ViennaFEM 1.0.0

User Manual

Institute for Microelectronics
Gußhausstraße 27-29 / E360
A-1040 Vienna, Austria/Europe

Institute for Analysis and Scientific Computing
Wiedner Hauptstraße 8-10 / E101
A-1040 Vienna, Austria/Europe

Author and Project Head:

Karl Rupp

Institute for Microelectronics
Vienna University of Technology
Gußhausstraße 27-29 / E360
A-1040 Vienna, Austria/Europe

Phone +43-1-58801-36001
FAX +43-1-58801-36099
Web http://www.iue.tuwien.ac.at

Institute for Analysis and Scientific Computing
Vienna University of Technology
Wiedner Hauptstraße 8-10 / E101
A-1040 Vienna, Austria/Europe

Phone +43-1-58801-10101
Web http://www.asc.tuwien.ac.at

Contents

Introduction 1

1 Installation 3

1.1 Dependencies . . . 3

1.2 Generic Installation of ViennaFEM . . . 3

1.3 Building the Examples and Tutorials . . . 4

2 The Finite Element Solver 6

2.1 Grid Setup . . . 6

2.2 Boundary Conditions . . . 6

2.3 PDE Specification . . . 7

2.4 Running the FE Solver . . . 8

2.5 Postprocessing Results . . . 9

2.6 LaTeX Simulation Protocol . . . 9

Change Logs 10

License 11

Bibliography 12

Introduction

A considerable amount of codes for the Finite Element Method (FEM) are freely available, e.g. deal.ii [1], Getfem++ [2], the FEniCS project [3], or Sundance [4]. In addition to different functionalities provided by the packages, e.g. dimensionality of the simulation domain, supported set of basis functions, etc., different software design approaches are also persued.

The main driver for the development of `ViennaFEM` is the observation that most packages do not preserve the weak formulation at source code level. Moreover, existing finite element software is often monolithic, making a reuse of components such as the grid handling for other purposes difficult or even impossible. This is particularly a concern in the engineering environment surrounding the author, where FEM is just one possible numerical method among many others. Therefore, it is not affordable to redevelop code that is a-priori not specific to FEM, for instance the grid handling.

To preserve productivity, `ViennaFEM` follows a library-centric approach: Components are kept separate to the largest extent possible, i.e. they are developed as individual libraries. This separation is achieved by the use of template metaprogramming techniques [5] and generic C++ programming tricks [6].

What `ViennaFEM` Currently Is

`ViennaFEM 1.0.0` is still a proof-of-concept. In particular, it shows that a symbolic math kernel (`ViennaMath`) can be combined with a grid handling library (`ViennaGrid`) by a data storage abstraction library (`ViennaData`) in such a way that each of these libraries can be developed separately and even be used in entirely different contexts. Thus, `ViennaFEM 1.0.0` already supports:

- Automatic derivation of the weak formulation for second-order PDEs
- Dimension-independent programming (1d, 2d, 3d)
- Multiple FEM runs on the same mesh for different equations and boundary conditions
- Mesh refinement for simplex-cells
- Support for graphics processing units in the linear solver step

Last, but not least, a unique feature of `ViennaFEM` is the LaTeX logger, which protocols all the high-level math carried out inside `ViennaFEM` during the simulation run.

What ViennaFEM Is Not (Yet)

ViennaFEM is certainly not the general answer to the numeral solution of second-order partial differential equations[1].

The first release of ViennaFEM cannot outnumber the features of other packages, which have been developed for more than a decade already. A roadmap of features scheduled for the next releases is as follows:

- Compiletime calculation of element matrices
- Arbitrary-order basis functions from different families
- Inhomogeneous Neumann boundary conditions
- Automatic time-discretization for time-dependent problems
- Automatic linearization of nonlinear problems
- Automatic deduction of error estimators
- Better support for coupled systems

Several of these planned features are only possible thanks to preserving the weak formulation in code. The library-centric design of ViennaFEM emphasizing well-defined interfaces between the different libraries is expected to aid in reaching these goals with shorter overall development effort compared to monolithic approaches.

[1] Neither is '42'.

Chapter 1

Installation

This chapter shows how ViennaFEM can be integrated into a project and how the examples are built. The necessary steps are outlined for several different platforms, but we could not check every possible combination of hardware, operating system, and compiler. If you experience any trouble, please write to the mailing list at

viennafem-support@lists.sourceforge.net

1.1 Dependencies

- A recent C++ compiler (e.g. GCC version 4.2.x or above and Visual C++ 2005 or above are known to work)
- ViennaCL [7], version 1.1.2 or above. To make ViennaFEM self-contained, a copy of the ViennaCL sources is available in the viennacl/ folder.
- ViennaData [8], version 1.0.1 or above. To make ViennaFEM self-contained, a copy of the ViennaData sources is available in the viennadata/ folder.
- ViennaGrid [9], version 1.0.0 or above. To make ViennaFEM self-contained, a copy of the ViennaGrid sources is available in the viennagrid/ folder.
- ViennaMath [10], version 1.0.0 or above. To make ViennaFEM self-contained, a copy of the ViennaMath sources is available in the viennamath/ folder.
- A recent version of the Boost libraries [11] for Boost.uBLAS. Version 1.42 is known to work, but other versions should work as well.
- [Optional] CMake [12] as build system for building the examples and the standalone executable.

1.2 Generic Installation of ViennaFEM

Since ViennaFEM is a header-only library, it is sufficient to copy the viennafem/, viennacl/, viennadata/, viennagrid/ and viennamath/ folders either into your project folder or to

File	Purpose
basic_poisson.cpp	Demonstrates the symbolic math operations in the background when solving the Poisson equation
poisson_1d.cpp	One-dimensional solution of the Poisson equation
poisson_2d.cpp	Two-dimensional solution of the Poisson equation on a triangular grid
poisson_2d_rect.cpp	Two-dimensional solution of the Poisson equation on a rectangular grid
poisson_3d.cpp	One-dimensional solution of the Poisson equation on a tetrahedral grid
poisson_3d_hex.cpp	One-dimensional solution of the Poisson equation on a hexahedral grid
sshape_2d.cpp	Two-dimensional solution of the Poisson equation on an S-shaped triangular grid
sshape_3d.cpp	Three-dimensional solution of the Poisson equation on an S-shaped tetrahedral grid

Table 1.1: Overview of the examples in the examples/tutorial/ folder

your global system include path. On Unix based systems, this is often /usr/include/ or /usr/local/include/.

On Windows, the situation strongly depends on your development environment. Please consult the documentation of your compiler or development environment on how to set the include path correctly. The include paths in Visual Studio are usually something like C:\Program Files\Microsoft Visual Studio 9.0\VC\include and can be set in Tools -> Options -> Projects and Solutions -> VC++-Directories.

1.3 Building the Examples and Tutorials

An overview of available examples and their purpose is given Tab. 1.1. For building the examples, we suppose that CMake is properly set up on your system. In the following, instructions on how to build the examples on different platforms are given.

1.3.1 Linux

To build the examples, open a terminal and change to:

```
$> cd /your-ViennaFEM-path/build/
```

Execute

```
$> cmake ..
```

to obtain a Makefile and type

```
$> make
```

to build the examples. If desired, one can build each example separately instead:

```
$> make poisson_1d        #builds the 'poisson_1d' tutorial
$> make sshape_3d         #builds the 'sshape_3d' tutorial
```

Speed up the building process by using jobs, e.g. `make -j4`.

1.3.2 Mac OS X

The tools mentioned in Section 1.1 are available on Macintosh platforms too. For the `GCC` compiler the Xcode [13] package has to be installed. To install `CMake`, external portation tools such as Fink [14], DarwinPorts [15], or MacPorts [16] have to be used.

The build process of `ViennaFEM` is similar to Linux.

1.3.3 Windows

In the following the procedure is outlined for `Visual Studio`: Assuming that an `OpenCL` SDK and `CMake` is already installed, Visual Studio solution and project files can be created using `CMake`:

- Open the `CMake` GUI.
- Set the `ViennaFEM` base directory as source directory.
- Set the `build/` directory as build directory.
- Click on 'Configure' and select the appropriate generator (e.g. `Visual Studio 9 2008`).
- Click on 'Configure' again.
- Click on 'Generate' in order to let `CMake` generate the project files for you.
- The project files can now be found in the `ViennaFEM` build directory, where they can be opened and compiled with Visual Studio (provided that the include and library paths are set correctly, see Sec. 1.2).

Chapter 2

The Finite Element Solver

In this chapter all steps required for solving a PDE using the finite element method in ViennaFEM are discussed. Basically, the discussion follows the tutorials in examples/tutorial/. Familiarity with the very basics of the finite element method is expected. However, since ViennaFEM aims at hiding unimportant low-level details from the user, no detailed knowledge is required.

2.1 Grid Setup

The first step is to set up the discrete grid using ViennaGrid, which is accomplished by reading the mesh from a file. File readers for the VTK format [17, 18] and the Netgen legacy format [19] are provided by ViennaGrid. The domain type is instantiated by first retrieving the domain type, and then by instantiating the domain as usual:

```
//retrieve domain type:
typedef viennagrid::config::triangular_2d                ConfigType;
typedef viennagrid::result_of::domain<ConfigType>::type  DomainType;

DomainType my_domain;
viennagrid::io::netgen_reader my_reader;
my_reader(my_domain, "../examples/data/square512.mesh");
```

Here, the Netgen file reader is used to read the mesh 'square512.mesh' included in the ViennaFEM release.

The easiest way of generating meshes for use within ViennaFEM is to use Netgen [19], which also offers a graphical user interface.

2.2 Boundary Conditions

The next step is to set boundary conditions for the respective partial differential equation. Only Dirichlet boundary conditions and homogeneous Neumann are supported in ViennaFEM 1.0.0. Inhomogeneous Neumann boundary conditions as well as Robin-type boundary conditions are scheduled for future releases.

Boundary conditions are imposed by interpolation at boundary vertices. Using `ViennaGrid`, this is achieved by iterating over all vertices and calling the function `set_dirichlet_boundary()` from `ViennaFEM` for the respective vertices. For example, to specify inhomogeneous Dirichlet boundary conditions $= 1$ for all vertices with x-coordinate equal to zero, the required code is:

```
typedef viennagrid::result_of::ncell_range<DomainType, 0>::type
                                                        VertexContainer;
typedef viennagrid::result_of::iterator<VertexContainer>::type
                                                        VertexIterator;

VertexContainer vertices = viennagrid::ncells<0>(my_domain);
for (VertexIterator vit = vertices.begin();
     vit != vertices.end();
     ++vit)
{
  if ( (*vit)[0] == 0.0 )
   viennafem::set_dirichlet_boundary(*vit, 1.0, 0);
}
```

The first argument to `set_dirichlet_boundary` is the vertex, the second argument is the Dirichlet boundary value, and the third argument is the simulation ID. The simulation ID allows to distinguish between different finite element solutions of possibly different partial differential equations and should be chosen uniquely for each solution to be computed. However, it is not required to pass a simulation ID to `set_dirichlet_boundary`, in which case it defaults to zero.

The identification of boundary vertices may not be possible in the cases where the simulation domain is complicated. In such cases, the indices of boundary vertices may be either read from a separate file, or boundary nodes are flagged in the VTK file already. In addition, boundary and/or interface detection functionality from `ViennaGrid` can be used.

2.3 PDE Specification

The actual PDE to be solved is specified using the symbolic math library `ViennaMath`. Either the strong form or the weak form can be specified. For the case of the Poisson equation

$$\Delta u = -1 \,, \tag{2.1}$$

the representation using `ViennaMath` is

```
function_symbol u;
equation poisson_eq = make_equation( laplace(u), -1);
```

where all functions and types reside in the namespace `viennamath`.

It is also possible to supply an inhomogeneous right hand side f instead of a constant value -1. In `ViennaMath 1.0.0`, f is required to be constant within each cell of the mesh. The values of f are accessed via objects of type `cell_quan<T>`, where `T` is the cell type of the `ViennaGrid` domain:

```
 viennafem::cell_quan<CellType>  f;
 f.wrap_constant( data_key );
```

Here, `data_key` denotes the key to be used in order to access the values of f of type **`double`** from the respective cell.

Have a look at `examples/tutorial/poisson_cellquan_2d` for an example of using `cell_quan`.

2.4 Running the FE Solver

After the boundary conditions and the partial differential equation are specified, the finite element solver can be started. This is accomplished in three steps:

- Instantiate the PDE assembler object, the system matrix and the load vector
- Run the assembly
- Solve the resulting system of linear equations

The first step is accomplished by the lines

```
viennafem::pde_assembler                              fem_assembler;
boost::numeric::ublas::compressed_matrix<double> system_matrix;
boost::numeric::ublas::vector<double>             load_vector;
```

Other matrix and vector types can also be used as long as they offer access to their entries using **`operator`**`()` as well as `size()` and `resize()` member functions.

The second step, namely the assembly of the linear system of equations, is triggered by passing the PDE system, the domain, the system matrix and the load vector objects to the functor interface:

```
fem_assembler(viennafem::make_linear_pde_system(poisson_eq, u),
              my_domain,
              system_matrix,
              load_vector);
```

The convenience function `make_linear_pde_system()` sets up a system given by the Poisson equation defined in the previous section, and specifies the unknown function to be `u`. By default, a linear finite element simulation using the simulation ID `0` is triggered. Additional options can be specified for the PDE system using the optional third parameter. In `ViennaFEM 1.0.0`, three parameters can be specified: The first parameter is the simulation ID, while the second and third parameter denote the basis function tag used for the trial space and the test space, respectively. Currently, only Lagrange basis functions are supported using the tag `lagrange_tag<order>`, where the parameter `order` denotes the polynomial order along cell edges. Note that only linear basis functions are supported at the moment. Thus, the previous call to `make_linear_pde_system` is equivalent to

```
make_linear_pde_system(poisson_eq, u,
                       make_linear_pde_options(0, lagrange_tag<1>(),
                                                  lagrange_tag<1>()) ),
```

The third step is to solve the assembled system of linear equations. For this purpose, the solvers in `ViennaCL` are used. For example, the iterative conjugate gradient solver is launched via

```
VectorType pde_result = solve(system_matrix,
                              load_vector,
                              cg_tag() );
```

where `solve()` and `cg_tag` reside in namespace `viennacl::linalg`. An overview of solvers and preconditioners available in `ViennaCL` can be found in the `ViennaCL` manual in `doc/viennacl.pdf`.

2.5 Postprocessing Results

For a visualization of results, a VTK exporter is included in `ViennaFEM` in the namespace `viennafem::io` and defined in the header file `viennafem/io/vtk_writer.hpp`

```
write_solution_to_VTK_file(pde_result, "filename", my_domain, 0);
```

The first argument is the computed solution vector, the second is the filename, the third is the domain object, and the forth parameter is the simulation ID (optional, defaults to 0).

2.6 LaTeX Simulation Protocol

`ViennaFEM` by default writes a protocol named `protocol_<ID>.tex`, where `<ID>` is replaced by the respective simulation ID. This file can directly be processed with a LaTeX compiler and prints a summary containing the strong formulation, the weak formulation, the basis functions used, etc. The document is compiled on Unix-based systems using either the standard LaTeX toolchain

```
$> latex  protocol_0.tex
$> dvips  protocol_0.dvi  #optional
$> ps2pdf protocol_0.ps   #optional
```

or using PDFLaTeX:

```
$> pdflatex protocol_0.tex
```

Any LaTeX-environment can be used on Windows-based systems. The resulting PDF file can then be viewed with your favorite PDF viewer.

Change Logs

Version 1.0.0

First release

License

Bibliography

[1] "Deal.ii." [Online]. Available: http://www.dealii.org/

[2] "Getfem++." [Online]. Available: http://home.gna.org/getfem/

[3] "Fenics." [Online]. Available: http://www.fenics.org/

[4] "Sundance 2.3." [Online]. Available: http://www.math.ttu.edu/~klong/Sundance/html/

[5] D. Vandevoorde and N. M. Josuttis, *C++ Templates*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[6] A. Alexandrescu, *Modern C++ Design: Generic Programming and Design Patterns Applied*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[7] "ViennaCL." [Online]. Available: http://viennacl.sourceforge.net/

[8] "ViennaData." [Online]. Available: http://viennadata.sourceforge.net/

[9] "ViennaGrid." [Online]. Available: http://viennagrid.sourceforge.net/

[10] "ViennaMath." [Online]. Available: http://viennamath.sourceforge.net/

[11] "Boost C++ Libraries." [Online]. Available: http://www.boost.org/

[12] "CMake." [Online]. Available: http://www.cmake.org/

[13] "Xcode Developer Tools." [Online]. Available: http://developer.apple.com/technologies/tools/xcode.html

[14] "Fink." [Online]. Available: http://www.finkproject.org/

[15] "DarwinPorts." [Online]. Available: http://darwinports.com/

[16] "MacPorts." [Online]. Available: http://www.macports.org/

[17] "VTK - Visualization Toolkit." [Online]. Available: http://www.vtk.org/

[18] "VTK File Formats." [Online]. Available: http://www.vtk.org/VTK/img/file-formats.pdf

[19] "Netgen Mesh Generator." [Online]. Available: http://sourceforge.net/projects/netgen-mesher/