Daniel Rings

EECS 587

HW 3

9 October 2014

**Matrix Decomposition:** I split the matrix horizontally into p sections of n/p rows. If n does not divide evenly into p, I put one more row in each section, starting with the first section. In this way, no section size will differ from another by more than one row. This made it easier to calculate the verification value and ensured that each section had at least 2 rows in it.

**Communication:** Since the top and bottom rows of each section (except the first and last) depend on rows in another section (and vice-versa), I have each section send its top and bottom rows to the sections that need it and have each section receive an extra row above and below their section. I call these rows “swapRows”, and they are similar to the “ghost values” discussed in class using the one-dimensional array. At each iteration, a section calls a non-blocking receive on the swap rows it needs, and fires off a non-blocking send on the rows that other sections depend on. However, MPI\_Wait is not called on the receive until after all calculations and operations have been performed on the rows that are only intra-section dependent. After that, the rows dependent on other sections are calculated. In this way, each section has the maximum amount of time possible to receive the swap row without halting processing.

**Timing:** The time for the average of runs on in each configuration (and verification values) are listed below:

N: 1000 P: 1

Time: 2.6464 Verification Sum: 490.817

Speedup: - Scaling (vs Serial): -

N: 1000 P: 4

Time: 0.613234 Verification Sum: 490.654

Speedup: 4.31548 Scaling (vs Serial): 1.07887

N: 1000 P: 16

Time: 0.459648 Verification Sum: 486.536

Speedup: 5.75745 Scaling (vs Serial): 0.35984

N: 1000 P: 36

Time: 0.11758 Verification Sum: 490.817

Speedup: 22.50723 Scaling (vs Serial): 0.6252

N: 5000 P: 1

Time: 63.7018 Verification Sum: 2486.12

Speedup: - Scaling (vs Serial): - Scaling (vs 1000, expected 25): 24.07

N: 5000 P: 4

Time: 19.8378 Verification Sum: 2486.12

Speedup: 3.21113 Scaling (vs Serial): 0.80278 Scaling (vs 1000): 32.35

N: 5000 P: 16

Time: 8.25357 Verification Sum: 2485.95

Speedup: 7.71809 Scaling (vs Serial): 0.48238 Scaling (vs 1000): 17.96

N: 5000 P: 36

Time: 5.12358 Verification Sum: 2483.91

Speedup: 12.43306 Scaling (vs Serial): 0.49732 Scaling (vs 1000): 43.575

Note: I believe discrepancies in the verification sums are due to floating point calculations, because there is no particular pattern to their variation. It is not just variation between serial and parallel sums. Some parallel sums differ from other parallel sums of the same problem. Since all the parallel sums are computed in the same manner, I believe the discrepancies are due to floating point calculations.

**Speedup and Scaling:** The speedup of solving the problem in parallel increased as p increased, but not nearly by a factor of p (although it was greater than a factor of p in one case). Some communication overhead exists in the parallel cases. Also, although the sends and receives were optimized to be non-blocking for as long as possible, it is possible that a receive had to wait for a message at some point, leading to less than perfect speedup. The scaling vs serial of the problems was less than perfect in most cases, as with the speedup. The scaling got worse as p increased. The scaling vs smaller problem size of most configurations was sometimes better than 25 (the expected) and sometimes worse. It depended heavily on how fast its equivalent smaller-size partner was. The scaling looks great for N = 5000, P = 16. It only took 17.96 times as long as the N = 1000, P = 16 configuration (when it was expected to take 25 times as long). But this was because the runs for N = 1000, P = 16 took quite a long time (relatively) to compute in the first place. It seems as though the number of processors affected scaling more consistently and adversely than the size of the problem.