Evaluating Active Learning with Cost and Memory Awareness

Dmitry Duplyakin
School of Computing
University of Utah
Salt Lake City, UT 84112, USA
dmitry.duplyakin@utah.edu

Jed Brown
Department of Computer Science
University of Colorado
Boulder, CO 80309, USA
jed.brown@colorado.edu

Donna Calhoun
Department of Mathematics
Boise State University
Boise, ID 83725 USA
donnacalhoun@boisestate.edu

***Abstract*—Active Learning (AL) is a methodology from Machine Learning and Design of Experiments (DOE) in which the quantities of interest are measured sequentially and the corresponding surrogate models are constructed incrementally. AL provides compelling optimizations over static DOE in applications with engineering processes where the cost of individual experiments is significant. It also helps perform series of computer experiments in parameter sweeps and performance analysis studies. One of the non-trivial tasks in the design of AL systems is the selection of algorithms for cost-efficient exploration of the input spaces of interest: AL needs to balance "exploitation" of experiments with modest costs and careful "exploration" of expensive configurations. Finding this balance in an automatic and general manner is challenging yet desirable in practice.**

In this paper, we investigate the application of AL algorithms to Adaptive Mesh Refinement (AMR) performed on a supercomputer. We use AL in conjunction with Gaussian Process Regression for the incremental modeling of cost and memory usage of a series of AMR simulations of a shock-bubble interaction phenomenon. In the studied 5-dimensional input parameter space – with physical, numerical, and machine parameters – we allow AL to guide experimentation across hundreds of configurations. We develop and evaluate a novel multi-objective AL experiment selection algorithm which prioritizes cost-efficient exploration of available configurations and at the same time avoids simulations that violate memory constraints.

***Index Terms*—Performance Modeling, Active Learning, Gaussian Process Regression, Adaptive Mesh Refinement**

I. Introduction

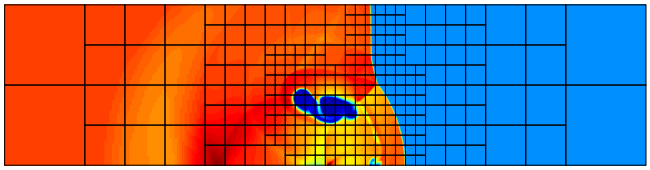
First introduced in the 1980s, Adaptive Mesh Refinement (AMR) is a popular technique in many areas of modern science and engineering. AMR refers to the methods which increase the resolution of the simulation in the regions of the computational domain where the solution exhibits features of interest. In a common approach to AMR, the computational domain is decomposed into logically Cartesian regions consisting of coarse (lower resolution), or fine (higher resolution) meshes. Depending on the implementation, finer regions may cover portions of the coarser grids (patch-based AMR). Alternatively, the decomposition may be a true partitioning of the domain (tree-based AMR). Simulations based on AMR typically see a significant reduction in computational and storage requirements when compared with solutions obtained on uniformly high-resolution meshes.

With AMR, it is difficult to predict how much refinement will be applied in a particular simulation. User-defined criteria for refinement may result in drastically varying simulation runtimes and memory usage. Even experienced users are likely to struggle with choosing the sufficient amount of computational resources that will allow them to meet their computing deadlines. As shown in Fig. 1, enabling additional levels of refinement in AMR reveals finer features of the simulated phenomenon. At the same time, the number of refinement levels is a factor that strongly impacts performance yet the exact corresponding growth of computational demands is difficult to predict. Moreover, when we vary one of many physical parameters – in the shock-bubble interaction, it could be viscosity, size and density of the bubble, change of pressure at the shock front, among others – we also observe drastic changes in performance characteristics.

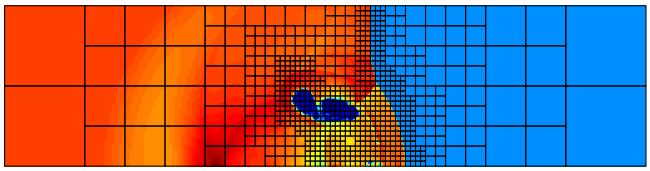
In real-world applications, AMR simulations run for significant periods of time even on supercomputers and computing clusters. Libraries such as MPI [1] and OpenMP [2] power many AMR libraries and packages by accelerating these computations via distributed-memory and shared-memory parallelism, respectively. However, as the number of machines simultaneously used for these computations increases, the cost of these computational experiments, typically expressed in node-hours or core-hours, also grows. Often, the total cost of desired series of AMR simulations exceeds even large supercomputer allocations and becomes impractical. In our modeling and the proposed DOE, we demonstrate how the magnitude and the variability of performance characteristics can be considered and describe the practical approach to selection of the most informative and cost-efficient simulations. Our approach aims to avoid running overly expensive computations if their performance characteristics can be predicted with sufficient accuracy. This optimization is vital in applications with modest time-to-solution and total compute budget constraints.

We extend our previous work described in [3]. We use a combination of Gaussian Process Regression (GPR) [4] and Active Learning (AL) [5] as a method that allows us to obtain high-confidence predictions across a large input space without the need for a static, and most likely inefficient, experiment design. In this paper, we describe how we use this approach and leverage the tools we have developed in the context of predictive

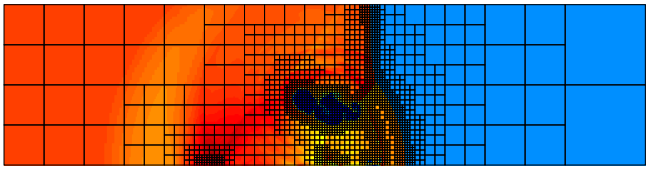
Fig. 1: Visualization of a 2D shock-bubble interaction problem simulated using FORESTCLAW package.



(a) AMR simulation with up to 4 levels of mesh refinement.



(b) AMR simulation with up to 5 levels of mesh refinement.



(c) AMR simulation with up to 6 levels of mesh refinement.

modeling in a 5-dimensional input parameter space. These input parameters, or *features*, are the number of processors used, discretization box size, maximum level of refinement, bubble size, and bubble density (discussed in detail in Section IV-A). The last two are physical properties of the simulated shock-bubble interaction phenomenon, whereas the others represent numerical and machine properties. We analyze the simulation cost and memory usage functions, or *responses*, which exhibit rapid, unpredictable growth along multiple dimensions in the analyzed input space. This growth occurs only at particular combinations of the selected parameters. In the rest of this input space, the responses change gradually or remain flat and can be predicted accurately with little experimental data.

Practical AL for supercomputer experiments needs to not only provide computational scientists with the cost-efficient iterative DOE but also should mitigate the risk of selecting simulations that will likely run out of available memory and crash. However, we find that the state-of-the-art DOE and AL for computer experiments, such as techniques described in [6], [7], and [8], among others, do not incorporate memory constraints in their decision making but focus exclusively on the runtime. On the other hand, while the resource utilization studies in High Performance Computing (HPC), such as [9] and [10], describe methods for prediction of applications' peak resident set size (RSS), such analysis does not provide experimenters with practical guidance on which simulations should be chosen and which are best to avoid. We aim to bridge this gap by proposing a multi-objective AL scheme that uses the feedback from both cost and memory models in its experiment selections.

Our key contributions are the following:

- To the best of our knowledge, we describe the first application of AL in the context of AMR. Ideally, AL techniques would promote more widespread use of AMR by allowing non-expert users to select efficient configurations in a systematic fashion.
- We develop a multi-objective AL scheme. Unlike most approaches applied to computer simulations, our scheme refines the underlying models while avoiding simulations that exceed memory constraints.
- In our simulation-based evaluation that uses AMR performance measurements obtained on a supercomputer, we characterize the developed scheme in terms of its mistakes and the corresponding opportunity costs, as well as describe the relevant cost-error trade-offs.

The remainder of this paper is organized as follows. In Section II, we summarize related work. Section III presents the mathematical constructs used for GPR and AL, and Section IV describes the practical aspects of our implementation of these constructs. In Section V, we present our evaluation of AL algorithms applied to a dataset with costs and memory usage of AMR simulations. We investigate the AL "progress" and characterize the trade-offs associated with the developed algorithms. We conclude the paper and outline our future work in Section VI.

II. BACKGROUND AND RELATED WORK

A. Adaptive Mesh Refinement

There exist several approaches to AMR, including the block-structured and the tree-based AMR (e.g., see [11], [12], [13], [14]), as well as many software tools for AMR, such as the libraries and packages listed at [15]. We focus on FORESTCLAW [16], the block-structured adaptive finite volume library for solving hyperbolic partial differential equations on mapped, logically Cartesian meshes. To achieve scalability, FORESTCLAW uses p4est [17], [18], a state-of-the-art library for grid management based on the forest-of-octrees idea.

In [19], the authors describe CLAWPACK (Conservation Law Package), an ecosystem of library and application code for solving nonlinear hyperbolic partial differential equations with high-resolution finite volume methods based on Riemann solvers and limiters. Adaptive versions of CLAWPACK include AMRCLAW and FORESTCLAW. The performance and scalability of these and alternative solutions are investigated in many studies, including [16], [20], and [21]. To our knowledge, none of the existing studies considers running series of AMR simulations using AL techniques. From the modeling perspective, our work extends beyond the common performance analysis setting where performance metrics are treated as functions of a single parameter - the number of processors. In contrast, we model cost and memory responses as functions of numerical, physical, and machine parameters in a 5-dimensional input space.

B. Gaussian Processes

Many studies propose optimizations of the surrogate modeling that uses GPR. In [8], the authors describe the Bayesian

treed GPR where the treed partitioning algorithms help overcome the challenges associated with the stationarity of GPR (i.e. the fitting with the same covariance structure applied to the entire input space) and its growing computational complexity. Similarly, [22] describes a method for accelerating GPR training by partitioning the training data in local regions and learning independent local Gaussian models inspired by the locally weighted projection regressions. Sparse Pseudoinput Gaussian Processes [23] and Sparse Spectrum Gaussian Processes [24] exploit sparsity in the input point space (by inducing pseudoinputs) and the kernel's spectral space, respectively, and drastically reduce computational complexity of the modeling. These optimizations and approximations are compatible with the cost- and memory-aware AL described in this paper. By optimizing the underlying surrogate modeling procedure, these techniques will allow AL to make intelligent experiment selection decisions based on massive experimental datasets.

C. Bayesian Optimization and Active Learning

The survey in [25] provides a detailed overview of research and development in the area of Bayesian Optimization (BO). In the context of our work, the most relevant concepts introduced in BO literature include *constrained BO* and the *cost sensitivity* analysis. In the former case, the desired experiment selection is required to respect constraints that make certain regions of the input space invalid. Such optimization problems can be handled using the techniques from [26] and [27]. The latter case refers to problems in which the cost of selected response evaluations varies throughout the input space, which is almost always true in studies concerning performance and scalability. The growth of computational complexity in AMR is a notable example of such cost variability. It is also worth mentioning the *regret* function in BO. Specifically, cumulative regret functions are constructed to characterize the optimality of selected sets of experiments: the lower the regret, the better the set. We formulate this concept for AMR simulations that satisfy or exceed memory constraints and describe our evaluation results in Section V.

The aforementioned and related BO concepts provided inspiration for our AL development and evaluation. The key difference about our work is about the objective of running computational experimentats: while BO's goal is to find a global maximizer (or minimizer) of an unknown objective function, we rely on the developed AL to obtain an accurate surrogate model across the entire input space rather than only in the neighborhood of the peak point. In other words, even when higher values of the analyzed response are clearly superior (e.g, for responses expressed in FLOPs or degrees of freedom per second), the following holds: it is still worth running experiments that are likely to be inexpensive and informative. These experiments, combined with a smaller number of cautiously selected expensive experiments, should provide much richer output than the location and the value of the peak. Thus, we develop AL algorithms that construct surrogate models for simulation runtime and memory usage that are reliable enough to be presented to the experimenter or used in such analysis as solving inverse problems, numerical integration, and multi-objective optimization.

III. Modeling and Active Learning

To apply AL to the regression analysis for cost and memory usage of AMR simulations, we use a process that produces estimates of the mean and the standard deviation of predictive distributions for these metrics in many possible configurations. We use GPR, also referred to as *kriging*, to obtain the desired information about the posterior probability distributions. Below we summarize the mathematical constructs used in this modeling. Our notation uses boldface for vectors and uppercase boldface symbols for matrices.

Let $\boldsymbol{X}$ be the design matrix where columns represent features, and let $\boldsymbol{y}$ be the vector of corresponding values for one of the responses, cost or memory usage; we define specific notation for each response at the end of this section. Our goal is to find an underlying function $f(\boldsymbol{x})$ which best fits the measurements with Gaussian noise:

$$\boldsymbol{y} = f(\boldsymbol{x}) + \mathcal{N}\left(0, \sigma_n^2\right), \tag{1}$$

where $\boldsymbol{x}$ is a row in matrix $\boldsymbol{X}$ and σ_n^2 is noise variance. For an arbitrary vector $\boldsymbol{x}_\star$, not necessarily from $\boldsymbol{X}$, the posterior distribution for individual predictions $f(\boldsymbol{x}_\star)$ takes form of a multivariate Gaussian distribution:

$$p(f(\boldsymbol{x}_\star) \mid \boldsymbol{x}_\star, \boldsymbol{X}, \boldsymbol{y}) \sim \mathcal{N}\left(\boldsymbol{\mu}_\star, \boldsymbol{\sigma}_\star^2 \boldsymbol{x}_\star\right), \tag{2}$$

with

$$\boldsymbol{\mu}_\star = \boldsymbol{k}_\star^\mathsf{T} \boldsymbol{K}_y^{-1} \boldsymbol{y}, \boldsymbol{\sigma}_\star^2 = \boldsymbol{k}_{\star\star} - \boldsymbol{k}_\star^\mathsf{T} \boldsymbol{K}_y^{-1} \boldsymbol{k}_\star, \boldsymbol{K}_y = \boldsymbol{K} + \sigma_n^2 \boldsymbol{I}. \tag{3}$$

The covariance function $k(\boldsymbol{x}_p, \boldsymbol{x}_q)$ affects the parameter estimation as follows. A covariance matrix:

$$[\boldsymbol{K}]_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j), \text{ for all columns } \boldsymbol{x}_i \text{ and } \boldsymbol{x}_j \text{ in } \boldsymbol{X}, \tag{4}$$

a covariance vector:

$$[\boldsymbol{k}_\star]_i = k(\boldsymbol{x}_\star, \boldsymbol{x}_i), \text{ for all columns } \boldsymbol{x}_i \text{ in } \boldsymbol{X}, \tag{5}$$

and a scalar,

$$k_{\star\star} = k(\boldsymbol{x}_\star, \boldsymbol{x}_\star) \tag{6}$$

are calculated using the selected kernel function $k(\boldsymbol{x}_p, \boldsymbol{x}_q)$.

Our choice of the covariance function is motivated by [28] and [4], where the authors describe the squared exponential, also referred to as the radial basis function or RBF, as a common solution:

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \sigma_f^2 \mathsf{exp}\left(-\frac{|\boldsymbol{x}_p - \boldsymbol{x}_q|^2}{2l^2}\right), \tag{7}$$

where the length scale l, the amplitude σ_f^2, and the noise level σ_n^2 are so-called hyperparameters. The hyperparameters need to be given appropriate values so the resulting regression best matches the given dataset $(\boldsymbol{X}, \boldsymbol{y})$. As an alternative to RBF, the authors in [8] and [6] consider the Matérn covariance function family and argue for its advantages, including the controllable smoothness of the model. To keep the evaluation results compatible with our previous analysis in [3], we continue to use RBF-based GPR modeling. Comparing kernels in the

context of AL is an interesting subject but it falls outside the scope of the current analysis.

According to the Bayesian inference approach with marginal likelihood, the following quantity, referred to as the log marginal likelihood (LML),

$$\log p(\boldsymbol{y} \mid \boldsymbol{X}, l, \sigma_f^2, \sigma_n^2) = -\frac{1}{2}\left(\boldsymbol{y}^\intercal \boldsymbol{K}_y^{-1}\boldsymbol{y} + \log|\boldsymbol{K}_y|\right) + \boldsymbol{C}, \tag{8}$$

is minimized with respect to the selected hyperparameters:

$$(l, \sigma_f^2, \sigma_n^2) = \underset{l,\sigma_f^2,\sigma_n^2}{\arg\min}(\log p(\boldsymbol{y} \mid \boldsymbol{X}, l, \sigma_f^2, \sigma_n^2). \tag{9}$$

After solving this optimization problem and fitting the hyperparameters, we can use the constructed GPR model to obtain the mean $\boldsymbol{\mu}_{f(x)}$ and the standard deviation $\boldsymbol{\sigma}_{f(x)}$ of the predictive distribution for the response function $f(\boldsymbol{x})$ at any point $\tilde{\boldsymbol{x}}$, in the input space, sampled in $\boldsymbol{X}$ or not.

When simulating cost- and memory-aware AL, we train two GPR models, as described above. The cost model is trained on $(\boldsymbol{X}, \boldsymbol{c})$ and the memory model is trained on $(\boldsymbol{X}, \boldsymbol{m})$, where $\boldsymbol{c}$ and $\boldsymbol{m}$ are vectors of cost and memory responses, respectively. In other words, vectors $\boldsymbol{c}$ and $\boldsymbol{m}$ substitute $\boldsymbol{y}$ in the GPR formulation from above. Each model independently updates its hyperparameters after every iteration. For any set of points $\{\tilde{\boldsymbol{x}}_i\}$ considered by AL, these models can characterize its i-th candidate using the following values: μ_{cost_i} and σ_{cost_i} – mean and standard deviation of cost predictions, as well as μ_{mem_i} and σ_{mem_i} – mean and standard deviation of memory predictions. As described in the following section, we implement five algorithms which make distinct decisions based on vectors $\boldsymbol{\mu}_{cost}$, $\boldsymbol{\sigma}_{cost}$, $\boldsymbol{\mu}_{mem}$, and $\boldsymbol{\sigma}_{mem}$ in the process of identifying $\hat{\boldsymbol{x}} \in \{\tilde{\boldsymbol{x}}_i\}$, the point that will be used to update both models at the next iteration.

IV. Implementation and Analyzed Datasets

We perform our AL analysis in Python using the tools for Machine Learning and data analysis from the package called *scikit-learn* [29]. Specifically, we use the code for Gaussian Processes [30] from version 0.18.1. Unlike earlier versions, it provides a completely revised Gaussian Process module and supports kernel engineering, among other features. To analyze batches of AL simulations in parallel, we leverage the process-based parallelism enabled by Python's *multiprocessing* module [31].

To drive the developed AL simulator, we partition the dataset, which we describe later in this section, into three subsets: *Initial* (for initial regression fitting), *Active* (for one-at-a-time experiment selection with AL), and *Test* (for prediction quality analysis). In each experiment with a dataset that contains n samples, we randomly shuffle its samples and assign $n_{Test} = 200$ samples to the Test partition. Using a tunable $r_{init} \in (0, 1)$ parameter, we then vary the number of Initial samples, defined as: $n_{Init} = \text{int}(r_{Init} \times (n - n_{Test}))$, and leave the remaining samples for AL. In this manner, we experiment with large ($r_{Init} = 0.3$), moderate ($r_{Init} = 0.1$), and minimal ($r_{Init} = 0.004$) Initial partitions. For a dataset with $n = 612$, the absolute values n_{Init} are 123, 41, and 1, respectively.

Algorithm 1 Active Learning Procedure

Input: Matrix $\boldsymbol{X} \in \mathbb{R}^{n_{Active} \times d}$, cost response vector $\boldsymbol{c} \in \mathbb{R}^{n_{Active}}$, and memory response vector $\boldsymbol{m} \in \mathbb{R}^{n_{Active}}$ (n_{Active} – number of samples in the the Active set, d – number of features); GPR_{cost} and GPR_{mem} are GPR models trained on the data from Initial set, $(\boldsymbol{X}_{Init}, \boldsymbol{c}_{Init})$ and $(\boldsymbol{X}_{Init}, \boldsymbol{m}_{Init})$, respectively.

Output: $\boldsymbol{X}_{Learned}$, trained GPR_{cost} and GPR_{mem} models

$\boldsymbol{X}_{Learned}[] \leftarrow 0, \boldsymbol{c}_{Learned}[] \leftarrow 0, \boldsymbol{m}_{Learned}[] \leftarrow 0$
for $i = 0 \rightarrow n_{Active} - 1$ **do** ▷ Iterate over all samples
  $\boldsymbol{\mu}_{cost}, \boldsymbol{\sigma}_{cost} \leftarrow GPR_{cost}.predict(\boldsymbol{X})$
  $\boldsymbol{\mu}_{mem}, \boldsymbol{\sigma}_{mem} \leftarrow GPR_{mem}.predict(\boldsymbol{X})$
  $idx \leftarrow select\_candidate(\boldsymbol{X}, \boldsymbol{\mu}_{cost}, \boldsymbol{\sigma}_{cost}, \boldsymbol{\mu}_{mem}, \boldsymbol{\sigma}_{mem})$
  ▷ Find the candidate using to the selected AL algorithm
  $\boldsymbol{X}_{Learned}[i] \leftarrow \boldsymbol{X}[idx, :]$
  $\boldsymbol{c}_{Learned}[i] \leftarrow \boldsymbol{c}[idx], \quad \boldsymbol{m}_{Learned}[i] \leftarrow \boldsymbol{m}[idx]$
  Remove idx-th row from $\boldsymbol{X}$
  Remove idx-th elements from $\boldsymbol{c}$ and $\boldsymbol{m}$
  $GPR_{cost}.fit([\boldsymbol{X}_{Learned}, \boldsymbol{X}_{Init}], [\boldsymbol{c}_{Learned}, \boldsymbol{c}_{Init}])$
  $GPR_{mem}.fit([\boldsymbol{X}_{Learned}, \boldsymbol{X}_{Init}], [\boldsymbol{m}_{Learned}, \boldsymbol{m}_{Init}])$
    ▷ Retrain both models on the data that includes samples from Initial set, all previously selected samples, and the sample selected at the current iteration; use old model's parameters as a starting point in hyperparameter fitting.
end for

The last case corresponds to the realistic practical scenario where an application is first run on a new platform or in a new configuration to verify the correctness and then the performance characteristics are collected for the following runs.

After fitting the models on the Initial set, our simulator repeats the following cycle: suggest a point for the next experiment, obtain the desired measurements for that experiment, and retrain the models with these measurements. This process guides the sequential exploration of the input space in an unpredictable yet efficient manner. The pseudocode for the exploration routine that trains cost and memory models is given in Algorithm 1. To select specific candidates, this routine calls functions that implement algorithms that we introduce in Section IV-B.

Our analysis framework runs in an "offline" mode, consulting a database of precomputed performance samples. This enables cross-validation and thus robust comparison of AL strategies with modest computational cost. In contrast, an "online" AL system makes decisions about what experiment to run next before the experiment has been run. If the same deterministic AL algorithm was run again, it would begin by repeating performance samples using the same parameters and yet would measure different performance (within machine variability) and thus choose to sample different configurations. This is wasteful and non-deterministic. Although we intend for our AL methods to ultimately be used in an active context, the quality and reliability of different algorithms is better analyzed offline.

We feed many shuffled copies of the dataset with precomputed samples into the developed AL algorithms as input, run many AL instances or *trajectories*, and draw conclusions in terms of performance quantiles. Similarly, simulations in the state-of-the-art study [7], which focuses on AL performance, run "offline" and use lookup tables with experimental data.

The terms that refer to the entries in performance dataset can be used interchangeably, but we intend to use them in the following contexts. *Jobs* refer to individual computations performed on a supercomputer; each job is run to obtain information about a configuration corresponding to the selected combination of feature values. After this information is processed, these jobs or *samples* are added to the dataset, which we later randomly partition. Among the samples in the Active partition, AL selects one *experiment* or *candidate* at each iteration and updates the underlying models.

A. Datasets

In order to study the efficiency of the proposed AL algorithms, we ran a large collection of shock-bubble situations with different parameters using the FORESTCLAW package. We ran over 1K computational jobs on Edison [32], at NERSC (US National Energy Research Computing Center). These jobs were scheduled to run on Edison's compute nodes (two-socket 12-core Intel "Ivy Bridge" processors running at 2.4 GHz and interconnected with the Cray Aries network with the Dragonfly topology running at 23.7 TB/s global bandwidth) using SLURM [33], the supercomputer's workload manager and scheduler. After completion, we collected all relevant information, including FORESTCLAW output, error logs, and scheduler accounting information, and transferred it from the supercomputer to a local workstation in order to analyze it using the developed simulator for AL evaluation.

In the processing phase, we realized that memory usage data was only available for 612 jobs. For the rest of them, SLURM reported zeros in the field called *MaxRSS* – the maximum memory resident set size for all job tasks. What appears to be a bug in the usage reporting system[1], only affected some of the least expensive jobs. Indeed, the longest job with MaxRSS $= 0$ ran for 139 seconds (the global maximum is over an hour), and the smallest non-zero MaxRSS reported by SLURM is around 16KB. Therefore, we believe that the most interesting aspects of AL, which manifest themselves when the most expensive experiments are selected and studied, can be analyzed using the reduced 612-sample dataset. In this set, the computational cost, expressed in node-hours, of the most expensive experiment exceeds the cost of the least expensive experiment by the factor of 5.9×10^3.

Table I provides details of the analyzed dataset. The intervals in which the responses and features change are relatively large, allowing us to explore the behavior of the application performance across a domain where the growth along some of the dimensions is significant. These 612 simulations represent a subset of the total 1920 possible combinations of all sampled values of 5 features. Of these simulations, 532 jobs represent unique parameter combinations, whereas the remaining 80 jobs provide 2nd and in some cases 3rd repetitive measurements, capturing the machine performance variability. Therefore, 532 combinations studied make up to 28% of the total 1920 combinations, a fraction we expect to provide sufficient data to adequately model and predict the responses within the parameter domain of interest. To collect this dataset, we used over 30K core-hours from our computing allocation at NERSC.

The job cost is calculated as a product of wall-clock time and the number of processors used, and the MaxRSS values characterize the peak memory per process consumption. Before partitioning the dataset, we pre-process it by applying log_{10} transformation to the cost and memory responses. Our analysis confirms that the discrepancy in the prediction quality for the extremes, the smallest and the largest response values, is reduced after we apply this transformation. It also eliminates the risk of obtaining nonsensical negative predictions that GPR occasionally produces for near-zero runtimes in the non-log space. In our evaluation, we fit GPR models for log-transformed cost and memory responses and analyze the means of predictive distributions $\boldsymbol{\mu}_{cost}$ and $\boldsymbol{\mu}_{mem}$, respectively. Using exponentiation, we obtain non-log predictions that are always positive and compare them with actual measurements in the dataset in our error analysis. Following common practice, we also scale all performance features to the unit cube $[0, 1]^d$, where $d = 5$ is the input space dimension.

B. Candidate Selection Algorithms

Aiming to explore a diverse set of AL schemes, we developed five algorithms that guide AL as follows.

- **RandUniform**: uniform random sampling. This scheme does not consider the model output for predicted responses. It is not useful in sequential AL, as this sampling can be performed in batches more efficiently, without the need to incrementally fit the model. In our evaluation, it serves as a reference point in the algorithm comparison.
- **MaxSigma**: at each iteration, this algorithm selects an experiment with the largest uncertainty characterized by the vector $\boldsymbol{\sigma}_{cost}$ produced by the current GPR model for the costs of remaining samples. In [3], we called this algorithm *Variance Reduction*, whereas the comprehensive survey of AL in classification problems [5] refers to this querying strategy as *Uncertainty Sampling* with candidate selection based on the *least confident* predictions.
- **MinPred**: With the candidate selection defined as:

$$x^* = \arg\max_i \left(\boldsymbol{\sigma}_{cost_i} - \boldsymbol{\mu}_{cost_i}\right),$$

we aim to implement the greedy cost efficient algorithm, which is equivalent to the scheme that finds the maximum of the element-wise ratio $\boldsymbol{\sigma}_{cost}/\boldsymbol{\mu}_{cost}$ in the non-log space. We expect to select the candidates with the largest prediction uncertainty per unit of cost. However, our analysis shows that this selection algorithm degrades to the state where its decisions are influenced primarily by $\boldsymbol{\mu}_{cost}$ due to the difference in scales of values in $\boldsymbol{\mu}_{cost}$

[1]This incident was reported to NERSC system administrators.

TABLE I: Parameters of the AMR shock-bubble simulation dataset with 612 samples.

	min	**median**	**mean**	**max**
Feature: *p*, # of nodes	4	8	12.740	32
Feature: *mx*, box size	8	16	20.520	32
Feature: *max_level*, max refinement level	3	5	4.720	6
Feature: *r0*, bubble size	0.200	0.300	0.340	0.500
Feature: *rhoin*, bubble density	0.020	0.100	0.160	0.500
Response: *wall clock time*, seconds	1.970	94.840	239.150	4262.730
Response: *cost*, node-hours	0.002	0.248	0.800	11.850
Response: *memory*, MB	0.020	7.980	7.500	32.560

and $\boldsymbol{\sigma}_{cost}$. We notice that the variations in the former are typically hundreds to times larger than the variations in the latter. Therefore, we give this policy the name MinPred, as it indeed selected the candidates with the smallest predicted costs $\boldsymbol{\mu}_{cost_i}$ in all relevant cases that we investigates. We address this strong skewness in the following algorithm.

- **RandGoodness**: In contrast to the greedy deterministic selection in MinPred, we propose a randomized algorithm that samples candidates with the candidate "goodness" measured as: $\boldsymbol{g}_{cost} = 10^{\boldsymbol{\sigma}_{cost} - \boldsymbol{\mu}_{cost}}$. Base 10 is the most intuitive option here since we apply the logarithm base 10 to our datasets in the pre-processing step; higher bases will lead to more skewed candidate distributions. After normalizing its components so they add up to 1.0, we can use $\boldsymbol{g}_{cost}$ as a discrete probability distribution for randomized candidate selection. This sampling is designed to choose candidates near those selected by MinPred in most cases and occasionally select more expensive candidates, as gauged by model predictions. It adds exploration ability in the candidate selection that otherwise purely exploits regions of the input space with the least expensive samples. We expect RandGoodness to acquire samples outside of those regions and improve its predictions throughout the entire input space.
- **RandGoodness with Memory Awareness (RGMA)**: This algorithm is a memory-aware extension of RandGoodness with a candidate filtering step. Given a specific memory limit, this algorithm leaves out the candidates for which memory predictions $\boldsymbol{\mu}_{mem}$ exceed this limit, marking them as undesirable. The rest of the candidates are considered safe and used in random drawing with $\boldsymbol{g}_{cost}$ distribution. Additional assumptions and implementation details are described in Section V-C.

Our implementations of these algorithms take the form of individual functions that can be called from Algorithm 1 for candidate selection. For the batch mode in our evaluation, we use an outer loop that runs the AL routine with each selection algorithm on the specified number of random partitions of the analyzed dataset. Each AL instance or *trajectory* captures the error and cost characteristic of the process that learns the data made available to it in the randomly selected Active partition. By processing a large of number of trajectories, we can reason about the statistical properties of the algorithms independent of the initial conditions related to specific partition properties.

V. Evaluation

In our experimental evaluation, we gain valuable insight into the following aspects of the AL-based experimentation and the proposed algorithms. First, we study the impact of AL decisions on the cost distribution of selected samples. Second, we define necessary evaluation metrics for comprehensive analysis of trade-offs. We also describe the assumptions we use in our simulation-based analysis of memory awareness with RGMA. Third, using such metrics as cumulative regret, cumulative cost, and an aggregate error measure, we present our findings about algorithm optimality and characterize how sensitive the algorithms are with respect to the amount of data used in the pre-AL model fitting. At the end of this section, we discuss additional optimizations and describe the audience that will likely find much practical value in our work.

A. Cost Distribution Analysis

We begin our evaluation with a set of experiments that aims to characterize the algorithms from the cost perspective and run instances of AL with all algorithms except for RGMA (memory-specific trade-offs are discussed later in this section). The goal of this analysis is to confirm our intuitions about the aforementioned skewness and exploratory abilities that are present in these algorithms.

Fig. 2 presents a high-level view of the candidate selection algorithms being evaluated. The violin-shaped filled areas depict the distributions of the samples selected in the first 150 AL iterations (roughly half of all samples in Active partitions) in a single trajectory for each algorithm. The violin width represents the relative frequency of the corresponding y-axis cost values (actual, not predicted), the thick vertical lines depict interquartile ranges (IQRs), and white dots show the median values. RandUniform and MaxSigma demonstrate similar cost distributions with no bias, therefore this high-level view provides no basis for choosing one algorithm over the other. In contrast, we can see that both RandGoodness and MinPred tend to select inexpensive experiments. At the same time, we confirm that RandUniform is able to select more expensive experiments than MinPred and find its long-tailed cost distribution the most preferable. In order to capture

Fig. 2: Cost of samples selected by AL algorithms.

specific trade-offs between these algorithms, we define cost- and memory-specific evaluation metrics, as described below.

B. Evaluation Metrics and Simulated Memory Awareness

We define the following metrics and record their values after every AL iteration:

- RMSE: Following common practice, we estimate the aggregate error as root-mean-square error:

$$RMSE = \frac{1}{\sqrt{n_{Test}}}\|\boldsymbol{e}\| = \sqrt{\frac{1}{n_{Test}}\boldsymbol{e}^{\mathbf{T}}\boldsymbol{e}}, \quad (10)$$

  where $\boldsymbol{e}_{cost} = \boldsymbol{\mu}_{cost}^{Test} - \boldsymbol{c}^{Test}$ and $\boldsymbol{e}_{mem} = \boldsymbol{\mu}_{mem}^{Test} - \boldsymbol{m}^{Test}$ can be used as vector $\boldsymbol{e}$ in the cost and memory error analysis, respectively. These calculations use non-log predictions (i.e. model output converted using exponentiation) and unmodified responses from the dataset. Superscript $Test$ denotes that predictions are obtained for samples in the Test partition, for which the corresponding cost and memory responses constitute $\boldsymbol{c}^{Test}$ and $\boldsymbol{m}^{Test}$ – the portion of the dataset used exclusively for error estimation, not for model training.
- Cumulative Cost: Not only are we interested in the reduction of RMSE, but we also want to quantify how quickly this reduction happens with respect to the growth of the sample cumulative cost in every AL trajectory. This metric is the sum of the costs of all samples $\hat{\boldsymbol{x}}_i$ selected before the current iteration: $CC = \sum_i c_i$.
- Cumulative Regret: This metric estimates the sum of all opportunity costs incurred when AL selections correspond to the jobs that exceed the maximum allowed memory usage L_{mem}. Formally defined as:

$$IR_i = \begin{cases} c_i, & \text{if } m_i \geq L_{mem}, \\ 0, & \text{otherwise,} \end{cases} \quad CR = \sum_i IR_i, \quad (11)$$

  CR is the total sum of the worst-case *individual regrets* IR_i incurred when such jobs run almost to completion but exceed the memory limit at the very end and crash. If AL algorithms had selected alternative candidates in such cases, the amount of computing cycles measured by c_i could have been used towards more productive jobs, resulting in no failures and providing useful data. With respect to a given memory limit L_{mem}, we can classify all candidates as *satisfying* ($\boldsymbol{\mu}_{mem} < L_{mem}$) and *exceeding* ($\boldsymbol{\mu}_{mem} \geq L_{mem}$) based on memory predictions. RGMA, unlike the rest of the algorithms, implements this classification and attempts to reduce CR by selecting only satisfying samples.

While running our pre-selected set of AMR simulations on Edison, we did not come close to using all available node memory. In fact, we made sure that the simulations we selected were guaranteed to complete and provide useful data. Therefore, we assume that L_{mem} arise from a hypothetical yet realistic experimentation workflow we choose to simulate and analyze:

- A relatively small set of simulations from the Initial partition is first run in an environment that can satisfy high memory demands. Experimenter's intuition rather than AL is used to select these simulations.
- After switching to an environment with less memory per node, an experimenter relies on AL in running additional simulations and refining the underlying models. In practice, this transition can be motivated by the difference in the costs and allocation limits.

The memory limit is not enforced in the first phase, but L_{mem} is set to the reduced memory amount in the second phase. Our motivation for analyzing this particular workflow comes from the following observations. First, many HPC systems have multiple queues, including queues with high-memory machines (often called *bigmem*) and regular queues. Switching between queues only requires a minimal change in the batch script for an individual simulation. Second, the knowledge obtained from initial experiments can be extremely valuable in mapping the regions where memory usage approaches and exceeds specific memory limits. The larger the set of Initial simulations contributing to this knowledge, the lower the CR values we expect to observe in AL guiding the experimentation in the second phase. Third, with the aforementioned tunable r_{Init} parameter in our evaluation, we can control the size of Initial partition. As this partition grows, we expect to observe the reduction of CR in RGMA selections. If Initial partition is small or does not include high memory simulations, AL has no option other than to learn from its own "mistakes". Thus, in the trial-and-error fashion, it should give high values to the selected configurations that exceed the limit L_{mem}, and learn over time to avoid approaching this limit too closely. Later in this section, we describe our simulations that investigate AL scenarios with large ($n_{Init} = 123$), moderate ($n_{Init} = 41$), and minimal ($n_{Init} = 1$) Initial partitions. In the last case, there is almost no knowledge of high-memory experiments coming from the first phase. We can view it as a simplified and more common experimentation scenario where the initial model fitting and AL run on the same system and face the same memory limit. By comparing these three cases we analyze how sensitive the RGMA's memory awareness is with respect to the amount of prior information about the undesirable configurations.

In our simulations, we use $L_{mem} = 13.67$ MB (95% of the largest log-transformed memory value, or, equivalently, 42% of the largest unmodified memory response in the dataset). Considering the long-tailed distribution for the analyzed memory responses, this limit splits the dataset into 551 satisfying samples and 61 exceeding samples. As shown in Algorithm 2,

Algorithm 2 RGMA with Cost and Memory Awareness

Input: Matrix $\boldsymbol{X}$ with considered candidates and four vectors: means $\boldsymbol{\mu}_{cost}$ and the standard deviations $\boldsymbol{\sigma}_{cost}$ of cost predictions, and means $\boldsymbol{\mu}_{mem}$ and the standard deviations $\boldsymbol{\sigma}_{mem}$ of memory predictions for all candidates in $\boldsymbol{X}$. L_{mem} – maximum allowed memory usage.

Output: Index of the recommended candidate

$filter \leftarrow \boldsymbol{\mu}_{mem} < L_{mem}$ ▷ Vector of $True$ and $False$

$all \leftarrow range(0, len(\boldsymbol{X})), \quad satisfying \leftarrow all[filter]$

$\boldsymbol{g}_{cost} \leftarrow 10^{\boldsymbol{\sigma}_{cost}[filter] - \boldsymbol{\mu}_{cost}[filter]}$

$\overline{\boldsymbol{g}_{cost}} \leftarrow \boldsymbol{g}_{cost} / \sum \boldsymbol{g}_{cost}$ ▷ Normal cost-based "goodness"

$idx \leftarrow$ draw value from vector $satisfying$ using the discrete probability distribution defined by $\overline{\boldsymbol{g}_{cost}}$

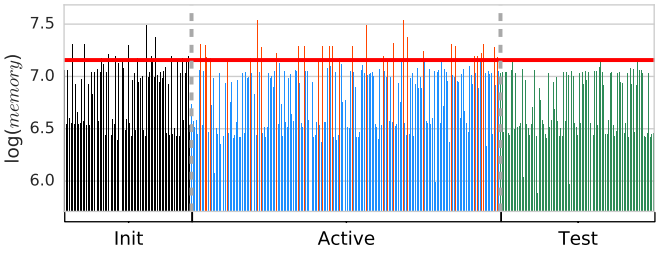
RGMA accepts L_{mem} as input and tries to predict whether the candidates are likely to satisfy or exceed this limit based on $\boldsymbol{\mu}_{mem}$ predictions. The exceeding candidates are then excluded from the randomized candidate drawing. It is worth mentioning that this implementation becomes equivalent to RandGoodness if vector $filter$ contains only $True$ values.

C. Analysis of Algorithm Trade-offs

As outlined earlier, our dataset partitions play distinct roles: initial model fitting, AL, and error evaluation. To match the simulated experimentation workflow, we partition shuffled copies of the dataset such that both Initial and Active partitions include samples with memory usage greater than L_{mem}, whereas the Test partition only contains values below L_{mem}, providing error estimates based on allowed configurations. This point is demonstrated in Fig. 3. With no particular order within partitions, available samples are shown as vertical lines that indicate their memory usage. With this structure in mind, we formulate the aforementioned sensitivity analysis as follows: after modeling a varying amount of data in the Initial partition, can AL efficiently sample data in Active partition while avoiding a moderate set of undesirable, exceeding samples?

To understand this sensitivity, we run evaluation experiments and show how CR grows under different AL algorithms in Fig. 4a. We run each algorithm with $n_{Init} = 41$ and for RGMA also include experiments that use Initial partitions of different sizes. Each variant runs on 32 shuffled copies of the dataset, partitioned as described above. All RGMA instances flatten out: CR does not increase significantly after AL selects enough samples. RGMA with $n_{Init} = 123$ flattens out at the lowest value as it extracts the most knowledge from the large Initial partition. We do not show RandGoodness on the plot since it behaves similar to MinPred and does not flatten out. For MaxSigma and RandUniform, our analysis shows that CR grows proportionally to CC. RandUniform's CR increases almost linearly, whereas MaxSigma's CR plateaus only because there are no expensive samples left in the Active partition after about 100 iterations. Below, we focus on RGMA and evaluate the knowledge obtained using the Initial partition from the cost-error perspective.

Fig. 3: Distribution of samples across partitions. Red vertical lines depict samples which exceed the memory limit L_{mem}, shown with the horizontal line, and should be avoided in AL.



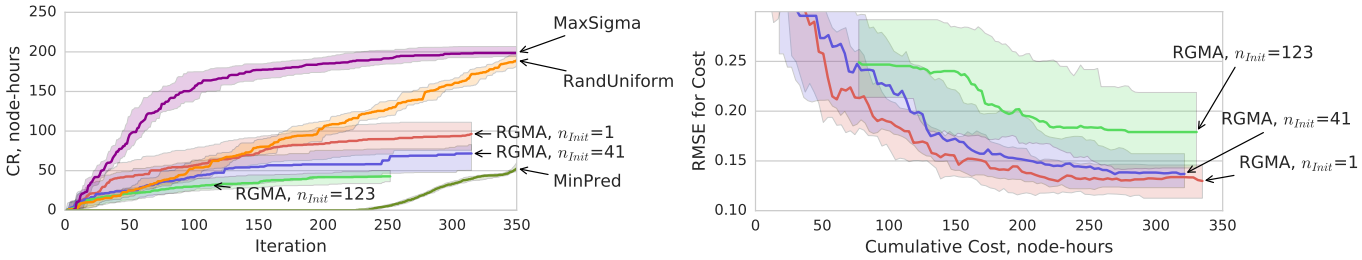
In the same set of evaluations, we analyze the relationship between $RMSE_{cost}$ and CC. The trade-off curves that plot the former as a function of the latter are shown in Fig. 4b. These curves provide a more practical view on the algorithm performance than these metrics plotted individually as functions of the iteration count. Indeed, an experimenter can select the algorithm that provides the most accurate cost model for a range of computing budgets dedicated to the desired experimentation. For fair cost comparison, each shown CC value includes the cost of all Initial samples since those samples are also used in the model fitting. The figure shows that increasing the size of the Initial partition moves the cost-error curve away from the lowest, optimal curve defined by RGMA with $n_{Init} = 1$.

By combining both the regret and the error perspectives, we arrive at the following conclusions. RGMA demonstrates that the underlying GPR-based memory model can be used to avoid experiments that violate memory constraints. RGMA's plateauing cumulative regret makes this algorithm an attractive candidate for practical AL-guided experimentation. Regarding its tuning, we find that increasing the number n_{Init} of pre-AL experiments might be beneficial as it lowers the amount of compute cycles spent on mistakenly selected failing experiments. However, in the studied case with costly AMR computations, the initial training should be minimal to allow AL select all experiments in a cost-efficient manner. Considering that RMSE is a strong metric that averages $n_{Test} = 200$ predictions in our case, even a small advantage provided by RGMA with $n_{Init} = 1$ over the second-best case with moderate $n_{Init} = 41$ Initial partition is significant.

D. Discussion

Below we summarize three directions that have potential to further improve the proposed AL and the underlying modeling.

First, we can tune the modeling to best fit features with specific growth patterns. It is common to view performance characteristics as functions of the number of processors used p which is selected among values such as 2^2, 2^3, 2^4, and so on, especially in the analysis that focuses on application scalability. If we adjust our pre-processing, we can train GPR models using this exponent as a feature such that, for instance, the point with 2^3 processors is spaced equally from 2^2 as it is from



(a) Growth of Cumulative Regret

(b) RMSE for cost predictions as a function of Cumulative Cost

Fig. 4: Trade-off analysis that uses three evaluation metrics: Cumulative Regret, Cumulative Cost, and RMSE. The plots depict IQRs (25th and 75th percentiles) as boundaries of the filled areas and the median values as bold lines inside IQRs.

2^4. This adjustment may provide advantages in the analysis where feature values span many orders of magnitude.

Second, finding optimal stopping conditions in AL is a non-trivial task. As demonstrated in Fig. 4a, our RGMA trajectories already stop before the rest of the algorithms. We added an early termination case into our RGMA implementation that is triggered only when all remaining samples are likely to exceed the memory limit. We also noticed that this is not a sufficient condition since in some cases CR demonstrated growth beyond the plateau values in the very last iterations before stopping, i.e. when the candidates are scarce. We may overcome this issue by employing the approach from [34] based on the idea of using stabilizing predictions as a heuristic for stopping AL. In practice, multiple factors, including stabilizing predictions, stabilizing hyperparameters, and the reduction of prediction uncertainty for both cost and memory metrics, should be considered in applications of AL to computer simulations.

Third, we can rewrite our error metric in (10) as:

$$RMSE = \sqrt{\boldsymbol{e}_{Test}^{\mathbf{T}} \boldsymbol{\rho} \boldsymbol{e}_{Test}}, \tag{12}$$

where $\boldsymbol{\rho}$ is the diagonal matrix $1/n_{Test}\boldsymbol{I}$. With this uniform weighting of samples, our error metric weights all samples equally. This is reasonable in the present context, but when parametrizing a sample space it is common for the number of expensive candidate samples to be much larger than the number of inexpensive samples, effectively leading to prioritization of accuracy for these large samples. This is especially likely when working with a high-dimensional parameter space or linearly spaced parameters. Indeed, our study would have suffered from this bias if we had not pre-selected our jobs to limit the total cost by more sparsely sampling the expensive parameter regimes. Rather than pruning the parameter space to correct for this bias, it can be more convenient to use a non-uniform diagonal weighting $\boldsymbol{\rho}$ which would allow the user to specify the relative priority of different regions of the parameter space. While the performance modeling in [35] calls for a scale-independent error metric, we argue that scale-dependent error metrics are superior in the cost-efficient AL: from the experiment's perspective, prediction errors for inexpensive experiments are more tolerable than the same errors in predictions for expensive experiments.

Our final remark relates to the potential of broader impact of AL in the AMR community. There appears to be a large gap between the developers of AMR codes and expert users on one side and non-expert users and interested practitioners on the other. The former group already uses back-of-the-envelope heuristics to manage their simulations, whereas the latter group may not have such insights and can benefit from supporting methodologies and tools. We hope that the algorithms and the analysis discussed in this paper will prove useful to the latter group and promote Active Learning techniques, including the cost- and memory-aware algorithms, as reliable and efficient.

VI. Conclusions and Future Work

We demonstrate how Active Learning combined with Gaussian Process Regressions can be used to optimize sequences of parallel computer experiments, specifically for Adaptive Mesh Refinement simulations. We describe how Active Learning techniques help incrementally model and predict performance characteristics such as the computational cost and memory usage. We present our experience with applying five different algorithms that guide experimentation in a 5-dimensional input space of machine-specific, numerical, and physical parameters. Our key contribution relates to the development and experimental evaluation of an algorithm that provides cost-efficient and memory-aware exploration of this space. We confirm that the algorithm learns from its mistakes and improves its ability to avoid simulations that violate memory constraints as the learning process evolves. Researchers and practitioners who use Adaptive Mesh Refinement are likely to benefit from employing this algorithm to guide efficient sampling when they run series of simulations in such types of analysis as parameter sweeps, optimization, and uncertainty quantification. While our work is developed in the Adaptive Mesh Refinement context, the key ideas can be applied to other types of computing experiments, especially in environments with severe cost and memory constraints. In applications that are more conducive to exploration, Active Learning may help efficiently sample input spaces with many dimensions and construct useful probabilistic surrogate models.

In our future work, we consider evaluating alternative kernel functions (e.g., anisotropic RBF kernels and Matérn kernels with controllable smoothness) and regressions for non-Gaussian

distributions. These model improvements are likely to provide significant advantages with broad applicability. With regard to the candidate selection in Active Learning, we would like to better understand the trade-offs associated with running multiple simulations in parallel at each iteration of Active Learning. Such schemes increase the scheduling overhead and result in less greedy and optimal selection strategies, but the achieved reduction of the time required to train accurate models may be advantageous in many applications. Finally, we may train multiple local performance models simultaneously and evaluate this extension of the current modeling approach in the context of Adaptive Mesh Refinement simulations.

REFERENCES

[1] M. P. Forum, "MPI: A Message-Passing Interface Standard," Knoxville, TN, USA, Tech. Rep., 1994.

[2] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[3] D. Duplyakin, J. Brown, and R. Ricci, "Active learning in performance analysis," in *Proceedings of the IEEE Cluster Conference*, Sep. 2016. [Online]. Available: http://www.flux.utah.edu/paper/duplyakin-cluster16

[4] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[5] B. Settles, "Active learning literature survey," *Computer Sciences Technical Report*, vol. 1648, 2009.

[6] T. J. Santner, B. J. Williams, and W. I. Notz, *The design and analysis of computer experiments*. Springer Science & Business Media, 2013.

[7] P. Balaprakash, R. B. Gramacy, and S. M. Wild, "Active-learning-based surrogate models for empirical performance tuning," in *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–8.

[8] R. B. Gramacy and H. K. Lee, "Adaptive design of supercomputer experiments," *Technical report, Dept of Applied Math & Statistics*, 2006.

[9] A. Matsunaga and J. A. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 495–504.

[10] E. R. Rodrigues, R. L. Cunha, M. A. Netto, and M. Spriggs, "Helping HPC users specify job memory requirements via machine learning," in *Proceedings of the Third International Workshop on HPC User Support Tools*. IEEE Press, 2016, pp. 6–13.

[11] A. Dubey, A. Almgren, J. B. Bell, M. Berzins, S. Brandt, G. Bryan, P. Colella, D. Graves, M. Lijewski, F. Läffler, B. OâShea, E. Schnetter, B. V. Straalen, and K. Weide, "A survey of high level frameworks in block-structured adaptive mesh refinement packages," *Journal of Parallel and Distributed Computing*, no. 0, pp. –, 2014.

[12] B. V. Straalen, J. Shalf, T. Ligocki, N. Keen, and W.-S. Yang, "Scalability Challenges for Massively Parallel AMR Applications," in *IPDPS '09 Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing*. IEEE Computer Society Washington, DC, 2009, pp. 1–12.

[13] L. Ivan, H. De Sterck, S. A. Northrup, and C. P. T. Groth, "Multi-dimensional finite-volume scheme for hyperbolic conservation laws on three-dimensional solution-adaptive cubed-sphere grids," *J. Comput. Phys.*, vol. 255, no. 0, pp. 205–227, 12 2013.

[14] A. Langer, J. Lifflander, P. Miller, K. C. Pan, L. V. Kalé, and P. Ricker, "Scalable Algorithms for Distributed-Memory Adaptive Mesh Refinement," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*, Oct 2012, pp. 100–107.

[15] D. Calhoun, "Adaptive mesh refinement resources," http://math.boisestate.edu/ calhoun/www_personal/research/amr_software/, accessed: 2016-10-18.

[16] C. Burstedde, D. Calhoun, K. Mandli, and A. R. Terrel, "ForestClaw: Hybrid forest-of-octrees AMR for hyperbolic conservation laws," *arXiv preprint arXiv:1308.1472*, 2013.

[17] C. Burstedde, L. C. Wilcox, and O. Ghattas, "p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees," *SIAM Journal on Scientific Computing*, vol. 33, no. 3, pp. 1103–1133, 2011.

[18] D. Calhoun and C. Burstedde, "ForestClaw : A parallel algorithm for patch-based adaptive mesh refinement on a forest of quadtrees," *arXiv:1703.03116*, 2017.

[19] K. T. Mandli, A. J. Ahmadia, M. Berger, D. Calhoun, D. L. George, Y. Hadjimichael, D. I. Ketcheson, G. I. Lemoine, and R. J. LeVeque, "Clawpack: building an open source ecosystem for solving hyperbolic pdes," *PeerJ Computer Science*, vol. 2, p. e68, 2016.

[20] A. C. Calder, B. C. Curtis, L. Dursi, B. Fryxell, P. MacNeice, K. Olson, P. Ricker, R. Rosner, F. Timmes, H. Tufo *et al.*, "High performance reactive fluid flow simulations using adaptive mesh refinement on thousands of processors," in *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2000, p. 56.

[21] C. Burstedde and J. Holke, "Coarse mesh partitioning for tree based amr," *arXiv preprint arXiv:1611.02929*, 2016.

[22] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Model learning with local gaussian process regression," *Advanced Robotics*, vol. 23, no. 15, pp. 2015–2034, 2009.

[23] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," in *Advances in neural information processing systems*, 2006, pp. 1257–1264.

[24] J. Quiñonero-Candela, C. E. Rasmussen, A. R. Figueiras-Vidal *et al.*, "Sparse spectrum gaussian process regression," *Journal of Machine Learning Research*, vol. 11, no. Jun, pp. 1865–1881, 2010.

[25] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[26] M. A. Gelbart, J. Snoek, and R. P. Adams, "Bayesian optimization with unknown constraints," *arXiv preprint arXiv:1403.5607*, 2014.

[27] J. Bernardo, M. Bayarri, J. Berger, A. Dawid, D. Heckerman, A. Smith, and M. West, "Optimization under unknown constraints," *Bayesian Statistics 9*, vol. 9, p. 229, 2011.

[28] E. Pasolli and F. Melgani, "Gaussian process regression within an active learning scheme," in *Geoscience and Remote Sensing Symposium (IGARSS), 2011 IEEE International*, July 2011, pp. 3574–3577.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[30] "scikit-learn-0.18.dev0 - Gaussian Processes," http://scikit-learn.org/dev/modules/gaussian_process.html, accessed: 2017-04-18.

[31] "Multiprocessing – process-based threading interface," https://docs.python.org/2/library/multiprocessing.html, accessed: 2017-10-21.

[32] NERSC. Edison. http://www.nersc.gov/users/computational-systems/edison/. Accessed: 2017-04-18.

[33] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60.

[34] M. Bloodgood and J. Grothendieck, "Analysis of stopping active learning based on stabilizing predictions," *arXiv preprint arXiv:1504.06329*, 2015.

[35] P. Reisert, A. Calotoiu, S. Shudler, and F. Wolf, "Following the blind seer–creating better performance models using less information," in *European Conference on Parallel Processing*. Springer, 2017, pp. 106–118.