

PDE MODELING WITH MODELICA VIA FMI IMPORT OF HIFLOW3 C++ COMPONENTS WITH PARALLEL MULTI-CORE SIMULATIONS

Kristian Stavåker, and Peter Fritzson
{ Kristian.Stavaker, Peter.Fritzson }@liu.se
Programming Environments Laboratory
Department of Computer and Information Science
Linköping University
Sweden

Chen Song, Martin Wlotzka, and Vincent Heuveline
{ Chen.Song, Martin.Wlotzka, Vincent.Heuveline }@iwr.uni-heidelberg.de
Engineering Mathematics and Computing Laboratory
Interdisciplinary Center for Scientific Computing
University of Heidelberg
Germany

ABSTRACT

The Modelica modeling and simulation language is widely used in academia and industry to model complex, coupled dynamic systems which can be described by systems of ordinary differential equations (ODE) or differential algebraic equations (DAE). Recent work by the authors showed a way to enable partial differential equation (PDE) modeling with Modelica via functional mock-up interface (FMI) import of C++ components based on the multi-purpose finite element library HiFlow3. The finite element method (FEM) is largely used in both research and industry as a reliable technique for solving PDE problems. In contrast to methods based on language extensions or automatic semi-discretizations in space, the approach with FMI import of HiFlow3 components into Modelica requires no change to the Modelica language, enables the use of specialized PDE solvers, and it allows for full flexibility in the choice of geometry, model parameters, and space discretization between simulation runs without recompilation. However, the computationally intensive PDE solving part in this approach can form a bottleneck in the simulations. In this work, we enhance the PDE solver by using a distributed memory parallelization based on a domain decomposition. As an example application, we consider a mechanical linear elasticity problem consisting of physical forces applied on a beam. Beams, plates and shells are common elements of solid structures with a sizable quantity of application in engineering design, appearing in fuselage, ship hulls, concrete roof structure, etc. The derivation of elastic stress strain relations is a crucial point for mechanical analysis and validation, as the bending properties of the structure effects greatly the stability properties. In this work the actual beam is modeled and solved in parallel using a C++ HiFlow3 component whereas the physical force acting on the beam is modeled using Modelica. We use the OpenModelica development environment but the same approach can be adapted to other Modelica environments.

Keywords: Modelica, HiFlow3, OpenModelica, Partial Differential Equations, Finite Element Method, Functional Mock-Up Interface, Multi-Core, Parallel Computing

INTRODUCTION

This paper is a continuation of the work in [20]. We discuss numerical simulation of models that couple partial differential equations (PDEs) and differential-algebraic equations (DAEs) in the context of the Modelica modeling and simulation language [14, 5, 6]. Modelica originated around the idea of solving complex coupled dynamic systems, which can be described by systems of ordinary differential equations (ODE) or DAE. Up to now, there is only limited support for working with PDEs, despite the fact that the number of Modelica users in academia and in industry has lately grown significantly.

Some attempts of incorporating PDE support into Modelica are described in [11], [12] and in Chapter 8 of [5]. In [18] and [17] two different approaches are investigated: (1) expressing the PDEs using a combination of new language constructs and a supporting Modelica PDE library using the method-of-lines; (2) exporting the PDE part to an external PDE FEM C++ tool which solves the PDE part of the total problem. Based on this work, an experimental implementation of PDE support was added to the OpenModelica [3] compiler. However, this implementation has not been maintained, even though there have recently been discussions in the OpenModelica community about re-activating these features. Only one simple PDE operator is currently in the official Modelica language specification: spatial distribution for 1D PDEs. In [10] a Modelica library with basic building blocks for solving one-dimensional PDE with spatial discretizations based on the method of lines or finite volumes is described. Although this approach is attractive due to its simplicity, it is not clear how it could be extended to higher dimensions, without increasing the complexity significantly.

In this work, we propose a way to allow for PDE modeling with Modelica by importing C++ components, written with the HiFlow3 multi-purpose finite element software [19], into Modelica using the Functional Mock-Up Interface (FMI) [15] import. FMI is a standard for model exchange and co-simulation between different tools. FMI supports only C but with correct linking it is possible to execute with C++ code. We use the OpenModelica [3] development environment but the same approach can be adapted to other Modelica environments. As

noted earlier, this paper is a continuation of the work in [20]. In that paper the heat distribution was computed for a copper beam. The heat source was controlled by a PID-controller. In this paper we continue to explore the Modelica-HiFlow3 coupling with parallel computations on multi-core architectures. We use a model of a steel beam with a force acting on it to demonstrate our implementation. We measure the elasticity deformation of the beam. In this work the actual membrane plate is modelled and solved in parallel using a C++ HiFlow3[13, 16] component whereas the physical force acting on the beam is modeled using Modelica.

The method described in this paper and in [20] has several advantages:

- HiFlow3 is well maintained and has strong support and capabilities for PDE modeling and solving;
- HiFlow3 and OpenModelica are free to download and use;
- The PDE structure is not lost but is maintained throughout the actual run-time simulation process. This allows for mesh refinement, solver run-time adjustments, etc.;
- It is possible to mix PDE and DAE systems in the same system setting. This is also possible in [10].

SIMULATION SCENARIO

We consider the deformation of a rectangular building element under a load. Fig.1 shows the setup of

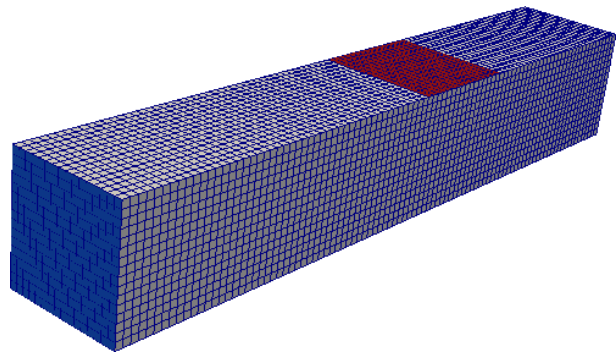


Figure 1: Geometry and computational mesh for the concrete element. The fixed front end is colored in blue, a force is acting on the red part.

the configuration. The element is fixed at both ends, and the load is modeled by an external force acting on a part of the upper boundary.

Linear Elasticity Model

The element occupies a domain $\Omega \subset \mathbb{R}^3$ with boundary $\Gamma := \partial\Omega$. The behaviour of the object subject to a force is described by means of the displacement \mathbf{u} and the stress tensor $\boldsymbol{\sigma}$. Conservation of momentum leads to the equilibrium equation

$$-\nabla \cdot ([I + \nabla \mathbf{u}] \boldsymbol{\sigma}) = \rho \mathbf{f} \quad \text{in } \Omega, \quad (1)$$

where ρ is the density of the material, \mathbf{f} is a volumetric force, and I denotes the identity matrix. In our scenario, gravity is the only volumetric force, therefore $\mathbf{f} = -g\mathbf{e}_z$.

According to Hooke's law for isotropic materials the stress tensor is related to the deformation tensor $\boldsymbol{\varepsilon}$ as

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} + \lambda\text{tr}(\boldsymbol{\varepsilon})I,$$

with material parameters μ and λ , known as the Lamé elasticity constants. Assuming small deformations, we neglect nonlinear terms in the deformation tensor resulting in the linearised form [2]

$$\boldsymbol{\varepsilon} \approx \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^\top),$$

and also the deformation gradient is simplified as

$$I + \nabla \mathbf{u} \approx I.$$

This leads to the following problem formulation:

Find a function $\mathbf{u} : \Omega \rightarrow \mathbb{R}^3$ as the solution of

$$-\mu \nabla \cdot (\nabla \mathbf{u} + \nabla \mathbf{u}^\top) - \lambda \nabla (\nabla \cdot \mathbf{u}) = \rho \mathbf{f} \quad \text{in } \Omega, \quad (2a)$$

$$\mathbf{u} = 0 \quad \text{on } \Gamma_0, \quad (2b)$$

$$\left[\mu (\nabla \mathbf{u} + \nabla \mathbf{u}^\top) + \lambda (\nabla \cdot \mathbf{u}) I \right] \cdot \mathbf{n} = p \quad \text{on } \Gamma_1, \quad (2c)$$

$$\left[\mu (\nabla \mathbf{u} + \nabla \mathbf{u}^\top) + \lambda (\nabla \cdot \mathbf{u}) I \right] \cdot \mathbf{n} = 0 \quad \text{on } \Gamma_f. \quad (2d)$$

The homogeneous Dirichlet boundary condition (2b) fixes the beam at its ends. The load on the beam acts as a pressure p through the Neumann boundary condition (2c), and the homogeneous Neumann condition (2d) is imposed on the free part of the boundary.

Variational Formulation

A well-established method for numerically solving PDEs is the finite element method, which is based on a variational formulation of the system (2).

Assuming that there is a classical solution $\mathbf{u} \in C^2(\Omega, \mathbb{R}^3)$ of problem (2), equation (2a) is multiplied by a test function $\mathbf{v} \in C_0^\infty(\Omega)$ and integrated over Ω :

$$\begin{aligned} & -\mu \int_{\Omega} \left[\nabla \cdot (\nabla \mathbf{u} + \nabla \mathbf{u}^\top) \right] \cdot \mathbf{v} \, dx \\ & -\lambda \int_{\Omega} \left[\nabla (\nabla \cdot \mathbf{u}) \right] \cdot \mathbf{v} \, dx = \int_{\Omega} \rho \mathbf{f} \cdot \mathbf{v} \, dx \end{aligned}$$

The divergence theorem yields (note that $\nabla \mathbf{v} = \nabla \mathbf{v}^\top$)

$$\begin{aligned} & -\mu \int_{\Omega} \left[\nabla \cdot (\nabla \mathbf{u} + \nabla \mathbf{u}^\top) \right] \cdot \mathbf{v} \, dx \\ & = \frac{\mu}{2} \int_{\Omega} (\nabla \mathbf{u} + \nabla \mathbf{u}^\top) : (\nabla \mathbf{v} + \nabla \mathbf{v}^\top) \, dx \end{aligned}$$

and

$$-\lambda \int_{\Omega} \left[\nabla (\nabla \cdot \mathbf{u}) \right] \cdot \mathbf{v} \, dx = \lambda \int_{\Omega} (\nabla \cdot \mathbf{u}) (\nabla \cdot \mathbf{v}) \, dx,$$

where the boundary integrals are omitted since they vanish as $\mathbf{v} = 0$ on $\partial\Omega$. This leads to

$$\begin{aligned} & \int_{\Omega} \frac{\mu}{2} (\nabla \mathbf{u} + \nabla \mathbf{u}^\top) : (\nabla \mathbf{v} + \nabla \mathbf{v}^\top) \\ & + \lambda (\nabla \cdot \mathbf{u}) (\nabla \cdot \mathbf{v}) \, dx = \int_{\Omega} \rho \mathbf{f} \cdot \mathbf{v} \, dx. \quad (3) \end{aligned}$$

For equation (3) to be well-defined, weaker regularity properties of \mathbf{u} and \mathbf{v} than in the classical context are sufficient. The problem can be formulated in terms of the Lebesgue space $[L^2(\Omega)]^3$ of square-integrable functions defined on Ω and with image in \mathbb{R}^3 , and the Sobolev space $[H^1(\Omega)]^3$ of functions in $[L^2(\Omega)]^3$ with square-integrable weak derivatives. We define the weak solution space

$$V := \{ \mathbf{u} \in [H^1(\Omega)]^3 : \mathbf{u} = 0 \text{ on } \Gamma_0 \},$$

the bilinear form

$$\begin{aligned} & a : [H^1(\Omega)]^3 \times [H^1(\Omega)]^3 \rightarrow \mathbb{R}, \\ & a(\mathbf{u}, \mathbf{v}) := \int_{\Omega} \frac{\mu}{2} (\nabla \mathbf{u} + \nabla \mathbf{u}^\top) : (\nabla \mathbf{v} + \nabla \mathbf{v}^\top) \\ & + \lambda (\nabla \cdot \mathbf{u}) (\nabla \cdot \mathbf{v}) \, dx, \end{aligned}$$

and the linear form

$$l : [H^1(\Omega)]^3 \rightarrow \mathbb{R},$$

$$l(v) := \int_{\Omega} \rho f \cdot v dx + \int_{\Gamma_0} p \cdot v ds.$$

Note that the bilinear form a is symmetric, continuous and V -elliptic. Now we can state the variational formulation of problem (2):

Find $u \in V$ as the solution of

$$a(u, v) = l(v) \quad \forall v \in V. \quad (4)$$

This variational formulation admits a unique solution, which is called the weak solution of the elasticity problem.

Finite Element Discretization

Let $T_h := \{K_1, \dots, K_N\}$ be a triangulation of Ω with N tetrahedron cells K_i ($i = 1, \dots, N$). We define the finite element space of piecewise linear functions

$$V_h := \{v \in V : v|_K \text{ is linear } (K \in T_h)\}.$$

V_h has the finite dimension $n := \dim(V_h)$. We give the problem formulation for a conforming finite element approximation of (4):

Find $u_h \in V_h$ as the solution of

$$a(u_h, v_h) = l(v_h) \quad \forall v_h \in V_h. \quad (5)$$

Let $\{\varphi_1, \dots, \varphi_n\}$ be a basis of V_h . We define the ansatz function as :

$$u_h(x) := \sum_{i=1}^n x_i \varphi_i(x)$$

with coefficients $x_i \in \mathbb{R}$ and insert it into (5), yielding

$$\sum_{i=1}^n x_i a(\varphi_i, \varphi_j) = b(\varphi_j) \quad (j = 1, \dots, n).$$

This can be written as the linear system

$$Ax = b, \quad (6)$$

where

$$A := \left(a(\varphi_j, \varphi_i) \right)_{i,j=1,\dots,n} \in \mathbb{R}^{n \times n}$$

is the stiffness matrix and

$$b := \left(l(\varphi_i) \right)_{i=1,\dots,n} \in \mathbb{R}^n$$

is the load vector. As the stiffness matrix is symmetric and positive definite [2], we employ the Conjugate Gradient (CG) method [7] for solving (6).

COUPLED IMPLEMENTATION

More details of the coupled simulation setup can be found in [20].

The Modelica Modeling and Simulation Language

Modelica is a language for equation-based object-oriented mathematical modeling which is being developed and standardized through an international effort in the Modelica Association [14]. The equation parts of Modelica requires a lot of the compiler developer: knowledge in compiler construction, symbolic manipulation of equations and associated mathematics, as well as knowledge of numerical mathematics. The simulation run-time system is also an important part and can be complex; various solver techniques for solving the differential equations can be applied. Modelica allows high-level concepts such as object-oriented modeling and component composition. Multi-domain modeling is also possible in Modelica with the possibility of combining model components from a variety of domains within the same application. There exist several mature and well-maintained Modelica development environments, such as Dymola, OpenModelica, MapleSim, Wolfram SystemModeler, Simulation X, and JModelica.org. See [5] and [6] for more information on the Modelica language.

The OpenModelica Environment

OpenModelica is a modeling and simulation environment, which is developed and supported by an international consortium, the Open Source Modelica Consortium (OSMC) [3]. This effort includes an open-source implementation of a Modelica compiler, a simulator and a development environment for research, education and industrial purposes.

The HiFlow3 Finite Element Library

HiFlow3 [19, 1] is a multi-purpose finite element software providing powerful tools for efficient and accurate solution of a wide range of problems modeled by partial differential equations (PDEs). Based on object-oriented concepts and the full capabilities of C++ the HiFlow3 project follows a modular and generic approach for building efficient parallel numerical solvers. It provides highly capable modules

dealing with mesh setup, finite element spaces, degrees of freedom, linear algebra routines, numerical solvers, and output data for visualization. Parallelism - as the basis for high performance simulations on modern computing systems - is introduced at two levels: coarse-grained parallelism by means of distributed grids and distributed data structures, and fine-grained parallelism by means of platform-optimized linear algebra back-ends.

HiFlow3-based PDE Component

HiFlow3 is a multi-purpose finite element method orientated mathematical library, it is well designed for modular and generic approach. Considering our application, HiFlow3's modules provide us an efficient computing platform [1] for things such as mesh setting, finite element function spaces, linear algebra structure, numerical solvers, etc. HiFlow3 also provides a well-defined parallel environment which allows to build up our elasticity solver as a robust component that we can execute on large scale machines with respect to high performance computing (HPC).

Modelica Model

Our (relatively simple) Modelica model is shown in Listing 1. It contains the PDE component and the force variable. The variables *hfBlock.stateVar* and *hfBlock.der_stateVar* don't do anything useful but are in the model because the OpenModelica runtime system needs at least one state variable to operate.

Listing 1: Modelica model.

```
// -----
// ElasticitySolver
// FMI application with
// HiFlow^3 block for PDE solving.
//
// Authors:
// Chen Song, Martin Wlotzka,
// Kristian Stavaker
//
// Main class
model ElasticitySolver
  // HiFlow^3 component
  ElasticitySolver_me_FMU hfBlock;
```

```
// Source for signals that
// should be constantly 0
Modelica.Blocks.Sources.Constant
  zeroSource(k=0.0);

Real u_center( start =0.0);
Real force( start =10.0);
equation
connect(hfBlock.u_center, u_center);
connect( force, hfBlock.force );
force = 10.0;
connect(hfBlock.der_stateVar,
  zeroSource.y);
connect(hfBlock.stateVar,
  zeroSource.y);
end ElasticitySolver ;
```

PARALLELIZATION CONCEPT

The parallelization concept of HiFlow3 is based on a decomposition of the spatial domain into a number of subdomains. For distributed memory systems, the message passing interface (MPI) [4] is used for data transfer. Each MPI process is dedicated to the computation for one of the subdomains. Hereby, work is distributed among the processes. After creating the mesh, that our finite element discretization is based on, the METIS graph partitioner [9] is used to determine a balanced partitioning of the mesh according to the number of MPI processes. Each process then only stores one part of the global mesh. Couplings between neighbouring parts are taken into account by means of a layer of ghost cells. Figure 2 shows an example of a domain decomposition into 8 parts. The matrix and vector data structures in HiFlow3 are

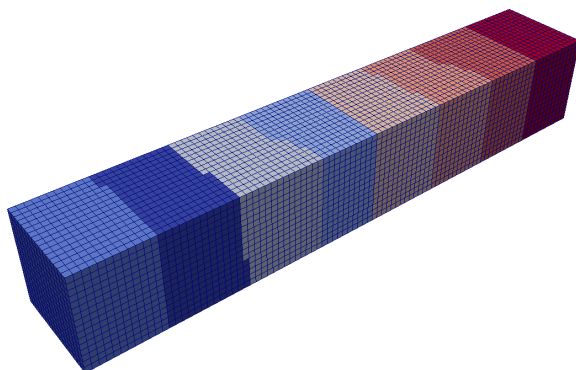


Figure 2: Partitioning of the mesh into 8 subdomains, indicated by different color.

distributed data structures which fit the partitioning imposed by the domain decomposition. Each process holds exactly those degrees of freedom of the finite element space which belong to its part of the domain. Couplings between different partitions are achieved by using ghost degrees of freedom. Only these have to be exchanged during parallel matrix-vector-product execution.

Assembly of the system matrix and right-hand-side vector, i.e. the computation of the entries, is done independently on each process for the corresponding subdomain. Hereby, the assembling process is designed in two levels: The global assembler iterates concurrently on each subdomain over the cells, while the local assembler computes the contributions for any single cell. Once the matrix and vector are assembled, the Conjugate Gradient linear solver takes advantage of the parallel implementation of the matrix-vector-operations when computing the solution.

Parallel Execution of the Model

The Modelica compiled model code is executed on a number of processes. The Modelica compiled model code is hereby replicated on each process. Whenever the HiFlow3 PDE component is called, it performs distributed parallel computations for solving the elasticity problem, hereby taking advantage of its parallelization concept based on the domain decomposition.

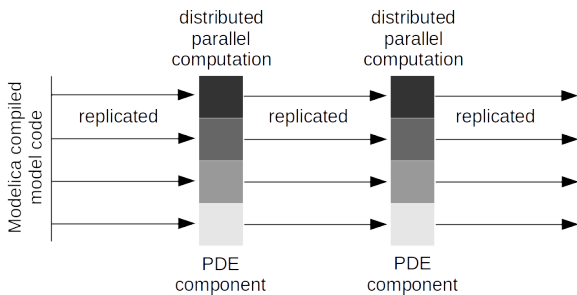


Figure 3: Replicated parallel execution of the Modelica compiled model code and distributed parallel computation in the HiFlow3 PDE component, sketched for 4 processes.

NUMERICAL EXPERIMENTS

We carried out a series of test runs with number of processes $n \in \{1, 2, 4, 8, 16\}$. We measured the run-

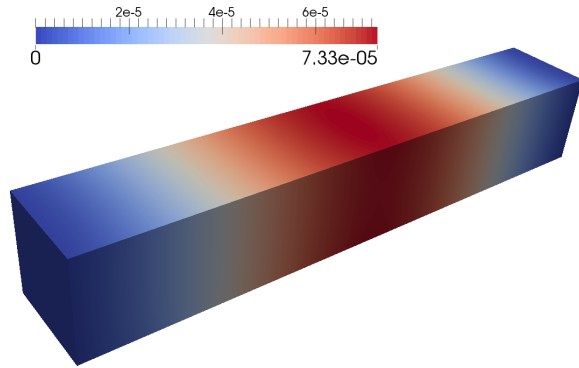


Figure 4: Visualization of the displacement in vertical direction.

time T_n for the PDE component when running on n processes. To assess the parallel performance of the solver, we computed the speedup

$$S_n := \frac{T_n}{T_1}$$

and the efficiency

$$E_n := \frac{S_n}{n},$$

where T_n is the run-time of the solver when executed on n MPI processes. Results are given in Table 1.

n	run-time T_n [sec]	speedup S_n	efficiency E_n
1	8.830	1.0	1.0
2	4.736	1.864	0.932
4	2.948	2.995	0.749
8	1.968	4.487	0.561
16	1.741	5.072	0.317

Table 1: Run-times for the PDE component with varying number of MPI processes.

Figure 5 shows a plot of the speedup and efficiency. The results show a parallel performance of the HiFlow3 PDE component within the Modelica context which is much poorer than the performance obtained for pure HiFlow3 applications. This is due to a technical reason: We needed to install a custom OpenMPI library, where we disabled the plugin architecture of the OpenMPI implementation on the machine while still using shared libraries. This was necessary for compiling the HiFlow3 PDE component into a dynamic shared object which can be loaded by the Modelica compiled model code during runtime. The diminished parallel efficiency is clearly due to the

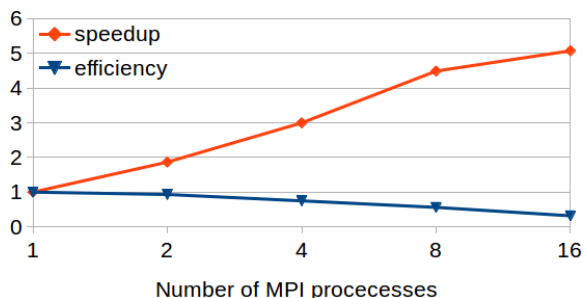


Figure 5: Parallel speedup and efficiency plot for $n = 1, 2, 4, 8, 16$ MPI processes.

use of such non-optimized MPI installation, since HiFlow3 shows good scalability on other machines with a high-performance MPI installation [8]. Nevertheless, we were able to leverage the parallel computing capabilities of HiFlow3 in the PDE component to introduce distributed memory parallelization for Modelica simulations. The performance tests show that even if parallel efficiency may not be optimal, our approach allows for solving large scale 3D PDE problems in high resolution on distributed memory machines. This is especially advantageous with respect to the amount of memory available, as the problem data can be split and distributed to several compute nodes, as opposed to a shared memory parallelization.

CONCLUSIONS

In this paper we have investigated a method of incorporating PDEs in the context of a Modelica model, by using FMI to import a PDE solver from the finite element library HiFlow3. Numerical results obtained using a simple coupled model that invoke a force and measures the elasticity deformation of a beam demoefficient and with already validated software. This paper is a continuation of a paper from the SIMS 2013 workshop [20]. In that paper the distribution of heat was measured in a piece of copper. The heat source was controlled by a PID-controller. A novelty in this paper is that we make use of parallel computations. The main advantages of this type of coupling include its simplicity and the possibility to reuse existing solver technology on multi-core and distributed memory architectures. In the results section we provided some run-time measurements of these parallel computations and compared to single-core computations. In-

duced by the needs for compiling the PDE component into a dynamic shared object and loading it by the Modelica compiled model code, limitations in the MPI library influenced the parallel performance. However, the speedup obtained is considerable in simulation practice, and the use of distributed memory architectures is a clear advantage with respect to memory, especially for large scale problems. In combination with our previous work reported in [20], this opens opportunities to address even more compute- and memory-intensive applications like instationary fluid dynamics problems. The parallelization approach with replication might seem somewhat clumsy. A better method would perhaps be to let the OpenModelica run-time system drive the parallel distribution. This is future work and it should be noted that the FMI places limits on parallel communication.

ACKNOWLEDGMENTS

Funded by the ITEA2 European Union MODRIO Project, by SSF in the EDOp project, and by the Swedish National Graduate School of Computer Science (CUGS).

REFERENCES

- [1] Hartwig Anzt, Werner Augustin, Martin Baumann, Thomas Gengenbach, Tobias Hahn, Andreas Helfrich-Schkarbanenko, Vincent Heuveline, Eva Ketelaer, Dimitar Lukarski, Andrea Nestler, Sebastian Ritterbusch, Bjoern Rucker, Staffan Ronnås, Michael Schick, Mareieke Schmidtreick, Chandramowli Subramanian, Jan-Philipp Weiss, Florian Wilhelm, and Martin Wlotzka. Hiflow3 a multi-purpose and flexible parallel finite element package. pages 1–15, 2011.
- [2] C.L. Dym and I.H. Shames. *Solid Mechanics: A Variational Approach, Augmented Edition*. Springer New York, 2013.
- [3] The Open-Source OpenModelica Development Environment. <http://www.openmodelica.org>.
- [4] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 3.0*. High-Performance Computing Center

- Stuttgart (HLRS), 2012.
<http://www.mpi-forum.org>.
- [5] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.
- [6] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach. (to appear)*. Wiley-IEEE Press, 2014.
- [7] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–436, 1952.
- [8] Vincent Heuveline, Eva Ketelaer, Staffan Ronnas, Mareike Schmidtbreich, and Martin Wlotzka. Scalability study of hiflow3 based on a fluid flow channel benchmark. *8th BFG/bw-GRID Workshop Proceedings*, 2012.
- [9] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, December 1998.
- [10] Zhihua Li, Ling Zheng, and Huili Zhang. Solving pde models in modelica. *2008 International Symposium on Information Science and Engineering. ISISE 08*, 1:53–57, 2008.
- [11] Manuel Ljubijankić and Christoph Nytsch-Geusen. 3d/1d co-simulation von raumlufströmungen und einer luftheizung am beispiel eines thermischen modellhauses. *Fourth German-Austrian IBPSA Conference, BauSIM 2012, Berlin University of the Arts*, 2012.
- [12] Manuel Ljubijankić, Christoph Nytsch-Geusen, Jörg Rädler, and Martin Löffler. Numerical coupling of modelica and cfd for building energy supply systems. *8th International Modelica Conference 2011*, 2011.
- [13] Engineering Mathematics and Computing Lab (EMCL) publication database., 2014.
<http://emcl.iwr.uni-heidelberg.de/79.html>.
- [14] Modelica and the Modelica Association.
<http://www.modelica.org>.
- [15] Modelica Association. *Functional Mock-up Interface for Model Exchange and Co-Simulation*, v. 2.0 beta 4 edition, 2012.
<https://www.fmi-standard.org/downloads> [Accessed 2013-08-06].
- [16] Preprint Series of Engineering Mathematics and Computing Lab (EMCL)., 2014.
<https://journals.ub.uni-heidelberg.de/index.php/emcl-pp/issue/archive>.
- [17] Levon Saldamli. *PDEModelica - A High-Level Language for Modeling with Partial Differential Equations*. Dissertation 1016, Department of Computer and Information Science, Linköping University, 2006.
<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-7281>.
- [18] Levon Saldamli, Bernhard Bachmann, Hans-Jürg Wiesmann, and Peter Fritzson. A framework for describing and solving pde models in modelica. *In Proceedings of the 4th International Modelica Conference, Hamburg, Germany, March 7-8, 2005*.
- [19] The HiFlow3 Multi-Purpose Finite Element Software. <http://www.hiflow3.org>.
- [20] Kristian Stavåker, Staffan Ronnås, Martin Wlotzka, Vincent Heuveline, and Peter Fritzson. Pde modeling with modelica via fmi import of hiflow3 c++ components. *Proceedings of SIMS 54th Conference, Bergen, Norway*, 2013.