Minimizing Thermal Variation Across System Components

Kaicheng Zhang, Seda Ogrenci-Memik, Gokhan Memik
Department of EECS
Northwestern University
{*kaicheng.zhang, seda, memik*}*@eecs.northwestern.edu*

Kazutomo Yoshii, Rajesh Sankaran, Pete Beckman
Mathematics and Computer Science Division
Argonne National Laboratory
{*kazutomo, rajesh, beckman*}*@mcs.anl.gov*

***Abstract*—Thermal overheating is a serious concern in modern supercomputing systems. Elevated temperature levels reduce the reliability and the lifetime of the underlying hardware and increase their power consumption. Previous studies on mitigating thermal hotspots at the hardware and run-time system levels have typically used approaches that trade off performance for reduced operating temperatures.**

In this paper, we first show that in a large-scale system, physical attributes cause an uneven temperature distribution. We then develop a model to characterize the thermal behavior of a complex system using various machine learning methods. We propose to improve application placement by incorporating thermal awareness into the decision-making process. Specifically, our system predicts the thermal condition of the system based on application mapping and uses these predictions to mitigate thermal hotspots without any performance loss. We provide two versions of our prediction mechanism. On a two-node configuration, these models achieve 72.5% and 78.8% success rates in their predictions, respectively. In other words, the scheduling decisions of our models result in a task placement that has a lower maximum average temperature. Overall, the more aggressive scheme reduces the average peak temperature by up to 11.9°C (2.3°C on average) without any performance degradation.

I. Introduction

Thermal implications are becoming increasingly influential in determining the cost of operating a high-performance computing (HPC) system. Furthermore, thermal effects are one of the primary factors limiting achievable peak performance. All major processor manufacturers correlate the maximum expected performance with the thermal design point (TDP). This metric represents the maximum amount of heat and power that can be sustained by the system during reasonably long execution intervals representative of typical workloads, while only short-lived (on the order of microseconds) crossing over this threshold may be allowed. In addition, increasing the operating temperatures reduces the long-term reliability of CMOS-based ICs exponentially [1].

In addition to the general relevance of thermal effects on performance and cost optimizations in all HPC systems, emerging energy-efficient cooling paradigms bring new challenges. Commercial data centers and supercomputing centers are under immense pressure to reduce their cooling cost and carbon footprint. A developing response to this pressure is a new paradigm allowing warmer inlet water temperatures for the liquid cooling commonly used in these systems. Several HPC systems—including SuperMUC, one of the world's top ten most powerful supercomputers—already operate under raised inlet coolant temperatures. These systems aim to aggressively exploit the thermal headroom (i.e., guard band) in the server chips. While tightly controlled systems can do so with smaller risks, it is unclear how far the envelope can be pushed unless the system is equipped with better awareness of the energy and thermal optimization objectives. All the system-level manifestations of thermal events—for example, imbalance in node performances, increased failure rates, and unpredictability—will only be exacerbated under this new cooling paradigm. Therefore, system management modules in HPC systems need to be more tightly involved in thermal management and be equipped with thermal-aware policies.

A complexity that arises in thermal management in large-scale systems is the effect of physical properties on the thermal behavior. For example, two identical CPUs on different parts of the system will exhibit different thermal responses. Hence, in this paper, we first demonstrate that physical properties that affect thermal response need to be incorporated into system-level decision making in an explicit manner and with minimal overhead. Specifically, our study indicates that a significant variation in thermal responses exists among functionally equivalent subsystem components (e.g., nodes, racks), which the system might otherwise view as identical under the same workload conditions. In turn, this varying thermal response can cause these seemingly identical subcomponents to perform nonuniformly, contrary to the abstract assumption of the system about these hardware components. Our preliminary experiments on a two-node HPC system (Intel Xeon Phi coprocessors) in fact revealed that a difference of over 20°C can occur between these two identical nodes although they are running the same application. Such imbalance can cause thermal throttling, which significantly degrades performance: our experiments reveal 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 in our target benchmarks.

To address the mitigation of thermal variation and thermal hotspots, we first present a methodology for performing thermal characterization of HPC systems through a combination of empirical data analysis and machine learning models. Our model is especially effective in discovering nonuniformities in thermal responses across a system. We also demonstrate how such a lightweight yet accurate model can be used to assess the thermal quality of different task-scheduling schemes and guide an optimal choice. Our thermal evaluation of a representative state-of-the-art HPC system indicates that different nodes within the system exhibit widely different thermal responses to the same workload and cooling conditions. This result shows that thermal modeling of a generic computing node cannot simply be replicated and combined to construct the thermal model of a large system. Hence, we have sought to develop a method to model the inherent variation within a target system as accurately as possible.

Another important goal of our study has been to develop a lightweight model that can be derived quickly from available system-level parameters. Leveraging machine learning methods, we have developed a model for predicting system temperatures. We demonstrate the effectiveness of this model for task scheduling in our target system, which has two Intel Xeon Phi cards. 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 two 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 72.5% and 78.8% for each version, respectively. We show that these scheduling schemes guided by our model can decrease the peak temperature of the system by as much as 11.9°C (2.3°C on average).

The rest of this paper is organized as follows. Section II gives an overview of related work. In Section III, we present our motivational experiments. We present our thermal characterization methodology and the resulting model in Section IV. Section V presents our experimental results. We discuss possible future directions in Section VI and summarize our main findings in Section VII.

II. Related Work

Several prior studies are concerned with guiding core-level DVFS schemes with direct physical sensor feedback or via prediction models [2, 3, 4, 5]. Other studies focused on system management tasks such as task migration, load balancing, etc. [6, 3, 7, 8, 9, 10, 11]. Choi et al. investigated thermal-aware temporal and spatial mitigation schemes [6]. 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 Linux 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.

Ramos and Bianchini [8] proposed a model which 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 thermo-mechanical properties.

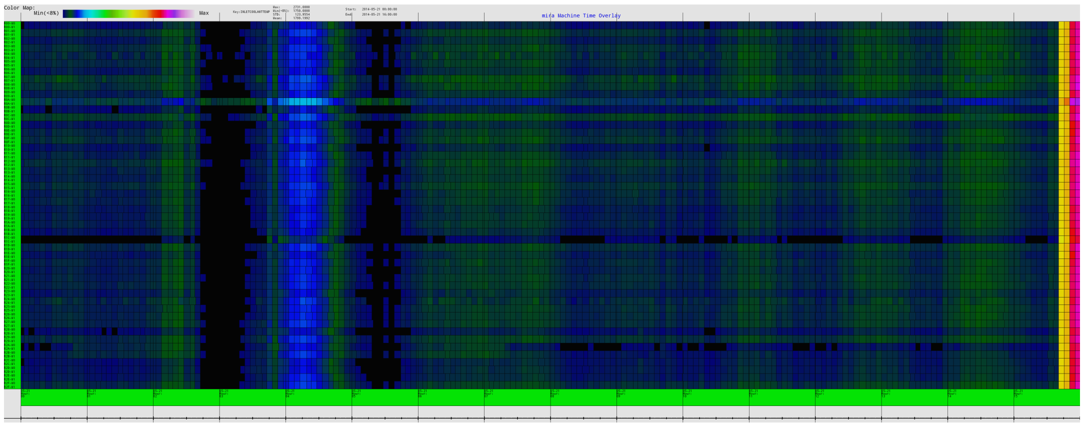
Moore et al. [12] proposed a framework to build a thermal model for a datacenter using readings taken from external temperature sensors, server instrumentation, and computer room air conditioning units. Their framework leverages a neural network based method. Ultimately, this model is used to provision 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 only trained to learn the system, unlike our model, which is capable of inferring both the system and the actual applications.

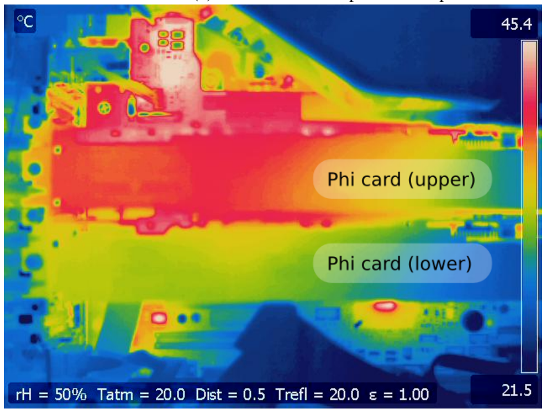
To the best of our knowledge, our study is the first to use a Gaussian process to create a temperature model for a computer system. Also, most previous works that developed prediction methods do not target a true HPC system or HPC workloads. Those that study thermal-aware management for HPC systems, on the other hand, utilize only limited physical thermal sensor feedback or require detailed knowledge on the thermo-mechanical properties of the system. Our framework provides a novel application of the Gaussian process based machine learning method, which solely leverages system status that is accessible to the operating system and no physical or domain-specific information about the system.

Furthermore, our thermal characterization framework extends beyond the level of processor cores and is capable of characterizing higher-level system components, such as nodes. On the other hand, most previous works focus solely on predicting and mitigating within-core and across-core thermal variation.

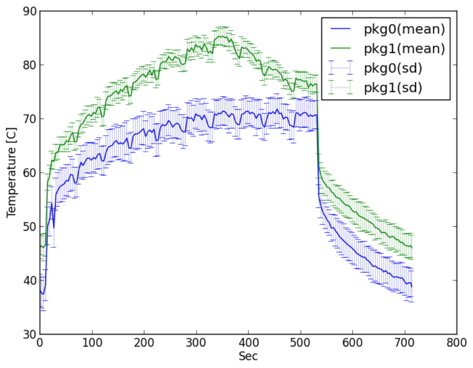
III. Motivation: Thermal Analysis of a HPC System

Our work has been motivated mainly 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 supercomputer [13], (2) a thermal map from a two-node system based on Intel Xeon Phi implemented as PCIe cards, where each card contains one processor with 61 cores and 4 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 from a third party [13]. We collected the data for the other two systems ourselves, using microbenchmarks. Figure 1(a) illustrates the inlet coolant temperature across


(a) Inlet coolant temperature map on MIRA supercomputer with water-cooled BG/Q nodes.



(b) IR camera image of two Intel Phi Coprocessor cards



(c) Temperature variation on Sandy Bridge

Figure 1: Temperature variation in different HPC systems: Figure 1a shows the inlet coolant temperature variation on Mira supercomputer. Each cell represents a machine, each row is a rack of the cluster, and the colors represent the temperature of inlet coolant. Lighter color indicates higher temperature. Variation and presence of hotspots are clearly visible.Figure 1b shows the temperature variation of two Intel Xeon Phi Coprocessor cards running the same workloads.Despite executing the same workloads, thermal variation across cards is significant. Figure 1c shows the temperature variation on two Intel Sandy Bridge processor package configurations, both within and across packages.

nodes in the Mira supercomputer. Lighter color indicates higher temperature. Figure 1(b) is an infrared image of the two Phi cards both executing the same FPU microbenchmark workload. Figure 1(c) plots the thermal behavior of all cores in the system, eight cores in package 0 and another eight in package 1. The figure also depicts the average temperature and standard deviation among all cores within package 0 and package 1, respectively.

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 largely present in different systems, as well as at 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°C. Clearly, some mappings incur significantly more thermal stress than others.

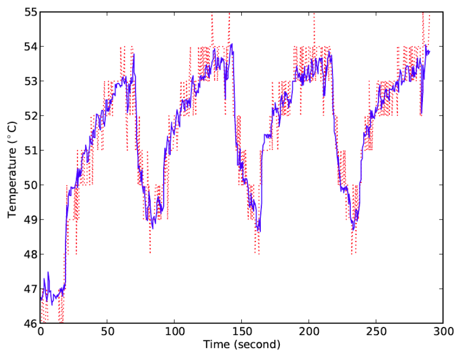
These observations motivated us to design tools for making thermal-aware decisions, mitigating the problem of thermal variation and reducing the hotspots in the system. Specifically, we present here a new framework to characterize the thermal behavior of a HPC system that is especially geared toward capturing thermal hotspots at system-level components. Then, we use this thermal model to perform task scheduling on a HPC system with multiple nodes of the same configuration. Our aim is to identify the best assignment of tasks to nodes such that the average temperature of the hottest system node is minimized.

IV. Model for Predicting System Temperature

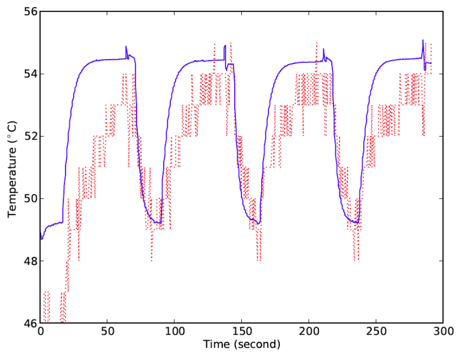
We have developed a temperature prediction model for a HPC system. Our model is designed to be reflective of the physical and architecture-driven sources of variation.

Our methodology comprises five steps:

1) For a specific machine in the system (e.g., a particular node in the cluster), we develop a thermal prediction model. Specifically, 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. We also collect the thermal response of these benchmarks on this machine 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. These are kept as logs 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) and application characteristics of this specific target workload preprofiled in Step 3 to predict the operating temperature.
5) Using the predicted operating temperatures, we can compare alternative task assignments with each other, and the system software will be provided with the suggestion for an assignment that is expected to result in lower average temperature for the hottest node.



(a) Online prediction using our model



(b) Static prediction using our model

Figure 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).

Our 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.). Ultimately the output of our model is the thermal response of the system. Our 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 our 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) are plotted along with the predicted response generated by our model (solid line). We can see that the model has high accuracy (less than 1°C difference on average) when used online. Our 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 mainly focused on in this work in order to aid static task assignment decisions. In this usage model, our prediction is not intended to calculate a spontaneous thermal response with absolute accuracy. The most important role for our model is to predict steady-state behavior as accurately as possible. Also, the model should be capable of tracking significant fluctuations along a longer time scale. The output of our model when used statically is illustrated in Figure 2b. As one can see, our model successfully captures long-term fluctuations and the steady-state behavior. In this study, we chose to first focus on evaluating the effectiveness of a universal model in aiding a static task scheduler. Dynamic scheduling aided by our model would be feasible as far as the accuracy of the temperature prediction goes. However, the effectiveness of the resulting dynamic scheduling, including migration overheads and the like, requires a further careful study. We plan to investigate this avenue in our future work.

A. Construction of the Modeling Framework

Construction of our model entails two main activities: (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, we collect a set of features at a time t from variety of sources, including hardware performance counters, temperature sensors, and kernel counters. Some of these features are highly correlated with application characteristics. These features do not change significantly when running the application on different nodes of the system. We denote these features as *application features*, and we denote the values of these features at time t as a vector $\boldsymbol{A}(t)$.

The remaining features are more strongly correlated with a node's physical condition. Even while running the same application, these features can vary drastically across nodes depending on each node's cooling conditions, location, and so forth. For example, temperature sensor readings are in this category. We refer to these features as *physical features* and denote them as a vector $\boldsymbol{P}(t)$. In Section V, we provide a full list of these features.

Our framework uses a set of representative benchmark applications to characterize a node and obtain a machine-specific model for that node. For the entire benchmark set, application and physical features are collected for a sufficiently long execution period. Using the combined application and physical features, we then train a machine learning model to represent the thermal behavior of this node. This model reflects a node's thermal response completely independent of any thermal coupling with other nodes in the system. We refer to this model as a *decoupled model*. As we decoupled thermal interference between the nodes, this modeling method suits large scale system best.

We leverage a machine learning method to generate this model. We will further elaborate on the derivation of this model in Sections IV-B and IV-C. Basically, this model will be embodied in the form of a function f_j generated for given node j, such that

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

In addition to generating the prediction model the framework also involves extraction of representative features from the actual workload set of the HPC system. Each target application is run on the node, and its application 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.

The resulting temperature prediction model is then used as follows. The task scheduler would be considering assigning an application to a specific node in the system at some point in time. 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 preprofiled data and at each step makes a temperature prediction. In this way, the model generates the expected thermal response over a period of time.

Note that a node's thermal response model is strictly specific to the node's own physical properties. As far as applications are concerned, the model uses general performance indicators from a representative benchmark set. The actual workloads that will be deployed on the node are not part of this data. The same universal model is used for any and all applications that may be deployed on this node henceforth.

In the abovementioned usage mode, just as with the training of the model, the prediction is performed without requiring or assuming any knowledge about the thermal or performance state of any other node in the system. As an alternative, both the training stage of the machine-specific models and the actual predictions can be provided with additional information on the activity of nearby nodes to reflect potential thermal coupling. We refer to this prediction method as the *coupled method*.

Training for decoupled models and execution of decoupled prediction is highly scalable. On the other hand, introducing coupling information will require collection of features from one application or node while mapping another paired application onto all nearby nodes in different combinations. Leaning toward decoupled methods in general is a necessary trade-off when we attempt to model thermal responses within a large-scale system, since scaling issues will dominate. There is also another intuitive reason that the decoupled modeling and prediction methods will fare well in practice, especially for guiding task scheduling. Intuitively, a partial ordering should exist across applications as well as system nodes in terms of relative expected location of a thermal hotspot. Hence, even if a strong thermal coupling exists between system nodes, being able to understand the ordering of applications and the susceptibility of system nodes should be sufficient to provide thermal-aware scheduling mechanisms. In fact, our results show that the accuracy of our decoupled method is minimally improved by considering coupling between system nodes.

Figure 2b illustrates how this approach works. As discussed earlier, the model performs static predictions. In the early phases, the prediction's absolute values do not necessarily match the real thermal state closely. Nevertheless, we observe that the predictor is successful in identifying the trends (i.e., the instances of dramatic jumps in temperature). It also is accurate in its prediction of the peaks and steady-state thermal conditions.

B. Evaluation of Alternative Modeling Approaches

The effectiveness of our framework largely depends on how well the function f described in Equation (1) can capture the thermal characteristics of the system. Note that we make no assumption on any detailed knowledge about the underlying system. For instance, our models are unaware of placement of nodes (e.g., which card is on top, which card is near the cooler air flow) and the geometry of the system. Similarly, our model does not make use of any domain-specific knowledge pertaining thermal modeling. In other words, our model has no knowledge of the thermal transfer properties of the materials involved. The model operates as a simple mapping function between features that are accessible to an operating system's kernel and the expected temperature at a node. This makes machine learning methods a natural candidate for our purposes. To evaluate a large number of available machine learning methods efficiently, we used WEKA [14] to explore various regression methods.

Our 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 have tested these methods to predict as far as 25 seconds into the future. From Figure 3, we observe that, as expected, 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 the shorter prediction windows. Overall, the Gaussian process, among all the methods, has the best prediction accuracy until the prediction time window reaches 25 seconds.

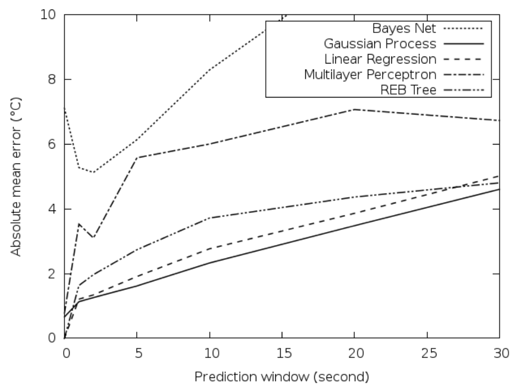


Figure 3: Performance of different machine learning methods when predicting future temperatures.

C. Prediction Model Using a Gaussian Process

As a result of our investigation described in Section IV-B we chose the Gaussian process [15] method to generate an approximation of f. A Gaussian process is a stochastic process of a set of random variables $(X_1, X_2, \ldots)$, where any subset of these variables constitute 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 pre-computed and reused each time a prediction is made. Hence, the matrix inversion step of this pre-computation 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.

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

D. Complexity Analysis and Runtime Overhead

The main contributor to the computational complexity of a Gaussian process is the matrix inversion at the precomputing

Model #	7120X
# of cores	61
Frequency	1238094 kHz
Last Level Cache Size	30.5 MB
Memory Size	15872 MB

Table I: Intel Xeon Phi coprocessor configuration.

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 $N = N_{\text{max}}$.

When 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 are using 30 different sources, and this incurs 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).

V. Experimental Results

We setup our test environment on 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 double-precision performance of a 61-core Phi coprocessor is 1.2 teraflops. Each coprocessor runs its independent operating system, and communicates with the host system through the PCIe interface. There is a System Management Controller on the same board along with the coprocessor, and it monitors a variety of on-board sensors. Since our system is comprised of two nodes (cards), we denote the bottom card as “mic0” and the top card as “mic1”. The configuration of the Phi coprocessor is shown in detail in Table I.

Table II shows the full list of benchmarks that were used to construct our thermal model and their configurations. We also developed a kernel module to collect all available system features. The kernel module performs the sampling at a fixed interval. The sampling has a runtime overhead of 20 ms; to balance this cost, we have chosen a sampling period of 500 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 III presents the list of the features we collect and their classification. The die temperature feature is the one that our model ultimately predicts.

app	data size, parameter	description
from Argonne National Laboratory		
XSBench	default	compute cross sections using the continuous energy format
RSBench	default	compute cross sections using the multi-pole representation format
from the NAS Parallel Benchmark		
BT	C	Block Tri-diagonal 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	Multi-Grid on a sequence of meshes
SP	C	Scalar Penta-diagonal 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 II: Applications used for our experiments.

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 III: List of features collected from the system.

As we include temperature sensors into our model, the model can be trained to capture the processor temperature as well as the ambient temperature, which also plays an important role to thermal models [16].

We run each application for five minutes. If the application finishes in under five minutes, we restart it. If the application runs longer than five minutes, we terminate it. We have confirmed that all applications perform a major portion of their main body of computation within this duration. In our 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.

A. *Model Selection*

We build our model based on a Gaussian process as described in Section IV-C. The core part of selecting a good Gaussian process model is the selection of kernel functions. Intuitively, the kernel function characterizes the inherent property of a problem. We have tested different types of kernel functions, and finally chose the cubic correlation function:

$$k(x_1, x_2) = \prod_{i=1}^{n} max(0, 1 - 3(\theta(x_1^{(i)} - x_2^{(i)}))^2 + 2(\theta(x_1^{(i)} - x_2^{(i)}))^3) \tag{6}$$

The θ we chose is 0.01. For our experiments, this value resulted in a good prediction accuracy.

When training a model, we randomly select a subset of 500 samples from all the available samples to limit the computation costs of the model. We limit the total number of samples used for training to reduce both time and space complexity of the model as described in Section IV-D. The number $N_{\text{max}} = 500$ is selected, because it provides a good trade off between prediction accuracy and model complexity. *We must 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.*

We use symbol $\boldsymbol{A}_{i,X,Y}(j)$ and $\boldsymbol{P}_{i,X,Y}(j)$ 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 only pick $\boldsymbol{A}_{i,X,NONE}$ 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, and is discussed in the following sections.

B. *Predicting Application Temperature*

We first test the decoupled method: 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. We report the individual peak temperature error and average temperature errors of our model in Figure 4. 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.

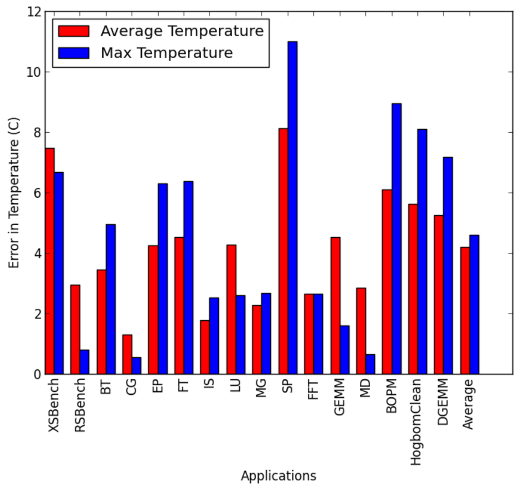


Figure 4: Temperature prediction error of the decoupled method.

We have to highlight a few properties of this experimental 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.

We observe that our model can predict the temperature very accurately for most of the applications. The average error is 4.2°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, when it comes to task placement problems, some bias error can be canceled out and lead to correct scheduling decision.

C. *Application Placement Guided by Temperature Prediction*

We test our model in a job assignment scenario: given two applications X and Y, which placement results in lower peak temperatures, (X $\rightarrow$ mic0, Y $\rightarrow$ mic1), or (Y $\rightarrow$ mic0, X $\rightarrow$ 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 due to the asymmetric physical placement of the two cards. Our goal is to minimize the average temperature of the hotter card:

$$X_0, X_1 = \underset{\substack{X_0=X, X_1=Y \\ or X_0=Y, X_1=X}}{\arg\min} \; max\{mean(\boldsymbol{P}^{(temp)}_{0,X_0,X_1}), mean(\boldsymbol{P}^{(temp)}_{1,X_0,X_1})\} \tag{7}$$

However, in practice, we cannot get P before we actually run the application. We use the prediction $\hat{\boldsymbol{P}}$ as $\boldsymbol{P}$ in Equation 7. 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 only requires 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 use the model f_1, which was trained without any knowledge of Y to predict the temperature when running Y on mic1. More specifically:

$$\begin{aligned} \boldsymbol{P}_{0,X_0,X_1} &\simeq \hat{\boldsymbol{P}}_{0,X_0,NONE} \\ \boldsymbol{P}_{1,X_0,X_1} &\simeq \hat{\boldsymbol{P}}_{1,NONE,X_1} \end{aligned} \tag{8}$$

We plot $\hat{T}_{XY} - \hat{T}_{YX}$ versus $T_{XY} - T_{YX}$ using the decoupled method in Figure 5. From Figure 5, we can see that $\hat{T}_{XY} - \hat{T}_{YX}$ has a positive correlation with $T_{XY} - T_{YX}$. Furthermore, 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 results in 2.1°C lower average temperature than the opposite placement on average. However, if we look at the pairs that have better scheduling opportunities, i.e. $|T_{XY} - T_{YX}| \geq 3$, our model has a 86.67% success rate. Furthermore, if we look at all the pairs where our model makes the wrong prediction, the average of $|T_{XY} - T_{YX}|$ is as low as 1.6°C. Hence, these are task placements where either configuration would be equally efficient.

The advantage of the decoupled method is its scalability. The model training and testing of each individual node is independent of other nodes.

We also experimented with the coupled method. In this method, we train a general model taking both mic0 and mic1 into consideration. Specifically, the model uses features from both mic0 and mic1 as inputs, and outputs the predicted state of mic0 and mic1. For an ordered pair of applications (X, Y), we use all the samples $(A_{0,X,Y}, A_{1,X,Y})$ and $(P_{0,X,Y}, P_{1,X,Y})$ to obtain a model f, where $X, Y \in \{\text{All of our applications}\} \setminus \{X, Y\}$. Then, we compute $\hat{\boldsymbol{P}}$ by:

$$\begin{aligned} (\hat{\boldsymbol{P}}_{0,X,Y}(i), \hat{\boldsymbol{P}}_{1,X,Y}(i)) &= f((\boldsymbol{X}_{0,X,Y}(i), \boldsymbol{X}_{1,X,Y}(i)), \\ &\quad (\boldsymbol{X}_{0,X,Y}(i-1), \boldsymbol{X}_{1,X,Y}(i-1)), \\ &\quad (\hat{\boldsymbol{P}}_{0,X,Y}(i-1), \hat{\boldsymbol{P}}_{1,X,Y}(i-1))) \\ (\hat{\boldsymbol{P}}_{0,X,Y}(1), \hat{\boldsymbol{P}}_{1,X,Y}(1)) &= (\boldsymbol{P}_{0,X,Y}(1), \boldsymbol{P}_{1,X,Y}(1)) \end{aligned} \tag{9}$$

We plot $\hat{T}_{XY} - \hat{T}_{YX}$ versus $T_{XY} - T_{YX}$ in Figure 6. From Figure 6, we can see that $\hat{T}_{XY} - \hat{T}_{YX}$ has a positive correlation with $T_{XY} - T_{YX}$. Furthermore, we found that 78.33% of points fell into the first and third quadrants, which means our model made a correct decision, and the overall success rate of our model is 78.33%. The decision of our model results in 2.3°C lower average temperature than the opposite placement on average. However, if we look at the pairs that have better scheduling opportunities, our model's success rate is 88.89%. Furthermore, if we look at all the pairs where our model makes the wrong prediction, the average of temperature difference is as low as 1.3°C.

As the coupled method utilizes more information, it has a slightly better performance than the decoupled method.

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

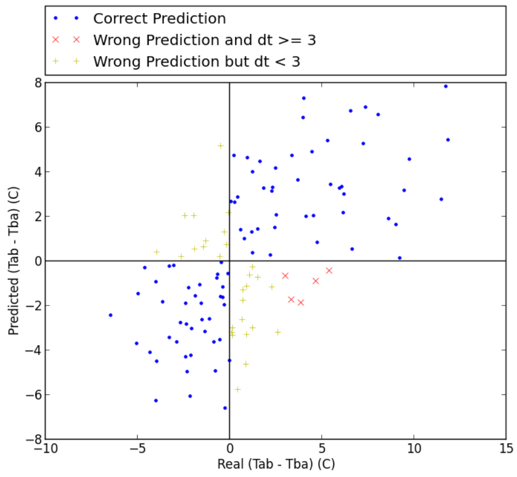


Figure 5: Correlation between the actual and predicted thermal variation while using the decoupled method

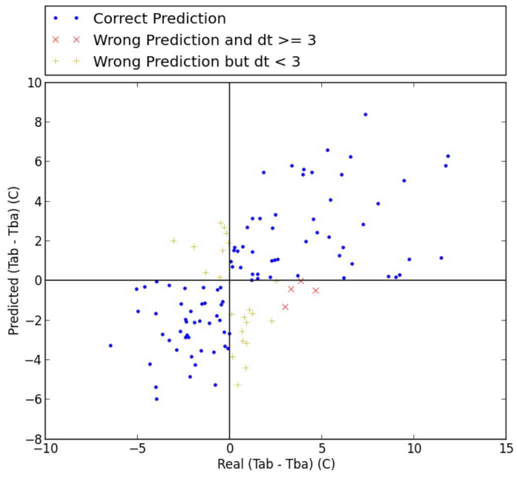


Figure 6: Correlation between the actual and predicted thermal variation while using the coupled method

VI. 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.

VII. 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. When application mapping is considered, the decoupled and coupled methods have 72.5% and 78.8% success rates, respectively. This results in a reduction of average temperature by up to 11.9°C (2.3°C and 2.1°C on average for the coupled and decoupled methods, respectively).

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 Argonne Leadership Computing Facility's Eric Pershey for providing us with Figure 1(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] "Uprating Semiconductors for High-Temperature Applications," Micron T., Tech. Rep., 2004.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.