Alexander Macri

What can you conclude about how grid size, number of threads, and locking granularity affect performance?

Larger grid makes better performance because locks are less likely to wait on each other for the resource. Smaller grid size makes lock wait for each other and also may cause more collision errors in resource management. More threads increase time but only with bigger grids and granularity that allows multiple resources to be acted upon at the same time. Fewer threads cause collision with resources and more wit time for threads to unlock their resources. Grid granularity causes less performance since each thread has to with for resource in grid even though that may not need the same cell or row. Row granularity increase performance but cell granularity allows the thread to interact directly with the resource being changed which allows multiple thread to interact with the grid at the same time.

What general results can you conclude from this exercise? When does fine granularity locking have the biggest effect?
What costs does it impose?

Threads are useful there is multiple that can interact separately on a task or function that cane be taken place in parallel instead of consecutively. When a lot of threads want to change the same value stored or interacts with the same resource. Fine granularity allows so the system has more control over the information being changed but cost time in the threads interacting with the given data.

Experiment with no locking. Without any modifications, you shouldn't get any integrity violation with 1 thread. Why? With a big enough grid and few enough threads, you might avoid data integrity violations. Why?

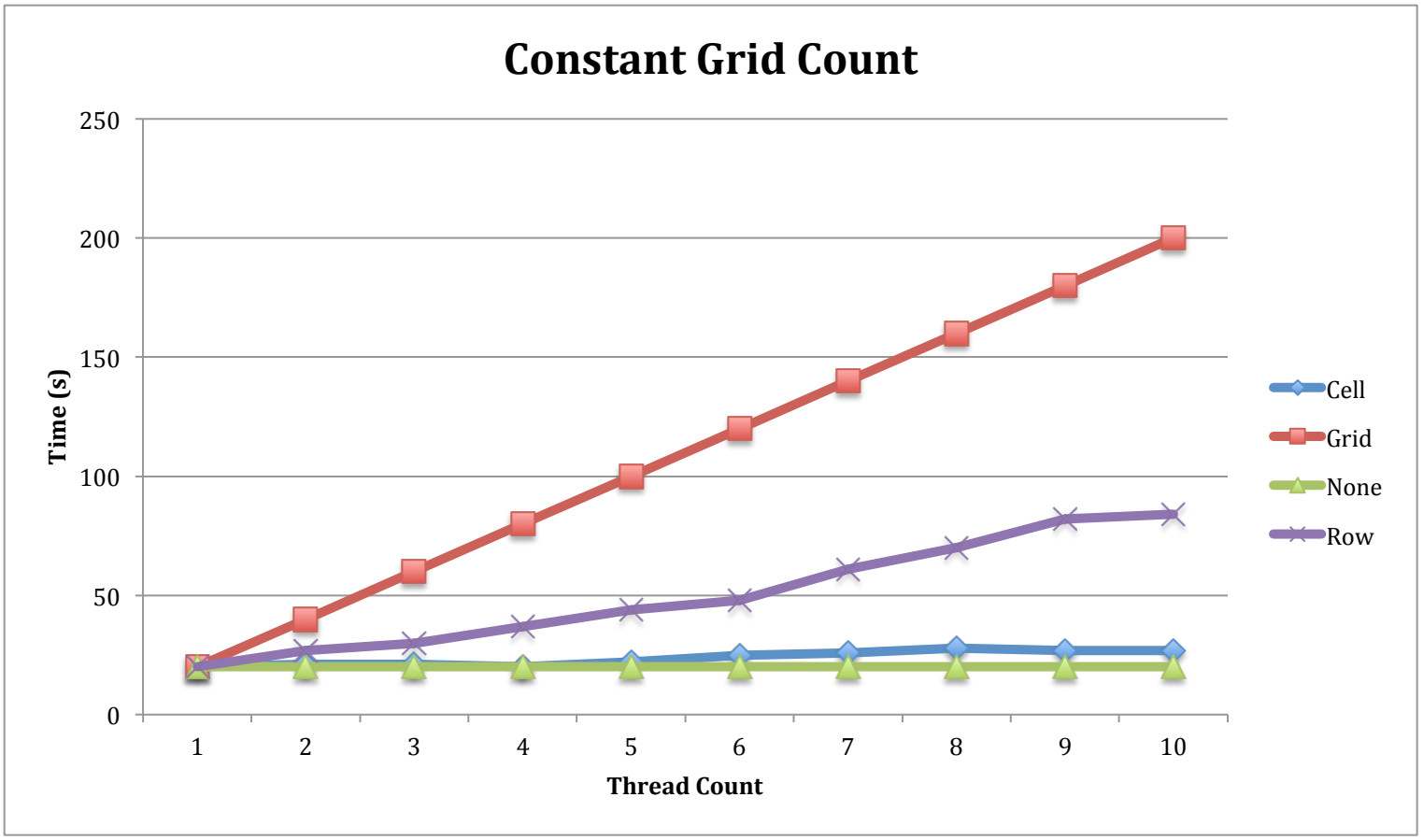
No violation with one thread is because there is only one operation taken place on the given data. Bigger grids allow threads more freedom to interact with the cell with any conflicts thus sometime giving the illusion that the threads are protecting the correctness of the data after there finished executing.

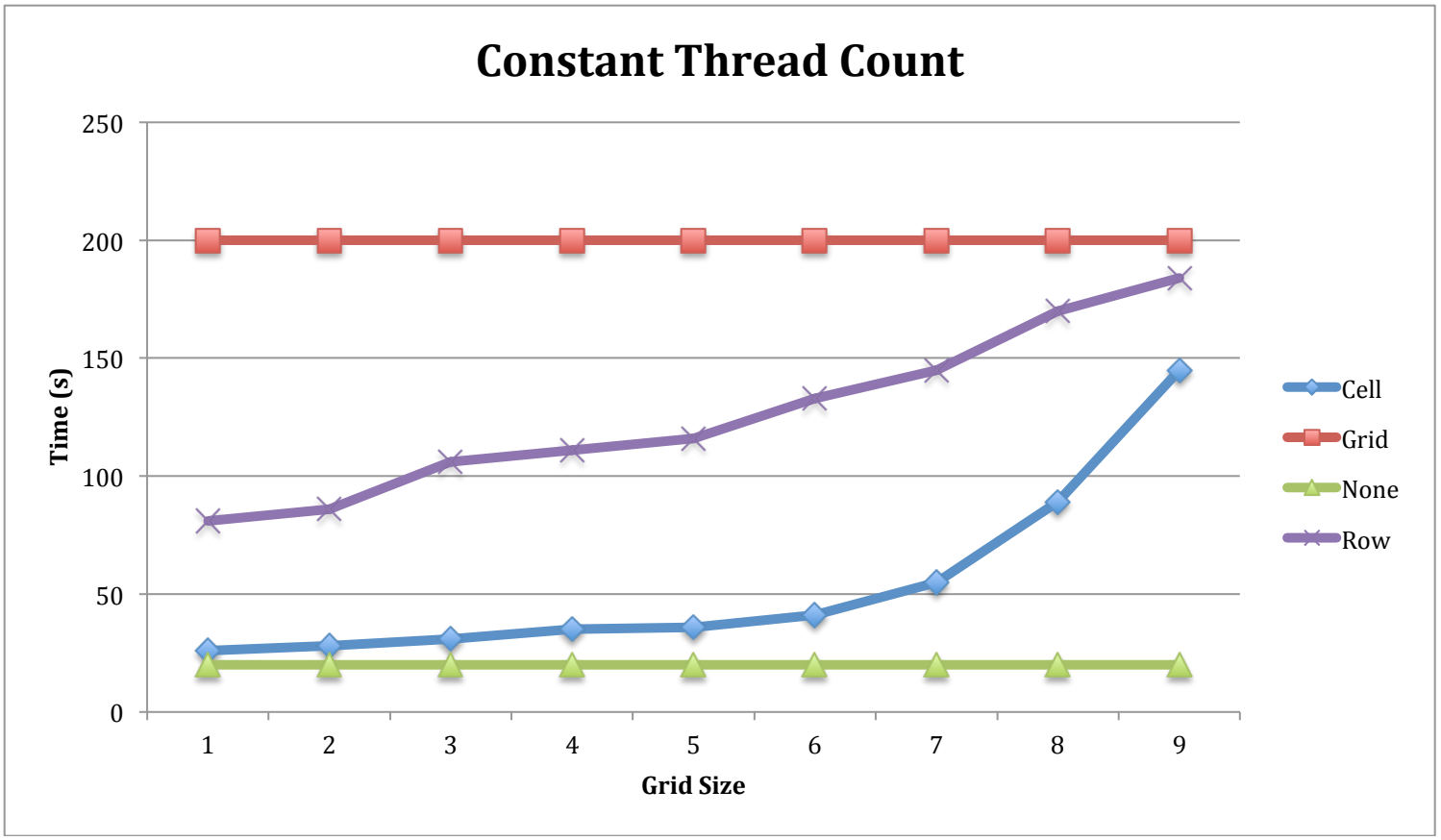
Why do you think it is there? Try commenting it out or moving it to a different location in the block of swapping code.
Sleep is there to slow down the thread interaction so the process of locking can be effective. Also to held the seed generator get new values so the randomizing of the grid swapping can take place.

Try moving it to just before the first line of the swapping code but after you have obtained your lock.
What effects does this have on performance and correctness of final sum? Try experimenting with all levels of granularity on a 10x10 grid with 10 threads.
Anything surprising?

The time elapse of the threads is reduced significantly and the effectiveness of the locks is diminished.

What can you say about why this sleep(1) is there and where it is placed?
To keep the swapping operation concurrent and make the thread process the transactions of the function.
If we didn't have this sleep(1) there, how would this change the assignment?
The program would run faster and the grid would not swap correctly usually resulting in a grid very mush the same as the initial grid.
What does this say about when multithreading and synchronization make sense?
That the time it takes for an operation has a lot do with the synchronization of thread.
Not only do you have to lock but also have to make sure the locks accurately stop other thread from using the same resource.


Constant Grid Count
250
200
150
100
50
0
Time (s)
1
2
3
4
5
6
7
8
9
10
Thread Count
Cell
Grid
None
Row


Constant Thread Count
250
200
150
100
50
0
Time (s)
1
2
3
4
5
6
7
8
9
Grid Size
Cell
Grid
None
Row