Programming Assignment 2: CSE6220A – Report

Sambhav Mattoo, Vishal Hariharan, Gautham Gururajan

Let's first discuss the implementation details of both our Broadcast function and the Prefix sum calculation. In the Broadcast part, we follow exactly the procedure laid out in topic-8 communication primitives, slide 9 for reference; we define the mask, the flip and the exact operations in the loop exactly as defined therein, with the number of iterations d being log of the number of processes to the base 2, as clear from the send/ receive diagram.

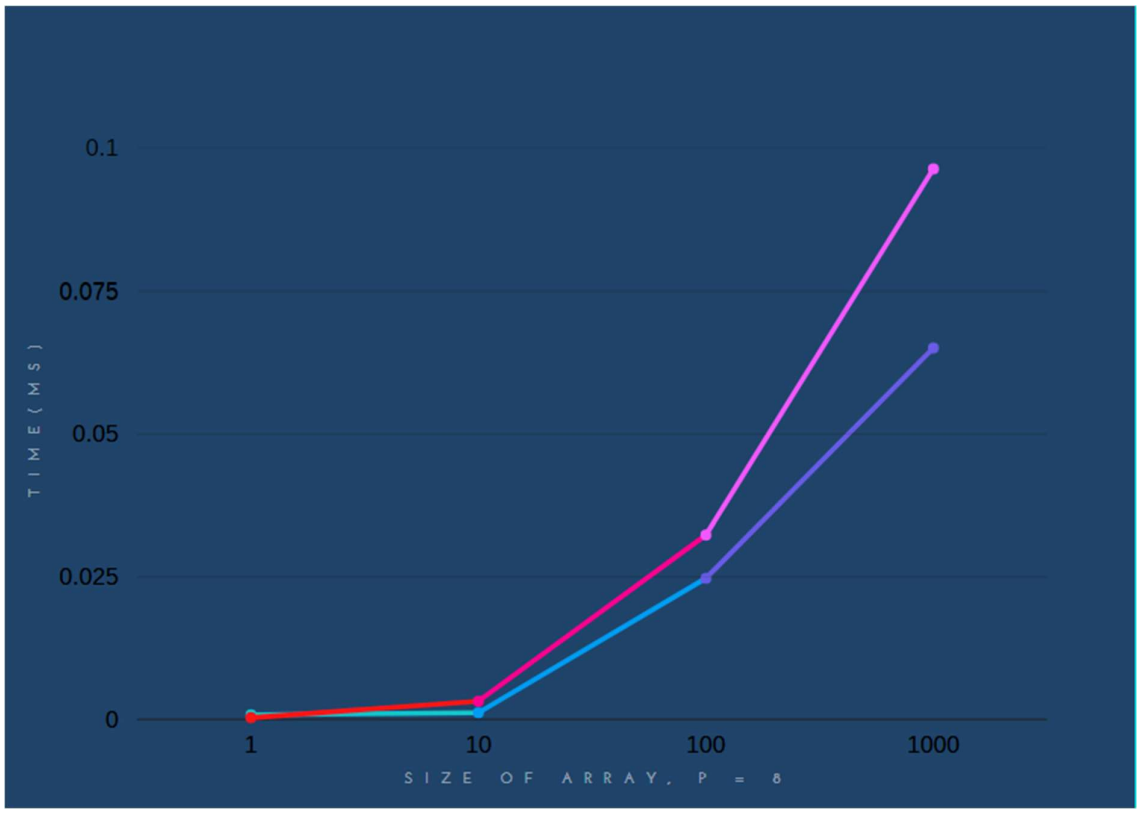
```
Algorithm (for P_rank)
flip ← 1 << (d-1)    // flip is 2^(d-1)
mask ← flip - 1
for j=d-1 to 0 do
   if (((rank XOR root) AND mask) == 0)
      if (((rank XOR root) AND flip) == 0)
         send x to (rank XOR flip)
       else
         receive x from (rank XOR flip)
   mask ← mask >> 1
   flip ← flip >> 1
endfor
```

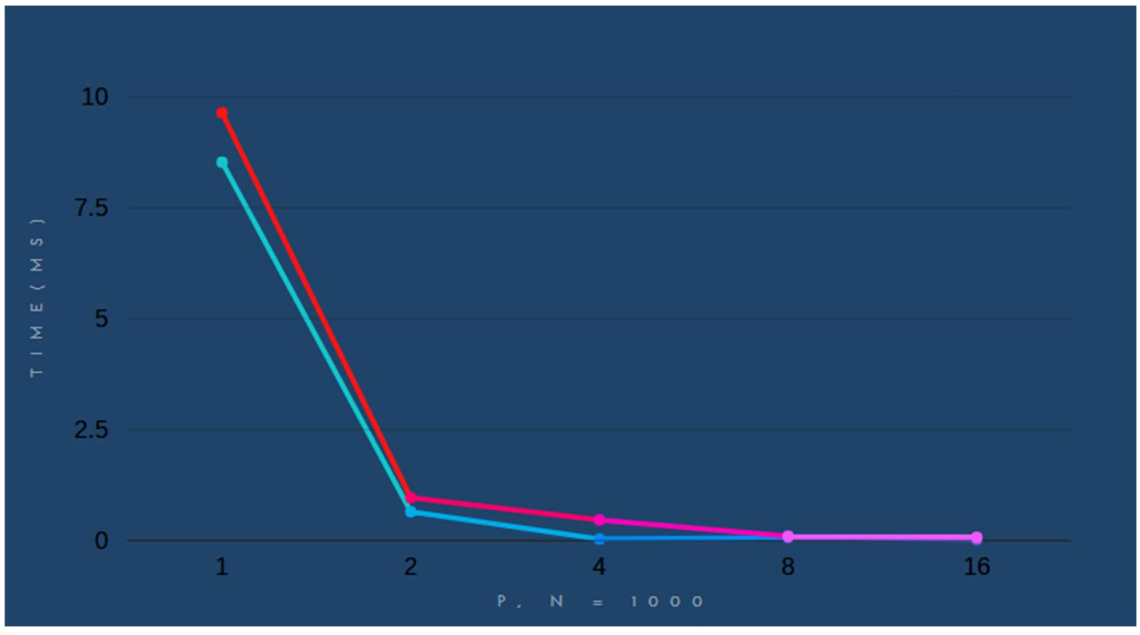
```
void HPC_Bcast(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm) {
  // TODO: Implement this function using only sends and receives for communication instead of MPI_Bcast.

  int world_rank;
  MPI_Comm_rank(comm, &world_rank);

  int world_size;
  MPI_Comm_size(comm, &world_size);

  MPI_Status status;
  int d = log2(world_size);
  int flip = 1 << (d-1);
  int mask = flip - 1;

  for(int j=d-1; j>=0; j--)
  {
      if (((world_rank ^ root) & mask) == 0)
      {
          int pair = world_rank ^ flip;
          if (((world_rank ^ root) & flip) == 0)
          {
              MPI_Send(buffer, count, datatype, pair, 111, comm);
          }
          else
          {
              MPI_Recv(buffer, count, datatype, pair, 111, comm, &status);
          }
      }
      mask = mask >> 1;
      flip = flip >> 1;
  }

  //MPI_Bcast(buffer, count, datatype, root, comm);
}
```

As you can see, they are quite similar line by line to the code explained by the professor in class. Coming to the prefix sum part, after initializing the MPI parameters (rank and world size), we do the approximate procedure laid down in the third algorithm of the prefix sum slides: we defined d = ceil(log2(world_size)). This in essence handles the case where p is not a power of 2: by doing an extra iteration to account for any "out of bound of power of 2" cases individually. Next, we initialize the total sum to the first work buffer, by invoking the prefix function of our choice (which, note, must be associative). We also use a counter to keep track of if we are updating the prefix sum for the first time or not. We then perform parallel prefix sum using the last local prefix

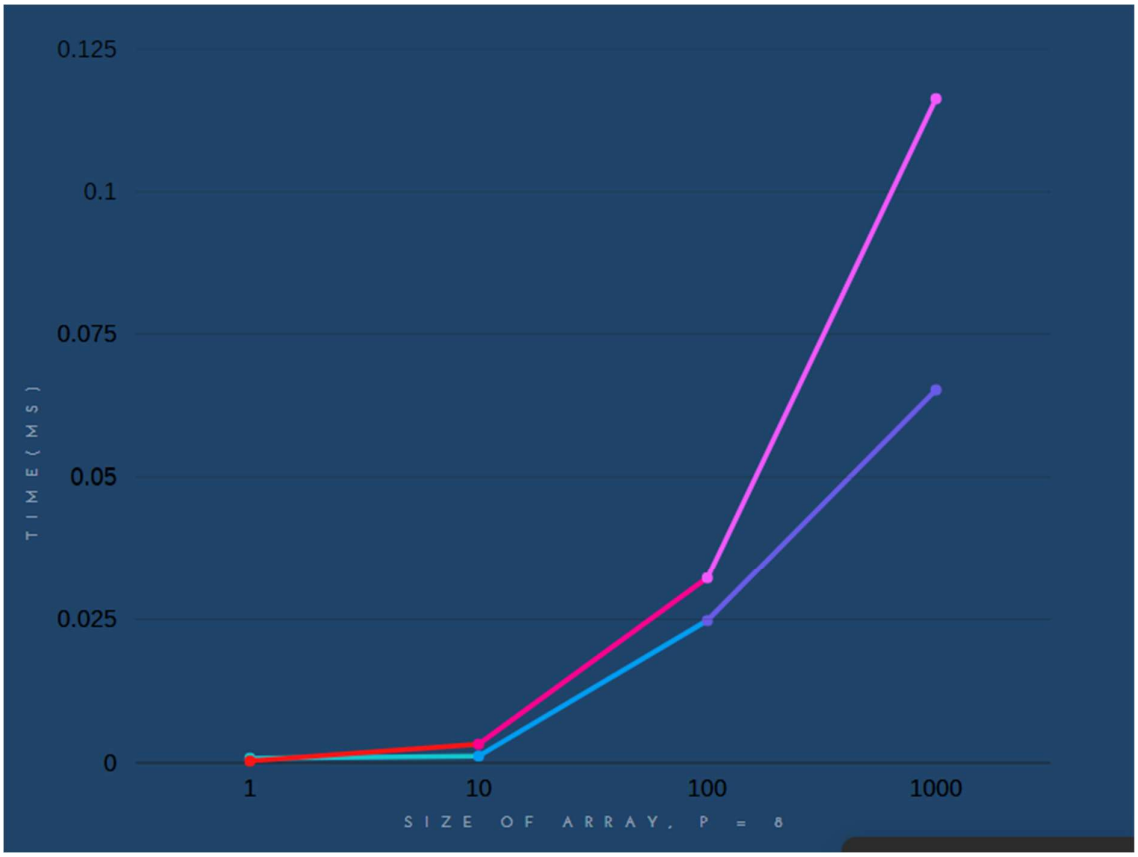
sum on each processor, by firstly calculating new_rank, i.e., a corresponding pair to send and receive from. We do the communication only if the processor in note exists, that is, if both new_rank and world_rank are within world_size. We then must carry out the sends and receives from our defined communicating pairs into out work buffers, with an if conditional before combining these using our prefix function, to handle the non-commutative case. We then initialize prefix to first element of the received data I*x = x and increment our counter. Finally, in the loop once the prefix has been initialized keep updating it with the received data. All in all, it follows the Alg-3 as defined in class quite exactly.

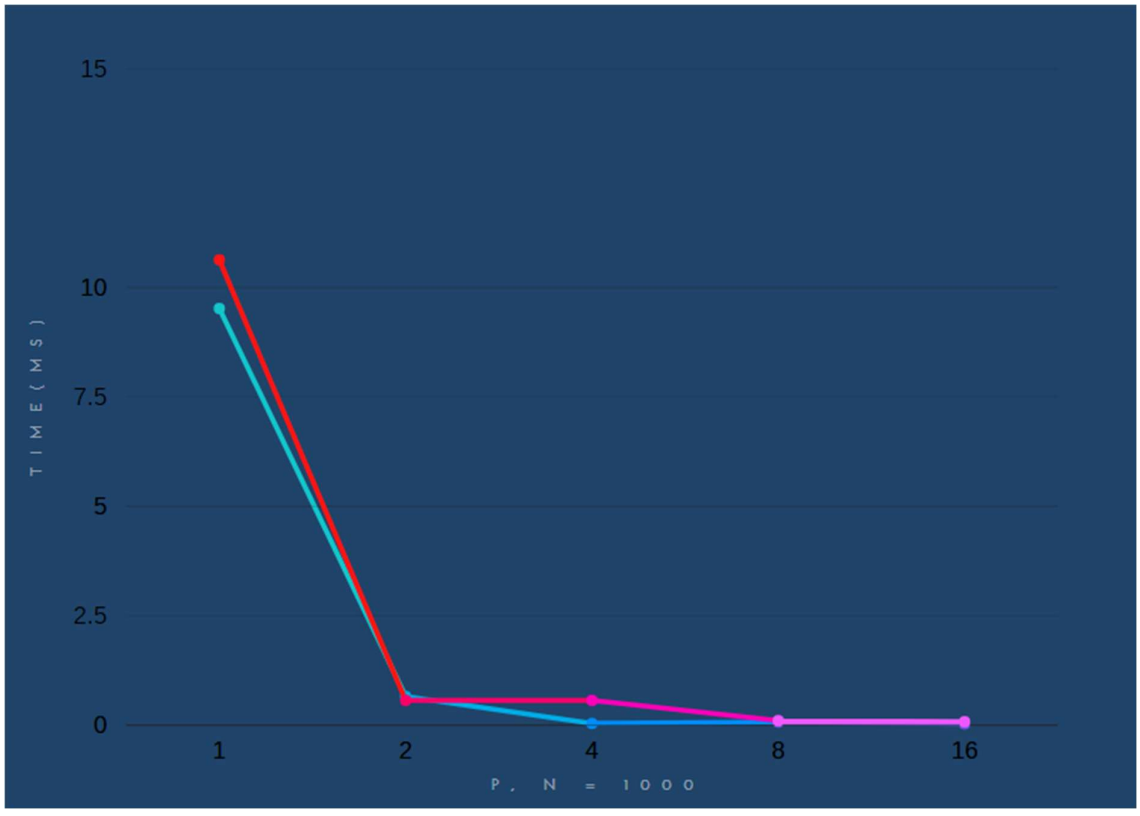
Now coming to the testing, we did 4 tests with N= 1, 10, 100, 1000 for P fixed at 8, and P = 1, 2, 4, 8, 16 for N fixed at 1000, for our functions (Blue) and MPI's functions (Red). The results for Broadcast were:





And those for prefix sum were:





Wee see something odd and unexpected here: while the response for both is normal exponential with respect to n and p i.e., grows with increase in n and sharply decreases with increase in parallelism, but our implementations are slightly faster than MPIs! The general reason this could be the case is because MPI_Bcast and Scan both pick out of a bunch of given algorithms to be globally synchronous based on some heuristics that somehow were either incorrect or capped out with respect to performance here, making the MPI code as good as our code or even slightly worse. The exact cause of this requires a deeper investigation of said heuristics with our knowledge of the trial input. There is also a discontinuity with our expected pattern at p = 4 for fixed n cases: again we have some hypothesis on why this could be occurring, namely, the 2-4 transition in the hypercubes dimensions being easier for sends/ receives to scale for. However, save these features, the plots are all in all congruent with our theoretical expectations of runtimes as we discussed in class. My largest test was of course N = 1000 on P = 16, which took only ~0.002 s approximately for both prefix sum and broadcasts.