empty() function is used in the worker() function instead of size(), because I perceive it as more secure. Furthermore, the empty() function checks a non-atomic Boolean called rFinished, that is set to true if the producer has completed its work, AND THEN checks whether the queue is empty or not. If the queue happened to be empty at the time of call and the producer was not finished yet, the thread would simply yield() and then skip the statistical part with continue. The biggest overheads are created by the SafeQ, as both pop and push operations are using the same lock.

Atomic performance issues:

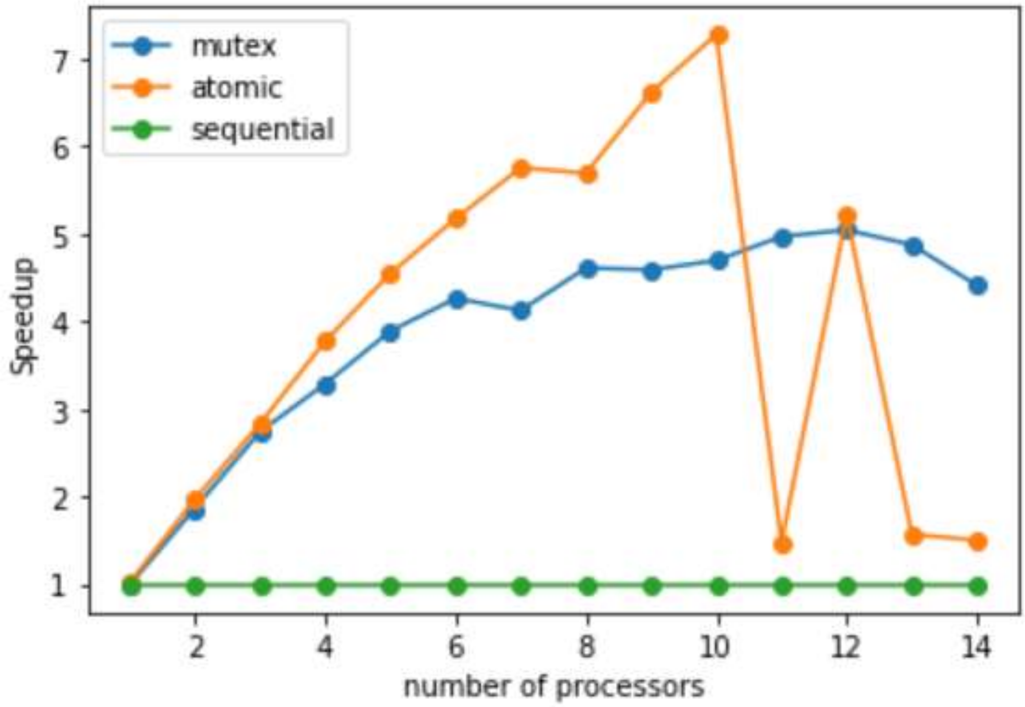
The atomic implementation goes very well with up to 10 processors. Any more than that can lead to underperformance as seen in the graph below. The reason is likely that the worker-threads outspeeded the producer-thread and are hindering the producer thread from entering its critical section. This_thread::yield() mitigates this problem somewhat, but the performance can drop nevertheless.

Description mutex implementation:

The mutex implementation is very similar to the atomic implementation, except pop and push are implemented with Mutex-lock instead of spin_lock and all the statistical parts are not atomic, but each needed their own locks, that are instantiated in main().

Mutex performance issues:

The mutex implementation is overall slower than the atomic implementation, but the speedup stays somewhat consistent even with more than 10 workerthreads. The likely reason is that in this implementation, the worker_threads are also sharing the locks of the statistical primitives, making them slower overall and unlikely to outspeed the producer.



Time Table Processors:	Sequential	Atomic		Mutex	
	Time	Speedup	Time	Speedup	Time
1	10.18	1.01092354	10.07	0.99027237	10.28
2	10.18	1.96525097	5.18	1.86106033	5.47
3	10.18	2.82777778	3.60	2.73655914	3.72
4	10.18	3.77037037	2.70	3.28387097	3.1
5	10.18	4.54464286	2.24	3.88549618	2.62
6	10.18	5.16751269	1.97	4.25941423	2.39
7	10.18	5.75141243	1.77	4.12145749	2.33
8	10.18	5.68715084	1.79	4.60633484	2.21
9	10.18	6.61038961	1.54	4.58558559	2.22
10	10.18	7.27142857	1.4	4.69124424	2.17
11	10.18	1.47536232	6.9	4.96585366	2.05
12	10.18	5.22051282	1.95	5.03960396	2.02
13	10.18	1.56615385	6.5	4.8708134	2.09
14	10.18	1.49926362	6.79	4.40692641	2.31