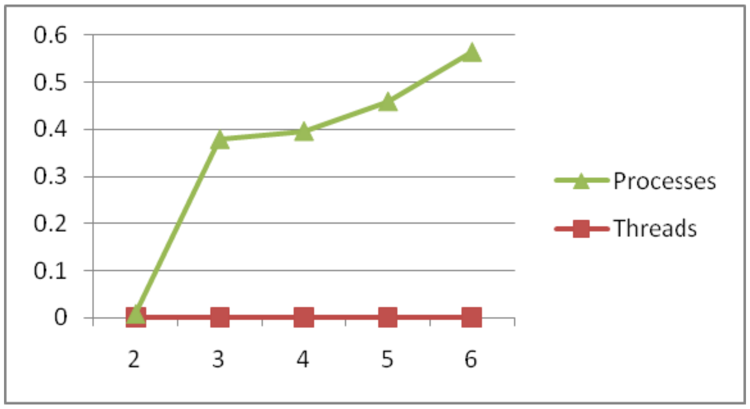
Harshil Shah
Matt Early
Anjal Ramani

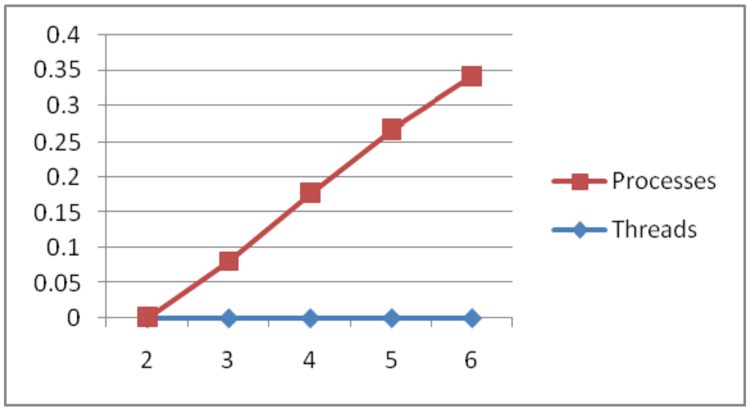
Graph 1 size:15

	Threads	Processes
2	0.000581	0.006251
3	0.000693	0.37762
4	0.000696	0.394838
5	0.000754	0.457621
6	0.000815	0.56286



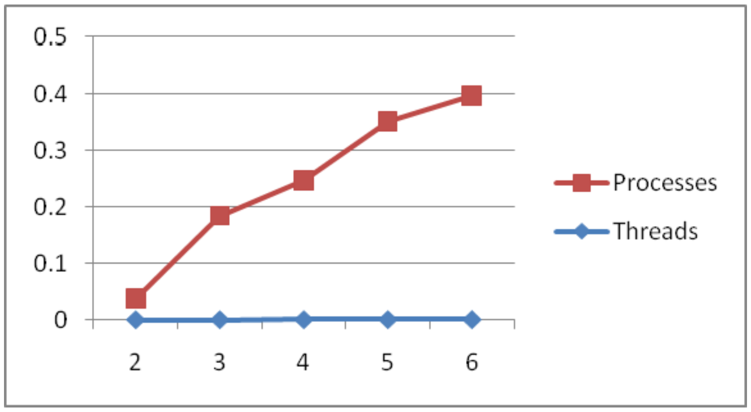
Graph 2 size:6

	Threads	Processes
2	0.000126	0.000585
3	0.000235	0.080307
4	0.000307	0.177202
5	0.000372	0.266586
6	0.000388	0.341526



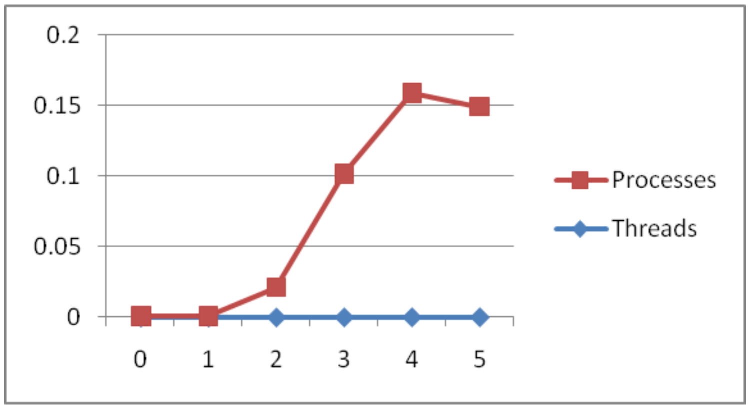
Graph 3 size:9

	Threads	Processes
2	0.000246	0.036527
3	0.000306	0.183351
4	0.00041	0.245528
5	0.000455	0.350477
6	0.000479	0.396371



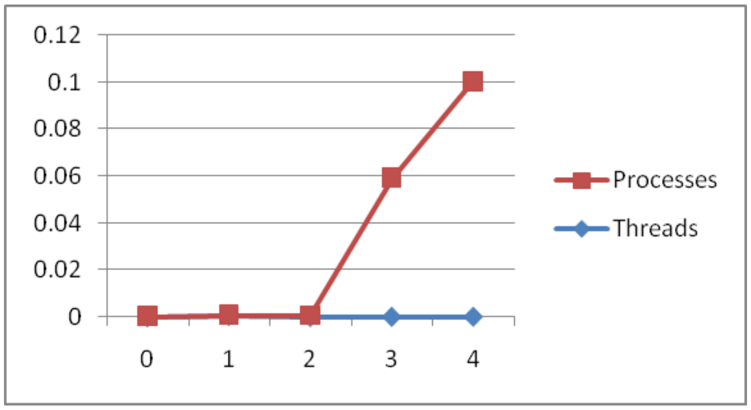
Graph 4 size:5

	Threads	Processes
0	0.000017	0.000346
1	0.000065	0.000501
2	0.000156	0.020705
3	0.000204	0.101312
4	0.000264	0.158681
5	0.00029	0.149028



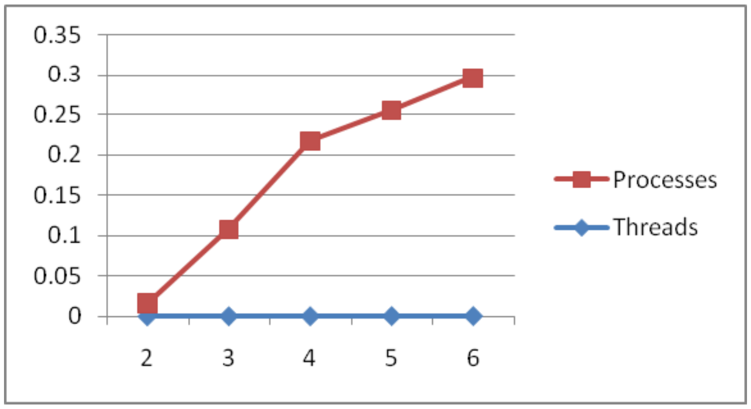
Graph 5 size:4

	Threads	Processes
0	0.000009	0.000179
1	0.00053	0.000402
2	0.000135	0.000532
3	0.000189	0.058994
4	0.000217	0.100203



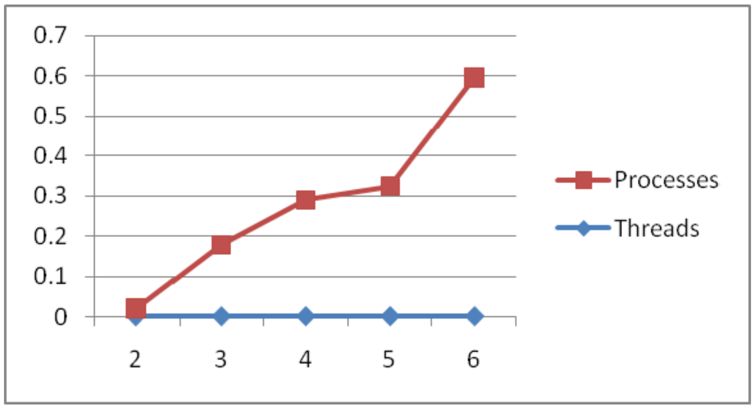
Graph 6 size:6

	Threads	Processes
2	0.000153	0.016175
3	0.000198	0.108114
4	0.000269	0.217641
5	0.000353	0.255436
6	0.000397	0.296353



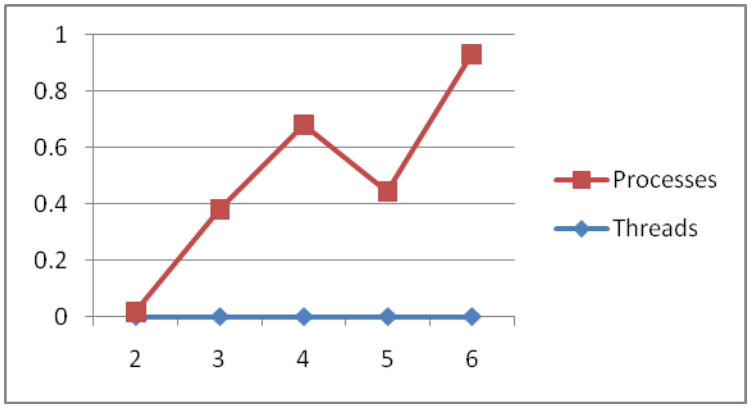
Graph 7 size:15

	Threads	Processes
2	0.00055	0.019766
3	0.000701	0.17714
4	0.000704	0.288633
5	0.000743	0.323017
6	0.000726	0.595056



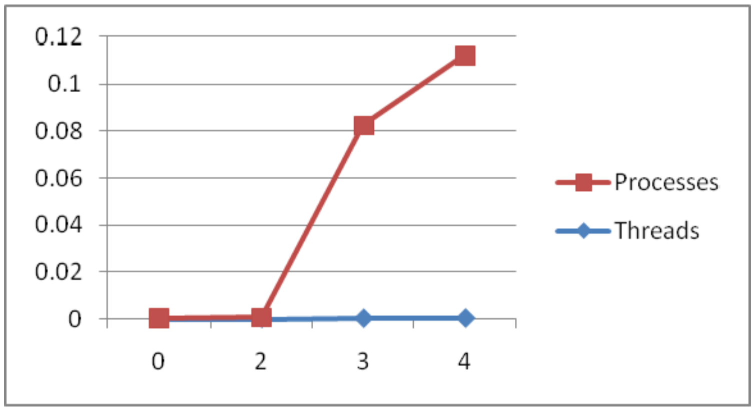
Graph 8 size:20

	Threads	Processes
2	0.001339	0.014146
3	0.00101	0.379861
4	0.0011	0.680696
5	0.001205	0.442051
6	0.001276	0.930821



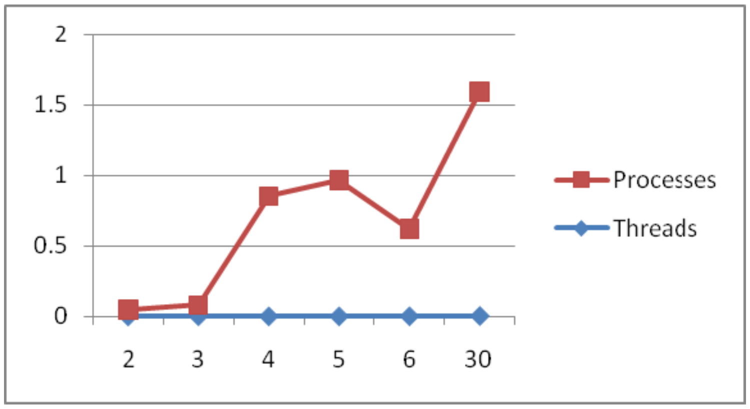
Graph 9 size:4

	Threads	Processes
0	0.00001	0.00018
2	0.000125	0.000359
3	0.00019	0.082314
4	0.000256	0.112134



Graph 10 size:30

	Threads	Processes
2	0.002284	0.043915
3	0.003652	0.074692
4	0.002591	0.849415
5	0.002682	0.963826
6	0.003211	0.614198
30	0.00556	1.586006



The above results have been run on wheaties.rutgers.edu. There are some issues regarding memory memory assignment for shared memory. The max size of matrix we could run on wheaties was 30 x 30 . (for processes)

Different machines have different memory/permissions and so hopefully you will consider that while grading but our both the programs run perfectly uptil 30 x 30 matrices. Threads work for bigger sizes cause they do not require shared memory to be assigned separately.

Our Conclusion:

The transitive closure of a graph is a very small algorithm and the work done by the actual algorithm is less compared to the overhead in creating threads/processes. Therefore in many cases we see that more threads/processes we create the more time it takes to run, very different from what one might think that more threads/processes less time waisted and faster your algorithm runs.

Also threads run much faster compared to processes because threads are “lightweight” i.e. they do not require too much memory or startup time unlike processes where fork() is expensive in terms of time and memory.

Talking about the time for threads and processes to complete the algorithm it is very clear that more processes/threads we create more time it takes(because of overhead is bigger than actual work time for algorithm). Also for a fixed no of threads/processes it takes more time to complete a given matrix of a larger size than that compared to a smaller one. Occasionally we do see some dips in time for a fixed graph size but different no of processes/threads, this is what is the intuitive thing that should happen, i.e. more processes/threads faster it takes to complete the work because less waiting time and splitting of work.