INDUSTRIAL TRAINING, SUMMARY OF THE DEVELOPED WORK AND REACHED OBJECTIVES.

INOCENCIO CASTAÑAR

Abstract. Internship is a powerful technique to take the first steps and assess the effectiveness of our studies. Actually, the purpose of all those who are currently enrolled in a master's or doctoral programme is to end up being able to apply all the knowledge they have learned. In this article, it is intended to deeply describe all the work performed during my industrial training course. The main goal of the work was to come up with the development of both a file output format using *HDF5* and a file to interoperate between *C++* and *Fortran 2003*. *HDF5* file format is widely suitable when working with high volume and complex data such that large scale applications. The interoperability between several programming languages is hugely recommended because it allows us to take advantage of the advantages of each one of them. Thanks to this industrial training, it is learned how to work with a real scientific group and never give up any complex problem.

Keywords: industrial training; HDF5 file format; interoperability, Object Oriented Programming.

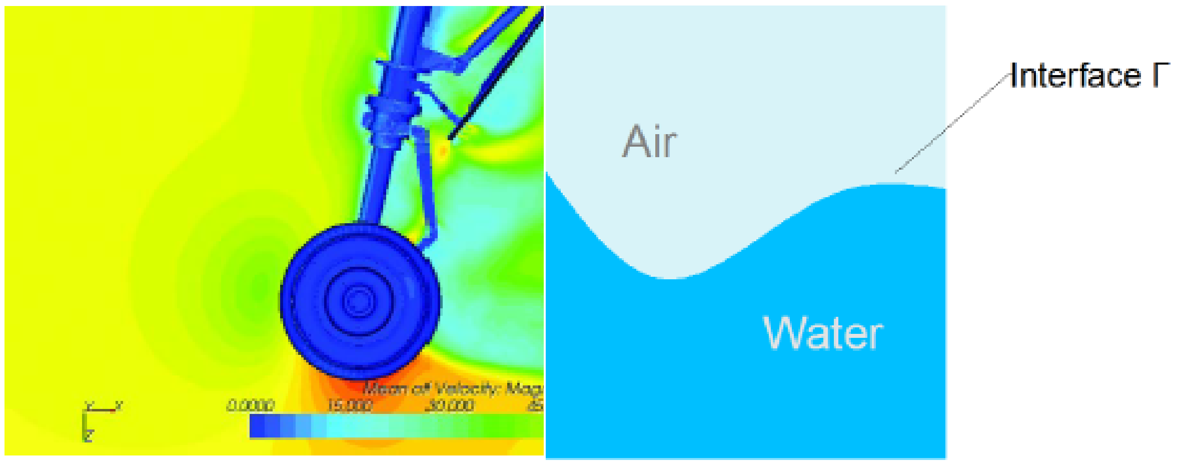
Contents

1. Introduction

Internship is a powerful technique to take the first steps and assess the effectiveness of our studies. In fact it is widely used in a lot of master's and university degrees as a compulsory subject in order to guarantee a successful incorporation of the students to the world of work. I am currently enrolled in the Master on Numerical Methods in Engineering. As a subject of my master's, it is needed to successfully complete 450 hours of work in a suitable industrial placement.

In my case, I decided to look for a group of CIMNE within the framework of the Structural Mechanics. This decision was taken due to the fact I was enrolled in two very interesting subjects: Coupled Problems and Domain Decomposition Methods for Large Scale Applications. The first one deals with those problems where multi-physics or multi-phase applications are involved (e.g., fluid-structure interaction (see Fig. 1a), multifluid flows (see Fig. 1b)) whereas the second one tries to come up with the most suitable techniques to face different problems in realistic large scale applications. As it is already known, it is very important to work in exciting areas because you will devote a lot of time working on it. For this reason I started my industrial training with Professor Joan Baiges as supervisor, the head teacher of Coupled Problems.

Date: January 9, 2018.
E-mail `inocasta@gmail.com` (IC).



(A) (B)

Figure 1. Examples of Coupled Problems.

The main goal of the work was to help the research group in the development of their in-house Finite Element Method code, named *FEMUSS* (**F**inite **E**lement **M**ethod **U**sing **S**ubgrid **S**cales). In my case, I was requested to perform two tasks with my classmate, Samuel Parada.

The first main task of the internship was to come up with the development of a file output format using a specific one, the *HDF5*(Hierarchical Data Format 5) standard. *HDF5* is a data model, library and file format for storing and managing data. It is quite convenient for flexible and for high volume and complex data due to the fact that it supports an unlimited variety of data types and it can be accessed in parallel if desired. Up to that point, *FEMUSS* was provided with two ways of storing data, *VTK* file format and *GiD* file format. Both of them are high-quality. However, they are not intended to work in large scale applications, where lots of processors have to access to the output file to store data. This procedure must be done at once in order to reduce waiting times. *HDF5* file format solves this problem and for this reason it is widely used in large scale computing programs.

On the other hand, the second task was to couple with *FEMUSS* an external library called *MUESLI* [1]. *MUESLI* (**M**aterial **U**niv**E**r**S**al **LI**brary) is a collection of C++ classes and functions designed to model material behaviour at the continuum level. Developed at IMDEA Materials, 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 both small and large strain solid mechanics, but also thermal analysis, small strain termomechanical behaviour, fluids and reduced dimensionality models(e.g., plane stress, shells, plates, beams). It is also a thread-safe library which means that it can be safely used in shared and distributed memory computers. This library is written in *C++* whereas *FEMUSS* is written in *Fortran 2033*. This task can be split in two steps: First of all it was needed to successfully build and link both codes and secondly to program a file to interoperate between those programming languages. This situation is very common in computational development due to the fact that there are several programming languages and each of them has its own advantages and drawbacks. Interoperability allows us to take advantage of the different benefits of each kind of programming languages.

The outline of the article is as follows. In Section 2, it is introduced the methodology and the strategy to carry out our tasks. In Section 3, it is shown some results to verify the implementation of our files. Finally, it is drawn some conclusions from the industrial training in Section 4.

2. Methodology

The aim of this section is to explain how we managed to complete the requested tasks during the industrial training. As it was mentioned just before, several works were performed along 450 hours of work. For this reason, this section is split in several subsections to explain each task in detail.

2.1. Learning basics of *FEMUSS* and its environment.

FEMUSS is an in-house code written in *Fortran 2003*. It is based on the principles of Object-Oriented Programming. Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another. There is significant diversity of OOP languages, but the most popular ones are class-based, meaning that objects are instances of classes, which typically also determine their type. Both *Fortran 2003* and *C++* are class-based.

The initial stage was basically dedicated first of all to learn the basics of the so-called *OOP* using the Fortran 2003 programming language. There were a lot of new concepts for us such that encapsulation, inheritance, polymorphism ... To see further details of Object-Oriented Programming take a look at [2].

FEMUSS is only available for Linux environments, so that we had to learn how to work in this environment. In my case, Linux environment was only used once during my course in Domain Decomposition Methods and Large Scale Applications so that this part was a little hard. 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*. It could seem very easy. However, for those who are not knowledgeable about Linux environment it can be a challenge. For this task, we were assisted by one of the PhD students of the group, which explained us 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.

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.

FEMUSS was built taking into account that it must be able to be adapted to newest codes and ways of programming.

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 higher level structure in *FEMUSS* are CASES. In each case, all the processes which take place on top of the same finite element mesh are grouped. As a consequence, each case has a mesh, an adaptive refiner, a file postprocessor and might have several physical problems. Cases are defined so 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 and with different mesh requirements.

The mesh object takes care of all the geometrical information (e.g., elements, connectivities, coordinates, etc). It also is in charge of loading shape functions etc to the element object, which is the geometrical entity in charge of providing this information in the elemental subroutine.

The Adaptive refiner object is an object in charge of defining the refinement strategy when adaptive mesh refinement is present. It interacts with the mesh, the physical problems etc in order to update all the arrays to the new geometrical configuration after refinement has taken place. The important point is that the mesh knows nothing about the refinement procedure, it is only provided with the new modified arrays, so after the refinement process the mesh is treated as a usual mesh without adaptive refinement.

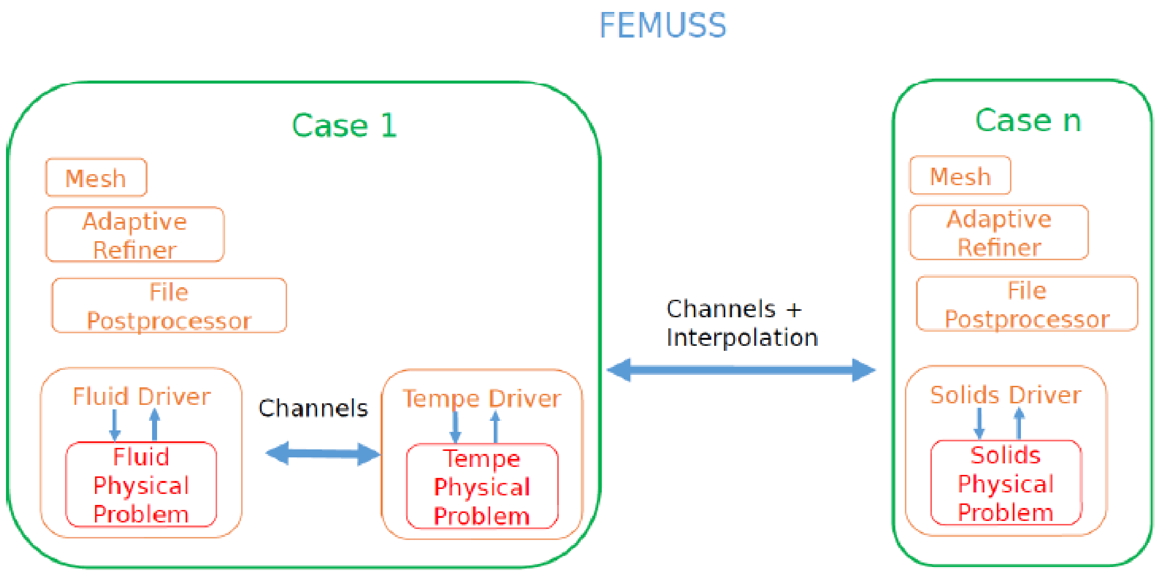


FIGURE 2. General Structure of FEMUSS: Flowchart .

File postprocessors are objects in charge of writing information to disk so that the results can be visualized. For instance they will be in charge of writing the velocity, pressure arrays of an incompressible Navier-Stokes problem to disk so that they can be open by *GiD*, *Paraview*, etc,

Another important concept are the Physical Problems and the Physical Problem Drivers. The main idea here is that a Physical Problem has no interaction with the exterior except through the Physical Problem Driver, which 'drives' or commands and tells what to do to the physical problem.

The final ingredient of the general structure of *FEMUSS* is the concept of Communication Channels. Communication channels are basically a set of pointers which allow to pass information between entities which, on the *FEMUSS* general structure, do not know anything about each other because they are at the same hierarchical level.

2.2. Task 1: HDF5 File Format.

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*, there is the necessity of creating an interoperable file between them, 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. By making some research throughout internet, we found an already created library able to perform that intermediate step of coupling our Fortran code with *HDF5*.

Within this framework, we discovered *XH5For* from *FEMPAR* group [4], 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 behaviour of the code and the programming technique (compilation, commands and directives, etc.) needed in order to include it within *FEMUSS*.

2.3. Task 2: Interoperability.

As pointed out above, one of the major characteristics of *MUESLI* library is that it is completely written in *C++*. This is obviously a drawback for our code, because *FEMUSS* is written in *Fortran 2003*. 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 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*.

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.

2.4. **Master Thesis.** The final part of the industrial training was completely dedicated to work on the future master thesis. Professor Joan Baiges proposed me to develop either an electromagnet hydrodynamics coupled problem or develop a code for topology optimization. The first one is related to study the magnetic properties of electrically conducting fluids. On the other hand, Topology Optimization (TO) is a mathematical method that optimizes material layout within a given design space, for a given set of loads, boundary conditions and constraints with the goal of maximizing the performance of the system. TO is different from shape optimization and sizing optimization in the sense that the design can attain any shape within the design space, instead of dealing with predefined configurations. Yet interesting, the electromagnet hydrodynamic problem has few realistic practical applications so that I decided to study and develop a code for Topology optimization for both small and large deformations.

Up to know, the work done for this part entails to study the already implementations done for topology optimization in *FEMUSS* and to do a bibliographical research on topology optimization methods and properties. [5] [6] .

3. Results

In this section, it is analysed that the programmed files are well-implemented. First of all our files are carried out several tests to check that they satisfy all the requirements of our code *FEMUSS*. Once our files have passed all tests, it is intended to validate our codes. Each file must be validated in different ways.

As it was explained in Section 1, HDF5 file format is very suitable when working in realistic large scale applications because it decreases waiting times when accesing data to store it. For this reason , it was checked that the time needed to store the data for HDF5 file was the shortest in comparison with the already existing output file formats. Unluckily, it would be needed some extra hours of supercomputers to extract more results.

Let us consider a simple problem, in 2D. Our goal is to show that the file does not perturb the parallelism propoerty of the code. In other words, that it is well-implemented and the data is accesed at once per each processor when storing the data. Let us consider a number of processors from 8 to 128. It can be seen at figure 3 that the execution time decreases when increasing the number of processors so as our code bahves correctly in parallel.

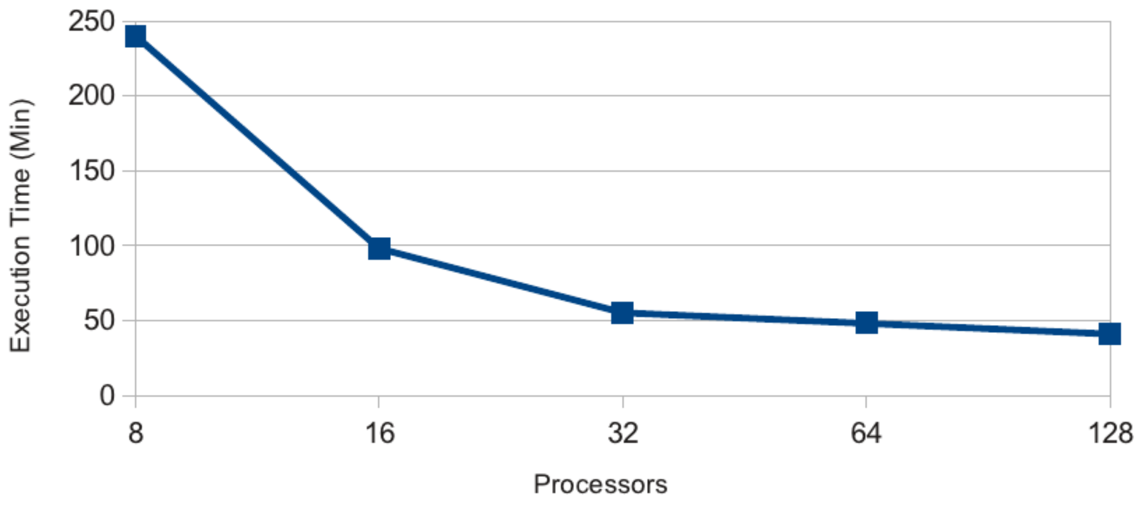


FIGURE 3. Execution Time of a given problem against number of processors.

On the other hand, to check the well implementation of our interoperability code, it was needed to compare some physical problems by using the external library of *MUESLI* against the same problems by using an already well-implemented code, such that the module for small strains of *FEMUSS* itself. Therefore, both modules gives the same results for different problems so as to we can conclude that our interoperability file was well-implemented.

4. Conclusions

To end up this article, it is drawn some conclusions with respect to the work performed along the industrial training.

In order to be able to start our tasks, it was needed some time to get used to *FEMUSS* and its environment. We were warned about how difficult this step was. Effectively, I can bear witness to it. There were a lot of new concepts such that Object-Oriented Programming, Linux environment or *FEMUSS* itself. It was pretty difficult but we managed to achieve it.

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*. As it was shown above, it leads our *FEM* code to decrease waiting time when storing data.

The second main task 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. As it was seen in Section 3, the file is well-implemented and it passes all tests.

Finally, the last part of the industrial training was mainly dedicated to work related with the Master thesis, in my case on the study and development of topology optimization, which are to be modelled with both small and large strain deformations.

References

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

[2] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

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

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

[5] Cintia Gomes Lopes, Renatha Batista dos Santos & Antonio André Novotny. *Topological Derivative-based Topology Optimization of Structures Subject to Multiple Load-cases.* Latin American Journal of Solids and Structures, 2015.

[6] M.P. Bendsøe & O.Sigmund *Topology Optimization: Theory, Methods and Applications.*Springer, 2003.

Universitat Politècnica de Catalunya, Jordi Girona 1-3, Edifici C1, 08034 Barcelona, Spain.