Designing and building applications for extreme scale systems
CS598 SP2016
HW9: Communication and Computation with MPI

Goals

- Gain experience with MPI collective communication
- Explore communication / computation overlap

Tasks

Implement a parallel matrix-vector multiply. The matrix is dense, the total size is M x M, and the matrix is stored by rows (C order).
The decomposition of matrix is by rows; the decomposition of the vector matches that of the matrix. For example, if there p processes and M = kp, then the process with rank r has rows kr through k(r+1)-1 of both the matrix and the vector.

To perform computation, gather the vector elements to each process and perform the multiplication. This can be done is several ways. The two that you should implement for this assignment are:

1. MPI_Allgather. Gather the entire vector up on each process

2. Circulate the vector elements in a ring. Each process sends to the process with rank = rank + 1 mode wsize and receives from the process with rank = rank – 1 + wsize mod wsize. Use nonblocking communication. That is, Ireceive the next block, Isend the current block, then compute with the current block. After the computation, wait on the nonblocking communication. This is a form of *double buffering*, and *permits* the MPI implementation to overlap communication and computation. Roughly, the code should look something like this:

```
for (i=0; i<wsize; i++) {
  MPI_Irecv(xbufnext, ln, MPI_DOUBLE, (rank + 1) % wsize, …, &r[0]);
  MPI_Isend(xbufcur, ln, MPI_DOUBLE,  (rank + wsize – 1) % wsize, …, &r[1]);
  mvmult(correct-block-of-mat, xbufcur, y, sizes…);
  MPI_Waitall(2, r, MPI_STATUSES_IGNORE);
  xtemp = xbufcur; xbufcur = xbufnext; xbufnext = xtemp;  // swap pointers
}
```

Where “mvmult” is a local matrix-vector multiply routine, and “correct-block-of-mat” refers to the part of the matrix that corresponds to the part of the vector that is stored in the storage pointed at by xbufcur. Note that this example is incomplete and is intended only to illustrate the use of two

buffers to overlap communication with computation. Your code will be somewhat different.

For each, also compute a simple performance expectation using the following assumptions. What is required for the assignment is just this:

1. Estimate the cost of the matrix-vector product on each process.

2. Estimate the cost of communication by ignoring the latency term (the "s" in the "s + rn" model) and just consider the amount of data that is moved (the "rn" term). Use the value of "r" that you found in the previous homework, or if you are using a different system, run a simple test case to get a value for "r".

Use the following test cases.

Wsize is the number of processes in MPI_COMM_WORLD, which should be 128, 256, 512, 1024 (if you can't use Blue Waters, use as many processes as possible upto 1024). Use 1 process per floating-point core; on Blue Waters, this is16 processes per node. Run tests for these different data sizes: M = 1024, 16384, 102400

When you measure time for these operations, ensure that you (a) measure a long enough time to be significant (use MPI_Wtick) and (b) make at least 5 separate measurements for each test. For each test, compute the maximum over all processes. Repeat that test at least 5 times and report the minimum time taken for the 5 measurements (this is the most likely to be repeatable).

When you run on Blue Waters, you need to pass two environment variables to enable the overlap of communication with computation. If you use the mpiexec script, the following will set the environment correctly:

```
mpiexec –n 256 –ppn 16 -env MPICH_NEMESIS_ASYNC_PROGRESS=1 \
-env MPICH_MAX_THREAD_SAFETY=multiple ../myhw9 16384
```

Submit
A PDF file containing table & plots for the results of the tests. A brief discussion is necessary: do the results match your expectations? In particular, is either communication approach always the best choice?

To think about (but not turn in)

1. Try different neighbors for the case of communication in a ring:

a. Choose neighbor: rank + 1 in comm world;
b. rank + 16 mod wsize in MPI_COMM_WORLD (running with 16 cores/node)
c. Another of your choosing (explain why)

2. How do you think the performance would differ if you used blocking communication instead of nonblocking communication in the ring?

3. MPI-3 adds nonblocking collective routines. How could you use MPI_Iallgather and still overlap communication and computation?

4. Refine the performance expectations. There are many different algorithms for MPI_Allgather. Two simple ones are:

    a. Recursive doubling, where at each step, twice as much data is sent as in the previous step. If step i has cost $(s + r2^i n_0)$ and there are log p steps, what is the total time for this algorithm? Here, n_0 is M/p (why?)
    b. Digital orrery. This is just the ring approach (method 2) above. If each step has cost (s +rM/p), what is the total cost? How might you take the overlap of communication and computation into account?

5. How would you take process topology into account in the ring communication approach?