Software Optimization for Performance, Energy, and Thermal Distribution: Initial Case Studies

Md. Ashfaquzzaman Khan Can Hankendi Ayse Kivilcim Coskun Martin C. Herbordt
Electrical and Computer Engineering Department, Boston University
{azkhan|hankendi|acoskun|herbordt}@bu.edu

***Abstract*—As an initial step in our *Green Software* research, this paper investigates whether software optimization at the application level can help achieve higher energy efficiency and better thermal behavior. We use both direct measurements and modeling to quantify power, energy and temperature for a given software method. The infrastructure includes a new power estimator for multicore systems developed by regressing measurements from a custom-designed suite of microbenchmarks. Using our evaluation methodology on a real-life multicore system, we explore two case studies. In the first one, we use software tuning for improving the scalability and energy-efficiency of a parallel application. The second case study explores the effect of temperature optimization on system-level energy consumption.**

I. Introduction

Energy efficiency is one of the central societal and technical issues of our time. Data center energy consumption is now 2-3% of total US electricity use and is increasing by 15% per year [12]. Recent studies show that *total cost of a server will be primarily a function of the power it consumes* [10]. A significant body of research exists in system-level energy management for using low power modes when there is slack. The common objective in such techniques is to maintain a desired level of performance while reducing energy consumption. A closely related issue is thermal management: High power consumption not only increases operational energy costs, but also causes high temperatures and thus dramatically raises cooling costs. There has been a move to significantly lower the energy costs of *cooling* by using less expensive infrastructures to replace the HVAC systems used today. Replacing the HVAC systems, however, results in much less predictability and significantly higher temperatures, degrading system reliability and performance.

Considering the complexity of large scale computing infrastructures as well as time-to-market restrictions, it is not cost-efficient to address these three pressing challenges—**P**erformance, **E**nergy, and **T**emperature (*PET*)—solely through novel hardware design. We know that workload has a significant effect on all three parameters. But while software optimization for performance is an established field and there has been significant developments in system-level energy/thermal management, optimizing software for energy and thermal efficiency simultaneously is still in its infancy. Obviously, better software gets better performance; less obvious is that there is also better software for reducing energy consumption and thermal problems. For example, the distance that data must be routed before they can be operated upon relates directly to the energy required for that operation. There are similar effects for the *variance* in component switching. Reducing this energy per computation often requires non-obvious software restructuring or introducing non-intuitive "extra" computations.

978-1-4577-1221-0/11/$26.00 ©2011 IEEE

A major challenge in jointly optimizing for P, E, and T is the complex interplay among these metrics. For example, while higher performance often implies higher power consumption, selecting an operating point with a higher power value does not always produce better performance. Also, a method optimizing the energy efficiency by clustering all the workload in a few resources temporally or spatially may harm the thermal profile by creating hot spots on the active resources [6]. Our hypothesis in this research is that optimizing for PET is different from optimizing for PE or PT only; and that considering all three parameters, PET, is necessary to improve energy efficiency while maintaining high performance and reliability. As a part of this larger hypothesis, in this paper we investigate whether software optimization at the application level can help achieve higher energy efficiency and better thermal behavior. We first present the measurement and modeling framework we have constructed to evaluate PET tradeoffs accurately. We then use this framework for two case studies:

- Software tuning for `dedup`, one of the applications in the PARSEC benchmark suite of parallel applications [3]. We show that by adopting the appropriate parallelization model, along with parameter tuning in the application, we can improve scalability of the application, and also reduce its energy consumption, temperature, and thermal variation.
- Effect of temperature optimization on system energy consumption. Specifically, we evaluate the PET tradeoffs of various combinations of a set of custom-designed microbenchmarks and the `mPrime` stress test [1]. We show that temperature reductions limited to a few degrees can result in considerable energy savings at system level due to the reduction in cooling power.

The rest of the paper is organized as follows. Section II discusses the background and related work. Section III explains our evaluation methodology. We present our case studies in Section IV and Section V concludes the paper.

II. Background

We have seen significant developments in energy management research at chip level in the last decade. While most

conventional runtime management techniques focus on performance as the primary objective, more recently researchers have developed energy-aware workload management techniques—especially in the embedded systems domain. A vast amount of prior research exists in dynamic energy and temperature management. Most well-known methods for controlling energy consumption are dynamic power management (DPM), where idle units are put into a low-power sleep state [2], and dynamic voltage and frequency scaling (DVFS).

Multicore systems are typically under-utilized most of their life-times, providing opportunities for low cost energy management through workload scheduling. Solutions to the workload scheduling problem in prior work include static optimization methods for *a priori* known workloads, such as integer linear program (ILP) based solutions for finding the optimal voltage schedule and task ordering [16] or minimizing the data transfer on the bus while guaranteeing average case performance [17].

Power-aware policies are generally not sufficient to prevent temperature induced problems, thus efficient thermal modeling and management methods have been proposed in the last decade following the gating limitations with temperature. High temperatures and large thermal variations cause severe challenges in reliability, leakage power, performance, and cooling cost. Static (design-time) methods for thermal management at chip-level are based on thermal characterization at design time using automated thermal models such as HotSpot [19]. Examples of static methods are temperature-aware floorplanning [18] and adjusting architectural configuration to improve temperature-dependent reliability [20].

For many systems (especially in high-end computing domain) workload is not known *a priori*; therefore, dynamic and fast decisions are crucial for achieving desired tradeoff points between performance and temperature. Conventional dynamic thermal management controls over-heating by keeping the temperature below a critical threshold on the chip [19]. There are also techniques that combine DVFS or clock-gating with thread scheduling to reduce the performance cost of thermal management [8], [13]. Temperature-aware workload scheduling is able to achieve high performance while minimizing the harmful thermal hot spots and gradients [6].

On the data center and HPC research fronts, energy management has been receiving more attention due to the pressing limitations resulting from high energy consumption. Power provisioning techniques analyze the trends in current data centers and HPC clusters to correlate energy consumption and performance [9]. Various recent papers focus on developing new, energy-efficient, high-performance architectures for large scale computing trends [14]. For virtualized environments, energy-aware scheduling of virtual nodes is shown to improve energy savings and performance at the same time [7].

The goal of this work is to extend methods developed for performance optimization to also account for energy and temperature, while most prior work targets only one or two of these parameters. We specifically target software design and optimization instead of system-level techniques such as OS scheduler enhancements. Our aims in the longer run are: (1) to develop widely applicable guidelines for software design, and (2) to provide an application-specific analysis enabling the design of better metrics and tools to evaluate P, E, and T of software. As an intial step in these directions, we provide an evaluation methodology and two case studies investigating the effect of software restructuring on P, E, and T.

III. Methodology

A fundamental component in this research project is to quantify the values of P, E, and T for a given software execution method. Our target system in this work is a high-end 1U server with a 12-core AMD Magny-Cours chip. We follow two directions for the quantification of P, E, T: direct measurement and modeling. While direct measurement on the hardware is useful for real-life verification of the techniques and concepts, modeling is essential for higher observability into the system parameters.

A. PET Measurements

Most modern CPUs provide performance counters that can be used to measure events such as number of retired instructions, cache misses, etc. We use these performance counters in the AMD Magny-Cours chip to measure detailed performance characteristics and also to predict per-core power consumption (see Section III-B). We use the `pfmon` [11] utility to collect performance counter data with a sampling rate of 40ms and multiplexing interval of 20ms (i.e., multiplexing for measuring a larger set of performance parameters than the number of

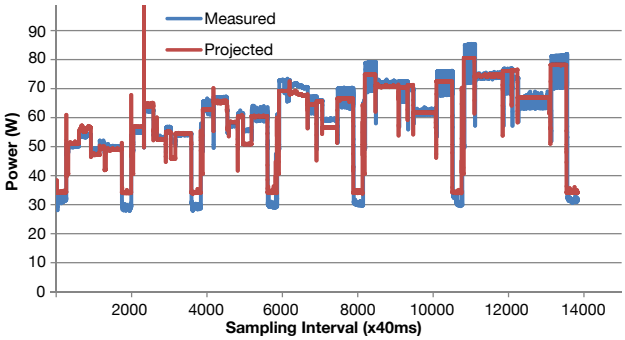

Fig. 1: Comparison of power estimation with measurement for microbenchmarks running with 1-6 threads.

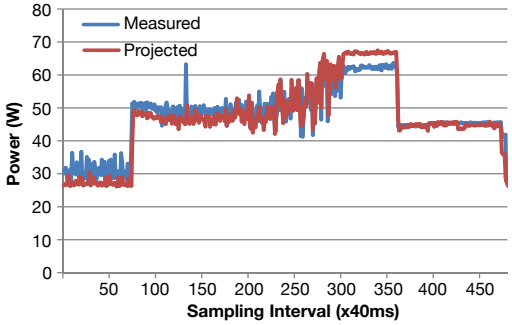

Fig. 2: Validation of the power model for the default dedup running on 6 cores.

available hardware counters). We also measure both system and chip level power consumption of the server. For chip power consumption, we identify the 12V input wires to the voltage regulator and use a Hall-ect clamp ammeter (Agilent 34134A) to measure the current flow. We then log the current measurements via an Agilent 34410 multimeter with a sampling rate of 40ms. For system level power consumption, we use a `Watts up? PROes` power meter with a sampling rate of 1s. For temperature measurements, we poll the temperature sensor on the Northbridge bus using the `lm-sensors` tool.

B. Power Estimation

The power and thermal measurements collected from the chip are not sufficiently fine-grained to determine per-core power and temperature; per-core data are necessary to create a thermal profile appropriate for making PET management decisions. For building a model to predict per-core power consumption, we have custom-designed a set of microbenchmarks. Microbenchmarks are useful in model building as they enable the stressing of different components of the CPU separately. Performance counter data and actual power consumption data are collected simultaneously while running these microbenchmarks. We then use linear regression to derive a model for predicting power from the performance data. The accuracy of the model depends on the nature of the microbenchmarks and the selection of hardware counters. Our multithreaded microbenchmarks are as follows.

In-cache matrix multiplication (double): This is a matrix multiplication program running on an array of `doubles`. Program data fit in L1 cache. The program uses vector instructions with aggressively unrolled loops and achieves slightly over 80% of the maximum achievable FLOPs. The performance scales almost linearly with the number of cores. Its purpose is to stress the floating point units of the CPU.

In-cache matrix multiplication (short): Similar to the previous benchmark, but uses the `short` data type. It achieves about 70% of the maximum CPU utilization. This program scales close to linearly. Its purpose is to stress the integer units.

Intensive memory access without sharing: This program continuously accesses data from the main memory. It consists of a `for` loop where in each iteration an integer in the next cache block is read, incremented, and written back. The data size is chosen to exceed the size of the largest cache so that data are actually read from main memory every time. Each thread accesses its own portion of the data with no sharing among the threads. Its purpose is to stress the memory units.

Intensive memory access with sharing: This program accesses data from main memory in a similar way as the previous one, but this time data are shared among threads. This program scales quite well, although not linearly. Its purpose is to represent shared cache access among the CPU cores.

Intensive memory access with frequent synchronization: This is another memory intensive program. Frequent spin-lock based synchronization is used before accessing shared data. This program scales poorly. Its purpose is to represent intense data sharing among the CPU cores.

In-cache matrix multiplication (short-simple): This version uses the simplest matrix multiplication program on the `short` data type. It uses three levels of loops with no loop unrolling or vector instructions. This program scales near linearly. Its purpose is to stress the branch prediction unit.

TABLE I: Hardware events and coefficients used in the regression model for predicting power.

Hardware Event	Coefficient
Constant Coefficient	28.1184
CPU_CLK_UNHALTED	6.0284
RETIRED_UOPS	2.989
RETIRED_MMX_AND_FP_INST:ALL	-1.9054
RETIRED_SSE_OP:ALL	0.094
L2_CACHE_MISS:ALL	695.9063
DISPATCH_STALLS	1.3658
DISPATCHED_FPU:ALL	4.6154
RETIRED_BRANCH_INST	31.9637

We use the hardware events listed in Table I as the input metrics for the regression-based power model. Also in the table are the coefficients computed using regression. All counter values are normalized with CPU_CLK_UNHALTED, which itself is normalized with 84,000,000 (the number of cycles in 40ms at 2.1 GHz frequency). The power model includes standby idle power. Power for individual cores is derived by using individual performance counters for each core and dividing the power accordingly. We do not explicitly model the temperature-dependence of leakage power as the temperature ranges we observe are mostly in the range of $50-60^oC$. Figure 1 compares the measured power and estimated power for the microbenchmarks.

We have validated our model using the PARSEC suite [3] running native input set. Figure 2 compares the measured power and estimated power for `dedup` (default version, no modifications to the benchmark). Figure 3 demonstrates the average estimation error for the entire PARSEC suite.

C. Thermal Model

For thermal simulations, we use HotSpot-5.0 [19]. HotSpot requires chip layout and power traces for the units on the die to calculate the transient temperature response. We derive the dimensions of each core and L2-caches from the layout

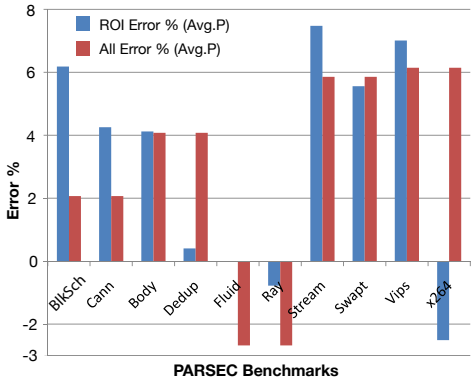

Fig. 3: Power estimation model error for PARSEC.

TABLE II: HotSpot Parameters

Parameter	Value
Die Thickness	0.1 mm
Core Area	13.44 mm^2
L2 Area	8.06 mm^2
Heat sink side	80 mm
Heat spreader side	40 mm
Convection Capacitance	140 J/K
Convection Resistance	0.1 K/W

provided by AMD [5]. AMD Magny-cours consists of two AMD Istanbul 6-core chips that reside side by side in the same package. For constructing the thermal model, we focus on one of the 6-core chips since temperature behavior of each chip is mostly independent from the other. Figure 4 shows the layout we use in the thermal simulations. Using our power estimation model, we generate the per-core power traces. We compute L2-cache power using CACTI [21] and use a fixed L2-cache power value of 1.5W. As caches have low temperature, using a fixed power value has acceptable inaccuracy. All thermal simulations are initialized with steady-state values, which is necessary when the simulation time is not long enough to warm up the package and the chip. Table II summarizes the HotSpot configuration.

IV. Case Studies

A. Parallelization of dedup

The purpose of the first case study is to illustrate the effect of code restructuring on PET. We use `dedup`, a kernel that executes a data compression method called "deduplication" [4]. It combines local and global compression to achieve very high compression ratios. Deduplication has become a mainstream method to compress storage footprints for new generation backup storage systems and to compress communication data for bandwidth-optimized networking appliances [4].

Pipelined Parallelization Model: The original `dedup` kernel in PARSEC [3], which we call the *default* version, uses a pipelined programming model to parallelize the compression. There are five stages, the middle three of which are parallel. The first stage reads the input stream and breaks it up into coarse-grained chunks to get independent work units for the

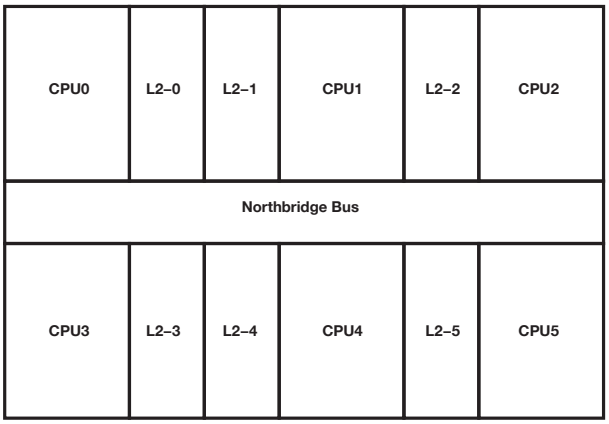

Fig. 4: HotSpot layout for AMD Istanbul 6-core chip

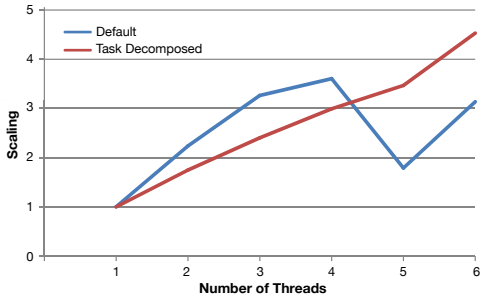

Fig. 5: Scaling of default version and the parameter-tuned task-decomposed version of `dedup`.

threads. The second stage anchors each chunk into fine-grained segments with rolling fingerprinting [15]. The third stage computes a hash value for each data segment. The fourth stage compresses each data segment with the Ziv-Lempel algorithm and builds a global hash table mapping hash values to data. The final stage assembles the deduplicated output stream consisting of hash values and compressed data segments.

The default version of `dedup` uses a separate thread pool for each parallel pipeline stage. Each thread pool has a number of threads equal to the number of available cores to allow the system to fully work on any stage, should the need arise. The OS scheduler is responsible for thread scheduling. In order to avoid lock contention, the number of queues is scaled with the number of threads, with a small group of threads sharing an input and output queue at a time.

Task-decomposed Parallelization Model: The pipelined parallelization results in threads that have widely varying workloads. This ultimately causes large differences in power consumption in both time and space. Due to a lack of data reuse and increased need for synchronization, performance and scaling also suffer. In this paper, we propose to parallelize `dedup` using a task-based scheme: the three parallel stages of the original method are merged into a single stage. Each thread now takes one task from the first stage and performs all computation of that task. Upon completion, the result of the task is supplied to the final stage. As before, the initial and final stages are performed in serial.

This method reduces synchronization overhead since three parallel stages are now merged into a single stage. It also increases data reuse. And since all threads now perform similar tasks, the power consumption and temperature profiles are more uniform. Figure 5 shows the scaling result of the default version and the final parameter-tuned task-based version. The proposed version achieves stable scalability and outperforms the default version as number of cores increases. We have observed this trend up to 12-cores with performance of our version nearly $1.8\times$ that of the default.

Parameter Tuning: The amount of work each thread does between two synchronizations has a direct impact on performance. If this amount is too small, threads will spend more time in synchronization. If this amount is too large, threads will spend more time waiting for the queue to have a sufficient number of tasks. Therefore, this amount must be tuned for each application-platform pair. The default version

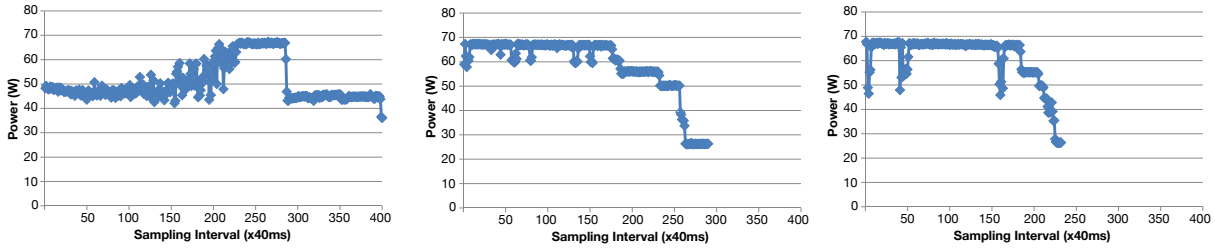

Fig. 6: Total chip power consumption of Dedup ROI (From left: default version, task-decomposed version before parameter tuning, task-decomposed version after parameter tuning).

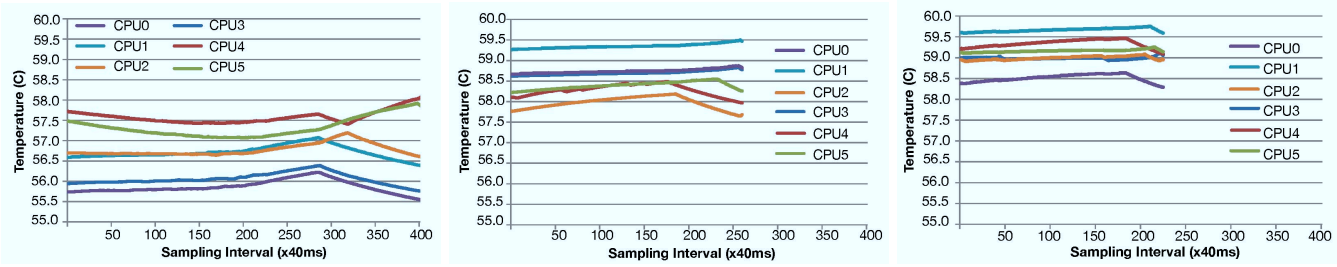

Fig. 7: Per-core temperature for Dedup ROI (From left: default version, task-decomposed version before parameter tuning, task-decomposed version after parameter tuning).

of PARSEC `dedup` takes 20 tasks from the queues every time. This is a parameterized value that can be changed in the program source code. For our task-decomposed version, we empirically determine that a value of 10 balances the amount of wait time and synchronization time.

Figure 6 shows the total chip power consumption for the three versions of `dedup`: the default version, the initial task-based version, and the parameter-tuned task-based version. The power consumption varies largely in the default version. This is due to the imbalance in load distribution of the pipelined model. The initial task-based version overcomes this because threads have more similar workloads. The parameter-tuned version improves the balance further, achieving a nearly uniform power trace throughout the ROI. The drop in power at the end occurs when the tasks finish. Figure 7 shows the thermal simulation results. The improved versions have more stable temperatures in both time and space.

Overall, our parameter-tuned task-based model improves energy efficiency with a 30% reduction in CPU energy and a 35% reduction in system energy compared to the default version. Per-core maximum temporal thermal variance and spatial thermal variance among cores are reduced by 56% and 41%, respectively.

B. Effect of Temperature Optimization

To investigate the effects of temperature optimization, we first run experiments to derive the temperature time constant. In Figure 8, we show that optimizing temperature at μs granularity has substantial benefits for reducing temperature. This granularity can easily be targetted at software level. We then investigate `mPrime` stress test as well as two of the microbenchmarks introduced in Section III: *in-cache matrix multiplication (double),* **MM**, and *Intensive memory access with frequent synchronization,* **Spinlock**.

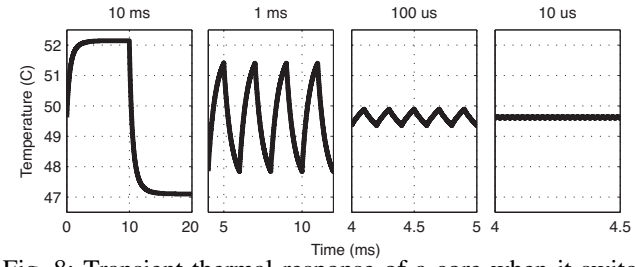

Fig. 8: Transient thermal response of a core when it switches between active and idle power states at a frequency of $10ms$, $1ms$, $100\mu s$, and $1\mu s$.

`mPrime` is a software application that is part of a distributed project GIMPS which aims to find new Mersenne prime numbers. `mPrime` includes a 'torture test' to stress various components of the processor and it is widely used for testing the reliability of the over-clocked systems. We run the torture test to explore the effects of temperature optimization on system and chip power consumption. Figure 9 shows both chip and system level power consumption as well as temperature sensor readings from the Northbridge bus. Our experiments show that 25^oC ΔT translates into $10W$ ΔChip-power and $17W$ ΔSystem-power. This implies that optimizing temperature would potentially result in lower chip power through reducing leakage and lower system energy through reducing the fan power.

In addition to `mPrime` stress test, we measure power and temperature for two of the microbenchmarks, MM and Spinlock. These two microbenchmarks have different characteristics in terms of power, temperature and energy. MM has the highest power consumption across all microbenchmarks, whereas Spinlock has the lowest power consumption. Figure 10 shows chip and system power measurements and peak core temperature values from HotSpot simulations. At chip

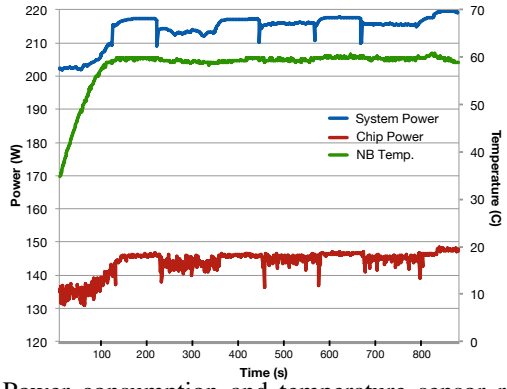

Fig. 9: Power consumption and temperature sensor readings from Northbridge bus for `mPrime` stress test

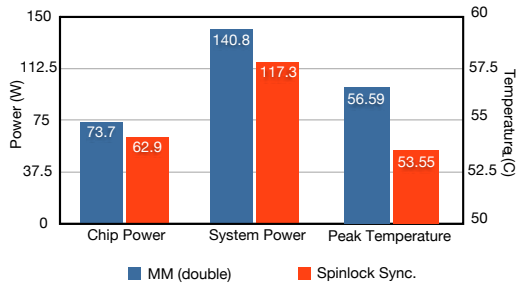

Fig. 10: Chip power, system power and peak temperature for matrix multiplication (MM) and spinlock synchronization microbenchmarks.

level, power consumption difference of these two microbenchmarks are $10.8W$ and the system-level power consumption difference is $23.5W$. After subtracting the chip-level power difference, there is still $12.7W$ system-level power difference. MM performs all of its task in cache. This property of MM minimizes the power consumed by memory, so memory power can only be a minor factor on the change in system level power consumption. Note that the system uses a dynamic fan control algorithm by default. Higher power consumption of MM increases the fan speed, potentially explaining the major portion of the system level power difference of $12.7W$. MM also has 3^oC observable difference in peak core temperature in comparison to Spinlock.

V. Conclusions

In this paper we have presented initial results in application-level software optimization for performance, energy, and thermal distribution. We believe that this approach fills an important gap, and has the potential for providing a hardware-independent reduction in computing cost.

We have set up an evaluation infrastructure and tested it in two case studies. The infrastructure includes a new power estimator, which was developed using a new suite of microbenchmarks. We showed the utility of the predictor with respect to the benchmarks from the PARSEC suite with an average error of less than 5%.

The first case study demonstrates the effect of code restructuring on performance, energy, and thermal distribution. The variation in effects illustrates the potential of the overall approach. In particular, the stability of the thermal behavior is likely to improve cooling efficiency. The second case study investigates the effects of temperature optimization. By analyzing the thermal time constant, we first show that optimizations at application level can potentially improve the thermal behavior. We also show potential savings both at system and chip level by studying temperature and power behavior of different workloads.

Our future work includes various other software optimization techniques to improve power, energy and temperature behavior of emerging parallel workloads that are frequently used in both HPC clusters and data centers.

Acknowledgement

This research has been partially funded through the Dean's Catalyst Award, College of Engineering at Boston University.

References

[1] http://www.mersenne.org/freesoft/.
[2] L. Benini, A. Bogliolo, and G. D. Micheli. A Survey of Design Techniques for System-Level Dynamic Power Management. *IEEE Trans. Very Large Scale Integr. Syst.*, 8(3):299–316, 2000.
[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *PACT*, 2008.
[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. Technical Report TR-811-08, Princeton University, January 2008.
[5] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes. Blade computing with the AMD opteron processor magny-cours. Hot Chips, 2009.
[6] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross. Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs. *IEEE Transactions on VLSI*, 16(9):1127–1140, Sept. 2008.
[7] G. Dhiman, G. Marchetti, and T. Rosing. vgreen: a system for energy efficient computing in virtualized environments. In *ISLPED*, 2009.
[8] J. Donald and M. Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration. In *ISCA*, pages 78–88, 2006.
[9] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA*, pages 13–23, 2007.
[10] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.
[11] S. Jarp, R. Jurga, and A. Nowak. Perfmon2: a leap forward in performance monitoring. *Journal of Physics: Conference Series*, 2008.
[12] J. G. Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3(3):034008, 2008.
[13] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha. HybDTM: A Coordinated Hardware-Software Approach for Dynamic Thermal Management. In *DAC*, pages 548–553, 2006.
[14] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *ISCA*, pages 315–326, 2008.
[15] U. Manber. Finding similar files in a large file system. In *Usenix Winter Technical Conference*, pages 1–10, 1994.
[16] P. Rong and M. Pedram. Power-Aware Scheduling and Dynamic Voltage Setting for Tasks Running on a Hard Real-Time System. In *ASPDAC*, pages 473–478, 2006.
[17] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano. Communication-Aware Allocation and Scheduling Framework for Stream-Oriented Multi-Processor System-on-Chip. In *DATE*, 2006.
[18] K. Sankaranarayanan, S. Velusamy, M. Stan, and K. Skadron. A Case for Thermal-Aware Floorplanning at the Microarchitectural Level. *The Journal of Instruction-Level Parallelism*, 7, 2005.
[19] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture. In *ISCA*, 2003.
[20] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The Case for Lifetime Reliability-Aware Microprocessors. In *ISCA*, 2004.
[21] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. CACTI 4.0. Technical Report HPL-2006-86, HP Laboratories Palo Alto, 2006.