RDCSS Discussion

Algorithm Implementation Approach

My implementation of RDCSS follows pretty closely to what Tim Harris provided in his paper. There were a few operations that were “left as an exercise to the reader”, which were *IsDescriptor(r)*, *CAS1(...)*, and calling atomic operations on dereferences.

For *IsDescriptor(r)*, I opted on utilizing bit marking of the descriptor pointer swapped into the data section, such that I can check the lowest bit in *r* is equal to 1. This implementation also restricted the data section to be a word sized pointer, such that the lowest bit of the data section’s value is always 0.

For *CAS1(...)*, I opted to perform a compare and swap from C’s atomic library, that would also return the old value at *a*, given that the expected old value does not match *a*. This allows us to always get the old value of *a*, while performing an atomic swap at *a*.

Lastly, during several sections, there were dereferences that were not atomic in Tim Harris’ implementation. There were potential problems with not setting these atomic, such that values may not be updated to the thread’s cache. So as you can see in my implementation, I chose several dereference sections to be *atomic_load()* instead.

Performance Analysis

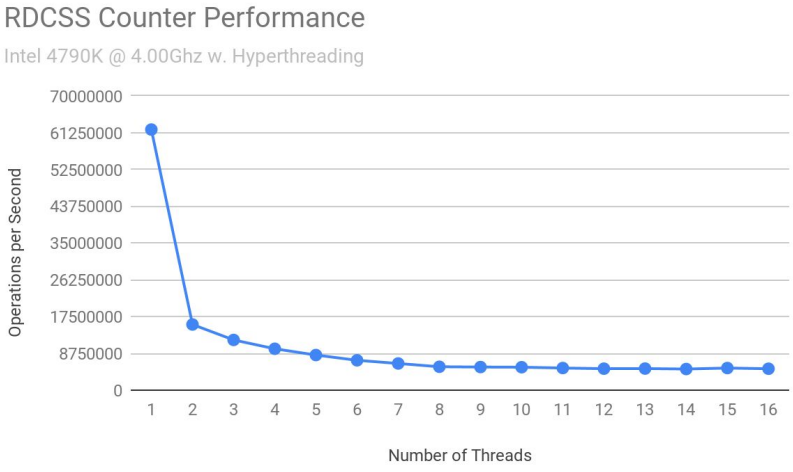

Fig 1. Performance plot of RDCSS on counter.

RDCSS performance declined in the counter test as more threads were added to the computation. You can see an exponential decay in operations per second (fig 1) as the number

of threads increases, due to the increased number of failed RDCSS calls. The reasoning behind this performance loss is that as more threads are added, there are more operations interleaving an RDCSS call that can update *o2* before another thread can inject it's descriptor in the data section, leading to a failed RDCSS update. This can occur many times, causing threads to make new RDCSS calls that do not update the data section with their descriptor. Furthermore, it is also the case that many threads may "get" the same descriptor from the *CAS1(...)* operation, which would cause many threads to perform the complete operation on the same descriptor, where all but one will finish succesfully. The combination of greatly increased chances of the data section not matching *o2*, and the redundancy brought by increasing number of threads performing the same computation, causes an increase of threads to bring performance losses. One could expect good performance, or enhanced performance with increasing number of threads, if only one thread could grab the descriptor in an efficient way: as well as good ordering of events, such that each thread can successfully inject a new descriptor with an update instruction, or if not, only one thread grabs the returning descriptor.

Number of Threads ▲	Final Count	Operations Per Second
1	4650107876	62001438.35
2	1171432552	15619100.69
3	895755500	11943406.67
4	738275686	9843675.813
5	626315044	8350867.253
6	532490452	7099872.693
7	475874618	6344994.907
8	417329656	5564395.413
9	412123876	5494985.013
10	409258946	5456785.947
11	394597742	5261303.227
12	383130492	5108406.56
13	384982188	5133095.84
14	376821666	5024288.88
15	394280142	5257068.56
16	382113066	5094840.88

Fig 2. Table chart describing performance per number of threads.