Eurographics Symposium on Parallel Graphics and Visualization (2021)
S. Frey, J. Huang, F. Sadlo (Editors)

Improving Parallel Particle Advection Performance With Machine Learning

Samuel D. Schwartz,[1] Hank Childs[1] and David Pugmire[2]

[1]Department of Computer and Information Science, University of Oregon, Oregon, United States
[2]Oak Ridge National Laboratory, Oak Ridge, Tennessee, United States

Abstract

Parallelized particle advection algorithms are a key visualization tool for domain scientists. They are also very computationally expensive to run. Machine learning techniques have been widely used in regression settings to predict results based on a set of input features. Our work describes an approach for parallel particle advection optimization; an approach which uses a machine learning algorithm at its core. We specifically investigate how our approach operates when applied to a GPU-based parallel particle advection algorithm. We examine the efficacy of 14 different machine learning models (including six neural network models) in our approach and validate the top theoretical performers' accuracy with respect to speedup over a baseline algorithm. From our investigations we find that the machine learning based optimization architecture yielded by our approach considers 45% of examined particle advection workloads to be "acceleratable," of which 74% truly do receive acceleration. Of the "acceleratable" workloads, our architecture achieves an overall average speedup of 12.7% and an average speedup of 20.3% for workloads that took longer than 60 seconds to execute.

CCS Concepts

• ***Human-centered computing*** → ***Scientific visualization; Visualization techniques;***

1. Introduction

Flow visualization is a foundational technique for understanding the behavior of vector fields in scientific data. The techniques that encompass flow visualization are varied, and include operations such as streamlines, pathlines, FTLE, Poincare analysis. All of these techniques are built upon a common foundational algorithm for performing particle advection.

In particle advection, a massless particle called a seed is placed in the vector field. The path of each seed is computed by successfully displacing the position by the velocities in the vector field. The computed path can be described using an ordinary differential equation, and in practice is computed iteratively using numerical techniques.

The costs of particle advection based flow visualization and analysis can be very expensive depending on the workload. Each of these workloads and intended use cases can have dramatically different performance results. As an example, when using streamlines to interactively understand a vector field, a small number of seeds might be placed at strategic locations within the dataset and advected for a large number of steps. It can also be useful to generate a large number of long streamlines and then either analyze their characteristics, or use some interactive methods to cull away streamlines and focus on a particular subset. In the case of FTLE or Poincare analysis, one or more seeds can be placed in every cell the mesh, and advected for either a short or long duration depending on the nature of the analysis. This can result in billions of particles being advected for thousands of steps. As a result, particle advection workloads can have excessive times for solving.

Particle advection algorithms become more complicated in the context of supercomputers. In these settings, the data are typically very large and must be distributed across the memory space of a number of nodes in the supercomputer. This requires the use of parallel particle advection algorithms which are notoriously difficult to efficiently parallelize. The basic challenge in parallel particle advection algorithms is to ensure that the data block and seed are brought together at the right time. It is difficult to coordinate this bringing together of data and particles, and so these methods are subject to load imbalance. In the context of in situ processing on a supercomputer, performance is even more critical. When analysis and visualization are expensive, they will be performed less frequently while a simulation is running. For an in situ setting, where all data cannot be written to disk, this results in lost opportunities for analysis of data and decreased understanding of scientific data.

The computational expense of algorithms, and the impact on the extraction of understanding from large data makes performance of these types of algorithms important. The performance of the particle advection algorithm does not correlate directly with the size of the data (both grid size and the number of particles and steps to

take). The performance is also correlated to the nature of the vector field, which varies across types of simulations, and the placement of the seeds. The data blocks that are needed at any one time are function of the seed placement, and the nature of the flow. The problem of bringing together of seeds and data in an optimal way is difficult in distributed memory setting. The relationship among these factors are not fully understood and is an active area of research. Further, particle advection algorithms have a number of configuration parameters that determine their behavior and performance. These additional parameters pose further challenges in gaining optimal performance.

These host of challenges makes it difficult to know how to configure and run a particle advection algorithm a priori. Our research question asks if we can use machine learning to optimize a parallel particle advection algorithm. Machine learning techniques are powerful tools that can be used to understand complex processes. These techniques can discover complex relationships among a set of inputs and a desired set of outputs. In this paper we generate a set of training data from over 2500 runs from a parameter sweep of a particle advection algorithm. We formulate the parameters of a parallel particle advection algorithm as an optimization problem and assess the efficacy of several machine learning algorithms in addressing the optimization problem. We trained a total of 14 different types of models to this data and selected a model that is best suited to predicting optimal parameter settings for the algorithm. We then take this model and run a set of validation experiments to confirm our model. Our results identified a large set of workloads that could be accelerated using optimal parameter settings. Our validation runs using the machine learning model resulted in 60% of these identified uses cases being accelerated. Our model can be used within a particle advection system. Given a workload specified by a user, the model can be used to predict optimal parameter settings for the algorithm.

2. Related Work

Related work is divided into three parts: machine learning overview (2.1), performance optimization (2.2), and particle advection (2.3).

2.1. Machine Learning, Supervised General Regression, and High Performance Contexts

Machine learning has been incorporated as a tool in a variety of settings to predict or infer optimal outcomes. Supervised regression is a type of machine learning wherein machine learning models predict new continuous scalar outcomes based on inputted features after being trained on a set of existing set of ordered pairs of (features, continuous scalar outcome).

One of the simplest and most well known supervised regression machine learning model is linear regression. Linear regression involves fitting a line of the form $f(\vec{x}) = \vec{x} \cdot \vec{\beta} + \beta_0$ to an existing set of points in $\mathbb{R}^n$ (typically by ordinary least squares). A subsequent prediction given an additional point $\vec{p} \in \mathbb{R}^n$ is the calculated outcome of $f(\vec{p})$.

Many other non-linear techniques for supervised regression exist. Often derived from supervised classification models, these classic supervised regression models include random forests, support vector machines (SVMs), k nearest neighbor variants, among many others. We find the textbooks [Bis06, HTF09] provide an excellent overview of all of these techniques.

Some kinds of neural network are also supervised machine learning techniques. While many architectures of neural network models are quite complex, other smaller and simpler models offer excellent predictive performance, too. Goodfellow et al. [GBC16], provide a strong starting point for understanding the theoretical underpinnings of neural networks.

2.2. Optimizing Algorithm Performance

Machine learning, in the broadest sense, has been used to autotune high performance algorithms by using optimization techniques such as genetic algorithms [MPP*18] and random correlation grid search, with Balaprakash et al. providing a comprehensive overview of autotuning oriented optimization techniques [BDG*18].

Tuncer et al. [TAZ*17] provided an early example of supervised machine learning in high performance algorithm analysis. They constructed a framework which decomposed time series data of anomalies in a high performance computing algorithm into trainable data and assessed the efficacy of several machine learning techniques, among them k nearest neighbors, support vector classifiers with a radial basis function kernel, and random forests. They found that the random forest model worked best for the problem they investigated. Their work was focused purely on anomaly detection, however, and only incidental to optimization.

2.3. Particle Advection in Parallelized Settings

Particle advection has been studied for a long time to find optimal ways for load balancing the problem. There are two fundamental ways of parallelizing the problem, Parallelize over Data (POD) and Parallelize over Particles (POP), and hybrid methods that use elements of both approaches. In the POD method [PPG12], the data are distributed over the processes. Each process advects the particles located in its data block until it terminates or exists the spatial bounds of the block. Particles are communicated to the process that owns the needed block when a particle when they exit. This process continues until all particles terminate.

A number of extensions to POD have been considered to improve the performance of the algorithm. Sisneros et al. [SP16] studied the impact of communication granularity in the POD algorithm. Optimizations for spatial decomposition have been studied in [PRN*11] to improve load balance. Work in hybrid parallelism (both shared and distributed memory) have also been studied for particle advection. Camp et al. [CGC*11] first looked at these methods for streamlines, followed by work looking at the performance on different types of workloads and hardware types [CKP*13, CBP*14]. Finally, Pugmire et al. [PYK*18] provided a hardware portable method for shared memory particle advection using VTK-m [MSU*16].

3. Algorithm and Data Corpus

In this section we describe the particle advection algorithm, and the experiments performed to generate our data corpus. The algorithm characteristics as well as the workload characteristics are used to train the machine learning models.

3.1. Particle Advection Algorithm

In this study we used the Parallelize-Over-Data algorithm (POD) [PPG12]. This algorithm divides the blocks of data over the nodes of computer and communicates particles between nodes as they travel between blocks. Each process is assigned one or more blocks and is responsible for advecting all of the particles that travel through these blocks. Each process advects the particles located in its data blocks until they terminate or exit the spatial bounds of the block. When a particle leaves a data block, asynchronous communication is used to send it to the process that owns the required data. A counter is maintained across all of the processes of the number of active particles. The algorithm exits once there are no more active particles.

When running the POD algorithm using a GPU, all of the particles on a process are loaded onto the GPU and advected until all have been processed. Processing on each particle will continue until it has exited the current data block, or a termination criterion has been met (e.g, maximum number of steps taken). In this work, we use the following two parameters to control the behavior of the POD algorithm.

- "Batch Size": which specifies how many advection steps to take before checking the status of particles. The GPU processes a large number of particles in parallel. The GPU will process these particles in rounds depending on how many hardware threads are available. If particles that have terminated – or have traveled out of the current block – can be removed from the GPU, then efficiency of the GPU can be improved by assigning active particles to these recently vacated threads. When "Batch Size" is greater than or equal to the total number of advection steps, all particles will be processed.
- "Delay Send": controls when communication is performed. When "Delay Send" is True, particles are communicated after all of the particles have been processed by the GPU. When False, communication is performed after each check for particle status, as specified by the "Batch Size" parameter.

The default settings for the POD algorithm is for the "Batch Size" to be ALL (i.e., process all particles without checking status), and for "Delay Send" to be True (i.e, communicate particles after all particles have been processed.

These parameter settings can impact algorithm performance in several different ways. Using the default settings for the POD algorithm ("Batch Size" = ALL and "Delay Send" = True), particles are advected until they either terminate, or leave the block. When running this algorithm on a many core device (e.g., GPU), all of the particles are sent to the GPU to be advected. Once all of the particles are finished, the state of each particle determines if it has terminated or needs to communicated to the process that has the data block. There are two implications of this. First, the GPU will be busy until *all* of the particles are done. Because the GPU can process a large number of particles in parallel it is likely that some particles will finish earlier than others. For example, assume that 1000 particles are loaded onto the GPU for advection. If 999 particles terminate after one advection step, the GPU will remain busy until the final particle finishes. Second, communication is performed only after all of the particles are finished advecting. This can result in fewer communications of larger groups of particles, which may be more efficient. But, it also slows down the rate at which busy processes can send particles to other processes, which are potentially idle.

Optimal settings for "Batch Size" and "Delay Send" for a given particle advection workload can decrease the time the algorithm takes. Optimal settings for "Batch Size" will increase GPU utilization by removing particles that become inactive. "Delay Send" will change the communication patterns in the algorithm. When True, larger sets of particles are communicated less frequently. This can result in a decrease in communication for some workloads. When False, particles are communicated more frequently. This allows particles to be moved more quickly to other processes, which are potentially idle. Knowing the proper parameter settings for this algorithm is difficult to know for a given workload.

3.2. Training Data Corpus

To collect data for training our machine learning models, we ran a total of 2513 different workloads of the particle advection algorithm. Each run considered was characterized by 4 workload features: flow field, number of particles, seeding strategy, and number of advection steps; and 2 algorithm features: Batch Size and Delay Send. For each workload feature, we ran the algorithm using the default parameter settings to collect a baseline performance characteristics. We then varied the algorithm features for each workload to collect data which could be compared against the baseline. The features we used are described below.

- **Flow Field:** We considered three datasets that have different types of flow characteristics. We chose these datasets as they span a range of scientific use cases and provide insight for the behavior of the algorithm. The first dataset is an astrophysics simulation of the magnetic field surrounding a solar core collapse resulting in a super nova. The vector field used is from the GenASiS code, a multi-physics code being developed and used for the simulation of astrophysical systems involving nuclear matter [ECBM12]. The second is a fusion simulation of the magnetically confined fusion in a tokamak device using the NIMROD code [SGG*04]. This dataset has the unusual properly that most of the magnetic field lines are approximately closed and repeatedly follow the toroidal vector field domain. The third is a thermal hydraulics simulation using the Nek5000 code [FLPS08]. This dataset consists of a vessel with two inlets where hot and cold air are pumped, and a single outlet where the air exists. Each dataset was uniform grid of size 2048^3 and decomposed into 64 spatial blocks, each with a size of 512^3.
- **Number of Particles:** The number of particles ranged in value from 1000 to 500 million.
- **Number of Advection Steps:** The number of advection steps ranged from 100 to 10,000.

- **Seeding Strategy:** Two seeding strategies were considered: sparse seeding, and dense seeding. In sparse seeding, seeds are randomly distributed throughout the bounding box of the entire domain. This provides an understanding of the entire flow in the flow field. In dense seeding, the seeds are placed in a small region of interest inside the flow field. This provides an understanding of the flow originating in a specific location.
- **Delay Send:** This is a Boolean with values of True and False.
- **Batch Size:** The Batch Size used depended on the number of advection steps in the workload. On average, 4 different Batch Size values were generated per baseline.

3.2.1. Performance Data

The performance data collected for each experiment consisted of four values. Two were time related: the execution time for a run, *Time*, and the sum over all proceeds of time spent in communication, *CommTimeSum*. The final two were metrics computed related to the communication patterns associated with the setting for the Batch Size and Delay Send. The first metric, *SumBatchSends* is the sum over all processes of the number of messages (groups of particles) that were sent. The second metric, *AverageBatchSends* is the average number of messages sent when the GPU is interrupted to inspect the workload. Our training was focused on the optimization of speedup, but these metrics were also used to examine the machine learning techniques' efficacy in learning other performance characteristics.

Given a workload $\vec{\theta}$, batch size B, and delay send D, the speedup is defined as:

$$\text{Speedup}(\vec{\theta}) := \frac{\text{Time for baseline algorithm}(\vec{\theta})}{\text{Time for algorithm}(B, D, \vec{\theta})}$$

4. Our Approach

Our approach has two phases. The first phase creates machine learning models that inform particle advection algorithm performance. The second phase constructs an "oracle" that visualization practitioners can then use to optimize particle advection.

We divide this section into two parts: Section 4.1 defines machine learning models and our oracle, while Section 4.2 describes the workflow we employ to build the models and oracles.

4.1. Definitions

4.1.1. Machine Learning Model

We consider a machine learning model to derive from two components: a data corpus and a machine learning technique.

A data corpus is a set of samples, where each sample is an ordered pair. The abscissa (i.e., the first member of the ordered pair) serves as inputs to the model, and the ordinate (i.e., the second member of the ordered pair) is the output of the model. In the context of this study, the abscissa (input) contains both the particle advection workload characteristics and the algorithm characteristics, and the ordinate (output) defines the execution characteristics that the model predicts. Explicitly, these characteristics are:

- Particle advection workload characteristics (abscissa/input)
  - Flow field
  - Seeding strategy
  - Number of particles
  - Number of advection steps
- Particle advection algorithm characteristics (abscissa/input)
  - Batch size
  - Delay Send
- Execution characteristics (ordinate/output)
  - Speedup (compared to a default batch size)

The values of one sample in the corpus would be of the form:

((FlowField-Value, SeedingStrategy-Value, #ofParticles-Value, #ofAdvectionSteps-Value, BatchSize-Value, DelaySend-Value), (Speedup-Value))

A machine learning (ML) technique trains on a data corpus. During this training, it infers relationships between the input (abscissa of the data corpus) and output (ordinate of the data corpus). After training, the ML technique can predict values, i.e., when given a specific value for the input, it can predict a value for the output. While we refer to this learning and prediction process using the general term of "machine learning," a more precise name of this machine learning process is "supervised regression."

Putting it all together, after a training phase, an ML model takes in particle advection workload characteristics and algorithm characteristics and predicts execution characteristics. For example, for a given number of seeds, steps, batch size, etc., an ML model can predict the speedup.

Looking ahead to our experiments, we considered seven machine learning techniques (see Section 5.1.3). We also considered two data corpora for training purposes: particle advection workloads that included flow field ("flow field sensitive") and those that excluded flow field ("flow field agnostic"). In all, this means we considered 14 types of machine learning models.

4.1.2. Oracle

Our oracle predicts whether speedup of a particle advection algorithm is possible compared to a baseline and, if so, provides the particle advection algorithm characteristics that will achieve maximum speedup. Note that this is subtly different than our ML model:

- For our machine learning model:
  - Input consists of particle advection workload characteristics and algorithm characteristics
  - Output consists of execution characteristics
- For our oracle:
  - Input consists of particle advection workload characteristics
  - Output consists of algorithm characteristics

In our approach, an oracle is constructed from a machine learning model. In essence, an oracle uses the predictions from a machine learning model to decide on the best algorithm characteristics. Explicitly, for a given number of seeds, steps, and particles, an

oracle returns a batch size and delay send value that it believes will provide the optimal performance.

Finally, there is some flexibility in how an oracle utilizes its machine learning model, and we consider two variants (see Section 5.2.2). In all, this means there are 28 possible oracles, one for each of the two oracle variants applied to each of the 14 machine learning models.

4.2. Workflow

This section is divided into two parts, with Section 4.2.1 describing the generation of ML models and Section 4.2.2 describing the generation of oracles. Both sections are reflected in Figure 1, which illustrates the workflow.

4.2.1. Phase 1: Generating a Machine Learning Model

The purpose of the first phase is to generate a good ML model that can be used in the oracle. It operates in two steps.

Step 1

The first step is to generate a diverse set of candidate ML models. This set of machine learning models should draw from different architectures, hyperparameters, and data preparation techniques. Step one is represented in Figure 1 as "Machine Learning Development Workflow."

Many authors have described techniques for developing and applying individual machine learning models to a specific problem. We find the nine-step technique by researchers from Microsoft in [ABB*19] to be particularly informative as a practical framework for constructing individual models.

There are two broad areas of critical importance for the successful modeling of phenomena. These include:

(a) Data collection and preparation of that data into appropriate input for a machine learning technique.
(b) Selection of the hyperparameters and other settings of the machine learning technique itself.

Frequent points of failure in the development of an ML model hinge on inappropriate choices made in (a) or (b).

Step 2

The second step is to assess which candidate ML models — if any — adequately capture particle advection performance. Step two is represented in Figure 1 as "Decision: Assess models."

Once each machine learning model in our set has been trained, it will then be assessed for efficacy in modeling phenomena. Models should be assessed on their own merits (e.g., does model X have high enough accuracy to be useful), as well as evaluated through direct comparison with the other models under assessment (e.g., is model X more accurate than model Y).

The end outcome of assessment determines whether there is a viable model that sufficiently predicts the targeted execution characteristic. Viable models can be used for subsequent inclusion in the creation of an oracle.

Finally, one possibility from this phase is that no ML model will adequately capture particle advection performance. If there is not a viable model, then we iterate through the model creation process again. We would particularly investigate the two common points for machine learning modeling failure: a lack of adequate data integrity and preparation, and insufficient tuning of settings and parameters of the machine learning architecture itself.

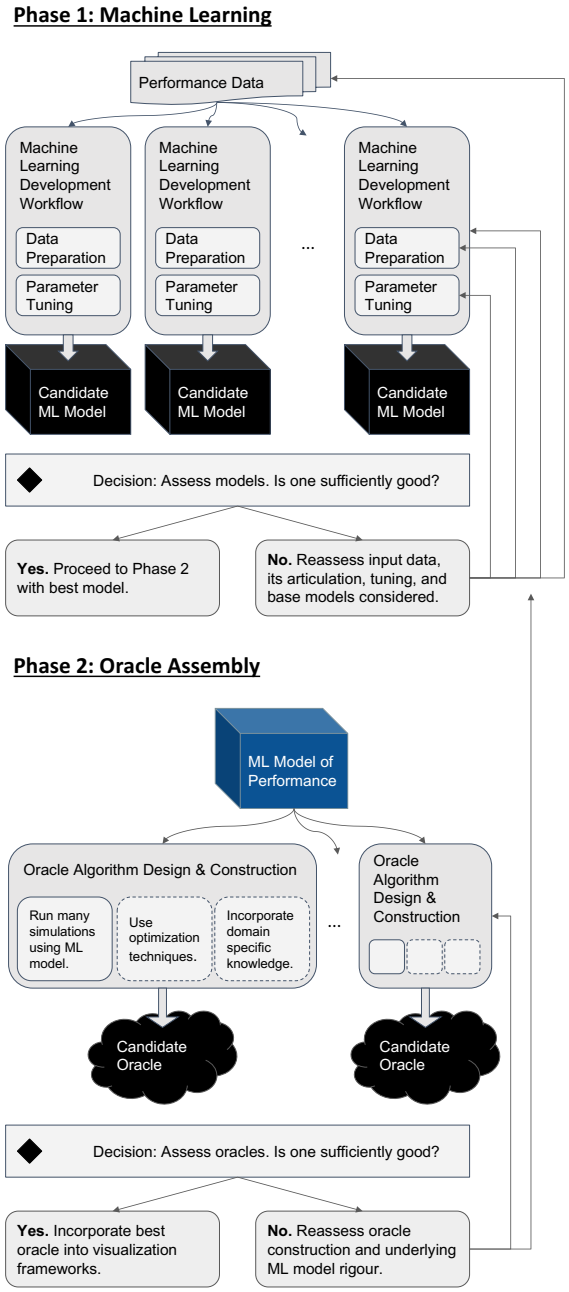

Figure 1: *Flowchart describing our approach for generating an oracle to optimize particle advection performance.*

4.2.2. Phase 2: Generating an Oracle

Oracle development, like machine learning model development, consists of two steps: generation and assessment.

Step 1

The first step of oracle development is to construct variant algorithms. Step one is represented in Figure 1 as "Oracle Algorithm Design & Construction." All oracle generation techniques which use our approach will utilize the predictive power of the ML models generated in Phase 1 to simulate many runs of our particle advection algorithms; far more runs than were provided to train our ML models in the first place. Oracles may optionally include optimization processes in their designs. Oracles also may optionally incorporate decision logic predicated on domain specific knowledge about the behavior of the algorithm; knowledge which may have only been implicitly learned by the ML model but is explicitly known by human developers.

Looking ahead to Section 5.2.2 we construct two oracle variants. Both of these oracle variants use a naive optimization technique, wherein we search over the ML model to quickly determine the algorithm characteristics which maximizes speedup across a wide swath of parameters. One oracle variant also incorporates our domain specific knowledge about the underlying behavior of the particle advection algorithm in its analysis.

Step 2

Once an oracle is constructed it will then be assessed for efficacy in predicting performance characteristics. This final step of our last phase is represented in Figure 1 as "Decision: Assess Oracles." Oracles that predict execution and algorithm characteristics when given workload characteristics should be assessed against the true execution characteristics of runs defined by those workload and algorithm characteristics. As with machine learning models, oracles should also be evaluated through direct comparison with the other oracles under assessment (e.g., is oracle X more accurate than oracle Y).

5. Experimental Overview

This section provides an overview of our experiments. We performed two sets of experiments, corresponding to the two phases of our approach. The experiments for Phase 1, described in Section 5.1, considered 14 machine learning models, and evaluated them for accuracy. The most accurate models were then used as input for the experiments for Phase 2, which are described in Section 5.2. These experiments considered oracles constructed from each of the most accurate models, and evaluated the extent to which they can improve particle advection performance.

5.1. Experimental Overview for Phase 1

This subsection divides the experimental overview of Phase 1 into four parts: description of the input data (Section 5.1.1), data preparation (Section 5.1.2), machine learning techniques (Section 5.1.3), and model assessment (Section 5.1.4).

5.1.1. Input Data

The input data corpus to Phase 1 of our approach was the performance data collect from running our particle advection algorithm. This corpus was described in section 3.2, and consisted of 2513 runs with varied workloads and non-default particle advection algorithm parameters compared with a baseline. Each run became one sample, meaning that each machine learning technique had 2513 samples to train on.

5.1.2. Data Preparation

This section describes how we transformed the performance data from our data corpus to a form that can be used by machine learning frameworks.

One important consideration from the preparation was whether to include information about the flow field in the training. (Note that we do not mean specific values from the flow field, but rather the identity of the overall flow field itself.) On the one hand, including flow field information would allow for models to capture flow field-specific behaviors and thus provide further optimization. On the other hand, generic, flow field-agnostic models are much more useful in practice. For our experiments, we decided to do both. In practice, this involved making a duplicate of our data corpus, and removing flow field information from the duplicate. We refer to the original data corpus as "flow field sensitive" and the modified version as "flow field agnostic." An important byproduct of this decision was that it doubled the number of machine learning techniques we needed to consider — for each technique we would train one instance with the "flow field sensitive" corpus and another instance with the "flow field agnostic" corpus.

The remainder of this section describes how we mapped a data corpus to a form that can be ingested by machine learning frameworks. The details here are not critical for understanding our approach and results, but we feel it is important to describe them to aid in reproducibility.

At a high level, the machine learning techniques we consider ingest data as single vectors, and so the challenge of this step was mapping performance data from each of our runs to this form (i.e., a single vector). For our experiments, we made the following transformations to create these vectors:

- Seeding Strategy and Delay-Send had two possible values and were encoded as 0 or 1 in the vector.
- Number of Advection Steps, Number of Particles, and Batch Size all have integers values. For each, their values were subjected to a natural logarithm, and then underwent linear transformation and scaling to obtain a mean of 0 and unit variance.
- When Flow Field was used, it was treated as a categorical feature — Fishtank $\mapsto$ 0, Fusion $\mapsto$ 1, and Astro $\mapsto$ 2.

Categorical inputs (i.e., Flow Field and Seeding Strategy) were one hot-encoded, which can improve model accuracy for many machine learning techniques. Finally, once the model was trained, new inputs underwent the same transformations as samples from the training data.

5.1.3. Machine Learning Techniques

The seven machine learning techniques we considered included three neural network variants and four classical machine learning models. Between these seven models and two data corpora ("flow field sensitive" and "flow field agnostic"), we considered fourteen machine learning models overall.

In the description of the seven models, we use the following notation: Let $\vec{x} \in \mathbb{R}^n$ denote an individual sample that is input to a machine learning model and let $\vec{y} \in \mathbb{R}^m$ denote the output of this model for an individual sample. Further, let X denote the set of inputs for all the samples used in training.

Neural Network Models:

All three of our neural network models shared the following characteristics:

- The model is a fully connected, feed-forward neural network.
- The activation function for the final layer is the identity function.
- Dropout is not incorporated at any point.
- The optimization algorithm used for training is Hinton's RMS Propagation [HSS12].
- The loss function is mean squared error.
- "Training batch size" (which we note here as a characteristic of our neural network's algorithmic framework and not the same as "batch size" as described in our data corpus) for the neural network was chosen to be $\lfloor\sqrt{|X|}\rfloor$.
- The number of epochs used for training was selected to be 250 for our "flow field agnostic" corpus and 500 for our "flow field sensitive" corpus. This selection was due to observed convergence of the loss function at these configurations.

The neural network models only differed in the composition of their hidden layers, all of which used Rectified Linear Units (ReLUs).

- **Neural Network A** (Model A): This model contains one hidden ReLU layer. The size of this layer is $\lceil\frac{1}{2}\cdot(n+m)\rceil$.
- **Neural Network B** (Model B): This model contains two hidden ReLU layers. The layers are of size: *input* $(n) \rightarrow \lceil\frac{2}{3}\cdot(n+m)\rceil \rightarrow \lceil\frac{1}{3}\cdot(n+m)\rceil \rightarrow$ *output* (m).
- **Neural Network C** (Model C): This model contains one hidden ReLU layer. The size of the layer is $\lceil\frac{1}{2}\cdot(n+m)\rceil^2$.

Classic Machine Learning Models: Our four classic regression models were:

- **Linear Regression** (Model D): We fit a linear model using ordinary least squares fitting. We expect this model to perform poorly, but use it as a point of comparison given its simplicity, speed, and widespread popularity.
- **Random Forest** (Model E): A forest of 100 regression trees, each using Gini impurity as its discrimination function. The forest included bootstrap sampling to build its trees, and used mean squared error as its discriminating criterion. Each tree was fully expanded until each leaf node was pure.
- **Support Vector Machine** (Model F): Support Vector Regression using a radial basis kernel was used. Gamma was selected to be $\frac{1}{n\cdot \mathrm{Var}(X)}$.
- **K Nearest Neighbors** (Model G): A k Nearest Neighbors model was used, where the number of neighbors = $\lfloor\sqrt{|X|}\rfloor$.

Implementation Details: The neural networks, models A, B, and C were implemented in Keras [C*15] with a Tensorflow [AAB*15] backend. Models D, E, F, and G were implemented with Scikit-Learn [PVG*11]. Unless otherwise specified, the default parameters for the machine learning algorithms implemented in these libraries were utilized.

5.1.4. Model Assessment

We used 10-fold cross-validation [Sto74] to assess the accuracy and robustness of a specific machine learning technique to model speedup.We recorded the true output versus the predicted output of the testing data assessed across each of the 10 folds. From this predicted versus truth output, we derived various metrics.

The criteria we considered when evaluating the machine learning models trained to predict speedup included: error, explained variance, and efficacy of the underlying machine learning technique when the same base architecture was retrained to model and predict other execution characteristics.

5.2. Experimental Overview for Phase 2

This subsection divides the experimental overview of Phase 2 into three parts: construction of a lookup table for optimal performance (Section 5.2.1), definition of our oracle variants (Section 5.2.2), and information on our verification runs (Section 5.2.3).

5.2.1. Lookup Table Construction

In an offline process, we calculated the optimal algorithm characteristics for a set of workload characteristics by using our selected machine learning model. This information was stored in a lookup table that could be used subsequently by our oracles. The oracles were then able to specify workload characteristics and receive the algorithm characteristics that provided the maximum predicted speedup. Since this was done as an offline process, the time to calculate the lookup table did not affect algorithm performance. If this research is adopted in the future, this offline process (which takes a few hours) would be done whenever a new supercomputer is stood up, and the resulting lookup table would be compiled into production visualization software.

For a given workload, the optimal algorithm characteristics were calculated in a brute force manner. We selected a set of $\approx$ 25,000 possible values for algorithm characteristics and we ran our ML model for each of them to see the predicted speedup. Whichever settings for algorithm characteristics had the highest speedup was the recommended value for that workload and was the value stored in the lookup table.

The $\approx$ 25,000 possible values spanned both batch size and "delay send." For batch size, the values were selected in a logarithmically-spaced manner from 2 to 100,000,000. For "delay send," the values were 0 and 1. The final set of values were the Cartesian product of options for batch size and "delay send."

5.2.2. Oracles Considered

We consider two variants of oracles. In both cases, the oracles take workload characteristics and produce the algorithm characteristics that are predicted to achieve maximum speedup. Further, the oracles rely heavily on the lookup table from Section 5.2.1, and modify its behavior only slightly. One important modification for the oracles involves the default configuration — the default configuration of "batch size == all" is actually a good setting in many cases, and so setting the batch size to a finite value will lead to slowdowns, not speedups. In this case, we want the oracles to recommend the default algorithm.

We define our oracle variants as follows. Let W be workload characteristics, LUT be the lookup table from Section 5.2.1, and A be the answer returned by the lookup table, i.e., $A = LUT(W)$. Further, assume the attributes of characteristics are accessible. Specifically, let $A.BATCH\_SIZE$ be the recommended batch size for A, $A.SPEEDUP$ be the predicted speedup for A, and $W.ADV\_STEPS$ be the number of advection steps for W. Finally, denote the algorithm characteristics for default execution as $DEFAULT$.

Algorithm 1: Oracle Variant I

```
A=LUT(W)
if A.SPEEDUP > 1 then
    return A
else
    return DEFAULT
```

Algorithm 2: Oracle Variant II

```
A=LUT(W)
if A.SPEEDUP > 1 and A.BATCH_SIZE <
  W.ADV_STEPS then
    return A
else
    return DEFAULT
```

Summarizing, both variants capture the desired behavior of using the default mode if no speedups are possible. The distinguishing component to Oracle Variant II is that it also checks to see if the batch size is less than the number of advection steps. This is because the batch size limits how many steps can be taken. If that batch size is larger than the number of advection steps, then imposing this limit just causes an overhead with no benefit.

5.2.3. External Verification Runs

Once each model was assessed and the best performing selected, we trained this best-performing model on the full data corpus of 2513 samples.

We then used the Cartesian product of the below features to construct runs:

- Number of particles $\in$ {1000, 10000, 100000, 1000000, 10000000, 100000000}
- Number of steps $\in$ {100, 1000, 10000}
- Seeding strategy $\in$ {sparse, whole}
- Flow Field $\in$ {Fishtank, Fusion, Astro}

This Cartesian product of features resulted in 108 runs for us to evaluate.

These 108 runs, with batch size and delay send selected for a theoretically optimal speedup, were then processed and their true real-world speedups calculated. We thresholded by runs which had corresponding baselines complete within a two hour time frame. We analyze these results in the context of their usage in an oracle in Section 6.

5.3. Hardware Used

The particle advection algorithm was implemented in VTK-m [MSU*16, PYK*18] and run using 64 GPUs on the Summit supercomputer at Oak Ridge National Laboratory [VdB*18]. Each Summit node consists of 6 NVIDIA Volta V100 GPUs, 2 POWER9 CPUs, 608 GB of RAM and connected with a Mellanox EDR 100G InfiniBand.

A personal computer, an Acer Aspire E 15 running on an Intel Core i7-6500U CPU at 2.50GHz-2.60GHz with 32 GB of RAM installed, was used for the machine learning model training and subsequent batch size optimization.

6. Results and Analysis

In this section we return to our research question, and demonstrate how machine learning can be used to accelerate the performance of a particle advection algorithm. This section provides results and analysis for the approach that was outlined in Section 4. In Section 6.1 we describe the results from the assessment step of Phase 1 and show the efficacy for each of the machine learning models that were considered. In Section 6.2, we describe results from the assessment step of Phase 2 where we used verification runs to select an oracle. Finally, in Section 6.3 we present the speedups that were achieved by our oracle for the particle advection algorithm.

6.1. Phase 1: Machine Learning Model Selection

Selection of the best machine learning models was driven by identifying models with the lowest mean absolute error and unexplained variance. Absolute error is defined as |truth - prediction|. Unexplained variance is a measure of the degree to which our model fails to explain variation in the data. These metrics provide evidence that the model has a good understanding of the relationship between a model's inputs and outputs and can therefore provide useful predictions.

Figure 2 plots the mean absolute error and unexplained variance for each of the seven models on both data corpora. This analysis concluded that the most effective model for the flow field agnostic corpus is Neural Network C. We made this conclusion by noting that it has nearly the lowest mean absolute error of all the models considered and the most explained variance. Unlike Neural Network B, however, Neural Network C consistently preforms well when trained on performance characteristics besides Speedup. This consistency led us to conclude that the Neural Network C model would be a better candidate for subsequent inclusion into

an oracle. We also concluded that the most effective model for the flow field sensitive corpus is our Random Forest model. This model clearly has the lowest error and unexplained variance. Random Forest models also consistently had the lowest error and unexplained variance when trained on performance characteristics besides Speedup.

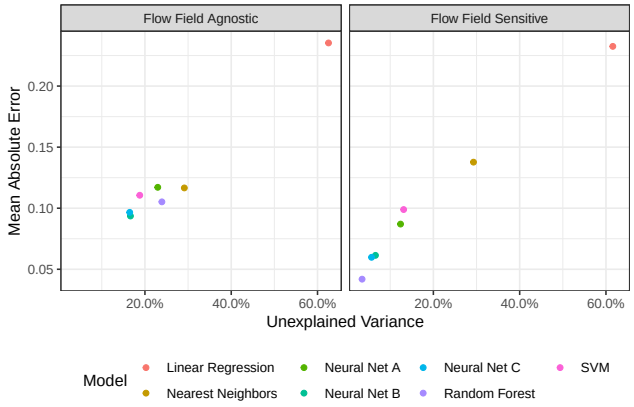

Figure 2: *Plots showing the performance of each model with respect to unexplained variance and mean absolute error. Plots for both flow field agnostic and flow field sensitive corpora are shown. Models closer to (0,0) are better.*

6.2. Phase 2: Oracle Selection

We used the models selected in Phase 1 as a starting point for selection of an oracle. For brevity, we term the flow field agnostic model, Neural Network C, *NN C*, and the flow field sensitive model, Random Forest, *RF*. We evaluated 108 validation runs (described in Section 5.2.3) optimized over 12,500 potential Batch Sizes and the two Delay Send options. The {Batch Size, Delay Send} combination with the highest corresponding speedup predicted by NN C, and again by RF was returned. This optimization is a one time run cost and the results can be stored in a lookup table.

We considered two oracle variants, "Oracle I" and "Oracle II," described in Section 5.2.2. The Cartesian product of these two oracles and two models results in four oracle-model configurations. We assess the error of these four oracle-models and the speedup gained by applying them.

6.2.1. Prediction Accuracy

In this subsection, we evaluate the accuracy of our predictions. We define "error" in this context to be the difference between our predicted speedup and the actual speedup. Positive values mean we predicted a larger speedup than was achieved, while negative values mean that the speedup was larger than anticipated. Further, we also consider the absolute value of error, which captures the extent to which our prediction was wrong. Finally, while we report outlier behavior (i.e., how much faster or slower a workload performed than our prediction), we feel the most important aspect is mean behavior (i.e., how much compute time is saved in aggregate by using the predicted best settings on a set of workloads).

Figure 3 shows the actual, observed speedups against each configuration's predicted speedup across all runs that were considered "acceleratable" by that oracle-model.

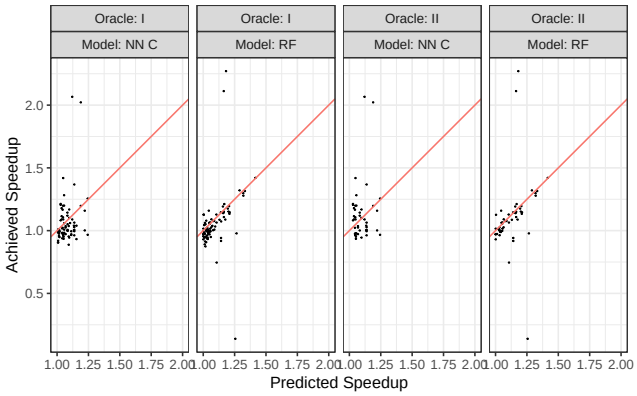

Figure 3: *Actually achieved speedup compared to the predicted speedup of the runs the given oracle-model configuration considered to be "acceleratable." The line is perfect accuracy: predicted speedup = actual speedup.*

Table 1 lists information about the error for each of the four oracle-model configurations. The distribution of errors are shown in Figure 4.

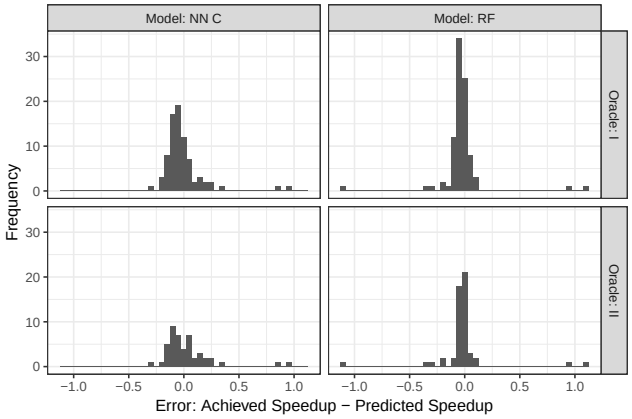

Figure 4: *Histogram of the errors for each of the four oracle-model configuration.*

One observation is that the NN C-based configurations do not have extreme outliers of low actual, observed speedups. While there are runs which underperformed, NN C - based configurations never resulted in runs which had significant slowdowns (0.5

Oracle Variant: ML Model:	I NN C	II NN C	I RF	II RF
Min |Error|	0.001	0.004	0.002	0.001
Mean |Error|	0.111	0.141	0.091	0.110
Max |Error|	0.945	0.945	1.118	1.118

Table 1: *Summary of absolute errors for each of the four oracle-model configurations.*

Oracle Variant: Machine Learning Model:	I NN C	II NN C	I RF	II RF
"Acceleratable" runs	76.70%	44.66%	85.71%	50.48%
"Acceleratable" runs that achieved speedup	56.96%	73.91%	61.11%	75.47%

Table 2: *Prediction percentages for each of the four oracle-model configurations.*

of baseline or lower) like the RF - based configurations did. This insight is tempered by the observation that the Variant II, NN C oracle-model configuration also has the highest mean absolute error of the configurations considered. We also note, however, that the mean error of this configuration is positive, whereas all other configurations are negative. This indicates that once the Variant II, NN C oracle-model configuration identifies a run to be acceleratable, it is more likely than not to have a higher achieved speedup than its predicted speedup.

6.3. Speedup Results

To test our research question on the four oracle-model configurations that were created, we performed runs of 108 different workloads, as discussed previously in Section 5.2.3.

Each of our four oracle-model configurations considered 108 runs with varying workload characteristics, and is used to predict the algorithm parameters for a given workload. Given a workload, if the model predicts that a speedup over the base line is possible, the settings for Batch Size and Delay Send are predicted that will maximize the speedup.

We call a run "acceleratable" if an oracle-model configuration predicts that a speedup greater than one is possible. We validated that each "acceleratable" run actually achieved speedup over the baseline. Some of our runs had workload characteristics which prevented the baseline algorithm from completing within a two-hour timeout window. These workload characteristics always involved many particles and many advection steps. These timeout cases constituted fewer than 5% of our validation runs and are excluded from further analysis.

Each validation run contains the following attributes:

- Workload characteristics (e.g., number of particles, number of advection steps, etc).
- Algorithm characteristics predicted by the oracle-model configuration.
- Speedup predicted by the oracle-model configuration.
- Speedup actually obtained.

The rates at which each oracle-model predict an "acceleratable" workload, and the rate at which acceleration is achieved is shown in Table 2. Oracle Variant II is more hawkish in its predictions of speedup for both NN C and RF models. However, when a prediction of "acceleratable is made, Variant II is much more accurate than Variant I. This is true for both NN C and RF models.

When an oracle-model configuration predicted a workload could achieve speedup, we validated it by running the workload with the algorithm parameters given. The results from these run are shown in Table 3 for each of the four oracle-model configurations. This table shows the minimum, maximum and mean speedup over all runs. It also shows results for use cases where execution times are greater than 5, 10 and 60 seconds. Plots of these speedups are shown for all four oracle-model configurations in Figure 5.

Oracle Variant: Machine Learning Model:	I NN C	II NN C	I RF	II RF
Maximum Speedup Achieved	2.066	2.066	2.271	2.271
Mean Speedup Achieved	1.067	1.127	1.058	1.107
Mean Speedup Achieved; Execution Time $>$ 5 seconds	1.131	1.149	1.134	1.134
Mean Speedup Achieved; Execution Time $>$ 10 seconds	1.158	1.173	1.156	1.156
Mean Speedup Achieved; Execution Time $>$ 60 seconds	1.203	1.203	1.171	1.171
Minimum Speedup Achieved	0.888	0.935	0.138	0.138

Table 3: *Speedups of workloads considered "acceleratable" when run with the particle advection parameters provided by the given oracle-model configuration.*

Using these data, we find that the oracle-model configuration Oracle Variant II – NN C consistently provided the highest mean speedups when compared to our other configurations. On average, it provided a 1.12X speedup across all workloads. As the execution time of the algorithm increases (i.e., larger workloads), the speedup achieved rises up to 1.203X when the execution time is above 60 seconds. This configuration also had a minimum speedup achieved of 0.935X, which was the best lower bound among all configurations. The Variant II – RF configuration had similar, although lower average speedups, and higher maximum speedups, but it also made poor predictions that resulted in very poor minimum speedups of 0.138X. Paired with the knowledge that this oracle-model configuration also has the highest precision of all the configurations considered, we conclude that Oracle Variant II undergirded by Neural Network C (flow field agnostic version) is the best predictor of speedups and algorithm settings of all the options we considered.

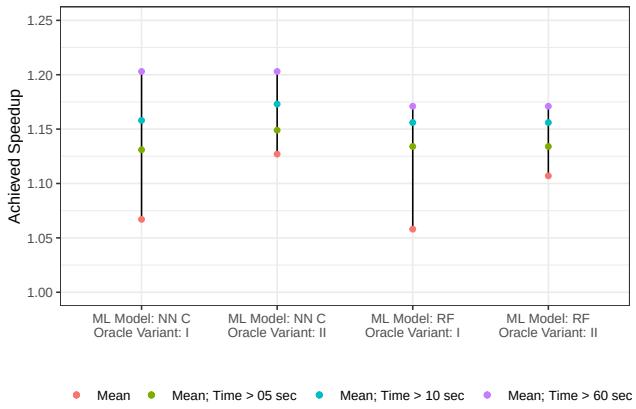

Figure 5: *Mean speedup of workloads that were considered "acceleratable" and used the particle advection algorithm parameters provided by the given oracle-model configuration.*

7. Conclusion and Future Work

This study was designed to answer the question of whether or not machine learning could be used as a tool to optimize the performance of a parallel particle advection algorithm. To probe this question we first generated a large collection of performance data from the algorithm with varying workloads and algorithm parameter settings. This collection of data serves as input to our two phased approach for the design of a machine learning based oracle for accelerating the algorithm. In phase 1 of our approach, a series of machine learning models are generated and their performance is assessed. Once suitable models are found, a set of oracles is built and then assessed. We then ran the algorithm with a number of workloads and used the oracle to predict algorithm parameters that would provide the best speedup. For workloads where speedup is possible, the oracle we describe in Section 6.3 was able to improve the speed over all workloads by an average of 12.7%. For larger workloads that take more time to complete, the oracle provided an average speedup of 20.3%.

In the future, we would like to apply this approach to algorithm acceleration to other visualization algorithms. We are also interested in further study of the neural network used in our oracle, and whether expanding its architecture (e.g., more nodes per layer, additional layers) improves oracle efficacy.

References

[AAB*15] ABADI M., AGARWAL A., BARHAM P., BREVDO E., CHEN Z., CITRO C., CORRADO G. S., DAVIS A., DEAN J., DEVIN M., GHEMAWAT S., GOODFELLOW I., HARP A., IRVING G., ISARD M., JIA Y., JOZEFOWICZ R., KAISER L., KUDLUR M., LEVENBERG J., MANÉ D., MONGA R., MOORE S., MURRAY D., OLAH C., SCHUSTER M., SHLENS J., STEINER B., SUTSKEVER I., TALWAR K., TUCKER P., VANHOUCKE V., VASUDEVAN V., VIÉGAS F., VINYALS O., WARDEN P., WATTENBERG M., WICKE M., YU Y., ZHENG X.: TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. URL: http://tensorflow.org/. 7

[ABB*19] AMERSHI S., BEGEL A., BIRD C., DELINE R., GALL H., KAMAR E., NAGAPPAN N., NUSHI B., ZIMMERMANN T.: Software engineering for machine learning: A case study. In *International Conference on Software Engineering (ICSE 2019) - Software Engineering in Practice track* (May 2019), IEEE Computer Society. URL: https://www.microsoft.com/en-us/research/publication/software-engineering-for-machine-learning-a-case-study/. 5

[BDG*18] BALAPRAKASH P., DONGARRA J., GAMBLIN T., HALL M., HOLLINGSWORTH J. K., NORRIS B., VUDUC R.: Autotuning in high-performance computing applications. *Proceedings of the IEEE 106*, 11 (2018), 2068–2083. 2

[Bis06] BISHOP C. M.: *Pattern Recognition and Machine Learning*. Springer, 2006. 2

[C*15] CHOLLET F., ET AL.: Keras. https://keras.io, 2015. 7

[CBP*14] CHILDS H., BIERSDORFF S., POLIAKOFF D., CAMP D., MALONY A. D.: Particle Advection Performance over Varied Architectures and Workloads. In *IEEE International Conference on High Performance Computing (HiPC)* (Goa, India, Dec. 2014), pp. 1–10. 2

[CGC*11] CAMP D., GARTH C., CHILDS H., PUGMIRE D., JOY K. I.: Streamline Integration Using MPI-Hybrid Parallelism on a Large Multicore Architecture. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 17*, 11 (Nov. 2011), 1702–1713. doi:http://doi.ieeecomputersociety.org/10.1109/TVCG.2010.259. 2

[CKP*13] CAMP D., KRISHNAN H., PUGMIRE D., GARTH C., JOHNSON I., BETHEL E. W., JOY K. I., CHILDS H.: GPU Acceleration of Particle Advection Workloads in a Parallel, Distributed Memory Setting. In *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)* (Girona, Spain, May 2013), pp. 1–8. 2

[ECBM12] ENDEVE E., CARDALL C. Y., BUDIARDJA R. D., MEZZACAPPA A.: Turbulent magnetic field amplification from spiral SASI modes in core-collapse supernovae. *J. Phys. Conf. Ser. 402* (2012), 012027. arXiv:1203.3385, doi:10.1088/1742-6596/402/1/012027. 3

[FLPS08] FISCHER P., LOTTES J., POINTER W., SIEGEL A.: Petascale algorithms for reactor hydrodynamics. *Journal of Physics: Conference Series 125* (08 2008), 012076. doi:10.1088/1742-6596/125/1/012076. 3

[GBC16] GOODFELLOW I., BENGIO Y., COURVILLE A.: *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 2

[HSS12] HINTON G., SRIVASTAVA N., SWERSKY K.: Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Neural networks for machine learning* (2012). 7

[HTF09] HASTIE T., TIBSHIRANI R., FRIEDMAN J. H.: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition.* Springer Series in Statistics. Springer, 2009. 2

[MPP*18] MOIZ A. A., PAL P., PROBST D., PEI Y., ZHANG Y., SOM S., KODAVASAL J.: A machine learning-genetic algorithm (ml-ga) approach for rapid optimization using high-performance computing. *SAE Int. J. Commer. Veh. 11* (04 2018), 291–306. URL: https://doi.org/10.4271/2018-01-0190, doi:10.4271/2018-01-0190. 2

[MSU*16] MORELAND K., SEWELL C., USHER W., LO L., MEREDITH J., PUGMIRE D., KRESS J., SCHROOTS H., MA K.-L., CHILDS H., LARSEN M., CHEN C.-M., MAYNARD R., GEVECI B.: VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. *IEEE Computer Graphics and Applications (CG&A) 36*, 3 (May/June 2016), 48–58. 2, 8

[PPG12] PUGMIRE D., PETERKA T., GARTH C.: Parallel integral curves. In *High Performance Visualization: Enabling Extreme Scale Scientific Insight*, Bethel E. W., Childs H., Hansen C., (Eds.). CRC Press, 2012. 2, 3

[PRN*11] PETERKA T., ROSS R., NOUANESENGSY B., LEE T., SHEN

H., KENDALL W., HUANG J.: A study of parallel particle tracing for steady-state and time-varying flow fields. In *2011 IEEE International Parallel Distributed Processing Symposium* (2011), pp. 580–591. 2

[PVG*11] PEDREGOSA F., VAROQUAUX G., GRAMFORT A., MICHEL V., THIRION B., GRISEL O., BLONDEL M., PRETTENHOFER P., WEISS R., DUBOURG V., VANDERPLAS J., PASSOS A., COURNAPEAU D., BRUCHER M., PERROT M., DUCHESNAY E.: Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830. 7

[PYK*18] PUGMIRE D., YENPURE A., KIM M., KRESS J., MAYNARD R., CHILDS H., HENTSCHEL B.: Performance-Portable Particle Advection with VTK-m. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)* (Brno, Czech Republic, June 2018), pp. 45–55. 2, 8

[SGG*04] SOVINEC C., GLASSER A., GIANAKON T., BARNES D., NEBEL R., KRUGER S., PLIMPTON S., TARDITI A., CHU M., THE NIMROD TEAM: Nonlinear magnetohydrodynamics with high-order finite elements. *J. Comp. Phys. 195* (2004), 355. 3

[SP16] SISNEROS R., PUGMIRE D.: Tuned to terrible: A study of parallel particle advection state of the practice. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2016), pp. 1058–1067. 2

[Sto74] STONE M.: Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological) 36*, 2 (1974), 111–133. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1974.tb00994.x, arXiv:https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1974.tb00994.x, doi:10.1111/j.2517-6161.1974.tb00994.x. 7

[TAZ*17] TUNCER O., ATES E., ZHANG Y., TURK A., BRANDT J., LEUNG V. J., EGELE M., COSKUN A. K.: Diagnosing performance variations in hpc applications using machine learning. In *High Performance Computing* (Cham, 2017), Kunkel J. M., Yokota R., Balaji P., Keyes D., (Eds.), Springer International Publishing, pp. 355–373. 2

[VdB*18] VAZHKUDAI S. S., DE SUPINSKI B. R., BLAND A. S., GEIST A., SEXTON J., KAHLE J., ZIMMER C. J., ATCHLEY S., ORAL S., MAXWELL D. E., LARREA V. G. V., BERTSCH A., GOLDSTONE R., JOUBERT W., CHAMBREAU C., APPELHANS D., BLACKMORE R., CASSES B., CHOCHIA G., DAVISON G., EZELL M. A., GOODING T., GONSIOROWSKI E., GRINBERG L., HANSON B., HARTNER B., KARLIN I., LEININGER M. L., LEVERMAN D., MARROQUIN C., MOODY A., OHMACHT M., PANKAJAKSHAN R., PIZZANO F., ROGERS J. H., ROSENBURG B., SCHMIDT D., SHANKAR M., WANG F., WATSON P., WALKUP B., WEEMS L. D., YIN J.: The design, deployment, and evaluation of the coral pre-exascale systems. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis* (2018), pp. 661–672. 8