FreeFem++ parallel solvers

Pierre Jolivet, F. Hecht,
F. Nataf, C. Prud'homme

Laboratoire Jacques-Louis Lions
Laboratoire Jean Kuntzmann
Inria Rocquencourt

Sixth workshop on FreeFem++

December 9, 2014

Outline

Bottlenecks with implicit methods

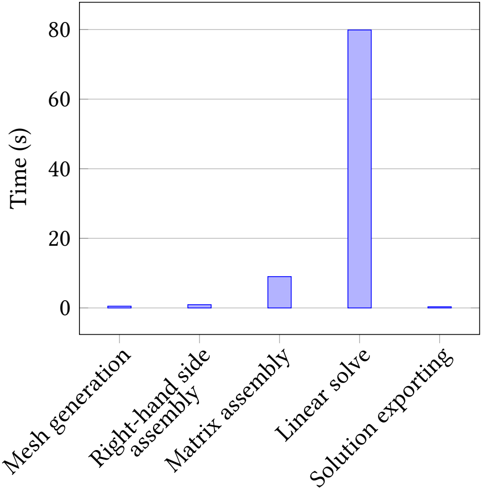
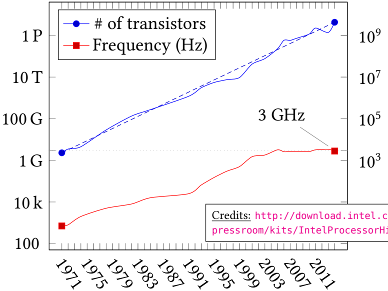


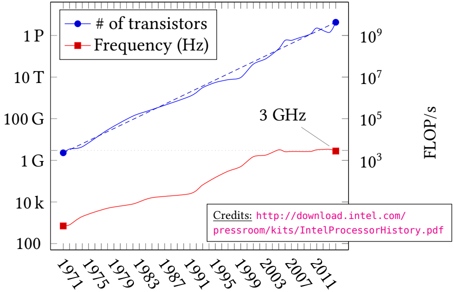
Fig: Wall-clock times spent in various steps of a complete FE simulation using Feel++ (Prud'homme 2006) for solving Stokes equations in 3D.

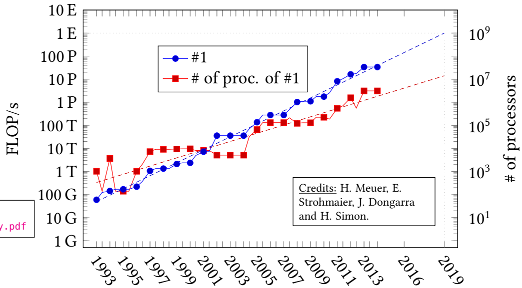
Possible improvements



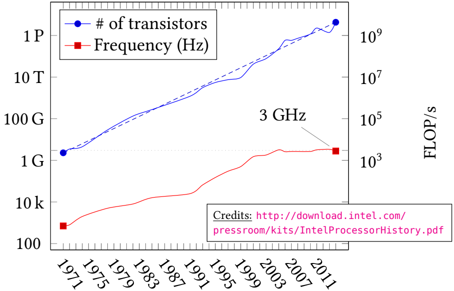
Credits: http://download.intel.com/pressroom/kits/IntelProcessorHistory.pdf

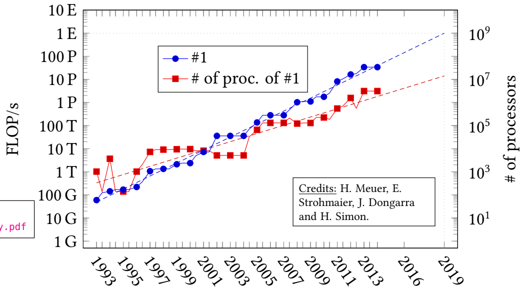
Possible improvements





Possible improvements





1. parallel solve, sections 2 and 3,
2. parallel assembly, section 3.

FreeFem++ and linear solvers

`set(A, solver = sparsesolver)` in FreeFem++

$\Downarrow$

instance of `VirtualSolver<T>` from which inherits a solver.

FreeFem++ and linear solvers

`set(A, solver = sparsesolver)` in FreeFem++

$\Downarrow$

instance of `VirtualSolver<T>` from which inherits a solver.

Interfacing a new solver basically consists in implementing:

1. the constructor `MySolver(const MatriceMorse<T>&)`,

FreeFem++ and linear solvers

`set(A, solver = sparsesolver)` in FreeFem++

$\Downarrow$

instance of `VirtualSolver<T>` from which inherits a solver.

Interfacing a new solver basically consists in implementing:

1. the constructor `MySolver(const MatriceMorse<T>&)`,
2. the pure virtual method

```
void Solver(const MatriceMorse<T>&, KN_<T>&,
            const KN_<T>&) const,
```

FreeFem++ and linear solvers

`set(A, solver = sparsesolver)` in FreeFem++

$\Downarrow$

instance of `VirtualSolver<T>` from which inherits a solver.

Interfacing a new solver basically consists in implementing:

1. the constructor `MySolver(const MatriceMorse<T>&)`,
2. the pure virtual method

```
void Solver(const MatriceMorse<T>&, KN_<T>&,
                  const KN_<T>&) const,
```

3. the destructor `MySolver(const MatriceMorse<T>&)`.

FreeFem++ and linear solvers

`set(A, solver = sparsesolver)` in FreeFem++

$\Downarrow$

instance of `VirtualSolver<T>` from which inherits a solver.

Interfacing a new solver basically consists in implementing:

1. the constructor `MySolver(const MatriceMorse<T>&)`,
2. the pure virtual method

```
void Solver(const MatriceMorse<T>&, KN_<T>&,
            const KN_<T>&) const,
```

3. the destructor `MySolver(const MatriceMorse<T>&)`.

`real[int] x = A^-1 * b` will call `MySolver::Solver`.

MUltifrontal Massively Parallel Sparse direct Solver

`http://graal.ens-lyon.fr/MUMPS`

Distributed memory direct solver.
Compiled by FreeFem++ with `--enable-download`.
Renumbering via AMD, QAMD, AMF, PORD, (Par)METIS, (PT-)SCOTCH.

Solves unsymmetric and *symmetric* linear systems !

MUMPS and FreeFem++

```
load "MUMPS"
int[int] l = [1, 1, 2, 2];
int master = 0;
bool isMaster = mpirank == master;
mesh Th = square((isMaster?150:1), (isMaster?150:1), label = l);
fespace Vh(Th, P2);
varf lap(u,v) = int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v)) + int2d(Th)(v)
     + on(1, u = 1);
real[int] b = lap(0, Vh);
matrix A = lap(Vh, Vh, solver = CG);
set(A, solver = sparsesolver, master = master);
Vh u;
u[] = A^-1 * b;
if(isMaster)
        plot(Th, u, wait = 1, dim = 3, fill = 1, value = 1);
```

Intel MKL PARDISO

`http://software.intel.com/en-us/intel-mkl`

Shared memory direct solver.
Part of the Intel Math Kernel Library.
Renumbering via Metis or threaded nested dissection.

Solves unsymmetric and *symmetric* linear systems !

PARDISO and FreeFem++

```
load "PARDISO"
int[int] l = [1, 1, 2, 2];
mesh Th = square(150, 150, label = l);
fespace Vh(Th, P2);
Vh u;
varf lap(u,v) = int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v)) + int2d(Th)(v)
      + on(1, u = 1);
real[int] b = lap(0, Vh);
matrix A = lap(Vh, Vh, solver = CG);
verbosity = 2;
set(A, solver = sparsesolver);
verbosity = 0;
Vh x;
x[] = A^-1 * b;
plot(Th, x, wait = 1, dim = 3, fill = 1, value = 1);
```

Is it really necessary ?

If you are using direct solvers with FreeFem++: yes !

Sooner or later, `UMFPACK` will blow up:

```
UMFPACK V5.5.1 (Jan 25, 2011): ERROR: out of memory

umfpack_di_numeric failed
```

Motivation

One of the most straightforward way to solve BVP in parallel.

Motivation

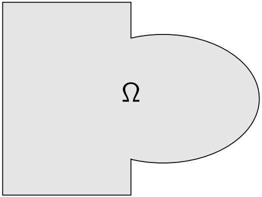
One of the most straightforward way to solve BVP in parallel.

Based on the “divide and conquer” paradigm:

1. assemble,
2. factorize, and
3. solve smaller problems.

Overlapping methods I

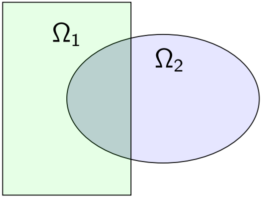
Consider the linear system: $Au = f \in \mathbb{R}^n$.



Overlapping methods I

Consider the linear system: $Au = f \in \mathbb{R}^n$.
Given a decomposition of $[\![1; n]\!]$, $(\mathcal{N}_1, \mathcal{N}_2)$, define:

- the restriction operator R_i from $[\![1; n]\!]$ into $\mathcal{N}_i$,
- R_i^T as the extension by 0 from $\mathcal{N}_i$ into $[\![1; n]\!]$.



Overlapping methods I

Consider the linear system: $Au = f \in \mathbb{R}^n$.
Given a decomposition of $[\![1; n]\!]$, $(\mathcal{N}_1, \mathcal{N}_2)$, define:

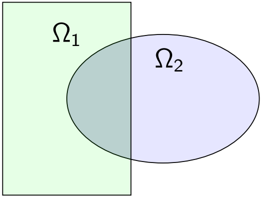
- the restriction operator R_i from $[\![1; n]\!]$ into $\mathcal{N}_i$,
- R_i^T as the extension by 0 from $\mathcal{N}_i$ into $[\![1; n]\!]$.

Then solve concurrently:

$$u_1^{m+1} = u_1^m + A_{11}^{-1} R_1 (f - Au^m) \qquad u_2^{m+1} = u_2^m + A_{22}^{-1} R_2 (f - Au^m)$$

where $u_i = R_i u$ and $A_{ij} := R_i A R_j^T$.

(Schwarz 1870)

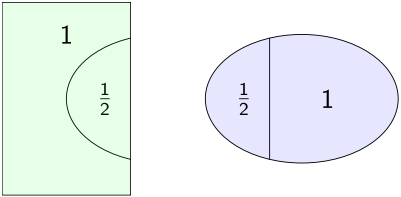


Overlapping methods II

Problem is effectively divided, but yet to be conquered.

Duplicated unknowns coupled via a *partition of unity*:

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

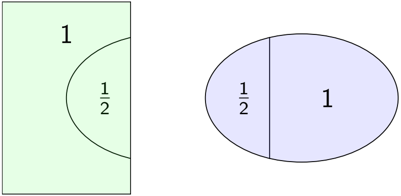


Overlapping methods II

Problem is effectively divided, but yet to be conquered.

Duplicated unknowns coupled via a *partition of unity*:

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



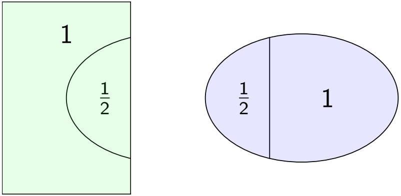
Then, $u^{m+1} = \sum_{i=1}^{N} R_i^T D_i u_i^{m+1}$.

Overlapping methods II

Problem is effectively divided, but yet to be conquered.

Duplicated unknowns coupled via a *partition of unity*:

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

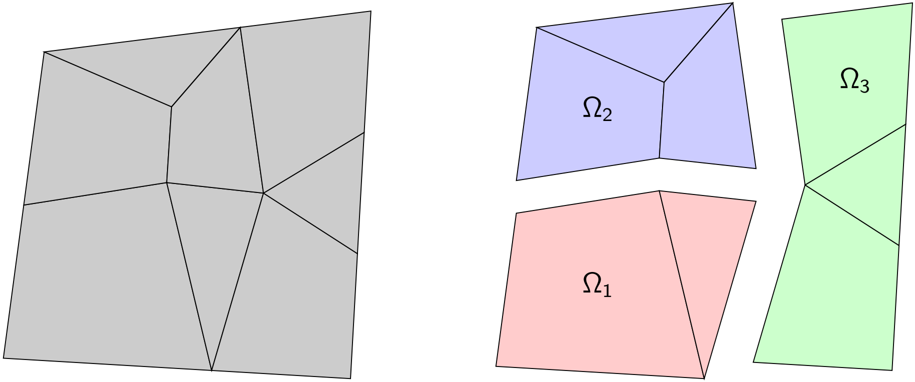


Then, $u^{m+1} = \sum_{i=1}^{N} R_i^T D_i u_i^{m+1}$.

$$M_{\text{RAS}}^{-1} = \sum_{i=1}^{N} R_i^T D_i A_{ii}^{-1} R_i$$

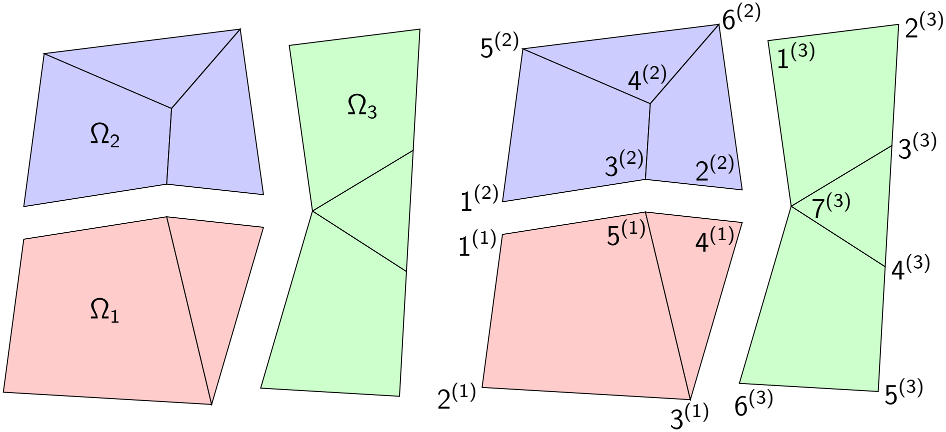
(Cai and Sarkis 1999)

Substructuring methods I



Subdomain tearing

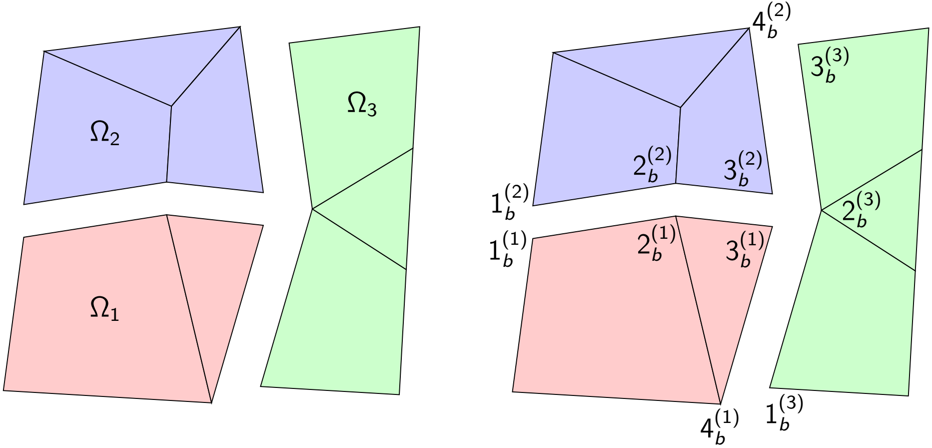
Substructuring methods I



Local numbering

$$A^{(k)} = \begin{bmatrix} A_{ii} & A_{ib} \\ A_{bi} & A_{bb} \end{bmatrix}$$

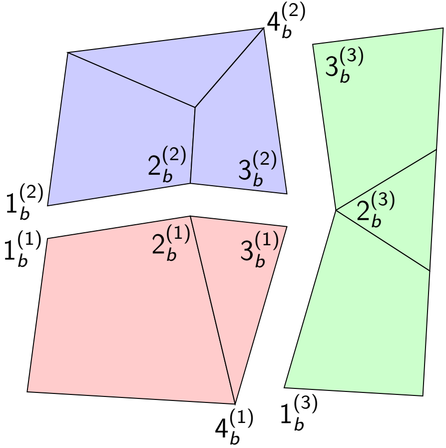
Substructuring methods I



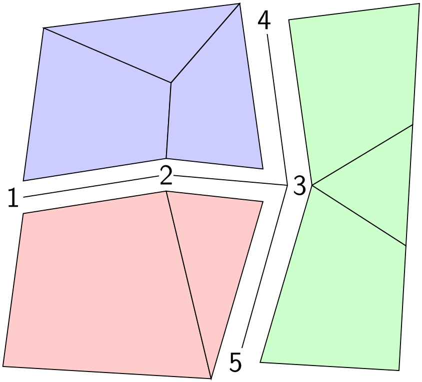
Elimination of interior d.o.f.

$$S^{(k)} = A_{bb} - A_{bi}A_{ii}^{-1}A_{ib}$$

Substructuring methods I

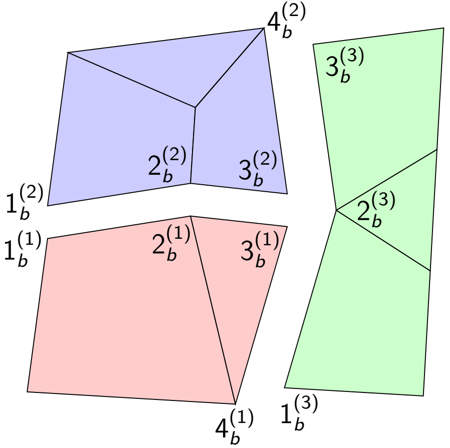


Jump operators: $\{B^{(i)}\}_{i=1}^{3}$

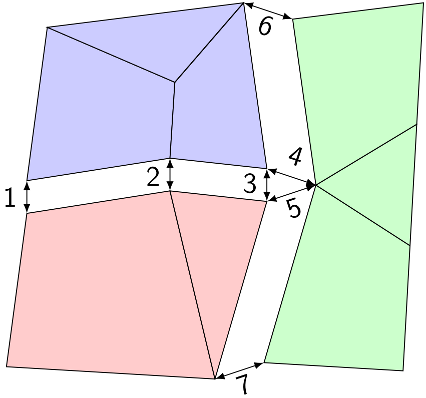


Primal constraints
(Mandel 1993)

Substructuring methods I



Jump operators: $\{\underline{B}^{(i)}\}_{i=1}^{3}$



Dual constraints
(Farhat and Roux 1991)

Substructuring methods II

The new system reads:

$$\forall i \in [\![1; N]\!],\ S^{(i)} u_b^{(i)} = g_i + \lambda_b^{(i)}$$

Substructuring methods II

The new system reads:

$$\forall i \in [\![1; N]\!],\ S^{(i)} u_b^{(i)} = g_i + \lambda_b^{(i)}$$
$$\sum_{i=1}^{N} \underline{B}^{(i)} u_b^{(i)} = 0$$

Substructuring methods II

The new system reads:

$$\forall i \in [\![1; N]\!],\ S^{(i)} u_b^{(i)} = g_i + \lambda_b^{(i)}$$
$$\sum_{i=1}^{N} \underline{B}^{(i)} u_b^{(i)} = 0$$
$$\sum_{i=1}^{N} B^{(i)} \lambda_b^{(i)} = 0\,.$$

Efficient preconditioners (based on scaled sum): (Dohrmann 2003; Farhat, Mandel, et al. 1994; Rixen and Farhat 1997)

Limitations of one-level methods

One-level methods don't require exchange of global information.

This hampers numerical scalability of such preconditioners:

$$\kappa(M^{-1}A) \leqslant C\frac{1}{H^2}\left(1 + \frac{H}{\delta}\right)$$

- level of overlap δ,
- characteristic size of a subdomain H.

(Le Tallec 1994; Toselli and Widlund 2005)

Two-level preconditioners I

A common technique in the field of DDM, MG, deflation:

introduce an auxiliary “coarse” problem.

Let Z be a rectangular matrix. Define

$$E := Z^T AZ \, .$$

Z has $\mathcal{O}(N)$ columns, hence E is much smaller than A.

Two-level preconditioners I

A common technique in the field of DDM, MG, deflation:

introduce an auxiliary “coarse” problem.

Let Z be a rectangular matrix. Define

$$E := Z^T A Z\,.$$

Z has $\mathcal{O}(N)$ columns, hence E is much smaller than A.
Enrich the original preconditioner, e.g. additively

$$P^{-1} = M^{-1} + Z E^{-1} Z^T,$$

cf. (Tang et al. 2009).

HPDDM

A framework for high-performance domain decomposition methods

`https://github.com/hpddm/hpddm`

Implements one- and two-level Schwarz methods, as well as FETI and BDD methods.
Compiled by FreeFem++ with `--enable-download`.
Depends on ARPACK and a direct solver (UMFPACK, CHOLMOD, MUMPS, PARDISO, PaStiX).

Solves elliptic and frequency domain linear systems !

HPDDM

A framework for high-performance domain decomposition methods

`https://github.com/hpddm/hpddm`

Implements one- and two-level Schwarz methods, as well as FETI and BDD methods.
Compiled by FreeFem++ with `--enable-download`.
Depends on ARPACK and a direct solver (UMFPACK, CHOLMOD, MUMPS, PARDISO, PaStiX).

Solves elliptic and frequency domain linear systems !

New types

`?schwarz`, `?feti`, `?bdd` (`?` = d or z)

PETSc

Portable, Extensible Toolkit for Scientific Computation

`http://www.mcs.anl.gov/petsc`

Can be used in conjunction with HPDDM.

Solves any kind of real linear system !

PETSc

Portable, Extensible Toolkit for Scientific Computation

`http://www.mcs.anl.gov/petsc`

Can be used in conjunction with HPDDM.

Solves any kind of real linear system !

New types

`dmatrix` (no complex unknowns interface with FreeFem++)

Tomorrow's tutorial

(Try to) start from scratch and understand the following examples:

1. one- and two-level Schwarz methods for a scalar diffusion equation, the system of linear elasticity, and the Helmholtz equation,
2. PETSc solvers for Stokes equation.

"Expert tutorial", with some "basic tutorial" materials.

To sum up

FreeFem++ is now interfaced with mature parallel solvers:
$\Longrightarrow$ better performance for solving (non)linear systems.

To sum up

FreeFem++ is now interfaced with mature parallel solvers:
$\Longrightarrow$ better performance for solving (non)linear systems.

Thank you !