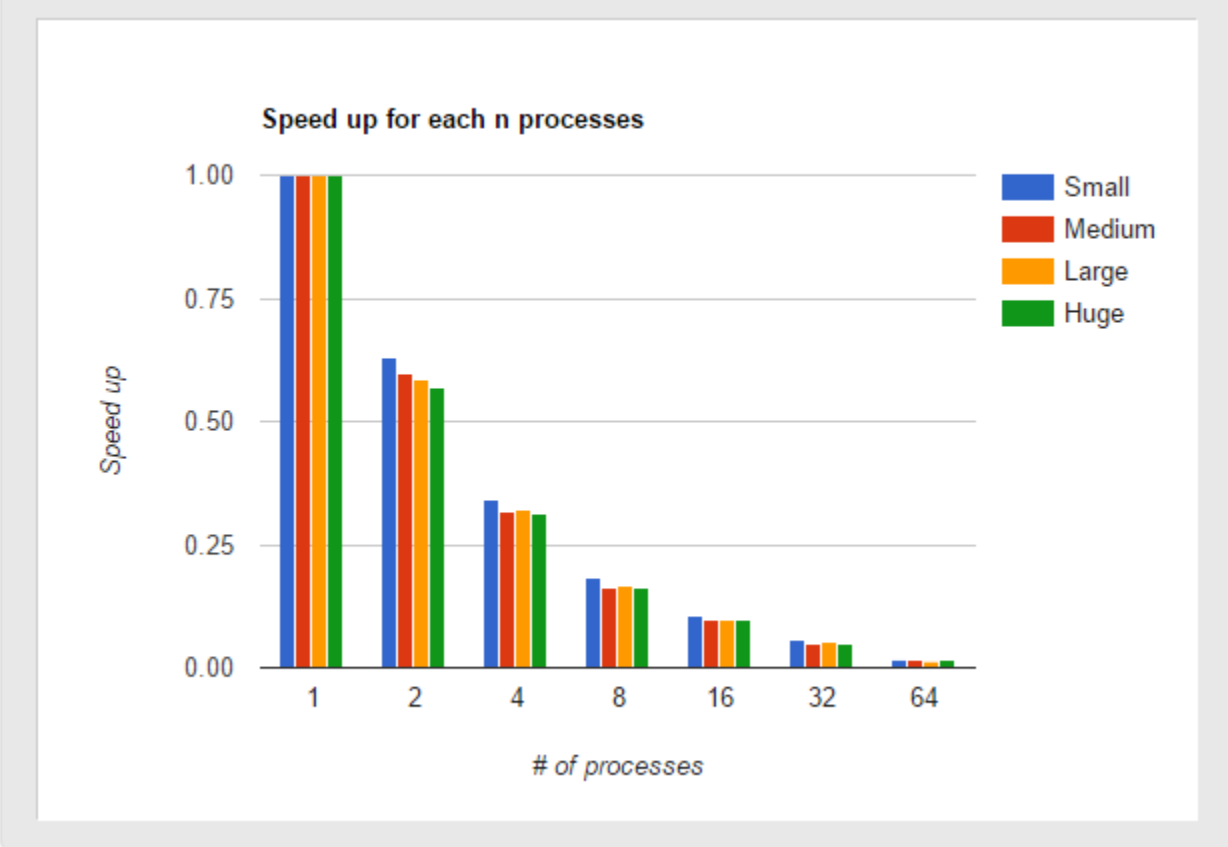
Matt Kwong

Parallel Computing Lab 1

Average speed over 5 runs using sys time

| | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|---|
| Small | .158 | .251 | .461 | .875 | 1.501 | 2.897 | 8.765 |
| Medium | .149 | .249 | .471 | .912 | 1.512 | 3.122 | 9.111 |
| Large | .149 | .255 | .466 | .902 | 1.498 | 2.903 | 10.716 |
| Huge | .146 | .256 | .470 | .908 | 1.487 | 3.098 | 9.419 |

Times are calculated as an average of 5 runs each




| | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|---|
| **Small** | 1 | 0.629482 | 0.342733 | 0.180571 | 0.105263 | 0.054539 | 0.018026 |
| **Medium** | 1 | 0.598394 | 0.316348 | 0.163377 | 0.098545 | 0.047726 | 0.016354 |
| **Large** | 1 | 0.584314 | 0.319742 | 0.165188 | 0.099466 | 0.051326 | 0.013904 |
| **Huge** | 1 | 0.570313 | 0.310638 | 0.160793 | 0.098184 | 0.047127 | 0.015501 |

As we can see, there is little change in timing across all sizes. Furthermore, we do not see a decrease in time taken to compute even the largest problem. The biggest reason for this is that communication between processes is **expensive**.

MPI is useful when there is little communication necessary and that there is a lot of computation that does not require data being updated or communicated. As we can see from my source code, aside from the usual initializing functions and finalize function, I only use Allgatherv. I use this function once every iteration of the algorithm described in the lab (Jacobi method) and essentially is a Gather and Bcast combined to share information from every process.

I considered in my analysis of the algorithm what variables needed to be shared, and the x vector is essential to update on every process after each iteration. As per matrix multiplication of an NxN matrix times a Nx1 (column vector), we need all values from the column vector to multiply one row in the NxN matrix. I considered assigning each process a subset of the column vector and calculating a small part of each row in the NxN matrix, but this would require that each process gather data from each other process to recalculate the new column vector required for the next matrix, and this requires way more communication.

In terms of load balancing, each process calculates the new x for each row assigned to them and broadcasts their new x and waits until all processes finish calculating and process before continuing to the next iteration. Considering this is the most calculation heavy part of the program, I consider this well balanced.

I also consider calculating on small matrixes an issue with not seeing an improvement on performance. The following issues illustrate this:

- Between the small and huge matrix, my program runs faster on the huge program because of variance, despite taking the average of 5 times
- Even using gsref, the time for both small and huge is .001 (I imagine it isn't too far apart even if I could see the time past 3 significant figures)
- For an NxN matrix, one iteration of my algorithm requires about $c*N^2$ calculations with c being relatively small (probably around 5). Small has n=5 and huge has n=50. Assuming c=5, one iteration on small is about 125 calculations, while one iteration on huge is around 12,500 calculations. This is not a lot of calculations, so I expect my program will see improvements in parallel speeds with large matrices with cost of calculations exceeding communication costs.
- My initial sequential program (same algorithm without any MPI) had time speeds of .001 on both huge and small

Lastly, I consider the lack of a general increase in timing for large matrices among most processes (even after taking the average of 5 speeds to reduce variance) to support my idea that the rate of computation difficulty grows slowly with N size. I would like to test my program on matrices with $N > 1000$ to see the benefits of parallelizing the Jacobi method.