

PRIMULA: A Framework Based on Finite Elements to Address Multi Scale and Multi Physics Problems

Alejandro Soba^(✉)

CNEA - CONICET Centro Atómico Constituyentes,
Av. Gral. Paz 1499, San Martín, Argentina
soba@cnea.gov.ar

Abstract. The PRIMULA code, a multi scale and multithreads open source framework based on finite elements applicable to the numerical resolution of partial differential equations is presented. PRIMULA is portable to LINUX/UNIX, where it is compiled with gfortran, and to WINDOWS, compiled in the Visual Studio environment. It can be compiled to run in series, with shared memory under the Standard OPENMP, in a distributed environment under Standard MPI and on hybrid systems, with a compilation that combines MPI-OPENMP. The code was tested with non-linear problems in a 16 cores Intel Xeon (R) E5-2630 v3 multiprocessor of 2.4 GHz and in TUPAC, with $4 \times$ Hexadeca core AMD Opteron 6276s processors. This paper presents results of scalability and computation times of some of the multiple tests to which it was submitted.

Keywords: PRIMULA · MPI-OPENP · FEM solver

1 Introduction

There are numerous software packages focused on problems based on differential equations in partial derivatives using the finite element method. The offer is wide for user looking for paid [13, 14], restricted [12] or free [11] licensed codes. Most of them consist of robust solvers that can be applied to the resolution of nonlinear, stationary or time-dependent problems, allowing them to approach numerical solutions of very complicated real systems. These packages usually support several parallelization modes that allow them to take advantage of the computing capacity of large computers, either distributed memory [22], shared [21] or lately with hybrid format, to profit the new generations of multithreads machines.

The geometries that these packages manipulate are also of great complexity. They are discretized using both structured and unstructured meshes with equal efficiency, to cover a wide range of problems from the spatial point of view, as well as numerical precision. In particular, this way of facing certain problems in simulation enters into what is ambiguously called multiphysics-multiscale. The multiplicity of analyzed physical problems as well as the different ranges of work scales supports this form of denomination. The method of finite elements [8], often combined with finite differences

and finite volumes, so versatile and adaptable to all kinds of geometries, has come to synergistically favour the limits of applicability of the packages mentioned in such a way that today it seems there is no physical problem that cannot be numerically dealt with.

Due not only to the cheapness of hardware but also to the growing importance of numerical simulation in various branches of science and technology, the acquisition of a computer with several thousand processors is an accessible goal even for developing countries, companies or scientific institutes at our national level. However, this supply of increasingly fast and powerful equipment is not accompanied by the development in equal measure and intensity of codes that take advantage of this availability. Rather, users often fall into the automatic use of packages already established in the market, either under paid or free license, but which show a certain robustness based on the number of groups in the world that use them or the number of publications that cite them. In some instances the scientific communities of certain countries can afford a supercomputer, but they do not seem to understand that such acquisition should be accompanied by a similar effort in time and money for the development of appropriate software, which would guarantee independence from the point of view of the generation and utilization of knowledge.

Currently in a computer like TUPAC [7], not only the administration tools are of foreign origin, but of free license [9, 10], but the 80% of the software that runs on the machine belongs to packages not developed by local groups. A great part of that time is dedicated to calculations with codes of first principle, of the so-called *ab initio* or similar, where by means of molecular dynamics or other approximate methods calculations at the atomic scale are performed. Also finite element codes like FLUENT [14], OPENFOAM [11], and the WRF-ARW [15] for climate analysis are employed in 80% of the studies. The remaining 20% uses more specific software, and only a minority of them written by local scientists.

Without diminishing merit to those groups that identify particular software and understand that this development is what they need for their research, this lack of local developments marks a trend towards dependence on the way in which scientific knowledge is manipulated. It also limits in a certain way the problems that can be faced. Due to the impossibility of adapting the code, a certain degree of simplification or idealization of the particular problem has to be assumed by the user so that the code can provide a solution.

In the Codes and Models Section of the Nuclear Fuel Cycle Management, CNEA we have started from a philosophy opposite to the one described above. Certainly, the type of institution we belong to and the character of the numerical challenges that the group needs to solve imposes that route. The confidentiality of certain problems as well as the difficulty of adapting these problems to the pre-existing packages in the market impelled us to develop our code at home, building the package that we have called DIONISIO [16–20], owned by CNEA. This code is devoted to the simulation of nuclear fuels behaviour during reactor operation, under normal and/or accident conditions. Until now it runs in desktop machines, with some computation sections parallelized under the standard Openmp [21]. DIONISIO is a multi-physic and multi-scale code, since it has models ranging from the level of a fuel grain, to the thermal-hydraulic description of heat removal by the coolant from a whole rod (typically some meters long).

The code has one, two and three-dimensional models, which operate in coupled form. However, the pretension to extend the analysis to the description of more realistic geometries, and to increase the dimension of the involved models, multiplies the computing time required. For this reason, the PRIMULA framework was designed. It is destined to solve coupled, non-linear differential equations, in two and three-dimensional geometries.

As the DIONSIO programmers and in general those of any type of application, are not experts in the writing of a distributed code, it is fundamental for the PRIMULA user that the framework structure remains hidden for him in such a way that he can parallelize his code without entering into the framework details. On the other hand, we intended to build a code that does not distinguish between operating systems (PRIMULA can be executed in a WINDOWS as well as in a LINUX environment without any further modification than, obviously, the way of compiling) or in which type of machine it is executed (it should be irrelevant for the user if the code is executed on a desktop machine or a supercomputer, so that it is transparent to serial or parallel use). Thus, through the modification of a couple of variables in the input file, the user decides whether the code will be executed serially or in parallel, and in the latter case, if the chosen parallelization will be over distributed or shared memory or a hybrid of both.

A final detail was considered in the design of PRIMULA: even for the simplest finite element packages users must be involved in programming certain details of the code to adapt it to their own particular problem. In many cases packages such as ABAQUS or OPENFOAM support writing and recompiling code segments. That is why today it is increasingly difficult to find pure users of a code, and in general, to achieve maximum software performance, the user becomes a programmer. Understanding this mutation of roles, PRIMULA also requires that the user actively intervene in the programming of certain code sectors. In the modification of the input and the output, as well as in the manipulation and creation of the variables that each user needs to introduce in his simulation, new programming is required. It should be noted that for each type of intervention, there are portions of code already written that facilitate the rapid learning of the novel programmer. Figure 1 shows a diagram of PRIMULA in which the separation between the systems to be solved and the kernel of the code is schematized. The IO of the same works through a series of input files that allow the selection of the different types of possible executions programmed so far in the kernel, as well as the manipulation of the system that the user wishes to solve.

PRIMULA is a code in development, which never seeks to consolidate in a fixed package, but is proposed as a framework that the user must adapt to his own particular problem and then continue with his own development. PRIMULA is completely versatile and has as one of its peculiarities of style (in its “to be delivered” state) to avoid FORTRAN language features that are not specific to the gfortran standard, this is, the most usual and common of the compilers of the moment. Each user, once he takes over the framework, can give him personal guidance related to the environment where he wants to execute the code under his own responsibility.

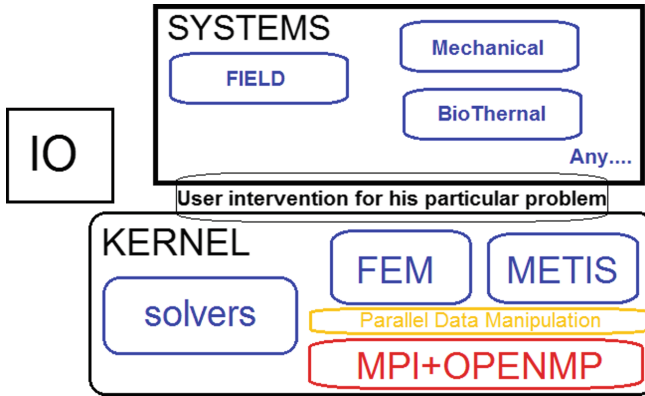


Fig. 1. Scheme of the engineering of the PRIMULA framework.

This work is proposed as a brief summary of the main characteristics of PRIMULA, in addition to a brief presentation of its capabilities, some results obtained and analysis of scalability and performance in the different environments where it was tested. Naturally, this report does not cover all the possible problems or tests to which it will be subjected in the near future, but is only intended to show the perspectives of its development.

2 PRIMULA General Features

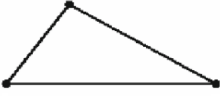
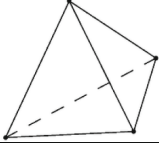
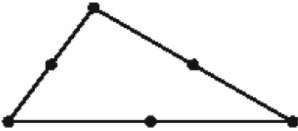
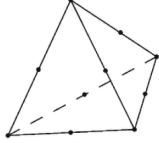
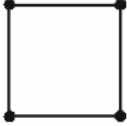
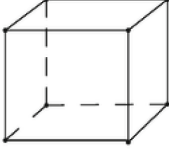
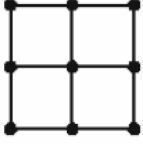
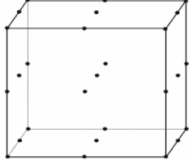
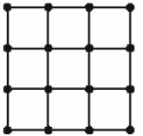
- i. **Mesh Generator:** The code has a built-in structured mesh generator, which allows meshing simple domains. The user can to modify the mesh and adapt it to others simples geometry. On the other hand the code can acquire meshes in files that agree with the preset format. The types of elements supported by the code are listed in Table 1. The integrations can be open or closed, with Gauss quadrant formulas for user defined points.
- ii. **Mesh partition:** Work distribution is achieved by partitioning the original mesh into sub domains (sub meshes) that run concurrently by the corresponding MPI processes. Mesh partitioning is carried out using METIS [2]. The mesh partitioning is done sequentially by the master while the workers wait for receiving their parts of the mesh.

In order to establish the communication scheme between processes, a common graphing algorithm is used. [1, 3]: firstly the nodes-elements adjacency graph is created, then for each node the elements it belongs to are obtained. Secondly the adjacency elements graph is created, to know the neighboring elements of each element.

After that METIS uses the adjacency elements graph and the number of sub-domains and gives an array assigning the sub domain to each element.

Based on this information, the communication arrays needed to exchange data between the workers are created. Finally the communication scheduling needed to

Table 1. I: Bi and tri-dimensional elements programmed in PRIMULA.

Lagrangian bi-dimensional	Tri-dimensional	
<p>Triangular three nodes</p> 	<p>Tetrahedrons four nodes</p> 	<p>Open or closed integration with triangular or tetrahedral coordinates.</p>
<p>Triangular six nodes</p> 	<p>Tetrahedrons ten nodes</p> 	
<p>Quadrangular four nodes</p> 	<p>Hexahedrons eight nodes</p> 	<p>Gaussian integration</p>
<p>Quadrangular nine nodes</p> 	<p>Hexahedrons twenty seven nodes</p> 	
<p>Quadrangular sixteen nodes</p> 		

determine in which order the workers have to interchange its data in pairs (through MPI_SendReceive) is computed. Once all this work is done, the distribution of the sub-meshes and corresponding element, boundary and node arrays to the corresponding workers is carried out (Fig. 2).

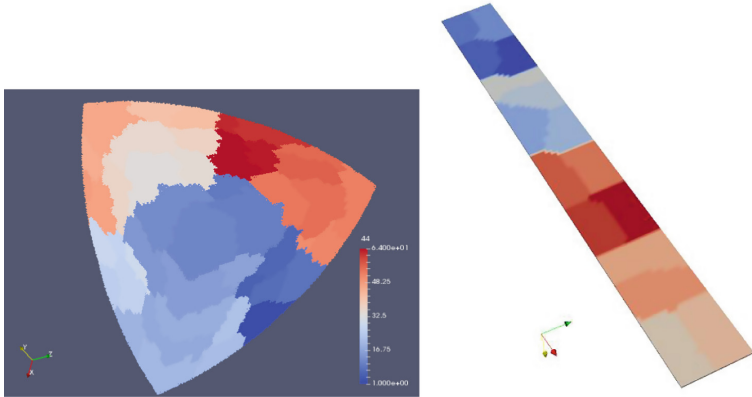


Fig. 2. Two tri-dimensional domains partitioned in 64 local regions using METIS.

iii. **Programmed systems:** PRIMULA is oriented to the resolution of stationary or time dependent equations in nonlinear partial derivatives. Field, bio-heat and mechanical equations for elasto-plasticity are presently codified (see Table 2), all types with Dirichlet, Neumann and Robin boundary conditions. Problems can be solved in two and three dimensions. In the case of two dimensions, approximations of plane stress, plane strain and axy symmetry are possible.

Table 2. scheme of the main equations solved in PRIMULA until now

System	Unknown	Representative Equation
Field	Φ (Field variable)	$\nabla \cdot (k(\Phi))\nabla\Phi + q = c(\Phi)\frac{\partial\Phi}{\partial t}$ (1)
Mechanical (Weak form)	u (displacements)	$\int_{\Omega} \bar{e}^T \bar{\sigma} d\Omega = \int_{\Omega} f_{\Omega}^{Ext} \bar{u} d\Omega + \int_{\Gamma} f_{\Gamma}^{Ext} \bar{u} d\Gamma + R$ (2)
Bioheat (coupled with (1))	T (temperature)	$\nabla \cdot (k(T, \Phi)\nabla T) - a(T)(T - T_a) + q + \lambda(\Phi, T) \nabla\Phi ^2 = 0$ (3)

Where k , c , a and λ are the non linear parameters of each physical problem and q corresponds to the volumetric heat source term.

iv. **Linear Equation Systems Solution:** The solver of the linear system is a gradient conjugated with Jacobi preconditioner designed to work in hybrid systems [4, 5]. Among the iterative methods, the Conjugate Gradient method is an algorithm for the numerical solution of particular systems of linear equations, for which matrix (A) is symmetric and positive-definite. Every iteration of the algorithm requires only a single Sparse Matrix-Vector multiplication (SpMV) and a small number of dot products.

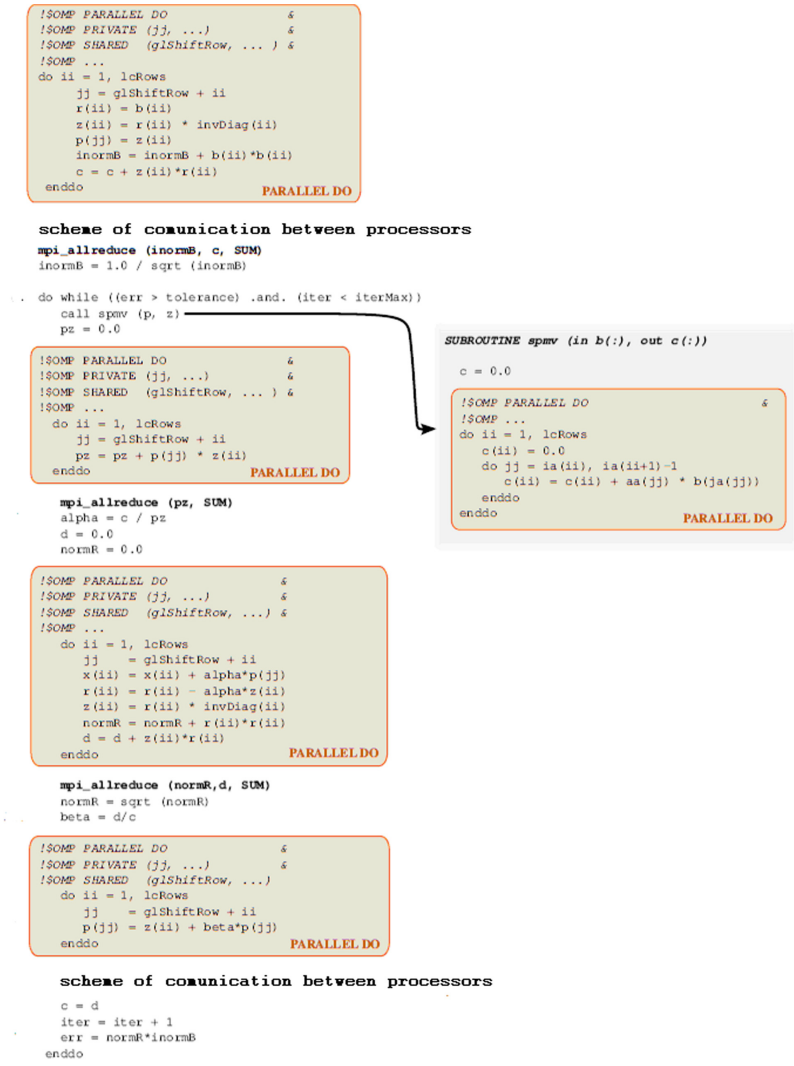


Fig. 3. Hybrid algorithm of JPCG. One matrix vector operation (spmv) and two reduction of dot product are needed in each iteration. After each iteration the communication between processors are needed in order to update the values of the coefficients in shared nodes.

The storage requirements are also very modest, since vectors can be overwritten. In practice, this method often converges in far fewer than a number of iterations less than the order of the matrix A. However, the conjugate gradient can still converge very slowly if the matrix A is ill-conditioned. The convergence can often be accelerated by preconditioning. The choice of P is crucial in order to obtain a fast converging iterative method. The Jacobi preconditioner is one of the most popular forms of preconditioning, in which P is a diagonal matrix with

diagonal entries equal to those of A . The advantages of this preconditioner are the facility of its implementation and the low amount of memory it needs.

A hybrid version of the JPCG was implemented in this work (see Fig. 3), establishing the parallelization across MPI of the matrix vector multiplication and the dot product using the portion of the matrix that each processor locally has. The algorithm also allows the OPENMP strategy (highlighted in the windows) of distribution in the internal and local operations in each processor. After each iteration the communication between processors are needed in order to update the values of the coefficients in shared nodes.

- v. **Post Process:** The entire post-process of the information generated in PRIMULA is adapted to the free distribution software PARAVIEW [6] under the data formats Comma Separate Values (CSV) and ENSI. ParaView is an open-source, multi-platform data analysis and visualization application. ParaView was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze datasets of petascale as well as on laptops for smaller data. The ParaView flexibility allows developers to quickly create applications that have specific functionality for a specific problem domain (Fig. 4).

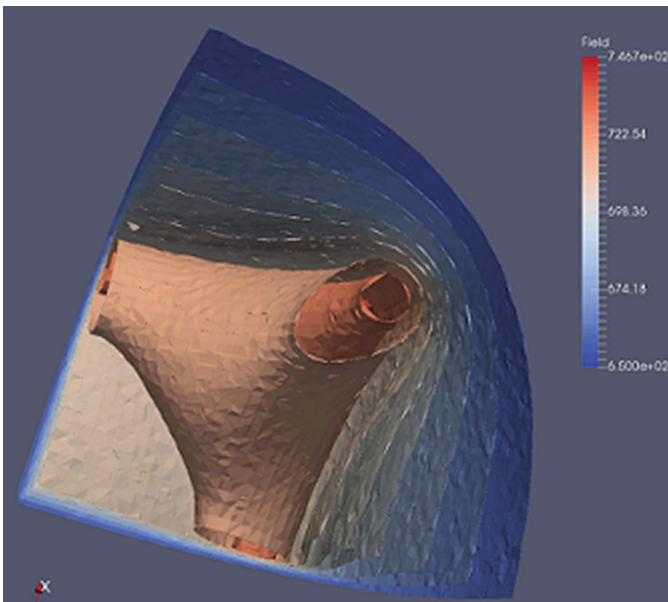


Fig. 4. A contours plot data from a field problem solved using PRIMULA.

- vi. **Parallel environment:** The code is designed to be run on hybrid machines, taking advantage of the availability of multiprocessors that share their node with multiple threads. In the case of TUPAC, each node is a 4x Hexadeca core AMD Opteron

6276s, with 64 threads and 8 gigabytes of RAM. In this way a correct use of this computing capacity must have the possibility of combining algorithms that work in a distributed and shared form. To this end, the standards Message Passing Interface [22] and OPENMP [21] will be used. The user selects the parallelization mode he wants to use and the amount of threads and MPI processes needed for his problem through input files.

3 Scalability Results

Intel Xeon(R) E5-2630 v3 2.4 Hgzs: Has 8 independent processing units (cores) and 2 threads per core, giving a total of 16 working threads. This allows using MPI and OPENMP or some combinations of both. In particular, to program PRIMULA in Windows OS we use FORTRAN in the VISUAL studio environment Visual Studio Pro 2012 [23]. Also we use MS-MPI [24] for distributed memory functionalities and OPENMP feature that by default contains the VS.

The main limitation that a code that handles large amounts of data possesses in the Windows environment is memory. Manipulating the variable STACK is inevitable if it is intended to work with a system of several million elements. On the other hand, there is a physical limit that can not be crossed and eventually works as a limiting problem to be established by RAM. Nevertheless, work in a desk computer with systems with several million of nodes efficiently, already is acceptable given the comfort they offer.

We have analyzed PRIMULA with a field problem (see Table 2) for domains with approximately one million variables obtaining some interesting results, plotted in Fig. 5 where the speedup for different code portions is showed. First of all, the solver of JPCG was always measured independently, since is the more demanding portion of the code. We plot too the pre-processing time, the general calculation zone (that includes the solver but also the time used in assembly the finite element matrix) and a measure of the total computing time of the code.

Analyzing these graphs, the first thing we observed is the region of super-scalability for the limit $p \leq 4$. It is interesting to note that the solver PJCG works optimally in this type of machines with a number of threads close to 4. From there the scalability is reduced to 86% with 8p and 58% with 16. The total computer time has an acceptable speedup up to 8 processors while the performance is greatly reduced to 16 where both the pre-process and the calculation saturate the computation times.

AMD OPTERON 6200: With 4 Hexadeca cores whereby each node has 64 threads. Two ways of executing the code were measured in this graph. First the application was executed in fully distributed mode, using pure MPI (plotted with dotted lines). Secondly, the code was run in hybrid form, with MPI + OPENMP, with 64 shared memory threads per node. We use an example of approximately 30 million elements for the calculation. There are several points in this comparison that are shown jointly in Figs. 6 and 7:

- (i) The solver time in distributed mode has scalability similar to that of the hybrid mode. However it is remarkable how the distributed mode loses scalability in the section that we call calculus, which is the arming of matrices and calculation of all the coefficients by elements. Clearly the use of hybridization reduces the time consumed in communication between processes and accelerates the computation locally.

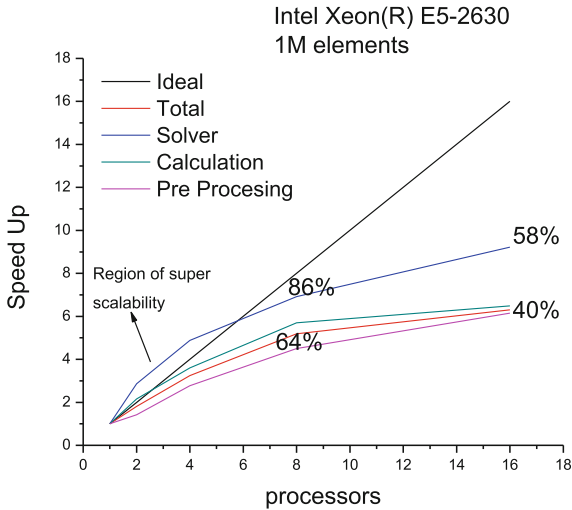


Fig. 5. Speed up for systems of 1M of elements approximately.

- (ii) It is observed in the graph that for the pure MPI mode, we obtain compute times for 32 and 64 processors. However, it is not possible to obtain reasonable computing time for the hybrid case. This is because dividing the general domain (2 nodes) into two local parts, the amount of information of a mesh of 30 million degrees of freedom that must be transferred between nodes is so large that the calculation is notoriously slowing down. On the other hand, structures also grow in size considerably so memory management is also affected. In fact already for the case of 2 nodes (128 threads) the processing time increases disproportionately, (see Fig. 7). For the hybrid case, it improves the performance of the solver for 1024 processors in relation to 512. This is related to the improvement in communication that is established for 16 nodes with respect to that of 8 MPI nodes. The graph colouring algorithm shows that there is an optimal number of divisions to minimize communication in the global domain and in the case of the type of problem we are solving, this happens for that number of 16.
- (iii) Figure 7 shows the pre-processing time each case consumes. We expect a similar time consumed in pre-processing for all the systems, with a grow of time consumed with the number of communication, but in case of 2 nodes plus 64 threads we highlight the fact mentioned in point (ii) about the time of post-processing in a hybrid case for low number of nodes. We have also added a point for 2048 processors. This point, although it degrades scalability

dramatically for the analyzed case, shows that PRIMULA works correctly for that number of processors. Finally, this graph tells us that the pre-processing times are lower for the hybrid case. This is for the same reason observed in item (i) in where we saw better calculation times in hybrid systems.

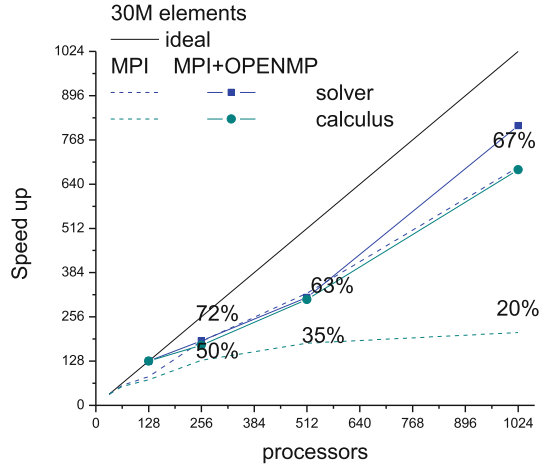


Fig. 6. Speed up for a system of 30 Millions of elements.

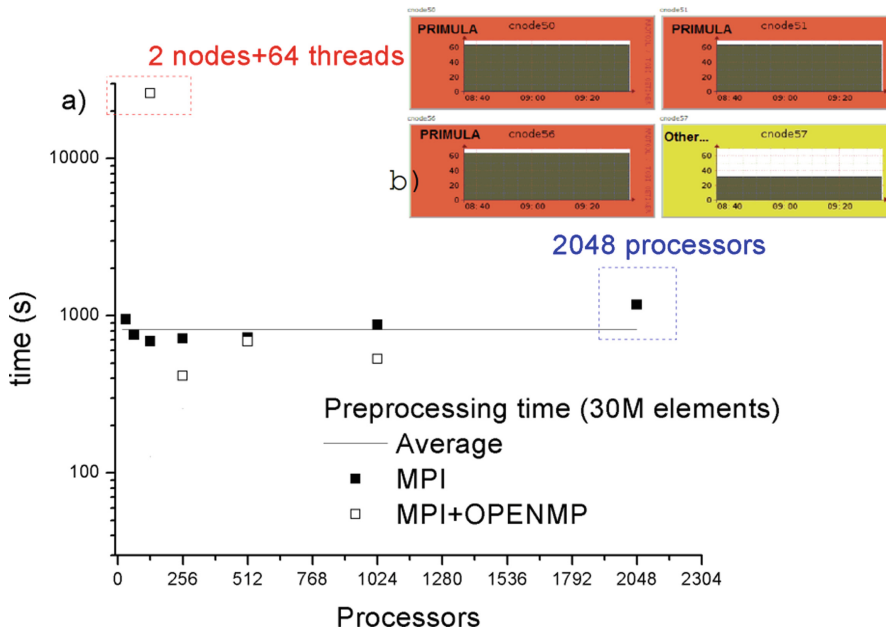


Fig. 7. (a) PRIMULA pre-processing times for a 30M system (approx). In the case 2 nodes and 64 threads the pre-processing time increases disproportionately due to the size of the generated arrays. (b) Use of processors per node in selected cores done by PRIMULA (Color figure online)

- (iv) In Fig. 7(b) we presents some results performed by the monitoring system GANGLIA that analyze cluster behaviour on real time. We follow the local behaviour of PRIMULA running on the highlighted nodes. Is necessary to note that if an efficient use of the cluster is wanted, it is advisable to write applications that use the 64 threads of each node, since the access of each user to each node is individual. That's mean that if one user occupied one node alone using only 50% of the computer capacity (as in the case highlighted in yellow) is a waste of computing time with the consequent wasted cost in cash.

4 An Example of Field: PLATE Fuel

The fuels used by the vast majority of research reactors currently in operation are so-called dispersed fuels, which consist of particles of a uranium compound dispersed in a matrix of a metal (generally aluminium), which ensures a good extraction of heat of the combustible particle. This material thus formed is sheathed by co lamination between two metal plates (also generally aluminium) to be introduced into the reactor. This type of fuel is built by joining a number of plates into prismatic cavities that will occupy certain positions within the core. Other plate configurations may curve or form rings, always looking for an optimal neutron flux distribution, better dispersion of generated heat, plus adequate mechanical stability of the system. (See Fig. 8).

One of the most important codes that make up the DIONISIO package refers to the so-called PLACA3D, whose function is to simulate the behaviour of a combustible plate under irradiation in a research reactor in operation. The different phenomena that occur in each of the plate materials are motorized by temperature. It is generated inside the fissile material inside the plate and the heat is extracted through the passage of water by the outer coating, generally constructed of aluminium. Note that a combustible plate has a dimension of 1.4 mm thick, with a region of material fisil of 750 microns of thickness, 65 mm wide and 650 mm long, thus dimensionally establishing a complicated domain to solve due to the differences in scale of its geometry. On the other hand, physically interesting problems occur within the fissile region, due to being the zone of fission heat production, but in addition, the temperature on the plate must be determined through a water contour condition flowing at the rate of about 10 m per second. On the other hand, the plate is growing as time passes in the reactor a layer of oxide that functions as a thermal insulation. This complex problem with models at such dissimilar scales must be solved in more or less discrete time steps.

As the first field test of the PRIMULA framework we used the geometry of a discretized fuel plate with 1.256 million elements. The objective of this first test was to analyze the performance of the framework in a unique analysis of nonlinear temperature, to obtain the temperature distribution over the whole domain with Robin boundary conditions and a stationary generation of heat. Note that the complete history of a fuel within a reactor may range from 140 to 160 days of permanence, which is usually equivalent to a number of numerical steps of approximately 200.

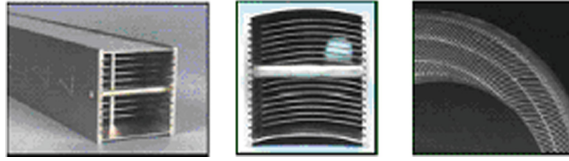


Fig. 8. Different types of fuel elements for research reactors: straight plates, curves or forming rings.

In this first estimation it was established that the calculation time by time step for a purely MPI computation with 64 processors (a TUPAC node) is approximately 36 s, so a complete history of this fuel element will oscillate between 2 and 2:30 h of computation. Figure 9 shows the temperature of the refrigerant, that obtained on the oxide layer that covers the plate and the temperature on the outside of the aluminium plate. In the inferior part the central temperature of the plate with the scale that appears in the same graph is presented. These values are correlated with knower experimental and analytical results.

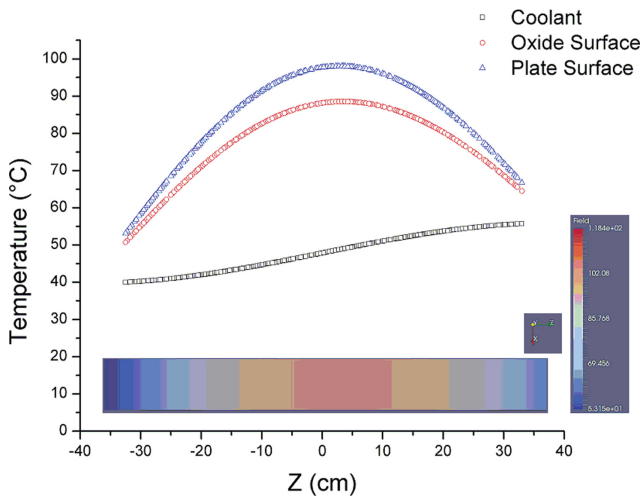


Fig. 9. Graph of temperature on the refrigerant, on the rust cover and on the aluminum cladding of the analyzed domain. The lower graph represents the central temperature of the plate.

5 Conclusions

Throughout this work we have given a first bounded description of the PRIMULA code, designed as a framework adaptable to problems of numerical resolution of differential equations in partial derivatives by the finite elements method. It responds to the usual denominations of multiscale and multiphysical, given the quality and precision with which it solves coupled problems of different dimensionalities and described physics through models of one two and three dimensions.

The code respects other interesting features such as:

- (i) It works for any operating system with minimal modifications in its compilation.
- (ii) It works in series or in parallel with minimal modifications in its input variables.
- (iii) Works for bi and tri-dimensional geometries.
- (iv) It is designed for any user to program their problem with a minimal intervention in the distribution of the particular information. That is, the user does not intervene in the distributed manipulation of its geometry or in the solver of its system.

Scalability analyzes have been performed for field problems, obtaining in different operative systems, results that justify its use given the savings in computing time that PRIMULA provides. Due to the characteristics of the problem analyzed and the type of solver used, a scalability that reaches 67% for 1024 processors is adequate.

The code is by no means finished and in fact as already warned in the introduction, it does not pretend to ever reach its final state. However there are some items in which we must continue working to make it a robust tool:

- (i) Firstly, the supply of linear system solvers available in the code should be expanded. For this purpose, it is planned to include the possibility of using the Petsc package in addition to programs other solvers of the type of Krilov spaces with specific preconditioners optimized for distributed calculation.
- (ii) The finite element types and the shapes functions offered in the code must be expanded. This is fundamental to more accurately encompass complicated domains.
- (iii) The number of systems treated by PRIMULA should be increased considerably. We are working on the inclusion of field equations coupled with thermal phenomena and also on including equations of fluid mechanics such as the incompressible Navier Stokes equation. However the list is still short and this point should be a priority if PRIMULA is intended to compete with packages already established.

In short, the concept of PRIMULA will not be exploited in all its possibilities until the number of users grows.

Acknowledgment. I would like to thank CSC researchers and the TUPAC team of administrators for ongoing support they provided me during all of PRIMULA's writing, installation and scalability analysis tasks.

References

1. Artigues, A., Houzeaux, G.: Parallel mesh partitioning in alya. www.prace-ri.eu
2. Metis: Serial Graph Partitioning and Fill-reducing Matrix Ordering (metis documentation)
3. Coloración de Grafos. María Rosa Murga Díaz. Tesis de Grado. U. de Cantabria (2013)
4. Shewchuk, J.R.: An introduction to the conjugate gradient method without the agonizing pain. Carnegie Mellon University (1994)

5. Sáez, X., Soba, A., Sánchez, E., Kleiber, R., Castejón, F., Cela, J.M.: Improvements of the particle-in-cell code EUTERPE for petascaling machines. *Comput. Phys. Commun.* **182**(9), 2047–2051 (2011)
6. www.paraview.org
7. tupac.conicet.gov.ar/stories/home/
8. Zienkiewicz, O.C., Taylor, R.L.: *The Finite Element Method*, vol. 1, 2 & 3. Butterworth Heinemann, Oxford (2000)
9. slurm.schedmd.com/
10. ganglia.sourceforge.net/
11. www.openfoam.com/
12. www.bsc.es/es/computer-applications/alya-system
13. www.3ds.com/products-services/simulia/products/abaqus/
14. www.ansys.com/Products/Fluids/ANSYS-Fluent
15. www.wrf-model.org/index.php
16. Soba, A., Denis, A.: Simulation with DIONISIO 1.0 of thermal and mechanical pellet-cladding interaction in nuclear fuel rods. *J. Nucl. Mater.* **374**, 32–43 (2008)
17. Lemes, M., Soba, A., Daverio, H., Denis, A.: Inclusion of models to describe severe accident conditions in the fuel simulation code DIONISIO. *Nucl. Eng. Design* **315**, 1–10 (2017)
18. Soba, A., Denis, A., Lemes, M., González, M.E.: Modelado del comportamiento del combustible nuclear bajo irradiación mediante DIONISIO 2.0 *Revista de la CNEA*, Vol. 53–54 (2014)
19. Soba, A., Denis, A., Romero, L., Villarino, E., Sardella, F.: A high burnup model developed for the DIONISIO code. *J. Nucl. Mater.* **433**, 160–166 (2013)
20. Soba, A., Denis, A.: PLACA/DPLACA: código para la simulación de un combustible tipo placa monolítico/disperso. *Rev. Int. Mét. Num. Cál. Dis. Ing.* **23**(2), 205–224 (2007)
21. www.openmp.org/
22. www.open-mpi.org/
23. *Visual_Studio_Pro_2013*. <https://www.visualstudio.com/es/> (License 62739385 COM.NAC. DE ENERGÍA ATÓMICA)
24. [msdn.microsoft.com/en-us/library/bb524831\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/bb524831(v=vs.85).aspx)