SIAM J. SCI. COMPUT.
Vol. 43, No. 5, pp. C335–C357

CHRONOS: A GENERAL PURPOSE CLASSICAL AMG SOLVER FOR HIGH PERFORMANCE COMPUTING*

GIOVANNI ISOTTON†, MATTEO FRIGO†, NICOLÒ SPIEZIA†, AND CARLO JANNA‡

Abstract. The numerical simulation of physical systems has become in recent years a fundamental tool to perform analyses and predictions in several application fields, spanning from industry to the academy. As far as large-scale simulations are concerned, one of the most computationally expensive tasks is the solution of linear systems of equations arising from the discretization of the partial differential equations governing physical processes. This work presents Chronos, a collection of linear algebra functions specifically designed for the solution of large, sparse linear systems on massively parallel computers. Its emphasis is on modern, effective, and scalable Algebraic Multigrid (AMG) preconditioners for high performance computing (HPC). This work describes the numerical algorithms and the main structures of this software suite, especially from an implementation standpoint. Several numerical results arising from practical mechanics and fluid dynamics applications with hundreds of millions of unknowns are addressed and compared with other state-of-the-art linear solvers, proving Chronos's efficiency and robustness.

Key words. parallel computing, HPC, preconditioning, algebraic multigrid

AMS subject classifications. 65F08, 65F10, 65F50, 68W10

DOI. 10.1137/21M1398586

1. Introduction.

The solution of linear systems of equations is a central problem in a huge number of applications in both engineering and science. These problems are particularly important in the simulation of physical processes through the solution of partial differential equations (PDEs) or systems of PDEs. In large-scale simulations, the solution of linear systems can be the most expensive task, accounting for up to 99% of the total simulation cost.

In this work, we are interested in developing fast solution algorithms, suitable for high performance computing (HPC) for the linear system

$$A\mathbf{x} = \mathbf{b}, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ is the system matrix, $\mathbf{b}$ and $\mathbf{x} \in \mathbb{R}^n$ are the right-hand side and solution vector, respectively, and n is the number of equations. Although extension to general matrices is also possible, the present work restricts its focus to symmetric and positive definite (SPD) matrices, which are very common in most mechanics and fluid dynamics applications.

In current industrial applications, n can easily grow up to few hundred million of unknowns. On the other side, systems with billions of unknowns have also been solved in research experiments. The main difference between industrial problems and these huge research experiments is that the former are characterized by complex geometries, irregular discretizations, and heterogeneities in the matrix coefficients, while the latter are generally obtained by successive refinements of regular or quite regular grids. Despite their smaller size, problems arising from real-world applications are very challenging, and even having large computational resources may not be enough.

*Submitted to the journal's Software and High-Performance Computing section February 12, 2021; accepted for publication (in revised form) July 21, 2021; published electronically October 6, 2021.

https://doi.org/10.1137/21M1398586

†M^{3}E s.r.l., via Giambellino 7, 35129 Padova, Italy (g.isotton@m3eweb.it, m.frigo@m3eweb.it, n.spiezia@m3eweb.it).

‡Corresponding author. M^{3}E s.r.l., via Giambellino 7, 35129 Padova, Italy (c.janna@m3eweb.it).

There are several methods to solve (1), both direct [2, 29, 34] and iterative [3, 15, 37], giving excellent performance on parallel computers. The former are generally preferred in industrial applications as they are typically more robust and require no experience from the user. The main downside is that, especially in 3D problems, the matrix factors require a huge amount of memory, which becomes the limiting factor for large-scale simulations. This work is focused on iterative methods, more specifically on Algebraic Multigrid (AMG) preconditioning of iterative methods, because these present far fewer memory restrictions and are suitable for highly efficient parallel implementations. Moreover, in several practical cases, AMG preconditioning guarantees convergence in a number of iterations that do not depend or only slightly depend on the mesh size [12, 40, 43], a property of paramount importance for the extreme-size simulations that are foreseen in the near future. The main drawback of AMG preconditioning is that it is still far from being a black-box method, requiring an experienced user and sometimes a fine tuning of the set-up parameters. For most AMG solvers, a wrong set-up can easily lead to slow convergence or overly expensive preconditioners and, in the worst case, even to a failure in the solution [30].

The Chronos package (https://www.m3eweb.it/chronos/) is a library of iterative methods and AMG preconditioners designed for high performance platforms to solve severely ill-conditioned problems arising in real-world applications. To be effective on a wide range of different problems, Chronos allows for the choice of several options, from the adaptive generation of the operator near-kernel to the smoother selection, from coarsening to prolongation, all of this in the framework of the classical AMG method. In particular, it will be shown that BAMG interpolation, an interpolation whose coefficients are computed through least squares minimization, makes this AMG extremely effective also on mechanical problems without the need to use an aggregation based coarsening. From the implementation standpoint, Chronos has been developed for HPC, adopting a distributed sparse matrix storage scheme where smaller blocks, stored in compressed sparse row (CSR) format, are nested into a global CSR structure. This storage format, together with the adoption of nonblocking send/receive messages, allows for a high overlap between communications and computations, thus hiding data-transfer latency even for a relatively small number of local operations. Finally, Chronos has a strongly object-oriented design in order to be readily linked to other software, to be used as an innermost kernel in more complex approaches, such as block preconditioners for multiphysics [1, 17, 19, 35, 42], and to be easily modified to support emerging hardware.

The algorithms and methods presented in this work are not radically new but are rather known algorithms revisited and highly tuned for challenging industrial problems from various fields. Particular care has been spent in the general design of the library in order to make it easily maintainable and amenable to improvements without sacrificing performance. The benchmarks provided in the numerical experiments are not derived from regular discretization of artificial problems; instead they have been collected, from sources including other research/industrial groups, with the specific purpose of verifying Chronos against the widest possible selection of test cases.

The remainder of the paper is organized as follows. In section 2, the classical AMG method will be briefly outlined with a large emphasis on the specific numerical algorithms implemented to increase effectiveness. In section 3, the design of the library is accurately described especially from the implementation standpoint. The performance of Chronos is finally assessed in section 4 on a set of problems representative of a wide set of real-world problems with a comparison to other state-of-the-art packages. The paper closes with some concluding remarks and ideas for future work.

2. Classical algebraic multigrid framework. One of the strengths of this library is that it offers several options for each AMG component to allow the user to choose the best combination for any specific problem.

Any AMG method is generally built on three main components, whose interplay gives the effectiveness of the overall method:

- smoothing, where an inner preconditioner is applied to damp the high-frequency error components;
- coarsening, in which coarse level variables are chosen for the construction of the next level;
- interpolation, defining the transfer operator between coarse and fine variables.

In Chronos, a fourth component, borrowed from the context of bootstrap [8] and adaptive AMG [10, 11], is added to the above three and consists in a method to unveil hidden components of the near-kernel of the linear operator whenever they are not a priori available.

As mentioned before, in the present work we are focused on the classical AMG setting, and below we will briefly recall the basic concepts behind this method, referring the interested reader to more detailed and rigorous descriptions in the works [38, 40, 43]. For the sake of clarity, we restrict this introduction to a two-levels-only scheme, as the multilevel version can be readily obtained by recursion.

The first component that has to be set up in AMG is the smoother, a stationary iterative method responsible for eliminating the error components associated with large eigenvalues of A, also referred to as the high-frequency errors. The smoother is generally defined from a rough approximation of $A^{-1} \simeq M^{-1}$, and its operator is represented by the following equation:

$$S = I - \omega M^{-1} A, \tag{2}$$

where I is the identity matrix and ω is a relaxation factor to ensure

$$\omega \rho(M^{-1}A) < 2; \tag{3}$$

see, for instance, [18] for a short explanation. Generally, the smoother is given by a simple pointwise relaxation method such as (block) Jacobi or Gauss–Seidel, with the second one often preferred even though its use on parallel computers is not straightforward. Unlike other AMG packages such as BoomerAMG [24] or GAMG [6], where traditional smoothers like Gauss–Seidel or Chebyshev are selected by default, Chronos implements the adaptive Factorized Sparse Approximate Inverse (aFSAI) with the matrix M^{-1} taking the explicit form

$$M^{-1} = G^T G \tag{4}$$

with G lower triangular. This choice is dictated by its almost perfect strong scalability and by its proven robustness in real engineering problems [4, 26, 27]. Moreover, the cost of aFSAI application is usually much lower than that of Gauss–Seidel and Chebyshev since, generally, the number of nonzeros of M^{-1} is only 20–40% of the number of nonzeros of A.

The second component of AMG is the so-called coarse-grid correction (CGC), which is the A-orthogonal projection operation that should take care of the low-frequency components of the error. To build CGC in classical AMG, the unknowns of a given level are partitioned into fine and coarse (F/C), with coarse variables becoming the unknowns of the next level. The choice of coarse variables is crucial in AMG, as

it determines both the rate at which the problem size is reduced and the convergence of the method. Here, we rely on the concept of strength of connection (SoC); i.e., we associate to each edge of the adjacency graph of A a measure of its relative importance. Then, using SoC, we rank the graph connections and filter out those deemed less important. A maximum independent set (MIS) is finally constructed on the filtered SoC graph to determine coarse variables using the well-known and efficient PMIS algorithm [14].

To facilitate explanation, the system matrix is reordered according to F/C partitioning of unknowns with first fine variables and second coarse ones:

$$A = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{fc}^T & A_{cc} \end{bmatrix} \tag{5}$$

with A_{ff} and A_{cc} square $n_f \times n_f$ and $n_c \times n_c$ matrices, respectively. In classical AMG, the interpolation operator P is written as

$$P = \begin{bmatrix} W \\ I \end{bmatrix}, \tag{6}$$

where W is an $n_f \times n_c$ matrix containing the weights for coarse-to-fine variable interpolation. As the system matrix is SPD, the restriction operator R is defined through a Galerkin approach as the transpose of P, and the coarse level matrix A_c is simply given by the triple matrix product

$$A_c = P^T A P. \tag{7}$$

In practice, fast convergence, and rapid coarsening, i.e., high F/C ratios, are always desired, and the construction of effective interpolations is of paramount importance to conciliate these conflicting requirements.

Having defined all of the above components, the set-up phase of the two-level multigrid method is completed and the iteration matrix is given by

$$(S)^{\nu_2} \left(I - P {A_c}^{-1} P^T A\right) (S)^{\nu_1} \tag{8}$$

with ν_1 and ν_2 representing the number of smoothing steps performed before and after the CGC, respectively.

Algorithms 1 and 2 briefly report the general AMG set-up phase and application in a V-cycle, respectively, in a multilevel framework, where it is conventionally assumed that $A_0 = A$, $\mathbf{y}_0 = \mathbf{y}$, and $\mathbf{z}_0 = \mathbf{z}$. Details on all the computational kernels sketched in 1 and their parallel implementation will be discussed in the next sections/subsections.

2.1. Unveiling the operator near-kernel. The kernel (or null space) associated with the homogeneous discretized operator arising from the most common PDEs or systems of PDEs is generally a priori known. For instance, it is well known that the constant vector is the kernel for the Laplace operator and rigid body modes constitute the kernel for linear elasticity problems. The information needed to build these spaces, usually referred to as test spaces in the adaptive AMG terminology, is readily available to the user from nodal coordinates or other data retrievable from the discretization. However, the homogeneous operator kernel is only an approximation of the true near-kernel associated with the fully assembled matrix and does not take into account all the peculiarities of the problem such as boundary conditions or the strong

Algorithm 1 AMG Set-up

procedure AMG_SetUp(A_k)
  Define Ω_k as the set of the n_k vertices of the adjacency graph of A_k;
  if n_k is small enough to allow for a direct factorization **then**
    Compute $A_k = L_k L_k^T$;
  else
    Compute M_k such that $M_k^{-1} \simeq A_k^{-1}$;
    Define the smoother as $S_k = \left(I_k - \omega_k M_k^{-1} A_k\right)$;
    Partition Ω_k into the disjoint sets $\mathcal{C}_k$ and $\mathcal{F}_k$ via coarsening;
    Compute the prolongation matrix P_k from $\mathcal{C}_k$ to Ω_k;
    Compute the new coarse level matrix $A_{k+1} = P_k^T A_k P_k$;
    Call AMG_SetUp(A_{k+1});
  end if
end procedure

Algorithm 2 AMG application in a V-cycle

procedure AMG_Apply(A_k, $\mathbf{y}_k$, $\mathbf{z}_k$)
  if k is the last level **then**
    Solve $A_k \mathbf{z}_k = \mathbf{y}_k$ using L_k, the exact Cholesky factor of A_k;
  else
    Compute $\mathbf{s}_k$ by applying ν_1 smoothing steps to $A_k \mathbf{s}_k = \mathbf{y}_k$ with $\mathbf{s}_0 = \mathbf{0}$;
    Compute the residual $\mathbf{r}_k = \mathbf{y}_k - A_k \mathbf{s}_k$;
    Restrict the residual to the coarse grid $\mathbf{r}_{k+1} = P_k^T \mathbf{r}_k$;
    Call AMG_Apply($A_{k+1}, \mathbf{r}_{k+1}, \mathbf{d}_{k+1}$);
    Prolongate the correction to the fine grid $\mathbf{d}_k = P_k \mathbf{d}_{k+1}$;
    Update $\mathbf{s}_k \leftarrow \mathbf{s}_k + \mathbf{d}_k$;
    Compute $\mathbf{z}_k$ by applying ν_2 smoothing steps to $A_k \mathbf{z}_k = \mathbf{y}_k$ with $\mathbf{z}_0 = \mathbf{s}_k$;
  end if
end procedure

heterogeneities in the material properties that often arise in real-world problems. In many circumstances, a better test space can be obtained by simply modifying the initial near-kernel suggested by the PDE. In the adaptive AMG literature [8, 10, 11, 31], the test space is found by simply running a few smoothing steps over a random test space or the initial near-kernel whenever available. However, since the near-kernel of A is related to the smallest eigenpairs of the generalized eigenproblem [9]

$$A\boldsymbol{\varphi} = \lambda \boldsymbol{\varphi}, \tag{9}$$

a better way to extract an effective test space could be by relying on an iterative eigensolver [22]. In the present implementation, we opt for the simultaneous Rayleigh quotient minimization (SRQCG) [18], whose cost per iteration is only slightly higher than a smoothing step. By contrast, SRQCG can provide a much better approximation of the smallest eigenpairs especially if a good preconditioner is provided. Since an approximation of A^{-1} is already available through the smoother, we simply reuse the previously computed M^{-1} inside the SRQCG iteration.

From a theoretical standpoint, instead of solving (9), the test space should be computed by solving the generalized eigenproblem

$$A\boldsymbol{\varphi} = \lambda M \boldsymbol{\varphi}. \tag{10}$$

However, the SRQCG solution to (10) needs the multiplication of M by a vector which, due to our choice of M (4), would result in a forward and backward triangular solve whose parallelization may represent an algorithmic bottleneck.

Unfortunately, extracting the eigenpairs of (9) with high accuracy is generally more expensive than solving the original system (1). For this reason, to limit the set-up cost, we only approximately solve (9) with a predetermined and small number of SRQCG iterations. This simple strategy usually gives satisfactory results whenever an initial test space is not available or boundary conditions and heterogeneity exert a strong influence, such as in geomechanical problems. Another appealing idea, though not explored in this work, is bootstrapping [8, 10, 11], which consists in computing a relatively cheap AMG preconditioner from a tentative test space, and then using AMG itself to better uncover the near null space and rebuild a more effective AMG.

Operatively, once the test space is found, we compute an orthonormal basis of it and collect the basis vectors into a (skinny) matrix V that may be subsequently used for the calculation of SoC and the prolongation.

2.2. Strength of connection. The construction of the coarse problem in Chronos is based on the definition of an SoC matrix that is used to filter out weak connections from the adjacency graph of A. There are three different SoC definitions available in the library:

1. classical SoC,

$$s_{ij} = \frac{-a_{ij}}{\max(\min_{j\neq i} a_{ij}, \min_{j\neq i} a_{ji})}, \tag{11}$$

2. SoC based on strong couplings,

$$s_{ij} = \frac{|a_{ij}|}{\sqrt{a_{ii}a_{jj}}}, \tag{12}$$

3. affinity-based SoC,

$$s_{ij} = \frac{(\sum_k v_{ik}v_{jk})^2}{(\sum_k v_{ik}^2)(\sum_k v_{jk}^2)}, \tag{13}$$

where s_{ij} denotes the SoC between nodes i and j and a_{ij} and v_{ij} denote the entries in row i and column j of the matrices A and V, respectively. SoC (11) is particularly effective for Poisson-like problems where the system matrix is close to an M-matrix. SoC (12) is generally used in smoothed aggregation AMG [41] and usually gives good results in structural problems. Finally, SoC (13) has been introduced in [32] and, though requiring a rather expensive computation, is able to accurately capture anisotropies, as is shown in [33].

After SoC is computed for every pair of nodes, weak connections are eliminated to determine MISs whose nodes become the unknown in the next level. The more aggressively the connections are eliminated, the more nodes are left in the next level. There are two ways of controlling SoC filtering in Chronos:

1. by a threshold, the traditional way of filtering, where we simply drop connections below a given θ;
2. prescribing an average number of connections per node.

On one side, guaranteeing an average number of connections per node is trickier, since it requires a preliminary sorting of all the SoCs, but it ensures a more regular coarsening through levels with an almost constant coarsening ratio. Moreover, in

affinity-based SoC, the strength values usually lie in a narrow interval close to unity so that a proper choice of the drop threshold is almost impossible.

Finally, MIS construction is performed by using the PMIS strategy which is perfectly parallel and generally gives rise to lower complexities than the Ruge–Stüben coarsening [13]. More aggressive coarsening methods require special care in the interpolation construction, as we will see in the next section.

2.3. Interpolation. We recall that the prolongation operator P should satisfy

$$\mathcal{V} \subseteq \text{range}(P), \tag{14}$$

where $\mathcal{V}$ is the near-kernel of A, or, more precisely, for a coarse space of given size n_c the optimal two-level prolongation as stated in [9, 43] should be such that

$$\text{span}(\mathbf{v}_i) = \text{range}(P), \tag{15}$$

where $\mathbf{v}_i$ are the eigenvectors associated with the smallest n_c eigenvalues of the generalized eigenproblem (10). To this aim, depending on the problem, we use two different strategies.

If a test space is available or it is relatively cheap to obtain a reasonable approximation of the near-kernel, then the so-called BAMG interpolation is used. The name BAMG is used because this interpolation based on least squares was proposed for the first time in the context of bootstrap AMG [8]. In this approach, the weights of prolongation w_{ij}, i.e., the entries of the W block in (6), are found through least square minimizations:

$$w_{ij} = \underset{j \in C_i}{\text{argmin}} \, \|\mathbf{v}_i - \sum_{j \in C_i} w_{ij} \mathbf{v}_j\|^2, \qquad i = 1, \ldots, n, \tag{16}$$

with $\mathbf{v}_k$ the kth row of V and C_i the interpolatory set for i. In our experience it is imperative for an effective interpolation that the norm $\|\mathbf{v}_i - \sum_{j \in C_i} w_{ij} \mathbf{v}_j\|$ must be reduced to zero, and, in the general case, this can be accomplished only if the cardinality of C_i, $|C_i|$, is equal to or larger than n_t, the number of test vectors. To guarantee an exact interpolation, it is often necessary to use neighbors at a distance larger than one, especially when dealing with systems of PDEs, with a consequent increase of the overall operator complexity. Moreover, it may happen that, even if $|C_i| \geq n_t$, some of the vectors $\mathbf{v}_k$ are almost parallel, and high conditioning of Φ, the dense matrix collecting $\mathbf{v}_k$ components, may produce large jumps in the weights. In turn, large jumps in P introduce high frequencies in the next level operator that the smoother hardly handles. To overcome these difficulties, we compute our BAMG interpolation with an adaptive procedure similar to those described in [18, 33]. More specifically, let us define the matrix Φ whose entries φ_{ij} correspond to the jth component of the ith test vector $\mathbf{v}_i$ for any j in the interpolatory set. For each fine node $i \in \mathcal{F}$ to be interpolated, we start by including in the interpolatory set all its coarse neighbors at a distance no larger than $l_{\min}$, and we select a proper basis for Φ by using a maxvol algorithm [23, 28]. If either the relative residual,

$$r_i = \frac{\|\varphi_i - \Phi \mathbf{w}_i\|}{\|\varphi_i\|}, \tag{17}$$

or the norm of the weights, $\|\mathbf{w}_i\|$, is larger than the user-defined thresholds ϵ and μ, respectively, then we extend by one the interpolation distance. We keep on increasing

Algorithm 3 BAMG prolongation adaptive set-up

procedure BAMG_PROLONGATION(S, V, $l_{\min}$, $l_{\max}$, ϵ, μ)
  for all $i \in \mathcal{C}$ **do**;
    Set $l = l_{\min}$;
    Set $\mathbf{r}_i = \varphi_i$;
    while $l \leq l_{\max}$ **and** ($r_i > \epsilon$ **or** $\mathbf{w}_i > \mu$) **do**
      Include in C_i all the coarse nodes at a distance at most l;
      Collect all the φ_k such that $k \in C_i$;
      Select from φ_k a *maxvol* basis Φ_i;
      Find the vector of weights $\mathbf{w}_i$ by minimizing $\|\Phi_i \mathbf{w}_i - \varphi_i\|$;
      $l = l + 1$;
    end while
  end for
end procedure

the interpolation distance up to $l_{\max}$ to limit the computational cost. This procedure, briefly sketched in Algorithm 3, though slightly expensive, allows us to compute an accurate and smooth prolongation without impacting operator complexity too much. In fact, including in C_i all the coarse nodes within a priori selected interpolation distance usually leads to higher operator complexities because several fine nodes may be interpolated with excessively large support. Moreover, limiting the number of nonzeros in the rows of P has the additional advantage that it is possible to perform prolongation smoothing without an exponential growth of the operator complexity. Prolongation smoothing is a very common practice in solving elasticity problems with aggregation-based AMG, and numerical results will show that it can be beneficial also in the context of classical AMG.

On the other hand, when there is no explicit knowledge of the test space or when the matrix at hand arises from the discretization of a Poisson-like problem, Chronos can also rely on more classical interpolation schemes. Below we briefly recall the expressions of some well-known interpolation formulas. First, using the concept of SoC, we define the following sets:

- $N_i = \{j \mid a_{ij} \neq 0\}$, the set of direct neighbors of i;
- $S_i = \{j \in N_i \mid j \text{ strongly influences } i\}$, the set of strongly connected neighbors of i;
- $F_i^S = F \cap S_i$, the set of strongly connected fine neighbors of i;
- $C_i^S = C \cap S_i$, the set of strongly connected coarse neighbors of i;
- $N_i^W = N_i \setminus (F_i^S \cup C_i^S)$, the set of weakly connected neighbors of i.

A generally accurate distance-one interpolation formula, introduced in [36], is the so-called classical interpolation. Unlike other distance-one formulas, here, the interpolation takes care of the contribution from strongly influencing points F_i^S, and the expression for the interpolation weight is given by

$$w_{ij} = -\frac{1}{a_{ii} + \sum_{k \in N_i^w \cup F_i^{S*}} a_{ik}} \left(\sum_{k \in F_i^s \setminus F_i^{S*}} \frac{a_{ik} \bar{a}_{kj}}{\sum_{m \in C_i^s} \bar{a}_{km}} \right), \qquad j \in C_i^S, \tag{18}$$

where

$$\bar{a}_{ij} = \begin{cases} 0 & \text{if } sign(a_{ij}) = sign(a_{ii}), \\ a_{ij} & \text{otherwise.} \end{cases} \tag{19}$$

It is worth noting that the original formula proposed in [36] is here corrected accordingly with the modification introduced in [24] where the set of strongly connected neighbors F_i^{S*}, that are F-points but do not have a common C-point, are subtracted to the fine strong neighbors F_i^S. This modification of the interpolation formula is needed to prevent the term $\sum_{m \in C_i^s} \bar{a}_{km}$ from vanishing. Indeed, using the PMIS-coarsening method no longer guarantees that two strongly connected F-points are interpolated by a common C-point. However, even if for a large class of problems the classical interpolation is very effective, it can lose efficiency for challenging problems such as rotated anisotropies or problems with large discontinuities. Hence, some more advanced interpolation formulas are required to overcome these difficulties. A widely used interpolation strategy, mainly for very challenging Poisson-like problems, is the Extended+i interpolation. This interpolation formula extends the interpolatory set including C-points that are at distance two away from the F-point considered. Furthermore, also coarse nodes connected to the fine neighbors of the fine point interpolation are considered. Hence, denoting with $\hat{C}_i = C_i \cup \bigcup_{j \in F_i^S} C_j$ the set of distance-two coarse nodes, the interpolation Extended+i formula takes the following form:

$$w_{ij} = -\frac{1}{\tilde{a}_{ii}} \left(a_{ij} + \sum_{k \in F_i^s} \frac{a_{ik} \bar{a}_{kj}}{\sum_{l \in \hat{C}_{i \cup \{i\}}} \bar{a}_{kl}} \right), \qquad j \in \hat{C}_i, \tag{20}$$

with

$$\tilde{a}_{ii} = a_{ii} + \sum_{n \in N_i^w \setminus \hat{C}_i} a_{in} + \sum_{k \in F_i^s} a_{ik} \frac{\bar{a}_{ki}}{\sum_{l \in \hat{C}_i \cup i} \bar{a}_{kl}}. \tag{21}$$

This Extended+i interpolation remedies many problems occurring with distance-one interpolation and provides better weights than other distance-two interpolation formulas at the cost of larger operator complexities. A possible way to reduce the complexities either without affecting or only mildly affecting the convergence rate is to consider a different interpolatory set, i.e., an interpolatory set larger than the distance-one set C_i^S but smaller than the distance-two $\hat{C}_i$. The idea is to consider an interpolatory set that only extends C_i^S for strong F-F connections without a common C-point, since in the other cases the point i is already surrounded by interpolatory points belonging to C_i^S. Here, we propose enriching the set C_i^S including the minimum number of distance-two coarse nodes guaranteeing that each F-F strong connection is covered by at least a C-point. Let us consider the example in Figure 1.

Notice that using the classical interpolation, the interpolatory set would be $C_i^S = \{o\}$ and there would be two fine neighbors of i, j, and k that do not share a C-point with i. On the other hand, using the Extended+i interpolation, we would have that $\hat{C}_i = \{m, n, l, o\}$ and each F-node, strongly connected with i, would share at least one C-node of the interpolatory set $\hat{C}_i$. However, to guarantee this last condition, it would be sufficient to only include the node n to the set C_i^S, so that the extended interpolatory set would become $\hat{C}_i^h = \{o, n\}$. In other words, we extend the set C_i^S by including the MIS of distance-two C-nodes such that we accomplish the above

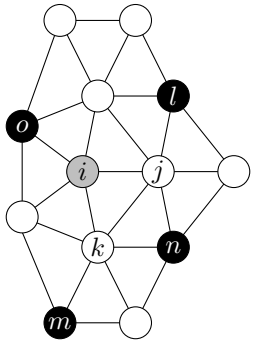

FIG. 1. *Example of the interpolatory points. The gray point is the point to be interpolated, black points are C-points, and white points are F-points.*

Algorithm 4 Computation of extended interpolation set $\hat{C}_i^h$

1: Set the initial interpolatory set $\hat{C}_i^h = C_i^S$
2: Compute the initial set $F^{'}$, such that:
3: $\quad F^{'} = \{j \in F_i^S \mid j$ is not strongly connected with at least one node in $C_i^S\}$
4: Compute the initial set of $C^{''}$, i.e., the set of the distance-two coarse nodes strongly connected with a $F^{'}$-point
5: Compute the vertex degree of each element in $C^{''}$ by taking into account only the connections with $F^{'}$
6: **while** $F^{'} \neq \emptyset$ **do**
7: $\quad$ Choose the node with maximum degree in $C^{''}$ and add it to $\hat{C}_i^h$
8: $\quad$ Update the set $F^{'}$
9: $\quad$ Update the set $C^{''}$
10: **end while**

condition. We call this interpolation Hybrid and we report the procedure to construct the interpolatory set $\hat{C}_i^h$ in Algorithm 4.

2.4. Filtering. One problem that may affect AMG methods, especially in parallel implementation, is the excessive stencil growth occurring in lower levels. This drawback is more pronounced if long-range interpolation or prolongation smoothing is used. Some authors have explored interesting solutions to reduce AMG complexity without detrimental effects on convergence [7, 15]. Simply eliminating small entries from the operators, as is done, for instance, with incomplete LU (ILU) factorizations or some approximate inverse preconditioners, may completely harm the effectiveness of CGC. This happens because removal of small entries from P_k or $A_{k+1} = P_k^T A_k P_k$ may induce an inaccurate representation of the near-kernel of A.

As a remedy, the authors in [15] propose compensating for the action of eliminated entries through a sort of stencil collapsing to guarantee that the filtered operator, say $\tilde{A}_{k+1}$, exerts the same action of A on the near-kernel

$$\tilde{A}_{k+1} W = A_{k+1} W \tag{22}$$

with W a matrix representation of the near-kernel. While it is relatively simple to enforce condition (22) for one-dimensional near-kernels, it is not immediate to accommodate the action on several vectors at the same time. With multiple vectors,

first the smallest entries of A_{k+1} are dropped to determine the pattern of $\tilde{A}_{k+1}$; then a correction to $\tilde{A}_{k+1}$, Δ_{k+1}, is computed by using least squares on

$$\|(A_{k+1} - \tilde{A}_{k+1})W - \Delta_{k+1}W\|_2. \tag{23}$$

In more detail, $\tilde{A}_{k+1}$ is computed row-wisely such that the absolute norm of each row is a given percentage ρ of the norm of the original one, and then the compensation is computed for the same row. We use the same procedure on the prolongation operator P_k with the only exception being that, instead of W, we use its injection in the coarse space. Operatively, the test space V is used when available, while in cases where V is not computed, such as in Poisson problems, we simply replace V with a constant vector.

Finally, we observe that $\tilde{A}_{k+1}$ is no more guaranteed to be SPD, and, especially when an aggressive dropping is enforced, the use of a nonsymmetric Krylov solver, such as GMRES or BiCGstab, is often needed instead of CG. Obviously, such care is not needed when only the prolongation is filtered, as $\tilde{P}_k^T A \tilde{P}_k$ is always SPD for any choice of ρ.

3. Library description. The Chronos package is a collection of classes and functions that implement linear algebra algorithms for distributed memory parallel computers. The library is written in C++, with message passing interface (MPI) and OpenMP directives used for communication among processes and multithread execution, respectively. The hybrid MPI-OpenMP implementation is more flexible in the use of modern computing resources, and it is generally more efficient than pure MPI due to its better exploitation of fine-grained parallelism.

Chronos has been developed using the potential of object-oriented programming (OOP). The abstraction introduced through the OOP allows the use of the same distributed matrix object to represent a linear system, a smoother, an AMG hierarchy, or a preconditioner itself. Another advantage of this approach is the possibility to use simpler classes to derive more advanced elements, such as block preconditioners. Moreover, whatever the preconditioner, the same iterative methods can be used for the linear system or eigenproblem solution. In addition, the modular structure allows us to easily integrate the CPU kernels with graphics processing units (GPUs) and field programmable gate array (FPGA), kernels leaving the overall structure of the library unchanged. A hybrid CPU-GPU version is already under development, and preliminary tests are encouraging [25].

A brief description of the main classes is reported in the next subsections.

3.1. Main classes. The level of abstraction and the hierarchy of the main classes are sketched in Figure 2. All these classes are shown so that the user may access the full range of Chronos functionalities.

The Distributed Dense Matrices (DDMats) and Distributed Sparse Matrices (DSMats) are managed by the $DDMat$ and the $DSMat$ classes, respectively. Both DDMat and DSMat storage schemes require the matrix to be subdivided into n_p horizontal stripes of consecutive rows, where n_p is the number of active MPI processes. In the DDMat, each stripe is stored row-wisely among the process to guarantee better access in memory during multiplication operations. This makes the DDMat very efficient for linear systems with multiple right-hand sides and eigenproblems, and distributed vectors are stored as a one-column DDMat. In the DSMat each stripe is subdivided into an array of CSR matrices. The CSR format is the Chronos standard format for shared sparse matrices, and their management is demanded to $CSRMAT$

class. The DSMat storage scheme adopted in Chronos is very effective in both the preconditioner set-up and the Sparse Matrix-Vector product (SpMV) because it allows a large superposition between communication and computation, as described in the next subsection.

The *Preconditioner* class manages the approximation of the inverse of a DSMat. It requires in input a DSMat as $DSMat$-type object and an optional test space as a $DDMat$-type object. The classes derived from *Preconditioner* are Jac, $aFSAI$, and $aAMG$ for Jacobi, aFSAI, and AMG, respectively. A useful feature is that each of these classes can be used as a smoother in the AMG.

Both the $DSMat$ and *Preconditioner* classes are derived from the $MatrixProd$ class, which manages the SpMV at the highest level of abstraction. The SpMV is the most expensive operation in any preconditioned iterative solver, and its management has defined the design of the whole library. With reference to Figure 2, the $MatrixProd$ class leads the Chronos structure together with the iterative solvers. Furthermore, more general $MatrixProd$ elements can be readily built using the $MatrixProdList$ class, which manages an implicit $MatrixProd$ object defined as a product of a sequence of $MatrixProd$ objects ordered into a list.

At the top of the hierarchy, there are also the iterative solvers for linear systems and eigenproblems, *LinSolver* and *EigSolver*, respectively. Currently, the classes derived from *LinSolver* are *PCG* and *BiCGstab*, for the preconditioned conjugate gradient (PCG) and the preconditioned biconjugate gradient stabilized (BiCGstab), respectively.

Finally, the Power Method and the Simultaneous Rayleigh Quotient Minimization (SRQCG) are implemented in the two classes $PowMeth$ and $SRQCG$ derived from $EigSolver$.

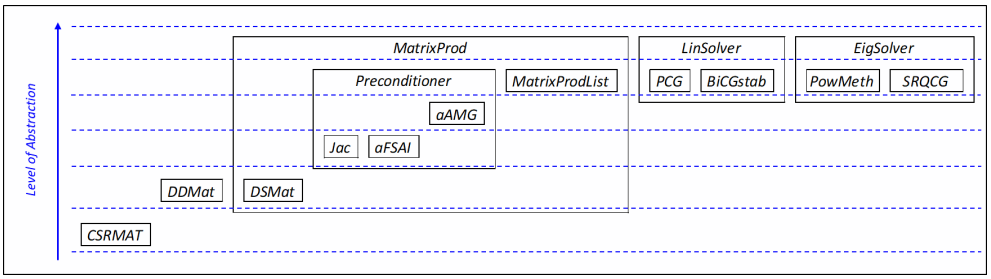

FIG. 2. *Chronos main classes and hierarchies.*

3.2. Distributed Sparse Matrix storage scheme. The DSMat storage scheme implemented in Chronos consists of a partitioning of the matrix into n_p horizontal stripes of consecutive rows. Each stripe is then divided into blocks by applying the same subdivision to the columns, as schematically shown in Figure 3, and each block is stored as a CSR matrix.

The CSR matrices have a local numbering, i.e., rows and columns of block IJ are numbered from 0 to n_{I-1} and from 0 to n_{J-1}, where n_I and n_J are the number of rows assigned to processes I and J, respectively. This expedient allows the use of 4-byte integers, saving memory and increasing efficiency.

Each process stores the diagonal block and the list of "Left" (with a lower index) and "Right" (with a higher index) blocks corresponding to the connections with

neighboring processes. With reference to Figure 3, process 3 stores the 5 blocks highlighted in red: 0, 1, and 2 as left neighbors, and the diagonal block with only internal connections and 6 as right neighbor.

This blocked scheme, although a bit cumbersome to implement, allows us to stress nonblocking send/receive communications with a large superposition between data-transfer and computation. It has proven to be very effective in all basic operations involving a DSMat, including SpMV product, matrix-by-matrix product, and matrix transposition.

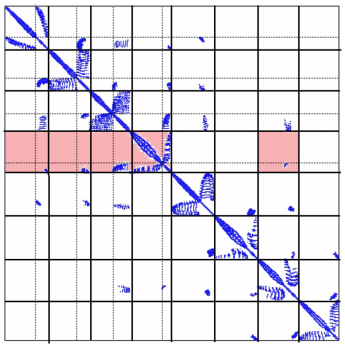

FIG. 3. *Schematic representation of the DSMat storage scheme implemented in Chronos using* 8 *MPI processes. The red blocks are assigned to process* 3. *(Color available online.)*

4. Numerical results. The numerical experiments have been performed using large sparse matrices arising from challenging real-world problems. As described in the previous sections, Chronos can exploit the effectiveness of an advanced and tunable smoother like aFSAI, vary the coarsening ratio, or switch between different interpolation methods, always finding an efficient set-up for any problem.

Chronos is benchmarked on a set of problems that can be grouped into two classes denoted as fluid dynamic (F) and mechanical (M). The first consists of a series of problems arising from the discretization of the Laplace operator and related to fluid dynamic problems, such as underground fluid flow (reservoir), compressible or incompressible airflow around complex geometries (CFD), or porous flow (porous flow). The second category includes problems related to mechanical applications such as subsidence analysis, hydrocarbon recovery, gas storage (geomechanics), mesoscale simulation of composite materials (mesoscale), mechanical deformation of human tissues or organs subjected to medical interventions (biomedicine), and design and analysis of mechanical elements, e.g., cutters, gears, air-coolers (mechanical).

In our experiments, we consider challenging test cases, not only for the high number of degrees of freedom (DOFs), but also because of their intrinsic ill-conditioning. Indeed, in real applications, we usually have to deal with large jumps of the physical proprieties, complicated geometries leading to highly distorted elements, heterogeneity, and anisotropy. The matrices considered in the experiments are listed in Table 1 with details about the size, the number of nonzeros, and the application field they arise from. The reader can refer to the supplementary material file M139858_Suppl_Mat.pdf [local/web 2.40MB] for a detailed description of each test case.

We subdivide the discussion of the results into two parts, the former collecting test cases from fluid dynamics and the latter from mechanics. We also provide strong and weak scalability analysis of the proposed implementation using large-scale computa-

TABLE 1

Benchmark matrices used in the numerical experiments. For each matrix, the class, the size, n, the number of nonzeros, nnz, the average number of nonzeros per row, and the application field are provided.

Matrix	Class	n	nnz	avg. nnz/row	Application field
`finger4m`	F	4,718,592	23,591,424	5.00	porous flow
`guenda11m`	M	11,452,398	512,484,300	44.75	geomechanics
`agg14m`	M	14,106,408	633,142,730	44.88	mesoscale
`M20`	M	20,056,050	1,634,926,088	81.52	mechanical
`tripod24m`	M	24,186,993	1,111,751,217	45.96	mechanical
`rtanis44m`	F	44,798,517	747,633,815	16.69	porous flow
`geo61m`	M	61,813,395	4,966,380,225	80.34	geomechanics
`poi65m`	F	65,939,264	460,595,552	6.99	CFD
`Pflow73m`	F	73,623,733	2,201,828,891	29.91	reservoir
`c4zz134m`	M	134,395,551	10,806,265,323	80.41	biomedicine

tional resources. The results are presented in terms of total number of computational cores used, n_{cr}, the grid and operator complexities, C_{gd} and C_{op}, respectively, the number of iterations, n_{it}, and the set-up, iteration, and total times, T_p, T_s, and $T_t = T_p + T_s$, respectively.

The right-hand side vector used for all test cases is a random vector. The linear systems are solved by PCG with a zero initial solution, and convergence is considered achieved when the $l2$-norm of the iterative residual becomes smaller than $10^{-8} \cdot \|b\|$. Chronos performance has been evaluated on Marconi100, a supercomputer hosted in the Italian consortium for supercomputing (CINECA). Marconi100, classified within the first ten positions of the TOP500 ranking [39] at the time of writing, is composed of 980 nodes based on the IBM Power9 architecture, each equipped with two 16-core IBM POWER9 AC922 at 3.1 GHz processors. For each test, the number of cores, n_{cr}, is selected to have a per core load of about 100–150,000 unknowns, and, consequently, different numbers of nodes are allocated for different problem dimensions. For all the tests, each node reserved for the run is always fully exploited by using 8 MPI tasks and 4 OpenMP threads for each task.

As a reference point to evaluate the performance of Chronos, we compare it with the state-of-the-art solvers BoomerAMG, a classical AMG, and GAMG, a smoothed aggregation-based AMG, as preconditioners in fluid dynamics and mechanical problems, respectively. BoomerAMG and GAMG have been chosen as baseline solvers because they are very well-known open-source packages whose performance has been demonstrated in several papers [6, 12, 16, 24].

4.1. Fluid dynamics test cases. The general-purpose AMG implemented in Chronos is highly tunable, offering several set-up options to effectively solve this set of problems, as will be shown below.

First, we start by comparing Chronos and BoomerAMG performance using a set-up that is as similar as possible. This comparison is intended to verify our HPC implementation and to demonstrate the efficiency of the DSMat storage scheme for SpMV product, considering the three test cases `finger4m`, `poi65m`, and `Pflow73m`. The comparison takes place with the same preconditioner configuration, i.e., Jacobi smoothing, classical SoC with $\theta = 0.25$, PMIS coarsening, and Extended+i prolongation. The only exception takes place for `Pflow73m`, where we used $\theta = 0.0$, which significantly increases performance. Table 2 provides for each test case the results obtained with these *standard set-ups* denoted as Chr-jac and Boomer-jac. The grid

TABLE 2

Solution of three fluid dynamic test cases among those reported in Table 1 *using Jacobi smoothing and Extended+i prolongation. For each run, the following information is provided: The number of cores n_{cr}, the grid C_{gd} and operator C_{op} complexities, the number of PCG iterations n_{it}, the set-up time T_p, the iteration time T_s, and the total time T_t.*

Matrix	n_{cr}	Solv. type	C_{gd}	C_{op}	n_{it}	T_p [s]	T_s [s]	T_t [s]
`finger4m`	32	Chr-jac	1.453	2.558	16	1.13	0.55	1.68
	32	Boomer-jac	1.454	2.574	16	0.81	0.70	1.51
`poi65m`	384	Chr-jac	1.327	4.036	16	3.81	1.65	4.46
	384	Boomer-jac	1.361	4.450	13	5.70	2.03	7.73
`Pflow73m`	480	Chr-jac	1.125	1.614	3308	14.1	611.9	626.0
	480	Boomer-jac	1.123	1.593	3576	26.1	771.7	797.8

TABLE 3

Solution of three fluid dynamic test cases among those reported in Table 1 *using default smoothers and Extended+i prolongation. For each run, the following information is provided: The number of cores n_{cr}, the grid C_{gd} and operator C_{op} complexities, the number of PCG iterations n_{it}, the set-up time T_p, the iteration time T_s, and the total time T_t.*

Matrix	n_{cr}	Solv. type	C_{gd}	C_{op}	n_{it}	T_p [s]	T_s [s]	T_t [s]
`finger4m`	32	Chr	1.453	2.558	7	3.71	0.33	4.04
	32	Boomer	1.454	2.574	12	0.79	0.94	1.73
`poi65m`	384	Chr	1.346	4.496	6	27.5	1.84	29.34
	384	Boomer	1.361	4.450	14	5.88	3.18	9.06
`Pflow73m`	480	Chr	1.125	1.614	240	46.5	64.2	110.7
	480	Boomer	1.123	1.593	2777	26.5	1042.3	1068.8

and operator complexities with the two softwares are basically the same, and also the iteration count turns out to be quite similar, showing that the two implementations are consistent. Only a slight difference occurs for `Pflow73m`, but it is compatible with very small differences in the code implementation.

Figure 4 provides the time for the preconditioner set-up (left) and for PCG (right) for each solving strategy. Each time reported in the figure is normalized with respect to the corresponding Boomer-jac time, which is our baseline. We observe that, while using Jacobi smoothing, Chronos is faster than BoomerAMG in the set-up for `poi65m` and `Pflow73m`, while BoomerAMG is better in `finger4m`. Differently, as to the iteration time, Chronos slightly outperforms BoomerAMG in all the tests. In all cases, the Chronos implementation turns out to be very efficient, and the total solution time is comparable to and sometimes even better than that of the BoomerAMG.

In Table 3, the labels Chr and Boomer identify the results obtained with Chronos and BoomerAMG when the default smoothers are selected, i.e., aFSAI and hybrid Gauss–Seidel, respectively. The use of a more elaborate/sophisticated smoother with respect to either Jacobi or hybrid Gauss–Seidel gives a significant advantage in terms of iteration count and solving time at the price of a more expensive set-up, as shown also in Figure 4. The use of aFSAI always allows for achieving a faster convergence. Furthermore, the more ill-conditioned the problem is, the better aFSAI compares with other smoothers. In `Pflow73m`, which is the hardest problem in fluid dynamics, Chronos with aFSAI smoothing is 6 times faster than BoomerAMG. The set-up time is larger, but the speed-up obtained in the iteration stage may justify this effort, especially in transient simulations where the user may have to repeatedly solve the

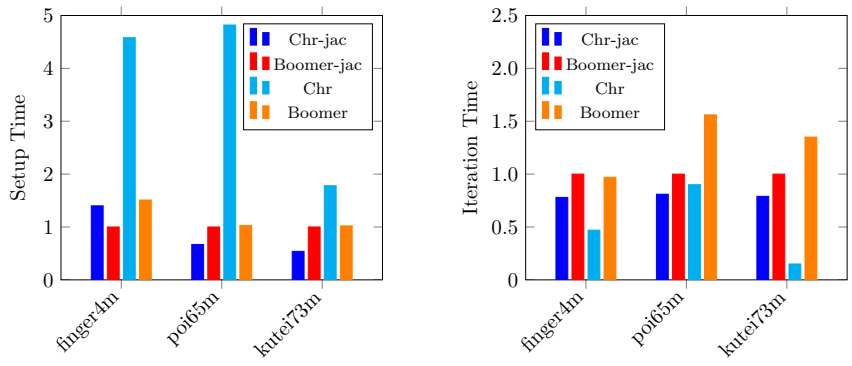

FIG. 4. *Comparison between Chronos and BoomerAMG by using the Extended+i prolongation and Jacobi or default smoothing. Left: Set-up time. Right: Solution time.*

TABLE 4

Comparison between different interpolation formulas in the solution of the fluid dynamic test problems from Table 1. *For each run, the following information is provided: Number of cores n_{cr}, prolongation type, grid C_{gd} and operator C_{op} complexities, number of iterations n_{it}, set-up time T_p, iteration time T_s, and total time T_t.*

Matrix	n_{cr}	Prol. type	C_{gd}	C_{op}	n_{it}	T_p [s]	T_s [s]	T_t [s]
`finger4m`	32	Chr-clas	1.467	1.871	31	3.57	1.32	4.89
	32	Chr-hybc	1.465	2.051	14	3.62	0.66	4.28
	32	Chr-exti	1.453	2.558	7	3.71	0.33	4.04
`rtanis44m`	384	Chr-clas	1.612	1.943	46	23.3	6.47	29.8
	384	Chr-hybc	1.585	2.030	36	26.6	6.09	32.8
	384	Chr-exti	1.572	2.580	16	34.0	2.90	36.9
`poi65m`	384	Chr-clas	1.381	2.339	21	17.9	4.69	22.59
	384	Chr-hybc	1.361	2.888	13	19.0	2.57	21.57
	384	Chr-exti	1.346	4.496	6	27.5	1.84	29.34
`Pflow73m`	480	Chr-clas	1.236	1.391	414	39.4	60.2	99.6
	480	Chr-hybc	1.234	1.448	416	40.4	67.1	107.5
	480	Chr-exti	1.234	2.346	410	57.7	120.9	178.6

same linear system and can take advantage of preconditioner recycling.

In fluid dynamics, the prolongations of choice for classical AMG are typically the classical or Extended+i interpolations. The latter is usually more effective, although more expensive, for challenging problems due to its ability to accurately interpolate to fine nodes having strong fine neighbors that do not share the same strong coarse node, possibly produced by high coarsening ratios. In Table 4, we compare these two well-known prolongations to the hybrid one that has been discussed in section 2.3. For `finger4m` and `poi65m`, we can observe that the Extended+i interpolation is the more accurate one, with higher operator complexity. As expected, this leads to a lower number of iterations, but a higher computational cost per iteration. On the contrary, the classical interpolation formula is the cheapest to compute, with very low operator complexity. However, taking into account only distance-one coarse nodes, the prolongation operator is not able to accurately reproduce the smooth error, causing an increase of the iteration count, up to twice the iteration count obtained with

Extended+i. For these two tests, the best configuration lies in the middle, i.e., the hybrid interpolation formula, which keeps the operator complexity small by only taking distance-two coarse nodes that are actually useful in the interpolation process into account. In this way, we are able to obtain a more accurate interpolation formula with a computational cost comparable to the classical one.

The behavior is quite different for the other two test cases.

In `rtanis44m`, a strong heterogeneity and anisotropy of permeability tensor dramatically increase the conditioning of the problem. Hence, the most accurate interpolation method, i.e., Extended+i, is needed to efficiently solve this problem. The iteration count is one third with respect to classical interpolation, and the solution time is approximately one half. Unlike before, the increased accuracy of the hybrid interpolation over classical is not enough to give a sufficient benefit in terms of solving time. It is worth noting that the increased set-up cost for Extended+i is in this case largely compensated for in the iteration stage. This gain is even more pronounced in cases where preconditioner recycling is possible such as in some transient or nonlinear simulations.

The last test case considered in this section is `Pflow73m`, a very challenging and severely ill-conditioned problem from underground flow. Even if this is a diffusion problem, the great jumps in permeability and the distorted mesh lead to a matrix whose near-kernel is not well represented by the constant vector. For this reason, the iterations to converge are much larger than in the other tests, and not even the most accurate interpolations such as Extended+i or hybrid give any benefit over classical interpolation, which, being the cheapest one, proves the most effective strategy for this test case.

4.2. Mechanical test cases. As seen above, Chronos allows for setting up a very flexible AMG preconditioner, adaptable to problem types the user has to solve, with different choices available for interpolation operators and smoothing methods. In addition to different available choices for interpolation and smoothing, Chronos allows the possibility to directly smooth the prolongation and/or filter it. Moreover, for mechanical test problems, we have seen that it is very helpful in keeping the grid complexity low, especially in cases of prolongation smoothing. This is easily achieved with default settings by dropping only a very small number of connections in the SoC graph. As in the previous section, we first define a baseline with state-of-the-art methods such as BoomerAMG (Boomer), with hybrid Gauss–Seidel smoothing, the unknown-based Boomer with separate treatment of unknowns relative to different directions (unk-based-Boomer) and the GAMG, a smoothed aggregation-based method.

We first refer to the test case `tripod24m`, whose results are provided in Table 5. With the standard Boomer, the solution is reached with a high number of iterations (more than 900), and the iteration time is responsible for most in terms of the time to solution. A significant improvement is obtained using the unknown-based version [5], where iterations are reduced by one third, and set-up and iteration times drop by 50%. The aggregation based AMG seems to be the most effective one for mechanical problems as, with GAMG, iterations are further reduced, and both T_p and T_s times decrease significantly. In this problem, Chronos with BAMG prolongation and aFSAI smoother (BAMG-aFSAI) is more effective than GAMG with a speed-up of two on the total time. The set-up time is longer, but the number of PCG iterations is lower, and the cost per iteration is one third that of GAMG. It is also possible to smooth the prolongation operator with a Jacobi step. We denote this method as SBAMG-aFSAI.

TABLE 5

Solution of the `tripod24m` *test case from Table* 1 *with different approaches. For each run, the following information is provided: The number of cores n_{cr}, the preconditioner type, the grid complexity C_{gd}, the operator complexity C_{op}, the number of iterations n_{it}, the set-up time T_p, the iteration time T_s, and the total time T_t.*

Matrix	n_{cr}	Prec. type	C_{gd}	C_{op}	n_{it}	T_p [s]	T_s [s]	T_t [s]
		Boomer	1.244	3.207	931	64.1	611.9	676.1
		unk-based-Boomer	1.328	3.669	335	43.8	262.4	203.5
`tripod24m`	160	GAMG	1.543	-	294	12.1	80.5	92.6
		BAMG-aFSAI	1.041	1.116	222	21.8	23.0	44.8
		SBAMG-aFSAI	1.041	1.322	118	36.7	16.1	52.9
		FBAMG-aFSAI	1.041	1.212	120	33.5	13.5	47.0

As could be expected, the operator complexity and the set-up time both increase, but, on the other hand, the number of iterations and solution time are smaller. The increase of operator complexity and set-up time can be limited by means of filtering (FBAMG-aFSAI) without compromising effectiveness. FBAMG-aFSAI requires the same number of iterations to converge but at a lower cost per iteration. These two last strategies are particularly effective in a FEM simulation where the preconditioner can be reused several times in different time-steps so that the set-up cost becomes negligible.

Chronos proved robust and efficient in addressing all the mechanical test cases. A comparison of the number of iterations and times obtained with GAMG and the three BAMG strategies outlined above is shown in Table 6. To highlight the speed-up, Figure 5 shows set-up and iteration time normalized to the GAMG times. Unfortunately, the comparison for the two largest cases, `geo61m` and `c4zz13m`, is not reported because these matrices have not been made available on file due to their large size, and the tests have been run by linking Chronos to the FEM program ATLAS [21]. For the three benchmarks, `guenda11m`, `tripod24m`, and `M20`, the number of PCG iterations required by GAMG and BAMG is comparable, but the overall solution time is significantly lower for BAMG with a speed-up of Chronos over GAMG up to 4 in these tests. The only exception is the matrix `agg14m`, where GAMG is able to produce a very effective preconditioner at the lowest set-up cost.

Finally, with the aid of Figure 6, we would like to point out how the total solution time depends only mildly on the problem nature but more significantly on its size. Figure 6 shows for each problem the total solution time divided by the number of nonzeros per allocated core, and this resulting time is further normalized with the average among all the experiments. In other words, the figure should show the solution time for each problem as if *exactly* the same resources were allocated for each nonzero. For a preconditioner that is totally independent of the problem nature, the same solution time for every problem is expected. It can be observed that, through a proper set-up, Chronos is able to produce total solution times very close to the average normalized solution time, thus showing only a mild dependence on the application at hand.

4.3. Strong and weak scalability. In this last subsection, we evaluate the strong and weak scalability of the AMG preconditioners implemented in Chronos. All three times, i.e., set-up T_p, iteration T_s, and total T_t times, are analyzed to assess scalability. The strong scalability test is shown in the top of Figure 7, on the left for `poi65m` with Extended+i prolongation and on the right for the `c4zz134m` test matrix

TABLE 6

Comparison between different interpolation formulas in the solution of the mechanical test problems from Table 1. For each run, the following information is provided: Number of cores n_{cr}, prolongation type, grid C_{gd} and operator C_{op} complexities, number of iterations n_{it}, set-up time T_p, iteration time T_s, and total time T_t.

Matrix	n_{cr}	Prol. type	C_{gd}	C_{op}	n_{it}	T_p [s]	T_s [s]	T_t [s]
guenda11m	64	GAMG	1.580	-	978	18.3	306.2	324.5
	64	BAMG	1.041	1.118	937	27.8	105.0	133.0
	64	SBAMG	1.041	1.354	638	50.3	96.3	147.0
	64	FBAMG	1.041	1.240	638	43.5	79.8	123.0
agg14m	128	GAMG	1.644	-	26	12.5	5.8	18.2
	128	BAMG	1.085	1.287	135	30.6	22.2	52.8
	128	SBAMG	1.085	2.264	31	114.4	8.1	122.6
	128	FBAMG	1.085	1.670	34	53.6	7.3	60.9
M20	128	GAMG	1.162	-	245	211.0	391.4	602.4
	128	BAMG	1.054	1.184	775	71.2	275.0	347.0
	128	SBAMG	1.054	1.677	151	158.0	71.2	229.2
	128	FBAMG	1.054	1.292	158	93.9	55.1	149.1
tripod24m	160	GAMG	1.543	-	294	12.1	80.5	92.6
	160	BAMG	1.041	1.116	222	21.8	23.0	44.8
	160	SBAMG	1.041	1.322	118	36.7	16.1	52.9
	160	FBAMG	1.041	1.212	120	33.5	13.5	47.0

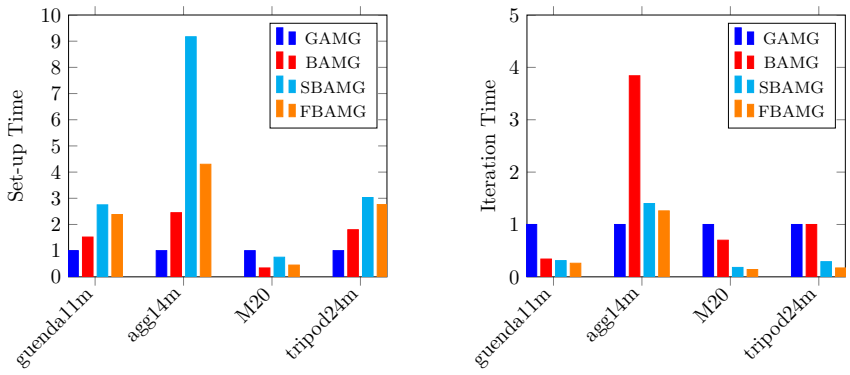

FIG. 5. *Comparison between GAMG and the BAMG strategies on the mechanical test cases. Left: Normalized T_p to the GAMG solution. Right: Normalized T_s to the GAMG solution.*

with BAMG prolongation. The number of cores varies from the minimum necessary to store matrix and preconditioner up to 8 times. In both tests, the times decrease as the computing resources increase, with a trend close to the ideal one.

Finally, the weak scaling is investigated with a standard 7-point finite difference discretization of the Poisson problem. Figure 7, bottom, shows both the total time spent in the set-up and solve phases, on the left, and the corresponding parallel efficiencies, on the right. Weak scaling efficiency up to N nodes is defined as $E = T_N/(NT_1)$, with T_1 the time required on a single node and T_N the time on N nodes. In this test, we always assign 218,750 unknowns per core.

The result shows that efficiency is very good and stays almost constant in the first two doubles of the cores, whereas a smaller efficiency occurs in the last one.

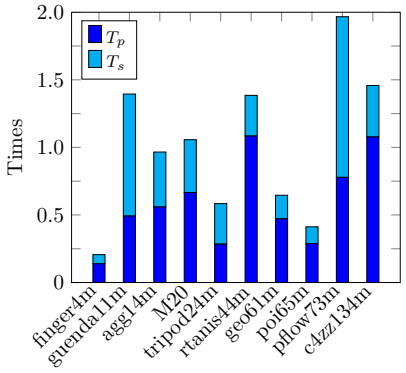

FIG. 6. *Set-up and iteration times, for all benchmark problems, normalized over the resources allocated per nonzero.*

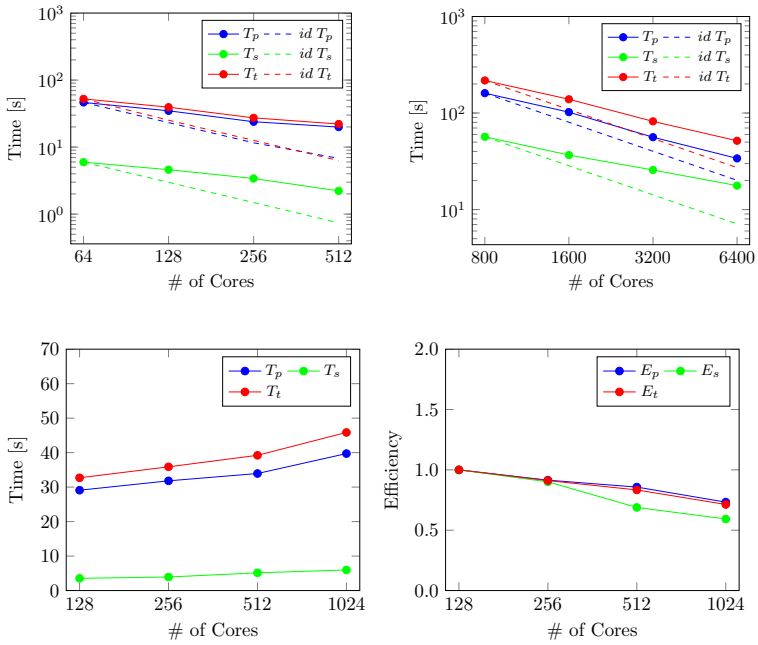

FIG. 7. *Strong scalability test for* `poi65m` *matrix and Extended+i prolongation (top-left) and* `c4zz134m` *matrix and BAMG prolongation (top-right). Weak scalability test on a standard* 7*-point finite difference discretization of the Poisson problem. Set-up, iteration, and total times versus* n_{cr} *on bottom left; corresponding efficiencies versus* n_{cr} *on bottom right.*

This performance dropdown can be ascribed to two distinct factors. First, while Marconi100 cores can be fully reserved for the test runs, the overall network is always shared with other users, and, consequently, the larger the resource allocation, the larger the disturbance from other running processes. Second, a performance dropdown is almost unavoidable in AMG methods, as the grid hierarchy always ends up with

small grids. The larger the number of resources allocated, the less efficient will be the software in dealing lower levels. Currently, to ease the implementation, Chronos uses all the allocated cores on each grid except the last one, where an `allgather` operation is called from a single core to solve the coarsest problem. In a future implementation, we plan to progressively reduce the number of resources with levels, thus reducing the network traffic and increasing efficiency.

5. Conclusions. In this work, the Chronos library for the solution of large and sparse linear algebra problems on high performance platforms has been presented with a deep analysis of its numerical and computational performance. Chronos, which will be freely accessible to research institutions [20], provides iterative solution methods for linear systems and eigenproblems along with advanced parallel preconditioners.

Although the library comprises classical and novel methods already known in the literature, all of its algorithms have been attentively revisited, tuned, and optimized on the basis of a large experimentation on real-world and industrial benchmarks arising from a wide variety of application fields. Moreover, every numerical kernel has been designed with special attention to its parallel performance and future extensibility to new numerical approaches and hardware.

The wide set of numerical experiments provided in the work clearly shows the ability of Chronos to give excellent performance for diverse applications with solution times no worse or even better than those offered by other widely used HPC linear solvers such as BoomerAMG and GAMG. Furthermore, this library offers great flexibility in the choice of the preconditioning strategy with the result that, once a proper set-up is found, total solution time depends solely or almost solely on the problem size and the number of computational resources allocated.

Our future work will be focused on porting Chronos on more energy-efficient and promising hardware such as GPU accelerators or FPGA, as well as using the innermost kernels of the library to develop advanced block preconditioners for multiphysics applications.

We also plan to build a stronger theoretical basis for the adaptive construction of the test space, unavoidable in problems lacking an initial guess for the operator near-kernel, and for the operator and prolongation filtering, which can greatly improve performance in tough problems where denser operators may be needed.

Acknowledgments. The authors gratefully thank Prof. S. Koric, G. Mazzucco, and E. L. Carniel, who provided the matrices M20, agg14m, and c4zz134m used in the experiments, and Dr. V. A. Paludetto Magri for his suggestions and support in using BoomerAMG.

REFERENCES

[1] F. P. Ali Beik and M. Benzi, *Iterative methods for double saddle point systems*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 902–921, https://doi.org/10.1137/17M1121226.

[2] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary, *Performance and scalability of the block low-rank multifrontal factorization on multicore architectures*, ACM Trans. Math. Software, 45 (2019), 2.

[3] S. Badia, A. F. Martín, and J. Principe, *Multilevel balancing domain decomposition at extreme scales*, SIAM J. Sci. Comput., 38 (2016), pp. C22–C52, https://doi.org/10.1137/15M1013511.

[4] R. Baggio, A. Franceschini, N. Spiezia, and C. Janna, *Rigid body modes deflation of the preconditioned conjugate gradient in the solution of discretized structural problems*, Computers & Structures, 185 (2017), pp. 15–26, https://doi.org/10.1016/j.compstruc.2017.03.003.

[5] A. H. Baker, T. V. Kolev, and U. M. Yang, *Improving algebraic multigrid interpolation operators for linear elasticity problems*, Numer. Linear Algebra Appl., 17 (2009), pp. 495–517.

[6] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc web page*, https://www.mcs.anl.gov/petsc, 2019.

[7] A. Bienz, R. D. Falgout, W. Gropp, L. N. Olson, and J. B. Schroder, *Reducing parallel communication in algebraic multigrid through sparsification*, SIAM J. Sci. Comput., 38 (2016), pp. S332–S357, https://doi.org/10.1137/15M1026341.

[8] A. Brandt, J. Brannick, K. Kahl, and I. Livshits, *Bootstrap AMG*, SIAM J. Sci. Comput., 33 (2011), pp. 612–632, https://doi.org/10.1137/090752973.

[9] J. Brannick, F. Cao, K. Kahl, R. Falgout, and X. Hu, *Optimal interpolation and compatible relaxation in classical algebraic multigrid*, SIAM J. Sci. Comput., 40 (2018), pp. A1473–A1493, https://doi.org/10.1137/17M1123456.

[10] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, *Adaptive smoothed aggregation (αSA) multigrid*, SIAM Rev., 47 (2005), pp. 317–346, https://doi.org/10.1137/050626272.

[11] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, *Adaptive algebraic multigrid*, SIAM J. Sci. Comput., 27 (2006), pp. 1261–1286, https://doi.org/10.1137/040614402.

[12] M. Brezina, C. Tong, and R. Becker, *Parallel algebraic multigrids for structural mechanics*, SIAM J. Sci. Comput., 27 (2006), pp. 1534–1554, https://doi.org/10.1137/040608271.

[13] H. De Sterck, R. D. Falgout, J. W. Nolting, and U. M. Yang, *Distance-two interpolation for parallel algebraic multigrid*, Numer. Linear Algebra Appl., 15 (2008), pp. 115–139.

[14] H. De Sterck, U. M. Yang, and J. J. Heys, *Reducing complexity in parallel algebraic multigrid preconditioners*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 1019–1039, https://doi.org/10.1137/040615729.

[15] R. D. Falgout and J. B. Schroder, *Non-Galerkin coarse grids for algebraic multigrid*, SIAM J. Sci. Comput., 36 (2014), pp. C309–C334, https://doi.org/10.1137/130931539.

[16] R. D. Falgout and U. M. Yang, *hypre: A library of high performance preconditioners*, in Proceedings of the International Conference on Computational Science-Part III, ICCS '02, Springer-Verlag, Berlin, Heidelberg, 2002, pp. 632–641, https://doi.org/10.1007/3-540-47789-6_66.

[17] M. Ferronato, A. Franceschini, C. Janna, N. Castelletto, and H. A. Tchelepi, *A general preconditioning framework for coupled multiphysics problems with application to contact- and poro-mechanics*, J. Comput. Phys., 398 (2019), 108887.

[18] A. Franceschini, V. A. Paludetto Magri, G. Mazzucco, N. Spiezia, and C. Janna, *A robust adaptive algebraic multigrid linear solver for structural mechanics*, Comput. Methods Appl. Mech. Engrg., 352 (2019), pp. 389–416.

[19] M. Frigo, N. Castelletto, and M. Ferronato, *A relaxed physical factorization preconditioner for mixed finite element coupled poromechanics* , SIAM J. Sci. Comput., 41 (2019), pp. B694–B720, https://doi.org/10.1137/18M120645X.

[20] M. Frigo, G. Isotton, and C. Janna, *Chronos web page*, https://www.m3eweb.it/chronos, 2021.

[21] M. Frigo, G. Isotton, C. Janna, N. Spiezia, and O. Tosatto, *ATLAS web page*, https://www.m3eweb.it/atlas, 2021.

[22] A. Frommer, K. Kahl, F. Knechtli, M. Rottmann, A. Strebel, and I. Zwaan, *A multigrid accelerated eigensolver for the Hermitian Wilson–Dirac operator in lattice QCD*, Comput. Phys. Commun., 258 (2021), 107615.

[23] S. A. Goreinov, I. V. Oseledets, D. V. Savostyanov, E. E. Tyrtyshnikov, and N. L. Zamarashkin, *How to find a good submatrix*, in Matrix Methods: Theory, Algorithms and Applications, World Scientific, Hackensack, NJ, 2010, pp. 247–256.

[24] V. E. Henson and U. M. Yang, *BoomerAMG: A parallel algebraic multigrid solver and preconditioner*, Appl. Numer. Math., 41 (2002), pp. 155–177, https://doi.org/10.1016/S0168-9274(01)00115-5.

[25] G. Isotton, C. Janna, and M. Bernaschi, *A GPU-accelerated adaptive FSAI preconditioner for massively parallel simulations*, Internat. J. High Performance Comput. Appl., (2021), https://doi.org/10.1177/10943420211017188.

[26] C. Janna, M. Ferronato, and G. Gambolati, *The use of supernodes in factored sparse approximate inverse preconditioning*, SIAM J. Sci. Comput., 37 (2015), pp. C72–C94,

https://doi.org/10.1137/140956026.

[27] C. Janna, M. Ferronato, F. Sartoretto, and G. Gambolati, *FSAIPACK: A software package for high-performance factored sparse approximate inverse preconditioning*, ACM Trans. Math. Software, 41 (2015), 10, https://doi.org/10.1145/2629475.

[28] D. E. Knuth, *Semioptimal bases for linear dependencies*, Linear Multilinear Algebra, 17 (1985), pp. 1–4.

[29] S. Koric and A. Gupta, *Sparse matrix factorization in the implicit finite element method on petascale architecture*, Comput. Methods Appl. Mech. Engrg., 302 (2016), pp. 281–292.

[30] S. Koric, Q. Lu, and E. Guleryuz, *Evaluation of massively parallel linear sparse solvers on unstructured finite element meshes*, Computers & Structures, 141 (2014), pp. 19–25, https://doi.org/10.1016/j.compstruc.2014.05.009.

[31] B. Lee, *Algebraic multigrid for systems of elliptic boundary-value problems*, Numer. Linear Algebra Appl., 17 (2020), pp. 495–21.

[32] O. E. Livne and A. Brandt, *Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver*, SIAM J. Sci. Comput., 34 (2012), pp. B499–B522, https://doi.org/10.1137/110843563.

[33] V. A. Paludetto Magri, A. Franceschini, and C. Janna, *A novel algebraic multigrid approach based on adaptive smoothing and prolongation for ill-conditioned systems*, SIAM J. Sci. Comput., 41 (2019), pp. A190–A219, https://doi.org/10.1137/17M1161178.

[34] F.-H. Rouet, C. Ashcraft, J. Dawson, R. Grimes, E. Guleryuz, S. Koric, R. F. Lucas, J. S. Ong, T. A. Simons, and T.-T. Zhu, *Scalability challenges of an industrial implicit Finite Element code*, in Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, Washington, DC, 2020, pp. 505–514.

[35] T. Roy, T. Jönsthövel, C. Lemon, and A. J. Wathen, *A constrained pressure-temperature residual (CPTR) method for non-isothermal multiphase flow in porous media*, SIAM J. Sci. Comput., 42 (2020), pp. B1014–B1040, https://doi.org/10.1137/19M1292023.

[36] J. W. Ruge and K. Stüben, *Algebraic multigrid*, in Multigrid Methods, Frontiers Appl. Math. 3, SIAM, Philadelphia, 1987, pp. 73–130, https://doi.org/10.1137/1.9781611971057.ch4.

[37] Y. Saad and H. A. Van der Vorst, *Iterative solution of linear systems in the* 20*th century*, J. Comput. Appl. Math., 123 (2000), pp. 1–33.

[38] K. Stüben, *A review of algebraic multigrid*, J. Comput. Appl. Math., 128 (2001), pp. 281–309, https://doi.org/10.1016/S0377-0427(00)00516-1.

[39] E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer, *Top*500: *The list of the* 500 *most powerful computer systems*, 2020, https://www.top500.org.

[40] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, San Diego, CA, 2001.

[41] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196, https://doi.org/10.1007/BF02238511.

[42] M. Wathen and C. Greif, *A scalable approximate inverse block preconditioner for an incompressible magnetohydrodynamics model problem*, SIAM J. Sci. Comput., 42 (2020), pp. B57–B79, https://doi.org/10.1137/19M1255409.

[43] J. Xu and L. Zikatanov, *Algebraic multigrid methods*, Acta Numer., 26 (2017), pp. 591–721, https://doi.org/10.1017/S0962492917000083.