**Report: Exercises 2**

Sevde Yanik & Sima Esmaeili

!! Forget abt the speedup in this exercise it doesn’t make sense for broadcasting.

**Problem 1**

Using MPI, a broadcast operation is implemented which broadcasts the whole content of a vector from the source process to all the other processes.

1. In the source process a vector with random integers is generated.
2. Timer is started and current time is retrieved as start\_time.
3. Broadcast operation takes place.
4. Timer stops and retrieves current time as end\_time.
5. The sum of the 3 least significant bits is calculated for each process and they are printed together with the ranks of the processes.
6. The source process calculates the total time for the broadcast operation by substracting end\_time and start\_time, then it prints the total\_time out.

The program is run with different vector sizes and different number of processes the runtimes are noted below. (time in ms)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Vector size | | | | | | | | |
| No. P |  | 100000 | 200000 | 500000 | 800000 | 1000000 | 5000000 | 8000000 | 10000000 |
| 2 | 0,26 | 0,48 | 0,52 | 0,64 | 1,31 | 3,61 | 5,78 | 7,66 |
| 4 | 0,64 | 1,09 | 0,67 | 0,65 | 1,45 | 3,97 | 7,83 | 8,68 |
| 6 | 1,09 | 1,57 | 0,62 | 0,73 | 1,41 | 4,19 | 5,33 | 8,4 |

**Problem 2**

The program is modified, it still uses MPI but this time the broadcast operation is a binomial tree broadcast. The content of the vector is sent to some of the processes and they pass them down to other processes.

1. In the source process a vector with random integers is generated.
2. Timer is started and current time is retrieved as start\_time.
3. Binomial tree broadcast operation takes place, using MPI\_Send() and MPI\_Recv()
   1. A variable called num\_steps keeps track of the steps taken in the binomial tree broadcast.
   2. A variable called power\_of\_two is used to keep track of the power of two used in each step.
   3. Inside of a while loop, each process checks if they send or receive vector\_data.
   4. The power\_of\_two is multiplied by 2 on each iteration of the while loop.
   5. The num\_steps is incremented on each iteration of the while loop.
   6. The while loop ends when power\_of\_two is greater than the number of processes.
4. Timer stops and retrieves current time as end\_time.
5. The sum of the 3 least significant bits is calculated for each process and they are printed together with the ranks of the processes.
6. The source process calculates the total time for the broadcast operation by substracting end\_time and start\_time, then it prints the total\_time out.

The program is run with different vector sizes and different number of processes the runtimes are noted below. (time in ms)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Vector size | | | | | | | | |
| No. P |  | 100000 | 200000 | 500000 | 800000 | 1000000 | 5000000 | 8000000 | 10000000 |
| 2 |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |

**Problem 3**

The program that uses parallelism to compute prefix-sums is implemented in C++ using MPI. The program computes the terms xi for i = 1, ..., n from a linear recurrence xi = ai \* xi-1 + bi where x0 = a0 for given sequences ai and bi. Breaking down the program step-by-step:

1. Three arrays a, b and x are initialized, a has values from 1 to 10 and b is initialized with 1 and x with 0.
2. The prefix sums of sequence a are calculated using MPI\_Scan. The results are stored in prefix\_sums\_a array. The MPI\_Scan performs a reduction operation on the input array.
3. The terms for x[i] are computed using the linear recurrence. The first term x[0] is set to a[0] and the rest of the terms are computed using xi-1, ai and bi elements.
4. The results of the prefix sums are printed out.