**Masterpraktikum Scientific Computing – High Performance Computing**

Gaurav Kukreja, Evangelos Drossos

**Exercise Sheet 3**

**Exercise 9**

In this exercise we implemented several versions of a broadcasting operation using MPI. We also run a number of tests to find out how each algorithm performs and we used MPI\_Bcast as a benchmark (broadcast.c). In the following, size denotes the total number of MPI tasks.

1. The trivial algorithm (trivial\_broadcast.c) lets the root process to send the array of length N to all other processes. It involves (size-1) send operations of N\*64 bytes each. For a standard array length of N = 1000 and 8 processes this translates to a message size of and a total data transfer of .

The critical path is of length 2 and N elements are sent through it.

1. The tree algorithm sends data according to a tree structure. The root process starts by sending the array to process 1. In the next iteration, the root process sends the data to process 2 and process 1 sends to process 3. Figure 1 shows the data transmission tree in the case of 8 processors. The total number of messages amounts to (size-1) and each message contains the whole array. The critical path has a length of and the amount of data sent over the critical path equals .

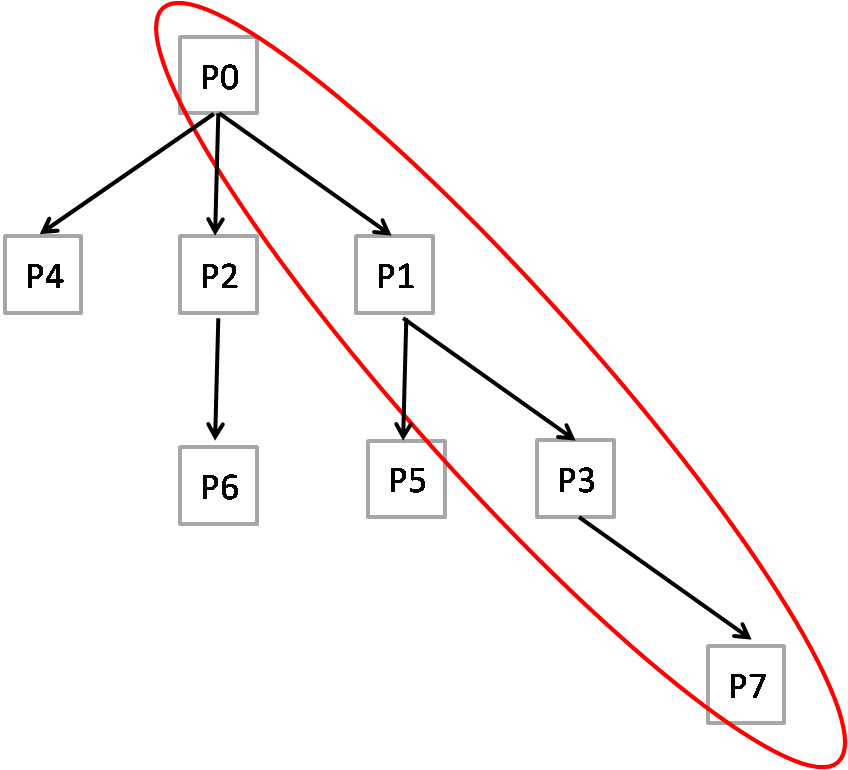


Figure 1 – tree\_broadcast data transmission tree for 8 processors. Critical path is circled in red.

1. The bonus method required that each process send at most elements and the number of elements sent on the critical path to be as well. We provide two similar implementations for this part, the bonus\_broadcast and the scatter\_broadcast algorithms.

The bonus\_broadcast algorithm starts by dividing the array of N elements into p equal parts , where p is the number of processors (p = size). If N is not divisible by p, then the array is resized through padding, with zero-valued elements added in the end of the array. Let denote the adjusted length of the array. In the first step the root process sends an equal sized but different part of the array to each of the other processes. Process 1 receives subarray , which is of length starting elements after the the beginning of the original array. Subsequently, all other processes i receive the corresponding subarray . A total of messages of size each have to be sent. An intermediate step is taken to place the received subarray in the correct order. The resulting allocation is demonstrated in figure 2:

Figure 2 – bonus\_broadcast: Result of step 1 in the case of 4 processors

The next step involves sending all subarrays to all processors that haven’t received them yet. The root process sends to all other processes. Subsequently each process i sends each part of the array to all other processes for a total of messages of length per processor. In the end, all processes posses the whole array of length . Figure 3 below demonstrates how the algorithm works.

The total number of MPI messages sent equals . The critical path is of length 2 (the algorithm proceeds in two steps) and the amount of data sent over it is equal to for a constant number of processors.

Figure 3 – bonus\_broadcast tree for 4 processors. Message length is the same for a level of the tree. Critical path is marked in red

The root process sends elements during the entire course of the program.

Every other process with rank sends elements.

A similar approach is followed in the implementation of the scatter\_broadcast algorithm. This algorithm also proceeds in two steps. In the first step the root process performs a scatter operation and in this second step an MPI\_Allgather operation is performed to gather all parts of the array scattered to the different processes and reconstruct the area in the order of the rank of the processes sending the messages. The MPI\_Allgather operation is equivalent to sending messages from each processor and arranging the received messages accordingly. It can be easily seen that this algorithm has many similarities to the bonus\_broadcast algorithm and also fulfills the criteria set in the exercise.

Figure 4 – scatter\_broadcast operations and data transfers in the case of 4 processors

Bandwidth measurement of different implementations:

Figure 5 – Bandwidth (in B) to number of tasks for N = 200,000

Figure 6 - Bandwidth (in B) to number of tasks for N = 100,000

Sidenote: the scatter\_broadcast algorithm does not seem to terminate when run with certain combinations of number of tasks and array size. We were able to get results for a large array size (N = 200000) and more than 16 tasks as well as for a smaller array size (N <= 10000 approximately) independent of the number of tasks. This is the reason why some values for this implementation were omitted.