ABSTRACT. We consider the speed-up that can be obtained for a parallel program in a distributed system. The modelling of parallel programs by acyclic graphs allows the study of the problem of distributed processing as a graph partitioning problem. Such a model consists of nodes representing the tasks that compose the program (set of instructions that must be executed sequentially), and of arcs that represent the precedence order between tasks. Thus, the problem turns out to find the task graph partitioning which maximizes execution speed-up of the parallel program. This problem is $\text{NP}$-complete. To optimize, we specially use a method based on the brain behavior, the random neural model of Gelenbe, and we compare it with other approximate methods: an algorithm based on the simulated annealing optimization heuristic, a heuristic using genetic algorithm and an algorithm based on Kernighan’s heuristic.

Keywords: distributed processing, neural networks, parallel programs.

1. Introduction

This paper concerns the execution speed-up of a parallel program in a distributed system. The program is modelled as an acyclic graph, where the nodes represent the tasks, and the arcs represent the relations between tasks. The modelling of parallel programs as acyclic graphs allow us to study the optimization of the execution speed-up as a graph partitioning problem.

The cost function that allows us to optimize the graph partitioning can be defined as the maximization of the acceleration on the parallel program execution, with a load balance constraint in the system. The graph partitioning problem with this cost function, is an $\text{NP}$-hard problem, because the possible solutions increase exponentially with the problem size.

We are interested in approximate solution methods, because exact solutions have impractical execution times when the graph size is large. The approximate methods give suboptimal solutions in a reasonable execution time (polynomial time). We specially present a method based on the brain behavior, the random neural model of Gelenbe [5, 6] and we compare it with other approximate methods: one based on the principles of the evolution biology (the genetic algorithms), one based on the simulated annealing optimization heuristic, and another derived of the Kernighan’s heuristic.

This work is organized in four parts. First, we define the problem. Then, we present the proposed methods and their applications to our problem. The third part presents the analysis of the results. Finally, we give some perspectives for future study.
2. Problem Definition

The objective of this paper is to study how to optimize the execution of a parallel program over a distributed architecture in order to improve the system performance. To study a parallel program, we model it as an acyclic direct graph. The graph is acyclic, because we are interested in program that terminate, and direct, because there are precedent relations between the tasks that determine their execution order. On our model, every node of the graph represents an individual task, and the direct arcs represent the dependence relation between the tasks. Every task has a different execution time and needs a processor for its execution.

The graph is characterized by three parameters: the number of tasks in the program, the execution time of each task and the average degree of the tasks. The mathematical formulation is as follows:

\[ \Pi = (N, A) \]

where, \( \Pi \) is a directed graph

- \( N \) is the set of \( n \) tasks that compose the program, which has a weight function \( \Omega \) associated to it that defines the execution time of every task. In our study: \( \Omega(i) = 1 \) \( \forall i = 1 \ldots n \).
- \( A = a_{ij} \) is the adjacent matrix. It represents the precedence order between the tasks. Besides, we number the tasks in a decreasing manner, such that: \( a_{ij} = 1 \) if \( i \) and \( j \) are adjacent and \( i < j \)
  \( 0 \) otherwise.

The problem consists in to distribute the execution of tasks among the different \( K \)-processors \( (\Pi_1, \ldots, \Pi_K) \) of the system, in such a way that optimizes its performance. We consider distributed systems, which consist of a collection of \( K \)-homogeneous computers, each with local CPU and memory. The computers are totally connected between them. This structure models a message passing parallel architecture.

Thus, the problem is characterized according to the following constraints:

- Communication between different processors of the system must be minimum.
- Load between different processors in the system must be balanced.
- Execution time of the parallel program must be minimum.

The modelling of the parallel program as a graph enables the definition of our problem as a graph partitioning problem, which is the same to the study in [1, 2], where every processor will be a partition. We use two types of graph to model the parallel program: the first, with only the acyclic constraint that represents the tasks execution order in a parallel program; and the second, with the constraint of a "series-parallel" structure on the graph, that represent the fork-join structure common to these types of programs.

To study this problem as a graph partitioning optimization problem, which maximizes the speed-up of a parallel program, we use the following cost function:
\[ F = \left( \frac{n}{h} \right) \max \left\{ L_c + \left( \sum_{i \in c} a_{ij} \right) \right\}, \quad 1 \leq c \leq h \quad i, j \in R_1 \]

where,  
- \( h \) = number of paths in the graph.  
- \( L_c \) = length of the path \( c \).  
- \( R_1 = \{ i \in \Pi_l, j \in \Pi_r, \& r \neq 1 \& a_{ij}=1 \} \)

and the constraint  
\[ n_1 \leq n/K + E_0 \quad \text{for } \forall 1 = 1 \ldots K \]

where,  
- \( E_0 \) is the imbalance interval.  
- \( n_1 \) is the number of nodes in each processor.

To define this formula, we use the works [3, 4], according to which the execution speed-up of a parallel program is defined as:

\[ S = \frac{Ts}{Tp} \]

where,  
- \( Ts \) is the sequential execution time.  
- \( Ts = \sum_{i=1}^{n} \Omega(i) = n \)
- \( Tp \) is the parallel execution time.

\( Tp \) is equivalent to the length of the critical path of a graph, if we assume an unlimited number of processors. That is, if the program represented by the tasks graph is executed with an unlimited number of processors, then \( Tp \) will be the corresponding total time of execution, assuming ideal conditions (for example: negligible communication time, unlimited memory, ...). Thus, \( Tp \) is the minimum execution time of a program and the execution time will always be at least as large as \( Tp \) when the number of processors is finite rather than unlimited.

The critical path is the largest path of the graph. A path is an ordered sequence of nodes \( (i_1 < \ldots < i_j) \), such that the first node \( (i_1) \) has no predecessor and the last \( (i_j) \) has no successor:

\[
\begin{align*}
    a(i, i_1) &= 0 & \text{for } i < i_1 \\
    a(i_j, j) &= 0 & \text{for } j > i_j \\
    a(i_k, i_{k+1}) &= 1 & \text{for } k = 1 \ldots j-1
\end{align*}
\]

The length of a path \( c \), is the sum of the execution times of the tasks on the path:

\[ L_c = \sum_{i_k \in c} \Omega(i_j) \]

And, the critical path is the path with the largest length of the graph:

\[ \max \left( \sum_{i_k \in c} \Omega(i_j) \right) = \max \left( L_c \right), \quad 1 \leq c \leq h \]

We add the communication cost of a task with their successors, if they are on a different processor. We suppose that the adjacent matrix defines the communication cost between two tasks \( i \) and \( j \) and will be equal to 1 if they are in different processors. We assume
that communication between task assigned to the same processor is negligible. Thus, the function is defined as:

\[ F_1 = \max ( \sum_{i_k \in c} \Omega(i_j) + \sum_{i_k, j \in R^2} a_{i_k, j} ) \]

for \( 1 \leq c \leq h \)

where \( R^2 = \{ i_k \in \prod I \& j \in \prod J \& r \neq 1 \& a_{i_k, j} = 1 \} \)

Finally, we optimize:

\[ F_0 = \min(F_1) \]

with the balance constraint \( n_1 \leq n/K + E \) for \( \forall 1 = 1 \ldots K \)

3. The Random Neural Model

This model has been developed by Gelenbe [5, 6]. It is an approximate technique based on the brain behavior. The model describes the dynamic system through a neural network, where every neuron has a nonlinear equation that describes its internal state. Thus, the neurons are defined by their internal states and their connections with other neurons or with the environment. This model proposes a scheme of interaction between the neurons and not a dynamic equation of the system.

The model is based in the queuing theory: the neurons are queues and the signals (coming from the environment or emitted between neurons) are flows of clients, which can be positive (excitation) or negative (inhibition). Gelenbe [5] has proved that if the external arrival is a poisson process and the service times are exponential, the network with positive and negative signals has a solution in product form. This result permits to calculate the excitation probability of the neurons.

Every neuron has a counter which is increased by one when a positive signal arrives, or is decreased by one, if it is positive and a negative signal arrives. The signal counter is known as the potential of the neuron "i" at the instant \( t \). If the potential is positive at the instant \( t \) (\( k_i(t) > 0 \)), the neuron "i" is excited and can emit positive and negative signals to the neuron \( j \) according to the probabilities \( p^+(i,j) \) and \( p^-(i,j) \) respectively, or can emit outside signals according to the probability \( d(i) \), with a firing rate \( r(i) \). According to this:

\[ \sum_{j=1}^{\infty} p(i,j) + d(i) = 1 \quad \text{for } 1 \leq i \leq n \quad \text{and} \quad p(i,j) = p^+(i,j) + p^-(i,j) \]

The excitation probability of a neuron "i" is the nonlinear equation given by:

\[ q(i) = \lambda^+(i)/(r(i) + \lambda^-(i)) \]

where:

\[ \lambda^+(i) = \sum_{j=1}^{\infty} q(j)r(j)p^+(j,i) + \lambda(i) \]

\[ \lambda^-(i) = \sum_{j=1}^{\infty} q(j)r(j)p^-(j,i) + \lambda(i) \]
and \[ \Lambda(i) = \text{arrival rate of external positive signals}, \]
\[ \lambda(i) = \text{arrival rate of external negative signals} \]

The next condition is sufficient for the existence of a unique solution for 2.1 and 2.2 equations, as is demonstrated in [5]

\[ \Lambda(i) + \sum_{j=1}^{n} r(j)p^+(j,i) < r(i) + \lambda(i) \]

Such that if \( q(i) \leq 1; \quad p(k) = \prod_{i=1}^{n} (1-q(i))q(i)^{k_i} \) is the stationary probability, where \( k \) is the neuron potential vector.

3.1. First Random Neural Model for our problem

In this model (RNM1), we study the space of possible solutions [2]. To model this, we use \( nK+K \) neurons of two types: \( nK \) neurons, \( N1(i,k) \), that represent one element of the solution space, where \( i \) is the task \( T_i \) and \( k \) the processor \( \prod_k \); and \( K \) neurons, \( N2(k) \), that represent the load regulator of every processor.

a) For \( N1 \), a probability value of excitation \( q1(i,k) \) near one signifies that this solution is admitted. They emit negative signals to:

- The incompatible solutions, with probability \( p1^-((i,k),(i,z)) \) (where \( N1(i,k), N1(i,z) \) and \( k \neq z \)).
- The successor tasks located on different processors, with probability \( p1^-((i,k),(j,z)) \) (where \( N1(i,k), N1(j,z), k \neq z \) and \( a_{ij}=1 \)).
- The regulator \( k \) if the task \( i \) belongs to this processor, with probability \( p1^-((i,k),k) \) (where \( N1(i,k) \) and \( N2(k) \)).

They emit excitation signals to:

- The successor tasks located on the same processor, with probability \( p1^+((i,k),(j,k)) \) (where \( N1(i,k), N1(j,k) \) and \( a_{ij}=1 \))

b) For \( N2(k) \), they emit only positive signals to:

- The tasks belonging to processor \( k \), with probability \( p2^+(k,(i,k)) \) (where \( N2(k), N1(i,k) \) and for \( i = 1 \ldots n \)).
- The other load regulators, with probability \( p2^+(k,z) \) (where \( N2(k), N2(z) \) and \( k \neq z \)).

A probability value of excitation \( q2(k) \) near 0, signifies that the processor has arrived to its maximal capacity. Besides, the neurons have a positive input of the environment, that defines the capacity of every processor.

\[ \Lambda2(k) = n/K \quad \forall k=1\ldots K \]

The equations of the system are:

\[ q1(i,k) = X1 / Y1 \]

where
\[ X_l = \sum_{z=k} \sum_{(j \neq i \text{ and } a_{ij} = 1)} q_1(j, z) p_1^+((j, z), (i, k)) r_1(j, z) + q_2(k) r_2(k) p_2^+(k, i, k) \]

\[ Y_l = r_1(i, k) + \sum_{z \neq k} \sum_{j=1} q_1(j, z) r_1(j, z) p_1^-((j, z), (i, k)) \]

\[ q_2(k) = \left[ \lambda_2(k) + \sum_{z \neq k} q_2(z) r_2(z) p_2^+(z, k) \right] / \left[ r_2(k) + \sum_{i=1} q_1(i, k) r_1(i, k) p_1^-((i, k), k) \right] \]

And, the model parameters:

- \( d_1(i, k) = d_2(k) = \lambda_1(i, k) = \lambda_2(k) = \Lambda_1(i, k) = 0 \)
- \( \Lambda_2(k) = n/K \)
- \( r_1(j, z) p_1^+((j, z), (i, k)) = 1 \) if \( (z=k \text{ and } a_{ij}=1) \)
- \( r_1(j, z) p_1^-((j, z), (i, k)) = 1 \) if \( (z \neq k \text{ and } (a_{ij}=1 \text{ or } j=i)) \)
- \( r_1(i, k) p_1^-((i, k), k) = 1 \) if \( i \in k \)
- \( r_2(k) p_2^+(k, (i, k)) = 1 \) if \( i \in k \)
- \( r_2(z) p_2^+(z, k) = 1 \) if \( z \neq k \)
- \( r_1(i, k) = (K^* n - 1) \text{degree}(i) + 1 \)
- \( r_2(k) = n + K - 1 \)

3.2. Second Random Model for our problem

In this model (RNM2), we start with an initial solution that we will try to improve \[2\]. To model this, we use \( n+K \) neurons of two types: the first \( n \) neurons, \( N_1(i, k) \), that represent the processor \( k \) where belongs the task \( i \); and the remaining \( K \) neurons, \( N_2(k) \), that represent the load regulator for every processor.

a) The \( N_1 \) neurons emit positive signals to:

- The successor tasks if are in the same processor, with probability \( p_1^+((i, k), (j, k)) \) (where \( N_1(i, k), N_1(j, k) \) and \( a_{ij}=1 \)).

And negative signals to:

- The successor tasks if are in different processors, with probability \( p_1^-((i, k), (j, z)) \) (where \( N_1(i, k), N_1(j, z), k \neq z \) and \( a_{ij}=1 \)).
- The regulator \( k \) if the task \( i \) belongs to this processor, with probability \( p_1^-((i, k), k) \) (where \( N_1(i, k) \) and \( N_2(k) \)).

A probability value of excitation \( q_1(i, k) \) near one, signifies that the task is accepted in that processor.

b) For \( N_2(k) \), they emit only positive signals to:

- The tasks belonging to processor \( k \), with probability \( p_2^+(k, (i, k)) \) (where \( N_2(k), N_1(i, k) \) and for \( i=1 \ldots n \)).
- The other load regulators, with probability \( p_2^+(k, z) \) (where \( N_2(k), N_2(z) \) and \( k \neq z \)).
Besides, the neurons have a positive input of the environment, that defines the capacity of every processor,

\[ A2(k) = \frac{n}{K} \quad \forall k = 1 \ldots K \]

A probability value of excitation \( q2(k) \) near 0, means that the processor has arrived to its maximal capacity.

The equations of the system are:

\[ q1(i,k) = X2 / Y2 \]

where

\[ X2 = \sum_{z=k} \sum_{j\neq i & a_{ji}=1} q1(j,z)p_{1}^+(j,z),(i,k) + q2(k)r_{2}(k)p_{2}^+(k,(i,k)) \]

\[ Y2 = r_{1}(i,k) + \sum_{z \neq k} \sum_{j\neq i & a_{ji}=1} q1(j,z)r_{1}(j,z)p_{1}^-(j,z),(i,k) \]

and,

\[ q2(k) = \frac{[A2(k) + \sum_{z \neq k} q2(z)r_{2}(z)p_{2}^+(z,k)]}{[r_{2}(k) + \sum_{i \in k} q1(i,k)r_{1}(i,k)p_{1}^-((i,k),k)]} \]

And the model parameters:

- \( d1(i,k) = d2(k) = A1(i,k) = A2(k) = A1(i,k) = 0 \)
- \( A2(k) = \frac{n}{K} \)
- \( r_{1}(j,z)p_{1}^+(j,z),(i,k) = 1 \) if \( (z=k \text{ and } a_{ji} = 1) \)
- \( r_{1}(j,z)p_{1}^-((j,z),(i,k)) = 1 \) if \( (z \neq k \text{ and } a_{ji} = 1) \)
- \( r_{1}(i,k)p_{1}^-((i,k),k) = 1 \) if \( i \in k \)
- \( r_{2}(k)p_{2}^+(k,(i,k)) = 1 \) if \( i \in k \)
- \( r_{2}(z)p_{2}^+(z,k) = 1 \) if \( z \neq k \)
- \( r_{1}(i,k) = \text{degree}(i) + 1 \)
- \( r_{2}(k) = \frac{n}{K+K-1} \)

5. Performance Evaluation

We compare the random neural model with the approximate methods studied in [1]: genetic algorithm (GA), simulated annealing (SA) and kernighan's heuristic (H). We have used the parameters that give the better performance in every method, according to the results of the work [1]. We have used a SUN SPARCstation IPC with 16M of memory and a matrix as data structure. The two types of graphs used: acyclic (G1) and series-parallel (G2); are defined for the number of nodes in the graph and the average degrees of the nodes (d). The execution time is in seconds.

5.1. Results analysis for acyclic graph

For this type of graph (figure 1, 2), the execution times are very large. The genetic algorithm, the simulated annealing and H, for
graphs of 50 or more nodes, need a very large time to reach the suboptimal solutions.

For graph of little size (of 15 or less nodes), the difference between the exact solution and the results of the other methods is important, but the execution times are similar, what makes more interesting the exact solutions. Otherwise, the approximate methods are more interesting, because they have a reasonable execution time.

Figure 1. Results of the simulation for acyclic graph with \( b = 1, K = 5 \) and \( d = 2 \)

Figure 2. Execution time of the simulation for acyclic graph with \( b = 1, K = 5 \) and \( d = 2 \)

5.2. Results analysis for series-parallel graph

With this type of graph (figure 3, 4), the execution time to reach a suboptimal solution is small. For graph of little size, \( H \) gives the better results. For graphs of 50 or more nodes, Genetic Algorithm, RNM1, RNM2, Simulated Annealing and \( H \) give approximately the same results, but the Genetic Algorithm and the Simulated Annealing need much more execution time.
Figure 3. Results of the simulation for series-parallel graph with $b = 1$, $K = 5$ and $d = 2$

Figure 4. Execution time of the simulation for series-parallel graph with $b = 1$, $K = 5$ and $d = 2$

6. Conclusions

In our study, the Genetic Algorithm gives the better result, but with a large execution time. The Random Neural Model gives good results with short execution time. In this model, the solution that works with all the space of solutions gives the better results.

The execution time for the Genetic Algorithm and Simulated Annealing are very large. For Genetic Algorithm, the reason is that generation calculations take relatively much more time. It is necessary to determine the better combination of genetics operators, to decrease the number of necessary generations to reach the suboptimal solution. For Simulated Annealing, since it is not possible determine coherent movements of nodes in every temperature level that decrease the energy, the solution is evaluated in a relatively longer time.

The Genetic Algorithm and the Random Neural Model are easy to implement in parallel machine, which can improve the results obtained
with these methods. Future works will go toward the use of combinatorial optimization methods on other types of NP-complete problems in the distributed system (task assignment, file allocation, ...), toward the study of a possible improvement of the Random Neural Model which will use the better solutions found through the evolution of the system as input of itself until it arrives to a suboptimal solution, and toward the implementation of the Random Neural Model and the Genetic Algorithm over parallel machines.

References

4 GELENBE, E. "Note on the critical path length of large acyclic task graph". Rapport interne. EHEI. Université Rene Descarte. 1989.