Prediction of Distributed Ultrasound Simulation Execution Time Using Machine Learning

Jiri Jaros
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
jarosjir@fit.vutbr.cz

Marta Jaros
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
jarosmarta@fit.vut.cz

Martin Buchta
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
tamabuch@gmail.com

2024 IEEE Congress on Evolutionary Computation (CEC) | 979-8-3503-0836-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/CEC60901.2024.10611947

***Abstract*—This study introduces a comprehensive system designed to predict the execution time of k-Wave ultrasound simulations, factoring in the domain size and allocated computing resources. The predictive models, developed using symbolic regression and neural networks, were trained on historical performance data acquired from the Barbora supercomputer. For domain sizes with optimal parameters, the symbolic regression model outperformed, achieving an average error of $5.64\,\%$. Conversely, the neural network showed commendable efficacy in general domain scenarios, with an average error of $8.25\,\%$. Notably, in both instances, the average error remained below the 10% threshold, aligning closely with the uncertainty inherent in the measured data and the execution of real large-scale jobs. Consequently, this predictive system is well-suited for deployment in resource optimization frameworks, significantly enhancing the efficiency of large-scale simulation executions.**

***Index Terms*—Prediction of Execution Time, Moldable tasks, Symbolic Regression, Neural Network, Supercomputer, Simulation, k-Wave, Ultrasound, HeuristicLab.**

I. Introduction

Biomedical ultrasound finds pivotal applications in the dynamic landscape of cancer diagnosis and treatment advancements. Noteworthy applications include non-invasive ablation of cancerous tissues, precision-targeted drug delivery, diagnostic imaging, and neurostimulation for conditions like epilepsy, Alzheimer's, or Parkinson's disease. In contrast to conventional approaches such as biopsy, open surgery, and radio- or chemo-therapy, ultrasound distinguishes itself by being non-invasive, non-ionizing, and associated with fewer post-treatment complications [1].

To customize therapeutic ultrasound procedures according to individual patient requirements, it is essential to assess complex physical models beforehand. One widely adopted physical model within the global scientific community is the open-source k-Wave toolbox designed for the time-domain simulation of acoustic waves propagating in tissues in 1, 2, or 3 dimensions [2]. The toolbox has been optimized to support various computing architectures, encompassing multi-core processors, graphics processing units, and particularly, distributed high-performance computing clusters consisting of numerous computing nodes [3], [4].

Over the last decade, k-Wave has attracted a lot of interest amongst biomedical physicists, ultrasonographers, neurologists, oncologists and other clinicians. Numerous applications of k-Wave have been reported in photoacoustic breast screening [5] and transcranial brain imaging [6]. k-Wave has also been used for exciting applications in HIFU, including treatment planning of kidney [7], liver [8] and prostate tumour ablations [9], ultrasound neurosurgery and targeted drug delivery [10], and neurostimulation [11].

These applications demand intricate and resource-intensive computations that typically surpass the capabilities of desktop computers or small servers. Therefore, it becomes crucial to delegate the computational workload to cloud or HPC clusters. Furthermore, these applications are not characterized by a single program execution; instead, they involve complex computing workflows. To address the execution complexity of ultrasound workflows, various workflow management systems, including tools like k-Dispatch or Pegasus, have been developed [12], [13]. These systems aim to achieve three primary objectives: (1) provide a user-friendly interface between users and HPC facilities, shielding them from technical details, (2) ensure efficient workflow execution in terms of time and cost, and (3) monitor and account for workflow execution. This paper specifically concentrates on the second objective: efficient performance-cost execution of complex ultrasound workflows on large-scale distributed computing clusters.

The physical models within the k-Wave toolbox are implemented as distributed moldable programs. This allows users to specify execution parameters based on program scaling, input data size and complexity, or actual cluster utilization. Execution parameters, in this context, refer to the amount of computing resources (number of computing nodes or processor cores) assigned to a particular task. However, as performance scaling is never perfect, the computing cost may increase with the amount of resources used. Performance scaling may also exhibit various oscillations, rendering some execution parameter configurations ineffective due to workload imbalance. It's also worth noting that the scaling behavior in general is strongly influenced by processor architecture, memory subsystem and interconnection. Moreover, data collected on one system is usually not portable across different systems.

Given the finite nature of available resources and the competition among users and workflows for these resources, it is critical to optimize the resource allocation for each task in the workflow. Numerous strategies have been developed

for fine-tuning the execution parameters of specific tasks. These include approaches like Greedy Algorithms, Simulated Annealing, and Genetic Algorithms, which focus on various objectives such as minimizing total execution time, reducing computational costs, improving cluster utilization, and decreasing queuing times [14]–[16].

These algorithms typically involve generating candidate solutions that define the execution parameters for individual tasks. These solutions are then evaluated by a cluster simulator, which measures the effectiveness of the workflow schedules against specific goals like execution time, cost, and queuing times, while complying with cluster execution policies and considering task priorities and dependencies.

However, prior to this evaluation, it's crucial to accurately estimate or predict the task execution time based on the chosen execution parameters and the type of simulation. This prediction is challenging due to the vast state space involved, as the simulation domain can vary in size from something as small as a fingertip to as large as an entire abdomen or head, and the computing resources can range from a single CPU core to hundreds of nodes with tens of thousands of cores.

Therefore, the primary objective of this paper is to develop a robust method that can reliably predict the execution time for a specific ultrasound task, considering its input data and execution parameters.

II. Proposed Technique

This section provides a comprehensive explanation of the simulation code, the focal point for estimating execution time. Following this, the process involved in gathering and preprocessing the data for training, validation, and testing will be meticulously outlined. Finally, we present two unique predictors and delve into the nuances of their implementation.

A. Performance Behaviour of k-Wave Simulations

The k-Wave distributed simulation code employs the Message Passing Interface (MPI) to facilitate collaboration among multiple computing processes, enabling them to work together by exchanging messages over the network. This approach involves decomposing and distributing the simulation domain across the cooperating processes, inherently represented as a Cartesian grid. Typically, the domain size varies between 128 and 2048 grid points in one direction.

To achieve parallelism, the simulation domain is partitioned into 2D slabs that are distributed among specific computing processes, referred to as ranks in MPI terminology. Each rank is mapped to a single computing core, and the application can span across hundreds of computing nodes (see Fig. 1). However, the slab decomposition imposes a limitation on the suitable number of ranks, restricting it to integer divisors of the z domain size. Choosing any other number may result in workload imbalance, as some ranks would have to process more slabs than others.

The strong scaling [17] of the code is determined by the time complexity of each simulation step. As k-Wave functions as

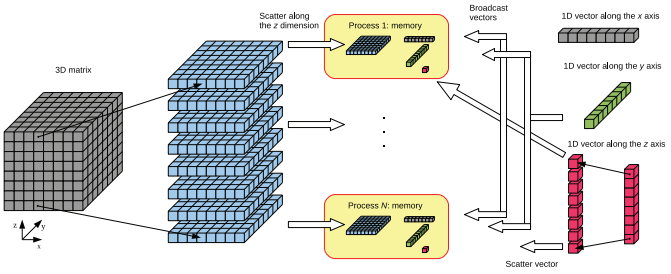

Fig. 1. Domain decomposition of the 3D simulation space in the distributed version of the k-Wave solver. The simulation space is divided into 2D slabs distributed among the computing processes [3].

a pseudospectral wave equation solver, its performance heavily depends on the computation of Fast Fourier Transforms (FFTs). Approximately 60% of the execution time is dedicated to calculating 3D FFTs. The remaining computations involve element-wise matrix operations with linear time complexity. Consequently, the code exhibits an asymptotic time complexity of $O(N \log N)$, dictated by the FFT.

However, the performance of FFT is not uniform across arbitrary domain sizes. It is well-established that optimal performance is achieved only for domain sizes that can be factorized into small primes (2, 3, 5, and 7) [18]. Otherwise, the computation time can increase by an order of magnitude.

Additionally, the distributed 3D FFT necessitates a global data exchange in the form of matrix transposition. This significantly impacts the interconnection network. The number of messages to be exchanged grows with P^2, where P represents the number of ranks. Furthermore, as the number of ranks increases, the messages become smaller, leading to reduced network bandwidth and increased latency. Since no distributed cluster offers a full interconnection, there is always competition over the interconnection links between messages, making accurate prediction of the execution time challenging.

Another factor influencing k-Wave's performance is the distribution of ranks over computing nodes. k-Wave is established as a memory-bound application, suggesting that it is often advantageous to underpopulate computing nodes by leaving some computing cores unused. This approach prevents memory congestion and allows for the utilization of more computing nodes, thereby achieving higher aggregated network bandwidth. However, since it is not feasible to allocate node partitions, users can only specify an integer number of nodes to be assigned to the computation. The optimal number of ranks is then calculated by following equations as a highest domain size factor smaller than the product of an assigned number of nodes and cores per node, see Eq. (1).

$$ranks = \max(factor(Nz) < cores * nodes) \tag{1}$$

$$occupation = \frac{\lceil ranks/nodes \rceil}{cores} \tag{2}$$

$$imbalance = \frac{Nz \bmod ranks}{ranks} \tag{3}$$

Here, Nz represents the number of grid points along the z direction, $cores$ denotes the number of cores per node, and $nodes$ signifies the assigned number of nodes. Occupancy is calculated as the fraction of active cores and load imbalance as the fraction of fully underutilized ranks (with full workload).

For instance, consider a simulation domain size of 1024^3 and an HPC system with 36 cores per node, aiming to use 15-18 nodes. The core range is 540-648, with the largest 1024 divisor being 512, leading to a node occupancy of 0.948 (15 nodes) to 0.79 (18 nodes). The ranks are evenly distributed, and each processes 2 slabs, resulting in perfect load balance. However, in cases like distributing this domain across 300 ranks in 10 nodes, we see an occupancy rate of 0.83 and a load imbalance of 0.413, leading to 6 idle cores per node and uneven slab allocation among ranks.

Fig. 2 illustrates the strong scaling efficacy of the k-Wave solver. The number of ranks is chosen to maximize occupancy and minimize load imbalance. This figure elucidates the variation in computational time as the number of allocated computing nodes increases for different cube simulation domains. While the scaling trends are near-ideal (implying that doubling the resources halves the computation time), it's important to note that the curve's slope and steps are influenced by factors such as occupancy and load imbalance, in accordance with Eqs. (2) and (3). Additionally, the increasing communication overhead also plays a significant role in this scaling behavior.

B. Collection of Performance Data

Given the impracticality of collecting a comprehensive dataset for every combination of simulation domain size and the assigned number of computing nodes, a representative dataset was gathered. The target HPC system, where k-Wave was deployed is the Barbora cluster. Barbora consists of 201 compute nodes, totaling over 7 thousand compute cores with 44 TB RAM. Nodes are interconnected through a fully non-blocking fat-tree InfiniBand network.

The dataset used for training includes 1,813 instances of the k-Wave simulation. The simulations covered a total of 39 different domain sizes ranging from 128^3 to 3072^3 grid points, consisting of 27 cubes, 10 prisms, and 2 cuboids. Seven of these dimension sizes exhibited suboptimal factorization into prime numbers (factors exceeding 7). For each domain size, simulations were executed using 1 to 32 computing nodes. Among the 1,813 instances, 414 cases indicated non-ideal distribution of the domain across available nodes.

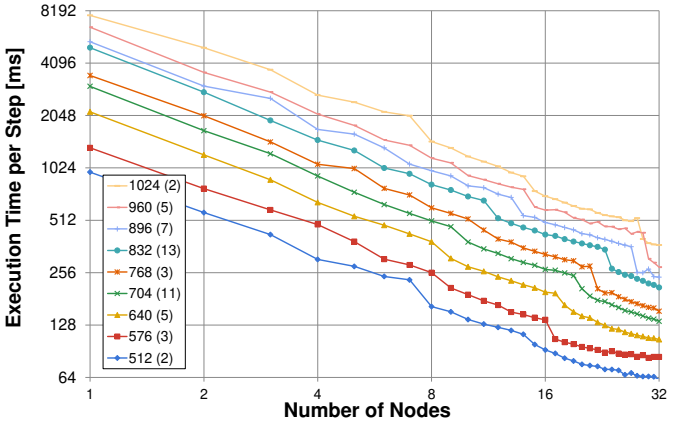

Fig. 2. Strong scaling performance of the distributed k-Wave solver across various domain sizes (highest factor indicated in brackets) and different numbers of computing nodes.

Gathering performance data is both time and cost-intensive in our context. In fact, the collection of the performance dataset utilized almost 4,000 node hours on the Barbora cluster, equating to over 10,000 USD.

For each instance, the following features were recorded:

- **Nx** – Number of grid points along the x direction,
- **Ny** – Number of grid points along the y direction,
- **Nz** – Number of grid points along the z direction,
- **Fx** – Highest prime factor of Nx,
- **Fy** – Highest prime factor of Ny,
- **Fz** – Highest prime factor of Nz,
- **Nodes** – Number of assigned nodes,
- **Ranks** – Number of assigned MPI ranks,
- **Occupancy** – Fraction of used cores, Eq. (2),
- **Imbalance** – Fraction of underutilized ranks, Eq. (3),
- **Time** – Execution time per simulation time step in ms.

C. Filtration of Performance Data

In the process of data collection, an extensive and varied database encompassing diverse simulation sizes and execution parameters was compiled. The database incorporates conventional production domain sizes (cubes ranging from 512^3 to 1024^3 with low prime factors) as well as more unconventional domain sizes with high prime factors and suboptimal distribution across ranks.

Nevertheless, this diversity has the potential to introduce noise into the training dataset and diminish the model's quality for suitable domain sizes and well-distributed scenarios. To address this, three distinct data filters were devised:

1) cubes, denoted as C,
2) suitable factor, denoted as F,
3) suitable decomposition, denoted as D.

These filters result in eight distinct datasets:

1) no filter (-),
2) cubes only (C),
3) domains with suitable factor (F),
4) domains with suitable decomposition (D),
5) domains with suitable factor and decomposition (FD),
6) cubes with suitable factor (CF),
7) cubes with suitable decomposition (CD),
8) cubes with suitable factor and decomposition (CFD).

Each dataset is subsequently partitioned into a training set (70%), a validation set (15%), and a testing set (15%).

D. Symbolic Regression Approach

The first approach used to predict the execution time of the distributed k-Wave simulation is based on symbolic regression. Symbolic Regression (SR) [19] is a type of regression analysis

[20] that aims to find a mathematical expression or formula that best describes the relationship between variables in a given dataset. In our scenario, we are investigating the correlation between the size of the simulation domain, its factorization, decomposition, and the mapping onto computing nodes concerning the execution time of the simulation.

Unlike traditional regression methods used in our previous research that typically use predefined functional forms (e.g., linear, quadratic), the SR algorithm explores a space of mathematical expressions, combining mathematical operations and functions, to create a formula that best fits the observed data. The goal is to find a symbolic expression that not only accurately represents the training data but also generalizes well to new, unseen data. Wilstrup et al. demonstrated that symbolic regression is well-suited for solving problems with limited datasets, which is particularly crucial in our case [21].

For training SR models, we employed HeuristicLab, a framework for heuristic and evolutionary algorithms developed by members of the Heuristic and Evolutionary Algorithms Laboratory since 2002 [22]. HeuristicLab offers a range of methods based on genetic programming [23], including various recombination operators and search space strategies.

The training process utilized the training dataset discussed in Section II-C. For validation and comparison purposes with other techniques, the validation testing set was employed to assess the algorithm's performance on previously unseen data.

Given the time-intensive nature of the training procedure, a clean virtual machine running Windows 11 and HeuristicLab, utilizing the QEMU virtualization toolkit, was deployed on the Barbora cluster. This setup allowed for the simultaneous execution of multiple instances, each running on a 36-core node with 192 GB of memory. Typically, training a single parameter configuration required between 10 and 20 hours.

Each training run was constrained to a maximum of 1 million generations, with each generation consisting of 100 individuals and the preservation of the best-performing individual (elitism). The regression tree's maximum height was limited to 16 and the size of the tree to 42 nodes. The mutation probability was set at 10%. All other parameters, including the optimization algorithm and tree interpreter, were maintained at their default settings. After a comprehensive evaluation, the mean square error function was adopted as the error (fitness) function. An example of regression tree is depicted in Fig. 3.

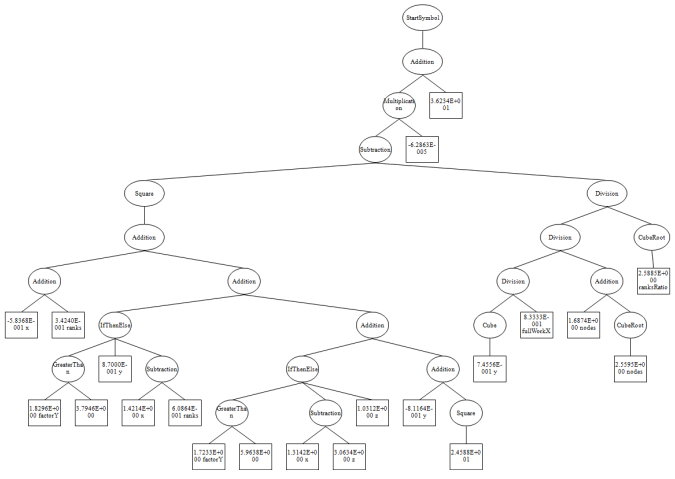

Fig. 3. Illustration of the best symbolic regression model for the dataset including only cubes with suitable factor and decomposition.

While HeuristicLab provides numerous mathematical operators, we extended its capabilities by developing a new plugin that introduces operators specifically tailored to our task. These operators encompass modulo operations, rounding up, down, and to the nearest integer. The final set of operators to be used in regression tree nodes includes:

- arithmetic operations (+, -, *, ÷),
- modulo,
- rounding (up, down, to the nearest integer),
- exponential function,
- logarithm,
- constant,
- input variable multiplied by a constant,
- power functions (square, cube, square root, cube root, general power, general root),
- conditional *if-then-else* statement,
- comparison ($>$, $<$),
- logical operators (and, or, not, xor).

E. Artificial Neural Network Approach

The second approach is based on Artificial Neural Network (ANN) [24]. The Tensorflow library was used to create and train ANNs in the Google Colab environment allowing to execute calculation on hardware accelerators.

The structure of the ANN is show in Listing 1. The data entering the ANN is first normalized using the Normalization layer, which adjusts the data to have a mean of 0 and a standard deviation of 1. The first layer of the ANN consists of 12 neurons according to the features defined in Sec. II-B. After that, a fully connected layer with the ReLU activation function and a Dropout layer, which is active only during training, is repeated 6 times. At the end, there is a final fully connected layer with one neuron, whose value represents the predicted output. This specific architecture was kept fixed after extensive experimentation involving hyperparameter adjustments and various network architectures. The mean square error function was again adopted as the error function.

The training utilized the Adam optimizer [25] with a step size parameter of 10^{-4}. To prevent overtraining and limit training time, the early stopping technique was employed to terminate the model training if the error on the validation dataset remained unchanged for the last 500 iterations. However, the inclusion of Dropout layers extended the required number of epochs. Each of the 8 filtered datasets was used to train the ANN, and the training process was halted as soon as the validation loss remained unchanged for 500 epochs. Fig. 4 depicts the training and validation error trends during training on all the data without any filtering.

```
model = tf.keras.Sequential([
    normalizer,
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(1),
])
```

Listing 1. Architecture of the Artificial Neural Network.

III. Experimental Results

In this chapter, we examine the accuracy of the developed models by comparing the real and predicted execution times. Initially, we discuss the general accuracy, highlighting how it varies with the applied data filters. Subsequently, we delve into two representative cases, detailing the model accuracy across training, validation, and testing datasets. Lastly, we present graphs comparing real and predicted scaling for domain sizes in the testing set (which were not previously exposed to the models), specifically tailored for the Barbora supercomputer.

A. Prediction Accuracy and Data Filtration Dependence

Table I concisely presents the average relative errors for both models evaluated across various datasets, filtered as outlined in Sec. II-C. These average errors were computed across all domain sizes and over the computing node range from 1 to 32. For clarity and conciseness, only the final model's error box plot is shown in Fig. 9. Additionally, Fig. 5 provides a succinct representation of the errors encountered in the testing datasets.

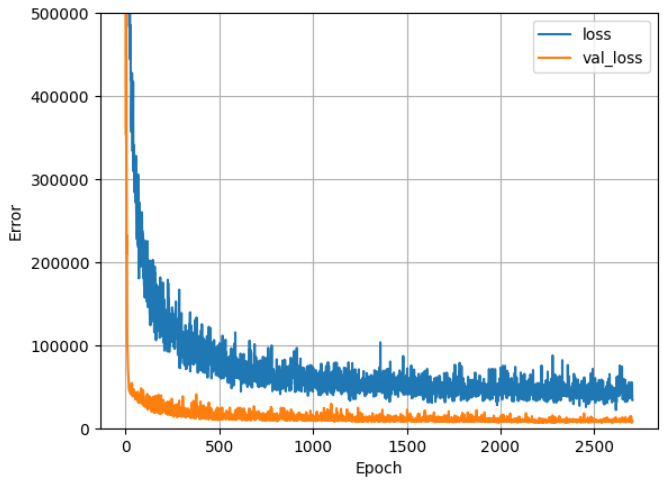

Fig. 4. Training and validation error of the ANN on the complete dataset without any filtering.

In the context of both validation and testing datasets, SR has shown enhanced precision in execution time estimates, especially for datasets with domain sizes amenable to effective factorization into small prime numbers. It also tends to yield superior outcomes for cubic domain configurations.

Conversely, ANN demonstrates a notable proficiency in scenarios where domain sizes include large prime factors, a situation where the FFT library's efficiency markedly reduces, leading to less predictable execution times. Interestingly, excluding instances of inefficient decompositions did not substantially improve the accuracy for any of the models. This outcome is possibly attributed to the robustness of the load imbalance feature, which seems to adeptly capture and represent this particular complexity.

B. Prediction Accuracy for Selected Cases

This section provides an in-depth analysis of specific datasets. Table II presents the average prediction errors for domain sizes with suitably small prime factors (up to 7). This subset is crucial since representing typical use cases.

Both models can be trained with a reasonably small error on the training dataset. The error produced by SR is around 5% with only a few outliers, namely the domain size of 512^3 where the execution time is significantly overshot. ANN is also able to capture the scaling of training dataset reasonably well. The validation set, however, exhibits significantly higher error, especially for ANN. Better results were achieved for the testing dataset, which clearly show the SR is better for these domain sizes.

Fig. 6 shows the real and predicted execution times for cubic domain of 1024^3 grid points. The scaling is precisely predicted by the SR with the average relative error of 4.78%.

TABLE I
The average relative prediction error observed in specific performance datasets over all domain sizes.

Filter Type	Dataset	SR	ANN
1. No filter	Training	13.70%	9.35%
	Validation	9.31%	10.33%
	Testing	13.55%	8.25%
2. Cubes	Training	7.78%	7.79%
	Validation	9.90%	34.42%
	Testing	9.97%	17.50%
3. Suitable Factor	Training	5.58%	5.75%
	Validation	11.26%	21.71%
	Testing	5.64%	12.46%
4. Suitable Decomposition	Training	18.62%	6.32%
	Validation	22.51%	6.49%
	Testing	15.16%	11.84%
5. Suitable Factor and Decomposition	Training	7.27%	7.50%
	Validation	14.01%	15.45%
	Testing	7.48%	19.38%
6. Cubes with Suitable Factor	Training	10.25%	5.99%
	Validation	11.65%	19.11%
	Testing	8.85%	21.16%
7. Cubes with Suitable Decomposition	Training	14.53%	6.83%
	Validation	16.38%	13.90%
	Testing	17.77%	7.83%
8. Cubes with Suitable Factor and Decomposition	Training	8.93%	7.11%
	Validation	7.94%	27.72%
	Testing	5.68%	23.25%

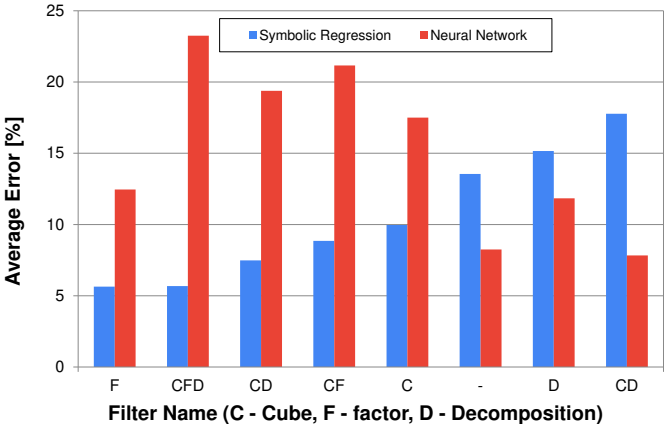

Fig. 5. Visualization of the average relative errors in trained models across datasets subjected to various filters. The datasets are arranged in order of increasing error magnitude as determined by the symbolic regression model.

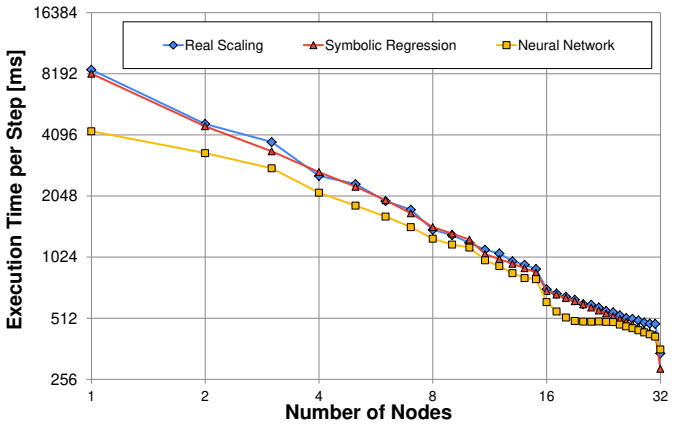

Fig. 6. Analysis of real vs. predicted execution times for a domain size of 1024^3 and 1-32 computing nodes utilizing SR and ANN models trained on a filtered dataset with suitable factors. Domain taken from the testing dataset.

All steps on the original curve are very well captured. The execution time is slightly overestimated, which is the better case, since execution run is not terminated prematurely. In contrast, the ANN's predictions for the scaling curve are considerably less accurate (error of 14.4%), characterized by a substantial underestimation of execution time and a slight delay in identifying the step points.

A similar pattern is evident in the cuboid simulation domain, see Fig. 7. SR here attains an error rate of 8.14%, leading to a more pronounced overestimation for 8 to 10 nodes, and the creation of two spurious steps in the scaling curve, absent in the original. This discrepancy is likely due to the significant size difference between the Ny and Nz sizes. While the ANN model successfully approximates the general trend of the curve, it underestimates the values. Notably, it fails to detect the step at 10 nodes and introduces an anomalous fluctuation at 24 nodes, resulting in elevated errors for larger node counts. In total, the average prediction error reaches 13.05%.

Table III shows the prediction error for the complete performance dataset, which includes cases with substantial prime factors of 13 and 23. Analysis of the testing datasets indicates that the ANN yields more uniform predictions, especially in scenarios involving higher prime factors. In scenarios where the largest factor is 23, the SR model demonstrates a tendency to underestimate execution time, with a significant margin of approximately 30.49%, which represents a critical issue in terms of accuracy and reliability. The data also suggests that the performance dataset might not be sufficiently extensive to train both models for precise scaling predictions. However, as previously noted, compiling this dataset is resource-intensive.

Fig. 8 shows the comparison of the real and predicted scaling on a domain of 736^3 grid points. It is immediately evident how much SR underestimates the execution time. However,

TABLE II
THE RELATIVE ERROR OF BOTH MODELS ON SELECTED DOMAIN SIZES WITH SUITABLE FACTORS. THE TRAINING, VALIDATION AND TESTING DATASETS HAD 24, 4 AND 4 DIFFERENT SIMULATION SIZES.

Training dataset			
Domain size	Factors	SR Error	ANN Error
$256 \times 256 \times 1536$	{2}, {2}, {2,3}	5.19%	12.13%
$256 \times 256 \times 2048$	{2}, {2}, {2}	8.66%	12.56%
$432 \times 432 \times 1728$	{2,3}, {2,3}, {2,3}	4.33%	6.87%
512^3	{2}	14.88%	7.29%
756^3	{2,3,7}	4.69%	5.13%
768^3	{2,3}	3.41%	3.42%
810^3	{2,3,5}	2.51%	7.16%
960^3	{2,3,5}	5.34%	4.22%
1280^3	{2,5}	6.21%	2.48%
2048^3	{2}	3.10%	4.00%
Validation dataset			
$256 \times 256 \times 1024$	{2}, {2}, {2}	13.48%	50.41%
$384 \times 384 \times 1024$	{2,3}, {2,3}, {2}	15.59%	32.13%
$512 \times 512 \times 1024$	{2}, {2}, {2}	11.53%	30.37%
675^3	{3,5}	10.12%	13.70%
Testing dataset			
$384 \times 384 \times 1536$	{2,3}, {2,3}, {2,3}	8.14%	13.05%
576^3	{2,3}	7.99%	15.24%
896^3	{2,7}	3.17%	6.12%
1024^3	{2}	4.78%	14.40%

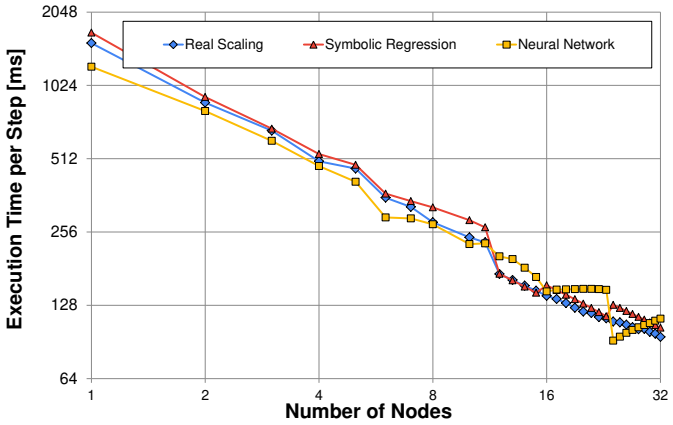

Fig. 7. Analysis of real vs. predicted execution times for a domain size of $384 \times 384 \times 1536$ and 1-32 computing nodes utilizing SR and ANN models trained on a filtered dataset with suitable factors. Domain taken from the testing dataset.

TABLE III
THE RELATIVE ERROR OF BOTH MODELS ON A UNFILTERED DATASET. THE TRAINING, VALIDATION AND TESTING DATASETS HAD 29, 5 AND 5 DIFFERENT SIMULATION SIZES.

Training dataset			
Domain size	Factors	SR Error	ANN Error
$256 \times 256 \times 2048$	{2}, {2}, {2}	19.23%	22.48%
$384 \times 384 \times 1024$	{2,3}, {2,3}, {2}	23.37%	16.69%
$384 \times 256 \times 1536$	{2,3}, {2,3}, {2,3}	8.11%	9.74%
648^3	{2,3}	16.14%	7.22%
650^3	{2,5,13}	10.46%	12.70%
756^3	{2,3,7}	11.99%	7.67%
800^3	{2,5}	5.96%	7.94%
832^3	{2,13}	7.33%	8.73%
1024^3	{2}	5.79%	2.91%
2048^3	{2}	4.24%	3.96%
Validation dataset			
$384 \times 512 \times 1536$	{2,3}, {2}, {2,3}	6.70%	7.08%
$512 \times 512 \times 1536$	{2}, {2}, {2}	11.91%	7.21%
$512 \times 512 \times 2048$	{2}, {2}, {2}	14.91%	11.96%
675^3	{3,5}	7.05%	8.59%
704^3	{3,11}	11.80%	15.48%
Testing dataset			
640^3	{3,5}	17.92%	10.82%
736^3	{3,23}	30.49%	5.42%
810^3	{2,3,5}	4.22%	6.34%
840^3	{2,3,5,7}	4.89%	6.09%
1536^3	{2}	8.45%	8.49%

the shape of the curve is captured very well. ANN almost perfectly matches the scaling for the number of computing nodes higher than 4.

C. Summary and Discussion

The presented results show that both models complement each other. This necessitated the development of a composite model, leveraging both algorithms for specific scenarios. The accuracy of this model, as illustrated in the box plot of Fig. 9, demonstrates notable efficiency. For domains with suitable factors, the error margin is kept below 5.64%, while for more general cases, the error does not exceed 8.25%.

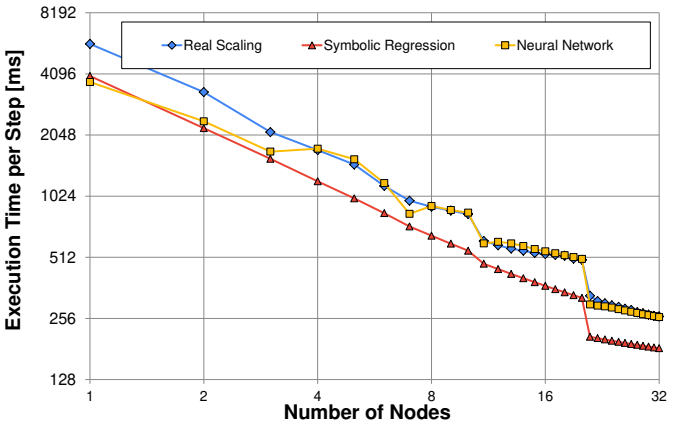

Fig. 8. Analysis of real vs. predicted execution times for a domain size of 736^3 (factor of 23) and 1-32 computing nodes utilizing SR and ANN models trained on the unfiltered dataset. Domain taken from the testing dataset.

This methodology marks a substantial advancement over our prior model, which relied on linear and quadratic interpolation techniques. In the case of cubic domains with small prime factors, linear interpolation demonstrated promising results, achieving errors as low as 4% on the training dataset. However, its performance diminished considerably on the validation and testing datasets, with errors escalating to nearly 25%. Conversely, quadratic interpolation managed to attain a more consistent error rate of 10.5%. Despite these improvements, it's important to note that both techniques exhibited significantly higher error rates, deemed unacceptable, in scenarios involving domains with large prime factors.

A comparable methodology was employed in [26], concentrated on estimating the execution time of data processing tasks within HPC systems. Mirroring our approach, Bielecki's study involved compiling a performance database using typical input data and then applying polynomial regression, as well as support vector and k-nearest neighbor regression techniques. Despite conducting these tasks on a single node, Bielecki reported average error rates ranging from 34% to 40%

In a related study, Phinjaroenphan [27] extended the scope to include executions on multiple nodes. Focusing on the execution time prediction for a matrix-matrix multiplication task, the author utilized the k-nearest neighbors algorithm and reported an estimation error of approximately 25%.

A seminal work on execution time prediction was presented in [28]. This publication corroborates that the technique outlined in our paper aligns well with current state-of-the-art methodologies, yielding estimations that are within the measurement uncertainty range typical for HPC systems, approximately around 10%.

IV. CONCLUSIONS

This research has been pivotal in advancing the prediction of execution time for distributed ultrasound simulations, with a primary focus on the interplay between simulation size and resource allocation. Through the establishment of a performance database, two machine learning models were trained. The Symbolic Regression model exhibited commendable accuracy, particularly in scenarios optimized for the k-Wave toolbox and adhering to its internal domain size factorization policy, achieving an average error of 5.64%. In more generalized situations, where domain sizes deviate from this policy, the trained Neural Network demonstrated acceptable predictive capability with an error margin of 8.25%.

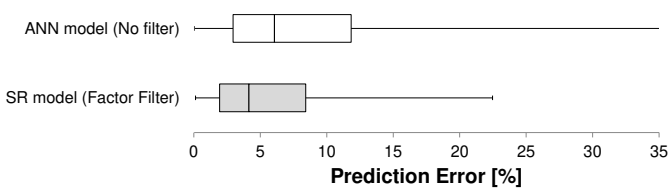

Fig. 9. Box plot summarizing the prediction error over every case from a testing dataset.

When comparing these results with our prior execution time estimation technique, which incorporated linear and quadratic splines, the estimation accuracy has improved from 25% and 10.5% respectively [29]. This enhancement represents a significant improvement, contributing to more effective job placement and reduced queueing times.

In summary, the outcomes of this study provide a solid foundation for future advancements in execution time prediction, with potential applications extending beyond the realm of ultrasound simulations. A notable strength of this approach is its adaptability. The Neural Network is preferred for blind predictions, while the Symbolic Regression model is effective when specific features, such as domain size factorization, node occupancy, and load imbalance, influence execution time.

Looking ahead, the development and implementation of this prediction system are prepared to take two significant directions. The first objective is to incorporate this prediction system into the k-Dispatch workflow management system [12]. This integration is anticipated to enable a fully automated optimization process for ultrasound workflows, significantly enhancing efficiency and resource management. Post-integration, the performance database will be enriched in real-time with data from successfully completed tasks. This ongoing data collection will facilitate continual refinement of the models, providing a more robust and comprehensive training dataset. Moreover, this adaptive model will extend its applicability to a variety of simulation codes and computing systems, broadening the scope and impact of this research.

V. Acknowledgment

This project has received funding from the European Unions Horizon Europe research and innovation programme under grant agreement No 101071008. This work was also supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90254), and by Brno University of Technology under project FIT-S-23-8141.

References

[1] Y.-F. Zhou, A. Syed Arbab, and R. X. Xu, "High intensity focused ultrasound in clinical tumor ablation." *World journal of clinical oncology*, vol. 2, no. 1, pp. 8–27, 2011.

[2] B. E. Treeby and B. T. Cox, "K-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave-fields," *Journal of Biomedical Optics*, vol. 15, no. 2, p. 21314, 2010.

[3] J. Jaros, a. P. Rendell, and B. E. Treeby, "Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound," *International Journal of High Performance Computing Applications*, vol. 30, no. 2, pp. 137–155, 2015.

[4] B. Treeby, F. Vaverka, and J. Jaros, "Performance and accuracy analysis of nonlinear k-Wave simulations using local domain decomposition with an 8-GPU server," in *Proceedings of Meetings on Acoustics*, vol. 34, no. 1, 2018, p. 022002.

[5] S. Manohar and M. Dantuma, "Current and future trends in photoacoustic breast imaging," *Photoacoustics*, vol. 16, p. 100134, dec 2019.

[6] L. Mohammadi, H. Behnam, J. Tavakkoli, and M. R. Avanaki, "Skull's photoacoustic attenuation and dispersion modeling with deterministic ray-tracing: Towards real-time aberration correction," *Sensors (Switzerland)*, 2019.

[7] A. Magda Abbas, C. Constatin-Coussios, and O. Robin Cleveland, "Patient Specific Simulation of HIFU Kidney Tumour Ablation," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual Conference*, vol. 2018, 2018, pp. 5709–5712.

[8] A. Grisey, S. Yon, V. Letort, and P. Lafitte, "Simulation of high-intensity focused ultrasound lesions in presence of boiling," *Journal of Therapeutic Ultrasound*, 2016.

[9] V. Suomi, B. Treeby, J. Jaros, P. Makela, M. Anttinen, J. Saunavaara, T. Sainio, A. Kiviniemi, and R. Blanco, "Transurethral ultrasound therapy of the prostate in the presence of calcifications: A simulation study," *Medical Physics*, vol. 45, no. 11, pp. 4793–4805, nov 2018.

[10] A. Pouliopoulos, S.-Y. Wu, M. Burgess, M. Karakatsani, H. Kamimura, and E. Konofagou, "A Clinical System for Non-invasive Blood-Brain Barrier Opening Using a Neuronavigation-Guided Single-Element Focused Ultrasound Transducer," *Ultrasound in Medicine and Biology*, vol. 46, 2019.

[11] V. Chaplin, M. Phipps, and C. Caskey, "A random phased-array for mr-guided transcranial ultrasound neuromodulation in non-human primates," *Physics in Medicine and Biology*, vol. 10, p. 105016, 12 2017.

[12] M. Jaros, B. E. Treeby, P. Georgiou, and J. Jaros, "k-dispatch: A workflow management system for the automated execution of biomedical ultrasound simulations on remote computing resources," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, ser. PASC '20. New York, NY, USA: ACM, 2020.

[13] E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. F. da Silva, G. Papadimitriou, and M. Livny, "The evolution of the pegasus workflow management software," *Computing in Science & Engineering*, vol. 21, no. 4, pp. 22–36, 2019.

[14] F. A. Omara and M. M. Arafa, "Genetic algorithms for task scheduling problem," *Journal of Parallel and Distributed Computing*, vol. 70, no. 1, pp. 13–22, 2010.

[15] H. Izadkhah and Y. Li, "Learning based genetic algorithm for task graph scheduling," *Appl. Comp. Intell. Soft Comput.*, vol. 2019, jan 2019.

[16] P.-F. Dutot, M. A. S. Netto, A. Goldman, and F. Kon, "Scheduling Moldable BSP Tasks," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 157–172.

[17] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," *Proceedings of the April 1820 1967 spring joint computer conference*, vol. 23, no. 4, pp. 483–485, 1967.

[18] M. Frigo and S. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, feb 2005.

[19] J. J. Schnur and N. V. Chawla, "Information fusion via symbolic regression: A tutorial in the context of human health," *Information Fusion*, vol. 92, pp. 326–335, apr 2023.

[20] N. R. Draper and H. Smith, *Applied Regression Analysis*, ser. Wiley Series in Probability and Statistics. Wiley, apr 1998.

[21] C. Wilstrup and J. Kasak, "Symbolic regression outperforms other models for small data sets," *CoRR*, vol. abs/2103.15147, 2021.

[22] S. Wagner, G. Kronberger, A. Beham, M. Kommenda, A. Scheibenpflug, E. Pitzer, S. Vonolfen, M. Kofler, S. Winkler, V. Dorfer, and M. Affenzeller, *Advanced Methods and Applications in Computational Intelligence*, ser. Topics in Intelligent Engineering and Informatics. Springer, 2014, vol. 6, ch. Architecture and Design of the HeuristicLab Optimization Environment, pp. 197–261.

[23] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[24] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[25] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.

[26] J. Bielecki and M. Śmiałek, "Estimation of execution time for computing tasks," *Cluster Computing*, vol. 26, no. 6, pp. 3943–3956, dec 2023.

[27] P. Phinjaroenphan, S. Bevinakoppa, and P. Zeephongsekul, "A Method for Estimating the Execution Time of a Parallel Task on a Grid Node," in *Lecture Notes in Computer Science*, 2005, pp. 226–236.

[28] J. Flores-Contreras, H. A. Duran-Limon, A. Chavoya, and S. H. Almanza-Ruiz, "Performance prediction of parallel applications: a systematic literature review," *The Journal of Supercomputing*, vol. 77, no. 4, pp. 4014–4055, apr 2021.

[29] M. Jaros and J. Jaros, "Optimization of execution parameters of moldable ultrasound workflows under incomplete performance data," in *Job Scheduling Strategies for Parallel Processing. JSSPP 2022*, ser. Lecture Notes in Computer Science, LNCS 13592, vol. 13592. Springer Nature Switzerland AG, 2023, pp. 152–171.