Domain decomposition methods in Feel++

Abdoulaye Samaké[1], Vincent Chabannes[1], Christophe Picard[1], and Christophe Prud'homme[2]

1 Introduction

Libraries to solve problems arising from partial differential equations (PDEs) through generalized Galerkin methods are a common tool among mathematicians and engineers. However, most libraries end up specializing in a type of equation, e.g. Navier-Stokes or linear elasticity models, or a specific type of numerical method, e.g. finite elements. The increasing complexity of differential models and the implementation of state of the art robust numerical methods, demand from scientific computing platforms general and clear enough languages to express such problems and provide a wealth of solution algorithms available in a minimal amount of code but maximum mathematical control. There are many freely available libraries which offer the capabilities described previously to a certain extent. To name a few: the Freefem software family [6, 9], the Fenics project [10], Getdp [8] or Getfem++ [17], or libraries or frameworks such as deal.II (C++) [2], Sundance (C++) [11], Analysa (Scheme) [1].

The library we present in this paper, called FEEL++, *Finite Element Embedded Language in* C++, see [14, 15], provides also a clear and easy to use interface to solve complex PDE systems. It aims at bringing the scientific community a tool for the implementation of advanced numerical methods and high performance computing. Some recent applications of FEEL++ to multiphysics problems can be found in the literature, see *e.g.* [13, 7, 5].

FEEL++ relies on a so-called *domain specific embedded language* (DSEL) designed to closely match the Galerkin mathematical framework. In computer science, DS(E)Ls are used to partition complexity and in our case the DSEL splits low level mathematics and computer science on one side leaving the FEEL++ developer to enhance them and high level mathematics as well as physical applications to the other side which are left to the FEEL++ user. This enables using FEEL++ for teaching purposes, solving complex problems with multiple physics and scales or rapid prototyping of new methods, schemes or algorithms.

The DSEL on FEEL++ provides access to powerful, yet with a simple and seamless interface, tools such as interpolation or the clear translation of a wide range of variational formulations into the variational embedded language. Combined with this robust engine, lie also state of the art arbitrary order finite elements — including

[1]Laboratoire Jean Kuntzmann, Université Joseph Fourier Grenoble 1, BP53 38041 Grenoble Cedex 9, France, Tel.: +33476635497, Fax: +33476631263, e-mail: {abdoulaye.samake}{vincent.chabannes}{christophe.picard}@imag.fr ·[2] Université de Strasbourg / CNRS, IRMA / UMR 7501, Strasbourg, F-67000, France, e-mail: prudhomme@unistra.fr

handling high order geometrical approximations, — high order quadrature formulas and robust nodal configuration sets. The tools at the user's disposal grant the flexibility to implement numerical methods that cover a large combination of choices from meshes, function spaces or quadrature points using the same integrated language and control at each stage of the solution process the numerical approximations.

This paper presents our ongoing work on building a computational framework for domain decomposition methods in FEEL++ including overlapping and nonoverlapping Schwarz methods (conforming and non-conforming) and mortar method. The complete examples are available in FEEL++ sources. Note that examples using the three fields method are also available in FEEL++.

The framework main objectives consist in *(i)* reproducing and comparing easily several of methods in the literature *(ii)* developing a teaching and research programming environment *(iii)* providing the methods at the functional level or at the algebraic level. In this context we have also developed also two alternatives: one which lets the user control the MPI communications and one which hides completely the MPI communications.

2 Schwarz Methods

Let Ω be a domain of $\mathbb{R}^d$, $d = 1,2,3$, and $\partial\Omega$ its boundary. We look for u the solution of the problem:

$$Lu = f \quad \text{in} \quad \Omega, \quad u = g \quad \text{on} \quad \partial\Omega \tag{1}$$

where L is a linear partial differential operator, and f and g are given functions. Let $\Omega_i (i = 1,...,N,\ N \in \mathbb{N},\ N \geq 2)$ the subdomain partitions of Ω such that $\overline{\Omega} = \cup_{i=1}^{N} \overline{\Omega}_i$ and $\Gamma_{ij} = \partial\Omega_i \cap \overline{\Omega}_j$ the interface between neighboring subdomains Ω_i and Ω_j. We denote $\mathscr{V}_{\Omega_i}$ the set of neighbors subdomains of Ω_i. In the case of nonoverlapping subdomains $\Gamma_{ij} = \Gamma_{ji}$. We are interested in the overlapping and nonoverlapping alternating Schwarz methods[16, 19] as solver in the general nonmatching grids and arbitrary number of subdomains. The generic Schwarz additive algorithm is given by (2) where u_i^0 is known on Γ_{ij}, $j \in \mathscr{V}_{\Omega_i}$, $k \geq 1$ the Schwarz iteration index and C_i is a partial differential operator.

$$Lu_i^k = f \text{ in} \quad \Omega_i, \quad u_i^k = g \text{ on} \quad \partial\Omega_i \setminus \Gamma_{ij}, \quad C_i u_i^k = C_i u_j^{k-1} \text{on} \quad \Gamma_{ij} \tag{2}$$

The algorithm (2) extends easily to the multiplicative version of Schwarz methods and treats different types of artificial boundary conditions such as Dirichlet-Dirichlet (DD), Dirichlet-Neumann (DN), Neumann-Neumann (NN) and Robin-Robin (RR) (see [20, 16, 19]) according the choice of the operator C_i that is assumed linear in our case. The above algorithm can also adapt to relaxation techniques(see [16]) necessary for the convergence of some types of interface conditions such as DN and NN without overlap.

In the following subsections 2.1 and 2.2, we discuss two different approaches for Schwarz methods in FEEL++ namely with explicit communications and with seamless communications. In the first approach, we deal different types of Schwarz methods(Additive, Multiplicative, with(out) Relaxation) with different artificial boundary conditions(DD, DN, NN, RR) while having the ability to process (non-)conforming meshes as well as being able to control the size of the overlap between neighboring subdomains. In the second approach, we use the parallel data structures of FEEL++ and the algebraic domain decomposition framework provided by PETSC.

2.1 Explicit Communication Approach

The Schwarz methods are used as solvers and the communications are handled explicitly by the user. Implementation-wise we use PETSC sequentially even though the code is parallel using `mpi` communicators. It requires explicitly sending and receiving complex data structures such as mesh data structures and elements of functions space(traces). A sequential interpolation operator is also used to make the transfer between the grids (overlapping or not, conforming or not). In this case each subdomain creates locally its mesh and its function space, the matrices and vectors associated to the discretization process are completely local.

The variational formulation of the problem (2) in the simplest form ($L := -\Delta$) in the subdomain Ω_i at iteration number k using Nitsche's method (see [12]) in the case of weak Dirichlet-Dirichlet artificial boundary conditions ($C_i = C_j = Id,\ j \in \mathcal{V}_{\Omega_i}$) is given by: find $u_i^k \in H^1(\Omega_i)$ such that $a(u_i^k, v) = l(v)\ \forall v \in H^1(\Omega_i)$ where

$$a(u_i^k,v) := \int_{\Omega_i} \nabla u_i^k \cdot \nabla v + \int_{\partial\Omega_i} -\frac{\partial u_i^k}{\partial n} v - \frac{\partial v}{\partial n} u_i^k + \frac{\gamma}{h} u_i^k v \tag{3}$$

$$l(v) := \int_{\Omega_i} f v + \int_{\partial\Omega_i \setminus \Gamma_{ij}} \Big(-\frac{\partial v}{\partial n} + \frac{\gamma}{h} v \Big) g + \sum_{j \in \mathcal{V}_{\Omega_i}} \int_{\Gamma_{ij}} \Big(-\frac{\partial v}{\partial n} + \frac{\gamma}{h} v \Big) u_j^{k-1} \tag{4}$$

where γ is a penalization parameter and h the maximum mesh size.

Other variants of artificial boundary conditions such as Dirichlet-Neumann($C_i = Id,\ C_j = \partial/\partial n,\ j \in \mathcal{V}_{\Omega_i}$), Neumann-Neumann ($C_i = C_j = \partial/\partial n,\ j \in \mathcal{V}_{\Omega_i}$) and Robin-Robin($C_i = C_j = (\partial/\partial n) + Id,\ j \in \mathcal{V}_{\Omega_i}$) are also treated. In the above variational formulation, only the terms colored in red in (4) requires communications between neighboring subdomains for each Schwarz iteration and interpolation between the grids. Note that the assembly of the other terms of the variational formulation is done once and is purely local. We make use of `Boost.MPI` and `Boost.Serialization` to ease the transfer of FEEL++ complex data structures such as meshes and (elements of) function spaces.

Listing 1 Feel++ snippet code for parallel Schwarz algorithm

```
// Create local mesh and function space on subdomain number i
```

```
auto mesh = createGMSHMesh(_mesh=mesh_type, ...);
auto Xh = space_type::New(mesh);
std::vector<mpi::request> reqs; // vector of Boost.MPI requests
for(int j=0, j< Nneighbors, ++j){
  // Extract trace mesh on interface number j
  trace_mesh_send[j]=mesh->trace(markedfaces(mesh,j));
  // Exchange trace mesh with neighbor subdomain number j
  auto req1=comm.isend( j,i,trace_mesh_send[j] );
  auto req2=comm.irecv( j,j,trace_mesh_recv[j] );
  reqs.push_back(req1); reqs.push_back(req2);
 } mpi::wait_all(reqs.begin(), reqs.end());// wait all requests
for(int j=0, j< Nneighbors, ++j){
  // Create trace function space for interface number j
  TXh[j] = trace_space_type::New(trace_mesh_recv[j]);
  // Create interpolation operator from Xh to TXh[j]
  opI[j]=operatorInterpolation(Xh,TXh[j]); }
while(!convergence) { // Schwarz iterations
  reqs.clear();
  for(int j=0, j< Nneighbors, ++j){
    // Non conforming interpolation for interface number j
    opI[j]->apply(solution,trace_solution_send[j]);
    // Exchange trace solution with neighbor subdomain number j
    auto req1=comm.isend( j,i,trace_solution_send[j] );
    auto req2=comm.irecv( j,j,trace_solution_recv[j] );
    reqs.push_back(req1); reqs.push_back(req2);
  } mpi::wait_all(reqs.begin(), reqs.end());// wait all requests
    // Update right hand side for each schwarz iteration
  for(int j=0, j< Nneighbors, ++j){
    form1( _test=Xh,_vector=F ) +=
      integrate(elements(trace_mesh_send[j]),
        -grad(v)*N()*idv(trace_solution_recv[j])
        +penaldir*idv(trace_solution_recv[j])*id(v)/hFace()); }
  solve(); }
```

To illustrate our implementation of the Schwarz method, we consider the problem (1) over a partition over the domain $\Omega = [0,1]^2$ into 128 overlapping subdomains (16×8) with non matching meshes. The boundary condition and the source write $g(x,y) = 0$ and $f(x,y) = \exp(-10xy)\cos(\frac{3\pi}{8})\sin(xy)$.

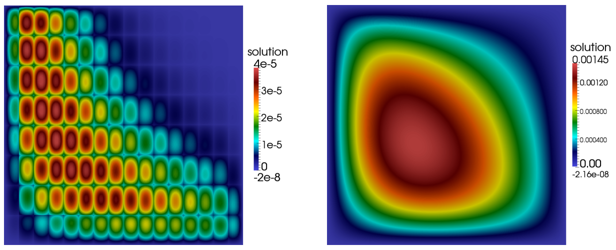

Fig. 1 Numerical solutions obtained by Schwarz parallel additive algorithm in 2D on 128 processors(1 subdomain/processor): First schwarz iteration(Left) and solution at convergence(Right)

The numerical solutions in Figure 1 are obtained using $\mathbb{P}_2$ Lagrange elements. The precision of the numerical solver is fixed to $1e-7$. The mesh size is 0.01 in each subdomain and the size of the overlap is 0.02 but we don't ensure that the grids are conforming. The total number of degree of freedom is 153600. The number of Schwarz iterations to convergence is 130 and the relative L^2 error $\|u-u_h\| = 1.164901e-06$. The listing 1 illustrates some aspects of the Schwarz algorithm using the `Feel++` language.

2.2 *Seamless Communication Approach*

Here we consider the domain decomposition methods with seamless communications in FEEL++. We provide a parallel data framework: we start with automatic mesh partitioning using GMSH(Chaco/Metis) — adding information about ghosts cells with communication between neighbor partition; — then FEEL++ data structures are parallel such as meshes, (elements of) function spaces — create a parallel degrees of freedom table with local and global views; — and finally we use the PETSC Krylov subspace solvers(KSP) coupled with PETSC preconditioners such as Block-Jacobi, ASM, GASM. The last preconditioner is an additive variant of the Schwarz alternating method for the case of many subregion, see [19]. For each sub-preconditioners(in the subdomains), PETSC allows to choose in the wide range of sequential preconditioners such, ilu, jacobi, ml.

To illustrate this, we perform a strong scalability test with a Laplace problem in 3D using P3 Lagrange elements (about 8 Millions degrees of freedom). The listing 2 corresponds to the code that allowed us to realize this test. The speedup displayed in table 1 corresponds to the assembly plus the solve times. We can see that the scaling is good except for the last configuration where the local problems is too small.

Listing 2 Laplacian Solver using continuous approximation spaces and PETSc in parallel

```
/* Create parallel function space and some associated elements */
auto Xh = space_type::New( _mesh=mesh );
/* Create the parallel matrix and vector of linear system */
auto A = backend()->newMatrix(_test=Xh, _trial=Xh);
auto F = backend()->newVector(Xh);
/* Parallel assembly of the right hand side */
form1( _test=Xh, _vector=F )=
    integrate( _range=elements( mesh ), _expr=f*id( v ) )
/* Parallel assembly of the global matrix */
form2( _test=Xh, _trial=Xh, _matrix=A ) =
    integrate( _range=elements( mesh ),
               _expr=gradt(u)*trans(grad(v)) );
/* Apply Dirichlet boundary conditions strongly */
form2( _test=Xh, _trial=Xh, _matrix=A ) +=
  on( _range=boundaryfaces(mesh),
      _element=u,_rhs=F, _expr=g );
/* solve system using PETSc parallel solvers/preconditioners */
backend()->solve( _matrix=A, _solution=u, _rhs=F );
```

Table 1 Strong scalability test

Number of Cores	Absolute Times	Speedup
1024	41.2	1
2048	18.2	2.26
4096	10	4.12
8192	7	5.88

3 Mortar Method

Consider the problem (1) where $L := -\Delta$ and homogeneous Dirichlet boundary conditions. We assume that Ω is partitioned into two nonoverlapping subdomains and it is a d-dimensional domain ($d = 2,3$), with a Lipschitz boundary $\partial\Omega$. We also assume that f belongs to $L^2(\Omega)$. The main idea of this method is to enforce the weak continuity between the solutions on each subdomain. This is achieved by introducing a Lagrange multiplier corresponding to this connection constraint [3]. Let us denote by V_{ih} the finite element approximation space on Ω_i, of basis $(\psi_{i,j})_{j=1,\cdots N_i}, i = 1,2$, and by W_h that of $\Gamma := \partial\Omega_1 \cap \partial\Omega_2$, of basis $(\phi_k)_{k=1,\cdots K}$ and $\Lambda := \left\{ \eta \in H^{1/2}(\Gamma) \mid \eta = v_{|\Gamma} \text{ for a suitable } v \in H^1(\Omega) \right\}$ the trace space. The mortar formulation is given by: for $i = 1,2$ find $u_i \in V_i := H^1(\Omega_i)$, $\lambda \in \Lambda$ such that

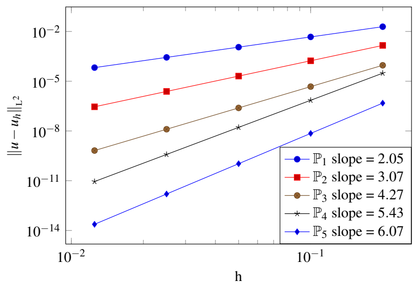

Fig. 2 Convergence results for Mortar Element Method in 2D with L^2 Errors curves

$$\begin{cases} \displaystyle\int_{\Omega_i} \nabla u_i \cdot \nabla v_i \pm \int_\Gamma \lambda v_i = \int_{\Omega_i} f v_i & \forall\, v_i \in H^1(\Omega_i) \\ \displaystyle\int_\Gamma \lambda (u_1 - u_2) = 0 & \forall\, \lambda \in \Lambda \end{cases} \tag{5}$$

Listing 3 Jump terms in the global matrix for mortar formulation

```
// product function spaces X_{h_1} × X_{h_2} × Λ_h for Ω_1 × Ω_2 × Γ
typedef meshes<mesh1_type,mesh2_type,trace_mesh_type> mesh_type;
typedef bases<Lagrange<2>,
              Lagrange<3>,
              Lagrange<2,Mortar> > basis_type;
typedef FunctionSpace< mesh_type, basis_type > space_type;
auto mesh = meshes( mesh1, mesh2, trace_mesh );
auto Xh = space_type::New( _mesh=mesh );
auto u = Xh->element();
auto u1 = u.element<0>();
auto u2 = u.element<1>();
auto mu = u.element<2>();
// assembly of jump terms in the global  matrix A
auto A = M_backend->newMatrix( _trial=Xh, _test=Xh );
form2( _trial=Xh, _test=Xh, _matrix=A ) +=
  integrate(elements(Xh->mesh<3>()),
```

The convergence results in figure 2 are obtained with the solution of the problem (1) using mortar formulation (5) by splitting the initial domain $\Omega = [0,1]\times[0,1]$ into two nonoverlapping subdomains $\Omega_1 = [0,0.45]\times[0,1]$ and $\Omega_2 = [0.45,1]\times[0,1]$ with $g(x,y) = \sin(\pi x)\cos(\pi y)$ is the exact solution and $f(x,y) = 2\pi^2 g$ the right hand side. The convergence tests are performed by taking different mesh sizes $h_{\Omega_1} = h \in \{0.2, 0.1, 0.05, 0.025, 0.0125\}$, $h_{\Omega_2} = h_{\Omega_1} + 10^{-3}$ and different Lagrange polygonal orders $P_k,\ k \in \{1,2,3,4,5\}$. We plot the linear regression lines of $\|u-u_h\|_{L^2}$ versus h, and we retrieve the optimal convergence properties provided by the mortar method. Note that the above 2D/3D mortar code in Listing 3 is purely sequential, the parallel version of 2D/3D mortar code for arbitrary number of subdomains is presented in [18].

4 Conclusion

We presented our ongoing work on building a flexible domain decomposition framework in FEEL++. A lot of work remains to be done, however we have already the toolbox to reproduce a large range of domain decomposition methods in sequential and to a lesser extent in parallel. Regarding the Schwarz methods, we are currently working on having them as preconditioners of Krylov subspace methods and building coarse grid preconditioners on massively parallel architectures, see [9]. As to the mortar methods, we have already a 2D/3D parallel code with some simple preconditioner strategy [18] and we develop scalable preconditioners for the constraint space formulation, see [4].

Acknowledgements The authors would like to thank Frédéric Nataf, Silvia Bertoluzza and Pierre Jolivet for many fruitful discussions. This ongoing work has been sponsored by ANR-Cosinus-HAMM and the Region Rhone-Alpes. It was granted access to the HPC resources of TGCC@CEA made available within the Distributed European Computing Initiative by the PRACE-2IP, receiving funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement RI-283493.

References

1. Bagheri, B., Scott, R.: Analysa. http://people.cs.uchicago.edu/ ridg/al/aa.ps
2. Bangerth, W., Hartmann, R., Kanschat, G.: `deal.II` Differential Equations Analysis Library, Technical Reference. URL `http://www.dealii.org`. `http://www.dealii.org`

3. Belgacem, F.B., Maday, Y.: The mortar element method for three-dimensional finite elements. R.A.I.R.O. Modl. Math. Anal. **31**, 289–302 (1997)
4. Bertoluzza, S., Pennacchio, M.: Analysis of substructuring preconditioners for mortar methods in an abstract framework. Applied Mathematics Letters **20**(2), 131 – 137 (2007). DOI 10.1016/j.aml.2006.02.029. URL `http://www.sciencedirect.com/science/article/pii/S0893965906001108`
5. Chabannes, V., Prud'homme, C., Pena, G.: High order fluid structure interaction in 2D and 3D: Application to blood flow in arteries. Journal of Computational and Applied Mathematics (Accepted) (2012)
6. Del Pino, S., Pironneau, O.: FreeFEM3D Manual. Laboratoire Jacques Louis Lions (2005)
7. Doyeux, V., Chabannes, V., Prud'homme, C., Ismail, M.: Simulation of two fluid flow using a level set method application to bubbles and vesicle dynamics. Journal of Computational and Applied Mathematics (Accepted) (2012)
8. Dular, P., Geuzaine, C.: Getdp: a general environment for the treatment of discrete problems. http://www.geuz.org/getdp
9. Jolivet, P., Dolean, V., Hecht, F., Nataf, F., Prud'homme, C., Spillane, N.: High performance domain decomposition methods on massively parallel architectures with FreeFem++. Journal of Numerical Mathematics **20**(3), to appear (2012)
10. Kirby, R.C., Logg, A.: Efficient compilation of a class of variational forms. ACM Trans. Math. Softw. **33**(3), 17 (2007). DOI http://doi.acm.org/10.1145/1268769.1268771
11. Long, K.: Sundance: Rapid development of high-performance parallel finite-element solutions of partial differential equations. http://software.sandia.gov/sundance/
12. Nitsche, J.: ber ein variationsprinzip zur lsung von dirichlet-problemen bei verwendung von teilrumen, die keinen randbedingungen unterworfen sind. Abh. Math. Sem. Univ. Hamburg **36**, 9–15 (1971). Collection of articles dedicated to Lothar Collatz on his sixtieth birthday
13. Pena, G., Prud'homme, C., Quarteroni, A.: High order methods for the approximation of the incompressible navierstokes equations in a moving domain. Computer Methods in Applied Mechanics and Engineering **209-212**, 197–211 (2011)
14. Prud'homme, C.: Life: Overview of a unified C++ implementation of the finite and spectral element methods in 1d, 2d and 3d. In: In Workshop On State-Of-The-Art In Scientific And Parallel Computing, Lecture Notes in Computer Science, page 10. Springer-Verlag (2007)
15. Prud'Homme, C., Chabannes, V., Doyeux, V., Ismail, M., Samake, A., Pena, G., et al.: Feel++: A computational framework for galerkin methods and advanced numerical methods. Esaim Proceedings (2012). URL `http://hal.archives-ouvertes.fr/docs/00/66/35/18/PDF/feel.pdf`. Accepted
16. Quarteroni, A., Valli, A.: Domain decomposition methods for partial Differential equations, chap. The Mathematical Fundation of Domain Decomposition Methods, pp. 1–39. Numerical Mathematics and Scientific Computation. Oxford University Press, New York (1999)
17. Renard, Y., Pommier, J.: Getfem++: Generic and efficient c++ library for finite element methods elementary computations. http://www-gmm.insa-toulouse.fr/getfem/
18. Samake A. Prud'Homme, C., Zaza, C., Chabannes, V.: Parallel implementation of the mortar element method. Esaim Proceedings (2013). In progress
19. Smith, B., Bjorstad, P., Gropp, W.: Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press (2004). URL `http://books.google.fr/books?id=dxwRLu1dBioC`
20. Toselli, A., Widlund, O.: Domain Decomposition Methods - Algorithms and Theory. Springer Series in Computational Mathematics. Springer (2004). URL `http://books.google.fr/books?id=tpSPx68R3KwC`