Exploiting Task Temperature Profiling in Temperature-Aware Task Scheduling for Computational Clusters

Daniel C. Vanderster, Amirali Baniasadi, Nikitas J. Dimopoulos

Department of Electrical and Computer Engineering,
University of Victoria,
P.O. Box 3055 STN CSC,
Victoria, B.C. V8W 3P6
Canada
{dvanders, amirali, nikitas}@ece.uvic.ca

Abstract. Many years of CMOS technology scaling have resulted in increased power densities and higher core temperatures. Power and temperature concerns are now considered to be a primary challenge for continued scaling and long-term processor reliability. While solutions for low-power and low-temperature circuits and microarchitectures have been studied for many years, temperature-awareness at the computational cluster level is a relatively new problem. To address this problem, we introduce a temperature-aware task scheduler based on task temperature profiling. We study the task characteristics and temperature profiles for a subset of SPEC'2K benchmarks. We exploit these profiles and suggest several scheduling algorithms aimed at achieving lower cluster temperature. Our findings show a clear trade-off between the overall queue servicing time and the cluster peak temperature. Whether the temperature reductions achieved are worth the extra delay is left to the designer/user to decide based on the case by case performance restrictions and temperature limitations.

1 Introduction

In recent years, the issues of power dissipation and energy consumption have come to the forefront of the minds of system designers [1] [2]. One specific area where this issue is relevant is in the realm of computational clusters [3]. These large systems often feature hundreds of servers and processors. Exploiting highly integrated high power density servers and processors in such systems has resulted in new thermal challenges. In fact, under new semiconductor technologies power density of the microprocessor core has exceeded 200 $^{\mathrm{W}}/_{\mathrm{cm}^2}$. Such high power densities can result in high temperatures which in turn can cause transient faults or permanent failures. Reducing the heat and lowering the cluster temperature is therefore a vital and important challenge. To address this challenge, two different approaches may be considered: First, we can develop more effective and often expensive ventilation and cooling systems to remove more heat at a higher rate.

Second, we can develop new design techniques at several levels to reduce the production of the excess heat.

As effective cooling mechanisms become more expensive, it is important that designers develop temperature reduction techniques at different levels including scheduling. This work aims at exploring such solutions. We introduce the *Profile-based Temperature-aware Scheduler*, or *PTS*, to address this problem at the scheduler level. PTS relies on identifying and using processor-task combinations resulting in better temperature conditions in the computational cluster.

In particular, we make the following contributions:

– First, we study task temperature profiles. A task temperature profile indicates the amount by which a given task will raise the temperature of a host processor. We use our findings to differentiate between hot and cold tasks.
– Second, we use task temperature profiles of a subset of the SPEC'2K benchmarks and show that early knowledge of such profiles can be used to produce a temperature-aware schedule reducing the cluster peak temperature.

The rest of the paper is organized as follows. In section 2 we present task temperature profiling. In section 3 we present and evaluate our scheduling policies. In section 4 we discuss some related work. In section 5 we offer concluding remarks.

2 Task Temperature Profiling

In this study, a task temperature profile is a time series measure of the amount by which a task will raise the temperature of a host processor. Our observations have shown that not all tasks generate the same amount of heat on a given processor. This could be explained by the task's behaviour including the time it spends performing integer or floating point calculations, reading and writing to memory, or waiting for asynchronous events. For example, a processor bound task can generate more heat compared to an I/O bound task.

Based on how a task impacts processor temperature, we can observe a spectrum of tasks ranging from *hot*, or high-temperature, tasks to *cold*, or low-temperature, tasks.

Our goal is to reduce the cluster peak temperature by assigning hot tasks to cold processors. To achieve this, a temperature-efficient schedule should exploit the variance in the cooling ability of the processors in a cluster. We have observed this variance is related to the physical proximity of a processor relative to a cooling vent, and is indicated by the fan-input temperature of the chassis housing the processor. We therefore classify processors based on their fan-input temperatures into groups, or *processor classes*. For example, the 168 processors in a typical rack may be classified into six processor classes, one for each of the rack-mounted chassis[1].

[1] In a typical computational cluster, the chassis closest to the floor vents will have the lowest fan-input temperature, and thus the best cooling ability.

In order to measure the task temperature profiles and classify the processors, we developed a tool which monitors several temperature metrics for a task executing on a blade in an IBM BladeCenter. Each blade contains two Intel Xeon processors and with 2 gigabytes of memory and runs Red Hat Enterprise Linux AS 3. The measurements are performed using the on-chassis and on-board thermal sensors present on the IBM BladeCenter chasses and blades. A monitoring daemon periodically polls the BladeCenter management interface using the Simple Network Management Protocol (SNMP). The measured metrics include the processor core temperatures of the both processors in a blade, the chassis fan-input temperature, the temperature of the management console located at the rear of the chassis, and the chassis blower speeds measured as of percentage relative to maximum blower RPM. To avoid interference from background applications, both processors in a blade are exclusively reserved for the duration of the tests.

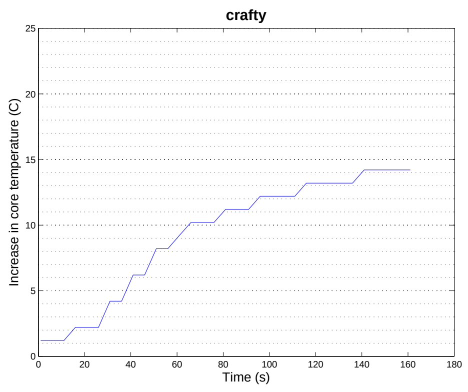
We generated task temperature profiles (shown in Figure 1) for a subset of the SPEC'2K benchmarks. Note that a single script executing each benchmark in succession is used to ensure the execution environment is identical for each benchmark. Prior to the execution of each benchmark, the processor is idled for 300 s to allow the system to return to its idle temperature. While these plots represent the profiles attained with a single execution of the benchmarks, further experiments on other processors demonstrated similar profiles.

In Figure 1 we see that longer benchmarks (e.g., *mcf* and *swim*) show higher peak temperatures. Accordingly, tasks with shorter runtimes (e.g., *crafty* and *gzip*) cause less heat and therefore lower peak temperatures. *Wupwise* has plateaued at between 20 °C and 25 °C, which is a lower peak compared to *mcf* and *swim*. It is notable that some of the tasks (e.g., *crafty*, *gzip*, and *vortex*) appear not to have reached a steady-state temperature – it is reasonable to expect that had those benchmarks continued executing for longer, their temperatures would have continued to rise. While noting this situation, we have chosen not to increase their execution times in this study because we are interested in varied profiles, including those that have and do not have a steady state. However, it is clear that further tests of processor, I/O, and memory bound tasks are needed to explore the full spectrum of temperature profiles.

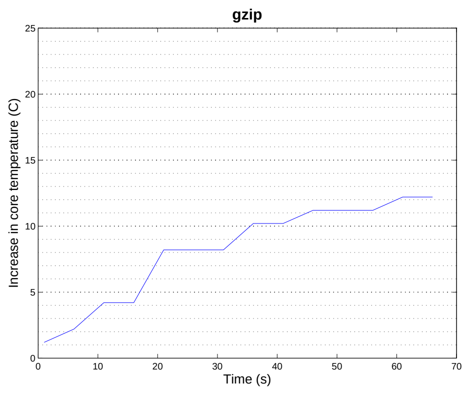
If we assume that our selection of SPEC'2K benchmarks represents a set of tasks queued at a computational cluster, then their temperature profiles demonstrate that a computational cluster will have a spectrum of tasks ranging from hot to cold. The hot tasks are those having the highest peak temperature, while the cold tasks are those having the lowest peak temperature. We exploit this spectrum of task profiles to come up with a schedule which minimizes cluster peak temperature.

3 Temperature-Aware Scheduling

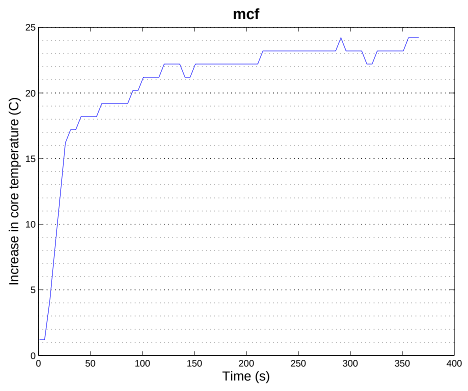
Our goal is to decrease the cluster peak temperature by distributing tasks amongst the processors so that hot tasks are assigned to processors with the



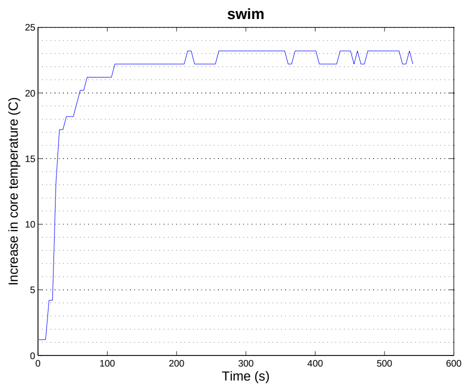
(a)



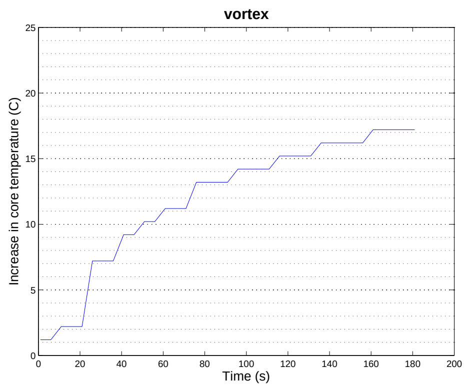
(b)



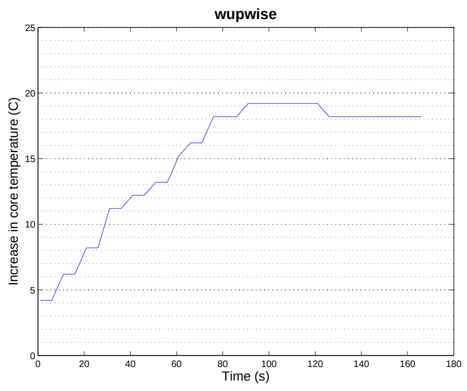
(c)



(d)



(e)



(f)

Fig. 1. Task temperature profiles for the SPEC'2K benchmarks (a) *crafty*, (b) *gzip*, (c) *mcf*, (d) *swim*, (e) *vortex*, and (f) *wupwise*.

best cooling ability. Our scheduling techniques rely on the assumption that the task temperature profiles are relatively invariant between subsequent executions of similar tasks. We assume that there exists a mechanism that can recognize the similarity of tasks by comparing task metadata including the submitting user, the task name, and invocation arguments. When a submitted task is recognized by this mechanism, a previously measured profile is used to schedule the task. When a new or unrecognized task is submitted to the scheduler, its metadata and temperature profile are recorded for subsequent executions of the task. Additionally, we simplify the environment by assuming that task preemption (and subsequent migration) at the batch scheduler level is disallowed[2]. All processors are assumed to be equal in performance.

3.1 A Profile-Based Temperature-Aware Scheduler

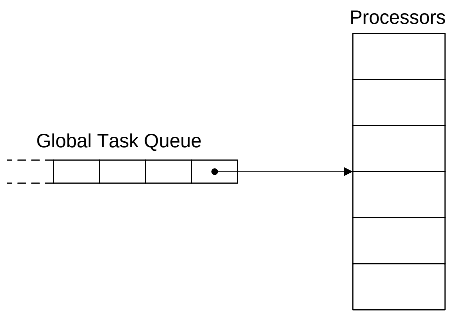


Fig. 2. Schematic of a simple scheduler having a single queue which assigns tasks to any processor in the cluster.

A simple computational cluster scheduler, shown in Figure 2, features a single queue which allocates jobs to all processors. Since there is no notion of temperature awareness, hot tasks may be assigned to the hottest processor, resulting in a high peak cluster temperature. The PTS scheduler, shown schematically in Figure 3, divides the processors into processor classes based on their cooling ability as described by their fan-input temperatures. The processors in each processor class are managed by a first-come-first-served task queue specific to the processor class. A task director assigns each task to a processor class queue by inspecting the temperature profiles and sorting the tasks in the global queue

[2] By assuming an absence of task preemption, we not only simplify the problem for simulation, but also allow the provided solution to be applicable to task sets in which the majority of tasks do not support checkpointability (which is common in our experience). A derivative adaptive scheduling strategy that re-evaluates and reallocates jobs is clearly possible in environments supporting preemption.

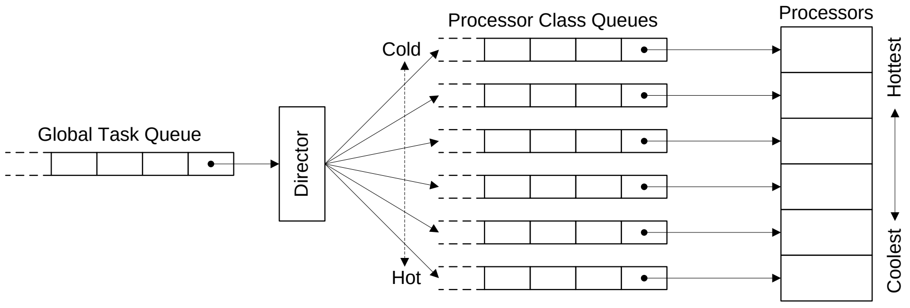


Fig. 3. Schematic of the PTS scheduler, featuring a director which sorts tasks in the global queue into sub-queues ranging from cold to hot. Each of the sub-queues assigns tasks to a processor in its associated processor class.

by their peak temperature. The task director allocates the tasks in the sorted global queue by assigning an equal number of tasks to each processor class queue starting with the coolest processor class. In general, this results in the hot tasks being assigned to the coolest processors and the cold tasks being assigned to the hottest processors.

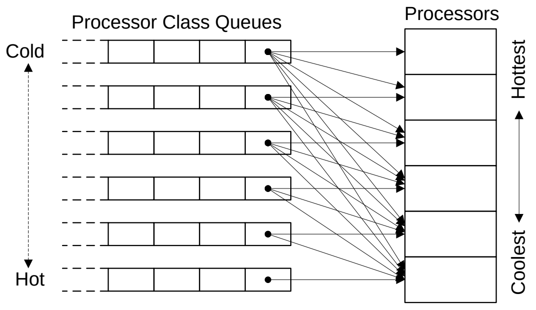


Fig. 4. Schematic of queue-processor relationship in the PTS-FM scheduler, which allows tasks to favourably migrate into colder idle queues.

Because the queues in Figure 3 will not necessarily empty at the same rate, the division of tasks equally into separate queues may result in an increase in the global queue-servicing-time. For example, there may be a number of tasks waiting to be executed on the hottest processors, while some colder processors are idle because their corresponding queues are empty. Recognizing that cold tasks may be assigned to any of the processor classes, we can allow these tasks to *favourably migrate* into colder queues. Namely, in the case that a queue Q is empty, we allow a colder task to migrate into Q (where "colder" implies that the processor class associated with Q has a lower fan-input temperature than that of the task's original queue). This technique, designated PTS with Favourable Migration (PTS-FM), is shown in Figure 4.

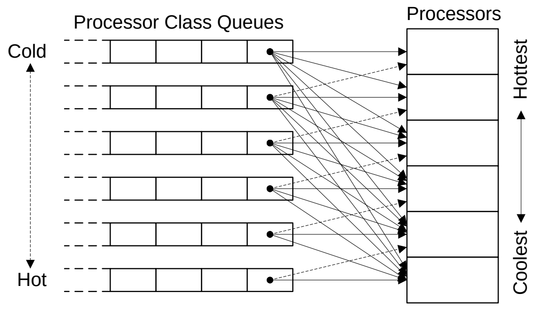


Fig. 5. Schematic of queue-processor relationship in the PTS-UM$_n$ scheduler, which allows tasks to unfavourably migrate into a tunable number (n) of hotter idle queues, in addition to the favourable migration into colder idle queues. PTS-UM$_1$ is shown.

To further increase processor utilization, the scheduler can be configured to allow the thermally unfavourable migration of tasks to a tunable number (n) of hotter idle queues. Shown in Figure 5, the PTS with Unfavourable Migration (PTS-UM$_n$) strategy realizes a trade-off between an increase in the peak cluster temperature and a decrease in the queue servicing time. The dotted lines in Figure 5 represent the allocation of tasks to thermally unfavourable processor classes. The shown PTS-UM$_1$ strategy allows tasks to unfavourably migrate by only one processor class. By increasing the allowed degree of unfavourable migrations, a further decrease in execution time will be realized along with a corresponding increase in the peak temperature. In the extreme case, where all of the queues can allocate tasks to all of the processor classes, the scheduler becomes a simple first-come-first-served (FCFS) scheduler.

For each scheduling strategy we measure the peak processor temperature as well as the overall queue servicing time (the time required to execute all tasks in a given queue). In all cases we compare our results with a baseline policy where the FCFS algorithm services the global queue in the order that tasks have arrived (referred to as the FCFS$_6$ policy). Additionally, we compare the PTS strategies to a simple FCFS-based power saving strategy, where the hottest processors are simply turned off. In the results below, these are referred to as FCFS$_5$ through FCFS$_1$, where the subscript represents the number of available processors.

It should be noted that in the following simulations we assume that the director has proceeded through a training period and all task temperature profiles are known. However, in practise, when the director receives a new, unrecognized, task, a conservative strategy could be followed while its temperature profile is recorded. Specifically, a conservative policy would specify that new tasks should be allocated only to the coldest processor class. This ensures that the peak cluster temperature will not be negatively affected by the execution of a hot task on a hot processor, but possibly increases the time that the task would have to wait in the queue. By employing the favourable migration strategy, this overhead may be decreased by executing the new task earlier on one of the marginally hotter processors.

3.2 Simulation Framework

In order to evaluate the performance of the presented scheduling algorithms, we have developed a cluster simulator within the SimGrid[3] framework [10]. To simplify the simulation, we model a simple six-processor cluster, with each processor representing a blade chassis in a typical six-chassis IBM BladeCenter. Each processor has an associated fan-input temperature; the fan-input temperatures vary linearly from 18.6 °C to 23.8 °C (corresponding to the temperatures measured for a six-chassis BladeCenter). Note that the maximum steady-state operating temperature of modern microprocessors is typically around 60 °C.

3.3 Results and Discussion

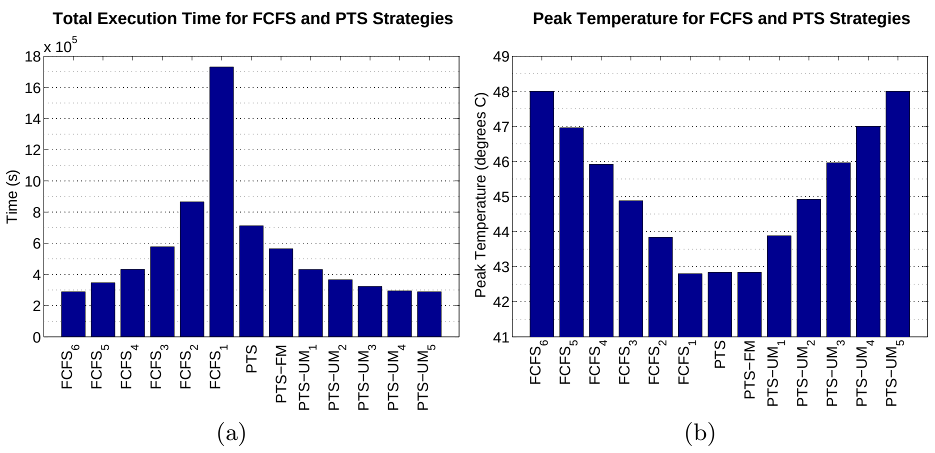


Fig. 6. Results for the FCFS and PTS scheduling strategies: (a) overall queue servicing time and (b) cluster peak temperature.

Figure 6 presents the total execution time and peak temperature results of the scheduler simulations having a global queue length of 6000 tasks (1000

[3] SimGrid provides a framework for task scheduler simulations. Tasks are characterized by a cost (the run-time) and processors are weighted with a relative performance.

instances of each of the six profiled SPEC'2K benchmarks). As expected, the $FCFS_6$ strategy has the fastest queue execution time of 288852 s, which represents nearly 100 % processor utilization, yet also results in the highest peak temperature (48.0 °C). The effect of turning off hot processors is shown in $FCFS_5$ through $FCFS_1$. In these results we see that the execution time increases proportionally to the number of processors turned off. Further, the peak temperatures decrease as the hotter processors are turned off.

By incorporating temperature-awareness into the scheduler, PTS results in the lowest temperature (42.8 °C, which corresponds to the ideal placement of the workload on the processors) but increases execution time to 712000 s, indicating that a number of processors were idle for a large portion of time. By allowing favourable migration, the PTS-FM strategy maintains the optimal temperature, but compared to PTS improves the execution time to 564616 s.

Finally, the strategies denoted by $PTS\text{-}UM_1$ through $PTS\text{-}UM_5$ demonstrate that by taking a more conservative approach, we can allow for faster execution times at the expense of higher temperatures. When we compare the execution times of $FCFS_n$ and $PTS\text{-}UM_n$ strategies that have similar peak temperatures, the $PTS\text{-}UM_n$ method performs better. For example, both $PTS\text{-}UM_3$ and $FCFS_4$ provide a decrease in peak temperature of approximately 2 °C. However, $PTS\text{-}UM_3$ achieves this at a slight cost in execution time in comparison to the 100000 s penalty created by $FCFS_4$.

It is important to note that the numerical results shown here are highly dependent on the cluster configuration and task profiles used. For example, the overall peak temperature reduction from 48.0 °C to 42.8 °C corresponds to the best possible placement with this set of tasks and processors. When used in environments with more diverse task temperature profiles, the PTS strategies are expected to achieve more dramatic temperature reductions.

As presented, the PTS schedulers can be used to decrease the cluster peak temperature. In cases of uncompromised temperature performance, system designers are encouraged to use the PTS-FM strategy, as it realizes the optimal temperature while minimizing the possible execution time (i.e. any decrease in the execution time would have required an increase in the peak temperature). In cases where the system designer is satisfied with a sub-optimal peak temperature, one of the $PTS\text{-}UM_n$ strategies can be used to improve execution time. The $PTS\text{-}UM_n$ strategy thus provides a mechanism for system designers to tune the time/temperature performance trade-off according to institutional priorities.

4 Related Work

Temperature-aware task scheduling for the computational cluster is a relatively new field of study. Bianchini and Rajamony presented a review of the energy and power management issues in computational clusters [3]. Much of the initial work in this area has been performed by Hewlett Packard ([4]-[7]). For example, in [6] Moore et al. present algorithms that leverage a thermodynamic formulation of steady-state hot- and cold-spots to achieve up to a factor of two reduction in cooling costs.

Patel et al. have presented a workload placement strategy for global computational Grids in which computational facilities are assigned energy-efficiency coefficients and workloads are allocated to the cluster having the best energy characteristics [8]. Weissel and Bellosa have developed OS-level power and thermal management methods that can be used to improve the thermal characteristics of the data center [9].

A recent study by Kurson et al. at IBM T. J. Watson Research Center investigated the decrease in on-chip temperatures seen while using thermally-aware thread scheduling [12]. Their *MinTemp* scheduling policy effectively lowers on-chip temperature by selecting the thread which has the lowest temperature for the current cycle's hottest thermally critical block.

In our work, we study task temperature profiles and exploit them to develop a temperature-aware data center task scheduler. This approach operates at the macroscopic system level to discover thermally beneficial workload placements within the data center.

5 Conclusions

We introduced temperature-aware scheduling policies for computational clusters. We used a task's temperature profile to quantify a task's heat producing capacity and to differentiate between cold and hot tasks. Our policies use different approaches to trade queue servicing time for lower peak temperatures. With the cluster configuration and temperature profiles we used, we conclude that a relatively balanced scheduling policy can effectively reduce cluster peak temperature at the expense of an increase in the queue servicing time.

In the future we plan to improve this work by evaluating the presented strategies using a more diverse set of tasks. Additionally, we will introduce the notion of task temperature profiling into the knapsack-based scheduler [11] in order to determine the potential for temperature-awareness in the presence of complex Quality-of-Service policies.

References

1. J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. *The Impact of Technology Scaling on Processor Lifetime Reliability.* In Proceedings of the International Conference on Dependable Systems and Networks (DSN-2004), June 2004.
2. S. Borkar. *Design challenges of technology scaling.* IEEE Micro, pp. 23–29, Jul.–Aug., 1999.
3. R. Bianchini and R. Rajamony. *Power and Energy Management for Server Systems.* IEEE Computer, 37(11), November 2004.
4. C.D. Patel. *A vision of energy aware computing - from chips to data centers.* The International Symposium on Micro-Mechanical Engineering. ISMME2003 -K 1 5. December 2003.

5. J. Moore et al. *Going Beyond CPUs: The Potential of Temperature-Aware Solutions for the Data Center*. Proc. 1st Workshop Temperature-Aware Computer Systems; www.cs.virginia.edu/~skadron/tacs/rang.pdf.
6. J. Moore, J. Chase, P. Ranganathan, and R. Sharma. *Making Scheduling "Cool":Temperature-Aware Workload Placement in Data Centers*. In Proceedings of the USENIX Annual Technical Conference, Anaheim, CA, April 2005.
7. J. Moore et al. *A Sense of Place: Towards a Location-aware Information Plane for Data Centers*. Hewlett Packard Technical Report TR2004-27.
8. Patel, C.D., Sharma, R.K, Bash, C.E. and Graupner, S. *Energy Aware Grid: Global Workload Placement based on Energy Efficiency*. IMECE 2003-41443. 2003 International Mechanical Engineering Congress and Exposition, Washington, DC.
9. A. Weissel and F. Bellosa. *Dynamic thermal management for distributed systems*. In Proceedings of the First Workshop on Temperature-Aware Computer Systems (TACS'04). June 2004.
10. Casanova, H. *Simgrid: A toolkit for the simulation of application scheduling*. Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'01). May, 2001.
11. Parra-Hernandez, R., Vanderster, D., and Dimopoulos, N.J. *Resource Management and Knapsack Formulations on the Grid*. Proceedings of Grid 2004 - 5th IEEE/ACM International Workshop on Grid Computing. pp. 94-101. Nov. 2004.
12. Kursun, E., Cher, C-Y., Buyuktosunoglu, A., and Bose, P. *Investigating the Effects of Task Scheduling on Thermal Behaviori*. Third Workshop on Temperature-Aware Computer Systems, Held in conjunction with ISCA-33, Boston, MA, USA, June 17-21, 2006.