Boreas: A Cost-Effective Mitigation Method for Advanced Hotspots using Machine Learning and Hardware Telemetry

Maziar Amiraski*#, David Werner*#, Alexander Hankin§¶, Julien Sebot†, Kaushik Vaidyanathan‡, Mark Hempstead*

* Tufts University, Medford, MA, USA
† Intel Corp., Hillsboro, OR, USA
‡ Google Inc., Sunnyvale, CA, USA
§ Harvard University, Cambridge, MA, USA
¶ Intel Labs, Hillsboro, OR, USA

{maziar.mehdizadehamiraski, david.werner}@tufts.edu

***Abstract*—Managing advanced hotspots on modern microprocessors is a critical and worsening issue, affecting performance, product reliability, and device lifetime. Many thermal management techniques focus primarily on remaining below a critical temperature, and while they are successful to that end, they come with a significant performance cost. This cost stems from the large temperature guardbands that are needed to ensure device safety and correct IC operation. These guardbands must be sized to account for multiple factors including 1) control-loop latency, 2) thermal sensor delay, 3) thermal gradients inside timing paths that could result in timing violations, and 4) the instantaneous temperature delta between a temperature sensor and the true peak temperature on the IC.**

This work demonstrates the need for novel hotspot avoidance techniques that can react quickly and simultaneously account for each of these concerns in order to safely maximize performance. Recently introduced hotspot metrics—*Hotspot-Severity* and Maximum Local Temperature Difference (MLTD)—are used in order to allow model designers to have a single optimization target that accounts for each of these thermal concerns simultaneously. We present Boreas, a novel hotspot mitigation technique that uses a Machine Learning model implemented in an on-chip specialized hardware accelerator that leverages micro-architectural performance counters. Boreas outperforms existing thermal management techniques while remaining lightweight and well-suited for implementation in hardware. Even with a conservative thermal sensor delay, Boreas is able to predict severity with high precision, resulting in effective hotspot mitigation on unseen workloads. These machine learning models were, therefore, able to select a frequency that was 4.5% better than thermal only models on average, and up to 9.6% higher in the best case, while having the same reliability budget as the thermal models.

I. Introduction

As industry scales to low single-digit process nodes, thermal hotspots have become a first-order design challenge for a wide range of applications. While hot chips have been around for years, recent work has shown the existence and abundance of *advanced* thermal hotspots, or portions of on-chip logic which do not just get hot, but heat up rapidly and in a manner that is both non-uniform and application dependent [1]. Addressing advanced thermal hotspots using conventional techniques like dynamic voltage and frequency scaling (DVFS) [2] would require large guardbands resulting in non-trivial performance loss. Instead, the systems community needs new hotspot mitigation methods which can avoid hotspots with limited performance loss.

Modern computer systems use an ensemble of technologies to prevent hotspots, from thermal sensors and DVFS [2] to floorplanning [3]–[5] and physical cooling [6]–[8]. However, due to the microsecond granularity at which advanced hotspots can occur [1], these techniques alone are no longer cost-effective and lightweight enough to maintain a safe CPU operating temperature. This is primarily because these techniques are reactive, and the time it takes for them to take effect is longer than the time it takes for hotspots to appear. In terms of techniques like floorplanning, prior work has studied the utility of extending existing floorplanning mitigation methods and has shown that even scaling the area of hotspot prone functional units by 10× in a 7nm processor still results in *Hotspot-Severity* worse than in 14nm [1].

Given the speed at which advanced hotspots occur, chip designers can no longer rely solely on temperature readings from thermal sensors or floorplanning based methods. Instead, new fine-grained, architecture level mitigations are needed which take into account multiple aspects of the system. To this end, there has been recent work which has provided updated models, metrics, and methods for characterizing advanced hotspots, which can be used by researchers as a foundation for building mitigation techniques. One contribution of the researchers is the design of a function called *Hotspot-Severity* which uses both absolute temperature and localized temperature differences to determine how imminently a hotspot will

#David Werner and Maziar Amiraski are co-first authors.

occur in a region of the chip. We build on this prior work to design the first machine learning-based architecture-level mitigation method for advanced hotspots. In this work, we also study the extent to which microarchitecture counters can inform us about whether a hotspot is occurring and whether we can use it to predict hotspots ahead of time as part of Boreas.

In summary, this paper presents a novel machine learning based hoptspot mitigation method which uses *Hotspot-Severity* and telemetry data to predict hotspots *before they arise* and dynamically scale voltage and frequency. By acting preemptively, it is possible to apply conventional techniques like DVFS without incurring the same performance penalty that would exist from using DVFS in the reactive manner. We name our mitigation after the Greek God of winter, Boreas, who is said to chill the air with his icy breath. To motivate Boreas, we first discuss and analyze 1) an oracle model, and 2) static, temperature-based techniques that rely only on temperature sensors. We then describe Boreas in detail, contextualize it among previous works, and perform a comparative evaluation of it with related work. For this evaluation, we use a simulated modern desktop CPU similar to Intel Skylake running the SPEC CPU2006 benchmarks. We show that Boreas surpasses the performance of temperature-only techniques, resulting in accurate severity prediction and mitigation on unseen workloads. Boreas can select a frequency that is 4.5% better than thermal only models on average, and up to 9.6% higher in the best case, while being as reliable as the thermal models.

II. Background

A. Advanced Thermal Hotspots

Advanced thermal hotspots have arisen due to recent chip design trends in the microprocessor industry. While transistors scale to ever smaller dimensions, die area remains about the same with the die packing more functionality (and transistors). With the end of Dennard scaling [9] and continuous increase in transistor density, power density for computational logic is increasing rapidly. Additionally, microprocessors are becoming increasingly heterogeneous [10], [11]. It is typical for modern processors to run a wide variety of workloads—everything from security kernels, to video encoding/decoding, to multiply-accumulate (MAC) rich kernels for machine learning. This exercises logic with very different switching activity that is very close to each other. Because of architecture heterogeneity, increased power density, and asymmetry in switching activity from workloads, advanced thermal hotspots are significantly faster, non-uniform, and application dependent than previously observed hotspots. These characteristics make it very challenging (if possible) to guarantee hotspot-free operation at design time using conventional mitigations.

B. HotGauge

The benefits of using a silicon-calibrated simulation environment such as *HotGauge* rather than real silicon are profound and essential to this work. First, a simulation environment allows for the evaluation of future technology nodes and features not present in current processors, including those that would require additional circuitry or modifications to the micro-architecture of the processor. Second, the detailed performance, power, and thermal simulations allow for low-level insights into the state of the processor that are simply not feasible in silicon; most notably, perfect knowledge of the thermal state of the entire thermal stack. Additionally, a simulation pipeline has the ability to simulate pushing a processor up to—and even past—it's thermal limits without destroying it or having to circumvent the thermal protection schemes currently implemented in both hardware and software.

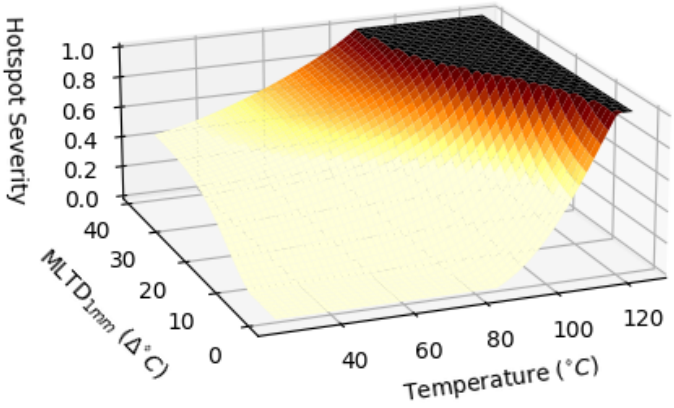


Fig. 1: *Hotspot-Severity* as in *HotGauge* [1]

HotGauge [1] is the first open source implementation of a simulation pipeline which includes performance, power, and thermal simulation as well as novel metrics for automatically classifying and detecting hotspots (including advanced hotspots). Now that such a methodology exists, this allows for mitigation methods to be implemented, tested, and compared using simulation. We utilize this methodology in our work to propose the first public study of a mitigation for advanced thermal hotspots. To the best of our knowledge, there have been no novel hotspot mitigation methods which address the speed, asymmetry, and application dependence of advanced thermal hotspots.

While previous works generally use temperature or some derivative (e.g., heat flux) as the metric to determine whether a hotspot has occurred [3], [6], [12]–[14], this disregards the effect that high *relative* temperature differences within a certain area can have on a chip. On the other hand, HotGauge characterizes hotspots using both absolute temperature and maximum localized temperature difference (MLTD) to model the two ways that hotspots can harm a system. This includes: (1) the effect of absolute temperature, especially when surpassing the thermal design point (TDP) of the chip, and (2) the relative temperature difference within some radius (i.e., MLTD) which can affect safe clock timing margins. When these margins are violated, the processor can either malfunction or potentially crash. These two factors are both combined into one *Hotspot-Severity* metric which can account

for both device failure and timing concerns using a single value. The values that *Hotspot-Severity* can take range between 0 and 1, where a value of 1 indicates that the chip is in immediate danger of malfunctioning or encountering permanent physical damage. This work uses the same *Hotspot-Severity* parameters as *HotGauge*, although it should be noted that they can also be tuned to fit other parameters of operation for other design-processes and circuit implementations. Given these parameters, the *Hotspot-Severity* behaves as shown in Fig. 1, and will be 1.0 under conditions including: 1) a temperature of 115C with little to no MLTD (i.e a uniformly hot chip), 2) a temperature of 80C with an MLTD of 40C (i.e. an advanced hotspot), and 3) somewhere between (1) and (2), with a temperature of 95C and an MLTD of 20C.

This work uses *Hotspot-Severity* as part of a machine learning model to predict the occurrence of hotspots. Then, hotspots can be avoided by selecting an appropriate voltage and frequency point for the processor during runtime. *Hotspot-Severity* can be used to compare the impact of different architectural changes (e.g., floorplans) on hotspot behavior, which is demonstrated in Section III by comparing the results of existing techniques with the novel machine learning models developed in this work (Section IV).

C. Previous Work on Machine Learning for Thermal Management

Researchers have previously utilized machine learning for thermal management on modern single and multi processors [15]–[22]. Least square regression (linear or non-linear) as a supervised learning method has been a popular approach to the thermal management problem on processors [21], [22]. Cochran and Reda [20] combined least square regression with k-means clustering to reduce the dimensionality of the data and consequently runtime overhead of prediction.

Another popular method in recent years has been the application of reinforcement learning, most generally Q-learning as an online training methodology. Lu et al. [17] employed Q-learning with linear combination of series of radial basis functions, using temperature readings from sensors to form the state space. Splash-2 benchmarks, McPAT, Sniper and HotSpot were used to generate power traces and convert them to temperature values, with the ultimate goal of reducing peak temperature throughout the chip. Iranfar et al. [16] implemented a heuristic approach alongside Q-learning to restrict the learning state space and achieve a faster convergence. States are based on temperature and temperature-gradients and actions are comprised of thread migration and DVFS. They have shown their method has improvement on average performance, thermal cycle amplitude and thermal cycle frequency compared to previous works. Das et al. [19] used Q-learning by forming the state space based on stress and aging values. The action space is comprised of thread affinity and CPU governors at fixed intervals. Ye and Xu [18] used a deep-Q-network using processor power state (run vs sleep) and task queue status as its state space. Their action space has two components, the core they assign the tasks to and the preset mode of that core. The result is an agent which seeks to reduce the power consumption with minimum impact on performance.

One thing that all of the mentioned reinforcement learning methods have in common is that they are online training algorithms. They start with no knowledge of the environment and gradually learn the optimal policy that result in highest reward (depending on the goal they've set such as lowest peak temperature, lowest stress, longest time to failure, etc.). During the course of this learning process, the agents make considerably sub-optimal decisions that necessitates having a backup control strategy to prevent catastrophic failures from happening. Additionally, another general problem with reinforcement learning methods is their sample inefficiency and the high number of iterations they need to achieve an optimal answer, which is why many of the mentioned previous works try to restrict the design space and implement heuristics to achieve a faster convergence. This again underpins the need for safety precautions when using any learning based methodology, before they reach an acceptable level of accuracy. **Finally, all of the previous work uses machine learning to predict temperature, but they have not been shown to predict hotspots.**

III. A Case Study on Static, Temperature-Only Based Voltage and Frequency Selection

Modern processors have to avoid thermal conditions that could result in timing violations or circuit damage while keeping performance high. The main focus of this work is to illustrate how using machine learning to predict advanced hotspots can allow for the use of higher performance Voltage and Frequency (*VF*) pairs for DVFS. In this section, a variety of *VF* selection algorithms are described and evaluated. The goal of each algorithm will be to choose the most performant *VF* pair such that the maximum hotspot severity remains below 1.0, thereby preventing device malfunction and damage. All of the algorithms evaluated in this work select a single frequency for the entirety of the application trace which each last between the orders of tens and hundreds of milliseconds, a realistic timescale for thermal management. This sections' proposal is not to replace DVFS with just one optimal VF point, but to illustrate how hotspot prediction can enable the selection of a close-to-performance-optimal VF point that avoids hotspots.

A. Simulation Environment

This work utilizes the same system models and configuration parameters as HotGauge, which have been validated and released open source [1]. This includes a performance, power, and thermal model of a desktop client processor based on an Intel Skylake. Like in the HotGauge work, we use the same voltage and frequency which is representative of running the processor in turbo mode, in order to obtain the worst case hotspot behavior of the chip to test our machine learning models on. The exact performance, power, and thermal models details, including the floorplans, are the same as those used in the HotGauge publication [1]. Similar to the HotGauge work, a

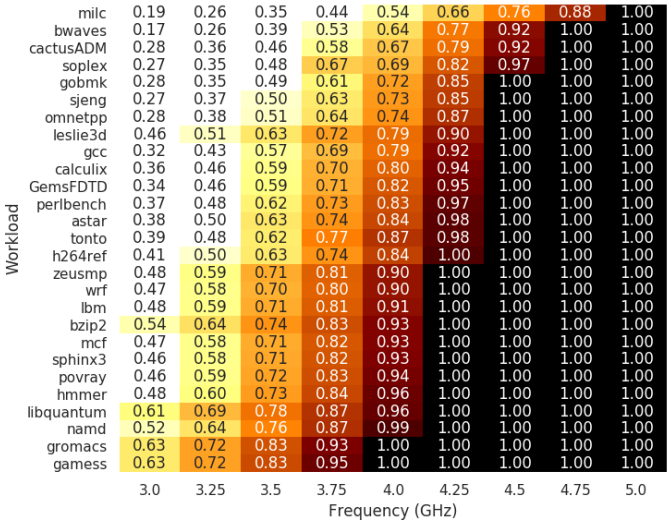


Fig. 2: The peak *Hotspot-Severity* of each workload over a range of frequencies. The combinations of workload and frequency with a *Hotspot-Severity* of 1.0 are not safe operating points and are shaded in black. Values 0.5 and below are white and the remaining values are shaded according to the magnitude of their *Hotspot-Severity*.

subset of 27 SPEC2006 workloads are evaluated in this work. A full list of these workloads can be seen in Fig. 2. Each workload executes on a single core while the others remain idle. For each workload, the frequency of the processor is varied from 2.0GHz to 5.0GHz in 250MHz steps, with the voltage set as shown in table I.

TABLE I: Select Voltage and Frequency (*VF*) pairs for the 7nm processor modeled in this work

Voltage [V]	0.64	0.71	0.77	0.87	0.98	1.15	1.4
Frequency [GHz]	2.0	2.5	3.0	3.5	4.0	4.5	5.0

This work includes evaluations where the thermal sensor is placed at various locations on the active core. The specific locations are chosen using the same methodology as HotGauge, namely by using K-means clustering to identify common areas on the core where hotspots arise, repeated for different values of k. Unless otherwise noted, temperature sensors in this work are located near the ALUs (in the EX stage of the pipeline), and correspond to sensor 3, which is shown later in this work, including in Fig. 5.

Given the rich dataset produced by the simulation pipeline, a variety of models can be constructed that are differentiated only by the information that is made available to them. The models include two baseline categories; 1) an optimally performing oracle (Sec. III-B) and 2) models that react based on thermal sensor readings similar to ones currently implemented in hardware (Secs. III-C and III-D). **Together, these models provide upper and lower bounds points of reference when evaluating the machine learning model developed as part of this work.**

B. VF Oracle

The *oracle-model* assumes perfect knowledge of application behaviour and the corresponding impact on IC thermals. While this model is by no means possible to implement, it serves as an important upper bound for how well a model could possibly perform. By definition, the oracle model will select the most performant VF point at which the maximum *Hotspot-Severity* remains below 1.0 for the duration of the trace.

This model is developed by computing the peak *Hotspot-Severity* while running each workload under all simulation configurations, and then choosing the optimal *VF* settings for each scenario. Fig. 2 shows a tabulation of this data. Based on the peak *Hotspot-Severity* in each simulation, none of the workloads can safely operate at 5.0GHz, while all of them can do so at 3.75GHz. The oracle selects the highest *VF* point where the *Hotspot-Severity* is below 1.0.

C. Global VF Limit

The most heavy handed way to eliminate hotspots is simply to set the maximum *VF* point such that the peak *Hotspot-Severity* of any workload remains below 1. This effectively models a default safe nominal frequency configured for most microprocessors but misses opportunities to boost frequency and increase single-thread performance for certain workloads using application-aware DVFS techniques like Turbo Boost [23]–[25].

Given Fig. 2, the globally safe *VF* limit for the system in this work with the given workloads is 3.75GHz. This results in optimal performance for only 2 of the 27 workloads. The majority of workloads are set to a frequency that is 13% lower than the oracle, with a worst case frequency reduction of 26%.

D. Thermal Aware VF Limits

In contrast to avoiding hotspots by setting a global VF limit, industry has opted to set dynamic limits based on observed processor temperature. To evaluate the effectiveness of this technique, models are constructed in this section that use only the current temperature value for their prediction of whether or not the peak *Hotspot-Severity* will be less than 1.0.

As previously noted, the thermal sensors in this work are considered under the most optimistic conditions in terms of location, noise, precision, and accuracy. This is done to evaluate the thermal-aware models developed in this section under the most generous of conditions.

1) Application Specific Critical Temperatures: In order to create thermal-aware models, *critical-temperatures* are first found for each workload at all possible frequencies. Here, the *critical-temperature* is defined to be the lowest observed temperature on a given sensor when the *Hotspot-Severity* is 1.0. The system uses this as a threshold for each workload to which it responds by decreasing the frequency.

From this data, the impact of sensor location on critical temperatures is apparent. When comparing critical temperatures across the top 4 sensors, all workloads had at least one frequency where the critical temperatures varied by at least 13C. Furthermore, about half—13 of the 27 workloads—had

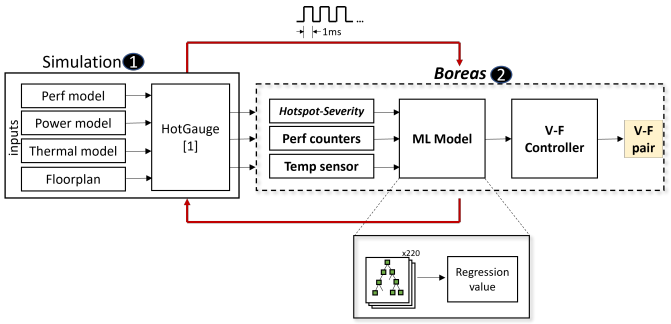


Fig. 3: Overview of Boreas, a framework to mitigate advanced hotspots using a low cost ML model and hardware telemetry data.

critical temperatures at some frequency that varied by *over 20C*, with a peak variance of over 37C between sensors. This means that, the temperature at which the processor needs to throttle varies by at least 13C based on sensor location, even if the workload is known. Sensor delay is also very impactful. For example, gromacs can safely run at 4.75GHz up to a temperature of 70C before throttling when the delay is only $180\mu s$ but can never run above 4.25GHz with a sensor delay of $960\mu s$ because a hotspot can arise in less time than it takes to read the sensor, even when starting from ambient temperatures. In comparison, *sjeng* can safely reach over 65C at 5GHz with a $960\mu s$ delay, which indicates that it doesn't have high spikes in power that could lead to fast hotspots.

2) Global Critical Temperatures: In order to complete this thermal aware frequency selection algorithm, all possible workloads must be considered. Thus, the algorithm must select the lowest critical temperature at every frequency across all workloads. Thus, workloads with particularly high critical temperatures are severely limited by those with lower critical temperatures. With a $960\mu s$ delay, *libquantum* cannot ever run above 4.0GHz, so every workload is limited to that. Even with 0 delay, workloads like *cactusADM*, which can run safely at 4.75GHz, are limited by the workloads with lower critical temperatures. For example, *games*, which has a critical temperature of as low as 60C even when running at 4.0GHz. So, even though *cactusADM* can run without concern at 4.0GHz, it will be throttled if the temperature sensor reaches 60C. The impact of this variation of critical temperatures between workloads has dramatic impact on frequency selection.

IV. Boreas: Main Ideas

An overview of Boreas is shown in Fig. 3. The HotGauge framework is used to generate the inputs to Boreas, which are *Hotspot-Severity*, performance counters, and thermal sensor readings. The configuration of HotGauge in this step is the same configuration used in Section III. Boreas then operates on these inputs by feeding them into the ML model composed of Gradient Boosting Trees (GBTs) and trains a regression model. The model is trained to accurately predict the severity of the future steps using the input attributes. The predicted output is then fed into the V-F controller which selects the appropriate V-F pair for the next time step. The model inputs—shown above as red arrows—are fed into Boreas every $\tilde{1}$ms ($960\mu s$). In the following subsections, we describe in detail each of the components of Boreas.

TABLE II: Boreas model parameters.

Dataset	500K instances extracted from SPEC2006 benchmarks shown in Table III
Features	Temperature sensor data alongside microarchitectural attributes shown in Table IV
Hyperparameters	$\alpha = 0.3, \gamma = 0$, max_depth = 3, n_estimators = 223

A. Regression Models with GBTs

Machine learning has been used in different areas of computer architecture research. Prior work has employed machine learning to study, design and improve prefetchers [26], [27], cache management techniques [28], [29], and branch predictors [30]. While striving to achieve models with the best accuracy, one major concern in all of these studies is how much the model costs. Cost can come in different forms such as memory-footprint, circuit, area, power consumption and prediction latency. While a deep learning model can probably provide the highest accuracy, it might have a prohibitively high cost, ruling it out for hardware implementation. Therefore, in this section we describe new hotspot prediction models that have acceptable cost while maintaining accuracy.

A regression model is trained to predict the hotspot severity of workloads. The models developed for this study use Gradient Boosting Trees (GBTs) using XGBoost library to train a regression model that predicts the maximum severity in the next time-steps. Training a GBT for regression is pretty straightforward. An initial value (usually the mean value of target label for all instances) is used as the initial prediction for severity. Using this initial number, the residuals are calculated. In this case residuals are the difference between the real values and the predicted values. Next, depending on the model configuration (e.g. number of trees, maximum depth of trees, maximum number of leaves, etc.) a tree is trained on the calculated residuals. At this point we have an original mean value and a tree and we have to calculate the new residuals. To do this, the attributes of each instance is used to traverse from the root to the leaf, and the value of that leaf multiplied by the learning rate is added to the original prediction. In other words, each prediction by a tree helps to bring the original number closer to the real value. This procedure is repeated until the number of trees in the configuration is exhausted. Table II shows the final configuration used to train the Boreas model where alpha is the learning rate, gamma is the minimum loss reduction needed for a split, max_depth is the maximum depth of trees (used instead of maximum leaf nodes) and n_estimators is the number of trained trees in the model.

GBTs are not only one of the most powerful models available, they also have several characteristics which makes them a

desirable choice for this application. GBTs do not need feature normalization, they can be visualized for interpretability and they have a manageable hardware implementation due to the simplicity of tree logic. However, one downside to these models is that they are easy to overfit. Too many trees or trees that are too deep can cause the model to memorize the input dataset, showing very high accuracy on the training set but failure to generalize on the data that it has not been exposed to (testing set). Therefore care must be taken when tuning the hyperparameters.

Train/Test sets. Before training the Boreas model, we first divide the set of workloads into training and test sets. When a workload is assigned to a set, every instance from that workload becomes exclusive to that set. This is done to prevent any knowledge leakage from the train set into the test set. We performed set assignment based on the data shown in Figure 2, where workloads are sorted by their peak *Hotspot-Severity*. Every fourth workload was assigned to the test set, with the remaining workloads being assigned to the training set. The goal of using this method of assignment is to impose as much diversity into the selection process as possible to prevent any bias in the application selection. In other words, workloads with a range of hotspot behavior are assigned to each of the sets to ensure that the model is able to learn, generalize and be evaluated on different workloads with a wide range of hotspot behaviour. The resulting training and test sets are shown in Table III. Our training set comprises 411K sample instances from workloads in the train set. The size of the test set is 70K instances from running workloads in the test set. Each instance is a row with 20 features (Table IV) extracted every 80 microseconds from 12 milliseconds of total runs.

Grid search CV. Next, grid-search cross-validation is performed on the instances in the training set. A grid-search is done on the model parameters such as the number of trees, the depth of trees, the sample instance weights, etc. to find the most accurate model with the lowest MSE (Mean Squared Error). Cross-validation is done in the form of a modified Leave-One-Out Cross-Validation (LOOCV) in which an application is designated as the validation application and every instance of that application is taken out from the training set. The rest of the applications comprise the training set which are used for grid-searched training. Every configuration result in a model that is validated against the validation application. In the end, the most accurate model is chosen from the list of all generated models. This decision is made based on the average and standard deviation of the trained model accuracies for each parameter set configuration. The grid-searched cross-validated model is then used on the test set. Our chosen model is comprised of 223 trees with each having the maximum depth of 3, and has MSE of 0.0094 which shows a highly accurate severity predictor.

B. Feature Selection

Models were initially trained using 78 system attributes. While XGBoost is designed to be able to handle a large number of features, using a large number of features also increases overhead. In the context of a hardware implementation, it is particularly important to limit overhead, as a higher number of features can directly increase the implementation cost in the form of increased area, latency, and/or power, alongside an increased memory footprint. In order to reduce these overhead, a feature selection study was conducted to find the most influential features on the models accuracy. Keeping these attributes and eliminating the other less effective ones should lead to the nearly the same level of accuracy from the model while reducing the total number of features used, thereby reducing the model overhead.

The feature selection process is as follows: We first trained a model using all 78 features that were extracted from our simulation runs. XGBoost uses gain and similarity scores to design nodes and the trees. The higher the gain of the feature, the more important it is in the regression evaluation. We sorted these features and iteratively removed them (from the least important) until we saw a decrease in the model accuracy at 20 features (Table III). This was expected since the top 20 features have 99% of the total normalized gain which allow the model to achieve very good accuracy.

Table IV shows the top 20 attributes selected to train the model sorted from the least to most important feature. Temperature sensor data is an obvious choice here due to its clear effect on emerging hotspots (78% of the total gain). However, the remaining 22% is divided between the microarchitectural features. Looking at top attributes, our expectation was to see frequency as a prominent feature but that's not the case. While frequency was passed to the feature selection algorithm during the training process, its effect seems to be reflected in other features. In other words, while frequency is not part of the top feature list, its effect is reflected in features such as total misses or committed instructions etc.

C. Temperature-Only Models

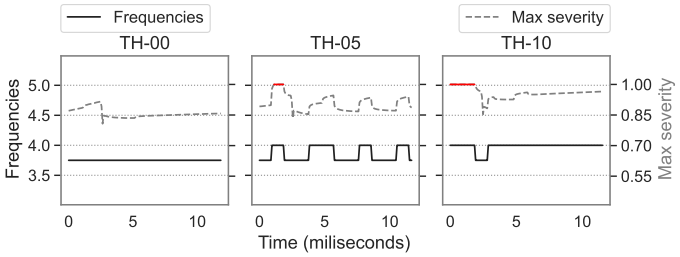
To show the importance of using microarchitectural features alongside temperature and illustrate the fact that sensor data alone is not indicative enough of the emergence of hotspots, a comparative study has been conducted with a previous work with a similar approach. Cochran and Reda [20] extract performance counters from each core and use this data to predict the temperature in the future. During the offline training process, steps are taken to reduce the complexity of the prediction process during system runtime. The raw performance data is first fed to a Principle Component Analysis (PCA) module which reduces the dimensionality of data without losing important information required for training an accurate model. The principle components are then used to create representative centeroids for workload phases using k-means clustering method. A phase in a workload is a period in which the workload exhibit similar power and temperature patterns. Next, the thermal prediction model is learned using a linear regression of the phases for each frequency. The model is then used during runtime operation to predict the future temperature and adjust the frequency to a safe threshold.

TABLE III: Workloads divided between train and test sets

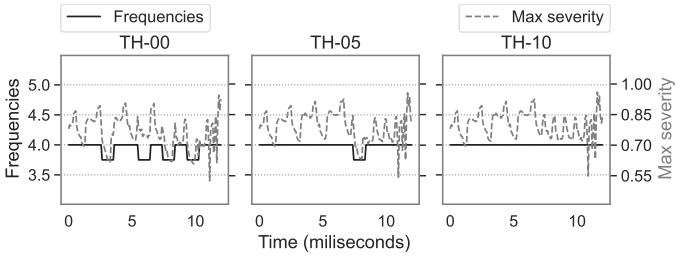
Train	milc, bwaves, soplex, gobmk, sjeng, leslie3d, gcc, calculix, perlbench astar, tonto, zeusmp, wrf, lbm, mcf sphinx3, povray, libquantum, namd, gromacs
Test	cactusADM, omnetpp, GemsFDTD, h264ref, bzip2, hmmer, gamess

TABLE IV: Top 20 attributes sorted by Normalized Gain (importance) that are used to train the model. There is no loss in regression accuracy if the model is trained on these top 20 features rather than all 78 features. Temperature sensor data is the most dominant feature in severity regression (78% normalized gain).

Attribute [importance]		Attribute [importance]		Attribute [importance]		Attribute [importance]
temperature_sensor_data [78.1%]		ROB_reads [2.2%]		itlb_total_misses [0.5%]		branch_mispredictions [0.3%]
cdb_alu_accesses [3.1%]		total_cycles [1.7%]		BTB_read_accesses [0.5%]		LSU_duty_cycle [0.3%]
committed_instructions [2.5%]	...	icache_read_accesses [1.3%]	...	dcache_read_misses [0.4%]	...	IFU_duty_cycle [0.3%]
dcache_read_accesses [2.5%]		committed_int_instructions [1.0%]		cdb_fpu_accesses [0.4%]		FPU_cdb_duty_cycle [0.2%]
busy_cycles [2.3%]		dtlb_total_accesses [0.8%]		MUL_cdb_duty_cycle [0.4%]		dcache_write_accesses [0.2%]



(a) gromacs



(b) gamess

Fig. 4: Frequency vs. max severity for *gromacs* and *gamess* using different thermal models. TH-00 is a thermal model based on globally safe threshold. TH-05 and TH-10 are thermal models with thresholds increased by 5 and 10 degrees respectively. The higher the threshold, the more aggressive frequencies adopted by the controller become. While safe for *gamess*, relaxing the global threshold results in reliability issues (hotspot incursions) in *gromacs* (shown in red for TH-05 and TH-10)

Figure 4 is a case study to show that even with a perfect prediction of temperature using thermal models, there is a considerable performance gap between what a system can, but does not achieve due to reliability concerns. In this figure TH-00 is a thermal model trained on a threshold that is safe for all workloads in the training set. The controller uses these thresholds and current temperature reading to increase or decrease the frequency of the system. TH-05 and TH-10 are the same thermal model but with a more relaxed thresholds. In these models the thermal thresholds are increased by 5 and 10 degrees respectively. Therefore the controller can choose more aggressive frequencies for the system.

The goal of the controller, as always, is to adjust the system frequency to prevent the severity going above 1. This is achieved by using TH-00 for *gromacs* in 4a. However, relaxing the threshold by 5 degrees (TH-05) would result in a bad decision around 1 ms. A higher frequency is chosen by the controller that raises the severity to 1 for several steps before correcting the course. The problem is exacerbated by using TH-10.

Figure 4b shows the same set of figures but this time for *gamess* in test set. Similar to *gromacs*, TH-00 is also safe for *gamess*. But contrary to what we saw with relaxed models for *gromacs*, *gamess* works reliably using TH-05 and TH-10. The controller sets the frequency at 3.75 GHz most of the time and occasionally adjusts it to 4 GHz when using TH-00. This meets the global thermal thresholds while training the model on the training set. Increasing the threshold will allow the controller to be more aggressive in selecting higher frequencies for workloads such as *gamess*, but can cause hotspot on one or more workloads (for example *gromacs*). Increasing the threshold to TH-10 gradually increases the frequencies chosen by the controller and will eventually lead to the frequency of 4 GHz kept by the controller for the whole run. This happens without any incursions into the dangerous 1.0 severity levels.

This case study clearly shows that the common practice of using global thermal thresholds to prevent reliability issues can adversely effect the performance of the system. As we will show, a non-agnostic model that uses system attributes alongside temperature can lead to a higher performance while being as reliable as thermal models.

V. Evaluation with Dynamic Simulations

To evaluate our model, dynamic architectural and thermal simulations are performed using HotGauge. During these simulations, collected microarchitectural attributes are combined with temperature data to create a feature vector. This feature vector is passed to the model to make a severity prediction. The result of this regression is used by the controller to adjust the frequency to a safe value. Alongside our model, thermal models are trained on perfect temperature data. The analysis shown in next sections illustrates the advantage of using a

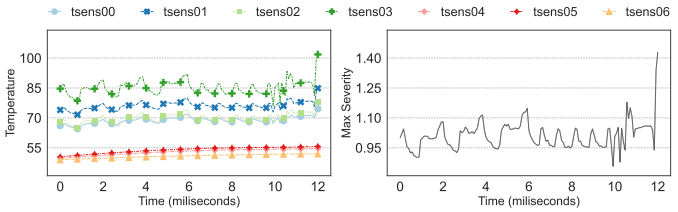


Fig. 5: Left figure shows temperature data from sensors placed in 7 different locations on the chip. 3 of the 7 sensors cannot show the temperature of the chip properly. The four others have twenty degrees difference between the value they report. Considering the best sensor location (tsens03), temperature is below 100 degrees almost all the time but severity reports very high values. This shows we can have hotspot incursions even if an acceptable temperature is reported by a sensor.

model sensitive to diverse behavior of applications compared to a thermal only model.

A. Frequency controller

The trained model introduced in section IV-A is deployed to make frequency decisions during system runtime. Each timestep is 80 Microseconds and decisions are made every 12 timesteps (960 μs). Therefore the controller can either increase or decrease the frequency around every 1 miliseconds. At this time, features extracted from system attributes alongside temperature sensor data is sent to the model, which makes a prediction about the future severity. If it finds it unsafe to continue with the same frequency, the controller decreases the frequency by 250 MHz for the next 1ms. However, if it finds it safe to stay at the same frequency, the controller checks the possibility of operating at a higher frequency. The model predicts the severity at 250 MHz higher, and if prediction shows an acceptable severity, the frequency is increased. Otherwise the system will operate at the current frequency for the next period.

B. Hotspot Mitigation Through Hotspot Prediction

Sensor placement can have a huge impact on how the controller reacts depending on its perceived temperature state. Figure 5 shows temperature data from sensors placed at 7 different locations on the chip. Three of these 7 sensors (tsens04, tsens05 and tsens06) are not placed in positions to correctly reflect the temperature variations on different parts of the chip. Their data only reflects the fact that the chip is gradually getting warmer. Therefore they are not good candidates to predict and prevent hotspots.

The remaining four (tsens0-3) capture these variations with different degrees of accuracy. tsens00 is the least and tsens03 is the most accurate sensor. Interestingly, there is up to 20 degrees difference in temperature values readings. Therefore we have to have a clear understanding of (micro)architecture components and heat dissipation structure of the chip to choose a perfect location for the sensor. Considering everything, even our ideal sensor location shows the temperature to be below 90 degrees, while severity metric is above 1 during all those periods. This illustrates the point discussed in section II-B that we can have hotspot incursions even if temperature sensors show relatively safe values. This is another reason why large guardbands are employed to prevent hotspots in a global manner.

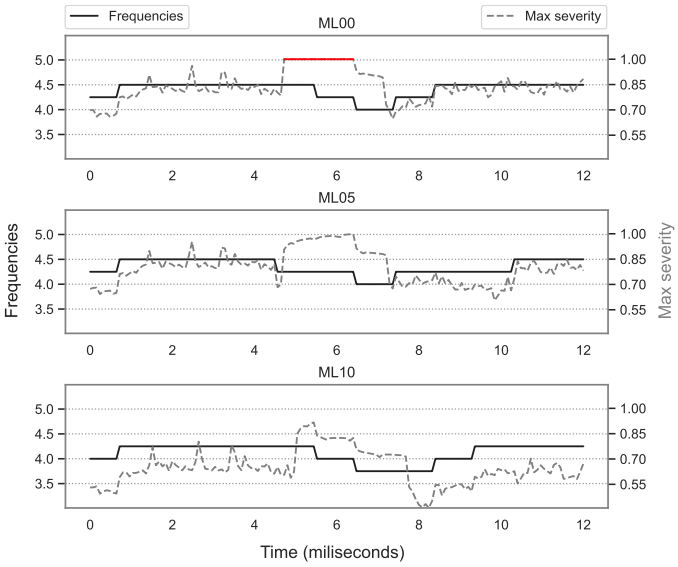


Fig. 6: Frequency vs. max severity for *bzip2* using ML models with different guardbands. ML00 is the model with no guardband (severity threshold at 1.0). ML05 and ML10 are models with 5 and 10 percent guardbands respectively (severity threshold at 0.95 and 0.9 respectively). The larger the guardband, the lower the frequencies selected by the controller (safer but with less performance). ML05 trades-off reliability and performance well.

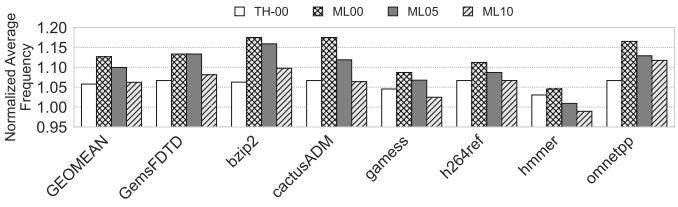


Fig. 7: Average frequency of unseen workloads using every model normalized to the baseline frequency. ML00 has the highest performance but the least reliability. ML10 is as safe as globally set thermal models but with very small performance advantage. ML05 achieves 4.5 percent performance gain compared to TH-00 on average without any hotspot incursions.

C. Targeting different relibility budgets by setting prediction guardbands

The goal is to train the most accurate model that can predict the future severity with high precision. This allows the controller to adjust the frequency so that the system can operate close to the max severity of 1. In practice, however, no matter how accurate the model is, there can be an instance where the model mispredicts. This misprediction can lead to a wrong decision regarding frequency adjustment, which in

turn can lead to dangerously high severity, something that is not acceptable to chip designers. This is the reason why many manufacturers set conservative thresholds regarding temperature to decrease the possibility of damage to a negligible amount.

To respect these concerns and to take the situations with sudden increase in severity into account, a range of guardbands were applied to the model's predictions. Controller considers these guardbands during the decision making process to increase or decrease the frequency. Figure 6 shows severity prediction vs frequency using three different guardbands (0, 5, 10) for *bzip2* as an illustrative example. The guardbands set the threshold offset for frequency controller. The original threshold is set to severity of 1. Anything above this threshold is detrimental to the system and therefore the controller uses the model's prediction to prevent this by making an appropriate decision after each interval. A guardband of zero is the same as having no guardband and the threshold stays at severity of 1 (ML00). A guardband of 5 decreases the threshold by 5 percent, meaning the severity of 0.95 (ML05). At this threshold, controller decides whether to decrease, or not to increase the frequency if the predicted severity is above 0.95. The same can be said about the guardband of 10 and severity threshold of 0.9 (ML10). The larger the guardband, the more conservative the controller becomes.

Clearly there is a trade off here. The threshold can be set high to make the system more reliable at the expense of lower frequency and lower performance. On the other hand if we set the guardbands too low, the system can operate at higher frequencies. However, this would result in an unreliable environment. First plot in figure 6 shows such situation for *bzip2*. At 0 guardband, controller sets the frequency too high and this results in several steps where the severity reaches 1 and hotspot incursions happen. But with the higher guardbands, while the severity gets very close to 1, it never reaches it. ML10 is the safe option. The large guardband creates a large gap between the maximum severity achieved in dynamic runs and the max severity of 1. However, this comes with the caveat that the adopted frequencies are too conservative. Therefore the performance gain is minimal. ML05 seems to be be the sweet spot. It trades-off reliability and performance well. It tries to get as close to severity of 1 as possible by adopting the highest safe frequency possible (ML05 gets close to 0.99 at 6.5 ms but reduces the frequency to prevent any incursions). This results in acceptable reliability and high performance at the same time.

D. Performance Analysis

By incorporating temperature sensor data and system attributes, the model predicts future severity. A safe guardband is applied to the threshold and the frequency is adjusted so that the system can perform with the highest reliable frequency. Figure 7 shows the average frequency of ML model with different guardbands alongside thermal model average frequency in all workloads. These average frequencies are normalized to the baseline frequency of 3.75 GHz, which is safe for all workloads, to give us a clear comparison point.

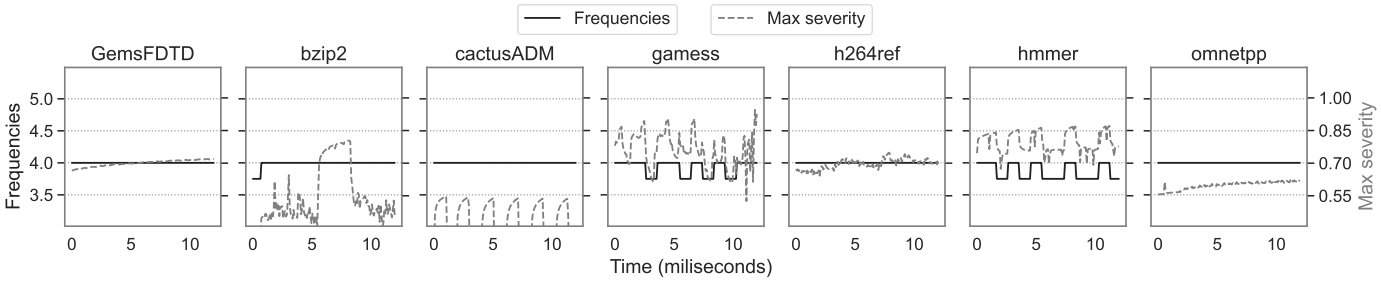
The thermal model achieves 5.7 percent improvement in average frequency compared to the baseline while ensuring a reliable environment for the chip. The story is different for ML models with different guardbands. As we have seen in figure 6, the guardband of 0 is not always safe for our workloads and can lead to hotspot incursions detrimental to our system. However, there is clear advantage in the performance. On the other hand, ML model with a guardband of 10 while safe, adjusts the frequency very conservatively and many opportunities for performance gain is lost. This is clear in *hmmer* where ML model performs worse compared to the thermal model, of which the latter is already cautious in choosing higher frequencies. The sweet spot is the guardband of 5, where we achieve the same level of reliability as the thermal model on both training and test sets, while having a 4.5 percent improvement in average frequency in all workloads.

Figure 8 shows the frequency vs. max severity plots for all unseen workloads in our test set using both Boreas (ML model with 5 percent guardband) and the thermal model. The left axis shows the frequency in GHz drawn with the solid black line. The right axis shows max severity drawn with a grey dashed line. As we can see in this figure, no application in the test set reaches the severity of 1 therefore both models have the same reliability and result in no hotspot incursions. However, as we have seen in figure 8, Boreas performs either at the same or one or two steps above the frequencies chosen by the thermal model (except in the case of *hmmer*). Boreas improves performance up to 9.6 percent in *bzip2*.

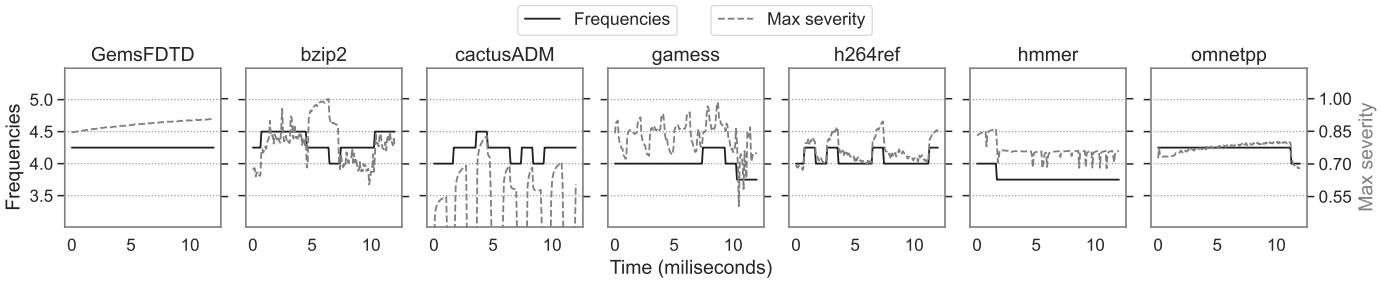
E. Overhead Analysis

Memory overhead. To have a high performance and reliable hardware implementation, the model must be accurate and cost-effective at the same time. Figure 9 shows the average MSE (Mean Squared Error) of cross-validated models vs. their size. Clearly the smallest models (a couple of shallow trees) cannot make good severity predictions and although small, have very low accuracy. As the size increases, the model gradually becomes more accurate and MSE decreases. This happens until model becomes too complicated and overfits on the training set. A small accurate model is selected for dynamic runs just before overfitting happens. Considering the model to be comprised of full trees with each node storing a 32-bit floating point value, the total size of model weights is less than 14KB. In addition to dedicated hardware implementation, the small model size allows it to be stored in lower level caches or its own scratch-pad memory to be invoked every 1ms.

Performance overhead. A model trained by XGBoost is a collection of trees where each tree contributes to bringing the prediction closer to the real value (making it more accurate). Considering this structure, the implementation is done using comparison operations (from the root of the tree to the leaf) and an add operation for the summation of the values in the leaves of trees. The Boreas model in the paper



(a) Thermal model



(b) Boreas

Fig. 8: Dynamic runs of unseen workloads shown using TH and Boreas (ML05) models for 150 timesteps (12 miliseconds). First this shows how Boreas allows controller to choose higher frequencies compared to TH-00 (except in *hmmer*) while keeping the severity below 1 at all times. Second it shows the range of behavior in different applications and the need for an adaptive, hardware telemetry dependent control mechanism to achieve higher performance.

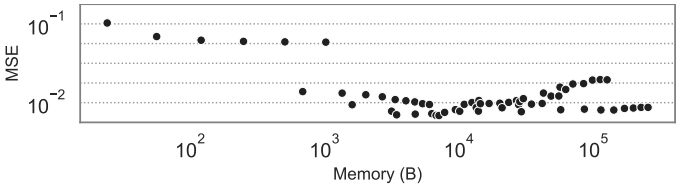


Fig. 9: Average MSE (Mean Squared Error) of cross-validated models vs. model size in Bytes. Increasing the size of the model, lowers the MSE until a certain point. After that the model overfits on the training set and MSE increases. The smallest, most accurate model is chosen for dynamic evaluations.

(ML05) comprises 223 trees each with a maximum depth of 3. Therefore for a single prediction, we are going to need 223 times 3 or 669 comparisons to traverse the paths from root to leaves and 222 add operations to accumulate the regression values in the leaves. This adds up to close to 1000 operations for one prediction of Boreas. This is of course the serial implementation with the worst case latency and lowest hardware cost. Parallelizing this would cut the latency by n while increasing the hardware implementation cost accordingly (n being the issue width).

VI. Conclusions

This work demonstrated the use of the *HotGauge* framework and the associated *Hotspot-Severity* metric to analyze the problem of selecting the *VF* point for the next time-step. The Novel machine learning models that were developed as part of Boreassurpassed the performance of the thermal-only models, resulting in accurate severity prediction and mitigation on unseen workloads. These machine learning models were, therefore, able to select a frequency that was 4.5% better than thermal only models on average, and up to 9.6% higher in the best case. This was achieved while having the same reliability budget as the thermal models.

References

[1] A. Hankin, D. Werner, J. Sebot, K. Vaidyanathan, M. Amiraski, and M. Hempstead, "Hotgauge: A methodology for characterizing advanced hotspots in modern and next generation processors," in *2021 IEEE International Symposium on Workload Characterization*, 2021.

[2] Q. Wu, M. Martonosi, D. W. Clark, V. J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks, "A dynamic compilation framework for controlling microprocessor energy and performance," in *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 38. USA: IEEE Computer Society, 2005, p. 271–282.

[3] B. C. Schäfer and T. Kim, "Autonomous temperature control technique in VLSI circuits through logic replication," *IET Computers & Digital Techniques*, vol. 3, pp. 62–71, 2009.

[4] C.-H. Tsai and S.-M. S. Kang, "Standard Cell Placement for Even On-chip Thermal Distribution," in *Proceedings of the 1999 International Symposium on Physical Design*, ser. ISPD '99. New York, NY, USA: ACM, 1999, pp. 179–184, event-place: Monterey, California, USA.

[5] R. Mukherjee, S. O. Memik, and G. Memik, "Temperature-aware resource allocation and binding in high-level synthesis," in *Proceedings. 42nd Design Automation Conference, 2005.*, Jun. 2005, pp. 196–201.

[6] Y. J. Lee, P. S. Lee, and S. K. Chou, "Hotspot Mitigating With Obliquely Finned Microchannel Heat Sink—An Experimental Study," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 3, no. 8, pp. 1332–1341, Aug. 2013.

[7] J.-C. Wang and T.-C. Chen, "Vapor chamber in high performance server," in *2009 4th International Microsystems, Packaging, Assembly and Circuits Technology Conference*, Oct. 2009, pp. 364–367, iSSN: 2150-5942.

[8] Lipeng Cao, J. P. Krusius, M. A. Korhonen, and T. S. Fisher, "Transient thermal management of portable electronics using heat storage and dynamic power dissipation control," *IEEE Transactions on Components, Packaging, and Manufacturing Technology: Part A*, vol. 21, no. 1, pp. 113–123, Mar. 1998.

[9] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[10] "M1 - apple." [Online]. Available: https://en.wikichip.org/wiki/apple/mx/m1

[11] M. J. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks, "The accelerator store: A shared memory framework for accelerator-based systems," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, jan 2012.

[12] A. Bar-Cohen, M. Arik, and M. Ohadi, "Direct Liquid Cooling of High Flux Micro and Nano Electronic Components," *Proceedings of the IEEE*, vol. 94, no. 8, pp. 1549–1570, Aug. 2006.

[13] P.-S. Lee and S. V. Garimella, "Hot-Spot Thermal Management With Flow Modulation in a Microchannel Heat Sink," in *Heat Transfer, Part A*, vol. 2005. Orlando, Florida, USA: ASME, 2005, pp. 643–647.

[14] B. C. Schafer and T. Kim, "Hotspots Elimination and Temperature Flattening in VLSI Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 11, pp. 1475–1487, Nov. 2008.

[15] S. Pagani, P. D. S. Manoj, A. Jantsch, and J. Henkel, "Machine Learning for Power, Energy, and Thermal Management on Multicore Processors: A Survey," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 1, pp. 101–116, Jan. 2020, conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[16] A. Iranfar, S. N. Shahsavani, M. Kamal, and A. Afzali-Kusha, "A heuristic machine learning-based algorithm for power and thermal management of heterogeneous MPSoCs," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Jul. 2015, pp. 291–296.

[17] S. J. Lu, R. Tessier, and W. Burleson, "Reinforcement Learning for Thermal-aware Many-core Task Allocation," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '15. New York, NY, USA: Association for Computing Machinery, May 2015, pp. 379–384. [Online]. Available: https://doi.org/10.1145/2742060.2742078

[18] R. Ye and Q. Xu, "Learning-based power management for multi-core processors via idle period manipulation," in *17th Asia and South Pacific Design Automation Conference*, Jan. 2012, pp. 115–120, iSSN: 2153-697X.

[19] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, Jun. 2014, pp. 1–6, iSSN: 0738-100X.

[20] R. Cochran and S. Reda, "Consistent runtime thermal prediction and control through workload phase detection," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: Association for Computing Machinery, Jun. 2010, pp. 62–67. [Online]. Available: https://doi.org/10.1145/1837274.1837292

[21] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, "Thermal and Energy Management of High-Performance Multicores: Distributed and Self-Calibrating Model-Predictive Controller," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 170–183, Jan. 2013, conference Name: IEEE Transactions on Parallel and Distributed Systems.

[22] ——, "A distributed and self-calibrating model-predictive controller for energy and thermal management of high-performance multicores," in *2011 Design, Automation & Test in Europe*, Mar. 2011, pp. 1–6, iSSN: 1558-1101.

[23] D. Lo and C. Kozyrakis, "Dynamic management of turbomode in modern multi-core chips," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 603–613.

[24] S. Yu, H. Yang, R. Wang, Z. Luan, and D. Qian, "Evaluating architecture impact on system energy efficiency," *PLOS ONE*, vol. 12, p. e0188428, 11 2017.

[25] S. Kondguli and M. Huang, "A case for a more effective, power-efficient turbo boosting," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 1, Mar. 2018. [Online]. Available: https://doi.org/10.1145/3170433

[26] R. Bera, K. Kanellopoulos, A. Nori, T. Shahroodi, S. Subramoney, and O. Mutlu, "Pythia: A customizable hardware prefetching framework using online reinforcement learning," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1121–1137.

[27] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 399–411.

[28] E. Teran, Z. Wang, and D. A. Jimenez, "Perceptron learning for reuse prediction," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, Oct. 2016, pp. 1–12.

[29] Z. Shi, X. Huang, A. Jain, and C. Lin, "Applying Deep Learning to the Cache Replacement Problem," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. Columbus, OH, USA: Association for Computing Machinery, Oct. 2019.

[30] D. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, 2001, pp. 197–206.