High-Dimensional Performance Modeling via Tensor Completion

Edward Hutter
Department of Computer Science
University of Illinois at Urbana-Champaign
hutter2@illinois.edu

Edgar Solomonik
Department of Computer Science
University of Illinois at Urbana-Champaign
solomon2@illinois.edu

***Abstract*—Performance tuning, software/hardware co-design, and job scheduling are among the many tasks that rely on models to predict application performance. We propose and evaluate low rank tensor decomposition for modeling application performance. We use tensors to represent regular grids that discretize the input and configuration domain of an application. Configuration execution times mapped within grid-cells are averaged and stored within tensor elements. We show that low-rank canonical-polyadic (CP) tensor decomposition is effective in approximating these tensors. We then employ tensor completion to optimize a CP decomposition given a sparse set of observed runtimes. We consider alternative piecewise/grid-based (P/G) and supervised learning models for six applications and demonstrate that P/G models are significantly more accurate relative to model size. Among P/G models, CP Decomposition of regular grids (CPR) offers higher accuracy and memory-efficiency, faster optimization, and superior extensibility via user-selected loss functions and domain partitioning. CPR models achieve a 2.18x geometric mean decrease in mean prediction error relative to the most accurate alternative models of size $\leq$10 kilobytes.**

***Index Terms*—performance modeling, tensor completion**

I. Introduction

A growing multiplicity of parameters influences application performance to enable performance portability and variable-resolution simulations across an increasingly diverse array of architectures. Application performance for a given set of inputs depends on both application-level parameters (e.g., block sizes, processor grid topology) and architectural parameters (e.g., processes-per-node, hyper-thread count). Interactions among these parameters motivate observation of an application's full configuration space [1], [2]. Performance models seek to model these effects and are increasingly relied upon by tasks such as program optimization [3]–[5], optimal tuning parameter selection [2], [6], architectural design [7], [8], dynamic load balancing, and machine allocation estimation (e.g., for weather forecasting and simulating virus spread dynamics).

The demand for increasingly accurate performance prediction is reflected in the abundance of available modeling frameworks [9]–[12]. Application performance models, which we survey in Section II, are typically either (1) global (non-piecewise) models, configured semi-analytically or automatically (such as via *multivariate adaptive regression splines* (MARS) [13]), (2) piecewise/grid-based models, which discretize the modeling domain, or (3) alternative models configured by supervised learning methods (such as neural networks). We instead consider tensor decompositions, which generalize low-rank matrix factorizations to higher dimensions [14] and are effective models for tensor completion [15], the problem of approximating a tensor given a subset of observed entries. Specifically, we employ tensor completion based on low-rank CP decomposition [16] for application performance modeling, and compare these models, which belong to type (2), to representative models of each type.

Performance models expressed as CP decompositions have the advantage of generalizing simple multilinear cost models. Discrete application performance is exactly expressible as a sum of piecewise multilinear cost functions, and rank-R CP decompositions attain an optimal R-term approximation, while providing systematic improvability (via increased rank). They also achieve linear model size with tensor order (number of application parameters) for a fixed rank. We provide specific details on how we partition the application parameter space to build a low-rank CP decomposition model in Section IV.

We pursue scale-independent minimization of prediction error, for which it is useful to consider alternative loss functions and data transformations. We contend in Section II that relative error offers a misleading assessment of performance model accuracy and propose the use of alternative error metrics. CP decompositions, which typically minimize the Frobenius norm of absolute errors, can directly minimize user-specified error metrics [17]. However, we find that CP decompositions optimized using least-squares loss functions with logarithmic transformation of execution times achieves superior accuracy. Our modeling software leverages efficient parallel optimization algorithms for tensor completion [18]–[20].

The most similar technique to CP tensor decomposition studied in prior work for application performance modeling is *sparse grid regression* (SGR) [21], [22]. SGR uses a hierarchical reduced-size model of a regular grid containing performance measurements. Both SGR and CP decomposition can compress performance across discrete modeling domains with high accuracy, but our experimental results in Section VI show that CP decomposition of a regular grid (CPR) offers improved accuracy and scalability in high-dimensional domains.

We use experimental analysis to demonstrate the efficacy of CP decomposition for application performance modeling. We generate datasets for computational kernels, communication

Metric	Mathematical Expression	Error-centric Expression
MAPE	$\sum_{k=1}^{M} \frac{\lvert m_k - y_k \rvert}{y_k}$	$\sum_{k=1}^{M} \lvert \epsilon_k \rvert$
MAE	$\sum_{k=1}^{M} \lvert m_k - y_k \rvert$	$\sum_{k=1}^{M} \lvert y_k \epsilon_k \rvert$
MSE	$\sum_{k=1}^{M} (m_k - y_k)^2$	$\sum_{k=1}^{M} (y_k \epsilon_k)^2$
SMAPE	$2\sum_{k=1}^{M} \frac{\lvert m_k - y_k \rvert}{y_k + m_k}$	$2\sum_{k=1}^{M} \lvert \frac{\epsilon_k}{2+\epsilon_k} \rvert$
LGMAPE	$\sum_{k=1}^{M} \log(\frac{\lvert m_k - y_k \rvert}{y_k})$	$\sum_{k=1}^{M} \log(\lvert \epsilon_k \rvert)$
MLogQ	$\sum_{k=1}^{M} \lvert \log(\frac{m_k}{y_k}) \rvert$	$\sum_{k=1}^{M} \lvert \frac{\epsilon_k}{1+\epsilon_k} \rvert + \mathcal{O}(\epsilon_k^2)$
MLogQ2	$\sum_{k=1}^{M} \log^2(\frac{m_k}{y_k})$	$\sum_{k=1}^{M} (\frac{\epsilon_k}{1+\epsilon_k})^2 + \mathcal{O}(\epsilon_k^4)$

TABLE I: Error metrics and corresponding expressions (scaled by M) given model predictions m, true outputs y, and errors $\epsilon = m/y - 1$. Mathematical and error-centric expressions are equivalent across rows 1-5, and equivalent to 1st-order approximation in ϵ_k across rows 6-7.

routines, and scientific applications (which invoke these kernels/routines) by executing randomly-selected configurations, which we use to train a collection of piecewise/grid-based and alternative supervised learning models. We assess the generated models by the accuracy and efficiency with which they predict unobserved performance and compress application performance, respectively. Low-rank CP decompositions are consistently more accurate relative to model size, and improve mean prediction accuracy by a geometric mean of 2.18x across six kernels/applications for models of size $\leq$10KB.

Our novel contributions towards performance modeling are as follows:

- a methodology for learning piecewise multilinear models via application of CP tensor decomposition,
- a software library for CPR that leverages open-source high-performance software for tensor completion,
- an evaluation of piecewise/grid-based models and alternative supervised learning methods using six applications,
- empirical evidence that CP decomposition models achieve highest prediction accuracy relative to model size.

II. Multi-Parameter Performance Modeling

Application performance models predict execution times given a configuration of application benchmark parameters. The structure of these models reflects application characteristics (e.g., execution requirements, scaling behavior) identified analytically and/or learned using measurements of executed configurations. We review methods to construct these models and summarize quantitative techniques to assess them.

A. Model Assessment

Performance models are assessed by size, trainability (e.g., model optimization costs), and the accuracy with which they predict application performance. Given a particular configuration of application parameters $(x_1, \ldots, x_d) \equiv \mathbf{x}$, measured execution time $y \equiv f(\mathbf{x})$, and a model output $m \equiv m(\mathbf{x})$, relative prediction error is given by $|m-y|/y$. Relative error is more appropriate than absolute error for assessing performance predictions spanning multiple orders of magnitude (e.g., kernel evaluation over a range of inputs), yet does not assign similar errors to mispredictions of a similar scale (i.e., $\log(m/y)$). For example, $\forall \tilde{m} : 0 < \tilde{m} \leq 0.5$, this metric will identify $\tilde{m}$ as a more accurate prediction of $y = 1$ than model output $m = 2$. Model selection via minimization of relative error thus exhibits bias towards under-prediction , which can misinform tasks that leverage model output e.g., to construct parallel schedules.

To assess performance prediction accuracy, we adopt *scale-independent* error metrics that assign equal error to under/over-predictions of the same scale. In particular, we penalize model outputs $m_k = ay_k$ and $\tilde{m}_k = y_k/a$ equally for positive factor a. Consider a collection of measured configurations $\{(\mathbf{x}_k, y_k)\}_{k=1}^{M}$ and model outputs $\{(\mathbf{x}_k, m_k))\}_{k=1}^{M}$. Among the aggregate error metrics listed in Table I, only the arithmetic mean of the absolute log accuracy ratios (MLogQ) and the arithmetic mean of the log-squared accuracy ratios (MLogQ2) achieve this interpretation of scale-independence [23], [24]. This table also demonstrates the equivalence of the symmetric mean absolute percentage error (SMAPE), the log-transformed geometric-mean relative error (LGMAPE), and MLogQ to first-order approximation for small relative errors.

B. Model Formulation

Analytic models decompose application execution into the execution times of computational and communication kernels executed along execution call paths [4], [10], [25]–[28]. Regression methods fit models to kernel benchmarking data or kernel execution costs (i.e., *requirements* models [4]) to instantiate analytic models of full applications. This composition of models facilitates incremental performance tuning and enables extrapolation from current to next-generation hardware. Many open-source cross-platform profiling frameworks (e.g., HPC-ToolKit, Scalasca, TAU) facilitate model generation by extracting statistical profiles of invoked kernels. However, domain expertise and static source code analysis alone have been shown to achieve optimal configuration selection for both non-distributed kernels [29]–[31] and distributed kernels [32], [33]. We evaluate regression methods for modeling application performance that do not rely on program analysis.

Clustering and correlation analysis guide the selection of application benchmark parameters. These statistical techniques identify both redundant parameters and those which correlate strongest with performance [7], [8]. The modeling domain (i.e., configuration space) is then the tensor product of the enumerated ranges of chosen parameters. The enumerated range of each numerical parameter (e.g., block size) is typically transformed (e.g., via logarithm) or normalized to enforce similar scales across parameters. Nominal and ordinal categorical parameter values (e.g., solver type) are mapped onto a range of real numbers with uniform spacing to supply a distance metric necessary for model optimization.

Domain expertise and/or additional offline statistical analysis guides the selection of an appropriate class of models. Global (non-piecewise) models capture distinct behaviors present across the full domain, while piecewise/grid-based models and supervised learning methods are well-suited for modeling more complex performance behavior [34]. Kernel methods, neural networks, and recursive partitioning methods configure models by exploiting dependencies among benchmark parameters, while instance-based methods (e.g.,

interpolation) configure models on-the-fly and rely solely on a specified distance metric. We summarize and evaluate representative models of each class.

Following the selection of class-specific hyper-parameters, a candidate model is configured by minimizing a particular aggregate error metric on measured execution times. The mean squared error (MSE), which is equivalent to the Frobenius norm of absolute errors and thus scale-dependent, is typically used as a loss function to select among candidate models. This least-squares regression assumes noisy execution data is sampled from a normal distribution and identifies the maximum-likelihood candidate, while alternative loss functions are considered for alternative underlying noise models. As we discuss in Section II-A, the desired assessment metric should also be considered when selecting a loss function.

C. Model Class

1) Global: Relationships between application benchmark parameters and execution data are expressed as a linear combination of explanatory variables (i.e., non-piecewise functions parameterized on a subset of benchmark parameters), each of which has global support across the configuration space. These variables model interaction effects and nonlinear scaling behavior among subsets of parameters. Ordinary least-squares regression (OLS) is a convex quadratic optimization problem that estimates unknown parameters expressed linearly with respect to explanatory variables using MSE loss (i.e., linear regression). Given a suitable collection of variables, the Gauss-Markov theorem guarantees that OLS will attain the maximum-likelihood model, provided the residuals between measured performance and model predictions are uncorrelated and exhibit zero-mean and constant-variance.

Expert insight and automated search methods have been proposed to find suitable explanatory variables. Discrete search methods circumvent requisite statistical analyses by searching within canonical classes of global models and performing OLS repeatedly on candidate models. In particular, the performance model normal form [5] expresses execution time as $m(\mathbf{x}) = \sum_{r=1}^{R} \alpha_r \prod_{j=1}^{d} x_j^{v_{r,j}} \log^{w_{r,j}}(x_j)$, (1) which instantiates a search space, the size of which is predicated on a user-specified set of rational numbers v, w.

2) Piecewise/Grid: Non-piecewise explanatory variables are insufficient for modeling complex performance behavior with high accuracy. Complex dependence of performance on input and configuration parameters arises even in simple programs due to e.g., memory misalignment, register spilling, and load imbalance, which motivates both empirical search and model-driven search as general strategies for optimal parameter selection [1]. This behavior can be more accurately modeled as a linear combination of spline functions, each of which consists of a tensor product of univariate functions defined piecewise by polynomials. These models introduce additional hyper-parameters, notably the selection of grid-points that partition the configuration space. Therefore, it is common to formulate global models, and only segment those parameters whose values correlate strongest with performance [7], [8].

Evaluation of performance on d-dimensional regular grids forgoes regression by modeling performance as a weighted average of 2^d neighboring grid-points, each of which stores the performance of a distinct configuration. For example, trilinear interpolation yields a prediction of a configuration (x_1, x_2, x_3)'s execution time as $\sum_{a=i_1}^{i_1+1} \sum_{b=i_2}^{i_2+1} \sum_{c=i_3}^{i_3+1} t_{a,b,c} \left(\frac{|x_1 - x_a|}{x_{i_1+1} - x_{i_1}} \right) \left(\frac{|x_2 - x_b|}{x_{i_2+1} - x_{i_2}} \right) \left(\frac{|x_{i_3} - x_c|}{x_{i_3+1} - x_{i_3}} \right)$, (2) where $x_{i_j} \leq x_j < x_{i_j+1}, \forall j \in \{1, 2, 3\}$ and $t_{a,b,c}$ denotes the execution time of grid-point (a, b, c) on a three-dimensional regular grid. Multilinear interpolation is suitable if performance behavior within grid-cells is sufficiently smooth. However, as observed in Section VI, this is rarely achieved without a significantly large number of observations.

Multiple frameworks, including *multivariate adaptive regression splines* (MARS) [13] and *sparse-grid regression* [21], [35], compress regular grids by selecting a subset of grid-points and corresponding spline functions automatically. MARS recursively constructs products of univariate hinge functions $h(x_i) = \{\mathrm{c}, \max(0, x_i - c)\}$, each characterized by a parameter x_i and a scalar position c within its range. The selection of these parameters involves repeatedly searching across the configuration space and invoking OLS as candidate models are constructed and pruned. MARS therefore relies on various heuristics to accelerate model construction in high-dimensional settings. As we show in Section VI, MARS performance models [36] can achieve only a coarse partitioning of the modeling domain and often instantiate global models.

Sparse-grid regression (SGR) instead discretizes high-dimensional continuous functions on anisotropic sparse-grids. Sparse-grids [37]–[39] are characterized by a hierarchical representation of piecewise-linear functions, the tensor-product of which yields piecewise d-linear basis functions. Only those functions with sufficiently-large support across the configuration space are retained. Sparse-grids initially utilize $\mathcal{O}(2^n n^{d-1})$ grid-points, where n is a user-specified discretization level. SGR models performance as a linear combination of $\mathcal{O}(2^n n^{d-1})$ multi-scale basis functions [22] and offers automated spatially-adaptive grid-refinement within the support of a specified number of basis functions [22]. Our methodology circumvents grid-refinement and compresses regular grids, which feature $\mathcal{O}(2^{nd})$ total grid-points, using low-rank tensor decompositions. We evaluate both methods in Section VI.

3) Kernel-based: Kernel methods project the modeling domain onto a higher-dimensional space characterized by user-specified kernel functions (e.g., sigmoid). Support vector machines (SVM) are global models formulated by partitioning the modeling domain so as to minimize a weighted distance between model predictions and training samples. Loss functions for SVM regression typically use MSE and are characterized by a threshold parameter ϵ such that prediction errors smaller than ϵ are not penalized. Gaussian process regression (GPR) assumes that observed execution data are samples of a multi-

variate normal distribution. The distributions are characterized by a mean vector and a positive-definite covariance matrix, the hyper-parameters of which are tuned by maximizing the log-marginal-likelihood using training data. We evaluate SVM kernel functions and GPR covariance kernels in this work.

4) Neural Networks: Multi-layer perceptrons model execution time as a composition of nonlinear activation functions [8], [40]. These models are constructed as stacks of layers, each of which consists of an array of units characterized by weights and an activation function. The output of each unit is used as an input to all units in the subsequent layer (i.e., fully-connected). Each activation function takes as input an inner product of a weight vector and those outputs. Optimization methods (e.g., back-propagation) optimize weights by minimizing a loss function using training data. These models feature a large design space, including layer size, number of layers, and activation function, each of which we tune in our evaluation. Neural networks have been shown to be effective at extrapolating small-scale performance [34], [41].

5) Recursive Partitioning: Decision trees partition the modeling domain into hyper-rectangles, each of which is assigned a constant value that models the performance of configurations mapped within its sub-domain. These hyper-rectangles are constructed by recursively splitting a parameter's range to minimize a loss function (e.g., MSE) on a subset of training data. Random forest regression (RTR) models performance as an average of these constants, each of which corresponds to a hyper-rectangle defined by a distinct decision tree. Each tree is constructed using a random sample of training data via bootstrap aggregation, while each tree's splits are constructed by minimizing a loss on a random sample of a subset of training data. Extremely-randomized tree regression (ETR) computes splits randomly rather than to minimize a specified error metric, Gradient-boosting regression (GBR) constructs trees sequentially by fitting residuals attained by the previously-constructed sequence of trees, which are proportional to the negative gradients of the MSE loss. In our experimental evaluation, we tune over a subset of hyper-parameters shared by each method, including forest size and tree-depth.

6) Instance-based: Instance-based models are constructed on-the-fly given a test configuration. k-nearest neighbors predicts execution time by interpolating the execution times of the k nearest training configurations per some specified distance metric. We vary k in our evaluation.

III. Tensor Completion

Tensor completion is the task of building a model to approximate a tensor based on a subset of observed entries [15]. These models are typically low-rank tensor decompositions. We review canonical-polyadic decomposition models and optimization methods for tensor completion in this section.

A. Low-rank tensor factorization models

A tensor $\mathcal{T} \in \mathcal{R}^{I_1 \times \cdots \times I_d}$ has order d (i.e., d modes/indices), dimensions I_1-by-...-by-I_d, and elements $t_{i_1,\ldots,i_d} = t_{\mathbf{i}}$ where $\mathbf{i} \in \otimes_{j=1}^{d}\{1,\ldots,I_j\}$. Tensors $\mathcal{T}$ exhibiting low-rank structure may be effectively modeled as a summation of tensor products (i.e., canonical-polyadic decomposition [16]) or using a high-order generalization of the singular value decomposition (i.e., Tucker decomposition) [14]. We leverage canonical-polyadic tensor decomposition, as it is relatively inexpensive to train, and leave exploration of other decompositions to future work.

The canonical-polyadic (CP) decomposition of an order three tensor $\mathcal{T} \in \mathcal{R}^{I_1 \times I_2 \times I_3}$ has the form $t_{i_1,i_2,i_3} =$

$$\sum_{r=1}^{R} u_{i_1,r} v_{i_2,r} w_{i_3,r} \equiv m_{i_1,i_2,i_3}, \tag{3}$$

where R denotes the rank of the decomposition, and $\mathbf{U} \in \mathcal{R}^{I_1 \times R}$, $\mathbf{V} \in \mathcal{R}^{I_2 \times R}$, $\mathbf{W} \in \mathcal{R}^{I_3 \times R}$ are factor matrices. The set of observed entries of an order-3 tensor $\mathcal{T}$ is represented by an index set $\Omega \subseteq \{1,\ldots,I_1\} \times \{1,\ldots,I_2\} \times \{1,\ldots,I_3\}$ so that every configuration $\mathbf{i} \equiv (i_1,i_2,i_3) \in \Omega$ has an associated measurement $t_{\mathbf{i}} \equiv t_{i_1,i_2,i_3}$. The size of these models grows linearly with tensor order for fixed dimensions and fixed rank, and linearly with CP rank for a fixed order and fixed dimensions. CP decomposition models generalize the class of global performance models in Equation 1 as multilinear cost functions $m(\mathbf{x}) = \sum_{r=1}^{R} u_r(x_1) v_r(x_2) w_r(x_3)$ composed of piecewise-linear functions $u_r, v_r, w_r, \forall r \in \{1,\ldots,R\}$.

B. Optimization methods

CP decomposition models must accurately represent observed entries, yet generalize effectively to unobserved entries. Optimizing these models involves minimizing an objective function that consists of a convex loss function and regularization terms in each factor matrix. Regularization decreases model parameter variance and thereby avoids over-fitting to observed entries within Ω. Choice of element-wise loss function ϕ reflects both the distribution of elements within Ω and the selected error metric. Given a training set Ω, CP rank R, and regularization parameter λ, tensor completion optimizes CP decompositions of order-3 tensors by minimizing objective function $f(U,V,W) = \lambda(||\mathbf{U}||_F^2 + ||\mathbf{V}||_F^2 + ||\mathbf{W}||_F^2) +$

$$\sum_{(i_1,i_2,i_3)\in\Omega} \phi(t_{i_1,i_2,i_3}, m_{i_1,i_2,i_3}). \tag{4}$$

We next summarize numerical methods that solve these nonlinear optimization problems for various loss functions.

1) Least-squares loss: Tensor completion fits CP decomposition models to partially-observed tensors by minimizing the least-squares loss function $\phi(t_{i_1,i_2,i_3}, m_{i_1,i_2,i_3}) = (t_{i_1,i_2,i_3} - m_{i_1,i_2,i_3})^2$. Both quadratic optimization and gradient descent have been applied to minimize Equation 4 instantiated with this loss function. The alternating least-squares method (ALS) sweeps over the rows of each factor matrix, fixing the elements of all but one and minimizing, over the i_1'th row of $\mathbf{U}$, the convex objective $f(\mathbf{u}_{\mathbf{i_1}}) =$

$$\frac{1}{|\Omega_i|}\left[\sum_{(i_2,i_3)\in\Omega_i} (t_{i_1,i_2,i_3} - m_{i_1,i_2,i_3})^2\right] + \lambda||\mathbf{u}_{i_1}||_2^2, \tag{5}$$

where $(i_2,i_3) \in \Omega_{i_1}$ if and only if $(i_1,i_2,i_3) \in \Omega$. Minimization of each row-wise objective involves solving a linear least-

squares problem. The arithmetic complexity of alternating least-squares (ALS) scales cubically with CP rank R and linearly with tensor mode-lengths I. This overhead may be circumvented by optimizing factor matrix elements individually. In particular, the coordinate descent method (CD) minimizes the scalar objective $f(u_{i,r})$ and thereby reduces the arithmetic complexity of ALS by a factor of R. Both ALS and CD achieve monotonic convergence, yet CD often exhibits slower convergence due to its decoupling of row-wise factor matrix updates. Stochastic gradient descent instead iteratively updates all factor matrix elements at once and calculates the partial derivative of the corresponding objective using a random subset of observations within Ω. The performance of these methods has been extensively studied for both matrix [42]–[48] and tensor completion [18]–[20].

2) Generalized loss functions: Tensor completion may fit CP decomposition models to partially-observed tensors by minimizing alternative loss functions [17], which generalizes tensor completion to a broader class of datasets. The corresponding loss functions often indirectly apply constraints to the elements within factor matrices (e.g., non-negativity), which are enforced by incorporating additional terms (e.g., penalty and barrier functions) into the objective. Alternating minimization and coordinate minimization may apply Newtons method to optimize row-wise and scalar subproblems, respectively. In particular, alternating minimization via Newtons method (AMN) updates initial row-vector iterates $\mathbf{u}_{i_1}^{(0)}$ as follows until convergence:

$$\mathbf{u}_{i_1}^{(l+1)} \leftarrow \mathbf{u}_{i_1}^{(l)} - \mathbf{H}_f^{-1}(\mathbf{u}_{i_1}^{(l)}) \nabla f(\mathbf{u}_{i_1}^{(l)}), \tag{6}$$

where $\nabla f(\mathbf{u}_{i_1}^{(l)}) = \sum_{(i_2,i_3)\in\Omega_i} \nabla\phi(\mathbf{u}_{i_1}^{(l)}) + 2\lambda\mathbf{u}_{i_1}^{(l)}$, $\mathbf{H}_f(\mathbf{u}_{i_1}^{(l)}) = \sum_{(i_2,i_3)\in\Omega_i} \mathbf{H}_\phi(\mathbf{u}_{i_1}^{(l)}) + 2\lambda\mathbf{I}$, and $\mathbf{H}_f, \mathbf{H}_\phi$ are Hessian matrices. Gradient-based methods such as stochastic gradient descent and L-BFGS have also been proposed [17], as have Gauss-Newton and quasi-Newton methods that also involve optimizing all factor matrices with each iteration [18]. We apply generalized CP decomposition via AMN to minimize alternative error metrics (e.g., relative error metrics described in Section II-A). Specifically, we leverage existing work [18] and the corresponding parallelization strategies therein, with minor modifications to account for non-negativity.

IV. High-Dimension Performance Modeling via Tensor Completion

We apply tensor completion to optimize low-rank canonical-polyadic (CP) tensor decompositions of application performance. Our modeling framework enables user-specified partitioning of the modeling domain and provides a collection of loss functions to minimize user-specified error metrics. In this section, we describe model training and inference for accurate and scalable high-dimensional performance prediction.

A. Tensor model formulation

We first map discrete d-parameter configuration spaces onto d-dimensional regular grids. The boundaries of each grid-cell

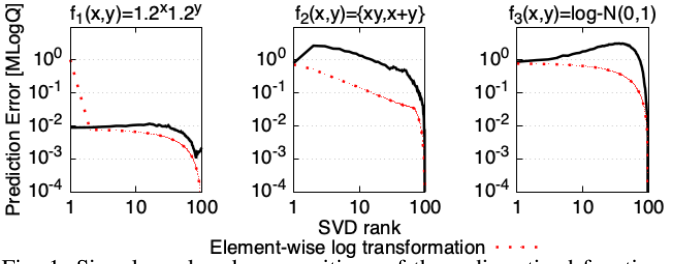

Fig. 1: Singular value decompositions of three discretized functions evaluated for $1 \leq x, y \leq 100$. The two behaviors of f_2 are split along $x+y \leq 100$. Each element of f_1, f_2 is multiplied by $(1+\mathcal{N}(0,.01))$, where $\mathcal{N}(\mu,\sigma)$ denotes a sample from a normal distribution.

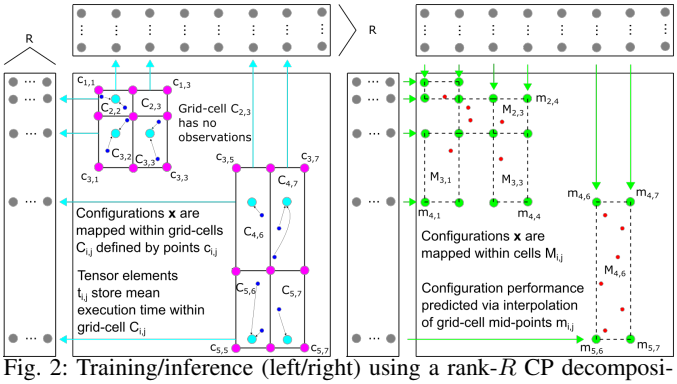

Fig. 2: Training/inference (left/right) using a rank-R CP decomposition model ●. Tensor element ● is the sample mean of the performance of intra-cell training configurations ●. Interpolation of approximate tensor elements ● models performance of test configuration ●.

are defined by 2^d grid-points, each of which corresponds to a distinct configuration. In particular, we formulate regular grids $\mathcal{G} \equiv \otimes_{j=1}^{d} \tilde{X}_j$ as tensor products of sub-sequences $\tilde{X}_j$ of each parameter's enumerated range of values $X_j : \tilde{X}_j \subseteq X_j$. Each grid-point $\mathbf{c_i} \in \mathcal{G}$ is uniquely identified by an index vector $(i_1, \ldots, i_d) \equiv \mathbf{i} \in \mathcal{Z}_+^d$, and its l'th element by scalar $c_{\mathbf{i}}^l \in \tilde{X}_j : c_{\mathbf{i}}^l \equiv \tilde{X}_j^l$. An arbitrary input $\mathbf{x} \equiv (x_1, \ldots, x_d) \in \otimes_{j=1}^{d} X_j$ is mapped within a d-dimensional grid-cell $\mathcal{C}_{\mathbf{i}} \equiv \otimes_{j=1}^{d} (\mathbf{c_i}, \mathbf{c_{i+e}}_j)$, which is defined as the tensor product of a pair of neighboring grid-points $(\mathbf{c_i}, \mathbf{c_{i+e}}_j)$ such that $c_{\mathbf{i}}^j \leq x_j < c_{\mathbf{i+e}_j}^j, \forall j \in \{1, \ldots, d\}$, where $\mathbf{e}_j$ denotes the j'th basis vector in the coordinate vector space.

We formulate tensors $\mathcal{T} \in \mathcal{R}_+^{I_1 \times \cdots \times I_d}$ from these regular grids by assigning the sample mean execution time across all configurations within a grid-cell $\mathcal{C}_{\mathbf{i}}$ to a distinct tensor element $t_{\mathbf{i}} \in \mathcal{T}$. Each tensor element thus approximates the execution time of the mid-point of its assigned cell and corresponds to an observed entry within Ω following notation from Section III. Tensor order d reflects the dimension of its underlying regular grid. Tensor mode-length I_j is given by $I_j \equiv |\tilde{X}_j| - 1$.

A grid-cell bounded instead by 2^d mid-points of neighboring grid-cells is constructed on-the-fly to facilitate model inference. This grid-cell $\mathcal{M}_{\mathbf{i}} \equiv \otimes_{j=1}^{d} (\mathbf{m_i}, \mathbf{m_{i+e}}_j)$ is the tensor product of the pair of mid-points of neighboring grid-cells $(\mathbf{m_i}, \mathbf{m_{i+e}}_j)$ such that $m_{\mathbf{i}}^j \leq x_j < m_{\mathbf{i+e}_j}^j, \forall j \in \{1, \ldots, d\}$. Each cell mid-point $\mathbf{m_i}$ is uniquely identified by an index vector $(i_1, \ldots, i_d) \equiv \mathbf{i} \in \mathcal{Z}_+^d$, and its l'th element by scalar $m_{\mathbf{i}}^l \equiv$

$\hat{X}_j^l$, where $\hat{X}_j^l$ is a mid-point of each parameter's sub-sequence $\tilde{X}_j$. Specifically, $\hat{X}_j^l = h_j^{-1}(h_j(\tilde{X}_j^l) + h_j(\tilde{X}_j^{l+1}))/2)$, where uniform and logarithmic spacing along the j'th tensor dimension uses $h_j(x) := x$ and $h_j(x) := \log(x)$, respectively. The endpoints of each parameter's range are included. We set $\tilde{X} \equiv X$ for categorical parameters, which have no mid-points.

Tensor model inference interpolates neighboring tensor elements. In particular, following identification of the two closest mid-points along each dimension, this model approximates the execution time $f(\mathbf{x})$ of configuration $\mathbf{x}$ as $\tilde{m}(\mathbf{x}) =$

$$\sum_{\mathbf{a}\in\{0,1\}^d} t_{\mathbf{i}+\mathbf{a}} \cdot \prod_{j=1}^{d} \left(\frac{|h(x_j) - h(m^j_{\mathbf{i}+\mathbf{a}})|}{h(m^j_{\mathbf{i}+\mathbf{e}_j}) - h(m^j_{\mathbf{i}})} \right), \tag{7}$$

where $m^j_{\mathbf{i}} \leq x_j < m^j_{\mathbf{i}+\mathbf{e}_j}, \forall j \in \{1,\ldots,d\}$. The selection of grid-spacing and grid-size $|\tilde{X}_j|$ along each dimension should facilitate accurate compression of tensor $\mathcal{T}$ with small CP rank. Our modeling framework therefore enables user-directed domain partitioning via specification of uniform, logarithmic, or custom grid-spacing to balance interpolation error and low-rank approximation error.

B. CP decomposition model formulation

We optimize low-rank CP decompositions of tensors $\mathcal{T}$ to achieve scale-independent minimization of prediction error. We first transform observed entries to enable the use of the MSE loss function, which among those summarized in Section II-A is most efficient and least susceptible to roundoff error. Specifically, we instantiate Equation 4 with $\phi(t_{\mathbf{i}}, m_{\mathbf{i}}) = (\log(t_{\mathbf{i}}) - m_{\mathbf{i}})^2$. This simple technique minimizes deviation in the scale of observed runtimes and thereby enhances the robustness of MSE to both noise and biased under-prediction. We show this in Figure 1 using singular value decompositions, which minimize $\sqrt{\text{MSE}}$. An increase in error with larger SVD rank signifies negative reconstructed matrix entries reset to 10^{-16} prior to evaluation of MLogQ. Use of logarithmic transformations attains non-negative CP decomposition model output implicitly without the requisite constraints to enforce non-negativity in factor matrix elements.

The resulting model approximates the execution time $f(\mathbf{x})$ of configuration $\mathbf{x}$, which we modify to feature $\hat{d} < d$ numerical benchmark parameters, as $m(\mathbf{x}) =$

$$\sum_{\mathbf{a}\in\{0,1\}^{\hat{d}}} \mathrm{e}^{m_{\mathbf{i}+\mathbf{b}}} \cdot \prod_{j=1}^{\hat{d}} \left(\frac{|h(x_j) - h(m^j_{\mathbf{i}+\mathbf{b}})|}{h(m^j_{\mathbf{i}+\mathbf{e}_j}) - h(m^j_{\mathbf{i}})} \right), \tag{8}$$

where $\mathbf{b} \equiv \mathbf{a} \oplus \{0\}^{d-\hat{d}}$ and $m^j_{\mathbf{i}} \leq x_j < m^j_{\mathbf{i}+\mathbf{e}_j}, \forall j \in \{1,\ldots,\hat{d}\}$. Our modeling framework supports user-specified loss functions given gradient and Hessian information, as well as data transformations, the inverse of which is applied upon model inference. We summarize training and inference across a two-dimensional domain in Figure 2.

V. Experimental Methodology

1) Machine: We use the Stampede2 supercomputer at Texas Advanced Computing Center (TACC) [53]. Stampede2 consists of 4200 Intel Knights Landing (KNL) compute nodes (each capable of a performance rate over 3 Teraflops/s) connected by an Intel Omni-Path (OPA) network with a fat-tree topology (achieving an injection bandwidth of 12.5 GB/sec). Each KNL compute node provides 68 cores with 4 hardware threads per core. We use Intel environment 18.0.2, Intel MPI environment 18.0.2, and corresponding icc, mpicc, and mpicxx compilers. We generate all models using a single KNL core. We benchmark applications using ≤ 64 KNL nodes and vary the number of processes-per-node and threads-per-process.

2) Application Libraries: We execute Intel MKL's GEMM (MM) routine $C_{m\times n} \leftarrow \alpha A_{m\times k} B_{k\times n} + \beta C_{m\times n}$ in a single-threaded environment with matrix dimensions $32 \leq m,n,k \leq 4096$. We execute SLATE's QR factorization (QR) routine [50] $A_{m\times n} \leftarrow Q_{m\times n} R_{n\times n}$ on $p = \{1,4,16,64\}$ nodes, first with 64 single-threaded processes-per-node (ppn), matrix dimensions $2^{10}\sqrt{p} \leq m \leq 2^{16}\sqrt{p}, 2^4\sqrt{p} \leq n \leq 2^{10}\sqrt{p}$, and tuning parameters for processor-grid $p_r \times p_c$ and outer/register block sizes *ob*/*ib* chosen such that $p_c = \min(\lfloor\sqrt{64p}\rfloor, \lfloor n/256\rfloor)$, $ob = \min(128, \lfloor n/p_c\rfloor)$, and $ib = \min(16, \lfloor n/p_c\rfloor)$. We then extend the dimension of the configuration space by varying each tuning parameter, yet still mapping a thread to each available core. We execute Intel MPI's Broadcast (BC) routine on $p = \{1,2,4,8,16,32\}$ nodes with ppn $n = \{1,2,4,8,16,32,64\}$, and message size $2^6 \leq m \leq 2^{24}$. ExaFMM [49] is an open source library for fast multipole methods aimed towards Exascale systems. We execute ExaFMM configurations on a single-node and model *Total FMM* time. SNAP [51] and Kripke [52] are proxy applications for modern discrete ordinates neutral particle transport applications. We execute configurations across multiple hardware configurations and model each library's *Solve Time*. Table II details each configuration space CS and defines $|CS|$ as the configuration space dimension (number of evaluated parameters).

3) Dataset Generation: To optimize and evaluate performance models, we generate training sets of size N and test sets of size M for each kernel/application, respectively. The test set for Kripke features 250 configurations, while all others feature 900 configurations. Individual configurations within each set are formulated using distinct sampling strategies for distinct parameter types. In particular, the specified ranges of both input parameters (e.g., matrix dimension, message size) and architectural parameters (e.g., node count, ppn) are sampled log-uniformally at random, while the ranges of each application-level tuning parameter (e.g., block size) are sampled uniformally at random. We optimize models using a random sample from each training set and log-transform execution times and configuration parameter values. Each kernel configuration is executed 50x or until the coefficient of variation of its corresponding execution time is < 0.01. Each application configuration is executed once.

4) Model Assessment: We assess model prediction error using MLogQ as detailed in Section II-A. We assess model size by writing configured models to a file via Python's *joblib* package and measuring the file size. We assess model time by measuring the time to configure a model on a single core. We

Library	**Application** [parameter description]	$\|CS\|$	Input	Architectural	Application-level
ExaFMM [49]	**Fast Multiple Method** [number of particles n, expansion order ord, #threads, particles per leaf ppl, tree level used for partitioning tl]	5	$2^9 \leq n \leq 2^{13}$, $0 \leq ord \leq 8$	$2^0 \leq \#threads \leq 2^6$	$4 \leq ppl \leq 128$, $0 \leq tl \leq 4$
SLATE [50]	**QR factorization** [matrix $m \times n$, #nodes p, processes-per-node ppn, outer block size ob, inner block size ib, processor grid $p_r \times p_c$]	7	$2^{10}\sqrt{p} \leq m \leq 2^{16}\sqrt{p}$, $2^4\sqrt{p} \leq n \leq 2^{10}\sqrt{p}$	$2^0 \leq p \leq 2^6$, $2^3 \leq ppn \leq 2^6$	$2^4\sqrt{p} \leq ob \leq 2^7\sqrt{p}$, $2^3 \leq ib \leq 2^6$, $2^1 \leq p_c \leq 2^5$
SNAP [51]	**Discrete ordinates transport** [see [51] for descriptions of {*nmom*, *nang*, *ng*, *ichunk*}, #nodes *N*, processes-per-node *ppn*, #threads/process *tp*]	7	$2^0 \leq nmom \leq 2^2$, $2^5 \leq nang \leq 2^{12}$, $2^3 \leq ng \leq 2^8$	$2^0 \leq N \leq 2^4$, $2^5 \leq ppn \leq 2^6$, $2^0 \leq tp \leq 2^2$	$2^0 \leq ichunk \leq 2^3$
KRIPKE [52]	**Discrete ordinates transport** [see [52] for descriptions of {*groups*, *legendre*, *quad*, *zones*, *dset*, *gset*, *zset*, *layout*, *solver*}, #nodes *N*, processes-per-node *ppn* #threads/process *tp*]	12	$2^3 \leq groups \leq 2^7$, $0 \leq legendre \leq 5$, $2^3 \leq quad \leq 2^7$, $2^3 \leq zones \leq 2^6$	$2^0 \leq N \leq 2^4$, $2^5 \leq ppn \leq 2^6$, $2^0 \leq tp \leq 2^2$	6 layouts, 2 solvers, $2^3 \leq dset \leq 2^5$, $2^0 \leq gset \leq 2^6$, $2^0 \leq zset \leq 2^3$

TABLE II: Configuration space (*CS*) description for parallel QR factorization and parallel scientific applications involving particle interaction and particle transport. Mapping of parameters to tensor modes reflects ordering in *parameter description*. See citations for further clarification.

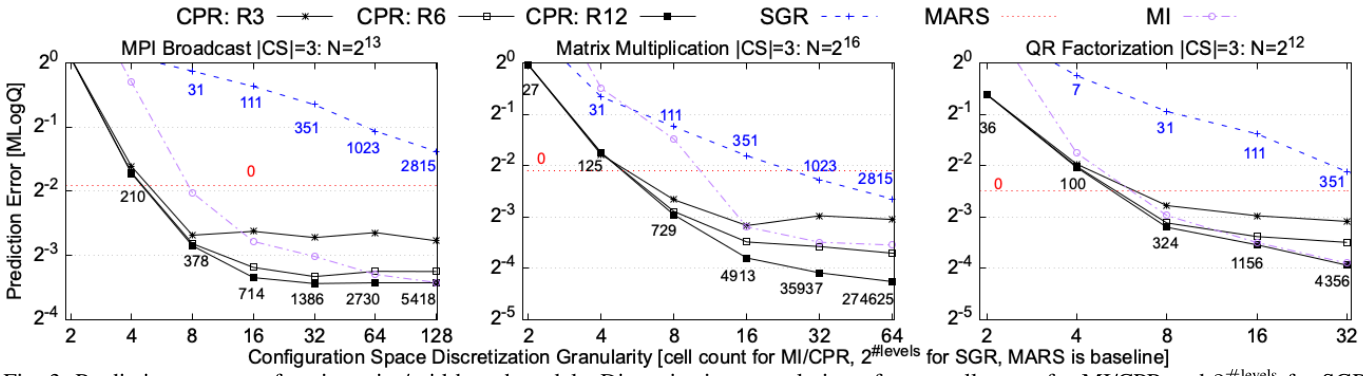

Fig. 3: Prediction accuracy for piecewise/grid-based models. Discretization granularity refers to cell count for MI/CPR and $2^{\#\text{levels}}$ for SGR. Rk denotes CP rank k. |CS| denotes #parameters. N denotes training-set size. Annotations show #grid-points (CPR/MI are same).

forgo consideration of models optimized in ≥ 1200 seconds in Figures 6,8, or models ≥ 1 megabyte in Figures 7,8.

5) Model Tuning: We evaluate the following prediction methods: grid-based multilinear interpolation (MI), sparse grid regression (SGR), multi-layer perceptrons (NN), random forests (RFR), gradient-boosting (GBR), extremely-randomized trees (ETR), Gaussian process regression (GPR), support vector machines (SVM), adaptive spline regression (MARS), k-nearest neighbors (KNN), and our methodology described in Section IV (CPR). We utilize the Cyclops Tensor Framework [54] for efficient CP decomposition optimization. We utilize the PyEarth library to evaluate MARS, the SG++ [21] library to evaluate SGR, and the Scikit-Learn [55] library to evaluate the remaining models. We evaluate all relevant model configurations using the same training set and forgo training via cross-validation. In particular, we evaluate CP rank $1 \rightarrow 16$ and grid-cell count $2 \rightarrow 256$ for CPR; $2 \rightarrow 8$ discretization level, $1 \rightarrow 16$ refinements, and $4 \rightarrow 32$ adaptive grid-points for SGR; max spline degree $1 \rightarrow 6$ for MARS; max tree depth $2 \rightarrow 16$ and max tree count $2 \rightarrow 256$ for RFR, GBR, and ETR; $1 \rightarrow 6$ neighbor count for KNN, out-of-box covariance kernels for GPR; {poly,rbf} kernels and degree $1 \rightarrow 3$ for SVM; layer size $2 \rightarrow 8192$ and $1 \rightarrow 8$ # of hidden layers for NN. CPR and MI models place grid-points equally-spaced on a log-scale/linear-scale along the ranges of input,architectural/application-level parameters, and interpolate as necessary. We select regularization parameter $\lambda = 10^{-k}, \forall k \in \{-6, \ldots, -1\}$ for CPR and SGR. We set a maximum number of 100 sweeps and 1000 iterations of alternating least-squares and conjugate gradient for CPR and SGR, respectively, and set a tolerance of $1e-4$ for SGR.

VI. Experimental Evaluation

We evaluate models generated by methods described in Sections II-C and tuned as specified in Section V-5. We ablate piecewise/grid-based models by varying the structure and refinement of underlying grids. We then incorporate alternative supervised learning methods to compare model accuracy as a function of model size and training set size. These studies highlight the CP rank, tensor density, and tensor size with which CP decompositions outperform alternative models.

A. Ablation Studies for Piecewise/Grid-based Models

We observe in Figure 3 that models able to interpolate regular grids (multilinear interpolation (MI), CP decomposition of regular grids (CPR)) achieve higher prediction accuracy than both anistropic grid models (sparse-grid regression (SGR)) and

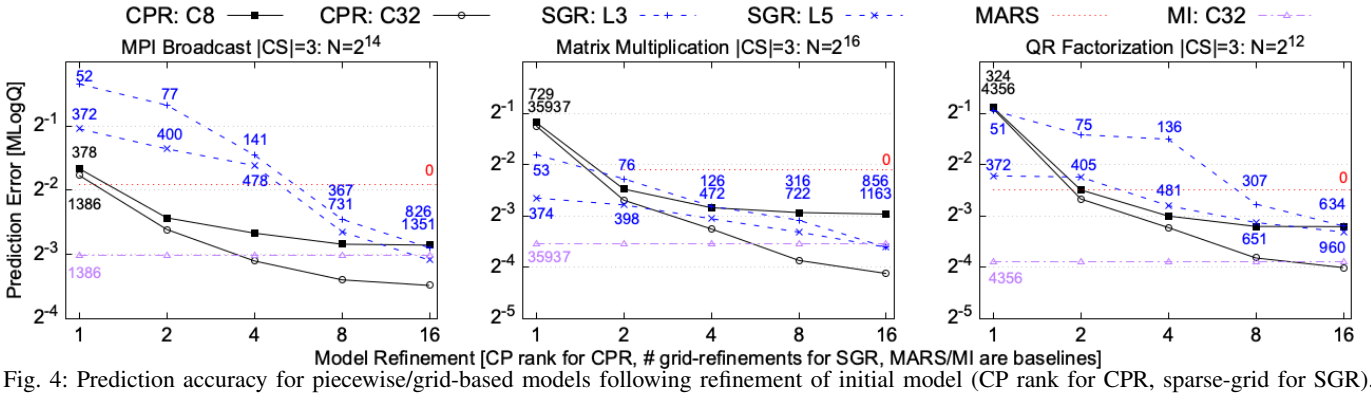

Fig. 4: Prediction accuracy for piecewise/grid-based models following refinement of initial model (CP rank for CPR, sparse-grid for SGR). Ck/Lk denote cell count/sparse-grid-level k. |CS| denotes #parameters. N denotes training-set size. Annotations show #grid-points.

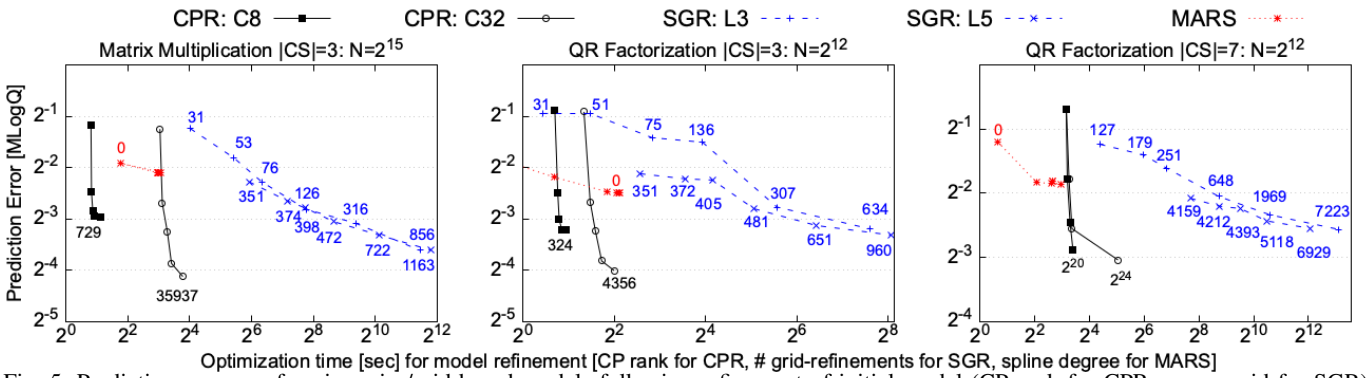

Fig. 5: Prediction accuracy for piecewise/grid-based models following refinement of initial model (CP rank for CPR, sparse-grid for SGR). Ck/Lk denote cell count/sparse-grid-level k. |CS| denotes #parameters. N denotes training-set size. Annotations show #grid-points.

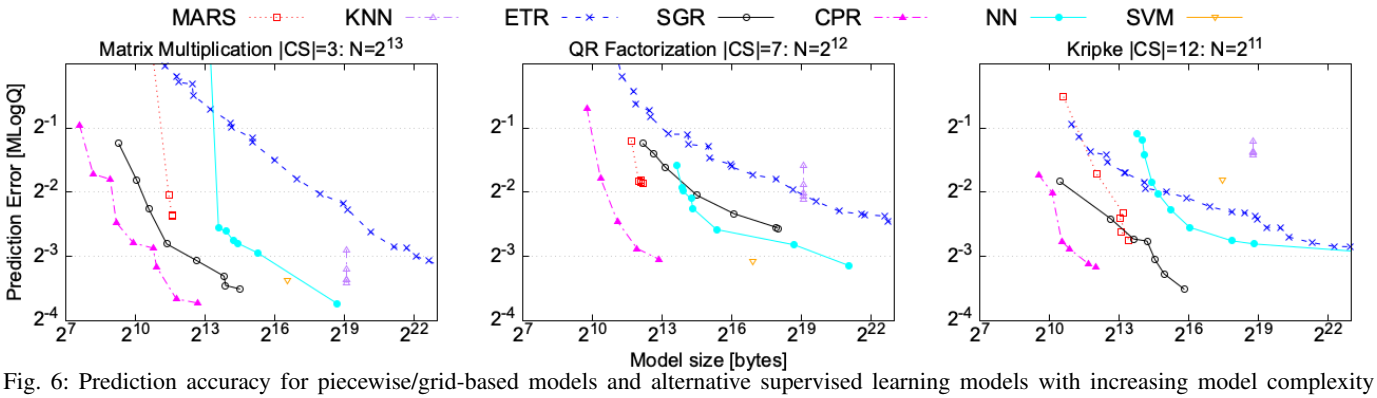

Fig. 6: Prediction accuracy for piecewise/grid-based models and alternative supervised learning models with increasing model complexity achieved by varying hyper-parameters described in Section V-5. |CS| denotes #parameters. N denotes training-set size.

global models with high-degree polynomials (MARS) across all three kernels. In particular, MI and CPR achieve superior accuracy using underlying grids/tensors of size kx6x7, kxkxk, and kxkx4 with $k \geq 8$ for the MPI Broadcast (BC), matrix multiplication (MM), and QR factorization (QR) kernels, respectively. Accuracy continues to improve even as tensors become increasingly sparse. In particular, the mean error of a CP rank $R = 12$ CPR model for the MM kernel is 2x smaller from $k = 16$ to $k = 128$ yet the density of the underlying tensor is 8x smaller ending at 12.5%. In comparison, MARS ineffectively searches for grid-points across the entire space, while the structured grid used by SGR does not map well to the underlying performance of any kernel without automated refinement. We conclude that CPR is the only method able to efficiently and systematically reduce interpolation error given user-specified grid structure.

We observe in Figure 3 that CPR models are consistently more accurate than MI despite being exponentially smaller in size. Both MI and CPR predict unobserved performance by interpolating nearby grid-points and tensor elements, respectively, yet CPR does so with averaged intra-cell performance. MI instead interpolates the performance of distinct configurations, which increases susceptibility to outliers. These interpolation strategies remain effective as tensor dimension k

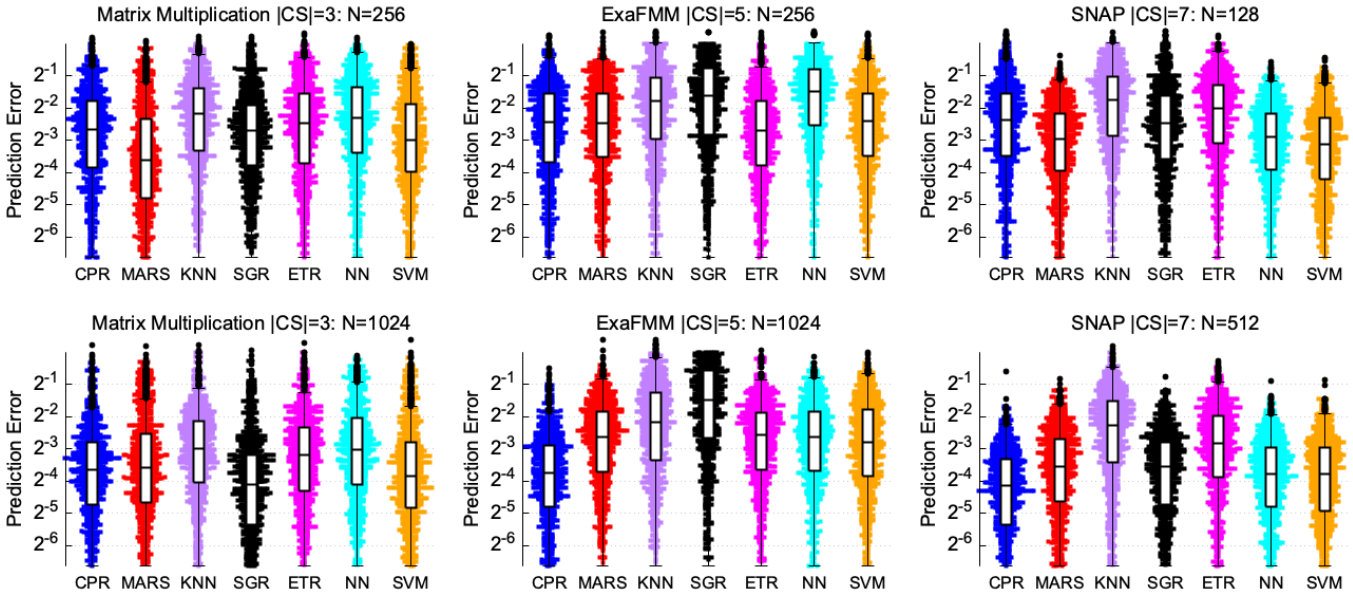

Fig. 7: Prediction accuracy of optimal model candidate selected by tuning (see Section V-5). Dots comprising each violin plot denote predictions on test-set configurations. Lines within boxplots denote median error. |CS| denotes #parameters. N denotes training-set size.

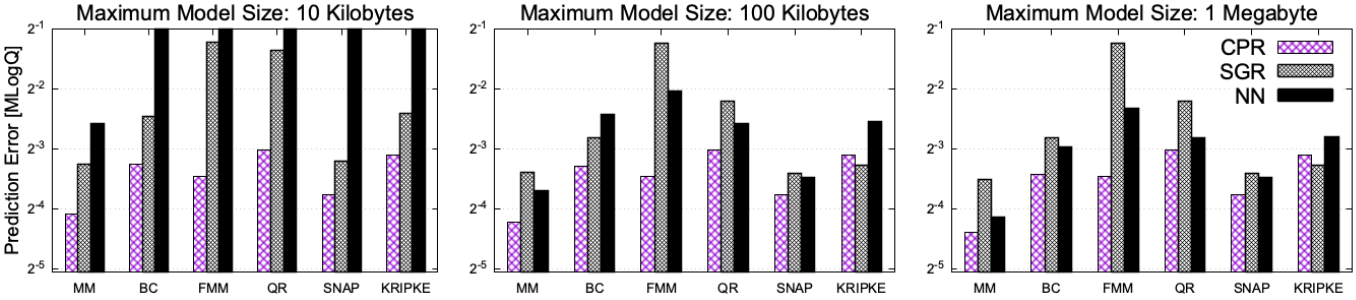

Fig. 8: Prediction accuracy for most accurate CP decomposition of regular grid (CPR), sparse grid (SGR), neural network (NN) models. Kernel/Application names are described in Section V. Training-set sizes: $(2^{14}, 2^{15}, 2^{16})$, $(2^{11}, 2^{12}, 2^{13})$, $2^{10}, 2^{12}, 2^{10}, 2^{11}$, respectively.

increases, yet CPR models can only leverage the decreasing interpolation error given sufficient CP rank. We observe that CP rank $R = 12$ is sufficient to mitigate low-rank approximation error, and systematic improvements in model accuracy re visible with increasing CP rank. In particular, Figure 4, which evaluates the same kernels as Figure 3, highlights that a CP rank $R : R < 17$ is sufficient to mask approximation error and reduce interpolation error beyond that achieved by MI with a size $k = 32$ grid.

We observe in Figure 5 that the most accurate CPR models take less time to optimize than the most accurate sparse-grid models. An important distinction between the grid-refinement strategies of each method is that, following specification of an initial grid, sparse-grid refinement via SGR is inherently sequential. SGR's refinement mechanism sweeps over training data to identify grid-points and corresponding basis functions contributing most to training error and increases the number of grid-points surrounding those grid-points, before restarting the procedure on the newly refined grid. As observed in Figures 3 and 4, sparse-grid models cannot achieve competitive accuracy without sufficient refinement for 3-dimensional configuration spaces. The underlying regular grid of a CPR model is instead fixed, and refinement consists solely of increasing the number of tensor products to achieve an improved fit for all tensor elements, or increasing the number of ALS sweeps to optimize for a particular CP rank. In the limit of increasing refinements, anisotropic grids become increasingly regular, and we demonstrate in Figure 4 that compression of regular grids via CP decomposition is more effective, especially in high-dimensional settings.

The scalability of CPR model optimization is evident in Figure 5 by varying both the training-set size N and the configuration space dimension $|CS|$. A rank $R = 8$ CP model for MM with $N = 65536$ achieves better accuracy and is > 256x faster to optimize. We observe a smaller speedup when reducing to training set to $N = 4096$ for QR with 3D modeling domain, yet a similar speedup is attained solely by increasing in dimension of modeling domain to 7D. Overall, we observe that CPR models can achieve higher accuracy than established piecewise/grid-based models, especially for

vendor-optimized kernels in high-dimensional settings, while relying on compressed and unrefined regular grids.

The most accurate CPR models for MM, QR, and Kripke in Figure 6 utilize CP ranks 16, 8, and 10 to model tensors of size 16x16x16, 32x32x4x4x16x8x6, and 5x5x3x3x3x2x6x6x2x4x4x4 that are 83%, 0.03%, and 0.03% dense, respectively. Thus, more than 99.9% of grid-cells are unobserved when modeling QR and Kripke, whereas the underlying tensor for GEMM is nearly dense. This disparity indicates that accurate CP decompositions are attainable as long as the size of each tensor mode's underlying projection sets is sufficiently large, despite large tensor sparsity and the presence of constraints among benchmark parameters. For a fixed training set size N, CPR models with fewer underlying grid-cells can utilize larger CP ranks, which we find a suitable trade-off when modeling the performance of GEMM. In particular, the projection sets of $kxkxk$ tensors given $N = 8192$ are too small to optimize a CP decomposition that further reduces test error if $k > 16$. However, further reduction is achieved in Figure 3 between $16 \leq k \leq 64$ with $N = 65536$. In contrast, we place significantly more grid-points along the higher-dimensional domains (e.g., those of Kripke, ExaFMM, SNAP) and decompose the corresponding tensors with smaller CP rank to account for small $N \leq 4096$.

B. Comparison Studies for Supervised-Learning Methods

We observe in Figure 6 that CPR models achieve highest prediction accuracy relative to model size for MM, QR, and Kripke. Alternative models require a 20.2x geometric mean factor increase in model size to achieve equivalent error. We further observe in Figure 7 that accurate CPR models can be optimized with small training sets, and that CPR model accuracy systematically improves with training set size. However, alternative models are more competitive in this regime, indicating that CPR is most effective relative to the state-of-the-art for training sets of size $N \geq 1000$ yet independent of the dimension of the modeling domain. In particular, for the first row in Figure 7, CPR models achieve a geometric mean decrease in prediction error of 0.68x for training sets of size $N \leq 256$, while for the second row that factor increases to 1.18x for $N \geq 512$. Finally, we highlight in Figure 8 that CPR models achieve a geometric mean decrease in prediction error of (2.18x,1.40x,1.35x) relative to the most accurate sparse-grid and neural network models of maximum size (10,100,1000) kilobytes, respectively, for sufficiently large training sets $N \geq 1024$. Other supervised learning methods cannot achieve competitive accuracy when considering *all* kernels/applications, despite using significantly larger models.

SGR is most competitive with CPR relative to model size, yet the automated and irregular placement of grid-points following iterative grid-refinement is not appropriate for all applications (e.g., ExaFMM in Figures 7,8) and requires expensive mapping logic which partially negates the storage benefits of sparse grids in high-dimensions. Further, SGR cannot efficiently leverage larger training sets to further reduce error via grid-refinement, as shown in Figure 5. Compositions of nonlinear functions (e.g., neural network (NN)) model complex performance behavior less directly than piecewise multilinear cost functions, and compensate by utilizing significantly larger models. However, as shown in Figure 8, neural networks are most competitive with CPR for large training sets, which both can efficiently utilize to reduce model error.

Non-piecewise polynomial cost models (e.g., MARS) cannot fit to complex performance data irregardless of specified spline degree, and thus cannot systematically improve model accuracy. Recursive partitioning methods (e.g., extremely-randomized tree regression (ETR)) can partition the modeling domain as a regular-grid, yet feature no compression mechanism. The alternative partitions it learns to retain scalability are consequently less accurate. We observe offline that ETR and support vector machines (SVM) are consistently more accurate and no less memory-efficient than other partitioning methods (e.g., gradient-boosting, random forest regression) and Gaussian-process regression, respectively. Instance-based methods (e.g., k-nearest neighbors (KNN)) achieve increasingly poor accuracy in higher dimensions due to increasingly sparse observation of the modeling domain.

VII. Conclusion

Our experimental analysis demonstrates that complex performance behavior is most efficiently modeled using CP decomposition models. These piecewise multilinear cost models are significantly more accurate than alternative models relative to model size and can both systematically and efficiently reduce prediction error with increasingly large training sets. Further, CP models are significantly more memory-efficient, more extensible, and easier to optimize than other piecewise/grid-based models, as user-directed domain partitioning enables application-specific and space-efficient tensor formulation through judicious placement of grid-points.

This work presents promising avenues for autotuning research involving the use of tensor factorizations as surrogate models of application performance. The prevalence of highly-constrained configuration spaces promotes further investigation into the use of tensor completion applied to datasets with structured sparsity. The efficacy of tensor completion applied to transfer learning tasks (e.g., extrapolation to unobserved scales of parallelism) also warrants investigation. Finally, the impetus for tensor completion driven by recommendation systems and image recognition ensures that modeling frameworks that adopt this technique will benefit from its dissemination among software libraries and continued optimization.

Acknowledgment

The first author would like to acknowledge the Department of Energy (DOE) and Krell Institute for support via the DOE Computational Science Graduate Fellowship (grant No. DE-SC0019323). This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562. Via XSEDE, the authors made use of the TACC Stampede2 supercomputer (allocation TG-CCR180006). This research has

also been supported by funding from the National Science Foundation (grant No. 2028861).

REFERENCES

[1] R. Vuduc, J. W. Demmel, and J. A. Bilmes, "Statistical models for empirical search-based performance tuning," *The International Journal of High Performance Computing Applications*, vol. 18, no. 1, pp. 65–94, 2004.

[2] R. B. Roy, T. Patel, V. Gadepally, and D. Tiwari, "Bliss: auto-tuning complex applications using a pool of diverse lightweight learning models," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021, pp. 1280–1295.

[3] F. Petrini, D. J. Kerbyson, and S. Pakin, "The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of asci q," in *SC'03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. IEEE, 2003, pp. 55–55.

[4] T. Hoefler, W. Gropp, W. Kramer, and M. Snir, "Performance modeling for systematic performance tuning," in *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2011, pp. 1–12.

[5] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, "Using automated performance modeling to find scalability bugs in complex codes," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.

[6] Y. Liu, W. M. Sid-Lakhdar, O. Marques, X. Zhu, C. Meng, J. W. Demmel, and X. S. Li, "Gptune: multitask learning for autotuning exascale applications," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 234–246.

[7] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," *ACM SIGOPS operating systems review*, vol. 40, no. 5, pp. 185–194, 2006.

[8] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee, "Methods of inference and learning for performance modeling of parallel applications," in *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2007, pp. 249–258.

[9] A. Calotoiu, D. Beckinsale, C. W. Earl, T. Hoefler, I. Karlin, M. Schulz, and F. Wolf, "Fast multi-parameter performance modeling," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2016, pp. 172–181.

[10] N. R. Tallent and A. Hoisie, "Palm: Easing the burden of analytical performance modeling," in *Proceedings of the 28th ACM international conference on Supercomputing*, 2014, pp. 221–230.

[11] M. Copik, A. Calotoiu, T. Grosser, N. Wicki, F. Wolf, and T. Hoefler, "Extracting clean performance models from tainted programs," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 403–417.

[12] J. Choi, D. F. Richards, L. V. Kale, and A. Bhatele, "End-to-end performance modeling of distributed gpu applications," in *Proceedings of the 34th ACM International Conference on Supercomputing*, 2020, pp. 1–12.

[13] J. H. Friedman, "Multivariate adaptive regression splines," *The annals of statistics*, pp. 1–67, 1991.

[14] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[15] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 208–220, 2012.

[16] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," *Journal of Mathematics and Physics*, vol. 6, no. 1-4, pp. 164–189, 1927.

[17] D. Hong, T. G. Kolda, and J. A. Duersch, "Generalized canonical polyadic tensor decomposition," *SIAM Review*, vol. 62, no. 1, pp. 133–163, 2020.

[18] N. Singh, Z. Zhang, X. Wu, N. Zhang, S. Zhang, and E. Solomonik, "Distributed-memory tensor completion for generalized loss functions in python using new sparse tensor kernels," *ArXiv e-prints*, pp. arXiv–1910, 2019.

[19] L. Karlsson, D. Kressner, and A. Uschmajew, "Parallel algorithms for tensor completion in the cp format," *Parallel Computing*, vol. 57, pp. 222–234, 2016.

[20] S. Smith, J. Park, and G. Karypis, "An exploration of optimization algorithms for high performance tensor completion," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 359–371.

[21] D. Pflüger, *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. München: Verlag Dr. Hut, Aug. 2010. [Online]. Available: http://www5.in.tum.de/pub/pflueger10spatially.pdf

[22] P. Neumann, "Sparse grid regression for performance prediction using high-dimensional run time data," in *European Conference on Parallel Processing*. Springer, 2019, pp. 601–612.

[23] C. Tofallis, "A better measure of relative prediction accuracy for model selection and model estimation," *Journal of the Operational Research Society*, vol. 66, no. 8, pp. 1352–1362, 2015.

[24] S. K. Morley, T. V. Brito, and D. T. Welling, "Measures of model performance based on the log accuracy ratio," *Space Weather*, vol. 16, no. 1, pp. 69–88, 2018.

[25] T. Rauber and G. Runger, "Modelling the runtime of scientific programs on parallel computers," in *Proceedings 2000. International Workshop on Parallel Processing*. IEEE, 2000, pp. 307–314.

[26] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive performance and scalability modeling of a large-scale application," in *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, 2001, pp. 37–37.

[27] T. Hoefler, W. Gropp, R. Thakur, and J. L. Träff, "Toward performance models of mpi implementations for understanding application scaling issues," in *European MPI Users' Group Meeting*. Springer, 2010, pp. 21–30.

[28] R. D. Friese, N. R. Tallent, A. Vishnu, D. J. Kerbyson, and A. Hoisie, "Generating performance models for irregular applications," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 317–326.

[29] T. M. Low, F. D. Igual, T. M. Smith, and E. S. Quintana-Orti, "Analytical modeling is enough for high-performance blis," *ACM Trans. Math. Softw.*, vol. 43, no. 2, aug 2016. [Online]. Available: https://doi.org/10.1145/2925987

[30] R. Li, A. Sukumaran-Rajam, R. Veras, T. M. Low, F. Rastello, A. Rountev, and P. Sadayappan, "Analytical cache modeling and tilesize optimization for tensor contractions," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–13.

[31] R. Li, Y. Xu, A. Sukumaran-Rajam, A. Rountev, and P. Sadayappan, "Analytical characterization and design space exploration for optimization of cnns," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 928–942. [Online]. Available: https://doi.org/10.1145/3445814.3446759

[32] E. Nuriyev and A. Lastovetsky, "Efficient and accurate selection of optimal collective communication algorithms using analytical performance modeling," *IEEE Access*, vol. 9, pp. 109 355–109 373, 2021.

[33] E. Georganas, J. González-Domínguez, E. Solomonik, Y. Zheng, J. Tourino, and K. Yelick, "Communication avoiding and overlapping for numerical linear algebra," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11.

[34] P. Malakar, P. Balaprakash, V. Vishwanath, V. Morozov, and K. Kumaran, "Benchmarking machine learning methods for performance modeling of scientific applications," in *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 2018, pp. 33–44.

[35] D. Pflüger, "Spatially adaptive refinement," in *Sparse Grids and Applications*, ser. Lecture Notes in Computational Science and Engineering, J. Garcke and M. Griebel, Eds. Berlin Heidelberg: Springer, Oct. 2012, pp. 243–262. [Online]. Available: http://www5.in.tum.de/pub/pflueger12spatially_preprint.pdf

[36] M. Courtois and M. Woodside, "Using regression splines for software performance analysis," in *Proceedings of the 2nd international workshop on Software and performance*, 2000, pp. 105–114.

[37] H. Yserentant, "Hierarchical bases," in *Proceedings of the second international conference on Industrial and applied mathematics*, 1992, pp. 256–276.

[38] S. A. Smolyak, "Quadrature and interpolation formulas for tensor products of certain classes of functions," in *Doklady Akademii Nauk*, vol. 148, no. 5. Russian Academy of Sciences, 1963, pp. 1042–1045.

[39] J. Garcke, "Sparse grids in a nutshell," in *Sparse grids and applications*. Springer, 2012, pp. 57–80.
[40] E. İpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz, "Efficiently exploring architectural design spaces via predictive modeling," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 5, pp. 195–206, 2006.
[41] A. Marathe, R. Anirudh, N. Jain, A. Bhatele, J. Thiagarajan, B. Kailkhura, J.-S. Yeom, B. Rountree, and T. Gamblin, "Performance modeling under resource constraints using deep transfer learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.
[42] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 69–77.
[43] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *Advances in neural information processing systems*, vol. 24, 2011.
[44] P. Jain, P. Netrapalli, and S. Sanghavi, "Low-rank matrix completion using alternating minimization," in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, 2013, pp. 665–674.
[45] R. Keshavan, A. Montanari, and S. Oh, "Matrix completion from noisy entries," *Advances in neural information processing systems*, vol. 22, 2009.
[46] H.-F. Yu, C.-J. Hsieh, S. Si, and I. Dhillon, "Scalable coordinate descent approaches to parallel matrix factorization for recommender systems," in *2012 IEEE 12th international conference on data mining*. IEEE, 2012, pp. 765–774.
[47] T. Hastie, R. Mazumder, J. D. Lee, and R. Zadeh, "Matrix completion and low-rank svd via fast alternating least squares," *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 3367–3402, 2015.
[48] C. Teflioudi, F. Makari, and R. Gemulla, "Distributed matrix completion," in *2012 ieee 12th international conference on data mining*. IEEE, 2012, pp. 655–664.
[49] R. Yokota, "An fmm based on dual tree traversal for many-core architectures," *Journal of Algorithms & Computational Technology*, vol. 7, no. 3, pp. 301–324, 2013.
[50] M. Gates, J. Kurzak, A. Charara, A. YarKhan, and J. Dongarra, "Slate: design of a modern distributed and accelerated linear algebra library," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–18.
[51] T. Deakin, S. McIntosh-Smith, and W. Gaudin, "Many-core acceleration of a discrete ordinates transport mini-app at extreme scale," in *International Conference on High Performance Computing*. Springer, 2016, pp. 429–448.
[52] A. J. Kunen, T. S. Bailey, and P. N. Brown, "Kripke-a massively parallel transport mini-app," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2015.
[53] D. Stanzione, B. Barth, N. Gaffney, K. Gaither, C. Hempel, T. Minyard, S. Mehringer, E. Wernert, H. Tufo, D. Panda, and P. Teller, "Stampede 2: The evolution of an XSEDE supercomputer," in *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, ser. PEARC17. New York, NY, USA: ACM, 2017, pp. 15:1–15:8. [Online]. Available: http://doi.acm.org/10.1145/3093338.3093385
[54] E. Solomonik, D. Matthews, J. R. Hammond, J. F. Stanton, and J. Demmel, "A massively parallel tensor contraction framework for coupled-cluster computations," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3176–3190, 2014.
[55] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

APPENDIX