Vyas Kattiganehalli Srinivasan & Irtaza Haider
CSE 6230

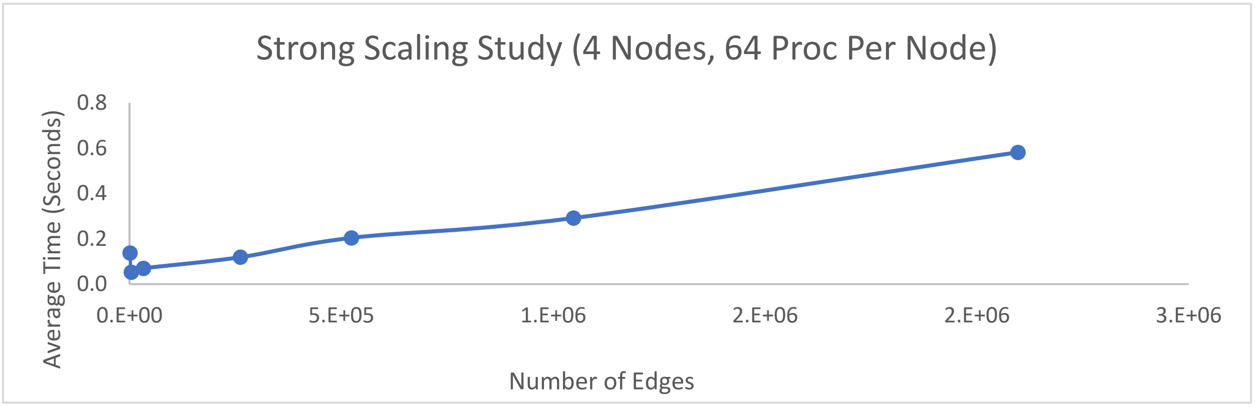
Compare the final performance of your code to the predictions you made in Checkpoint 2

We assumed in Checkpoint 2 that Stampede2 had a 2D mesh network architecture and we predicted the Alltoallv cost (i.e. the driving factor of our performance) would scale linearly with problem size:

Alltoallv communication cost = (2(sqrt(p) - 1)*alpha + p*(sqrt(p) – 1)*beta*(8*2*m/(p^2)))

Note: The Allreduce and Alltoall costs in our algorithm are constant regardless of problem size. ‘m’ is number of edges in graph, ‘p’ is number of procs, alpha is latency, beta is bandwidth

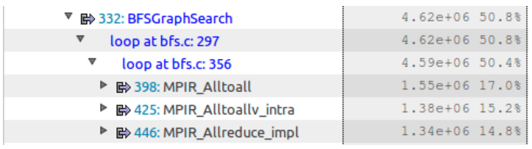
To compare our performance to our prediction, we conducted a strong scaling study:

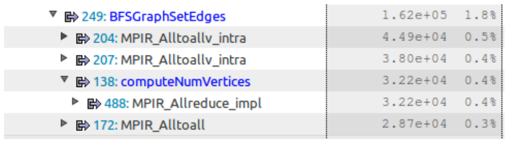


In reality, our performance does not seem to scale linearly with edges. The slight curvature of the points above suggest perhaps a x^(3/2) scaling.

Are the bottlenecks as expected? If they differ, what was wrong with your initial performance model?

Yes, the bottlenecks are as expected. We predicted in Checkpoint 2 that the bottlenecks would be the communication costs of our Alltoallv, Alltoall, and Allreduce calls. The profiling throughout our journal indicated that these three calls did in fact represent nearly all of our runtime and were unavoidable from an algorithmic standpoint:

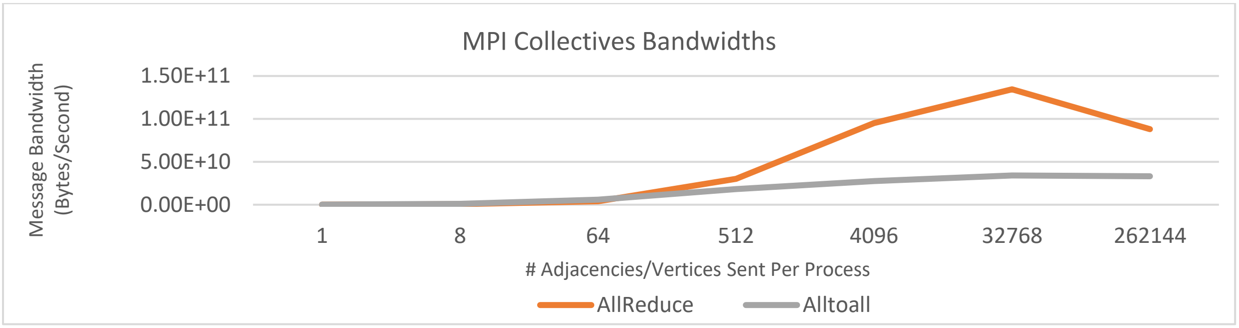

```
▼ 332: BFSGraphSearch              4.62e+06 50.8%
  ▼ loop at bfs.c: 297             4.62e+06 50.8%
    ▼ loop at bfs.c: 356           4.59e+06 50.4%
      ▶ 398: MPIR_Alltoall         1.55e+06 17.0%
      ▶ 425: MPIR_Alltoallv_intra  1.38e+06 15.2%
      ▶ 446: MPIR_Allreduce_impl   1.34e+06 14.8%
```


```
▼ 249: BFSGraphSetEdges          1.62e+05  1.8%
  ▶ 204: MPIR_Alltoallv_intra    4.49e+04  0.5%
  ▶ 207: MPIR_Alltoallv_intra    3.80e+04  0.4%
  ▼ 138: computeNumVertices      3.22e+04  0.4%
    ▶ 488: MPIR_Allreduce_impl   3.22e+04  0.4%
  ▶ 172: MPIR_Alltoall           2.87e+04  0.3%
```

We also predicted, however, that our quicksort of edges in our graph creation step would be a bottleneck. It turns out that while this O(n^2) worst case operation can be a computational bottleneck, it pales in comparison to the bottlenecks associated with communication.

Do the performance parameters (Bandwidths, flop rates) seem to differ significantly from your prediction / the reported values ? If so, why?

TACC reported that the interconnect bandwidth on Stampede2 is 100 gigabits per second (1.25e+10 B/s). Our algorithm boils down to repeated calls to MPI_Alltoall(v) and MPI_Allreduce. To determine the max communication bandwidth achieved by our calls, we ran Exercise 5's code for MPI_Alltoall and MPI_Allreduce (we used larger message sizes and 4 nodes 64 procs per node to produce the below chart):



We get better bandwidths than what TACC says is possible. This is because the 100 Gb/s is the bandwidth between nodes. Though we have 256 processes running, there are only 4 nodes on which they run (64 proc per node) – Communicating between procs on the same node occurs at a much higher bandwidth than 100 Gb/s. Thus, the bandwidths we observe are some blend of network bandwidth and on-node bandwidth that averages to >100 Gb/s. The differences in the bandwidths between Allreduce and Alltoall likely results from differences in how often each call uses the network to communicate (i.e. Allreduce can run reductions for each node first and then communicate these 4 values between nodes across the network).

If there are optimizations you would like to make but don't have time for, explain why you think they would work here.

1. Instead of doing the blocking MPI_Alltoallv, have each process do a series of p-1 MPI_Sendrecv's followed by an MPI_Waitsome. Rather than forcing every process to wait for the full Alltoallv to complete, we could instead overlap processing incoming edges and communication using the above strategy. This lets us also explicitly avoid including processes who send/receive nothing from joining the communication.
2. Try a 2D partitioning of the graph rather than a 1D partitioning of the vertices. By using a 2D partition (i.e. a group of processes collectively own all the edges of a given vertex), we could reduce the number of processes involved in a collective communication from |p| processes to sqrt(|p|) processes. This would reduce overall communication costs.