Thermal-Aware Scheduling in Multicore Systems Using Chaotic Attractor Predictors

Adam Lewis and Nian-Feng Tzeng

Center for Advanced Computer Studies, University of Louisiana at 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 throttle down the processor when necessary. However, DVFS tends to yield marked degradation in both application performance and system reliability. Thus, pro-active scheduling techniques that avoid thermal emergencies are preferable over reactive hardware techniques like DTM. Based on our previously introduced thermal Chaotic Attractor Predictors (CAPs), which take into account key thermal indicators and system performance metrics for overall energy consumption estimation, we have developed and evaluated an effective thread scheduler for multicore systems. Besides CAPs, our scheduler makes use of two basic principles to minimize server energy consumption: (1) selecting the thread with the least probability of causing a DTM during the subsequent time quantum, for execution in each core, and (2) migrating execution threads on thermally overextended cores to other cool cores via load balancing so as to observe the thermal envelope. Our developed scheduler is evaluated in practice to assess its potential advantage resulting from thermal-awareness by incorporating thermal CAPs (for temperature prediction) and basic principles (for energy reduction) into the existing scheduler in the FreeBSD operating system. Our implemented scheduler is run on a server with the Intel Xeon processor for gathering measures of interest (including die temperature readings and run times) when benchmark codes from the PARSEC suite are executed. The gathered results reveal that our proposed scheduler exhibits reduction in mean core on-die temperatures by up to 12.8°C while mostly experiencing only 1% to 4% performance degradation.**

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 (known as Dynamic Voltage and Frequency Scaling (DVFS)) to throttle down the processor whenever necessary. However, the use of DVFS tends to cause significant negative impacts on application performance and system reliability [5], [7], [8]. This paper introduces effective scheduling for preventive thermal management that minimizes server energy consumption by (1) selecting the subsequent thread with the smallest thermal impact in the next time quantum to execute on an available core, and (2) relocating threads run on thermally overextended cores to other available cores for load balancing. As opposed to prior pursuit of thermal-aware scheduling [6], [20], [21] that aimed to bound temperatures below critical thresholds, our work considers how to schedule high workload for die temperature management across all cores.

The current generation of operating systems treats those cores in multicore and virtualized multi-threaded processors (like those with Intel's HyperThreading technology) as distinct logical units, called *logical cores*, which are scheduled independently. However, dependency and contention exist in shared resources among those logical cores, and hence they need to be taken into account upon their scheduling to address performance and energy efficiency. It thus calls for the need of intelligent, thermal-aware load balancing and scheduling within the operating system, achievable via modeling full-system energy consumption based on computational load for effectively predicting future energy consumption and its associated thermal change.

To this end, we arrive at a thermal model which relates server energy consumption to the overall thermal envelope, establishing an energy relationship between workload and overall system thermodynamics. According to our analysis of experimental measurements of key processor performance counter readings and performance metrics, it is found that the measured readings and metrics do not possess linearity and are *chaotic in nature*. Hence, our thermal model, based on a thermal Chaotic Attractor Predictor (tCAP), 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. This work demonstrates that effective scheduling can result from taking advantage of our devised tCAP when dispatching jobs to confine server power consumption within a given power budget and thermal envelope while avoiding detrimental impact on thread execution performance.

Dealing with all cores in a multicore system, a thermal-aware thread scheduler is highly desirable to ensure that all of the cores in such a system are kept equally busy, realized by migrating threads over all the cores (no matter whether in the same processors or different ones) when necessary for load balancing in the system. Existing schedulers aid in processor thermal management by dispatching workloads to cores as tightly as possible within the same processors to expose more opportunities for power management via shutting down unused cores. However, computation-bound server workload fully utilizes available system cores in most times, thereby rendering such schedulers ineffectual. Our Thermal-Aware Scheduler (TAS) addresses this problem by thermally balancing system workload with as little performance degradation as possible.

Our TAS is demonstrated and evaluated by adding thermal-awareness to the existing scheduler in the FreeBSD operating system executed on Intel Xeon (Woodcrest) processors. Benchmarks from the PARSEC suite which represent typical application server workloads, are chosen to evaluate our TAS. The gathered results unveil that TAS achieves reduction in mean core on-die temperatures by up to 12.8°C (from 44.8°C down to 32.0°C) under PARSEC benchmarks, while mostly experiencing only 1% to 4% performance degradation. Our TAS compares favorably with a recent energy-aware scheduling technique [20], which gets core temperature reduction by up to 4°C (from 63°C down to 59°C) when four parallel scientific applications compatible to PARSEC benchmarks were executed on a physical 4-core Intel Xeon 5520 processor (similar to our testbed processor).

II. Background and Related Work

Traditional interactive load balancing assumes workload behavior involves independent tasks that remain quiet for extended periods when making thread placement decisions. Server workloads, on the other hand, are assumed to contain large numbers of threads which are highly independent of each other and use synchronization objects to ensure mutual exclusion on small data items. Modern operating systems make load balancing more power-aware by placing workloads to cores as tightly as possible, thereby presenting more opportunities for power management software to shutdown unused resources.

A. Thermal Modeling

Existing thermal models all required to fit targeted mathematical expressions based on time-series observations of temperatures during the course of executing various workloads. Expression coefficients were estimated by minimizing total least square errors between modeling results and actual temperature readings. After proper calibration, such a mathematical expression becomes a model for extrapolating future temperatures. Different modeling techniques have been devised using (1) integer linear programming [13] to meet real-time deadlines while minimizing hot spots and spatial temperature differentials across the die, (2) dynamic thermal modeling guided thread migration [10], (3) priority adjustment of threads assigned to overheated cores [2], or (4) schedule rearrangement favoring threads that cause the greatest temperature hike while avoiding DTM events [21]. Schedulers based on prior thermal modeling all rely on readings of hardware performance counters and temperature sensors. They can be improved by analyzing on-die thermal variations to aid in system power and thermal management [3].

However, preceding techniques are reactive to the temperature approaching the DTM threshold rather than trying to avoid reaching that temperature in the first place. A proactive solution with multi-tier prediction was suggested earlier [2], where a core level predictor was employed to convert temperature observations to operating frequency estimates while a control-theoretic based scheduler was followed at the socket level for process level scheduling. Separately, a scheduling policy was proposed for scheduling memory-bound tasks at slower frequencies by sorting the tasks in each core's run queue according to (1) memory intensity, (2) the contribution of each task to the system power consumption and (3) the current processor temperature [19]. Meanwhile, Cool Loop [6] and Dimentrodon [3] address a lack of heat slack by inserting additional cycles into the task scheduling to create thermal slack, naturally leading to performance degradation. However, such approaches work ineffectively under many server cases where the slack in deadlines usually is unavailable.

B. Earlier Scheduling with Load Balancing Support

Modern multiprocessor operating systems often take a two-level approach to task scheduling for maximized system resource utilization. Such an approach uses a distributed run queue and follows fair scheduling policies to manage each core at the first level. Its second level balances the work load by redistributing tasks across the queues. Such two-level schedulers work under three assumptions: (1) threads are independent, (2) load is governed by queue length, and (3) locality exists and is important [12]. In practice, however, common servers often have the following characteristics: (1) their threads are logically related, with data and control dependencies among threads, and (2) their threads have equally long life spans [12], rendering previous two-level scheduling ineffective.

TABLE I: PeCs and performance metrics for Intel Xeon server

Variable	**Measurement** (for m cores and k fans)
Application length	
IR	Instructions retired
Application data set	
QPL_{P_t}	Transactions on QPLs between Pair P_t of cores
QPL_{IO}	Transactions on QPLs for IO Handler
CM_i	Last-level cache misses in Core i, $0 \leq i < m$
D_r	Disk bytes read
D_w	Disk bytes written
Physical core temperature	
T_{C_i}	Core i on-die temperature, $0 \leq i < m$
System temperature	
T_{A_j}	Ambient temperature sensor j, $0 \leq j < 3$
Electromechanism	
F_{C_j}	Memory cooling fan C_j speed, $0 \leq j < k$

Meanwhile, existing operating systems have made their load balancing schemes more power-aware by taking advantages of their power management drivers which intend to find as compact an allocation of processes to run-queues as possible. This way presents more opportunities for the power management software to shutdown unused resources. Such power-aware load balancing, while effective for interactive workloads, works only if system resources are not completely utilized; it becomes nonviable if the system workload is high (common to high-performance servers).

C. Chaotic Behavior of Energy Consumption

An analytical model of server energy consumption was built earlier [14], [15] 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 expresses energy consumption in the time domain as follows:

$$E_{system} = f(E_{proc}, E_{mem}, E_{em}, E_{board}, E_{hdd}) \quad (1)$$

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, and (5) E_{hdd}: energy consumed by the hard disk drive during the system's operation.

The continuous system in Eq. (1) can be viewed as a multi-variate differential equation in the time domain that can be estimated using a time series representation. This time series is constructed 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 $t + \Delta t$.

Observations from each time series 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. The inputs for our predictor is listed in Table I. They are classified into five groups, each associated with one server energy contributor. As one contributor, application length is estimated in each time period by the quantity IR, the total number of retired instructions of the thread. Among the contributor of "application data set" listed in Table I, QPL_{P_t} and QPL_{IO}

are relevant to QuickPath Links, and they are associated with E_{proc} and E_{mem}, respectively. Each CM_i measures the total last-level cache miss counts due to Core *i*, $0 \leq i$ ¡ m, m being the number of cores in the processor, and they determine E_{mem}. As the contributor of "physical core temperature," the core die temperature measures are pertinent to E_{proc}. The subsequent measures dictate E_{board}, obtained from 3 temperature sensors placed on the board for ambient temperature readings and Finally, F_{C_j} measures speed information of those nine memory cooling fans which determines E_{em}.

D. Limitations of Existing Scheduling

Current power management software that utilizes DVFS techniques to address DTM events has been effective in addressing thermal emergencies [8], [11], commonly implemented in modern server processors. However, handling DTM through DVFS can be problematic due to issues with program phase behavior and contention for shared resources [5], [7], resulting directly from slow transitions between the active and the idle device states and also from inability to access resources associated with idle processors. When the power phase changes frequently, abundant thermal variations among cores within the processor occurs, leading to decreased reliability [7], [13].

A study of OS-level thermal migration using Linux on the IBM POWER5 processor [6] discovered that the rise and fall times of core temperatures vary in the order of hundreds of milliseconds. As most operating systems choose scheduler ticks to be of 10 ms or less, it often is impossible to react to thermal conditions before a critical state is reached. As a result, three improvement mechanisms for managing thermal states have been pursued: (1) core hopping for leveraging spatial heat slack, (2) task scheduling for leveraging temporal heat slack, and (3) SMT scheduling for leveraging temporal heat slack. In the presence of slack, each of those mechanisms may reduce core die temperatures by 3 to 5° C on an average, at the expense of 3% mean performance degradation [6], [2]. However, in the absence of slack commonly found under heavy workloads in high-performance servers, the three mechanisms becomes ineffective, calling for suitable scheduling with thermal awareness proactively.

III. Proposed Thermal Aware Scheduler

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

A. Thermal Predictors

An application A is composed of p execution threads, with a data set D_A consisting of d_i data per processor, for $1 \leq i \leq p$. Thus, energy consumed by executing application A with data set D_A can be expressed as:

$$E_A(A, D_A, t) = \sum_{i=1}^{p} W(\tau_i, d_i, t_i) \qquad (2)$$

for $1 \leq i \leq p$, where W is the workload associated with execution thread τ_i involved in application A, d_i is the corresponding data set for that thread, and t_i is its thread execution time. The workload W is measured in terms of the number of bytes operated upon or transferred over, during the course of completing those instructions retired by the logical core for each thread p. We relate system energy expenditure (upon application execution) to corresponding joule heating by defining "Thermal Equivalent of Application" (TEA) as the electrical work converted to heat in running the application, leading to die temperature change and ambient temperature change of the system:

$$\Theta(A, D_A, T, t) = \frac{E_A(A, D_A, t)}{\lim\limits_{T \to T_{th}} J_e(D_A, \Psi_{cp})(T - T_{nominal})}, \quad (3)$$

where T_{th} denotes the threshold temperature at which a thermal emergency event will occur, and $T_{nominal}$ refers to the nominal temperature as reported by the system when the operating system is running without any application execution. The term J_e is the "electrical equivalent of heat" for the chip, which reflects the processing the data bits during application execution and the black body thermal properties of the chip packaging as well as the cooling mechanisms around the chip, as defined by the parameter Ψ_{cp}. The value of Ψ_{cp} is an ambient thermal characterization parameter provided by the hardware manufacturer to relate temperature to power for cooling purposes [1]. We compute the achieved performance per unit energy consumed by the chip:

$$C_\theta(A, D_A, T, t) = \frac{\Theta(A, D_A, t)}{E_{system}(A, D_A, t)} \qquad (4)$$

This normalized quantity indicates the "cost" of executing an application on a given processor, with $E_{system}(A, D_A, t)$ obtained from energy consumption of individual physical components (processor, DRAM units, HDD, motherboard, and electrical/electromechanism) given by Eq. (1).

Following the procedure reviewed in Section II-C (and detailed in [15]), we create thermal Chaotic Attractor Predictors (tCAP) for Θ and C_θ as a means for predicting the thermal behavior during application execution. We enhance the existing FreeBSD operating system to maintain information required by the thermal estimator. Our design is based on the concept of Task Activity Vectors (TAVs) introduced earlier [18], with a vector for each kernel thread to store the required history in order to make sound prediction. Generally, the more additional space is employed for history maintenance, the higher benefit our thermal scheduling gains.

B. Thread Scheduling

The scheduler for each logical core uses the cost predictor for C_θ to predict a thread's impact on core temperature and

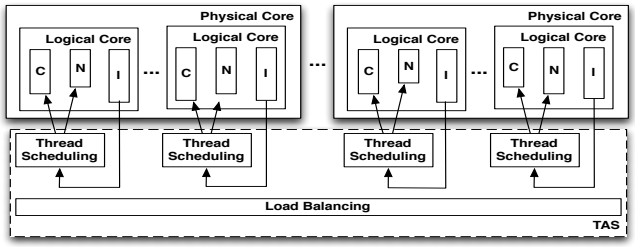

Fig. 1: Internal structure of the TAS scheduler, with C (or N) denoting current (or next) queue and I denoting idle queue.

adjust the thread priority as required to prevent an increase in core temperature. Each core will have one process in a running state on each scheduling interval. All other threads are at one of three queue structures maintained per logical core: an idle queue, a current queue, and the next queue (illustrated in Fig. 1). When a thread becomes idle, it is added to the idle queue. Threads in the current (C) queue are executed in order, and after all threads therein are exhausted, the next (N) queue is swapped with the current queue so that threads in the next queue get executed in subsequent intervals. If both C and N queues are empty, no thread in the logical core is ready for execution. For performance reasons, our implementation follows the convention used by the existing FreeBSD ULE scheduler, placing real-time and interrupt threads on the current queue. This is reasonable, given that real-time thread deadlines must be satisfied and interrupt service routines are typically short in length. Meanwhile, all other non-idle threads are assigned to either the current or next queue based upon computing an interactivity score, I, for each thread, as follows:

$$I = \begin{cases} S/(SL/RUN) & \text{if sleep time} \geq \text{run time} \\ (S/(SL/RUN)) + S & \text{if run time} < \text{sleep time} \end{cases} \quad (5)$$

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

For our TAS, the interactivity score I is scaled by the predicted value of C_θ, normalized to a percentage value. It was shown previously [22] that the greatest thermal benefit occurred when a scheduler favored the thread which moved the core temperature as close as possible to the DTM threshold without actually triggering a DTM event. TAS achieves a similar effect by scaling the interactivity of a thread by its normalized execution cost, thereby giving less "thermally costly" threads greater opportunities for execution so as to moderate the processor temperature. Penalizing more thermally costly threads reduces the opportunities for such threads to gain access to logical cores, presenting similar advantages of techniques which artificially inject slack into thread scheduling but without extra scheduling overhead incurred to those techniques. Threads with higher "thermal cost" will be scheduled onto the next queue and avoid contributing to the thermal load of the processor. Threads on the next queue will be scheduled to run when the queues are switched and are guaranteed to run at least once every two queue switches, which maintains fair sharing of the processor.

C. Load Balancing

Load balancing distributes workload evenly across the available logical cores, with current implementation mostly aiming to maximize performance. This work applies load balancing to minimize thermal stress while seeking best performance. Specifically, TAS organizes system cores into "thermal clans" based upon the temperature and execution frequency. Specifically, a local core is assigned to one of the three thermal clans: Hot, Warm, and Cold, if its on-die temperature is respectively 90% or higher, between 75% and 90%, and below 75% of the DTM threshold temperature. Note that if a physical

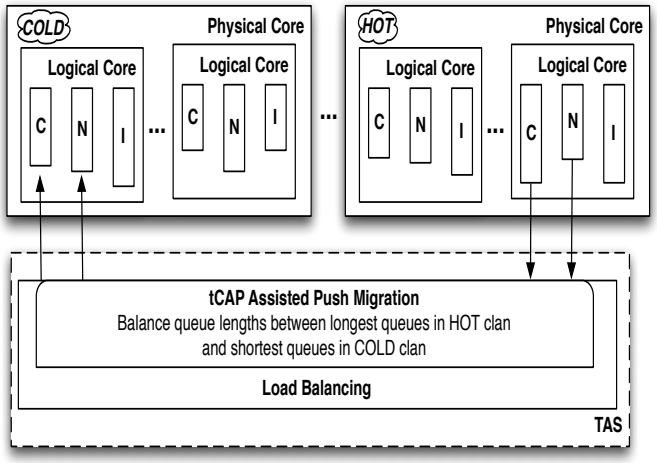

Fig. 2: TAS thermal load balancing.

core supports α threads, α logical cores will result from the physical core, with their temperatures all equal to that of the physical core. In addition, logical cores are grouped into fast and slow clans according to the execution frequencies of their underlying physical cores. This two-level categorization allows TAS to manage work load distribution better from the thermal and performance prospectives, by migrating work away from hot units with negligible execution performance degradation.

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

On a periodic basis (i.e., once in every 500 ms), the TAS scheduler balances the workload among logical cores per the process shown in Fig. 2, giving sufficient times for overtaxed resources to thermally recover. It is realized by using tCAP to estimate the Θ value for each thread in the C and N queues to predict the resulting change in temperature if this thread were to execute. TAS then moves the thread with the greatest temperature impact from the "Hot" clan to the least loaded logical core in the "Cold" clan. This process repeats until the two involved logical cores have the same number of threads in their respective two C and N queues combined, effectively moving workload away from thermally stressed logical cores while maintaining execution performance.

Periodically, the system reads the temperatures of it all logical cores and, if required, also moves a logical cores to a different thermal clan. It should be noted that the time required for a logical core to recover from a thermal event is significantly longer than the interval used for thread scheduling [6]. This allows our TAS to use a much larger interval (2 seconds) between scans (across core temperatures) to determine if any logical core must be assigned to a new thermal clan for better scheduling outcomes.

IV. Experimental Evaluation and Results

We have evaluated our TAS (Thermal Aware Scheduler) using the FreeBSD operating system run on a commodity server. Our TAS implementation modified the existing

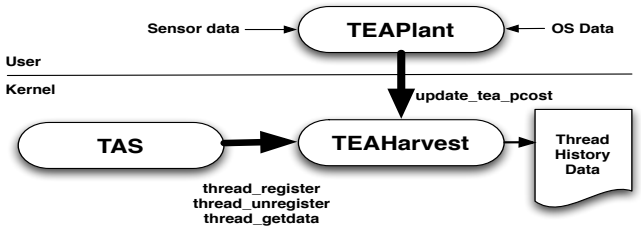

Fig. 3: TEAPlant and TEAHarvest data collection.

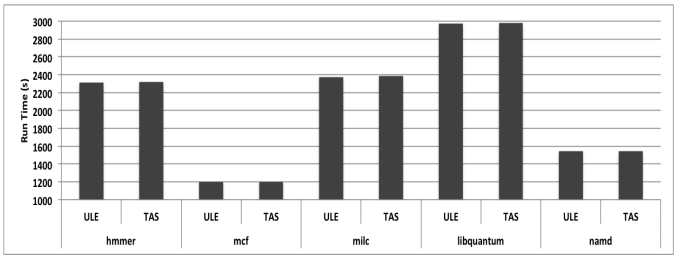

Fig. 4: Comparison of SPEC CPU2006 performance for ULE and TAS schedulers.

thread selection and load balancing in FreeBSD's ULE scheduler [17] to take into account both thermal behavior and system performance, as elaborated in Section III.

Fig. 3 shows how the TAS was incorporated into FreeBSD. A user-level daemon process collects required information to compute the tCAP predictions. Temperature readings are collected by this process from the digital temperature sensor associated with a core. Similarly, processor performance counters are gathered by the same process through standard tools provided for the data collection purpose by FreeBSD (i.e., the `coretemp` and `hwpmc` kernel extensions), with both sets of metrics used to generate estimates. Estimates are posted via a system call interface to a device driver that collects the data for use by the currently executing thread. The scheduler queries this driver via a kernel function call interface when making scheduling decisions to determine the Θ and C_θ estimates associated with a thread. The correct behavior of our TAS modifications to the FreeBSD scheduler was confirmed using the standard regression testing provided by FreeBSD. Further confirmation was attained by comparing the performance of single SPEC CPU2006 benchmarks executed under both schedulers, as shown in Fig. 4, where similar execution times were observed since TAS then failed to benefit from thermal scheduling and load balancing as only one logical core across the whole system was busy in benchmark execution during each time interval.

The processor thermal model was calibrated by measuring behavior outcomes of the testbed with TAS at idle and under high load stress using common utilities from the FreeBSD regression test suites and software collections. Benchmarks from the Princeton Application Repository for Shared-Memory Computers (PARSEC) suite [4] were then evaluated on the testbed to assess TAS in terms of key metrics of interest (i.e., die temperature, benchmark run time, and mean power dissipation) under high parallelism in the thread level.

A. Experiment Setup

Experimental evaluation was conducted on our TAS using an Intel Xeon 5300 (Woodcrest) test server with 4MB L2 cache and 8GB DDR2 667Mhz Memory. The key metrics of interest were gathered during the course of application execution. Power dissipated was measured by a WattsUP power meter [9], connected between the AC Main and the server testbed. The power meter measured the total and average wattage, voltage, and amperage over the run of a workload. The internal memory of the power meter was cleared at the start of each run and the measures collected during the runs were downloaded (after execution completion) from the meter's internal memory into a spreadsheet. Core die temperatures were measured using physical sensors in the processor as reported by the FreeBSD `coretemp` kernel extension.

B. tCAP Establishment

Since our TAS relies on tCAP (thermal Chaotic Attractor Predictor) of the testbed for effectively predicting its thermal behavior during application execution, it is required first to establish tCAP using suitable applications. Two training scenarios were used to determine the coefficients of the chaotic attractor predictor: the idle scenario and the most stress scenario. The idle scenario referred to no application thread ready for execution and thus the testbed runs only the OS threads, whereas the most stress scenario leads to the highest processor utilization coupled with heaviest memory activities, achieved by launching multiple instances of the FreeBSD `cpuburn` stress testing codes for concurrent execution, one per core with symmetric multiple threads disabled. The `cpuburn` stress test code implements a single-threaded infinite loop with a sequence of integer and floating-point operations and large blocks of memory reads and writes to thermally stress the testbed system, driving it close to a thermal emergency. These two extreme thermal scenarios are effective for tCAP establishment, because any typical application execution will lie between them, as far as the coefficients of tCAP are concerned.

These two training scenarios run for 600 seconds, with PeCs sampled at an interval of $t = 5$ seconds for establishing tCAP. The procedure for tCAP establishment is the same as what was described earlier [15], and it involved three steps. First, the training set is constructed by consolidating the observed PeCs into a time series using geometric means. In the second step, the Takens Embedding Theorem [16] is applied to find delay embedding, with the nearest neighbors algorithm then employed to identify the set of attractors using the embedded set. Finally, our tCAP is established by solving the linear least squares problem resulted from fitting a multi-variate polynomial to the attractor set. The time complexity of establishing tCAP equals O(n^2), where n is the number of sampled PeCs [15]. Note that the establishment of the attractor set for tCAP is required only once for a processor type, irrespective of applications executed on the processor.

C. PARSEC Benchmark Results and Discussion

Selected benchmarks from the PARSEC [4] suite were used to evaluate our TAS. They were compiled with the POSIX `pthreads` library and executed using the PARSEC `native` input sets. Being parallel workloads selected from the fields of computer vision, computational finance, enterprise servers, and animation physics, PARSEC benchmarks represent real life applications that can benefit markedly from scheduling their abundant independent threads freely based on thermal prediction.

Three behavior metrics of interest under TAS are gathered: (1) the average core on-die temperature, (2) the benchmark run time, and (3) mean system power dissipation. The outcomes of average core on-die temperatures upon executing those seven PARSEC benchmarks are depicted in Fig. 5. As can be seen from the figure, the core on-die temperatures range

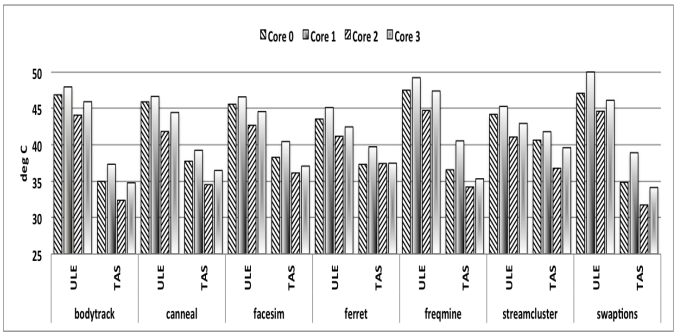

Fig. 5: Comparison of PARSEC benchmark average core die temperatures under ULE and TAS schedulers.

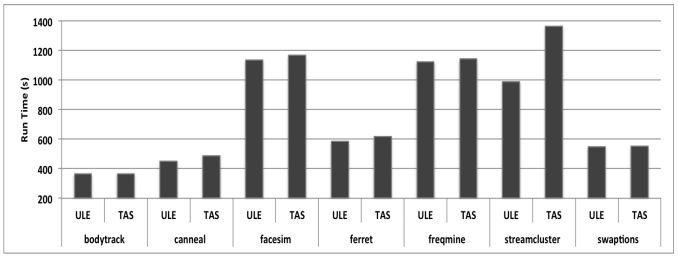

Fig. 6: Comparison of PARSEC benchmark performance under ULE and TAS schedulers.

from 32°C (for the `swaption` benchmark) to 42°C (for the `streamcluster` benchmark) under TAS, with `swaption` and `bodytrack` (or `streamcluster`) experiencing the lowest (or highest) mean temperature across the four cores. Temperature outcomes for the same benchmarks under the classical ULE scheduler are also included in Fig. 5 for comparison. It is found that temperature reduction amounts are larger for benchmarks with smaller working sets (like `bodytrack` and `swaptions`). This is because such a benchmark has a lower cache requirement [4] and thereby lets TAS schedule threads more freely according to thermal prediction with negligible performance degradation, resulting in better temperature reduction. Consequently, TAS enjoys temperature reduction by up to 12.8°C (from 44.8°C down to 32.0°C) for the `swaptions` benchmark.

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

Execution performance (measured in terms of the benchmark runtime) under TAS is depicted in Fig. 6. When compared with the runtime results under ULE included in the figure, it can be observed that TAS leads to negligible performance degradation, by no more than 3.3% for all benchmarks examined except `streamcluster`. Considerable performance degradation is seen for the `streamcluster` benchmark, however, likely due to its high computational intensity as the data set grows with rising dimensionality, requiring to better observe cache affinity upon load balancing. Further rectification for TAS to preserve cache affinity is expected to avoid noticeable performance degradation.

Mean system power dissipation for each of the PARSEC benchmarks run on our testbed under both schedulers is shown in Fig. 7. When compared with ULE, TAS is observed to reduce power dissipation markedly, from 14W (for the

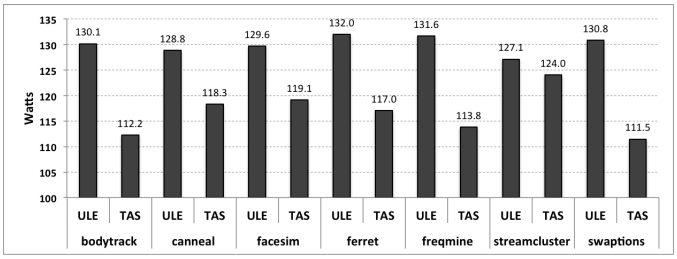

Fig. 7: Comparison of PARSEC benchmark average power dissipation under ULE and TAS schedulers.

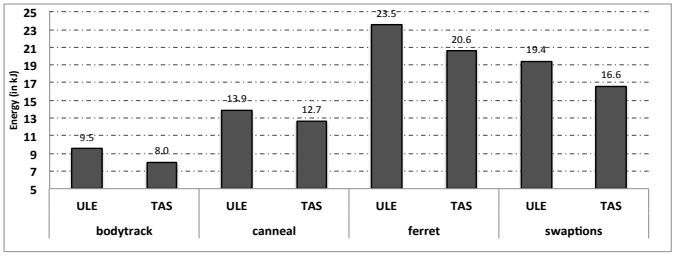

Fig. 8: Energy comparsion of PARSEC non-streaming benchmarks under ULE and TAS schedulers.

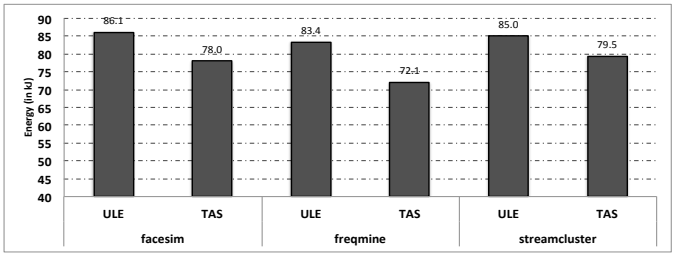

Fig. 9: Energy comparsion of PARSEC streaming benchmarks under ULE and TAS schedulers.

`ferret` benchmark) to 19W (for `swaptions`), due to their relatively small working sets. On the other hand, less reduction in power dissipation is found for benchmarks with large working sets, lowering power by the range of 3W (for `streamcluster` to 10W (for `facesim`).

Total energy consumption for benchmark execution better reflects advantages of TAS, since it takes into consideration both power and execution time, namely, energy (in joule) = power (in W) * time (in sec). Energy consumption results for PARSEC benchmarks with smaller working sets (i.e., those non-streaming ones) are depicted in Fig. 8. They show energy savings under TAS by up to 16% (for `bodytrack`). Energy consumption outcomes for the three PARSEC streaming benchmarks are demonstrated in Fig. 9, where they are all drastically larger than those illustrated in Fig. 8, resulting mainly from their longer execution times.

V. Conclusion

Dense servers pose power and thermal challenges, which often cannot be addressed satisfactorily by conventional DVFS and DTM mechanisms, especially under heavy workloads common for high-performance systems. We have investigated into thermal-aware scheduling to deal with such challenges, capable of managing system energy consumption within a given power and thermal envelope effectively. As opposed to prior work aiming to bound temperatures below critical thresholds, our proposed scheduler considers how to dispatch heavy workloads in the high-performance multicore system for die temperature management across all cores. It is based on the

thermal Chaotic Attractor Predictors (tCAPs) we develop to guide thread selection and load balancing, taking into account key thermal indicators and system performance metrics for preventing DTM instances proactively. The proposed tCAP-oriented scheduling (dubbed the TAS scheduler) has been implemented to augment the original scheduler of the FreeBSD operating system (called the ULE scheduler) for evaluation on a testbed server under benchmarks from the PARSEC suite. Experimental results demonstrate that our TAS scheduler can lower the mean on-die core temperature by up to 12.8°C (from 44.8°C down to 32.0°C), in comparison to the ULE scheduler. When compared with a recent energy-aware scheduling technique reported to attain core temperature reduction by up to 4°C (from 63°C down to 59°C) upon executing four parallel scientific applications compatible to PARSEC benchmarks on an Intel Xeon 5520 4-core processor [20], our TAS clearly enjoys better thermal reduction under multi-threaded execution.

Acknowledgments

This work was supported in part by the U.S. Department of Energy (DOE) under Award Number DE-FG02-04ER46136 and by the Board of Regents, State of Louisiana, under Contract Number DOE/LEQSF(2004-07)-ULL.

References

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

[2] R. Ayoub, K. Indukuri, *et al.* Temperature aware dynamic workload scheduling in multisocket CPU servers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(9):1359 –1372, Sept. 2011. ISSN 0278-0070.

[3] P. Bailis, V. J. Reddi, *et al.* Dimentrodon: Processor-level preventive thermal management via idle cycle injection. In *Proc. of the 48th Design Automation Conference (DAC 2011)*. ACM, New York, NY, USA, June 2011.

[4] C. Bienia. *Benchmarking Modern Multiprocessors*. Ph.D. Dissertation, Princeton University, January 2011.

[5] 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*, pp. 327–338. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-158-3.

[6] J. Choi, C.-Y. Cher, *et al.* 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.

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

[8] 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*, pp. 78–88. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2608-X.

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

[10] M. Gomaa, M. D. Powell, *et al.* Heat-and-Run: leveraging SMT and CMP to manage power density through the operating system. *SIGOPS Oper. Syst. Rev.*, 38(5):260–270, 2004. ISSN 0163-5980.

[11] H. Hanson, S. W. Keckler, *et al.* Thermal response to DVFS: analysis with an Intel Pentium M. In *Proc. of the 2007 international symposium on Low power electronics and design*, ISLPED '07, pp. 219–224. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-709-4.

[12] S. Hofmeyr, C. Iancu, *et al.* Load balancing on speed. In *Proc. of the 15th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, PPoPP '10, pp. 147–158. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-877-3.

[13] E. Kursun and C.-Y. Cher. Temperature variation characterization and thermal management of multicore architectures. *IEEE Micro*, 29:116–126, 2009. ISSN 0272-1732.

[14] A. Lewis, S. Ghosh, *et al.* 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.

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

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

[17] M. K. McKusick and G. V. Neville-Neil. *The Design and Implementation of the FreeBSD Operating System*. Pearson Education, 2004. ISBN 0201702452.

[18] 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.

[19] A. Merkel, J. Stoess, *et al.* Resource-conscious scheduling for energy efficiency on multicore processors. In *Proc. of the 5th European conference on Computer systems*, EuroSys '10, pp. 153–166. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-577-2.

[20] O. Sarood, A. Gupta, *et al.* 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*, pp. 796 –803. May 2011. ISSN 1530-2075.

[21] J. Yang, X. Zhou, *et al.* 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.

[22] X. Zhou, J. Yang, *et al.* Performance-aware thermal management via task scheduling. *ACM Transactions on Architecture and Code Optimization (TACO)*, 7(1):1–31, 2010.