Energy Conservation and Thermal Management in High-Performance Server Architectures

A Dissertation

Presented to the

Graduate Faculty of the

University of Louisiana at Lafayette

In Partial Fulfillment of the

Requirements for the Degree

Doctor of Philosophy

Adam Wade Lewis

Spring 2012

Energy Conservation and Thermal Management in High-Performance Server Architectures

Adam Wade Lewis

APPROVED:

Dedication

Dedicated to my parents, Laura Mae and Jeter Wade Lewis, who have been a loving and supportive presence in my life. It has been through their support that this work has been possible.

Epigraph

Never travel without a towel.

–Douglas Adams, *The Hitchhiker's Guide to the Galaxy*, 1980

Acknowledgments

The author extends his thanks to the many people whose contributions and support have improved the overall quality of this dissertation. It has been an honor to work with Professor Nian-Feng Tzeng, my adviser and committee chair. I am profoundly grateful to him for his perceptive and extremely patient mentoring over the past seven years. I have become a much more competent and confident scholar as result of his honest and supportive feedback. I thank Professors Magdy Bayoumi, D'mitri Perkins, and Ashok Kumar for serving on my committee and providing feedback during the defense of my dissertation.

This research was made possible by assistance and support of my collaborators and co-authors. In particular, I am grateful to Soumik Ghosh for his assistance in the early development of the energy and thermal models in this research and to Jim Simon for his assistance in the collection and interpretation of the results that lead to the development of Chaotic Attractor Predictors. This work would not have been possible without the support and assistance of the members of the Computer Architecture and Networking research group in the Center for Advanced Computer Studies. You have my thanks for your willingness to tolerate my quirkiness over the past seven years.

This research was supported in part through a fellowship and funding provided by the Louisiana Board of Regents and through funding provided by the United States Department of Energy.

Table of Contents

Dedication iv

Epigraph v

Acknowledgments vi

List of Tables ix

List of Figures x

Chapter 1. Introduction 1

Chapter 2. Background and Related Work 6

2.1 Power Modeling 6

2.2 Thermal Modeling 9

2.3 Chaotic Behavior of Energy Consumption 11

2.4 Limitations of Existing Scheduling 13

Chapter 3. System Modeling 15

3.1 Processor energy consumption 16

3.2 DRAM energy consumption 18

3.3 Hard disk energy consumption 19

3.4 Board energy consumption 20

3.5 Electromechanical energy consumption 21

3.6 Extending the model for thermal behavior 22

Chapter 4. Effective Prediction 24

4.1 Performance counters and metrics 24

4.2 Linear Auto-Regressive Models 27

4.3 Shortcomings of Linear Models 29

4.4 Chaotic prediction 31

4.4.1 Chaotic Attractor Predictors 33

4.4.2 CAP Creation 35

4.4.3 Time Complexity 36

4.5 Thermal Chaotic Attractor Predictors 38

4.5.1 Thermal CAP Creation 38

Chapter 5. Chaotic Attractor Predictor Evaluation 40

5.1 Evaluation environment 40

5.2 Results 42

5.3 Further discussion 46

Chapter 6. A Thermal Aware Scheduler 48
6.1 Thermal Predictors 48
6.2 Thread Scheduling 49
6.3 Load Balancing 51

Chapter 7. Thermal Aware Scheduler Evaluation 55
7.1 Experiment Setup 55
7.2 SPEC Benchmark Results and Discussion 56
7.3 PARSEC Benchmark Results and Discussion 59

Chapter 8. Conclusions 63
8.1 Future Directions 64

Bibliography 68

Abstract 76

Biographical Sketch 78

List of Tables

Table 3.1 Hitachi HDT725025VLA360 disk power parameters 19

Table 4.1 PeCs and performance metrics for AMD Opteron server 26

Table 4.2 PeCs and performance metrics for Intel Nehalem server 26

Table 4.3 Indications of chaotic behavior in power time series (AMD, Intel) 32

Table 4.4 Chaotic behavior in core die temperature time series 33

Table 4.5 SPEC CPU2006 benchmarks used for model calibration 36

Table 5.1 Server configurations for evaluation 41

Table 5.2 SPEC CPU2006 benchmarks used for evaluation 41

Table 5.3 Branch and Memory Access Patters of Evaluation Benchmarks 42

Table 5.4 Model errors for CAP (under $n = 5$, $p = 100$, $r = 14$), AR(1), MARS, and EWMA on AMD Opteron server 42

Table 5.5 Model errors for CAP (under $n = 5$, $p = 100$, $r = 19$), AR, MARS, and EWMA predictors on Intel Nehalem server 46

Table 7.1 Processor used in evaluation 56

Table 7.2 Branch and Memory Access Patters of Evaluation Benchmarks 56

Table 7.3 Temperature and runtime results under benchmark mixes 57

Table 7.4 PARSEC benchmarks used in evaluation 58

List of Figures

Figure 3.1 AMD Opteron architecture. 16

Figure 3.2 Intel Xeon (Nehalem) architecture. 16

Figure 4.1 Actual power results versus AR(1) predicted results for AMD Opteron and Intel processors. 30

Figure 4.2 CAP time complexity versus no. of future observations. 37

Figure 4.3 CAP time complexity versus no. of past observations. 38

Figure 5.1 Root Mean Square Error (RMSE) for different values of p. 43

Figure 5.2 Actual power results versus predicted results for AMD Opteron. 44

Figure 5.3 Actual power results versus predicted results for an Intel Nehalem server. 45

Figure 6.1 TEAPlant and TEAHarvest data collection. 49

Figure 6.2 Internal structure of the TAS scheduler. 49

Figure 6.3 TAS thermal queue selection. 51

Figure 6.4 TAS thermal load balancing. 52

Figure 6.5 Pseudo code of TAS balancing algorithm. 53

Figure 7.1 Comparison of PARSEC benchmark average core die temperatures under ULE and TAS schedulers. 59

Figure 7.2 Comparison of PARSEC benchmark performance under ULE and TAS schedulers. 61

Figure 7.3 Comparison of PARSEC benchmark average power dissipation under ULE and TAS schedulers. 61

Figure 7.4 Comparison of PARSEC total energy consumption under ULE and TAS schedulers. 62

Chapter 1

Introduction

The upwardly spiraling operating costs of the infrastructure for enterprise-scale computing demand efficient power management in data centers. It is difficult to achieve efficient power management in such environments, as they usually over-provision their power capacity to address the worst case scenarios. A recent study by the United States Department of Energy [Linter et al. 2011] reports that energy consumed by information technology equipment in data centers accounts for over half of an entire facility's energy use. As servers with multi-core processors has become more common in information technology equipment used in data centers, opportunities for greater power and thermal efficiencies arise from: (1) improved performance with the same power and cooling load and (2) consolidation of shared devices onto a single processor package. However, applications such as high performance computing codes have characteristics that make ineffectual techniques that have been used in the past to take advantage of these opportunities. This calls for the development of alternative methods that better reflect the quantitative relationship between power consumption and thermal envelope at the system level and better optimize the use of deployed power capacity in the data center.

Current AMD and Intel processors employ different techniques for processor-level power management [AMD 2008; Intel 2009]. In general, those techniques take advantage of the fact that reducing switching activity within the processor lowers energy consumption, and that application performance can be adjusted to utilize otherwise idle time slacks on the processor for energy savings [Contreras and Martonosi 2005]. Modern processors crudely manage thermal emergencies through Dynamic Thermal Management (DTM), where the processor monitors the die temperature and dynamically adjusts the processor voltage and frequency (known as Dynamic Voltage and Frequency Scaling (DVFS)) to throttle down the processor whenever necessary.

However, the use of DVFS tends to cause significant negative impacts on application performance and system reliability [Donald and Martonosi 2006; Bircher and John 2008; Coskun et al. 2008]. Power profiles of the Intel Pentium architecture have been studied for workstations [Isci and Martonosi 2003a; 2003b; 2006] and servers [Bircher et al. 2004; Bircher and John 2007; Lee and Skadron 2005]. Recent investigation into the power profiles of servers based on the NUMA architecture (for example, the AMD64 [AMD 2007], the Intel Nehalem [Intel 2009], and IBM POWER7 processors [Ware et al. 2010; Brochard et al. 2010]) indicates that managing server power profiles is complicated by the existence of multiple cores per processor [Kansal et al. 2010; Tsirogiannis et al. 2010; Lewis et al. 2010; McCullough et al. 2011].

The current generation of operating systems treats the cores in multi-core and virtualized multi-threaded processors (like those with Intel's HyperThreading technology) as distinct logical units, called *logical cores*, which are scheduled independently. However, dependency and contention exist in shared resources among those logical cores, and hence they need to be taken into account upon their scheduling to address performance and energy efficiency. One approach based on software optimization at the application level aimed to make codes themselves more aware of existing dependencies [Khan et al. 2011]. While effective, such an approach may not be applicable in all cases and does not possess economies of scale. It thus calls for the need of intelligent, thermal-aware load balancing and scheduling within the operating system, achievable via modeling full-system energy consumption based on computational load for effectively predicting future energy consumption and its associated thermal change.

To this end, we arrive at a energy and thermal model which relates server energy consumption to the overall thermal envelope, establishing an energy relationship between workload and overall system thermodynamics. Our model treats a single server blade as one closed black-box system, relying on the fact that measured energy input into the system can be a function of the work done by the system (in executing its computational tasks) and of residual thermal energy given off by the system during execution. It utilizes the hardware performance counters (PeCs) of the server for relating power consumption to its consequent thermal envelope, enabling dynamical control of the thermal footprint under given workload. With PeCs, the computational load of a server can be estimated using relevant PeC readings together with a set of operating system's performance metrics. The selection of PeCs and metrics is guided by the observation that the computational load in processors that use a Non-Uniform Memory Access (NUMA) architecture can be estimated by the amount of data transferred to/from memory and amongst processors over system buses. The result is that the underlying system is a dynamic system whose behavior can be described as a system of deterministic differential equations whose solution can be estimated via time-series approximation. Our energy consumption prediction treats observed readings of available PeCs in a discrete time series and resorts to statistical approximation to compensate for those model components that cannot be determined exactly.

Generally, a time series model observes past outcomes of a physical phenomenon to anticipate future values of that phenomenon. Many time series-based models of processor energy consumption have been considered [Rivoire 2008; Bhattacharjee and Martonosi 2009; Powell et al. 2009; Reich et al. 2010; Bircher and John 2011], with recent work extending such models into the thermal domain [Coskun et al. 2008]. Time series are typically handled in three broad classes of mechanisms: auto-regressive, integrated, and moving average models [Box et al. 1994]. Each of these classes assumes a *linear relationship* between the dependent variable(s) and previous data points. However, energy consumption, ambient temperature, processor die temperatures, and CPU utilization in computers can be affected by more than just past values of those measurements made in isolation from each other [Bertran et al. 2010; McCullough et al. 2011]. Our analysis of experimental measurements of key processor PeC readings and performance metrics reveals that the measured readings and metrics of a server over the time series do not possess linearity and are *chaotic in nature*.

This chaotic nature may result from the behavior of various server components, like DC-DC switching power converters commonly used by power circuitry in modern computers [Hamill et al. 1997; Tse and Di Bernardo 2002]. It thus leads to our development of Chaotic Attractor Predictor (CAPs) that model the dynamics of the underlying system of differential equations. Hence, our thermal model takes into account key thermal indicators (like ambient temperatures and die temperatures) and system performance metrics (like performance counters) for system energy consumption estimation within a given power and thermal envelope.

Servers based on the HyperTransport bus [HyperTransport Technology Consortium 2007] and the Intel QuickPath Links [Intel 2009] are chosen as representatives to validate our CAPs for estimating run-time power consumption and thermal prediction. CAPs takes into account key thermal indicators (like ambient temperatures and die temperatures) and system performance metrics (like performance counters) for system energy consumption estimation within a given power and thermal envelope. It captures the chaotic dynamics of a server system without complex and time-consuming corrective steps usually required by linear prediction to account for the effects of non-linear and chaotic behavior in the system, exhibiting polynomial time complexity.

Starting with CAPs as a predictive tool, effective scheduling methods can be constructed to dispatch jobs to confine server power consumption within a given power budget and thermal envelope while avoiding detrimental impact on thread execution performance. Our scheduling techniques for preventive thermal management minimizes server energy consumption by (1) for each logical processor, selecting the next thread to execute based upon an estimate of which thread will have the largest thermal impact in the next quantum, and (2) adjusting the load balance allocation of threads to available logical processors to migrate workload away from thermally overextended resources on the processor. As opposed to prior work on thermal scheduling [Gomaa et al. 2004; Choi et al. 2007; Yang et al. 2008; Sarood et al. 2011] that focused on bounding temperatures below critical thresholds, our work considers how we can schedule high utilization workloads so as to manage average temperatures via balancing of thermal load across logical processors.

The component of the operating system most aware of the existence of multiple cores is the thread scheduling algorithm. The scheduler is responsible for ensuring that all of the cores across all processors in a system is kept busy; the scheduler must be able to efficiently migrate the state of threads across processors regardless if that migration remains on a core or across processors. Schedulers in the current generation of operating systems aid in processor thermal management by assigning workloads to logical processors as tightly as possible to provide more opportunities for power management to shutdown unused resources. However, compute bound server workloads common to the data center fully utilize available system resources, rendering these schemes ineffectual. Our Thermally-Aware Scheduler (TAS) addresses this problem by attempting to thermally balance the system with as little impact on performance as possible.

Our TAS is demonstrated and evaluated by adding thermal-awareness to the existing scheduler in the FreeBSD operating system executed on Intel Xeon (Woodcrest) processors. Benchmarks from SPEC CPU2006 [Henning 2006] and PARSEC [Bienia 2011] suites which represent typical application server workloads are chosen to evaluate our TAS. The gathered results reveal that TAS achieves reduction in mean core on-die temperatures by up to 12.8°C (from 44.8°C down to 32.0°C) under PARSEC benchmarks (which have multi-threaded workloads) and by up to 3.3°C under mixes of SPEC CPU2006 benchmarks for concurrent execution on all four server cores, while experiencing only 1% to 4% performance degradation. Our TAS compares favorably with a recent energy-aware scheduling technique [Sarood et al. 2011], which gets core temperature reduction by up to 4°C (from 63°C down to 59°C) when four parallel scientific applications compatible to the PARSEC benchmarks were executed on a physical 4-core Intel Xeon 5520 processor (similar to our testbed processor).

Chapter 2

Background and Related Work

The thread scheduler of an OS kernel is responsible for placing threads in the dispatch queue, deciding which threads to run next on available logical cores, and managing thread migration to balance the workload amongst all cores. In addition, a scheduler must fulfill two major applications requirements: (1) making equal progress on all threads of a given application, and (2) exploiting as much hardware parallelism as possible [Hofmeyr et al. 2010]. Traditional server load balancing has various assumptions about workload behavior when making thread placement decisions. In particular, interactive workloads are assumed to involve independent tasks that remain quiet for extended periods. Server workloads, on the other hand, are assumed to contain large numbers of threads which are highly independent of each other and use synchronization objects to ensure mutual exclusion on small data items. Modern operating systems (such as Linux and Solaris) make load balancing more power-aware by placing workloads to cores as tightly as possible (inside fewest processors possible), thereby presenting more opportunities for power management software to shutdown unused resources [Sun Microsystems 2009; Crew et al. 2009; Xia et al. 2010].

The following sections provide in sequence, background and prior work pertinent to power modeling, thermal modeling, chaotic behavior of energy consumption, and scheduling with load balancing support,

2.1 Power Modeling

Used to guide power management mechanisms in server systems, power models can be classified into two broad categories: simulation-based and analytical models. Although simulation may provide details and breakdowns of energy consumption, it is usually done statically in an off-line manner, is slow, and does not scale well nor apply favorably to realistic applications and large data sets. For example, the HotSpot thermal simulator [Skadron et al. 2004] models thermal behavior (and, indirectly, power behavior) by building a network of thermal resistances and capacitances to account for heating and power dissipation on a circuit.Such models must accurately reflect empirical data from the real systems being simulated. Studies of the power characteristics of the SPEC CPU2000 and CPU2006 benchmarks suites executed on the AMD Athlon 64 processor [Mesa-Martinez et al. 2007] and thermal characteristics [Mesa-Martinez et al. 2010] show that simulation-based models (1) need long execution times to accurately predict power behavior and (2) have problems dealing with key thermal metrics. Mercury [Heath et al. 2006] attempts to address these issues by adopting a hybrid approach with an out-of-band solver that processes component observations collected from a monitor daemon on the target system. The solver in this system uses the collected information to simulate the thermal behavior in the system. The Mercury infrastructure was used to construct Freon and FreonEC for thermal and energy management of clusters. However, statistical analysis of methods similar to those used in Freon indicate significant issues in this case [Davis et al. 2011]. Therefore, simulation-based models are unsuitable when dynamic power and thermal optimization are concerned [Economou et al. 2006].

Analytical models, by contrast, rely on power measurements at the constituent components of a server via sampling its hardware and software performance metrics, i.e., processor PeCs (performance counters) and operating system performance metrics. PeCs are hardware registers that can be configured to count various micro-architectural events, such as branch mis-predictions and cache misses. Attempts have been made to reconcile analytic models by mapping program phases to events [Isci and Martonosi 2006]. Common techniques for associating PeCs and/or performance metrics with energy consumption adopt linear regression to map collected metrics to energy consumed during program execution [Contreras and Martonosi 2005; Economou et al. 2006; Isci and Martonosi 2003b].

Models have been built for the processor, storage devices, single systems, and groups of systems in data centers [Kadayif et al. 2001; Isci and Martonosi 2003b]. For example, memory manufacturers provide clear architectural documentation concerning the power consumption of their products [Micron, Inc. 2007]; this information is regularly used as the foundation for modeling DRAM power usage [Liu et al. 2008; Lewis et al. 2008]. Similarly, detailed thermal and power models of storage devices [Gurumurthi et al. 2005] have been proposed. However, those models considered only the maximum internal data rate, thereby inadequate for our purpose of run-time measurement. At the processor level, Bellosa [2003] and Contreras *et al.* [2005] created power models which linearly correlated power consumption with PeCs. Those models are simple and fast with low-overhead [Bircher and John 2007; Powell et al. 2009; Bircher and John 2011], but they are sensitive to benchmarks chosen for model calibration and often suffer from large worst-case errors, making them unsuitable for predictive purposes [McCullough et al. 2011]. Furthermore, they do not consider full-system power consumption.

The black-box approach, without detailed knowledge of underlying hardware, has been applied recently to full-system modeling [Bircher and John 2007; 2011] by using PeCs to estimate power consumption of I/O devices, in addition to the processor. In general, full-system models can be created using operating system CPU utilization [Fan et al. 2007] and similar metrics [Heath et al. 2005] in addition to PeCs. Such full-system models, like MANTIS [Economou et al. 2006; Rivoire 2008] and trickle-down power modeling [Bircher and John 2011], relate usage information to the power of the entire system rather than of its individual components. Each of those models requires one or more calibration phases to determine the contribution of each system component to overall power consumption. The accuracy and the portability of full system power models were assessed earlier [Rivoire et al. 2008], revealing that reasonable accuracy across machines and different workloads was achievable by considering both PeCs (performance counters) and operating system performance metrics, since they together may capture all components of system dynamic power consumption.

Auto-regressive (AR) techniques have been adopted popularly to construct power and temperature models for servers [Coskun et al. 2008; Powell et al. 2009; Bircher and John 2011], since they are simple and efficient. The linear AR models attempt to correlate PeCs to system behavior, with an average error of five to twenty-five percent (depending on architecture and benchmarks). Unfortunately, their maximum errors tend to be ill-conditioned, because of their *stationary* nature. In a stationary process, the probability distribution does not change with time, nor do the mean and the variance. Hence, AR and auto-regressive/moving average (ARMA) models are not suited for data that exhibits sudden bursts of large amplitudes at irregular time epochs, due to their assumptions of normality [Tong 1993]. Given workload dynamics of a server vary in time and its power profiles often diverge over time, effort has been made to accommodate this diverse behavior [Mesa-Martinez et al. 2007; Coskun et al. 2008] so as to permit continuing use of AR and ARMA models. This way, however, negatively impacts the performance advantage resulting from auto-regressive techniques, namely, their simplicity.

2.2 Thermal Modeling

Existing thermal models all required to fit targeted mathematical expressions based on time-series observations of temperatures during the course of executing various workloads. Expression coefficients were estimated by minimizing total least square errors between modeling results and actual temperature readings. After proper calibration, such a mathematical expression becomes a model for extrapolating future temperatures. Different modeling techniques have been devised. In particular, a model based on integer linear programming was adopted by an earlier task scheduler [Kursun and Cher 2009], which aimed to meet real-time deadlines while minimizing hot spots and spatial temperature differentials across the die. Meanwhile, dynamic thermal modeling includes Heat-and-Run [Gomaa et al. 2004], HybDTM [Ayoub et al. 2011] and ThreshHot [Yang et al. 2008]. Heat-and-Run distributes work among available cores until the DTM events arise, and it then migrates threads from the overheated cores to other cool cores. On the other hand, HybDTM enhances DTM with a thread migration strategy which lowers the priority of jobs executed on hot cores. Separately, ThreshHot employs an on-line temperature estimator to determine the proper order to schedule threads across cores, favoring those threads which cause the greatest temperature hikes while avoiding DTM events to occur. Schedulers based on prior thermal modeling all rely on readings of hardware performance counters and temperature sensors. They can be improved by analyzing on-die thermal variations to aid in system power and thermal management [Kursun and Cher 2009; Bailis et al. 2011; Murali et al. 2008].

However, preceding techniques are reactive to the temperature approaching the DTM threshold rather than trying to avoid reaching that temperature in the first place. A proactive solution with multi-tier prediction was suggested earlier [Ayoub et al. 2011], where a core level predictor was employed to convert temperature observations to operating frequency estimates while a control-theoretic based scheduler was followed at the socket level for process level scheduling. Separately, a scheduling policy for sorting the tasks in each core's run queue according to memory intensity was considered so as to schedule memory-bound tasks at slower frequencies [Merkel and Bellosa 2008a; Merkel et al. 2010]. Later, the process scheduling policy was modified in [Bellosa et al. 2003] to allocate time slices following (1) the contribution of each task to the system power consumption and (2) the current processor temperature. A similar approach made use of idle cycle injection to manage CPU time slice allocation, aiming to maintain a lower average temperature over time, as opposed to managing temperatures against a critical threshold. Meanwhile, Cool Loop [Choi et al. 2007] and Dimentrodon [Bailis et al. 2011] address a lack of heat slack by inserting additional cycles into the task scheduling to create thermal slack, naturally leading to performance degradation. A variation of preceding schemes relied on system level compiler support to insert a run-time profiling code into applications for providing hints on the thermal intensity of a task [Li 2008]. However, such an approach works ineffectively under many server cases where the slack in deadlines usually is unavailable.

2.3 Chaotic Behavior of Energy Consumption

Lately, the power consumption data collected during benchmark execution have signified their increasing non-linearity over time. For example, analysis of published results for the SPECpower_ssj2008 benchmark [Varsamopoulos et al. 2010; Hsu and Poole 2011] revealed maximum errors as large as 40% when modeling the benchmark results using linear two-point interpolation with the observation that the behavior of the power curves of this benchmark was neither linear nor convex. A variation of the MANTIS power model [Economou et al. 2006] was used to analyze the power curves of the SPEC CPU2006, PARSEC multi-threaded benchmark suite [Bienia 2011], and a set of synthetic system stress benchmarks [McCullough et al. 2011]. This work found for multi-core processors that the mean relative error doubled as compared to when the benchmark was restricted to executing on a single core. Furthermore, mean relative errors for individual subsystems such as the CPU were significantly higher, with an average error of 10-14% and the largest error as much as 150% for some workloads.

The cause of non-linear behavior in power data may be attributed to a number of factors. First, while the assumption of linearity depends upon homogeneous workloads, observations of the behavior of SPEC CPU benchmarks in both physical and virtual machine environments reveal that distributing execution tasks across multi-core processors is rarely uniform to have homogeneous workloads [Kansal et al. 2010]. Second, modern processors have no simple features which can be abstracted easily to permit linear models (for example, effects such as cache contention, processor performance optimizations, and hidden device states) [McCullough et al. 2011].

Dealing with workload dynamics without high computational complexity requires efficient estimation able to address inherent non-linearity in the time series. One approach follows local polynomial fitting via weighted least squares regression [Fan and Gijbels 1996] or its variation [Singh et al. 2009]. Another approach is to utilize a moving average technique for smoothing the data by computing an average of the a subset of data from the time series observations. For example, an exponentially-weighted moving average will weight the members of the subset in geometrically decreasing order so as give less weight to samples as they are further removed in time [NIST 2010]. A third approach to fitting non-linear curves with varying degrees of smoothness in different locations makes use of the derivatives of an approximating function with discontinuities. This can be accomplished by employing splines with discontinuities existing at points identified as knots. An example of this approach is Multivariate Adaptive Regression Splines (MARS) [Friedman 1991], which models the time series as a weighted sum of basis functions $B_i(x)$ and constant coefficients c_i:

$$f(x) = \sum_{i=1}^{k} c_i B_i(x), \tag{2.1}$$

where each of the basis functions can take the form of (1) a constant 1, with only one such term present, the intercept, (2) a hinge function in the form of $max(0, x - c)$ or $max(0, c - x)$, or (3) a product of two or more hinge functions. MARS is suitable for modeling power behavior because of their good balance between bias and variance. A model with low bias signifies that it is flexible enough to address non-linearity while sufficiently constrained to maintain low variance.

Chaotic systems are ubiquitous in nature, found in many different physical domains. It has been shown that the DC-DC power converters commonly used by power circuitry in modern computers demonstrates such behavior [Hamill et al. 1997; Tse and Di Bernardo 2002]. It follows then that any system that models such a physical system must reflect this behavior. One key feature of such a dynamic system is its sensitivity to initial conditions, whose small difference δx can result in the marked difference of $\delta x e^{\lambda t}$ after time t, for a certain $\lambda > 0$. This exponential separation signifies that even a small difference or error may lead to large divergence in the near future.

From a mathematical standpoint, chaos can be produced by both continuous and discrete systems. A continuous system expressed by a differential equation

$$\frac{dx(t)}{dt} = F(x(t)), \tag{2.2}$$

with at least three degrees of freedom $x(t) = [x_1(t), x_2(t), x_3(t), \ldots, x_m(t)]$, can be related to a companion discrete system of

$$x_{n+1} = f(x_n), \tag{2.3}$$

by considering Equation (2.3) as a projection of the flow for Equation (2.2) on a surface. Three conditions are necessary for such a system to show chaos: (1) the differential equation and companion discrete system are deterministic, (2) the functions of f and F must be nonlinear, and (3) the discrete system must have a positive Lyapunov exponent [Liu 2010].

2.4 Limitations of Existing Scheduling

Current power management software that utilizes DVFS techniques to address DTM events has been effective in addressing thermal emergencies [Donald and Martonosi 2006; Hanson et al. 2007], commonly implemented in modern server processors [AMD 2007; Intel 2009]. However, handling DTM through DVFS can be problematic due to issues with program phase behavior and contention for shared resources [Bircher and John 2008; Coskun et al. 2008], resulting directly from slow transitions between the active and the idle device states and also from inability to access resources associated with idle processors. When the power phase changes frequently, abundant thermal variations among cores within the processor occurs, leading to decreased reliability [Rosing et al. 2007; Coskun et al. 2008; Kursun and Cher 2009]. For example, the Intel XScale processor was reported to decrease its component MTTF (Mean Time to Failure) by 12% to 34%, depending upon the selected power management strategy [Rosing et al. 2007].

Work migration for energy savings and thermal management has a long history in the SMP, SMT, and CMP environments [Yao et al. 1995; Gomaa et al. 2004; Kumar et al. 2006; Yang et al. 2008]. A study of OS-level thermal migration using Linux on the IBM POWER5 processor [Choi et al. 2007] discovered that the rise and fall times of core temperatures vary in the order of hundreds of milliseconds. As most operating systems choose scheduler ticks to be of 10 ms or less, it often is impossible to react to thermal conditions before a critical state is reached. As a result, three improvement mechanisms for managing thermal states have been pursued: (1) core hopping for leveraging spatial heat slack, (2) task scheduling for leveraging temporal heat slack, and (3) SMT scheduling for leveraging temporal heat slack. In the presence of slack, each of those mechanisms may reduce core die temperatures by 3 to 5° C on an average, at the expense of 3% mean performance degradation [Choi et al. 2007; Ayoub and Rosing 2009]. However, in the absence of slack commonly found under heavy workloads in high-performance servers, the three mechanisms becomes ineffective, calling for suitable scheduling with thermal awareness proactively.

Chapter 3

System Modeling

In order to develop a deterministic continuous energy consumption model based on computational load of the system, we consider E_{dc}, the total DC power input to the system, at the output of the power supply. Most servers operate on the AC input, with efficiency of power conversion from AC to DC equal to 72 - 80% (depending on the system load [Ton et al. 2008]) and with the DC output delivered in the domains of +/-12V, +/-5V, and +/-3.3V [Server System Infrastructure Consortium 2004]. Typically, two 12 Vdc lines supply power to the processor's hard drive(s) and cooling fans in the system. The 5 Vdc and 3.3 Vdc lines are dedicated to supplying power to the support chips and peripherals on the board.

Energy delivered to a server system, $E_{dc} = E_{system}$, can be expressed as a sum of energy consumed by constituent sub-systems in the server. Generally, there are five sources of energy consumption in a server system:

E_{proc}: Energy consumed in the processor due to computation,

E_{mem}: Energy consumed by DRAM chips,

E_{hdd}: Energy consumed by the hard disk drive(s),

E_{board}: Energy consumed by peripherals in support of board the operations, including all devices in multiple voltage domains across the board (like chip-set chips, voltage regulators, bus control chips, connectors, interface devices, etc.),

E_{em}: Energy consumed by all electrical and electromechanical components in the server, including fans and other support components.

Total energy consumed by the system with a given computational workload can be expressed as:

$$E_{system} = E_{proc} + E_{mem} + E_{hdd} + E_{board} + E_{em}. \tag{3.1}$$

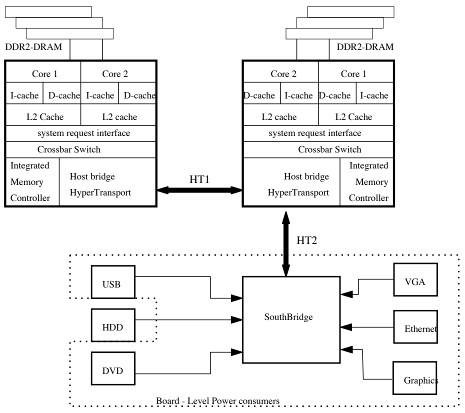


Figure 3.1: AMD Opteron architecture.

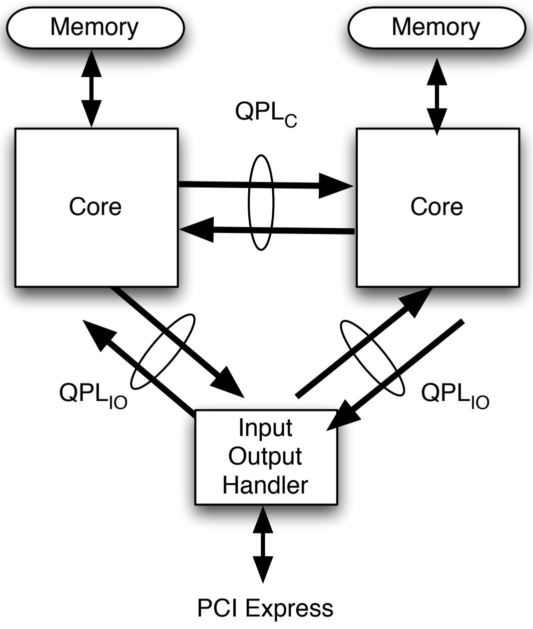


Figure 3.2: Intel Xeon (Nehalem) architecture.

Each of the above terms is explored in turn by following an energy conservation principle. In order to get a true measure of the computational load on the system, our approach snoops on bus transactions per unit time (indicated by PeC readings), measures the temperature changes (in die and ambient sensor readings), and records the speeds of cooling fans, in the course of job execution. The use of those PeCs and metric readings fits well to NUMA-based multi-core processors.

3.1 Processor energy consumption

Consider the AMD Operton architecture and the Intel Nehalem architecture, as depicted in Figures 3.1 and 3.2. The former is a NUMA-based processor (Figure 3.1), with Northbridge functionality incorporated in the processor core and each core responsible for local access to the memory connected to that Northbridge logic (shown in Figure 3.1 as “Integrated Memory Controller”). Processor cores on a single die are connected via a crossbar to the HyperTransport bus (i.e., HT1) between processors. A coherent bus protocol is used to ensure memory consistency between processor cores on each die. In addition, the master processor in the system is connected via a second HyperTransport bus (i.e., HT2) to the Southbridge device that manages connections to the outside world. A similar structure exists in the Intel Xeon Nehalem architecture. Unlike the AMD Operton, each Nehalem processor is connected to an Input-Output handler, which provides the Southbridge with connecting functions for off-chip resources.

It is observed that work done by any of the processors, as the heart of energy consumption in a server system, can be quantified in terms of bus transactions in and out of the processors. Traffic on the external buses provides a measure of how much data is processed by the processor. In the case of the NUMA architecture, this quantity can be measured by determining the amount of data transmitted over interconnect between processor cores (HyperTransport for AMD processors, QPI links for recent Intel processors). Our energy consumption model aims to treat each processor as a black box, whose energy consumption is a function of its work load (as manifested by core die temperatures measured at the system level by `ipmitool` through sensors on the path of the outgoing airflow from the processor). In practice, when estimating processor power consumption based on PeCs (performance counters), there are only a limited number of PeCs for tools, like `cpustat`, to track simultaneously.

For the AMD dual-core Operton architecture (shown in Figure 3.1), traffic on the HT buses is viewed as a representative of the processor workload, reflecting the amount of data being processed by a processor (i.e., its involved cores). The HT2 bus is non-coherent and connects one of the two processors to the Southbridge (whereas the Northbridge is included on the Opteron processor die). Thus, traffic on the HT2 bus reveals hard-disk and network transactions. This model scales by considering the effect of network traffic and disk I/O transactions. HT1 is a coherent bus between the two SMP processors and, as such, PeCs on HT1 provide accurate estimation on the processing load of cores executing jobs. Per-core die temperature readings are directly affected by the number of transactions over the HT1 bus. A similar observation holds for the QPL links present in the Intel Nehalem architecture, with traffic between its two cores reflected by transactions on QuickPath Links between the cores, denoted by QPL_C (see Figure 3.2).

Therefore, total processor power consumption at time t, $P_{proc}(t)$, is related to processor temperature readings and the estimated amount of data being processed at the time, and it can be expressed as a function of three metrics: die temperature readings for processors 0 and 1, and the number of bus transactions (i.e., traffic over HT1 for the AMD server and over QPL_C for the Intel server). We have processor energy consumption between times t_1 and t_2 as follows:

$$E_{proc} = \int_{t_1}^{t_2} (P_{proc}(t))\, dt. \tag{3.2}$$

3.2 DRAM energy consumption

Energy consumed by the DRAM banks is directly related to the number of DRAM read/write operations involved during the time interval of interest, and the number is reflected by (1) the last-level cache misses for all N constituent cores ($CM_i(t)$, $i = 1, 2, ..., N$) in the server when executing jobs and (2) the data amount due to disk accesses for OS support (like page tables, checkpoints, virtual environments) and due to performance improvement for peripheral devices (like buffered data for disks and optical devices, spooled printer pages). The data amount in (2) above, named $DB(t)$, is reflected by traffic over HT_2 (or QuickPath links between the two cores and the Input/Output handler, denoted by QPL_{IO}) for the AMD Opteron server (or the Intel Xeon server), as demonstrated in Figure 3.1 (or Figure 3.2). This is because network traffic does not exist in either testing server, which comprises only a single chip. Additional energy contributors include activation power and DRAM background power (due to leaking currents), represented by P_{ab}. As stated earlier [Micron, Inc. 2007], DRAM activation power and background power can be obtained from the DRAM documentation, and they together amount to 493 mW for one DRAM module in our AMD Opteron server. Consumed energy over the time

Table 3.1: Hitachi HDT725025VLA360 disk power parameters

Parameter	**Value**
Interface	Serial ATA
Capacity	250 GB
Rotational speed	7200 rpm
Power	
Spin up	5.25 W (max)
Random read, write	9.4 W (typical)
Silent read, write	7 W (typical)
Idle	5 W (typical)
Low RPM idle	2.3 W (typical for 4500 RPM)
Standby	0.8 W (typical)
Sleep	0.6 W (typical)

interval between t_1 and t_2 can be expressed by

$$E_{mem} = \int_{t_1}^{t_2} \left((\sum_{i=1}^{N} CM_i(t) + DB(t)) \times P_{DR} + P_{ab} \right) dt,$$

where P_{DR} refers to DRAM read/write power per unit data.

3.3 Hard disk energy consumption

Energy consumed by the hard disk(s) is approximated by using a combination of relevant PeCs and drive ratings. Building a full-system model of disk drive energy consumption is complicated by the distance (from the processor) and the high latency of the disk drive. Two earlier modeling approaches exist, with one analyzing drive performance and capacity based on the physical characteristics of the drive [Gurumurthi et al. 2005] and the other utilizing PeCs (which measure DMA accesses on the memory bus, uncacheable accesses, and interrupts) to estimate the value of E_{hdd} [Bircher and John 2011].

By contrast, our disk model is interested in the amount of information being transferred back and forth to the device. So, it is possible to calculate this value using information collected by the operating system and physical characteristics of the disk drive. Both our test servers use the Hitachi's SATA hard disk (whose specification and relevant power consumption figures are listed in Table 3.1). Based on the physical, electrical, and electromechanical parameters of a hard disk, one can construct its detailed power consumption model. However, a cruder but simpler model can be obtained from the typical power consumption data of hard disks and pertinent PeCs, including (1) the number of reads and writes per second to the disk and (2) the amount of data (in kilobytes) read from and written to the disk. Those PeCs can be measured by the tool of `iostat`, arriving at approximate disk power consumption, E_{hdd}, as:

$$\begin{aligned} E_{hdd} = & P_{spin-up} \times T_{su} + P_{read} \sum N_r \times T_r \\ & + P_{write} \sum N_w \times T_w + \sum P_{idle} \times T_{id} \end{aligned}$$

where $P_{spin-up}$ is the power required to spin-up the disk from 0 to full rotation, and T_{su} is the time required to achieve spin up, typically about 10 sec. P_{read} (or P_{write}) is the power consumed per kilobyte of data read from (or written to) the disk, whereas N_r (or N_w) is the number of kilobytes of data reads (or data writes) in time-slice T_r from (or to) the disk. The Hitachi disk achieves read operations at 1.5 Gbits/s, when consuming 530 mA current at +5V, thereby exhibiting approximately $13.3\mu W$/Kbyte. Similarly, it is found to consume $6.67\mu W$/Kbyte for write operations. The numbers of N_r and N_w can be obtained using `iostat` according to the chosen time slice.

There are two idle states for the disk: idle and unloaded idle (when disk read/write heads are unloaded). The time to go from the unloaded idle state to the idle state is usually less than 1 second (smaller than the resolution of `iostat`). Thus, a history match count in the `iostat` statistics with zero reads and writes signifies the periods in which the disk is idle, permitting us to compute idle energy consumption accordingly. `iostat` readings for the durations of switching to different disk power states may be obtained with a more in-depth analysis, which is not consider in this work.

3.4 Board energy consumption

The quantity of E_{board} represents energy consumption caused by the support chipsets, control logic, buses, signal links, etc., and it usually falls into the 3.3V and 5V power domains. Exhibiting little variation, E_{board} can be difficult to measure, as chipset power is drawn from multiple domains. Hence, earlier models [Kansal et al. 2010; Bircher and John 2011] treated E_{board} as a constant. In our case, however, this value is obtained using current probe-based measurements. The results are measured over an interval of interest, $t_{interval}$ and exclude the effects of processor, disks, fans, and optical devices. Note that introducing the current sensors (possibly taking up to 28 for a server [Server System Infrastructure Consortium 2004]) to the power lines on the board will provide instantaneous current readings for use in E_{board}.

Aggregated power consumption effects on the board may be captured using ambient temperature readings on the board. Such readings can be obtained using system management tools commonly found in server environments (such as IPMI), and they are included in the set of our PeCs for energy consumption estimation.

3.5 Electromechanical energy consumption

A server always involves electromechanical energy consumption, E_{em}, which is mainly due to the electromechanical functions related to system cooling. Multiple fans often exist in a server for cooling. Power drawn by the i^{th} fan at time t can be given by the following equation:

$$P^i_{fan}(t) = P_{base} \times \left(\frac{RPM^i_{fan}(t)}{RPM_{base}} \right)^3 \tag{3.3}$$

where P_{base} defines the base power consumption of the unloaded system when running only the base operating system and no application workload. The P_{base} value is obtained experimentally by measuring the current drawn on the +12V and +5V lines, using a current probe and an oscilloscope. There is a current surge at system start, which is neglected. Under nominal conditions, the +12V line draws approximately 2.2A to power both blower fans in the AMD testing server. Total electromechanical energy consumption over a given task execution period of T_p equals:

$$E_{em} = \int_0^{T_p} \left(\sum_{i=1}^{N} P^i_{fan}(t) \right) dt.$$

3.6 Extending the model for thermal behavior

An application is composed of p execution threads, each associated with a data set sized d_i, for $1 \le i \le p$, in a processor. The total data associated with A is the sum of the data associated with its component threads:

$$D_A = \sum_{i=1}^{p} d_i. \tag{3.4}$$

We assume that the activities take place (1) in a staging area, which contains both main and virtual memory operating space, and (2) in the processor with its cores and their associated caches. The execution time measurement includes computation time and the time to move application data from the staging area (on peripherals off the chip like DRAM and HDD) to a computation or operation area (on the chip, such as the caches and the cores).

Each application A with the problem size of D_A involves workload $W(\tau_i, d_i, t_i)$, for $1 \le i \le p$, comprising two components: (1) a count of the operations performed by the computational core, and (2) the count of communication operations required for transferring data, instructions, and data coherency and book-keeping functionality traffic. They are measured in terms of the number of bytes operated upon or transferred over, during the course of completing those instructions retired by the logical core for each thread p. Thus, energy consumed by executing application A with data set D_A can be expressed as:

$$E_A(A, D_A, t) = \sum_{i=1}^{p} W(\tau_i, d_i, t_i) \tag{3.5}$$

for $1 \le i \le p$, where τ_i is an execution thread involved in application A, d_i is the corresponding data set for that thread, and t_i is its thread execution time.

In order to relate system energy expenditure (upon application execution) to corresponding joule heating, we define the term of "Thermal Equivalent of Application" (TEA), as the electrical work converted to heat in running the application, leading to die temperature change and ambient temperature change of the system. Hence, the

TEA of application A is expressed by:

$$\Theta(A, D_A, T, t) = \frac{E_A(A, D_A, t)}{\lim\limits_{T \to T_{th}} J_e(D_A, \Psi_{cp})(T - T_{nominal})}, \tag{3.6}$$

where T_{th} denotes the threshold temperature at which a thermal emergency event will occur, and $T_{nominal}$ refers to the nominal temperature as reported by the system when it is in a quiescent state, i.e., only the operating system is running without any application execution. The term J_e is the “electrical equivalent of heat” for the chip, which reflects the *informational entropy* of the system due to processing the data bits during application execution and to the black body thermal properties of the chip packaging as well as the cooling mechanisms around the chip, as defined by the parameter Ψ_{cp}. The value of Ψ_{cp} is an ambient thermal characterization parameter provided by the hardware manufacturer to relate temperature to power for cooling purposes [Int 2006]. Thus, TEA is a dimensionless quantity, with both denominator and numerator expressing work done or energy consumed in finishing an application.

We combine these metrics into achieved performance per unit energy consumed by the chip:

$$C_\theta(A, D_A, T, t) = \frac{\Theta(A, D_A, t)}{E_A(A, D_A, t)} \tag{3.7}$$

This normalized quantity indicates the “cost” of executing an application on a given processor, with $E_A(A, D_A, t)$ obtained from energy consumption of individual physical components (processor, DRAM units, HDD, motherboard, and electrical/electromechanism) given by Equation (3.1).

Chapter 4

Effective Prediction

The current generation of server systems lacks (1) the complete set of measurement and monitoring capabilities and (2) data flow state capture mechanisms required in order to formulate the parameters of an exact analytical model. For example, the system board DC and AC power consumption cannot be easily split in measurements or analyses, due to the presence of large numbers of voltage/current domains, each with multiple components. Therefore, effective prediction on future power consumption based on past power consumption readings/measurements (obtained from PeCs and performance metrics) is critical.

In Equation (3.1), E_{system} signifies total energy consumed by the system for a given computational workload, equal to the sum of five terms: E_{proc}, E_{mem}, E_{hdd}, E_{board}, and E_{em}. Adopting Equation (3.1) for server energy consumption estimation, one needs to predict change in E_{system} over the time interval of $(t, t + \Delta t)$. Such prediction, following a time series to make observations of the server system, based on PeCs and performance metrics, can be approximated by

$$E_{system} = \hat{f}(E_{proc}, E_{mem}, E_{hdd}, E_{board}, E_{em}), \tag{4.1}$$

where the involved parameters correspond to the five server energy contributors modeled in Sections 3.1 to 3.5. Similar time series are created for the quantities defined $Theta(A, D_A, t)$ and $C_{theta}(A, D_A, T, t)$ for use in thermal prediction, with parameters also taken from the five server energy contributors defined in Sections 3.1 to 3.5.

4.1 Performance counters and metrics

In our prediction approach, fourteen (14) observable PeCs and accessible performance metrics (referred to as "measures" collectively for simplicity) are involved in the AMD server, as listed in Table 4.1. They are grouped into five clusters, depending on their relevance to the server energy contributors. More specifically, the top three measures are related to E_{proc}, named $MP_{proc}^{AMD} = [T_{C_0}, T_{C_1}, HT_1]$. The next five measures dictate E_{mem}, denoted by $MP_{mem}^{AMD} = [HT_2, CM_0, CM_1, CM_2, CM_3]$. Those CM_i measures, capturing the total L2 cache miss counts due to Core i, are registered at `PAPI_L2_TCM` (being OpenSolaris generic events equivalent to the matching event in the Linux PAPI performance counter library [London et al. 2001]) and mapped to the AMD performance counters at 0x7E (as defined in [AMD 2006]). The following two measures are pertinent to E_{hdd}, represented by $MP_{hdd}^{AMD} = [D_r, D_w]$, which refer to the total numbers of bytes in disk reads and disk writes, respectively, during a period of 5 seconds (in our experiments) for all I/O devices (which are limited to the disk only, since no network traffic nor optical disks exist in the two testing servers), as recorded by the system activity monitor. The next two measures are related to E_{board}, indicated by $MP_{board}^{AMD} = [T_{A_0}, T_{A_1}]$, which register the temperature readings of two board locations where temperature sensors are placed. Finally, the last two measures determine E_{em}, shown by $MP_{em}^{AMD} = [F_C, F_M]$, which provide speed information of the CPU cooling fan and the memory cooling fan. Collectively, each observation at time t includes the 14 measures of $MP^{AMD}(t) = \left[MP_{proc}^{AMD}, MP_{mem}^{AMD}, MP_{hdd}^{AMD}, MP_{board}^{AMD}, MP_{em}^{AMD}\right]^T$.

On the other hand, the Intel Nehalem server involves nineteen (19) measures, as listed in Table 4.2. Again, they are classified into five groups, each associated with one server energy contributor. Notice that QPL_C and QPL_{IO} are relevant to QuickPath Links (depicted in Figure 3.2), and they are associated with E_{proc} and E_{mem}, respectively. In practice, however, there is just one single PeC for holding aggregated QPL_C and QPL_{IO} together. Among those measures listed in Table 4.2, the top three are pertinent to E_{proc}, comprising MP_{proc}^{Intel}. The next three measures determine E_{mem}, forming MP_{mem}^{Intel}. Those two CM_i measures indicate the total L3 cache miss counts due to Core i, $i = 0$ or 1. The cache miss counts record the last-level cache (i.e., L3) misses for the Intel Xeon processor on which our testing Intel server is built. They are reflected by the OpenSolaris generic event, `PAPI_L3_TCM` (as detailed in

Table 4.1: PeCs and performance metrics for AMD Opteron server

Variable	**Measurement**
T_{C_0}	CPU0 Die Temp
T_{C_1}	CPU1 Die Temp
HT_1	HT1 Bus X-Actions
HT_2	HT2 Bus X-Actions
CM_0	Last-level Cache Misses due to Core0
CM_1	Last-level Cache Misses due to Core1
CM_2	Last-level Cache Misses due to Core2
CM_3	Last-level Cache Misses due to Core3
D_r	Disk bytes read
D_w	Disk bytes written
T_{A_0}	Ambient Temp0
T_{A_1}	Ambient Temp1
F_C	CPU Cooling Fan Speed
F_M	Memory Cooling Fan Speed

Table 4.2: PeCs and performance metrics for Intel Nehalem server

Variable	**Measurement**
T_{C_0}	CPU0 Die Temp
T_{C_1}	CPU1 Die Temp
QPL_C	Transactions on QPL between Cores
QPL_{IO}	Transactions on QPLs for IO Handler
CM_0	Last-level Cache Misses due to Core0
CM_1	Last-level Cache Misses due to Core1
D_r	Disk bytes read
D_w	Disk bytes written
T_{A_0}	Ambient Temp0
T_{A_1}	Ambient Temp1
T_{A_2}	Ambient Temp2
F_C	Memory Cooling Fan Speed
F_{M2a}	Memory Cooling Fan Speed 2a
F_{M2b}	Memory Cooling Fan Speed 2a
F_{M3a}	Memory Cooling Fan Speed 3a
F_{M3b}	Memory Cooling Fan Speed 3b
F_{M4a}	Memory Cooling Fan Speed 4a
F_{M4b}	Memory Cooling Fan Speed 4b
F_{M5a}	Memory Cooling Fan Speed 5a
F_{M5b}	Memory Cooling Fan Speed 5b

[Sun Microsystems 2008] and [Intel 2009]). The next two measures are related to E_{hdd} (and constitute MP_{hdd}^{Intel}), signifying the total numbers of bytes in disk reads and disk writes, respectively, during a period of 5 seconds. The subsequent three measures dictate E_{board}, obtained from 3 temperature sensors placed on the board for ambient temperature readings; they form MP_{board}^{Intel}. Finally, the last nine measures determine E_{em}, offering speed information of those nine memory cooling fans, to constitute MP_{em}^{Intel}. As a result, each observation for the Intel server at time t comprises the 19 measures of $MP^{Intel}(t) = \left[MP_{proc}^{Intel}, MP_{mem}^{Intel}, MP_{hdd}^{Intel}, MP_{board}^{Intel}, MP_{em}^{Intel}\right]^T$.

We use the measured PeCs listed in Table 4.2 to estimate the quantities of $\Theta(A, D_A, t)$ and $E_A(A, D_A, t)$ in Equation (3.7). Specifically, thread length is estimated in each time period by the PeC measure of IR, namely, the total number of instructions retired during that thread execution. The total data amount associated with thread execution is measured by summing up (1) the data bytes moved across the QuickPath links on the processor (reflected by QPL_C and QPL_{IO}), (2) last-level cache misses (CM_i, $1 \leq i \leq 4$), and (3) data read from or written to disks. as listed under the contributor of "application data set" in Table 4.2. System temperature is measured using the ambient temperatures reported by the system, as listed under the contributor of "system temperature."

4.2 Linear Auto-Regressive Models

A common prediction approach follows the linear auto-regressive (AR) combination of observation measures to predict the quantities in Equation (4.1) [Lewis et al. 2008]. It yields E_{system} by adding up $f_{co}(MP_{co})$ for all server energy contributors, with each f_{co} (due to Contributor co) being a linear summation of its constituent measures. We consider three regression-based prediction models: a linear AR(1) model [Box et al. 1994], a MARS model [Friedman 1991], and an EWMA model. Each model is an approximations to the dynamic system in Equation (3.1), following regressive combinations of five energy contributors to a server, as given in Equation (4.1) [Lewis et al. 2008].

Two methods were considered for consolidation: arithmetic mean (average) and geometric mean. Trial models were constructed using each method and a statistical analysis of variance was performed to determine which model generated the best fit to the collected data with a time interval $t = 5$ seconds. Note that the statistical coefficients need to be computed only once using some benchmarks, for a given server architecture. The coefficients obtained can then be provided through either the system firmware or the operating system kernel for use in the server executing any application.

Under linear auto-regression, energy consumed by the processor for the AMD server, E_{proc}^{AMD} as defined by Equation (3.2), is a *linear combination* of MP_{proc}^{AMD} measures (stated in Section 4.1) as [Lewis et al. 2008]:

$$E_{proc}^{AMD} \approx 0.49 * T_{C_0} + 0.50 * T_{C_1} + 0.01 * HT_1.$$

For the Intel server, its energy consumption by the processor, E_{proc}^{Intel}, is a function of MP_{proc}^{Intel} (detailed in Section 4.1), leading to its estimated energy as follows:

$$E_{proc}^{Intel} \approx 2.29 * T_{C_0} + 0.03 * T_{C_1} + 0.52 * QPL_C.$$

In a similar fashion, energy consumed by the memory subsystem in the AMD server, E_{mem}^{AMD}, is a function of MP_{mem}^{AMD}, yielding

$$E_{mem}^{AMD} \approx 0.01 * HT_2 + 0.003 * CM_0 + 0.003 * CM_1 + 0.014 * CM_2 + 0.01 * CM_3.$$

Energy consumption for the memory subsystem in the Intel server, E_{mem}^{Intel}, is a function of MP_{mem}^{Intel}, giving rise to

$$E_{mem}^{Intel} \approx 0.52 * QPL_{IO} + 0.35 * CM_0 + 0.31 * CM_1.$$

Energy consumed as a result of disk activities in the AMD server (or the Intel server) is a function of MP_{hdd}^{AMD} (or MP_{hdd}^{Intel}), arriving at

$$E_{hdd}^{AMD} \approx 0.014 * D_r + 0.007 * D_w$$

and

$$E_{hdd}^{Intel} \approx 0.01 * D_r + 0.01 * D_w.$$

Energy consumed by the board in the AMD server is a function of MP_{board}^{AMD}, whose components are added in a linearly weighted fashion to derive E_{board}^{AMD} (or E_{board}^{Intel}), as follows:

$$E_{board}^{AMD} \approx 0.101 + 0.81 * T_{A_0} + 0.62 * T_{A_1}$$

and

$$E_{board}^{Intel} \approx 2.53 + 0.03 * T_{A_0} + 0.01 * T_{A_1} + 0.01 * T_{A_2}.$$

Finally, energy consumed by electromechanical elements in the AMD server, E_{em}^{AMD}, is a linear function of MP_{em}^{AMD}, leading to

$$E_{em}^{AMD} \approx 0.001 * F_C + 0.001 * F_M.$$

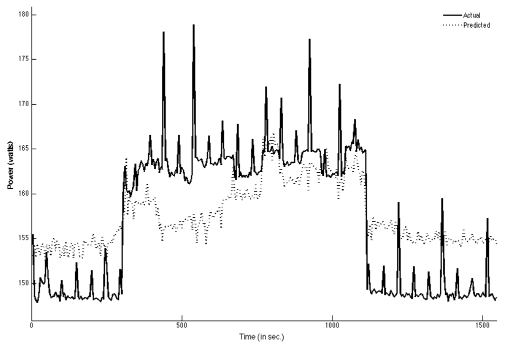
Similarly, energy consumption attributed to electromechanical elements in the Intel server, E_{em}^{Intel}, equals

$$\begin{aligned} E_{em}^{Intel} \approx 4.85 * F_C + 6.61 * F_{M2a} + 3.92 * F_{M2b} + 0.28 * F_{M3a} + 0.52 * F_{M3b} \\ + 0.01 * F_{M4a} + 0.01 * F_{M4b} + 0.78 * F_{M5a} + 0.61 * F_{M5b}. \end{aligned}$$

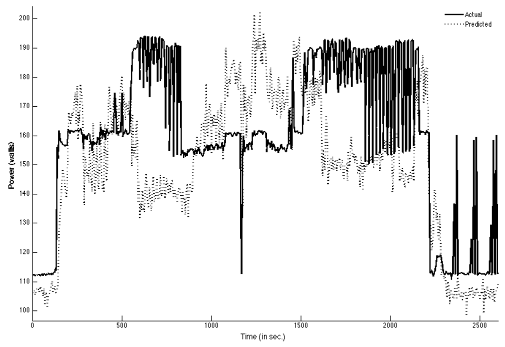
Total energy consumption for the AMD server (or the Intel server) under AR(1) equals the summation of above five consumption contributors [Lewis et al. 2008].

4.3 Shortcomings of Linear Models

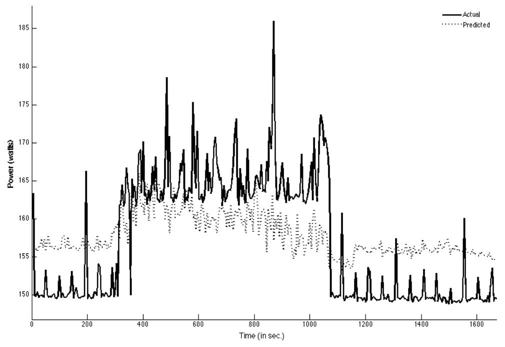
The models for AMD and Intel severs reveal the issues of adopting linear regression to obtain individual component contribution. Consider the Intel processor as an example. The coefficients for temperature sensors are significantly larger than those for the workload-related PeCs, with those coefficients apparently overbalancing the remaining model components. This fact is quite non-intuitive, as one would like to derive certain physical interpretation on each constant to understand the behavior of its associated model component. In addition, other processor models have negative coefficients for similar model components, making linear regression deemed unsuitable for such modeling [Bertran et al. 2010; McCullough et al. 2011].



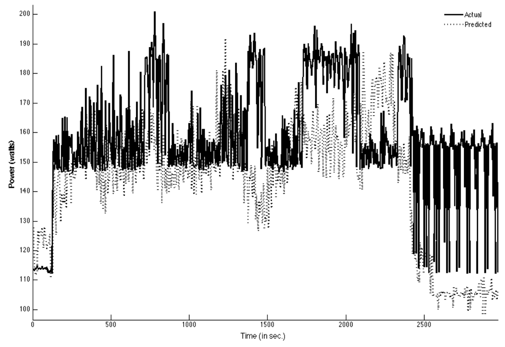
(a) Astar/AMD.



(b) Astar/Intel.



(c) Zeusmp/AMD.



(d) Zeusmp/Intel.

Figure 4.1: Actual power results versus AR(1) predicted results for AMD Opteron and Intel processors.

Consider the traces of actual power shown in Figure 4.1 SPEC CPU2006 `astar` and `zeusmp` benchmark as executed on an AMD Opteron or Intel Nehalem server. We saw indications of (1) periodic behavior and (2) large swings in the power draw throughout the course of the benchmark run. Similar scenarios were observed for other benchmarks on the AMD Opteron server and the Intel Nehalem server under this work. Linear regression-based prediction for power draw can mis-predict substantially (up to 44%, as indicated in Table 5.5). Thus, it is reasonably conjectured that *non-linear dynamics* do exist in server systems. Given large swings in power draw usually occur to a typical server and cannot be completely attributed to noise, more accurate prediction than linear auto-regression and MARS [Friedman 1991] is indispensable.

4.4 Chaotic prediction

The continuous systems expressed in Equation (3.1), (3.6), and (3.7) can be viewed as a multi-variate differential equations in the time domain (energy being power used in a time period). The time series approximation of a system solution can be viewed as a projection of the flow of Equation (3.1) onto a surface [Liu 2010]. The projection is defined in a way that the behavior (i.e., energy consumption) of the dynamic system is reflected in our discrete approximation (i.e., our time series measures).

We performed an analysis on the data collected from our test systems to determine if the behavior of our time series can be attributed to some form of chaotic behavior. A chaotic process is one which is highly sensitive to a set of initial conditions. Small differences in those initial conditions yield widely diverging outcomes in such chaotic systems. In order to determine whether a process is chaotic, we must be able to show that (1) it demonstrates high sensitivity to initial conditions and topological mixing, and (2) its periodic orbits are dense [Sprott 2003]. After analyzing our experimental data, we believe that the power consumption of a server demonstrates *chaotic behavior*.

In order to evaluate a server's sensitivity to initial conditions, we consider the

Table 4.3: Indications of chaotic behavior in power time series (AMD, Intel)

Benchmark	**Hurst Parameter** (H)	**Average Lyapunov Exponent**
bzip2	(0.96, 0.93)	(0.28, 0.35)
cactusadm	(0.95, 0.97)	(0.01, 0.04)
gromac	(0.94, 0.95)	(0.02, 0.03)
leslie3d	(0.93, 0.94)	(0.05, 0.11)
omnetpp	(0.96, 0.97)	(0.05, 0.06)
perlbench	(0.98, 0.95)	(0.06, 0.04)

Lyapunov exponents of the time series data observed while running those benchmarks described in the previous section. The Lyapunov exponent quantifies the sensitivity of a system such that a positive Lyapunov exponent indicates that the system is chaotic [Sprott 2003]. The average Lyapunov exponent can be calculated using $\lambda = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} ln|f'(X_n)|$.

We found a positive Lyapunov exponent when performing this calculation on our data set, ranging from 0.01 to 0.28 (or 0.03 to 0.35) on the AMD (or Intel) test server, as listed in Table 4.3, where each pair indicates the parameter value of the AMD server followed by that of the Intel server. Therefore, our data has met the first and the most significant criterion to qualify as a chaotic process.

The second indication of the chaotic behavior of the time series in Equation (4.1) is an estimate of the Hurst parameter H for the data sets collected in each benchmark. A real number in the range of $(0, 1)$, the Hurst parameter is in the exponents of the covariance equation for Fractional Brown motion (fBm) [Sprott 2003]. If the value of the Hurst parameter is greater than 0.5, an increment in the random process is positively correlated and long range dependence exists in the case of time series. In a chaotic system, a value of H approaching 1.0 indicates the presence of self-similarity in the system. As demonstrated in Table 4.3, the time series data collected in our experiments all have values of H close to 1.0, ranging from 0.93 to 0.98 (or 0.93 to 0.97) on the AMD (or Intel) test server.

In the thermal domain, a study of the time-series observations of core die temperatures for the Intel processor in our evaluation indicates the presence of chaotic behavior for the thermal domain as well as the power domain. As demonstrated in Table 4.4, the time series data collected in our experiments all have H values close to 1.0, ranging from 0.95 to 0.99 for the Intel server in our test environment.

Table 4.4: Chaotic behavior in core die temperature time series

	Hurst exp. (H)	Lyaponov exp. (λ)
Core 0	0.99	0.051
Core 1	0.98	0.019
Core 2	0.97	0.034
Core 3	0.95	0.040

From a predictive standpoint, the unpredictable deterministic behavior of chaotic time series means that it is difficult to build a predictor that takes a global parametric view of the data in the series. However, it is possible to generate a highly accurate short-term prediction by reconstructing the attractor in the phase space of the time series and applying a form of least square prediction to the resulting vector space [Itoh 1995; Li-yun Su 2010].

4.4.1 Chaotic Attractor Predictors

With the time series introduced in Equation (4.1), let y_t be the value of E_{system} at time t, r be the total number of PeCs and performance measures to provide metric readings, and X_t be the vector of those r metric readings at time t. According to Taken's Delay Embedding Theorem [Sprott 2003], there exists a function $\hat{f}(X_t)$ whose behavior in the phase space reflects the behavior of the attractors in the original time series values y_t. Consequently, for given $\hat{f}$, a known X_t reveals system energy consumption at time t, namely, y_t. If X_t can be predicted accurately for future time t (likely based on past metric readings), system energy consumption at future t can be estimated properly. To this end, it is necessary to find a means for approximating $\hat{f}$.

We introduce the concept of Chaotic Attractor Prediction (CAP) that defines $\hat{f}$ in terms of least squares regression of a multivariate local polynomial of degree r.

Multivariate local regression is a common non-parametric technique for time series approximations. With CAP, we extend this concept to predict the behavior of a chaotic time series by following the approximation method to take advantage of its polynomial time complexity while capturing the behavior dynamics of testing systems.

Let X be an observation (involving r measures of

$$MP(t + \Delta t) = [MP_{proc}, MP_{mem}, MP_{hdd}, MP_{board}, MP_{em}]^T$$

for a given server, as described earlier) at some future time $t + \Delta t$ and X_u be a prior observation (involving r metric readings of $MP(u)$) at time u for
$u = t - 1, t - 2, \ldots, t - p$. CAP localizes and addresses the possibility of noise in our observations through *kernel weighting.* This process starts with the standard multivariate normal density function of $K(x) = (2\pi)^{-\frac{m}{2}} exp(-\|X\|^2/2)$ (where $\|X\|$ is the norm of vector X) for smoothing out values of a local neighborhood, over which our CAP is defined. Let the bandwidth of β be a non-negative number and $K_\beta(X)$ equal $K(X/\beta)/\beta$ [Fan and Gijbels 1996]. The function of K_β serves as a *kernel* to smooth out noise in our original observations in a non-parametric manner. It has been shown that β determines the degree of smoothing produced by the kernel [Fan and Yao 2005]. Selection of a small β value does not adequately address issues of noise, while a too large β value results in excessive bias in the results and may hide important dynamics of the underlying function $\hat{f}$ [Turlach 1993]. A preferred choice for β can be computed by: $\beta = \left(\frac{4}{3p}\right)^{\frac{1}{5}} \sigma$, where σ is the standard deviation of observed values, estimated via the formula of $\bar{\sigma} = median(|x_i - \bar{\mu}|)/0.6745$, with $\bar{\mu}$ being the median of observed values [Bowman and Azzalini 1997].

An approximation for $\hat{f}$ is defined subsequently in terms of a locally weighted average [Box et al. 1994; Fan and Gijbels 1996] over the next n observations, based on the prior p observations of $X_{t-1}, \ldots, X_u, \ldots, X_{t-p}$ (each with r measures, namely,

$MP(u) = [MP_{proc}, MP_{mem}, MP_{hdd}, MP_{board}, MP_{em}]^T$, as described earlier):

$$\hat{f}(X) = \frac{\sum_{d=t}^{t+n-1} O_p * K_\beta(X_d - X)}{\sum_{d=t}^{t+n-1} K_\beta(X_d - X)}$$

with $O_p = (X_{t-1}, X_{t-2}, \ldots, X_{t-p})$.

The process can be improved by defining a local approximation via applying a truncated Taylor series expansion of $\hat{f}(X)$ for X at nearby x:

$$\hat{f}(X) = \hat{f}(x) + \hat{f}'(x)^T(X - x).$$

The coefficients of the polynomial $\hat{f}$ are then determined by minimizing

$$\sum_{d=t}^{t+n-1} \left(X_d - a - b^T(X_d - x)\right)^2 * K_\beta(X_d - x). \tag{4.2}$$

with respect to a and b, which are estimators to $\hat{f}(x)$ and $\hat{f}'(x)$, respectively. The predictor generated by solving Equation (4.2) can be explicitly written, according to [Box et al. 1994], as

$$\hat{f}(x) = \frac{1}{n} \sum_{d=t}^{t+n-1} (s_2 - s_1 * (x - X_d))^2 * K_\beta((x - X_d)/\beta) \tag{4.3}$$

with $s_i = \frac{1}{n} \sum_{d=t}^{t+n-1} (x - X_d)^i * K_\beta((x - X_d)/\beta)$, for $i = 1$ or 2.

4.4.2 CAP Creation

There are three steps involved in the process of creating a CAP predictor: (1) creating a training set for the process, (2) using the observations from the training set to find the appropriate delay embedding using Takens Theorem and then apply the nearest neighbors algorithm in the embedded set to identify the attractors, and (3) solving the resulting linear least squares problem that arises from applying Equation (4.2) to the attractors using the function expressed by Equation (4.3).

Table 4.5: SPEC CPU2006 benchmarks used for model calibration

Integer Benchmarks		
bzip2	C	Compression
mcf	C	Combinatorial Optimization
omnetpp	C++	Discrete Event Simulation
FP Benchmarks		
gromacs	C/F90	Biochemistry/Molecular Dynamics
cacstusADM	C/F90	Physics/General Relativity
leslie3d	F90	Fluid Dynamics
lbm	C	Fluid Dynamics

The training set for the predictor is constructed from a consolidated time series created by executing the SPEC CPU2006 [Henning 2006] benchmarks listed in Table 4.5 on target systems. The benchmarks were selected using two criteria: sufficient coverage of the functional units in the processor and reasonable applicability to the problem space. Components of the processor affect the thermal envelope in different ways [Kumar et al. 2008]. This issue is addressed by balancing the benchmark selection between integer and floating point benchmarks in the SPEC CPU2006 benchmark suite. Second, the benchmarks were selected from the suite based upon fit into the problem space. Each benchmark represents an application typical of the problems solved on high-performance application servers.

Each benchmark in the calibration set was executed with a sampling interval of $t = 5$ seconds. The observations from each time interval were consolidated on this time interval using two methods: arithmetic mean (average) and geometric mean. Trial models were constructed using each method and a statistical analysis of variance indicated that time series generated from the geometric mean produced the best fit to the collected data.

4.4.3 Time Complexity

The time complexity of creating a predictor is governed by the third step in the process. The task of reconstructing the state space by delay embedding is linear in time

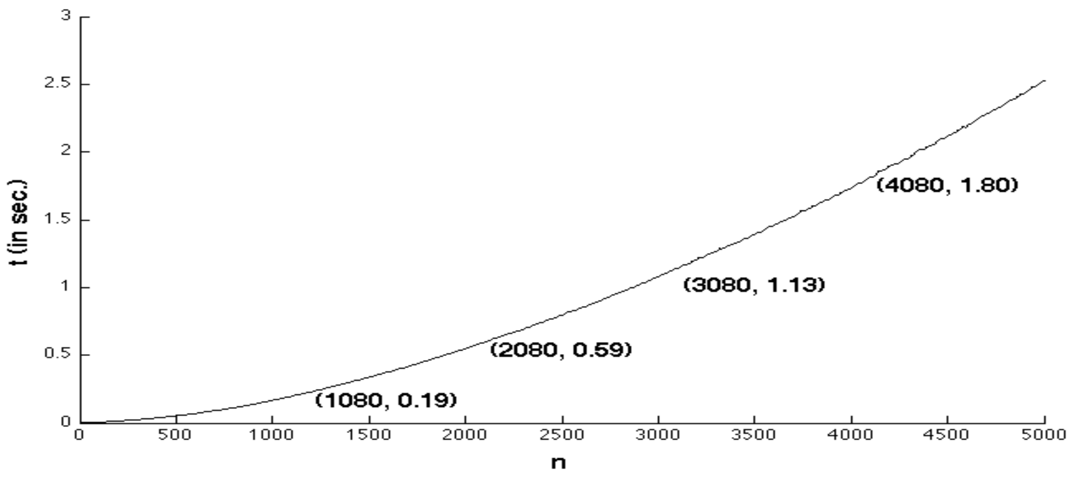


Figure 4.2: CAP time complexity versus no. of future observations.

as one must make up to d passes through the observations, under the embedding dimension of d. Thus, the time required is $O(dn)$, where n is the number of future observations. Then, it becomes a matter of applying a naive form of k^{th} nearest neighbors algorithm to identify the points in the attractors. This step involves finding the squared distance of all the points in the nearest analogs in the Takens set and then sorting the result to determine the d-nearest neighbors. This step takes $O(n \log n + n)$. The high cost of computing the linear least squares solution in the third step is avoided by using the explicit expression given in Equation (4.3). The time complexity of computing this expression can be shown to be $O(n * n)$, with $O(n)$ due to computing s_i, for $i = 1$ or 2. As a result, the time complexity for establishing a CAP predictor equals $O(n^2)$. It should be noted that the construction of a CAP predictor is done only once for a given server, irrespective of applications executed on the server. Such construction is based on past PeC observations (totally, p of them) to predict the future PeC readings. As p grows (with more past PeC observations involved), the time complexity of CAP increases linearly, as can be obtained in Equation (4.2). The actual computation time results under different n and p values for our CAP code implemented using MATLAB are provided in the next section.

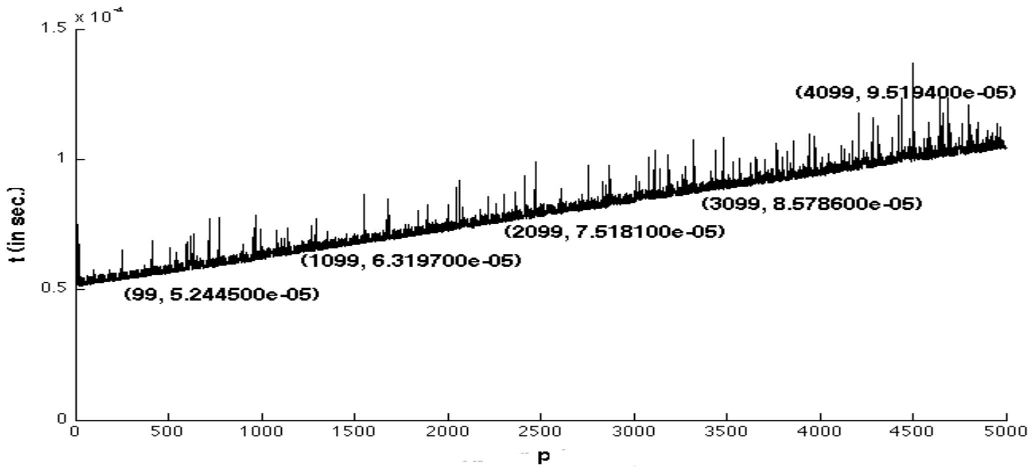


Figure 4.3: CAP time complexity versus no. of past observations.

4.5 Thermal Chaotic Attractor Predictors

We use the measured PeCs listed in Table 4.2 to estimate the quantities of $\Theta(A, D_A, t)$ and $E_A(A, D_A, t)$ in Equation (3.7). Specifically, thread length is estimated in each time period by the PeC measure of IR, namely, the total number of instructions retired during that thread execution. The total data amount associated with thread execution is measured by summing up (1) the data bytes moved across the QuickPath links on the processor (reflected by QPL_C and QPL_{IO}), (2) last-level cache misses (CM_i, $1 \leq i \leq 4$), and (3) data read from or written to disks. as listed under the contributor of "application data set" in Table 4.2. System temperature is measured using the ambient temperatures reported by the system, as listed under the contributor of "system temperature."

4.5.1 Thermal CAP Creation

We establish our thermal CAPs using two applications that are expected to yield the extreme thermal results for training execution: the idle scenario and the most stress scenario. An idle application referred to no application thread ready for execution and thus the testbed runs only the OS threads, whereas the most stress application leads to the highest processor utilization coupled with heaviest memory activities, achieved by launching multiple instances of the FreeBSD `cpuburn` stress testing codes for concurrent execution, one per core with symmetric multiple threads disabled. The `cpuburn` stress test code implements a single-threaded infinite loop with a sequence of integer and floating-point operations and large blocks of memory reads and writes to thermally stress the testbed system, driving it close to a thermal emergency. The use of these two extreme thermal scenarios is preferred over employing SPEC CPU2006 benchmarks for tCAP establishment, since any typical application execution will lies between the two extremes, as far as the coefficients of tCAP are concerned.

These two training scenarios run for 600 seconds, with PeCs sampled at an interval of $t = 5$ seconds for establishing tCAP. The procedure for establishing tCAP is the same as what was described earlier [Lewis et al. 2010], and it involved three steps. First, the training set is constructed by consolidating the observed PeCs into a time series using geometric means. In the second step, the Takens Embedding Theorem [Li-yun Su 2010] is applied to find delay embedding, with the nearest neighbors algorithm then employed to identify the set of attractors using the embedded set. Finally, our tCAP is established by solving the linear least squares problem resulted from fitting a multi-variate polynomial to the attractor set. The time complexity of establishing tCAP equals $O(n^2)$, where n is the number of sampled PeCs [Lewis et al. 2010]. Note that the establishment of the attractor set for tCAP is required only once for a processor type, irrespective of applications executed on the processor.

Chapter 5

Chaotic Attractor Predictor Evaluation

Experiments were carried out to evaluate the performance of the CAP power models built for approximating dynamic system solutions. The first experiment aimed to confirm the time complexity of CAP. Making use of MATLAB, our CAP code was executed on the two servers specified in Table 7.1. As the execution times on both servers provide the same trend, only those collected from the SUN Fire 2200 server (AMD Opteron) are demonstrated here. The code execution time results versus n (the number of future observations) is illustrated in Figure 4.2, where the result curve confirms CAP time complexity in $O(n^2)$. Separately, the CAP execution time as a function of p (the number of past observations) on the SUN Fire server is shown in Figure 4.3, where the time indeed is linear with respect to p, as claimed earlier in our time complexity subsection. In practice, a moderate p (of, say, 100) and a small n (of, say, 5) may be chosen under CAP for high accuracy and very low time complexity in real-time applications. The results of distant future (corresponding to a larger n) can be predicted by a step-wise process, with each step predicting the near future outcomes (using $n = 5$).

5.1 Evaluation environment

The operating system used in our test servers is OpenSolaris (namely, Solaris 11). Evaluation results are collected from the system baseboard controller using the IPMI interface via the OpenSolaris `ipmitool` utility. Processor performance counters are collected on a system-wide basis by means of `cpustat`, `iostat`, and `ipmitool` utilities in OpenSolaris. Of these, `iostat` and `ipmitool` are available across all UNIX-based operating systems commonly employed by data centers, while `cpustat` is an OpenSolaris-specific utility (which is being ported to Linux).

Four benchmarks from the SPEC CPU2006 benchmark suite were used for the

Table 5.1: Server configurations for evaluation

	Sun Fire 2200	**Dell PowerEdge R610**
CPU	2 AMD Opteron	2 Intel Xeon (Nehalem) 5500
CPU L2 cache	2x2MB	4MB
Memory	8GB	9GB
Internal disk	2060GB	500GB
Network	2x1000Mbps	1x1000Mbps
Video	On-board	NVIDA Quadro FX4600
Height	1 rack unit	1 rack unit

Table 5.2: SPEC CPU2006 benchmarks used for evaluation

Integer Benchmark		
Astar	C++	Path Finding
Gobmk	C	Artificial Intelligence: Go
FP Benchmarks		
Calculix	C++/F90	Structural Mechanics
Zeusmp	F90	Computational Fluid Dynamics

evaluation purpose, as listed in Table 5.2), and they are different from those employed earlier for CAP creation (as listed in Table V). It is noted that selection of benchmarks for both calibration and evaluation were selected to sufficiently exercise processor, cache, and memory again per the decision criteria in [Phansalkar et al. 2007], with the additional criterion of selecting workloads more common to server environments. As illustrated in Table 5.3 (per results from [Phansalkar et al. 2007]), the mix of instructions exercises the key components of processor, memory, and cache in the dynamic system which we model with CAP. In addition, the four benchmarks used in our evaluation represent the type of high-utilization workloads in server environments per typical industry use of the SPEC CPU2006 benchmark suite [Cisco Systems 2010].

The benchmarks were executed on the two servers specified in Table 7.1, with performance metrics gathered during the course of execution. Power consumed is measured by a WattsUP power meter [Electronic Educational Devices, Inc. 2006], connected between the AC Main and the server under test. The power meter measures the total and average wattage, voltage, and amperage over the run of a workload. The

Table 5.3: Branch and Memory Access Patters of Evaluation Benchmarks

Benchmark	Inst Count (Billions)	Branches	Loads	Stores
astar	1,200	15.57%	40.34%	13.75%
gobmk	1,603	19.51%	29.72%	15.25%
calculix	3,041	4.11%	40.14%	9.95%
zeusmp	1,566	4.05%	36.22%	11.98%

Table 5.4: Model errors for CAP (under $n = 5$, $p = 100$, $r = 14$), AR(1), MARS, and EWMA on AMD Opteron server

	CAP			**AR**		
Benchmark	**Avg Err %**	**Max Err %**	**RMSE**	**Avg Err %**	**Max Err %**	**RMSE**
Astar	0.9%	5.5%	0.72	3.1%	8.9%	2.26
Games	1.0%	6.8%	2.06	2.2%	9.3%	2.06
Gobmk	1.6%	5.9%	2.30	1.7%	9.0%	2.30
Zeusmp	1.0%	5.6%	2.14	2.8%	8.1%	2.14
	MARS			**EWMA**		
Benchmark	**Avg Err %**	**Max Err %**	**RMSE**	**Avg Err %**	**Max Err %**	**RMSE**
Astar	2.5%	9.3%	2.12	1.2%	7.9%	2.40
Games	3.0%	9.7%	2.44	1.1%	7.8%	1.42
Gobmk	3.0%	9.1%	2.36	1.0%	6.9%	2.30
Zeusmp	2.8%	7.9%	2.34	1.8%	9.2%	2.01

internal memory of the power meter is cleared at the start of each run and the measures collected during the runs are downloaded (after execution completion) from meter's internal memory into a spreadsheet. Current flow on different voltage domains in the server is measured using an Agilent MSO6014A oscilloscope, with one Agilent 1146A current probe per server power domain (12v, 5v, and 3.3v). This data was collected from the oscilloscope at the end of each benchmark execution on a server and then stored in a spreadsheet on the test host.

5.2 Results

While the number of measures per observation (r) is fixed for a given server in our evaluation (equal to 14 for the Sun Fire server and to 19 for Dell PowerEdge server), the CAP prediction time and accuracy depend on p (the number of past

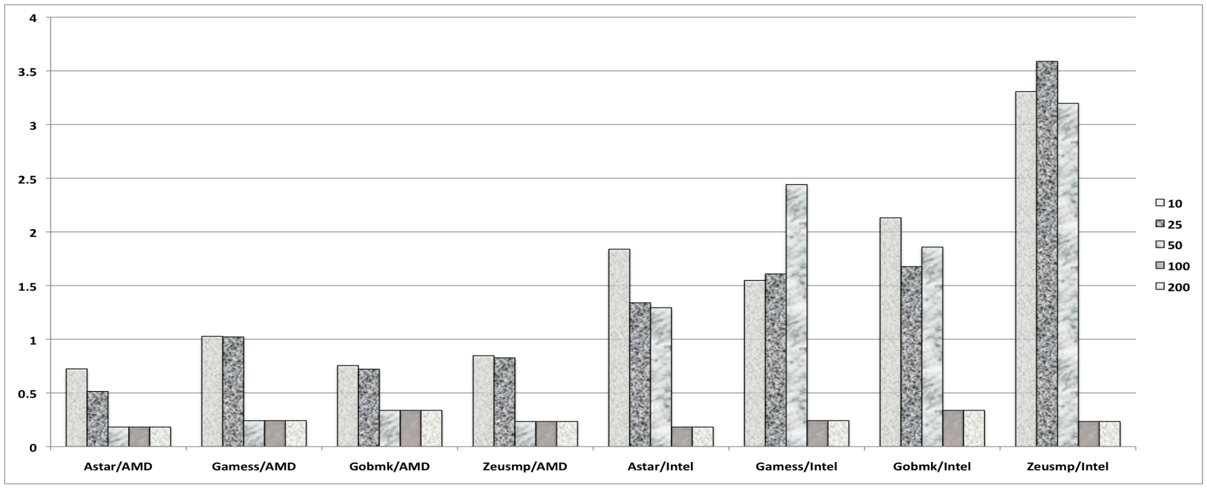
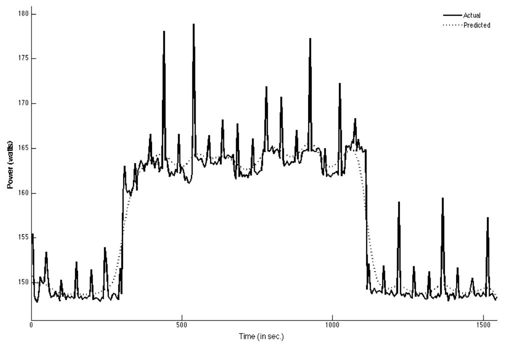


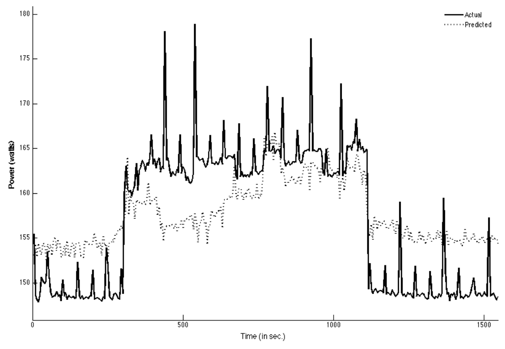
Figure 5.1: Root Mean Square Error (RMSE) for different values of p.

observations) and n (the number of future observations), as stated in Section 4.4.2. In our evaluation, the CAP prediction error rates of various benchmark codes for a range of p under a given n were gathered, as demonstrated in Figure 5.1, where n equals 5. It can be seen from the figure that the error rates are fairly small (and stay almost unchanged) when p is within 100 to 200, but they rise considerably when p drops to 50 or below. In subsequent figures, the prediction results of CAP include only those for $n = 5$ and $p = 100$. Each benchmark was executed to collect the first $p = 100$ points on the attractor, at which point the next n=5 points were used to compute the estimated power for the $t, t + 1, \ldots, t + 5$ energy estimates in each time cycle.

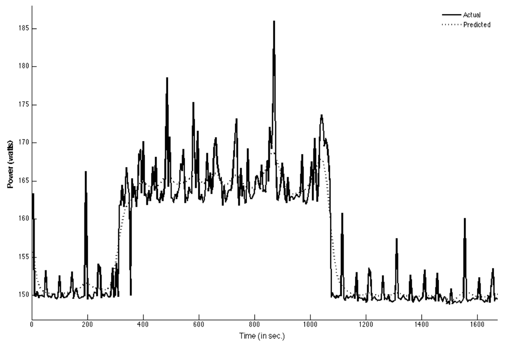
The predicted power consumption results of CAP during the execution of Astar and Zeusmp on a HyperTransport-based server are demonstrated in Figures 5.2(a) and 5.2(c). The predicted values are seen to track closely to the measured readings (obtained using the WattsUP power meter and indicated by solid curves), with the error rate ranging between 0.9% and 1.6%. For comparison, the predicted power consumption outcomes during the execution of same selected benchmarks under AR(1) are depicted in Figures 5.2(b) and 5.2(d). As expected, AR(1) exhibits poor outcomes over any given short execution window, with maximum errors ranging from 7.9% to 9.3%, despite that the prediction error over the whole execution period may be less.



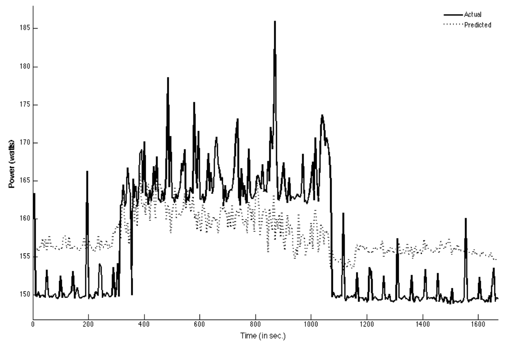
(a) Astar/CAP.



(b) Astar/AR(1)).

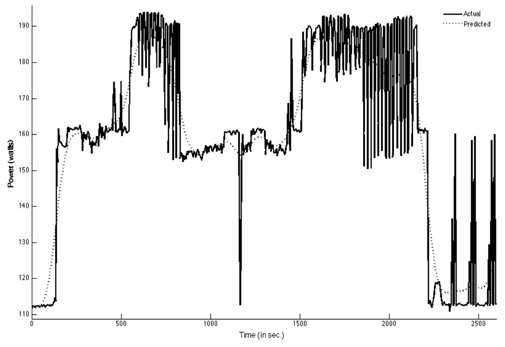


(c) Zeusmp/CAP.

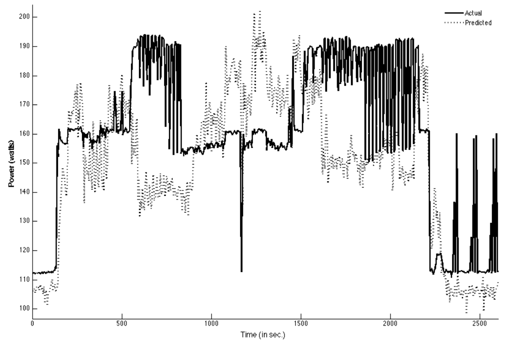


(d) Zeusmp/AR(1).

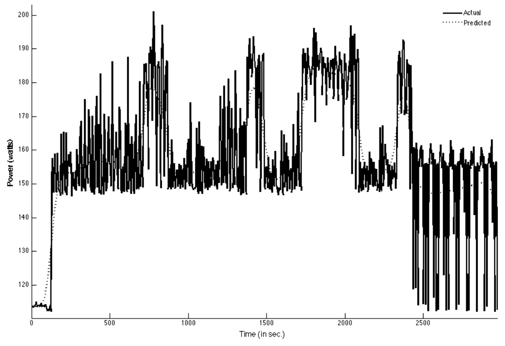
Figure 5.2: Actual power results versus predicted results for AMD Opteron.



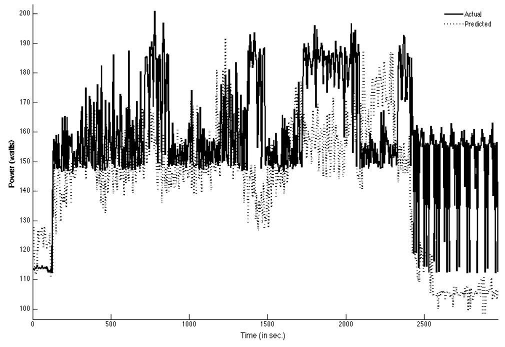
(a) Astar/CAP.



(b) Astar/AR(1)).



(c) Zeusmp/CAP.



(d) Zeusmp/AR(1).

Figure 5.3: Actual power results versus predicted results for an Intel Nehalem server.

CAP enjoys much better prediction behavior than its linear regressive counterpart.

The predicted power consumption results under CAP over the benchmark execution period for the QPL-based server (Dell PowerEdge) are demonstrated in Figures 5.3(a) and 5.3(c), where the actual power consumption amounts obtained by the WattsUP meter are shown by solid curves. Again, CAP is seen to exhibit impressive performance, tracking the actual amounts closely, with the error rate ranging between 1.0% and 3.3%. The root mean square errors for CAP remain within small values. In contrast, AR(1) suffers from poor prediction behavior, as can be discovered in Figures 5.3(b) and 5.3(d), where outcomes of same benchmarks executed on the Dell PowerEdge server are depicted. It yields the maximum error up to 20.8% (or 20.6%) for the Astar (or Zeusmp) benchmark.

Table 5.5: Model errors for CAP (under $n = 5$, $p = 100$, $r = 19$), AR, MARS, and EWMA predictors on Intel Nehalem server

	CAP			**AR**		
Benchmark	**Avg Err %**	**Max Err %**	**RMSE**	**Avg Err %**	**Max Err %**	**RMSE**
Astar	1.1%	20.8%	1.83	5.9%	28.5%	4.94
Games	1.0%	14.8%	1.54	5.6%	44.3%	5.54
Gobmk	1.0%	21.5%	2.13	5.3%	27.8%	4.83
Zeusmp	3.3%	20.6%	3.31	7.7%	31.8%	7.24
	MARS			**EWMA**		
Benchmark	**Avg Err %**	**Max Err %**	**RMSE**	**Avg Err %**	**Max Err %**	**RMSE**
Astar	5.4%	28.0%	4.97	3.7%	32.4%	2.98
Games	4.7%	33.0%	4.58	1.8%	27.3%	2.19
Gobmk	4.1%	27.9%	4.73	3.9%	28.4%	2.73
Zeusmp	11.6%	32.2%	8.91	5.0%	31.3%	2.81

5.3 Further discussion

Tables 5.4 (or 5.5) compares the errors of evaluation benchmarks for the server with the HyperTransport (or QPL) structure, under four different prediction mechanisms: CAP, AR, MARS, and EWMA. Large errors exhibited by AR, MARS, and EWMA overwhelm the advantages gained from their simplicity. The table results indicate the limitations entailed by using a linear technique, such as AR time series, to predict dynamic system behavior; similar issues exist for piecewise and moving average techniques, such as MARS and EWMA. Earlier attempts were made to address this issue by incorporating corrective mechanisms in combination with these predictors. An example attempt employed machine learning to monitor mis-prediction, with recalibration invoked when required [Coskun et al. 2008]. CAP eliminates the need for any corrective mechanism by directly addressing the system dynamics, thereby avoiding wide drifts in prediction experienced by other prediction techniques.

The model developed in this work is valid for any dual-core/dual-processor system using NUMA memory access connected in a point-to-point manner using the HyperTransport or the QPL structures. However, it can be scaled to quad-core dual processors based on those two structures. One would expect to see a slight difference or variation in power prediction due to a greater or less affect of die temperatures on the other performance measures. Under a dual-core quad-processor server, for example, additional regression variables would be incorporated in E_{proc}, giving rise to more performance measures (i.e., a larger r). Similarly, more PeCs related to cache misses would then be involved in E_{mem}. The solution approach of CAP remains exactly identical, except for a larger r in its prediction computation.

Chapter 6

A Thermal Aware Scheduler

The scheduler in an operating system is responsible for making two decisions in each time quantum: (1) thread scheduling, i.e., deciding the next thread to run on an available core and (2) load balancing, namely, distributing workload evenly across all cores, with existing implementations mostly focusing on performance. Our TAS (Thermal Aware Scheduler) incorporates a heuristic scheduling algorithm in a popular scheduler (i.e., ULE in the FreeBSD operating system) for thermal stress reduction on a multi-core processor while meeting the Single Process, Multiple Data requirements of equal execution progress and maximum parallelism exploitation.

6.1 Thermal Predictors

We enhance the existing FreeBSD operating system to maintain information required by the thermal estimator. Our design is based on the concept of Task Activity Vectors (TAVs) introduced earlier [Merkel and Bellosa 2008b], with a vector for each kernel thread to store the required history in order to make sound prediction. Generally, the more additional space is employed for history maintenance, the higher benefit our thermal scheduling gains.

The high-level system design of TAS is shown in Figure 6.1. A user-level daemon process collects required information to compute the time-series predictions for Θ and C_θ. Similar to the energy CAPs described in Section 4.4, tCAPs use $n = 5$ future observations and $p = 100$ past observations to compute predictions. Temperature readings are collected by this process from the digital temperature sensor associated with a core. Similarly, processor performance counters are gathered by the same process, with both sets of metrics used to generate estimates. Estimates are posted via a system call interface to a device driver that collects the data for use by the currently executing thread. The scheduler queries this driver via a kernel function call interface

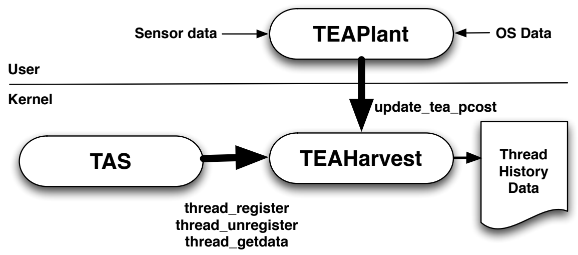


Figure 6.1: TEAPlant and TEAHarvest data collection.

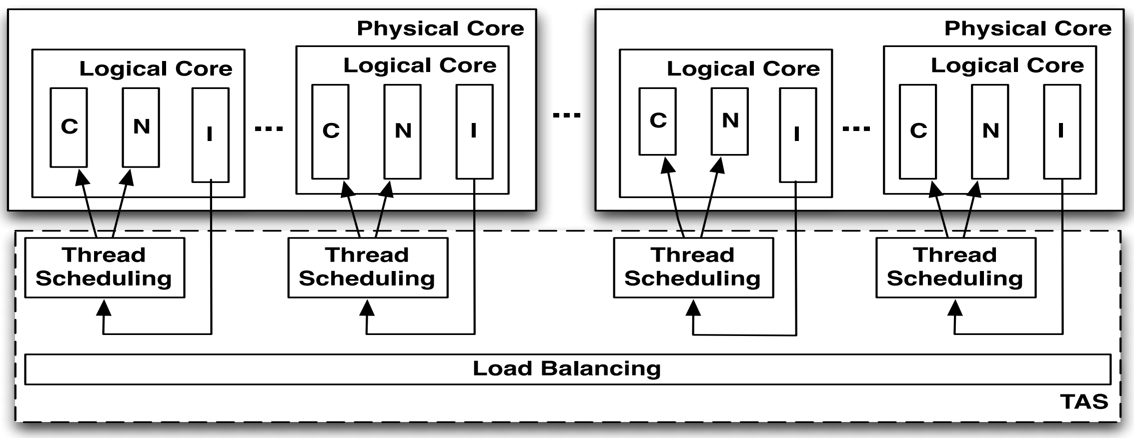


Figure 6.2: Internal structure of the TAS scheduler.

when making scheduling decisions to determine the Θ and C_θ estimates associated with a thread.

6.2 Thread Scheduling

The scheduler uses the cost predictor for C_θ to predict a thread's impact on core temperature and adjust the thread priority as required to prevent an increase in core temperature. Each core will have one process in a running state on each scheduling interval. All other threads will assigned by TAS to one of three queue structures maintained per logical core: an idle queue, a current queue, and the next queue (illustrated in Figure 6.2). All idle threads are added to the idle queue and threads from this queue are executed only when the other two queues are empty. The logical core executes all work on the current queue in priority order and then swaps its current queue and its next queue. For performance reasons, our implementation follows the convention used by the existing FreeBSD ULE scheduler, placing real-time and interrupt threads on the current queue. This is reasonable, given that real-time thread deadlines must be satisfied and interrupt service routines are typically short in length. Meanwhile, all other non-idle threads are assigned to either the current or next queue based upon computing an interactivity score, I, for each thread, as follows:

$$I = \begin{cases} S/(SL/RUN) & \text{if sleep time} \geq \text{run time} \\ (S/(SL/RUN)) + S & \text{if run time} < \text{sleep time} \end{cases} \tag{6.1}$$

where S is the scaling factor of the maximum interactivity score divided by two, and SL and RUN refer respectively to the cumulative sleep and run times for the thread. A thread with its I score smaller than a predefined threshold means that the thread had run for a very short duration during the last time window, indicating an interactive nature of the thread (e.g., an interactive thread). Hence, threads whose scores fall below a predefined threshold are considered interactive and are assigned to the current queue, with all other non-idle threads assigned to the next queue.

For our TAS, the interactivity score I is scaled by the predicted value of C_θ, normalized to a percentage value. It was shown previously [Zhou et al. 2010] that the greatest thermal benefit occurred when a scheduler favored the thread which moved the core temperature as close as possible to the DTM threshold without actually triggering a DTM event. TAS achieves a similar effect by scaling the interactivity of a thread by its normalized execution cost, thereby giving less "thermally costly" threads greater opportunities for execution so as to moderate the processor temperature. Penalizing more thermally costly threads reduces the opportunities for such threads to gain access to logical cores, presenting similar advantages of techniques which artificially inject slack into thread scheduling but without extra scheduling overhead incurred to those techniques. As shown in Figure 6.3, threads with higher "thermal cost" will be scheduled onto the next queue and avoid contributing to the thermal load of the processor. Threads on the next queue will be scheduled to run when the queues are switched and are guaranteed to run at least once every two queue switches, which

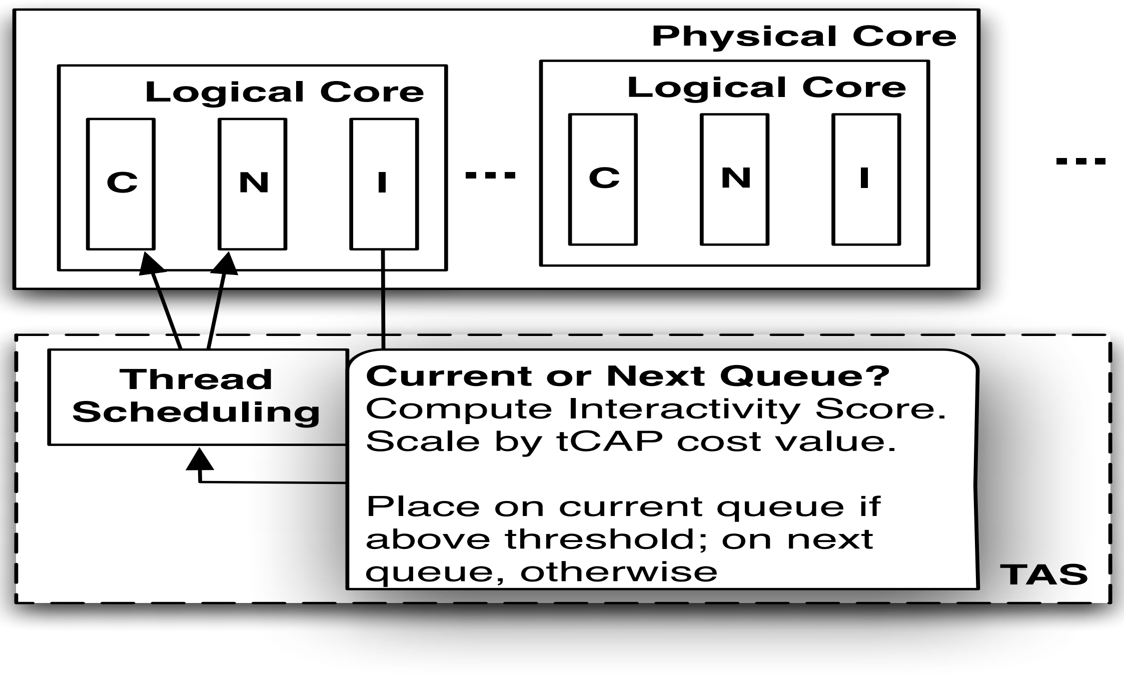

Physical Core
Logical Core
C
N
I
...
Logical Core
C
N
I
...
Thread Scheduling
Current or Next Queue?
Compute Interactivity Score.
Scale by tCAP cost value.
Place on current queue if above threshold; on next queue, otherwise
TAS

Figure 6.3: TAS thermal queue selection.

maintains fair sharing of the processor.

6.3 Load Balancing

Load balancing distributes workload evenly across the available logical cores, with the current implementation in the ULE scheduler mostly aiming to maximize performance. This work applies load balancing to minimize thermal stress while seeking best performance. As shown in Figure 6.4, TAS extends push migration by organizing system cores into "thermal clans" based upon the temperature and execution frequency. Specifically, a local core is assigned to one of the three thermal clans: Hot, Warm, and Cold, if its on-die temperature is respectively 90% or higher, between 75% and 90%, and below 75% of the DTM threshold temperature. Note that if a physical core supports α threads, α logical cores will result from the physical core, with their temperatures all equal to that of the physical core. In addition, logical cores are grouped into fast and slow clans according to the execution frequencies of their

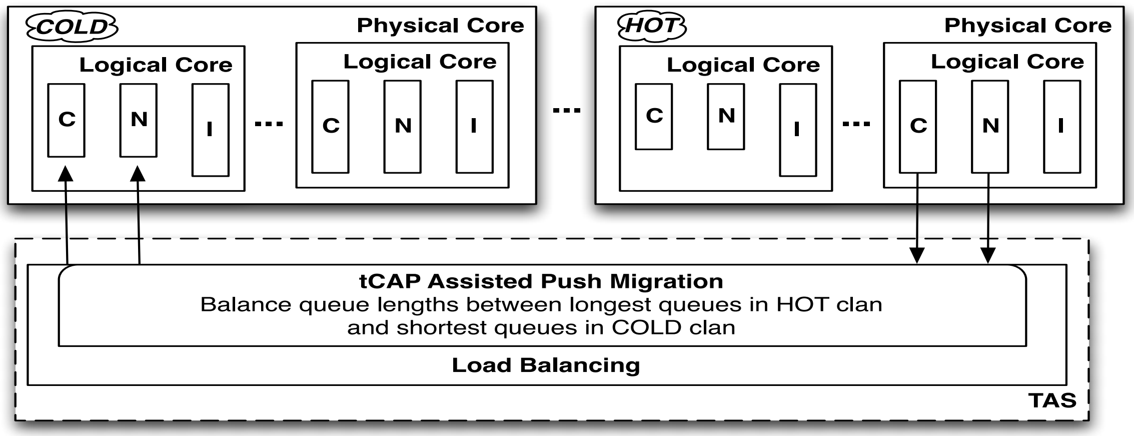

COLD
Physical Core
Logical Core
C
N
I
Logical Core
C
N
I
HOT
Physical Core
Logical Core
C
N
I
Logical Core
C
N
I
tCAP Assisted Push Migration
Balance queue lengths between longest queues in HOT clan
and shortest queues in COLD clan
Load Balancing
TAS

Figure 6.4: TAS thermal load balancing.

underlying physical cores. This two-level categorization allows TAS to manage work load distribution better from the thermal and performance prospectives, by migrating work away from hot units with negligible execution performance degradation.

For performance reasons, information about thermal clans of all logical cores is maintained by the TEAHarvest thermal predictor driver. The driver allocates threads to local cores for execution, according to thermal efficiency and cost estimates. TAS scheduler queries the driver to determine whether a core will move towards the DTM threshold temperature if a thread becomes ready to execute on one of its logical core. In this way, TAS predicts whether a thread moves its assigned core closer to a DTM event and adjusts the core's run-queue accordingly to prevent DTM occurrence.

On a periodic basis (i.e., once in every 500 ms), the TAS scheduler executes the algorithm listed in Figure 6.5 to balance the workload among logical cores, giving sufficient time for overtaxed resources to thermally recover. For each thread in the run queue, TAS uses the tCAP to estimate the value of Θ for each thread in the queue and predict the resulting change in temperature if this thread were to execute. It then moves the thread with the greatest temperature impact to the least loaded logical core in the "Cold" clan, as illustrated in Figure 6.4. This way results in workloads being moved away from thermally stressed logical cores, while maintaining execution performance.

```
BEGIN
  Determine if the logical core
    is in the Hot, Warm, or Cold clan.
  For each thread in the run queue
    BEGIN
      Use tCAP to estimate resulting
        temperature change if the thread
        is executed.
    END
  Determine least loaded core in the
      Cold clan.
  Migrate the thread with worst impact
      on temperature to logical core in
      the Cold clan most suitable from
      the speed standpoint.
END
```

Figure 6.5: Pseudo code of TAS balancing algorithm.

Execution performance is maintained by taking note of the location of this logical core in the processor group topology. The scheduler inventories the processor when it starts execution and creates a processor group structure that records the location of each logical core n the hierarchy of physical processor sockets and physical core [McKusick and Neville-Neil 2004b]. The decision of where to migrate the thread is made based upon which logical cores in the Cold clan are located in the topology, with preference given to logical cores that are sharing last-level caches on the same processor. For the example in Figure 6.4, the TAS would look to choose a processor from the Cold clan by selecting (1) a thread on a logical core with whom it shares a last-level cache, (2) a core on the same processor, and, finally (3) to a core on the other processor. Thus, TAS respects thread cache affinity and minimizes the performance impact of migration of threads between logical cores.

Periodically, the system reads the temperatures of all logical cores and, if required, also moves a logical cores to a different thermal clan. It should be noted that the time required for a logical core to recover from a thermal event is significantly longer than the interval used for thread scheduling [Choi et al. 2007]. This allows our

TAS to use a much larger interval (2 seconds) between scans (across core temperatures) to determine if any logical core must be assigned to a new thermal clan for better scheduling outcomes.

Chapter 7

Thermal Aware Scheduler Evaluation

We have evaluated our TAS (Thermal Aware Scheduler) using the FreeBSD operating system run on a commodity server. Our TAS implementation modified the existing push migration in FreeBSD's ULE scheduler [Roberson 2003; McKusick and Neville-Neil 2004a; 2004b] to take into account both thermal behavior and system performance, as elaborated in Chapter 6. PeC data were collected at the user level through standard tools provided for the data collection purpose by FreeBSD (i.e., the `coretemp` and `hwpmc` kernel extensions). They were collected and collated by a FreeBSD kernel extension, made available to the operating system scheduler for query when making scheduling decisions (Figure 6.1)

The processor thermal model was calibrated by measuring behavior outcomes of the testbed with TAS at idle and under high load stress using common utilities from the FreeBSD regression test suites and software collections. The CPU-related behavior of our scheduler was then characterized using integer and floating point benchmarks from the SPEC CPU2006 [Henning 2006] benchmark suite. Benchmarks from the Princeton Application Repository for Shared-Memory Computers (PARSEC) suite [Bienia et al. 2008] were then evaluated on the testbed to assess TAS in terms of key metrics of interest (i.e., die temperature and benchmark run time) under high parallelism in the thread level.

7.1 Experiment Setup

Experimental evaluation was conducted on our testbed running TAS, with its hardware specified in Table 7.1. The key metrics of interest were gathered during the course of application execution. Power consumed was measured by a WattsUP power meter [Electronic Educational Devices, Inc. 2006], connected between the AC Main and the server testbed. The power meter measured the total and average wattage, voltage,

Table 7.1: Processor used in evaluation

	Dell Precision 490
CPU	Intel Xeon 5300 (Woodcrest)
CPU L2 cache	4MB
Memory	8GB, DDR2 667Mhz with ECC
Internal disk	500GB
Network	1x1000Mbps
Video	NVIDA Quadro FX3400

Table 7.2: Branch and Memory Access Patters of Evaluation Benchmarks

Benchmark	**Inst Count (Billions)**	**Branches**	**Loads**	**Stores**
namd	2.483	4.28%	35.45%	8.83%
hmmer	3,363	7.08%	47.36%	17.68%
mcf	327	21.17%	37.99%	10.55%
milc	937	1.51%	40.15%	12.98%

and amperage over the run of a workload. The internal memory of the power meter was cleared at the start of each run and the measures collected during the runs were downloaded (after execution completion) from the meter's internal memory into a spreadsheet.

7.2 SPEC Benchmark Results and Discussion

Benchmarks from the SPEC CPU2006 suite [Henning 2006] were used to evaluate the thermal behavior of CPU-bound workloads executed on our testbed with the TAS scheduler. The benchmarks were selected to represent real life applications under various levels of thermal stress. It has been shown previously [Choi et al. 2007; Cher and Kursun 2011] that significant core-to-core and functional unit-to-functional unit thermal variations occur on modern processors, with the result that different workload characteristics have strong influence on the power and thermal behavior of the processor [Jiménez et al. 2010]. We thus consider mixing integer and floating-point SPEC CPU2006 benchmarks for concurrent execution on the four cores of our testbed processor.

Three workload mixes are considered, and they span a range of cases, including

Table 7.3: Temperature and runtime results under benchmark mixes

Workload	**Core die temperature reduction**				**Runtime**
Mix	**Core 0**	**Core 1**	**Core 2**	**Core 3**	**increase**
A	2.8°C	1.0°C	1.7°C	2.0°C	2.1%
B	1.0°C	0.8°C	2.0°C	1.1°C	2.9%
C	3.1°C	3.3°C	3.0°C	2.9°C	2.5%

high-IPC/low-IPC, hot/cold thermal profiles [Kursun 2008], and integer/floating-point combinations [Phansalkar et al. 2007] for better evaluating our scheduler under diverse workload variations. As illustrated in Table 7.2, the mix of instructions in the selected benchmarks benchmark exercises the key components of the processor, memory, and cache to create workloads with varying levels of thermal stress. The three workload mixes are as follows: Workload Mix A = {`namd`, `namd`, `hmmer`, `mcf`}, Workload Mix B = {`milc`, `milc`, `namd`, `hmmer`} and Workload Mix C = {`milc`, `mcf`, `mcf`, `hmmer`}. For example, Workload Mix B involved three copies of floating-point benchmarks (i.e., `milc`, `milc`, and `namd`) and one integer benchmark, `hmmer`. The memory intensive benchmarks (for instance, the `milc` benchmark) tend to consume more power when executed as opposed to other benchmarks in the suite while benchmarks with higher IPC (the `mcf` and `hmmer` benchmarks) tends to increase core on-die temperatures.

It was observed that as execution progressed, the threads of those benchmarks again tended to pin themselves to particular cores, depending upon cache and memory utilization of other concurrent threads. Core temperatures are reduced under TAS in comparison to those under ULE, with the reduction amounts listed in Table 7.3 for the three workload mixes. Under Workload Mix C, for example, TAS lowers core temperatures by at least 2.9°C and by as much as 3.3°C. The benchmark run times (which reflect performance degradation) increase negligibly always, ranging from 2.1% to 2.9%.

Involving both thermal-aware scheduling and load balancing, TAS compares favorably with earlier workload migration-oriented methods, with comparable reductions in average core die temperatures and performance impact for similar

Table 7.4: PARSEC benchmarks used in evaluation

bodytrack	Computer vision image tracking application
canneal	Simulated annealing chip routing cost computation
facesim	Physical simulation of facial behavior
ferret	Content-based similarity search
freqmine	Simulate FP-growth method for Frequent Itemset Mining
streamcluster	Online clustering algorithm for data mining
swaptions	Simulated pricing of portfolio options

workloads and processors. Specifically, Variation-Aware Core Hopping [Kursun and Cher 2009] examined both thread migration, by using off-line profiling of processor thermal variation to guide assignment of threads to cores, and thread selection, by using the profiling information to guide scheduling decisions in an enhanced Linux scheduler. This technique was demonstrated to lower core temperatures ranging from 2.0°C to 5.5°C with pairs of SPEC CPU2006 benchmarks executed an IBM POWER5 processor with minimal overhead in the decision process. However, their implementation did not performance impact of logical cores sharing resources and the related migration costs entailed by not considering cache affinity in scheduling decisions. Meanwhile, Predict-and-Act [Ayoub et al. 2011] employs a hybrid reactive/proactive migration policy that controls system fan speeds while scheduling threads at the core level. This hybrid method was shown to reduce on-die core temperatures by 3°C to 4°C on a "simulated" pair of quad-core Intel Xeon processors (rather than a real implementation) when executing mixes of SPEC2006 benchmarks similar to those in our study.

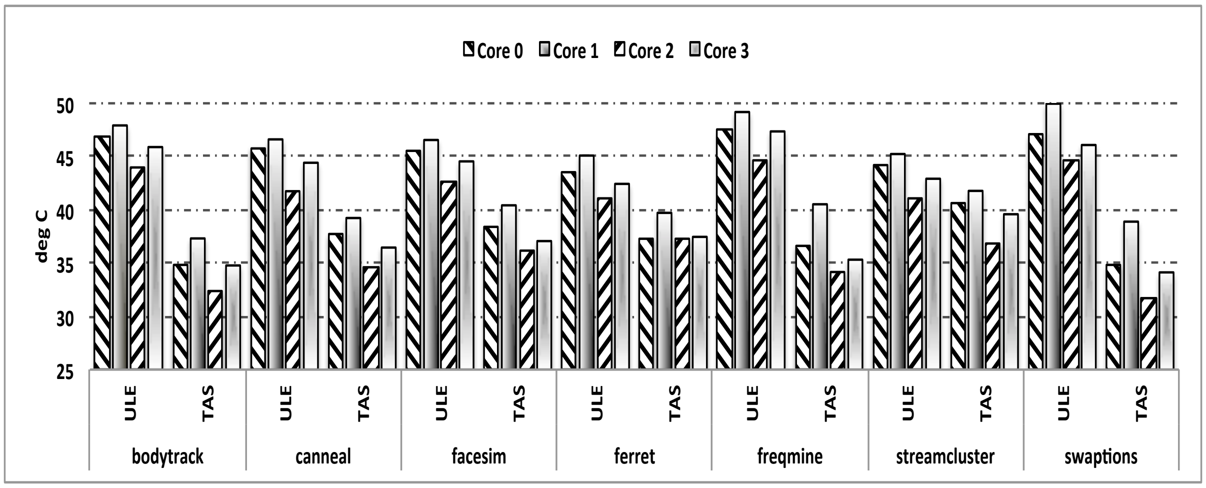


Figure 7.1: Comparison of PARSEC benchmark average core die temperatures under ULE and TAS schedulers.

7.3 PARSEC Benchmark Results and Discussion

Selected benchmarks from the PARSEC [Bienia et al. 2008] suite, described in Table 7.4, were used to evaluate our TAS as well. They were compiled with the POSIX `pthreads` library and executed using the PARSEC `native` input sets. Being parallel workloads selected from the fields of computer vision, computational finance, enterprise servers, and animation physics (see Table 7.4), PARSEC benchmarks better represent real life applications than SPEC CPU2006 and can benefit markedly from scheduling their abundant independent threads freely based on thermal prediction.

Three behavior metrics of interest under TAS are gathered: (1) the average core on-die temperature, (2) the benchmark run time, and (3) mean system power dissipation. The outcomes of average core on-die temperatures upon executing those seven PARSEC benchmarks are depicted in Figure 7.1. As can be seen from the figure, the core on-die temperatures range from 32°C (for Benchmark `swaption`) to 42°C (for Benchmark `streamcluster`) under TAS, with `swaption` and `bodytrack` (or `streamcluster`) experiencing the lowest (or highest) mean temperature across the four cores. Temperature outcomes for the same benchmarks under the classical ULE scheduler are also included in Figure 7.1 for comparison. It is found that temperature reduction amounts are larger for benchmarks with smaller working sets (like `bodytrack` and `swaptions`). This is because such a benchmark has a lower cache requirement [Bienia 2011] and thereby lets TAS schedule threads more freely according to thermal prediction with negligible performance degradation, resulting in better temperature reduction. Consequently, TAS enjoys temperature reduction by more than 12.8°C (from 44.8°C down to 32.2°C) for the `swaptions` benchmark.

Benchmarks with streaming functions, such as `facesim`, `freqmine`, and `streamcluster`, all have large working sets, which hinder TAS from scheduling threads freely based on thermal prediction, since doing so degrades performance considerably. Hence, those benchmarks tend to yield less temperature reduction under TAS, with the average core on-die temperature lowered by 3-6°C.

Execution performance (measured in terms of the benchmark runtime) under TAS is depicted in Figure 7.2. When compared with the runtime results under ULE included in the figure, it can be observed that TAS leads to negligible performance degradation, by no more than 3.3% for all benchmarks examined except `streamcluster`. Considerable performance degradation is seen for Benchmark `streamcluster`, however, likely due to its high computational intensity as the data set grows with rising dimensionality and thereby calling for more synchronization required. As a result, TAS experiences noticeable performance degradation when scheduling threads based on thermal prediction.

Mean system power dissipation for each of the PARSEC benchmarks run on our testbed under both schedulers is shown in Figure 7.3. When compared with ULE, TAS is observed to reduce power dissipation markedly, from 14W (for the `ferret` benchmark) to 19W (for `swaptions`), due to their relatively small working sets. On the other hand, less reduction in power dissipation is found for benchmarks with large working sets, lowering power by the range of 3W (for `streamcluster` to 10W (for `facesim`). The total energy consumption for each of the PARSEC benchmarks run on our testbed is shown in Figure 7.4. In comparison with ULE, TAS is observed to reduce energy consumption between 2.8kJ (for `ferret`) to 2.9kJ (for `swaptions`) for

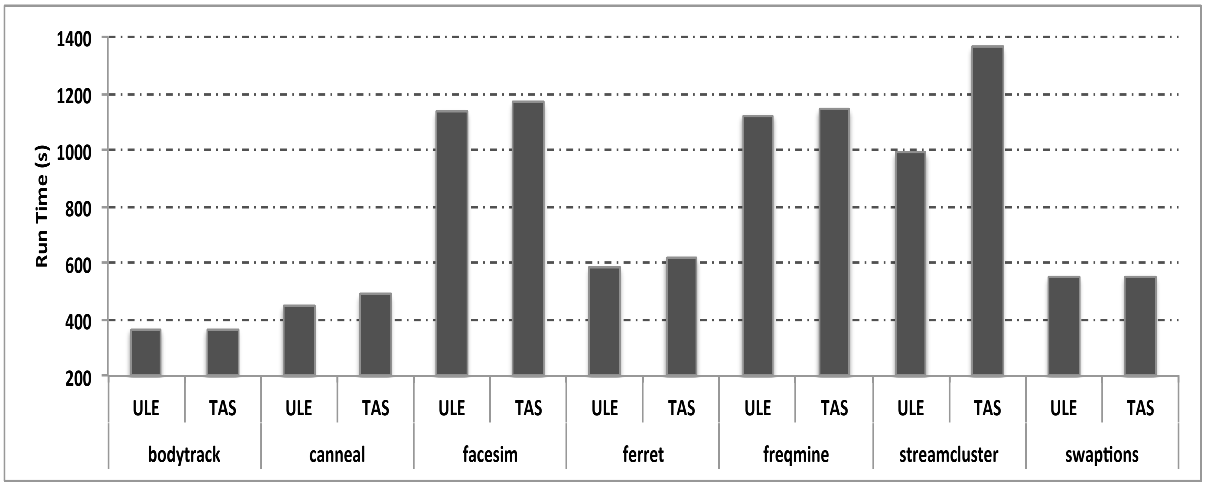


Figure 7.2: Comparison of PARSEC benchmark performance under ULE and TAS schedulers.

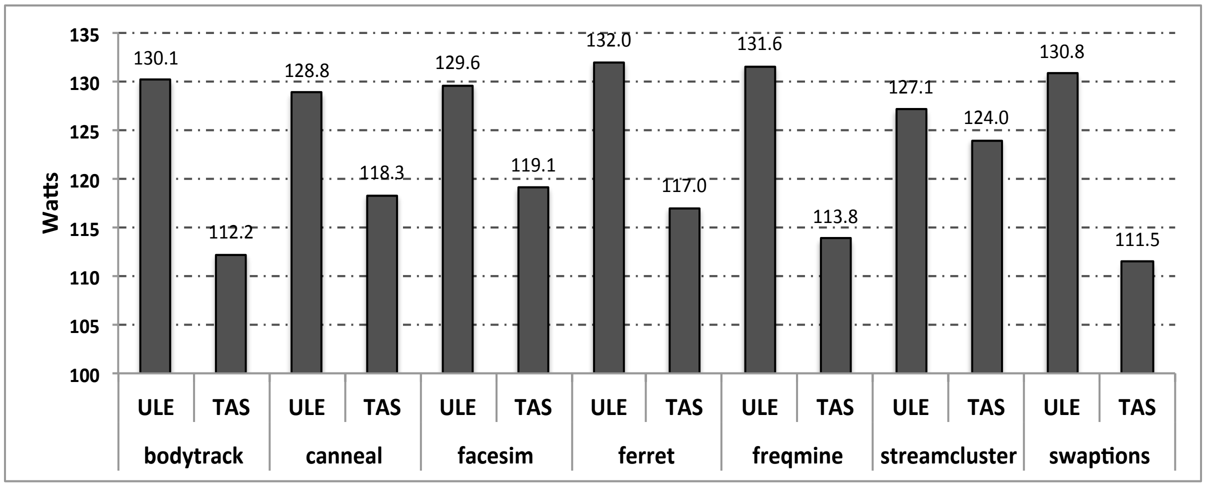


Figure 7.3: Comparison of PARSEC benchmark average power dissipation under ULE and TAS schedulers.

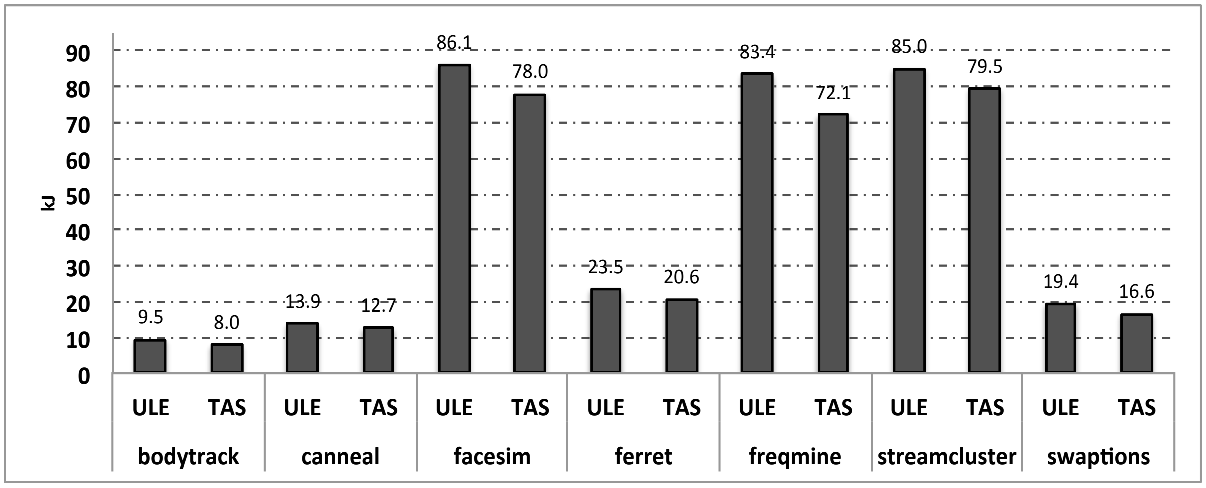


Figure 7.4: Comparison of PARSEC total energy consumption under ULE and TAS schedulers.

benchmarks with relatively small working sets, resulting in 14% to 17% reduction.

Chapter 8

Conclusions

A fast and accurate model for energy consumption and thermal envelope in a server is critical to understanding and solving the power management challenges unique in dense servers. In this work, we have introduced a comprehensive model of energy consumption by servers as a continuous system of differential equations. The model measures energy input to the system as a function of the work done for completing tasks being gauged and the residual thermal energy given off by the system as a result. Traffic on the system bus, misses in the L2 cache, CPU temperatures, and ambient temperatures are combined together to create a model, which can be employed to manage the processor thermal envelope.

The model serves as a predictive tool by approximating observed performance metrics in a discrete time series for estimating future metrics, and thus corresponding energy consumption amounts. It was found through experimental validation that commonly used techniques of regressive time series forecasting, while attractive because of their simplicity, inadequately capture the non-linear and chaotic dynamics of metric readings for typical server systems. Therefore, a chaotic time series approximation for run-time power consumption is adopted to arrive at Chaotic Attractor Prediction (CAP), which exhibits polynomial time complexity. Our proposed model is the first step towards building solutions for power and thermal management in data centers usually housing many servers.

Dense servers pose power and thermal challenges, which often cannot be addressed satisfactorily by conventional DVFS and DTM mechanisms, especially under heavy workloads common for high-performance systems. We have investigated into thermal-aware scheduling to deal with such challenges, capable of managing system energy consumption within a given power and thread envelope effectively. As opposed to prior work aiming to bound temperatures below critical thresholds, our proposed scheduler considers how to dispatch heavy workloads in the high-performance multi-core system for die temperature management across all cores. It is based on the thermal Chaotic Attractor Predictors (tCAPs) we develop to guide thread selection and load balancing, taking into account key thermal indicators and system performance metrics for preventing DTM instances proactively. The proposed tCAP-oriented scheduling (dubbed the TAS scheduler) has been implemented to replace the original scheduler of the FreeBSD operating system (called the ULE scheduler) for evaluation on a testbed server under benchmarks from the SPEC CPU2006 and PARSEC suites. Experimental results demonstrate that our TAS scheduler can lower the mean on-die core temperature by up to 12.8°C (from 44.8°C down to 32.0°C) under PARSEC benchmarks and by up to 3.3°C under mixes of SPEC benchmarks for concurrent execution, while exhibiting negligible performance degradation, in comparison to the ULE scheduler. When compared with a recent energy-aware scheduling technique reported to attain core temperature reduction by up to 4°C (from 63°C down to 59°C) upon executing four parallel scientific applications compatible to PARSEC benchmarks on an Intel Xeon 5520 4-core processor [Sarood et al. 2011], our TAS clearly enjoys better thermal reduction under multi-threaded execution.

8.1 Future Directions

The topics of power and thermal management in large scale computing have only recently begun to receive systematic attention. Our work serves as a starting point for a longer-term research program to consider some unanswered question related to these topics.

The experimental validation of CAP reveals opportunities for further investigation of power modeling. The model developed in this work is valid for any dual-core/dual-processor system using NUMA memory access connected in a point-to-point manner using the HyperTransport or the QPL structures. However, it can be scaled to quad-core dual processors based on those two structures. One would expect to see a slight difference or variation in power prediction due to a greater or less affect of die temperatures on the other performance measures. Under a dual-core quad-processor server, for example, additional regression variables would be incorporated in E_{proc}, giving rise to more performance measures (i.e., a larger r). Similarly, more PeCs related to cache misses would then be involved in E_{mem}. The solution approach of CAP remains exactly identical, except for a larger r in its prediction computation. CAP has been validated for NUMA-based servers, built on AMD Operton processors and Intel Xeon processors with Nehalem architecture; it requires validation on other architectures, like NVIDA GPU processors and IBM Cell BE processors. Further studies on the power and thermal envelope of multi-chip server systems, which involve network traffic and off-chip synchronization traffic, is required to understand their contributions to the system thermal envelope.

Operating system schedulers such as the FreeBSD ULE scheduler were introduced before the wide-spread adoption of multi-core processors such as those used in our evaluation. Thus, their scheduling decisions rarely take into account the interactions within multi-core processors that arise from contention of shared resources. At best, as we have seen with our extensions to push migration in TAS, they make very coarse-grain decisions about how to address such resource contention. For instance, recent studies [McCullough et al. 2011] identified that many methods used for power modeling suffer high prediction errors due to inherent complexities of multiple cores, hidden device states, and large dynamic power components. The question of how such errors impacts scheduling decisions has yet to be explored for both existing schedulers and energy/thermal aware schedulers such as TAS.

Our Thermal Aware Scheduler considers scheduling only within a single server blade. High-performance computing applications distribute workload across multiple environments using interfaces such as MPI and OpenMP. A topic for further research is how to extend the CAP and Thermal Aware Scheduling into such environments. A recent study of power profiles of a four-node cluster executing MapReduce-style codes [Davis et al. 2011] found that inter-node variability in homogeneous clusters leads to substantially different single-node models with higher error rates for cluster level measurements. This study shows the need to carefully consider the dynamics of the interactions between nodes and tools like our chaotic attractor predictors are well-positioned as a predictive tool for use in this use-case scenario.

A related topic is consideration of the effect of operating-system virtualization on Thermal Aware Scheduling. Some attention has been applied to this area in recent work [Merkel et al. 2010] but questions remain open considering issues of scheduler interference between host and virtual operating systems and the impact of virtualization on power management in the clustered, grid, and cloud environment. Open questions remain as to how a scheduler in the virtual machine manager coordinates energy and thermal management decisions with an operating system in a virtual machine that is unaware of the virtual environment. Scheduling decisions made within the virtual machine may lead to unexpected contention for resources in use by other virtual machines with unexpected consequences for thermal management.

Bibliography

2006. *Dual-Core Intel Xeon Processor 5100 Series Thermal/Mechanical Design Guidelines.*

AMD. 2006. *BIOS and Kernel Developer's Guide for AMD Athlon 64 and AMD Opteron Processors*, 26094 Rev 3.30 ed. AMD.

AMD. 2007. *AMD Opteron Processor Data Sheet*, 3.23 ed. AMD.

AMD. 2008. *Software Optimization Guide for AMD Family 10h Processors*, 3.06 08 ed. AMD.

Ayoub, R., Indukuri, K., and Rosing, T. 2011. Temperature aware dynamic workload scheduling in multisocket CPU servers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30,* 9 (Sept.), 1359 –1372.

Ayoub, R. Z. and Rosing, T. S. 2009. Predict and act: dynamic thermal management for multi-core processors. *Proc. of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design*, 99–104.

Bailis, P., Reddi, V. J., Gandhi, S., Brooks, D., and Seltzer, M. 2011. Dimentrodon: Processor-level preventive thermal management via idle cycle injection. In *Proceedings of the 48th Design Automation Conference (DAC 2011).* ACM, New York, NY, USA.

Bellosa, F., Weissel, A., Waitz, M., and Kellner, S. 2003. Event-driven energy accounting for dynamic thermal management. *Proceedings of the 2003 Workshop on Compilers and Operating Systems for Low Power*.

Bertran, R., Gonzalez, M., Martorell, X., Navarro, N., and Ayguade, E. 2010. Decomposable and Responsive Power Models For Multicore Processors Using Performance Counters. In *Proceedings of the 24th ACM International. Conf. on Supercomputing.* ICS '10. ACM, New York, NY, USA, 147–158.

Bhattacharjee, A. and Martonosi, M. 2009. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. *Proceedingsof the 36th International. Symposium on Computer Architecture*, 290–301.

Bienia, C. 2011. Benchmarking Modern Multiprocessors. Ph.D. thesis, Princeton University.

Bienia, C., Kumar, S., Singh, J. P., and Li, K. 2008. The PARSEC benchmark suite: characterization and architectural implications. In *Proceedings of the 17th International. Conf. on Parallel Architectures and Compilation Techniques.* PACT '08. ACM, New York, NY, USA, 72–81.

Bircher, W. and John, L. 2007. Complete system power estimation: A trickle-down approach based on performance events. *Ispass*, 158–168.

Bircher, W. and John, L. 2011. Complete System Power Estimation using Processor Performance Events. *IEEE Transactions on Computers Pre-print.*

Bircher, W., Law, J., Valluri, W., and John, L. 2004. Effective use of performance monitoring counters for run-time prediction of power. Tech. Rep. TR-041104-01, The University of Texas at Austin. November.

Bircher, W. L. and John, L. K. 2008. Analysis of dynamic power management on multi-core processors. In *Proceedingsof the 22nd International. Conf. on Supercomputing*. ACM, New York, NY, USA, 327–338.

Bowman, A. and Azzalini, A. 1997. *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-Plus Illustrations*. Oxford University Press.

Box, G., Jenkins, G., and Reinsel, G. 1994. *Time Series Analysis, Forecasting and Control*. Prentice Hall, New York, NY, USA.

Brochard, L., Panda, R., and Vemuganti, S. 2010. Optimizing Performance and Energy of HPC Applications on POWER7. *Computer Science – Research and Development 25*, 135–140. 10.1007/s00450-010-0123-3.

Cher, C.-Y. and Kursun, E. 2011. Exploring the effects of on-chip thermal variation on high-performance multicore architectures. *ACM Trans. on Architecture and Code Optimization 8*, 2:1–2:22.

Choi, J., Cher, C.-Y., Franke, H., Hamann, H., Weger, A., and Bose, P. 2007. Thermal-aware task scheduling at the system software level. *Proceedings of the ACM International Symposium on Low Power Electronics and Design*, 213–218.

Cisco Systems, I. 2010. Using SPEC CPU2006 Benchmark Results To Compare the Compute Performance of Servers. white paper.

Contreras, G. and Martonosi, M. 2005. Power prediction for Intel XScale® processors using performance monitoring unit events. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*. ACM, New York, NY, USA, 221–226.

Coskun, A. K., Rosing, T. S., and Gross, K. C. 2008. Proactive temperature balancing for low cost thermal management in MPSoCs. *Proceedings of the 2008 IEEE/ACM International. Conf. on Computer-Aided Design*, 250–257.

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

Crew, J., Kuruvilla, K., Saxe, E., Vanoni, R., Moore, S., Dhillon, I., Gao, Y., Song, Y., Seberger, D., and Copty, N. 2009. The Solaris Operating System - Optimized for the Intel Xeon Processor 5500 Series. Tech. rep., Sun Microsystems, Inc.

Davis, J., Rivoire, S., Goldszmidt, M., and Ardestani, E. 2011. Accounting for variability in large-scale cluster power models. In *Proceedings of the 2nd Exascle Evaluation and Research Techniques Workshop*.

Donald, J. and Martonosi, M. 2006. Techniques for multicore thermal management: classification and new exploration. In *Proceedings of the 33rd International Symposium on Computer Architecture*. IEEE Computer Society, Washington, DC, USA, 78–88.

Economou, D., Rivoire, S., Kozyrakis, C., and Ranganathan, P. 2006. Full-system power analysis and modeling for server environments. In *Proceedings of the 2008 Workshop on Modeling Benchmarking and Simulation.*

Electronic Educational Devices, Inc. 2006. WattsUp Power Meter.

Fan, J. and Gijbels, I. 1996. *Local Polynomial Modeling and Its Applications.* Chapman & Hall, London, UK.

Fan, J. and Yao, Q. 2005. *Nonlinear Time Series.* Springer New York, New York, NY, USA.

Fan, X., Weber, W.-D., and Barroso, L. A. 2007. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th International Symposium on Computer Architecture*. 13–23.

Friedman, J. 1991. Multivariate Adaptive Regression Splines. *Annals of Statistics 19*, 1–142.

Gomaa, M., Powell, M. D., and Vijaykumar, T. N. 2004. Heat-and-Run: leveraging SMT and CMP to manage power density through the operating system. *SIGOPS Oper. Syst. Rev. 38,* 5, 260–270.

Gurumurthi, S., Sivasubramaniam, A., and Natarajan, V. 2005. Disk drive roadmap from the thermal perspective: A case for dynamic thermal management. In *Proceedings of the 32nd International. Symposium on Computer Architecture*. 38 – 49.

Hamill, D., Deane, J., and Aston, P. 1997. Some Applications of Chaos In Power Converters. In *IEEE Colloquium on Update on New Power Electronic Techniques (Digest No: 1997/091).*

Hanson, H., Keckler, S. W., Ghiasi, S., Rajamani, K., Rawson, F., and Rubio, J. 2007. Thermal response to DVFS: analysis with an Intel Pentium M. In *Proceedings of the 2007 international symposium on Low power electronics and design*. ISLPED '07. ACM, New York, NY, USA, 219–224.

Heath, T., Centeno, A. P., George, P., Ramos, L., Jaluria, Y., and Bianchini, R. 2006. Mercury and Freon: Temperature emulation and management for server systems. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, New York, NY, USA, 106–116.

Heath, T., Diniz, B., Carrera, E. V., Jr., W. M., and Bianchini, R. 2005. Energy conservation in heterogeneous server clusters. In *Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 186–195.

Henning, J. L. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comp. Archit. News 34,* 4, 1–17.

Hofmeyr, S., Iancu, C., and Blagojević, F. 2010. Load balancing on speed. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP '10. ACM, New York, NY, USA, 147–158.

Hsu, C.-H. and Poole, S. 2011. Power signature analysis of the SPECpower_ssj2008 benchmark. In *Proceedings of the 2011 IEEE International. Symposium on Performance Analysis of Systems and Software*. 227 –236.

HyperTransport Technology Consortium. 2007. HyperTransport I/O Link Specification. Specification 3.00c, HyperTransport Technology Consortium. September.

Intel. 2009. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. Intel Corporation, P.O. Box 5937; Denver, CO.

Isci, C. and Martonosi, M. 11-15 Feb. 2006. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. *Proceedings of the 12th International Symposium on High-Performance Computer Architecture*, 121–132.

Isci, C. and Martonosi, M. 2003a. Identifying program power phase behavior using power vectors. *Proceedings of the IEEE 2003 International Workshop on Workload Characterization*, 108–118.

Isci, C. and Martonosi, M. 2003b. Runtime power monitoring in high-end processors: Methodology and empirical data. *Proceedings of the 36th IEEE/ACM International Symposium on Microarchitecture*, 93–104.

Itoh, K. 1995. A Method for Predicting Chaotic Time-series With Outliers. *Electronics and Communications in Japan, Part 3: Fundamental Electronic Science 78,* 5 (Apr), 1529–1536.

Jiménez, V., Cazorla, F., Gioiosa, R., and Valero, M. 2010. Power and thermal characterization of POWER6 system. *Proceedings of the 19th International Conf. on Parallel Architectures and Compilation Techniques*.

Kadayif, I., Chinoda, T., Kandemir, M., Vijaykirsnan, N., Irwin, M., and Sivasubramaniam, A. 2001. vEC: Virtual energy counters. In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*. ACM, 28–31.

Kansal, A., Zhao, F., Liu, J., Kothari, N., and Bhattacharya, A. A. 2010. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC '10. ACM, New York, NY, USA, 39–50.

Khan, M., Hankendi, C., Coskun, A., and Herbordt, M. 2011. Software optimization for performance, energy, and thermal distribution: initial case studies. In *Proceedings of the 2011 International. Green Computing Conference and Workshops*. 1 –6.

Kumar, A., Shang, L., Peh, L.-S., and Jha, N. 2008. System-Level Dynamic Thermal Management for High-Performance Microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27,* 1 (Jan.), 96–108.

Kumar, A., Shang, L., Peh, L.-S., and Jha, N. K. 2006. HybDTM: a coordinated hardware-software approach for dynamic thermal management. *Proceedings of 43th ACM/IEEE Design Automation Conference*, 548–553.

Kursun, E. 2008. Variation-aware thermal characterization and management of multi-core architectures. In *Proceedings of the 2008 IEEE International Conference on Computer Design*. IEEE, 280–285.

Kursun, E. and Cher, C.-Y. 2009. Temperature variation characterization and thermal management of multicore architectures. *IEEE Micro 29*, 116–126.

Lee, K.-J. and Skadron, K. 2005. Using performance counters for runtime temperature sensing in high-performance processors. *Proceedings of the 19th IEEE International Symposium Parallel and Distributed Processing*.

Lewis, A., Ghosh, S., and Tzeng, N.-F. 2008. Run-time energy consumption estimation based on workload in server systems. *Proceedings of the 2008 Workshop on Power Aware Computing and Systems (Hotpower'08)*.

Lewis, A., Simon, J., and Tzeng, N.-F. 2010. Chaotic attractor prediction for server run-time energy consumption. *Proceedings of the 2010 Workshop on Power Aware Computing and Systems (Hotpower'10)*.

Li, K. 2008. Performance Analysis of Power-Aware Task Scheduling Algorithms on Multiprocessor Computers with Dynamic Voltage and Speed. *IEEE Transactions on Parallel and Distributed Systems 19,* 11 (Nov.), 1484–1497.

Li-yun Su. 2010. Prediction of multivariate chaotic time series with local polynomial fitting. *Computers & Mathematics with Applications 59,* 2, 737 – 744.

Linter, W., Tschudi, B., and VanGeet, O. 2011. Best Practices Guide for Energy-Efficent Data Center Design. Tech. rep., U.S. Dept. of Energy.

Liu, S., Memik, S. O., Zhang, Y., and Memik, G. 2008. A power and temperature aware DRAM architecture. In *Proceedings of the 45th Annual Design Automation Conference*. DAC '08. ACM, New York, NY, USA, 878–883.

Liu, Z. 2010. Chaotic Time Series Analysis. *Mathematical Problems in Engineering 2010*.

London, K., Moore, S., Mucci, P., Seymour, K., and Luczak, R. 2001. The PAPI Cross-platform Interface to Hardware Performance Counters. *Department of Defense Users' Group Conference Proceedings*.

McCullough, J. C., Agarwal, Y., Chandrashekar, J., Kuppuswamy, S., Snoeren, A. C., and Gupta, R. K. 2011. Evaluating the effectiveness of model-based power characterization. In *Proceedings of the 2011 USENIX ATC*.

McKusick, M. K. and Neville-Neil, G. V. 2004a. *The Design and Implementation of the FreeBSD Operating System*. Pearson Education.

McKusick, M. K. and Neville-Neil, G. V. 2004b. Thread scheduling in FreeBSD 5.2. *Queue 2*, 58–64.

Merkel, A. and Bellosa, F. 2008a. Memory-aware scheduling for energy efficiency on multicore processors. *Proceedings of the 2008 Workshop on Power Aware Computing and Systems (Hotpower'08)*.

Merkel, A. and Bellosa, F. 2008b. Task activity vectors: a new metric for temperature-aware scheduling. *Proceedings of 3rd ACM SIGOPS/Eurosys European Conference on Computer Systems*, 1–12.

Merkel, A., Stoess, J., and Bellosa, F. 2010. Resource-conscious scheduling for energy efficiency on multicore processors. In *Proceedings of the 5th European conference on Computer systems*. EuroSys '10. ACM, New York, NY, USA, 153–166.

Mesa-Martinez, F., Nayfach-Battilana, J., and Renau, J. 2007. Power model validation through thermal measurements. *Proceedings of the 34th International Conference on Computer Architecture*, 302–311.

Mesa-Martinez, F. J., Ardestani, E. K., and Renau, J. 2010. Characterizing Processor Thermal Behavior. *SIGARCH Comput. Archit. News 38*, 193–204.

Micron, Inc. 2007. Calculating Memory System Power for DDR3. Tech. Note TN41_01DDR3 Rev.B, Micron, Inc. August.

Murali, S., Mutapcic, A., Atienza, D., Gupta, R., Boyd, S., Benini, L., and De Micheli, G. 2008. Temperature control of high-performance multi-core platforms using convex optimization. In *Proceedings of the 2010 Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE, 110–115.

NIST. 2010. NIST/SEMATECH e-Handbook of Statistical Methods.

Phansalkar, A., Joshi, A., and John, L. K. 2007. Analysis of Redundancy and Application Balance In the SPEC CPU2006 Benchmark Suite. *SIGARCH Comput. Archit. News 35*, 412–423.

Powell, M., Biswas, A., Emer, J., Mukherjee, S., Sheikh, B., and Yardi, S. 2009. CAMP: A technique to estimate per-structure power at run-time using a few simple parameters. In *Proceedings of the IEEE 15th International. Symposium on High Performance Computer Architecture*. 289 –300.

Reich, J., Goraczko, M., Kansal, A., Padhye, J., and Computing, N. 2010. Sleepless in Seattle no longer. *Proceedings of the 2010 USENIX Annual Tech. Conference*.

Rivoire, S. 2008. Models and Metrics for Energy-efficient Computer Systems. Ph.D. thesis, Stanford University.

Rivoire, S., Ranganathan, P., and Kozyrakis, C. 2008. A comparison of high level full-system power models. In *Proceedings of 2008 USENIX Workshop on Power Aware Computing and Systems*.

Roberson, J. 2003. ULE: a modern scheduler for freebsd. In *Proceedings of the BSD Conference 2003 on BSD Conference*. USENIX Association, Berkeley, CA, USA, 3–3.

Rosing, T. S., Mihic, K., and De Micheli, G. 2007. Power and reliability management of SoCs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 15,* 4 (Apr.), 391–403.

Sarood, O., Gupta, A., and Kale, L. 2011. Temperature aware load balancing for parallel applications: Preliminary work. In *Proceedings of the 2011 IEEE International. Symposium on Parallel and Distributed Processing Workshops and Ph.D. Forum*. 796 –803.

Server System Infrastructure Consortium. 2004. EPS12v Power Supply Design Guide, V2.92. Spec. 2.92, Server System Infrastructure Consortium.

Singh, K., Bhadauria, M., and McKee, S. A. 2009. Real Time Power Estimation and Thread Scheduling Via Performance Counters. *SIGARCH Comput. Archit. News 37,* 2, 46–55.

Skadron, K., Stan, M. R., Sankaranarayanan, K., Huang, W., Velusamy, S., and Tarjan, D. 2004. Temperature-aware Microarchitecture: Modeling and Implementation. *ACM Transactions on Architecture and Code Optimization 1,* 1, 94–125.

Sprott, J. 2003. *Chaos and Time-Series Analysis.* Oxford University Press, New York, NY, USA.

Sun Microsystems. 2009. OpenSolaris CPU Power Management - Project Tesla.

Sun Microsystems, I. 2008. Solaris System Administration Guide: Advanced Administration.

Ton, M., Fortenbery, B., and Tschudi, W. 2008. DC Power for Improved Data Center Efficiency. Tech. rep., Lawrence Berkeley National Laboratory.

Tong, H. 1993. *Non-linear Time Series: A Dynamical System Approach.* Oxford University Press, New York, NY, USA.

Tse, C. and Di Bernardo, M. 2002. Complex Behavior In Switching Power Converters. *Proceedings of the IEEE 90,* 5 (May), 768 –781.

Tsirogiannis, D., Harizopoulos, S., and Shah, M. A. 2010. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 International. Conference on Management of Data.* SIGMOD '10. ACM, New York, NY, USA, 231–242.

Turlach, B. 1993. Bandwidth Selection in Kernel Density Estimation: A Review. *CORE and Institut de Statistique*, 23–493.

Varsamopoulos, G., Abbasi, Z., and Gupta, S. 2010. Trends and effects of energy proportionality on server provisioning in data centers. In *Proceedings of the 2010 International. Conference on High Performance Computing.* 1 –11.

Ware, M., Rajamani, K., Floyd, M., Brock, B., Rubio, J., Rawson, F., and Carter, J. 2010. Architecting for power management: The IBM POWER7 approach. In *Proceedings of the IEEE 16th International. Symposium on High Performance Computer Architecture.* 1 –11.

Xia, L., Zhu, Y., Yang, J., Ye, J., and Gu, Z. 2010. Implementing a Thermal-Aware Scheduler in Linux Kernel on a Multi-Core Processor. *The Computer Journal 53,* 7, 895–903.

Yang, J., Zhou, X., Chrobak, M., Zhang, Y., and Jin, L. 2008. Dynamic thermal management through task scheduling. *Proceedings of the 2008 IEEE International Symposium on Performance Analysis of Systems and Software*, 191–201.

Yao, F., Demers, A., and Shenker, S. 1995. A scheduling model for reduced cpu energy. *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, 374.

Zhou, X., Yang, J., Chrobak, M., and Zhang, Y. 2010. Performance-aware thermal management via task scheduling. *ACM Transactions on Architecture and Code Optimization (TACO) 7,* 1, 1–31.

Lewis, Adam Wade. Bachelor of Arts, The University of the South, 1986; Master of Science, University of Tennessee at Chattanooga, 1989; Doctor of Philosophy, University of Louisiana at Lafayette, Spring 2012

Major: Computer Science

Title of Dissertation: Energy Conservation and Thermal Management in High-Performance Server Architectures

Dissertation Director: Dr. Nian-Feng Tzeng

Pages in Dissertation: 88; Words in Abstract: 476

ABSTRACT

Modern processors crudely manage thermal emergencies through Dynamic Thermal Management (DTM), where the processor monitors the die temperature and dynamically adjusts the processor voltage and frequency (DVFS) to throttle down the processor when necessary. However, DVFS tends to yield marked degradation in both application performance and system reliability. Thus, pro-active scheduling techniques that avoid thermal emergencies are preferable over reactive hardware techniques like DTM. This dissertation proposes a run-time model which relates server energy consumption to its overall thermal envelope, using hardware performance counters and experimental measurements. While previous studies have attempted system-wide modeling of server power consumption through subsystem models, our approach is different in that it links system energy input to subsystem energy consumption based on a small set of tightly correlated parameters. The proposed model takes into account processor power, bus activities, and system ambient temperature for real-time prediction on the power consumption of long running jobs. Using the HyperTransport and QuickPath Link structures as case studies and through electrical measurements on example server subsystems, we develop a chaotic time–series approximation for run-time power consumption, arriving at the Chaotic Attractor Predictor (CAP). With polynomial time complexity, CAP exhibits high prediction accuracy, having the prediction errors within 1.6% (or 3.3%) for servers based on the HyperTransport bus (or the QuickPath Links), as verified by a set of common processor benchmarks. Our CAP is a superior predictive mechanism over existing linear auto-regressive methods, which require expensive and complex corrective steps to address the non-linear and chaotic aspects of the underlying physical system.

Based on our Chaotic Attractor Predictors, we have developed and evaluated an

effective thread scheduler for multi-core systems. Besides CAPs, our scheduler makes use of two basic principles to minimize server energy consumption: (1) selecting the thread with the least probability of causing a DTM in the subsequent time quantum, for execution on the next available core, and (2) migrating execution threads on thermally overextended cores to other cool cores via load balancing so as to observe the thermal envelope. Our developed scheduler is evaluated in practice to assess its potential advantage resulting from thermal-awareness by incorporating CAPs (for temperature prediction) and basic principles (for energy reduction) into the existing scheduler in the FreeBSD operating system. Our implemented scheduler is run on a server with the Intel Xeon processor for gathering measures of interest (including die temperature readings and execution times) when benchmark codes from PARSEC and SPEC CPU2006 suites are executed. The gathered results reveal that our proposed scheduler exhibits reduction in mean core on-die temperatures by up to 12.8°C under PARSEC benchmarks (which have multi-threaded workloads) and by up to 3.3°C under mixes of SPEC CPU2006 benchmarks for concurrent execution on all four server cores, while experiencing only 1% to 4% performance degradation.

Biographical Sketch

Adam Wade Lewis received a Bachelor of Arts degree in Mathematics and a Bachelor of Arts degree in Theatre from the The University of the South in 1986 and a Master of Science degree in Computer Science from The University of Tennessee at Chattanooga in 1989. His background includes experience in the retail information technology and networking engineering fields as a software developer, business analyst, and project manager for NCR Corporation throughout the world. He is a member of SIAM, the ACM, and the IEEE Computer Society.