A new implementation of asynchronous iterations with flexible communication on a network of symmetric multi-processor

Mohamed Jarraya, Didier El Baz

LAAS du CNRS, 7, avenue du Colonel Roche, 31077 Toulouse Cedex France

Abstract *In this paper we propose a new implementation of asynchronous iterations with flexible communication on a network of SMP* [1]*. This new implementation uses both a message-passing environment and a thread library. We study the influence of the network bandwidth and the message-passing environment on the performance of synchronous and asynchronous parallel methods applied to nonlinear network flow problems.*

Keywords: Network of SMP, message-passing, threads, asynchronous iterations, nonlinear network flow problems.

1 Introduction

The technological progress in the field of micro-electronics and local area networks always improves the performance of processors and the bandwidth of communication links. A network of machines can be seen as a virtual parallel architecture with distributed memory. Thus, it is important to design efficient parallel algorithms for this type of architecture. In this paper we study the implementation of a class of parallel iterative algorithms : asynchronous iterations with flexible communication whereby computations are carried out in parallel without any order nor synchronization and each processor can have access to the current state of other processors at any time and without any fixed rule.

[1]Symmetric Multi-Processor

Various implementations of asynchronous iterations with flexible communication were proposed in [7] and [3] for a network of single processor workstations by using LANDA, a parallel programming environment with message-passing (see [9]) and a four processors SMP workstation by using the Posix thread library.

In this paper we propose a new implementation of asynchronous iterations with flexible communication on a network of SMP by using the MPICH message-passing environment and POSIX thread library. This implementation permits us to use all the material resources of the parallel architecture. 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.

The theoretical study of asynchronous iterative algorithms with flexible communication developed in [5] and [8] shows that this kind of algorithm has good convergence properties for many numerical problems and is well suited to the type of architecture considered in this paper. Asynchronous iterations with flexible communication are applied here to the solution of nonlinear network flow problems.

Section 2 deals with the primal and dual formulation of nonlinear network flow problems. We present in section 3 asynchronous iterations with flexible communication. Section 4 deals with the implementation of asynchronous iterations with flexible communication; in this section we present and analyze the experimental results on a network of SMP.

2 Convex network flow problem

2.1 Problem formulation

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 $i \in N$, we have $\sum_{i \in N} b_i = 0$. The problem is to minimize total cost subject to a conservation of flow constraint at each node:

$$min \sum_{(i,j)\in A} c_{ij}(f_{ij}), \text{ subject to}$$

$$\sum_{(i,j)\in A} f_{ij} - \sum_{(m,i)\in A} f_{mi} = b_i, \forall i \in N. \quad (2.1)$$

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

$$c^*_{ij}(t_{ij}) = \sup_{f_{ij}}\{t_{ij}.f_{ij} - c_{ij}(f_{ij})\}, \quad (2.2)$$

is real valued, i.e. $-\infty < c^*_{ij}(t_{ij}) < +\infty$ for all real t_{ij}. The last assumption implies that $lim_{|f_{ij}| \to \infty} c_{ij}(f_{ij}) = +\infty$. Therefore the objective function of problem (2.1) has bounded level sets (see [1]). It follows that there exists an optimal solution for problem (2.1) which must be unique in view of the strict convexity of c_{ij}.

2.2 The dual problem

A dual problem for (2.1) is given by

$$\min_{p \in R^n} q(p), \quad (2.3)$$

subject to no constraints on the vector $p = \{p_i | i \in N\}$, where q is the dual functional given by :

$$q(p) = \sum_{(i,j)\in A} c^*_{ij}(p_i - p_j) - \sum_{i \in N} b_i.p_i. \quad (2.4)$$

We refer to p as a price vector and its components as prices. 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:

$$min_{p \in P} q(p). \quad (2.5)$$

In the sequel $g(p)$ will denote the gradient of the dual functional. From (2.4) it follows that the components $g_i(p)$ of $g(p)$ are given by:

$$g_i(p) = \frac{\partial q(p)}{\partial p_i} = \sum_{(i,j)\in A} \nabla c^*_{ij}(p_i - p_j) -$$

$$\sum_{(m,i)\in A} \nabla c^*_{mi}(p_m - p_i) - b_i, i \in N. \quad (2.6)$$

Since $g_i(p)$ is a partial derivative of a differentiable convex function it is continuous and monotonically non-decreasing in the i-th coordinate (see [10]).

2.3 Gradient type method

We have considered a gradient type mapping which is related to the gradient mapping : $F'_i(p) = p_i - \frac{1}{\alpha} g_i(p)$, for all $i \in N - d$ and $p \in P$, where α is a positive constant. Clearly F' is continuous since g is continuous. We assume that c_{ij} is strongly convex with modulus $\frac{1}{\beta}$. It was shown in ([4], Theorem 2.2) that 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.

The gradient type method is defined by $\check{F}_i(p) = p_i^{q'}$, 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 smaller positive integer such that $|g_i(p_i^{q'}; p)| \leq \epsilon'$, ϵ' being a positive constant. For results concerning asynchronous relaxation mappings, the reader is referred to [1] and [2].

3 Asynchronous iterations with flexible communication

Asynchronous iterative algorithms with flexible communication were first proposed in [8] for the solution of boundary value problems in the M-function context. They were applied to the solution of convex network flow problems in [5].

We assume that there is an infinite set of times $T = \{0, 1, 2, ...\}$ at which one component of vector p is updated by some processor. Let 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 the gradient type mapping $\check{F} : P \to P$ and the starting point $p(0) \in P$ which is a subsolution, is a sequence $\{p(k)\}$ of vectors of P such that for $i \in N - \{d\}$, we have:

$$p_i^{k+1} = \check{F}_i(\tilde{p}^k), \forall k \in T_i; p_i^{k+1} = p_i^k, \forall k \notin T_i, \tag{3.1}$$

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

$$\tilde{p}^0 = p^0 \text{ and } \tilde{p}^k \in \langle \max\{p^{\rho(k)}, \tilde{p}^l\}, p^k \rangle, k \geq 1, \tag{3.2}$$

and the order interval $\langle p', p" \rangle = \{p \in R^n \mid p' \leq p \leq p"\}$, $\tilde{p}^l$ is the vector used in the last 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 $i \in N - \{d\}$.

These algorithms allow the use of partial updates arising from computations in progress. Asynchronous iterations with flexible communication defined by (3.1) to (3.2) 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 for ever and old values of each component of the iteration vector are abandoned as the computation progresses. The use of the value of components resulting of intermediary steps of updating presents a particular interest in this context; intuitively, it can speed up the convergence.

4 Implementation on a network of SMP

4.1 Decomposition of a connected graph and assignment to processors of the parallel architecture

We have decomposed the graph into two levels. The first level associates a subset of nodes to a thread which is assigned to one processor of the SMP in order to execute part of the iterative algorithm. We denote by S_i, $i = 1, ..., \alpha$ the subsets, $S_i \subset N$, $S_i \cap S_j = \emptyset$, $\forall i \neq j$ and $\cup_{i=1}^{\alpha} S_i = N$. The second level associates a subset $E_k = \cup_{i=1}^{\beta_k} S_i$, $E_k \subset N$ to a SMP. The total number of threads is $\alpha = \sum_{k=1}^{\gamma} \beta_k$, where β_k is the number of threads created on the k-th SMP and γ is the number of SMP in the network.

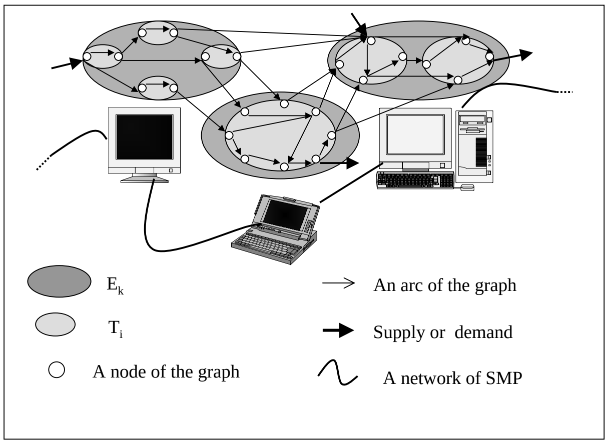

Figure 1: Decomposition of a connected graph on a network of SMP

4.2 Parallel implementation

According to Figure 1, every SMP executes an instance of the program using a multithreaded implementation. We have used both a message passing environment and a thread library in order to implement parallel iterative algorithms on this architecture. We have assigned all the inter SMP communications 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. In order to synchronize threads in a SMP, we have implemented a synchronization barrier by using a mutex and condition variables. At the beginning of the synchronization barrier, all the threads 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 all the waiting threads and sends data to the others machines. The synchronization between distant threads is accomplished by using a blocking receive function.

Asynchronous iterations with flexible communication are implemented as follows. We consider two levels. In the first level which corresponds to a SMP every thread access when needed 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. 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 by distant threads by using non blocking send and receive functions of MPICH. 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. A policy consists for example in sending partial updates after a given number of steps of the gradient type method implemented by this thread. 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. We have chosen a policy which does not saturate the buffer. The receive functions takes into account the last values received by the SMP. The following algorithm given for asynchronous iterations with flexible communication is implemented by a thread created on a SMP with rank i. We denote by N_SMP the number of SMP and we denote by bc_j the border component of a SMP. We denote also by S the subset of steps which the partial updates are communicated. The value of begin_node and end_node enclose the implementation of the gradient type method on a subdomain of the graph.

iteration=0
Termination = FALSE
WHILE (non Termination)
  iteration=iteration+1
  FOR l=begin_node TO end_node
    $q = 0$
    WHILE $(g_l(p_l^q; \tilde{p}) > \epsilon')$
      $q = q + 1$
      IF (communicating thread)
        FOR $j = 1$ TO N_SMP
          RECEIVE bc_j FROM j
        END FOR
      END IF
      update $p_l^q = F_l'(p_l^{q-1}; \tilde{p})$
      IF ($q \in S$ AND communicating thread)
        FOR $j = 1$ TO N_SMP
          SEND p_l^q TO j
        END FOR
      END IF
    FOR $j = 1$ TO N_SMP
      SEND bc_i TO j
    END FOR
  END FOR
  IF $(g_d(p_d^q; \tilde{p}) < \epsilon)$
    Termination =TRUE
  END IF
END WHILE

The component p_l^q is function of partial updates computed by local threads created on the same SMP which are read directly from the shared memory and function of partial updates computed by distant threads created on the others SMP and received by the communicating thread.

4.3 Experimental results

We have considered problems of turbulent flows in pipe with the following cost functions: $c_{ij}(f_{ij}) = |f_{ij}|^{\frac{1+b}{b}}$, where $b = 1.85$. The conjugate gradient of c_{ij} is $\nabla c^*_{ij}(p_i - p_j) = sign(p_i - p_j)\,|p_i - p_j|^{1.85}$. The network topology always corresponds to a grid-like network with 192 nodes, 2937 arcs and node degree $\max_{i \in N} a_i = 32$. We have considered the gradient type method which is implemented with the accuracy $\epsilon' = 10^{-4}$ for the test $\left|g_i(p_i^{q'}; p)\right| \leq \epsilon'$. The stepsize of the gradient and gradient type methods is 0.34. The algorithm terminates when the deficit at node d is less than 10^{-2}.

We have implemented synchronous and asynchronous iterations with flexible communication on a network of Personal Computer with bi-processor Pentium II whereby every processor has a clock frequency of 233 MHz. Two types of networks were considered : an Ethernet network with a bandwidth of 100 Mbits/s and a Myrinet network with a bandwidth of 1 Gbits/s.

We have measured the performances of synchronous and asynchronous iterations with flexible communication on the Ethernet and Myrinet networks. Table 1 and 2 show the elapsed time (T) and the efficiency (E) of the parallel implementations in function of the number of threads running in each SMP; this number is represented by a 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 the respective number of threads that are carried out on the corresponding SMP. In Table 1 and 2 we have displayed the results respectively for the Ethernet and Myrinet networks by using the MPICH environment and a POSIX thread library. We note that we have used the MPICH-gmm version of MPICH on the Myrinet network. MPICH is a portable implementation of MPI that supports a wide range of systems : workstation clusters, shared memory systems and massively parallel supercomputers. We have used a network of four bi-processor machines, so the implementation can be carried out at most on 8 processors. The asynchronous iterations with flexible communication are more efficient than the synchronous iterations for the studied problem. This is due in fact to the non deterministic load balancing of the gradient type method which induces idle time in order to synchronize iterations. The experimental results show that using light weight processes permits one to improve the performance of asynchronous parallel implementation.

The communication time depends on the type of the network and the parallel environment. Table 1 and 2 show that asynchronous iterations are more efficient on the Myrinet Network which has the highest bandwidth than on the Ethernet network. We have measured a bandwidth of 500 Mbits/s and a latency of $27\mu s$ on the Myrinet network and a bandwidth of 65 Mbits/s and a latency of $250\mu s$ on the Ethernet network.

The performance of the parallel implementations on a high bandwidth network is very close to the performance obtained on a supercomputer such as the CRAY T3E (see Table 3, see also [3] and [7]).The 233 MHz Pentium II is clearly faster than the 300 MHz Digital α EV5 processor of the CRAY T3E for floating point calculations. We note however that the regular interconnection architecture of CRAY T3E permits us to obtain better efficiency when the number of processor increase. We note that the results displayed in Table 3 correspond respectively to a synchronous and an asynchronous implementation with flexible communication of the gradient type algorithm presented in section 2.3. Asynchronous implementation with flexible communication was carried out by using the SHMEM library of CRAY. In particular we have used the get() function of the SHMEM library in order to pick up the current value of each component of the iterate vector when needed; we note that this value is not necessarily labelled by an iteration number. The results show the interest of the approach developed in this paper.

	Asynchronous		Synchronous	
Threads	T(sec)	E(%)	T(sec)	E(%)
(1,0,0,0)	199.97			
(2,0,0,0)	100.82	99.2	113.60	88.0
(1,1,0,0)	107.96	92.6	118.38	84.5
(2,1,0,0)	70.22	94.9	80.93	82.4
(2,2,0,0)	52.82	94.6	63.29	78.9
(1,1,1,0)	77.21	86.3	85.55	77.9
(2,1,1,0)	55.76	89.7	64.19	77.9
(2,2,1,0)	43.80	91.3	52.94	75.4
(2,2,2,0)	37.31	89.3	45.50	73.1
(1,1,1,1)	62.32	80.2	68.19	73.1
(2,1,1,1)	47.48	84.2	54.32	73.5
(2,2,1,1)	38.92	85.6	45.86	72.6
(2,2,2,1)	33.26	85.9	40.38	70.7
(2,2,2,2)	30.07	83.1	36.21	69.0

Table 1 : Elapsed time and efficiency of the parallel implementations on the Ethernet network by using MPICH

	Asynchronous		Synchronous	
Threads	T(sec)	E(%)	T(sec)	E(%)
(1,1,0,0)	105.65	94.6	118.45	84.4
(2,1,0,0)	69.22	96.3	81.18	82.1
(2,2,0,0)	51.95	96.2	63.09	79.1
(1,1,1,0)	73.45	90.7	84.94	78.4
(2,1,1,0)	54.16	92.3	63.94	78.2
(2,2,1,0),	42.64	93.8	52.46	76.2
(2,2,2,0)	36.31	91.8	45.03	74.0
(1,1,1,1)	57.18	87.4	67.55	74.0
(2,1,1,1)	44.90	89.0	53.97	74.1
(2,2,1,1)	37.25	89.5	45.34	73.5
(2,2,2,1)	31.49	90.7	39.82	71.7
(2,2,2,2)	28.51	87.6	35.60	70.2

Table 2 : Elapsed time and efficiency of the parallel implementation on the Myrinet network by using MPICH-gmm

	Asynchronous		Synchronous	
process	T(sec)	E(%)	T(sec)	E(%)
1	273.64			
2	138.18	99.0	161.46	84.7
3	99.51	98.6	116.18	78.5
4	69.47	98.5	91.96	74.3
5	56.26	97.3	77.36	70.7
6	46.87	97.3	67.12	67.9
7	40.80	95.8	59.72	65.4
8	35.62	96.0	54.20	63.1

Table 3 : Elapsed time and efficiency of the parallel implementation on the CRAY T3E

5 Conclusions

In this paper we have studied the implementation of asynchronous iterations with flexible communication on a network of SMP. This new implementation uses a message passing environment and a library of thread in order to manage the parallelism between SMP and between local processors of the same SMP. We have shown that the performance of the parallel implementation is very good on a high speed network of SMP by using a message passing environment that gives a good bandwidth and a short latency. Moreover the efficiency of parallel implementation increases when the latency decreases and the bandwidth increases. The numerical experiments show that the performance of asynchronous iterations with flexible communication is better than the performance of synchronous iterations when we have non deterministic load unbalancing. We note that asynchronous iterations with flexible communication are particularly well suited to the solution of network flow problems on a network of SMP. Finally we note the interest of the networks of SMP connected via a Myrinet network.

References

[1] D.P. Bertsekas and D. El Baz. *Distributed asynchronous relaxation methods for con-*

vex network flow problems. SIAM J. Control and Optimization, 25:74–85, 1987.

[2] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computing, Numerical Methods.* Prentice Hall Englewood Cliffs N.J., 1989.

[3] D. El Baz, D. Gazen, M. Jarraya, P. Spiteri and J.C. Miellou. *Flexible communication for parallel asynchronous methods with application to nonlinear optimization problem.* Advances in Parallel Computing: Fundamentals, Applications and New directions, 12:429–436, 1998.

[4] D. El Baz. *Asynchronous gradient algorithms for a class of convex separable problems.* Computational Optimization and Applications, 5:187–205, 1996.

[5] D. El Baz, P. Spiteri, J.C. Miellou and D. Gazen. *Asynchronous iterative algorithms with flexible communication for nonlinear network flow problems.* Journal of Parallel and Distributed Computing,38:1-15 1996.

[6] W. Gropp, E. Ewing lusk. *User's Guide for MPICH, a portable Implementation of MPI, version 1.2.0.* Mathematics and Computer Science Division MIT, 1999

[7] M. Jarraya, D. El Baz and D. Gazen. *Mise en œuvre de méthodes itératives asynchrones avec communication flexible 2, Implémentation sur CRAY T3E, SMP et réseau de stations.* Calculateurs Parallèles Réseaux et Systèmes répartis, 10, 4:439–447, 1998.

[8] J.C. Miellou, D. El Baz and P. Spiteri. *A new class of iterative algorithms with order intervals.* Mathematics of Computation, 67:237–255,1998.

[9] T. Monteil, J.M Garcia, D. Gauchard and O. Brun. *Communication Kernel for high speed networks in the parallel environment LANDA-HSN* 1st IEEE Computer Society International Workshop on Cluster Computing (Iwcc 99), Melbourne, Australie, 63-70, 1999.

[10] P. Tseng and D. Bertsekas. *Relaxation methods for problems with strictly convex separable costs and linear constraints.* Mathematical Programming, 38:303–321, 1987.