JuliaFEM — open source solver for both industrial and academia usage

Tero Frondelius¹ and Jukka Aho

Summary. The JuliaFEM software library is a framework that allows for the distributed processing of large Finite Element Models across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. The basic design principle is: Everything is non-linear. All physics models are non-linear from which the linearizations are made as special cases. This is a work in progress. Thus, if you share the vision, contribute and join the community.

Key words: JuliaFEM, FEM, open source, Julia, julialang, natural frequency, www.juliafem.org

Introduction

The JuliaFEM project was started in May 2015 and is MIT licensed [6]. However, the story actually starts from years back of studying and using other open-source FEM packages. The findings were mainly divided into two categories: namely first, academic projects with pure academic goals and parallelism missing and, second, fast parallel codes that were so hard to start using and typically written-in statically compiled languages. Customizing software written using such languages requires significant developer time, thus sacrificing human convenience and even more importantly, typically very limited developer resources for software execution speed.

JuliaFEM is an open-source finite element solver written in the Julia programming language, which is comprehensibly described in [3]. All the same reasoning are true for why Julia and why JuliaFEM, in other words why new programming language was needed. Reference [3] answers the first question throughly, thus the authors encourage readers to read it or visiting https://julialang.org/. JuliaFEM enables flexible simulation models, takes advantage of the scripting language interface which is easy to learn and embrace. In addition, it is a real programming environment where simulation can be combined with other analyses and work-flows. These features introduce a place for testing new ideas and simulation models to the academic world.

¹Corresponding author. tero.frondelius@wartsila.com
Julia introduces the following features: JIT/LLVM [10], multiple dispatch, metaprogramming, very powerful macros (full programming language with Julia syntax), static and dynamic typing. The speed of Julia is really close to the speed of C/FORTRAN code. Julia shows that one can have machine performance without sacrificing human convenience. Julia combines expertise from the diverse fields of computer science and computational science to create a new approach to numerical computing. Julia is designed to be easy and fast to use. One of the key features of Julia is multiple dispatch, a technique from computer science which automatically picks the right algorithm for the right circumstance [3].

Julia provides a wide range of libraries and a very skilled community in both skills and education, which enables a high level of abstraction in the programming code, such as using the automatic differentiation library ForwardDiff [16] for calculating gradients, jacobians and hessians for all types of field data. Using automatic differentiation in a FEM package is not a new idea; for example FEniCS [1], an open-source FEM solver, uses it comprehensively. Another important feature of the open-source software development is the method of building on top of existing open-source work. Thus JuliaFEM offers an io-interface to CODE ASTER [17], also an open source FEM solver. Developing JuliaFEM also encourages good practices, starting from unit testing both for smaller and larger functions and continuing to full integration testing of different platforms.

At the moment, users can perform the following analyses with JuliaFEM: elasticity, thermal, eigenvalue, contact mechanics, and quasi-static solutions. For visualization, JuliaFEM uses Paraview [2] which prefers XDMF [13] file format using XML to store light data and HDF [18] to store large data-sets, which is more or less the open-source standard.

Vision

On one hand, the vision of the JuliaFEM includes the opportunity for massive parallelization using multiple computers with MPI and threading as well as cloud computing resources in Amazon, Azure [7] and Google cloud services together with a company internal server [9]. And on the other hand, the real application complexity including the simulation model complexity as well as geometric complexity, see some examples in [4, 8, 19]. Not to forget that the reuse of the existing material models [11] as well as the whole simulation models are considered crucial features of the JuliaFEM package [5, 12].

Recreating the wheel again is definitely not anybody’s goal, and thus we try to use and embrace good practices and formats as much as possible. We have implemented Abaqus/Calculix [21] input-file format support and maybe will in the future extend to Other FEM solver formats. Using modern development environments encourages the user towards fast development time and high productivity. For developing and creating new ideas and tutorials, we have used Jupyter notebooks [15] to make easy-to-use handouts.

The user interface for JuliaFEM is Jupyter Notebook [15], and Julia language itself is a real programming language. This makes it possible to use JuliaFEM as a part of a bigger solution cycle, including for example data mining, automatic geometry modifications, mesh generation, solution, and post-processing and enabling efficient optimization loops. In the next section a concrete example including statistical inference combined to FEM calculation is shown.
Big numerical example

Typical examples in industrial applications include non-linear solid mechanics [11], contact mechanics [20], finite strains [14], and fluid structure interaction problems. Here there is some simulation of machine parts having different amounts of elements and DOF’s for comparison reasons. These simulations take a lot of computational resources, and here are specs of the used hardware: 24 x Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz with 512 GB total memory.

<table>
<thead>
<tr>
<th>DOFS</th>
<th>Assembly time (s)</th>
<th>Solution time (s)</th>
<th>Total time (human readable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0 M</td>
<td>458</td>
<td>312</td>
<td>16 min</td>
</tr>
<tr>
<td>10.8 M</td>
<td>3257</td>
<td>3255</td>
<td>2 h 4 min</td>
</tr>
<tr>
<td>12.6 M</td>
<td>3654</td>
<td>6318</td>
<td>3 h 4 min</td>
</tr>
</tbody>
</table>

Table 1. This table demonstrates the solution times for some real industrial size FEM models

Figure 1. Real industrial size FEM model.

References


Tero Frondelius
Wärtsilä
Järvikatu 2-4
65100 Vaasa
tero.frondelius@wartsila.com

Jukka Aho
Global Boiler Works Oy
Lumijoentie 8
90400 Oulu
jukka.aho@gbw.fi