# Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction

Rakesh Kumar[†],Keith I. Farkas[‡],Norman P. Jouppi[‡],Parthasarathy Ranganathan[‡],Dean M. Tullsen[†]

[†]Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114

[‡]HP Labs
1501 Page Mill Road
Palo Alto, CA 94304

## Abstract

*This paper proposes and evaluates single-ISA heterogeneous multi-core architectures as a mechanism to reduce processor power dissipation. Our design incorporates heterogeneous cores representing different points in the power/performance design space; during an application's execution, system software dynamically chooses the most appropriate core to meet specific performance and power requirements.*

*Our evaluation of this architecture shows significant energy benefits. For an objective function that optimizes for energy efficiency with a tight performance threshold, for 14 SPEC benchmarks, our results indicate a 39% average energy reduction while only sacrificing 3% in performance. An objective function that optimizes for energy-delay with looser performance bounds achieves, on average, nearly a factor of three improvement in energy-delay product while sacrificing only 22% in performance. Energy savings are substantially more than chip-wide voltage/frequency scaling.*

## 1 Introduction

As processors continue to increase in performance and speed, processor power consumption and heat dissipation have become key challenges in the design of future high-performance systems. For example, Pentium-4 class processors currently consume well over 50W and processors in the year 2015 are expected to consume close to 300W [1]. Increased power consumption and heat dissipation typically leads to higher costs for thermal packaging, fans, electricity, and even air conditioning. Higher-power systems can also have a greater incidence of failures.

In this paper, we propose and evaluate a *single-ISA heterogeneous multi-core architecture* [26, 27] to reduce processor power dissipation. Prior chip-level multiprocessors (CMP) have been proposed using multiple copies of the same core (i.e., homogeneous), or processors with co-processors that execute a different instruction set. We propose that for many applications, core diversity is of higher value than uniformity, offering much greater ability to adapt to the demands of the application(s). We present a multi-core architecture where all cores execute the same instruction set, but have different capabilities and performance levels. At run time, system software evaluates the resource requirements of an application and chooses the core that can best meet these requirements while minimizing energy consumption. The goal of this research is to identify and quantify some of the key advantages of this novel architecture in a particular execution environment.

One of the motivations for this proposal is that different applications have different resource requirements during their execution. Some applications may have a large amount of instruction-level parallelism (ILP), which can be exploited by a core that can issue many instructions per cycle (i.e., a wide-issue superscalar CPU). The same core, however, might be wasted on an application with little ILP, consuming significantly more power than a simpler core that is better matched to the characteristics of the application.

A heterogeneous multi-core architecture could be implemented by designing a series of cores from scratch, by reusing a series of previously-implemented processor cores after modifying their interfaces, or a combination of these two approaches. In this paper, we consider the reuse of existing cores, which allows previous design effort to be amortized. Given the growth between generations of processors from the same architectural family, the entire family can typically be incorporated on a die only slightly larger than that required by the most advanced core.

In addition, clock frequencies of the older cores would scale with technology, and would be much closer to that of the latest processor technology than their original implementation clock frequency. Then, the primary criterion for selecting between different cores would be the performance of each architecture and the resulting energy dissipation.

In this paper, we model one example of a single-ISA heterogeneous architecture – it includes four representative cores (two in-order cores and two out-of-order cores) from an ordered complexity/performance continuum in the Alpha processor roadmap. We show that typical applications not only place highly varied demands on an execution architecture, but also that that demand can vary between phases of the same program. We assume the ability to dynamically switch between cores. This allows the architecture to adapt to differences between applications, differences between phases in the same application, or changing priorities of the processor or workload over time. We show reductions in processor energy-delay product as high as 84% (a six-fold improvement) for individual applications, and 63% overall. Energy-delay$^2$ (the product of energy and the square of the delay) reductions are as high as 75% (a four-fold improvement), and 50% overall. Chip-wide voltage/frequency scaling can do no better than break even on this metric. We examine oracle-driven core switching, to understand the limits of this approach, as well as realistic runtime heuristics for core switching.

The rest of the paper is organized as follows. Section 2 discusses the single-ISA heterogeneous multi-core architecture that we study. Section 3 describes the methodology used to study performance and power. Section 4 discusses the results of our evaluation while Section 5 discusses related work. Section 6 summarizes the work and discusses ongoing and future research.

## 2 Architecture

This section gives an overview of a potential heterogeneous multi-core architecture and core-switching approach.

The architecture consists of a chip-level multiprocessor with multiple, diverse processor cores. These cores all execute the same instruction set, but include significantly different resources and achieve different performance and energy efficiency on the same application. During an application's execution, the operating system software tries to match the application to the different cores, attempting to meet a defined objective function. For example, it may be trying to meet a particular performance requirement or goal, but doing so with maximum energy efficiency.

### 2.1 Discussion of Core Switching

There are many reasons why the best core for execution may vary over time. The demands of executing code vary widely between applications; thus, the best core for one application will often not be the best for the next, given a particular objective function (assumed to be some combination of energy and performance). In addition, the demands of a single application can also vary across phases of the program.

Even the objective function can change over time, as the processor changes power conditions (e.g., plugged vs. unplugged, full battery vs. low battery, thermal emergencies), as applications switch (e.g., low priority vs. high priority job), or even within an application (e.g., a real-time application is behind or ahead of schedule).

The experiments in this paper explore only a subset of these possible changing conditions. Specifically, it examines adaptation to phase changes in single applications. However, by simulating multiple applications and several objective functions, it also indirectly examines the potential to adapt to changing applications and objective functions. We believe a real system would see far greater opportunities to switch cores to adapt to changing execution and environmental conditions than the narrow set of experiments exhibited here.

This work examines a diverse set of execution cores. In a processor where the objective function is static (and perhaps the workload is well known), some of our results indicate that a smaller set of cores (often two) will suffice to achieve very significant gains. However, if the objective function varies over time or workload, a larger set of cores has even greater benefit.

### 2.2 Choice of cores.

To provide an effective platform for a wide variety of application execution characteristics and/or system priority functions, the cores on the heterogeneous multi-core processor should cover both a wide and evenly spaced range of the complexity/performance design space.

In this study, we consider a design that takes a series of previously implemented processor cores with slight changes to their interface – this choice reflects one of the key advantages of the CMP architecture, namely the effective amortization of design and verification effort. We include four Alpha cores – EV4 (Alpha 21064), EV5 (Alpha 21164), EV6 (Alpha 21264) and a single-threaded version of the EV8 (Alpha 21464), referred to as EV8-. These cores demonstrate strict gradation in terms of complexity and are capable of sharing a single executable. We assume the four cores have private L1 data and instruction caches and share a common L2 cache, phase-lock loop circuitry, and pins.

We chose the cores of these off-the-shelf processors due to the availability of real power and area data for these processors, except for the EV8 where we use projected numbers [10, 12, 23, 30]. All these processors have 64-bit archi-

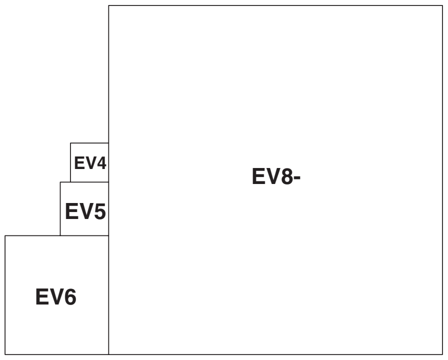

**Figure 1. Relative sizes of the cores used in the study**

tectures. Note that technology mapping across a few generations has been shown to be feasible [24].

Figure 1 shows the relative sizes of the cores used in the study, assuming they are all implemented in a 0.10 micron technology (the methodology to obtain this figure is described in the next section). It can be seen that the resulting core is only modestly (within 15%) larger than the EV8-core by itself.

Minor differences in the ISA between processor generations are handled easily. Either programs are compiled to the least common denominator (the EV4), or we use software traps for the older cores. If extensive use is made of the software traps, our mechanisms will naturally shy away from those cores, due to the low performance.

For this research, to simplify the initial analysis of this new execution paradigm, we assume only one application runs at a time on only one core. This design point could either represent an environment targeted at a single application at a time, or modeling policies that might be employed when a multithreaded multi-core configuration lacks thread parallelism. Because we assume a maximum of one thread running, the multithreaded features of EV8 are not needed. Hence, these are subtracted from the model, as discussed in Section 3. In addition, this assumption means that we do not need more than one of any core type. Finally, since only one core is active at a time, we implement cache coherence by ensuring that dirty data is flushed from the current core's L1 data cache before execution is migrated to another core.

This particular choice of architectures also gives a clear ordering in both power dissipation and expected performance. This allows the best coverage of the design space for a given number of cores and simplifies the design of core-switching algorithms.

### 2.3 Switching applications between cores.

Typical programs go through phases with different execution characteristics [35, 39]. Therefore, the best core during one phase may not be best for the next phase. This observation motivates the ability to dynamically switch cores in mid execution to take full advantage of our heterogeneous architecture.

There is a cost to switching cores, so we must restrict the granularity of switching. One method for doing this would switch only at operating system timeslice intervals, when execution is in the operating system, with user state already saved to memory. If the OS decides a switch is in order, it powers up the new core, triggers a cache flush to save all dirty cache data to the shared L2, and signals the new core to start at a predefined OS entry point. The new core would then power down the old core and return from the timer interrupt handler. The user state saved by the old core would be loaded from memory into the new core at that time, as a normal consequence of returning from the operating system. Alternatively, we could switch to different cores at the granularity of the entire application, possibly chosen statically. In this study, we consider both these options.

In this work, we assume that unused cores are completely powered down, rather than left idle. Thus, unused cores suffer no static leakage or dynamic switching power. This does, however, introduce a latency for powering a new core up. We estimate that a given processor core can be powered up in approximately one thousand cycles of the 2.1GHz clock. This assumption is based on the observation that when we power down a processor core we do not power down the phase-lock loop that generates the clock for the core. Rather, in our multi-core architecture, the same phase-lock loop generates the clocks for all cores. Consequently, the power-up time of a core is determined by the time required for the power buses to charge and stabilize. In addition, to avoid injecting excessive noise on the power bus bars of the multi-core processor, we assume a staged power up would be used.

In addition, our experiments confirm that switching cores at operating-system timer intervals ensures that the switching overhead has almost no impact on performance, even with the most pessimistic assumptions about power-up time, software overhead, and cache cold start effects. However, these overheads are still modeled in our experiments in Section 4.4.

## 3 Methodology

This section discusses the various methodological challenges of this research, including modeling the power, the real estate, and the performance of the heterogeneous multi-core architecture.

| Processor | EV4 | EV5 | EV6 | EV8- |
|---|---|---|---|---|
| Issue-width | 2 | 4 | 6 (OOO) | 8 (OOO) |
| I-Cache | 8KB, DM | 8KB, DM | 64KB, 2-way | 64KB, 4-way |
| D-Cache | 8KB, DM | 8KB, DM | 64KB, 2-way | 64KB, 4-way |
| Branch Pred. | 2KB,1-bit | 2K-gshare | hybrid 2-level | hybrid 2-level (2X EV6 size) |
| Number of MSHRs | 2 | 4 | 8 | 16 |

**Table 1. Configuration of the cores**

### 3.1 Modeling of CPU Cores

The cores we simulate are roughly modeled after cores of EV4 (Alpha 21064), EV5 (Alpha 21164), EV6 (Alpha 21264) and EV8-. EV8- is a hypothetical single-threaded version of EV8 (Alpha 21464). The data on the resources for EV8 was based on predictions made by Joel Emer [12] and Artur Klauser [23], conversations with people from the Alpha design team, and other reported data [10, 30]. The data on the resources of the other cores are based on published literature on these processors [2, 3, 4].

The multi-core processor is assumed to be implemented in a 0.10 micron technology. The cores have private first-level caches, and share an on-chip 3.5 MB 7-way set-associative L2 cache. At 0.10 micron, this cache will occupy an area just under half the die size of the Pentium 4. All the cores are assumed to run at 2.1GHz. This is the frequency at which an EV6 core would run if its 600MHz, 0.35 micron implementation was scaled to a 0.10 micron technology. In the Alpha design, the amount of work per pipe stage was relatively constant across processor generations [7, 11, 12, 15]; therefore, it is reasonable to assume they can all be clocked at the same rate when implemented in the same technology (if not as designed, processors with similar characteristics certainly could). The input voltage for all the cores is assumed to be 1.2V.

Note that while we took care to model real architectures that have been available in the past, we could consider these as just sample design points in the continuum of processor designs that could be integrated into a heterogeneous multiple-core architecture. These existing designs already display the diversity of performance and power consumption desired. However, a custom or partially custom design would have much greater flexibility in ensuring that the performance and power space is covered in the most appropriate manner, but sacrificing the design time and verification advantages of the approach we follow in this work.

Table 1 summarizes the configurations that were modeled for various cores. All architectures are modeled as accurately as possible, given the parameters in Table 1, on a highly detailed instruction-level simulator. However, we did not faithfully model every detail of each architecture; we were most concerned with modeling the approximate spaces each core covers in our complexity/performance continuum.

Specific instances of deviations from exact design parameters include the following. Associativity of the EV8- caches is double the associativity of equally-sized EV6 caches. EV8- uses a tournament predictor double the size of the EV6 branch predictor. All the caches are assumed to be non-blocking, but the number of MSHRs is assumed to double with successive cores to adjust to increasing issue width. All the out-of-order cores are assumed to have big enough re-order buffers and large enough load/store queues to ensure no conflicts for these structures.

The various miss penalties and L2 cache access latencies for the simulated cores were determined using CACTI. CACTI [37] provides an integrated model of cache access time, cycle time, area, aspect ratio, and power. To calculate the penalties, we used CACTI to get access times and then added one cycle each for L1-miss detection, going to L2, and coming from L2. For calculating the L2 access time, we assume that the L2 data and tag access are serialized so that the data memories don't have to be cycled on a miss and only the required set is cycled on a hit. Memory latency was set to be 150ns.

### 3.2 Modeling Power

Modeling power for this type of study is a challenge. We need to consider cores designed over the time span of more than a decade. Power depends not only on the configuration of a processor, but also on the circuit design style and process parameters. Also, actual power dissipation varies with activity, though the degree of variability again depends on the technology parameters as well as the gating style used.

No existing architecture-level power modeling framework accounts for all of these factors. Current power models like Wattch [8] are primarily meant for activity-based architectural level power analysis and optimizations within a single processor generation, not as a tool to compare the absolute power consumption of widely varied architectures. We integrated Wattch into our architectural simulator and simulated the configuration of various cores implemented in their original technologies to get an estimate of the maximum power consumption of these cores as well as the typical power consumption running various applications. We found that Wattch did not, in general, reproduce published peak and typical power for the variety of processor configurations we are using.

Therefore we use a hybrid power model that uses estimates from Wattch, along with additional scaling and offset factors to calibrate for technology factors. This model not only accounts for activity-based dissipation, but also accounts for the design style and process parameter differences by relying on measured datapoints from the manufacturers.

To solve for the calibration factors, this methodology requires peak and typical power values for the actual processors and the corresponding values reported by Wattch. This allows us to establish scaling factors that use the output of Wattch to estimate the actual power dissipation within the expected range for each core. To obtain the values for the processor cores, we derive the values from the literature; Section 3.2.1 discusses our derivation of peak power, and Section 3.2.2 discusses our derivation of typical power. For the corresponding Wattch values, we estimate peak power for each core given peak activity assumptions for all the hardware structures, and use the simulator to derive typical power consumed for SPEC2000 benchmarks.

This methodology then both reproduces published results and scales reasonably accurately with activity. While this is not a perfect power model, it will be far more accurate than using Wattch alone, or relying simply on reported average power.

### 3.2.1 Estimating Peak Power

This section details the methodology for estimating peak power dissipation of the cores. Table 2 shows our power and area estimates for the cores. We start with the peak power data of the processors obtained from data sheets and conference publications [2, 3, 4, 10, 23]. To derive the peak power dissipation in the core of a processor from the published numbers, the power consumed in the L2 caches and at the output pins of the processor must be subtracted from the published value. Power consumption in the L2 caches under peak load was determined using CACTI, starting by finding the energy consumed per access and dividing by the effective access time. Details on *bitouts*, the extent of pipelining during accesses, etc. were obtained from data sheets (except for EV8-). For the EV8 L2, we assume 32 byte (288 bits including ECC) transfers on reads and writes to the L1 cache. We also assume the L2 cache is doubly pumped.

The power dissipation at the output pins is calculated using the formula: $P = (1/2)CV^2f$.

The values of V (bus voltage), f (effective bus frequency) and C (load capacitance) were obtained from data sheets. Effective bus frequency was calculated by dividing the peak bandwidth of the data bus by the maximum number of data output pins which are active per cycle. The address bus was assumed to operate at the same effective frequency. For processors like the EV4, the effective frequency of the bus connecting to the off-chip cache is different from the effective frequency of the system bus, so power must be calculated separately for those buses. We assume the probability that a bus line changes state is 0.5. For calculating the power at the output pins of EV8, we used the projected values for V and f. We assumed that half of the pins are input pins. Also, we assume that pin capacitance scales as the square root of the technology scaling factor. Due to reduced resources, we assumed that the EV8- core consumes 80% of the calculated EV8 core-power. This reduction is primarily due to smaller issue queues and register files. The power data was then scaled to the 0.10 micron process. For scaling, we assumed that power dissipation varies directly with frequency, quadratically with input voltage, and is proportional to feature-size.

The second column in Table 2 summarizes the power consumed by the cores at 0.10 micron technology. As can be seen from the table, the EV8- core consumes almost 20 times the peak power and more than 80 times the real estate of the EV4 core.

CACTI was also used to derive the energy per access of the shared L2 cache, for use in our simulations. We also estimated power dissipation at the output pins of the L2 cache due to L2 misses. For this, we assume 400 output pins. We assume a load capacitance of 50pF and a bus voltage of 2.5V. Again, an activity factor of 0.5 for bit-line transitions is assumed. We also ran some experiments with a detailed model of off-chip memory access power, but found that the level of off-chip activity is highly constant across cores, and did not impact our results.

### 3.2.2 Estimating Typical Power

Values for typical power are more difficult to obtain, so we rely on a variety of techniques and sources to arrive at these values.

Typical power for the EV6 and EV8- assume similar peak to typical ratios as published data for Intel processors of the same generation (the 0.13 micron Pentium 4 [5] for EV8-, and the 0.35 micron late-release Pentium Pro [18, 22] for the EV6).

EV4 and EV5 typical power is extrapolated from these results and available thermal data [2, 3] assuming a approximately linear increase in power variation over time, due to wider issue processors and increased application of clock gating.

These typical values are then scaled in similar ways to the peak values (but using measured typical activity) to derive the power for the cores alone. Table 2 gives the derived typical power for each of our cores. Also shown, for each core, is the range in power demand for the actual applications we run, expressed as a percentage of typical power.

DOCKET ALARM

# 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.