Advances in Engineering Software 146 (2020) 102827

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Protocol-free asynchronous iterations termination

Guillaume Gbikpi-Benissan[*,a], Frédéric Magoulès[b,c]

[a] Engineering Academy, Peoples' Friendship University of Russia (RUDN University), 6 Miklukho-Maklaya St, Moscow, 117198, Russian Federation
[b] CentraleSupélec, Université Paris-Saclay, 3 rue Joliot-Curie, Gif-sur-Yvette 91190, France
[c] Faculty of Engineering and Information Technology, University of Pécs, H-7622 Pécs, Vasvári Pál utca 4, Hungary

ARTICLE INFO

Keywords:
Asynchronous iterations
Convergence detection

ABSTRACT

In this paper, we tackled the convergence detection problem arisen from the absence of synchronization during asynchronous iterative computation. We showed that, when one arbitrarily takes the local components of a global solution vector, an upper bound can be established on the difference between a residual error evaluated from this global vector and the inconsistent residual error evaluated without synchronizing the involved computing processes. This allows for accurate termination of asynchronous iterations without implementing any particular detection protocol. Termination delay has be handled too for not slowing down the overall asynchronous solver, by appropriately setting the convergence threshold criterion. We therefore ensured effectiveness while reaching better efficiency in terms of overall execution time of the solver, in comparison with the current most efficient exact snapshot-based approach.

1. Introduction

Asynchronous iterative methods [1,2] constitute a class of parallel iterative algorithms where processing units are free to iterate on their local data at their own pace, independently from the frequency of interface data exchange. They therefore constitute so far the sole parallel iterative methods where iterations are actually not sequential, due to their mathematical formulation inherently allowing for non-deterministic sequences of iterations. Due to such a property, they are becoming more and more popular in the field of high-performance scientific computing, where communication between processing units is one of the major performance issues.

To take full advantage of the asynchronous iterative model, synchronization between processing units is avoided. It follows that no globally consistent solution vector is directly accessible during the computation, and therefore, detecting the moment when an asynchronous iterative procedure reaches convergence constitutes an issue to be particularly handled in the most both effective and efficient way. First approaches [2–4] suggested to add few restrictions on the computational model such that the iterative procedure terminates on its own when significant changes do not occur anymore in any of the local processes. Then, termination is detected through classical distributed termination algorithms [5,6]. These approaches require additional assumptions on the iterative model and has got the main disadvantage of being highly intrusive. At the opposite side, Evans and Chikohora [7] devised a method for approximately predict the number of iterations necessary for reaching convergence, which therefore requires no convergence detection protocol at all. Still, actual convergence needs to be checked after the given number of iterations, which could result in either several global synchronization operations or delayed termination. A more recent approach from Bahi et al. [8,9] consists of monitoring the persistence of local convergence of the parallel processes without altering the computational model. It yields several executions of a reduction operation in parallel of the iterative procedure, for gathering local convergence states and checking with a high probability of accuracy whether such states are permanent or not. However, local convergence of all of the processes does not necessarily implies a global convergence if these local convergence states are not carefully evaluated based on globally consistent data. From Miellou et al. [10], ideas of macro-iterations and successive evaluations of diameter of solutions nested sets provide an accurate way of evaluating these local convergence states. Still, these monitoring approaches need to resort to piggybacking of control information on top of computational messages, which also implies intrusive and quite non trivial implementation. The first distributed and non-intrusive exact algorithmic solution to the asynchronous convergence detection problem goes back to [4], in 1996, where a globally coherent solution vector can be asynchronously built and evaluated. Still, its implementation in an asynchronous iterations programming library appeared only in 2018, and is due to [11], where authors also derived a simpler and more efficient non-blocking global

[*] Corresponding author.
E-mail address: guibenissan@gmail.com (G. Gbikpi-Benissan).

https://doi.org/10.1016/j.advengsoft.2020.102827
Received 14 November 2019; Received in revised form 13 March 2020; Accepted 4 May 2020

synchronization operation for building the successive solution vectors. This introduced distributed snapshot of asynchronous iterations however needs exchange of interface data, which implies a possible communication overhead depending on sizes of interfaces. The very recent piece of work in [12] allows for avoiding such an overhead by making use of empty snapshot messages, while a knowledge of maximum possible delays in message exchange is assumed.

One can see that research on asynchronous convergence detection features some sparsity in time, with main periods of activity having been around 1989–1996 and 2005–2008. Despite the fact that only few new results appeared recently in 2018, a lot of attention is still paid to the development of asynchronous iterations themselves, regardless of the convergence detection issue (see, e.g., [13–17]). More or less complete surveys can be found, e.g., in [18–20]. Despite such advances, implementation of asynchronous iterations in engineering software would remain limited if the accuracy of the calculated solution cannot be guaranteed by an effective convergence detection mechanism. On the other hand, time saving and synchronization avoidance remain important goals, so that it is still of interest to reduce as much as possible the detection delay to also reach efficient convergence detection.

This paper is based on [21] where authors experimentally evaluated a generalization of the approximate residual error computation proposed by Magoulès and Gbikpi-Benissan [12], resulting in detecting the convergence state of an asynchronous iterative procedure without any dedicated protocol. Here, we first provide a more rigorous and precise derivation of such a protocol-free approach, and then suggest an improvement of its accuracy by more carefully evaluating local convergence.

Section 2 recalls the asynchronous convergence detection problem on a pure algorithmic aspect, as well as the exact and the approximate snapshot-based approaches. Section 3 generalizes the theoretical analysis of the approximate snapshot-based approach, then suggests an improved implementation, relatively to Gbikpi-Benissan and Magoulès [21]. Numerical experiments are finally described in Section 4.

2. Computational framework

2.1. Asynchronous convergence detection

Asynchronous iterations are given by a computational model

$$x_i^{k+1} = \begin{cases} f_i(x_1^{\tau_{i,1}(k)}, \ldots, x_p^{\tau_{i,p}(k)}), & i \in P^{(k)} \\ x_i^k, & i \in \{1, \ldots, p\} \setminus P^{(k)} \end{cases} \tag{1}$$

with

$$p \in \mathbb{N}, \qquad P^{(k)} \subseteq \{1, \ldots, p\}, \quad \tau_{i,j}(k) \le k, \quad \forall\, k \in \mathbb{N}.$$

In parallel computation context, p can be taken as the number of processes and each x_i would be a block-component of a vector x. Such a computational model describes a sequence $\{x^k\}_{k\in\mathbb{N}}$ of vectors where, at each iteration, each component of x^k is computed on the basis of arbitrarily chosen previous values. We recall two classical assumptions on asynchronous iterative computing which stand for the fact that no process stops iterating and sending updated interface data, and updated interface data keep being delivered.

Assumption 1. $\forall\, i \in \{1, \cdots, p\}, \quad \mathrm{card}\{k \in \mathbb{N} \mid i \in P^{(k)}\} = +\infty.$

Assumption 2. $\forall\, i, j \in \{1, \cdots, p\}, \quad \lim\limits_{k\to+\infty} \tau_{i,j}(k) = +\infty.$

In practice then, both the sequence $\{P^{(k)}\}_{k\in\mathbb{N}}$ and the functions $\tau_{i,j}(k)$ are freely determined by each actual execution of the iterative procedure. Therefore, one actually generates p sequences $\{x_i^{k^{(i)}}\}_{k^{(i)}\in\mathbb{N}}$ with no explicit knowledge of the relation between $k^{(i)}$ and k. While k is the global iteration variable related to the unique implicit sequence $\{x^k\}_{k\in\mathbb{N}}$, $k^{(i)}$ with $i \in \{1, \cdots, p\}$ denotes the local iteration variable on the process i, related to the explicit sequence $\{x_i^{k^{(i)}}\}_{k^{(i)}\in\mathbb{N}}$. In classical synchronous computing, we would implicitly have

$$x_i^{k^{(i)}} = (x^k)_i \quad \Leftrightarrow \quad k^{(i)} = k,$$

where $(x^k)_i$ denotes the i-th component of x^k. Considering then each of the local sequences $\{x_i^{k^{(i)}}\}_{k^{(i)}\in\mathbb{N}}$, the asynchronous iterative model (1) can be rewritten as

$$x_i^{k^{(i)}+1} = f_i\left(x_1^{k^{(1)}-d_{i,1}(k^{(i)})}, \ldots, x_p^{k^{(p)}-d_{i,p}(k^{(i)})}\right), \qquad \forall\, i \in \{1, \ldots, p\} \tag{2}$$

with

$$0 \le d_{i,j}(k^{(i)}) \le k^{(j)}, \qquad \forall\, i, j \in \{1, \ldots, p\}.$$

Regarding the global iteration variable k, $k^{(i)}$ increases each time i belongs to $P^{(k)}$. Assumption 1 is therefore ensured by considering $k^{(i)} \in \mathbb{N}$, while Assumption 2 would be rewritten as follows.

Assumption 3. $\forall\, i, j \in \{1, \cdots, p\}, \qquad \lim\limits_{k^{(i)}\to+\infty k^{(j)}\to+\infty} k^{(j)} - d_{i,j}(k^{(i)}) = +\infty.$

Let us introduce the following notation.

Notation 1. For any local iteration $k^{(i)}$ on a process $i \in \{1, \ldots, p\}$, there exist $p-1$ local iterations $k^{(j)}$ on processes $j \in \{1, \ldots, p\}, j \ne i$, such that

$$x_i^{k^{(i)}+1} = f_i\left(x_1^{k^{(1)}-d_{i,1}(k^{(i)})}, \ldots, x_p^{k^{(p)}-d_{i,p}(k^{(i)})}\right).$$

Such a set $\{k^{(j)}\}_{j\in\{1,\ldots,p\}}$ corresponding to any particular $k^{(i)}$ will be implied when using the shortest notation

$$x^{(i),k^{(i)}} := \begin{bmatrix} x_1^{k^{(1)}-d_{i,1}(k^{(i)})} & \cdots & x_p^{k^{(p)}-d_{i,p}(k^{(i)})} \end{bmatrix}^{\mathsf{T}}.$$

Then, applying Notation 1 in Equation (2), we can write

$$x_i^{k^{(i)}+1} = f_i\left(x^{(i),k^{(i)}}\right), \qquad \forall\, i \in \{1, \ldots, p\}.$$

We can see that each process i handles, at each iteration $k^{(i)}$, a global vector $x^{(i),k^{(i)}}$ where the components $x_j^{(i),k^{(i)}}$ with $j \ne i$ are received from processes j with a delay of $d_{i,j}(k^{(i)})$. In practice though, these components can only be actually needed interface data received from some processes j. For locally computed components, one can reasonably assume

$$x_i^{(i),k^{(i)}} = x_i^{k^{(i)}}.$$

Let us now consider that convergence is reached when

$$\begin{bmatrix} x_1^{k^{(1)}} & \cdots & x_p^{k^{(p)}} \end{bmatrix}^{\mathsf{T}} \in S^*$$

with S^* being a set of admissible solutions. Let also r be a convergence evaluation function such that

$$r(x) < \varepsilon \quad \Rightarrow \quad x \in S^*, \qquad \varepsilon > 0.$$

In parallel computation context, r is usually evaluated in a distributed form

$$r(x) = \sigma(r_1(x), \cdots, r_p(x)),$$

where one would take

$$r_i(x) = \|f_i(x) - x_i\|, \qquad \sigma(\alpha_1, \ldots, \alpha_p) = \left(\sum_{j=1}^{p} \alpha_j^q\right)^{1/q}, \quad q \ge 1,$$

with $\alpha_1, \ldots, \alpha_p \in \mathbb{R}$. Evaluating such a function is therefore nontrivial in

```
1: Build a global state $\overline{x} := \begin{bmatrix} x_1^{k_0^{(1)}} & \dots & x_p^{k_0^{(p)}} \end{bmatrix}^\mathsf{T}$
2: Iterate on $\overline{x}$ to compute each $f_i(\overline{x})$
3: Compute each $r_i(\overline{x})$
4: Apply the reduction operation $r(\overline{x}) := \sigma(r_1(\overline{x}), \dots, r_p(\overline{x}))$
5: return $r(\overline{x})$
```

Algorithm 1. Computing global asynchronous residual error.

a distributed asynchronous way, as it requires to identify and iterate on a unique global vector without synchronizing the set of processes.

2.2. Exact direct convergence evaluation

Direct convergence evaluation is the explicit evaluation of a relation of the form

$$r(x_1^{k^{(1)}}, \dots, x_p^{k^{(p)}}) < \varepsilon,$$

which therefore requires to follow the steps in Algorithm 1.

To constitute the global state vector $\bar{x}$, each process i, with $i \in \{1, \dots, p\}$, provides, at some local iteration $k_0^{(i)}$, its corresponding local vector $x_i^{k_0^{(i)}}$. $k_0^{(i)}$ is thus the particular value of the local iteration variable $k^{(i)}$ at which the process i records its local part of the global state $\bar{x}$. Building the global state $\bar{x}$ has been first investigated by Savari and Bertsekas [4], then simplified and improved by Magoulès and Gbikpi-Benissan [12], where authors also show the link with the more general distributed snapshot protocol [22]. Nevertheless, the differences between [4] and [12] mainly lie in determining the conditions under which the processes must start building the global state. The improvement from Algorithm 2 consisted of making a process enter the snapshot phase each time local convergence is observed, which reduced the delay of global convergence detection.

A further improvement of the detection delay is due to Listing 1, where authors even removed the local convergence condition for triggering snapshots. They investigated snapshot procedures as simple non-blocking collective operations repeated in combination with non-blocking reduction operations.

The function *Copy* at the line 8 of Listing 1 corresponds to the line 2 of Algorithm 2. The non-blocking function *My_Isnapshot* performs the lines 3 to 5 of Algorithm 2. Therefore, its completion implies that the line 1 of Algorithm 1 is performed. The function *ComputeLocalResidual* then corresponds to the lines 2 and 3 of Algorithm 1.

2.3. Approximate direct convergence evaluation

For building a global state $\bar{x}$, each process i actually builds a local version $\bar{x}^{(i)}$ from which one then takes

$$\bar{x} := [\bar{x}_1^{(1)} \ \cdots \ \bar{x}_p^{(p)}]^\mathsf{T}.$$

In case of exact convergence evaluation, the snapshot protocols therefore ensure

$$\bar{x}_j^{(i)} = \bar{x}_j^{(j)}, \qquad \forall\, i, j \in \{1, \dots, p\}, \tag{3}$$

```
1: if $\|x_i^{k^{(i)}} - x_i^{k^{(i)}-1}\| < \varepsilon$ then
2:     $\overline{x_i} := x_i^{k^{(i)}}$
3:     Send $\overline{x_i}$ to processes $j \neq i$
4:     Receive $\overline{x_j}$ from processes $j \neq i$
5:     return $\overline{x}$
6: end if
```

Algorithm 2. Non-First-In-First-Out Asynchronous Iterations Snapshot protocol 2 (NFAIS 2) [12].

so that we also satisfy

$$\bar{x} = \bar{x}^{(1)} = \dots = \bar{x}^{(p)}.$$

Algorithm 3 introduces an approximate convergence evaluation, where Equation (3) is no more satisfied.

In such a case, Algorithm 1 generalizes into Algorithm 4, where $k_i^{(j)}$ with $j \in \{1, \dots, p\}$ denotes some particular value of the local iteration variable $k^{(j)}$ of the process j, from which the process i sets the j-th component of its global state vector $\bar{x}^{(i)}$.

While Equation (3) is not necessarily satisfied, Algorithm 3 ensures that

$$\|\bar{x}_j^{(i)} - \bar{x}_j^{(j)}\| < m\varepsilon, \qquad m \in \mathbb{N},$$

where m is an assumed known upper bound on message delivery delays. For instance, if m messages are successively sent by a process i to a process j, the first message sent by i cannot be received by j after the last one. Alternatively, m can also be expressed as a number of iterations on any process i covering for sure the transfer of a message toward any other process j. The approximate snapshot protocol therefore also ensures the following result.

Proposition 1. *The approximate residual, $\tilde{r}(.)$, returned by Algorithm 4 and the exact one, $r(.)$, returned by Algorithm 1 are related by*

$$r(\bar{x}) < \tilde{r}(\bar{x}^{(1)}, \dots, \bar{x}^{(p)}) + p^{1/q} m \delta(f) \varepsilon,$$

where $\delta(f)$ is defined as

$$\delta(f) := \max_{i=1}^{p} \sum_{j=1}^{p} \delta_{i,j}(f)$$

with $\delta_{i,j}(f)$ being the smallest values satisfying

$$\|x_j - y_j\| < \varepsilon \quad \Rightarrow \quad \|f_i(x) - f_i(x_1, \dots, y_j, \dots, x_p)\| < \delta_{i,j}(f)\varepsilon.$$

Proof. See [12]. □

One can therefore guarantee actual global convergence by considering

$$\begin{aligned}
&\tilde{r}(\bar{x}^{(1)}, \dots, \bar{x}^{(p)}) && < \varepsilon, \\
&\tilde{r}(\bar{x}^{(1)}, \dots, \bar{x}^{(p)}) + p^{1/q} m \delta(f) \varepsilon && < \varepsilon + p^{1/q} m \delta(f) \varepsilon, \\
&r(\bar{x}) && < (1 + p^{1/q} m \delta(f)) \varepsilon,
\end{aligned}$$

which means that to ensure

$$r(\bar{x}) < \varepsilon,$$

one has to set the convergence threshold at

$$\frac{\varepsilon}{1 + p^{1/q} m \delta(f)}.$$

In this paper, we investigate a more general case where m is not assumed to be known and where, therefore, global states $\bar{x}^{(i)}$ can be taken arbitrarily without any snapshot protocol.

3. Protocol-free direct convergence evaluation

3.1. Generalization of the approximate direct convergence evaluation

Let us first consider the following contraction condition for convergence of fixed-point iterative methods.

Assumption 4. $\|f(x) - f(y)\|_q \le \alpha \|x - y\|_q, \quad \alpha < 1.$

The following preliminary bounds can therefore be highlighted.

Lemma 1. *Let $k_1^{(i)}, k_2^{(i)} \in \mathbb{N}$ be two local iterations on a process $i \in \{1, \dots, p\}$. Then, under Assumption 4, we have*

$$\left\| x_i^{k_1^{(i)}} - x_i^{k_2^{(i)}} \right\| \le \alpha p^{1/q} \|x^{(i),k_1^{(i)}-1} - x^{(i),k_2^{(i)}-1}\|_\infty.$$

```
reduce_end_flag = 1;
snapshot_end_flag = 1;
r = 1e-6;
while (r >= 1e-6) {
  PerformComputingAndNonBlockingDataExchange();
  if (reduce_end_flag) {
    if (snapshot_end_flag) {
      Copy(x_i, bar_x_i);
      My_Isnapshot(bar_x_i, bar_x_j, &snapshot_req);
    }
    My_Test(&snapshot_req, &snapshot_end_flag);
    if (snapshot_end_flag) {
      ComputeLocalResidual(bar_x_i, bar_x_j, &tmp_r);
      MPI_Iallreduce(MPI_IN_PLACE, &tmp_r, 1, MPI_DOUBLE, MPI_MAX,
          MPI_COMM_WORLD, &reduce_req);
      reduce_end_flag = 0;
    }
  }
  else {
    MPI_Test(&reduce_req, &reduce_end_flag, MPI_STATUS_IGNORE);
    if (reduce_end_flag) {
      r = tmp_r;
    }
  }
}
```

Fig. 1. Asynchronous iterative scheme with non-blocking synchronization [11].

```
1:  if $\|x_i^{k^{(i)}} - x_i^{k^{(i)}-1}\| < \varepsilon$ then
2:      $k_0^{(i)} := k^{(i)}$
3:  end if
4:  if $\|x_i^{k^{(i)}} - x_i^{k^{(i)}-1}\| < \varepsilon \quad \forall k^{(i)} \in \{k_0^{(i)}, \ldots, k_0^{(i)} + m\}, m \in \mathbb{N}$ then
5:      $\overline{x}_i^{(i)} := x_i^{k^{(i)}}$
6:      Send a notification to processes $j \neq i$
7:  end if
8:  if notification received from a process $j \neq i$ then
9:      $\overline{x}_j^{(i)} := x_j^{(i),k^{(i)}}$
10: end if
11: if $\forall j \in \{1, \ldots, p\}$, $\overline{x}_j^{(i)}$ is set then
12:     return $\overline{x}^{(i)}$
13: end if
```

Algorithm 3. NFAIS 5 [12].

```
1: Build global states $\overline{x}^{(i)} := \begin{bmatrix} x_1^{k_i^{(1)}} & \ldots & x_p^{k_i^{(p)}} \end{bmatrix}^{\mathsf{T}}$
2: Iterate on each $\overline{x}^{(i)}$ to compute each $f_i(\overline{x}^{(i)})$
3: Compute each $r_i(\overline{x}^{(i)})$
4: Apply the reduction operation $\widetilde{r}(\overline{x}^{(1)}, \ldots, \overline{x}^{(p)}) := \sigma(r_1(\overline{x}^{(1)}), \ldots, r_p(\overline{x}^{(p)}))$
5: return $\widetilde{r}(\overline{x}^{(1)}, \ldots, \overline{x}^{(p)})$
```

Algorithm 4. Computing approximate asynchronous residual error.

Proof. We have

$$\begin{aligned}\left\|x_i^{k_1^{(i)}} - x_i^{k_2^{(i)}}\right\| &= \|f_i(x^{(i),k_1^{(i)}-1}) - f_i(x^{(i),k_2^{(i)}-1})\| \\ &\le \|f(x^{(i),k_1^{(i)}-1}) - f(x^{(i),k_2^{(i)}-1})\|_q.\end{aligned}$$

According to Assumption 4, we therefore have

$$\begin{aligned}\left\|x_i^{k_1^{(i)}} - x_i^{k_2^{(i)}}\right\| &\le \alpha\, \|x^{(i),k_1^{(i)}-1} - x^{(i),k_2^{(i)}-1}\|_q \\ &= \alpha\left(\sum_{j=1}^{p}\left\|x_j^{k_1^{(j)}-1-d_{i,j}(k_1^{(i)}-1)} - x_j^{k_2^{(j)}-1-d_{i,j}(k_2^{(i)}-1)}\right\|^q\right)^{1/q} \\ &\le \alpha\left(p\max_{j=1}^{p}\left\|x_j^{k_1^{(j)}-1-d_{i,j}(k_1^{(i)}-1)} - x_j^{k_2^{(j)}-1-d_{i,j}(k_2^{(i)}-1)}\right\|^q\right)^{1/q} \\ &= \alpha p^{1/q}\, \|x^{(i),k_1^{(i)}-1} - x^{(i),k_2^{(i)}-1}\|_\infty,\end{aligned}$$

which concludes the proof. □

Corollary 1. *Let $k_1^{(i)}, k_2^{(i)} \in \mathbb{N}$ be two local iterations on a process $i \in \{1, ..., p\}$. Then, under Assumption 4, there exists a constant $m \in \mathbb{N}$, there exists a sequence of processes $j_1, ..., j_{m+1} \in \{1, ..., p\}$ and there exists a sequence of local iterations sets $\{k_1^{(1)}, ..., k_1^{(p)}\}, ..., \{k_{2m+2}^{(1)}, ..., k_{2m+2}^{(p)}\} \subset \mathbb{N}^p$ such that*

$$\left\|x_i^{k_1^{(i)}} - x_i^{k_2^{(i)}}\right\| \le \alpha^m p^{m/q}\left\|x_{j_{m+1}}^{k_{2m+1}^{(j_{m+1})}} - x_{j_{m+1}}^{k_{2m+2}^{(j_{m+1})}}\right\|,$$

and such that

$$k_{2m+1}^{(j_{m+1})} = 0 \quad \text{or} \quad k_{2m+2}^{(j_{m+1})} = 0.$$

Proof. Let $j_1 = i$ and consider some component $j_2 \in \{1, ..., p\}$ such that

$$\left\|x_{j_2}^{(i),k_1^{(i)}-1} - x_{j_2}^{(i),k_2^{(i)}-1}\right\| = \|x^{(i),k_1^{(i)}-1} - x^{(i),k_2^{(i)}-1}\|_\infty.$$

According to Notation 1, we more explicitly have

$$\left\|x_{j_2}^{(i),k_1^{(i)}-1} - x_{j_2}^{(i),k_2^{(i)}-1}\right\| = \left\|x_{j_2}^{k_1^{(j_2)}-1-d_{i,j_2}(k_1^{(i)}-1)} - x_{j_2}^{k_2^{(j_2)}-1-d_{i,j_2}(k_2^{(i)}-1)}\right\|.$$

Let us then define two other local iterations

$$k_3^{(j_2)} := k_1^{(j_2)} - 1 - d_{i,j_2}(k_1^{(i)} - 1), \qquad k_4^{(j_2)} := k_2^{(j_2)} - 1 - d_{i,j_2}(k_2^{(i)} - 1),$$

which satisfy

$$k_3^{(j_2)} < k_1^{(j_2)}, \qquad k_4^{(j_2)} < k_2^{(j_2)}. \tag{4}$$

The previous norm equality can now be rewritten as

$$\left\|x_{j_2}^{(i),k_1^{(i)}-1} - x_{j_2}^{(i),k_2^{(i)}-1}\right\| = \left\|x_{j_2}^{k_3^{(j_2)}} - x_{j_2}^{k_4^{(j_2)}}\right\|.$$

Applying Lemma 1, we have

$$\left\|x_{j_2}^{k_3^{(j_2)}} - x_{j_2}^{k_4^{(j_2)}}\right\| \le \alpha p^{1/q}\left\|x^{(j_2),k_3^{(j_2)}-1} - x^{(j_2),k_4^{(j_2)}-1}\right\|_\infty$$

and, therefore,

$$\begin{aligned}\left\|x_i^{k_1^{(i)}} - x_i^{k_2^{(i)}}\right\| &\le \alpha p^{1/q}\, \|x^{(i),k_1^{(i)}-1} - x^{(i),k_2^{(i)}-1}\|_\infty \\ &\le \alpha^2 p^{2/q}\left\|x^{(j_2),k_3^{(j_2)}-1} - x^{(j_2),k_4^{(j_2)}-1}\right\|_\infty.\end{aligned}$$

By recursion, we reach, for some $m \in \mathbb{N}$,

$$\left\|x_i^{k_1^{(i)}} - x_i^{k_2^{(i)}}\right\| \le \alpha^m p^{m/q}\left\|x^{(j_m),k_{2m-1}^{(j_m)}-1} - x^{(j_m),k_{2m}^{(j_m)}-1}\right\|_\infty.$$

From inequalities (4), such a recursion can be applied until, when considering $j_{m+1} \in \{1, ..., p\}$ such that

$$\left\|x_{j_{m+1}}^{(j_m),k_{2m-1}^{(j_m)}-1} - x_{j_{m+1}}^{(j_m),k_{2m}^{(j_m)}-1}\right\| = \left\|x^{(j_m),k_{2m-1}^{(j_m)}-1} - x^{(j_m),k_{2m}^{(j_m)}-1}\right\|_\infty,$$

we have either

$$k_{2m+1}^{(j_{m+1})} := k_{2m-1}^{(j_{m+1})} - 1 - d_{i,j_{m+1}}\left(k_{2m-1}^{(j_m)} - 1\right) = 0,$$

or

$$k_{2m+2}^{(j_{m+1})} := k_{2m}^{(j_{m+1})} - 1 - d_{i,j_{m+1}}\left(k_{2m}^{(j_m)} - 1\right) = 0.$$

Recalling that, according to Notation 1,

$$x_{j_{m+1}}^{(j_m),k_{2m-1}^{(j_m)}-1} = x_{j_{m+1}}^{k_{2m-1}^{(j_{m+1})}-1-d_{i,j_{m+1}}\left(k_{2m-1}^{(j_m)}-1\right)} = x_{j_{m+1}}^{k_{2m+1}^{(j_{m+1})}},$$

we conclude the proof. □

Finally, let us consider each arbitrary global state

$$\bar{x}^{(i)} := \begin{bmatrix} x_1^{k_i^{(1)}} & \cdots & x_p^{k_i^{(p)}} \end{bmatrix}^{\mathsf{T}}, \qquad i \in \{1, ..., p\},$$

where $k_i^{(j)}$ with $j \in \{1, ..., p\}$ denotes some particular value of the local iteration variable $k^{(j)}$ of the process j, from which the process i sets the j-th component of its global state vector $\bar{x}^{(i)}$. Let us also consider the unique consistent global state

$$\bar{x} := [\bar{x}_1^{(1)} \ \cdots \ \bar{x}_p^{(p)}]^{\mathsf{T}},$$

and additionally define

$$\bar{X} := [\bar{x}^{(1)} \ \ldots \ \bar{x}^{(p)}]^{\mathsf{T}}, \qquad g(\bar{X}) := [f_1(\bar{x}^{(1)}) \ \cdots \ f_p(\bar{x}^{(p)})]^{\mathsf{T}}.$$

Then, from Algorithms 1 and 4, we respectively have

$$r(\bar{x}) = \|f(\bar{x}) - \bar{x}\|_q, \qquad \tilde{r}(\bar{X}) = \|g(\bar{X}) - \bar{x}\|_q.$$

The following general bound can now be derived.

Theorem 1. *Under Assumption 4, there exists a constant $m \in \mathbb{N}$, there exist two local iterations $k_1^{(i_0)}, k_2^{(i_0)} \in \mathbb{N}$ on a process $i_0 \in \{1, ..., p\}$ such that*

$$r(\bar{x}) \le \tilde{r}(\bar{X}) + \alpha^{m+1}p^{(m+2)/q}\left\|x_{i_0}^{k_1^{(i_0)}} - x_{i_0}^{k_2^{(i_0)}}\right\|$$

and such that

$$k_1^{(i_0)} = 0 \quad \text{or} \quad k_2^{(i_0)} = 0.$$

Proof. We have

$$\begin{aligned} r(\bar{x}) &= \|f(\bar{x}) - \bar{x}\|_q \\ &= \|f(\bar{x}) - g(\bar{X}) + g(\bar{X}) - \bar{x}\|_q \\ &\le \|f(\bar{x}) - g(\bar{X})\|_q + \|g(\bar{X}) - \bar{x}\|_q \\ &= \tilde{r}(\bar{X}) + \|f(\bar{x}) - g(\bar{X})\|_q \\ &= \tilde{r}(\bar{X}) + \left(\sum_{i=1}^{p}\|f_i(\bar{x}) - f_i(\bar{x}^{(i)})\|^q\right)^{1/q}.\end{aligned}$$

From Assumption 4, we also have

$$\begin{aligned}\|f_i(\bar{x}) - f_i(\bar{x}^{(i)})\| &\le \|f(\bar{x}) - f(\bar{x}^{(i)})\|_q \\ &\le \alpha\|\bar{x} - \bar{x}^{(i)}\|_q.\end{aligned}$$

It follows that

$$\begin{aligned} r(\bar{x}) &\le \tilde{r}(\bar{X}) + \left(\sum_{i=1}^{p}\|f_i(\bar{x}) - f_i(\bar{x}^{(i)})\|^q\right)^{1/q} \\ &\le \tilde{r}(\bar{X}) + \left(\sum_{i=1}^{p}\alpha^q\, \|\bar{x} - \bar{x}^{(i)}\|_q^q\right)^{1/q} \\ &\le \tilde{r}(\bar{X}) + \left(\alpha^q p \max_{i=1}^{p}\|\bar{x} - \bar{x}^{(i)}\|_q^q\right)^{1/q}.\end{aligned}$$

Let us take $i_1, i_2 \in \{1, ..., p\}$ such that

$$\|\bar{x}-\bar{x}^{(i_1)}\|_q = \max_{i=1}^{p}\|\bar{x}-\bar{x}^{(i)}\|_q, \qquad \|\bar{x}_{i_2}-\bar{x}_{i_2}^{(i_1)}\|=\max_{i=1}^{p}\|\bar{x}_i-\bar{x}_i^{(i_1)}\|.$$

Then, we have

$$\begin{aligned} r(\bar{x}) &\le \tilde{r}(\bar{X}) + (\alpha^q p\,\|\bar{x}-\bar{x}^{(i_1)}\|_q^q)^{1/q}\\ &= \tilde{r}(\bar{X}) + \alpha p^{1/q}\,\|\bar{x}-\bar{x}^{(i_1)}\|_q \\ &= \tilde{r}(\bar{X})+\alpha p^{1/q}\big(\textstyle\sum_{i=1}^{p} \|\bar{x}_i-\bar{x}_i^{(i_1)}\|^q\big)^{1/q}\\ &\le \tilde{r}(\bar{X})+\alpha p^{1/q}p^{1/q}\|\bar{x}_{i_2}-\bar{x}_{i_2}^{(i_1)}\|\\ &= \tilde{r}(\bar{X})+\alpha p^{2/q}\|\bar{x}_{i_2}-\bar{x}_{i_2}^{(i_1)}\|.\end{aligned}$$

From definition of $\bar{x}$ and $\bar{x}^{(i)}$, we have

$$\bar{x}_{i_2} = \bar{x}_{i_2}^{(i_2)} = x_{i_2}^{k_{i_2}^{(i_2)}}, \qquad \bar{x}_{i_2}^{(i_1)} = x_{i_2}^{k_{i_1}^{(i_2)}},$$

from which we can then rewrite

$$r(\bar{x})\le \tilde{r}(\bar{X})+\alpha p^{2/q}\|x_{i_2}^{k_{i_2}^{(i_2)}} - x_{i_2}^{k_{i_1}^{(i_2)}}\|.$$

Given these two local iterations $k_{i_2}^{(i_2)}$ and $k_{i_1}^{(i_2)}$ on the process i_2, Corollary 1 leads to

$$\begin{aligned} r(\bar{x}) &\le \tilde{r}(\bar{X})+\alpha p^{2/q}\alpha^m p^{m/q}\left\|x_{j_{m+1}}^{k_{2m+1}^{(j_{m+1})}} - x_{j_{m+1}}^{k_{2m+2}^{(j_{m+1})}}\right\| \\ &= \tilde{r}(\bar{X}) + \alpha^{m+1}p^{(m+2)/q}\left\|x_{j_{m+1}}^{k_{2m+1}^{(j_{m+1})}} - x_{j_{m+1}}^{k_{2m+2}^{(j_{m+1})}}\right\|\end{aligned}$$

with

$$k_{2m+1}^{(j_{m+1})}=0 \quad\text{or}\quad k_{2m+2}^{(j_{m+1})}=0.$$

By taking $i_0 := j_{m+1}$, $k_1^{(i_0)} := k_{2m+1}^{(j_{m+1})}$ and $k_2^{(i_0)} := k_{2m+2}^{(j_{m+1})}$, we conclude the proof. □

3.2. Practical accuracy improvement

Let us consider a Message Passing Interface (MPI) implementation of a synchronous iterative procedure, given by Listing 2.

The function *Init* sets initial values $x_1^0, \dots, x_p^0$ and sets $r(x^0)$ to a temporary arbitrary value greater or equal to *1e-6*. The function *RequestInterfaceReception* invokes several *MPI_Irecv* to request the reception of interface data into the temporary buffer *tmp_x_j*. The function *Iterate* performs the iterations given by Eq. 2. In this synchronous case, it generates

$$k^{(1)} = \cdots = k^{(p)} = k, \qquad d_{i,j}(k^{(i)}) = 0, \qquad \forall\, i,j \in \{1,\dots,p\}.$$

The lines 7 and 8 evaluate the convergence detection function

$$r(x^k) := \|x^{k+1}-x^k\|_\infty.$$

The function *Swap* exchanges the pointer variables *tmp_x_j* and *x_j* to allow for reception of new interface data while iterating, without altering the interface data which are being used for iterating.

Switching now to asynchronous iterations, an implementation of Algorithm 4 based on arbitrarily inconsistent global states can be given by Listing 3.

Here, the functions *RequestInterfaceReception* and *RequestInterfaceSending* may encapsulate a more elaborate procedure for invoking functions *MPI_Irecv* and *MPI_Isend*, according to the initiation and completion of all these successive communication requests. For more details, we refer to the work in [11], as well as for an alternative way of performing non-blocking reduction operations in an MPI 2 environment.

From experiments in [21], we see that, in practice, the bound

$$\alpha^{m+1}p^{(m+2)/q}\left\|x_{i_0}^{k_1^{(i_0)}} - x_{i_0}^{k_2^{(i_0)}}\right\|$$

in Theorem 1 can be finely tuned, depending on the stability of the computing environment, such that m is implicitly taken small enough not to necessarily reach

$$k_1^{(i_0)} = 0 \quad\text{or}\quad k_2^{(i_0)} = 0.$$

Still, Listing 3 evaluates a convergence detection function

$$\tilde{r}\,(x^{(1),k^{(1)}},\dots,x^{(p),k^{(p)}}) := \left\|\begin{bmatrix} x_1^{(1),k^{(1)}+1} & \cdots & x_p^{(p),k^{(p)}+1}\end{bmatrix}^{\mathsf{T}} - \begin{bmatrix} x_1^{(1),k^{(1)}} & \cdots & x_p^{(p),k^{(p)}}\end{bmatrix}^{\mathsf{T}}\right\|_\infty .$$

Then, using the idea of macro-iterations from [10], such a function could be generalized to

$$\tilde{r}\,(x^{(1),k^{(1)}},\dots,x^{(p),k^{(p)}}) := \left\|\begin{bmatrix} x_1^{(1),k^{(1)}+m_1} & \cdots & x_p^{(p),k^{(p)}+m_p}\end{bmatrix}^{\mathsf{T}} - \begin{bmatrix} x_1^{(1),k^{(1)}} & \cdots & x_p^{(p),k^{(p)}}\end{bmatrix}^{\mathsf{T}}\right\|_\infty$$

with

$$m_i \ge 1, \qquad \forall\, i \in \{1,\dots,p\},$$

```
1   Init(x_i, x_j, &r);
2   while (r >= 1e-6) {
3     RequestInterfaceReception(tmp_x_j);
4     Copy(x_i, prev_x_i);
5     Iterate(x_i, x_j);
6     RequestInterfaceSending(x_i);
7     ComputeDifferenceMax(x_i, prev_x_i, &r);
8     MPI_Allreduce(MPI_IN_PLACE, &r, 1, MPI_DOUBLE, MPI_MAX,
          MPI_COMM_WORLD);
9     WaitForRequestedReception();
10    WaitForRequestedSending();
11    Swap(tmp_x_j, x_j);
12  }
```

Fig. 2. MPI-based synchronous iterations.

```
Init(x_i, x_j, &r);
tmp_r = r;
MPI_Iallreduce(MPI_IN_PLACE, &tmp_r, 1, MPI_DOUBLE, MPI_MAX,
    MPI_COMM_WORLD, &reduce_req);
while (r >= 1e-6) {
  RequestInterfaceReception(x_j);
  Copy(x_i, prev_x_i);
  Iterate(x_i, x_j);
  RequestInterfaceSending(x_i);
  MPI_Test(&reduce_req, &reduce_end_flag, MPI_STATUS_IGNORE);
  if (reduce_end_flag) {
    r = tmp_r;
    ComputeDifferenceMax(x_i, prev_x_i, &tmp_r);
    MPI_Iallreduce(MPI_IN_PLACE, &tmp_r, 1, MPI_DOUBLE, MPI_MAX,
        MPI_COMM_WORLD, &reduce_req);
  }
}
```

Fig. 3. Arbitrarily inconsistent snapshot.

```
Init(x_i, x_j, &r);
tmp_r = r;
MPI_Iallreduce(MPI_IN_PLACE, &tmp_r, 1, MPI_DOUBLE, MPI_MAX,
    MPI_COMM_WORLD, &reduce_req);
Copy(x_i, prev_x_i);
while (r >= 1e-6) {
  RequestInterfaceReception(x_j);
  Iterate(x_i, x_j);
  RequestInterfaceSending(x_i);
  MPI_Test(&reduce_req, &reduce_end_flag, MPI_STATUS_IGNORE);
  if (reduce_end_flag) {
    r = tmp_r;
    ComputeDifferenceMax(x_i, prev_x_i, &tmp_r);
    MPI_Iallreduce(MPI_IN_PLACE, &tmp_r, 1, MPI_DOUBLE, MPI_MAX,
        MPI_COMM_WORLD, &reduce_req);
    Copy(x_i, prev_x_i);
  }
}
```

Fig. 4. Robust arbitrarily inconsistent snapshot.

which improves its robustness by the possibility to take values m_i large enough to cover at least one macro-iteration. We highlight the fact that too large values would result in important termination delays. Still, ensuring macro-iterations would also result in applying a global control protocol. Instead, one natural way to arbitrarily take values m_i which dynamically adapt to the stability of the environment is given by Listing 4.

Here, then, the function *Copy* is called only at the end of each reduction operation. Therefore, on each process i, and depending on the communication speed of the environment, including temporary perturbations, m_i iterations separate the current local solution from the previous one used to evaluate *residual*. In the next section, we experimentally evaluate this more robust protocol-free approach.

4. Numerical experiments

4.1. Problem and experimental settings

Experimental investigation is led as in [21], using the JACK2 [11,23] asynchronous programming library within a scientific code solving a convection-diffusion problem of the form

$$\frac{\partial u}{\partial t} - \nu \Delta u + a.\,\nabla u = s,$$

where u is the unknown function defined on $\mathbb{R}^+ \times ([0, 1])^3$. Both second-order central finite-differences and Backward Euler schemes are used for spatial and time integration, respectively. We are interested in the resolution of linear systems of the form

$$Ax = b, \qquad x \in \mathbb{R}^n,$$

which occur at each time step. The computational environment consists of an SGI ICE X supercomputer with each node consisting of two 12-cores Intel Haswell Xeon CPUs at 2.30 GHz and 48 GB RAM allocated. Always mapping one MPI process onto one processor core, the experiments were led with up to 25 nodes, which corresponds to 600 processor cores. Nodes are interconnected through an FDR Infiniband network (56 Gb/s). The SGI-MPT middleware is loaded for MPI communication. From several executions, we report, in the next section, values obtained about actual final convergence state given by

$$r^* = \|A\tilde{x}^* - b\|_\infty,$$

where $\tilde{x}^*$ is the computed solution, about overall execution time of each solver, in seconds, and about maximum numbers of iterations over the sets of MPI processes, denoted as $k_{\max}$. We compare the straightforward implementation of the protocol-free asynchronous iterations termination (PFAIT) corresponding to Listing 3, the more robust PFAIT (rPFAIT) corresponding to Listing 4, and the condition-free non-blocking synchronization (NBS) protocol corresponding to Listing 1which was shown in [11] to be the most efficient snapshot-based termination protocol.

4.2. Numerical results

Table 1 reports minimum and maximum final residual errors observed over all of the executions launched, for an error threshold set at 10^{-6}, and a problem of size $n = 150^3$. Table 2 shows corresponding average execution times. On this small-size problem, the effectiveness and efficiency of each termination approach can be analyzed for a better tuning of the convergence detection based on inconsistent snapshot. While the PFAIT allowed here for obtaining the fastest solver, it did not guarantee effective convergence at the desire precision threshold. We saw in the preliminary experimental work [21] that this threshold should be set at 10^{-7} to reliably expect a precision of at most 10^{-6}. As expected, the rPFAIT showed more robustness, however it featured an important termination delay, which made him very slightly less efficient than the NBS, in terms of overall execution time of the solver. In such a case, the residual error threshold can be increased in order to compensate this delay and terminate earlier while keeping ensuring a precision of at most 10^{-6}. Table 3 reports minimum and maximum final residual errors, for error thresholds set at 5×10^{-6} for the rPFAIT, 10^{-7} for the PFAIT, and 10^{-6} for the NBS. Table 4 shows corresponding average execution times. In this new configuration

Table 1
Final global residual errors for $\varepsilon = 10^{-6}$ and $n = 150^3$.

	rPFAIT (Listing 4)		PFAIT (Listing 3)		NBS (Listing 1)	
p	min r^*	max r^*	min r^*	max r^*	min r^*	max r^*
48	8.41e−08	1.03e−07	1.00e−06	1.21e−06	6.76e−07	7.51e−07
96	6.09e−08	7.34e−08	1.00e−06	1.33e−06	4.67e−07	6.16e−07
144	6.08e−08	6.59e−08	8.70e−07	1.29e−06	4.43e−07	5.46e−07
192	3.89e−08	5.82e−08	9.68e−07	1.59e−06	5.07e−07	6.29e−07
240	4.02e−08	4.86e−08	1.00e−06	1.25e−06	4.78e−07	5.20e−07
360	4.26e−08	4.57e−08	1.00e−06	1.80e−06	5.40e−07	5.74e−07
480	3.87e−08	5.84e−08	1.18e−06	1.86e−06	4.82e−07	6.23e−07
600	3.42e−08	4.38e−08	1.04e−06	1.48e−06	4.11e−07	5.37e−07

PFAIT: protocol-free asynchronous iterations termination rPFAIT: robust PFAIT NBS: non-blocking synchronization

Table 2
Wall-clock execution times for $\varepsilon = 10^{-6}$ and $n = 150^3$.

	rPFAIT (Listing 4)		PFAIT (Listing 3)		NBS (Listing 1)	
p	time (sec)	$k_{\max}$	time (sec)	$k_{\max}$	time (sec)	$k_{\max}$
48	67	25217	61	20889	67	23095
96	36	28291	31	22583	34	24853
144	24	28279	21	23104	23	25402
192	18	29377	16	23396	17	25603
240	15	28558	13	22848	14	24565
360	10	32793	9	25935	9	27689
480	8	34451	7	26537	7	28575
600	7	33556	5	25419	6	27526

Table 3
Final global residual errors for $\varepsilon = 5 \times 10^{-6}$, $\varepsilon = 10^{-7}$ and $\varepsilon = 10^{-6}$, respectively, and $n = 150^3$.

	rPFAIT (Listing 4)		PFAIT (Listing 3)		NBS (Listing 1)	
p	min r^*	max r^*	min r^*	max r^*	min r^*	max r^*
48	3.97e−07	7.55e−07	9.78e−08	1.17e−07	6.76e−07	7.51e−07
96	3.70e−07	4.02e−07	9.32e−08	9.55e−08	4.67e−07	6.16e−07
144	2.64e−07	3.74e−07	9.12e−08	1.50e−07	4.43e−07	5.46e−07
192	2.29e−07	3.23e−07	1.05e−07	3.62e−07	5.07e−07	6.29e−07
240	2.22e−07	2.72e−07	1.00e−07	1.29e−07	4.78e−07	5.20e−07
360	1.96e−07	3.65e−07	1.01e−07	1.98e−07	5.40e−07	5.74e−07
480	1.74e−07	2.64e−07	1.25e−07	2.16e−07	4.82e−07	6.23e−07
600	1.43e−07	1.65e−07	1.45e−07	1.92e−07	4.11e−07	5.37e−07

Table 4
Wall-clock execution times for $\varepsilon = 5 \times 10^{-6}$, $\varepsilon = 10^{-7}$ and $\varepsilon = 10^{-6}$, respectively, and $n = 150^3$.

	rPFAIT (Listing 4)		PFAIT (Listing 3)		NBS (Listing 1)	
p	time (sec)	$k_{\max}$	time (sec)	$k_{\max}$	time (sec)	$k_{\max}$
48	59	22136	71	24568	67	23095
96	31	24413	37	26518	34	24853
144	21	25040	24	27084	23	25402
192	16	26365	18	27441	17	25603
240	13	26003	15	27017	14	24565
360	10	30259	11	30522	9	27689
480	7	30649	8	30739	7	28575
600	6	30015	7	29934	6	27526

Table 5
Final global residual errors for $\varepsilon = 5 \times 10^{-6}$, $\varepsilon = 10^{-7}$ and $\varepsilon = 10^{-6}$, respectively, and $n = 185^3$.

	rPFAIT (Listing 4)		PFAIT (Listing 3)		NBS (Listing 1)	
p	min r^*	max r^*	min r^*	max r^*	min r^*	max r^*
240	2.27e−07	2.92e−07	1.07e−07	1.13e−07	5.23e−07	5.68e−07
360	1.73e−07	4.28e−07	1.06e−07	1.54e−07	5.29e−07	5.39e−07
480	1.73e−07	2.82e−07	1.22e−07	1.57e−07	5.00e−07	5.05e−07
600	2.12e−07	2.86e−07	1.59e−07	2.06e−07	4.80e−07	5.22e−07

where effectiveness was guaranteed for all of the three termination methods, we can observe that the rPFAIT featured the best efficiency in terms of termination delay, and therefore provided the faster solver.

At last, Table 5 and Table 6 confirm these results on a relatively large − size problem with $n = 185^3$.

5. Conclusion

We can see in this paper that robust asynchronous convergence detection can be achieved without the need of implementing any

Table 6
Wall-clock execution times for $\varepsilon = 5 \times 10^{-6}$, $\varepsilon = 10^{-7}$ and $\varepsilon = 10^{-6}$, respectively, and $n = 185^3$.

	rPFAIT (Listing 4)		PFAIT (Listing 3)		NBS (Listing 1)	
p	time (sec)	$k_{\max}$	time (sec)	$k_{\max}$	time (sec)	$k_{\max}$
240	355	370007	395	382273	368	356621
360	245	374634	265	375228	247	350270
480	195	420111Z	213	425962	199	397352
600	166	458669	178	455426	167	426655

distributed protocol, but with an appropriate approach of what could be seen as implying local convergence. Considering a standard single-site supercomputer with Infiniband interconnection between nodes, we compared the performance of two protocol-free detection methods and a highly efficient exact snapshot-based approach. A main theoretical result about the protocol-free approach was first shown, which consists of bounding the difference between an exact residual error from a consistent snapshot and the inconsistent computed one from arbitrary snapshot. The existence of such a bound was then experimented and confirmed on both a trivial and a more robust implementations of the protocol-free asynchronous iterations termination. On efficiency aspects, we saw that the robust implementation introduces a higher termination delay, however, setting the residual error threshold accordingly allowed us to reach a faster solver, compared to the efficient exact snapshot-based method. Further experiments still need to be conducted on other types of computing platforms.

CRediT authorship contribution statement

Guillaume Gbikpi-Benissan: Conceptualization, Methodology, Software, Formal analysis, Investigation, Writing - original draft, Writing - review & editing. **Frédéric Magoulès:** Supervision, Funding acquisition.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This paper has been prepared with the support of the “RUDN University Program 5-100”, the French national program LEFE/INSU, the project ADOM (Méthodes de décomposition de domaine asynchrones) of the French National Research Agency (ANR).

References

[1] Chazan D, Miranker W. Chaotic relaxation. Linear Algebra Appl 1969;2(2):199–222.
[2] Bertsekas DP, Tsitsiklis JN. Parallel and distributed computation: numerical methods. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.; 1989.
[3] Bertsekas DP, Tsitsiklis JN. Convergence rate and termination of asynchronous iterative algorithms. Proceedings of the 3rd international conference on supercomputing, 5–9 June 1989, Crete, Greece. 1989. p. 461–70.
[4] Savari SA, Bertsekas DP. Finite termination of asynchronous iterative algorithms. Parallel Comput 1996;22(1):39–56.
[5] Rana SP. A distributed solution of the distributed termination problem. Inf Process Lett 1983;17(1):43–6.
[6] Mattern F. Algorithms for distributed termination detection. Distrib Comput 1987;2(3):161–75.
[7] Evans DJ, Chikohora S. Convergence testing on a distributed network of processors. Int J Comput Math 1998;70(2):357–78.
[8] Bahi JM, Contassot-Vivier S, Couturier R, Vernier F. A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms. IEEE Trans Parallel Distrib Syst 2005;16(1):4–13.
[9] Bahi JM, Contassot-Vivier S, Couturier R. An efficient and robust decentralized algorithm for detecting the global convergence in asynchronous iterative algorithms. High performance computing for computational science - VECPAR 2008. Lecture Notes in Computer Science 5336. Springer Berlin Heidelberg; 2008. p. 240–54.
[10] Miellou J, Spiteri P, El Baz D. A new stopping criterion for linear perturbed asynchronous iterations. J Comput Appl Math 2008;219(2):471–83.
[11] Magoulès F, Gbikpi-Benissan G. JACK2: An MPI-based communication library with non-blocking synchronization for asynchronous iterations. Adv Eng Softw 2018;119:116–33.
[12] Magoulès F, Gbikpi-Benissan G. Distributed convergence detection based on global residual error under asynchronous iterations. IEEE Trans Parallel Distrib Syst 2018;29(4):819–29.
[13] Bethune I, Bull JM, Dingle NJ, Higham NJ. Performance analysis of asynchronous Jacobi's method implemented in MPI, SHMEM and OpenMP. Int J High Perform Comput Appl 2014;28(1):97–111.
[14] Magoulès F, Szyld DB, Venet C. Asynchronous optimized Schwarz methods with and without overlap. Numer Math 2017;137(1):199–227.
[15] Magoulès F, Venet C. Asynchronous iterative sub-structuring methods. Math Comput Simulation 2018;145(Supplement C):34–49.
[16] Magoulès F, Gbikpi-Benissan G. Asynchronous parareal time discretization for partial differential equations. SIAM J Sci Comput 2018;40(6):C704–25.
[17] Glusa C, Ramanan P, Boman EG, Chow E, Rajamanickam S. Asynchronous one-level and two-level domain decomposition solvers. CoRR 2018;abs/1808.08172.
[18] Frommer A, Szyld DB. On asynchronous iterations. J Comput Appl Math 2000;123(1–2):201–16.
[19] Bahi JM, Contassot-Vivier S, Couturier R. Parallel iterative algorithms: from sequential to grid computing. Boca Raton, FL: Chapman & Hall/CRC; 2007.
[20] Spiteri P. Synthetic presentation of iterative asynchronous parallel algorithms. In: Iványi P, Topping B, editors. Proceedings of the sixth international conference on parallel, distributed, GPU and cloud computing for engineering. Stirlingshire, UK: Civil-Comp Press; 2019.
[21] Gbikpi-Benissan G, Magoulès F. Distributed asynchronous convergence detection without detection protocol. In: Iványi P, Topping B, editors. Proceedings of the sixth international conference on parallel, distributed, GPU and cloud computing for engineering. Stirlingshire, UK: Civil-Comp Press; 2019.
[22] Chandy KM, Lamport L. Distributed snapshots: determining global states of distributed systems. ACM Trans Comput Syst 1985;3(1):63–75.
[23] Magoulès F, Gbikpi-Benissan G. JACK: An asynchronous communication kernel library for iterative algorithms. J Supercomput 2017;73(8):3468–87.