November 2015

On thermal sensor calibration and software techniques for many-core thermal management

Shiting Lu
University of Massachusetts - Amherst

ON THERMAL SENSOR CALIBRATION AND SOFTWARE TECHNIQUES FOR MANY-CORE THERMAL MANAGEMENT

A Dissertation Presented

by

SHITING LU

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2015

Electrical and Computer Engineering

ON THERMAL SENSOR CALIBRATION AND SOFTWARE TECHNIQUES FOR MANY-CORE THERMAL MANAGEMENT

A Dissertation Presented

by

SHITING LU

Approved as to style and content by:

Russell Tessier, Chair

Wayne Burleson, Member

Csaba Andras Moritz, Member

Farshid Hajir, Member

Christopher V. Hollot, Department Chair
Electrical and Computer Engineering

ACKNOWLEDGMENTS

I would like to thank my advisor Russell Tessier for his academic guidance and financial support. I sincerely appreciate his devotion on my research work and so many constructive suggestions which make this work possible. His enthusiasm on research has greatly affected me and will affect me in my future career. I also would like to thank Professor Wayne Burleson for his advices on my research. The discussions with him were always very helpful and enlightening and his valuable ideas and comments on my work are greatly appreciated. I also want to thank Professor Csaba Andras Moritz and Professor Farshid Hajir for serving as my PhD committee members, reading the dissertation and giving valuable comments.

I want to thank my friend Jia Zhao for recommending me to Prof. Tessier. We have known each other for many years and studied in the same labs both in China and in USA. I appreciate his help on both my research and personal life. I also want to thank Deepak Unnikrishnan for many useful discussions on various topics. It was a memorable experience to study, do research and have fun in Reconfigurable Computing Group for the past four and half years. Thank all former and current RCG members I spent time with. Thank all my friends in Umass Amherst without whom this journey would be much less colorful.

I want to thank my family for their strong and consistent support throughout all these years. Their love and encouragement helped me get over frustrations and negative moods. Special thanks go to my little nephew who sent me so many joyful messages. I wish him a very splendid future.

ABSTRACT

ON THERMAL SENSOR CALIBRATION AND SOFTWARE TECHNIQUES FOR MANY-CORE THERMAL MANAGEMENT

SEPTEMBER 2015

SHITING LU

B.Sc., HARBIN INSTITUTE OF TECHNOLOGY, HARBIN, CHINA

M.Sc., FUDAN UNIVERSITY, SHANGHAI, CHINA

M.Sc., ROYAL INSTITUTE OF TECHNOLOGY, STOCKHOLM, SWEDEN

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Russell Tessier

The high power density of a many-core processor results in increased temperature which negatively impacts system reliability and performance. Dynamic thermal management applies thermal-aware techniques at run time to avoid overheating using temperature information collected from on-chip thermal sensors. Temperature sensing and thermal control schemes are two critical technologies for successfully maintaining thermal safety. In this dissertation, on-line thermal sensor calibration schemes are developed to provide accurate temperature information. Software-based dynamic thermal management techniques are proposed using calibrated thermal sensors.

Due to process variation and silicon aging, on-chip thermal sensors require periodic calibration before use in DTM. However, the calibration cost for thermal sensors can

be prohibitively high as the number of on-chip sensors increases. Linear models which are suitable for on-line calculation are employed to estimate temperatures at multiple sensor locations using performance counters. The estimated temperature and the actual sensor thermal profile show a very high similarity with correlation coefficient ~ 0.9 for SPLASH2 and SPEC2000 benchmarks. Moreover, the proposed estimation model is capable of adapting to changing cooling conditions.

A calibration approach is proposed to combine potentially inaccurate temperature values obtained from two sources: thermal sensor readings and temperature estimations. A data fusion strategy based on Bayesian inference, which combines information from these two sources, is demonstrated. Our strategy is tested on two benchmarks suites: SPLASH-2 and SPEC2000. The result shows the strategy can effectively recalibrate sensor readings in response to inaccuracies caused by process variation and environmental noise. The average absolute error of the corrected sensor temperature readings is $< 1.5^oC$ and the standard deviation of error is less than $< 0.5^oC$ for tested benchmarks.

A dynamic task allocation strategy is proposed to address localized overheating in many-core systems due to both processor core and router power consumption. Our approach employs reinforcement learning, a dynamic machine learning algorithm that performs task allocation based on current temperatures and a prediction regarding which assignment will minimize the peak temperature. Experiments show that the proposed technique is capable of capturing the complex on-chip thermal environment induced by dynamic work load distribution. Our results show that the proposed technique is fast (scheduling performed in $< 1ms$) and can efficiently reduce peak temperature by up to 8^oC in a 49-core processor (6% on average) versus a leading competing task allocation approach for a series of SPLASH-2 benchmarks.

Reinforcement learning has also been applied to 3D integrated circuits to allocate tasks with thermal awareness. To avoid significant performance degradation and

computational overhead for large numbers of cores, a cluster based approach is used to apply reinforcement learning. Our results show that the proposed technique is fast (scheduling performed in < 0.2 ms) and can efficiently reduce peak temperature by $\sim 2^oC$ in average or up to 10^oC versus task scheduling without thermal awareness. It also reduces peak temperatures by $\sim 0.47^oC$ on average compared with a previous approach (balance-by-stack). Peak temperature reduction avoids $\sim 36\%$ of thermal emergencies which trigger performance throttling to alleviate thermal stress.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS . . . . . **iv**

ABSTRACT . . . . . **v**

LIST OF TABLES . . . . . **xii**

LIST OF FIGURES . . . . . **xiv**

CHAPTER

1. INTRODUCTION **1**

1.1 Motivation and Challenges 3
1.2 Contributions 5
1.3 Dissertation Organization 10

2. BACKGROUND **11**

2.1 Thermal Management in Contemporary Microprocessors 11
2.2 On-chip Thermal Sensors and Calibration Techniques 12

2.2.1 Heat and Read 14
2.2.2 Thermal Imaging 14
2.2.3 Design for Calibration 15
2.2.4 Thermal Estimation 16

2.3 Dynamic Thermal Management 19

2.3.1 Dynamic Voltage and Frequency Scaling 19
2.3.2 Thermal-aware task scheduling and allocation 20
2.3.3 Network-level Thermal Management 24
2.3.4 Thermal-Aware Workload Allocation in Data Centers 24

3. ON-CHIP THERMAL ESTIMATION VIA PERFORMANCE COUNTERS 26

3.1 Thermal Correlation With Performance Counters 26
3.2 Temperature Estimation Using Performance Counters 29

3.2.1 Linear Model for Absolute Temperature Estimation 30
3.2.2 Linear Model for Incremental Temperature Estimation 31

3.3 Linear Model Training 33

3.3.1 Model Training for Absolute Temperature Estimation 34
3.3.2 Model Training for Incremental Temperature Estimation 35

3.4 Infrastructure and Experimental Approach 36

3.4.1 Architectural Simulator 36
3.4.2 Thermal Simulator 38

3.5 Model Evaluation 38

3.5.1 Evaluation of Absolute Temperature Estimation 38
3.5.2 Evaluation of Incremental Temperature Estimation 40
3.5.3 Estimated Thermal Profile for Incremental Temperature Estimation 45
3.5.4 Temporal Evolution of Incremental Temperature Estimation 45

3.6 Principal Components of Performance Counter Vectors 47
3.7 Dynamic Model for Changing Cooling Conditions 49
3.8 Summary 51

4. MULTI-SENSOR COLLABORATIVE CALIBRATION 52

4.1 Dynamic Calibration Strategy: Approach Overview 52
4.2 Problem Formulation 54
4.3 MSCCA and Δ-MSCCA 56
4.4 Performance and Storage Evaluation 59

4.4.1 Computational Complexity 60
4.4.2 Memory Overhead 61

4.5 Implementation Issues 62
4.6 Experiments and Results 63

4.6.1 Methodology 64
4.6.2 Effectiveness Verification 65

4.6.3 Temperature Tracking Using Δ-MSCCA . . . 66
4.6.4 Estimation Error Comparison . . . 68
4.6.5 Impact of Sensor Reading Noise for Δ-MSCCA . . . 69
4.6.6 Run Time Comparison . . . 70

4.7 Summary . . . 71

5. THERMAL-AWARE TASK ALLOCATION BASED ON REINFORCEMENT LEARNING . . . 73

5.1 Thermal Profile of a 16-Core Processor including Data Traffic . . . 73
5.2 Thermal Aware Task Allocation Using Reinforcement Learning . . . 76

5.2.1 Reinforcement Learning . . . 76
5.2.2 Chip Thermal States . . . 78
5.2.3 Task Allocation Actions . . . 79
5.2.4 Thermal Reward . . . 79
5.2.5 Utility Function Approximation . . . 80
5.2.6 Basis Functions . . . 81

5.3 Implementation of RL-based Task Allocator . . . 82

5.3.1 Task Allocation Algorithm Description . . . 82
5.3.2 Memory and Computational Complexity . . . 83

5.4 Experimental Approach . . . 84

5.4.1 Many-core Floorplan and Sensor Allocation . . . 84
5.4.2 Synthetic Workload . . . 85
5.4.3 Simulation Flow . . . 87

5.5 Results . . . 88

5.5.1 Effectiveness Validation . . . 88
5.5.2 Peak Temperature Reduction . . . 90
5.5.3 Memory and Computational Complexity . . . 91

5.6 Summary . . . 95

6. THERMAL AWARE TASK ALLOCATION FOR 3D MANY-CORES . . . 96

6.1 Thermal Behavior in 3D Integrated Circuits . . . 97
6.2 Revisiting Reinforcement Learning . . . 99

6.2.1 3D Many-Core Model . . . 101
6.2.2 A New Definition of Thermal Reward . . . 102

6.3 Cluster-based Reinforcement Learning for 3D ICs 103

6.3.1 Reinforcement Learning on Clusters 104
6.3.2 Communication Analysis 105

6.4 Experimental Setup 108

6.4.1 3D Floorplan and Thermal Simulation 108
6.4.2 HotSpot Integration 108
6.4.3 Benchmark Workload 109

6.5 Results 111

6.5.1 Peak Temperature Reduction 112
6.5.2 Reduction of Thermal Emergencies 113
6.5.3 Computational and Memory Overhead 116

6.6 Summary 118

7. CONCLUSION AND FUTURE WORK 119

BIBLIOGRAPHY 123

LIST OF TABLES

Table **Page**

3.1 Performance Counters Provided by SESC 27

3.2 Average estimation error for each sensor over all benchmarks 42

3.3 Average estimation error for each benchmark over all sensors 43

3.4 Average absolute error of temperature estimation for all sensors and benchmarks if the number of performance counters is limited to specific quantities 47

4.1 Operations required by MSCCAs and Kalman filtering for p sets of sample readings for 24 thermal sensors 61

4.2 Benchmarks used in experimental evaluation 63

4.3 The standard deviation of the error for the corrected temperatures over 10,000 time instances for increasing sensor error 71

4.4 Run time comparison (in seconds) between MSCCA and KF approaches for **10000** time instances 71

5.1 Core Configuration 86

5.2 Router Configuration 86

5.3 Peak temperature comparison between reinforcement learning (ours) and adaptive random [24]. The value ρ indicates the average number of cores used during execution. Percentage peak temperature reductions are shown. 92

5.4 Time Cost For RL Task Allocation (ms) 94

5.5 Memory Cost For a 16-Core System (KB) 94

6.1 Four clusters in a 5x5 system 105

6.2 Average number of executing tasks for a collection of system configurations 111

6.3 Empirically determined parameters in used experiments. 111

6.4 Time Cost For RL Task Allocation (ms) 117

6.5 Memory Cost for Different Core Numbers (KB)) 118

LIST OF FIGURES

Figure **Page**

1.1 Trend of IC power density with ITRS projection (Trend 1) and technology innovation (Trend 2) [54] 1

1.2 POWER7 chip floorplan with 44 digital thermal sensors (DTS) [34] 2

1.3 12 thermal sensors per core in a 4-core SandyBridge processor [73] 3

2.1 Two implementations of digital thermal sensors 14

2.2 Co-located thermal and process sensors for calibration purpose [28] 15

2.3 Kalman Filter Approach [91] (1) off-line model calibration (2) on-line estimation 18

2.4 Thermal-aware scheduling for a task graph [51]. The tasks are labeled as bubbles in the DAG. Schedule time is shown on the vertical axis on the right. 20

2.5 Different task distribution in a many-core [110] [37] 21

3.1 Correlation between temperature and some system statistics for the *radix* benchmark across 24 thermal sensors 27

3.2 Runtime recording of some system statistics and temperature change rates for three function units: integer scheduler (left column), floating point scheduler (middle column) and L1 data cache (right column) for the *equake* benchmark. Temperature changes are in ^oC 28

3.3 Runtime recording of thermal gradient and temperature change rates for three function units. Temperature changes are in ^oC 29

3.4 Floorplan of the Athlon 64 processor [71] 39

3.5 The correlation between estimated and actual temperature profiles 40

3.6 The estimated and actual processor temperature profile at one time instance for four SPLASH2 benchmarks. Each of the 24 thermal sensors in the processor are represented on the horizontal axis for the time instance. 41

3.7 The estimated and actual processor temperature profile at four time instances for the SPLASH-2 *ocean* benchmark. 44

3.8 Temperature estimation over 6 seconds for sensor 8 (integer scheduler), 20 (floating point scheduler) and 12 (ALU). The data is collected through simulation by running *applu* on the AMD floorplan. 46

3.9 The correlation between estimated and actual temperature profiles for various benchmarks. 46

3.10 Eigenvalues calculated for 34 principal components from 34 performance counters collected using CINT2000 benchmarks and SESC simulator. 48

3.11 Prediction accuracy comparison for different numbers of principal components used in the model. 49

3.12 (a) One β model using cubic fitting for integer module (block 8 in Fig. 3.4) . (b) One β model using cubic fitting for ALU module (block 12 in Fig. 3.4). 50

4.1 Our dynamic calibration scheme for on-chip thermal sensors 53

4.2 Sensor data and thermal estimation merging scheme. Sensor readings are artificially generated shown in the left flow; Estimated temperature from performance counter are obtained by the right flow. 64

4.3 The thermal profile of the processor for one time instance of the *lu* benchmark. Each point on the horizontal axis represents a single sensor located in a block in Figure 3. 65

4.4 The estimated and actual processor temperature profile at four time instances. 67

4.5 Temperature tracking over 6 seconds for thermal sensor 8 (integer scheduler), 15 (L1 data cache) and 20 (floating point scheduler) in Fig. 3.4 running the *twolf* benchmark. 68

4.6 Temperature tracking with various fan speeds running the benchmark *volrend* 69

4.7 (a) The average absolute error for sensor readings, temperatures generated with MSCCA, and with KF (b) The standard deviation of errors for sensor readings, temperatures generated with MSCCA, and KF approaches. 70

5.1 Thermal map for three different scenarios (a) all tasks running independently (b) tasks running on cores 14 and 16 communicate with each other (c) tasks running on cores 2 and 14 communicate with each other. 75

5.2 Reinforcement Learning Scheme for Thermal Aware Task allocation 78

5.3 Single block floorplan generated by HotFloorplan 84

5.4 Deployment of 9 thermal sensors in a 16-core device 85

5.5 The simulation flowchart 87

5.6 Convergence of θ_i values for a 25-core processor 88

5.7 Convergence of Q values for a fixed thermal condition 89

5.8 A thermal map snapshot for a 36-core 90

5.9 Q values for different actions at the thermal state given in Fig. 5.8. 91

5.10 Peak temperature comparison over time for $\rho = 0.4$ for reinforcement learning (RL) and adaptive random (AR) 93

5.11 Estimation results for RL if router temperature is considered or omitted 94

6.1 Typical 3D layout of an integrated circuit 98

6.2 Two tasks are allocated to neighboring cores on the same layer 99

6.3 Two tasks are allocated to neighboring cores on two layers 100

6.4 Thermal maps resulting from Fig. 6.2 assignment 100

6.5 Thermal maps resulting from Fig. 6.3 assignment 101

6.6 Peak temperatures for assignment schemes for thermally-steady states. 101

6.7 A three layer many-core system with 16 cores on each layer 102

6.8 T_{post} and T_{pre} peak temperature illustration in the task arrival and departure time line 103

6.9 Spatial thermal correlation 105

6.10 Clustering schemes for 25, 36, 49 and 64-core systems 106

6.11 Communication latency comparison. *Theoretical* indicates random assignment of tasks to cores. 107

6.12 A single allocation cycle implemented in HotSpot 110

6.13 Distributions of peak temperature reduction under various core count and system utilization configurations 114

6.14 Average peak temperature comparison for random, balance-by-statck [120] and RL approaches 114

6.15 The thermal profile for RL allocator after 5000 allocation 115

6.16 The thermal profile for balance-by-stack allocator after 5000 allocation 115

6.17 Peak temperature distribution comparison between random, balance_by_stack and RL approaches. 116

6.18 Thermal emergency incidents for various core counts and system utilization settings 117

CHAPTER 1

INTRODUCTION

Thermal management is a critical problem for modern microprocessors due to high transistor density. This characteristic increases power and heat density [70] [54] in a small silicon area causing performance degradation and decreased system reliability. Fig. 1.1 shows the increasing trends of IC power density. The temperature dependency of silicon reliability can be empirically modeled by the Arrhenius Equation [6]

$$MTTF = MTTF_0 \times exp\left(\frac{E_a}{kT}\right), \tag{1.1}$$

where $MTTF_0$, E_a and k are constants. The mean-time-to-failure ($MTTF$) decreases exponentially with the increasing of the temperature.

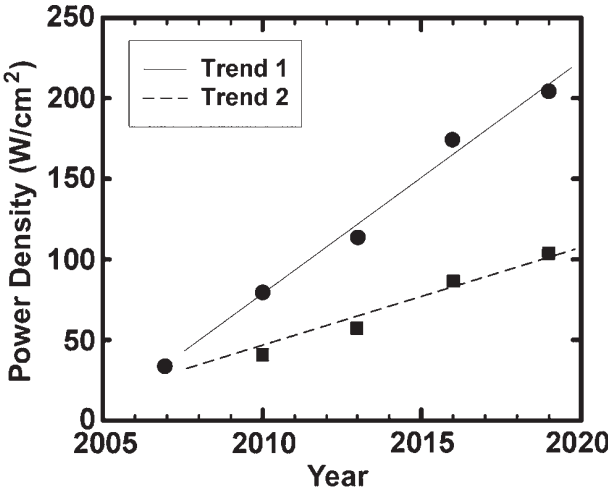


Figure 1.1: Trend of IC power density with ITRS projection (Trend 1) and technology innovation (Trend 2) [54]

The use of cooling technologies alone cannot meet system thermal design specifications and system-level thermal management techniques are necessary to alleviate chip thermal stress. As a result, performance as well as temperature become first order considerations for system design and run-time system management. Dynamic thermal and power management strategies are often employed to tackle run-time thermal and power issues [9][48] in order to achieve reliable, long-term system operation. The thermal gradient can be fairly large [70] within a processor core, so multiple on-chip thermal sensors at different positions are necessary to assist dynamic thermal management (DTM). For example, there are five thermal sensors per core implemented in the Power 7 EnergyScale infrastructure [35] [34], as shown in Fig. 1.2, and twelve sensors in each CPU core in the Intel 4-core Sandybridge processor [84] [73], as shown in Fig. 1.3. Recent trends indicate increased future use to assess thermal gradients and perform fine-grained thermal management with more on-chip thermal sensors.

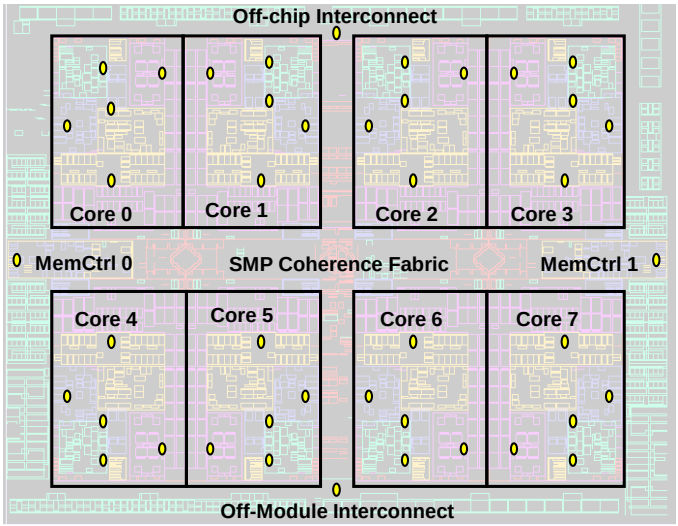


Figure 1.2: POWER7 chip floorplan with 44 digital thermal sensors (DTS) [34]

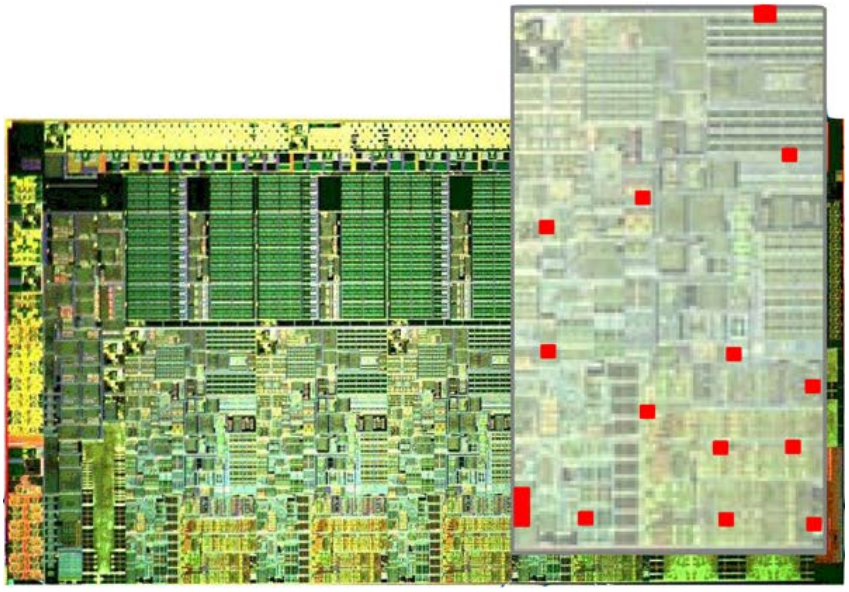


Figure 1.3: 12 thermal sensors per core in a 4-core SandyBridge processor [73]

1.1 Motivation and Challenges

The demands of high performance computing and the prevalence of distributed computing have forced processor architecture into the many-core realm. For example, Intel's exascale supercomputing CPU, Knights Landing, which will debut in 2015 on Intel's 14nm process, will have up to 72 Atom cores and 16GB 3D stacked DRAM. Temperature control techniques must be used dynamically in an aggressive way to maximize performance. These techniques heavily rely on thermal sensors embedded in the processor core and other locations on the die. Therefore, multiple technologies must be considered. First, one must consider the design and deployment of on-chip thermal sensors. This issue requires that thermal sensors be implemented with ultra low cost considering the proliferation of thermal sensors as core count increases. Sensors should be properly positioned to capture temperature hot spots. Second, hardware and software techniques are needed to control temperatures within a safe range with minimized performance degradation.

The efficiency and effectiveness of a thermal management strategy relies on accurate thermal sensor measurements. However, low cost on-chip thermal sensors are sensitive to process variations and might report temperature values which deviate from actual ones. In some cases, the temperature reading error of *uncalibrated* thermal sensors can be substantial (up to 34^oC at 95^oC [82]) which adversely impacts DTM strategy.

Two main issues exist in using these sensors: (1) detecting if a sensor is providing erroneous readings [118] and (2) recalibrating the sensor, if necessary. Often, thermal sensor calibration involves performing thermal imaging using an infrared camera while capturing the physical readings of thermal sensors [39]. As the sensor count on a silicon die increases, the per-chip calibration cost can be prohibitively high, leading to on-chip thermal sensor use without individual sensor calibration. Even if thermal sensors are initially well-calibrated, their readings gradually drift away from actual temperature values due to device wear-out. Often, the degree of aging varies across the chip due to the activity variation of different subcircuits. Therefore, recalibration on thermal sensors is needed to regain the required accuracy. In general, it is not practical to perform in-field calibration with thermal imaging since end users typically do not have access to expensive calibration equipments.

Due to process variation and silicon aging, on-chip thermal sensors require periodic calibration [7]. On-line techniques are necessary to dynamically calibrate thermal sensors for DTM. However, the lack of knowledge of actual temperatures poses several challenges for correcting sensor readings. Thermal models can be used to predict actual temperatures from the average power dissipation of functional components [91]. However, since the power profile can vary for different applications or for different phases of one application, the use of average power dissipation to estimate temperature can potentially be inaccurate. Furthermore, on-chip heat flux causes spatial thermal correlations among different components which require extraction of ther-

mal parameters like heat resistance and heat capacitance, a non-trivial task. Finally, computational overhead is a major concern since calibration is performed on-line and can degrade overall system performance.

This dissertation also considers the design of thermal management schemes using information from calibrated on-chip thermal sensors. In contemporary many-cores, the power consumption of network-on-chip (NoC) routers, as well as processor cores, is a significant concern [90]. Many parallel and data intensive applications implemented on many-cores benefit from low latency and high bandwidth on-chip communication. Many-cores often require NoC routers with significant control circuitry and buffer storage, leading to substantial power consumption [99]. The heat dissipated by a router not only affects router temperature, but also the temperature of neighboring cores. Effective task scheduling or allocation for thermal management considers all many-core components, including NoCs. Effective task management for the numerous tasks dynamically assigned to cores is particularly important as the number of cores per chip scales.

There are several challenges associated with thermal-aware task scheduling and allocation. First, predictive thermal models are usually necessary to account for the thermal impacts of task allocation decisions. These thermal models are limited by their prediction accuracy and computational overhead. Second, task allocation is a global management decision which requires consideration of all cores, so the scalability of allocation algorithms is a major design concern. Finally, a trade-off between performance overhead and thermal benefits should be carefully evaluated for different system specifications.

1.2 Contributions

Several contributions have been made in this research to address the noted technology challenges. For thermal sensors, a collaborative calibration scheme is developed.

Thermal estimation is used during calibration to obtain accurate readings from low cost thermal sensors. For thermal management, an algorithm for allocating tasks to many-cores is devised to lower peak many-core temperature. Detailed contributions are summarized below.

- **Thermal Estimation and Sensor Calibration**

  (1) A fine-grained thermal estimation technique is developed and validated. Two linear models were built to estimate the steady and transient temperatures of architectural components. The steady state temperature is estimated by an absolute temperature estimation model [63] and the transient temperature is estimated by an incremental temperature estimation model [65]. To dynamically account for changing processor activities, collections of performance counter values are included to estimate the chip thermal profile at run time. A performance counter selection method is employed to reduce the intercorrelations between readings and improve estimation accuracy. Our results show that the correlation coefficient between estimated and actual thermal profiles is ~ 0.9 on a collection of benchmarks. The estimation model can be adapted to changing cooling conditions via parameter modeling.

  (2) Multiple sensors deployed in the processor are dynamically calibrated via the proposed Multi-Sensor Collaborative Calibration Algorithm (MSCCA) [63] and Δ-based Multi-Sensor Collaborative Calibration Algorithm (Δ-MSCCA) [65]. Our calibration approach combines potentially inaccurate temperature values obtained from two sources: temperature readings from thermal sensors and temperature estimations using system performance counters. A data fusion strategy based on Bayesian inference, which combines information from these two sources, is demonstrated along with a

temperature estimation approach using performance counters. The result shows that the strategy can effectively recalibrate sensor readings in response to inaccuracies caused by process variation and environmental noise. The average absolute error of the corrected sensor temperature readings is $< 1.5^oC$ and the standard deviation of error is less than $< 0.5^oC$ for tested benchmarks. The strategy incurs significantly reduced computational cost versus a previously-developed Kalman filtering technique [92] and is appropriate for on-line usage.

- **System Thermal Management**

  (3) A dynamic task allocation strategy is proposed to address localized overheating in many-core systems due to both processor core and router power consumption [64]. Our approach employs *reinforcement learning*, a dynamic, machine learning algorithm that performs task allocation based on current temperature and a prediction regarding which assignment will minimize maximum temperature. The algorithm updates prediction models after each allocation based on feedback regarding the accuracy of previous predictions. Our new algorithm is verified via detailed many-core simulation which includes on-chip routing. The experiments show that the proposed technique is capable of capturing the complex on-chip thermal environment induced by dynamic work load distribution. The results show that the proposed technique is fast (scheduling performed in $< 1ms$) and can efficiently reduce peak temperature by up to 8^oC in a 49-core processor (6% on average) versus a leading competing task allocation approach for a series of SPLASH-2 benchmarks.

  (4) Reinforcement learning is also applied to 3D integrated circuits to allocate tasks using the influence of thermal information. To avoid significant per-

formance degradation and computational overhead with a large core count, allocations using clusters of cores, rather than all cores, is considered. Our results show that the proposed technique based on reinforcement learning is fast (scheduling performed in < 0.2 ms) and can efficiently reduce peak temperature by $\sim 2^oC$ on average and up to 10^oC versus the approach without thermal awareness. It also reduces peak temperatures by $\sim 0.47^oC$ in average compared with a previous approach called balance-by-stack [120]. The peak temperature reduction can avoid $\sim 36\%$ thermal emergencies which trigger performance throttling in order to alleviate the thermal stress.

- **Publications**

1. Shiting Lu, Russell Tessir, Wayne Burleson, "Thermal-Aware Task Allocation for 3-D Many-Cores Using Reinforcement Learning", *ACM Transaction on Design Automation of Electronic Systems (TODAES)*, 22 pages, to be submitted.

2. Shiting Lu, Russell Tessier, Wayne Burleson,"Reinforcement Learning for Thermal-Aware Many-core Task Allocation," in *Proceedings of the 25th ACM Great Lakes Symposium on VLSI* (GLSVLSI'15), pp. 379-384. Pittsburgh, PA, May 2015.

3. Jia Zhao, Shiting Lu, Wayne Burleson and Russell Tessier, "A Broadcast-Enabled Sensing System for Embedded Multi-core Processors", *VLSI (ISVLSI), 2014 IEEE Computer Society Annual Symposium on*, pp. 190-195, Tampa, Florida, July 2014.

4. Shiting Lu, Russell Tessier and Wayne Burleson, "Dynamic On-Chip Thermal Sensor Calibration Using Performance Counters", *Computer-Aided*

Design of Integrated Circuits and Systems, IEEE Transaction on, vol. 33, no. 6, pp. 853-866, June 2014.

5. Shiting Lu, Paul Siqueira, Vishwas Vijayendra, Harikrishnan Chandrikakutty, and Russell Tessier, "Real-Time Differential Signal Phase Estimation for Space-based Systems Using FPGAs", in *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 2, pp. 1192-1209, April 2013.

6. Jia Zhao, Shiting Lu, Wayne Burleson and Russell Tessier, "Run-time Probabilistic Detection of Miscalibrated Thermal Sensors in Many-core Systems", in *Proceedings of the IEEE/ACM Design Automation and Test in Europe Conference (DATE)*, pp. 1395-1398, Grenoble, France, March 2013.

7. Shiting Lu, Russell Tessier, Wayne Burleson,"Collaborative calibration of on-chip thermal sensors using performance counters," *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on* , pp. 15-22, San Jose, CA, November 2012

8. Shiting Lu, Russell Tessier and Wayne Burleson. "On-Chip Thermal Sensor Collaborative Calibration Using Bayesian Estimation", *SRC TECHCON*, 4 pages, Austin, September, 2012.

9. Deepak Unnikrishnan, Shiting Lu, Lixin Gao, and Russell Tessier, "ReClick - A Modular Dataplane Design Framework for FPGA-Based Network Virtualization", in *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pp. 145-155, Brooklyn, NY, October 2011.

1.3 Dissertation Organization

The organization of this dissertation is as follows. Chapter 2 introduces on-chip digital sensor calibration techniques and software level thermal management techniques. Chapter 3 investigates the correlation between global performance counters and local thermal sensors and a linear thermal estimation model is presented. Chapter 4 describes and evaluates the Multi-Sensor Collaborative Calibration Algorithm (MSCCA) and Δ-MSCCA for dynamic on-chip thermal sensor calibration. Chapter 5 introduces a thermal-aware task allocation technique for many-cores using reinforcement learning. Chapter 6 presents thermal-aware task allocation in a 3D context. Chapter 7 summarizes the dissertation and discusses future research directions.

CHAPTER 2

BACKGROUND

In this chapter, state-of-the-art on-chip thermal sensors are briefly reviewed. Techniques to calibrate on-chip thermal sensors are then discussed. Dynamic thermal management techniques using on-chip thermal sensors are also summarized in this chapter.

2.1 Thermal Management in Contemporary Microprocessors

High temperatures result in reduced reliability, degraded performance and increased cooling cost, so temperature is a indispensable component of system management in modern processors. Significant effort has been devoted to exploring chip level techniques to prevent processor overheating. Thermal management techniques can be based on either hardware and software.

Hardware techniques include dynamic frequency and voltage scaling (DVFS), thermal-aware floorplanning, and thermal-aware wire routing. DVFS performs runtime frequency and voltage modifications to ensure thermal stability and required performance [14][3]. DVFS requires hardware to regulate chip-level voltage and clock generation. In contrast, thermal-aware floorplanning [75] [43] and thermal-aware wire routing [113] are statically applied to a chip at design time. These approaches use thermal simulation and analysis to disperse thermal hot spots, so that peak temperature and thermal gradients are reduced.

Software-based techniques include thermal-aware task scheduling [20], thermal-aware task allocation in many-cores [120], thermal-aware thread migration [117],

thermal-aware compilation [88] [86], and thermal-aware packet routing [79]. These software approaches improve thermal conditions by distributing workloads (tasks, threads and packets) temporally and/or spatially. Runtime techniques (dynamic task scheduling, allocation and thread migration) are implemented in the OS core and applied online for thermal optimization. Correspondingly, static software based techniques (compilation [86] and static task mapping [25]) are employed offline before application execution.

Since DVFS and dynamic thermal-aware task scheduling/allocation are performed at run time to accommodate a changing thermal environment, they can be considered part of dynamic thermal management (DTM). To effectively apply DTM, the temperature must be monitored. Detailed and accurate thermal information is needed to perform fine-grained thermal management. Typically, multiple on-chip thermal sensors are positioned at strategic locations in the processor to collect temperatures [112]. The number of used thermal sensors is often constrained by the silicon area required to implement them. Generally, the fabrication and design of digital systems demand that these on-chip sensors be implemented in digital logic to reduce design complexity and increase fabrication yield. Low cost design and process variation can introduce inaccuracies into thermal sensor readings, requiring the sensors to be carefully calibrated. In the following section, more information on thermal sensors and their calibration techniques is presented. Subsequently, DTM strategies are introduced and their advantages and limitations are discussed.

2.2 On-chip Thermal Sensors and Calibration Techniques

Digital thermal sensors (DTS) are typically implemented using delay chains whose signal propagation latency depends on temperature. One implementation style is based on a ring oscillator, as shown in Fig. 2.1a [97]. The circuit oscillates at the frequency (f) determined by the number of stages (N) and the temperature-

dependent propagation delay ($d(T)$) as given by (2.1). In a certain temperature range, (2.1) can be rewritten in a liner [116] or quadratic [108] form, as shown by (2.2) and (2.3), respectively.

$$f = \frac{1}{2 \times N \times d(T)} \tag{2.1}$$

$$T = w_0 + w_1 f \tag{2.2}$$

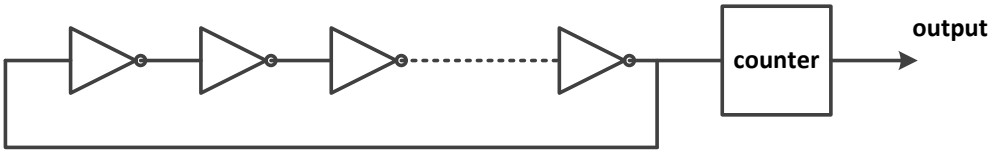
$$T = w_0 + w_1 f + w_2 f^2 \tag{2.3}$$

Here, w_0, w_1 and w_2 are calibration parameters. By measuring the frequency, f, via the counter which operates at a reference frequency, one can obtain the temperature via (2.2) and (2.3) if the sensor reading is calibrated properly.

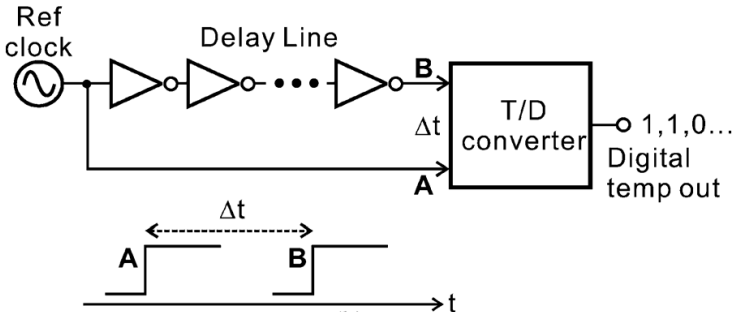
Another digital thermal sensor implementation utilizes the same physical principle but detects time difference instead of frequency, as shown in Fig. 2.1b. The time difference between signal A and B is determined by the propagation delay and the number of stages [103] [76]. The time/delay (T/D) converter measures Δt and gives a digital output.

$$\Delta t = N \times d(T) \tag{2.4}$$

Other sensor implementations, based on bandgap voltage references, require expensive analog circuitry [93] [94] [108]. Due to parameter drifts introduced by process variation, on-chip thermal sensor readings need proper calibration before use. The reading drift from a sensor can be significant. For example, one study showed a reading range from 61^oC to 109^oC for a true temperature of 95^oC [82]. There are several ways to calibrate on-chip thermal sensors to achieve better measurement accuracy. We divide these approaches into four main categories: heat and read, thermal imaging, design for calibration, and thermal estimation.



(a) Ring oscillator based thermal sensor



(b) Delay line based thermal sensor [38]

Figure 2.1: Two implementations of digital thermal sensors

2.2.1 Heat and Read

One straightforward but time-inefficient way to perform calibration is to heat the silicon die to a preset temperature and then read values from thermal sensors [68] [89] [107] [62]. The silicon components containing the thermal sensors are placed in a thermally-isolated container and the responses are read at fixed temperatures. Temperature sensors are calibrated using these measurements. The limitation of this method is that the thermal chamber requires time to reach thermal equilibrium and temperature variations must be kept minimal to determine actual chip temperatures. Generally, the temperature around a sensor differs from the ambient temperature due to temperature sensor self-heating.

2.2.2 Thermal Imaging

Another way of calibrating on-chip thermal sensors measures the thermal profile of a running chip using thermal imaging technologies and reports the sensor readings at that time instant [53]. Sensor model parameters can then be obtained by solving

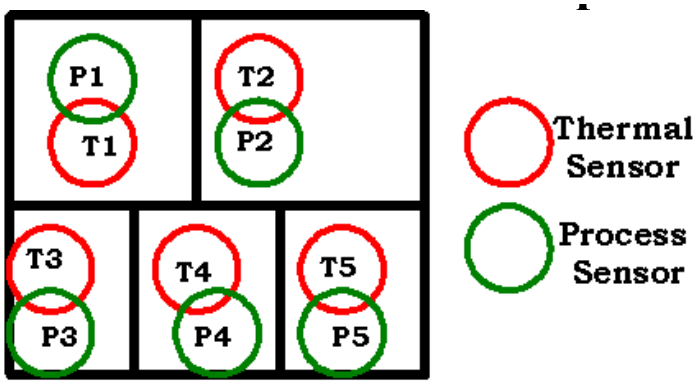


Figure 2.2: Co-located thermal and process sensors for calibration purpose [28]

a series of equations or by using statistical parameter inference [81]. Usually, the calibration cost associated with this approach is very high since every chip experiences a different thermal imaging response and the amount of effort increases with the number of on-chip thermal sensors. Although the approach could be used after chip fabrication, it cannot be used effectively at run time.

2.2.3 Design for Calibration

A second calibration technique uses design-for-calibration (DFC). This approach is implemented by integrating dedicated hardware circuits on chip which monitor the process variation around the thermal sensors [28]. Process sensors are co-located with thermal sensors and the process variation is compensated based on process measurements [28], as shown in Fig. 2.2. With the knowledge of the chip process variation, the errors in thermal sensor readings can be compensated and model parameters can be optimized to reflect the physical relationship between the temperature and physical quantities. Since the process variation monitoring hardware consumes silicon real estate, it raises the chip cost when a large number of sensors are integrated. Another DFC approach estimates the gain of thermal sensor via a small set of wokloads and power information [109].

2.2.4 Thermal Estimation

A third approach uses accurate on-chip thermal estimation instead of thermal imaging to determine actual temperatures. This information can then be used for calibration of specific sensors. In Liu [57] and Cochran and Reda [21], the authors describe methods to construct thermal estimates for numerous points in a processor from measurement data from a sparse set of thermal sensors. In Ranieri *et al.* [80], the overall thermal map is recovered from a reduced set of sensors by selecting principal eigenvectors of the whole-chip temperature vector. In Zhou *et al.* [120], an information-theoretic framework is proposed to find the optimal location for sensor deployment and full-chip thermal monitoring. Since the thermal sensor measurements are subject to noise, the amount of error at each specific sensor can be difficult to determine. In Zhang and Srivistava [114], the temperature for noisy sensors is estimated using statistics. As a result, most recent approaches for sensor calibration use a combination of sensor readings *and* other on-chip information to generate estimates of actual temperature.

Chip Level Temperature Estimation Using Performance Counters: Kumar *et al.* [48] use the 22 performance counters in an Intel Pentium-4 processor to estimate temperature. A linear combination of these performance counters predicts the overall chip temperature (2.5). For multiple sensor calibration, it is necessary to estimate the temperature at the micro-architectural level due to thermal gradients within the silicon die, so an estimation strategy with finer granularity is needed. Also, this method is only suitable for steady temperature estimation for applications with stable activity, but the estimation is not effective if the power profile of the application changes quickly.

$$T_{overall} = w_{const} + \sum_{i=1}^{22} w_i \frac{u_i}{t_{total}} \tag{2.5}$$

where $T_{overall}$ is the overall chip temperature; u_i is the value of a performance counter and t_{total} is the elapsed time. w_{const} and w_i are coefficients of the linear model.

Architectural Component Temperature Estimation Using Performance Counters: Lee *et al.* [49] proposed a run-time temperature sensing strategy using performance counters in high-performance processors. In this strategy, performance counters are used to estimate the power dissipation for each hardware component and the estimated power traces are then used to estimate the temperature trace based on the thermal model implemented in a thermal simulator (HotSpot). The mapping from power to temperature requires a complex thermal model which characterizes the thermal RC network of the given chip. A drawback is that the detailed in-system thermal simulation causes significant performance degradation and generates heat.

Architectural Component Temperature Estimation Using Fusion Techniques: In two recent papers [91][115], two sources of temperature information are combined to generate temperature estimates: (1) noisy sensor readings and (2) localized power consumption which is related to temperature. The technique used to integrate data from these two data sources is Kalman filtering (KF). A thermal model is calibrated offline and then used by a Kalman filter for estimation, as shown in Fig. 2.3(a). The power estimates for each architectural component in Fig. 2.3(b) are also obtained offline.

Although power traces can be accurately estimated at run time [78], a thermal RC model is required to determine the mapping coefficients required to convert power dissipation to temperature in the prediction step of KF approaches. Unfortunately, the derivation of this model is not trivial due to the complexity of silicon materials. KF-based approaches have shown the ability to track the temperature profile of a chip at a high computational cost since KF is performed each time a temperature estimation is made.

Collaborative Calibration for Wireless Sensor Networks: Although the calibration of a number of sensors on a silicon die using performance information from multiple sensors is a new challenge, similar problems have been studied in the wire-

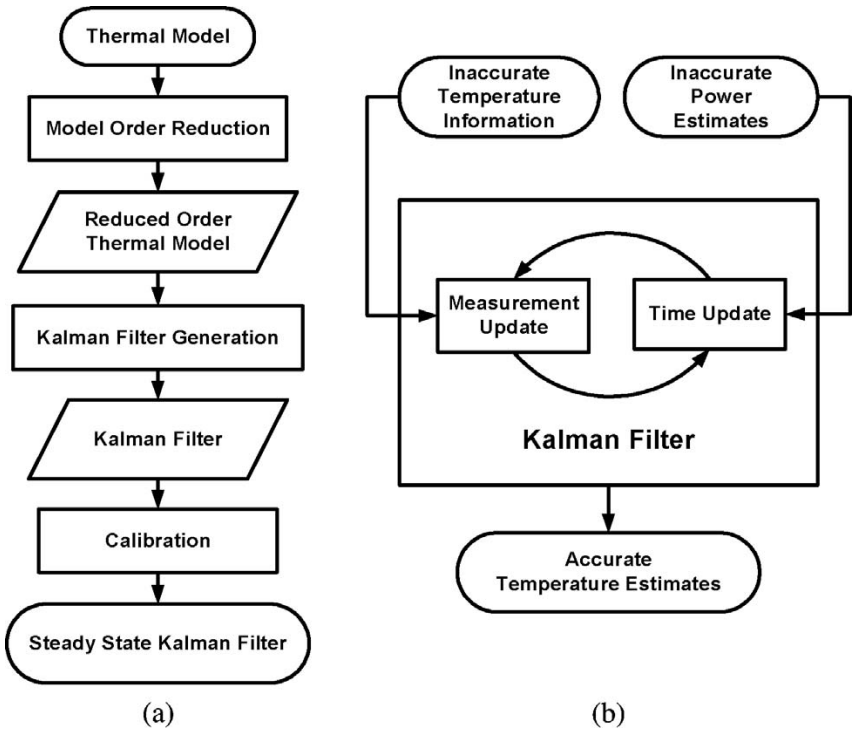


Figure 2.3: Kalman Filter Approach [91] (1) off-line model calibration (2) on-line estimation

less sensor network community for years. For example, a Bayesian inference method was employed to reduce the noise of sensor data [31]. The approach combines a priori knowledge of the expected reading, the noise characteristics of the sensors, and an observed noisy reading to obtain a more accurate reading estimate. Whitehouse, et al. [102] formulated the calibration of a large sensor network into a parameter estimation problem. They determined that micro-calibration (the calibration of sensors one-by-one) is sometimes problematic due to the lack of a calibration interface and unobservable environments. Thus, the need for macro-calibrations (collaborative calibrations) which utilize the correlation among sensors arises.

2.3 Dynamic Thermal Management

In this section, several dynamic thermal management techniques are summarized, including software-based techniques for many-cores. Overall, these techniques involve core-level thermal management and network-level thermal management. Core-level thermal management is a focus of this dissertation. Some thermal management approaches for data centers are also summarized to show similarity between chip-level thermal management and data center-level thermal management.

2.3.1 Dynamic Voltage and Frequency Scaling

As mentioned in Section 2.1, dynamic voltage and frequency scaling (DVFS) is a popular hardware-based thermal management technique for modern processors. Dynamic power consumption is determined from supply voltage and frequency as shown in (2.6)

$$P_{dyn} = \alpha C V^2 f \tag{2.6}$$

Here, α is the activity factor, i.e., the fraction of the circuit that is switching; C is the capacitance; V is the supply voltage; f is the frequency. The dynamic power can be changed by adjusting voltage and frequency with resulting changes in temperature.

Many DVFS algorithms have been proposed to utilize voltage and frequency adjustments to control temperature [47]. DVFS actions can be applied reactively and proactively for thermal management. The latter approach generally needs a predictive thermal model to estimate temperature trends and throttle the system in advance if there is a predicted thermal emergency. The reactive method lowers voltage and frequency once the thermal redline is crossed. Although DVFS can effectively reduce temperatures with a sacrifice in performance, some DVFS actions can be avoided if tasks can be scheduled or allocated in a thermally-efficient way.

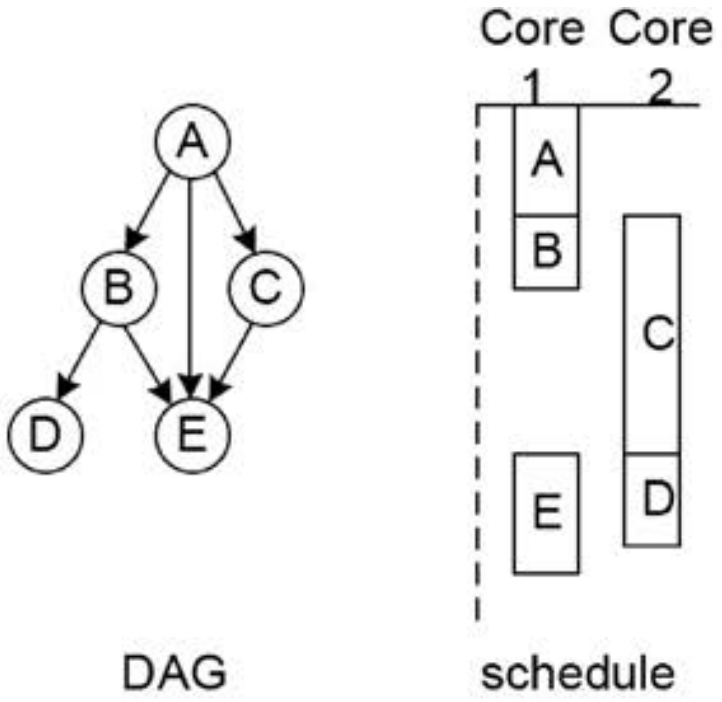


Figure 2.4: Thermal-aware scheduling for a task graph [51]. The tasks are labeled as bubbles in the DAG. Schedule time is shown on the vertical axis on the right.

2.3.2 Thermal-aware task scheduling and allocation

Thermal-aware task scheduling and allocation techniques have been widely studied for both single-core and multicore systems to reduce peak temperature and balance heat distribution.

Thermal-Aware Task Scheduling: For modern multi-task time-sharing systems, tasks are scheduled into time slots on a single core processor. Thermal-aware task scheduling optimizes the thermal condition of the chip. Temporal thermal correlation is taken into account in these problems to avoid hot jobs that are executed in a short time period. In Liu *et al.* [59], a thermal-aware scheduling algorithm with stochastic workload is presented to reduce to peak temperature. In Li *et al.* [50], the authors utilized compilation and dynamic instrumentation to identify process thermal intensity and then applied a scheduling algorithm to reduce temperatures.

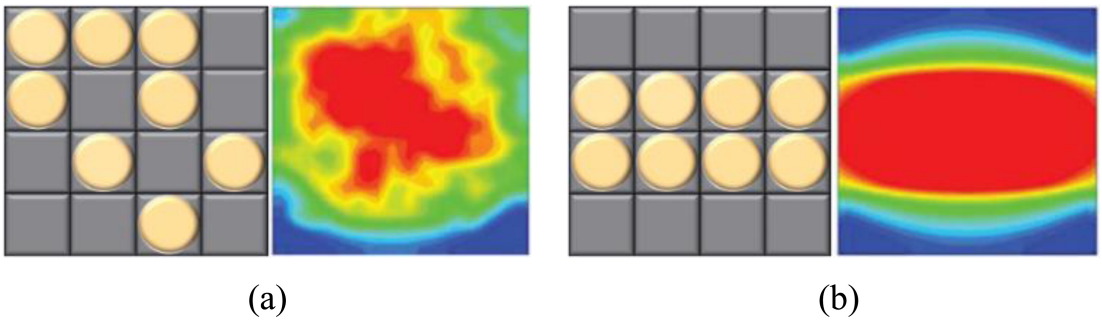


Figure 2.5: Different task distribution in a many-core [110] [37]

Generally, the scheduler has knowledge of the task dependency graph before it performs scheduling and the algorithm assigns tasks to time slots and cores, as shown in Fig. 2.4. These techniques were demonstrated in system software as an assistive feature to improve thermal conditions [18] [106].

Thermal-Aware Task Allocation in Many-Cores: For multi/many core systems, task allocation mechanisms have been developed to assign tasks to cores to provide improved thermal conditions. As shown in Fig. 2.5, different task distributions result in quite different chip thermal profiles. Thermal-aware task allocation generally considers the spatial thermal correlation between cores. Liu *et al.* [58] proposed a temperature prediction model which takes the temperature of neighboring cores into consideration. The approach was validated on a quad-core processor and showed throughput improvements under peak temperature constraints. Additional techniques to address thermal-aware task allocation are summarized subsequently.

Optimization Formulation with Constraints: Fisher *et al.* [33] considered heat transfer in a homogeneous multicore and presented global scheduling algorithms to minimize the peak temperature. In Coskun *et al.* [23], the authors formulated the task scheduling statically as a integer linear programing problem (ILP) by taking temperature into account in a multicore system. The authors do not take heat interaction among neighboring cores into account. In Chantem *et al.* [11], the au-

thors present a mixed-integer linear programming (MILP) technique for assigning and scheduling tasks to minimize peak temperature with real-time constraints on an MPSoC. Similar techniques were used in [40] [110] with different optimization goals. These works formulate the optimization problems with constraints and use integer linear programing to derive solutions. For example, the task scheduling problem is formulated as shown in (2.8) [110] to simultaneously maximize throughput and meet maximum temperature requirements. Here, f_i is frequency and w_i is the weight of core i. The throughput is defined as a weighted sum of all frequencies. All core temperatures must be lower than the preset redline temperature (T_{max}). Other constraints might be necessary if more factors are taken into account.

$$\text{Maximize:} \qquad \sum_{i=1}^{N}(w_i \times f_i) \tag{2.7}$$

$$\text{Constraints:} \qquad t_i \leq T_{max} \tag{2.8}$$

A limitation of these works is the inclusion of thermal models in the optimization problem for predictive decision making. Moreover, the knowledge of power consumption for each application is required to conduct thermal modeling, but runtime power estimation is quite challenging. Although various techniques are proposed to accelerate computation to reach an optimal goal, the solution doesn't scale well when the number of cores is large in a many-core system.

Adaptive Random Task Allocation: An intuitive approach to perform thermal-aware allocation is to select the coolest core for assignment [120]. The main problem with the *coolest* selection approach is that it doesn't differentiate between the thermal stresses of cores at the same temperature. For example, a core at the corner of a floorplan has higher horizontal thermal resistance than a core at the center of the floorplan, so it is more likely to exhibit a higher temperature in future evaluations. The heuristic *adaptive random* algorithm improves on the *"coolest"* approach by se-

lecting the coolest core for task allocation [24] under a set of calculated probabilities. Potential allocations of tasks to cores are assigned weights based not only on the cores' current temperatures, but also on their thermal history. Weights measuring thermal history are adjusted in real time as the cores execute a dynamic workload. Stochastic assignment is employed to allocate a new task to a core based on its current temperature, thermal history, and the thermal condition of neighboring cores. The memory cost associated with this method is significant. The temperature values for each core are stored for a period of time (around 1∼10K temperatures samples for each core) to capture the thermal characteristics of a core for specific workloads.

Learning Based Approaches: Machine leaning has been widely adopted for dynamic thermal and power management, and reinforcement learning based techniques are favored for DTM/DPM, especially for DVFS action strategies. The basic idea of these approaches is to learn a policy to configure frequency and voltage settings based on the current power and the thermal state of the system. Ge and Qiu [36] proposed a temperature reduction technique based on reinforcement learning for media applications. The agent learns the workload and dynamically adjusts frequency to control thermal violations. Similar techniques were applied in a power management context [30] [77] [111] [16] [45]. An advantage of reinforcement learning is that it does not require an explicit model for power or temperature. It learns the best policy to perform actions according to a standard procedure.

Thermal Management in 3-D and Heterogeneous Systems: In a 3-D system, silicon layers are stacked to achieve better performance and higher integration. Generally, stacking deteriorates the thermal environment of devices since the 3-D technology increases chip power density and slows heat dissipation. Many thermal aware task scheduling techniques have been proposed for 3-D processor chip systems [121] [66] [17] [60] [110]. Thermal aware techniques were also proposed for heterogeneous MPSoC systems [92].

2.3.3 Network-level Thermal Management

In contemporary many-cores, the power consumption of both network on chip (NoC) routers and processor cores is a significant concern. Many parallel and data intensive application can be adapted and implemented on many-cores to benefit from the low latency and high bandwidth of on-chip communication [98] [15]. The size of many-cores requires NoC routers to have significant control circuitry and buffer storage, leading to increased power consumption. The heat dissipated by the routers not only affects router temperature, but also the temperature of neighboring cores. Shang et al. [90] determined that chip temperature is impacted by thermal correlations among all on-chip components.

A number of thermal-aware routing algorithms have been proposed to control NoC router run-time temperatures [90] [12] [56] [26] [79] [55]. In general, these works are limited: (a) The thermal impact of processing cores is underestimated or ignored [79]; (b) Application specific designs are employed, so the solutions lack generality [79] [26]. Static task mapping on NoC systems can also achieve thermal balance and communication cost minimization [42]. In these designs, the communication paths among tasks are predetermined and fixed. Therefore, they are not suitable for a dynamic system where tasks arrive and depart in a random fashion.

2.3.4 Thermal-Aware Workload Allocation in Data Centers

Cooling in a data center environment is a big challenge due to the high energy consumption of dense server arrays [8]. Numerous studies have examined data center thermal issues [101] [13] [72] [96]. Server blades in a rack are physically close to each other and back-to-back racks are often laid out in rows. Although cooling systems are typically deployed in a data center environment, local thermal imbalances can create hot spots if workloads are clustered in physically close servers. Hot spots can cause hardware failures and permanent damage to electronic components. In many

circumstances, the allocation of server workloads must consider thermal effects to avoid overheating.

Many thermal distribution similarities can be observed between many-core environments and data centers. In many-cores, processor cores are positioned on a small silicon die and they exhibit thermal correlation. In a data center or a server rack, servers are thermally correlated due to constrained space. Workloads are assigned to different nodes by a master node in both many-cores and data centers. Although workload distribution in a data center is a macro scale problem which differs from a small silicon die, similar allocation techniques can be effectively applied considering that both aim to improve thermal conditions for thermally-correlated working nodes. Chen *et al.* [13][100] proposed workload allocation based on reinforcement learning to reduce the peak temperature in a data center. The approach avoids local heating by assigning workloads in a spatially-dispersed fashion. In this dissertation, reinforcement learning is applied to many-cores to reduce the maximum on-chip temperature.

CHAPTER 3

ON-CHIP THERMAL ESTIMATION VIA PERFORMANCE COUNTERS

To dynamically account for changing processor activities, collections of performance counter values can be used to estimate the chip thermal profile at run time. In this chapter, the correlation between global performance counters and temperatures of architectural components is explored. Two thermal estimation techniques are proposed to estimate temperature via linear regression. A performance counter selection method is employed to reduce the intercorrelations between performance counter readings and improve estimation accuracy. In this dissertation, *estimated temperature* is exclusively used to refer to a temperature obtained from thermal estimation using performance counters.

3.1 Thermal Correlation With Performance Counters

In a microprocessor, performance counters monitor run-time system statistics, such as the floating point instruction rate, the load/store rate, branch prediction miss rate, amount of cache misses, and instructions per cycle (IPC), among others, for various system management purposes. Typical system events recorded by performance counters are listed in Table 3.1. Since these statistics contain the activity information of functional units in the processor, they can be used to estimate the power consumption at a per-structure granularity using linear regression [78] or unit power consumption [49][105]. Unlike power consumption estimation, functional unit temperature estimation is more complex due to spatial thermal correlation resulting from heat flow across the chip.

Table 3.1: Performance Counters Provided by SESC

general rate	IPC, integer rate, load rate store rate, floating point rate
cache	Dcache read miss rate, Dcache write miss rate, Icache miss rate
buffer and queue usage	load queue, store queue, ROB, Iwin, TLB
branch	BTB (branch target buffer utilization), RAS (return address stack size)

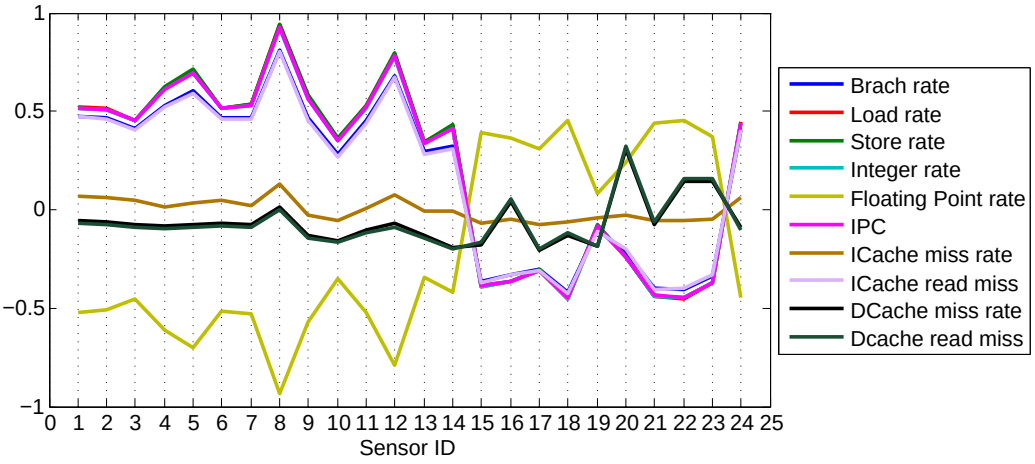


Figure 3.1: Correlation between temperature and some system statistics for the *radix* benchmark across 24 thermal sensors

It can be shown that the temperatures of functional units are correlated with values read from on-chip performance counters. Fig. 3.1 shows the correlation coefficients of component temperatures and various system statistics for the SPLASH-2 *radix* benchmark [104]. Most temperature-statistic pairs show non-zero correlation coefficients. For example, the floating point rate shows a negative correlation with integer units (e.g. the integer scheduler) and a positive correlation with floating point components (e.g. the floating point scheduler).

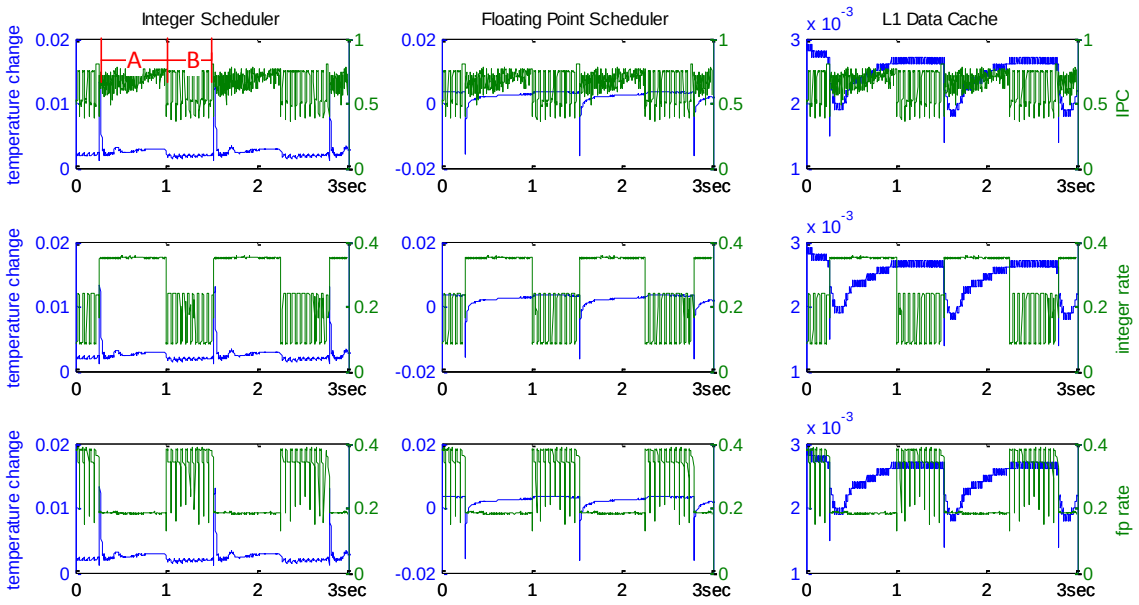


Figure 3.2: Runtime recording of some system statistics and temperature change rates for three function units: integer scheduler (left column), floating point scheduler (middle column) and L1 data cache (right column) for the *equake* benchmark. Temperature changes are in ^oC

To explore the correlation between the application characteristics and temperature changes over a short time period, system statistics and a temperature trace were recorded for every millisecond via simulation using SESC [83] and HotSpot [2]. Fig. 3.2 illustrates the relation between the system statistics: IPC (the first row), integer instruction rate (the second row) and floating point instruction rate (the third row), and temperature change rates for three functional units: integer scheduler (left column), floating point scheduler (middle column) and L1 data cache (right column). The performance counter data was obtained by repeatedly running the *equake* benchmark from the SPEC2000 benchmark suite using the SESC simulator. The temperature trace was generated by HotSpot.

As seen in the figure, higher average IPC (phase A in the top-left sub-figure) results in a higher temperature change rate for the integer scheduler at the start of phase A. However, this change rate is negative for the floating point scheduler

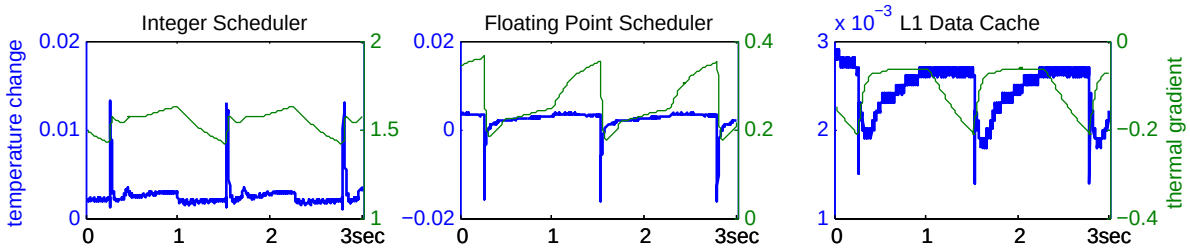


Figure 3.3: Runtime recording of thermal gradient and temperature change rates for three function units. Temperature changes are in ^oC

at the same point. The floating point instruction rate is higher in phase B and the scheduler is more active during this phase leading to a floating point scheduler instruction temperature surge every time the application transitions from phase A to phase B, although it is short. In this case, the temperature of the component is impacted by its surroundings due to heat flow.

The temperature difference (referred to as thermal gradient in the figure) between the specific component and a neighboring component is plotted to illustrate this point in Fig. 3.3. The thermal gradient in the figure is obtained by taking the maximum temperature difference among all neighboring blocks. In the integer scheduler, for example, the temperature surge causes an increase in the thermal gradient which accelerates heat flow from the integer scheduler to its neighbors. A new thermal balance is quickly reached after a short time.

3.2 Temperature Estimation Using Performance Counters

By virtue of these complexities, the temperature at a specific position on the chip is generally not linearly related to one particular performance counter, so it is not possible to construct a thermal map using a few independent performance counters. However, performance counters are correlated with each other as discussed in previous section.

The non-linear impact of performance counter correlation on temperature can be illustrated via a simple example (Note: the real relationships generally are more complex). Consider two performance counters, x and y, that are quadratically related, as shown in (3.1).

$$y = \alpha_1 x + \alpha_2 x^2 \tag{3.1}$$

Also, the temperature, T, has a closed form representation based on these two variables given by the following equation.

$$T = \gamma_1 x + \gamma_2 x^2 + y \tag{3.2}$$

By replacing x^2 in Equation (3.2) with a reordered version of Equation (3.1), the following linear representation is obtained, where A and B are determined by Equation (3.4).

$$T = Ax + By \tag{3.3}$$

$$A = \gamma_1 - \alpha_1\gamma_2/\alpha_2 \quad \text{and} \quad B = 1 + \gamma_2/\alpha_2 \tag{3.4}$$

Effectively, since α and γ are constant, temperature can be approximated with a linear representation. Although this example is trivial compared to actual on-chip thermal analysis, it provides a basis for our model derivation in the next section assuming a sufficient amount of performance counters are available to be used to provide accuracy.

The linear approximation is shown to be effective empirically in developing an on-chip thermal profile. These thermal estimates can then be merged (Chapter 4) with sensor readings to reduce spatial (across sensors) and temporal (across time) noise.

3.2.1 Linear Model for Absolute Temperature Estimation

Similar to the method in [48], a linear model can be built using values from system performance counters in the processor to estimate the *absolute* temperature of

a specific component in the processor rather than the whole chip temperature in [48]. As shown in the following equation (3.5), the temperature for an integer scheduler (unit 8 in floorplan shown in Fig. 3.4) is estimated via a linear combination of performance counter values. M_{ij} is the accumulated value of a system event recorded by performance counter j at time instance i and t_i is the elapsed time. Therefore, $\frac{M_{ij}}{t_i}$ is the average rate for a particular event. T_i^8 is the estimated temperature at time instance i using these counter values. The coefficients, β_j^8, are determined in the model training phase.

$$T_i^8 = \beta_0^8 + \beta_1^8 \frac{M_{i1}}{t_i} + \beta_2^8 \frac{M_{i2}}{t_i} ... + \beta_k^8 \frac{M_{in}}{t_i} \tag{3.5}$$

The above equation can be rewritten in matrix form, as shown in (3.6), to represent thermal estimation for multiple locations. At time instance i, $\mathbf{T_i}$ is an $m \times 1$ column temperature vector and $\mathbf{M_i}$ is a $n \times 1$ column performance counter vector. $\boldsymbol{\beta}$ is an $m \times n$ coefficients matrix.

$$\mathbf{T_i} = \boldsymbol{\beta} \times \mathbf{M_i} \tag{3.6}$$

The estimated temperatures can be calculated quickly in real time because only scalar multiplications and additions are involved in (3.5).

3.2.2 Linear Model for Incremental Temperature Estimation

The drawback of the absolute temperature estimation is that it essentially estimates the steady state temperatures instead of transient ones by including the averages rates of performance counters in the linear model. For application with drastic activity changes, this technique is limited by its incapability to track the temperature at run time. However, we demonstrate that a linear model can be used to estimate on-chip temperature changes at specific temperature sensors using multiple performance counters if the time interval is small. In the following derivation we are interested

in determining ΔT estimates for specific sensors over a time interval, rather than absolute T values.

In developing a linear model for a specific thermal sensor i over a time interval, a row vector ($\mathbf{x^i}$) contains recorded performance counter values for the interval, $\mathbf{M}$ (listed in Table 3.1), thermal gradient information, $\mathbf{G^i}$, and temperature, T^i. Values $\mathbf{G^i}$ measure the thermal gradient between other thermal sensors and sensor i at the beginning of the interval. Value T^i measures the temperature of sensor i at the beginning of the interval. Since the correct temperature before the first interval is unknown, the thermal gradients and T^i can be approximated by using thermal sensor readings at the start of estimation. For other intervals, the temperature estimation from the previous interval is used. The combination of these variables $\mathbf{M}$, $\mathbf{G^i}$, and T^i forms $\mathbf{x^i}$:

$$\mathbf{x^i} = [\mathbf{M}, \mathbf{G^i}, T^i] \tag{3.7}$$

For example, for the integer scheduler (unit 8), $\mathbf{G^8}$ includes all temperature differences between the integer scheduler and other components at the beginning of the measurement interval. Here, the superscripts of T indicate the hardware components in the floorplan (Fig. 3.4 in Section 3.4). The thermal gradient vector for the integer unit is given by (3.8). There are 24 architectural components in the studied processor, so $\mathbf{G^8}$ contains 23 elements which are temperature differences between the components and the integer scheduler, except itself. The performance counter vector $\mathbf{M}$ can be represented by (3.9), where u is the number of performance counters used in the model.

$$\mathbf{G^8} = [T^1 - T^8, ..., T^7 - T^8, T^9 - T^8, ..., T^{24} - T^8] \tag{3.8}$$

$$\mathbf{M} = [P^1, P^2, ..., P^u] \tag{3.9}$$

The T^i value in (3.7) is used to take static power (which is dependent on temperature) into account. Therefore, the sampled vector at a particular time step is

given by (3.10) for the integer scheduler. It is apparent that both hardware activities (as measured by performance counters) and thermal gradients impact temperature change during the sampling interval.

$$\mathbf{x^8} = [P^1, P^2, ..., P^u, T^1 - T^8, ..., T^7 - T^8, T^9 - T^8, ..., T^{24} - T^8, T^8] \tag{3.10}$$

Performance counter values represent *changes* in the respective event counters during the sampling interval. Only events happening in a specific interval are evaluated for the corresponding performance counter monitors. Using the above $\mathbf{x}$ vector, it is possible to estimate the temperature change of a particular component which contains the thermal sensor during the sampling interval using a linear equation. For example, the equation for thermal sensor i is:

$$\Delta T^i = \mathbf{x^i} \cdot \boldsymbol{\beta}^i \tag{3.11}$$

and for the sensor in the integer scheduler:

$$\Delta T^8 = \mathbf{x^8} \cdot \boldsymbol{\beta}^8 \tag{3.12}$$

Here, $\boldsymbol{\beta}^8$ is a column vector whose elements are coefficients of the linear model. The coefficient vector $\boldsymbol{\beta}^8$ can be determined through model training which will be discussed in the next subsection. Each sensor i is trained separately to obtain its own $\boldsymbol{\beta}$ coefficient vector. The linear model only needs multiplications and additions to calculate the results, so the time cost is low and calculations can be done in real time.

3.3 Linear Model Training

As mentioned earlier in this chapter, the first step in developing a relationship between performance counter values and estimated temperatures (e.g. $\boldsymbol{\beta}$ vectors)

involves training. The accuracy of the coefficient vector $\boldsymbol{\beta^i}$ impacts the model accuracy for sensor i. In the training step, accurate known temperatures for the sensors must be available to develop the relationships. These relationships can be determined via architectural and thermal simulation during design once physical characteristics of the chip have been determined or during post-fabrication testing using thermal imaging. The accuracy of the model trained by simulation can be limited by the effectiveness of the simulators since they cannot simulate every detail of a real system. In post-fabrication testing, it is possible to feed real workloads to the system and read performance counter registers. At the same time temperature values can be captured through infrared imaging of the running system. Unlike per-chip calibration, it is only necessary to perform data capturing on a small amount of sample chips to get the general information of a particular chip series. We assume that the specific information of an individual chip caused by process variation is reflected in the thermal sensors.

3.3.1 Model Training for Absolute Temperature Estimation

In the training phase, sensor temperatures and performance counter values are recorded at each time instance for a series of time instances (n performance counters). The estimation error of the absolute temperature for module k at time instance i is given by

$$e_i^k = \sum_{j=0}^{n} (\hat{\beta}_{kj} M_{ij}) - T_i. \tag{3.13}$$

The ordinary least squares (OSL) regression method minimizes the sum of squares of errors for l time instances:

$$S = \sum_{i=1}^{l} (e_i^k)^2 \tag{3.14}$$

Equation (3.15) shows the estimator of coefficients, $\boldsymbol{\beta}$, for the multi-variable least squares method.

$$\hat{\boldsymbol{\beta}} = (\mathbf{M}'\mathbf{M})^{-\mathbf{1}}\,\mathbf{M}'\mathbf{T} \tag{3.15}$$

Here, each column of $\mathbf{M}$ is a time series of a particular performance counter and each column of $\mathbf{T}$ is a time series of temperature at a particular location. $\mathbf{M}'$ is the transposition of $\mathbf{M}$. Once $\hat{\boldsymbol{\beta}}$ is calculated, it can be stored in the programmable registers or re-programmable ROM region of the system.

3.3.2 Model Training for Incremental Temperature Estimation

Similarly, we assume that accurate temperatures changes and $\mathbf{x}$ values consisting of $\mathbf{M}$ and $\mathbf{G}$ are available for all sensors. The most straightforward way to train the linear model to determine $\boldsymbol{\beta}$ vectors in (3.11) is to use an ordinary least square method (OLS). The coefficient vector for the incremental temperature model obtained through OLS is given by (3.16).

$$\boldsymbol{\beta}^{i}{}_{ols} = (\mathbf{X^{i}}^{\mathbf{T}}\mathbf{X^{i}})^{-1}\mathbf{X^{i}}^{\mathbf{T}}\mathbf{y^{i}} \tag{3.16}$$

Here, $\mathbf{X^{i}}$ is a matrix consisting of row vectors $\mathbf{x^{i}}$ calculated over a series of $\mathbf{N}$ sampling intervals. Each row of $\mathbf{X^{i}}$ represents $\mathbf{x^{i}}$ for one sample interval. $\mathbf{y^{i}}$ is a column vector comprised of accurate *actual* temperature changes for sensor i which occur during the respective training intervals. Although OLS is capable of training the linear model, its somewhat simplistic formulation does not consider the intercorrelation of performance counters, limiting accuracy.

A more advanced, iterative mathematical approach can be used to determine $\boldsymbol{\beta}$ values. As an alternative to OLS, we use automatic relevance determination (ARD), which was developed by MacKay [67] and Neal [74]. The coefficient vector $\boldsymbol{\beta}^{i}$ for sensor i can be represented by the following expressions (3.17) and (3.18).

$$\boldsymbol{\beta}^{i} = \delta^{-2}\mathbf{S}\mathbf{X^{i}}^{T}y^{i} \tag{3.17}$$

$$\mathbf{S} = \left(\mathbf{A} + \delta^{-\mathbf{2}}\mathbf{X^{i^T}}\mathbf{X^i}\right)^{-1} \tag{3.18}$$

In (3.18), $\mathbf{A} = \text{diag}(\alpha_1, ..., \alpha_M)$, which is a diagonal matrix. Each α_j in $\mathbf{A}$ represents the relevance of an input vector variable to the result such that:

$$\alpha_j = \frac{1 - \alpha_j {S_{jj}}^2}{\beta_j^i} \tag{3.19}$$

$$\delta^2 = \frac{\sum_{n=1}^{N}(y_n^i - {x^i}_n \cdot \beta_n^i)^2}{N - \sum_{j=1}^{M}(1 - \alpha_j S_{jj})} \tag{3.20}$$

Since (3.17) and (3.18) depend on (3.19) and (3.20) and vice versa, multiple iterations are needed to achieve convergence of the β^i unknowns. These iterations calculate α_j in diagonal matrix $\mathbf{A}$ and δ. In the above equations, S_{jj} are elements of $\mathbf{S}$. y_n is the n^{th} element of $\mathbf{y^i}$ and $\mathbf{x}_n$ is the n^{th} row vector of $\mathbf{X^i}$. N is the number of training samples and M is the length of vector $\mathbf{x^i}$ and the dimension of the matrix $\mathbf{S}$. Values for α_j and δ^2 are determined by alternating evaluation of the above four equations until convergence. From our experiments, around four iterations are performed until these parameters reach convergence.

3.4 Infrastructure and Experimental Approach

A simulation-based method is employed for data collection, model construction and verification. Two simulators used by this work and other experimental infrastructure are described in this section.

3.4.1 Architectural Simulator

We use the SESC simulator [83] as the infrastructure for collecting system statistics. SESC is a cycle-accurate simulator which models a full out-of-order pipeline with branch prediction, caches, buses, and other components of a modern processor. It can also report power traces of system components which are used for thermal simu-

lation. The simulator was modified to support the on-the-fly dumping of performance counter recordings which are synchronized with power traces.

The SESC simulator provides abundant system statistics for architectural analysis. Table 3.1 lists a subset of these statistics. Some simulation-related metrics, e.g. simulation speed, are not used in our strategy since it would not be available to a typical many-core user at run-time. The selection of performance counters is critical for achieving a good temperature estimation. Performance counters that give little correlation with temperature for most functional units are excluded from the estimation. The performance counter selection procedure involves a select-and-test iteration during the model training period, i.e. train the model using a set of selected performance counters and perform a cross-benchmark test (different benchmarks are used for training and testing) on the trained model.

During linear model training for β parameters and for model verification, SESC is used to record the power trace for applications. This information is used by HotSpot [2], a thermal simulator, to determine *actual* temperatures that can be used for training or for comparisons versus thermal estimates to verify our approach. However, since only dynamic power consumption is reported by the simulator and static leakage power accounts for a non-negligible part of total power dissipation for submicron technology nodes (about 40% for our chosen node of 45 nm), we add a static power estimate to the SESC power estimate for each functional unit. First, a dynamic power trace of all function units for a specific application is generated. A percentage of processor dynamic power (40% based on the prediction in [29]) is used to estimate static power and a portion of this power is added to the dynamic power trace for each functional unit (proportional to area). To account for the effects of temperature on static power, the power trace is fed to HotSpot and thermal simulation is performed. The static power for each functional unit is then adjusted using thermal dependency linearization [61]. A combination of the adjusted static power and the

dynamic power is then used for model training and verification using HotSpot. We have found that the effects of temperature-dependent static power are small over the temperature change range considered.

3.4.2 Thermal Simulator

The HotSpot simulation tool, which takes power traces from SESC, target processor geometry and material parameters as inputs, is used to generate accurate "golden" temperatures. As mentioned in the previous subsection, it is assumed that the HotSpot generated temperature values are the *actual* temperatures for training considering the sophisticated thermal diffusion model implemented by HotSpot. An AMD Athlon64 processor is used to assess our approach. The floorplan of AMD Athlon64 processor is shown in Fig. 3.4. The processor includes 24 functional blocks, each of which is labeled in the figure. Each block contains a thermal sensor. According the processor specification of AMD Athlon 64 fabricated under 130 nm SOI technology, the reported die size is 193 mm^2 [1]. After technology scaling, the area of the processor is estimated to be 24 mm^2 in 45 nm technology. The frequency of the processor is configured at 1 GHz in simulation and the overall initial temperature of the processor is set to 50^oC.

3.5 Model Evaluation

The SPLASH-2 and SPEC2000 benchmark suites were used to validate the effectiveness of our linear models for both absolute and incremental temperature estimation. Section 3.5.1 presents the results for absolute temperature estimation, and other sections are dedicated to incremental temperature estimation.

3.5.1 Evaluation of Absolute Temperature Estimation

The *velosity* benchmark is used to train the linear model (find $\boldsymbol{\beta}$) values and the trained model is tested for temperature estimation on other benchmarks. Figure

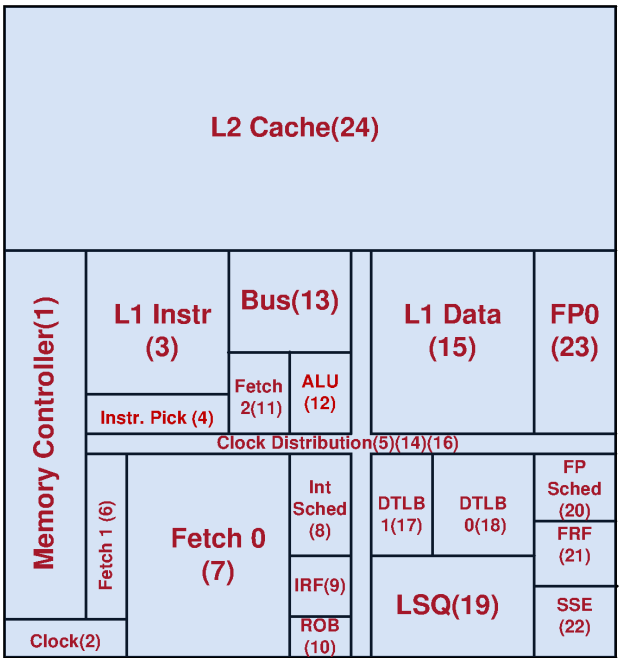


Figure 3.4: Floorplan of the Athlon 64 processor [71]

3.6 shows the estimated and actual temperature profile for several benchmarks at a representative time instance. In the subfigures, each point on the x axis represents a thermal sensor value in the processor and there are 24 total sensors integrated on the chip. Sensor 8 and sensor 19 correspond to the integer scheduler and load/store unit, respectively, and they have relatively high temperatures due to high activity. Sensor 24 is in the L2 cache of the processor and its temperature is low because of its large area and relatively low activity.

The estimated temperature profile and the actual temperature profile have very similar shapes in the graphs, so the relative relationship among sensors are estimated correctly. Graphs at other time points are similar. Figure 3.5 shows the correlation between the estimated and actual temperature profile curves for the benchmarks over a series of 3,000 time points. For all benchmarks, the correlation coefficient is larger than 0.9 which indicates a good linear relationship between the two curves. However, the estimated and actual curves are offset in terms of the absolute temperature value,

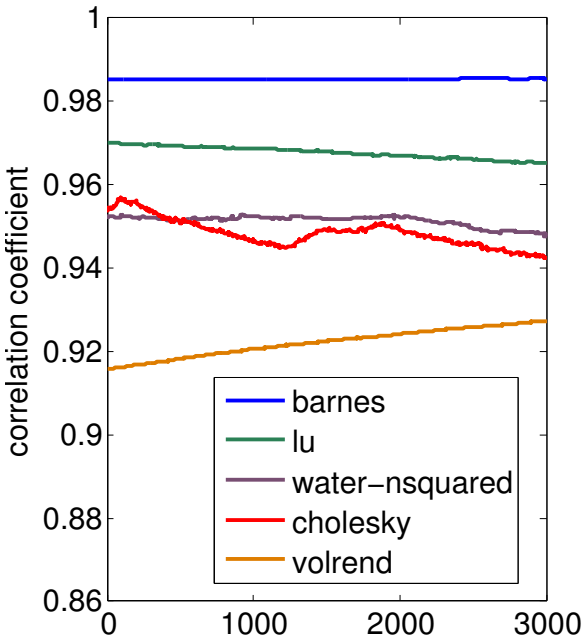


Figure 3.5: The correlation between estimated and actual temperature profiles

as shown in Figure 3.6. In next chapter, this systematic drift is offset by adding constant values to temperatures determined from sensor readings.

3.5.2 Evaluation of Incremental Temperature Estimation

Mixed samples from a subset of benchmarks were used to train the linear model (find $\boldsymbol{\beta}$ values) and the trained model was tested for temperature estimation on the rest of the benchmarks. To evaluate the accuracy of the linear model, Tables 3.2 and 3.3 show the estimation error for one time interval. In this case, (3.11) is evaluated for one time interval, using known T values to determine $\mathbf{G}$ gradients and measured performance counter values $\mathbf{P}$. Errors between the actual ΔT and ΔT values determined with (3.11) are then calculated.

Table 3.2 gives the average absolute error and error standard deviation for each sensor for a single interval using the β values determined through training. The error is averaged over all 16 test benchmarks (benchmarks described in more detail in Section 3.4). Using information from this table it is possible to evaluate the trained models for

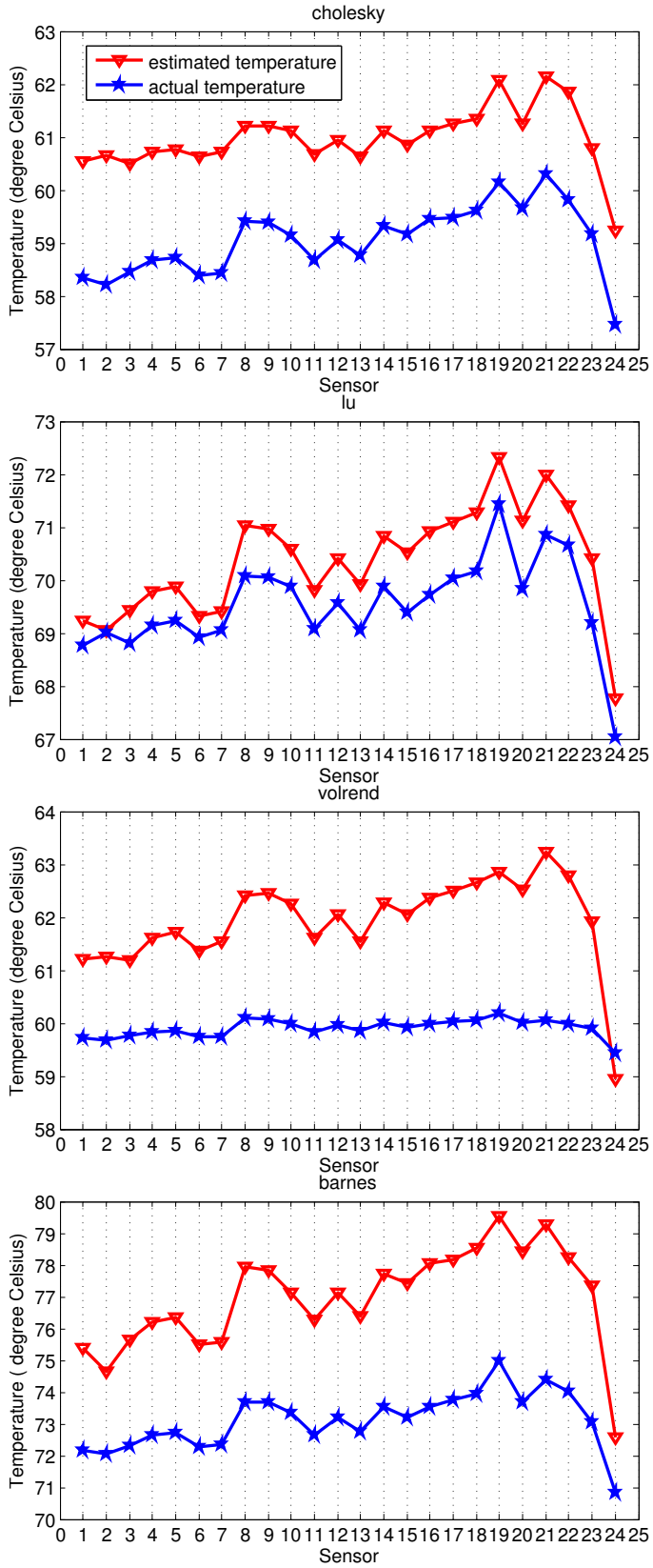


Figure 3.6: The estimated and actual processor temperature profile at one time instance for four SPLASH2 benchmarks. Each of the 24 thermal sensors in the processor are represented on the horizontal axis for the time instance.

Table 3.2: Average estimation error for each sensor over all benchmarks

Sensor ID	1	2	3	4	5	6
Avg. abs. error (^oC)	0.0003	0.0002	0.0003	0.0002	0.0002	0.0002
Std. abs. error (^oC)	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002
Avg. change (^oC)	0.0011	0.0010	0.0078	0.0081	0.0069	0.0010
Std. change (^oC)	0.0007	0.0005	0.0037	0.0043	0.0047	0.0007
Sensor ID	7	8	9	10	11	12
Avg. abs. error (^oC)	0.0002	0.0035	0.0003	0.0002	0.0002	0.0014
Std. abs. error (^oC)	0.0002	0.0045	0.0002	0.0002	0.0002	0.0013
Avg. change (^oC)	0.0022	0.0838	0.0288	0.0222	0.0046	0.0274
Std. change (^oC)	0.0016	0.0272	0.0122	0.0096	0.0035	0.0090
Sensor ID	13	14	15	16	17	18
Avg.abs. error (^oC)	0.0002	0.0003	0.0002	0.0004	0.0003	0.0003
Std. abs.error (^oC)	0.0002	0.0002	0.0002	0.0003	0.0002	0.0002
Avg. change (^oC)	0.0166	0.0133	0.0032	0.0094	0.0139	0.0140
Std. change (^oC)	0.0088	0.0088	0.0024	0.0058	0.0078	0.0076
Sensor ID	19	20	21	22	23	24
Avg.abs. error (^oC)	0.0016	0.0072	0.0003	0.0015	0.0014	0.0005
Std. abs.error (^oC)	0.0008	0.0049	0.0003	0.0010	0.0009	0.0003
Avg. change (^oC)	0.0102	0.0437	0.0139	0.0126	0.0120	0.0001
Std. change (^oC)	0.0056	0.0119	0.0060	0.0049	0.0044	0.0001

Table 3.3: Average estimation error for each benchmark over all sensors

benchmark	mcf	vortex	swim	art	apsi	water-spatial
Avg. abs. error (^oC)	0.0017	0.0005	0.0008	0.0019	0.0006	0.0004
Std. abs. error (^oC)	0.0022	0.0003	0.0009	0.0004	0.0006	0.0004
Avg. change (^oC)	0.0082	0.0117	0.0365	0.0338	0.0048	0.0181
Std. change (^oC)	0.0002	0.0116	0.0058	0.0002	0.0004	0.0172
benchmarki	radiosity	ocean	radix	parser	twolf	fft
Avg. abs. error (^oC)	0.0006	0.0011	0.0032	0.0004	0.0002	0.0008
Std. abs. error (^oC)	0.0007	0.0009	0.0010	0.0007	0.0001	0.0010
Avg. change (^oC)	0.0086	0.0409	0.0078	0.0113	0.0036	0.0195
Std. change (^oC)	0.0045	0.0123	0.0060	0.0036	0.0031	0.0112
benchmark	vpr	ammp	applu	barnes		
Avg.abs. error (^oC)	0.0003	0.0005	0.0005	0.0007		
Std. abs. error (^oC)	0.0004	0.0006	0.0010	0.0003		
Avg. change (^oC)	0.0068	0.0047	0.0178	0.0046		
Std. change (^oC)	0.0046	0.0006	0.0178	0.0022		

all sensors. Sensor 8 (integer scheduler) and sensor 20 (FP scheduler) report relatively high error and error variation compared with other sensors due to their high activity. Table 3.3 gives the average absolute error and the associated standard deviation for each benchmark. The error is averaged over all sensors for each benchmark. From this table, we can evaluate how the trained models work for all benchmarks. In general, average absolute error and standard deviation are low in the tables. During experimentation we found that the trained model was most effective for benchmarks which have similar execution characteristics to the benchmark training set. However, the use of a broad class of benchmarks for training helps minimize error across a larger number of benchmarks.

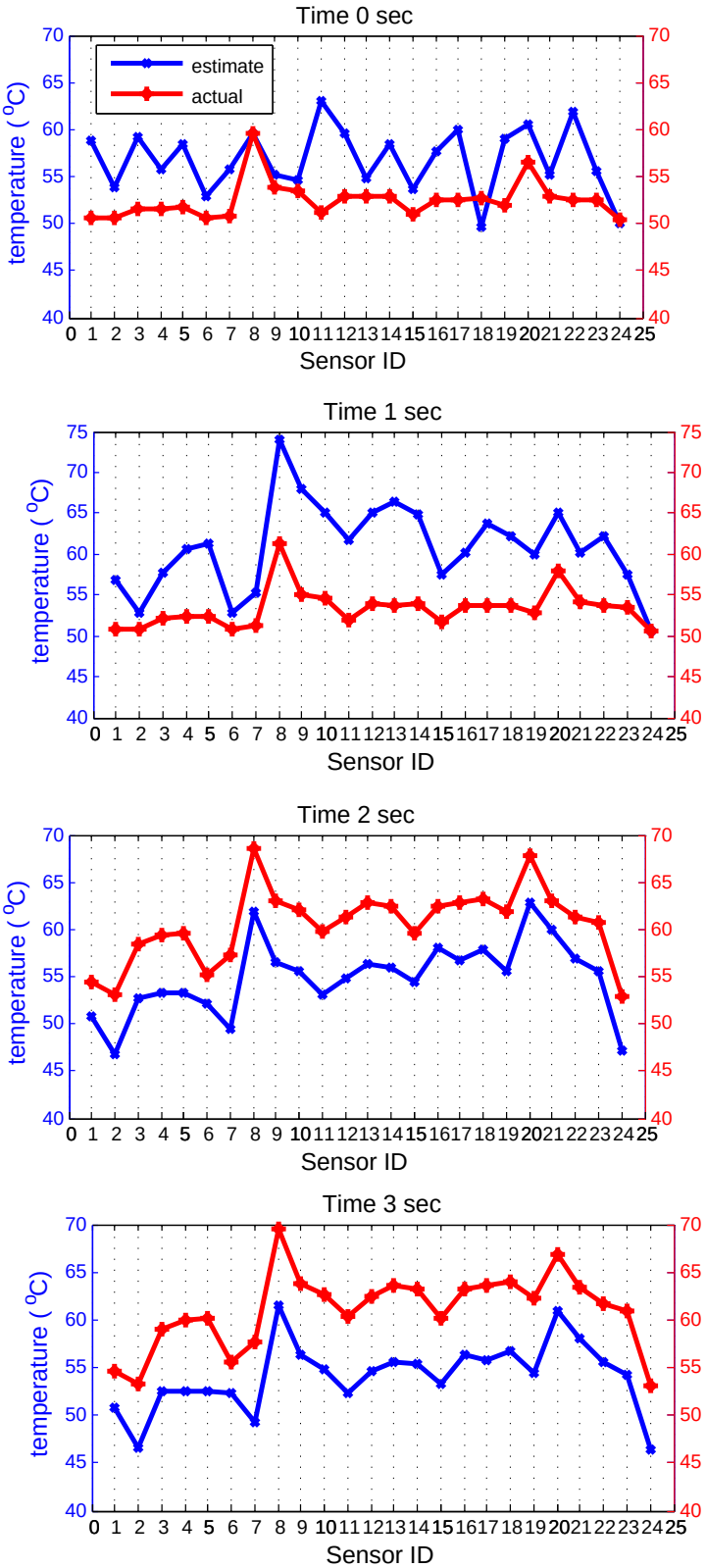


Figure 3.7: The estimated and actual processor temperature profile at four time instances for the SPLASH-2 *ocean* benchmark.

3.5.3 Estimated Thermal Profile for Incremental Temperature Estimation

Fig. 3.7 shows the thermal profile across all sensors at four time instances of the SPLASH-2 *ocean* benchmark. Experiments with other benchmarks created similar graphs. At the first time point, the estimated profile is inaccurate due to the lack of knowledge of initial temperatures. In succeeding time points, the estimated temperature profile more closely matches the actual thermal profile. At the 3^{rd} second, a close temperature profile match is achieved. It should be noted that while an *absolute* temperature match is not achieved, a *relative* match across the sensors is provided. In Section 4, this systematic drift is offset by adding constant values to temperatures estimated from sensor readings.

3.5.4 Temporal Evolution of Incremental Temperature Estimation

Fig. 3.8 shows how the estimated temperature progresses over time for three sensors (integer scheduler, ALU and floating point scheduler). Other sensors show similar trends. Since the initial temperatures are randomly chosen around 55^oC for all sensors, the estimation mainly reflects heat diffusion during the first 3 seconds. Over time, the temperatures of these three components are corrected to match their approximate relative values (the floating point scheduler is the hottest and ALU is the coolest). Fig. 3.9 shows the correlation coefficient between these two values over time.

To evaluate accuracy, the effect of limiting the number of performance counters used to generate thermal estimates is also considered. Table 3.4 indicates the average absolute error over all sensors and benchmarks for different numbers of performance counters used to generate estimates. Fourteen of the counters provide little benefit in terms of absolute error.

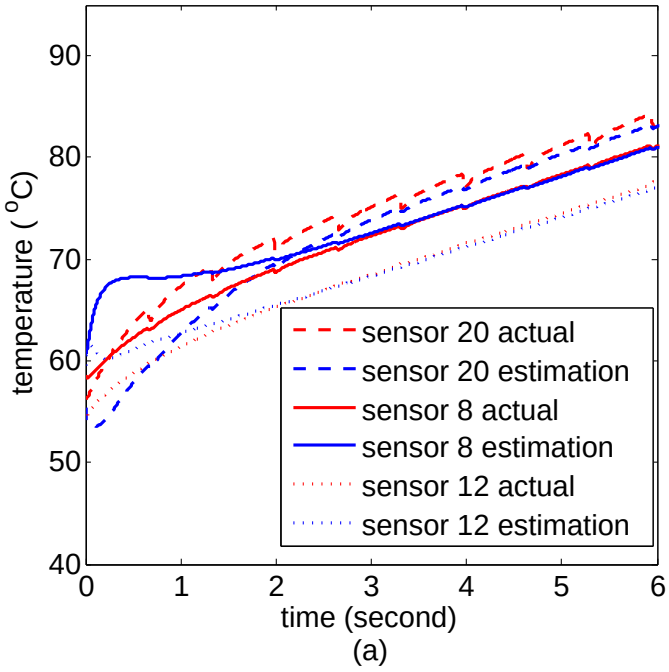


Figure 3.8: Temperature estimation over 6 seconds for sensor 8 (integer scheduler), 20 (floating point scheduler) and 12 (ALU). The data is collected through simulation by running *applu* on the AMD floorplan.

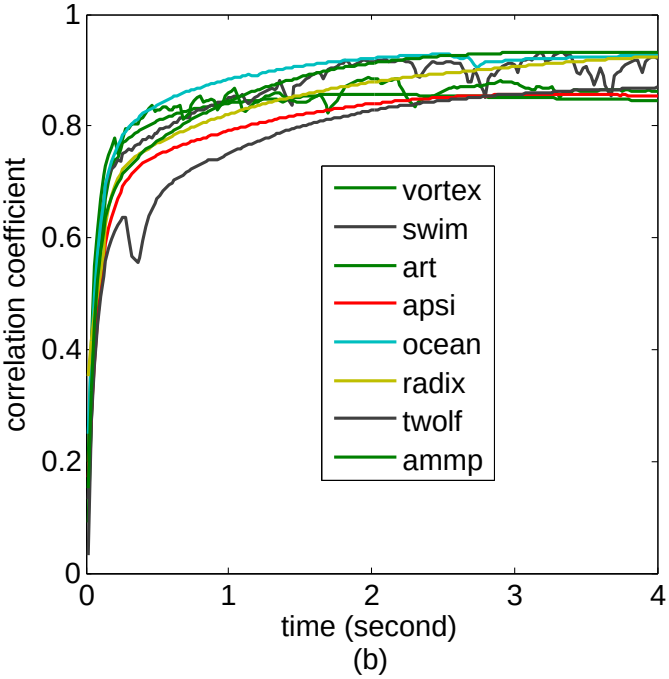


Figure 3.9: The correlation between estimated and actual temperature profiles for various benchmarks.

Table 3.4: Average absolute error of temperature estimation for all sensors and benchmarks if the number of performance counters is limited to specific quantities

No. counters	Abs. error (^oC)
5	0.5000
10	0.0164
15	0.0031
20	0.0009
34	0.0009

3.6 Principal Components of Performance Counter Vectors

In Section 3.2 it was shown that combinations of performance counter changes and thermal gradients can be combined to estimate temperature changes. However, it has previously been determined that performance counter values are correlated, potentially leading to model instability [22]. For example, a branch miss prediction may lead to a pipeline flush which impacts IPC. To explore the impact of this issue, experiments were performed to replace the P^i values in (3.9) and (3.10) with uncorrelated *principal components* [44].

Principal component analysis (PCA) transforms an input vector (in this case u performance counter values) into a new vector set by multiplying the input values with a matrix of the eigenvectors derived from the set, as shown in (3.21).

$$P'_{1\times u} = P_{1\times u} * C \tag{3.21}$$

$P_{1\times u}$ is the original vector of u performance counter values collected from the processor and C is a coefficient matrix of eigenvectors determined during model training. $P'_{1\times u}$ is the principal component vector. In many cases, depending on the eigenvalues of the original data set, some of the $P'_{1\times u}$ set may be ignored, leading to a reduced dimension vector $P'_{1\times v}$. Rather than inserting $P_{1\times u}$ performance counters into the linear model in (3.11), the reduced dimension principal component estimates are inserted instead.

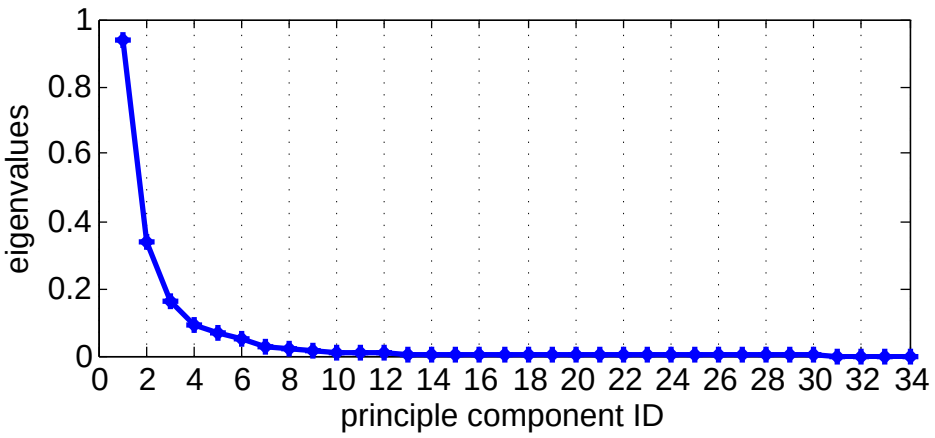


Figure 3.10: Eigenvalues calculated for 34 principal components from 34 performance counters collected using CINT2000 benchmarks and SESC simulator.

Since variables in $P'_{1\times v}$ are orthogonal with each other, the multicollinearity problem is eliminated.

Experimentation showed that the largest 14 principal component estimates correspond to non-zero eigenvalues, but the remaining 20 have eigenvalues close to zero. In general, to maintain maximum accuracy, the number of principal components (e.g. the dimension of v in $P'_{1\times v}$) used in the model should include the number of non-zero eigenvalues, in this case 14.

Fig. 3.11 shows the thermal estimation error for different numbers of principal components used in the model. As expected, accuracy is improved as the number of principal components is increased from 5 to 14. Principal component count increases beyond this value do not improve accuracy. To assess the benefits of PCA, the experiments described in Section 3.5 were performed using the fourteen PCA values in place of the thirty-four P^i values as part of the $\mathbf{x^i}$ vector in (3.11) after model retraining. In all our experiments, the estimated temperature results were nearly identical, indicating the negligible effect of performance counter correlation. As a result, the rest of our reported results use P^i values in (3.11) rather than PCA values.

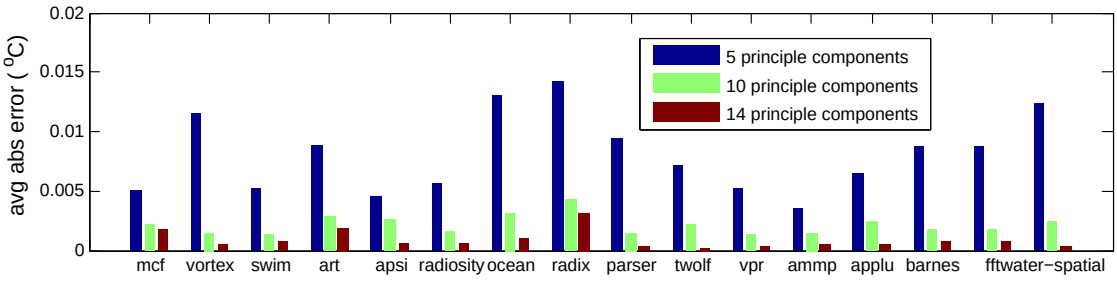


Figure 3.11: Prediction accuracy comparison for different numbers of principal components used in the model.

3.7 Dynamic Model for Changing Cooling Conditions

In contemporary computer systems, a variety of cooling technologies (e.g. fans, liquid) are used to efficiently remove heat from the microprocessor and protect it from overheating. Often, the amount of cooling (e.g. fan speed, fluid flow speed) is dynamically adjusted based on a processor's thermal situation. As a result, the β parameters determined through training in Section 3.2 are only valid for a specific cooling amount. In systems with multiple cooling levels, effectively (3.11) for thermal sensor i can be restated as:

$$\Delta T^i = \mathbf{x^i} \cdot \boldsymbol{\beta}(s)^i \tag{3.22}$$

where β(s) values have been determined using the training method described in Section 3.3 for a specific cooling amount (e.g. fan speed). In this case, β(s) training (Section 3.3) is performed at each cooling amount, s. In performing calibration, the appropriate set of β(s) values can be used based on the current cooling amount. The drawback of this method is that it increases storage cost incurred by storing multiple model parameters.

Although this multiple training approach can be effectively used for multiple, discrete cooling amounts, it does not address the issue of a large number of possible cooling amounts. The model to dynamically adapt $\boldsymbol{\beta}$(s) for an s which was not

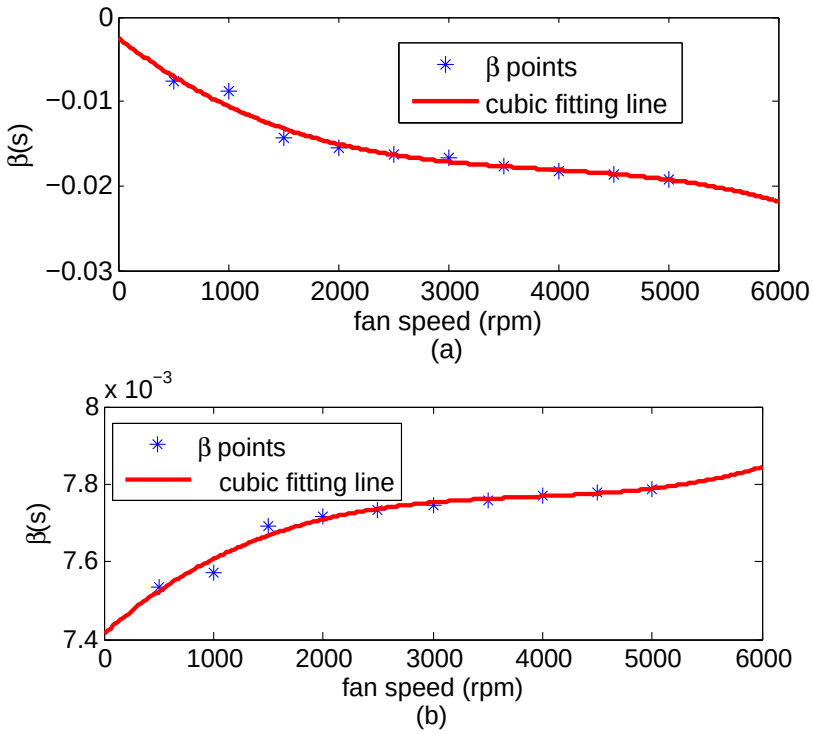


Figure 3.12: (a) One β model using cubic fitting for integer module (block 8 in Fig. 3.4) . (b) One β model using cubic fitting for ALU module (block 12 in Fig. 3.4).

previously trained can be achieved using polynomial fitting. Fig. 3.12 shows known β parameters determined through training for integer scheduler as blue stars. $\beta(s)$ can be determined by a cubic fitting using Equation. (3.23).

$$\beta(s) = a_0 + a_1 s + a_2 s^2 + a_3 s^3 \tag{3.23}$$

The β for the integer scheduler module in Fig. 3.12(a) decreases with increasing fan speed based on its principal component. The β for the integer scheduler module in Fig. 3.12(b) increases with higher fan speed. Since the first principal component is negatively related to the temperature of ALU, the β increases even with improved cooling The example shown in Fig. 3.12 exhibits a cubic fit. By building $\boldsymbol{\beta}$(s) models, only a few $\boldsymbol{\beta}$ parameters for specific s cooling amounts must be stored, saving storage space.

3.8 Summary

In this chapter, two linear models which are suitable for on-line calculation are employed to estimate the temperatures of multiple sensor locations on the silicon die. The estimated sensor and actual sensor thermal profiles show a very high similarity with correlation coefficient ~ 0.9 for most tested benchmarks.

Unlike previous techniques, we *directly* use information from performance counters for temperature estimation rather than using power consumption as an intermediate value for conversion between performance counter information and estimated temperature. This direct approach reduces run time and eliminates the need to estimate per-functional unit power consumption. The proposed estimation model can be adapted to changing cooling conditions via parameter modeling.

CHAPTER 4

MULTI-SENSOR COLLABORATIVE CALIBRATION

On-chip digital thermal sensors, such as ring oscillators, often are affected by noise due to process variation and gradual device wear-out. As a result, their readings may drift away from accurate values. In this chapter we show that sensor readings can be combined with estimates derived using the performance counter approach from the previous chapter to generate more accurate *corrected* temperature readings. Although it is expected that sensor readings will track corrected readings for long periods of time, if the reading for a specific sensor significantly differs from its corrected readings for a number of samples, the sensor can be recalibrated.

To determine accurate temperature values, estimated temperature values obtained in Chapter 3 and readings taken from sensors are merged via a Multi-Sensor Collaborative Calibration Algorithm (MSCCA) and Δ-MSCCA. These algorithms can be executed at run time using a block of consecutive sensor readings. *Corrected temperature* values obtained from the algorithm are then used to adjust the mapping of thermal sensor parameters to temperature readings. A Bayesian technique integrated into MSCCA utilizes the implicit physical proximity of the estimated temperature locations (spatial correlation) to correct sensor reading errors.

4.1 Dynamic Calibration Strategy: Approach Overview

Fig. 4.1 shows our strategy for dynamic on-chip sensor calibration. This flow can be broken down into four steps, one which is performed once during the design or post-silicon phase and three which are performed repetitively at run time.

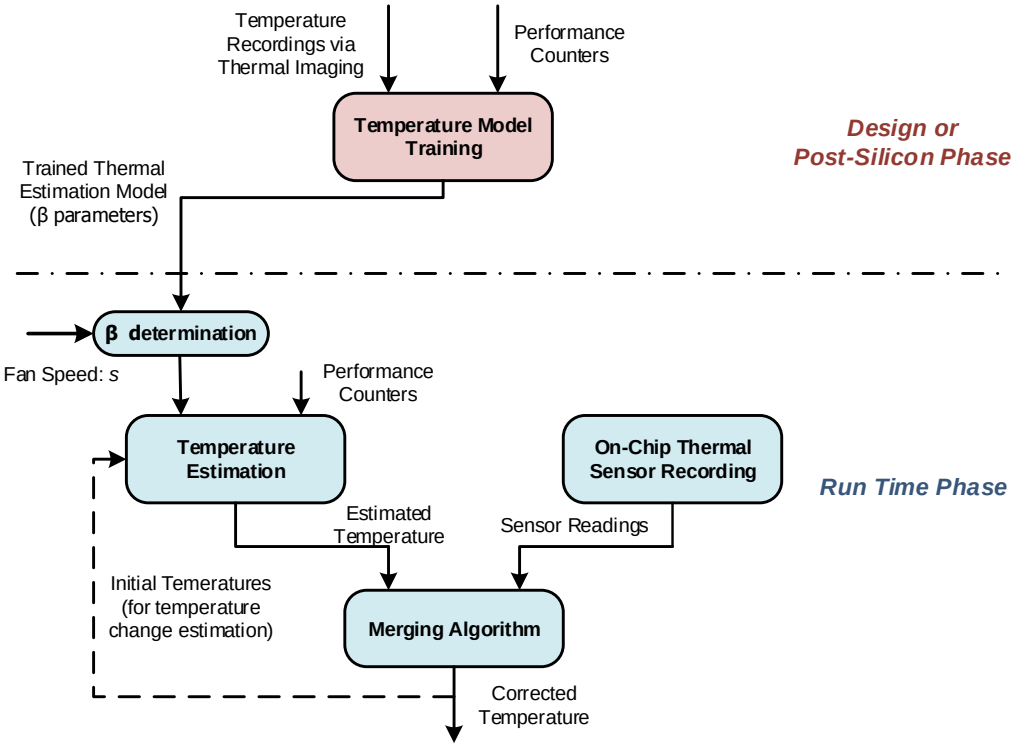


Figure 4.1: Our dynamic calibration scheme for on-chip thermal sensors

1. **Temperature model training** - In the design or post-silicon phase, a thermal estimation model is developed based on accurate temperature recordings through thermal imaging technology and system statistics from performance counters. The model training outputs a set of parameters called β parameters. These β parameters define the relationship between performance counter values and estimated temperatures.

2. **Temperature estimation** - The β parameters are used in a series of linear equations to convert performance counter values to temperature estimates. Although useful, temperatures obtained from this model often do not meet accuracy requirements since performance counters cannot capture all on-chip thermal details precisely.

3. **On-chip thermal sensor recording** - Potentially noisy thermal sensor readings are collected from on-chip thermal sensors.

4. **Merging algorithm** - To calibrate a thermal sensor, we combine thermal estimations and sensor readings using a Bayesian-based fusion algorithm. This *MSCCA algorithms* generate *corrected* temperature values and identifies how much a thermal sensor should be adjusted in calibration, if needed. Note that the initial temperature feedback (dashed line) only effective for Δ-MSCCA because no initial temperature is assumed for MSCCA.

In Chapter 3, temperature estimation and model training are considered. This chapter describes our merging algorithms and the techniques used for on-chip sensor readings.

4.2 Problem Formulation

Bayes' theorem presents the relationship between a known (priori) probability distribution and a posterior probability distribution; it is widely used for parameter inference. The unknown parameter distribution is represented by $p(\theta)$, which represents the prior knowledge of θ and the distribution of random variable x for a given θ is $p(x|\theta)$. The distribution of θ after an observation can be calculated using the following formula.

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} \tag{4.1}$$

For our sensor calibration problem, the actual temperatures of sensors are unknown attributes which are estimated by Bayesian inference. The following definitions are used for the formulation of the sensor calibration problem.

- $\mathbf{t}$ and $p(\mathbf{t})$: the random vector of the actual temperatures and its probability distribution;

- $\mathbf{r}$ and $p(\mathbf{r})$: the random vector of the thermal sensor readings and its probability distribution;
- $\mathbf{e}$ and $p(\mathbf{e})$: the random vector of the estimated temperatures and its probability distribution;
- $\mathbf{\Sigma_r}$: the covariance matrix of the random vector $\mathbf{r}$;
- $\mathbf{\Sigma_e}$: the covariance matrix of the random vector $\mathbf{e}$;
- $p(\mathbf{r}|\mathbf{t})$: the probability distribution of the sensor readings given the actual temperatures (sensor noise distribution);
- $p(\mathbf{t}|\mathbf{r})$: the probability distribution of the actual temperatures given the sensor readings (statistical inference after an observation);

The probability distribution of the actual temperature $\mathbf{t}$ is given by the following formula. Note that $\mathbf{t}$ and $\mathbf{r}$ are multivariate random variables.

$$p(\mathbf{t}|\mathbf{r}) = \frac{p(\mathbf{r}|\mathbf{t})p(\mathbf{t})}{p(\mathbf{r})} \tag{4.2}$$

In the above equation, the *priori* knowledge of the actual temperature distribution is $p(\mathbf{t})$, which can be obtained via thermal estimation discussed in Chapter 3. So, the *priori* knowledge is $p(\mathbf{e})$. The *posteriori* inference of an actual temperature after an observation is $p(\mathbf{t}|\mathbf{r})$.

Since the temperature change rate is less than 0.1^oC per millisecond [91], we assume that the actual temperature keeps constant during a 1 millisecond period. For today's high performance processors, this corresponds to several million clock cycles and enough sensor and performance counter readings can be obtained to perform the

calibration algorithm. The corrected temperature is defined as the expected value of the conditional random vector $\mathbf{t}|\mathbf{r}$ which is calculated by the following equation.

$$\boldsymbol{\mu}_t = E(\mathbf{t}|\mathbf{r}) = \int \mathbf{t} \times p(\mathbf{t}|\mathbf{r}) d\mathbf{t} \tag{4.3}$$

The covariance matrix of the corrected temperature is given as:

$$\boldsymbol{\Sigma}_t = E[(\mathbf{t} - \boldsymbol{\mu}_t)(\mathbf{t} - \boldsymbol{\mu}_t)'] \tag{4.4}$$

The probability distribution can be characterized by collecting a time series of sensor readings.

Because there are many factors, such as supply voltage, process variation and ambient temperature fluctuation which impact the sensor readings, the noise of a thermal sensor follows a Gaussian distribution, i.e. $\mathbf{r}|\mathbf{t} \sim \mathcal{N}(\mathbf{t}, \boldsymbol{\Sigma}_\mathbf{r})$. In the Gaussian case, (4.3) and (4.4) have closed form representations as follows [31].

$$\boldsymbol{\mu}_\mathbf{t} = \boldsymbol{\mu}_\mathbf{e} + \boldsymbol{\Sigma}_\mathbf{e}(\boldsymbol{\Sigma}_\mathbf{e} + \boldsymbol{\Sigma}_\mathbf{r})^{-1}(\mathbf{r} - \boldsymbol{\mu}_\mathbf{e}) \tag{4.5}$$

$$\boldsymbol{\Sigma}_\mathbf{t} = \boldsymbol{\Sigma}_\mathbf{e} - \boldsymbol{\Sigma}_\mathbf{e}(\boldsymbol{\Sigma}_\mathbf{e} + \boldsymbol{\Sigma}_\mathbf{r})^{-1}\boldsymbol{\Sigma}'_\mathbf{e} \tag{4.6}$$

Thus, the expected actual temperature given $\mathbf{r}$, ($\mu_\mathbf{t}$), and its covariance ($\boldsymbol{\Sigma}_\mathbf{t}$) can be determined directly from sensor readings and estimated temperature values from performance counters.

4.3 MSCCA and Δ-MSCCA

The Bayesian inference of the actual temperature is used to perform calibration on m thermal sensors once per every p readings (time instances). The goal of the MSCCA algorithm is to determine the corrected temperature ($\mu_\mathbf{t}$) and covariance

Algorithm 1 Multi-Sensor Collaborative Calibration Algorithm – MSCCA

1: Initialize $\boldsymbol{w} \leftarrow \mathbf{0}$.
2: **while** Invocation count $\leq \frac{p}{l}$ **do**
3: Store sensor readings in $\mathbf{R}$ matrix for next l time instances.
4: Store estimated temperatures determined from approach in Section 3 in $\mathbf{E}$ matrix.
5: Adjust $\mathbf{R}$ matrix by adding offset $\boldsymbol{w}$ to each row.
6: Adjust $\mathbf{E}$ matrix by subtracting a constant value c.
7: The vector $\mathbf{r}$ is the columnwise mean of $\mathbf{R}$.
8: The vector $\boldsymbol{\mu}_e$ is the columnwise mean of $\mathbf{E}$.
9: Calculate the covariance matrices $\boldsymbol{\Sigma}_\mathbf{r}$ and $\boldsymbol{\Sigma}_\mathbf{e}$.
10: Perform Bayesian inference using Equations (4.5) and (4.6), and get the corrected temperature $\boldsymbol{\mu}_\mathbf{t}$.
11: $\boldsymbol{w} \leftarrow \boldsymbol{\mu}_\mathbf{t}$ - $\mathbf{r}$.
12: **end while**

($\boldsymbol{\Sigma}_\mathbf{t}$) for each temperature sensor once per l samples. Two calibration algorithms are described for the absolute and incremental temperature estimation respectively.

MSCCA: The algorithm is shown in Algorithm 1. The steps described in Section 4.2 are performed multiple times per calibration period to refine intermediate results to a final value. In the following description, each algorithm *invocation* is performed on readings from l consecutive time instances. A total of $\frac{p}{l}$ invocations are performed per calibration. The calibration offset for sensor i, w_i, is defined as the difference between the corrected temperature and sensor reading at a specific time point. The $\mathbf{w}$ vector contains all w_i values. The $\mathbf{R}$ matrix ($l \times m$) is initialized with raw sensor data in each invocation and each column represents a time series of readings from one sensor. The $\mathbf{E}$ matrix ($l \times m$) is initialized with raw estimated temperatures in each invocation and each column represents a time series of estimation for one sensor. Step 5 updates the sensors' readings by adding the $\mathbf{w}$ offsets from the previous invocation and Step 6 adjusts the estimated temperatures since these temperatures have systematic error, as mentioned in the previous chapter. The value c is the mean value of all elements in $\mathbf{R}$. Overall, algorithm 1 shows the multi-sensor collaborative calibration algorithm

using Bayesian inference over multiple invocations until all p readings for m sensors have been processed.

Δ-MSCCA: The major difference between MSCCA and Δ-MSSCA is the additional corrected temperature feedback for Δ-MSSCA. Since Δ-MSSCA needs initial temperature to estimate the temperature changes, current corrected temperatures are used as initial temperatures for the calculation of the next block. As shown in Fig. 4.1, temperature change estimation requires an initial temperature profile of the silicon die which may not be available at run-time. For initialization of the estimation approach, it is possible to assign an arbitrary temperature to each thermal sensor or to use thermal sensor readings as initial temperatures. As seen in Algorithm 2, during each of l samples, temperature sensor readings $\mathbf{r}$ and performance counter values are read. For the sample, the sensor readings from all temperature sensors form a row in an $\mathbf{R}$ matrix (line 8). Additionally, the performance counter values are converted to estimated temperature changes for each sensor using (3.11) (line 5). These temperature changes are added to the estimated temperatures from the previous sample (line 6) and the results for each sensor is stored in an $\mathbf{E}$ matrix (line 7).

After processing l samples, corrected temperature values for each temperature sensor are determined (line 14) using (4.5) and (4.6). Vectors $\mathbf{r}$ and $\mu_{\mathbf{e}}$ used in the corrected temperature calculation are determined from the columnwise mean of the $\mathbf{E}$ and $\mathbf{R}$ matrices (lines 11 and 12). As noted in previous chapter and shown in Fig. 3.7, the use of performance counters to estimate temperature shows a strong relative match, although an absolute offset for the actual temperature is often present. To address this issue, a per-sensor offset value $\mathbf{w}$ is added to each $\mathbf{r}$ reading. Although we found that $\mathbf{w}$ values are constant for each sensor across time and across benchmarks, the values are recalculated in the algorithm for consistency. In our experimentation, calculation was performed over p total samples with l samples per invocation. A total of $\frac{p}{l}$ invocations are used for the p sample set.

Algorithm 2 Δ-based Multi-Sensor Collaborative Calibration Algorithm – Δ-MSCCA

1: Initialize $\boldsymbol{w} \leftarrow \mathbf{0}$.
2: Initialize temperature profile
3: **while** Invocation count $\leq \frac{p}{l}$ **do**
4: **for** $i = 0; i < l; i{+}{+}$ **do**
5: Estimate Δ temperatures determined from performance counters using (3.11)
6: Add Δ temperatures to previous corrected temperatures and get updated temperature profile.
7: Store the updated temperatures as a row in $\mathbf{E}$ matrix.
8: Store sensor readings as a row in $\mathbf{R}$ matrix.
9: **end for**
10: Adjust $\mathbf{R}$ matrix by adding offset $\boldsymbol{w}$ to each row.
11: The vector $\mathbf{r}$ is the columnwise mean of $\mathbf{R}$.
12: The vector $\boldsymbol{\mu}_e$ is the columnwise mean of $\mathbf{E}$.
13: Calculate the covariance matrices $\boldsymbol{\Sigma}_{\mathbf{r}}$ and $\boldsymbol{\Sigma}_{\mathbf{e}}$.
14: Perform Bayesian inference using Equations (4.5) and (4.6), and get the corrected temperature $\boldsymbol{\mu}_{\mathbf{t}}$.
15: $\boldsymbol{w} \leftarrow \boldsymbol{\mu}_{\mathbf{t}}$ - $\mathbf{r}$.
16: **end while**

4.4 Performance and Storage Evaluation

This section analyzes the computational complexity of both MSCCA approaches and compares it with the complexity of using Kalman filtering to generate corrected temperatures for thermal sensors. Although a full discussion of the KF algorithm for temperature estimation can be found in [116], we provide a brief overview of the required operations here. The KF approach requires two estimation steps to convert performance counter values to estimated temperature. First, the power consumption of individual functional units is determined using a linear set of equations which have been determined via linear regression [78]. Per-functional unit power values are then converted to estimated temperature via a second set of linear equations [116] whose derivation require the difficult approximation of thermal resistance and capacitance for on-chip functional units. To develop corrected temperature values from estimates and sensor readings, KF then uses cross correlation with previously-determined noise values to merge the estimates and readings together. Unlike our

approach, where corrected temperatures are generated every l samples, KF requires corrected temperature evaluation for every sample, a significant time penalty. Thus, our approach has two significant practical benefits versus KF:

- Estimated temperatures are determined directly from performance counter values rather than requiring power as an intermediate value. The elimination of power as a transition metric also eliminates the need for complicated thermal resistance and capacitance calculation.
- MSCCAs requires many fewer operations and can be performed less frequently reducing run time.

4.4.1 Computational Complexity

The computational cost for MSCCAs and KF approaches (not considering model training which takes place only once at design time) can be broken down into two parts: temperature (or power) estimation and temperature correction. The computational cost of the power estimation for KF and temperature estimation for MSCCAs is the time required to calculate a linear combination of scaled performance counter values. As a result, the estimation complexity is $O(np)$, where n is the number of performance counters and p is the number of sample sets. The *estimation* column in Table 4.1 shows the number of operations needed to perform this estimation (thermal for MSCCAs and power for KF). There are $n = 34$ performance counters included in our linear regression model, so we specify complexity in terms of this value.

The MSCCAs approach stores samples and performs Bayesian estimation once per l time steps. Table 4.1 shows the number of operations performed for p sets of readings. For the MSCCAs, l time instances (sets) of readings per invocation are used. As noted in the previous subsection, calibration can be simultaneously performed for multiple consecutive sensor readings for each sensor in one invocation. In our implementation, there are m=24 thermal sensors, so the matrix dimensions of

Table 4.1: Operations required by MSCCAs and Kalman filtering for p sets of sample readings for 24 thermal sensors

Operation	Estimation	MSCCAs Approach		KF Approach	
		correction	total	correction	total
scalar addition	34p×24	$\frac{p}{l}(444l - 48)$	$\frac{p}{l}(1404l - 48)$	0	$816p$
scalar multiplication	34p×24	$\frac{p}{l}(300l + 48)$	$\frac{p}{l}(1260l + 48)$	0	$816p$
matrix addition	0	$\frac{p}{l}$	$\frac{p}{l}$	$2p$	$2p$
matrix multiplication	0	$\frac{p}{l}$	$\frac{p}{l}$	$10p$	$10p$
matrix-vector multiplication	0	$\frac{p}{l}$	$\frac{p}{l}$	$3p$	$3p$
matrix inversion	0	$\frac{p}{l}$	$\frac{p}{l}$	p	p
vector addition	0	$\frac{p}{l}$	$\frac{p}{l}$	$3p$	$3p$

Σ_r and Σ_e are 24×24. If the matrix operations are converted to scalar operations, there are about $150,000p$ additions and $150,000p$ multiplications required for the KF method. In our method, the numbers of additions and multiplications are about $\frac{p}{l}(1404l + 14,000)$ and $\frac{p}{l}(1260l + 14,000)$.

In contrast, KF-based algorithms predict and update the temperature for *each* set of sample readings, resulting in more matrix operations. In Table 4.1, the $34p$ scalar operations for KF represent the operations to convert power estimates to temperature estimates for a single sensor. The remaining operations represent merging computations for temperature estimates and temperature sensor readings.

4.4.2 Memory Overhead

Since samples must be stored in matrices for a period of time before they are processed, MSCCAs does require more memory usage than the KF-based approach. In general, the KF approach does not require storage for the power estimates and sensor

reading samples. At each time step, the thermal sensor samples (sensor readings) and temperature estimates determined from power estimations are used to update temperatures, and then these samples are thrown away. The MSCCAs approach must store thermal sensor samples $\mathbf{R}$ and temperature estimates $\mathbf{E}$ for l samples in memory until the next MSCCAs evaluation. As a result, the memory complexity for the KF approach is $O(1)$ and for MSCCAs is $O(ml)$. In our experiments, l is several hundred and $m = 24$ sensors are used. So the memory storage of samples is around several kilobytes.

4.5 Implementation Issues

The use of thermal calibration raises concerns about overburdening the hardware and operating system of the target processor. However, the nature of our calibration approach and recent trends in on-chip monitoring for microprocessors lessen this concern. In general, thermal sensor calibration is expected to be performed once every few seconds, rather than milliseconds. In Section 4.6, it is shown that algorithm execution time is on the order of tens of milliseconds for evaluation that is performed every ten seconds. This overhead limits the operating system and processor-level power and temperature impact of the algorithm itself.

Independent of this overhead limit, recent trends indicate that microprocessors increasingly include *dedicated* circuitry to perform monitoring and monitor data processing which is separate from the main OS/processor compute platform. For example, IBM EnergyScale [35] uses temperature and critical path monitors along with a microcontroller for sensor data processing. Intel's Active Management Technology provides a separate on-chip communications channel and controller to monitor device operation and control system responses at the operating system level. Often, these monitoring and monitor data processing infrastructures can be quite small compared to the main processing infrastructure (e.g. 0.2% of overall processor area [119]),

Table 4.2: Benchmarks used in experimental evaluation

Suite Name	Set I	Set II
SPEC2000 CINT	*bzip2, crafty, gzip*	*gap, gcc*
SPEC2000 CFP	*equake, mgrid*	*mesa, sixtrack, wupwise*
SPLASH2	*volrend, cholesky, raytrace*	*lu, water-nsquared, fmm*
Suite Name	Set III	Set IV
SPEC2000 CINT	*mcf vortex*	*parser, twolf, vpr*
SPEC2000 CFP	*swim, art, apsi*	*ammp, applu*
SPLASH2	*radiosity, ocean, radix*	*barnes, fft, water-spatial*

limiting system performance impact. These effects can be weighed against the benefits of a more accurate DTM approach due to improved thermal sensor calibration. Additionally, the processing of monitor data often takes place in an area which is isolated, limiting self heating issues in the main processor due to the monitoring data processing.

4.6 Experiments and Results

For training and verification of our new calibration approach and for comparison to KF, we use the applications listed in Table 4.2 from the SPEC2000 and SPLASH-2 benchmark suites. SPEC2000 is an industry-standardized CPU-intensive benchmark suite which include both integer and floating point applications. To diversify the test benchmarks, we mixed SPEC2000 and SPLASH-2 in the same test sets. These benchmarks were randomly divided into four sets: Set I, Set II, Set III and Set IV. Our thermal estimation model (β values) was trained using benchmark sets I and II. Our models and algorithms are verified with the remaining sets. Although SESC supports multi-threaded simulation for multi-core systems, all benchmarks are configured to run in a single processor in our experiments.

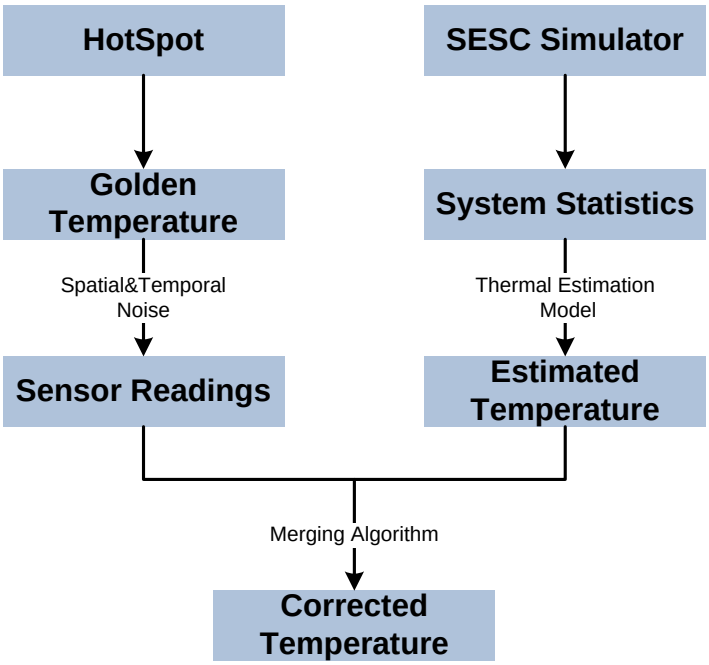


Figure 4.2: Sensor data and thermal estimation merging scheme. Sensor readings are artificially generated shown in the left flow; Estimated temperature from performance counter are obtained by the right flow.

4.6.1 Methodology

Simulation Scheme: The HotSpot simulation tool, which takes power traces from SESC, target processor geometry and material parameters as inputs, is used to generate accurate "golden" temperatures. As mentioned in the previous subsection, it is assumed that the HotSpot generated temperature values are the *actual* temperatures considering the sophisticated thermal diffusion model implemented by HotSpot (Fig. 4.2).

Spatial and Temporal Noise: Variations in sensor accuracy across temperature sensors on the die (spatial noise) is mainly caused by process variation which is relatively static, so we consider spatial noise to be constant for short time periods (several hours). Unlike spatial noise, variations in a specific sensor's accuracy over time (temporal noise) is caused by environmental effects like voltage and ambient temperature fluctuation, so its value varies for each temperature sample.

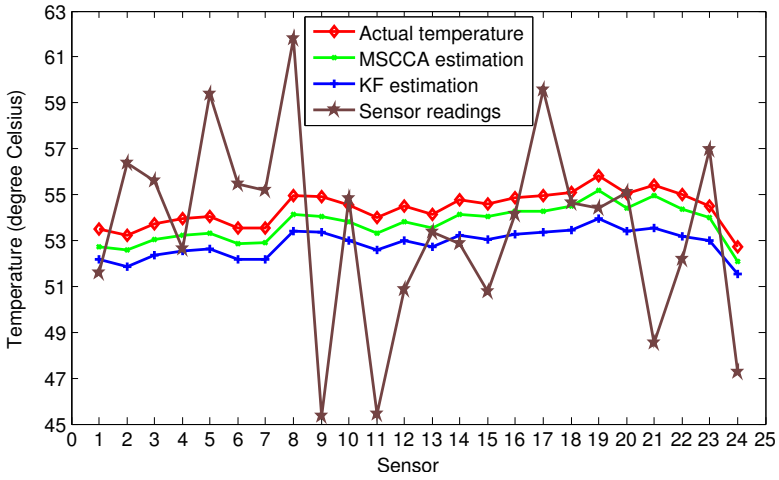


Figure 4.3: The thermal profile of the processor for one time instance of the *lu* benchmark. Each point on the horizontal axis represents a single sensor located in a block in Figure 3.

4.6.2 Effectiveness Verification

In a first series of experiments, the thermal profiles for the AMD Athlon 64 were determined using Algorithm 1 and 2. For these experiments, the standard deviation of temporal and spatial noises were both set to 4^oC and 6000 total time instances of readings were processed. MSCCA uses l=100 time instances per invocation.

MSCCA: In Figure 4.3, we demonstrate the thermal profile of the AMD Athlon 64 processor for the *lu* benchmark after the 2000th time instance. The horizontal axis represents thermal sensors for each functional block in Fig. 3.4. In the figure, the actual temperature, sensor readings, and corrected temperature from the KF based implementation and from MSCCA are plotted. Both constant spatial noise due to process variations and temporal noise (shown in the plot of the sensor readings) are taken into account in our simulation. The first observation is that both methods effectively reduce the sensor reading errors: the sum of the square errors of all sensors for the corrected readings is much smaller than that of sensor readings. The second observation is that the thermal profile is recovered after synthesizing two data

sources (the estimated temperature and sensor readings in our case, the statistical characteristics of the power dissipation and sensor readings in the KF case).

Δ-MSCCA: In Fig. 4.4, we demonstrate the thermal profile of the AMD Athlon 64 processor for the *bzip2* benchmark. In the figure, the actual temperature, sensor readings (only spatial noise is shown for clarity but the experiment is performed with both spatial and temporal noises), corrected temperature from the KF-based implementation, and corrected temperature from Δ-MSCCA using thermal estimates from performance counters are plotted. It is apparent that both methods effectively reduce the sensor reading errors: the sum of the square errors of all sensors for the corrected readings is much smaller than that of sensor readings.

Although corrected temperatures initially differ from actual temperatures due to incorrect initial estimates of temperature, the corrected temperatures determined by Δ-MSCCA quickly converge. In both the Δ-MSCCA and KF cases, the thermal profile is recovered after synthesizing two data sources (the estimated temperature and sensor readings in the Δ-MSCCA case, the statistical characteristics of the power dissipation and sensor readings in the KF case). The Δ-MSCCA case has the benefit of faster calculation (contrasted in Section 4.6.6) and a much simpler model training process (no power-to-temperature model needed).

4.6.3 Temperature Tracking Using Δ-MSCCA

Fig. 4.5 shows the temperature tracking results for three thermal sensors: Sensor 8 (integer scheduler), Sensor 15 (L1 data cache) and Sensor 20 (floating point scheduler). The results from other sensors are similar. For each sensor, four curves are plotted in the figure: the actual temperature, Δ-MSCCA corrected temperature, noisy sensor readings and noisy sensor reading with temporal noise pruned out. Since we assume that the spatial noise does not change in a short time period, the green curve has

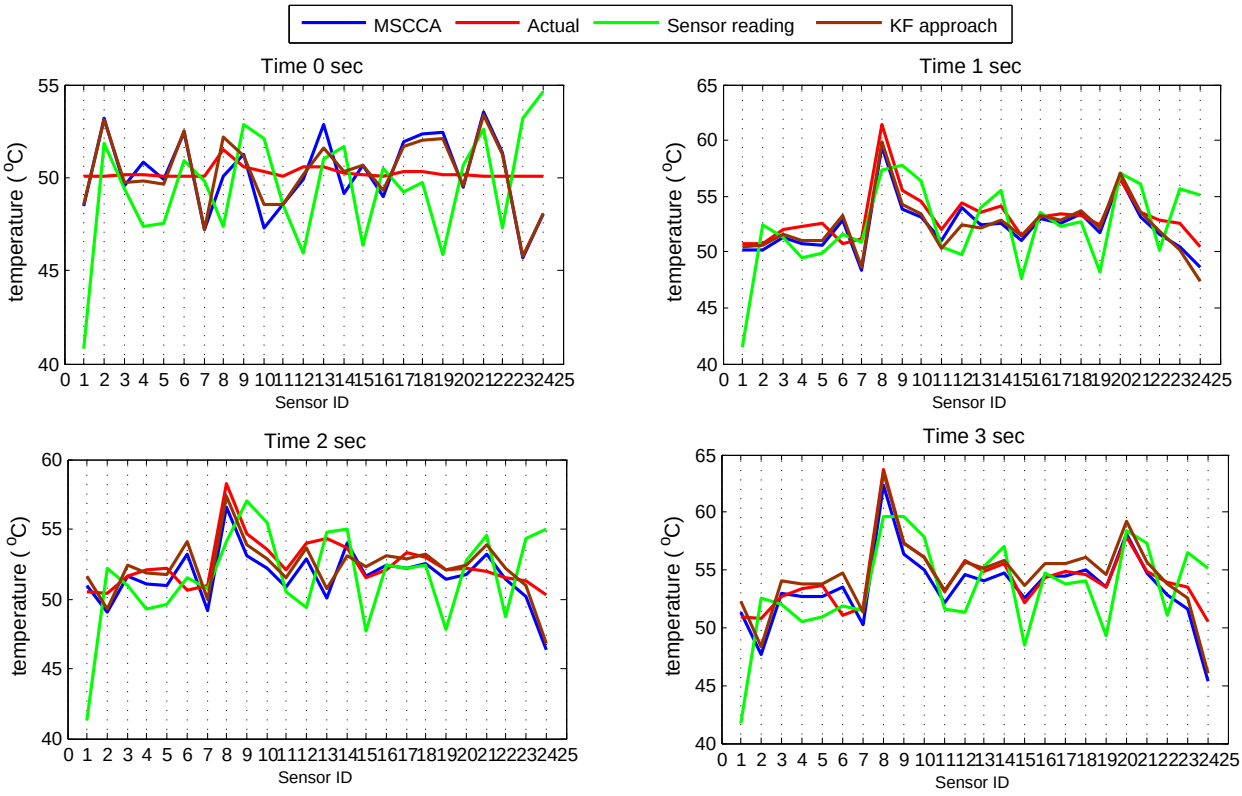


Figure 4.4: The estimated and actual processor temperature profile at four time instances.

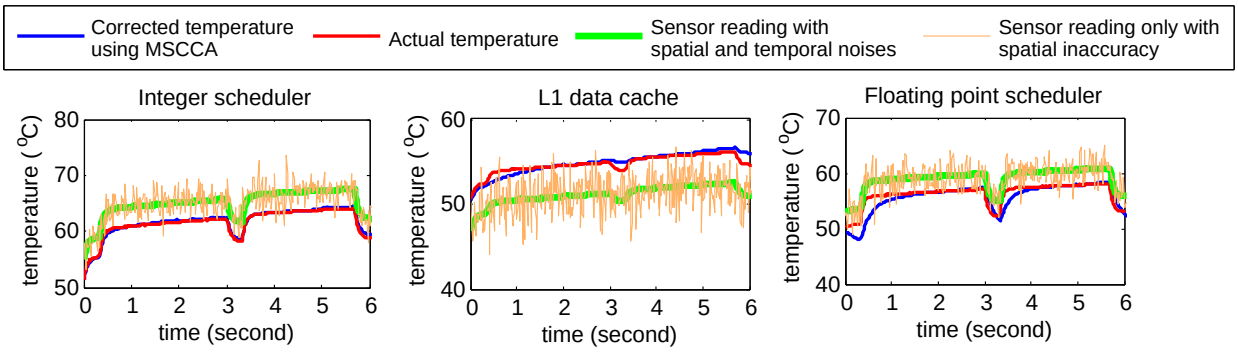


Figure 4.5: Temperature tracking over 6 seconds for thermal sensor 8 (integer scheduler), 15 (L1 data cache) and 20 (floating point scheduler) in Fig. 3.4 running the *twolf* benchmark.

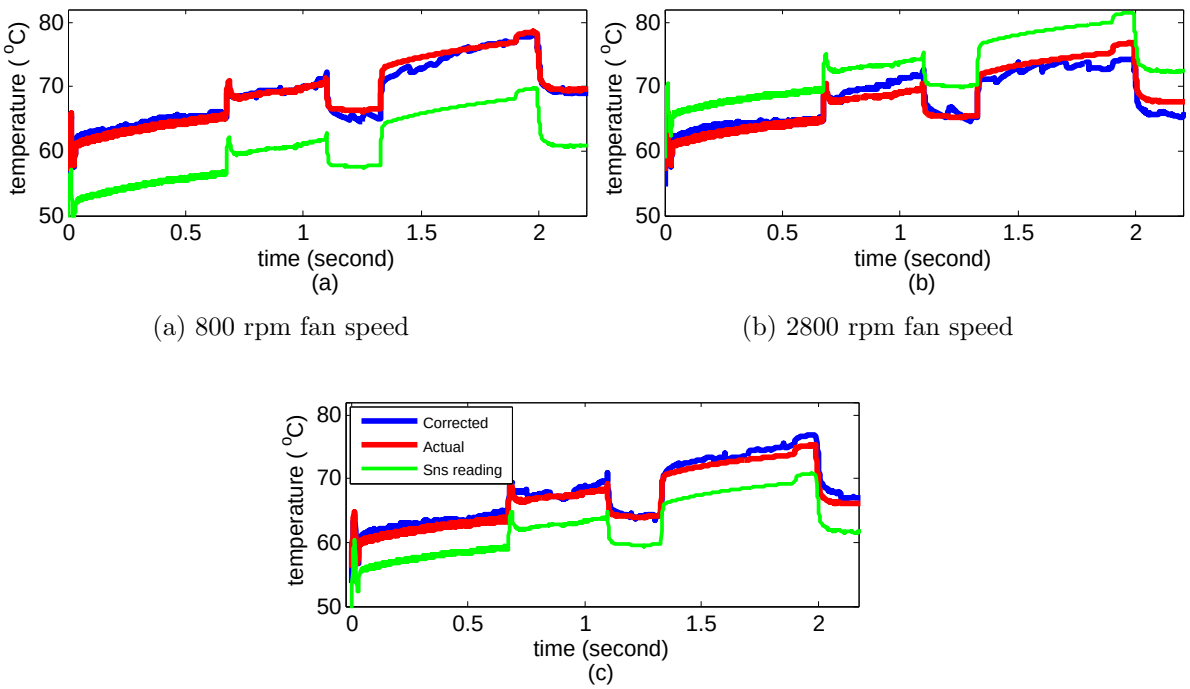
a constant offset from red curve. The simulation lasts for 6 seconds and the initial temperatures for all sensors are randomly generated.

Although the actual initial temperatures for the three sensors are not 50^oC, Δ-MSCCA estimation results were generated using this initial value. Estimated temperature values converge to the actual temperature over time. The figure indicates that sensors 8 and 20 show good calibration accuracy since the estimation curves are closer to the actual curves than the sensor reading curves.

Fig. 4.6 shows temperature tracking results with different fan speeds, 800, 2800 and 4800 rpm respectively. A different set of $\boldsymbol{\beta}$ parameters are used for each fan speed, as discussed in Section 3.7. The thermal model can effectively track the actual temperature values for all fan speed values, as expected.

4.6.4 Estimation Error Comparison

The experiments in the previous subsection qualitatively show the effectiveness of dynamic sensor calibration using data fusion. In this section, the error of the both MSCCA approaches are quantitatively evaluated. Fig. 4.7 shows the average absolute error and the standard deviation of the errors for original sensor readings, MSCCA, Δ-MSCCA and the KF approach. The MSCCA results in Fig. 4.7 were determined



(a) 800 rpm fan speed

(b) 2800 rpm fan speed

(c) 4800 rpm fan speed

Figure 4.6: Temperature tracking with various fan speeds running the benchmark *volrend*

using the same training set as the Δ-MSCCA algorithm. The spatial noise added to sensor readings is Gaussian with standard deviation 6^oC and the temporal noise is Gaussian with 4^oC standard deviation.

In Fig. 4.7(a), the average absolute error is reduced by 5^oC (from 6^oC to 1.2^oC) with respect to the original sensor readings. In Fig. 4.7(b), the standard deviation of the error is reduced by a factor of 10 (from 3^oC to 0.2^oC) from the original sensor readings with limited computational effort.

4.6.5 Impact of Sensor Reading Noise for Δ-MSCCA

The standard deviation of the errors of the corrected temperature increases as the noise of the sensor readings becomes larger. The experiments in Section 4.6.4 were repeated, this time with varying amounts of noise in the sensor readings. Experiments

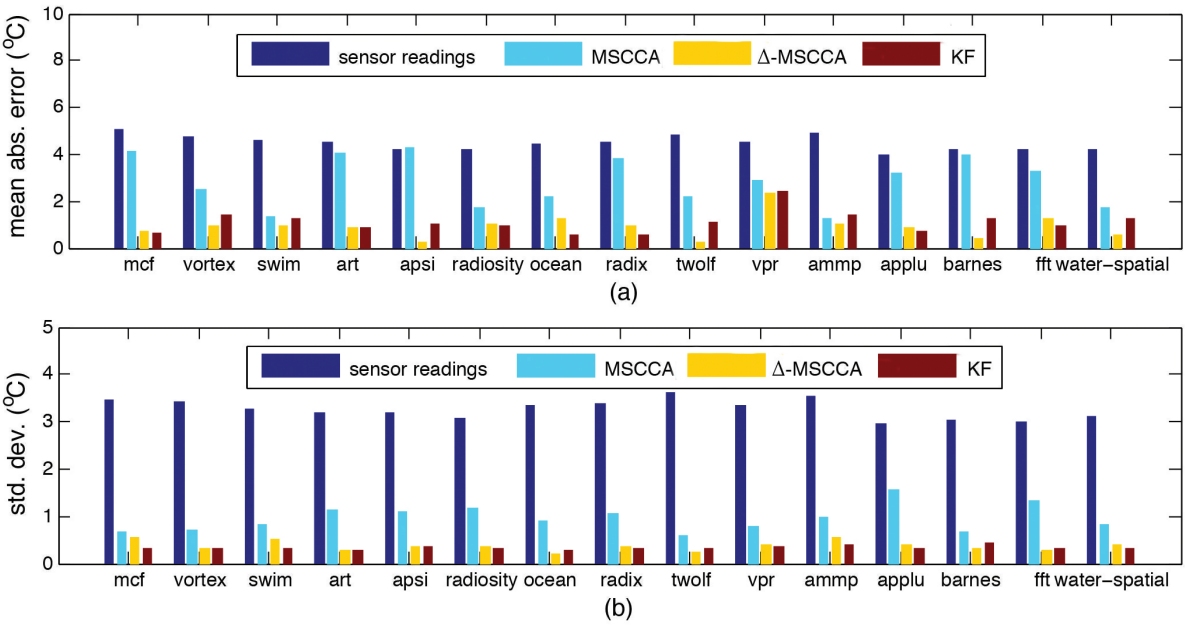


Figure 4.7: (a) The average absolute error for sensor readings, temperatures generated with MSCCA, and with KF (b) The standard deviation of errors for sensor readings, temperatures generated with MSCCA, and KF approaches.

of 10,000 time instances each were performed. To better evaluate the effect of noise, one set of 10,000 random noise values was determined for each noise amount (e.g. each column in Table 4.3). These values were added to read values and the results are used for comparison across configurations. The table shows the standard deviations of corrected temperatures for sensor readings with four different sensor noise levels. As predicted, the less accurate the sensor readings are, the larger error seen in the corrected temperature.

4.6.6 Run Time Comparison

Table 4.4 shows the total run time including both temperature estimation and correction for both MSCCA and KF approaches. The total number of samples is 10,000, collected in 10 seconds. The total run time per sampling interval for MSCCA is < 0.004 seconds for MSCCA and about 0.2 seconds for the KF-based approach. As seen in the table, the run time of MSCCA decreases as the number of time instances

Table 4.3: The standard deviation of the error for the corrected temperatures over 10,000 time instances for increasing sensor error

Time instances	Std dev of sensor error ^oC			
per invocation l	2	4	6	8
100	0.1059	0.1575	0.2047	0.2798
200	0.1137	0.1663	0.2159	0.2935
300	0.1203	0.1711	0.2241	0.3040
400	0.1276	0.1732	0.2284	0.3093
500	0.1311	0.1819	0.2372	0.3187

Table 4.4: Run time comparison (in seconds) between MSCCA and KF approaches for **10000** time instances

Instances/invocation	100	200	400	500	1000
MSCCA run time	0.0387	0.0301	0.0242	0.0228	0.0169
KF run time	1.9076	1.912	1.9289	1.9133	1.8946

per MSCCA invocation increases and it remains constant for the KF approach since corrected temperature calculation is performed for each sample.

As we mentioned in Section 4.3, our approach can be inserted as system management code in an operating system. Since system tick time is normally on the order of milliseconds, the thermal estimation can be invoked when the OS core performs thread scheduling. Based on our results from Table 4.4, the time needed to do estimation calculation is < 0.004 millisecond, so the integration of the calibration approach into the OS will minimally affect system performance.

4.7 Summary

In this chapter, estimated temperatures generated using our approach from Chapter 3 and thermal sensor readings are merged using the Multi-Sensor Collaborative

Calibration Algorithm and corrected temperature readings for thermal sensors are achieved. Our strategy is evaluated using SPLASH-2 and SPEC2000 benchmarks suites. Results show that the strategy can effectively recalibrate sensor readings in response to inaccuracies caused by process variation and environmental noise. The average absolute error of the corrected sensor temperature readings is $< 1.5^oC$ and the standard deviation of error is less than $< 0.5^oC$ for tested benchmarks. Our overall estimation and correction run time is significantly reduced versus Kalman filtering (at least $50\times$ faster) to make our strategy favorable for real time implementation.

CHAPTER 5

THERMAL-AWARE TASK ALLOCATION BASED ON REINFORCEMENT LEARNING

As many-core systems scale, thermal problems becomes more complex due to increased computation and communication resources. In contemporary many-cores, the power consumption of network on chip (NoC) routers, as well as processor cores, is a significant concern. Many parallel and data intensive applications benefit from the low latency and high bandwidth of on-chip NoC communication [98] [15]. The heat dissipated by the NoC routers not only affects router temperature, but also the temperatures of neighboring cores.

Effective many-core management schedules and allocates tasks considering thermal impacts. Effective task scheduling for thermal management considers all many-core components, including NoCs. This chapter presents a task allocation technique based on reinforcement learning for many-cores. The effectiveness of the approach in reducing maximum temperature is evaluated.

5.1 Thermal Profile of a 16-Core Processor including Data Traffic

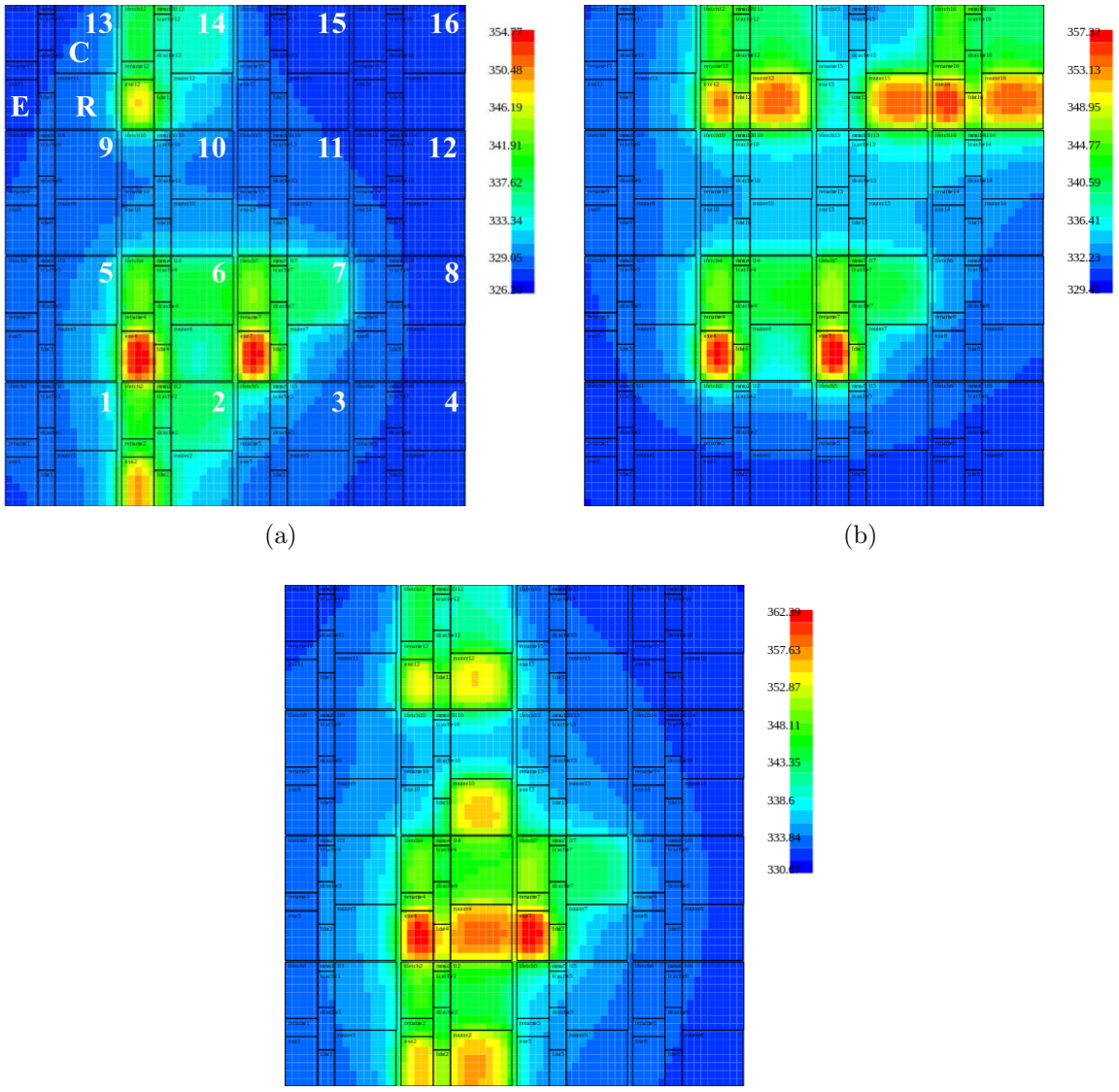
In this section, we examine the thermal profile of a 16-core processor to motivate our use of both router and core temperarature in determining task allocation. The power consumed by an on-chip network infrastructure (eg. routers) has become quite significant (up to 39% of total power [99] [27]). Shang *et al.* determined that chip temperature is impacted by thermal correlations among all on-chip components [90],

increasing maximum chip temperature. To compare thermal profiles for a many-core processor with different levels of network traffic, a 16-core system interconnected in a mesh architecture was studied. A detailed discussion of the core and router configuration and experimental methodology for thermal tracking is provided in Section 5.4. Fig. 5.1 shows three thermal maps which were generated by executing four tasks on four separate cores in the 16-core system. The Splash-2 *fft* benchmark was run on cores 6 and 7 with limited data transmission for all three scenarios. Two other tasks were allocated to cores 2 and 14 in Figs. 5.1a and 5.1c, and cores 14 and 16 in Fig. 5.1b.

In Fig. 5.1a, four tasks are running independently in their corresponding cores and the rest of cores are in an idle state. Since there is no data transmission in the network, all routers are relatively cool except those in core 2, 6 and 14 due to the spatial correlation of temperature across neighboring active processor core components.

In Fig. 5.1b, tasks running on core 14 and 16 communicate with each other. As a result, routers in core 14, 15 and 16 are in active state which leads to a rise in temperature. The thermal hot spots are present in not only execution components but also routers. Although workload intensity in core 6 and 7 is the same as scenario a), the hot spot temperature is around 3^oC higher than a) due to spatial thermal correlation.

In Fig. 5.1c, the communication link is established between core 2 and core 14 via core 6. The temperatures of all routers on the link are pushed up due to circuit routing activities. Although tasks in three scenarios incur the same power dissipation, the peak temperature in c) is 5^oC higher than that in b) and almost 8^oC higher than that in c).



(a) (b)

(c)

Figure 5.1: Thermal map for three different scenarios (a) all tasks running independently (b) tasks running on cores 14 and 16 communicate with each other (c) tasks running on cores 2 and 14 communicate with each other.

5.2 Thermal Aware Task Allocation Using Reinforcement Learning

In this section, we introduced a machine learning based algorithm which takes router and processor core thermal effects into account by "awarding" task allocations which are likely to lead to better thermal results.

The definition of a many-core task allocation problem is as follows. A many-core system is composed of a processor array and an on-chip network. Tasks are executed in parallel on multiple processors. Although a single core processor has multi-tasking capability and task ordering impacts the thermal profile due to temporal thermal effects, we focus on the spatial distribution of tasks in this dissertation. Static global optimization of this goal is infeasible as formulated in [41] since it requires the knowledge of all tasks in advance. In our case, we assume that tasks arrive stochastically and the duration time of task execution is also random, ie. one task may finish earlier than other tasks even though it is started later. Task allocation is initiated under the following two circumstances: 1) a new task arrives; 2) a thermal emergency occurs and tasks on one or more cores need to be swapped out. The assumption is close to a server which accepts stochastic workloads and dispatches them to processing cores. The goal is to find a thermal-aware allocation policy which reduces the maximum temperature on the chip to maintain healthy thermal environment.

5.2.1 Reinforcement Learning

In reinforcement learning (RL) [5], an agent (the task allocator in our case) explores an environment by taking actions and observing the resultant reward. The reward of a particular action (assigning a task to a specific core) reflects the metric to be optimized (maximum temperature). For our system, as task allocations are performed, the model used to make assignments is refined in a learning process. The task allocator gradually refines the model based on temperatures measured a time pe-

riod after an allocation is performed. Effectively, the allocator *learns* how to respond to a specific environmental condition (e.g. temperatures measured from temperature sensors) based on the results of previous assignments when cores and routers had a similar temperature profile.

Formally, reinforcement learning consists of the following

- A set of environment states: $\mathcal{S}$, in this case temperature readings from on-chip sensors;
- A set of available actions on the current state : $\mathcal{A}$, task assignments to specific cores;
- A rule to evaluate the reward for taking the action at a specific state: $\mathcal{R}$;
- The goal is to find a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, i.e. what action (assignment) should be taken at the current environmental (temperature) state.

One reinforcement learning technique is Q learning which provides a model free reinforcement learning formulation for task allocation. A *utility function* can be developed to allow for the desired mapping. The utility value is defined to find the optimal policy π in Q learning as follows.

$$Q(s,a) = E\left[\sum_{i=0}^{\infty}(\gamma^i r_{t+i}|s_t = s, a_t = a)\right] \tag{5.1}$$

The utility $Q(s,a)$ indicates the maximum temperature *rewards* (both present, $i = 0$, and future) which can be obtained by performing task assignment action a for temperature vector state s at time step t. The accumulated future reward is discounted via the discount factor γ. Therefore, the optimal policy is to take action which maximizes the utility Q. During each task assignment at time $t+1$, utility Q for temperature vector s and assignment a in (5.1) can be approximated as follows [5]:

$$Q_{t+1}(s_{t+1},a) = Q_t(s_t,a) + \alpha(\underbrace{r_t(s,a)}_{\substack{\text{current}\\ \text{reward}}} + \gamma \underbrace{\max_{a'} Q_t(s_{t+1},a')}_{\text{future reward}} - Q_t(s_t,a)) \tag{5.2}$$

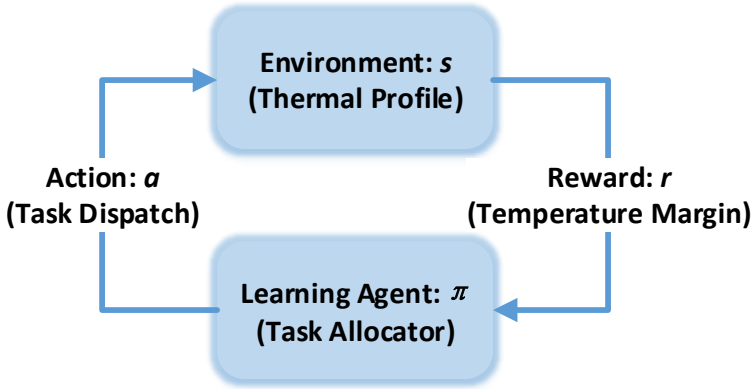


Figure 5.2: Reinforcement Learning Scheme for Thermal Aware Task allocation

In the above equation, α is the learning rate and γ is the discount rate. The new utility is the currently observed reward plus the maximum discounted future reward. The difference of utility values is used to update $Q_t(s, a)$. The iteration equation is known to converge to the optimal policy [5]. Fig. 5.2 shows the reinforcement learning iteration process. Each cycle represents one task allocation. As mentioned above, in our implementation, the task allocator is served as a learning agent and the environment state is the chip thermal profile which is read from on-chip temperature sensors. The task allocator interacts with the thermal environments through on-chip thermal sensors. The task allocation decisions also impact the thermal condition for the whole chip. After each task allocation episode, the allocator collects all system thermal information to assess the reward of the last allocation action and select cores for incoming tasks. The details on how to apply Q and update the model used to determine it are discussed subsequently.

5.2.2 Chip Thermal States

On-chip thermal sensors are often deployed in the processors to assist thermal management [84]. In our approach, a subset of thermal sensor readings from m temperature sensors are used to represent the thermal state of the silicon. In order

to apply RL in thermal-aware task allocation, the environment state are defined as the chip thermal state which is a temperature vector,

$$s = \left[s^1, s^2, s^3, ..., s^m\right]. \tag{5.3}$$

Each temperature value in the vector is a temperature reading from one of m different on-chip thermal sensors deployed in different locations on the chip.

5.2.3 Task Allocation Actions

In a many-core, the task allocator is implemented on a dedicated core which typically performs other system-level management functions. This core dispatches the incoming tasks to other working nodes. So a task allocation action is to make a decision which processor core should be receiving the workload and assign the task the selected. If processor nodes are indexed from 1 to n, the possible actions are given by the following set.

$$\mathcal{A} = \{1, 2, 3, ..., n-1, n\} \tag{5.4}$$

In real implementation, an action is selected from idle cores to achieve high computing performance by utilizing the parallelism of the many-core. We refer legitimate actions to task allocations to idle cores. An assignment to a specific processor is an action denoted as a appeared in (5.2).

5.2.4 Thermal Reward

The construction of the reward function is a key step in effectively performing RL-based task allocation. Since the peak temperature adversely impacts the performance and reliability of the system, reduction of this value is the goal. Generally, an emergency temperature is set as an alarm when the peak temperature crosses the emergency line. System remediation (e.g. frequency and voltage scaling, task migration, and etc.) is needed to avoid severe performance degradation and device damage

if a maximum temperature on the die passes this point. We define the reward of a thermal state as the difference between the emergency temperature and the peak temperature.

$$r = T_{\text{th}} - T_{\max} \tag{5.5}$$

Based on the above equation, the reward r is defined as the margin between the emergency temperature and the peak temperature. The higher the reward, the bigger the temperature margin.

5.2.5 Utility Function Approximation

In our approach, a processor which has a high utility Q value is more likely to to receive a task assignment. If the number of possible temperatures for a core and associated router is relatively small, the utility function $Q(s, a)$ in (5.2) can be represented as a lookup table using temperature vector s and target processor core a as inputs. In other words, for every input temperature vector s, a Q value which has been previously determined and refined for a core a can be identified and used to make the current allocation decision. This approach leads to two issues: (1) Q values must be learned over time and stored in the lookup table and (2) temperature readings can span a large range of continuous values that would have to be discretized. As the state space of temperatures becomes large, using a lookup table for Q learning becomes intractable due to memory limitations and the difficulty of updating it in a timely fashion. Therefore, a continuous function is needed to map state-action (temperature-target processor) pairs to Q values.

$$Q\colon \mathcal{S} \times \mathcal{A} \to \mathcal{Q} \tag{5.6}$$

Due to high complexity of the value function 5.2, it is usually not realistic to find a closed form function for the Q function. However, the value function can be approximated by the linear combination of a series of basis functions, $\phi_i(s)$.

$$Q(s,a) = \sum_{i=0}^{k} \theta_i^a \times \phi_i(s) \tag{5.7a}$$

$$a = 1, 2, ..., n \tag{5.7b}$$

Here, θ_i^a are k weight parameters for core a that are refined after each allocation to the core (k is defined in the next section). Each task assignment to a core (e.g. an action) corresponds to a set of weight parameters θ_i^a for core a. Following the updating of Q values at time $t+1$, weight parameters (θ_i^a) for the processor selected during the previous allocation at time t are updated according to the gradient descent technique [4].

$$\theta_i^a(t+1) = \theta_i^a(t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a))\phi_i(s_t) \tag{5.8}$$

5.2.6 Basis Functions

We need to specify basis function for value function approximation. The basis scheme selected in this paper is radial basis functions (RBF)[4] which is defined as follows.

$$\phi_i(s) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-||c-s||^2/2\sigma^2} \tag{5.9}$$

$$c = \left[c^1, c^2, c^3, ..., c^m\right] \tag{5.10}$$

There are two constant parameters: c and σ^2. Here, c is a m-element vector and σ^2 is a scalar. The elements of $c - s$ provide context regarding the temperature difference between sensor readings s and typical temperature measurements c. Temperature *centers* are specified in this range. The value of each of the elements in c is defined as one of v temperature centers within this range:

$$c^1, c^2, ..., c^m \in \{340, 350\} \tag{5.11}$$

The above example shows $v = 2$ centers. Since each of the values in the c vector can take on any of the v values, there are $k = v^m$ combinations for the c vector. Thus,

v^m basis functions are available to approximate one Q function for each processor a. Since there are n total processors (possible allocation actions), the total number of parameters is $n * v^m$. Note that v is generally quite small (2 or 3).

5.3 Implementation of RL-based Task Allocator

In a many-core system, the task dispatcher is responsible for monitoring thermal state and assigning tasks to cores. At a high level, the steps which take place during each task allocation can be described as follows:

1. Temperature readings are collected from temperature sensors located in each processor core.
2. The maximum temperature T_{max} among all sensors is recorded.
3. The temperature values are used to determine the best assignment of a task to an idle processor core based on a temperature-based *utility function*.
4. The model used to formulate the utility function is updated.

As formulated previously based on reinforcement learning, the utility function effectively determines which assignment is likely to affect the maximum temperature of the chip the least. This effect is determined by considering the processing core's instantaneous temperature and the temperature of the attached router and surrounding cores.

5.3.1 Task Allocation Algorithm Description

Algorithm 3 describes the task allocation procedure performed by the dispatcher. Initially, the θ_i^a weight parameters of the Q function (5.7a) are initialized to zero to define the initial thermal state of the chip. Steps 5-10 are performed for each task allocation. An allocation can be invoked when a new task arrives or an overheating

Algorithm 3 Task Allocator Algorithm

1: Initialize weight parameters $\theta_i^a \leftarrow 0$;
2: Read temperature values s from temperature sensors;
3: Apply a random task allocation;
4: **for** each task allocation episode **do**
5: Get current temperature values s_{t+1};
6: Calculate reward function for the last action based on (5.5);
7: For state s_{t+1}, calculate utility value, $Q(s_{t+1}, a)$, for all processors a;
8: Find maximum Q value from the above step and update the selected processor for the task: $a \leftarrow \text{argmax}_{a'} Q(s_t, a')$;
9: Update weight parameter for θ_i^a according to (5.8);
10: Apply action a with probability p and the rest of all legitimate actions with probability $1 - p$;
11: **end for**

situation is detected and a task must be migrated. The reward is calculated for the last allocation action at Step 6 and current thermal states are obtained by collecting temperatures from thermal sensors at Step 5. Step 7 and 8 determine the task assignment to an idle core (action a) which leads to the maximum Q. Information is updated once the appropriate task allocation action is determined. Our technique is stochastic, i.e. the determined action is taken with probability p, and all other assignments for a specific allocation are applied with combined probability of $1 - p$. Using this approach, potential good actions are not excluded and the environment is extensively explored.

5.3.2 Memory and Computational Complexity

The main memory cost of the allocation algorithm is the storage of weight parameters. The total number of θ_i^a is v^m, which is exponential. However, m (the number of thermal sensors) can be controlled properly to meet performance and storage requirement. To realistically apply the technique, we can cluster cores in several groups and apply task allocation for each cluster. So the memory cost can be reduced significantly.

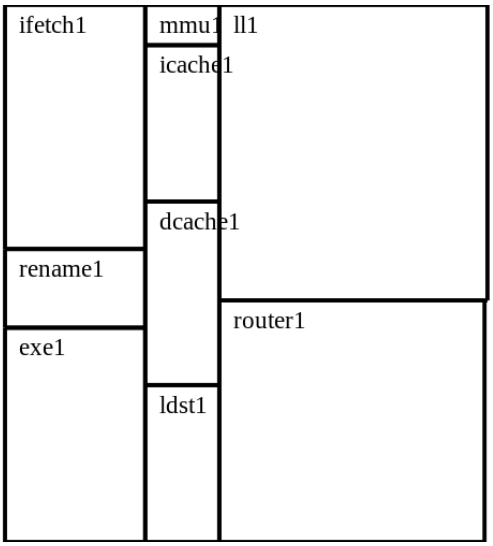


Figure 5.3: Single block floorplan generated by HotFloorplan

The computational complexity is $O(2^m)$ for the weight parameter θ_i^a update. In our experimentation, we use nine sensor readings to represent the thermal state in a 16-core system. The time overhead of the task allocation is less than 0.2 ms, which does not impact performance if allocation is performed once per second.

5.4 Experimental Approach

Simulation is employed to verify the proposed task allocation scheme. Power, temperature and performance were simulated to verify the effectiveness of our new task allocation scheme and to perform comparisons to *adaptive random* task allocation. In this section, a detailed description of the simulation platform, task models and the simulator flow are presented.

5.4.1 Many-core Floorplan and Sensor Allocation

We use a mesh topology to build many-core systems for verification purposes. Both routers and processor cores in 45 nm technology were evaluated. McPAT[52] was used to estimate the area and power for each architectural component in the processor based on the parameters in Table 5.1, and DSENT [95] was used to estimate the area

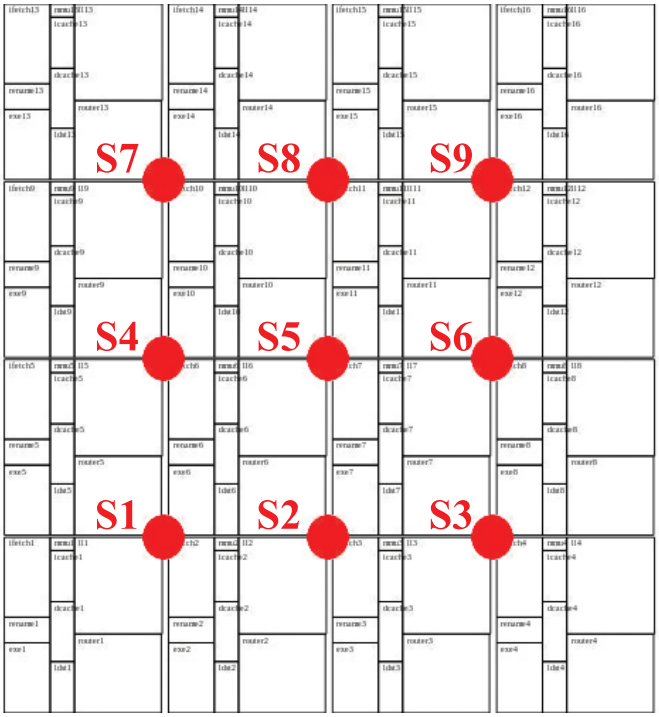


Figure 5.4: Deployment of 9 thermal sensors in a 16-core device

and power for the router based on the configuration in Table 5.2. HotFloorPlan was fed with processor core and router area information to generate the floorplan for a single core. Then, the many-core floorplan was obtained by replicating single-core building blocks. Fig. 5.3 shows the HotFloorplan generated floorplan for a single core-router block.

We assume that multiple thermal sensors exist in a block (including a core and a router) and they are capable of identifying thermal hot spots. Thermal sensors used by (5.3) to record thermal states are spread over the chip. Fig. 5.4 shows that 9 thermal sensors are evenly distributed in a 16-core to characterize the thermal profile. Similar deployments are used for results generation.

5.4.2 Synthetic Workload

Twelve benchmarks from the SPLASH-2 suite are used to test our platform. Each allocated task consisted of an instantiation of one of the benchmarks. Communi-

Table 5.1: Core Configuration

L1-I	16KB
L1-D	16KB
L2	256KB
ITLB	16 entries
DTLB	16 entries

Table 5.2: Router Configuration

message class	3
port number	5
frequency	2.0 Ghz
VC per port	8
flit size	144 bits
buffer length	24 flits

cation between tasks was randomly assigned. To determine dynamic temperature values during many-core execution, power values for all processor core components and associated routers were determined. The power traces of the SPLASH-2 benchmarks were captured using the McPAT-integrated Sniper simulator [10]. HotSpot [2] was used to convert calculated power values and the floorplan of components into temperature values. Note that HotSpot considers the impact of power consumption in neighboring cores and routers in addition to the local core in determining local core power.

We use an $M/M/c$ queuing model to mimic the task arrival and task execution duration in the many-core system. In this model, task arrival is modeled as a Poisson process whose inter-arrival time is exponentially distributed; the execution time of tasks is also exponentially distributed. Value n is the number of cores in the system. The task arrival rate is defined as λ and the service rate is defined as μ. The system utilization, ρ, is given by the following equation.

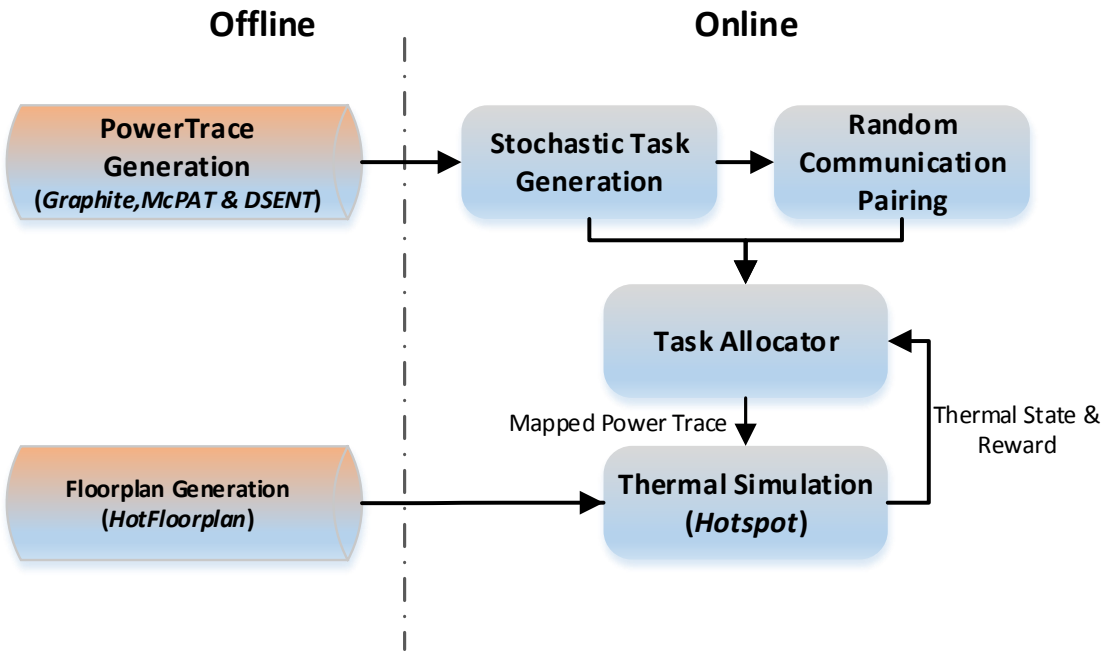


Figure 5.5: The simulation flowchart

$$\rho = \frac{\lambda}{n\mu} \tag{5.12}$$

Effectively, $n\rho$ defines the steady-state number of processor cores which are assigned workload.

5.4.3 Simulation Flow

The simulation flow is shown in Fig. 5.5. As a first step, the power traces for each benchmark for a single-core floorplan are generated. The power of a router under different loads is also calculated. The task allocator is implemented in conjunction with HotSpot which reports simulated chip temperatures. The task allocator retrieves temperature points which represent the s vector in (5.3). When a new task is generated for allocation, its communication is paired with other tasks. The task allocator assigns the appropriate power trace from the stochastic task generator and maps it onto the floorplan. The HotSpot simulator reads the mapped power trace

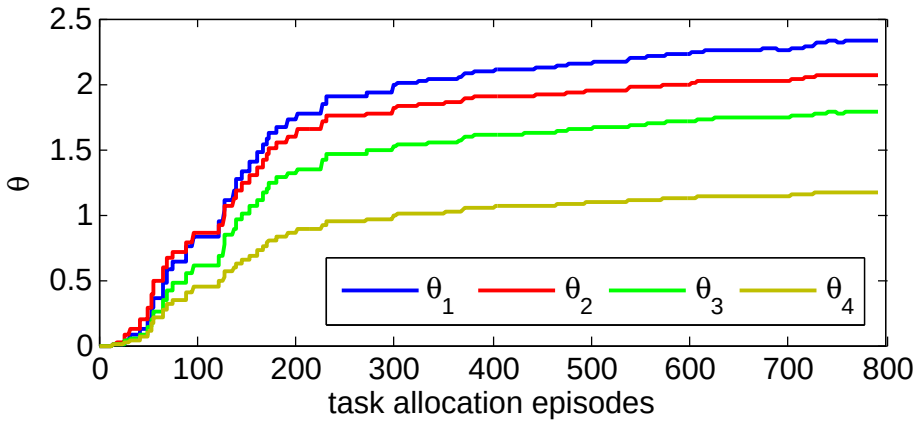


Figure 5.6: Convergence of θ_i values for a 25-core processor

from the task allocator and performs thermal simulation. The maximum temperature is reported to the allocator for reward calculation.

5.5 Results

Our approach has been validated via simulation using 16-core, 25-core, 36-core and 49-core systems. To implement the reinforcement learning technique, the learning rate is set to $\alpha = 0.8$ and the discount rate is set to $\gamma = 0.8$. These parameters were determined empirically.

5.5.1 Effectiveness Validation

In an initial experiment, the convergence of our reinforcement learning model is evaluated over a series of task allocations. In the experiment, allocations are performed to all 25 processor cores. A selection of θ_i values is shown in Fig. 5.6. Initially, θ_i values are all zeros and they begin to converge after $200 \sim 300$ allocation episodes. Other θ_i values showed similar behavior.

The Q values for a fixed thermal profile are also evaluated as task allocation with reinforcement learning proceeds. The thermal profile is represented by a sextuple collected from thermal sensors ($s_0 = [343, 347, 340, 339, 342, 339]$). We evaluate Q

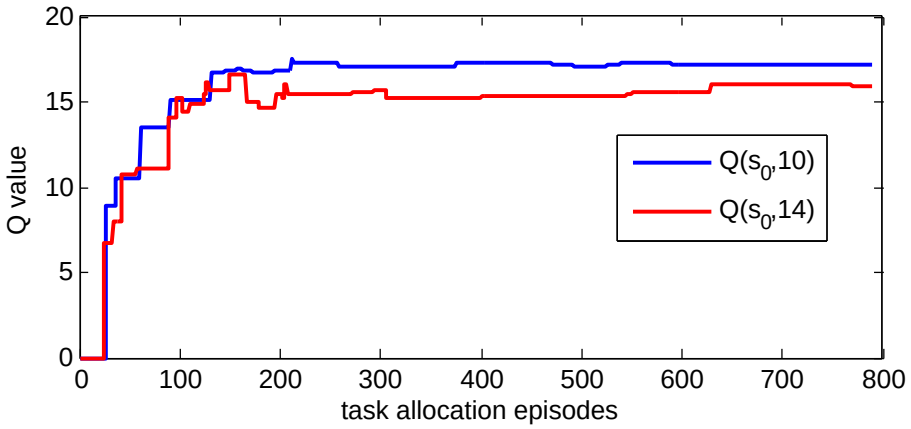


Figure 5.7: Convergence of Q values for a fixed thermal condition

values for two actions (allocation to processors 10 and 14, respectively) under thermal profile, s_0, for individual allocation episodes over a total of 800 task allocations. Fig. 5.7 shows Q value evolvement over time as tasks are allocated and θ_i weight parameters for function approximation are updated. Initially, Q values are all zeros and they begin to converge after $200 \sim 300$ allocation episodes.

Fig. 5.8 shows a thermal snapshot of a 36-core processor at the seven minute time point. Q values are calculated for all possible allocation choices at this time point. Fig. 5.9 shows the magnitude of Q values for corresponding cores indexed in Fig. 5.8. As seen in the figure, the Q value for action 22 is lowest among all actions because the heat stress for core 22 is significant. An allocation to core 22 will negatively impact the chip peak temperature. Actions 3, 18, 29 and 35 have relatively high Q values. From the chip thermal map, it is seen that core 3, 18, 29 and 35 are relatively cool and their neighboring cores are also in a favorable thermal condition. Effectively, the allocator has learned how to respond this thermal environment. We also notice that core 1 and 31 are cool but the Q values are not as high as the previous four cores. These two cores are in the corner of the chip and the thermal conductivity of air is much lower than silicon. The allocator effectively learns this information over time via reinforcement.

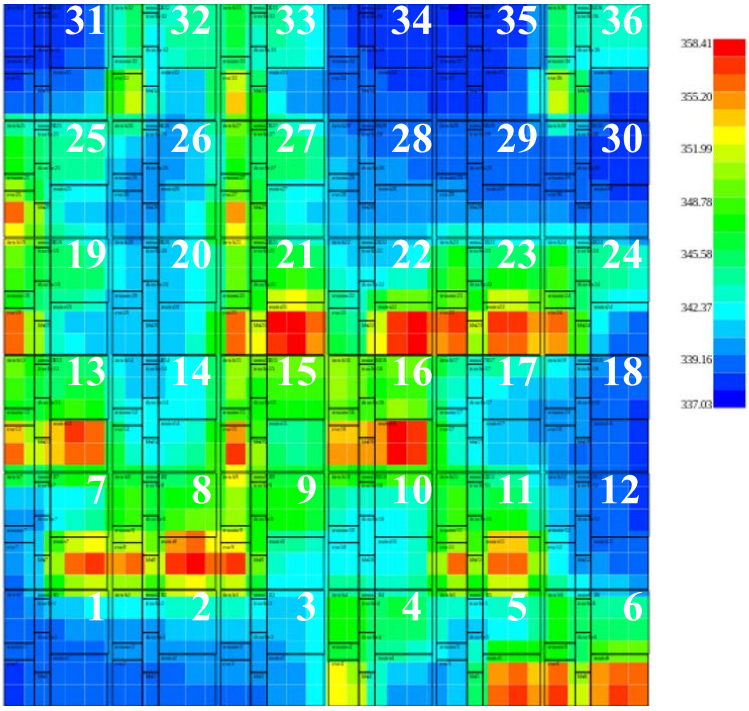


Figure 5.8: A thermal map snapshot for a 36-core

5.5.2 Peak Temperature Reduction

The peak temperature can be effectively reduced versus previous approaches through the use of the proposed allocation technique. A series of experiments are conducted to observe the peak temperature of the chip in comparison with the *adaptive random* approach [24] over five minute execution runs. This allocator assigns tasks to one of the coolest available cores in a multi-core based on probabilities determined from core temperature histories. In our implementation of the adaptive random approach we also included the impact of router power consumption on task allocations to allow for a fair comparison versus the reinforcement learning based technique.

Table 5.3 shows the average peak temperature over time for different core counts and system utilization ratios for the two approaches. For a low system utilization ($\rho = 0.1$), adaptive random and reinforcement learning have almost the same performance in terms of peak temperature. The main reason is that the chip temperature is

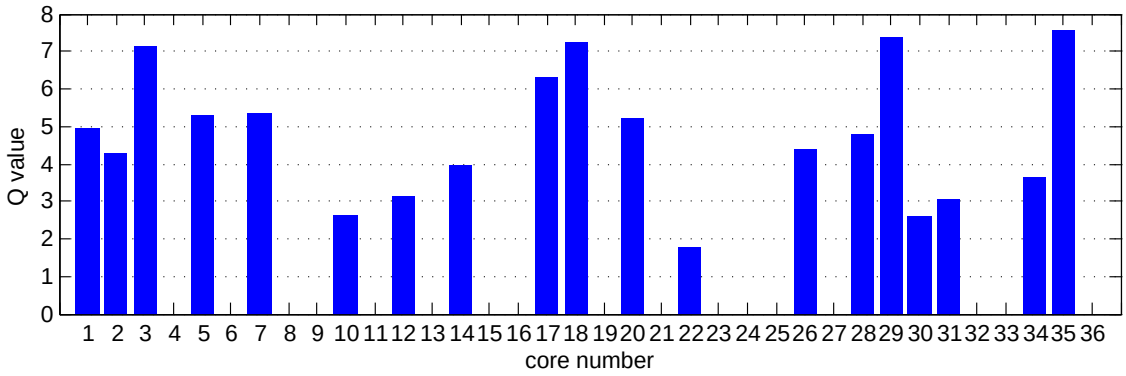


Figure 5.9: Q values for different actions at the thermal state given in Fig. 5.8.

less likely impacted by the distribution of tasks and the hot spot is mainly induced by the workload intensity inside the core. Our technique performs better when the system is moderately loaded (about half used). For example, compared to adaptive random, our approach reduces peak temperature by 9.4% in a 49-core system (6.2% across all many-core configurations). When the system is heavily utilized ($\rho = 0.8$), the performance of two techniques is almost the same since feasible choices for task assignment are limited. Proactive circuit level DTM can be used in this case to avoid overheating.

Fig. 5.10 shows the differences in the peak temperatures over time between the two approaches for 16-, 25-, and 36-core systems. Comparisons between reinforcement learning and adaptive random indicate that the former approach is more effective in reducing the peak temperature.

The importance of including router temperature in many-core task scheduling is apparent from Fig. 5.11. All other reported results in this chapter consider the impact of router temperature.

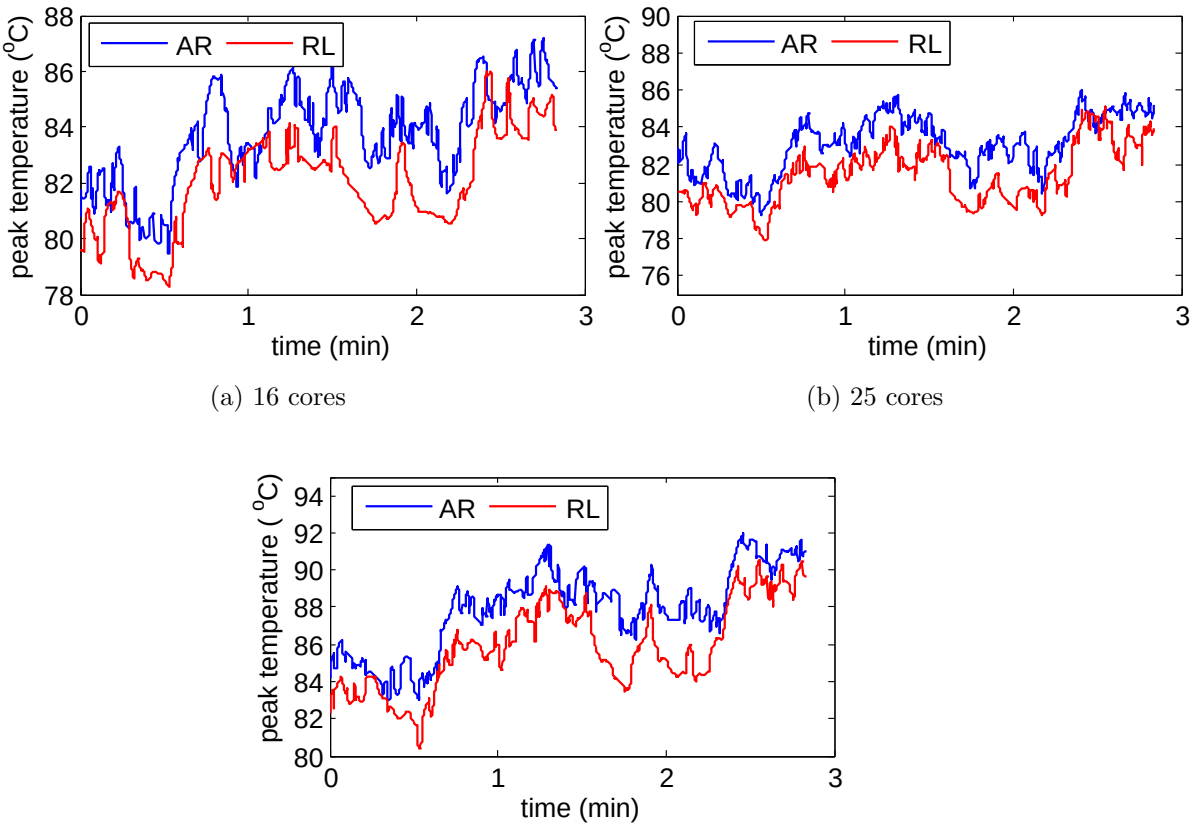
5.5.3 Memory and Computational Complexity

The time cost of the reinforcement learning allocator was evaluated for different numbers of sensors, m, and two sets of temperature centers, v. Table 5.4 shows the

Table 5.3: Peak temperature comparison between reinforcement learning (ours) and adaptive random [24]. The value ρ indicates the average number of cores used during execution. Percentage peak temperature reductions are shown.

	System Utilization					
	$\rho = 0.1$			$\rho = 0.4$		
core number	ours (^oC)	[24] (^oC)	%	ours (^oC)	[24] (^oC)	%
16	73.2	74.1	1.2%	81.5	86.4	5.9%
25	74.6	74.3	-0.4%	83.1	86.7	4.2%
36	72.8	73.2	0.5%	83.8	88.3	5.1%
49	70.5	71.4	1.3%	78.0	86.1	9.4%
	System Utilization					
	$\rho = 0.6$			$\rho = 0.8$		
core number	ours (^oC)	[24] (^oC)	%	ours (^oC)	[24] (^oC)	%
16	86.8	88.5	1.9%	93.8	93.5	-0.3%
25	87.5	90.8	3.6%	95.1	96.0	0.9%
36	87.3	91.1	4.2%	96.4	96.1	0.3%
49	86.2	90.8	5.1%	93.1	93.8	0.7%

average computational time for one task allocation. Typically, temperatures collected from less than 10 sensors are representative of the chip thermal profile although more sensors are needed to recover whole chip thermal map. The temperature centers used in (5.10) are selected from two sets, $\{340, 350\}$ and $\{335, 345, 355\}$, respectively. For most cases, the computational time of task allocation is $< 1ms$. Since the task allocation is only invoked when there is an incoming task or a thermal emergency, the frequency of allocation is typically on the order of seconds. Therefore, the percentage time cost of allocation with respect to the allocation interval is negligible. Although the adaptive random technique also has very low overhead time cost ($<< 1\ ms$) for



(a) 16 cores

(b) 25 cores

(c) 36 cores

Figure 5.10: Peak temperature comparison over time for $\rho = 0.4$ for reinforcement learning (RL) and adaptive random (AR)

each task allocation, it must track temperature sensor readings over time (one sample per 100 ms) regardless of task allocation rate. As a result, the technique becomes less favorable in a system which has a low task arrival rate.

The primary memory cost of reinforcement learning is the storage of θ_i weight parameters. In Table 5.5, the memory overheads are listed for typical processor core configurations and most of them are less than one Mbyte. For contemporary servers with gigabyte memories, this is a very small fraction of total memory. The adaptive random approach requires a similar overhead (100s KByte) to track thermal history for thermal index adjustment.

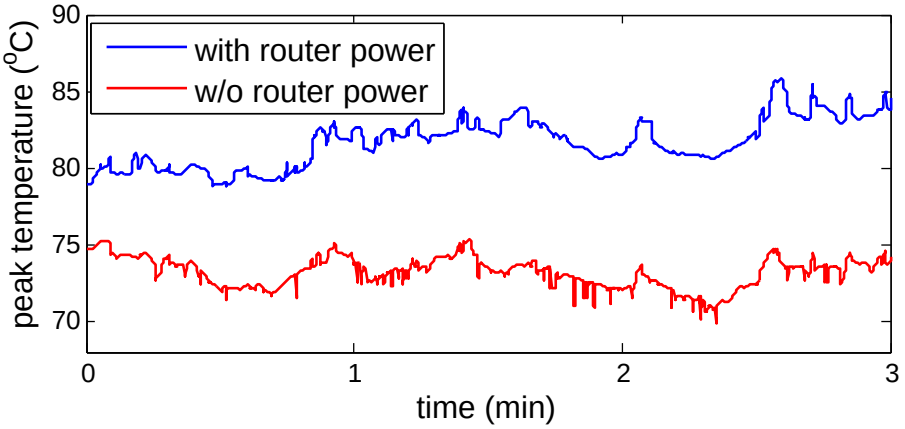


Figure 5.11: Estimation results for RL if router temperature is considered or omitted

Table 5.4: Time Cost For RL Task Allocation (ms)

Temperature Center Set	Sensor Numbers 5	6	7	8	9
{340, 350}	0.014	0.024	0.043	0.081	0.150
{335, 345, 355}	0.074	0.216	0.635	1.927	6.210

Table 5.5: Memory Cost For a 16-Core System (KB)

Temperature Center Set	Temperature Sensor Count m 5	6	7	8	9
{340, 350}	2	4	8	16	32
{335, 345, 355}	15.6	46.7	140.0	420	1,259.7

5.6 Summary

In this chapter, a reinforcement learning based task allocation strategy is proposed to address localized overheating in many-core systems due to both processor core and router power consumption. Function approximation is employed to evaluate quality metrics (Q values) to find optimized allocation decisions. Our algorithm is verified via detailed many-core simulation which includes on-chip routing. The experiments show that the proposed technique is capable of capturing the complex on-chip thermal environment induced by dynamic work load distribution. Our results show that the proposed technique is fast (scheduling performed in $<$1 ms) and can efficiently reduce peak temperature by 6% on average in moderately-loaded many-core processors for a collection of SPLASH-2 benchmarks.

CHAPTER 6

THERMAL AWARE TASK ALLOCATION FOR 3D MANY-CORES

Three dimensional integrated circuits (3D-ICs) stack multiple silicon dies vertically and use through silicon vias (TSVs) for inter-die communication. Although 3D technology can effectively reduce circuit footprint and increase system performance, the thermal behavior of 3D many-core processors has grown to become a major concern in terms of both reliability and performance. High temperatures resulting from the thermal proximity of stacked silicon dies in 3D-ICs and longer heat dissipation paths impact circuit reliability and chip lifetime. To address the issue, hardware based remediation techniques such as dynamic voltage and frequency scaling are initiated during thermal emergencies when a core temperature value passes a predefined temperature line. These events inevitably cause speed degradation which offset the performance benefits of 3D technology.

To alleviate heat issues, advanced cooling technologies have been proposed to accelerate heat removal from stacked silicon, including liquid cooling using microchannels [85] and superlattice-based thermoelectric coolers [19]. In 3D systems, thermal stress varies considerably across silicon layers. Spatial thermal correlation due to horizontal and vertical heat transfer plays an important role [87] in shaping the chip thermal profile. It is challenging to dynamically allocate workloads based on the chip thermal profile due to this correlation. In a contemporary 3D many-core system, various components, such as processors, memories, and on-chip routers, among others, are included. The increased power consumption in the communication infrastructure

coupled with a multitude of stacked processing components makes the thermal profile difficult to track using traditional diffusion simulation technology.

In this chapter, the reinforcement learning technique introduced in the previous chapter is improved and applied to a 3D many-core system. As described in the previous chapter, a reinforcement learning agent adaptively learns the thermal stress of each core and makes an allocation decision based on the current thermal profile obtained from thermal sensor readings. Since algorithm scalabilty is a major concern for 3D systems, a cluster based algorithm is presented to accelerate RL-based allocation.

6.1 Thermal Behavior in 3D Integrated Circuits

The physical structure of a 3D integrated circuit provides insights into its thermal behavior. Figure 6.1 shows a typical implementation of a 3D integrated circuit. In this example, three silicon layers are stacked on top of each other. The thickness of the silicon layers in 3D-ICs is much thinner than in 2D-ICs. It is around $15\mu m$ -$60\mu m$ compared with $600\mu m$-$900\mu m$ for 2D-ICs [110]. However, the silicon substrate of a 3D-IC (the layer closest to heat sink - layer 2 in Figure 6.1) is thicker than interior layers and comparable to the thickness of a 2D IC. Since a heat sink is more capable of conducting heat flux than a printed circuit board, the layer closest to the heat sink is often cooler than the interior layers. TSVs have better thermal conductance than bonding material, so a larger TSV density results in higher thermal conductance and stronger inter-layer thermal interference (also called vertical thermal correlation). Thermal interference on the same layer is horizontal thermal correlation. In 3D chips, vertical thermal correlation is stronger than corresponding horizontal values due to the larger contact area between two layers versus the contact area between two adjacent cores on the same layer.

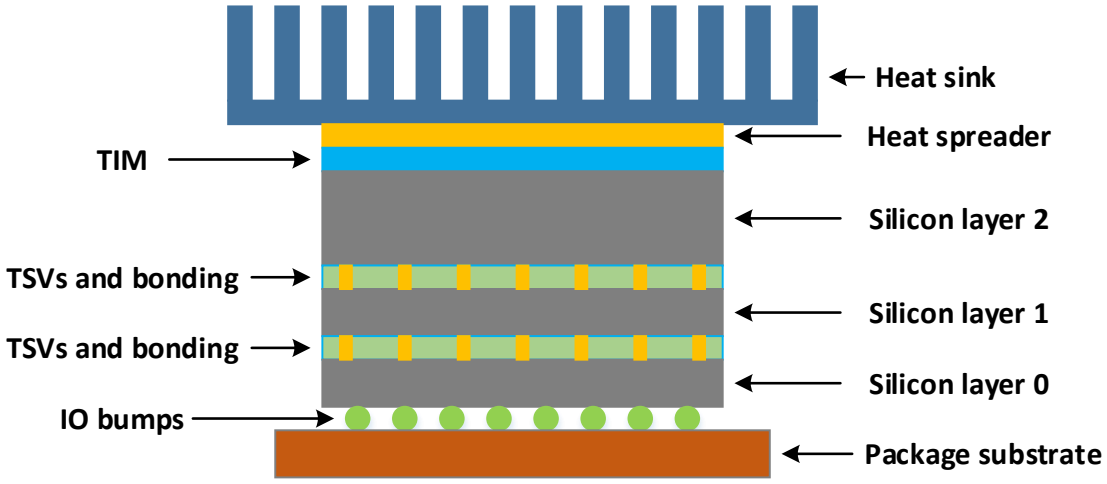


Figure 6.1: Typical 3D layout of an integrated circuit

A simple example is used to illustrate this observation. In a three layer many-core system, two tasks are placed on two neighboring cores. In Figure 6.2, the tasks are assigned to cores 14 and 15 on layer 2. In Figure 6.3, the same two tasks are assigned to core 14 on layer 1 and core 14 on layer 2. The resulting thermal maps for these two assignment schemes are shown in Figures 6.4 and 6.5. The second assignment results in more thermal stress than the first. The peak temperature of the first and second thermal maps are 66.34^oC and 69.39^oC, respectively.

Figure 6.6 plots peak temperatures for various assignment schemes for thermal steady states. An assignment is represented by a tuple with four numbers. For example, (2, 14, 1, 15) represents that one task is assigned to layer 2 at core 14 and the other task is assigned to layer 1 at core 15. The peak temperatures vary for different task assignment schemes, as observed in the figure. The peak temperature difference is over 6^oC for assignment $(0, 14, 1, 14)$ versus assignment $(2, 14, 2, 22)$. From these simple examples, we can obtain the following insights.

- Insight 1: Tasks should be assigned to the layer closest to the heat sink since the heat flux can be more easily conducted.

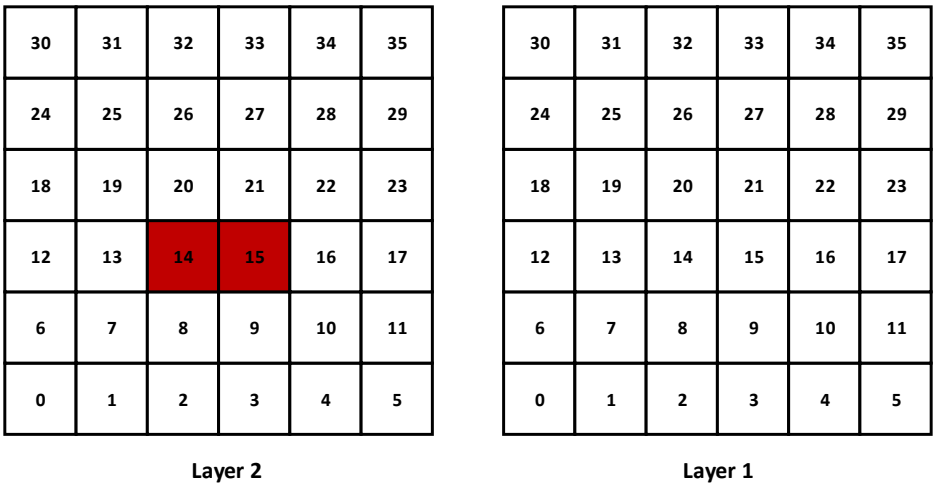


Figure 6.2: Two tasks are allocated to neighboring cores on the same layer

- Insight 2: Back-to-back assignment (two neighboring cores on different layers, as shown in Figure 6.3) produces more thermal stress than side-by-side assignment (two neighboring cores on the same layer, as shown in Figure 6.2). Therefore, back-to-back assignment should be avoided if possible.

- Insight 3: The cores at the corners are more easily heated leading to higher peak temperature, so corner assignment should be given low priority.

6.2 Revisiting Reinforcement Learning

As mentioned in the previous chapter, a reinforcement learning task allocator uses chip temperature information to dispatch tasks in a thermal-aware fashion. Essentially, reinforcement learning is a trial-and-error process between an agent and a dynamic environment. For thermal-aware task allocation, the agent is a task allocator and the environment is the thermal condition of the silicon die. They interact with each other by repeatedly performing the following two steps.

1. The allocator reads the temperature profile of the die and calculates quality value Q (defined as expected accumulated rewards) for each possible allocation.

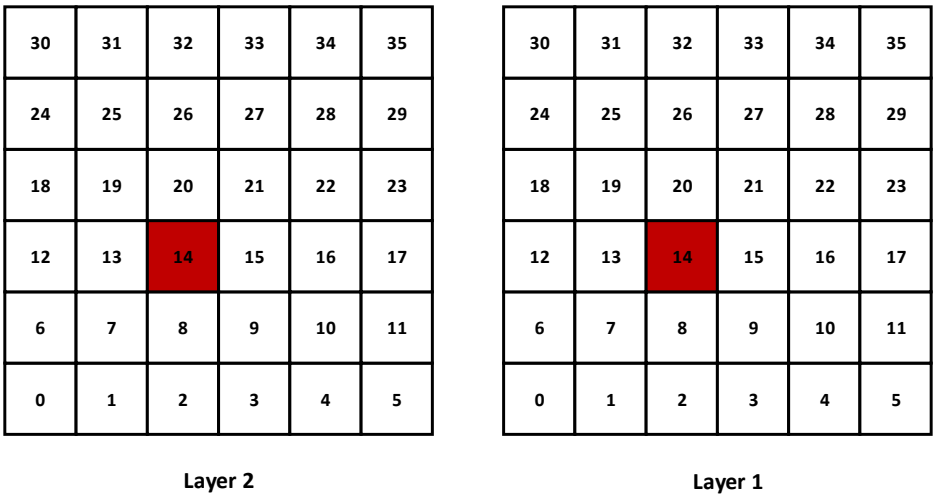


Figure 6.3: Two tasks are allocated to neighboring cores on two layers

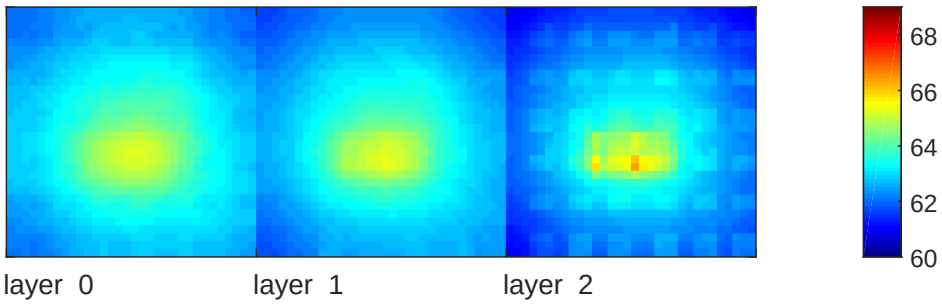


Figure 6.4: Thermal maps resulting from Fig. 6.2 assignment

The processor core which has the highest quality value is selected to receive the task.

2. For each task assignment, the resulting thermal profile is retrieved by reading post-assignment temperature values from on-chip thermal sensors. The previous allocation is evaluated in terms of thermal reward based on an optimization goal. The result of the evaluation is used to update weights for future quality value calculation.

More processor cores are typically included in a 3D system versus a 2D system, so global optimization must generally consider temperatures from more thermal sensors

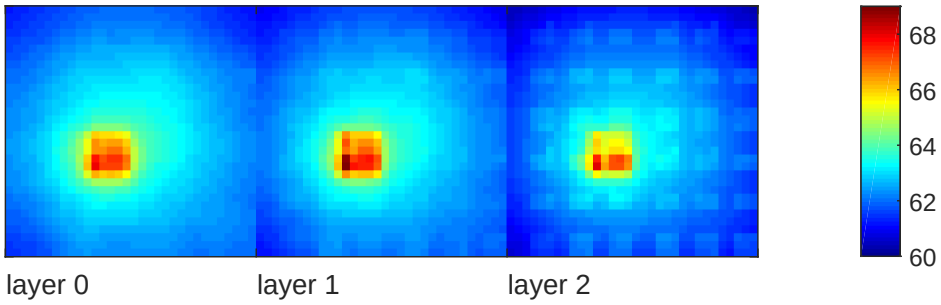


Figure 6.5: Thermal maps resulting from Fig. 6.3 assignment

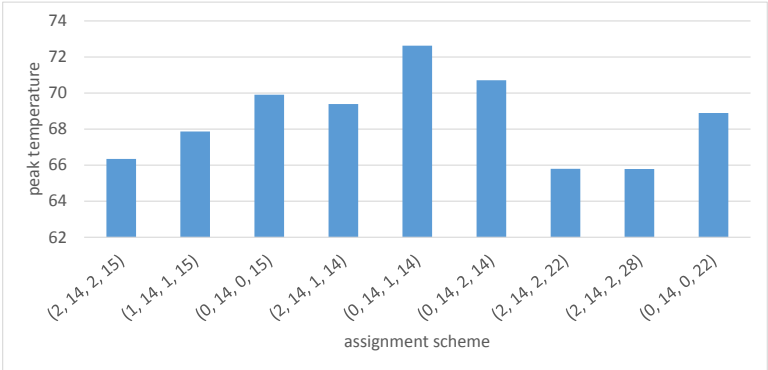


Figure 6.6: Peak temperatures for assignment schemes for thermally-steady states.

in generating a detailed thermal profile. Thus, the computational overhead to run reinforcement learning becomes high as the number of basis functions grows exponentially with the number of sensors. Communication latency can also limit the overall system performance because thermal aware techniques intrinsically place tasks apart to reduce thermal interference. A hierarchical approach to task allocation is proposed in the next section to reduce computational overhead and communication latency.

6.2.1 3D Many-Core Model

The 3D many-core model used in this dissertation is similar to those used in previous work [121] [110]. We assume that there are three computing layers with the same core count in each layer. Figure 6.7 shows a three layer many-core system with

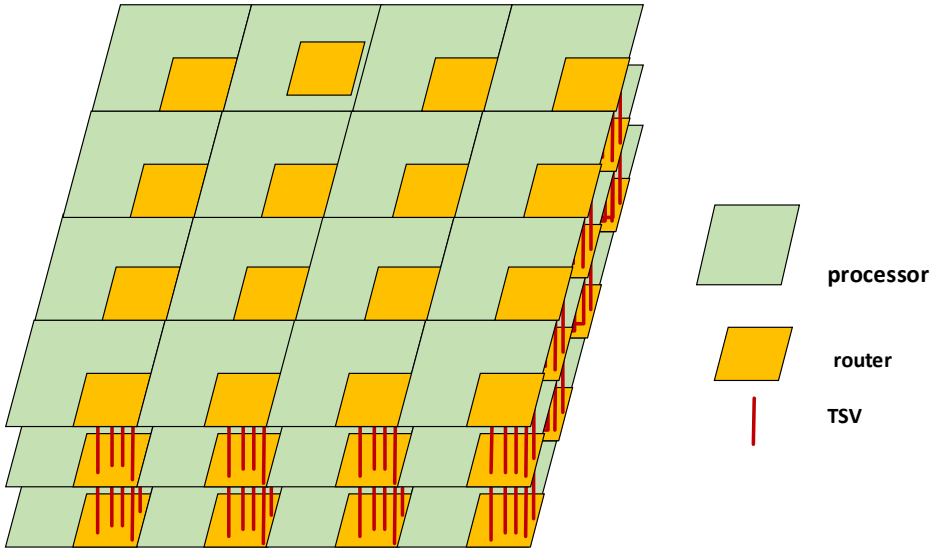


Figure 6.7: A three layer many-core system with 16 cores on each layer

16 cores on each layer. The processor nodes are connected in a 3D mesh topology. We assume that TSVs only pass between layers in the router region.

6.2.2 A New Definition of Thermal Reward

The construction of the reward function is a key step in effectively performing RL-based task allocation. Since the *peak* temperature adversely impacts the performance and reliability of a multi- or many-core, the reduction of this value is the goal. Instead of using the temperature margin between the peak temperature and thermal emergency line as in Chapter 5, a new reward function is defined when applying reinforcement learning in a 3D many-core system.

We determine the reward of a task assignment based on the peak temperature change before allocation and after allocation, as shown in Equation 5.5. T_{pre} is the peak temperature before an allocation takes place and T_{post} is the temperature after the allocation, as shown in Figure 6.8. It is important to note that T_{pre} is the peak temperature right before the allocation. Since temperature takes some time to reach a steady state (usually hundreds of milliseconds), T_{post} is the steady state temperature

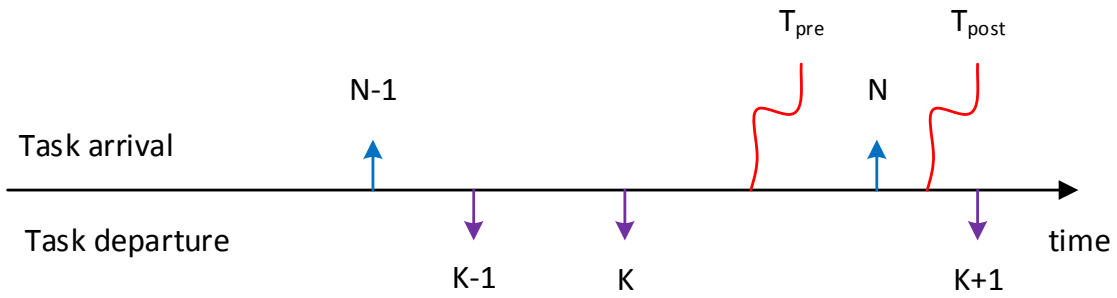


Figure 6.8: T_{post} and T_{pre} peak temperature illustration in the task arrival and departure time line

value after the stabilization period. This value reflects the steady state temperature impact of an allocation. T_{post} is always greater than T_{pre} because the allocation of an additional task in this manner increases temperature. A constant value C is used to guarantee that the reward is a positive value. From our experiments, the typical peak temperature change is around a couple of degrees, so squaring is used to increase mathematical sensitivity.

$$r = C - (T_{post} - T_{pre})^2 \tag{6.1}$$

Based on the above equation, the reward r of an allocation is an indication of how much temperature increase occurred for that allocation. The higher the reward, the less the temperature increase.

6.3 Cluster-based Reinforcement Learning for 3D ICs

As processor core count increases, especially for 3D systems which have multiple computing layers, an increased number of thermal sensors are necessary to represent the chip thermal state at a sufficient level of detail. Since the number of basis functions (defined in Section 5.2.6) increases exponentially with the number of thermal sensors (2^m for m sensors with two temperature centers), computational complexity can become unreasonable for on-line task allocation if m is large. To increase the scal-

ability of the allocator for 3D many cores, we introduce a distributed RL allocation approach which replaces the centralized RL agent for the system with a distributed collection of agents. One agent is used per cluster of cores.

6.3.1 Reinforcement Learning on Clusters

Heat transfer results in spatial correlation between cores, although this correlation decreases as the physical distance between cores increases. Fig. 6.9 shows the thermal correlation coefficients between a selection of cores, determined via simulation. The x axis represents distance, where one λ represents the width of a single core block. According to the experiment, the correlation decreases to < 0.2 for cores which are 3 blocks away. Therefore, it can be said that the Q value for a core is less related to temperatures at remote locations.

A cluster based method is now described that performs distributed task allocation. Processing nodes are grouped into clusters based on their physical closeness and each cluster has its own RL allocator which incurs limited computational overhead. Fig. 6.10 shows clustering for several many-core systems (25-core, 36-core, 49-core, 64-core)[1]. Consider the 25-core processor as an example. There are 4 clusters in this system and each cluster is a 4×4 processor array. Table 6.1 lists cores included in each cluster (it is allowable to have overlaps between clusters). Similar clustering techniques are performed for the other many-cores in Fig. 6.10. In a 3D chip, the clustering is performed on each computing layer separately.

Task allocation is performed hierarchically. A cluster is first selected for allocation and then RL allocation is performed among the cores in the cluster. Algorithm 4 shows the allocation procedure on a clustered many-core system. Initially, the reinforcement learning allocator for each cluster is initialized. Upon a new task arrival, one of clusters is selected for the task based on cluster average temperature. The

[1]2D layouts are shown for clarity but the same concept applies for 3D layouts

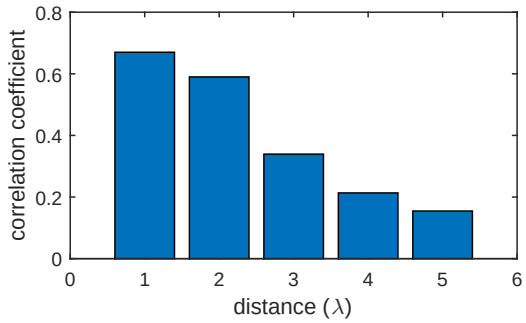


Figure 6.9: Spatial thermal correlation

Algorithm 4 Clustered RL Algorithm

1: Initialize weight parameters $\theta_i^a \leftarrow 0$ for each RL cluster;
2: Read temperature values s from temperature sensors;
3: Calculate average temperatures for all clusters, $\{C_i\}$
4: Find the cluster which has the lowest average temperature, C_{min}
5: Apply reinforcement learning on Cluster C_{min} according to Algorithm 3 and find the core to which the task will be assigned.
6: Apply task allocation according to the previous result;
7: Update weight parameters for the selected cluster based on the resulting thermal reward.

average temperature is calculated from thermal sensor readings in each cluster and the *coolest* cluster is picked for task assignment.

6.3.2 Communication Analysis

The goals for inter-task communication and thermal optimization are not consistent. To limit communication, it is more favorable to place two tasks close together to reduce latency and overall network traffic. However, from a thermal standpoint,

Table 6.1: Four clusters in a 5x5 system

Cluster 0	0, 1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 13, 15, 16, 17, 18
Cluster 1	1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 16, 17, 18, 19
Cluster 2	5, 6, 7, 8, 10, 11, 12, 13, 15, 16,17, 18, 20, 21, 22, 23
Cluster 3	6, 7, 8, 9, 11, 12, 13, 14, 16,17, 18, 19, 21, 22, 23, 24

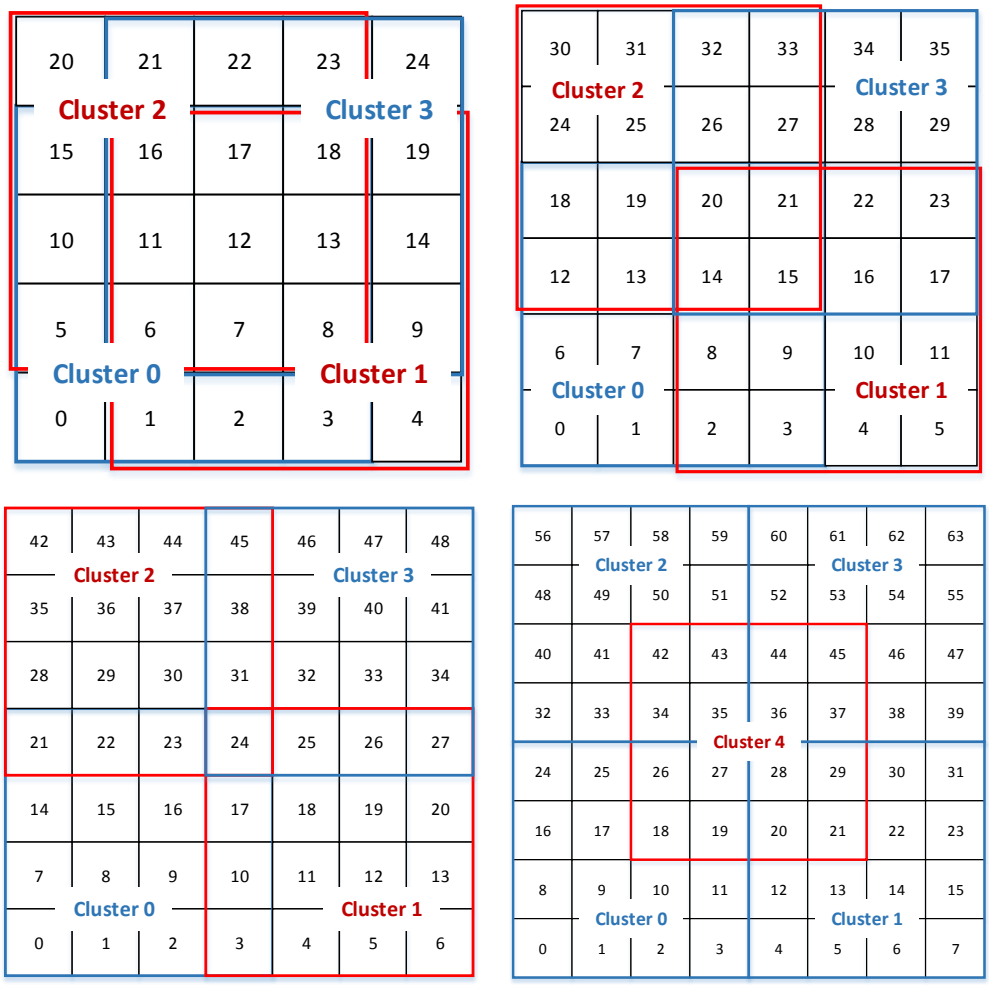


Figure 6.10: Clustering schemes for 25, 36, 49 and 64-core systems

it is more desirable to place tasks apart to limit cross-core thermal interference and achieve a cooler thermal profile. Our cluster based technique can be modified to improve the efficiency of inter-task communication. Instead of picking the *coolest* cluster, the following two policies are employed to determine the target cluster.

1. If the task does not communicate with other nodes, pick the coolest cluster among all clusters for assignment.

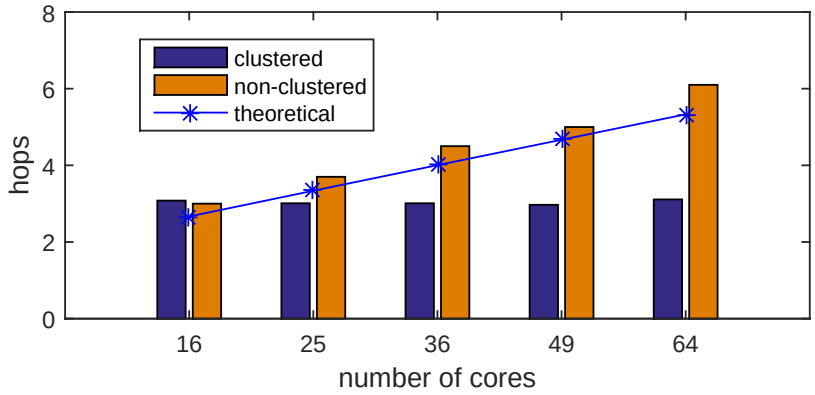


Figure 6.11: Communication latency comparison. *Theoretical* indicates random assignment of tasks to cores.

2. If the task communicates with other nodes, assess clusters which contain cores with which the task communicates and then pick the coolest cluster among these clusters.

Policy 1 is the original cluster determination policy which picks the globally coolest cluster. Policy 2 guarantees that the target task will be assigned to a subregion which incurs limited communication overhead. The tradeoff between communication and temperature is controlled by the cluster size. As shown in Figure 6.10, a 4×4 cluster size is used for our implementation. An evaluation of the average latency is presented in Figure 6.11 where we use hops as the metric for the latency. From this figure, the average latency remains constant for the clustered approach, but it increases linearly for the non-clustered method. We also plot a line for an allocator which randomly assigns tasks. The RL-enabled non-clustered allocators give larger average communication latency because the RL allocator tends to place tasks apart from each other.

6.4 Experimental Setup

To verify the effectiveness of the new task allocation scheme, power and temperature simulations were performed on 3D many-core models. The simulation tools and verification flow described and used in Chapter 5 were used to perform assessment in this chapter.

6.4.1 3D Floorplan and Thermal Simulation

Both routers and processor cores in a 3D mesh topology were evaluated in 45 nm technology. Many-core floorplans were obtained by replicating the single-core building blocks shown in Figure 5.3. In our experiments, a 3D chip is composed of 3 homogeneous computing layers with TSVs between layers, as shown in Figure 6.7. We assume that there are only connections between on-chip routers, so TSVs only exist in router regions and the TSV density is set to 10% considering IR drop noise [46]. TSVs usually have better thermal conductivity than thermal interface materials (TIM). HotSpot with a 3D extension [69] was used to convert simulated power traces and the floorplan of components into temperature values.

6.4.2 HotSpot Integration

Our allocation scheme was integrated with 3D Hotspot using Python/C APIs. The main reason for using Python was the rich Python libraries for matrix and numerical computation which can be used by the RL scheduler. Code size is reduced significantly with Python compared with C implementation and much less debug effort is required. It is also much easier to adjust parameters and configurations with a Python implementation, so scheduler tuning is much less time consuming.

Figure 6.12 shows a single allocation cycle implemented in 3D HotSpot. The work is divided into two phases: task removal and task assignment. Every time a task arrived, running tasks are checked to see if they have expired. Expired tasks are removed from the current task list and a new power map is generated based on the

updated task map. The heat diffusion solver in HotSpot is called with the updated power trace as inputs. After the solver finishes calculations, temperature values at preselected locations where thermal sensor are deployed are identified. At this point, the pre-allocation peak temperature (T_{pre}) can also be obtained from the HotSpot solver. These steps are shown in the figure from Step 1 to Step 4. This is called the task removal phase since these steps are mainly associated with the deletion of expired tasks in the simulation framework.

The task assignment phase starts with RL allocation, the fifth step in Figure 6.12. Using the thermal sensor readings from the final step in the task removal phase, the scheduler determines which cluster to assign the new task and runs the corresponding reinforcement learning engine using that cluster. The task map is updated by the scheduler to add the new task and the power map is regenerated using the new task map. The heat diffusion solver is called again to perform thermal simulation using the updated power trace. We can extract thermal sensor readings and T_{post} after the temperature reaches a steady state. As a last step, the coefficients of the basis functions are updated using the reward calculated from T_{pre} and T_{post}.

6.4.3 Benchmark Workload

The SPLASH-2 [104] benchmark suite is used to test the proposed task allocation strategy for 3D many-core systems. A benchmark is randomly selected for each incoming task and an $M/M/c$ queuing model is used to determine task arrival times and task execution durations in the many-core system. Task arrival is modeled as a Poisson process whose inter-arrival time is exponentially distributed; the execution time of tasks is also exponentially distributed. n is the number of cores in the system. The task arrival rate is defined as λ and the service rate is defined as μ. The system utilization, ρ, is given by the following equation.

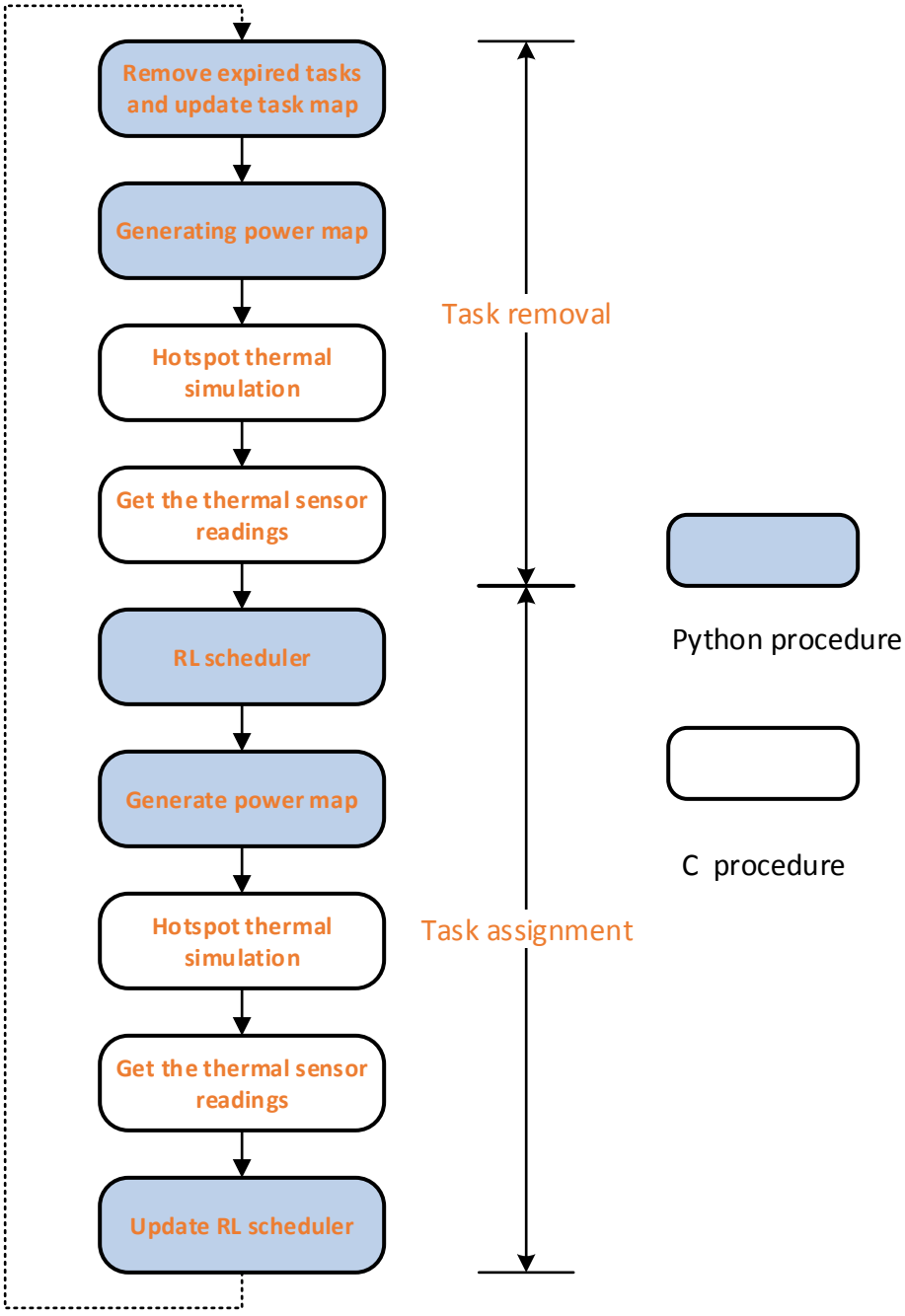


Figure 6.12: A single allocation cycle implemented in HotSpot

Table 6.2: Average number of executing tasks for a collection of system configurations

Config.	$\rho = 0.2$	$\rho = 0.5$	$\rho = 0.7$
16×3	9.6	24	33.6
25×3	15	37.5	52.5
36×3	21.6	54	75.6
49×3	29.4	73.5	102.9
64×3	38.4	96	134.4

$$\rho = \frac{\lambda}{n\mu} \tag{6.2}$$

Effectively, $n\rho$ defines the steady-state number of processor cores which are used to service tasks. In our experiment, the arrival rate is set to 4 tasks per second to accelerate the simulation speed. The service rate is adjusted based on system utilization and core count. Table 6.2 lists the average number of tasks running in the system for various configurations.

6.5 Results

Our approach has been validated via simulation using three layer many-core systems, each layer of which includes 16, 25, 36, 49, or 64 cores. To implement the reinforcement learning technique, the parameters in Table 6.3 were determined empirically.

Table 6.3: Empirically determined parameters in used experiments.

PARAMETER	SELECTED VALUE
learning rate α in Eqn. (5.8)	0.9
discount factor γ in (5.8)	0.8
probability p in Algorithm 3	0.95
σ in (5.9)	17

6.5.1 Peak Temperature Reduction

In this section, we show that the peak temperature can be effectively reduced versus a previous approach [120] through the use of the proposed allocation technique. To quantitatively evaluate peak temperature reduction, a series of experiments are performed for different core count configurations and system utilization rates. Due to the limits of parallelism and thermal design power (TDP), the dark silicon rate is expected to exceed 50% for future technology generations [32]. Therefore, it is reasonable to assume that the system utilization rate is under 70% for our 3D many core systems for experimentation. The balance-by-stack assignment scheme [120] bundled vertically-stacked cores at the same 2D position as a "super-core". Super-cores are sorted by their average temperatures and the coolest stack is selected for assignment. For each super-core, tasks are allocated onto cores in order of decreasing temperature, i.e., the most power consuming task is assigned to the core closest to the heat sink. In the following discussion, we refer to this approach as the balance-by-stack approach.

Figure 6.13 shows the peak temperature reduction with respect to a non-thermal aware task allocator for a variety of configurations. Each plot in the figure shows the peak temperature reduction distribution, where the x axis represents the amount of reduction and the y axis represents counts of corresponding reductions. For each configuration in Figure 6.13, experiments were performed with 5000 incoming tasks. From these figures, one can observe that the RL allocator consistently reduces the peak temperature for $\sim$ 88% of the cases. The average temperature reduction is $1.7 \sim 2.6^oC$ and the maximum peak temperature reduction is $> 10^oC$ (as shown in experiments for "$16 \times 3, \rho = 0.2$", "$16 \times 3, \rho = 0.5$", and "$64 \times 3, \rho = 0.2$"). For most experiments shown in the figure, over 20% of the reduction values are 4^oC or more.

Figure 6.14 shows the peak temperature comparison for random, balance-by-stack [120] and RL approaches. The system utilization is set to 0.5 for all experiments in

producing the results. The two thermal aware approaches reduce the peak temperatures with respect to the non-thermal-aware approach. The RL approach performs slightly better than the balance-by-stack approach. The average peak temperature is 0.47^oC less for the RL approach.

Figures 6.15 and 6.16 show thermal profiles for RL and balance-by-stack allocators after 5000 allocations under the same task sequence. The thermal maps are plotted for all three layers. Layer 2 is the closest to the heat sink and Layer 0 is the farthest from the heat sink. It is interesting to see that the two resulting thermal profiles are quite different: the balance-by-stack allocator exhibits a checker board profile and the hot region for the RL allocator is centered in the middle of the chip. This observation is consistent with Insight 3 summarized earlier in this chapter. RL agents learn that assignment to the corner cores would result in added thermal stress. We also observe that the thermal maps for the three layers are similar with each other due to strong thermal correlation in the vertical direction. Layer 2 gives the coolest thermal maps for both allocators thanks to the shorter heat dissipation path to the heat sink.

6.5.2 Reduction of Thermal Emergencies

Although cluster-based RL is effective in reducing peak temperature, as shown in the previous section, average temperature reduction is not as significant. However, it can be seen that a significant number of thermal emergencies can be avoided by utilizing information from the tail part of temperature distribution. The upper portion of Figure 6.17 shows the peak temperature distribution for random, balance-by-stack and RL allocators. The figure on the top is the distribution results for non-thermal-aware (random) and RL task allocation approaches. The thermal emergency line is set to 100^oC. There are 1675 thermal-emergency incidents out of 5000 allocations for the non-thermal-aware (random) allocator but only 988 incidents for the RL alloca-

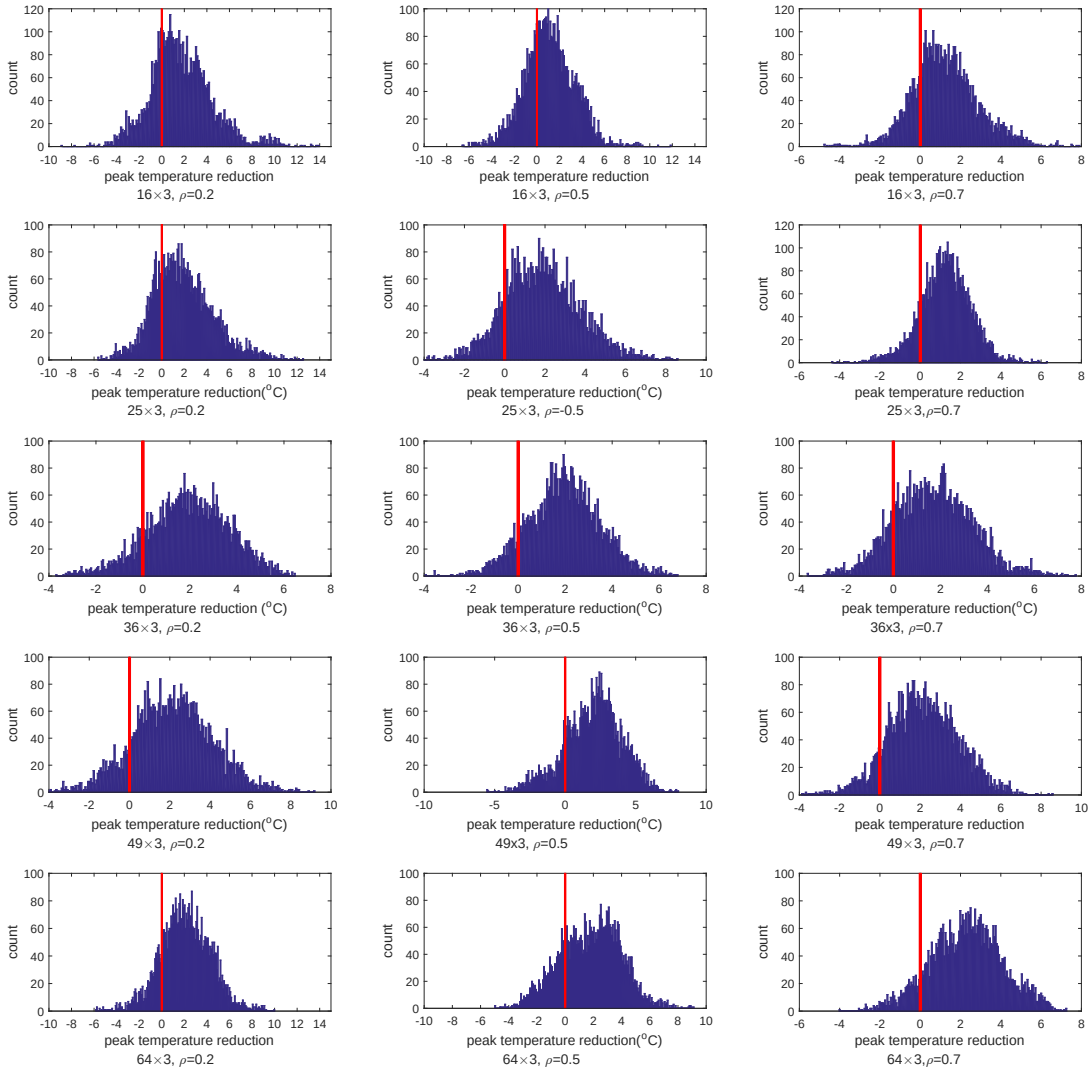


Figure 6.13: Distributions of peak temperature reduction under various core count and system utilization configurations

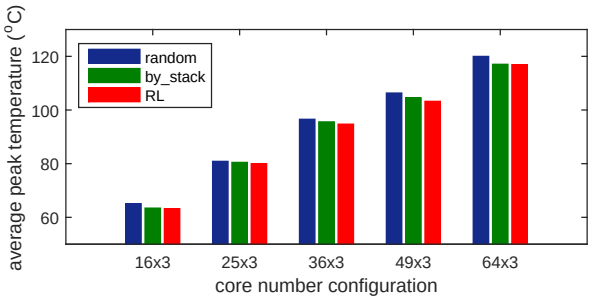


Figure 6.14: Average peak temperature comparison for random, balance-by-statck [120] and RL approaches

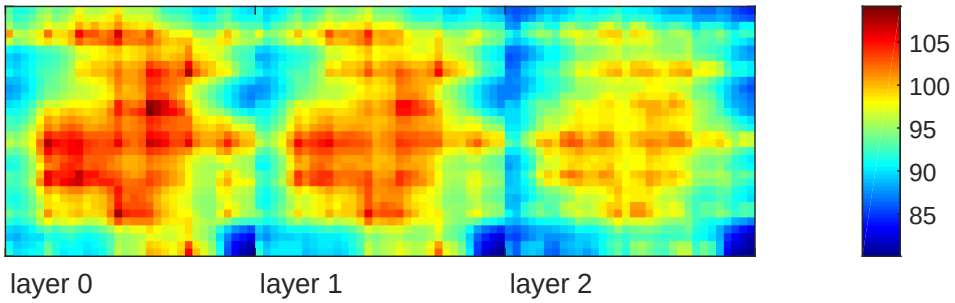


Figure 6.15: The thermal profile for RL allocator after 5000 allocation

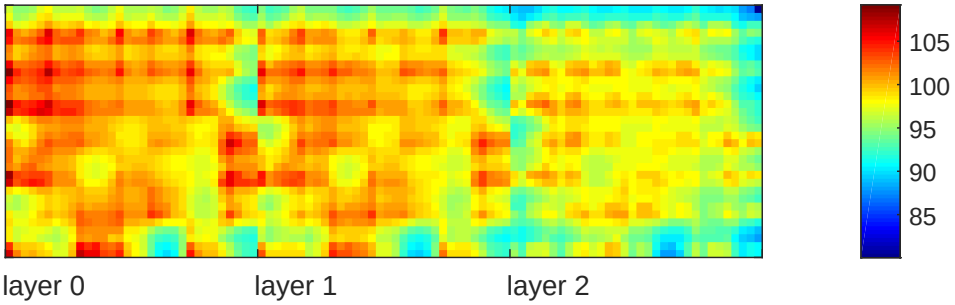


Figure 6.16: The thermal profile for balance-by-stack allocator after 5000 allocation

tor. Therefore, 41% of potential thermal emergencies can be avoided by using the RL approach.

Similarly, the lower portion of Figure 6.17 shows the distribution comparison between balance-by-stack and RL approaches. There are 1162 thermal-emergency incidents for the balance-by-stack approach. In comparison, RL shows a 14.9% reduction versus this number. Since a thermal emergency triggers performance throttling (e.g. frequency reduction), the removal of thermal emergencies brings performance benefits. We assume a 20% performance penalty for each thermal emergency in determining potential performance benefits (note that a DVFS strategy usually has multiple frequency and voltage settings). The removal of emergencies by the RL allocator roughly accounts for 3% and 0.7% performance improvements versus random and balance-by-stack approaches, respectively.

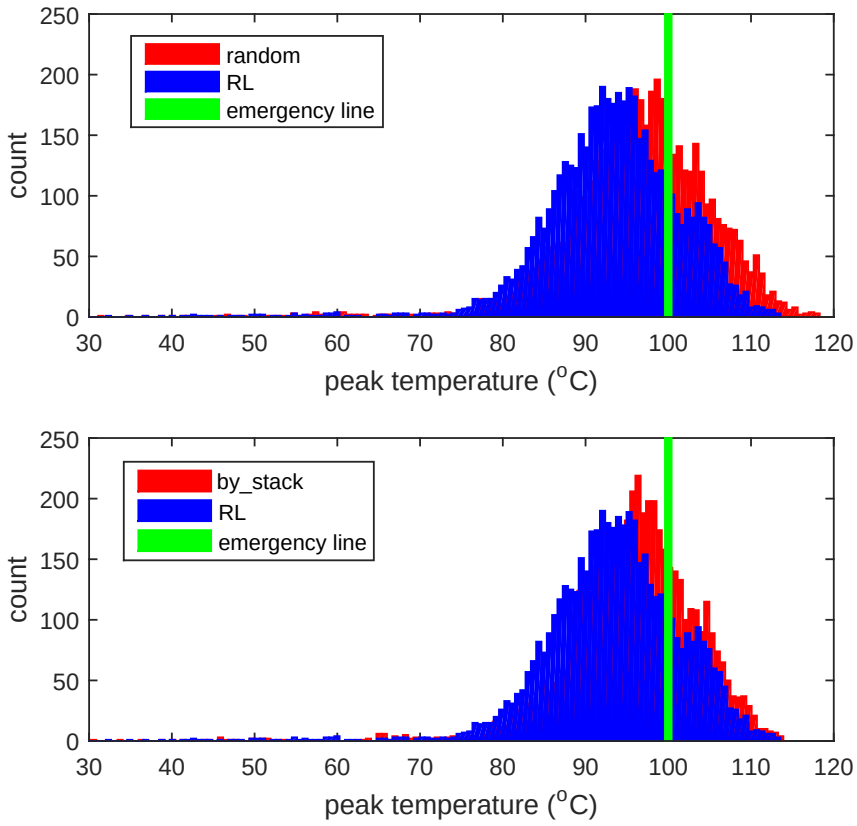


Figure 6.17: Peak temperature distribution comparison between random, balance_by_stack and RL approaches.

Figure 6.18 shows the number of thermal emergencies for various core counts and system utilizations. The numbers are normalized to the corresponding random cases. The average reduction of emergencies is 36% compared with the random approach and 9% compared with the balance-by-stack approach.

6.5.3 Computational and Memory Overhead

The computational overhead of our approach is comprised of two components: cluster selection and reinforcement learning. The complexity of reinforcement learning is $O(2^m)$ for weight parameter (θ_i^a) update as shown in Chapter 5. The time cost of cluster selection is negligible (linear in number of clusters), since it simply involves the selection of the coolest cluster.

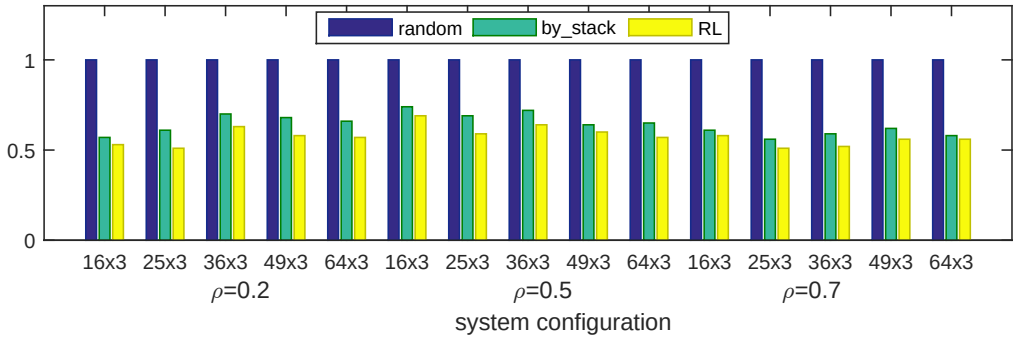


Figure 6.18: Thermal emergency incidents for various core counts and system utilization settings

Table 6.4: Time Cost For RL Task Allocation (ms)

Sensor Count m				
5	6	7	8	9
0.014	0.024	0.043	0.081	0.150

The time cost of the allocator was evaluated for different values of sensor count, m. Table 6.4 shows the average computational time for one task allocation. Two temperature centers were used to generate basis function. The maximum computation time of task allocation is $0.15ms$. Since task allocation is only invoked when there is an incoming task or a thermal emergency, the frequency of allocation is typically on the order of seconds. Therefore, the percentage time cost of allocation with respect to the allocation interval is negligible. For example, we use nine sensor readings to represent the thermal state in a 16-core cluster, so the overhead is $< 0.2ms$.

The memory cost of the allocation algorithm is based on the storage of the θ_i^a weight parameters for each cluster. The total number of parameters for each core is v^m (v and m are defined in Section 5.2.6). However, m (number of temperature sensors) is generally small and can be limited to reduce memory impact. For $v = 2$ and $m = 9$, the memory overhead is only 32KB. Based on our clustering scheme

Table 6.5: Memory Cost for Different Core Numbers (KB))

Core Count				
16×3	25×3	36×3	48×3	64×3
96	384	384	384	480

shown in Figure 6.10, the memory overhead is listed in Table 6.5. For contemporary servers with gigabyte memories, this is a small fraction of total memory.

6.6 Summary

In this chapter, reinforcement learning-based task allocation is applied to address localized overheating in 3-D many-core systems. A new cost function and distributed use of allocation to clusters is added to the model from Chapter 5 to support 3D systems. Our algorithm is verified via detailed many-core simulation which includes on-chip routing. The experiments show that the technique is capable of capturing the complex on-chip thermal environment induced by dynamic work load distribution. Our results show that the proposed technique is fast (scheduling performed in <0.2 ms) and can efficiently reduce peak temperature by $\sim 2^oC$ on average or up to 10^oC for a collection of SPLASH-2 benchmarks. The approach reduces thermal emergency count by 36% versus non-thermal-aware allocation.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Temperature control is a fundamental issue for integrated circuits as it impacts circuit reliability and limits performance. Significant previous work has been conducted to explore different aspects of thermal issues, ranging from on-chip thermal sensing to temperature control techniques. The first half of this dissertation proposed temperature estimation and calibration techniques which utilize system statistics and thermal sensor information to achieve more accurate sensing results. This work falls into the thermal sensing category. In the second half of the dissertation, software techniques are explored to reduce the peak temperature on a silicon die during application execution. This work falls into the temperature control category.

To summarize, the four main contributions made in this dissertation include:

(1) **A fine-grained thermal estimation technique**.

Two linear models were built to estimate the steady and transient temperatures of a variety of architectural components in a microprocessor. The steady state temperature is estimated using a absolute temperature estimation model and the transient temperature is estimated using an incremental temperature estimation model. To dynamically account for changing processor activities, collections of performance counter values were used to estimate the chip thermal profile at run time. A performance counter selection method is employed to reduce the intercorrelations between readings and improve estimation accuracy. Our results show that the correlation coefficient between estimated and actual thermal profiles is ~ 0.9 on a collection of benchmarks. The proposed

estimation model can be adapted to changing cooling conditions via parameter modeling.

(2) **An on-line thermal data fusion strategy**. Multiple sensors deployed in a processor are dynamically calibrated via the Multi-Sensor Collaborative Calibration Algorithm (MSCCA) and the Δ-based Multi-Sensor Collaborative Calibration Algorithm (Δ-MSCCA). Our calibration approach combines potentially inaccurate temperature values obtained from two sources: temperature readings from thermal sensors and temperature estimations using system performance counters. A data fusion strategy based on Bayesian inference, which combines information from these two sources, is demonstrated along with a temperature estimation approach using performance counters. The result shows the strategy can effectively recalibrate sensor readings in response to inaccuracies caused by process variation and environmental noise. The average absolute error of the corrected sensor temperature readings is $< 1.5^oC$ and the standard deviation of error is less than $< 0.5^oC$ for tested benchmarks. The strategy incurs significantly reduced computational cost versus a previously-developed Kalman filtering technique and is appropriate for on-line usage.

(3) **A dynamic task allocation strategy with thermal awareness**. The goal of the developed algorithm is to overcome localized overheating in many-core systems due to processor core and router power consumption. Our approach employs *reinforcement learning*, a dynamic, machine learning algorithm that performs task allocation based on current temperature and a prediction regarding which assignment will minimize maximum temperature. The algorithm updates prediction models after each allocation based on feedback regarding the accuracy of previous predictions. Our new algorithm is verified via detailed many-core simulation which includes on-chip routing. The experiments show

that the proposed technique is capable of capturing the complex on-chip thermal environment induced by dynamic work load distribution. The results show that the proposed technique is fast (scheduling performed in $< 1ms$) and can efficiently reduce peak temperature by up to 8^oC in a 49-core processor (6% on average) versus a leading competing task allocation approach for a series of SPLASH-2 benchmarks.

(4) **A dynamic strategy to control temperatures in 3D ICs**. Reinforcement learning has been applied to 3D integrated circuits to allocate tasks with thermal awareness. To avoid significant performance degradation and computational overhead for large core counts, a cluster based approach is used when applying this dynamic learning technique. Our results show that the proposed technique is fast (scheduling performed in < 0.2 ms) and can efficiently reduce peak temperature by $\sim 2^oC$ on average or up to 10^oC versus the previous task allocation approach for a series of SPLASH-2 benchmarks. The peak temperature reduction eliminates 36% of thermal emergencies which occur when non-thermal-aware task allocation is used.

Thermal control remains an active research area, particularly as three dimensional integrated circuits become more common. It is desirable to obtain detailed and accurate temperature information spatially and temporally to perform fine grained temperature control strategies. In the future, the following research areas warrant exploration.

- Low cost thermal sensors. The design of on-chip thermal sensors should attract increased research interest to push sensor design to physical limits. Another important aspect of thermal sensor design is the exploration of design automation tools to automatically embed these sensors at the best die locations. This goal requires a better understanding of the thermal implications of circuit design

and the use of appropriate sensor allocation strategies to identify thermal hot spots.

- Cooling technologies and low power techniques. As 3D technology becomes more prevalent, it is important to address temperature issues. Advanced technologies should be developed to remove heat efficiently. Additionally, both hardware and software designers should take temperature factors into account to avoid unnecessary overheating. This second observation requires designers to consider the temperature as a first order design factor. The use of low power techniques is perhaps the most important approach which can be used to reduce system-level heat sources.

BIBLIOGRAPHY

[1] Athlon 64 Clawhammer. http://www.cpu-world.com/.

[2] Hotspot. http://lava.cs.virginia.edu/HotSpot/.

[3] Andrei, A., Eles, P., Peng, Zebo, Schmitz, M.T., and Hashimi, B.M.A. Energy optimization of multiprocessor systems on chip by voltage selection. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 15*, 3 (March 2007), 262–275.

[4] Baird, Leemon, and Moore, Andrew W. Gradient descent for general reinforcement learning. *Advances in neural information processing systems* (1999), 968–974.

[5] Barto, Andrew G. *Reinforcement learning: An introduction*. MIT press, 1998.

[6] Bayle, F., and Mettas, A. Temperature acceleration models in reliability predictions & improvements. In *Reliability and Maintainability Symposium (RAMS), 2010 Proceedings - Annual* (Jan 2010), pp. 1–6.

[7] Bharath, K., Yao, Chunhua, Kim, Nam Sung, Ramanathan, P., and Saluja, K.K. A Low Cost Approach to Calibrate On-chip Thermal Sensors. In *Quality Electronic Design (ISQED), 2011 12th International Symposium on* (March 2011), pp. 1 –5.

[8] Breen, T.J., Walsh, E.J., Punch, J., Shah, A.J., and Bash, C.E. From chip to cooling tower data center modeling: Part i influence of server inlet temperature and temperature rise across cabinet. In *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2010 12th IEEE Intersociety Conference on* (June 2010), pp. 1–10.

[9] Brooks, D., and Martonosi, M. Dynamic thermal management for high-performance microprocessors. In *International Symposium on High Performance Computer Architecture* (Jan. 2001), pp. 171 –182.

[10] Carlson, Trevor E., Heirman, Wim, and Eeckhout, Lieven. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (Nov. 2011).

[11] Chantem, T., Dick, R.P., and Hu, X.S. Temperature-aware scheduling and assignment for hard real-time applications on mpsocs. In *Design, Automation and Test in Europe, 2008. DATE '08* (March 2008), pp. 288–293.

[12] Chao, Chih-Hao, Jheng, Kai-Yuan, Wang, Hao-Yu, Wu, Jia-Cheng, and Wu, An-Yeu. Traffic- and thermal-aware run-time thermal management scheme for 3d noc systems. In *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on* (2010), pp. 223–230.

[13] Chen, Hui, Kesavan, Mukil, Schwan, Karsten, Gavrilovska, Ada, Kumar, Pramod, and Joshi, Yogendra. Spatially-aware optimization of energy consumption in consolidated data center systems. In *ASME 2011 Pacific Rim Technical Conference and Exhibition on Packaging and Integration of Electronic and Photonic Systems* (2011), American Society of Mechanical Engineers, pp. 461–470.

[14] Chen, Jian-Jia, and Kuo, Chin-Fu. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms. In *Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE International Conference on* (Aug 2007), pp. 28–38.

[15] Chen, Rong, Chen, Haibo, and Zang, Binyu. Tiled-mapreduce: Optimizing resource usages of data-parallel applications on multicore with tiling. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques* (New York, NY, USA, 2010), PACT '10, ACM, pp. 523–534.

[16] Chen, Zhuo, and Marculescu, Diana. Distributed reinforcement learning for power limited many-core system performance optimization. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition* (San Jose, CA, USA, 2015), DATE '15, EDA Consortium, pp. 1521–1526.

[17] Cheng, Yuanqing, Zhang, Lei, Han, Yinhe, and Li, Xiaowei. Thermal-constrained task allocation for interconnect energy reduction in 3-d homogeneous mpsocs. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 21*, 2 (Feb 2013), 239–249.

[18] Choi, Jeonghwan, Cher, Chen-Yong, Franke, Hubertus, Hamann, Henrdrik, Weger, Alan, and Bose, Pradip. Thermal-aware task scheduling at the system software level. In *Proceedings of the 2007 international symposium on Low power electronics and design* (2007), ACM, pp. 213–218.

[19] Chowdhury, Ihtesham, Prasher, Ravi, Lofgreen, Kelly, Chrysler, Gregory, Narasimhan, Sridhar, Mahajan, Ravi, Koester, David, Alley, Randall, and Venkatasubramanian, Rama. On-chip cooling by superlattice-based thin-film thermoelectrics. *Nature Nanotechnology 4*, 4 (2009), 235–238.

[20] Chrobak, Marek, Dürr, Christoph, Hurand, Mathilde, and Robert, Julien. Algorithms for temperature-aware task scheduling in microprocessor systems. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management* (Berlin, Heidelberg, 2008), AAIM '08, Springer-Verlag, pp. 120–130.

[21] Cochran, R., and Reda, S. Spectral techniques for high-resolution thermal characterization with limited sensor data. In *ACM/IEEE Design Automation Conference* (July 2009), pp. 478–483.

[22] Cochran, Ryan, and Reda, Sherief. Thermal prediction and adaptive control through workload phase detection. *ACM Trans. Des. Autom. Electron. Syst. 18*, 1 (Jan. 2013), 7:1–7:19.

[23] Coskun, A.K., Rosing, T.S., Whisnant, K.A., and Gross, K.C. Static and dynamic temperature-aware scheduling for multiprocessor socs. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 16*, 9 (Sept 2008), 1127–1140.

[24] Coskun, Ayse Kivilcim, Rosing, Tajana Simunic, and Whisnant, Keith. Temperature aware task scheduling in MPSoCs. In *IEEE/ACM Conf. on Design, Automation and Test in Europe* (Mar. 2007), pp. 1659–1664.

[25] Cox, M., Singh, A.K., Kumar, A., and Corporaal, H. Thermal-aware mapping of streaming applications on 3d multi-processor systems. In *Embedded Systems for Real-time Multimedia (ESTIMedia), 2013 IEEE 11th Symposium on* (Oct 2013), pp. 11–20.

[26] Daneshtalab, M., Sobhani, A., Afzali-Kusha, A., Fatemi, O., and Navabi, Z. Noc hot spot minimization using antnet dynamic routing algorithm. In *Application-specific Systems, Architectures and Processors, 2006. ASAP '06. International Conference on* (Sept 2006), pp. 33–38.

[27] Das, Reetuparna, Narayanasamy, Satish, Satpathy, Sudhir K., and Dreslinski, Ronald G. Catnap: Energy proportional multiple network-on-chip. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2013), ISCA '13, ACM, pp. 320–331.

[28] Datta, B., and Burleson, W. Calibration of on-chip thermal sensors using process monitoring circuits. In *International Symposium on Quality Electronic Design* (March 2010), pp. 461 –467.

[29] Dilip, B, Prasad, P Surya, and Bhavani, RSG. Leakage power reduction in cmos circuits using leakage control transistor technique in nanoscale technology. *International Conference on Advances in Electrical and Electronics Engineering (AEEE), ISBN*, 978–93.

[30] Ebi, T., Kramer, D., Karl, W., and Henkel, J. Economic learning for thermal-aware power budgeting in many-core architectures. In *Proc. CODES+ISSS* (Oct 2011).

[31] Elnahrawy, Eiman, and Nath, Badri. Cleaning and querying noisy sensors. In *International Conference on Wireless Sensor Networks and Applications* (Sept. 2003), pp. 78–87.

[32] Esmaeilzadeh, Hadi, Blem, Emily, St. Amant, Renee, Sankaralingam, Karthikeyan, and Burger, Doug. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2011), ISCA '11, ACM, pp. 365–376.

[33] Fisher, Nathan, Chen, Jian-Jia, Wang, Shengquan, and Thiele, Lothar. Thermal-aware global real-time scheduling on multicore systems. In *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE* (April 2009), pp. 131–140.

[34] Floyd, M., Allen-Ware, M., Rajamani, K., Brock, B., Lefurgy, C., Drake, A.J., Pesantez, L., Gloekler, T., Tierno, J.A., Bose, P., and Buyuktosunoglu, A. Introducing the adaptive energy management features of the power7 chip. *Micro, IEEE 31*, 2 (March 2011), 60–75.

[35] Floyd, M., Ware, M., Rajamani, K., Gloekler, T., Brock, B., Bose, P., Buyuktosunoglu, A., Rubio, J.C., Schubert, B., Spruth, B., Tierno, J.A., and Pesantez, L. Adaptive energy-management features of the IBM POWER7 chip. *IBM Journal of Research and Development 55*, 3 (2011), 8:1–8:18.

[36] Ge, Yang, and Qiu, Qinru. Dynamic thermal management for multimedia applications using machine learning. In *Proc. DAC* (June 2011), pp. 95–100.

[37] Gupta, M.P., Cho, Minki, Mukhopadhyay, S., and Kumar, Satish. Thermal mangament of multicore processors using power multiplexing. In *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2010 12th IEEE Intersociety Conference on* (June 2010), pp. 1–7.

[38] Ha, Dongwan, Woo, Kyoungho, Meninger, S., Xanthopoulos, T., Crain, E., and Ham, Donhee. Time-domain cmos temperature sensors with dual delay-locked loops for microprocessor thermal monitoring. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 20*, 9 (Sept 2012), 1590–1601.

[39] Hamann, H. F., Weger, A., Lacey, J. A., Hu, Z., Bose, P., Cohen, E., and Wakil, J. Hotspot-limited microprocessors: Direct temperature and power distribution measurements. *IEEE Journal of Solid-State Circuits 42*, 1 (Jan. 2007), 56–65.

[40] Hanumaiah, V., Rao, R., Vrudhula, S., and Chatha, K.S. Throughput optimal task allocation under thermal constraints for multi-core processors. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE* (July 2009), pp. 776–781.

[41] Hanumaiah, V., and Vrudhula, S. Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling. *Computers, IEEE Transactions on 63*, 2 (Feb 2014), 349–360.

[42] Hung, W., Addo-Quaye, C., Theocharides, T., Xie, Y., Vijakrishnan, N., and Irwin, M.J. Thermal-aware ip virtualization and placement for networks-on-chip architecture. In *Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on* (2004), pp. 430–437.

[43] Hung, W. L, Link, G.M., Xie, Yuan, Vijaykrishnan, N., and Irwin, M.J. Interconnect and thermal-aware floorplanning for 3d microprocessors. In *Quality Electronic Design, 2006. ISQED '06. 7th International Symposium on* (March 2006), pp. 6 pp.–104.

[44] Johnson, R., and Wichern, D. *Applied Multivariate Statistical Analysis*. Prentice Hall, 2007.

[45] Juan, Da-Cheng, and Marculescu, Diana. Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design* (New York, NY, USA, 2012), ISLPED '12, ACM, pp. 97–102.

[46] Jung, Moongon, and Lim, Sung Kyu. A study of ir-drop noise issues in 3d ics with through-silicon-vias. In *3D Systems Integration Conference (3DIC), 2010 IEEE International* (Nov 2010), pp. 1–7.

[47] Kim, Wonyoung, Gupta, M.S., Wei, Gu-Yeon, and Brooks, D. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on* (Feb 2008), pp. 123–134.

[48] Kumar, A., Shang, Li, Peh, Li-Shiuan, and Jha, N.K. System-level dynamic thermal management for high-performance microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27*, 1 (Jan. 2008), 96 –108.

[49] Lee, K.-J., and Skadron, K. Using performance counters for runtime temperature sensing in high-performance processors. In *IEEE International Parallel and Distributed Processing Symposium* (April 2005), p. 232.

[50] Li, Dong, Chang, Hung-Ching, Pyla, Hari K., and Cameron, K.W. System-level, thermal-aware, fully-loaded process scheduling. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on* (April 2008), pp. 1–7.

[51] Li, Jiayin, Qiu, Meikang, Niu, Jian-Wei, Yang, Laurence T., Zhu, Yongxin, and Ming, Zhong. Thermal-aware task scheduling in 3d chip multiprocessor with real-time constrained workloads. *ACM Trans. Embed. Comput. Syst. 12*, 2 (Feb. 2013), 24:1–24:22.

[52] Li, Sheng, Ahn, Jung-Ho, Strong, R.D., Brockman, J.B., Tullsen, D.M., and Jouppi, N.P. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on* (2009), pp. 469–480.

[53] Lian, Chenzhou, Knox, M., Sikka, K., Wei, Xiaojin, and Weger, A.J. Development of a flexible chip infrared (IR) thermal imaging system for product qualification. In *Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM), 2012 28th Annual IEEE* (2012), pp. 337–343.

[54] Lin, Sheng-Chih, and Banerjee, K. Cool chips: Opportunities and implications for power and thermal management. *Electron Devices, IEEE Transactions on 55*, 1 (Jan 2008), 245–255.

[55] Lin, Shu-Yen, Yin, Tzu-Chu, Wang, Hao-Yu, and Wu, An-Yeu. Traffic-and thermal-aware routing for throttled three-dimensional network-on-chip systems. In *VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on* (April 2011), pp. 1–4.

[56] Liu, Feiyang, Gu, Huaxi, and Yang, Yintang. Dtbr: A dynamic thermal-balance routing algorithm for network-on-chip. *Computers & Electrical Engineering 38*, 2 (2012), 270–281.

[57] Liu, Frank. A general framework for spatial correlation modeling in VLSI design. In *IEEE/ACM Design Automation Conference* (June 2007), pp. 817–822.

[58] Liu, Guanglei, Fan, Ming, and Quan, Gang. Neighbor-aware dynamic thermal management for multi-core platform. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012* (March 2012), pp. 187–192.

[59] Liu, Shaobo, and Qiu, Meikang. Thermal-aware scheduling for peak temperature reduction with stochastic workloads. In *16th IEEE Real-Time and Embedded Technology and Applications Symposium, Stockholm: WIP* (2010), pp. 59–62.

[60] Liu, Shaobo, Zhang, Jingyi, Wu, Qing, and Qiu, Qinru. Thermal-aware job allocation and scheduling for three dimensional chip multiprocessor. In *Quality Electronic Design (ISQED), 2010 11th International Symposium on* (March 2010), pp. 390–398.

[61] Liu, Yongpan, Dick, R., Shang, Li, and Yang, Huazhong. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *Design, Automation and Test in Europe* (2007).

[62] Lopez-Buedo, Sergio, and Boemo, Eduardo. Making visible the thermal behaviour of embedded microprocessors on fpgas: A progress report. In *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays* (New York, NY, USA, 2004), FPGA '04, ACM, pp. 79–86.

[63] Lu, Shiting (Justin), Tessier, Russell, and Burleson, Wayne. Collaborative calibration of on-chip thermal sensors using performance counters. In *Proceedings of the International Conference on Computer-Aided Design* (New York, NY, USA, 2012), ICCAD '12, ACM, pp. 15–22.

[64] Lu, Shiting (Justin), Tessier, Russell, and Burleson, Wayne. Reinforcement learning for thermal-aware many-core task allocation. In *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI* (New York, NY, USA, 2015), GLSVLSI '15, ACM, pp. 379–384.

[65] Lu, S.J., Tessier, R., and Burleson, W. Dynamic on-chip thermal sensor calibration using performance counters. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 33*, 6 (June 2014), 853–866.

[66] Lung, Chiao-Ling, Ho, Yi-Lun, Kwai, Ding-Ming, and Chang, Shih-Chieh. Thermal-aware on-line task allocation for 3d multi-core processor throughput optimization. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011* (March 2011), pp. 1–6.

[67] MacKay, David. Bayesian interpolation. *Neural Comput. 4*, 3 (May 1992), 415–447.

[68] McGowen, R., Poirier, C.A., Bostak, C., Ignowski, J., Millican, M., Parks, W.H., and Naffziger, S. Power and temperature control on a 90-nm itanium family processor. *Solid-State Circuits, IEEE Journal of 41*, 1 (Jan 2006), 229–237.

[69] Meng, Jie, Kawakami, Katsutoshi, and Coskun, Ayse K. Optimizing energy efficiency of 3-d multicore systems with stacked dram under power and thermal constraints. In *Proceedings of the 49th Annual Design Automation Conference* (New York, NY, USA, 2012), DAC '12, ACM, pp. 648–655.

[70] Mesa-Martinez, Francisco Javier, Ardestani, Ehsan K, and Renau, Jose. Characterizing processor thermal behavior. In *ACM SIGARCH Computer Architecture News* (2010), vol. 38, ACM, pp. 193–204.

[71] Mesa-Martinez, Francisco Javier, Nayfach-Battilana, Joseph, and Renau, Jose. Power model validation through thermal measurements. In *International Symposium on Computer Architecture* (June 2007), pp. 302–311.

[72] Mukherjee, K., Khuller, S., and Deshpande, A. Algorithms for the thermal scheduling problem. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on* (May 2013), pp. 949–960.

[73] Naveh, Alon, Rajwan, Doron, Ananthakrishnan, Avinash, and Weissmann, Eli. Power management architecture of the 2nd generation intel® core microarchitecture, formerly codenamed sandy bridge.

[74] Neal, Radford M. *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[75] Nookala, Vidyasagar, Lilja, David J., and Sapatnekar, Sachin S. Temperature-aware floorplanning of microarchitecture blocks with ipc-power dependence modeling and transient analysis. In *Proceedings of the 2006 International Symposium on Low Power Electronics and Design* (New York, NY, USA, 2006), ISLPED '06, ACM, pp. 298–303.

[76] Paek, Seungwook, Shin, Wongyu, Lee, Jaeyoung, Kim, Hyo-Eun, Park, Jun-Seok, and Kim, Lee-Sup. All-digital hybrid temperature sensor network for dense thermal monitoring. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International* (Feb 2013), pp. 260–261.

[77] Pan, Gung-Yu, Jou, Jing-Yang, and Lai, Bo-Cheng. Scalable power management using multilevel reinforcement learning for multiprocessors. *ACM Trans. Des. Autom. Electron. Syst. 19*, 4 (Aug. 2014), 33:1–33:23.

[78] Powell, M.D., Biswas, A., Emer, J.S., Mukherjee, S.S., Sheikh, B.R., and Yardi, S. CAMP: A technique to estimate per-structure power at run-time using a few simple parameters. In *IEEE International Symposium on High Performance Computer Architecture* (Feb. 2009), pp. 289–300.

[79] Qian, Zhiliang, and Tsui, Chi-Ying. A thermal-aware application specific routing algorithm for network-on-chip design. In *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific* (2011), pp. 449–454.

[80] Ranieri, J., Vincenzi, A., Chebira, A., Atienza, D., and Vetterli, M. Eigenmaps: Algorithms for optimal thermal maps extraction and sensor placement on multicore processors. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE* (2012), pp. 636–641.

[81] Reda, S. Thermal and power characterization of real computing devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems 1*, 2 (June 2011), 76 –87.

[82] Remarsu, Spandana, and Kundu, Sandip. On process variation tolerant low cost thermal sensor design in 32nm CMOS technology. In *ACM Great Lakes Symposium on VLSI* (May 2009), pp. 487–492.

[83] Renau, Jose, Fraguela, Basilio, Tuck, James, Liu, Wei, Prvulovic, Milos, Ceze, Luis, Sarangi, Smruti, Sack, Paul, Strauss, Karin, and Montesinos, Pablo. SESC simulator, January 2005. http://sesc.sourceforge.net.

[84] Rotem, E., Naveh, A., Rajwan, D., Ananthakrishnan, A., and Weissmann, E. Power management architecture of the 2nd generation Intel core microarchitecture, formerly codenamed Sandy Bridge. In *Hot Chips: A Symposium on High Performance Chips* (August 2011).

[85] Sabry, M.M., Sridhar, A., Meng, Jie, Coskun, A.K., and Atienza, D. Greencool: An energy-efficient liquid cooling design technique for 3-d mpsocs via channel width modulation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 32*, 4 (April 2013), 524–537.

[86] Sabry, Mohamed M., Ayala, José L., and Atienza, David. Thermal-aware compilation for system-on-chip processing architectures. In *Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI* (New York, NY, USA, 2010), GLSVLSI '10, ACM, pp. 221–226.

[87] Sankaranarayanan, Karthik, Velusamy, Sivakumar, Stan, Mircea, and Skadron, Kevin. A case for thermal-aware floorplanning at the microarchitectural level. *Journal of Instruction-Level Parallelism 7*, 1 (2005), 8–16.

[88] Schafer, B.C., Lee, Yongho, and Kim, Taewhan. Temperature-aware compilation for vliwprocessors. In *Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE International Conference on* (Aug 2007), pp. 426–431.

[89] Schlaepfer, E. External temperature sensor calibration for the max16031/max16032 system monitors. Tech. rep., Maxim application note 4284, 2008.

[90] Shang, Li, Peh, Li-Shiuan, Kumar, Amit, and Jha, Niraj K. Temperature-aware on-chip networks. *IEEE Micro 26*, 1 (Jan. 2006), 130–139.

[91] Sharifi, S., and Rosing, T.S. Accurate direct and indirect on-chip temperature sensing for efficient dynamic thermal management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 29*, 10 (Oct. 2010), 1586–1599.

[92] Sharifi, S., and Rosing, T.S. Accurate direct and indirect on-chip temperature sensing for efficient dynamic thermal management. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 29*, 10 (2010), 1586–1599.

[93] Shor, J., Luria, K., and Zilberman, D. Ratiometric bjt-based thermal sensor in 32nm and 22nm technologies. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International* (Feb 2012), pp. 210–212.

[94] Shor, J.S., and Luria, K. Miniaturized bjt-based thermal sensor for microprocessors in 32- and 22-nm technologies. *Solid-State Circuits, IEEE Journal of 48*, 11 (Nov 2013), 2860–2867.

[95] Sun, Chen, Chen, C.-H.O., Kurian, G., Wei, Lan, Miller, J., Agarwal, A., Peh, Li-Shiuan, and Stojanovic, V. Dsent - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on* (2012), pp. 201–210.

[96] Tang, Q., Gupta, S.K.S., and Varsamopoulos, G. Thermal-aware task scheduling for data centers through minimizing heat recirculation. In *Cluster Computing, 2007 IEEE International Conference on* (Sept 2007), pp. 129–138.

[97] Testi, N., and Xu, Yang. A 0.2nj/sample 0.01mm2 ring oscillator based temperature sensor for on-chip thermal management. In *Quality Electronic Design (ISQED), 2013 14th International Symposium on* (March 2013), pp. 696–702.

[98] Tripathy, Aalap, Patra, Atish, Mohan, Suneil, and Mahapatra, Rabi. Distributed collaborative filtering on a single chip cloud computer*. In *Proceedings of the 2013 IEEE International Conference on Cloud Engineering* (Washington, DC, USA, 2013), IC2E '13, IEEE Computer Society, pp. 140–145.

[99] Vangal, S., Singh, A., Howard, J., Dighe, S., Borkar, N., and Alvandpour, A. A 5.1ghz 0.34mm2 router for network-on-chip applications. In *VLSI Circuits, 2007 IEEE Symposium on* (2007), pp. 42–43.

[100] Wang, Lizhe, Khan, SameeU., and Dayal, Jai. Thermal aware workload placement with task-temperature profiles in a data center. *The Journal of Supercomputing 61*, 3 (2012), 780–803.

[101] Wang, Xiaodong, Wang, Xiaorui, Xing, Guoliang, Chen, Jinzhu, Lin, Cheng-Xian, and Chen, Yixin. Intelligent sensor placement for hot server detection in data centers. *Parallel and Distributed Systems, IEEE Transactions on 24*, 8 (Aug 2013), 1577–1588.

[102] Whitehouse, Kamin, and Culler, David. Calibration as parameter estimation in sensor networks. In *ACM International Workshop on Wireless Sensor Networks and Applications* (Sept. 2002), pp. 59–67.

[103] Woo, Kyoungho, Meninger, S., Xanthopoulos, Thucydides, Crain, E., Ha, Dongwan, and Ham, D. Dual-dll-based cmos all-digital temperature sensor for microprocessor thermal monitoring. In *Solid-State Circuits Conference - Digest of Technical Papers, 2009. ISSCC 2009. IEEE International* (Feb 2009), pp. 68–69,69a.

[104] Woo, S., Ohara, M., Torrie, E., Singh, J., and Gupta, A. The SPLASH-2 programs: characterization and methodological considerations. In *Proceedings of the International Symposium on Computer Architecture* (1995), pp. 24–36.

[105] Wu, Wei, Jin, Lingling, Yang, Jim, Liu, Pu, and Tan, S.X.-D. A systematic method for functional unit power estimation in microprocessors. In *ACM/IEEE Design Automation Conference* (July 2006), pp. 554 –557.

[106] Xia, Liang, Zhu, Yongxin, Yang, Jun, Ye, Jingwei, and Gu, Zonghua. Implementing a thermal-aware scheduler in linux kernel on a multi-core processor. *The Computer Journal 53*, 7 (2010), 895–903.

[107] Xie, Shuang, and Ng, Wai Tung. A low power all-digital self-calibrated temperature sensor using 65nm fpgas. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on* (May 2013), pp. 2617–2620.

[108] Yang, Teng, Kim, Seongjong, Kinget, P.R., and Seok, Mingoo. 16.4 0.6-to-1.0v 279 μm 2, 0.92μw temperature sensor with less than $+3.2/-3.4c$ error for on-chip dense thermal monitoring. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International* (Feb 2014), pp. 282–283.

[109] Yao, Chunhua, Saluja, Kewal K, and Ramanathan, Parmesh. Calibrating on-chip thermal sensors in integrated circuits: A design-for-calibration approach. *Journal of Electronic Testing 27*, 6 (2011), 711–721.

[110] Yu, C.H., Lung, C., Ho, Y., Hsu, R., Kwai, D., and Chang, S. Thermal-aware on-line scheduler for 3-d many-core processor throughput optimization. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 33*, 5 (May 2014), 763–773.

[111] Yu, Heng, Syed, Rizwan, and Ha, Yajun. Thermal-aware frequency scaling for adaptive workloads on heterogeneous mpsocs. In *Proceedings of the Conference on Design, Automation & Test in Europe* (3001 Leuven, Belgium, Belgium, 2014), DATE '14, European Design and Automation Association, pp. 291:1–291:6.

[112] Yuffe, M., Knoll, E., Mehalel, M., Shor, J., and Kurts, T. A fully integrated multi-cpu, gpu and memory controller 32nm processor. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International* (Feb 2011), pp. 264–266.

[113] Zhang, Tianpei, Zhan, Yong, and Sapatnekar, S.S. Temperature-aware routing in 3D ICs. In *Design Automation, 2006. Asia and South Pacific Conference on* (Jan 2006), pp. 309–314.

[114] Zhang, Yufu, and Srivastava, A. Accurate temperature estimation using noisy thermal sensors. In *ACM/IEEE Design Automation Conference* (July 2009), pp. 472 –477.

[115] Zhang, Yufu, and Srivastava, A. Adaptive and autonomous thermal tracking for high performance computing systems. In *ACM/IEEE Design Automation Conference* (June 2010), pp. 68 –73.

[116] Zhang, Yufu, and Srivastava, A. Accurate Temperature Estimation Using Noisy Thermal Sensors for Gaussian and Non-Gaussian Cases. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 19*, 9 (Sept. 2011), 1617 –1626.

[117] Zhao, Dali, Homayoun, H., and Veidenbaum, A.V. Temperature aware thread migration in 3d architecture with stacked dram. In *Quality Electronic Design (ISQED), 2013 14th International Symposium on* (March 2013), pp. 80–87.

[118] Zhao, Jia, Lu, Shiting (Justin), Burleson, Wayne, and Tessier, Russell. Runtime probabilistic detection of miscalibrated thermal sensors in many-core systems. In *Proceedings of the Conference on Design, Automation and Test in Europe* (San Jose, CA, USA, 2013), DATE '13, EDA Consortium, pp. 1395–1398.

[119] Zhao, Jia, Madduri, S., Vadlamani, R., Burleson, W., and Tessier, R. A dedicated monitoring infrastructure for multicore processors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 19*, 6 (2011), 1011–1022.

[120] Zhou, Huapeng, Li, Xin, Cher, Chen-Yong, Kursun, E, Qian, Haifeng, and Yao, Shi-Chune. An information-theoretic framework for optimal temperature sensor allocation and full-chip thermal monitoring. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE* (2012), pp. 642–647.

[121] Zhou, Xiuyi, Yang, Jun, Xu, Yi, Zhang, Youtao, and Zhao, Jianhua. Thermal-aware task scheduling for 3d multicore processors. *Parallel and Distributed Systems, IEEE Transactions on 21*, 1 (2010), 60–71.