AMS 530 PRINCIPLES OF PARALLEL COMPUTING
PROJECT 1

Wenhan Gao
Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY 11794, USA
wenhan.gao@stonybrook.edu

1 Problem 1: Parallel Hello

1.1 Problem Description

A huge difference between normal sequential programming and parallel programming is that, in parallel programming, the order is non-deterministic even if you run the same program on the same machine.

In this problem, we wish to write a parallel program to print out "Hello from Processor X" in a fixed order, for example,

Hello from Processor 0
Hello from Processor 1
...
Hello from Processor 23

1.2 Algorithm Description and Pseudo-code

In order to print in order, we first identify each processor by its ranks. Then, we let Processor 0 print out "Hello" and sent a message to its successor, Processor 1. When Processor 1 has received the message from Processor 0, it prints out "Hello" and sent a message to its successor, Processor 2. Finally, we can print out "Hello"-s in order by continuing this process of receiving messages from predecessors and sending messages to successors.

In this way, each Processor except Processor 0 will not print until it receives a message from its predecessor and its predecessor would have already printed; therefore, processors will print out hellos in order. Algorithm 1.1 is the pseudo-code for Parallel Hello.

Algorithm 1.1: Parallel Hello World

Result: Printing out "Hello from Processor X" in order
$N :=$ total number of Processors(size);
Identify each Processor's rank by an integer in the range $[0, N-1]$;
if *rank* $= 0$ **then**
    Print "Hello from Processor 0" ;
    Send a message to Processor 1;
else
    Receive a message from Processor rank-1 ;
    Print("Hello from Processor %d", rank) ;
    if *rank* $<= N-2$ **then**
        Send a message to Processor rank+1;
end

1.3 Results

Figure 1.1 shows the sample output of running Parallel Hello with 24 processors.

Figure 1.1: Sample Output

1.4 Analysis of Program Performance

For further inspection, we employ MPI_Wtime() as a timer to keep track of the running time. The global elapsed run-time (the maximum of all individual run-time by MPI_Reduce) for Parallel Hello in fixed order is 0.013037 seconds. Meanwhile, the global elapsed run-time is 0.003160 seconds if we allow processors to print in any order. Therefore, the program is slowed down by approximately 0.01 seconds for the processors to communicate and cooperate to print in the desired order. Although a 0.01-second communication penalty seems expensive compared to a total run-time of 0.013037 seconds, the task of printing out hellos itself is a very simple task, and the communication penalty is relatively "exaggerated" because of it. Overall, the program is efficient and robust.

1.5 Source Code for Submission

Source code will be submitted along with this report. Source code can also be found in this git repo (Clickable Link).

This git repo will be deleted after the end of the semester to prevent future students from reusing the code.

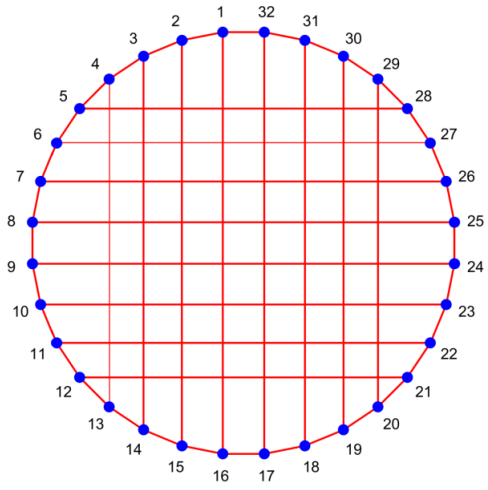
2 Problem 3: Network Broadcast

2.1 Problem Description

In the problem, we wish to broadcast 16 floating-point numbers from any designated vertex to the entire system represented by a (connected) 3-regular graph with 32 vertices. Vertices can be labeled in any order as long as each vertex gets one index. Since all connected 3-regular graphs with 32 vertices are isomorphic to each other, without loss

of generality, we consider the following (32,3) graph, where the blue dots represent vertices, red line segments represent undirected edges, and vertex "1" is the starting vertex.

Figure 2.1: ((32,3) Regular Graph



2.2 Algorithm Description and Pseudo-code

We use **greedy algorithm** to broadcast the information. In every iteration, as long as a vertex has the information and is not saturated, we send the information to one of its "uninformed" neighbors, if there are multiple "uninformed" neighbors, we choose the one that has more "uninformed" neighbors itself (we call such neighbors the "most uninformed" neighbors for the sending vertex).

To implement this algorithm in computer programming, we may use a hashmap (dictionary if you prefer python) structure to represent such graph, where the keys are vertices, 1 to 32 and the value with respect to each key is a set contains its neighbors. For example, 1 correponds to [2, 32, 16] and 2 corresponds to [1, 3, 15]. Algorithm 2.1 describes how this greedy algorithm works.

Algorithm 2.1: Network Broadcast

Require: Dictionary representation of the (32,3) graph

Initiate informed vertex set $\mathcal{V} = \{1\}$;
Initiate $t = 0$;
while $len(\mathcal{V}) < 32$ **do**
    $t+ = 1$;
    for $v \in \mathcal{V}$ **do**
        Send information to a "most uninformed" neighbor, call it u ;
        $\mathcal{V}$.add(u) ;
    end
end

2.3 Results

The process takes 9 steps. Figure 2.2 shows the links that are involved during each step as well as the number of edges involved in this step.

2.4 Analysis of Program Performance

It takes 9 iterations to complete the broadcast. I don't think that it is optimal. However, overall this problem is difficult to optimize as each decision we make is a local decision meaning that a smart move in the current iteration

Figure 2.2: Sample Output

```
step 1 :
['1 - 16']
The total number of informed vertices is: 2 . The number of edges involved in this step is 1
step 2 :
['16 - 15', '1 - 32']
The total number of informed vertices is: 4 . The number of edges involved in this step is 2
step 3 :
['16 - 17', '1 - 2', '32 - 31', '15 - 14']
The total number of informed vertices is: 8 . The number of edges involved in this step is 4
step 4 :
['2 - 3', '14 - 13', '17 - 18', '31 - 30']
The total number of informed vertices is: 12 . The number of edges involved in this step is 4
step 5 :
['3 - 4', '13 - 12', '18 - 19', '30 - 29']
The total number of informed vertices is: 16 . The number of edges involved in this step is 4
step 6 :
['4 - 5', '12 - 11', '19 - 20', '29 - 28']
The total number of informed vertices is: 20 . The number of edges involved in this step is 4
step 7 :
['5 - 6', '11 - 10', '12 - 21', '28 - 27']
The total number of informed vertices is: 24 . The number of edges involved in this step is 4
step 8 :
['6 - 7', '10 - 9', '11 - 22', '27 - 26']
The total number of informed vertices is: 28 . The number of edges involved in this step is 4
step 9 :
['7 - 8', '9 - 24', '10 - 23', '26 - 25']
The total number of informed vertices is: 32 . The number of edges involved in this step is 4
```

might hurt the algorithm in the long run. It can an interesting work to explored a global optimization algorithms for this broadcasting problem given any graph.

2.5 Source Code for Submission

Not applicable.

References