Operating Systems and Asymmetric Single-ISA CMPs: The Potential for Saving Energy

Jeffrey C. Mogul, Jayaram Mudigonda, Nathan Binkert, Partha Ranganathan, Vanish Talwar
HP Laboratories Palo Alto
HPL-2007-140
August 22, 2007*

CMP, multi-core, energy savings, operating systems

CPUs consume too much power. Modern complex cores sometimes waste power on functions that are not useful for the code they run. In particular, operating system kernels do not benefit from many power-consuming features that were intended to improve application performance. We propose using asymmetric single-ISA CMPs (ASISA-CMPs), multicore CPUs where all cores execute the same instruction set architecture but have different performance and power characteristics, to avoid wasting power on operating systems code. We describe various design choices for both hardware and software, describe Linux kernel modifications to support ASISA-CMP, and offer some quantified estimates that support our proposal.

* Internal Accession Date Only

Operating Systems and Asymmetric Single-ISA CMPs: The Potential for Saving Energy

Jeffrey C. Mogul Jayaram Mudigonda Nathan Binkert Partha Ranganathan Vanish Talwar
Jeff.Mogul@hp.com, Jayaram.Mudigonda@hp.com, binkert@hp.com, Partha.Ranganathan@hp.com, Vanish.Talwar@hp.com
HP Labs, Palo Alto, CA 94304

Abstract

CPUs consume too much power. Modern complex cores sometimes waste power on functions that are not useful for the code they run. In particular, operating system kernels do not benefit from many power-consuming features that were intended to improve application performance. We propose using asymmetric single-ISA CMPs (ASISA-CMPs), multicore CPUs where all cores execute the same instruction set architecture but have different performance and power characteristics, to avoid wasting power on operating systems code. We describe various design choices for both hardware and software, describe Linux kernel modifications to support ASISA-CMP, and offer some quantified estimates that support our proposal.

1 Introduction

While Moore's Law has delivered exponential increases in computation over the past few decades, two well-known trends create problems for computer systems: CPUs consume more and more power, and operating systems do not speed up as rapidly as most application code does. Many people have addressed these problems separately; we propose to address them together.

Until recently, designers of high-end CPU chips tended to improve single-stream performance as much as possible, by exploiting instruction-level parallelism and decreasing cycle times. Both of these techniques are now hard to sustain, so recent CPU designs exploit shrinking VLSI feature sizes by using multiple cores, rather than faster clocks. Examples of these *Chip Multi-Processors* (CMPs) include the Sun Niagra processor with eight cores, the quad-core Intel Xeon, and dual-core systems from several vendors.

All commercially-available general-purpose CMPs, as of mid-2007, are *symmetrical*: each CPU is identical, and typically, all run at the same clock rate. However, in 2003 Kumar *et al.* [13] proposed heterogeneous (or asymmetrical) multi-core processors, as a way of reducing power requirements. Their proposal retains the *single-Instruction-Set-Architecture* (single-ISA) model of symmetrical CMPs: all cores can execute the same machine code. They observed that different implementations of the same ISA had order-of-magnitude differences in power consumption (assuming a single VLSI process). They further observed that in a multi-application workload, or even in phases of a single application, one does not always need the full power and functionality of the most complex CPU core; if a CMP could switch a process between cores with different capabilities, one could maintain throughput while decreasing power consumption.

Since the original study by Kumar *et al.*, several other studies [3, 8, 14] have highlighted the benefits from heterogeneity. (Keynote speakers from some major processor vendors have also suggested that heterogeneity might be commercially interesting [2, 22].) However, all these studies looked only at user-mode execution. But we know that many workloads spend much or most of their cycles in the operating system [23]. We also know that operating system (OS) code differs from application code: it avoids the floating point processor, it branches more often, and it has lower cache locality (all reasons why OS speedups lag application speedups on modern CPUs). An *asymmetric single-ISA CMP* (ASISA-CMP) might therefore save power, without reducing throughput, by executing OS code on a simple, low-power core, while using more complex, high-power cores for application code.

The main contribution of this paper is to propose and evaluate the ASISA-CMP model, in which (1) a multi-core, single-ISA CPU includes some "OS-friendly" cores, optimized to execute OS code without wasting energy, and (2) the OS statically and/or dynamically decides which code to execute on which cores, so as to optimize throughput per joule.

To optimally exploit ASISA-CMP, we expect that the OS and the hardware both must change. This paper explores the various design considerations for co-evolving the OS and hardware, and presents experimental results.

2 Related work

Ousterhout [20] may have been the first to point out that "operating system performance does not seem to be improving at the same rate as the base speed of the underlying hardware." He speculated that causes include memory latencies and context-switching overheads.

Nellans *et al.* [19] measured the fraction of cycles spent in the OS for a variety of applications, and re-examined how OS performance scales with CPU performance, suggesting that interrupt-handling code interferes with caches and branch-prediction history tables. They found that many applications execute a large fraction of cycles in the OS, and observed that "a classic 5 stage pipeline [such as] a 486 is surprisingly close in performance to a modern Pentium 4 when executing [OS code]." However, instead of proposing an ASISA-CMP, they suggest adding a dedicated OS-specific core. (It is not entirely clear how far their proposal is from a single-ISA CMP.) They did not evaluate this proposal in detail.

Chakraborty *et al.* [11] proposed refactoring software so that similar "fragments" of code are executed on the same core of a CMP. Their initial study treated the OS and the user-mode application as two coarse-grained fragments, and found speedup in some cases. However, they did not examine asymmetric CMPs or the question of power reduction.

Sanjay Kumar *et al.* [15] propose a "sidecore" architecture to support hypervisor operations. In their approach, a hypervisor is restructured into multiple components, with some running on specialized cores. Their goal was to avoid the expensive internal state changes triggered via traps (e.g., VMexit in Intel's VT architecture) to perform privileged hypervisor functions. Instead, the sidecore approach transfers the operation to a remote core "that is already in the appropriate state." This also avoids polluting the guest-core caches with code and data from hypervisor operations. (The sidecore approach is not specifically targeted at saving energy.)

3 Design overview

Our goal is to address two major challenges for multi-core systems: how to minimize power consumption while maintaining good performance, and how to exploit the parallelism offered by a multi-core CPU. These two issues are closely linked, but we will try to untangle them somewhat.

3.1 Proportionality in power consumption

We want to maximize the energy efficiency of our computer systems, which could be expressed as the useful computational work (throughput) per joule expended. Fan *et al.* [6] have observed that the ideal system would consume power directly proportional to the rate of useful computational work it completes. We refer to this as the "proportionality rule." Such a system would need no additional power management algorithms, except as might be needed to avoided exceeding peak power or cooling limits.

Fan *et al.* argue that "system designers should consider power efficiency not simply at peak performance levels but across the activity range." We believe that the ASISA-CMP approach, with a careful integration of OS and hardware design, can help address this goal. Of course, it is probably impossible to design a system that truly complies with the proportionality rule, especially since many components consume considerable power even when the CPU is idle.

There are at least two ways that one might design a system to address the proportionality rule. First, one could design individual components whose power consumption varies with throughput, such as a CPU that supports voltage and frequency scaling. Second, one could design a system with a mix of both high-power/high-performance and low-power/low-performance components, with a mechanism for dynamically varying which components are used (and powered up) based on system load.

The original ASISA-CMP model, as proposed by Kumar *et al.* [13], follows the second approach, without precluding the first one. In times of light load, activity shifts to low-power cores; in times of heavy load, low-power cores can offer additional parallelism without significant increases in area or power consumption.

In this paper, we extend the ASISA-CMP model by asserting that the ideal low-power core is one that is specialized to execute operating system code. (More broadly, we consider "OS-like code," which we will define in Sec. 4.3.1.) This stems from several observations:

- OS code does not proportionately benefit from the potential speedup of complex, high-frequency cores. Thus, running OS code on a simpler core is a better use of power and chip area.
- Most computer systems (with certain exceptions, such as scientific computing) are often idle. If we could power down complex CPU cores during periods when they would otherwise be idle, we could improve proportionality.

The designs explored in this paper include:

- Multi-core CPUs with a mix of high-power, high-complexity application cores, and low-power, low-complexity OS-friendly cores.
- Operating system modifications to dynamically shift load to the most appropriate core, and (potentially) to power down idle cores.
- Modest hardware changes to improve the efficiency of core-switching.

Of course, the CPU is not the only power-consuming component in a system, and ASISA-CMP does not address the power consumed by memory, busses, and I/O devices, or the power wasted in power supplies and cooling systems. Therefore, even if the CPU were perfectly proportional, the entire system would still fail to meet the proportionality rule. However, as long as CPUs represent the largest single power draws in a system (see Sec. 3.3.1), improving their proportionality is worthwhile.

3.2 Core heterogeneity

The promise of ASISA-CMP depends critically on two facts of CPU core design: (1) for a given process technology, a complex core consumes much more power and die area than a simple core, and (2) a complex core does not improve OS performance nearly as much as it improves application performance.

Table 1 shows the relative power consumption, performance (in terms of instructions per cycle, or IPC), and sizes of various generations of Alpha cores, scaled as if all were implemented in the same process technology. Clearly, the smallest core delivers significantly more performance per watt and per mm^2. In fact, these performance results were based on the SPEC CPU benchmark suite; since operating system performance generally scales worse than application performance [20], we believe the IPC ratios would be even smaller for OS code.

Table 1: Power and relative performance of Alpha cores

Alpha core	**Peak power**	**Average power**	**Normalized vs. EV4 IPC**	**area**	**power**
EV4	4.97W	3.73W	1.00	1.00	1.00
EV5	9.83W	6.88W	1.30	1.76	1.84
EV6	17.8W	10.68W	1.87	8.54	2.86
EV8	92.88W	46.44W	2.14	82.2	12.45

All cores scaled to 0.1 μm; IPC based on SPEC CPU benchmarks
Based on data from Kumar *et al.* [13]

3.3 Complicating issues

Various issues complicate the question of whether we can improve throughput/joule by running OS (or OS-like) code on special a OS-friendly core. We cover many details in subsequent sections of this paper; here, we expose some general questions. Many of these can only be resolved by experimentation (possibly through simulation); we describe our experiments later, in Sec. 7.

The two key issues, as mentioned above, are the relative power consumption levels for various system components, and the relative performance costs and benefits of switching cores. In order for ASISA-CMP to pay off, we require that it does not reduce performance faster than it reduces power consumption.

3.3.1 How important is CPU power?

Any real system includes multiple components that draw power, and ASISA-CMP will not significantly change the energy consumption of components other than the CPU. In fact, if ASISA-CMP increases the time required to complete a job (or set of jobs), the resulting increase in energy consumed by other system components may outweight the savings from the CPU cores.

Component power consumption varies tremendously across the variety of computer systems in use. In particular,

Table 2: Example power budgets for two typical systems

System	Watts						
	Tot.	CPU	(%)	Mem	Disk	PCI	Other
Blade servers (all with multiple CPUs; after [16])							
Small	248	70	28%	48	10	50	70
Med.	442	170	39%	112	10	50	100
Large	1025	520	51%	320	10	75	100
Laptop (after [17])							
Idle	13.1	2.0	15%	0.4	0.6	N.A.	10.1
Busy	25.8	13.4	52%	1	1	N.A.	10.3

laptops and servers have vastly different balances between components. Table 2 shows a power breakdown for several typical systems. The blade server results, taken from [16], show "nameplate" (maximum) power budgets, and all have either two or (for the "Large" configuration) four CPUs. The laptop results, taken from [17], show measured results for an idle system and for one running the PCMark CPU benchmark; in both cases, Dynamic Voltage Scaling was disabled and the screen was at full brightness.

In all cases, except for the idle laptop, CPU power consumption was the largest single component of system power consumption. For the Large server and the busy laptop, the CPU (or CPUs) consumed slightly more than half of the total power. This suggests that techniques, such as ASISA-CMP, that address CPU power consumption can have meaningful effects on whole-system power consumption.

3.3.2 How does ASISA-CMP affect performance?

ASISA-CMP can affect performance in several ways:

- **Running OS code on a slower CPU**: By design, ASISA-CMP concedes some performance by running OS code on a slower core. As argued in Sec. 3.2, this slowdown might be minimal. However, an application that spends much of its time in the OS could see a significant performance decline.
- **Core switching costs**: ASISA-CMP inherently moves a thread of execution from one core to another for certain system calls. Core-switching creates longer code paths for these system calls, and adds state-saving overhead.
- **Cache affinity vs. cache interference**: We assume that the cores in a CMP CPU share a single L2 cache but have private L1 caches. Core-switching could affect cache performance in at least two ways: it could harm cache affinity, by requiring cache lines (e.g., for the data buffer of a **write** system call) to move between L1 caches, or it could reduce cache interference, by keeping some OS code and data out of the application core's cache.
- **Available parallelism**: Given that the incremental cost (in power and area) for adding an OS core to a CMP is much lower than for an application core (see Sec. 3.2), if there is available parallelism in the workload that extend to OS processing, an ASISA-CMP CPU could support more parallelism than a symmetric CMP CPU with similar power and area. For example, a parallel "make" command might benefit from having an OS core run I/O processing while the application core is dedicated to an optimizing compiler. Not all workloads will have this kind of parallelism, of course.

3.3.3 Idle time

We have described the example in Figure 1 as if the application's system call does useful work. However, it could also be blocked waiting for some external event, such as disk I/O or the arrival of a Web request. Numerous studies (e.g., [6, 21]) have shown that most computers are idle most of the time. Therefore, as pointed out by Fan *et al.* [6], a useful design for meeting the portionality rule must significantly reduce power consumption during idle periods.

ASISA-CMP offers the option of powering down the high-power application core(s), while maintaining OS functions on a low-power core. For example, the arrival of a new Web (HTTP/TCP) connection normally precedes the arrival of the actual HTTP request by at least one network round-trip time (typically on the order of milliseconds or more). This would allow an OS core to handle the initial TCP connection request and then awaken the application core soon enough to process the HTTP-level request without any delay.

3.4 Competing approaches

Given that the goal of ASISA-CMP is to improve performance (in terms of throughput/joule), and it would require changes to the design of CPU chips, we have to compare it against possible competing approaches.

Other potential alternatives include:

- **Complex-core with dynamic Voltage/Frequency**

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.