**Homework 3 Report**

This report has two parts. First, This document contains an overview of the project and then breakdowns of charts with key timing information for the two algorithms. The attached excel file has the source data used to generate this and four different sample outputs. For both cyclic reduction (CR) and recursive doubling (RD), I’ve provided a large sample output (N=511 or 512, respectively) and a small sample output (N=31 or 32).

This project has two big conclusions: First, CR is substantially faster than RD – on the order of about 4 times quicker. Second, communication overhead is huge: In CR, for example, 64 cores was not the fastest method until we reached N = 2^22. At large N, however, we do see the expected results: namely, that doubling the number of processes halves the total processing time. This effect is slightly smaller in RD, where doubling the number of processors fails to fully half processing time, though these results do demonstrate very efficient parallelization.

All in all, it seems very clear that CR is a far better algorithm in this situation. In addition to the performance problems already noted, it’s also important to note that RD suffers from some rather severe numerical stability problems. The calculation of the V vectors results in multiplication and division by small doubles, which was severe enough to present errors with magnitudes up to 1e-2 even with small N.

This shows the time it took to solve a matrix of size N, with a different line for each processor. Despite communication overhead dominating at first, P=64 does become the fastest after about N=2^22. Interestingly, each processor was the fastest for at least one N.

Note the log scales on both the N and T axis. That means that we are seeing, in the long term, approximately linear scaling regardless of the number of processors.

Up to P=16, doubling the number of processors approximately halves calculation times, for sufficiently large N. Above P=16, doubling the number of processors doesn't save nearly as much time, probably because RD requires so much communication.

This shows the same data as the previous chart, but for CR.

Note the key differences:

* First, this chart shows drastically smaller computation times all around – a final improvement of 4x+ over RD.
* Second, even across many machines, parallelization has tremendous speed ups for large N, improving by nearly double when you double the processor count, even when that requires increasing the number of machines.
* Next, 4 Processors becomes the fastest for very small N, likely due to how few messages this algorithm needs. However, 64 processes once again becomes quickest at N=2^22, since message passing between machines has a very high cost.

This is another presentation of the data above, except now we hold N constant and analyze the speed up encountered by adding processors. For sufficiently large N in RD, this shows almost linear increases in performance, but it deviates slightly at N=32 and N=64, since adding new machines increases the cost of message passing overhead.

Once again, note how much quicker CR is than RD. This is true regardless of the number of processors. Also note that performance improves linearly with P, given N > 2^22. Less than that, and sitting at 16 processes seems to be the best.