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

Proceedings of the 36th International Symposium on Microarchitecture (MICRO-36 2003)
0-7695-2043-X/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

Petitioner Samsung Ex-1032, 0001

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-

IEEE COMPUTER SOCIETY

Petitioner Samsung Ex-1032, 0002

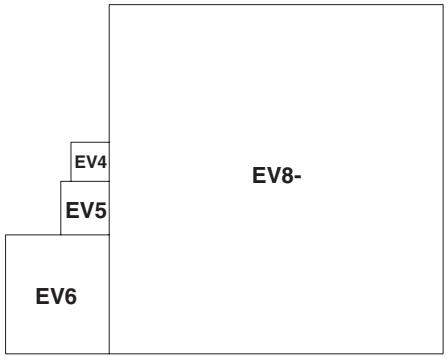

Figure 1. Relative sizes of the cores used in the study

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.

IEEE COMPUTER SOCIETY

Petitioner Samsung Ex-1032, 0003

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.

IEEE COMPUTER SOCIETY

Petitioner Samsung Ex-1032, 0004

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.

IEEE COMPUTER SOCIETY

Petitioner Samsung Ex-1032, 0005

Core	Peak-power (Watts)	Core-area (mm^2)	Typical-power (Watts)	Range (%)
EV4	4.97	2.87	3.73	92-107
EV5	9.83	5.06	6.88	89-109
EV6	17.80	24.5	10.68	86-113
EV8-	92.88	236	46.44	82-128

Table 2. Power and area statistics of the cores

Program	Description
ammp	Computational Chemistry
applu	Parabolic/Elliptic Partial Differential Equations
apsi	Meteorology:Pollutant Distribution
art	Image Recognition/Neural Networks
bzip2	Compression
crafty	Game Playing:Chess
eon	Computer Visualization
equake	Seismic Wave Propagation Simulation
fma3d	Finite-element Crash Simulation
gzip	Compression
mcf	Combinatorial Optimization
twolf	Place and Route Simulator
vortex	Object-oriented Database
wupwise	Physics/Quantum Chromodynamics

Table 3. Benchmarks simulated.

3.2.3 Power Model Sensitivity

While our methodology includes several assumptions based on common rules-of-thumb used in typical processor design, we performed several sensitivity experiments with widely different assumptions about the range of power dissipation in the core. Our results show very little difference in the qualitative results in this research. *For any reasonable assumptions about the range, the power differences between cores still dominates the power difference between applications on the same core.* Furthermore, as noted previously, the cores can be considered as just sample design points in the continuum of processor designs that could be integrated into a heterogeneous multiple-core architecture.

3.3 Estimating Chip Area

Table 2 also summarizes the area occupied by the cores at 0.10 micron (also shown in Figure 1). The area of the cores (except EV8-) is derived from published photos of the dies after subtracting the area occupied by I/O pads, interconnection wires, the bus-interface unit, L2 cache, and control logic. Area of the L2 cache of the multi-core processor is estimated using CACTI.

The die size of EV8 was predicted to be 400 mm^2 [33]. To determine the core size of EV8-, we subtract out the estimated area of the L2 cache (using CACTI). We also account for reduction in the size of register files, instruction queues, reorder buffer, and renaming tables to account for the single-threaded EV8-. For this, we use detailed models of the register bit equivalents (rbe) [31] for register files, reorder buffer and renaming tables at the original and reduced sizes. The sizes of the original and reduced instruction queue sizes were estimated from examination of MIPS R10000 and HP PA-8000 data [9, 25], assuming that the area grows more than linear with respect to the number of entries ($num_entries^{1.5}$). The area data is then scaled for the 0.10 micron process.

3.4 Modeling Performance

In this paper, we simulate the execution of 14 benchmarks from the SPEC2000 benchmark suite, including 7 from SPECint and 7 from SPECfp. These are listed in Table 3.

Benchmarks are simulated using SMTSIM, a cycle-accurate, execution-driven simulator that simulates an out-of-order, simultaneous multithreading processor [38], used in non-multithreading mode for this research. SMTSIM executes unmodified, statically linked Alpha binaries. The simulator was modified to simulate a multi-core processor comprising four heterogeneous cores sharing an on-chip L2 cache and the memory subsystem.

In all simulations in this research we assume a single thread of execution running on one core at a time. Switching execution between cores involves flushing the pipeline of the "active" core and writing back all its dirty L1 cache lines to the L2 cache. The next instruction is then fetched into the pipeline of the new core. The execution time and energy of this overhead, as well as the startup effects on the new core, are accounted for in our simulations of the dynamic switching heuristics in Section 4.4.

The simpoint tool [36] is used to determine the number of committed instructions which need to be fast-forwarded so as to capture the representative program behavior during simulation. After fast-forwarding, we simulate 1 billion instructions. All benchmarks are simulated using *ref* inputs.

4 Discussion and Results

This section examines the effectiveness of single-ISA heterogeneous multi-core designs in reducing the power dissipation of processors. Section 4.1 examines the relative energy efficiency across cores, and how it varies by application and phase. Later sections use this variance, demonstrating both oracle and realistic core switching heuristics to maximize particular objective functions.

4.1 Variation in Core Performance and Power

As discussed in Section 2, this work assumes that the performance ratios between our processor cores is not constant, but varies across benchmarks, as well as over time on a single benchmark. This section verifies that premise.

IEEE COMPUTER SOCIETY

Petitioner Samsung Ex-1032, 0006

Figure 2(a) shows the performance measured in million instructions committed per second (IPS) of one representative benchmark, *applu*. In the figure, a separate curve is shown for each of the five cores, with each data point representing the IPS over the preceding 1 million committed instructions.

With *applu*, there are very clear and distinct phases of performance on each core, and the relative performance of the cores varies significantly between these phases. Nearly all programs show clear phased behavior, although the frequency and variety of phases varies significantly.

If relative performance of the cores varies over time, it follows that energy efficiency will also vary. Figure 3 shows one metric of energy efficiency (defined in this case as $IPS^2/Watt$) of the various cores for the same benchmark. $IPS^2/Watt$ is merely the inverse of Energy-Delay product. As can be seen, the relative value of the energy-delay product among cores, and even the ordering of the cores, varies from phase to phase.

4.2 Oracle Heuristics for Dynamic Core Selection

This section examines the limits of power and efficiency improvements possible with a heterogeneous multi-core architecture. The ideal core-selection algorithm depends heavily on the particular goals of the architecture or application. This section demonstrates oracle algorithms that maximize two sample objective functions. The first optimizes for energy efficiency with a tight performance threshold. The second optimizes for energy-delay product with a looser performance constraint.

These algorithms assume perfect knowledge of the performance and power characteristics at the granularity of intervals of one million instructions (corresponding roughly to an OS time-slice interval). It should be noted that choosing the core that minimizes energy or the energy-delay product over each interval subject to performance constraints does not give an optimal solution for the global energy or energy-delay product; however, the algorithms do produce good results.

4.2.1 Oracle based on energy metric

The first oracle that we study seeks to minimize the energy per committed instruction (and thus, the energy used by the entire program). For each interval, the oracle chooses the core that has the lowest energy consumption, given the constraint that performance has always to be maintained within 10% of the EV8- core for each interval. This constraint assumes that we are willing to give up performance to save energy but only up to a point. Figure 2(b) shows the core selected in each interval for *applu*.

For *applu*, we observe that the oracle chooses to switch to EV6 in several phases even though EV8- performs bet-

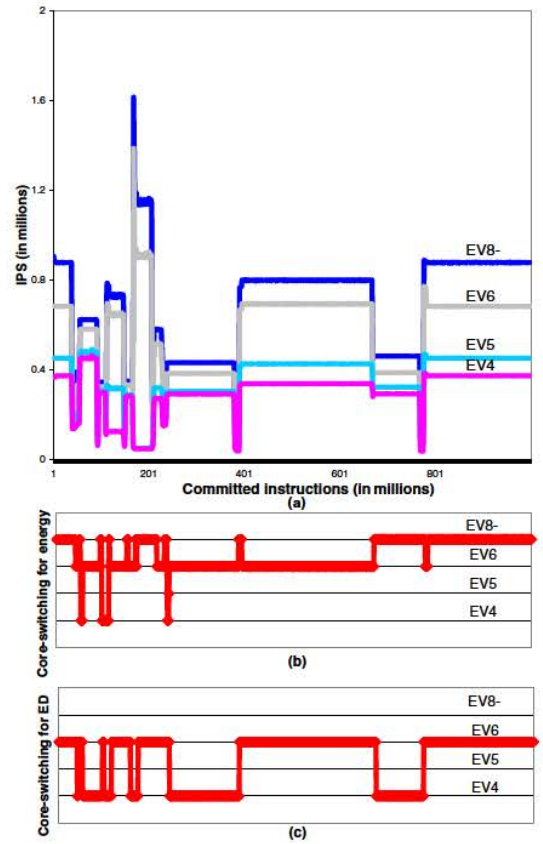

Figure 2. (a) Performance of *applu* on the four cores (b) Oracle switching for energy (c) Oracle switching for energy-delay product.

ter. This is because EV6 is the less power-consuming core and still performs within the threshold. The oracle even switches to EV4 and EV5 in a small number of phases. Table 4 shows the results for all benchmarks. In this, and all following results, performance degradation and energy savings are always given relative to EV8- performance. As can be seen, this heuristic achieves an average energy reduction of 38% (see column 8) with less than 4% average performance degradation (column 9). Five benchmarks (*ammp, fma3d, mcf, twolf, crafty*) achieve no gain because switching was denied by the performance constraint. Excluding these benchmarks, the heuristic achieves an average energy reduction of 60% with about 5% performance degradation.

4.2.2 Oracle based on the energy-delay metric.

Our second oracle utilizes the *energy-delay product* metric. The energy-delay product seeks to characterize the impor-

IEEE COMPUTER SOCIETY

Petitioner Samsung Ex-1032, 0007

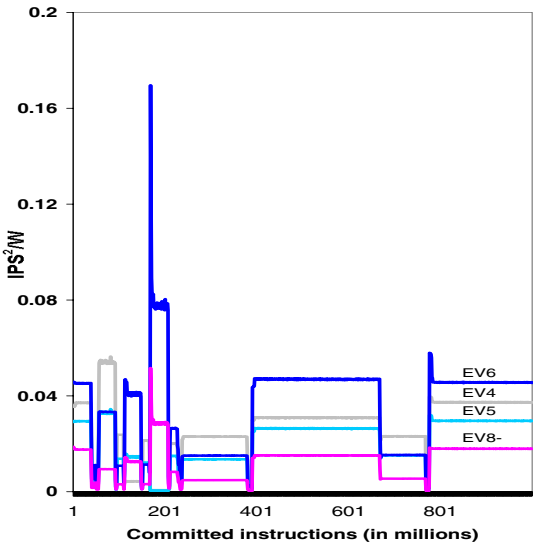

Figure 3. *applu* **energy efficiency.** IPS^2/W **varies inversely with energy-delay product**

tance of both energy and response time in a single metric, under the assumption that they have equal importance. Our oracle minimizes energy-delay product by always selecting the core that maximizes $IPS^2/Watt$ over an interval. We again impose a performance threshold, but relax it due to the fact that energy-delay product already accounts for performance degradation. In this case, we require that each interval maintains performance within 50% of EV8-.

Figure 2(c) shows the cores chosen for *applu*. Table 5 shows the results for all benchmarks. As can be seen, the average reduction in energy-delay is about 63%; the average energy reductions are 73% and the average performance degradation is 22%. All but one of the fourteen benchmarks have fairly significant (47% to 78%) reductions in energy-delay savings. The corresponding reductions in performance ranges from 4% to 45%. As before, switching activity and the usage of the cores varies. This time, EV8 never gets used. EV6 emerges as the dominant core. Given our relaxed performance constraint, there is a greater usage of the lower-power cores compared to the previous experiment.

4.2.3 Comparison with chip-wide voltage/frequency scaling

Both Tables 4 and 5 also show results for Energy-Delay^2 [40] improvements. Improvements are 35-50% on average. This is instructive because chip-wide voltage/frequency scaling can do no better than break even on this metric, demonstrating that this approach has the potential to go well beyond the capabilities of that technique. In other experiments specifically targeting the ED^2 metric (again with the 50% performance threshold), we saw 53.3% reduction in energy-delay^2 with 14.8% degradation in performance.

4.3 Static Core Selection

This section examines the necessity of dynamic switching between cores by measuring the effectiveness of an oracle-based static assignment of benchmark to core (for just one of our sample objective functions). This models a system that accurately selects a single core to run for the duration of execution, perhaps based on compiler analysis, profiling, past history, or simple sampling.

Table 6 summarizes the results when a static oracle selects the best core for energy. As in the earlier dynamic results, a performance threshold (this time over the duration of the benchmark) is applied. EV6 is the only core other than EV8 which gets used. This is because of the stringent performance constraint. Average energy savings is 32%. Excluding the benchmarks which remain on EV8, average energy savings is 74.3%. Average performance degradation is 2.6%. This low performance loss leads to particularly high savings for both energy-delay and energy-delay^2. Average energy savings is 31% and average energy-delay^2 savings is 30%.

Also shown is a corresponding dynamic technique. For a fair comparison, we apply a global runtime performance constraint rather than a per-interval constraint. That is, any core can be chosen in an interval as long as the accumulated runtime up to this point (including all core choices made on earlier intervals) remains within 10% of the EV8- alone. This gives a more fair comparison with the static technique.

4.4 Realistic Dynamic Switching Heuristics

This section examines the extent to which the energy benefits in the earlier sections can be achieved with a real system implementation that does not depend on oracular future knowledge. We do, however, assume an ability to track both the accumulated performance and energy over a past interval. This functionality either already exists or is easy to implement. This section is intended to be an existence proof of effective core selection algorithms, rather than a complete evaluation of the switching design space. We only demonstrate a few simple heuristics for selecting the core to run on. The heuristics seek to minimize overall energy-delay product during program execution.

Our previous oracle results were idealized not only with respect to switching algorithms, but also ignored the cost of switching (power-up time, flushing dirty pages to the L2 cache and experiencing cold-start misses in the new L1 cache and TLB) both in performance and power. The

IEEE COMPUTER SOCIETY

Petitioner Samsung Ex-1032, 0008

Benchmark	Total switches	% of instructions per core				Energy Savings(%)	ED Savings(%)	ED^2 Savings(%)	Perf. Loss (%)
		EV4	EV5	EV6	EV8-				
ammp	0	0	0	0	100	0	0	0	0
applu	27	2.2	0.1	54.5	43.2	42.7	38.6	33.6	7.1
apsi	2	0	0	62.2	37.8	27.6	25.3	22.9	3.1
art	0	0	0	100	0	74.4	73.5	72.6	3.3
equake	20	0	0	97.9	2.1	72.4	71.3	70.1	3.9
fma3d	0	0	0	0	100	0	0	0	0
wupwise	16	0	0	99	1	72.6	69.9	66.2	10.0
bzip	13	0	0.1	84.0	15.9	40.1	38.7	37.2	2.3
crafty	0	0	0	0	100	0	0	0	0
eon	0	0	0	100	0	77.3	76.3	75.3	4.2
gzip	82	0	0	95.9	4.1	74.0	73.0	71.8	3.9
mcf	0	0	0	0	100	0	0	0	0
twolf	0	0	0	0	100	0	0	0	0
vortex	364	0	0	73.8	26.2	56.2	51.9	46.2	9.8
Average	1(median)	0.2%	0%	54.8%	45.0%	38.5%	37.0%	35.4%	3.4%

Table 4. Summary for dynamic *oracle* switching for energy

Benchmark	Total switches	% of instructions per core				Energy-delay Savings(%)	Energy Savings(%)	Energy-delay^2 Savings(%)	Perf. Loss (%)
		EV4	EV5	EV6	EV8-				
ammp	0	0	0	100	0	63.7	70.3	55.7	18.1
applu	12	32.3	0	67.7	0	69.8	77.1	59.9	24.4
apsi	0	0	0	100	0	60.1	69.1	48.7	22.4
art	619	65.4	0	34.5	0	78.0	84.0	69.6	27.4
equake	73	55.8	0	44.2	0	72.3	81.0	59.2	31.7
fma3d	0	0	0	100	0	63.2	73.6	48.9	28.1
wupwise	0	0	0	100	0	68.8	73.2	66.9	10.0
bzip	18	0	1.2	98.8	0	60.5	70.3	47.5	24.8
crafty	0	0	0	100	0	55.4	69.9	33.9	32.5
eon	0	0	0	100	0	76.2	77.3	75.3	4.2
gzip	0	0	0	100	0	74.6	75.7	73.5	4.2
mcf	0	0	0	100	0	46.9	62.8	37.2	24.3
twolf	0	0	0	100	0	26.4	59.7	-34.2	45.2
vortex	0	0	0	100	0	68.7	73.0	66.7	9.9
Average	0(median)	11.0%	0.1%	88.9%	0%	63.2%	72.6%	50.6%	22.0%

Table 5. Summary for dynamic *oracle* switching for energy-delay

Benchmark	Core	Static Selection				Dynamic Selection	
		Energy savings (%)	Energy-delay savings (%)	Energy-delay^2 savings (%)	Perf loss (%)	Energy savings(%)	Perf. loss (%)
ammp	EV8-	None	None	None	None	36.1	10.0
applu	EV8-	None	None	None	None	49.9	10.0
apsi	EV8-	None	None	None	None	42.9	10.0
art	EV6	74.4	73.5	72.6	3.3	75.7	10.0
equake	EV6	73.4	72.3	70.8	4.5	74.4	10.0
fma3d	EV8-	None	None	None	None	28.1	10.0
wupwise	EV6	73.2	70.5	66.9	10.0	49.5	10.0
bzip	EV8-	None	None	None	None	47.7	10.0
crafty	EV8-	None	None	None	None	17.6	10.0
eon	EV6	77.3	76.3	75.3	4.2	77.3	9.8
gzip	EV6	75.7	74.6	73.5	4.3	76.0	10.0
mcf	EV8-	None	None	None	None	19.9	10.0
twolf	EV8-	None	None	None	None	8.1	10.0
vortex	EV6	73.0	70.3	66.7	9.9	52.0	10.0
Average	-	31.9%	31.3%	30.4%	2.6%	46.8%	10.0

Table 6. Oracle heuristic for static core selection – energy metric. Rightmost two columns are for dynamic selection

IEEE COMPUTER SOCIETY

simulations in this section account for both, although our switching intervals are long enough and switchings infrequent enough that the impact of both effects is under 1%.

In this section, we measure the effectiveness of several heuristics for selecting a core. The common elements of each of the heuristics are these: every 100 time intervals (one time interval consists of 1 million instructions in these experiments), one or more cores are sampled for five intervals each (with the results during the first interval ignored to avoid cold start effects). Based on measurements done during sampling, the heuristic selects one core. For the case when one other core is sampled, the switching overhead is incurred once if the new core is selected, or twice if the old core is chosen. The switching overhead is greater if more cores are sampled. The dynamic heuristics studied here are:

- **neighbor**. One of the two neighboring cores in the performance continuum is randomly selected for sampling. A switch is made if that core has lower energy-delay over the sample interval than the current core over the last run interval.
- **neighbor-global**. Similar to neighbor, except that the selected core is the one that would be expected to produce the lowest accumulated energy-delay product to this point in the application's execution. In some cases this is different than the core that minimizes the energy-delay product for this interval.
- **random**. One other randomly-chosen core is sampled, and a switch is made if that core has lower energy-delay over the sample interval.
- **all** All other cores are sampled.

The results are shown in Figure 4. The results are all normalized to EV8- values. This figure also includes oracle results for dynamic switching based on the energy-delay metric when core selection is not hampered with performance constraints. Lower bars for energy and energy-delay, and higher bars for performance are desirable.

Our heuristics achieve up to 93% of the energy-delay gains achieved by the oracle-based switcher, despite modeling the switching overhead, sampling overhead, and non-oracle selection. The performance degradation on applying our dynamic heuristics is, on average, *less* than the degradation found by the oracle-based scheme. Also, although not shown in the figure, there is a greater variety in core-usage between applications.

It should be noted that switching for this particular objective function is not heavy; thus, heuristics that find the best core quickly, and minimize sampling overhead after that, tend to work best. The best heuristic for a different objective function, or a dynamically varying objective function may be different. These results do show, however, that for a given objective function, very effective realtime and predictive core switching heuristics can be found.

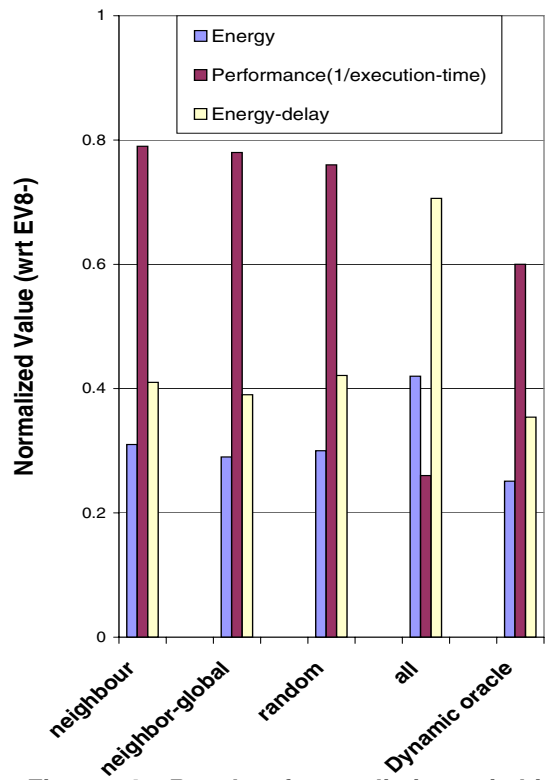

Figure 4. Results for realistic switching heuristics - the last one is a constraint-less dynamic oracle

4.5 Practical heterogeneous architectures

Although our use of existing cores limits design and verification overheads, these overheads do scale with the number of distinct cores supported.

Some or our results indicate that in specific instances, two cores can introduce sufficient heterogeneity to produce significant gains. For example the (minimize energy, maintain performance within 10%) objective function relied heavily on the EV8- and the EV6 cores. The (energy-delay, performance within 50%) objective function favored the EV6 and EV4. However, if the objective function is allowed to vary over time, or if the workload is more diverse than what we model, wider heterogeneity than 2 cores will be useful. Presumably, other objective functions than those we model may also use more than 2 cores.

IEEE COMPUTER SOCIETY

Petitioner Samsung Ex-1032, 0010

5 Related Work

There has been a large body of work on power-related optimizations for processor design. These can be broadly classified into two categories: (1) work that uses gating for power management, and (2) work that uses voltage and frequency scaling of the processor core to reduce power.

Gating-based power optimizations [6, 13, 14, 17, 20, 28, 29] provide the option to turn off (gate) portions of the processor core that are not useful to a workload. However, for all these techniques, gating benefits are limited by the granularity of structures that can be gated, the inability to change the overall size and complexity of the processor. Also, these designs are still susceptible to static leakage inefficiencies.

Chip-wide voltage and frequency scaling reduces the parameters of the entire core [16, 32]. While this reduces power, the power reductions are uniform – across both the portions of the core that are performance-critical for this workload as well as the portions of the core that are not. Furthermore, voltage and frequency scaling is fundamentally limited by the process technology in which the processor is built. Heterogeneous multi-core designs address both these deficiencies.

Fine-grained voltage/frequency scaling techniques using multiple clock domains have been proposed recently [21, 34] which obviate some of the disadvantages of conventional scaling-based techniques discussed elsewhere in this paper. However, similar to gating-based approaches, the benefits are likely to be limited by static leakage inefficiencies as well as the number of voltage domains that can be supported on a chip.

Core switching to reduce power was introduced previously in [26] and [27]. Recently, it has also been used for reducing power density [19] in a homogeneous multiple-core architecture, through the use of frequent core switches to idle processors.

Overall, having heterogeneous processor cores provides potentially greater power savings compared to previous approaches and greater flexibility and scalability of architecture design. Moreover, these previous approaches can still be used in a multi-core processor to greater advantage.

6 Summary and Conclusions

This paper introduces and seeks to gain some insights into the energy benefits available for a new architecture, that of a heterogeneous set of cores on a single multi-core die, sharing the same ISA. The particular opportunity examined is a single application switching among cores to optimize some function of energy and performance.

We show that a sample heterogeneous multi-core design with four complexity-graded cores has the potential to increase energy efficiency (defined as energy-delay product, in this case) by a factor of three, in one experiment, without dramatic losses in performance. Energy efficiency improvements significantly outdistance chip-wide voltage/frequency scaling. It is shown that most of these gains are possible even by using as few as two cores.

This work demonstrates that there can be great advantage to diversity within an on-chip multiprocessor, allowing that architecture to adapt to the workload in ways that a uniform CMP cannot. A multi-core heterogeneous architecture can support a range of execution characteristics not possible in an adaptable single-core processor, even one that employs aggressive gating. Such an architecture can adapt not only to changing demands in a single application, but also to changing demands between applications, changing priorities or objective functions within a processor or between applications, or even changing operating environments.

These results indicate that not only is there significant potential for this style of architecture, but that reasonable runtime heuristics for switching cores, using limited runtime information, can achieve most of that potential.

Acknowledgments

The authors would like to thank the anonymous reviewers for their useful comments. This work was supported in part by NSF Grant No. CCR-0105743, an HP internship, and a grant from the Semiconductor Research Corporation (contract 2001-HJ-897).

References

[1] International Technology Roadmap for Semiconductors 2002, http://public.itrs.net.

[2] Alpha 21064 and Alpha 21064A: Hardware Reference Manual. Digital Equipment Corporation, 1992.

[3] Alpha 21164 Microprocessor:Hardware Reference Manual. Digital Equipment Corporation, 1998.

[4] Alpha 21264/EV6 Microprocessor:Hardware Reference Manual. Compaq Corporation, 1998.

[5] Intel Pentium 4 Processor in the 423-pin Package Thermal Design Guidelines. Nov. 2000.

[6] D. H. Albonesi. Selective cache-ways: On demand cache resource allocation. In *International Symposium on Microarchitecture*, Nov. 1999.

[7] W. Bowhill *et al*. A 300-MHz 64-b quad-issue CMOS microprocessor. In *ISSCC Digest of Technical Papers*, Feb. 1995.

[8] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *International Symposium on Computer Architecture*, June 2000.

[9] J. Burns and J.-L. Gaudiot. SMT layout overhead and scalability. In *IEEE Transactions on Parallel and Distributed Systems, Vol 13, No. 2*, Feb. 2002.

IEEE COMPUTER SOCIETY

Petitioner Samsung Ex-1032, 0011

[10] K. Diefendorff. Compaq chooses SMT for Alpha. In *Microprocessor Report, Vol 13, No. 16*, Dec. 1999.

[11] D. W. Dobberpuhl *et al*. A 200-MHz 64-b dual-issue CMOS Microprocessor. In *IEEE Journal of Solid-State Circuits, Vol 27, No. 11*, Nov. 1992.

[12] J. Emer. EV8:The Post-ultimate Alpha. In *PACT Keynote Address*, Sept. 2001.

[13] D. Folegnani and A. Gonzalez. Reducing power consumption of the issue logic. In *Workshop on Complexity-Effective Design*, June 2000.

[14] S. Ghiasi, J. Casmira, and D. Grunwald. Using IPC variation in workloads with externally specified rates to reduce power consumption. In *Workshop on Complexity Effective Design.*, June 2000.

[15] B. Gieseke *et al*. A 600-MHz Superscalar RISC Microprocessor with Out-of-Order Execution. In *ISSCC Digest of Technical Papers*, Feb. 1997.

[16] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power cpu. In *International Conference on Mobile Computing and Networking*, Nov. 1995.

[17] D. Grunwald, A. Klauser, S. Manne, and A. Pleskun. Confidence estimation for speculation control. In *International Symposium on Computer Architecture*, June 1998.

[18] S. H. Gunther, F. Binns, D. Carmean, and J. C. Hall. Managing the Impact of Increasing Microprocessor Power Consumption. In *Intel Technology Journal*, 1st Quarter 2001.

[19] S. Heo, K. Barr, and K. Asanovic. Reducing power density through activity migration. In *International Symposium on Low Power Electronics and Design*, Aug. 2003.

[20] A. Iyer and D. Marculescu. Power aware microarchitecture resource scaling. In *IEEE Design, Automation and Test in Europe Conference*, 2001.

[21] A. Iyer and D. Marculescu. Power-performance exaluation of globally-asynchronous, locally-synchronous processors. In *International Symposium on Computer Architecture*, 2001.

[22] R. Joseph and M. Martonosi. Run-time Power Estimation in High-Performance Microprocessors. In *The International Symposium on Low-Power Estimation and Design*, Aug. 2001.

[23] A. Klauser. Trends in high-performance microprocessor design. In *Telematik-2001*, 2001.

[24] J. Kowaleski *et al*. Implementation of an Alpha Microprocessor in SOI. In *ISSCC Digest of Tecnical Papers*, Feb. 2003.

[25] A. Kumar. The HP PA-8000 RISC CPU. In *Hot Chips VIII*, Aug. 1996.

[26] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. A multi-core approach to addressing the energy-complexity problem in microprocessors. In *Workshop on Complexity-Effective Design*, June 2003.

[27] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Processor power reduction via single-ISA heterogeneous multi-core architectures. In *Computer Architecture Letters, Vol 2*, Apr. 2003.

[28] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: Speculation control for energy reduction. In *International Symposium on Computer Architecture*, June 1998.

[29] R. Maro, Y. Bai, and R. Bahar. Dynamically reconfiguring processor resources to reduce power consumption in high-performance processors. In *Workshop on Power-aware Computer Systems*, Nov. 2000.

[30] R. Merritt. Designers cut fresh paths to parallelism. In *EE Times.*, Oct. 1999.

[31] J. M. Mulder, N. T. Quach, and M. J. Flynn. An area model for on-chip memories and its applications. In *IEEE Journal of Solid State Circuits, Vol 26, No. 2*, Feb. 1991.

[32] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *International Symposium on Low Power Electronics and Design*, Aug. 1998.

[33] J. M. Rabaey. The quest for ultra-low energy computation opportunities for architectures exploiting low-current devices. Apr. 2000.

[34] G. Semeraro, G. Maklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *International Symposium on High-Performance Computer Architecture*, Feb. 2002.

[35] T. Sherwood and B. Calder. Time varying behavior of programs. In *UC San Diego Technical Report UCSD-CS-99-630*, Aug. 1999.

[36] T. Sherwood, E. Perelman, G. Hammerley, and B. Calder. Automatically characterizing large-scale program behavior. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[37] P. Shivakumar and N. Jouppi. CACTI 3.0: An integrated cache timing, power and area model. In *Technical Report 2001/2, Compaq Computer Corporation*, Aug. 2001.

[38] D. Tullsen. Simulation and modeling of a simultaneous multithreading processor. In *22nd Annual Computer Measurement Group Conference*, Dec. 1996.

[39] D. Wall. Limits of instruction-level parallelism. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr. 1991.

[40] V. Zyuban. Unified architecture level energy-efficiency metric. In *2002 Great Lakes Symposium on VLSI*, Apr. 2002.

IEEE COMPUTER SOCIETY

Petitioner Samsung Ex-1032, 0012