GetDDM: an Open Framework for Testing Optimized Schwarz Methods for Time-Harmonic Wave Problems

C. Geuzaine[1], X. Antoine[2], D. Colignon[1], M. El Bouajaji[2], N. Marsic[1], B. Thierry[1,3], S. Tournier[1,4], A. Vion[1]

1 - University of Liège, Belgium
2 - University of Lorraine, France
3 - University of Paris 6, France
4 - Pontificia Universidad Católica de Chile, Chile

http://onelab.info/wiki/GetDDM
http://onelab.info/wiki/DDM_for_Waves

DD23, July 6^{th} 2013

Outline

Reference problem

GetDDM

Full example provided with the software

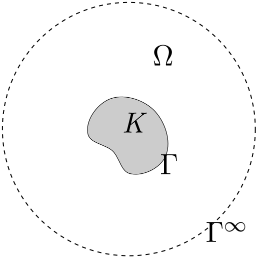
***

Reference problem

GetDDM

Full example provided with the software

Reference problem



Scattered field u solution of

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

Reference problem

Non-overlapping additive Schwarz method at iteration $n+1$

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

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

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

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

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

Where:

- $\Sigma_{ij} := \partial\Omega_i \bigcap \partial\Omega_j$, $\Gamma_i = \Gamma \bigcap \partial\Omega_i$ and $\Gamma_i^{\infty} = \Gamma^{\infty} \bigcap \partial\Omega_i$
- $\mathcal{S}$: transmission operator

Reference problem

Recast the DDM into the linear system:

$$(\mathcal{I} - \mathcal{A})g = b, \tag{1}$$

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

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

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

with

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

$g = (g_{ij})_{j \neq i}$, 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) can be solved using a Krylov subspaces solver (gmres, ...).

***

Reference problem

GetDDM

Full example provided with the software

GetDDM

Open-source codes providing facilities to solve DDM problem. . .

- Purely algebraic (e.g. PETSc)
- Linked to a finite element kernels (e.g. HPDDM)

Complimentary point of view of GetDDM

Focus on the (optimized) Schwarz method where the transmission conditions plays a central role by:

- Providing a simple, flexible and ready-to-use environment
- Direct link between discrete and continuous weak-formulations

GetDDM

Generalized Environnement Treatment for Domain Decomposition Method

- Based on GMSH and GetDP.
- Written in C++/PETSc ...
- ... but code's build-in language.
- Works on laptop, super-computer and smartphone[1] without changing the (Ascii) input files.
- Parallelism made simple for the user.
- Full and ready-to-use example is provided: Helmholtz/Maxwell, multiples geometries, preconditioners, ...

[1] Available on Android and iOS markets (search for "ONELAB")

GetDDM: Generalized Environnement Treatment for Domain Decomposition Method

Main steps to solve a problem from scratch:

1. Build the geometries and the meshes (GMSH)
2. Transcribe the weak formulations
3. Distribute the subdomains to the MPI processes
4. Specify the topology (optional)
5. Setting the iterative linear solver (i.e.: defining $\mathcal{A}$)

Remark: the full example provides all these steps in different files.

GetDDM: Generalized Environnement Treatment for Domain Decomposition Method

Main steps to solve a problem from scratch:

1. Build the geometries and the meshes (GMSH)
2. Transcribe the weak formulations
3. Distribute the subdomains to the MPI processes
4. Specify the topology (optional)
5. Setting the iterative linear solver (i.e.: defining $\mathcal{A}$)

Remark: the full example provides all these steps in different files.

Weak formulations: volume PDE

$$\left\{\begin{array}{l}\text{Find } u_i^{(n+1)} \text{ in } H_0^1(\Omega_i) \text{ such that, for every } u_i' \in H_0^1(\Omega_i)\text{:} \\ \displaystyle\int_{\Omega_i} \nabla u_i^{(n+1)} \cdot \nabla u_i' \,\mathrm{d}\Omega_i - \int_{\Omega_i} k^2 u_i^{(n+1)} u_i' \,\mathrm{d}\Omega_i - \int_{\Gamma_i^\infty} iku_i^{(n+1)} u_i' \,\mathrm{d}\Gamma_i^\infty \\ \qquad \displaystyle + \sum_j \int_{\Sigma_{ij}} \mathcal{S} u_i^{(n+1)} u_i' \,\mathrm{d}\Sigma_{ij} = \sum_j \int_{\Sigma_{ij}} g_{ij}^{(n)} u_i' \,\mathrm{d}\Sigma_{ij}.\end{array}\right.$$

```
Formulation {
  { Name Vol~{idom}; Type FemEquation;
    Quantity {
      { Name u~{idom}; Type Local; NameOfSpace Hgrad_u~{idom}; }
    }
    Equation {
      Galerkin { [ Dof{Grad u~{idom}}, {Grad u~{idom}} ];
        In Omega~{idom}; Jacobian JVol; Integration I1; }

      Galerkin { [ - k[]^2 * Dof{u~{idom}}, {u~{idom}} ];
        In Omega~{idom}; Jacobian JVol; Integration I1; }

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

      Galerkin { [ - $ArtificialSource ? g_in~{idom}[] : 0), {u~{idom}} ];
        In Sigma~{idom}; Jacobian JSur; Integration I1; }
    }
  }
}
```

Weak formulations: surface PDE

$$\left\{ \begin{array}{l} \text{Find } g_{ji}^{(n+1)} \text{ in } H^1(\Sigma_{ij}) \text{ such that, for every } g'_{ji} \in H^1(\Sigma_{ij})\text{:} \\ \displaystyle\int_{\Sigma_{ij}} g_{ji}^{(n+1)} g'_{ji} \,\mathrm{d}\Sigma_{ij} = -\int_{\Sigma_{ij}} g_{ij}^{(n)} g'_{ji} \,\mathrm{d}\Sigma_{ij} + 2\int_{\Sigma_{ij}} \mathcal{S} u_i^{(n+1)} g'_{ji} \,\mathrm{d}\Sigma_{ij}. \end{array} \right.$$

```
Formulation {
  { Name Sur~{idom}; Type FemEquation;
    Quantity {
      { Name u~{idom}; Type Local; NameOfSpace Hgrad_u~{idom}; }
      { Name g_out~{idom}; Type Local; NameOfSpace Hgrad_g_out~{idom}; }
    }
    Equation {
      Galerkin { [ Dof{g_out~{idom}} , {g_out~{idom}} ];
        In Sigma~{idom}; Jacobian JSur; Integration I1; }

      Galerkin { [ $ArtificialSource ? g_in~{idom}[] : 0 , {g_out~{idom}} ];
        In Sigma~{idom}; Jacobian JSur; Integration I1; }
    }
  }
}
```

<u>Remark:</u> these codes are taken from the file `Helmholtz.pro`.

Transmission Operators for Helmholtz

Original operator:

Sommerfeld transmission boundary condition IBC(0) [Després, 1991], or IBC(χ) [Bendali & Boubendir, 2008]

$$\mathcal{S}_{\mathsf{IBC}(\chi)} u = (-\imath k + \chi) u$$

with χ is real constant.

```
If(TC_TYPE == 0) // IBC
  Galerkin { [ - I[] * kIBC[] * Dof{u~{idom}} , {u~{idom}} ];
    In Sigma~{idom}; Jacobian JSur; Integration I1; }
EndIf
```

Transmission Operators for Helmholtz

Improved operator:

Optimized Order 2 [Gander, Magoulès and Nataf, 2002]

$$\mathcal{S}_{\mathsf{OO2}(\kappa)} u = a(\kappa) u + b(\kappa) \Delta_\Sigma u,$$

with $a(\kappa)$ and $b(\kappa)$ complex constants obtained by solving a min-max optimization problem (depending on a real parameter κ) and Δ_Σ is the Laplace-Beltrami operator on the interface Σ.

```
If(TC_TYPE == 1) // GIBC(a, b)
  Galerkin { [ a[] * Dof{u~{idom}} , {u~{idom}} ];
    In Sigma~{idom}; Jacobian JSur; Integration I1; }
  Galerkin { [ - b[] * Dof{d u~{idom}} , {d u~{idom}} ];
    In Sigma~{idom}; Jacobian JSur; Integration I1; }
EndIf
```

Transmission Operators for Helmholtz

Padé-localized square-root transmission condition

[Boubendir, Antoine and Geuzaine, 2012]:

$$\mathcal{S}_{\mathsf{GIBC}(N_p,\,\alpha,\,\varepsilon)}u = -ikC_0(\alpha)u - ik\sum_{\ell=1}^{N_p} A_\ell(\alpha)\mathsf{div}_\Sigma\left(\frac{1}{k_\varepsilon^2}\nabla_\Sigma\right)\left(\mathcal{I} + B_\ell(\alpha)\mathsf{div}_\Sigma\left(\frac{1}{k_\varepsilon^2}\nabla_\Sigma\right)\right)^{-1} u,$$

where $k_\varepsilon = k + \imath\varepsilon$, with $\varepsilon = 0.39k^{1/3}\mathcal{H}^{2/3}$, $\mathcal{H}$ being the local mean curvature of the interface, and where $C_0(\alpha)$, $A_\ell(\alpha)$ and $B_\ell(\alpha)$ are constants linked to the complex Padé approximation of $\sqrt{1+z}$, using a rotation α of the branch cut.

Transmission Operators for Helmholtz

```
If(TC_TYPE == 2) // GIBC(NP_OSRC, theta_branch, eps)
  Galerkin { [ - I[] * k[] * OSRC_C0[]{NP_OSRC,theta_branch} * Dof{u~{idom}} ,
      {u~{idom}} ];
    In Sigma~{idom}; Jacobian JSur; Integration I1; }
  For iSide In {0:1}
    For j In{1:NP_OSRC}
      Galerkin { [ I[] * k[] * OSRC_Aj[]{j,NP_OSRC,theta_branch} / keps[]^2 *
          Dof{d phi~{j}~{idom}~{iSide}} , {d u~{idom}} ];
        In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }

      Galerkin { [ - Dof{u~{idom}} , {phi~{j}~{idom}~{iSide}} ];
        In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
      Galerkin { [ - OSRC_Bj[]{j,NP_OSRC,theta_branch} / keps[]^2 *
          Dof{d phi~{j}~{idom}~{iSide}} , {d phi~{j}~{idom}~{iSide}} ];
        In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
      Galerkin { [ Dof{phi~{j}~{idom}~{iSide}} , {phi~{j}~{idom}~{iSide}} ];
        In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
    EndFor
  EndFor
EndIf
```

See the `Helmholtz.pro` file for the complete finite element formulation.

GetDDM: Generalized Environnement Treatment for Domain Decomposition Method

Main steps to solve a problem from scratch:

1. Build the geometries and the meshes (GMSH)
2. Transcribe the weak formulations
3. Distribute the subdomains to the MPI processes
4. Specify the topology (optional)
5. Setting the iterative linear solver (i.e.: defining $\mathcal{A}$)

Operator $\mathcal{A}$ and setting the iterative linear solver

```
IterativeLinearSolver["I-A", SOLVER, TOL, MAXIT, RESTART,
                      {ListOfFields()}, {ListOfConnectedFields()}, {}]
{
    Call SolveVolumePDE;
    Call SolveSurfacePDE;
    Call UpdateSurfaceFields;
}
```

Listing 1: See the Schwarz.pro file.

```
For ii In {0: #ListOfSubdomains()-1}
    idom = ListOfSubdomains(ii);
    GenerateRHSGroup[Vol~{idom}, Region[{Sigma~{idom}, GammaD~{idom}] ];
    SolveAgain[Vol~{idom}];
EndFor
```

Listing 2: Macro SolveVolumePDE

```
For ii In {0: #ListOfSubdomains()-1}
    idom = ListOfSubdomains(ii);
    PostOperation[g_out~{idom}];
EndFor
```

Listing 3: Macro UpdateSurfaceFields

***

Reference problem

GetDDM

Full example provided with the software

How to install it?

1. Download and uncompress the pre-compiled GetDDM code:
   - `http://onelab.info/files/gmsh-getdp-Windows64.zip`
   - `http://onelab.info/files/gmsh-getdp-MacOSX.zip`
   - `http://onelab.info/files/gmsh-getdp-Linux64.zip`

   (Simply browse to `http://onelab.info` and follow the links if you don't want to write down the URLs.)

2. Launch Gmsh (e.g. double-click on the `gmsh.exe` executable on Windows).

3. Open the `models/ddm_waves/main.pro` file with the `File>Open` menu.

4. Click on `Run`.

GUI and provided example

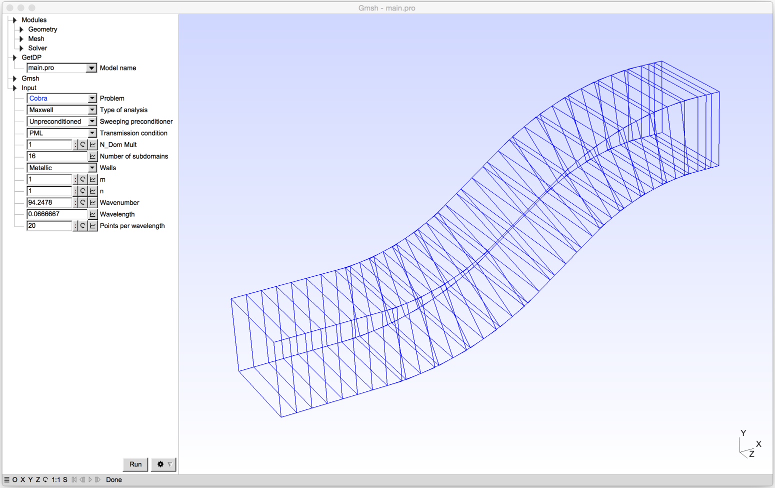
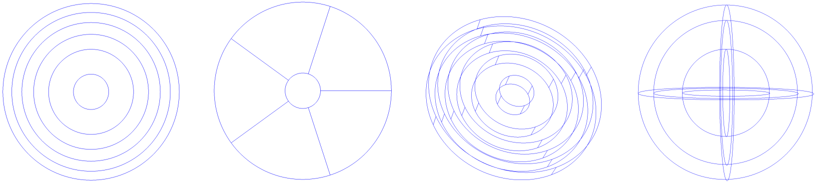
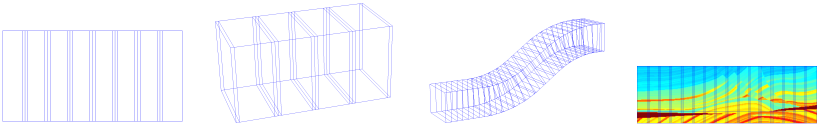


Figure: Graphical user interface of GetDDM. (Displayed test-case is `cobra`, with PML transmission conditions.)

GUI and provided example



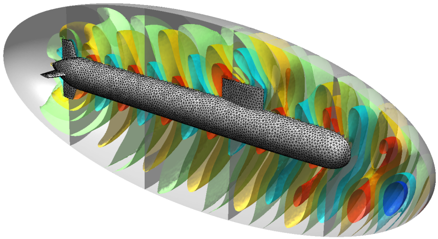
a. `circle_concentric` b. `circle_pie` c. `cylinder_concentric` d. `sphere_concentric`

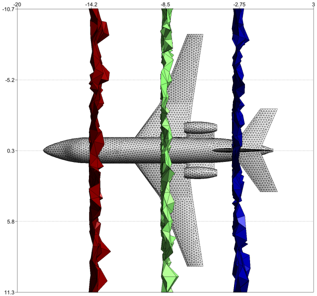


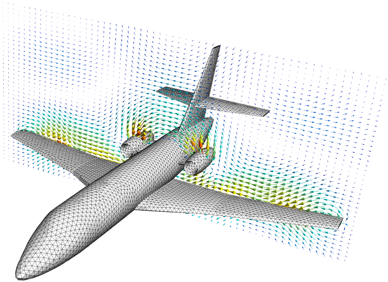
e. `waveguide2d` f. `waveguide3d` g. `cobra` h. `marmousi`

Figure: Sample models available online at `http://onelab.info/wiki/GetDDM`.

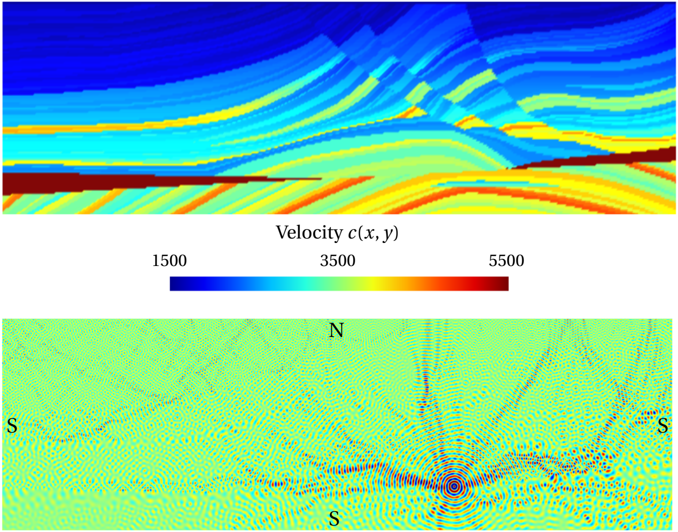
Remark: also works on non academic cases







Example: Marmousi model



Velocity profile and pressure field. Dimensions: $9192m \times 2904m$. $700Hz$ (approx. 4000 wavelengths in the domain) with $N = 358$ subdomains on 4296 CPUs: > 2.3 billions unknowns.

Other features

Other features

- PML transmission condition
- Preconditioners (sweeping, . . .)
- Overlap decomposition
- Mixte formulations
- Non-conforming meshes at interfaces

Conclusion

Open source implementation readily available for testing:

- Preprint, code and examples on `http://onelab.info/wiki/GetDDM`
- Work from laptops to massively parallel computer clusters:
  - `marmousi.pro` test-case (Helmholtz) at $700Hz$ (approx. 4000 wavelengths in the domain) with $N = 358$ subdomains on 4296 CPUs: > 2.3 billions unknowns.
  - `waveguide3d.pro` test-case (Maxwell) with $N = 3,500$ subdomains on 3,500 CPUs (cores): > 300 million unknowns.

Ongoing works

- Cross-point
- Automatic partitioning (currently, waveguide style only)
- High order finite elements (see N. Marsic talk tomorrow!)
- Other equations (elasticity, ...)

Thank you for your attention

Codes/examples/papers/preprints:

`http://onelab.info/wiki/DDM_for_Waves`

Note: I'm here all the week, do not hesitate to ask me if you want to try it (and/or have a beer)!