# Team algorithms in distributed load flow computations

B. Barán
E. Kaszkurewicz
D.M. Falcão

*Indexing terms: Parallel processing, System decomposition*

**Abstract:** The paper reports the use of a general technique to combine several different methods to solve complex systems of algebraic equations in the context of load flow calculations of electrical power networks. Such combinations of methods, referred to as 'team algorithms', seem specially well suited to be used with distributed memory computer systems, in an asynchronous environment. Experimental results solving example problems in a commercially available parallel computer system show that a 'synergetic effect' with considerable speedup can be obtained using these 'team algorithms'.

## 1 Introduction

Parallel and distributed processing applications in power system problems have received a great attention in recent years [1]. Computation speedup, favourable cost-to-performance ratio, scalability etc., are some of the advantages offered by this type of data processing scheme. To fully exploit the potential of parallel and distributed systems, there is a need to develop or adapt algorithms specially tailored for this new computer environment.

Parallel asynchronous implementations of numerical iterative algorithms are establishing themselves as good choices for high performance computation in distributed memory environments, in view of several attractive features that they possess, such as ease of implementation, facility with load balancing, shorter convergence times and so on [2]. A new possibility in this context was opened with the concept of 'team algorithms' which are hybrids of different algorithms.

In power system applications, such combinations of algorithms were first proposed in [3] for sequential computers, and later again in [4–6], where they were first called 'team algorithms' (TA) in the context of parallel computing. In such methods these algorithms have a natural implementation [7].

The main idea behind a TA is to partition a complex problem in several subproblems and to solve each subproblem in a different processor of a distributed computer system, choosing for each subproblem one or more methods that best solve it. That way, if there are subproblems with different characteristics for which it have been chosen different methods, the resulting combination of methods constitutes a TA.

In this paper, the TA concept is illustrated through the solution of the load flow problem in a distributed computer environment. The objectives are twofold: (a) to demonstrate experimentally that TAs can exhibit adequate performance in cases where individual members fail, and (b) to show the efficiency of this approach in distributed memory parallel computers.

To demonstrate the advantages in using a TA, two examples of load flow calculations are presented. For these examples, neither the fast decoupled method nor the block Jacobi version of the $Y$ matrix method can properly solve the problems; however, a TA combining both methods solves the problems with the additional advantage of being easily parallelised in an asynchronous environment and with an excellent speedup. In that way, it is possible to use all the potential of distributed memory computers, as the Intel iPSC/860 hypercube of 8 nodes used for the implementations described herein.

## 2 Team algorithms

In this Section a system of $n$ nonlinear algebraic equations given by

$$\Phi(x) = 0 \quad \text{for} \quad x \in \mathbf{R}^n \tag{1}$$

is considered.

The basic idea of a distributed approach for the solution of eqn. 1 is to use a system of $p$ processors in such a way that each processor *solves* only part of the whole system and communicates its partial results to the other processors to finally solve the *global* problem.

Assuming eqn. 1 is partitioned as

$$\Phi(x) = \left[\Phi_1^T(x) \;\; \Phi_2^T(x) \;\; \ldots \;\; \Phi_m^T(x)\right]^T \tag{2}$$

$$x = \left[x_1^T \;\; x_2^T \;\; \ldots \;\; x_m^T\right]^T \quad \text{for} \quad x_i \in \mathbf{R}^{n_i} \tag{3}$$

it may be rewritten as a set of equations

$$\Phi_i(x) = 0 \quad \text{for} \quad i = 1, 2, \ldots, m \tag{4}$$

Therefore, each subproblem, $\Phi_i(x) = 0$, may be solved by an iterative algorithm represented by the maps $G_i$ that update $x_i$, that is,

$$x_i(k+1) = G_i(x(k)) \quad \text{for} \quad i = 1, 2, \ldots, m \quad \text{and} \quad k = 0, 1, \ldots \tag{5}$$



*IEE Proceedings* online no. 19952249

Paper first received 18th January 1995 and in revised form 21st July 1995

The authors are with COPPE/ the Center for Parallel Computing, Federal University of Rio de Janeiro, PO Box 68516, 21945-970 Rio de Janeiro RJ, Brazil

The iterative algorithm is chosen such that the fixed point $x^* = G(x^*)$ of the map:

$$G(x) = \left[G_1^T(x) \;\; G_2^T(x) \;\; \ldots \;\; G_m^T(x)\right]^T \qquad (6)$$

is a solution of eqn. 1, that is, $\Phi(x^*) = 0$.

The expression for $G_i(x)$ depends on the specific algorithm chosen to solve the corresponding subproblem eqn. 4. In general, no one method can solve all subproblems equally well and it may be useful to choose, for each subproblem, the one that is best suited for it. When two or more different methods are chosen, this hybrid of methods is called team algorithm (TA).

The parallel synchronous version of a block team algorithm can be understood as the implementation of the recurrence eqn. 5 where each algorithm $G_i$ is assigned to a different processor, and where at the time instants $t_1$, $t_2$, $t_3$, etc., the exchange of data between all the processors, in order to update the current estimates of $x_i$, is performed *simultaneously*. This scheme, in general, introduces synchronisation delays in the process since not all processors conclude their computation at the same time, and these delays deteriorate the performance of the parallel implementation [2]. To circumvent this, asynchronous implementations are used.

## 2.1 Asynchronous block team algorithms

Each processor $i$ used in an asynchronous implementation of a *block TA* attempts to solve its local subproblem eqn. 4 using a map $G_i$ that updates $x_i$ and transmits the updated value to the other processors without synchronisation delays, blocking or interruptions, that occur in the synchronous implementations.

In this case, the general parallel block asynchronous iterative method based on eqn. 5 may be written as

$$x_i(k+1) = G_i[x^i(k)] \quad \text{for} \quad i = 1, 2, \ldots, m \qquad (7)$$

Eqn. 7 represents an *asynchronous block TA:* each processor $i$ tries to solve its local subproblem eqn. 4 by updating $x_i$, that is, applying map $G_i$ and using the most recently received values of $x_j$ for all $j \neq i$ that compose the vector $x^i$, that is, $x^i(k)$ is the most recently received version available to processor $i$ at the time instant $k$. This is also called the ULD (use latest data) scheme [4]. After obtaining the new value of $x_i$, processor $i$ communicates its results to those processors that need it, without synchronisation delays, blocking or interruptions of the latter, and initiates a new iteration. The process continues until a given tolerance $\varepsilon$ is satisfied.

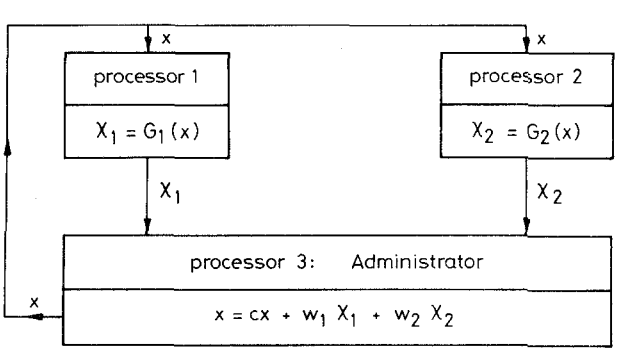


**Fig. 1** *Fully overlapped team algorithm*

## 2.2 Overlapped Team Algorithms

The use of several methods in parallel, with the scope of combining the best properties of each one, was firstly introduced in [4], where it was proposed to solve the algebraic system (eqn. 1), using two (or more) iterative methods which solve the same (global) problem in parallel, combining the partial results in a process called the 'Administrator', as shown in Fig. 1. This paper considers an asynchronous implementation of the TA shown in Fig. 1 implemented in a distributed memory computing system, which is mathematically represented, in the case of two subsystems, as

$$\begin{aligned} \chi_1(k+1) &= \mathbf{G}_1[x^1(k)] \\ \chi_2(k+1) &= \mathbf{G}_2[x^2(k)] \\ x(k+1) &= cx(k) + w_1\chi_1^3(k) + w_2\chi_2^3(k) \end{aligned} \qquad (8)$$

where $x^i(k)$ represents the most recently received value of vector $x$ in processor $i$ at iteration $k$, and $\chi_i^3(k)$, $(i = 1, 2)$, represents the most recent value of $\chi_i$ at iteration $k$, available to processor 3 (the Administrator).

The TA given by eqn. 8 works as follows: each processor $i$ $(i = 1, 2)$ tries to solve problem eqn. 4 using an iterative algorithm represented by the map $\mathbf{G}_i$, using the most recent value of $x$ received from the Administrator, and transmitting the updated value $\chi_i$. At the same time, the Administrator updates $x$ combining the most recently received values of $\chi_i$, using properly chosen nonnegative weights $(w_1, w_2 \geq 0)$. The constant $c$ is chosen to ensure that, for a solution $x^*$, the following condition holds: $x = \chi_1 = \chi_2 = x^* \in \mathbf{R}^n$, which, in turn, implies that $c = 1 - w_1 - w_2$.

The choice of the weights $w_1$, $w_2$ in the computations reported in this paper was done heuristically. A satisfactory initial guess is such that $w_1 + w_2 \approx 1$ and where $w_1 \approx w_2 \approx 0.5$; some guidelines for a good choice can be used on the basis of the convergence criterion given in [8]; the idea of an 'Intelligent Administrator' suggested in [4] could be also implemented in order to adequately adjust the weights during the computations.

## 2.3 Generalised team algorithm

The use of fully overlapped TAs makes possible the solution of problems that no one of the individual algorithms is able to solve independently; usually, however, the fully overlapped TA does not present a significant speedup.

To enhance the TAs performance in a distributed computer system, this paper introduces the concept of *generalised* TA (GTA).

In fact, the GTAs include as special cases the block TAs, the fully overlapped TAs, as well as other cases corresponding to algorithms associated with 'overlapping decompositions' [10–12].

A GTA is basically a block TA with partial overlapping; that is, one given subproblem can be assigned to different processors running different algorithms. The partial results of those different processors are combined by the Administrator using the same weighted sum described for the fully overlapped TA.

To illustrate the above, consider the following example: a TA that combines two different algorithms $(p = 2)$ that are represented by their corresponding maps $G_1$ and $G_2$. This TA is used to solve a problem partitioned into three blocks $(m = 3)$: $x_1$ is updated only by $G_1$; $x_2$ is updated only by $G_2$, and $x_3$ is updated by both $G_1$ and $G_2$.

Considering a parallel asynchronous environment, a subvector $x_i$ updated by more than one processor has several different versions available, depending on the processor (or map) considered. In this context, the following notation is used:

$\chi_{ij}$ denotes the version of subvector $x_i$ calculated using map $G_j$;

$x_i$ denotes the subvector calculated in the Administrator from the $\chi_{ij}$s.

$G_{ij}$ denotes operator $G_j$ updating subvector $x_i$.

As an example, a GTA is illustrated in Fig. 2. Note that the Administrator calculates $x_3$ using a weighted sum of the different versions of subvector $x_3$, that may be represented as

$$x_3(k+1) = cx_3(k) + w_1\chi_{31}^3(k) + w_2\chi_{32}^3(k) \quad (9)$$

It should be noted that $x_3$, $\chi_{31}$, $\chi_{32} \in \mathbf{R}^{n_3}$; are different 'versions' of the same subvector.

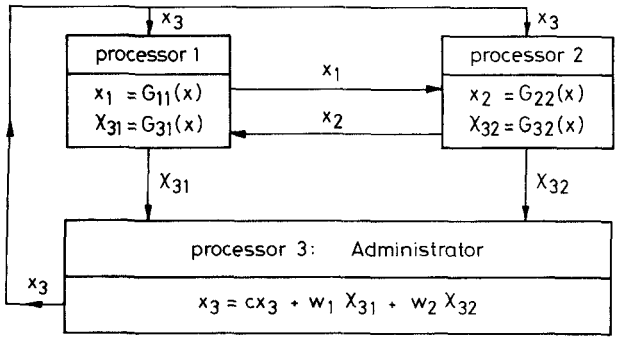


**Fig.2** *Version A of GTA*

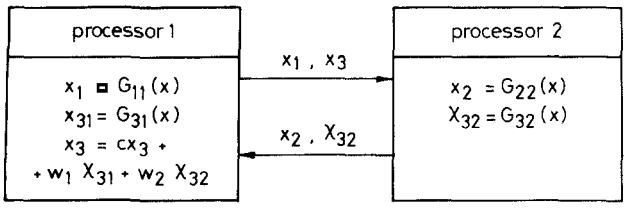


**Fig.3** *Version B of GTA*

The GTA illustrated in Fig. 2, called version A, has an Administrator implemented in a dedicated processor. However, an Administrator only calculates a weighted sum of vectors which is very little task when compared with solving the subproblems. The resulting inefficiency in using a dedicated processor as an Administrator may be avoided by assigning the Administrator's task to any other processor. The version B of a TA, implementing this idea, is illustrated in Fig. 3. The Administrator's task of versions A and B is done by only one processor; however, it may be done by several processors. As an example, Fig. 4 illustrates the version C of a GTA, with a replicated Administrator. Other versions of GTA's and their convergence analysis are discussed in [8]. Of special interest are the ones with *implicit* Administrators. That is, the weighted sum of the different versions of a subvector is done as part of the algorithm to be applied and not as special processes (or additional equations) as illustrated in Fig. 5.

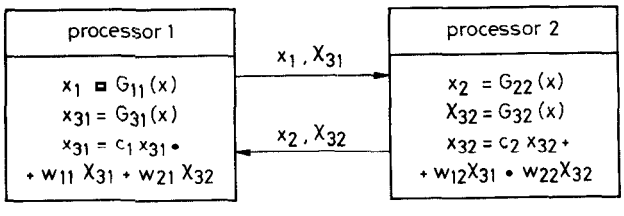


**Fig.4** *Version C of GTA*

## 3 Application to load flow problem

The Load Flow problem can be formulated as an almost linear equation of the type

$$I(x) = Yx \quad (10)$$

where $Y$ is the nodal admittance matrix, $x$ is the nodal voltage vector, and $I(x)$ is the vector of nodal injected currents. The voltage at node $k$ can be expressed as $x_k = V_k e^{j\theta_k}$. In practical applications, constraints are imposed in the solution vector like, for instance, the specification of the voltage magnitude at certain buses (PV buses).

Considering the interest of this paper in block asynchronous methods, the load flow problem eqn. 10 may be rewritten in the form of eqn. 4, where

$$\Phi_i(x) = \sum_{j=1}^{m} Y_{ij}x_j - I_i(x_i) \quad \text{for} \quad i = 1, 2, \ldots, m \quad (11)$$

To solve eqn. 11 using a distributed parallel computer system, this paper considers a block Jacobi (BJ) version of the $Y$-matrix load flow formulation, a fast decoupled (FD) method [19], and several TAs combining both methods. The FD method was considered due to its recognised efficiency as a load flow solution method and the BJ method was chosen because of its ease of implementation in parallel. Replacing the FD method, the Newton method might be used as well.

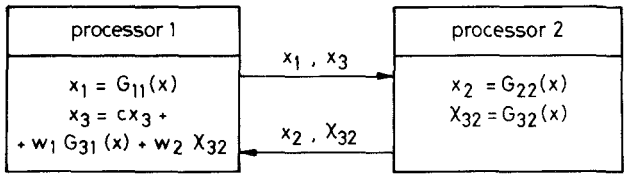


**Fig.5** *Version E of GTA*

### 3.1 Block Jacobi method

The $Y$-matrix based BJ method is simple to use with low memory requirements; however, it does not present a good convergence for electrical systems with PV buses. In fact, in the next Section two examples in which the method does not converge are presented. A discussion on several variations of the method that try to overcome the difficulty when PV buses are present may be found in [18].

The method may not be very practical for most applications; however, it is easily parallelised and it may work well if no PV bus is present in the system (experimental results are presented in [9,13]); thus, it may be useful when combined with other methods. Considering only systems without PV buses, the block Jacobi (BJ) version of the $Y$-matrix method in a parallel asynchronous environment may be represented by the map:

$$G_{BJi}(x) = Y_{ii}^{-1}I_i(x_i) - \sum_{j=1, j\neq i}^{m} Y_{ii}^{-1}Y_{ij}x_j \quad (12)$$

Sufficient conditions for the convergence of the asynchronous BJ algorithm can be found in [9].

### 3.2 Fast decoupled method

In each iteration of the FD method, the following linearised subsets of equations, defining increments in the active and reactive power injections, have to be solved:

$$\Delta P/V = B'\Delta\theta \quad (13)$$

$$\Delta Q/V = B''\Delta V \quad (14)$$

where the elements of $B'$ and $B'$ are functions of the network parameters.

Although the FD method performs well in a sequential environment, for most problems it is not easy to parallelise and presents convergence problems in special cases where the system has transmission lines with a high $R/X$ (resistance/reactance) ratio or with neighbouring nodes with highly different phases (discussed in [15]). Modifications introduced in the classical formulation of the FD method [16,17] have made its convergence behaviour less affected by the above network characteristics.

The convergence of the FD method in a sequential environment is demonstrated in [15] where it is represented by the iteration:

$$\xi(k+1) = \phi\big(\xi(k)\big) \tag{15}$$

$$\xi = \begin{bmatrix}\theta_1 & \dots & \theta_n & V_1 & \dots & V_q\end{bmatrix}^T \in \mathbf{R}^{n+q} \tag{16}$$

The FD method can be considered as a contraction iteration method (CI) in a sequential environment [15]. Considering the one to one relation between $x_k = V_k e^{j\theta_k} \in C$ and $\xi_k = [V_k, \theta_k]^T \in \mathbf{R}^2$, the FD method may be seen not only as an operator $\phi$ working on $\xi$, but also as an operator $G_{FD}$ working on $x$, that is,

$$x(k+1) = G_{FD}\big(x(k)\big) \tag{17}$$

To solve the load flow problem given by eqns. 13 and 14 in a parallel asynchronous environment, it is possible to choose the FD method for each subproblem $\Phi_i(x) = 0$. That way, a block asynchronous FD method can be constructed:

$$x_i(k+1) = G_{FDi}\big(x^i(k)\big) \quad \text{for} \quad i = 1, \dots, m \tag{18}$$

### 3.3 Team algorithms in load flow computations

To illustrate the potential of TAs, we examine problems that can not be solved conveniently using neither the fast decoupled, nor the block Jacobi methods alone. However, they are easily solved using a TA that combines the advantages of the BJ and FD methods. The idea, based on [3], is to use the fast decoupled method when there are PV buses in a block, while the block Jacobi is used when no PV bus is present in a block with high $R/X$ ratio or with highly different neighbouring node phase angles. Thus, the idea is to exploit all the potential of the FD method, avoiding its problems in the specific cases where it fails to have nice convergence properties.

## 4 Computational results

The experimental results were obtained in computations performed in an eight processor Intel iPSC/860 parallel computer. A considerable number of different cases was tested for different teams, implementations, partitions, initial conditions, tolerances, etc. The most representative data are displayed in Tables 1 and 2. They essentially confirm the experience reported in [4] that the team, in general, is much robust and faster than its individual members.

### 4.1 Test systems

Both test systems were built up with data extracted from the IEEE 118 standard test system and an industrial electrical system of an off-shore oil production platform. Some network element parameters were modified to obtain high resistance to reactance ratios.

*Example 1:* An electrical system of 616 buses and 945 branches consisting of eight different interconnected subsystems with the following characteristics. All PV buses belong to subsystem 1; subsystems 2 – 8 have several branches with high $R/X$ ratio and neighbouring buses with highly different phases.

Choosing for each subsystem the method best suited for its solution, the following combination of algorithms was used:

Subsystem 1: FD method $(G_1 = G_{FD1})$

Subsystems 2-8: BJ method $(G_i = G_{BJi},\ i = 2,8)$

Since the eight subsystems are not strongly interconnected, the following asynchronous block TA (without overlapping) was used:

$$Processor\ 1:\ x_1(k+1) = G_{FD1}\big(x^1(k)\big) \tag{19}$$

$$Processor\ i = 2, \dots, 8:\ x_i(k+1) = G_{BJi}\big(x^i(k)\big) \tag{20}$$

The next Section presents experimental results showing that the TA given by eqns. 19 and 20 conveniently solves example 1 for different cases that the FD and BJ methods do not solve by themselves.

**Table 1: Results for example 1 using block team algorithms (ε = 0.001 and x(0) = flat start)**

| Method | $p$ | *iter* | sec | $S_r$ | $\eta_r$ | $S_p$ | $\eta$ | $S_a$ |
|---|---|---|---|---|---|---|---|---|
| Jacobi | 1 | ∞ | ∞ | – | – | – | – | – |
| FD | 1 | 131 | 9.92 | 1 | 100 | 0.12 | 12 | – |
| Seq. TA | 1 | 33 | 1.16 | 8.52 | 852 | 1 | 100 | – |
| Synchronous | 2 | 30 | 1.02 | 9.8 | 488 | 1.15 | 57 | 1 |
| TA | 4 | 30 | 0.81 | 12.3 | 307 | 1.4 | 36 | 1 |
| | 8 | 31 | 0.50 | 19.9 | 249 | 2.34 | 29 | 1 |
| Asynchronous | 2 | 65 | 0.65 | 15.3 | 766 | 1.8 | 90 | 1.56 |
| TA | 4 | 43 | 0.48 | 20.5 | 512 | 2.4 | 60 | 1.67 |
| | 8 | 32 | 0.38 | 25.9 | 324 | 3.1 | 38 | 1.30 |

However, harder problems that can not be easily solved by a block TA sometimes arise. In such cases the option of using a GTA with properly chosen partial overlapping, as illustrated in the following example.

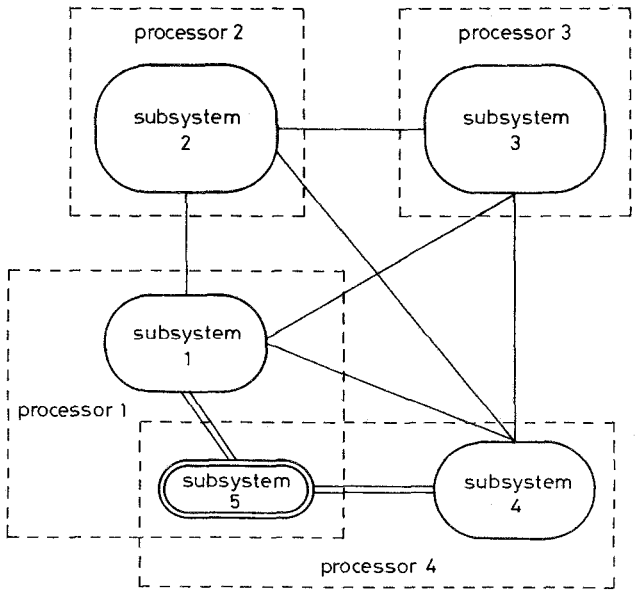


**Fig. 6** *Decomposition structure of electrical system of example 2*

**Table 2: Results for example 2 using team algorithms with partial overlapping**

| Method | Overlapping | $p$ | $\varepsilon = 0.001$ and $x(0)$ = flat start | | | | $\varepsilon = 0.001$ and $x(0) \approx x^*$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | *iter* | seconds | $S_p$ | $\eta$ | *iter* | seconds | $S_p$ | $\eta$ |
| Jacobi | | | – | | – | – | – | no convergence | – | – |
| FD | no | 1 | – | no convergence | – | – | 5 | 1.052 to1.053 | 1 | 100 |
| Sequential TA | | | – | | – | – | – | no convergence | – | – |
| Sequential TA | yes | 1 | 53 | 2.33 to 2.33 | 1 | 100 | 31 | 1.497 to 1.498 | 0.7 | 70 |
| Synchronous | no | 2 | – | no | – | – | – | – | no | – |
| TA | | 4 | – | convergence | – | – | – | – | convergence | – |
| Asynchronous | no | 2 | – | occasional | – | – | – | occasional | – | – |
| TA | | 4 | – | convergence | – | – | – | convergence | – | – |
| Synchronous | yes | 2 | 64 | 1.825 to1.827 | 1.3 | 64 | 20 | 0.880 to 0.882 | 1.2 | 60 |
| TA | | 4 | 84 | 1.324 to 1.326 | 1.8 | 44 | 28 | 0.637 to 0.640 | 1.7 | 41 |
| Asynchronous | version A | | 138 to 184 | 1.736 to 2.308 | 1.3 | 45 | 61 to 85 | 0.853 to 1.107 | 1.2 | 61 |
| TA with | version D | 2 | 146 to 189 | 1.571 to 2.033 | 1.5 | 74 | 78 to 103 | 0.794 to 1.032 | 1.3 | 66 |
| overlapping | version E | | 172 to 194 | 1.767 to 2.103 | 1.3 | 66 | 69 to 92 | 0.769 to 0.972 | 1.4 | 68 |
| Asynchronous | version A | | 62 to 93 | 1.111 to 1.650 | 2.1 | 42 | 28 to 44 | 0.505 to 0.768 | 2.1 | 42 |
| TA with | version D | 4 | 83 to 410 | 0.971 to 1.265 | 2.4 | 60 | 42 to 59 | 0.497 to 0.692 | 2.1 | 53 |
| overlapping | version E | | 68 to 791 | 0.791 to 0.897 | 3 | 74 | 36 to 68 | 0.430 to 0.808 | 2.5 | 61 |

*Example 2:* An electrical system of 616 buses and 945 branches illustrated in Fig. 6, consists of five different interconnected subsystems with the following characteristics: all PV buses belong to subsystem 1; subsystems 2 – 4 have several branches with a high $R/X$ ratio and neighbour buses with highly different phases; subsystem 5 is the smallest (10 buses) and it is strongly interconnected with subsystems 1 and 4. The option of using one processor for each block is not attractive for this example because it implies cutting several *strong* branches (large admittances) and may conduce to an iterative process with poor convergence characteristics [11,12,14]. The solution, illustrated in Fig. 6, is to use partial overlapping to avoid cutting strong branches.

Note the use of $p = 4$ processors to solve the $m = 5$ subsystems, because $x_5$ is updated by processors 1 and 4 which also update their corresponding subvectors ($x_1$ and $x_4$). Subsystems 1 – 4 could be further decomposed in other subsystems to map the problem onto a larger number of processors without affecting the convergence process. To assign the algorithm to each processor we follow the same reasoning of example 1. The resulting asynchronous TA may be implemented using several different versions. As an example, TA version E may be represented as

*Processor 1:*

$$\begin{bmatrix} x_1(k+1) \\ x_5(k+1) \end{bmatrix} = \begin{bmatrix} G_{FD11}(\hat{x}^1(k)) \\ G_{FD51}(\hat{x}^1(k)) \end{bmatrix} \tag{21a}$$

$$\hat{x}^1(k) = \left[x_1^{1T}(k), \ldots, x_4^{1T}(k), (w_1 x_5^1(k) + w_2 \chi_{54}^1(k))^T\right]^T \tag{21b}$$

$$\text{Processor } i = 2,3: \; x_i(k+1) = G_{BJi}(x^i(k)) \tag{22}$$

*Processor 4:*

$$\begin{bmatrix} x_4(k+1) \\ \chi_{54}(k+1) \end{bmatrix} = \begin{bmatrix} G_{BJ44}(x^4(k)) \\ G_{BJ54}(x^4(k)) \end{bmatrix} \tag{23a}$$

$$x^i(k) = \left[x_1^{iT}(k), \ldots, x_5^{iT}(k)\right]^T \tag{23b}$$

Experimental results presented in the following Section show that, for example 2, methods as the FD, BJ and several block TAs, fail to find a solution, while the GTA (eqn. 21*a* to eqn. 23*b*) properly solves the problem, with a considerable speedup.

## 4.2 Efficiency and speedup

The standard definition of 'speedup' of an algorithm [2] is

$$S_p = \frac{\text{best sequential time of any algorithm}}{\text{parallel time of the algorithm considered}}$$

Also is of interest, the efficiency, $\eta$, is defined in [2] as

$$\eta = 100\, S_p/p$$

where $p$ represents the number of processors utilised, and $\eta$ is represented as a percentage.

To measure the advantage of using asynchronous implementations instead of synchronous ones, the concept of 'speedup due to asynchronism' is introduced:

$$S_a = \frac{\text{parallel synchronous time}}{\text{parallel asynchronous time}}$$

A figure of merit convenient for measuring the advantage of using a parallel TA, the 'relative speedup of a TA', is defined as

$$S_r = \frac{\text{best sequential time of the TA members working alone}}{\text{parallel time of the TA}}$$

The corresponding efficiency measure $\eta_r$ is defined as

$$\eta_r = 100\, S_r/p$$

with $\eta$ represented as a percentage. Finally, a figure of merit that measures the gain in using a TA with respect of any of the member algorithms, in a sequential environment, is defined as

$$S_s = \frac{\text{best sequential time of the TA members working alone}}{\text{time using a sequential TA}}$$

## 4.3 Results using block team algorithms

This Section presents experimental results for the solution of example 1 using BJ and FD methods, as well as several different implementations of block TAs (without overlapping) using 1, 2, 4 and 8 processors. The main conclusions of the computational experiments are summarised below:

(i) A block TA is clearly better than any of the member algorithms working alone. Moreover, if the convergence tolerance is decreased ($\varepsilon \leq 0.0005$), only the TA was able to solve example 1.

(ii) An asynchronous TA is significantly better than its synchronous counterpart, as illustrated in Table 1.

(iii) The 'best sequential time' is obtained using a sequential TA, while the 'best parallel time' is obtained by an asynchronous implementation. The following relation holds: $S_r = S_s \times S_p \times S_a$; that is, the large value of $S_r$ (relative speedup of a TA) obtained in our implementations (see Table 1) was possible as a combined effect of three factors: the use of a TA ($S_s$), the use of parallelism ($S_p$ referred to the synchronous version of the algorithm), and the use of asynchronism ($S_a$).

(iv) The value of the sequential speedup ($S_s$) for example 1 is 8.52 (see $S_r$ for the sequential TA), which is a considerable contribution for the performance of the parallel TA.

(v) $S_p$ increases with the number $p$ of processors ($p \leq 8$), while $\eta$ decreases.

(vi) The advantage in $S_p$ of using an asynchronous TA increases when a smaller tolerance $\varepsilon$ is imposed or an initial point $x(0)$ closer to the solution is used (in fact, for a *bad* initial point, the asynchronous version may not converge even when the sequential TA converges).

### 4.4 Results using team algorithms with partial overlapping

In this Section the use of block TAs, as well as several GTAs, with partial overlapping in example 2 are reported. Typical results are illustrated in Table 2, where the range of computation time and number of iterations for successive executions of the same problem, with the same data, are shown. The main conclusions already mentioned for example 1 are valid for example 2. In addition:

(i) For example 2, the figure of merit $S_s$ has to be considered in two cases. For the case of a 'flat start', $S_s$ is infinity, since none of the sequential members of the TA converges (see Table 2); and for the case where $x(0) \approx x^*$, the situation is such that $S_s = 0.7$, that is, the best sequential time is of the FD method. In other words, the contribution of the team arrangement is not effective; therefore, the contribution to the final performance is due to parallelism and asynchronism.

(ii) For this case, methods without overlapping do not converge. An exception is the asynchronous TA that occasionally converges depending on the specific asynchronism used. Therefore, the use of overlapping is the only way to ensure the solution of example 2.

(iii) The advantage of using asynchronism is present in almost all cases and is even greater for smaller tolerance requirements.

### 4.5 Application to actual power systems

Although representing hypothetical situations, the examples used correspond to load flow studies that may have to be performed in practice. For instance, in voltage stability analysis it may be necessary to represent the medium and low voltage areas of the power network (subtransmission and distribution). In this case, one of the subnetwork partitions might correspond to the high voltage network in which several PV buses are represented, whereas the other partitions correspond to medium and low voltage networks in which there are few or none PV buses and high $R/X$ ratio branches may be encountered.

The application of TA methods, introduced in this paper, to actual power systems requires the system decomposition in a suitable number of subsystems. Decomposition methods, like those reported in [10,14] (see also references therein), are essential in this context.

## 5 Conclusions

This paper has presented an application of the concept of team algorithms to the solution of the load flow problem in a distributed computer environment. The results of computational experiments have shown that the proposed approach makes possible the solution of problems unsolvable by conventional algorithms. Also, considerable speedup was achieved in the parallel implementations of these algorithms. In some cases, superlinear speedups were obtained. Further research in this field may produce methods able to deal with important practical problems, like the load flow solution close to the voltage collapse point, with high computational efficiency.

## 6 References


1 TYLAVSKY,D.J., and BOSE,A.: 'Parallel processing in power systems computation', *IEEE Trans. Power Syst.*, 1992, **7**, pp. 629–637

2 BERTSEKAS,D.P., and TSITSIKLIS,J.N.: 'Parallel and distributed computation' (Prentice Hall, New York, 1989)

3 DUSONCHET,Y.P., TALUKDAR,S.N., SINNOT,H.E., and EL-ABIAD A.H.: 'Load flows using a combination of point Jacobi and Newton's method', *IEEE Trans. Power Appar. & Syst.*, 1971, **90**, pp. 941–949

4 TALUKDAR,S.N., PYO,S.S., and MEHROTA,R.: 'Design algorithms and assignment for distributed processing'. EPRI Report EL-3317, 1983

5 MEHROTA,R., and TALUKDAR,S.N.: 'Task scheduling on multiprocessors for power system problems', *IEEE Trans. Power Appar. & Syst.*, 1983, **102**, pp. 3590–3597

6 TALUKDAR,S.N., PYO,S.S., and GIRAS,T.C.: 'Asynchronous procedures for parallel processing', *IEEE Trans. Power Appar. & Syst.*,1983, **102**, pp. 3652–3659

7 RAMESH,V.C., QUADREL,R., DE SOUZA,P.S., and TALUKDAR,S.N.: 'Asynchronous teams', 1991 Summer National Meeting of the American Institute of Chemical Engineers, Pittsburgh, Aug. 1991, pp. 18–21

8 BARÁN,B., KASZKUREWICZ,E., and BHAYA,A.: 'Parallel asynchronous team algorithms: convergence and performance analysis'. COPPE/UFRJ Internal Report BKB94/EE/01, 1994

9 BHAYA,A., KASZKUREWICZ,E., and MOTA,F.: 'Asynchronous block-iterative methods for almost linear equations', *Linear Algebra & Appl.*, 1991, **154–156**, pp. 487–508

10 IKEDA,M., and ŠILJAK,D.D.: 'Overlapping decomposition, expansions and conractions of dynamic systems', Large Scale Sytem 1 (North-Holland Publishing Co., 1980, pp. 29–38)

11 CALVET,J.L., and TITLI,A.: 'Overlapping vs. Partitioning in block-iteration methods: application in large-scale system', *Automatica*, 1989, **25**, (1), pp. 137–145

12 ZEĆEVIĆ,A.I., and ŠILJAK,D.D.: 'A block-parallel Newton method via overlapping epsilon decompositions', Proceedings of the 1992 American Control Conference, 1992, **155**, pp. 487–508

13 BHAYA,A., FALCÃO,D.M., KASZKUREWICZ,E., and ROQUEIRO,N.: 'Parallel block-iterative methods: application to the load flow problem', Proceedings of the IFAC symposium on *Large scale systems: theory and applications*, Berlim, RDA, Aug. 1989

14 VALE,M.H.M., FALCÃO,D.M., and KASZKUREWICZ,E.: 'Electrical power network decomposition for parallel computations', IEEE symposium on *Circuits and systems*, San Diego, CA, May 1992, ,pp. 2761–2764

15 WU,F.F.: 'Theoretical study of the convergence of the fast decoupled load flow', *IEEE Trans. Power Appar. & Syst.*, 1977, **96**, pp. 268–275

16 VAN AMERONGEN,R.A.M.: 'A general purpose version of the fast decoupled loadflow', *IEEE Trans. Power Syst.*, 1989, **4**, pp. 760–770

17 MONTICELLI,A., GARCIA,A., and SAAVEDRA,O.R.: 'Fast decoupled load flow: hypothesis, derivations, and testing', *IEEE Trans. Power Syst.*, 1990, **5**, pp. 1425–1431

18 STOTT,B.: 'Review of load flow calculation methods', *Proc. IEEE*, 1974, **62**, pp. 916–929