The Need for Power Debugging in the Multi-Core Environment

Jie Chen, Guru Venkataramani, Gabriel Parmer
The George Washington University, Washington, DC

Abstract—Debugging an application for power has a wide array of benefits ranging from minimizing the thermal hotspots to reducing the likelihood of CPU malfunction. In this work, we justify the need for power debugging, and show that performance debugging of a parallel application does not automatically guarantee power balance across multiple cores. We perform experiments and show our results using two case study benchmarks, Volrend from Splash-2 and Bodytrack from Parsec-1.0.

Index Terms—Multi-cores, Power Debugging, Power Imbalance.

1 Introduction

Multi-core processors have fueled computing performance to greater scales by offering parallelism on chip. As processors include higher order numbers of cores, power consumption is becoming an important constraint for nearly all types of computer systems.

In particular, uneven power consumption across different cores could complicate heat dissipation and electric supply in multi-core systems [13]. This is because local thermal hotspots are largely dependent on power over time. Also, as processor demands rapidly change the current consumption over a short timeframe, supply voltage perturbations may occur. This leads to power-delivery subsystem having large parasitic inductance causing voltage ripples on the chip's supply lines. As voltage ripples become significant and exceed the tolerance range, CPUs may begin to malfunction [7]. Such undesired effects can be avoided if we balance the power consumption across the cores, and reduce the total power where possible.

Existing research is mostly dedicated for debugging performance and load balance in applications [1], [8]. Although, performance is critical to realizing the promise of multi-core architectures, we note that achieving power balance between the cores is equally important for scalability of systems, and reducing the cooling costs of large scale machines.

The main goal of this article is to motivate the need for power debugging in the multi-core environment. Specifically, for a parallel application running on a multi-core processor, we show that balanced performance between threads does not automatically guarantee power-balance across the cores. We define performance balance as each thread taking equal amount of execution time inside a parallel section, and power balance as each core consuming equal amounts of power (in a Symmetric Multi-core system). The degree of performance (or power) imbalance between threads is measured as the difference between the highest and the lowest execution time (or power). While performance debugging is primarily just about reducing the execution time of a slow running thread, power debugging entails more careful optimizations to distribute power across the threads. For heterogeneous multi-cores, where individual cores can have varying capabilities, careful allocation of power budget to the cores is even more significant. In order to effect proportional and balanced power consumption across multiple cores, we note that understanding the existence of power imbalance is necessary.

The main contributions of our work are:

- We motivate the need for power debugging, and show that a performance balanced program can still suffer from power imbalance.
- We study the power consumption by individual threads on different μarchitectural units and observe that power imbalance can stem from certain units more than others. We also perform studies to explain the power variations by mapping power to the code executed and the events experienced by each thread.
- We conduct experiments using SESC simulator [17], and applications from SPLASH-2 [19] and PARSEC-1.0 [3]. We show our results using two case study benchmarks namely Volrend (SPLASH-2) and Bodytrack (PARSEC-1.0).

2 Power Debugging

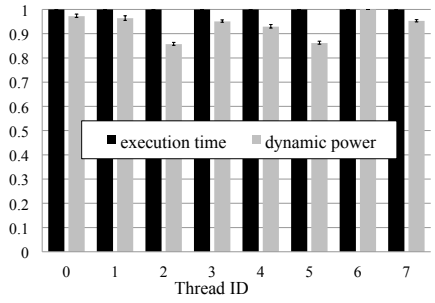
2.1 Motivation

Intuitively, one can think of achieving power balance simply by eliminating performance imbalance between the threads. However, in reality, we find numerous instances of parallel sections from real-world applications

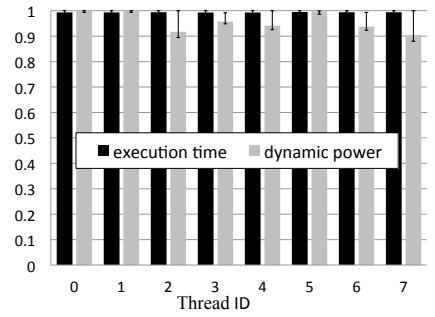
Manuscript submitted: 14-Oct-2011. Manuscript accepted: 30-Dec-2011. Final manuscript received: 30-Dec-2011.

TABLE 1
Performance and Dynamic Power imbalance in SPLASH-2 and PARSEC-1.0 benchmarks with 8 threads.

Application	Parallel Section (File/Function)	Num. of Dynamic Instances	% of Appln. Exec. Time	Avg. Performance Imbalance	Avg. Dynamic Power Imbalance
Volrend (SPLASH-2)	adaptive.c/ray_trace(...)	3	28.39%	0.02%	14.32%
Barnes (SPLASH-2)	load.C/maketree(...)	4	69.03%	1.95%	17.25%
Cholesky (SPLASH-2)	solve.C/Go(...)	1	16.23%	4.64%	36.31%
Bodytrack (PARSEC-1.0)	WorkerGroup.cpp/ WorkerGroup::Run()	82	78.64%	1.08%	9.61%
Canneal (PARSEC-1.0)	annealer_thread.cpp/ annealer_thread::Run()	1	0.08%	1.98%	13.47%

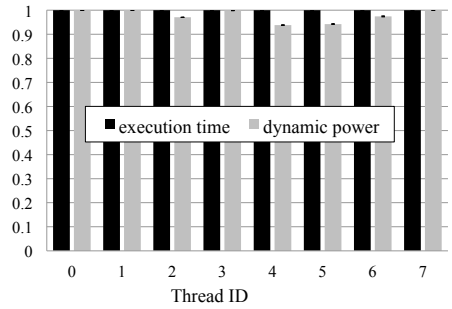


(a) Volrend

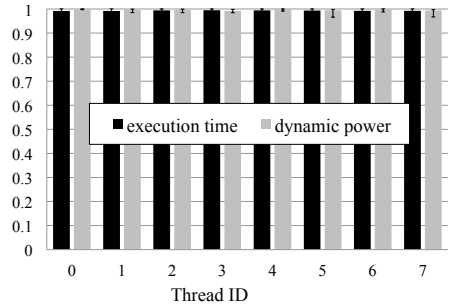


(b) Bodytrack

Fig. 1. Execution time and Dynamic Power of parallel sections normalized to the thread with the highest execution time. Individual cores are 4-wide, out-of-order processors.



(a) Volrend



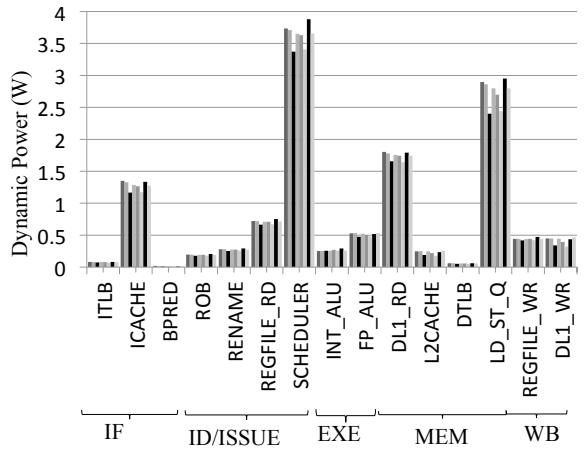
(b) Bodytrack

Fig. 2. Execution time and Dynamic Power of parallel sections normalized to the thread with highest execution time. Individual cores are 4-wide, in-order processors.

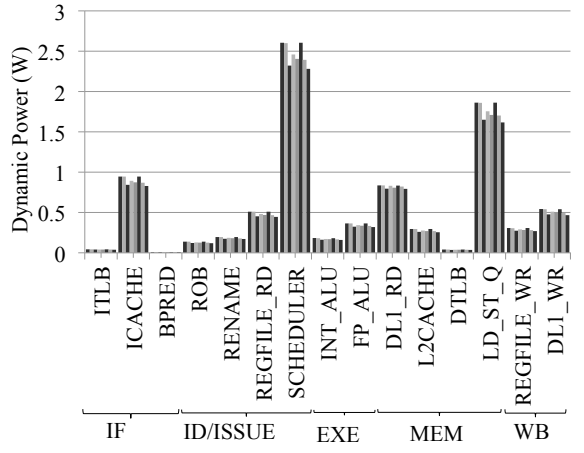
such as SPLASH-2 [19] and PARSEC-1.0 [3], where performance is mostly balanced and the dynamic power remains imbalanced across the cores. We note that both of our benchmark suites have been extensively studied and are well-tuned for performance.

Table 1 shows example parallel sections with 8 threads running on 8 cores and their corresponding performance and power imbalance. In all of our experiments, by default, we use SESC [17], a cycle-accurate, multi-core architecture simulator, to model an 8-core Intel Core i7-like processor running at 3 GHz [9]. Each core is 4-wide, out-of-order with 32 KB private L1 caches, 256 KB private L2 cache and a 16 MB shared L3 cache [12]. For power estimation, we use Wattch model [5]. Furthermore, we focus on dynamic power consumption; we plan to consider leakage power as part of our future work.

In benchmarks like volrend, barnes and bodytrack, certain parallel sections, that account for a significant amount of the total execution time, have less than 2% performance imbalance and up to 17.25% imbalance in dynamic power consumption between the threads. We find that performance balance across the threads can be achieved through hardware optimizations such as out-of-order execution, prefetching data, and so on. How-



(a) Volrend



(b) Bodytrack

Fig. 3. Breakdown of dynamic power consumption by individual μarchitectural units across 8 threads.

ever, the power consumption by individual functional units and the processor cores is still determined by the amount of work to be performed. *Therefore, we make a case that achieving performance balance between threads does not automatically guarantee power balance between the cores*.

We pick two case studies, Volrend and Bodytrack, based on their significant power imbalance despite almost perfect performance balance, and the number of dynamic instances in their parallel section. In volrend, the parallel section under consideration first pre-shades and then non-adaptively raytraces from its node. In bodytrack, the parallel section under consideration first determines the rank value for each thread, fires off the threads that calculate the weight of particles, and later filter them based on calculated weights.

Figure 1 shows the normalized average execution time and power consumption across all the dynamic instances, and the range of normalized values are shown on top of each bar. All the numbers are normalized with respect to the execution time and power of the

TABLE 2
Correlation co-efficient between Scheduler (SCHED) Power and Load/Store Queue (LD_ST_Q) Power against the rate of load instructions.

	Volrend		Bodytrack	
Thread ID	Correlation between rate of Load Instructions and Power consumed by			
	SCHED	LD_ST_Q	SCHED	LD_ST_Q
0	0.99	0.99	0.99	0.98
1	0.97	0.98	0.99	0.98
2	0.98	0.98	0.99	0.99
3	0.99	0.99	0.99	0.99
4	0.97	0.98	0.99	0.99
5	0.97	0.99	0.99	0.98
6	0.98	0.98	0.99	0.99
7	0.85	0.91	0.99	0.99

slowest thread in the corresponding dynamic instance. Our experimental results show that even though the threads have well-balanced execution time ($\approx$ 1%), dynamic power imbalance is still quite high (approximately 14.3% in volrend and 9.6% in bodytrack).

In order to find the reasons behind this power imbalance despite the performance balance across the threads, we repeat our experiments on SESC that models an 8-core processor with in-order cores, while keeping all other parameters unchanged. Figure 2 shows the results. The intuition behind this experiment is that the dynamic power needed to execute a set of instructions should reasonably track performance in in-order cores (especially in the absence of other hardware effects such as out-of-order issue and speculative execution of instructions). Experimentally, we verified that factors such as cache hit and miss rates in L1 and L2, branch misprediction rates remained unchanged between in-order and out-of-order versions. We observed that the average IPC (across cores) of 0.57 in 4-wide out-of-order dropped to 0.18 in 4-wide in-order for volrend, and similarly, the average IPC dropped from 0.89 in 4-wide out-of-order to 0.18 in 4-wide in-order for bodytrack. This proves that hardware optimizations such as out-of-order execution, while improving performance, may not necessarily aid power balance between the cores.

2.2 Breakdown by μarchitectural components

To characterize power consumption behavior by the multi-core application and effect changes in program code for more balanced power consumption across cores, fine-grain information is necessary. We perform experiments to measure the power consumption in individual μarchitectural units across all threads. Figure 3 shows the results of our experiments. We note that the scheduler and load/store queue (LD_ST_Q) are major power consuming units in both benchmarks. For example, in volrend, LD_ST_Q shows up to 23% variation in power between ThreadID #2 and ThreadID #6, even though these threads have similar performance.

To further explain the variations in power consumption between threads on specific μarchitectural units, we correlated power consumed by Scheduler and load/store queue against the rate of load instructions executed by the core. Table 2 shows the results of our experiments. In bodytrack, we observed strong correlation ($\geq$0.98) across all of the threads. In volrend, we find high correlation coefficients ($>$0.9) for LD_ST_Q. From these experiments, we infer that variations in power consumed by these μarchitectural units are caused by the rate of load instructions executed by the core. Note that this behavior may vary across applications– for example, in an application dominated by branch instructions, a branch predictor with higher misprediction rate may lead to power imbalance between different cores.

3 Related Work

A number of prior works have proposed μarchitecture power estimation models. As examples, Wattch [5] provides high-level simulator framework to quantify power for major units of the processor based on structure types and wire delay; McPAT [14] has an integrated power, area, and timing modeling framework for multicore processor configurations; Intel's Architecture Level Power Simulator (ALPS) [18] was developed to profile power consumption from individual Functional Units in Pentium 4 processor chip; IBM's PowerTimer toolset [4] facilitates early-stage power and performance analysis of processor design. We note that such frameworks can be used to observe the power consumption of the μarchitectural units, and hence help programmers with power debugging their software.

Prior works have also studied runtime power estimation. Isci et al. [10] use sampled multimeter power measurements and estimates based on performance counter readings to calibrate power consumption for individual functional units. Powell et al. [16] measure utilization statistics such as IPC, and build linear regression models to estimate activity factor and dynamic power for the core.

To reduce power consumption in multicores, prior works look at putting idle threads into low power states [13], stretching computation of non-critical thread via DVFS [15], [6], [2] and so on. Kumar et al. [11] reduce power consumption on heterogeneous cores by scheduling threads on core with the best performance to power ratio. We note such techniques can be applied as power saving strategies once we identify code regions that are responsible for high power consumption.

4 Conclusions and Future Work

In this work, we propose the need for power debugging, and show that performance debugging of a parallel application does not automatically guarantee power balance across the multiple cores. Our experiments show that significant power variations exist across threads even for programs that have performance balance.

As future work, we plan to develop a hardware-software cooperative framework that shall efficiently aggregate power-related metrics and attribute them to program code. This will help the programmer or the software layers to power debug their code easily. We will expand our studies to other non-scientific and commercial applications that can yield us further insight into power debugging. We will also study scalable ways to extend our framework to large scale systems.

5 Acknowledgments

This material is based upon work supported in part by the National Science Foundation under Grant No. CCF-1117243.

References

[1] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *ISCA*, 2005.

[2] A. Bhattacharjee and M. Martonosi. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. In *ISCA*, 2009.

[3] C. Bienia, S. Kumar, J.P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. *Princeton University Technical Report TR-811-08*, January 2008.

[4] D. Brooks, P. Bose, V. Srinivasan, M. K. Gschwind, P. G. Emma, and M. G. Rosenfield. New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors. *IBM J. Res. Dev.*, 47, September 2003.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA*, 2000.

[6] Q. Cai, J. González, R. Rakvic, G. Magklis, P. Chaparro, and A. González. Meeting points: using thread criticality to adapt multicore hardware to parallel regions. In *PACT*, 2008.

[7] M. S. Gupta, J. L. Oatley, R. Joseph, G-Y Wei, and D. M. Brooks. Understanding voltage variations in chip multiprocessors using a distributed power-delivery network. In *DATE*, 2007.

[8] M. D. Hill and M. R. Marty. Amdahl's law in the multicore era. *Computer*, 41(7), July 2008.

[9] Intel Corporation. Intel Core i7 Processor Family for the LGA-2011 Socket. *Datasheet*, 1, 2011.

[10] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO*, 2003.

[11] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In *ISCA*, 2004.

[12] D. Levinthal. Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 processors. *Intel Performance Analysis Guide*, 2009.

[13] J. Li, J. F. Martinez, and M. C. Huang. The thrifty barrier: Energy-aware synchronization in shared-memory multiprocessors. In *HPCA*, 2004.

[14] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, 2009.

[15] S. Park, W. Jiang, Y. Zhou, and S. Adve. Managing energy-performance tradeoffs for multithreaded applications on multiprocessor architectures. In *SIGMETRICS*, 2007.

[16] M. D. Powell, A. Biswas, J. S. Emer, S. S. Mukherjee, B. R. Sheikh, and S. Yardi. CAMP: A technique to estimate per-structure power at run-time using a few simple parameters. In *HPCA*, 2009.

[17] J. Renau et al. SESC. *http://sesc.sourceforge.net*, 2006.

[18] D. M. Carmean S. H. Gunther, F. Binns and J. C. Hall. Managing the impact of increasing microprocessor power consumption. In *Intel Technology Journal Q1 2001*, 2001.

[19] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *ISCA*, June 1995.