Distributed Convergence Detection Based on Global Residual Error Under Asynchronous Iterations

Frédéric Magoulès, *Member, IEEE* and Guillaume Gbikpi-Benissan

Abstract—Convergence of classical parallel iterations is detected by performing a reduction operation at each iteration in order to compute a residual error relative to a potential solution vector. To efficiently run asynchronous iterations, blocking communication requests are avoided, which makes it hard to isolate and handle any global vector. While some termination protocols were proposed for asynchronous iterations, only very few of them are based on global residual computation and guarantee effective convergence. But the most effective and efficient existing solutions feature two reduction operations, which constitutes an important factor of termination delay. In this paper, we present new, non-intrusive, protocols to compute a residual error under asynchronous iterations, requiring only one reduction operation. Various communication models show that some heuristics can even be introduced and formally evaluated. Extensive experiments with up to 5,600 processor cores confirm the practical effectiveness and efficiency of our approach.

Index Terms—Asynchronous iterations, convergence detection, global residual, distributed snapshot, parallel computing

1 Introduction

Reducing the impact of communication on the efficiency of a parallel computation is often achieved by optimizing a graph of data dependency between the computing units. However, for iterative methods, another important aspect to take into account is how often data transfers occur. Indeed, in a classical parallel procedure, the computation has to pause each time a remote data is needed due to dependency. This can result in a notable global slowdown of the procedure, according to the properties of the underlying communication platform. Asynchronous iterations are thus interesting to minimize the impact of this second aspect. By not requiring synchronization at each iteration, asynchronous methods avoid idling while waiting for a data exchange, thus ideally reduce the wasted time. This kind of iterative methods was first experienced in [1] as part of a study on the simulation of parallel processing. It follows from the asynchronism that the components of a global vector are iteratively computed without the precedence order ensured by synchronous iterations, which obviously introduces some convergence issues. A first convergence result was established in [2] for the solution of algebraic linear systems, then non linear systems were investigated as well (see, e.g., [3], [4]). Performance comparison against synchronous iterations was first conducted on a parallel computer in [5]. Many studies confirmed the efficiency of asynchronous methods in various mathematical fields such as the obstacle problem (see, e.g., [6], [7], [8]), dynamic programming (see, e.g., [9]), optimization and flow problems (see, e.g., [10], [11]), partial differential equations (see, e.g., [12], [13]), differential-algebraic systems (see, e.g., [14]) Markov chains and optimal control (see, e.g., [15]).

Nowadays, asynchronous parallel algorithms are particularly investigated for taking full advantage of massively parallel architectures and largely distributed platforms. Indeed, in such environments, the most part of the efficiency of parallel algorithms relies on the management of interprocess communication. Yet, these new computational environments raise efficiency and accuracy issues about evaluating the convergence state of asynchronous parallel iterative processes. Indeed, with such increasing communication loads, there is no trivial efficient way to compute a consistent residual error from the distributed components of a global, potential solution, vector. Therefore, a well designed detection technique is required, in order to avoid both untimely and delayed termination. In this paper, we propose new efficient methods to accurately evaluate the residual of a computation during asynchronous iterations. Such a matter is not related to conditions under which an asynchronous iterative algorithm is guaranteed to converge, but rather consists in designing some efficient and effective way of asserting that an ongoing asynchronous iterative computation has actually reached its convergence state.

Section 2 gives a brief overview of main existing approaches and protocols for terminating asynchronous iterations. Section 3 presents the asynchronous iterations model that is under consideration, then formally states the convergence detection problem that is addressed in this

- *F. Magoulès is with the CentraleSupélec, Université Paris-Saclay, Gif-sur-Yvette 91191, France. E-mail: frederic.magoules@hotmail.com.*
- *G. Gbikpi-Benissan was with the IRT SystemX, Paris-Saclay 91127, France. E-mail: guibenissan@gmail.com.*

Manuscript received 28 May 2017; revised 28 Oct. 2017; accepted 4 Dec. 2017. Date of publication 7 Dec. 2017; date of current version 9 Mar. 2018.
(Corresponding author: Guillaume Gbikpi-Benissan)
Recommended for acceptance by S.S. Vadhiyar.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2017.2780856

study. Section 4 details basic ideas of snapshot protocols, leading to our propositions for asynchronous iterations termination in First-In-First-Out (FIFO) communication environments. Then, Section 5 tackles various non-FIFO communication contexts. Two new protocols are proposed for arbitrary non-FIFO communication, another one for non-FIFO communication only on messages which have different labels, and at last two others based on heuristics, for non-FIFO communication only within successive finite sets of exchanged messages. Section 6 comments some experimental results on two different computation platforms, using up to 5,600 processor cores. Effectiveness and efficiency are discussed against two existing termination methods. Section 7 summarizes our conclusions.

2 Related Works

The problem of terminating asynchronous iterations was well discussed in, e.g., [16], where the authors introduced a first approach which consists in altering the asynchronous iterative algorithm such that it terminates in finite time and then applying one of the classical termination detection protocols available in the distributed algorithms field (see, e.g., [17], [18], [19], [20]). Indeed, these termination protocols are designed for parallel applications that are executed in a finite number of steps, that is to say, there is a point, during their execution, from where all single processes are idle. Since this is not natively the case for a large class of iterative algorithms, different modifications have been proposed (see, e.g., [21], [22], [23]) for detecting their convergence by means of a classical distributed termination protocol. Basically, any process under some local conditions (relative to local convergence) stops sending new data to its neighbors in the communication graph, so that the termination condition may consist of having all processes under this local condition, without any message in transit. Another kind of alteration has been discussed in [24], which consists in turning back to synchronous iterations at some point of the execution where local convergence seems to persist on one of the processes.

A second approach, called *supervised termination*, consists in using a supervisory algorithm to take a snapshot of the computation, in order to construct and evaluate a global solution in parallel of the iterative process. Considering the well-known snapshot protocol due to Chandy and Lamport [25], it is still interesting to see how it applies for asynchronous iterations termination in a simplified form (see Section 4.2). Yet, the main disadvantage of such a protocol is the FIFO property required on the communication channels. Attempts to achieve general non-FIFO snapshots are based either on message acknowledgment and delayed delivering, or on piggybacking of control information on top of application messages (see [26] for an introductory overview). Such approaches thus turn out to be quite intrusive and, furthermore, not easy to implement. In [21], some supervised termination protocols, more or less centralized, were designed over both star and tree network topologies, introducing a new non-FIFO, but simplified, snapshot. The less centralized approach therein involves a spanning tree over the network graph where local convergence notifications propagate from the leaves to the root process. This one then triggers the simplified snapshot allowing each process to evaluate a globally coherent local solution. The centralization is thus limited to the notifications gathering phase for coordination purpose. Consistency, for non-FIFO channels, is guaranteed by inserting computation message data into the snapshot messages, which introduces a non-negligible overhead for communication.

A third approach in [27] is based on a leader election protocol on tree topology [28, Section 4.4.3] wherein the authors introduced cancellation messages to manage the false convergence issue. The algorithm however requires to estimate an upper bound on the communication delay between any two processes. Then, in [29], these authors proposed a new solution which takes off this requirement, as well as cancellation messages, by performing a verification phase after a presumed global convergence. The leading idea is to monitor the persistence of this convergence state within a period which must last enough to have every dependencies updated with data at least as recent as the presumed detection time. Global convergence is confirmed if during this period no process ever left its local convergence state. As an inconvenient for non-FIFO environments, piggybacking techniques must be used to distinguish data emitted within the verification phase period. While such an approach can avoid premature termination with a high probability, it does not provide a way of evaluating a consistent global residual. Yet, its reliability could be guaranteed by mixing it with the formal analysis from [30] where the convergence tests are based on the diameter of successive nested sets, which are identified by means of macro-iterations defined as minimal sets of iterations within which all of the solution vector components are updated at least once. Nevertheless, just as in [21], this third approach also features a first gathering phase through the leader election, which actually acts as a dynamically centralized coordination.

In summary, second and third approaches allow us to detect the convergence of asynchronous iterations without altering the main computation process. But for both, current solutions somehow require two gathering phases, one for coordination and another for convergence state evaluation. In very large distributed systems, such reduction operations would constitute the most costly part of these convergence detection protocols. We investigate here new methods, mostly non-intrusive, to evaluate the convergence residual of a computation during asynchronous iterations, using only one reduction operation. Furthermore, some non-FIFO cases are managed through strong heuristics, without piggybacking or over-exchange of computation data.

3 Problem Formulation

3.1 Asynchronous Iterations

Let $X = X_1 \times \cdots \times X_n$ be a product of vector spaces, and let us consider a mapping

$$\begin{aligned} f : X_1 \times \cdots \times X_n &\to X_1 \times \cdots \times X_n, \\ x = (x_1, \ldots, x_n) &\mapsto (f_1(x), \ldots, f_n(x)), \end{aligned}$$

where $f_i : X \to X_i$, $i \in \{1, \ldots, n\}$, are given. Now let $\{I^k\}_{k\in\mathbb{N}}$ be a sequence of integer subsets such that

$$\forall k \in \mathbb{N}, \quad I^k \subseteq \{1, \ldots, n\}, \quad I^k \neq \emptyset.$$

Asynchronous iterations exhibit a sequence $\{x^k\}_{k\in\mathbb{N}}$ of vectors in X such that

$$x_i^{k+1} = \begin{cases} f_i(x_1^{\rho_1^i(k)}, \ldots, x_n^{\rho_n^i(k)}), & i \in I^k, \\ x_i^k, & i \notin I^k, \end{cases} \tag{1}$$

where ρ_j^i, with $i, j \in \{1, \ldots, n\}$, are integer-valued functions on $\mathbb{N}$, satisfying

$$\rho_j^i(k) \leq k, \quad \forall k \in \mathbb{N},$$

which denotes a delay on the version of the component j used to update the component i. I^k is thus the set of components updated at iteration k. For convergence analysis, the computational model (1) is generally completed by the two following assumptions, which ensure that, for any given $k_0 \in \mathbb{N}$ before convergence, no component sequence $\{x_i^{k_0}, x_i^{k_0+1}, \ldots\}$ definitively freezes or is generated by using some other fixed component $x_j^{k_1}$, $k_1 \in \mathbb{N}$.

Assumption 1. $\forall i \in \{1, \ldots, n\}$, $\mathrm{card}\{k \in \mathbb{N} | i \in I^k\} = +\infty$.

Assumption 2. $\forall i, j \in \{1, \ldots, n\}$, $\lim_{k\to+\infty} \rho_j^i(k) = +\infty$.

3.2 Convergence Detection

Let us consider a sequence $\{x^k\}_{k\in\mathbb{N}}$ of vectors in X satisfying the asynchronous iterations model (1), and define n sequences $\{y^{1,k}\}_{k\in\mathbb{N}}, \ldots, \{y^{n,k}\}_{k\in\mathbb{N}}$ of vectors in X such that

$$y^{i,k} = (x_1^{\rho_1^i(k)}, \ldots, x_n^{\rho_n^i(k)}), \quad \forall i \in \{1, \ldots, n\}, \forall k \in \mathbb{N}. \tag{2}$$

$y^{i,k}$ thus denotes the global vector used to update the component i of the solution vector x at the iteration $k+1$. Additionally, we assume to have

$$\rho_i^i(k) = k, \quad \forall i \in \{1, \ldots, n\}, \forall k \in \mathbb{N}. \tag{3}$$

At last, let $\bar{x}$ be a vector in X given by

$$\bar{x} = (y_1^{1,k_1}, \ldots, y_n^{n,k_n}), \quad k_1, \ldots, k_n \in \mathbb{N},$$

which denotes a global vector built from an arbitrary version of each local component. We will address in this paper the problem of evaluating a relation

$$\|f(\bar{x}) - \bar{x}\| < \varepsilon, \quad \varepsilon \in \mathbb{R}, \tag{4}$$

where $\|.\|$ is a norm on X. To solve this problem, attention will be mainly paid to the computation of $f(\bar{x})$.

One notices that synchronous iterations correspond to the case where we have $I^k = \{1, \ldots, n\}$ and $\rho_j^i(k) = k$, for all $k \in \mathbb{N}$ and all $i, j \in \{1, \ldots, n\}$. It follows that by taking $\bar{x} = (y_1^{1,k}, \ldots, y_n^{n,k})$, for any $k \in \mathbb{N}$, we obtain, for all $i \in \{1, \ldots, n\}$

$$\begin{aligned} f_i(\bar{x}) &= f_i(y_1^{1,k}, \ldots, y_n^{n,k}), \\ &= f_i(x_1^k, \ldots, x_n^k), \\ &= f_i(x_1^{\rho_1^i(k)}, \ldots, x_n^{\rho_n^i(k)}), \\ &= x_i^{k+1}, \end{aligned}$$

which implicitly gives $f(\bar{x}) = (y_1^{1,k+1}, \ldots, y_n^{n,k+1})$.

We point out that the relation (4) can be more generally given by:

$$\bar{x} \in S^*, \tag{5}$$

where S^* is the set of admissible solutions, as also suggested in [21]. Then actually, the quality of the solution $\bar{x}$ would depend on the suitable choice of a residual evaluation function $r(\bar{x})$, which is application-dependent, regardless the context of asynchronous iterations. By considering however a residual evaluation function of the form (4), we intend to provide a better understanding of the subsequent discussions, without losing their general applicability to (5).

4 Determining a Global Solution Vector

4.1 The Chandy–Lamport Snapshot (CLS)

The basic idea within the CLS protocol is to record, not only the local state of each process, but also the state of each communication channel. Any process (possibly several processes) can initiate the protocol by recording its local state and sending a "marker" to all of its neighbors in the communication graph. Non-initiators do the same when they receive a marker for the first time. As soon as a process records its local state, it starts recording the state of its reception channels. From then, and before marker reception on any channel, any message received is appended to the state of this channel. Consequently, the recording ends when a marker is received from all of the neighboring processes. Algorithm 1 outlines the rules that fully describe the protocol.

Algorithm 1. CLS Protocol

```
if initiator then
    if state not recorded then
        Record state
        Send a marker to each neighbor in the communication
          graph
    end if
end if
if marker received then
    if state not recorded then
        Record state
        Send a marker to each neighbor in the communication
          graph
    end if
    if marker received from each neighbor then
        Return state and state of each reception channel
    end if
end if
if computation message received then
    if state recorded and marker not received from the
      sender then
    Add the message to the state of the corresponding recep-
      tion channel
    end if
end if
```

To give an intuitive understanding of the consistency of the global state built by this snapshot protocol, we show, in Fig. 1, a simple example involving two processes, denoted by p and q. Let us consider events consisting in sending and receiving a message. In this example, the process p records its local state after the event $e1$ and sends a marker (dotted arrow) to the process q. On reception of the marker, the process q records its local state after the event $e4$, then records the state of its reception channel as an empty set, and finally,

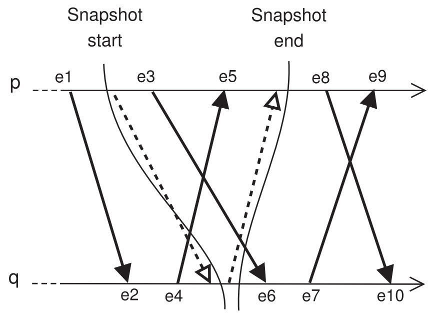

Fig. 1. Example of a CLS protocol execution with two processes.

sends the marker back to the process p. Before receiving the marker from the process q, the process p received a computation message from q as event $e5$. Therefore, the state of the reception channel of the process p corresponds to the set $\{e5\}$. It is clear from this example that the communication channels need to be FIFO. Otherwise, if for instance the marker sent by the process q is received by the process p before the event $e5$, therefore the state of the channel is an empty set, which causes an information lost about the event $e5$.

This example builds a global state relative to last events $\{e1, e4\}$ and records the set of pending messages relative to event $e5$. However, according to the events sequence, this state does not match any of the states the system actually went through. Indeed, one can see that the event $e3$ should be taken into account as we consider the state of the system just after the event $e4$. Therefore, let us highlight what is relevant about the state recorded by an execution of the CLS protocol.

Theorem 1 (Chandy & Lamport, 1985). *Let $\mathcal{S}(\mathcal{C}) = \{s^t\}_{t\in\mathbb{N}}$ denote the global states sequence generated by a computation $\mathcal{C}$. Let $\bar{s}$ be the global state recorded by an execution of the CLS protocol on $\mathcal{C}$. Then there exists an equivalent permutation $\mathcal{P}(\mathcal{C})$ of $\mathcal{C}$ such that $\bar{s} \in \mathcal{S}(\mathcal{P}(\mathcal{C}))$.*

Proof. See [25]. □

4.2 New Asynchronous Iterations Snapshots (AIS)

Let again sequences $\{x^k\}_{k\in\mathbb{N}}$ and $\{y^{i,k}\}_{k\in\mathbb{N}}$, $i \in \{1,\dots,n\}$, be defined as in Section 3.2. Let us suppose an associated parallel computation involving n processes, and let each process $i \in \{1,\dots,n\}$ record a vector $\bar{y}^i \in X$ by following the rules described either in Algorithm 2 or in Algorithm 3.

We should mention that the variable k therein may have different values at different places in the algorithms, as the rules conditions may be fulfilled at different times. To be more precise, we would then have

$$\bar{y}^i = \left(y_1^{i,k_{i,1}}, \dots, y_n^{i,k_{i,n}}\right), \qquad k_{i,j} \in \mathbb{N}, \quad i,j \in \{1,\dots,n\}.$$

One can notice that, contrarily to the CLS protocol, there is no rule for channel record at computation message reception. More, in Algorithm 3, recording the local state is not required at the first marker reception. However, for both algorithms, we still need the following preliminary assumptions.

Assumption 3. *Each process performs at least one iteration, which means*

$$\forall i \in \{1,\dots,n\}, \quad \exists k < k_{i,i} : i \in I^k.$$

Assumption 4. *After computation of $y_i^{i,k+1}$ (i.e., $i \in I^k$), $y_i^{i,k+1}$ is sent to each process $j \neq i$, before any other communication toward j.*

Assumption 5. *Communication channels are FIFO.*

A consistent global solution vector, under asynchronous iterations, is then given by the following result.

Proposition 1. *Let a sequence $\{x^k\}_{k\in\mathbb{N}}$, satisfying the asynchronous iterations model (1), be generated by a computation $\mathcal{C}$ involving n processes. Let sequences $\{y^{1,k}\}_{k\in\mathbb{N}}, \dots, \{y^{n,k}\}_{k\in\mathbb{N}}$ be defined by the rewriting (2). Let, at last, $\bar{y}^1, \dots, \bar{y}^n$ be the vectors returned by an execution of either the AIS protocol 1 or the AIS protocol 2 on $\mathcal{C}$. Then, under Assumptions 3, 4, and 5, we have*

$$\bar{y}^1 = \bar{y}^2 = \cdots = \bar{y}^n.$$

Proof. Let $i, j \in \{1,\dots,n\}$ be two any process identifiers. According to the local state recording rule and Assumption 3, there exists $k_0^i < k_{i,i}$, with $i \in I^{k_0^i}$, satisfying

$$\forall k \in \{k_0^i + 1, \dots, k_{i,i} - 1\}, \quad i \notin I^k,$$

so that we have

$$y_i^{i,k_{i,i}} = y_i^{i,k_{i,i}-1} = \cdots = y_i^{i,k_0^i+1}.$$

With Assumptions 4 and 5, it follows that there also exists $k_0^j \in \mathbb{N}$, $k_0^j \le k_{j,i}$, such that

$$y_i^{j,k_0^j} = y_i^{i,k_0^i+1}.$$

Assumption 5 implies that

$$\forall k \in \{k_0^j + 1, \dots, k_{j,i}\}, \quad y_i^{j,k} = y_i^{j,k_0^j}.$$

Then, in particular, we have

$$y_i^{j,k_{j,i}} = y_i^{j,k_0^j} = y_i^{i,k_0^i+1} = y_i^{i,k_{i,i}},$$

and thus

$$\bar{y}_i^i = \bar{y}_i^j, \quad \forall i,j \in \{1,\dots,n\}, \tag{6}$$

which concludes the proof. □

We can thus have a vector $\bar{x} = (\bar{y}_1^1, \dots, \bar{y}_n^n)$, so that we implicitly obtain

$$f(\bar{x}) = (f_1(\bar{y}^1), \dots, f_n(\bar{y}^n)).$$

Assumptions 3 and 4 are pretty natural conditions that are easily satisfied in an iterative loop where the AIS protocol rules are called after the main computation and message sending part. They are necessary to be mentioned, however, especially for multi-threaded processes. Assumption 5 is then the sole actual constraint in the above protocols. The next section discusses about taking off such requirement.

5 New Non-FIFO Asynchronous Iterations Snapshots

5.1 Arbitrary Non-FIFO Communication

The FIFO condition is essential to avoid the two situations depicted in Fig. 2, where a marker (dotted arrow) crosses a computation message. In such cases, the equality in (6) is no

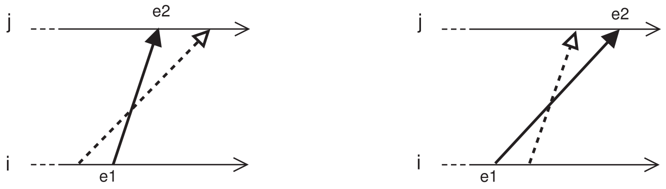

Fig. 2. Non-FIFO snapshot issues.

Algorithm 2. AIS Protocol 1

```
if $\|y_i^{i,k} - y_i^{i,k_0^i}\| < \varepsilon$, with $y_i^{i,k} = y_i^{i,k_0^i+1}$, $i \in I^{k_0^i}$ then
    if $\bar{y}_i^i$ undefined then
        $\bar{y}_i^i := y_i^{i,k}$
        for all process $j \neq i$ do
            Send a marker to $j$
        end for
    end if
end if
if marker received from a process $j \neq i$ then
    $\bar{y}_j^i := y_j^{i,k}$
    if $\bar{y}_i^i$ undefined then
        $\bar{y}_i^i := y_i^{i,k}$
        for all process $j \neq i$ do
            Send a marker to $j$
        end for
    end if
    if $\bar{y}_j^i$ defined for all $j$ then
        return $\bar{y}^i$
    end if
end if
```

Algorithm 3. AIS Protocol 2

```
if $\|y_i^{i,k} - y_i^{i,k_0^i}\| < \varepsilon$, with $y_i^{i,k} = y_i^{i,k_0^i+1}$, $i \in I^{k_0^i}$ then
    if $\bar{y}_i^i$ undefined then
        $\bar{y}_i^i := y_i^{i,k}$
        for all process $j \neq i$ do
            Send a marker to $j$
        end for
    end if
end if
if marker received from a process $j \neq i$ then
    $\bar{y}_j^i := y_j^{i,k}$
end if
if $\bar{y}_j^i$ defined for all $j$ then
    return $\bar{y}^i$
end if
```

more satisfied. Then, one can apply ideas from [21] to AIS protocols 1 and 2, as described by Algorithms 4 and 5, respectively. Here, markers contain computation data, so that these solutions actually even handle crossed computation messages. Proposition 1 becomes the following, which does not need any of the previous assumptions:

Proposition 2. *Let a sequence $\{x^k\}_{k\in\mathbb{N}}$, satisfying the asynchronous iterations model (1), be generated by a computation $\mathcal{C}$ involving n processes. Let sequences $\{y^{1,k}\}_{k\in\mathbb{N}}, \ldots, \{y^{n,k}\}_{k\in\mathbb{N}}$ be defined by the rewriting (2). Let, at last, $\bar{y}^1, \ldots, \bar{y}^n$ be the vectors returned by an execution of either the non-FIFO AIS protocol 1 or the non-FIFO AIS protocol 2 on $\mathcal{C}$. Then, we have*

$$\bar{y}^1 = \bar{y}^2 = \cdots = \bar{y}^n.$$

Proof. By construction, we trivially satisfy the equality in (6). □

Algorithm 4. Non-FIFO AIS Protocol 1

```
if $\|y_i^{i,k} - y_i^{i,k_0^i}\| < \varepsilon$, with $y_i^{i,k} = y_i^{i,k_0^i+1}$, $i \in I^{k_0^i}$ then
    if $\bar{y}_i^i$ undefined then
        $\bar{y}_i^i := y_i^{i,k}$
        for all process $j \neq i$ do
            Send a marker $\bar{y}_i^i$ to $j$
        end for
    end if
end if
if marker $\bar{y}_j^j$ received from a process $j \neq i$ then
    $\bar{y}_j^i := \bar{y}_j^j$
    if $\bar{y}_i^i$ undefined then
        $\bar{y}_i^i := y_i^{i,k}$
        for all process $j \neq i$ do
            Send a marker $\bar{y}_i^i$ to $j$
        end for
    end if
    if $\bar{y}_j^i$ defined for all $j$ then
        return $\bar{y}^i$
    end if
end if
```

Algorithm 5. Non-FIFO AIS Protocol 2

```
if $\|y_i^{i,k} - y_i^{i,k_0^i}\| < \varepsilon$, with $y_i^{i,k} = y_i^{i,k_0^i+1}$, $i \in I^{k_0^i}$ then
    if $\bar{y}_i^i$ undefined then
        $\bar{y}_i^i := y_i^{i,k}$
        for all process $j \neq i$ do
            Send a marker $\bar{y}_i^i$ to $j$
        end for
    end if
end if
if marker $\bar{y}_j^j$ received from a process $j \neq i$ then
    $\bar{y}_j^i := \bar{y}_j^j$
end if
if $\bar{y}_j^i$ defined for all $j$ then
    return $\bar{y}^i$
end if
```

5.2 Inter-Protocol Non-FIFO Communication

In the communication model considered now, FIFO channels are used at least for computation messages. This is a highly realistic model, as being a natural expectation to achieve minimum delays during asynchronous iterations, and moreover, it is a requirement for classical iterations. Still, the problem of markers crossing computation messages remains. We propose, with Algorithm 6, a snapshot solution which, outrightly, do not need marker exchange, and is based on only computation messages, even without piggybacking. Here, just as local solution buffers, each process i maintains access to the two latest received messages, for all neighbor processes $j \neq i$. Then, process i can detect by itself local convergence of process j and immediately record the last value received. Proposition 1 becomes the following:

Algorithm 6. Non-FIFO AIS Protocol 3

```
if $\|y_i^{i,k} - y_i^{i,k_i^i}\| < \varepsilon$, with $y_i^{i,k} = y_i^{i,k_i^i+1}$, $i \in I^{k_i^i}$ then
    if $\bar{y}_i^i$ undefined then
        $\bar{y}_i^i := y_i^{i,k}$
    end if
end if
if $\|y_j^{i,k} - y_j^{i,k_j^i}\| < \varepsilon$, with $\rho_j^i(k_j^i) = k_j^j$, $\rho_j^i(k) = k_j^j + 1$ then
    if $\bar{y}_j^i$ undefined then
        $\bar{y}_j^i := y_j^{i,k}$
    end if
end if
if $\bar{y}_j^i$ defined for all $j$ then
    return $\bar{y}^i$
end if
```

Proposition 3. *Let a sequence $\{x^k\}_{k\in\mathbb{N}}$, satisfying the asynchronous iterations model (1), be generated by a computation $\mathcal{C}$ involving n processes. Let sequences $\{y^{1,k}\}_{k\in\mathbb{N}}, \dots, \{y^{n,k}\}_{k\in\mathbb{N}}$ be defined by the rewriting (2). Let, at last, $\bar{y}^1, \dots, \bar{y}^n$ be the vectors returned by an execution of the non-FIFO AIS protocol 3 on $\mathcal{C}$. Then, we have*

$$\bar{y}^1 = \bar{y}^2 = \cdots = \bar{y}^n.$$

Proof. Let $i, j \in \{1, \dots, n\}$ be two any process identifiers. Remind $\bar{y}_j^i = y_j^{i,k_{i,j}}$, $k_{i,j} \in \mathbb{N}$. Then according to (2) and (3), we have

$$\bar{y}_j^i = x_j^{\rho_j^i(k_{i,j})} = x_j^{\rho_j^j(\rho_j^i(k_{i,j}))} = y_j^{j,\rho_j^i(k_{i,j})}.$$

By construction, we satisfy

$$\rho_j^i(k_{i,j}) = k_j^j + 1, \qquad y_j^{j,k_{j,j}} = y_j^{j,k_j^j+1},$$

and thus

$$\bar{y}_j^i = y_j^{j,k_j^j+1} = y_j^{j,k_{j,j}} = \bar{y}_j^j,$$

which concludes the proof. □

5.3 Non-FIFO Communication with Bounded Number of Cross Messages

In case of very large problems, non-FIFO AIS protocols 1 to 3 may introduce non-negligible overhead costs, either for communication or for memory. But on another hand, for such large problems, deciding to compute a solution may depend on guaranteeing a minimum performance level of the parallel computation platform. Especially, when a given maximum execution time is expected, this most likely includes to ensure a bound on communication delays. We thus reasonably make here a preliminary assumption.

Assumption 6. *A message can cross at most η other messages.*

Let us then consider Algorithm 7. Here, a process i sends its marker to a process $j \neq i$ only when local convergence persists on process i for some iterations k_l^i, with $i \in I^{k_l^i}$ and $l \in \mathbb{N}$. Such iterations will be referred to as 'steady iterations'. This way, even if the marker is received on the process j before the latest message sent by the process i, the message recorded by the process j is still relevant in the sense that the two latest messages from process i contain very close data (due to the persistence of the local convergence). Then, a second type of marker (dashed arrow in Fig. 3) is sent by the process i to transmit a binary flag after some additional iterations. If local convergence still persists during these iterations, the flag is armed, which confirms the relevance of the message data recorded by the process j, even if it corresponds to the message sent by the process i after the first marker (again, due to local convergence persistence after sending the first marker). Otherwise, processes i and j discard the corresponding records and try again. One can also see that the algorithm still works even in the case where the flag-marker crosses the first one, as depicted in Fig. 3 (right).

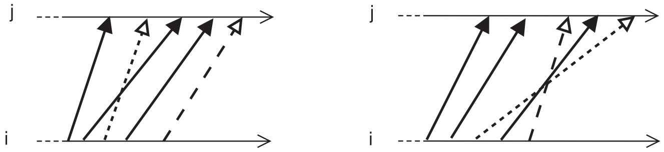

Fig. 3. Examples of issues handled by non-FIFO AIS protocol 4.

Algorithm 7. Non-FIFO AIS Protocol 4

```
if $\|y_i^{i,t+1} - y_i^{i,t}\| < \varepsilon, \forall t \in \{k_0^i, \dots, k-1\} : i \in I^t$ then
    if $\bar{y}_i^i$ undefined then
        $\bar{y}_i^i := y_i^{i,k}$
        for all process $j \neq i$ do
            Send a marker to $j$
        end for
        $k_{i,i} := k$
        Mark $\phi_i^i$ as undefined
    end if
end if
if $\|y_i^{i,t+1} - y_i^{i,t}\| < \varepsilon, \forall t \in \{k_{i,i}, \dots, k-1\} : i \in I^t$ then
    if $\phi_i^i$ undefined then
        $\phi_i^i := 1$
        for all process $j \neq i$ do
            Send a flagged marker $\phi_i^i$ to $j$
        end for
    end if
else
    if $\phi_i^i$ undefined then
        $\phi_i^i := 0$
        for all process $j \neq i$ do
            Send a flagged marker $\phi_i^i$ to $j$
        end for
        Mark $\bar{y}_i^i$ as undefined
    end if
end if
if marker received from a process $j \neq i$ then
    $\bar{y}_j^i := y_j^{i,k}$
end if
if flagged marker $\phi_j^j$ received from a process $j \neq i$ then
    $\phi_j^i := \phi_j^j$
    if $\phi_j^i = 0$ then
        Mark $\bar{y}_j^i$ as undefined
    end if
end if
if $\bar{y}_j^i$ defined and $\phi_j^i = 1$ for all $j$ then
    return $\bar{y}^i$
end if
```

As a particular case of this communication model, one may further assume that the crossing ability is tightly related to the size of the messages. Indeed, if control messages (e.g., markers) are transmitted far faster than computation messages (due to the difference in size), we may assume that, from a process to another process, a computation message sent later than a control message cannot be received earlier than this one. Then in such case, flagged markers would not be necessary any more, which rather simplifies the protocol and provides Algorithm 8.

Algorithm 8. Non-FIFO AIS Protocol 5

```
if $\|y_i^{i,t+1} - y_i^{i,t}\| < \varepsilon, \forall t \in \{k_0^i, \dots, k-1\} : i \in I^t$ then
    if $\bar{y}_i^i$ undefined then
        $\bar{y}_i^i := y_i^{i,k}$
        for all process $j \neq i$ do
            Send a marker to $j$
        end for
    end if
end if
if marker received from a process $j \neq i$ then
    $\bar{y}_j^i := y_j^{i,k}$
end if
if $\bar{y}_j^i$ defined for all $j$ then
    return $\bar{y}^i$
end if
```

Now, let us define the mapping

$$g: \quad X^n \;\rightarrow\; X_1 \times \cdots \times X_n, \\ (y^1, \dots, y^n) \mapsto (f_1(y^1), \dots, f_n(y^n)),$$

and the vector $\bar{y} = (\bar{y}^1, \dots, \bar{y}^n)$, so that we implicitly obtain

$$g(\bar{y}) = (f_1(\bar{y}^1), \dots, f_n(\bar{y}^n)).$$

In the following, we establish the reliability of the approximated residual

$$\|g(\bar{y}) - \bar{x}\|, \quad \bar{x} = (\bar{y}_1^1, \dots, \bar{y}_n^n),$$

compared to the exact one given by $\|f(\bar{x}) - \bar{x}\|$. Let then $\|.\|_{(i)}$, $i \in \{1, \dots, n\}$, be a given norm defined on X_i, and let us consider $\mathcal{L}_p$-norms, $p \in [1, +\infty)$, defined on X by

$$\|x\|_p = \left(\sum_{i=1}^{n} \|x_i\|_{(i)}{}^p \right)^{1/p}.$$

Maximum norms could be considered as well, as particular cases. We assume the following property for the mapping f.

Assumption 7. *For any i and j in $\{1, \dots, n\}$, there exists $\delta_{i,j}$ in $\mathbb{R}^{+*}$ such that*

$$\|x_j - x'_j\|_{(j)} < \varepsilon,$$

implies

$$\|f_i(x) - f_i(x_1, \dots, x'_j, \dots, x_n)\|_{(i)} < \delta_{i,j}\varepsilon,$$

with x and x' in X.

Notation 1. $\delta(f) = \max_{i=1}^{n} \sum_{j=1}^{n} \delta_{i,j}(f)$, *where $\delta_{i,j}(f)$ are the smallest $\delta_{i,j}$ satisfying Assumption 7.*

At last, we also need the following assumption.

Assumption 8. *A process sends its markers and armed flag-markers after at least η steady iterations.*

Then, we give an essential result about the accuracy of our heuristics.

Proposition 4. *Let a sequence $\{x^k\}_{k\in\mathbb{N}}$, satisfying the asynchronous iterations model (1), be generated by a computation $\mathcal{C}$ involving n processes. Let sequences $\{y^{1,k}\}_{k\in\mathbb{N}}, \dots, \{y^{n,k}\}_{k\in\mathbb{N}}$ be defined by the rewriting (2). Let, at last, $\bar{y}^1, \dots, \bar{y}^n$ be the vectors returned by an execution of the non-FIFO AIS protocol 4 on $\mathcal{C}$. Then, under Assumptions 4 and 6, 7, and 8, we have*

$$\|f(\bar{x}) - \bar{x}\|_p - \|g(\bar{y}) - \bar{x}\|_p < n^{1/p}\eta\delta(f)\varepsilon,$$

with $\bar{y} = (\bar{y}^1, \dots, \bar{y}^n)$ and $\bar{x} = (\bar{y}_1^1, \dots, \bar{y}_n^n)$.

Proof. Let us take again

$$\bar{y}_j^i = y_j^{i,k_{i,j}}, \quad \forall i, j \in \{1, \dots, n\},$$

with $k_{i,j} \in \mathbb{N}$. Then according to (2) and (3), we have

$$\bar{y}_i^j = y_i^{j,k_{j,i}} = x_i^{\rho_i^j(k_{j,i})} = x_i^{\rho_i^i(\rho_i^j(k_{j,i}))} = y_i^{i,\rho_i^j(k_{j,i})}.$$

Assumptions 4, 6 and 8 ensure

$$\left| \left\{ k \in \{\rho_i^j(k_{j,i}), \dots, k_{i,i} - 1\} \mid i \in I^k \right\} \right| \leq \eta. \tag{7}$$

Let us then consider

$$\{k_1^i, \dots, k_{m_i}^i\} = \left\{ k \in \{\rho_i^j(k_{j,i}), \dots, k_{i,i} - 1\} \mid i \in I^k \right\},$$

with $m_i \in \mathbb{N}^*$. It follows:

$$\begin{aligned} \|\bar{y}_i^i - \bar{y}_i^j\|_{(i)} &= \left\| y_i^{i,k_{i,i}} - y_i^{i,\rho_i^j(k_{j,i})} \right\|_{(i)}, \\ &= \left\| y_i^{i,k_{m_i}^i+1} - y_i^{i,k_1^i} \right\|_{(i)}, \\ &= \left\| y_i^{i,k_{m_i}^i+1} - y_i^{i,k_{m_i}^i} + y_i^{i,k_{m_i}^i} - y_i^{i,k_{m_i-1}^i} \right. \\ &\quad \left. + \cdots + y_i^{i,k_2^i} - y_i^{i,k_1^i} \right\|_{(i)}, \\ &\leq \left\| y_i^{i,k_{m_i}^i+1} - y_i^{i,k_{m_i}^i} \right\|_{(i)} \\ &\quad + \left\| y_i^{i,k_{m_i}^i} - y_i^{i,k_{m_i-1}^i} \right\|_{(i)} \\ &\quad + \cdots + \left\| y_i^{i,k_2^i} - y_i^{i,k_1^i} \right\|_{(i)}, \\ &< m_i\varepsilon. \end{aligned}$$

Now take, as always, $\bar{x} = (\bar{y}_1^1, \dots, \bar{y}_n^n)$. Then we have, for all $i \in \{1, \dots, n\}$

$$\begin{aligned} \|f_i(\bar{x}) - f_i(\bar{y}^i)\|_{(i)} &= \|f_i(\bar{y}_1^1, \dots, \bar{y}_n^n) - f_i(\bar{y}_1^i, \dots, \bar{y}_n^i)\|_{(i)}, \\ &\leq \|f_i(\bar{y}_1^1, \dots, \bar{y}_n^n) - f_i(\bar{y}_1^i, \bar{y}_2^2, \dots, \bar{y}_n^n)\|_{(i)} \\ &\quad + \|f_i(\bar{y}_1^i, \bar{y}_2^2, \dots, \bar{y}_n^n) \\ &\quad - f_i(\bar{y}_1^i, \bar{y}_2^i, \bar{y}_3^3, \dots, \bar{y}_n^n)\|_{(i)} + \cdots \\ &\quad + \|f_i(\bar{y}_1^i, \dots, \bar{y}_{n-1}^i, \bar{y}_n^n) - f_i(\bar{y}_1^i, \dots, \bar{y}_n^i)\|_{(i)}. \end{aligned}$$

Accounting Assumption 7 on f, it follows

$$\|f_i(\bar{x}) - f_i(\bar{y}^i)\|_{(i)} < \sum_{\substack{j=1\\ j\neq i}}^{n} \delta_{i,j}(f) m_j \varepsilon.$$

Consider finally $\bar{y} = (\bar{y}^1, \ldots, \bar{y}^n)$. Then we have

$$\begin{aligned}
\|f(\bar{x}) - \bar{x}\|_p &= \|f(\bar{x}) - g(\bar{y}) + g(\bar{y}) - \bar{x}\|_p, \\
&\leq \|g(\bar{y}) - \bar{x}\|_p \\
&\quad + \left(\sum_{i=1}^{n} \|f_i(\bar{x}) - f_i(\bar{y}^i)\|_{(i)}{}^p \right)^{1/p}, \\
&< \|g(\bar{y}) - \bar{x}\|_p \\
&\quad + \left(\sum_{i=1}^{n} \left(\sum_{\substack{j=1\\ j\neq i}}^{n} \delta_{i,j}(f) m_j \varepsilon \right)^p \right)^{1/p}, \\
&< \|g(\bar{y}) - \bar{x}\|_p + n^{1/p} \max_{i=1}^{n} \sum_{\substack{j=1\\ j\neq i}}^{n} \delta_{i,j}(f) m_j \varepsilon.
\end{aligned}$$

Applying (7), which means $m_j \leq \eta$, and using Notation 1, we conclusively obtain

$$\|f(\bar{x}) - \bar{x}\|_p < \|g(\bar{y}) - \bar{x}\|_p + n^{1/p} \eta \delta(f) \varepsilon.$$

□

Relatively to weighted maximum norms, defined on X by

$$\|x\|^w_\infty = \max_{i=1}^{n} \frac{\|x_i\|_{(i)}}{w_i}, \quad w \in (\mathbb{R}^{+*})^n,$$

let us assume that f is contractive, i.e.:

Assumption 9. *There exists a real* $\alpha < 1$ *such that*

$$\|f(x) - f(x')\|^w_\infty \leq \alpha \|x - x'\|^w_\infty, \quad \forall x, x' \in X.$$

Then, one may want to apply the following practical result.

Proposition 5. *Let a sequence* $\{x^k\}_{k\in\mathbb{N}}$*, satisfying the asynchronous iterations model (1), be generated by a computation* $\mathcal{C}$ *involving* n *processes. Let sequences* $\{y^{1,k}\}_{k\in\mathbb{N}}, \ldots, \{y^{n,k}\}_{k\in\mathbb{N}}$ *be defined by the rewriting (2). Let, at last,* $\bar{y}^1, \ldots, \bar{y}^n$ *be the vectors returned by an execution of the non-FIFO AIS protocol 4 on* $\mathcal{C}$*. Then, under Assumptions 4, 6, 8 and 9*

$$\|g(\bar{y}) - \bar{x}\|^w_\infty \leq \varepsilon = \frac{\varepsilon'}{1 + \eta \min_{i=1}^{n} w_i},$$

implies

$$\|f(\bar{x}) - \bar{x}\|^w_\infty < \varepsilon',$$

with $\bar{y} = (\bar{y}^1, \ldots, \bar{y}^n)$, $\bar{x} = (\bar{y}^1_1, \ldots, \bar{y}^n_n)$ *and* $\varepsilon' \in \mathbb{R}$.

Proof. Considering the proof of Proposition 4, we recall

$$\|\bar{y}^i_i - \bar{y}^j_i\|_{(i)} < m_i \varepsilon, \quad i, j \in \{1, \ldots, n\}.$$

According to Assumption 9, we have

$$\|f(\bar{x}) - f(\bar{y}^i)\|^w_\infty \leq \alpha \|\bar{x} - \bar{y}^i\|^w_\infty, \quad \forall i \in \{1, \ldots, n\},$$

and then, in particular

$$\begin{aligned}
\|f_i(\bar{x}) - f_i(\bar{y}^i)\|_{(i)} &\leq w_i\, \alpha \|\bar{x} - \bar{y}^i\|^w_\infty, \\
&\leq w_i\, \alpha \max_{j=1}^{n} \frac{\|\bar{y}^j_j - \bar{y}^i_j\|_{(i)}}{w_j}, \\
&< w_i\, \alpha \max_{j=1}^{n} \frac{m_j}{w_j} \varepsilon.
\end{aligned}$$

It follows

$$\begin{aligned}
\|f(\bar{x}) - \bar{x}\|^w_\infty &\leq \|g(\bar{y}) - \bar{x}\|^w_\infty + \max_{i=1}^{n} \frac{\|f_i(\bar{x}) - f_i(\bar{y}^i)\|_{(i)}}{w_i}, \\
&< \|g(\bar{y}) - \bar{x}\|^w_\infty + \alpha \max_{j=1}^{n} \frac{m_j}{w_j} \varepsilon.
\end{aligned}$$

Accounting $m_j \leq \eta$ and $\alpha < 1$, we deduce

$$\|f(\bar{x}) - \bar{x}\|^w_\infty < \|g(\bar{y}) - \bar{x}\|^w_\infty + \eta \max_{i=1}^{n} \frac{1}{w_i} \varepsilon.$$

Then, by ensuring $\|g(\bar{y}) - \bar{x}\|^w_\infty \leq \varepsilon$, and taking

$$\varepsilon = \frac{\varepsilon'}{1 + \eta \min_{i=1}^{n} w_i},$$

we conclusively satisfy

$$\|f(\bar{x}) - \bar{x}\|^w_\infty < \varepsilon + \eta \min_{i=1}^{n} w_i \varepsilon, < \varepsilon'.$$

□

6 Numerical Results

6.1 Problem and Experimental Settings

We are now interested in showing some experimental behavior of such asynchronous iterations snapshot protocols. For that, we consider the convection-diffusion problem

$$\frac{\partial u}{\partial t} - \nu \Delta u + \vec{a}.\nabla u = s, \quad t \in \mathbb{R}^+,$$

where u and s are functions defined on $\mathbb{R}^+ \times ([0,1])^3$. Conditions and parameters are set to arbitrary values

$$\begin{cases}
u(0, x, y, z) & = & 0, \quad \forall x, y, z \in (0, 1), \\
u(t, x, y, z) & = & 0, \quad \forall x, y, z \in \{0, 1\}, \forall t \in \mathbb{R}^+, \\
\nu & = & 0.5, \\
\vec{a} & = & (0.1, -0.2, 0.3),
\end{cases}$$

just as the function s given by

$$s(t, x, y, z) = \sin(x) \sin(y) \sin(z),$$

$\forall x, y, z \in [0, 1]$, $\forall t \in \mathbb{R}^+$. By using a finite-difference discretization and the backward Euler integration scheme, we obtain a sparse linear system

$$\mathcal{A} U^{t_i} = B^{t_i, t_{i-1}},$$

with $U^{t_i}, B^{t_i,t_{i-1}} \in \mathbb{R}^m$, $m \in \mathbb{N}$, at each time $t_i \in \mathbb{R}^+$, $i \in \mathbb{N}^*$, $t_0 = 0$, for which we find an approximated solution $\widetilde{U}^{t_i}$ by means of successive relaxations of the form

$$U^{t_i, k+1} := \mathcal{M}^{-1} \mathcal{N} U^{t_i, k} + \mathcal{M}^{-1} B^{t_i, t_{i-1}},$$

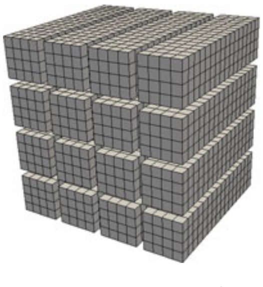

Fig. 4. Domain discretization and partitioning (16 sub-domains).

with $k \in \mathbb{N}$, and $\mathcal{A} = \mathcal{M} - \mathcal{N}$ being a convergent splitting. While plenty of parallel executions were conducted using synchronous and asynchronous iterations k, we comment here only few of them which however accurately represent the overall results. Fig. 4 illustrates the geometrical discretization and distribution of the domain $([0,1])^3$ over parallel processes. Each process handles exactly one sub-domain, and the number of processes always equals the number of processor cores used. Most of the simulations have been run for 5 time steps of size $\Delta t = 0.01$. We implemented the synchronous and asynchronous iterative methods using JACK [31], our MPI-based communication library where we additionally introduced the various convergence detection methods.

6.2 Effectiveness

First experiments are led on a cluster of 68 nodes SGI Altix ICE 8400 LX with Quad Data Rate (QDR) Infiniband interconnect (40 Gbit/s). Each node consists of two 6-cores Intel Xeon X5650 Central Processing Units (CPU) at 2.66 GHz, and 21 GB Random Access Memory (RAM) allocated to parallel jobs. The Message Passing Interface (MPI) library SGI-MPT is loaded as communication middleware.

On practical aspects, we make few remarks about the proposed methods. First, non-FIFO AIS protocols 1 (NFAIS1) and 2 (NFAIS2) are very close to AIS protocols 1 (AIS1) and 2 (AIS2), respectively, and differ only on the content of the marker. Furthermore, AIS2 turns out to be a particular instance of the non-FIFO AIS protocol 5 (NFAIS5), when one consider $\eta = 0$. Second, the non-FIFO AIS protocol 4 (NFAIS4) is a generalization of NFAIS5, based on a behavior not likely to occur in most single-site high performance computing platforms. At last, the non-FIFO AIS protocol 3 (NFAIS3) is designed for very specific circumstances where markers exchange in NFAIS2 is to be avoided. Table 1 thus summarizes accuracy results of NFAIS1, NFAIS2 and NFAIS5, which are similar to the other AIS protocols. Not surprisingly, as shown by these test cases, we did not face premature termination for any of our simulation runs. It is even noticeable that for any of the featured termination methods, the final residual tends to revolve around 5.5e-7 (±1e-7), regardless of both the number of processor cores and the size of the linear system. Such an experimental behavior strengthen the reliability of our protocols. Yet, compared to synchronous iterations which terminate at 8.3e-7, a few delay of 2.8e-7 (±1e-7) is introduced, however, as we shall see in the sequel, this does not prevent asynchronous iterations from terminating earlier than synchronous ones, in terms of execution time.

TABLE 1
Effectiveness of AIS Protocols, with Residual Threshold Set to 1e-6

		Sync. iter.		NFAIS1	
n	$\sqrt[3]{m}$	min r_i	max r_i	min r_i	max r_i
48	150	8.3e-7	8.3e-7	4.6e-7	6.9e-7
120	150	8.3e-7	8.3e-7	3.3e-7	5.0e-7
240	150	8.3e-7	8.3e-7	4.6e-7	5.6e-7
240	180	8.3e-7	8.3e-7	4.8e-7	6.5e-7
360	180	8.3e-7	8.3e-7	4.6e-7	5.5e-7
504	180	8.3e-7	8.3e-7	4.6e-7	5.8e-7
		NFAIS2		NFAIS5	
n	$\sqrt[3]{m}$	min r_i	max r_i	min r_i	max r_i
48	150	5.4e-7	6.7e-7	5.2e-7	6.1e-7
120	150	4.6e-7	6.1e-7	5.2e-7	6.5e-7
240	150	3.8e-7	6.3e-7	4.8e-7	6.2e-7
240	180	4.5e-7	5.6e-7	4.7e-7	7.2e-7
360	180	5.0e-7	5.6e-7	4.3e-7	6.4e-7
504	180	4.8e-7	5.5e-7	5.5e-7	5.9e-7

$r_i = \|\mathcal{A}\widetilde{U}^{t_i} - B^{t_i,t_{i-1}}\|_\infty$, $\widetilde{U}^{t_i}, B^{t_i,t_{i-1}} \in \mathbb{R}^m$.
n : number of processors.

TABLE 2
Efficiency of AIS Protocols, for 5 Time Step Resolutions, 180^3 Unknowns and 504 Cores

method	time	mean #it.	mean #ss	mean r_i
Sync. iter.	806	131,867	131,867	8.3e-7
SB96 [21]	651	182,004	13	8.5e-7
BCVC08 [29]	642	180,782	8	5.3e-7
NFAIS1	626	176,203	1,143	5.2e-7
NFAIS2	620	174,429	107	5.1e-7
NFAIS5	624	175,072	111	5.6e-7

time : total execution time, in seconds.
#it. : maximum number of local iterations over the set of processes.
#ss. : number of snapshots.

6.3 Efficiency

Table 2 features total execution times and some mean measurements for one time step resolution. We introduce implementation of two other termination methods from [21] (SB96) and [29] (BCVC08), respectively, as described in Section 2. While discussing the effectiveness of these other methods is beyond the scope of this paper, we successfully verify that our AIS protocols do not introduce larger termination delays, regarding both execution times and maximum numbers of iterations. The maximum number of iterations over the set of processes quite well describes the resolution speed, as it produces the same ranking than execution time.

It is noticeable that our approach was more efficient despite a higher number of snapshots. Indeed, as the communication overhead cost is very low and that our methods run faster (only one reduction operation), they are more often executed to more quickly detect the actual convergence time, without impacting the iterations speed.

A part of the experiments involving much more processor cores has been conducted on another cluster of 5,040 nodes Bullx B510, also with QDR Infiniband interconnect. Each node consists of two 8-cores Intel Sandy Bridge E5-2680 CPUs at 2.7 GHz, and 64 GB RAM. The Bullxmpi

TABLE 3
Efficiency of AIS Protocols, for 5 Time Step Resolutions and More than 1,000 Cores

		Sync. iter.		SB96	
n	$\sqrt[3]{m}$	time	mean r_i	time	mean r_i
1,024	180	251	8.3e-7	132	7.0e-7
2,048	185	453	8.3e-7	195	7.7e-7
5,600	185	530	8.4e-7	112	2.9e-7
		BCVC08		AIS1	
n	$\sqrt[3]{m}$	time	mean r_i	time	mean r_i
1,024	180	126	8.8e-7	124	7.0e-7
2,048	185	185	7.4e-7	179	8.5e-7
5,600	185	108	5.3e-7	99	8.1e-7

(OpenMPI) library is used as communication middleware. Here, we present in Table 3 some results for AIS1 protocol in an environment which however does not surely satisfy the FIFO assumption. First, we see that, with such a data transfer rate, Assumption 6 could be considered for AIS1 as well, with η sufficiently small to avoid premature termination, even without implementing some adaptation of Proposition 5. Second, it turned out that this slightly weakened version of AIS1 led to final residuals much closer to synchronous iterations ones, compared to results in Table 1. At last, regarding execution times, its efficiency is confirmed, again compared to existing methods.

7 Conclusion

Asynchronous iterations raise a non-trivial convergence detection issue that has been tackled in many various ways. Very few existing termination protocols are based on the computation of a global residual error, while mostly, more or less robust heuristics have been investigated. The most prominent approaches however require to perform two reduction operations, while we managed here to achieve effective convergence detection, using only one. On practical aspects, it is noticeable that highly robust heuristics not based on global residual lead to quite intrusive, and often complicated, solutions which do not necessarily provide a substantial efficiency gain.

We proposed in this paper seven new asynchronous iterations termination methods based on global residual, under various communication models. For FIFO communication environments, we proposed two protocols, AIS1 and AIS2, which we extended as NFAIS1 and NFAIS2 to any arbitrary non-FIFO communication model. Rightly considering that FIFO communication is however essential for computation messages in parallel iterative processes, we exhibited a possible fifth protocol (NFAIS3) which avoids control messages in a context where the FIFO delivering is not guaranteed for messages of different types. This solution can however be slightly intrusive at implementation, and should be considered if marker-based non-FIFO methods are not easily applicable. We then characterized a general non-FIFO model where, on every channel (in one direction), the number of messages that a given message can cross is bounded. The arbitrary non-FIFO model actually corresponds to the particular case where this maximum number always exceeds the number of messages emitted. We showed here how strong heuristics (NFAIS4 and NFAIS5) could be used to avoid including computation data into control messages, which constitutes an improvement of NFAIS1 and NFAIS2, in terms of communication overhead costs. We formally established the reliability of these heuristics, providing a practical way of accurately setting the convergence residual threshold. Finally, experiments on supercomputers confirmed the effectiveness and efficiency of our approach versus prominent existing methods.

Acknowledgments

A part of this work was performed using HPC resources from GENCI-TGCC (Grant 2014-t2014069065).

References

[1] J. L. Rosenfeld, "A case study in programming for parallel-processors," *Commun. ACM*, vol. 12, no. 12, pp. 645–655, 1969.

[2] D. Chazan and W. Miranker, "Chaotic relaxation," *Linear Algebra Appl.*, vol. 2, no. 2, pp. 199–222, 1969.

[3] J. D. P. Donnelly, "Periodic chaotic relaxation," *Linear Algebra Appl.*, vol. 4, no. 2, pp. 117–128, 1971.

[4] J. C. Miellou, "Algorithmes de relaxation chaotique à retards," *ESAIM: Math. Model. Numerical Anal. - Modélisation Mathématique et Analyse Numérique*, vol. 9, no. R1, pp. 55–82, 1975.

[5] G. M. Baudet, "Asynchronous iterative methods for multiprocessors," *J. ACM*, vol. 25, no. 2, pp. 226–244, 1978.

[6] S. Benjelloun, P. Spitéri, and G. Authié, "Parallel algorithms for solving the obstacle problem," *Comput. Mech. Publ.*, vol. 2, pp. 275–281, 1989.

[7] P. Spitéri, J. Miellou, and D. El Baz, "Asynchronous Schwarz alternating method with flexible communication for the obstacle problem," *Réseaux et Systèmes Répartis - Calculateurs Parallèles*, vol. 13, no. 1, pp. 47–66, 2001.

[8] M. Chau, R. Couturier, J. M. Bahi, and P. Spiteri, "Parallel solution of the obstacle problem in grid environments," *Int. J. High Perform. Comput. Appl.*, vol. 25, no. 4, pp. 488–495, 2011.

[9] A. Üresin and M. Dubois, "Parallel asynchronous algorithms for discrete data," *J. ACM*, vol. 37, no. 3, pp. 588–606, 1990.

[10] E. D. Chajakis and S. A. Zenios, "Synchronous and asynchronous implementations of relaxation algorithms for nonlinear network optimization," *Parallel Comput.*, vol. 17, no. 8, pp. 873–894, 1991.

[11] M. Chau, P. Spiteri, and H. C. Boisson, "Parallel numerical simulation for the coupled problem of continuous flow electrophoresis," *Int. J. Numerical Methods Fluids*, vol. 55, no. 10, pp. 945–963, 2007.

[12] K. Li-Shan, C. Yu-Ping, S. Le-Lin, and Q. Hui-Yun, "The asynchronous parallel algorithms S-COR for solving P.D.E.'s on multiprocessors," *Int. J. Comput. Math.*, vol. 18, no. 2, pp. 163–172, 1985.

[13] L. Hart and S. McCormick, "Asynchronous multilevel adaptive methods for solving partial differential equations on multiprocessors: Basic ideas," *Parallel Comput.*, vol. 12, no. 2, pp. 131–144, 1989.

[14] J. Bahi, E. Griepentrog, and J. C. Miellou, "Parallel treatment of a class of differential-algebraic systems," *SIAM J. Numerical Anal.*, vol. 33, no. 5, pp. 1969–1980, 1996.

[15] M. Jarraya, "Mise en œuvre et étude de performance d'algorithmes itératifs parallèles sur diverses architectures. Application à l'optimisation, la commande et la résolution de systèmes Markoviens," Ph.D. dissertation, Institut de Recherche en Informatique de Toulouse, Université Paul Sabatier, Toulouse, France, Oct. 2000.

[16] D. P. Bertsekas and J. N. Tsitsiklis, "Some aspects of parallel and distributed iterative algorithms—a survey," *Automatica*, vol. 27, no. 1, pp. 3–21, 1991.

[17] E. W. Dijkstra and C. S. Scholten, "Termination detection for diffusing computations," *Inf. Process. Lett.*, vol. 11, no. 1, pp. 1–4, 1980.

[18] N. Francez and M. Rodeh, "Achieving distributed termination without freezing," *IEEE Trans. Softw. Eng.*, vol. SE-8, no. 3, pp. 287–292, May 1982.

[19] S. P. Rana, "A distributed solution of the distributed termination problem," *Inf. Process. Lett.*, vol. 17, no. 1, pp. 43–46, 1983.

[20] F. Mattern, "Algorithms for distributed termination detection," *Distrib. Comput.*, vol. 2, no. 3, pp. 161–175, 1987.

[21] S. A. Savari and D. P. Bertsekas, "Finite termination of asynchronous iterative algorithms," *Parallel Comput.*, vol. 22, no. 1, pp. 39–56, 1996.

[22] D. El Baz, "A method of terminating asynchronous iterative algorithms on message passing systems," *Parallel Algorithms Appl.*, vol. 9, no. 1/2, pp. 153–158, 1996.

[23] M. Chau, "Algorithmes parallèles asynchrones pour la simulation numérique," Ph.D. dissertation, Institut National Polytechnique de Toulouse, Toulouse, France, Nov. 2005.

[24] D. J. Evans and S. Chikohora, "Convergence testing on a distributed network of processors," *Int. J. Comput. Math.*, vol. 70, no. 2, pp. 357–378, 1998.

[25] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 1, pp. 63–75, 1985.

[26] A. D. Kshemkalyani, M. Raynal, and M. Singhal, "An introduction to snapshot algorithms in distributed computing," *Distrib. Syst. Eng.*, vol. 2, no. 4, pp. 224–233, 1995.

[27] J. M. Bahi, S. Contassot-Vivier, R. Couturier, and F. Vernier, "A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 1, pp. 4–13, Jan. 2005.

[28] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann, 1996.

[29] J. M. Bahi, S. Contassot-Vivier, and R. Couturier, "An efficient and robust decentralized algorithm for detecting the global convergence in asynchronous iterative algorithms," in *Proc. Int. Conf. High Perform. Comput. Comput. Sci.*, 2008, pp. 240–254.

[30] J. Miellou, P. Spiteri, and D. ElBaz, "A new stopping criterion for linear perturbed asynchronous iterations," *J. Comput. Appl. Math.*, vol. 219, no. 2, pp. 471–483, 2008.

[31] F. Magoulès and G. Gbikpi-Benissan, "JACK: An asynchronous communication kernel library for iterative algorithms," *J. Supercomput.*, vol. 73, no. 8, pp. 3468–3487, 2017.

Frédéric Magoulès received the BSc degree in engineering sciences, the MSc degree in applied mathematics, the MSc degree in numerical analysis, and the PhD degree in applied mathematics from the Université Pierre et Marie Curie, France, in 1993, 1994, 1995, and 2000, respectively. He is currently a professor in the Department of Mathematics and in the Department of Computer Science, CentraleSupélec, Université Paris-Saclay, France. His research interests include parallel computing and iterative methods. He is a fellow of the IMA, a fellow of the BCS, a member of the ACM, a member of the SIAM, and a member of the IEEE.

Guillaume Gbikpi-Benissan received the MSc degree in computer science from Ecole Supérieure Polytechnique, Université Cheikh Anta Diop, Sénégal and the MSc degree in computer science from the Université Pierre et Marie Curie, France, in 2009 and 2012, respectively. He is currently working toward the PhD degree at CentraleSupélec, Université Paris-Saclay, France. His research interests include parallel computing, distributed algorithms, and iterative methods.