ECE 4730: Parallel Systems

Project 1: Floyd's Algorithm

William Benton, Mitchell Cassaday

11/08/2019

Project Overview:

This project takes a look at how Floyd's algorithm operates while also comparing the run time when performed in serial vs in parallel. A function was made to create a matrix to perform Floyd's algorithm on. The matrix was created using user inputs to determine how large the matrix would be, the max value in each cell of the matrix, and the probability that there was a connection between the nodes. Once the matrix was created, it was stored in a file to use for the algorithm. The floyd-*.c files read in the matrix. In the floyd-parallel.c file, the matrix is then split up evenly among the tasks using the MPI function Create_Cart. Each process gets assigned a chunk from the cartesian coordinate system. Each of these chunks performs floyd's algorithm on a subset of the entire matrix. In the floyd-serial.c file, the algorithm is simply performed serially. The run time of both of these was taken and the results can be seen below.

Results:

Serial Floyd's Run Times in Seconds

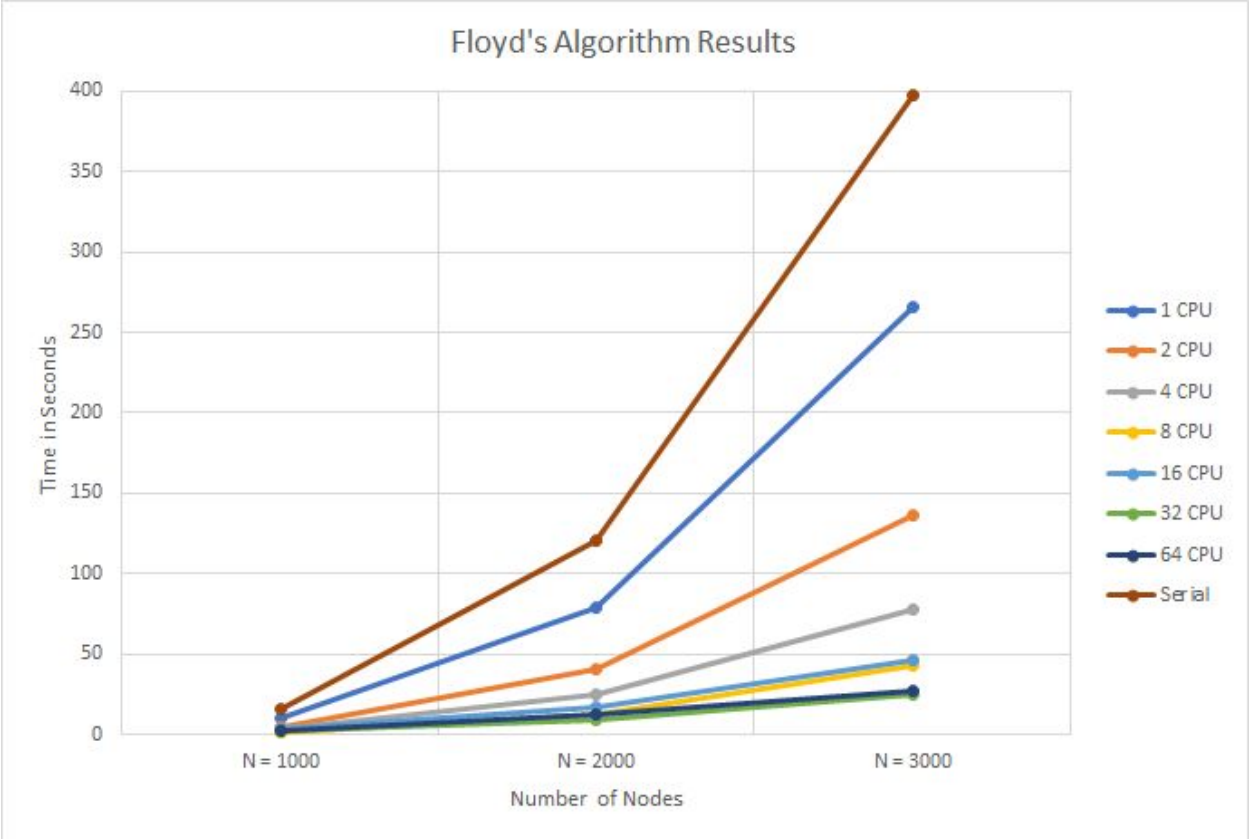
	N = 1000	**N = 2000**	**N = 3000**
Floyd's	16.1	120.45	396.98
Total	16.38	121.59	399.41

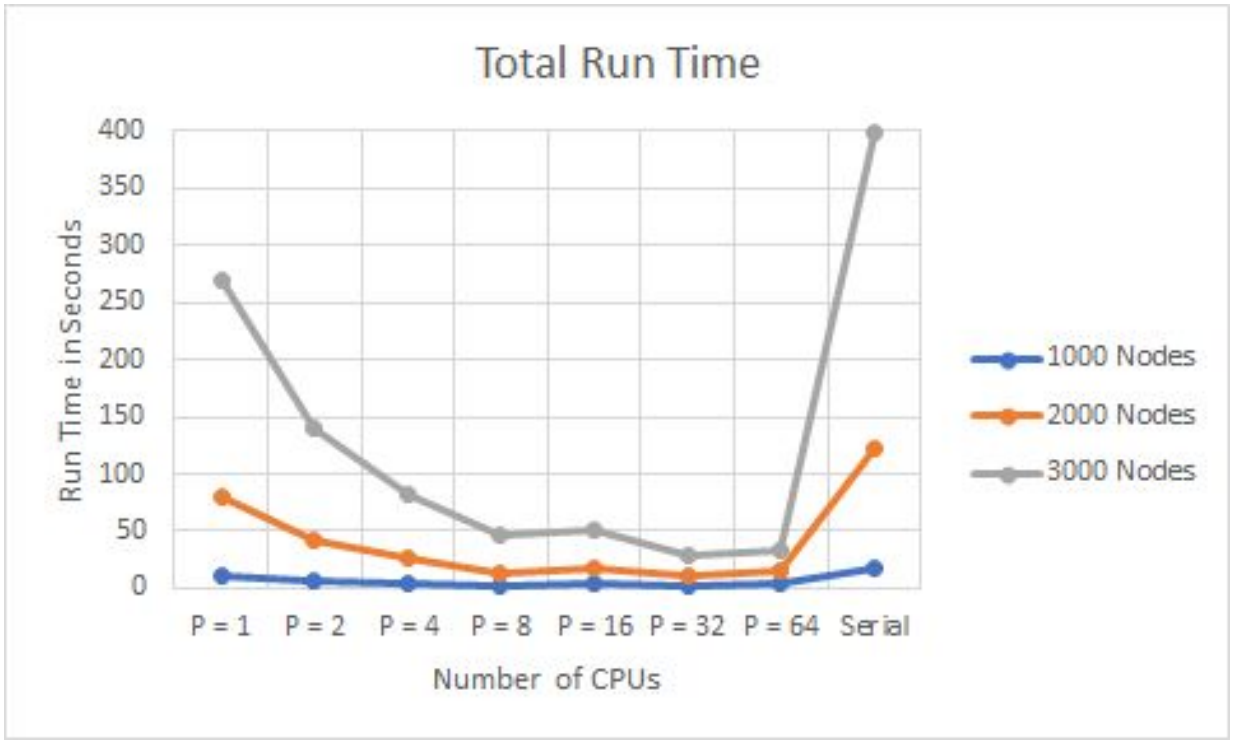
Parallel Floyd's Run Times in Seconds

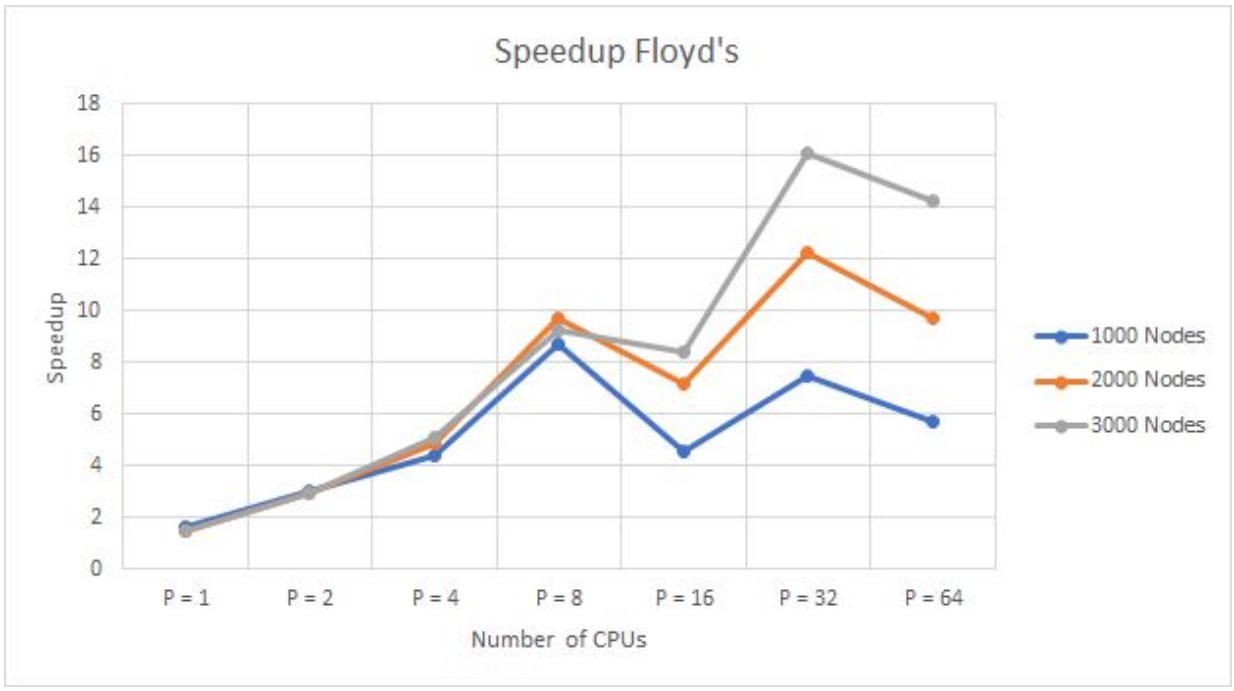
	N = 1000	**N = 2000**	**N = 3000**
P = 1	9.92	79.28	265.42
P = 2	5.36	40.70	136.09
P = 4	3.68	24.71	78.11
P = 8	1.85	12.35	43.00
P = 16	3.51	16.77	46.99
P = 32	2.16	9.80	24.59
P = 64	2.81	12.38	27.83

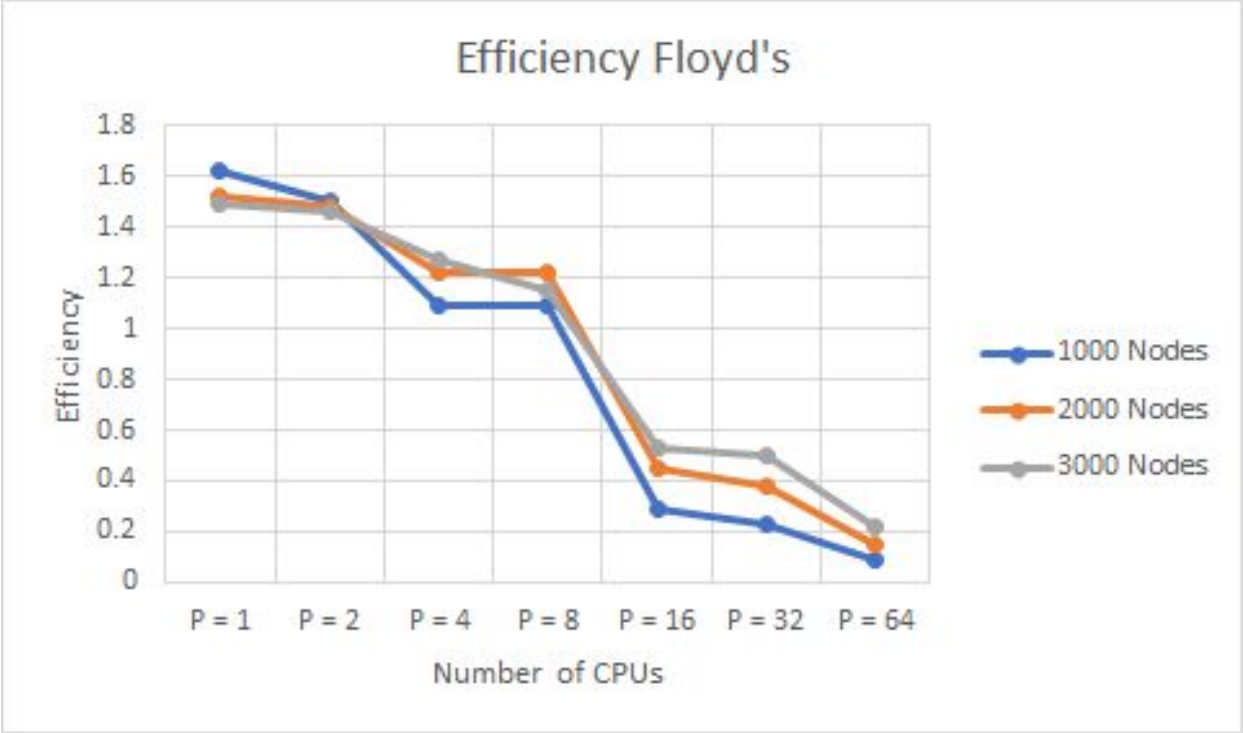
Floyd's Total Run Time in Seconds

Total	N = 1000	N = 2000	N = 3000
P = 1	10.37	80.86	268.89
P = 2	5.36	42.24	139.53
P = 4	4.08	26.31	81.62
P = 8	2.25	13.96	46.63
P = 16	3.96	18.39	50.63
P = 32	2.6	11.45	28.42
P = 64	3.48	14.35	31.96




Total Run Time
Run Time in Seconds
400
350
300
250
200
150
100
50
0
P = 1
P = 2
P = 4
P = 8
P = 16
P = 32
P = 64
Serial
Number of CPUs
1000 Nodes
2000 Nodes
3000 Nodes


Speedup Floyd's
Speedup
18
16
14
12
10
8
6
4
2
0
P = 1
P = 2
P = 4
P = 8
P = 16
P = 32
P = 64
Number of CPUs
1000 Nodes
2000 Nodes
3000 Nodes



These results were somewhat surprising. The speed improvements were roughly expected, except the dip in performance from 8 CPUs to 16 CPUs and the dip in performance from 32 CPUs to 64 CPUs. I expect either a steady rise in performance relative to the number of cores or a peak and then it settling back down. I would guess the performance would have peaked in one spot because of the time it takes for communication to occur, but it dropped at 16 CPUs, rose at 32 CPUs, and then dropped again for 64 CPUs. However, overall, we saw a good speedup curve and relatively good efficiency up until 16 CPUs. It appears that if efficiency is no matter, that 32 CPUs performed the best and if efficiency is the goal, 8 cores was the most efficient. The speedup was calculated with the formula: $T_{Serial} / T_{Parallel}$ and the efficiency was calculated using the formula: $T_{Serial} / (\text{Number of CPUs} * T_{Parallel})$. The results for the total speed up and total efficiency were not included in this report as they were extremely similar to the results of just the Floyd's time, but the graphs for those can be seen in the Excel file.