Component Solution Method in a Parallel Asynchronous Environment

Benjamín Barán and Eugenius Kaszkurewicz
Parallel Computing Centre
COPPE, Federal University of Rio de Janeiro
Fax: 55 21 290-6626
P.O.Box 68516, Rio de Janeiro, RJ 21945970 - Brazil

Abstract

The Component Solution method (CS) has a natural implementation in a parallel computing environment and it has been already studied in a synchronous computing environment. Attractive advantages arise when implementing the CS method in an asynchronous environment, as studied in this work. Also, a sufficient condition for the convergence of the CS method in that asynchronous environment is derived.

The main advantage of the asynchronous CS method is its ability for solving problems in fewer iterations than any other asynchronous block iterative method satisfying some standard conditions, as shown experimentally and proved in the main theorem of this work.

About the Authors

Benjamín Barán received an Engineering degree in electronic at the National University of Asuncion (Paraguay) in 1982, a M.Sc. in Computer Engineering at Northeastern University (USA) in 1987 and a D.Sc. degree in System and Computer Engineering at the Federal University of Rio de Janeiro (Brazil) in 1993. He is an Associated Professor at the National University of Asuncion working at the Computing National Center (CNC) and lecturing at the Catholic University of Paraguay. His work was awarded by the Paraguayan Scientific Society. His research interest includes parallel and distributed processing, numerical methods, computer architecture and performance as well as team algorithms. Email: bbaran@una.py.

Eugenius Kaszkurewicz received an Engineering degree in aeronautic at the Aeronautic Technology Institute (ITA) in 1970, a M.Sc. (1972) and a D.Sc. (1981) in System and Computer Engineering at the Federal University of Rio de Janeiro (Brazil), where he works as a Full Professor. He did post-doctoral studies at the University of Santa Clara - California, from 1985 to 1987. He was Chairman of the Parallel Computing Centre and the Electrical Department at the Federal University of Rio de Janeiro. His research interest includes parallel processing, numerical methods, system control and stability as well as team algorithms. Email: eugenius@coep.ufrj.br.
1 Introduction

A Distributed System consists of several loosely coupled processors, each one with its own local memory module and the capability of interconnecting to the other processors of the system by sending/receiving messages through a communication facility (as a network). Well known examples are the LANs and several parallel computers such as the hypercubes.

The number of commercially available distributed systems has grown in an impressive way during the last decade and most of the latest computers in the market have some kind of distributed facilities. The main two reasons for this substantial development of distributed systems are:

- a very low cost per instruction, when compared to traditional uniprocessor systems (as mainframes);
- an excellent modularity that permits to increase power according to necessity, without a large initial investment.

Since the first Von-Newman computer model, the computer field has been developed mainly based on a sequential model; however, the availability of distributed systems in most organizations brings the necessity of re-thinking several procedures and the opportunity to develop new parallel techniques, more adapted for a distributed environment. A good example of this is the Component Solution (CS) method discussed in this work.

In fact, the CS method is an iterative algorithm useful when solving large systems of algebraic equations in a distributed system, as described in Section 2 for a synchronous environment. Section 3 describes a sequential implementation of the CS method, while the analysis of the method in an asynchronous environment is presented in Section 4, including the derivation of a sufficient condition for the convergence of the method. Section 5 discusses the main advantages of the CS method in an asynchronous environment, first discussing experimental results and later formally proving a theorem. The final conclusions are left for Section 6.

2 Parallel Synchronous Environment

The Component Solution method is analyzed in [5], considering a parallel synchronous computer environment, as an approach for solving the following algebraic system of equations:

\[ \Phi(x) = 0, \quad x \in \mathbb{R}^n, \quad \Phi : D \rightarrow \mathbb{R}^n, \]

where \( D \) is a closed subset of \( \mathbb{R}^n \).

The main idea is to use a distributed system of \( m \) processors in such a way that each processor solves only a part of the whole system and communicates its partial result to the other processors to finally solve the 'global' problem (1). To formalize this, the following notation and assumptions are needed.
Let a Cartesian decomposition of $\mathbb{R}^n$ be given by:

$$\mathbb{R}^n = \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_m}, \quad n_1 + \cdots + n_m = n. \quad (2)$$

Let $m := \{1, 2, \ldots, m\}$ and $D \subset \mathbb{R}^n$ be a domain such that

$$D = \hat{D}_1 \times \cdots \times \hat{D}_m, \quad \hat{D}_i \subset \mathbb{R}^{n_i}, \quad \forall i \in m. \quad (3)$$

A vector $x \in D$ is conformally partitioned as:

$$x = [x_1^T, x_2^T, \ldots, x_m^T]^T, \quad x_i \in \hat{D}_i, \quad \forall i \in m, \quad (4)$$

and $\Phi(x)$ as:

$$\Phi(x) = [\Phi_1^T(x), \Phi_2^T(x), \ldots, \Phi_m^T(x)]^T, \quad \Phi_i : D \to \mathbb{R}^{n_i}. \quad (5)$$

Equation (1) may then be rewritten as:

$$\forall i \in m, \quad \Phi_i(x) = 0. \quad (6)$$

Remark 1 The partition of a large problem $\Phi(x) = 0$ in $m$ smaller (and probably easier) subproblems $\Phi_i(x) = 0$ may be accomplished using any of the partition techniques already published, as in [1,9,10,12,13,14]. For this paper, we assume the partition is given.

The CS method is an iterative method that tries to solve equation (1) by solving at every iteration $k$, each subproblem $\Phi_i(x) = 0$. The $m$ subproblems may be solved in parallel using $m$ different processors of the distributed system; therefore, each processor $i$ solves the subproblem $\Phi_i(x) = 0$ updating the subvector $x_i$ at each iteration, after which, a global communication is required before beginning the following iteration. Mathematically, the CS method described above is represented by:

$$\forall i \in m, \quad x_i(k+1) = G_i(x(k)). \quad (7)$$

where the maps $G_i$ represent how the subvector $x_i$ is updated depending on the value of $x(k)$.

Denoting as $\bar{x}_i(k)$ the local solution of the subproblem $\Phi_i(x) = 0$ at iteration $k$ in processor $i$, i.e.

$$\Phi_i(x_1(k), \ldots, \bar{x}_i(k), \ldots, x_m(k)) = 0 \quad (8)$$

what the CS method does at iteration $k$ may be represented as:

$$x_i(k+1) = \bar{x}_i(k) \quad (9)$$

In general, equation (7) represents a synchronous block iterative algorithm [2], where the maps $G_i$ are chosen such that the fixed point $x^* = G(x^*)$ of the map

$$\forall i \in m, \quad G(x) = [G_1(x)^T, \ldots, G_m(x)^T]^T, \quad G_i(x) : D \to D_i. \quad (10)$$
is a solution of (1), i.e. $\Phi(x^*) = 0$. Of course, the CS method may be considered as a block iterative method where $G_i$ is chosen to satisfy (8), i.e.

$$x_i(k+1) = \bar{x}_i(k) = G_i(x_1(k), \ldots, \bar{x}_i(k), \ldots, x_m(k))$$ (11)

Throughout this work, it is assumed that a single general norm $\| \cdot \|$ is used on all spaces $\mathbb{R}^{n_i}, i \in m$ and on $\mathbb{R}^n$ as well. At the same time, the following assumption is made:

**ASSUMPTION 1 (LOCAL UNIQUENESS OF FIXED POINT)**

*Given the system of equations (1) defined in a closed set $D \subset \mathbb{R}^n$, there exists a map $G$ satisfying equations (7) and (10), such that $G(D) \subset D$ and such that $D$ contains exactly one fixed point

$$x^* = \begin{bmatrix} x_1^* \\ \vdots \\ x_m^* \end{bmatrix}$$ \quad \forall i \in m

of the map $G$, which is a solution of (1). Therefore, we can write:

$$x_i^* = G_i(x^*) \text{ and } \Phi_i(x^*) = 0, \quad \forall i \in m$$ (12)

The convergence of the CS method in a synchronous environment is analyzed in [5] assuming $G$ is a block-contraction mapping, i.e.

**DEFINITION 1 (BLOCK-CONTRACTION MAPPING)**

*The map $G$ satisfying (10) is a block-contraction mapping in $D$ if

$$\forall x, y \in D, \forall i \in m, \quad \|G_i(x) - G_i(y)\| \leq \alpha \|x - y\|$$

where $x$ and $y$ satisfy (4) and $0 \leq \alpha < 1$.

Thus, a slight variation of the following theorem is derived in [5] giving a sufficient condition for the convergence of the CS method:

**THEOREM 1 (CONVERGENCE OF THE CS METHOD FOR BLOCK-CONTRACTION)**

*If the map $G$ that represents a CS method is a block-contraction mapping in $D$, the method converges geometrically to the unique solution $x^*$ in $D$.

### 3 Sequential Environment

The Component Solution method described above is inherently parallel with each processor solving another subproblem; however, it may be easily implemented in a sequential environment by doing the work of each processor, one at a time, using only one processor (what is
known as block-Jacobi algorithm). The convergence condition given by Theorem 1 remains valid.

Alternatively, an improvement in the convergence rate may be obtained in a sequential environment using the Gauss-Seidel algorithm to implement the CS method, i.e.

\[ x_i(k + 1) = G_i(x_1(k + 1), \ldots, x_{i-1}(k + 1), x_i(k), \ldots, x_m(k)) \]  

(13)

A sufficient condition for the convergence of the above algorithm, similar to the one given by theorem 1, is derived in [5].

The description of the CS method in a sequential environment was simply done for completeness, because the main advantage of the method is the easiness of implementing in a distributed computing environment. However, for real applications, it is not easy to have a well balanced load and the result when using a synchronous implementation is that several processors are waiting for communication while others are still solving their local problems. Therefore, using an asynchronous parallel implementation brings a better performance, as discussed in the following section.

4 Parallel Asynchronous Environment

Parallel Asynchronous implementations of iterative algorithms are now 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 which the load can be balanced, shorter convergence times and so on [2,3,4,5].

This section considers an asynchronous implementation of the CS method, in which each processor \( i \) of a distributed system solves in \( x_i \) its local subproblem \( \Phi_i(x) = 0 \) using the most recently updated value of \( x \). After finding a new value for \( x_i \), processor \( i \) transmits \( x_i \) to the other processors of the computing system without blocking or interruptions; i.e. the value of \( x \) used when applying \( G_i \) according to (7), in processor \( i \), may have delayed or ‘old’ components. To be more specific, it will be assumed that, at the time of iteration \( k \), processor \( i \) (which updates \( x_i \)) receives information from another processor \( j \) with a time-varying delay of \( k - d_{ij}^k(k) \) units. The important assumption that the delays are uniformly bounded in time (over all processors) by a positive integer \( d \) will be made. This is also referred to as partial asynchronism [5] and stated formally as a restriction on the range of the positive integer-valued functions \( d_{ij}^k(\cdot) \) in Assumption 2 below.

**Assumption 2 (Uniform Bound on Delays)**

\[ \exists d \in \mathbb{N}, \forall k \in \mathbb{N}, \forall i, j \in m, \text{ such that } d_{ij}^k(k) \in \{k, k - 1, \ldots, k - d\}. \]  

(14)

The version of the vector \( x \), available in processor \( i \) at iteration \( k \), is denoted as \( x^i(k) \) and given by:

\[ x^i(k) := \begin{bmatrix} x_1(d_i^i(k)) \\ \vdots \\ x_m(d_m^i(k)) \end{bmatrix}, \quad \forall i \in m \]  

(15)
Using this notation, the general bounded-delay block asynchronous iterative method based on (7) may be written as:

\[ x_i(k + 1) = G_i\left(x_i(k)\right), \quad \forall i \in m \]  

(16)

Equation (16) represents the following implementation of a general asynchronous block algorithm: each processor \( i \) tries to solve its 'local' subproblem (6) by updating \( x_i \), i.e. applying map \( G_i \) and using the most recently received values of \( x_j \) for all \( j \neq i \). After obtaining the new value of \( x_i \), processor \( i \) communicates its results to those processors that need it, without blocking or interruptions of the latter, and initiates a new iteration. The process continues until a given tolerance \( \epsilon \) is satisfied.

For the specific case of the asynchronous block CS method described above, the value of \( x_i(k + 1) \) is a solution of the local subproblem at iteration \( k \) according to (9), i.e.

\[ \Phi_i(\bar{x}_i(k)) = 0, \quad \forall i \in m \]  

(17)

where \( \bar{x}_i(k) \) represents the value of \( x_i(k) \) after updating \( x_i \) with the local solution \( \bar{x}_i(k) \), i.e.

\[ \bar{x}_i(k) := \begin{bmatrix} x_1(d_1^i(k)) \\ \vdots \\ x_i(k) \\ \vdots \\ x_m(d_m^i(k)) \end{bmatrix}, \quad \forall i \in m \]  

(18)

thus, by Assumption 1

\[ \bar{x}_i(k) = G_i(\bar{x}_i(k)), \quad \forall i \in m \]  

(19)

With the objective of deriving a sufficient convergence condition for the asynchronous CS method, similar to the one given by Theorem 1, the block-Lipschitz continuity concept is introduced.

**Definition 2 (Block-Lipschitz Continuity)**

A function \( G(x) \) satisfying (10) is called block-Lipschitz continuous in a given norm with respect to a Cartesian decomposition \( D_1 \times \cdots \times D_m \) defined in (3) if \( \forall x, y \in D \) satisfying (4) we can write:

\[ \|G(x) - G(y)\| \leq \sum_{i=1}^{m} L_i \|x_i - y_i\| \]  

(20)

The concept of block-Lipschitz continuity is equivalent to Lipschitz continuity, but often more convenient, as stated in the following lemma (see [2] for an immediate proof):

**Lemma 1**

A function \( G(x) \) is block-Lipschitz continuous iff it is Lipschitz continuous.
ASSUMPTION 3 (CONTRACTIVE BLOCK-LIPSCHITZ CONTINUITY)
Each map $G_i(x)$ of equation (16) is block-Lipschitz continuous, i.e.

$$
\forall i \in m, \quad \forall x, y \in D, \quad \|G_i(x) - G_i(y)\| \leq \sum_{j=1}^{m} l_{ij} \|x_j - y_j\| \tag{21}
$$

with $0 \leq l_{ii} < 1$.

To begin the derivation of a sufficient condition for the convergence of the CS method in an asynchronous environment, an error vector $e = [e_1^T, \ldots, e_m^T]^T \in \mathbb{R}^n$ is defined as:

$$
\forall i, j \in m,
\begin{cases}
e_i(k+1) := x_i(k+1) - x_i^* \\
e_j(d_j^i(k)) := x_j(d_j^i(k)) - x_j^*
\end{cases}
$$

and a reduced error vector $z = [z_1, \ldots, z_m]^T \in \mathbb{R}^m$ as:

$$
\forall i, j \in m,
\begin{cases}
z_i(k+1) := \|e_i(k+1)\| \\
z_j(d_j^i(k)) := \|e_i(d_j^i(k))\|
\end{cases}
$$

(23)

Subtracting (12) from (19) and using (9),

$$
\|x_i(k+1) - x_i^*\| = \|G_i(\bar{x}^i(k)) - G_i(x^*)\|, \quad \forall i \in m
$$

and by Assumption 3:

$$
\|x_i(k+1) - x_i^*\| \leq l_{ii} \|x_i(k+1) - x_i^*\| + \sum_{j=1, j \neq i}^{m} l_{ij} \|x_j(d_j^i(k)) - x_j^*\|, \quad \forall i \in m
$$

using the reduced error notation (23):

$$
z_i(k+1) \leq l_{ii} z_i(k+1) + \sum_{j=1, j \neq i}^{m} l_{ij} z_j(d_j^i(k)), \quad \forall i \in m
$$

now, subtracting $l_{ii} z_i(k+1)$ and dividing by $(1 - l_{ii}) > 0$ we get, as in [6], the asynchronous difference inequality,

$$
z_i(k+1) \leq \sum_{j=1, j \neq i}^{m} \left( \frac{l_{ij}}{1 - l_{ii}} \right) z_j(d_j^i(k)), \quad \forall i \in m
$$

(24)

The convergence of the reduced error vector $z$ to zero as $k \to \infty$, follows from the Asynchronous Comparison Lemma proved in [2] and [4] following a previous work of [6]:

LEMMA 2 (ASYNCHRONOUS COMPARISON LEMMA)
Under assumptions 1 and 2, given a non-negative matrix $H := (h_{ij}) \geq 0$ and a non-negative time-varying vector $z \geq 0$, satisfying the inequation

$$
z_i(k+1) \leq \sum_{j=1}^{m} h_{ij} z_j(d_j^i(k)) \quad \forall i \in m,
$$

(25)
then \( \rho(H) < 1 \) is a sufficient condition for \( z(k) \) to tend to zero exponentially as \( k \to \infty \), i.e.,

\[
\lim_{k \to \infty} z(k) = 0 \quad \text{if} \quad \rho(H) < 1
\]

where \( \rho(H) \) denotes the spectral radius of matrix \( H \).

**Definition 3 (Comparison Matrix)**

The matrix \( H \) associated with the inequality (25) is referred to as a Comparison Matrix.

The following theorem is an immediate consequence of applying Lemma 2 to equation (24), as in [2]:

**Theorem 2 (Convergence of the Asynchronous Block CS Method)**

Under assumptions 1 to 3, the asynchronous block CS method represented by (16) converges to the unique fixed point \( x^* \) in \( D \) if

\[
\rho(H_{cs}) < 1
\]

where \( H_{cs} \) is given by:

\[
H_{cs} := (h_{ij}), \quad \text{where} \quad h_{ij} = \begin{cases} 0 & \text{if } j = i \\ \frac{l_{ij}}{1 - l_{ii}} & \text{if } j \neq i \end{cases}
\]  

\[
(26)
\]

**Remark 2** For a nonnegative matrix \( H \), there are several conditions equivalent to \( \rho(H) < 1 \), some of which are easier to check, as proved in [8].

## 5 Comparison of Asynchronous Block Iterative Methods

Advantages of using asynchronous block iterative algorithms implemented in a distributed computing system, specially when considering a combination of different algorithms (known as Team Algorithms) were already discussed in [2,3,4]. Some of those advantages are: low cost per instruction, modularity, easiness of implementation, no large penalty with unbalanced partitions, capability of solving problems that are not easily solved with standard sequential algorithms and so on.

Assuming the advantages of using asynchronous block iterative methods are well understood, this section emphasizes the advantage in convergence of using the CS method instead of other block asynchronous iterative methods. To begin, a small example is considered:

**Example 1**

Using an asynchronous block iterative algorithm with 2 processors \( (m = 2) \), solve the following system of equations

\[
\begin{align*}
x_1^2 + 2.1 \cdot x_1 + 0.9 \cdot x_2 &= 0 \\
x_2^2 + 0.9 \cdot x_1 + 800 \cdot x_2 &= 0
\end{align*}
\]  

\[
(27)
\]
in the domain $D = \{x \in \mathbb{R}^2 | \|x\|_\infty \leq 1 \}$.

Different methods may be used to solve problem (27). Table 1 presents experimental results considering several asynchronous block iterative methods as the Component Solution (CS), block-Jacobi (BJ), Simplified Newton (SN), Newton-Raphson (NR) as well as a Team Algorithm combining the CS and BJ methods. Simulations were done by randomly chosen 100 initial conditions in $D$.

Table 1 gives the number of problems correctly solved by each method (out of 100), as well as the number of times those solutions were found with a number of iterations less or equal the number of iterations required by the other methods considered in the table (optimum solution). The methods are ordered in Table 1 according to the spectral radius $\rho(H)$ of the corresponding Comparison Matrix (see Definition 3). The value of $\rho(H)$ is specially useful when using the Asynchronous Comparison Lemma (see Lemma 2) as a sufficient condition for the convergence of asynchronous block iterative methods [2,3].

The first thing to notice in Table 1 is that in fact, all methods with $\rho(H) < 1$ solve the problems for every initial condition (as stated in Lemma 2) while the others may or may not solve the problem depending on the initial conditions. A second interesting property shown in the table is a correlation between the number of optimum solutions and the spectral radius when $\rho(H) < 1$, i.e. the smaller the spectral radius of $H$, the greater the number of optimal solutions. This experimental correlation was found in most of the problems experimentally implemented and may be understood considering the Comparison Matrix $H$ as a kind of upper bound on the real iteration matrix (see [8]). In fact, it has been already established an equivalent relation between the convergence rate of a linear iterative algorithm and the spectral radius of its iteration matrix [15]. Consequently, the value of $\rho(H)$ may be used as a Figure of Merit for apriori comparison of the convergence rate of asynchronous block iterative methods.

**Remark 3** The fact that a method 1 has a figure of merit smaller than another method 2 does not mean that it will always converge in fewer iterations. What it really means is that method 1 has a larger probability of converging in fewer iterations when the initial condition is randomly chosen in the given domain $D$. 

![Table 1: Experimental Results of Example 1](image)
The final and most important property shown in Table 1 is that the CS method is the one with the smallest \( \rho(H) \) and the largest number of solutions with the minimum number of iterations. In fact, the CS method has the smallest figure of merit \( \rho(H) \) in every implemented problem. This interesting property is easily understood considering the following theorem:

**Theorem 3** If there exists an asynchronous block iterative method represented by an operator \( G \) with comparison matrix \( H \) that satisfies the sufficient conditions for convergence given by Lemma 2, i.e., \( \rho(H) < 1 \), then the comparison matrix \( H_{cs} \) of the Component Solution method satisfies the relation:

\[
\rho(H_{cs}) \leq \rho(H) < 1
\]

**Proof.**

Considering the comparison matrix \( H \) given, as in [3], by

\[
H := (h_{ij}), \quad \text{where} \quad h_{ij} = l_{ij}, \quad (28)
\]

then, by (26), the comparison matrix of the CS method is given by:

\[
H_{cs} := (h_{ij}), \quad \text{where} \quad h_{ij} = \begin{cases} 
0 & \text{if } j = i \\
\frac{l_{ij}}{1-l_{ii}} & \text{if } j \neq i
\end{cases} \quad (29)
\]

The proof is based in the following relation given in [11] for a non-negative matrix \( H \):

\[
\rho(H) = \inf_{U \in \mathcal{P}} \| U^{-1} H U \|_{\infty} \quad (30)
\]

where \( \mathcal{P} \) is the set of all diagonals positive matrices. Clearly, for any diagonal positive matrix \( U \in \mathcal{P} \), we have [15]:

\[
\rho(H) \leq \| U^{-1} H U \|_{\infty} \quad (31)
\]

Now, by hypotheses and relation (30)

\[
\frac{1}{u_i} \sum_{j=1}^{m} u_j l_{ij} < 1 \quad \forall i \in m
\]

and multiplying by \( u_i l_{ii} \geq 0 \),

\[
l_{ii} \sum_{j=1}^{m} u_j l_{ij} \leq u_i l_{ii}
\]

i.e.,

\[
0 \leq u_i l_{ii} (1 - l_{ii}) - l_{ii} \sum_{j \neq i} u_j l_{ij}
\]

adding \( \sum_{j \neq i} u_j l_{ij} \)

\[
\sum_{j \neq i} u_j l_{ij} \leq u_i l_{ii} (1 - l_{ii}) + (1 - l_{ii}) \sum_{j \neq i} u_j l_{ij}
\]
thus,
\[ \sum_{j \neq i} u_j l_{ij} \leq (1 - l_{ii}) \sum_{j=1}^{m} u_j l_{ij} \]

remembering that \( \rho(H) < 1 \) implies that \( l_{ii} < 1 \) \[7\], and dividing by \( u_i(1 - l_{ii}) > 0 \):
\[ \frac{1}{u_i} \sum_{j \neq i} u_j \left( \frac{l_{ij}}{1 - l_{ii}} \right) \leq \frac{1}{u_i} \sum_{j=1}^{m} u_j l_{ij} \quad \forall i \in m \]

thus, by hypotheses and using relations (28) and (29) we have:
\[ \|U^{-1} H_{cs} U\|_\infty \leq \|U^{-1} HU\|_\infty < 1 \quad (32) \]

with this result and relation (30) for the positive diagonal matrix \( U \) for which the minimum is obtained:
\[ \|U^{-1} H_{cs} U\|_\infty \leq \rho(H) = \inf_{U \in \mathcal{P}} \|U^{-1} HU\|_\infty \quad (33) \]

finally, with (31) and (33),
\[ \rho(H_{cs}) \leq \|U^{-1} H_{cs} U\|_\infty \leq \rho(H) \quad (34) \]

therefore, by hypotheses:
\[ \rho(H_{cs}) \leq \rho(H) < 1 \quad (35) \]

\[ \blacksquare \]

**Remark 4** Satisfied Assumption 1, if any algorithm can prove its convergence using Lemma 2, then the CS method also converges; and even more, it probably solves the problem in fewer iterations because it has a smaller Figure of Merit \( \rho(H) \).

## 6 Conclusions

The *Component Solution* method is a typical algorithm mainly developed for a parallel environment. The conceptual ideas of the method were introduced in a parallel synchronous environment as well as in a sequential environment. However, the main advantage of using the CS method arises in an asynchronous environment where it has a natural implementation without wasting time synchronizing the different processors of a distributed system, probably with unbalanced computation load.

An asynchronous implementation of the CS method was described in section 4 and a sufficient condition for its convergence was derived. The main advantage of the method was studied in section 5, i.e. the ability of the asynchronous CS method of solving problems in fewer iterations than any other asynchronous block iterative method that satisfies the conditions of Lemma 2. This important property of the CS method was discussed by analyzing experimental results of a very simple example problem and later formally proved in Theorem 3, which may be considered as the main contribution of this work.
References


