Scott Todd March 17, 2013

Parallel Computing
Programming Assignment #3: Parallel Matrix Multiply on the IBM Blue Gene/Q
Write-up

Notes about my experiments:

I used 13 buffers for network communications – one working buffer containing the section of matrix B currently being used for multiplication, 6 MPI_Irecv buffers waiting for incoming communications, and 6 MPI_Isend buffers queuing up outgoing matrices. I suspected that designing that flexibility into my code would allow for managing the tradeoff between memory/network usage and accounting for delays between nodes.

In measuring the time spent in MPI_Isend and MPI_Irecv, I included other surrounding code involved in network communications, since I believed the intent of the project was to experiment with the tradeoff between time spent in communications and time spent multiplying matrix elements that naturally comes from altering the number of subdivisions of the matrix multiply problem.

I initially measured only time spent actually in the MPI_Isend and MPI_Irecv operations, but I found that up to 80% of the total time measured was not represented in the sum of the time spent in multiplication, MPI_Isend, and MPI_Irecv. After including surrounding code in my measurements, only up to 5% of the total time was unaccounted for in the mapping enabled tests, up to 20% was unaccounted for in the mapping disabled tests, and 10% was unaccounted for in the kratos test.

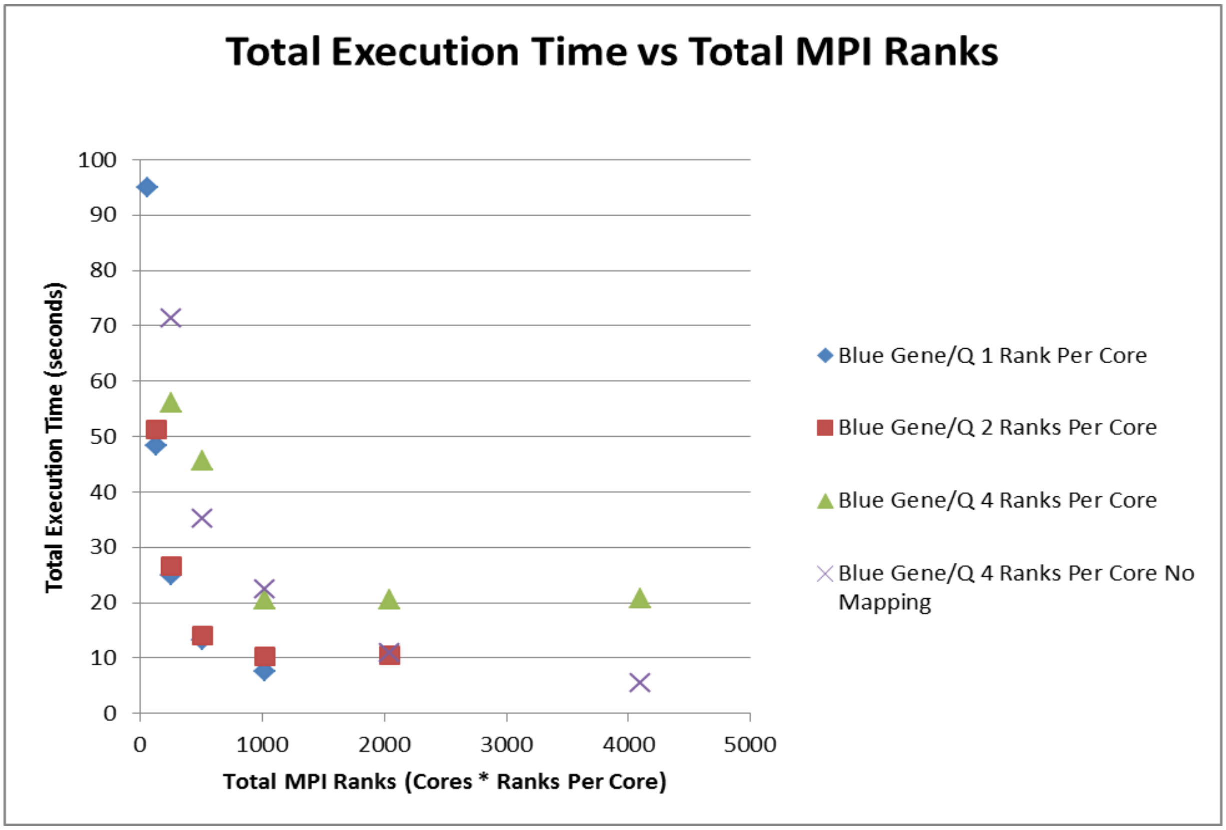
As a result of rerunning my tests to alter my timing measurements, I noticed how consistent the runtimes are on the Blue Gene/Q. Between three tests, the largest percent change in total execution time was 4%, while the smallest was 0.03%.

My finalized data sets can be viewed in data.xlsx, and the tests can be run from the source code as well, using the provided makefile and python scripts (batch_creator.py and output_parser.py). Note that the source code (mm.c) sets the clock rate to 2666700000, change this to 1600000000 to run on the Blue Gene/Q.

Scott Todd March 17, 2013

Graphs

Total Execution Time



This graph shows total execution time, measured as the number of CPU cycles completed between program initialization and program completion in rank 0 divided by the clock rate, plotted against the total number of MPI ranks used by the program.

For reference, 8 cores of Kratos took 1663 seconds to complete.

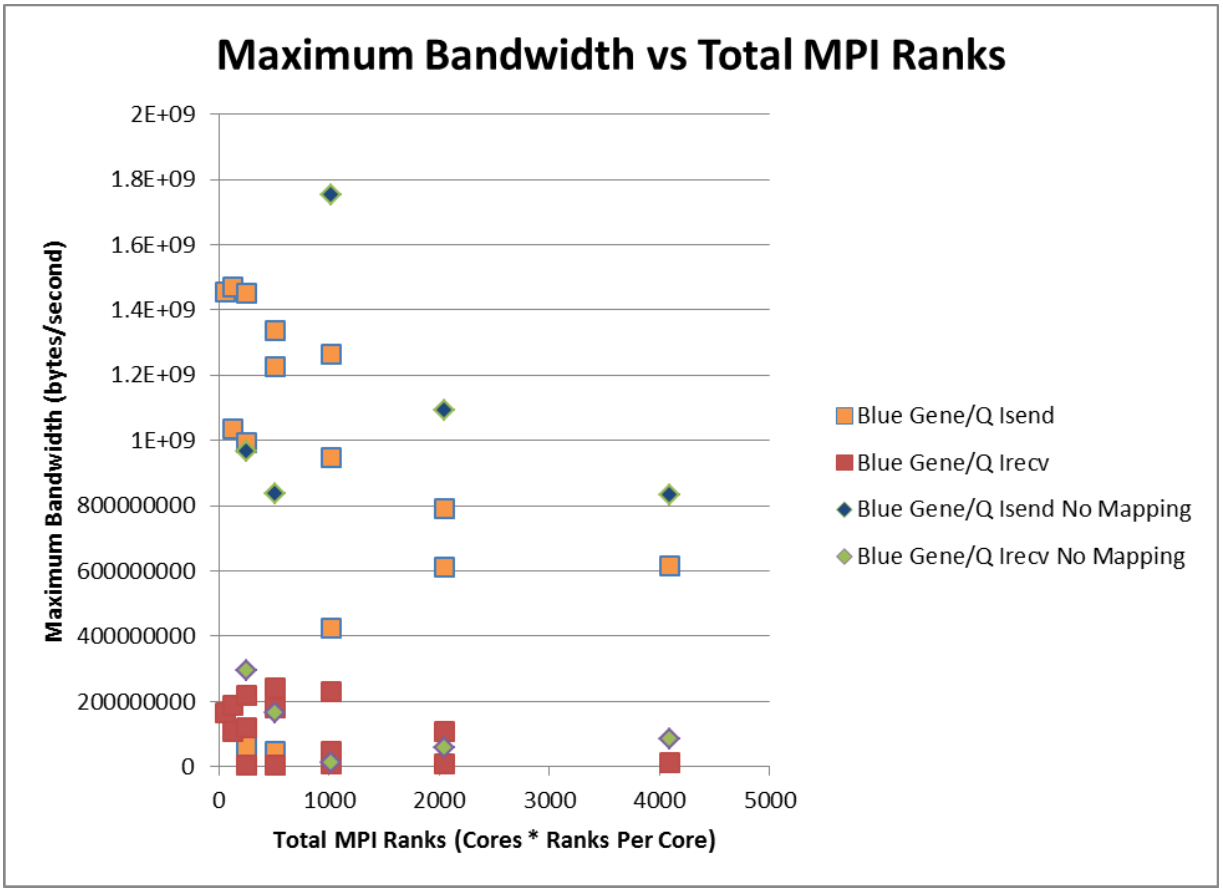
From this graph, we can see that for relatively low (less than 1024) numbers of MPI ranks, performance reliably improves as the large matrix multiplication problem is subdivided and each MPI rank has a still large portion of the matrix to calculate. Also, at these low numbers of MPI ranks, using two ranks per core and half as many cores yields about the same performance, but using four ranks per core results in a performance decrease.

At the higher counts of MPI ranks, total execution time no longer improved by further dividing the problem, due to increased overhead and communication costs.

Enabling the mapping option increased performance at low numbers of MPI ranks, but the default mapping outperformed our custom mapping at most other stages, with the best performance coming from using 4 ranks per core, no mapping option, and the maximum number of cores. My program passes messages in a circle, so I am confused by this result. Perhaps the Blue Gene/Q is optimized for the default mapping and custom mappings incur some additional overhead, requiring special applications to be useful.

Scott Todd March 17, 2013

Maximum Stats



This graph shows the maximum bandwidth over all MPI ranks, measured as the total bytes sent/received by each rank, divided by the minimum time spent processing MPI_Irecv and MPI_Isend operations (calculated using CPU cycles and the clock rate), plotted against the total number of MPI ranks used by the program.

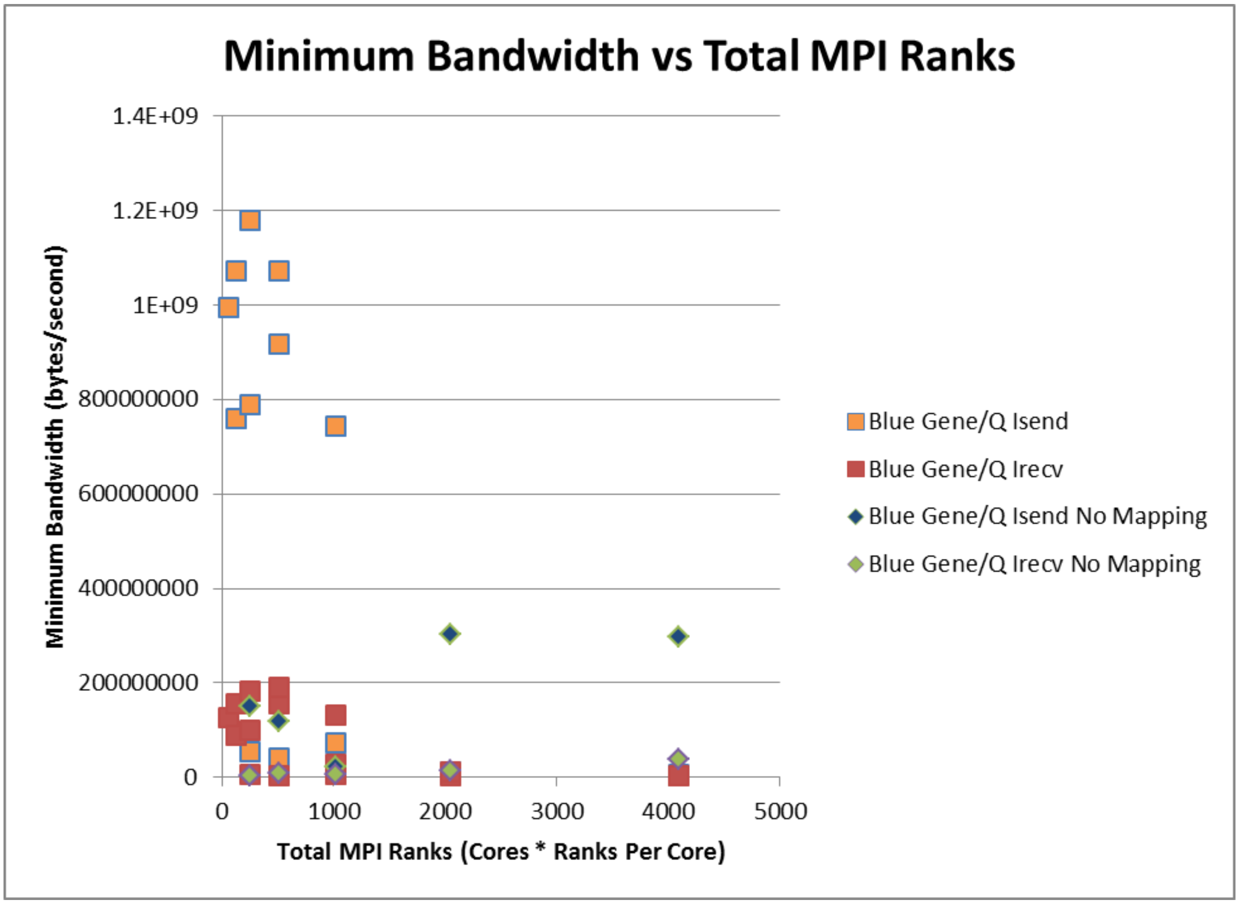
For reference, 8 cores of Kratos achieved 1.621E+09 bytes/second for Isend and 204015 bytes/second for Irecv maximum.

This graph shows that, regardless of the mapping option, Isend bandwidth exceeded Irecv bandwidth. Additionally, as MPI ranks were added, bandwidth decreased, as a larger number of smaller messages were sent.

Turning off the mapping option generally increased both Isend and Irecv bandwidth, particularly with high numbers of MPI ranks.

Scott Todd March 17, 2013

Minimum Stats



This graph shows the minimum bandwidth over all MPI ranks, measured as the total bytes sent/received by each rank, divided by the maximum time spent processing MPI_Irecv and MPI_Isend operations (calculated using CPU cycles and the clock rate), plotted against the total number of MPI ranks used by the program.

For reference, 8 cores of Kratos achieved 3485766 bytes/second for Isend and 70945 bytes/second for Irecv minimum.

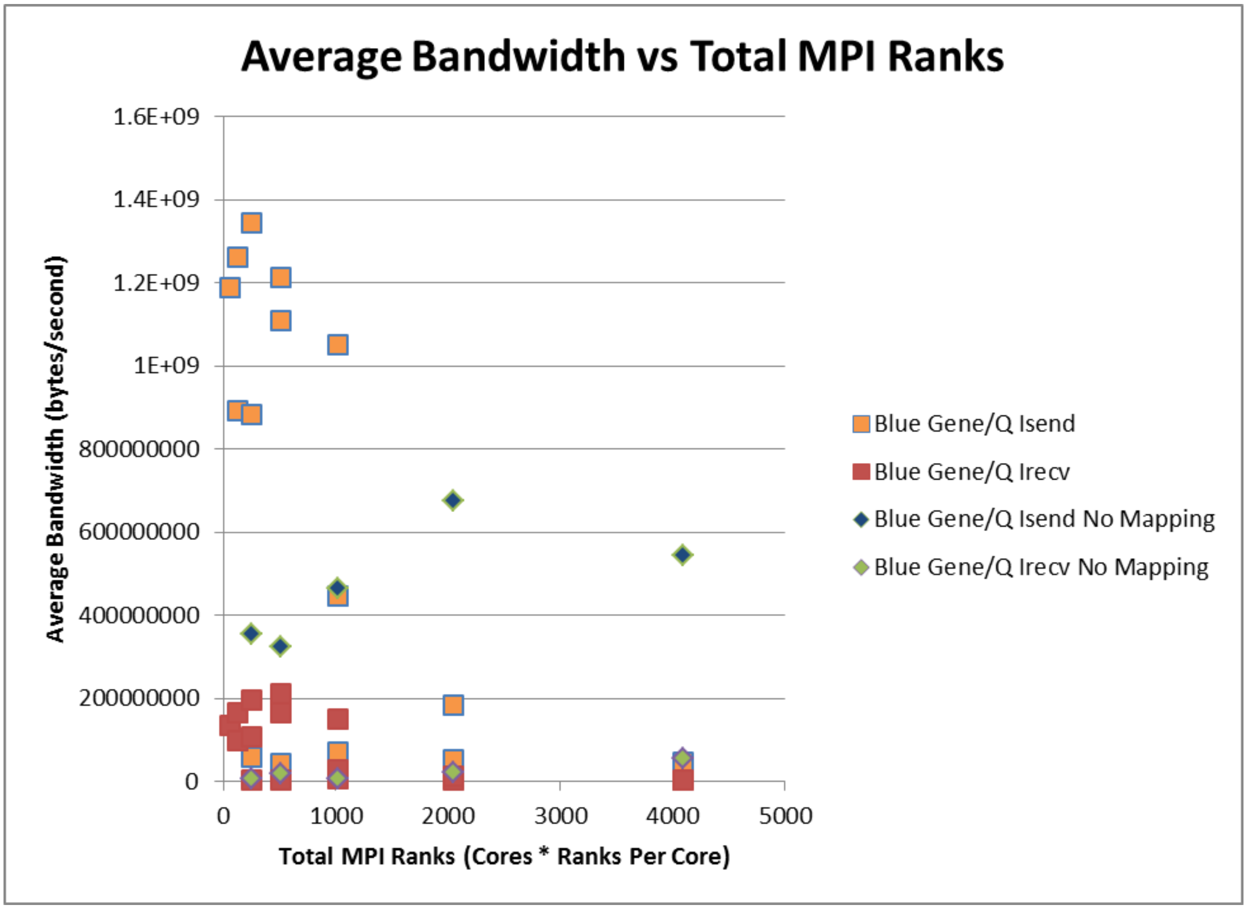
Like with the maximum stats, this graph shows that, regardless of the mapping option, Isend bandwidth exceeded Irecv bandwidth. Additionally, as MPI ranks were added, bandwidth decreased, as a larger number of smaller messages were sent.

Turning off the mapping option generally increased both Isend and Irecv bandwidth, particularly with high numbers of MPI ranks.

Compared to the maximum stats, this graph appears to have less variance (more tight clusters) and shows more of a drastic drop in bandwidth for Isend with the mapping option (the orange square data points are hidden under the red square data points at 2048 and 4096 MPI ranks). Some MPI ranks were able to maintain relatively high bandwidth at high counts of MPI ranks, while other ranks suffered from a high drop-off using the mapping option and multiple MPI ranks per core.

Scott Todd March 17, 2013

Average Stats



This graph shows the average bandwidth over all MPI ranks, measured as the total bytes sent/received by each rank, divided by the average time spent processing MPI_Irecv and MPI_Isend operations (calculated using CPU cycles and the clock rate), plotted against the total number of MPI ranks used by the program.

For reference, 8 cores of Kratos achieved 10627181 bytes/second for Isend and 113029 bytes/second for Irecv on average. I suspect that these bandwidths are much lower than those on the Blue Gene/Q due to operating system jitter and desyncing of the multiplication operations causing Irecv operations to take much longer to complete (waiting for other MPI ranks to complete their computations).

Like with the maximum and minimum stats, this graph shows that, regardless of the mapping option, Isend bandwidth exceeded Irecv bandwidth. Additionally, as MPI ranks were added, bandwidth decreased, as a larger number of smaller messages were sent.

Turning off the mapping option generally increased both Isend and Irecv bandwidth, particularly with high numbers of MPI ranks.

Like the minimum stats, this graph appears to have less variance than the maximum stats, and a sharp drop-off can be viewed for Isend bandwidth at high numbers of MPI ranks.

Scott Todd March 17, 2013

Final Comments

With the mapping option enabled, I found that it was more efficient to use additional cores than additional MPI ranks per core, and between 1024 and 2048 MPI ranks, there was a performance decrease.

At four MPI ranks per core, it became more efficient to turn off the mapping option and use the default mapping. This stemmed from a large decrease in bandwidth for some MPI ranks at high counts of MPI ranks when using our custom mapping. While some MPI ranks were still able to maintain relatively high bandwidths at high counts of MPI ranks, as indicated by the maximum graph, several MPI ranks still suffered, as indicated by the minimum graph.