Thermal Load Balancing of Server Workload Given Prediction of Thread Energy Consumption

Adam Lewis, Soumik Ghosh, N.-F. Tzeng
Center for Advanced Computer Studies
University of Louisiana, Lafayette, Louisiana 70504
{awlewis,sxg5317,tzeng}@cacs.louisiana.edu

ABSTRACT

1. INTRODUCTION

Power management techniques developed for mobile and desktop computers are not always appropriate for real-time and high-performance computing applications as these techniques take advantage of slack periods that occur within the application workloads. Real-time applications have strict deadline requirements that cannot be impact by slowing down the processor as required by DVFS. In the case of high-performance computing, typical workloads result in the application server being fully utilized with few resources available for reactive scheduling. As such it becomes necessary to find power management techniques that are pro-active in nature rather than reactive.

In this work, we introduce a technique for thermal managagement that minimizes server energy consumption by adjusting the allocation of thread groups to available cores so as to produce the least contention for memory accesses amongst thread groups. This must be balanced against the performance benefits that are gained through cache affintiy; by allocating thread groups to last utilzied cores as as to gain performance by taking advantage of data still existing in cache will lead to thermal issues through more active cache access.

2. PRIOR WORK

Previous attempts to implement effective thermal management in processors have taken two distinct approaches: hardware based implementations of Dynamic Thermal Management (DTM) combined with Dynamic Voltage-Frequency Sacaling (DVFS) or operating system based multiprocessor or multi-core thread migration approaches.

Commercial processors crudely regulate thermal stress using DTM by slowing down the processor using DVFS whenever processor temperature exceeds a certain threshold. A categorization of the different thermal management techniques based on this approach can be found in [7]. In this work, the authors claimed the best approach to using DTM/DVFS includes a combination of a control-theoretic distributed DVFS system and a sensor-based migration policy. The issue with this approach is the granularity of DVFS; existing commercial systems do not allow for independent frequency scaling of cores and threads.

The idea of migration of work for energy savings and thermal management has a long history in the SMP, SMT, and CMP environments [24]. Static methods been applied in attempts to solve this problem. In [6], integer linear programming was used to obtain a task schedule that met real-time deadlines while attempting to minimize hotspots and spatial temperature differentials across the die.

Examples of dynamic methods to solve this problem incude Heat-and-Run [10],HybDTM [14], and Thresh-Hot [23]. Heat-and-Run proposed to distribute work amongst available cores until the DTM events occur and then migrate threads from the overheated cores to other non-heated coraes. HybDTM combined DTM techniques with a thread migration strategy that reduces the thread priority of jobs on cores that are running hot. Thresh-Hot uses an on-line temperature estimator to determine in what order threads should be scheduled onto cores. In all three cases, these techniques utilize data from hardware performance counters and hardware temperature sensors to guide the scheduling decision.

A related approach to migration is to control the CPU time slice allocation. In [3], it was proposed to modify the process scheduler to allocate time slices as indicated by the contribution of each task to the system power consumption and the current temperature of the processor. A variation on this scheme was proposed in [16] where system level compiler support was used to insert run-time profiling code into applications to provide hints to the thermal intensity of a task. In [18], a scheduling policy was proposed that sorts the tasks in each core's run queue by memory intensity so as to schedule memory-bound tasks at slower frequencies.

2.1 Power and Energy Models

Power models are used to predict to when to apply power management techniques to a working system. These models can be classified into two broad categories: detailed analytical power models and high-level blackbox models [20]. Analytical models use detailed knowledge of the underlying hardware to either simulate the energy consumption or directly measure energy comsumption at the hardware level. Simulation can provide detailed analysis and breakdown of energy consumption; however, they are slow and do not scale well to realistic applications and large data sets. Simulation-based models do not fit well into scensarios where dynamic optimization of application performance is required [8].

High level black-box models sacrifice some accuracy by avoiding extensive detailed knowledge of the underlying hardware. At the processor level, [5], and [3] created power models that linearly correlated power consumption with performance counters. Models have been built for the processor, storage devices, single systems, and groups of systems in data centers. These models have the advantage of being simple, fast and low-overhead but do not model the full-system power consumption.

Measurement-based models attempt to collate power measurements with hardware and software performance metrics. Two distinct classes of metrics have been used in these models: processor performance counters and operating system performance metrics. Processor performance counters are hardware registers that can be configured to count various microarchitectural events such as branch mispredictions and cache misses. In general, the number of countable events exceed the number of available registers. As result, models that use these counters time-multiplex different sets of events on the available registers. While this allows for more events to be monitored, it results in increased overhead and lower accuracy due to sampling issues [8][19].

Attempts have been made to reconcile these approaches by attempting to map programs phases to events [12]. The most common technique used to associate PeCs and/or operating systems metrics to energy consumption use linear regression models to the collected metrics to the energy consumed during the execution of a program [5][8][13][4].

In server envrionments, it has been shown that full-system models using operating system CPU utiliziation can be highly accurate [9]. Others have used similar approaches [11] to develop linear models for energy-aware server consolidation in clusters. Full-system models such as MANTIS [8][19] relate usage information to the power of the entire system rather than an individual component. The accuracy and portability of full system power models is considered in [20]. This analysis indicated that to ensure reasonable accuracy across machines and workload required a model based on a combination of both PeCs and operating system metrics.

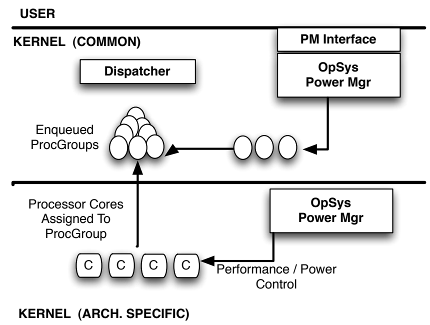

Figure 1: Solaris Power-Aware Dispatcher design.

3. BACKGROUND

It is the role of the kernel dispatcher to manage the placement of threads in a dispatch queue, decide which thread to run on a processor, and manage the movement of threads to and from processors. At the fundamental level, the kernel dispatcher is a queue management system that must perform four core functions [17]: (1) queue management, (2) thread selection, (3) processor selection, and (4) context switching.

3.1 Solaris Power-Aware Dispatcher

OpenSolaris defines locality groups (lgroups) to provide a locality-oriented mechanism to represent a collection of CPUs and memory resources that are within some latency of each other. Lgroups are hierarchical and are created automatically by Solaris based on the systemâĂŹs conïňĄguration and different levels of locality. The system places a new thread into a home lgroup that is based on load averages although applications can give a thread a different home lgroup. Lgroups help to control where threads and memory are allocated. When a thread is scheduled to run, the thread is assigned to the available CPU nearest to the home lgroup, and memory is gotten either from the home lgroup or some parent lgroup.

Inside the Solaris kernel, logical CPUs will belong to one or more CPU partitions. An overall system CPU partition is created at system initialization. At the user level, a logical CPU will be a member of one or more processor groups. Processor groups are implemented inside the kernel as CPU partitions. The Solaris Power-Aware Dispatcher (PAD) (Figure 1) [22] extended the

Solaris kernel dispatcher to enumerate logical CPUs that represent active and idle power domains. The concept of processor groups has been extended to include the concept of logical CPUs sharing one of two power domains: active and idle. At system start-up, the dispatcher will query the operating system's power manager to determine if a logical CPU is a member of either of these domains. If so, the CPU is added to the appropriate processor group. As the workload progresses, the power manager tracks utilization in each power domain. The power manager utilizes this information to decide how to change power states for each CPU. This capability allows the dispatcher to attempt to concentrate light workloads on a smaller number of logical CPUs and make more resources available for scheduling.

3.2 Energy Models and Performance Counters

The foundation of our scheduler is the model for runtime energy consumption detailed in [15]. This model provides a system-wide view of the energy consumption by using hardware performance counters to relate system power consmption to its overall thermal envelope.

Modern processor cores support between 30 to 500 PeCs (depending upon the processor) but only permit 2 to 18 of these counters to be read at one time (again depending upon processor). Furthermore, certain combinations of PeCs may not be permitted to be collected at the same time. Using PeCs to gain insight into architectural events (in our case bus transactions)is complicated by the fact that the types of events, number of events, and use cases varies widely, not only across architectures, but across systems sharing the same Instruction Set Architecture (ISA). For instance, the Intel and AMD implementations of performance counters have very little in common in spite of the processor families using the same ISA [21][1].

As an example, to derive the thermal impact of application on cache, the scheduler needs to collect nine PeCs to calculate shared and unshared cache activity on the AMD Opteron processor: $RetiredInstructions$, $DCAccesses$, $DCRefillsL2$, $DCRefillsFromSystem$, $ICFetches$, $ICrefillsFromL2$, $ICRefillsFromSystem$, $L2RequestsTLB$, and $L2MissesTLB$. The model must calculate the L2 miss rate and L2 miss ratio using the following set of formulas with the PeCs as inputs:

$$L2MissRate = L2misses/RetiredInstructions$$
$$L2MissRatio = L2misses/L2Requests$$

These measures indicate cache activity which our model uses as an estimator for energy consumption and potential for a DTM event.

Note that we have to collect nine different PeCs even though the processor can only collect four of these counters at a time. For real-time situations, time multiplexing is the most popular solution to this problem [2]. In this approach the performance counters are reconfigured for different sets of counter events at regular time intervals. However, time multiplexing introduces issues with reconfiguration overhead and time alignment of samples.

4. THERMAL MODEL

There are two major metrics of interest for the thermal workload of a multicore processor. The first is the total time of execution (denoted by L for length) of a certain application (denoted by A),

$$L(A(p_i), D_A, t),$$

where application A has p processes, each associated with a data set of size d_i in a single chip. D_A is the total data associated with application A, and $D_A = \sum_{i=1}^{p} d_i$. We assume that the activities are taking place in a staging area which contains the main and virtual memory operating spaces, as well as the processor with its cores and their associated caches and shared cache.

This time of execution measurement includes both computation time, and the time to move the data for the problem from the staging area (peripherals off the chip like DRAM and HDD) to a computation or operation area (on the chip such as the caches and the cores).

The second major metric of interest is the energy consumption or energy workload of an application, $U(A(p_i), d_i)$. For each application A and problem size D_A, a measure of the workload, $W(A(p_i), d_i)$, is defined in a data-operation dependent and system-independent way. W contains two components, one being the operations count that is performed by the computational core, and the other is the communication operations in transferring data and instructions and data coherency and book-keeping operations. These are measured in terms of the number of bytes operated ON, or number of bytes transferred. Thus the energy workload of an application A operating on a data set D_A can be expressed as :

$$U(A(p_i), d_i) = \lim_{n \to k_e} n \times W(A(p_i), d_i) \times L_n(A(p_i), d_i, t)$$

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

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

$$\Theta_A(A(p_i), d_i, T, t) = \frac{U(A(p_i), d_i)}{\lim_{T \to T_{th}} J_e \times (T - T_{nominal})} \tag{1}$$

$$= \frac{\lim_{n \to k_e} n \times W(A(p_i), d_i) \times L_n(A(p_i), d_i, t)}{\lim_{T \to T_{th}} J_e \times (T - T_{nominal})} \tag{2}$$

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

For managing the thermal envelope of applications on server systems as well as embedded systems, we are interested in the thermal efficiency of the operation, that is, the thermal cost of taking an application to completion. In general the efficiency $\eta(A(p_i), d_i, T)$ is defined as

$$\eta(A(p_i), d_i, T, t) = \frac{\Theta_A(A(p_i), d_i, T, t)}{\Theta_A(A_e(p_i), d_i, T_{me}, t_e)}$$

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

An important metric finally is the achieved performance per unit power consumed by the chip,

$$C_\theta(A(p_i), d_i, T, t) = \frac{\Theta_A(A(p_i), d_i, T, t)}{P(A(p_i), d_i)} \tag{3}$$

$$= \frac{\Theta_A(A(p_i), d_i, T, t)}{\sum^{chip,DRAM,HT,HDD} \int_{t=0}^{t=L_A} v(t)i(t)dt} \tag{4}$$

where $P(A(p_i), d_i)$ is the overall power consumed during the application lifetime. the quantity $\int_{t=0}^{t=L_A} v(t)i(t)dt$ is the total power consumed by a single physical component (processor, DRAM units, HDD, HT or FSB) during the length of the application L_A, with $v(t)$ and $i(t)$ being the instantaneous voltage and currents respectively.

This normalized quantity C_θ gives some indication of the "cost" of executing the benchmark on the given chip.

The final optimization function could be thought of as :

$$\frac{\partial^2 C_\theta(A(p_i), d_i, T, t)}{\partial T \partial t} = \frac{\partial^2}{\partial T \partial t} \frac{\lim_{n \to k_e} n \times W(A(p_i), d_i) \times L_n(A(p_i), d_i, t}{\lim_{T \to T_{th}} J_e \times (T - T_{nominal})}$$

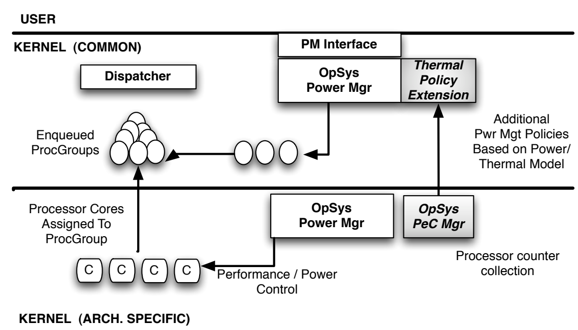

Figure 2: PAD thermal enhancement.

5. DESIGN AND IMPLEMENTATION

The design intent is for our scheduler to be implemented as an extension to the existing dispatcher and power management infrastructure in the operating system. Our initial prototype is an enhancement to the PAD as illustrated in Figure 2.

5.1 Power Management Thermal Enhancements

The concept of processor group is extended to include three new processor domains: hot, warm, and cold. Logical CPUs are classified into each of these domains according to their die temperature as reported by hardware sensors. As the temperature of the processor increases, the processor may be assigned to a different domain based on how close the temperature is to causing a DTM event. The Thermal Manager decides on how to assign a processor into a domain depending upon it's predicted temperature change in the upcoming time period based upon the thermal model.

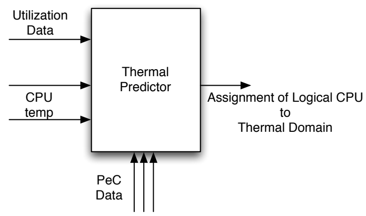

Figure 3: Thermal Predictor Inputs

5.2 Dispatcher Thermal Enhancements

The dispatcher is responsible for deciding where to insert threads into run queues; i.e., when and where a thread will next execute. We adjust this process to move threads away from potentially overheated processor groups.

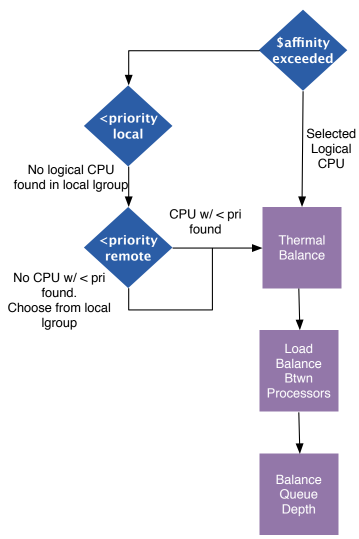

Figure 4: Thread Queue Insertion

6. CONCLUSIONS AND FUTURE WORK

7. REFERENCES

[1] AMD. *Software Optimization Guide for AMD Family 10h Processors*, 3.06 08 edition, September 2005.

[2] R. Azimi, M. Stumm, and R. W. Wisniewski. Online performance analysis by statistical sampling of microprocessor performance counters. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 101–110, New York, NY, USA, 2005. ACM.

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

[4] W. Bircher and L. John. Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events. *Ispass*, 0:158–168, 2007.

[5] G. Contreras and M. Martonosi. Power Prediction for Intel XScale®Processors Using Performance Monitoring Unit Events. In *ISLPED '05: Proc. of the 2005 Intl. Symp. on Low Power Electronics and Design*, pages 221–226, New York, NY, USA, 2005. ACM.

[6] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross. Temperature-aware mpsoc scheduling for reducing hot spots and gradients. In *ASP-DAC '08: Proceedings of the 2008 conference on Asia and South Pacific design automation*, pages 49–54, Los Alamitos, CA, USA, 2008. IEEE Computer Society Press.

[7] J. Donald and M. Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration. In *ISCA '06: Proc. of the 33rd Intl. Symp. on Computer Architecture*, pages 78–88, Washington, DC, USA, 2006. IEEE Computer Society.

[8] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-System Power Analysis and Modeling for Server Environments. In *Workshop on Modeling Benchmarking and Simulation (MOBS) at ISCA*, 2006.

[9] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proc. of the 34th Intl. Symp. on Computer architecture*, pages 13–23, New York, NY, USA, 2007. ACM.

[10] M. Gomaa, M. D. Powell, and T. N. Vijaykumar. Heat-and-run: leveraging smt and cmp to manage power density through the operating system. *SIGOPS Oper. Syst. Rev.*, 38(5):260–270, 2004.

[11] T. Heath, B. Diniz, E. V. Carrera, W. M. Jr., and R. Bianchini. Energy Conservation in Heterogeneous Server Clusters. In *PPoPP '05: Proc. of the 10th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pages 186–195, New York, NY, USA, 2005. ACM.

[12] C. Isci and M. Martonosi. Phase Characterization

for Power: Evaluating Control-flow-based and Event-counter-based Techniques. *The 12th Intl. Symp. on High-Performance Computer Architecture*, pages 121–132, 11-15 Feb. 2006.

[13] C. Isci and M. Martonosi. Runtime Power Monitoring in High-end Processors: Methodology and Empirical Data. *MICRO-36: Proc. 36th IEEE/ACM Intl. Symp. on Microarchitecture*, pages 93–104, 3-5 Dec. 2003.

[14] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha. Hybdtm: a coordinated hardware-software approach for dynamic thermal management. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 548–553, New York, NY, USA, 2006. ACM.

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

[16] D. Li, H.-C. Chang, H. Pyla, and K. Cameron. System-level, thermal-aware, fully-loaded process scheduling. pages 1–7, April 2008.

[17] R. McDougall and J. Mauro. *Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture*. Prentice Hall, 2nd edition, 2007.

[18] A. Merkel and F. Bellosa. Memory-aware Scheduling for Energy Efficiency on Multicore Processors. In *Proc. of the USENIX Workshop on Power-aware Computing and Systems (HotPower'08)*.

[19] S. Rivoire. *Models and Metrics for Energy-efficient Computer Systems*. PhD thesis, Stanford University, 2008.

[20] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A Comparison of High Level Full-System Power Models. In *Proc. of USENIX Workshop on Power Aware Computing and Systems (HotPower'08)*, 2008.

[21] R. Singhal. Inside Intel Next Generation Nehalem Microarchitecture. In *Intel Developer Forum, Shanghai, China*, 2008.

[22] Sun Microsystems. OpenSolaris CPU Power Management - Project Tesla, June 2009.

[23] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin. Dynamic thermal management through task scheduling. *Performance Analysis of Systems and software, 2008. ISPASS 2008. IEEE International Symposium on*, pages 191–201, April 2008.

[24] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *FOCS '95: Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, page 374, Washington, DC, USA, 1995. IEEE Computer Society.