Faculté des Sciences & Techniques — Université de Limoges

Master 1^{rst} year

Parallelism & Applications

Examination – December 2024

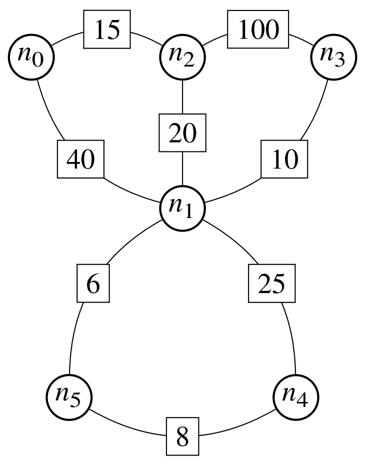
Duration: 1h30 — documents authorized

1 – Parallelization of Dijkstra's « *shortest path tree* » algorithm:
14pts

from Wikipedia

Initially, we consider that the distances from each node to the **starting node** are infinite, except for the i**starting node** for which the distance is zero.
Initially, all selected nodes are empty.
During each iteration, we choose a **selected node** of minimum distance among the nodes not yet selected, and we add it to the selected nodes.
Then, we update the distance of each neighboring node of the **selected node**,: the new distance of the **neighboring node** is the minimum between the existing distance and the distance calculated by adding the sum of,:

▷ the distance between the **starting node** and the **selected node** (contained in the distance table)
▷ the distance from **selected node** to this **neighbor node** (contained in the adjacency matrix).



The graph expressed as an adjacency matrix:

```
#define MAX_INT 32768

#define     N   6
#define TRUE   1
#define FALSE  0

int graphe[N][N] = {
     { 0, 40,  15,   0,  0,  0 },
     {40,  0,  20,  10, 25,  6 },
     {15, 20,   0, 100,  0,  0 },
     { 0, 10, 100,   0,  0,  0 },
     { 0, 25,   0,   0,  0, 8},
     { 0,  6,   0,   0,  8,  0},
};
```

The prototypes of the functions to write/use:

```
// a function that returns the index of the node with the minimum distance
// from the set of nodes not already included in the shortest path tree
int distanceMin(int distance[], int ensembleSPT[]) {
      int min = MAX_INT, index_min;
...

// the function that implements Dijkstra's shortest path algorithm
// from a starting node for a graph represented by an adjacency table
void dijkstra(int graphe[N][N], int depart) {
      int distance[N];  // the output array which will contain the shortest distance
                        // from the starting node

    int ensembleSPT[N]; // ensembleSPT[i] is true if the node is included in the
                        // shortest path tree
...
```

Questions :

a. Run the algorithm by hand on the graph and give the array `distance` result. *(2pts)*
In how many steps does the algorithm arrive at the result?

b. Write the code for the `min distance` function using the code provided. *(2pts)*
Why do we need `max int`?

c. write the code for the `dijkstra` function in sequential version using the code provided. *(3pts)*

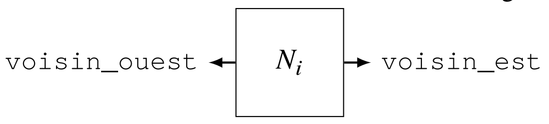
d. **we always consider the construction of the « shortest path tree » for a single starting node**. *(6pts)*
What are the potential sources of parallelization that can be exploited with openmp in this algorithm?
What type of parallelism will you exploit?
For the different OpenMP threads, what are the algorithm data:
- ◇ privates ;
- ◇ shareds ;

Are there any risks/problems for accessing this data?
Are the « *tasks* » interesting or not?
You will explain why.
Describe the parallelization of the `dijkstra` function that you are going to apply.
Provide the code using OpenMP corresponding to your parallelization proposal.

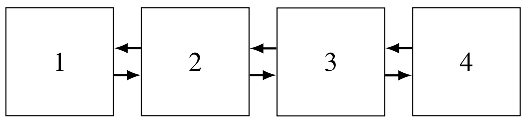
e. Is your solution **optimal cost**? *(1pt)*

2 – 6pts For an algorithm, we want to set up a **ring** structure on a « *MPI cluster* ».

Each node in the cluster will have at most 2 neighbors:



Example on a cluster of 4 nodes defining a ring of 4:



We will consider that the number of nodes in the ring is chosen by the user when launching the program.

Questions :

a. Give the code carrying out the following processing: *(1pt)*
- ▷ integrate the node into the ring,;
- ▷ define in relation to the rank of a node, the rank in the ring of each of its neighbors if it exists.

b. Give the code to successively send by each node, an integer value contained in the variable `coefficient` present on each node to each of its neighbors, if it exists, first to `east neighbor`, then to `western neighbor`. *(2pts)*

c. When receiving the `coefficient` value by a node from each of its neighbors: *(2pts)*
- ◇ is the sending and receiving going to be i**synchronized**?
- ◇ will the node receive the values of each of its neighbors in a **predefined order**?
  Why?
- ◇ can the node **know from which node** it receives each value?
  Can this **introduce delays** in the processing of each reception?
  How can we **correct** these problems?
  You will give the instructions to use for the reception of the different neighbors.

d. At the end of the algorithm, one of the nodes of the ring is responsible for recovering from each of the nodes of the ring, a final value, stored in the floating variable `result` present on each node. *(1pt)*

What MPI operation will you use to carry out this reception from each node of the ring, knowing that it is the first node of the ring which will be responsible for recovering all these values.