XLI CILAMCE
XLI Ibero-Latin American Congress on Computational Methods in Engineering
16-19 November, 2020 FOZ DO IGUAÇU PARANÁ - BRAZIL

Performance portability in meshless methods using a parallel implementation procedure based on meta-programming

Marlucio Barbosa[1], Edmundo Guimarães de Araújo Costa[1], Jose Claudio de Faria Telles[1], Jose Antonio Fontes Santiago[1], Edivaldo Figueiredo Fontes Junior[2]

[1]*Civil Engineering Programme of COPPE, Universidade Federal do Rio de Janeiro*
Caixa Postal 68506, CEP 21941-972, Rio de Janeiro, RJ, Brazil
marlucio@coc.ufrj.br, telles@coc.ufrj.br, santiago@coc.ufrj.br, edmundo_costa@coc.ufrj.br
[2]*Department of Mathematics, Universidade Federal Rural do Rio de Janeiro*
UFRRJ, CEP 23897-000, Seropédica, RJ, Brazil
fontesjunior@ufrrj.br

Abstract. The present paper focuses upon the development of a parallel implementation procedure to the numerical simulation of boundary value problems by using the principles of functional programming and memory polymorphism. The proposed procedure aims to evaluate the portability of computational performance in different execution spaces using meta-programming and simplified representation of physical-mathematical models through simple usage of declarative programming. For this purpose, an execution model for meshless methods is presented, under the premise that specializations for the execution and memory spaces must occur during the compilation time. This means that the declared model can be simulated in distinct architectures without any changes in the implementation, allowing ensuring the portability of the computational performance and reducing the complexity of the meshless numerical models. At the end of the paper, an example is carried out aiming to evaluate the computational performance and the efficiency of the parallel implementation procedure applied to the Helmholtz problem.

Keywords: meshless method, parallel implementation procedure, functional programming

1 Introduction

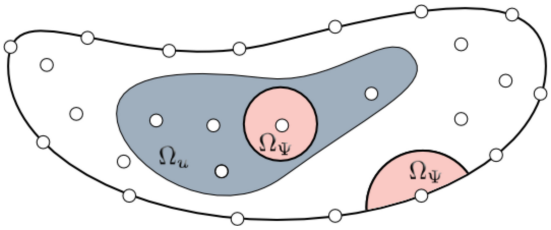
Recently, many meshless methods have been widely applied to the numerical simulation of several engineering problems. These methods have the ambition of avoiding predefined relationships between nodal points for reducing the computational requirements resulting from the change of topology during the process for obtaining a solution. Indeed, the main objective of this approach is to obtain greater flexibility for complex domains, which are changing during the calculation and new boundary conditions are enforced. In effect, the representation of these domains must be adapted (see Berger [1], Berger et al. [2], Hughes et al. [3]) for extreme cases, in which mesh-based methods are to be used. Such strategy can significantly increase the requirements of the solution processes. By definition, the meshless methods are generally more suitable in this context. Nevertheless, the price for this is a mathematically more complex formulation and an increased computational cost in its constitutive algorithms, particularly when weak formulations are used.

Therefore, this work aims to propose advanced techniques to accelerate the processing of meshless methods through parallel or massively parallel processing, particularly, in its weak forms. The proposed methodology is based on advanced functional programming techniques for reducing the complexity of computational implementation, especially when parallel computing using hybrid architectures are considered. In this sense, a discussion on the Meshless Local Petrov-Galerkin (MLPG) method is presented in section 2. The proof of concepts that relates the functional programming and memory polymorphisms is presented in sections 3 and 4. The present programming model, which aims for obtaining the performance portability for different architectures, is presented in section 5. In section 6, a numerical example is carried out and a detailed discussion on an implementation of the proposed methodology is presented and, in section 7, conclusions are presented.

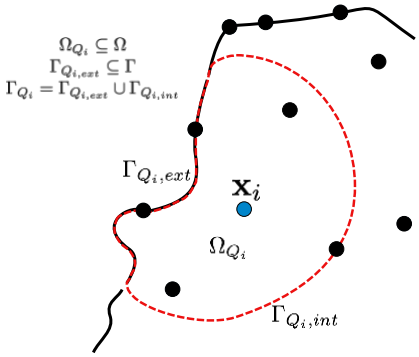
2 The MLPG method

The MLPG is a numerical method based on weak formulation and can be obtained by means of the weighted residual method. Unlike the Finite Element Method (FEM), in which the trial and test functions are chosen from the same space, the MLPG uses the trial function and test function from different space (see Atluri and Zhu [4]). This group of formulations makes use of the Petrov-Galerkin concept, in which the test functions can be different (subdomain Ω_Ψ) from the trial function (subdomain Ω_u), as shown in Fig. 1a.

The test functions can be defined so that it is different from zero in one subdomain and is equal to zero anywhere else in the considered domain. Furthermore, it assumes that in the MLPG, the domain is already discretized by using a cloud of arbitrarily dispersed points, without any imposition of structure on them. As seen, this type of formulation is natural for meshless methods. The local subdomain, known as the support domain, can be prescribed for each point-cloud by means of a simple regular shape that represents the support of the test function (see Fig. 1a), where it is different from zero.



(a) Definition of subdomain of the trial and test functions for the MLPG.



(b) Relationship between the quadrature and global domains belong to the boundary.

Figure 1. Domain definitions of the MLPG.

Atluri and Shen [5] propose a classification of MLPG formulations, which differ from each other in the form of the chosen test and trial functions and, consequently, are designed in the literature by MLPG-1, MLPG-2, MLPG-3, MLPG-4 (or LBIE), MLPG-5 and MLPG-6.

Following to formulation proposed by Liu and Gu [6] and by Trobec and Kosec [7], and bearing in mind that the test function can to be any non-null function, the following equation can be written:

$$\int_{\Gamma_{Q_i}} \mathscr{L}^{(r-1)}(\hat{\mathbf{u}})\mathbf{n}_x \hat{W}_i d\Gamma_{Q_i} - \int_{\Omega_{Q_i}} \mathscr{L}^{(r-1)}(\hat{\mathbf{u}})\hat{W}_{i,x} d\Omega_{Q_i} - \int_{\Omega_{Q_i}} g(\mathbf{x})\hat{W}_i d\Omega_{Q_i} = 0, \tag{1}$$

where Ω_{Q_i} represents the domain of integration associated to the point $\mathbf{x}_i$.

Equation 1 represents the expression that needs to obtain the weak form to the point $\mathbf{x}_i$. Therefore, considering the particular interest in MLPG-1, it is arbitrated that the test functions $\hat{W}_i$ is equivalent to the MLS weight function. In this case, should be noted that the integral over the boundary Γ_{Q_i} can be non-null only when the intersection between the quadrature domain and the boundary of the global domain Γ occurs (see Fig. 1b), that is, the integral over Γ_{Q_i} comes down to line integral over $\Gamma_{Q_{i,ext}}$ since the line integral over $\Gamma_{Q_{i,int}}$ is equal to zero by definition of the weight function $\hat{W}_i$.

Consider that the approximate solution corresponds to eq. (2):

$$\hat{u}(\mathbf{x}) = \sum_{\mathbf{x}_j \in \Omega_S(\mathbf{x})} \phi_j(\mathbf{x}) u_j, \tag{2}$$

where $\phi_j(\mathbf{x})$ is the function associated with node j, belonging to the domain of influence $\mathscr{D}_i$ of node $\mathbf{x}_i$.

Substituting eq. (2) in eq. (1), we have:

$$\sum_{x_j \in \mathscr{D}_i}^{N} u_j \left(\int_{\Omega_{Q_i}} \mathscr{L}^{(r-1)}(\phi_j) \hat{W}_{i,x} d\Omega_{Q_i} - \int_{\Gamma_{Q_i}} \mathscr{L}^{(r-1)}(\phi_j) \mathbf{n}_x \hat{W}_i d\Gamma_{Q_i} \right) = \int_{\Gamma_{Q_i}} \bar{h}_i \hat{W}_i d\Gamma_{Q_i} - \int_{\Omega_{Q_i}} g(\mathbf{x}) \hat{W}_i d\Omega_{Q_i}. \tag{3}$$

Atluri and Shen [8] present an approach for the MLPG-1 based on the penalty method for imposing essential boundary conditions. Thus, eq. (3) can be rewritten of the following form:

$$K_{ij} = \begin{cases} \int_{\Omega_{Q_i}} \mathscr{L}^{(r-1)}(\phi_j) \hat{W}_{i,\mathbf{x}} d\Omega_{Q_i} - \int_{\Gamma_{Q_i}} \mathscr{L}^{(r-1)}(\phi_j) \mathbf{n}_{,\mathbf{x}} \hat{W}_i d\Gamma_{Q_i} & : \mathbf{x}_i \notin \Gamma_e, \mathbf{x}_j \in \mathscr{D}_i \\ \int_{\Omega_{Q_i}} \mathscr{L}^{(r-1)}(\phi_j) \hat{W}_{i,\mathbf{x}} d\Omega_{Q_i} - \int_{\Gamma_{Q_i}} \mathscr{L}^{(r-1)}(\phi_j) \mathbf{n}_{,\mathbf{x}} \hat{W}_i d\Gamma_{Q_i} + P & : \mathbf{x}_i \in \Gamma_e, \mathbf{x}_j \in \Omega_{S_i} \\ 0 & : \text{ otherwise} \end{cases} \tag{4}$$

and

$$f_i = \begin{cases} \alpha \int_{\Gamma_{Q_i}} \overline{\mathbf{u}}(\mathbf{x}_i) \hat{W}_i d\Gamma_{Q_i} - \int_{\Omega_{Q_i}} g(\mathbf{x}) \hat{W}_i d\Omega_{Q_i}(\mathbf{x}_i) & : \mathbf{x}_i \in \Gamma_e \\ \int_{\Gamma_{Q_i}} \bar{h}_i \hat{W}_i d\Gamma_{Q_i} - \int_{\Omega_{Q_i}} g(\mathbf{x}) \hat{W}_i d\Omega_{Q_i} & : \mathbf{x}_i \in \Gamma_n \\ - \int_{\Omega_{Q_i}} g(\mathbf{x}) \hat{W}_i d\Omega_{Q_i} & : \text{otherwise} \end{cases} \tag{5}$$

where $P = \alpha \int_{\Gamma_{Q_i}} \phi_j \hat{W}_i d\Gamma_{Q_i}(\mathbf{x}_i)$, $\bar{u}_i = \overline{u}(\mathbf{x}_i)$, $\bar{h}_i = \overline{h}(\mathbf{x}_i)$ and a penalty parameter $\alpha \gg 1$ is used to impose the essential boundary conditions.

It is important to note that $K_{ij} = 0$ for any node $\mathbf{x}_j$ that is not present in the support domain of any point belonging to Ω_{Q_i}. In other words, the K matrix is sparse.

Notice that the designer can achieve greater productivity if he can adopt a computing model that makes the implementation strategy for eq. (4) and eq. (5) transparent. Indeed, the designer can perform the implementation independently of the differential operator, the shape function and the test function adopted, providing only the mathematical expression for assembling a group of lines of the matrix K and of the vector f. In addition, if the model allows hardware scalability, the designer can obtain a computational performance gain for more complex cases.

3 Functional Programming

The definition of functional programming is not consensual among functional programmers, being the same defined in several ways and in most cases using concepts of functional programming. In order to synthesize the authors' thinking, the definition of a functional program is given according to the Hindley and Seldin [9]:

Definition 3.1 *A functional program consists of an E expression (representing both the algorithm and the input). This E expression is subject to some rewriting rules. The reduction consists of replacing a part P of E with another expression P, according to the rules of rewriting given. In schematic notation $E[P] \rightarrow E[P']$, as long as $P \rightarrow P'$ is in accordance with the rules. This reduction process will be repeated until the resulting expression has no more parts that can be rewritten. This so-called normal form E^* of the expression E consists of the output of the given functional program.*

Based on described above definition, functional programming can be understood as a programming style in which the main building blocks of the program are functions, instead of objects used in object-oriented programming and procedures used for programming structured. A program written in the functional style does not specify the commands that must be executed to achieve the result, but defines what the result is.

The concept presented can be extrapolated, for example, for numerical integration. In fact, instead of defining step by step how to calculate the numerical integral, we indicate what the numerical integral is. Note that functional programming requires a change in the way of thinking of the programmer, but this comes with some benefits, such as: readability, testability, parallelism. That is, the source code is easier to understand and test; in addition, as the functions are self-contained, they can be easily parallelized.

Programmers looking for extreme performance, usually opt for imperative and established languages like Fortran and C/C++. On the other hand, recent advances in compiler technology make use of functional programming

principles in multi paradigm languages. That said, the C ++ language deserves to be highlighted in its evolution, whose introduction of templates in the language and the creation of the Standard Template Library (STL), which uses only inheritance functions and virtual members, made it a multi paradigm language.

4 Memory and execution space polymorphism

For a long period of time in the history of computing, researchers sought to develop computational techniques that would allow the same source code to generate binaries compatible with different operating systems and/or hardware. This principle that guided the development of several technologies is known as portability.

On the other hand, to say that a code is portable does not mean to say that its performance is also portable. As for example, some scalable algorithms or techniques in CPU do not obtain the expected performance in GPU and vice versa. Indeed, in order to ensure that the same source code obtains optimum performance in the same architecture, there is a need for a minimum guarantee of morphism between the implemented algorithms and the memory model of the execution space. Therefore, if it is desired that the implementation obtains a similar performance in different architectures, it is necessary that it be able to adapt each target of the architecture of execution, in a kind of polymorphism.

The conventional algorithms that aim to obtain high performance are implemented aiming at the execution space and its specificities, as a memory access standard. In such a way that, even if it is possible to compile and execute in different architectures, the performance achieved does not often follow the gain curve of the initial execution space. In this sense, writing source code that, when compiled in a specific architecture, manages to obtain the best performance in a different architecture is a complex and highly expensive task.

Edwards et al. [10] present the Kokkos library in order to provide a memory polymorphism model that allows performance portability across different architectures, whether the same multicores or many cores. This library aims to allow an implementation to compile and run on different architectures, obeying the high performance memory access standards of the target architecture and taking advantage of the architecture's specific resources whenever possible. In order to achieve the proposed objective, Edwards et al. [10] provide a polymorphic data layout that can be mapped in the most diverse multicore and many core architectures. In other words, KOKKOS implements a set of back ends that map the source code provided to a low-level for a specialized portable model.

The polymorphism model of the memory space proposed by Edwards et al. [10] is adopted in this work in conjunction with metaprogramming for a specialization, of fine granularity, which aims to obtain the best performance in a specific execution space.

5 Functional Model

One of the merits of functional programming is to move the focus from how something is done to what it means. In other words, considering as an example the numerical integration of a function $h(\mathbf{x})$, the focus is on its definition, which is given by the reduction of two lists (weight and abscissa) by the product between weight, the function h evaluated on the abscissa and the Jacobian determinant corresponding to spatial transformation.

Consider as a premise the need for an implementation to be able to be specialized for the execution and memory spaces targeted by the compilation. In other words, based on the weak form of eq. (1) and the flow described in Fig. 2a, the designer defines the core of the integrals and, through metaprogramming, a programming model efficiently maps the definitions to the architecture of the target space of execution.

In this sense, it is essential to have a series of functions (or function objects), self-contained, in order to deliver to the designer the data necessary for the processing of the integral's core, in compilation/execution time, through a friendly interface.

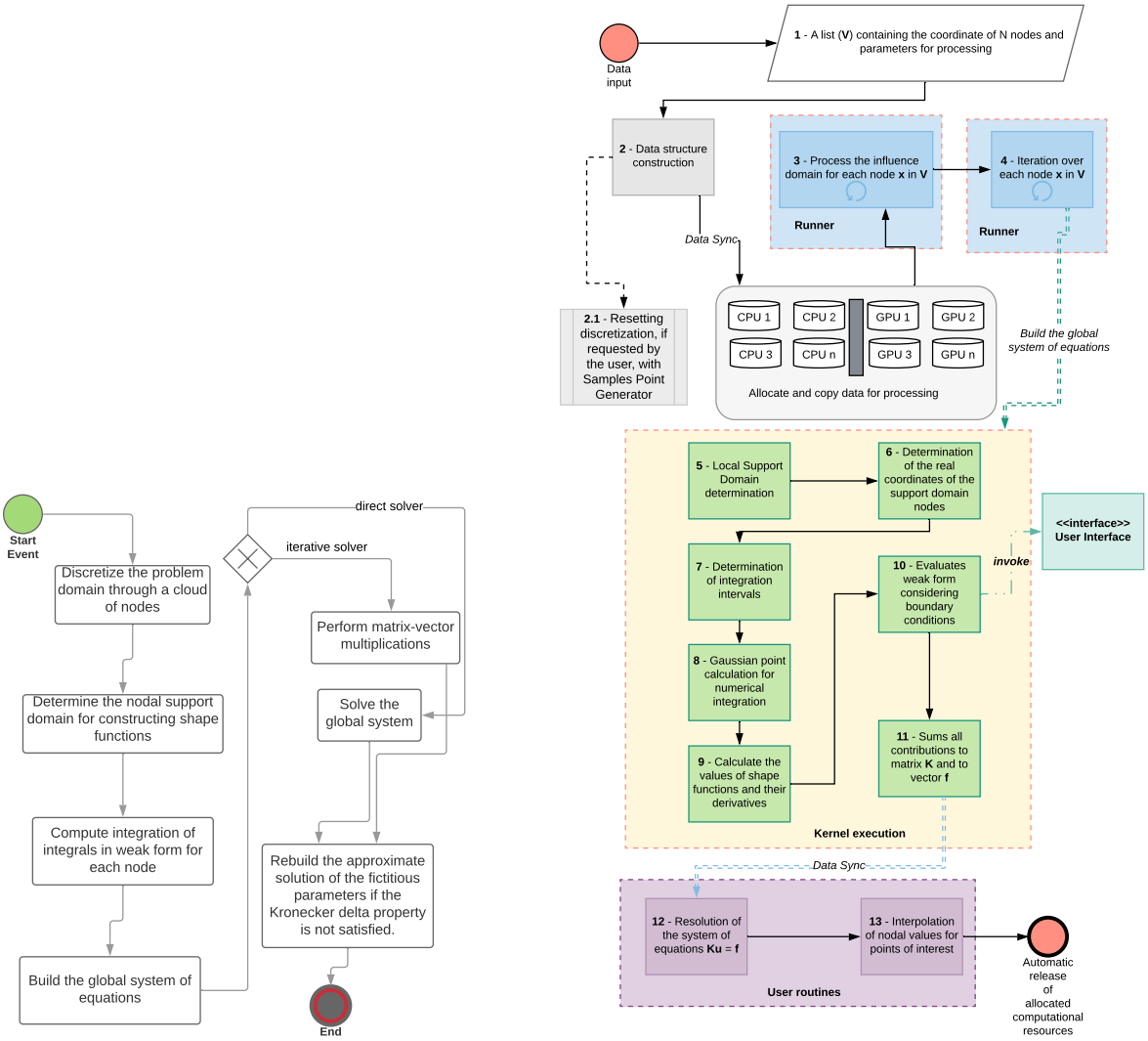
Figure 2b summarizes the process of executing the proposed model. It should be noted that the execution of *Data Sync* operations occurs if and only if the execution space is distinct from the space where the data was initially allocated. For example, data is provided by a multicore processor-based architecture for processing in many core architecture with different memory arrangements. *Data Sync* operations can be highly costly depending on the volume of data. Thus, a functional controller makes the orchestration of these synchronisms transparent to the designer and the end user, in order to minimize the complexity of the problem.

The aforementioned implementations must be made in a generic way, leaving the description of the differential operators and integral cores under the responsibility of the user interface. That said, the user can implement any solutions based on the defined numerical method, having to implement the necessary pure functions in the user interface.

In problems where the size of the necessary allocations is incompatible with the memory availability of the target architecture, the functional orchestrator partitions, whenever possible, the execution in smaller work

packages, using the divide-and-conquer algorithm. The strategy to pre-allocate spaces, previously, aiming to reduce the high cost of allocation existing in some types of architecture, thus avoiding performance degradation.

At this point, the use of the view types, built at compile time (template meta-programming), are responsible for the memory polymorphism and for mapping to the target execution architecture.



(a) Meshless methods based on the weak form.

(b) Process of execution of the strategy.

Figure 2. Meshless methods based on the weak form and the proposed implementation model.

6 Numerical results

In order to validate the proposed implementation model, numerical results are compared with the ones given by analytical solution reported in the literature. For this proposal, consider that the explained differential equation is solved according to the procedure described in section 2 and that the end user offers the expressions related to equations 4 and 5 through the user interface. The FSerial, FOpenMP and FGPU program corresponds, respectively, to the compilations for the execution space formed by a single core processor, a multicore processor and an NVIDIA GPU. The numerical simulations use a computer composed of an Intel Xeon E5-2690 v3 2.60GHz v3 (Broadwell) processor, 110 GB of RAM and NVIDIA Tesla V100 GPU.

6.1 Example - Helmholtz problem

In this case study, consider the Dirichlet-Helmholtz problem defined over the rectangular domain $[-1, 1]^2$

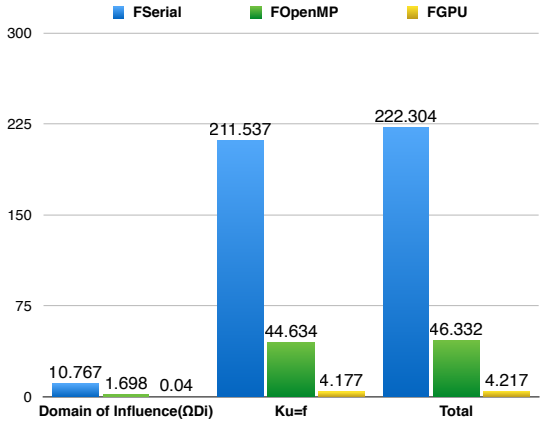
$$\nabla^2 u + k(x, y)u = f(x, y) \tag{6}$$

where the variable wave number is defined by

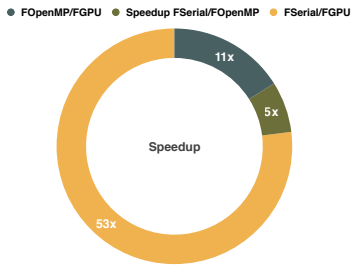
$$k(x, y) = (x^2 + (y + 1)^2)sin(x(y + 1))^2 \tag{7}$$

with the analytical solution $u(x, y) = cos(cos(x(y + 1)))$.

Figure 3a shows the execution time for the assembly of the Ku = f system for each simulation execution space, using a discretization containing 65536 knots. Figure 3b compares the performance gain obtained between the serial version and the parallel versions. Note that by specializing the code for each architecture, the programming model is able to persist the approximation error, but obtaining a performance gain if the target architecture of the compilation has more units for processing.



(a) Runtime for assembly Ku=f.



(b) Comparison between the exact and the numerical solutions.

Figure 3. Runtime for assembly of system and speedup.

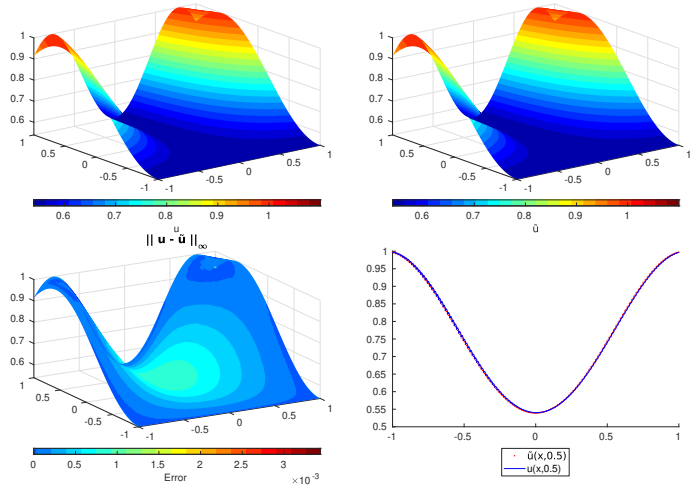


Figure 4. Comparison between the exact and the numerical solutions.

Figure 4 shows the numerical solution obtained and compared to the analytical one, processed in the same point cloud. The infinite norm is adopted in the construction of the graph aiming at a better nodal comparison between the exact and approximate solutions.

7 Conclusions

The experiments showed that the model gains performance as the number of available processing units increases. Or rather, the models scaled performance according to the execution space. This provides to the designer the ability to implement a computationally efficient numerical simulation, using meshless methods. In this point, it stands out that the requirement to be a high-performance computing expert is removed from the designer, simplifying the implementation of the numerical methods. In addition, the proposed model added in its architecture a range of algorithms that allows solving common problems of engineering. This allows minimizing prospecting and simulation times through implementations that guarantee both performance portability and functional isolation.

This analysis made it clear that the parallel implementation procedure based on functional programming could be a very interesting tool for the numerical simulation of boundary value problems using meshless methods.

Acknowledgements. The authors acknowledge the financial support of the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq). This study was also financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] Berger, M. J., 1982. Adaptive mesh refinement for hyperbolic partial differential equations. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE.

[2] Berger, M. J., Colella, P., et al., 1989. Local adaptive mesh refinement for shock hydrodynamics. *Journal of computational Physics*, vol. 82, n. 1, pp. 64–84.

[3] Hughes, T. J., Cottrell, J. A., & Bazilevs, Y., 2005. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering*, vol. 194, n. 39-41, pp. 4135–4195.

[4] Atluri, S. N. & Zhu, T., 1998. A new meshless local petrov–galerkin (mlpg) approach in computational mechanic. *Computational Mechanics*, vol. 22, n. 2, pp. 117–127.

[5] Atluri, S. N. & Shen, S., 2005. The basis of meshless domain discretization: the meshless local petrov–galerkin (mlpg) method. *Advances in computational mathematics*, vol. 23, n. 1-2, pp. 73–93.

[6] Liu, G.-R. & Gu, Y.-T., 2005. *An introduction to meshfree methods and their programming*. Springer Science & Business Media.

[7] Trobec, R. & Kosec, G., 2015. *Parallel Scientific Computing*. Springer International Publishing, London, 1 edition.

[8] Atluri, S. N. & Shen, S., 2002. The meshless local petrov-galerkin (mlpg) method: a simple & less-costly alternative to the finite element and boundary element methods. *Computer Modeling in Engineering & Sciences*, vol. 3, n. 1, pp. 11–51.

[9] Hindley, J. R. & Seldin, J. P., 1986. *Introduction to Combinators and (lambda) Calculus*, volume 1. CUP Archive.

[10] Edwards, H. C., Trott, C. R., & Sunderland, D., 2014. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, vol. 74, n. 12, pp. 3202 – 3216. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.