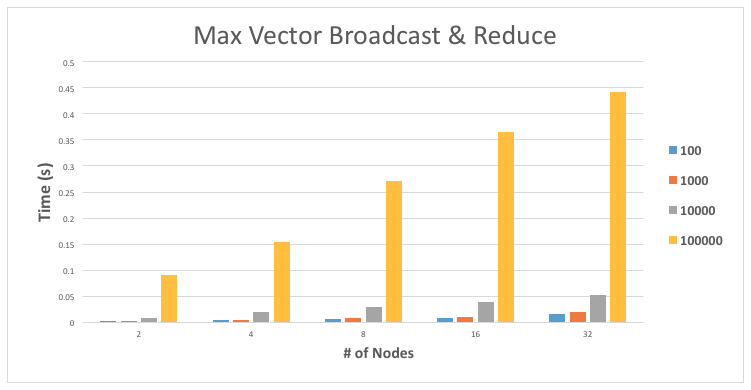
Project #3 – MPI Reduce & Broadcast  
Kyle Pontius

# Introduction

The dawn of the transistor, and with it, the computer has brought on arguably the greatest age of innovation and technological growth this world has ever seen. Few lives have remained untouched by this relatively new and extraordinary development. One computer alone holds untold power, yet tethering these machines together helps us reach new heights unachievable with single units. MPI is one of the most powerful, modern tools we have to create such a network of machines. The principle of this project is to flex MPI’s communication muscle to create a meaningful example using vector max reduce & broadcast.

# Methods

To facilitate efficient communication between our nodes, for vector max reduce & broadcast, we use the hypercube approach; which acts much like a binary tree and functions with logarithmic behavior. Broadcasts begin at the root node, then uses MPI\_Send() to message two children. This pattern repeats for each segment of the hypercube, where each child node then sends to two new children nodes of its own (hence the logarithmic growth). This is also the pattern of reduction, where two children reduce to a single parent, with each parent in turn reducing with another parent until the root node is reached.

Our goal is to create a program which does the following: First, each machine generates a new array of random numbers (size n). Second, the node takes the two arrays reduced to its position, then chooses the highest value at a given position from the two arrays, at a given position (on the first iteration, this skips directly to step 3). Third, the array reduces, or in other words, sends it’s now maximized array to the designated parent node “below” it. Finally, after all the child nodes have reduced to their parent nodes, ending up at the root node, the final array contains the maximum value for each position in the array, over all nodes in the tree.

General performance results from my algorithm are charted above. This graph depicts the execution time of different problem sizes, ranging from 100- to 100,000- sized arrays of randomly generated integers. These numbers show us some very interesting results, a few of which I’d like to touch on. First, for a given number of nodes, say 32, the execution time takes almost 10x longer for each graph respectively. In other words, the time for a 10,000-sized array to execute on 32 nodes takes 10x less than 100,000. This is interesting because it shows us that the communication time is nearly constant, or at least that there’s no significant difference between the two array sizes. Second, observe the difference in speed for each vector size, for a given number of nodes. The coefficient of growth between for problem size 1000 on 2, 4, 8, 16, and 32 nodes is nearly identical to the scale of growth on problem size 100,000 for 2, 4, 8, 16, and 32 nodes. Again, I believe this attests to the fact that communication time between the nodes is not the primary factor in increasing execution time. However, this leads us to the finals point I’d like to make.