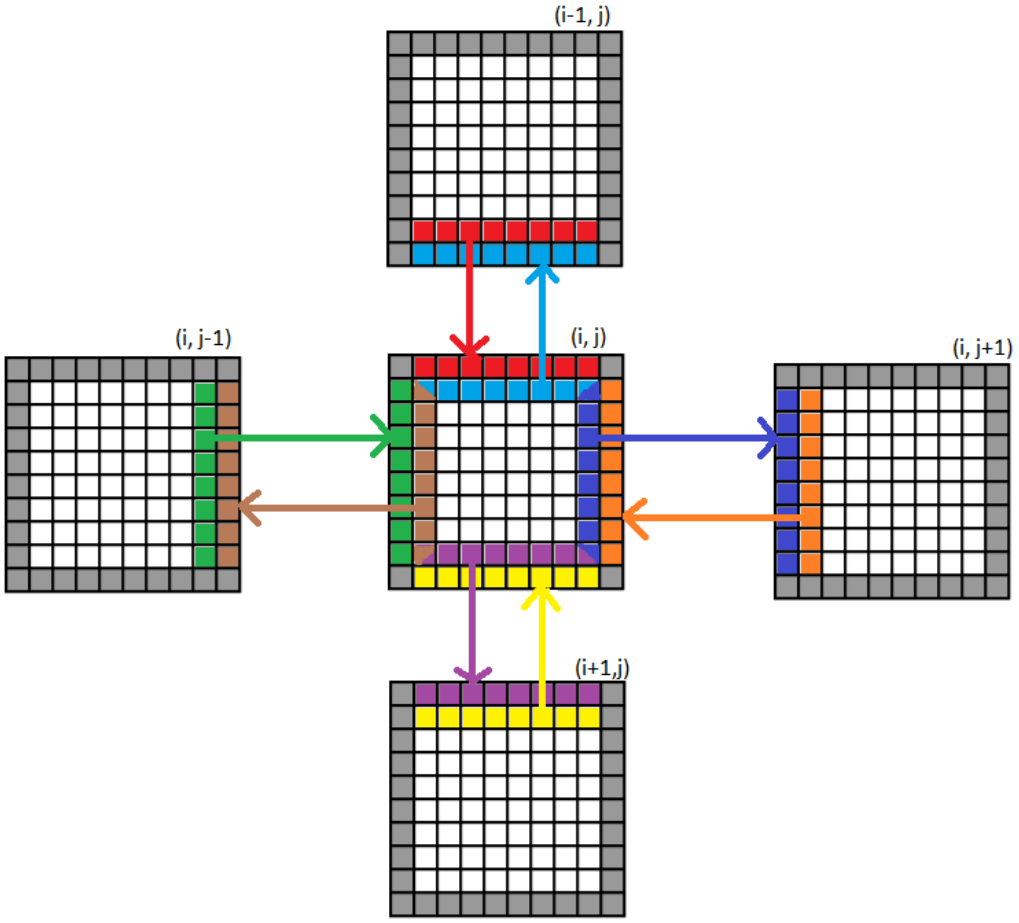
Wesley Chen
CS205 HW3 P2

My Masterpiece Diagram:



For design, I chose to use comm.split as I was more familiar with this (I did an N-body problem over the summer with Cris.) Getting a unique column and row rank also made the most intuitive sense – rather than working with modulus to send down the "columns". I would use cartcomm as that was designed for such data, but I couldn't find it documented well.

A cool thing about the code is that I initialize only the portion that each processor needs – so the domain decomposition starts at the initialization phase. In discussion with other students, some did the global initializing then a "cut" to get the domain of interest but this was a waste. This code only initializes the area that each processor is responsible for.
I ran the code with varying Nx and Ny as well as with Px and Py. The following was run with the Send/recv version as that was the faster one. Analysis will come later.

With Nx=Ny=256, Px=8, Py=4
Time: 9.527354
Time/iteration: 0.012098

With Nx=Ny=256, Px=4, Py=4
Time: 4.780443
Time/iteration: 0.006070

With Nx=Ny=256, Px=4, Py=2
Time: 2.408534
Time/iteration: 0.003058

With Nx=Ny=256, Px=2, Py=2
Time: 1.033291
Time/iteration: 0.001312

With Nx=Ny =128, Px = 4, Py=2
Time: 2.178619
Time/iteration: 0.002767

With Nx=Ny=64, Px=2, Py= 1
Time: 0.463661
Time/iteration 0.000589

With Nx=Ny=256, Px=8, Py=2
Time: 4.693171
Time/iteration: 0.005960

With Nx=Ny=256, Px=16, Py=1
Time: 3.563081
Time/iteration: 0.004525

With Nx=Ny=256, Serial Version takes 8.599 seconds timed by time.time().
With Nx=Ny=128, Serial Version takes 0.8545
With Nx=Ny=64, Serial Version takes 0.15027

The program exhibits weak scaling because the overhead scales linearly with number of processors but the amount of communication is constant. The slowness of each added processor comes from the fact that there is a scaling overhead, however, this gets cancelled out when you divide by the number of processors in the efficiency calculation. We notice from the data that when the amount of work per processor is fixed (like Nx=Ny=256, Px=8, Py=4 and then looking at Nx=Ny=128, Px=4, Py=2,) we see that

the efficiency is the same – though the times are not. Other trends to note are that if we use the same amount of processors, the time changes a bit but not by much – as the communication overhead is still there but some Px by Py dimensions allow for marginally less or more communication depending on how long the “borders” are.

Finally, as expected, if we keep the same number of processors but quadruple our grid size, then we see greater than a quadruple slowdown as expected – as each processor must now do 4 times the work, but also need to communicate more as well, though this increase communication is not quadratic.

This data did not seem sensitive to multiple runs – as consecutive runs gave the same results (unlike some of the other tests we did with MR and other MPI runs).