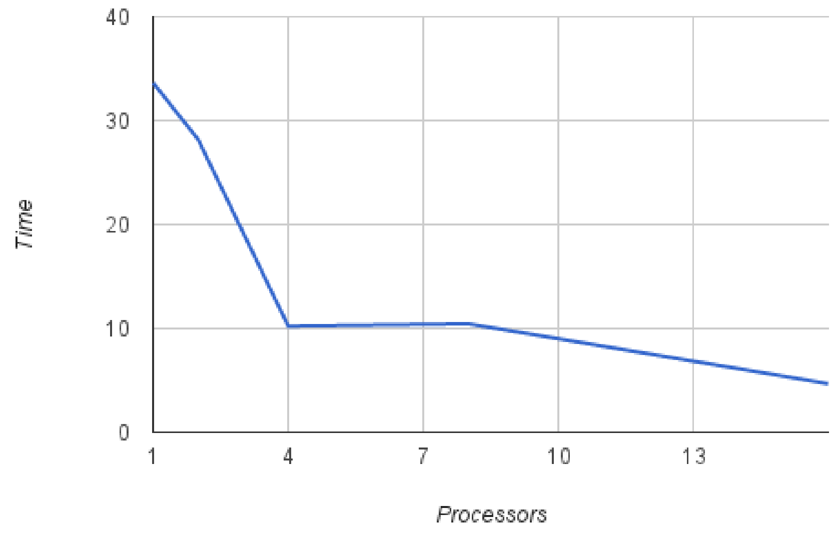
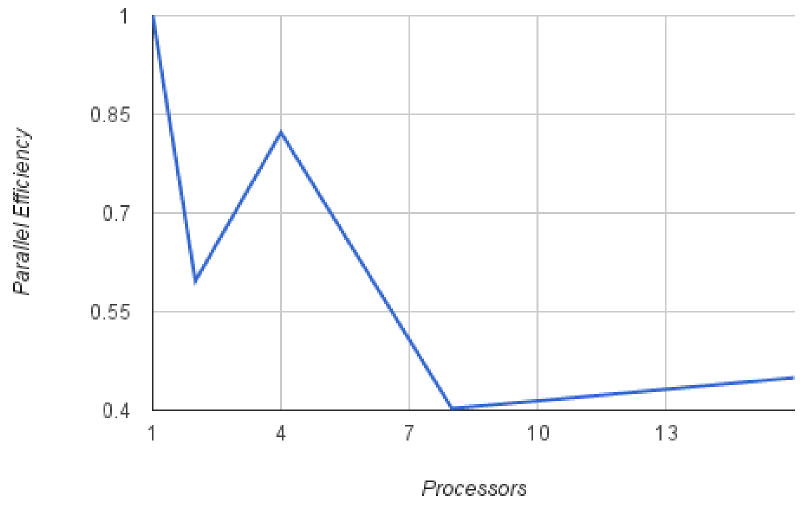
# Assignment 2

**Liam Cardenas, CSIL: liamcardenas**
**Steven Cabral, CSIL: scabral**

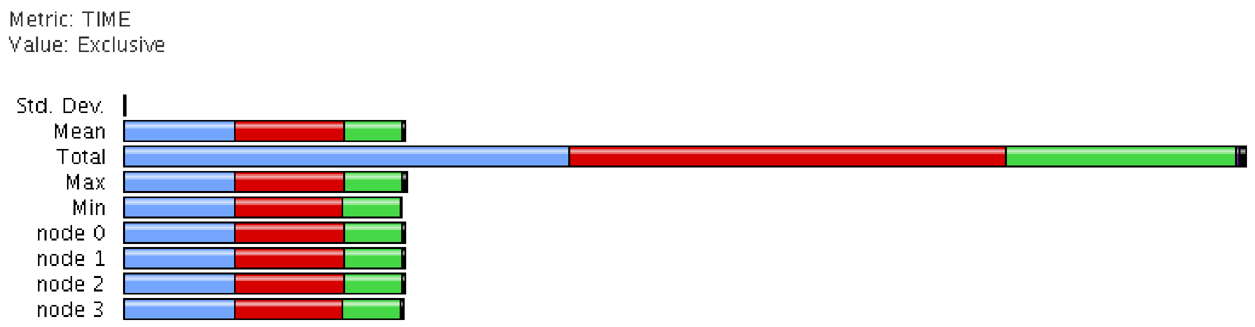
1. Where size = 6500





| Time | Processors | Efficiency |
|---|---|---|
| 33.637731 | 1 | 1 |
| 28.181199 | 2 | .5968 |
| 10.226924 | 4 | .8223 |
| 10.442877 | 8 | .4026 |
| 4.678492 | 16 | .4494 |

2. One of the largest chunks of time was  caused by the initialization of MPI. This shows that distributed applications, such as this one, are more suited for long running jobs where the long initialization time is marginalized. Other than that, generating the matrix and multiplying the vector against the matrix took longer than the communication between the nodes, showing that these calculations are good for being distributed across large distributed systems. In this case, the reward for partitioning the data outways the cost. However, there were some configurations, illustrated in part 1, in which adding more cores did slow down the computation time. The time breakdown is clearly illustrated by this paraprof diagram (4 cores, size = 10000, 1000 iterations; blue is initialization, red is matrix vector multiplication, green is generating the matrix, and the blackish area at the end is everything else) :



Overall, as more cores are added, the time does go down, however, so does the efficiency per core. This is due to the costs of initializing, communicating between, and synchronizing all the cores. The question “where is the data?” really comes into play here, because the fact that the data must be passed between different processes severely slows down the program. That being said, low efficiency does not necessarily translate to worse overall time performance. In part 1, as the cores increased from 1 to 16, time went down along with the efficiency, however, there were times in which efficiency and time went up, such as the jumps from 8 to 16 and 4 to 8, respectively. This is most likely due to the marginal initialization time of running on 8 processors being longer than the marginal matrix multiplication and generation speed increases. Despite this, the trend is clearly that of a decreasing curve that is asymptotically approaching a much smaller value-- meaning that, as the number of cores gets arbitrarily large, the value of adding more cores decreases. It is plausible that one could also add so many cores that the addition of more would slow down the program. This would be the case if there were so many, that the cost of broadcasting was higher than the computational workload given to a process. These results are consistent with the theoretical expectations of a distributed program, where the parallel efficiency is always less than or equal to 1.