Grey-Box Machine Learning Prediction of Parallel Application Scaling

Anonymous Author(s)

ABSTRACT

Amdahl's law and similar performance laws are inexpensive and effective tools for estimating application speedup with changes in numbers of processors and workload. However, their accuracy can be poor on realistic systems and workloads where it is necessary to estimate key model parameters from limited data sets and to address realistic system overheads such as latency and network congestion. This paper introduces a novel and inexpensive method to predict the overhead incurred by an application based on the problem size and resource configuration. This method uses simple machine-learning methods to estimate the overhead of an application for a given problem size and resource configuration as a coefficient of the estimated speedup provided by performance laws. We evaluate this approach on two HPC mini-applications and two full applications with varying patterns of computation and communication. In addition, we also evaluate the prediction accuracy of our models on runs with varying processors-per-node configurations. Our results demonstrate that this method can significantly improve the prediction accuracy of standard performance laws, and that performance prediction accuracy is relatively insensitive to the size of the training dataset.

ACM Reference Format:

Anonymous Author(s). 2024. Grey-Box Machine Learning Prediction of Parallel Application Scaling . In *Proceedings of 31st IEEE International Conference on High Performance Computing, Data, Analytics (HiPC'24).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

High performance computing (HPC) systems and users need accurate performance prediction to improve overall system resource utilization. Unfortunately, existing performance prediction methods are generally prohibitively expensive, or work only in very narrow areas [3] [28] [13] [25] [23]. As a result, users do not understand how long their applications take to run, and often overestimate the wall time information that they provide to scheduling systems, resulting in poor system utilization [21] [9] [20]. This can also limit the effectiveness of next-generation intelligent job schedulers currently in development [1].

In this paper, we show how to use machine-learning models to improve the predictive capabilities of well-known performance laws and models. Specifically, integrating machine learning models trained on easy-to-acquire data with an enhanced version of Amdahl's Law yields performance models that are accurate on modern applications and systems. In addition, the resulting machine learning-enhanced predictions are simple to understand because they are tied to well-known performance laws.

This paper makes the following specific contributions:

- We integrate machine learning models into a generalization of strong-scaling and weak-scaling laws based on Amdahl's law. This integration enables accurate prediction of application runtimes even when accurate measurements for percentage parallel (p) are not available.
- We evaluate the accuracy of this extended model in predicting the strong and hybrid scaling behavior of a selection of HPC mini-applications and full applications on modern hardware systems.
- We assess the predictive ability of different machine learning methods, including decision trees, random forests, and neural networks. We include a comparative accuracy study using these different methods and their sensitivity to changes in the size of the training set.
- We demonstrate that this model outperforms Amdahl's law performance predictions when provided the same training set to estimate the overhead and model performance parameters. Our approach can be competitive with Amdahl's law predictions even if Amdahl's law is provided with optimal model parameters not available to the machine learning model.
- We compare the performance of our model in predicting strong-scaling speedups to that of the current state-of-the-art model [17] with similar data requirements to that of ours.

The remainder of this paper is structured as follows. First Section 2 presents our general modeling and prediction approach based on a generalized version of Amdahl's law with additional parameters to quantify overheads not typically captured by Amdahl's law. Section 3 then describes our approach to estimating these overhead functions both in terms of the overhead formulation used and the machine learning approach chosen. Following this, Sections 4 and 5 present the experimental methodology we used to evaluate the effectiveness of this approach to predicting application scaling performance and the results of this evaluation. Finally, Sections 6 and 7 describe related work, directions for future work, and conclude.

2 MODELING APPROACH

Our overall approach corrects Amdahl's law with a learned function. This correction accounts for the additional overhead in real benchmarks and applications that standard performance laws do not account for, and is robust to inaccuracies resulting from model parameter estimation challenges. By combining an easily-understood

HiPC'24, December 18-21 2024, Bengaluru, India

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

performance law with machine learning approaches, we seek to 1) reduce the amount of data required to train a model to predict applications' performance, and 2) make the learned function understandable and informative to application designers and system analysts. In the remainder of this section, we describe the general formulation of our model and its relation to previous performance law-based performance prediction approaches.

2.1 Model Structure

Recall that Amdahl's Law gives the theoretical maximum speedup of a program from parallelism in the following form when percentage parallel p and the number of processors N are known:

$$S(N,p) = \frac{1}{1-p+\frac{p}{N}} \tag{1}$$

To support grey-box modeling of application performance speedup, we formulate performance speedup laws of the general form $Speedup = S(N,w,p)$ where w is the ratio of the original problem size to that of the new problem size, N and p are the number of processors, and the percentage parallel respectively.

For the performance law proposed in this paper, we modify Amdahl's law to generalize it to strong-scaling, weak-scaling, and hybrid-scaling problems. In particular, we introduce the application problem size scaling parameter w, which represents the change in the problem size of an application with respect to the original problem size, to study the use of a single performance law to estimate and study the application overheads.

This results in the following performance law:

$$S(N,w,p) = \frac{w}{1-p+\frac{p}{N}} \tag{2}$$

When the original problem size ($m_{original}$) and scaled problem size (m_{scaled}) are known, w can be computed as $w = \frac{m_{original}}{m_{scaled}}$. In the case of strong scaling ($m_{original} = m_{scaled}$), if the number of processors $N = 2$, then $w = 1$, while in the case of weak scaling ($m_{scaled} = m_{original} * N$), if $N = 2$, $w = \frac{1}{N} = 0.5$. Note that weak-scaling speedups in this performance law are not equivalent to that of Gustafson's law [10]. Gustafson's law assumes that the serial portions of the applications remain constant when the problem sizes grow with an increase in parallelism. In contrast, the performance law presented above assumes that the serial portion of the code grows proportionally with an increase in problem size.

Our above-described formulation however does not account for overhead from parallelization in real systems. To tackle this problem, we estimate p as a constant for a given application/system and introduce a separate multiplicative correction function τ, which learns the expected overhead for specific parameters. Then, the corrected speedup is:

$$Speedup' = S(N,w,p) * \tau(N,w,p) \tag{3}$$

Note that on multi-core systems, N is the product of the number of nodes and the number of processors per node, and that our approach chooses only a single percentage parallelism parameter p based on the total number of processes executing. That is, the Amdahl's law prediction on which we base our learning approach does not distinguish between intra- and inter-node parallelism. However, it is important to note that the learned overhead function *can* take into account different overhead amounts based on the source of parallelism, as the number of nodes and processors-per-node used in a given run are used for estimating this overhead function, as described in Section 2.2.

2.2 Parameter Estimation

Our approach relies on the estimation of the percentage parallel parameter, p, prior to learning the overhead function.

We use a straightforward method shown in Equation 4.

$$p = \underset{p_j}{\operatorname{argmin}} \left(\Sigma_i (S_i - S(N_i, w_i, p_j))^2 \right) \quad \{p_j \in \mathbb{R} \mid 0 \le p_j \le 1\}, i \in D \tag{4}$$

We collect empirical timing data (D) for a given application/system pair for configurations with a cutoff for a maximum processor count of c_f, and a cutoff for a maximum problem size of w_f. We then create a dataset of speedups (S_i) by isolating runs with single processor counts ($N_i = 1$), and by computing the relative speedup for runs of the same problem size (w_i) when the number of processors is increased ($1 < N_i \le c_f$), where i represents an arbitrary data point in the collected timing sample. From this dataset, we estimate the optimal value of p by regressing to a value (p_j) that results in the least mean-squared error for estimated speedups ($S(N_i, w_i, p_j)$) when plugged into equation 2.

The training dataset D_t is derived from the dataset of runtimes by setting cutoff values for the problem sizes and processor counts - w_f and c_f respectively. These values are explained in more detail in section 3.

Estimating p for most applications is not trivial, as it can vary greatly based on their problem size, resource configuration, and limitations of the compute/communication hardware. Factors such as network latency, memory-bus/network bandwidth limitations, and thread-creation overheads can reduce the observed value of p, leading to greatly overestimated speedups at large processor counts. In addition, p for the same application can drastically vary between different systems due to changes in hardware capabilities and will need to be recomputed when the hardware is changed.

In this work, we provide an inexpensive way to perform this calculation empirically, via machine learning, with support for changing problem sizes. Finally, we note that our formulation is similar to that of Höfinger et. al [12], who represent overhead as an analytically calculated multiplicative correction function, and restrict their formulation to a fixed problem size. Their model however only classifies time spent in MPI calls as overhead, and relies on MPI profiling information to compute the ground truth values for overhead. On the other hand, our model relies only on timing information from application runs and computes overhead as any deviation from the expected Amdahl's Law speedup predictions. In addition, their formulation does not account for changes in problem sizes, while our model predicts performance as a function of both - change in the number of processors and change in problem size.

The machine-learning formulation that we use to learn overhead is described in detail in Section 3.

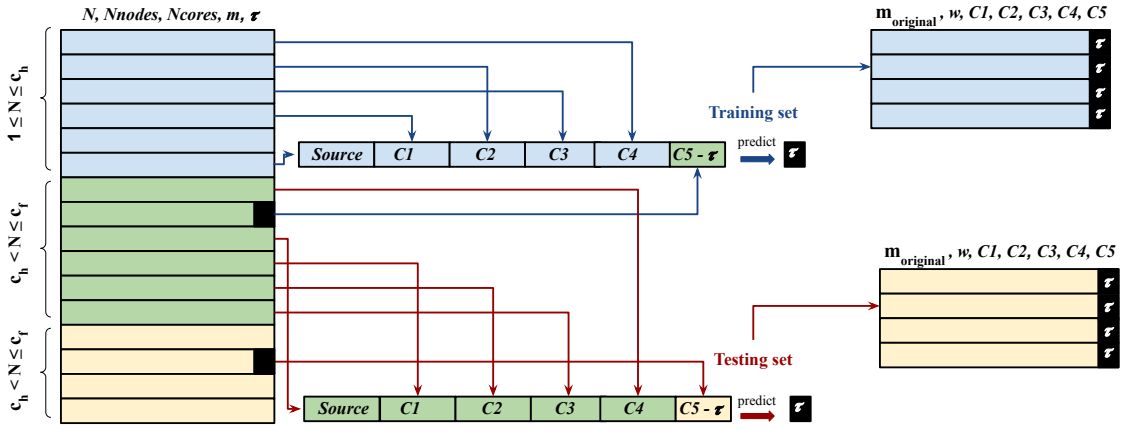


Figure 1: Visualization of training and testing dataset construction from timing information for a given application/system pair.

3 MACHINE LEARNING FORMULATION

Given the modeling in Section 2, our goal is to use machine learning to learn the overhead function given a set of input points with varying problem sizes and number of processes across various levels of parallelism. We hypothesize that the overhead for an application with a given hardware configuration is an application and system-dependent function of its problem size, total number of processors, and processor configuration, with the following general formulation for two levels of parallelism - N_{nodes}, and N_{cores}.

$$\tau(N, N_{nodes}, N_{cores}, w, p) = \frac{T_{estimated}(N, w, p)}{T_{observed}(N, w)} \quad \{N = N_{nodes} * N_{cores}\} \tag{5}$$

$$T_{estimated}(N, w, p) = \frac{T_{observed}(1, 1)}{S(N, w, p)} \tag{6}$$

where N is the total number of processors, N_{nodes} is the number of nodes, N_{cores} is the number of processors in each node. $T_{estimated}(N, w, p)$ is the computed execution time based on equation 2's predicted speedup. $T_{observed}(1, 1)$ is the program execution time on a single process with the original problem size, and $T_{observed}(N, w)$ is the observed execution time with a scaled number of processors (N) and problem size (w). Note that we represent a GPU as a single processor.

Then, to ensure the validity of our results, we use data collected from a small number of processes and problem sizes to train a machine-learning model to predict the overhead $\tau(N, N_0, N_1, w, p)$ of runs with larger number of processes and problem sizes. For example, we might collect run time data for an application using 50 to 500 processes for problem sizes 100 - 1000; c_f is a cutoff threshold to split the data into training (e.g., runs using a number of processes less than or equal to c_f) and testing (e.g., runs using a number of processes larger than c_f); similarly, w_f is a cutoff threshold used to the split the data based on problem sizes. In Section 5, we show the impact in accuracy of varying c_f and w_f's values.

To train this model, we construct training inputs by randomly selecting sets of 5 samples of scaled application overheads, paired with a serial run of problem size $m_{original}$. Each of the 5 scaled samples has the same problem size m, and contains the following values C: 1) N - Total number of processes, 2) N_{nodes} - Number of nodes, 3) N_{cores} - Number of processors per node, 4) m - Problem size, 5) τ - Observed overhead. These assembled samples are then further reduced to vectors with the following features: 1) $m_{original}$ - The problem size of the original serial application run, 2) w - ratio of the original problem size to the scaled problem sizes (common value for all 5 samples), 3) $C1$ - C_5 - as described previously, where C_5's τ is the target attribute. Figure 1 provides a visualization of the dataset assembly.

In each training input, 4 samples have $N \leq c_h$, for $c_h = \frac{c_f}{2}$ and the 5^{th} sample has $c_h < N \leq c_f$. τ is computed as described in equation 5 using equation 4 to estimate p for *all* elements of the dataset that satisfies the training set restrictions, i.e. $N \leq c_f$. That is, the resulting sample is of the following format:

$$D_i = [m_{original}, w, C_1, C_2, ..., C_5] \tag{7}$$

$$D = [D_1, D_2, ..., D_n] \tag{8}$$

with the additional constraints to D_i in our training set:

- $1 < N_{i,j} \leq c_h \forall (1 \leq j \leq 4)$
- $c_h < N_{i,5} \leq c_f$

We set $\tau_{i,5}$ (where $N > c_h$) as the target attribute, and the below-described models learn to predict $\tau_{i,5}$ for a given application run when provided with a data point $D_{i,X} = [C_1, C_2, .., (C_5 - \tau_5)]$

The machine learning models that we considered are: Random Forest Regression [11], Gradient Boosted Trees [29] and a simple MLP Neural Network [19]. We decided to rely only on classical ML models because they require significantly less training data than architectures like deep neural networks or transformers (i.e., several orders of magnitude less), and they have shown to outperform such architectures when dealing with tabular data [8].

4 EXPERIMENTAL METHODOLOGY

To evaluate the efficacy of our approach to predicting scaling speedups and overheads of applications in modern systems, we collected timing information for a number of HPC applications on a range of problem sizes and process configurations on both modern CPU and GPU systems. We start our evaluation by primarily focusing on single-level parallelism predictions in order to evaluate our methods with a simpler test case before adding in the additional complexity of multi-level parallelism.

4.1 Test Applications

Using the methodology described above, we examined the scaling behavior of the following applications chosen for their diverse computational approaches.

4.1.1 Beatnik. Beatnik is a benchmark based on an open-source implementation [6] implementation of Pandya and Shkoller's 3D fluid interface "Z-Model" [18] written in the Kokkos-based Cabana/Cajita performance portability framework [24]. For Beatnik runs, we ran a periodic multi-mode rocket rig test using a low-order solver with the initial interface distributed according to a cosine function. Beatnik supports both CPU and GPU execution, and can also utilize MPI to run on distributed memory systems. The Beatnik low-order solver includes both halo exchanges for surface calculations and fast fourier transforms for global force calculations, with global 2D FFTs from the using HeFFTe [4] library generally dominating performance. The problem sizes for Beatnik runs range between 250 and 8000, with a median size of 3500.

4.1.2 MiniFE. MiniFE is a proxy application for unstructured implicit finite element codes [15]. Its main computational loop is based on preconditioned conjugate gradient solves of a sparse linear system, and it supports parallel execution on CPUs, GPU, and over MPI. The problem sizes for MiniFE runs range between 50 and 330 with a median size of 165.

4.1.3 LAMMPS. The Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) is a classical molecular dynamics simulation application focusing on materials modeling [26]. It supports CPU, GPU, and distributed memory parallelism via MPI. The problem sizes for LAMMPS runs range between 10 and 150 with a median size of 90.

4.1.4 FIESTA. Fast InterfacES and Transport in the Atmosphere (FIESTA) is a highly-scalable finite-volume compressible flow solver [22] for simulating shock hydrodynamics problems. It can run serially on CPUs and in parallel on GPUs with support for MPI. The problem sizes for FIESTA runs range between 10 and 210 with a median size of 115.

4.2 Test System

We collected data running these applications on the following systems:

Machine	CPU/GPU	Interconnect
LLNL Quartz	Intel Xeon E5-2695	Cornelis Networks Omni-Path
LLNL Lassen	IBM Power 9 NVIDIA V100	IB EDR

Tests run on Quartz range from 1 to 500 nodes, while tests run on Lassen use between 1 and 50 nodes, though not all node counts were used. The median node count for Quartz runs was 150 while the median node count for Lassen runs was 10.

In addition, Quartz runs used between 1 and 32 CPUs per node, with a median count of 16 CPUs. Note that we use pure process-based parallelism for all of our runs. We treat on-node and off-node parallelism as two different levels. Lassen tests only used one NVIDIA V100 GPU per node; we have not yet attempted to predict performance with multiple processes or GPUs per node on Lassen. All of the Lassen application runs have a single level of parallelism.

4.3 Data Collection

The data collection process is straightforward and simply requires timing information from various run configurations for every application. These run configurations vary by application and machine, and consist of all possible permutations of a constant set of input sizes, number of nodes, and number of processors per node. We timed application runs by placing markers at the start and the end of their main function in order to eliminate startup noise from the job launcher. It was also ensured that all of the jobs were running in exclusive mode, and no two jobs were being co-scheduled to run on the same node at any given time.

4.4 Training and Testing Sets

We partition our performance data into training and testing sets by selecting appropriate constants c_f and w_f, where c_f is the cutoff for the maximum number of processors, and w_f is the cutoff for the maximum problem size in the training set. Note that in runs with multiple levels of parallelism, the processor count N is computed as the product of processor counts across the two levels of parallelism. Runs with problem size or number of processors lesser than or equal to the median were used in the training set, and the remaining were used in the testing set. On the Quartz system, this results in $c_f = 100$ for single-level parallelism runs, and $c_f = 500$ for multi-level parallelism runs, and on Lassen, this results in $c_f = 10$ unless otherwise specified.

5 RESULTS AND DISCUSSION

We evaluate the model described in the previous sections in multiple ways. First, we evaluate the efficacy of different machine learning techniques in using the approach described in this paper to predict runtime scaling of a range of applications. We then compare the efficacy of this approach to the standard approach based on determining the optimal parallel percentage for an application/system pair. We then examine how changes in the size of the training set impact performance prediction accuracy. In each of these cases, we limit the evaluation to a single process per node to reduce the complexity of the evaluation and focus on scaling accuracy and sensitivity on per-process basis. Following these evaluations, we then extend our evaluation to multi-level parallelism, where varying numbers of MPI processes are run on each node; this analysis again includes both scaling accuracy and sensitivity analyses. Furthermore, we compare the performance of our best-performing machine learning models to the current state-of-the-art scaling model with similar data requirements as that of our models [17].

We use Root Mean Squared Error (RMSE) as our evaluation metric for the models and also present their performance distributions using Root Squared Error (RSE). Smaller RMSE/RSE values indicate better performance and values lower than 0.5 generally indicate acceptable prediction accuracy.

5.1 Comparison Baseline

We use the enhanced Amdahl's law (Equation 2) as a baseline against which to compare the machine learning predictions. We estimate the p parameter for this equation using three approaches:

Aggregate p: Use p that results in the lowest mean-squared prediction error using Equation 2 in the entire training set per Equation 4.

Optimal p: Compute the p that results in the lowest mean-squared prediction error using Equation 2 for the specific problem size being tested, as shown in Equation 9. Note that this approach will compute p using data not in the training set for problem sizes greater than w_f.

Optimal p **FD**: Same as above, but ignore both cutoffs - c_f and w_f when computing p. This approach utilizes both, the training and testing datasets to estimate the optimal p for every problem size m.

$$p(m) = \underset{p_j}{\operatorname{argmin}} \left(\Sigma_i (y_i - S(N_i, w_i, p_j))^2 \right) \quad \{p_j \in \mathbb{R} \mid 0 \leq p_j \leq 1\}, i \in D_m \tag{9}$$

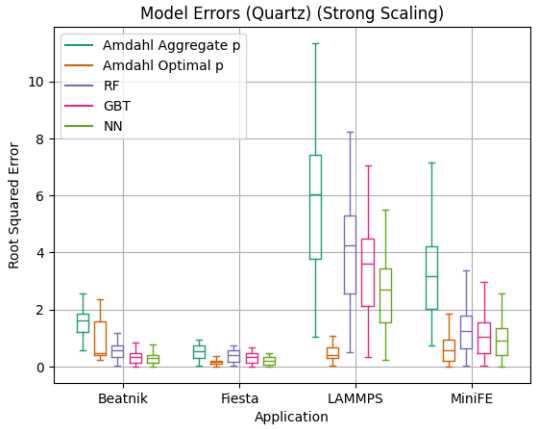
where m is some problem size for a given application, and D_m is a subset of the training set D that only contains runs of problem size m.

5.2 Impact of Machine Learning Method on Prediction Accuracy

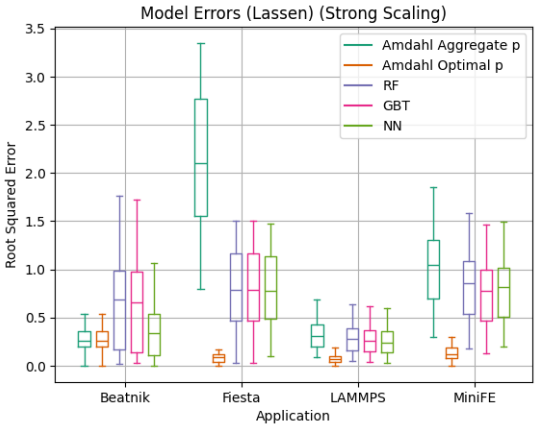
We first examine how different approaches to learning the overhead function impact performance prediction accuracy and compare these predictions with standard Amdahl's law predictions and with Amdahl's law modified to handle changes in problem sizes (Equation 2). To do so, we train the following machine learning models with the above-described data:

- Random Forest Regression using the scikit-learn library [14], using a "Friedman mean squared error" *criterion*, and a *max depth* of 5.
- Gradient Boosted Tree Regression using the XGBoost library [7], using a *squared error* objective function, and a *max depth* of 5.
- Neural Networks using the TensorFlow library [5]. We construct a fully connected Neural Network with an input layer of 16, 3 hidden layers of dimensions 16x16x8 using *relu* activation functions, and an output layer of dimension 1 using a *softplus* activation function.

Tables 1 and 2, and Figure 2 show the prediction errors for single-level parallelism strong scaling tests ($w = 1$) on both the Quartz and Lassen systems along with the error rate of unmodified Amdahl's law scaling predictions using a fixed aggregate percentage parallelism measured using the training set as described in Section 2.2. Specifically, Table 1 shows the root mean squared error across all predictions in each case, while Figure 2 shows the full distribution of root squared prediction errors across all sizes in the testing set. Tables 3 and 4, and Figure 3 similarly show the root mean squared error and distribution of root squared prediction errors, respectively, where problem size *and* number of processes change simultaneously ($w \leq 0.5$).



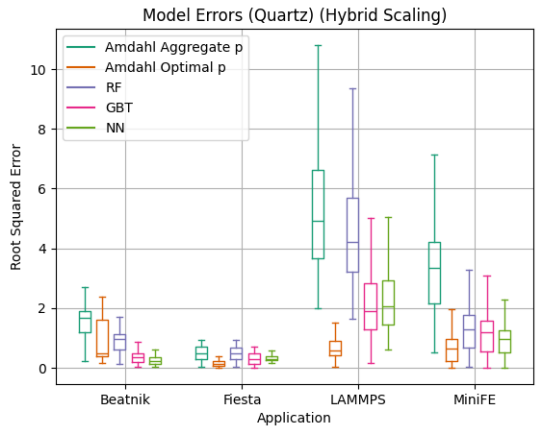
(a) Quartz



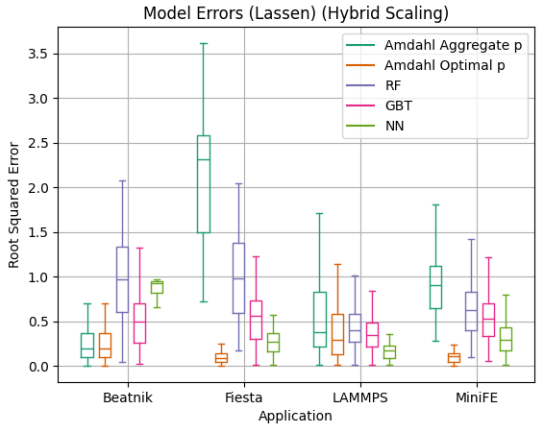
(b) Lassen

Figure 2: Strong scaling prediction error of various machine learning models and Amdahl's law projections. Machine learning models and percentage parallel parameters were fit with data where both the number of processes and size of the problem were less than or equal to the median of the entire data set (Quartz $c_f = 100$, Lassen $c_f = 10$).

These results demonstrate that the proposed machine learning model is (1) able to more accurately predict performance in the testing set than Amdahl's law predictions and (2) that the neural network approach more accurately predicts scaling performance than the other machine learning approaches that we used for comparison, with gradient-boosted trees being a close second. Specifically, the neural network machine learning predictions are very good ($RMSE < 1$, often much less than 1) on Lassen for all applications, are good on Quartz with the exception of LAMMPS (where $RMSE > 2$), which has a very short run time. For those cases, where



(a) Quartz



(b) Lassen

Figure 3: Hybrid scaling ($w \leq 0.5$) prediction error of various machine learning models and modified Amdahl's law projections. Machine learning models and percentage parallel parameters were fit with data where both the number of processes and size of the problem were less than or equal to the median of the entire data set (Quartz $c_f = 100$, Lassen $c_f = 10$).

the RMSE is high, particularly for LAMMPS on Quartz, Figure 2 shows that this is due to large variance in prediction errors across the large range of problem sizes in the testing set.

We hypothesize that these performance differences are due to (1) the ability of the overhead-based model to correct for inaccurate percentage parallel estimates when the problem size is not in the training set and (2) neural network approaches are better able to numerically extrapolate data beyond the training set than tree-based machine learning methods whose outputs are more closely tied to the training sets.

5.3 Parallel Percentage Approach Impact

To better understand the strengths and limitations of both our proposed approach and the traditional Amdahl's Law approach, we next compared the performance of the neural network model to Amdahl's law with a problem-specific percent parallel estimate. In contrast to the experiment in the previous section, Amdahl's Law is given a percentage parallel estimate for the problem chosen to minimize mean-squared prediction error on that specific problem in the training set. That is, we compare neural network performance versus an *optimal* Amdahl's law performance where the optimal Amdahl's law estimates use per-problem estimates of the parallel percentage model parameter, including for problem sizes *not in the testing set if the target problem size is greater than* w_f.

To this end, Figures 2 and 3, as well as Tables 1, 2, 3, and 4 also show the results of these tests along with the Amdahl's law prediction that minimizes mean squared error for all problem sizes in the training set for comparison. These results demonstrate that Amdahl's law generally outperforms our proposed machine learning model when full performance information is available for a specific problem size/system combination. However, with only limited percentage parallel performance information, our machine learning model significantly outperforms Amdahl's law predictions. In addition, when trained with hybrid-scaling data, the neural networks

Model	Applications			
	Beatnik	MiniFE	Fiesta	LAMMPS
RF	0.561	1.337	0.371	4.01
GBT	0.332	1.129	0.31	3.382
NN	0.298	0.983	0.21	2.561
Oliveira	0.746	1.324	0.4	2.139
Amdahl Aggregate p	1.576	3.351	0.505	5.708
Amdahl Optimal p	0.843	0.662	0.155	0.556
Amdahl Optimal p FD	1.522	0.491	0.152	0.596

Table 1: Model RMSE (Quartz, Strong Scaling)

Model	Applications			
	Beatnik	MiniFE	Fiesta	LAMMPS
RF	0.688	0.815	0.798	0.278
GBT	0.672	0.735	0.798	0.263
NN	0.378	0.782	0.811	0.253
Oliveira	0.272	0.366	0.475	0.191
Amdahl Aggregate p	0.309	1.006	2.124	0.317
Amdahl Optimal p	0.309	0.13	0.083	0.074
Amdahl Optimal p FD	0.309	0.045	0.033	0.065

Table 2: Model RMSE (Lassen, Strong Scaling)

Model	Applications			
	Beatnik	MiniFE	Fiesta	LAMMPS
RF	0.915	1.357	0.478	4.647
GBT	0.363	1.206	0.301	2.147
NN	0.25	1.002	0.321	2.309
Oliveira	N/A	N/A	N/A	N/A
Amdahl Aggregate p	1.615	3.441	0.483	5.365
Amdahl Optimal p	0.861	0.709	0.152	0.706
Amdahl Optimal p FD	1.372	0.491	0.14	0.794

Table 3: Model RMSE (Quartz, Hybrid Scaling)

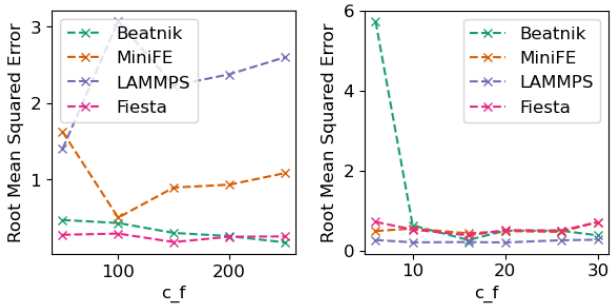
Model	Applications			
	Beatnik	MiniFE	Fiesta	LAMMPS
RF	1.01	0.65	0.998	0.442
GBT	0.508	0.557	0.53	0.377
NN	0.871	0.316	0.277	0.186
Oliveira	N/A	N/A	N/A	N/A
Amdahl Aggregate p	0.254	0.925	2.113	0.571
Amdahl Optimal p	0.254	0.103	0.093	0.396
Amdahl Optimal p FD	0.254	0.043	0.057	0.39

Table 4: Model RMSE (Lassen, Hybrid Scaling)

and gradient-boosted trees outperform Amdahl's law, even when full performance information is available for Beatnik on Quartz.

5.4 Sensitivity to Training Set Size

Finally, we examined the sensitivity of our approach to changes in the training set size. Figure 4 shows the results for varying the max process count in the training set for single-level parallelism runs by changing c_f from its median value (100 on Quartz and 10 on Lassen) to both higher and lower values that either increased or decreased the size of the training set. For these tests, w_f was fixed at the median problem size.

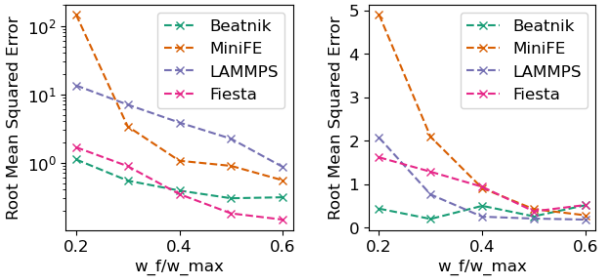


(a) Quartz **(b) Lassen**

Figure 4: Sensitivity of prediction error of different applications to changes in the largest number of processes in the training set.

These results show that model performance is relatively insensitive to process counts. We hypothesize that the general insensitivity to process count is due to the overall structure of Amdahl's law handling changes in process count while the machine learning model corrects for inaccuracies in estimating percentage parallelism.

For changes in the problem size training set, we fixed c_f at the median process count and varied the ratio of the largest problem size in the training set to the largest problem size (w_f/w_{max}) between 0.2 and 0.6, where w_{max} is the largest problem size that a given application was run on. In contrast to process-count sensitivity, our proposed method is much more sensitive to the size of the problem training set. As discussed above, the machine learning model must correct for inaccuracies in the estimation of the available parallelism in a given problem size. Because of this, it needs a relatively significant amount of training data to learn these



(a) Quartz **(b) Lassen**

Figure 5: Sensitivity of prediction error of different applications to changes in the largest problem size in the training set (w_f) versus the maximum problem size (w_{max}) for each application.

Model	Applications			
	Beatnik	MiniFE	Fiesta	LAMMPS
RF	15.374	1.869	1.34	3.973
GBT	1.753	1.41	0.345	2.375
NN	0.808	0.868	1.389	73.708
Oliveira	0.997	1.272	0.918	1.323
Amdahl Single Mesh	1.215	4.264	26.411	6.692
Amdahl Multi Mesh	1.185	0.466	8.431	0.92
Amdahl Multi Mesh FD	1.215	0.498	0.499	0.578

Table 5: Model RMSE (Quartz, Multi-Core, Strong Scaling)

trends, particularly for MiniFE and LAMMPS. We also recognize that the errors for MiniFE and LAMMPS predictions are significantly high due to the low time to completion of these problems at large processor counts. The runtimes of these applications even at the largest possible problem sizes for strong-scaling runs are a few seconds long, so any amount of prediction deviation results in largely magnified errors.

5.5 Modeling Multi-Level Parallelism Scaling

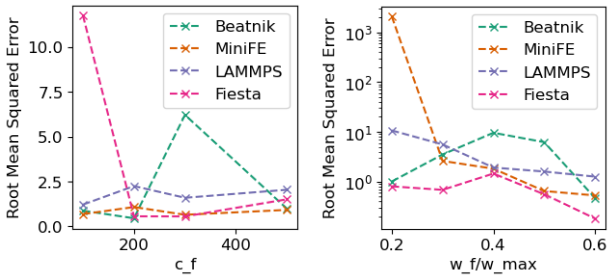
Furthermore, we evaluate the accuracy of the proposed approach when addressing multi-level parallelism, particularly the two levels of parallelism when running multiple single-core MPI processes per node on the LLNL Quartz system. We first examine how accurate the approach is based on the total number of processes running regardless of the number of processes per node. Finally, we examine the overall sensitivity of this approach to the amount of data used to train the models. We have not yet examined model accuracy on multi-core multi-GPU systems.

Tables 5 and 6 compare the multi-core/multi-process strong and hybrid scaling prediction accuracy of Amdahl's law with aggregate parallelism information, Amdahl's law with per-problem parallelism information, and our proposed approach with aggregate parallelism information augmented with machine learning models trained to estimate additional parallel overheads. The machine learning approaches have similar RMSEs as compared to the single-core cases, with slightly better performance for LAMMPS and MiniFE, except for neural networks, where the performance dramatically

Model	Applications			
	Beatnik	MiniFE	Fiesta	LAMMPS
RF	0.922	3.496	1.838	5.147
GBT	1.199	1.389	0.445	1.744
NN	0.684	0.529	0.711	1.292
Oliveira	N/A	N/A	N/A	N/A
Amdahl Single Mesh	1.097	6.067	21.203	7.118
Amdahl Multi Mesh	1.058	0.607	0.557	1.007
Amdahl Multi Mesh FD	1.097	0.561	0.557	0.91

Table 6: Model RMSE (Quartz, Multi-Core, Hybrid Scaling)

worsens with LAMMPS predictions in the strong-scaling case. We believe this is because of a few outliers in the training data set that may have caused the neural networks to falsely correlate some configurations of nodes/cores with large overheads. We do not see this behavior in the tree-based approaches as they are generally more robust against outliers than neural networks. Otherwise, we believe that the overall increase in performance, especially in hybrid-scaling predictions is because of the increased amount of training data available to the models when training on multi-level parallelism and hybrid-scaling data.



(a) Quartz (c_f Sensitivity) **(b) Quartz (w_f Sensitivity)**

Figure 6: Sensitivity of prediction error of different applications to changes in the largest number of processes (c_f) and largest problem size (w_f) in the training set versus the maximum problem size (w_{max}) for each application aggregated across 1, 2, 4, 8, 16, and 32 process-per-node runs for Quartz.

Finally, Figure 6 shows the sensitivity of the neural network approach to the amount of training data provided on multi-core systems in terms of number of processes and problem size. The results are in many similar to the single process per node case shown in the previous subsection; sensitivity to the number of processes in the training set is relatively low but sensitivity to the range of problem sizes in the training set is much higher. There are two important differences in sensitivity compared to the single process per node case, however. First, MiniFE sensitivity to the number of processes is lower, potentially because of the increased number of processes in the data set. Second, MiniFE is even *more* sensitive to changes in the amount of problem size data in the training set than in the single core case; note the use of a log scale on the Y-axes for problem size sensitivity. We have not yet determined the cause of this increased sensitivity but believe it may be due to the increased number of parameters the neural network model must be trained to account for.

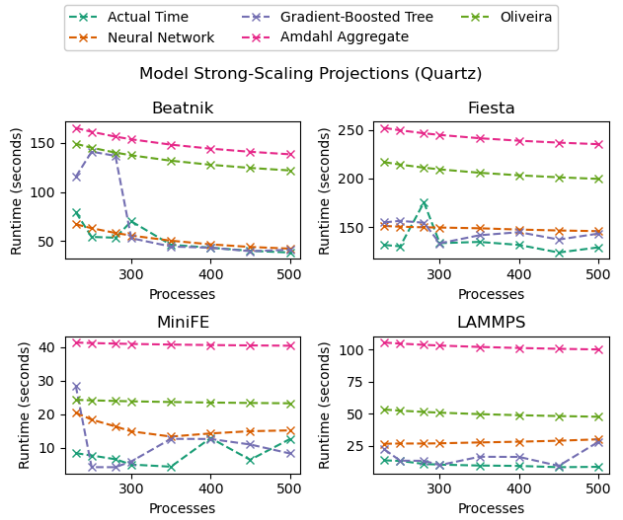
5.6 Comparison to the Current State-Of-The-Art Model

Finally, we evaluate the strong-scaling performance of our models against the current state-of-the-art model with similar data requirements to that of ours. We choose Oliveira et. al's model [17] as it is the best-performing peer-reviewed model, that we could find, that can predict strong-scaling overhead-corrected speedups with limited timing data without relying on additional application profiling.

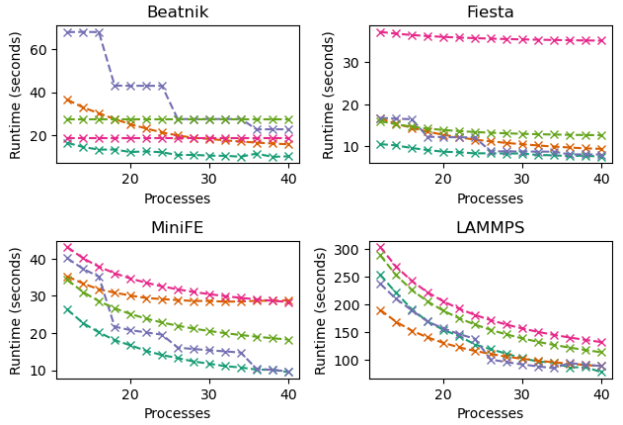
We implement Oliveira's speedup model [17], and train it using speedup data collected from our set of applications and machines. We set the cutoff values c_f and w_f to the median values of their respective applications and systems, to evaluate the performance of this model on larger unseen problem sizes and processor counts. This speedup model is fit using a particle swarm optimization (PSO) algorithm [27], exactly as described in [17]. We used 100 estimators over 1000 iterations, which ensured that the optimizer converged when fit with data from every application/system pair.

Tables 1 and 2 show the prediction errors for single-level parallelism strong-scaling tests on both Quartz and Lassen systems for our chosen subset of well-performing machine learning models, Oliveira's speedup model, and Amdahl's Law. These results demonstrate that the neural network and gradient-boosted tree models, on average, perform better than Oliveira's speedup model on Quartz, except in the case of LAMMPS, where our models perform slightly worse. Meanwhile, prediction errors for gradient-boosted trees and neural networks on Lassen are slightly worse than Oliveira's speedup model predictions on Lassen for all of the applications. Table 5 shows the prediction errors for multi-level parallelism strong-scaling tests for the same models. Beatnik and MiniFE predictions of the neural network are slightly better than that of Oliveira's model, while Fiesta predictions are slightly worse. We note that the neural network performs significantly worse than Oliveira's model for LAMMPS predictions when being evaluated on multi-level parallelism data.

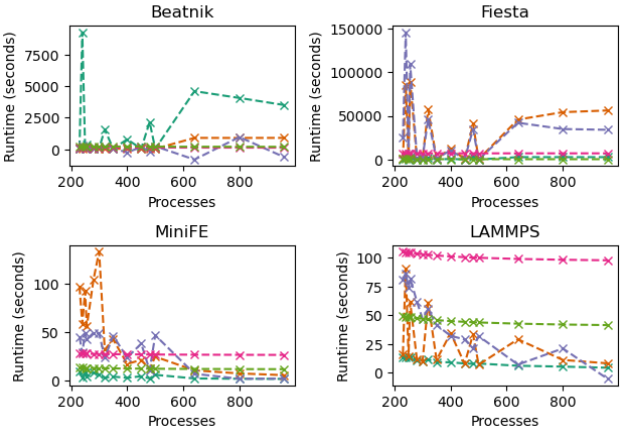
We further evaluate the performance of these models by plotting the strong-scaling projections of the largest available problem sizes for all of the applications on processes larger than c_f. Figure 7 shows these projections for single-level parallelism runs on Quartz and Lassen, and multi-level parallelism runs on Quartz. The neural networks and gradient-boosted trees consistently outperform all other models when running with single-level parallelism on Quartz. On Lassen, the neural network projections are on par with that of Oliveira's model, except in the case of MiniFE where the speedups are underestimated at higher processor counts. However, gradient-boosted trees seem to perform better than both the neural network and Oliveira's speedup model in the same case. Multi-level parallelism runs on Quartz result in inconsistent and erratic scaling behavior, as shown in the corresponding plots for Beatnik. Outliers in the training data set of the machine learning models may have caused them to falsely correlate some configurations of nodes/cores with large overheads. Amdahl's law and Oliveira's speedup model, being analytical models, are more robust against outliers.



(a) Quartz



(b) Lassen



(c) Quartz (Multi-Level Parallelism)

Figure 7: Strong-Scaling projections for the largest available problem size

6 RELATED WORK

Similar to our work, Höfinger et. al [12] devise a multiplicative overhead factor to Amdahl's law that captures the costs incurred from parallelism. This overhead factor is assumed to strictly be the time spent in MPI operations and is represented as a multi-parameter expression where the parameters are obtained by fitting its curve to the applications' runtime data. However, this approach is limited to fixed problem sizes, and only considers strong-scaling scenarios. In addition, it also requires MPI profiling data to compute ground truths for the overheads.

Oliveira et. al [17] propose a model derived from Amdahl's Law that contains a parameter for overhead incurred from parallelism that is a function of the problem size and the number of processors. However, this model can only predict speedups in strictly strong-scaling scenarios.

Narayanan et. al [16] propose a Serial Scaling Model (SSM) that uses six empirically determined parameters to estimate the serial and parallel scaling of an application based on the input sizes and the number of threads in a multi-core setting. This model can be adjusted to represent both - Amdahl's and Gustafson's laws based on the application in use. However, this approach is more invasive than our approach and the others listed above as it requires thread-activity instrumentation. In addition, this approach is also limited to multi-core architectures and cannot be extended to distributed computing settings.

7 FUTURE WORK AND CONCLUSION

In this paper, we demonstrate the augmenting an extended version of Amdahl's law with a machine-learning-determined overhead factor provides more accurate performance predictions. Augmenting Amdahl's law in this way allows the machine learning model to compensate for how changes in the problem size and differences between inter- and intra-node parallelism impact performance in ways that Amdahl's law cannot. This improvement is particularly significant when the training set used to determine the percentage of the workload that can be parallelized does not include the specific workload whose performance is being predicted and aggregate statistics must be used to predict these value. In addition, we demonstrate that neural network machine learning models are most effective of the studied models, and that the trained models are relatively insensitive to the number of processes in the training set.

We've also compared the performance of these augmented models to the current state-of-the-art model with similar data requirements. When predicting speedups in cases with single-level parallelism, our models generally outperform [17], but perform worse with multi-level parallelism predictions due to outliers in the training set.

We plan on further evaluating the performance of our proposed models by using them to predict weak-scaling application runs and more complex system configurations where resource limitations (e.g. network or cache bandwidth) make performance prediction more challenging. We also plan on addressing the limitations of our models in multi-level parallelism settings by making them more robust to outliers in the training data. In addition, we plan on further gathering data from more machines and applications that add to

the diversity of our current pool. As not all applications scale the serial portion of their code as the problem size increases, this will require adding additional parameters to our proposed model that can adapt the assumptions of both Amdahl [2], and Gustafson [10] regarding the scaling of the serial portions of the applications.

ACKNOWLEDGMENT

Hidden to comply with the double-blind review process.

REFERENCES

[1] Dong H. Ahn et al. "Flux: Overcoming scheduling challenges for exascale workflows". In: *Future Generation Computer Systems* 110 (Sept. 2020), pp. 202–213. ISSN: 0167-739X. DOI: 10.1016/j.future.2020.04.006. URL: https://www.sciencedirect.com/science/article/pii/S0167739X19317169 (visited on 11/22/2023).

[2] Gene M. Amdahl. "Validity of the single processor approach to achieving large scale computing capabilities". In: *Proceedings of the April 18-20, 1967, spring joint computer conference*. AFIPS '67 (Spring). New York, NY, USA: Association for Computing Machinery, Apr. 1967, pp. 483–485. ISBN: 978-1-4503-7895-6. DOI: 10.1145/1465482.1465560. URL: https://dl.acm.org/doi/10.1145/1465482.1465560 (visited on 06/28/2023).

[3] Yehia Arafa et al. "PPT-GPU: Scalable GPU Performance Modeling". In: *IEEE Computer Architecture Letters* 18.1 (Jan. 2019). Conference Name: IEEE Computer Architecture Letters, pp. 55–58. ISSN: 1556-6064. DOI: 10.1109/LCA.2019.2904497.

[4] Alan Ayala et al. *heFFTe: Highly Efficient FFT for Exascale*. Place: Amsterdam, Netherlands. 2020. DOI: https://doi.org/10.1007/978-3-030-50371-0_19.

[5] Ekaba Bisong. "TensorFlow 2.0 and Keras". en. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Ed. by Ekaba Bisong. Berkeley, CA: Apress, 2019, pp. 347–399. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_30. URL: https://doi.org/10.1007/978-1-4842-4470-8_30 (visited on 08/15/2023).

[6] Patrick Bridges et al. *CUP-ECS/beatnik: Initial Cabana/Cajita Low/High-order Z-model Interface Solver. Benchmark for evaluating the performance of algorithms requiring global communication. Beatnik is also a precursor to potential later a High Performance Parallel Interface solver.* URL: https://github.com/CUP-ECS/beatnik/tree/main (visited on 10/10/2022).

[7] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. New York, NY, USA: Association for Computing Machinery, Aug. 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: https://dl.acm.org/doi/10.1145/2939672.2939785 (visited on 04/27/2024).

[8] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. "Why do tree-based models still outperform deep learning on typical tabular data?" en. In: *Advances in Neural Information Processing Systems* 35 (Dec. 2022), pp. 507–520. (Visited on 12/12/2023).

[9] Jian Guo et al. "Machine learning predictions for underestimation of job runtime on HPC system". In: *Supercomputing Frontiers: 4th Asian Conference, SCFA 2018, Singapore, March 26-29, 2018, Proceedings 4*. Springer International Publishing, 2018, pp. 179–198.

[10] John L. Gustafson. "Reevaluating Amdahl's law". In: *Communications of the ACM* 31.5 (May 1988), pp. 532–533. ISSN: 0001-0782. DOI: 10.1145/42411.42415. URL: https://dl.acm.org/doi/10.1145/42411.42415 (visited on 06/29/2023).

[11] Tin Kam Ho. "Random decision forests". In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. Vol. 1. Aug. 1995, 278–282 vol.1. DOI: 10.1109/ICDAR.1995.598994. URL: https://ieeexplore.ieee.org/abstract/document/598994 (visited on 12/11/2023).

[12] Siegfried Höfinger and Ernst Haunschmid. "Modelling parallel overhead from simple run-time records". en. In: *The Journal of Supercomputing* 73.10 (Oct. 2017), pp. 4390–4406. ISSN: 1573-0484. DOI: 10.1007/s11227-017-2023-9. URL: https://doi.org/10.1007/s11227-017-2023-9 (visited on 07/31/2023).

[13] Clayton Hughes et al. *GPGPU-Sim Overview*. Tech. rep. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2019.

[14] Oliver Kramer. "Scikit-Learn". en. In: *Machine Learning for Evolution Strategies*. Ed. by Oliver Kramer. Studies in Big Data. Cham: Springer International Publishing, 2016, pp. 45–53. ISBN: 978-3-319-33383-0. DOI: 10.1007/978-3-319-33383-0_5. URL: https://doi.org/10.1007/978-3-319-33383-0_5 (visited on 08/15/2023).

[15] Paul Lin, Michael Heroux, and Richard Barrett. *Miniapplications: a Promising Approach to Improve the Performance of Computational Mechanics Codes*. en.

[16] Surya Narayanan, Bharath N. Swamy, and André Seznec. "Impact of Serial Scaling of Multi-threaded Programs in Many-Core Era". In: *2014 International Symposium on Computer Architecture and High Performance Computing Workshop*. Oct. 2014, pp. 36–41. DOI: 10.1109/SBAC-PADW.2014.9.

[17] Victor H. F. Oliveira et al. "Application Speedup Characterization: Modeling Parallelization Overhead and Variations of Problem Size and Number of Cores." In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE '18. New York, NY, USA: Association for Computing Machinery, Apr. 2018, pp. 43–44. ISBN: 978-1-4503-5629-9. DOI: 10.1145/3185768.3185770. URL: https://dl.acm.org/doi/10.1145/3185768.3185770 (visited on 08/01/2023).

[18] Gavin Pandya and Steve Shkoller. *3D Interface Models for Rayleigh-Taylor Problems*. arXiv:2201.04538 [physics]. Jan. 2022. DOI: 10.48550/arXiv.2201.04538. URL: http://arxiv.org/abs/2201.04538 (visited on 10/10/2022).

[19] *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*. EN. Archive Location: world. DOI: 10.5555/104279. URL: https://dl.acm.org/doi/abs/10.5555/104279 (visited on 12/11/2023).

[20] Ivy Peng et al. "A Holistic View of Memory Utilization on HPC Systems: Current and Future Trends". In: *Proceedings of the International Symposium on Memory Systems*. MEMSYS '21. New York, NY, USA: Association for Computing Machinery, May 2022, pp. 1–11. ISBN: 978-1-4503-8570-1. DOI: 10.1145/3488423.3519336. URL: https://dl.acm.org/doi/10.1145/3488423.3519336 (visited on 08/05/2023).

[21] Gonzalo P. Rodrigo et al. "Enabling Workflow-Aware Scheduling on HPC Systems". In: *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. HPDC '17. New York, NY, USA: Association for Computing Machinery, June 2017, pp. 3–14. ISBN: 978-1-4503-4699-3. DOI: 10.1145/3078597.3078604. URL: https://dl.acm.org/doi/10.1145/3078597.3078604 (visited on 08/05/2023).

[22] Brian Romero et al. "Simulations of the shock-driven Kelvin–Helmholtz instability in inclined gas curtains". en. In: *Physics of Fluids* 33.6 (June 2021), p. 064103. ISSN: 1070-6631, 1089-7666. DOI: 10.1063/5.0051459. URL: https://pubs.aip.org/aip/pof/article/1065635 (visited on 07/20/2023).

[23] Karan Singh et al. "Predicting parallel application performance via machine learning approaches". en. In: *Concurrency and Computation: Practice and Experience* 19.17 (2007). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.1171, pp. 2219–2235. ISSN: 1532-0634. DOI: 10.1002/cpe.1171. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.1171 (visited on 08/17/2022).

[24] Stuart Slattery et al. "Cabana: A performance portable library for particle-based simulations". In: *Journal of Open Source Software* 7.72 (2022), p. 4115.

[25] Jingwei Sun et al. "Automated Performance Modeling of HPC Applications Using Machine Learning". In: *IEEE Transactions on Computers* 69.5 (May 2020). Conference Name: IEEE Transactions on Computers, pp. 749–763. ISSN: 1557-9956. DOI: 10.1109/TC.2020.2964767.

[26] Aidan P. Thompson et al. "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales". en. In: *Computer Physics Communications* 271 (Feb. 2022), p. 108171. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2021.108171. URL: https://www.sciencedirect.com/science/article/pii/S0010465521002836 (visited on 07/20/2023).

[27] Dongshu Wang, Dapei Tan, and Lei Liu. "Particle swarm optimization algorithm: an overview". en. In: *Soft Computing* 22.2 (Jan. 2018), pp. 387–408. ISSN: 1433-7479. DOI: 10.1007/s00500-016-2474-6. URL: https://doi.org/10.1007/s00500-016-2474-6 (visited on 04/28/2024).

[28] Gen Xu et al. "Simulation-Based Performance Prediction of HPC Applications: A Case Study of HPL". In: *2020 IEEE/ACM International Workshop on HPC User Support Tools (HUST) and Workshop on Programming and Performance Visualization Tools (ProTools)*. Nov. 2020, pp. 81–88. DOI: 10.1109/HUSTProtools51951.2020.00016.

[29] Jerry Ye et al. "Stochastic gradient boosted distributed decision trees". In: *Proceedings of the 18th ACM conference on Information and knowledge management*. CIKM '09. New York, NY, USA: Association for Computing Machinery, Nov. 2009, pp. 2061–2064. ISBN: 978-1-60558-512-3. DOI: 10.1145/1645953.1646301. URL: https://dl.acm.org/doi/10.1145/1645953.1646301 (visited on 04/24/2024).