UPC - BARCELONA TECH
Master on Numerical Methods in Engineering
Fall 2017

Industrial training

Industrial Training Report

Samuel Parada Bustelo

December 2017

1 Introduction

In the present report it is intended to deeply describe all the work performed during the industrial training course of the Master on Numerical Methods in Engineering. In order to complete the internship it is needed to successfully finish 450 hours of work. I have developed my industrial training within the framework of the Structural Mechanics group of CIMNE, with Professor Joan Baiges as supervisor. The main goal of the work was to help the research group in the development of their in-house Finite Element Method (FEM) code, named `FEMUSS` (Finite Element Method Using Subgrid Scales).

In my personal case, I have divided the industrial training into two parts. The first 100 hours were completed prior to the summer break. The second part includes from September to the end of the semester, in December.

2 Development

In this section, it is described the different parts of the internship. The section is divided into three distinct parts, one for the initial stage (basically July), the next one dedicated to the core part of the industrial training were the main tasks were performed, and a last part related with the final stage, mainly dedicated to the start of the Master thesis.

2.1 Initial stage: July

The initial stage was basically dedicated to not only learn the basics of the so-called `OOP` (Object Oriented Programming) using the Fortran 2003 programming language, but also to get used to the `Linux` environment and installing the `Femuss` code.

For the basics of `OOP`, I was provided with several documents and book chapters to study the theory and be able to understand some practical examples that were proposed. I have to say that, as a novel programmer using Fortran, the work seemed to be really challenging at the beginning. Important concepts that were treated in detail in this part were for instance, subroutine, module and function definitions, objects and class definitions, polymorphism and inheritance, etc. Basically, I was studying the novel features introduced in the Fortran 2003 standard, which is the one used in `Femuss`.

Another important part of this initial stage of the internship was to get used to the `Linux` environment. In my case, this part was not that hard, as I had some previous training thanks to an internship done during my senior year of the bachelor degree. But still, getting used to deal with the `Linux` terminal and basic commands such as copy or move file, being able to modify them, etc. takes its time. Also, I was introduced to the `Kate` file-modifying program.

The next step was directly to start downloading and installing the code `Femuss`. For this task, I was assisted by one of the PhD students of the group, which explained me the basic steps of not only installing the code, but also the ones related with the compiling process of all the files. At the end of this first day with `Femuss`, we were even able to run a very simple case of flow on a channel, using the module dedicated to the Navier-Stokes equations.

In order to solve a problem using this `FEM` code, we can first define our geometry features using the `GiD` preprocessor developed here at CIMNE. After this, all the files generated regarding mesh and conditions will be used by `Femuss` to compute the solution of the

problem. Finally, the post-processing stage can be done using again GiD or any other visualization software such as Paraview.

Within this part, we had the pleasure of assisting to a seminar giving by Professor Joan Baiges, to other researchers in CIMNE, where he basically explained and introduced the basics of this simulation tool to potential future users from other research groups here at UPC.

In order to understand the structure and the operation basics of Femuss, I was provided with a guide of the code [3]. Therefore, the next step was to read through it in order exactly understand the programming structure of the code, which will be very important when performing new implementations in the near future.

The structure of Femuss is based on hierarchical relations between objects. The main characteristic of this organization is that objects from below in the hierarchy should no 'know' about objects which are above them. This is an important feature of the code, which permits to have a clean programming technique and an agile software development. The higher level structure in Femuss are referred as CASES. In each case, all the processes which take place on top of the same mesh are grouped. As a consequence, each case has a MESH, a FILE POSTPROCESSOR and might have several PHYSICAL PROBLEMS. This CASES are defined in such a way that the code is capable of dealing with, for instance, fluid-structure interaction problems, where each of the problems is defined in a different computational domain. The mesh object takes care of all the geometrical information, i.e., elements, connectivities, coordinates, shape functions, etc. We also have the file postprocessor which contains objects in charge of writing information to the disk so that the results can be lately visualized. For example, it is in charge of writing the velocity and pressure arrays of an incompressible Navier-Stokes problem to disk so that they can be open using GiD or Paraview. The final ingredient of this structure is the concept of COMMUNICATION CHANNELS. These are basically a set of pointers which allow to pass information between entities which are on the same hierarchical level.

2.2 Middle stage: September to mid November

The first main task of the internship was to come up with the development of a file output format using the HDF5 (Hierarchical Data Format 5) standard. This would have to be successfully included within the hierarchical organization of Femuss.

HDF5 is a data model, library, and file format for storing and managing data. It supports an unlimited variety of data types, and is designed for flexible and for high volume and complex data. It is portable and is extensible, allowing applications to evolve in their use. It is also the most used format for large scale supercomputing applications. This format has its own programming strategy based on directives and calls to functions provided with the library.

One of the major problems of using directly the HDF5 is that it is written in C language, even though it includes wrappers to couple its use with Fortran. Since our FEM code is written in Fortran, an intermediate step of developing a interoperability strategy would be necessary, in order to allow both platforms to communicate. This step would introduce some difficulty in the programming of the whole thing and thus, we wanted to avoid that intermediate step. Therefore, we made some research to try to find a library which was already able to perform that intermediate step of coupling our Fortran code with HDF5.

Within this framework, we discovered XH5For, which is a library to read and write parallel partitioned FEM meshes taking advantage of the input/outputs provided by the

HDF5 library, using directly Fortran code. In other words, it basically allows `HDF5` to interpret calls from a Fortran code. This was exactly what we were looking for.

The next obvious step was to read the `XH5For` guide with examples, so as to understand the behavior of the code and the programming technique (compilation, commands and directives, etc.) needed in order to include it within `Femuss`. Another important feature of this library is that it allows to go parallel which is key for large scale applications.

The following weeks were therefore spent in the development of the output file using `XH5For`. Once it was finished, with obviously some help from the supervisor, we run both serial and parallel cases with `Femuss` so as to test whether the implementation was successful or not. After testing, the implementation was approved by the supervisor and nowadays, it is fully available in `Femuss` for its users in the research group.

Once we finished that first main task, we headed to the second one. In this case it is wanted to coupled with `Femuss` an external library called `MUESLI`. `MUESLI`, a **M**aterial **U**niv**E**r**S**al **LI**brary, is a collection of C++ classes and functions designed to model material behavior at the continuum level. It is available to the material science and computational mechanics community as a suite of standard models and as a platform for developing new ones.

One of the main features of this library is that it provides well-tested implementations of several standard material models for small and large strain solid mechanics, fluid mechanics, and others. In addition to this, it uses high-level C++ classes that simplify the tensor algebra involved, especially, in complex material models. Finally, it also includes an interface to couple the `MUESLI` material models with commercial codes such as `Abaqus`. `MUESLI` is a thread-safe library and it can be safely used in shared and distributed memory computers but, prior to that, it needs to be built and linked to the main program (in our case `Femuss`). Therefore, the first step of the task was to successfully build and link both codes.

As pointed out above, one of the major characteristics of this library is that it is completely written in C++. This is obviously a disadvantage for our code, because `Femuss` is written in Fortran. Therefore, as we described above for the case of `HDF5`, we need to have a kind of interpreter so that `MUESLI` can understand all the information from `Femuss`.

This situation, even though someone could think that is strange, is actually pretty common in computational engineering development. There are many programming languages available nowadays, and we, as programmers, have to choose one at the end of the day to develop our implementations. Therefore, most of the time, we find ourselves in the position of using an external library which is not in the same language as our code. In this special case, languages are C++ and Fortran. This issue is complicated by the fact that both languages have been around for a long time, and various recent language standards have introduced mechanisms to facilitate interoperability. However, there is still a lot of old code around, and not all compilers support the latest standards.

Luckily for us, Fortran 2003 provides a standardized mechanism for interoperating with C, and thus C++. This support covers the following Fortran features:

- Interoperability of procedures - a means of referencing procedures that are defined in the C programming language, or that can be represented by C language prototypes, and a means of specifying that a procedure defined in Fortran can be called from C.

- Interoperability of types - a means of declaring types and enumerations in Fortran that correspond to C types.

- Interoperability of global data objects - a means of declaring global variables that are associated with C variables with external linkage.
- An intrinsic module (ISO_C_BINDING) that provides access to named constants and procedures relevant to C interoperability.

Clearly, any interoperable entity must be such that equivalent declarations can be made in the two languages. This is enforced within the Fortran program by requiring all such entities to be interoperable.

Even though `MUESLI` provides a wide range of material models, in our case we just wanted to implement the solid mechanics part, since fluid models are already extensively implemented in `Femuss`. The final implementation took about an entire month of code developing to translate all the features of `Muesli` to `Femuss`. Again, at the end, we tested the developments by simulating some easy solid mechanics problem, to check that everything worked properly. Therefore, we were able to successfully coupled the external library written in C++ with our Fortran code, by means of the ISO_C_BINDING standard strategy.

Finally, to conclude this section, let us point out that, in order to be able to use the `Muesli` and `XH5For` libraries in a complete way, we needed also to adapt the `GiD` pre-processing stage for `Femuss`, i.e. we need to change the Problem Type definition.

2.3 Final stage: mid November to end of the semester

The final part of the industrial training was completely dedicated to work on the future master thesis. In my personal case, Professor Joan Baiges proposed me to develop a fractional step method (also known as projection methods) for compressible flows, a method which is not yet implemented within the `Femuss` environment. This is to be done for the solution of the compressible Navier-Stokes equations.

Up to know, the work done for this part entails to study the already implementations done for compressible flows in `Femuss` and to do a bibliographical research on fractional step methods applied to compressible flows.

3 Conclusions

In this final section of the report, we include some conclusions with respect to the work performed during the internship, which was composed of 450 hours of dedicated work completed in two main stages (the moth of July prior to summer break, and the first semester from September to December).

During the very first part of the internship, I became familiar with the basics of Object Oriented Programming using the standards of Fortran 2003, and got used to the `Linux` environment. The first main goal of the internship was to investigate different file output formats for the management of extremely large and complex data collections used in large scale supercomputing applications. Within this framework, an `HDF5` file output format was developed for `Femuss`.

The second main part was dedicated to learn how to link and compile external libraries with our FEM code. Related to this, we have coupled `FEMUSS`, written in Fortran 2003, with `MUESLI`, a solid mechanics library written in C++, by developing an interface for programming languages interoperability.

Finally, the last part of the industrial training was mainly dedicated to work related with the Master thesis, in my case on the development of fractional step methods for compressible flows, which are to be modeled with the so-called compressible Navier-Stokes equations.

4 References

References

[1] Portillo, D., Pozo, D. D., Rodríguez-Galán, D., Segurado, J., & Romero, I. (2017). *MUESLI: A Material UnivErSal LIbrary.* Advances in Engineering Software, 105, 1-8.

[2] Santiago Badia, Alberto F. Martín and Javier Principe *FEMPAR: An object-oriented parallel finite element framework.*

[3] Joan Baiges, *Femuss documentation. Technical report.* International Center for Numerical Methods in Engineering, 2014