Open-source finite element solver for domain decomposition problems

C. Geuzaine[1], X. Antoine[2,3], D. Colignon[1], M. El Bouajaji[3,2] and B. Thierry[4]

1 - University of Liège, Belgium
2 - University of Lorraine, France
3 - INRIA Nancy Grand-Est, France
4 - OPERA, Applied Geophysical Research Group, Pau, France

DD22, September, 2013

Outline

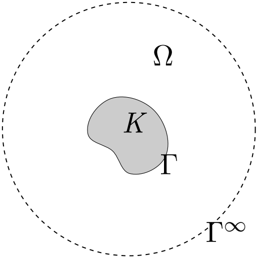
***

GetDP

Results

Conclusions and perspectives

Reference problem



Scattered field u solution of

$$\begin{cases} (\Delta + k^2)u = 0 & \text{in } \Omega := \mathbb{R}^d \setminus \overline{K}, \\ u = -u^{inc} & \text{on } \Gamma, \\ \partial_{\mathbf{n}} u - iku = 0 & \text{on } \Gamma^\infty, \end{cases}$$

Reference problem

Non-overlapping DDM, iteration $n+1$

For $j = 0, \dots, N_{\text{dom}} - 1$, do ($\Omega := \bigcup_j \Omega_j$):

1. Compute the fields $u_j^{n+1} := u^{n+1}|_{\Omega_j}$:

$$\left\{ \begin{array}{rcll} (\Delta + k^2) u_j^{n+1} & = & 0 & \text{in } \Omega_j, \\ u_j^{n+1} & = & -u^{inc} & \text{on } \Gamma_j, \\ \partial_{\mathbf{n}} u_j^{n+1} - iku_j^{n+1} & = & 0 & \text{on } \Gamma_j^{\infty}, \\ \partial_{\mathbf{n}} u_j^{n+1} + \mathcal{S} u_j^{n+1} & = & g_{ij}^n & \text{on } \Sigma_{ij} \ (\forall i \neq j). \end{array} \right.$$

2. Update the data g_{ji}^{n+1}

$$g_{ji}^{n+1} = -g_{ij}^n + 2\mathcal{S} u_j^{n+1}, \quad \text{on } \Sigma_{ij}.$$

Where:

- $\Sigma_{ij} := \partial\Omega_i \bigcap \partial\Omega_j$, $\Gamma_j = \Gamma \bigcap \partial\Omega_j$ and $\Gamma_j^{\infty} = \Gamma^{\infty} \bigcap \partial\Omega_j$
- $\mathcal{S}$: transmission operator (Sommerfeld, OO2, GIBC,...)

Reference problem

Recast the DDM into the linear system:

$$(\mathcal{I} - \mathcal{A})g = b, \tag{1.1}$$

where $\mathcal{I}$ = identity operator and ...

$b = (b_{ij})_{j \neq i}$, with

$$b_{ij} = 2\mathcal{S} v_j \quad (\Sigma_{ij}),$$

with

$$\left\{ \begin{array}{rcll} (\Delta + k^2) v_j & = & 0 & (\Omega_j), \\ v_j & = & -u^{inc} & (\Gamma_j), \\ \partial_{\mathbf{n}} v_j - ikv_j & = & 0 & (\Gamma_j^\infty), \\ \partial_{\mathbf{n}} v_j + \mathcal{S} v_j & = & 0 & (\Sigma_{ij}). \end{array} \right.$$

$g = (g_{ij})_{j \neq i}$, with

$$(\mathcal{A}g)_{ji} = -g_{ij} + 2\mathcal{S} w_j \quad (\Sigma_{ij}),$$

with

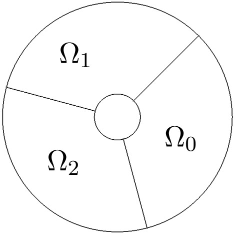
$$\left\{ \begin{array}{rcll} (\Delta + k^2) w_j & = & 0 & (\Omega_j), \\ w_j & = & 0 & (\Gamma_j), \\ \partial_{\mathbf{n}} w_j - ikw_j & = & 0 & (\Gamma_j^\infty), \\ \partial_{\mathbf{n}} w_j + \mathcal{S} w_j & = & g_{ij} & (\Sigma_{ij}). \end{array} \right.$$

System (1.1) can be solved using a Krylov subspaces solver (gmres, ...).

Reference problem

Main steps

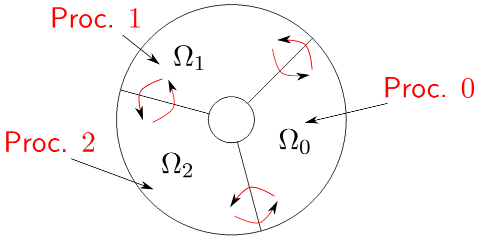
1. Compute b: solve N_{dom} (Helmholtz + surface PDE)
2. Iterative linear solver to solve $(\mathcal{I} - \mathcal{A})g = b$:
    - Solve N_{dom} (Helmholtz + surface PDE): $(g_{ij}^{n+1})_{j \neq i}$
3. Compute u: solve N_{dom} Helmholtz



Reference problem

Main steps in parallel

1. Compute b: solve N_{dom} (Helmholtz + surface PDE) in parallel
2. Iterative linear solver to solve $(\mathcal{I} - \mathcal{A})g = b$:
    - Exchange data between subdomains/MPI process
    - Solve N_{dom} (Helmholtz + surface PDE) in parallel: $(g_{ij}^{n+1})_{j \neq i}$
3. Exchange data between subdomains/MPI process
4. Compute u: solve N_{dom} Helmholtz in parallel



GetDP

GetDP=General Environment for the Treatment of Discrete Problems

Authors

- C. Geuzaine and P. Dular

History

- Started at the end of 1996 (first binary release mid 1998)
- Open-Source under GNU GPL (since 2004)

Design

- Small, fast, no GUI
- End-users: not yet another library
- Limit external dependencies to a minimum

How ?

- Clear *mathematical structure*

Languages

- C/C++ with PETSc

GetDP: DD add-on

Motivation of the add-on

Solve a linear system $AX = b$ with a “matrix-free” vector product.

Goal of the add-on

- DD made simple: passing from mono-domain to multi-domain must not be painful
- Parallelization: easy and prevent the user from writing in MPI
- General method: no restriction to a PDE, a Krylov solver, ...
- Fast and efficient
- Robust and scalable
- Open-source

GetDP: DD add-on

When the user have a code solving a mono-domain PDE with GetDP ...

What the user must do...

- All the mathematics of DDM
- Build/Decompose mesh with *e.g.* GMSH (some plugins exist)
- Define the matrix-free vector product
- Define parameters: solver, tolerance, ...

If(mpirun)

- Chose a distribution of the domains/unknown to MPI process
- Opt.: specify subdomains' neighbors (speed up)

What GetDP will do...

- Manages the iterative linear solver (PETSc)
- If(mpirun): Automatically exchanges data between process (MPI)

GetDP: DD add-on

Remark: surface unknown g are interpolable

- Mixed formulations
- Non conform mesh at interfaces

GetDP: code

Main steps in parallel

1. Compute b: solve N_{dom} Helmholtz + N_{dom} surface PDE in parallel
2. Iterative linear solver to solve $(\mathcal{I} - \mathcal{A})g = b$:
   - Exchange data between subdomains/MPI process
   - Solve N_{dom} Helmholtz + N_{dom} surface PDE in parallel: $(g_{ij}^{n+1})_{j \neq i}$
3. Exchange data between subdomains/MPI process
4. Compute u: solve N_{dom} Helmholtz

GetDP: code

Weak formulations ($\mathcal{S} := -ik$ (Sommerfeld) and $\mathcal{B} := -ik$ (Sommerfeld)).
For $j = 0, \ldots, N_{\text{dom}} - 1$ (v_j:= test functions):

$$\underbrace{\int_{\Omega_j} \nabla u_j \cdot \overline{\nabla v_j} - \int_{\Omega_j} k^2 u_j \cdot \overline{v_j}}_{\textit{Helmholtz equation}} \underbrace{- \int_{\Gamma_j^\infty} iku_j \cdot \overline{v_j}}_{\textit{Sommerfeld ABC}} \underbrace{- \int_{\Sigma_j} g_j^{in} \cdot \overline{v_j} - \int_{\Sigma_j} iku_j \cdot \overline{v_j}}_{\textit{Transmission condition}} = 0$$

```
For j In {0:N_DOM-1}
  [...]
  Galerkin { [ Dof{Grad u~{j}} , {Grad u~{j}} ] ;
    In Omega~{j}; Jacobian JVol ; Integration I1 ; }
  Galerkin { [ -k^2 * Dof{u~{j}} , {u~{j}} ] ;
    In Omega~{j}; Jacobian JVol ; Integration I1 ; }

  Galerkin { [ - I[] * k * Dof{u~{j}} , {u~{j}} ] ;
    In GammaInf~{j}; Jacobian JSur ; Integration I1 ; }

  Galerkin { [ - g_in~{j}[] , {u~{j}} ] ;
    In Sigma~{j}; Jacobian JSur; Integration I1 ; }
  Galerkin { [ -I[] * k * Dof{u~{j}} , {u~{j}} ] ;
    In Sigma~{j}; Jacobian JSur ; Integration I1 ; }
  [...]
EndFor
```

GetDP: code

More complicated transmission condition (OO2: Gander, Magoulès, Nataf, 2002):

$$a u_j - b\Delta_{\Sigma_j} u_j - g_j^{in} = 0, \quad \text{on } \Sigma_j.$$

Weak formulation ($v_j :=$ test function):

$$\int_{\Sigma_j} 2 a u_j \cdot \overline{v_j} - \int_{\Sigma_j} b \nabla_{\Sigma_j} u_j \cdot \nabla_{\Sigma_j} \overline{v_j} + \int_{\Sigma_j} g_j^{in} \cdot \overline{v_j} = 0.$$

```
Galerkin { [ a[]* Dof{u~{j}} , {u~{j}} ] ;
  In Sigma~{j}; Jacobian JSur ; Integration I1 ; }
Galerkin { [ -b[] * Dof{d u~{j}} , {d u~{j}} ] ;
  In Sigma~{j}; Jacobian JSur ; Integration I1 ; }
Galerkin { [ - g_in~{j}[] , {u~{j}} ] ;
  In Sigma~{j}; Jacobian JSur ; Integration I1 ; }
```

GetDP: code

Main steps in parallel

1. Compute b: solve N_{dom} (Helmholtz + surface PDE) in parallel
2. Iterative linear solver to solve $(\mathcal{I} - \mathcal{A})g = b$:
    - Exchange data between subdomains/MPI process
    - Solve N_{dom} (Helmholtz + surface PDE) in parallel: $(g_{ij}^{n+1})_{j \neq i}$
3. Exchange data between subdomains/MPI process
4. Compute u: solve N_{dom} Helmholtz

GetDP: code

Acoustic:

```
IterativeLinearSolver["I-A", solver, tol, maxit, ...]
{
  SetCommSelf ; // Sequential mode
  For ii In {0: #ListOfDom()-1}
    j = ListOfDom(ii) ;
    // Solve Helmholtz on Ω_j
    GenerateRHSGroup[Helmholtz~{j}, Sigma~{j}] ;
    SolveAgain[Helmholtz~{j}] ;
    // Compute g_ji
    GenerateRHSGroup[ComputeG~{j}, Sigma~{j}] ;
    SolveAgain[ComputeG~{j}] ;
  EndFor
  // Update g_ji in memory
  For ii In {0: #ListOfDom()-1}
    j = ListOfDom(ii) ;
    PostOperation[g_out~{j}] ;
  EndFor
  SetCommWorld ;// Parallel mode
}
```

GetDP: code

And for Maxwell's equation . . .

```
IterativeLinearSolver["I-A", solver, tol, maxit, ...]
{
  SetCommSelf ; // Sequential mode
  For ii In {0: #ListOfDom()-1}
    j = ListOfDom(ii) ;
    // Solve Maxwell on Ω_j
    GenerateRHSGroup[Maxwell~{j}, Sigma~{j}] ;
    SolveAgain[Maxwell~{j}] ;
    // Compute g_ji
    GenerateRHSGroup[ComputeG~{j}, Sigma~{j}] ;
    SolveAgain[ComputeG~{j}] ;
  EndFor
  // Update g_ji in memory
  For ii In {0: #ListOfDom()-1}
    j = ListOfDom(ii) ;
    PostOperation[g_out~{j}] ;
  EndFor
  SetCommWorld ;// Parallel mode
}
```

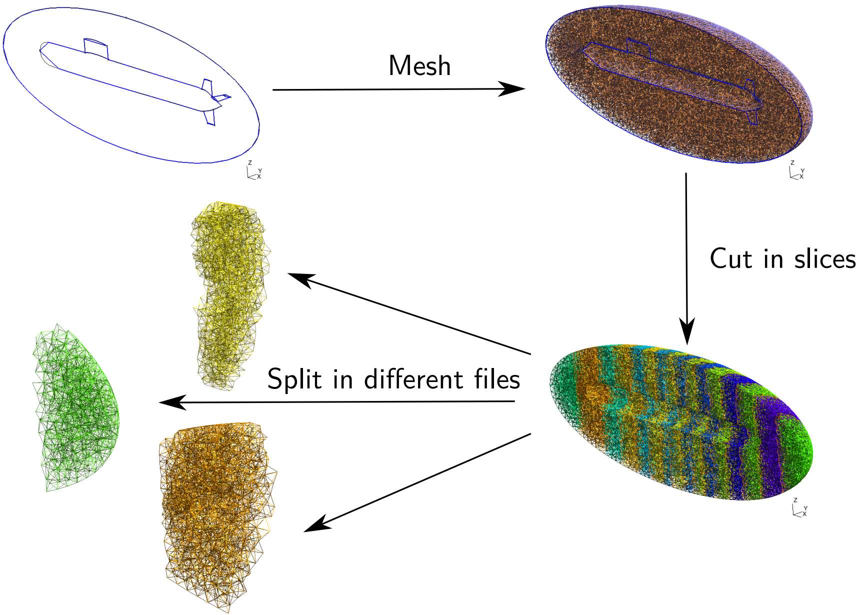
***

GetDP

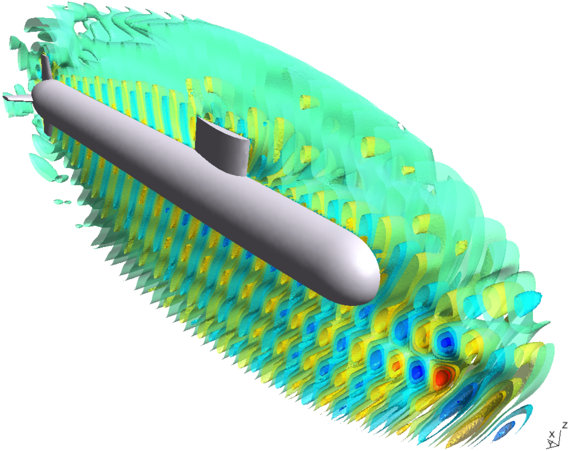
Results

Conclusions and perspectives

Acoustic: 3D submarine



Acoustic: 3D submarine



Submarine problem with 16 subdomains: iso-surfaces of the real part of the scattered field for $k = 41\pi$.

Acoustic: 3D submarine

Vega Cluster (ULB, Belgium): 42 nodes, each with 256GB of RAM and 4 AMD 6722 processors, with 16 cores at 2.1GHz (MPI/OpenBlas).

$k = 10\pi$ and $L_{sub} = 1$

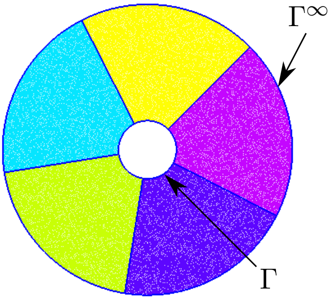
Nb. domains	Nb. nodes	Nb. MPI/nodes	Nb. threads/MPI	Nb. vertices (M.)	Nb. iteration	CPU time (min)	Max RAM (GB)	Size volume PDE	Size surface PDE	Size GMRES
16	16	1	16	40	127	302	137	27×10^5	10×10^4	9×10^6
32	16	2	8	2.5	113	9	3	1.2×10^5	1.6×10^4	2.9×10^6
				5	121	21	7	2.1×10^5	2.5×10^4	4.5×10^6
				10	129	42	13	4.1×10^5	4×10^4	7.2×10^6
				20	158	90	?	8×10^5	6.7×10^4	12×10^6
				40	171	205	60	15×10^5	10×10^4	18×10^6
	16	2	8	40	171	182	63	15×10^5	10×10^4	18×10^6
				80	184	495	141	29×10^5	16×10^4	29×10^6
				120	192	795	220	42×10^5	21×10^4	38×10^6
64	32	2	16	40	240	250	40	8×10^5	10×10^4	38×10^6
				80	259	817	85	16×10^5	16×10^4	60×10^6
				120	270	1072	118	23×10^5	21×10^4	78×10^6

Acoustic: 3D submarine

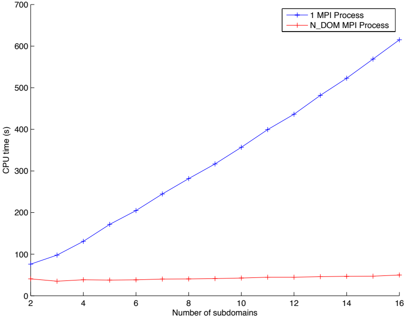
NIC4 cluster (ULg, Belgium): 120 nodes, each with 64GB of RAM and 2 Intel E2650 processors with 8 cores at 2GHz (MPI/MKL).
$k = 10\pi$ and $L_{sub} = 1$.

Nb. domains	Nb. nodes	Nb. MPI/nodes	Nb. threads/MPI	Nb. vertices (M.)	Nb. iterations	CPU time (min)	Max RAM (GB)	Size volume PDE	Size surface PDE	Size GMRES
32	32	1	16	40	171	100	59.5	14.9×10^5	10^5	1.8×10^7
64	64	1	16	40	240	160	39.4	8.8×10^5	10^5	3.8×10^7
96	96	1	16	40	303	195	36.3	6.7×10^5	10^5	5.7×10^7
120	120	1	16	80	371	399	63.0	10.8×10^5	1.6×10^5	11.4×10^7

Acoustic: scalability (2D)

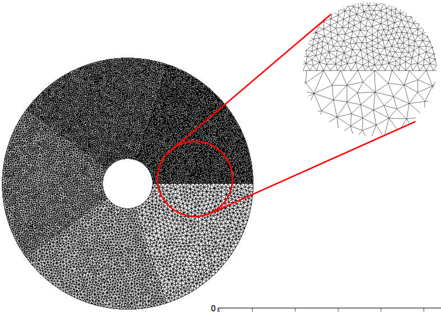


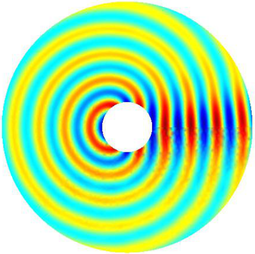
Circle-pie decomposition (2D).

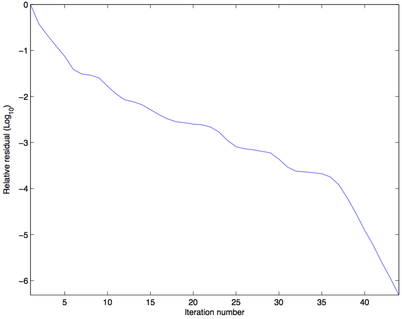


CPU time vs. N_{dom} with 1 or N_{dom} number of MPI process.

Acoustic: non conform (simple case)





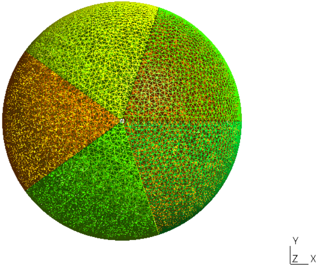


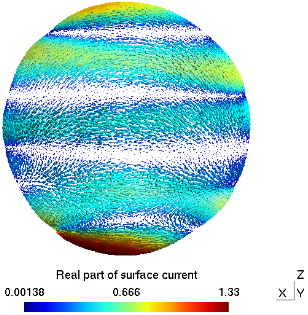
$k = 2\pi$

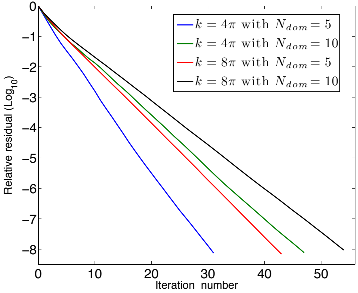
$N_{dom} = 5$

tol: 10^{-6}

Electromagnetism 3D







- Perfectly conducting sphere
- Two concentric spheres
- $N_{\text{dom}} = 5$ or 10

***

GetDP

Results

Conclusions and perspectives

Conclusions

Our software...

- Quite easy to pass from mono-domain to multi-domain
- Fast and efficient
- Has been successfully tested on cluster
- Adapted to scalar/vector/tensor unknown

What can be done with it...

- 1D/2D/3D without change (GetDP is 3D native)
- Automatic partitioning ("waveguide-style")
- With or without overlap
- Mixte formulations
- Non conform mesh at interfaces (interpolation)
- Right (or left) preconditioning: $(\mathcal{I} - \mathcal{A})\mathcal{F}^{-1}\widetilde{g} = b$ with $\mathcal{F}g = \widetilde{g}$.

Perspectives

- Publish DDM codes online (acoustic/electromagnetism)
- Documentation
- Computational optimizations (storage of unknown, . . .)
- Optimize/Make easier mesh decomposition (topological tools)
- Really automatic partitioning algorithms (metis/chaco)
- Hope some of you will use it :-)

Perspectives

- Publish DDM codes online (acoustic/electromagnetism)
- Documentation
- Computational optimizations (storage of unknown, . . .)
- Optimize/Make easier mesh decomposition (topological tools)
- Really automatic partitioning algorithms (metis/chaco)
- Hope some of you will use it :-)

Thank you very much !