On the resilience of parallel sparse hybrid solvers

Emmanuel Agullo, Luc Giraud, Mawussi Zounon

On the resilience of parallel sparse hybrid solvers

Emmanuel Agullo
Inria
Bordeaux, France
Emmanuel.agullo@inria.fr

Luc Giraud
Inria
Bordeaux, France
Luc.giraud@inria.fr

Mawussi Zounon
Inria
Bordeaux, France
Mawussi.zounon@inria.fr

***Abstract*—As the computational power of high performance computing (HPC) systems continues to increase by using a huge number of CPU cores or specialized processing units, extreme-scale applications are increasingly prone to faults. Consequently, the HPC community has proposed many contributions to design resilient HPC applications. These contributions may be system-oriented, theoretical or numerical. In this study we consider an actual fully-featured parallel sparse hybrid (direct/iterative) linear solver,** MAPHYS**, and we propose numerical remedies to design a resilient version of the solver. The solver being hybrid, we focus in this study on the iterative solution step, which is often the dominant step in practice. We furthermore assume that a separate mechanism ensures fault detection and that a system layer provides support for setting back the environment (processes, ...) in a running state. The present manuscript therefore focuses on (and only on) strategies for recovering lost data *after* the fault has been detected (a separate concern beyond the scope of this study), *once* the system is restored (another separate concern not studied here). The numerical remedies we propose are twofold. Whenever possible, we exploit the natural data redundancy between processes from the solver to perform exact recovery through clever copies over processes. Otherwise, data that has been lost and no longer available on any process is recovered through a so-called interpolation-restart mechanism. This mechanism is derived from [1] by carefully taking into account the properties of the target hybrid solver. These numerical remedies have been implemented in the** MAPHYS **parallel solver so that we can assess their efficiency on a large number of processing units (up to $12,288$ CPU cores) for solving large-scale real-life problems.**

***Keywords*-Linear algebra; algorithms; HPC; fault tolerance; resilience; numerical methods; solver; Krylov; Gmres**

I. INTRODUCTION

Parallel sparse linear algebra solvers are often the innermost numerical kernels in scientific and engineering applications; consequently, they are one of the most time consuming parts. In order to cope with the hierarchical hardware design of modern large-scale supercomputers, the HPC solver community has proposed new sparse hybrid (direct / iterative) methods. To achieve a high scalability, algebraic domain decomposition methods are commonly employed to split a large size linear system into smaller size linear systems. To achieve this goal, the Schur complement method is often used to design sparse hybrid linear solvers [2], [3], [4], [5].

However, today's extreme-scale simulations require solving linear systems so large that the time between two consecutive faults may be smaller than the time required by linear algebra solvers to complete. Consequently, it becomes critical to enhance sparse solvers so that they can provide a correct solution in the presence of faults. This challenge is tremendous and requires a strong involvement of the whole HPC community ranging from hardware and system expert [6], [7] (e.g. for designing low-level support) to theoreticians [8], [9] (e.g. for deciding the optimum trade-off between numerical remedies and system recoveries) through numerical analysis experts [10], [11].

In this study we consider an actual fully-featured parallel sparse hybrid (direct/iterative) linear solver, MAPHYS[1] [12], and we propose numerical remedies to design a resilient version of the solver. The solver being hybrid, we focus in this study on the iterative solution step, which is often the dominant step in practice. We furthermore assume that a separate mechanism ensures fault detection and that a system layer provides support for setting back the environment (processes, ...) in a running state. The present manuscript therefore focuses on (and only on) strategies for recovering lost data *after* the fault has been detected (a separate concern out of the scope of this study) and *once* the system is back in a running state (another separate concern not studied

[1] https://project.inria.fr/maphys/

here either). The numerical remedies we propose are twofold. Whenever possible, we exploit data redundancy between processes from the solver to perform an exact recovery through clever copies over processes. Otherwise, data that has been lost and is no longer available on any process is recovered through a so-called interpolation-restart (IR) mechanism. This mechanism is derived from [1] by carefully taking into account the properties of the considered hybrid solver.

To successfully deal with faults in parallel distributed environments, three main issues are commonly addressed as follows. The first issue is how to prevent the fault from affecting the whole system. Secondly one must provide a system mechanism to replace the failed component and finally a strategy to recover lost data. As mentioned above, the first two challenges are a separate problem out of the scope of this paper. Our focus is to provide remedies for recovering lost data assuming that the fault is successfully handled and the failed component replaced. For that purpose, in previous works [1], [13], we have developed a new class of numerical fault-tolerant algorithms at application level called interpolation-restart (IR) approaches, that do not require extra resources, i.e., computational unit or computing time, when no fault occurs. The IR approach consists in extracting relevant information from available data after a fault. After data extraction, a well chosen part of missing data is regenerated through interpolation strategies to constitute meaningful inputs to numerically restart the algorithm. The contribution of this paper consists of:

- designing a new IR strategy for sparse hybrid methods;
- proposing a parallel implementation of a fully-featured resilient hybrid solver by extending MAPHYS;
- assessing the numerical behavior at large-scale (up to $12,288$ CPU cores) on real-life problems.

The remainder of this paper is organized as follows: Section II presents our context in more details. In Section III we explain how we designed a resilient version of our sparse hybrid solver. In Section IV we present numerical experiments that assess the robustness and the overhead of the resilient numerical remedies that have been proposed followed by some conclusions and prospects in Section V.

II. BACKGROUND

We present in this section the building on which blocks we rely in the paper. Section II-A presents the context in more details. Section II-B presents the basics of domain decomposition Schur complement methods, which are common to most sparse hybrid solvers [2], [3], [4], [5]. Sections II-C and II-D present the method used for preconditioning the reduced system in MAPHYS and the parallel implementation of the solver, respectively. These properties will be essential in designing an efficient hybrid solver (Section III): we will exploit the parallel design to benefit from available redundancy as well as the preconditioner to design an efficient IR method.

A. Context

MAPHYS mainly consists of four steps: *(1)* partitioning of the matrix adjacency graph, *(2)* local factorization of interiors and computation of the local Schur complement, *(3)* computation of the preconditioner and *(4)* the solve step which itself consists of two sub-steps: *(4a)* iterative solving of the linear system associated with the interface and *(4b)* back substitution on the interiors. As the object of the present study is the iterative part of solver, we focus on designing a resilient version of step (4a). The first three steps may be viewed as preprocessing steps for forming the reduced system and computing the associated preconditioner.

Taking into consideration the fault model introduced in [10] which distinguishes three categories of data: *computational environment*, *static data* and *dynamic data*, we assume in this paper, that static data are all data available before the iterative solution step. Furthermore, we assume that the Schur complement, the preconditioner and the right-hand side are static, while the current iterate and any other data generated during the step *(4a)* are the dynamic data. We recall that a fully resilient strategy must provide mechanisms to change any failed process, processor, core or node as well as strategies to retrieve the computational environment and lost data. However in this paper, we focus on numerical strategies for retrieving meaningful dynamic data. For this purpose, we assume that there is a mechanism to replace lost computational resources, restore the computational environment and load static data (for instance from disk). We thus address the following problem: *How can lost dynamic data be recovered when a fault occurs in the iterative solve step of* MAPHYS*?*

In this context, we simulate a process fault by overwriting its dynamic data (an actual fault injection is an orthogonal problem out of the present scope) and we then use either data redundancy or IR techniques (or both) to regenerate the lost dynamic data. We simulate the crash of one single process (denoted *single process fault* in the rest of the paper) and the crash of multiple concurrent processes that are neighbors with respect to the domain decomposition (denoted *neighbor processes fault case*). When a single fault occurs, we exploit data redundancy intrinsic to MAPHYS to retrieve all lost dynamic data. When faults are simultaneously injected into neighbor processes, part of the data is definitely lost on all processes; the strategy then consists in exploiting data redundancy wherever possible enhanced with an IR scheme to regenerate definitely lost dynamic data.

B. Domain decomposition Schur complement methods

This section describes how to rely on the Schur complement method to solve linear systems. Let us assume that the problem is subdivided in subdomains. We distinguish two types of unknowns: the interior unknowns $x_{\mathcal{I}}$ and the interface unknowns x_Γ. With respect to such a decomposition, the linear system $\mathcal{A}x = b$ in the corresponding block reordered form reads:

$$\begin{pmatrix} \mathcal{A}_{\mathcal{I}\mathcal{I}} & \mathcal{A}_{\mathcal{I}\Gamma} \\ \mathcal{A}_{\Gamma\mathcal{I}} & \mathcal{A}_{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} x_{\mathcal{I}} \\ x_\Gamma \end{pmatrix} = \begin{pmatrix} b_{\mathcal{I}} \\ b_\Gamma \end{pmatrix}. \quad (1)$$

Eliminating $x_{\mathcal{I}}$ from the second block-row of Equation (1) leads to the reduced system

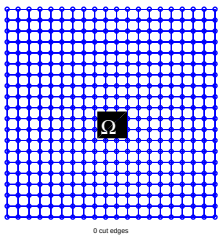
$$\mathcal{S}x_\Gamma = f, \quad (2)$$

where

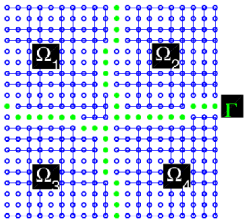
$$\mathcal{S} = \mathcal{A}_{\Gamma\Gamma} - \mathcal{A}_{\Gamma\mathcal{I}}\mathcal{A}_{\mathcal{I}\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I}\Gamma} \text{ and } f = b_\Gamma - \mathcal{A}_{\Gamma\mathcal{I}}\mathcal{A}_{\mathcal{I}\mathcal{I}}^{-1}b_{\mathcal{I}}. \quad (3)$$

The matrix $\mathcal{S}$ is referred to as the *Schur complement* matrix. This reformulation leads to a general strategy for solving (1). A sparse direct method is used to apply $\mathcal{A}_{\mathcal{I}\mathcal{I}}^{-1}$ and form (2). This latter system associated with the Schur complement is solved with an iterative method on which we will focus. Once x_Γ is known, $x_{\mathcal{I}}$ can be computed with one additional direct back-solve step.

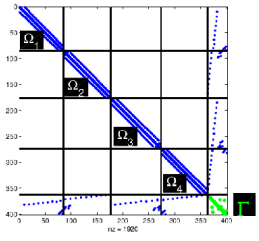
For the sake of simplicity, we assume that $\mathcal{A}$ is symmetric in pattern and we denote $G = \{V, E\}$ the adjacency graph associated with $\mathcal{A}$. In this graph, each vertex is associated with a row or column of the matrix $\mathcal{A}$. There exists an edge between the vertices p and q if and only if the entry $a_{p,q}$ is non zero. Figure 1a shows such an adjacency graph.



(a) Graph representation.



(b) Domain decomposition.



(c) Block reordered matrix.

Figure 1: Domain decomposition into four subdomains $\Omega_1, \ldots, \Omega_4$. The initial domain Ω may be algebraically represented with the graph G associated with the sparsity pattern of matrix $\mathcal{A}$ (a). The local interiors $\mathcal{I}_1, \ldots, \mathcal{I}_N$ form a partition of the interior $\mathcal{I} = \sqcup\mathcal{I}_p$ (blue vertices in (b)). They interact with each other through the interface Γ (green vertices in (b)). The block reordered matrix (c) has a block diagonal structure for the variables associated with the interior $\mathcal{A}_{\mathcal{I}\mathcal{I}}$.

A non-overlapping decomposition of a domain Ω into subdomains $\Omega_1, \ldots, \Omega_N$ corresponds to a *vertex split* of the graph G. V is decomposed into N subsets $V_1, \ldots, V_N$ of interiors $\mathcal{I}_1, \ldots, \mathcal{I}_N$ and boundaries $\Gamma_1, ..., \Gamma_N$ (algebraic view). Figure 1b depicts the algebraic view of the domain decomposition into four subdomains.

Local interiors are disjoint and form a partition

of the interior $\mathcal{I} = \sqcup \mathcal{I}_p$ (blue vertices in Figure 1b). Two subdomains Ω_p and Ω_q may share part of their interface ($\Gamma_p \bigcap \Gamma_q \neq \emptyset$), such as Ω_1 and Ω_2 in Figure 1b which share eleven vertices. Altogether the local boundaries form the overall interface $\Gamma = \cup \Gamma_p$ (green vertices in Figure 1b), which is thus not necessarily a disjoint union. One may note that the local interiors and the (global) interface form a partition of the original graph: $V = \Gamma \bigsqcup \sqcup \mathcal{I}_p$ (the original graph in Figure 1a is exactly covered with blue and green points in Figure 1b).

Because interior vertices are only connected to vertices of their subset (either on the interior or on the boundary), matrix $\mathcal{A}_{\mathcal{I}\mathcal{I}}$ associated with the interior has a block diagonal structure, as shown in Figure 1c. Each diagonal block $\mathcal{A}_{\mathcal{I}_p\mathcal{I}_p}$ corresponds to a local interior. On the other hand, to handle shared interfaces with a local approach, the coefficients on the interface may be weighted so that the sum of the coefficients on the local interface submatrices are equal to one. To that end, we introduce the *weighted local interface* matrix $\mathcal{A}^w_{\Gamma_p\Gamma_p}$ which satisfies $\mathcal{A}_{\Gamma\Gamma} = \sum_{p=1}^N \mathcal{R}^T_{\Gamma_p} \mathcal{A}^w_{\Gamma_p\Gamma_p} \mathcal{R}_{\Gamma_p}$, where $\mathcal{R}_{\Gamma_p} : \Gamma \to \Gamma_p$ is the canonical point-wise restriction which maps full vectors defined on Γ into vectors defined on Γ_p. For instance, the ten points on the red interface shared by subdomains Ω_1 and Ω_2 in Figure 1b may get a weight $\frac{1}{2}$ as they are shared by two subdomains. In matrix terms, a subdomain Ω_p may then be represented by the *local matrix* $\mathcal{A}_p$ defined by

$$\mathcal{A}_p = \begin{pmatrix} \mathcal{A}_{\mathcal{I}_p\mathcal{I}_p} & \mathcal{A}_{\mathcal{I}_p\Gamma_p} \\ \mathcal{A}_{\Gamma_p\mathcal{I}_p} & \mathcal{A}^w_{\Gamma_p\Gamma_p} \end{pmatrix}. \tag{4}$$

The global Schur complement matrix S from (3) can then be written as the sum of elementary matrices

$$\mathcal{S} = \sum_{p=1}^N \mathcal{R}^T_{\Gamma_p} \mathcal{S}_p \mathcal{R}_{\Gamma_p},$$

where

$$\mathcal{S}_p = \mathcal{A}^w_{\Gamma_p\Gamma_p} - \mathcal{A}_{\Gamma_p\mathcal{I}_p} \mathcal{A}^{-1}_{\mathcal{I}_p\mathcal{I}_p} \mathcal{A}_{\mathcal{I}_p\Gamma_p} \tag{5}$$

is a *local Schur complement* associated with the subdomain Ω_p. This local expression allows for computing local Schur complements independently from each other.

C. Additive Schwarz preconditioning of the Schur Complement

The preconditioner originally proposed in [14] aims at being highly parallel. The most straightforward method for building a preconditioner from the information provided by the local Schur complements would consist of performing their respective inversion. Such a preconditioner would write $M_{NN} = \sum_{p=1}^N \mathcal{R}^T_{\Gamma_p} \mathcal{S}_p^{-1} \mathcal{R}_{\Gamma_p}$ and corresponds to a Neumann-Neumann [15] preconditioner applied to the Schur complement. However, even in the (symmetric definite positive) SPD case, the local Schur complement can be singular and additional computation are required to form the preconditioner. Therefore, we consider the *local assembled Schur complement* $\bar{\mathcal{S}}_p = \mathcal{R}_{\Gamma_p} S \mathcal{R}^T_{\Gamma_p}$, which corresponds to the restriction of the global Schur complement to the interface Γ_p and which cannot be singular in the SPD case (as $\mathcal{S}$ is SPD as well [14]). This preconditioner reads:

$$M_{AS} = \sum_{p=1}^N \mathcal{R}^T_{\Gamma_p} \bar{\mathcal{S}}_p^{-1} \mathcal{R}_{\Gamma_p}. \tag{6}$$

This local assembled Schur complement can be built from the local Schur complements $\mathcal{S}_p$ by assembling their diagonal blocks. If we consider a planar graph partitioned into horizontal strips (1D decomposition), the resulting Schur complement matrix has a block tridiagonal structure as depicted as follows

$$\begin{pmatrix} \ddots & & & & \\ & \boxed{\begin{matrix} S_{i-1,i-1} & S_{i-1,i} \\ S_{i,i-1} & S_{i,i} \end{matrix}} & \boxed{S_{i,i+1}} & & \\ & \boxed{\begin{matrix} S_{i+1,i} \end{matrix}} & \boxed{S_{i+1,i+1}} & & \\ & & & \ddots & \end{pmatrix}.$$

For that particular structure of $\mathcal{S}$, the submatrices in boxes correspond to the $\bar{\mathcal{S}}_p$. Such diagonal blocks, which overlap, are similar to the classical block overlap of the Schwarz method when written in matrix form for a 1D decomposition. Similar ideas have been developed in a pure algebraic context in earlier papers [16], [17] for the solution of general sparse linear systems. Because of this link, the preconditioner defined by (6) is referred to as algebraic additive Schwarz preconditioner for the Schur complement. This is the preconditioner we deal with in the rest of this study.

D. Parallel implementation

Given a linear system $\mathcal{A}x = b$ in a parallel distributed environment, MAPHYS proceeds as follows. It relies on graph partitioning tools such SCOTCH [18] or METIS [19] to partition the related adjacency matrix, which leads to subgraphs. These subgraphs correspond to subdomains while shared edges correspond to interface unknowns as early depicted in Figure 1a. Each subgraph interior is mapped to only one process whereas each local interface is replicated on each process connected to it.

With this data distribution, each process p concurrently eliminates the internal unknowns using a sparse direct method. The factorizations of the local interiors are performed by each process independently from each other and require no communication. The global linear system to solve in parallel is thus reduced to the linear system associated with the interface, which is solved with an iterative method.

For the computation of the Schur complement, each process computes $\mathcal{S}_p$ defined in Equation (7) in parallel using PASTIX [20] (or MUMPS [21]), which is a sparse direct solver that also computes:

$$\mathcal{S}_p = \mathcal{A}_{\Gamma_p\Gamma_p} - \mathcal{A}_{\Gamma_p\mathcal{I}_p}\mathcal{A}^{-1}_{\mathcal{I}_p\mathcal{I}_p}\mathcal{A}_{\mathcal{I}_p\Gamma_p}. \qquad (7)$$

Once the local Schur complements have been computed, each process communicates with its neighbors (in the graph) to assemble its local Schur complement $\bar{\mathcal{S}}_p$ (a dense matrix) and perform its factorization using the Intel MKL library. This step only requires a few point-to-point communications. Finally, the last step is the iterative solution of the interface problem (2). For that purpose, we use Krylov method subroutines developed in [22].

III. A RESILIENT SPARSE HYBRID LINEAR SOLVER

We have described the design of MAPHYS in the previous section. Here we explain how we exploit properties intrinsic to MAPHYS in order to design a resilient extension. While all the proposed design and implementation has been done within the MAPHYS package to be able to process any algebraic problem, for the sake of exposition, we consider the 1D domain decomposition depicted in Figure 2 to describe how data are allocated over processes. Without loss of generality, we will also use this example for illustrating all the recovery mechanisms throughout this section. In this example, the domain is decomposed in four subdomains Ω_1, Ω_2, Ω_3 and Ω_4 with the associated interface $\Gamma = \Gamma_A \sqcup \Gamma_B \sqcup \Gamma_C$. Interface Γ_A is shared by subdomains Ω_1 and Ω_2, Γ_B by Ω_2 and Ω_3, Γ_C by Ω_3 and Ω_4.

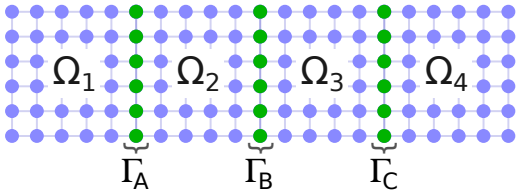


Figure 2: 1D domain decomposition. The originally rectangular domain is partitioned into four subdomains with three interfaces.

With such a decomposition, the linear system associated with the Schur complement is described by the following equation

$$\begin{pmatrix} \mathcal{S}_{A,A} & \mathcal{S}_{A,B} & \\ \mathcal{S}_{B,A} & \mathcal{S}_{B,B} & \mathcal{S}_{B,C} \\ & \mathcal{S}_{C,B} & \mathcal{S}_{C,C} \end{pmatrix} \begin{pmatrix} x_{\Gamma_A} \\ x_{\Gamma_B} \\ x_{\Gamma_C} \end{pmatrix} = \begin{pmatrix} f_A \\ f_B \\ f_C \end{pmatrix},$$

where x_{Γ_A}, x_{Γ_B} and x_{Γ_C} are the unknowns associated with the interfaces Γ_A, Γ_B and Γ_C, respectively. Following a classical parallel implementation of finite element substructuring approaches, each submatrix described by Equation (4) associated with a given subdomain is allocated to a process. A direct consequence is that each process can compute its local Schur complement and the unknowns associated with a given interface are naturally replicated on the processes sharing this interface. This is the choice made in MAPHYS and, with respect to this choice, processes p_1, p_2, p_3, p_4, are mapped on Γ_A, $\Gamma_A \sqcup \Gamma_B$, $\Gamma_B \sqcup \Gamma_C$ and Γ_C, respectively. Consequently, x_{Γ_A} is replicated on processes p_1 and p_2, x_{Γ_B} is replicated on processes p_2 and p_3, and so on. During the parallel solution of the Schur complement system, the Krylov solver computes redundantly and consistently dynamic data associated with these replicated unknowns. As a Krylov subspace method, we use FGMRES [22] presented in Algorithm 1, instead of GMRES in the released version of MAPHYS, because it has attractive properties for resilience [11], [1].

Given an initial guess x^0, FGMRES computes the Krylov basis V_k and the search space Z_k (Algorithm 1, lines 5 to 14), and the Hessenberg matrix H_k (line 15). Once these three elements

Algorithm 1 FGMRES, given a matrix $\mathcal{A}$, a preconditioner $\mathcal{M}$, a right hand side b, and an initial guess $x^{(0)}$

1: Set the initial guess x^0;
2: **for** $k = 0, 1, \ldots,$ until convergence, **do**
3: $r_0 = b - \mathcal{A}x^0$; $\beta = \|r_0\|$
4: $v_1 = r_0/\|r_0\|$;
5: **for** $j = 1, \ldots, m$ **do**
6: $z_j = \mathcal{M}^{-1}v_j$
7: $w_j = \mathcal{A}z_j$
8: **for** $i = 1$ to j **do**
9: $h_{i,j} = v_i^T w_j$; $w_j = w_j - h_{i,j}v_i$
10: **end for**
11: $h_{j+1,j} = \|w_j\|$
12: If $(h_{j+1,j}) = 0$; $m = j$; goto 15
13: $v_{j+1} = w_j/h_{j+1,j}$
14: **end for**
15: Define the $(m+1) \times m$ upper Hessenberg matrix $\bar{H}_m$
16: Solve the least squares problem $y_m = \arg\min \|\beta e_1 - \bar{H}_m y\|$
17: Set $x^0 = x^0 + Z_m y_m$
18: **end for**

are computed, FGMRES solves a least squares problem $y_k = \arg\min \|\beta e_1 - \bar{H}_k y\|$ (line 16) then updates the current iterate $x = x^0 + Z_k y_k$ (line 17). It is important to note that the basis V_k, the search space Z_k and the Hessenberg matrix H_k are critical data because on the one hand, they are dynamic and on the other hand, they are essential for the computation of the current iterate. According to our assumptions, the basis V_k and Z_k are distributed, whereas the Hessenberg matrix is replicated on each process. For our 1D decomposition example, the block-row $V_{\Gamma_A,:}$ is replicated on processes p_1 and p_2, the block-row $V_{\Gamma_B,:}$ is replicated on processes p_2 and p_3, and the block-row $V_{\Gamma_C,:}$ is replicated on processes p_3 and p_4. The matrix Z_k is distributed in the same way.

A. Single fault cases

In this section, we explain the strategy for surviving single process faults in the iterative solve step of MAPHYS (step (4a)). One advantage of having redundant local interfaces is that dynamic data on each process is also computed on neighbor processes. So, when a single process fails, we retrieve all its dynamic data from its neighbors. Once all dynamic data are recovered, FGMRES iterations can continue with exactly the same data as before the fault. Consequently, the numerical behavior and the solution from the faulty execution is the same compared to the corresponding fault-free execution. Indeed, the unique penalty is the communication time to reconstitute lost data.

Let us come back to the 1D decomposition from Figure 2 to illustrate how to retrieve lost data. We present two examples. First we illustrate how to retrieve data when a process with only one neighbor fails. Second we illustrate the case of a process with two neighbors. For the first case, we assume that process p_1 fails. The Hessenberg matrix is retrieved from any process because it is a small matrix that is fully replicated to reduce the communication when forming and solving the least square problem. The block-rows $V_{\Gamma_A,:}$ and $Z_{\Gamma_A,:}$ are retrieved from process p_2. For the second case, we assume that a fault occurs on process p_2. The Hessenberg matrix is retrieved from any surviving process. The block-rows $V_{\Gamma_A,:}$ and $Z_{\Gamma_A,:}$ are retrieved from process p_1 while the block-rows $V_{\Gamma_B,:}$ and $Z_{\Gamma_B,:}$ are retrieved from process p_3. Once all lost data are retrieved, FGMRES iterations continue in the same state as before the fault, exhibiting the same numerical behavior as in the non faulty case.

B. Interpolation-restart strategy for the neighbor processes fault cases

When a fault occurs on neighboring processes, some data remain lost despite data redundancy. We describe how the IR strategy presented in [1] can be modified to take advantage of the features of

MAPHYS's preconditioner in order to efficiently survive neighbor processes faults.

To illustrate a fault on neighbor processes, we reconsider the example of a 1D decomposition illustrated in Figure 2 and we assume that processes p_2 and p_3 have both failed. In this latter case, it is not possible to retrieve the dynamic data associated with the interface Γ_B shared by the failed processes. We propose to use an interpolation strategy to regenerate the entries of $x_{\Gamma_B}^{(k)}$.

A first possibility is to use the linear interpolation (LI) strategy developed in [1]. By using the LI strategy, processes p_2 and p_3 solve the local linear system

$$\mathcal{S}_{B,B}x_{\Gamma_B} = f_B - \mathcal{S}_{B,A}x_{\Gamma_A}^{(k)} - \mathcal{S}_{B,C}x_{\Gamma_C}^{(k)}$$

to interpolate $x_{\Gamma_B}^{(k)}$. This direct application of the LI strategy requires the factorization of $\mathcal{S}_{B,B}$. However it is possible to design an interpolation-restart strategy that exploits the features of the MAPHYS preconditioner and consequently avoids the additional factorization. As discussed in Section II-D, the factorization of the local assembled Schur complement is the main building block of the preconditioner. In our 1D example, the local preconditioner of p_2 is $\bar{\mathcal{S}}_{p_2}^{-1}$ with

$$\bar{\mathcal{S}}_{p_2} = \begin{pmatrix} \mathcal{S}_{A,A} & \mathcal{S}_{A,B} \\ \mathcal{S}_{B,A} & \mathcal{S}_{B,B} \end{pmatrix}.$$

These factorizations of local assembled Schur complements are computed before the iterative solve step and are considered as static in our model. Consequently they are available after a fault. Based on these matrix factorizations we designed an interpolation variant referred to as LI^{AS} (AS standing for Additional Schwarz). With the LI^{AS} approach, process p_2 solves the local linear system

$$\begin{pmatrix} \mathcal{S}_{A,A} & \mathcal{S}_{A,B} \\ \mathcal{S}_{B,A} & \mathcal{S}_{B,B} \end{pmatrix} \begin{pmatrix} x_{\Gamma_A} \\ x_{\Gamma_B} \end{pmatrix} = \begin{pmatrix} f_A \\ f_B - \mathcal{S}_{B,C}x_{\Gamma_C}^{(k)} \end{pmatrix}$$

in the same way, p_3 solves

$$\begin{pmatrix} \mathcal{S}_{B,B} & \mathcal{S}_{B,C} \\ \mathcal{S}_{C,B} & \mathcal{S}_{C,C} \end{pmatrix} \begin{pmatrix} x_{\Gamma_B} \\ x_{\Gamma_C} \end{pmatrix} = \begin{pmatrix} f_B - \mathcal{S}_{B,A}x_{\Gamma_A}^{(k)} \\ f_C \end{pmatrix}.$$

Consequently, different values of x_{Γ_B} are available on p_2 and p_3 and new entries of x_{Γ_A} and x_{Γ_C} are computed. The entries of x_{Γ_A} and x_{Γ_C} computed are not used since they are available on p_1 and p_4, respectively. Finally we make the value of x_{Γ_B} consistent on p_2 and p_3 by simply averaging these values,

$$x_{\Gamma_B}^{(LI^{AS})} = \frac{1}{2}(x_{\Gamma_B 2}^{(LI^{AS})} + x_{\Gamma_B 3}^{(LI^{AS})}),$$

where $x_{\Gamma_B 2}^{(LI^{AS})}$ and $x_{\Gamma_B 3}^{(LI^{AS})}$ are the entries of $x_{\Gamma_B}^{(k)}$ regenerated by p_2 and p_3 respectively. Once all the missing entries are computed, the current iterate thus regenerated is used as an initial guess to restart FGMRES.

The presented LI^{AS} strategy naturally extends to general decompositions based on the same idea and can be summarized as follows into four main steps:

1) *Computation of non faulty entries*: All still alive processes compute the entries of the current iterate on which they are mapped.
2) *Computation of right-hand side contribution*: The neighbors of the failed processes compute the contributions required to update the right-hand sides of the interpolation linear systems. The computation of the right-hand sides associated with the linear interpolation may require significant communication time depending on the number of neighbors of the failed processes. Indeed, to update the right-hand side, a failed process needs contributions from all its neighbors. On the other hand, neighbors participate in the interpolation by computing locally matrix vector multiplications required for right-hand side update.
3) *Communication*: Each failed process retrieves lost entries of the current iterate except the entries definitively lost, which are shared by failed processes. At the same time, failed processes receive the contributions from neighbors to update the right-hand side associated with the local interpolation.
4) *Interpolation*: Each failed process solves the interpolation linear system, and failed processes communicate to maintain the same value of the interpolated entries. This is essentially reduced to applying the local components of the preconditioner.

At the end of the these four steps, a consistent state is obtained and FGMRES can be restarted with the interpolated iterate as a new initial guess. In contrast to the single process fault case, the numerical behavior is no longer the same as the non faulty case anymore. We assess its effects in the next section.

IV. Experimental results

In this section, we present experimental results for the resilient sparse hybrid linear solver proposed above. As explained in Section II-A, we recall that instead of actually crashing a process, we simulate its crash by deleting its dynamic data, since we do not tackle the systems mechanism issues in the present study. Therefore, the cost of resetting the system in a coherent state (such as creating a new process and adapting communicators) and retrieving static data is not taken into account. In the single process fault case, we assess only the communication time required to retrieve the lost dynamic data. In the neighbor processes fault case, we present the numerical behavior of LI^{AS} as well as a performance analysis.

We have performed extensive experiments and only report the behavior observed on a few examples that are representative of our observations. For all the experiments, the convergence threshold on the scaled residual is set as ϵ = 1e-10. We report results for the two matrices described in Table I. `Matrix211` is a non-symmetric matrix of dimension 0.8M from the fusion energy study code M3D-C[2]. Importantly, previous studies have demonstrated that iterative methods may suffer from slow convergence [23] for processing matrix `Matrix211` and that hybrid methods are likely to be dominated by the iterative step. Matrix `Nachos4M` is of order 4M and comes from a numerical simulation of the exposure of a full body to electromagnetic waves with discontinuous Galerkin method.

Matrix	`Matrix211`	`Nachos4M`
N	0.8M	4M
Nnz	129M	256M
Symmetry	no	no

Table I: Description of the matrices considered for experimentation.

The experiments were performed on the `Hopper` platform[3]. Each node on `Hopper` has two twelve-core AMD 'MagnyCours' 2.1-GHz processors. MaPhyS as well as the proposed resilient extension have been written in Fortran 90 and support two levels of parallelism (MPI + Thread). As discussed in Section II-D, MaPhyS is modular and relies on state-of-the-art packages for performing domain decomposition and direct factorization. For the experiments, we have used the METIS package, the PaStiX package and the Intel Mkl libraries. Once the problem is partitioned, each subdomain is mapped to one process. We use three threads per process and eight processes per node, which leads to a total of 24 threads per node in order to exploit the 24 cores on each node. For each matrix and a given number of processes, we performed many experiments by varying the iteration when the fault is injected with only one fault by experiment, and we report the average overhead.

We note T_r and T_f the time spent in the iterative solution step (step (4a)) without fault (reference) and with fault, respectively. The overhead then corresponds to: $\frac{T_f - T_r}{T_r}$.

A. Single fault cases

In this section, we present results for single process fault cases. We recall that in this case, the numerical behavior is the same as the non faulty execution and the overhead is only due to communications. To solve the linear systems associated with `Matrix211`, we vary the number of cores from 384 to 3,072 (Table II). Regardless of the number of cores, the overhead induced by the fault recovery strategy remains low. One can also observe the decrease of the overhead when the number of cores varies between 348 and 1,536. Indeed, when the number of cores increases, the volume of data associated with each process decreases. This leads to the decrease of the volume of data loss when a fault occurs. However with 3,072 cores, the overhead increases. This is due to the limitation of the gain associated with the increase of the number of processes. On the other hand, according to the size of the matrix, beyond a given number of processes, the fault recovery involves many processes. This may be penalizing because of MPI communication synchronization.

Nb of cores	384	768	1,536	3,072
Overhead	2.10%	1.18%	0.05%	0.38%

Table II: Variation of the overhead in the case of a single process fault while increasing the number of cores using `Matrix211`.

If we consider the result of `Nachos4M` presented in Table III, one can observe that even

[2]Center for Extended MHD Modeling (CEMM) URL: http://w3.pppl.gov/cemm/

[3]https://www.nersc.gov/users/computational-systems/hopper/configuration/compute-nodes/

with 12,288 cores, the overhead keeps decreasing because Nachos4M has a larger size. Furthermore, since the size of Nachos4M allows us to exploit larger numbers of processes, the induced overheads are very low, which demonstrate the potential of such strategies for large-scale problems.

Nb of cores	1,536	3,072	6,144	12,288
Overhead	0.84%	0.82%	0.76%	0.02%

Table III: Variation of the overhead in the case of a single process fault while increasing the number of cores using Nachos4M.

B. Neighbor processes fault cases

In this section, we present results for the LI^{AS} strategy designed to handle neighbor processes faults. We recall that LI^{AS} exploits data redundancy to retrieve available entries from surviving neighbors, before interpolating missing entries taking advantage of the additive Schwarz preconditioner. The overhead of the LI^{AS} strategy includes the communication time to retrieve available entries from surviving neighbors, the computational time to interpolate missing entries and the overhead induced by a possible numerical penalty. The numerical penalty may be induced by the quality of interpolated entries and the necessity to restart after a neighbor processes fault. The numerical penalty often leads to additional iterations, which may increase the computational time. The results for Matrix211 is reported in Table IV. With 384 cores, we have an overhead of 3.65%, but with the increase of the number of cores, the overhead decreases significantly down to 0.12%.

Nb of cores	384	768	1,536	3,072
Overhead	3.65%	1.31%	0.12%	0.45%

Table IV: Variation of the overhead in the case of neighbor processes fault while increasing the number of cores using Matrix211.

Even in the case of neighbor processes fault, the overhead associated with Nachos4M remains very low, from 1.70% down to 0.06%. This demonstrates again the attractive potential of our strategies for large-scale problems.

Nb of cores	1,536	3,072	6,144	12,288
Overhead	1.70%	1.26%	0.67%	0.06%

Table V: Variation of the overhead in the case of neighbor processes fault while increasing the number of cores using Nachos4M.

V. Concluding remarks and prospects

The main objective of this paper was to combine implementation and numerical features to design a resilient solution for large sparse linear systems on large massively parallel platforms. For that purpose, we have considered the fully-featured parallel sparse hybrid solver MaPhyS. We have exploited the solver properties to design two different resilient solutions for the iterative solve step: one to recover from single faults and another one extended to survive faults on neighbor processes.

In the case of a single fault, we have exploited the natural data redundancy to retrieve all dynamic data from neighbors. Once all dynamic data are recovered, the iterations continue with exactly the same data as before the fault. This solution only requires communications to reconstitute lost data. This solution has no numerical penalty so it exhibits the same convergence behavior as a fault-free execution. In the case of a fault on neighbor processes, we have designed the LI^{AS} strategy which takes advantage of the features of MaPhyS's preconditioner so that it does not require any additional factorization. All the experiments show that our strategies for both single process fault and neighbor processes fault have a very low overhead.

We have developed a numerical resilience approach for an algebraic domain decomposition technique. The same approach does apply to substructuring classical non-overlapping domain decompositions where the redundancy is naturally implemented in a finite element framework. These strategies can also be extended and applied to many classical domain decomposition methods for PDE solution. Finally, we assessed the effectiveness of our resilient algorithms by simulating process crashes. This study motivates the design of a full resilient parallel hybrid solver with system fault tolerant supports such ULFM [9].

References

[1] E. Agullo, L. Giraud, A. Guermouche, J. Roman, and M. Zounon, "Towards resilient parallel linear Krylov solvers: recover-restart strategies," Inria, Research Report RR-8324, July 2013, preliminary version of a paper to appear in Numerical Linear Algebra with Applications.

[2] I. Yamazaki and X. S. Li, "On techniques to improve robustness and scalability of a parallel hybrid linear solver," in *VECPAR*, 2010, pp. 421–434.

[3] S. Rajamanickam, E. G. Boman, and M. A. Heroux, "ShyLU: A hybrid-hybrid solver for multicore platforms," *Parallel and Distributed Processing Symposium, International*, vol. 0, pp. 631–643, 2012.

[4] L. Giraud and A. Haidar, "Parallel algebraic hybrid solvers for large 3D convection-diffusion problems." *Numerical Algorithms*, vol. 51, no. 2, pp. 151–177, 2009.

[5] J. Gaidamour and P. Hénon, "HIPS: a parallel hybrid direct/iterative solver based on a schur complement approach," *Proceedings of PMAA*, 2008.

[6] C.-C. Li and W. Fuchs, "Catch-compiler-assisted techniques for checkpointing," in *Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium*, June 1990, pp. 74–81.

[7] G. E. Fagg and J. Dongarra, "FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world," in *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*. London, UK: Springer-Verlag, 2000, pp. 346–353.

[8] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC'11. New York, NY, USA: ACM, 2011, pp. 33:1–33:11. http://doi.acm.org/10.1145/2063384.2063428

[9] K. Teranishi and M. A. Heroux, "Toward local failure local recovery resilience model using MPI-ULFM," in *Proceedings of the 21st European MPI Users' Group Meeting*, ser. EuroMPI/ASIA '14. New York, NY, USA: ACM, 2014, pp. 51:51–51:56. http://doi.acm.org/10.1145/2642769.2642774

[10] J. Langou, Z. Chen, G. Bosilca, and J. Dongarra, "Recovery Patterns for Iterative Methods in a Parallel Unstable Environment," *SIAM J. Sci. Comput.*, vol. 30, pp. 102–116, November 2007.

[11] P. G. Bridges, K. B. Ferreira, M. A. Heroux, and M. Hoemmen, "Fault-tolerant linear solvers via selective reliability," *CoRR*, vol. abs/1206.1390, 2012.

[12] E. Agullo, L. Giraud, A. Guermouche, and J. Roman, "Parallel hierarchical hybrid linear solvers for emerging computing platforms," *Compte Rendu de l'Académie des Sciences - Mécanique*, vol. 339, no. 2-3, pp. 96–105, 2011.

[13] E. Agullo, L. Giraud, P. Salas, and M. Zounon, "On resiliency in some parallel eigensolvers," Inria, Research Report 8625, 2015.

[14] L. M. Carvalho, L. Giraud, and G. Meurant, "Local preconditioners for two-level non-overlapping domain decomposition methods," *Numerical Linear Algebra with Applications*, vol. 8, no. 4, pp. 207–227, 2001.

[15] Y.-H. De Roeck and P. Le Tallec, "Analysis and test of a local domain decomposition preconditioner," in *Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, R. Glowinski, Y. Kuznetsov, G. Meurant, J. Périaux, and O. Widlund, Eds. SIAM, Philadelphia, PA, 1991, pp. 112–128.

[16] X.-C. Cai and Y. Saad, "Overlapping domain decomposition algorithms for general sparse matrices," *Numerical Linear Algebra with Applications*, vol. 3, pp. 221–237, 1996.

[17] G. Radicati and Y. Robert, "Parallel conjugate gradient-like algorithms for solving nonsymmetric linear systems on a vector multiprocessor," *Parallel Computing*, vol. 11, pp. 223–239, 1989.

[18] C. Chevalier and F. Pellegrini, "PT-SCOTCH: a tool for efficient parallel graph ordering," *Parallel Computing*, vol. 34, no. 6-8, 2008.

[19] G. Karypis and V. Kumar, "Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0," 1995.

[20] P. Hénon, P. Ramet, and J. Roman, "PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems," *Parallel Computing*, vol. 28, no. 2, pp. 301–321, Jan. 2002.

[21] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, "Hybrid scheduling for the parallel solution of linear systems," *Parallel Computing*, vol. 32, no. 2, pp. 136–156, 2006.

[22] V. Frayssé, L. Giraud, and S. Gratton, "Algorithm 881: A set of FGMRES routines for real and complex arithmetics on high performance computers," *ACM Transactions on Mathematical Software*, vol. 35, no. 2, pp. 1–12, 2008.

[23] X. S. Li, M. Shao, I. Yamazaki, and E. Ng, "Factorization-based sparse solvers and preconditioners," in *Journal of Physics: Conference Series*, vol. 180, no. 1. IOP Publishing, 2009, p. 012015.