A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores

Jean-Marc Gratien

A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores

Jean-Marc Gratien

Computer Science Department, 1 et 4 av Bois Préau, 92500 Rueil Malmaison, France

ARTICLE INFO

Article history:
Received 29 December 2018
Received in revised form 8 November 2019

Keywords:
Domain decomposition methods
Linear algebra
Parallel computing
Runtime systems
Multi-core architecture
Reservoir simulation

ABSTRACT

With the evolution of High Performance Computing, multi-core and many-core systems are a common feature of new hardware architectures. The required programming efforts induced by the introduction of these architectures are challenging due to the increasing number of cores. Parallel programming models based on the data flow model and the task programming paradigm intend to fix this issue. Iterative linear solvers are a key part of petroleum reservoir simulation as they can represent up to 80% of the total computing time. In these algorithms, the standard preconditioning methods for large, sparse and unstructured matrices – such as Incomplete LU Factorization (ILU) or Algebraic Multigrid (AMG) – fail to scale on shared-memory architectures with large number of cores. Multi-level domain decomposition (DDML) preconditioners recently introduced seem to be both numerically robust and scalable on emerging architectures because of their parallel nature. This paper proposes a parallel implementation of these preconditioners using the task programming paradigm with a data flow model. This approach is validated on linear systems extracted from realistic petroleum reservoir simulations. This shows that, given an appropriate coarse operator in such preconditioners, the method has good convergence rates while our implementation ensures interesting scalability on multi-core architectures.

1. Introduction

In basin modeling or reservoir simulations, multiphase porous media flow models lead to solve complex non-linear Partial Differential Equations (PDE) systems. These PDEs are discretized following a Finite Volume (FV) scheme which leads to a non-linear system solved with an iterative Newton solver. At each Newton step, the system is linearized and then solved with an iterative methods such as Biconjugate Gradient Stabilized (BiCGStab) [1] or Generalized Minimal RESidual (GMRES) [2] algorithms, well suited for large sparse and unstructured systems. Since this resolution part representing 60% up to 80% of the global simulation time, then the efficiency of linear solvers has a straightforward effect on the simulators performance. The choice of a robust preconditioner is therefore important to reduce the number of iterations required to converge and to optimize in that way the cost of such iterative methods.

Nowadays, modern hardware architectures with an increasing number of computational nodes, and with possibly several multi-core processors require an important programming effort. To be efficient on such architectures, several levels of parallelism need to be handled. The coarse grain-size level enables to handle parallelism at the node cluster level and to manage distributed memory. The finer grain-size levels deal with intra-node parallelism and local memory

E-mail address: jean-marc.gratien@ifpen.fr.

https://doi.org/10.1016/j.cam.2019.112614

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

management. Task programming paradigm is now a common way to manage all these levels of parallelism. Task based runtime systems provide tools to guarantee a good load balancing while taking care of data management.

The need of multi-level parallelism also induces difficulties from the algorithmic point of view. To reduce the number of iterations, preconditioners must not only be numerically robust regarding to reservoir simulation's cases but also scalable on multi-core systems. *Incomplete LU Factorization* (ILU) preconditioner – described in [3] – is widespread, but the method is not scalable on recent architectures. The *Algebraic Multigrid* (AMG) [4] is well-known in the reservoir simulation domain for its numerical performances with large, sparse and unstructured matrices. However, it has be shown that the parallel method has difficulties to take advantage of modern multi-core architectures [5,6]. Recently, *Multi-Level Domain Decomposition* (DDML) [7] preconditioner has been introduced and seems to be efficient on recent architectures. Such a preconditioner can also be tuned by choosing an appropriate coarse space correction such as the one presented in [8] which guarantees numerical robustness on heterogeneous coefficient problems.

The main contributions presented in this paper are:

- a way to parallelize preconditioning algorithms on multi-core machines by using task programming model using HARTS [9] runtime system.
- a Finite Volume adaptation of the GenEO method presented in [8] to handle linear systems coming from application based on Finite Volume discretization.
- an efficient parallel implementation of this method in the DDML preconditioner, using coarse and fine grain parallelism.
- a study of the numerical robustness and scalability performances of the preconditioner of linear systems coming from reservoir simulations.

This work is organized as follows: after introducing in Section 2 the computational architecture trends that motivates our work, Section 3 presents a way to implement algorithms taking advantage of different levels of parallelism and memory. Section 4 consists of an overview of common preconditioners used in the reservoir simulation domain. The DDML preconditioner with an adaptation of the GenEO method to handle linear systems coming from Finite Volume application is detailed in Section 5. Our parallel implementation and various performance issues on many-core systems are discussed in Section 6. Our work is compared with related work in Section 7 before concluding in Section 8.

2. Computer architecture and parallel programming model

Conceiving parallel algorithms is getting all the more challenging since new computer architectures are getting more and more complex. Nowadays, multi-cores chips invade large scale architectures, while the number of cores per node continues to increase. Taking advantage of multi-levels of parallelism is the key point to reach high performances. Moreover, applications benefit from several levels of memory. For this reason, data exchange optimization is now unavoidable. This section relies first on computer architecture trends. We then review some emerging tools that help to efficiently program parallel applications.

2.1. Computational trend

Nowadays, massively parallel systems based on multi-core architecture increase the complexity of programming. The rise of the number of cores imposes to extract even more parallelism from algorithms. As applications may benefit from a coarse-grain parallelism at node level and a fine-grain parallelism among available processors in a node, load balancing is therefore important among computing units to gain in efficiency. Memory hierarchy and data transfers between local memories has to be also considered. Communication optimization at any parallelism level is required to avoid memory contention and data latency.

2.2. Parallel programming model

Programming efficient programs for these multi-core architectures is challenging. Emerging runtime systems are based on parallel programming models. Task-based runtime systems enable to split computations in smaller pieces of work (i.e., a task), which are scheduled to favor efficient load balancing. Tasks are organized in a *Direct Acyclic Graph* (DAG), according to data flow computations. A task is moved on a computational unit then executed when it is in ready state – i.e., when there is no more dependencies on it.

Our work is based on the HARTS [9] runtime system which relies on abstract concepts to manage the layer between application and hardware. It therefore helps to distribute work between computation units and the associated data movements between the different memories. The library is based on a hardware model, a task model, a data model and an executing model. Its hardware model is based on the *Hardware Locality* software *Hwloc* [10] and enables to describe various heterogeneous architectures with different kinds of computer units, different levels of memories and different kind of connections between each units. Its task model enables to create and manage tasks objects with multiple representations for the target devices on which they may be executed. Task creation can be dynamic at runtime. All tasks are managed in a central task pool which can be operated by all the working threads. An algorithm is described as a DAG

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

with tasks and their associated dependencies. At runtime, a task can be replayed several times because of the persistence in memory of the DAGs of tasks. The Data model enables to encapsulate the data manipulated by task objects in *DataHandler* objects, managed and organized in a centralized data manager. It also provides data partitioning tools and partial views of each sub part of the data. Finally the Executing model provides concepts to describe how tasks can be executed on each specific devices with the better performance.

3. A task-based linear algebra framework for multi-level parallelism

To handle the complexity of programming efficient linear algebra algorithms on new parallel architectures, a specific API dedicated to the linear algebra domain has been developed. It is aimed at expressing at a high level linear algebraic algorithms with an implicit parallelism which hides the complexity of managing multi level parallelism, the various memory configurations and low level optimizations.

3.1. Abstract Linear Algebra API

Linear solver algorithms, in particular Krylov methods, can be viewed as sequences of successive linear algebra operations which are executed iteratively until reaching the convergence criteria. These operations are mostly level 1 or level 2 BLAS (Basic Linear Algebra Subprograms) vector operations, some sparse matrix–vector products or some specific sparse matrix preconditioning operations. We can consider for example a part of the BiCGStab [1] algorithm, described in Algorithm 1.

Algorithm 1: BiCGStab Algorithm

```
Matrix A;
Vector b, p, pp, r, v;
Scalar a;
do
    pp = inv(P).p;
    v = A.p;
    r += v;
    a = dot(p,r);
    if(a==0) break;
    ...;
while (|r|< tol * |b|);
```

These steps are implemented with our abstract algebraic API hiding the parallelism and the underlying technical implementation details. That helps to take advantage of complex architectures with a high level syntax as illustrated in Listing 1 for the BiCGStab algorithm. The API provides data allocators, most of the levels 1 to 2 BLAS functionalities and some specific preconditioning operations.

Listing 1: BiCGStab sequence

```
PartitionerType partitioner ;
AlgebraKernelType alg(partitioner);
Matrix A; Vector p,pp,r,v;
double alpha;
SequenceType seq = alg.newSequence();
alg.exec(precond,p,pp,seq);
alg.mult(A,pp,v,seq);
alg.axpy(1.,r,v,seq);
alg.dot(p,r,alpha,seq);
alg.assertNull(alpha,seq);
while(!iter.stop())
{
  alg.process(seq);
}
```

At the API level, parallelism is implicit and semantic is sequential. Each API's function calls sets of tasks according to the data partition and the algorithm.

A Partitioner object computes a data distribution according to the number of desired partitions. This object also provides data views and range iterators at execution time to operate task's data. Data flow dependencies are explicitly expressed. Tasks are then organized in a *Direct Acyclic Graph* (DAG). One of the main features of the API is to deal with iteration. A Sequence object is a sub-set of tasks that can be replayed several times to enable loop algorithms execution as in Listing 1.

3.2. Multi level parallelism

A Task decomposition is created from data structure partitioning. Sparse matrices are stored in *Compressed Sparse Row* (CSR) format. The parallelization of linear algebra algorithms is based on matrix dual graph partition techniques well

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

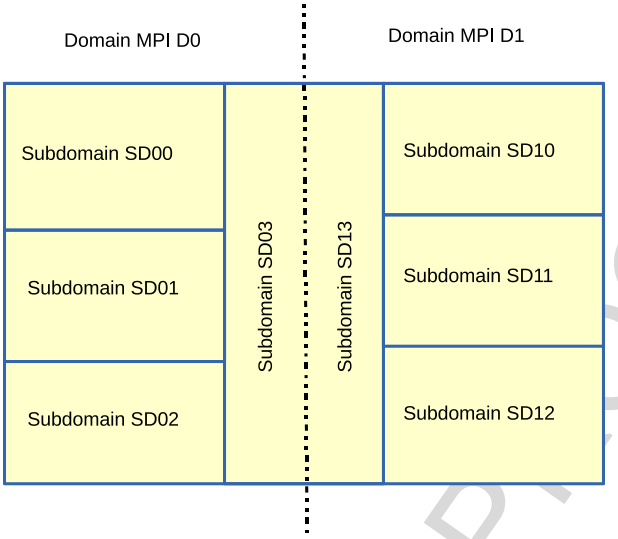

Fig. 1. Multi level domain partition.

described in [11]. The dual graph $G_A(V, E)$ of a matrix $A = (a_{i,j})$ is a set V of vertices v_i representing the rows i of A, and a set E of edges $e_{i,j}$ between vertices v_i and v_j representing non zero entries $a_{i,j} \neq 0$. Graph practitioners aim to split the matrix dual graph in sub-domains, while balancing the non-zero entries of the matrix and limiting the communication with the neighborhood. Associated to renumbering techniques, algebraic data are split regarding the graph partition, then gathered contiguously in memory for memory locality reasons. The graph partition is computed once and then handled by the API. The *Partitioner* object returns a set of range of matrix row indexes. They are associated with contiguous memory range. Algorithms are then split in tasks processing each of these ranges of memory.

Hierarchical partitioning algorithms enable to handle the distributed memory system and several levels of parallelism. The first level of partition aims at distributing data between MPI processes, at separating interface data that require synchronizations with MPI communication, and at handling coarse grain size parallelism. The other levels of partition aim at managing data locality and fine grain parallelism levels. Fig. 1 illustrates a two-level partition leading to 2 coarse domains (*MPI* D_0 and *MPI* D_1) processed by 2 MPI processes and 8 finer domains ($SD_{0,0}$, …, $SD_{0,3}$ for *MPI* D_0 and $SD_{1,0}$,…, $SD_{1,3}$ for *MPI* D_1) processed by 8 threads (4 by MPI processes) to handle MPI communications and fine level parallelism. In this figure, it is noticeable that the boundary between the two coarse domains is only shared by the two subdomains $SD_{0,3}$ and $SD_{1,3}$. Thus the subdomains $SD_{0,0}$, $SD_{0,1}$ and $SD_{0,2}$ of *MPI* D_0 are not connected to any subdomain of *MPI* D_1. In the same way the subdomains $SD_{1,0}$, $SD_{1,1}$ and $SD_{1,2}$ of *MPI* D_1 are not connected to any subdomain of *MPI* D_0. With such kind of partition, one can easily understand that only tasks related to $SD_{0,3}$ and $SD_{1,3}$ depend on MPI communications. The other tasks related to one MPI process can be executed independently of the tasks related to the other processes.

At the coarser level of parallelism, each MPI process manages algorithms related to level 0 coarse subdomains. These algorithms are then organized in smaller tasks related to level 1 finer subdomains. MPI communications are encapsulated in tasks related to level 1 subdomains connected to the boundaries between level 0 MPI subdomains. The parallelism between tasks of a same MPI processes relies on data dependencies and is managed by HARTS runtime system. These tasks may have access to data related to other subdomains of a same MPI process. Reducing the size of boundaries between level 1 subdomains aims to reinforce data locality and to prevent access to data that may be far in memory.

4. Preconditioning methods overview

Preconditioners aim to improve the convergence rates of iterative linear solver algorithms. The Incomplete Factorization preconditioner (ILU) [3], the Algebraic MultiGrid (AMG) preconditioner [4] are the most common methods used in reservoir simulation. The Multi-Level Domain Decomposition (DDML) methods have gained in popularity with the last developments proposed in recent research works that improve their convergence rates on heterogeneous problems. In this section, these preconditioning techniques and their parallel implementation with a task programming model are reviewed. First, ILU preconditioner is detailed, then AMG and finally DDML are considered.

4.1. Incomplete LU factorization (ILU)

Given a factorization of a large sparse matrix A such that

$$A = LU, \tag{1}$$

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

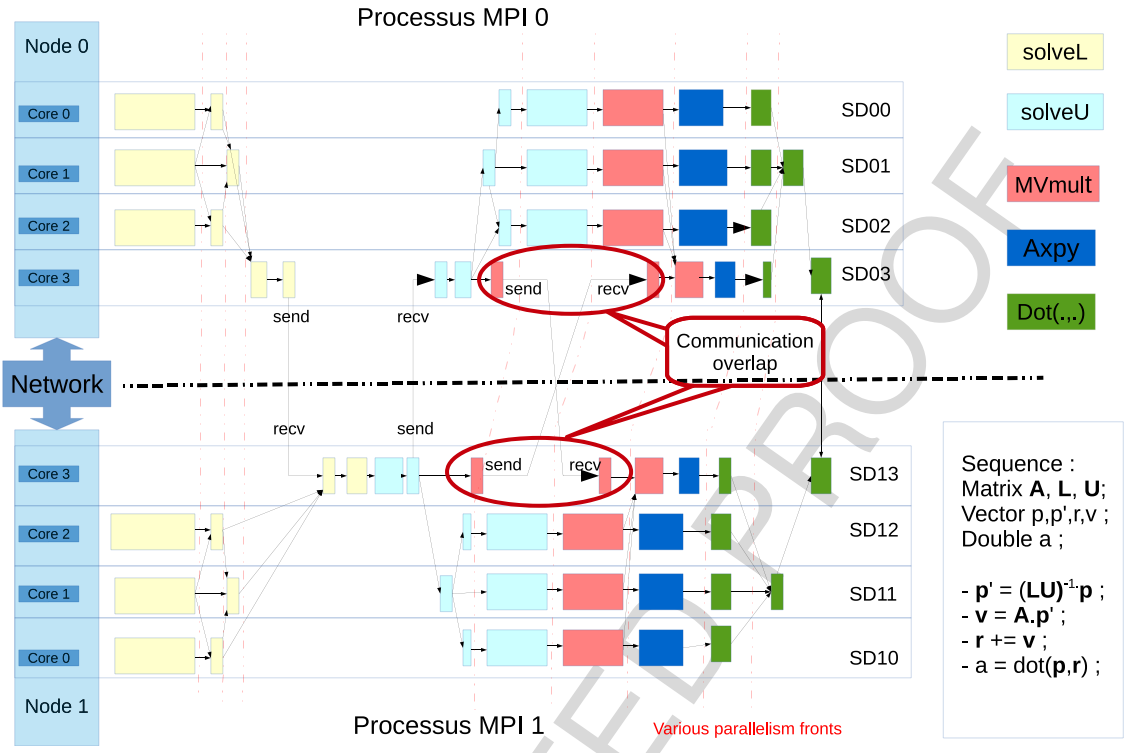

Fig. 2. ILU0 BicgStab sequence DAG.

where L is a lower triangular matrix, and U an upper triangular matrix. It is well known that usually in the factorization procedure, the matrices L and U have more non zero entries than A. These extra entries are called fill-in entries. The *incomplete LU factorization* (**ILU**) [11] consists in dropping some of these elements. In the zero degree incomplete version, ILU(0), all these entries are dropped. The preconditioning operation, not naturally parallel, solves $LUx = y$ with a backward substitution followed by a forward substitution.

This preconditioner, efficient for standard cases (i.e., not ill-conditioned and moderate problem size), is not naturally parallel, since its algorithm is recursive. This algorithm may be parallelized at a coarse grain size as presented in [12,13]. In that case, task programming associated with domain decomposition and renumbering techniques enables to extract different hidden levels of parallelism. Fig. 2 illustrates the DAG of the BiCGStab sequence written in Listing 1 with the ILU0 preconditioner and the partition of Fig. 1. The partitioner splits the global domain into 6 not connected interior domains and 2 interface domains connecting the previous ones. The chosen partitioner algorithm aims to maximize the size of the independent interior domains and to minimize the size of interface domains creating dependencies between all the domains. Thus, tasks associated with interior domains can be executed in parallel, while the runtime system scheduler extracts automatically a two levels parallelism between tasks associated with interface domains. We have implemented the *ILU*0, *ILU*0 − *MPI* and *ILU*0 − *MPIX* variant of the ILU0 preconditioner with respectively thread parallelism, MPI parallelism and with a hybrid MPI and thread parallelism.

In [14] a variant of the *ILU*0 algorithm is proposed. This variant is well adapted for fine grained parallelization. It consists in replacing the factorization, the backward and forward substitution by several steps of SpMV operations. That enables SIMD optimizations combined with task programming parallelization. This method denoted $ILU0_F$ avoids in one hand renumbering techniques which may have a negative impact on the matrix condition number. It depends in the other hand of two parameters $niter_F$ and $niter_S$, the number of steps to perform respectively the factorization and the resolution phase of the preconditioner. These parameters have an impact on the robustness of the method regarding the standard coarse grained parallelized algorithm *ILU*0. Indeed even if the parallelization of $ILU0_F$ may be more efficient, more iterations are often required nevertheless to converge with this preconditioner than with *ILU*0.

4.2. Algebraic Multigrid Methods (AMG)

Algebraic MultiGrid (**AMG**) [4] is a complex algorithm widely spread in domains dealing with diffusive problems because of its robustness properties on large sparse and unstructured systems. The setup phase is non-negligible in the total solving time because of the cost at each level of the construction of the coarse grid, the interpolation and the restriction

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

operators. The setup time may be longer than the solving time when the solver requires few iterations to converge. The solving phase is composed of two complementary operations, the smoothing and the coarse grid correction steps. The first one attempts to reduce high-frequency error by the application of a smoother, also called the relaxation method. The coarse grid correction eliminates low-frequency error. It performs a transfer of information to a coarser grid (also called the restriction operator), a coarse-grid system is then solved and finally during the interpolation operation, the solution is send back to the fine grid. AMG is an $O(n)$ method where the cost on each level is of the order of the number of degrees of freedom of the level. This algorithm is well known for its extensibility as its convergence rate is usually independent of system sizes for diffusive problems which is the case of reservoir simulation.

This preconditioner is provided by the Hypre [15] library under the name of *BoomerAMG* as a black box with both an OpenMP-based [16] and a MPI-based implementation. As in our current version, we cannot manage in a same run concurrent pool of threads, for instance the one provided by the OpenMP runtime system of the Hypre library and the one provided by HARTS, we focus only on the MPI-based implementation of BoomerAMG. We have integrated it in our task-based API using a coarse grained MPI parallelism paradigm. Following this formalism, the setup phase and the solving phase of AMG have been split in two different tasks which have been inserted in the DAG. In such MPI version, when we use only one thread by MPI processes, tasks are executed sequentially on each of them.

5. A parallel implementation of the DDML method for FV scheme

5.1. Multi-Level Domain Decomposition method

Multi-Level Domain Decomposition preconditioners are based on the classical Additive Schwarz method (ASM) [17] introducing a coarse grid correction in order to have a scalable method. The original ASM method is well known to be fully parallel however it suffers of slow convergence in iterative algorithms especially for large number of domains due to the lack of global information transfer on the all domain. This issue has been fixed by introducing a coarse space correction built on top of two main ingredients: a matrix $Z \in R^{n\times m}$ where n is the dimension of the linear system, $m \gg n$ and a coarse correction consisting in solving a coarse grid problem of size $m \times m$. The spirit of the method is to use a deflation technique by defining a coarse space spanned by the columns of the matrix Z representing the vectors responsible of the stagnation of the convergence of the ASM method. The variants of *Multi-level Domain Decomposition* (**DDML**) methods [7] differ from each other by the chosen coarse operator. From a linear algebra point of view, the stagnation of the convergence rate corresponds to a few very low eigenvalues in the spectrum of the preconditioned system.

Our work is focused on the Generalized Eigenvalue in the Overlap method (**GenEO**) [8], which incorporates the low frequency modes in the coarse grid construction. This method is interesting as it enables to achieve scalability on highly heterogeneous problem. The coarse operator is originally designed for finite element discretizations. It requires to build and then to solve independent generalized eigenvalue problems on each sub-domain. In the standard method this is realized using the basis function structures of the FE discretization method. This is not possible in the context of Finite Volume discretizations as such structures are not available. To build equivalent generalized eigenvalue problems, at a algebraic level, we have introduced some algebraic sub-matrices extraction procedure with modifications that mimic the behavior of FV boundary conditions on sub-domains boundaries.

To better explain the introduced extraction procedure, we consider for instance the following PDE representing an elliptic model diffusion problem 5.1:

We denote $\Omega \subset \mathbb{R}^d$ with $d \in \{1, 2, 3\}$, $\Gamma_D \cup \Gamma_N = \partial\Omega$

The problem reads:

find $u : \Omega \to \mathbb{R}$ verifying:

$$\begin{cases} \nabla \cdot (-\kappa \nabla u) = 0 & \text{in } \Omega, \\ u = g & \text{on } \Gamma_D, \\ \partial_n u = f & \text{on } \Gamma_N. \end{cases}$$

We suppose that in the discretization of our PDE on T_h, a simplicial triangulation of Ω, we need to solve the linear system $A\mathbf{x} = \mathbf{b}$ with $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$. The vector $\mathbf{x}$ represents the values of the discrete solution u_h on the degrees of freedom.

Domain decomposition methods consist in introducing a domain partition $\Omega = \cup_i \Omega_i$ with p sub-domains without overlap. Sub-domains completed with an overlap are denoted Ω_i^δ. We introduce p sub-meshes T_h^i with $T_h = \cup_i T_h^i$. At the continuous level, p independent sub-problems P_i are defined and their discretization leads to p independent linear systems $A_i\mathbf{x} = \mathbf{b}_i$.

To ensure the extensibility, DDML methods introduce a coarse grid correction with the form $I - ZE^{-1}Z^T$ where $E = Z^TAZ$ is the matrix of the coarse problem. The GenEO method consists in defining Z with the lowest eigenvectors of several independent generalized eigenvalue problems built by discretizing the Dirichlet-to-Neumann (DtN) map on each sub-domain, defined in the following way:

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

For any function $u_{\Gamma_i} : \Gamma_i \to R$
$u_{\Gamma_i} \to DtN_{\Omega_i}(u_{\Gamma_i}) = \frac{\partial v}{\partial n_i}|_{\Gamma_i}$ where v satisfies:

$$\begin{cases} \nabla \cdot (-\kappa \nabla v) = 0 & \text{in } \Omega_i, \\ v = u_{\Gamma_i} & \text{on } \Gamma_i, \\ v = 0 & \text{on } \partial\Omega_i \cap \partial\Omega. \end{cases}$$

The idea is to incorporate the smallest eigenvalues, responsible of the convergence stagnation.
We define the deflation matrix Z as:

$$Z = \begin{pmatrix} W_1 & 0 & \dots & 0 \\ 0 & W_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & W_p \end{pmatrix} \tag{2}$$

where $W_k = \left[D_k X_{k1}, \ \dots, \ D_k X_{k\nu_k}\right]$ are defined with ν_k eigenvectors kX_{k_i} corresponding to the ν_k smallest eigenvalues λ_{k_i} of the following generalized eigenproblems G_k: $DtN_{\Omega_k}(u) = \lambda u$.

The variational formulation reads: Find (v, λ) such that

$$\forall w, \int_{\Omega_k} \kappa \nabla v \cdot \nabla w = \lambda \int_{\partial\Omega_k} \kappa v w$$

At a algebraic level, this formulation can be written as follows:

$$\tilde{A}_k X_k = \lambda_k D_k \tilde{A}_k^0 D_k X_k$$

where:

- $\tilde{A}_k$ is restriction on Ω_k^δ.
- $\tilde{A}_k^0$ represents restriction on overlap.
- D_k stands for the Unity partition as defined in [7].

This defines p generalized eigenvalue problems $P_k : A_k x = \lambda B_k x$ of size n_k dimension of Ω_k^δ. The matrices A_k and B_k come from the discretization of the continuous generalized eigenvalue problems with the discretization scheme used to discretize the main PDE problem.

Let $T_h = \cup_k \tau_i$ where τ_i denotes the cells of the mesh. Let $\partial\tau_i = \cup_j \sigma_j$ where (σ_j) denotes the faces of the cells τ_i.

In the original context of Finite Element or Discontinuous Galerkin, it is generally possible to have access to the local matrices representing the local contribution of basis function to the global linear system. We denote ϕ_k the basis function of the FE discretization method. The matrix $\tilde{A}_k$ is built considering only the internal basis functions of Ω_k^δ, the basis functions related to the internal cells:

$$(\tilde{A}_k)_{i,j} = \sum_{\tau \in \Omega_k^\delta} a_\tau(\phi_j, \phi_i)$$

That only requires to know $\mathbf{A}^\tau = (a_\tau(\phi_k, \phi_l))_{k,l}$.

In the context of Finite Volume schemes, we have introduced an extraction procedure to build in an algebraic way the matrices A_i and B_i. This consists in extracting the two matrices from the assembled global matrix A, with some local transformations designed on each boundary Γ_i to mimic the behavior of partial assembly of basis function contribution. In Finite Volume methods based on flux formulation, the discretization procedure consists in integrating the strong formulation of the PDE on each cell τ_k :

$$eq_k : \int_{\tau_k} \nabla \cdot \kappa \nabla u.$$

In our system $A\mathbf{x} = \mathbf{b}$, the vector solution $x = (x_i)$ represents usually the discrete values of the piecewise constant function u_h on the degrees of freedom. The lines k of the matrix A related to the equation eq_k may be written for our model problem as follows:

$$\sum_{\sigma \in \partial\tau_k} F_\sigma(x) = 0.$$

where σ are the faces of τ_k and $F_\sigma(x) = \sum_{j \in J_\sigma} T_j^\sigma x_j$ represents the linear discretization of the flux across the face σ.

For $\tau_k \in \Omega_k^\delta$, considering the discrete flux $F_\sigma(x) = \sum_{j \in J_\sigma} T_j^\sigma x_j$, we can partition $J_\sigma = J_{int} \cup J_{ext}$ into J_{int} related to indices of degrees of freedom in Ω_k^δ and J_{ext} related to indices of degrees of freedom external to the domain. For instance, let us consider a two-point flux approximation. In Fig. 3, the equation related to the cell τ_i reads: $a_{i,i}x_i + a_{i,j_1}x_{j_1} + a_{i,j_2}x_{j_2} + a_{i,j_3}x_{j_3} =$

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

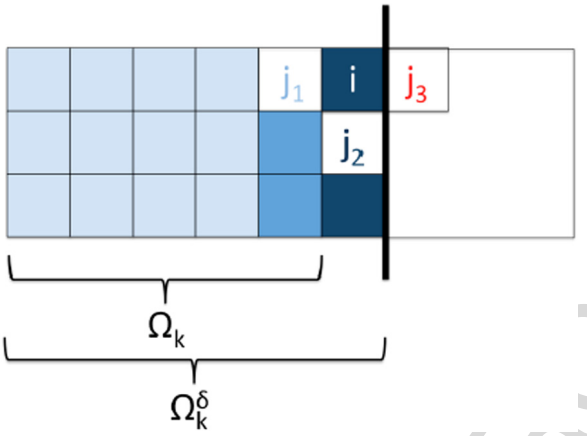

Fig. 3. FV discretization on the overlap.

0 with $a_{i,i} = T_{j_1} + T_{j_2} + T_{j_3}$, $a_{i,j_1} = T_{j_1}$, $a_{i,j_2} = T_{j_2}$ and $a_{i,j_3} = T_{j_3}$. The discretization of fluxes on the faces of τ_i involve the indices of $J = \{j_1, j_2, j_3\}$. We have then $J_{int} = \{j_1, j_2\}$ and $J_{ext} = \{j_3\}$.

At an algebraic level, considering only the global matrix, we cannot handle local matrices or partial assembly based on basis function. In order to mimic the discretization of the Neumann boundary condition with a Two-Point Flux Approximation, we introduce extra equations between external unknowns $(x_i)_{i\in J_{ext}}$ and internal ones $(x_i)_{i\in J_{int}}$. With these extra equations, external matrix entries can then be eliminated with a kind of Schur procedure that leads to modify the internal matrix entries. For instance, in the case of 3, with a two-point approximation of the Neumann boundary condition, we introduce the equation $x_i = x_{j3}$. We eliminate the extra diagonal entry T_{i,j_3} and modify the diagonal entry as follows $a_{i,i} = a_{i,i} + a_{i,j_3}$.

$$
\begin{aligned}
(\tilde{A}_k)_{i,j} &= \sum_{j\in V_i\cap\Omega_k^\delta} T_{ij}(u_i - u_j) \\
\tilde{A}_k &= \begin{pmatrix} A_{II} & A_{IO} \\ A_{OI} & \tilde{A}_{OO} \end{pmatrix}
\end{aligned}
\tag{3}
$$

The new restricted equation is written: $\tilde{a_{i,i}}x_i + a_{i,j_1}x_{j_1} + a_{i,j_2}x_{j_2} = 0$ with a TPFA diagonal correction: $\tilde{a_{i,j}} = a_{i,j} - a_{i,j_3}$

To summarize, our method consists in building each generalized eigenvalue problem for each sub-domain Ω_i^δ by extracting the matrices A_i and B_i from the global matrix A: only the lines related to the equations linked to the cells $\tau_k \in \Omega_i^\delta$ are considered; the external entries of these lines are dropped, some corrections are realized to mimic the discretization of a Neumann boundary condition.

5.2. Elements on the DDML preconditioner extensibility and robustness theoretical analysis

The GenEO preconditioner has been originally developed in the context of the Finite Element and Galerkin methods. A theoretical analysis of its extensibility and its robustness for diffusive problems is realized in [8].

The variational formulation of the diffusive problem 5.1 reads:

Find $u \in H_0^1(\Omega)$ such that $\forall v \in H_0^1(\Omega)$, $\int_\Omega \kappa \nabla u \cdot \nabla v = \int_\Omega f v$.

For any domain $D \subset \Omega$ we use the norms $\|.\|_{L_2(D)}$ and seminorm $|.|_{H_D^1}$

The discretization of 5.2 on a mesh T_h with a $\mathbb{P}_1$ Finite Element method leads to solve a linear system $A\mathbf{x} = \mathbf{b}$.

We denote $\lambda_{min} \leq \cdots \leq \lambda_m \leq \cdots \leq \lambda_{max}$ the eigenvalues of the preconditioned system $\mathbf{M}_{GenEO}^{-1}\mathbf{A}$, where $\mathbf{M}_{GenEO}^{-1}$ represents the GenEO preconditioner. We consider a domain partition $\Omega = \cup_i \Omega_i$ with p non overlapping sub-domains and Ω_i^δ the sub-domains completed with their overlap. We denote $\{E_i\}_{1\leq i\leq p}$ and $\{R_i\}_{1\leq i\leq p}$, a partition of unity related to $\{\Omega_i\}_{1\leq i\leq p}$ and its matrix representation.

The analysis detailed in [8] proves that the bound of the condition number:

$$
cond(\mathbf{M}_{GenEO}^{-1}\mathbf{A}) = \frac{\lambda_{max}}{\lambda_{min}}
$$

depends neither on the number of sub-domains p nor on the variation of the values of the tensor κ. Here are the key elements of this analysis.

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

Definition 1. Let be $k_0 = max_{\tau \in T_h}(\#\{\Omega_j : 1 \leq j \leq p, \tau \in \Omega_j\})$, the maximum number of sub-domains sharing one grid cell $\tau \in T_h$.

Lemma 1. $\lambda_{max}(\mathbf{M}_{GenEO}^{-1}\mathbf{A}) \leq k_0 + 1$

Definition 2. Given a coarse space $V_H \subset V_h$, some local sub-spaces $\{V_{h,0}(\Omega_j)\}_{1\leq j\leq p}$ and a constant C_0, a C_0-stable decomposition of $v \in V_h$ is a family of functions $\{z_j\}_{0\leq j\leq p}$ that satisfies:

- $v = \Sigma_{j=0}^{p} R_j^T z_j$, with $z_0 \in V_H$, $z_j \in V_{h,0}(\Omega_j)$ for $j \geq 1$
- and $\|z_0\|^2 + \Sigma_{j=1}^{p} \|z_j\|^2 \leq C_0^2 \|v\|^2$.

Theorem 2. *If every $v \in V_h$ admit a C_0-stable decomposition (with uniform C_0) then $\lambda_{min}(\mathbf{M}_{GenEO}^{-1}\mathbf{A}) \geq C_0^{-2}$.*

Corollary 1. $cond(\mathbf{M}_{GenEO}^{-1}\mathbf{A}) \leq C_0^2(k_0 + 1)$.

Lemma 3. *The GenEO method consists in building a C_0-stable decomposition with $C_0^2 = 2 + k_0(2k_0 + 1)max_{1\leq j\leq p}(1 + \frac{1}{\lambda^j_{m_j+1}})$.*

Considering the p generalized eigenproblems G_j defined in the previous section, a number of m_j of eigenvectors corresponding to the lowest eigenvalues of each G_j are selected to build a coarse space V_H and a family of projectors $\{\Pi_j\}_{1\leq j\leq p}$ to build the local sub-spaces $\{V_{h,0}(\Omega_j)\}_{1\leq j\leq p}$ of the C_0-stable decomposition.

Remark 1. For any $1 \leq j \leq p$, let $m_j := min\{m : \lambda^j_{m+1} > \frac{\delta_j}{H_j}\}$ where δ_j is a measure of the width of the overlap of Ω_j^δ and $H_j = diam(\Omega_j)$, then

$$cond(\mathbf{M}_{GenEO}^{-1}\mathbf{A}) \leq 2 + k_0(2k_0 + 1)max_{1\leq j\leq N}(1 + \frac{H_j}{\delta_j})$$

which proves the extensibility of the method regarding the number of sub-domains p and its robustness regarding the values of the tensor κ.

The DDML method aims to extend the GenEO method to linear systems coming from problems discretized with Finite Volume methods.

Using TPFA (Two-Point Flux Approximation) methods, the discretization of diffusive problems on Cartesian or K-orthogonal meshes leads to linear systems with properties equivalent to the ones obtained using $\mathbb{P}_1$ FE methods. It is then reasonable to extend the analysis realized for the GenEO method to this case.

For the large family of MPFA (Multi-Point Flux Approximation) [18], VAG (Vertex Approximation Gradient) [19], DDFV (Discrete Dual Finite Volume) [20] and Hybrid Mimetic Mixed (HMM) [21] methods, the Gradient Discretization (GD) formalism presented in [22] gives an abstract unified framework providing tools for functional analysis. Such framework including conforming and non conforming FE methods like the $\mathbb{P}_k$, the $\mathbb{RT}_k$ methods give a unified perspective enabling to extend the analysis of GenEO method. Considering then the extra stability terms, usually required to guaranty numerical properties such as coercivity, symmetry and positiveness of matrices, …of these methods, this perspective may enable to extend the DDML method to linear systems coming from the discretization of diffusive problem using more these large family of FV methods.

5.3. A task based parallel implementation with HARTS

Considering Z the deflation matrix, E the coarse matrix $E = Z^T AZ$, M the block diagonal matrix representing the ASM preconditioner, the DDML preconditioner algorithm consists in computing:

$$u^{m+1} = u^m + P^{-1}(b - Au_m)$$

as follows:

- Compute residual: $r^m = b - Au^m$
- Compute restriction: $r_c = Z^T r^m$
- Solve coarse problem: $Ev^m = r_c$
- Compute interpolation: $e^m = Zv^m$
- Compute residual: $r^{m+1} = b - A(u^m + e^m)$
- Solve local problem: $Me^{m+1} = r^{m+1}$
- Update iterate: $u^{m+1} = u^m + e^{m+1}$

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

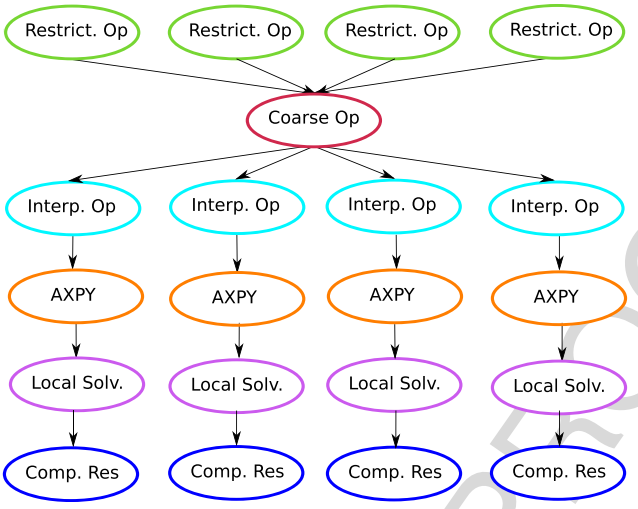

Fig. 4. Multi-Level Domain Decomposition meth. DAG.

This can be written algebraically as follows:

$$\begin{aligned} u^{m+1} &= u^m + ZE^{-1}Z^T(b - Au^m) + M^{-1}(b - A(u_m + ZE^{-1}Z^T(b - Au_m))) \\ &= u_m + \left[M^{-1}(I_n - AZE^{-1}Z^T) + ZE^{-1}Z^T\right](b - Au_m). \end{aligned}$$

Introducing the following operators:

- *Comp Res. Op*: $y = b - Ax$
- *Interp. Op*: $y = Zx$
- *Restric. Op*: $y = Z^T x$
- *Coarse Op*: $y = E^{-1}x$
- *Local Solv. Op* $y = M^{-1}x$

we have implemented our DDML method with the task programming framework based on HARTS described in Section 3.1. The algorithm is composed of several parallel operations (*Comp Res. Op*, *Interp. Op*, *Restric. Op* and *Local Solv. Op*) that can be applied independently to each partition domain. There is also *Coarse Op* the coarse operation applied to solve a global coarse system. All these operations are implemented within tasks organized in a DAG as illustrated in Fig. 4.

The two most consuming phases are the resolution of the local problems implemented by the *Local Solv* tasks and the resolution of the coarse problem implemented the *Coarse Op* task. To solve the local problems we use either direct LU solvers or iterative methods (such as BiCGStab or ILU(0)-preconditioned BiCGStab method. The coarse problems with generally small sizes are solved with a direct sparse LU solver. Linear solver algorithms are implemented with structures of the Eigen [23] library. We use also implementations with our own data structures and algorithms with specific optimizations like SIMD instructions. BLAS operations (*AXPY* tasks), the deflation operations (*Interp. Op*, *Restric. Op*) are implemented using the MKL library.

5.4. Parallelization issues on shared and distributed memory architecture

To build algebraically the independent general eigenvalues problems with only the global assembled matrix, various parallel issues have to be handled. The parallelization of the method is based on a multi level partition of the mesh T_h on which the PDE problem has been discretized. The first level aims to manage coarse grain parallelism with MPI, while the other levels aim to manage fine grain parallelism. Fig. 5 illustrates a two level mesh partition with two MPI domains $P0$ and $P1$. Each of the MPI domains is partitioned at a second level in two other sub-domains. $P0$ is divided in the two sub-domains *sD0P0* and *sD1P0*. For each sub-domain Ω_k, the overlap is divided in two parts, the first one related to the intersection between sub-domains of the same MPI domain (for instance *SD0P0* and *sD1P0*), and a second one related to the intersection of sub-domains of different MPI domains (for instance *sD0PO* and sub-domains of the MPI domain $P1$).

Data related to overlaps shared between sub-domains are duplicated. Some synchronizations are required then to ensure the coherency of these duplicated data. Within a same MPI domain, these synchronizations are realized with simple data copy operations thanks to the shared memory. The overhead due to the cost of such operations is related to the performance of the memory bandwidth. For the overlaps between sub-domains that do not belong to the same MPI domain, some MPI communications are required. The overhead due to the cost of these communications is related to the performance of the interconnect network.

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

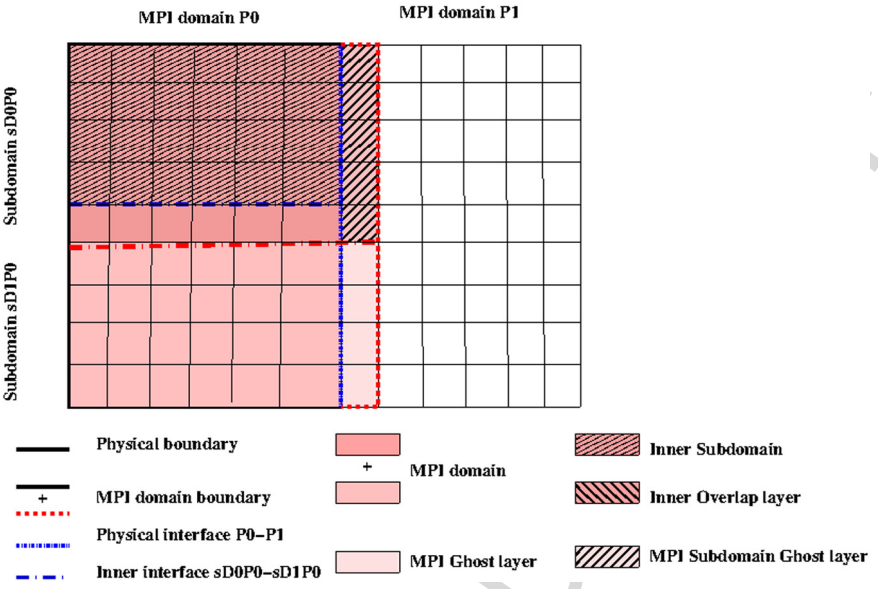

Fig. 5. Two level partition for Hybrid MPI-X parallelism.

Table 1
Number of iterations.

Task Op	Size	S(n_p)
LocalSolv. Op	$n_l * n_l$	↘
(Rest./Inter.)Op	$n_l * n_{ev}$	↘
(Blas,Res)Op	n_l	↘
Coarse Op	$n_c * n_c$	↗

In the building phase of the generalized eigenvalue problem $A_i\mathbf{x} = \lambda B_i\mathbf{x}$ on each sub-domain Ω_i^δ, the extraction of the matrices A_i and B_i requires matrix manipulations on the entries related to the intersection of Ω_i^δ with the other sub-domains Ω_j. When the two sub-domains belongs to the same MPI domain, these manipulations are simple as all equation entries are available in memory. This is not the case when they belong to two different MPI domains. The matrix entries of the external part of the matrix are not available and some MPI communications are required. The restricted equation extraction has to be realized on the processor where the whole equation is available in memory, then a communication is required to send the results to the processor on which the eigenvalue problem is built. For instance on sub-domain *sD0P0* the manipulations related to the overlap shared with the MPI domain *P*1 have to be computed on the MPI process managing domain *P*1. The results have then to be communicated to the processor managing domain *P*0 where the generalized eigenproblem of sub-domain *sD0P0* is built and solved.

5.5. Parallel performance analysis

In this section we analyze the influence of the sizing parameters of the DDML algorithm on the performance of the method. We denote n_t the global size of the linear system, n_p the number of sub-domain of the mesh partition and n_{ev} the number of eigenvalues used to build the coarse system. We denote n_l the average size of the local problem on each sub-domains and n_c the size of the coarse system. We have then $n_l = \frac{n_t}{n_p}$ and $n_c = n_p * n_{ev}$. To ensure the extensibility of the methods n_{ev} could be automatically determined with the formula given by [8]. In practice, n_{ev} is increased manually according to n_p.

In Table 1 we gather the sizes of the main algebraic operators involved in the algorithm and their evolution with respect to n_p the number of sub-domains.

Analyzing Table 1, we can notice that the cost of the parallel operators *Local Solv. Op*, *(Rest./Inter.) Op*, *(Blas,Res) Op* decreases when n_p increases. As these problems are independent and can be solved in parallel, increasing the number of domains reduces the global cost of these local operators. The cost of the coarse operator is proportional to the number of partitions times the number of eigenvalues to incorporate. The performance of this sequential part increases then with

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

Table 2
Available data structures and direct solver algorithms.

Matrix format	Algo	Mem	Optim
CSR	SparseLU	Low	seq
Dense	DenseLU	High	seq
Bande	SIMDBDLU	Medium	simd
BlockTile	ParLU	High	par

the number of sub-domains. Thus a compromise has to be done to optimize the performance of the parallel region while limiting the cost of coarse operator. The associated task of this last operator is indeed a synchronization point for all the threads. To reduce the complexity of the global algorithm, we chose a number of partitions that enables to reduce the cost of each local resolution and to fit local memory caches while preserving a size of the coarse problem that limits the penalty of this synchronization part preventing it to become a real bottleneck.

We can analyze also the impact of the sizing parameters on the choice of data structures and solver algorithms. In Table 2 we gather the various available matrix formats and solver algorithms. We indicate their impact on memory and eventually their optimization features.

Analyzing this table, we can understand that the choice of the matrix format and the solver algorithms deeply depends on the sizing parameters (n_t, n_p and n_{ev}). Regarding the target hardware architecture, the algorithm to choose the best matrix format and algorithms for *LocalSolv. Op* and the *CoarseOp* may be complex and require expertise.

6. Experiments

6.1. Experimental protocol

The experiments have been run on a node of a linux machine[1] with a Knights Landing processor which handles 32 bi-cores processors, 16 GB of MCDRAM high bandwidth memory and 64 GB of DRAM standard memory. The KNL processor cores topology was configured with the Quadrant mode. The MCDRAM memory was configured as a cache for DRAM.

The library has been compiled with the Intel 2018 compiler with GCC 7.3 compatibility, activating AVX512 vector instructions. Data structures are split via a graph partitioner based on the Metis library. All the BLAS kernels are performed with the MKL library with $MKL\_NUM\_THREADS = 1$ to deactivate the internal library multi-threading feature. The coarse systems of the DDML preconditioners are solved with the *SuperLU* algorithm provided by the *Eigen 3.4* library. For the local systems on each subdomains, either a AVX512 optimized LU algorithm with matrices with a Band Structure format is used, or the *SuperLU* algorithm of *Eigen 3.4* library with matrices with the CSR format.

The linear systems used for the experiments were extracted either from simple 2D Laplacian problems on a unit square discretized with grids of size $N_x \times N_x$ leading to systems of size $N_{rows} = N_x * N_x$, or from the simulation of the well known realistic reservoir study case SPE10 [24] leading to systems of size $N_{rows} = 10^6$.

6.2. Sizing parameters influence performance analysis

In this section, we analyze the influence on the performance of the solver of the sizing parameters N_{rows} the linear system global size, n_p the number of subdomains used in the DDML method and n_{ev} the number of eigenvalues used to build the coarse system. We solve linear systems with Laplacian matrices of sizes $N_{rows} = 2.5 * 10^4$ and $N_{rows} = 10^6$. We evaluate the performance using $n_p = 64$ and $n_p = 128$ and with values of n_{ev} in $\{4, 8, 16, 24, 32\}$.

The benchmark is realized with two configurations of the solver, the first one C_{th} with a thread based parallelization and a second one C_{mpi} with an MPI based parallelization.

In Figs. 6 and 7 we plot in function of the number of cores nc, the inverse $\frac{1}{T_{n_c}}$ of the execution time T_{n_c} of the resolution with configuration C_{th}, for $N_{rows} = 2.5 * 10^4$ and $N_{rows} = 10^6$, respectively.

In Figs. 8 and 9, we plot as well the same results with configuration C_{mpi}, for $N_{rows} = 2.5 * 10^4$ and $N_{rows} = 10^6$, respectively.

Analyzing these figures, we can see the effects of both n_p and n_{ev} on the performances of the solver. Considering the smaller case, $N_{rows} = 2.5 * 10^4$ the best performances are obtained for $n_p = 64$ while for the larger case, $N_{rows} = 10^6$, the best performances are obtained for $n_p = 128$. We observe the influence of n_{ev} on the performances depending on both N_{rows} and the number of cores nc. We obtain the best performance with $nc = 32$ and $n_{ev} = 24$ for $N_{rows} = 2.5 * 10^4$, and with $nc = 32$ and $n_{ev} = 32$ for $N_{rows} = 10^6$. The runs on 64 cores suffer from the fact that the resolution of the coarse problem is not parallelized, and this effect is all the more important since n_{ev} or n_p is high. All these observations are coherent with the fact that the size of the coarse system is equal to $nev * n_p$.

[1] Linux machine of LIP at ENS Lyon.

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

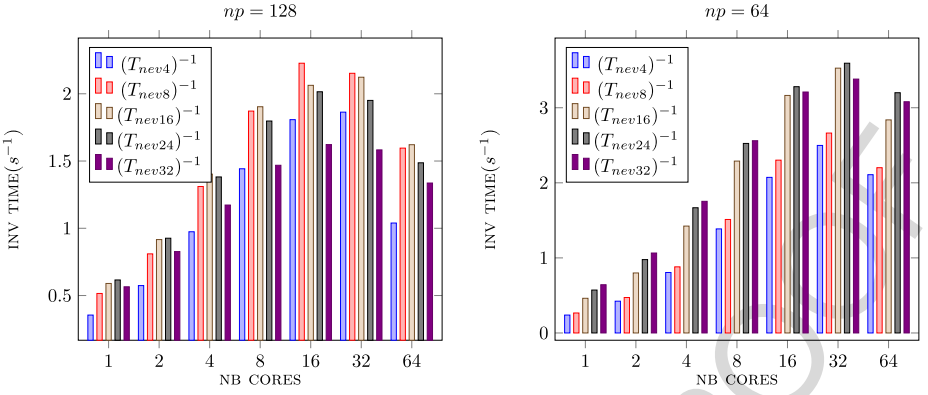

Fig. 6. Configuration C_{th}, Laplacian matrices: $N_{rows} = 2.510^4$.

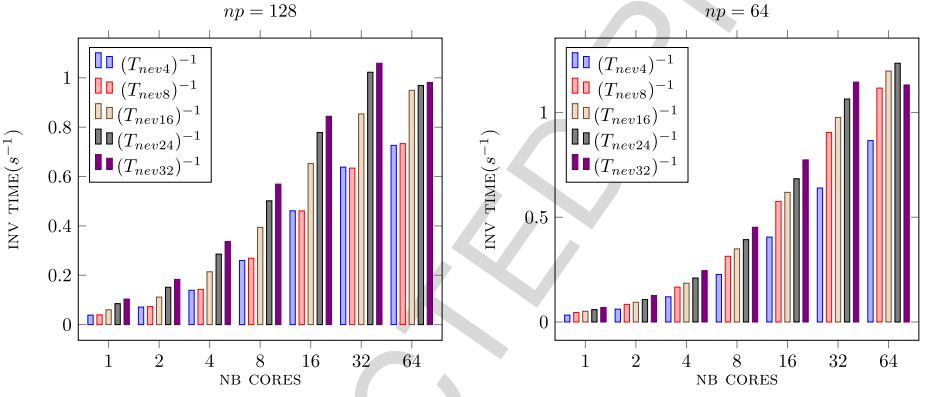

Fig. 7. Configuration C_{th}, Laplacian matrices: $N_{rows} = 10^6$.

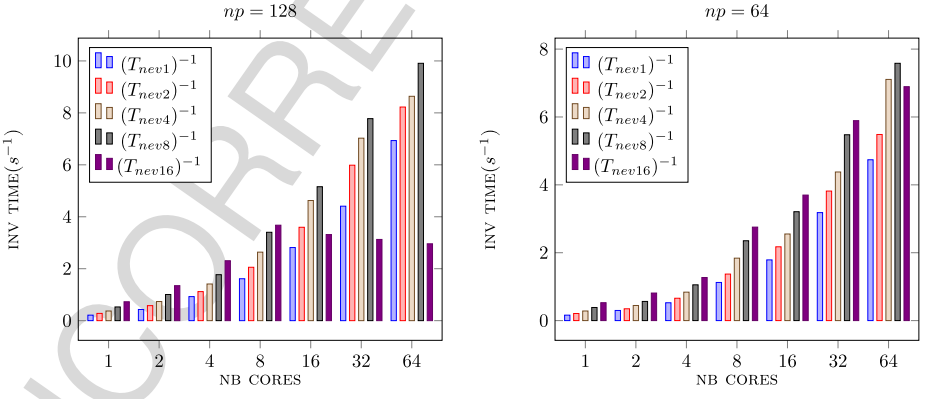

Fig. 8. Configuration C_{mpi}, Laplacian matrices: $N_{rows} = 2.510^4$.

6.2.1. *Preconditioners benchmark on SPE10 study case*

We tested different configurations of the preconditioners on a linear system extracted from a simulation of the SPE10 reservoir study case. This study case is well known for its highly heterogeneous data leading to ill conditioned linear systems. For the ILU(0) preconditioner, we test the *ILU*0, $ILU0_{64}$ and $ILU0_{mpi}$ configurations corresponding respectively to the multi-thread version using a number of partitions equal to the number of cores and to the multi-thread version using 64 partitions and to the MPI version. For the DDML preconditioner, we have tested the $DDML_4$, $DDML_8$, $DDML_{16}$ and $DDML_{32}$ configurations corresponding to a number of eigenvalues *nev* equal to respectively 4, 8, 16 and 32, and a number of partitions equal to 256. We have also tested the AMG preconditioner using the MPI version of the Hypre library without multi-threading. The option parameters of the AMG algorithm have been tuned for linear systems extracted from realistic

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

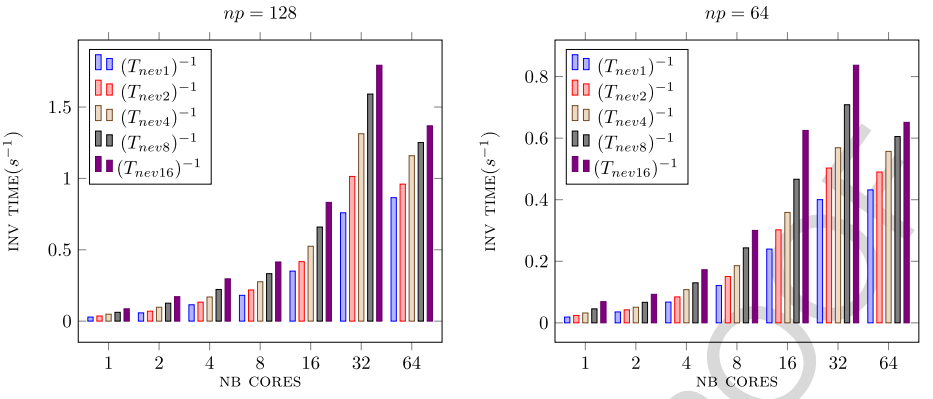

Fig. 9. Configuration C_{mpi}, Laplacian matrices: $N_{rows} = 10^6$.

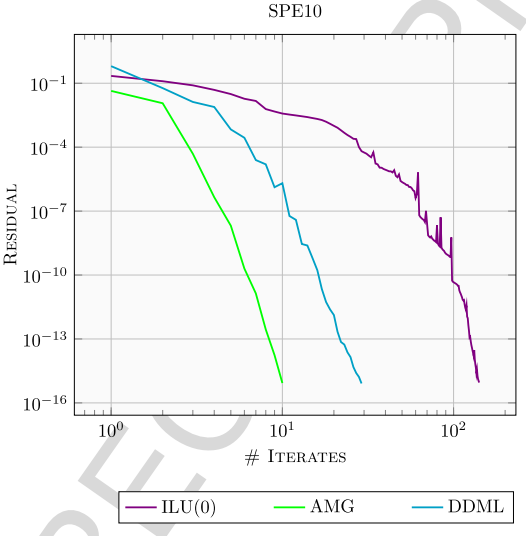

Fig. 10. Comparison of the residual of preconditioned BiCGStab of SPE10 matrix system.

Table 3
Number of iterations.

*ILU*0	$ILU0_{64}$	$ILU0_{mpi}$	*AMG*
370	382	370	3
$DDML_4$	$DDML_8$	$DDML_{16}$	$DDML_{32}$
33	15	9	6

reservoir simulation. The strong threshold is set to 0.15, the coarsening option to PMIS with the extended interpolation option. The other parameters are set to the default settings of the version 2.10 of the library.

In Fig. 10 we plot the residual of the linear solver in function of the number of iterations.

The analysis of the figure shows that the convergence behavior of the DDML method is close to the behavior of the AMG preconditioner, the state of art preconditioner in reservoir simulation.

In Table 3 we gather the number of iterations N_{iter} required to converge for a tolerance value of 10^{-4}.

We plot in Fig. 11(a) the execution time in function of the number of cores n_c. In 11(b) we plot the parallel speed-up $S = \frac{T_1}{T_{n_c}}$ where T_1 and T_{n_c} are the execution time on 1 and n_c cores, respectively.

For the preconditioner DDML, we can see in Table 3 that when the number of eigenvalues nev grows, N_{iter} decreases. The best performances are obtained for $nev = 16$. The DDML preconditioner has a very good parallel efficiency up to the 32 cores. Beyond 32 cores, its efficiency decreases. This is due to the fact that the coarse solver, not yet parallelized, becomes a bottleneck all the more since its cost grows with nev as for instance $nev = 32$. The ILU0 preconditioner is the less efficient preconditioner. The required number of iterations N_{iter} is much greater than for the two previous preconditioners. The parallel efficiency of *ILU*0 decreases quickly with the number of cores n_c.

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

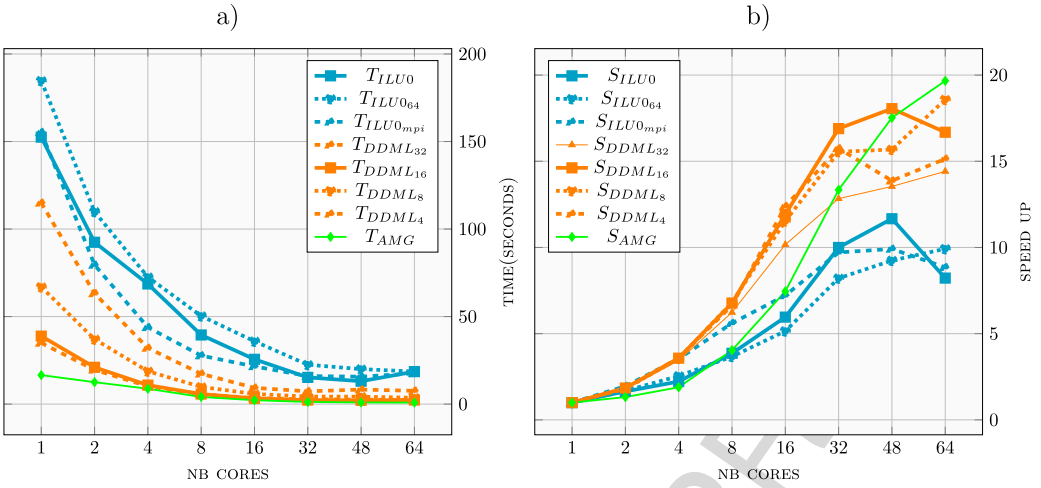

Fig. 11. KNL Single node performance results.

7. Related works

Our work has been motivated by the works described in the reference duplication [7] in which multi level decomposition domain methods are studied on a large range of diffusive problems, among them Darcy problems, elasticity problems, Helmholtz problems …. Most of these works refer to Finite Element discretization methods. In [25] the authors realize a performance study on a large scale multi-node cluster with MPI parallelization. In [26], an overview of Two-Level methods is realized and the DDML method is compared to other similar multi level methods. The AMG remains a state of art method in Reservoir simulation and its MPI version is very efficient. In [16], some improvements of AMG for shared memory machines are presented. These improvements rely mainly on the OpenMP paradigm. Other Multi level methods are available in package like Trilinos (the ML method) [27] and Dune-ISTL [28].

The API of our framework is very similar to those provided by popular frameworks like PETSc [29], Hypre [15], Trilinos [30]: it exposes at a high level linear algebra functions while hiding the low level complexity of parallelism thanks to a *Partitioner* object aimed at splitting matrices and vectors into continuous and coherent sub vectors and sub matrices. Definitely the originality of our approach is to clearly separate the definition in a declarative way of algorithms by sequences of function objects, from the execution of these sequences that can be performed several times within iterative loops. That enables cross optimizations between different linear operators within each sequence, but also at the iterative level for multiple executions of a same sequence. Such optimizations are more difficult to realize in frameworks where operations are provided as black boxes. Recently, developers of PETSc introduce the GPU implementation of the library [31]. However, it shows that moving from one architecture to another is not easy and requires a huge programming effort. In our work, we can show how such integration like for instance the introduction of new kernel implementations with SIMD optimizations, is easier in our framework designed on top of a runtime system. Contrary to PETSc for the GPU implementation, we do not have to take care about data allocation and movement between the CPU and the GPU as it is managed through the runtime system.

Approaches based on runtime systems for sparse linear algebra with large unstructured matrices is only tackled in few research works like in the GHOST framework [32]. In fact, runtime systems like Cilk [33] do not manage data dependencies while the well known ones like Ompss [34], StarPU [35] have proved their efficiency for coarse grained parallelism, but not really for very fine grained parallelism. This issue, all the more important for sparse linear algebra since many-core processors or GP-GPU become a common feature in hardware architectures, is discussed on the other hand in [36] with Xkaapi runtime.

8. Conclusion and future work

This paper demonstrates the interest of using an abstract linear algebraic API aiming at implementing parallel algorithms. The API has been developed on top of HARTS, a task-based runtime system which manages complex parallel architectures. It provides common sparse linear algebra kernels. A distributed task queue is set up to enhance load balancing and a scheduling policy is provided to enforce data locality.

Furthermore an adaptation of the GenEO method presented in [8] has been proposed to handle at an algebraic level linear systems coming from the Finite Volume discretization of diffusive problem. Some details on the implementation of the proposed variant of the GenEO method in the DDML preconditioner with our task based linear framework have been given.

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

This new approach has been benchmarked with large sparse linear systems coming from reservoir simulation, showing the ability of the API to provide efficient preconditioners on KNL architectures with both thread based and MPI based parallelism. It also confirms that DDML preconditioner is able to scale very well on this architectures, meanwhile some optimizations are still required on the local and coarse solvers. In reservoir simulation, with highly heterogeneous data that lead to ill conditioned linear system, AMG remains a state of art preconditioner.

As a perspective, to become really competitive regarding the AMG preconditioner, we still need to improve the numerical behavior of the DDML method to be able to reduce the number of iterations as much as AMG. Concerning our implementation, we will introduce parallel direct solver to solve the coarse system for large number of domains and eigenvectors. We are working on a hybrid version DDML-MPIX of our DDML preconditioner which can perform on several nodes with KNL processors or standard multi-core processors like our ILU0-MPIX preconditioner. Even if the KNL micro architecture will not be anymore supported by Intel, our methodology remains interesting for the recent architectures like the Skylake, KabyLake that integrate AVX512 instructions, large number of cores, right now up to 28 per processor, up to 56 within one dual socket node.

References

[1] H.A. van der Vorst, Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 13 (2) (1992) 631–644, arXiv:http://dx.doi.org/10.1137/0913035.

[2] Y. Saad, M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (3) (1986) 856–869, arXiv:http://dx.doi.org/10.1137/0907058.

[3] S.T. Barnard, L.M. Bernardo, H.D. Simon, An MPI implementation of the SPAI preconditioner on the t3E, Int. J. High Perform. Comput. Appl. 13 (1999) 107–128.

[4] A. Brandt, S.F. Mccormick, J.W. Ruge, in: D.J. Evans (Ed.), Algebraic Multigrid (AMG) for Sparse Matrix Equations, Cambridge University Press, New York, 1984.

[5] A.H. Baker, T. Gamblin, M. Schulz, U.M. Yang, Challenges of scaling algebraic multigrid across modern multicore architectures, in: Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International, 2011, pp. 275–286, http://dx.doi.org/10.1109/IPDPS.2011.35.

[6] J. Park, M. Smelyanskiy, U.M. Yang, D. Mudigere, P. Dubey, High-performance algebraic multigrid solver optimized for multi-core based distributed parallel systems, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15, ACM, New York, NY, USA, 2015, pp. 54:1–54:12, http://dx.doi.org/10.1145/2807591.2807603.

[7] V. Dolean, P. Jolivet, F. Nataf, An Introduction to Domain Decomposition Methods, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2015, arXiv:http://epubs.siam.org/doi/pdf/10.1137/1.9781611974065.

[8] N. Spillane, V. Dolean, P. Hauret, F. Nataf, C. Pechstein, R. Scheichl, A robust two-level domain decomposition preconditioner for systems of PDEs, C. R. Math. 349 (23) (2011) 1255–1259, http://dx.doi.org/10.1016/j.crma.2011.10.021.

[9] J.M. Gratien, An abstract object oriented runtime system for heterogeneous parallel architecture, in: Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International, 2013, pp. 1203–1212, http://dx.doi.org/10.1109/IPDPSW.2013.144.

[10] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, R. Namyst, hwloc: a generic framework for managing hardware affinities in HPC applications, in: PDP 2010 - the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing, Pisa, IEEE, Italy, 2010, http://dx.doi.org/10.1109/PDP.2010.67.

[11] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[12] J. Magras, P. Quandalle, P. Bia, et al., High-performance reservoir simulation with parallel ATHOS, in: SPE Reservoir Simulation Symposium, Society of Petroleum Engineers, 2001.

[13] J.-M. Gratien, T. Guignon, J.-F. Magras, P. Quandalle, O.M. Ricois, et al., Scalability and load balancing problems in parallel reservoir simulation, in: SPE Reservoir Simulation Symposium, Society of Petroleum Engineers, 2007.

[14] E. Chow, A. Patel, Fine-grained parallel incomplete LU factorization, SIAM J. Sci. Comput. 37 (2) (2015) C169–C193.

[15] R.D. Falgout, J.E. Jones, U.M. Yang, The design and implementation of hypre, a library of parallel high performance preconditioners, in: A.M. Bruaset, A. Tveito (Eds.), Numerical Solution of Partial Differential Equations on Parallel Computers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 267–294, http://dx.doi.org/10.1007/3-540-31619-1_8.

[16] C. Feng, S. Shu, X. Yue, An improvement to the OpenMP version of BoomerAMG, in: Y. Zhang, K. Li, Z. Xiao (Eds.), High Performance Computing: 8th CCF Conference, HPC 2012, Zhangjiajie, China, October 29–31, 2012, Revised Selected Papers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 1–11, http://dx.doi.org/10.1007/978-3-642-41591-3_1.

[17] H.A. Schwarz, Uber einen grenzubergang durch alternierendes verfahren, 1870, pp. 272–286.

[18] I. Aavatsmark, T. Barkve, Ø. Bøe, T. Mannseth, Discretization on non-orthogonal, quadrilateral grids for inhomogeneous, anisotropic media, J. Comput. Phys. 127 (1) (1996) 2–14.

[19] R. Eymard, C. Guichard, R. Herbin, R. Masson, Vertex-centred discretization of multiphase compositional Darcy flows on general meshes, Comput. Geosci. 16 (4) (2012) 987–1005.

[20] F. Boyer, F. Hubert, S. Krell, Nonoverlapping Schwarz algorithm for solving two-dimensional m-DDFV schemes, IMA J. Numer. Anal. 30 (4) (2010) 1062–1100.

[21] J. Droniou, R. Eymard, T. Gallouët, R. Herbin, A unified approach to mimetic finite difference, hybrid finite volume and mixed finite volume methods, Math. Models Methods Appl. Sci. 20 (02) (2010) 265–295.

[22] J. Droniou, R. Eymard, R. Herbin, Gradient schemes: generic tools for the numerical analysis of diffusion equations, ESAIM Math. Model. Numer. Anal. 50 (3) (2016) 749–781.

[23] G. Guennebaud, B. Jacob, et al., Eigen v3, 2010, http://eigen.tuxfamily.org.

[24] M. Christie, M. Blunt, Tenth SPE comparative solution project: A comparison of upscaling techniques, SPE Reserv. Eval. Eng. 4 (2) (2001) 308–317.

[25] P. Jolivet, F. Hecht, F. Nataf, C. Prud'Homme, Scalable domain decomposition preconditioners for heterogeneous elliptic problems, in: SC13 - International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, Denver, United States, 2013, pp. 80:1–80:11, http://dx.doi.org/10.1145/2503210.2503212.

[26] J.M. Tang, R. Nabben, C. Vuik, Y.A. Erlangga, Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods, J. Sci. Comput. 39 (3) (2009) 340–370, http://dx.doi.org/10.1007/s10915-009-9272-6.

[27] M.W. Gee, C.M. Siefert, J.J. Hu, R.S. Tuminaro, M.G. Sala, ML 5.0 Smoothed Aggregation User's Guide, Tech. Rep., Technical Report SAND2006-2649, Sandia National Laboratories, 2006.

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.

[28] M. Blatt, P. Bastian, The iterative solver template library, in: International Workshop on Applied Parallel Computing, Springer, 2006, pp. 666–675.
[29] PETSc home page, 2018, https://www.mcs.anl.gov/petsc/.
[30] Trilinos home page, 2018, https://trilinos.org.
[31] V. Minden, B. Smith, M.G. Knepley, Preliminary implementation of PETSc using GPUs, in: D.A. Yuen, L. Wang, X. Chi, L. Johnsson, W. Ge, Y. Shi (Eds.), GPU Solutions to Multi-Scale Problems in Science and Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 131–140, http://dx.doi.org/10.1007/978-3-642-16405-7.
[32] M. Kreutzer, J. Thies, M. Röhrig-Zöllner, A. Pieper, F. Shahzad, M. Galgon, A. Basermann, H. Fehske, G. Hager, G. Wellein, GHOST: building blocks for high performance sparse linear algebra on heterogeneous systems, Int. J. Parallel Program. 45 (5) (2017) 1046–1072.
[33] R.D. Blumofe, C.F. Joerg, B.C. Kuszmaul, C.E. Leiserson, K.H. Randall, Y. Zhou, Cilk: An efficient multithreaded runtime system, J. Parallel Distrib. Comput. 37 (1) (1996) 55–69, http://dx.doi.org/10.1006/jpdc.1996.0107.
[34] E. Ayguadé, R.M. Badia, P. Bellens, D. Cabrera, A. Duran, R. Ferrer, M. Gonzàlez, F. Igual, D. Jiménez-González, J. Labarta, L. Martinell, X. Martorell, R. Mayo, J.M. Pérez, J. Planas, E.S. Quintana-Ortí, Extending OpenMP to survive the heterogeneous multi-core era, Int. J. Parallel Program. 38 (5) (2010) 440–459, http://dx.doi.org/10.1007/s10766-010-0135-4.
[35] C. Augonnet, S. Thibault, R. Namyst, P.-A. Wacrenier, StarPU: a unified platform for task scheduling on heterogeneous multicore architectures, Concurr. Comput. Pract. Exp. 23 (2) (2011) 187–198.
[36] J.V.F. Lima, T. Gautier, V. Danjean, B. Raffin, N. Maillard, Design and analysis of scheduling strategies for multi-CPU and multi-GPU architectures, Parallel Comput. 44 (2015) 37–52, http://dx.doi.org/10.1016/j.parco.2015.03.001.

Please cite this article as: J.-M. Gratien, A robust and scalable multi-level domain decomposition preconditioner for multi-core architecture with large number of cores, Journal of Computational and Applied Mathematics (2019) 112614, https://doi.org/10.1016/j.cam.2019.112614.