Homework 2 - Parallel Programming for Large Scale Problems - SF2568

Gabriel Carrizo

December 2017

1 Broadcast Operation

1. Design an algorithm for the broadcast operation only using point-to-point comunication.

Solution: The brodcast operation is an operation that sends the information of one node to all other processes. Using recursive doubling one can 'spread' amongst processes, much like an infection moves through a host.

The idea is that the process with the data sends its data to another process. Once that process also possesses the data, both processes send the data to two other processes. Now we have four processes with data and all four can send data to four other processes. This procedure repeats until all processes posses the data (see figure 1).

The complexity of the solution in parallel is dependent on the number of processors used.

See Listing 3 for pseudoecode. (Made under the assumption that $P = 2^D$)

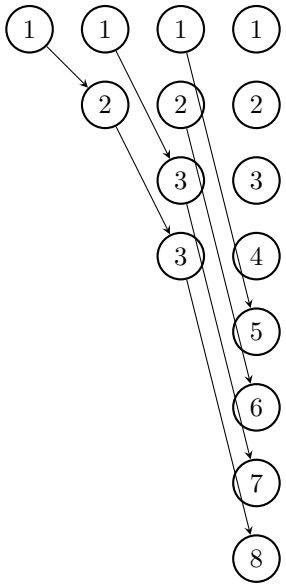

Figure 1: Graph representation of broadcasting with recursive doubling for $P = 8$.

```
"""
Pseudocode for broadcast operation using only
point-to-point operations.

Note that the elif statement triggers before the
if statement for every process except for process 0.
"""

x;
for d = 0->D-1:
    dest = bitflip(rank,d) #send or recieve destination
    if (rank < 2^d): #if rank is less than 2^d we send to dest
        send(x, dest)
    elif (rank < 2^(d+1)): #elif rank is less than 2^(d+1) we recieve.
        recieve(x, dest) #overwrite x
```

Listing 1: Pseudocode for broadcasting operation.

2. Do a (time-)performance analysis for your algorithm.

Solution: For this implementation we are looking at:

Local computation time:

$t_{comp,1} = 1t_a$ (where t_a is the time for one basic operation) for allocating memory for x.

Recursive doubling step:

$t = D(t_{startup} + w_p t_{data} + 3t_a) = \log P(t_{startup} + wt_{data} + 3t_a)$

Where $3t_a$ is from the (at most) two if statement evaluations and the computation of the destination.

Total computation time:

$T_p = t_{comp,1} + t = t_a + \log P(t_{startup} + wt_{data} + 3t_a)$

3. How can the scatter operation be implemented using $O(logP)$ communication steps?

Solution: You could broadcast the data and have each process select its portion of the data, however this would result in high memory overhead.

A solution to this is that the ground process sends the upper half the data to the middle node by performing the bitflip in reverse, from highest bit to lowest bit. This way, if memory management is done properly there wont be any memory overhead. (See figure 2)

2 Transpose

Solution: Assuming that what the algorithm from class leaves us with a grid with load balanced data distribution where each process has address $P[row, col]$:

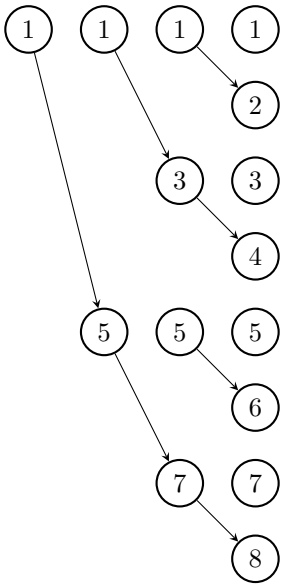

Figure 2: Graph representation of scatter with recursive doubling for $P = 8$.

$$M = \begin{bmatrix} y_0 & y_0 & \dots & y_0 \\ y_1 & y_1 & \dots & y_1 \\ \dots & \dots & \dots & \dots \\ y_P & y_P & \dots & y_P \end{bmatrix}$$

Which we want to transpoe into:

$$M^T = \begin{bmatrix} y_0 & y_1 & \dots & y_P \\ y_0 & y_1 & \dots & y_P \\ \dots & \dots & \dots & \dots \\ y_0 & y_1 & \dots & y_P \end{bmatrix}$$

There are two approaches that work with the broadcast algorithm suggested in the previous section.

The first one is to send the data from the first column to the first row, such that P[0,1] sends to P[1,0], P[0,2] send to P[2,0], etc. Once the first row is filled with its initial value the broadcast be performed, just like in the previous section but for each column individually.

The second approach I will suggest is to perform a cyclic shift for the rows of each column s.t. every diagonal element is shifted to the the first row of matrix M. This method requires some adjustments so the processes send and recieve to and from the correct index (see Listing 2)

The first approach requires an extra communication step and will also require sending data to some processes on the diagonal that already posses the data. The second approach requires some extra computation steps but should overall be quicker since it requires fewer communication steps.

```
"""
Builds on broadcast function from previous part.

Broadcasts values such as to transpose y.

Performs cyclic shift on all rows of each column such that the diagonal value
    is on the first row

Processes are distributed on a square matrix [P x P]

Each process will posses its location as location(p) = (row,col)

"""

x;
#--------- Main change from last section -----
new_row = row - column #shift rows in process

if new_row < 0:
    new_row = new_row + P #if row is negative we need to adjust with P for
    cyclic shift.

for d 0->D-1:

    if new < 2^d:
        dest = col + bitflip(new_row, d) #shifted bitflip

    if dest >= P:
        dest = dest - P #adjust for cyclic shift
#-----------------------------------------------

    if (new_row < 2^d): #if row is less than 2^d we send to dest
        send(x, dest, column)
    elif (new_row < 2^(d+1)): #elif row is less than 2^(d+1) we recieve.
        recieve(x, dest, column)
```

Listing 2: Pseudocode for transpose with the broadcast operation operation.

4. Performance analysis:

   The first approach would require an additional communication step in the beginning which would result in the following performance:

   $T_p = t_{comp,1} + t = t_a + \log P(t_{startup} + wt_{data} + 3t_a) + t_{startup} + t_{data}$

   The second approach (seen in psuedocode in Listings 2) would add one additional computation for the cyclic shifts of the process address and the recipient address. Luckily, this approach requires just as many communication steps as the broadcast approach in section 1.

   $t_{comp,1} = 4t_a$

   Recursive doubling:

   $t = \log P(t_{startup} + wt_{data} + 6t_a)$

   Total:

   $T_P = t_{comp,1} + t = 4t_a + \log P(t_{startup} + wt_{data} + 6t_a)$

3 Solving DE with Poisson - Jacobi Iterations

In this section the data was linearly distributed over the processes. Each process would need the results of its neighbouring processes and this problem was solved using red-black communication. This procedure ensures that all processes communicate with each other in the correct order such that there is no deadlocking.

In the code, red-black communication looks like this:

```
1
2  if (color == 0) {
3    if (p != P-1) {
4      MPI_Send(&u[I], 1, MPI_DOUBLE, p+1, tag, MPI_COMM_WORLD);
5      MPI_Recv(&u[I+1], 1, MPI_DOUBLE, p+1, tag, MPI_COMM_WORLD, &status);
6    }
7    if (p != 0) {
8      MPI_Send(&u[1], 1, MPI_DOUBLE, p-1, tag, MPI_COMM_WORLD);
9      MPI_Recv(&u[0], 1, MPI_DOUBLE, p-1, tag, MPI_COMM_WORLD, &status);
10   }
11 }else{
12   MPI_Recv(&u[0], 1, MPI_DOUBLE, p-1, tag, MPI_COMM_WORLD, &status);
13   MPI_Send(&u[1], 1, MPI_DOUBLE, p-1, tag, MPI_COMM_WORLD);
14   if (p != P-1) {
15     MPI_Recv(&u[I+1], 1, MPI_DOUBLE, p+1, tag, MPI_COMM_WORLD, &status);
16     MPI_Send(&u[I], 1, MPI_DOUBLE, p+1, tag, MPI_COMM_WORLD);
17   }
18 }
```

Listing 3: Red-black communication.

Before the implementation could begin functions $f(x)$ and $r(x)$ were computed such that:

$$f(x) = u'' - r(x)u \tag{1}$$

and

$$r(x) \leq 0 \; \forall \; x \; \epsilon \; [0, 1] \tag{2}$$

with a u that satisfies the boundary conditions:

$$u(0) = u(1) = 0 \tag{3}$$

$$u(x) = x(x-1)$$

Which gives us the following $f(x)$:

$$f(x) = 2 - x^2(x-1) \tag{4}$$

The following result was obtained for $1e6$ iterations on 8 processors:

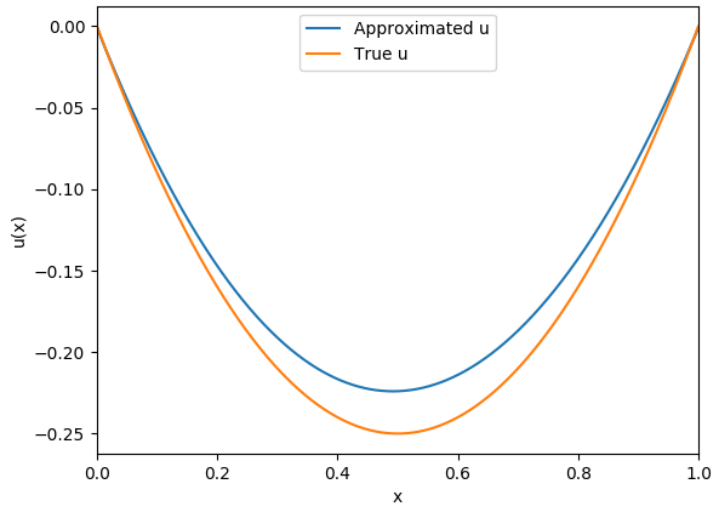

Figure 3: Output generated from our poisson equation solver generated by jacobi iterations compared to true solution.