ELSEVIER

The Journal of Systems and Software 60 (2002) 141–148

The Journal of Systems and Software

www.elsevier.com/locate/jss

Implementation of distributed iterative algorithm for optimal control problems on several parallel architectures

Mohamed Jarraya *, Didier El Baz

LAAS-CNRS, 7 Avenue Colonel Roche 31077, Toulouse Cedex 4, France

Received 1 March 2001

Abstract

We consider an optimal control problem without constraint. The solution of this problem leads to a large scale block linear system. We present a two-stage method which is well suited to an asynchronous implementation with flexible communication whereby every processor can have access at any time and without any fixed rule to the current value of the components of the iteration vector updated by any other processor. The implementation is carried out on three types of architecture: a super computer with distributed memory CRAY T3E, a shared memory symmetric multiprocessor (SMP) and a high speed network of SMP.

Keywords: Optimal control; Two-stage method; Asynchronous iterations with flexible communication; CRAY T3E; SMP; Network of SMP

1. Introduction

The need for optimal control of continuous complex dynamic process subject to some disturbance can require the solution of a large scale system. Taking into account the current development of computer architectures, it appears very useful to obtain optimal control law by iterative methods which are well adapted to parallel computation. We present in this paper the solution of an optimal control problem without constraint by a two-stage method (see Frommer and Szyld, 1994). This method is well suited to an asynchronous implementation with flexible communication. This new class of asynchronous iterative algorithms was proposed in Miellou et al. (1998) for the solution of boundary value problems in the M-function context. It was applied in El Baz (1996) to the solution of nonlinear network flow problems. In this context, every processor can have access at any time and without any fixed rule to the current value of the components of the iteration vector updated by any other processor; this value can correspond to a new update or a partial update which is not labelled explicitly by an iteration number. Asynchronous iterations with flexible communication are tightly bound to the concept of submappings and supermappings which is associated to the generation of monotone sequence of vectors. This is the reason that led us to consider the approach proposed in Jacquemard (1977). This approach generates an increasing sequence of subsolutions and a decreasing sequence of supersolutions which converge towards a single solution of the problem.

The implementation is carried out on three types of architecture: a super computer with distributed memory CRAY T3E by using message-passing libraries such as MPI or SHMEM, a shared memory symmetric multiprocessor (SMP) by using a POSIX thread library and a network of SMP by using a message-passing and POSIX thread libraries. The later implementation permits us to use all the material resources of the distributed system. We thus considered two types of parallelism: one between the different machines of the local network and the other between the processors of the SMP.

Sections 2 and 3 deal with the description and study of the optimal control problem. An asynchronous two-stage algorithm with flexible communication is presented in Section 4. The implementation and the experimental results are presented and analyzed in Section 5.

* Corresponding author. Tel.: +33-5-61-33-64-50; fax: +33-5-61-33-69-69.

E-mail address: jarraya@laas.fr (M. Jarraya).

0164-1212/02/$ - see front matter
PII: S0164-1212(01)00086-3

2. Description of the problem

Let us consider a vertical oven with 12 heating zones (see Spiteri, 1984). The study consists in bringing temperature z raised on n points of the oven at a temperature z_d within finite time $(T' - t_0)$ which represents the horizon of the command. It is thus necessary to obtain the command u that corresponds to the intensity of the currents of regulations, such that, at time T', temperature z of the oven is uniformly equal to z_d. We consider the following criterion:

$$J = \frac{1}{2}\int_{t_0}^{T'} (|Cy(t) - z_d|_2^2 + k|u(t) - u_d|_2^2)\,\mathrm{d}t. \tag{1}$$

$y \in \mathbb{R}^{2n}$ is the state of the system which is represented by the temperature of n points of the bar and the temperature of n points of the oven, $u \in \mathbb{R}^m$ is the control vector. The following model was obtained by studying the evolution of the system:

$$\begin{cases} \frac{\mathrm{d}y(t)}{\mathrm{d}t} + Ay(t) = Bu(t) \quad \forall t \in [t_0, T'], \\ y(t_0) = y_0, \\ z(t) = Cy(t), \end{cases} \tag{2}$$

where A, B and C are matrices with respective dimension $(2n, 2n)$, $(2n, m)$ and $(n, 2n)$, A is also an M-matrix, i.e., $a_{ii} > 0$ for all i, $a_{ij} \leqslant 0$ for all $i \neq j$, A is nonsingular and $A^{-1} \geqslant 0$. Eq. (2) is the state equation of the model and variable z is the observation of the system. The Hamiltonian of the optimal control problem is given as follows:

$$H = \frac{1}{2}(|Cy(t) - z_d|_2^2 + k|u(t) - u_d|_2^2) + p^{\mathrm{T}}(t)[-Ay(t) + Bu(t)], \tag{3}$$

where p is the costate vector. The canonical equations of the problem to be solved are:

$$\begin{cases} \frac{\partial H}{\partial y} = -\frac{\mathrm{d}p}{\mathrm{d}t}, \\ \frac{\partial H}{\partial u} + \partial \Psi_{U_{\mathrm{ad}}} \ni 0, \\ \frac{\partial H}{\partial p} = \frac{\partial y}{\partial t}, \end{cases} \tag{4}$$

where $\partial \Psi_{U_{\mathrm{ad}}}$ is the subdifferential mapping of the indicator function $\Psi_{U_{\mathrm{ad}}}$ of the convex set U_{ad}.

3. Study of an optimal control problem without constraint

If there is no constraint of command admissibility, then $\partial \Psi_{U_{\mathrm{ad}}} = 0$. After development, simplification and discretization in the interval of time $[0, T']$ by integrating the state and the costate, respectively, in the direct and reverse sense with an approximation by Euler method we have:

$$\begin{cases} \frac{y(t+h)-y(t)}{h} + Ay(t+h) + \frac{1}{k}BB^*p(t+h) = B_{u_d}, \\ \frac{p(t)-p(t+h)}{h} + A^*p(t) - C^*Cy(t) = -C^*z_d, \\ y(0) = p(T') = 0. \end{cases} \tag{5}$$

We can transform the system (5) into a linear fixed point equation

$$\begin{pmatrix} Y \\ P \end{pmatrix} = \tilde{A}\begin{pmatrix} Y \\ P \end{pmatrix} + \tilde{B},$$

where $\tilde{A}$ is an M-matrix associated with the system (5), $Y^{\mathrm{T}} = [y_1, \ldots, y_N]$ and $P^{\mathrm{T}} = [p_0, \ldots, p_{N-1}]$ are, respectively, the state and costate vectors with N blocks that represent the number of discretization points. We present now a splitting of the matrix $\tilde{A}$ in two nonnegative matrices $\tilde{A}_1$ and $\tilde{A}_2$ such that $\tilde{A} = \tilde{A}_1 - \tilde{A}_2$ and $\rho(\tilde{A}_1 + \tilde{A}_2) < 1$. This splitting enables us to determine two monotone sequences that represent, respectively, an approximation of the solution by a subsolution and supersolution (see Jacquemard, 1977). The choice of a constant vector in time as subsolution or supersolution of the system (5) is equivalent to determine these two vectors from a reduced linear system such as:

$$\begin{cases} Ay(t+h) + \frac{1}{k}BB^*p(t+h) = Bu_d, \\ A^*p(t) - C^*Cy(t) = -C^*z_d. \end{cases} \tag{6}$$

Let us denote by $u^{(0)} = (\check{y}^{(0)}, \check{p}^{(0)})^{\mathrm{T}}$ and $v^{(0)} = (\hat{y}^{(0)}, \hat{p}^{(0)})^{\mathrm{T}}$, respectively, a subsolution and a supersolution of system (6); then $U^{(0)} = (\check{y}^{(0)}, \ldots, \check{y}^{(0)}, \check{p}^{(0)}, \ldots, \check{p}^{(0)})^{\mathrm{T}}$ and $V^{(0)} = (\hat{y}^{(0)}, \ldots \hat{y}^{(0)}, \hat{p}^{(0)}, \ldots \hat{p}^{(0)})^{\mathrm{T}}$ with dimension $4nN$ are, respectively, a subsolution and a supersolution of (5). The numerical computation of $u^{(0)}$ and $v^{(0)}$ was carried out by a construction process suggested by Krasnosel'skii et al. (1972). The subsolution and the supersolution verify the following inequalities:

$$\begin{cases} U^{(0)} \leqslant V^{(0)}, \\ U^{(0)} \leqslant \tilde{A}_1 U^{(0)} - \tilde{A}_2 V^{(0)} + \tilde{B}, \\ V^{(0)} \geqslant \tilde{A}_1 V^{(0)} - \tilde{A}_2 U^{(0)} + \tilde{B}. \end{cases} \tag{7}$$

4. Parallel iterative scheme

In this paragraph, we present a parallel scheme which relies on a two-stage iterative method as suggested in Frommer and Szyld (1994). We show that the iterative scheme of computation is based on a fixed point mapping which is a submapping. The iterative methods can be implemented in a distributed way. We wish to implement asynchronous iterations with flexible communication. Asynchronous iterations with flexible communication are general methods whereby each processor can have access to the current state of the other processors. Thus, this new class of parallel algorithms allows exchange of values which were not allowed by the classical asynchronous iterative models

studied in Baudet (1978). Indeed, the current value of each component of the updated vector can be read or communicated to the various processors at any time and without any fixed rule, whereas a communication occurs only at the end of each updating in the standard asynchronous iterative scheme.

4.1. Two-stage iterative scheme

System (5) can be written as follows:

$$\begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{11}^* \end{pmatrix} \begin{pmatrix} Y \\ P \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \end{pmatrix}, \tag{8}$$

where

$$M_{11} = \begin{pmatrix} (hA+I) & \mathbb{O}\ldots & \mathbb{O} \\ -I & (hA+I) & \vdots \\ & \ddots & \mathbb{O} \\ \mathbb{O} & & -I\ (hA+I) \end{pmatrix},$$

$$M_{12} = \begin{pmatrix} \mathbb{O}\ \frac{h}{k}BB^* & \\ \vdots & \ddots \\ & \frac{h}{k}BB^* \\ \mathbb{O}\ldots & \mathbb{O} \end{pmatrix},$$

$$M_{21} = \begin{pmatrix} \mathbb{O} & \ldots & \mathbb{O} \\ -hC^*C & \vdots & \\ & \ddots & \\ & -hC^*C & \mathbb{O} \end{pmatrix},$$

$$M_{11}^* = \begin{pmatrix} (hA^*+I) & -I & \ldots & \mathbb{O} \\ \mathbb{O} & (hA^*+I) & & \vdots \\ & \ddots & & -I \\ \mathbb{O} & & \mathbb{O} & (hA^*+I) \end{pmatrix}.$$

Let us decompose one of the two halves of the system, the other one can be decomposed by using the same method. The splitting of M_{11} in $M_{11}^1 - M_{11}^2$, where M_{11}^1 is a nonsingular matrix, gives the following iterative scheme:

$$M_{11}^1 Y^{(r+1)} = M_{11}^2 Y^{(r)} - M_{12}P^{(r)} + E_1. \tag{9}$$

We suppose that M_{11}^1 is a bloc diagonal matrix such that $M_{11}^1 = \mathrm{diag}(M_{11}^{11}, \ldots, M_{11}^{1N}) = \mathrm{diag}(hA+I, \ldots, hA+I)$. In addition, we suppose that it is difficult to solve directly the system with the matrix $(hA+I)$. In this case, we can decompose this matrix into $R-S$, with $R = \mathrm{diag}(ha_{11}+1, ha_{22}+1, \ldots, ha_{(2n)(2n)}+1)$. The two-stage iterative scheme is represented in Fig. 1.

Let $i \in \{1, \ldots, N\}$, $t_i = t_{i-1} + h$, $y(t_i) = y_i$ and $p(t_i) = p_i$. The system (5) can be rewritten as follows:

$$\begin{cases} (hA+I)y_i = y_{i-1} - \frac{h}{k}BB^*p_i + hBu_d, \\ (hA^*+I)p_i = p_{i+1} + hC^*Cy_i - hC^*z_d, \end{cases} \tag{10}$$

where y_i and p_i are, respectively, the ith block of the state and costate vectors. We denote by $y_{i,j}$ the jth component

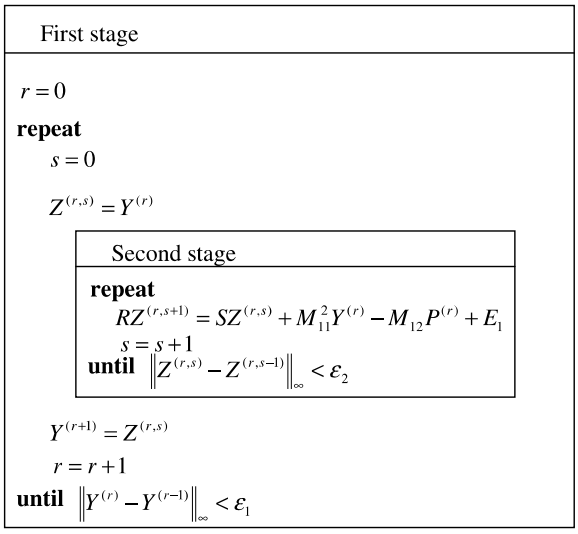

Fig. 1. Two-stage iterative scheme.

of the ith block of the state vector and we denote by $p_{i,j}$ the jth component of the ith block of the costate vector. The jth component of the ith block of the state and costate vectors can be expressed by:

$$\begin{cases} (ha_{jj}+1)y_{i,j} + \sum_{l=1,l\neq j}^{2n} ha_{jl}y_{i,l} \\ \quad = y_{i-1,j} - \frac{h}{k}\sum_{l=1}^{2n}(BB^*)_{jl}P_{i,l} + h(Bu_d)_j, \\ (ha_{jj}^*+1)p_{i,j} + \sum_{l=1,l\neq j}^{2n} ha_{jl}^*p_{i,l} \\ \quad = p_{i+1,j} + h\sum_{l=1}^{2n}(C^*C)_{jl}y_{i,l} - h(C^*z_d)_j. \end{cases} \tag{11}$$

We can introduce a fixed point mapping F defined as follows: for $q \in \{1, \ldots, 2nN\}$, with $q = 2n(i-1)+j$,

$$F_q\begin{pmatrix} Y \\ P \end{pmatrix} = \frac{1}{ha_{jj}+1}\left[-\sum_{l=1,l\neq j}^{2n} ha_{jl}y_{i,l} + y_{i-1,j} - \sum_{l=1}^{2n}\frac{h}{k}\left((BB^*)_{jl}P_{i,l} + h(Bu_d)_j\right)\right],$$

and for $q \in \{2nN+1, \ldots, 4nN\}$, with $q = 2n(i-1)+j+2nN$,

$$F_q\begin{pmatrix} Y \\ P \end{pmatrix} = \frac{1}{ha_{jj}^*+1}\left[-\sum_{l=1,l\neq j}^{2n} ha_{jl}^*p_{i,l} + p_{i+1,j} + \sum_{l=1}^{2n} h\left((C^*C)_{jl}Y_{i,l} - h(C^*z_d)_j\right)\right].$$

In the sequel we shall use the notation $X = (Y,P)^{\mathrm{T}}$.

4.2. Asynchronous parallel iterations with flexible communication

Asynchronous iterative algorithms with flexible communication were first proposed in Miellou et al. (1998) for the solution of boundary value problems in

the M-function context. They were applied to the solution of convex network flow problems in El Baz (1996).

We assume that there is an infinite set of times $T = \{0, 1, 2, \ldots\}$ at which one component of the iterate vector is updated by some processor. Let T_q be the infinite subset of times at which component X_q is updated such that $T_q \cap T_l = \emptyset$ for all $q, l \in \{1, \ldots, 4nN\}$, $q \neq l$. An asynchronous iteration with flexible communication associated with the mapping $F : \mathbb{R}^{4nN} \to \mathbb{R}^{4nN}$ and the starting point $X_q^{(0)} \in \mathbb{R}^{4nN}$, is a sequence $\{X_q^{(r)}\}$ of vectors of $\mathbb{R}^{4nN}$, such that for $q \in \{1, \ldots, 4nN\}$, we have:

$$\begin{cases} X_q^{(r+1)} = F_q(\tilde{X}^{(r)}) & \forall r \in T_q, \\ X_q^{(r+1)} = X_q^{(r)} & \forall r \notin T_q, \end{cases} \tag{12}$$

where for $r = 0, 1, \ldots, \tilde{X}^{(r)}$ is the vector of $\mathbb{R}^{4nN}$ such that

$$\tilde{X}^{(0)} = X^{(0)} \quad \text{and} \quad \tilde{X}^{(r)} \in \langle \max\{X^{(\rho(r))}, \tilde{X}^{(l)}\} X^{(r)} \rangle, \quad r \geqslant 1 \tag{13}$$

and the order interval $\langle X', X'' \rangle = \{X \in \mathbb{R}^{4nN} \,|\, X' \leqslant X \leqslant X''\}$, $\tilde{X}^{(l)}$ is the vector used in the last update of the same component, and the vector $X^{(\rho(r))}$ with components $X_q^{(\rho_q(r))}$ is introduced in order to model the chaotic behavior of the iterative scheme (we have: $0 \leqslant \rho_q(r) \leqslant r$, $\rho_q(r) = r$ if $r \in T_q$, $\rho_q(r)$ is monotonically increasing, and $\lim_{r \to \infty} \rho_q(r) = +\infty$ for all $q \in \{1, \ldots, 4nN\}$.

These algorithms allow the use of partial updates arising from computations in progress. Asynchronous iterations with flexible communication defined by (12) and (13) describe general iterative methods whereby computations are carried out in parallel without any order nor synchronization. The only restrictions are that no component of the iteration vector is abandoned forever and old values of each component of the iteration vector are abandoned as the computation progresses. The use of the current value of components resulting from intermediary steps of updating presents a particular interest in this context of monotone convergence; intuitively, it can speed up the convergence.

4.3. Termination procedure

At the iteration (r), the solution is approximated by $\frac{1}{2}(U^{(r)} + V^{(r)})$ with maximum error $e^{(r)} = \frac{1}{2}(V^{(r)} - U^{(r)})$. The algorithm terminates when $\|e^{(r)}\|_\infty < \epsilon$, where ϵ is the precision of termination. The sequence $\{U^{(r)}\}$ increases and the sequence $\{V^{(r)}\}$ decreases thus the residue $e^{(r)}$ decreases. We propose now a termination procedure based on the circulation of a token between processors. We fix a relationship between processors defined in the following way (Fig. 2).

The node (0) is the father of the node (1) which is the father of the node (2), etc., the node $(\alpha - 1)$ does not have a child and the node (0) does not have a father. Initially, the token is in the node $(\alpha - 1)$. When the last processor detects that its local termination condition is satisfied ($\|e_{\alpha-1}\|_\infty < \epsilon$), it sends the token to his father. Then the father can send the token only if its local termination condition ($\|e_{\alpha-2}\|_\infty < \epsilon$). This scheme progresses gradually until the token arrives to the node (0) which sends a global termination message to all the nodes; then the distributed algorithm finishes.

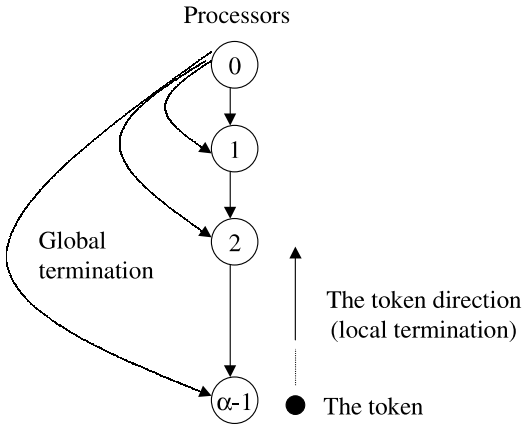

Fig. 2. Termination procedure.

5. Implementation on a parallel architecture

5.1. Decomposition of the problem on a parallel architecture

Let us consider a block decomposition of the system (5). In this case, each processor updates a set of blocks of components of the vector $U^{\mathrm{T}} = (Y, P)$ and exchanges the value of the blocks of components at the border with the other processors. If we consider the matrix of the system (8), the submatrice M_{11}^* has an upper block diagonal, therefore the Gauss Seidel method is like a block Jacobi method. In order to accelerate the relaxation method we transpose the costate block vector according to the following integration direction ($\{0, 1, \ldots, N-1\} \to \{N-1, N-2, \ldots, 0\}$). Thus we obtain the following iterative scheme:

$$\begin{cases} (hA + I) y_i^{(r+1)} = y_{i-1}^{(r+1)} - \frac{h}{k} BB^* p_i^{(r)} + hBu_d, \\ (hA^* + I) p_i^{(r+1)} = p_{i+1}^{(r+1)} + hC^* C y_i^{(r+1)} - hC^* z_d. \end{cases} \tag{14}$$

In this case, the dependency between the block-components of the state and the costate vectors at the iteration $(r+1)$ for subsolutions is presented in Fig. 3, where the relation $a \to b$ implies that b is function of a.

In order to minimize the interactions between processors, we propose that the components with the same index are updated by the same processor. Fig. 4 shows a decomposition of the state and costate vectors on a parallel architecture, where $\check{y}$, $\hat{y}$, $\check{p}$ and $\hat{p}$ are, respectively, the state subsolution, the state supersolution, the costate subsolution and the costate supersolution.

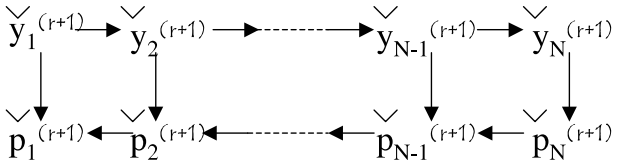

Fig. 3. The dependency between the block-components of the state and the costate vectors.

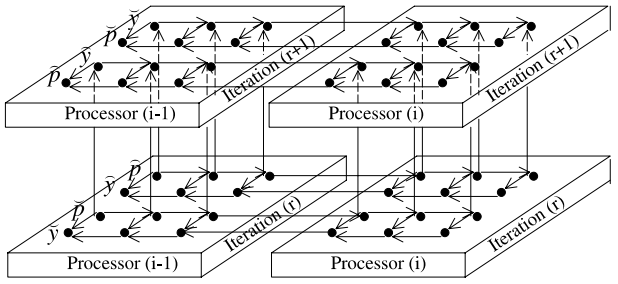

Fig. 4. Decomposition on a parallel architecture.

5.2. Implementation on parallel architectures

The implementation is carried out on three types of architecture: a four processor SMP with a shared memory, a supercomputer CRAY T3E with distributed memory and a high speed network of SMP.

5.2.1. Implementation on a SMP

In the case of synchronous implementation, all threads must be synchronized. We have implemented a synchronization barrier by using a mutex and conditions variables. At the beginning of the synchronization barrier, all the treads execute a lock function to protect a variable that counts the number of threads which are waiting. The last thread that enters the synchronization barrier executes a broadcast signal to the others machines.

In the case of asynchronous iterations with flexible communication, every thread has access when needed to the current value of the components of the iteration vector stored in the shared memory. This access is made before each new updating by a simple read function. Fig. 5 shows models of synchronous and asynchronous iterations with flexible communication on a SMP.

5.2.2. Implementation on a network of SMP

We have used MPI message-passing and Posix thread libraries in order to implement the two-stage iterative algorithms on this architecture. We have assigned all the communications between SMP to only one thread of each SMP because the driver of communications denies concurrent access by several threads. In the case of synchronous implementation, all threads running on a SMP must be synchronized and distant threads must also be synchronized. The synchronization of threads running on a SMP is similar to the case presented in the previous subsection. The synchronization between distant threads is accomplished by using a blocking receive function. Fig. 6 shows a model of synchronous iterations on a network of SMP.

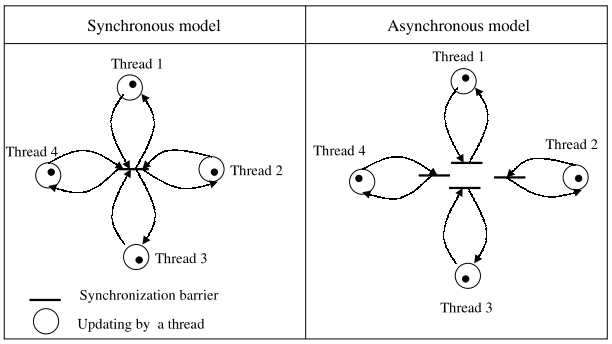

Fig. 5. Synchronous and asynchronous models on a SMP.

Asynchronous iterations with flexible communication are implemented as follows: we consider two levels. In the first level which corresponds to a SMP, every thread has access when needed to the current value of the components of the iterate vector. Clearly the current values which are read do not necessarily correspond to the result of an updating phase and are not necessarily labelled by an iteration number. In the second level which corresponds to the network of SMP, partial updates are exchanged via distant threads by using non blocking send and receive functions of MPICH (c.f. Gropp and Lusk, 1999). Each SMP sends from time to time, the current state of each component of the iterate vector. Only one thread of each machine sends the partial updates. We note that the current value of each component of the iterate vector computed by the other threads of the same SMP are sent at the same instant. The receive function takes into account the last values received by the SMP.

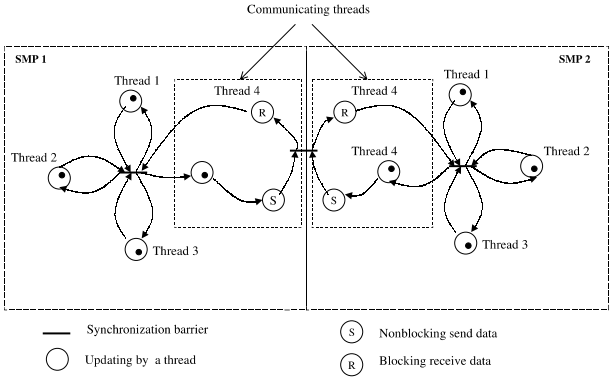

Fig. 6. Synchronous model on a network of SMP.

5.2.3. Implementation on CRAY T3E

On the supercomputer CRAY T3E, synchronous and asynchronous iterations with flexible communications are implemented by using two types of message-passing libraries. The synchronous algorithm uses the MPI library in order to communicate data between processors. The updating and data exchange are made sequentially. Every processor sends the border block-components to the others processors by using the nonblocking MPI_Isend() function and receives data by using a blocking MPI_Recv() function that makes the synchronization between all processors. According to Fig. 4, the updating of the first block-component "first_block" needs the value of the last block-component "last_block" which is updated by his father (see the processors relationship in Section 4.3) and the updating of the last block-component needs the value of the first block-components which is updated by his child. The following function defines the synchronous protocol between processors, where Z is the iterate vector.

FUNCTION : exchange_data_mpi(Z,j)
if (j = begin_block) **then**
  if (current_process <> process_number-1) **then**
    MPI_Isend($Z_{(last\_block)}$,current_process+1)
  end if
  if (current_process <> 0) **then**
    MPI_Recv($Z_{(begin\_block-1)}$,current process-1)
  end if
end if
if (j = last_block) **then**
  if (current_process <> 0) **then**
    MPI_Isend($Z_{(begin\_block)}$,current_process-1)
  end if
  if (current_process <> process_number-1) **then**
    MPI_Recv($Z_{(last\_block+1)}$,current_process+1)
  end if
end if

In the case of asynchronous iterations with flexible communication, each processor calls the shmem_get() function of the SHMEM library in order to receive the current value of components updated by the other processors. This value can be a partial updating which is not labelled explicitly by a number of iteration. The function exchange_data_shmem() shows how the flexible communications are implemented between processors.

FUNCTION : exchange_data_shmem(Z,j)
if (j = begin_block) **then**
  if (current_process <> 0) **then**
    shmem_get($Z_{(begin\_block-1)}$,current_process-1)
  end if
end if
if (j = last_block) **then**
  if (current_process <> process_number-1) **then**
    shmem_get($Z_{(last\_block+1)}$,current_process+1)
  end if
end if

Remarks.

- It has be shown in Jarraya et al. (1998) that the use of shmem_get() is the most efficient for asynchronous iterations with flexible communication.
- We can use SHMEM functions in order to obtain synchronization, but with MPI the implementation is more natural.

5.3. Experimental results

We have implemented the two-stage iterative scheme applied to a vertical oven with 12 heating zones. The matrices A, B and C are given as follows:

$$A = \begin{pmatrix} D_1 & S+\theta & \epsilon S & \epsilon^2 S & \dots & \epsilon^9 S & \epsilon^{10} S \\ \theta & D & S+\theta & \epsilon S & \dots & \epsilon^8 S & \epsilon^9 S \\ 0 & \theta & D & S+\theta & & . & . \\ 0 & 0 & \theta & D & & . & . \\ . & . & & & & S+\theta & \epsilon S \\ . & . & & & \theta & D & S+\theta \\ 0 & 0 & . & . & 0 & \theta & D_{12} \end{pmatrix},$$

where each block, D_1, D, D_{12}, S and θ, are matrices defined as follows:

$$D1 = \begin{pmatrix} 0.38 & -0.196 \\ -0.0068 & 0.0629 \end{pmatrix},$$

$$D = \begin{pmatrix} 0.39 & -0.196 \\ -0.0068 & 0.0682 \end{pmatrix},$$

$$D_{12} = \begin{pmatrix} 0.44 & -0.196 \\ -0.0068 & 0.0559 \end{pmatrix},$$

$$S = \begin{pmatrix} -0.0057 & -0.012 \\ -0.0003 & -0.0008 \end{pmatrix},$$

$$\theta = \begin{pmatrix} -0.0651 & 0 \\ 0 & -0.0031 \end{pmatrix}.$$

$$B_{ij} = \begin{cases} \delta & \text{if } i = 2^p - 1 \text{ and } j = i - p; p \in \mathbb{N}, \\ \mathbb{0} & \text{if not}, \end{cases}$$

$$C_{ij} = \begin{cases} 1 & \text{if } j = 2i, \\ 0 & \text{if not}, \end{cases}$$

with $\epsilon = 0.67$ and $\delta = 0.00195$. Constant vectors u_d and u_d are given by: $u_d^{\mathrm{T}} = [1075, 825, 840, 842, 845, 850, 858, 871, 893, 930, 958, 1483]$ cal/min and $z_d^{\mathrm{T}} = [30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30]$°C.

The interval of time is equal to 180 min with a step of discretization equal to 6 s, thus the number of blocks is 1800, so the size of the problem is 86 400. The precision required for the first stage is $\epsilon_1 = 10^{-4}$ and for the second stage $\epsilon_2 = 10^{-9}$. We have chosen $k = 0.0012$.

We have used a four processors Pentium II Personal Computer whereby every processor has a clock

frequency of 200 MHz, the CRAY T3E of IDRIS-CNRS having 256 DIGITAL α processors and a network of Personal Computer with bi-processor Pentium II whereby every processor has a clock frequency of 233 MHz. Two types of network were considered: an Ethernet network with a bandwidth of 100 Mbits/s and Myrinet network with a bandwidth of 1 Gbits/s.

We have measured the speed up of synchronous and asynchronous iterations with flexible communications compared to the sequential execution on one processor. Tables 1 and 2 show the execution time and the speed up of the parallel implementations in function of the number of threads running, respectively, on a SMP and on network of SMP. This number is represented by an n-uple (a, b, c, d) where n denotes the total number of SMP and the value of each component a, b, c and d gives, respectively, the number of threads that are carried out on the corresponding SMP. Figs. 7 and 8 show, respectively, the execution time and the speed up of the parallel implementation on the CRAY T3E.

Table 1 shows that asynchronous iterations with flexible communications are more efficient than synchronous iterations on a SMP. The number of steps in the second stage varies from iteration to iteration and

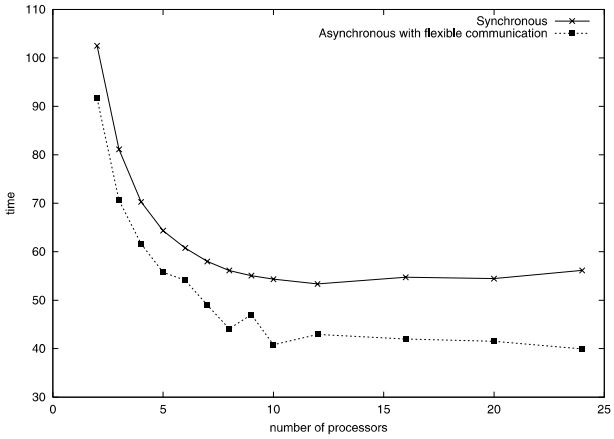

Fig. 7. Execution time on CRAY T3E.

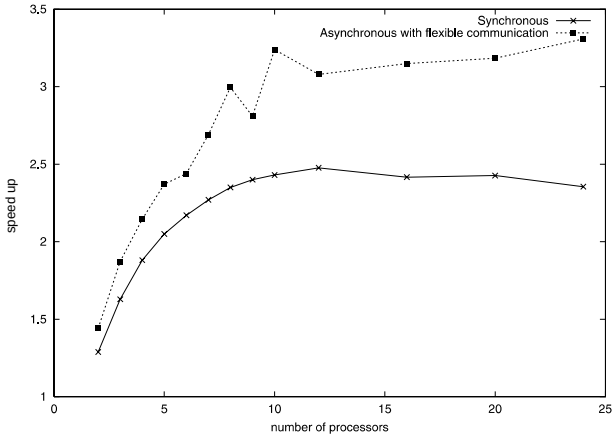

Fig. 8. Speed up on CRAY T3E.

Table 1
Implementation on a SMP

Threads	Synchronous		Asynchronous with FC	
	Time (s)	Speed up	Time (s)	Speed up
1	677.13	1	677.13	1
2	512.36	1.321	465.00	1.455
3	401.90	1.684	347.51	1.948
4	347.96	1.946	313.72	2.158

Table 2
Implementation on a network of SMP

Threads	Synchronous		Asynchronous with FC	
	Times (s)	Speed up	Time (s)	Speed up
On Myrinet network of SMP using MPICH-gmm				
(1,0,0)	361.94	1	361.94	1
(1,1,0)	275.24	1.314	326	1.109
(2,1,0)	220.50	1.641	242.63	1.491
(2,2,0)	191.75	1.887	205.15	1.764
(1,1,1)	213.79	1.692	259.69	1.393
(2,1,1)	183.45	1.972	212.70	1.701
(2,2,1)	171.25	2.113	185.31	1.953
(2,2,2)	159.57	2.268	160.51	2.254
On Ethernet network of SMP using MPICH				
(1,1,0)	276	1.310	419.77	0.862
(2,1,0)	228.92	1.581	280.07	1.292
(2,2,0)	194.95	1.856	285.78	1.266
(1,1,1)	214.64	1.686	277.45	1.300
(2,1,1)	183.76	1.969	229.15	1.570
(2,2,1)	185.21	1.954	220.27	1.640
(2,2,2)	179.47	2.016	183.50	1.972

the access to the current value permits us to use the best approximation to the solution of the problem. The same result was obtained on the CRAY T3E by using a get function of the SHMEM library which has access to the current value of the iterate vector. In the case of a network of SMP synchronous iterations are more efficient than asynchronous iterations with flexible communication. This is due in fact firstly to the nature of the communication function of MPICH which cannot offer the possibility to have an easy access to the current value of the iterate vector updated by other SMP and secondly to the latency time to the communication functions. On the Myrinet network we have measured a latency equal to 27 μs and on the Ethernet network a latency equal to 250 μs. On the CRAY T3E the latency time is about 1 μs. Table 2 shows that the parallel implementation are more efficient on the Myrinet network which has the highest bandwidth and the lower latency than on the Ethernet network. We note also that the regular interconnection architecture of CRAY T3E permits us to obtain better speed up when the number of processor increases. The use of the shmem_get() function of the SHMEM library of the CRAY T3E in the case of asynchronous iterations with flexible communication is quite similar to the use of a read function of a shared memory architecture and the performance on CRAY T3E below four processors is very close to the performance on a SMP.

6. Conclusion

In this paper we have studied and implemented parallel synchronous and asynchronous iterations with flexible communication in order to solve an optimal control problem without constraint. The basic iterative procedure relies on a two-stage iterative scheme of computation. The implementation is carried out on three types of architecture: a shared memory SMP, a network of SMP and a supercomputer CRAY T3E with distributed memory. We have shown that the asynchronous method is faster than the synchronous method on a SMP and on the CRAY T3E and lightly slower on a network of SMP. This type of parallel implementation requires shared memory or distributed memory architectures for which all processors can have an access to the current value of the iterate vector updated by the other processors. However we have not obtained excellent speed up because the costate relaxation method which integrates the state and the costate vectors, respectively, in the direct and reverse directions is by nature very sequential.

Acknowledgements

The authors wish to thank P. Spiteri and J.C. Miellou for their advices and IDRIS-CNRS for its support.

References

Baudet, G.M., 1978. Asynchronous iterative methods for multiprocessors. J. Assoc. Comput. 2, 226–244.

El Baz, D., Spiteri, P., Miellou, J.C., Gazen, D., 1996. Asynchronous iterative algorithms with flexible communication for nonlinear network flow problems. J. Parallel Distrib. Comput. 38, 1–15.

Frommer, A., Szyld, D.B., 1994. Asynchronous two-stage iterative methods. Numer. Math. 69, 141–153.

Gropp, W., Lusk, E., 1999. User's Guide for MPICH, A Portable Implementation of MPI Version 1.2.0. Mathematics and Computer Science Division, MIT, Cambridge, MA, pp. 1999.

Jarraya, M., El Baz, D., Gazen, D., 1998. Mise en œuvre de méthodes itératives asynchrones avec communication flexible 2, Implémentation sur CRAY T3E, SMP et réseau de stations. Calculateurs Parallèles Rèseaux et Systèmes répartis 10 (4), 439–447.

Jacquemard, C., 1977. Contribution à l'étude d'algorithme de relaxation à convergence monotone, Thèse de 3ème cycle, Université de Besancon.

Krasnosel'skii, M.A., Vainikko, G.M., Zabreiko, P.P., Rutitskii, Ta.B., Stetsnko, V.Ya., 1972. Approximate Solution of Operator Equations. Wolters-Noordhoff, Groningen.

Miellou, J.C., El Baz, D., Spiteri, P., 1998. A new class of iterative algorithms with order intervals. Math. Comput. 67, 237–255.

Spiteri, P., 1984. Contribution à l'étude de grands systèmes non linaires: Comportement d'algorithme itératifs, stabilité de systèmes continus. Thèse de Doctorat d'état, Faculté des sciences et des techniques de l'université de Franche-Comté.