Machine Learning-Based Temperature Prediction for Runtime Thermal Management across System Components

Kaicheng Zhang, Akhil Guliani, Seda Ogrenci-Memik, Gokhan Memik, Kazutomo Yoshii, Rajesh Sankaran, Pete Beckman

Abstract—Elevated temperatures limit the peak performance of systems because of frequent interventions by thermal throttling. Non-uniform thermal states across system nodes also cause performance variation within seemingly equivalent nodes leading to significant degradation of overall performance. In this paper we present a framework for creating a lightweight thermal prediction system suitable for run-time management decisions. We pursue two avenues to explore optimized lightweight thermal predictors. First, we use feature selection algorithms to improve the performance of previously designed machine learning methods. Second, we develop alternative methods using neural network and linear regression-based methods to perform a comprehensive comparative study of prediction methods. We show that our optimized models achieve improved performance with better prediction accuracy and lower overhead as compared with the Gaussian process model proposed previously. Specifically we present a reduced version of the Gaussian process model, a neural network–based model, and a linear regression–based model. Using the optimization methods, we are able to reduce the average prediction errors in the Gaussian process from $4.2\,^{\circ}\mathrm{C}$ to $2.9\,^{\circ}\mathrm{C}$. We also show that the newly developed models using neural network and Lasso linear regression have average prediction errors of $2.9\,^{\circ}\mathrm{C}$ and $3.8\,^{\circ}\mathrm{C}$ respectively. The prediction overheads are $0.22\,\mathrm{ms}$, $0.097\,\mathrm{ms}$, and $0.026\,\mathrm{ms}$ per prediction for reduced Gaussian process, neural network, and Lasso linear regression models, respectively , compared with $0.57\,\mathrm{ms}$ per prediction for the previous Gaussian process model. We have implemented our proposed thermal prediction models on a two-node system configuration to help identify the optimal task placement. The task placement identified by the models reduces the average system temperature by up to $11.9\,^{\circ}\mathrm{C}$ without any performance degradation. Furthermore, these models respectively achieve $75\,\%$, $82.5\,\%$, and $74.17\,\%$ success rates in correctly pointing to those task placements with better thermal response, compared with $72.5\,\%$ success for the original model in achieving the same objective. Finally, we extended our analysis to a 16-node system and we were able to train models and execute them in real time to guide task migration and achieve on average $17\,\%$ reduction in the overall system cooling power.

Index Terms—thermal modeling, many-core processors, operating systems, high performance computing systems

1 Introduction

Thermal implications are becoming increasingly influential in determining the cost of operating a high-performance computing (HPC) system. On one hand, modern supercomputers consume an enormous amount of power, where a significant fraction is dedicated for cooling [1]. Tianhe-2, the No.1 system currently in the top500 list, consumes up to $17\,\mathrm{MW}$ with approximately an additional 7MW for cooling. This places current HPC systems and future exascale systems under immense pressure to manage the cooling complexity and cost of the resulting heat output. Roadmaps target the design of hardware and system software to draw electrical power of 20 MW using 100 million processors in the 2020 timeframe [2]. One current trend is allowing warmer inlet water temperatures for the liquid cooling commonly used in HPC systems. Several HPC systems operate under raised inlet coolant temperatures. These systems aim to aggressively exploit the thermal headroom (i.e., guard band) in the server chips.

On the other hand, chips manufactured with advanced processing technologies exhibit increasing variation in performance and vulnerability to thermal stress. All major processor manufacturers correlate the maximum expected performance with the thermal design power (TDP), which places conservative bounds (by either limiting clock frequency or imposing power caps) on achievable performance to ensure reliable and consistent performance across all system nodes. Pushing the thermal headroom margins in an effort to minimize cooling cost at the expense of the underlying hardware then becomes a source of performance bottlenecks.

Thermal stress induced on hardware by these increasingly aggressive management schemes needs to be mitigated. Therefore, the system management modules at all levels, ranging from rack-level management to the operating system need to be enhanced with thermal-aware policies.

A complexity that arises in thermal management in large-scale systems is the effect of physical properties (e.g., physical location, access to cooling, and the inherent variation among seemingly identical processors in response to raised temperatures) on the thermal behavior. Our previous work [3] demonstrated that two identical CPUs placed in

- *Kaicheng Zhang, Akhil Guliani, Seda Ogrenci-Memik, Gokhan Memik are with the Department of Electrical Engineering and Computer Science, Northwestern University, 2145 Sheridan Road, Evanston, IL 60208. E-mail: {agn253,kaicheng.zhang, seda, memik}@eecs.northwestern.edu*
- *Kazutomo Yoshii, Rajesh Sankaran, Pete Beckman are with the Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439. Email: {kazutomo, rajesh, beckman}@mcs.anl.gov*

Manuscript received March 1, 2016.

different physical locations of the same system will exhibit different thermal responses even though they are running the same application. On a two-node system (Intel Xeon Phi coprocessors) we observed a difference of over $20\,^{\circ}\mathrm{C}$ between two identical nodes running the same application. In turn, this varying thermal response can cause these seemingly identical subcomponents to perform non-uniformly, contrary to the abstract assumption of the operating system about these hardware components. As a result of such temperature gradients different nodes are subject to varying amounts of thermal throttling (e.g., DVFS, power capping). In certain cases a particular node may be subject to throttling consistently more often than another certain node. Such imbalance widens the performance gap between different system nodes. Our experiments revealed that throttling even a single thread among 128–169 (the number depends on the application) threads results in a 31.9% average system performance degradation.

Previously, we presented a methodology for characterizing the thermal response across system nodes with the ability to capture inherent variations using the Gaussian process method[3]. The resulting temperature prediction model for compute nodes can be utilized by varying levels of the system management modules to execute static thermal-aware policies. In this work, we extend this framework towards a wide variety of machine learning-based methods and demonstrate improved accuracy as well as reduced overhead in capturing the thermal response of compute nodes in a system. The resulting predictors are highly suited for dynamic/run-time management modules. Furthermore, we achieved a deeper understanding of the large space of system parameters that are used to develop the thermal models and performed a systematic analysis of their relative importance. In order to demonstrate the benefit of a thermal-aware policy guided by the resulting thermal models, we implemented a task placement method.

Specifically, we used two lightweight machine learning methods:(i) artificial neural networks (ANN) and (ii) least absolute shrinkage and selection operator (lasso) regression. ANN's have been used for time series forecasting since 1980s, with over 5,000 publications until 2007 [4]. ANN's are widely used because they are capable of handling noisy and incomplete data. They are also able to deal with non-linear systems, and once trained can predict at a high speed [5]. Lasso linear regression is another popular regression analysis method that is also capable of variable selection. This helps enhance the prediction accuracy, reduce the overhead, and improve the interpretability of the statistical model it produces [6].

We demonstrate that we are able to reduce the error in temperature prediction in the Gaussian process model from $4.2\,^{\circ}\mathrm{C}$ to $2.9\,^{\circ}\mathrm{C}$ using the feature reduction methods. Furthermore, the newly developed models using neural network and lasso linear regression reduced errors to $2.9\,^{\circ}\mathrm{C}$, $3.8\,^{\circ}\mathrm{C}$ respectively. We also show that using the new models we were able to reduce the overhead of prediction to $0.22\,\mathrm{ms}$, $0.097\,\mathrm{ms}$ and $0.026\,\mathrm{ms}$ from $0.57\,\mathrm{ms}$ per prediction for Gaussian process using reduced Gaussian process, neural network and lasso linear regression respectively.

We finally demonstrate the effectiveness of the new models for task scheduling in our target system. Specifically, we consider the placement of two random applications on the two different cards. Our model predicts the thermal response of the two possible allocations and then selects the one that results in lower average temperature for the hottest component in the system. We provide three versions of our prediction mechanism. Our results show that our model is capable of scheduling tasks with awareness of thermal impact, identifying the thermally optimal schedule (under the same performance) with a success rate of $75\,\%$, $82.5\,\%$ and $74.17\,\%$ for reduced Gaussian process, Neural network and Lasso linear regression respectively compared to $72.5\,\%$ in our previous work. We show that these scheduling schemes guided by our model can decrease the peak temperature of the system by as much as $11.9\,^{\circ}\mathrm{C}$ ($4.4\,^{\circ}\mathrm{C}$ on average). We also extended our analysis to a 16-node system and we were able to train models and execute them in real time to guide task migration and achieve on average $17\,\%$ reduction in the overall system cooling power.

The rest of this paper is organized as follows. Section 2 gives an overview of related work. In Section 3, we present our motivational experiments. We present the Gaussian process based thermal characterization methodology developed previously, our approach to optimize the model using feature selection and the newly developed models in Section 4. Section 5 presents our experimental results. We discuss possible future directions in Section 6 and summarize our main findings in Section 7.

2 Related Work

Several prior studies have focused on guiding fine-grained core-level dynamic voltage and frequency scaling schemes with direct physical sensor feedback or with prediction models [7, 8, 9, 10]. Other studies have focused on system management tasks such as task migration and load balancing [11, 8, 12, 13, 14, 15, 16, 17, 18]. Choi et al. investigated thermal-aware temporal and spatial mitigation schemes [11]. They conducted experiments on a POWER5 system, which allowed them to access the 24 on-chip thermal sensors at every OS scheduler tick. They modified the task scheduler in the Linux kernel to receive feedback from physical sensors and demonstrated their hot-spot mitigation technique. However, their scheme relies on direct feedback from a large network of well-calibrated sensors, which POWER5 provides, but not many other systems do.

Ramos and Bianchini [13] proposed a model that predicts the thermal impact of a given thermal management policy. Their model is based on heat transfer relationships and therefore requires detailed knowledge of the underlying hardware system and its thermomechanical properties.

Yang et al. [19] showed that application ordering on a process causes a different thermal profile. They proposed a heuristic technique to predict such a behavior and developed a kernel-level thermal-aware scheduler. Their model is heuristics based, meant for reducing dynamic thermal management within core hardware, and does not consider HPC applications.

Piatek et al. [20] proposed a framework for thermal modeling of HPC systems by considering the node arrangement, power consumption, and cooling system configuration to estimate the thermal parameters and develop a thermal

model of the system. Their model is specific to system configuration such as cooling system type, power consumption metrics, and node type and requires considerable effort to build. In contrast, in our approach we leverage the knowledge available only to the node and do not require any specific information about the system.

Moore et al. [21] proposed a framework to build a thermal model for a datacenter using readings taken from external temperature sensors, server instrumentation, and computer room air conditioning units. Their framework leverages a neural network–based method. Ultimately, this model is used to provision a power budget across the datacenter, and it does not have a notion of real application workloads. Hence, the model has no learning component for the workloads. The model is trained only to learn the system, unlike our model, which is capable of inferring both the system and the actual applications.

Chaudhary et al. [22] presented a survey of power saving techniques for green datacenters. They also provided a detailed survey of thermal-aware schedulers that use thermal models for prediction and policy implementation.

Most previous works that developed prediction methods do not target a true HPC system or HPC workloads. Those that study thermal-aware management for HPC systems, on the other hand, utilize only limited physical thermal sensor feedback or require detailed knowledge of the thermomechanical properties of the system.

In our previous work [3], we developed a framework that provides a novel application of the Gaussian process–based machine learning method, solely leveraging the system status accessible to the operating system, without requiring physical or domain-specific information about the system. This thermal characterization framework extends beyond the level of processor cores and is capable of characterizing higher-level system components, such as nodes. In contrast, most related works focus solely on predicting and mitigating within-core and across-core thermal variation.

For this framework two kinds of models were developed: (1) a decoupled model, using information only from the node under observation during analysis; and (2) a coupled model, using information from all the nodes in the system in addition to the node under observation during analysis. The coupled model is more comprehensive yet it does not scale well in terms of adding new nodes.

In this paper, we extended this framework and the thermal models with the following specific new contributions: We identify the opportunities to improve the accuracy of the decoupled models and enhance their suitability for dynamic prediction by minimizing their overhead. Specifically, we reduce the set features used to predict the state of the system. We leveraged feature selection [23] and Hall et al. [24] for this purpose. We also explored other machine learning methods to extend the existing framework and achieved new models with lower overhead.

3 Motivation: Thermal Analysis of a HPC System

Our work has been motivated by a detailed thermal analysis of representative HPC systems. We have thermal analysis data collected from three systems: (1) inlet coolant temperature data across nodes in the Mira Blue Gene/Q supercomputer [25]; (2) a thermal map from a two-node system based on the Intel Xeon Phi implemented as PCIe cards, where each card contains one processor with 61 cores and four hardware threads per core; and (3) core-level temperature data from a two-package Intel Sandy Bridge processor configuration. The data from Mira was provided to us by a third party [25]. We collected the data for the other two systems ourselves, using microbenchmarks.

The variation in thermal behavior is significant in all three cases. In the Intel Xeon Phi-based system, we observe over 20 °C difference in temperature between the hottest and coldest cards under the same workload. Furthermore, the upper card is always consistently hotter than the lower card. This specific system has also served as our testbed for the remainder of our experiments in this study. In the Sandy Bridge system, a clear variation in temperature occurs among the eight cores within the same package as well as between the two packages.

Thermal variation is present in different levels of the system, not only among individual cores, but also across nodes; and there are thermal hotspots that largely deviate from the average behavior of the systems, causing imbalance in performance, excessive power consumption, and threats to reliability. The distribution of the hotspots and the variation in peak temperature across different system components can be attributed to two factors. First, it depends on the physical attributes of the hardware components. For example, presumably the reason for the upper card in the Intel Xeon Phi-based system being hotter than the lower card is that the upper card intakes hotter air. Second, the distribution depends on the placement of the workloads. To demonstrate how placement can affect hotspots, we performed an experiment on our testbed. We ran every possible pair of applications from a benchmark set on the two-node system. We performed two runs for each pair, swapping the placement of the applications between the top and bottom card. We found that between two alternative mappings of a pair of applications to the two cards, the difference between the peak observed temperature in the system can be as high as $11.9\,^{\circ}\mathrm{C}$. Clearly, some mappings incur significantly more thermal stress than other mappings do.

These observations motivated us to design tools for making thermal-aware decisions, mitigating the problem of thermal variation and reducing the hotspots in the system. In our previous work [3] we developed a new framework to characterize the thermal behavior of a HPC geared toward capturing thermal hotspots at system-level components. In our present work we extend this framework to use more machine learning models and fewer features in order to both improve accuracy and reduce overhead, making it more suitable for dynamic prediction.

4 Model for Predicting System Temperature

In this section, we first provide the background on the framework developed in our previous work. We then describe the evaluation criteria used to select various prediction models. Note that, we used standard libraries for

providing us with optimized implementations of each of the model types. For each model we applied a 10-fold cross-validation. Next, we present the details of the optimizations and prediction methods developed during this work. We conclude the section with a discussion on the complexity and overhead of the various methods.

4.1 The Thermal Characterization and Prediction Framework

The baseline temperature prediction model for an HPC system is designed to reflect the physical and architecture-driven sources of variation. The methodology used to create this model comprises five steps.

1) For a specific machine in the system (e.g., a particular node in the cluster), we run a series of benchmarks on that machine and collect data on their performance related characteristics (such as cache misses, fraction of floating-point instructions). These properties are largely correlated with the application's own nature. Hence, they are invariant as long as the architectural configuration of the machine remains the same. The thermal responses of these benchmarks on this machine are also collected during these runs.
2) Using the cumulative data collected from all benchmarks, we generate a machine-specific model that maps representative application characteristics and an initial physical state to a predicted temperature expected to manifest on that machine after a certain time interval.
3) Independently, for the same physical system, for each actual target application of this HPC system, we collect a time series set of samples of application-dependent properties, which are stored in log files by the system software.
4) When an application has to be scheduled on the HPC system, we use the machine model from Step 2, which is generic and was built without any information directly from this particular target application. We also use application characteristics of this specific target workload preprofiled in Step 3 to predict the operating temperature.
5) Using the predicted operating temperatures, we compare alternative task assignments. The system software is provided with a suggestion for an assignment that is expected to result in lower average temperature for the hottest node.

The framework is designed to be general so that it can apply easily to a different architecture as well as at a different level of granularity (individual chip, node, rack, etc.). The block diagram for the framework is given in Figure 1. Ultimately the output of the model is the thermal response of the system.

We outline the two main stages of the model development: (i) collection of data from various sources as described in Steps (1) and (3) above and (ii) construction of a thermal model.

For a given system, a set of features at a time t comes from various sources, including hardware performance counters, temperature sensors, and kernel counters. Some of these features represent application characteristics and do not change significantly when running the application on different nodes of the system. We call these features *application features*, and we denote the values of these features at time t as a vector $\boldsymbol{A}(t)$. The remaining features represent a node's physical condition. Even while running the same application, these features can vary dramatically across nodes depending on each node's cooling conditions, location, and so forth. For example, temperature sensor readings are in this category. We call these features *physical features*, and we denote them as a vector $\boldsymbol{P}(t)$. In Section 5, we provide a full list of these features.

The framework uses a set of representative benchmark applications to characterize a node and obtain a machine-specific model for that node. In our present work we extract representative features from the actual workload set of an HPC system. Each target application is run on the node, and its application features and physical features are collected. We set the length of the profiling run to ensure that each application reaches well beyond its sections of interest. At the same time, this duration is set to be long enough for the system to reach its thermal steady-state.

Based on how we use the features, we have two choices for constructing the thermal prediction system:

1) Use the application and physical features from only the target node. This model reflects a node's thermal response completely independent of any thermal coupling with other nodes in the system. This is referred to as a *decoupled model*.
2) Use the application and physical features from the target node and the surrounding node. The activity of nearby nodes can represent potential thermal coupling that may be present in the system. This is referred to as a *coupled model*.

Training for decoupled models and execution of decoupled prediction are highly scalable. Introducing coupling information requires the collection of features from one application or node while mapping another paired application onto all nearby nodes in different combinations. This process was discussed in [3], where we showed that the accuracy of the model is minimally improved by considering coupling between system nodes.

We leverage the collected features using a variety of machine learning methods to generate the prediction model. In Sections 4.2 we discuss the selection of these models for our study, and in Sections 4.3, 4.6, and 4.5 we describe the selected models. Basically, the model is embodied in the form of a function f_j generated for given node j, such that

$$\boldsymbol{P_j}(i) = f_j(\boldsymbol{A}(i), \boldsymbol{A}(i-1), \boldsymbol{P}(i-1)). \quad (1)$$

The resulting temperature prediction model is then used as follows. At some point in time, the task scheduler considers assigning an application to a specific node in the system. At that instant, our model is invoked. It receives the representative preprofiled application features of this application in the form of a series of samples $(\boldsymbol{A}(1), \boldsymbol{A}(2), \ldots, \boldsymbol{A}(N))$. It also receives from the system the state of the initial physical features of the node. It then iterates through the time series of the pre-profiled data and at each step makes

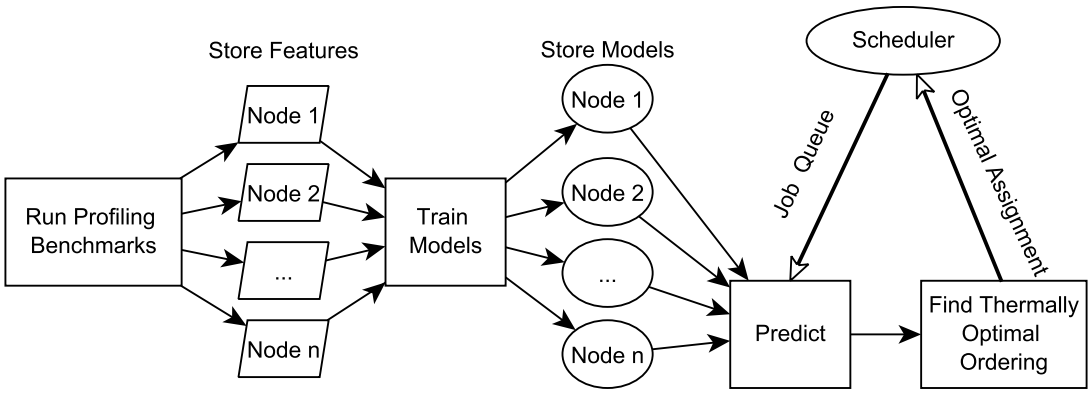
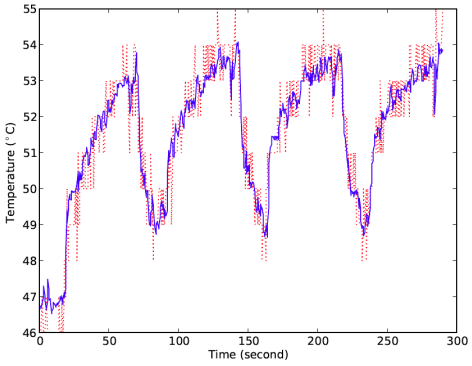
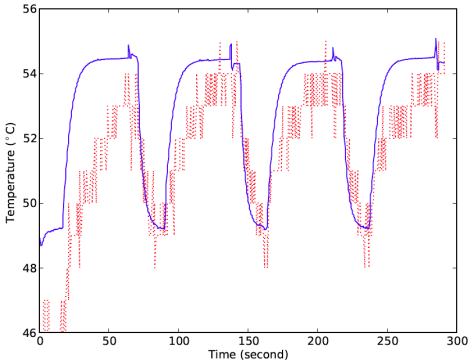


Fig. 1: Representative block diagram of the framework

a temperature prediction. In this way, the model generates the expected thermal response over a period of time.



(a) Online prediction using our model



(b) Static prediction using our model

Fig. 2: (2a) Online temperature predictions of our model (blue solid line) versus actual temperature sensor readings (red dotted line). (2b) Static temperature predictions of our model (blue solid line) versus actual temperature sensor readings (red dotted line).

The model can be used in an online fashion to predict temperature from application characteristics sampled by the system at run time. A thermal response generated by the model used in online mode is depicted in Figure 2a. In this figure the thermal response derived from actual temperature sensors on a node (dotted line) is plotted along with the predicted response generated by the model (solid line). We can see that the model has high accuracy (less than 1°C difference on average) when used online. The model can also be used statically to assess thermal peaks in steady state that would result from a given task assignment. Static prediction is what we have focused on in this work in order to aid static task assignment decisions. Here, the prediction is not intended to calculate a spontaneous thermal response with absolute accuracy; rather, it should be able to predict steady-state behavior as accurately as possible. Also the prediction mechanism should be capable of capturing significant fluctuations in the thermal response over time. The output of the model developed previously, used statically, is illustrated in Figure 2b. As one can see, the model successfully captures long-term fluctuations and the steady-state behavior of the thermal response.

The effectiveness of such a universal model in aiding a static task scheduler was evaluated previously. Dynamic scheduling, aided by the model, would be feasible as far as the accuracy of the temperature prediction goes. In our present work we improve the suitability of the prediction methods by improving accuracy and reducing prediction overhead as discussed in Section 4.4 and Section 4.7. However, the effectiveness of the resulting dynamic scheduling, including migration overheads and the like, requires further careful study. We plan to investigate this avenue in our future work.

4.2 Evaluation of Alternative Machine Learning Approaches

The effectiveness of the framework for prediction depends on the ability of the function f, described in Equation (1), to capture the thermal characteristics of the system. Note that there are no assumptions of any detailed knowledge about the underlying system. For instance, the framework is unaware of placement of nodes (e.g., which card is far from and which card is near the cooler air flow) and the geometry of the system. Also, the framework does not make use of any domain-specific knowledge pertaining to thermal modeling. In other words, the framework has no knowledge of the thermal transfer properties of the materials involved. It operates as a simple mapping function between features that are accessible to an operating system and the expected temperature at a node. This approach makes machine learning methods a natural candidate for our purposes. We evaluated a large number of machine learning methods, and we used WEKA [26] to explore various regression methods. From these, we selected three methods and performed a more in-depth evaluation of each.

The main evaluation criterion was how well each method can predict the temperature for a given state of application features dt seconds into the future. Figure 3 presents our evaluation of several methods in terms of mean absolute error in prediction versus the length of the prediction window. We tested these methods to predict as far as 25 seconds into the future. From Figure 3 we observe that for all models the prediction errors tend to grow as the prediction window extends farther into the future. In addition, we observe that some of the techniques, such as neural networks and Bayesian networks, experience instabilities. Linear regression models exhibit acceptable performance, particularly for shorter prediction windows. Overall, the Gaussian process, among all the methods, has the best prediction accuracy until the prediction time window reaches 25 seconds.

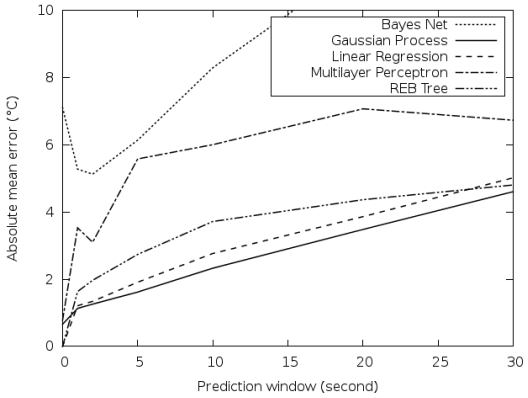


Fig. 3: Performance of different machine learning methods when predicting future temperatures.

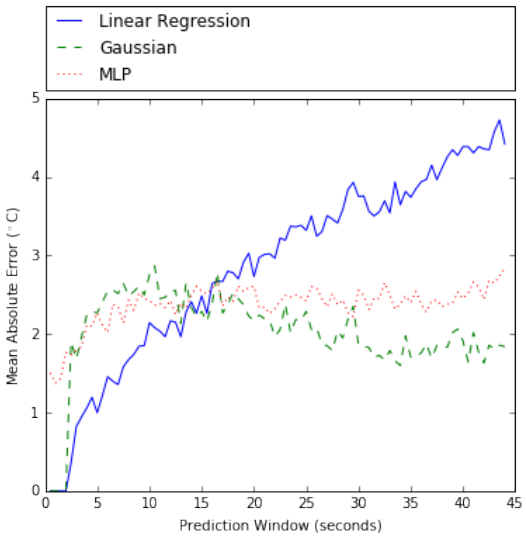


Fig. 4: Performance of different machine learning methods using optimized models for predicting future temperatures.

We also took a closer look at three models: linear regression, Gaussian process regression, and a neural network method called multilayer perceptron (MLP). In this analysis we developed optimized models for them and then compared the results. To have deeper insight into the methods, we predicted one time step into the future at a time and used this predicted value as part of the input for the next prediction until reaching dt seconds into the future for a given set of applications. This strategy is more realistic for assessing the suitability of the methods for a dynamic prediction scheme. Figure 4 presents our evaluation of these three methods in terms of mean absolute error in prediction versus the length of the prediction window. We observe a similar growth in the errors with an increase in the time window. We also observe that the neural network model is stable in this experiment compared with previous results. Moreover, the performance in terms of errors for the three methods at the 20-second mark is comparable. The Gaussian process method is still the best-performing model when the prediction reaches 30 seconds, but MLP is a close second.

Based on this analysis we decided to optimize the Gaussian process model by reducing the feature set. The method used for feature selection for the reduced Gaussian process is discussed in Section 4.4. Further, we decided to implement a linear regression and MLP model and study their suitability in terms of prediction error and overhead. The models are discussed in Sections 4.5 and 4.6, respectively. The overhead analysis for all the methods is detailed in Section 4.7. The suitability in terms of prediction errors is detailed and the results are presented in Section 5.

4.3 Prediction Model Using The Gaussian Process Method

A Gaussian process [27] is a stochastic process of a set of random variables $(X_1, X_2, \ldots)$, where any subset of these variables constitutes a joint Gaussian distribution.

We assume that any subset of elements of the physical attribute vectors in our problem $(\boldsymbol{P}(i_1), \boldsymbol{P}(i_2), \ldots, \boldsymbol{P}(i_n), \boldsymbol{P}(i_{n+1}))$ similarly has a joint Gaussian distribution:

$$(\boldsymbol{P}(i_1), \boldsymbol{P}(i_2), \ldots, \boldsymbol{P}(i_n), \boldsymbol{P}(i_{n+1})) \sim \mathcal{N}(0, \boldsymbol{K}), \quad (2)$$

where $\boldsymbol{K} \in \mathbb{R}^{(n+1)\times(n+1)}$ is the covariance matrix. The selection of 0 as the mean of the Gaussian distribution is a common choice. The elements of this matrix are computed by a *kernel* function k:

$$\begin{aligned} K^{(j,k)} &= k(\boldsymbol{X}(i_j), \boldsymbol{X}(i_k)), \\ \boldsymbol{X}(i_j) &= (\boldsymbol{A}(i_j), \boldsymbol{A}(i_j - 1), \boldsymbol{P}(i_j - 1)), \\ \boldsymbol{X}(i_k) &= (\boldsymbol{A}(i_k), \boldsymbol{A}(i_k - 1), \boldsymbol{P}(i_k - 1)). \end{aligned} \quad (3)$$

The kernel function essentially evaluates the correlation between two samples $\boldsymbol{X}(i_j)$ and $\boldsymbol{X}(i_k)$ and populates the covariance matrix. For a set of observations $\boldsymbol{P}(i_1), \boldsymbol{P}(i_2), \ldots, \boldsymbol{P}(i_n)$ and $\boldsymbol{X}(i_1), \boldsymbol{X}(i_2), \ldots, \boldsymbol{X}(i_n), \boldsymbol{X}(i_{n+1})$, the model then generates the expected value of $\boldsymbol{P}(i_{n+1})$ (i.e., the prediction) as follows:

$$\begin{aligned} &\mathbb{E}(\boldsymbol{P}(n+1)|X, P, \boldsymbol{X}_{N+1}) = K(X_{i_{n+1}}, X)K(X, X)^{-1}P, \\ &\text{where, } P = (\boldsymbol{P}(i_1); \boldsymbol{P}(i_2); \ldots; \boldsymbol{P}(i_n)), \\ &X = (\boldsymbol{X}(i_1); \boldsymbol{X}(i_2); \ldots; \boldsymbol{X}(i_n)). \end{aligned} \quad (4)$$

Note that $K(X, X)^{-1}\boldsymbol{P}$ can be precomputed and reused each time a prediction is made. Hence, the matrix inversion step of this precomputation occurs only once.

Essentially, X and P constitute the training set for our model, since they are the observations that are used to estimate $\boldsymbol{P}(i_{n+1})$. Hence, the temperature prediction model f that we aimed at can be expressed as follows.

$$f = \mathbb{E}(\boldsymbol{P}(i_{n+1})|X, P, (\boldsymbol{A}(i_{n+1}), \boldsymbol{A}(i_{n+1}-1), \boldsymbol{P}(i_{n+1}-1))) \tag{5}$$

4.4 Feature Selection

The feature selection method used is based on the correlation feature selection (CFS) algorithm [28]. In our work we used the CfsSubsetEval method provided by WEKA data mining software [26] for preliminary analysis. A detailed analysis of the dataset then was done by using correlation analysis and hierarchical clustering methods, provided by Rattle [29], to identify relationships among the features and to validate the relevance of the selected features. We iteratively tested the regression model with the selected features obtained above and used selectKbest search, provided by the Scikit-Learn Python package [30], to arrive at 16 features. This reduced feature set provided us with improved accuracy and reduced overhead compared with using all the features while performing the prediction.

In this work we used a linear regression–based model as the regressor for the selectKbest method. With this model we repeatedly use cross-validated regression weights to identify the K best features that are used repeatedly by the model. The good performance of the linear regression model during model selection in Section 4.2 on our dataset indicated that this technique is a good fit for our case. The results for this feature selection are discussed in Section 5.

We also tested principal component analysis [31] as a possible feature reduction method. This technique is widely used; it transforms the given input feature set into a reduced set of principal components that are then fed into the learning model. The algorithm requires this transformation as the first preprocessing step before the machine learning model. Since we are targeting an HPC environment, this added preprocessing set would have increased the total overhead for the model. Hence we chose a simpler method.

4.5 Prediction Model Using Lasso Linear Regression

Least absolute shrinkage and selection operator (Lasso) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces. Introduced by Tibshirani in 1996 [6], it is an extension of least-squares optimization with L1-norm regularization; in other words, the method penalizes the feature weights if the feature introduces errors for the training set. This method is well suited to our case because it does an additional step of feature selection during training, producing more accurate results than with simple linear regression. Once this model is trained, it has the lowest overhead for prediction because the output is equal to the sum of the weighted inputs [32] [33]. For our analysis we use the implementation provided by the Python package Scikit-Learn [30]

4.6 Prediction Model Using Multilayer Perceptron

The artificial neural network (ANN) is an information-processing paradigm inspired by the way biological nervous systems, such as the brain, process information. It comprises a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. These elements can be physical devices, computer programs, or pure mathematical constructs. ANN is widely accepted as a machine learning method offering an alternative way to tackle complex and ill-defined problems. ANNs learn from examples; they are fault tolerant, since they can handle noisy and incomplete data; they are able to deal with nonlinear problems; and, once trained, they can perform prediction and generalization at high speed [5]. ANNs have been widely used for time series forecasting since the 1980s. As indicated by Crone and Nikolopoulos [4], by 2007, there have been more than 5,000 publications on ANNs for forecasting. The well-known ANN predictors can be listed as multilayer perceptron (MLP), recurrent neural networks (RNN), and radial basis function networks (RBFN). To train these models, that is, to find the suitable parameters for them, gradient descent methods are often used.

In our case we use MLP. It is a feed-forward artificial neural network model that maps sets of input data onto a set of appropriate outputs. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron (or processing element) with a nonlinear activation function [34].

In our study we use the Pybrain neural network library [35] for python to develop the MLP predictor. Designing an ANN typically requires the following five basic steps:

1) Collecting data: ANNs are as good as the data that is used to train them. Hence, collection of good-quality data and a significant amount of data is of paramount importance for building a successful ANN model. The data collection for our model is explained in Section 4.
2) Preprocessing data: In this step the input data collected is normalized to range from -1 to 1. According to the literature, normalization of the dataset improves the training efficiency [36] and reduces errors in the output activation function.
3) Building the network: An ANN is built by specifying the number of hidden layers, neurons in each layer, transfer function in each layer, training function, weight/bias learning function, and performance parameter to be checked during training. For our model we use an MLP network with one input layer, two hidden layers, and one output layer. The output activation function used is Sigmoid. These choices allow for identifying regions in the data with arbitrary complexity [5]. The parameters defining the configurations of the layers were determined empirically. Using at least twice as many neurons as there are inputs in the first layer provided a good approximation to extrapolate the information for the given number of inputs. In the second hidden layer, we now compress the information to one-fourth of

the number of neurons in the first hidden layer leading to the model output layer.

4) Training the network: During the training process, weights are adjusted in order to make actual outputs close to the target outputs of the network. MLP utilizes a supervised learning technique called back propagation for training the network. [36]
5) Testing the performance: For this step we use leave-one-out cross-validation and testing mechanism, where we keep all the data for one benchmark application out of the training set and use it to test the performance of the model. To eliminate bias, we do this iteratively for all benchmarks.

4.7 Complexity Analysis and Runtime Overhead

The complexity and runtime overhead analysis for the three algorithms is presented in the following subsections.

4.7.1 Analysis of the Gaussian Process

The main contributor to the computational complexity of a Gaussian process is the matrix inversion at the precomputing phase. This inversion operation has a time complexity of $O(N^3)$ and is executed only once. Each subsequent evaluation of f has a time complexity of $O(MN)$, where N is the total number of samples in the training set and M is the number of features of each sample.

To reduce the training and prediction time, we use a variant of the Gaussian process called the subset data Gaussian process. For a large data set, we randomly select $N_{\max}$ samples from the data set, thus limiting $N = N_{\max}$.

Before attaching this model to the system, the model is precomputed offline. Hence, it needs only to simulate the system status. This simulation involves two steps:

- Gathering the current system state $\boldsymbol{P}(1)$. This step requires I/O queries to all the feature sources. We use 30 different sources, which incur a total communication cost of 22 ms.
- Simulating the system for a given time duration. We simulate the system for five minutes. On average, this costs 0.57 ms per prediction and 344.1 ms per application (performing a total of 600 predictions).

To further reduce the training and prediction time, we used feature selection algorithms to select the 15 features, using which we get similar results from the Gaussian process model while reducing the simulation time. The simulation using the reduced Gaussian process on average costs 0.22 ms per prediction and 132.94 ms per application (performing a total of 600 predictions).

4.7.2 Analysis for Multilayer Perceptron

In our search for a low-overhead model we also looked at multilayer perceptron. The time complexity of MLP trained for regression (the forward propagation) can be calculated by evaluating the time complexity of its two major components:

- The number of multiplications needed to compute the activation of all neurons: This is a vector product, and in the i^{th} layer of the network it equals $N(i) * N(i-1)$, where N is the number of nodes in layer. Our network consists of four layers. Assuming the network has n neurons in all, the output layer contains only one neuron for the output of the regression. The remaining $n-1$ neurons need to be distributed among the remaining three layers. We can assume for simplicity that in the worst case the nodes are equally distributed, which would result in $((n-1)/3)^3$ multiplications, making the time complexity of this operation equal to $O(n^3)$.
- The output activation function of neurons. In general these are simple functions; hence one can assume the time costs to be constant per neuron. This assumption implies that given a network with n neurons, the time complexity of this step would be $O(n)$.

The overall complexity would be dominated by the multiplications resulting in an overall complexity of $O(n^3)$. Since we know that the number of neurons for a given problem is fixed, and hence n is constant, the overall complexity is reduced from $O(n^3)$ to $O(1)$.

The simulation time using MLP, on average, is 0.097 ms per prediction and 58.23 ms per application (performing a total of 600 predictions).

4.7.3 Analysis of the Lasso linear regression

Lasso linear regression is a least-squares method. Hence, the overall complexity of a trained model would be dominated by the multiplications of features with their respective weights, namely, $O(MN)$, where M is the number of features and N is the number of input data points.

The simulation time using the Lasso linear regression, on average, is 0.026 ms per prediction and 15.37 ms per application (performing a total of 600 predictions).

5 Experimental Results

The test environment used for this study is an Intel workstation with two Intel Xeon Phi coprocessors. The Intel Xeon Phi coprocessor is a promising platform featuring up to 61 cores and 244 hardware threads per chip. The theoretical peak performance of a 61-core Phi coprocessor is 1.2 Teraflops in double-precision calculations. Each coprocessor runs its independent operating system and communicates with the host system through the PCIe interface. The system management controller in the coprocessor monitors a variety of on-board sensors. Our experiment system consists of two nodes (cards): the bottom card denoted as "mic0" and the top card as "mic1." The configuration of the Phi coprocessor is described in Table 1.

Table 2 shows the full list of benchmarks and their configurations that were used to construct our thermal model. We collected all available system features using a kernel module was previously developed in [3] The kernel module performs the sampling at a fixed interval. Its sampling has a runtime overhead of $20\,\mathrm{ms}$; to balance this cost, we therefore selected a sampling period of $500\,\mathrm{ms}$. For cumulative features such as instruction count, the module records the increase since the last interval. For instantaneous features, the module records the reading of the attribute. Table 3 lists the features collected and their classification. The die

Coprocessor Configuration	
Model #	7120X
# of cores	61
Frequency	1238 MHz
Last Level Cache Size	30.5 MB
Memory Size	15872 MB
Host Configuration	
CPU	E5-2609 v2
Frequency	2.5GHz
# of Cores	4
Last Level Cache	10MB
Memory	32GB

TABLE 1: Intel Xeon Phi coprocessor and host configurations.

App.	Data Size, Parameter	Description
from the CESAR benchmark		
XSBench	default	compute cross sections using the continuous energy format
RSBench	default	compute cross sections using the multipole representation format
from the NAS parallel benchmark		
BT	C	block tridiagonal solver
CG	C	conjugate gradient, irregular memory access and communication
EP	C	embarrassingly parallel
FT	B	discrete 3D fast Fourier transform
IS	C	integer sort, random memory access
LU	C	lower-upper Gauss-Seidel solver
MG	B	multigrid on a sequence of meshes
SP	C	scalar pentadiagonal solver
from the scalable heterogeneous computing benchmark		
FFT	-s 4	fast Fourier transform
GEMM	-s 4	general matrix multiplication
MD	-s 4	performance test for a simplified molecular dynamics kernel
miscellaneous applications		
BOPM	default	binomial options pricing model
HogbomClean	default	Hogbom clean deconvolution
DGEMM	default	double-precision general matrix multiplication by Intel

TABLE 2: Applications used for our experiments.

temperature feature is the one that the framework ultimately predicts.

Because the framework includes temperature sensors, it can be trained to predict the processor temperature as well as the ambient temperature, which also plays an important role in thermal models [37].

We ran each application for five minutes. If the application finished in under five minutes, it was restarted. If the application ran longer than five minutes, it was terminated. We thus ensure that all applications used for profiling perform a major portion of their main body of computation within this duration. Specifically, for the given hardware configuration, five minutes is sufficient time for all the applications to run through their setup phase and reach the steady-state behavior and temperature.

5.1 Feature Selection

We analyzed the collected data for correlations and used the correlation matrix to build a hierarchical cluster of the above mentioned features. The output dendogram of the hierarchical clustering is presented in Figure 5.

The feature selection process resulted in the following 16 parameters: volt, cyc, inst, instv, fp, fpv, fpa, l1dr, l1dw,

Name	Description
App. Features	
freq	Frequency
cyc	# of Cycles
inst	# of Instructions
instv	# of Instructions in V-Pipe
fp	# of Floating-Point Instructions
fpv	# of floating point instructions in V-pipe
fpa	# of VPU elements active
brm	# of branch misses
l1dr	# of L1 data reads
l1dw	# of L1 data writes
l1dm	# of L1 data misses
l1im	# of L1 instruction misses
l2rm	# of L2 read misses
mcyc	# of cycles microcode is executing
fes	# of cycles that front end stalls
fps	# of cycles that VPU stalls
Physical Features	
die	max die temperature from on-die sensors
tfin	fan inlet temperature
tvccp	VCCP VR temperature
tgddr	GDDR temperature
tvddq	VDDQ VR temperature
tvddg	Uncore power
tfout	fan outlet temperature
avgpwr	average power
pciepwr	PCIe input power reading
c2x3pwr	2x3 input power reading
c2x4pwr	2x4 input power reading
vccppwr	core power
vddgpwr	uncore power
vddqpwr	memory power

TABLE 3: List of features collected from the system.

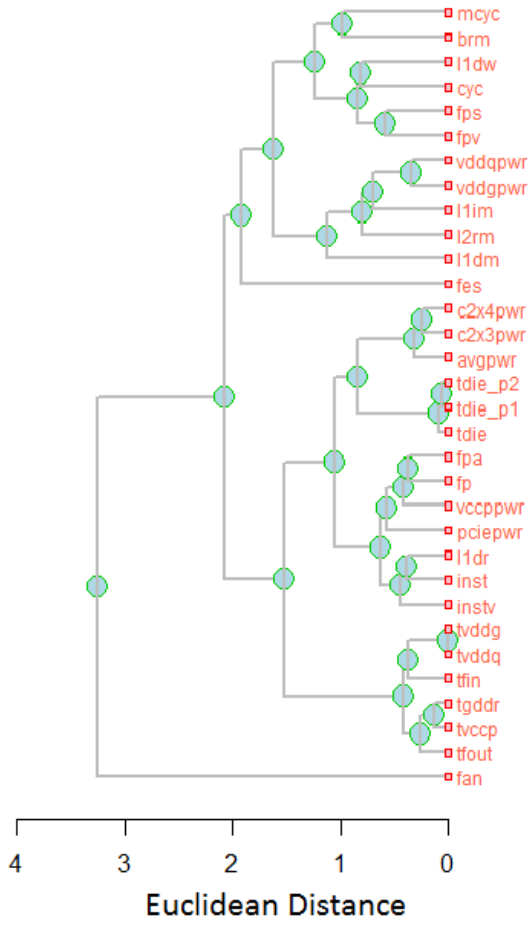


Fig. 5: Dendogram representing the various clusters present among the given features. The x-axis represensts the Euclidean distance between the clusters

l1dm, l1im, l2rm, brm, fes, and fps. Here we observe that most of the features chosen are application features. The only physical features we consider are the die temperature (also the target) and the voltage drawn by the system. Iteratively we found that this combination provides a reasonable steady-state response for the Gaussian process that is approximately three times faster and nearly as accurate as the previously proposed Gaussian process where all the parameters are used.

In a similar manner we found that the MLP model requires only six parameters to achieve a model, which is approximately five times faster than the previously proposed Gaussian process and nearly as accurate. These parameters are fp, fpv, l1dm, l1dr, inst, instv, and tdie. Also note that both these models look back two time steps into the die temperature history.

5.2 Predicting Application Temperature

For predicting the thermal response of a system we first need to train the prediction model. To this end, we randomly select a subset of 500 samples from all the available samples. of the model We limit the total number of samples used for training in order to limit the computation costs and to reduce both the time and space complexity of the model as described in Section 4.7. The number $N_{\max} = 550$ is selected because it provides a good tradeoff between prediction accuracy and model complexity. We highlight that *in all our experiments, the model is trained using samples from all applications except the target application*; in other words, the training model never includes samples from the application(s) used in testing. This method is also referred to as leave-one-out cross-validation.

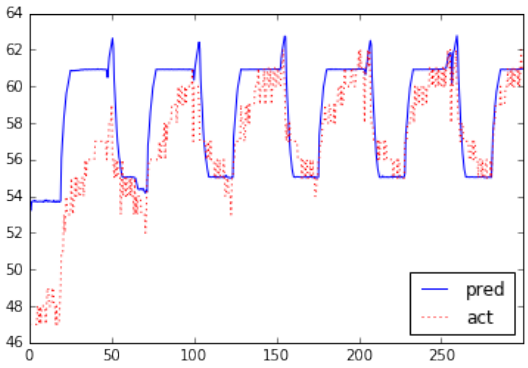
We use symbols $\boldsymbol{A}_{i,X,Y}(j)$ and $\boldsymbol{P}_{i,X,Y}(j)$ respectively to denote the application and physical features of the jth sample we collected on mic$i$, when running application X on mic0 and Y on mic1. Note that when we use $\boldsymbol{A}$ as the application profile, we pick only $\boldsymbol{A}_{i,X,NONE}$, as when we do the profiling, and do not run any applications on the other node. The selection of i depends on the node our model is trained for.

For each application X, we train the model for mic0 using samples from all applications except X. Then we test the model by performing temperature prediction for application X, using X's preprofiled application features collected on mic1. Figure 6 shows the prediction output compared with the actual output for the models.

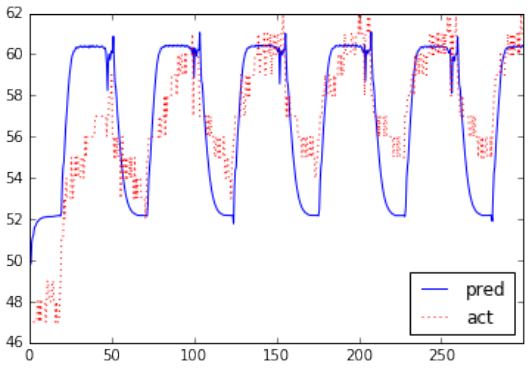
We report the individual peak temperature error and average temperature errors of our models in Figure 7, Figure 8, and Figure 9. The error of our model is computed by taking the difference between our model's prediction for a time instant and the thermal sensor reading collected from that node for the same instant.

We highlight key properties of this experiment setup:

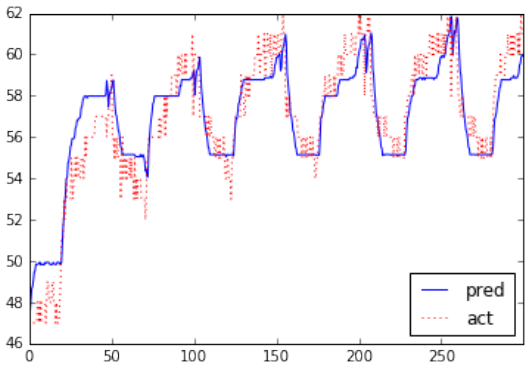
- When predicting for application X, no information of application X is used during the training of the model.
- After training, while we are predicting the temperature for X running on mic0, we actually use the application features of X collected on mic1, in order to validate our assumption that application features remain the same across different nodes.
- For each application, the features are collected only once statically. These logs are repeatedly used for predicting temperature in different machines.



(a) Static prediction using reduced Gaussian process model



(b) Static prediction using MLP model



(c) Static prediction using Lasso linear regression model

Fig. 6: ((6a)(6b)(6c) Static temperature predictions of our models (blue solid line) versus actual temperature sensor readings (red dotted line).

We observe that our model can predict the temperature accurately for most of the applications. Compared with the previously proposed Gaussian process model, the average error in the reduced feature set Gaussian process model decreases from 4.2°C to 2.9°C. For the MLP model the average error decreases to 2.9°C, and for the Lasso regression model the average error decreases to 3.8°C. We believe that we can further improve the prediction accuracy of our model by incorporating more applications during training to cover extreme cases. As we will see in the following section, for task placement problems, some bias error can be canceled

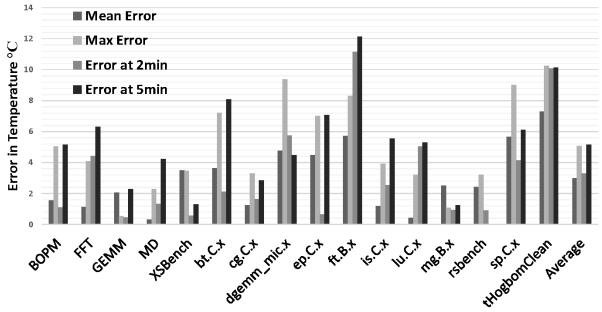


Fig. 7: Temperature prediction error of the decoupled method for Gaussian process with the reduced feature set.

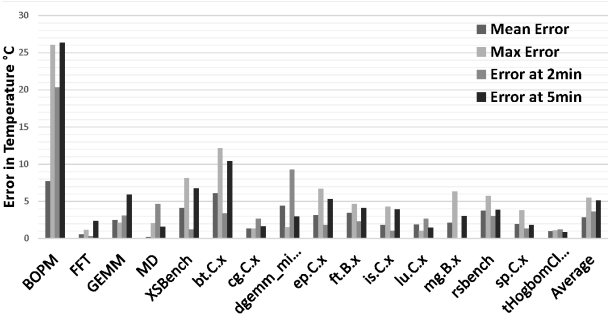


Fig. 8: Temperature prediction error of the decoupled method for MLP.

out and lead to correct scheduling decision.

5.3 Comparison with Linear Regression Methods

We also compared our temperature prediction model with a linear model, a common approach of temperature prediction in literature [20, 21]. We implemented a recently proposed model using linear regression [20] to compare with our methods.

This linear model has the form:

$$T_{cpu}(t+\Delta t) = T^{\infty}_{cpu}(t+\Delta t)+(T_{cpu}(t)-T^{\infty}_{cpu}(t+\Delta t))e^{-\frac{\Delta t}{R(t+\Delta t)C}},$$

with $T^{\infty}_{cpu}(t) = P_{cpu}(t)R(t) + T_{amb}(t)$, $R(t) = R_{cond} + \frac{1}{k_n V(t)^n}$ and $V(t) = \sqrt[3]{k_p P_{fan}(t)}$. C, R_{cond}, k_n and k_p are constants to be estimated. The model assumes a fixed cooling condition, making $P_{fan}(t)$ constant. Therefore, $R(t)$ is a constant and can be denoted as R. Hence, the model can be re-expressed as:

$$\begin{aligned} T_{cpu}(t+\Delta t) &= (1-e^{-\frac{\Delta t}{RC}})RP_{cpu}(t+\Delta t) \\ &\quad + e^{-\frac{\Delta t}{RC}}T_{cpu}(t) + (1-e^{-\frac{\Delta t}{RC}})T_{amb}(t+\Delta t). \quad \text{(i)} \end{aligned}$$

$T_{amb}(t)$ is also expressed as a linear combination of the CPU and the inlet temperatures as prescribed by Piatek et al's linear modeling proposal:

$$T_{amb}(t) = \lambda_1 T_{cpu}(t) + \lambda_2 T_{in}(t), \quad \text{(ii)}$$

where $\lambda_1, \lambda_2 > 0$. We substitute (ii) into (i), making λ_1 and λ_2 additional parameters to be estimated. Note that if we use $T_{amb}(t+\Delta t)$ in (i), we will have a $T_{in}(t+\Delta t)$ term, meaning that we need to also predict the inlet temperature. However, there was no method to predict inlet temperature described in [20]. Given our observation that inlet temperature has a slow rate of change with the run time of each

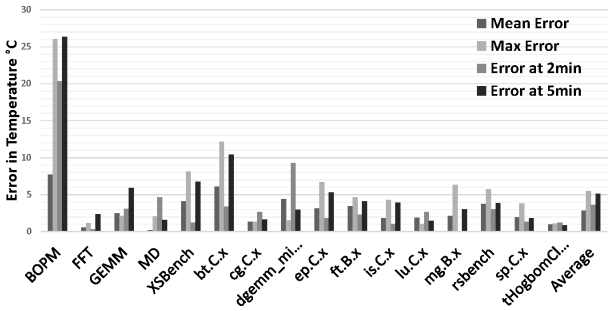


Fig. 9: Temperature prediction error of the decoupled method for Lasso linear regression.

application, we use $T_{in}(t)$ to replace $T_{in}(t+\Delta t)$. We still need T_{in} as a variable in the model, because it may be different across longer time windows. Therefore, we have:

$$\begin{aligned} &T_{cpu}(t+\Delta t) = \\ &\frac{(1-e^{-\frac{\Delta t}{RC}})RP_{cpu}(t+\Delta t) + e^{-\frac{\Delta t}{RC}}T_{cpu}(t) + (1-e^{-\frac{\Delta t}{RC}})\lambda_2 T_{in}(t)}{1-\lambda_1(1-e^{-\frac{\Delta t}{RC}})}. \end{aligned} \quad \text{(iii)}$$

The essential coefficients are, $P_{cpu}(t+\Delta t)$, $T_{cpu}(t)$, and $T_{in}(t)$. Given that Δt is fixed for Piatek et al.'s model, the coefficient space covered by (iii) is equivalent to

$$T_{cpu}(t+\Delta t) = \beta_1 T_{cpu}(t) + \beta_2 P_{cpu}(t+\Delta t) + \beta_3 T_{in}(t), \quad \text{(iv)}$$

where β_1, β_2 and β_3 have the same sign. If we assume that $0 < \lambda_1 < 1$, then the constraint can be further tightened to β_1, β_2, and β_3 being positive.

As the purpose of adding constraints usually is to prevent over-fitting, we also test a more general model with intercept added and relaxing the constraints in (iv):

$$T_{cpu}(t+\Delta t) = \beta_0 + \beta_1 T_{cpu}(t) + \beta_2 P_{cpu}(t+\Delta t) + \beta_3 T_{in}(t), \quad \text{(v)}$$

which is the model used for our comparison. We tested both models specified by (v) without constraints and (iv) with constraints, and have the following observations:

- (v) without constraints has better prediction accuracy over (iv) with constraints.
- β_1, β_2, and β_3 estimated in (v) without constraints are positive, automatically satisfying the constraints proposed in [20]. It means the regression in (v) aligns with the intuition from thermal model derived in [20].

Therefore, our comparison is presented using (v).

We used the same setup that was used in the previous sections for our methods. We compared the temperature prediction results of the methods in two scenarios: 1) online prediction, where we apply the method once to get the temperature prediction for one second later in time, 2) and static prediction, where we apply the method iteratively to obtain the temperature prediction into a far horizon, specifically into a time frame of five minutes. We show the results in Table 4. Each cell in the table shows the mean absolute error of the respective model (column) when applied to a particular application (row) in °C. We observe that the linear

	Linear Regression		Gaussian Process		MLP	
	Online	Static	Online	Static	Online	Static
BT	0.63	18.85	2.10	7.12	0.85	2.99
CG	0.54	15.05	0.61	2.05	0.59	2.66
EP	0.65	17.62	1.58	1.38	0.74	4.07
FT	0.51	15.94	1.05	4.64	0.64	2.15
LU	0.52	18.17	0.56	0.89	4.54	4.54
MG	0.56	15.88	0.72	2.63	0.62	2.92
SP	0.54	15.07	0.53	0.94	0.95	5.36
UA	0.58	15.58	0.86	2.19	0.75	1.46
mean	0.57	16.52	1.00	2.73	1.21	3.27

TABLE 4: Temperature prediction results of different methods.

method has accurate predictions when predicting temperature only for one short step forward. While this model has been reported to provide good performance when applied iteratively in simulated environments only (Fig 4 in [20]), it exhibits larger discrepancies to actual temperature when used for static predictions in a real system. The error is over 15 °C, meaning it does not have the capability in predicting temperature in the real system scenarios we aim to tackle. Our methods, both GP and MLP, have on par performance as Linear Regression in the short horizon of one second, single step setup. Furthermore, when prediction into a longer horizon is needed, our methods can still provide good prediction of the temperature in a time frame in the order of minutes.

When applying the method iteratively, the predicted temperature given by the linear model will converge to the predicted stable temperature $T_{cpu}(\infty)$, if the application is running at a stable state:

$$T_{cpu}(\infty) = \beta_0 + \beta_1 T_{cpu}(\infty) + \beta_2 P_{cpu}(\infty) + \beta_3 T_{fin}(\infty)$$

And,

$$T_{cpu}(\infty) = \frac{\beta_0 + \beta_2 P_{cpu}(\infty) + \beta_3 T_{fin}(\infty)}{1 - \beta_1}$$

We can observe the following two factors that cause the linear model to exhibit low performance in real world scenarios: 1) As the step size is small, the temperature does not have large change from one step to the next step. Therefore, the proposed linear model will regress β_1 close to 1. In our experiments, β_1 is fitted to 0.95. In the experiments in [20], β_1 is 0.98. Thus, the estimation of the temperature contribution of CPU power and inlet temperature must be very accurate to have a good prediction of stable temperatures. 2) In real world scenarios, temperature has other sources of influence. Thus, the simple linear relationship might not represent an accurate approximation to the true model. In addition, the sensor reading errors (granularity, delay, etc.) will also induce larger errors on the coefficient estimations.

5.4 Application Placement Guided by Temperature Prediction

We test our model in a job assignment scenario: Given two applications X and Y, determine which placement results in lower peak temperatures, (X → mic0, Y → mic1), or (Y → mic0, X → mic1). Since the two cards have the same architectural configuration, the applications will have the same performance on them. However, different placement can result in different thermal properties because of the asymmetric physical placement of the two cards. Our goal is to minimize the average temperature of the hotter card.

$$X_0, X_1 = \underset{\substack{X_0=X, X_1=Y \\ or X_0=Y, X_1=X}}{\arg\min} max\{mean(\boldsymbol{P}^{(temp)}_{0,X_0,X_1}), mean(\boldsymbol{P}^{(temp)}_{1,X_0,X_1})\} \quad (6)$$

In practice, however, we cannot get P before we actually run the application. We use the prediction $\hat{\boldsymbol{P}}$ as $\boldsymbol{P}$ in Equation 6. We let

$$\hat{T}_{XY} = \max\{mean(\hat{\boldsymbol{P}}^{(temp)}_{0,X,Y}), mean(\hat{\boldsymbol{P}}^{(temp)}_{1,X,Y})\},$$

$$\hat{T}_{YX} = \max\{mean(\hat{\boldsymbol{P}}^{(temp)}_{0,Y,X}), mean(\hat{\boldsymbol{P}}^{(temp)}_{1,Y,X})\},$$

$$T_{XY} = \max\{mean(\boldsymbol{P}^{(temp)}_{0,X,Y}), mean(\boldsymbol{P}^{(temp)}_{1,X,Y})\},$$

$$T_{YX} = \max\{mean(\boldsymbol{P}^{(temp)}_{0,Y,X}), mean(\boldsymbol{P}^{(temp)}_{1,Y,X})\}.$$

If $\hat{T}_{XY} - \hat{T}_{YX}$ and $T_{XY} - T_{YX}$ have the same sign, the model makes a prediction that can reduce the maximum average temperature of a specific application pair.

The runtime overhead of our sampling method is negligible because each prediction requires only one-time sampling of the initial value of the physical features.

For the placement that assigns X to mic0 and Y to mic1, we use the model f_0, which was trained without any knowledge of X to predict the temperature when running X on mic0, and we use the model f_1, which was trained without any knowledge of Y to predict the temperature when running Y on mic1. More specifically:

$$\begin{aligned} \boldsymbol{P}_{0,X_0,X_1} &\simeq \hat{\boldsymbol{P}}_{0,X_0,NONE} \\ \boldsymbol{P}_{1,X_0,X_1} &\simeq \hat{\boldsymbol{P}}_{1,NONE,X_1}. \end{aligned} \quad (7)$$

In Figure 10, Figure 11, and Figure 12, respectively, we plot $\hat{T}_{XY} - \hat{T}_{YX}$ versus $T_{XY} - T_{YX}$ for the decoupled method using the reduced Gaussian process model, MLP model, and Lasso linear regression model. Note that for some cases presented some points are completely overlapping, giving the visual perception of there being fewer points. Nevertheless, there are an equal number of points in each case.

From Figure 10, Figure 11, and Figure 12 we can see that $\hat{T}_{XY} - \hat{T}_{YX}$ has a positive correlation with $T_{XY} - T_{YX}$. Furthermore, for the Gaussian process–based prediction developed previously we found that $72.5\,\%$ of points fell into the first and third quadrants, which means our model made a correct decision and, hence, in $72.5\,\%$ of the cases identified the better placement among the two possible options. We define this as the success rate of our model. The decision of our model resulted in $2.1\,^{\circ}\mathrm{C}$ lower average temperature than the opposite placement on average. However, among the pairs with better scheduling opportunities, namely, $|T_{XY} - T_{YX}| \geq 3$, our model had a $86.67\,\%$ success rate.

Figure 10 shows prediction results for the reduced Gaussian process model. We found that $75\,\%$ of the points fall into the first and third quadrants; hence, in $75\,\%$ of the cases it identified the better placement among the two possible options. The decision of our model results in $4.3\,^{\circ}\mathrm{C}$ lower average temperature than the opposite placement on

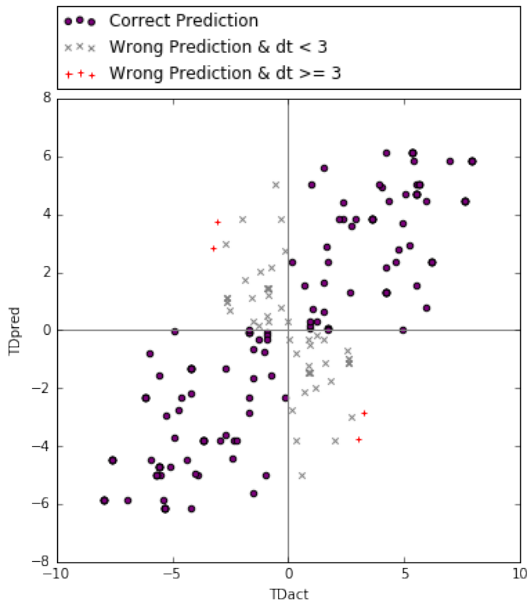


Fig. 10: Correlation between the actual and predicted thermal variation while using the decoupled method using reduced feature set Gaussian process prediction model.

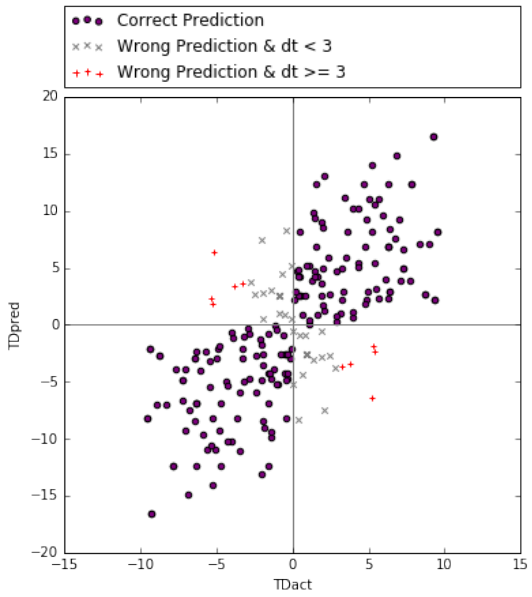


Fig. 11: Correlation between the actual and predicted thermal variation while using the decoupled method using MLP Prediction model.

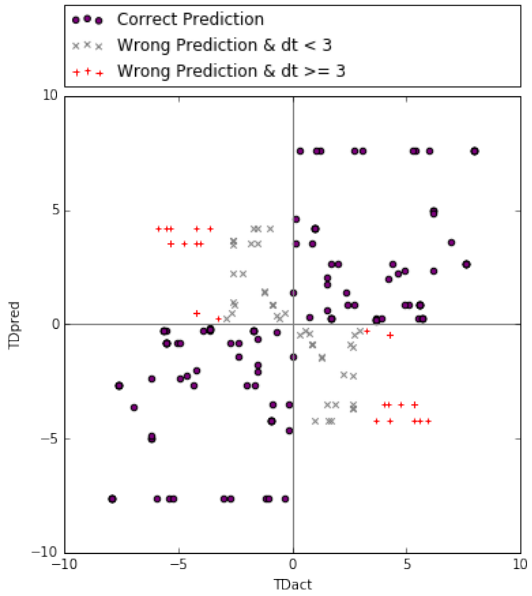


Fig. 12: Correlation between the actual and predicted thermal variation while using the decoupled method using the Lasso linear regression prediction model.

average. For the case $|T_{XY} - T_{YX}| \geq 3$, our model has a $92.9\,\%$ success rate. For the MLP-based prediction shown in Figure 11, we found that $82.5\,\%$ of the points fall into the first and third quadrants; hence, in $82.5\,\%$ of the cases identified the better placement among the two possible options. The decision of our model results in $4.4\,\%$ lower average temperature than the opposite placement on average. If we look only at $|T_{XY} - T_{YX}| \geq 3$, then our model has a $97\,\%$ success rate. For the Lasso linear regression based prediction shown in Figure 12, we found that $74.17\,\%$ of points fall into the first and third quadrants; hence, in $74.17\,\%$ of the cases it identified the better placement among the two possible options. The decision of our model results in $4.3\,^{\circ}\mathrm{C}$ lower average temperature than the opposite placement on average. If we look only at $|T_{XY} - T_{YX}| \geq 3$, then our model has a $94.83\,\%$ success rate.

Furthermore, if we look at all the pairs where our models make the wrong prediction, the average of $|T_{XY} - T_{YX}|$ is as low as $1.6\,^{\circ}\mathrm{C}$. Hence, these are task placements where either configuration would be equally efficient.

In summary, all the methods identify the best case when the temperature gains are maximum, and all choose the better placement scheme yielding an average of $11.9\,^{\circ}\mathrm{C}$ lower average temperature than the opposite placement scheme. In addition, we evaluated the optimal solution that could be obtained from an oracle scheduler. The optimal task schedules would result in $2.9\,^{\circ}\mathrm{C}$ lower average temperature than the opposite placement on average.

5.5 Scalability

We have also extended our methodology to a 16-node Haswell architecture cluster. We have preliminary results for a prediction module and its uses for task placement on this larger system. This new set of experiments focused on how a prediction scheme could lead to better energy efficiency and, particularly, help reduce the cooling energy spent by the cooling fans.

These experiments showed that certain task assignments have better energy efficiency than others and lead to execution at equivalent performance with reduced cooling requirements. Our machine learning approach in this case has been tuned to correlate task assignment with power consumption and cooling effort. For a 16-node system, we were able to train models and execute them in real time to guide task migration and achieve on average $17\,\%$ reduction in the cooling power.

Implementing our prediction algorithm at a larger scale has two sources of overheads. First, unlike the two-node Xeon Phi Co-processor system, dynamic migration on this

larger system takes significantly less time due to better single-core performance and smaller number of total cores. The time to perform cross-node migration is only around one minute. Considering the long running time of HPC applications, this overhead is manageable. Second, while doing one prediction over a specific task assignment scheme can be quick, the overhead adds up when we need to predict many different schemes to make one decision. We apply two simple techniques to reduce this overhead. The first technique is only randomly predicting $2\,\%$ of the total number of assignment schemes. This percentage is based on our observation that the best $1\,\%$ across all schemes already provide good power reduction. The second technique is to cache the prediction results. For different assignment schemes, it is possible to make predictions on same situations. For instance consider the scenario of application A running on node1-CPU1 and application B running on node1-CPU2. This prediction can be needed (and hence be reused) multiple times on any workload containing both application A and B. Thus caching them can save a lot of redundant computations.

6 Future Work

The accuracy of the prediction methods can be improved further. One item we are planning to improve is the guided selection of subset data for the training set. Currently we do a random selection. However, we can select the samples according to their representativeness, making the dataset cover more cases so that it can predict better.

The next major step is to apply the same method to other architectures, or a higher level, such as rack level. This is where our method's strength will shine: it is designed to be easily applied to other architectures with little knowledge and effort to build the model.

7 Conclusions

In this paper, we presented a novel framework to characterize thermal behavior of a HPC system. We have evaluated various prediction models empirically and selected Gaussian process as the base of our model. Our system predicts the average operating temperature of different applications to be scheduled on different parts of the system and selects the application mapping that reduces the maximum average temperature. We have discussed two models: the decoupled method and the coupled method. The decoupled method uses information on only the target platform, while the coupled method also considers information on the neighboring components. We also developed a reduced feature set Gaussian process and MLP based decoupled models. We demonstrate the methods developed we were able to reduce the error in prediction in Gaussian process model from $4.2\,^{\circ}\mathrm{C}$ to $2.9\,^{\circ}\mathrm{C}$ using the feature reduction. Further the newly developed models using neural network and lasso linear regression reduced errors to $2.9\,^{\circ}\mathrm{C}$, $3.8\,^{\circ}\mathrm{C}$ respectively. We also show that using the new models we were able to reduce the overhead of prediction to $0.22\,\mathrm{ms}$, $0.097\,\mathrm{ms}$ and $0.026\,\mathrm{ms}$ for reduced Gaussian process, neural network and lasso linear regression, respectively from $0.57\,\mathrm{ms}$ per prediction using the previously developed Gaussian process model.

When application mapping is considered, the decoupled reduced Gaussian process, neural network and lasso linear regression have $75\,\%$, $82.5\,\%$ and $74.17\,\%$ success rates, respectively compared to $72.5\,\%$ for the previously developed Gaussian process model. This results in a reduction of average temperature by up to $11.9\,^{\circ}\mathrm{C}$ ($4.3\,^{\circ}\mathrm{C}$, $4.4\,^{\circ}\mathrm{C}$ and $4.3\,^{\circ}\mathrm{C}$ on average, respectively compared to $2.1\,^{\circ}\mathrm{C}$ for the previously developed Gaussian process model.

Acknowledgments

This work has been partially funded by DOE grant DE-SC0012531 and by NSF grant CCF-1422489. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357. We thank Eric Pershey of the Argonne Leadership Computing Facility for providing us with Figure **??**(a). We also gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

References

[1] W.-c. Feng and K. Cameron, "The green500 list: Encouraging sustainable supercomputing," *Computer*, vol. 40, no. 12, pp. 50–55, 2007.

[2] R. F. Service, "Who will step up to exascale?" *Science*, vol. 339, no. 6117, pp. 264–266, 2013. [Online]. Available: http://science.sciencemag.org/content/339/6117/264

[3] K. Zhang, S. Ogrenci-Memik, G. Memik, K. Yoshii, R. Sankaran, and P. Beckman, "Minimizing Thermal Variation across System Components," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, May 2015, pp. 1139–1148.

[4] S. F. Crone, M. Hibon, and K. Nikolopoulos, "Advances in Forecasting with Neural Networks: Empirical Evidence from the NN3 Competition on Time Series Prediction," *International Journal of Forecasting*, vol. 27, no. 3, pp. 635–660, July 2011. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0169207011000616

[5] A. K. Jain, J. Mao, and K. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, no. 3, pp. 31–44, 1996.

[6] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996. [Online]. Available: http://www.jstor.org/stable/2346178

[7] R. Cochran and S. Reda, "Thermal Prediction and Adaptive Control Through Workload Phase Detection," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 1, 2013.

[8] O. Sarood, A. Gupta, and L. V. Kale, "Temperature Aware Load Balancing for Parallel Applications: Preliminary Work," in *IPDPS Workshops*, 2011.

[9] J. S. Lee, K. Skadron, and S. W. Chung, "Predictive Temperature-Aware DVFS," *IEEE Transactions on Computers*, vol. 59, no. 1, pp. 127–133, Jan 2010.

[10] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha, "HybDTM: A Coordinated Hardware-Software Approach for Dynamic Thermal Management," in *Proceedings of the 43rd annual Design Automation Conference*. ACM, 2006.

[11] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose, "Thermal-aware Task Scheduling at the System Software Level," in *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*. New York, NY, USA: ACM, 2007.

[12] O. Sarood and L. V. Kale, "A 'Cool' Load Balancer for Parallel Applications," in *Supercomputing*, 2011.

[13] L. Ramos and R. Bianchini, "C-Oracle: Predictive Thermal Management for Data Centers," in *Symposium on High-Performance Computer Architecture*, 2008.

[14] T. Mukherjee, Q. Tang, and C. Ziesman, "Software Architecture for Dynamic Thermal Management in Datacenters," in *Proceedings of COMSWARE*, 2007.

[15] J. Moore, J. Chase, P. Ranganathan, and R. Sharma, "Making Scheduling "Cool": Temperature-aware Workload Placement in

Data Centers," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, 2005.
[16] L. Wang, G. von Laszewski, J. Dayal, X. He, A. Younge, and T. Furlani, "Towards Thermal Aware Workload Scheduling in a Data Center," in *10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN)*, Dec 2009.
[17] B. Acun and L. V. Kale, "Mitigating processor variation through dynamic load balancing."
[18] B. Salami, H. Noori, F. Mehdipour, and M. Baharani, "Physical-Aware Predictive Dynamic Thermal Management of Multi-Core Processors," *Journal of Parallel and Distributed Computing*, 2016.
[19] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic Thermal Management through Task Scheduling," in *Performance Analysis of Systems and software, 2008. ISPASS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 191–201.
[20] W. Piatek, A. Oleksiak, and G. Da Costa, "Energy and Thermal Models for Simulation of Workload and Resource Management in Computing Systems," *Simulation Modelling Practice and Theory*, vol. 58, pp. 40–54, 2015.
[21] J. Moore, J. Chase, and P. Ranganathan, "Weatherman: Automated, Online and Predictive Thermal Mapping and Management for Data Centers," in *IEEE International Conference on Autonomic Computing*, June 2006.
[22] M. T. Chaudhry, T. C. Ling, A. Manzoor, S. A. Hussain, and J. Kim, "Thermal-Aware Scheduling in Green Data Centers," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, p. 39, 2015.
[23] I. Guyon and A. Elisseeff, "An Introduction to Variable and Feature Selection," *The Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=944968
[24] M. A. Hall, "Correlation-Based Feature Selection for Machine Learning," Ph.D. dissertation, The University of Waikato, 1999.
[25] S. Coghlan, K. Kumaran, R. M. Loy, P. Messina, V. Morozov, J. C. Osborn, S. Parker, K. Riley, N. A. Romero, and T. J. Williams, "Argonne Applications for the IBM Blue Gene/Q, Mira," *IBM Journal of Research and Development*, vol. 57, no. 1/2, pp. 12–1, 2013.
[26] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.
[27] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.
[28] M. A. Hall, "Correlation-based feature subset selection for machine learning," Ph.D. dissertation, University of Waikato, Hamilton, New Zealand, 1998.
[29] G. J. Williams, *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*, ser. Use R! Springer, 2011.
[30] 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.
[31] I. Jolliffe, *Principal Component Analysis*. John Wiley & Sons, Ltd, 2014. [Online]. Available: http://dx.doi.org/10.1002/9781118445112.stat06472
[32] A. C. Davison, *Statistical Models*. Cambridge University Press, 2008.
[33] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer, 2009.
[34] F. Rosenblatt, "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms," DTIC Document, Tech. Rep., 1961.
[35] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber, "PyBrain," *Journal of Machine Learning Research*, vol. 11, pp. 743–746, 2010.
[36] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," DTIC Document, Tech. Rep., 1985.
[37] H. B. Jang, J. Choi, I. Yoon, S.-S. Lim, S. Shin, N. Chang, and S. W. Chung, "Exploiting application/system-dependent ambient temperature for accurate microarchitectural simulation," *IEEE Transactions on Computers*, vol. 62, no. 4, pp. 705–715, 2013.

Kaicheng Zhang Kaicheng Zhang is a PhD student at the Electrical Engineering and Computer Science Department of Northwestern University. His thesis work focuses on architectural and system-level solutions to thermal management problems in high performance computing systems.

Akhil Guliani Akhil Guliani has completed his MS Degree in Computer Science at Northwestern University. He has interests in computer systems, high performance computing, resource virtualization, embedded systems and applications of machine learning in the control systems. He has worked for two years as a project engineer at GAIL India Ltd. He holds his B.E in Instrumentation and Control Engineering from Delhi University, India.

Kazutomo Yoshii Kazutomo Yoshii is a senior software development specialist at the Argonne National Laboratory. He received his M.S. in computer science from Toyohashi University of Technology in Japan in 1994. After several year's experience in industry, he joined Argonne National Laboratory in 2004. He now focuses on operating system and system software for next-generation supercomputers. He recently started exploring reconfigurable computing in preparation for the post-Moore era.

Rajesh Sankaran Rajesh Sankaran is a computer scientist in the Mathematics and Computer Science Division at Argonne National Laboratory. Dr. Sankaran graduated with a Ph.D. in electrical and computing engineering from Louisiana State University in 2011. He received a B.E. in electronics and communication engineering from Periar University, India in 2004.

Seda Ogrenci-Memik Seda Ogrenci-Memik received her BS degree in Electrical and Electronic Engineering from Boazii University, Istanbul, Turkey and her PhD in Computer Science from University of California, Los Angeles. She is currently a Professor at the Electrical Engineering and Computer Science Department of Northwestern University. Her research interests include embedded and reconfigurable computing, high level synthesis, thermal-aware design automation, and thermal management for microprocessor systems.

Gokhan Memik Gokhan Memik is a Professor at the Electrical Engineering and Computer Science Department of Northwestern University. He received the B.S. degree in Computer Engineering in 1998 from Bogazici University and PhD in Electrical Engineering from University of California at Los Angeles (UCLA) in 2003. His research area is computer architecture. He is the author of 2 book chapters and over 120 refereed journal/conference publications.

Pete Beckman Pete Beckman is the co-director of the Northwestern-Argonne Institute for Science and Engineering. From 2008-2010 he was the director of the Argonne Leadership Computing Facility, where he led the Argonne team working with IBM on the design of Mira, a 10 petaflop Blue Gene/Q. Pete also has experience in industry. After working at Los Alamos National Laboratory on extreme-scale software for several years, he founded a Turbolinux-sponsored research laboratory in 2000 that developed the world's first dynamic provisioning system for cloud computing and HPC clusters. Dr Beckman has a Ph.D. in computer science from Indiana University (1993) and a BA in Computer Science, Physics, and Math from Anderson University (1985).