Energy Efficiency Modeling of Parallel Applications

Mark Endrei*, Chao Jin*, Minh Ngoc Dinh*, David Abramson*,
Heidi Poxon†, Luiz DeRose† and Bronis R. de Supinski‡
*Research Computing Center and School of ITEE, The University of Queensland, QLD, Australia
†Cray Inc., Bloomington, MN, USA
‡Lawrence Livermore National Laboratory, Livermore, CA, USA
Email: *{mark.endrei, c.jin, m.dinh1, david.abramson}@uq.edu.au, †{heidi, ldr}@cray.com, ‡bronis@llnl.gov

***Abstract*—Energy efficiency has become increasingly important in high performance computing (HPC), as power constraints and costs escalate. Workload and system characteristics form a complex optimization search space in which optimal settings for energy efficiency and performance often diverge. Thus, we must identify trade-off options for performance and energy efficiency to find the desired balance between them. We present an innovative statistical model that accurately predicts the Pareto optimal performance and energy efficiency trade-off options using only user-controllable parameters. Our approach can also tolerate both measurement and model errors. We study model training and validation using several HPC kernels, then explore the feasibility of applying the model to more complex workloads, including AMG and LAMMPS. We can calibrate an accurate model from as few as 12 runs, with prediction error of less than 10%. Our results identify trade-off options allowing up to 40% improvement in energy efficiency at the cost of under 20% performance loss. For AMG, we reduce the required sample measurement time from 13 hours to 74 minutes (about 90%).**

I. Introduction

Presently power constraints limit the number of both CPU cores and compute nodes in a supercomputer [1]. We must develop innovative methods to predict the relationship between performance and energy usage for parallel computing. Understanding the available trade-off options between performance and energy efficiency is critical at both application development [1] and deployment [2]. System resource management must maximize performance subject to a given power budget by configuring an application correctly at runtime [2], [3]. In addition, programmers need practical tools that efficiently identify and select trade-off options to optimize energy efficiency of their parallel applications.

Unfortunately, many factors influence the trade-off. These factors include application characteristics, like computational intensity, and memory and communication access patterns, and system factors, like cache design, and memory and network bandwidth. Thus, tools to optimize this multi-objective problem [4] must search a large and complicated space.

Pareto frontiers typically represent the trade-off options between energy use and performance [4]. Thus, a model that can predict optimal configurations along the Pareto front can estimate the trade-off options. We propose a regression modeling technique that predicts Pareto-optimal energy efficiency and performance trade-off options but does not require complete search space explorations.

Our technique has several advantages over existing model-based methods [5], [6], [7], [8] to improve energy efficiency for parallel computing. First, it only needs a minimal set of input variables that supercomputer users can simply access and control by themselves. In contrast, most existing models [5], [6] build on several runtime measurements that directly reflect energy efficiency, such as computational intensity, cache miss rate, stall cycles, the number of memory accesses, and the amount of network communications. While these models provide valuable insight, such parameters are very difficult for application users to control. The process of acquiring these measurements also increases the difficulty of automating these models as practical tools. Second, our model considers both sample measurement error and model prediction error in predicted trade-off options. Third, most existing models do not directly expose Pareto-efficient configurations [5], [6], [7], [8].

We demonstrate that our model accurately predicts, at low cost, the Pareto front of optimal configurations for hybrid MPI/OpenMP programs. We present a study that assesses the fit and variance of our models for representative parallel kernels and applications. From the study, we find that our methodology and models can successfully identify trade-off options that are consistent with measured data for a broad set of the optimization search space. Further, building our model only needs a small set of sample measurements, which minimizes the effort and resources required for training. Overall, our model is an ideal candidate for a practical, automated tool.

Specifically, this paper presents the following contributions:

- Energy and performance trade-offs using parameters that parallel application users can directly control;
- A practical method to predict energy efficiency and performance trade-off options from few input measurements;
- Multi-objective energy efficiency and performance models that accurately predict parallel application responses;
- A trade-off zone approach that improves Pareto optimization, given measurement and/or modeling error.

The rest of this paper is organized as follows. Section II provides an overview of related work and our motivation. Section III introduces our modeling and error mitigation methodology. Sections IV, V and VI describe the evaluation of our methodology using a total of 2,420 experimental runs on an HPC cluster. Section VII provides further validation of model fit and variance. Our conclusions follow in Section VIII.

SC18, November 11-16, 2018, Dallas, Texas, USA
978-1-5386-8384-2/18/$31.00

II. Related Work

A wide-range of prior work [1] explores energy-efficient computing. We briefly discuss some state-of-the-art techniques to improve energy efficiency of HPC systems and their relationship to our work. Most existing work that optimizes power usage for parallel computing consists of three groups.

The first group aims to lower energy consumption without decreasing performance [9], [10], [11], [12]. Typically, these "best effort" methods identify the opportunity of load imbalance and processor underutilization at runtime to lower power consumption without significantly hurting performance. Often, profiling or tracing identifies finer regions with different degrees of idleness in a parallel program. According to each region's pattern, such as phases with memory stalls and intensive computation, techniques use Dynamic Voltage and Frequency Scaling (DVFS) to select a CPU frequency or Dynamic Concurrency Throttling (DCT) [13] to adjust the number of active cores to decrease the power wasted by idleness. The power usage for the regions of MPI communications is also optimized specifically according to communication patterns [14], [15], [16]. Other techniques detect MPI processes that are not on the critical path and slow them to save energy [17], [18].

The second group allows users to make trade-offs between power and performance, such as power-constrained performance optimization and auto-tuning. Auto-tuning [19], [20], [21], [22], [23], [24] methods typically search for the right parameter settings and code transformations to deliver optimal performance and power usage on a target platform. The state-of-the-art auto-tuning tools apply various search algorithms, such as machine learning algorithms and heuristic algorithms, to prune the search space that consists of environment settings, compiler flags and application-specific parameters. Auto-tuning methods use either a combined single objective [4], such as $energy \times delay$ ($E \times D$) and $energy \times delay \times delay$ ($E \times D^2$), or multi-objective optimization to explore the optimal system configurations to balance performance and energy use [21]. Multi-objective optimization provides a Pareto set of optimal performance and power configurations. The Pareto-efficient configurations can guide users to achieve energy-optimal computing [25] and can provide a power and performance model that assists resource management, such as over-provisioned job scheduling [2]. Power-constrained performance optimization [3] also relies on the Pareto-efficient configurations to detect the optimized performance under a power budget.

The third group is model-based methods [5], [6], [7], [8], [26], [27], [28]. Most power models extend performance models, such as Amdahl's Law, iso-efficiency, and the Roofline model. The energy efficiency extension of Amdahl's Law [27] and the energy roofline model [26] provide a high-level view of the relationship between energy use and performance while varying parallelism and operational intensity.

Most other models [5], [6], [7] formulate the power and performance relationship using "white-box" methods that rely on runtime measurements, such as CPU cycles, stall cycles, cache and memory bandwidth, and program structures, such as loop nesting. For example, the iso-energy-efficiency model [6] extends the traditional performance iso-efficiency function to analyze the combined effect of performance and power while adjusting system parameters. Iso-energy-efficiency is defined as the ratio between energy consumed by sequential and parallel executions. About 30 system and application dependent parameters are measured on a small scale to build the model. The model can then be used to predict the performance and power behavior on a large-scale system with different system parameters, such as CPU frequency and node count. The accuracy of these "white-box" models is highly dependent on runtime measurements that are collected using traces or by instrumentation to query performance counters, which is often not a trivial process. Typically, performance and energy counter events should be collected in separate runs to avoid counter multiplexing. Otherwise, it may cause application perturbation and decrease measurement accuracy [29]. The process of building these models requires user intervention to collect runtime traces or to analyze performance counters, which complicates automation of these models. Importantly, these existing "white-box" methods do not expose the Pareto frontier of energy efficient configurations so users cannot directly apply these models to find effective "sweet spots" that balance performance and energy consumption.

Unlike prior work, our "black-box" model predicts the Pareto-efficient configurations on the targeted platform. In comparison to previous work, our approach only relies on a minimum set of parameters that are fully controlled by users, such as the number of threads, CPU frequency and the number of compute nodes. Thus, our method is easy to automate because it does not need any external instrumentation tools, and its application across different platforms is less restricted. Most importantly, we demonstrate that our "black-box" method can also provide high accuracy in terms of energy and performance prediction. Previous performance prediction methods [30], [31] have adopted regression-based models to estimate the performance for various system configurations. However, to the best of our knowledge, we present the first method to use regression-based methods to explore Pareto-efficient configurations that allow users to make trade-offs between performance and energy use.

III. Methodology

This section describes our methodology to create and to validate our models of energy efficiency and performance.

A. Model Predictors

The initial step selects model input parameters, or *predictors*, that can accurately model our *responses*: system energy efficiency and performance. Prior work [29] showed that a complex relationship exists between system characteristics, workload features, concurrency, CPU frequency scaling and the observed system response. We use the definitions in Table I to express these relationships as shown in Equations 1 and 2.

TABLE I
PARAMETERS USED IN THE MODELS

Parameter	Description
w_n	Workload or application to be evaluated
s_m	System or architecture to be evaluated
$E(w_n, s_m), e_\eta$	Energy efficiency of workload n on system m
$P(w_n, s_m), p_\mu$	Performance of workload n on system m
n_i	Count of homogeneous nodes to be evaluated
c_j	Count of cores or threads to be evaluated
f_k	CPU frequency or DVFS setting to be evaluated
$bs(x)$	Transform to generate B-spline bases for parameter x
$F_e(n_i, c_j, f_k)$	Energy efficiency function of nodes, cores, frequency
$F_p(n_i, c_j, f_k)$	Performance function of nodes, cores, frequency
e_T	Total energy required to execute workload
t_T	Total time required to execute workload

$$E(w_n, s_m) = F_e(n_i, c_j, f_k) \tag{1}$$

$$P(w_n, s_m) = F_p(n_i, c_j, f_k) \tag{2}$$

B. Model Formulation

To formulate the multiple regression models we first analyze the responses of a set of representative software kernels, using node count, core count, and CPU frequency setting as the model inputs. The observed curvilinear performance and energy efficiency responses mean the models require polynomial terms.

The inflexibility of polynomials [32] can dramatically impact basic polynomial fit. It may result in undesirable oscillations in the predicted response, or an overall response shape that is unduly affected by a small subset of samples. To minimize the impact of this inflexibility, we begin with a B-spline piecewise polynomial model using ordinary least squares regression. We find that splines consistently outperform basic polynomials in our experiments.

Spline functions are used to fit a smooth curve along a series of points, or knots. We construct the curve that joins each pair of points, or *spline*, piece-wise from polynomial functions. A B-spline, or basis spline function, improves the continuity at the knots, which may otherwise affect model continuity.

We have several configuration options to tune the accuracy and efficiency of the models. We consider options that include polynomial degree and degrees of freedom for splines. Increasing these model parameters allows the spline to fit more complex curves. We also consider interactions between terms, linear and non-linear terms, transforms to reduce effects of data distribution skew, and sampling method.

To guide model tuning, we observe the level of variability that each predictor drives in the response and we use knowledge of expected interactions between predictors. For example, increasing core counts drive resource contention that produces non-linear responses. Increasing CPU frequencies typically drive linear responses until core contention within the node starts to dominate. Such observations lead us to the following model settings and simplifications:

- B-spline degrees of freedom is three;
- Node and core count are quadratic polynomial terms;
- CPU frequency is a linear term;
- Node and core count have an interaction term;
- Core count and frequency have an interaction term;
- Frequency and node count have no interaction term;
- Natural logarithm transform of response.

To evaluate model settings experimentally, we use an iterative approach that assesses the data distribution and correlation, the model fit, and the coefficient magnitudes. For example, we experimentally compare model fit with CPU frequency as a polynomial or linear term and find the linear term option provides a slight improvement in average RMS error across our study test cases (2.3%). We also experimentally determine the optimal polynomial term order and B-spline degrees of freedom at the transition point between model underfit and overfit. For example, the near zero coefficient for the core count and frequency interaction term confirms that we can remove it from the model with little impact.

As a result, our statistical models for energy efficiency and performance of a parallel application that runs on a homogeneous HPC cluster use Equations 3 and 4.

$$\begin{aligned} log_e(e_\eta) \sim\ & bs(n_i) + bs(c_j) + f_k + bs(c_j) : f_k \\ & + bs(n_i) : bs(c_j) \end{aligned} \tag{3}$$

$$\begin{aligned} log_e(p_\mu) \sim\ & bs(n_i) + bs(c_j) + f_k + bs(c_j) : f_k \\ & + bs(n_i) : bs(c_j) \end{aligned} \tag{4}$$

These equations are based on Wilkinson notation [33], which is accepted by regression analysis tools such as MATLAB, or R and Python programming libraries. The operators used are '$\sim$' meaning *is modeled by*, '$+$' to add a term to the model, and ':' to include interactions between terms.

In addition to energy efficiency and performance rates, the models can support cumulative responses such as total energy and total time based on Equations 5 and 6.

$$\begin{aligned} log_e(e_T) \sim\ & bs(n_i) + bs(c_j) + f_k + bs(c_j) : f_k \\ & + bs(n_i) : bs(c_j) \end{aligned} \tag{5}$$

$$\begin{aligned} log_e(t_T) \sim\ & bs(n_i) + bs(c_j) + f_k + bs(c_j) : f_k \\ & + bs(n_i) : bs(c_j) \end{aligned} \tag{6}$$

We apply the natural exponential to model predictions to reverse the logarithm transformation.

Equation 7 shows the right-hand side expansion for Equations 3 to 6. Each model has 20 predictor terms (1 intercept, 1 linear term, 2 $\times$ 3 spline B-spline terms, 1 $\times$ 3 spline B-spline $\times$ 3 spline B-spline interaction term, 1 $\times$ 3 spline B-spline $\times$ 1 linear term). We collect predictor and response training data then use least squares regression to determine the term coefficients, β_0 to β_{19}.

$$
\begin{aligned}
&\beta_0 + \beta_1 \cdot f_k \\
&+ \beta_2 \cdot bs_1(n_i) + \beta_3 \cdot bs_2(n_i) + \beta_4 \cdot bs_3(n_i) \\
&+ \beta_5 \cdot bs_1(c_j) + \beta_6 \cdot bs_2(c_j) + \beta_7 \cdot bs_3(c_j) \\
&+ \beta_8 \cdot bs_1(n_i) \cdot bs_1(c_j) + \beta_9 \cdot bs_2(n_i) \cdot bs_1(c_j) \\
&+ \beta_{10} \cdot bs_3(n_i) \cdot bs_1(c_j) + \beta_{11} \cdot bs_1(n_i) \cdot bs_2(c_j) \\
&+ \beta_{12} \cdot bs_2(n_i) \cdot bs_2(c_j) + \beta_{13} \cdot bs_3(n_i) \cdot bs_2(c_j) \\
&+ \beta_{14} \cdot bs_1(n_i) \cdot bs_3(c_j) + \beta_{15} \cdot bs_2(n_i) \cdot bs_3(c_j) \\
&+ \beta_{16} \cdot bs_3(n_i) \cdot bs_3(c_j) \\
&+ \beta_{17} \cdot bs_1(c_j) \cdot f_k + \beta_{18} \cdot bs_2(c_j) \cdot f_k \\
&+ \beta_{19} \cdot bs_3(c_j) \cdot f_k
\end{aligned}
\tag{7}
$$

C. Model Evaluation

To evaluate the model, we use correlation analysis, k-fold cross validation, and RMS error and R^2 statistics. We derive these quantities from many measurement samples in order to provide a statistically significant validation of model accuracy. Our results in Sections V, VI and VII demonstrate that a small sample of measured data is sufficient to fit system- and workload-specific coefficients for the models.

Spearman's rank correlation coefficient, ρ, is a non-parametric measure of the correlation between predictors and responses. Values near zero indicate weak correlation. We use this statistic to validate hypotheses when we select predictors.

The R^2 statistic measures how well a model fits training data. R^2 values near 100% indicate better fit. RMS error is the standard deviation of the prediction error, or regression residuals. It measures how closely observed data fits data forecasts that the model generates. RMS error values near zero indicate a close fit between model forecasts and observed data.

The k-fold cross validation method partitions the data set into k equal-size subsets. $k-1$ subsets serve as model training data and the remaining subset is the model test data. We repeat the process k times to test model accuracy across the full data set. We use 3-fold cross validation to confirm that the model does not overfit a subset of the data.

D. Pareto Front Evaluation

We calculate RMS error between our observed and predicted Pareto fronts, but also use further, specific techniques to assess the accuracy of the predicted fronts. We assess Pareto front accuracy using the following metrics:

- Overlapping point count, i.e., the count of points that occur in both observed and predicted fronts;
- Non-overlapping point count on observed front, grouped by distance to nearest neighbor on predicted front;
- Non-overlapping point count on predicted front, grouped by distance to nearest neighbor on observed front;
- Predicted and observed minimums and maximums for each objective (energy efficiency or performance);
- Predicted and observed trade-off ranges for each objective compared to their values when threads and CPU frequency are at the maximum settings.

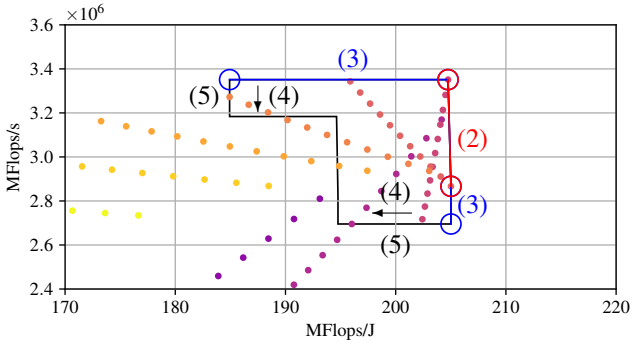

Fig. 1. Pareto Trade-off Zone Construction Steps

We measure the distance to the nearest neighbor as a percentage of the search space dimension. For example, our thread count per node dimension has 11 increments from 4 to 44 in steps of 4. The distance from 36 to 40 would be $1/11 = 9\%$, and 36 to 44 would be $2/11 = 18\%$. This choice normalizes across dimensions.

E. Measurement and Modeling Error

Performance and power measurements exhibit experimental error and noise due to the non-deterministic nature of systems. These measurement errors, which are often normally distributed, result from a range of random factors. Normality tests can validate that our power and performance measurements are normally distributed. We use the Q-Q (quantile-quantile) plot to compare our measured data distribution visually with the standard normal distribution. We use the t-distribution to analyze measurement confidence intervals. The log_e transform that our models use mitigates normal distribution deviations.

Model forecasts exhibit prediction error. Thus, we must allow for both measurement and prediction errors when we identify trade-off options between performance and energy efficiency. Valid trade-off options within error limits lie in a zone close to the Pareto front, rather than only lying directly on it. We introduce the concept of a *trade-off zone* that includes all values near the Pareto front that are not statistically distinguishable from those on the front.

Figure 1 shows our steps to construct the trade-off zone:

1) Plot the data set against the trade-off parameters (energy efficiency and performance in our case);
2) Plot the Pareto front along the Pareto-optimal points;
3) Extend the Pareto front outer limits horizontally and vertically to encompass all points that are off the front but are within the error limits for the respective axes (we also use this curve to calculate the RMS error of the predicted Pareto front compared to the observed one, after interpolating each curve to the same line-space);
4) Scale and translate the Pareto front by the axes error limits to set the inner limits of the trade-off zone;
5) Close the two curves, which creates a polygon that represents the trade-off zone.

The trade-off zone polygon encloses the set of points that may be Pareto optimal when we compensate for the error

TABLE II
SYSTEM SPECIFICATION

Component	Specification
CPU model	Intel Xeon CPU E5-2699 v4 (Broadwell)
CPU clock	2.2 GHz
Sockets (NUMA Nodes)	2 per compute node
Cores	22 per socket
Last Level Cache (LLC)	55 MB per socket
Main memory (DRAM)	64 GB per socket
Memory bandwidth	76.8 GB/s max

limits of each axis. These points provide the trade-off options for energy efficiency and performance. Since measured and predicted error limits may differ, we set error limits individually for the measured and predicted Pareto fronts.

Our models only use measured data for the response variable, which provides two important benefits. First, our fitted model coefficients are not biased by random error in the response measurements [34]. This means our regression estimates tend to average values of the training data, plus or minus measurement error. Second, our model can be used to make predictions for unseen predictor data, so we can practically explore a large parameter space with a small number of training measurements.

We can control our measurement error levels to around 5% by ensuring that overall execution time is large compared to program initialization and shutdown time and that execution time is large compared to the temporal resolution of system power counters. In less controlled environments with larger measurement error, the size of the trade-off zone will increase as more data points fall within the error limits. A full experimentally measured sweep of the search space will be similarly impacted by increased measurement error. We use estimated measurement error and model error to assess the level of alignment or overlap between the observed and predicted trade-off zones.

IV. STUDY OVERVIEW

This section provides a summary of the platform, parameters and phases of our experimental study.

A. Platform

We conduct experiments on a Cray XC system equipped as Table II shows. Our runs use up to 86, exclusively allocated 44-core nodes, or 3,784 cores in total. We use the Python programming platform [35] for correlation analysis, fitting model coefficients, evaluating the responses, and plotting the results [36], [37], [38], [39]. The Nimrod toolkit [40] is used to orchestrate experiment tasks.

B. Power Measurement

The Cray XC system has node-level sensors to measure temperature, current, and voltage. Cray power management counters (pm_counters) provide real-time power and energy measurements, which are updated at a frequency of 10 Hz. CrayPAT [41] is a performance analysis tool that uses performance counters, including pm_counters and hardware performance counters (HWPC), to evaluate program behavior. It instruments the program, collects the specified counters at runtime, and reports the collected counters. Cray pm_counters can also be read directly from the Linux sysfs folder, /sys/cray/pm_counters, as a program launches and terminates. We monitor energy and power consumption for all nodes with pm_counters. To assess energy efficiency, we use operations per Joule (for example, Flops/J, Bytes/J, Updates/J).

TABLE III
EXPERIMENTAL PARAMETERS

Parameter	Configuration
Compute Nodes	20, 42, 64 and 86
MPI Ranks Per Node	2
OpenMP Threads per Rank	2 to 22 incrementing by 2
OpenMP Threads per Core	1
CPU Frequency	1.2 to 2.2 GHz incrementing by 0.1
Total CPU Cores	3,784

C. Study Parameters

Table III lists configuration parameters and associated ranges that we use in our study. These parameters generate a full factorial design with 484 combinations.

D. Experiments

We conduct our study experiments in two main phases:

1) Model design and evaluation using kernels;
2) Model evaluation using applications.

The kernels and applications all use the hybrid MPI/ OpenMP programming model. We configure experiment jobs to allocate one MPI Rank per CPU socket and one OpenMP thread per CPU core. We distribute OpenMP threads uniformly across the available sockets and nodes using the Scatter thread placement policy. For each experiment we collect the full 484 sample combinations as listed in Table III, making 2,420 tests in total for three kernels and two applications.

V. KERNELS STUDY

This section demonstrates that we can accurately predict Pareto-optimal energy and performance trade-off options with low cost for several scientific kernels that focus on specific computational idioms. We choose the Parallel Research Kernels (PRK) [42], a collection of programs that cover common patterns of communication, computation, and synchronization encountered in parallel HPC applications. The PRK come with performance metric reporting, which allows us to focus on power consumption attributes to identify energy optimization opportunities. Out of the available hybrid MPI/OpenMP kernels in PRK, we focus on the following three kernels:

1) **Stencil:** a kernel that performs data-parallel stencil operation to a two-dimensional array;
2) **Transpose:** a kernel that stresses communication and memory bandwidth;

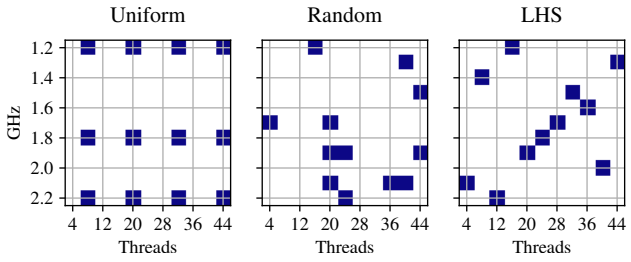

Fig. 2. Uniform, Random, and Latin Hypercube Sampling

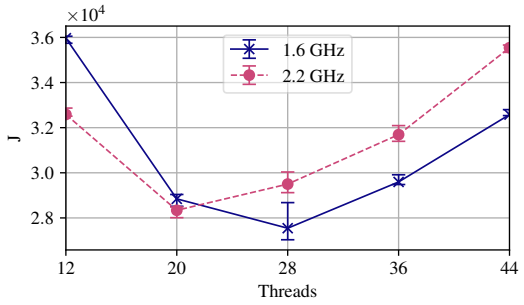

Fig. 3. Stencil Experimental Measurement Error

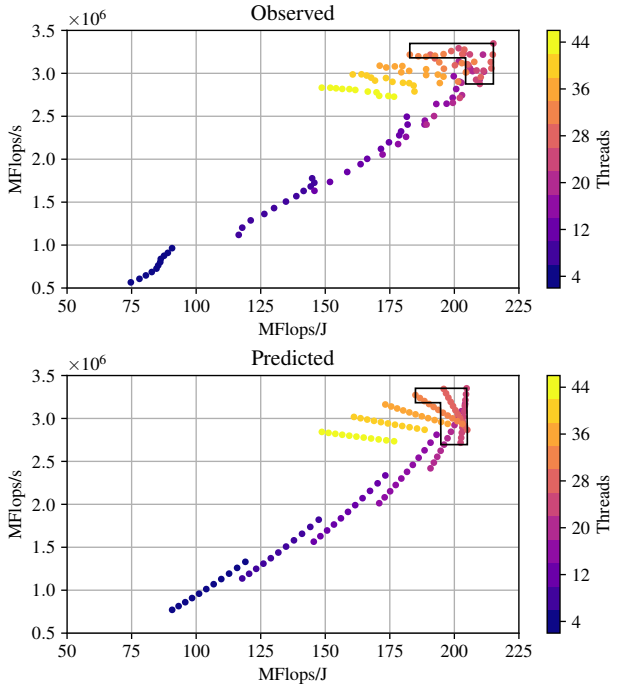

Fig. 4. Stencil Pareto Front – Observed and Predicted

3) **Nstream:** an embarrassingly parallel kernel that computes memory bandwidth.

A. Stencil Kernel

The model response terms for stencil are energy efficiency in MFlops/J and performance in MFlops/s. We use stencil radius of 2, grid size of 400k, and set iterations to ensure that run time is at least 10 times the measurement sample rate.

We evaluate several methods to generate model training data including uniform sampling, random sampling, and latin hypercube sampling [43]. Figure 2 shows our 11 core count $\times$ 11 frequency search space with uniform or non-random samples, random samples, and latin hypercube samples which have a randomly selected sample from each row and column.

We evaluate each sampling method using RMS error and R^2 statistics, as described in Section III-C. We achieve the best fit with the fewest observations across our test cases using uniform sampling. Our method requires 12 samples or 10% of the search space for model training (4 core count $\times$ 3 frequency samples). Our core count and frequency samples from Figure 2 are 8, 20, 32, and 44 cores, and 1.2, 1.8, and 2.2 GHz respectively. We also evaluate the model at node counts of 20, 42, 64 and 86, for a total data set size of 484 samples.

Figure 3 includes error bars for the 95% t-distribution confidence interval for the mean of five samples. The measurement error margins are about 5% for our experiments. We expect that setting up a tool using our model will include a calibration step that calculates measurement confidence intervals for each response variable. This strategy eliminates sample repetitions for each training sample to calculate their confidence intervals. Instead, we alert the user if the statistical significance of the regression results is not within set limits.

Figure 4 shows the experimentally observed and model predicted Pareto fronts for stencil on 64 nodes. Points off the front are not Pareto optimal as points on the front always provide an improvement in one parameter with less impact on the other. We scale the trade-off zone along the measured and predicted Pareto fronts for a 5% error margin.

The interaction between thread count and frequency determines the shape of the Pareto front. Figure 4 shows that data points are grouped by thread count, and rotate as we vary frequency. This rotation defines the shape of the Pareto front, which sets the energy versus performance trade-off ranges.

Table IV shows the RMS Error between the full 121 observed and predicted values is 4.8% for energy efficiency and 4.8% for performance. The RMS Error for the Pareto front is 5.1% for energy efficiency and 11.4% for performance, which we calculate as described in Step 3) in Section III-E. The Pareto Front section in Table IV shows similar observed and predicted Pareto point counts, with all non-overlapping points except one within 9% or one search step of an overlapping point (4 threads or 0.1 GHz).

Table IV also shows $Baseline$ performance and energy efficiency at maximum cores and threads, and their minima and maxima along the Pareto front, P_{min} and P_{max}.

The observed and predicted energy efficiency gains are similar (around 40%). The performance gain approaches 20%. The observed and model predicted Pareto fronts provide consistent views that core and frequency tuning can provide a 40% gain in energy efficiency with minimal impact on performance.

The surfaces in Figure 5 and 6 represent observed and predicted energy efficiency and performance across the CPU frequency and thread count search space. We plot Pareto points

TABLE IV
STENCIL RESULTS SUMMARY

Model Training Observations	Energy		Performance	
	R^2	RMS Error	R^2	RMS Error
12	0.993	4.8	0.997	4.8

Pareto Front	Points	0%	9%	18%	27%	36%
Observed (5% error)	31	29	2	0	0	0
Forecast (5% error)	39	29	9	1	0	0

Energy (Flops/J)	Baseline	P_{min}	P_{max}	% Range
Observed	149M	183M	215M	22.9 to 44.7
Forecast	149M	185M	205M	24.4 to 37.9
Percent error	0.0	1.2	-4.7	
Performance (Flops/s)	**Baseline**	**P_{min}**	**P_{max}**	**% Range**
Observed	2.83T	2.88T	3.35T	1.5 to 18.2
Forecast	2.84T	2.70T	3.35T	-5.2 to 17.9
Percent error	0.4	-6.3	0.1	

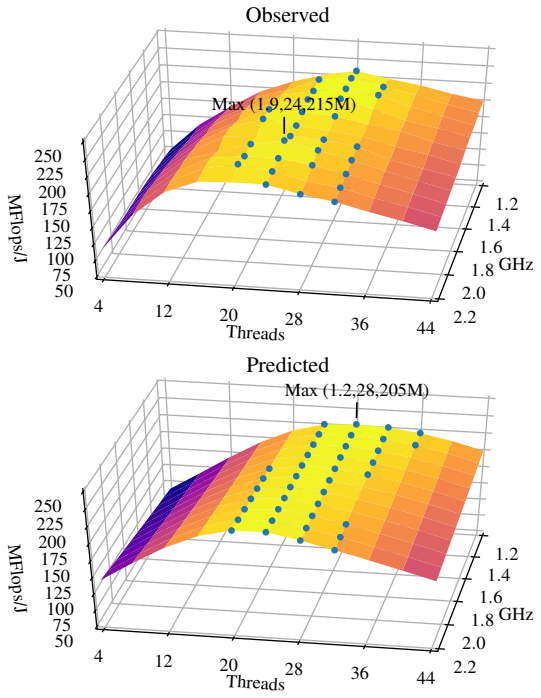

Fig. 5. Stencil Energy Efficiency – Observed and Predicted

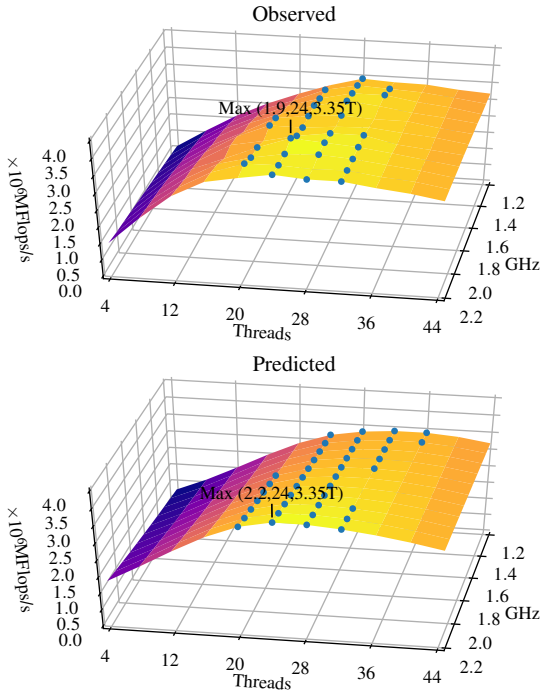

Fig. 6. Stencil Performance – Observed and Predicted

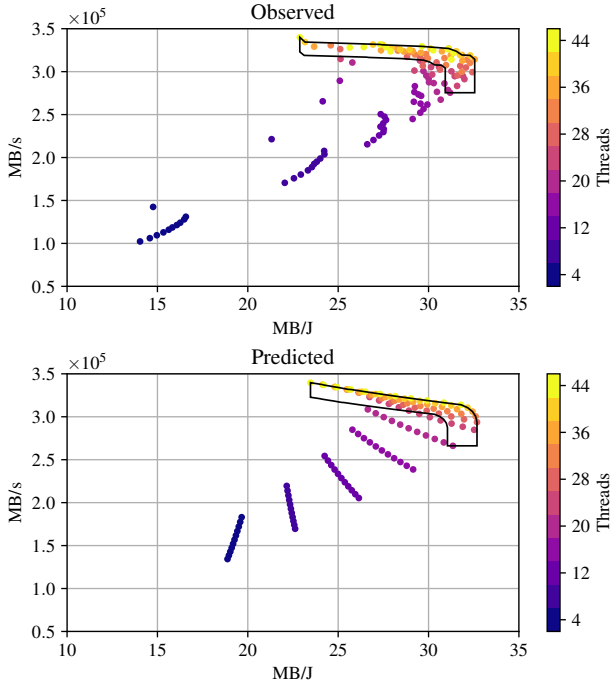

Fig. 7. Transpose Pareto Front – Observed and Predicted

on these surfaces to show their context in the search space.

Figure 6 shows that performance increases and then levels off as frequency and core count increase. The leveling-off marks the start of the trade-off zone, where energy efficiency starts to drop significantly, as Figure 5 shows, due to resource contention. This divergence between energy efficiency and performance means that tuning can increase energy efficiency significantly over a strategy that only minimizes run time.

B. Transpose Kernel

The transpose model response terms are energy efficiency in MB/J and performance in MB/s. We use a 200k transpose matrix order, disable blocking/tiling, and set iterations to ensure run time is at least 10 times the measurement sample rate. Figure 7 shows the experimentally observed and model predicted Pareto fronts for 64 nodes.

Table V shows the RMS Error between the full 121 observed and predicted values is 6.6% for energy efficiency and 4.5% for performance. The RMS Error for just the Pareto front is 4.4% for energy efficiency and 2.2% for performance. The

TABLE V
TRANSPOSE RESULTS SUMMARY

Model Training Observations	Energy		Performance	
	R^2	RMS Error	R^2	RMS Error
12	0.919	6.6	0.999	4.5

Pareto Front	Points	0%	9%	18%	27%	36%
Observed (5% error)	59	55	4	0	0	0
Forecast (5% error)	64	55	7	2	0	0

Energy (B/J)	Baseline	P_{min}	P_{max}	% Range
Observed	22.9M	22.9M	32.6M	0.0 to 42.3
Forecast	23.5M	23.5M	32.7M	0.0 to 39.3
Percent error	2.6	2.6	0.4	
Performance (B/s)	**Baseline**	P_{min}	P_{max}	**% Range**
Observed	340G	275G	340G	-19.0 to 0.0
Forecast	340G	266G	340G	-21.6 to 0.0
Percent error	-0.1	-3.4	-0.1	

Pareto Front section shows similar observed and predicted Pareto point counts, with most of the non-overlapping points within 9% or one search step of an overlapping point.

The observed and predicted energy efficiency gain are similar (around 40%). The gain requires a trade-off in performance of 20%. The observed and model predicted Pareto fronts agree that core and frequency tuning can increase energy efficiency by up to 40% in exchange for up to 20% performance loss.

C. Nstream Kernel

As with transpose, the nstream model responses are efficiency in MB/J and performance in MB/s. We use a vector length of 40G. We construct the experimentally observed and model predicted Pareto fronts for 64 nodes.

The RMS Error between observed and predicted values for nstream is 8.5% for energy efficiency and 7.5% for performance. The RMS Error for the Pareto front is 6.4% for energy efficiency and 2.3% for performance. The observed and predicted Pareto point counts are again similar, with all non-overlapping points within 9%.

The observed and model predicted Pareto fronts provide consistent views that core and frequency tuning can increase energy efficiency up to 40% with little performance impact.

VI. APPLICATIONS STUDY

This section demonstrates that we can accurately predict Pareto-optimal energy and performance trade-off options with low cost for more complex workloads. We apply our energy modeling method to AMG, a parallel algebraic multigrid solver for linear systems [44], and to LAMMPS, a classical molecular dynamics simulator [45].

A. AMG Application Study

AMG, parallelized using hybrid MPI/OpenMP, is well-known for its demands on main memory bandwidth. We study AMG because it contains microkernels that resemble the PRK kernels that we studied in Section V. First, AMG performs compressed sparse row (CSR) matrix vector multiplication.

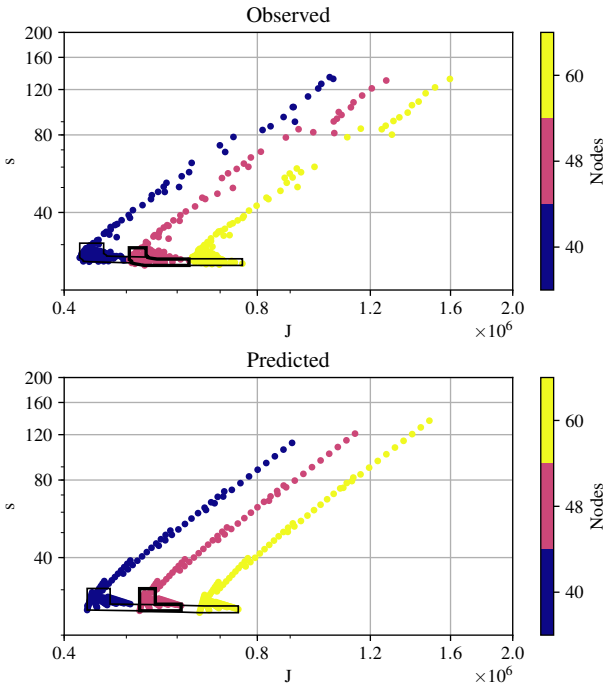

Fig. 8. AMG Pareto Front for 40, 48 and 60 Nodes – Observed and Predicted

Second, in optimizing the coarsening process, matrix *transpose* is essential. Finally, as the core of AMG, the algebraic multigrid mesh relaxation process uses a 27-point *stencil*.

The AMG model responses are cumulative functions rather than rate functions as with the PRK kernels, so we use model Equations 5 and 6. We measure AMG solve time in seconds as the performance metric, and adopt total energy in Joules as a cumulative metric for energy use. AMG also differs from the PRK kernels in that the problem size scales as the processor topology scales (i.e., weak scaling). These differences help further validate the generality of our method.

Our node count, MPI rank, and OpenMPI thread count setting also must fit within AMG processor topology restrictions. Thus, we use 20, 40, 60, and 80 nodes to retain the same rank and thread count settings that we used for the PRK. We use AMG problem size per processor of 256, which results in wall clock times of at least 30 seconds for each measurement.

1) Single Node Count Predictions: The model performs at a similar level for AMG on 60 nodes, as we saw with the kernels in Section V. In summary, the RMS error between the observed and predicted values is 6.6% for energy efficiency and 8.8% for performance. The RMS error for the Pareto front is 5.9% for energy usage and 7.1% for performance. The observed and predicted energy usage reductions are similar (around 15%) and incur about a 20% performance drop. The observed and model predicted Pareto fronts provide consistent views that core and frequency tuning can reduce energy usage up to 15% but at up to 20% performance loss.

2) Multiple Node Count Predictions: Our models can predict trade-off options for node counts for which we do not have training data. Figure 8 shows the experimentally observed and

TABLE VI
AMG RESULTS SUMMARY FOR 48 NODES

Model Training Observations	Energy		Performance	
	R^2	RMS Error	R^2	RMS Error
48	0.997	7.6	0.992	9.2

Pareto Front	Points	0%	9%	18%	27%	36%
Observed (5% error)	51	39	10	2	0	0
Forecast (5% error)	49	39	8	1	1	0

Energy (J)	Baseline	P_{min}	P_{max}	% Range
Observed	626k	626k	505k	0.0 to -19.3
Forecast	609k	609k	524k	0.0 to -13.8
Percent error	-2.8	-2.8	3.8	
Performance (s)	**Baseline**	P_{min}	P_{max}	% Range
Observed	26.0	29.2	24.9	12.1 to -4.3
Forecast	26.0	30.3	24.9	16.5 to -4.4
Percent error	-0.1	3.9	-0.2	

model predicted Pareto fronts for AMG for 48 nodes (thick outline), and across 40, 48 and 60 nodes (thin outline). The fronts appear at the lower left of the data set as our objective is now to minimize solve time and energy use.

The model training data now includes measurements across multiple node counts in the search space. Our previous examples use 12 samples at the required node count. We now use 12 samples at 20, 40, 60, and 80 nodes, or 48 samples in total. The extra samples significantly expand the search space coverage of the model. We can now make trade-off predictions at intervening node counts with similar accuracy.

We use sample time-stamps to determine that it takes 12 hours and 54 minutes to collect the required 484 AMG test samples, which corresponds to a mean time to collect each sample of 1 minute and 35 seconds. Using the same approach, the time to collect our 48 training samples is 1 hour and 14 minutes, or 9.6% of the time to collect the full 484 samples.

We can also speed up the process by collecting sample data in parallel. For example, our 86 node cluster allows us to allocate 20 and 40 node runs in parallel.

Table VI shows the RMS Error between the full 121 observed and predicted values is 7.6% for energy efficiency and 9.2% for performance. The RMS Error for just the Pareto front is 4.3% for energy efficiency and 7.5% for performance.

B. LAMMPS Application Study

This section presents energy and performance trade-off findings for LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator). LAMMPS is a full-scale scientific application that is parallelizable using hybrid MPI/OpenMP.

As with AMG, the LAMMPS model responses are energy use in Joules and performance in seconds. We use the Lennard-Jones LAMMPS Benckmark which simulates an atomic fluid with Lennard-Jones potential. We configure this benchmark for 32,000,000 atoms and 6,000 timesteps on 80 nodes.

Figure 9 shows LAMMPS's experimentally observed and model predicted Pareto fronts for 80 nodes. Table VII shows that the RMS error between the observed and predicted values

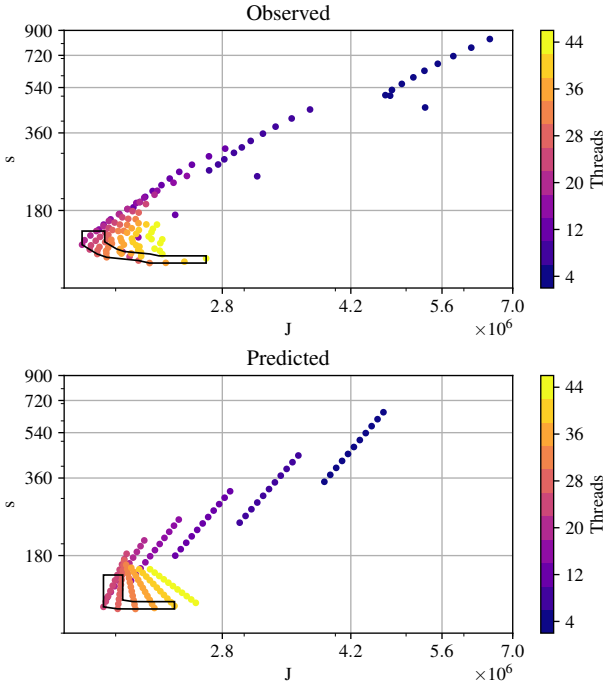

Fig. 9. LAMMPS Pareto Front – Observed and Predicted

TABLE VII
LAMMPS RESULTS SUMMARY

Model Training Observations	Energy		Performance	
	R^2	RMS Error	R^2	RMS Error
12	0.922	14.3	0.999	15.4

Pareto Front	Points	0%	9%	18%	27%	36%
Observed (5% error)	27	20	5	2	0	0
Forecast (5% error)	24	20	0	0	0	0

Energy (J)	Baseline	P_{min}	P_{max}	% Range
Observed	2.66M	2.66M	1.80M	0.0 to -32.4
Forecast	2.58M	2.41M	1.92M	-6.5 to -25.3
Percent error	-3.2	-9.5	7.0	
Performance (s)	**Baseline**	P_{min}	P_{max}	% Range
Observed	117	150	112	27.7 to -4.1
Forecast	118	151	112	28.2 to -5.3
Percent error	0.6	1.1	-0.7	

is 14.3% for energy use and 15.4% for performance. This error mostly occurs at low thread counts, far from the Pareto front. If we exclude thread counts of four from the RMS error calculation, it drops to 7.1% for energy and 2.8% for performance. The RMS error for the Pareto front is 7.7% for energy use and 8.3% for performance.

The observed and model predicted Pareto fronts show that core and frequency tuning can increase energy efficiency up to 30% but at up to 30% performance loss.

C. Observations

Our results show that the models also perform well on the AMG and LAMMPS applications, which introduce a range of

new dimensions. However, several new challenges are apparent when we model complex workloads.

First, complex applications typically require many system resources, so the cost of obtaining training samples must be amortized over many live runs of an application. We minimize the required payback period by only needing training data for a small fraction of the search space, but our approach is only useful if payback can be achieved. In practice, production applications are run many times with the same data dimensions and can accrue benefits that offset training costs. A weather model that runs daily is a good example. We are also currently researching the practicality of further reducing training costs by using reduced application iteration/time-step configurations for training compared to live runs.

Second, complex interactions between applications and systems may lead to irregular energy or performance response curves, particularly when there is a mismatch between application data/processing dimensions and system topology. Irregular response curves have an impact on the statistical significance of the regression results, so we can alert users that further analysis is needed when the results are not within limits.

VII. 3-Fold Cross Validation

We use cross validation to show that our models do not overfit a subset of the data. With 3-fold cross validation, we randomly partition the data set into three equal sized subsets. Two subsets are used as model training data with the remaining subset used as model test data. We repeat the process three times to test model accuracy across the full data set.

Table VIII shows cross validation results for our energy efficiency models for the study kernels/applications. The data sets for each kernel have 484 samples, so the three folds consist of two folds with 161 samples and one with 162 samples. We show the percentage of predicted values from each fold that are below 20%, 10% and 5% and the RMS Error for the fold. The relatively even RMS Error results across the folds demonstrates that our energy efficiency models are not overfitting any subset of the data.

Table IX shows the results for our performance models using the same data sets. The consistent RMS Error results across the folds show that our performance models are also not overfitting any subset of the data.

TABLE VIII
3-Fold Cross Validation Summary – Energy Efficiency Models

Kernel	Fold	20%	10%	5%	RMS Error %
Stencil	1	100.0	98.1	87.0	3.9
	2	100.0	98.8	91.3	4.1
	3	100.0	97.5	85.1	3.3
Transpose	1	100.0	90.7	81.5	5.1
	2	100.0	96.9	87.0	5.4
	3	100.0	91.3	84.5	4.8
Nstream	1	100.0	92.6	67.3	5.7
	2	100.0	98.1	78.9	6.4
	3	100.0	90.7	69.6	5.6
AMG	1	100.0	100.0	95.1	3.6
	2	100.0	94.4	82.6	4.6
	3	100.0	98.1	91.3	5.8
LAMMPS	1	100.0	90.1	78.4	7.9
	2	100.0	78.9	65.2	9.5
	3	100.0	95.0	82.0	10.6

TABLE IX
3-Fold Cross Validation Summary – Performance Models

Kernel	Fold	20%	10%	5%	RMS Error %
Stencil	1	100.0	94.4	79.0	4.5
	2	100.0	97.5	83.9	5.2
	3	100.0	91.3	81.4	3.9
Transpose	1	100.0	99.4	88.3	3.6
	2	100.0	98.8	87.0	3.9
	3	100.0	95.7	83.2	3.7
Nstream	1	100.0	90.7	66.0	5.6
	2	100.0	94.4	77.6	6.9
	3	100.0	89.4	68.9	5.7
AMG	1	100.0	98.8	93.2	6.1
	2	100.0	93.8	83.9	6.5
	3	100.0	96.3	88.2	7.9
LAMMPS	1	100.0	86.4	77.8	8.0
	2	100.0	87.0	70.8	8.1
	3	100.0	96.9	87.6	8.7

VIII. Conclusion

We developed a methodology to predict optimal performance and energy efficiency settings for parallel kernels and extended our approach to more complex application workloads. To evaluate our technique, we defined the novel concept of a trade-off zone for Pareto optimal fronts that captures the impact of measurement or modeling error. Statistical analysis of our model shows good overall fit between predicted and measured values. We showed that our model can accurately identify trade-offs between performance and energy. Our approach requires measurement samples for a small fraction of the search space, that can be run in parallel when sufficient resources are available. In the future, we will use our model to tune applications automatically according to a specified performance and energy efficiency trade-off policy. Our work will lead to a practical tool that allows scientists to tune their energy use and performance.

Acknowledgment

This research was supported by the Australian Research Council under the Linkage grant scheme (project number LP150100837), and was supported by Cray Inc. This article has been authored by Lawrence Livermore National Security, LLC under Contract DE-AC52-07NA27344 with the U.S. Department of Energy. Accordingly, the United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains, a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or to allow others to do so, for United States Government purposes. Released as LLNL-CONF-755907.

REFERENCES

[1] C. Jin, B. R. de Supinski, D. Abramson, H. Poxon, L. DeRose, M. N. Dinh, M. Endrei, and E. R. Jessup, "A survey on software methods to improve the energy efficiency of parallel computing," *The International Journal of High Performance Computing Applications*, vol. 31, no. 6, pp. 517–549, 2017.

[2] O. Sarood, A. Langer, A. Gupta, and L. Kale, "Maximizing throughput of overprovisioned HPC data centers under a strict power budget," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, Conference Proceedings, pp. 807–818.

[3] P. E. Bailey, A. Marathe, D. K. Lowenthal, B. Rountree, and M. Schulz, "Finding the limits of power-constrained application performance," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 79.

[4] P. Balaprakash, A. Tiwari, and S. M. Wild, "Multi objective optimization of HPC kernels for performance, power, and energy," in *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation*. Springer, 2013, pp. 239–260.

[5] C. Lively, X. Wu, V. Taylor, S. Moore, H.-C. Chang, C.-Y. Su, and K. Cameron, "Power-aware predictive models of hybrid (MPI/OpenMP) scientific applications on multicore systems," *Computer Science-Research and Development*, vol. 27, no. 4, pp. 245–253, 2012.

[6] S. Song, C. Y. Su, R. Ge, A. Vishnu, and K. W. Cameron, "Iso-energy-efficiency: An approach to power-constrained parallel computation," in *2011 IEEE International Parallel & Distributed Processing Symposium*, 2011, Conference Proceedings, pp. 128–139.

[7] J. Hofmann and D. Fey, "An ECM-based energy-efficiency optimization approach for bandwidth-limited streaming kernels on recent Intel Xeon processors," in *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*. IEEE Press, 2016, Conference Proceedings, pp. 31–38.

[8] R. Lavanya, "Energy-time performance of heterogeneous computing systems: Models and analysis," Ph.D. dissertation, 2016.

[9] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding energy consumption in large-scale MPI programs," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. ACM, 2007, p. 49.

[10] C.-h. Hsu and W.-c. Feng, "A power-aware run-time system for high-performance computing," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2005, p. 1.

[11] R. Ge, X. Feng, W.-c. Feng, and K. W. Cameron, "CPU MISER: A performance-directed, run-time system for power-aware clusters," in *Parallel Processing, 2007. ICPP 2007. International Conference on*. IEEE, 2007, pp. 18–18.

[12] V. W. Freeh and D. K. Lowenthal, "Using multiple energy gears in MPI programs on a power-scalable cluster," in *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2005, pp. 164–173.

[13] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos, "Online strategies for high-performance power-aware thread execution on emerging multiprocessors," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 8–pp.

[14] A. Venkatesh, A. Vishnu, K. Hamidouche, N. Tallent, D. Panda, D. Kerbyson, and A. Hoisie, "A case for application-oblivious energy-efficient MPI runtime," in *High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for*. IEEE, 2015, pp. 1–12.

[15] K. Kandalla, E. P. Mancini, S. Sur, and D. K. Panda, "Designing power-aware collective communication algorithms for InfiniBand clusters," in *Parallel Processing (ICPP), 2010 39th International Conference on*. IEEE, 2010, pp. 218–227.

[16] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs," in *SC 2006 conference, proceedings of the ACM/IEEE*. IEEE, 2006, pp. 14–14.

[17] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: Making DVS practical for complex HPC applications," in *Proceedings of the 23rd international conference on Supercomputing*. ACM, 2009, pp. 460–469.

[18] V. W. Freeh, N. Kappiah, D. K. Lowenthal, and T. K. Bletsch, "Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs," *Journal of Parallel and Distributed Computing*, vol. 68, no. 9, pp. 1175–1185, 2008.

[19] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely, "Auto-tuning for energy usage in scientific applications," in *Euro-Par 2011: Parallel Processing Workshops*. Springer, 2011, Conference Proceedings, pp. 178–187.

[20] J. Li and J. F. Martinez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," in *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*. IEEE, 2006, pp. 77–87.

[21] P. Gschwandtner, J. J. Durillo, and T. Fahringer, "Multi-objective auto-tuning with Insieme: Optimization and trade-off analysis for time, energy and resource usage," in *Euro-Par 2014 Parallel Processing*. Springer, 2014, pp. 87–98.

[22] M. Sourouri, E. B. Raknes, N. Reissmann, J. Langguth, D. Hackenberg, R. Schöne, and P. G. Kjeldsberg, "Towards fine-grained dynamic tuning of HPC applications on modern multi-core architectures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 41.

[23] S. F. Rahman, J. Guo, and Q. Yi, "Automated empirical tuning of scientific codes for performance and power consumption," in *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*. ACM, 2011, Conference Proceedings, pp. 107–116.

[24] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe, "OpenTuner: An extensible framework for program autotuning," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*. ACM, 2014, Conference Proceedings, pp. 303–316.

[25] R. Sen and D. A. Wood, "Pareto governors for energy-optimal computing," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 1, p. 6, 2017.

[26] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A roofline model of energy," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, Conference Proceedings, pp. 661–672.

[27] D. H. Woo and H.-H. S. Lee, "Extending Amdahl's law for energy-efficient computing in the many-core era," *Computer*, vol. 41, no. 12, 2008.

[28] S. Cho and R. G. Melhem, "On the interplay of parallelization, program performance, and energy consumption," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 3, pp. 342–353, 2010.

[29] M. Endrei, C. Jin, M. Dinh, D. Abramson, H. Poxon, L. DeRose, and B. R. de Supinski, "A bottleneck-centric tuning policy for optimizing energy in parallel programs," *Advances in Parallel Computing*, vol. 32, pp. 265–276, 2018.

[30] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz, "A regression-based approach to scalability prediction," in *Proceedings of the 22nd annual international conference on Supercomputing*. ACM, 2008, pp. 368–377.

[31] 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*. ACM, 2007, pp. 249–258.

[32] L. Schumaker, *Spline functions: Basic theory*. Cambridge University Press, 2007.

[33] G. Wilkinson and C. Rogers, "Symbolic description of factorial models for analysis of variance," *Applied Statistics*, pp. 392–399, 1973.

[34] R. J. Carroll, D. Ruppert, C. M. Crainiceanu, and L. A. Stefanski, *Measurement error in nonlinear models: A modern perspective*. Chapman and Hall/CRC, 2006.

[35] Python Software Foundation, "Python language reference, version 2.7," http://www.python.org, 2018.

[36] S. Seabold and J. Perktold, "Statsmodels: Econometric and statistical modeling with Python," in *Proceedings of the 9th Python in Science Conference*, vol. 57. SciPy society Austin, 2010, p. 61.

[37] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/

[38] 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.

[39] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.
[40] D. Abramson, R. Sosic, J. Giddy, and B. Hall, "Nimrod: A tool for performing parametrised simulations using distributed workstations," in *High Performance Distributed Computing, 1995., Proceedings of the Fourth IEEE International Symposium on*. IEEE, 1995, Conference Proceedings, pp. 112–121.
[41] L. DeRose, B. Homer, D. Johnson, S. Kaufmann, and H. Poxon, "Cray performance analysis tools," in *Tools for High Performance Computing*. Springer, 2008, pp. 191–199.
[42] R. F. Van der Wijngaart and T. G. Mattson, "The Parallel Research Kernels," in *HPEC*, 2014, Conference Proceedings, pp. 1–6.
[43] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000.
[44] J. Park, M. Smelyanskiy, U. M. Yang, D. Mudigere, and P. Dubey, "High-performance algebraic multigrid solver optimized for multi-core based distributed parallel systems," in *SC15: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2015, pp. 1–12.
[45] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of computational physics*, vol. 117, no. 1, pp. 1–19, 1995, website: http://lammps.sandia.gov.

APPENDIX
ARTIFACT DESCRIPTION: ENERGY EFFICIENCY MODELING OF PARALLEL APPLICATIONS

A. Abstract

This appendix provides instructions for setting up the required tools, launching experiments, and analyzing the results, as presented in the paper.

B. Description

1) Check-list (artifact meta information):

- Algorithm: Regression model
- Program: nimrodo, hpcprobe
- Compilation: Python, gcc
- Transformations: None
- Binary: None
- Data set: YAML
- Run-time environment: CrayPE 2.5
- Hardware: Intel Xeon CPU E5-2699 v4 (Broadwell)
- Output: YAML, Encapsulated PostScript
- Experiment workflow: See section D below.
- Publicly available?: On request

2) How software can be obtained: The source code and dataset archive is available using DOI `doi.org/10.14264/uql.2018.463`. To request access, send an email to `data@library.uq.edu.au`. Select the **HPC Model** dataset.

3) Hardware dependencies: The software has been tested on Intel Haswell and Broadwell servers running CrayPE version 2.5.

4) Software dependencies: Build instructions and dependencies are provided in the dataset archive `README.md` for each program under the following folders:

- `tools/nimrodo`
- `tools/hpcprobe`

Third party source code is available as follows:

- Parallel Research Kernels version 2.17 `github.com/ParRes/Kernels`
- Algebraic multigrid benchmark commit 09fe8a7 `github.com/LLNL/AMG`
- LAMMPS version 14 May 2016 `lammps.sandia.gov`

5) Datasets: All experiment configuration files and YAML results files are available in the dataset archive under the following folders:

- `exp/sc18/config`
- `exp/sc18/results`

File names include the kernel name and node count. All model responses and data plots are generated from these data sets. The full data set includes Stencil node counts of 20, 42, 64, 86, Transpose node counts of 20, 42, 64, 86, Nstream node counts of 20, 42, 64, 86, AMG node counts of 20, 40, 48, 60, 80, and LAMMPS node counts of 20, 40, 60, 80.

C. Installation

To build and install the required tools and test codes:

1) Extract the dataset archive
2) Build nimrodo as per `README.md` instructions from the dataset
3) Build hpcprobe tools as per `README.md` instructions from the dataset
4) Clone the PRK ParRes/Kernels GitHub repository
5) Build the required kernels. We use the MPI/OpenMP stencil, transpose, and nstream kernels
   - Edit `Kernels/common/make.defs` to suit your environment. For CrayPE, we set
     - `MPICC=cc`
     - `CC=cc`
     - `DEFAULT_OPT_FLAGS:=-hpic -dynamic`
   - Make the required kernels, for example
     - `cd Kernels/MPIOPENMP/Stencil`
     - `make stencil`
   - We instrument the kernels with CrayPAT, for example
     - `pat_build -w stencil`

     The job post processing command option mentioned in section F can be used to collect power measurements using another mechanism
6) Clone the LLNL/AMG GitHub repository
7) Build AMG
   - Edit `Makefile.include` to suit, we set
     - `CC = cc`
     - `INCLUDE_CFLAGS = -O2 -DTIMER_USE_MPI -DHYPRE_USING_OPENMP -h omp -DHYPRE_HOPSCOTCH -DHYPRE_USING_PERSISTENT_COMM -DHYPRE_BIGINT -hpic -dynamic`
     - `INCLUDE_LFLAGS = -lm -h omp`
   - Make and instrument AMG

    - `make`
    - `pat_build -w test/amg`
8) Download LAMMPS
9) Build LAMMPS
    - Edit `MAKE/Makefile.mpi` to suit
    - Make and instrument LAMMPS
        - `make -j8 mpi`
        - `pat_build -w lmp_mpi`

D. Experiment workflow

We use the following steps to run experiments and analyze the results for the paper:

1) Create the experiment configuration file. The configuration files for all our experiments are available in the dataset archive under the `exp/sc18/config` folder
2) Launch the experiment using the following command
    - `hpcprobe.py -i <config file>`
3) Analyze the experiment results using the Python API provided by the hpcmodel and hpcplot modules.

Analysis data and plots are generated from the `summary.yml` files output from `hpcprobe.py`. The scripts for generating plots and tables in the paper are available in the dataset archive under the `exp/sc18/analysis` folder. The `README.md` file in this folder provides further detail.

E. Evaluation and expected result

The paper describes our model evaluation process in detail.

There are several log files to monitor that an experiment is progressing as expected:

1) `log/exp.log` for overall experiment logs
2) `log/job.out` for Nimrod/O logs
3) `log/qsub.out` for PBS logs
4) `log/job-<id>.out` for each measurement log

The scripts we used for generating the experimental and model data in the paper log to standard output. The logging level can be set via the command line with the `--info` or `--debug` arguments.

F. Experiment customization

Customization options for the hpcprobe tools include:

- Measurement data parsing using regular expressions defined in the `hpcplot.yml` file;
- Experiment initialization using configuration files. Configurable options include PBS job submission details, software module requirements, job post processing command details, and repeat count for measurement confidence intervals;
- Results analysis using Python scripts with the hpcmodel and hpcplot modules. The hpcmodel module implements the regression models in the paper. Plots can be generated from experiment `summary.yml` files using the hpcplot module.