MPI: Practical session 4

1 Send/Receive

Write a program that does the following, using `MPI_Send` and `MPI_Recv`

- On process zero, initialize an integer v with value 0.
- Send v from process 0 to process 1, and on process 1, add 1. Display the value of v.
- ...
- Send v from process n to process $n+1$, and on process $n+1$, add n. Display the value of v.
- ...
- Finally send this integer back to process zero, and print out its final value.

The final value of v should be $\frac{N(N-1)}{2}$. Check that this is the case.

Of course, this is a very contrived example, and would not be useful in practical code because of the inherent serialization that results.

2 Reduce

You have previously used `MPI_Reduce` to sum a value across multiple processes. Now, reimplement this using `MPI_Send` and `MPI_Recv` alone.

Implement this using both of the following approaches separately:

2.1 Sum on root

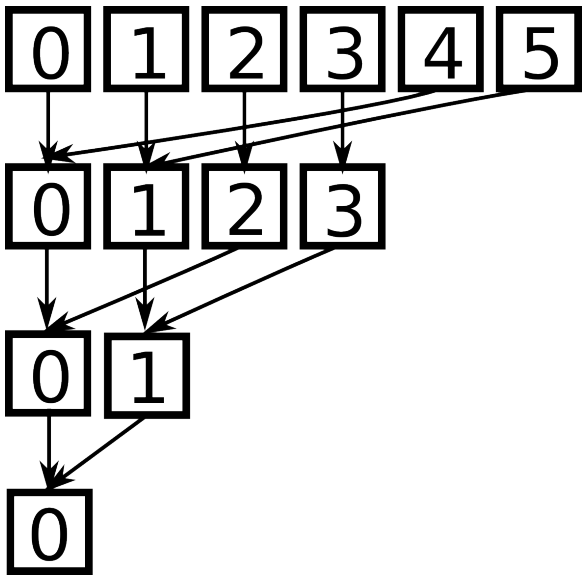
- Send each value from process n to process 0 and perform the sum on process 0.

2.2 Tree-based sum

Compute $p = \lfloor \log_2 N \rfloor$.

- On process n find the sum of the total from process n and process $n + 2^p$.
- On process n find the sum of the total from process n and process $n + 2^{p-1}$.
- Repeat until the final total is on process 0 and print it out.

To illustrate, for 6 processes, and where the arrows denote which pairs of processors' values are added together:



2.3 Comparison

Which of these two appproaches do you think is more efficient? If it helps, imagine different computers with wildly different costs for communication between processes and for performing the addition.

This thought experiment (and the effort involved in getting the tree reduction correct) suggests that you should always use MPI's collective operations where possible. The MPI implementation writers will have spent much more time optimizing the collective operations for various numbers of processors, interconnect, etc. than you are able to.