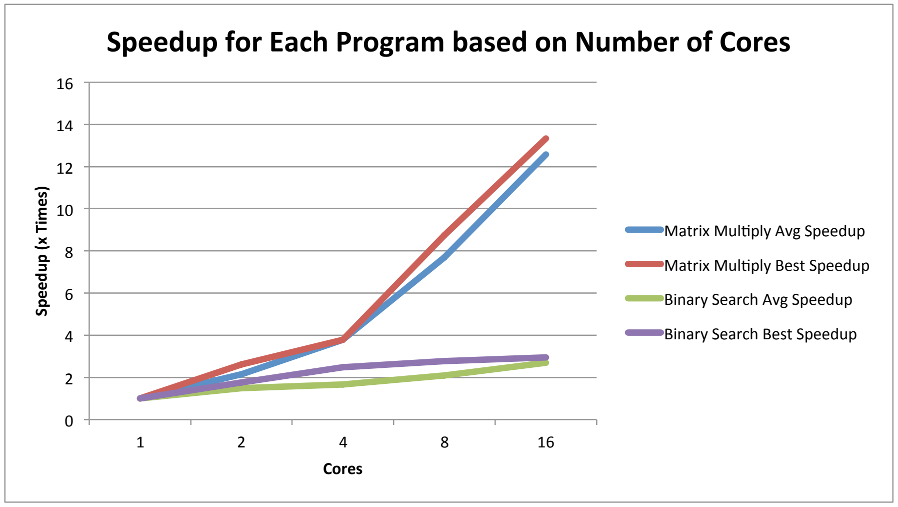
Nathan Tornquist - ECE 563 - Small Project Part 1

Matrix Multiply OpenMP
Two 900x900 Matricies Multiplied Together

Core	1	2	4	8	16
Run 1 (s)	10.911	6.312	3.457	1.533	1.163
Run 2 (s)	13.371	5.713	2.907	1.545	0.861
Run 3 (s)	11.539	5.642	3.228	1.803	0.818
Run 4 (s)	11.348	4.169	2.882	1.249	0.908
Average	11.79225	5.459	3.1185	1.5325	0.9375
Avg Speedup	1	2.16014838	3.78138528	7.69477977	12.5784
Best Speedup	1	2.61717438	3.78591256	8.73578863	13.3386308

Binary Search
Searching a Tree Organized by Name, for All Nodes of a Given Age

Core	1	2	4	8	16
Run 1 (s)	1.003	0.545	0.761	0.452	0.399
Run 2 (s)	0.964	0.634	0.389	0.826	0.492
Run 3 (s)	1.038	1.003	0.7	0.348	0.326
Run 4 (s)	1.313	0.714	0.741	0.439	0.387
Average	1.0795	0.724	0.64775	0.51625	0.401
Avg Speedup	1	1.4910221	1.66653802	2.09104116	2.69201995
Best Speedup	1	1.76880734	2.4781491	2.77011494	2.95705521



Matrix Multiply:

Matrix Multiply scales very well to more cores. The arrays can be divided nicely, and each core performes the same amount of work. This means that increasing the number of cores directly increases the speed at which the program completes.

Binary Search:

The binary search algorithm does not scale as well. If I had used a fully balanced binary tree, I would have expected to see results very similar to those of the matrix multiply. However, a fully balanced tree is unlikely in every use case, so I wanted to see how the program scaled when the tree could be of any shape. This means that the nodes given to a specific core may not be equal in number of type. The results clearly show an increase as you add cores, but as expected it is not as sharp. When running the analysis, some cores would find upwards of 10x, 100x, or 1000x more matches than other cores. This is because of distribution with random numbers and the fact that I distribute the work using a breadth first search of three tree until I have enough parent nodes to give each core its own tree.