GetDP Workshop

GmshFEM - future direction, High Performance Computing

Anthony Royer(1)

Joint work with: Eric Béchet(2) and Christophe Geuzaine(1)

(1) Université de Liège, Institut Montefiore B28, 4000 Liège (Belgium)
(2) Université de Liège, Département Aérospatiale et Mécanique B52, 4000 Liège (Belgium)

{anthony.royer, cgeuzaine, eric.bechet}@uliege.be

April 23th, 2021

LIÈGE université

What is GmshFEM?

- GmshFEM is an open source C++ Finite Element library based on the application programming interface of Gmsh (`https://gmsh.info`).

What is GmshFEM?

- GmshFEM is an open source C++ Finite Element library based on the application programming interface of Gmsh (`https://gmsh.info`).
- Current status:
  - → arbitrarily high-order Lagrange and hierarchical basis functions
  - → scalar and vector fields (L^2, H^1, $\mathbf{H}(\mathbf{curl})$) on hybrid, curved meshes
  - → symbolic, general coupled formulations in 1D, 2D, 2D-axi and 3D
  - → real and complex arithmetic, single or double precision
  - → multi-threaded using OpenMP, linear algebra using Eigen and PETSc, eigensolver using SLEPc
  - → still only about 30k lines of C++

What is GmshFEM?

- GmshFEM is an open source C++ Finite Element library based on the application programming interface of Gmsh (`https://gmsh.info`).
- Current status:
  - → arbitrarily high-order Lagrange and hierarchical basis functions
  - → scalar and vector fields (L^2, H^1, $\mathbf{H}(\mathbf{curl})$) on hybrid, curved meshes
  - → symbolic, general coupled formulations in 1D, 2D, 2D-axi and 3D
  - → real and complex arithmetic, single or double precision
  - → multi-threaded using OpenMP, linear algebra using Eigen and PETSc, eigensolver using SLEPc
  - → still only about 30k lines of C++

Public Git repository: `https://gitlab.onelab.info/gmsh/fem`

This talk

FEM

1. Strategic choices
2. Simple example:
   - → `Domain` class
   - → `Function` class
   - → `Field` class
   - → `Formulation` class
   - → Post-processing functions
3. Assembly algorithm
4. Parallel efficiency
5. Conclusion and perspectives

The design

The GmshFEM library is designed:

- to be **fast** and **scalable** $\Rightarrow$ multi-core CPUs with SIMD instruction sets.

The design

The GmshFEM library is designed:

- to be **fast** and **scalable** $\Rightarrow$ multi-core CPUs with SIMD instruction sets.
- to be **light** $\Rightarrow$ easy to maintain and to integrate in third-part projects.

Strategic choices

The design

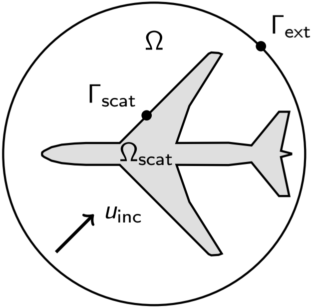
The GmshFEM library is designed:

- to be **fast** and **scalable** $\Rightarrow$ multi-core CPUs with SIMD instruction sets.
- to be **light** $\Rightarrow$ easy to maintain and to integrate in third-part projects.
- to be **user-friendly** with a **symbolic high-level description** of weak formulations

  $\rightarrow$ problems defined in a natural mathematical manner

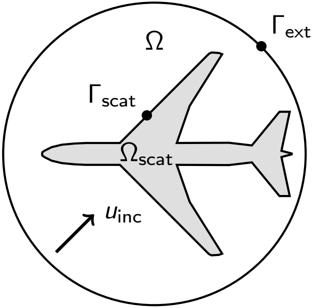
  $\rightarrow$ amenable to scripting without pre- or re-compilation.

Simple example



2D time-harmonic acoustic scattering

$$\begin{cases} (\Delta + k^2)u = 0 & \text{in } \Omega, \\ u = -u_{\text{inc}} & \text{on } \Gamma_{\text{scat}}, \\ \partial_{\mathbf{n}} u + iku = 0 & \text{on } \Gamma_{\text{ext}}, \end{cases}$$



2D time-harmonic acoustic scattering

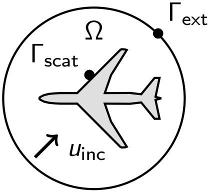
$$\begin{cases} (\Delta + k^2)u &= 0 & \text{in } \Omega, \\ u &= -u_{\text{inc}} & \text{on } \Gamma_{\text{scat}}, \\ \partial_{\mathbf{n}} u + iku &= 0 & \text{on } \Gamma_{\text{ext}}, \end{cases}$$

The weak form

The associated weak form is: Find u in $H^1(\Omega)$ such that

$$\int_{\Omega} (k^2 u\, v - \mathbf{grad}\, u \cdot \mathbf{grad}\, v)\, \mathrm{d}\Omega - \int_{\Gamma_{\mathbf{ext}}} iku\, v\, \mathrm{d}\Gamma_{\text{ext}} = 0$$

holds for every test function $v \in H^1(\Omega)$.

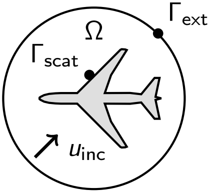


All the data related to the geometry, the mesh and the topology is manipulated with the help of `Domain` objects.

- Based on the notion of physical group in Gmsh:
  - → created by a (dim, tag) integer pairs: e.g. `domain::Domain omega(3,3);`
  - → created by a physical name: e.g. `domain::Domain omega("omega");`
- Binary operations of set algebra implemented through operators overloading:
  - → Union: e.g. `domain::Domain union = domain1 | domain2;`
  - → Intersection: e.g. `domain::Domain intersection = domain1 & domain2;`
  - → Complement: e.g. `domain::Domain complement = ~domain1;`

```
domain::Domain omega(3,3), gammaScat(2,1), gammaExt(2,2);
```

All symbolic mathematical expressions are managed with the help of `Function` classes (e.g. `ScalarFunction`, `VectorFunction` or `TensorFunction`).



- Manage scalar, vector or tensor functions and operations between them.
- Usual mathematical functions are readily available (e.g. `sin`, `sqrt`, ...).
- New functions can be added without modifying the library, going as far as hard-coding the full terms necessary for e.g. assembling complex weak formulation or post-processing computations.

$$u_{\text{inc}} = e^{-ikx} = \cos(kx) - i\sin(kx)$$

```
function::ScalarFunction<std::complex<double>> duInc =
  function::cos(k * function::x<std::complex<double>>()) -
  im * function::sin(k * function::x<std::complex<double>>());
```

The `Field` class is designed to store information about a finite element field and its associated discrete function space.

$$\int_\Omega (k^2 u\, v - \mathbf{grad}\, u \cdot \mathbf{grad}\, v)\, \mathrm{d}\Omega - \int_{\Gamma_{\mathbf{ext}}} iku\, v\, \mathrm{d}\Gamma_{\mathrm{ext}} = 0$$

- Is represented by any differential form and is templated on a scalar type `T_Scalar` that could be any real or complex arithmetic field in both single or double floating points precisions:
  - → `field::Field< T_Scalar, form::Form0 > form0Field;`
  - → `field::Field< T_Scalar, form::Form1 > form1Field;`
  - → `field::Field< T_Scalar, form::Form2 > form2Field;`
  - → `field::Field< T_Scalar, form::Form3 > form3Field;`
- Is constructed with a name, a domain of definition, a function space and an order of interpolation if needed.

```
field::Field<std::complex<double>, form::Form0>
  u("u", omega|gammaScat|gammaExt, functionSpaceH1::HierarchicalH1, 6);
```

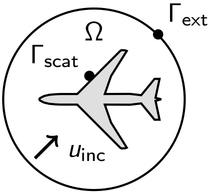
The Formulation stores the symbolic representation of the weak formulation of the problem, and can evaluate linear and bilinear forms, store the corresponding matrix systems, and request their solution through external linear algebra packages.

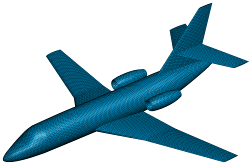
$$\int_\Omega (k^2 u\, v - \mathbf{grad}\, u \cdot \mathbf{grad}\, v)\, \mathrm{d}\Omega - \int_{\Gamma_{\mathbf{scat}}} \partial_{\mathbf{n}} u_{\mathrm{inc}}\, v\, \mathrm{d}\Gamma_{\mathrm{scat}} + \int_{\Gamma_{\mathbf{ext}}} iku\, v\, \mathrm{d}\Gamma_{\mathrm{ext}} = 0$$

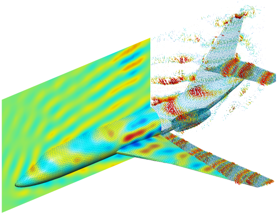
```
problem::Formulation<std::complex<double>> formulation("helmholtz");

using namespace gmshfem::equation;

formulation.galerkin(- grad(dof(u)), grad(tf(u)), omega, "Gauss12");
formulation.galerkin(k * k * dof(u), tf(u), omega, "Gauss12");
formulation.galerkin(- im * k * dof(u), tf(u), gammaExt, "Gauss12");

formulation.pre();
formulation.assemble();
formulation.solve();
```

Once a problem is solved, fields can be post-processed using a variety of operations.

- Fields, or any function that involves fields, can be saved easily.
- Integrals can be computed in one line of code.



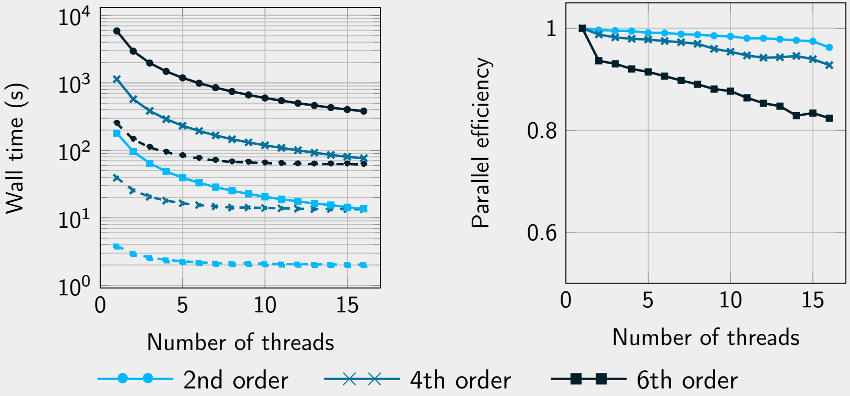




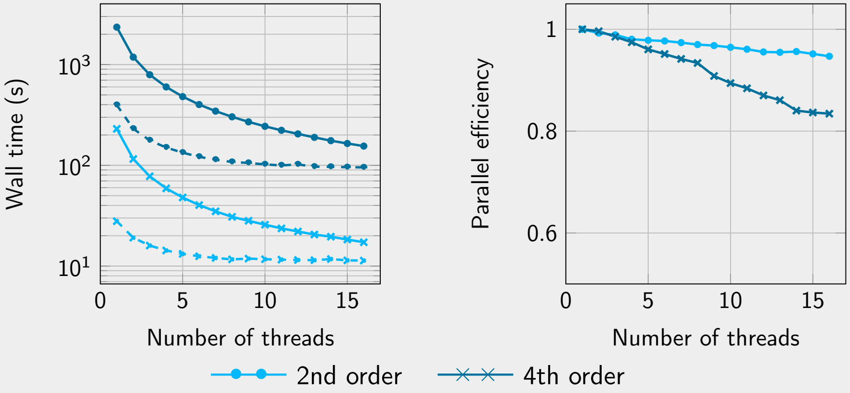
```
post::save(u, omega, "u");
post::save(function::grad(u), omega, "grad_u");
```

Parallel efficiency

Acoustic waves

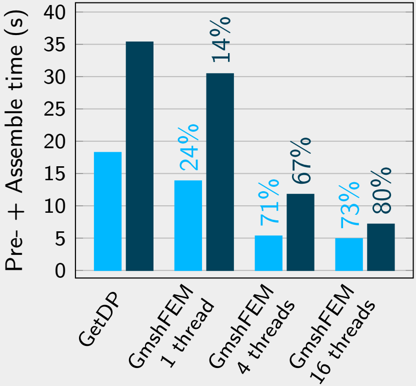


Electromagnetic waves



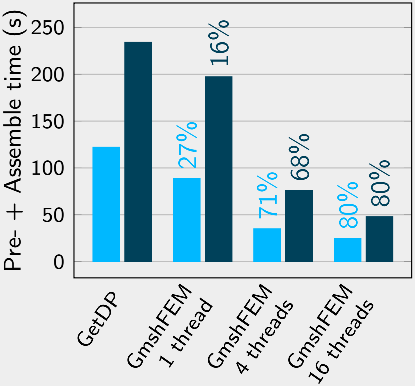
Small problem

- Acoustic wave scattering problem.
- **1st order**: 219,642 dofs
- **2nd order** (curved) : 877,312 dofs

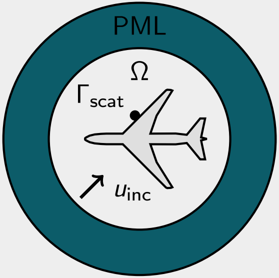


Bigger problem

- Acoustic wave scattering problem.
- **1st order**: 1,366,438 dofs
- **2nd order** (curved) : 5,462,612 dofs

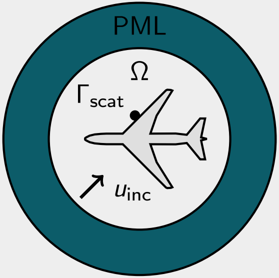


Adding a Perfectly Matched Layer (PML)



- Avoid reflections on the exterior boundary
- Same purpose as the *iku* term
- It's like a material property that dissipates the wave

Adding a Perfectly Matched Layer (PML)



- Avoid reflections on the exterior boundary
- Same purpose as the *iku* term
- It's like a material property that dissipates the wave

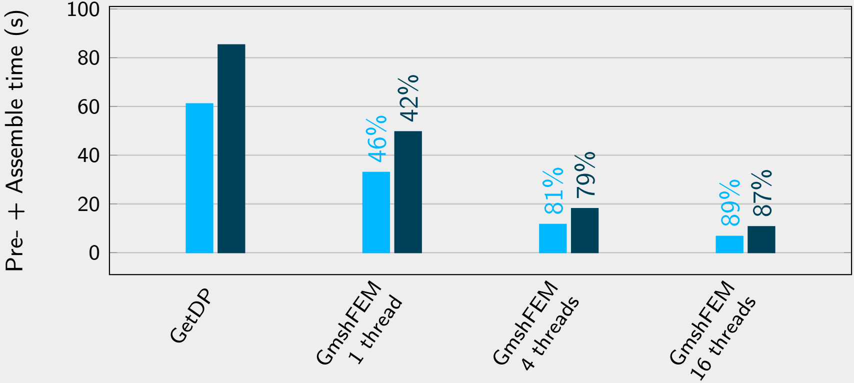
Weak form

$$\int_\Omega \left(k^2 E\, u\, v - \mathbb{D}\, \mathbf{grad}\, u \cdot \mathbf{grad}\, v\right) \mathrm{d}\Omega - \cancel{\int_{\Gamma_{\text{ext}}} iku\, v\, \mathrm{d}\Gamma_{\text{ext}}} = 0$$

where E is a scalar and $\mathbb{D}$ is a tensor.

Small problem with PML (material property)

- Acoustic wave scattering problem with PML.
- **1st order**: 219,642 dofs
- **2nd order** (curved) : 877,312 dofs



Conclusion

- GmshFEM is a new, open source C++ finite element library based on the Gmsh API.

Conclusion

- GmshFEM is a new, open source C++ finite element library based on the Gmsh API.
- Excellent parallel efficiency for high-order scalar and vector problems.

Conclusion and perspectives

Conclusion

- GmshFEM is a new, open source C++ finite element library based on the Gmsh API.
- Excellent parallel efficiency for high-order scalar and vector problems.
- Problem definitions are close to GetDP ones.

Conclusion

- GmshFEM is a new, open source C++ finite element library based on the Gmsh API.
- Excellent parallel efficiency for high-order scalar and vector problems.
- Problem definitions are close to GetDP ones.
- Currently used to solve extreme-scale time-harmonic finite element problems in combination with an optimized DDM solver.

Perspectives

- Offloading of linear algebra on GPUs.
- Python and Julia bindings for scripting without re-compilation.

Conclusion and perspectives

Conclusion

- GmshFEM is a new, open source C++ finite element library based on the Gmsh API.
- Excellent parallel efficiency for high-order scalar and vector problems.
- Problem definitions are close to GetDP ones.
- Currently used to solve extreme-scale time-harmonic finite element problems in combination with an optimized DDM solver.

Perspectives

- Offloading of linear algebra on GPUs.
- Python and Julia bindings for scripting without re-compilation.

A. Royer, É. Béchet, and C. Geuzaine, "Gmsh-Fem: An Efficient Finite Element Library Based On Gmsh", presented at the 14th WCCM-ECCOMAS Congress, 2021, pp. 1–13.