Méthodes asynchrones de décomposition de domaine pour le calcul massivement parallèle

Tete Guillaume Gbikpi Benissan

NNT : 2017SACLC071

Thèse de doctorat de l'Université Paris-Saclay préparée à CentraleSupélec

Ecole doctorale n°573
INTERFACES
Approches Interdisciplinaires
Fondements, Applications et Innovation
Spécialité de doctorat: Informatique

par

M. Guillaume Gbikpi-Benissan

Asynchronous domain decomposition methods for massively parallel computing

Thèse présentée et soutenue à Palaiseau, le 18 décembre 2017.

Composition du Jury :

M.	Stéphane Vialle	Professeur GeorgiaTech	(Président du jury)
M.	Raphaël Couturier	Professeur Institut FEMTO-ST	(Rapporteur)
M.	Pierre Spiteri	Professeur IRIT	(Rapporteur)
Mme	Fabienne Jezequel	Maître de Conférences LIP6	(Examinatrice)
M.	Frédéric Magoulès	Professeur MICS	(Directeur de thèse)

NNT : 2017SACLC071

Thèse de doctorat de l'Université Paris-Saclay préparée à CentraleSupélec

Ecole doctorale n°573
INTERFACES
Approches Interdisciplinaires
Fondements, Applications et Innovation
Spécialité de doctorat: Informatique

par

M. Guillaume Gbikpi-Benissan

Asynchronous domain decomposition methods for massively parallel computing

Thèse présentée et soutenue à Palaiseau, le 18 décembre 2017.

Composition du Jury :

M.	Stéphane Vialle	Professeur GeorgiaTech	(Président du jury)
M.	Raphaël Couturier	Professeur Institut FEMTO-ST	(Rapporteur)
M.	Pierre Spiteri	Professeur IRIT	(Rapporteur)
Mme	Fabienne Jezequel	Maître de Conférences LIP6	(Examinatrice)
M.	Frédéric Magoulès	Professeur MICS	(Directeur de thèse)

Acknowledgments

This PhD thesis is the outcome of a three-years period of intense work and learning, continuously fed by passion, over frustration. Here yet, some key people should be mentioned as well, as having been, at some extent, essential parts of my way toward this achievement.

I would first like to thank my supervisor at the MICS laboratory of CentraleSupélec, F. Magoulès, who gave me the opportunity of getting involved in such research activities. He devotedly brought me in facing more and more interesting challenges, which increasingly pulled out the best of me.

I would also like to thank my tutor in the APA project at IRT SystemX, M. Chau, for having patiently guided me in the understanding of various subtopics, and his early invaluable help in some implementation tasks.

Many thanks to the examiners of this dissertation, R. Couturier, F. Jezequel, P. Spiteri and S. Vialle, for having taken time to scan and evaluate the work, and for their interesting insights into various related matters.

Many thanks to my research colleagues, not only for the very helpful technical discussions, but also for that "philosophical" debates, which made me improve some ideas more than they can imagine. I am thinking of A.-K. Cheik Ahamed, H. Zhang, Q. Zou, at the MICS laboratory, and S. Gavoille in the APA project.

I would not forget my parents who, many times, literally held their breath. I am really grateful to my family members for their unswerving support.

I would finally like to thank C. Couassi for having made me actually take the very first step toward a PhD thesis, and I. Fall at ESP, Dakar, for his accurate pieces of advice which led me onto this particular path after the ESP's Engineering Degree.

Many thanks to several other friends for their either direct or indirect involvement that I am surely not forgetting.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and motivation

High performance computing (HPC) platforms are providing more and more powerful computational resources, allowing scientists to address the simulation of larger and larger modeled problems through parallel computing. Yet for a very wide class of software applications, it is still a challenge to take full advantage of massively parallel platforms. On one hand, these applications have to face a non negligible probability of resource failure, and on the other hand, they inherently feature an efficiency limit well known through the Amdahl's law [1]. About this second aspect, Figure 1.1 shows how small serial portions of an overall parallel algorithm may decrease its expected performance, while the number of processing units (processors) increases. More importantly, one can notice that there is an upper bound on how faster (speedup) we may expect to be, no matter the number of processors we use. For instance with 5% of sequential steps, we are not able to compute more than 20 times faster by using parallel machines, and actually this bound is nearly reached (19 times) with only 512 processors! An important part of the parallel scientific computing field therefore aims to reduce these serial proportions as much as possible, in

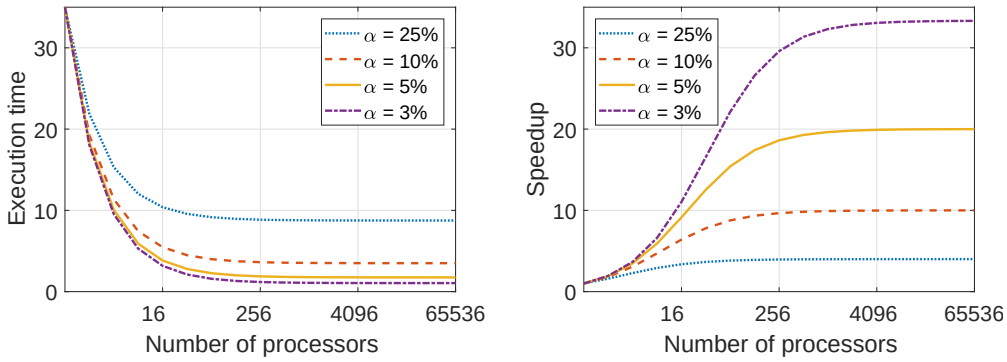


Figure 1.1: Limits of parallel computing. Illustration with α serial portion.

order to put such scalability limits off.

To design highly scalable parallel algorithms, domain decomposition methods (DDM) [2] introduce a natural approach which basically consists of

- restricting a problem to different parts of a domain on which the solution is defined,
- separately solving each resulting sub-problem,
- ensuring that the aggregation of the sub-solutions actually gives the solution of the initial global problem.

On one hand, as the sub-problems can be separately solved, the processing steps are totally parallel. On the other hand however, ensuring the global consistency of the sub-solutions (interface problem) requires data management steps, which introduces globally serial portions. These data management steps are achieved by communication between the parallel processes either via shared-memory access or through message passing, often based on a graph of data dependency. A lot of efforts is therefore made in graph partitioning [3] to minimize the inter-process dependency while ensuring a well-balanced workload between the sub-domains. Furthermore, as the interface problem is generally solved in an iterative manner, many studies propose interface conditions that improve the convergence rate of the iterative algorithm by either a continuous (see, e.g., [4, 5, 6]) or a discrete approach (see, e.g., [7, 8, 9]). All these advances significantly increase the upper bound on the expected speedup, however classical iterations remain sequential, so that the parallel processes have to complete communication steps at each iteration. The performance of such applications thus strongly depends on the performance of underlying communication platforms.

On another hand, parallel-in-time algorithms propose fully parallel time-dependent iterations. Although breaking the time-line causality appears unnatural, some attempts did succeed, like waveform relaxation methods [10, 11] which distribute the space domain of time-dependent partial differential equations (PDE) and then perform a slightly improved Picard iteration on each subproblem. Multiple shooting [12] and time-multigrid [13] methods have led to the recent Parareal method [14] and, more generally, parallel implicit time-integration algorithms (PITA) [15], where a time domain is decomposed to consider two levels of time grid for the time-integration of PDEs. Solution of the problem on the coarse grid provides initial boundary values (IBV) for each subproblem defined on the fine grid in each time sub-domain. Then parallel solution of subproblems is used for the correction of IBVs obtained from a next coarse solution. In this sense, the Parareal computational model can also be considered as an iterative predictor-corrector scheme. What is actually notable for parallel-in-time methods is that, while fully parallelizing time-dependent iterations, a second level of iterations is however required, which

on the other side remain sequential. They thus face same speedup bounds previously pointed out.

In such a context, the only way to totally break sequential computation is to resort to asynchronous iterations [16], which therefore, by removing speedup bounds and self-adapting to resource failures, could constitute so far the most interesting algorithmic approach for HPC applications. Basically, asynchronous iterations are described by a computational model where each component of the solution vector is updated on the basis of arbitrary several previous iterations, not necessarily only the last one. There is therefore no more a single sequence of iterations for approximating the whole solution, here each component is updated by following its own successive iterations. Communication can thus be completely overlapped by computation, which results in a totally parallel procedure without speedup bound, albeit such a computational model particularly exacerbates issues about convergence conditions, convergence rate and convergence detection. Several asynchronous fixed-point computational models have been proved to be convergent under various general contraction conditions (see, e.g., [17, 18, 19, 20]), particularly implied by some practical properties of the equations being solved (see, e.g., [21, 22, 23, 24]). Despite lower convergence rates in number of iterations, their practical efficiency in execution times over classical fixed-point methods is undoubtedly established, thanks to extensive studies addressing various computational problems (see, e.g., [18, 25, 26, 27, 28, 29, 30, 31]). An exhaustive overview of application fields can be found in [32].

In the end, despite a concise corresponding algorithmic pattern, random execution of asynchronous iterations requires much more than a straightforward update of classical iterations loops, depending however on the communication middleware under use. Moreover, because of their non-deterministic behavior, the accurate evaluation of a residual-based stopping criterion becomes an issue, as introduced by [33]. In classical programming patterns, to be able to detect the moment when parallel iterations have converged, a reduction operation is performed at each iteration. For efficient random execution of asynchronous iterations, blocking communication requests are avoided, which makes it hard to isolate and handle any global vector, even less in message-passing environments. One thus has to deal with various convergence detection protocols, according to both effectiveness and efficiency expectation (see, e.g., [16, 34, 35, 36, 37]). Two main libraries therefore emerged to help easily assess asynchronous iterative methods. Based on Java Remote Method Invocation (RMI), Jace [38] provides a message-passing framework which particularly suits to asynchronous iterations. The library is continuously improved, mainly in view of resource failure support (see [39]), and it now includes the convergence detection protocol from [40], which is an extension of [37] to volatile computing environments (see [41]). Still, such an approach for terminating asynchronous iterations is quite robust but not exact. As an alternative to Java performance limits, Crac [42] is a similar C++ communication library featuring same message-passing semantics. These libraries particularly target grid computing environments and manage multi-

site cluster architectures, however they are not based upon the Message Passing Interface (MPI) standard, which is undoubtedly the most popular communication framework for scientific distributed computing. Indeed, since the MPI specification has been designed on general basis for all kinds of distributed algorithms, the peculiar communication pattern of asynchronous iterations does not naturally fit into such a framework, even less in a non-intrusive manner which would totally encapsulate the management of pure communication-related objects.

In this study, we address the development of asynchronous iterative algorithms within either time or space domain decomposition frameworks. Every main aspect of such an application of asynchronous iterations is covered, from theoretical convergence analysis to efficient implementation, including accurate termination.

1.2 Outline

Chapter 2 proposes an overview of convergence conditions established for asynchronous iterations. This includes various computational models related to relaxation of linear algebraic equations and, more generally, arbitrary fixed-point problems. In linear cases, Theorem 2.2 provides a sufficient and necessary condition on the relaxed constraint. Theorem 2.3 allows us then to deduce several more practical sufficient conditions on the initial problem, provided classical relaxation schemes are used. Theorem 2.7 and Theorem 2.8 extend these sufficient conditions to two-stage relaxations (either classical or allowing *flexible communication*) arising from block decomposition of the equations system. In any cases, convergence of fixed-point methods may be addressed through Theorem 2.4, which features sufficient contraction conditions with respect to a component-wise vector norm. This result is further generalized by Theorem 2.5, using a weighted maximum norm. Finally, norm-based conditions are strictly included into a contraction framework inspired from level sets techniques, yielding Theorem 2.6. Corollary 2.2 and Corollary 2.3 deduce same results for non-stationary iterative mappings which are a generalization of the classical two-stage relaxation model. At last, Theorem 2.9 extends the maximum-norm-based contraction condition to iterative mappings defined on a couple of vectors. This generalizes the computational model of two-stage relaxations with flexible communication. The even more general Theorem 2.10 and Theorem 2.11 apply the respective component-wise and maximum norms contraction conditions to iterative mappings defined on an arbitrary number of vectors, referred to as asynchronous iterations with memory.

Chapter 3 extends asynchronous iterations models to a sub-structuring framework for the solution of algebraic linear systems, generally arising from either finite elements or finite differences discretization of PDEs. The partitioning framework is first described (illustrated by Figure 3.1), leading to a corresponding Schur complement inversion problem defined on the interface between the different substructures. Lemma 3.1, Lemma 3.2 and Lemma 3.3 first recall useful results in the theory of M-

matrices, then Proposition 3.1 and Proposition 3.2 provide two respective practical splittings of the Schur complement, under common conditions which are sufficient for the convergence of any classically derived asynchronous iterative model. Then, a parallel scheme actually based on the partitioning of the discrete domain is analyzed through the particular iterative model it implies on the assembled interface Schur problem. Lemma 3.5 first recall a preliminary key result from the Perron-Frobenius theory of nonnegative matrices, then Lemma 3.6 and its Corollary 3.1 lead to Theorem 3.1, which asserts asynchronous convergence conditions equivalent to those of classical algebraically non-overlapping additive Schwarz methods [43, 44]. From there, considering a sub-structuring parallel scheme where only the interface nodes are assembled, Lemma 3.7 and Theorem 3.2, also based on Corollary 3.1, follow same analysis principles to establish nearly same asynchronous convergence conditions for a new derived sub-structuring method.

While main studies about asynchronous iterations focused on space domain decomposition, Chapter 4 advocates a novel research direction which tackles time domains, by introducing an asynchronous parallel-in-time method derived from the Parareal approach. The goal is to verify the reliability of the inherent prediction and correction operators when they are not synchronized. In view of comparison on both convergence conditions and computational efficiency sides, Parareal iterations are first analyzed through a discrete fixed-point formulation. Proposition 4.1 exhibits an exponential upper bound of the iterated error in a similar manner as in the continuous case for ordinary differential equations (see [45]). Corollary 4.1 therefore highlights the induced convergence condition, which depends on the number of time sub-domains. While Proposition 4.3 ensures correct finite termination of Parareal-based asynchronous iterations, it does not provide information about possible performance gain over fully sequential resolution. This is rather obtained by Proposition 4.4 in the norm-based contraction framework from Theorem 2.11, featuring a convergence condition asymptotically equivalent to its counterpart in the synchronous case (Corollary 4.1). Proposition 4.5 successfully establishes a corresponding exponential upper bound of the iterated error, and the norm-based convergence result is generalized by Proposition 4.7 to allow for the coupling of different asynchronous parallel-in-time schemes. At last on computational efficiency side, Proposition 4.2 and Proposition 4.6 give time complexities of Parareal and Parareal-based asynchronous iterations respectively, both for which Corollary 4.2 and Corollary 4.3 reveal speedup limits, however an expectable better performance of asynchronous iterations.

Chapter 5 addresses the problem of terminating asynchronous iterations. We propose simple non-intrusive exact methods to compute a convergence residual during asynchronous iterations, in order to accurately assess whether convergence is reached or not. Principles of distributed snapshot [46] are followed to devise Algorithm 3 and Algorithm 4 which assemble a global iterated vector under first-in-first-out (FIFO) restriction on the communication channels. Algorithm 5 and Algorithm 6

embed computation data into the snapshot messages, which removes the FIFO condition. Algorithm 9, on another hand, squarely avoids using snapshot messages, and is based instead on comparison between two successively received data on each channel. Nonetheless, to effectively manage the non-FIFO context without such both communication and memory overhead costs, Algorithm 10 and Algorithm 11 assume an approximation of the maximum degree of out-of-order message delivering. Proposition 5.3 formally establishes their accuracy, from which Proposition 5.4 provides a practical way of setting residual-based stopping criteria for contracting mappings in weighted maximum norms.

Chapter 6 then focuses on the design of Jack2, a non-intrusive MPI-based communication library which allows quick efficient implementation of asynchronous iterations. Algorithm 13 and Algorithm 14 respectively give distributed synchronous and asynchronous iterative schemes describing the algorithmic framework taken as the leading implementation pattern. In this framework, Algorithm 15 is derived from snapshot-based convergence detection protocols to introduce non-blocking global synchronization of asynchronous iterative processes. Listing 6.1 illustrates a classical programming pattern of iterative methods within the MPI framework, from which Listing 6.2 shows a slight improvement by use of persistent requests. Listing 6.3 then presents the corresponding Jack2 framework, within which Listing 6.5 specifically describes the implementation of asynchronous iterations featuring a classical global residual-based stopping criterion evaluated by means of the non-blocking synchronization routine. Through Listing 6.6 which achieves a non-intrusive implementation of the snapshot-based convergence detection approach, Listing 6.7 successfully provides a unique Jack2-based programming pattern for both classical and asynchronous iterative methods. Finally, after the main overall architecture of the library is described by Figure 6.1, the management of point-to-point communications is discussed around Listing 6.8 and Listing 6.9. The chapter ends with the detailed description of non-blocking collective routines (Figure 6.2 to Figure 6.7), especially related to convergence detection.

At last, Chapter 7 presents experimental results related to implementation, termination and efficiency of asynchronous distributed iterative methods. The practical interest of studying asynchronous iterations is clearly introduced with Table 7.1 which shows their performance even within a poorly scalable domain partitioning configuration (Figure 7.1), on a convection-diffusion problem decomposed by means of an additive Schwarz scheme. Table 7.2 then focuses on the MPI options for implementing point-to-point communication, relatively to both message reception and message sending rates, while Table 7.3 addresses the effectiveness and efficiency of snapshot-based convergence detection protocols. Sub-structuring methods are then targeted on a gravitational potential Poisson problem illustrated through Figure 7.7 to Figure 7.9. Table 7.5 and Table 7.6 report scalability results comparing both classical and substructures-based parallel schemes of matrix Jacobi splittings, on two different problem sizes, respectively. Finally, experiments of the Parareal

time-integration method are presented on a heat evolution problem illustrated by Figure 7.11. Table 7.7 compares execution times of synchronous and asynchronous iterations while varying both the simulated physical time and the number of processor cores.

1.3 Contribution

As one can see, this work addresses several matters covering the whole procedure of designing and experimenting distributed asynchronous iterative methods. With Chapter 2, we first provide a mathematical toolbox for the application of the asynchronous iterations theory, which gives a quick access to main general asynchronous iterations models, along with various useful properties guaranteeing their convergence. We then achieved the following results by successfully targeting some particular domain decomposition schemes.

The parallel iterative model (3.11) and the related Theorem 3.2 constitute our main result in the application of asynchronous iterations theory to space domain decomposition methods. It consists of a new asynchronous iterative sub-structuring method which converges under conditions barely stronger than those of classical asynchronous algebraically non-overlapping additive Schwarz methods [43, 44]. Same conditions were derived for the iterative sub-structuring method investigated in [47], however our iterative model (3.11) clearly implies a far more efficient parallel computation. Our other related contributions include:

- the parallel iterative model (3.8) and the related Theorem 3.1, which strongly suggest the possibility of a general unified framework for both asynchronous convergence analysis of Schwarz and sub-structuring methods;
- Proposition 3.1 and Proposition 3.2, which basically provide asynchronously convergent splittings of a Schur complement, without requiring it to be explicitly generated, contrarily to classical splittings (Jacobi, Gauss-Seidel and their derivatives);
- and, to the best of our knowledge, Corollary 3.1, which proposes a new general application of the Perron-Frobenius theory of nonnegative matrices, more precisely, based on relations between spectral radii and weighted maximum norms.

Proposition 4.4 is our main result related to the Parareal time-integration method. Its significance lies in its comparison with Corollary 4.1 which we derived from our Proposition 4.1 obtained from a discrete Parareal scheme on partial differential equations, similar to the continuous approach from [45] about ordinary differential equations.The convergence of asynchronous iterations generally requires restrictions on the iterations mapping which are stronger, comparatively to their synchronous

counterparts. This aforementioned set of new theoretical results suggests, for the Parareal time-integration method, synchronous and asynchronous convergence conditions which are asymptotically equivalent, as the number of time sub-domains grows. Our other related contributions include:

- Proposition 4.3 which ensures, at least, a finite termination of Parareal asynchronous iterations, with the same solution as that given by a corresponding sequential fine time integration. This kind of result was shown in [15] for the synchronous PITA, which is a generalization of the Parareal.
- Proposition 4.5 gives an upper bound of the asynchronous iterative error which, compared to Proposition 4.1, suggests a better convergence rate of synchronous iterations, which however gets closer to the asynchronous one as the number of time sub-domains grows. Still, it also clearly shows how this actually depends on the gap of execution speed between each synchronous and asynchronous iteration.
- Corollary 4.3 sets a limit on the possible performance gain from asynchronous iterations,
- and Proposition 4.7 finally suggests general coupling of different convergent asynchronous parallel-in-time iterative schemes.

Our related publications include:

- F. Magoulès, G. Gbikpi-Benissan, Asynchronous parareal time discretization for partial differential equations, SIAM Journal of Scientific Computing *(under revision)*.

Focusing now on experimental matters, methods for terminating asynchronous iterations (i) either require a modification of the iterative algorithm such that the computation actually terminates in finite time, (ii) or involve another algorithm concurrently monitoring the convergence state. In view of being non-intrusive, for evident practical reasons, most of the latest results in this field follow the second approach. Nevertheless, to the best of our knowledge, only the snapshot-based method from [35] successfully managed to be both decentralized, formally accurate and totally non-intrusive. Indeed, even though the verification phase idea from [37] is quite robust, it is not totally effective, and piggybacking requirements make it slightly intrusive in the end, on implementation aspects. Similarly, while the macro-iterations solution from [48] provides a thorough formal result, its implementation also suggests intrusive piggybacking techniques. Algorithm 10 and Algorithm 11 constitute our main results related to the termination of asynchronous iterations, particularly improving [35] by successfully avoiding supplementary exchange of application data, which could constitute non-negligible communication overhead costs. Our Proposition 5.3 and Proposition 5.4 formally establish the reliability of these new snapshot-based algorithms. Our other related contributions include:

- Algorithm 3 and Algorithm 4 which show direct possible extensions of the reference snapshot protocol [46] to the asynchronous iterations termination problem;
- Algorithm 5 and Algorithm 6 which take off a prior tree-based centralization phase featured by [35] (also featured by [37], but in a non-deterministic way), and thus are the first accurate, totally non-intrusive and totally decentralized convergence detection protocols;
- and, at last, Algorithm 9 which derives a message-free monitoring algorithm based on only application messages.

Our related publications include:

- F. Magoulès, G. Gbikpi-Benissan, Distributed convergence detection based on global residual error under asynchronous iterations, IEEE Transactions on Parallel and Distributed Systems, 2017 *(see [49])*.

Last matters cover the efficient implementation of asynchronous iterations, requiring however the minimum possible amount of changes in classical iterations programming patterns. The MPI specification does not naturally extend to asynchronous iterations, which makes it hard to simply render the asynchronous semantics by means of non-blocking MPI communication procedures. Therefore, well-known libraries providing general communication semantics which easily include asynchronous iterations does not rely on MPI libraries, even though the MPI currently constitutes the most prominent communication framework for implementing distributed scientific applications. Our contribution here is the design and implementation of an MPI-based communication library which proposes a unique message-passing programming interface for implementing both synchronous and asynchronous iterative methods, such that the asynchronous semantics can now be rendered at run-time, at an only internal communication level. Our related results include:

- Algorithm 15 which introduces a non-blocking collective synchronization routine for convergence detection, derived from our analysis of snapshot-based termination algorithms. Listing 6.5 and Listing 6.6 propose a corresponding new iterations programming pattern where local residual is no more computed at each local iteration, which clearly leads to far faster iterative solvers, as shown in Figure 7.4. This also allows a classical monitoring of global residual during asynchronous iterations, as shown in Figure 7.5.
- Listing 6.8 and Listing 6.9, at last, propose a point-to-point communication semantic which provides a finer, more efficient, handling of communicated data, compared to the currently advocated non-MPI *put/get* semantic.

Our related publications include:

- F. Magoulès, G. Gbikpi-Benissan, JACK2: an MPI-based communication library with non-blocking synchronization for asynchronous iterations, Advances in Engineering Software, 2018 *(see [50])*;
- G. Gbikpi-Benissan, F. Magoulès, JACK2: A new high-level communication library for parallel iterative methods, Proceedings PARENG, 2017 *(see [51])*;
- F. Magoulès, G. Gbikpi-Benissan, JACK: an asynchronous communication kernel library for iterative algorithms, The Journal of Supercomputing, 2017 *(see [52])*.

Chapter 2

Asynchronous iterations

2.1 Introduction

This chapter is a quick survey of several very useful results from the theory of asynchronous iterations. They all relate to particular contraction properties over the iterative computational model derived from either a linear or an arbitrary fixed-point problem. We exhibit three kinds of iterations mappings, characterizing a general fixed-point problem of the form

$$f^k(x, x, \ldots, x) = x, \quad \forall k \in \mathbb{N}, \qquad f^k : E^m \to E, \quad m \in \mathbb{N},\ m > 0,$$

where k is an iteration number, f^k denotes a mapping depending on k, and E^m, the Cartesian product of m sets E. For each of these three successive cases, given by

1. $m = 1, \quad f^0 \equiv f^1 \equiv \cdots,$
2. $m = 1,$
3. $f^0 \equiv f^1 \equiv \cdots,$

state-of-the-art convergence conditions are presented, first in an initial linear framework, then for its corresponding general arbitrary mapping.

2.2 Fixed-point iterations

2.2.1 Linear model

Let n be a strictly positive integer, let A be a nonsingular $n \times n$ real matrix, and let b be a vector with n real components. We consider the problem of determining a vector x such that

$$Ax = b. \tag{2.1}$$

Let then M and N be two matrices such that

$$A = M - N,$$

with M being nonsingular. Instead of directly finding the vector x^* which satisfies (2.1), a sequence $\{x^k\}_{k\in\mathbb{N}}$ of vectors is iteratively generated such that for all k, x^{k+1} satisfies the relaxed constraint

$$Mx^{k+1} = Nx^k + b, \tag{2.2}$$

with x^0 being given. As examples, the Jacobi relaxation consists of taking

$$\left\{ \begin{array}{rcll} m_{i,i} & = & a_{i,i}, & \\ m_{i,j} & = & 0, & i \neq j \end{array} \right. , \quad i,j \in \{1,\ldots,n\},$$

while the Gauss-Seidel relaxation is given by:

$$\left\{ \begin{array}{rcll} m_{i,j} & = & a_{i,j}, & j \leq i, \\ m_{i,j} & = & 0, & j > i \end{array} \right. , \quad i,j \in \{1,\ldots,n\},$$

where $m_{i,j}$ and $a_{i,j}$ are entries of M and A respectively. Now let T and c be the matrix and the vector defined by:

$$T = M^{-1}N, \qquad c = M^{-1}b,$$

and let f be the mapping given by:

$$\begin{array}{rccc} f: & \mathbb{R}^n & \to & \mathbb{R}^n \\ & x & \mapsto & Tx + c \end{array} . \tag{2.3}$$

We then have

$$x^{k+1} = f(x^k), \quad \forall k \in \mathbb{N}, \tag{2.4}$$

and it clearly appears that

$$Ax = b \quad \Longleftrightarrow \quad x = f(x),$$

which reformulates (2.1) as the problem of finding a fixed point of f.

Definition 2.1 (Convergent iterations)**.** *An iterative computational model is convergent when, for any matching sequence* $\{x^k\}_{k\in\mathbb{N}}$*,*

$$\lim_{k\to\infty} x^k = x^*.$$

Theorem 2.1 (also see, e.g., Theorem 2.1 in [53])**.** *The computational model* (2.4) *is convergent for any given* $x^0 \neq x^*$*, if, and only if, the spectral radius* $\rho(T)$ *of* T *satisfies:*

$$\rho(T) < 1.$$

Proof. We have:

$$\begin{aligned} x^k - x^* &= f(x^{k-1}) - f(x^*), \\ &= T^k(x^0 - x^*), \end{aligned}$$

and then,

$$\begin{aligned} \lim_{k\to\infty} x^k = x^* \quad &\iff \quad \lim_{k\to\infty} T^k(x^0 - x^*) = \begin{bmatrix} 0 & \cdots & 0 \end{bmatrix}^\mathsf{T}, \\ &\iff \quad \lim_{k\to\infty} T^k = O, \end{aligned}$$

where O is the null matrix. Let λ_i, with $1 \leq i \leq n$, be any eigenvalue of T and let u be the associated eigenvector. We have:

$$\begin{aligned} \lim_{k\to\infty} T^k = 0 \quad &\iff \quad \lim_{k\to\infty} T^k u = \begin{bmatrix} 0 & \cdots & 0 \end{bmatrix}^\mathsf{T}, \\ &\iff \quad \lim_{k\to\infty} \lambda_i^k u = \begin{bmatrix} 0 & \cdots & 0 \end{bmatrix}^\mathsf{T}, \\ &\iff \quad \lim_{k\to\infty} \lambda_i^k = 0, \\ &\iff \quad -1 < \lambda_i < 1, \end{aligned}$$

and then,

$$\begin{aligned} \lim_{k\to\infty} x^k = x^* \quad &\iff \quad |\lambda_i| < 1, \quad \forall i \in \{1, \ldots, n\}, \\ &\iff \quad \rho(T) < 1. \end{aligned}$$

□

At last, let $p, n_1, n_2, \ldots, n_p$ be strictly positive integers satisfying:

$$\sum_{i=1}^{p} n_i = n, \tag{2.5}$$

and let f_i, with $1 \leq i \leq p$, be mappings defined such that

$$f(x) = \begin{bmatrix} f_1(x_1, \ldots, x_p) & \cdots & f_p(x_1, \ldots, x_p) \end{bmatrix}^\mathsf{T}. \tag{2.6}$$

For instance, with $n_i \times n$ matrices T_i such that

$$T = \begin{bmatrix} T_1 & \cdots & T_p \end{bmatrix}^\mathsf{T},$$

one would have

$$f_i(x) = T_i x + c.$$

Now let $\{P^k\}_{k\in\mathbb{N}}$ be a sequence of integer subsets, with

$$P^k \subseteq \{1,\ldots,p\},$$

and satisfying:

$$\forall i \in \{1,\ldots,p\}, \quad \text{card}\{k \in \mathbb{N} \mid i \in P^k\} = \infty. \tag{2.7}$$

Let also τ_j^i, with $1 \leq i \leq p$ and $1 \leq j \leq p$, be nonnegative integer-valued functions, with

$$\tau_j^i(k) \leq k,$$

and satisfying:

$$\forall i,j \in \{1,\ldots,p\}, \quad \lim_{k\to\infty} \tau_j^i(k) = \infty. \tag{2.8}$$

We consider here asynchronous iterations generating a sequence $\{x^k\}_{k\in\mathbb{N}}$ such that

$$x_i^{k+1} = \begin{cases} f_i\left(x_1^{\tau_1^i(k)},\ldots,x_p^{\tau_p^i(k)}\right), & i \in P^k, \\ x_i^k, & i \notin P^k. \end{cases} \tag{2.9}$$

Classical iterations modeled by (2.4), e.g., Jacobi relaxations, are therefore particular cases of asynchronous iterations where $P^k = \{1,\ldots,p\}$ and $\tau_j^i(k) = k$, which yields:

$$x_i^{k+1} = f_i\left(x_1^k,\ldots,x_p^k\right), \quad i \in \{1,\ldots,p\}. \tag{2.10}$$

From a computational point of view, (2.7) and (2.8) are trivially satisfied when none of the parallel processes definitively stops neither iterating nor receiving updated data for all its dependencies.

Theorem 2.2 (Chazan and Miranker, 1969)**.** *The computational model* (2.9) *is convergent if, and only if,*

$$\rho(|T|) < 1,$$

where $|T|$ *denotes the matrix whose each entry is the absolute value of the corresponding entry of* T*.*

Proof. See [21]. □

Remark 2.1 (see, e.g., [21])**.** *If the convergence condition in Theorem 2.1 is fulfilled for the Jacobi splitting of* $A^+ = (a_{i,j}^+)$*, with*

$$\begin{cases} a_{i,i}^+ & := & a_{i,i}, \\ a_{i,j}^+ & := & -|a_{i,j}|, \quad i \neq j, \end{cases}$$

then the convergence condition in Theorem 2.2 is fulfilled for the Jacobi splitting of A*.*

Definition 2.2 (Diagonally dominant matrix)**.** *An $n \times n$ matrix $B = (b_{i,j})$ is diagonally dominant when*

$$\forall i \in \{1, \ldots, n\}, \quad \sum_{\substack{j=1 \\ j \neq i}}^{n} |b_{i,j}| \leq |b_{i,i}|,$$

where $b_{i,j}$ are entries of B. B is strictly diagonally dominant when this inequality is strict (for all $i \in \{1, \ldots, n\}$).

Definition 2.3 (Irreducible matrix)**.** *A square matrix B is irreducible when there exists no permutation matrix $\mathcal{P}$ such that*

$$\mathcal{P} B \mathcal{P}^{\mathsf{T}} = \begin{bmatrix} B_{1,1} & B_{1,2} \\ O & B_{2,2} \end{bmatrix},$$

where $B_{1,1}$ and $B_{2,2}$ are two square sub-matrices.

Definition 2.4 (Irreducibly diagonally dominant matrix)**.** *A square matrix B is irreducibly diagonally dominant when B is irreducible and diagonally dominant with strict inequality holding for at least one row.*

Definition 2.5 (Stieltjes matrix)**.** *A square matrix B is a Stieltjes matrix when all off-diagonal entries of B are non-positive, and B is symmetric and positive definite.*

Remark 2.2 (see, e.g., [21])**.** *If the matrix A is either*

- *symmetric and strictly diagonally dominant, or*
- *irreducibly diagonally dominant, or*
- *a Stieltjes matrix,*

then the convergence condition in Theorem 2.2 is fulfilled for the Jacobi splitting of A.

Consider now that the matrices A and M are of the form

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,p} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p,1} & A_{p,2} & \cdots & A_{p,p} \end{bmatrix}, \qquad M = \begin{bmatrix} A_{1,1} & O & \cdots & O \\ O & A_{2,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & O \\ O & \cdots & O & A_{p,p} \end{bmatrix},$$

where $A_{i,i}$, with $1 \leq i \leq p$, are nonsingular $n_i \times n_i$ sub-matrices. Such a decomposition yields the block-Jacobi relaxation given by:

$$A_{i,i} x_i^{k+1} = (N x^k + b)_i, \quad \forall i \in \{1, \ldots, p\}. \tag{2.11}$$

Definition 2.6 (Comparison matrix)**.** *The comparison matrix* $\langle B \rangle = (\langle b \rangle_{i,j})$ *of a square matrix* $B = (b_{i,j})$ *is defined by:*

$$\begin{cases} \langle b \rangle_{i,i} & := & |b_{i,i}|, \\ \langle b \rangle_{i,j} & := & -|b_{i,j}|, \quad i \neq j. \end{cases}$$

Definition 2.7 (M-matrix)**.** *A square matrix* B *is an* M*-matrix when there exists a real* $\gamma_0 > 0$ *and a matrix* $C \geq O$ *such that*

$$B = \gamma_0 I - C, \qquad \gamma_0 > \rho(C).$$

Remark 2.3 (see, e.g., [54])**.** *A square matrix* $B = (b_{i,j})$ *is an* M*-matrix when* B *is nonsingular and*

$$\begin{cases} b_{i,j} & \leq & 0, \quad i \neq j, \\ B^{-1} & \geq & O. \end{cases}$$

Remark 2.4 (see, e.g., Corollary 3.24 in [55])**.** *If a matrix* B *is a Stieltjes matrix, then* B *is an* M*-matrix.*

Definition 2.8 (H-matrix)**.** *A square matrix* B *is an* H*-matrix when its comparison matrix* $\langle B \rangle$ *is an* M*-matrix.*

Remark 2.5 (see, e.g., [56])**.** *If a matrix* B *is either*

- *strictly diagonally dominant, or*
- *irreducibly diagonally dominant, or*
- *an* M*-matrix,*

then B *is an* H*-matrix.*

Notation 2.1 (Matrix splitting)**.** *A splitting* $(\mathcal{M}, \mathcal{N})$ *of a matrix* B *is a couple of matrices satisfying:*

$$B = \mathcal{M} - \mathcal{N}.$$

Definition 2.9 (H-splitting)**.** *A splitting* $(\mathcal{M}, \mathcal{N})$ *of a matrix* B *is an* H*-splitting when the matrix* $\langle \mathcal{M} \rangle - |\mathcal{N}|$ *is an* M*-matrix.*

Remark 2.6 (see, e.g., Proof of Lemma 4.2 in [57])**.** *If a matrix* B *is an* H*-matrix, then a block-Jacobi splitting of* B *is an* H*-splitting.*

Theorem 2.3 (Frommer and Szyld, 1992)**.** *If the splitting* (M, N) *of* A *is an* H*-splitting, then the convergence condition in Theorem 2.2 is fulfilled.*

Proof. See [56]. □

2.2.2 General model

Now let

$$f : E \to E, \qquad f_i : E \to E_i, \quad 1 \leq i \leq p, \tag{2.12}$$

be arbitrary mappings, but still satisfying (2.6), with

$$E = E_1 \times \cdots \times E_p.$$

We consider here the general fixed-point problem consisting of determining a vector x such that

$$f(x) = x. \tag{2.13}$$

Let then

$$\|.\|_{(i)} : E_i \to \mathbb{R}, \quad 1 \leq i \leq p,$$

be p given norms, and ψ, the norm given by:

$$\begin{array}{rccl} \psi : & E & \to & \mathbb{R}^p \\ & x & \mapsto & \big(\|x_1\|_{(1)}, \ldots, \|x_p\|_{(p)}\big) \end{array}.$$

Theorem 2.4 (Miellou, 1975)**.** *If there exists a matrix $\mathcal{T} \geq O$ (nonnegative matrix), with $\rho(\mathcal{T}) < 1$, such that*

$$\forall x, y \in E, \quad \psi(f(x) - f(y)) \leq \mathcal{T}\psi(x - y),$$

then the problem (2.13) *has a unique solution x^*, and the computational model* (2.9) *is convergent.*

Proof. See [17]. □

Let $\|.\|_\infty^w$, with $w \in (\mathbb{R}^{+,*})^p$, denote weighted maximum norms given by:

$$\|x\|_\infty^w = \max_{1 \leq i \leq p} \frac{\|x_i\|_{(i)}}{w_i}, \quad x \in E. \tag{2.14}$$

Theorem 2.5 (El Tarazi, 1982)**.** *If the problem* (2.13) *has a solution x^*, if there exists a vector $w > 0$ (positive vector) and a positive real $\alpha < 1$ such that*

$$\forall x \in E, \quad \|f(x) - x^*\|_\infty^w \leq \alpha \|x - x^*\|_\infty^w,$$

then the computational model (2.9) *is convergent.*

Proof. See [19]. □

Corollary 2.1 (El Tarazi, 1982)**.** *If there exists a vector $w > 0$ and a positive real $\alpha < 1$ such that*

$$\forall x, y \in E, \quad \|f(x) - f(y)\|_\infty^w \leq \alpha \|x - y\|_\infty^w,$$

then the problem (2.13) *has a unique solution x^*, and the computational model* (2.9) *is convergent.*

Proof. See [19]. □

At last, let $\{E^t\}_{t\in\mathbb{N}}$ be a sequence of nonempty subsets of E, with

$$E^{t+1} \subset E^t, \quad \forall t \in \mathbb{N},$$

satisfying, for any sequence $\{x^t\}_{t\in\mathbb{N}}$ such that $x^t \in E^t$:

$$x \in \{x^t\}_{t\to\infty} \iff x = f(x),$$

where $\{x^t\}_{t\to\infty}$ denotes the set of limit points of $\{x^t\}_{t\in\mathbb{N}}$. Let also $\{E_i^t\}_{t\in\mathbb{N}}$, with $1 \le i \le p$, be sequences of subsets of E_i such that

$$\forall t \in \mathbb{N}, \quad E^t = E_1^t \times \cdots \times E_p^t.$$

Theorem 2.6 (Bertsekas, 1983)**.** *If the mapping f satisfies:*

$$\forall t \in \mathbb{N}, \quad x \in E^t \implies f(x) \in E^{t+1},$$

then for any sequence $\{x^k\}_{k\in\mathbb{N}}$ matching the computational model (2.9)*,*

$$x \in \{x^k\}_{k\to\infty} \iff x = f(x),$$

for any given $x^0 \in E^0$.

Proof. See [20] or, e.g., [16]. □

2.3 Two-stage fixed-point iterations

2.3.1 Linear model

Consider again the problem (2.1) and the relaxed constraint (2.2). Let, as an example, M satisfy the block-Jacobi relaxation (2.11). Let then $M^{(i)}$ and $N^{(i)}$, with $1 \le i \le p$, be matrices such that

$$A_{i,i} = M^{(i)} - N^{(i)},$$

with $M^{(i)}$ being nonsingular. Once more, instead of directly finding a vector $(x^{k+1})^*$ which satisfies (2.2), sequences

$$\left\{ \left(x_i^{k+1}\right)^{k^{(i)}} \right\}_{k^{(i)}\in\mathbb{N}}, \qquad 1 \le i \le p,$$

are iteratively generated such that for all $k^{(i)}$, a secondary relaxed constraint

$$M^{(i)} \left(x_i^{k+1}\right)^{k^{(i)}+1} = N^{(i)} \left(x_i^{k+1}\right)^{k^{(i)}} + \left(Nx^k + b\right)_i, \tag{2.15}$$

is satisfied, with

$$\left(x_i^{k+1}\right)^0 = x_i^k.$$

Let f_i^t, with $t \in \mathbb{N}$ and $1 \leq i \leq p$, be mappings recursively given by:

$$f_i^t(x) = {M^{(i)}}^{-1} N^{(i)} f_i^{t-1}(x) + {M^{(i)}}^{-1}(Nx+b)_i, \qquad x \in \mathbb{R}^n,$$

with $f_i^0(x) = x_i$, so that we thus have

$$\left(x_i^{k+1}\right)^{k^{(i)}+1} = f_i^{k^{(i)}+1}\left(x^k\right).$$

At last, let $\{m_i^k\}_{k\in\mathbb{N}}$, with $1 \leq i \leq p$, be sequences of positive integers. We are now interested in extending the model (2.9) to:

$$x_i^{k+1} = \begin{cases} f_i^{m_i^k}\left(x_1^{\tau_1^i(k)}, \ldots, x_p^{\tau_p^i(k)}\right), & i \in P^k, \\ x_i^k, & i \notin P^k. \end{cases} \tag{2.16}$$

Definition 2.10 (H-compatible splitting)**.** *A splitting $(\mathcal{M}, \mathcal{N})$ of a matrix B is an* H*-compatible splitting when*

$$\langle B \rangle = \langle \mathcal{M} \rangle - |\mathcal{N}|.$$

Remark 2.7 (see, e.g., Section 2.6 of [58])**.** *Jacobi and Gauss-Seidel relaxations are* H*-compatible splittings.*

Theorem 2.7 (Frommer and Szyld, 1994)**.** *If (M, N) is an* H*-splitting, and $(M^{(i)}, N^{(i)})$ are* H*-compatible splittings, then the computational model* (2.16) *is convergent.*

Proof. See [22]. □

2.3.2 General non-stationary model

Now suppose, just as before, the general fixed-point problem (2.13), and let

$$f^k : E \to E, \qquad f_i^k : E \to E_i, \quad 1 \leq i \leq p, \qquad k \in \mathbb{N},$$

be arbitrary mappings, with

$$f^k(x) = \begin{bmatrix} f_1^k(x) & \cdots & f_p^k(x) \end{bmatrix}^{\mathsf{T}}.$$

We consider here a secondary fixed-point problem reformulating (2.13) by:

$$f(x) = x \quad \Longleftrightarrow \quad f^k(x) = x, \quad \forall k \in \mathbb{N}.$$

A more general formulation of (2.16) is then given by:

$$x_i^{k+1} = \begin{cases} f_i^k\left(x_1^{\tau_1^i(k)}, \ldots, x_p^{\tau_p^i(k)}\right), & i \in P^k, \\ x_i^k, & i \notin P^k. \end{cases} \tag{2.17}$$

Corollary 2.2 (of Theorem 2.5; Frommer and Szyld, 1994)**.** *If the problem* (2.13) *has a solution x^*, if there exists a vector $w > 0$ and a positive real $\alpha < 1$ such that*

$$\forall x \in E, \quad \forall k \in \mathbb{N}, \qquad \|f^k(x) - x^*\|_\infty^w \leq \alpha \|x - x^*\|_\infty^w,$$

then the computational model (2.17) *is convergent.*

Proof. See [22]. □

Corollary 2.3 (of Theorem 2.6; Frommer and Szyld, 2000)**.** *If*

$$\forall t \in \mathbb{N}, \quad \forall k \in \mathbb{N}, \qquad x \in E^t \implies f^k(x) \in E^{t+1},$$

then for any sequence $\{x^k\}_{k\in\mathbb{N}}$ matching the computational model (2.17)*,*

$$x \in \{x^k\}_{k\to\infty} \iff x = f(x),$$

for any given $x^0 \in E^0$.

Proof. See [57]. □

2.4 Two stages with flexible communication

2.4.1 Linear model

Consider once more the linear problem (2.1), the two-stage relaxation (2.15) and the derived computational model (2.16). Here, instead of delivering x_i^{k+1}, with $i \in P^k$, only after arbitrary m_i^k inner iterations, one would like to deliver another sequence $\{x^{k'}\}_{k'\in\mathbb{N}}$ such that

$$x_i^{k'} = \left(x_i^{k+1}\right)^{k^{(i)}}, \qquad k^{(i)} \in \{0, \ldots, m_i^k\}, \quad i \in P^{k'},$$

where $\{P^{k'}\}_{k'\in\mathbb{N}}$ is another sequence of subsets of $\{1, \ldots, p\}$ also satisfying (2.7). This yields the more flexible delivering scheme

$$M^{(i)} x_i^{k'+1} = N^{(i)} x_i^{k'} + \left(Nx^k + b\right)_i, \quad i \in P^{k'},$$

with

$$k' \geq k + k^{(i)}.$$

Let then $\widetilde{f}_i$, with $1 \leq i \leq p$, be mappings given by:

$$\widetilde{f}_i(x, y) = {M^{(i)}}^{-1} N^{(i)} x_i + {M^{(i)}}^{-1} (Ny + b)_i, \qquad x, y \in \mathbb{R}^n.$$

The model (2.9) is thus extended to:

$$x_i^{k'+1} = \begin{cases} \widetilde{f}_i\left(x^{k'}, x_1^{\tau_1^i(k')}, \ldots, x_p^{\tau_p^i(k')}\right), & i \in P^{k'}, \\ x_i^{k'}, & i \notin P^{k'}. \end{cases} \tag{2.18}$$

Theorem 2.8 (Frommer and Szyld, 1998)**.** *If (M, N) is an* H*-splitting, and $(M^{(i)}, N^{(i)})$ are* H*-compatible splittings, then the computational model* (2.18) *is convergent.*

Proof. See [59]. □

2.4.2 General multi-variable model

Finally, let

$$\widetilde{f} : E \times E \to E, \qquad \widetilde{f}_i : E \times E \to E_i, \quad 1 \leq i \leq p,$$

be arbitrary mappings, with

$$\widetilde{f}(x, y) = \left[\widetilde{f}_1(x, y) \quad \cdots \quad \widetilde{f}_p(x, y) \right]^{\mathsf{T}},$$

and consider the secondary fixed-point problem reformulating (2.13) by:

$$f(x) = x \quad \Longleftrightarrow \quad \widetilde{f}(x, x) = x.$$

A more general formulation of (2.18) is then also given by:

$$x_i^{k+1} = \begin{cases} \widetilde{f}_i \left(x_1^{\rho_1^i(k)}, \ldots, x_p^{\rho_p^i(k)}, x_1^{\tau_1^i(k)}, \ldots, x_p^{\tau_p^i(k)} \right), & i \in P^k, \\ x_i^k, & i \notin P^k, \end{cases} \tag{2.19}$$

where ρ_j^i, with $1 \leq i \leq p$ and $1 \leq j \leq p$, are nonnegative integer-valued functions, with

$$\rho_j^i(k) \leq k,$$

and satisfying:

$$\lim_{k \to \infty} \rho_j^i(k) = \infty.$$

Theorem 2.9 (Frommer and Szyld, 1998)**.** *If the problem* (2.13) *has a solution* x^**, if there exists a vector* $w > 0$ *and a positive real* $\alpha < 1$ *such that*

$$\forall x, y \in E, \quad \|\widetilde{f}(x, y) - x^*\|_\infty^w \leq \alpha \max \left\{ \|x - x^*\|_\infty^w, \|y - x^*\|_\infty^w \right\},$$

then the computational model (2.19) *is convergent.*

Proof. See [59]. □

Even more generally, let us consider mappings

$$\widetilde{F} : E^m \to E, \qquad \widetilde{F}_i : E^m \to E_i, \quad 1 \leq i \leq p,$$

satisfying:

$$\widetilde{F}(x^1, \ldots, x^m) = \left[\widetilde{F}_1(x^1, \ldots, x^m) \quad \cdots \quad \widetilde{F}_p(x^1, \ldots, x^m) \right]^{\mathsf{T}},$$

with $m \in \mathbb{N}$, E^m being the Cartesian product of m sets E, and such that

$$f(x) = x \quad \Longleftrightarrow \quad \widetilde{F}(x, x, \ldots, x) = x.$$

It yields an iterative model of the form:

$$x_i^{k+1} = \begin{cases} \widetilde{F}_i\left(\left(x_1^{\tau_{1,1}^i(k)}, \ldots, x_p^{\tau_{p,1}^i(k)}\right), \ldots, \left(x_1^{\tau_{1,m}^i(k)}, \ldots, x_p^{\tau_{p,m}^i(k)}\right)\right), & i \in P^k, \\ x_i^k, & i \notin P^k, \end{cases} \tag{2.20}$$

introduced by [18] as *asynchronous iterations with memory.*

Theorem 2.10 (Baudet, 1978)**.** *If there exists a matrix $\mathcal{T} \geq O$ (nonnegative), with $\rho(\mathcal{T}) < 1$, such that*

$$\forall X, Y \in E^m, \quad \psi(\widetilde{F}(X) - \widetilde{F}(Y)) \leq \mathcal{T} \max\{\psi(x^1 - y^1), \ldots, \psi(x^m - y^m)\},$$

with

$$X := (x^1, \ldots, x^m), \quad Y := (y^1, \ldots, y^m),$$

then the problem (2.13) *has a unique solution x^*, and the computational model* (2.20) *is convergent.*

Proof. See [18]. □

Theorem 2.11 (El Tarazi, 1982)**.** *If the problem* (2.13) *has a solution x^*, if there exists a vector $w > 0$ and a positive real $\alpha < 1$ such that*

$$\forall x^1, \ldots, x^m \in E, \quad \|\widetilde{F}(x^1, \ldots, x^m) - x^*\|_\infty^w \leq \alpha \max\left\{\|x^1 - x^*\|_\infty^w, \ldots, \|x^m - x^*\|_\infty^w\right\},$$

then the computational model (2.20) *is convergent.*

Proof. See [19]. □

2.5 Conclusion

Linear fixed-point methods are classically derived from a splitting

$$A = M - (M - A)$$

of a matrix A, where M is a nonsingular matrix, in order to solve an equation

$$Ax = b,$$

x being a column vector, by means of iterations

$$Mx^{k+1} = (M - A)x^k + b.$$

A sufficient and necessary condition for the convergence of such methods is to have a contracting behavior implied by the spectral radius bound

$$\rho(I - M^{-1}A) < 1.$$

In [21] then, the slightly more restrictive bound

$$\rho(|I - M^{-1}A|) < 1$$

is established for asynchronous convergence, indicating as well that asynchronous convergence implies classical synchronous convergence. We then saw that many sufficient conditions were derived according to some properties of both A and M, related, for instance, to diagonal dominance, irreducibility, Stieltjes matrices, M-matrices, H-splittings, and other close matrix characterizations.

While generalizing the spectral radius bound to arbitrary fixed-point problems, key sufficient contraction conditions were established with component-wise norms (see [17]), weighted maximum norms (see [19]) and nested sets (see [20]). The most general formulation of the norm-based results lies in the framework of asynchronous iterations with memory, which features mappings with an arbitrary number of vector-inputs (see [18, 19]). This framework will constitute our main theoretical tool for designing new asynchronous domain decomposition methods, both in space and time. The next chapter addresses the case of space domains.

Chapter 3

Asynchronous space domain decomposition

3.1 Introduction

Asynchronous iterations were largely applied to overlapping Schwarz domain decomposition methods (see, e.g., [60, 61, 62, 29]), according to their tight relation with block-Jacobi and Gauss-Seidel fixed-point methods (see, e.g., [44]). Recently, optimized Schwarz methods have been investigated (see [63]), which can be non-overlapping, just as sub-structuring methods, for which some primal approach has been asynchronously applied in [47].

Primal domain sub-structuring methods yield a Schur complement inversion problem defined on the junction interface shared by the different sub-domains. Ideally however, this Schur complement is disassembled over the set of sub-domains, for instance in an additive way which corresponds to an interface problem of the form

$$\sum_{i=1}^{p} S^{(i)} z = \sum_{i=1}^{p} d^{(i)},$$

where p is the number of sub-domains. Direct application of asynchronous iterations based on classical matrix splittings would lead to a completely assembled Schur complement, and hence, one would loose the advantages from substructures-based parallel computation, where even the sub-complements $S^{(i)}$ do not need to be explicitly generated. We therefore address here the design and analysis of more suitable matrix splittings which effectively allow us to ensure, at convergence, interface continuity given by

$$z^* = z^{(1)*} = \cdots = z^{(p)*},$$

without assembling the whole interface problem.

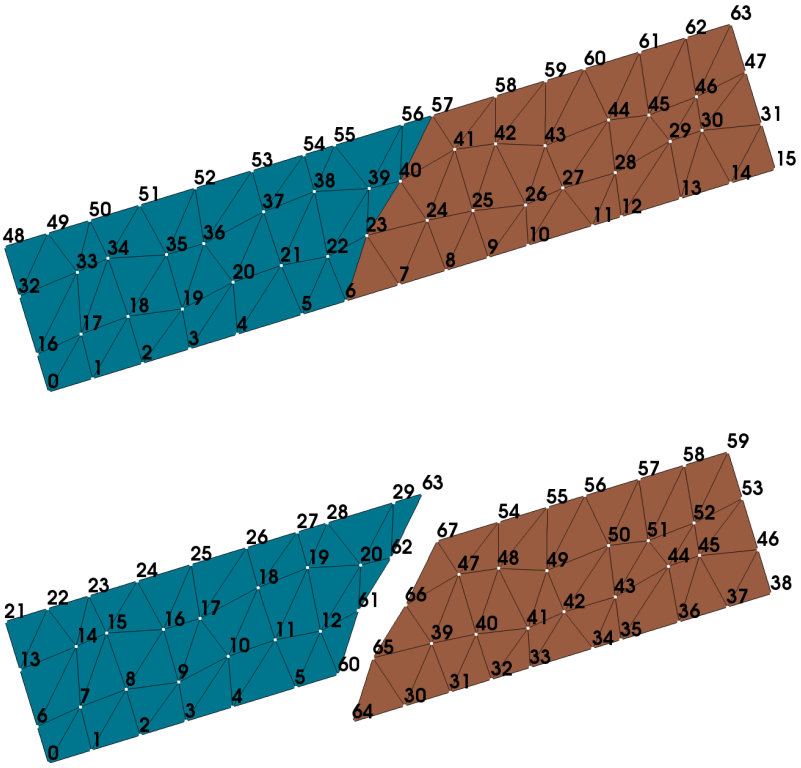


Figure 3.1: Partitioning and reordering of a finite element mesh.

3.2 Partitioning-based Schur complement

3.2.1 Problem formulation

Let us consider a partial differential equation (PDE) which solution function u^* is defined on an arbitrary geometric shape Ω. Let x^* be a discrete approximation of u^*, based on a finite-element meshing of Ω, such that x^* satisfies some linear algebraic equation

$$Ax = b. \tag{3.1}$$

Consider now a partitioning of Ω, with a reordering of the nodes of the finite elements mesh, as illustrated for instance by Figure 3.1. By also considering, however, a non-disjoint set of interface nodes (nodes 6, 23, 40 and 57 of Figure 3.1, top), the matrix

A arising from the discrete formulation of the PDE has the form

$$A = \begin{bmatrix} A_{1,1} & O & \cdots & O & A_{1,p+1} \\ O & A_{2,2} & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & O & A_{p-1,p+1} \\ O & \cdots & O & A_{p,p} & A_{p,p+1} \\ A_{p+1,1} & \cdots & A_{p+1,p-1} & A_{p+1,p} & A_{p+1,p+1} \end{bmatrix},$$

where lines i, with $1 \leq i \leq p$, relate to the n_i internal nodes of the substructure i, and lines $p+1$ relate to the n_{p+1} interface nodes (see [64]). The problem (3.1) can therefore be expanded as:

$$\begin{cases} A_{i,i}x_i + A_{i,p+1}x_{p+1} & = b_i, \quad \forall i \in \{1, \ldots, p\}, \\ \sum_{i=1}^{p} A_{p+1,i}x_i + A_{p+1,p+1}x_{p+1} & = b_{p+1}, \end{cases}$$

so that, by considering

$$x_i = -A_{i,i}^{-1}\left(A_{i,p+1}x_{p+1} - b_i\right)$$

in the second equation, one reduces (3.1) to:

$$-\sum_{i=1}^{p} A_{p+1,i}A_{i,i}^{-1}\left(A_{i,p+1}x_{p+1} - b_i\right) + A_{p+1,p+1}x_{p+1} = b_{p+1},$$

$$\left(A_{p+1,p+1} - \sum_{i=1}^{p} A_{p+1,i}A_{i,i}^{-1}A_{i,p+1}\right)x_{p+1} = -\sum_{i=1}^{p} A_{p+1,i}A_{i,i}^{-1}b_i + b_{p+1},$$

which corresponds to a problem of the form

$$Sx_{p+1} = d, \tag{3.2}$$

with S being the Schur complement defined on the interface between the discrete substructures of Ω. While from there, classical matrix splittings (Jacobi, Gauss-Seidel, SOR, ...) could be applied to $S = (s_{i,j})$, with $i, j \in \{1, \ldots, n_{p+1}\}$, our goal here is however to yield an iterative mapping which does not require to explicitly generate coefficients $s_{i,j}$, mainly in view of possible substructures-based parallel computation.

3.2.2 Practical matrix splittings

Let then, first, (M, N) be a splitting of S given by:

$$M := \gamma I. \tag{3.3}$$

Lemma 3.1 (also see, e.g., Theorem 3.18 in [55])**.** *If a square matrix $B = (b_{i,j})$ is an* M*-matrix, then*

$$b_{i,i} > 0, \quad \forall i.$$

Proof. By Definition 2.7 of M-matrices, we have

$$B = \gamma_0 I - C,$$

with

$$C = (c_{i,j}) \geq O, \quad \gamma_0 > \rho(C).$$

We know that (see, e.g., [65])

$$C \geq O \quad \Longrightarrow \quad \rho(C) \geq c_{i,i}, \ \forall i.$$

The lemma thus directly follows as:

$$\begin{aligned} \gamma_0 > \rho(C) \geq c_{i,i}, \quad \forall i, \\ b_{i,i} = \gamma_0 - c_{i,i} > 0, \quad \forall i. \end{aligned}$$

□

Definition 3.1 (M-splitting)**.** *A splitting $(\mathcal{M}, \mathcal{N})$ of a matrix B is an* M*-splitting when $\mathcal{M}$ is an* M*-matrix and $\mathcal{N} \geq O$.*

Lemma 3.2 (Frommer and Szyld, 1992)**.** *An* M*-splitting of an* M*-matrix is both an* H*-splitting and an* H*-compatible splitting.*

Proof. See [56]. □

Proposition 3.1. *If S is an* M*-matrix, and*

$$\gamma \geq \max_{1 \leq i \leq n_{p+1}} s_{i,i}, \tag{3.4}$$

then the splitting (3.3) *of S induces a convergent asynchronous iterative model* (2.9)*.*

Proof. S being an M-matrix, we have, from Lemma 3.1, that

$$s_{i,i} > 0, \quad \forall i \in \{1, \ldots, n_{p+1}\},$$

which implies, with (3.4), that

$$\gamma > 0,$$

and then both $M = \gamma I$ is an M-matrix and

$$N = \gamma I - S \ \geq \ O.$$

It follows by Definition 3.1 that (M, N) is an M-splitting of S, and S being an M-matrix, (M, N) is also an H-splitting, according to Lemma 3.2. From Theorem 2.3, this implies that

$$\rho(|T|) < 1, \qquad T := M^{-1}N,$$

which is a sufficient condition for the convergence of an asynchronous iterative model (2.9), according to Theorem 2.2. □

Now let $(M_{p+1,p+1}, N_{p+1,p+1})$ be a splitting of $A_{p+1,p+1}$ and consider another splitting (M, N) of S given by:

$$M := M_{p+1,p+1}. \tag{3.5}$$

Definition 3.2 (Principal sub-matrix)**.** *A sub-matrix $B(i_1, \ldots, i_m; j_1, \ldots, j_m)$, with $m \in \mathbb{N}$, formed by rows $i_1, \ldots, i_m$ and columns $j_1, \ldots, j_m$ of a matrix B, is a principal sub-matrix of B when*

$$\{i_1, \ldots, i_m\} = \{j_1, \ldots, j_m\}.$$

Remark 3.1 (see, e.g., [55], p. 92)**.** *If B is an* M*-matrix, then any principal sub-matrix of B is an* M*-matrix.*

Lemma 3.3 (Crabtree and Haynsworth, 1969)**.** *If B is an* M*-matrix, then any Schur complement from B is an* M*-matrix.*

Proof. See [66]. □

Proposition 3.2. *If A is an* M*-matrix, and $(M_{p+1,p+1}, N_{p+1,p+1})$ is an* M*-splitting of $A_{p+1,p+1}$, then the splitting* (3.5) *of S induces a convergent asynchronous iterative model* (2.9)*.*

Proof. $(M_{p+1,p+1}, N_{p+1,p+1})$ being an M-splitting, we have that $M = M_{p+1,p+1}$ is an M-matrix, and

$$N_{p+1,p+1} \geq O.$$

A being an M-matrix, we have, $\forall i \in \{1, \ldots, p\}$,

$$A_{p+1,i} \leq O, \quad A_{i,p+1} \leq O,$$

and $A_{i,i}$ are M-matrices too, which implies that

$$A_{i,i}^{-1} \geq O,$$

and then

$$N = N_{p+1,p+1} + \sum_{i=1}^{p} A_{p+1,i} A_{i,i}^{-1} A_{i,p+1} \geq O.$$

It therefore follows that (M, N) is an M-splitting of S, and S being an M-matrix too, due to Lemma 3.3, (M, N) is also an H-splitting, according to Lemma 3.2. From Theorem 2.3, this implies that

$$\rho(|T|) < 1, \qquad T := M^{-1}N,$$

which is a sufficient condition for the convergence of an asynchronous iterative model (2.9), according to Theorem 2.2. □

Remark 3.2. *If a matrix $B = (b_{i,j})$ is an* M*-matrix, then the Jacobi splitting $(\mathcal{M}, \mathcal{N})$ of B is an* M*-splitting, since, on one hand, $b_{i,i} > 0$ and then $\mathcal{M}$ is an* M*-matrix, and, on the other hand, $-b_{i,j\neq i} \geq 0$ and then $\mathcal{N} \geq O$.*

Remark 3.3 (also see, e.g., Lemma 1 in [65])**.** *If A is an* M*-matrix, then we have:*

$$A_{p+1,p+1} \geq S,$$

since, from the proof of Proposition 3.2,

$$\sum_{i=1}^{p} A_{p+1,i} A_{i,i}^{-1} A_{i,p+1} \geq O.$$

According to the former splitting (3.3), Remark 3.3 therefore allows us to choose

$$\gamma \geq \max_{1 \leq i \leq n_{p+1}} a_{p+1,p+1}^{(i,i)},$$

where $a_{p+1,p+1}^{(i,i)}$ are diagonal entries of $A_{p+1,p+1}$, and thus we satisfy the convergence condition (3.4) without the explicit knowledge of the diagonal entries of S.

3.3 Vector-decomposition of assembled interface

3.3.1 Iterative model

The iterative mapping $T := M^{-1}N$ induced by the splitting (3.3) can be formulated as:

$$T = I - \frac{1}{\gamma} S,$$

which yields iterations given by:

$$\begin{aligned} x_{p+1}^{k+1} = x_{p+1}^{k} &- \frac{1}{\gamma} \left(A_{p+1,p+1} - \sum_{i=1}^{p} A_{p+1,i} A_{i,i}^{-1} A_{i,p+1} \right) x_{p+1}^{k} \\ &- \frac{1}{\gamma} \left(\sum_{i=1}^{p} A_{p+1,i} A_{i,i}^{-1} b_i - b_{p+1} \right). \end{aligned} \tag{3.6}$$

Let us consider, additionally, a partitioning of the interface nodes into p subsets, such that the matrix A can be expanded as:

$$A_{p+1,p+1} = \begin{bmatrix} A_{p+1,p+1}^{(1,1)} & \cdots & A_{p+1,p+1}^{(1,p)} \\ \vdots & \ddots & \vdots \\ A_{p+1,p+1}^{(p,1)} & \cdots & A_{p+1,p+1}^{(p,p)} \end{bmatrix},$$

$$A_{p+1,i} = \begin{bmatrix} A_{p+1,i}^{(1)} \\ \vdots \\ A_{p+1,i}^{(p)} \end{bmatrix}, \qquad A_{i,p+1} = \begin{bmatrix} A_{i,p+1}^{(1)} & \cdots & A_{i,p+1}^{(p)} \end{bmatrix}, \qquad \forall i \in \{1, \ldots, p\},$$

which yields:

$$A = \begin{bmatrix} A_{1,1} & O & \cdots & O & A_{1,p+1}^{(1)} & \cdots & A_{1,p+1}^{(p)} \\ O & A_{2,2} & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \ddots & \ddots & O & A_{p-1,p+1}^{(1)} & \cdots & A_{p-1,p+1}^{(p)} \\ O & \cdots & O & A_{p,p} & A_{p,p+1}^{(1)} & \cdots & A_{p,p+1}^{(p)} \\ A_{p+1,1}^{(1)} & \cdots & A_{p+1,p-1}^{(1)} & A_{p+1,p}^{(1)} & A_{p+1,p+1}^{(1,1)} & \cdots & A_{p+1,p+1}^{(1,p)} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{p+1,1}^{(p)} & \cdots & A_{p+1,p-1}^{(p)} & A_{p+1,p}^{(p)} & A_{p+1,p+1}^{(p,1)} & \cdots & A_{p+1,p+1}^{(p,p)} \end{bmatrix}.$$

From Figure 3.1, top, we would have, for instance, the two interface nodes subsets $\{6, 23\}$ and $\{40, 57\}$. With a corresponding interface vector x_{p+1} decomposed as:

$$x_{p+1} = \begin{bmatrix} x_{p+1}^{(1)} & \cdots & x_{p+1}^{(p)} \end{bmatrix}^{\mathsf{T}},$$

iterations (3.6) can then be distributed on each process $i \in \{1, \ldots, p\}$ as:

$$\begin{aligned} x_{p+1}^{(i),k+1} = x_{p+1}^{(i),k} &- \frac{1}{\gamma} \left(A_{p+1,p+1}^{(i,1..p)} - \sum_{j=1}^{p} A_{p+1,j}^{(i)} A_{j,j}^{-1} A_{j,p+1} \right) x_{p+1}^{k} \\ &- \frac{1}{\gamma} \left(\sum_{j=1}^{p} A_{p+1,j}^{(i)} A_{j,j}^{-1} b_j - b_{p+1}^{(i)} \right), \end{aligned}$$

where

$$A_{p+1,p+1}^{(i,1..p)} := \begin{bmatrix} A_{p+1,p+1}^{(i,1)} & \cdots & A_{p+1,p+1}^{(i,p)} \end{bmatrix}.$$

This leads to:

$$\begin{aligned} x_{p+1}^{(i),k+1} &= x_{p+1}^{(i),k} - \frac{1}{\gamma} \sum_{j=1}^{p} \left(A_{p+1,p+1}^{(i,j)} x_{p+1}^{(j),k} - A_{p+1,j}^{(i)} A_{j,j}^{-1} \left(A_{j,p+1} x_{p+1}^{k} - b_j \right) \right) + \frac{1}{\gamma} b_{p+1}^{(i)}, \\ &= x_{p+1}^{(i),k} - \frac{1}{\gamma} \sum_{j=1}^{p} \left(A_{p+1,p+1}^{(i,j)} x_{p+1}^{(j),k} - A_{p+1,j}^{(i)} A_{j,j}^{-1} \left(\sum_{l=1}^{p} A_{j,p+1}^{(l)} x_{p+1}^{(l),k} - b_j \right) \right) \\ &\quad + \frac{1}{\gamma} b_{p+1}^{(i)}, \end{aligned}$$

which is generalized, for each process $i \in P^k$, to:

$$x_{p+1}^{(i),k+1} = x_{p+1}^{(i),\tau_i^i(k)} - \frac{1}{\gamma}\sum_{j=1}^{p}\left(A_{p+1,p+1}^{(i,j)}x_{p+1}^{(j),\tau_j^i(k)} - A_{p+1,j}^{(i)}A_{j,j}^{-1}\left(\sum_{l=1}^{p}A_{j,p+1}^{(l)}x_{p+1}^{(l),\tau_l^i(k)} - b_j\right)\right) + \frac{1}{\gamma}b_{p+1}^{(i)}.$$

Consider then that each process $j \in \{1,\ldots,p\}$ evaluates

$$-A_{p+1,j}^{(i)}A_{j,j}^{-1}\left(\sum_{l=1}^{p}A_{j,p+1}^{(l)}x_{p+1}^{(l),\tau_l^i(k)} - b_j\right),$$

which however means that it accesses the delayed components $x_{p+1}^{(l),\tau_l^i(k)}$ of the process i. If we therefore take, on each process $i \in \{1,\ldots,p\}$,

$$A_{i,i}x_i^{(j),k+1} = -\sum_{l=1}^{p}A_{i,p+1}^{(l)}x_{p+1}^{(l),\tau_l^j(k)} + b_i, \quad \forall j \in \{1,\ldots,p\},$$

we thus reach an iterative model given by:

$$\begin{cases} A_{i,i}x_i^{(j),k+1} & = -\sum_{l=1}^{p}A_{i,p+1}^{(l)}x_{p+1}^{(l),\tau_l^j(k)} + b_i, \ \forall j \in \{1,\ldots,p\}, \qquad i \in \{1,\ldots,p\}, \\ x_{p+1}^{(i),k+1} & = \begin{cases} x_{p+1}^{(i),\tau_i^i(k)} - \frac{1}{\gamma}\sum_{j=1}^{p}\left(A_{p+1,p+1}^{(i,j)}x_{p+1}^{(j),\tau_j^i(k)} + A_{p+1,j}^{(i)}x_j^{(i),k+1}\right) & i \in P^k, \\ \quad + \frac{1}{\gamma}b_{p+1}^{(i)}, & \\ x_{p+1}^{(i),k}, & i \notin P^k, \end{cases} \end{cases}$$

which however synchronizes processes on interface data $A_{p+1,j}^{(i)}x_j^{(i),k+1}$.

Similarly, the splitting (3.5), which yields:

$$T := M_{p+1,p+1}^{-1}\left(N_{p+1,p+1} + \sum_{i=1}^{p}A_{p+1,i}A_{i,i}^{-1}A_{i,p+1}\right),$$

implies iterations given by:

$$\begin{aligned} M_{p+1,p+1}x_{p+1}^{k+1} = & \left(N_{p+1,p+1} + \sum_{i=1}^{p}A_{p+1,i}A_{i,i}^{-1}A_{i,p+1}\right)x_{p+1}^{k} \\ & - \left(\sum_{i=1}^{p}A_{p+1,i}A_{i,i}^{-1}b_i - b_{p+1}\right). \end{aligned} \tag{3.7}$$

Assuming then that $M_{p+1,p+1}$ is of the form

$$M_{p+1,p+1} = \begin{bmatrix} M_{p+1,p+1}^{(1,1)} & O & \cdots & O \\ O & M_{p+1,p+1}^{(2,2)} & \ddots & \vdots \\ \vdots & \ddots & \ddots & O \\ O & \cdots & O & M_{p+1,p+1}^{(p,p)} \end{bmatrix},$$

iterations (3.7) can be distributed on each process $i \in \{1, \ldots, p\}$ as:

$$\begin{aligned} M_{p+1,p+1}^{(i,i)} x_{p+1}^{(i),k+1} &= \left(N_{p+1,p+1}^{(i,1..p)} + \sum_{j=1}^{p} A_{p+1,j}^{(i)} A_{j,j}^{-1} A_{j,p+1} \right) x_{p+1}^{k} \\ &\quad - \left(\sum_{j=1}^{p} A_{p+1,j}^{(i)} A_{j,j}^{-1} b_j - b_{p+1}^{(i)} \right), \\ &= \sum_{j=1}^{p} \left(N_{p+1,p+1}^{(i,j)} x_{p+1}^{(j),k} + A_{p+1,j}^{(i)} A_{j,j}^{-1} \left(\sum_{l=1}^{p} A_{j,p+1}^{(l)} x_{p+1}^{(l),k} - b_j \right) \right) \\ &\quad + b_{p+1}^{(i)}, \end{aligned}$$

which is generalized, for each process $i \in P^k$, to:

$$\begin{aligned} M_{p+1,p+1}^{(i,i)} x_{p+1}^{(i),k+1} &= \sum_{j=1}^{p} \left(N_{p+1,p+1}^{(i,j)} x_{p+1}^{(j),\tau_j^i(k)} + A_{p+1,j}^{(i)} A_{j,j}^{-1} \left(\sum_{l=1}^{p} A_{j,p+1}^{(l)} x_{p+1}^{(l),\tau_l^i(k)} - b_j \right) \right) \\ &\quad + b_{p+1}^{(i)}. \end{aligned}$$

We thus reach another partitions-based parallel iterative model given by:

$$\begin{cases} A_{i,i} x_i^{(j),k+1} & = -\sum_{l=1}^{p} A_{i,p+1}^{(l)} x_{p+1}^{(l),\tau_l^j(k)} + b_i, \ \forall j \in \{1, \ldots, p\}, \qquad i \in \{1, \ldots, p\}, \\ M_{p+1,p+1}^{(i,i)} x_{p+1}^{(i),k+1} & = \begin{cases} \sum_{j=1}^{p} \left(N_{p+1,p+1}^{(i,j)} x_{p+1}^{(j),\tau_j^i(k)} - A_{p+1,j}^{(i)} x_j^{(i),k+1} \right) + b_{p+1}^{(i)}, & i \in P^k, \\ M_{p+1,p+1}^{(i,i)} x_{p+1}^{(i),k}, & i \notin P^k, \end{cases} \end{cases}$$

which, just as the former splitting, also suggests both remote access and partial synchronization.

Let us therefore consider a more desirable parallel scheme given by:

$$\begin{cases} A_{i,i} x_i^{k+1} & = -\sum_{j=1}^{p} A_{i,p+1}^{(j)} x_{p+1}^{(j),k} + b_i, \\ M_{p+1,p+1}^{(i,i)} x_{p+1}^{(i),k+1} & = \sum_{j=1}^{p} \left(N_{p+1,p+1}^{(i,j)} x_{p+1}^{(j),k} - A_{p+1,j}^{(i)} x_j^{k+1} \right) + b_{p+1}^{(i)}, \end{cases}$$

where the inter-process communication can be restricted to interface data $x_{p+1}^{(j)}$ and $A_{p+1,j}^{(i)}x_j$, at the small computational cost of additionally evaluating, in each process i, the interface products

$$A_{p+1,i}^{(1)}x_i, \ldots, A_{p+1,i}^{(i-1)}x_i, A_{p+1,i}^{(i+1)}x_i, \ldots, A_{p+1,i}^{(p)}x_i.$$

Obviously, by fully expressing x_j^{k+1} in the second relation, we still verify:

$$M_{p+1,p+1}^{(i,i)}x_{p+1}^{(i),k+1} = \sum_{j=1}^{p}\left(N_{p+1,p+1}^{(i,j)}x_{p+1}^{(j),k} + A_{p+1,j}^{(i)}A_{j,j}^{-1}\left(\sum_{l=1}^{p}A_{j,p+1}^{(l)}x_{p+1}^{(l),k} - b_j\right)\right) + b_{p+1}^{(i)},$$

$$M_{p+1,p+1}x_{p+1}^{k+1} = \left(N_{p+1,p+1} + \sum_{j=1}^{p}A_{p+1,j}A_{j,j}^{-1}A_{j,p+1}\right)x_{p+1}^{k} - \sum_{j=1}^{p}A_{p+1,j}A_{j,j}^{-1}b_j + b_{p+1},$$

which corresponds to the same latter splitting (3.5) of the Schur complement S. We will then consider in the sequel the fully asynchronous iterative model given by:

$$\begin{cases} A_{i,i}x_i^{k+1} & = \begin{cases} -\displaystyle\sum_{j=1}^{p}A_{i,p+1}^{(j)}x_{p+1}^{(j),\tau_j^i(k)} + b_i, & i \in P^k, \\ A_{i,i}x_i^k, & i \notin P^k, \end{cases} \\ M_{p+1,p+1}^{(i,i)}x_{p+1}^{(i),k+1} & = \begin{cases} \displaystyle\sum_{j=1}^{p}\left(N_{p+1,p+1}^{(i,j)}x_{p+1}^{(j),\tau_j^i(k)} - A_{p+1,j}^{(i)}x_j^{\rho_j^i(k+1)}\right) & i \in P^k, \\ \quad + b_{p+1}^{(i)}, & \\ M_{p+1,p+1}^{(i,i)}x_{p+1}^{(i),k}, & i \notin P^k. \end{cases} \end{cases} \tag{3.8}$$

3.3.2 Asynchronous convergence

Lemma 3.4 (see, e.g., Corollary 1.6 in [55])**.** *Let B be a square matrix. Then, for any matrix norm $\|.\|$, we have:*

$$\rho(B) \leq \|B\|.$$

Proof. Let λ be an eigenvalue of B and let w be the associated eigenvector. Then we have:

$$\|B\|\|w\| \geq \|Bw\| = \|\lambda w\| = |\lambda|\ \|w\|,$$

which implies:

$$\|B\| \geq |\lambda|,$$

and concludes the proof. □

Definition 3.3 (Matrix weighted maximum norm)**.** *The matrix norm induced by a weighted maximum norm* (2.14) *applied to scalar components is given by:*

$$\|B\|_\infty^w := \max_i \frac{1}{w_i} \sum_j |b_{i,j}| w_j,$$

where $B = (b_{i,j})$ is a square matrix, and $w = (w_i)$, a strictly positive vector.

Notation 3.1. *Let $B = (b_{i,j})$ be a matrix, and let $w = (w_j)$ and $v = (v_i)$, with $v > 0$, be two vectors having as many components as, respectively, the number of columns and the number of rows in B. We denote by $|B|_v^w = ((|B|_v^w)_i)$ the vector given by:*

$$(|B|_v^w)_i := \frac{1}{v_i} \sum_j |b_{i,j}| w_j.$$

Notation 3.2 (Component-wise vectors division)**.** *Let $w = (w_i)$ and $v = (v_i)$ be two vectors of the same size, with $v > 0$. We denote by $w/v = ((w/v)_i)$ the vector given by:*

$$(w/v)_i := \frac{w_i}{v_i}.$$

Remark 3.4. *We have:*

$$\|B\|_\infty^w = \max_i |B|_w^w,$$

and we also have:

$$|B|_v^w = (|B|w)/v = \sum_j w_j |B_j|/v,$$

where B_j is the j-th column of B.

Lemma 3.5 (see, e.g., Corollary 6.1 in [67])**.** *Let B be a square matrix. Then the three following statements are equivalent:*

1. $\rho(|B|) < 1$.
2. $\exists\ w > 0 : \|B\|_\infty^w < 1$.
3. $\exists\ \alpha \in (0,1), \exists\ w > 0 : |B|\, w \leq \alpha w$.

Proof. Assume that the statement 1 holds. Then we have:

$$\exists \epsilon > 0 : \quad \rho(|B|) + \epsilon < 1.$$

From Perron–Frobenius theory of nonnegative matrices (see, e.g., Proposition 6.6 in [67]), we have that for every $\epsilon > 0$,

$$\exists w > 0 : \quad \||B|\|_\infty^w \leq \rho(|B|) + \epsilon.$$

We thus deduce the statement 2, for such a vector w, by:

$$\|B\|_\infty^w = \||B|\|_\infty^w \leq \rho(|B|) + \epsilon < 1.$$

Now assume that the statement 2 holds. Then, according to Notation 3.1, Notation 3.2 and Remark 3.4, we have:

$$\|B\|_\infty^w = \max_i |B|_w^w = \max_i \left(|B|w\right)/w < 1,$$

which means:

$$\begin{aligned} \frac{(|B|w)_i}{w_i} < 1, \quad \forall i, \\ |B|\, w < w, \end{aligned}$$

and therefore,

$$\exists \alpha \in (0,1): \quad |B|\, w \leq \alpha w.$$

Finally by assuming the statement 3, we have:

$$\begin{aligned} (|B|\, w)_i \leq \alpha w_i, \quad \forall i, \\ \sum_j |b_{i,j}| w_j \leq \alpha w_i, \quad \forall i, \\ \frac{1}{w_i} \sum_j |b_{i,j}| w_j \leq \alpha, \qquad \forall i, \end{aligned}$$

which means:

$$\||B|\|_\infty^w = \|B\|_\infty^w \leq \alpha < 1.$$

By considering Lemma 3.4, we deduce the statement 1 by:

$$\rho(|B|) \leq \||B|\|_\infty^w < 1,$$

which concludes the proof. □

Lemma 3.6. *Let $C = (c_{i,j})$ and $B = (b_{j,l})$ be two matrices such that the number of columns in C equals the number of rows in B. Let $z = (z_j)$, with $z > 0$, $v = (v_i)$, with $v > 0$ and $w = (w_l)$ be three vectors having as many components as, respectively, the number of columns in C, the number of rows in C and the number of columns in B. Let, at last, $u = (u_j)$ be the vector with as many components as the number of rows in B and given by:*

$$u_j := 1, \quad \forall j.$$

Then we have:

$$|B|_z^w < u \quad \Longrightarrow \quad |CB|_v^w < |C|_v^z.$$

Proof. We have, according to Remark 3.4:

$$|CB|_v^w = \sum_l w_l |(CB)_l|/v,$$

where $(CB)_l$ is the l-th column of the product $(CB) = ((CB)_{i,l})$. We also have:

$$(CB)_{i,l} := \sum_j c_{i,j} b_{j,l},$$
$$(CB)_l = \sum_j b_{j,l} C_j,$$

where C_j is the j-th column of C. These imply:

$$\begin{aligned} |CB|_v^w &= \sum_l w_l \left| \sum_j b_{j,l} C_j \right| /v \\ &\leq \sum_l w_l \sum_j |b_{j,l} C_j|/v \\ &= \sum_l \sum_j w_l |b_{j,l}||C_j|/v. \end{aligned} \tag{3.9}$$

By Notation 3.1, each entry of the vector $|B|_z^w = ((|B|_z^w)_j)$ is given by:

$$(|B|_z^w)_j := \sum_l w_l \frac{|b_{j,l}|}{z_j}.$$

This implies then, by resuming (3.9):

$$\begin{aligned} |CB|_v^w &\leq \sum_l \sum_j w_l |b_{j,l}||C_j|/v \\ &= \sum_l \sum_j w_l \frac{|b_{j,l}|}{z_j} z_j |C_j|/v \\ &= \sum_j \left(\sum_l w_l \frac{|b_{j,l}|}{z_j} \right) z_j |C_j|/v \\ &= \sum_j (|B|_z^w)_j z_j |C_j|/v, \end{aligned}$$

which means, for each entry of the vector $|CB|_v^w = ((|CB|_v^w)_i)$:

$$(|CB|_v^w)_i \leq \sum_j (|B|_z^w)_j z_j (|C_j|/v)_i,$$

where $(|C_j|/v)_i$ is the i-th entry of the vector $|C_j|/v$. If therefore we have:

$$|B|_z^w < u,$$

which means:

$$(|B|_z^w)_j < u_j = 1, \qquad \forall j,$$

then we also have:

$$\begin{aligned} (|B|_z^w)_j z_j (|C_j|/v)_i < z_j (|C_j|/v)_i, \qquad & \forall j,\ \forall i, \\ (|CB|_v^w)_i \leq \sum_j (|B|_z^w)_j z_j (|C_j|/v)_i < \sum_j z_j (|C_j|/v)_i, \qquad & \forall i, \end{aligned}$$

which means:

$$|CB|_v^w < \sum_j z_j |C_j|/v.$$

This concludes the proof, given that (still from Remark 3.4):

$$|C|_v^z = \sum_j z_j |C_j|/v.$$

□

Corollary 3.1. *Let B be a square matrix block-decomposed as:*

$$B = \begin{bmatrix} O & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}.$$

Then we have:

$$\rho(|B|) < 1 \quad \Longrightarrow \quad \rho(|B_{2,2}| + |B_{2,1}B_{1,2}|) < 1.$$

Proof. According to Lemma 3.5, we have:

$$\rho(|B|) < 1 \quad \Longrightarrow \quad \exists w > 0 : \|B\|_\infty^w < 1.$$

Let such a vector w be decomposed as:

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix},$$

and let $u_1 = (1, 1, ..., 1)$ and $u_2 = (1, 1, ..., 1)$ be respective size-corresponding vectors of units. Then, recalling that

$$\|B\|_\infty^w = \max |B|_w^w,$$

we also have:

$$\begin{aligned} \max |B_{1,2}|_{w_1}^{w_2} < 1, \qquad & \max \left(|B_{2,1}|_{w_2}^{w_1} + |B_{2,2}|_{w_2}^{w_2} \right) < 1, \\ |B_{1,2}|_{w_1}^{w_2} < u_1, \qquad & |B_{2,1}|_{w_2}^{w_1} + |B_{2,2}|_{w_2}^{w_2} < u_2. \end{aligned}$$

Lemma 3.6 therefore implies:

$$\begin{aligned} |B_{2,1}B_{1,2}|_{w_2}^{w_2} &< |B_{2,1}|_{w_2}^{w_1}, \\ |B_{2,1}B_{1,2}|_{w_2}^{w_2} + |B_{2,2}|_{w_2}^{w_2} &< |B_{2,1}|_{w_2}^{w_1} + |B_{2,2}|_{w_2}^{w_2} < u_2, \\ |(|B_{2,1}B_{1,2}| + |B_{2,2}|)|_{w_2}^{w_2} &< u_2, \\ \max |(|B_{2,1}B_{1,2}| + |B_{2,2}|)|_{w_2}^{w_2} &< 1, \\ \|(|B_{2,1}B_{1,2}| + |B_{2,2}|)\|_{\infty}^{w_2} &< 1, \end{aligned}$$

which concludes the proof, given that

$$\rho(|B_{2,1}B_{1,2}| + |B_{2,2}|) \leq \|(|B_{2,1}B_{1,2}| + |B_{2,2}|)\|_{\infty}^{w_2},$$

according to Lemma 3.4. □

Remark 3.5. *Let B be a square matrix block-decomposed as:*

$$B = \begin{bmatrix} O & \cdots & O & B_{1,m} \\ \vdots & \ddots & \vdots & \vdots \\ O & \cdots & O & B_{m-1,m} \\ B_{m,1} & \cdots & B_{m,m-1} & B_{m,m} \end{bmatrix},$$

with $m \in \mathbb{N}$. Then Corollary 3.1 is obviously generalized as follows:

$$\rho(|B|) < 1 \quad \Longrightarrow \quad \rho(|B_{m,m}| + \sum_{i=1}^{m-1} |B_{m,i}B_{i,m}|) < 1,$$

by using same proof principles and Lemma 3.6 to notice that:

$$|B_{m,i}B_{i,m}|_{w_m}^{w_m} < |B_{m,i}|_{w_m}^{w_i}, \quad \forall i \in \{1, \ldots, m-1\}.$$

Theorem 3.1. *Let $(M_{p+1,p+1}, N_{p+1,p+1})$ be a splitting of $A_{p+1,p+1}$, which yields a splitting* (3.5) *of the Schur complement*

$$S := A_{p+1,p+1} - \sum_{i=1}^{p} A_{p+1,i} A_{i,i}^{-1} A_{i,p+1}.$$

Let $(\mathcal{M}, \mathcal{N})$ *be a corresponding splitting of the initial matrix* A*, given by:*

$$\mathcal{M} := \begin{bmatrix} A_{1,1} & O & \cdots & O & O \\ O & A_{2,2} & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & O & O \\ O & \cdots & O & A_{p,p} & O \\ O & \cdots & O & O & M_{p+1,p+1} \end{bmatrix}.$$

If we have:

$$\rho(|\mathcal{T}|) < 1, \qquad \mathcal{T} := \mathcal{M}^{-1}\mathcal{N},$$

then the computational model (3.8) *is convergent.*

Proof. From the first equation of the iterative model (3.8), we have:

$$x_j^{\rho_j^i(k+1)} = -A_{j,j}^{-1}\left(\sum_{l=1}^{p} A_{j,p+1}^{(l)} x_{p+1}^{(l),\tau_l^j(\rho_j^i(k+1)-1)} - b_j\right), \quad \forall j \in \{1,\ldots,p\}, \quad i \in P^k.$$

We can thus rewrite iterations (3.8) as, for $i \in P^k$:

$$M_{p+1,p+1}^{(i,i)} x_{p+1}^{(i),k+1} = \sum_{j=1}^{p}\Big(N_{p+1,p+1}^{(i,j)} x_{p+1}^{(j),\tau_j^i(k)} + A_{p+1,j}^{(i)} A_{j,j}^{-1}\left(\sum_{l=1}^{p} A_{j,p+1}^{(l)} x_{p+1}^{(l),\tau_l^j(\rho_j^i(k+1)-1)} - b_j\right)\Big) + b_{p+1}^{(i)}.$$

This implies supplementary delay functions $\tau_{l,j}^i$ given by:

$$\tau_{l,j}^i(k) := \tau_l^j(\rho_j^i(k+1) - 1),$$

which still satisfy:

$$k \to \infty \quad \Longrightarrow \quad \rho_j^i(k+1) - 1 \to \infty \quad \Longrightarrow \quad \tau_{l,j}^i(k) \to \infty,$$

and verify:

$$\rho_j^i(k+1) \leq k+1,$$
$$\tau_{l,j}^i(k) \leq \rho_j^i(k+1) - 1 \leq k.$$

We thus have:

$$M_{p+1,p+1}^{(i,i)} x_{p+1}^{(i),k+1} = \sum_{j=1}^{p}\left(N_{p+1,p+1}^{(i,j)} x_{p+1}^{(j),\tau_j^i(k)} + A_{p+1,j}^{(i)} A_{j,j}^{-1}\left(\sum_{l=1}^{p} A_{j,p+1}^{(l)} x_{p+1}^{(l),\tau_{l,j}^i(k)} - b_j\right)\right) + b_{p+1}^{(i)},$$

which induces a mapping F given by:

$$F_i(\widetilde{x},\widetilde{y}^1,\ldots,\widetilde{y}^p) := M_{p+1,p+1}^{(i,i)^{-1}}\left(N_{p+1,p+1}^{(i,1..p)}\widetilde{x}+\sum_{j=1}^{p}A_{p+1,j}^{(i)}A_{j,j}^{-1}\left(A_{j,p+1}\widetilde{y}^j-b_j\right)+b_{p+1}^{(i)}\right),$$

$$F(\widetilde{x},\widetilde{y}^1,\ldots,\widetilde{y}^p) := M_{p+1,p+1}^{-1}\left(N_{p+1,p+1}\widetilde{x}+\sum_{j=1}^{p}A_{p+1,j}A_{j,j}^{-1}\left(A_{j,p+1}\widetilde{y}^j-b_j\right)+b_{p+1}\right),$$

with, by analogy,

$$\widetilde{x}=\begin{bmatrix}x_{p+1}^{(1),\tau_1^i(k)}\\ \vdots\\ x_{p+1}^{(p),\tau_p^i(k)}\end{bmatrix},\qquad \widetilde{y}^j=\begin{bmatrix}x_{p+1}^{(1),\tau_{1,j}^i(k)}\\ \vdots\\ x_{p+1}^{(p),\tau_{p,j}^i(k)}\end{bmatrix},\quad 1\leq j\leq p.$$

The computational model (3.8) thus lies in the general framework (2.20) of asynchronous iterations with memory. Let us then consider

$$\widetilde{X}=\left(\widetilde{x},\widetilde{y}^1,\ldots,\widetilde{y}^p\right),\qquad \widehat{X}=\left(\widehat{x},\widehat{y}^1,\ldots,\widehat{y}^p\right).$$

We have:

$$\begin{aligned}\left|F(\widetilde{X})-F(\widehat{X})\right| &= \left|M_{p+1,p+1}^{-1}\left(N_{p+1,p+1}(\widetilde{x}-\widehat{x})+\sum_{j=1}^{p}A_{p+1,j}A_{j,j}^{-1}A_{j,p+1}(\widetilde{y}^j-\widehat{y}^j)\right)\right|\\ &\leq \left|M_{p+1,p+1}^{-1}N_{p+1,p+1}\right|\left|\widetilde{x}-\widehat{x}\right|\\ &\quad+\sum_{j=1}^{p}\left|M_{p+1,p+1}^{-1}A_{p+1,j}A_{j,j}^{-1}A_{j,p+1}\right|\left|\widetilde{y}^j-\widehat{y}^j\right|\\ &\leq \left(\left|M_{p+1,p+1}^{-1}N_{p+1,p+1}\right|+\sum_{j=1}^{p}\left|M_{p+1,p+1}^{-1}A_{p+1,j}A_{j,j}^{-1}A_{j,p+1}\right|\right)\\ &\quad\max\left\{\left|\widetilde{x}-\widehat{x}\right|,\max_{1\leq j\leq p}\left\{\left|\widetilde{y}^j-\widehat{y}^j\right|\right\}\right\},\end{aligned}$$

and let us define:

$$Q := \left|M_{p+1,p+1}^{-1}N_{p+1,p+1}\right|+\sum_{j=1}^{p}\left|M_{p+1,p+1}^{-1}A_{p+1,j}A_{j,j}^{-1}A_{j,p+1}\right|.$$

According then to Theorem 2.10, convergence is guaranteed if

$$\rho\left(Q\right)<1,$$

which is easily deduced from Corollary 3.1, given that we assume:

$$\rho\left(|\mathcal{T}|\right)<1,\qquad \mathcal{T}:=\mathcal{M}^{-1}\mathcal{N},$$

and by noticing (see Remark 3.5) that the mapping $\mathcal{T}$ is explicitly given by:

$$\mathcal{T} = \begin{bmatrix} O & \cdots & O & -A_{1,1}^{-1}A_{1,p+1} \\ \vdots & \ddots & \vdots & \vdots \\ O & \cdots & O & -A_{p,p}^{-1}A_{p,p+1} \\ -M_{p+1,p+1}^{-1}A_{p+1,1} & \cdots & -M_{p+1,p+1}^{-1}A_{p+1,p} & M_{p+1,p+1}^{-1}N_{p+1,p+1} \end{bmatrix}.$$

□

3.4 Sub-structuring method

3.4.1 Iterative model

Consider again the splitting (3.5), yielding:

$$S = M_{p+1,p+1} - \left(N_{p+1,p+1} + \sum_{i=1}^{p} A_{p+1,i}A_{i,i}^{-1}A_{i,p+1} \right),$$

and iterations

$$M_{p+1,p+1}x_{p+1}^{k+1} = \left(N_{p+1,p+1} + \sum_{i=1}^{p} A_{p+1,i}A_{i,i}^{-1}A_{i,p+1} \right) x_{p+1}^{k} - \left(\sum_{i=1}^{p} A_{p+1,i}A_{i,i}^{-1}b_i - b_{p+1} \right).$$

Let us consider, additionally, a substructures-based distribution of $N_{p+1,p+1}$ given by:

$$N_{p+1,p+1} = \sum_{i=1}^{p} N_{p+1,p+1}^{(i)}, \tag{3.10}$$

which yields:

$$\begin{aligned} M_{p+1,p+1}x_{p+1}^{k+1} &= \left(\sum_{i=1}^{p} N_{p+1,p+1}^{(i)} + \sum_{i=1}^{p} A_{p+1,i}A_{i,i}^{-1}A_{i,p+1} \right) x_{p+1}^{k} - \sum_{i=1}^{p} A_{p+1,i}A_{i,i}^{-1}b_i \\ &\quad + b_{p+1} \\ &= \sum_{i=1}^{p} \left(N_{p+1,p+1}^{(i)} + A_{p+1,i}A_{i,i}^{-1}\left(A_{i,p+1}x_{p+1}^{k} - b_i \right) \right) + b_{p+1}. \end{aligned}$$

If, for instance, $M_{p+1,p+1}$ is the diagonal part of $A_{p+1,p+1}$, such a distribution (3.10) of its off-diagonal coefficients can be seen as tearing the interface edges or faces, as shown in Figure 3.2. Let us then consider p identical iterations, each one given by:

$$M_{p+1,p+1}x_{p+1}^{(i),k+1} = \sum_{j=1}^{p} \left(N_{p+1,p+1}^{(j)} + A_{p+1,j}A_{j,j}^{-1}\left(A_{j,p+1}x_{p+1}^{(i),k} - b_j \right) \right) + b_{p+1},$$

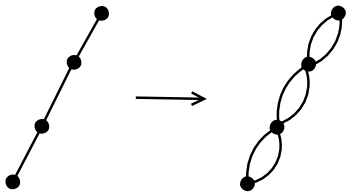


Figure 3.2: Interface edges tearing.

which, furthermore, are interleaved by using each other iterative solution, such that:

$$M_{p+1,p+1}x_{p+1}^{(i),k+1} = \sum_{j=1}^{p}\left(N_{p+1,p+1}^{(j)} + A_{p+1,j}A_{j,j}^{-1}\left(A_{j,p+1}x_{p+1}^{(j),k} - b_j\right)\right) + b_{p+1}.$$

Therefore, by taking on each process i, with $1 \leq i \leq p$:

$$A_{i,i}x_i^{k+1} = -A_{i,p+1}x_{p+1}^{(i),k} + b_i,$$

and, afterward:

$$y_{p+1}^{(i),k+1} = N_{p+1,p+1}^{(i)}x_{p+1}^{(i),k} - A_{p+1,i}x_i^{k+1},$$

we can write:

$$M_{p+1,p+1}x_{p+1}^{(i),k+1} = \sum_{j=1}^{p} y_{p+1}^{(j),k+1} + b_{p+1}.$$

While such an algorithm is still sequential, we can generalize it (and its convergence analysis) to a fully parallel sub-structuring method by means of the asynchronous iterative model given by:

$$\begin{cases} A_{i,i}x_i^{k+1} & = -A_{i,p+1}x_{p+1}^{(i),k} + b_i, & i \in P^k, \\ y_{p+1}^{(i),k+1} & = N_{p+1,p+1}^{(i)}x_{p+1}^{(i),k} - A_{p+1,i}x_i^{k+1}, & i \in P^k, \\ M_{p+1,p+1}x_{p+1}^{(i),k+1} & = \begin{cases} \displaystyle\sum_{j=1}^{p} y_{p+1}^{(j),\tau_j^i(k+1)} + b_{p+1}, & i \in P^k, \\ M_{p+1,p+1}x_{p+1}^{(i),k}, & i \notin P^k. \end{cases} \end{cases} \tag{3.11}$$

3.4.2 Asynchronous convergence

Lemma 3.7. *Let*

$$\mathcal{F}_1 : E \times E \mapsto E, \qquad \mathcal{F}_2 : E \times E \mapsto E,$$

be two mappings, and let $\mathcal{P}_1 \geq O$, $\mathcal{P}_2 \geq O$ be two nonnegative matrices, with:

$$\rho(\mathcal{P}_1) < 1, \qquad \rho(\mathcal{P}_2) < 1,$$

and such that, for any vectors $\widetilde{x}, \widetilde{y}, \widehat{x}, \widehat{y} \in E$,

$$|\mathcal{F}_i(\widetilde{x}, \widetilde{y}) - \mathcal{F}_i(\widehat{x}, \widehat{y})| \leq \mathcal{P}_i \max\left\{|\widetilde{x} - \widehat{x}|, |\widetilde{y} - \widehat{y}|\right\}, \quad \forall i \in \{1, 2\}.$$

Now let

$$\mathcal{H} : E^4 \mapsto E^2,$$

be a mapping given by:

$$\mathcal{H}(\widetilde{x}_1, \widetilde{y}_1, \widetilde{x}_2, \widetilde{y}_2) := \begin{bmatrix} \mathcal{F}_1(\widetilde{x}_1, \widetilde{y}_1) \\ \mathcal{F}_2(\widetilde{x}_2, \widetilde{y}_2) \end{bmatrix}.$$

Then, there exists a matrix $\mathcal{Q} \geq O$, *with*

$$\rho(\mathcal{Q}) < 1,$$

such that, for any vectors $\widetilde{x}_i, \widetilde{y}_i, \widehat{x}_i, \widehat{y}_i \in E$, *with* $i := 1, 2$,

$$|\mathcal{H}(\widetilde{x}_1, \widetilde{y}_1, \widetilde{x}_2, \widetilde{y}_2) - \mathcal{H}(\widehat{x}_1, \widehat{y}_1, \widehat{x}_2, \widehat{y}_2)| \leq \mathcal{Q} \max\left\{ \left| \begin{bmatrix} \widetilde{x}_1 \\ \widetilde{x}_2 \end{bmatrix} - \begin{bmatrix} \widehat{x}_1 \\ \widehat{x}_2 \end{bmatrix} \right|, \left| \begin{bmatrix} \widetilde{y}_1 \\ \widetilde{y}_2 \end{bmatrix} - \begin{bmatrix} \widehat{y}_1 \\ \widehat{y}_2 \end{bmatrix} \right| \right\}.$$

Proof. We have:

$$\begin{aligned} |\mathcal{H}(\widetilde{x}_1, \widetilde{y}_1, \widetilde{x}_2, \widetilde{y}_2) - \mathcal{H}(\widehat{x}_1, \widehat{y}_1, \widehat{x}_2, \widehat{y}_2)| &= \begin{bmatrix} |\mathcal{F}_1(\widetilde{x}_1, \widetilde{y}_1) - \mathcal{F}_1(\widehat{x}_1, \widehat{y}_1)| \\ |\mathcal{F}_2(\widetilde{x}_2, \widetilde{y}_2) - \mathcal{F}_2(\widehat{x}_2, \widehat{y}_2)| \end{bmatrix} \\ &\leq \begin{bmatrix} \mathcal{P}_1 \max\left\{|\widetilde{x}_1 - \widehat{x}_1|, |\widetilde{y}_1 - \widehat{y}_1|\right\} \\ \mathcal{P}_2 \max\left\{|\widetilde{x}_2 - \widehat{x}_2|, |\widetilde{y}_2 - \widehat{y}_2|\right\} \end{bmatrix} \\ &= \begin{bmatrix} \mathcal{P}_1 & O \\ O & \mathcal{P}_2 \end{bmatrix} \begin{bmatrix} \max\left\{|\widetilde{x}_1 - \widehat{x}_1|, |\widetilde{y}_1 - \widehat{y}_1|\right\} \\ \max\left\{|\widetilde{x}_2 - \widehat{x}_2|, |\widetilde{y}_2 - \widehat{y}_2|\right\} \end{bmatrix} \\ &= \mathcal{Q} \max\left\{ \left| \begin{bmatrix} \widetilde{x}_1 \\ \widetilde{x}_2 \end{bmatrix} - \begin{bmatrix} \widehat{x}_1 \\ \widehat{x}_2 \end{bmatrix} \right|, \left| \begin{bmatrix} \widetilde{y}_1 \\ \widetilde{y}_2 \end{bmatrix} - \begin{bmatrix} \widehat{y}_1 \\ \widehat{y}_2 \end{bmatrix} \right| \right\}, \end{aligned}$$

with

$$\mathcal{Q} := \begin{bmatrix} \mathcal{P}_1 & O \\ O & \mathcal{P}_2 \end{bmatrix}.$$

$\rho(\mathcal{Q}) < 1$ then obviously follows from $\rho(\mathcal{P}_1) < 1$ and $\rho(\mathcal{P}_2) < 1$, which concludes the proof. □

Remark 3.6. *Lemma 3.7 can be recursively applied to the case of* $m \in \mathbb{N}$ *contracting mappings*

$$\mathcal{F}_i : E^m \mapsto E, \qquad 1 \leq i \leq m,$$

yielding a global contracting mapping

$$\mathcal{H} := \begin{bmatrix} \mathcal{F}_1 \\ \vdots \\ \mathcal{F}_m \end{bmatrix}.$$

In the case of the computational model (3.11)*, we have by analogy:*

$$\mathcal{F}_1 \equiv \mathcal{F}_2 \equiv \cdots \equiv \mathcal{F}_p,$$

therefore its convergence can be analyzed through the sole unique mapping applied on every process. Obviously, at convergence, one should thus have:

$$x_{p+1}^{(1)*} = x_{p+1}^{(2)*} = \cdots = x_{p+1}^{(p)*}.$$

Theorem 3.2. *Let* $(M_{p+1,p+1}, N_{p+1,p+1})$ *be a splitting of* $A_{p+1,p+1}$*, which yields a splitting* (3.5) *of the Schur complement*

$$S := A_{p+1,p+1} - \sum_{i=1}^{p} A_{p+1,i} A_{i,i}^{-1} A_{i,p+1}.$$

Let $(\mathcal{M}, \mathcal{N})$ *be a corresponding splitting of the initial matrix* A*, given by:*

$$\mathcal{M} := \begin{bmatrix} A_{1,1} & O & \cdots & O & O \\ O & A_{2,2} & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & O & O \\ O & \cdots & O & A_{p,p} & O \\ O & \cdots & O & O & M_{p+1,p+1} \end{bmatrix}.$$

Furthermore, let $N_{p+1,p+1}$ *be given by:*

$$N_{p+1,p+1} = \sum_{i=1}^{p} N_{p+1,p+1}^{(i)}.$$

If we have:

$$\rho(|\mathcal{T}|) < 1, \qquad \mathcal{T} := \mathcal{M}^{-1}\mathcal{N},$$

and, additionally:

$$\left| \sum_{i=1}^{p} M_{p+1,p+1}^{-1} N_{p+1,p+1}^{(i)} \right| = \sum_{i=1}^{p} \left| M_{p+1,p+1}^{-1} N_{p+1,p+1}^{(i)} \right|,$$

then the computational model (3.11) *is convergent.*

Proof. Following the same pattern as in the proof of Theorem 3.1, we have that the computational model (3.11) induces a mapping F given by:

$$F(\widetilde{y}^1,\ldots,\widetilde{y}^p) := M^{-1}_{p+1,p+1}\left(\sum_{j=1}^{p}\left(N^{(j)}_{p+1,p+1} + A_{p+1,j}A^{-1}_{j,j}\left(A_{j,p+1}\widetilde{y}^j - b_j\right)\right) + b_{p+1}\right).$$

Let us then consider

$$\widetilde{X} = \left(\widetilde{y}^1,\ldots,\widetilde{y}^p\right), \qquad \widehat{X} = \left(\widehat{y}^1,\ldots,\widehat{y}^p\right).$$

We have:

$$\begin{aligned}
\left|F(\widetilde{X}) - F(\widehat{X})\right| &= \left|M^{-1}_{p+1,p+1}\sum_{j=1}^{p}\left(N^{(j)}_{p+1,p+1} + A_{p+1,j}A^{-1}_{j,j}A_{j,p+1}\right)\left(\widetilde{y}^j - \widehat{y}^j\right)\right| \\
&\leq \sum_{j=1}^{p}\left|M^{-1}_{p+1,p+1}\left(N^{(j)}_{p+1,p+1} + A_{p+1,j}A^{-1}_{j,j}A_{j,p+1}\right)\right| \max_{1\leq j\leq p}\left\{\left|\widetilde{y}^j - \widehat{y}^j\right|\right\} \\
&\leq \sum_{j=1}^{p}\left(\left|M^{-1}_{p+1,p+1}N^{(j)}_{p+1,p+1}\right| + \left|M^{-1}_{p+1,p+1}A_{p+1,j}A^{-1}_{j,j}A_{j,p+1}\right|\right) \\
&\quad \max_{1\leq j\leq p}\left\{\left|\widetilde{y}^j - \widehat{y}^j\right|\right\} \\
&= \left(\sum_{j=1}^{p}\left|M^{-1}_{p+1,p+1}N^{(j)}_{p+1,p+1}\right| + \sum_{j=1}^{p}\left|M^{-1}_{p+1,p+1}A_{p+1,j}A^{-1}_{j,j}A_{j,p+1}\right|\right) \\
&\quad \max_{1\leq j\leq p}\left\{\left|\widetilde{y}^j - \widehat{y}^j\right|\right\},
\end{aligned}$$

and let us define:

$$Q := \sum_{j=1}^{p}\left|M^{-1}_{p+1,p+1}N^{(j)}_{p+1,p+1}\right| + \sum_{j=1}^{p}\left|M^{-1}_{p+1,p+1}A_{p+1,j}A^{-1}_{j,j}A_{j,p+1}\right|.$$

If therefore

$$\left|\sum_{j=1}^{p}M^{-1}_{p+1,p+1}N^{(j)}_{p+1,p+1}\right| = \sum_{j=1}^{p}\left|M^{-1}_{p+1,p+1}N^{(j)}_{p+1,p+1}\right|,$$

we have:

$$\begin{aligned}
Q &= \left|\sum_{j=1}^{p}M^{-1}_{p+1,p+1}N^{(j)}_{p+1,p+1}\right| + \sum_{j=1}^{p}\left|M^{-1}_{p+1,p+1}A_{p+1,j}A^{-1}_{j,j}A_{j,p+1}\right| \\
&= \left|M^{-1}_{p+1,p+1}N_{p+1,p+1}\right| + \sum_{j=1}^{p}\left|M^{-1}_{p+1,p+1}A_{p+1,j}A^{-1}_{j,j}A_{j,p+1}\right|.
\end{aligned}$$

According then to Theorem 2.10, convergence is guaranteed if

$$\rho\left(Q\right) < 1,$$

which is easily deduced from Corollary 3.1, given that we assume:

$$\rho\left(|\mathcal{T}|\right) < 1, \qquad \mathcal{T} := \mathcal{M}^{-1}\mathcal{N},$$

and by noticing (see Remark 3.5) that the mapping $\mathcal{T}$ is explicitly given by:

$$\mathcal{T} = \begin{bmatrix} O & \cdots & O & -A_{1,1}^{-1}A_{1,p+1} \\ \vdots & \ddots & \vdots & \vdots \\ O & \cdots & O & -A_{p,p}^{-1}A_{p,p+1} \\ -M_{p+1,p+1}^{-1}A_{p+1,1} & \cdots & -M_{p+1,p+1}^{-1}A_{p+1,p} & M_{p+1,p+1}^{-1}N_{p+1,p+1} \end{bmatrix}.$$

□

3.5 Conclusion

Classical matrix splittings for asynchronous iterations cannot be applied in a domain sub-structuring framework, where the interface problem is not explicitly given, in practice. For primal approaches leading to the inversion of a Schur complement, we proposed here two suitable matrix splittings, still applicable to derive classical asynchronous iterative models. We further analyzed one of these two splittings (and reasonably expect similar results for the other one), both in an additive Schwarz and a primal sub-structuring frameworks. Surprisingly, very close sufficient asynchronous convergence conditions are obtained for these two main types of domain decomposition, and moreover, by means of the same theoretical tool based on a component-wise norm under the model of asynchronous iterations with memory (see [18] or Theorem 2.10).

Methods for space domains decomposition are commonly designed, and such new results easily suggest further developments in the application of the asynchronous iterations theory. To therefore keep enlarging this application scope, we now tackle, in the subsequent discussion, the under-investigated field of decomposition methods targeting domains defined as simulated time intervals.

Chapter 4

Asynchronous time domain decomposition

4.1 Introduction

Inherently, time-parallel methods somehow constitute a kind of asynchronous time-iterations, as they intend to perform "future" iterations without necessarily following the precedence order implied by the elapsing time which is simulated. Unfortunately, successful attempts, till now, require another level of classical iterations which remains sequential. The only application of the asynchronous iterations theory to this second iterative level has been targeting Waveform relaxation methods (see, e.g., [60, 57]), which however still feature the decomposition of a space domain whereon the time-dependent problem is solved. In this chapter, we address actual time domain decomposition through the well-known Parareal method [14], for the solution of time-dependent partial differential equations (PDE). From an algebraic fixed-point formulation of the method, we successively analyze synchronous and asynchronous iterations for a theoretical comparison about both convergence conditions and algorithmic time complexity.

4.2 Problem formulation

Let Ω be a bounded domain in $\mathbb{R}^3$ with a Lipschitz boundary $\partial\Omega$, and let $[0,T]$, with $T \in \mathbb{R}$, be considered as a time domain. One would like to solve, in parallel, a time-dependent PDE

$$\left\{\begin{array}{lcll} \dfrac{\partial u}{\partial t}(x,t) + \mathcal{L}u(x,t) & = & f(x,t), & (x,t) \in \Omega \times [0,T], \\ u(x,t=0) & = & u_0(x), & x \in \Omega, \\ \mathcal{BC}(u(x,t)) & = & g(x), & (x,t) \in \partial\Omega \times [0,T], \end{array}\right. \tag{4.1}$$

where $\mathcal{L}$ is a second-order linear elliptic operator, and $\mathcal{BC}(u)$ are suitable boundary conditions. To this end, we consider a decomposition of the time domain $[0,T]$ into N time frames $[T_n, T_{n+1}]$, $n \in \{0,1,\ldots,N-1\}$, such that

$$T_n = n\Delta T, \quad \forall n \in \{0,1,\ldots,N\},$$

with $\Delta T = T/N$. The problem (4.1) reduces in each time sub-domain to

$$\left\{\begin{array}{lcll} \dfrac{\partial u_n}{\partial t}(x,t) + \mathcal{L}u_n(x,t) & = & f_n(x,t), & (x,t) \in \Omega \times [T_n, T_{n+1}], \\ u_n(x, t = T_n) & = & \lambda_n(x), & x \in \Omega, \\ \mathcal{BC}(u_n(x,t)) & = & g(x), & (x,t) \in \partial\Omega \times [T_n, T_{n+1}], \end{array}\right. \tag{4.2}$$

with $0 \leq n < N$ and $f_n = f_{|[T_n,T_{n+1}]}$. It results the problem of "predicting" a collection $\{\lambda_n\}_{n\in\{0,\ldots,N\}}$ which will satisfy

$$\lim_{\varepsilon \to 0} u_n(x, T_{n+1} - \varepsilon) = \lambda_{n+1}(x), \quad \forall n \in \{0,\ldots,N-1\}, \tag{4.3}$$

with $\varepsilon > 0$, so that subproblems (4.2) can then be solved independently.

Let us assume that the solution of (4.1) can be sufficiently approximated by using some given time and space discretization schemes with a time step δt. In the remainder, we write λ_n to denote the discrete approximation of a function $\lambda_n(x)$, with $x \in \Omega$, and Γ_n is the corresponding vector space. Then, based on this discretization, one defines a *fine propagator* F to solve (4.2) such that we have

$$\lambda^*_{n+1} = F(\lambda^*_n), \quad \forall n \in \{0,\ldots,N-1\},$$

where each λ^*_n is a sufficiently fine approximation of $u(x,T_n)$. Similarly, a *coarse propagator* G is considered, based on the time step ΔT, and which can be used to less accurately solves (4.1). We may thus set

$$w_{n+1} := G(\lambda^*_n), \quad \forall n \in \{0,\ldots,N-1\},$$

with w_{n+1} being a coarse approximation of $u(x,T_{n+1})$. The Parareal iterative scheme defines sequences $\{\lambda^k_n\}_{k\in\mathbb{N}}$ which are expected to converge to the collection $\{\lambda^*_n\}$ such that

$$\left\{\begin{array}{lcl} \lambda^0_0 & = & u_0, \\ \lambda^0_{n+1} & = & G(\lambda^0_n), \quad 0 \leq n < N, \\ \lambda^{k+1}_0 & = & \lambda^k_0, \\ \lambda^{k+1}_{n+1} & = & G(\lambda^{k+1}_n) + F(\lambda^k_n) - G(\lambda^k_n), \quad 0 \leq n < N. \end{array}\right. \tag{4.4}$$

Now let $\{P^k\}_{k\in\mathbb{N}}$ be a sequence of integer subsets, with

$$P^k \subseteq \{0,\ldots,N-1\},$$

let ρ_n and τ_n, with $0 \leq n < N$, be nonnegative integer-valued functions, with

$$\rho_n(k) \leq k, \qquad \tau_n(k) \leq k+1,$$

all satisfying :

$$\begin{cases} \lim\limits_{k\to\infty} \tau_n(k) = \lim\limits_{k\to\infty} \rho_n(k) = \infty, \\ \text{card}\{k \in \mathbb{N} \mid n \in P^k\} = \infty. \end{cases} \tag{4.5}$$

Our interest is to derive a convergent asynchronous iterative scheme given by :

$$\begin{cases} \lambda_0^0 &= u_0, \\ \lambda_{n+1}^0 &= G(\lambda_n^0), \quad 0 \leq n < N, \\ \lambda_0^{k+1} &= \lambda_0^k, \\ \lambda_{n+1}^{k+1} &= \begin{cases} G(\lambda_n^{\tau_n(k)}) + F(\lambda_n^{\rho_n(k)}) - G(\lambda_n^{\rho_n(k)}), & n \in P^k, \\ \lambda_{n+1}^k, & n \notin P^k, \end{cases} \end{cases} \tag{4.6}$$

according to which the Parareal iterative model corresponds to the particular instance where we have, for any k, $P^k = \{0, \ldots, N-1\}$, $\rho_n(k) = k$, and $\tau_n(k) = k+1$.

4.3 Parareal iterations

4.3.1 Convergence conditions

We assume, for any $\lambda_n \in \Gamma_n$:

$$\begin{aligned} G(\lambda_n) &= R\lambda_n + h, \\ F(\lambda_n) &= \widehat{r}\lambda_n + \widehat{h}, \end{aligned}$$

with

$$\widehat{r} = r^{\frac{\Delta T}{\delta t}},$$

where R (resp. r) and h (resp. $\widehat{h}$) are a matrix and a vector associated to the coarse (resp. fine) time and space discretization of the PDE. Let us define

$$\Gamma = \prod_{n=0}^{N} \Gamma_n.$$

Further, provide all Γ_n with a norm $\|.\|$, and consider the maximum norm given by:

$$\|\lambda\|_\infty = \max_{0\leq n\leq N} \|\lambda_n\|, \quad \lambda \in \Gamma,$$

which induces, on an $N \times N$ matrix $\mathcal{M}$ defined on Γ:

$$\|\mathcal{M}\|_\infty = \max_{0\leq n\leq N} \sum_{m=0}^{N} \|\mathcal{M}_{n,m}\|,$$

where $\mathcal{M}_{n,m}$ are its block-entries defined on Γ_n.

Proposition 4.1. *There exists $\theta \geq \|R\|$, with $\theta \neq 1$, such that*

$$\|\lambda_n^k - \lambda_n^*\| \leq \alpha^k e_0, \qquad \forall n \in \{0, \ldots, N\},$$

with

$$\alpha = \frac{1-\theta^N}{1-\theta}\|\widehat{r} - R\|,$$

and

$$e_0 = \max_n \left\| (R^n - \widehat{r}^n)\lambda_0^* + \sum_{i=0}^{n-1} R^i h - \widehat{r}^i \widehat{h} \right\|.$$

Proof. Let $\lambda, \widetilde{\lambda} \in \Gamma$. Parareal iterations apply a global mapping $\mathcal{T}$ such that $\widetilde{\lambda} = \mathcal{T}(\lambda)$ if

$$\left\{ \begin{array}{rcl} \widetilde{\lambda}_0 & = & \lambda_0^*, \\ \widetilde{\lambda}_n & = & R\widetilde{\lambda}_{n-1} + (\widehat{r} - R)\lambda_{n-1} + \widehat{h}, \qquad n \in \{1, \ldots, N\}, \end{array} \right.$$

which leads to :

$$\mathcal{T}(\lambda) = A^{-1}B\lambda + A^{-1}c, \tag{4.7}$$

with

$$A = \begin{bmatrix} I & O & O & \cdots & O \\ -R & I & O & \cdots & O \\ O & -R & I & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & O \\ O & \cdots & O & -R & I \end{bmatrix}, \; B = \begin{bmatrix} O & O & O & \cdots & O \\ \widehat{r} - R & O & O & \cdots & O \\ O & \widehat{r} - R & O & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & O \\ O & \cdots & O & \widehat{r} - R & O \end{bmatrix}, \; c = \begin{bmatrix} \lambda_0^* \\ \widehat{h} \\ \widehat{h} \\ \vdots \\ \widehat{h} \end{bmatrix}.$$

We can explicitly express :

$$(A^{-1}\lambda)_n = \sum_{i=0}^{n} R^i \lambda_{n-i},$$

and

$$(A^{-1}B\lambda)_n = \sum_{i=0}^{n-1} R^i (\widehat{r} - R)\lambda_{(n-1)-i},$$

so that we have

$$\begin{aligned}\|\mathcal{T}(\lambda)-\mathcal{T}(\widetilde{\lambda})\|_\infty &\leq \|A^{-1}B\|_\infty\|\lambda-\widetilde{\lambda}\|_\infty,\\ &\leq \max_n \sum_{i=0}^{n-1}\|R^i(\widehat{r}-R)\|\|\lambda-\widetilde{\lambda}\|_\infty,\\ &\leq \sum_{n=0}^{N-1}\|R^n(\widehat{r}-R)\|\|\lambda-\widetilde{\lambda}\|_\infty,\\ &\leq \sum_{n=0}^{N-1}\|R\|^n\|\widehat{r}-R\|\|\lambda-\widetilde{\lambda}\|_\infty.\end{aligned}$$

We can set $\theta = \|R\|$ if $\|R\| \neq 1$, otherwise take $\theta = \|R\| + \epsilon$, with $\epsilon > 0$, then the constant α is deduced by summing the geometric series. Now let $\{\lambda^k\}_{k\in\mathbb{N}}$ be a Parareal iterates sequence. According to the fixed-point mapping formulation, we verify

$$\lambda^k - \lambda^* = (A^{-1}B)^k(\lambda^0-\lambda^*),$$

and it follows, for all $n \in \{0,\ldots,N\}$,

$$\|\lambda_n^k - \lambda_n^*\| \leq \alpha^k\|\lambda^0-\lambda^*\|_\infty.$$

Further, we have $\lambda_0^0 = \lambda_0^*$ and, for $n > 0$,

$$\lambda_n^0 = R\lambda_{n-1}^0 + h = R^n\lambda_0^* + \sum_{i=0}^{n-1} R^i h,$$

and thus

$$\lambda_n^0 - \lambda_n^* = (R^n - \widehat{r}^n)\lambda_0^* + \sum_{i=0}^{n-1} R^i h - \widehat{r}^i\widehat{h},$$

which concludes the proof. □

Remark 4.1.

$$\lim_{N\to\infty}\frac{1-\theta^N}{1-\theta}\|\widehat{r}-R\| < 1 \quad \Longrightarrow \quad \theta < 1.$$

Corollary 4.1. *For $\theta < 1$, there exists a positive function e on $\mathbb{N}$, with*

$$\lim_{i\to\infty} e(i) = 0, \quad i \in \mathbb{N},$$

such that if

$$\|R\| + \|\widehat{r}-R\| < 1 + e(N),$$

then the computational model (4.4) *is convergent.*

Proof. It follows from Proposition 4.1 that the Parareal iterates λ_n^k converge to the fine approximations λ_n^* if

$$\begin{aligned} \frac{1-\theta^N}{1-\theta}\|\widehat{r}-R\| < 1, \qquad \theta \geq \|R\|, \quad \theta \neq 1, \\ (1-\theta^N)\|\widehat{r}-R\| + \theta < 1. \end{aligned}$$

This requires at least $\theta < 1$, and thus $\|R\| < 1$, which leads to

$$\begin{aligned} (1-\|R\|^N)\|\widehat{r}-R\| + \|R\| < 1, \\ \|\widehat{r}-R\| + \|R\| < 1 + \|R\|^N\|\widehat{r}-R\|, \end{aligned}$$

and concludes the proof. □

4.3.2 Computational efficiency

Given any standard time discretization scheme, we might consider one time step resolution as elementary operation. Then, in each time frame, the coarse propagator G requires one operation while, for the fine propagator F, one performs $\Delta T/\delta t$ operations. However, as the discretization scheme used for G can differ from that of F, the underlying elementary operations should be characterized by two different unitary complexities that we denote by $\mathcal{C}_{1,G}$ and $\mathcal{C}_{1,F}$, respectively. Let us mention that a sequential resolution of the PDE would consist of applying F over the whole time domain. The sequential time complexity is thus obviously given by

$$\mathcal{C}(1) = \frac{T}{\delta t}\mathcal{C}_{1,F}.$$

We assume in the sequel that fine step resolutions are performed in parallel by N processes.

Proposition 4.2. *The computational time induced by the model* (4.4) *for* k *iterations and* N *processes is given by :*

$$\mathcal{C}^k(N) = (k+1)\left(N - \frac{k}{2}\right)\mathcal{C}_{1,G} + k\frac{\Delta T}{\delta t}\mathcal{C}_{1,F}.$$

Proof. Any Parareal iteration i on a process n is given by :

$$\begin{aligned} \lambda_{n+1}^i &= G(\lambda_n^i) + F(\lambda_n^{i-1}) - G(\lambda_n^{i-1}), \\ &= G(\lambda_n^i) + F(\lambda_n^{i-1}) - w_n^{i-1}, \end{aligned}$$

which corresponds to the computational time

$$\mathcal{C}_{1,G} + \frac{\Delta T}{\delta t}\mathcal{C}_{1,F}.$$

On the other hand, it is trivial that

$$\forall n < i, \quad G(\lambda_n^i) = G(\lambda_n^{i-1}),$$

and therefore, at iteration i, G needs to be sequentially evaluated only over $(N-i)$ time frames. Then, after k iterations, and accounting the N initializations

$$\lambda_{n+1}^0 = G(\lambda_n^0),$$

we have

$$\begin{aligned}
\mathcal{C}^k(N) &= N\mathcal{C}_{1,G} + \sum_{i=1}^{k} (N-i)\,\mathcal{C}_{1,G} + k\frac{\Delta T}{\delta t}\mathcal{C}_{1,F}, \\
&= N\mathcal{C}_{1,G} + \left(kN - \frac{k(k+1)}{2}\right)\mathcal{C}_{1,G} + k\frac{\Delta T}{\delta t}\mathcal{C}_{1,F}, \\
&= (k+1)\left(N - \frac{k}{2}\right)\mathcal{C}_{1,G} + k\frac{\Delta T}{\delta t}\mathcal{C}_{1,F},
\end{aligned}$$

which concludes the proof. □

Corollary 4.2. *Let $\mathcal{E}^k(N)$ denote the parallel efficiency induced by the model (4.4) for k iterations and N processes. Then, at the maximum expected speedup for $k > 1$, we have*

$$\mathcal{E}^k(N) \leq \frac{1}{2}.$$

Proof. The theoretical speedup obtained after k Parareal iterations is given by

$$\mathcal{S}^k(N) = \frac{\mathcal{C}(1)}{\mathcal{C}^k(N)} = \frac{1}{(k+1)\left(N - \frac{k}{2}\right)\frac{\delta t}{T}\frac{\mathcal{C}_{1,G}}{\mathcal{C}_{1,F}} + \frac{k}{N}},$$

which reaches its maximum for

$$N = \widehat{N} := \sqrt{\frac{k}{k+1}\frac{T}{\delta t}\frac{\mathcal{C}_{1,F}}{\mathcal{C}_{1,G}}}.$$

This yields the efficiency

$$\mathcal{E}^k(\widehat{N}) = \frac{\mathcal{S}^k(\widehat{N})}{\widehat{N}} = \frac{1}{2k - k\sqrt{\frac{k(k+1)}{4}\frac{\delta t}{T}\frac{\mathcal{C}_{1,G}}{\mathcal{C}_{1,F}}}},$$

and we have, for any given T, δt, $\mathcal{C}_{1,G}$ and $\mathcal{C}_{1,F}$,

$$\mathcal{E}^{\widehat{N}}(\widehat{N}) < \mathcal{E}^{\widehat{N}-1}(\widehat{N}) < \cdots < \mathcal{E}^1(\widehat{N}).$$

Considering $\widehat{N} \geq 1$, we ensure

$$\frac{T}{\delta t} \geq \frac{k+1}{k} \frac{\mathcal{C}_{1,G}}{\mathcal{C}_{1,F}},$$

and particularly for $k = 2$, it implies

$$\mathcal{E}^2(\widehat{N}) = \frac{1}{4 - 2\sqrt{\frac{3}{2}\frac{\delta t}{T}\frac{\mathcal{C}_{1,G}}{\mathcal{C}_{1,F}}}} \leq \frac{1}{2},$$

which concludes the proof. □

Figure 4.1 (left) illustrates the speedup evolution as the number of time frames grows, for different numbers of iterations.

4.4 Parareal asynchronous iterations

4.4.1 Convergence conditions

Proposition 4.3. *The contraction condition in Theorem 2.6 is fulfilled for the computational model* (4.6).

Proof. We define the set

$$\Gamma^k = \{\lambda \in \Gamma \mid \forall n \in \{0, \ldots, \min\{k, N\}\}, \quad \lambda_n = \lambda_n^*\}, \qquad k \in \mathbb{N}.$$

Then we have

$$\Gamma^{k+1} \subset \Gamma^k \subset \cdots \subset \Gamma^0 \subset \Gamma. \tag{4.8}$$

Now let $\widetilde{\lambda}, \widehat{\lambda} \in \Gamma^k$, and let $\lambda = \widetilde{\mathcal{T}}(\widetilde{\lambda}, \widehat{\lambda})$, with $\widetilde{\mathcal{T}}$ being defined such that

$$\begin{cases} \lambda_0 = \lambda_0^*, \\ \lambda_n = G(\widetilde{\lambda}_{n-1}) + F(\widehat{\lambda}_{n-1}) - G(\widehat{\lambda}_{n-1}), \qquad 1 \leq n \leq N. \end{cases}$$

Then for all $n \in \{1, \ldots, \min\{k+1, N\}\}$,

$$\begin{aligned} \lambda_n &= G(\lambda_{n-1}^*) + F(\lambda_{n-1}^*) - G(\lambda_{n-1}^*), \\ &= \lambda_n^*, \end{aligned}$$

which implies :

$$\widetilde{\mathcal{T}}(\widetilde{\lambda}, \widehat{\lambda}) \in \Gamma^{k+1}, \qquad \forall \widetilde{\lambda}, \widehat{\lambda} \in \Gamma^k. \tag{4.9}$$

Further, let $\{\widetilde{\lambda}^k\}_{k \in \mathbb{N}}$ be a sequence such that $\widetilde{\lambda}^k \in \Gamma^k$, for any $k \in \mathbb{N}$. Then,

$$\forall k \geq N, \quad \widetilde{\lambda}^k = \lambda^*,$$

which implies, for any fixed N,

$$\lim_{k\to\infty} \widetilde{\lambda}^k = \lambda^*. \tag{4.10}$$

At last, let us define the sets

$$\Gamma_n^k = \begin{cases} \{\lambda_n^*\}, & 0 \le n \le \min\{k, N\}, \quad k \in \mathbb{N}, \\ \Gamma_n, & k+1 \le n \le N, \qquad 0 \le k \le N-1. \end{cases}$$

Then for all $n \in \{0, \ldots, N\}$ and $k \in \mathbb{N}$,

$$\begin{cases} \Gamma_n^k \subset \Gamma_n, \\ \Gamma^k = \Gamma_0^k \times \cdots \times \Gamma_N^k. \end{cases} \tag{4.11}$$

The proposition follows from (4.8)–(4.11). □

Proposition 4.4. *If there exists a matrix norm $\|.\|$ such that*

$$\|R\| + \|\widehat{r} - R\| < 1,$$

then the computational model (4.6) *is convergent.*

Proof. Let $\widetilde{\lambda}, \widehat{\lambda} \in \Gamma$. Parareal-based asynchronous iterations apply mappings given by :

$$\begin{cases} \widetilde{\mathcal{T}}_0(\widetilde{\lambda}, \widehat{\lambda}) &= \lambda_0^*, \\ \widetilde{\mathcal{T}}_n(\widetilde{\lambda}, \widehat{\lambda}) &= R\widetilde{\lambda}_{n-1} + (\widehat{r} - R)\widehat{\lambda}_{n-1} + \widehat{h}, \qquad 1 \le n \le N, \end{cases}$$

which leads to

$$\widetilde{\mathcal{T}}(\widetilde{\lambda}, \widehat{\lambda}) = A\widetilde{\lambda} + B\widehat{\lambda} + c, \tag{4.12}$$

with

$$A = \begin{bmatrix} O & O & O & \cdots & O \\ R & O & O & \cdots & O \\ O & R & O & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & O \\ O & \cdots & O & R & O \end{bmatrix}, \; B = \begin{bmatrix} O & O & O & \cdots & O \\ \widehat{r} - R & O & O & \cdots & O \\ O & \widehat{r} - R & O & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & O \\ O & \cdots & O & \widehat{r} - R & O \end{bmatrix}, \; c = \begin{bmatrix} \lambda_0^* \\ \widehat{h} \\ \widehat{h} \\ \vdots \\ \widehat{h} \end{bmatrix}.$$

Then for any $\widetilde{\lambda}$, $\widehat{\lambda}$, $\widetilde{\widetilde{\lambda}}$ and $\widehat{\widehat{\lambda}}$ in Γ,

$$\begin{aligned} \|\widetilde{\mathcal{T}}(\widetilde{\lambda}, \widehat{\lambda}) - \widetilde{\mathcal{T}}(\widetilde{\widetilde{\lambda}}, \widehat{\widehat{\lambda}})\|_\infty &= \|A(\widetilde{\lambda} - \widetilde{\widetilde{\lambda}}) + B(\widehat{\lambda} - \widehat{\widehat{\lambda}})\|_\infty, \\ &\le \|A\|_\infty \|\widetilde{\lambda} - \widetilde{\widetilde{\lambda}}\|_\infty + \|B\|_\infty \|\widehat{\lambda} - \widehat{\widehat{\lambda}}\|_\infty, \\ &\le (\|A\|_\infty + \|B\|_\infty) \max\left\{ \|\widetilde{\lambda} - \widetilde{\widetilde{\lambda}}\|_\infty, \|\widehat{\lambda} - \widehat{\widehat{\lambda}}\|_\infty \right\}, \\ &\le (\|R\| + \|\widehat{r} - R\|) \max\left\{ \|\widetilde{\lambda} - \widetilde{\widetilde{\lambda}}\|_\infty, \|\widehat{\lambda} - \widehat{\widehat{\lambda}}\|_\infty \right\}. \end{aligned}$$

If therefore

$$\|R\| + \|\widehat{r} - R\| < 1,$$

then the contraction condition in Theorem 2.11 is fulfilled, which concludes the proof. □

Proposition 4.5. *There exists a nonnegative integer-valued function σ on $\mathbb{N}$, satisfying :*

$$\lim_{k\to\infty} \sigma(k) = \infty, \tag{4.13}$$

such that

$$\|\lambda_n^k - \lambda_n^*\| \leq \alpha^{\sigma(k)} \|\lambda^0 - \lambda^*\|_\infty, \qquad \forall n \in \{0, \ldots, N\},$$

with

$$\alpha = \|R\| + \|\widehat{r} - R\|.$$

Proof. Let us define the set

$$\Gamma^k = \left\{ \lambda \in \Gamma \mid \|\lambda - \lambda^*\|_\infty \leq \alpha^{\sigma(k)} \|\lambda^0 - \lambda^*\|_\infty \right\}, \quad k \in \mathbb{N},$$

with $\sigma(0) = 0$. We obviously verify :

$$\lambda^0 \in \Gamma^0,$$

then let us assume :

$$\forall i \in \{0, \ldots, k\}, \quad \lambda^i \in \Gamma^i. \tag{4.14}$$

According to the asynchronous iterative scheme (4.6), we have

$$\begin{cases} \|\lambda_{n+1}^{k+1} - \lambda_{n+1}^*\| &= \|\lambda_{n+1}^k - \lambda_{n+1}^*\|, & n \notin P^k, \\ \|\lambda_{n+1}^{k+1} - \lambda_{n+1}^*\| &\leq \alpha \max\left\{ \|\lambda_n^{\tau_n(k)} - \lambda_n^*\|, \|\lambda_n^{\rho_n(k)} - \lambda_n^*\| \right\}, & n \in P^k. \end{cases}$$

Additionally, we may consider a sequence $\{Q^k\}_{k\in\mathbb{N}}$ of subsets of P^k such that

$$\tau_n(k) = k + 1 \quad \Longleftrightarrow \quad n \in Q^k.$$

Then, for $n \in Q^k$, we verify :

$$\begin{aligned}
\|\lambda_{n+1}^{k+1} - \lambda_{n+1}^*\| &\leq \alpha \max\left\{\|\lambda_n^{k+1} - \lambda_n^*\|, \|\lambda_n^{\rho_n(k)} - \lambda_n^*\|\right\}, \\
&\leq \alpha \max\left\{\begin{array}{l} \alpha \max\left\{\|\lambda_{n-1}^{\tau_{n-1}(k)} - \lambda_{n-1}^*\|, \|\lambda_{n-1}^{\rho_{n-1}(k)} - \lambda_{n-1}^*\|\right\}, \\ \|\lambda_n^{\rho_n(k)} - \lambda_n^*\| \end{array}\right\}, \\
&\leq \alpha \max\left\{\begin{array}{l} \alpha^2 \max\left\{\|\lambda_{n-2}^{\tau_{n-2}(k)} - \lambda_{n-2}^*\|, \|\lambda_{n-2}^{\rho_{n-2}(k)} - \lambda_{n-2}^*\|\right\}, \\ \alpha\|\lambda_{n-1}^{\rho_{n-1}(k)} - \lambda_{n-1}^*\|, \\ \|\lambda_n^{\rho_n(k)} - \lambda_n^*\| \end{array}\right\}, \\
&\leq \alpha \max\left\{\begin{array}{l} \alpha^{i_n} \max\left\{\|\lambda_{n-i_n}^{\tau_{n-i_n}(k)} - \lambda_{n-i_n}^*\|, \|\lambda_{n-i_n}^{\rho_{n-i_n}(k)} - \lambda_{n-i_n}^*\|\right\}, \\ \alpha^{i_n-1}\|\lambda_{n-(i_n-1)}^{\rho_{n-(i_n-1)}(k)} - \lambda_{n-(i_n-1)}^*\|, \\ \ldots, \\ \alpha^{i_n-i_n}\|\lambda_{n-(i_n-i_n)}^{\rho_{n-(i_n-i_n)}(k)} - \lambda_{n-(i_n-i_n)}^*\| \end{array}\right\},
\end{aligned}$$

with $i_n \in \{1, \ldots, n\}$ satisfying :

$$\left\{\begin{array}{lcll} n - i_n & \in & P^k \setminus Q^k, & \\ n - i & \in & Q^k, & \forall i \in \{0, \ldots, i_n - 1\}. \end{array}\right.$$

One should notice that, as

$$\lambda_0^{k+1} = \lambda_0^k = \lambda_0^{\tau_0(k)},$$

we can somehow consider

$$0 \in P^k \setminus Q^k,$$

and therefore i_n necessarily exists. Further, let $j_n \in \{0, \ldots, i_n\}$ satisfy:

$$\alpha^{j_n}\|\lambda_{n-j_n}^{\rho_{n-j_n}(k)} - \lambda_{n-j_n}^*\| = \max_{0 \leq j \leq i_n} \alpha^{i_n - j}\|\lambda_{n-(i_n-j)}^{\rho_{n-(i_n-j)}(k)} - \lambda_{n-(i_n-j)}^*\|,$$

and let $l_n \in \{i_n + 1, j_n + 1\}$ satisfy:

$$\alpha^{l_n} = \max\{\alpha^{i_n+1}, \alpha^{j_n+1}\}.$$

Then we can write:

$$\begin{aligned}
\|\lambda_{n+1}^{k+1} - \lambda_{n+1}^*\| &\leq \alpha \max\left\{\alpha^{i_n}\|\lambda_{n-i_n}^{\tau_{n-i_n}(k)} - \lambda_{n-i_n}^*\|, \alpha^{j_n}\|\lambda_{n-j_n}^{\rho_{n-j_n}(k)} - \lambda_{n-j_n}^*\|\right\}, \\
&\leq \alpha^{l_n} \max\left\{\|\lambda_{n-i_n}^{\tau_{n-i_n}(k)} - \lambda_{n-i_n}^*\|, \|\lambda_{n-j_n}^{\rho_{n-j_n}(k)} - \lambda_{n-j_n}^*\|\right\}.
\end{aligned}$$

It follows:

$$\|\lambda^{k+1}-\lambda^*\|_\infty \leq \max\left\{\begin{array}{l}\max\limits_{n\notin P^k}\|\lambda_{n+1}^k-\lambda_{n+1}^*\|,\\ \max\limits_{n\in P^k\setminus Q^k}\alpha\max\left\{\|\lambda_n^{\tau_n(k)}-\lambda_n^*\|,\|\lambda_n^{\rho_n(k)}-\lambda_n^*\|\right\},\\ \max\limits_{n\in Q^k}\alpha^{l_n}\max\left\{\|\lambda_{n-i_n}^{\tau_{n-i_n}(k)}-\lambda_{n-i_n}^*\|,\|\lambda_{n-j_n}^{\rho_{n-j_n}(k)}-\lambda_{n-j_n}^*\|\right\}\end{array}\right\},$$

and applying (4.14), we obtain

$$\|\lambda^{k+1}-\lambda^*\|_\infty \leq \max\left\{\begin{array}{l}\alpha^{\sigma(k)},\\ \max\limits_{n\in P^k\setminus Q^k}\max\left\{\begin{array}{l}\alpha^{\sigma(\tau_n(k))+1},\\ \alpha^{\sigma(\rho_n(k))+1}\end{array}\right\},\\ \max\limits_{n\in Q^k}\max\left\{\begin{array}{l}\alpha^{\sigma(\tau_{n-i_n}(k))+l_n},\\ \alpha^{\sigma(\rho_{n-j_n}(k))+l_n}\end{array}\right\}\end{array}\right\}\|\lambda^0-\lambda^*\|_\infty.$$

We can therefore have

$$\|\lambda^{k+1}-\lambda^*\|_\infty \leq \alpha^{\sigma(k+1)}\|\lambda^0-\lambda^*\|_\infty,$$

with $\sigma(k+1)$ satisfying the appropriate relation by analogy with the previous inequality. The classical assumption (4.5) on P^k, τ and ρ implies that σ also satisfies (4.13), which concludes the proof. □

4.4.2 Computational efficiency

Let

$$\kappa_n(k) := \mathrm{card}\{i \leq k \mid n \in P^i\}$$

denote the number of iterations performed by the process n. Let then

$$\kappa(k) := \max_{0\leq n<N}\kappa_n(k)$$

be the maximum number of times the propagator F is executed.

Proposition 4.6. *The computational time induced by the model* (4.6) *for k iterations and N processes is given by :*

$$\mathcal{C}^k(N) = (\kappa(k)+N)\,\mathcal{C}_{1,G} + \kappa(k)\frac{\Delta T}{\delta t}\mathcal{C}_{1,F}.$$

Proof. Without loss of generality, we assume for an iteration k that

$$\rho_n(k) \leq \tau_n(k).$$

In a practical context, it is then reasonable to also assume that

$$\exists k_0 \leq k : \begin{cases} n \in P^{k_0} \\ \lambda_n^{\rho_n(k)} = \lambda_n^{\tau_n(k_0)}, \end{cases}$$

which, just as in classical Parareal iterations, prevents the propagator G from being evaluated twice. The complexity of a single iteration performed by each process is thus here also given by:

$$\mathcal{C}_{1,G} + \frac{\Delta T}{\delta t}\mathcal{C}_{1,F}.$$

We derive that after k global asynchronous iterations, and accounting initialization, we have, for each process n,

$$\mathcal{C}^{k,n}(1) = \mathcal{C}_{1,G} + \kappa_n(k)\left(\mathcal{C}_{1,G} + \frac{\Delta T}{\delta t}\mathcal{C}_{1,F}\right).$$

Finally, as G is sequentially evaluated only for initialization, we obtain the overall complexity

$$\mathcal{C}^k(N) = N\mathcal{C}_{1,G} + \max_n \kappa_n(k)\left(\mathcal{C}_{1,G} + \frac{\Delta T}{\delta t}\mathcal{C}_{1,F}\right),$$

which concludes the proof. □

Corollary 4.3. *Let k_s and k_a denote numbers of Parareal synchronous and asynchronous iterations, respectively. Let also $\mathcal{C}_s$ and $\mathcal{C}_a$ denote the corresponding time complexities. Then, assuming that*

$$\kappa(k_a) \geq k_s,$$

we have

$$\mathcal{C}_s^{k_s}(N) - \mathcal{C}_a^{k_a}(N) \leq \frac{1}{2}(N-1)(N-2)\mathcal{C}_{1,G}.$$

Proof. We have

$$\mathcal{C}_s^{k_s}(N) - \mathcal{C}_a^{k_a}(N) = \left(\left(N - \frac{k_s+1}{2}\right)k_s - \kappa(k_a)\right)\mathcal{C}_{1,G} + (k_s - \kappa(k_a))\frac{\Delta T}{\delta t}\mathcal{C}_{1,F}.$$

Under the assumption

$$\kappa(k_a) \geq k_s,$$

the maximum gain is naturally expected for

$$\kappa(k_a) = k_s.$$

It follows that

$$\mathcal{C}_s^{k_s}(N) - \mathcal{C}_a^{k_a}(N) \leq \left(N - 1 - \frac{k_s+1}{2}\right)k_s\mathcal{C}_{1,G},$$

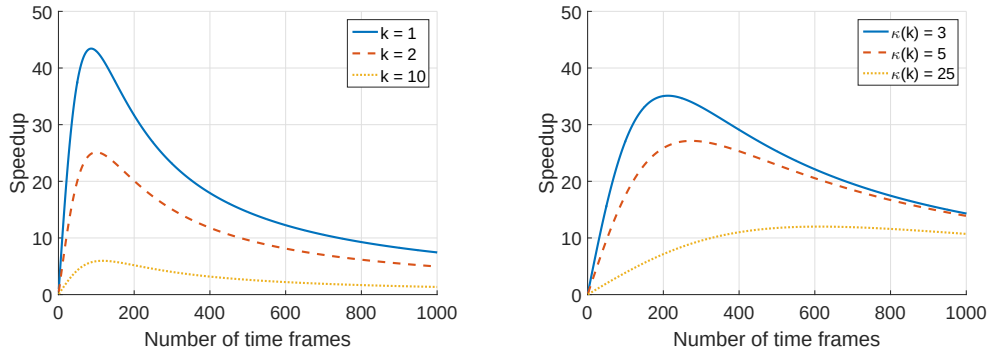


Figure 4.1: Theoretical speedup of Parareal synchronous (left) and asynchronous (right) iterations, for $T = 30$, $\delta t = 0.002$ and $\mathcal{C}_{1,G} = \mathcal{C}_{1,F}$.

which reaches its maximum for

$$k_s = N - \frac{3}{2},$$

and thus for

$$k_s = \widehat{k}_s := N - 2.$$

Replacing k_s by $\widehat{k}_s$ in the inequality leads to the upper bound result. □

Figure 4.1 (right) illustrates the speedup evolution as the number of time frames grows, for different maximum numbers of iterations.

4.4.3 Coupling parallel-in-time methods

Let us still consider the time domain decomposition (4.2), for which we assume here two compatible parallel-in-time methods, i.e., respectively based on some given operators

$$\mathcal{P} : \Gamma \to \Gamma, \qquad \mathcal{Q} : \Gamma \to \Gamma.$$

Consider then two asynchronous parallel-in-time methods, respectively based on operators

$$\widetilde{\mathcal{P}} : \Gamma \times \Gamma \to \Gamma, \qquad \widehat{\mathcal{Q}} : \Gamma \times \Gamma \to \Gamma,$$

defined such that

$$\widetilde{\mathcal{P}}(\lambda^*, \lambda^*) = \mathcal{P}(\lambda^*), \qquad \widehat{\mathcal{Q}}(\lambda^*, \lambda^*) = \mathcal{Q}(\lambda^*).$$

Combining such two operators, we can derive an asynchronous parallel-in-time scheme of the form

$$\left\{ \begin{array}{rcl} \widetilde{\lambda}_{n+1}^{k+1} & = & \left\{ \begin{array}{ll} \widetilde{\mathcal{P}}_{n+1}(\widetilde{\lambda}^{\tau(k)}, \widehat{\lambda}^{\rho(k)}), & n \in P_{\widetilde{\mathcal{P}}}^k, \\ \widetilde{\lambda}_{n+1}^k, & n \notin P_{\widetilde{\mathcal{P}}}^k, \end{array} \right. \\ \widehat{\lambda}_{n+1}^{k+1} & = & \left\{ \begin{array}{ll} \widehat{\mathcal{Q}}_{n+1}(\widetilde{\lambda}^{\tau(k)}, \widehat{\lambda}^{\rho(k)}), & n \in P_{\widehat{\mathcal{Q}}}^k, \\ \widehat{\lambda}_{n+1}^k, & n \notin P_{\widehat{\mathcal{Q}}}^k, \end{array} \right. \end{array} \right. \tag{4.15}$$

with simplified notations

$$\widetilde{\lambda}^{\tau(k)} := \begin{bmatrix} \widetilde{\lambda}_0^{\tau_0(k)} & \cdots & \widetilde{\lambda}_N^{\tau_N(k)} \end{bmatrix}^{\mathsf{T}}, \qquad \widehat{\lambda}^{\rho(k)} := \begin{bmatrix} \widehat{\lambda}_0^{\rho_0(k)} & \cdots & \widehat{\lambda}_N^{\rho_N(k)} \end{bmatrix}^{\mathsf{T}}.$$

One notices that the Parareal-based asynchronous method corresponds to the particular case where

$$\widetilde{\mathcal{P}} = \widehat{\mathcal{Q}} = \widetilde{\mathcal{T}},$$

with $\widetilde{\mathcal{T}}$ being the Parareal-based global iterative operator.

Proposition 4.7. *If there exists a vector $w > 0$ and reals $\alpha_1, \alpha_2 \in [0,1)$ such that, for any $\widetilde{\lambda}$, $\widetilde{\widetilde{\lambda}}$, $\widehat{\lambda}$ and $\widehat{\widehat{\lambda}}$ in Γ, we have :*

$$\left\{ \begin{array}{rcl} \|\widetilde{\mathcal{P}}(\widetilde{\lambda},\widehat{\lambda}) - \widetilde{\mathcal{P}}(\widetilde{\widetilde{\lambda}},\widehat{\widehat{\lambda}})\|_\infty^w & \leq & \alpha_1 \max\left\{ \|\widetilde{\lambda} - \widetilde{\widetilde{\lambda}}\|_\infty^w, \|\widehat{\lambda} - \widehat{\widehat{\lambda}}\|_\infty^w \right\}, \\ \|\widehat{\mathcal{Q}}(\widetilde{\lambda},\widehat{\lambda}) - \widehat{\mathcal{Q}}(\widetilde{\widetilde{\lambda}},\widehat{\widehat{\lambda}})\|_\infty^w & \leq & \alpha_2 \max\left\{ \|\widetilde{\lambda} - \widetilde{\widetilde{\lambda}}\|_\infty^w, \|\widehat{\lambda} - \widehat{\widehat{\lambda}}\|_\infty^w \right\}, \end{array} \right. \tag{4.16}$$

then the computational model (4.15) *is convergent.*

Proof. Let us consider the operator

$$\begin{array}{rccc} \mathcal{H}: & \Gamma \times \Gamma & \to & \Gamma \times \Gamma \\ & (\widetilde{\lambda},\widehat{\lambda}) & \mapsto & (\widetilde{\mathcal{P}}(\widetilde{\lambda},\widehat{\lambda}), \widehat{\mathcal{Q}}(\widetilde{\lambda},\widehat{\lambda})) \end{array}.$$

The iterative scheme (4.15) can then be written in a more general form :

$$\gamma_i^{k+1} = \left\{ \begin{array}{ll} \mathcal{H}_i(\gamma_0^{\sigma_0(k)}, \ldots, \gamma_{2N+1}^{\sigma_{2N+1}(k)}), & i \in P^k, \\ \gamma_i^k, & i \notin P^k, \end{array} \right.$$

with

$$\mathcal{H}_i = \left\{ \begin{array}{ll} \widetilde{\mathcal{P}}_i, & 0 \leq i \leq N, \\ \widehat{\mathcal{Q}}_{(i-1)-N}, & N < i \leq 2N+1 \end{array} \right., \qquad \sigma_i = \left\{ \begin{array}{ll} \tau_i, & 0 \leq i \leq N, \\ \rho_{(i-1)-N}, & N < i \leq 2N+1 \end{array} \right..$$

We thus have, for any $\widetilde{\lambda}$, $\widetilde{\widetilde{\lambda}}$, $\widehat{\lambda}$ and $\widehat{\widehat{\lambda}}$ in Γ,

$$\|\mathcal{H}(\widetilde{\lambda},\widehat{\lambda}) - \mathcal{H}(\widetilde{\widetilde{\lambda}},\widehat{\widehat{\lambda}})\|_\infty^z = \max\left\{ \|\widetilde{\mathcal{P}}(\widetilde{\lambda},\widehat{\lambda}) - \widetilde{\mathcal{P}}(\widetilde{\widetilde{\lambda}},\widehat{\widehat{\lambda}})\|_\infty^{z_1}, \|\widehat{\mathcal{Q}}(\widetilde{\lambda},\widehat{\lambda}) - \widehat{\mathcal{Q}}(\widetilde{\widetilde{\lambda}},\widehat{\widehat{\lambda}})\|_\infty^{z_2} \right\},$$

with

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}, \qquad z_1, z_2 \in \Gamma.$$

Applying (4.16), and by choosing

$$z_1 = z_2 = w, \qquad \beta = \max\{\alpha_1, \alpha_2\},$$

we obtain :

$$\begin{aligned}\|\mathcal{H}(\widetilde{\lambda},\widehat{\lambda}) - \mathcal{H}(\widetilde{\widetilde{\lambda}},\widehat{\widehat{\lambda}})\|_\infty^z &\leq \beta \max\left\{\|\widetilde{\lambda} - \widetilde{\widetilde{\lambda}}\|_\infty^{z_1}, \|\widehat{\lambda} - \widehat{\widehat{\lambda}}\|_\infty^{z_2}\right\}, \\ &\leq \beta\ \|(\widetilde{\lambda},\widehat{\lambda}) - (\widetilde{\widetilde{\lambda}},\widehat{\widehat{\lambda}})\|_\infty^z,\end{aligned}$$

which is a sufficient condition for convergence according to Theorem 2.5. □

4.5 Conclusion

The Parareal method can be seen as a dual approach for time domain sub-structuring, where interface transmission conditions are defined on initial values of time intervals, such that continuity is eventually ensured at convergence. We first extended, to PDEs case, within an algebraic fixed-point framework, sufficient convergence conditions previously established for ordinary differential equations (ODE). Then, under a model of asynchronous iterations with memory, we obtained sufficient asynchronous convergence conditions by means of the corresponding weighted maximum norms analysis tool (see [19] or Theorem 2.11). Surprisingly, it turned out that these sufficient synchronous and asynchronous convergence conditions asymptotically coincide as the number of time sub-domains increases. From this norm-based contraction property, we finally derived the asynchronous convergence of a slightly more general model allowing the coupling of two time-parallel methods. Still, the Parareal method features a speedup limit not only related to inter-process communication but also to a sequential propagation of some computed values. Therefore, even if a straightforward application of asynchronous iterations, as we considered here, improved this speedup limit, further developments are needed for achieving a completely unbounded efficiency.

Summarily, the first part of this dissertation has mainly consisted in designing new asynchronous iterative methods within some particular domain decomposition frameworks. We targeted a primal sub-structuring approach for space domains, while showing similarities with the overlapping Schwarz approach. A dual approach has just been addressed as well for the asynchronous decomposition of time domains. In the second coming part, we are rather interested in general implementation issues related to asynchronous iterative models, which includes

- distributed algorithms for detecting the convergence state of an ongoing asynchronous computation,
- and programming patterns for efficiently handling underlying communication middleware.

Effective numerical experiments are thus successfully conducted, which gives a practical insight into various methods, algorithms and implementation options.

Chapter 5

Termination of asynchronous iterations

5.1 Introduction

The problem of terminating asynchronous iterations was well discussed in, e.g., [33], where the authors introduced a first approach which consists of 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., [68, 69, 70, 71]). Indeed, these termination protocols are designed for parallel applications that are executed in a finite number of steps, i.e., there is a moment during their execution from where all processes are idle. Since this is not natively the case for asynchronous iterations, different modifications have been proposed (see, e.g., [35, 34, 32]) for detecting their convergence by means of a classical distributed termination protocol. Basically, any process under some local condition (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.

A kind of non-intrusive slight alteration has been discussed in [36], which consists of synchronizing the processes after predicted numbers of iterations, which are expected for reaching convergence. Here, even if the convergence test is thus blocking, it is expected to occur very few times, depending however on the accuracy of the heuristics, which in turn is related to the properties of both the solver and the execution platform.

A second approach uses a supervisory algorithm to take a partial 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 [46], the main disadvantage of such a protocol is the first-in-first-out (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 [72] for an introductory overview). Such approaches thus turn out to be quite intrusive and, furthermore, not easy to implement. In [35], some snapshot-based supervised termination protocols, more or less centralized, were designed over either star or tree network topology, without FIFO requirements. The less centralized approach therein involves a spanning tree over the communication graph where local convergence notifications propagate from the leaves to the root process. This one then triggers the partial 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 data into the snapshot messages, which introduces a possibly non-negligible overhead costs for communication.

A third approach in [73] consists of a supervised termination method based on a leader election protocol (see, e.g., [74, Section 4.4.3]) over tree network topology, 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 [37], 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 error. Moreover, just as in [35], it also features a first gathering phase through the leader election, which actually acts as a dynamically centralized coordination.

The reliability of [37] could be guaranteed by introducing the formal analysis from [48] where the convergence tests are based on the diameter of the nested sets from Theorem 2.6, which are built by means of identifying macro-iterations defined as minimal sets of iterations within which all of the solution vector components have been updated at least once. Still, such a method would also require at least intrusive piggybacking techniques, and possible other practical issues could need to be further discussed.

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 reduction phases, one for coordination and another one for convergence state evaluation. In very large distributed systems, such reduction operations would constitute the most costly part of these convergence detection protocols. Moreover, the only totally non-intrusive exact solution

(from [35]), in non-FIFO environments, embeds computation data into supervisory messages, which could also results in non-negligible communication delays. We investigate here new methods, mostly non-intrusive, to exactly evaluate the convergence residual error of a computation during asynchronous iterations, using only one reduction operation. More, non-FIFO environments are successfully managed without piggybacking techniques and overhead communication costs.

5.2 Building a distributed global vector

5.2.1 Problem formulation

Let us consider a sequence $\{x^k\}_{k\in\mathbb{N}}$ of vectors satisfying the asynchronous iterations model (2.9), with arbitrary mappings (2.12). Let us define p other sequences $\{y^{1,k}\}_{k\in\mathbb{N}}, \ldots, \{y^{p,k}\}_{k\in\mathbb{N}}$ such that

$$y^{i,k} = \left[x_1^{\tau_1^i(k)} \quad \cdots \quad x_p^{\tau_p^i(k)}\right]^{\mathsf{T}}, \qquad 1 \leq i \leq p. \tag{5.1}$$

Additionally, we assume to have

$$\tau_i^i(k) = k. \tag{5.2}$$

Let then $\bar{x}$ be given by:

$$\bar{x} = \left[y_1^{1,k_1} \quad \cdots \quad y_p^{p,k_p}\right]^{\mathsf{T}}, \qquad k_1, \ldots, k_p \in \mathbb{N}.$$

We address here the problem of evaluating a relation

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

for which we shall mainly pay attention to the computation of $f(\bar{x})$. Note that in the particular case of synchronous iterations, one can take

$$\bar{x} = \left[y_1^{1,k} \quad \cdots \quad y_p^{p,k}\right]^{\mathsf{T}}, \qquad k \in \mathbb{N},$$

and obtain, for all $i \in \{1, \ldots, p\}$:

$$\begin{aligned} f_i(\bar{x}) &= f_i\left(y_1^{1,k}, \ldots, y_p^{p,k}\right), \\ &= f_i\left(x_1^k, \ldots, x_p^k\right), \\ &= f_i\left(x_1^{\tau_1^i(k)}, \ldots, x_p^{\tau_p^i(k)}\right), \\ &= x_i^{k+1}, \end{aligned}$$

which implicitly gives:

$$f(\bar{x}) = \left[y_1^{1,k+1} \quad \cdots \quad y_p^{p,k+1}\right]^{\mathsf{T}}.$$

5.2.2 The Chandy–Lamport snapshot

The basic idea within the Chandy–Lamport snapshot (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 the neighboring processes. Algorithm 1 outlines the rules which fully describe the protocol. To give an intuitive understanding of the

Algorithm 1 CLS protocol (Chandy and Lamport, 1985)

```
1: if initiator then
2:    if state not recorded then
3:       Record state
4:       Send a marker to each neighbor in the communication graph
5:    end if
6: end if
7: if marker received then
8:    if state not recorded then
9:       Record state
10:      Send a marker to each neighbor in the communication graph
11:   end if
12:   if marker received from each neighbor then
13:      Return state and state of each reception channel
14:   end if
15: end if
16: if computation message received then
17:   if state recorded and marker not received from the sender then
18:      Add the message to the state of the corresponding reception channel
19:   end if
20: end if
```

consistency of the global state built by this snapshot protocol, we show in Figure 5.1 a simple example involving two processes, denoted by $p1$ and $p2$. Let us consider events consisting of sending or receiving a message. In this example, the process $p1$ records its local state after the event $e1$ and sends a marker (dotted arrow) to the process $p2$. On reception of the marker, the process $p2$ records its local state after the event $e4$, then records the state of its reception channel as an empty set, and finally, sends the marker back to the process $p1$. Before receiving the marker from the process $p2$, the process $p1$ received a computation message from $p2$ as event $e5$.

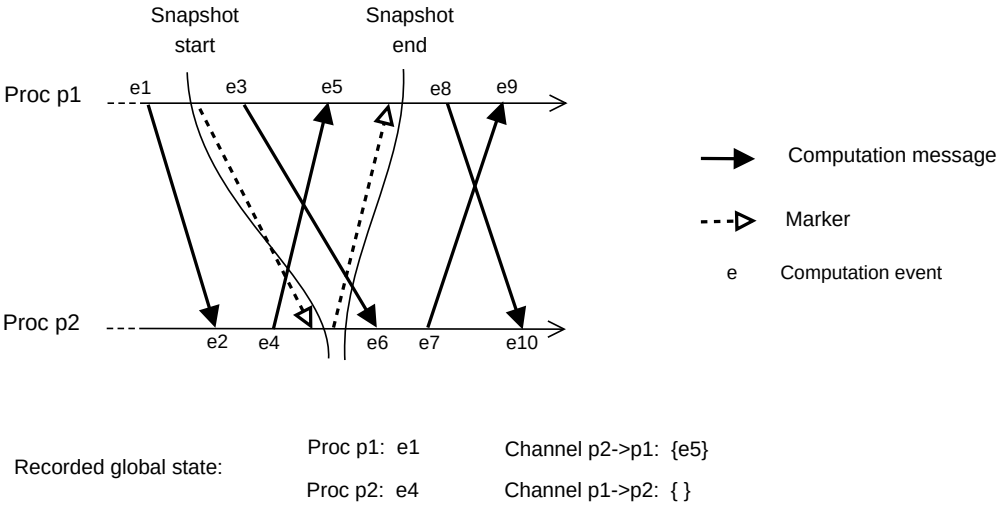


Figure 5.1: Example of a CLS protocol execution with two processes.

Therefore, the state of the reception channel of the process $p1$ corresponds to the set $\{e5\}$. It is clear from this example that the communication channels need to be FIFO. Otherwise, if the marker sent by the process $p2$ had been received by the process $p1$ before the event $e5$ occurred, the state of the channel would have been an empty set, which would cause 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 have been 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 5.1 (Chandy and 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 [46]. □

Consider now again sequences $\{x^k\}_{k\in\mathbb{N}}$ and $\{y^{i,k}\}_{k\in\mathbb{N}}$, with $1 \leq i \leq p$, from (5.1). Let us suppose an associated parallel computation involving p processes. Algorithm 2 therefore describes an application of the CLS protocol to asynchronous iterations, where $\bar{y}_i^i$ denotes the recorded state of process i, and $\{\bar{y}_j^{i,1}, \ldots, \bar{y}_j^{i,l_{i,j}}\}$, with $j \neq i$ and $l_{i,j} \in \mathbb{N}$, the recorded state of its reception channel associated to the process j.

Algorithm 2 CLS protocol on asynchronous iterative processes

if initiator **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 **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
  if marker received from each neighbor **then**
    return $\bar{y}_i^i$, $\{\bar{y}_j^{i,1}, \ldots, \bar{y}_j^{i,l_{i,j}}\}_{j \neq i}$
  end if
end if
if computation message $y_j^{i,k}$ received **then**
  if $\bar{y}_i^i$ defined and marker not received from j **then**
    $\bar{y}_j^{i,l_{i,j}} := y_j^{i,k}$
    $l_{i,j} := l_{i,j} + 1$
  end if
end if

5.2.3 Partial extension to asynchronous iterations

Instead of Algorithm 2, let each process i assemble a vector $\bar{y}^i$ by following the asynchronous iterations snapshot (AIS) rules given by either Algorithm 3 or Algorithm 4, where the criterion

$$\|y_i^{i,k} - y_i^{i,k_0^i}\| < \varepsilon, \qquad y_i^{i,k} = y_i^{i,k_0^i+1}, \quad i \in P^{k_0^i}$$

describes a local condition often considered as indicating local convergence. As the rules conditions may be fulfilled at different iterations, we should have

$$\bar{y}^i = \left[y_1^{i,k_{i,1}} \quad \cdots \quad y_p^{i,k_{i,p}} \right]^{\mathsf{T}}, \qquad k_{i,j} \in \mathbb{N}, \quad j \in \{1, \ldots, p\}.$$

Here, contrarily to the CLS protocol, there is no rule for channel record at computation message reception, and in Algorithm 4, recording the local state is not required

Algorithm 3 AIS protocol 1

1: **if** $\|y_i^{i,k} - y_i^{i,k_0^i}\| < \varepsilon$, with $y_i^{i,k} = y_i^{i,k_0^i+1}$, $i \in P^{k_0^i}$ **then**
2: **if** $\bar{y}_i^i$ undefined **then**
3: $\bar{y}_i^i := y_i^{i,k}$
4: **for all** process $j \neq i$ **do**
5: Send a marker to j
6: **end for**
7: **end if**
8: **end if**
9: **if** marker received from a process $j \neq i$ **then**
10: $\bar{y}_j^i := y_j^{i,k}$
11: **if** $\bar{y}_i^i$ undefined **then**
12: $\bar{y}_i^i := y_i^{i,k}$
13: **for all** process $j \neq i$ **do**
14: Send a marker to j
15: **end for**
16: **end if**
17: **if** $\bar{y}_j^i$ defined for all j **then**
18: **return** $\bar{y}^i$
19: **end if**
20: **end if**

Algorithm 4 AIS protocol 2

1: **if** $\|y_i^{i,k} - y_i^{i,k_0^i}\| < \varepsilon$, with $y_i^{i,k} = y_i^{i,k_0^i+1}$, $i \in P^{k_0^i}$ **then**
2: **if** $\bar{y}_i^i$ undefined **then**
3: $\bar{y}_i^i := y_i^{i,k}$
4: **for all** process $j \neq i$ **do**
5: Send a marker to j
6: **end for**
7: **end if**
8: **end if**
9: **if** marker received from a process $j \neq i$ **then**
10: $\bar{y}_j^i := y_j^{i,k}$
11: **end if**
12: **if** $\bar{y}_j^i$ defined for all j **then**
13: **return** $\bar{y}^i$
14: **end if**

at the first marker reception. We however still need the following preliminary assumptions.

Assumption 5.1. *Each process performs at least one iteration, which means:* $\forall i \in \{1, \ldots, p\}, \exists\ k < k_{i,i} : i \in P^k$.

Assumption 5.2. *After computation of* $y_i^{i,k+1}$ *(i.e.,* $i \in P^k$*),* $y_i^{i,k+1}$ *is sent to each process* $j \neq i$*, before any other communication toward* j.

Assumption 5.3. *Communication channels are FIFO.*

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

Proposition 5.1. *Under Assumption 5.1 to Assumption 5.3, Algorithm 3 and Algorithm 4 satisfy:*

$$\bar{y}^1 = \bar{y}^2 = \cdots = \bar{y}^p. \tag{5.3}$$

Proof. Let $i, j \in \{1, \ldots, p\}$ be two any process identifiers. According to the local state recording rule and Assumption 5.1,

$$\exists\ k_0^i < k_{i,i}, \quad i \in P^{k_0^i} : \quad \forall k \in \{k_0^i + 1, \ldots, k_{i,i} - 1\}, \quad i \notin P^k.$$

We thus have

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

With Assumption 5.2 and Assumption 5.3, it follows that

$$\exists\ k_0^j \leq k_{j,i} : \quad y_i^{j,k_0^j} = y_i^{i,k_0^i+1}.$$

Assumption 5.3 implies :

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

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

$$\bar{y}_i^i = \bar{y}_i^j, \qquad \forall i, j \in \{1, \ldots, p\}, \tag{5.4}$$

which concludes the proof. □

It follows from this result that by taking

$$\bar{x} = \begin{bmatrix} \bar{y}_1^1 & \cdots & \bar{y}_p^p \end{bmatrix}^\mathsf{T},$$

we explicitly obtain

$$f(\bar{x}) = \begin{bmatrix} f_1(\bar{y}^1) & \cdots & f_p(\bar{y}^p) \end{bmatrix}^\mathsf{T}.$$

Assumption 5.1 and Assumption 5.2 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 5.3 thus is the sole actual constraint in the above protocols. The next section discusses about taking off such a requirement.

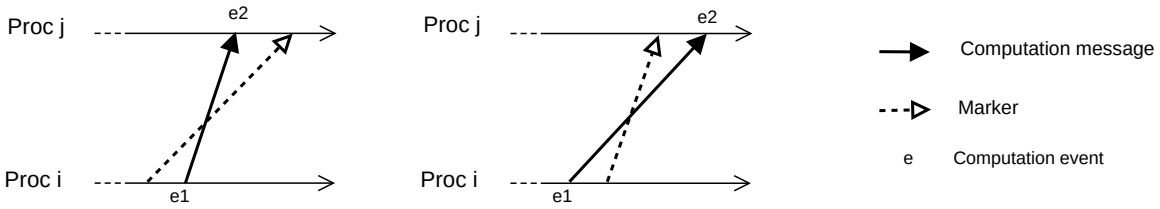


Figure 5.2: Non-FIFO snapshot issues.

5.3 Non-FIFO asynchronous iterations snapshots

5.3.1 Arbitrary non-FIFO communication

The FIFO condition is essential to avoid the two situations depicted in Figure 5.2, where a marker (dotted arrow) crosses a computation message. To still satisfy (5.4) in such non-FIFO (NF) cases, one could apply either Algorithm 5 or Algorithm 6, based on ideas from [35]. Markers therein contain computation data, so that these

Algorithm 5 NFAIS protocol 1

1: **if** $\|y_i^{i,k} - y_i^{i,k_0^i}\| < \varepsilon$, with $y_i^{i,k} = y_i^{i,k_0^i+1}$, $i \in P^{k_0^i}$ **then**
2: **if** $\bar{y}_i^i$ undefined **then**
3: $\bar{y}_i^i := y_i^{i,k}$
4: **for all** process $j \neq i$ **do**
5: Send a marker $\bar{y}_i^i$ to j
6: **end for**
7: **end if**
8: **end if**
9: **if** marker $\bar{y}_j^j$ received from a process $j \neq i$ **then**
10: $\bar{y}_j^i := \bar{y}_j^j$
11: **if** $\bar{y}_i^i$ undefined **then**
12: $\bar{y}_i^i := y_i^{i,k}$
13: **for all** process $j \neq i$ **do**
14: Send a marker $\bar{y}_i^i$ to j
15: **end for**
16: **end if**
17: **if** $\bar{y}_j^i$ defined for all j **then**
18: **return** $\bar{y}^i$
19: **end if**
20: **end if**

solutions actually even handle crossed computation messages. It obviously follows that (5.4) is thus satisfied without any of the assumptions in Proposition 5.1.

Algorithm 6 NFAIS protocol 2

1: **if** $\|y_i^{i,k} - y_i^{i,k_0^i}\| < \varepsilon$, with $y_i^{i,k} = y_i^{i,k_0^i+1}$, $i \in P^{k_0^i}$ **then**
2: **if** $\bar{y}_i^i$ undefined **then**
3: $\bar{y}_i^i := y_i^{i,k}$
4: **for all** process $j \neq i$ **do**
5: Send a marker $\bar{y}_i^i$ to j
6: **end for**
7: **end if**
8: **end if**
9: **if** marker $\bar{y}_j^j$ received from a process $j \neq i$ **then**
10: $\bar{y}_j^i := \bar{y}_j^j$
11: **end if**
12: **if** $\bar{y}_j^i$ defined for all j **then**
13: **return** $\bar{y}^i$
14: **end if**

The decentralized protocol from [35] additionally suggests a coordination phase designed upon tree network topology, prior to the actual partial snapshot phase. In the coordination phase, local convergence is notified from the leaves to the root of the tree, as described by Algorithm 7. When local convergence happens on a leaf process (having no children in the communication tree), a notification marker is sent to its father in the tree. Internal processes (having children and father) do the same if, additionally, all of their children have notified local convergence. Under same two conditions, the root process instead triggers the partial snapshot phase by recording its current local component $y_i^{i,k}$ and sending it to its neighbor processes in the initial communication graph. Rules given by Algorithm 8 are then followed to isolate the vector $\bar{y}^i$ in each process i. Here as well, local convergence must be once more verified by non-root processes to make them record their local component, but additionally, only if they had already received the recorded component of at least one of their neighbor processes. This necessarily happens, as the root process sends its recorded component to initiate the partial snapshot phase.

5.3.2 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 scenario, as being a minimum requirement for expecting efficient parallel implementation of iterative methods. Still, the problem of markers crossing computation messages remains unavoidable. Algorithm 9 proposes a partial snapshot solution which, outright, do not need marker exchange, and is based on only computation messages, even without piggybacking of control information. Here, just as local solution buffers, each process i maintains

Algorithm 7 Coordination phase of SB96 protocol (Savari and Bertsekas, 1996)

if $\|y_i^{i,k} - y_i^{i,k_0^i}\| < \varepsilon$, with $y_i^{i,k} = y_i^{i,k_0^i+1}$, $i \in P^{k_0^i}$ **then**
  if leaf process **then**
    Send a marker to father
    return
  end if
  if internal process **then**
    if marker received from all children **then**
      Send a marker to father
      return
    end if
  end if
  if root process **then**
    if marker received from all children **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
      return
    end if
  end if
end if

Algorithm 8 Snapshot phase of SB96 protocol (Savari and Bertsekas, 1996)

if $\|y_i^{i,k} - y_i^{i,k_0^i}\| < \varepsilon$, with $y_i^{i,k} = y_i^{i,k_0^i+1}$, $i \in P^{k_0^i}$ **then**
  if $\bar{y}_i^i$ undefined **and** $\bar{y}_j^i$ defined for any $j \neq i$ **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

Algorithm 9 NFAIS protocol 3

if $\|y_i^{i,k} - y_i^{i,k_i^i}\| < \varepsilon$, with $y_i^{i,k} = y_i^{i,k_i^i+1}$, $i \in P^{k_i^i}$ **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 $\tau_j^i(k_j^i) = k_j^j$, $\tau_j^i(k) = k_j^j + 1$ **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

access to the two latest received messages from 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 5.1 becomes the following:

Proposition 5.2. *Algorithm 9 satisfies:*

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

Proof. Let $i, j \in \{1, \ldots, p\}$ be two any process identifiers. Remind that

$$\bar{y}_j^i = y_j^{i,k_{i,j}}, \qquad k_{i,j} \in \mathbb{N}.$$

Then according to (5.1) and (5.2), we have

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

By construction, we satisfy:

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

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. □

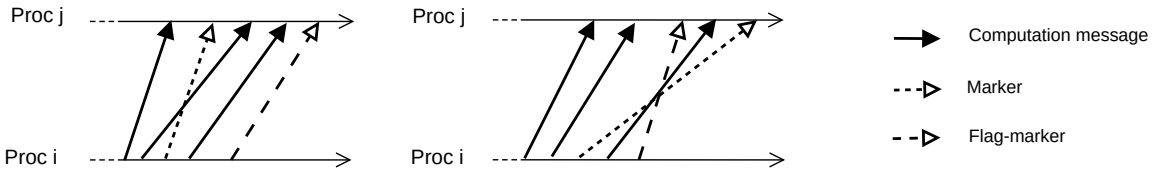


Figure 5.3: Examples of issues handled by the non-FIFO AIS protocol 4.

5.3.3 General communication model

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 from 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 therefore reasonably make here a preliminary assumption.

Assumption 5.4. *A message can cross at most η other messages.*

The previous case of arbitrary non-FIFO communication thus actually consists of taking η arbitrarily large. Let us then consider Algorithm 10. 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 $i \in P^{k_l^i}$ 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 Figure 5.3) 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, the corresponding records are discarded. The algorithm works even in the case depicted in Figure 5.3 (right) where the flag-marker crosses the classical one.

One may additionally 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, flag-markers would not be necessary any more, which rather simplifies the protocol and provides Algorithm 11.

Algorithm 10 NFAIS protocol 4

1: **if** $\|y_i^{i,t+1} - y_i^{i,t}\| < \varepsilon, \forall t \in \{k_0^i, \ldots, k-1\} : i \in P^t$ **then**
2: **if** $\bar{y}_i^i$ undefined **then**
3: $\bar{y}_i^i := y_i^{i,k}$
4: **for all** process $j \neq i$ **do**
5: Send a marker to j
6: **end for**
7: $k_{i,i} := k$
8: Mark ϕ_i^i as undefined
9: **end if**
10: **end if**
11: **if** $\|y_i^{i,t+1} - y_i^{i,t}\| < \varepsilon, \forall t \in \{k_{i,i}, \ldots, k-1\} : i \in P^t$ **then**
12: **if** ϕ_i^i undefined **then**
13: $\phi_i^i := 1$
14: **for all** process $j \neq i$ **do**
15: Send a flagged marker ϕ_i^i to j
16: **end for**
17: **end if**
18: **else**
19: **if** ϕ_i^i undefined **then**
20: $\phi_i^i := 0$
21: **for all** process $j \neq i$ **do**
22: Send a flagged marker ϕ_i^i to j
23: **end for**
24: Mark $\bar{y}_i^i$ as undefined
25: **end if**
26: **end if**
27: **if** marker received from a process $j \neq i$ **then**
28: $\bar{y}_j^i := y_j^{i,k}$
29: **end if**
30: **if** flagged marker ϕ_j^j received from a process $j \neq i$ **then**
31: $\phi_j^i := \phi_j^j$
32: **if** $\phi_j^i = 0$ **then**
33: Mark $\bar{y}_j^i$ as undefined
34: **end if**
35: **end if**
36: **if** $\bar{y}_j^i$ defined and $\phi_j^i = 1$ for all j **then**
37: **return** $\bar{y}^i$
38: **end if**

Algorithm 11 NFAIS protocol 5

```
1: if ‖y_i^{i,t+1} − y_i^{i,t}‖ < ε, ∀t ∈ {k_0^i, ..., k − 1} : i ∈ P^t then
2:    if ȳ_i^i undefined then
3:       ȳ_i^i := y_i^{i,k}
4:       for all process j ≠ i do
5:          Send a marker to j
6:       end for
7:    end if
8: end if
9: if marker received from a process j ≠ i then
10:   ȳ_j^i := y_j^{i,k}
11: end if
12: if ȳ_j^i defined for all j then
13:   return ȳ^i
14: end if
```

Now let us consider the mapping

$$\begin{array}{rccc} g: & E \times \cdots \times E & \to & E_1 \times \cdots \times E_p \\ & \begin{bmatrix} y^1 & \cdots & y^p \end{bmatrix}^{\mathsf{T}} & \mapsto & \begin{bmatrix} f_1(y^1) & \cdots & f_p(y^p) \end{bmatrix}^{\mathsf{T}} \end{array},$$

and the vector

$$\bar{y} = \begin{bmatrix} \bar{y}^1 & \cdots & \bar{y}^p \end{bmatrix}^{\mathsf{T}}.$$

Let $\|.\|_q$, with $q \in [1, +\infty)$, be vector norms given by

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

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

Assumption 5.5. *Let x and x' be vectors related by :*

$$x' = \begin{bmatrix} x_1 & \cdots & x'_j & \cdots & x_p \end{bmatrix}^{\mathsf{T}}, \qquad 1 \leq j \leq p,$$

with

$$\|x_j - x'_j\|_{(j)} < \varepsilon, \qquad \varepsilon \in \mathbb{R}.$$

Then

$$\forall i \in \{1, \ldots, p\}, \quad \exists\ \delta_{i,j} \in \mathbb{R}: \quad \|f_i(x) - f_i(x')\|_{(i)} < \delta_{i,j}\varepsilon.$$

Notation 5.1.

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

where $\delta_{i,j}(f)$ are the smallest $\delta_{i,j}$ satisfying Assumption 5.5.

$\delta(f)$ is thus a constant related to the variation of f upon variation of x. At last, we also need the following assumption.

Assumption 5.6. *A process sends its marker and armed flag-marker after at least η steady iterations.*

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

Proposition 5.3. *Under Assumption 5.2, and Assumption 5.4 to Assumption 5.6, Algorithm 10 satisfies:*

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

Proof. Consider again

$$\bar{y}_j^i = y_j^{i,k_{i,j}}, \qquad k_{i,j} \in \mathbb{N}.$$

Then according to (5.1) and (5.2), we have

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

Assumption 5.2, Assumption 5.4 and Assumption 5.6 ensure that

$$\operatorname{card}\left\{k \in \{\tau_i^j(k_{j,i}), \ldots, k_{i,i} - 1\} \mid i \in P^k\right\} \leq \eta. \tag{5.5}$$

Let us then consider:

$$\{k_1^i, \ldots, k_{m_i}^i\} = \left\{k \in \{\tau_i^j(k_{j,i}), \ldots, k_{i,i} - 1\} \mid i \in P^k\right\},$$

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

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

Then we have, for all $i \in \{1, \ldots, p\}$,

$$\begin{aligned}
\left\| f_i(\bar{x}) - f_i(\bar{y}^i) \right\|_{(i)} = & \left\| f_i\left(\bar{y}_1^1, \ldots, \bar{y}_p^p\right) - f_i\left(\bar{y}_1^i, \ldots, \bar{y}_p^i\right) \right\|_{(i)}, \\
\leq & \left\| f_i\left(\bar{y}_1^1, \ldots, \bar{y}_p^p\right) - f_i\left(\bar{y}_1^i, \bar{y}_2^2, \ldots, \bar{y}_p^p\right) \right\|_{(i)} \\
& + \left\| f_i\left(\bar{y}_1^i, \bar{y}_2^2, \ldots, \bar{y}_p^p\right) - f_i\left(\bar{y}_1^i, \bar{y}_2^i, \bar{y}_3^3, \ldots, \bar{y}_p^p\right) \right\|_{(i)} \\
& + \cdots \\
& + \left\| f_i\left(\bar{y}_1^i, \ldots, \bar{y}_{p-1}^i, \bar{y}_p^p\right) - f_i\left(\bar{y}_1^i, \ldots, \bar{y}_p^i\right) \right\|_{(i)}.
\end{aligned}$$

Accounting Assumption 5.5, it follows that

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

and finally,

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

Applying (5.5), which means $m_j \leq \eta$, and using Notation 5.1, we conclusively obtain that

$$\left\| f(\bar{x}) - \bar{x} \right\|_q < \left\| g(\bar{y}) - \bar{x} \right\|_q + p^{1/q}\ \delta(f)\ \eta\ \varepsilon.$$

□

Assumption 5.7. *There exists a real $\alpha < 1$ and a vector $w > 0$ such that*

$$\forall x, x' \in E, \quad \|f(x) - f(x')\|_\infty^w \leq \alpha \|x - x'\|_\infty^w.$$

Proposition 5.4. *Under Assumption 5.2, Assumption 5.4, Assumption 5.6 and Assumption 5.7, Algorithm 10 satisfies:*

$$\|g(\bar{y}) - \bar{x}\|_\infty^w \leq \varepsilon \quad \Longrightarrow \quad \|f(\bar{x}) - \bar{x}\|_\infty^w < \varepsilon',$$

with

$$\varepsilon = \frac{\varepsilon'}{1 + \eta \min\limits_{1 \leq i \leq p} w_i}.$$

Proof. Considering the proof of Proposition 5.3, we recall that

$$\left\|\bar{y}_i^i - \bar{y}_i^j\right\|_{(i)} < m_i\ \varepsilon.$$

According to Assumption 5.7, we have

$$\left\|f(\bar{x}) - f(\bar{y}^i)\right\|_\infty^w \leq \alpha \left\|\bar{x} - \bar{y}^i\right\|_\infty^w,$$

and then in particular,

$$\begin{aligned}\left\|f_i(\bar{x}) - f_i(\bar{y}^i)\right\|_{(i)} &\leq w_i\ \alpha \left\|\bar{x} - \bar{y}^i\right\|_\infty^w, \\ &\leq w_i\ \alpha \max_{1\leq j\leq p} \frac{\left\|\bar{y}_j^j - \bar{y}_j^i\right\|_{(i)}}{w_j}, \\ &< w_i\ \alpha \max_{1\leq j\leq p} \frac{m_j}{w_j}\ \varepsilon.\end{aligned}$$

It follows:

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

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

$$\|f(\bar{x}) - \bar{x}\|_\infty^w < \|g(\bar{y}) - \bar{x}\|_\infty^w + \eta \max_{1\leq i\leq p} \frac{1}{w_i}\ \varepsilon.$$

Then, by ensuring that

$$\|g(\bar{y}) - \bar{x}\|_\infty^w \leq \varepsilon, \qquad \varepsilon = \frac{\varepsilon'}{1 + \eta \min\limits_{1\leq i\leq p} w_i},$$

we conclusively satisfy:

$$\begin{aligned}\|f(\bar{x}) - \bar{x}\|_\infty^w &< \varepsilon + \eta \min_{1\leq i\leq p} w_i\ \varepsilon, \\ &< \varepsilon'.\end{aligned}$$

□

5.4 Conclusion

Building a distributed global solution vector $\bar{x}$ under asynchronous iterations makes it possible to design a termination criterion of the form

$$\bar{x} \simeq x^* \quad \Longleftrightarrow \quad r(\bar{x}) \simeq 0,$$

where r is a residual error evaluation function, just as in classical synchronous computation cases. This is therefore an efficient, decentralized and exact approach for the asynchronous iterations termination problem, which is achieved, till now, only through snapshot-based supervised termination, proposed in [35]. We presented here a thorough investigation of such termination methods, based on the well-known Chandy & Lamport snapshot algorithm [46], under various message delivering properties. We thus ended up improving [35] by minimizing overhead communication costs from $\mathcal{O}(n)$ to $\mathcal{O}(1)$, according to a general characterization of non-FIFO message delivering.

In synchronous iterations implementation, evaluating such a residual function is classically achieved through a global reduction operation. Therefore, one may expect very few implementation tasks to switch an existing solver into its asynchronous counterpart, as long as non-blocking snapshot and reduction tools are provided. This thus clearly depends on communication libraries, a matter which is discussed in the next chapter, for the well-known Message Passing Interface (MPI) specification.

Chapter 6

Implementation of asynchronous iterations

6.1 Algorithmic framework

6.1.1 Distributed iterative computing

Let us still consider arbitrary mappings (2.12), for which there exists a vector x^* such that $f(x^*) = x^*$. Algorithm 12 describes a trivial parallel procedure implementing the classical computational model (2.10). For each iteration on each process, compu-

Algorithm 12 Trivial parallel iterative scheme

1: $k := 0$
2: **repeat**
3: $\quad x_i^{k+1} := f_i\left(x_1^k, \ldots, x_p^k\right)$
4: $\quad$ **for all** $j \in \{1, \ldots, i-1, i+1, \ldots, p\}$ **do**
5: $\quad\quad$ Request reception of x_j^{k+1} from process j
6: $\quad\quad$ Request sending of x_i^{k+1} to process j
7: $\quad$ **end for**
8: $\quad$ Wait for communication completion
9: $\quad k := k + 1$
10: **until** $x^k \simeq x^*$
11: **return** x^k

tation is performed first, then messages are sent and received. In the MPI framework, any process willing to send data to a slower one thus needs to wait until the latter finishes its computation phase and requests the reception of the message. Such dedicated communication phases could therefore considerably decrease the parallel computational efficiency. A well known improvement consists of Algorithm 13 which partially overlaps communication and computation phases. In this scheme, message

Algorithm 13 Overlapping parallel iterative scheme

1: $k := 0$
2: **repeat**
3: **for all** $j \in \{1, \ldots, i-1, i+1, \ldots, p\}$ **do**
4: Request reception of x_j^{k+1} from process j
5: **end for**
6: $x_i^{k+1} := f_i\left(x_1^k, \ldots, x_p^k\right)$
7: **for all** $j \in \{1, \ldots, i-1, i+1, \ldots, p\}$ **do**
8: Request sending of x_i^{k+1} to process j
9: **end for**
10: Wait for communication completion
11: $k := k + 1$
12: **until** $x^k \simeq x^*$
13: **return** x^k

reception is requested from the beginning of the iteration, therefore a process can start sending data immediately after its computation phase even when the destination process is still computing. This requires a little more memory space, since the buffers x_j^{k+1} and x_j^k are simultaneously accessed, but the overall computation time is generally decreased compared to the trivial scheme. Indeed, there is almost no more time dedicated to communication on slowest processes, which, on another hand, does not provide much performance gain when the workload is perfectly balanced.

Yet, to reach a full overlapping algorithmic scheme where processes never idle while waiting for communication completion, both the sets P^k and the functions τ_j^i from the general asynchronous iterations model (2.9) are left randomly defined by every execution of Algorithm 14. Quite simply, as soon as a process terminates an iteration, it starts a next one without waiting for the completion of communication requests. If new data were not received, latest ones can just be used in the next computation phase. Note that the global iteration variable k here remains abstract.

6.1.2 Asynchronous convergence detection

The loop stopping criterion in Algorithm 14 (line 11) ensures that the returned vector surely belongs to a set S^* of admissible solutions. It requires however to explicitly build a global vector $\bar{x} = \left(x_1^{k_1}, \ldots, x_p^{k_p}\right)$ which, then, will be evaluated as a potential solution. We saw in Chapter 5 that the snapshot-based supervised termination approach is currently the sole decentralized way of handling a global vector during distributed asynchronous iterations. Few remarks directly follow. First, the coordination phase in the SB96 protocol [35] (Algorithm 7) is not required to be able to build $\bar{x}$, even though it is useful to avoid unnecessarily performing snapshots at a stage where convergence is not yet likely to be reached. Similarly, partial snapshots

Algorithm 14 Asynchronous parallel iterative scheme

1: $k_i := 0$
2: **repeat**
3: **for all** $j \in \{1, \ldots, i-1, i+1, \ldots, p\}$ **do**
4: Request reception of $x_j^{\tau_j^i(k)}$ from process j
5: **end for**
6: $x_i^{k_i+1} := f_i\left(x_1^{\tau_1^i(k)}, \ldots, x_i^{k_i}, \ldots, x_p^{\tau_p^i(k)}\right)$
7: **for all** $j \in \{1, \ldots, i-1, i+1, \ldots, p\}$ **do**
8: Request sending of $x_i^{k_i+1}$ to process j
9: **end for**
10: $k_i := k_i + 1$
11: **until** $\left(x_1^{k_1}, \ldots, x_p^{k_p}\right) \simeq x^*$
12: **return** $\left(x_1^{k_1}, \ldots, x_p^{k_p}\right)$

could be arbitrarily initiated, regardless local convergence conditions, just as a collective operation requested by the application. This leads to Algorithm 15, which thus introduces a general partial snapshot operation where each process records some local data and send it (or a part of it) to its neighbor processes. Actually, such an

Algorithm 15 General partial snapshot operation

1: **if** $\bar{x}_i$ undefined **then**
2: $\bar{x}_i := x_i^{k_i+1}$
3: **for all** neighbors $j \neq i$ **do**
4: Send $\bar{x}_i$ to j
5: **end for**
6: **end if**
7: **if** $\bar{x}_j$ received from all neighbors $j \neq i$ **and** $\bar{x}_i$ defined **then**
8: **return** $\bar{x}$
9: **end if**

operation is really meaningful as a non-blocking global synchronization of parallel processes, which allows one to completely overlap computation and global synchronization phases, hence to run random asynchronous iterations with same stopping criteria used in the classical synchronous context.

6.2 Parallel programming pattern

6.2.1 MPI programming framework

Let us consider, for instance, a straightforward MPI-based implementation of Algorithm 13 (line 2 to 12), as shown in Listing 6.1. Both declaration and initialization of variables are omitted, as they can be easily deduced.

Listing 6.1: MPI-based overlapping scheme

```
1  while (res_vec_norm >= 1e-8) {
2    for (i = 0; i < numb_neighb; i++) {
3      MPI_Irecv(recv_buf2[i], rbuf_size[i], dtype, neighb_rank[i], tag,
           comm, &(recv_request[i]));
4    }
5    Copy(sol_vec_buf, sol_vec_buf2);
6    Compute(recv_buf, sol_vec_buf);
7    Map(sol_vec_buf, send_buf);
8    for (i = 0; i < numb_neighb; i++) {
9      MPI_Isend(send_buf[i], sbuf_size[i], dtype, neighb_rank[i], tag,
           comm, &(send_request[i]));
10   }
11   for (i = 0; i < sol_vec_size; i++) {
12     res_vec_buf[i] = abs(sol_vec_buf[i] - sol_vec_buf2[i]);
13   }
14   MPI_Allreduce(MPI_IN_PLACE, res_vec_buf, res_vec_size, dtype,
        MPI_MAX, comm);
15   res_vec_norm = max(res_vec_buf);
16   MPI_Waitall(numb_neighb, recv_request, MPI_STATUSES_IGNORE);
17   MPI_Waitall(numb_neighb, send_request, MPI_STATUSES_IGNORE);
18   Swap(recv_buf, recv_buf2);
19 }
```

A procedure *Compute* encloses the main computation part, which corresponds to

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

The variable *sol_vec_buf* (solution vector buffer) would therefore contain x_i^k data as input, and then x_i^{k+1} data as output, while the variable *recv_buf* would contain the set $\{x_j^k\}_{j \neq i}$. Using the procedure *Copy*, x_i^k is first saved into the temporary variable *sol_vec_buf2*, in view of a comparison between x_i^k and x_i^{k+1}. Next, a procedure *Map* is called to set a message sending buffer for each neighbor process, based on the newly computed solution. Here, for example, all of the sending buffers would contain the same x_i^{k+1} data. For the stopping criterion, we assume for instance an iterative method verifying:

$$x^{k+1} \simeq x^* \quad \Longleftrightarrow \quad \left\|x^{k+1} - x^k\right\|_\infty \simeq 0.$$

A function *abs* is therefore used to get the absolute value (or the module) of a real (or a complex) number, while a function *max* gives the maximum entry of an array.

The keyword *MPI_IN_PLACE* indicates that the variable *res_vec_buf* (residual vector buffer) is used as both input and output of the collective reduction operation. Finally, the procedure *Swap* exchanges two pointers, which is useful to avoid copy of data when handling the temporary buffers *recv_buf2*.

Classically, MPI routines *MPI_Irecv* and *MPI_Isend* are used to request message reception and sending, respectively, while the norm-based stopping criterion is evaluated by means of the collective procedure *MPI_Allreduce*. As suggested however in [27], it would be more efficient to take advantage of persistent requests *MPI_Recv_init* and *MPI_Send_init*. This reduces the routines invocation overhead, as shown by Listing 6.2, even if, on the other hand, it prevents one from swapping pointers on reception buffers.

Listing 6.2: MPI-based overlapping scheme with persistent requests

```
1  for (i = 0; i < numb_neighb; i++) {
2    MPI_Recv_init(recv_buf2[i], rbuf_size[i], dtype, neighb_rank[i],
         tag, comm, &(recv_request[i]));
3    MPI_Send_init(send_buf[i], sbuf_size[i], dtype, neighb_rank[i], tag,
         comm, &(send_request[i]));
4  }
5  while (res_vec_norm >= 1e-8) {
6    MPI_Startall(numb_neighb, recv_request);
7    Copy(sol_vec_buf, sol_vec_buf2);
8    Compute(recv_buf, sol_vec_buf);
9    Map(sol_vec_buf, send_buf);
10   MPI_Startall(numb_neighb, send_request);
11   for (i = 0; i < sol_vec_size; i++) {
12     res_vec_buf[i] = abs(sol_vec_buf[i] - sol_vec_buf2[i]);
13   }
14   MPI_Allreduce(MPI_IN_PLACE, res_vec_buf, res_vec_size, dtype,
         MPI_MAX, comm);
15   res_vec_norm = max(res_vec_buf);
16   MPI_Waitall(numb_neighb, recv_request, MPI_STATUSES_IGNORE);
17   MPI_Waitall(numb_neighb, send_request, MPI_STATUSES_IGNORE);
18   Copy(recv_buf2, recv_buf);
19 }
```

Implementing asynchronous iterative methods within the MPI framework obviously raises nontrivial questions related to the management of successive communication requests, the management of associated buffers and the evaluation of such a stopping criterion. On performance side, for instance, it might be preferable to allow for several message reception requests during computation phase, in order to use the least delayed data. On another hand, one notices that overlapping (Algorithm 13) and asynchronous schemes (Algorithm 14) nearly follow the same algorithmic pattern. It could therefore be desirable to maintain a unique implementation code where a parameter will be checked in order to produce, at runtime, either classical or random asynchronous iterations.

Recently, Jack [52] was proposed as a C++ MPI-based communication library

for distributed iterative computing, where the asynchronous convergence detection issue was handled by means of the heuristic from [73]. It provides an application programming interface (API) very close to MPI routines, including the definition of a new type of communication request devoted to the automatic reception of any incoming message during computation phases. The last received data are however delivered into the computation reception buffer only when explicitly requested. Not surprisingly, some communication objects still have to be managed outside the library, just as in classical MPI-based programming. We has therefore developed a second version, Jack2 [51], by following a quite different design approach which provides a completely encapsulating API and allows exact convergence testing through actual global residual computation.

6.2.2 Jack2 programming framework

In a study from [75] about the scalability of MPI-based applications, the authors pointed out the fact that the MPI specification does not allow one to initially provide information about communication graphs, while this could help optimizing communication resources. Such a graph should however be handled in a distributed way to avoid memory overhead costs. Somehow though, [27] expressed a similar programming pattern through the use of persistent requests, where message buffers and neighbor processes are specified once for all before the iterations loop (see Listing 6.2). Jack2 follows same ideas to provide a complete initialization part for repetitive communication parameters. Of course, our design remains flexible enough to allow several ways of interacting with the library, which as well implies several levels of encapsulation of communication objects.

Full initialization of a Jack2 communicator object therefore requires to explicitly specify variables about communication graph, communication buffers and residual evaluation. Listing 6.3 describes an implementation of the overlapping scheme within the Jack2 framework.

Listing 6.3: Jack2-based overlapping scheme

```
1  JACKSyncComm<double,int> sync_comm;
2  sync_comm.Init(mpi_comm);
3  sync_comm.InitRecv(recv_buf, rbuf_size, neighb_rank, numb_neighb);
4  sync_comm.InitSend(send_buf, sbuf_size, neighb_rank, numb_neighb);
5  sync_comm.InitAllReduce(res_vec_buf, res_vec_buf, res_vec_size,
       MPI_MAX);
6  while (res_vec_norm >= 1e-8) {
7    Copy(sol_vec_buf, sol_vec_buf2);
8    Compute(recv_buf, sol_vec_buf);
9    Map(sol_vec_buf, send_buf);
10   sync_comm.Send();
11   for (i = 0; i < sol_vec_size; i++) {
12     res_vec_buf[i] = abs(sol_vec_buf[i] - sol_vec_buf2[i]);
13   }
```

```
14    sync_comm.AllReduce();
15    res_vec_norm = max(res_vec_buf);
16    sync_comm.Recv();
17  }
18  sync_comm.Finalize();
```

As soon as the communicator object is initialized with communication graph and buffers, incoming communication channels are opened via MPI non-blocking reception requests, so that each process is ready to receive computation data from any of its neighbors. The method *Recv* of the class *JACKSyncComm* is meant to deliver pending messages received from all of the neighbors, then to throw new reception requests. The method *Finalize* therefore ensures the cancellation of the last, unfulfilled, ones. Relatively to the MPI-based implementation, the temporary reception buffers *recv_buf2* are now internally managed by the library, and collective reduction is made persistent too. The application thus does not need to handle any pure communication-related object, barring communicators themselves. One notices, as minor drawback, the method *Send* which is blocking when invoked on a Jack2 synchronous communicator. This prevents one from overlapping message sending and residual computation. Our send/receive semantic however follows the put/get paradigm advocated by related works, e.g., [38] (Jace) and [42] (Crac).

With such an API, it becomes quite straightforward to handle communication requests during random asynchronous iterations. Considering a communicator of type *JACKAsyncComm*, only lines 11–15 of Listing 6.3 would be rearranged, as shown in Listing 6.4 (lines 6–12). The initialization part of the communicator is omitted, since the same interface is shared by both *JACKAsyncComm* and *JACKSyncComm*.

Listing 6.4: Jack2-based asynchronous scheme without accurate termination

```
1  while (res_vec_norm >= 1e-8) {
2    Copy(sol_vec_buf, sol_vec_buf2);
3    Compute(recv_buf, sol_vec_buf);
4    Map(sol_vec_buf, send_buf);
5    async_comm.Send();
6    async_comm.AllReduce(&end_flag);
7    if (end_flag) {
8      res_vec_norm = max(res_vec_buf);
9      for (i = 0; i < sol_vec_size; i++) {
10       res_vec_buf[i] = abs(sol_vec_buf[i] - sol_vec_buf2[i]);
11     }
12   }
13   async_comm.Recv();
14 }
```

Here, the methods *Recv* and *Send* are non-blocking, and the library regulates underlying MPI requests itself. Of course, messages can be received many times between two successive *Recv* calls. The non-blocking version of the method *AllReduce* returns a flag indicating whether the reduction operation is completed or not. Its input-output parameter *res_vec_buf* can be safely accessed once the flag is armed, then

the next *AllReduce* call automatically starts a new collective reduction. Nonetheless, in order to ensure correct termination, one needs to add the partial snapshot operation previously described by Algorithm 15. Lines 1, 2 and 8–24 of Listing 6.5 indicate a possible pattern related to the integration of such a non-blocking collective operation.

Listing 6.5: Jack2-based asynchronous scheme

```
1  async_comm.InitSnapshot(ss_sol_vec_buf, ss_recv_buf, sol_vec_buf,
       sol_vec_size);
2  reduce_flag = 0;
3  while (res_vec_norm >= 1e-8) {
4    Copy(sol_vec_buf, sol_vec_buf2);
5    Compute(recv_buf, sol_vec_buf);
6    Map(sol_vec_buf, send_buf);
7    async_comm.Send();
8    if (!reduce_flag) {
9      async_comm.Snapshot(&end_flag);
10     if (end_flag) {
11       Copy(ss_sol_vec_buf, sol_vec_buf2);
12       Compute(ss_recv_buf, ss_sol_vec_buf);
13       for (i = 0; i < sol_vec_size; i++) {
14         res_vec_buf[i] = abs(ss_sol_vec_buf[i] - sol_vec_buf2[i]);
15       }
16       reduce_flag = 1;
17     }
18   } else {
19     async_comm.AllReduce(&end_flag);
20     if (end_flag) {
21       res_vec_norm = max(res_vec_buf);
22       reduce_flag = 0;
23     }
24   }
25   async_comm.Recv();
26 }
```

Just like the method *AllReduce*, an armed flag is returned to indicate the end of the snapshot operation. When so, the variables *ss_sol_vec_buf* and *ss_recv_buf* respectively contain $\bar{x}_i$ and $\{\bar{x}_j\}_{j\neq i}$ data, so that invoking the procedure *Compute* allows one to perform a global iteration

$$z_i := f_i\left(\bar{x}_1, \ldots, \bar{x}_p\right), \quad \forall i \in \{1, \ldots, p\},$$

whereupon each local residual vector is set to $|z_i - \bar{x}_i|$. A flag-type variable *reduce_flag* is then armed to make the process enter the reduction phase which will evaluate the global residual norm as

$$\max_{1\leq i\leq p} \max |z_i - \bar{x}_i|.$$

It thus turns out that the stopping criterion implies:

$$f\left(x_1^{k_1},\ldots,x_p^{k_p}\right) \simeq x^* \quad \Longleftrightarrow \quad \left\|f\left(x_1^{k_1},\ldots,x_p^{k_p}\right) - \left(x_1^{k_1},\ldots,x_p^{k_p}\right)\right\|_\infty \simeq 0,$$

with $k_i \leq k$, $\forall i \in \{1,\ldots,p\}$.

Finally, to produce a unique code for both overlapping and asynchronous schemes, the top front-end interface of Jack2 provides two classes, *JACKComm* and *JACKConv*, which manage communication requests and convergence detection, respectively. On message sending and reception aspects, *JACKComm* obviously features the same init/send/recv interface shared by *JACKSyncComm* and *JACKAsyncComm*. Methods *SwitchSync* and *SwitchAsync* allows one to choose either blocking or non-blocking mode at any time. Contrariwise, a general convergence detection object needs one more layer of abstraction, due to the differences introduced for handling the termination of asynchronous iterations. Moreover, initiating snapshots at an early stage of the iterative process unnecessarily holds communication resources. The convergence detector therefore implements some asynchronous iterations termination protocols described in Chapter 5. This leads to one another collective routine which encompasses local convergence monitoring, snapshot and reduction phases, according to each protocol. Such a routine thus additionally requires an input flag-type parameter indicating at any time whether the process is under local convergence or not. We mention however that the synchronous mode directly jumps to the reduction phase.

Still, Listing 6.5 makes two calls of the procedure *Compute*, the second one being for performing an iteration with the snapshot-based solution vector. Being so intrusive makes it hard to design a standalone method which would enclose the whole global residual evaluation. Listing 6.6 instead describes a programming pattern which shows the possibility to keep the application code separated from the convergence detection procedure.

Listing 6.6: Non-intrusive Jack2-based asynchronous scheme

```
1  swap_flag = 0;
2  reduce_flag = 0;
3  while (res_vec_norm >= 1e-8) {
4    Copy(sol_vec_buf, sol_vec_buf2);
5    Compute(recv_buf, sol_vec_buf);
6    Map(sol_vec_buf, send_buf);
7    async_comm.Send();
8    if (swap_flag) {
9      for (i = 0; i < sol_vec_size; i++) {
10       res_vec_buf[i] = abs(sol_vec_buf[i] - sol_vec_buf2[i]);
11     }
12     Swap(recv_buf, ss_recv_buf);
13     Swap(sol_vec_buf, ss_sol_vec_buf);
14     reduce_flag = 1;
15     swap_flag = 0;
16   }
```

```
17    if (!reduce_flag) {
18      async_comm.Snapshot(&end_flag);
19      if (end_flag) {
20        Swap(recv_buf, ss_recv_buf);
21        Swap(sol_vec_buf, ss_sol_vec_buf);
22        swap_flag = 1;
23      }
24    } else {
25      async_comm.AllReduce(&end_flag);
26      if (end_flag) {
27        res_vec_norm = max(res_vec_buf);
28        reduce_flag = 0;
29      }
30    }
31    async_comm.Recv();
32  }
```

Here now, once a snapshot is completed, buffers are temporarily swapped (lines 20 and 21) such that, at the next iteration, variables *sol_vec_buf* and *recv_buf* actually contain the output of the snapshot, and hence the residual vector buffer is rightly filled up using globally consistent data. Once done, the variables are swapped back (lines 12 and 13), in order to avoid disrupting the main iterative process. Snapshot buffers and swapping procedures could thus be managed by the standalone residual evaluation routine if we explicitly distinguish local and global residual data. More precisely, the library can internally make a copy of the residual vector buffer at this iteration where it contains snapshot-based data, and then use its own internal buffer as both input and output of the reduction operation. Listing 6.7 describes the resulting general Jack2-based programming pattern for both classical and random asynchronous iterations.

Listing 6.7: Jack2-based overlapping/asynchronous scheme

```
1  JACKComm<double,int> comm;
2  JACKConv<double,int> conv;
3  comm.Init(mpi_comm);
4  comm.InitRecv(recv_buf, rbuf_size, neighb_rank, numb_neighb);
5  comm.InitSend(send_buf, sbuf_size, neighb_rank, numb_neighb);
6  conv.Init(mpi_comm, neighb_rank, numb_neighb, neighb_rank,
       numb_neighb);
7  conv.InitAllReduce(res_vec_buf, res_vec_buf, res_vec_size, MPI_MAX);
8  if (async_flag) {
9    conv.InitSnapshot(&sol_vec_buf, sol_vec_size, &recv_buf, rbuf_size,
         send_buf, sbuf_size, &local_conv_flag);
10   comm.SwitchAsync();
11   conv.SwitchAsync();
12 }
13 while (res_vec_norm >= 1e-8) {
14   Copy(sol_vec_buf, sol_vec_buf2);
15   Compute(recv_buf, sol_vec_buf);
16   Map(sol_vec_buf, send_buf);
```

```
17    comm.Send();
18    for (i = 0; i < sol_vec_size; i++) {
19      res_vec_buf[i] = abs(sol_vec_buf[i] - sol_vec_buf2[i]);
20    }
21    local_conv_flag = (max(res_vec_buf) < 1e-8);
22    conv.SnapReduce(&end_flag);
23    if (end_flag) {
24      res_vec_norm = max(res_vec_buf);
25    }
26    comm.Recv();
27  }
```

6.3 Architecture and design of Jack2

6.3.1 Overall features

As a C++ library, Jack2 follows, as much as possible, object-oriented paradigms thoroughly integrated by the Unified Modeling Language (UML) [76], which is one of the top popular theoretical tools for software modeling. Figure 6.1 proposes an implementation-level UML class diagram describing the architecture of the library. Even though the classes *JACKSyncComm* and *JACKAsyncComm* share the same types of operations, switching between blocking (*SwitchSync*) and non-blocking (*SwitchAsync*) communication routines is a specific behavior of *JACKComm*, which therefore is not a simple interface or an abstract class. Instead, delegation is considered by means of function pointers which allows us to invoke methods of the appropriate class instance, according to the communication mode, without using *if* conditions. The same pattern applies to classes *JACKConv*, *JACKSyncConv* and *JACKAsyncConv*. Both communicators and convergence detectors rely on instances of a class *JACKAllReduce*. For blocking modes, this simply results in *MPI_Allreduce* calls, which will be the case for non-blocking modes too, with the routine *MPI_Iallreduce* in versions supporting the third MPI specification. Our own non-blocking reduction implementation is based on a distributed, non-deterministic, leader election protocol designed over acyclic graphs (see, e.g., [74, Section 4.4.3]). A class *JACKSpanningTree* therefore builds a distributed spanning tree upon the application communication graph, yielding, for each process, neighbors in a communication tree (*tneighb*). If no graph information is specified, an all-to-all connectivity (complete graph) is assumed, as currently implied by the MPI standard. The communication tree is produced by means of an echo algorithm [77], which, contrarily to a structural minimum spanning tree, effectively minimizes traversals time by taking into account actual communication delays. At last, the class *JACKSnapshot* implements the general partial snapshot given by Algorithm 15, while classes *JACKSnapshotXXX* relate to the snapshot-based supervised termination protocols SB96 (Algorithm 7 and Algorithm 8), NFAIS2 (Algorithm 6) and NFAIS5 (Algorithm 11),

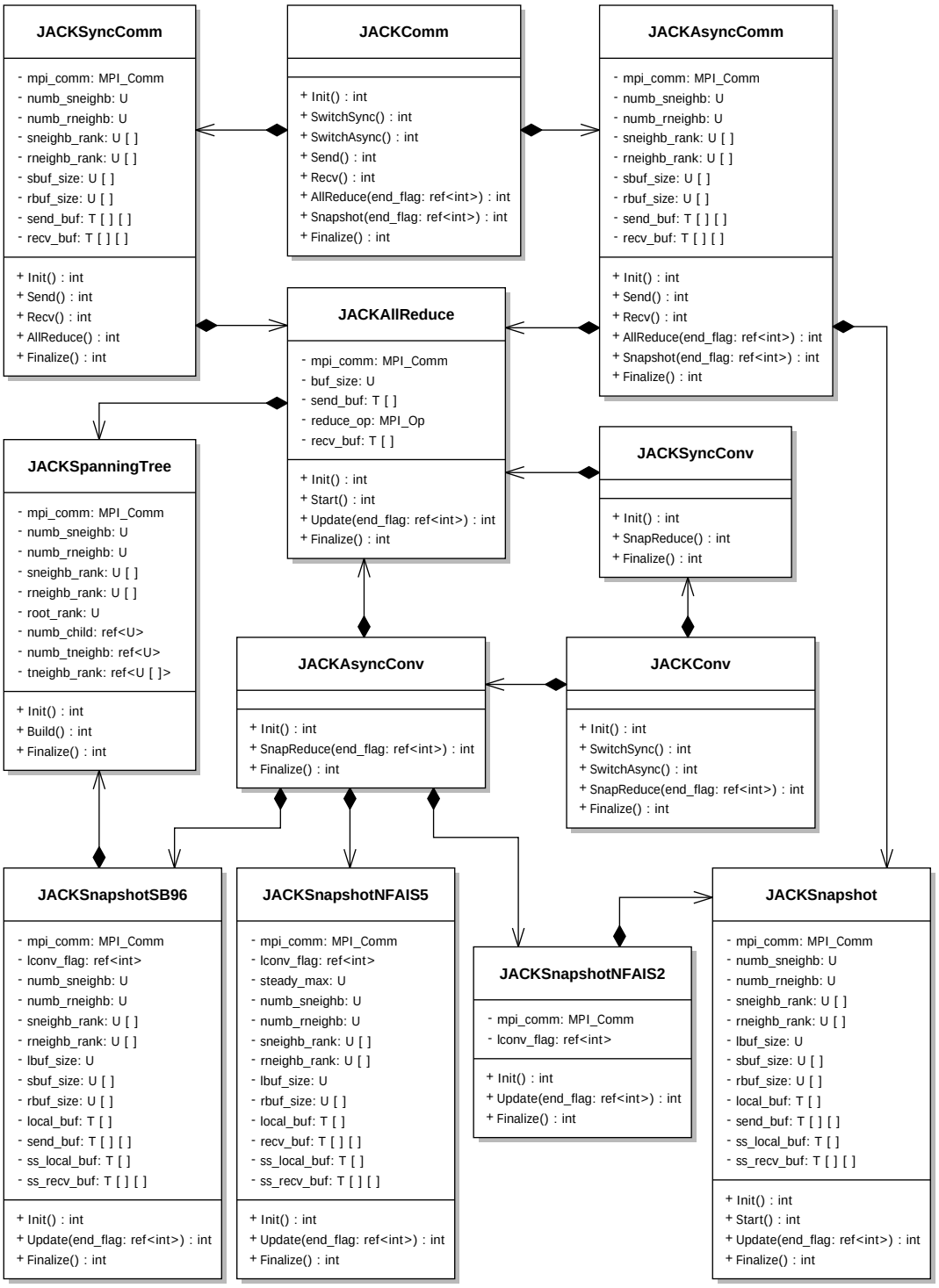


Figure 6.1: Main classes featuring the basic interface of Jack2.

respectively. Other non-intrusive termination protocols can be added, following the same pattern.

Jack2 does not launch any thread for its non-blocking collective operations. Instead, the execution of the underlying distributed protocols progresses each time a call is made to the corresponding routine. Considering the non-blocking MPI operations which are almost always running in background, this prevents the library for introducing additional concurrency against the computation part of the application, which therefore can remain the main executed steps during asynchronous iterations. Users thus have enough flexibility to regulate themselves the execution frequency of Jack2 objects. Next sections give further details about their specific behavior.

6.3.2 Point-to-point communication

While it is commonly admitted that, during random execution of asynchronous iterations, processes should benefit from the least delayed data, the classical corresponding *put/get* semantic implies to only fill up the application buffers with the last of the messages received between two successive calls of the message reception routine. To be more precise, let us once more consider the computational model (2.9), and take n_i, with $i \in \{1, \dots, p\}$, and n as in (2.5). Actually, convergence could be guaranteed for a more general model featuring:

$$x_l^{k+1} = f_l\left(x_1^{\rho_1^l(k)}, \dots, x_n^{\rho_n^l(k)}\right), \quad l \in N^k, \tag{6.1}$$

with, as well,

$$\rho_t^l(k) \leq k, \quad \forall l, t \in \{1, \dots, n\}, \qquad N^k \subseteq \{1, \dots, n\}.$$

Nonetheless, delivering only last received messages consists of particularly setting:

$$\begin{cases} \rho_1^l(k) = \cdots = \rho_{n_1}^l(k), & \dots, \quad \rho_{n-n_p+1}^l(k) = \cdots = \rho_n^l(k), \quad \forall l \in \{1, \dots, n\}, \\ \rho_t^1(k) = \cdots = \rho_t^{n_1}(k), & \dots, \quad \rho_t^{n-n_p+1}(k) = \cdots = \rho_t^n(k), \quad \forall t \in \{1, \dots, n\}. \end{cases}$$

To be able to effectively implement the model (6.1) while allowing the possibility to have $\rho_t^{l_2}(k) > \rho_t^{l_1}(k)$, with $l_1 < l_2$ and both l_1 and l_2 belonging to, for instance, the set $\{1, \dots, n_1\}$, Jack2 instead delivers any incoming message directly into the corresponding message reception buffer of the application. Therefore, at a given loop iteration, processes can possibly use more recent data for last updated local components, instead of only benefiting from data received at the beginning of the computation phase. Such an approach for message reception provides more flexibility since here also, users still have the possibility to manage two reception buffers if they rather wish to avoid including data on the fly.

To achieve that, Jack2 is parameterized to activate several MPI reception requests on each same reception buffer provided by the user application. The MPI

specification guarantees FIFO delivering on communication links which are identified by the triplet communicator/source/tag. Therefore, the method *Recv* of the class *JACKAsyncComm* is only meant to reactivate completed requests, as shown for instance by Listing 6.8.

Listing 6.8: Method *Recv* of class *JACKAsyncComm*

```
for (i = 0; i < numb_rneighb; i++) {
  for (j = 0; j < numb_req_per_rneighb; j++) {
    MPI_Test(&(recv_req[i][j]), &flag, MPI_STATUS_IGNORE);
    if (flag) {
      MPI_Start(&(recv_req[i][j]));
    }
  }
}
```

Users are allowed to configure such a parameter (the number of reception requests per neighbor) which defines a maximum message reception rate. This implies on the other hand that allowing a higher message sending rate could lead to a counter performance, as the number of pending MPI sending requests may quickly increase, which would yield much more delayed iterations data. Jack2 therefore discards message sending requests on busy communication links, as one can see in Listing 6.9.

Listing 6.9: Method *Send* of class *JACKAsyncComm*

```
for (i = 0; i < numb_sneighb; i++) {
  MPI_Test(&(send_req[i]), &flag, MPI_STATUS_IGNORE);
  if (flag) {
    MPI_Start(&(send_req[i]));
  }
}
```

With a sufficiently high message reception rate, all of the message sending requests are theoretically able to be satisfied. We however note that, on homogeneous computing resources anyway, it is unlikely to have a process running several iterations while its neighbor is completing only one. On such platforms, very few incoming messages are therefore expected during computation phases. In any case, discarding message sending requests can be unavoidable if data transmission itself actually covers at least two loop iterations, which somehow exhibits advantages of running asynchronous iterations. In the *put/get* semantic, while a message is buffered for being sent, it can be replaced by a new *put* call. This can be achieved here through the MPI synchronous mode routine *MPI_Issend*, since data are not buffered by the MPI library, if it follows the standard recommendation, and hence, the actual message sending buffer is directly updated by the user application (procedure *Map* in Listing 6.7, for instance).

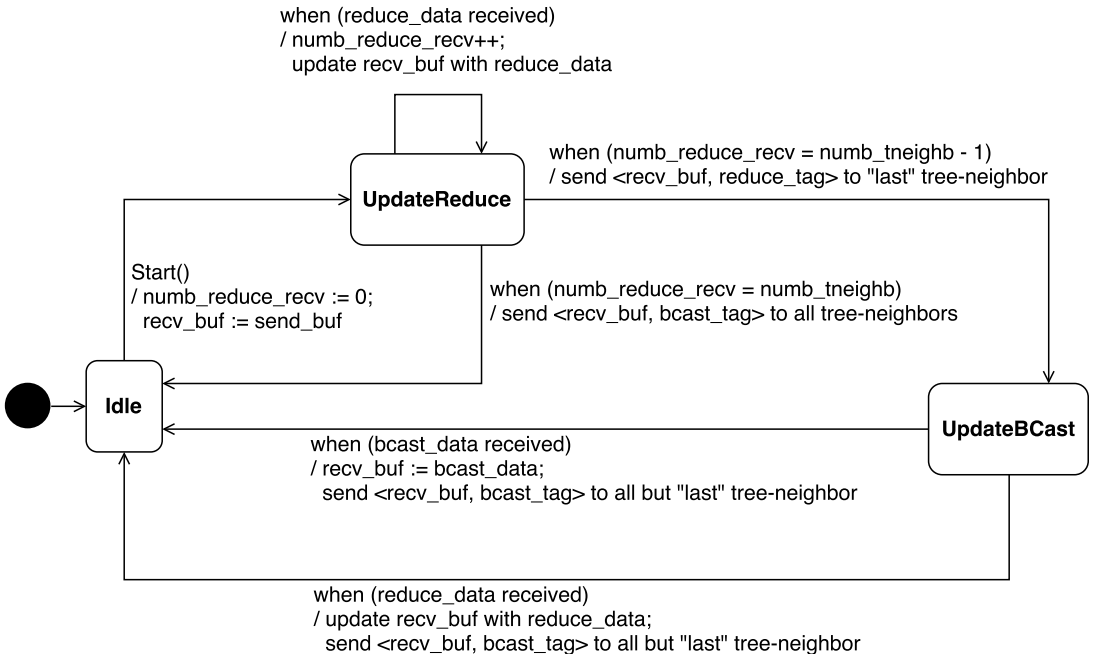


Figure 6.2: States and behavior of a *JACKAllReduce* instance.

6.3.3 Collective operations

We shall finally focus on non-blocking collective operations, which are not executed within parallel threads, but instead progress at the pace of explicit requests from the user application. These classes are designed from a state machine view, such that each time their method *Update* is called, a set of apt conditions (seen as occurring events) is checked in order to step forward in the execution of the distributed protocol. Each particular set of concurrently expected events therefore defines a state of the running class instance. Here as well, in order to avoid *if* conditions, a function pointer is moved throughout a set of private methods implementing each behavioral state of the class.

Figure 6.2 presents an UML statechart diagram of the class *JACKAllReduce*. Just like in the MPI specification, this operation successively consists of a reduction and a broadcast phases. We recall that the reduction operation is performed along with a tree-based leader election protocol described in [74, Section 4.4.3] (also see, e.g., [52]). The broadcast messages then follow the inverse path of the reduction messages. Upon start request, processes enter an *UpdateReduce* state where three types of event are expected. Leaves in the tree potentially trigger the transition toward an *UpdateBCast* state, since they have only one neighbor in the tree, and therefore at least satisfy the condition "*numb_reduce_recv = numb_tneighb - 1*". They thus send their local input data to that neighbor. The other processes loop on the *UpdateReduce* state while combining their local data with received ones, accord-

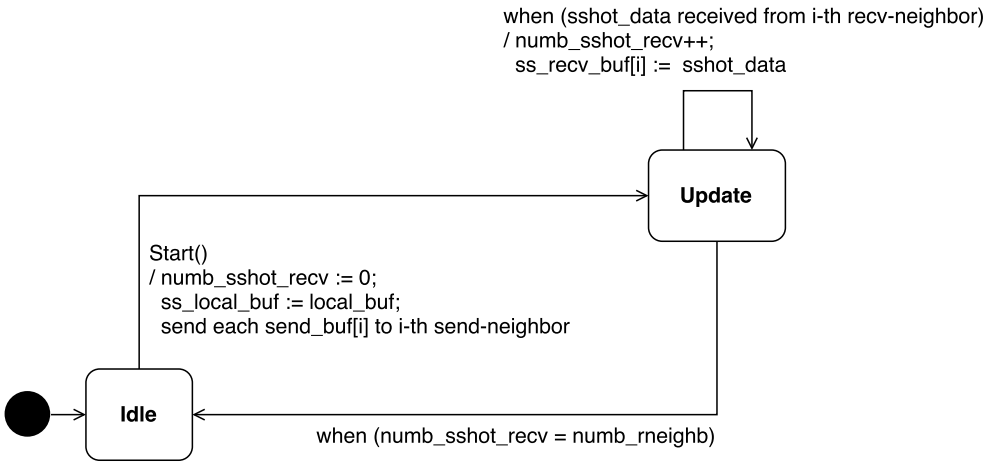


Figure 6.3: States and behavior of a *JACKSnapshot* instance.

ing to the type of reduction, until they evolve toward either the *UpdateBCast* or the initial *Idle* state. We should recall that the looping transition and the transition to *UpdateBCast* are based on concurrent events, so that the former might be triggered while the *when* condition of the latter is satisfied, which would results in reaching the other condition "*numb_reduce_recv = numb_tneighb*". Actually, this may happen to only one of the processes (the unique elected "leader"), which thus initiates the broadcast phase. While being in the *UpdateBCast* state, processes thus expect a broadcast message (containing the result of the reduction operation) from the neighbor to which they previously sent their reduction message. The received data are then forwarded to their other neighbors in the tree, and so on to the leaves. It may however happen that all of the processes enter this *UpdateBCast* state. In this case, two of them (the elected "co-leaders") receive a last reduction message from each other, upon which they both deduce the result of the reduction and broadcast it to their other neighbors in the tree, then also return to the *Idle* state.

The protocols implemented as classes *JACKSnapshot* and *JACKSnapshotXXX* have already been fully described through Algorithm 15 and Chapter 5. As summarized by Figure 6.3, the general partial snapshot exhibits only one set of expected events, which results in two quite simple states. It however clearly points out the fact that the message sending buffer provided by the user application must always contain consistent data, regarding the solution vector buffer (which is the local buffer here). This implies, for instance in Listing 6.6, that the procedure *Map* should always precede the call of the method *Snapshot*. The same remark also applies to snapshot-based supervised termination. The SB96 protocol (Algorithm 7 and Algorithm 8) is implemented by means of the statechart diagram in Figure 6.4. There, from an *UpdateCoordStart* state, the leaves of the spanning tree instantly evolve toward an *UpdateSShotStart* state, if they are under local convergence. Otherwise, they wait

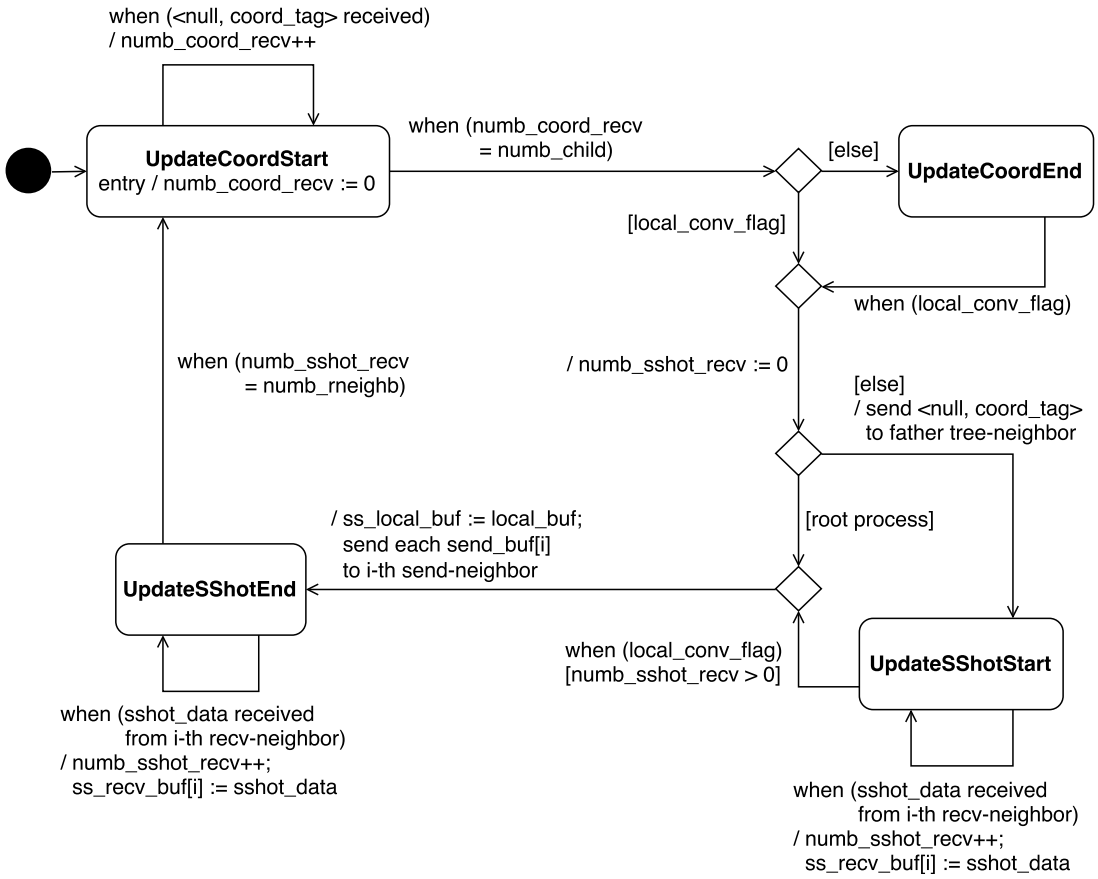


Figure 6.4: States and behavior of a *JACKSnapshotSB96* instance.

in an *UpdateCoordEnd* state for local convergence. Except the root of the tree, the other processes trigger the same transitions once they have received a coordination message from all of their children. The root process instead initiates the snapshot phase and enters an *UpdateSShotEnd* state. Once processes in the *UdpateSShotStart* state again observes local convergence, they evolve to the *UpdateSShotEnd* state too, but only if they received at least one snapshot message which triggered the looping transition. Finally, from *UpdateSshotEnd*, processes switch back to *UpdateCoord-Start* once all of their expected snapshot messages have been received.

Figure 6.5 to Figure 6.7 relate to the NFAIS1 (Algorithm 5), NFAIS2 (Algorithm 6) and NFAIS5 (Algorithm 11) protocols, respectively, where one can see differences about local convergence monitoring phases.

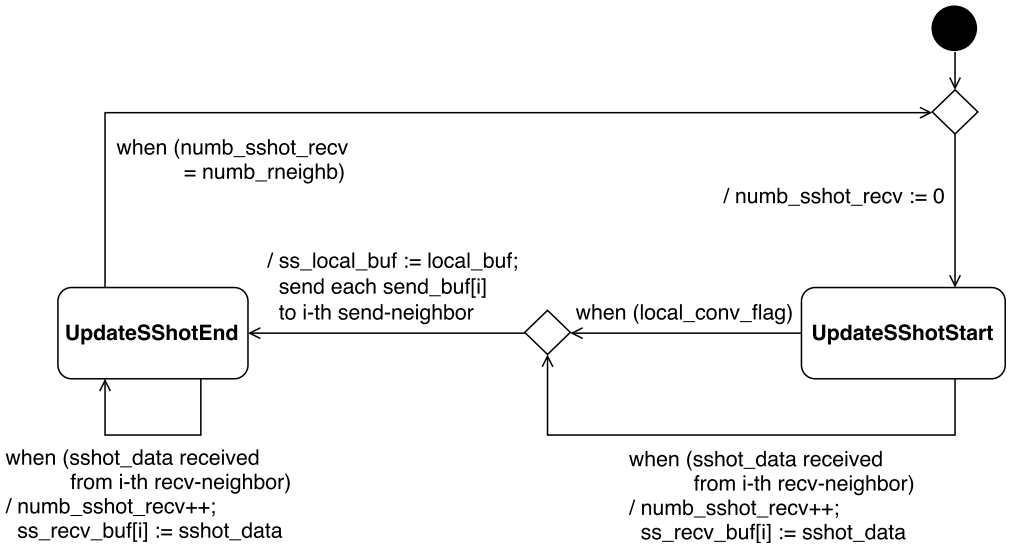


Figure 6.5: States and behavior of a *JACKSnapshotNFAIS1* instance.

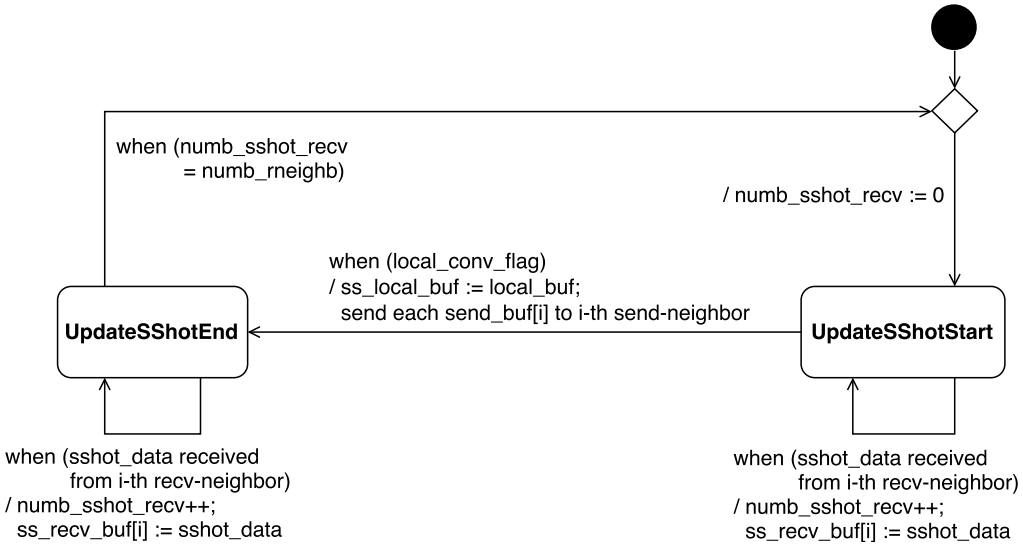


Figure 6.6: States and behavior of a *JACKSnapshotNFAIS2* instance.

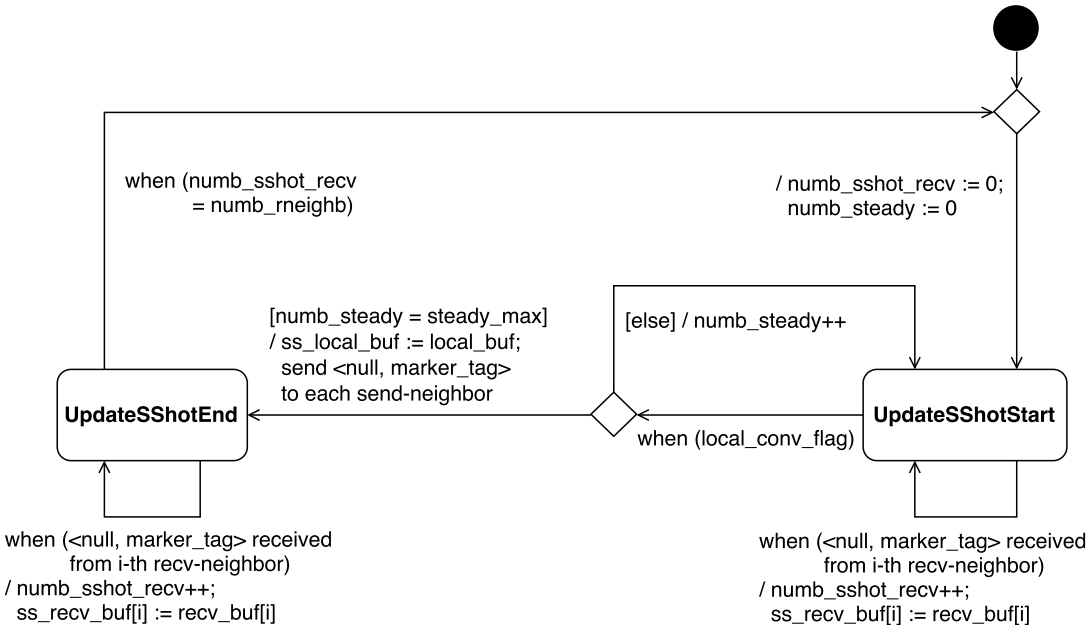


Figure 6.7: States and behavior of a *JACKSnapshotNFAIS5* instance.

Chapter 7

Experimental results

7.1 Implementation of asynchronous iterations

7.1.1 Experimental setting

To investigate implementation options of the proposed programming framework, we plugged Jack2 into an existing scientific application devoted to the parallel resolution of convection-diffusion problems of the form

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

where u is the unknown function defined on $\mathbb{R}^+ \times ([0,1])^3$. The host software application implements regular finite-differences and backward Euler discretization schemes, which results in solving successive sparse linear systems of the form

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

over small constant time steps. The three-dimensional domain $([0,1])^3$ is decomposed in a two-dimensional coarse grid on the (x,y)-plane, as shown on Figure 7.1. Parallel solutions are computed by following a non overlapping additive Schwarz pattern [43] (see also, e.g., [44]) where Gauss-Seidel iterations are performed inside each sub-domain. The main program sets up corresponding band-diagonal matrices, each band being stored as a separate one dimensional array, then it sequentially runs a time iterations loop wherein the linear system solver is invoked by each of the parallel processes. Communication requests inside the solver procedure have been reshaped from the MPI to the Jack2 framework. Each process handles exactly one sub-domain, and the number of processes always equals the number of processor cores used. Figure 7.2 gives an example of one linear system resolution with 16 processes, where we notice, for asynchronous iterations, the temporary discontinuity of the solution over the interface between the sub-domains.

Experiments have been led on two mono-site computational platforms.

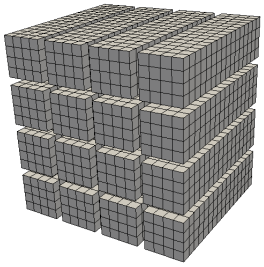


Figure 7.1: Domain discretization and partitioning schemes for Jack2 experimentation.

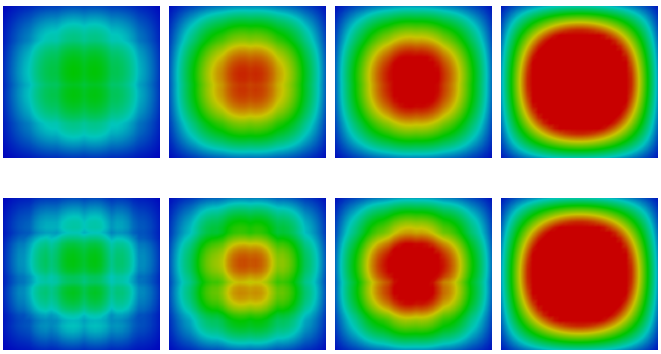


Figure 7.2: Comparison example of synchronous (top) and asynchronous (down) iterative solutions.

- An SGI ICE X supercomputer of 92 nodes on an FDR Infiniband network (56 Gb/s), provided by CentraleSupélec, Université Paris-Saclay, France. Each node consists of two Intel Haswell Xeon CPUs with 12 cores (24 cores per node) at 2.30 GHz, and 60 GB RAM allocated to running jobs. The MPI library SGI-MPT has been loaded as communication middleware.
- A Bullx B510 supercomputer of 5040 nodes, on a QDR Infiniband network (40 Gb/s), provided by the GENCI (Grand Equipement National de Calcul Intensif), at the TGCC (Très Grand Centre de calcul du CEA), France, under the Grant 2014-t2014069065. Each node consists of two Intel Sandy Bridge Xeon CPUs with 8 cores (16 cores per node) at 2.70 GHz, and 64 GB RAM allocated to running jobs. The MPI library Bullxmpi (OpenMPI) has been loaded as communication middleware.

Subsequent results summarize a wide set of experimental measurements related to each linear system resolution. Featured items are:

- residuals $r := \|A\widetilde{x} - b\|_\infty$, where $\widetilde{x}$ is the returned solution,

- execution times in seconds, denoted as "wt" (wall-clock time),
- numbers of iterations $k := \max_{1 \leq i \leq p}\{k_i\}$, where k_i is the number of iterations on the i-th process (we see from Table 7.2 that, for asynchronous iterations, the maximum number did quite well correlate with the execution time),
- numbers of snapshots, denoted as "ss",
- and maximum numbers of discarded send requests, denoted as "unsent".

Experimental parameters include the problem size n, the number of MPI processes p, the maximum admissible residual $r^* = 10^{-6}$, and the maximum number of active MPI reception requests concurrently handled for each neighbor of a process, denoted as "rpn" (requests per neighbor).

7.1.2 Worse-cases scalability

Although it is well established for additive Schwarz domain decomposition methods that random asynchronous iterations run faster than their synchronous counterparts, we show here, for record, interesting scalability results obtained on high numbers of processor cores. Regardless the iterations schemes, the two-dimensional grid partitioning (Figure 7.1) is a typical case of poorly scalable parallel configuration where, despite an increasing number of sub-domains, interface sizes far less decrease (while sub-domain size is divided by q, interface size is divided by $\sqrt{q}$), which yields strong impacts of communication times on the expected performance of the algorithms. This allows us to observe asynchronous iterations behavior in such highly communication-dependent configurations, even on homogeneous computational platforms. Table 7.1 illustrates typical execution times that we measured on the Bullx B510 cluster, while varying both the number of sub-domains and the size of the discrete problem. Figure 7.3 provides a corresponding quick overview. We

			Sync. iter.			Async. iter.		
p	$\sqrt[3]{n}$	n/p	$r \times 10^7$	wt	k	$r \times 10^7$	wt	k
256	170	19191	8.32	56	70099	7.07	37	85365
512	175	10468	8.33	48	106930	6.18	33	138214
1024	180	5695	8.33	66	184614	6.85	33	245365
2048	185	3092	8.46	126	443067	8.45	50	626641
5600	185	1131	8.88	150	479154	8.64	28	817913

Table 7.1: Worse-cases scalability of asynchronous iterations.

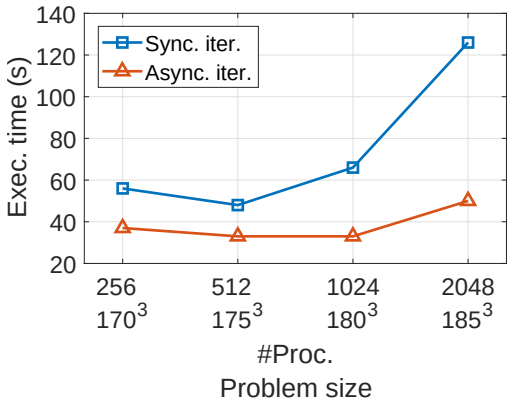


Figure 7.3: Worse-cases scalability of asynchronous iterations.

see, indeed, that, from 512 processes, despite a decreasing sub-domain size, execution time increases for both synchronous and asynchronous iterations. Nonetheless, the latter clearly features far less sensitivity to communication delays, allowing a speedup which increased from 1.45 (for 512 processes) to 2.52 (for 2048 processes), over the former. More importantly, with a fixed problem size $n = 185^3$, synchronous iterations did not successfully scale to 5600 processes, while asynchronous iterations still ran almost twice faster than with 2048 processes, reaching a speedup of 5.36 over the former.

7.1.3 Point-to-point communication

We now focus on items discussed about point-to-point communication management in Jack2. Table 7.2 reports some results illustrating the general trend observed on the SGI ICE X cluster, where corresponding synchronous iterations featured $r = 8.33 \times 10^{-7}$, wt $= 255$ and $k = 128504$. With only one active reception request per neighbor, the *MPI_Isend*-based implementation of Jack2 made the solver procedure exceed the limit wall-clock time set to 4 hours. While looking at the corresponding results for the *MPI_Issend*-based implementation, we see that the number of discarded message sending requests was tremendously high, revealing that up to 48% of iterative updates remained unsent. As expected, with a non-buffering routine though (*MPI_Issend*), actual sending buffers were continuously updated even if requests were dismissed, which yielded less delays on dependencies. It followed that the execution time did not depend at all on the MPI sending request rate. Note however that, since the computational platform is homogeneous, processes performed their iterations approximately at the same pace, which required no more than two concurrently active reception requests per neighbor for making the *MPI_Isend*-based implementation fully effective. It even ran slightly faster than the *MPI_Issend*-based one, which suggests better performance from buffering the

	MPI_Issend				MPI_Isend			
rpn	$r \times 10^7$	wt	k	unsent	$r \times 10^7$	wt	k	unsent
1	6.21	239	146440	84579	-	>4h	-	-
2	6.61	236	144723	747	6.38	228	140157	356
3	6.67	236	144243	533	5.90	228	140327	341
4	6.28	236	144325	543	6.31	228	139779	349
5	6.46	236	144392	532	6.03	229	140035	362
20	6.61	237	144256	626	5.91	229	140624	344
50	6.25	241	144958	586	6.63	232	141332	350

Table 7.2: Performance of *MPI_Issend*-based and *MPI_Isend*-based implementations of Jack2, with $p = 192$, $n = 180^3$.

communicated data.

7.1.4 Convergence detection

A third set of experiments particularly addresses convergence detection matters and the efficiency of the corresponding collective routines. We consider here, on one hand, the method *SnapReduce* from Listing 6.7, which implements snapshot-based supervised termination protocols in Chapter 5, and on the other hand, the programming pattern from Listing 6.6, which makes direct use of the general methods *Snapshot* (Algorithm 15) and *AllReduce* (based on leader election [74, Section 4.4.3]). On practical aspects, few remarks follow about the proposed AIS protocols. 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 about the content of the marker. Furthermore, AIS2 turns out to be a particular instance of the non-FIFO AIS protocol 5 (NFAIS5), when one would consider $\eta = 0$. Secondly, the non-FIFO AIS protocol 4 (NFAIS4) is a generalization of NFAIS5, based on a behavior not likely to occur in most mono-site high performance computing platforms. At last, the non-FIFO AIS protocol 3 (NFAIS3) is designed for very specific circumstances where markers exchange is preferably to be avoided. We therefore focus here on NFAIS1, NFAIS2 and NFAIS5, which should behave very similarly to the other AIS protocols.

Table 7.3 highlights comparative performance of the two asynchronous scheme programming patterns on the SGI ICE X cluster. Contrarily to the discussion about the scalability results, the termination protocols were less sensitive here to the partitioning scheme, as mostly, differences in termination delay depend on local convergence monitoring. On such an homogeneous platform, the supplementary reduction

	Sync.			Snapshot+AllReduce (SSAR)			
p	$r \times 10^7$	wt	k	$r \times 10^7$	wt	k	ss
168	8.33	701	281916	5.03	536	346226	12456
240	8.31	516	284118	6.18	378	366231	11033
360	8.33	382	287557	5.72	250	355394	7972
480	8.32	302	289933	5.40	202	406611	8275
600	8.32	278	292163	5.23	168	432390	6856

	SnapReduce SB96 [35]				SnapReduce NFAIS1			
p	$r \times 10^7$	wt	k	ss	$r \times 10^7$	wt	k	ss
168	6.55	641	319703	140	5.94	650	323569	2620
240	6.52	462	342476	118	6.56	465	344438	2528
360	6.71	310	335008	94	5.11	316	339933	2001
480	6.43	249	380524	101	5.22	254	388084	2162
600	6.55	207	404544	82	5.11	213	417004	2186

	SnapReduce NFAIS2				SnapReduce NFAIS5			
p	$r \times 10^7$	wt	k	ss	$r \times 10^7$	wt	k	ss
168	5.90	644	320335	354	6.54	640	319349	299
240	5.95	466	344229	305	6.42	463	343295	256
360	5.33	314	339274	248	5.19	314	339204	300
480	5.62	252	384967	276	6.63	250	383745	219
600	5.13	210	411327	259	6.06	209	410621	255

Table 7.3: Performance of convergence detection procedures, with $n = 185^3$.

operation in the coordination phase of the SB96 protocol is fast enough to allow approximately same execution times than simulations making use of coordination-free AIS protocols. We therefore, as expected, only confirm here the accuracy of the snapshot-based supervised termination approach. It is even noticeable that the final residuals were quite stable, especially for SB96 ($6 \times 10^{-7} < r < 7 \times 10^{-7}$) and NFAIS2 ($5 \times 10^{-7} < r < 6 \times 10^{-7}$), which strengthens the practical reliability of such protocols.

Nonetheless, a quite unexpected efficiency result came out, as clearly depicted on Figure 7.4. While the local convergence monitoring led to fewer numbers of snapshots than the SSAR-based pattern, this was not relevant for the solver to execute faster.

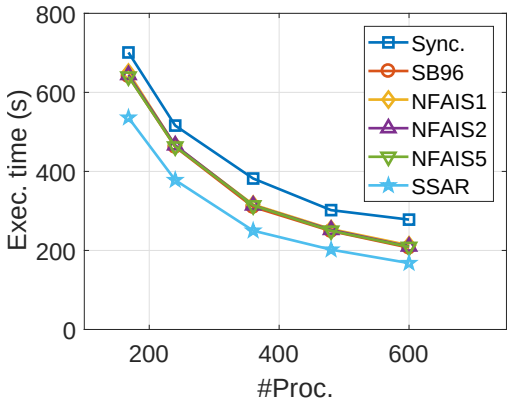


Figure 7.4: Performance of convergence detection procedures, with $n = 185^3$.

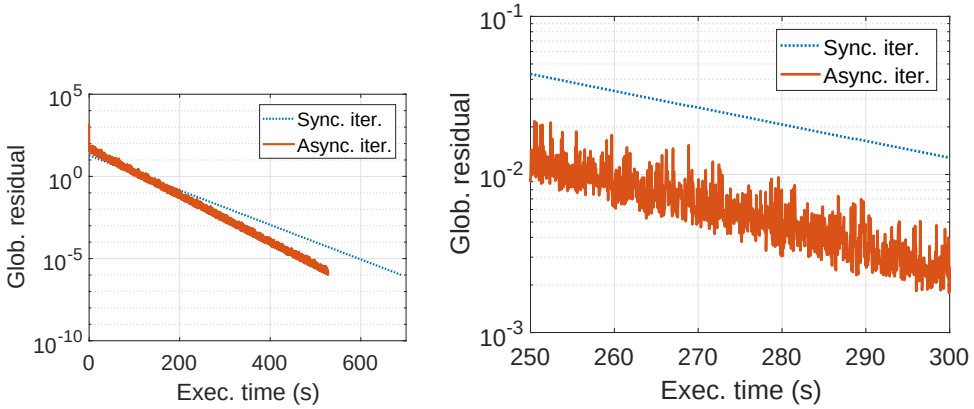


Figure 7.5: Monitored convergence of asynchronous iterations, with $p = 168$, $n = 185^3$.

Instead, we notice that the monitoring-based programming pattern yielded longer iterations, since related results feature lower numbers of iterations despite far higher execution times. This came from the fact that, in the SSAR approach, local residual norm is not required to be computed at every iteration. It therefore turned out that the communication overhead costs introduced by regular and frequent snapshot executions were negligible compared to the impact of the systematic evaluation of a local convergence criterion. Besides that, the SSAR-based pattern made it possible to analyze actual global convergence behavior of asynchronous iterations, just like for synchronous ones, as shown in Figure 7.5.

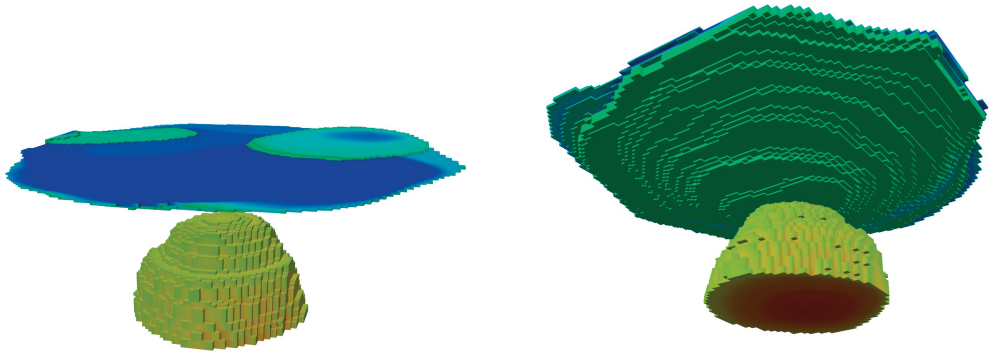


Figure 7.6: Close up view of the salt dome geometry.

7.2 Asynchronous space domain decomposition

7.2.1 Experimental setting

To illustrate our substructures-based iterative model, we consider test cases related to the study of the Chicxulub impact crater, located underneath the town of Chicxulub, in the southwest of Mexico on the Yucatán Peninsula. One of the techniques used to study this geological formation is the gravity method, which allows to quantify differences in the Earth's gravitational field at specific locations. Detected anomalies of the gravitational field allow scientists to draw conclusions about the geological structures, and to determine the depth, density and geometry of the gravitational anomaly sources. For these numerical experiments, data acquisition have been performed on a physical domain covering the Yucatán Peninsula with an area of 250 km $\times$ 250 km and reaching 15 km in depth. From these measurements, the localization and the concentration of the salt-dome for instance can be determined with strong accuracy. The value of the density ρ (see equation (7.1)) obtained with data acquisition is shown in Figure 7.6. Using the data thus collected, one of the techniques to analyze geological formation is the method of potential, which describes the subsurface and which consists in determining mass density distribution correlated with seismic velocities. For completeness, we described this in more detail. The gravity force is the resultant of the gravitational attraction and the centrifugal force. The gravitational potential of a spherical density distribution is equal to

$$\Phi(r) = G\frac{m}{r},$$

with m being the mass of the object, r being the distance to the object and G being the universal gravity constant given by:

$$G = 6.672 \times 10^{-11} m^3 kg^{-1} s^{-2}.$$

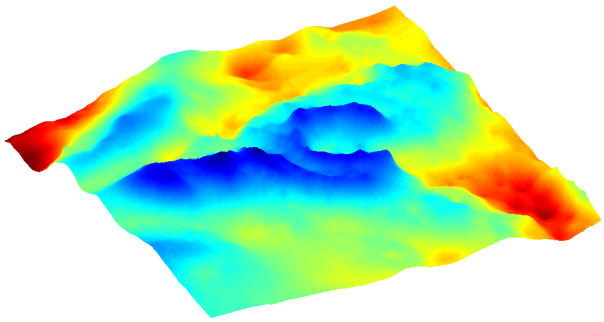


Figure 7.7: Measurements of the gravitational potential in the Yucatán Peninsula.

The gravitational potential at a given position x initiated by an arbitrary density distribution ρ is given by:

$$\Phi(x) = G \int (\rho(x')/||x - x'||)\, dx', \tag{7.1}$$

where x' represents one point position within the density distribution. Here, we consider only the regional scale of the gravimetry equations, therefore we do not take into account the effects of the centrifugal force. As a consequence, the gravitational potential Φ of a density anomaly distribution $\delta\rho$ is given as the solution of the Poisson equation

$$\Delta\Phi = -4\pi G\delta\rho,$$

where $\delta\rho$ was obtained from field measurements. Measurements of the gravitational potential in Yucatán Peninsula are shown in Figure 7.7 and they illustrate the strong variation of the potential in the center of this area.

The numerical solution of the gravitational potential equation required for this study was done on a parallelepiped geometric domain of dimensions 750 km $\times$ 750 km $\times$ 45 km. At the center of this geometry, the volume of dimensions 250 km $\times$ 250 km $\times$ 15 km is meshed with small regular-size finite elements, and outside this volume, the mesh is composed of larger finite elements. One example of the meshes used for the simulation is shown in Figure 7.8, along with a partitioning into 64 sub-domains. The finite element solution on a plane at 1.5 km below the surface of the domain is shown in Figure 7.9, illustrating that, indeed, there is a variation of the gravity in the region which is consistent with the impact of a large object.

For the parallel methods performance, we considered two meshing consisting of, respectively, 71407 and 525213 degrees of freedom (DOF), along with 60000 and 480000 finite elements. Experiments were led on an heterogeneous cluster of 3 sets of 4 nodes (12 nodes) on a star-shaped 10Mb/s Ethernet network. Each node of a first set consists of an Intel Xeon E5410 CPU with 8 cores, at 2.33 GHz, and 8 GB RAM. A second set consists of Intel Core i7 CPUs (8 cores), at 2.80 GHz, and 8 GB

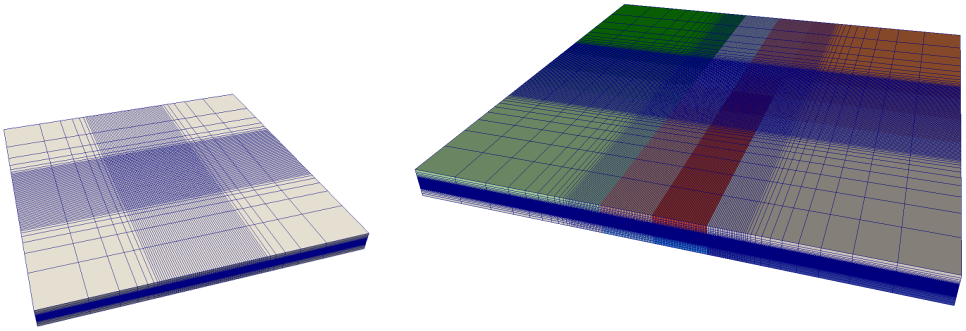


Figure 7.8: Finite element mesh (first) and partitioning (second).

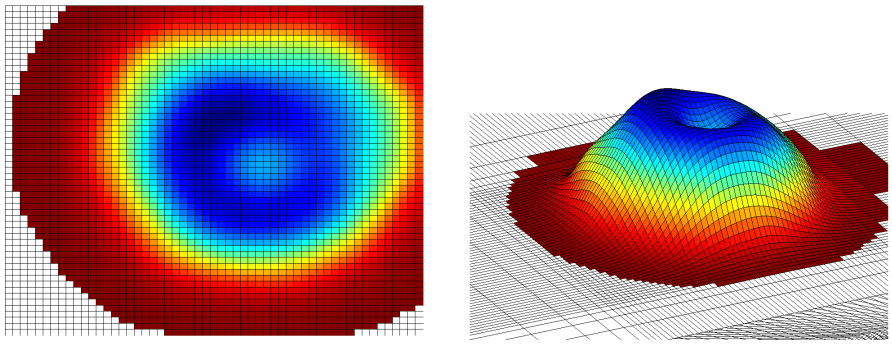


Figure 7.9: Finite element solution in the center of the Yucatán Peninsula, 1.5 km below the surface.

RAMs. The nodes of the third set consist of Intel Xeon E5-2609 CPU with 24 cores, at 2.10 GHz, and 16 GB RAM. This provides a total of 160 CPU cores. The MPI library OpenMPI has been used as communication middleware. Parameters include the problem size n (number of DOF), the number of sub-structures p (which equals the number of MPI processes) and the maximum admissible residual set to10^{-6}. Subsequent results feature:

- execution times in seconds, denoted as "wt" (wall-clock time),
- and efficiencies (in percentage) over sequential execution, denoted as "eff".

7.2.2 Performance results

We first give an overview of the computational cost in Table 7.4 which shows results from sequential execution of a classical Jacobi iterative method. For parallel execution, we considered a row-band decomposition of the Jacobi iterative mapping, and the sparsity of the matrix was taken into account so that communication is avoided

n	k	wt
71407	6234	159.5
525213	13791	2713.0

Table 7.4: Reference sequential results for space domain sub-structuring test cases.

between processes which do not share any non-zero coefficient. Such a scheme is compared to the substructures-based Jacobi splitting where communication is more strictly related to interface nodes only. We see the clear performance gain in both Table 7.5 and Table 7.6, which was exacerbated by the flattened shape of the domain geometry. Still, on scalability aspects of these two methods, increasing the number of processor cores hardly improved the execution time, especially for the small problem (Table 7.5). On the bigger case (Table 7.6), where the workload per process becomes quite more important than the interface size (which however increases too), the performance of the row-band Jacobi got outright decreasing, while the sub-structuring method kept slightly scaling. Finally, as expected, introducing asynchronous iterations into the sub-structuring method led us to even better scalability, and it is rather noticeable to have the efficiency possibly increase (from 48 to 64 processes, 38.16% to 42.01%).

	Row-band Jacobi		Sub-struct. Jacobi		Sub-struct. Async.	
p	wt	eff	wt	eff	wt	eff
16	133.1	7.49	22.9	43.58	32.7	30.52
48	151.7	2.19	22.9	14.48	21.9	15.19
64	149.2	1.67	22.6	11.02	16.8	14.80

Table 7.5: Performance of asynchronous sub-structuring, with $n = 71407$.

	Row-band Jacobi		Sub-struct. Jacobi		Sub-struct. Async.	
p	wt	eff	wt	eff	wt	eff
16	7399.3	2.29	273.7	61.95	243.8	69.55
48	10769.9	0.52	191.9	29.46	148.1	38.16
64	11921.9	0.36	164.6	25.76	100.9	42.01

Table 7.6: Performance of asynchronous sub-structuring, with $n = 525213$.

7.3 Asynchronous time domain decomposition

7.3.1 Experimental setting

For experiments about asynchronous time domain decomposition, we consider a decentralized implementation of the Parareal, where the coarse propagator is distributed upon the N processes, as shown by Algorithm 16 (also see, e.g., [78]). The main advantage is that this avoids any risk of bottleneck when the number of

Algorithm 16 Decentralized Parareal

$n :=$ process rank
$N :=$ number of processes
$\lambda_0^0 := u_0$
for all $i \in \{1, \ldots, n+1\}$ **do**
  $w_i^0 := G(\lambda_{i-1}^0)$
  $\lambda_i^0 := w_i^0$
end for
$k := 0$
repeat
  if $n \leq k$ **then**
    $\lambda_n^{k+1} := \lambda_n^k$
  else
    Request reception of λ_n^{k+1} from process $n-1$
  end if
  $v_{n+1}^k := F(\lambda_n^k)$
  if $n > k$ **then**
    Wait for reception of λ_n^{k+1}
  end if
  $w_{n+1}^{k+1} := G(\lambda_n^{k+1})$
  $\lambda_{n+1}^{k+1} := w_{n+1}^{k+1} + v_{n+1}^k - w_{n+1}^k$
  if $k \leq n < N-1$ **then**
    Request sending of λ_{n+1}^{k+1} to process $n+1$
  end if
  $k := k+1$
until $\|\lambda_{n+1}^k - \lambda_{n+1}^{k-1}\| \simeq 0$, $\forall n \in \{0, \ldots, N-1\}$

time sub-domains increases. For performance comparison, the asynchronous version follows the same decentralized scheme, which also provides a better rate of communication and then accelerates its convergence.

The experimental case consists in simulating an apartment gradually cooled by an air conditioner. The apartment is 10 meters length, 6.4 meters width and 3.4 meters height. It is composed of an American kitchen (a bar, two stools) and a living room

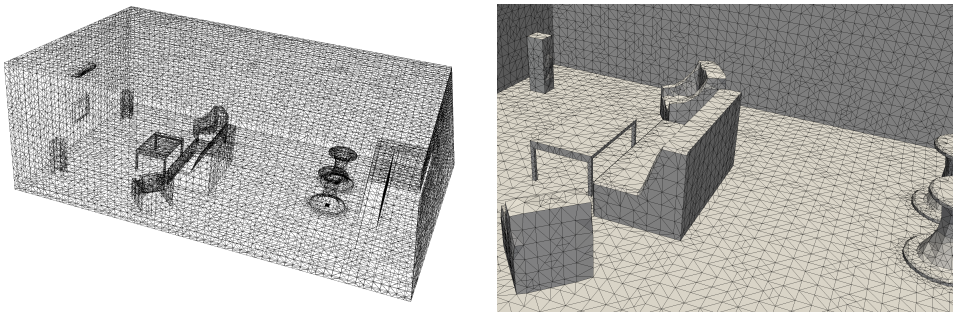


Figure 7.10: Finite element mesh for the Parareal test case.

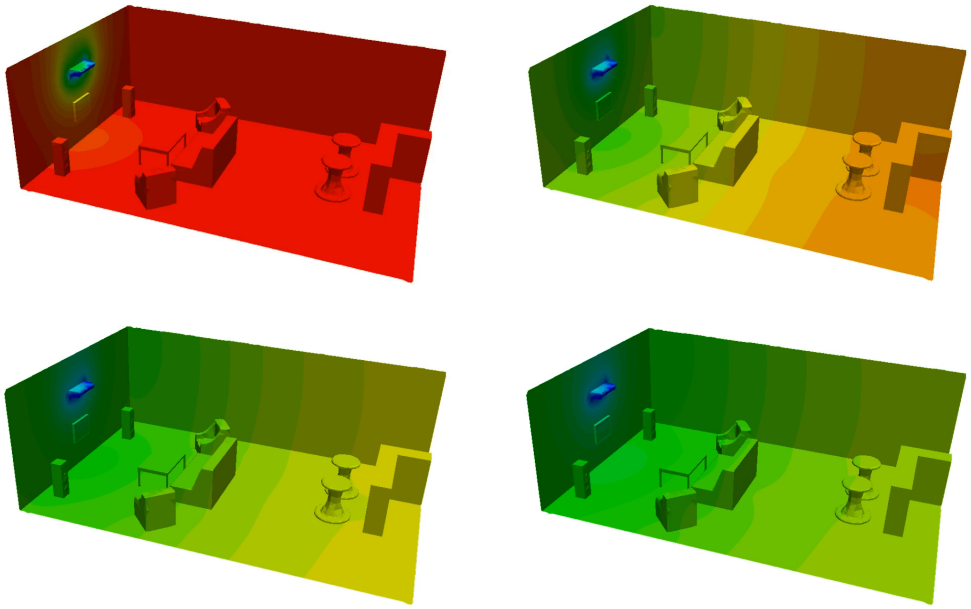


Figure 7.11: Simulation of heat distribution: times $t = 1$, $t = 10$, $t = 20$ and $t = 30$.

(one sofa, two chairs, one television and one air conditioner). The Computer Aided Design (CAD) model was performed with the *CATIA V5* proprietary software from © Dassault Systèmes, then the air volume inside the apartment was discretized with finite elements. Figure 7.10 shows the corresponding finite element mesh composed of 171478 tetrahedra and 33796 nodes. The initial temperature is set to 86.0 degrees Fahrenheit (30 degrees Celsius), and the air cooling system continuously diffuses a gentle breeze at 73.4 degrees Fahrenheit (23 degrees Celsius). Boundary conditions are in Robin form on the floor and walls, in Neumann form on objects inside the room, and in Dirichlet form on the air conditioner. We consider trapezoidal rule and backward Euler discretization, respectively for the fine and the coarse propagators, and with respective time steps $\delta t = 0.002$ and $\Delta T = 0.2$ ($\Delta T/\delta t = 100$). Figure 7.11 shows the evolution of the temperature over time.

Experiments have been led on an SGI Altix ICE supercomputer of 68 nodes on a QDR Infiniband network (40 Gb/s), provided by CentraleSupélec, Université Paris-Saclay, France. Each node consists of two Intel Xeon X5650 CPUs with 6 cores (12 cores per node) at 2.66 GHz, and 21 GB RAM allocated to running jobs. The MPI library SGI-MPT has been loaded as communication middleware. Parameters include the number of time sub-domains N (which equals the number of MPI processes), the total simulated physical time T, and the maximum admissible residual $r^* = 10^{-6} > \max_{1 \leq n \leq N} \|\lambda_n^{k_n+1} - \lambda_n^{k_n}\|$. Subsequent results feature:

- residuals $r := \max_{1 \leq n \leq N} \|\widetilde{\lambda}_n - \lambda_n^*\|$, where $\widetilde{\lambda}$ is the returned solution, and λ^*, the solution from the fine sequential integration,
- execution times in seconds, denoted as "wt" (wall-clock time),
- and numbers of iterations $k := \max_{1 \leq n \leq N}\{k_n\}$, where k_n is the number of iterations on the n-th process.

7.3.2 Performance results

With a constant workload per time sub-domain (100 fine time steps), several executions were conducted for various number of processes. We therefore observed performance evolution while the simulated physical time was increasing. Table 7.7 reports average results. In spite of always approximately twice more iterations

		Parareal sync. iter.			Parareal async. iter.		
N	T	r	wt	k	r	wt	k
16	3.2	1.49E-07	280	10	5.01E-08	342	24
24	4.8	1.86E-07	420	14	1.06E-07	502	34
32	6.4	3.75E-08	691	20	1.71E-08	644	44
48	9.6	4.95E-11	1169	32	3.33E-11	922	66
64	12.8	3.75E-08	1488	44	2.79E-10	1255	92
90	18.0	4.75E-11	2676	64	4.78E-11	1938	149

Table 7.7: Performance of Parareal asynchronous iterations, with $\delta t = 0.002$, $\Delta T = 0.2$.

performed (see Figure 7.12, right), the asynchronous iterative scheme successfully converged faster than the classical Parareal for 32 and more processes. We was then able, for instance, to save around 12 minutes over 44, at 90 time sub-domains.

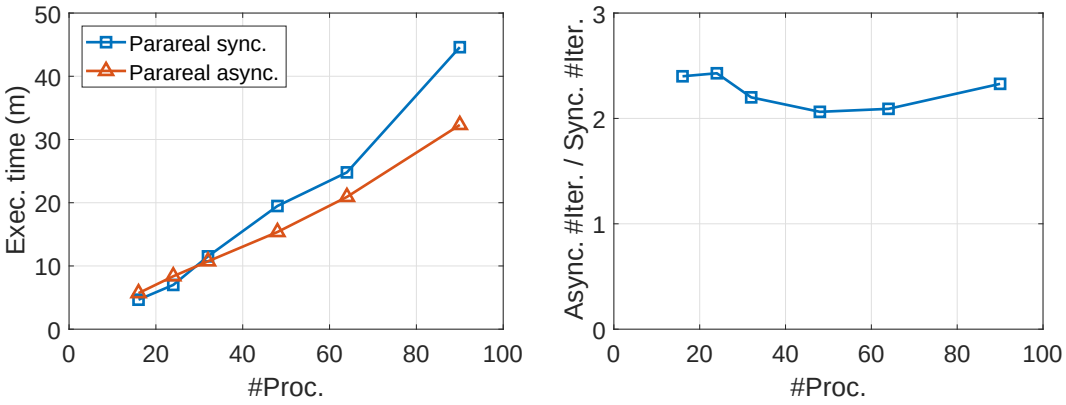


Figure 7.12: Performance of Parareal asynchronous iterations, with $\delta t = 0.002$, $\Delta T = 0.2$.

Chapter 8

Conclusion

8.1 Theoretical aspects

There is undoubtedly an interest in HPC about the ability to run parallel iterations with highly flexible synchronization requirements. So far, very few general computational models successfully design asynchronous iterations, however an extensive literature is available for their convergence analysis. When therefore it comes out to apply or extend such models to some specific parallel framework, an accurate overview of applicable convergence results would constitute a non-negligible asset. We thus firstly provide in this report a quite complete theoretical tool as a quick reference guide for establishing the convergence of new asynchronous iterative methods, based on existing general results.

While studying the application of asynchronous iterations to the space domain decomposition framework, we saw for instance that analyzing the convergence of a primary basic asynchronous iterations model could be done through the analysis of a secondary corresponding model of asynchronous iterations with memory. More precisely, given a linear problem

$$Ax = b,$$

and a splitting

$$A = M - N,$$

an induced asynchronous iterative domain decomposition mapping $f(x)$ (overlapping or not) could be equivalently analyzed through another asynchronous iterative mapping $g(y^1, \ldots, y^m)$ defined on an interface-based Schur complement in A. Such relations therefore provide much more flexibility in the design and the convergence analysis of new parallel computational models based on the various domain decomposition methods. This came out from our theoretical investigation of both a partitioning-based additive Schwarz parallel scheme and a domain sub-structuring approach where a sub-part of the interface, containing off-diagonal coefficients of the corresponding sub-matrix, is disassembled. Using then the framework of asynchronous iterations with memory $g(y^1, \ldots, y^m)$ on the Schur complement inversion

problem, we successfully developed a new asynchronous iterative sub-structuring method which is convergent under the same conditions than the asynchronous sub-structuring approach recently proposed in [47], but is however far more efficient for parallel computation.

While asynchronous iterations are usually applied in parallel computing on space domain decomposition, we addressed in a part of this thesis their application to time domain decomposition for the discretization of time-dependent PDEs

$$f(u(t), t) = 0,$$

focusing precisely on the Parareal iterative scheme. By considering a totally distributed iteration model, our proposition was to introduce asynchronism at the coarse propagator level, which is the sequential part of the algorithm. Expectedly, it resulted that for this asynchronous parallel-in-time method derived from the Parareal scheme, convergence is guaranteed under conditions slightly more restrictive, but still applicable. Further, and quite surprisingly, these conditions for asynchronous convergence asymptotically coincide with their synchronous counterpart as the number of time sub-domains grows. It follows from our convergence analysis that different iterative schemes could be intertwined to design an asynchronous parallel-in-time method based on multiple operators. Theoretical performance analysis confirmed a possible performance gain from asynchronous iterations, however the Parareal scheme features an inherent speedup limit due to an unavoidable sequential propagation of some computed values, regardless the inter-process synchronization time.

8.2 Implementation aspects

On implementation aspects, the problem of detecting the convergence of asynchronous iterations has been tackled in many various ways. Nonetheless, very few existing termination methods are based on the computation of a global residual error. Furthermore, mostly, more or less intrusive approaches were investigated, turning out to be quite complicated, without necessarily providing a minimal termination delay. For instance, most prominent asynchronous convergence tests feature at least two reduction operations, while we managed here to achieve effective detection with only one, and at lower communication costs. We proposed in this thesis several asynchronous iterations termination methods based on global residual error, under various communication models. First-in-first-out (FIFO) environments allow us to rely on simple protocols using control messages devised from the well-known Chandy–Lamport distributed snapshot. Considering computational performance, FIFO communication is essential at least for computation messages. We therefore managed to exhibit another method 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

only when other non-FIFO methods are not applicable. To therefore characterize a general non-FIFO model, we assume on every channel (in one direction) a maximum number of messages that can be crossed by any given other message. Arbitrary non-FIFO delivering actually corresponds to the asymptotic case where this maximum number exceeds the total amount of messages emitted on the channel. We showed here how approximations could be used to avoid including computation data into control messages, which constitutes an improvement of existing residual-based termination methods, in terms of communication overhead costs. The reliability of these approximations have been formally established, providing a practical way of accurately setting the convergence residual threshold.

Taking full advantage of asynchronous iterations is still a challenging computational matter which possibly requires more suitable parallel programming patterns. By designing specialized libraries, researchers therefore aim to provide efficient asynchronous iterations environments requiring lowest possible implementation costs for users. We addressed during this thesis the development of Jack2, an MPI-based communication library for distributed iterative computing. It features an API quite close to the MPI specification, which makes it easy to upgrade existing MPI-based applications toward the Jack2 programming framework, wherein a handful set of tools is already available for the experimental study of asynchronous iterations. Key results from our investigation are twofold. First, regarding the convergence detection issue arising from random execution of asynchronous iterations, existing distributed approaches involve some local convergence criterion, as researchers rightly argued so far that it is ineffective to check global convergence at early stages of the computation where local convergence is not even somehow persistent on any of the processes. This perfectly matches the classical parallel programming pattern where a local convergence residual is evaluated at each iteration. By considering here an exact general snapshot-based approach to directly evaluate consistent global residuals, we showed that systematic irrelevant local residual computation could be, on the contrary, far more time consuming than a distributed protocol uselessly executed concurrently with the main computation phase. Such an important practical aspect seems to have been missed till now. While focusing, on another hand, on delays minimization in the iterative process, it came out from obtained experimental execution times that best performances could be reached by suitably combining outgoing message buffering with the MPI synchronous communication mode. This is currently under consideration for further experiments.

8.3 Prospects and future work

The innovative nature of this research piece of work is undoubtedly widening, a little more, the application field of asynchronous iterations. Besides possible further theoretical analysis, fast and cheap experimental studies on much more scientific problems are now feasible with our non-intrusive MPI-based C++ programming library,

which could thus be accordingly improved at the same time for more applicability and efficiency. Further investigation about non-blocking global synchronization for distributed convergence detection is needed as well, and could even be an interesting approach for synchronous iterations.

Domain decomposition methods constitute a top prominent research field in parallel numerical simulation. While overlapping decomposition was successfully investigated for the application of asynchronous iterations, we have highlighted here an analysis framework for primal and dual sub-structuring methods, respectively on space and time domains. In the case of space domains, for instance, we have proposed two practical primal approaches, while we fully investigated only one of them. A direct extension of our results can thus be obtained by applying same analysis principles to the second approach. More generally, this would lead us to consider various other primal sub-structuring frameworks, as well as dual and mixed approaches.

Still, preconditioning techniques are one important missing aspect in this dissertation, nonetheless, we are currently achieving some preliminary steps toward asynchronous application of coarse space correction, and more generally, multigrid methods. For now, being able to experiment the performance of our asynchronous space domain decomposition methods against current fastest synchronous solutions, on more than five thousand distributed processor cores, is a very next step to be taken, as we believe that this could reveal new surprising results.

Appendix A

Résumé

A.1 Contexte et motivation

La loi d'Amdahl [1] établit une limite sur l'accélération que l'on peut espérer de la parallélisation d'une tâche, indépendamment du nombre d'unités de traitement (processeur) utilisés. Cela est dû à une portion non-parallélisable, liée à la gestion des données, inévitable pour garantir l'équivalence entre des exécutions monoprocesseur et multiprocesseur. Cette accélération maximale possible est donnée par:

$$s(p, \alpha) = \frac{1}{\alpha + \frac{1-\alpha}{p}} < \frac{1}{\alpha},$$

avec $p \geq 1$ étant le nombre de processeurs et, $\alpha < 1$, la proportion non-parallélisable. Par ailleurs, atteindre en pratique cette accélération maximale requiert un mécanisme d'équilibrage de charge effectif afin d'obtenir la division exacte

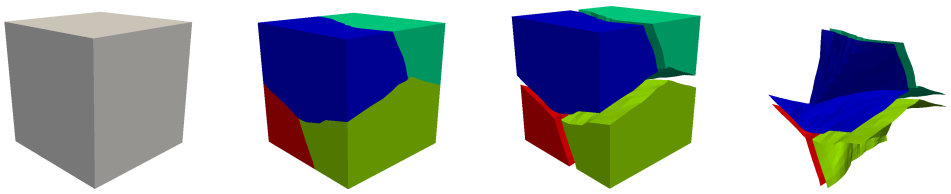
$$\frac{1-\alpha}{p}$$

de la partie strictement parallèle. Enfin, dans l'éventualité alors où un très haut degré de parallélisation est atteint, se pose également la problématique de la tolérance aux pannes, ce qui introduit potentiellement une couche supplémentaire d'utilisation des ressources de traitement.

Nous nous intéressons, dans cette étude, à la simulation numérique parallèle de phénomènes modélisés à l'aide d'équations aux dérivées partielles

$$\delta(u(s,t), s, t) = 0, \qquad t \in [0, T],\ T \in \mathbb{R}, \quad s \in \Omega \subset \mathbb{R}^3,$$

où u est une fonction dans $\mathbb{R}$ ou $\mathbb{C}$, décrivant l'état du phénomène dans le domaine spatial Ω et le domaine temporel $[0, T]$. Les méthodes de décomposition de domaine [2] permettent de traiter séparément, en parallèle, différentes parties d'un même domaine, tout en assurant une cohérence globale qui se traduit par une



(a) Domaine (b) Partition (c) Sous-domaines (d) Interface

Figure A.1: Décomposition de domaine.

cohérence de part et d'autre des interfaces de jointure entre ces sous-domaines. Figure A.1 en donne une illustration empirique. Ces méthodes maximisent alors la parallélisation des solveurs numériques, en ce sens qu'elles rendent parallélisables les éventuelles opérations d'entrée-sortie et de pré ou post traitement. Elles ont par ailleurs l'intérêt d'être naturellement adaptées à une conception de système complexe par assemblage de composants. Néanmoins, les approches de décomposition les plus efficaces à ce jour font intervenir des méthodes itératives pour résoudre un problème dérivé posé aux interfaces, ce qui introduit une séquence d'étapes parallèles, et donc une portion strictement séquentielle de gestion de données correspondant aux synchronisations bloquantes inter-processeurs. Ce séquencement déterministe rend également toute défaillance d'un processeur fatale au calcul en cours d'exécution.

Ainsi, la possibilité de séquences aléatoires sur chaque sous-domaine, qui convergeraient néanmoins toujours vers une même solution globale (très sensiblement), serait la clé pour aboutir à des solveurs à la fois naturellement résilients aux défaillances (temporaires) et potentiellement parallélisables à l'infini. Les méthodes itératives asynchrones [16], introduites contemporainement à la loi d'Amdahl, modélisent cette classe d'algorithmes itératifs sans portion non parallélisable. L'objectif de cette thèse est d'étudier l'application de la théorie des itérations asynchrones aux méthodes de décomposition de domaine, incluant l'analyse des conditions de convergence, la détection de l'état de convergence d'un calcul en cours, et l'implémentation rapide et efficace, voire non-intrusive, d'itérations asynchrones.

A.2 Itérations asynchrones

A.2.1 Modèle calculatoire

Soit, à résoudre, une équation

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

x étant un vecteur de n inconnues complexes. Soit M, une matrice inversible, et f, une fonction donnée par

$$f(x) = (I - M^{-1}A)x + M^{-1}b.$$

En remarquant que

$$A = M - (M - A),$$

l'on formule ainsi un problème de recherche de point fixe de f, étant donné que

$$Ax = b \quad \Longleftrightarrow \quad f(x) = x.$$

Une approche itérative classique pour trouver ce point fixe x^* consiste à générer une suite $\{x^k\}_{k\in\mathbb{N}}$ telle que

$$x^{k+1} = f(x^k), \tag{A.1}$$

et telle que, pour n'importe quel vecteur initial x^0 donné, elle converge vers une solution unique, i.e,

$$\lim_{k\to+\infty} x^k = x^*.$$

Dans un contexte parallèle avec p processeurs, $p \leq n$, l'on considère une décomposition de vecteur de la forme

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}, \qquad f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_p(x) \end{bmatrix},$$

ce qui conduit aux itérations parallèles données par:

$$x_i^{k+1} = f_i(x_1^k, \ldots, x_p^k), \qquad \forall i \in \{1, \ldots, p\}.$$

L'exemple d'exécution suivant, correspondant à Figure A.2, illustre les portions séquentielles marquées par des temps d'attente dus aux délais de communication:

$x_1^1 := f_1(x_1^0, x_2^0)$	$x_2^1 := f_2(x_1^0, x_2^0)$
attente	attente
$x_1^2 := f_1(x_1^1, x_2^1)$	$x_2^2 := f_2(x_1^1, x_2^1)$
$x_1^3 := f_1(x_1^2, x_2^2)$	attente
attente	$x_2^3 := f_2(x_1^2, x_2^2)$
attente	$x_2^4 := f_2(x_1^3, x_2^3)$

Un mode d'exécution parallèle désynchronisée, illustré par Figure A.3, correspondrait donc à la séquence itérative suivante, où les temps de communication sont entièrement recouverts par le calcul:

$x_1^1 := f_1(x_1^0, x_2^0)$	$x_2^1 := f_2(x_1^0, x_2^0)$
$x_1^2 := f_1(x_1^1, x_2^0)$	$x_2^2 := f_2(x_1^0, x_2^1)$
$x_1^3 := x_1^2$	$x_2^3 := f_2(x_1^1, x_2^2)$
$x_1^4 := f_1(x_1^3, x_2^2)$	$x_2^4 := f_2(x_1^2, x_2^3)$
$x_1^5 := f_1(x_1^4, x_2^3)$	$x_2^5 := f_2(x_1^2, x_2^4)$

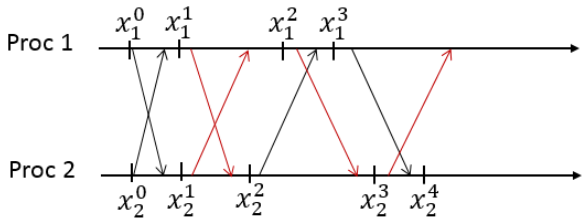


Figure A.2: Exécution parallèle synchrone, avec temps d'attente (en rouge).

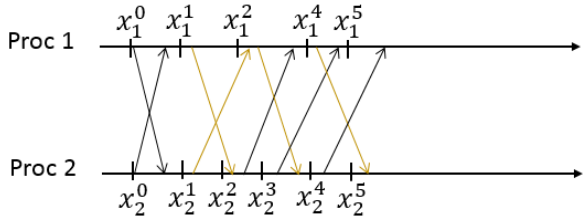


Figure A.3: Exécution parallèle asynchrone, avec recouvrement des retards (en orange).

Ainsi, les méthodes itératives asynchrones génèrent des suites $\{x^k\}_{k\in\mathbb{N}}$ telles que

$$x_i^{k+1} = \begin{cases} f_i\left(x_1^{\tau_1^i(k)}, \ldots, x_p^{\tau_p^i(k)}\right), & \forall i \in P_k, \\ x_i^k, & \forall i \notin P_k, \end{cases} \tag{A.2}$$

avec

$$\tau_j^i(k) \leq k, \qquad P_k \subseteq \{1, \ldots, p\}.$$

Ce modèle est complété par deux hypothèses de non interruption définitive, concernant, respectivement, la mise à jour et la transmission de composante i:

$$\forall i \in \{1, \ldots, p\}, \quad \text{card}\{k \in \mathbb{N} \mid i \in P_k\} = +\infty,$$

$$\forall i, j \in \{1, \ldots, p\}, \quad \lim_{k\to+\infty} \tau_j^i(k) = +\infty.$$

De façon plus générale, l'on peut considérer une fonction quelconque, non nécessairement linéaire,

$$f : E^m \to E, \qquad m \in \mathbb{N}^*,$$

avec E^m étant le produit cartésien de m ensembles E, et un problème de recherche de point fixe x tel que

$$f(x, x, \ldots, x) = x.$$

Le modèle itératif asynchrone correspondant est donné par

$$x_i^{k+1} = \begin{cases} f_i\left(x_1^{\tau_{1,1}^i(k)}, \ldots, x_p^{\tau_{p,1}^i(k)}, \ldots, x_1^{\tau_{1,m}^i(k)}, \ldots, x_p^{\tau_{p,m}^i(k)}\right), & \forall i \in P_k, \\ x_i^k, & \forall i \notin P_k. \end{cases} \tag{A.3}$$

A.2.2 Conditions de convergence

Théorème A.1 (voir, par exemple, Theorem 2.1 dans [53]). *Le modèle itératif synchrone* (A.1) *est convergent si, et seulement si,*

$$\rho(I - M^{-1}A) < 1,$$

$\rho(.)$ *désignant le rayon spectral.*

Théorème A.2 (Chazan & Miranker, 1969 [21]). *Le modèle itératif asynchrone* (A.2) *est convergent si, et seulement si,*

$$\rho(|I - M^{-1}A|) < 1,$$

$|.|$ *désignant la valeur absolue.*

Théorème A.3 (Miellou, 1975 [17]). *Le modèle itératif asynchrone* (A.3)*, avec* $m = 1$*, est convergent s'il existe une matrice* $\mathcal{T} \geq O$ *(non-négative), de rayon spectral* $\rho(\mathcal{T}) < 1$*, telle que*

$$\forall x, y \in E, \quad |f(x) - f(y)| \leq \mathcal{T}|x - y|.$$

Théorème A.4 (El Tarazi, 1982 [19]). *Le modèle itératif asynchrone* (A.3)*, avec* $m = 1$*, est convergent s'il existe un vecteur* $w > 0$ *(positif) et un réel positif* $\alpha < 1$ *tels que*

$$\forall x, y \in E, \quad \|f(x) - f(y)\|_\infty^w \leq \alpha \|x - y\|_\infty^w,$$

$\|.\|_\infty^w$ *désignant la norme infinie pondérée.*

Théorème A.5 (Bertsekas, 1983 [20, 16]). *Le modèle itératif asynchrone* (A.3)*, avec* $m = 1$*, est convergent s'il existe une suite d'ensembles* $\{S^{(t)}\}_{t\in\mathbb{N}}$*, avec*

$$S^{(t)} = S_1^{(t)} \times \cdots \times S_p^{(t)}, \qquad S^{(t+1)} \subset S^{(t)}, \qquad \lim_{t\to+\infty} S^{(t)} = \{x^*\},$$

et telle que

$$\forall x \in S^{(t)}, \quad f(x) \in S^{(t+1)}, \qquad x^0 \in S^{(0)}.$$

Théorème A.6 (Baudet, 1978 [18]). *Le modèle itératif asynchrone* (A.3) *est convergent s'il existe une matrice* $\mathcal{T} \geq O$*, de rayon spectral* $\rho(\mathcal{T}) < 1$*, telle que*

$$\forall X, Y \in E^m, \quad |f(X) - f(Y)| \leq \mathcal{T} \max(|x^{(1)} - y^{(1)}|, \ldots, |x^{(m)} - y^{(m)}|),$$

avec

$$X = (x^{(1)}, \ldots, x^{(m)}), \quad Y = (y^{(1)}, \ldots, y^{(m)}).$$

Théorème A.7 (El Tarazi, 1982 [19]). *Le modèle itératif asynchrone* (A.3) *est convergent s'il existe un vecteur* $w > 0$ *et un réel positif* $\alpha < 1$ *tels que*

$$\forall X, Y \in E^m, \quad \|f(X) - f(Y)\|_\infty^w \leq \alpha \max\{\|x^{(1)} - y^{(1)}\|_\infty^w, \ldots, \|x^{(m)} - y^{(m)}\|_\infty^w\},$$

avec

$$X = (x^{(1)}, \ldots, x^{(m)}), \quad Y = (y^{(1)}, \ldots, y^{(m)}).$$

A.3 Décomposition asynchrone spatiale

A.3.1 Modèle calculatoire

Soit, à résoudre, une équation

$$\begin{bmatrix} A_{1,1} & O & A_{1,\Gamma} \\ O & A_{2,2} & A_{2,\Gamma} \\ A_{\Gamma,1} & A_{\Gamma,2} & A^{(1)}_{\Gamma,\Gamma} + A^{(2)}_{\Gamma,\Gamma} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_\Gamma \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b^{(1)}_\Gamma + b^{(2)}_\Gamma \end{bmatrix},$$

où x_i, avec $i \in \{1, 2\}$, est un vecteur d'inconnues défini sur un domaine Ω_i, et x_Γ, un vecteur d'inconnues défini sur un domaine interface joignant Ω_1 et Ω_2. En procédant par substitution de soit x_1, soit x_2, un tel système devient, soit

$$\begin{bmatrix} A_{1,1} & A_{1,\Gamma} \\ A_{\Gamma,1} & A^{(1)}_{\Gamma,\Gamma} + A^{(2)}_{\Gamma,\Gamma} - A_{\Gamma,2}A^{-1}_{2,2}A_{2,\Gamma} \end{bmatrix} \begin{bmatrix} x_1 \\ x_\Gamma \end{bmatrix} = \begin{bmatrix} b_1 \\ b^{(1)}_\Gamma + b^{(2)}_\Gamma - A_{\Gamma,2}A^{-1}_{2,2}b_2 \end{bmatrix},$$

soit

$$\begin{bmatrix} A_{2,2} & A_{2,\Gamma} \\ A_{\Gamma,2} & A^{(1)}_{\Gamma,\Gamma} + A^{(2)}_{\Gamma,\Gamma} - A_{\Gamma,1}A^{-1}_{1,1}A_{1,\Gamma} \end{bmatrix} \begin{bmatrix} x_2 \\ x_\Gamma \end{bmatrix} = \begin{bmatrix} b_2 \\ b^{(1)}_\Gamma + b^{(2)}_\Gamma - A_{\Gamma,1}A^{-1}_{1,1}b_1 \end{bmatrix}.$$

L'on suppose alors un contexte de résolution où, de chacun de ces points de vue $i \in \{1, 2\}$, les données liées au domaine Ω_j, avec $j \in \{1, 2\}$ et $j \neq i$, sont inconnues, ce qui conduit à deux différentes nouvelles équations,

$$\begin{bmatrix} A_{1,1} & A_{1,\Gamma} \\ A_{\Gamma,1} & A^{(1)}_{\Gamma,\Gamma} + \Lambda^{(1)}_{\Gamma,\Gamma} \end{bmatrix} \begin{bmatrix} x_1 \\ x^{(1)}_\Gamma \end{bmatrix} = \begin{bmatrix} b_1 \\ b^{(1)}_\Gamma + \lambda^{(1)}_\Gamma \end{bmatrix}$$

et

$$\begin{bmatrix} A_{2,2} & A_{2,\Gamma} \\ A_{\Gamma,2} & A^{(2)}_{\Gamma,\Gamma} + \Lambda^{(2)}_{\Gamma,\Gamma} \end{bmatrix} \begin{bmatrix} x_2 \\ x^{(2)}_\Gamma \end{bmatrix} = \begin{bmatrix} b_2 \\ b^{(2)}_\Gamma + \lambda^{(2)}_\Gamma \end{bmatrix},$$

dont les solutions, quelles que soient $\Lambda^{(1)}_{\Gamma,\Gamma}$ et $\Lambda^{(2)}_{\Gamma,\Gamma}$ choisies, sont celles de l'équation globale si, et seulement si,

$$\begin{cases} x^{(1)}_\Gamma = x^{(2)}_\Gamma, \\ \lambda^{(1)}_\Gamma - \Lambda^{(1)}_{\Gamma,\Gamma}x^{(1)}_\Gamma = -\left(\lambda^{(2)}_\Gamma - \Lambda^{(2)}_{\Gamma,\Gamma}x^{(2)}_\Gamma\right), \end{cases}$$

conditions par lesquelles elles sont rendues cohérentes sur l'interface de jointure entre Ω_1 et Ω_2. En substituant de nouveau x_1 et x_2, nous aboutissons à un problème défini

uniquement sur cette interface, donné par:

$$\begin{cases} \left(A_{\Gamma,\Gamma}^{(1)} - A_{\Gamma,1}A_{1,1}^{-1}A_{1,\Gamma} + \Lambda_{\Gamma,\Gamma}^{(1)}\right) x_\Gamma^{(1)} = b_\Gamma^{(1)} + \lambda_\Gamma^{(1)} - A_{\Gamma,1}A_{1,1}^{-1}b_1, \\ \left(A_{\Gamma,\Gamma}^{(2)} - A_{\Gamma,2}A_{2,2}^{-1}A_{2,\Gamma} + \Lambda_{\Gamma,\Gamma}^{(2)}\right) x_\Gamma^{(2)} = b_\Gamma^{(2)} + \lambda_\Gamma^{(2)} - A_{\Gamma,2}A_{2,2}^{-1}b_2, \\ x_\Gamma^{(1)} = x_\Gamma^{(2)}, \\ \lambda_\Gamma^{(1)} - \Lambda_{\Gamma,\Gamma}^{(1)}x_\Gamma^{(1)} = -\left(\lambda_\Gamma^{(2)} - \Lambda_{\Gamma,\Gamma}^{(2)}x_\Gamma^{(2)}\right). \end{cases}$$

Identifions les compléments de Schur,

$$S_{\Gamma,\Gamma}^{(i)} := A_{\Gamma,\Gamma}^{(i)} - A_{\Gamma,i}A_{i,i}^{-1}A_{i,\Gamma}, \qquad i \in \{1,2\},$$

et désignons:

$$d_\Gamma^{(i)} := b_\Gamma^{(i)} - A_{\Gamma,i}A_{i,i}^{-1}b_i, \qquad i \in \{1,2\}.$$

L'approche dite duale de résolution de ce problème d'interface consiste à éliminer par substitution les inconnues

$$x_\Gamma^{(i)} = \left(S_{\Gamma,\Gamma}^{(i)} + \Lambda_{\Gamma,\Gamma}^{(i)}\right)^{-1}\left(d_\Gamma^{(i)} + \lambda_\Gamma^{(i)}\right), \qquad i \in \{1,2\}.$$

Nous considérerons à l'inverse, pour cette toute première étude de l'application des itérations asynchrones dans un tel cadre de décomposition de domaine, l'approche dite primale où sont plutôt éliminées les inconnues

$$\lambda_\Gamma^{(i)} = \left(S_{\Gamma,\Gamma}^{(i)} + \Lambda_{\Gamma,\Gamma}^{(i)}\right)x_\Gamma^{(i)} - d_\Gamma^{(i)}, \qquad i \in \{1,2\}.$$

Ceci conduit à:

$$\begin{cases} x_\Gamma^{(1)} = x_\Gamma^{(2)}, \\ S_{\Gamma,\Gamma}^{(1)}x_\Gamma^{(1)} + S_{\Gamma,\Gamma}^{(2)}x_\Gamma^{(2)} = d_\Gamma^{(1)} + d_\Gamma^{(2)}, \end{cases}$$

d'où le problème primal d'interface

$$\left(S_{\Gamma,\Gamma}^{(1)} + S_{\Gamma,\Gamma}^{(2)}\right)x_\Gamma = d_\Gamma^{(1)} + d_\Gamma^{(2)},$$

avec

$$x_\Gamma = x_\Gamma^{(1)} = x_\Gamma^{(2)}.$$

Soit donc, pour p domaines joints, un problème primal d'interface

$$\sum_{i=1}^{p} S_{\Gamma,\Gamma}^{(i)}x_\Gamma = \sum_{i=1}^{p} d_\Gamma^{(i)}.$$

Une résolution itérative asynchrone requiert la formulation d'un problème de point fixe à partir d'une matrice M, en considérant une scission (*splitting*) de la forme

$$\sum_{i=1}^{p} S_{\Gamma,\Gamma}^{(i)} = M - \left(M - \sum_{i=1}^{p} S_{\Gamma,\Gamma}^{(i)}\right).$$

Cependant, les formes classiques de scission (Jacobi, Gauss-Seidel, SOR, ...) impliquent au minimum la connaissance de

$$\operatorname{diag} \sum_{i=1}^{p} S_{\Gamma,\Gamma}^{(i)},$$

ce qui n'est pas réalisable en pratique dans un tel contexte de décomposition de domaine où le problème d'interface n'est qu'implicitement posé. Nous proposons donc dans cette thèse une forme de scission avec

$$M = M_{\Gamma,\Gamma} := \sum_{i=1}^{p} M_{\Gamma,\Gamma}^{(i)},$$

déduite implicitement de scissions

$$A_{\Gamma,\Gamma}^{(i)} = M_{\Gamma,\Gamma}^{(i)} - \left(M_{\Gamma,\Gamma}^{(i)} - A_{\Gamma,\Gamma}^{(i)} \right), \qquad i \in \{1, \ldots, p\}.$$

Néanmoins, une résolution par le modèle itératif asynchrone (A.2), avec

$$f(x_\Gamma) = \left(I - \left(\sum_{i=1}^{p} M_{\Gamma,\Gamma}^{(i)} \right)^{-1} \sum_{i=1}^{p} S_{\Gamma,\Gamma}^{(i)} \right) x_\Gamma + \left(\sum_{i=1}^{p} M_{\Gamma,\Gamma}^{(i)} \right)^{-1} \sum_{i=1}^{p} d_\Gamma^{(i)}, \tag{A.4}$$

induirait une décomposition de vecteur, telle qu'on aurait

$$x_\Gamma = \begin{bmatrix} (x_\Gamma)_1 \\ \vdots \\ (x_\Gamma)_p \end{bmatrix}.$$

Nous proposons donc également, à partir de cette nouvelle forme de scission, un tout autre modèle itératif permettant de conserver l'approche initiale de décomposition de domaine, où l'on a

$$x_\Gamma = x_\Gamma^{(1)} = \cdots = x_\Gamma^{(p)}.$$

Ce nouveau modèle est obtenu en considérant sur chaque processeur $i \in \{1, \ldots, p\}$, les relaxations successives du problème interface entier, données par:

$$\begin{aligned} \sum_{j=1}^{p} M_{\Gamma,\Gamma}^{(j)} x_\Gamma^{(i),k+1} &= \left(\sum_{j=1}^{p} M_{\Gamma,\Gamma}^{(j)} - \sum_{j=1}^{p} S_{\Gamma,\Gamma}^{(j)} \right) x_\Gamma^{(i),k} + \sum_{j=1}^{p} d_\Gamma^{(j)} \\ &= \sum_{j=1}^{p} \left(M_{\Gamma,\Gamma}^{(j)} - S_{\Gamma,\Gamma}^{(j)} \right) x_\Gamma^{(i),k} + d_\Gamma^{(j)}, \end{aligned}$$

puis en les entrelaçant en

$$\sum_{j=1}^{p} M_{\Gamma,\Gamma}^{(j)} x_{\Gamma}^{(i),k+1} = \sum_{j=1}^{p} \left(M_{\Gamma,\Gamma}^{(j)} - S_{\Gamma,\Gamma}^{(j)} \right) x_{\Gamma}^{(j),k} + d_{\Gamma}^{(j)},$$

ce qui permet de conserver le calcul de chaque terme j de la somme à droite de l'égalité sur le processeur j associé. De là, on déduit le modèle itératif asynchrone défini par:

$$\begin{aligned} y_{\Gamma}^{(i),k} &= \left(M_{\Gamma,\Gamma}^{(i)} - S_{\Gamma,\Gamma}^{(i)} \right) x_{\Gamma}^{(i),k} + d_{\Gamma}^{(i)}, \quad \forall i \in \{1,\dots,p\}, \\ \sum_{j=1}^{p} M_{\Gamma,\Gamma}^{(j)} x_{\Gamma}^{(i),k+1} &= \begin{cases} \displaystyle\sum_{j=1}^{p} y_{\Gamma}^{(j),\tau_j^i(k)}, & \forall i \in P_k, \\ \displaystyle\sum_{j=1}^{p} M_{\Gamma,\Gamma}^{(j)} x_{\Gamma}^{(i),k}, & \forall i \notin P_k. \end{cases} \end{aligned} \tag{A.5}$$

A.3.2 Conditions de convergence

Notons:

$$A := \begin{bmatrix} A_{1,1} & O & \cdots & O & A_{1,\Gamma} \\ O & A_{2,2} & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & O & A_{p-1,\Gamma} \\ O & \cdots & O & A_{p,p} & A_{p,\Gamma} \\ A_{\Gamma,1} & \cdots & A_{\Gamma,p-1} & A_{\Gamma,p} & \displaystyle\sum_{i=1}^{p} A_{\Gamma,\Gamma}^{(i)} \end{bmatrix}.$$

Proposition A.1. *Le modèle itératif asynchrone* (A.2)*, avec f définie par* (A.4)*, est convergent si A et $\sum_{i=1}^{p} M_{\Gamma,\Gamma}^{(i)}$ sont des* M*-matrices, et si*

$$\sum_{i=1}^{p} M_{\Gamma,\Gamma}^{(i)} - A_{\Gamma,\Gamma}^{(i)} \geq O.$$

Une M-matrice est une matrice dont les coefficients hors-diagonaux sont non-positifs, et dont l'inverse est non-négative. L'on peut donc voir, par exemple, dans [55, Corollary 3.24], qu'une matrice de Stieltjes est une M-matrice. Une matrice de Stieltjes est une matrice symétrique définie positive dont les coefficients hors-diagonaux sont non-positifs.

Corollaire A.1. *Le modèle itératif asynchrone* (A.2)*, avec f définie par* (A.4)*, est convergent si A est une* M*-matrice, et si*

$$M_{\Gamma,\Gamma}^{(i)} = \operatorname{blockdiag} A_{\Gamma,\Gamma}^{(i)}, \qquad \forall i \in \{1,\dots,p\}.$$

Proposition A.2. *Le modèle itératif asynchrone* (A.2)*, avec f définie par* (A.4)*, est convergent si $\sum_{i=1}^{p} S_{\Gamma,\Gamma}^{(i)}$ est une* M*-matrice, et si*

$$\sum_{i=1}^{p} M_{\Gamma,\Gamma}^{(i)} = \gamma I, \qquad \gamma \geq \max \operatorname{diag} \sum_{i=1}^{p} S_{\Gamma,\Gamma}^{(i)}.$$

Lemme A.1 (Crabtree & Haynsworth, 1969 [66])**.** *Un complément de Schur d'une* M*-matrice est une* M*-matrice.*

Corollaire A.2. *Le modèle itératif asynchrone* (A.2)*, avec f définie par* (A.4)*, est convergent si A est une* M*-matrice, et si*

$$\sum_{i=1}^{p} M_{\Gamma,\Gamma}^{(i)} = \gamma I, \qquad \gamma \geq \max \operatorname{diag} \sum_{i=1}^{p} A_{\Gamma,\Gamma}^{(i)}.$$

Remarque A.1. *Le modèle itératif asynchrone* (A.5)*, avec*

$$P_k = \{1, \ldots, p\}, \quad \tau_j^i(k) = k, \qquad \forall i, j \in \{1, \ldots, p\}, \quad \forall k \in \mathbb{N},$$

est convergent si le modèle itératif asynchrone (A.2)*, avec f définie par* (A.4)*, est convergent.*

Notons:

$$\widetilde{M} := \begin{bmatrix} A_{1,1} & O & \cdots & O & O \\ O & A_{2,2} & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & O & O \\ O & \cdots & O & A_{p,p} & O \\ O & \cdots & O & O & \displaystyle\sum_{i=1}^{p} M_{\Gamma,\Gamma}^{(i)} \end{bmatrix}.$$

Théorème A.8. *Le modèle itératif asynchrone* (A.5) *est convergent si*

$$\rho(|I - \widetilde{M}^{-1} A|) < 1,$$

et si

$$\left| \sum_{i=1}^{p} \left(\sum_{j=1}^{p} M_{\Gamma,\Gamma}^{(j)} \right)^{-1} \left(M_{\Gamma,\Gamma}^{(i)} - A_{\Gamma,\Gamma}^{(i)} \right) \right| = \sum_{i=1}^{p} \left| \left(\sum_{j=1}^{p} M_{\Gamma,\Gamma}^{(j)} \right)^{-1} \left(M_{\Gamma,\Gamma}^{(i)} - A_{\Gamma,\Gamma}^{(i)} \right) \right|.$$

A.4 Décomposition asynchrone temporelle

A.4.1 Modèle calculatoire

Soit, à résoudre, une équation aux dérivées partielles (EDP)

$$\delta(u(s,t),s,t)=0, \qquad t\in[0,T], \quad s\in\Omega,$$

avec une condition initiale, $u(\Omega,0)$, donnée. Considérons, comme précédemment, deux domaines temporels, $[0,T_1]$ et $[T_1,T]$, de telle sorte qu'en équivalence, l'on pourrait écrire:

$$\begin{cases} \delta(u_0(s,t),s,t)=0, & t\in[0,T_1], \quad s\in\Omega, \\ u_0(\Omega,0)=u(\Omega,0), & \\ \delta(u_1(s,t),s,t)=0, & t\in[T_1,T], \quad s\in\Omega, \\ u_1(\Omega,T_1)=u_0(\Omega,T_1). & \end{cases}$$

Ainsi, en supposant, sur chaque domaine, des conditions initiales inconnues, l'on peut formuler deux équations indépendantes,

$$\begin{cases} \delta(u_0(s,t),s,t)=0, & t\in[0,T_1], \quad s\in\Omega, \\ u_0(\Omega,0)=\lambda_0(\Omega), & \end{cases}$$

et

$$\begin{cases} \delta(u_1(s,t),s,t)=0, & t\in[T_1,T], \quad s\in\Omega, \\ u_1(\Omega,T_1)=\lambda_1(\Omega), & \end{cases}$$

dont les solutions sont celles de l'équation globale si, et seulement si,

$$\begin{cases} \lambda_0(\Omega)=u(\Omega,0), \\ \lambda_1(\Omega)=u_0(\Omega,T_1). \end{cases}$$

Soit maintenant F, une fonction dite de propagation fine, qui, à une condition initiale donnée sur un domaine temporel, fait correspondre la fonction solution de l'EDP à l'instant final. Précisément, l'on a:

$$F(\lambda_0)=u_0(\Omega,T_1), \qquad F(\lambda_1)=u_1(\Omega,T).$$

Soit également G, une fonction dite de propagation grossière, qui, à une condition initiale donnée sur un domaine temporel, fait correspondre une approximation grossière, peu coûteuse à évaluer, de la fonction solution de l'EDP à l'instant final. Précisément, l'on a:

$$G(\lambda_0)=\widetilde{u}_0(\Omega,T_1), \qquad G(\lambda_1)=\widetilde{u}_1(\Omega,T).$$

L'approche dite du Parareal [14] consiste à déterminer, de façon itérative, $u_0(\Omega, T_1)$ et $u_1(\Omega, T)$, sans nécessairement effectuer une propagation fine séquentielle sur le domaine global. Il s'agit donc de tendre vers une évaluation temps-réel de l'état d'un phénomène à un instant T quelconque à venir. Pour cela, les solutions $\widetilde{u}_0(\Omega, T_1)$ et $\widetilde{u}_1(\Omega, T)$ sont "prédites" à l'aide du propagateur G, puis, tenant compte de l'écart avec les solutions $u_0(\Omega, T_1)$ et $u_1(\Omega, T)$ issues du propagateur F appliqué aux mêmes conditions initiales que G, les prédictions à l'itération suivante sont réajustées. Cela donne, par exemple, l'exécution parallèle suivante, que l'on voit résulter néanmoins en une propagation fine séquentielle, du fait des deux derniers pas sur $[T_1, T]$, ce qui traduit une non convergence des prédictions itératives:

$\lambda_0^0 := u(\Omega, 0)$		
$\lambda_1^0 := G(\lambda_0^0)$	attente	$[0, T_1]$
	$\lambda_2^0 := G(\lambda_1^0)$	$[T_1, T]$
$F(\lambda_0^0)$	$F(\lambda_1^0)$	$[0, T_1],\ [T_1, T]$
$\lambda_0^1 := \lambda_0^0$		
$\lambda_1^1 := G(\lambda_0^1) + F(\lambda_0^0) - G(\lambda_0^0)$	attente	$[0, T_1]$
	$\lambda_2^1 := G(\lambda_1^1) + F(\lambda_1^0) - G(\lambda_1^0)$	$[T_1, T]$
$\lambda_0^2 := \lambda_0^1;\ \lambda_1^2 := \lambda_1^1$	$F(\lambda_1^1)$	$[T_1, T]$
	$\lambda_2^2 := G(\lambda_1^2) + F(\lambda_1^1) - G(\lambda_1^1)$	$[T_1, T]$

Soit donc, pour p domaines temporels joints, le modèle itératif défini par:

$$\left\{\begin{array}{lcll} \lambda_0^0 & = & u(\Omega, 0), & \\ \lambda_i^0 & = & G(\lambda_{i-1}^0), & \forall i \in \{1, \ldots, p\}, \\ \lambda_0^{k+1} & = & \lambda_0^k, & \\ \lambda_i^{k+1} & = & G(\lambda_{i-1}^{k+1}) + F(\lambda_{i-1}^k) - G(\lambda_{i-1}^k), & \forall i \in \{1, \ldots, p\}. \end{array}\right. \tag{A.6}$$

Nous étudions ici sa généralisation en:

$$\left\{\begin{array}{lcll} \lambda_0^0 & = & u(\Omega, 0), & \\ \lambda_i^0 & = & G(\lambda_{i-1}^0), & \forall i \in \{1, \ldots, p\}, \\ \lambda_0^{k+1} & = & \lambda_0^k, & \\ \lambda_i^{k+1} & = & \left\{\begin{array}{ll} G(\lambda_{i-1}^{\tau_{i-1,1}(k)}) + F(\lambda_{i-1}^{\tau_{i-1,2}(k)}) - G(\lambda_{i-1}^{\tau_{i-1,2}(k)}), & \forall i \in P_k, \\ \lambda_i^k, & \forall i \notin P_k. \end{array}\right. & \end{array}\right. \tag{A.7}$$

A.4.2 Conditions de convergence

Considérons une discrétisation temporelle et spatiale telle que

$$G(\lambda_i) = \mathcal{G}\lambda_i + c, \qquad F(\lambda_i) = \mathcal{F}\lambda_i + d, \qquad i \in \{0, 1, \ldots, p-1\},$$

où $\mathcal{G}$ et $\mathcal{F}$ sont des opérateurs linéaires, et posons:

$$\lambda = \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_p \end{bmatrix}, \qquad \lambda^* = \begin{bmatrix} u(\Omega, 0) \\ u(\Omega, T_1) \\ \vdots \\ u(\Omega, T) \end{bmatrix}.$$

Nous étendons explicitement aux PDE les résultats de [45] sur les équations différentielles ordinaires, pour le modèle itératif synchrone (A.6):

Proposition A.3. *Le modèle itératif synchrone* (A.6) *génère une suite* $\{\lambda^k\}_{k\in\mathbb{N}}$ *telle que*

$$\|\lambda^k - \lambda^*\|_\infty \leq \alpha^k \|\lambda^0 - \lambda^*\|_\infty,$$

avec

$$\alpha = \frac{1-\theta^p}{1-\theta}\|\mathcal{F} - \mathcal{G}\|, \qquad \theta \neq 1, \quad \theta \geq \|\mathcal{G}\|.$$

Remarque A.2.

$$\lim_{p\to+\infty} \frac{1-\theta^p}{1-\theta}\|\mathcal{F} - \mathcal{G}\| = \begin{cases} +\infty, & \theta > 1, \\ \\ \dfrac{1}{1-\theta}\|\mathcal{F} - \mathcal{G}\|, & \theta < 1. \end{cases}$$

Le modèle itératif synchrone (A.6) *est donc stable, quelque soit p, si*

$$\theta < 1,$$

ce qui implique également que

$$\|\mathcal{G}\| < 1.$$

Corollaire A.3. *Dans sa région de stabilité, le modèle itératif synchrone* (A.6) *est convergent si*

$$\|\mathcal{G}\| + \|\mathcal{F} - \mathcal{G}\| < 1 + \|\mathcal{G}\|^p \|\mathcal{F} - \mathcal{G}\|.$$

Remarque A.3.

$$\|\mathcal{G}\| < 1 \quad \Longrightarrow \quad \lim_{p\to+\infty} \|\mathcal{G}\|^p \|\mathcal{F} - \mathcal{G}\| = 0.$$

Ainsi, dans sa région de stabilité, le modèle itératif synchrone (A.6) *est asymptotiquement convergent si*

$$\|\mathcal{G}\| + \|\mathcal{F} - \mathcal{G}\| < 1.$$

Nous montrons alors que:

Proposition A.4. *Le modèle itératif asynchrone* (A.7) *génère une suite* $\{\lambda^k\}_{k\in\mathbb{N}}$ *telle que*

$$\|\lambda^k - \lambda^*\|_\infty \leq \widetilde{\alpha}^{\tau(k)}\|\lambda^0 - \lambda^*\|_\infty,$$

avec

$$\widetilde{\alpha} = \|\mathcal{G}\| + \|\mathcal{F} - \mathcal{G}\|, \qquad \lim_{k\to+\infty} \tau(k) = +\infty.$$

Corollaire A.4. *Le modèle itératif asynchrone* (A.7) *est convergent si*

$$\|\mathcal{G}\| + \|\mathcal{F} - \mathcal{G}\| < 1.$$

A.5 Terminaison d'itérations asynchrones

A.5.1 Détection de convergence asynchrone

Soit, en cours de résolution itérative sur p processeurs, un problème de point fixe

$$f(x) - x = 0,$$

où x est un vecteur d'inconnues complexes. Soit r une fonction de x telle que

$$r(x) \simeq 0 \quad \Longrightarrow \quad x \simeq x^*.$$

La détection de convergence asynchrone consiste donc à évaluer (de façon non-bloquante) une assertion

$$r(\bar{x}) \simeq 0, \qquad \bar{x} = \begin{bmatrix} x_1^{k_1} \\ \vdots \\ x_p^{k_p} \end{bmatrix},$$

à partir d'une suite $\{x^k\}_{k\in\mathbb{N}}$ en cours de génération suivant le modèle itératif asynchrone (A.2). Les principales approches (distribuées) dans l'état de l'art sont basées sur

- une modification du processus itératif afin d'en assurer une terminaison en temps fini (Bertsekas et Tsitsiklis, 1989 [33]; El Baz, 1996 [34]; Savari et Bertsekas, 1996 [35]),
- une évaluation exacte du résidu $r(\bar{x})$ à partir d'une capture d'état global $\bar{x}$ (Savari et Bertsekas, 1996 [35]),
- une prédiction approximative du nombre d'itérations nécessaire à la convergence (Evans et Chikohora, 1998 [36]),

- une supervision de la concordance et de la persistance des convergences locales (Bahi *et al.*, 2005 [73], 2008 [37]),
- une évaluation de diamètre d'ensembles emboîtés à partir de "macro-itérations" (Miellou *et al.*, 2008 [48]).

L'approche par modification a l'inconvénient majeur d'être intrusive et de requérir la prise en compte, au niveau du modèle itératif, d'hypothèses supplémentaires. L'approche par ensembles emboîtés, abordée sur un plan essentiellement mathématique, laisse entrevoir, sur le plan d'une implémentation distribuée, des techniques intrusives d'encapsulation de messages de contrôle (*piggybacking*). Les approches par supervision des convergences locales et par prédiction de nombre d'itérations ne garantissent pas l'exactitude d'une convergence détectée. Enfin, bien que l'approche par capture d'état global permette une évaluation exacte du résidu global, sa mise en œuvre proposée par [35] requiert des messages de contrôle de même taille que les messages de calcul.

Nous proposons donc, dans cette thèse, l'évaluation exacte, non-intrusive, du résidu global avec des messages de contrôle de taille unitaire constante. Table A.1 donne un aperçu des caractéristiques de ces principales méthodes, NFAIS (*Non-FIFO Asynchronous Iterations Snapshot*) étant celle que nous proposons et détaillons dans la section suivante.

A.5.2 Approche exacte par résidu global

L'évaluation exacte d'un résidu global $r(\bar{x})$ passe par la capture d'un état global $\bar{x}$, suivie d'une opération classique de réduction pour y appliquer la fonction r, le tout de façon distribuée et non-bloquante, en parallèle du processus itératif. La capture distribuée de l'état global d'un système distribué a été introduite par [46], où le protocole suivant fut proposé afin d'assurer un état global cohérent:

- initiateur(s) ou sur première réception d'un marqueur:
  1. enregistrer l'état local,
  2. envoyer un marqueur aux voisins,
  3. débuter l'enregistrement des messages de calcul reçus;
- sur réception d'un marqueur:
  1. cesser l'enregistrement des messages de calcul, pour le voisin correspondant;
- sur réception d'un message de calcul:
  1. enregistrer le message comme faisant partie de l'état du canal de communication correspondant.

	Intrusion	Centralisation
Bertsekas & Tsitsiklis, 1989	itérations modifiées	–
El Baz, 1996	itérations modifiées	–
Savari & Bertsekas, 1996	itérations modifiées	–
Savari & Bertsekas, 1996	non-intrusif	2 réductions
Evans & Chikohora, 1998	non-intrusif	aucune centralisation
Bahi *et al.*, 2005	non-intrusif	1 réduction
Bahi *et al.*, 2008	encapsulation	2 réductions
Miellou *et al.*, 2008	–	–
NFAIS	non-intrusif	1 réduction
	Robustesse	Taille des messages
Bertsekas & Tsitsiklis, 1989	fiable	–
El Baz, 1996	fiable	–
Savari & Bertsekas, 1996	fiable	–
Savari & Bertsekas, 1996	résidu exact	$\mathcal{O}(n)$
Evans & Chikohora, 1998	heuristique	0
Bahi *et al.*, 2005	heuristique	$\mathcal{O}(1)$
Bahi *et al.*, 2008	heuristique	$\mathcal{O}(1)$
Miellou *et al.*, 2008	fiable	–
NFAIS	fiable	$\mathcal{O}(1)$

Table A.1: Méthodes de terminaison d'itérations asynchrones.

Conçu dans un modèle calculatoire où chaque réception de message induit un changement explicite d'état, ce protocole ne peut s'appliquer tel quel dans un contexte d'exécution aléatoire d'itérations asynchrones, comme le montre l'exemple suivant correspondant à Figure A.4, où l'on a, à la fin de la capture, des données manquantes sur le processeur 1, rendant impossible sa part d'évaluation distribuée d'un résidu $r(x_1^{k_1}, x_2^{k_2})$:

	$x_1^1 := f_1(x_1^0, x_2^0)$	x_2^1	$x_2^1 := f_2(x_1^0, x_2^0)$	
	$x_1^2 := f_1(x_1^1, x_2^0)$		$x_2^2 := f_2(x_1^0, x_2^1)$	
x_1^2	$x_1^3 := x_1^2$		$x_2^3 := f_2(x_1^1, x_2^2)$	x_1^1
	$x_1^4 := f_1(x_1^3, x_2^2)$		$x_2^4 := f_2(x_1^2, x_2^3)$	x_1^2
	$x_1^5 := f_1(x_1^4, x_2^3)$		$x_2^5 := f_2(x_1^2, x_2^4)$	
	$\bar{x}^{(1)} := (x_1^2, ?)$		$\bar{x}^{(2)} := (x_1^2, x_2^1)$	

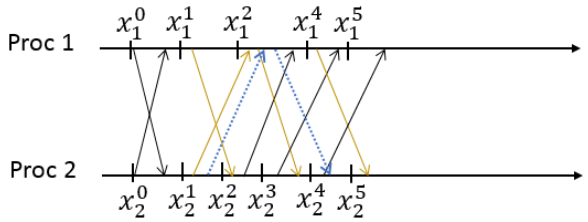


Figure A.4: Exécution parallèle asynchrone, avec marqueurs d'état global (en bleu).

Nous proposons donc l'application suivante du protocole, avec réadaptation, au contexte particulier de la capture d'un vecteur itératif global asynchrone (AIS: *Asynchronous Iterations Snapshot*):

- sur convergence locale ou première réception d'un marqueur:
  1. enregistrer l'état local,
  2. envoyer un marqueur aux voisins;
- sur réception d'un marqueur:
  1. enregistrer le dernier message reçu du voisin correspondant.

Ceci conduit, cette fois, à:

$$
\begin{array}{lll}
 & x_1^1 := f_1(x_1^0, x_2^0) & \\
 & x_1^2 := f_1(x_1^1, x_2^0) & \\
x_1^2 & x_1^3 := x_1^2 & x_2^1 \\
 & x_1^4 := f_1(x_1^3, x_2^2) & \\
 & x_1^5 := f_1(x_1^4, x_2^3) & \\
 & \bar{x}^{(1)} := (x_1^2, x_1^1) &
\end{array}
\qquad
\begin{array}{lll}
x_2^1 & x_2^1 := f_2(x_1^0, x_2^0) & \\
 & x_2^2 := f_2(x_1^0, x_2^1) & \\
 & x_2^3 := f_2(x_1^1, x_2^2) & \\
 & x_2^4 := f_2(x_1^2, x_2^3) & x_1^2 \\
 & x_2^5 := f_2(x_1^2, x_2^4) & \\
 & \bar{x}^{(2)} := (x_1^2, x_1^1) &
\end{array}
$$

et nous obtenons ainsi:

$$\bar{x} = \bar{x}^{(1)} = \bar{x}^{(2)}.$$

L'inconvénient majeur de ce protocole réside dans la nécessité d'assurer, sur chaque canal de communication (unidirectionnel), qu'un marqueur envoyé après un message de calcul ne puisse en aucun cas être délivré avant ce message, et vice-versa. Pour lever cette hypothèse de communication FIFO (*first in, first out*), la solution proposée dans [35] consiste à introduire, dans les marqueurs, les données d'interface correspondant au vecteur solution local enregistré. Afin donc d'éviter ce surcoût de communication en $\mathcal{O}(n)$, nous caractérisons les environnements non-FIFO comme suit:

Hypothèse A.1. *Un message peut croiser au plus m autre messages sur chaque canal de communication unidirectionnel.*

De là, nous proposons un protocole AIS non-FIFO (NFAIS) où:

1. un marqueur n'est envoyé qu'après m itérations sous convergence locale,
2. et un second marqueur, booléen, est envoyé après m itérations suivantes, invalidant le premier marqueur en cas de non-persistance continue de la convergence locale.

Remarque A.4. *En pratique, un marqueur est toujours transmis beaucoup plus vite que des données d'interface, et dans ce cas, le second marqueur s'avérerait inutile.*

Bien qu'un tel protocole NFAIS rende impossible l'évaluation exacte directe d'un résidu global, du fait qu'il génère en sortie

$$\bar{x}^{(1)} \neq \cdots \neq \bar{x}^{(p)},$$

prenons toutefois

$$\bar{x} = \begin{bmatrix} \bar{x}_1^{(1)} \\ \vdots \\ \bar{x}_p^{(p)} \end{bmatrix},$$

et considérons une fonction résidu $r(.)$ distribuée comme suit:

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

où $\sigma(.)$ est une fonction de réduction. Le résidu global approximatif directement évalué sur la base de ce protocole NFAIS correspond donc à

$$\widetilde{r}(\bar{x}^{(1)}, \ldots, \bar{x}^{(p)}) := \sigma(r_1(\bar{x}^{(1)}), \ldots, r_p(\bar{x}^{(p)})).$$

Soit, enfin, un critère de convergence locale défini par:

$$r_i\left(x_1^{\tau_1^i(k)}, \ldots, x_p^{\tau_p^i(k)}\right) < \varepsilon, \qquad i \in \{1, \ldots, p\}, \quad \varepsilon \in \mathbb{R}.$$

Nous montrons que:

Proposition A.5.

$$r(\bar{x}) < \widetilde{r}(\bar{x}^{(1)}, \ldots, \bar{x}^{(p)}) + c_r(p, m)\varepsilon,$$

où $c_r(p, m)$ désigne une fonction constante de p et m, dont l'expression dépend de celle de $r(.)$.

L'on peut alors, de façon fiable, évaluer un critère de convergence global sur le résidu approximatif, de telle sorte à impliquer une convergence globale vis-à-vis du résidu exact. Nous montrons de surcroît que l'utilisation d'un unique seuil de résidu comme critère de convergence, aussi bien locale que globale, reste tout à fait possible en pratique:

Corollaire A.5. *Si $r(.)$ est la norme infinie pondérée $\|.\|_\infty^w$, avec*

$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix},$$

alors

$$\widetilde{r}(\bar{x}^{(1)}, \ldots, \bar{x}^{(p)}) < \varepsilon \quad \Longrightarrow \quad r(\bar{x}) < \widetilde{\varepsilon},$$

avec

$$\varepsilon = \widetilde{c}(m, w)\widetilde{\varepsilon},$$

où $\widetilde{c}(m, w)$ désigne une fonction constante de m et w.

Bibliography

[1] G. M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, in: Proceedings of the Spring Joint Computer Conference, April 18-20, 1967, Atlantic City, NJ, USA, AFIPS '67 (Spring), 1967, pp. 483–485.

[2] A. Toselli, O. Widlund, Domain Decomposition Methods - Algorithms and Theory, Vol. 34 of Springer Series in Computational Mathematics, Springer-Verlag Berlin Heidelberg, 2005.

[3] U. Elsner, Graph Partitioning: A Survey, Technische Universität Chemnitz, Chemnitz, Germany, 1997.

[4] S. Ghanemi, A domain decomposition method for helmholtz scattering problems, in: Ninth International Conference on Domain Decomposition Methods, 1996, pp. 105–112.

[5] P. Chevalier, F. Nataf, Symmetrized method with optimized second-order conditions for the Helmholtz equation, in: Domain decomposition methods, 10 (Boulder, CO, 1997), American Mathematical Society, Providence, RI, 1998, pp. 400–407.

[6] M. Gander, L. Halpern, F. Magoulès, An optimized Schwarz method with two-sided Robin transmission conditions for the Helmholtz equation, International Journal for Numerical Methods in Fluids 55 (2) (2007) 163–175.

[7] F. Magoulès, F.-X. Roux, S. Salmon, Optimal discrete transmission conditions for a non-overlapping domain decomposition method for the Helmholtz equation, SIAM Journal on Scientific Computing 25 (5) (2004) 1497–1515.

[8] F.-X. Roux, F. Magoulès, L. Series, Y. Boubendir, Approximation of optimal interface boundary conditions for two-Lagrange multiplier FETI method, in: Proceedings of the 15th International Conference on Domain Decomposition Methods, Berlin, Germany, July 21-15, 2003, Lecture Notes in Computational Science and Engineering, Springer-Verlag, Heidelberg, 2005.

[9] F. Magoulès, F.-X. Roux, L. Series, Algebraic approach to absorbing boundary conditions for the Helmholtz equation, International Journal of Computer Mathematics 84 (2) (2007) 231–240.

[10] E. Lelarasmee, A. E. Ruehli, A. L. Sangiovanni-Vincentelli, The waveform relaxation method for time-domain analysis of large scale integrated circuits, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 1 (3) (1982) 131–145.

[11] M. J. Gander, H. Zhao, Overlapping Schwarz waveform relaxation for the heat equation in n dimensions, BIT Numerical Mathematics 42 (4) (2002) 779–795.

[12] P. Chartier, B. Philippe, A parallel shooting technique for solving dissipative ODE's, Computing 51 (3) (1993) 209–236.

[13] B. Leimkuhler, Timestep acceleration of waveform relaxation, SIAM Journal on Numerical Analysis 35 (1) (1998) 31–50.

[14] J.-L. Lions, Y. Maday, G. Turinici, Résolution d'EDP par un schéma en temps "pararéel", C. R. Acad. Sci. Paris Sér. I Math. 332 (7) (2001) 661–668.

[15] C. Farhat, M. Chandesris, Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid–structure applications, International Journal for Numerical Methods in Engineering 58 (9) (2003) 1397–1434.

[16] D. P. Bertsekas, J. N. Tsitsiklis, Some aspects of parallel and distributed iterative algorithms – a survey, Automatica 27 (1) (1991) 3–21.

[17] J. C. Miellou, Algorithmes de relaxation chaotique à retards, ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique 9 (R1) (1975) 55–82.

[18] G. M. Baudet, Asynchronous iterative methods for multiprocessors, Journal of the ACM 25 (2) (1978) 226–244.

[19] M. N. El Tarazi, Some convergence results for asynchronous algorithms, Numerische Mathematik 39 (3) (1982) 325–340.

[20] D. P. Bertsekas, Distributed asynchronous computation of fixed points, Mathematical Programming 27 (1) (1983) 107–120.

[21] D. Chazan, W. Miranker, Chaotic relaxation, Linear Algebra and its Applications 2 (2) (1969) 199–222.

[22] A. Frommer, D. B. Szyld, Asynchronous two-stage iterative methods, Numerische Mathematik 69 (2) (1994) 141–153.

[23] J. C. Miellou, D. E. Baz, P. Spiteri, A new class of asynchronous iterative algorithms with order intervals, Mathematics of Computation 67 (221) (1998) 237–255.

[24] D. El Baz, A. Frommer, P. Spiteri, Asynchronous iterations with flexible communication: contracting operators, Journal of Computational and Applied Mathematics 176 (1) (2005) 91 – 103.

[25] K. Li-Shan, C. Yu-Ping, S. Le-Lin, Q. Hui-Yun, The asynchronous parallel algorithms s-cor for solving p.d.e.'s on multiprocessors, International Journal of Computer Mathematics 18 (2) (1985) 163–172.

[26] L. Hart, S. McCormick, Asynchronous multilevel adaptive methods for solving partial differential equations on multiprocessors: Basic ideas, Parallel Computing 12 (2) (1989) 131–144.

[27] M. Chau, D. E. Baz, R. Guivarch, P. Spiteri, MPI implementation of parallel subdomain methods for linear and nonlinear convection–diffusion problems, Journal of Parallel and Distributed Computing 67 (5) (2007) 581–591.

[28] R. Bru, V. Migallón, J. Penadés, D. B. Szyld, Parallel, synchronous and asynchronous two-stage multisplitting methods, Electronic Transactions on Numerical Analysis 3 (1995) 24–38.

[29] P. Spiteri, J.-C. Miellou, D. El Baz, Parallel asynchronous schwarz and multisplitting methods for a nonlinear diffusion problem, Numerical Algorithms 33 (1) (2003) 461–474.

[30] M. Chau, R. Couturier, J. Bahi, P. Spiteri, Parallel solution of the obstacle problem in grid environments, International Journal of High Performance Computing Applications 25 (4) (2011) 488–495.

[31] E. D. Chajakis, S. A. Zenios, Synchronous and asynchronous implementations of relaxation algorithms for nonlinear network optimization, Parallel Computing 17 (8) (1991) 873–894.

[32] M. Chau, Algorithmes parallèles asynchrones pour la simulation numérique, Ph.D. thesis, Institut National Polytechnique de Toulouse, Toulouse, France (Nov. 2005).

[33] D. P. Bertsekas, J. N. Tsitsiklis, Convergence rate and termination of asynchronous iterative algorithms, in: Proceedings of the 3rd International Conference on Supercomputing, 5–9 June 1989, Crete, Greece, 1989, pp. 461–470.

[34] D. El Baz, A method of terminating asynchronous iterative algorithms on message passing systems, Parallel Algorithms and Applications 9 (1-2) (1996) 153–158.

[35] S. A. Savari, D. P. Bertsekas, Finite termination of asynchronous iterative algorithms, Parallel Computing 22 (1) (1996) 39–56.

[36] D. J. Evans, S. Chikohora, Convergence testing on a distributed network of processors, International Journal of Computer Mathematics 70 (2) (1998) 357–378.

[37] J. M. Bahi, S. Contassot-Vivier, R. Couturier, An efficient and robust decentralized algorithm for detecting the global convergence in asynchronous iterative algorithms, in: High Performance Computing for Computational Science - VECPAR 2008, Vol. 5336 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 240–254.

[38] J. Bahi, S. Domas, K. Mazouzi, Jace: a java environment for distributed asynchronous iterative computations, in: 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing. 11–13 February 2004, A Coruña, Spain., 2004, pp. 350–357.

[39] J. M. Bahi, R. Couturier, P. Vuillemin, Jacev: A programming and execution environment for asynchronous iterative computations on volatile nodes, in: High Performance Computing for Computational Science - VECPAR 2006, Vol. 4395 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, pp. 79–92.

[40] J.-C. Charr, R. Couturier, D. Laiymani, A decentralized and fault tolerant convergence detection algorithm for asynchronous iterative algorithms, The Journal of Supercomputing 53 (2) (2010) 269–292.

[41] J.-C. Charr, R. Couturier, D. Laiymani, JACEP2P-V2: A fully decentralized and fault tolerant environment for executing parallel iterative asynchronous applications on volatile distributed architectures, in: Advances in Grid and Pervasive Computing: 4th International Conference, GPC 2009, Geneva, Switzerland, May 4-8, 2009. Proceedings, Springer Berlin Heidelberg, 2009, pp. 446–458.

[42] R. Couturier, S. Domas, Crac: a grid environment to solve scientific applications with asynchronous iterative algorithms, in: IEEE International Parallel and Distributed Processing Symposium. 26–30 March 2007, Long Beach, California, USA., 2007, pp. 1–8.

[43] O. Widlund, M. Dryja, An additive variant of the Schwarz alternating method for the case of many subregions, Tech. rep., Department of Computer Science, Courant Institute, New York, USA, Technical Report 339, Ultracomputer Note 131 (1987).

[44] M. J. Gander, Schwarz methods over the course of time, Electronic Transactions on Numerical Analysis 31 (2008) 228–255.

[45] M. J. Gander, S. Vandewalle, Analysis of the parareal time-parallel time-integration method, SIAM J. Sci. Comput. 29 (2) (2007) 556–578.

[46] K. M. Chandy, L. Lamport, Distributed snapshots: Determining global states of distributed systems, ACM Trans. Comput. Syst. 3 (1) (1985) 63–75.

[47] F. Magoulès, C. Venet, Asynchronous iterative sub-structuring methods, Mathematics and Computers in Simulation 145 (Supplement C) (2018) 34–49.

[48] J. Miellou, P. Spiteri, D. El Baz, A new stopping criterion for linear perturbed asynchronous iterations, Journal of Computational and Applied Mathematics 219 (2) (2008) 471–483.

[49] F. Magoulès, G. Gbikpi-Benissan, Distributed convergence detection based on global residual error under asynchronous iterations, IEEE Transactions on Parallel and Distributed Systems (preprint, 2017). doi:10.1109/TPDS.2017.2780856.

[50] F. Magoulès, G. Gbikpi-Benissan, Jack2: an mpi-based communication library with non-blocking synchronization for asynchronous iterations, Advances in Engineering Software (preprint, 2018). doi:10.1016/j.advengsoft.2018.01.009.

[51] G. Gbikpi-Benissan, F. Magoulès, JACK2: A New High-Level Communication Library for Parallel Iterative Methods, in: P. Iványi, B. Topping, G. Várady (Eds.), Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Civil-Comp Press, Stirlingshire, UK, 2017.

[52] F. Magoulès, G. Gbikpi-Benissan, JACK: an asynchronous communication kernel library for iterative algorithms, The Journal of Supercomputing 73 (8) (2017) 3468–3487.

[53] J. M. Bahi, S. Contassot-Vivier, R. Couturier, Parallel Iterative Algorithms: from sequential to grid computing, Numerical Analysis and Scientific Computing, Chapman & Hall/CRC, 2007.

[54] K. Fan, Note on M-matrices, The Quarterly Journal of Mathematics 11 (1) (1960) 43–49.

[55] R. S. Varga, Matrix Iterative Analysis, Vol. 27 of Springer Series in Computational Mathematics, Springer-Verlag Berlin Heidelberg, 2000.

[56] A. Frommer, D. B. Szyld, H-splittings and two-stage iterative methods, Numerische Mathematik 63 (1) (1992) 345–356.

[57] A. Frommer, D. B. Szyld, On asynchronous iterations, Journal of Computational and Applied Mathematics 123 (1-2) (2000) 201–216.

[58] A. Neumaier, New techniques for the analysis of linear interval equations, Linear Algebra and its Applications 58 (1984) 273–325.

[59] A. Frommer, D. B. Szyld, Asynchronous iterations with flexible communication for linear systems, Calculateurs Parallèles 10 (1998) 421–429.

[60] J. Bahi, E. Griepentrog, J. C. Miellou, Parallel treatment of a class of differential-algebraic systems, SIAM Journal on Numerical Analysis 33 (5) (1996) 1969–1980.

[61] A. Frommer, H. Schwandt, D. B. Szyld, Asynchronous weighted additive Schwarz methods, Electronic Transactions on Numerical Analysis 5 (1997) 48–61.

[62] P. Spitéri, J.-C. Miellou, D. El Baz, Asynchronous Schwarz alternating method with flexible communication for the obstacle problem, Réseaux et Systèmes Répartis - Calculateurs Parallèles 13 (1) (2001) 47–66.

[63] F. Magoulès, D. B. Szyld, C. Venet, Asynchronous optimized Schwarz methods with and without overlap, Numerische Mathematik 137 (1) (2017) 199–227.

[64] J. S. Przemieniecki, Matrix structural analysis of substructures, American Institute of Aeronautics and Astronautics Journal 1 (1) (1963) 138–147.

[65] D. E. Crabtree, Applications of M-matrices to non-negative matrices, Duke Math. J. 33 (1) (1966) 197–208.

[66] D. E. Crabtree, E. V. Haynsworth, An identity for the Schur complement of a matrix, Proc. Amer. Math. Soc. 22 (2) (1969) 364–366.

[67] D. P. Bertsekas, J. N. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[68] E. W. Dijkstra, C. S. Scholten, Termination detection for diffusing computations, Information Processing Letters 11 (1) (1980) 1–4.

[69] N. Francez, M. Rodeh, Achieving distributed termination without freezing, IEEE Trans. Softw. Eng. 8 (3) (1982) 287–292.

[70] S. P. Rana, A distributed solution of the distributed termination problem, Information Processing Letters 17 (1) (1983) 43–46.

[71] F. Mattern, Algorithms for distributed termination detection, Distributed Computing 2 (3) (1987) 161–175.

[72] A. D. Kshemkalyani, M. Raynal, M. Singhal, An introduction to snapshot algorithms in distributed computing, Distributed Systems Engineering 2 (4) (1995) 224–233.

[73] J. M. Bahi, S. Contassot-Vivier, R. Couturier, F. Vernier, A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms, IEEE Trans. Parallel Distrib. Syst. 16 (1) (2005) 4–13.

[74] N. A. Lynch, Distributed Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[75] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, S. Kumar, E. Lusk, R. Thakur, J. L. Träff, MPI on a Million Processors, in: M. Ropo, J. Westerholm, J. Dongarra (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface: 16th European PVM/MPI Users' Group Meeting, Espoo, Finland, September 7-10, 2009. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 20–30.

[76] J. Rumbaugh, I. Jacobson, G. Booch, The unified modeling language reference manual, Vol. 1 of Object technology series, Addison-Wesley, Boston, USA, 1999.

[77] E. J. H. Chang, Echo algorithms: Depth parallel operations on general graphs, IEEE Transactions on Software Engineering SE-8 (4) (1982) 391–401.

[78] E. Aubanel, Scheduling of tasks in the parareal algorithm, Parallel Computing 37 (3) (2011) 172–182.

université PARIS-SACLAY
ÉCOLE DOCTORALE
INTERFACES
Approches Interdisciplinaires : Fondements, Applications et Innovation

Titre : Méthodes asynchrones de décomposition de domaine pour le calcul massivement parallèle

Mots clefs : Itérations asynchrones, Méthode de sous-structuration, Algorithme temps-parallèle, Détection de convergence, Calcul distribué.

Résumé : Une large classe de méthodes numériques possède une propriété d'échelonnabilité connue comme étant la loi d'Amdahl. Elle constitue l'inconvénient majeur limitatif du calcul parallèle, en ce sens qu'elle établit une borne supérieure sur le nombre d'unités de traitement parallèles qui peuvent être utilisées pour accélérer un calcul. Des activités de recherche sont donc largement conduites à la fois sur les plans mathématiques et informatiques, pour repousser cette limite afin d'être en mesure de tirer le maximum des machines parallèles. Les méthodes de décomposition de domaine introduisent une approche naturelle et optimale pour résoudre de larges problèmes numériques de façon distribuée. Elles consistent en la division du domaine géométrique sur lequel une équation est définie, puis le traitement itératif de chaque sous-domaine, séparément, tout en assurant la continuité de la solution et de sa dérivée sur leur interface de jointure. Dans le présent travail, nous étudions la suppression de la limite d'accélération en appliquant des itérations asynchrones dans différents cadres de décomposition, à la fois de domaines spatiaux et temporels. Nous couvrons plusieurs aspects du développement d'algorithmes asynchrones, de l'analyse théorique de convergence à la mise en oeuvre effective. Nous aboutissons ainsi à des méthodes asynchrones efficaces pour la décomposition de domaine, ainsi qu'à une nouvelle bibliothèque de communication pour l'expérimentation asynchrone rapide d'applications scientifiques existantes.

Title : Asynchronous domain decomposition methods for massively parallel computing

Keywords : Asynchronous iterations, Sub-structuring method, Time-parallel algorithm, Convergence detection, Distributed computing.

Abstract : An important class of numerical methods features a scalability property well known as the Amdahl's law, which constitutes the main limiting drawback of parallel computing, as it establishes an upper bound on the number of parallel processing units that can be used to speed a computation up. Extensive research activities are therefore conducted on both mathematical and computer science aspects to increase this bound, in order to be able to squeeze the most out of parallel machines. Domain decomposition methods introduce a natural and optimal approach to solve large numerical problems in a distributed way. They consist in dividing the geometrical domain on which an equation is defined, then iteratively processing each sub-domain separately, while ensuring the continuity of the solution and of its derivative across the junction interface between them. In the present work, we investigate the removal of the scalability bound by the application of the asynchronous iterations theory in various decomposition frameworks, both for space and time domains. We cover various aspects of the development of asynchronous iterative algorithms, from theoretical convergence analysis to effective parallel implementation. Efficient asynchronous domain decomposition methods are thus successfully designed, as well as a new communication library for the quick asynchronous experimentation of existing scientific applications.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France