Flexible communication for parallel asynchronous methods with application to a nonlinear optimization problem

D. El Baz[a] *, D. Gazen[a], M. Jarraya[a], P. Spiteri[b], and J. C. Miellou[c]

[a]LAAS du CNRS, 7, avenue du Colonel Roche, 31077 Toulouse Cedex 4, France

[b]ENSEEIHT-IRIT/LIMA, 2, rue Camichel, 31071 Toulouse Cedex 4, France

[c]LCS, Université de Franche Comté, 16, Route de Gray, 25030 Besançon Cedex, France

We consider the solution of a particular class of nonlinear optimization problems: convex network flows and we show how we have implemented new parallel iterative methods: asynchronous iterations with flexible communication on two types of parallel architectures: a distributed memory supercomputer Cray T3E and a Symmetric MultiProcessor. We deal with asynchronous algorithms whereby every processor can exchange partial updates resulting from intermediate steps of computation with other processors; i.e. every processor can have access to the current value of the components of the iteration vector updated by any other processor. Computational results are presented and analyzed. In particular, the performance of asynchronous iterations with flexible communication is compared with the performance of synchronous and standard asynchronous algorithms.

1. INTRODUCTION

We consider the strictly convex network flow problem. This problem occurs in many domains: electrical networks, gas or water distribution, financial models, communication and transportation networks. Nonlinear network flow problems require intensive computation. We concentrate on the dual problem which is unconstrained, differentiable and well suited for solution via parallel iterative methods. It was shown in [1] and [3] that the structure of the dual problem allows the successful application of asynchronous relaxation and gradient algorithms whereby iterations are carried out in parallel without any order nor synchronization.

In this paper, we study the implementation of a new class of asynchronous iterative methods: asynchronous iterations with flexible communication. This new class of algorithms was first proposed in [7] for the solution of boundary value problems in the M-function context. It was applied in [4] to the solution of nonlinear network flows in a more general context. Data exchange occurs only at the end of each updating phase in a standard asynchronous implementation. In an asynchronous implementation with flexible communication, every processor can exchange partial updates resulting from intermediate steps of computation with other processors. Every processor can have access at any time

*this study was partially supported by IDRIS-CNRS France

and without any fixed rule to the current value of the components of the iteration vector updated by any other processor. We will see in the sequel that the use of partial updates can speed up the convergence. Implementations of asynchronous iterative methods with flexible communication on a Cray T3E and a Sun Sparcstation 20 with four processors are presented. Finally, computational results are given and analyzed.

The second Section deals with the convex network flow problem. Asynchronous iterations with flexible communication are presented in Section three. Implementations are displayed in Section four. Experimental results are given and analyzed in Section five.

2. THE PROBLEM

Let $G = (N, A)$ be a connected directed graph. N is referred to as the set of nodes, $A \subset N \times N$ is referred to as the set of arcs, and the cardinal number of N is denoted by n. Let $c_{ij} : R \to (-\infty, +\infty]$ be the cost function associated with each arc $(i, j) \in A$, c_{ij} is a function of the flow of the arc (i, j) which is denoted by f_{ij}. Let b_i be the supply or demand at node $i \in N$, satisfying $\sum_{i \in N} b_i = 0$. The problem is to minimize the total cost subject to a conservation of flow constraint at each node:

$$min \sum_{(i,j)\in A} c_{ij}(f_{ij}), \text{ subject to } \sum_{(i,j)\in A} f_{ij} - \sum_{(m,i)\in A} f_{mi} = b_i, \forall i \in N. \tag{1}$$

We assume that problem (1) has a feasible solution. We consider the following standing assumptions on c_{ij}.
Assumption 2.1. c_{ij} is strictly convex.
Assumption 2.2. c_{ij} is lower semicontinuous.
Assumption 2.3. The conjugate convex function of c_{ij}, defined by

$$c^*_{ij}(t_{ij}) = \sup_{f_{ij}} \{t_{ij}.f_{ij} - c_{ij}(f_{ij})\}, \tag{2}$$

is real valued, i.e. $-\infty < c^*_{ij}(t_{ij}) < +\infty$ for all real t_{ij}.

Assumption 2.3 implies that $\lim_{|f_{ij}|\to\infty} c_{ij}(f_{ij}) = +\infty$. Therefore the objective function of problem (1) has bounded level sets (see [1]). It follows that there exists an optimal solution for problem (1) which must be unique in view of Assumption 2.1. By the strict convexity of c_{ij}, c^*_{ij} is also continuously differentiable.

A dual problem for (1) is given by

$$\min_{p\in R^n} q(p), \text{ subject to no constraints on the vector } p = \{p_i | i \in N\}, \tag{3}$$

where q is the dual functional given by

$$q(p) = \sum_{(i,j)\in A} c^*_{ij}(p_i - p_j) - \sum_{i\in N} b_i.p_i. \tag{4}$$

We refer to p as a price vector and its components as prices. The i-th price, p_i, is a Lagrange multiplier associated with the i-th conservation of flow constraint. Existence of an optimal solution of the dual problem can be guaranteed under an additional regular feasibility assumption (see [1]). On the other hand the optimal solution of the dual problem is never unique since adding the same constant to all coordinates of a price vector

p leaves the dual cost unaffected. We can remove this degree of freedom by constraining the price of one node, say a destination node d, to be zero. Consider the set $P = \{p \in R^n | p_d = 0\}$. We concentrate on the reduced dual problem:

$$min_{p \in P} q(p). \tag{5}$$

Assumption 2.4. The reduced dual optimal solution set P^* is nonempty and compact.

Assumption 2.4 is not overly restrictive, the reader is referred to [1] for various examples. It follows from Assumption 2.4 that there exists a minimal and a maximal optimal solution of the reduced dual problem, i.e. there exists $\underline{p}, \overline{p} \in P^*$ such that $\underline{p} \leq p \leq \overline{p}$, for all $p \in P^*$, where $\underline{p} \leq p$ is a component-wise partial ordering relation on R^n (see [1]). In the sequel $g(p)$ will denote the gradient vector of the dual functional with components:

$$g_i(p) = \frac{\partial q(p)}{\partial p_i} = \sum_{(i,j) \in A} \nabla c^*_{ij}(p_i - p_j) - \sum_{(m,i) \in A} \nabla c^*_{mi}(p_m - p_i) - b_i, i \in N. \tag{6}$$

Since $g_i(p)$ is a partial derivative of a differentiable convex function, it is continuous and monotonically nondecreasing in the i-th coordinate. For simplicity $(\hat{p}_i; p)$ will denote in the sequel the vector of R^n with i-th component equal to $\hat{p}_i$ and j-th component equal to $p_j, j \in N - \{i\}$.

3. PARALLEL ITERATIVE ALGORITHMS

The reduced dual problem is unconstrained and differentiable. Relaxation methods or approximate relaxation methods are well suited to the solution of this class of problems.

Given a price vector $p \in P$, a node $i \in N - \{d\}$ is selected and its price p_i is changed to a value $\hat{p}_i$ such that the dual cost is minimized at $\hat{p}_i$ with respect to the i-th price (i.e. $g_i(\hat{p}_i; p) = 0$), all others prices being kept constant. The algorithm proceeds by relaxing the price of all nodes belonging to $N - \{d\}$ in cyclic order and repeating the process (see [1]). Consider now the point-to-set mapping $F_i, i \in N - \{d\}$, which assigns to a price vector $p \in P$, the set of all prices $\hat{p}_i$ that minimize the dual cost along the i-th price starting from p, i.e. $F_i(p) = \{\hat{p}_i | g_i(\hat{p}_i; p) = 0\}$. It is well known that a real valued convex function having one compact level set, has all its level sets compact. Therefore, under Assumption 2.4 the sets $F_i(p), p \in P, i \in N - \{d\}$, are all nonempty, compact intervals. It follows that the minimal and maximal relaxation mappings denoted by $\underline{F}$ and $\overline{F}$, respectively, with components:

$$\underline{F}_i(p) = \min_{\hat{p}_i \in F_i(p)} \hat{p}_i \text{ and } \overline{F}_i(p) = \max_{\hat{p}_i \in F_i(p)} \hat{p}_i, i \in N - \{d\}, \tag{7}$$

respectively, are well defined on the set P (see [1]). We present now a class of approximate relaxation mappings called submappings which is of primary importance in the definition of parallel asynchronous iterations with flexible communication.

3.1. Submappings

Definition 3.1. A mapping $\check{F}$, with components $\check{F}_i, i \in N - \{d\}$, is a submapping associated with the minimal relaxation mapping $\underline{F}$ on $P' = \{p \in P | p \leq \underline{p}\}$, we also say that $\check{F}$ is a submapping on P' for simplicity, if for all $i \in N - \{d\}$ and $p \in P'$ such that $p_i \leq \underline{F}_i(p)$, we have $\check{F}_i(p) \in [p_i, \underline{F}_i(p)]$ and $\check{F}_i(p) \neq p_i$ if $\underline{F}_i(p) \neq p_i$.

Now we introduce a property referring to continuity.

Definition 3.2. A submapping $\check{F}$ is m-continuous on P' if there exists a submapping $\check{F}'$ on P' such that for all $i \in N - \{d\}$ and $p \in P'$, satisfying $p_i \leq \underline{F}_i(p)$, we have:

$$\check{F}'_i(p) \in [p_i, \check{F}_i(p)], \tag{8}$$

and

$$p(k) \uparrow p' \in P', k \to \infty \Rightarrow \check{F}'_i(p(k)) \uparrow \check{F}'_i(p'), k \to \infty, \text{ for all } i \in N - \{d\}, \tag{9}$$

where $p(k) \uparrow p', k \to \infty$, means that $p(0) \leq p(1) \leq ... \leq p(k) \leq p(k+1) \leq ... \leq p'$ and $\lim_{k\to\infty} p(k) = p'$.

We present now submappings which are related to the gradient mapping. The components F'_i of the gradient mapping F' are defined by

$$F'_i(p) = p_i - \frac{1}{\alpha} g_i(p), \text{ for all } i \in N - \{d\} \text{ and } p \in P. \tag{10}$$

where α is a positive constant. Clearly F' is continuous since g is continuous.

Assumption 3.2. c_{ij} is strongly convex with modulus $\frac{1}{\beta}$.

It was shown in [3] that under Assumptions 2.1 to 2.4 and Assumption 3.2, there exists a constant $\alpha = \beta . \max_{i \in N} a_i$, where a_i denotes the degree of node $i \in N$, such that the gradient mapping F' is monotone on P. It was shown in [4] that if Assumptions 2.1 to 2.4 and 3.2 hold and $\alpha = \beta . \max_{i \in N} a_i$, then the approximate relaxation mapping $\check{F}$ defined as follows is an m-continuous submapping on $P' : \check{F}_i(p) = p_i^{q'}$, for all $i \in N - \{d\}$ and $p \in P$, where $p_i^q = F'_i(p_i^{q-1}; p), q = 1, ..., q', p_i^0 = p_i$, and q' is the smallest positive integer such that $\left| g_i(p_i^{q'}; p) \right| \leq \epsilon'$, ϵ' being a given constant. This mapping performs an approximate line search.

3.2. Asynchronous iterations with flexible communication

It follows from equation (6) that only local data, i.e. prices of adjacent nodes, are needed to update a given price. Iterative methods associated with a submapping can be implemented in parallel. Since idle time due to synchronization may be nonnegligible, parallel implementation may be improved by using procedures whereby computations are carried out concurrently without any order nor synchronization, namely asynchronous procedures. For further details about asynchronous iterations the reader is referred to [2], [5], and [6]. We note that Bertsekas and El Baz have shown the convergence of asynchronous relaxation methods under Assumptions 2.1 to 2.4 (see [1]). Satisfactory convergence properties of asynchronous gradient algorithms were shown in [3] using Assumptions 2.1 to 2.3 and 3.2. In the case of standard asynchronous iterations, a speedup resulting from a communication frequency increase is sometimes observed (see in particular [5]); however processors cannot exchange the result of intermediate steps of updating with other processors. The exchange of partial updates can speed up an algorithm particularly in the case of a monotone convergence. Asynchronous iterations with flexible communication which were proposed in [7] allow such exchanges of partial updates. These parallel schemes of computation can be associated with approximate relaxation mappings which are submappings. They were applied to the solution of boundary value problems in [7] and convex network flow problems in [4].

Definition 3.3. Let $T = \{0, 1, 2, ...\}$ be the infinite set of times at which one component of vector p is updated by some processor and T_i be the infinite subset of times at which component p_i is updated such that $T_i \cap T_j = \emptyset$, for all $i, j \in N - \{d\}$, $i \neq j$. An asynchronous iteration with flexible communication associated with an m-continuous submapping $\check{F} : P \to P$ and the starting point $p(0) \in P'$, satisfying $p(0) \leq \underline{F}(p(0))$, is a sequence $\{p(k)\}$ of vectors of P such that for all $i \in N - \{d\}$ we have:

$$p_i^{k+1} = \check{F}_i(\tilde{p}^k), \forall k \in T_i;\; p_i^{k+1} = p_i^k, \forall k \notin T_i, \tag{11}$$

where for $k = 0, 1, ...,$ the vector $\tilde{p}^k \in P$ is such that

$$\tilde{p}^0 = p^0 \text{ and } \tilde{p}^k \in \langle \max\{p^{\rho(k)}, \tilde{p}^l\}, p^k \rangle, k \geq 1, \tag{12}$$

and $\langle p', p" \rangle = \{p \in R^n \mid p' \leq p \leq p"\}$, $\tilde{p}^l$ is the vector used for the latest update of the same component, and the vector $p^{\rho(k)}$ with components $p_i^{\rho_i(k)}$ is introduced in order to model the chaotic behavior of the iterative scheme (we have: $0 \leq \rho_i(k) \leq k$, $\rho_i(k) = k$ if $k \in T_i$, $\rho_i(k)$ is monotonically increasing, and $\lim_{k\to\infty} \rho_i(k) = +\infty$, for all $i \in N - \{d\}$).

In this model, the prices $\tilde{p}_i(k)$ which are exchanged can correspond to new updates of p_i generated by the submapping $\check{F}$ or to partial updates not explicitly labelled by an iteration number and produced by some steps of the iterative process which generates the submapping $\check{F}$ (i.e. $\tilde{p}_i(k)$ can result from computations which are in progress); a restriction being that $\tilde{p}^k$ takes value between $\max\{p^{\rho(k)}, \tilde{p}^l\}$ and p^k. The above model describes asynchronous schemes of computation whereby each processor can have access to the current state of other processors at any time and without any fixed rule. It has been shown in [4] that if $\check{F}$ is an m-continuous submapping on P', $p(0) \in P'$ is a starting point such that $p(0) \leq \underline{F}(p(0))$, and $\underline{p}$ denotes the minimal optimal solution of the problem, then asynchronous iterations with flexible communication $\{p(k)\}$ associated with $\check{F}$ and $p(0)$ are well defined and satisfy $p(k) \uparrow \underline{p}, k \to \infty$.

4. IMPLEMENTATION

4.1. Cray T3E

We have implemented the approximate relaxation algorithm described at the end of subsection 3.1 on a distributed memory supercomputer Cray T3E.

4.1.1. Parallel synchronous implementation

In the case of synchronous implementation, updating and data exchange are made sequentially. Processors communicate the updates at the end of each updating phase. Processors are synchronized by message exchange. Communications are implemented via the MPI Library. More precisely, each processor sequentially sends prices by using the Isend() function and receive data by using Recv() function. The Isend() function is nonblocking and the synchronization is realized via Recv() which is a blocking function. Synchronous communications are implemented according to the rendez-vous principle.

4.1.2. Parallel asynchronous implementation

Asynchronous implementation was mainly performed by using the put() function of the SHMEM library of Cray. The put() function permits one source processor to make a copy from a part of its own memory in the memory of a target processor. We note that the put() function allows to cover communications by computations since processors are not blocked till the completion of the communication. We also note that such a copy cannot be implemented via the first version of MPI.

4.1.3. Parallel asynchronous implementation with flexible communication

Asynchronous implementation with flexible communication was also accomplished by using the SHMEM library. Partial updates obtained after a given number of gradient steps are communicated to the memory of other processors by using the put() function, in such a way that they can be used in new computations. The basic updating procedure of a price p_i, $i \in N - \{d\}$ at iteration $k+1$, $k \in T_i$, is given by the following algorithm where S denotes the subset of steps at which partial updates are communicated.

$q = 0$
$p_i^q \leftarrow p_i$
WHILE $|g_i(p_i^q; p)| > \epsilon'$
  $q = q + 1$
  $p_i^q \leftarrow F_i'(p_i^{q-1}; p)$
  IF $q \in S$ THEN
    DO $j = 1$ FOR Number-of-target-processors
      put(p_i^q,P_j)
    ENDDO
  ENDIF
ENDWHILE
$p_i^{k+1} \leftarrow p_i^q$
DO $j = 1$ FOR Number-of-target-processors
  put(p_i^{k+1},P_j)
ENDDO

Basic updating procedure of price p_i at iteration $k+1$

4.2. Symmetric MultiProcessor

The approximate relaxation algorithm described at the end of subsection 3.1 was also carried out on a Symmetric MultiProcessor, the target being a shared memory SUN Sparcstation 20 with up to four processors. The implementation relies on multithreading techniques and uses the Posix library available with the Solaris 2 system. Parallel algorithms are carried out according to the SPMD model. There are as many threads as processors. Each thread runs the same function on different data and processors. Data exchanges are implemented via direct access to the shared memory.

4.2.1. Parallel synchronous implementation

Synchronous implementation requires that every new iteration of a given thread uses the updates produced at the previous iteration of the other threads. Synchronization is implemented via a synchronization barrier multithreaded library that uses the lock functions of the Posix thread library.

4.2.2. Parallel asynchronous implementation with flexible communication

Only asynchronous iterations with flexible communication were implemented on the multiprocessor workstation since these schemes of computation are natural and particularly well suited to shared memory machines. As a matter of fact the definition of local variables in order to implement standard asynchronous iterations would be artificial for such an architecture. The Posix library is used to create the threads allocated to the different processors. We note that we can hardly ascertain that each thread has been scheduled on the same processor by the Solaris 2 system all along an application.

5. COMPUTATIONAL RESULTS

5.1. Problem and algorithms

We have considered hydraulic network flow problem with the following cost function: $c_{ij}(f_{ij}) = \frac{b}{1+b} . |f_{ij}|^{\frac{1+b}{b}}$. This cost function satisfies Assumptions 2.1 to 2.3 and Assumption 3.2 on a bounded subdomain. We have taken $b = 1.85$; this case corresponds to turbulent flow in pipes (see [4]). The network topology always corresponds to a grid with maximum node degree equal to 32. The number of nodes and arcs are equal to 96 and 1401, respectively. There is one traffic input which is equal to 1 and one traffic output also equal to 1. The accuracy ϵ' of the line search test $\left| g_i(p_i^{q'}; p) \right| \leq \epsilon'$ is equal to 10^{-4}. The size of the gradient steps is equal to 0.0425. Algorithms terminate when the deficit at node d is less than 10^{-2}.

Synchronous and asynchronous algorithms with or without flexible communication associated with the approximate relaxation method presented at the end of subsection 3.1 were carried out on a distributed memory supercomputer Cray T3E by using up to 24 processors and a shared memory SUN Sparcstation 20 with four 100 Mhz processors; they are denoted by S, F, and A, respectively. For all algorithms, the starting point is always the same subsolution: $p_i = 0$ for all $i \in N - \{d\}$. We have balanced the number of nodes assigned to the different processors. Task scheduling is made according to static mode.

5.2. Experimental results

Figure 1 displays the efficiency of S, A, and F (where partial updates are sent every five gradient steps) in function of the number of processors of the Cray T3E. Table 1 gives the time and efficiency of S and F on the SMP.

Table 1
Time and efficiency of S and F on a Sparcstation 20

number of	S		F	
processors	time (s)	efficiency	time (s)	efficiency
1	49.40			
2	28.06	0.88	24.38	1.01
3	19.85	0.83	16.64	0.99
4	15.87	0.78	12.99	0.95

5.3. Analysis of the computational experience

Figure 1 shows that asynchronous implementations were very efficient on the Cray T3E and that F was more efficient than A, except for a small number of processors. Moreover A was more efficient than S. Parallel approximate relaxation algorithms lead to nondeterministic load unbalancing since the components of the submapping are generated by an iterative process, the number of steps of which varies in function of the difficulty of approaching a minimum with respect to a given direction, as a result, idle time due to

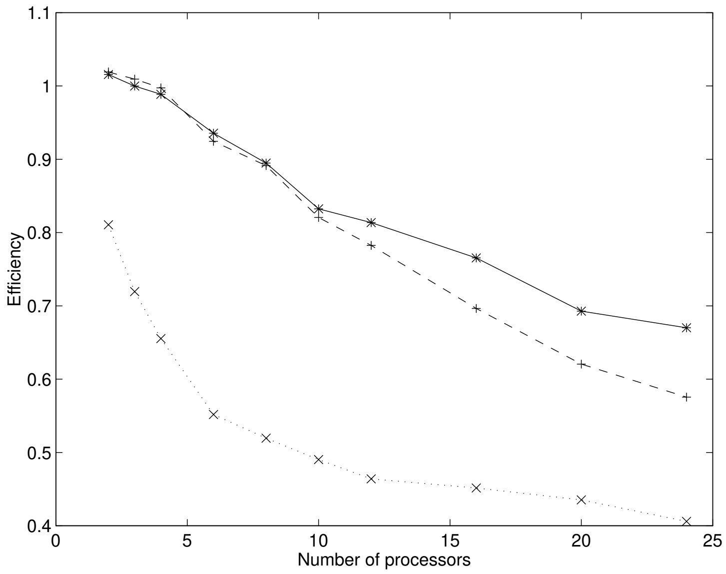

Figure 1. Efficiency of F (solid), A (dashed), and S (dotted) on the CRAY T3E

synchronization may be nonnegligible. We also note that asynchronous implementations gave superlinear speedups for a small number of processors (i.e. for a large granularity), this performance is due to the use of the cache memory of the RISC processor of the Cray T3E.

Table 1 shows that the asynchronous implementation with flexible communication was also very efficient on the Sparcstation 20. The algorithm F was more performant than S for the reasons invoked in the case of the implementation on the Cray T3E.

REFERENCES

1. D. P. Bertsekas, D. El Baz, Distributed asynchronous relaxation methods for convex network flow problems *SIAM Journal on Control and Optimization* **25** (1987) 74–85
2. D. P. Bertsekas, J. Tsitsiklis, *Parallel and Distributed Computation, Numerical Methods* (Prentice Hall, Englewood Cliffs, N.J., 1989)
3. D. El Baz, Asynchronous gradient algorithms for a class of convex separable network flow problems *Computational Optimization and Applications* **5** (1996) 187–205
4. D. El Baz, P. Spiteri, J. C. Miellou, D. Gazen, Asynchronous iterative algorithms with flexible communication for nonlinear network flow problems *Journal of Parallel and Distributed Computing,* **38** (1996) 1–15
5. R. Hiromoto, B. Wienke, R. Brickner, The performance of asynchronous iteration schemes applied to the linearized Boltzmann transport equation *Parallel Computing,* **18** (1992) 241–268
6. J. C. Miellou, Algorithmes de relaxation chaotique à retards *RAIRO,* **R1** (1975) 55–82
7. J.C. Miellou, D. El Baz, P. Spiteri, A new class of asynchronous iterative algorithms with order intervals *LCS Report* **1994-16** *to appear in Mathematics of Computation*