Benjamin Mizera

Homework 2

For this Homework, added on some message passing functions to Homework 1 along with slightly varied ways of working with memory to allow parallel function using MPI.

Because of the size of these graphs, it doesn't make sense to include them here (even at a small size and single space they take up more than a single page). So I'm going to leave the file on the server named as below.

128x128 100 ticks 0% threshold: Zero_Perc.log

Well, like the 16x16 version, we have a nice collection of random shapes that the game of life has allowed to exist. As the ticks continue we will see more movement among the shapes, however, no random reset events will occur, thus ensuring the natural state of the game of life.

128x128 100 ticks 25% threshold: Twent_Perc.log

I had accidently had the threshold set to 20 and there was a nice graph that had clearly been reset not too long ago. I could easily talk about how the reset was clearly start to be taken back by the random nature. However, when I correctly set the threshold to 25 percent, a reset has clearly occurred a tick ago. So it looks like a random graph, which it is. There's not much else to say.

128x128 100 ticks 50% threshold: Fifty_Perc.log

Same thing with 50 as with 25. A reset occurred in the not too distance path, making the graph just a random graph. I hoped it would be otherwise, but the game of life is strict.

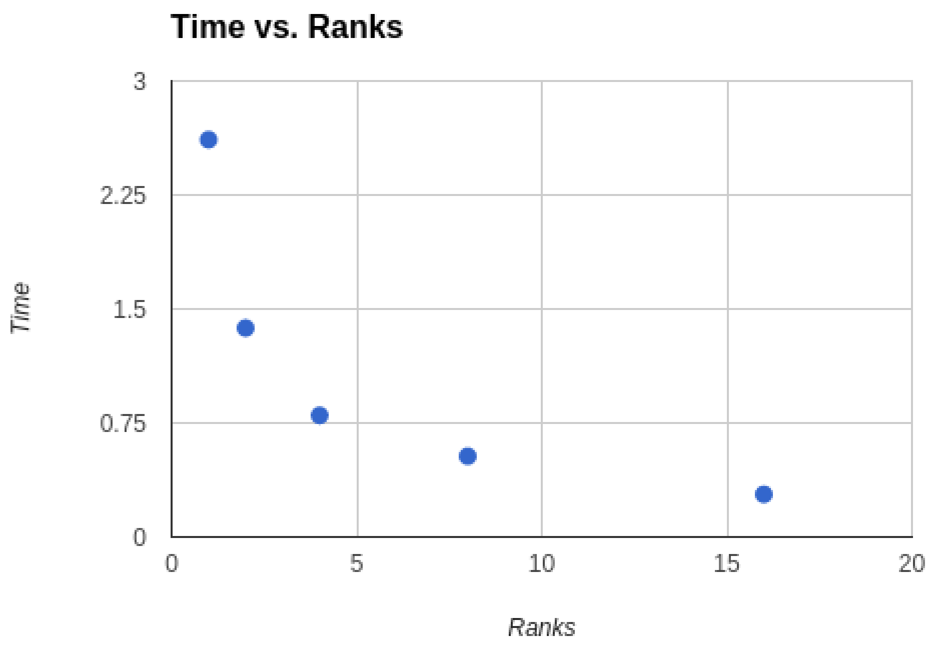
128x128 100 ticks 75% threshold: Seven_Perc.log

The chance things is really chancy. At the 100th tick a reset has occurred, yes, but probably about 10 ticks ago. So I can't say anything about what was going on before that. However, the dying process has begun to take effect as patterns are clearly emerging. It's really funny that it's all reset on the next tick.

128x128 100 ticks 90% threshold: Ninty_Perc.log

A reset occurred in the not too distant past (either this tick or the one before). There's not much more I can say without repeating what I've said for other files.

Given Ranks	Execution Time on Kratos(sec)
1	2.6139
2	1.3746
4	.79996
8	.53145
16	.28063



Out of curiosity, I also took note of how fast it would run with 32 and 64 ranks. And oddly enough, at 32 ranks, the compute time is longer than with 16 ranks. This occurs consistently. Run after run. 64 ranks is only marginally faster than 16 (around .25 seconds), but 32 floats much closer to .3 than 16 does.

Anyway, weird scaling aside, the greatest speed up occurs when I go from having 1 rank to having 2. It's a 90% speed up, or it takes 53% less time. All other speed ups are lower in comparison, partially because of the increase in overhead costs as you become more and more parallel. If we go by the measure of speedup/MPI ranks, again, simply having 2 ranks gives the greatest speedup per rank. While the 16 rank version is impressively fast compared to the 2 rank, we still have to consider the power being used is much greater for a far lower result.

I could also say that one could likely make a more efficient system than what I’ve created. There are some tricks with giving time for the messages to pass before requiring them (call the MPI_Irecv then do some calculations while waiting for the result). That being said, I don’t think they would create an immense difference from the simpler version I’ve already written.