Thermal~~ly~~-aware Scheduling and Load Balancing Using Predictive Energy Consumption Models in High-Performance Systems

Adam Lewis and Nian-Feng Tzeng

Center for Advanced Computer Studies, University of Louisiana, Lafayette, Louisiana 70504
{awlewis,tzeng}@cacs.louisiana.edu

***Abstract*—Modern processors crudely manage thermal emergencies through Dynamic Thermal Management (DTM) where the processor monitors the die temperature and dynamically adjusts the processor voltage and frequency (DVFS) to slow down the processor. However, DVFS has a significant negative impact on application performance and system reliability. Thus, proactive scheduling techniques that avoid thermal emergencies are preferable to reactive hardware techniques such as DTM.**

~~Our analysis of experimental measurements of key processor performance counter readings and performance metrics reveals that the measured readings and metrics do not possess linearity and are *chaotic in nature*. Thus,~~ our ~~development of~~ thermal Chaotic Attractor Predictors (CAPs) that take 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.~~

~~The component of the operating system most aware of the existence of multiple cores is the thread scheduler.~~ ~~We use~~ CAPs ~~to create two scheduling techniques for thermal management that~~ minimize 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.~~ ~~This is demonstrated by adding thermal-awareness to~~ the existing scheduler in the FreeBSD operating system ~~executed on an Intel Xeon processor. Subsets of the SPEC CPU2006 and PARSEC benchmark suites are used to simulate application server workloads and evaluate scheduler performance.~~

I. Introduction

Modern processors crudely manage thermal emergencies through Dynamic Thermal Management (DTM) where the processor monitors the die temperature and dynamically adjusts the processor voltage and frequency to slow down the processor. However, the use of Dynamic Voltage and Frequency Scaling (DVFS) can cause significant negative impact upon application performance and system reliability [1], [2], [3]. ~~Thus, pro-active scheduling techniques that avoid thermal emergencies are preferable to reactive hardware techniques.~~

This paper introduces scheduling techniques for preventive thermal management that minimizes server energy consumption by (1) ~~for each logical processor,~~ selecting the next thread to execute based upon an estimate of which thread in the next quantum, and (2) ~~adjusting the load balance allocation of threads to available logical processors to migrate workload away from thermally overextended resources on the processor.~~ As opposed to prior work on thermal scheduling [4], [5], [6], [7] that focused on bounding temperatures below critical thresholds, our work considers how ~~we can~~ schedule high utilization workloads ~~so as to manage average temperatures~~ via balancing ~~of~~ thermal load across logical processors.

The current generation of operating systems ~~treat~~ the cores in multicore ~~and~~ virtualized multi-threaded processors (such as Intel's HyperThreading technology) as distinct logical processors that ~~behave as~~ independent devices for ~~scheduling purposes.~~ However, ~~there are dependencies~~ and contention ~~for~~ shared resources ~~between these~~ logical processors that need to be taken into account when ~~balancing~~ performance and energy efficiency. One approach to solving this problem is software optimization at the application level to make ~~applications~~ more aware of ~~these~~ dependencies [8]. This approach, while effective, may not be applicable in all cases and does not provide economies of scale. Thus, the need for intelligent thermal~~ly~~-aware load balancing and scheduling within the operating system. The provision of such capabilities requires ~~a full system model of~~ energy consumption based on computational load ~~of system that can effectively predict~~ future energy consumption ~~and~~ resulting changes in thermal load.

Our thermal model relates server energy consumption to the overall thermal envelope, establishing an energy relationship between the workload and the overall thermodynamics of the system. Our analysis of experimental measurements of key processor performance counter readings and performance metrics reveals that the measured readings and metrics do not possess linearity and are *chaotic in nature*. Thus, our development of a thermal Chaotic Attractor Predictor (tCAP) that 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. Effective scheduling can result from taking advantage of the proposed tCAP when dispatching jobs to confine server power consumption within a given power budget and thermal envelope while ~~minimizing~~ impact upon server performance.

~~The component of the operating system~~ most aware of the existence of multiple cores ~~is the~~ thread scheduling ~~algorithm.~~ ~~The scheduler is responsible for ensuring~~ that all of the cores across ~~all~~ processors in a system ~~is~~ kept busy~~; the scheduler must be able to efficiently migrate the state of~~ threads ~~across processors regardless if that migration remains on a core or across processors;~~ Schedulers in the current generation of operating systems aid in processor thermal management ~~by~~ workloads to logical processors as tightly as possible ~~allow~~ provide more opportunities for power management to shutdown unused resources. However, ~~compute bound~~ server workloads fully utilize available system ~~resources,~~ rendering ~~these schemes~~ ineffectual. Our Thermally-Aware Scheduler (TAS) addresses this problem by ~~attempting to~~ thermally ~~balance~~ the ~~system~~ with as ~~little~~ impact on performance as possible. This is demonstrated by adding thermal-awareness to the existing scheduler in the FreeBSD operating system executed on an Intel Xeon (Woodcrest) processor. A subset of the SPEC CPU2006 benchmark suite ~~is used to simulate~~ application server workloads ~~and~~ evaluate ~~scheduler performance~~.

After an overview of pertinent background and related work is provided in ~~SectionII,~~ this article introduces our proposed thermal model in Section III ~~and explains in Section III-A how the model can be used for predictive purposes~~. Section IV details the design of our ~~Thermally Aware Scheduler.~~ ~~The behavior of our scheduler~~ is experimentally evaluated in Section V~~. experimental results.~~ Section VI offers our ~~conclusions.~~

II. Background and Related Work

~~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 processors. A scheduler must fulfill this role while addressing two major applications requirements: (1) ~~threads composing an application must make equal progress and (2) the maximum level of hardware parallelism is exploited~~ [9]. Traditional server load balancing makes assumptions about workload behavior when making ~~decisions about thread placement.~~ 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. Modern operating systems (such as Linux and Solaris) attempt to make load balancing more power-aware by compacting workloads as tightly as possible onto available logical processors ~~to provides~~ more opportunities for power management software to shutdown unused resources [10], [11], [12].

In the current generation of microprocessors and operating systems, power management software that utilizes DVFS techniques to address DTM events has been shown ~~to be an effective method of~~ addressing thermal emergencies [1], [13] and has been implemented in modern server processors [14], [15]. However, addressing DTM events through DVFS can be problematic due to issues with program phase behavior and contention for shared resources [2], [3]. Compute-bound workloads are ~~impacted~~ due to slow transitions between active and idle state devices while memory-bound workloads are ~~effected~~ by the need for active threads to access resources ~~that are bound to~~ idle processors. Furthermore, as the number of power phase changes occur due to frequency changes, the increased thermal cycling within the processor results in large thermal variations amongst cores with a resulting decrease in reliability [16], [3], [17].

~~The idea of migration of work~~ for energy savings and thermal management has a long history in the SMP, SMT, and CMP environments [18], [4], [19], [6]. A study of OS-level thermal migration using Linux on the IBM POWER5 processor [5] ~~determined~~ that the rise and fall times of core temperatures vary in the order of hundreds of milliseconds. As most operating ~~system schedulers operate with~~ scheduler ticks ~~of~~ ~~10ms~~ or less, ~~then~~ it is ~~possible~~ to react to thermal conditions before they reach a critical state. As ~~result,~~ three means for improvement exist for managing thermal ~~state;~~ (1) core hopping for leveraging spatial heat slack, (2) task scheduling for leveraging temporal heat slack, and (3) SMT scheduling for leveraging temporal heat slack. In each of these cases, it has been shown [5],[20] that these methods can improve reduce core die temperatures an average of 3 to 5° Celsius with an average of 3% degradation in performance. Key in each of these cases is the availability of slack, either in time with scheduling of threads or in space with scheduling of resources that can be conserved to better manage the energy or thermal profiles. However, the available slack in deadlines decreases as the system load increases and contention for resources becomes more common.

A common theme amongst these works is fitting a mathematical model to time-series observations of temperatures arising from execution of workloads. Model coefficients are estimated to give total least square errors between model results and actual temperatures. Calibrated models can be used to extrapolate future temperatures. Different modeling techniques have been applied in attempts to solve this problem. In Coskun *et al.* [21], 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 [4],HybDTM [19], and ThreshHot [6], [22]. 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, favoring those threads that produced the greatest increase in temperature without causing a DTM event to occur. In all three cases, scheduling decisions are made utilizing data from hardware performance counters and hardware temperature sensors. These methods can be enhanced by through the use on the analysis of on-die thermal variation to assist system power and thermal management [23], [17], [24].

However, these techniques are reacting to the temperature approaching the DTM threshold rather than trying to avoid reaching that temperature in the first place. A multi-tier solution to this issue is suggested in Ayoub *et. al*[25] that employed a core level predictor to convert temperature observations to the frequency domain for predictive purposes while at the socket level used a control-theoretic based scheduler for process level scheduling. In Merkel `et.al.` [26], [27], 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. In [28], 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 similar approach is the use of idle cycle injection to manage CPU time slice allocation in an attempt to maintain a lower average temperature over time as opposed to managing temperatures against a critical threshold. Cool Loop [5] and Dimentrodon [29] address a lack of heat slack by inserting additional cycles into the task scheduling to create additional thermal slack, with a resulting negative impact on performance. A variation on this scheme was proposed in [30] 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. These approaches are limited for the many server use cases where the available slack in deadlines does not exist.

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 is based upon three principles: (1) threads are assumed to be independent, (2) load is equated to queue length, and (3) locality is important [9]. Traditional server load balancing makes assumptions about workload behavior when making decisions about thread placement. 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. 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 lifespans [9].

For example, the FreeBSD ULE scheduler [31], [32], [33] uses a combination of push and pull thread migration for load balancing. *Push migration* scans the run queues associated with each logical processor every 500 milliseconds to pick the most-loaded and least-loaded logical processors and equalizes their run-queues. In the SMT case, a graph of *processor groups* is used to define the physical arrangement of the logical processor units in the system. When doing push migration, the processor uses this structure to first attempt to pick a core in the same processor group (and,thus, in the same physical package) so as to minimize migration cost. In *pull migration*, logical processors will mark a bit in a processor group shared bitmask when that processor has not work in any of its queues. When a logical processor wants to add new work to its run queue, it checks to see if it has excess work and whether or not another processor in the system is idle. If so, it uses an interprocessor interrupt to migrate this thread to the idle processor.

Other operating systems (such as Linux and Solaris) implement load balancing in a manner similar to FreeBSD's pull migration. In addition, these operating systems have attempted to make their load balancing schemes more power-aware by adding linkages between their schedulers and power management drivers that attempt to find as compact an allocation of processes to run-queues as possible [10], [11], [12], [7]. In doing so, this provides more opportunities for the power management software to shutdown logical processors and shared resources. Such schemes, while effective for many interactive workloads, assume that the system utilization is not completely allocated and that resources can be taken off-line, which may not be viable when the system resources are fully utilized.

III. System Thermal Model

An analytical model of server energy consumption was built earlier [34], [35] by modeling energy consumption as a function of the work done by the system in executing its computational tasks and of residual thermal energy given off by the system in doing that work. The resulting dynamic system consumes energy expressed in the time domain as follows:

$$\begin{aligned} E_{system} &= \frac{dP_{system}}{dt} \\ &= f(E_{proc}, E_{mem}, E_{em}, E_{board}, E_{hdd}) \end{aligned} \qquad (1)$$

where each of the terms in the above equation is defined as: (1) E_{proc}: energy consumed in the processor due to computations, (2) E_{mem}: energy consumed in the DDR SDRAM chips, (3) E_{em}: energy taken by the electromechanical components in the system, (4) E_{board}: energy consumed by peripherals that support the operation of the board,x and (5) E_{hdd}: energy consumed by the hard disk drive during the system's operation.

Starting from Equation 1, we can derive metrics for the thermal workload of an application executing on a multi-core processor. The length $L(A, D_A, t)$ of an application A is the total time of execution t of the application working upon a set of data D_A. An application is composed of p threads of execution, with each thread associated with a data set of size d_i, for $1 \leq i \leq p$, in a single logical processor. The total data associated with an application A is the sum of the data associated with its component thread:

$$D_A = \sum_{i=1}^{p} d_i. \qquad (2)$$

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).

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

$$E_A(A, D_A, t) = \lim_{n \to k_e} n(p_i, d_i, t) L_n(A_n, D_{A_n}, t), \quad (3)$$

for $1 \leq i \leq p$. 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.

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:

$$\Theta_A(A, D_A, T, t) = \frac{E_A(A, D_A, t)}{\lim\limits_{T \to T_{th}} J_e(T - T_{nominal})}. \quad (4)$$

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.

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

$$C_\theta(A, D_A, T, t) = \frac{\Theta_A(A, D_A, T, t)}{E_{sys}(A, D_A, t)} \quad (5)$$

where $E_{sys}(A, D_A, t)$ is the overall power consumed during the application lifetime. This normalized quantity indicates the "cost" of executing an application on the given chip. We can 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 by applying Equation 1 to Equation 5.

A. Thermal Chaotic Attractor Predictors

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. Therefore, effective prediction of future thermal events based on past power consumption readings/measurements (obtained from hardware performance counters (PeCs) and performance metrics) is critical.

Following the process defined in Lewis `et al.` [35], we define time series representations for the thermal equivalent of application θ and thermal cost C_θ to create an energy consumption solution for this system by considering (1) an initial energy state E_{system} at time $t = 0$ and (2) a set of physical predictors that approximate the values of E_{proc}, E_{mem}, E_{board}, and E_{hdd} at the next interval $t+\Delta t$. The result is a time series

$$\hat{E}_{sys}(t) = \hat{f}(e\_proc_t, e\_mem_t, e\_em_t, e\_board_t, e\_hdd_t) \quad (6)$$

where each of quantities e corresponds to one or more physically observable predictors of the quantities in Eq. (1). The function $\hat{f}$ captures the method used to combine the physical observations into $\hat{E}_s ys$. We place two requirements upon $\hat{f}$: (1) it must quickly compute estimates to be suitable for real-time prediction of energy and temperature changes, and (2) it must approximate the behavior of the original function f to an acceptable accuracy. Observations from each time series

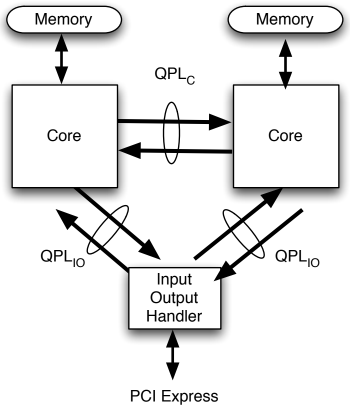


Fig. 1: Intel Xeon (Woodcrest) architecture.

are measured using (1) the appropriate PeCs for the targeted processor (2) operating system kernel virtual memory statistics, and (3) processor die temperatures and chassis ambient temperatures. Seventeen (24) server measures are input to our predictor, as listed in Table I. They are classified into five groups, each associated with one server energy contributor. Notice that QPL_C and QPL_{IO} are relevant to QuickPath Links (depicted in Fig. 1), 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 I, the top three are pertinent to E_{proc}, comprising MP_{proc}. The next three measures determine E_{mem}, forming MP_{mem}. Those two

TABLE I: PeCs and performance metrics for Intel Xeon server

Variable	Measurement
T_{C_0}	Core 0 Die Temp
T_{C_1}	Core 1 Die Temp
T_{C_2}	Core 2 Die Temp
T_{C_3}	Core 3 Die Temp
QPL_C	Transactions on the between Cores
QPL_{IO}	Transactions on QPLs for IO Handler
CM_0	Last-level Cache Misses due to Core 0
CM_1	Last-level Cache Misses due to Core 1
CM_2	Last-level Cache Misses due to Core 2
CM_3	Last-level Cache Misses due to Core 3
D_r	Disk bytes read
D_w	Disk bytes written
T_{A_0}	Ambient Temp 0
T_{A_1}	Ambient Temp 1
T_{A_2}	Ambient Temp 2
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
IR	Instructions Retired

CM_i measures indicate the total L3 cache miss counts due to Core i, i = 0,1,2, or 3. 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. The next two measures are related to E_{hdd} (and constitute MP_{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}. Finally, the last nine measures determine E_{em}, offering speed information of those nine memory cooling fans, to constitute MP_{em}. As a result, each observation for the Intel server at time t comprises the 24 measures of $MP(t) = [MP_{proc}, MP_{mem}, MP_{hdd}, MP_{board}, MP_{em}]^T$. Application length is estimated in each time period by the quantity IR, the total number of instructions retired by the processor for this process.

1) Chaotic Predictors: The continuous systems in Equations 4 and 5 can be viewed as multi-variate differential equations in the time domain. The time series approximation of a system solution for these equations can be a viewed as a projection of the flow of each equation onto a surface[36]. This projection is defined so that the behavior of the dynamic system is reflected in our discrete approximation. The functions $\hat{f}$, $\hat{C}$, and $\hat{\theta_A}$ are expressed in terms of a thermal Chaotic Attractor Predictor (tCAP), following the approximation method proposed earlier for energy consumption energy consumption [35]. Each tCAP is a linear least squares regression of a multivariate local polynomial of degree r.

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 [37]. After analyzing our experimental data, we believe that the temperature measurements and associated thermal metrics demonstrate *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 [37]. The average Lyapunov exponent can be calculated using $\lambda = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} ln|f'(X_n)|$.

TABLE II: Chaotic behavior in core temperature time series

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

We found a positive Lyapunov exponent when performing this calculation on our data set, ranging from 0.019 to 0.051 on our Intel test server, as listed in Table II. 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 (6) 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) [37]. 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 II, the time series data collected in our experiments all have values of H close to 1.0, ranging from 0.95 to 0.99 for the Intel server in our test environment.

2) Predictor Design: The functions $\hat{f}$, $\hat{C}$, and $\hat{\theta_A}$ are defined in terms of least squares regression of multivariate polynomial of degree r. Consider how we create this polynomial for the $\hat{\theta_A}$ (with similar processes used for $\hat{C}$). Let x be an observation (involving r metric readings) at some future time $t + \Delta t$ and X_u be a prior observation (involving r metric readings) at time u for $u = t-1, \ldots, t-p$. For tCAP, we use the standard d-variate normal density function, with $\|x\|$

being the norm of vector x:

$$K(x) = (2\pi)^{-\frac{m}{2}} exp(-\|x\|^2/2)$$

as a tool to localize the neighborhood in which we define our polynomial. We do this through *kernel weighting* with a defined bandwidth matrix H for localization by assigning a weight of $K_H(x) = |H^{-1}|K(H^{-1}x)$. This can be simplified by taking the bandwidth matrix $H = hI_r$, with h being a scalar value and I_r being the identity matrix of order r.

A local constant approximation for $\hat{\theta}$ is defined next in terms of a locally weighted average [38] over the next n observations, based on the prior p observations of $X_{t-1}, \ldots, X_{t-p}$ (each with r metric readings):

$$\hat{\theta}(x) = \frac{\sum_{t=p+1}^{n+p} O_p * K_H(X_{t-1} - x)}{\sum_{t=p+1}^{n+p} K_H(X_{t-1} - x)} \tag{7}$$

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

IV. A Thermally-Aware Scheduler

The scheduler in an operating system is responsible for making two decisions in each time quantum: (1) thread scheduling, i.e., deciding the next thread to run on a processor and (2) load balancing, distributing workload evenly across logical CPUs, with current implementations focusing on performance. 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. Our Theramlly-Aware Scheduler (TAS) extends the existing scheduler in the operating system with thermally-aware load balancing and thread selection.

A. Thermal Predictors

We enhance the existing operating system infrastructure to maintain the information required by the thermal estimator. Our design is based upon the concept of Task Activity Vectors (TAVs) as introduced by [39]. A vector for each kernel thread stores 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. A user-level daemon process collects the information required to compute estimates of $\hat{\theta}$ and $\hat{C}_{theta}$ estimates. Temperature readings are collected by this process from the digital temperature sensor associated with a core. Similarly, processor performance counters are collected by the same process and both sets of metrics are used to estimate $\hat{\eta}$ and $\hat{C}_{theta}$. Estimates are posted via a system call interface to a device driver that collects the data and allocates the data to the currently executing thread. The scheduler queries this driver via a kernel function call interface when making scheduling decisions to determine the $\hat{\theta}$ and $\hat{C}_{\theta}$ values associated with a thread.

B. Thread Selection

The scheduler predicts a thread's impact on the logical processor temperature using the cost predictor $\hat{C}_{\theta}$ (as discussed in Section III-A) and accordingly adjusts the thread priority. The TAS maintains maintains three queue structures per logical processor: an idle queue, current queue, and next queue. The logical processor executes all work on the current queue and then swaps the current and next queue. For performance reasons in our implementation, we follow the convention used by the existing FreeBSD ULE scheduler and place real-time and interrupt threads on the `current` queue.

All other threads are scheduled in terms of an interactivity score. In the original scheduler, the interactivity of a process was determined by the formulas

$$I = \frac{S}{\frac{SL}{RUN}} \tag{8}$$

and, for cases where thread's runtime exceeds its sleep time,

$$I = \frac{S}{\frac{SL}{RUN}} + S \tag{9}$$

where I is the interactivty score, S, the scaling factor of the maximum interactivty score divided by two and SL and RUN the respective cumlative sleep and run times for the thread.

For our TAS, this interactivty score is scaled by the quantity $\hat{C}_{\theta}$, normalized to a percentage value. It was shown in Zhou *et al.* [22] that the greatest thermal benefit occurs by selecting the thread to execute a scheduler should favor the thread that moves the temperature as close as possible to a DTM event without actually triggering that event. The TAS achieves the same effect by scaling the interactivity of the threads by the normalized cost, we give less "thermally costly" threads greater opportunity for access to the processor so as to moderate the processor temperature.

C. Load Balancing

Load balancing distributes workload evenly across the available logical CPUs; current implementations distribute to maximize performance, we enhance the concept to distribute to minimize thermal stress while seeking best performance. The TAS extends the concept of push migration amongst processor groups into the thermal domain by organizing the logical processors 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.

For performance reasons, the thermal processor group topology is maintained by the TEAHarvest thermal predictor driver. The driver allocates thermal efficiency and cost estimates

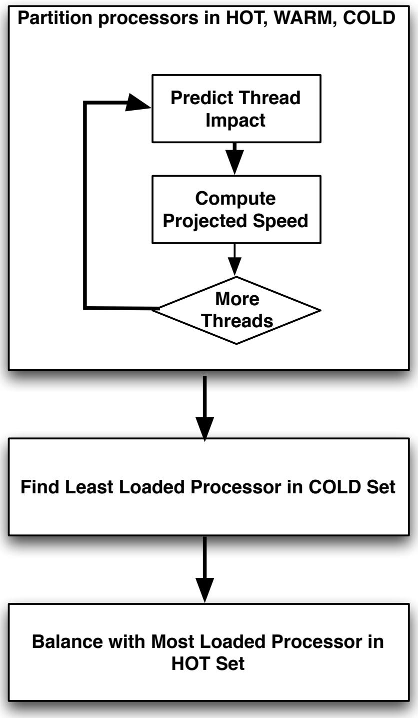

Partition processors in HOT, WARM, COLD
Predict Thread Impact
Compute Projected Speed
More Threads
Find Least Loaded Processor in COLD Set
Balance with Most Loaded Processor in HOT Set

Fig. 2: Design of Thermal Load Balancing in TAS.

Listing 1: TAS Balancing Algorithm

```
begin
 Determine if the logical processor is HOT, WARM
     , or COLD
    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
 Determine the least loaded processor in the ``
     COLD'' set.
 Migrate the thread with the ``worst'' impact on
      temperature to the
 logical CPU in the COLD set most suitable from
     a speed standpoint.
 end
```

to executing threads and maintains the thermal processor group information. The TAS scheduler queries the driver to determine whether a processor will move towards the DTM temperature if a thread becomes ready to execute on the appropriate run-queue. In this way we predict whether a thread moves the logical processor closer to a DTM event and adjust the processor run-queue accordingly to prevent occurrence of the DTM event.

On a perioidic basis (every 500ms), the scheduler executes the algorithm in Listing 1 in attempt to give overtaxed resources additional time to thermally recover. This algorithm addresses thermal issues by moving the work away from thermally stressed logical processors while attempting to maintain system performance by selecting the logical processor with the least load in the “COLD” set.

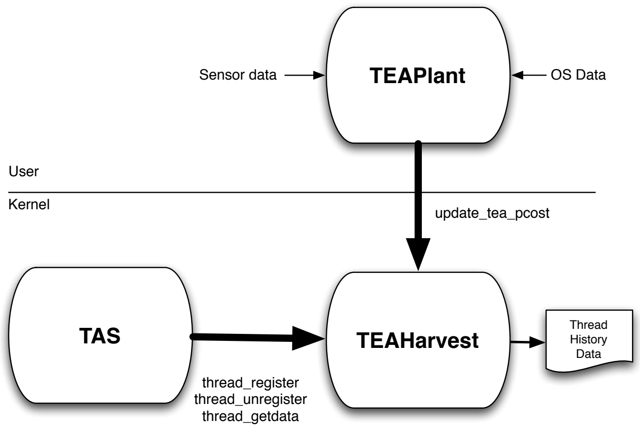


Fig. 3: TEA-Plant - TEA-Harvest Data Collection.

V. Evaluation

We evaluated our Thermally-Aware Scheduler using the FreeBSD operating system on commodity server hardware described in Table III. Our implementation modifies the existing the push migration in FreeBSD’s ULE scheduler [31] to take into account both thermal behavior and system performance per the algorithms in Section IV. PeC data is collected at the user the level using the standard tools provided for that purpose by FreeBSD (the `coretemp` and `hwpmc` kernel extensions). This information is collected and collated by a FreeBSD kernel extension which is queried by the operating system scheduler when making scheduling decisions (Fig. 3)

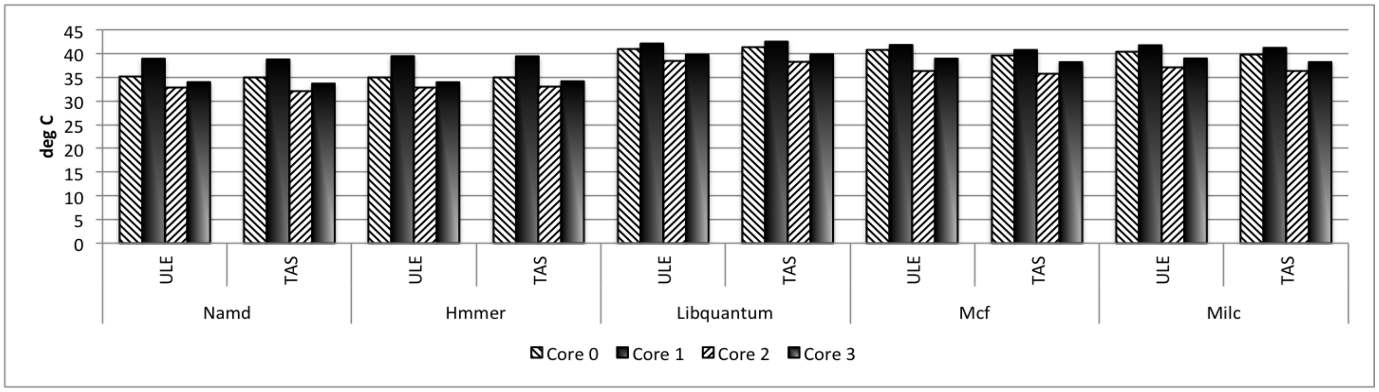
We calibrate the underlying model by considering the behavior of the modified system at idle and under high load stress using common utilities from the FreeBSD regression test suites and software collections. Then, we characterize the CPU-related behavior of our scheduler using integer and floating point benchmarks from the SPEC CPU2006 [40] benchmark suite. Finally, we use benchmarks from the Princeton Application Repository for Shared-Memory Computers (PARSEC) benchmark suite [41] to consider the thread-level and latency behavior of our scheduler.

TABLE III: Server/Processor Used In Evaluation

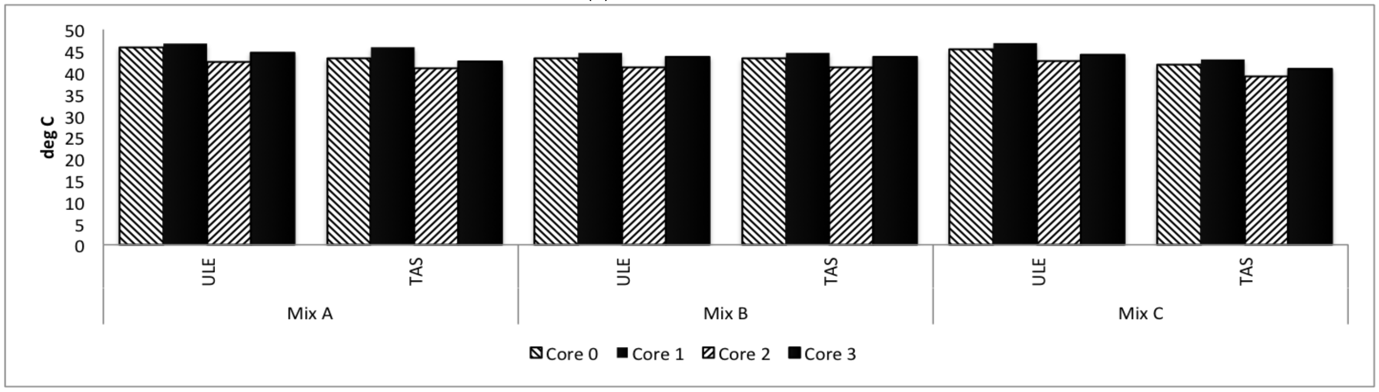
	Dell Precision 490
CPU	Intel Xeon 5300 (Woodcrest)
CPU L2 cache	4MB
Memory	8GB, DDR2 667Mhz ECC
Internal disk	500GB
Network	1x1000Mbps
Video	NVIDA Quadro FX3400

A. Experiment Setup

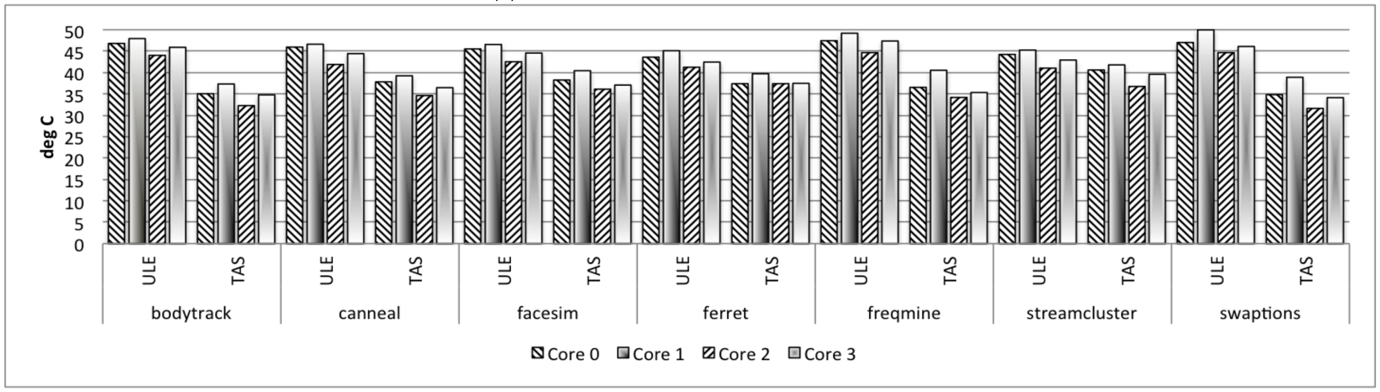
Experiments were executed on the test server specified in Table III, with performance metrics gathered during the course



(a) SPEC CPU2006.



(b) Mixed workloads from SPEC CPU2006.



(c) PARSEC.

Fig. 4: Comparision of average core die temperatures for the ULE and TAS schedulers for selected benchmarks.

of execution. Power consumed is measured by a WattsUP power meter [42], connected between the AC Main and the server under test (SUT). The power meter 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 each run and the measures collected during the runs are downloaded (after execution completion) from meter's internal memory into a spreadsheet.

B. Benchmark Selection Criteria

Components of the processor affect the thermal envelope in different ways [43]. We address this issue by using three criteria for selecting benchmarks from the FreeBSD stress testing, SPEC CPU 2006 and PARSEC benchmark suites (Table IV): (1) sufficient coverage of the functional units in the processor, (2) varying levels of CPU and memory intensity to emulate real-life use of the processor, and (3) levels of reasonable applicability to the problem space.

The benchmarks used in our evaluation address the first criteria in two ways. The FreeBSD stress testing benchmarks operate by repeating a tight set of integer and floating point instructions while doing multiple reads and writes of large memory blocks. This stresses the integer and floating units in the processor while engaging in large amounts of cache activity. Further stress of the integer and floating points unit is addressed by the balancing the selection of benchmarks from the SPEC CPU2006 benchmark suite between integer and floating point benchmarks.

The second criteria is addressed by selecting benchmarks from the SPEC CPU2006 and PARSEC suites that exhibit varying levels of CPU and memory intensity. This is demonstrated by (1) executed benchmarks singly to focus upon each functional unit in the processor, and (2) executed in combination across a mix of integer and floating point benchmarks allocated across all logical processors in the system. Finally, the benchmarks were selected from the SPEC CPU2006 and PARSEC suites based upon fit into the problem space. Each benchmark represents an application typical of the problems solved on high-performance application servers.

C. Calibration and Validation

The thermal model in Section III was calibrated by comparing the behavior of each scheduler at idle against behavior of the scheduler under heavy performance and thermal stress. The system idle workload represents an idle operating system and used as a frame of reference for comparing other workloads. The behavior of the system under high processor utilization and high memory pressure was simulated using multiple instances of the FreeBSD `cpuburn` stress testing package running in parallel, one per core with symmetric multiple threads disabled. The `cpuburn` stress test package implements a single-threaded infinite loop with a sequence of integer and floating-point operations to thermally stress a processor. These synthetic benchmarks was executed for 600 seconds with PeCs in the model sampled at an interval of $t = 5$ seconds. Per the procedure in [35], these time-series observations were consolidated using geometric means and chaotic attractor predictors were built for the metrics in Section III.

TABLE IV: Benchmarks Used For Evaluation

Integer Benchmarks	
hmmer	Search gene sequence
mcf	Combinatorial optimization
Floating Benchmarks	
milc	Quantum Chromodynamics
gromacs	Biochemistry/Molecular Dynamics
libquantum	Physics/General Relativity
namd	Fluid Dynamics

D. CPU-Bound Behavior

It has been shown in prior work [5], [24] that significant core-to-core and functional unit-to-functional unit thermal variation occurs on modern processors. Benchmarks from the SPEC CPU2006 suite [44], (as listed in Table IV), were used to compare the thermal behavior of CPU-bound workloads executed on the TAS scheduler vs. same workload for the ULE scheduler.

Fig. 4(a) compares the average per-core temperature difference between the ULE and TAS schedulers for each benchmark in Table IV. As each of these benchmarks are CPU-bound with activity focused on different core functional units, threads for these processes tend to execute on the same core due to cache affinity. Thus, we see little variation in the temperatures between the two schedulers as very little cross-core activity occurs as the workload executes. In Fig. 5, we consider the average power usage per scheduler for each benchmark. Again, the amount of variation between schedulers is small.

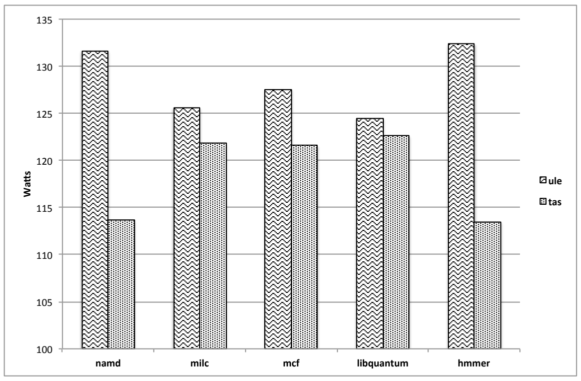


Fig. 5: Average power usage (Watts) per scheduler for selected SPEC CPU2006 benchmarks..

TABLE V: Cross-Functional unit benchmark combinations

Workload	Thread 0	Thread 1	Thread 2	Thread 3
A	namd	namd	hmmer	mcf
B	milc	milc	namd	hmmer
C	milc	mcf	mcf	hmmer

1) Cross-Functional Unit Evaluation: In this section, we consider the effect of mixed workloads multithreading across multiple multiple logical processors. We construct mixed workloads from combinations of the the SPEC CPU2006 benchmarks that span a range of cases including high-IPC/low-IPC, hot/cold thermal profiles (per [23]), and integer/floating-point combinations (Table V).

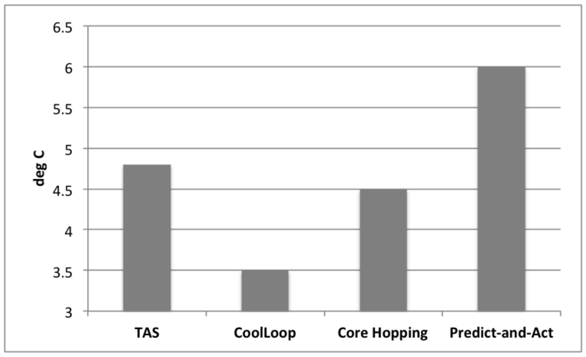


Fig. 6: Average reduction in core die temperature reduction for mixed SPEC CPU2006 workloads.

Figure 4(b) compares the average die temperatures between the ULE and TAS schedulers for each workload mix. In this case, we see temperature improvements between 1.7% and 7.1% (as shown in Table VI) with a performance impact of between 2.1% and 2.9%. This compares favorably with techniques from prior work such as Core Hopping [5], Variation-Aware [23], and Predict-and-Act [20], with comparable reduc-

TABLE VI: Cross-functional Unit Execution

Workload	Core 0	Core 1	Core2	Core 3	Perf. Impact
A	6.0%	2.1%	3.9%	4.5%	2.1%
B	2.2%	1.7%	4.8%	2.5%	2.9%
C	6.9%	7.1%	7.1%	6.5%	2.5%

tions in average core die temperatures and performance impact for similar workloads and processors (see Fig. 6).

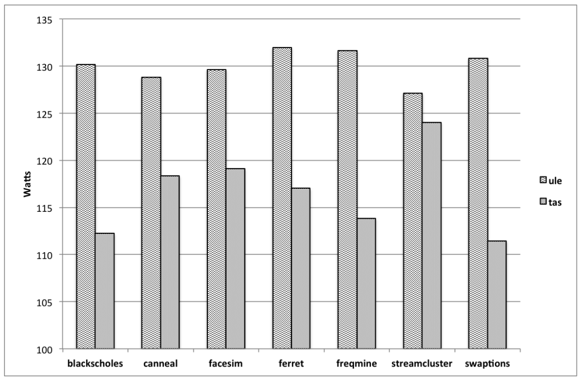


Fig. 7: Average power usage (Watts) per scheduler for selected PARSEC benchmarks

E. Multithreaded behavior

We evaluate behavior of our scheduling algorithms when executing highly parallel workloads using selected benchmarks from the PARSEC [41] suite. Benchmarks in this suite use different combinations of parallel models with workloads selected chosen from the fields of computer vision, computational finance, enterprise servers, and animation physics as described in Table VII. The benchmarks were compiled with the POSIX `pthreads` library and executed using the PARSEC `native` input sets. Figure 4(c) compares the average die temperature observed for each benchmark executed on the unmodified ULE scheduler and our TAS scheduler while Fig. 7 compares the average power consumption for each benchmark between the two schedulers.

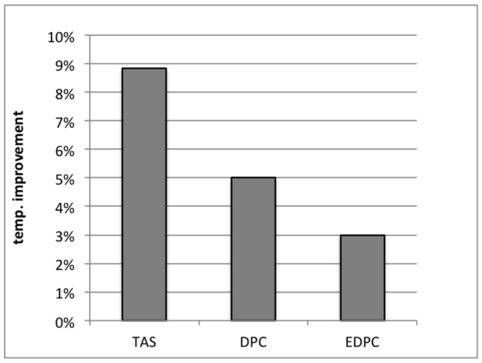


Fig. 8: Average percentage reduction in core die temperature for the `facesim` benchmark for various optimization strategies.

TABLE VII: PARSEC benchmarks used in evaluation

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

In Fig. 8, we contrast the optimization strategy used by TAS versus other possible optimizations schemes for managing the trade-off between performance and energy. In this figure, we compare the results from the TAS scheduler's performance executing the PARSEC `facesim` benchmark as compared to schemes that minimize delay under peak power constraints (DPC) and minimize energy under peak power and delay constraints (EPDC). In this case, the optimizations used by TAS outperforms both DPC and EPDC for the `facesim` benchmark by 4% to 6% as reported in prior work [45].

VI. Conclusion

Dense servers pose power and thermal challenges that pose problems for processor that crude methods such as DVFS and DTM do not address in an effective and efficient manner. As result, system performance can be compromised when such servers are highly utilized. In this work, we have presented scheduling techniques that utilize chaotic time series approximations of thermal metrics to guide load balancing and thread selection. It was found through experimental validation that these techniques are capable of reducing the average on-die core temperatures of a typical server processor between 2-7% as opposed to existing scheduling technology.

References

[1] J. Donald and M. Martonosi, "Techniques for multicore thermal management: classification and new exploration," in *Proc. of the 33rd Int'l Symp. on Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 78–88.

[2] W. L. Bircher and L. K. John, "Analysis of dynamic power management on multi-core processors," in *Proc.of the 22nd Int'l. Conf. on Supercomputing*. New York, NY, USA: ACM, 2008, pp. 327–338.

[3] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross, "Static and dynamic temperature-aware scheduling for multiprocessor {SoCs}," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 9, pp. 1127–1140, 2008.

[4] M. Gomaa, M. D. Powell, and T. N. Vijaykumar, "Heat-and-run: leveraging SMT and CMP to manage power density through the operating system," *SIGOPS Oper. Syst. Rev.*, vol. 38, no. 5, pp. 260–270, 2004.

[5] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose, "Thermal-aware task scheduling at the system software level," *Proc. of the ACM Int'l Symp. on Low Power Electronics and Design*, pp. 213–218, 2007. [Online]. Available: http://doi.acm.org/10.1145/1283780.1283826

[6] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic thermal management through task scheduling," *Proc. of the 2008 IEEE Int'l Symp. on Performance Analysis of Systems and Software*, pp. 191–201, Apr. 2008.

[7] O. Sarood, A. Gupta, and L. Kale, "Temperature aware load balancing for parallel applications: Preliminary work," in *Proc. of the 2011 IEEE Int'l. Symp. on Parallel and Distributed Processing Workshops and Ph.D. Forum*, May 2011, pp. 796 –803.

[8] M. Khan, C. Hankendi, A. Coskun, and M. Herbordt, "Software optimization for performance, energy, and thermal distribution: initial case studies," in *Proc. of the 2011 Int'l. Green Computing Conference and Workshops*, July 2011, pp. 1 –6.

[9] S. Hofmeyr, C. Iancu, and F. Blagojević, "Load balancing on speed," in *Proc. of the 15th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, ser. PPoPP '10. New York, NY, USA: ACM, 2010, pp. 147–158.

[10] Sun Microsystems. (2009, June) OpenSolaris CPU Power Management - Project Tesla.

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

[12] L. Xia, Y. Zhu, J. Yang, J. Ye, and Z. Gu, "Implementing a Thermal-Aware Scheduler in Linux Kernel on a Multi-Core Processor," *The Computer Journal*, vol. 53, no. 7, pp. 895–903, 2010.

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

[14] AMD, *AMD Opteron Processor Data Sheet*, 3rd ed., AMD, March 2007.

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

[16] T. S. Rosing, K. Mihic, and G. De Micheli, "Power and Reliability Management of SoCs," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 4, pp. 391–403, Apr. 2007.

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

[18] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," *Proc. of the 36th IEEE Symp. on Foundations of Computer Science*, p. 374, 1995.

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

[20] R. Z. Ayoub and T. S. Rosing, "Predict and act: dynamic thermal management for multi-core processors," *Proc. of the 14th ACM/IEEE Int'l Symp. on Low Power Electronics and Design*, pp. 99–104, 2009.

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

[22] X. Zhou, J. Yang, M. Chrobak, and Y. Zhang, "Performance-aware thermal management via task scheduling," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 7, no. 1, pp. 1–31, 2010.

[23] E. Kursun, "Variation-aware thermal characterization and management of multi-core architectures," in *Proc. of the 2008 IEEE International Conference on Computer Design*. IEEE, Oct. 2008, pp. 280–285.

[24] C.-Y. Cher and E. Kursun, "Exploring the effects of on-chip thermal variation on high-performance multicore architectures," *ACM Trans. on Architecture and Code Optimization*, vol. 8, pp. 2:1–2:22, February 2011. [Online]. Available: http://doi.acm.org/10.1145/1952998.1953000

[25] R. Ayoub, K. Indukuri, and T. Rosing, "Temperature aware dynamic workload scheduling in multisocket CPU servers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 9, pp. 1359 –1372, Sept. 2011.

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

[27] A. Merkel, J. Stoess, and F. Bellosa, "Resource-conscious scheduling for energy efficiency on multicore processors," in *Proc. of the 5th European conference on Computer systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 153–166.

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

[29] P. Bailis, V. J. Reddi, S. Gandhi, D. Brooks, and M. Seltzer, "Dimentrodon: Processor-level preventive thermal management via idle cycle injection," in *Proc. of the 48th Design Automation Conference (DAC 2011)*. New York, NY, USA: ACM, June 2011. [Online]. Available: http://www.eecs.berkeley.edu/ pbailis/papers/dimetrodon-dac2011.pdf

[30] K. Li, "Performance Analysis of Power-Aware Task Scheduling Algorithms on Multiprocessor Computers with Dynamic Voltage and Speed," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1484–1497, Nov. 2008.

[31] J. Roberson, "Ule: a modern scheduler for freebsd," in *Proc. of the BSD Conference 2003 on BSD Conference*. Berkeley, CA, USA: USENIX Association, 2003, pp. 3–3.

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

[33] ——, "Thread scheduling in freebsd 5.2," *Queue*, vol. 2, pp. 58–64, October 2004.

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

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

[36] Z. Liu, "Chaotic Time Series Analysis," *Mathematical Problems in Engineering*, vol. 2010, 2010.

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

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

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

[40] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Comp. Archit. News*, vol. 34, no. 4, pp. 1–17, 2006.

[41] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," in *Proc. of the 17th Int'l. Conf. on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 72–81. [Online]. Available: http://doi.acm.org/10.1145/1454115.1454128

[42] Electronic Educational Devices, Inc., "WattsUp Power Meter," December 2006.

[43] A. Kumar, L. Shang, L.-S. Peh, and N. Jha, "System-Level Dynamic Thermal Management for High-Performance Microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 96–108, Jan. 2008.

[44] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *Computer Architecture News*, vol. 34, no. 4, Sept 2006.

[45] R. Cochran, C. Hankendi, A. Coskun, and S. Reda, "Identifying the optimal energy-efficient operating points of parallel workloads," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, nov. 2011, pp. 608 –615.