Cool and Save: Cooling Aware Dynamic Workload Scheduling in Multi-socket CPU Systems

Raid Ayoub Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego
La Jolla CA 92093-0404
{rayoub, tajana}@cs.ucsd.edu

Abstract— Traditionally CPU workload scheduling and fan control in multi-socket systems have been designed separately leading to less efficient solutions. In this paper we present *Cool and Save*, a cooling aware dynamic workload management strategy that is significantly more energy efficient than state-of-the art solutions in multi-socket CPU systems because it performs workload scheduling in tandem with controlling socket fan speeds. Our experimental results indicate that applying our scheme gives average fan energy savings of 73% concurrently with reducing the maximum fan speed by 53%, thus leading to lower vibrations and noise levels.

I. Introduction

The unprecedent demand for intensive computation makes the use of multi-socket CPUs a common design practice [1, 2]. However, this enormous level of integration coupled with high performance lead to high power density [3]. The primary consequence of such power density is the increase in thermal stress. Unfortunately, dissipating the high temperature requires a large and energy hungry cooling system which increases the cost and the size of the product. Furthermore, the increase in temperature has significant impact on leakage power and reliability, both of which are exponentially related to temperature [4]. Additionally, performance degrades as high temperatures magnify the interconnect delay [5].

The cooling in high-end servers usually relies on the forced convection phenomena to enhance the heat transfer between the heat sink and the ambient. As the CPU power density escalates, the air flow rate must enlarge at a similar rate. Unfortunately, the fan power increases substantially due to the cubic relationship between fan speed and its power [7]. The fan system power in high-end servers is as much as 80 Watts in 1U rack servers [7] and 240 Watts in 2U rack servers or more [1]. Figure 1 shows that fan power can reach up to 51% [22] of the overall server power budget. In addition to the increase in fan power, high fan speed introduces large noise levels in the systems. It is shown that the acoustic noise level increases by 10 dB as air flow rate increases by 50% [8]. This indicates that minimizing air flow to what is just needed for cooling is an essential metric to deliver energy efficiency at lower acoustic noise levels.

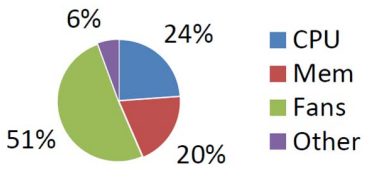

Fig. 1. IBM P670 Server power breakdown

In current systems CPU workload management and fan control operate independently. The threads on the CPU are managed by the operating system (OS) scheduler, while the fan control algorithms are governed by the CPU socket temperature. State of the art dynamic load balancing scheduling algorithms implemented in OS do not consider either thermal/cooling characteristics of the processors they run on, or the inherent power characteristics of the threads running in the system. We identify this as a big drawback since the heat being dissipated by these running threads actually contributes to the thermal profile of the system at any given point in time. These observations motivate us to propose a scheduling algorithm called '*Cool and Save*', which performs thread scheduling based on the dynamic characterization of the thermal profile of the active threads. The algorithm incorporates an air forced convective model to understand the interaction between thermal characteristics and the fans. It uses performance counters to predict workload induced power density, and therefore the likely temperature on the die with associated cooling costs. Such a model helps it to estimate the new thermal state and the cooling cost associated with any scheduling event. Based on this understanding, it is able to dynamically consolidate or spread the running threads, hence achieving an overall balanced thermal profile which results in higher energy savings due to lower fan speeds. We evaluate our system using benchmarks with varying runtime power characteristics, and show that *Cool and Save* is able to achieve up to 85% reduction in cooling related energy consumption compared to state of the art scheduling policies.

The rest of the paper is organized as follows: The related work is given in section 2. In Section 3 we described our fan modeling scheme, followed by Section 4, which focuses on our workload scheduling scheme. In section 5 the evaluation methodology and the experimental results are discussed, followed by the conclusions in section 6.

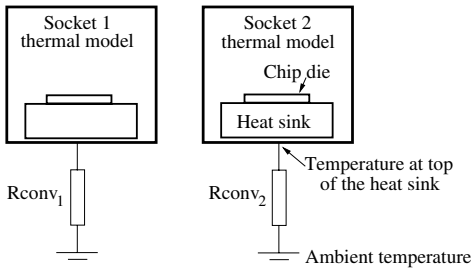

Fig. 2. Multi-socket thermal modeling

II. Related work

In recent years, number of processor level dynamic thermal techniques have been suggested. They can be broadly classified into reactive and proactive. Reactive techniques include clock gating [10], DVFS [12], activity migration [6], etc.. Recently introduced proactive techniques manage overheating by predicting the temperature and rescheduling the workload appropriately [21].

The authors in [11] suggest a fan control algorithm that manages temperature based on input from thermal sensors only. A class of techniques have been suggested to improve cooling efficiency at the data center level [13, 14]. The research in these techniques suggest the use of workload scheduling to help with the air circulation problem in the data center; hence, better cooling efficiency. However, these techniques can not be reused at the socket level since their air flow is highly contained. The authors in [7] propose a methodology for modeling the convection thermal resistance between the heat sink and ambient temperature as a function of the air flow rate, which we leverage in our work.

To the best of our knowledge, there is no prior work on cooling aware dynamic workload scheduling for a multi-socket/multi-core CPU system. The primary contributions of our work can be summarized as: 1) We incorporate a model of fan control with the socket level thermal model. 2) We propose a new cooling aware dynamic workload scheduling algorithm that combines workload scheduling and fan control. 3) We present a thorough evaluation and discussion of the proposed technique with benefits and overhead.

III. Air forced Thermal Modeling and Cost

To provide sufficient cooling to the sockets, each socket is normally associated with fans. For example, the Sunfire x4270 server has two sockets where each has two sets of fans [1]. Figure 2 shows a thermal model of two socket CPU. The modeling of the individual sockets can be done using an RC network similar to what is done in HotSpot [3]. To simplify the modeling, the heat path between the two sockets can be safely neglected since there is no effective conductive or convective heat path between them as they typically placed apart in the motherboard. To evaluate the fan speed as a function of the temperature, we use the model in Figure 2. The convective resistance, R_{conv}, between the top of the heat sink and the ambient temperature which can be computed as:

$$R_{conv} = \frac{1}{hA} \tag{1}$$

where A is the heat sink area and h is the convective heat transfer we use. Unfortunately, HotSpot does not model R_{conv} as a function of the air flow rate. To model air forced convection, we use parameter h, whose value increases with the increase in air flow rate, which lowers R_{conv}. We use the analysis in [7] to estimate the value of h as a function of air flow rate V:

$$h(V) \propto V^{\alpha} \tag{2}$$

where α is in the range of (0.8-1.0). Using Ohm's law one can see that the heat sink temperature increase with the increase in R_{conv}. Consequently, the temperature is related to the air flow rate as follows:

$$V \propto T^{-\alpha} \tag{3}$$

The air flow is proportional to fan speed [7]. As a result, the cooling cost of changing the air flow rate from V_1 to V_2 can be computed as:

$$\frac{P_{F2}}{P_{F1}} = (\frac{V_2}{V_1})^3 \tag{4}$$

where P_{F1} and P_{F2} represent the fan power dissipation at T_1 and T_2 respectively. Using (3) and (4), the cost of changing the heat sink temperature from T_{hs1} to T_{hs2} by changing fan speed can be estimated as follows:

$$\frac{P_{F2}}{P_{F1}} = (\frac{T_{hs1}}{T_{hs2}})^{3/\alpha} \tag{5}$$

where P_{F1} and P_{F2} represent the fan power dissipation at $T_{_{hs1}}$ and T_{hs2} respectively. This formula shows that optimizing the fan speed is an essential issue for power savings since reducing the temperature requires an increase in the order of $3 + \alpha$ in the fan power.

IV. Cooling Aware Dynamic Workload Scheduling

A. Motivation

Typical CPU socket level package includes heat sink as the primary component since it is responsible for dissipating most of the heat from the die. Temperature distribution of the heat sink exhibits low spatial variability and can be assumed to be an isothermal layer, [3], due to its high thermal conductance that strengthens the lateral heat flow and minimizes the spatial variations. This implies that the heat sink temperature is a function of the total power dissipated by the socket cores rather than specific power distribution of the cores on the die. As a result, the core temperature can be approximated as:

$$T_{core} = PRv_{core} + T_{hs} \tag{6}$$

where T_{core} is the core temperature, P is the core power dissipation, Rv_{core} is the core's vertical thermal resistance and T_{hs} is the heat sink temperature (temperature from the bottom of heat sink to ambient). In this equation we neglect the lateral heat flow between cores since the ratio between the core area to the die thickness is high [6]. This equation tell us that the core temperature is directly related to the core power and heat sink temperature. As the heat sink is isothermal, the hot core temperature would stay in the same range irrespective of the temperature of other cores under the condition of the same socket power.

TABLE I
IMPACT OF WORKLOAD ASSIGNMENT ON THERMAL BEHAVIOR

Socket workload	Heat sink temp.	Max core temp.
A ={10W, 5W, 5W}	55-56 ^{o}C	90.1 ^{o}C
B ={10W, 5W, 5W}	55-56 ^{o}C	90.1 ^{o}C
A ={10W, 10W}	55-56 ^{o}C	89.7 ^{o}C
B ={5W, 5W, 5W, 5W}	55-56 ^{o}C	75.2 ^{o}C

TABLE II
FAN COOLING SAVINGS

Socket workload	Air Flow Rate
A ={10W, 5W, 5W}	12 CFM
B ={10W, 5W, 5W}	12 CFM
A ={10W, 10W}	11.9 CFM
B ={5W, 5W, 5W, 5W}	1 CFM

We can save energy in cooling subsystem by intelligently distributing the workload across the CPU sockets. Lets assume a system of two sockets (A and B) where each has 4 cores, one fan per socket, 65nm technology, temperature threshold is 85 ^{o}C, heat sink with R_{conv} = 0.24 ^{o}C/W at 15 CFM and ambient temperature of 40 ^{o}C. For thermal simulations, we use HotSpot, which we extend to include the air flow based thermal and cooling model described in the previous section. Also, lets assume each socket is running 3 threads where each thread is dissipating power as follows: A ={10W, 5W, 5W}, and B = {10W, 5W, 5W}. Table I shows the impact of workload assignment on thermal stress. It is evident from this table that both sockets have similar heat sink temperature. The heat sink temperature is similar since the total power in each socket is the same, an issue that confirm our analysis. Furthermore, the maximum temperature across the two sockets is similar, as (6) predicted, since the hottest thread in both sockets dissipate 10W and the heat sink temperature is the same. The fan speed in both cases is the same due to the similarity in heat sink and core maximum temperature II. To minimize the cooling costs we can migrate the 10W thread from B to A and migrate the two 5W threads from A to B to maintain similar power in both cases. After this migration the workload in A and B will be: A ={10W, 10W}, and B = {5W, 5W, 5W, 5W}. This eliminates the hot spots in B completely while maintaining a similar heat sink and maximum core temperature in A compared to the previous thread assignment. This means that we can deliver around 47% savings by running the fan of socket B in idle state (e.g. 1 CMF) while keeping the socket A fan at a speed similar to the previous case. The results in table II confirm the cooling savings. The migration overhead is expected to be affordable since it is in the rage of μ seconds and infrequent. Clearly scheduling the workload while considering the cooling costs is an effective way to minimize the overall energy. In the following subsections we focus on developing a general approach for cooling aware scheduling.

B. Scheduling framework

To leverage the benefits of controlling both cooling and workload scheduling we implement our technique at the operating system (OS) level. Figure 3 depicts the operational framework of our approach. The input to the scheduler consists of thermal sensors at each socket, run time workload characterization and the incoming workload. Based on this set of inputs the OS scheduler decides whether to redistribute or consolidate the current workload through migrating some of the threads across the sockets. The rescheduling period is large

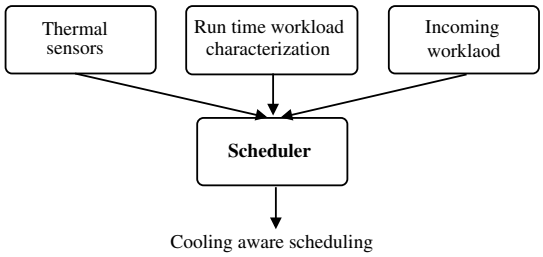

Fig. 3. Cooling aware scheduling framework

(order of seconds) since the heat sink has a large thermal time constant. This implies that the overhead of rescheduling the workload is negligible since it is in the range of μ seconds. The rescheduling overhead primarily comes from the OS, transferring the state of the threads across the cores and any L2 cache misses. Incoming workload is always assigned to the sockets with lower fan speeds.

C. Scheduling mechanism

Our **Cooling Aware Scheduling policy (CAS)** lowers cooling costs through migrating some of the threads across the available sockets. Since the overheating occurs only when the core temperature surpasses a given threshold, we classify the threads into *hot* and *cold* where the cold threads are the ones that have their temperature below the threshold. At each rescheduling point, the algorithm traverses the running threads in the system and migrates the workload based on the the following rules.

- If the current thread is a *cold* one, the scheduler may initiate migration for this thread only if it is running on a socket that has a fan running faster than the fan of another socket available in the system to reduce cooling cost. The ideal situation is when the destination socket stays below the critical temperature after the migration.
- If the current thread is a *hot one*, and it is the only hot thread in the socket then it can be moved to another socket with a running fan if cooling savings can be maintained. To maintain savings, we should consider moving one or more of the cold threads from the destination socket to the source to compensate for the power of the current hot thread. We denote such migration as *hot thread consolidation*. Such migrations allow turning off the source fan while maintaining the destination fan at a similar speed.
- If the current thread is a *hot one* and it is not the only hot thread in the socket, then it is migrated to a socket with lower fan speed to achieve better cooling balancing. If the destination socket is fully utilized then we swap the hot thread with cold threads from the destination socket. We denote such migration as *hot thread spreading*.

If additional threads have to be migrated as a results of migrating the current thread in the traversing list, those threads are marked and excluded from the traversing process. After each migration event the thermal state of the system is updated and saved. The complexity of this algorithm is O(n) leading to negligible runtime overhead. Two important issues remain: 1) how to estimate the new thermal state after migrating a thread across sockets so that we know ahead of time

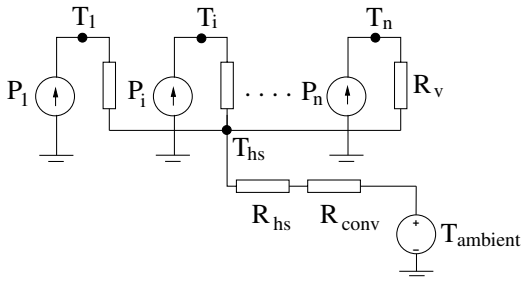

Fig. 4. Single socket thermal modeling

if migration is beneficial and 2) how to quantify the associated cooling costs of such migration.

To get the new thermal state we use circuit theory to calculate it. We use the steady-state version of the equivalent thermal network as it can deliver adequate estimates with affordable complexity. This assumption is justifiable due to the large differences in the time constants between the cores and the heat sink. Figure 4 shows an example of the equivalent thermal model of a multi-core socket [3]. In this figure, the current sources represent the average power dissipation of the individual units (cores and L2 cache) , Rv_i is the vertical thermal resistance of the chip components including the interface resistance, R_{hs} represents the sum of heat spreader and the heat sink resistances and R_{conv} is the convection resistance. The heat sink and heat spreader are modeled with one node to simplify the run time computations since these layers are almost isothermal. For a given assignment, the value of T_{hs} can be computed using superposition theory as follows:

$$T_{hs} = (R_{hs} + R_{conv}) \sum P_i \tag{7}$$

From this equation it can be seen that adding or removing a thread alters the value of T_{hs} by a value that is equivalent to $(R_{hs} + R_{conv})(P_{core} + P_{core\_L2})$ where P_{core} and P_{core_L2} corresponds to the core power and the fraction of L2 cache power that is consumed by the current thread respectively. Determining the new value of T_{hs} is key to estimate the cooling costs and to serve as a reference for the system's new thermal state after migration.

We next show how to estimate the value of T_{hs} at run time. To do so, we first need to estimate the power of the individual cores and L2 cache. The formula to calculate the power of a given core, P_i, can be extracted from equation (6) as follows.

$$P_i = \frac{T_{hs} - T_i}{Rv_{core}} \tag{8}$$

For estimating the power dissipation due to accessing L2 cache, we multiply the total accesses to the cache by the power of each access. The cache accesses can be extracted easily using performance counters. Such *workload characterization* information can be collected easily by the OS at runtime. Using (8) and L2 power we can estimate the heat sink temperature as:

$$T_{hs} = \frac{R_{hs} + R_{conv}}{1 + N\frac{R_{hs}+R_{conv}}{R_{core}}}(P_{L2} + \sum_{i=1}^{N} \frac{\beta T_i}{R_{core}}) \tag{9}$$

where N is the number of cores, P_{L2} is the total power dissipation by L2 cache and β is a correction factor to allow estimating the average temperature of the cores form the thermal sensors reading, a factor which can be estimated at the design time. The heat sink temperature modeling can be utilized to build a cooling cost function as explained in the following discussion. It should be noted that such estimation does not need to be highly accurate since we only need to know relative comparison between the threads.

Cooling cost: As migrating threads across sockets alters the T_{hs}, the fan speed needs to be changed to compensate for the change in temperature. The cooling savings of changing the workload can be evaluated using equation (5) and (9) as:

$$\frac{P_{F2}}{P_{F1}} = (\frac{T_{hs1}}{T_{hs2}})^{3/\alpha} \tag{10}$$

where T_{hs1} and T_{hs2} represent the heat sink temperature before and after changing the workload. The heat sink temperature values can be computed using thermal sensors value and performance counters as described earlier.

TABLE III
SIMULATION PARAMETERS

Parameter	Value
Issue width	4
ROB	128
Functional units	4 IntALU, 1 IntMult/Div 1 FPALU, 1 FPMult/Div
Branch predictor	Tournament 2048 local predictor 8192 global predictor
BTB	2K entries, 1 way
LSQ	32
L1 I-cache	32KB, 4 ways, 32B blocks, 1 cycle
L1 D-cache	32KB, 4 ways, 32B blocks, 1 cycle
L2	4MB, 8 ways, 64B blocks, 12 cycles
Memory latency	200 cycles

V. Evaluation

A. Methodology

For the experimental evaluation we assume a platform that consists of dual sockets where each socket has 4 cores (each core is a 4-issue ALPHA like processor that runs single thread). Table III gives the simulation parameters that we have used in our simulations. We utilized three simulators, M5 [16], Wattch [17], and HotSpot-4.0 [3]. The M5 simulator is used to obtain the architectural level performance simulation. The M5 results are fed into Wattch to obtain the power values of the processor functional units. The power values are then used to estimate the temperature through the HotSpot simulator. We further extend M5 to emulate our CAS algorithm in the OS scheduler. The OS initiates CAS every 3 sec which is sufficient since the heat sink thermal time constant is in the range of (15-30 sec). To compare our scheme against state of the art OS scheduling we also emulated *Dynamic Load Balancing (DLB)* in M5. The *DLB* enhances the utilization of the system resources by scheduling the workload in a way that minimizes the difference in task queue length across the individual sockets as well as the individual cores. For the base line scheduling, DLB, we implement a typical fan control algorithm using industry standard closed loop feedback controller (PI, Proportional-Integral) that adjusts the fan speed based on the thermal sensors readings. When the temperature is below the threshold the fan speed is set to idle speed. In case of overheating, the fan algorithm adjust the fan speed accordingly to ensure that the cores temperature do not exceed the thermal threshold.

Figure 5 shows the floorplan obtained by scaling ALPHA 21264 processor into 65nm technology. In our simulations we

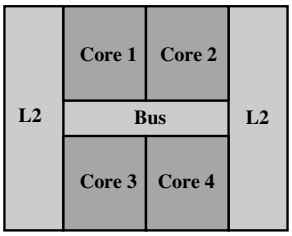

Fig. 5. Socket floorplan

TABLE IV
BENCHMARKS CHARACTERISTICS

Benchmark	Avg. dynamic power (W)	Power Standard deviation
gcc	4.12	1.8
gzip	5.51	1.65
swim	6.35	1.77
bzip2	7.9	2.25
crafty	9.11	0.66

use processor clock speed of 2GHz. L2 cache area is estimated based on Cacti simulation tool [18]. In our experiments we use a die thickness is 0.2mm. To account for CPU cores leakage power temperature dependency, we used the second-order polynomial model that is proposed in [19]. We extracted the model coefficients empirically based on the given normalized leakage values. To estimate the leakage values for 65nm technology we incorporate the reported value of leakage power density ($0.5W/mm^2$ at 383K [20]) in the second-order polynomial model. For the crossbar bus power consumption, we scaled the power depending on the activity in the cores and L2 cache. In our results we also account for the temperature warmup, as the heat sink has a longer time constant than the die. We assume the critical temperature as 85 ^{o}C which is common to such designs [21]. We use a socket package with convection thermal resistance equal to 0.18 ^{o}C/W at air flow of 23 CFM which is in the range to what is being deployed in the state of the art quad core Xeon processor used in server platforms [23].

We use benchmarks from the SPEC2000 suite for our workloads (see Table IV). We selected a set of benchmarks that exhibit various levels of power intensity to represent real life applications. We ran each benchmark for a representative interval of 5 seconds and then repeated the bench execution to obtain a total execution time of 300 seconds which is sufficient to evaluate our polices. To better evaluate our policies under various workload conditions, the selected benchmark combinations (see Table V) have different average power intensity and variance.

B. Results

TABLE V
WORKLOAD LIST

Workload #	Benchmarks (socket 1/socket 2)
1	{crafty+gzip+gcc}/{crafty+gzip+gcc}
2	{bzip2+gzip+swim+swim}/{crafty+gzip+gcc}
3	{bzip2+swim+gcc+gcc}/{bzip2+swim+gzip+gzip}
4	{crafty+crafty+swim}/{gzip+swim+swim}
5	{bzip2+bzip2+bzip2+bzip2}/{gcc+gcc+gcc+gcc}

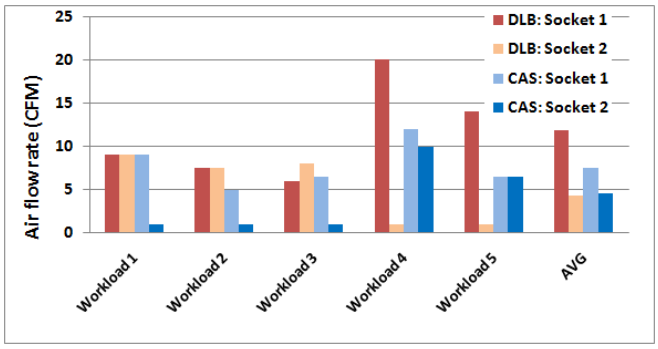

Fig. 6. Air flow rate in the multi-socket system

The reported results show that our policy, CAS, outperforms the state of the art DLB drastically in energy savings and minimizing air flow rate that reaches as high as **85%** and **53%** respectively. Minimizing these metrics has a big advantage on lowering the overall cooling costs and acoustic noise.

Figure 6 shows the air flow rate breakdown across two sockets when using the DLB and CAS polices for all the workload given in table V. We report the results in air flow rate instead of absolute fan speed since this is a standard metric for characterizing the fans. The air flow rate is proportional to fan speed. The first observation is that CAS is able to reduce the maximum air flow rate by as high as **53%**. This figure shows cases for two sources of air flow improvements, hot cores *consolidation* (workloads 1, 2, 3) and *spreading* (workloads 4, 5). For the consolidation case, we can observe that CAS is able to maintain cooling with half of the fans only running at similar speed to the case of DLB. To explain this, lets take the case of workload 1:$\{crafty + gzip + gcc\}/\{crafty + gzip + gcc\}$. Before applying CAS, the power is balanced across the two sockets where each is required an air flow rate of 9 CFM. To improve cooling efficiency, CAS migrates the hot thread, $crafty$, from socket 2 to 1 and the two cold threads, gzip and gcc, from socket 1 to 2 to maintain power balancing. As a result, socket 1 will be executing $\{2crafty\}$ while socket 2 runs $\{gcc+gzip+gcc+gzip\}$. Although socket 1 is running two hot threads, its fan speed stays almost the same since the socket total power is maintained at the same level. This indicates that thread consolidation is an effective way to improve cooling efficiency.

The CAS algorithm performs hot threads spreading when there is a strong imbalance in the socket power the workload

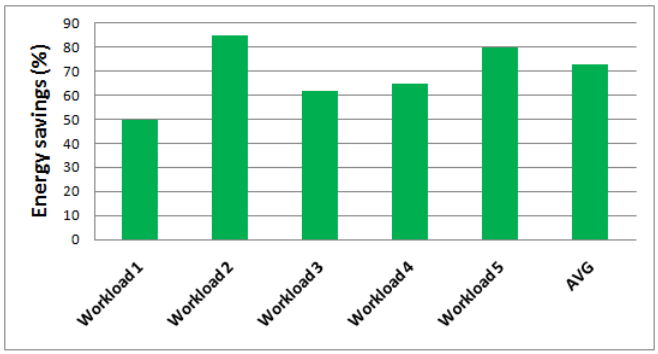

Fig. 7. Energy savings over DLB

5. Prior to applying CAS the workload is assigned to the two sockets as: $\{bzip2 + bzip2 + bzip2 + bzip2\}/\{gcc + gcc + gcc + gcc\}$. In this case, socket 2 fan is in idle state (gcc threads are cold) while the air flow in socket 1 is at a high rate. In this case, the power dissipation in socket 1 is about double what is in socket 2. Such power imbalance between them leads to cooling inefficiency due to non-liner relation between power cost and temperature reduction equation (5). To lower the cooling costs, CAS algorithm balances the power across the two cores by migrating two instances of $bzip2$ to socket 2 and two instances of gcc to socket 1. From the results, it can be seen that the air flow is lowered from one fan running at 20 CFM to two fans running at range close to 10 CFM which leads to drastic power savings.

Figure 7 shows the cooling energy savings of applying CAS over the DLB. The cooling savings is calculated by applying equation (4) to the results of Figure 6. It is clear from the results that applying CAS results in significant energy savings; the improvement as high as 85%. The appreciable improvement comes from applying our consolidating or spreading the hot threads as required. For the consolidating cases, the savings come from running one of the sockets fans in idle mode while not increasing the speed of the running fan. The savings in case of workload 2:$\{bzip2 + gzip + swim + swim\}/\{crafty + gzip + gcc\}$ is the highest, 85%, that is attained by migrating $crafty$ to socket 1 and the two instances of $swim$ to socket 2. The improvement is high in this case, since it reduces the total power dissipation in socket 1 while allowing socket 2 to run its fan in idle mode since its temperature become below the threshold. When CAS performs hot threads spreading, the energy savings come from exploiting the nonlinearity between cooling cost and temperature reduction. This can be seen in the case of workload 5, where the savings is 80%. The overhead of migrating the thread and running CAS is expected to be negligible since it is infrequent (order of seconds). As a results, one can conclude that employing CAS algorithm is a highly effective approach for reducing the cooling costs.

Overhead of CAS: In our experiments we run CAS every 3 seconds to reschedule the workload if required. The total overhead of each run including the migration overhead is less than 5ms on average. Consequently, the overall overhead is below 1% which is negligible.

VI. Conclusion

In this work, we have proposed a new workload scheduling technique to minimize the overall energy costs in multi-socket CPU platforms. Our algorithm incorporate continuous temperature measurements, workload characterization and cooling cost estimation to guide the OS schedular in allocating the workload in a thermally sensitive matter. Our algorithm deliver cooling savings through consolidating or spreading the hot threads depending on the system's current thermal state. We also developed a simple yet accurate run time cooling cost estimator to evaluate the benefits of workload rescheduling. This algorithm exhibits negligible overhead since it has a complexity of O(n) and runs infrequently. Our results show that applying our algorithm delivers energy savings of 73% concurrently with reducing the maximum fan speed on an average by 53%.

Acknowledgements

This work has been funded by NSF Project GreenLight grant 0821155, NSF SHF grant 0916127, GSRC grant 2003-DT-660, UC Micro grant 08-039, Texas Instruments, Sun Microsystems, and Cisco. We thank Mr. Gaurav Dhiman for his valuable comments and suggestions to improve this work.

VII. References

[1] www.sun.com/servers/x64/x4270/
[2] www-03.ibm.com/systems/x/hardware/rack/x3550m2/
[3] K. Skadron, M. Stan, K Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan Temperature-Aware Microarchitecture: Modeling and Implementation *TACO*, pages 94ï£¡-125, 2004.
[4] M. Pedram and S. Nazarian. Thermal modeling, analysis, and management in vlsi circuits: principles and methods. Proceedings of the IEEE, pages 1487–1501, 2006.
[5] A. Ajami, K. Banerjee, and M. Pedram. Modeling and analysis of nonuniform substrate temperature effects on global interconnects. *IEEE Trans. on CAD*, pages 849–861, 2005.
[6] S. Heo, K. Barr, and K. Asanovic. Reducing power density through activity migration. *ISLPED*, pages 217–222, 2003.
[7] M. Patterson. The effect of data center temperature on energy efficiency. *ITHERM*, pages 1167–1174, 2008.
[8] R. Lyon, and A. Bergles. Noise and Cooling in Electronics Packages. *IEEE Trans on Components and Packaging Technologies*, Vol. 29, No.3, pp. 535- 542, 2006.
[9] J. Choi, C. Cher, H. Franke, H. H. A., Weger, and P. Bose. Thermal-aware task scheduling at the system software level. *ISLPED*, pages 213–218, 2007.
[10] G. Hinton, D. Sagar, M. Upton, D. Boggs, D. Carmean, and P. Roussel. The microarchitecture of the pentium 4 processor. *Intel Technology Journal*, 5(1):1–12, 2001.
[11] H. Chiueh, L. Luh, J. Draper, and J. Choma. A Novel Fully Integrated Fan Controller for Advanced Computer Systems. *SSMSD*, paged 191-194, 2000.
[12] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. *HPCA*, pages 171–182, 2001.
[13] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. *USENIX*, pages 61ï£¡-75, 2005.
[14] Q. Tang, S. Gupta, G. Varsamopoulos. Thermal-aware task scheduling for data centers through minimizing heat recirculation *ICCC*, pages 129–138.
[15] K. Skadrony, T. Abdelzahery and M. Stanz. Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. *HPCA*, pages 17–28 2002.
[16] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, 2006.
[17] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *ISCA*, pages 83–94, 2000.
[18] D. Tarjan, S. Thoziyoor, and N. Jouppi. Cacti 4.0. *Technical report, HP Laboratories*, pages 1–15, 2006.
[19] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif. Full chip leakage estimation considering power supply and temperature variations. pages 78–83, 2003.
[20] P. Bose. Power-efcient microarchitectural choices at the early design stage. *Workshop on Power-Aware Computer Systems*, 2003.
[21] A. Coskun, T. Rosing, and K. Gross. Proactive temperature management in MPSOC. *ISLPED*, pages 165–170, 2008.
[22] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler and T. Keller, Energy management for commercial servers *IEEE Computer*, pages 39–48, 2003.
[23] Quad-Core Intel Xeon Processor 5300 Series: Thermal/Mechanical Design Guidelines