Neural Network-Based Task Scheduling with Preemptive Fan Control

Bilge Acun, Eun Kyung Lee, Yoonho Park
IBM T. J. Watson Research Center
Yorktown Heights, NY, 10598
Email: {bacun, eunkyung.lee, yoonho}@us.ibm.com

Laxmikant V. Kale
University of Illinois at Urbana-Champaign
Urbana, IL, 61801
Email: kale@illinois.edu

***Abstract*—As cooling cost is a significant portion of the total operating cost of supercomputers, improving the efficiency of the cooling mechanisms can significantly reduce the cost. Two sources of cooling inefficiency in existing computing systems are discussed in this paper: *temperature variations*, and *reactive fan speed control*. To address these problems, we propose a learning-based approach using a neural network model to accurately predict core temperatures, a preemptive fan control mechanism, and a thermal-aware load balancing algorithm that uses the temperature prediction model. We demonstrate that temperature variations among cores can be reduced from 9° C to 2° C, and that peak fan power can be reduced by 61%. These savings are realized with minimal performance degradation.**

***Index Terms*—Neural networks, Supercomputers, Temperature control, Power control, Fans**

I. Introduction

Building an exascale system within the 20 MW limit set by the U.S. Department of Energy is a significant challenge because power demands are not scaling with processor speeds [3], [4]. Up to half of the power demand is due to cooling components such as fans, and water pumps [22]. We believe that there is significant opportunity to reduce cooling costs through proactive actions to eliminate hot spots, and power peaks. For example, a thermal-aware scheduler can distribute, and migrate work to eliminate hot spots. To support proactive cooling actions, we propose a learning-based approach for making accurate thermal predictions. We utilize this learning-based approach in a thermal-aware load balancing algorithm that also employs a preemptive fan control mechanism. We demonstrate that this mechanism can reduce core temperature variations, and power peaks.

Accurately predicting individual core temperatures in a multicore system is difficult because of the complex thermodynamics in a data center, and manufacturing differences in the assembly process. Previous model-based approaches focus on predicting CPU temperatures with an assumption that all instances of a given hardware component behave similarly [11], [13]. Model parameters are fixed for a given hardware type. Trying to account for variations among hardware instances using these models requires that each instance be treated as a separate hardware type, substantially increasing the human involvement required for tuning model parameters.

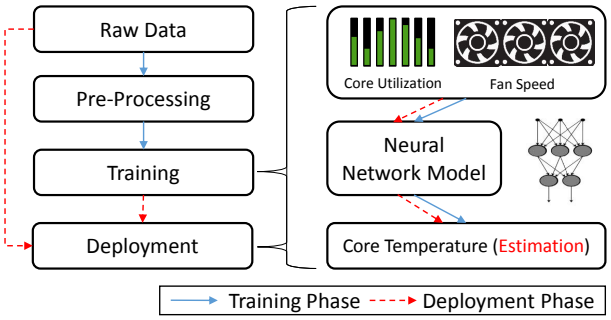

Fig. 1: Neural network-based thermal prediction approach.

We propose a learning-based approach for temperature prediction that can accommodate on-chip thermal variations caused by complex thermodynamics, and manufacturing differences. Figure 1 illustrates our approach, which consists of four steps: (1) monitor core temperatures, and utilization rates, (2) pre-process data, (3) train the neural network model, and (4) deploy the neural network to provide core temperature predictions. Because the neural network is trained on each specific processor separately, our approach provides more accurate temperature predictions than model-based approaches. Different neural-network back-propagation algorithms are studied in order to gain an understanding of the memory, and computation requirements involved in deploying the neural network.

We use our neural network temperature predictor in a novel thermal-aware load balancing algorithm that balances core temperatures, and utilizes a preemptive fan control mechanism. Our algorithm achieves temperature balance by greedily moving work from high-temperature cores to low-temperature cores. The neural network is used to predict how much workload must be moved to achieve the desired temperature balance. At the same time, a preemptive fan control mechanism reduces power, and temperature peaks significantly compared to reactive (regular) fan control. Our thermal-aware load balancing mechanism can: (1) reduce peak temperature, and thermal wear, (2) increase reliability, (3) achieve efficient cooling, (4) accurately control core temperature, and (5) prevent power peaks caused by reactive fan control.

We use the Charm++ runtime system to implement our

978-1-5090-3856-5/16 $31.00 © 2016 IEEE
DOI 10.1109/E2SC.2016.6

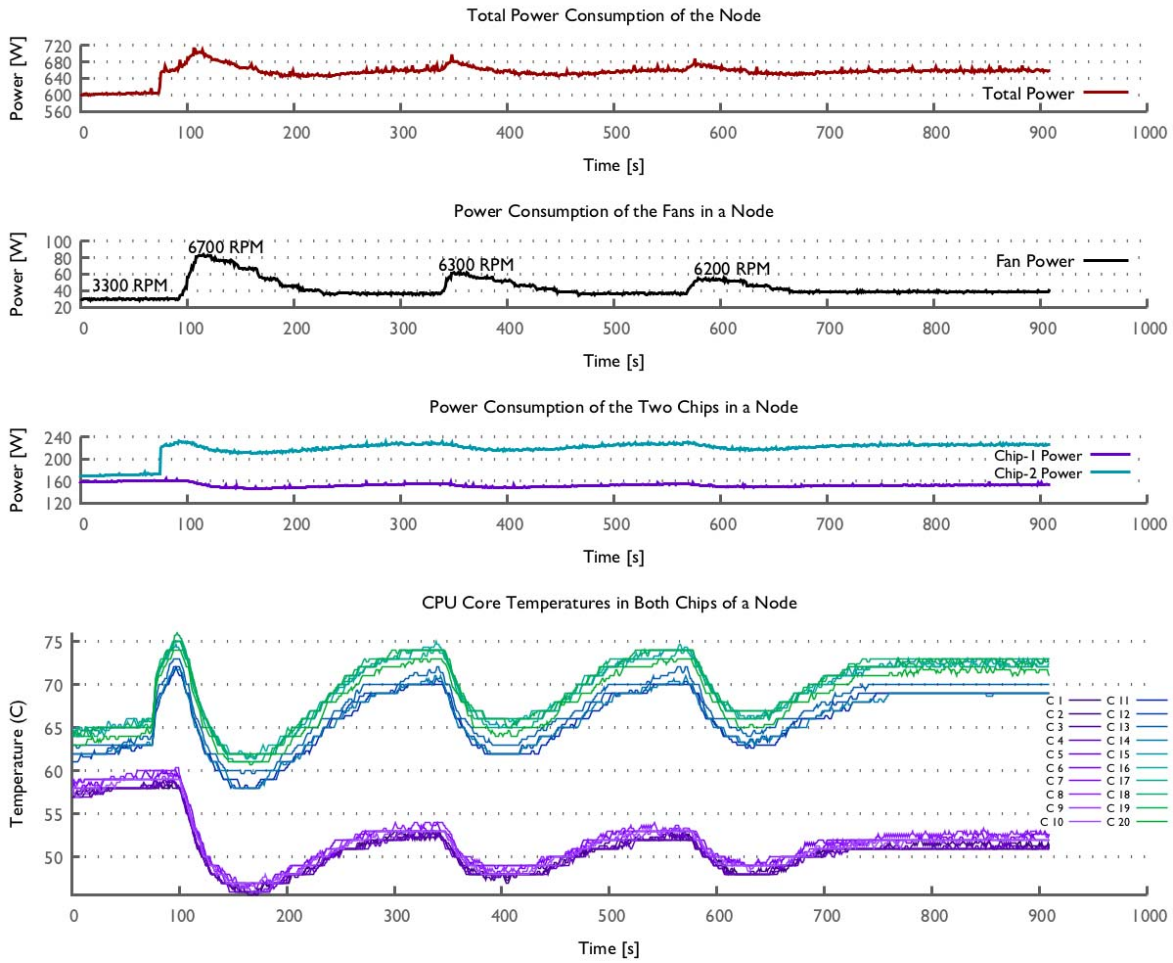

Fig. 2: Core temperatures imbalance, and power consumption peaks observed in different nodes.

load balancing algorithm. CHARM++ 's task-migration-based load balancing framework provides a database to store processor load, power, and temperature information [1]. We believe that task-based parallel programming models are a good fit for optimizing systems with variations, as they provide the adaptability, information, and application control that are necessary for making optimal decisions at fine granularity.

The contributions of this paper are as follows:

- A simple, and powerful learning-based approach to accurately predicting core temperatures (Section IV-B).
- Validation, and evaluation of our proposed learning-based approach (Section IV-B).
- A preemptive fan control mechanism that can reduce the peak power by 58% i.e. 54 Watts (Section IV-C).
- A novel thermal-aware load balancing algorithm to reduce cooling system power consumption by balancing core temperatures (Section IV-D). This algorithm uses the learning-based approach to predict temperatures, and the preemptive fan control mechanism to achieve further reductions in fan power by 61%. We show that temperature variations can be reduced from 9° C to 2° C.

II. MOTIVATION AND OBSERVATIONS

Thermal hot spots can cause cooling inefficiencies, component failures, and ultimately system outages. In an ideal environment, components would have uniform temperatures because even one hot core can cause various levels of the cooling system, starting with the node fans, to be activated. Of course, activating a fan increases power consumption of the system. Because CPU cores tend to be the highest-temperature components in a busy node, much of our focus is on core temperatures.

Figure 2 shows an example of temperature imbalance among the cores in a node, and how the power consumption of the node, fans, and chips changes over time. An artificial workload generator utilizes 99% of all Chip-2 cores while Chip-1 is idle. Key observations from this figure are:

- Even when idle, the temperatures of the Chip-1 cores (C1-C10), and the Chip-2 cores (C11-C20) differ by around 5° C. There is also an up to 5° C variation among the cores within Chip-2.
- When Chip-2 temperatures reach the fan threshold temperature of 73° C at 95 seconds, the fan speed increases

from 3300 RPM to 6700 RPM, causing a 54 W increase in power consumption.
- High temperatures in Chip-2 cause all 4 fans in the node to activate, resulting in Chip-1 core temperatures dropping unnecessarily (to below idle temperature).
- Fans show an oscillatory behavior in speed, and power consumption before finally stabilizing after 10 minutes.
- Power consumption of the chips is mostly stable because the workload is stable. The slight change in power that accompanies the change in core temperatures is not significant.

These observations point to two opportunities to reduce cooling costs. First, temperature variation can be reduced by a thermal-aware load balancing technique. Second, power peaks can be eliminated by a preemptive fan control mechanism. We analyze these two ideas in Section IV.

III. Related Work

Recent research supports the existence of thermal, and manufacturing variations in supercomputing systems [2], [12], [24]. Thermal-aware job scheduling strategies have been proposed to cope with this problem. However, because variations can be observed even within one node, workload scheduling strategies would not be sufficient to address the temperature variations, especially for large-scale HPC applications. Moreover, HPC applications may have communication patterns that let them benefit from topology-aware mapping strategies [6]. Job topology requirements can conflict with thermal-aware job scheduling techniques. Therefore, we propose a runtime-based technique for thermal-aware task scheduling that is less intrusive than job-level scheduling, allowing the job to continue to specify topology constraints. Only a small fraction of the tasks will be mapped to a location other than their 'best' location.

Past research in thermal-aware load balancing studied reactive techniques, where the runtime system constantly monitors core temperatures, and makes decisions based on the readings. For example, Sarood et al. proposed a temperature-restraining load balancing algorithm [14], [21]. In this work, the runtime system applies DVFS to the processors that are exceeding a threshold temperature. However, because each processor can potentially have a different frequency level, a load imbalance could be created among processors. Then, the runtime system applies load balancing to compensate for the disparate frequencies. This approach requires the system to monitor temperatures frequently, and apply DVFS empirically when temperatures exceed a threshold. The frequency setting are found by trial-error. Another limitation of the work is that the object migrations are done without taking into account the temperature changes of the host and donor processors after load balancing. This will again result in repetitive need of frequent temperature tracking and frequency control. In our approach, we use neural networks to guide load balancing decisions. A neural network can precisely predict core temperatures for given workloads, and for different workload-to-processor mappings. Therefore, it eliminates the need of monitoring core temperatures frequently, and replaces empirical decision making with precise model-based decisions. The model can predict how the temperatures of the cores are going to be after doing load balancing. Another difference of our approach is that, our load balancer aims to balance the core temperatures, it does not try to bring the core temperatures under a certain temperature threshold.

The neural network approach has been used in data center resource management. Moore et al. used neural networks to build an online thermal mapping, and management system for data centers [16]. This approach is coarse-grained, and cannot predict core temperatures. Duy et al. used neural networks to predict user demand in order to turn servers on or off in their "green" scheduling algorithm for cloud data centers [9]. Neural networks have also been used to improve job scheduling decisions by predicting resource status (i.e. the load of the data center based on history information) [10]. Another study uses neural networks to predict the core, and Network-on-Chip component temperatures of the chip, and the predictions are used for energy efficient data exchange [5]. However, they heavily relied on the simulated data to evaluate their model, and their neural network models are not thoroughly validated in the literature. Other machine learning methods have also been used to predict data center temperatures [24]. However, neural networks have shown many advantages over statistical methods, due to their capability in handling nonlinear behavior. Neural networks have been applied to many other areas, such as stock markets, pattern recognition, data mining, and medical diagnosis.

A forward-looking fan control mechanism, where the system uses system utilization information to predict temperatures, and controls the fan preemptively to dampen power peaks, has been patented [23]. However, the effectiveness of the mechanism has not been reported, and is an open research question. Our approach is similar to this idea, but we provide evaluations from our initial experiments.

IV. Proposed Solution

In this section, we describe our experimental set up (Section IV-A), our neural network, and its validation (Section IV-B), our preemptive fan control mechanism (Section IV-C), and finally our thermal-aware load balancing algorithm (Section IV-D).

A. Experimental Setup

For our experiments, we used an IBM cluster with POWER8 processors running at 3.5 GHz. Each node in the cluster has two sockets, and each socket has 10 physical cores, and 80 SMT cores. The On-Chip Controller (OCC) provides temperature and power data. The Baseboard Management Controller (BMC) provides fan speed data, and fan speed control. Each node has 4 cooling fans that operate synchronously. The ambient temperature in the cluster room is 20° C.

The MATLAB Neural Network Toolbox [8] is used for training. Then, the neural network parameters such as weights,

biases, and min/max values are extracted, and transferred into parameters stored in the CHARM++ runtime system.

Benchmarks: *lookbusy* is a synthetic load generator for Linux systems [7]. We use *lookbusy* to train our neural network model by generating different combinations of CPU loads. We use *Stencil3D* for our HPC workload. *Stencil3D* is a 7-point stencil application based on a 3D grid.

B. Neural Network Model

A neural network is inspired by biological neural networks to predict or approximate functions that can depend on a large number of inputs. There are two phases in using a neural network: training, and deployment. During the training phase, neurons in each layer are adjusted iteratively using training data. Then the trained neurons are used to predict the new output in the deployment phase. Figure 1 shows our neural network design. First, input data is pre-processed because training data selection, and specific input variables have a large impact on the overall accuracy of the temperature prediction. We collected data only after the temperature stabilized because we need to predict the stable core temperatures using this model

Pre-processed input data is fed to the input layer of the neural network. The input layer consists of core utilization rate, and fan speed. The output layer consists of core temperatures. After the training phase, the neural network is deployed to provide core temperature predictions. The information flow of the training phase, and the deployment phase is shown in Figure 1 as the solid blue line, and dotted red line, respectively.

In a neural network each connection between neighboring layers has a weight to scale data, and a bias that allows shifting of the activation function. Data (with 10 data points) from the input layer are inserted as inputs to the next consecutive layers (hidden layers). Then, hidden layers (each with 10 nodes) sum the data fed to them, and scale (weight) the data, and process it until the data reaches the last layer (output layer with 10 data points). We selected 10 data points because there are 10 temperature sensors per socket, and the temperature prediction is per socket. The weights, and biases of one neural network are updated as follows in the training phase:

$$w_{ij}(k+1) = w_{ij}(k) - \eta \frac{\delta e_k}{\delta w_{ij}} \quad (1)$$

η is the learning rate parameter, which determines the rate of learning. w_{ij} represents the scalar value of weight on the connection from layer i to j. e_k represents the error of NN at the kth iteration. $\delta e_k / \delta w_{ij}$ determines the weighted search direction for this iterative method.

The weights, and biases of the network are updated only after the entire training set has been applied to the network. The gradients calculated for each training set are added together to determine the change in weights, and biases. The weights, and biases are updated in the direction of the negative gradient of the performance function. We tried different back-propagation algorithms in our neural network. *Levenberg-Marquardt* updates weight, and bias values according to the Levenberg-Marquardt optimization [17]. It is often the fastest back-propagation algorithm, but it requires more memory than other algorithms. *Bayesian regularization* also updates the weight, and bias values according to the Levenberg-Marquardt optimization. However, it minimizes a combination of squared errors, and weights, then determines the correct combination to produce a network that generalizes well. *Scaled conjugate gradient* updates weight, and bias values according to the scaled conjugate gradient method [15]. *Resilient* updates weight, and bias values according to the resilient back-propagation algorithm [20].

Model Validation: Figure. 3 shows the average prediction error ($|T_{real} - T_{est}|$), which is a good indicator of predictor performance. Estimation error tends to decrease with the number of training samples. Our neural network becomes more knowledgeable on the relationship between temperature changes, and resource allocation as it processes more training samples.

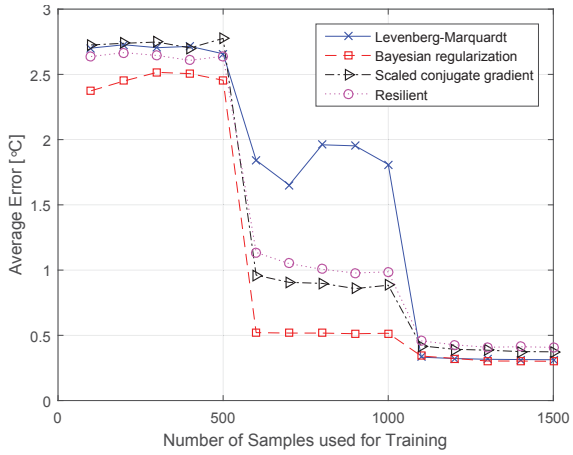

Fig. 3: Average error in temperature with different neural network back propagation algorithms.

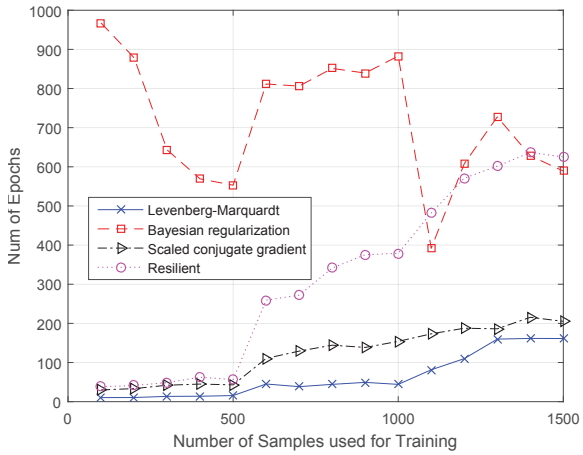

Fig. 4: Elapsed time with different neural network back propagation algorithms.

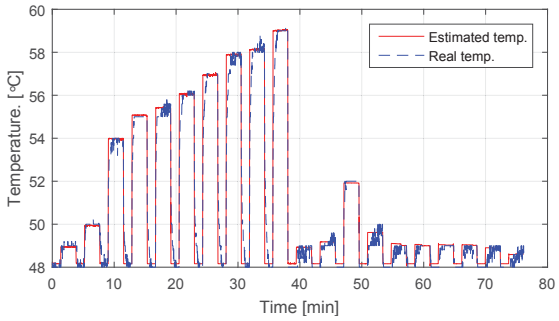

Fig. 5: Real temperature, and the predicted temperature of a core.

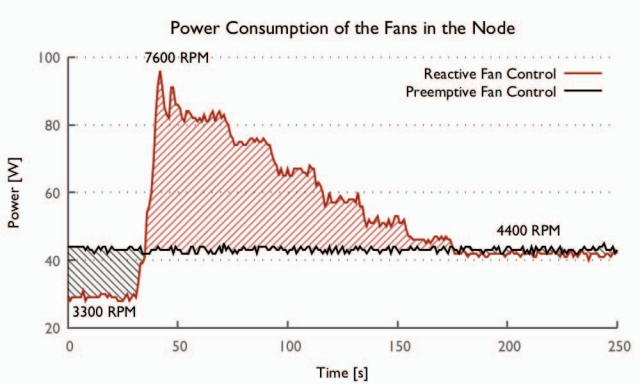

Fig. 6: Preemptive fan control reduces the peak power by 58%, and saves energy (2790 J).

The Bayesian regularization algorithm performs best in terms of the accuracy (showing minimum average error), but has more computational overhead (showing higher execution time for training). The time elapsed for training is shown in the Figure 4. Unfortunately, it is difficult to conclude which algorithm is best overall because each does well on different criteria. Also, data selection becomes important because some algorithms are sensitive to the training data, and readily over-fitted. We employed the *Bayesian regularization* back-propagation algorithm because it performed best when using a smaller number of samples. However, the *Levenberg-Marquardt*, *Scaled conjugate gradient*, or *Resilient* algorithms can be used in case learning overhead is a hard constraint.

We verified the accuracy of neural network approach by repeating the experiment, and comparing the predicted core temperature over time with the actual core temperature as shown in Figure 5. Our neural network is able to make accurate predictions despite slight hardware differences, variable heat, and air circulation patterns (thermodynamic phenomena) of different regions inside a data center.

C. Preemptive Fan Control

The cooling fans in a node react to processor temperature changes by changing their speed. However, this causes spikes (or peaks) in their power consumption. In this section, we show how a preemptive fan control mechanism can remove the power spikes, and save energy without any performance overhead to applications.

Figure 6 shows the power behavior of the fans in two different scenarios: reactive fan control, and preemptive fan control. With reactive fan control, which is the default fan control mode, the triggering of the fan causes a power peak. When a core temperature hits the threshold of 73° C, the fan speed increases rapidly to be able to control the temperature quickly, and decreases slowly (with steps of 400 RPM) as the temperature falls. When both chips are idle, the fans operate at 3,300 RPM. This idle speed is determined by the ambient temperature. When a workload starts running, the temperature of the cores starts to increase, followed by an increase in the fan speed up to 7,600 RPM. Within ten seconds, the fan power jumps from 30 W to 96 W. Then as the processor cools down, the fan speed slowly decreases, and stabilizes at 4,400 RPM. On the other hand, with our preemptive fan control, the stable fan speed of the application is predicted before the application starts. Preemptive fan control not only removes the power peaks by 58% but also reduces energy consumption.

The removal of power peaks makes total power consumption more stable, and predictable. This is important for power-aware resource management in power-constrained systems. In such systems, each application is allocated a strict power limit [18]. Moreover, it has been shown that a power-cap which is lower than what an application consumes without a power-cap can cause severe performance degradation [19]. Avoiding power peaks gives applications a better chance of staying within strict power limits, and avoiding severe performance degradation.

Preemptive fan control removes not only power peaks but temperature peaks as well. Figure 7 shows core temperatures with regular fan control, and with preemptive fan control. Regular fan control creates temperature peaks, whereas preemptive fan control reduces these peaks significantly. With preemptive fan control, the core temperatures slowly increase, and stabilize at the same level as with the regular fan control mechanism.

The optimal fan speed of an application can be determined simply by its profile. We leave finding an optimal fan speed for different application profiles as a future work. Once the optimal fan speed is determined, either the resource manager or runtime system can apply the appropriate fan speed. We do not suggest taking over control of fan speed from the hardware entirely. Our proposed mechanism can work together with the hardware in determining the fan speed. For example, the emergency shut-down mechanism when core temperatures hit a certain limit is still necessary.

An important preemptive fan control decision is exactly when to set the fan speed. In our experiment in Figure 7, the fan speed was set well before the application starts. If it was set later, the temperatures may not have been stabilized at the same level. As future work, we will study the optimal time to set the fan speed.

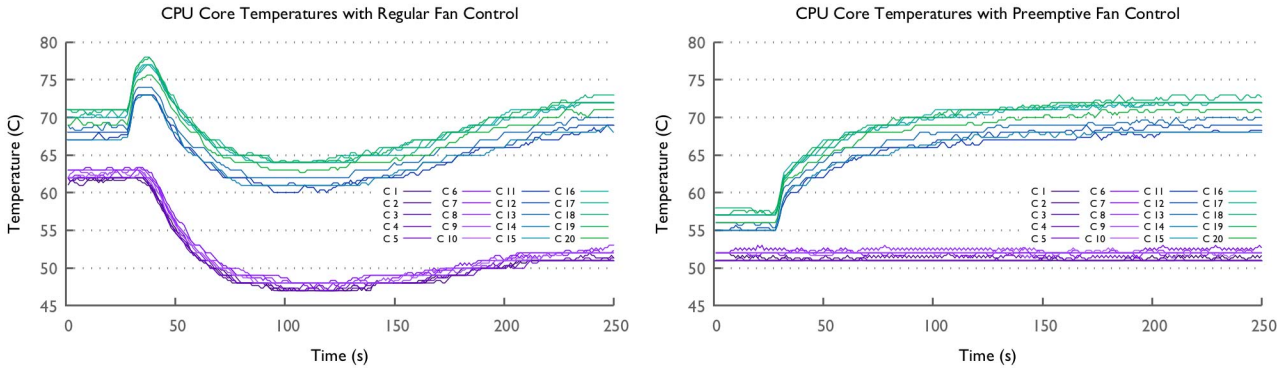

Fig. 7: Left plot shows regular fan control. Right plot shows preemptive fan control. Preemptive fan control removes the power peaks, and still keeps the temperature at the same level.

D. Thermal-Aware Load Balancing

We propose a new load balancing strategy based on the neural network model we presented in Section IV-B, and the preemptive fan control mechanism in Section IV-C. We first discuss the system we use for doing load balancing.

CHARM++ and Load Balancing: CHARM++ is a task-based parallel programming system [1]. It is supported by an adaptive runtime system, which enhances user productivity, and allows programs to run port ably on systems ranging from small multicore computers to the largest supercomputers. CHARM++ has three main attributes: over-decomposition, asynchronous message-driven execution, and migratability. Over-decomposition means having the programmer divide the computation into small work, and data units, which are C++ objects, so that there are many more such units than the number of processors. Message-driven execution involves scheduling work units based on when a message for them is received. Migratability refers to the ability to move data, and work units between processors. These attributes enable the CHARM++ adaptive runtime system to provide many useful features, one of which is dynamic load balancing. CHARM++ uses a measurement-based mechanism for load balancing. It collects information about the application, and the system in a distributed database. The information includes processors loads, loads of each object, communication patterns, and core temperatures. This approach has the advantage that it provides an automatic, application-independent way of obtaining load statistics without any input from the user. This information is used by different modules of the adaptive runtime to make decisions about improving load balance, handling faults, and enforcing power constraints.

We implemented a new load balancing algorithm using CHARM++ 's load balancing framework. Our algorithm uses a greedy approach where the runtime system moves objects from high-temperature (above average) cores to low-temperature (below average) cores in order to create temperature balance. The neural network predicts core temperatures of potential object migration, and allows us to determine the best workload

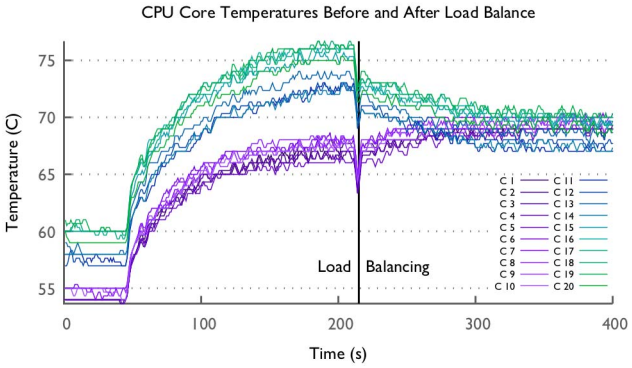

Fig. 8: CPU core temperature variations almost disappear after load balancing.

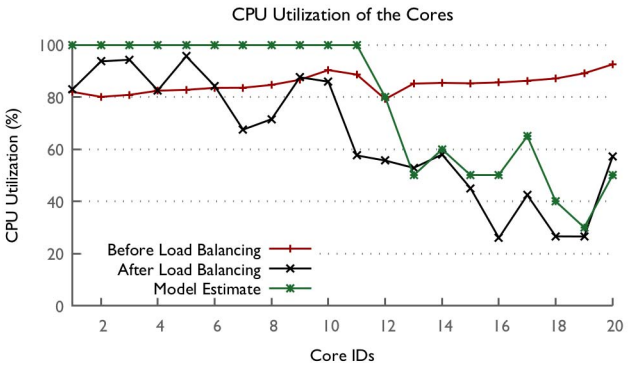

Fig. 9: Desired CPU utilization levels which keep temperature balanced.

distribution. Because the neural network uses CPU utilization as input, we need to approximate the load of each task in terms of CPU utilization. We do this by calculating the ratio of a task's execution time, and the total execution time of the tasks on the same processing core from historical information stored in CHARM++ 's runtime database. This approximation works well for CPU-intensive applications because not much time

is spent on communication or memory operations where the CPU is idle. In the future, we will improve this approximation for a more diverse set of applications.

Figure 8 shows the core temperatures before, and after load balancing when running the *Stencil3D* application. Load balancing, which is triggered around 200 seconds, reduces the core-to-core temperature variations within the node from 9° C to 2° C. 7% of the workload (947 objects out of 12,000) was migrated to achieve this balance. Note that load balancing is done after the temperatures stabilize to show the temperature difference clearly (i.e. for visual purposes), it can also be done from the start of the application, and if the application has different stages with different temperature profiles, it can be detected by the runtime, and load balancing can be triggered again. In this experiment, the fan speed was preemptively set to 3,900 RPM, which is 500 RPM lower than the stable fan speed without load balancing. With load balancing the peak fan power can be reduced by an additional 5 Watts to 61%. Furthermore, Figure 9 shows CPU utilization before load balancing, CPU utilization after load balancing, and the CPU utilization predicted by the neural network. Ideally, the neural network prediction, and the actual CPU levels after load balancing should match. However there are small errors. For example, the neural network predicts cores 1-10 to have 100% utilization but the application is not able to achieve that level on those cores even though they do have the additional load. This error is likely due to object communication dependencies, and increased memory access load of the corresponding cores. Changing the object location likely changes the communication load as well. Since the *Stencil3D* application has neighbor-to-neighbor communication, this is an important factor in performance. As average utilization of all the cores is reduced by 20%, we see a 20% slowdown in iteration time of the application as well. Theoretically, this slowdown should only be by 7%, if the actual utilization levels matched the estimated neural network utilization levels. Not being able to estimate post load balancing CPU utilization levels well enough could be a reason of this discrepancy. Currently, the CPU load estimation of each object is done by dividing the total CPU utilization by the total number of objects in the core. However, that level may not remain the same when the object is moved to a different core. We are going to work on improving this in the future. Furthermore, at larger scale, this slowdown is likely to be less since a large number of processors will have temperatures closer to average, requiring less movement to achieve temperature balance.

V. Conclusions & Future Work

We have proposed a solution to address two problems we identified in supercomputing systems: temperature variations, and power peaks caused by cooling fans. For the first problem, we propose a load balancing mechanism based on a neural network temperature prediction model. For the second problem, we propose a preemptive fan control mechanism. We also present preliminary results. Future work includes more extensive evaluations of the two mechanisms, extending the neural network to consider fan speed, frequency, and more detailed workload profiling, finding optimal fan speeds based on application profiles, and determining the time necessary to apply preemptive fan speeds.

References

[1] B. Acun, A. Gupta, N. Jain, A. Langer, H. Menon, E. Mikida, X. Ni, M. Robson, Y. Sun, E. Totoni, et al. Parallel programming with migratable objects: charm++ in practice. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 647–658. IEEE, 2014.

[2] B. Acun, P. Miller, and L. V. Kale. Variation among processors under turbo boost in hpc systems. In *Proceedings of the 2016 International Conference on Supercomputing*, page 6. ACM, 2016.

[3] ASCAC Subcommittee. Top ten exascale research challenges. *US Department Of Energy Report*, 2014.

[4] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, et al. The opportunities and challenges of exascale computing. *Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee*, pages 1–77, 2010.

[5] S. Aswath Narayana. An artificial neural networks based temperature prediction framework for network-on-chip based multicore platform. Master's thesis, Rochester Institute of Technology, 2016.

[6] A. Bhatele. *Automating Topology Aware Mapping for Supercomputers*. PhD thesis, Dept. of Computer Science, University of Illinois, August 2010. http://hdl.handle.net/2142/16578.

[7] D. Carraway. lookbusy – a synthetic load generator. https://www.devin.com/lookbusy/.

[8] H. Demuth and M. Beale. Neural Network Toolbox for Use with MATLAB. http://www.mathworks.com/help/nnet/.

[9] T. V. T. Duy, Y. Sato, and Y. Inoguchi. Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010.

[10] J. Huang, H. Jin, X. Xie, and Q. Zhang. Using narx neural network based load prediction to improve scheduling decision in grid environments. In *Third International Conference on Natural Computation (ICNC 2007)*, volume 5, pages 718–724. IEEE, 2007.

[11] W. Huang, C. Lefurgy, W. Kuk, A. Buyuktosunoglu, M. Floyd, K. Rajamani, M. Allen-Ware, and B. Brock. Accurate fine-grained processor power proxies. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 224–234, 2012.

[12] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda, et al. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 78. ACM, 2015.

[13] E. K. Lee, H. Viswanathan, and D. Pompili. Vmap: Proactive thermal-aware virtual machine allocation in hpc cloud datacenters. In *High Performance Computing (HiPC), 2012 19th International Conference on*, pages 1–10, 2012.

[14] H. Menon, B. Acun, S. G. De Gonzalo, O. Sarood, and L. Kalé. Thermal aware automated load balancing for hpc applications. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–8. IEEE, 2013.

[15] M. F. Moller. A scaled conjugate gradient algorithm for fast supervised learning. *NEURAL NETWORKS*, 6(4):525–533, 1993.

[16] J. Moore, J. S. Chase, and P. Ranganathan. Weatherman: Automated, online and predictive thermal mapping and management for data centers. In *2006 IEEE International Conference on Autonomic Computing*, pages 155–164. IEEE, 2006.

[17] J. J. More. The levenberg-marquardt algorithm: Implementation and theory. *Numerical Analysis, ed. G. A. Watson, Lecture Notes in Mathematics 630, Springer Verlag*, pages 105–116, 1977.

[18] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. L. Rountree, M. Schulz, and B. R. de Supinski. Practical resource management in power-constrained, high performance computing. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 121–132. ACM, 2015.

[19] K. Pedretti, S. L. Olivier, K. B. Ferreira, G. Shipman, and W. Shu. Early experiences with node-level power capping on the cray xc40 platform. In *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing*, page 1. ACM, 2015.

[20] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591, 1993.

[21] O. Sarood and L. V. Kale. A 'cool' load balancer for parallel applications. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 21. ACM, 2011.

[22] R. Sawyer. Calculating total power requirements for data centers. *White Paper, American Power Conversion*, 2004.

[23] D. Tobias. Forward-looking fan control using system operation information, Feb. 7 2006. US Patent 6,996,441.

[24] K. Zhang, S. Ogrenci-Memik, G. Memik, K. Yoshii, R. Sankaran, and P. Beckman. Minimizing thermal variation across system components. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 1139–1148. IEEE, 2015.