Energy Conservation and Thermal Management in High-Performance Server Architectures

A Prospectus

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 2011

Energy Conservation and Thermal Management in High-Performance Server Architectures

Adam Wade Lewis

APPROVED:

Nian-Feng Tzeng, Chair
Professor of Computer Engineering

Magdy Bayoumi
Head and Professor of Computer Engineering

Dmitri Perkins
Associate Professor of Computer Science

Ashok Kumar
Assistant Professor of Computer Science

C. E. Palmer
Dean of the Graduate School

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The upwardly spiraling operating costs of the infrastructure for enterprise-scale computing demand efficient power management in server environments. It is difficult in practice to achieve efficient power management as data centers usually over-provision their power capacity to address the worst case scenarios. This results in either waste of considerable power budget or severe under-utilization of capacity. Thus, it is critical to quantitatively understand the relationship between power consumption and thermal envelope at the system level so as to optimize the use of deployed power capacity in the data center.

Chip multiprocessors (CMP) and processors employing simultaneous multi-threading (SMT) have become common in server blades due to advantages these devices have in low design cost and better performance. As these technologies have evolved, there has been an evolution from *multi-core* processors composed of a limited number of homogeneous cores to the idea of *many-core* processors composed of many heterogeneous processors cores. However, the greater chip complexity entailed by many-core processors lead to larger power envelopes, elevated peak chip die temperatures, and imbalanced thermal gradients. So, as the workload increases on the server, so does the thermal stress placed on the processor with the resulting probability of damage to the machine.

Modern processors crudely manage this problem through Dynamic Thermal Management (DTM) where the processor monitors the die temperature and dynamically adjusts the processor voltage and frequency (DVFS) to slow down the processor. However, DVFS has a significant negative impact on application performance. An examination of the impact of dynamic power adaption on the performance and power envelope of a Quad-Core AMD Opteron processor found that

significant performance loss is entailed due to slow transitions between different levels of DVFS [Bircher and John 2008]. Indirect effects due to shared, power-managed resources such as cache and memory can greatly reduce performance; this is a significant problem for memory-bound workloads typical of server workloads. Thus, pro-active scheduling techniques that avoid thermal emergencies are preferable to reactive hardware techniques such as DTM.

Furthermore, there is a significant cost to the use of DVFS as a method to avoid thermal emergencies. A categorization of the different thermal management techniques based on this approach can be found in [Donald and Martonosi 2006]. In that work, the authors claimed the best approach to using DTM/DVFS includes a combination of a control-theoretic distributed DVFS system and a sensor-based migration policy. The issue with this approach is the granularity of DVFS; existing commercial systems do not allow for independent frequency scaling of cores and threads. A related approach is the capability of turning off the power to unused cores in the processor. However, use of this capability for power management can lead to reduced reliability in the processor [Rosing et al. 2007; Coskun et al. 2008a]. In addition, operating system schedulers operate on a millisecond time scale but architectural events such as cache misses occur vary on a nanosecond time scale. Thus, operating system driven DVFS can be too slow to adapt to fine scale variations in program behavior [Rangan et al. 2009].

The component of the operating system most aware of the existence of multiple cores is the thread scheduler. The current generation of operating systems treat the CPU cores in a multi-core processor as distinct physical processors when making scheduling decisions. Furthermore, operating systems treat virtualized SMT processors on a single physical core (such as Intel's HyperThreading technology) as "logical CPUs" that behave as independent processors for scheduling purposes. The thread scheduler is responsible for ensuring that all of the logical CPUs 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 is amongst the logical CPUs on a single

processor or across physical processors. However, data and instruction dependencies exist between logical CPUs that must be taken into account when balancing performance and energy efficiency. These dependencies lead to contention for shared resources that result in performance penalties and inefficient use of energy.

Modern multiprocessor operating systems such as Windows, Linux, Solaris, and FreeBSD take a two-level approach to scheduling in an attempt to maximize system resources. The first level manages each core using a distributed run queue model with per logical CPU queues and fair scheduling policies. The second level attempts to balance the resource load by redistributing tasks across the queues on each logical CPU. The design of this type of scheduler is based upon three principles: (1) threads are assumed to be independent, (2) load is equated to queue length, and (3) locality is important [Hofmeyr et al. 2010].

Traditional server scheduling makes assumptions about workload behavior to make scheduling decisions. Interactive workloads are characterized by the independent tasks that remain quiet for extended periods. Server workloads contain large numbers of threads that are highly independent of each other that use synchronization objects to ensure mutual exclusion on small data items. In parallel applications, there is a higher levels of interaction between threads; consequently, load balancing such applications requires additional mechanisms.

Parallel applications are implemented most often on contemporary processors using a Single Process, Multiple Data (SPMD) programming model where logical CPUs simultaneously execute the same program at independent points. Tasks execute independently and communicate with other nodes by sending and receiving messages with some form of barrier synchronization implemented as part of the message interface. The details of the message process is isolated from the application by standard interfaces such as MPI and OpenMP. Note how this programming model contravenes the assumptions for system level load balancing: (1) threads are logically related, (2) have data and control dependencies between threads, and (3) have equally

long life-spans [Hofmeyr et al. 2010].

In this work, we introduce two techniques for thermal management that minimizes server energy consumption by (1) for each logical CPU, selecting the next thread to execute based upon an estimate of which thread has the least probability of causing a DTM in the next quantum, and (2) adjusting the load balance allocation of threads to available logical CPUs so as to migrate workload away from thermally overextended resources on the processor. These techniques manage power density and thermal stress in two ways. In the first case, we seek to find the best temporal solution to the problem of maximizing duty cycle, and thus maximizing the amount of computation that occurs per unit cooling time. Thermally-aware load balancing as suggested by our second contribution is a spatial solution for managing power density where we move computation in a hot resource to redundant or under-utilized resources.

Intelligent thermally-aware scheduling decisions requires an full-system model of energy consumption based on computational load of system that can be used to effectively predict future energy consumption and resulting changes in thermal load. This Prospectus presents a full-system model that provides run-time system-wide prediction of energy consumption on server blades as a continuous system of differential equations, taking thermal effects into account. Our model considers a single server blade as a 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 relates server energy consumption to its overall thermal envelope, establishing an energy relationship between the workload and the overall thermodynamics of the system.

Our approach measures the total DC power input to the server at its power supply output, treating total energy delivered to the system as a sum of the energy components consumed by different sub-systems in the server. The approach 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. Given the current generation of server systems lack (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, our approach resorts to statistical approximation to compensate for those model components which cannot be determined exactly. In other words, our energy consumption prediction model is approximated following a discrete time series of observed readings of available PeCs.

Generally, time series models operate by observing past outcomes of a physical phenomenon in order to anticipate future values of that phenomenon. Many time series-based models of processor energy consumption have been proposed [Rivoire 2008; Bhattacharjee and Martonosi 2009; Reich et al. 2010], 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, and processor die temperatures in computers can be affected by more than just past values of those measurements made in isolation from each other. 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*. It thus leads to our development of a Chaotic Attractor Predictor (CAP).

Servers based on the HyperTransport bus [HyperTransport Technology Consortium 2007] and the Intel QuickPath Links [Intel 2009] are chosen as representatives to valid valid our CAP for estimating run-time power consumption. CAP 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. This work demonstrates that appropriate provision of additional PeCs beyond what are provided by a typical server is required to obtain more accurate prediction of system-wide energy consumption. Effective scheduling can result from taking advantage of the proposed CAP when dispatching jobs to confine server power consumption within a given power budget and thermal envelope while minimizing impact upon server performance.

Chapter 2

Background and Related Work

Power management techniques developed for mobile and desktop computers have been applied with some success to managing the power consumption of microprocessors used in server hardware. The current generation of AMD and Intel processors employs different techniques for processor-level power management, including (1) per core clock gating, (2) power-gating functional blocks (for processors to turn off certain blocks not in use), (3) multiple clock domains, (4) multiple voltage domains for cores, caches, and memory, (5) dynamic voltage and frequency scaling per core and per processor, and (6) hardware support for virtualization techniques [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 on the processor for energy savings [Contreras and Martonosi 2005]. Power profiles of the Intel Pentium architecture have been studied for workstations [Isci and Martonosi 2003a; 2003b;] and servers [Bircher et al. 2004; Bircher and John 2007; Lee and Skadron 2005]. However, little consideration has ever been given to the power profiles of servers constructed using the NUMA-based architecture, such as the AMD64 [AMD 2007] and the Intel Nehalem [Intel 2009] processors. Management of system-critical thermal issues due to excessive power consumption is further complicated by the existence of multiple cores per processor.

Power models are the basis for power management, but existing power models mostly do not take thermal effects of power dissipation into consideration, despite that such effects are essential to power management in severs. They can be classified into two broad categories: simulation-based models and detailed analytical power 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. Existing simulation-based models do not fit well in scenarios where dynamic power and thermal optimization for application performance is required [Economou et al. 2006].

Analytical models use detailed knowledge of underlying hardware to directly estimate energy consumption at the hardware level. They rely on power measurements at the micro-architectural units of the processor via sampling hardware and software performance metrics. Two distinct classes of metrics have been used in those models: 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. Typically, analytic models do not take into account the energy consumption of devices other than the processor, and they require detailed knowledge of the micro-architecture of the processor. Attempts have been made to reconcile analytic models by mapping program phases to events [Isci and Martonosi]. 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; Bircher and John 2007; Lewis et al. 2008].

In general, the number of recordable events exceeds the number of available counters (i.e., PeCs). As a result, models that use these counters must time-multiplex different sets of events on the available PeCs. While it allows for more events to be monitored, time-multiplexing leads to increased overhead and lower accuracy due to sampling issues [Economou et al. 2006; Rivoire 2008]. A power model is desirable to involve the fewest possible metrics (either PeCs or OS performance measures) required to accurately capture the system behavior in order to avoid the need for time-multiplexing. High level black-box models sacrifice accuracy by avoiding extensive detailed knowledge of underlying hardware. At the processor level, Contreras *et al.* [2005] and Bellosa [2003] created power models that linearly correlated power consumption with PeCs. Meanwhile, 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]. Those models have the advantage of being simple and fast with low-overhead, but they do not consider full-system power consumption.

In server environments, full-system models can be created using operating system CPU utilization [Fan et al. 2007] and similar metrics [Heath et al. 2005]. Such full-system models, like MANTIS [Economou et al. 2006; Rivoire 2008], relate usage information to the power of the entire system rather than 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 and operating system performance metrics. This is because PeCs and OS metrics together may capture all components of system dynamic power consumption.

Auto-regressive techniques have been adopted popularly to construct power and temperature models for servers [Coskun et al. 2008], since they can yield predictors that are both quick and accurate. However, such techniques are *stationary* in nature. In a stationary process, the probability distribution does not change with time, nor do the mean and the variance. Hence, auto-regressive (AR) and auto-regressive/moving average (ARMA) models are not suited for data that exhibits sudden bursts of large amplitude 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 so as to permit continuing use of AR and ARMA models. For example, one recent work [Coskun et al. 2008] included a machine learning-based element in its predictors for on-line adaption over times. This way, however, negatively impacts the performance advantage resulting from auto-regressive techniques, namely, their simplicity.

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: given a time series of $X_1, X_2, \ldots, X_n$, a k-step forecasting result can be obtained using a function of $m(x) = E(X_{i+k}|X_i = x)$ by constructing a set of observations (X_i, Y_i), for $i = 1, \ldots, n-k$, with $Y_i = X_{i+k}$. Function $m(x)$ can be approximated by a Taylor series expansion, which realizes local fitting in the time series via solving a weighted least squares regression problem [Fan and Gijbels 1996]. In [Singh et al. 2009], a variation of this approach was proposed using piece-wise linear functions rather than Taylor series expansion.

Another 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. More details about MARS can be found in Appendix.

The idea of migration of work for energy savings and thermal management has a long history in the SMP, SMT, and CMP environments [Yao et al. 1995]. Static methods been applied in attempts to solve this problem. In Coskun `et. al`[Coskun et al. 2008b], integer linear programming was used to obtain a task schedule that met

real-time deadlines while attempting to minimize hot spots and spatial temperature differentials across the die.

Examples of dynamic methods to solve this problem include Heat-and-Run [Gomaa et al. 2004],HybDTM [Kumar et al. 2006], and ThreshHot [Yang et al. 2008; Zhou et al. 2010]. Heat-and-Run proposed to distribute work amongst available cores until the DTM events occur and then migrate threads from the overheated cores to other non-heated cores. HybDTM combined DTM techniques with a thread migration strategy that reduces the thread priority of jobs on cores that are running hot. ThreshHot uses an on-line temperature estimator to determine in what order threads should be scheduled onto cores. In all three cases, these techniques utilize data from hardware performance counters and hardware temperature sensors to guide the scheduling decision.

A related approach to migration is to control the CPU time slice allocation. In Bellosa et al. [2003], it was proposed to modify the process scheduler to allocate time slices as indicated by the contribution of each task to the system power consumption and the current temperature of the processor. A variation on this scheme was proposed in [Li 2008] where system level compiler support was used to insert run-time profiling code into applications to provide hints to the thermal intensity of a task. In Merkel `et.al.` [Merkel and Bellosa ; Merkel et al. 2010], a scheduling policy was proposed that sorts the tasks in each core's run queue by memory intensity so as to schedule memory-bound tasks at slower frequencies.

Chapter 3

System Modeling

In order to develop an 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:

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

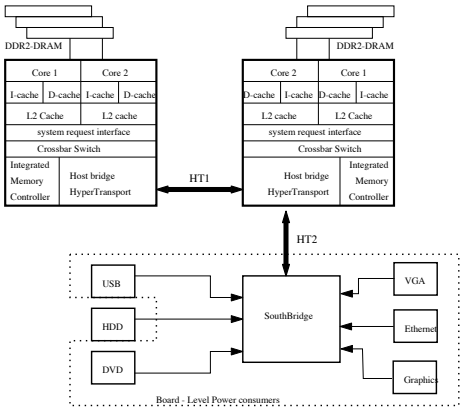


Figure 3.1: AMD Opteron architecture.

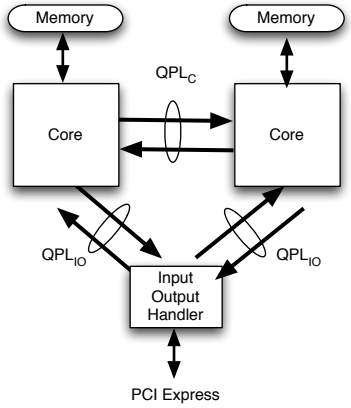


Figure 3.2: Intel Xeon (Nehalem) architecture.

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. 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 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. Both our test servers use the Hitachi's SATA hard disk

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)

(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. In our case, this value is obtained using current probe-based measurements. The results measured over an interval of interest, $t_{interval}$, excluded the effects of processor, disks, fans, and optical devices, leading to:

$$E_{board} = \left(\sum V_{power-line} \times I_{power-line}\right) \times t_{interval}. \tag{3.3}$$

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 Equation (3.3).

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:

$$P^i_{fan}(t) = P_{base} \times \left(\frac{RPM^i_{fan}(t)}{RPM_{base}} \right)^3 \tag{3.4}$$

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.

A server with N cooling fans results in electromechanical power at time t as

$$P_{elect}(t) = V(t) \cdot I(t) + \sum_{i=1}^{N} P^i_{fan}(t)$$

where the first term is instantaneous DC power output from the power supply, representing DC power consumed by the server, and $P^i_{fan}(t)$ is expressed in Equation (3.4).

Total electromechanical energy consumption over a given task execution period of T_p equals:

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

3.6 Proposed power distribution

The input to our derived energy consumption model is current readings at the power supply lines to the server components. While not available yet, current sensors (like MAXIM's 4473 [Maxim 2008]) may be placed at the outputs of the power supply unit (PSU, see Figure 3.3) to dynamically track DC power draws during job execution as system load varies. A power distribution diagram, with current sensors incorporated as measurable PeCs, is depicted in Figure 3.3, where a rack-level DC power source is used for power savings due to its better power efficiency.

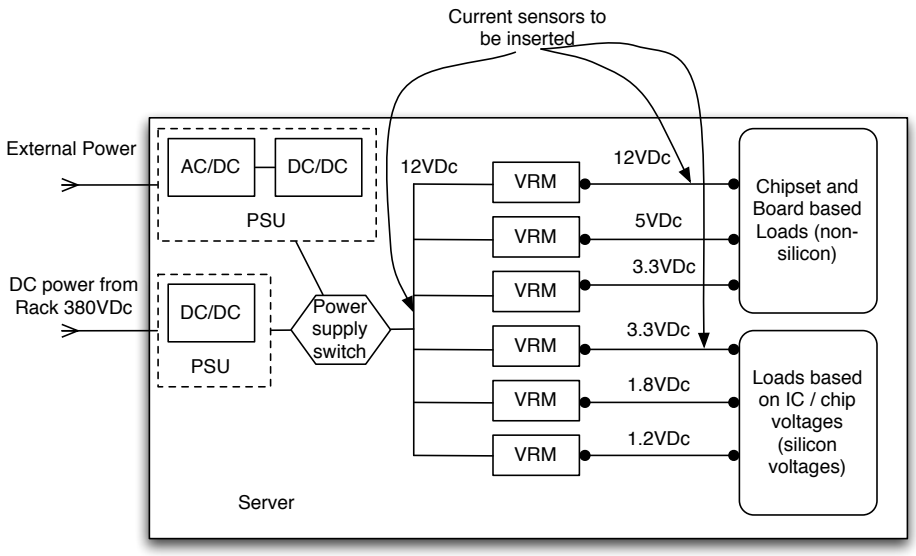


Figure 3.3: Proposed power distribution for servers.

With current sensors provided at the output of the DC voltage lines delivering power to the hard disk drives, for example, one may measure hard disk power consumption in real-time. Let the current sensors at +5V (or +12V) lines to the hard disk be denoted by $v_1(t)$ and $i_1(t)$ or ($v_2(t)$ and $i_2(t)$), which is no larger than 730mA (or 630mA), we have E_{hdd} as follows:

$$E_{hdd} = \int_{t1}^{t2} (v_1(t) \times i_1(t) + v_2(t) \times i_2(t))\, dt.$$

Given no current sensors available in our experimental servers, we obtained the readings by a current probe, logged through an oscilloscope.

Typically, a server at the idling state (when running the operating system and no application jobs) consumes some 40 to 42% of its rated power (which is 450W in our case). Conversion efficiency increases to about 80% when the server is heavily loaded, and the SMPS regulates the power supply to work at 75% conversion efficiency when power drawing is over 50% of the rated one. Hence, conversion losses of the power supplied to the system is at least 20%, even for the best conversion scenario. Earlier studies [Ton et al. 2008] have shown that most DC systems perform better in terms of power efficiency than their AC counterparts. A typical AC-based server system has a

power supply efficiency of 73%, as compared with a 92% for a DC system. Also, the overall system efficiency for an AC system is 61%, in comparison to 85% for a DC system.

While a rack level DC power distribution system easily translates into large power savings at the server level, our model considers a part of this power conversion unit, since it is not software tunable in its present state. Currently, our model externally monitors the input power to the system and controls the power, and consequently, thermal envelope of the system, based on the processing load.

To understand how our analytical model can be improved by the addition of such sensors (as valuable PeCs), consider how one might partition E_{dc} by having access to this information. Most power supplies limit the total power delivered through the 5V and 3.3V lines to about 20% of the rated power supply (P_R). Assuming that each of the voltage lines in domain j, $v_k^j(t)$, draws current $i_k^j(t)$, then each line draws instantaneous power of $p_k^j(t) = v_k^j(t) \cdot i_k^j(t)$. Given a voltage domain with M_j DC lines as output, the total power delivered for voltage domain j is:

$$p_v^j(t) = \sum_{k=1}^{M_j} v_k^j(t) \cdot i_k^j(t)$$

If the board has N voltage domains: v_1, v_2, ..., v_N, total DC power delivered into the system equals:

$$\begin{aligned} p_{dc}(t) &= \sum_{j=1}^{N} p_v^j(t) \\ &= \sum_{j=1}^{N} \sum_{k=1}^{M_j} v_k^j(t) \cdot i_k^j(t). \end{aligned}$$

Total energy delivered to the system between times t_2 and t_1 is:

$$E_{dc} = \int_{t_1}^{t_2} p_{dc}(t)dt = \int_{t_1}^{t_2} \sum_{j=1}^{N} \sum_{k=1}^{Mj} v_k^j(t) \cdot i_k^j(t)dt.$$

For the 3.3V and 5V lines, we have the following constraint:

$$\begin{aligned} E_{dclv} &= \int_{t_1}^{t_2} p_{dclv}(t)dt \\ &= \int_{t_1}^{t_2} \left(\sum_{k=1}^{M_1} v_k^1(t) \cdot i_k^1(t) + \sum_{k=1}^{M_2} v_k^2(t) \cdot i_k^2(t) \right) dt \\ &\leq 0.2P_R \end{aligned}$$

where M_1 and M_2 are the total 3.3V and 5V lines, respectively. Thus, in our 450W rated system, the power delivered by the 3.3V and 5V lines is capped at 90W.

3.7 Thermal Extensions to the Model

There are two metrics of interest for the thermal workload of a multi-core processor. An application A has a length $L(A, D_A, t)$ which is the total time of execution t of the application working upon a set of data D_A. The application A is composed of p processes, with each process associated with a data set of size d_i, for $1 \leq i \leq p$, in a single logical CPU. The total data associated with an application A is the sum of the data associated with its component processes:

$$D_A = \sum_{i=1}^{p} d_i. \tag{3.5}$$

We assume that the activities are taking place in a staging area which contains the main and virtual memory operating spaces, as well as the processor with its cores and their associated caches and shared cache. This time of execution measurement includes both computation time and the time to move the data for the problem from the staging area (peripherals off the chip like DRAM and HDD) to a computation or operation area (on the chip such as the caches and the cores).

The second metric of interest is the energy consumption or energy workload of an application, $U(A, D_A, t)$. For each application A and problem size D_A, we define the the workload $W(p_i, d_i, t)$, $1 \leq i \leq p$, in a data-operation dependent and system-independent way. The workload W contains two components: (1) the operations count that is performed by the computational core, and (2) the

communication operations required for transfer of data, instructions, and data coherency and book-keeping operations. These are measured in terms of the number of bytes operated upon, or number of bytes transferred. Thus the energy workload of an application A operating on a data set D_A can be expressed as:

$$U(A, D_A, t) = \lim_{n \to k_e} n \times W(p_i, d_i, t) \times L_n(A_n, D_{A_n}, t), 1 \leq i \leq p \tag{3.6}$$

where $L_n(A_n, D_{A_n}, t)$ is the total time to execute n applications using the chip. The term k_e is the total number of applications that can be executed with the associated length of time for L_n, at which point a "thermal event" will occur causing the applications and the system to catastrophically fail, or shut down.

It is easy to see that the above term is energy consumption of the system till a thermal event occurs. In order to relate the energy expenditure of the system while running applications, to the corresponding joule heating, we define the term "Thermal Equivalent of Application" (TEA), which is defined as the electrical work converted to heat in running an application and is measured in terms of die temperature change and ambient temperature change of the system. Thus for the application A we express TEA as :

$$\Theta_A(A, D_A, T, t) = \frac{U(A, D_A, t)}{\lim_{T \to T_{th}} J_e \times (T - T_{nominal})} \tag{3.7}$$

The quantity T_{th} refers to the threshold temperature at which a DTM triggered event will occur. $T_{nominal}$ refers to the nominal temperature as reported by the DTM counters/registers when the system is in a quiescent state, i.e., only the operating system is running and no application is being executed. The term J_e is the "electrical equivalent of heat" for the chip, which reflects the *informational entropy* of the system associated with processing the data bits that application A computes and communicates, as well as the black body thermal properties of the chip packaging and the cooling mechanisms around the chip. Thus, TEA is a dimensionless quantity with both denominator and numerator expressing work done or energy consumed in finishing a task.

For managing the thermal envelope of applications on server systems as well as embedded systems, we are interested in the thermal efficiency of the operation, that is, the thermal cost of taking an application to completion. The thermal efficiency is defined as:

$$\eta(A, D_A, T, t) = \frac{\Theta_A(A, D_A, T, t)}{\Theta_A(A_e, D_{A_e}, T_{me}, L_e)} \tag{3.8}$$

where T_{me} is the maximum temperature which the core will carry over until a DTM triggered event occurs and A_e refers to the application whose energy consumption has caused the DTM triggered event to take place. L_e is the execution time of application A_e. Thus $\eta(A, D_A, T, t)$ is a measure of the "thermal efficiency of the application", which implies how much an application affects temperature change without compromising it's throughput and/or leads to a thermal event. Thus the definition of η is linked to the definition of the thermal and energy workload.

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

$$C_\theta(A, D_A, T, t) = \frac{\Theta_A(A, D_A, T, t)}{E_{sys}(A, D_A, t)} \tag{3.9}$$

where $E_{sys}(A, D_A, t)$ is the overall power consumed during the application lifetime. We can apply Equation 3.1 to Equation 3.9 to apportion the total power consumed by a single physical component (processor, DRAM units, HDD, motherboard, and electrical/electromechanical) during the length L_A of the application.

This normalized quantity C_θ gives some indication of the "cost" of executing an application on the given chip. The problem of application scheduling requires that we find the best application ordering that minimizes:

$$\frac{\partial^2 C_\theta(A, D_A, T, t)}{\partial T \partial t} = \frac{\partial^2}{\partial T \partial t} \left(\Theta_A(A, D_a, T, t) \times C_\theta(A, D_A, T, t)\right). \tag{3.10}$$

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 and temperature changes based on past reading/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.

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

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

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

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

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^{Intel}_{proc}. The next three measures determine E_{mem}, forming MP^{Intel}_{mem}. 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 [Sun Microsystems 2008] and [Intel 2009]). The next two measures are related to E_{hdd} (and constitute MP^{Intel}_{hdd}), 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 $MP^{Intel}(t) = \left[MP_{proc}^{Intel}, MP_{mem}^{Intel}, MP_{hdd}^{Intel}, MP_{board}^{Intel}, MP_{em}^{Intel}\right]^T$.

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, as detailed in Appendix. Such a linear AR approach has characteristics that make it unsuitable for modeling server systems. Consider the traces of actual power shown in Figures 6.3 and 6.4 for the SPEC CPU2006 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 6.4). 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.

4.2 Chaotic prediction

The continuous system expressed in Equation (3.1) can be viewed as a multi-variate differential equation 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 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

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)

(or 0.93 to 0.97) on the AMD (or Intel) test server.

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].

4.2.1 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.

4.2.2 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 4.4: 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 4.4 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. 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 indicated that time series generated from the geometric mean produced the best fit to the collected data.

4.2.3 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 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

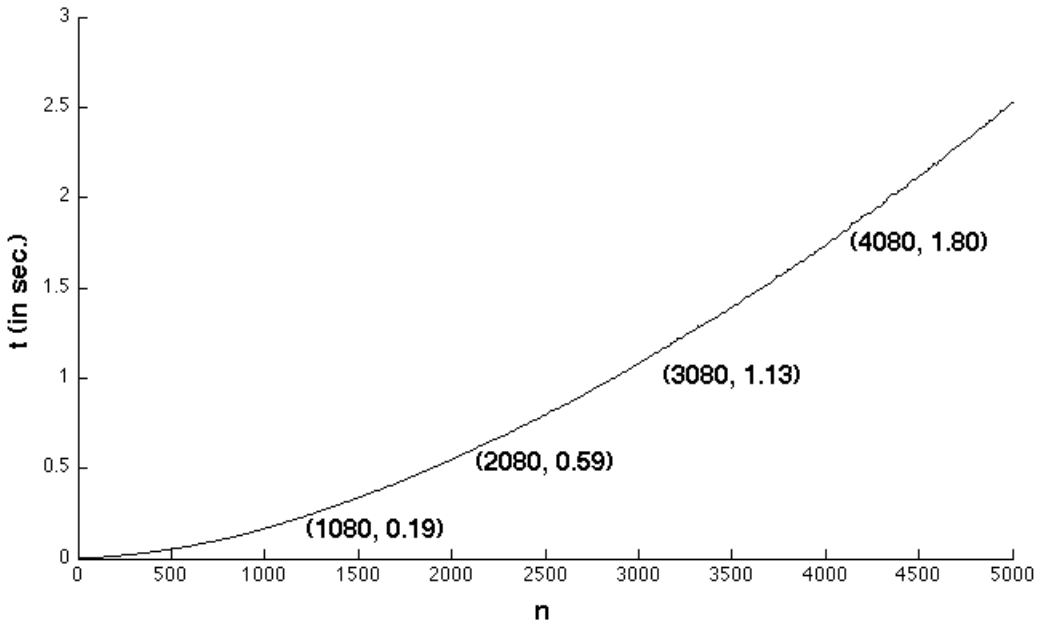


Figure 4.1: CAP time complexity versus no. of future observations.

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 CAT 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 Chapter 6.

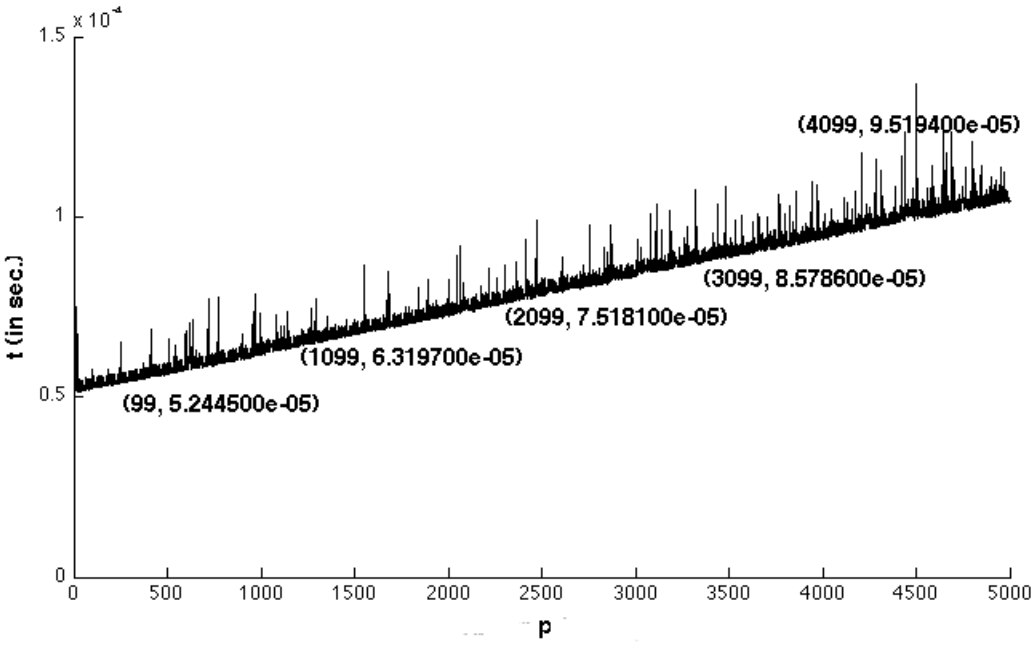


Figure 4.2: CAP time complexity versus no. of past observations.

4.3 Effective Thermal Prediction

CAP is an estimator for the system level energy consumption as expressed in 3.1. In turn, the estimate can be combined with temperature measurements, PeCs, and system metrics to create a Thermal Chaotic Attractor Predictor (TCAP). TCAPs differ from CAPs by their nature as thread level rather than system level predictors. As such, we keep track on a per-thread basis of the observables used in prediction.

Table 4.5: Additional thermal benchmarks

hpl	C	Linear algebra
stream	C	Memory stress testing
aluadd	Assembly	Processor
therm	Assembly	Thermal stress test

We define a TCAP for each of the metrics in Equation 3.7, Equation 3.8, and Equation 3.9 using the procedure defined in Section 4.2.2. For thermal purposes, we construct the training set using the benchmarks in 4.4 + the benchmarks listed in Table 4.5. The additional benchmarks address the issue in the training set of the behavior of the system under extreme thermal stress.

Chapter 5

Thermal-Aware Scheduling

The current generation of operating systems treat the cores in a multicore processor as distinct physical processors. Furthermore, the advent of simultaneous multi-threading (such as Intel's HyperThreading technology) has led to operating systems treating such virtualized processors as equal "logical CPUs" that behave as independent processors for scheduling purposes. However, there are dependencies between these logical CPUs that need to be taken into account when balancing performance and energy efficiency. These dependencies lead to contention for shared resources which, in turn, leads to performance penalties and inefficient use of energy. 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.

Modern multiprocessor operating systems such as Windows, Linux, Solaris, and FreeBSD take a two-level approach to scheduling in an attempt to maximize system resources. The first level manages each core using a distributed run queue model with per core queues and fair scheduling policies. The second level attempts to balance the resource load by redistributing tasks across the queues on each core. The design of such schedulers are based upon three principles: (1) threads are assumed to be independent, (2) load is equated to queue length, and (3) locality is important [Hofmeyr et al. 2010].

Traditional server load balancing makes assumptions about workload behavior to make scheduling decisions. Interactive workloads are characterized by the independent tasks that remain quiet for extended periods. Server workloads contain large numbers of threads that are highly independent of each other that use

synchronization objects to ensure mutual exclusion on small data items. In parallel applications, there is a higher levels of interaction between threads; consequently, load balancing such applications requires additional mechanisms.

Parallel applications are implemented most often on contemporary processors using a Single Process, Multiple Data (SPMD) programming model where logical CPUs simultaneously execute the same program at independent points. Tasks execute independently and communicate with other nodes by sending and receiving messages with some form of barrier synchronization implemented as part of the message interface. The details of the message process is isolated from the application by standard interfaces such as MPI and OpenMP. Note how this programming model contravenes the assumptions for system level load balancing: (1) threads are logically related, (2) have data and control dependencies between threads, and (3) have equally long life-spans [Hofmeyr et al. 2010].

Our scheduler extends the existing dispatcher and power management infrastructure in the operating system. It is the role of the kernel thread scheduler to manage the placement of threads in a dispatch queue, decide which thread to run on a processor, and manage the movement of threads to and from processors so as to balance the workload amongst logical CPUS. A scheduler must fulfill this role while addressing two major requirements for SPMD applications: (1) threads composing an application must make equal progress and (2) the maximum level of hardware parallelism is exploited.

However, these schemes do not take into account the increase in thermal stress placed upon the processor entailed by focusing on maximum performance. Modern processors crudely manage this problem through Dynamic Thermal Management (DTM) where the processor monitors the die temperature and dynamically adjusts the processor voltage and frequency (DVFS) to slow down the processor. Prior work [Bircher and John 2008; Donald and Martonosi 2006; Coskun et al. 2008a] has shown that this technique has significant negative impact on both performance and reliability

in the processor. Thus, pro-active scheduling techniques that avoid thermal emergencies are preferable to reactive hardware techniques such as DTM.

5.1 Thread Selection

The scheduler is responsible in each quantum for deciding which thread to run on a processor. We introduce in this work a heuristic scheduling algorithm that reduces the thermal stress on a multicore processor while addressing the SPMD requirements of equal progress and maximum exploitation of parallelism. The thermal predictor introduced in Chapter 3.7 is used to construct a on-line thermal estimator for temperature, which is then used to predict which thread to next execute on each logical CPU's run queue.

We enhance the existing thread infrastructure to maintain the information required by the thermal estimator. Our implementation is based upon the concept of Task Activity Vectors (TAVs) as introduced by Merkel and Bellosa [2008]. Each task maintains a vector with the required history needed to make a prediction; in effect, we trade the additional space required for keeping this history for the benefits gained from the thermal scheduling. At each scheduling quantum, each TAV is updated with counts from the performance counters representing the hardware resources used in the predictor. The scheduler predicts the impact on the logical CPU temperature using the thermal predictor to map the PeCs values into an estimate of the "thermal efficiency of application" (TEA) (as discussed in 3.7).

The predictions are then used by a policy-based heuristic predictor to decide which thread to execute given the TEA of each candidate thread. The operating system kernel is enhanced to allow the policy to be selected as a kernel configuration option. We examine the behavior of our predictor against three well known thermal scheduling heuristics: (1) Greedy, (2) MinTemp [Kursun et al. 2006], and (3) ThreshHot [Zhou et al. 2010]. The Greedy heuristic selects the coolest job to execute in each scheduling epoch, MinTemp selects the coolest possible thread if over a set threshold, and ThreshHot selects the hottest thread that is predicted to not cause the temperature to

exceed that set threshold or the hottest job in the queue otherwise.

5.2 Load Balancing

Load balancing distributes workload evenly across the available logical CPUs; current implementations distribute to maximize performance, we enhance the concept to minimize thermal stress while seeking best performance. We begin by organizing the logical CPUs in the system into three categories based upon the temperature at which a DTM event occurs: Hot (90% of DTM temperature), warm (between 75% and 90% of DTM temperature), and cold (less than 75% of the DTM temperature). In addition, we classify each logical CPU into "clans" depending whether a logical CPU is considered a "fast processor" or "slow processor" depending upon the current processor frequency of the underlying hardware. This two-level categorization allows us to manage the distribution of work such that we can migrate work away from heat sources while minimizing the impact on performance of threads.

The movement of logical CPUs between processor groups as temperature increases is managed by a Thermal Predictor. The Thermal Predictor collect the values of the PeCs, temperature readings from the logical CPUS, and utilization data from the dispatcher to decide whether a processor will move towards the DTM temperature. In this way we can anticipate the DTM event and prevent its occurrence.

The dispatcher is responsible for deciding where to insert threads into run queues; i.e., when and where a thread will next execute. The steps in this process are illustrated in Figure 5.1. First, the dispatcher checks to see if the logical CPU where the thread was running satisfies the cache affinity criteria (the default for OpenSolaris is the thread was last running within the past 3 cycles). If that test fails, it tries to find the logical CPU in the local group running the lowest priority thread. If no thread is found in the local group, then the dispatcher will check through known remote groups. If no logical CPU is found in either case, it will choose a logical CPU in the local group.

At this point, the system will attempt to load balance the workload. Two possibilities exist in the current system: equal balance across all physical processors in

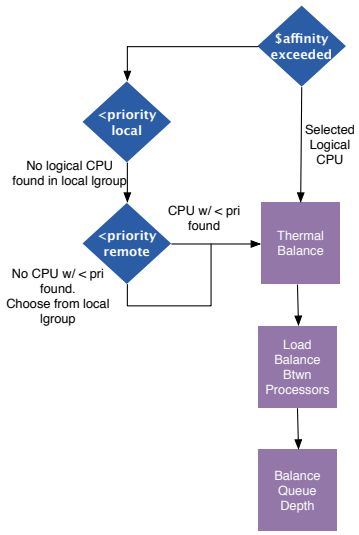


Figure 5.1: Thread Queue Insertion

the system and coalesce work onto one chip or the other. The first case is traditional example of load balancing while the second case is attempting to free up as many resources as possible to provide opportunities for power management to find logical units to shut down. We add a third sort of load balancing: temperature balance. In this case, the dispatcher will attempt to load logical CPUs in the "COLD" processor group first, followed by the "WARM" processor group, and finally the "HOT" group.

5.2.1 Algorithm

Our Thermal-Aware Balancer (TAB) uses two classes of threads: a global Thermal Predictor Thread (TPT) and multiple Thermal Balancing Threads (TBT), one TBT per logical CPU. The Thermal Predictor Thread is a temperature monitor that watches the temperature of the logical CPU and adjusts the HOT, WARM, COLD lists on a periodic basis. The TBTs on each logical CPU performs a mix of speed and thermal balancing [Hofmeyr et al. 2010] on queues assigned to each logical CPU.

The TPT awakes periodically and enumerates the known logical CPUs to rearrange the HOT, WARM, COLD list. The uses the thermal predictor defined in the previous chapter to evaluate the past behavior of each logical CPU and adds/deletes this unit from the appropriate lists.

Listing 5.1: TAB Balancing Algorithm

```
begin
Determine if logical CPU is HOT, WARM, or COLD
if logical cpu is HOT
begin
   for each thread in the run queue
     begin
         Project the resulting change in temperature if this thread
         executes
         Compute the projected logical CPU speed over the elapsed
            balance
         interval
     end
   Compute the global core speed as average over all logical CPUs
   Migrate the thread with the ‘‘worst’’ impact on temperature to the
   logical CPU in the COLD set most suitable from a speed standpoint.
end
```

Periodically, the TBT on each core wakes up, check for imbalances on the on the core and pulls threads from the hotter cores to local cores and return to sleep. Note that this is a distributed algorithm where each TBT operates independently and without any global synchronization. A TBT executes the code in Listing 5.1 when it awakes in order to adjust the thermal balance of the queues in the designated logical CPU.

Chapter 6

Initial Evaluation and Results

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 6.1. 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 4.1, 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 4.2, 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 $n = 5$).

Table 6.1: 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

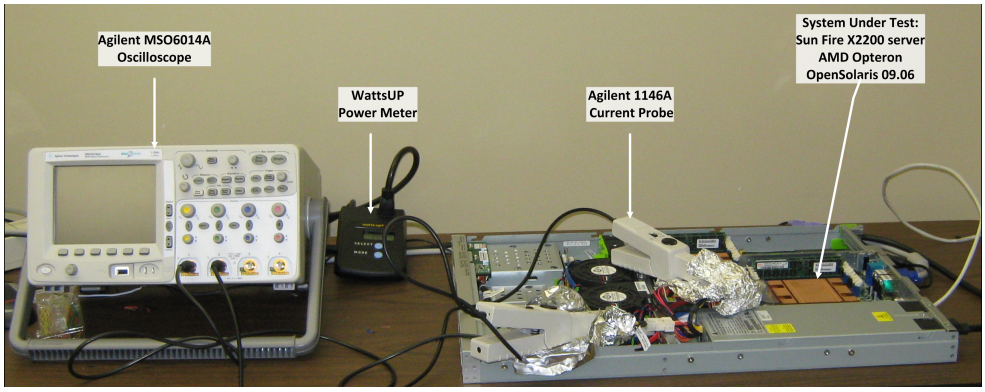


Figure 6.1: Hardware test setup.

Table 6.2: 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

6.1 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 evaluation purpose, as listed in Table 6.2), and they are different from those employed earlier for CAP creation (as shown in Table V). They were executed on the two servers specified in Table 6.1, 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 (SUT). The power meter

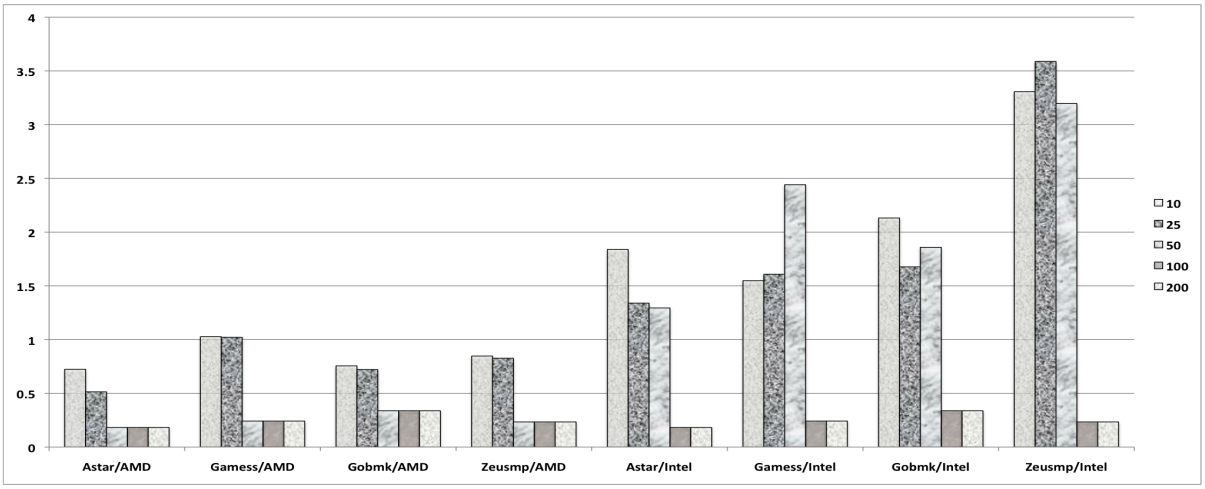


Figure 6.2: Root Mean Square Error (RMSE) for different values of p.

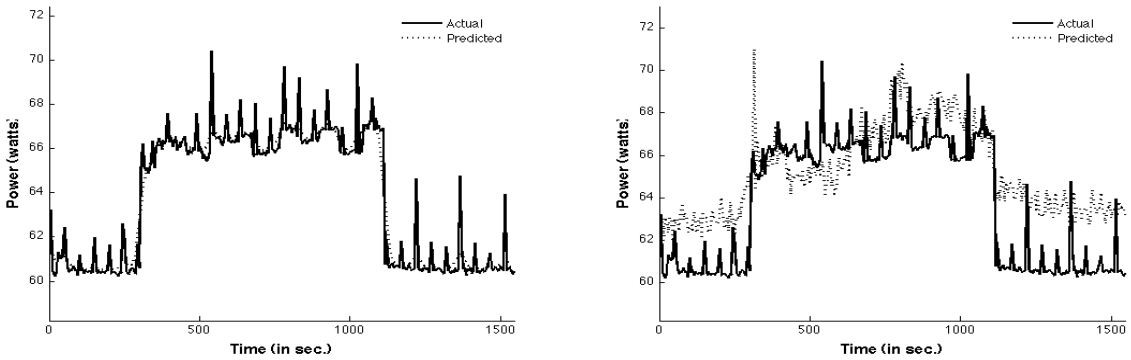
measures the total and average wattage, voltage, and amperage over the run of a workload. The internal memory of the power meter is cleared at the start of the run and the measures collected during the runs are downloaded (after execution completion) from meter's internal memory into a spreadsheet. Current 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.

Table 6.3: Model errors for CAP, AR(1), and MARS on AMD Opteron server

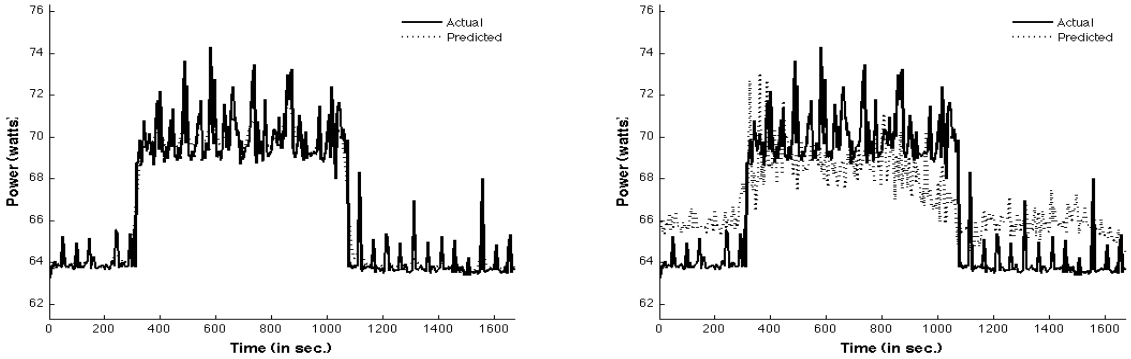
	CAP ($n = 5$, $p = 100$, $r = 14$)			**AR**			**MARS**		
Benchmark	**Avg Err %**	**Max Err %**	**RMSE**	**Avg Err %**	**Max Err %**	**RMSE**	**Avg Err %**	**Max Err %**	**RMSE**
Astar	0.9%	5.5%	0.72	3.1%	8.9%	2.26	2.5%	9.3%	2.12
Games	1.0%	6.8%	2.06	2.2%	9.3%	2.06	3.0%	9.7%	2.44
Gobmk	1.6%	5.9%	2.30	1.7%	9.0%	2.30	3.0%	9.1%	2.36
Zeusmp	1.0%	5.6%	2.14	2.8%	8.1%	2.14	2.8%	7.9%	2.34

6.2 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



(a) Astar/CAP. (b) Astar/AR(1)).

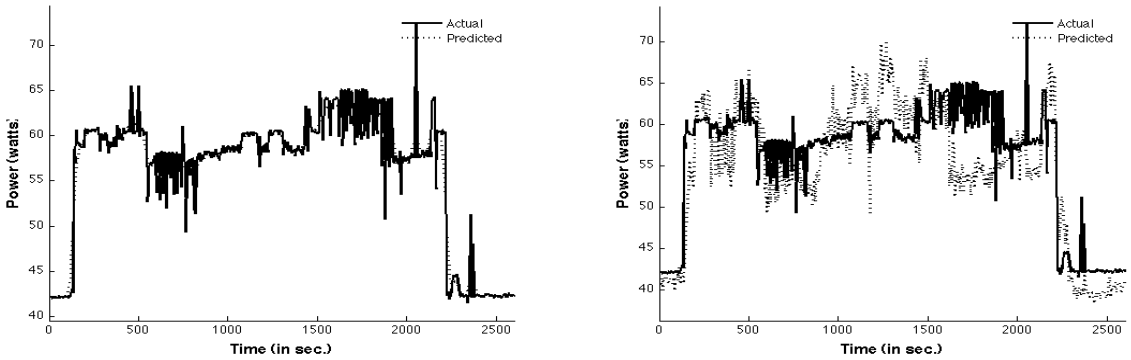


(c) Zeusmp/CAP. (d) Zeusmp/AR(1).

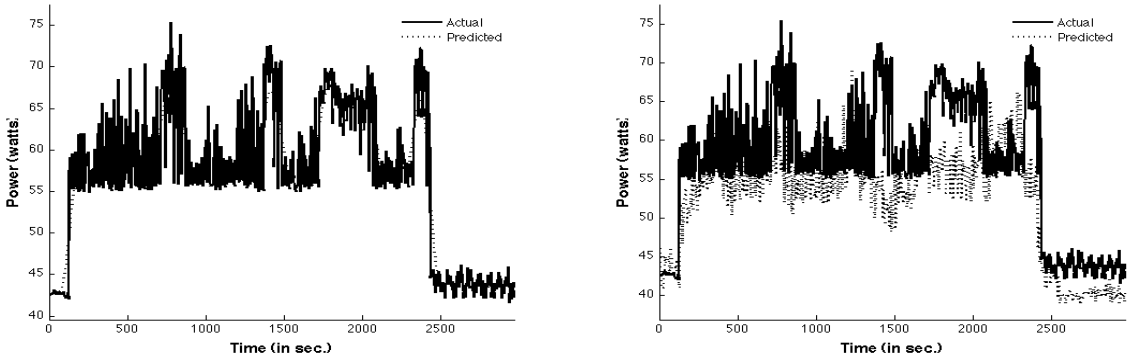
Figure 6.3: Actual power results versus predicted results for AMD Opteron.

observations) and n (the number of past observations), as stated in Section 4.2.2. 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 6.2, 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 $p = 100$.

The predicted power consumption results of CAP during the execution of Astar and Zeusmp on a HyperTransport-based server are demonstrated in Figures 6.3(a) and 6.3(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



(a) Astar/CAP. (b) Astar/AR(1)).



(c) Zeusmp/CAP. (d) Zeusmp/AR(1).

Figure 6.4: Actual power results versus predicted results for an Intel Nehalem server.

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 6.3(b) and 6.3(d), where details of AR(1) can be found in Appendix. 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. 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 6.4(a) and 6.4(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 6.4(b) and 6.4(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 6.4: Model errors for CAP, AR, and MARS on Intel Nehalem server

	CAP ($n = 5$, $p = 100$, $r = 19$)			**AR**			**MARS**		
Benchmark	**Avg Err %**	**Max Err %**	**RMSE**	**Avg Err %**	**Max Err %**	**RMSE**	**Avg Err %**	**Max Err %**	**RMSE**
Astar	1.1%	20.8%	1.83	5.9%	28.5%	4.94	5.4%	28.0%	4.97
Games	1.0%	14.8%	1.54	5.6%	44.3%	5.54	4.7%	33.0%	4.58
Gobmk	1.0%	21.5%	2.13	5.3%	27.8%	4.83	4.1%	27.9%	4.73
Zeusmp	3.3%	20.6%	3.31	7.7%	31.8%	7.24	11.6%	32.2%	8.91

6.3 Further discussion

Tables 6.3 (or 6.4) compares the errors of evaluation benchmarks for the server with the HyperTransport (or QPL) structure, under three different prediction mechanisms: CAP, AR, and MARS. Details of AR and MARS predictors can be found in Appendix. Large errors exhibited by AR and MARS 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. Earlier attempts were made to address this issue by incorporating corrective mechanisms in such a linear predictor. An example attempt employed machine learning to monitor for 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 drifts in prediction experienced by other prediction techniques.

The model developed in this paper 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.

The experimental validation of CAP reveals opportunities for further investigation. 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.

Chapter 7

Current State, Plan for Completion, and Future Plans

We presented in Lewis et al. [2008; Lewis et al. [2010; Lewis et al. [2011] a full-system power and thermal mode that provides a run-time system-wide prediction of energy consumption on server blades as a continuous system of differential equations. Using chaotic time series, we construct discrete approximations of the solution of this system so as to provide a full system approximation to the thermal behavior of the processor. In this prospectus, we propose to use this predictor to implement a prototype of the scheduler described in Chapter 5 that is predicted to have a measurable reduction in energy consumption and thermal stress as compared to existing thread schedulers.

7.1 Plan for Completion

Table 7.1 lists the tasks required to complete this work. Task #1 addresses any required changes that arise from the peer review of [Lewis et al. 2011].

A prototype for the scheduler described in Chapter 5 will be implemented for the FreeBSD operating system [McKusick and Neville-Neil 2004]. We use FreeBSD for two reasons: (1) the scheduler architecture in FreeBSD is well documented and can be simply modified compared to other choices (such as Linux or OpenSolaris), and (2) the infrastructure for managing PMCs in FreeBSD has many of the advantages of OpenSolaris but is not tied to a specific kernel version as is the Linux infrastructure.

Table 7.1: Completion Task List

Id	Task
1	Respond to review comments for [Lewis et al. 2011].
2	Implement scheduler prototype in FreeBSD.
3	Evaluate scheduler performance using parallel benchmarks.
4	Document results and submit to archival journal.
5	Create dissertation from this document & output from Task 4.
6	Defend dissertation.
7	Respond to comments from committee and Graduate School editor.
8	Submit final version of document.

The performance of our prototype is considered in Task #3. The baseline performance of the scheduler will be determined by examining the system performance, thermal behavior, and energy consumption of three standard benchmarks suites: (1) SPEC CPU2006 [Henning 2006], (2) High Performance Linpack(HPL) [Dongarra 1989], and (3) the Stream memory benchmark [McCalpin 1995].

7.2 Future Directions

The topics of power and thermal management in large scale computing have only recently begun to receive systematic attention. As such, we anticipate that the work in this Prospectus will serve as a starting point for a longer-term research program. Examples of longer-term questions that will need to be addressed include:

- 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 TCAP and Thermal-Aware Scheduling into such environments.
- A related topic for the future involves 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.
- An interesting extension of the system model described in 3 involves considering whether we can quantify the energy expended per bit transfer as data moves through the system. From this, we can build a more detailed analytical model of the thermodynamics of the system than the current practice of RC-thermal modeling [Skadron et al. 2004].

7.3 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 paper, 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.

We extend the CAP concept to the thermal domain through the concept of Thermal Chaotic Attractor Prediction (TCAP). Our system model introduced three metrics that model the system thermal behavior: Thermal Equivalent of Application, Thermal Efficiency, and Thermal Cost. TCAP is used to approximate future values of each of these quantities given the energy consumption predicted by CAP. The thermal predictions are used as optimization metrics in our Thermal-Aware Scheduler for thread scheduling and load balancing purposes. Our proposed model is the first step towards building solutions for power and thermal management in data centers usually housing many servers.

Bibliography

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.

Bellosa, F., Weissel, A., Waitz, M., and Kellner, S. 2003. Event-driven Energy Accounting for Dynamic Thermal Management. *Proc. of the 2003 Workshop on Compilers and Operating Systems for Low Power*.

Bhattacharjee, A. and Martonosi, M. 2009. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. *Proc.of the 36th Int'l. Symp. on Computer Architecture*, 290–301.

Bircher, W. and John, L. 2007. Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events. *Ispass*, 158–168.

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 *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*. ACM, New York, NY, USA, 327–338.

Bowman, A. and Azzalini, A. 1997. *Applied Smoothing Techniques for Data Analysiks: 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.

Contreras, G. and Martonosi, M. 2005. Power Prediction for Intel XScale\textregistered˜Processors Using Performance Monitoring Unit Events. In *Proc. of the 2005 Int'l Symp. 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. *Proc. of the 2008 IEEE/ACM Int'l. Conf. on Computer-Aided Design*, 250–257.

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

Coskun, A. K., Rosing, T. S., Whisnant, K. A., and Gross, K. C. 2008b. Temperature-aware MPSoC scheduling for reducing hot spots and gradients. *Proc. of the 2008 Asia and South Pacific Design Automation Conference*, 49–54.

Donald, J. and Martonosi, M. 2006. Techniques for Multicore Thermal Management: Classification and New Exploration. In *Proc. of the 33rd Int'l Symp. on Computer Architecture*. IEEE Computer Society, Washington, DC, USA, 78–88.

Dongarra, J. 1989. Performance of various computers using standard linear equations software. Tech. Rep. CS-89-85, The University of Tennessee.

Economou, D., Rivoire, S., Kozyrakis, C., and Ranganathan, P. 2006. Full-System Power Analysis and Modeling for Server Environments. In *Proc. 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 *Proc. of the 34th Int'l Symp. 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.

Heath, T., Diniz, B., Carrera, E. V., Jr., W. M., and Bianchini, R. 2005. Energy Conservation in Heterogeneous Server Clusters. In *Proc. of the 10th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming.* 186–195.

Henning, J. L. 2006. SPEC CPU2006 benchmark descriptions. *Computer Architecture News 34,* 4 (Sept).

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

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. Phase Characterization for Power: Evaluating Control-flow-based and Event-counter-based Techniques. *Proc. of the 12th Int'l Symp. on High-Performance Computer Architecture*, 121–132.

Isci, C. and Martonosi, M. 27 Oct. 2003a. Identifying Program Power Phase Behavior Using Power Vectors. *Proc. of the IEEE 2003 Int'l Workshop on Workload Characterization*, 108–118.

Isci, C. and Martonosi, M. 3-5 Dec. 2003b. Runtime Power Monitoring in High-end Processors: Methodology and Empirical Data. *Proc. of the 36th IEEE/ACM Int'l Symp. 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.

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.

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., Cher, C.-y., Buyuktosunoglu, A., and Bose, P. 2006. Investigating the Effects of Task Scheduling on Thermal Behavior. *Proceedings of the 3rd Workshop on Temperature-Aware Computer Systems*.

Lee, K.-J. and Skadron, K. 2005. Using performance counters for runtime temperature sensing in high-performance processors. *Proc. of the 19th IEEE Int'l Symp. 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 conference on Power aware computing and systems*, 4–4.

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

Lewis, A., Tzeng, N.-F., and Ghosh, S. 2011. Time series approximation of run-time energy consumption based on server workload. Under review for publication in ACM Transactions on Architecture and Code Optimization.

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.

Liu, Z. 2010. Chaotic time series analysis. *Mathematical Problems in Engineering 2010*, 31.

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*.

Maxim. 2008. Practical Considerations for Advanced Current Sensing in High-Reliability Systems.

McCalpin, J. D. 1995. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, 19–25.

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

Merkel, A. and Bellosa, F. 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. 2008. 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.

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

Rangan, K. K., Wei, G.-Y., and Brooks, D. 2009. Thread motion: fine-grained power management for multi-core systems. *SIGARCH Comput. Archit. News 37,* 3, 302–313.

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

Rivoire, S., Ranganathan, P., and Kozyrakis, C. 2008. A Comparison of High Level Full-System Power Models. In *Proc. of 2008 USENIX Workshop on Power Aware Computing and Systems.*

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

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

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

Ton, M., Fortenbery, B., and Tschudi, W. 2008. DC Power for Improved Data Center Efficiency.

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

Turlach, B. 1993. Bandwidth selection in kernel density estimation: A review. *CORE and Institut de Statistique*, 23–493.

Yang, J., Zhou, X., Chrobak, M., Zhang, Y., and Jin, L. 2008. Dynamic Thermal Management through Task Scheduling. *Proc. of the 2008 IEEE Int'l Symp. on Performance Analysis of Systems and Software*, 191–201.

Yao, F., Demers, A., and Shenker, S. 1995. A scheduling model for reduced CPU energy. *Proc. of the 36th IEEE Symp. 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.