High Performance Computing (SS13)
Assignment for Block 3: MPI library implementation

Jesper Larsson Träff, Sascha Hunold, Francesco Versaci
TU Wien

April 29, 2013

The goal of this assignment is to implement three broadcast algorithms for MPI, benchmark them under different circumstances to verify/falsify the claims of the lecture, and estimate for each of the implementations the parameter range(s) (processes, data sizes) for which this algorithm/implementation is the better of the three. The ultimate goal (you do not have to accomplish this) would be to use the three implementations together in an `MPI_Bcast` implementation that behaves smoothly and well over the whole range of data sizes and processes.

For all solutions, data may be assumed to be buffers of consecutive integers or floats, i.e., MPI datatypes describing structured buffers can be ignored. The implementations should, on the other hand, work correctly for any data sizes and any number of processes.

The implementations should be benchmarked on the Jupiter system, if possible with all nodes and cores of the system (36×16). Use the NEC MPI library. State compilation flags and other information nececessary to make your experiments reproducible.

Advice: Start small as long as the implementations are not correct (especially: deadlock-free)

Linear pipeline

Implement the simple linear pipeline algorithm. Estimate the "best block size" using measured α and β, experiment with different block sizes, and determine a value (or values) that seem to perform well in practice. Compare the time to broadcast with `MPI_Bcast`.

Binomial tree

Implement the binomial tree ("minimum spanning tree") algorithm. Compare the time to broadcast with `MPI_Bcast`.

Pipelined binary tree

Implement the pipelined binary tree algorithm, and try to determine a best block size. Compare the time to broadcast with `MPI_Bcast`.

Benchmarking and discussion

Benchmark (for each data size and each number of processes) by recording the minimum of the time for the slowest ("last") process over a number of iterations. Benchmark over a range of values that are not only powers of two.

For all implementations, use both a well-behaved communicator - consecutive ranks inside the SMP nodes of Jupiter - like `MPI_COMM_WORLD`, and a communicator where many processes on a node at the same time would have to communicate with processes at other nodes; for instance a random communicator. Discuss how to avoid such performance problems, and which means MPI provides - however, you do not have to implement this.

Submission deadline: Monday, 24.6, 24:00