Nathan Hernandez / Juan Diego Mendez
CS377p
11/27/2023

Parallel pagerank algorithm Report

1. **Report the running time of the serial code for each input graph.**

   RMAT = Elapsed process CPU time = 8670898 nanoseconds

   ROAD-NY = Elapsed process CPU time = 121944926 nanoseconds

2. **Write a short description of your implementations to help us understand your programs.**
   1. Mutex on each node
      a. We used a pointer to an array of mutexes that was declared in the private section of the CSRGraph class. Then I dynamically allocate a separate mutex for each node in my init_sync function. Then in my mutex function when we are updating the pagerank label, we use the method lock_guard which grabs the lock for the specified mutex. Lock_guard will acquire the lock and then will automatically release the lock when it goes out of scope. After it gains the lock it performs the label update. This prevents multiple threads from concurrently updating the pagerank label for the same node. After the program is done a destructor is called to free the array of mutexes.
   2. Spin-lock on each node
      a. We used a pointer to an array of atomic_flags which we named spin_lock. We also make this a private member in the CSRGraph class. Then we dynamically allocate the array with each node receiving a spin_lock. Atomic_flags are an atomic boolean that we are using to make a simple spin_lock. Then in the same method we clear each lock to make sure each spin_lock starts in an unlocked state. Then we made a method to lock the node with a test and set it with a while loop. The while loop will be busy waiting and trying to set the flag repeatedly. Test and set will return the previous value, which is useful as it will keep trying till it sets the flag. Then we made a method to unlock the node with clear which released the atomic_flag. Then with the spin_lock method we call the lock the spin_lock, then set the label, then finally unlock the spin_lock. After the program is done a destructor is called to free the array of atomic_flags.
   3. Compare and swap
      a. We used a pointer to an array of atomic doubles that is declared as a private member of the CSR Graph class. Then in the init_sync method we dynamically allocated an atomic variable for each node. With an atomic load we load the current value and we use that to compare with the new value which is the load plus our contribution. We do this comparison with the method compare_exchnage_weak and the loop continues until the

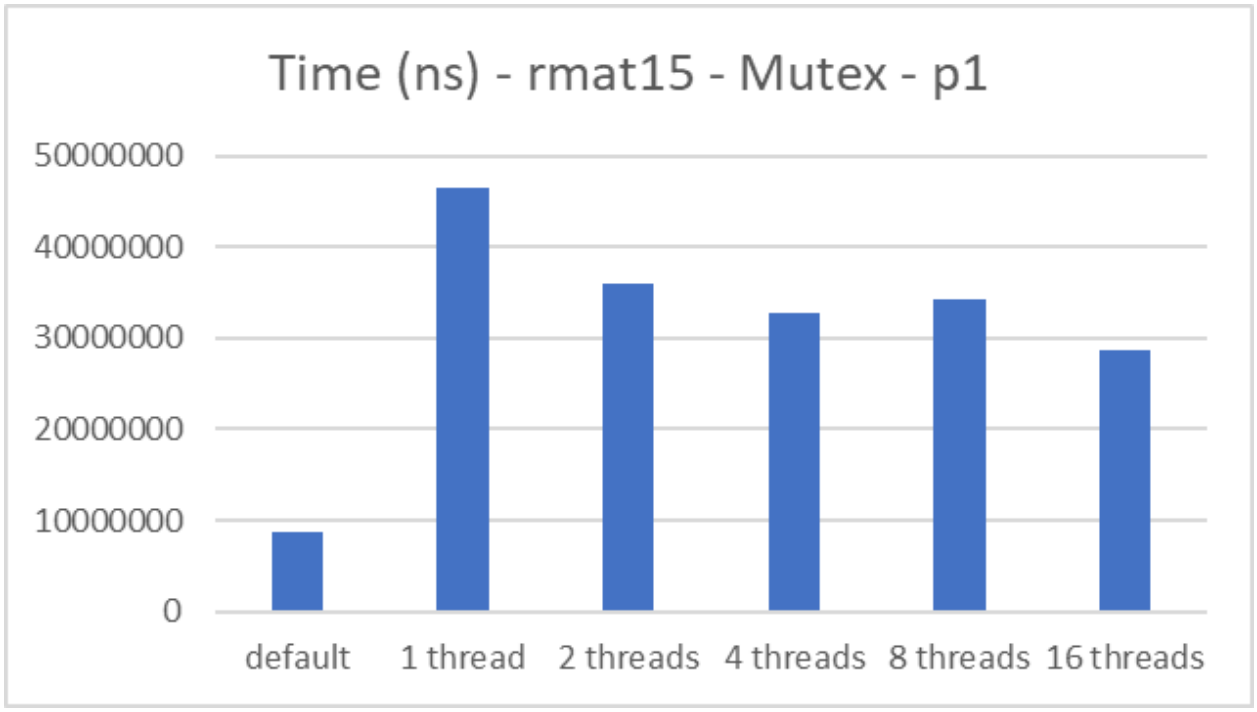
comparison and update is successful, then we set the label. We also make sure that the new value is actually a change from the old value with the condition statement. After the program is done a destructor is called to free the array of atomic doubles.

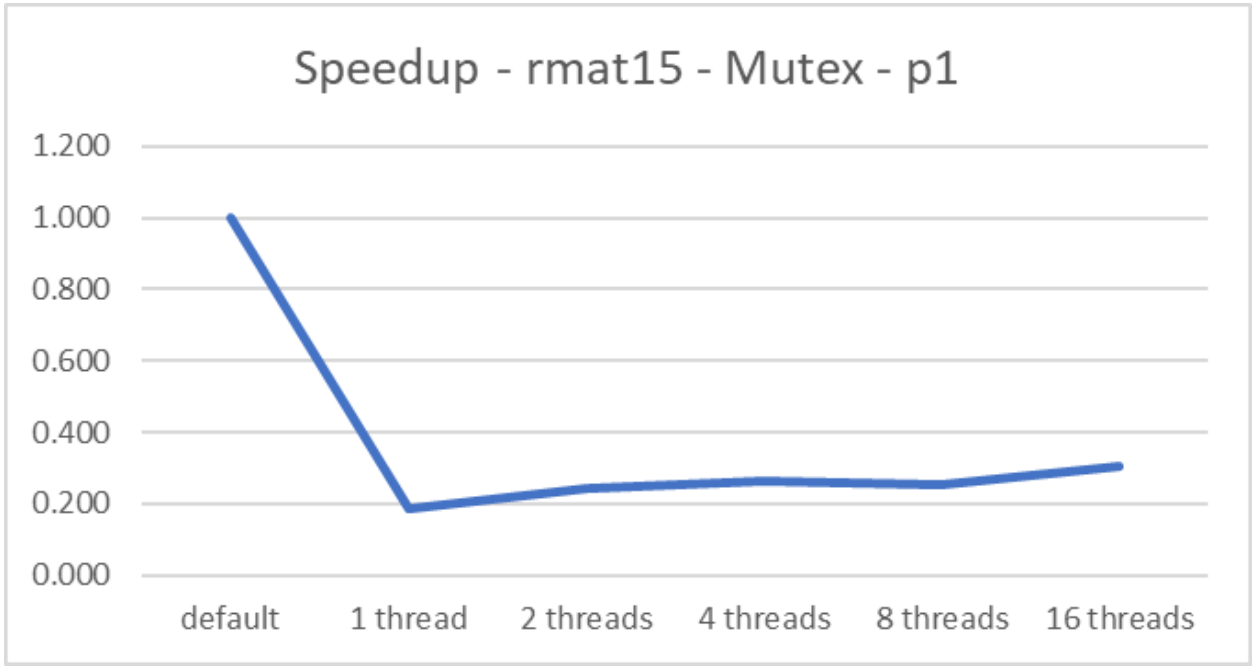
4. Section 2 Compare and Swap.
    a. We first initialize the atomic variables. Then we calculate the edges that each thread will receive. Then we have a vector pair to store the ranges of edges for each thread. Then the loop computes the start and end of the indices for the ranges of the edges that thread was assigned. Then it is stored in the vector. Then we use omp to start the critical section with the number of threads. Then each thread performs a binary search that finds the range of the vertices it is assigned to based on the ranges of edges. Then after that it computes the contribution based on the current pagerank and the out-degree of the source vertex. Then the threads iterates over their assigned edges and updates the pagerank using the compare and swap method. After the program is done a destructor is called to free the array of atomic doubles.

3. Report the running times and speedups for 1,2,4,8,16 threads for rmat15 and road-NY (baselines for speedup are the times for your serial code) using the three ways of implementing atomic updates discussed above. Graph the running times and speedups for each input graph as a function of the number of threads. Based on these experiments, what is the best way to implement the atomic updates for pagerank?

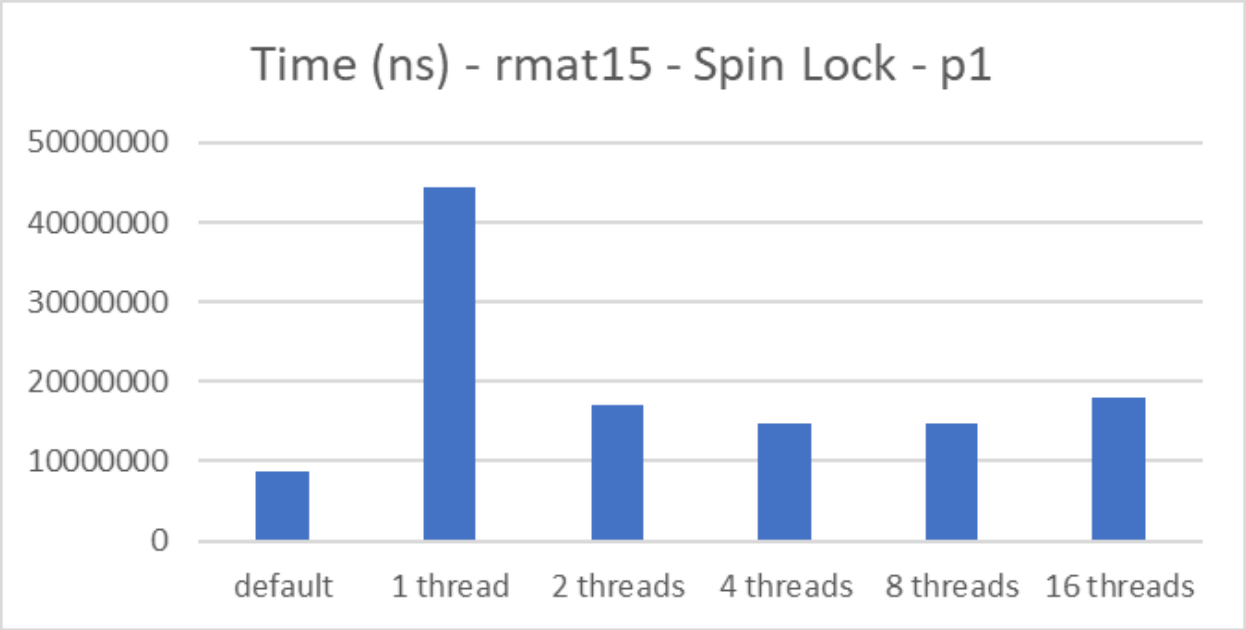
RMAT15

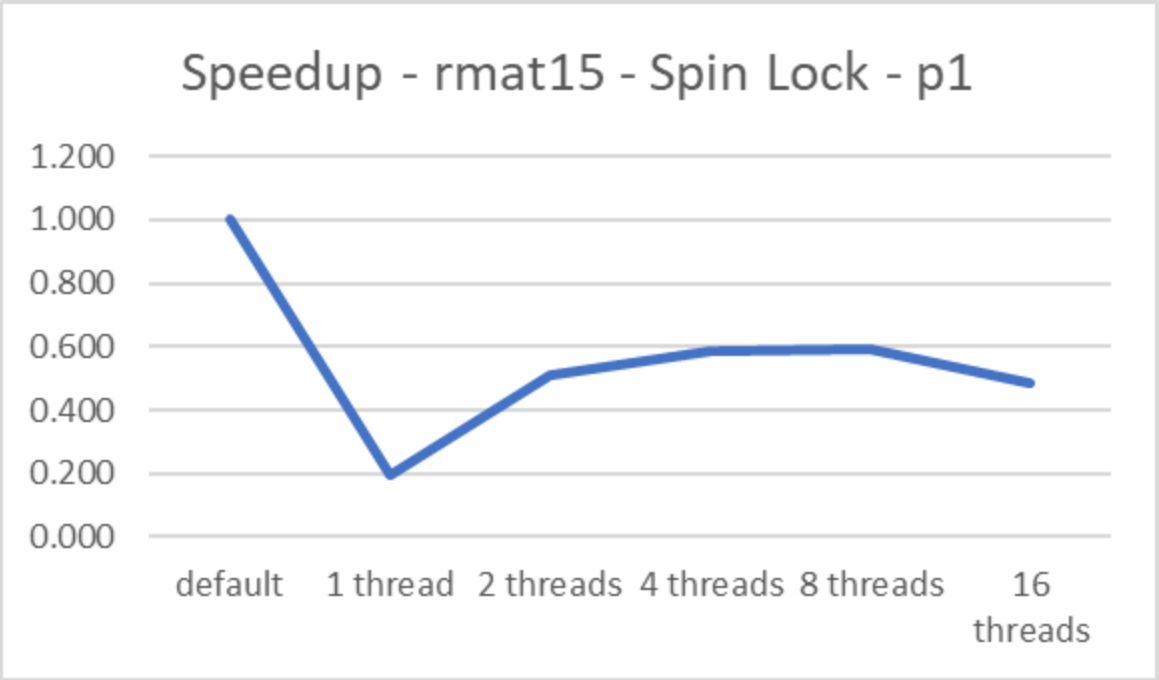
P1 - rmat15 - Mutex	Time (ns)	Speedup
default	8670898	1.000
1 thread	46530391	0.186
2 threads	35924083	0.241
4 threads	32866068	0.264
8 threads	34344796	0.252
16 threads	28627378	0.303


Time (ns) - rmat15 - Mutex - p1
50000000
40000000
30000000
20000000
10000000
0
default
1 thread
2 threads
4 threads
8 threads
16 threads

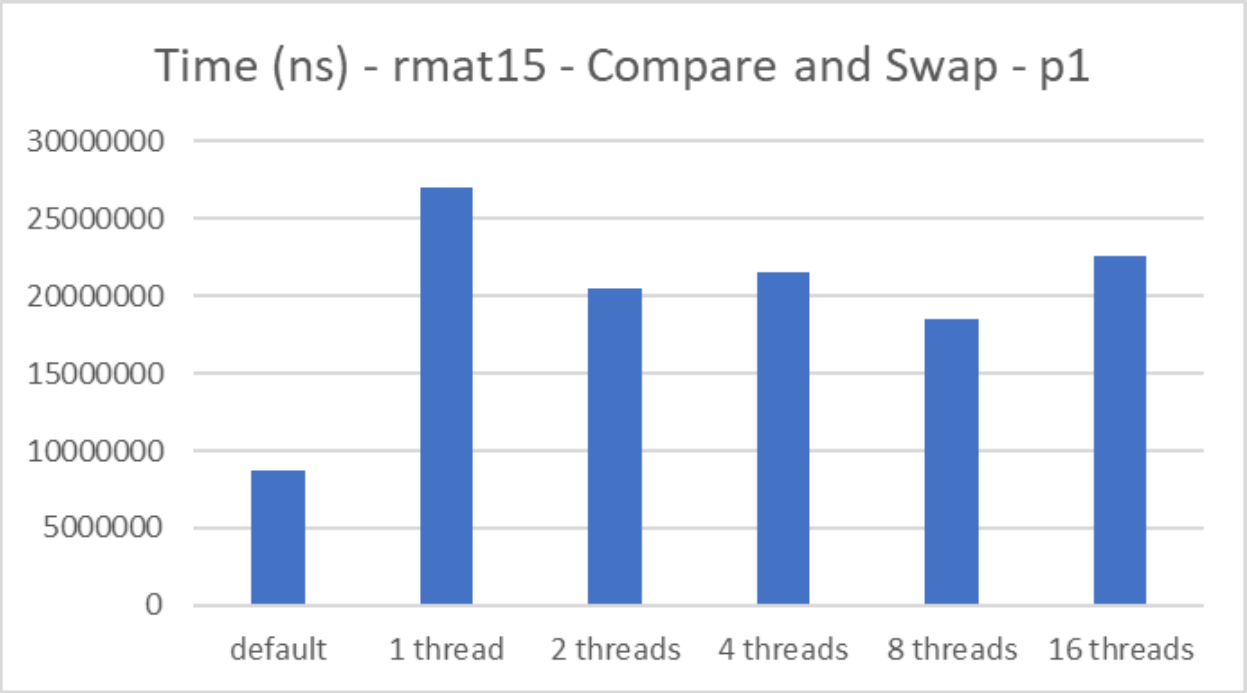

Speedup - rmat15 - Mutex - p1
1.200
1.000
0.800
0.600
0.400
0.200
0.000
default
1 thread
2 threads
4 threads
8 threads
16 threads

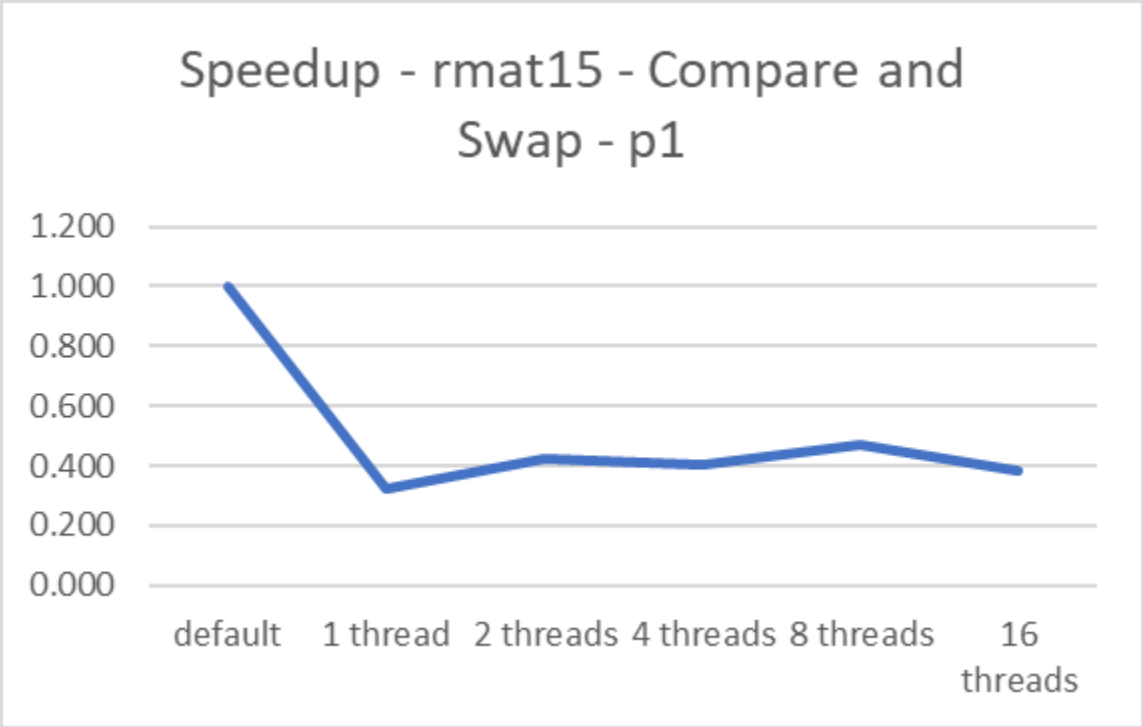
P1 - rmat15 - Spin Lock	Time (ns)	Speedup
default	8670898	1.000
1 thread	44475084	0.195
2 threads	17073179	0.508
4 threads	14837556	0.584
8 threads	14669020	0.591
16 threads	17976841	0.482





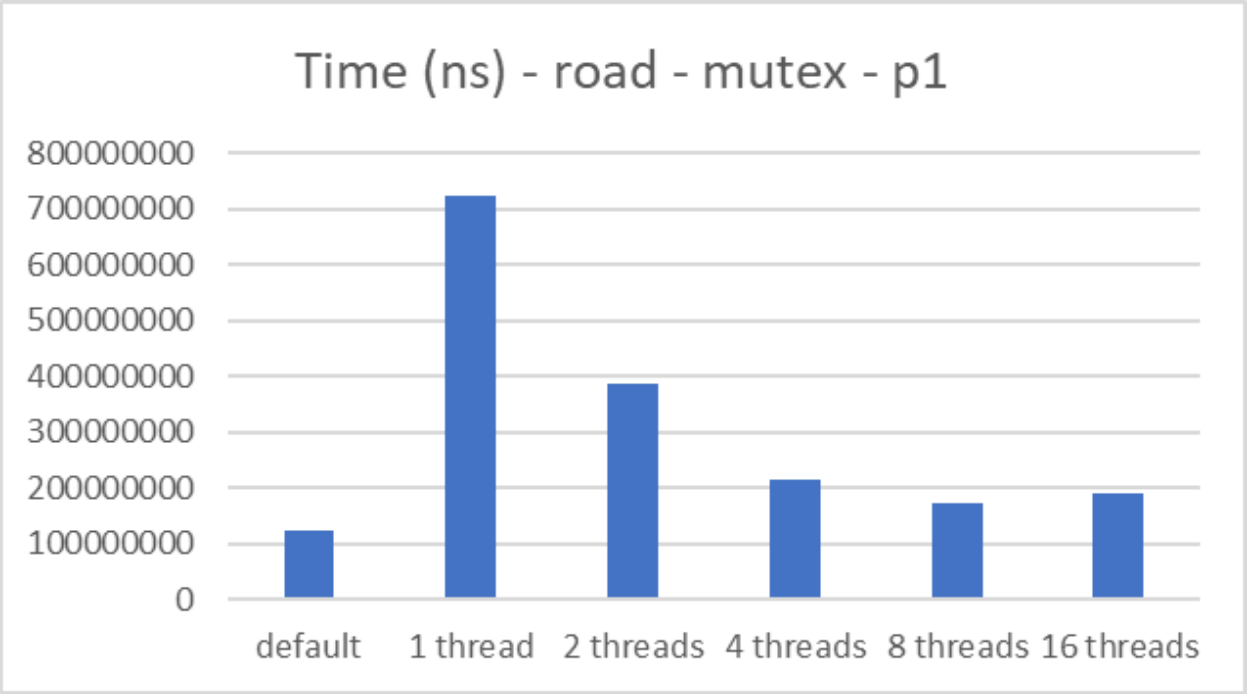
P1 - rmat15 - Compare and Swap	Time (ns)	Speedup
default	8670898	1.000
1 thread	27033660	0.321
2 threads	20529281	0.422
4 threads	21549899	0.402
8 threads	18535969	0.468
16 threads	22618500	0.383

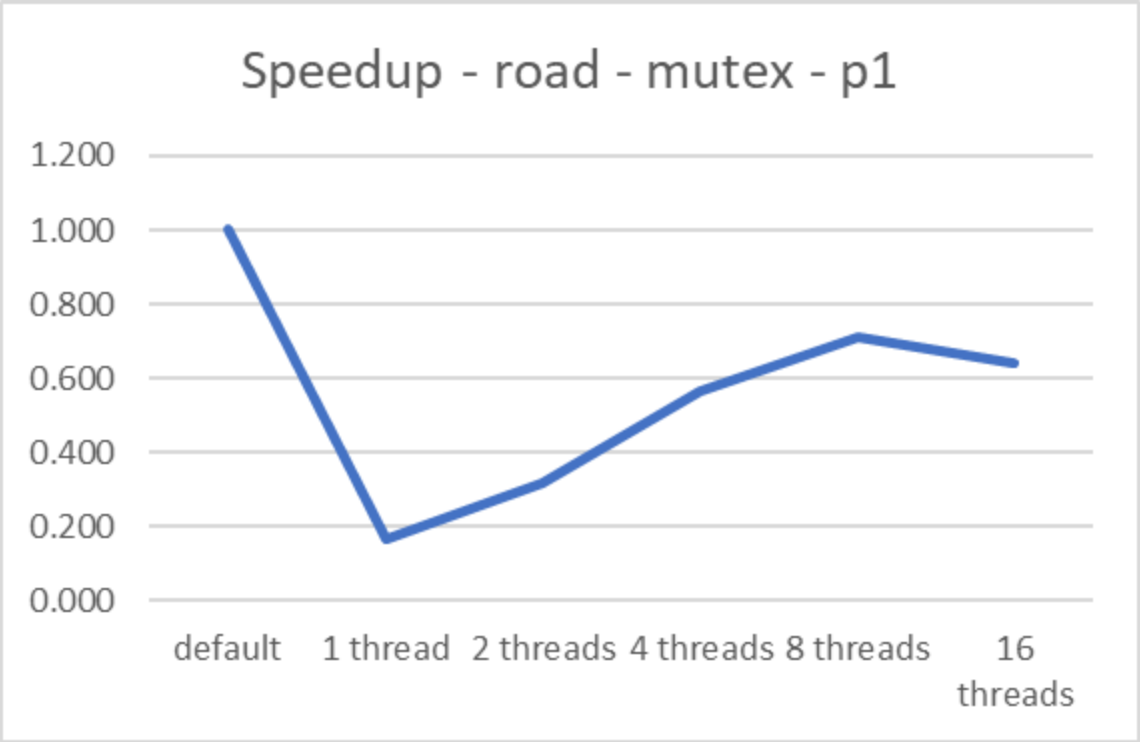

Time (ns) - rmat15 - Compare and Swap - p1
30000000
25000000
20000000
15000000
10000000
5000000
0
default
1 thread
2 threads
4 threads
8 threads
16 threads


Speedup - rmat15 - Compare and Swap - p1
1.200
1.000
0.800
0.600
0.400
0.200
0.000
default
1 thread
2 threads
4 threads
8 threads
16 threads

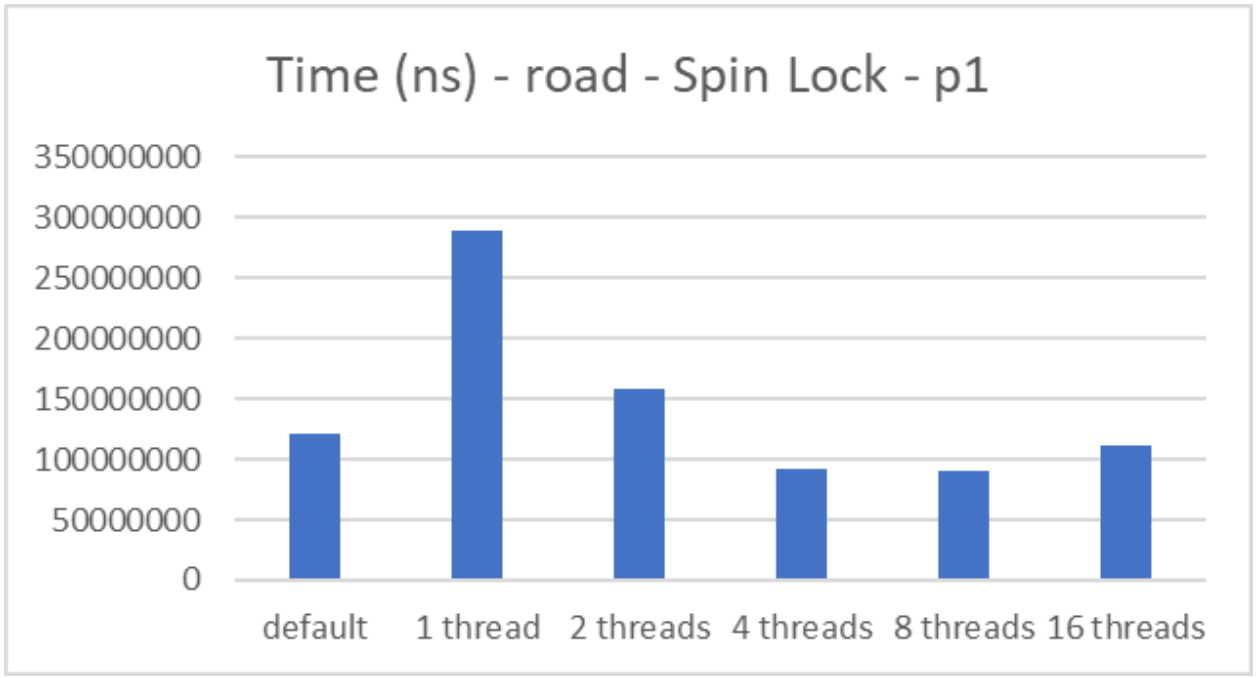
ROAD-NY

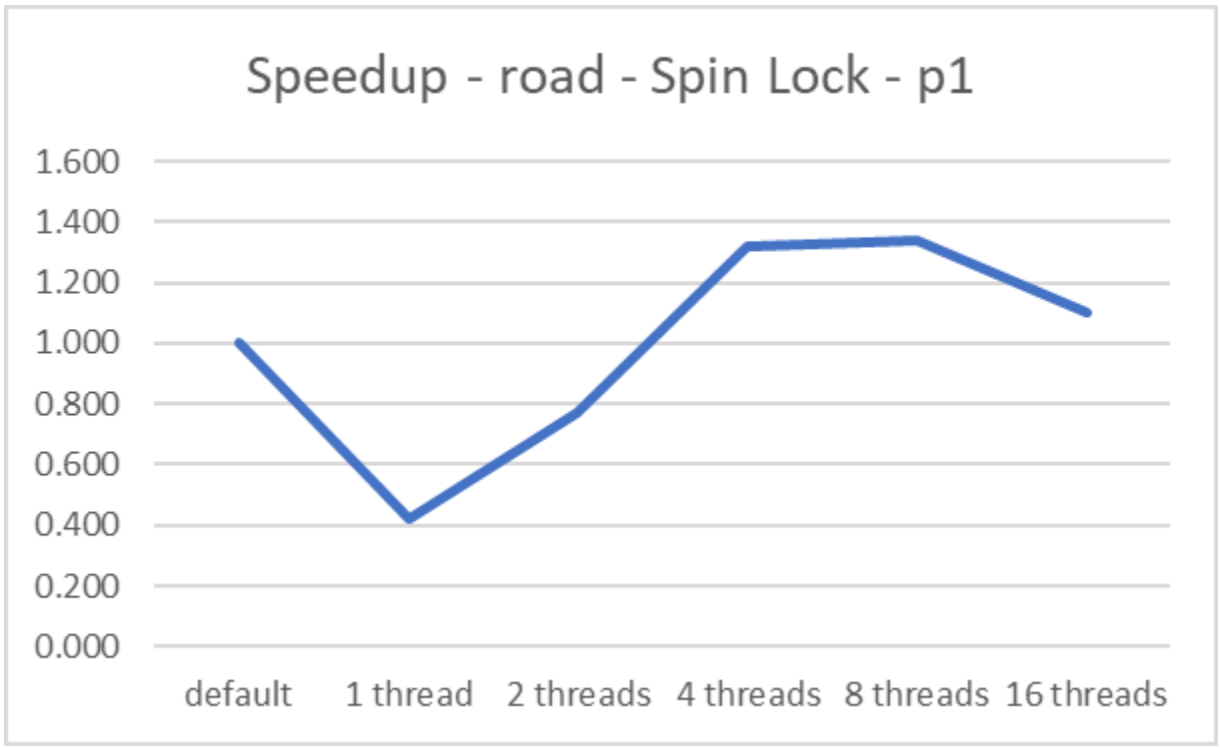
P1 -road - Mutex	Time (ns)	Speedup
default	121944926	1.000
1 thread	725909278	0.168
2 threads	386715978	0.315
4 threads	214765379	0.568
8 threads	171717483	0.710
16 threads	190872673	0.639



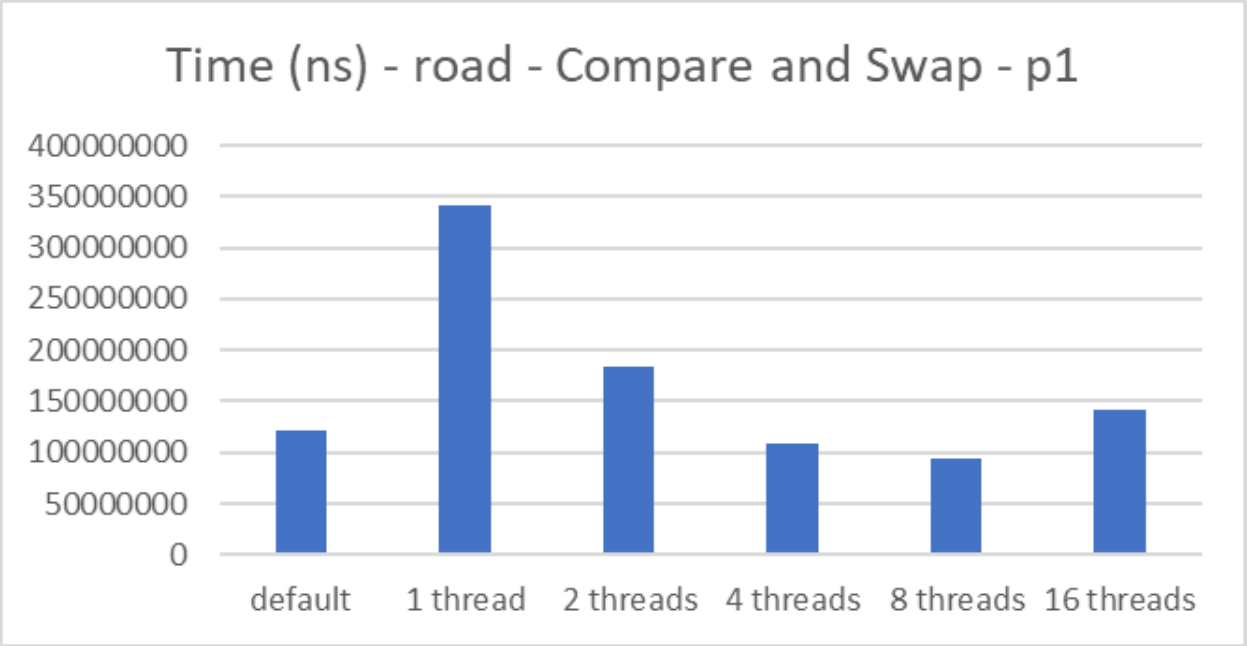


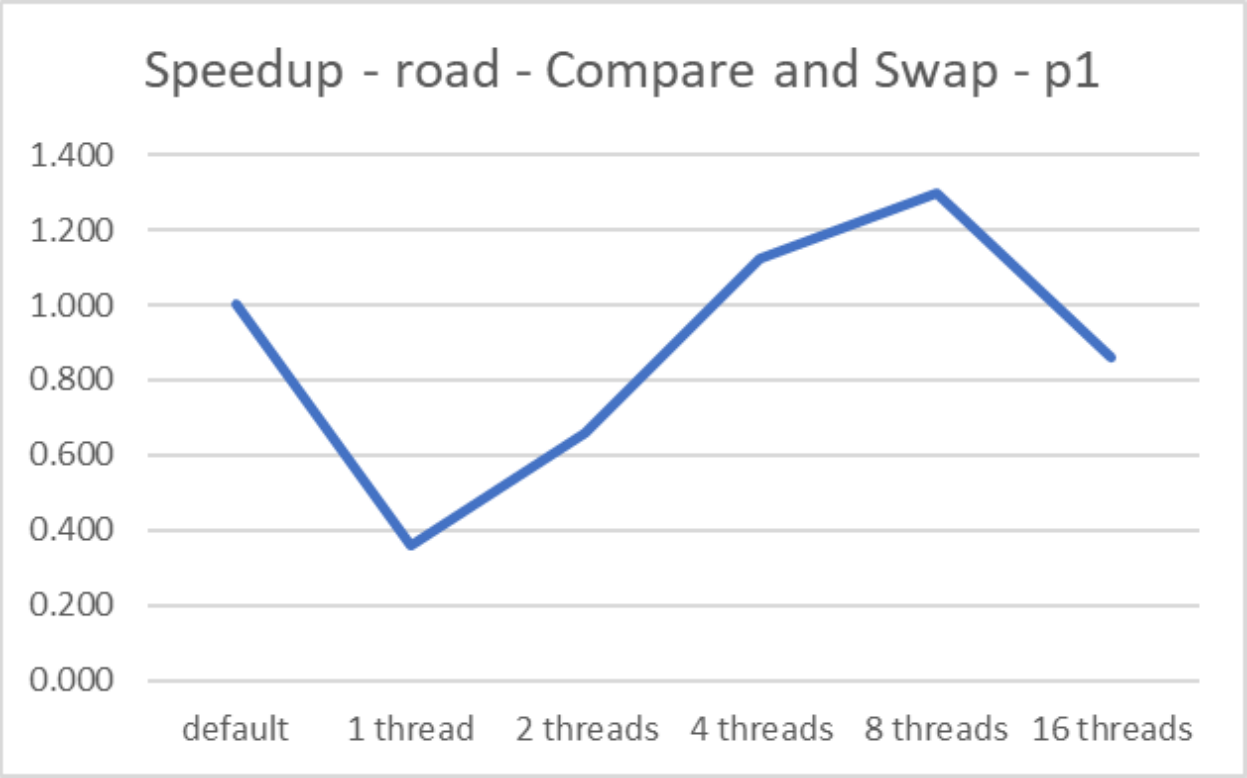
P1 - road - Spin Lock	Time (ns)	Speedup
default	121944926	1.000
1 thread	289627482	0.421
2 threads	157720987	0.773
4 threads	92273730	1.322
8 threads	91181396	1.337
16 threads	110875849	1.100


Time (ns) - road - Spin Lock - p1
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
default
1 thread
2 threads
4 threads
8 threads
16 threads


Speedup - road - Spin Lock - p1
1.600
1.400
1.200
1.000
0.800
0.600
0.400
0.200
0.000
default
1 thread
2 threads
4 threads
8 threads
16 threads

P1 - road - Compare and Swap	Time (ns)	Speedup
default	121944926	1.000
1 thread	340802456	0.358
2 threads	184532768	0.661
4 threads	108705576	1.122
8 threads	93786983	1.300
16 threads	141822357	0.860





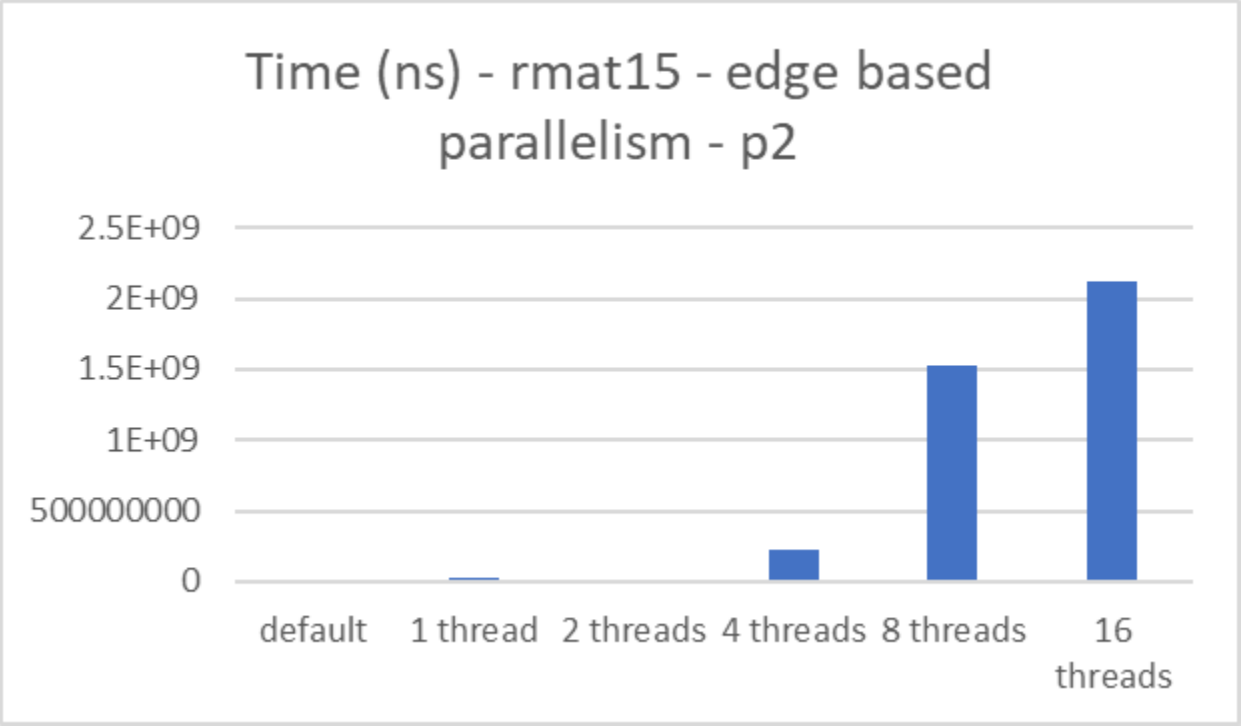
Which one is the best:

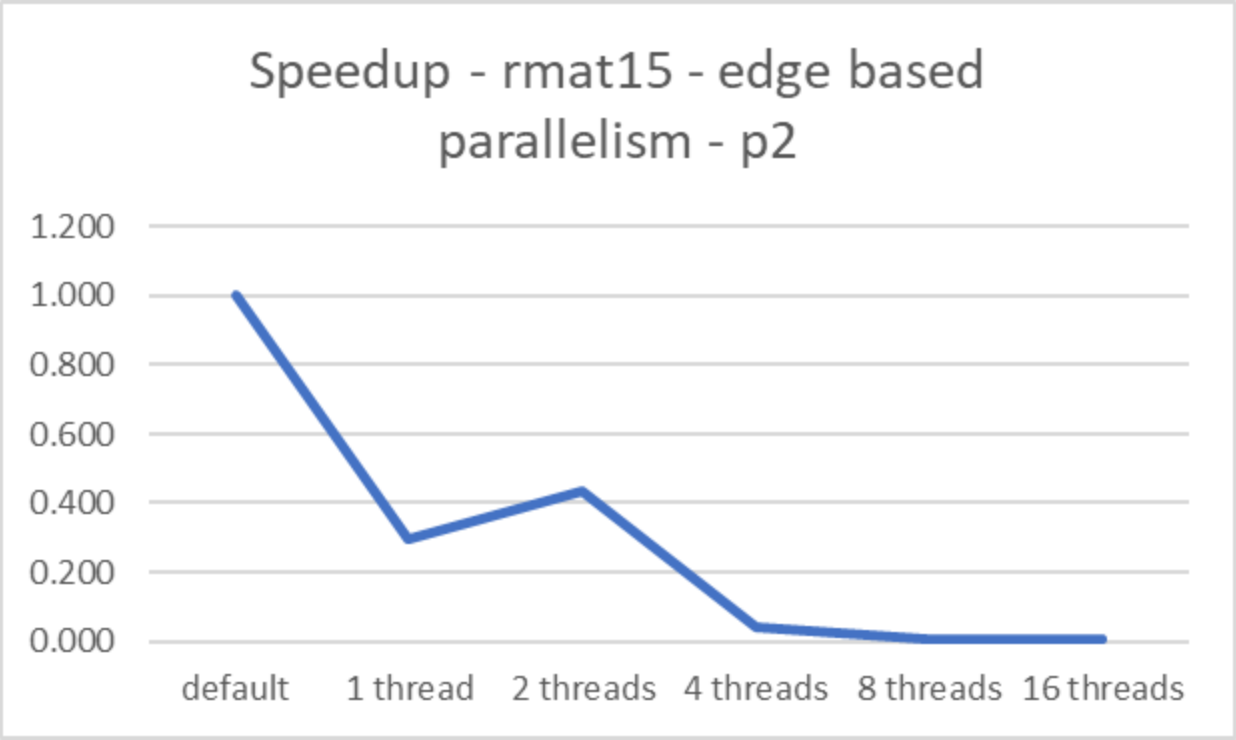
1. The best one for both rmat15 and road-ny is the spin lock. I believe this is due to the spin_locks advantage of having a short critical section. Spin_locks also avoids the context switching that mutexes have. I think CAS did so poorly because of the heavy overhead that came with the atomic operations not being worth it for the small critical sections. With the graphs for Road-NY we can see that spin-locks are more consistently at near the 1.4 mark compared to the compare and swap, with both spin_locks and CAS dropping at 16 threads. Mutexes also do not even go over 1 on the speed up graph, however the performance does improve until 16 threads. The graphs for rmat-15 show that all of them are terrible; however, the spin_locks are the most consistent and higher with it hovering around .6 before dropping off at 16 threads.

4. Repeat part 2 with the edge-based assignment of work to threads. Do you get better performance?

RMAT15

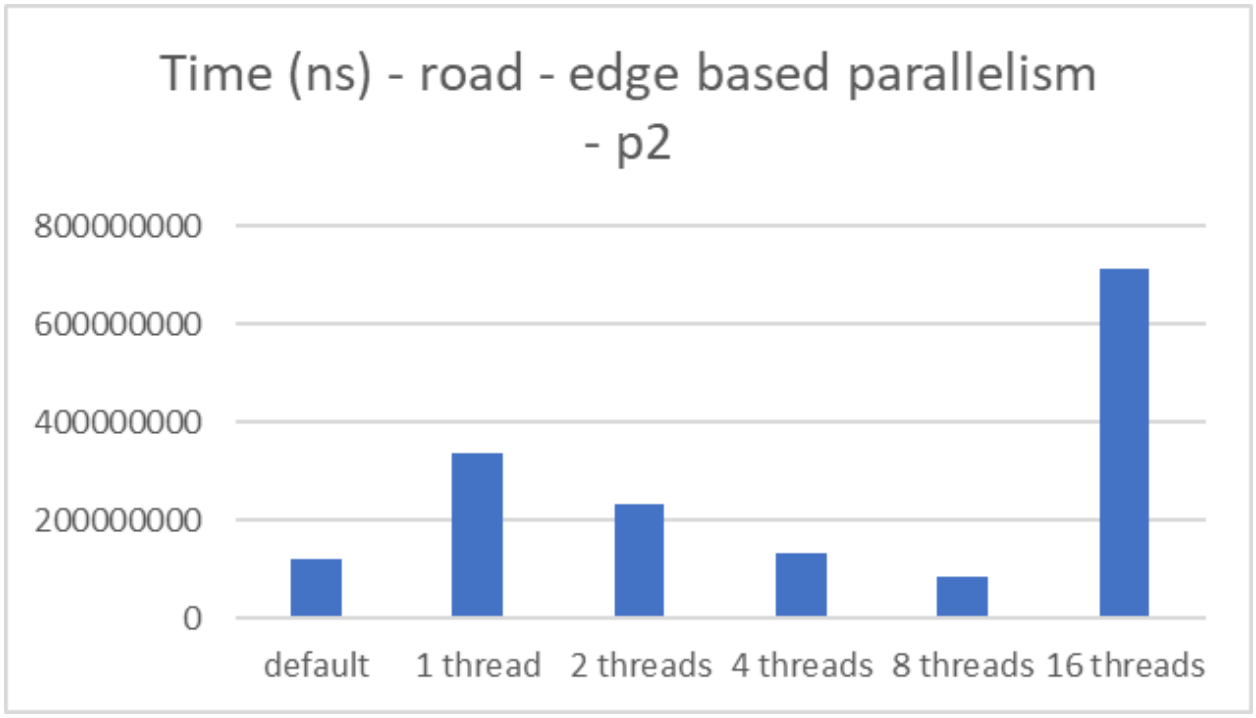
P2 - rmat15 - CaS	Time (ns)	Speedup
default	8670898	1.000
1 thread	29079583	0.298
2 threads	20047083	0.433
4 threads	223922480	0.039
8 threads	1521795241	0.006
16 threads	2114303363	0.004

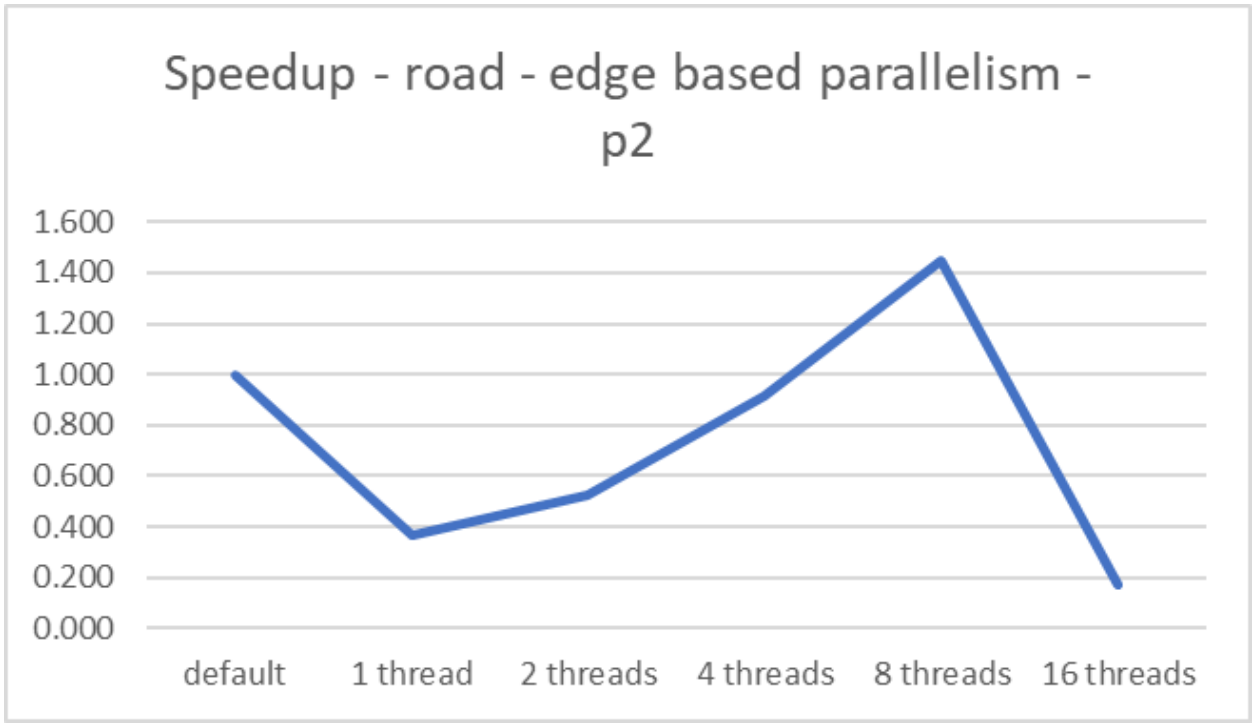




ROAD-NY

P2 - road - CaS	Time (ns)	Speedup
default	121944926	1.000
1 thread	336422641	0.362
2 threads	233369172	0.523
4 threads	132870255	0.918
8 threads	84406056	1.445
16 threads	711901480	0.171


Time (ns) - road - edge based parallelism - p2
800000000
600000000
400000000
200000000
0
default
1 thread
2 threads
4 threads
8 threads
16 threads


Speedup - road - edge based parallelism - p2
1.600
1.400
1.200
1.000
0.800
0.600
0.400
0.200
0.000
default
1 thread
2 threads
4 threads
8 threads
16 threads

Do you get Better Performance:

1. For rmat15 the performance does not improve with the edge based assignment of work. From thread one to thread two the performance increased a little bit but after that it dropped in performance. The performance for road-ny did improve with the edge based assignment. With the addition of each thread performance increased even past the performance of the spin_lock. However at 16 threads performance took a nose dive.