Performance Modeling of MARE2DEM's Adaptive Mesh Refinement for Makespan Estimation

Bruno da Silva Alves, Lucas Mello Schnorr
Graduate Program in Computer Science (PPGC)
Institute of Informatics – UFRGS
Porto Alegre, Brazil
{bsalves,schnorr}@inf.ufrgs.br

***Abstract*—Adaptive Mesh Refinement (AMR) is a widely known technique to adapt the accuracy of a solution in critical areas of the problem domain instead of using regular or irregular but static meshes. The MARE2DEM is a parallel application that employs the AMR technique to model 2D electromagnetics in oil and gas exploration. The modeling consists in iteratively applying a data inversion based on a set of measurements collected and registered by a survey on an area of interest. The parallelism of the MARE2DEM works by dividing the workload into a set of refinement groups that represent overlapping areas of the problem domain. Each refinement group can be computed independently of the others by a set of workers, carrying out the AMR in the meshes when necessary. The shape and compute performance of the refinement group depend directly of a set of user-defined parameters. In this article, we provide a method to estimate the MARE2DEM performance for all possible values that can be used in the influencing parameters of the application for a given case study. Our relatively cheap method enables the geologist to configure MARE2DEM correctly and extract the best performance for a given cluster configuration. We detail how the method works and evaluate its effectiveness with success, pinpointing the best values for the creating refinement groups using a real case study from the Marlim field on the coast of Rio de Janeiro, Brazil. Although we demonstrate our evaluation with this scenario, our method works for any input of MARE2DEM.**

***Index Terms*—Marine CSEM, Performance Modeling, MARE2DEM, Adaptive Mesh Refinement**

I. Introduction

Electromagnetic (EM) survey techniques have been standard for the offshore exploration of oil and gas in the recent past. An example of EM survey is the marine Controlled-Source Electromagnetic Method (mCSEM) [1]. The technique works by placing electromagnetic receivers in the ocean floor and moving a strong emitter close to the seabed in a predetermined area of interest. The collected information, comprising several data points of electromagnetic fields, must then be processed through a data inversion method to compute the most likely rock resistivity to explain those data points. Since oil and gas have known resistivity signatures compared to other materials, this valuable complements the knowledge provided by traditional seismic methods [2] [3]. The mCSEM method is also beneficial in regions where the application of the seismic method needs additional satisfactory answers [4].

The MARE2DEM stands for Modeling with Adaptively Refined Elements for 2D Electromagnetics and is an open-source and scalable parallel application that employs the Adaptive Mesh Refinement (AMR) technique to model 2D electromagnetics [5] [6] through data inversion. The modeling consists of costly operations based on the electromagnetism equations proposed by Maxwell [7]. In each iteration, the application carries out four steps. (1) It calculates the electromagnetic fields from the current resistivity model; (2) It determines the difference between the calculated fields and the fields from the mCSEM survey. This difference depends on the Jacobian and Smoothing phases to compute the error of the current model for each of the triangles in the mesh, called the *misfit*; (3) It then applies the Adaptive Mesh Refinement (AMR) method, refining the triangles with a large misfit, and in this way, breaking them down into smaller triangles using the Delaunay algorithm [8] using the Triangle software [9]; (4) It computes the global *misfit* after the AMR procedure, and a new resistivity model to be used in the next iteration. The data inversion continues until it meets the maximum allowed error (the target *misfit*) or reaches a specific number of iterations.

The MARE2DEM application requires the following inputs to carry out these four steps: (a) the geometry of the region of interest, (b) an initial resistivity model, (c) the collected mCSEM data, and (d) a set of user-defined parameters that control the level of parallelism. The region's geometry consists of a mesh of polygons to determine areas such as air, marine water, soil, and marine subsoil. The initial resistivity model breaks the inner regions of each polygon into triangles, thus defining a mesh of triangles. Each of these triangles receives an initial resistivity value for each of the described regions. Values can be arbitrary or defined by technical criteria from previous studies. The CSEM data contains the measurements of the survey consisting of the number of emitters (`Tx`), receivers (`Rx`), and frequencies (`Fq`). The last input is a set of user-defined parameters, including the target *misfit*, the maximum number of data inversion iterations, and a configuration directly influencing the maximum degree of parallelism.

The parallel architecture of the MARE2DEM application consists of one coordinator and a set of workers. During the initialization phase, the coordinator creates the work units consisting of a set of refinement groups based on the mCSEM survey data and the position of the receivers and emitters. The refinement group creation is dictated by the user-defined configuration determining the maximum number of emitters, receivers, and frequencies that should belong to each refinement group. Throughout this work, we identify such configuration with the following triple term: $\texttt{Tx}_{max}$-$\texttt{Rx}_{max}$-$\texttt{Fq}_{max}$. The application employs such limits to instantiate groups with a certain number of valid Tx-Rx pairs, according to the distance between the emitter and receiver [1]. As a result, we can have a different amount of refinement groups and several Tx-Rx pairs per group. The number of pairs per refinement group is known impact the computational performance [5] [6]. The application's coordinator process dynamically distributes the refinement groups among workers on demand whenever a worker becomes idle to tackle such compute diversity.

The user choice of a $\texttt{Tx}_{max}$-$\texttt{Rx}_{max}$-$\texttt{Fq}_{max}$ configuration is commonly used to instantiate many refinement groups whose total is at least equal to the same amount of workers. Very frequently, the user needs to be made aware of the performance impact of such a choice, despite similar choices being capable of generating completely different workloads. The Figure 1 provides an overview of how different the makespan can be in the same computing platform depending on the user choice of these parameters, ranging from 18 minutes to 3.5 hours here. We depict 50 configurations of our main case study for varying maximum numbers of emitters and receivers for the refinement groups, ordered from the faster to the slower configuration. The user generally has very little information to discern configurations that would lead to better performance.

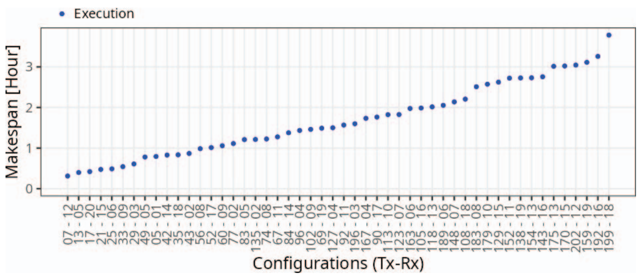

Fig. 1. The real execution time (Y-axis, in hours) of 50 user-defined possible configurations (on the X-axis, order from faster to slower).

Related Work. Predicting what would be the application performance involves both creating performance models of the fundamental arithmetic operations but also understanding how the AMR behaves, particularly for this type of mesh, where the errors tend to be greater closer to where emitters and receivers are positioned in the mesh of the mCSEM data. The authors have initially presented the performance modeling of the MARE2DEM [10], where the performance model indicates that the makespan is influenced by the number of vertices in the mesh and emitters (Tx). They also assess the scalability of the application in parallel computing systems. When generalizing the obtained performance model, they assume that a) any mesh needs four refinement operations to reach a good resistivity approximation and b) the mesh refinement doubles the number of vertices present. From this, the authors conclude that the best makespan appears when the number of processors is large enough to process the mesh for each emitter (Tx) separately.

Anticipating the AMR time is the most challenging task since the level of refinement required by each group depends on several characteristics such as group composition, frequency used, conductivity of the mesh regions, and distance between Tx and Rx, among others. Furthermore, exhaustively testing all configurations is unfeasible since there may be too many possible configurations as the space equals the product of Tx, Rx, and Fq present in the CSEM survey data. As far as we know, no methods attempt to predict the performance of mCSEM data inversion when AMR is part of the application. Existing methods [11] present predictions for regular meshes without refinement only. Consequently, a method capable of estimating the makespan for a given configuration is vital so the users of MARE2DEM may better choose initial parameters to control the parallelism.

Contribution. In this article, we describe our method to estimate the makespan of MARE2DEM with minimal cost. We detail the following contributions: (1) the characterization of the MARE2DEM performance, (2) a method for estimating mesh refinement and its compute performance, (3) estimating the execution order of several configurations at a cost much smaller than actually executing all configurations possibilities for a given case study. These contributions are possible through the combination of several steps involving the characterization of the application through the acquisition of execution traces, analysis of the traces using performance visualization methods [12], characterization of the refinement groups, and modeling of the primary operations present in the MARE2DEM application code. Estimating the refinement of the meshes made in each of the refinement groups is the critical point to determining the execution time of each configuration since it is expected that meshes that demand further refinement require a higher processing cost. We detail how the method works and evaluate its effectiveness with success, pinpointing the best values for the refinement groups creation, using a real case study containing the geoelectric model of the Marlim field in the state of Rio de Janeiro, Brazil [13].

The open-source MR3D (Marlim R3D) dataset [14] has been considered one of the essential standards for evaluating mCSEM data inversion methods because of the widespread knowledge about the oilfield characteristics. It possesses 25 horizontal lines (west-east) and 20 vertical lines (north-south) in which the emitter has evolved, capturing six frequencies ranging from 0.125Hz to 1.25Hz. We use one arbitrary line for our evaluation since the MARE2DEM application process a single line at a time, and the lines have similar characteristics. The Figure 2 depicts the chosen inline referenced as line `04Tx013a`. We show the model height and length on the Y and X axis. The A plot shows the initial mesh refinement with a heavily refined rectangular area with 64 Km of length and 6 Km of depth. The B plot shows a smaller area closer to the transmitters and receivers. The blue points identify the 206 transmitters, and the red triangles identify the 20 receivers placed on the irregular seafloor. Although we use

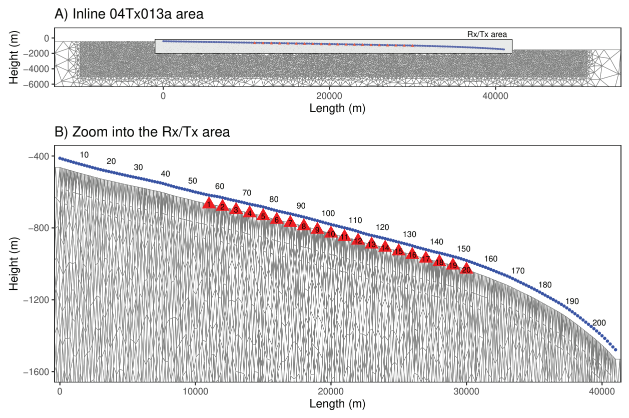

Fig. 2. Line `04Tx013a` of the MR3D dataset with the initial mesh (A), and the rectangular Rx/Tx area (B), where the numbered red triangles represents the 20 receivers, and the blue points represents the 206 transmitters.

this dataset, we expect that our method works for any input for the MARE2DEM application.

The rest of the paper is structured as follows. Section II presents our method to estimate the MARE2DEM's mesh refinement behavior and the application makespan, for a set of user configurations. Section III presents the experimental evaluation of the proposed method, including the makespan estimation for multiple configurations, enabling one to select the best possible configuration for a given computational platform, and considerations on the generality of the model definition. Finally, Section IV concludes this paper with a discussion and further considerations of investigation. A public companion[1] contains the software modifications, data, and instructions to reproduce our analysis.

II. Method: An estimate for MARE2DEM's mesh refinement and application makespan

The general goal is to determine which MARE2DEM's configuration is best for a given computing platform before the actual execution of the application. We describe the method to predict the MARE2DEM's makespan for different configurations. The Figure 3 depicts the method with its three main phases: (A) Performance Characterization (in blue, top box), (B) Makespan Estimative (in red, middle box), and (C) Method Evaluation (in green, bottom box). Each phase contains a set of subphases identified as the areas divided by the dashed lines, and each subphase contains the steps (rounded corner gray boxes) done to process the inputs (rectangular white boxes). The first phase describes the process of understanding the behavior of the application and its performance. The second phase is the core of our contribution, and it contains the effort to predict the application's makespan at a reasonable cost. Finally, the third phase consists of evaluating the proposed prediction against real executions in an HPC cluster. These three phases are the skeleton of this work, which has been stable throughout its development. The depicted subphases and steps within each phase result from iterative refinements over the method and represent our research's current and final state. The following subsections (Section II-A, Section II-B, and Section II-C) describe in detail each of the phases and the processes adopted in the method.

A. Performance Characterization

The performance characterization phase of our method is where we identify the MARE2DEM's behavior and performance when running the data inversion. We adopt a similar strategy [15] for the performance characterization of MARE2DEM. However, we extend the earlier work by considering a list of configurations for trace acquisition, and not just one. Understanding the application's behavior from a broader perspective is essential since different configurations provide different workloads to be distributed among the parallel workers. The first phase of the method contains two subphases: Data Acquisition and Model Definition. We collect data for the Model Definition during the Data Acquisition subphase. Then, during the Model Definition, we determine the performance models for the relevant MARE2DEM operations when computing the CSEM data inversion.

The first subphase of Data Acquisition has three inputs: the MARE2DEM instrumented code, the MARE2DEM input files, and a list of configurations. After a profound inspection, we have manually instrumented the MARE2DEM source code to identify the code's relevant operations. We then mark those operations' scope to obtain the execution traces with performance-related parameters. Besides the operations instrumentation, we identify relevant factors within each region to also record them in the traces associated with each regions' scope. The MARE2DEM input files describe the application workload and consist of the region's geometry, the initial resistivity model, and the collected CSEM data of the survey. The last input is a list of configurations where each configuration is a triple integer value $\mathtt{Tx}_{max}$, $\mathtt{Rx}_{max}$, $\mathtt{Fq}_{max}$ that controls the workload parallel division as described in the Section I. The final step of this subphase is the MARE2DEM's execution for each of the listed configurations, gathering traces with substantial information for performance modeling.

The Figure 4 depicts one of the traces collected during the Data Acquisition. It shows the second refinement group of the configuration 1-1-1. We show the application's execution time at the X-axis and the function call stack level at the Y-axis. The first stack level shows the operation that processes the refinement group. The second stack level shows the subsets of the given refinement group, where we can observe that the groups are split into 30 subsets. The third and last stack level shows the smaller operations invoked for each subset. We refer to them as microkernels. The application applies the AMR during the `local_refinement` and `estimate_error`. The AMR algorithm refines the triangles surrounding the `Tx` and `Rx` during the `local_refinement`, while in the `estimate_error` it refines 10% of the mesh's triangles with the highest errors. The `em_derivs` and `em_primal`

[1]https://gitlab.com/Alves_Bruno/companion-sbac-pad-2023

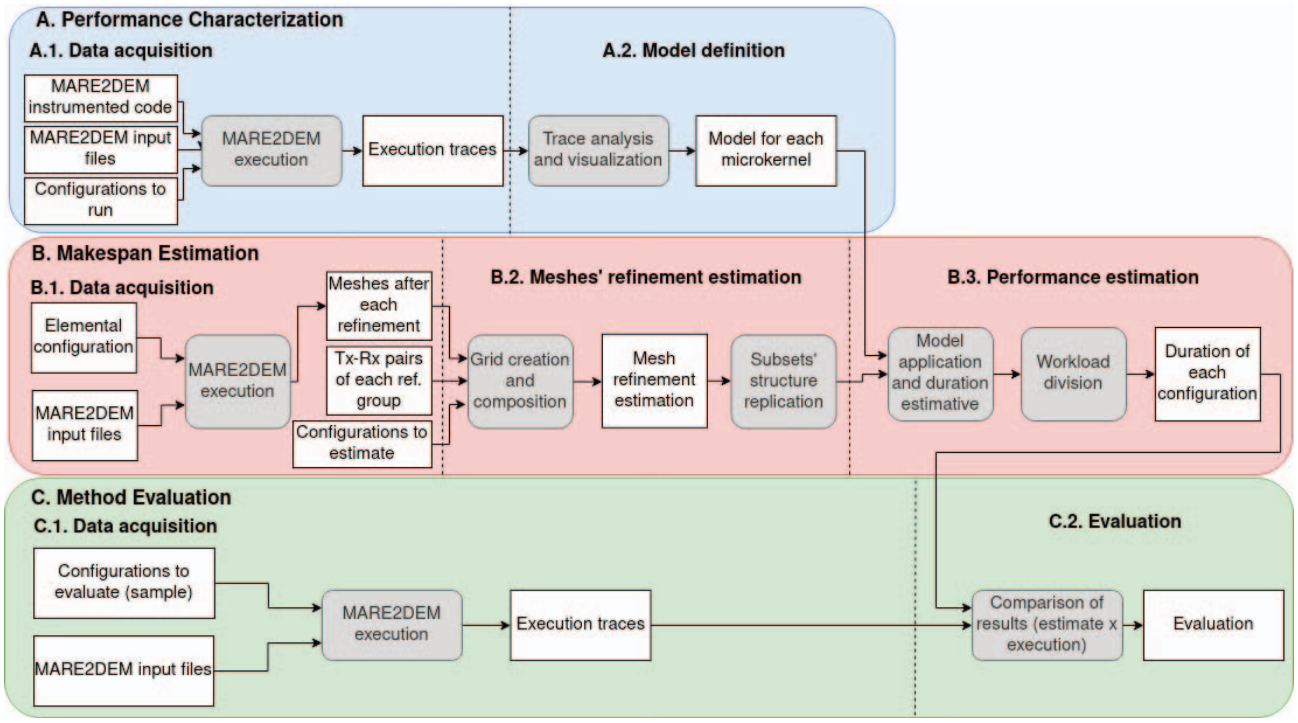
A. Performance Characterization
A.1. Data acquisition
MARE2DEM instrumented code
MARE2DEM input files
Configurations to run
MARE2DEM execution
Execution traces
A.2. Model definition
Trace analysis and visualization
Model for each microkernel
B. Makespan Estimation
B.1. Data acquisition
Elemental configuration
MARE2DEM input files
MARE2DEM execution
Meshes after each refinement
Tx-Rx pairs of each ref. group
Configurations to estimate
B.2. Meshes' refinement estimation
Grid creation and composition
Mesh refinement estimation
Subsets' structure replication
B.3. Performance estimation
Model application and duration estimative
Workload division
Duration of each configuration
C. Method Evaluation
C.1. Data acquisition
Configurations to evaluate (sample)
MARE2DEM input files
MARE2DEM execution
Execution traces
C.2. Evaluation
Comparison of results (estimate x execution)
Evaluation

Fig. 3. Our method's three phases (A, B, and C) to estimate the MARE2DEM's makespan for different configurations.

are the microkernels that process the refined meshes. Besides that, the AMR only happens in the subsets 1, 6, 11, 16, 21, and 26. The subsets in between those groups reuse the previously refined meshes. The bottom facet of Figure 4 shows a zoom covering only the first five subsets of the same refinement group. It shows that each `local_refinement` is followed by the `em_primal` and `estimate_error` microkernels. The iteration can repeat many times depending on the necessary mesh refinement to reduce the data inversion error. The in-between subsets (2 to 5) always have an `em_primal` operation followed by an `em_derivs` operation. Those operations describe the application's behavior during the Jacobian phase. This behavior is similar to the other refinement groups, but the `em_derivs` operation is absent in the Smoothing phase. The refinement groups in other configurations replicate this same behavior. However, the number of refinements needed per subset may change depending on the characteristics of the pairs present in each group.

The Model Definition subphase receives the previously collected traces as the input to determine the code's behavior through a series of analyses. The goal is to determine a model for each relevant region identified earlier during the code inspection. In this subphase, we explore performance visualization techniques to fully characterize the application as it has proven its value in different scenarios [12], including the geophysics context [16]. Then, we conclude the performance characterization phase by creating a performance model for each relevant code's region. We have considered the following

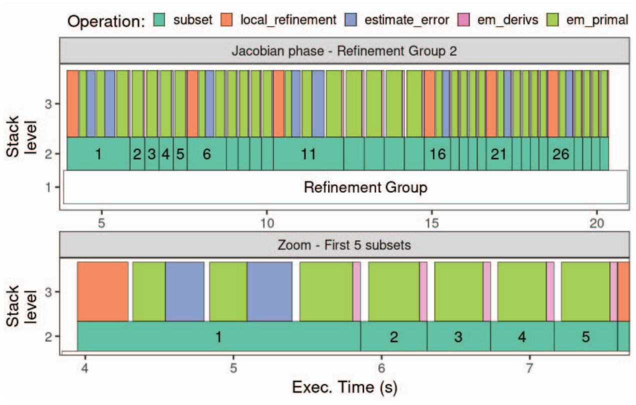

Fig. 4. Trace visualization of the second refinement group of the configuration 1-1-1. The top facet shows the Jacobian's phase refinement group, subsets and microkernels. The bottom facet shows a zoom into the X and Y axes by showing the first five subsets and the stack levels 2 and 3.

configurations 1-1-1, 2-1-1, 4-1-1, 1-2-1, 1-4-1, and 10-10-1. Our evaluations using these configurations have shown that the operations' duration mainly depends on the number of triangles present at each mesh. We consider those specific sets of configurations because they can represent the application's behavior when varying the maximum number of `Tx` (2-1-1, 4-1-1), the `Rx` (1-2-1, 1-4-1), and both (10-10-1).

Table I shows the best models (the second column) we were capable of obtaining after the trace analysis and visualization for each of the traced operations (the first column). The explanatory variables used on the multiple linear regression models to estimate the operation's duration consists of static information, such as the input parameters (nTx, nRx, nP), and dynamic information, such as the number of triangles in the mesh (nTri). The nTri parameter is the number of mesh triangles, while the nTx, nRx, and nP are the number of Tx, Rx, and pairs on each refinement group. The third column shows the quality of these models R^2, which can vary from 0 to 1, and values close to 1 are usually a good indicator that the model fits the data. Further details about the operations' models are given in Section III.

TABLE I
THE BEST PERFORMANCE MODEL FOR EACH MICROKERNEL.

Microkernel	Model	R^2
em_primal	duration ∼ nTri * nTx	0.994
em_derivs	duration ∼ nTri * nP + nTx + nRx	0.987
estimate_error	duration ∼ nTri * nTx	0.994
local_refinement	duration ∼ nTri * nTx * nRx * nP	0.835

B. *Makespan Estimation*

The models obtained in the previous phase showed that duration mainly depends on the number of triangles of the meshes. Therefore, predicting the mesh refinement algorithm is crucial to determine the duration of each MARE2DEM's configuration. In this way, this phase first addresses the AMR prediction during the Data Acquisition and Meshes' Refinement Estimation subphases, and then based on that, it calculates the makespan estimation during the Performance Estimation subphase. As mentioned, the refinement groups have a set of Tx-Rx pairs, and each of those pairs impacts the AMR by demanding a deeper refinement of the areas surrounding the Tx and Rx. Our solution to predict the AMR explores combining refinement from each pair of Tx-Rx.

The Figure 5 details the proposed solution by showing the first refinement group of the configuration 4-1-1 (top left image). This refinement group has four pairs of Tx-Rx, with 4 Tx and 1 Rx. The top right image shows the mesh state after the AMR for the area surrounding the Tx, and our solution aims to estimate such a final mesh refinement state. We only show this small region due to visualization, and for the sake of this explanation, however, this solution needs to consider the hole mesh. Then, the next step is to decompose the refinement group in its pairs and calculate the mesh refinement separately for each pair. We modify the MARE2DEM source code to write the meshes to binary files after each refinement operation during the CSEM data inversion. Then, we select the elemental configuration (1-1-1), which is the one with only one pair of Tx and Rx for each of the available CSEM frequencies. We then run the application and store the captured pair's meshes during the Data Acquisition. Figure 5 (middle) shows the meshes for the decomposed pairs of the select example group. The rectangles A to D show the mesh refinement demanded by each Tx, and the solution to replicate the AMR relies on composing those meshes together, as shown in Figure 5 (bottom). This solution can estimate the AMR for the refinement groups of any configuration with the cost of running only the elemental configuration and the costs of composing the pairs into more complex groups.

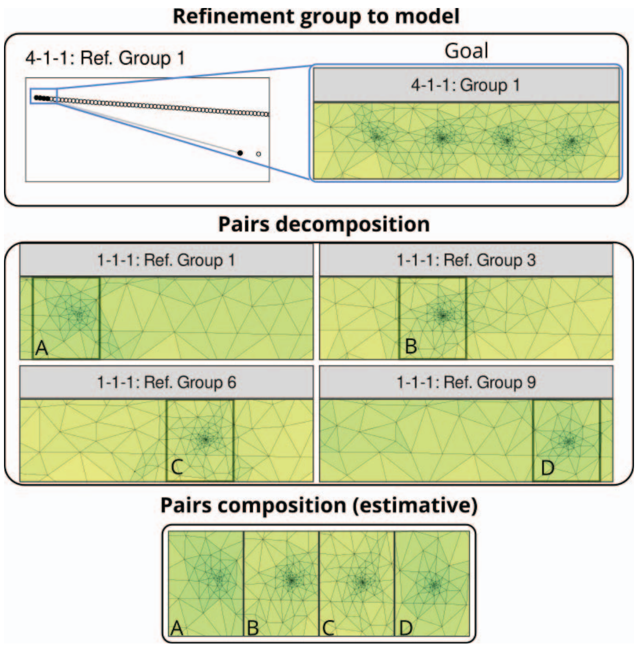

Fig. 5. An example of the proposed solution for mesh refinement estimation. We select the first refinement group of the configuration 4-1-1 and show its refined mesh on the top facet. The middle facet shows how the given group can be decomposed with refinement groups from the elemental configuration. The bottom facet shows the combination of elemental pairs to estimate the mesh refinement.

The composition of the refinement groups would take too much time to do by manually inspecting the meshes and grouping them. To tackle this issue, we need to automate the process. The Figure 6 depicts how we use dynamic grids for such automation. First, we create a grid for each of the pairs' meshes. This step dynamically divides the grid until all quadrants have only one triangle midpoint. With that, we compose the grids by searching for the most refined quadrant among the input grids. Figure 6 at Grid Composition (middle) shows an example of the composing algorithm used. Each graph node represents a quadrant, where the root node represents the main quadrant with the mesh's size. In blue, we visually represent the grids A, B and the resulting composed grid A+B. In the Result Check (bottom), we can see the final composed grids and a comparison of our estimation against the reference goal. Despite the minor differences, our estimation can even replicate the areas where the refinement is intense. If we compare the number of triangles on each grid (at the facet label), the estimation overestimates the number with a slight difference of ≈0.27% from the goal.

The Meshes' Refinement Estimation subphase receives as

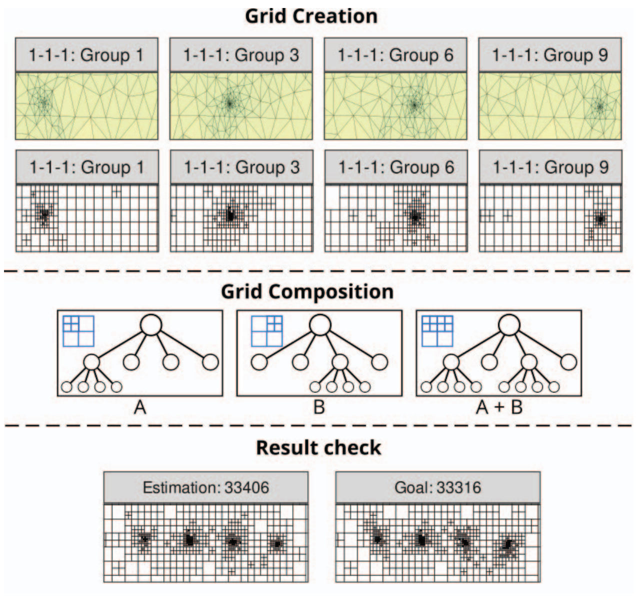

Fig. 6. The implementation of the proposed solution to estimate mesh refinement. The top facet shows the grid creation from the acquired meshes of the elemental configuration. The middle facet depicts the algorithm used to compose the given grids into one. We show the grids (in blue) and the quad-trees that represent the grids. The bottom facet compares the obtained grid with the grid created from the calculated mesh. The facet label shows the number of triangles on each grid.

input the previously captured meshes, a list of configurations to estimate the makespan, and the pairs of `Tx-Rx` present on each refinement group of the list of configurations. For example, the 7 depicts the Meshes' Refinement Estimation of a specific refinement group of the configuration 4-1-1. Our method generalizes for any refinement group at any configuration. The top facet shows the trace collected for the refinement group 1818, and the final goal is to estimate the duration of each microkernel as registered in the application traces. The first step for the duration estimation is the grid creation and composition, where we estimate the number of triangles on the meshes refined by the `local_refinement` and `estimate_error` microkernels. In the example, we focus on the subsets 1 to 5, and show the grid created from the mesh defined by the last `estimate_error` done in those subsets. The `local_refinement` and `estimate_error` microkernels refine the meshes differently. The first operates around the `Tx` and `Rx`, and the second operates in a broader area by refining the triangles with the highest errors. In order to represent the mesh refinement done by each microkernel, we select a smaller area for the `local_refinement` and a larger area for the `estimate_error`, as shown in the first grid. We show the grids of the pairs A, B, C, and D that composes the refinement group 1818. We also show the refinement iteration that occurred on each microkernel as it can repeat N times depending on the refinement demanded by the application. The $A + B + C + D$ facet shows the composed grids on each microkernel and iteration. Pairs in the same group can have different iterations, as shown in iteration 2 of the `estimate_error`. In those cases, we do the composition by adding the present grids with the grids from previous iterations, as depicted by the blue arrows. Then, we output the mesh refinement estimation, as the orange and blue blocks show with the respective number of triangles.

After the mesh estimation, the subsets' structure replication begins, as shown in Figure 7. We first replicate the subset structure of the goal trace by adding the remaining microkernels. We add the `em_primal` after the `local_refinement` iterations and after each `estimate_error`. Then, we add the `em_derivs` as the last microkernel. In order to replicate the subsets from 2 to 5, we copy the last `em_primal` and `em_derivs` microkernels. This procedure concludes the estimation for the first five subsets, and we repeat this process for the subsets 6 to 10, 11 to 15, 16 to 20, 21 to 25, and 26 to 30. At the end of the Meshes' Refinement Estimation, we output the number of triangles used as input for each of the microkernels present at the refinement groups of the list of configurations.

The last subphase is Performance Estimation, where we apply the previously defined models and estimate the duration of each microkernel. The number of triangles in the meshes previously estimated, the number of `Tx-Rx` pairs, and the number of `Tx` and `Rx` are the parameters used to estimate the duration with the performance models. After that, we aggregate the microkernels by combining operations from the same refinement groups. This step outputs a makespan for each of these groups. Finally, we define the configuration makespan by replicating the MARE2DEM's workload distribution algorithm for a predefined number of parallel workers of the target computing platform we intend to use.

C. *Method Evaluation*

This phase evaluates the makespan prediction obtained by comparing it to a list of configurations that have been actually executed. For the data acquisition, we define a sample of configurations to run the application since running all possible configurations demands a tremendous cost, even considering the execution of only one application iteration. We then collect the traces for each configuration and save the makespan. Then, the subphase of evaluation begins, where we compare our estimation to the execution. This phase outputs a final evaluation of the method that can be used as a new starting point to refine the method's subphases.

III. Evaluation and Results

We present the software and hardware environment configuration of our experiments. Then, we present the main results, including the makespan estimation and validation. The last part of our results includes a discussion involving considerations of our model definition.

A. *Experimental Setup: Software & Hardware*

We used a series of software tools in the development of this work. The Score-P-7.0 tool [17] captures the traces of the

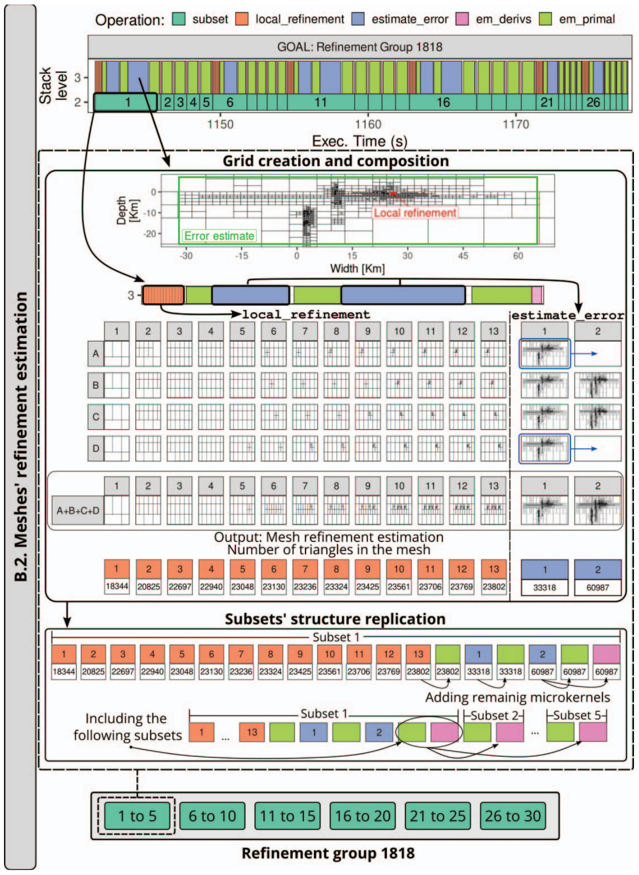

Fig. 7. The Meshes' Refinement Estimation subphase. We show the created grids and the composition for the `local_refinement` and `estimate_error` microkernels. We show regions with different areas due to the impact of each microkernel refinement. The areas are depicted in green and red. Then, we show the grids for each elemental group (A, B, C, and D) and the iterations of each microkernel. The A+B+C+D facet shows the composed grids. The orange and blue blocks show the number of quadrants on each composed grid. At the bottom, we show how we replicate the traces and each refinement group.

MARE2DEM application using the OTF2 format (Open Trace Format Version 2). The OTF2 files are then converted to CSV (Comma Separated Values) files with the otf2csv [2] tool. We improved the otf2csv tool to correlate the factors with the regions encapsulating each recorded factor. Furthermore, we incorporate routines into the MARE2DEM code to write the refined meshes into binary files. We develop the Rdgrid [3] tool for dynamic grids creation and composition. Finally, the experiments' creation, analysis, interpretation, and visualization heavily used of reproducible notebooks in org-mode format with the R language and the Tidyverse package. We made available a reproducible companion that contains the data and visualizations codes used to present the results. In order to run MARE2DEM, we use three computing nodes from our local cluster [4]. Each node has 2 Intel Xeon E5-2650v3 processors (20 cores, 40 threads) running at 2.3 GHz with 128GB DDR4 RAM. The nodes run the Debian 10 (buster) operating system with the Linux kernel 4.19.0-20-amd64 and are interconnected with a 1 Gbit/s local network. We tell OpenMPI 3.1.4 to use all cores of each node, leading to 20 processes per node that were mapped into 59 MARE2DEM workers and one coordinator.

The MARE2DEM can take up to 4 hours to run a single iteration in the platform for the given input configuration. At the same time, we need a representative set of experiments to evaluate the effectiveness of our method for the makespan estimation. In this way, we select a sample of 50 configurations from the 8240 available options. We apply the Latin Hypercube Sampling (LHS) [18] method to generate a near-random sample that can cover a significant area when considering the configurations that use one CSEM frequency. The Figure 8 shows the configurations adopted for validating our method, with the number of `Tx` in the X-axis and the number of `Rx` in the Y-axis. We used those configurations to run and trace the MARE2DEM and as input for the makespan estimation.

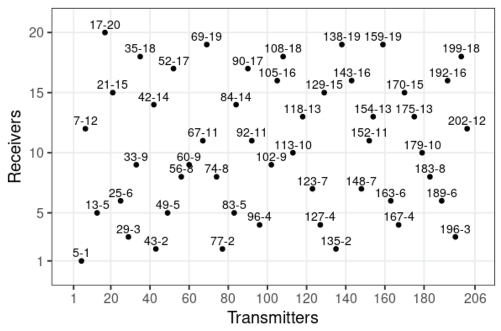

Fig. 8. The sample of configurations used for the method validation. We apply the Latin Hypercube Sampling (LHS) method to generate the near-random sample.

B. Makespan estimation for multiple configurations

The performance estimation provided by our method allows one to estimate the makespan for any given configuration. However, the estimation comes with the cost of: (A) obtaining the elemental meshes for each CSEM `Tx`-`Rx` pair; (B) estimating the meshes' refinement with the creation and composition of the grid; (C) defining the microkernels model; and (D) applying the model for each microkernel to later estimate the makespan. The makespan estimation spends most of the time on A and C, since those steps involve the execution of MARE2DEM. In C, we need to calibrate the microkernels' model with a representative application trace. The time spent in B is relatively low since we can run the grid creation and composition in parallel workers for each subset in the refinement groups. Furthermore, we use a caching system by reusing the already composed grids shared among different configurations. Despite the involved costs, we assume our

[2]otf2csv: https://github.com/schnorr/otf2utils

[3]Rdgrid: https://github.com/Alves-Bruno/rdgrid

[4]UFRGS-PCAD cluster: http://gppd-hpc.inf.ufrgs.br/

method has a low cost since we only need to run two instances of the MARE2DEM in order to estimate the makespan for any of the 8240 possible configurations.

We apply our method to estimate the makespan of the 50 sample configurations. In Figure 9, we compare the makespan estimation against the real makespan of MARE2DEM for the configurations detailed in the X-axis. The execution makespan determines the order of the configurations from the fastest to the slowest. The gray bars show the error of the estimation. One can observe that the estimation error grows as the execution makespan grows. The estimation achieves its best results when the execution is faster, as in the first 16 configurations. However, estimating precisely the makespan of each configuration is nonessential since we only want to indicate the fastest configuration to the geologist in charge of running the application. In this sense, we classify the estimate as reasonable if it can able to represent the order of growth of the execution makespan.

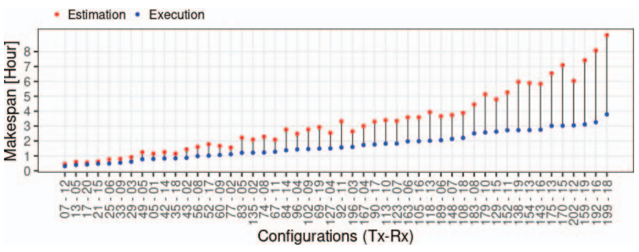

Fig. 9. Makespan estimation compared with running each configuration. Despite the differences in the absolute values, our method can indicate the order of growth for the configurations makespan.

The Figure 9 shows the makespan estimation for the 50 sample configurations. It shows the makespan estimation in red and the execution makespan in blue. The X-axis shows the configurations ordered by the makespan of the execution, and the Y-axis shows the makespan obtained in hours. The first ten faster estimations achieve similar makespan values compared to the execution. One can observe that our method overestimates the makespan for all configurations. Despite the differences in the makespan estimation, our method can indicate the order of the makespan. Successfully indicating the order of the configuration's duration is considered more important than obtaining an exact estimation, as the order determines which configuration the geologist should consider to extract the best performance of the cluster. To depict how good we are from the order perspective between the select configurations, the Figure 10 shows the order of growth of the estimation. The X-axis shows the execution order, and the Y-axis shows the estimation order. The blue diagonal line represents the best estimation, which got the order of all configurations right. The red line represents the estimation order. In general, our estimate manages to capture the makespan behavior. The estimation order is better for the first ten and the last 15 configurations, closely matching the execution order.

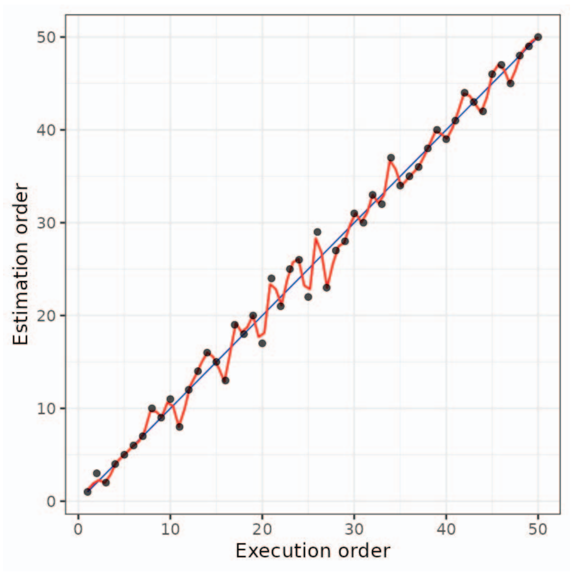

Fig. 10. The makespan estimation order compared to the execution order. The blue line represents the best estimation. The cases where the red line can closely match the diagonal line is considered a good estimation.

C. Considerations on the performance model definition

The makespan estimation relies on the mesh refinement estimation and the microkernels' performance models. The model receives the traces from a given configuration to instantiate it. Different inputs to the model can lead to different calibrations since the configurations can lead to refinement groups with particular characteristics. In this section, we evaluate if the trace used to instantiate the model can impact the quality of our estimation. We investigate by calibrating the microkernels' models with each of the 50 available traces. Then, we estimate the makespan with the 50 obtained models, and each estimation produces a result similar to the one shown in Figure 9. The Figure 11 summarizes the quality of the obtained estimation for each of the 50 models. The points in the figure represent each of the traces used to calibrate the models, while the number of `Tx` and `Rx` of those traces are used to position each point in the X and Y axes. The points' colors depict the quality of the estimation by showing the order error. The order error measures the estimation quality by aggregating the distance from the execution order to the estimated order for each configuration. In summary, the blueish points represent reasonable estimations, which can indicate the order of the execution. The black circles highlight the cases where we could not indicate the execution order with good precision. The cases in which the estimation fails are related to the calibration of the model with traces of configurations with only a few emitters or receivers. The blue rectangle shows the area where we believe configurations should be selected from, especially for model calibration because there are no cases where the estimation fails inside this region.

We depict the problematic estimations in Figure 12. The

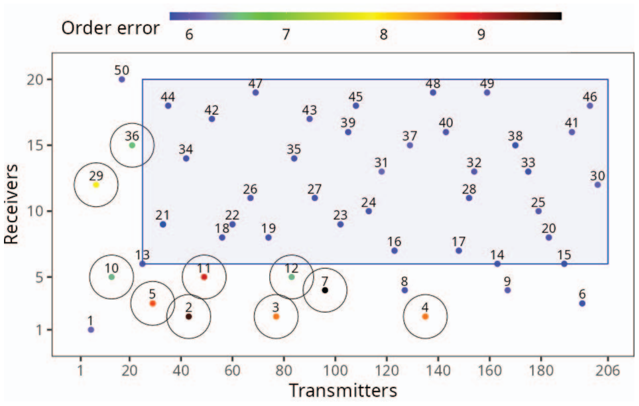

Fig. 11. The quality of the estimation when using different traces for the model instantiation. The blueish points represent the cases where the estimation successfully indicates the order of the execution. The black circles highlight the cases where the estimation fails.

estimations 2, 3, and 4 present similar behavior, where our result underestimates most of the configurations' makespan by assigning an estimation value closer to 0. In those cases, the configuration used to calibrate the model only had 2 `Tx`, resulting in a model that could not extrapolate the duration of the microkernels from configurations with a higher number of `Rx`. The estimations 5, 7, 10, 11, and 12 also use configurations with fewer `Rx`. However, in those cases, the estimations for the first 35 configurations are better against the models with only 2 `Rx` (facets 2 to 4). The estimation for the slower configurations could be more capable of achieving good results. Finally, the last two cases, numbers 29 and 36, have more `Rx` but the limitation factor is the number of `Tx`.

The results show that our method can successfully indicate the execution order when using most of the executed configurations for model calibration. However, the Figure 11 shows that most failed estimation relies on calibrating the model with a configuration that has less than 5 `Rx`. In this way, we conduct another set of estimations by removing the `local_refinement` microkernel from the duration estimation since the duration of this microkernel is influenced by the number of `Rx`. The number of `Rx` also influences the `em_derivs`'s duration, but we do not remove this microkernel because the number of CSEM pairs is the most impacting factor after the number of triangles at the mesh. The Figure 13 shows the estimated order when using each available trace for model calibration, but now considering only the `local_refinement`, `em_primal`, and `em_derivs` operations. We show that all configurations can be used for the model calibration without impacting the estimation quality, as the red line follows the behavior of the best estimation case.

IV. Conclusion and Future Work

This work presents a performance characterization and modeling study for the MARE2DEM, an oil and gas exploration application that uses the established CSEM method for data inversion. We developed a low-cost method to estimate the MARE2DEM's makespan for different configurations. We aim to provide a tool for helping the geologist decide which configuration can extract the best performance for a given cluster configuration. Our method includes three main phases: Performance characterization, Makespan estimation, and Method evaluation. In the first phase, we investigate the application's performance by analyzing and visualizing traces collected during the data inversion. We identify that the application's configuration determines the workload's size and characteristics by defining the maximum number of `Tx`, `Rx`, and `Fq` that each workload, called refinement groups, can have. We identify four operations of the refinement group's processing, which we call microkernels because they represent the building blocks to calculate the CSEM data inversion on each refinement group. Then, in the second phase, we present the core of our contribution, where we detail the process to obtain the makespan of any configuration (8240 options for our case study) by paying the cost of running only two configurations: 1) the elemental configuration used to obtain the refined meshes, and 2) a representative configuration used to calibrate the models established during the first phase. Then, in the third phase, we validate our model with a sample of 50 configurations from the LHS method to cover a representative area of the configuration space with one CSEM frequency.

The presented results show that our method can effectively estimate the order of the makespan for the chosen sample of configurations. The estimations show that the error is smaller for the faster configurations and more significant for the slowest ones. Our investigation shows that this error results from the mesh refinement estimation error, as the lowest configurations demand a higher mesh refinement that our grid estimation procedure cannot replicate. As a consequence, further investigation into this topic is needed. However, the top priority of our work is to indicate the order of the configuration's duration and not the duration precisely. We also show that the configuration used to calibrate the method can impact the quality of the order estimation. However, we overcome this issue by removing the `local_refinement` microkernel from the performance estimation subphase. We also consider that the sampling size and filter for the configuration with one CSEM frequency may represent a limitation of our work. However, we were limited by the execution time demanded by each configuration. The presented method also is limited to homogeneous computing platforms. Thus, as future work, we intend to validate and extend our method in heterogeneous platforms to explore a more extensive case study.

Acknowledgements

This study was financed in part by the "Coordenação de Aperfeiçoamento de Pessoal de Nível Superior" - Brasil (CAPES) - Finance Code 001, the National Council for Scientific and Technological Development (CNPq), and the projects FAPERGS (16/354-8, 16/488-9, 19/711-6), the Inria associated team ReDaS, the CNPq (447311/2014-0), the CAPES (Brafitec 182/15, Cofecub 899/18), and Petrobras (2018/00263-5). Some experiments in this work used the PCAD infrastructure, http://gppd-hpc.inf.ufrgs.br, at INF/UFRGS.

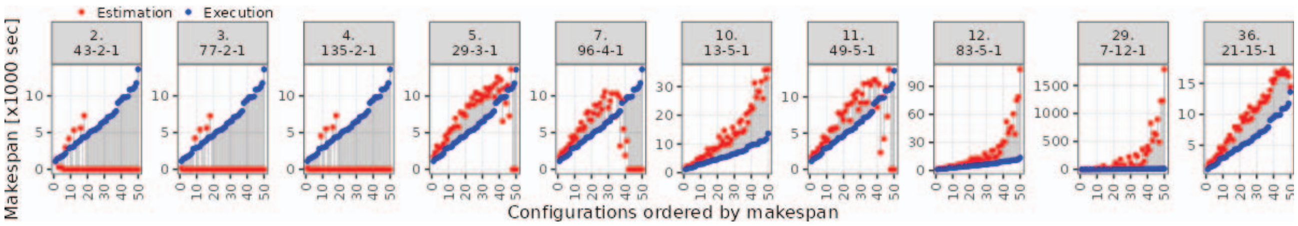

Fig. 12. The makespan estimation for the cases where the order error indicates a wrong estimation.

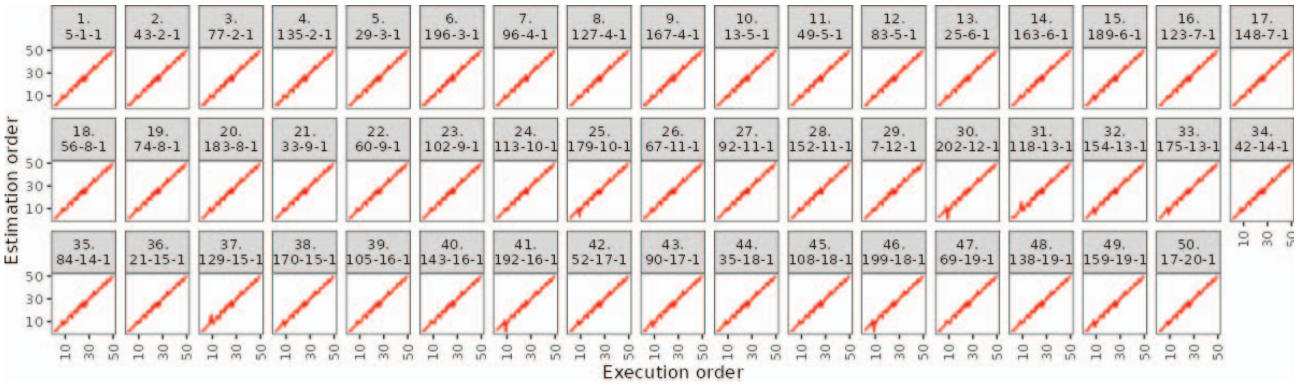

Fig. 13. The estimation order considering only three operations and just one trace out of the 50 for model calibration (facets). We can see that no matter which trace is used for calibration, our performance model captures the order with sufficient precision.

References

[1] K. Key, "1D inversion of multicomponent, multifrequency marine CSEM data: Methodology and synthetic studies for resolving thin resistive layers," *Geophysics*, vol. 74, no. 2, pp. F9–F20, 02 2009.

[2] S. Constable, "Ten years of marine csem for hydrocarbon exploration," *GEOPHYSICS*, vol. 75, no. 5, pp. 75A67–75A81, 2010. [Online]. Available: https://doi.org/10.1190/1.3483451

[3] P. T. L. Menezes, J. L. Correa, L. M. Alvim, A. R. Viana, and R. C. Sansonowski, "Time-lapse csem monitoring: Correlating the anomalous transverse resistance with sophih maps," *Energies*, vol. 14, no. 21, 2021. [Online]. Available: https://www.mdpi.com/1996-1073/14/21/7159

[4] R. Cooper and L. MacGregor, "Csem: Back from the brink," *GEO ExPro Magazine*, 2020.

[5] K. Key, "MARE2DEM: a 2-D inversion code for controlled-source electromagnetic and magnetotelluric data," *Geophysical Journal International*, vol. 207, no. 1, pp. 571–588, 08 2016. [Online]. Available: https://doi.org/10.1093/gji/ggw290

[6] K. Key, "MARE2DEM: an open-source code for anisotropic inversion of controlled-source electromagnetic and magnetotelluric data using parallel adaptive 2D finite elements (Invited)," in *AGU Fall Meeting Abstracts*, vol. 2013, Dec. 2013, pp. GP22A–01.

[7] M. S. Zhdanov, *Geophysical electromagnetic theory and methods*. Elsevier, 2009.

[8] J. R. Shewchuk, "Delaunay refinement algorithms for triangular mesh generation," *Computational geometry*, vol. 22, no. 1-3, pp. 21–74, 2002.

[9] ——, "Triangle: Engineering a 2d quality mesh generator and delaunay triangulator," in *Workshop on applied computational geometry*. Springer, 1996, pp. 203–222.

[10] K. Key and J. Ovall, "A parallel goal-oriented adaptive finite element method for 2.5-D electromagnetic modelling," *Geophysical Journal International*, vol. 186, no. 1, pp. 137–154, 07 2011. [Online]. Available: https://doi.org/10.1111/j.1365-246X.2011.05025.x

[11] J. I. Dagostini, H. C. P. da Silva, V. G. Pinto, R. M. Velho, E. S. L. Gastal, and L. M. Schnorr, "Improving workload balance of a marine csem inversion application," in *IEEE Intl. Par. and Distr. Processing Symposium Workshops (IPDPSW)*, 2021, pp. 704–713.

[12] K. E. Isaacs, A. Giménez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer, "State of the Art of Performance Visualization," in *EuroVis - STARs*, R. Borgo, R. Maciejewski, and I. Viola, Eds. The Eurographics Association, 2014.

[13] T. M. Nascimento, P. T. L. Menezes, and I. L. Braga, "High-resolution acoustic impedance inversion to characterize turbidites at marlim field, campos basin, brazil," *Interpretation*, vol. 2, no. 3, pp. T143–T153, 2014. [Online]. Available: https://doi.org/10.1190/INT-2013-0137.1

[14] J. L. Correa and P. T. L. Menezes, "Marlim r3d (mr3d) - the full azimuth csem dataset," May 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1256787

[15] B. da Silva Alves, L. P. Gaspary, and L. M. Schnorr, "Towards Parameter-Based Profiling for MARE2DEM Performance Modeling," in *High Performance Computing*, P. Navaux, C. J. Barrios H., C. Osthoff, and G. Guerrero, Eds. Springer Intl., 2022, pp. 63–77.

[16] A. L. Veroneze Solórzano, L. Leandro Nesi, and L. Mello Schnorr, *Using Visualization of Performance Data to Investigate Load Imbalance of a Geophysics Parallel Application*. New York, NY, USA: Association for Computing Machinery, 2020, p. 518–521.

[17] A. Knupfer, C. Rossel, D. a. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschuter, M. Wagner, B. Wesarg, and F. Wolf, "Score-p: A joint performance measurement run-time infrastructure for periscope,scalasca, tau, and vampir," in *Tools for High Performance Computing 2011*, H. Brunst, M. S. Muller, W. E. Nagel, and M. M. Resch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 79–91.

[18] M. Stein, "Large sample properties of simulations using latin hypercube sampling," *Technometrics*, vol. 29, no. 2, pp. 143–151, 1987.