Estimating the execution time of fully-online multiscale numerical simulations

Juan H. L. Fabian[1], **Antônio T. A. Gomes**[1], **Eduardo Ogasawara**[2]

[1]Laboratório Nacional de Computação Científica (LNCC)
Petrópolis – RJ – Brazil

[2]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)
Rio de Janeiro – RJ – Brazil

{juanhlf,atagomes}@lncc.br, eogasawara@ieee.org

Abstract. *In this paper, we propose a methodology for estimating the execution time of simulations driven by multiscale numerical methods. The methodology explores the idiosyncrasies of multiscale simulators to reduce the uncertainty of predictions. We use the multiscale hybrid-mixed (MHM) finite element method to validate our methodology. We compare our proposed technique with prediction models automatically selected and calibrated by Auto-WEKA. We show that the models obtained with our technique are competitive when compared with the models coming from Auto-WEKA, being interpretable and with much less computational effort during the learning process.*

1. Introduction

Properly configuring numerical simulations with respect to the intended approximation error rates and the number of computing resources involved is particularly important in the context of shared computing infrastructures. In these infrastructures, workload management systems regulate users' access to computing nodes. These systems arbitrate resource contention, managing queues of users' jobs. Crucially, users and infrastructure providers benefit from job specifications that provide accurate estimates of total execution time, because they enable shorter job queue times and higher resource utilization.

Physicists, chemists, and engineers run numerical simulators in shared computing infrastructures to understand complex phenomena. For this paper, we consider the specific family of simulators based on *multiscale finite element methods*. These methods decompose a physical problem into a global formulation defined over the problem domain and a collection of local formulations defined over disjoint partitions of this domain [Efendiev and Hou, 2009; Weinan, 2011]. As a result, simulators based on these methods allow for lower approximation errors whilst achieving higher levels of intrinsic parallelism in comparison with classical finite element methods. Nonetheless, the decomposition approach incurs a two-stage process and, consequently, a large parameterization space that makes it difficult to provide accurate estimates of the number of computational resources needed for the simulations and the total time of execution of these simulations, given the sought accuracy of the approximation.

To tackle the aforementioned difficulty, we are currently developing a methodology—called NAZCA[1]—that employs machine learning to explore the idiosyn-

[1]The name NAZCA was inspired by the Nazca Lines in Peru, which are sometimes related with ceremonial activities involving prediction [Silverman, 1993].

crasies of simulators based on multiscale finite element methods for making predictions. In this paper, we are interested in the specific problem of estimating the execution time of these simulators; our hypothesis is that we can reduce the uncertainty of predictions if we account for these idiosyncrasies. As far as we know, research on the prediction of the execution time of applications in shared computing infrastructures has only targeted general-purpose code kernels and parallel execution patterns; no pieces of work have been found in the literature that aims at simulators based on (either classical or multiscale) finite element methods.

In Fabian et al. [2020], we presented our first step toward tacking the problem stated above. In that paper, we take a simplified version of the problem; we consider that the computation of the local formulations may be computed *offline*. Therefore, only the computation of the global formulation must be actually allocated in shared computing infrastructure. This assumption suits well stationary and transient linear problems (e.g., diffusion processes, elastostatic, and elastodynamic models). However, for nonlinear problems (e.g., phenomena governed by the Navier-Stokes equations), the local formulations must be computed *online*, i.e., within the same job instantiation as the computation of their corresponding global formulation.

The present paper extends our work in Fabian et al. [2020] by aiming at the prediction of the total time to jointly compute local and global formulations. As in that work, we use the MHM method proposed by Araya et al. [2013] 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 as MHM (e.g., [Guiraldello et al., 2018; Arbogast et al., 2007]).

We compare the prediction models devised in this paper with models obtained using the automated machine learning approach offered by Auto-WEKA [Kotthoff et al., 2017]. We found that our models are quite competitive; they offer errors of the same order of magnitude as the best models selected by Auto-WEKA, but with much less computational effort during the learning process. Besides, our approach produces an interpretable model, which is not guaranteed while using Auto-WEKA.

We organized the remainder of this paper as follows. We analyze some related work in Section 2. The MHM method, on which the proposed methodology is based, is described in Section 3. In Section 4, we present the proposed methodology. Some experiments are presented in Section 5. Finally, we present some concluding remarks and perspectives for future work in Section 6.

2. Related work

In recent years, many approaches have been proposed to predict the execution time of HPC applications, as well as the main kernels present in some of these applications. In the following, we describe representative pieces of work that have in common the use of machine learning to predict the performance of diverse kinds of applications or kernels.

Predicting the execution time of applications is commonly studied [Hieu et al., 2016; Malakar et al., 2018; Kim et al., 2019]. [Hieu et al., 2016] used applications in computational fluid dynamics (CFD). Those CFD applications were executed in a cloud

environment. The authors started by classifying the final status of the execution (executed or not) using a decision tree (C4.5) and then, the execution time was predicted using a multilayer perceptron. The models were assessed by using the accuracy measure for the tree, and the coefficient of determination (R) and mean absolute relative error (MARE) for the perceptron. [Malakar et al., 2018] described a benchmark study using 11 machine learning techniques and they assess the behavior of four scientific applications on four HPC platforms. These applications covered three types of problems: molecular dynamics, adaptive mesh refinement, and unstructured implicit finite element analysis. The experiments carried out by the authors sought to assess the influence of feature engineering and the size of the training set. [Kim et al., 2019] proposed a scheme, called EXTES, for the estimation of execution time. The authors demonstrated the use of EXTES with 16 applications in diverse fields of computational science and engineering. For each of these applications, the authors reported good accuracy in the models.

Predicting the execution time in application kernels is studied in [Tiwari et al., 2012; Martínez et al., 2017]. In Tiwari et al. [2012], the kernels were matrix multiplication, stencil computation, and LU factorization. The authors used a multilayer perceptron for each kernel. The data was collected using a tool called PowerMon and the authors analyzed the influence of the training dataset size on the model accuracy. [Martínez et al., 2017] described a process to predict the execution time of two stencil kernels (7-point Jacobi and seismic wave modeling) on multicore architectures. The process used three different data sources: configuration parameters in the stencil implementation, hardware counters, and performance metrics. First, intermediate models were built relating configuration parameters and hardware counters. Then, final models were built using hardware counters and performance metrics (execution time). Intermediate and final models were based on a support vector machine (SVM). For each of them, the authors reported high accuracy.

The main drawback of these approaches is that none of them used domain-specific information about the applications as predictors. In this paper, as in Fabian et al. [2020], we use information about the multiscale numerical simulations to build prediction models. To the best of our knowledge, no other work has done this before for numerical simulations based on (classical or multiscale) finite element methods.

3. MHM: a family of multiscale numerical methods

In this section, we briefly describe the Multiscale Hybrid-Mixed methodology (MHM), which encompasses a family of finite element methods aimed at solving large problems with multiple scales. The MHM methodology departs from a partial differential equation (PDE) that represents the physical problem to be simulated. Validated problem examples of the MHM methodology in the literature include the Darcy equation with rough coefficients [Araya et al., 2013], the Stokes and Brinkman equations [Araya et al., 2017], and the Helmholtz equation [Chaumont-Frelet and Valentin, 2020]. For the sake of illustration, we consider a boundary value problem for a diffusive process in domain Ω, with a highly oscillatory $\mathcal{K}$ coefficient as its main multiscale feature:

Find the pressure $u : \Omega \to \mathbb{R}$ *in the domain* Ω *such that*

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

In the MHM methodology, a hybrid finite element formulation is applied to the PDE. The hybridization procedure decomposes Ω into subdomains, and the formulation considers the continuity of the solution space across the subdomains using Lagrange multipliers. To do so, the following (infinite dimensional) function spaces are defined:

- $\mathbf{V}$: the space of u living over Ω; and
- $\mathbf{\Lambda}$: 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:

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

where V_0 is the space in which the kernel of the differential operator lives—in our illustrative example, the Laplacian (Δ) operator. This hybrid formulation is rewritten to obtain two types of problems, which are then discretized: global and local.

The global problem is solved on the skeleton of a mesh of elements $\mathcal{T}_H = \{K\}$ that discretizes the domain Ω. This skeleton is defined by $\mathcal{E}_H = \{\partial K\}_{K \in \mathcal{T}_H}$, where ∂K is the boundary of K. It, therefore, corresponds to the set of element faces in $\mathcal{T}_H$; $H > 0$ is the characteristic measure (*i.e.*, the measure of the largest face) of $\mathcal{T}_H$ and reflects its level of refinement.

A local problem is associated with each subdomain of Ω, and it can be solved independently of the local problems associated with the other subdomains. For the sake of implementation simplicity, we take each element K of $\mathcal{T}_H$ as a subdomain, so that we have one local problem per K.

The approximate (finite dimensional) 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 face of $\mathcal{E}_H$, and the parameter l defines the degree of Lagrange polynomials in each such partition. Importantly, the finite set of basis functions that span Λ_l^m are not known *a priori*; they are computed by the local problems and "upscaled" to the global problem. This is how the MHM methods (and multiscale methods in general) are capable of capturing multiscale features. At the local level, each K has its space $\tilde{V_K}$ formed by Lagrange polynomials of degree k that live on a "sub-mesh" within K; $h > 0$ is the characteristic measure of this sub-mesh. Further details about the MHM method applied to diffusion problems are in [Harder et al., 2013; Araya et al., 2013].

In computational terms, an MHM method (and again multiscale methods in general) can be seen as a two-stage process:

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

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

Since the local problems are individually much cheaper computationally than the global problem, they may be performed offline when the upscaling procedure can be done only once for a single simulation setup. This is often the case for stationary and transient

linear problems (e.g., the diffusive process illustrated above), but not for nonlinear problems (e.g., phenomena governed by the Navier-Stokes equations); in the latter case, the multiscale basis functions that span Λ_l^m must be computed at each step of an iterative linearization process. This renders an algorithm in which the local problems and the global problem must be solved online, i.e., within a same application instantiation.

3.1. On the estimation of the execution time of MHM simulations

It is well known in the literature (see, for instance, [Farmaga et al., 2011]) that the time spent on finite element simulations is mainly due to the solution of their underlying linear system of equations. Therefore, these linear systems are the main target of our learning approach to estimating the execution time of MHM simulations. Nevertheless, the matter becomes somewhat more complex for the MHM simulations, as different linear systems appear at the global and local levels. For the sake of argument, we only consider direct approaches—i.e., matrix factorizations—to the solution of these linear systems.

The linear system associated with the global problem has the general form:

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

The dimension of A is determined by l, m, and $\#\mathcal{E}_H$. The dimensions of B and B^T are proportional to $\#\mathcal{T}_H$. In Fabian et al. [2020], we analyzed the influence of these parameters on the estimation of the execution time for solving the global problem alone, so in Section 5 we will only reproduce our main results from that work and compare them with the new results obtained with Auto-WEKA.

For each local problem, there is also a set of $N_I + 1$ linear systems of the form:

$$\begin{aligned} LC &= F, \\ LD_i &= N_i,\ \forall i \in \{0, ..., N_I - 1\}. \end{aligned}$$

The dimension of L is determined by k and h. N_I is defined by the number of faces in the element K and by l and m. We can observe that these systems share the same matrix L, which means that L can be factorized only once; hence, increasing the values of l and m and the number of faces in the element K only increases the number of matrix-vector multiplications, which are much less computationally expensive than factorizations.

Since we have as many local problems as the number of elements in $\mathcal{T}_H$, and these local problems are independent from each other, we can simply distribute the computation of their corresponding linear systems across all the cores made available by the shared computing infrastructure. Estimating the execution time for solving the local problems must therefore take into account this distribution.

4. Methodology

This paper describes part of a machine learning-based methodology under development, called NAZCA, to assist users of multiscale simulations in the configuration of the simulations themselves and the computing resources used for these simulations. The learning process in the NAZCA methodology departs from a set of three *parameter spaces*: (1) the characterization of the numerical method, (2) the computational architecture, and (3) the

performance metrics. Table 1 illustrates these spaces for MHM simulations. They are the ones used for the experiments described in Section 5.

It is important to highlight that different combinations of parameters in these spaces can be used to produce different prediction models that output different responses. The users inform the values associated with the parameters in the numerical method and computational architecture parameter spaces. The parameters associated with the performance metric parameter space are collected while the simulations run. 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. Parameters in the NAZCA methodology for MHM simulations.

Parameter space	Parameter	Symbol
Numerical method	dimension of the domain (2D, 3D)	**Dim**
	physical phenomenon (diffusion, elasticity, etc)	**Phys**
	characteristic measure of the mesh	**H**
	level of refinement for the sub-mesh	**submesh**
	characteristic measure of the sub-mesh	**h**
	degree of polynomial in the element - local problems	**k**
	degree of polynomial on the edge/face (2D/3D) - global problem	**l**
	number of divisions on the edge/face (2D/3D) - global problem	**m**
Computational architecture	number of computational nodes	**Nodes**
	number of cores per node	**Cores**
	total RAM in the computational nodes	**RAM**
Performance metric	success of the simulation	**S**
	numerical error in the L2-norm	**L2**
	numerical error in the H1-norm	**H1**
	total execution time	**TE**
	partial time of the global problem	**TPG**
	partial time of the local problems	**TPL**
	RAM usage in the local problems	**RAM-PL**
	RAM usage in the global problem	**RAM-PG**

4.1. Estimating the execution time from numerical method parameters

In this paper, we set the parameter space to a single computational architecture and we intend to use the parameter space that characterizes the numerical method as a way to predict a single performance metric: the execution time of a simulation.

As explained in Section 3, the execution time of MHM simulations encompasses the time used in the asynchronous and coupled stages. In Fabian et al. [2020], we have already considered the coupled stage, so we only describe the asynchronous stage herein. The time used in the asynchronous stage is influenced by the number of local problems and computational architecture. For a specific local problem, k and h are determinants for the sparsity pattern of its linear system of equations.

For the sake of interpretability, we look for univariate regression models. So, we employed a feature engineering procedure to derive from k and h a single attribute GLL that represents the number of degrees of freedom in a specific linear system. Then, for the asynchronous stage, we define the attribute GLLT representing the total number of degrees of freedom of all linear systems allocated to a single core. GLLT is computed using the Equation 1:

$$GLLT = \lceil \frac{\#LocalProblems}{\#CoresInArchitecture} \rceil \times GLL. \quad (1)$$

Similarly, we define the attribute TMPL as the total execution time of all linear systems allocated to the most demanding core.

Despite the feature engineering procedure described above, k and h continue to determine the computational pattern in the asynchronous stage. To isolate the effect of h (a continuous parameter) in GLLT, we devised a tree-based architecture that handles each possible value of k (which is discrete). GLLT and TMPL are then employed respectively as the predictor and the target variable of several univariate regression models, each one of them living on a different leaf of the tree. Figure 1 depicts the tree architecture.

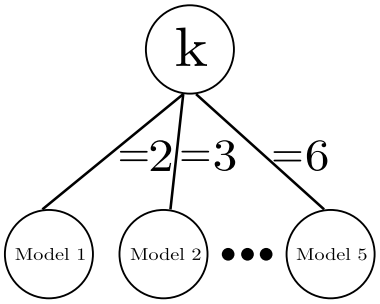


Figure 1. Tree-based architecture for handling prediction models.

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 many times,[2] 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: There is low variability in results, i.e., changes in training and test data have little effect on the model. We verify this hypothesis by ascertaining that a fitted model is within the area bounded by the prediction bands of all other models;
- H_0^R: The model is reliable. This hypothesis is verified by ascertaining that the data samples are all contained within some prediction band.

5. Experimental Evaluation

5.1. Dataset

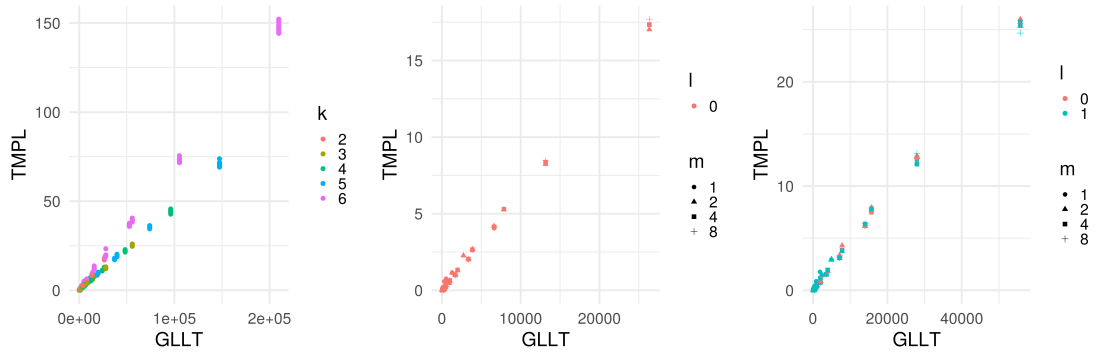
Using Table 1 as a reference, we fixed $\mathbf{Dim}$ = '2D' and $\mathbf{Phys}$ = 'Diffusion' to match the diffusive process described in Section 4. For the other parameters, we considered the combination of the values listed in Table 2. For a single combination, two different simulations were performed to enrich the dataset—each one based on a different refinement pattern for H (criss-cross and irregular).

For the computational architecture, we set 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 1800 simulations in our experimental dataset. 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.

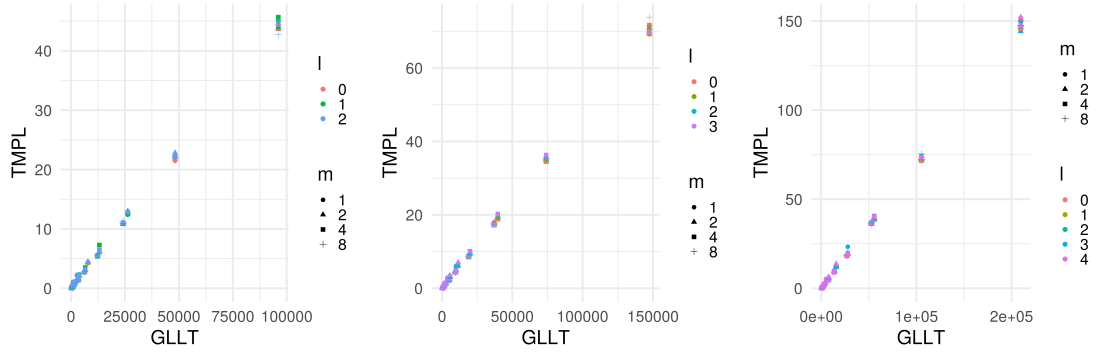
[2] We used 1000 repetitions as in the traditional bootstrap setup [Efron and Tibshirani, 1993].

5.2. Exploratory Data Analysis

In Figure 2, we analyze the relation between TMPL (the target variable) and GLLT. We can confirm in Figure 2(a) our assertions in Sections 3 and 4 that there are different computational patterns when we set the value of the parameter k. Besides, we can see that for a fixed value of the parameter k, there is no discernible pattern influenced by the combining values of the parameters l and m, as we can see in Figures 2(b)- 2(f). That is the reason for adopting the one-level tree-based architecture depicted in Figure 1.



(a) different parameter values of k (b) $k = 2$ (c) $k = 3$



(d) $k = 4$ (e) $k = 5$ (f) $k = 6$

Figure 2. TMPL vs GLLT.

5.3. Model Building and Assessment

Similarly to what we did in [Fabian et al., 2020] for the coupled stage, here we consider different kinds of models for our analysis of the asynchronous phase: $y = a_0 + a_1 x$ (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

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

4) and $y = a_0 + a_1 x + a_2 x^2$ (model 5). In choosing these models, we took account of the computational complexity of solving linear systems, which is $O(n^3)$, but bearing in mind that the actual performance may vary a lot depending on the form of the linear system.

In the following, we analyze hypotheses H_0^V and H_0^R for cases $k = 2$ and $k = 5$. Figures 3 and 4 plot, for cases $k = 2$ and $k = 5$, respectively, the data samples (dots) and the fitted models (green) with their upper (blue) and lower (red) prediction bands. We chose these two cases because they are representative of the behavior of the other three cases ($k = 2$ being similar to $k = 6$; and $k = 5$ being similar to $k = 3$ and $k = 4$). For case $k = 2$, we have a filtered dataset with 95 simulations and a bad behavior for the empirical analysis. For case $k = 5$, we have a filtered dataset with 467 simulations and a good behavior for the empirical analysis. We can see that an imbalanced distribution of data affects our empirical analysis, but this is a consequence of the specific well-posedness constraints of the mathematical formulation of MHM for diffusive processes (c.f. [Araya et al., 2013]).

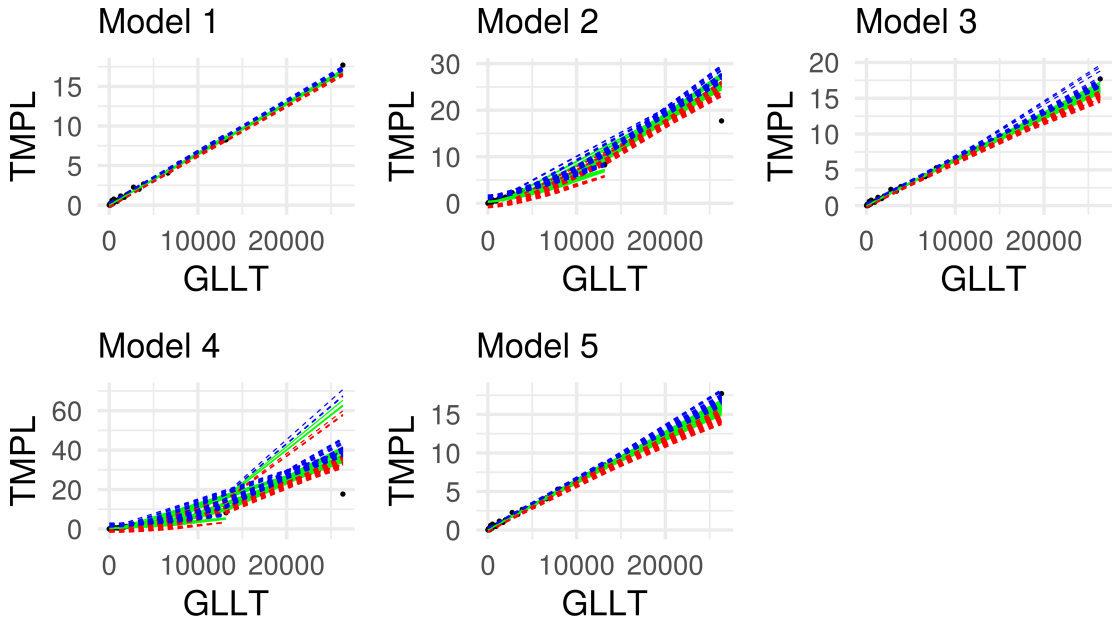


Figure 3. An empirical analysis for $k = 2$.

We can observe in Figure 3 that for each model for $k = 2$ the hypotheses H_0^V and H_0^R are both refuted. So, our technique cannot select a proper model using empirical analysis. As a consequence, the strategy adopted for $k = 2$ is to select the most straightforward model (model 1). As for Figure 4, the hypothesis H_0^R for $k = 5$ is refuted for models 1, 2 and 3, and the hypothesis H_0^V is refuted for models 2 and 4. We, therefore, select model 5 for the case when $k = 5$. In Table 3, we summarize the models selected by our technique for each leaf of our model tree. This model selection procedure by our technique took a couple of minutes in an ordinary laptop computer with a single 4-core socket and 8 GB of RAM.

5.4. Comparison with Auto-WEKA

In this section, we include the prediction models obtained in [Fabian et al., 2020] for the coupled stage in the evaluation; however, we now take them together with the pre-

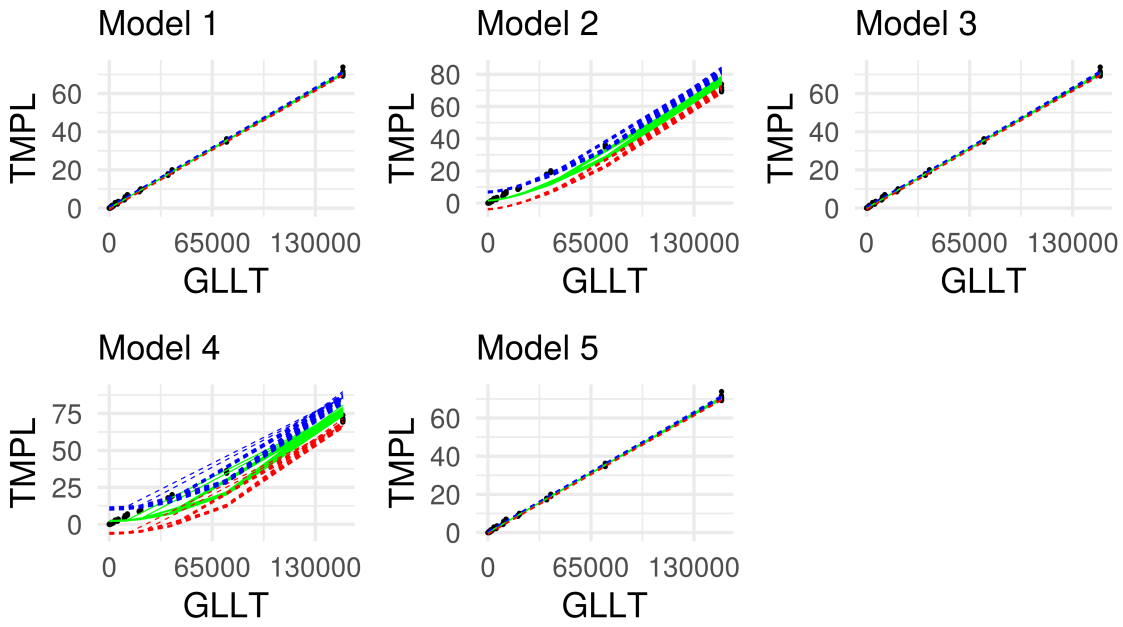


Figure 4. An empirical analysis for $k = 5$.

Table 3. Models for different values of the parameter k.

k	Model
2	$a_0 + a_1 x$
3	$a_0 + a_1 x + a_2 x^2$
4	$a_0 + a_1 x + a_2 x^2$
5	$a_0 + a_1 x + a_2 x^2$
6	$a_0 + a_1 x$

diction models obtained in this paper for the asynchronous stage, and compare both of them with Auto-WEKA [Kotthoff et al., 2017]. Auto-WEKA searches the better models between the learning algorithms and their hyper-parameters implemented in the WEKA workbench [Witten et al., 2011]. In seeking a fair comparison with our technique, we also split the learning process of Auto-WEKA to the asynchronous and coupled stages. We then compare their performance with our technique on a stage-by-stage basis, using the root-mean-square error (RMSE) as the measure of fit quality. Importantly, Auto-WEKA is a time-sensitive approach to finding the best model. For each stage, we let Auto-WEKA run for approximately two days in the same ordinary laptop computer where our model selection procedure was run.

For the asynchronous stage, Auto-WEKA selected a KStar model [Cleary and Trigg, 1995]. For the coupled stage, it selected an M5P model [Quinlan, 1992]. Table 4 shows the quality of fit obtained for each stage by our technique and by the best models selected by Auto-WEKA.

We conclude that using domain-specific information related to the numerical method generated competitive prediction models for both simulation stages. These models obtained errors of the same order of magnitude as the best models selected by Auto-WEKA, but with much less computational effort during the learning process.

Table 4. Comparison of regression approaches for each stage.

	Technique	RMSE
Asynchronous	NAZCA	0.367
	KStar	0.286
Coupled	NAZCA [Fabian et al., 2020]	0.272
	M5P	0.636

6. Conclusion

Simulations based on multiscale numerical methods are seen computationally as a two-stage process. Predicting the execution time for these simulations should, therefore, consider adequate predictions in each stage. In this work, we applied our NAZCA methodology for building models to accurately predict the execution time of these simulations.

To evaluate our approach, we gathered performance data from simulations based on the MHM method applied to a diffusive process. For each stage of the simulation, we proposed a tree-based technique that, for its building, considers some parameters of the numerical method. On the leaves of the tree, we carried out empirical analyses for model selection. We then compared the selected models with an automated machine learning approach, Auto-WEKA. We concluded that our proposed technique is competitive when compared with the models selected by Auto-WEKA, with much less computational effort.

Many different pieces of future work could be considered for our research. One of them is related to the numerical simulations; we could consider other physics and dimensions to apply our methodology. We also intend to strengthen our technique by including performance data from other computational architectures. This is particularly important when we consider the asynchronous stage of the simulation. For a better prediction of the execution time, we will also consider the use of a hybrid approach combining the models obtained by our technique and the models obtained by Auto-WEKA. Finally, we will also consider the application of the methodology described herein to other multiscale numerical methods in which the two-stage process is observed.

References

R. Araya, C. Harder, D. Paredes, and F. Valentin. Multiscale Hybrid-Mixed Method. *SIAM Journal on Numerical Analysis*, 51(6):3505–3531, 2013.

R. Araya, C. Harder, A. H. Poza, and F. Valentin. Multiscale hybrid-mixed method for the Stokes and Brinkman equations – The method. *Computer Methods in Applied Mechanics and Engineering*, 324:29–53, 2017.

T. Arbogast, G. Pencheva, M. F. Wheeler, and I. Yotov. A multiscale mortar mixed finite element method. *Multiscale Modeling & Simulation*, 6(1):319–346, 2007.

T. Chaumont-Frelet and F. Valentin. A multiscale hybrid-mixed method for the Helmholtz equation in heterogeneous domains. *SIAM Journal on Numerical Analysis*, 58(2):1029–1067, 2020.

J. G. Cleary and L. E. Trigg. K*: An instance-based learner using an entropic distance measure. In *12th International Conference on Machine Learning*, pages 108–114, 1995.

Y. Efendiev and T. Y. Hou. *Multiscale Finite Element Methods*. Springer, 2009.

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Number 57 in Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, Boca Raton, Florida, USA, 1993.

J. H. L. Fabian, A. T. A. Gomes, and E. Ogasawara. Estimating the execution time of the coupled stage in multiscale numerical simulations. In *The Latin America High Performance Computing Conference (to appear)*, 2020.

I. Farmaga, P. Shmigelskyi, P. Spiewak, and L. Ciupinski. Evaluation of computational complexity of finite element analysis. In *11th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, pages 213–214, 2011.

R. T. Guiraldello, R. F. Ausas, F. S. Sousa, F. Pereira, and G. C. Buscaglia. The multiscale robin coupled method for flows in porous media. *Journal of Computational Physics*, 355:1–21, 2018.

C. Harder, D. Paredes, and F. Valentin. A family of Multiscale Hybrid-Mixed finite element methods for the Darcy equation with rough coefficients. *Journal of Computational Physics*, 245:107–130, 2013.

D. N. Hieu, T. Tieu Minh, T. Van Quang, B. X. Giang, and T. Van Hoai. A Machine Learning-Based Approach for Predicting the Execution Time of CFD Applications on Cloud Computing Environment. In *Future Data and Security Engineering*, pages 40–52, 2016.

S. Kim, Y. Suh, and J. Kim. EXTES: An Execution-Time Estimation Scheme for Efficient Computational Science and Engineering Simulation via Machine Learning. *IEEE Access*, 7:98993–99002, 2019.

L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 18(25):1–5, 2017.

P. Malakar, P. Balaprakash, V. Vishwanath, V. Morozov, and K. Kumaran. Benchmarking machine learning methods for performance modeling of scientific applications. In *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 33–44, 2018.

V. Martínez, F. Dupros, M. Castro, and P. Navaux. Performance Improvement of Stencil Computations for Multi-core Architectures based on Machine Learning. *Procedia Computer Science*, 108:305–314, 2017.

R. J. Quinlan. Learning with Continuous Classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, Singapore, 1992. World Scientific.

H. Silverman. *Cahuachi in the Ancient Nasca World*. University of Iowa Press, 1993. ISBN 9780877454076.

A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely. Modeling Power and Energy Usage of HPC Kernels. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pages 990–998, 2012.

E. Weinan. *Principles of Multiscale Modeling*. Cambridge University Press, 2011.

I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Amsterdam, 3 edition, 2011. ISBN 978-0-12-374856-0.