ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Ecole Polytechnique Fédérale de Lausanne

***Program Parellelization of PC Clusters project*:**

Ant Colony traveling salesman problem optimization

with MPI

Marc Schär

November 23, 2015

Contents

1 Introduction

In the context of the *Program Parallelization on PC Clusters* course, we have to analyze and implement a parallel application to observe the improvements brought by the parallelization of a serial application. The goal of this project is to anticipate the performance of the parallelized application and to confirm or invalidate the claimed theory.

For this project, I chose to implement an *ant colony optimization algorithm* which gives a solution to the *Traveling Salesman Problem*.

In this report, I will first present a description of the algorithm with a generic pseudo-code, my choices to implement it, the possible problems and solutions and an example of representation of the map. Finally, I will make a theoretical analysis of the serial and parallel implementations of the algorithm.

1.1 Algorithm

1.1.1 Description

We want to solve the *Traveling salesman problem* with ants. The idea is to have several ants that are spread randomly on the map. Each ant will go through the map and find a path between all cities without visiting twice the same city. At each step, the ant will select the next city with some probability depending on distance and pheromon value. When all ants are done, the pheromons on all the map are reduced by a factor such that edges that are less used have a smaller value. Finally, it updates the visited edges of the best path[1] with pheromons to give more importance to this path for the next iterations. Then, it starts from the beginning until a termination condition is met.

The final goal is to find the shortest path that starts in a city and that arrives in this city going only once through each other city.

The standard termination condition is defined as a maximum number of iteration or the same shortest solution after a certain number of iterations.

1.1.2 Pseudo-code

The general algorithm of the *ant colony optimization algorithm* is the following[2] :

```
while (termination condition) {
  for each ant :
    has to visit all cities once
    chooses next city based on distance and pheromon value
  endfor

  reduce the global pheromon values
  update pheromons based on the best path
}
```

[1] It is a choice to keep only the best path between all ants.

[2] This algorithm is based on the *Ant Colony optimization algorithms* found on wikipedia (in french) : https://fr.wikipedia.org/wiki/Algorithme_de_colonies_de_fourmis

1.1.3 Possible problems and solutions

There are several problems that can happen with this algorithm :

- The ant can be stuck in a city if the graph is not fully connected.
  - A possible solution is to kill ants that are in this situation;
  - Another possibility is to transform the graph in a fully connected one with unreasonnable edges for non existing paths[3].
- A suboptimal solution can be found
  - To solve this, it is sufficient to run the algorithm several times[4].
- An isolated city with only one connection to the rest of the graph will make the resolution of the *TSP* impossible.
  - The solution is to assume that each city has at least two connected edges.
- To parallelize this problem, the exchange of pheromon values can be a bottleneck (lot of communications).
  - A possible solution is to run several iterations locally before sharing the result;
  - Another possibility is to send only the updated values and not all the matrix.

1.1.4 Edge selection

An ant will select a path according to some probabilities that includes edge's length and pheromon values :

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}(t)^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in J_i^k} \tau_{il}(t)^\alpha \cdot \eta_{il}^\beta} & if \quad j \in J_i^k \\ 0 & if \quad j \notin J_i^k \end{cases}$$

where J_i^k is the list of possible moves for an ant k in the city i, η_{ij} is the *visibility*, which is the inverse of the distance between two cities i and j and τ_{ij} the intensity of a path (the pheromon values). α and β are parameters to control the algorithm.

Finally, I also define $\rho\tau_{ij}(t)$ which represent the pheromon evaporation that is computed at each iteration.

1.1.5 Pheromon update

At the end of each *external* loop, the pheromon values are updated according to the best path of the current iteration. To spread the pheromon such that shortest paths between several iterations will have more pheromons that longer one, I define this formula :

$$\Delta\tau_{ij}^k(t) = \begin{cases} \frac{Q}{L^k(t)} & if \quad (i,j) \in T^k(t) \\ 0 & if \quad (i,j) \notin T^k(t) \end{cases}$$

where $T^k(t)$ is the best path of ant k at iteration t, $L^k(t)$ is the length of the path and Q is a parameter to be defined.

[3] I chose this solution.
[4] As it is not the purpose of this project to have an optimal solution, I will simply run the algorithm once.

1.1.6 Representation

To represent this algorithm in a computer, we have to model a map with a graph. Each vertex will represent a city and edges will represent connections between cities. Each edge has a weight representing the distance between all cities. As we want to have a fully connected graph, edges that connect unlinked cities will have a maximal weight (which will be defined in the implementation of the algorithm). Figures 1 and 2 show how to represent the map.

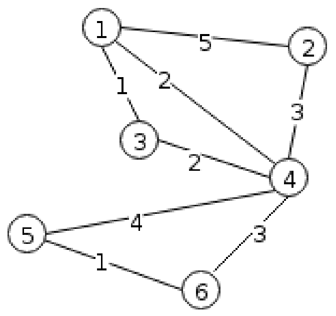

Figure 1: A map composed by cities and paths

$$\begin{pmatrix} 0 & 5 & 1 & 2 & \infty & \infty \\ 5 & 0 & \infty & 3 & \infty & \infty \\ 1 & \infty & 0 & 2 & \infty & \infty \\ 2 & 3 & 2 & 0 & 4 & 3 \\ \infty & \infty & \infty & 4 & 0 & 1 \\ \infty & \infty & \infty & 3 & 1 & 0 \end{pmatrix}$$

Figure 2: The matrix representing figure 1.

Each ant will simply be represented by finding a path on the map from different starting points. To find a path, the solution is to select a path from the current city (select an element in a column) being careful to not select an already visited city and selecting a city following the formula in section 1.1.4.

We can note that the pheromon values need to be stored in a matrix, too. This matrix will have the same construction as the one in figure 2.

2 Theoretical Analysis

2.1 Serial application

2.1.1 Computational complexity

The serial *ant colony optimization algorithm* is composed by a first *initialization* part that initializes all variables, including the matrix that represents the graph and the one with pheromon values (complexity of $O(2 \cdot c^2)$ for c cities). Then, the algorithm will loop until a termination condition is reached[5]. This "big" loop will make at most L iterations. In this loop, there is another loop for simulating ants (we have A ants). In it, there is a loop to go through each city (c cities) where we have to compute the probabilities to go to a specific city ($O(c)$). Finally, we have to perform the pheromon evaporation ($O(c^2)$) and we have to update the pheromon's matrix with the new pheromon's values from ants ($O(c)$ because we have to update only the best path.

The skeleton of the serial application is like this pseudo-code :

```
init() // 2*c*c operations
while (termination condition) { // at most L loops
  for each ants { // A iterations
    for each city { // c iterations
      find next city to visit // O(1)
```

[5]Either the maximum number of iterations is reached or the solution is stabilized.

```
    }
  }
  pheromon evaporation // O(c*c)
  update bestPath's pheromons // O(c)
}
```

It gives us a computational complexity of $O(c^2 + L \cdot (cA + c^2 + c))$.

We can define the number of ants to the number of city (even if it is useless to have so much ants) and we can consider that the number of *external* loops is a constant, we have a complexity of $O(L \cdot c^2)$.

With this complexity, we can observe that number of ants and the size of the map are the two factors that makes the computations complex or not.

2.2 Parallel Application

For the parallel implementation of the *ant colony optimization algorithm*, we have to define what is parallelized and what are the content of communications.

2.2.1 Basic Parallel Application

For the analysis below, let us consider we have c cities, A ants, n nodes (including the master) and we assume that the external loop has a fixed number of iterations such that we do not have to take it into account. Indeed, we can compute the complexity for one *external* iteration only. That allows us to remove the factor L.

2.2.1.1 Computation to communication time For the initialization part, the graph will be spread by the master to each node ($O(c^2)$) using broadcasting. And, at the beginning of each iteration, the master needs to send the pheromon's matrix.

Then, each node will compute the paths of multiple ants ($O(c \cdot \frac{A}{n})$). We assume that each node has the same number of ants. The computation on each node is exactly the same as in the serial application except that each node has only a subset of ants. We can add that if we want to do several iteration before merging the results, we will simply multiple this cost by a constant, which will not change the bounded complexity.

Then, when they all have finished to compute the new map with new pheromon's values, all nodes send their local best path to the master. The cost of sending all vectors is $O(cn)$.

Finally, the master will merge all received vectors ($O(c \cdot n)$ additions) and will perform the *pheromon evaporation* ($O(c^2)$) and decide if another iteration is needed or not.

Thus, for computation time we have a cost of $O(c \cdot \frac{A}{n} + c \cdot n)$ and for communication time we have a cost of $O(c^2 + cn + c^2)$ because the master must resend the pheromon's matrix after each merge operations.

The *Computation/Communication* ratio is $\frac{O(c \cdot \frac{A}{n} + cn + c^2)}{O(c^2 + cn)}$ which tells us that the size of the problem will not make the communications less important. In fact, the only way to have more computations than communications is to have $\frac{A}{n} > c$.

Nevertheless, we can implement the master such that it does not send all elements in the matrix but only the half of them (as it is symmetric). But, in a theoretical analysis, we conclude that the communications are really costly for this algorithm.

2.2.1.2 Amdhal's law The Amdhal's law gives us a theoretical speedup when we improve ressource for a problem of a fixed size. For the *ant colony optimization problem*, we can define this theoretical upper bound limitation as follows, where n is the number of nodes and f is a fraction of the code that cannot be parallelized :

$$S_n \leq \frac{n}{1 + (n-1) \cdot f}$$

2.2.1.2.1 Without data transfer If we do not account for data transfers, the fraction of the code that cannot be parallelized is when the master creates the matrix at the beginning of the program and when it merges all results from nodes.

Considering the complexities defined above, the serial part will have a complexity of $O(c^2)$ to initialize the matrix and then a complexity of $O(c \cdot n)$ to update the matrix and $O(c^2)$ to perform the *pheromon evaporation*. It gives us $c \cdot (n + 2c)$ operations.

For the parallel part, we have $c \cdot \frac{A}{n} \cdot c$ operations[6].

Thus,

$$f = \frac{c \cdot (n + 2c)}{c \cdot (n + 2c) + c^2 \cdot \frac{A}{n}} = \frac{n + 2c}{n + 2c + \frac{cA}{n}} = \frac{n + 2c}{n + c(2 + \frac{A}{n})}$$

Here, we observe that A and n are important. To have a small f, we need to have more ants than nodes. It is logical, because to have an efficient parellelization of this program, nodes must have something to compute (the computation time on nodes must be big compared to communications.

2.2.1.2.2 More accurate estimation Now, if we consider the transfer of data between nodes, we must modify the value of f. Indeed, we have to consider the communication time in our computations. To send a matrix from the master to each node, we have c^2 values sent to n nodes using broadcast and we have $c \cdot n$ values returned from n nodes. Thus, we have :

$$f = \frac{c \cdot (n + 2c) + c \cdot (c + n)}{c \cdot (n + 2c) + c^2 \cdot \frac{A}{n} + cn \cdot (c + n)} = \frac{2n + 3c}{n^2 + n + 2c + \frac{cA}{n}}$$

Again here, we observe that the number of ants compared to the number of nodes is important to have a good speedup. We can already affirm that adding ressources for solving this problem will not be efficient if the problem is not big enough (not enough cities and ants).

2.2.2 Improved Parallel Application

As we can observe with the parallel implementation where we send the whole matrix at each iteration, we can reduce the communications with computing several iterations locally before sending the matrix to the master (we reduce the communications having more accurate local solutions before merging). We also can send to the master only the best path over several iterations.

As we want to improve the performances of the parallel application we can consider both improvements together. Thus, we need to define the variable l which represents the number of iterations to perform locally before merging the best path to the master. The theoretical speedup upper bound is presented below.

[6] We go through each city and in each city we have to compute the probabilities for each other cities.

2.2.2.1 Computation to communication time The initialization part stays the same as in the basic parallel application : $O(c^2)$ to initialize the matrix and $O(c^2)$ to send it to the nodes.

For the computation complexity on nodes, we have now a complexity of $O(l \cdot c \cdot \frac{A}{n})$. The update of pheromon values is done locally and is kept between local and external iterations[7].

Then, for the merge to the master, we only send the best path (which needs to be kept in a local variable and updated after each ant execution). Thus the complexity is $O(c \cdot n)$.

Finally, the master has to merge all nodes results : $O(c \cdot n)$.

The *Computation/Communication* ratio is

$$\frac{c^2 + l \cdot c \cdot \frac{A}{n} + c \cdot n}{c^2 + c \cdot n} = \frac{c + l \cdot \frac{A}{n} + n}{c + n}$$

.

Here we can conclude that the larger is the problem (in term of ants, city and thus local iterations), the smaller is the impact of communication. Nevertheless, more we have nodes, more are the communications important (this seems totally logic).

2.2.2.2 Amdahl's law

2.2.2.2.1 Without data transfer Without considering the data transfer, we have to compute in a serial manner the initialization of the matrices (c^2) and the merge of all best paths ($n \cdot c$). For the parallel part, we have the computation of the best path for each ant with several local iterations ($l \cdot (c \cdot \frac{A}{n})$).

Thus, we have the following non-parallel fragment :

$$f = \frac{c^2 + n \cdot c}{c^2 + n \cdot c + \frac{lcA}{n}} = \frac{c + n}{c + n + \frac{lA}{n}}$$

We observe again that the number of ants and the number of local iterations is important to maximize the speedup. Thus, more we have nodes, more we need to have a bigger problem to solve to keep a reasonnable speedup.

2.2.2.2.2 More accurate estimation For having a more accurate estimation, we consider the communication time. As we defined above, we consider the send of the whole matrix (c^2) to each node and the send of all best paths ($c \cdot n$). Thus, we have :

$$f = \frac{c^2 + n \cdot c + c^2 + c \cdot n}{c^2 + n \cdot c + c^2 + c \cdot n + \frac{lcA}{n}} = \frac{2c + 2n}{2c + 2n + \frac{lA}{n}} = \frac{2c + 2n}{2c + 2n + \frac{lA}{n}}$$

Again, we can observe that the number of ants and the number of local iterations is important to define the speedup, but, here, the number of cities makes the communication more dominant. It is logic, because the vectors will be bigger so the communications increase with the number of cities and the number of nodes.

[7] Thus, we have some history in each node to help it to find a solution faster.

2.2.3 Conclusion on Amdahl's law analysis

The vision of Amdhal's law is pessimist because the communications will take a lot of time if we increase the resources without increasing the problem. On the contrary, Gustafson's law is more optimistic because it adapts the task to the number of computation resources. Nevertheless, Amdahl's law gives us a good idea of how many cores we need for a specific problem's size and in the *ant colony optimization problem* it is important to avoid having too much nodes compared to the size of problem, according to the theoretical analysis.

2.3 Theoretical analysis with MPI

Now, I will establish an analysis of the speedup using the framework *Message Passing Interface (MPI)* which allows several computational nodes to work together in a distributed memory fashion.

As discussed in part 2.2, we have the following sequence of message passing for one master and 3 nodes.

For spreading the matrix into all nodes, we can use `MPI_BCAST` which sends to all nodes the same data. It allow us to improve a bit the communications.

2.3.1 Sequence of message passing

We have a kind of *DPS flow graph* like in figure 3 where we can observe the general scheme of the algorithm. We also have two *MPI* operations which are when the master send the matrix to each node and when they send back their own best path.

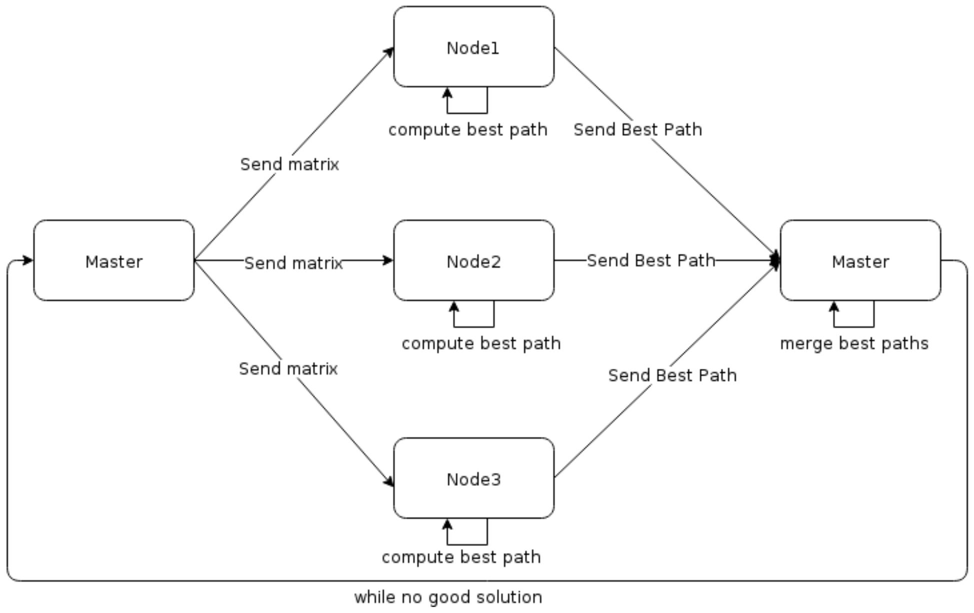

Figure 3: *General scheme of the application*

The sequence of message passing is presented in figure 4 for one iteration, without considering the local computations. The timeline has no meaning in term of size. This sequence diagram is here only to show in which order are sent messages between nodes and master.

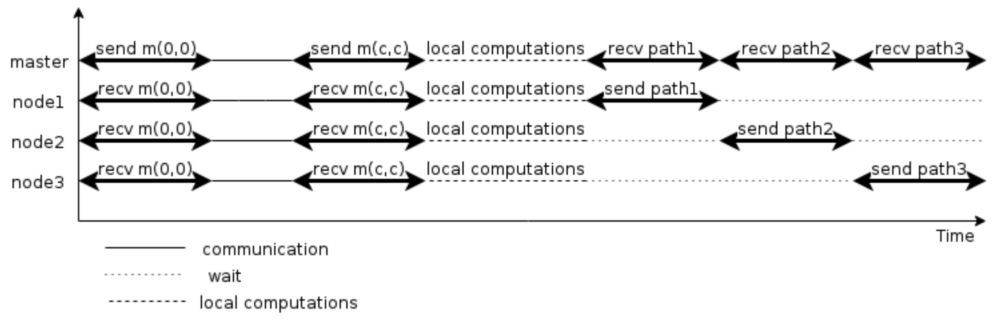

Figure 4: *Sequence of message passing*

2.3.2 Approximative Timing Diagram

Figure 5 presents an approximative timing diagram of computations and communications between one master and 2 slave nodes. We suppose, for keeping small this diagram, that there are only 3 cities and 3 ants (one per node). The computation phases are made bigger than the communication ones, but in reality, with this example, the communications should take more time[8]. I also consider that all similar operations take the same time to be executed.

In fact, this is true. Indeed, all comunications are sending only one value so, without considering problems on the network, it should always take the same time to be executed. For computations, if all nodes have the same number of ants to manage, they will have a similar computation time (if the machines have an identical architecture and hardware) because in all cases the map has the same size, and each city has the same number of neighbours to consider.

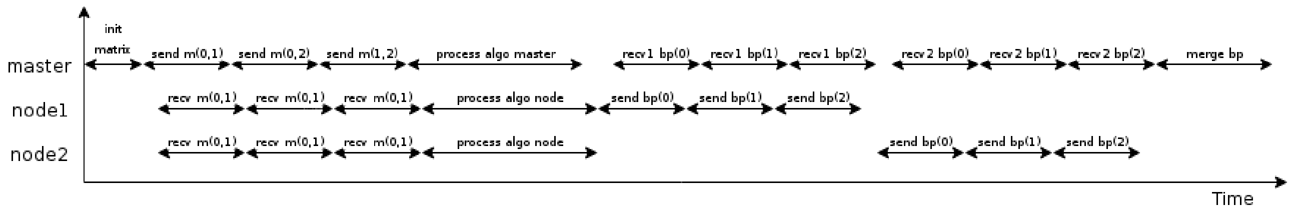

Figure 5: *Timing diagram for initialization + one iteration with 3 nodes. bp means best path.*

To have a better idea of the different timings, we can suppose we have a network with a throughput of 125 MB/s[9] for data. Knowing that each message sent by a node or the master will contain 3 integers (a value and its indices in a matrix) and that an integer is 4 bytes, we can assume that a message takes $9.2 \cdot 10^{-8}$ seconds to be sent[10].

For the computation time, I ran one iteration of the algorithm with one ant on an *Intel core i5* machine with 4 CPU at 2.60 GHz for a map of 1000 cities. It took 0.021076 seconds. For initializing variable (especially map and pheromons matrix), it took 0.139912 seconds, even if this time can vary because the map is read from a file. For the merging part, it took 0.000009 seconds.

With these values, we can do some computations to estimate the time for computations with multiple nodes and multiple ants for a fixed map size of 1000 cities[11]. It is presented in figure 6[12].

[8] I consider a huge problem where the communications are less important than computations.
[9] 125MB = 131'072'000 bytes.
[10] We neglect the small latency.
[11] Of course, I will try with different map sizes when the parallel algorithm will be implemented.
[12] The communication parts are not considered when there is only one node.

nodes / ants	**1**	**2**	**4**	**8**	**16**
1	63.368182	126.596182	253.052182	505.964182	1011.788182
2	65.297422	65.297422	128.525422	254.981422	507.893422
4	65.849422	65.849422	65.849422	129.077422	255.533422
8	66.953422	66.953422	66.953422	66.953422	130.181422
16	69.161422	69.161422	69.161422	69.161422	69.161422

Figure 6: *Theoretical computations for 1000 cities and 3000 iterations (30 external iterations and 100 internal ones) based on one machine measurements. The values are computed as follows (in seconds) :* $t_{init} + 30 * (\frac{1000^2}{2} - 1000) * t_{send} + 100 * (\lceil \frac{n_{ants}}{n_{nodes}} \rceil \cdot t_{computation}) + n_{nodes} \cdot 1000 \cdot t_{send} + t_{merge})$.

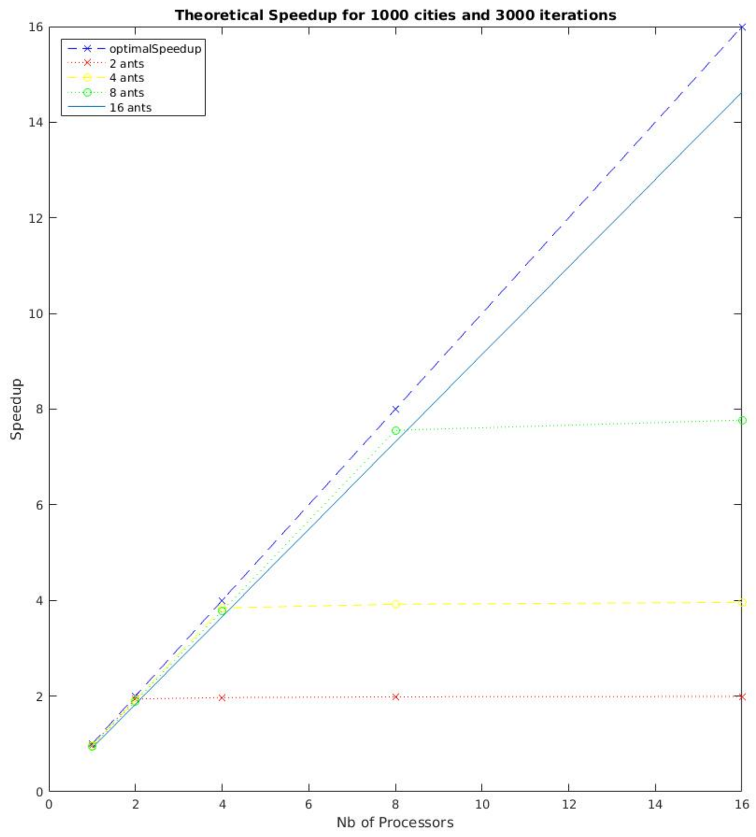

Figure 7: *Theoretical speedup for 1000 cities and 3000 iterations (30 * 100) based on figure 6.*

We can observe in figures 6 and 7 that with a few ants, adding nodes is not a good solution, especially if we add nodes that have nothing to do. On the contrary, with more ants, we clearly see that adding nodes improves performances. We can already expect that with the variance of the network speed, the practical speedup will not be as good as here.

2.3.3 Critical Path and Theoretical Speedup

As we can observe on figure 5, the critical path is the following one :

$$init\,matrix \rightarrow send\,m(0,1) \rightarrow send\,m(0,2) \rightarrow send\,m(1,2) \rightarrow process\;algo\;node \rightarrow recv1\;bp(0) \rightarrow recv1\;bp(1) \rightarrow$$
$$recv1\;bp(2) \rightarrow recv2\;bp(0) \rightarrow recv2\;bp(1) \rightarrow recv2\;bp(2) \rightarrow merge\;bp$$

With this critical path, we can define a theoretical speedup. The speedup is defined as follows :

$$S_n = \frac{t_s}{t_p}$$

where t_s is the time of the serial application and t_p is the time of the parallel application.

In this theoretical speedup analysis, we will keep t_s as such[13], but we will define morce precisely t_p. With the critical path, we can define t_p as follows :

$$t_p = t_{init} + t_l + (\frac{c^2}{2} - c) \cdot t_{send} + \frac{t_s \cdot l}{L} + t_l + n \cdot c \cdot t_{send}$$

where t_{init} is the time to initialize the matrix, t_l is the minimal time to start a transmission, c is the number of cities[14], t_{send} is the time to send one element of the matrix, L is the number of *external* iterations in the serial program, l is the number of iterations to process locally in a node before merging to the master and n is the number of nodes (without considering the master).

We can reduce t_p :

$$t_p = t_{init} + 2t_l + t_{send} \cdot \frac{c^2 + 2(n-1)c}{2} + \frac{t_s \cdot l}{L}$$

Finally, the speedup is as follows :

$$S_n = \frac{t_{init} + t_s}{t_{init} + 2t_l + t_{send} \cdot \frac{c^2+2(n-1)c}{2} + \frac{t_s \cdot l}{L}}$$

To have good performances, we need to have t_p as small as possible. Ideally, t_{send} should be really small and l should be reasonnably small compared to L.

2.3.3.1 Remarks

We must note that, to compare the serial execution and the parallel execution, we need to have the same number of iterations in the algorithm. Indeed, as the termination conditions of the algorithm are a maximum number of iterations or the same best path after a certain number of iteration, the algorithm can converge at different speed (because of the randomness of path choice). Thus, to compare improvement of the algorithm, I will only implement the maximum number of loops for termination condition.

[13]We add to it the initialization phase because it is also considered in the parallel computations.
[14]We send only the upper part of the matrix because it is symmetric and the diagonal is null.