Application Level Optimizations for Energy Efficiency and Thermal Stability

Md. Ashfaquzzaman Khan Can Hankendi Ayse Kivilcim Coskun Martin C. Herbordt

Department of Electrical and Computer Engineering
Boston University; Boston, MA 02215

Abstract: As part of our *Green Software* project we investigate whether application-level software optimization can improve energy efficiency and thermal behavior. We use both direct measurements and modeling to quantify power, energy, and temperature for a given software method. We introduce a new power estimator for multicore systems developed by regressing measurements from a suite of custom-designed microbenchmarks. We use our evaluation methodology on a real-life multicore system to explore two issues: (i) software tuning to improve scalability and energy-efficiency, and (ii) the effect of temperature optimization on system-level energy consumption.

1. INTRODUCTION

Recent studies show that the *total cost of a server will be primarily a function of the power it consumes* [2]. 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. Considering the complexity of large scale computing infrastructure, it is not cost-efficient to address the 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 metrics, but also that their complex interplay makes their joint optimization a major challenge. For examples, reducing the energy per computation often requires non-obvious software restructuring or the introduction of non-intuitive "extra" computations. Also, a method optimizing energy efficiency by clustering the workload in a few resources, temporally or spatially, can harm the thermal profile by creating hot spots on the active resources.

Our hypothesis is that optimizing for PET is different from optimizing for PE or PT only; and that considering all three parameters is necessary to improve energy efficiency while maintaining high performance and reliability. As a part of this larger hypothesis, we investigate whether software optimization at the application level can help achieve higher energy efficiency and better thermal behavior.

2. METHODS

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

2.1 PET Measurements

Most modern CPUs provide performance counters that can be used to measure events such as number of retired instructions, cache misses, and many others. We use these to measure detailed performance characteristics and to predict per-core power consumption. We also collect measurements for both system and chip level power consumption. For chip level, we identify the 12V input wires to the voltage regulator and use a Hall-ect clamp ammeter (Agilent 34134A) to measure the current flow. For system level power consumption, we use a `Watts up? PROes` meter. For temperature measurements, we poll the temperature sensor on the Northbridge bus using the lm sensors hardware monitoring tool.

2.2 Power Estimation

The power and thermal measurements collected from the chip are not sufficiently fine-grained to determine per-core power and temperature, which are necessary to create a thermal profile appropriate for dynamically managing PET. To build a model to predict per-core power consumption, we design a set of microbenchmarks. We collect performance and power data, and then use linear regression to derive the model. Our microbenchmarks are as follows.

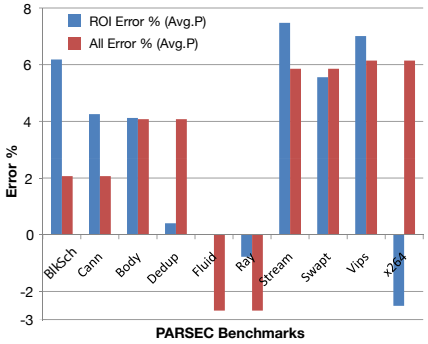

Figure 1: Power estimation model error.

- In-cache matrix multiplication (double) – MM
- In-cache matrix multiplication (short)
- Memory access without sharing
- Memory access with sharing
- Memory access with frequent synchronization – Spinlock
- In-cache matrix multiplication (short-simple)

Respectively, these exercise floating point units, integer units, memory, shared caches, data sharing, and branch predictors. We validate our model using the PARSEC benchmark suite [1] running the native input set. Figure 1 demonstrates the average estimation error for the entire PARSEC suite.

2.3 Thermal Model

For thermal simulations, we use HotSpot-5.0 [3]. HotSpot requires chip layout and power traces for each unit to calculate the transient temperature response. We derive the dimensions of cores and L2-caches from the layout provided by AMD. 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. Using our power estimation model, we generate the per-core power traces. Average power consumption per core across the PARSEC benchmarks is 8.5W. For computing L2-cache power traces, we utilize CACTI [4] and use a fixed L2-cache power value of 1.5W. As caches have low temperature, using a fixed power value has acceptable accuracy. All thermal simulations are initialized with steady-state values to warm up the package and the chip.

3. CASE STUDIES

3.1 Code Optimization for Energy Efficiency

The purpose of the first case study is to illustrate the effect of code restructuring on PET. For this we use `dedup`, a kernel that executes a next-generation data compression method called "deduplication" [1]. The original kernel uses a pipelined programming model to parallelize the compression. There are five stages, the middle three of which are executed in parallel. Each stage uses a separate thread pool; 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. The OS 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.

The pipelined parallelization results in threads having widely varying workloads. This ultimately causes varying power consumption in both time and space. Due to a lack of data reuse and increased need

for synchronization, performance and scaling also suffer. To optimize `dedup`, we parallelize using a task-based scheme: the three parallel stages are merged into a single thread. 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.

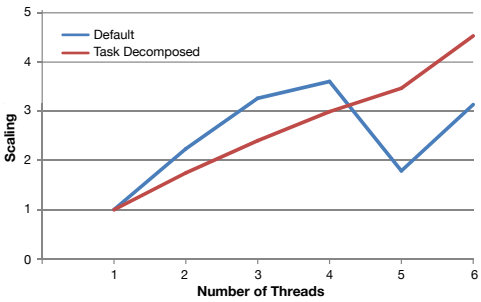

Figure 2: Performance scaling: original and optimized.

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 should be more uniform. Figure 2 shows the scaling result of the original and optimized versions: The latter achieves stable scalability and outperforms the default version as number of cores increases. We observe this trend up to 12-cores with performance improvement of nearly $1.8\times$.

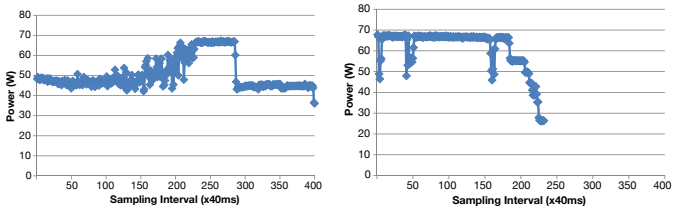

Figure 3: Total chip power consumption: original and optimized.

Figure 3 shows the total chip power consumption for the original and optimized versions. Note that the power consumption varies widely in the default version due to the load imbalance, while most of this is removed in the optimized version. The drop in power at the end is due to the end of available tasks. Figure 4 shows the results from the thermal simulation. The optimized version has more stable temperature in both time and space. Overall, the optimized model improves energy efficiency with a 30% reduction in CPU energy and a 35% reduction in system energy compared with the original version. Per core maximum temporal thermal variance is reduced by 56% and overall thermal variance among the cores is reduced by 41%.

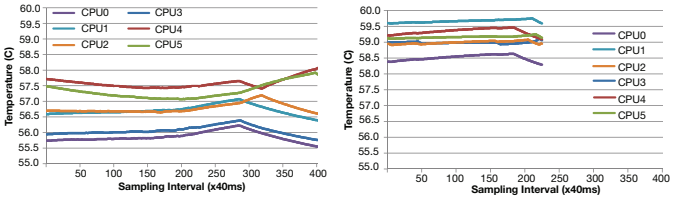

Figure 4: Per-core temperature: original and optimized.

3.2 Effect of Temperature Optimization

To investigate the effects of temperature optimization, we first derive the temperature time constant. In Figure 5, we show that optimizing temperature at a $100\mu s$ granularity has the potential for substantial benefit in reducing temperature. This granularity can easily be targetted at the application software level.

We next experiment with the `mPrime` stress test and two of the microbenchmarks (from Section 2), MM and Spinlock. Figure 6 shows both chip and system level power consumption as well as temperature sensor readings from the Northbridge bus. Our experiments show that a 25^oC ΔT translates into $10W$ ΔChip-power and $17W$ ΔSystem-power. This implies that optimizing temperature potentially reduces chip power due to lower leakage power, and reduces system-power by decreasing the cooling cost (fan power).

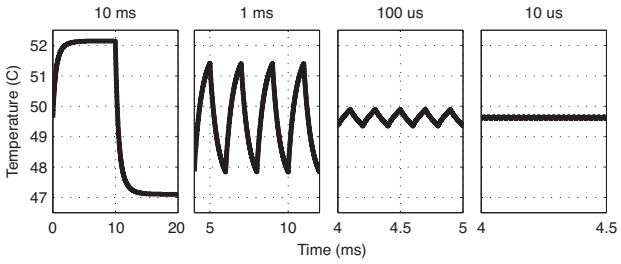

Figure 5: Transient thermal response of a core when it switches between active and idle power states at various frequencies.

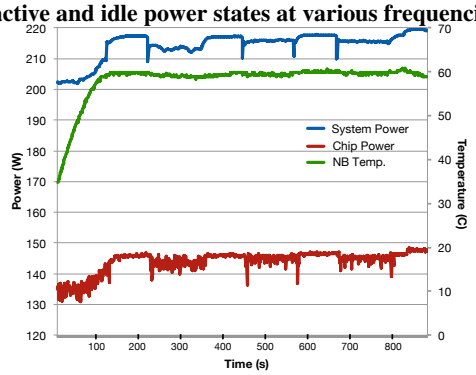

Figure 6: Power consumption and temperature sensor readings from Northbridge bus for `mPrime` stress test

Figure 7 shows actual chip and system power measurements and peak core temperatures from HotSpot thermal simulations for MM and spinlock. MM has the highest power consumption across all microbenchmarks, while spinlock has the lowest. At the chip level, the difference is $10.8W$, at the system-level $23.5W$. After subtracting the chip-level power difference, there is still a $12.7W$ system-level power difference. Note that the system uses dynamic fan control by default. The higher power consumption of MM increases the fan speed, explaining the major portion of the system level power difference. MM also has a 3^oC higher peak core temperature. These studies underline the importance of temperature optimization and show potential for system level energy savings.

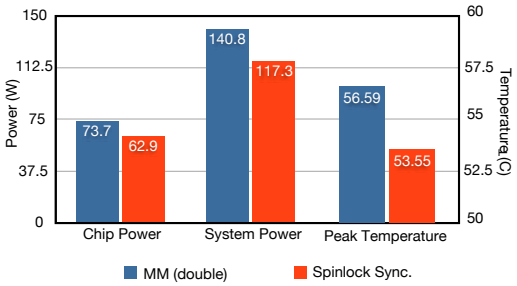

Figure 7: Comparing chip power, system power, and temperature.

4. CONCLUSIONS

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

5. REFERENCES

[1] Bienia, C., Kumar, S., Singh, J. P., and Li, K. The parsec benchmark suite: Characterization and architectural implications. In *PACT* (2008).

[2] Hoelzle, U., and Barroso, L. A. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 1st ed. Morgan and Claypool Publishers, 2009.

[3] Skadron, K., Stan, M., Huang, W., Velusamy, S., Sankaranarayanan, K., and Tarjan, D. Temperature-Aware Microarchitecture. In *ISCA* (2003).

[4] Tarjan, D., Thoziyoor, S., and Jouppi, N. P. CACTI 4.0. Tech. Rep. HPL-2006-86, HP Laboratories Palo Alto, 2006.