Multi-Process Matrix Multiply using MPI

A Comparison Between Dense and Sparse Implementations

Jesse Bannon
University of Washington, Tacoma

July 19, 2017

In this assignment, I outline the performance characteristics between a multi-process implementation of sparse and dense matrix-matrix multiply $C = AB$ using MPI. Each implementation stores matrices row-major in memory and uses double-precision floating points. These metrics were gathered on the UW 16-node virtual cluster. Each process is ran with a single thread.

Dense Multiply The master node p_0 partitions A by consecutive rows to form submatrices A_i, and sends each process p_i both A_i and B asynchronously to compute $C_i = A_i B$. Once C_i is computed, p_i sends C_i to p_0 to its respective location to form C.

Sparse Multiply The master node p_0 partitions the sparse elements of A evenly amongst all p_i, forming A'_i. p_0 sends each process p_i A'_i and B to compute $C'_i = A'_i B$. C'_i is then collected by p_0 to compute $C = \sum_{i=0} C'_i$.

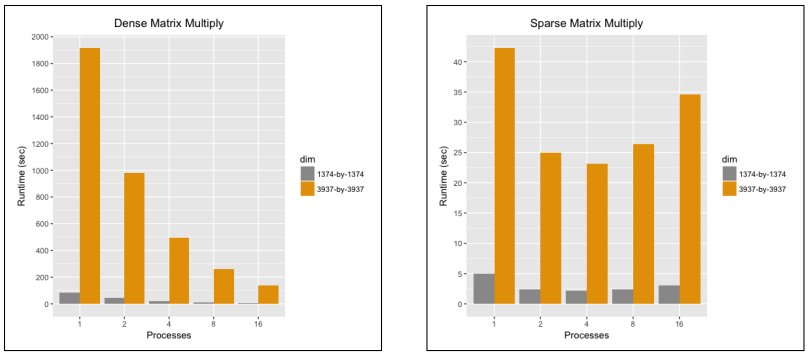

Figure 1: Elapsed time to perform dense and sparse matrix-matrix multiply using MPI

The dense multiply runtime scales linearly with respect to additional processes. The sparse multiply peaks at four nodes and degrades when adding more nodes. This is most likely caused by the serialized code executed on p_0 during the aggregation stage, when computing $C = \sum_{i=0} C'_i$. The better approach would be to perform a reduce when computing C to take all the computional burden off p_0.

Conclusion The dense matrix multiply distributed the workload successfully whereas the sparse matrix multiply suffered from a bottleneck at the aggregation stage. Despite that, I learned efficient scripting techniques to manage a homogeneous cluster.