

# Tradeoff to minimize extra-computations and stopping criterion tests for parallel iterative schemes

O. Beaumont<sup>1</sup>, E.M. Daoudi<sup>2\*</sup>, N. Maillard<sup>3</sup>, P. Manneback<sup>4</sup>, J.-L. Roch<sup>5\*</sup>

<sup>1</sup> Olivier.Beaumont@labri.fr, <sup>2</sup> mdaoudi@sciences.univ-oujda.ac.ma, <sup>3</sup> nmaillard@inf.ufrgs.br,  
<sup>4</sup> Pierre.Manneback@fpms.ac.be, <sup>5</sup> Jean-Louis.Roch@imag.fr.

## Abstract

Parallel synchronous iterative algorithms are often penalized by global synchronization, due to the cost of stopping tests that are achieved. It is well known that such global synchronizations are expensive for parallel implementations on distributed systems, especially on clusters of processors or computational grids, where the heterogeneity and the number of processors imply a large overhead for this global operation. The aim of this work is to propose a new control technique for the stopping tests, original to the best of our knowledge, which enables to reduce the number of global synchronization near to the optimum, while keeping the number of iterations close to the number performed by standard synchronous algorithm. The main advantage of the proposed technique is that the semantic of the sequential algorithm is not modified, so that convergence is preserved and identical outputs are guaranteed. Our method is based on an amortized technique inspired by Floyd's and Brent's algorithms to detect periodicity in a sequence.

## 1 Introduction

In general, iterative schemes consist in iterating some computation until global convergence is reached, i.e. until some stopping test criterion is fulfilled. Standard algorithm performs the stopping test after each iteration, so that each step of the algorithm can be decomposed into two phases: computation of the body of the iteration and control of the stopping test. Let  $n_{\text{iter}}$  denote the exact number of iterations obtained with the standard algorithm. The execution time  $T_{\text{standard}}$  of the standard algorithm is given by  $T_{\text{standard}} = n_{\text{iter}}(t_{\text{comp}} + t_{\text{sync}})$ , where  $t_{\text{comp}}$  denotes the execution time of one iteration and  $t_{\text{sync}}$  denotes the cost of evaluating a stopping test.

On a parallel architecture, the evaluation of the stopping criterion test is often achieved using global synchronization. Unfortunately, for a parallel implementation on a distributed memory architecture (typically a cluster of processors or a computational grid), it is well known that such global synchronization is extremely expensive, since it requires a global waiting time. Indeed, each processor must wait for the contribution of the slowest processor in order to continue its work.

---

\*This work is supported by the "Comité Mixte Franco-Marocain - Action Intégrée" MA/01/19.

The problem of reducing global synchronization cost without modifying the semantic of the standard sequential algorithm and consequently the behavior of the convergence constitutes the main goal of this paper. An important motivation is the problem of checkpointing of parallel applications: a global synchronization must be performed between all processes before each checkpoint in order to ensure a consistent global state [7]. Checkpointing is used to design fault-tolerant parallel algorithms or to track bugs [9], because an error may occur on a processor and causes the crash of the application at some timestep. If so, the application is restarted from the last global checkpoint, till it possibly crashes again. A major concern is to find a strategy that minimizes the number of checkpoints, each one implying a global synchronization. In the framework of checkpoint/restart, simple amortized techniques are often used, consisting in doubling the time interval between two consecutive checkpoints [9].

In order to reduce the number of global synchronizations, different approaches have been proposed and studied in the literature.

A first approach consists in gathering all global synchronizations of one iteration in order to perform them together. This technique is used e.g. in [1] for Conjugate Gradient method. However, it does not reduce the number of stopping criterion tests and still requires one global synchronization by iteration.

A second approach consists in desynchronizing communications and iterations, resulting in an asynchronous algorithm (see e.g. [3, 4, 6, 8]). Unfortunately, the desynchronization does not preserve the semantic of the standard algorithm and, consequently, it does not ensure convergence or an equivalent number of iterations.

A third approach, called *k-steps* in the sequel, is often used in practice. It consists in performing the stopping tests after each group of  $k$  iterations (i.e. stopping tests are evaluated at iterations  $k, 2k, \dots, q.k, \dots$ ), while the stopping test is not fulfilled. This method performs at most  $\#I = n_{\text{iter}} + k$  iterations and  $\#T = \lceil n_{\text{iter}}/k \rceil$  tests. However, since  $n_{\text{iter}}$  is a priori unknown, the crucial problem is to choose  $k$  in order to achieve a good trade-off between  $\#I$  and  $\#T$ .

In what follows, we concentrate on a general iterative methods, but we restrict to the case where the only global synchronization is the one required by the control of the stopping test. Moreover, we assume that iterations converge to a fixed point. It means that if the stopping test condition is fulfilled at the end of iteration  $n$ , it will also be fulfilled at the end of any following iteration.

In this work, we propose a new control technique, original to the best of our knowledge, which delivers the result of iteration  $n_{\text{iter}}$  after a small number  $\#T = \log_{1+o(1)} n_{\text{iter}}$  of stopping criterion tests, while requiring only the computation of order  $\#I = n_{\text{iter}} + o(n_{\text{iter}})$  iterations. The advantage of our technique lies in the fact that the semantic of the initial algorithm is not modified, and consequently the convergence is ensured and the outputs are identical. Our method is based on an amortized technique inspired from Floyd's and Brent's algorithm to detect periodicity in a sequence [5, 10]. It consists in computing two numbers  $n'_1$  and  $n_1$  such that  $n'_1 \leq n_{\text{iter}} \leq n_1$ , and then determining the exact value of  $n_{\text{iter}}$ . Like in the *k-steps* method, the proposed method performs a limited number of stopping tests, but no longer performed regularly, but rather at steps  $\rho^{f(0)}, \rho^{f(1)}, \dots, \rho^{f(i)} = n_1$ , where  $1 < \rho < 2$  and  $f$  satisfies  $\forall i, f(i) \leq i$ . The choice of  $f$  is based on a tradeoff between  $\#I$  and  $\#T$ . In this paper, we provide asymptotic results for different choices of  $f$ : namely  $f(i) = i^\alpha, 0 < \alpha \leq 1$ ;  $f(i) = \frac{i}{\log i}$  and  $f(i) = \frac{i}{\log^* i}$  ( $\log^* i$  is the iterated logarithm, i.e.  $\log^* i = \min\{j \geq 0, \log^{(j)} i \leq 1\}$ ). We prove

that  $n_{\text{iter}}$  can be determined with little more than  $\#T = \log n_{\text{iter}}$  tests – which is a lower bound for  $\#T$  – while performing an asymptotic optimal number of iterations  $\#I = n_{\text{iter}} + o(n_{\text{iter}})$ .

In Section 2, we present in detail the  $k$ -step method. Sections 3 and 4 are devoted to the presentation of  $\rho$ -amortized and generalized  $\rho$ -amortized methods, which constitute our principal contribution. In section 5, we present some preliminary experiments on a generic iteration method. Finally, some conclusions are drawn.

## 2 $k$ -step method with restart

In  $k$ -step method, stopping tests are no longer performed after each iteration (like in the standard algorithm), but rather after each group of  $k$  successive iterations, for instance at iterations  $k, 2k, \dots, qk, \dots$  where  $q \in \mathbb{N}^*$ . Let  $q$  denote the first index such that the stopping test is fulfilled. Then, this strategy requires  $qk \geq n_{\text{iter}}$  iterations and  $q = \lceil \frac{n_{\text{iter}}}{k} \rceil$  synchronizations. It is clear that the choice of  $k$  depends on various criteria: not only on the method itself, but also on the input data. In [2], P.E. Bernard et al. use such a strategy to control the convergence and the load balance of a dynamic molecular simulation. In this case, the execution stops as soon as the stopping test is fulfilled, i.e. after  $qk \geq n_{\text{iter}}$  iterations. Consequently, the output values of the simulation are different from the results of the standard algorithm, which provides the output values at the end of iteration  $n_{\text{iter}}$ .

Nevertheless, this method can be modified in order to provide the same outputs as the standard algorithm. In this case, the modified method will be called  *$k$ -step method with restart*. More precisely, the  $k$ -step method with restart can be decomposed into two phases.

- During the first phase, the stopping tests are performed after each group of  $k$  successive iterations (just as described above).
- During the second phase, the exact iteration  $n_{\text{iter}}$  is determined by applying the principle of the standard algorithm (stopping tests are then performed after each iteration), but starting from the iteration  $(q - 1)k + 1$ , for which the context of the iteration has been saved. Since  $n_{\text{iter}} \in [(q - 1)k + 1, qk]$ , this phase requires  $(n_{\text{iter}} - (q - 1)k)$  extra iterations and  $(n_{\text{iter}} - (q - 1)k)$  synchronizations.

The two phases lead to a total of

$$qk + (n_{\text{iter}} - (q - 1)k) = n_{\text{iter}} + k \text{ iterations}$$

and

$$q + (n_{\text{iter}} - (q - 1)k) = q + r \text{ synchronizations}$$

where  $1 \leq r < k$ , so that at most  $q + k - 1$  synchronizations are required.

Therefore, the total cost of the  $k$ -step algorithm with restart is given by

$$T_{k\text{-step}} \leq (n_{\text{iter}} + k) \cdot t_{\text{comp}} + \left( \left\lceil \frac{n_{\text{iter}}}{k} \right\rceil + n_{\text{iter}} \bmod k \right) \cdot t_{\text{sync}}.$$

It is clear that the best choice of  $k$  depends on the value of  $n_{\text{iter}}$  which is unknown. If  $k$  is large, a few number of synchronizations is needed in the first phase

but during the second phase, this number becomes large (of order of  $k$ ). In a dual way, a small value of  $k$  leads to a large number of synchronizations during the first phase (of order  $\frac{n_{\text{iter}}}{k}$ ) but during the second phase the number of synchronizations will be reduced. Compared to the standard algorithm, the improvement of the  $k$ -step method with restart is given by:

$$T_{\text{standard}} - T_{k\text{-step}} \geq -k \cdot t_{\text{comp}} + \left[ \left\lfloor \frac{n_{\text{iter}}}{k} \right\rfloor \cdot (k-1) + 1 \right] \cdot t_{\text{sync}}$$

Therefore, if  $k$  is chosen so that  $\frac{k^2}{k-1} \leq n_{\text{iter}} \frac{t_{\text{sync}}}{t_{\text{comp}}}$ , the  $k$ -step method with restart is more efficient (in terms of number of synchronization steps) than the standard one.

### 3 $\rho$ -amortized control

This technique is based on an amortized technique inspired from Floyd's and Brent's algorithm to detect periodicity in a sequence [10]. It consists in two phases. The first phase consists in computing two numbers  $n'_1$  and  $n_1$  such that  $n'_1 \leq n_{\text{iter}} \leq n_1$  and the second phase consists in determining the exact value of  $n_{\text{iter}}$ .

Like the  $k$ -step technique with restart, this method performs a limited number of stopping tests during the first phase. However tests are no longer performed at regular steps, but rather at steps  $\rho^0, \rho^1, \dots, \rho^i, \dots$ . Let  $\rho^{k_1} = n_1$  denote the first index  $k_1$  such that the stopping test is fulfilled. Then,  $\frac{n_1}{\rho} = \rho^{k_1-1} < n_{\text{iter}} \leq \rho^{k_1} = n_1$ . Therefore, this phase requires  $\rho^{k_1}$  iterations and  $k_1$  tests. During the second phase, we apply a recursive technique or a dichotomic search for determining the exact value  $n_{\text{iter}}$ .

#### 3.1 Recursive $\rho$ -amortized control

The iteration  $n_{\text{iter}}$ , is obtained by applying recursively the same process as for the first phase, but starting from the iteration  $\frac{n_1}{\rho} + 1 = \rho^{k_1-1} + 1$ , for which context has been saved. Stopping tests are performed at iterations  $\frac{n_1}{\rho} + 1, \frac{n_1}{\rho} + \rho^1, \dots, \frac{n_1}{\rho} + \rho^{k_2}$ , where  $k_2$  is the smallest index such that the stopping test is fulfilled,  $k_2 \leq \lceil \log(n_1/\rho) \rceil$ . Then,

$$\frac{n_1}{\rho} < \frac{n_1}{\rho} + \rho^{k_2-1} < n_{\text{iter}} \leq \frac{n_1}{\rho} + \rho^{k_2} \leq n_1.$$

Let  $n_2 = \rho^{k_2}$ . We re-apply the same process, starting from the iteration  $\frac{n_1}{\rho} + \frac{n_2}{\rho} + 1$ . We recursively determine a sequence of indices  $k_l$ , with  $l \leq \log_{\rho} n_{\text{iter}}$ , such that

$$\sum_{i=1}^{l-1} \rho^{k_i-1} + \rho^{k_l-1} < n_{\text{iter}} \leq \sum_{i=1}^{l-1} \rho^{k_i-1} + \rho^{k_l}.$$

Therefore, by construction of sequence of indices  $k_1, \dots, k_l$ ,

$$n_{\text{iter}} = \sum_{i=1}^{\log_{\rho} n_{\text{iter}}} \rho^{k_i-1}.$$

- The total number  $S$  of synchronizations is given by  $S = k_1 + k_2 + \dots + k_l$ . Since  $k_i \leq \log_\rho n_{\text{iter}}$ ,

$$S = \sum_{i=1}^l k_i \leq \lceil \log_\rho(n_{\text{iter}}) \rceil^2.$$

- The total number of iterations  $N$  (in the worst case) is given by

$$N = \sum_{i=1}^l \rho^{k_i} = \rho \cdot \sum_{i=1}^l \rho^{k_i-1} \leq \rho \cdot n_{\text{iter}}.$$

Finally, the total executing time  $T_{\rho\text{-amortized}}$  of the  $\rho$ -amortized control technique is given by

$$T_{\rho\text{-amortized}} \leq \rho \cdot n_{\text{iter}} \cdot t_{\text{comp}} + \lceil \log_\rho(n_{\text{iter}}) \rceil^2 \cdot t_{\text{sync}}.$$

Therefore, the recursive  $\rho$ -amortized method allows to decrease exponentially the number of global synchronizations, while increasing execution time of order  $\rho n_{\text{iter}}$ . In what follows, we provide choices for  $\rho$  such that the proposed algorithm will be more efficient than the standard one. From previous equations,

$$\frac{T_{\rho\text{-amortized}}}{T_{\text{standard}}} \leq \rho \frac{t_{\text{cal}}}{t_{\text{cal}} + t_{\text{sync}}} + \frac{\log_\rho^2(n_{\text{iter}})}{n_{\text{iter}}} \frac{t_{\text{sync}}}{t_{\text{cal}} + t_{\text{sync}}}.$$

Thus, if  $\log_\rho^2(n_{\text{iter}}) < n_{\text{iter}}$  and  $t_{\text{sync}} > t_{\text{comp}}$ , then, by choosing  $\rho < 1 + \frac{t_{\text{sync}}}{t_{\text{cal}}}$ , the  $\rho$ -amortized method is more efficient than the standard method. Moreover, the performance of the  $\rho$ -amortized method is also better than the performance of the  $k$ -step algorithm with restart.

### 3.2 Dichotomic $\rho$ -amortized control

In this method, the value  $n_{\text{iter}}$ , that lies in the search interval  $[\rho^{k_1-1}, \rho^{k_1}]$ , is obtained using dichotomic technique which consists in  $\log(\rho^{k_1} - \rho^{k_1-1})$  steps and proceeds in the following manner.

**First step:**

- compute all iterations  $\rho^{k_1-1} + 1, \rho^{k_1-1} + 2, \dots, mid$ , where  $mid$  is the middle of the search interval  $[\rho^{k_1-1} + 1, \rho^{k_1}]$ ,
- perform the stopping test only for iteration  $mid$ . The iteration  $n_{\text{iter}}$  lies in one of the two following reduced search intervals:  $[\rho^{k_1-1} + 1, mid]$  or  $[mid + 1, \rho^{k_1}]$ .

**For the remaining steps,** we apply recursively the same processes, using the new computed search interval until the length of the search interval is reduced to 1.

Since, at each step the search interval is divided by 2,

- the number of stopping tests is  $\log(\rho^{k_1} - \rho^{k_1-1}) < \log \rho^{k_1} = k_1 \log \rho$ ,
- the number of iterations is bounded by  $2(\rho^{k_1} - \rho^{k_1-1})$ .

From phases 1 and 2, we deduce that

- the overall number  $\#I$  of iterations satisfies

$$\#I \leq \rho^{k_1} + 2(\rho^{k_1} - \rho^{k_1-1}) = (3\rho - 2) \cdot \rho^{k_1-1} < (3\rho - 2)n_{\text{iter}},$$

- the overall number  $\#T$  of synchronizations is bounded by

$$\#T < k_1 + (k_1) \log \rho = (\log \rho + 1)(k_1)$$

Since  $\rho^{k_1-1} < n_{\text{iter}} \leq \rho^{k_1}$  by construction, the number  $S$  of synchronizations is bounded by:

$$S < (\log \rho + 1)(\log n_{\text{iter}} + 1).$$

## 4 Generalized $\rho$ -amortized control

### 4.1 General considerations

The scheme we propose in this section is based on the  $\rho$ -amortized method. The main difference is that tests are no longer performed at steps  $\rho^0, \rho^1, \dots, \rho^i, \dots$  but rather at steps  $\rho^{f(0)}, \rho^{f(1)}, \dots, \rho^{f(i)}, \dots$  where  $1 < \rho < 2$  and  $f$  satisfies  $\forall i, f(i) \leq i$ . Our aim is to determine  $f$  so that we can find the exact value of  $n_{\text{iter}}$  with the best tradeoff between the:

- synchronization cost, i.e. a number of stopping tests as close as possible to  $\theta(\log n_{\text{iter}})$ ;
- iteration cost, i.e. a number of iterations as close as possible to  $n_{\text{iter}}$ .

The proposed scheme is decomposed into two phases:

- during the first phase, the stopping tests are performed after iterations  $\rho^{f(0)}, \rho^{f(1)}, \dots, \rho^{f(i)}, \dots$ . We stop the iterations at the first index  $k$  such that the stopping test is first fulfilled. Then,

$$\rho^{f(k)} < n_{\text{iter}} \leq \rho^{f(k+1)}.$$

At the end of this step,  $S_c = (k + 1)$  tests have been performed and at most  $\rho^{f(k+1)}$  iterations have been computed, so at most  $\rho^{f(k+1)} - \rho^{f(k)}$  extra iterations have been computed.

- The second phase is devoted to determine  $n_{\text{iter}}$ , starting from iteration  $\rho^{f(k)} + 1$ . In order to determine the exact value of  $n_{\text{iter}}$ , we perform a dichotomic search of  $n_{\text{iter}}$  between  $\rho^{f(k)}$  and  $\rho^{f(k+1)}$ . At each step of the dichotomic search, we need to recompute all the iterates between the lower bound and the middle of the search interval, but we compute the stopping criterion for the middle value only. Thus, the overall dichotomic search induces at most

$$R_d = 2(\rho^{f(k+1)} - \rho^{f(k)})$$

extra computations and

$$S_d = \log_2 \left( \rho^{f(k+1)} - \rho^{f(k)} \right) < \lceil \log_2 n_{\text{iter}} \rceil$$

tests, since  $\rho < 2$  and  $\forall i, f(i) \leq i$ .

Finally,

- the overall number  $R$  of extra iteration computations is bounded by  $3(\rho^{f(k+1)} - \rho^{f(k)})$  and therefore  $R = \theta(\rho^{f(k+1)} - \rho^{f(k)})$ . In all the sequel, upper bounds are provided on  $R$  that are close to  $n_{\text{iter}}$ .
- The overall number of stopping tests  $S$  is  $S = S_c + S_d \leq k + 1 + \lceil \log_2 n_{\text{iter}} \rceil$  tests; in the sequel, in order to bound  $S$ , we study the number  $S_c$  of tests performed during the first phase, before the dichotomic search.

In what follows, we provide asymptotic results for  $R$  and  $S_c$  associated to different choices of  $f$ , namely  $f(k) = k^\alpha$ ,  $0 < \alpha < 1$ ,  $f(k) = \frac{k}{\log k}$  and  $f(k) = \frac{k}{\log^* k}$ . We prove that it is possible to determine  $n_{\text{iter}}$  with little more than  $\log n_{\text{iter}}$  tests, while performing of order  $o(n_{\text{iter}})$  extra iteration computations.

## 4.2 $f(i) = i^\alpha$ , where $0 < \alpha < 1$

**Lemma 1** *The number  $S_c$  of tests during the first phase is lower than  $S_c \leq 1 + (\log_\rho n_{\text{iter}})^{\frac{1}{\alpha}}$  tests.*

**Proof:** By construction,

$$\begin{aligned} \rho^{k^\alpha} < n_{\text{iter}} \leq \rho^{(k+1)^\alpha} &\Rightarrow k^\alpha < \log_\rho n_{\text{iter}} \leq (k+1)^\alpha \\ &\Rightarrow k < (\log_\rho n_{\text{iter}})^{\frac{1}{\alpha}} \leq k+1, \end{aligned}$$

what achieves the proof of Lemma 1.

**Lemma 2** *The number  $R$  of extra computations is of order  $o(n_{\text{iter}})$ .*

**Proof:**

$$\begin{aligned} \rho^{f(k+1)} - \rho^{f(k)} &= \rho^{(k+1)^\alpha} - \rho^{k^\alpha} \\ &= \rho^{k^\alpha} \left( \rho^{(k+1)^\alpha - k^\alpha} - 1 \right) \\ &= \rho^{k^\alpha} \left( \log(\rho) \alpha k^{\alpha-1} + o(k^{\alpha-1}) \right). \end{aligned}$$

By construction,  $\rho^{k^\alpha} < n_{\text{iter}}$  and  $\log(\rho) \alpha k^{\alpha-1} = o(1)$  (since  $\alpha < 1$ ). Therefore,  $\rho^{f(k+1)} - \rho^{f(k)} = o(n_{\text{iter}})$ , what achieves the proof of Lemma 2.

Therefore, if  $f(k) = k^\alpha$ , the number of stopping criterion tests is poly-logarithmic in  $n_{\text{iter}}$ , whereas the overall number of computed iterations is of order  $(n_{\text{iter}} + o(n_{\text{iter}}))$ .

## 4.3 $f(i) = \frac{i}{\log i}$

**Lemma 3** *The number  $S_c$  of tests during the first phase is lower than  $S_c \leq \log n_{\text{iter}} \cdot \log(\log n_{\text{iter}}) \left( 1 + \mathcal{O}\left(\frac{1}{\log \log \log n_{\text{iter}}}\right) \right)$ .*

**Proof:** By construction,

$$\rho^{\frac{k}{\log k}} < n_{\text{iter}} \leq \rho^{\frac{k+1}{\log(k+1)}}.$$

Therefore,

$$\begin{aligned}
\frac{k}{\log k} < \log_\rho n_{\text{iter}} \leq \frac{k+1}{\log(k+1)} &\Rightarrow k < \log_\rho n_{\text{iter}} \log k \\
&\Rightarrow k < \log_\rho n_{\text{iter}} \cdot \log(\log_\rho n_{\text{iter}} \cdot \log k) \\
&\Rightarrow k < \log_\rho n_{\text{iter}} \cdot \log \log_\rho n_{\text{iter}} + \log_\rho n_{\text{iter}} \cdot \log \log k
\end{aligned}$$

Since  $k = \mathcal{O}(\log n_{\text{iter}})$ , we deduce that

$$k < \log_\rho n_{\text{iter}} \log \log_\rho n_{\text{iter}} \cdot \left(1 + \mathcal{O}\left(\frac{1}{\log \log \log n_{\text{iter}}}\right)\right) \sim_{n_{\text{iter}} \rightarrow \infty} \log_\rho n_{\text{iter}} \log \log_\rho n_{\text{iter}}$$

which achieves the proof of Lemma 3.

**Lemma 4** *The number  $R$  of extra computations is of order  $o(n_{\text{iter}})$ .*

**Proof:**

$$\begin{aligned}
\rho^{f(k+1)} - \rho^{f(k)} &= \rho^{\frac{k+1}{\log(k+1)}} - \rho^{\frac{k}{\log k}} \\
&= \rho^{\frac{k}{\log k}} \left( \rho^{\frac{k+1}{\log(k+1)} - \frac{k}{\log k}} - 1 \right),
\end{aligned}$$

and since

$$\begin{aligned}
\frac{k+1}{\log(k+1)} - \frac{k}{\log k} &< \frac{1}{\log k} \\
\rho^{f(k+1)} - \rho^{f(k)} &< \rho^{\frac{k}{\log k}} \left( \rho^{\frac{1}{\log k}} - 1 \right)
\end{aligned}$$

Now, since  $1 < \rho < 2$  and  $\log k > 1$ , when  $n_{\text{iter}} \rightarrow \infty$  we have:

$$\rho^{\frac{1}{\log k}} - 1 < \frac{\log \rho}{\log k} + \left(\frac{\log \rho}{\log k}\right)^2 < \frac{\rho - 1}{\log k} < \frac{1}{\log k}.$$

Moreover,  $\rho^{f(k)} = \rho^{\frac{k}{\log k}} < n_{\text{iter}}$  by construction; thus,  $\rho^{\frac{k+1}{\log(k+1)}} - \rho^{\frac{k}{\log k}} < \frac{n_{\text{iter}}}{\log k} = o(n_{\text{iter}})$ , which achieves the proof of Lemma 4.

Therefore, if  $f(k) = \frac{k}{\log k}$ , the number of stopping tests is asymptotically close to  $\log n_{\text{iter}} \log \log n_{\text{iter}}$ , whereas the overall number of computed iterations is  $(n_{\text{iter}} + o(n_{\text{iter}})) \sim n_{\text{iter}}$ .

#### 4.4 $f(i) = \frac{i}{\log^* i}$

**Lemma 5** *The number  $S_c$  of tests during the first phase is lower than  $S_c \leq 1 + \log n_{\text{iter}}(1 + \log^* \log_\rho n_{\text{iter}})$ .*

**Proof:** By construction,

$$\frac{k}{\log^* k} < \log_\rho n_{\text{iter}} \leq \frac{k+1}{\log^*(k+1)} < \frac{k}{\log^* k} \left(1 + \frac{1}{k}\right).$$



Moreover,

$$k < \exp\left(\frac{k}{\log^*(k)}\right) \Rightarrow \log^* k < \log^*\left(\frac{k}{\log^*(k)}\right) + 1$$

and

$$k > \left(\frac{k}{\log^*(k)}\right) \left(1 + \frac{1}{k}\right) \Rightarrow \log^* k > \log^*\left(\frac{k}{\log^*(k)}\right) \left(1 + \frac{1}{k}\right).$$

Therefore,  $\log^* k - 1 < \log^*(\log_\rho n_{\text{iter}}) < \log^* k$  and, since

$$k < \log^* k \log_\rho n_{\text{iter}} < k + 1,$$

$$k < (\log_\rho n_{\text{iter}}) (1 + \log^*(\log_\rho n_{\text{iter}})) < k + 1.$$

This achieves the proof Lemma 5.

**Lemma 6** *The number  $R$  of extra computations is of order  $o(n_{\text{iter}})$ .*

$$\begin{aligned} \rho^{f(k+1)} - \rho^{f(k)} &= \rho^{\frac{k+1}{\log^*(k+1)}} - \rho^{\frac{k}{\log^*(k)}} \\ &= \rho^{\frac{k}{\log^*(k)}} \left( \rho^{\frac{k+1}{\log^*(k+1)} - \frac{k}{\log^*(k)}} - 1 \right) \\ &< \rho^{\frac{k}{\log^*(k)}} \left( \rho^{\frac{1}{\log^* k}} - 1 \right). \end{aligned}$$

Now, since  $1 < \rho < 2$  and  $\log^* k > 1$ , we have:

$$\rho^{\frac{1}{\log^* k}} - 1 < \frac{\log \rho}{\log^* k} + \left( \frac{\log \rho}{\log^* k} \right)^2 < \frac{\rho - 1}{\log^* k} < \frac{1}{\log^* k}.$$

Moreover,  $\rho^{f(k)} = \rho^{\frac{k}{\log^* k}} < n_{\text{iter}}$ ; thus,  $\rho^{\frac{k+1}{\log^*(k+1)}} - \rho^{\frac{k}{\log^* k}} < \frac{n_{\text{iter}}}{\log^* k} = o(n_{\text{iter}})$ , which achieves the proof of Lemma 6.

Therefore, if  $f(k) = \frac{k}{\log^* k}$ , the number of stopping tests is asymptotically bounded by  $(\log n_{\text{iter}} \log^* \log n_{\text{iter}})$ , whereas the overall number of computed iterations is  $n_{\text{iter}} + o(n_{\text{iter}})$ . It is therefore possible to achieve a number of tests close to the lower bound  $\log n_{\text{iter}}$  while performing an asymptotically optimal number  $n_{\text{iter}}$  of iteration computations.

## 4.5 Summary of theoretical results

Table 1 summarizes the theoretical complexity results we have obtained in terms of the number of iterations and number of synchronizations, for different choices of the function  $f$ , where  $1 < \rho < 2$ ;  $n_{\text{iter}}$  denotes the number of iterations required by the standard algorithm which is *a priori* unknown. The overall number of synchronizations  $S$  verifies  $S \leq S_c + \log_2 n_{\text{iter}} \sim \log_2 n_{\text{iter}}$ .

This table shows that  $n_{\text{iter}}$  can be determined with little more than  $\log n_{\text{iter}}$  synchronizations, which is a lower bound of the total number of tests, while performing an asymptotic optimal number of iterations  $n_{\text{iter}} + o(n_{\text{iter}})$ .

Functions	Iterations	Synchronizations $S_c$ (first phase)	Synchronizations $S$ (both phases)
$f(i) = i^\alpha, 1 < \alpha < 1$	$n_{\text{iter}} + o(n_{\text{iter}})$	$\sim (\log_\rho n_{\text{iter}})^{\frac{1}{\alpha}}$	$\sim \log_2 n_{\text{iter}}$
$f(i) = \frac{i}{\log i}$	$n_{\text{iter}} + o(n_{\text{iter}})$	$\sim \log n_{\text{iter}} \log \log n_{\text{iter}}$	$\sim \log_2 n_{\text{iter}}$
$f(i) = \frac{i}{\log^* i}$	$n_{\text{iter}} + o(n_{\text{iter}})$	$\sim \log n_{\text{iter}} \log^* \log n_{\text{iter}}$	$\sim \log_2 n_{\text{iter}}$

Table 1: *Asymptotic cost of the three algorithms.*

## 5 Experiments on a generic parallel iteration scheme

In order to test our theoretical results, we have sketched a generic parallel iteration scheme. To analyze the impact of a decrease in the number of global synchronizations, we consider for the sake of simplicity that communications occur only for the evaluation of the stopping criterion. For  $1 \leq i \leq p$ , let  $X_i$  be the random variable denoting the duration of the computations performed by processor  $P_i$  at each iteration step. Let  $X_i^t$  denote the duration of iteration  $t$  on processor  $P_i$ . We assume that all  $X_i^t$  are independent and follow the same distribution law with expectation  $E(X)$ . If the stopping criterion is evaluated after each step, then the expectation of the duration of each step, from a given step  $\tau$ , is

$$E\left(\max_{i=1}^p X_i^\tau\right).$$

But, if the synchronization is performed only after  $K$  steps only, then the expectation of the amortized duration of one step is given by

$$\frac{1}{K} E\left(\max_{i=1}^p \sum_{t=\tau+1}^{\tau+K} X_i^t\right) = E\left(\max_{i=1}^p \frac{1}{K} \sum_{t=\tau+1}^{\tau+K} X_i^t\right);$$

which is always smaller than  $E(\max_{i=1}^p X_i^t)$ , from Jensen's inequality. Thus, increasing the number  $K$  of iterations between two synchronizations will always be better than synchronizing the processes at each step.

Moreover, in previous  $\rho$ -amortized algorithms,  $K$  increases and becomes very large when  $n_{\text{iter}}$  increases. Also, using the strong law of large numbers,

$$\lim_{K \rightarrow \infty} \frac{1}{K} \sum_{t=\tau+1}^{\tau+K} X_i^t = E(X).$$

Thus, for large values of  $K$ , the duration of each step tends to be equal to the mean value, and the effects of variations due to external effects such as hardware tend to diminish. This property is a major advantage for  $\rho$ -amortized methods in cases where the duration of a local iteration on a single processor may vary.

To illustrate this, the following experiment has been performed. A MPI program is performing  $n_{\text{iter}}$  iterations. At each iteration, each process makes a Lapack `dgesv` call to solve a  $N \times N$  system of linear equations. After the resolution, the process  $i = 1, \dots, p$  exchanges the result vector with its neighbors  $i - 1, i + 1$  (with exception of the processes 0 and  $p - 1$ , of course, which only send their vectors to

the processes 1 and  $p - 2$ , respectively). After the communication phase, a global synchronization is done through a `MPI_Allgather`, a MPI collective communication that broadcasts to all processes the fusion of pieces of data collected on all other ones. Such an iteration is meant to be representative of a real numerical code, for instance for domain decomposition computations, where a local computation such as a LU factorization is performed, followed by an exchange of the domain intersections between the processes, and eventually by a test of the convergence of the norm of the global solution.

Two implementations of the global synchronization have been made. The first one, called “Global Allgather”, broadcasts to all the processes the total solution vector (thus, the volume of the total communication is  $\Theta(pN)$ ). The other implementation first computes locally the norm of each part of the solution vector, and then makes the broadcast of the partial norms (thus with a cost of  $\Theta(p)$ ). We call it “Reduced Allgather”. This second solution is more realistic, yet is only valid when the norm of the global vector may be obtained as a linear combination of the partial results.

Based on this MPI template code, it is simple to test and time different schemes for the iterative control of the convergence. We present here the measures made on the standard control algorithm, on the  $k$ -step one, and on the generalized  $\rho$ -amortized method, obtained with  $f(i) = i/\log i$  (with the value  $\rho$  set to 1.95, and  $k = 10$ ). This includes, when it is necessary, the dichotomic steps. All the necessary operations have been programmed, as in a realistic implementation (copies of the solution for the restarts, norm computation, etc...). The dichotomic search has been programmed such as to perform the worst case number of iterations.

The testing platform is the INRIA’s I-Cluster2. It is constituted of 20 nodes Itanium-2, all chosen between the 100 available in order to be connected by a unique switch Fast Ethernet. Notice that this choice, obviously, minimizes the time required for a global synchronization.

We have design a limit experimentation where the  $k$ step method with a small  $k$  can be better than the  $\rho$ -amortized ones: the number of processor (20) is small and the number  $n_{\text{iter}} = 1000$  of iterations is small too. On this platform, the size of the matrix for the Lapack call has been fixed to  $N = 300$ , and the convergence value  $n_{\text{iter}} = 1000$  is imposed. Typically, such a program runs in about 3 minutes on 20 I-Cluster2 nodes. Each version of the control algorithm has been run 10 times, in order to obtain statistically significant results.

For each run, the total number of iterations  $R$  has been measured, as well as the number of synchronization and the time that they lasted. Of course, the timing results highly depend on the hardware and software implementation. Table 2 presents the results obtained on each algorithm, for each one of the two implemented Allgatherers.

Of course, the number of iterations and synchronizations is equal with both Allgatherers. The standard deviations are not presented in the table, because they are insignificant. These measures show, without surprise, the number of synchronizations that are spared by the  $k$ -step and  $i/\log i$  algorithms. Due to the very low value for  $k$ , the  $k$ -step algorithm appears as specially interesting. Regarding timings, as far as 38 seconds are gained, on a total of some 180 (*i.e.* 20%), due to a reduced use of synchronizations, in the case of the Global Allgather. With an optimized Reduced Allgather, the gain is less (some 2 seconds), yet visible.

These measures, even for a template application running on few nodes, may be

Global Allgather			
	Standard Algorithm	$k$ -step algorithm	$i/\log i$ algorithm
$n_{\text{iter}}$	1000	1000	100
$R$	1000	1047	1078
$S$	1000	28	72
$S_c$	1000	?	?
$S_d$	0	?	?
$T_{\text{iter}}$ (mean)	185.7 s	181.4 s	188.2 s
$T_{\text{synch}}$ (mean)	40.2 s	1.0 s	2.1 s
$T_{\text{iter}}/n_{\text{iter}}$ (mean)	185.7 ms	181.4 ms	188.2 ms
$T_{\text{synch}}/n_{\text{iter}}$ (mean)	40.2 ms	1.0 ms	2.1 ms
Reduced Allgather			
	Standard Algorithm	$k$ -step algorithm	$i/\log i$ algorithm
$T_{\text{synch}}$ (mean)	2.8 s	0.9 s	2.8 s

Table 2: Number of iterations, of synchronizations, and associated cumulated time for each, using the Global and the Reduced Allgather.

seen as promising. The mere increase in the number of nodes or, even simpler, the same runs, but on 20 nodes that would not be interconnected by a complete network, would simply amplify the gain that is already measured with this limited benchmark.

## 6 Conclusion

Minimizing the number of global synchronizations in a distributed computation enables to increase efficiency of synchronous algorithms. In this paper, we focus on iterative methods where a global stopping criterion is evaluated at each step. We analyzed various strategies in order to decrease the number of synchronizations while ensuring that the computed output result is the same than the one delivered by the sequential method. We propose a new control technique named  $\rho$ -amortized, original to the best of our knowledge, where the global test is performed only after iteration  $\rho^{f(i)}$  with  $1 < \rho < 2$  ( $f$  being a non-decreasing function). Analyzing various trade-offs for  $\rho$  and  $f$ , we prove that, when the total number of iterations  $n_{\text{iter}}$  performed by the sequential method is unknown, the whole number of global synchronizations may be reduced close to the lower theoretical bound ( $\log n_{\text{iter}}$ ) on the overall number of tests, while performing an asymptotically optimal number of iterations ( $n_{\text{iter}} + o(n_{\text{iter}})$ ).

Experiments have been run on a synthetic benchmark, meant as a template for scientific computing based on iterative methods. Running on a small number of nodes, using a complete interconnecting network, they have shown a clear gain in time spent doing synchronization. This implementation, as well as theoretical considerations, show that the proposed methods are robust and may be implemented efficiently.

## References

- [1] Baserman (A.), Reichel (B.) et Scheltoff (C.) - Preconditioned CG methods for sparse matrices on massively parallel machines - *Parallel Computing* 23 (1997).
- [2] Bernard (Pierre-Eric), Gautier (Thierry) et Trystram (Denis) - Large Scale Simulation of Parallel Molecular Dynamics - Proceedings of Second Merged Symposium IPPS/SPDP 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, San Juan, Puerto Rico, avril 1999.
- [3] Bahi (J.), Miellou (J.-C.), and Rhofir (K.)- Asynchronous multisplitting methods for nonlinear fixed point problems - *Numerical Algorithms* 15 - pp. 315-345,1997.
- [4] Bertsekas (Dimitri D.) et Tsitsiklis (John N.) - Parallel and distributed computation, numerical methods - Prentice-Hall, Englewood Cliffs N.J. 1989.
- [5] Cormen (T.H.), Leiserson (C.E.), Rivest (R.L.) and Stein (C.) - Introduction to Algorithms - Second Edition, McGraw-Hill, 2001.
- [6] El Baz (D.), Spiteri (P.), Miellou (J.-C) and Gazen (D.) - Asynchronous Iterative Algorithms with Flexible Communication for Nonlinear Network Flow Problems. *J. Parallel Distrib. Comput.* 38 - pp 1-15 - 1996
- [7] Elnozahy (E.N.), Alvisi (L.), Wang (Y.-M.) and Johnson (D.B.) - A survey of rollback-recovery protocols in message-passing systems - *ACM Computing Survey* 34(3), pp 375–408, 2002.
- [8] Frommer (A.), Szyld (D.B.) - On asynchronous iterations - *J. Comp. Appl. Math.* 123 - pp 201–216 - 2000.
- [9] Kalaiselvi, (S.) et Rajaraman (V.) - A survey of checkpointing algorithms for parallel and distributed computers - *Sadhana* 25(5), pp 489-510 - 2000
- [10] Knuth (Donald E.) - The Art of Computer Programming - Vol. 2 - 3rd edition - §3.1 p.7 - Addison Wesley Longman - 1997.
- [11] Maillard (Nicolas) - Calcul haut-performance et mécanique quantique: analyse des ordonnancements en temps et en mémoire - Thèse, ID-IMAG, 19 Novembre 2001.