Heuristic Partitioning Algorithm with Partial Overlapping for a System of Equations Distributed Solution

Benjamín Barán and Diana Benítez
National Computing Center (CNC)- National University of Asuncion (UNA)
San Lorenzo, P.O. BOX 1439
Paraguay

ABSTRACT

Parallel asynchronous implementations of iterative algorithms in heterogeneous computer environments are becoming a very convenient choice for solving large scale systems of equations; however, the usefulness of this approach is limited by the need of partitioning a large system of equations in smaller subproblems. To overcome this difficulty, this paper presents a heuristic technique of 4 phases. It considers the relative performance of processors during the partitioning process and recommends partial overlapping of critical unknown when convenient for the resolution process.

In order to demonstrate the advantage of the proposed method using partial overlapping, this paper presents a mathematical analysis of small linear problems and experimental results using the proposed heuristic algorithm to partition systems of equations with different dimensions and characteristics.

Keywords: *Partition, Decomposition, Iterative Method, Partial overlapping, Asynchronous Implementations.*

1. INTRODUCTION

With the advent of high speed communication technologies, the aggregate CPU power in a LAN can easily exceed that of a supercomputer [1]. Therefore, parallel asynchronous implementations of iterative algorithms in heterogeneous computer environments are becoming a very convenient choice for solving large scale systems of equations, specially when considering advantages such as efficient exploitation of existing computing resources, cost effectiveness, shorter convergence time, and easier implementations [2]. Unfortunately, the usefulness of this approach is limited by the need of partitioning a large system of equations in smaller subproblems to be solved by individual processors of a distributed computing environment. This partitioning has sometimes two conflicting objectives: balanced loading among processors and good convergence of the resulting iterative implementation.
A good survey of the partitioning problem may be found in [3], beginning with the first studies by Carre in the 60's [4]. However, only during the 90´s this problem has gotten a lot of attention, being the ε-*Decomposition* [5] the best known technique, because of its simplicity by ignoring all data with values below a given ε during the partitioning process. However, the ε-*Decomposition* is not able to control the size of each subproblem; therefore, it has load balancing problems. To overcome this difficulty, Vale et al. [6] proposed a heuristic technique that assures load balancing for parallel (homogeneous) computers, that was later refined by Barán et al. [7,8] to assure a load balancing proportional to the relative processors performance *w* in a network of heterogeneous processors. Based on these previous works, the authors propose to further refine their technique [7,8], using partial overlapping [9].

This paper is organized as follows. Section 2 presents the Mathematical Background. Section 3 summarizes the concept of partial overlapping, while the Heuristic Algorithm is presented in Section 4. Experimental results are presented in Section 5 and the concluding remarks are left to Section 6.

2. MATHEMATICAL BACKGROUND

The idea behind the method is to transform a problem, with difficulties to be solved in parallel, in a new (possibly expanded) problem that can be efficiently solved using a heterogeneous distributed computer system. In this context, a system of *m* equations with *m* unknowns is given by:

$$\mathbf{F}(\mathbf{y}) = 0,\ \mathbf{F}:\Re^m \to \Re^m,\ \mathbf{y}=\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \in \Re^m,\ y_i \in \Re \qquad (1)$$

The objective of the partitioning method is to find a linear transformation **P,** in such a way that:

$$\mathbf{x} = \mathbf{P}\mathbf{y} \qquad (2)$$

the original problem (1) is transformed into a new system of *n* equations ($n \geq m$):

$$\mathbf{\Phi}(\mathbf{x}) = 0,\ \mathbf{\Phi}:\Re^n \to \Re^n,\ \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \Re^n \qquad (3)$$

that can be efficiently solved with *p* processors using any known parallel method. To do so, (3) should be partitioned according to [2]:

$$\mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} \mathbf{\Phi}_1(\mathbf{x}) \\ \vdots \\ \mathbf{\Phi}_p(\mathbf{x}) \end{bmatrix},\ \mathbf{\Phi}_i:\Re^n \to \Re^{n_i},\ n = \sum_{i=1}^{p} n_i \qquad (4)$$

with

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_p \end{bmatrix}, \qquad \text{where } \mathbf{x}_i \in \Re^{n_i},\ \forall i \in \{1,\dots,p\} \tag{5}$$

Thus, equation (3) may be rewritten as:

$$\mathbf{\Phi}_i(\mathbf{x}) = 0,\ \forall i \in \{1,\dots,p\} \tag{6}$$

that may be solved using an iterative method

$$\mathbf{x} \leftarrow \mathbf{G}(\mathbf{x}) \tag{7}$$

that can be chosen in such a way that:

$$\mathbf{G}(\mathbf{x}) = \begin{bmatrix} \mathbf{G}_i(\mathbf{x}) \\ \vdots \\ \mathbf{G}_p(\mathbf{x}) \end{bmatrix}, \quad \mathbf{G}_i(\mathbf{x}): \Re^n \to \Re^{n_i} \tag{8}$$

In this way, problem (6) can be solved in parallel, by assigning each subproblem to a different processor that updates $\mathbf{x}_i$ according to:

$$\mathbf{x}_i \leftarrow \mathbf{G}_i(\mathbf{x}) \tag{9}$$

That way, each processor i updates its local variable $\mathbf{x}_i$ using the best known value of $\mathbf{x}$, which was in part received from the other processors, and communicates its new value to the others (in a synchronous or asynchronous way [10]). The iterative process continues until the global solution is reached.

The synchronous implementation of (7) for processor i may be written as:

$$\mathbf{x}_i(k+1) \leftarrow \mathbf{G}_i(\mathbf{x}(k)) \tag{10}$$

where, processor i requires the information of the whole vector $\mathbf{x}$ calculated in the previous iteration to begin the next iteration; therefore, a *dead time* normally exists between iterations [2]. To overcome this *dead time* problem, the following asynchronous implementation may be used [2]:

$$\mathbf{x}_i(k+1) = \mathbf{G}_i(\mathbf{x}^i(k)), \quad \forall i \in \{1,\dots,p\} \tag{11}$$

where $\mathbf{x}^i(k)$ represents the value of $\mathbf{x}$, available in processor i at iteration k; i.e. processor i uses the most updated value of $\mathbf{x}$ it has, at the moment it begins a new iteration, avoiding synchronization time [10].

The implementation of (11) was studied in [2], where the following sufficient convergence condition was derived.

THEOREM 1 (Barán et al. [2]**):** *under assumptions of: uniform bound on delays, uniqueness of solution (in the given domain) and block-Lipschitz continuity of the operators, the asynchronous algorithm (11) converges to the solution if:*

$$\rho(\mathbf{H}) < 1 \tag{12}$$

where **H** *is the comparison matrix (given by the block-Lipschitz constants).*

+

As a consequence, the spectral radius of the comparison matrix $\rho(\mathbf{H})$ may be used to assure that a given algorithm converges to the solution; therefore, it will be used to select good partitions.

In short, the goal of a partitioning algorithm is to find **P** and to compute the dimensions n_i of each subproblem (4).

When considering the already published methods without *partial overlapping* [3-8], $\mathbf{P} = \{p_{ij}\}$ is a **permutation matrix** [11] with:

$$n = m, \qquad \sum_{i=1}^{n} p_{ij} = 1, \quad \text{and} \quad \sum_{j=1}^{n} p_{ij} = 1 \tag{13}$$

i.e., **P** is not more than an ordering of the unknowns.

<u>*EXAMPLE 1:*</u> decompose the following linear system to be solved using block-Jacobi's method in a distributed system with two identical processors (P_1 and P_2)

$$\begin{bmatrix} 2y_1 + 12y_2 \\ 0.08y_1 + y_2 + 0.1y_3 \\ 12y_2 + 3y_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{14}$$

Solution:

$$\mathbf{F}(\mathbf{y}) = \begin{bmatrix} 2y_1 + 12y_2 - b_1 \\ 0.08y_1 + y_2 + 0.1y_3 - b_2 \\ 12y_2 + 3y_3 - b_3 \end{bmatrix} \tag{15}$$

If the decomposition method proposes:

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \quad n_1 = 2, \text{ and } n_2 = 1 \tag{16}$$

equation (3) may be rewritten as:

$$\mathbf{\Phi}(x) = \begin{bmatrix} \mathbf{\Phi}_1(x) \\ \hdashline \mathbf{\Phi}_2(x) \end{bmatrix} = \begin{bmatrix} x_1 + 0.1x_2 + 0.08x_3 - b_2 \\ 12x_1 + 3x_2 + b_3 \\ \hdashline 12x_1 + 2x_3 + b_1 \end{bmatrix} \tag{17}$$

+

Note that the mapping between the variables x and y is biunique, i.e. $\mathbf{x} = \mathbf{P}\,\mathbf{y} \Rightarrow \mathbf{y} = \mathbf{P}^{-1}\mathbf{x}$. Thus, by solving (3), we solve (1).

An interesting problem arises when a simple reordering of variables does not allow a good partition because some special variables (known as *critical variables*) are required in two or more processors to assure convergence, due to their strong links to other variables distributed in different processors of the computing system. To solve this problem, Ikeda & Šiljak [9] proposed a *Partial Overlapping* method that replicates *critical variables* in two or more processors.

In example 1, the critical variable is y_2 because it is strongly coupled with the variables y_1 and y_3. Then, *partial overlapping* may be used, replicating the second equation (critical equation) in both processors. So, P_1 can solve equations 1 and 2, while P_2 solves equations 2 and 3. In this case, m=3, n=4 (n>m), and the suggested decomposition method would propose:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; n_1 = 2, \text{ and } n_2 = 2 \qquad (19)$$

Then, the expanded problem to be solve will be:

$$\mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} \mathbf{\Phi}_1(\mathbf{x}) \\ \hdashline \mathbf{\Phi}_2(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 12x_2 - b_1 \\ 0.08x_1 + x_2 + 0.1x_4 - b_2 \\ \hdashline 0.08x_1 + x_3 + 0.1x_4 - b_2 \\ 12x_3 + x_4 - b_3 \end{bmatrix} \qquad (20)$$

In this case:

$$\sum_{i=1}^{n} p_{ij} = N_j \geq 1, \quad \text{and} \quad \sum_{j=1}^{n} p_{ij} = 1 \qquad (21)$$

where N_j is the number of replications of variable y_j.

Note that again, given **P**, there is one biunique mapping between x and y, i.e. $\mathbf{x} = \mathbf{P}\,\mathbf{y} \Rightarrow \mathbf{y} = \mathbf{P}^{+}\mathbf{x}$, where $\mathbf{P}^{+} = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T$ is the *pseudo-inverse* of **P** [11].

Remark 1: the goal of the proposed partitioning technique, given a system of equations $\mathbf{F}(\mathbf{y}) = 0$, is to find a mapping **P**; so that, with $\mathbf{x} = \mathbf{P}\,\mathbf{y}$, we obtain a new system of equations $\mathbf{\Phi}(\mathbf{x}) = 0$ that can be efficiently solved with p possibly heterogeneous processors, using decomposition (4). To do so, our technique calculates the values n_i proportionally to the relative performance w of the p processors to be used in the resolution process.

3. PARTIAL OVERLAPPING

As mentioned, there are systems of equations that are very hard to partition in subproblems because of *critical equations* that are strongly related to many other equations. For those problems, it is very difficult to decide which processor should solve a given *critical equation* and sometimes, the best solution may be to solve it in several processors at the same time, as shown in the following examples.

Let return to example 1 of the section 2, where:

$$\mathbf{F}(\mathbf{y}) = \begin{bmatrix} 2y_1 + 12y_2 - b_1 \\ 0.08y_1 + y_2 + 0.1y_3 - b_2 \\ 12y_2 + 3y_3 - b_3 \end{bmatrix} \qquad (22)$$

can be represented by the following graph:

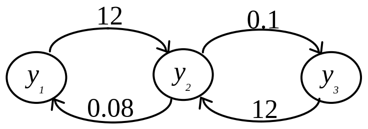


Figure 3.1: Graph of Example 1.

where the nodes represent the unknowns and the links represent the coupling value between variables (given by the coefficients of the problem).

The system can be partitioned without *Partial Overlapping* in three different ways with n_1= 2, n_2= 1 [12], that will be called Decompositions A, B and C respectively.

At the same time, the system can be partitioned using *Partial Overlapping*, with

$$n_1 = 2, \quad n_2 = 2 \quad \text{and} \quad \mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

That way, the *critical unknown* y_2 is replicated in both processor due to its strong links with y_1 and y_3. In other words, processor P_1 will solve equations 1 and 2, while processor P_2 will solve equations 2 and 3.

In this case, the new system of equation, called *expanded system* [7], has an expanded dimension n'=4, and can be represented by the following graph:

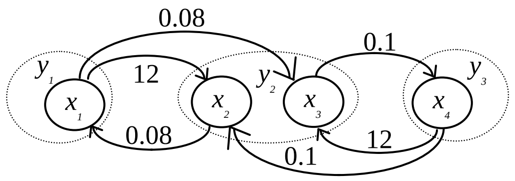


Figure 3.2: Graph for the expanded system of example 1.

Table 3.1 shows the experimental results obtained when solving example 1 using two identical processors. As can be seen, the block-Jacobi iterative algorithm converges only when *Partial Overlapping* is implemented.

	Decomposition			
	with *Partial Overlapping*			without *Partial Overlapping*
	A	B	C	
$\rho(\mathbf{H})$	2.148	1.789	1.039	0.7845
Iterations	Do not converge			29
Time	∞			1 s.

Table 3.1: Experimental results solving Example 1.

As a conclusion of example 1, it can be stated that thanks to the *Partial Overlapping* technique, it is possible to solve in parallel problems that otherwise would not be solvable with parallel implementations.

EXAMPLE 2: decompose the following linear system to be solved using block-Jacobi's method, in a distributed system with three identical processors (P_1, P_2 and P_3).

$$\mathbf{F}(\mathbf{y}) = \begin{bmatrix} 10y_1 + 2y_2 + 2y_3 + 10y_4 - b_1 \\ y_1 + 10y_2 + y_3 + 10y_4 - b_2 \\ 2y_1 + y_2 + 10y_3 + 10y_4 - b_3 \\ 5y_1 + 5y_2 + 5y_3 + 100y_4 - b_4 \end{bmatrix} \qquad (23)$$

Using the decomposition method without *Partial Overlapping*, several decompositions can be found, as an example:

n_1= 2, n_2= 1, n_3= 1 with a permutation matrix:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

So, the new system will be:

$$\mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} \mathbf{\Phi}_1(\mathbf{x}) \\ \mathbf{\Phi}_2(\mathbf{x}) \\ \mathbf{\Phi}_3(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 10x_1 + 10x_4 + 2x_2 + 2x_3 - b_1 \\ 5x_1 + 100x_4 + 5x_2 + 5x_3 - b_4 \\ \hdashline x_1 + 10x_4 + 10x_2 + x_3 - b_2 \\ \hdashline 2x_1 + 10x_4 + x_2 + 10x_3 - b_3 \end{bmatrix} \qquad (24)$$

The resulting spectral radius of the comparison matrix is $\rho(\mathbf{H}) = 0.4505 < 1$; thus, the sufficient convergence condition given by Theorem 1, is satisfied.

As another partitioning alternative, the method proposed in next section suggests a decomposition with *Partial Overlapping*, with the replication of the fourth equation (*critical equation*) in the three processors. In this case, there exists three versions of the same variable y_4 (x_2, x_4 and x_6); therefore, the total number of variables increases in the expanded system (from n=4 to n'=6), but the presence of the *critical equation* 4 justifies the use of *partial overlapping*, as shown below for the case with:

$$n_1 = 2,\ n_2 = 2,\ n_3 = 2 \ \text{and} \ \mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

Thus, the expanded system will be:

$$\mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} \mathbf{\Phi}_1(\mathbf{x}) \\ \mathbf{\Phi}_2(\mathbf{x}) \\ \mathbf{\Phi}_3(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 10x_1 + 10x_2 + 2x_3 + 2x_5 - b_1 \\ 5x_1 + 100x_2 + 5x_3 + 5x_5 - b_4 \\ \hdashline x_1 + 10x_3 + 10x_4 + x_5 - b_2 \\ 5x_1 + 5x_3 + 100x_4 + 5x_5 - b_4 \\ \hdashline 2x_1 + x_3 + 10x_5 + 10x_6 - b_3 \\ 5x_1 + 5x_3 + 5x_5 + 100x_6 - b_4 \end{bmatrix} \qquad (25)$$

The spectral radius of the comparison matrix, for this system, is $\rho(\mathbf{H}') = 0.2105 < 1$, which indicates that the sufficient convergence condition of Theorem 1 is satisfied.

By comparison of the spectral radius, the following relation can be written:

$$\rho(\mathbf{H}') = 0.2105 < \rho(\mathbf{H}) = 0.4505 < 1$$

threfore, it may be expected that the iterative algorithm (with *Partial Overlapping*) will converge faster.

In fact, when both iterative algorithms were implemented, we measured the experimental results shown in table 3.2, where it can be seen that the implementation with *Partial Overlapping* converges in fewer iterations and faster, despite its greater dimension. Note that, if the spectral radius is used as a *Figure of Merit* to select the best decomposition, according [7-8] recommendation, the decomposition with *Partial Overlapping* is correctly selected as the better one.

	Decomposition	
	with *Partial Overlapping*	without *Partial Overlapping*
Spectral radius	0.4505	0.2105
Iterations	14	9
Time	0.435 s	0.3976 s

Table 3.2: Results of solving the Example 2.

4. DECOMPOSITION METHOD

This section presents the proposed decomposition method, based on previous works of the authors [7-8], with the significant improvement of semi-automatic partitioning with *Partial Overlapping*.

Given a system of m equations with m unknown to be solved in a distributed system with p processors, the proposed method uses a matrix **M,** of dimension m x m, whose elements m_{ij} $(i \neq j)$ represents the degree of dependency (link value) between the variables y_j and y_j respectively.

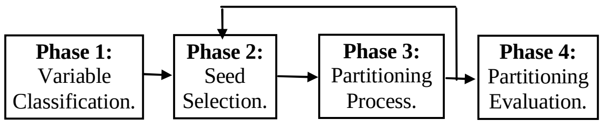
The variables y_i and y_j are not adjacent if $m_{ij} = m_{ji} = 0$; otherwise, y_i and y_j are adjacent. In case they are adjacent, they are called *weakly coupled* if m_{ij} and m_{ji} are small, and *strongly coupled* if m_{ij} and/or m_{ji} are large (with respect to other values of m_{kl}).

The main idea behind the method is to partition de main problem in subproblems that agglomerate together unknowns that are strongly coupled, while letting weakly coupled variables to be calculated in different processors. The size of each subproblem should be (as much as possible) in direct relation with the *Relative Performance* of the processor of the heterogeneous distributed computing system where it is to be solved.

Basically, the method can be understood as the formation of p

sub-systems, beginning with p initial variables, called *seeds*. The decomposition of the system is accomplished by assigning variables to the different partitions (or *seeds*), trying always to maintain the number of assigned variables proportional to the *relative performance* $\mathbf{w} \in \Re^p$ of the processors. As a result of the decomposition process, the method gives the **permutation matrix P** and the subproblem dimensions n_i (see Remark 1).

The method consists of four phases represented in figure 4.1.



Phase 1: Variable Classification.
Phase 2: Seed Selection.
Phase 3: Partitioning Process.
Phase 4: Partitioning Evaluation.

Figure 4.1: Phases of the proposed decomposition method.

<u>PHASE 1:</u> Variable Classification (Algorithm 4.1).
a ranking table is built based on a predefined weight proportional to the level of coupling among variables. The weight may be defined in several ways [6-8]. For the experimental results of next section, the following weights (called *Peso$_i$*), were used:

$$Peso_i = \sum_{j=1,\, j\neq i}^{m} (m'_{ij})^{z_{ij}} \ , \quad \text{with:}$$

$$m'_{ij} = \text{Sup}\{\, |m_{ij}| \ , \ |m_{ji}| \,\} \ ; \text{ and } \ z_{ij} = m'_{ij} \, / \, D \geq 0 \, ,$$

where D is the mean of all values of $m_{ij} \neq 0$.

Algorithm 4.1: Variable Classification.

```
Input:   matrix M

FROM $i$ = 1 TO $m$            /* for each of the $m$ variables. */
   Calculate the weight $Peso_i$ ;
   Include $Peso_i$ in the ordered ranking of variables;
Calculate the weight average $Pm$;
Calculate the weight standard deviation $De$;

Output: Weights $Peso_i$ of the $m$ variables ordered in a ranking, $Pm$ (weight average) and $De$ (weight standard deviation).
```

<u>PHASE 2:</u> Seed Selection (Algorithm 4.2).
Each of the p processors selects one variable as its own *seed*. Normally, a seed is a variable with a high weight that is not too close to other seeds, behaving as an agglomeration center of unknowns. The user has the choice of forcing one or more variables to be used as seeds. In general, several sets of different seeds may be obtained at the end of this phase.

Algorithm 4.2: Seeds Selection.

```
Input: matrix M, weights $Peso_i$ , number of processors $p$, and predefined parameters ($vlim$, $ngrup$, $nvec$).
$vlim$ is the minimum weight required to consider a variable as a candidate to be a seed;
$ngrup$ is the number of variables to be grouped around each seed candidate to check if it is a center of agglomerated unknowns; and
$nvec$ is a parameter used to avoid two strongly coupled variables being seed at the same time.

Initialize set $K$ as empty;
                              /* $K$, set of possible seed candidates */
FOR each variable $y_i$
  IF ($Peso_i \geq vlim$) THEN
    Include variable $y_i$ in set $K$ ;
FOR each variable $y_i$ in $K$
  Initialize the set $I_i$ as empty;
   /* $I$, set of variables grouped around each seed candidate */
  Include variable $y_i$ in $I_i$ ;
  Initialize the set $CIA_i$ as empty;
                         /* $CIA$, set of adjacent variables of set $I_i$ */
  Include in $CIA_i$ the adjacent variables of variable $y_i$ ;
  FROM 1 TO $ngrup$
        /* $ngrup$ variables are grouped around each candidate /
     Include variable with highest weight of $CIA_i$ into $I_i$ ;
     Eliminate this variable from $CIA_i$;
     Include in $CIA_i$ new adjacent variables of moved variable;
FOR each set $I_i$
  Calculate the weighted sum of all the variables in $I_i$ ;
Initialize the set $S$;
                        /* $S$, set of chosen seeds to begin a partition */
Select in $K$ variable $y_k$ with the largest sum of weights in $I_k$ ;
Include $y_k$ in $S$ as first seed ;
Eliminate $y_k$ from $K$ ;
WHILE (number of seeds < $p$)
  Select in $K$ variable $y_s$ with the largest sum of weights in $I_s$;
  IF ($y_s$ is not between the $nvec$ first variables from set $I$ of
  previous selected seeds) THEN
   Select $y_s$ as seed ;
   Include $y_s$ in $S$;
   Eliminate $y_s$ from $K$;
 ELSE
   Eliminate $y_s$ from $K$ ;

Output: For each set of parameters, there will be one set $S$ of $p$ seeds
```

<u>Phase 3:</u> Partitioning Process (Algorithm 4.3).
The decomposition of the system is accomplished by assigning variables to different partitions (or *seeds*) according to the relative performance $\boldsymbol{w}_i$ of each processor i. In this way, the load balance is maintained between desired levels. Good convergence properties are obtained when variables are assigned to partitions to which they have their strongest link. In case a variable is strongly coupled to several partitions, the

need of *Partial Overlapping* is analyzed and eventually recommended by the method. The user has the choice of selecting a partition with or without partial overlapping for *"critical variables"* Eventually, the user may choose both partitions; therefore, several partitions may be obtained.

Algorithm 4.3: Partitioning Process.

```
Input: set of seeds $S = \{s_1,...,s_p\}$, matrix $\mathbf{M}$, weights $Peso_i$
and $\alpha$, where $\alpha$ is the minimal difference between two links
to consider them as not equally strong.
--------------------------------------------------------------------------
Calculate LimOver;
Initialize the vectors $\mathbf{C} \in \mathbf{N}^p$ and $\mathbf{Q} \in \mathbf{N}^p$ ;
        /* C and Q are vectors used to control load balancing */
                /* $c_i$ is the size of sub-problem $i$ and $q_i = c_i/w_i$ */
FOR each seed $s_i \in S$
        /* J, set of variables assigned to a given subproblem */
  Initialize set $J_i$ as empty;
  Include in $J_i$ the seed $s_i$ ;
  Initialize set $CIA_i$ as empty;
                /* CIA, set of adjacent variables to set $I_i$ */
  Include in $CIA_i$ the adjacent variables to the seed $s_i$ ;
Update the vectors C and Q ;
WHILE there exist variables no grouped
  FOR all the sets $J_i$ that need to annex variables
    Select heaviest variable in CIA as candidate to be include;
    Control if there exist coincidences of candidates;
    IF there exist coincidences of candidates $\hat{y}_k$ THEN
      Sort the link values between variables of J and $\hat{y}_k$ ;
      IF (there is no overlapping option OR the difference
      between the greatest link value and the following one is
      not greater than $\alpha$ ) THEN
        Include the candidate variable $\hat{y}_k$ in the corresponding
        $J_i$ with the mayor coupling;
      ELSE there exists a coincidence
                        /* overlapping may be useful */
        IF weight of variable $\hat{y}_k \geq LimOver$ THEN
                        /* make overlapping */
          Form the set Coincidence Co with all the variables
          most strongly coupled to $\hat{y}_k$ ;
          Include $\hat{y}_k$ in all the coincident subsets J;
        ELSE
          Include $\hat{y}_k$ in the first J that fights for $\hat{y}_k$ ;
    ELSE
      Include $\hat{y}_k$ in the first J;
    Eliminate $\hat{y}_k$ from all the CIAs in J ;
    Include in the corresponding CIAs the adjacent variables to
    the recently included variable;
    Actualize C and Q;
$n_i = c_i$;
  /* $n_i$ is the dimension of subproblem assigned to processor $i$*/
--------------------------------------------------------------------------
Output: partition in $p$ subproblems (equivalent to matrix P)
and dimensions $n_i$ of each subsystem.
```

<u>Phase 4:</u> Partitioning Evaluation (Algorithm 4.3).
All the decompositions generated by the proposed method and other decompositions eventually recommended by the user, should be compared to chose the more promising one. For that task, several criterions may be used, but we found [7-8] that the best one is to compare the spectral radius of the comparison matrix [2]. That way, the different decompositions are ranked and the user can choose the best one(s).

Algorithm 4.4: Partitioning Evaluation.

```
Input: all the partitions generated by algorithm 4.3 or any
other partition introduced by the user.
--------------------------------------------------------------------------
FOR each partition
  Calculate $\rho(\mathbf{H})$ ;
  Include the value of $\rho(\mathbf{H})$ in a ranking;
Select as best partition the one with smaller $\rho(\mathbf{H})$ ;
Write explicitly permutation matrix P of the selected partition;
--------------------------------------------------------------------------
Output: ranking of partitions, with explicit values of P and $n_i$
for the recommended decomposition.
```

5. EXPERIMENTAL RESULTS

The advantages of solving large systems using heterogeneous distributed computing system are well established in the literature [1-3]. Experimental results using different partitioning methods (without overlapping), have already been presented [6-8]. Therefore, this section presents experimental results with partial overlapping using a distributed computing environment with three personal computers (p = 3) with similar performance ($\mathbf{w} = [1,1,1]^T$). Let consider the linear system of 13 equations and 13 unknowns $\mathbf{A}\,\mathbf{x} = \mathbf{b}$, with:

$$A = \begin{bmatrix}
10 & 2 & 2 & 10 & 1 & 0 & 0 & 0 & 0 & 0 & 3 & 3 & 3 \\
1 & 10 & 1 & 10 & 3 & 3 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 10 & 10 & 0 & 0 & 0 & 3 & 3 & 3 & 0 & 0 & 0 \\
5 & 5 & 5 & 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 3 & 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 3 & 0 & 0 & 0 & 10 & 3 & 0 & 0 & 0.5 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 & 0 & 3 & 10 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 & 0 & 0 & 10 & 3 & 0 & 0 & 0.5 & 0 \\
0 & 0 & 3 & 0 & 0 & 0 & 0 & 3 & 10 & 3 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 & 0.5 & 0 & 0 & 3 & 10 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 10 & 3 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 10
\end{bmatrix}$$

By using de decomposition method of section 4 with M = A, two different partitions are chosen [12]:

- ♦ one without overlapping, with $n_1 = 4$, $n_2 = 5$ and $n_3 = 4$;

♦ and another with partial overlapping that replicates x_4 in the three processors, and dimensions $n_1 = n_2 = n_3 = 5$.

Experimental results, solving the above system with block-Jacobi's method, are presented in Table 5.1. There, it can be seen that the proposed decomposition method finds a partition with overlapping that solves the problem in fewer iterations, and consequently, in less time than the one without overlapping. Note that the decomposition method presented in section 4 would choose in Phase 4 the partition with overlapping as the recommended one, because it has a smaller ρ(**H**).

Similar results are reported in [12] where there is an example in which no partition without overlapping solves a 13 X 13 linear system of equations, while the decomposition method of section 4 finds a partition with overlapping that can be solved in parallel without difficulties.

	Decomposition	
	without Overlapping	with Overlapping
Dimension	$m = 13$	$n = 15$
ρ(**H**)	0.8172	0.3780
Iterations	18	9
Time	0.6179 s	0.3676 s

Table 5.1 Experimental results solving the 13 X 13 system.

Another (sparse) linear system of 100 equations with 100 unknowns is presented in [12] and solved with the same computing resources, with the results presented in Table 5.2. Again, the proposed decomposition method is able to find a partition with partial overlapping that can be solved in fewer iterations than the one without partial overlapping. Once more, the solution with partial overlapping is faster.

	Decomposition	
	without Overlapping	with Overlapping
Dimension	$m = 100$	$n = 102$
Iterations	21	12
Time	0.78939 s	0.45108 s

Table 5.2 Experimental results solving a 100 X 100 system.

In short, the systems of equations presented above are useful to probe that partial overlapping may be used not only with small problems, as the one presented in section 3, but with larger ones. Moreover, the examples are interesting to show that the decomposition method presented in section 4 is able to recommend good partitions using partial overlapping. However, if partial overlapping is that useful,

why most professionals do not use it?.

To have a better insight of the way the decomposition method works and to try to find an answer of the above question, a number of randomly generated linear systems of equations of dimension 100, with different percentage C of zeros (C= 0%, 30%, 50%, 70%, and 90%, to experiment with sparse matrices), were solved with different number of processors (p = 2, 3, 4, 5, ...). Each 100X100 randomly generated problem was partitioned using the decomposition method of section 4, for different values of the parameters α and *LimOver* used by the method of section 4.

After studying 100 randomly generated problems for each set of parameters [C, p, α, *LimOver*], we got to the results and recommendations that follows.

♦ The number of overlapped unknowns increases with *LimOver*, as expected. It was found experimentally that a good *a-priori* choice may be:
$$LimOver = Pm + 2\,De;$$
that way, only highly coupled variables can be overlapped.

♦ The parameter α, used to decide if two links are equally strong, is not very important for the decomposition method. In fact, if it is reasonable in a wide range (1% to 10% of the m_{ij} values), the same decomposition is recommended by the method. As expected, the number of recommended overlapping may increase slightly with α.

♦ The percentage C of zeros does not change significantly the possibility of using partial overlapping. However, it was noted that the number of overlapped variables recommended by the decomposition method may decrease for extreme values of C (too large or too small).

♦ In general, there is a very small number of problems for which partial overlapping is recommended. According to our experiments, no more than 3% of the problems can benefit from using partial overlapping. This result seems the main reason why people do not bother to use partial overlapping, unless it is too obvious to use a decomposition technique.

♦ It was noted that the number of overlapped variables increases with the number of processors; therefore, partial overlapping may become more important when we try to solve very large problems with a large number of processors.

In conclusion, partial overlapping has been rarely used in practical problems because it is very difficult to find *a-priori* good partitions with partial overlapping and an exhausted method to find a good partition is out of possibilities becuase the space of possible partitions is combinatorial. To make it worth for practical applications, only a very small percentage of problems can benefit from the partial overlapping technique when solved in parallel. In consequence, the partial overlapping technique was not thoroughly studied even though it is known for more than a decade [9]. This situation may now change with the proposed decomposition technique that automatically recognizes situations for which the use of partial overlapping technique may be very useful and recommends good partitions that can benefit from this technique.

6. CONCLUSIONS

In view of the advent of parallel and distributed computer systems, as the existing networks, several decomposition methods have been published with the idea of solving a large problem using the existing computing resources. However, most of the published techniques do not have the ability of controlling the size of each subproblem in such a way that load balancing between heterogeneous processors can be accomplished.

To overcome this problem, the authors proposed an heuristic technique, consisting of 4 sequential phases, that has the ability of decomposing a problem in a given number of subproblems, with a load balance proportional to the relative performance of the processors to be used in the resolution. In each phase of the proposed method, an expert user can interact with the method by suggesting different parameters, the interest in considering partial overlapping, or even partitions that may look good *a-priri*. Phase 4 of the method makes a ranking of the better partitions, based on a parameters that can assure convergence of an iterative implementation, even in an asynchronous environment.

A very unique feature of the proposed method is its ability to recommend partial overlapping in situations where it may be very useful. In fact, in section 3 it was presented examples where a problem can be better solved in parallel, when partial overlapping technique is used. Other examples with larger systems of equations (13X13 and 100X100) where presented in section 5, showing that partial overlapping may be vary useful when we have the ability of finding when and how to used it.

To understand why most professionals do not use partial overlapping, a number of randomly generated systems of equations were studied, concluding that only in a small amount of problems (less than 3%) can benefit from this technique. Therefore, it was not worth finding good partitions with partial overlapping, specially because there was no published method to do it automatically.

This problem may be now overcame with the presented method that is able to recommend good partitions with partial overlapping, that may have the additional benefit of using *Asynchronous Team Algorithms* [2] to solve each version of a critical variable with a different algorithm.

As a consequence, the interest in partial overlapping may increase and more professionals may benefit from this technique when solving large problems using heterogeneous distributed computing facilities, as the existing computer networks which increase their aggregate CPU power with an impressive speed.

7. REFERENCES

[1] Hiu Ch. and Chanson S., "Allocating Task Interaction Graphs to Processors in Heterogeneous Networks". *IEEE Transactions on Parallel & Distributed Systems.* Vol. 8, No. 9 , pp. 908-925, Sep. 1997.

[2] Barán B., Kaszkurewicz E. and Bhaya A., "Parallel Asynchronous Team Algorithms: Convergence and Performance Analysis". *IEEE Transactions on Parallel & Distributed Systems.* Vol. 7, No. 7 , pp. 677-688, Jul. 1996.

[3] Vale M.H., "*Descomposição de Redes Eléctricas para Procesamiento Paralelo*." Doctoral Dissertation COPPE/UFRJ. Rio de Janeiro, Brazil, 1995.

[4] Carré B.A., "Solution of Load-Flow by Partitioning Systems into Trees", *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-88, pp. 1931-1938, Nov. 1968.

[5] Sezer M. and Šiljak D.D., "Nested epsilon decomposition of complex systems". IFAC 9th *World Congress*, Budapest, Hungary. Jul. 1984.

[6] Vale M.H., Falcão D.M. and Kaszkurewicz E., "Electrical Power Network Decomposition for Parallel Computations". *IEEE International Symposium on Circuits and Systems*, (ISCAS 92). San Diego, California, 1992.

[7] Barán B., Benítez D. and Ramos R., "Partición de Sistemas de Ecuaciones para su Resolución Distribuida", *XXII Latino-American Conference - CLEI 96*, Bogota - Colombia. Jun. 1996.

[8] Barán B., Benítez D. and Ramos R., "Partición de Sistemas Eléctricos en Subsistemas Menores para su Resolución Distribuida", *VII Encuentro Regional Latinoamericano de la CIGRÉ - VII ERLAC*, Puerto Yguazú – Argentina, May 1997.

[9] Ikeda M. and Šiljak D.D., "Overlapping decomposition, expansions and contractions of dynamic systems". *Large Scale System 1*, North-Holland Publishing Co., pp.29-38, 1980.

[10] Bertsekas D.P. y Tsitsiklis J.N., *Parallel and Distributed Computation. Numerical Methods*, Editorial Prentice-Hall, 1989.

[11] Strang G., *Linear Algebra and its Applications*, Second Edition, Academic Press Inc. Florida, pp. 77-80.

[12] Benítez D. and Barán B. "*Descomposición Solapada de Sistemas de Ecuaciones para su Resolución en Sistemas Distribuidos*," Technical Report RT-001/98. Asuncion, Paraguay, 1998.