Assignment 2 Report

CMPT – 431

Joshua Campbell - jkcampbe@sfu.ca

Adam Penner – adpenner@sfu.ca

Test Machines:

AMD Phenom II X4 965 @ 3.4 GHz (4 GB DDR3 RAM, 7200 RPM HDD)

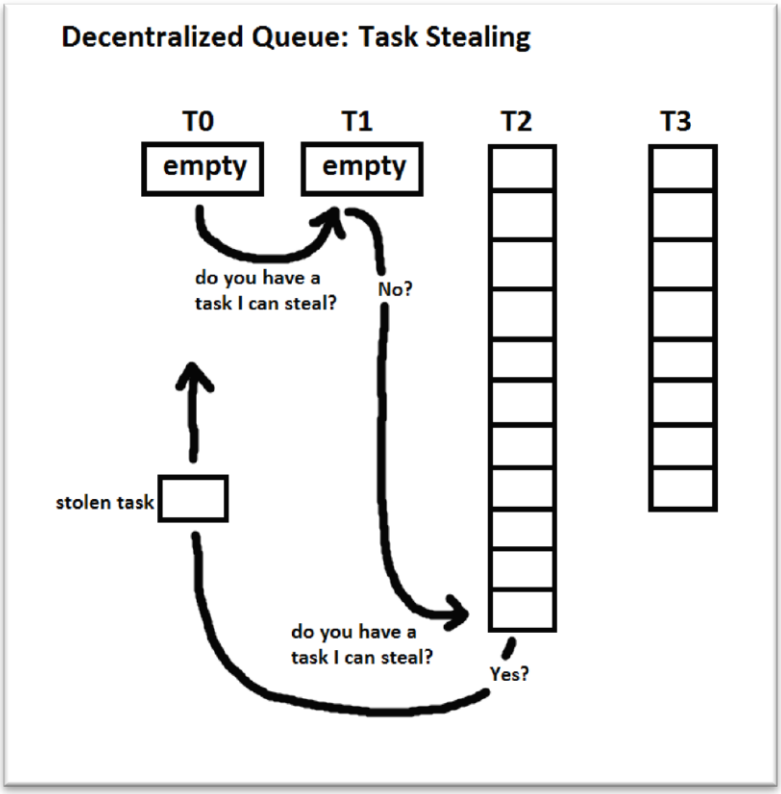
Amoeba-N3

AMD FX-8350 Eight-Core @ 4.0 GHz (10 GB DDR3 RAM, 7200 RPM HDD)

Assumptions:

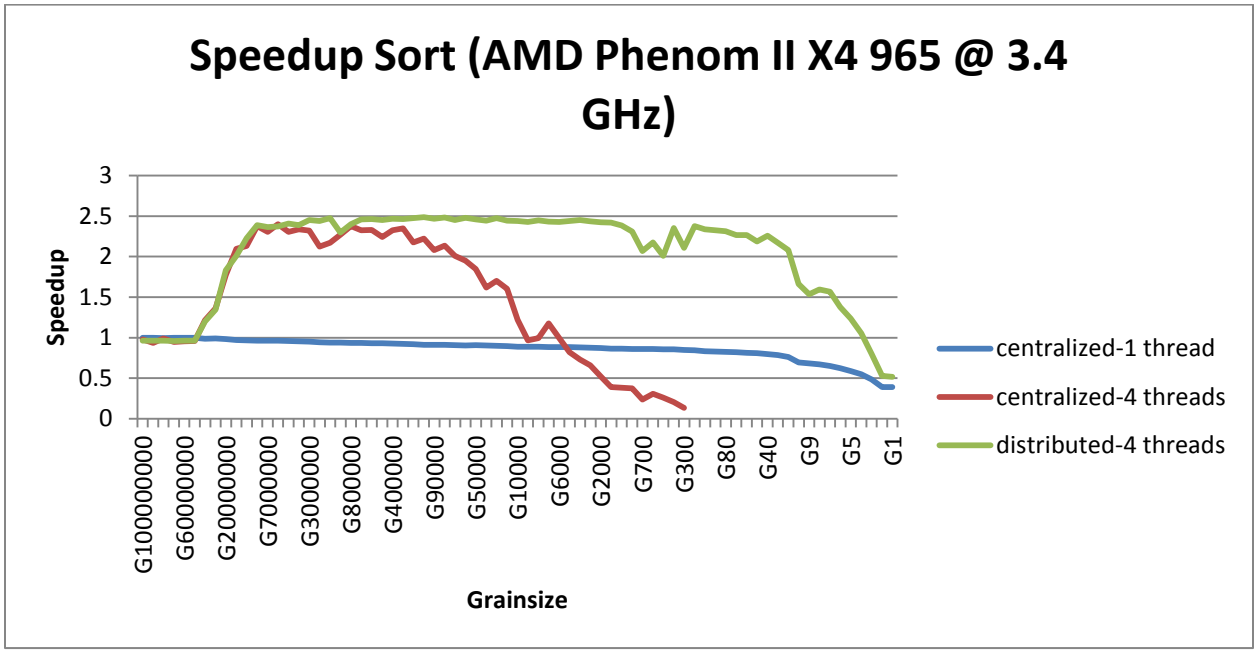
Due to the nature of how these programs run (splitting the data/particles based on the size of the grainsize) in order to normalize to the infinite grainsize, we chose to run from 10,000,000-100,000,000 being the base case for the infinite grainsize (the infinite grainsize must be greater than or equal to the particle size) depending on the number of particles for the test.

Distributed Queue with Stealing:



In order to create the distributed queue, the program was modified such that each thread would pull from their own individual queues (in order to reduce contention). In order to load balance, the steal() function was utilized using a round-robin stealing technique where each queue would try to steal from its neighbour when their queue became empty (this attempt to steal would carry on until a task is stolen or the caller made it back to itself).

Driver-Sort (40,000,000 particles):



Smallest grainsize that is within 5% of the infinite grainsize?:

For the Driver-Sort program, the infinite grainsize was 40,000,000 (the same size as the number of particles) because the grainsize was equal to the number of particles and hence would not be broken up into multiple tasks.

For the centralized queue using a single thread, the execution time for this grainsize was 2.50773 seconds (using the AMD Phenom II X4 965 machine). The smallest grainsize that was still within 5% of this was 20,000,000 (2.5193 seconds). The main reason for this was that as the grainsizes became smaller, the execution time increased on the single threaded run due to the overhead caused by having more than one task.

For the centralized queue using four threads, the execution time for this grainsize was 2.03851 seconds. The smallest grainsize that was still within 5% of this was 10,000 (2.03104 seconds). This grainsize is much smaller than the single threaded run due to the fact that as the problem is split up into more tasks, the program greatly benefits from the increased parallelism that is brought from using 4 threads/cores instead of just the one.

For the distributed queue using four threads, the execution time for this grainsize was 2.06513 seconds. The smallest grainsize that was still within 5% of this was 5 (2.00607 seconds). This is even smaller than the centralized task queue even though they both were running on the same number of threads/cores. The reason for this is that due to the nature of the centralized queue, as the tasks become smaller, the queue experiences more and more contention as the threads

finish their tasks faster and then try to pull from the queue again. The distributed queue does not suffer from this problem as each thread only pulls out tasks from their own queue (eliminating the contention of the centralized queue).

Maximum Speedup (4 cores):

For this data, the base case will be the infinite grainsize (40,000,000) running on the centralized task queue using a single thread/core (2.50773 seconds on the AMD Phenom II X4 965 machine).

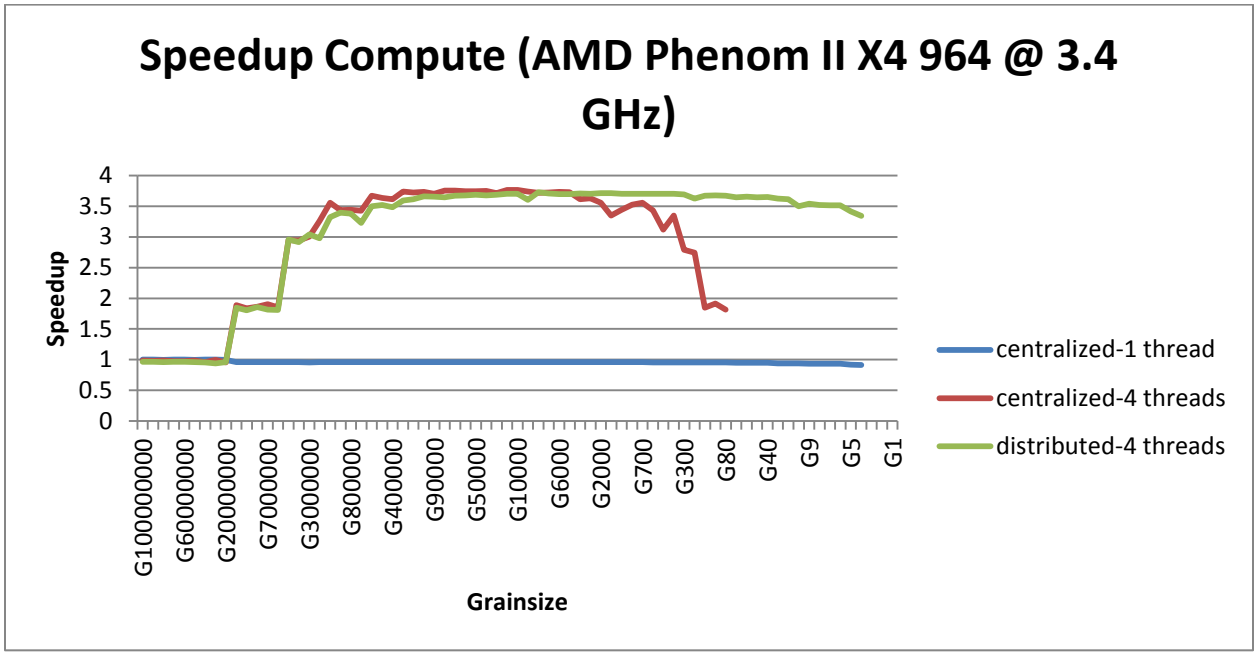
For the centralized task queue (using 4 cores), the maximum speedup occurred at approximately the grainsize of 6,000,000 with a maximum speedup of 2.4008 times that of the single core runtime.

For the distributed task queue (using 4 cores), the maximum speedup occurred at approximately the grainsize of 100,000 with a maximum speedup of 2.4875 times that of the single core runtime.

The Decentralized Task Queue:

Based on the data gathered, the distributed queue is almost always faster or at least equal to the centralized queue due to it not having the same queue contention overhead found when using multiple cores/threads. Not only is the decentralized task queue faster than the centralized task queue (especially as task sizes become smaller), but it also can run at much smaller task sizes due to the nature of the centralized task queue overflowing the stack space of the thread that it is hosted on. Another reason why the distributed queue might be performing better at smaller grainsizes is that in a centralized queue, which thread gets which part is completely random based on when the finish. This can cause cache-line conflicts in the array that it is sorting which would impact the performance. The distributed queue avoids this problem for the most part by dividing up its own parts into smaller pieces in its own queue.

Driver-Compute (10,000,000 particles):



Smallest grainsize that is within 5% of the infinite grainsize?:

For the Driver-Compute program, the infinite grainsize was 10,000,000 (the same as the number of particles).

For the centralized queue using a single thread, the execution time for this grainsize was 8.3768 seconds (using the AMD Phenom II X4 965 machine). The smallest grainsize that was still within 5% of this was 5 (8.77581 seconds). In comparison to the Driver-Sort program, this grainsize is much smaller. The reason for this is because the overhead caused by separating the particles into smaller tasks in the Driver-Compute program is miniscule in comparison to actually processing the tasks themselves.

For the centralized queue using four threads, the execution time for this grainsize was 4.26789 seconds. The smallest grainsize that was still within 5% of this was 80 (4.4401 seconds).

For the distributed queue using four threads, the execution time for this grainsize was 4.36134 seconds. The smallest grainsize that was still within 5% of this was 4 (2.41156 seconds).

Maximum Speedup (4 cores):

For this data, the base case will be the infinite grainsize (10,000,000) running on the centralized task queue using a single thread/core (8.3768 seconds on the AMD Phenom II X4 965 machine).

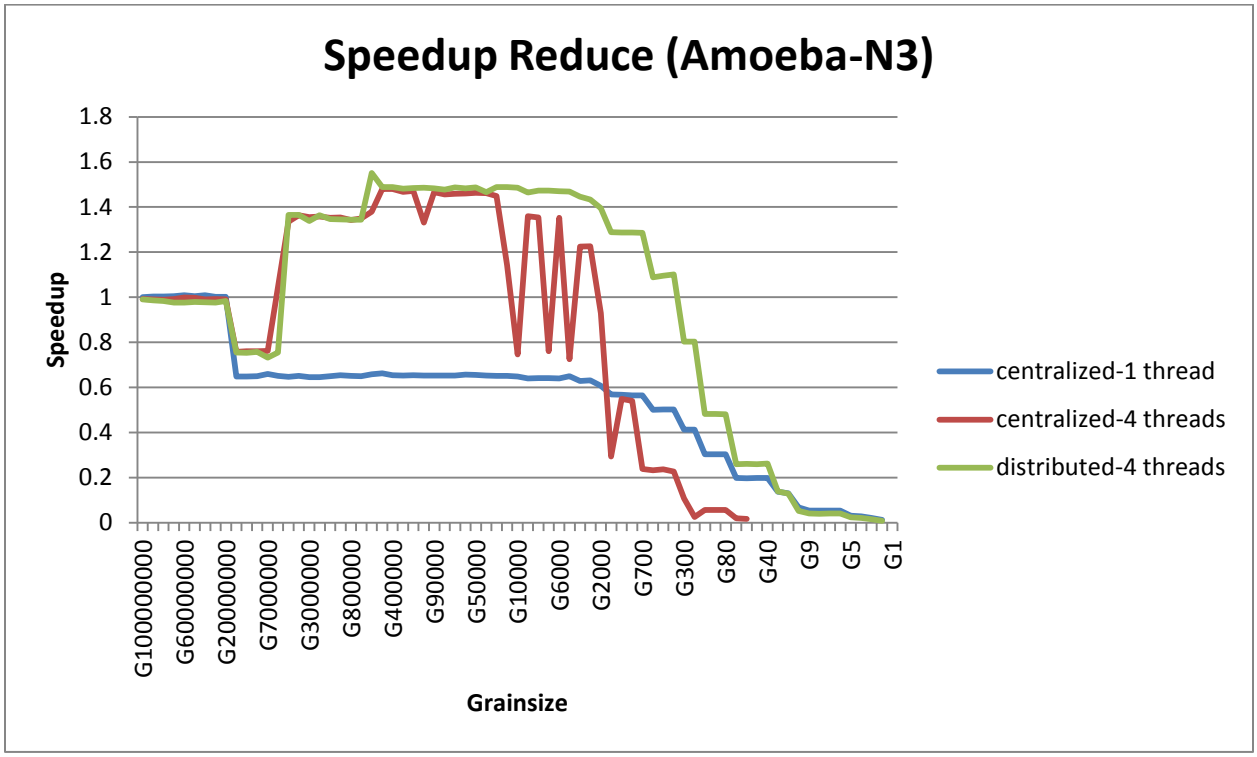
For the centralized task queue (using 4 cores), the maximum speedup occurred at approximately the grainsize of 20,000 with a maximum speedup of 3.7668 times that of the single core runtime.

For the distributed task queue (using 4 cores), the maximum speedup occurred at approximately the grainsize of 8,000 with a maximum speedup of 3.7269 times that of the single core runtime.

The Decentralized Task Queue:

The decentralized task queue seems to perform slower in comparison to the centralized task queue in this program when using large grainsizes. However, when it comes to the smaller grainsizes, the decentralized queue still outperforms the centralized queue due to the contention between the threads on the central queue as well as the possible cache line conflicts between the threads.

Driver-Reduce (10,000,000 particles):



Smallest grainsize that is within 5% of the infinite grainsize?:

For the Driver-Reduce program, the infinite grainsize was 10,000,000 (the same as the number of particles).

For the centralized queue using a single thread, the execution time for this grainsize was 0.0246491 seconds (using the Amoeba-N3 machine, the data gathered for on the AMD Phenom II X4 965 machine did not match the data found on the AMD FX-8350 machine and the Amoeba-N3 machine). The smallest grainsize that was still within 5% of this was 3000 (0.0253152 seconds).

For the centralized queue using four threads, the execution time for this grainsize was 0.0211167 seconds. The smallest grainsize that was still within 5% of this was 2000 (0.0172128 seconds).

For the distributed queue using four threads, the execution time for this grainsize was 0.0211698 seconds. The smallest grainsize that was still within 5% of this was 200 (0.019911 seconds).

Maximum Speedup (4 cores):

For this data, the base case will be the infinite grainsize (10,000,000) running on the centralized task queue using a single thread/core (0.0246491 seconds on the Amoeba-N3 machine).

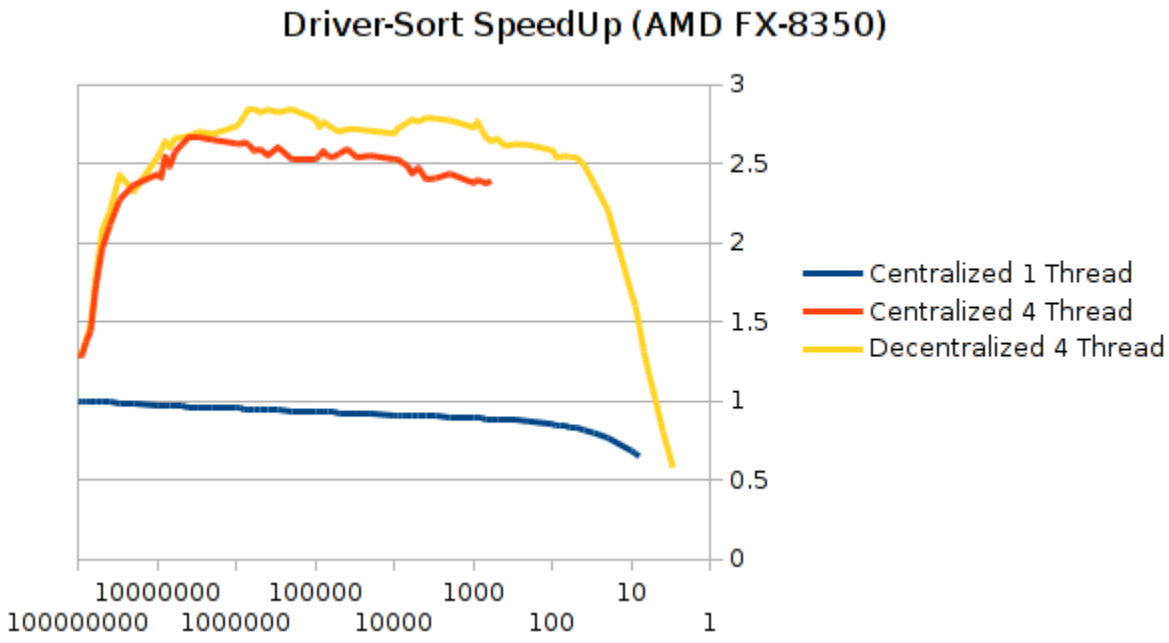
For the centralized task queue (using 4 cores), the maximum speedup occurred at approximately the grainsize of 400,000 with a maximum speedup of 1.480 times that of the single core runtime.

For the distributed task queue (using 4 cores), the maximum speedup occurred at approximately the grainsize of 600,000 with a maximum speedup of 1.551 times that of the single core runtime.

The Decentralized Task Queue:

Based on the data gathered on the Amoeba-N3 machine, the distributed queue tends to outperform the centralized queue only when the grainsizes are small. It would appear that contention between the threads using the centralized queue is what is causing this once again. In this case, the centralized queue dropped in computing power at larger grainsizes than the previous two programs. This is because the reduce function is less computationally intensive resulting in more contention on the central queue quicker as the threads complete their tasks at faster rates. Once again, cache line conflicts in the centralized task queue could be exacerbating the problem.

Driver-Sort (100,000,000 Particles):



Smallest grainsize that is within 5% of the infinite grainsize?:

The smallest grainsize that is still within 5% of the infinite grainsize is a grainsize of 700,000 (with an infinite grainsize of 100,000,000). The infinite grainsize time was 6.08371 seconds while the other grainsize was 6.39825 seconds. As the assignment stated, the reason why using the particle size as the grainsize is due to the fact that the code simply acts on the sorting algorithm then. But once we start splitting into smaller grainsizes, there is added overhead from the action of splitting the array.

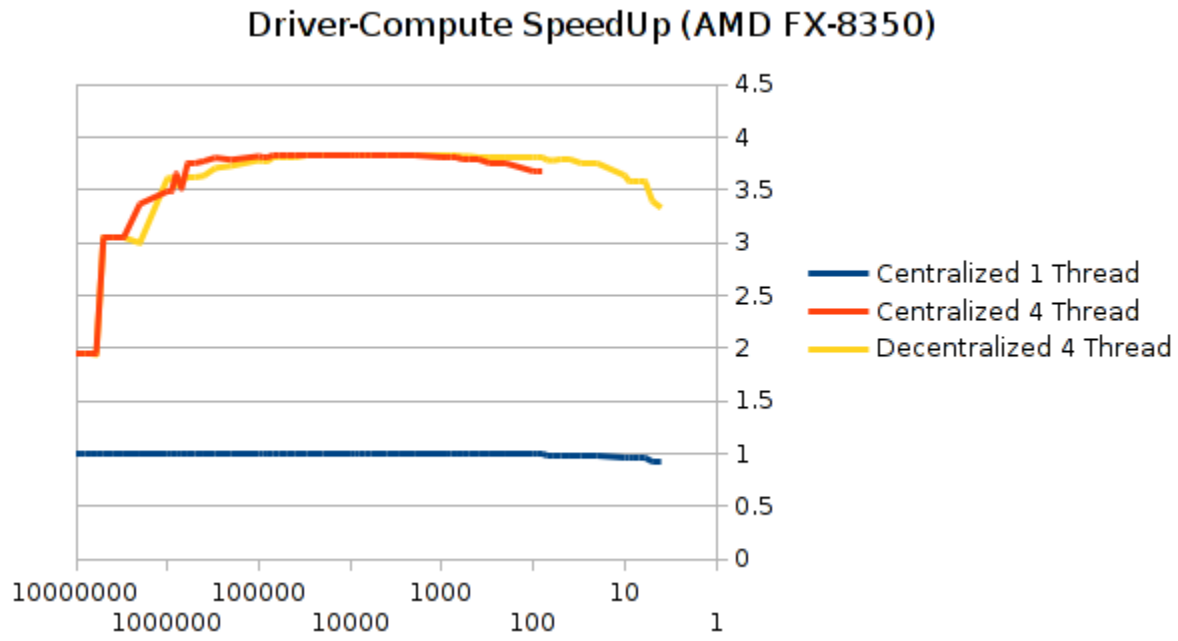
Maximum Speedup (4 cores):

The maximum speedup on four cores is 2.8585x when comparing the grainsize of 20,000 between single core and four cores.

The Decentralized Task Queue:

For my tests the driver-sort code, the distributed queuing was always quicker than the centralized version (other than a single outlier). The distributed queue helps reduce contention as it allows for more tasks per thread to be done before attempting to steal from another thread. Once the stealing starts though, it could also be distributed across multiple threads which could cause for a lesser rate of contention. With the distributed queuing though, there is a balance between thread work load and granularity. Both curves, especially the decentralized version, shows this balance perfectly. With the higher grainsize, we have a load imbalance dominates as there is more work to do, while on the smaller grainsize we have increased overhead due to enqueuing and dequeuing many tasks.

Driver-Compute (10,000,000 Particles):



Smallest grainsize that is within 5% of the infinite grainsize?:

The smallest grainsize that is still within 5% of the infinite grainsize is a grainsize of 6 (with an infinite grainsize of 10,000,000). The infinite grainsize took 7.71435 seconds while the other took 8.04149 seconds.

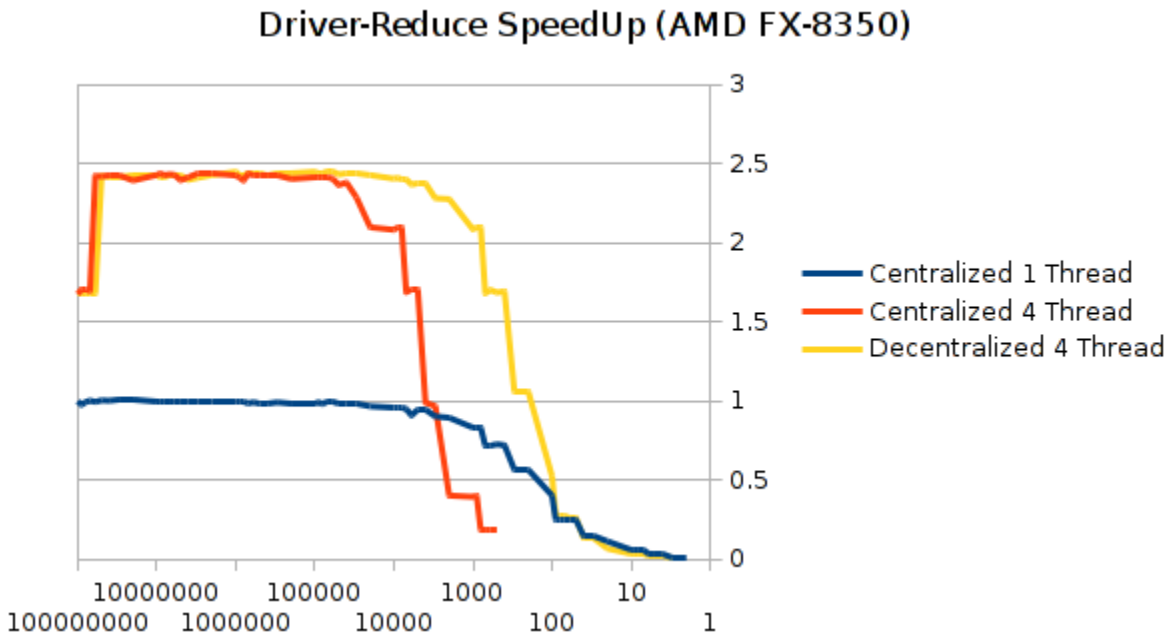
Maximum Speedup (4 cores):

The maximum speedup on four cores was 3.8364x compared to the single core. With the grainsize of 9000, the four cores took 2.01337 seconds to compute and 7.72407 seconds for the respective single core.

The Decentralized Task Queue:

The distributed queue was not always the fastest. In the case of centralized and distributed queues, there was barely any difference between the times. The major difference that was noted is that with the distributed queuing system we are able to use smaller grainsizes before hitting a high overhead due to queuing and dequeuing tasks. There was also a small blip in between 100K and 1M grainsize where the distributed was actually worse than the centralized.

Driver-Reduce (100,000,000 Particles):



Smallest grainsize that is within 5% of the infinite grainsize?:

The smallest grainsize that is still within 5% of the infinite grainsize is a grainsize of 7,000 (with an infinite grainsize of 100,000,000). The infinite grainsize took 0.1184 seconds while the other took 0.1245 seconds.

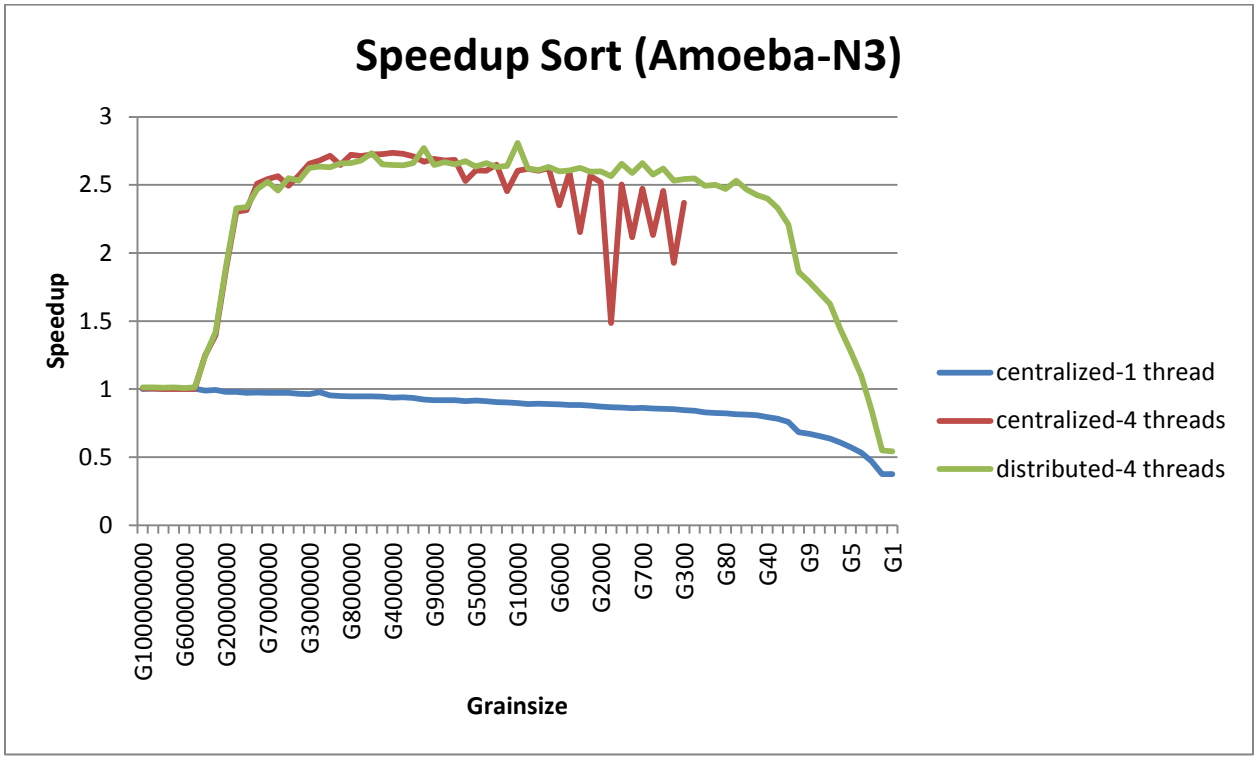
Maximum Speedup (4 cores):

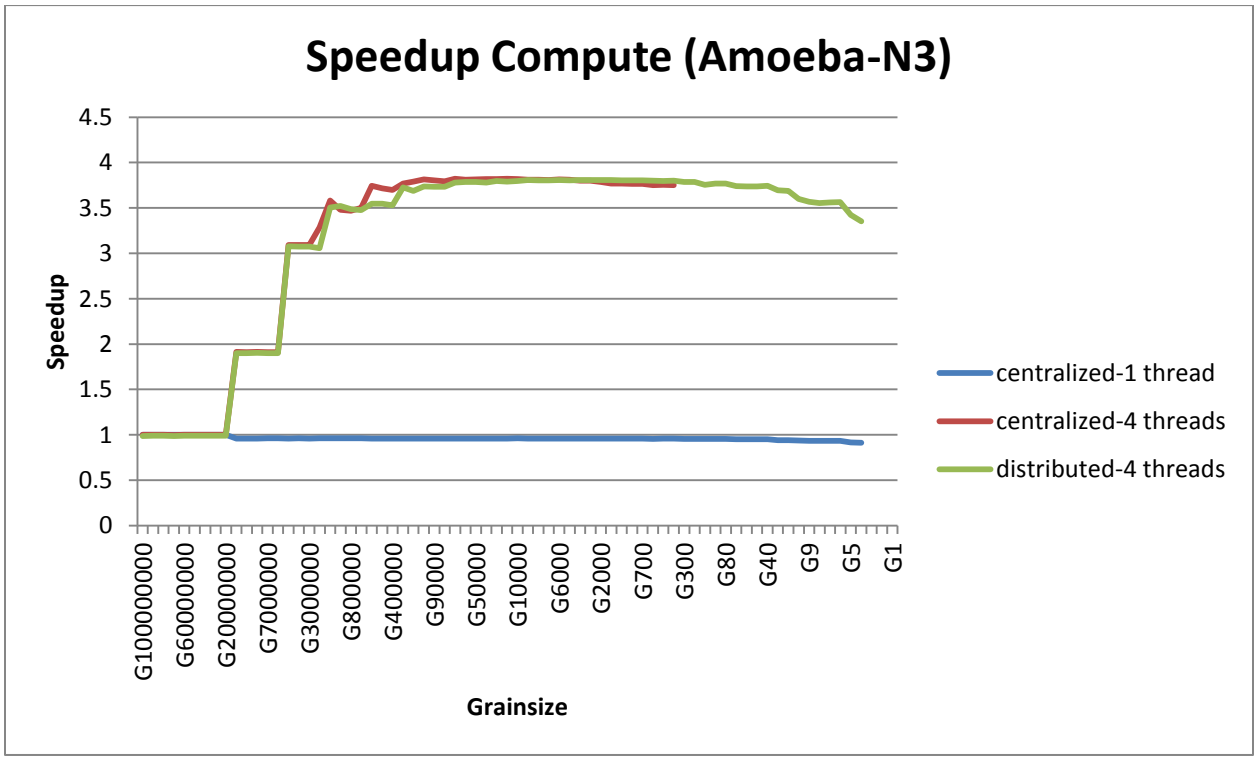
The maximum speedup on four cores was 2.4709x compared to the single core. With the grainsize of 400000, the four cores took 0.0487 seconds and 0.1204 seconds for the respective single core.

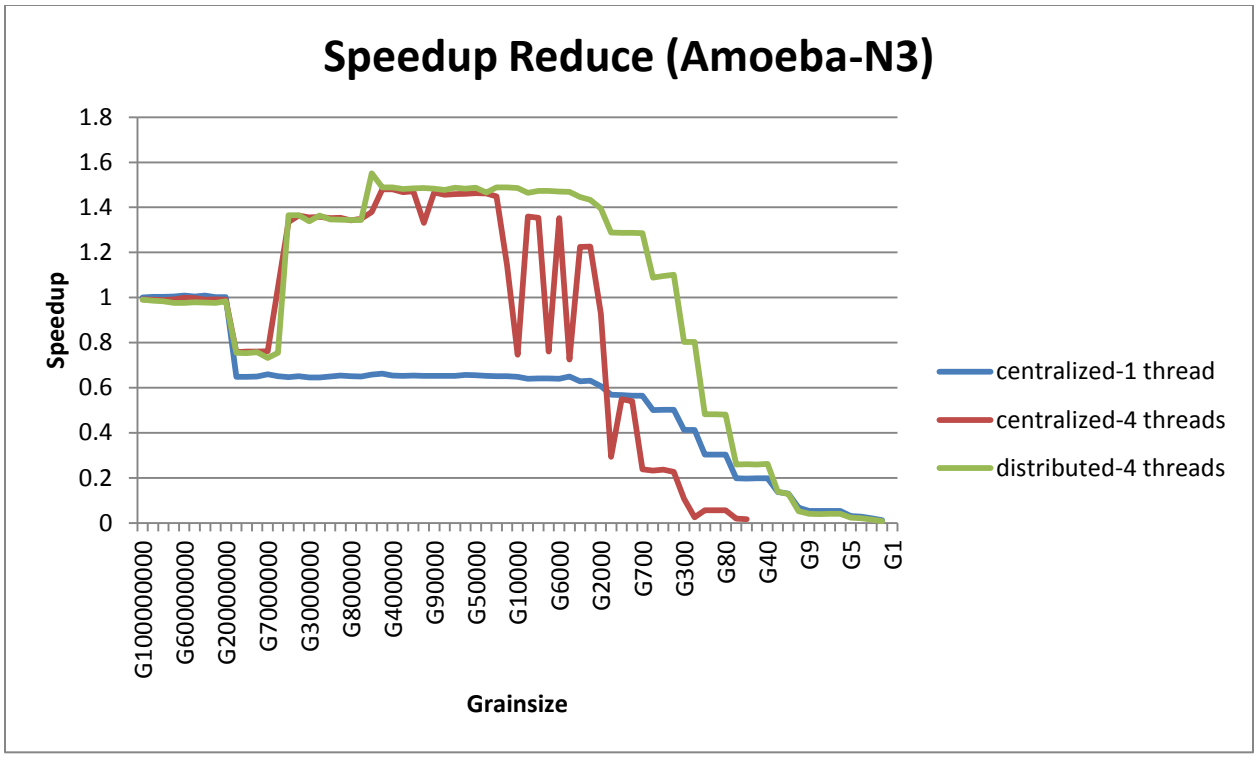
The Decentralized Task Queue:

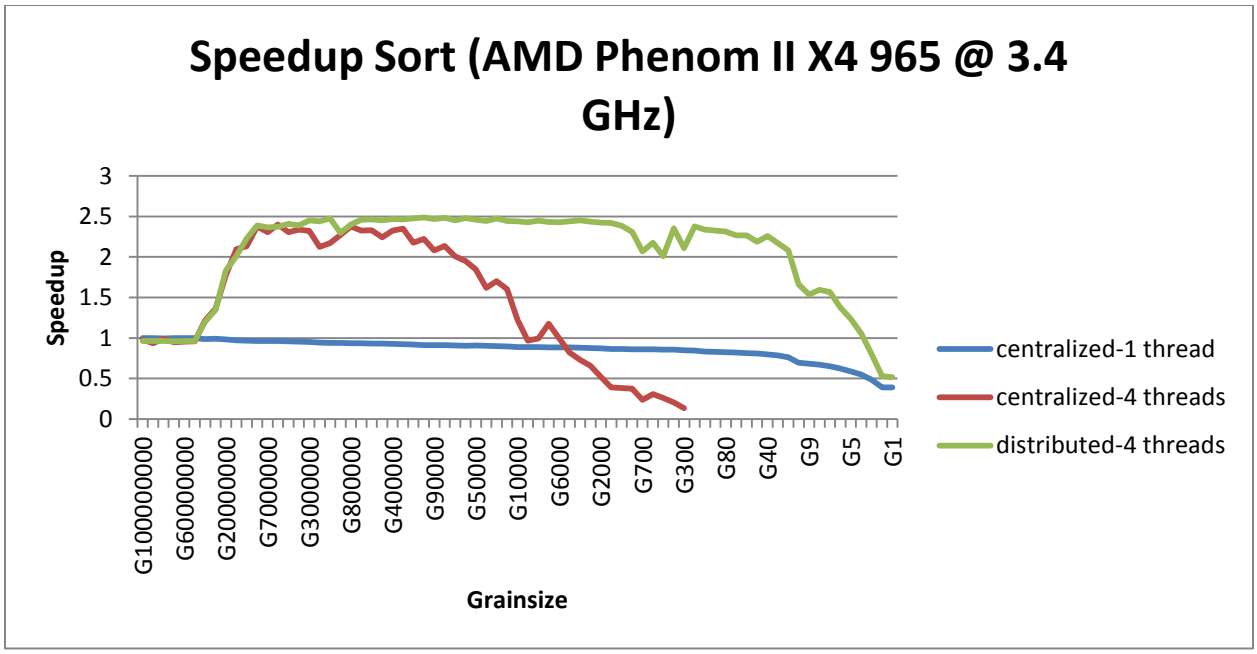
The queuing was not always the fastest in this case. For grainsizes equal or greater than a million, both distributed and centralized queuing systems were almost the same. Less than a million though, centralized queuing slowly deteriorated and then explosively became worse at about the 5000 grainsize mark. Ultimately the decentralized had the same downfall in performance, but the because of the distributed queuing it allowed us use a smaller grainsize before succumbing to enqueuing and dequeuing overheads.

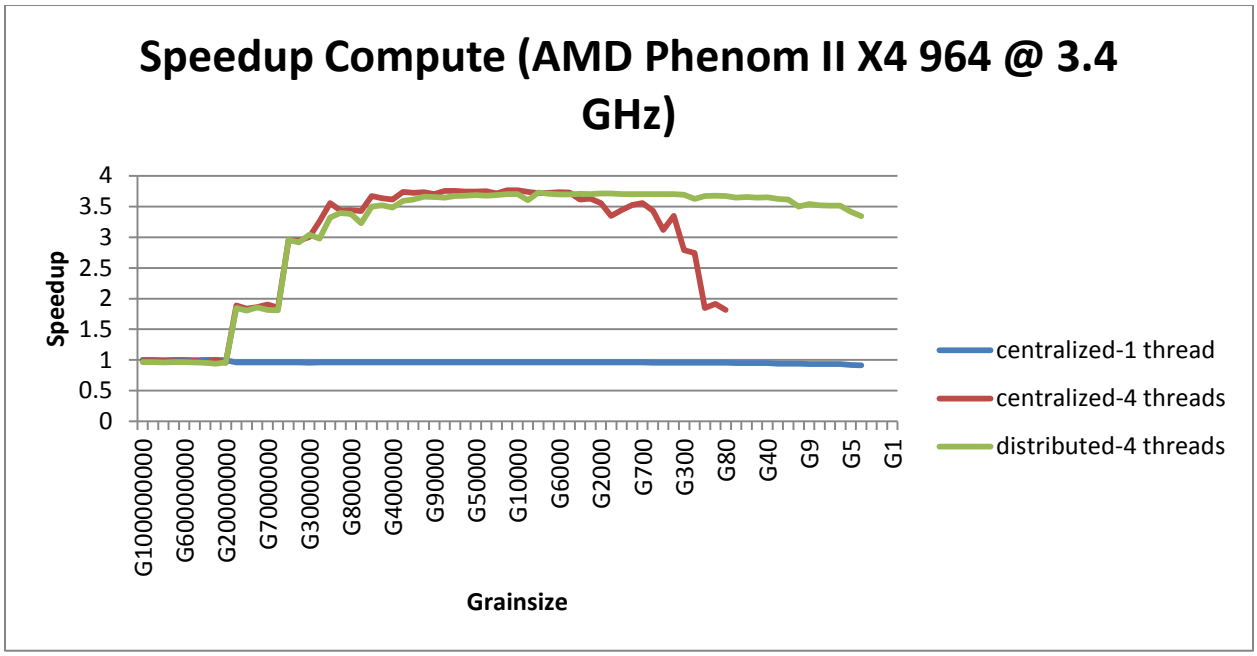
All Data Tables:

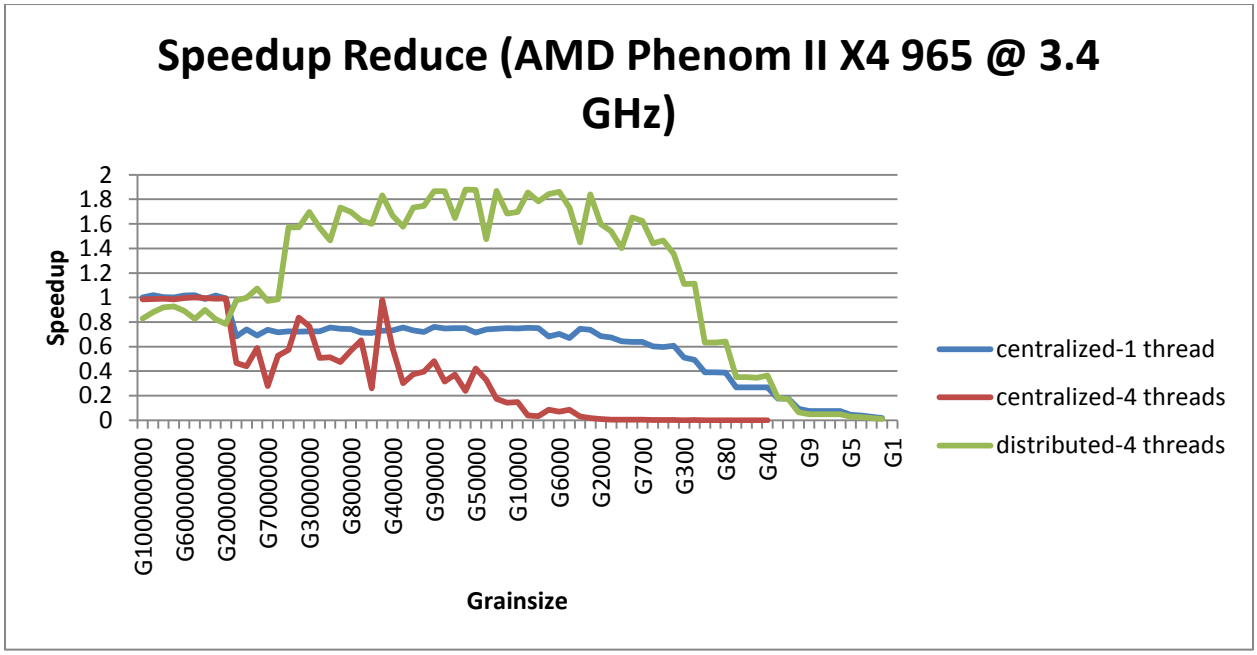


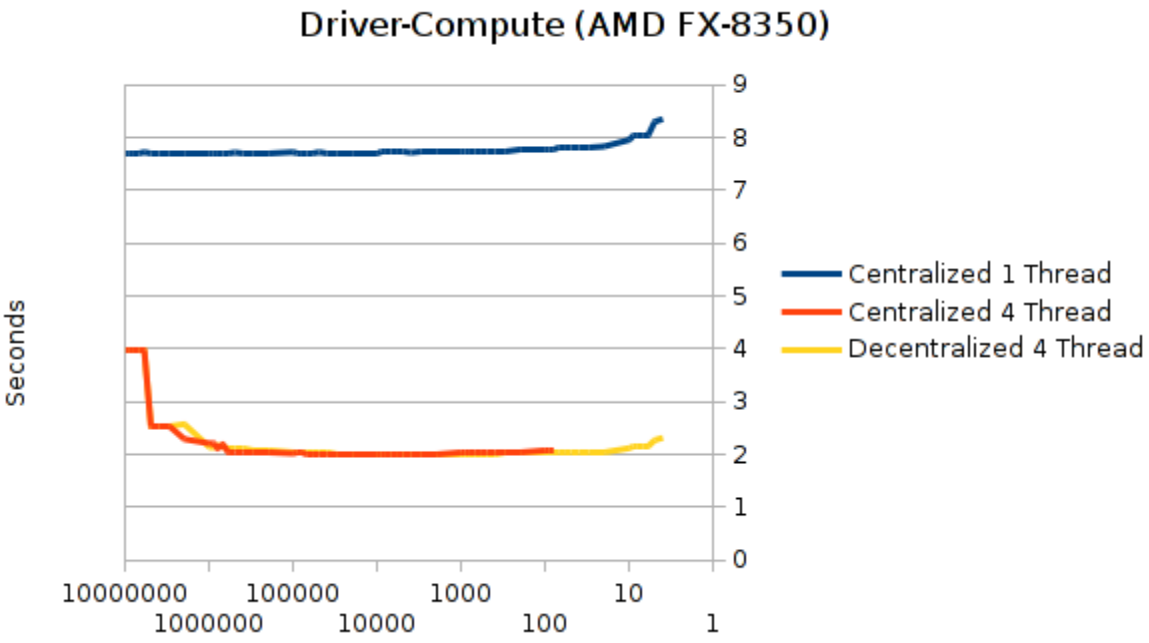


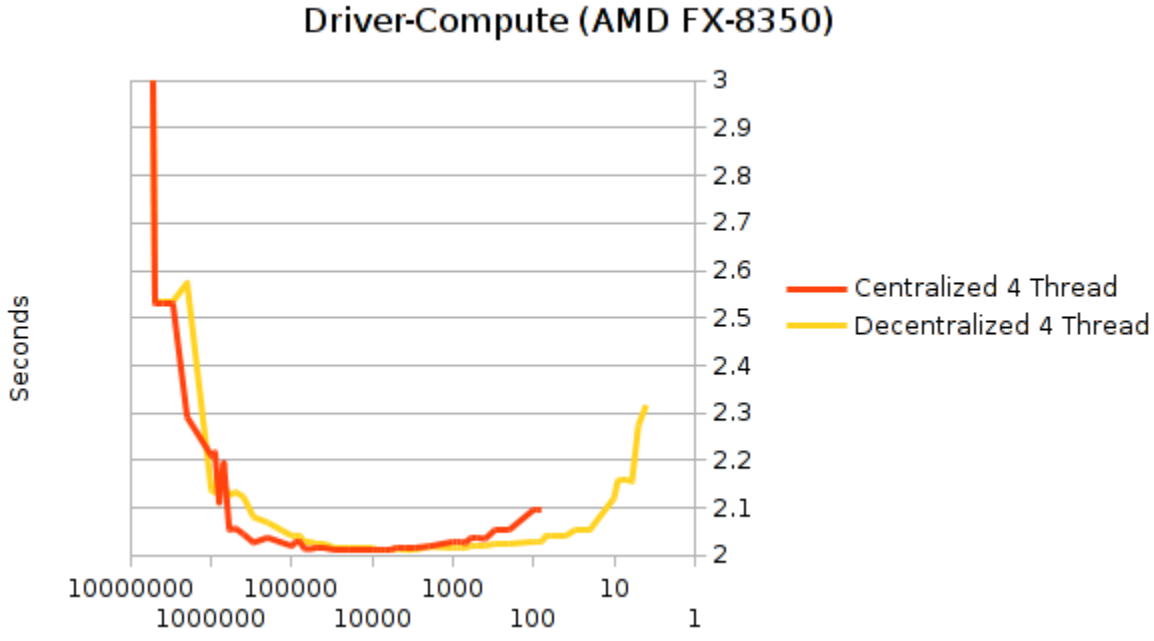

Speedup Reduce (Amoeba-N3)
Speedup
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
G100000000
G60000000
G20000000
G7000000
G3000000
G800000
G400000
G90000
G50000
G10000
G6000
G2000
G700
G300
G80
G40
G9
G5
G1
Grainsize
centralized-1 thread
centralized-4 threads
distributed-4 threads

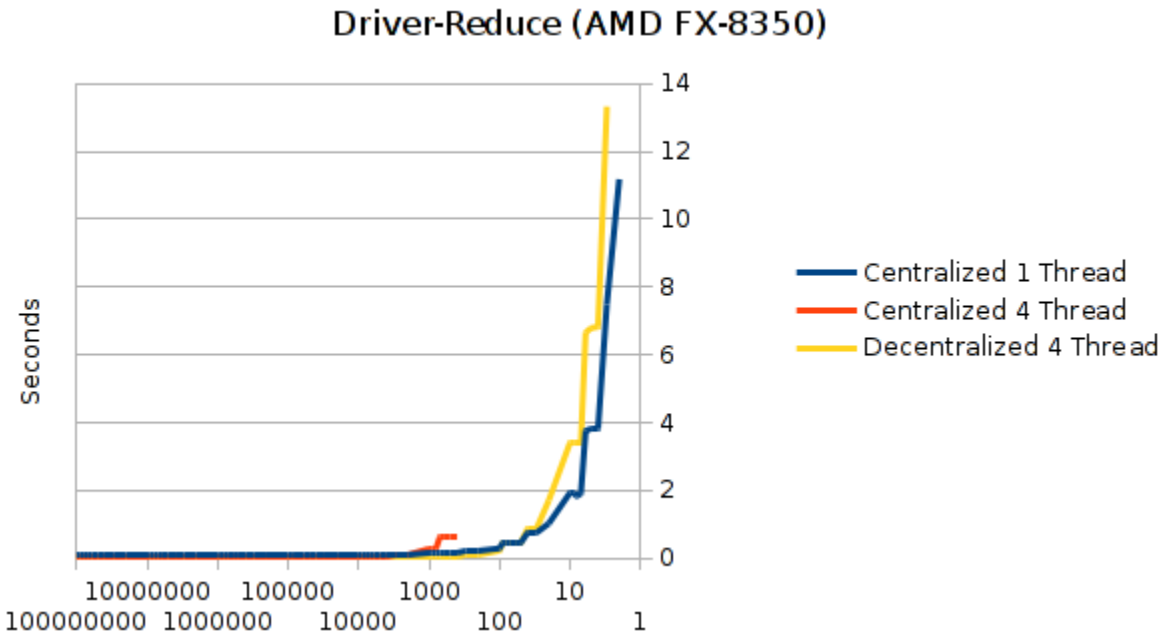

Speedup Sort (AMD Phenom II X4 965 @ 3.4 GHz)
Speedup
3
2.5
2
1.5
1
0.5
0
G100000000
G60000000
G20000000
G7000000
G3000000
G800000
G400000
G90000
G50000
G10000
G6000
G2000
G700
G300
G80
G40
G9
G5
G1
Grainsize
centralized-1 thread
centralized-4 threads
distributed-4 threads

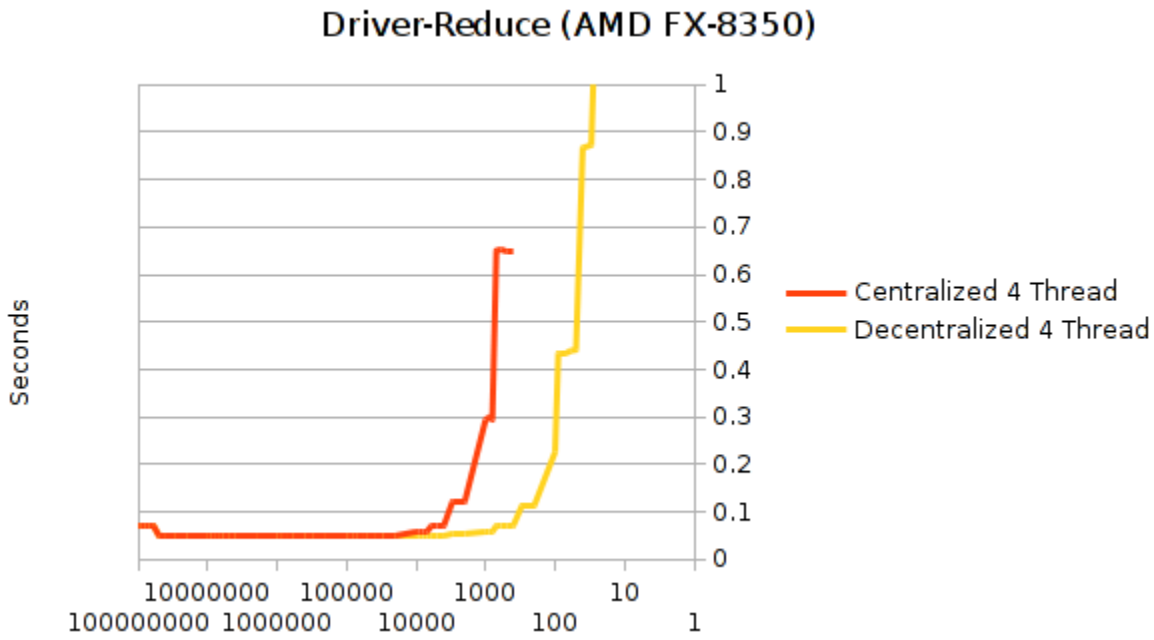

Speedup Compute (AMD Phenom II X4 964 @ 3.4 GHz)
Speedup
4
3.5
3
2.5
2
1.5
1
0.5
0
G100000000
G60000000
G20000000
G7000000
G3000000
G800000
G400000
G90000
G50000
G10000
G6000
G2000
G700
G300
G80
G40
G9
G5
G1
Grainsize
centralized-1 thread
centralized-4 threads
distributed-4 threads

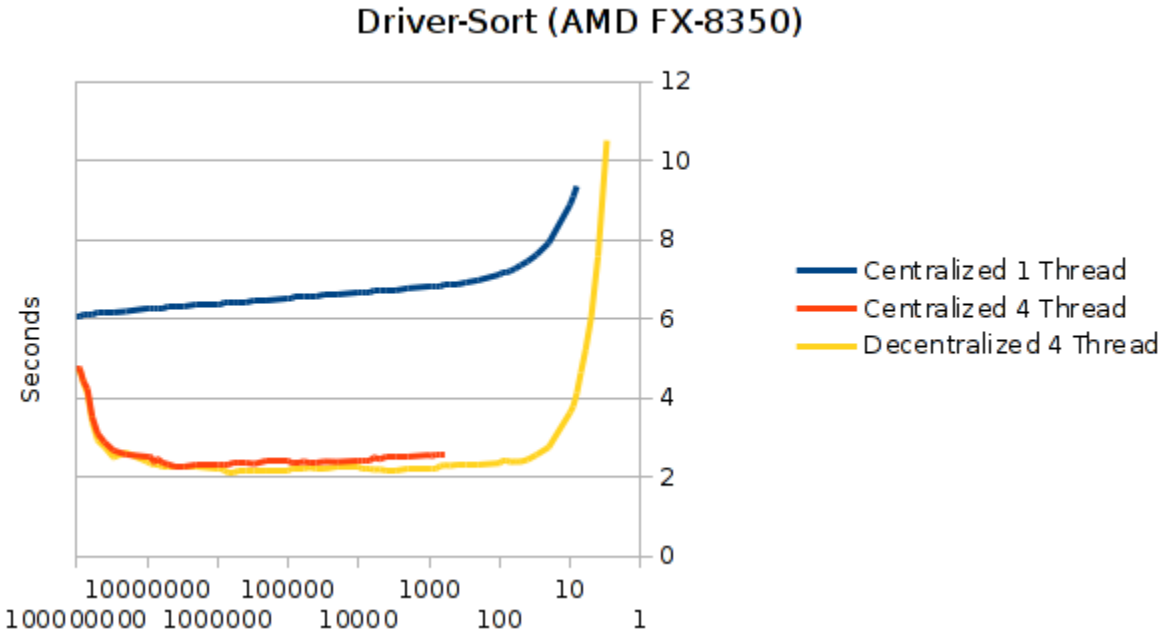

Speedup Reduce (AMD Phenom II X4 965 @ 3.4 GHz)
Speedup
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
G100000000
G60000000
G20000000
G7000000
G3000000
G800000
G400000
G90000
G50000
G10000
G6000
G2000
G700
G300
G80
G40
G9
G5
G1
Grainsize
centralized-1 thread
centralized-4 threads
distributed-4 threads

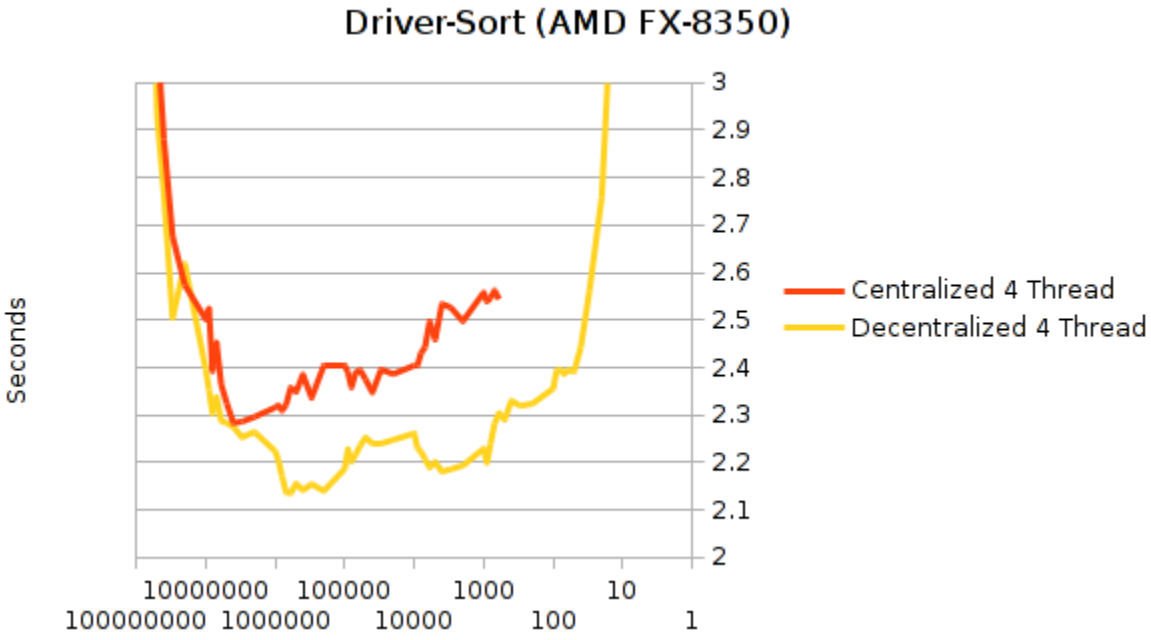

Driver-Compute (AMD FX-8350)
Seconds
9
8
7
6
5
4
3
2
1
0
10000000
1000000
100000
10000
1000
100
10
1
Centralized 1 Thread
Centralized 4 Thread
Decentralized 4 Thread


Driver-Compute (AMD FX-8350)
Seconds
3
2.9
2.8
2.7
2.6
2.5
2.4
2.3
2.2
2.1
2
10000000
1000000
100000
10000
1000
100
10
1
Centralized 4 Thread
Decentralized 4 Thread


Driver-Reduce (AMD FX-8350)
Seconds
14
12
10
8
6
4
2
0
10000000
100000
1000
10
100000000
1000000
10000
100
1
Centralized 1 Thread
Centralized 4 Thread
Decentralized 4 Thread


Driver-Reduce (AMD FX-8350)
Seconds
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10000000
100000
1000
10
100000000
1000000
10000
100
1
Centralized 4 Thread
Decentralized 4 Thread


Driver-Sort (AMD FX-8350)
Seconds
12
10
8
6
4
2
0
10000000
100000
1000
10
100000000
1000000
10000
100
1
Centralized 1 Thread
Centralized 4 Thread
Decentralized 4 Thread


Driver-Sort (AMD FX-8350)
Seconds
3
2.9
2.8
2.7
2.6
2.5
2.4
2.3
2.2
2.1
2
10000000
100000
1000
10
100000000
1000000
10000
100
1
Centralized 4 Thread
Decentralized 4 Thread