Homework 2- Parallel Computations for Large-Scale Problems

Oussama Kaddami

February 2022

Part I: Collective communication

a. We suppose that we have a root process that is willing to broadcast a message to P - 1 other processes. We adopt a binary tree with this process as the root. The root will initially send the message to two neighbors, that will in turn send it to two other nodes and so on. Those operations by different nodes are sent in parallel, so the number of communication steps is the depth of the binary tree which is $log(P) \times 2$ since each node needs to communicate the message to two of its neighbors.

b.Time analysis:

We consider the following algorithm for the purpose:

Rank = getRank()

If rank = 0:

send(message, $2^{rank} + 1$)

send(message, 2^{rank+1})

Else:

receive(message, $2^{\frac{rank}{2}}$)
/* Here we mean the integer division */
send(message, $2^{rank} + 1$)

send(message, 2^{rank+1})

$T_p = T_{comm} + T_{comp}$
where $T_{comm} = 2.log(P) \times T_{startup} + w.T_{data}$ and $T_{comp} = O(1)$
P is the number of processes and w is the size of the message.

c. How can the scatter operation be implemented using communication steps?
It could be implemented in the same way as the broadcast [i.e Using a binary tree based structure.]

Initially the root process split the array into two parts, sending each part into rank 1 and $\frac{P}{2}$, and also two indices (left and right) that will be used to compute the rank of the destination nodes. Initially, the root will send left = 0 and right = P, where P is the number of process which is the size of the array as well.

Then a receiving node **extract the first element from the array**, split the array and send to the node whose rank is *left + 1* the first half with left = left + 1 and right = $\frac{right-left}{2}$ and to the node whose rank is $\frac{right-left}{2}$ the second half with left) $\frac{right-left}{2}$ and right = right. The indices should be verified further but this is the general principle.

Part II: Matrix-vector product

a. Design an algorithm for this transposition.

Supposing that in the algorithm, each node computes one (or a set of) element(s) of the vector Y. The simplest way is to use AlltoAll in order to have the vector Y in the memory of all the processes (Nodes). One implementation could be the implementation using the butterfly recursive doubling algorithm.

So the algorithm can be resumed in one instruction : **alltoall()**.

b. Make a performance analysis of your transposition algorithm
The performance of this solution will depend on the implementation of **alltoall()**.

In the case of the butterfly implementation, we will have log(P) steps; so the performance is: $T_p = \sum_{i=0}^{log(P)-1} T_{startup} + 2^i \times T_{data}$.

Since at each step the size of the buffer is doubled.

Part III: Differential Equation

From the provided skeleton we implement a parallel resolution base on a linear distribution of the solution over the process with ghost values that are communicated by neighbors (using red/black communication to avoid deadlocks) at every iteration.
Please find the code attached.

We applied the algorithm for the resolution of the differential equation with:

- r(x) = 1 and f(x) = sin(x). The theoritical solution is A.cos(x) + B.sin(x) - $\frac{x.cos(x)}{2}$ $where\ A,\ B \in R^2$ The obtained result with our algorithm on [0, 1] are coherent with the solution:

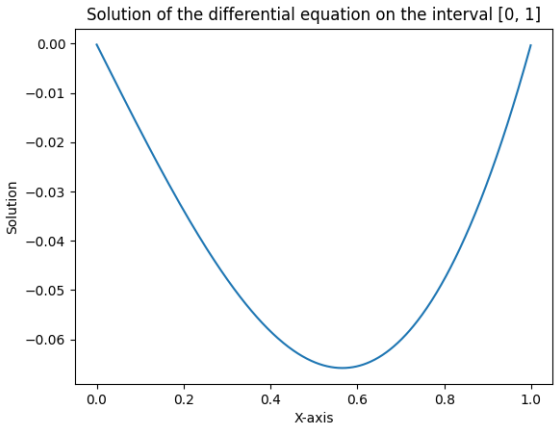

Figure 1: r(x) = 1 and f(x) = sin(x)

- r(x) = x + 1 and f(x) = sin(25πx):

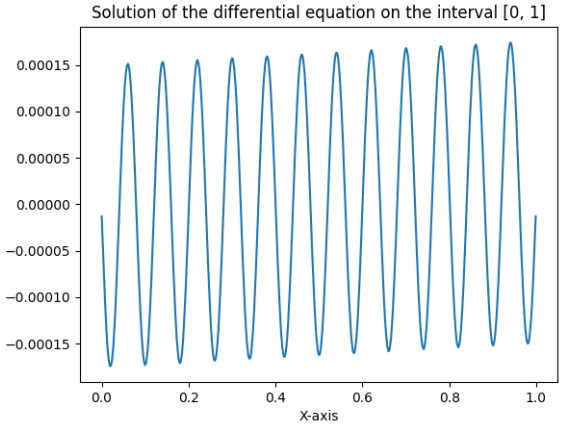

Figure 2: r(x) = 1 and f(x) = sin(x)