Taylor Kirkpatrick

Project 1

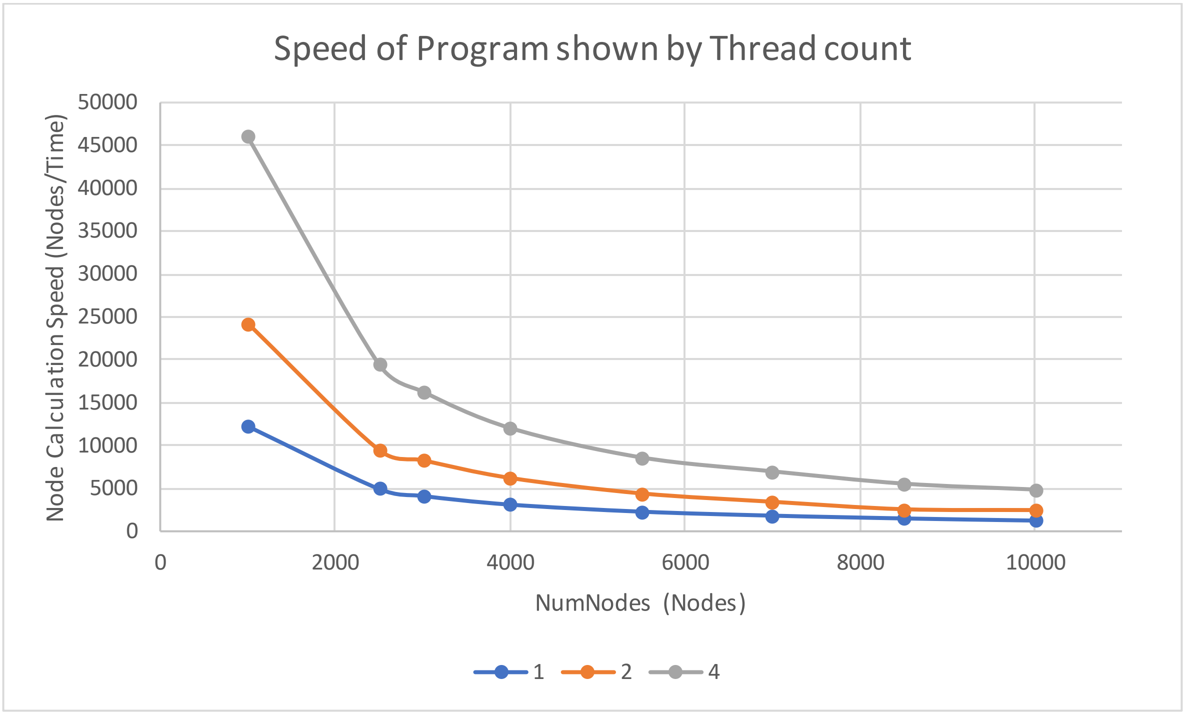
1. Run on the flip server, ssh-d in from my home desktop.
2. Every number I got was 25 and some change, so I have no reason to debate that number. The number gets a little different with more nodes, but that's similar to the infinite coastline paradox and it would be silly to keep extending the number of nodes to find an illogical volume. However, I'll say that using more nodes means theoretically more accurate, and so the number I got with 10000 nodes (roughly 25.31250084) is a good enough guess.
3. See the data and graphs on the next couple of pages.
4. To nobody's great surprise, the more threads you have the faster things go, but the more nodes you have (that is, things to process) the slower things go.
5. Kinda answered this one in #4, but more threads means you can process all your things faster, but higher node count means more things to be processed. This causes the fastest time to be at 4 threads 1000 nodes while the slowest run was 1 thread 10000 nodes (0.021 sec and 8.001 sec respectively).
6. After calculating speedup to be roughly 3.76…, I found that my F_p= .97 (roughly), or 97% of my program is parallel. A lot more than I was expecting. Note that I used the timing from when there were 5500 nodes, as that seemed to be a nice middle value to run these numbers from.
7. Thus my maximum speedup is 33.3…, a far cry from my current speedup. I suppose I'm only using four threads though, if I used many more I might be able to approach that. I also wonder if using multithreading (as opposed to multiprocessing) is adding enough overhead to have a significant impact on the speed. Though probably not at this stage.

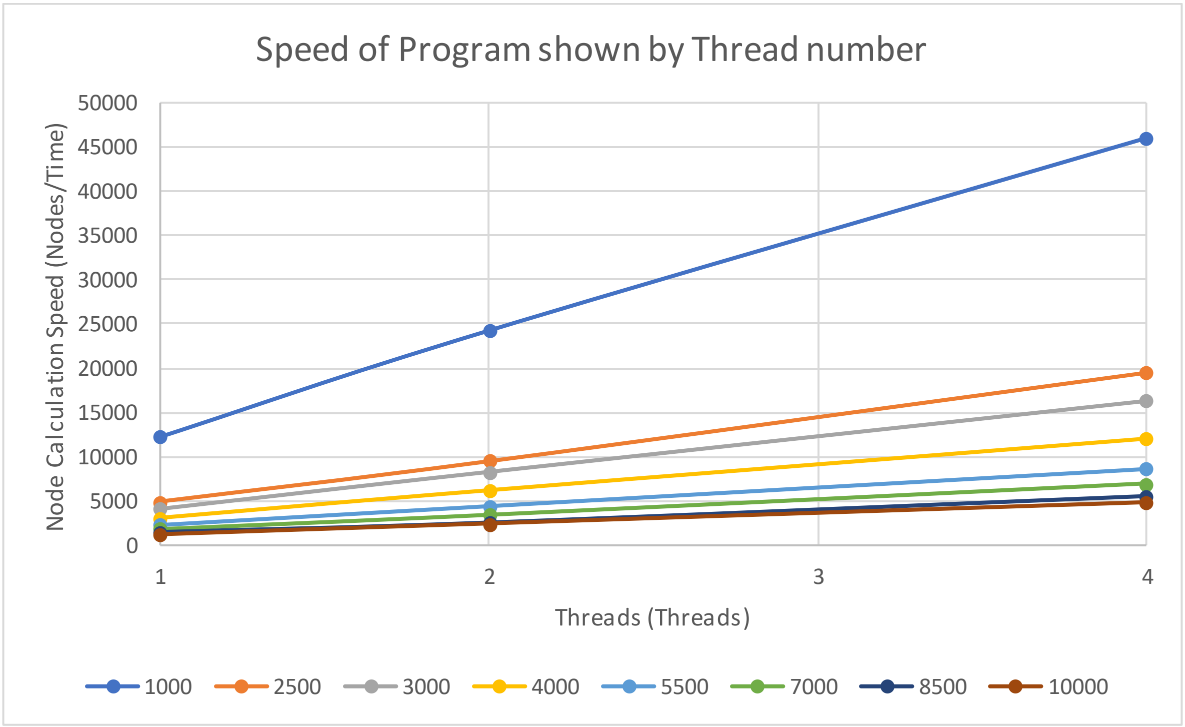
Data and graphs on next pages. Both are also contained in the submitted .xlsx document (since a .csv would probably mess with the graphs).

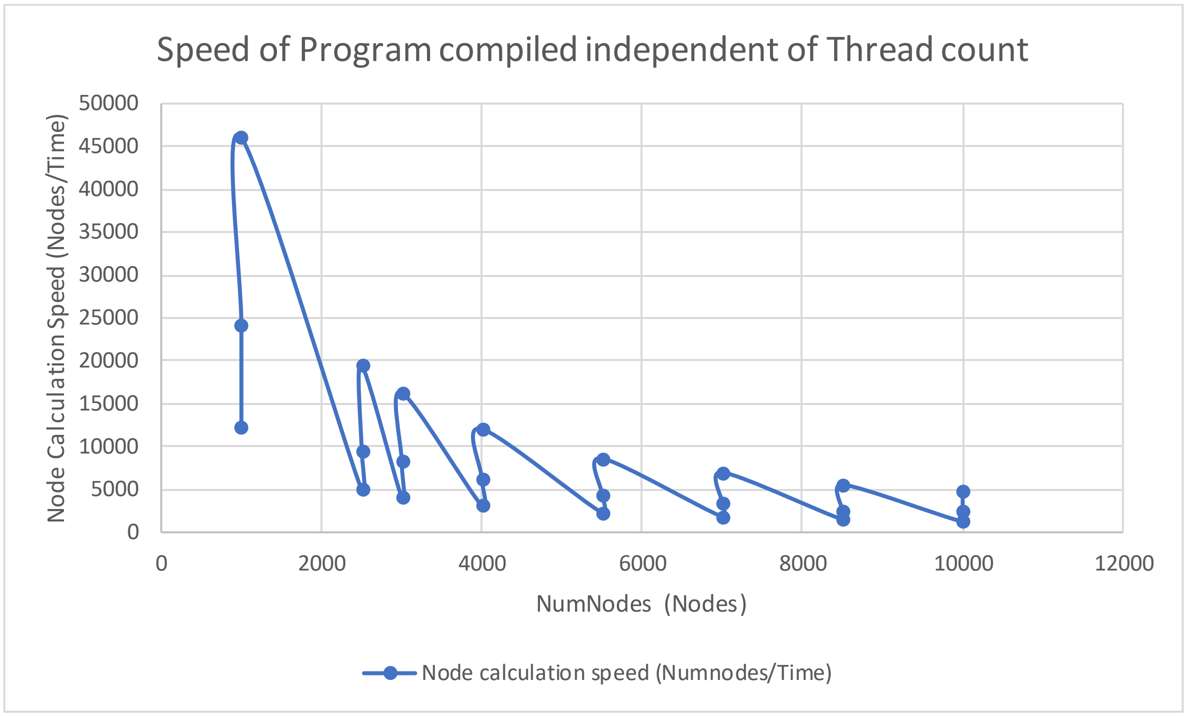
Data:

Numnodes	Threads	Volume	Time	Node calculation speed (Numnodes/Time)
1000	1	25.31250888	0.081625629	12251.05411
1000	2	25.31250888	0.041217048	24261.80562
1000	4	25.31250888	0.021732055	46014.97723
2500	1	25.312501	0.504989617	4950.59684
2500	2	25.312501	0.263076356	9502.944472
2500	4	25.312501	0.128247932	19493.49178
3000	1	25.31250084	0.728091618	4120.360578
3000	2	25.31250084	0.361649606	8295.322177
3000	4	25.31250084	0.183944995	16309.22335
4000	1	25.3124997	1.285705121	3111.133287
4000	2	25.3124997	0.642468436	6225.9868
4000	4	25.3124997	0.331883507	12052.42176
5500	1	25.31249963	2.401732437	2290.013623
5500	2	25.31249963	1.243504407	4422.983923
5500	4	25.31249963	0.638228837	8617.598701
7000	1	25.31250015	3.836173665	1824.734908
7000	2	25.31250015	2.031870808	3445.100924
7000	4	25.31250015	1.000586016	6995.90029
8500	1	25.31250129	5.665936893	1500.1932
8500	2	25.31250129	3.301512096	2574.577876
8500	4	25.31250129	1.527856527	5563.349601
10000	1	25.31250084	8.001517383	1249.762954
10000	2	25.31250084	4.038223842	2476.336229
10000	4	25.31250084	2.052332329	4872.505228

Graphs:


Speed of Program shown by Thread count
Node Calculation Speed (Nodes/Time)
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2000
4000
6000
8000
10000
NumNodes (Nodes)
1
2
4


Speed of Program shown by Thread number
Node Calculation Speed (Nodes/Time)
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
1
2
3
4
Threads (Threads)
1000
2500
3000
4000
5500
7000
8500
10000



This last graph is probably the best one to visualize the data. Thread count appears to matter far more with a lower number of nodes (1000 in this case), but as the nodes increase, it is no longer quite as big a difference. Thread count still makes a difference at higher node numbers, but the difference 2 threads to 1 thread makes is far less at 10000 nodes than it is at 1000 nodes.