Estimating the Execution Time of the Coupled Stage in Multiscale Numerical Simulations

Juan H. L. Fabian[1], Antônio T. A. Gomes[1(✉)], and Eduardo Ogasawara[2]

[1] Laboratório Nacional de Computação Científica (LNCC), Petrópolis, RJ, Brazil
{juanhlf,atagomes}@lncc.br
[2] Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ), Rio de Janeiro, RJ, Brazil
eogasawara@ieee.org

Abstract. Estimating the execution time of high-performance computing (HPC) applications is an issue that affects both shared computing infrastructures and their users. The goal of the present work is to estimate the execution time of simulation applications driven by multiscale numerical methods. In computational terms, these methods induce a two-stage simulation process. Fundamentally, the number of possibilities for configuring this two-stage process tends to be much larger than that of classical, one-stage numerical methods. This scenario makes it harder to provide accurate estimates of the execution time of multiscale simulations by using classical regression techniques. We propose a methodology that explores the idiosyncrasies of multiscale simulators to reduce the uncertainty of predictions. We applied it in this paper to the specific challenge of estimating the execution time of these simulators based on knowledge about the influence of each parameter of the numerical method they employ. We consider the multiscale hybrid-mixed (MHM) finite element method as a specific multiscale method to validate our methodology. We compared our proposed technique with 3 well-known regression approaches: a model-based tree (M5P), a bayesian nonparametric method (GPR), and a state-of-the-art ensemble method (Random Forest). We found that the root-mean-square error (RMSE) of the test dataset for our technique was considerably less than that obtained by these 3 approaches. We conclude that an educated consideration of the numerical parameters of the MHM method to estimate the execution time of the simulations helps to obtain more accurate models. We believe such conclusion can be easily generalized to other multiscale numerical methods.

Keywords: Multiscale simulations · Performance prediction · Machine learning

The authors thank CAPES (finance code 001), FAPERJ, and CNPq for partially funding this research.

S. Nesmachnow et al. (Eds.): CARLA 2020, CCIS 1327, pp. 86–100, 2021.
https://doi.org/10.1007/978-3-030-68035-0_7

1 Introduction

Simulators are computational tools used to assist in the understanding of complex natural, artificial, and social-cultural phenomena. Phenomena with multiscale characteristics require the use of sophisticated numerical methods to deal with these characteristics in terms of not only the quality of approximation but also computational performance. The so-called *multiscale numerical methods* tackle both issues. These methods achieve low approximation error rates and incorporate the granularity of the new generations of massively parallel architectures. For this paper, we consider multiscale numerical methods for finite element analysis. From a mathematical viewpoint, these methods are composed of: (i) a global formulation defined in the skeleton of a mesh of elements; and (ii) a collection of local problems, element by element, guided by the problem data—which is inherently multiscale [5]. In computational terms, this formulation induces a two-stage process:

Asynchronous stage. It solves the local problems independently, without communication between the involved processors;

Coupled stage. It collects the solutions of the local problems to build a single, coupled problem that uses all available processors synchronously.

It is important to note that the computational effort to solve the problems in the asynchronous stage can be performed offline. Besides, the problem solved at the coupled stage—although it is usually carried out online—is typically smaller than that found in a classical numerical method and, therefore, computationally advantageous. The drawback of this two-stage process is that it increases the number of configuration possibilities, as there is not only an additional stage to be configured, but also the interface between the stages.

Consider the context of shared computing infrastructures, such as clusters in supercomputing centers. The configuration problem mentioned above becomes particularly important. In these clusters, workload management systems are responsible for regulating users' access to computing nodes. These systems implement scheduling strategies that arbitrate resource contention, managing queues of jobs sent by users. Typically, users and the supercomputing center benefit from job specifications that provide accurate estimates of total execution time. It enables shorter queue times and better backfill scheduling performance. Nonetheless, it is difficult to provide accurate estimates for simulations based on multiscale numerical methods. Each configuration possibility impacts the quality of approximation and computational performance achieved.

So far, research on predicting the execution time of high-performance computing (HPC) applications has sought generality, targeting general-purpose code kernels and parallel execution patterns. We believe that exploring the idiosyncrasies of specific application families—such as that of simulators based on multiscale numerical methods—helps to reduce the uncertainty of predictions.

We propose a methodology that employs machine learning to explore the aforementioned idiosyncrasies. In this paper, we applied this methodology to build models for the prediction of the execution time of multiscale simulators

based on knowledge about the influence of each parameter of the numerical method they employ. We use the MHM method proposed by Araya et al. [2] as a frame of reference for training and testing the prediction models. Nevertheless, it is crucial to bear in mind that this study is also applicable to simulators based on other multiscale numerical methods; notably, the ones with the same parallel execution pattern [3,7].

As local problems can be computed offline, we disregard their cost for predicting the execution time of MHM simulations in this work. This simplification does not make the prediction task less difficult, though. The parameters that affect the quality of approximation of the asynchronous phase also affect some characteristics related to the computational performance of the global problem, such as the matrix conditioning and the sparsity pattern of the underlying system of linear equations.

We compared the models we built in this paper for the prediction of the execution time of MHM simulations with 3 well-known regression approaches: (i) a model-based tree (M5P), (ii) a bayesian nonparametric method (GPR), and (iii) a state-of-the-art ensemble method (Random Forest). We found that our models achieved the lowest errors among them.

We organized the remainder of this paper in the following way. In Sect. 2, we analyze some related work. The MHM method, on which the proposed methodology is based, is described in Sect. 3. It also presents the problem statement for this work. In Sect. 4, we present the proposed methodology. Some experiments are analyzed in Sect. 5. Finally, in Sect. 6, we present some concluding remarks and perspectives for future work.

2 Related Work

In the last few years, there have been many initiatives that applied machine/statistical learning for predicting the execution time of HPC applications. We describe below the most representative ones according to the data collected, techniques, and results achieved.

Matsunaga and Fortes [15] applied machine learning to predict the time and resources consumed by applications. These applications may be used in different computing infrastructures. To estimate the optimal resource usage for them is a complex task. Thus, a tree algorithm called Predicting Query Runtime (PQR) was applied to predict execution time and memory required. This algorithm enables defining different machine learning models on the leaves. Models in the leaves can be defined as linear regression or SVM. They are built from information about the application and the computing infrastructure. The approach was evaluated using two bioinformatics applications, BLAST, and RAxML, and showed good accuracy for each prediction model.

Huang et al. [11] also studied the prediction of execution time in HPC applications. They built the prediction models by using a statistical technique called sparse polynomial regression (*SPORE*). The use of this technique is justified by many predictors (*features*) considered for each application. The paper also

investigated the relationship between the predictors and the target variable and which predictors were the most relevant to predict the elapsed time. Three applications (Lucene search engine and two image processing algorithms) were used to validate the method and compare it with other statistical techniques.

Tiwari et al. [19] used machine learning to model the performance of HPC *kernels*. The data was collected using a tool called PowerMon. The assessed kernels were matrix multiplication, stencil computation, and LU factorization. It used a multilayer perceptron as the machine learning technique. Models were built regarding energy usage and execution time for each *kernel*, and the authors analyzed the influence of the training dataset size on the model accuracy.

Hieu et al. [10] studied the predictions for the execution time of applications in computational fluid dynamics (CFD). Those CFD applications were executed in a cloud environment. The prediction of the execution time was executed in two steps. Firstly, a decision tree (C4.5) was built to classify the final status of the execution (executed or not). Secondly, a multilayer perceptron was built to predict the execution time. The authors assessed the models by using the accuracy measure for the classifier, and the coefficient of determination (R) and mean absolute relative error (MARE) for the regression.

Martínez et al. [13] described a process to improve the performance of stencil kernels on multicore architectures. The process used machine learning to predict the GFLOPS and execution time of this kind of kernel. It used three different data sources: configuration parameters in the stencil implementation, hardware counters, and performance metrics. The final models were built in two steps. Both were based on SVM. In the first, intermediate models were built relating configuration parameters and hardware counters. Then, final models were built using hardware counters and performance metrics. The authors considered two kernels—7-point Jacobi and seismic wave modeling—for experimentation and reported high accuracy in the performance prediction.

Tanash et al. [18] considered a supervised machine learning technique to predict needed resources in HPC systems. The authors were interested in predicting the required memory and time for a job and in improving the Slurm resource manager used in the HPC systems. HPC log files were used as input for the model. By using a Slurm simulator, the authors observed that the model could help the resource manager to use the HPC resources in a better way.

Kim et al. [12] proposed a scheme to estimate execution time in computational science and engineering simulations. The scheme, called EXTES, is based on machine learning, and it is applied to obtain efficient simulations. The authors demonstrated the use of EXTES in a web-based platform named EDISON. They considered 16 simulation programs and observed better accuracy in the models for each simulation program.

These pieces of work have in common the use of machine learning as a tool to predict the performance of diverse kinds of applications or kernels. None of them, however, considered as predictors domain-specific information about the applications or kernels. We believe the lack of such type of information in pre-

diction models potentially reduces their accuracy. In this paper, we consider this type of information, in the specific context of multiscale numerical simulations.

3 MHM: A Multiscale Numerical Method

In this section, we briefly describe the Multiscale Hybrid-Mixed method (MHM), a type of finite element method that aims to solve large problems with multiple scales. The application of this method departs from a partial differential equation (PDE) that represents the physical problem to be simulated. A hybrid finite element formulation is proposed for this PDE that considers the continuity of its solution space using Lagrange multipliers. The hybrid formulation is then rewritten to obtain the MHM method. This rewriting leads to two types of problems: global and local. They are then discretized to obtain proper numerical approximations to the solution of the original PDE. The global problem is solved on the skeleton of a fixed finite element mesh that discretizes the domain of the PDE. The local problems are independent of each other and are solved in parallel for each element of the mesh. Each local problem considers its corresponding element of the mesh, a domain of its own. Therefore, these elements may also be discretized by a "sub-mesh". Since the local problems may be computed offline, we do not detail them in the remainder of this section.

Different physical problems can be modeled and simulated with the MHM method [2,9]. For this paper, we consider in the following the Darcy equation in a two-dimensional domain[1] defined as a boundary value problem for a diffusive process.

Diffusion Problem: Find the pressure $u : \Omega \to \mathbb{R}$ in the domain Ω s.t.:

$$\begin{cases} -\mathcal{K}\Delta u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega. \end{cases}$$

For the hybridization procedure, the MHM method first considers the decomposition of Ω into subdomains. It then defines the following function spaces:

- $\mathbf{V}$: the space of u living over Ω; and
- $\boldsymbol{\Lambda}$: the space of Lagrange multipliers living over the skeleton formed by the decomposition of Ω. This space is associated with the normal fluxes over the subdomains' boundaries.

The solution u can then be characterized as:

$$u = u_0 + \tilde{u} + u^\lambda, \text{ with } u_0 \in V_0, \tilde{u} \in \tilde{V}, u^\lambda \in \boldsymbol{\Lambda}, \text{ and } \mathbf{V} = V_0 \bigoplus \tilde{V},$$

where V_0 is the space in which the kernel of the Laplacian operator (Δ) lives.

[1] Much of the description in this section also applies to a three-dimensional domain setting, if one considers faces instead of edges as composing the skeleton of the mesh that discretizes the domain.

For the discretization procedure, the MHM method first considers a regular mesh $\mathcal{T}_H$ of elements K that discretizes the domain Ω. $H > 0$ is the characteristic measure (*i.e.*, the level of refinement) of $\mathcal{T}_H$. For simplicity, let us map each K to a unique subdomain of Ω. Each element K has its boundary ∂K, and $\mathcal{E}_H = \{\partial K\}_{K \in \mathcal{T}_H}$ defines the skeleton (*i.e.*, the set of edges) of $\mathcal{T}_H$. K can be further discretized as a local sub-mesh; $h > 0$ is the characteristic measure of this sub-mesh. The approximate function spaces are then:

$$\Lambda_H = \Lambda_l^m \subset \Lambda \text{ and } \tilde{V}_h = \bigoplus_{K \in \mathcal{T}_H} \tilde{V_K} \subset \tilde{V}.$$

The parameter m in the space of Lagrange multipliers defines the number of partitions of each edge of ∂K, and the parameter l defines the degree of Lagrange polynomials in each such partition. At the local level, each K has its space $\tilde{V_K}$ formed by Lagrange polynomials of degree k. Further details about the MHM method applied to diffusion problems are in [2,8].

It is worth remarking that, on average, approximately 94% of the time spent on the global problem is due to the solution of its underlying system of linear equations. Because of MHM's hybridization procedure, this system is of the form:

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ u_0 \end{pmatrix} = \begin{pmatrix} g_f \\ g_0 \end{pmatrix},$$

in which the dimension of A is determined by l, m, and $\#\mathcal{E}_H$, and the dimensions of B and B^T are proportional to $\#\mathcal{T}_H$.

The linear system above is a *saddle-point system*, thus presenting important challenges to linear solvers [4]. The larger the parameters l and m, and the level of refinement of the mesh, the more challenging the linear system for the solvers. Moreover, these parameters affect the linear system differently; refining the mesh—*i.e.*, increasing $\#\mathcal{E}_H$ and $\#\mathcal{T}_H$ only—increases the dimensions of the matrix, while increasing l or m makes the matrix not only bigger but also denser. The consideration of these aspects has an important impact on the quality of the predictions of the time to run simulations based on the MHM method.

4 Methodology

This paper describes part of a methodology under development, called NAZCA,[2] to assist users of multiscale simulations in the configuration of the simulations themselves and the computing resources used for these simulations. Figure 1 depicts the workflow for prediction models proposed in the NAZCA methodology.

The NAZCA methodology has two steps: learning and production. The *learning* step defines the process of building predictive models. The process departs from a set of three *parameter spaces*: (1) the characterization of the numerical method, (2) the computational architecture, and (3) the performance metrics.

[2] The name NAZCA was inspired by the Nazca Lines in Peru, which are sometimes related with ceremonial activities involving prediction [17].

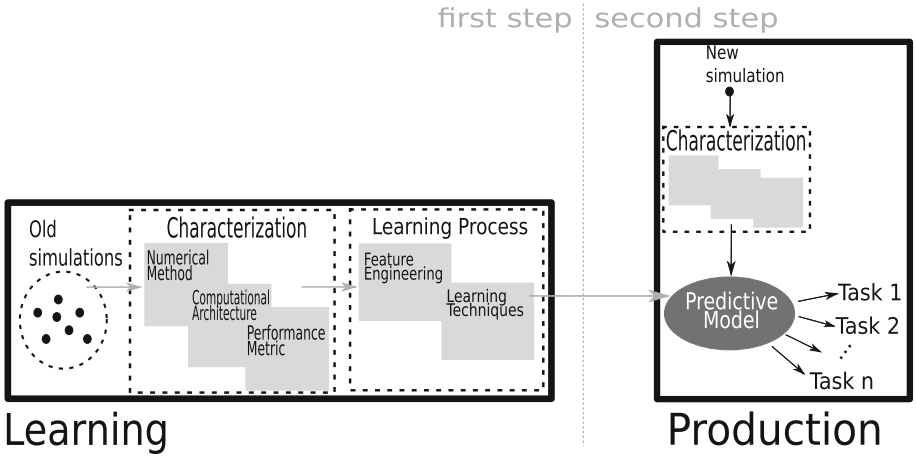


Fig. 1. NAZCA: The workflow for learning and operating prediction models.

For each intended predictive model, we need to do feature engineering in the (raw) data collected from a subset of the parameter spaces and explore diverse kinds of machine/statistical learning techniques over the data. The collected data is then used to train a model. The model typically outputs a response living in one of the parameter spaces. It is important to highlight that different combination of these parameter spaces as predictors can be used to produce different models that output different responses. In this paper, we aim to use the parameter space that characterizes the numerical method as a way to predict the execution time of a simulation. In the *production* step, predictive models are put into operation for new simulations. The feature engineering accomplished in the learning process is considered for these models as well.

A dataset in the NAZCA methodology is organized as a table with attributes as columns and samples as lines. The attributes are grouped in the three parameter spaces described above. To characterize a numerical method, we may define attributes related to the physical phenomenon, the mesh of the domain, and the numerical parameters. For the computational architecture, we may define attributes related to the number of computational nodes, the number of cores per node, and the RAM size in each node that is used in the simulation. Finally, for the performance metric, we may define attributes to analyze the performance of the numerical method (like errors in L2- and H1-norms) and of the simulation as a whole (such as success or failure, and execution time). Some of these attributes may be interrelated: for example, only when the simulation ends successfully, is it possible to obtain information on RAM usage and execution time.

Table 1 presents an example of attributes for MHM simulations.[3] These attributes were used for the proof of concept in the experiments described

[3] We differentiate **h** and **submesh** because the characteristic measure has an absolute value, whereas the level of refinement for the sub-mesh has local meaning.

in Sect. 5. It is also described the type of each attribute. We do not apply any attribute transformations. The users inform the values associated with the attributes in the numerical method and computational architecture parameter spaces. The attributes associated with the performance metric parameter space are collected while the simulations run.

Table 1. Attributes from different parameter spaces: Numerical Method, Computational Architecture and Performance Metric.

Parameter space	Attribute	Nomenclature	Type
Numerical method	Dimension of the domain (2D, 3D)	**Dim**	Nominal
	Physical phenomenon (diffusion, elasticity, etc.)	**Phys**	Nominal
	Characteristic measure of the mesh	**H**	Continuous
	Level of refinement for the sub-mesh	**submesh**	Discrete
	Characteristic measure of the sub-mesh	**h**	Continuous
	Degree of polynomial in the element - local problems	**k**	Discrete
	Degree of polynomial on the edge/face (2D/3D) - global problem	**l**	Discrete
	Number of divisions on the edge/face (2D/3D) - global problem	**m**	Discrete
Computational architecture	Number of computational nodes	**Nodes**	Discrete
	Number of cores per node	**Cores**	Discrete
	Total RAM in the computational nodes	**RAM**	Discrete
Performance metric	Success of the simulation	**S**	Binary
	Numerical error in the L2-norm	**L2**	Continuous
	Numerical error in the H1-norm	**H1**	Continuous
	Total execution time	**TE**	Continuous
	Partial time of the global problem	**TPG**	Continuous
	Partial time of the local problems	**TPL**	Continuous
	RAM usage in the local problems	**RAM-PL**	Discrete
	RAM usage in the global problem	**RAM-PG**	Discrete

In the learning step, the data is randomly divided into training and test datasets using the 80-20 strategy. The model is trained only using the training dataset, and it is assessed in the test dataset. We do not optimize the hyperparameters of the models, therefore, a validation set is not defined.

Estimating the Execution Time from Numerical Method Attributes. We explained in Sect. 3 that the execution time of the global problem in MHM simulations is influenced by the parameters l, m, $\#\mathcal{E}_H$, and $\#\mathcal{T}_H$. Besides, l and m are determinants for the sparse pattern of the system of linear equations associated with the global problem, affecting its computational complexity (as verified in Subsect. 5.2). We, therefore, devised a tree-based architecture that handles each possible combination of l and m. Moreover, we employed a feature engineering procedure to derive from l, m, $\#\mathcal{E}_H$, and $\#\mathcal{T}_H$ an additional attribute (GLG) that represents the total number of degrees of freedom in the linear system solved by the global problem. This new attribute is employed as the

predictor of several univariate regression models, each one of them living on a different leaf of the tree. Figure 2 depicts the tree architecture.

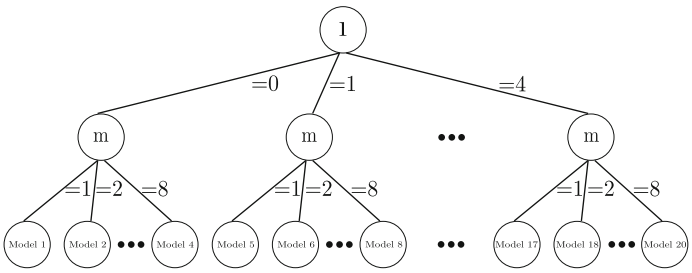


Fig. 2. Tree-based architecture for handling prediction models.

On each leaf of the model tree, we use empirical analysis to select the best univariate regression model. The empirical analysis consists of repeating the training and testing of a given model,[4] with a random division of training and test data for each repetition. We then collect for each such repetition the fitted model and its associated prediction band, and analyze two hypotheses over them:

- H_0^V: The model suffers little effect from changes in training and test data. We verify this hypothesis by ascertaining that the fitted models are confined within the area bounded by the prediction bands;
- H_0^R: The model is reliable—we verify this hypothesis by ascertaining that the data samples are all contained within some prediction band.

5 Experimental Evaluation

In this section, we describe the proof of concept of the methodology explaining the experimental part of the research. We start by defining the data used for building the prediction models, in which the attributes in the data were defined by experts. Next, we look for any possible patterns in the data. Finally, we apply our methodology for building predictive models of the execution time.

5.1 Dataset

Using Table 1 as a reference, we fixed **Dim** = '2D' and **Phys** = 'Diffusion' to match the diffusion problem described in Sect. 4. For the other parameters, we considered the combination of the values listed in Table 2. For a single combination, two different simulations were performed to enrich the dataset—each one based on a different refinement pattern for H (criss-cross and irregular). For the computational architecture, we fixed a single configuration, consisting of a workstation with two 12-core sockets and 320 GB of RAM. All the simulations that were run to collect performance metric data used 2 MPI processes. This setup amounts to a total of $1,800$ simulations in our experimental dataset.

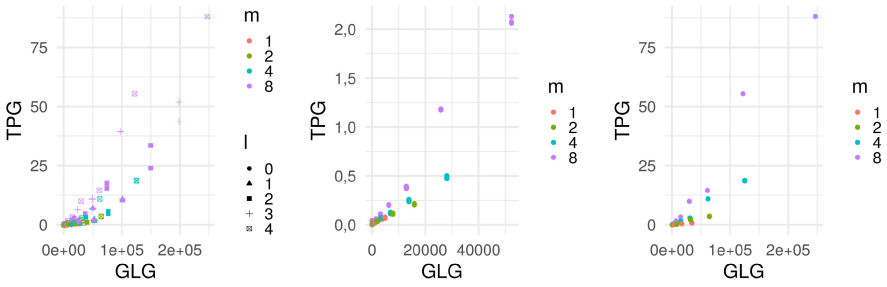
[4] We used $1,000$ repetitions as in the traditional bootstrap setup [6].

5.2 Exploratory Data Analysis

In Fig. 3, we analyze the relation between TPG (the target variable) and GLG. We can confirm in Fig. 3(a) our assertions in Sects. 3 and 4 that there are different patterns when we combine the values of parameters l and m. Besides, we can see that for a fixed value of the parameter l, there are patterns influenced by the values in the parameter m, as we can see in Figs. 3(b) and 3(c).

Table 2. Parameters used in the experimental evaluation

Parameters	Values
Submesh	1, 2, 4, 8
m	1, 2, 4, 8
k	2, 3, 4, 5, 6
l	0, 1, 2, 3, 4



(a) Combining l and m (b) $l = 0$ (c) $l = 4$

Fig. 3. TPG vs GLG.

We show in Fig. 4 the distribution of training and test data on each of these leaves. Each time we increase the value of l and m, the amount of data available for training and test decreases. This skewed distribution is a consequence of the specific constraints of the well-posedness of a formulation in MHM,[5] and it may affect the performance of the model, as we show in the following section.

5.3 Model Building and Assessment

We consider different kinds of models for our analysis: $y = a_0 + a_1 x$ (model 1); $y = a_0 + a_1 x^{3/2}$ (model 2); $y = a_0 + a_1 x + a_2 x^{3/2}$ (model 3); $y = a_0 + a_1 x^2$ (model 4) and $y = a_0 + a_1 x + a_2 x^2$ (model 5). Models 2 and 3 were considered

[5] Briefly speaking, increasing l without increasing m also increases the minimal accepted value for k, thus reducing the amount of possible combinations in Table 2.

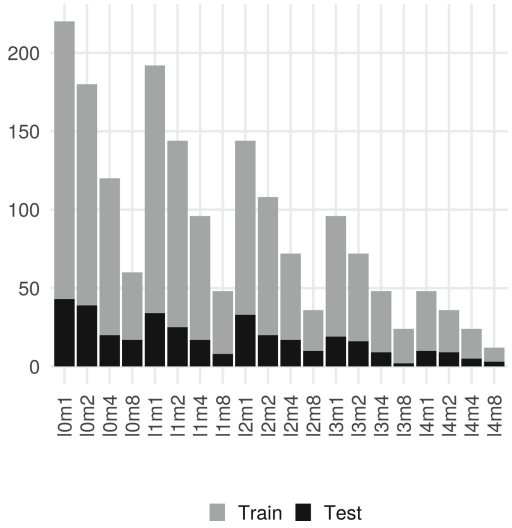


Fig. 4. Distribution of training and test data according to each case.

because we use the MUMPS parallel linear solver [1] for the global problem, and Mary [14] shows that this solver has an asymptotic time complexity of $\mathcal{O}(n^{3/2})$.

In the following, we analyze cases $l = 0, m = 2$, and $l = 4, m = 8$. In the first case, we have a filtered dataset with 200 simulations and a clear behavior for the empirical analysis (c.f. Fig. 5). In the second case, we have a filtered dataset with only ten simulations and a fuzzier behavior (c.f. Fig. 6).

As for the selection of model 2 in the example of Fig. 5, we observe that in models 1, 3, and 5, there are data points that fall out of the prediction band. For this reason, we refute the hypothesis H_0^R. Concerning the hypothesis H_0^V, we refute it in models 3, 4, and 5. We then selected model 2 because it was the only one in which we could not refute both hypotheses. As for the case shown in Fig. 6, at least one of the hypotheses was refuted by each model. In this case, our technique cannot select a proper model using empirical analysis. Thus, the strategy adopted based on Occam's razor was to select the most straightforward model (model 1). In Table 3, we summarize the models selected by our technique for each leaf of our model tree.

Table 3. Models for different values of the parameters l and m.

l	m	Model	l	m	Model	l	m	Model	l	m	Model	l	m	Model
0	1	$a_0 + a_1 x$	1	1	$a_0 + a_1 x^{3/2}$	2	1	$a_0 + a_1 x^{3/2}$	3	1	$a_0 + a_1 x^{3/2}$	4	1	$a_0 + a_1 x^{3/2}$
	2	$a_0 + a_1 x^{3/2}$		2	$a_0 + a_1 x^{3/2}$		2	$a_0 + a_1 x^{3/2}$		2	$a_0 + a_1 x^{3/2}$		2	$a_0 + a_1 x$
	4	$a_0 + a_1 x^{3/2}$		4	$a_0 + a_1 x^{3/2}$		4	$a_0 + a_1 x^{3/2}$		4	$a_0 + a_1 x$		4	$a_0 + a_1 x$
	8	$a_0 + a_1 x^{3/2}$		8	$a_0 + a_1 x^{3/2}$		8	$a_0 + a_1 x$		8	$a_0 + a_1 x$		8	$a_0 + a_1 x$

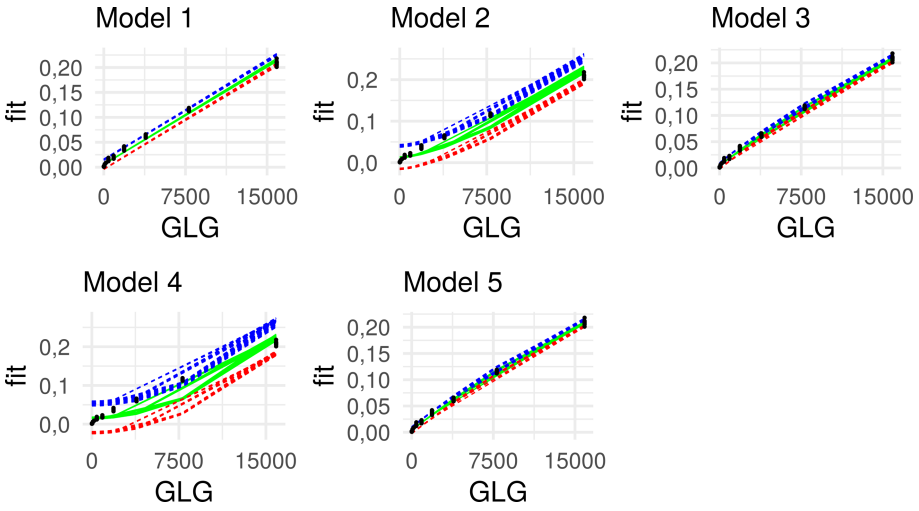


Fig. 5. An empirical analysis for $l = 0, m = 2$. We plot the data samples (dots) and the fitted models (green) with their prediction bands, upper (blue) and lower (red) bound. (Color figure online)

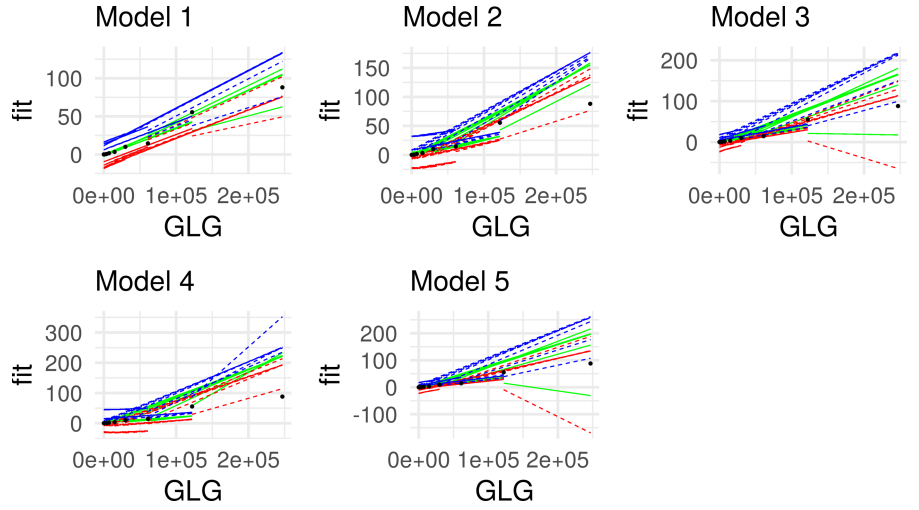


Fig. 6. An empirical analysis for $l = 4, m = 8$. We plot the data samples (dots) and the fitted models (green) with their prediction bands, upper (blue) and lower (red) bound. (Color figure online)

We calculated the error obtained in the test dataset for each leaf of our model tree. After that, we computed the error for the complete test dataset and arrived at an RMSE of 0.272. In Fig. 7, we can see the squared error for each leaf of our model tree. We can identify which of them have the higher errors. A high error

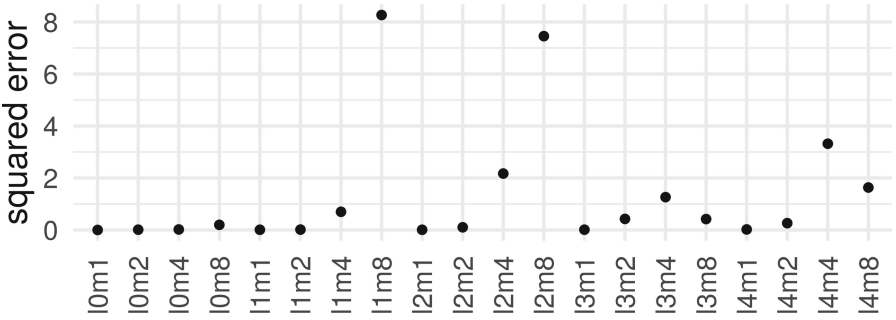


Fig. 7. Squared errors for each case during the tests.

could be caused for diverse reasons, such as a small dataset or a poor model. Reducing the errors for these cases is the subject of future work.

We compared the performance of our technique in the test dataset with 3 well-known regression approaches implemented in the WEKA workbench [21]: (i) M5P ([16,20]), (ii) Gaussian Process Regression (GPR), and (iii) Random Forest (RF). Table 4 summarizes the results obtained for each technique.

Table 4. Comparison of regression approaches.

Technique	RMSE
NAZCA	0.272
M5P	2.111
RF	2.446
GPR	3.477

We can conclude that the considerations related to the numerical parameters proposed in our technique allowed a better generalization of the model.

6 Conclusion

Predicting the execution time of simulations based on multiscale numerical methods is complex due to their two-stage process. We presented NAZCA, a methodology capable of dealing with this situation. NAZCA aims to build prediction models for these simulations based on machine learning, benefiting both computing infrastructure providers and their users.

We applied the MHM method in a diffusion equation, and we conducted some experiments to obtain a dataset used for training and test. Different machine learning techniques are explored to build prediction models. We proposed a technique inspired in a tree, which for its building, considers some parameters

of the numerical method. On each leaf of the tree, we carried out an empirical analysis for model selection. To validate our approach, we compared it with three well-known regression approaches. We concluded that our proposed technique achieves a small error of generalization compared to them.

Some future work could be considered for this research. In our proposed technique, we established some assumptions and limitations that could be tackled. Values for the numerical parameters are not limited to the experiments considered here, so another type of learning technique may be needed for this situation. In our technique, we also observed that the small amount of data influenced the achievement of accurate models, and a better analysis of how to get around this difficulty could be studied. Finally, we consider the application of the methodology described herein to other multiscale numerical methods in which the two-stage process is observed.

References

1. Amestoy, P., Duff, I.S., Koster, J., L'Excellent, J.Y.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM J. Matrix Anal. Appl. **23**(1), 15–41 (2001)
2. Araya, R., Harder, C., Paredes, D., Valentin, F.: Multiscale hybrid-mixed method. SIAM J. Numer. Anal. **51**(6), 3505–3531 (2013)
3. Arbogast, T., Pencheva, G., Wheeler, M.F., Yotov, I.: A multiscale mortar mixed finite element method. Multiscale Model. Simul. **6**(1), 319–346 (2007)
4. Benzi, M., Golub, G.H., Liesen, J.: Numerical solution of saddle point problems. Acta Numerica **14**, 1–137 (2005)
5. Efendiev, Y., Hou, T.Y.: Multiscale Finite Element Methods. Springer, New York (2009). https://doi.org/10.1007/978-0-387-09496-0
6. Efron, B., Tibshirani, R.J.: An Introduction to the Bootstrap. No. 57 in Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, Boca Raton (1993)
7. Guiraldello, R.T., Ausas, R.F., Sousa, F.S., Pereira, F., Buscaglia, G.C.: The multiscale robin coupled method for flows in porous media. J. Comput. Phys. **355**, 1–21 (2018)
8. Harder, C., Paredes, D., Valentin, F.: A family of multiscale hybrid-mixed finite element methods for the Darcy equation with rough coefficients. J. Comput. Phys. **245**, 107–130 (2013)
9. Harder, C., Paredes, D., Valentin, F.: On a multiscale hybrid-mixed method for advective-reactive dominated problems with heterogeneous coefficients. Multiscale Model. Simul. **13**(2), 491–518 (2015)
10. Hieu, D.N., Tieu Minh, T., Van Quang, T., Giang, B.X., Van Hoai, T.: A machine learning-based approach for predicting the execution time of CFD applications on cloud computing environment. In: Dang, T.K., Wagner, R., Küng, J., Thoai, N., Takizawa, M., Neuhold, E. (eds.) FDSE 2016. LNCS, vol. 10018, pp. 40–52. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48057-2_3
11. Huang, L., Jia, J., Yu, B., Chun, B.G., Maniatis, P., Naik, M.: Predicting execution time of computer programs using sparse polynomial regression. In: Advances in Neural Information Processing Systems 23, pp. 883–891. Curran Associates, Inc. (2010)

12. Kim, S., Suh, Y., Kim, J.: EXTES: an execution-time estimation scheme for efficient computational science and engineering simulation via machine learning. IEEE Access **7**, 98993–99002 (2019)
13. Martínez, V., Dupros, F., Castro, M., Navaux, P.: Performance improvement of stencil computations for multi-core architectures based on machine learning. Proc. Comput. Sci. **108**, 305–314 (2017)
14. Mary, T.: Block Low-Rank multifrontal solvers: complexity, performance, and scalability. Ph.D. thesis, Université Paul Sabatier - Toulouse III (2017)
15. Matsunaga, A., Fortes, J.A.B.: On the use of machine learning to predict the time and resources consumed by applications. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 495–504 (2010)
16. Quinlan, R.J.: Learning with continuous classes. In: 5th Australian Joint Conference on Artificial Intelligence, pp. 343–348. World Scientific, Singapore (1992)
17. Silverman, H.: Cahuachi in the Ancient Nasca World. University of Iowa Press, Iowa City (1993)
18. Tanash, M., Dunn, B., Andresen, D., Hsu, W., Yang, H., Okanlawon, A.: Improving HPC system performance by predicting job resources via supervised machine learning. In: Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning), pp. 1–8 (2019)
19. Tiwari, A., Laurenzano, M.A., Carrington, L., Snavely, A.: Modeling power and energy usage of HPC kernels. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum, pp. 990–998 (2012)
20. Wang, Y., Witten, I.H.: Induction of model trees for predicting continuous classes. In: Poster Papers of the 9th European Conference on Machine Learning. Springer (1997)
21. Witten, I.H., Frank, E., Hall, M.A.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Series in Data Management Systems, 3rd edn. Morgan Kaufmann, Amsterdam (2011)