HAL
open science

High performance domain decomposition methods on massively parallel architectures with FreeFEM++

Pierre Jolivet, Victorita Dolean, Frédéric Hecht, Frédéric Nataf, Christophe Prud'Homme, Nicole Spillane

To cite this version:

Pierre Jolivet, Victorita Dolean, Frédéric Hecht, Frédéric Nataf, Christophe Prud'Homme, et al.. High performance domain decomposition methods on massively parallel architectures with FreeFEM++. Journal of Numerical Mathematics, 2012, 20 (3-4), pp.287-302. 10.1515/jnum-2012-0015 . hal-00835298

HAL Id: hal-00835298

https://hal.science/hal-00835298v1

Submitted on 18 Jun 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

J. Numer. Math., Vol. 0, No. 0, pp. 1–16 (2013)
DOI 10.1515/ JNUM.2013.000
© de Gruyter 2013 *Prepared using* jnm.sty *[Version: 20.02.2007 v1.3]*

High performance domain decomposition methods on massively parallel architectures with `FreeFem++`

Pierre Jolivet[*,§] Victorita Dolean,[¶] Frédéric Hecht,[*]
Frédéric Nataf,[*] Christophe Prud'homme,[§] and Nicole Spillane[*]

Received XX, 2012 *Communicated by XX XX*
Received in revised form XX, 2012

Abstract — In this document, we present a parallel implementation in `FreeFem++` of *scalable* two-level domain decomposition methods. Numerical studies with highly heterogeneous problems are then performed on large clusters in order to assert the performance of our code.

Keywords: domain decomposition methods, high performance computing

1. Introduction

As the performance of modern computers keeps increasing, it is important to develop and implement efficient numerical software to take full advantage of their power. The first yet difficult problem is the parallelization of algorithms. In this paper, we will consider domain decomposition methods [12,14,17], which constitute one of the dominant paradigms in contemporary large-scale partial differential equation simulation. The second challenge is the actual implementation of these methods. Here, the `C++` *domain specific language* (DSL) `FreeFem++` is used to solve partial differential equations (PDE) through generalized Galerkin methods. Other alternatives include `deal.II` [1], `GetFem++`, or the *domain specific embedded language* (DSEL) `Feel++` [11]...

In sections 2 and 3, a model problem is introduced, as well as the basics of domain decomposition methods. Details on the `FreeFem++` implementation of these methods are given in section 4 and some numerical results are gathered in section 5.

[*]Laboratoire Jacques-Louis Lions, CNRS UMR 7598, Université Pierre et Marie Curie, 75005 Paris, France

[§]Laboratoire Jean Kuntzmann, CNRS UMR 5224, Université Joseph Fourier, 51 rue des Mathématiques, BP53, 38041 Grenoble Cedex 9, France

[¶]Laboratoire Jean-Alexandre Dieudonné, CNRS UMR 6621, Université Nice Sophia Antipolis, Parc Valrose, 06108 Nice Cedex 2, France

2. Model problem and discretization

We focus on the solution of the Darcy equation on a domain $\Omega \subset \mathbb{R}^d$ ($d = 2$ or 3),

$$\begin{aligned} \nabla \cdot (\kappa \nabla u) &= F \quad \text{in } \Omega \\ B(u) &= 0 \quad \text{on } \partial\Omega \end{aligned} \tag{2.1}$$

where u is the solution, F is the source term, B represents the boundary conditions and κ is the diffusivity. For the sake of simplicity, we will assume in this section that Dirichlet boundary conditions are used on $\partial\Omega$, meaning $B = \mathrm{Id}$. It is then possible to write the *variational formulation* of (2.1), which is: find $u \in H_0^1(\Omega)$ such that

$$\int_\Omega \kappa \nabla u \cdot \nabla v = \int_\Omega F\, v \qquad \text{for all } v \in H_0^1(\Omega) \tag{2.2}$$

A feature of `FreeFem++` is to penalize Dirichlet boundary conditions. Let $\rho \gg 1$ be a very large real number, the above variational formulation is approximated by: find $u \in H^1(\Omega)$ such that

$$\underbrace{\int_\Omega \kappa \nabla u \cdot \nabla v + \rho \int_{\partial\Omega} u\, v}_{a(u,v)} = \underbrace{\int_\Omega F\, v}_{l(v)} \qquad \text{for all } v \in H^1(\Omega)\,. \tag{2.3}$$

Let us consider a simplicial mesh $\mathcal{T} = \bigcup_{i=1}^{e} \{\mathcal{K}_i\}$ of Ω, i.e. a tessellation of Ω made of e triangles or tetrahedra, on which a finite set of n basis functions $\{\varphi_i\}_{i=1}^n$ spans the finite element space V so that all functions of $u \in H^1(\Omega)$ can be discretized as:

$$u \approx \sum_{i=1}^{n} u[][i]\varphi_i \tag{2.4}$$

where $u[]$ is the vector of degrees of freedom (*d.o.f.*) of the unknown approximation of u and $u[][i]$ denotes the value of the ith component of $u[]$ for all $1 \leqslant i \leqslant n$. The basis functions are frequently chosen as continuous linear functions ($\mathbb{P}_1$ finite elements) or continuous quadratic functions ($\mathbb{P}_2$ finite elements). The finite element method then leads to the following linear system:

$$Au[] = f. \tag{2.5}$$

where $(A_{ij})_{1 \leqslant i,j \leqslant n} = a(\varphi_i, \varphi_j)$, $(f_i)_{1 \leqslant i \leqslant n} = l(\varphi_i)$. The size of this linear system (2.5) increases with the number of elements e or with the complexity of the finite element space V. For large simulations, solving (2.5) by a direct method requires too much memory and can hardly ever be done in a timely fashion on

a single computer. By exploiting parallel algorithm, these kinds of limitations can be avoided on current multi-core or many-core computer architectures. This can be done naturally using domain decomposition methods, which are some sort of "divide & conquer" algorithms where the workload is spread among a group of processes so that each subproblem can be solved efficiently. This will be explained in detail in the next section, and implemented in section 4.

3. Domain decomposition methods

There are two main branches in domain decomposition: *non-overlapping* methods and *overlapping* methods. Only overlapping methods are considered in this document. Moreover, domain decomposition methods are seldom used as stand alone linear solvers. Most of the time, they are indeed used to precondition Krylov methods such as the conjugate gradient method (CG), the biconjugate gradient stabilized method (BiCGSTAB) [18] or the generalized minimal residual method (GMRES) [13] for unsymmetric problem. That is because fixed-point algorithms — which domain decomposition methods are part of, c.f. (3.3a) — are slower than Krylov methods to solve linear systems such as (2.5).

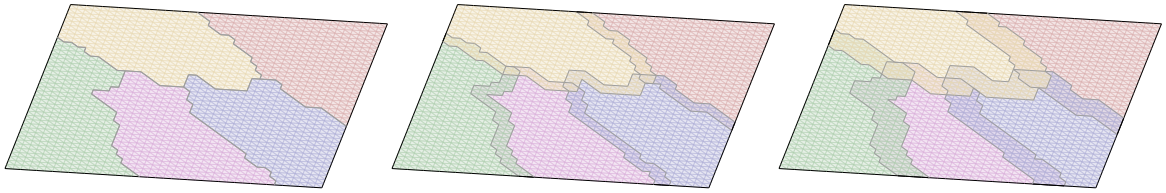
3.1. One-level methods

Let us assume that all the elements $\bigcup_{i=1}^{e} \{\mathcal{K}_i\}$ have been partitioned into N sets $\left\{\mathcal{T}_i^0\right\}_{i=1}^{N}$. We refer to the subdomains associated to $\mathcal{T}_i^0$ by Ω_i^0 so that:

$$\begin{aligned} \Omega &= \bigcup_{i=1}^{N} \Omega_i^0, \\ \forall i \in [\![1;e]\!], \exists! j \in [\![1;N]\!] \ &: \ \mathcal{K}_i \in \mathcal{T}_j^0. \end{aligned} \tag{3.1}$$

The one-overlap partition $\left\{\mathcal{T}_i^1\right\}_{i=1}^{N}$ of the elements is obtained by including all the adjacents elements of $\mathcal{T}_i^0$[I]into $\mathcal{T}_i^1$. Recursively, the partition with l levels of overlap $\left\{\mathcal{T}_i^l\right\}_{i=1}^{N}$ can be defined, and $\left\{\Omega_i^l\right\}_{i=1}^{N}$ accordingly, c.f. fig. 1 for a simple example. Note that the width of the overlap is of size "$2l$ elements".

[I] all the elements of $\mathcal{T}$ sharing at least one vertex, one node or one face (in $\mathbb{R}^3$) with at least one element of $\mathcal{T}_i^0$



(a) $l = 0$ **(b)** $l = 1$ **(c)** $l = 2$

Figure 1: Partition of $\Omega = [0;1]^2$ into $N = 5$ subdomains with different values for the overlap parameter

For clarity, the level of overlap for the partition of $\mathcal{T}$ is now considered to be a fixed *positive* integer l, and the superscript ^{l} will be omitted.
On each Ω_i, the finite element space V_i is built using the same basis functions as in section 2 which support intersects Ω_i. We say that the basis function φ_k is associated either to an interior *d.o.f.* of subdomain i if $supp(\varphi_k) \subset \overline{\Omega}_i$ or to an interface *d.o.f.* otherwise. The *local* number of freedom of each V_i will be refered to as n_i. In order to write algorithms that act on *global* vectors in V, restriction operators $\{R_i\}_{i=1}^N$ from global functions in $V^\star$ to local functions in $V_i^\star$ are needed. They are from a discrete point of view, rectangular matrices of size $n_i \times n$ filled with zeros and a single non-zero coefficient equal to one per line. Their dual operators, i.e. the extension operators from functions in V_i to V are simply $\left\{R_i^T\right\}_{i=1}^N$. Because of the overlap, some unknowns are associated to elements that lie within multiple subdomains so that $\sum_{i=1}^N n_i > n$. In fact, all elements lying in an overlapping region have their unknowns considered at least twice. These unknowns are being "duplicated" in each of the local finite element spaces and they are usually weakly coupled with a *partition of unity*, which is a set of diagonal matrices $\{D_i \in \mathbb{R}^{n_i} \times \mathbb{R}^{n_i}\}_{i=1}^N$ so that:

$$I = \sum_{i=1}^N R_i^T D_i R_i \tag{3.2}$$

where I is the identity matrix of $\mathbb{R}^n \times \mathbb{R}^n$ and $(D_i)_{kk} = 0$ for all φ_k associated to interface *d.o.f.* of Ω_i. This algebraic partition of unity plays an essential role in the formulation and implementation of domain decomposition methods. The Restricted Additive Schwarz method (RAS) is commonly used in domain decomposition [3]. It can be used to precondition a Krylov method, but also in a simpler way to solve (2.5) using a stationary iterative method that reads at step m:

$$u[]^{m+1} = u[]^m + M_{RAS}^{-1}(f[] - Au[]^m) \tag{3.3a}$$

where $u^m \in V$ and

$$M_{RAS}^{-1} := \sum_{i=1}^{N} R_i^T D_i \left(R_i A R_i^T \right)^{-1} R_i. \tag{3.3b}$$

In a parallel framework, it is usually more convenient to formulate algorithms as much as possible in terms of variables that are local to each finite element spaces $\{V_i\}_{i=1}^N$. Thus, (3.3a) has to be adapted considering it relies on global variables on V as above-formulated. Let us define the following rectangular (or square) matrices:

$$\forall (i,j) \in [\![1;N]\!]^2 \qquad \widetilde{A}_{ij} := R_i A R_j^T \in \mathbb{R}^{n_i} \times \mathbb{R}^{n_j}. \tag{3.4}$$

Due to the fact the diagonal entries of D_i are null for interface *d.o.f.*, the following equality holds for all $1 \leqslant i,j \leqslant N$:

$$\forall u_j \in V_j \qquad \widetilde{A}_{ij} D_j u_j[] = R_i A R_j^T D_j u_j[] = R_i R_j^T R_j A R_j^T D_j u_j[] = R_i R_j^T \widetilde{A}_{jj} D_j u_j[]\,. \tag{3.5}$$

In order to reformulate the RAS method in terms of local variables, the restriction operator R_i can be applied to the left of (3.3a):

$$\begin{aligned} R_i u[]^{m+1} &= R_i u[]^m + \sum_{j=1}^{N} R_i R_j^T D_j \widetilde{A}_{jj}^{-1} R_j \left(f[] - \sum_{k=1}^{N} A R_k^T D_k R_k u[]^m \right) \\ &= R_i u[]^m + \sum_{j=1}^{N} R_i R_j^T D_j \widetilde{A}_{jj}^{-1} \left(R_j f[] - \sum_{k=1}^{N} \widetilde{A}_{jk} D_k R_k u[]^m \right) \\ &= R_i u[]^m + \sum_{j=1}^{N} R_i R_j^T D_j \widetilde{A}_{jj}^{-1} \left(R_j f[] - \sum_{k=1}^{N} R_j R_k^T \widetilde{A}_{kk} D_k R_k u[]^m \right). \end{aligned} \tag{3.6}$$

The last line comes from identity (3.5) applied to $R_k u^m \in V_k$. At first sight, (3.6) might look painful to compute: two large sums of vector/matrix products, but using some simple considerations on the restriction and prolongation operators, the number of products in practice is to be greatly decreased. Indeed,

$$\forall (i,j) \in [\![1;N]\!]^2 \qquad (\Omega_i \cap \Omega_j)^\circ = \emptyset \implies \widetilde{A}_{ij} = 0 \ \text{ and } \ R_i R_j^T = 0\,.$$

In all that follows, we will refer to the *neighboring* subdomains using the following sets:

$$\forall i \in [\![1;N]\!] \qquad \begin{array}{l} \mathcal{O}_i := \{ j \in [\![1;N]\!] \ : \ j \neq i \ \text{ and } \ (\Omega_i \cap \Omega_j)^\circ \neq \emptyset \} \\ \overline{\mathcal{O}}_i := \mathcal{O}_i \cup \{i\} \end{array}$$

so that (3.6) can be reformulated as

$$u_i^{m+1} = u_i^m + \sum_{j \in \overline{\mathcal{O}}_i} R_i R_j^T D_j \widetilde{A}_{jj}^{-1} \left(R_j f[] - \sum_{k \in \overline{\mathcal{O}}_j} R_j R_k^T \widetilde{A}_{kk} D_k u_k^m \right) \tag{3.7}$$

where $u_i^m := R_i u[]^m$. This reformulation of the RAS algorithm has the advantage that the non-local computations reduce to the application of $R_i R_l^T$ for $l \in \mathcal{O}_i$. These operations corresponds to point-to-point communications: subdomain l sends to subdomain i data related to their intersection, c.f. fig. 3 page 9. This is the starting formula for our parallel implementation of this one-level method.

3.2. Two-level methods

As it is well-known, one-level methods are not *scalable* as they involve communications only between neighboring subdomains as is clear from (3.7). For large numbers of subdomains, convergence of these methods will suffer from long *plateaux*. This can be fixed using an additional preconditioner, that will couple all subdomains. This is explained in this section.
We will consider three types of preconditioners [9,6]. Let M^{-1} be the original one-level preconditioner, and Z be a deflation subspace matrix, i.e. a matrix associated to a subspace to be projected out of the residual of the preconditioned solver. Typically the number of columns of Z is of the order of the number of subdomains. Each column is associated to a subdomain. The vector space spanned by the columns of Z defines the coarse space. Three operators can now be defined:

1. the coarse operator $E = Z^T \widetilde{A} Z$
2. the correction matrix $\Xi = Z E^{-1} Z^T$
3. the deflation matrix $P = I - \widetilde{A}\Xi$

Note that since Z is a rectangular matrix belonging to $\mathbb{R}^n \times \mathbb{R}^p$ with $n \gg p$, it is much less consuming to solve a linear system involving $E \in \mathbb{R}^p \times \mathbb{R}^p$ than one involving the original coefficient matrix A. These three operators yield various preconditioners thoroughly studied in [16]. We just considered the followings in the numerical experiments:

$$\mathcal{P}_{BNN} = P^T M^{-1} P + \Xi \tag{3.8a}$$

$$\mathcal{P}_{A-DEF1} = M^{-1} P + \Xi \tag{3.8b}$$

$$\mathcal{P}_{A-DEF2} = P^T M^{-1} + \Xi \tag{3.8c}$$

4. Implementation

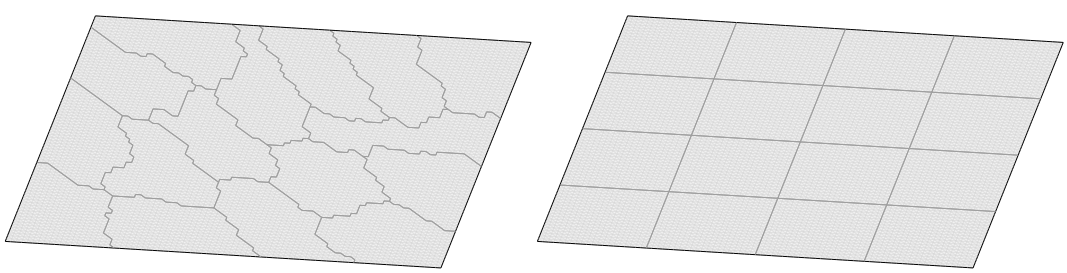
This section is a short overview of the principles behind our implementation in `FreeFem++` of one-level and then two-level Schwarz methods. `FreeFem++` is used for all the computations related to the finite element method (mesh generation, discretization, matrix assembly). Additionaly, `ARPACK` [8] is used as an eigenvalue solver, `BLAS` [2], `MKL`[II] or `ESSL`[III] are the possible linear algebra back ends chosen accordingly to the hardware architecture.

4.1. Pre-processing

4.1.1. Mesh partitioning. The obvious first step of all domain decomposition methods is to split the *global* mesh $\mathcal{T}$ as done in fig. 2: that leads to a natural decomposition of the original problem into smaller subproblems. From the point of view of the finite elements framework, a partitioning is a piecewise (with respect to $\mathcal{T}$) constant discontinuous $\mathbb{P}_0$ function p, meaning that each element of $\mathcal{T}$ is given a constant value $i \in [\![1; N]\!]$ that will correspond to the subdomain Ω_i^0 to which it belongs, formally: $\Omega_i^0 = p^{-1}(i)$.

User-defined partitioning. If the shape of Ω is regular with some symmetry properties (e.g. rectangles in $\mathbb{R}^2$ — cuboids in $\mathbb{R}^3$), an analytical partitioning of Ω can be defined.

Algorithmic graph partitioning. When Ω defines a complex geometry, it is best to use graph partitioners such as `Metis` [5] or `SCOTCH` [4] that will lead to more "balanced" decompositions (less communication, similar workload and such). These partitioners can be directly called from within the `FreeFem++` *DSL*.



(a) With a graph partitioner **(b)** With a user-defined partitioning

Figure 2: Examples of 16-way partitioning of $\Omega = [0; 1]^2$

[II] Intel® Math Kernel Library
[III] IBM Engineering and Scientific Subroutine Library for AIX/Linux on POWER

4.1.2. Local mesh construction. Now that the non-overlapping decomposition $\left\{\Omega_i^0\right\}_{i=1}^N$ is known, all $\{\Omega_i^0\}_{i=1}^N$ can be independently refined by splitting each triangle or each tetrahedron into s smaller elements. Very fine meshes can thus be generated in parallel while starting from a coarse global mesh. The next step is to build the overlapping decomposition $\left\{\Omega_i^l\right\}_{i=1}^N$ where the level of overlap l is a runtime constant. In the end, the characteristic width of the overlap is of size "$2l$ fine elements".

4.1.3. Discrete partition of unity. The approach to build the partition of unity is the same as in most domain decomposition methods, see for example [7]. Let $\widetilde{\chi}_i$ be a continuous piecewise linear function of Ω_i^l defined as such:

$$\widetilde{\chi}_i = \begin{cases} 1 & \text{on all nodes of } \Omega_i^0 \\ 1 - \dfrac{m}{l} & \text{on all nodes of } \Omega_i^m \setminus \overline{\Omega}_i^{m-1} \ \forall m \in [\![1; l]\!] \end{cases} \tag{4.1}$$

The local partition of unity is defined as:

$$\chi_i = \frac{\widetilde{\chi}_i}{\displaystyle\sum_{j \in \overline{\mathcal{O}}_i} \widetilde{\chi}_j|_{\Omega_i^l \cap \Omega_j^l}} \tag{4.2}$$

so that the support of the non negative function χ_i is Ω_i^l and

$$\sum_{i=1}^N \chi_i = 1. \tag{4.3}$$

The operator of multiplication by χ_i corresponds, at the discrete level, to the diagonal matrix D_i so that (3.2) holds. Note that for all $k \in \mathcal{O}_i$, the restriction of χ_k to $\Omega_i^l \cap supp(\chi_k)$ is stored locally.

4.2. Point-to-point transfers

The `MPI` process instantiation is achieved using `mpirun` or `bgrun`: each process runs a copy of `FreeFem++`, and each copy will initialize variables and functions locally. As the implementation is fully parallel, it is useless (and even "impossible") to consider for example the *global* matrix A or the *global* vector u in (2.5). Formalism of section 3.1 is used: only *local* matrices $\left\{\widetilde{A}_{ii}\right\}_{i=1}^N$ and *local* vectors $\{R_i u\}_{i=1}^N$ are considered, with respect to the decomposition of Ω. In (3.7), for a given subdomain index i, and for all neighbors (including subdomain i) $k \in \overline{\mathcal{O}}_i$, the *prolongation/restriction* operators $R_i R_k^T$ are needed. When $k = i$, this lead to the identity matrix of V_i, but then again,

the other terms for $k \neq i$ cannot be simply considered as products of two matrices, as for each of those matrices, one of the dimension is equal to n, the number of degrees of freedom of the *global* problem (2.5). However, when a vector v_k of Ω_k^l is considered, computing $R_i R_k^T v_k$ is rigorously equivalent to computing $R_{i\leftrightarrow k} R_{k\leftrightarrow i}^T v_k$ where $R_{i\leftrightarrow k}$ is defined as the linear interpolator from the finite element space defined on the mesh of the intersection $\Omega_i^l \cap \Omega_k^l$ to V_i. In other words, it amounts to transferring $v_k|_{\Omega_i^l \cap \Omega_k^l}$ from Ω_k^l to Ω_i^l. The goal of algorithm 1 is to construct for a given i, each $R_{i\leftrightarrow k}$ for $k \in \mathcal{O}_i$. The first step is to mesh for all $k \in \mathcal{O}_i$ the intersection between the support of the partition of unity χ_k and Ω_i^l and then define a finite element space on it. Eventually, $R_{i\leftrightarrow k}$ is nothing else than the linear interpolator from this finite element space to V_i.

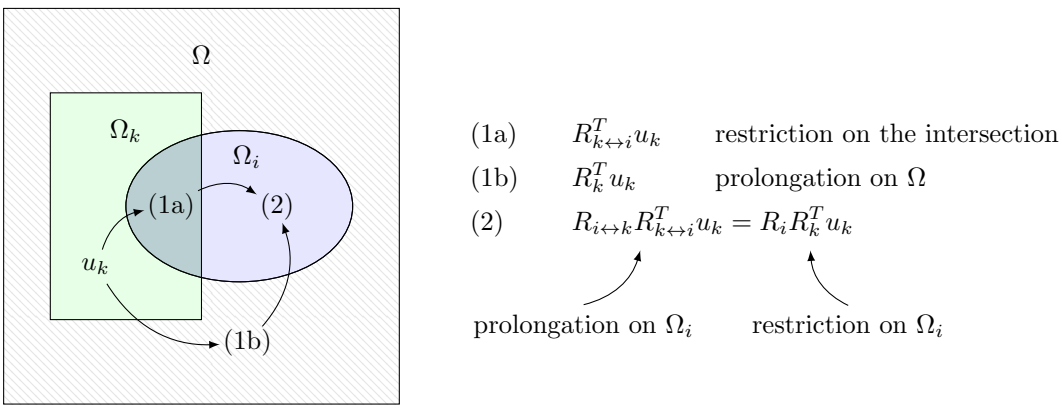


Figure 3: Two ways to transfer data to a neighboring subdomain using linear interpolators

Algorithm 1 `FreeFem++` construction of the prolongation/restriction operators on Ω_i

```
procedure RESTRICTION
    for k ∈ O_i do
        mesh T_intersection ← trunc(T_i^l, χ_k|_{Ω_i^l ∩ Ω_k^l} > 0)
        fespace V_intersection(T_intersection, P_b)          ▷ b = 0, 1, 2, ...
        matrix R_{i↔k} ← interpolate(V_intersection → V_i)
```

4.3. Coarse operator construction

This particular coarse space was introduced in [10] and the broad strokes of its theoretical construction are given in this section. Let us consider at the continuous level, for a given subdomain index $i \in [\![1; N]\!]$, the *Dirichlet to*

Neumann operator (a.k.a. *Poincaré-Steklov* operator) DtN_i defined on the interface Γ_i:

$$\mathrm{DtN}_i(\Psi_i) = \kappa \left.\frac{\partial v}{\partial \mathbf{n}_i}\right|_{\Gamma_i} \tag{4.4}$$

where $\mathbf{n}_i$ is the outward unit normal to Ψ_i and v satisfies the homogeneous boundary value problem (2.1) with Ψ_i as Dirichlet boundary conditions on Γ_i. The construction of the deflation subspace is then based on local low frequency modes of DtN_i: one looks for the eigenpairs (Ψ_i, λ_i) verifying

$$\mathrm{DtN}_i(\Psi_i) = \lambda_i \kappa \Psi_i \implies \widetilde{B}_{ii}\Lambda_i = \lambda_i V_{ii}\Lambda_i \tag{4.5}$$

where Λ_i is the harmonic extension of Ψ_i on Ω_i, $\widetilde{B}_{ii}$ is the coefficient matrix of the variational form: $H^1(\Omega_i) \times H^1(\Omega_i) \to \mathbb{R}$

$$a_i(u,v) = \int_{\Omega_i} \kappa \nabla u \cdot \nabla v \tag{4.6}$$

and V_{ii} is the coefficient matrix of the variational form: $H^1(\Omega_i) \times H^1(\Omega_i) \to \mathbb{R}$

$$b_i(u,v) = \int_{\Gamma_i} \kappa\, uv \tag{4.7}$$

$\widetilde{B}_{ii}$ cannot be inferred from $\widetilde{A}_{ii}$ since it refers to the matrix prior to assembly with the neighboring subdomains. However both matrices $\widetilde{B}_{ii}$ and V_{ii} can be easily assembled as implemented in algorithm 2.

Algorithm 2 `FreeFem++` construction of the discrete eigenvalue problem

1: **procedure** ASSEMBLY
2: $\quad$ varf $\mathbb{W}(u,v) \leftarrow \int_{\Omega_i} \kappa \nabla u \cdot \nabla v + \mathsf{on}(\partial\Omega \cap \partial\Omega_i, u = 0)$
3: $\quad$ varf $\mathbb{U}(u,v) \leftarrow \int_{\Gamma_i} \kappa\, uv$
4: $\quad$ matrix $\widetilde{B}_{ii} \leftarrow \mathbb{W}(V_i, V_i)$
5: $\quad$ matrix $V_{ii} \leftarrow \mathbb{U}(V_i, V_i)$

Once the two matrices are stored in memory, the `ARPACK` library is called and it solves the eigenvalue problem using the Implicitly Restarted Lanczos Method (IRLM) as in this case the matrix $\widetilde{B}_{ii}$ is symmetric. As specified earlier in this paragraph, only low eigenvalues are considered in the construction of the deflation subspace: a threshold criterion is used to select the ν_i right eigenvalues. The sets of eigenpairs are then locally indexed by increasing magnitude of the eigenvalues:

$$\left\{(\Lambda_{i_1}, \lambda_{i_1}), (\Lambda_{i_2}, \lambda_{i_2}), \ldots, \left(\Lambda_{i_{\nu_i}}, \lambda_{i_{\nu_i}}\right)\right\} \text{ such that } 0 \leqslant \lambda_{i_1} \leqslant \lambda_{i_2} \leqslant \cdots \leqslant \lambda_{i_{\nu_i}}.$$

The deflation subspace matrix Z is then defined as:

$$Z = \begin{bmatrix} W_1 & 0 & \cdots & 0 \\ 0 & W_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W_N \end{bmatrix} \in \mathbb{R}^{\sum_{i=1}^N n_i} \times \mathbb{R}^{\sum_{i=1}^N \nu_i} \quad (4.8)$$

where

$$\left\{ W_i = \left[D_i \Lambda_{i_1} \;\; D_i \Lambda_{i_2} \;\; \cdots \;\; D_i \Lambda_{i_{\nu_i}} \right] \in \mathbb{R}^{n_i} \times \mathbb{R}^{\nu_i} \right\}_{i=1}^N \quad (4.9)$$

Notice that with the definition of Z, $E := Z^T \widetilde{A} Z$ is a matrix of size $\sum_{k=1}^N \nu_k \times \sum_{k=1}^N \nu_k$.

We define as $\mathcal{R}$ the function from $\left[\!\left[1; \sum_{k=1}^N \nu_k \right]\!\right] \to [\![1; N]\!]$ such that:

$$\mathcal{R} : j \mapsto \max \left\{ i : \sum_{k=1}^{i} \nu_k < j \right\} \quad (4.10a)$$

and φ the function from $\left[\!\left[1; \sum_{k=1}^N \nu_k \right]\!\right] \to [\![1; \max_{1 \leqslant k \leqslant N}(\nu_k)]\!]$ such that:

$$\varphi : j \mapsto j - \sum_{k=1}^{\mathcal{R}(j)} \nu_k \,. \quad (4.10b)$$

$\mathcal{R}$ is the function that maps each column (or row) index of E to its corresponding subdomain index, and φ the local number of an eigenvector within a subdomain. If all ν_i equals ν (e.g. when uniform decompositions are considered), then those two functions can be seen as:

$$\mathcal{R}(i) = \left\lfloor \frac{i}{\nu} \right\rfloor \quad (4.11a)$$

$$\varphi(i) = (i \bmod \nu) + 1 \,. \quad (4.11b)$$

Then, $\widetilde{A} Z$ is a block matrix of size $\sum_{i=1}^N n_i \times \sum_{i=1}^N \nu_i$ whose block (i, j) of size $n_i \times 1$ equals:

$$(\widetilde{A} Z)_{ij} = \widetilde{A}_{i\mathcal{R}(j)} D_{\mathcal{R}(j)} \Lambda_{\mathcal{R}(j)_{\varphi(j)}} \in \mathbb{R}^{n_i} \qquad \forall (i, j) \in [\![1; N]\!] \times \left[\!\left[1; \sum_{k=1}^N \nu_k \right]\!\right] \quad (4.12)$$

and,

$$\begin{aligned} E_{ij} &= \left(Z^T(\widetilde{A}Z)\right)_{ij} \\ &= \Lambda^T_{\mathcal{R}(i)_{\varphi(i)}} D_{\mathcal{R}(i)} (\widetilde{A}Z)_{\mathcal{R}(i)j} \in \mathbb{R} \qquad \forall (i,j) \in \left[\!\left[1; \sum_{k=1}^{N} \nu_k\right]\!\right]^2 \\ &= \Lambda^T_{\mathcal{R}(i)_{\varphi(i)}} D_{\mathcal{R}(i)} \widetilde{A}_{\mathcal{R}(i)\mathcal{R}(j)} D_{\mathcal{R}(j)} \Lambda_{\mathcal{R}(j)_{\varphi(j)}} \end{aligned} \tag{4.13}$$

In practice, the block sparse matrix E needs to be assembled on a single root process. By taking a closer look at its structure, it can be observed that it is made of chunk of rows that depends on only one *local* matrix, using the same transformation as for the RAS method in (3.6), i.e. the identity (3.5) applied to $D_{\mathcal{R}(j)}\Lambda_{\mathcal{R}(j)_{\varphi(j)}}$:

$$\begin{aligned} \widetilde{A}_{\mathcal{R}(i)\mathcal{R}(j)} D_{\mathcal{R}(j)} \Lambda_{\mathcal{R}(j)_{\varphi(j)}} &= R_{\mathcal{R}(i)} R^T_{\mathcal{R}(j)} \widetilde{A}_{\mathcal{R}(j)\mathcal{R}(j)} D_{\mathcal{R}(j)} \Lambda_{\mathcal{R}(j)_{\varphi(j)}} \\ &= R_{\mathcal{R}(i)\leftrightarrow\mathcal{R}(j)} R^T_{\mathcal{R}(j)\leftrightarrow\mathcal{R}(i)} \widetilde{A}_{\mathcal{R}(j)\mathcal{R}(j)} D_{\mathcal{R}(j)} \Lambda_{\mathcal{R}(j)_{\varphi(j)}} \end{aligned} \tag{4.14}$$

We use this observation by computing each of these chunks on the appropriate process. Then, each chunk of rows is sent to the root process in charge of assembling E.

5. Numerical experiments

Two dimensional test case. The model problem (2.1) is solved on $\Omega = [0;1]^2$ with mixed Dirichlet and homogenous Neumann boundary conditions using $\mathbb{P}_2$ finite elements. The diffusivity is a highly heterogeneous function of $\Omega \to \mathbb{R}$, c.f. fig. 4, defined as:

$$\kappa(x,y) = \begin{cases} 10^5(\lfloor 9x \rfloor + 1) & \text{if } \lfloor 9x \rfloor \equiv 0 \bmod 2 \text{ and } \lfloor 9y \rfloor \equiv 0 \bmod 2 \\ 1 & \text{otherwise} \end{cases}$$

Three dimensional test case. The same model problem (2.1) as for the two dimensional test case is solved once again with mixed Dirichlet and homogenous Neumann boundary conditions on $\Omega = [0;1]^3$ using $\mathbb{P}_2$ finite elements. The diffusivity is defined as:

$$\kappa(x,y,z) = \begin{cases} 10^5(\lfloor 9x \rfloor + 1)\kappa_\uparrow(z) & \text{if } \lfloor 9x \rfloor \equiv 0 \bmod 2 \text{ and } \lfloor 9y \rfloor \equiv 0 \bmod 2 \\ 1 & \text{otherwise} \end{cases}$$

where

$$\kappa_\uparrow(z) = \begin{cases} \lfloor 9z \rfloor & \text{if } \lfloor 9z \rfloor \not\equiv 0 \bmod 3 \\ 1 & \text{otherwise} \end{cases}$$

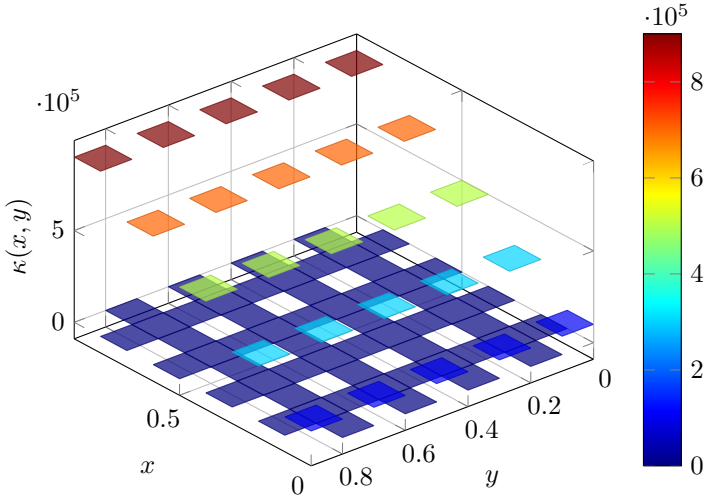


Figure 4: Two dimensional diffusivity κ in our test cases

It is important to note here that using traditional solvers such as Krylov methods (even with simple preconditioners) would yield really poor results for those two test cases because of the heterogeneous inclusions, which is why designing preconditioners robust in heterogeneities is of paramount importance for real world engineering problems (such as black-oil reservoir simulation).

5.1. Scalability tests

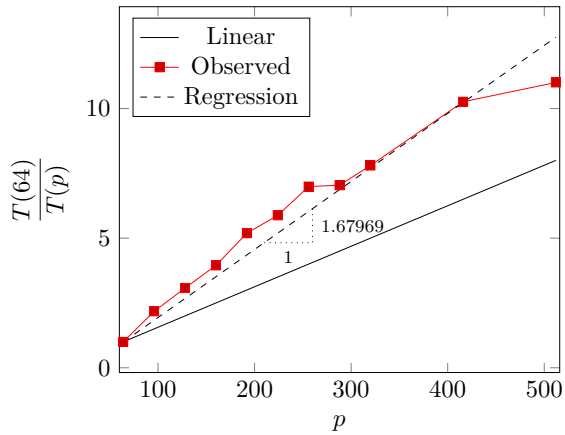
All the elapsed time are obtained using the routine `MPI_Wtime()`. Only the GMRES is timed, meaning that the construction of the meshes $\left\{\mathcal{T}_i^l\right\}_{i=1}^N$, the partitioning of unity $\left\{D_i^l\right\}_{i=1}^N$ and the construction (transfers plus factorization) of the coarse operator E are not considered. The stopping criterion is chosen so that the relative residual of the GMRES is inferior to a certain tolerance ε at convergence. The first way to assess the performance of the implementation of our parallel solver was done by checking its speedup when increasing the number of processes. These particular tests were performed on `titane`, a 40960-core computer hosted at CEA,[IV] Bruyères-le-Châtel.

A *super linear* speedup is observed in two dimensions, c.f. fig. 5 page 14, as well as in three dimensions, c.f. fig. 6 page 15. Some tests were carried out on `babel`, a 121912-core computer hosted at IDRIS,[V] Orsay, to prove the robustness of the two-level preconditioner. Even when greatly increasing

[IV] Commissariat à l'Energie Atomique et aux Energies Alternatives
[V] Institut du Développement et des Ressources en Informatique Scientifique

p	T	$\sum_{i=1}^{N} \nu_i$	$\sum_{i=1}^{N} n_i$
64	65.7 s	1,890	$36.5 \cdot 10^6$
128	21.4 s	3,810	$36.7 \cdot 10^6$
192	12.7 s	5,730	$36.9 \cdot 10^6$
256	9.4 s	7,650	$37.1 \cdot 10^6$
320	8.4 s	9,570	$37.2 \cdot 10^6$
512	6.0 s	15,330	$37.5 \cdot 10^6$

(a) The 3rd column represents the size of the coarse operator E, and the 4th column is the total number of unknowns. Notice that it keeps on slightly increasing as the number of duplicated unknowns increase when more subdomains are built.



(b) Speedup normalized to 64 subdomains with a linear regression of $T(p)$

Figure 5:
Strong scaling observed when solving a two dimensional test case on a fixed size global problem using $\mathbb{P}_2$ finite elements and a tolerance $\varepsilon = 10^{-9}$

the number of subdomains, the Krylov method still converges quite rapidly in terms of number of iterations, i.e. in less than 25 iterations for decomposition made of up to 4096 subdomains.

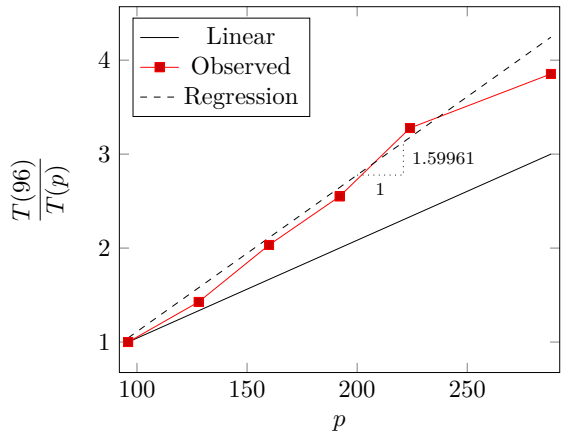
6. Conclusion

The results displayed in this document assess the efficiency of our framework for parallel domain decomposition methods for solving a simple boundary value problem — the Darcy equation — in two and three dimensions with various finite elements. Our goal is now to tackle more challenging problems, such as the equations of linear elasticity whose theory has been covered in [15]. As the size of the coarse space keeps on growing, we are also currently looking into adding another coarse problem, that could potentially lead to "multi-level" preconditioners, similar to multigrid preconditioner, [19]. Readers willing to try these algorithms should go to `http://www.freefem.org/ff++` and download the open source software.

Acknowledgments. This work has been granted access to the HPC resources of CCRT and IDRIS under the allocation 2011-6654 made by GENCI through ANR project PETALh (ANR-10-COSI-013). The assistance of Philippe Wautelet from IDRIS played an integral role in the completion of the scalability tests.

p	T	$\sum_{i=1}^{N} \nu_i$	$\sum_{i=1}^{N} n_i$
96	26.5 s	1,920	$6.9 \cdot 10^6$
128	18.6 s	2,560	$7.1 \cdot 10^6$
160	13.0 s	3,200	$7.4 \cdot 10^6$
192	10.4 s	3,840	$7.6 \cdot 10^6$
224	8.1 s	4,480	$7.7 \cdot 10^6$
288	6.9 s	5,760	$8.1 \cdot 10^6$

(a) The 3rd column represents the size of the coarse operator E, and the 4th column is the total number of unknowns. Notice that it keeps on slightly increasing as the number of duplicated unknowns increase when more subdomains are built.



(b) Speedup normalized to 96 subdomains with a linear regression of $T(p)$

Figure 6:
Strong scaling observed when solving a three dimensional test case on a fixed size global problem using $\mathbb{P}_2$ finite elements and a tolerance $\varepsilon = 10^{-12}$

References

1. W. Bangerth, R. Hartmann, and G. Kanschat, *deal.II — a general-purpose object-oriented finite element library*, ACM Transactions on Mathematical Software **33** (2007), no. 4, 24–27.
2. L.S. Blackford, A. Petitet, R. Pozo, K. Remington, R.C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, et al., *An updated set of basic linear algebra subprograms (BLAS)*, ACM Transactions on Mathematical Software **28** (2002), no. 2, 135–151.
3. X.C. Cai and M. Sarkis, *Restricted additive Schwarz preconditioner for general sparse linear systems*, SIAM Journal on Scientific Computing **21** (1999), no. 2, 792–797.
4. C. Chevalier and F. Pellegrini, *PT-Scotch: a tool for efficient parallel graph ordering*, Parallel Computing **34** (2008), no. 6, 318–331.
5. G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing **20** (1998), no. 1, 359–392.
6. J. Frank and C. Vuik, *On the construction of deflation-based preconditioners*, SIAM Journal on Scientific Computing **23** (2002), no. 2, 442–462.
7. J.H. Kimn and M. Sarkis, *Restricted overlapping balancing domain decomposition methods and restricted coarse problems for the Helmholtz problem*, Computer Methods in Applied Mechanics and Engineering **196** (2007), no. 8, 1507–1514.
8. R.B. Lehoucq, D.C. Sorensen, and C. Yang, *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*, vol. 6, Society for Industrial and Applied Mathematics, 1998.

9. J. Mandel, *Balancing domain decomposition*, Communications in Numerical Methods in Engineering **9** (1993), no. 3, 233–241.

10. F. Nataf, H. Xiang, V. Dolean, and N. Spillane, *A coarse space construction based on local Dirichlet to Neumann maps*, SIAM Journal on Scientific Computing **33** (2011), no. 4, 1623–1642.

11. C. Prud'homme, V. Chabannes, V. Doyeux, M. Ismail, A. Samake, and G. Pena, *Feel++: A computational framework for Galerkin methods and advanced numerical methods*, ESAIM: Proceedings, 2012, preprint http://hal.archives-ouvertes.fr/hal-00662868.

12. A. Quarteroni and A. Valli, *Domain decomposition methods for partial differential equations*, vol. 10, Clarendon Press, 1999.

13. Y. Saad and M.H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing **7** (1986), no. 3, 856–869.

14. B.F. Smith, P.E. Bjørstad, and W. Gropp, *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*, Cambridge University Press, 2004.

15. N. Spillane, V. Dolean, P. Hauret, F. Nataf, C. Pechstein, and R. Scheichl, *A robust two-level domain decomposition preconditioner for systems of PDEs*, Comptes Rendus Mathematique **349** (2011), no. 23, 1255–1259.

16. J.M. Tang, R. Nabben, C. Vuik, and Y.A. Erlangga, *Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods*, Journal of Scientific Computing **39** (2009), no. 3, 340–370.

17. A. Toselli and O.B. Widlund, *Domain decomposition methods — algorithms and theory*, Series in Computational Mathematics, vol. 34, Springer, 2005.

18. H.A. Van der Vorst, *Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing **13** (1992), 631–644.

19. P.S. Vassilevski, *Multilevel block factorization preconditioners: matrix-based analysis and algorithms for solving finite element equations*, vol. 10, Springer, 2008.