An Agglomeration Strategy for the Parallel Processes Mapping onto a Distributed Computing Architecture

Conference Paper · March 2015

DOI: 10.1007/978-3-319-32243-8_14

2 authors:

Juan Carlos Catana Salazar
Universidad Nacional Autónoma de México
3 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Jorge L. Ortega-Arjona
Universidad Nacional Autónoma de México
90 PUBLICATIONS **137** CITATIONS
SEE PROFILE

An Agglomeration Strategy for the Parallel Processes Mapping onto a Distributed Computing Architecture

Juan C. Catana-Salazar[1] and Jorge L. Ortega-Arjona[2]

[1] Posgrado en Ciencia e Ingeniería de la Computación
Universidad Nacional Autónoma de México
j.catanas@uxmcc2.iimas.unam.mx
[2] Departamento de Matemáticas, Facultad de Ciencias
Universidad Nacional Autónoma de México
jloa@ciencias.unam.mx

Abstract. Parallel processes, by nature, tend to interchange a high amount of data between them to maintain a highly cohesive system. Nevertheless, when a parallel system is executed on a distributed computing architecture, communications over a network and the time spent by them become very important.
This paper introduces an strategy to agglomerate and allocate parallel processes onto a distributed computing architecture. The main goal of the strategy is to decrease the amount of remote communications and increase the amount of local communications, as a result the performance of the parallel system is favored, just by allocating processes "carefully" over the distributed nodes.

Keywords: Parallel Process, Mapping Problem, Networks Flows

1 Introduction

A parallel system is a set of processes that communicate each other and collaborate to accomplish a common goal. A parallel system not only have multiple instruction flows executing at the same time, but also multiple data flows between processes [7].

A parallel system, by nature, involve problems like synchronization and race conditions, coupled with the mapping (allocation or assigment) problem and load balancing problem. All factors mentioned before affect directly the performance of the system [1, 7, 9].

A parallel system can be classified depending on its own communications or synchronization needs. The granularity g_{p_i} is a qualitative measure rate between processing time t_{proc} and communication time t_{com} of a process p_i[5], see equation 1.

$$g_{p_i} = \frac{t_{proc}}{t_{com}} \tag{1}$$

Three types of granularity are derived from the relation between processing and communication, which are shown next:

1. **Fine granularity:** Says a process is fine grained if $t_{com} > t_{proc}$.
2. **Medium granularity:** A process is medium grained when $t_{com} \simeq t_{proc}$.
3. **Coarse granularity:** Says a process is coarse grained when $t_{com} < t_{proc}$.

Any parallel machine or multiprocessor system must implement communications via one or more memory blocks. There is a broad variety of memory architectures which mainly differs on the access method[9]. Two of them are presented in the following:

1. **Shared memory:** Memory is directly accessed, commonly through a bus, by every processor in the system. Every processor has a common "snapshot" of the shared memory[7].
2. **Distributed memory:** There are many memory blocks hosting many processes. Processes hosted in a memory can only "see" the local memory. Process needs a network channel to interchange data with other processes hosted on different memories[7, 9].

The main difference between memory architectures is the communication time, in the shared memory architecture communication time is fast and uniform in access time, due to the "closeness" between memory and processors. On the other hand, in a distributed memory architecture communication time is variable and depends on external characteristics related with the network channel, network protocols etc[7].

For all communication c_{ij} between two processes p_i and p_j of a parallel system P, such that both are hosted at the same memory block, then, is added a constant communication time t_{cons} to the execution time $ET(P)$ of the parallel system P. On the other hand, for all communication c_{kl} between two processes p_k and p_l, such that both are hosted on different memory blocks, then, is added a variable communication time t_{var} to the execution time $ET(P)$, where:

$$t_{cons} << t_{var} \tag{2}$$

The execution time $ET(P)$ of a parallel system P can be seen (in a simplified way) as, the processing time $PT(P)$ plus the time spent by the whole communications $CT(P)$ of the system.

Let $c_l(P)$ be the amount of local communications of the parallel system P. Let $c_r(P)$ be the amount of remote communications of the same parallel system. Then $CT(P)$ is equal to $c_l(P)$ constant time communications plus $c_r(P)$ variable time communications. See equation 3.

$$ET(P) = PT(P) + c_l(P) * t_{cons} + c_r(P) * t_{var} \tag{3}$$

This paper presents an strategy which main goal is to maximize local communications and minimize remote communications among processes of a parallel system, considering its execution onto a distributed memory architecture with k processing nodes.

In section 2 are introduced a set of definitions to help to establish a few tools needed by the strategy proposed in this work. Section 3 describes the strategy a in detailed way and shows additional considerations. Section 4 shows a case study where is applied the proposed strategy in a real life case. Finally section 5 shows the conclusions of this work.

2 Backgrounds

In this section are presented two main themes, a parallel software methodology and a miscellaneous definitions of network flows.

2.1 Parallel Software Methodology

There are many parallel software methodologies proposed in the literature to design software, the common goal of every methodology is to have an easy way to translate a sequential problem into a parallel system. Also is desired to consider factors such as performance and efficiency.

In the following, are presented four common steps that can be found in every parallel software methodology:

1. **Partitioning or Decomposition.** The partitioning stage involves to divide a general problem into a set of independent modules that can be executed in parallel. That does not implies to have a number of processes same as the number of processors, however, this stage is concerned about to express the parallelism in every opportunity, no matter how many resources the system has. The partitioning of the problem must be enough to decrease the communication latency between processes[4, 5].

2. **Comunication.** Parallel modules, generated by the previous stage, can be executed concurrently but not independently. The processing of every parallel task is linked to data provided by other tasks, so that every data needs to be transferred between parallel tasks in order to accomplish the global task of the system[1].
   Every message sent involve a physical cost, so that is desired to avoid unnecessary communications. Local communications implies two geographically close communicating processes. In contrast, remote communications implies two processes that communicates through a network medium[1].

3. **Agglomeration and Granularity Adjustment.** The agglomeration stage is responsible for control the granularity to increase the processing or decrease the communication costs. The main idea is to use the locality, i.e., to group some tasks will help to reduce the communications over the network[4]. The main goals of the agglomeration stage are: workload balancing, decrease of communications costs, and decrease of the allocation management delay[1,

5].

4. **Mapping.** The parallel modules must be mapped or allocated into the processors to be executed. The mapping problem consist of how to assign process into processors, also it is defined as the problem of maximize the number of communicating processes pairs allocated in directly connected processors[2, 4]. The allocation can be specify statically by a load balance algorithm or can be determined in execution time[3].

2.2 Network Flows

A network flow $G_{nf} = (V, E)$ is a strictly directed graph, where each edge $a = (u, v) \in E$ has a capacity $c(a) \geq 0$. If E has an edge $a = (u, v)$, then there is not an edge $a_r = (v, u)$ in the opposite direction[6].

A network flow has two special vertices, one source vertex s, and one target vertex t. The source vertex is responsible of generate flow to be route through the edges of the network[3, 6].

Assume that, $\forall\, v \in V$ there is a *path* $s \to v \to t$, such that the graph is connected. Note that $v \in V - \{s\}$ has at least one incident edge, then $|E| \geq |V| - 1$[6].

There are two major problems in this type of graphs, which are presented below:

Let A and B be two subsets in V, then:

1. **Min Cut Problem:**
   An $s - t$ cut, where $s \in A$ and $t \in B$, is a partition of the set V into two groups $V = \{A, B\}$.
   The capacity of a cut is defined as $c(A, B)$, which is equal to the sum of capabilities of each edge $e \in E$ that goes out from A [3].

$$cap(A, B) = \sum_{e\ goes\ out\ A} c(e) \tag{4}$$

   The *minimum cut problem* refers to find an $s - t$ cut, such that $c(A, B)$ is the minimum possible[3].

2. **Max Flow Problem:**
   An $s - t$ flow is defined as a function that satisfies two properties[3]:
   (a) **Capacity:** All assigned flow to an edge e should be less or equal than its capacity $c(e)$.

$$0 \leq f(e) \leq c(e), \;\; \forall\, e \in E \tag{5}$$

   (b) **Preservation:** The total flow entering to a vertex $v \in V - \{s, t\}$, should be equal to the total flow coming out from it.

$$\sum_{e\ goes\ to\ v} f(e) = \sum_{e'\ goes\ out\ v} f(e'), \;\; \forall\, v \in V(G) - \{s, t\} \tag{6}$$

The *maximum flow problem* refers to find an $s-t$ flow of maximum value, with no infringement of the capacity and preservation properties[3, 6].

Every edge $e=(u,v)$ in a network flow G_{nf} has a residual edge $e_r=(v,u)$ associated to it, such that $c(e_r)=f(e)$. The residual edge is allowed to transfer flow directed to the target vertex t, so that when a flow g pass through a residual edge e_r then $f(e)=f(e)-g$.

It turns out that the min cut problem and the max flow problem are closely related, and it is shown by the next lemma.

The *net flow* across a cut (A,B) is the sum of the flow on its edges from A to B, minus the sum of the flow on its edges from B to A.

Lemma 1 (Flow value). *Let f be any flow assigned to edges of E. Let (A,B) any $s-t$ cut from the network flow. Then, the net flow sent across the cut is equal to the value of f[3].*

$$val(f)=net(f)=\sum_{e\ goes\ out\ A} f_{out}(e)-\sum_{e\ enters\ to\ A} f_{in}(e) \tag{7}$$

By the previous lemma is easy to see the duality of this problems, such that the maximum value of a flow f, across a (A,B) cut, should be less or equal than the minimum cut's capacity on the network.

$$val(f)=net(f)\ across\ (A,B)\leq c(A,B) \tag{8}$$

There exist an algorithm to find the max flow and the min cut over a network flow called Ford-Fulkerson algorithm. The algorithm is based on one important concept called *augmenting path*. An augmenting path is a simple directed path, from the source vertex s to the target vertex t, with positive capacities edges, such that, the flow over the network can be increased[6].

Theorem 1. Augmenting Path Theorem
A flow f, which is obtained by the Ford-Fulkerson algorithm, is maximum if and only if, there is no more augmenting paths in the network flow[6].

As a corollary by previous theorem and lemmas:

Theorem 2. Maximum Flow Minimum Cut Theorem
Let f be a $s-t$ flow such that there is no an augmented path in the graph G. Let (A,B) be an $s-t$ cut in G such that $net(f)=c(A,B)$. f is the maximum flow value in G, and $c(A,B)$ is the minimum capacity for every $s-t$ cut in G [6].

3 The Agglomeration Problem

As mentioned in the *Agglomeration and Granularity Adjustment* stage of the methodology presented in section 2.1, is in this step where communication costs of a parallel system can be addressed. To agglomerate a set of processes is necessary to group some of them in accordance to a given criterion, for the purposes of this work the main criterion is the minimization of communication costs. The agglomeration problem can be reduced into a more general problem called graph partitioning problem, most of this problems are known to be NP-Hard problems, meaning that there is no an efficient way to solve them, instead, are proposed some heuristics and approximation algorithms[11].

Let $G = (V, E)$ be a graph with weighted edges, such that $|V| = n$. The (k, v)-balanced partitioning problem, for some $k \geq 2$, aims to decompose G into subsets at most size $v\frac{n}{k}$, while the total weight of the edges connecting two vertices from different components are minimum[11]. Particularly, the k-balanced partitioning problem is shown as a *NP-Complete* problem in [11]. Even $(2, 1)$-balanced partitioning problem, which seems to be more easy, is also an *NP-Complete* problem[8, 10].

The minimum set of edges to disconnect a graph into two components can be efficiently found by the Ford-Fulkerson algorithm[3]. Note that the minimum cut set can isolate even just one vertex from the graph, resulting in a very unbalanced partition, but the balanced partitioning is not the propose of this work.

Consider the network flow shown in figure 1. Such graph has a minimum cut shaped by edges $(s, 2)$ y $(3, 5)$ of value 19.

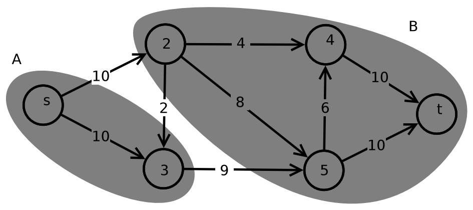

Fig. 1. Network Flow. Grey shapes represent the two partitions (A, B) computed by the Ford-Fulkerson algorithm.

Note that edge $(2, 3)$ is incident to partition A, recalling cut's capacity definition says:

> *"The capacity of a cut is equal to the sum of capacities of the edges pointing outward the partition."*

Remark 1. By definition the capacity of a cut does not consider any edge incident to partition A, but in the context of parallel processes such edges are communications among processes of the system, such that, the flow transmitted by those edges must be considered for the agglomeration step.

therefor it can be say that:

Definition 1. *A minimum communication cut, for the parallel processes agglomeration problem, is the sum of capacities of edges directed to partition A plus the sum of capacities of edges directed to partition B.*

Taking into account the remark 1, it is easy to see that in figure 1 the value of the cut turns into a cut of value 21. If it is found a minimum communication cut over the network flow shown in figure 1, there is one of value 20 given by the edges $(2, 4)$, $(5, 4)$ y $(5, t)$ (see figure 2).

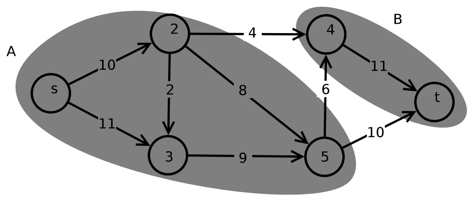

Fig. 2. Minimum communication cut from the network flow shown in figure 1.

3.1 The minimum communication cut algorithm

In the following is presented the algorithm 1 to compute the minimum communication cut over a network flow G.

Algorithm 1 Minimum Communication Cut(G, s, t)

```
$mincut(G) \leftarrow$ Ford-Fulkerson$(G, s, t)$
while $\exists\ e \in mincut(G)$ incident to $A$ do
    for all $e \in mincut(G)$ incident to $A$ do
        $G \leftarrow G - e$
        $G \leftarrow G + e_{inverse}$
    end for
    $mincut(G) \leftarrow$ Ford-Fulkerson$(G, s, t)$
end while
```

The algorithm 1 takes the Ford-Fulkerson algorithm as a black box to compute a minimum cut $mincut(G)$ over the network G. Once the minimum cut is known, algorithm 1 checks if there is at least one edge e in $mincut(G)$ incident to partition A, in such case, algorithm 1 exchanges e by its inverse $e_{inverse}$. The algorithm finishes when a minimum cut with no edges incident in partition A is computed.

Note that reversing an edge whose flow is greater than zero may cause the violation of conservation and capacity properties. Because of this, is important to prove the next lemma.

Let $G = \{V, E\}$ be a network flow. Let $mincut(G) = \{e_1, e_2, ..., e_k\}$ be the minimum set of edges to disconnect G into two subsets $A, B \subset V$, such that every edge in $mincut(G)$ goes from partition A to partition B or viceversa.

Lemma 2. *Every edge $e = (y, x) \in mincut(G)$ where $y \in B$ y $x \in A$ has a flow assigned $f(e) = 0$.*

Proof. By contradiction suppose that $f(e) > 0$.

Note that $mincut(G)$ is a minimum cut of G, then for all edge $e' \in mincut(G)$ that goes from partition A to partition B has a flow $f(e') = c(e')$, otherwise it would not be a minimum cut.

W. l. g. suppose there is at least one edge e that goes from partition B to partition A, which by assumption has flow $f(e) > 0$, therefor, there is a residual edge e_r assigned to e with $c(e_r) > 0$, thus there are two cases:

1. There is at least one path P_{s-t} that uses e_r to transfer flow from the vertex s to vertex t, thus $mincut(G)$ is not a minimum cut of G.
2. There is no a path P_{s-t} that uses e_r to transfer flow from the vertex s to vertex t, meaning that there is one $mincut(G)'$ such that $c(mincut(G)') < c(mincut(G))$.

Any case contradicts the assumptions $\Rightarrow\Leftarrow$.

4 The Agglomeration Strategy

In this section are presented the three steps of the proposed strategy, and are described some additional considerations.

4.1 Building the Software Graph

Let consider the parallel processes defined in the decomposition stage of the methodology presented in section 2.1, such processes can be represented as a set of nodes or vertices V_{sw}. Let consider now the set of relations between two process E_{sw} established in the communications stage 2.2 from the same methodology. A software graph is defined as $G_{sw} = \{V_{sw}, E_{sw}\}$, which is an abstract representation to model a parallel system structure.

Let $p_i, p_j \in V_{sw}$ be two processes. If p_i establish a directed communication channel to p_j, then there is an edge e_{ij} in the set of edges E_{sw}.

$$E_{sw} = E_{sw} \cup \{e_{ij}\} \mid e_{ij} = (p_i, p_j) \tag{9}$$

since e_{ij} has an associated capacity $c(e_{ij})$ where:

$$c(e_{ij}) = n, \mid n \in \mathbb{N}^+ \tag{10}$$

An edge capacity $c(e_{ij})$ is equal to the sum of data units interchanged by processes p_i and p_j during execution time. The amount of communications are totally dependent by the nature of the parallel system, and if the amount of communications can be well defined and represented as a positive integer, otherwise has no sense to agglomerate by using this strategy.

4.2 Transformation to a Network Flow

In order to transform the software graph into a network flow, is necessary to classify the vertices of G_{sw} considering the following criterion:

- **Initial Vertices or Flow Generators:** Most of them are main processes which generates and sends information to other processes in the system. Sometimes they communicate each other or do some processing.
- **Dealers or Processing Vertices:** They receive data from flow generator vertices. Usually they do some processing but its main function is to distribute data across end vertices.
- **End Vertices:** End vertices represent slave processes, which main purpose is to do processing tasks with data received from initial or dealer vertices. Sometimes they communicate each other.

Let $G_{sw} = \{V_{sw}, E_{sw}\}$ be the software graph. Let $G_{nf} = \{V_{nf}, E_{nf}\}$ be the network flow, such that G_{nf} is directed and has no parallel edges.

1. Let s, t be two vertices, s be a source vertex and t be target vertex, then:

$$V_{nf} = V_{sw} \cup \{s, t\} \tag{11}$$

2. Vertex s should have a directed edge e_g for every generator vertex v_{gen} in the software graph, such that:

$$E_{nf} = E_{sw} \cup \{e_g\} \ \mid \ e_g = (s, v_{gen}) \ \forall \, v_{gen} \ \in \ V_{sw} \tag{12}$$

The capacity $c(e_g)$ of every edge $e_g = (s, v_{gen})$, should be equal or greater than the sum of capacities of every edge that goes out from v_{gen}.

$$c(e_g) \geq \sum c(e_{out}) \ \mid \ e_{out} = (v_{gen}, u), \ \ \forall \, e_{out} \in v_{gen} \tag{13}$$

3. For every end vertex v_f, there is a directed edge e_f ending at t:

$$E_{nf} = E_{sw} \cup \{e_f\} \ \mid \ e_f = (v_{end}, t), \ \ \forall \, v_{end} \ \in \ V_{sw} \tag{14}$$

The capacity $c(e_f)$ of every edge $e_f = (v_{fin}, t)$, should be equal o greater than the sum of capacities of every edge that ends in v_{end}.

$$c(e_f) \geq \sum c(e_{in}) \ \mid \ e_{in} = (u, v_{gen}), \ \ \forall \, e_{in} \in v_{end} \tag{15}$$

4.3 Applying the Algorithm

Once the network flow is builded, is necessary to apply a partitioning algorithm over G_{sw} to obtain the agglomeration of processes. The algorithm 1 obtains a bipartition through the minimum communication cut, which holds thats is the minimum set of edges with less weight to divide the graph into two subsets.

The algorithm 1 takes as input parameter a network flow G, one source vertex s, and one target vertex t. The algorithm 1 gives as output a couple of subgraphs A and B.

The algorithm 1 can be applied recursively over the subgraphs A and B in order to obtain k agglomerations of the parallel processes, thereby can be assigned each agglomeration directly over the k processing units or nodes of the distributed system.

5 Case Study

If a matrix is symmetric and positive defined, then it has an special triangular decomposition. symmetric and positive defined matrices are very special and they appear very frequent in some applications. It turns that this types of matrices has an special factorization method called Cholesky decomposition, which it two times faster than other alternatives to solver linear system equations[11].

Due to symmetry of this matrices it is clear enough to work only at one side, say the inferior one. Figure 3 shows the data dependency to compute the Cholesky decomposition in a 4x4 matrix.

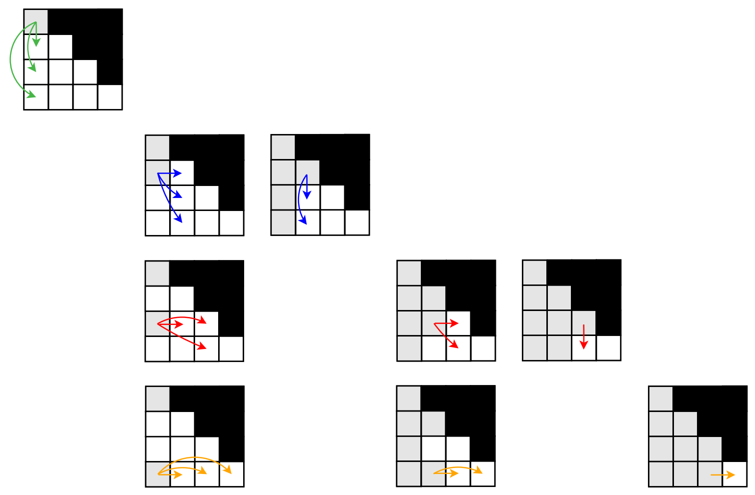

Fig. 3. Data dependency for the Cholesky decomposition on a 4x4 matrix. Gray cells denote an element already computed at that position. Arrows denote the need to transfer information between cells.

Particularly for this instance the problem is decomposed by cells, meaning that every cells from the matrix represents a parallel process. Arrows pointing a cell represent a communication necessity. For this problem the amount of data transfer on each edge is one unit. Figure 4 shows the software graph to model de communications structure of the parallel system.

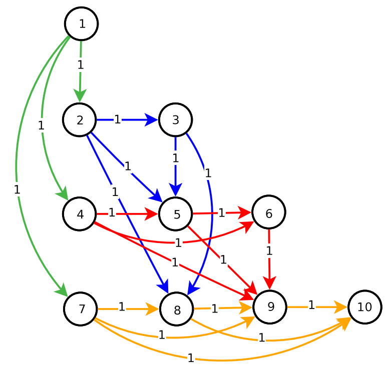

Fig. 4. Software graph for a cell decomposition on a 4x4 matrix.

The total amount of communicant between two processes of the system shown in figure 4 is 20 units.

Based on the vertex classification presented in section 4.2, is easy to see that vertex 1 is the only one flow generator vertex, on the other hand vertex 10 is the only one end vertex, so that for this example is not necessary to add the special vertices s and t.

Consider the network flow shown in figure 4. For the agglomeration stage, consider the execution of the parallel system onto a *cluster* of 4 distributed nodes. Figure 5 shows the partition obtained by the algorithm 1 executed over the network flow shown in figure 4. The local and remote communication costs of the agglomeration are shown in the following:

$$A : c_l = 0,\ c_r = 3$$
$$B : c_l = 0,\ c_r = 0$$
$$C : c_l = 0,\ c_r = 1$$
$$D : c_l = 9,\ c_r = 7$$

Such that, the communication time CT of the agglomeration shown in figure 5 is:

$$CT = 9 * t_{constant} + 11 * t_{variable} \tag{16}$$

In order to establish a benchmark for network flow based strategy is needed to agglomerate or map the same parallel system but with a different strategy.

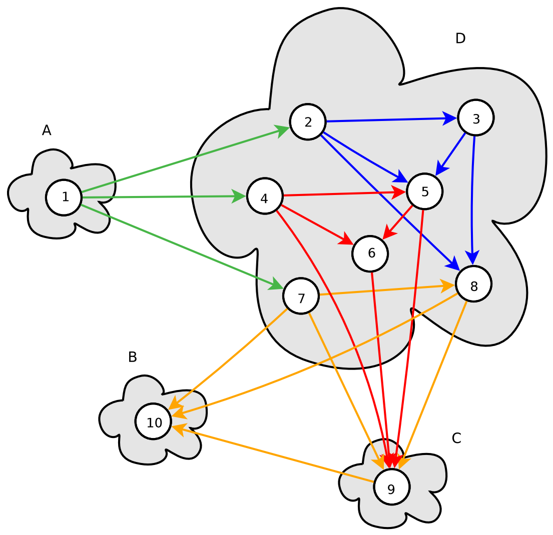

Fig. 5. Agglomeration of the network flow of the Cholesky decomposition problem.

For this purpose consider the well-known strategy called Round Robin, which is widely used by most operating systems for scheduling processes.

The Round Robbin mapping proceeds assigning the process 1 to agglomeration A, process 2 is allocated to agglomeration B, process 3 is allocated to agglomeration C, process 4 is allocated to agglomeration D, and in a circular way the next process is allocated once again to agglomeration A. This assignment is affected consecutively until the processes are completely assigned. Figure 6 shows the agglomeration into 4 group based with the Round Robbin technic.

The communications costs of the agglomerations shown in figure 6 are:

$$A : c_l = 1,\ c_r = 5$$

$$B : c_l = 0,\ c_r = 4$$

$$C : c_l = 0,\ c_r = 5$$

$$D : c_l = 0,\ c_r = 5$$

The total amount of communications time is:

$$CT = 1 * t_{constant} + 19 * t_{variable} \tag{17}$$

6 Conclusiones

The performance of a parallel system is inherently affected by communications between processes. The communication time added to the execution time of the

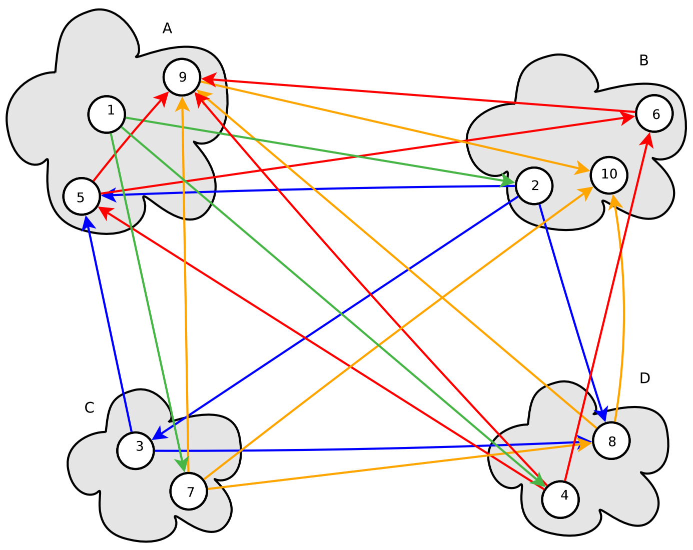

Fig. 6. Agglomeration affected with the round robin technique

system is proportional to the amount of data exchanged by the parallel system and by the type of communication that implements.

In general, communications through shared memory are less expensive than communications via a network medium. Such that, it is necessary to maximize the amount of local communications $c_l(P)$ and to minimize (to the possible extent) the amount of remote communications $c_r(P)$, to minimize the total communication time in order to mitigate the impact on the execution time of the parallel system.

References

1. Foster, I.: Design and Building Parallel Programs v1.3: An Online Publishing Project (1995)
2. Bokhari, S.: On the Mapping Problem: IEEE Transactions on Computers, Vol. c-30, No. 3, March (1981)
3. Kleinberg, J., Tardos, E.: Algorithm Design: Pearson-Addison Wesley (2005)
4. Chandy, M., Taylor, S.: An Introduction to Parallel Programming: Part II Parallel Program Design Chapter 7, Jones and Bartlett (1992)
5. Culler, D., Singh, J., Gupta A., Kaufmann, K.: Parallel Computer Arquitecture, A Hardware / Software Approach,: Chapter 2 Parallel Programs, (1997)
6. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, Section 26.2: The Ford-Fulkerson method, MIT Press and McGraw-Hill Second Ed. (2001)
7. Blaise B.: Introduction to Parallel Computing: Lawrence Livermore National Laboratory.
8. Steven S.: The Algorithm Design Manual, Chapter 4 Sorting and Searching, Second Edition Springer (2008)

9. Bondy, J., Murty, U.: Graph Theory: Springer, (2008)
10. Garey, M., Johnson D.: Computers and Intractability, A Guide of the Theory of NP-Completeness: Bell Telephone Laboratories, (1979)
11. Andreev, K., Racke, H.: Balanced Graph Partitioning