Low-Latency Adaptive Mode Transitions and Hierarchical Power Management in Asymmetric Clustered Cores

ERAN SHIFER, Tel Aviv University and Intel Corporation

SHLOMO WEISS, Tel Aviv University

Recently, engineering solutions that include asymmetric multicores have been fabricated for low form-factor computing devices, indicating a potential direction for future evolution of processors. In this article we propose an asymmetric clustered core architecture, exhibiting low-latency switching between modes relative to asymmetric multicores, and having similarities with the same asymmetric multicore architecture in the context of a wider dynamic range of the processor power-performance characteristic. Asymmetric clustered cores incur additional microarchitectural complexity and area cost inside a core but exhibit better chip-level integration characteristics compared to asymmetric multicores. Focusing on power efficiency of asymmetric clustered cores, we describe: (1) a hierarchical power management partitioning between the operating system and on-die firmware for coarse-grain switch policies, and (2) core-internal tracking hardware for fine-grain switching. The mode switch policies of the core's tracking hardware are dependent on higher-level directives and hints from the operating system, on-die firmware, and compiler or profiling software. We further explore the potential power management benefits of asymmetric clustered cores relative to asymmetric multicores, demonstrating that the ability of asymmetric clustered cores to use tight training periods for adaptive behavior, with low overhead switching between modes, results in a more efficient utilization of power management directives.

Categories and Subject Descriptors: C.2.2 [Computer Architecture]: Power Management

General Terms: Design, Performance

Additional Key Words and Phrases: Architecture, power, asymmetric multicores, clustered cores, heterogeneous systems, core morphing, energy delay

ACM Reference Format:

Shifer, E. and Weiss, S. 2013. Low-latency adaptive mode-transitions and hierarchical power management in asymmetric clustered cores. *ACM Trans. Architect. Code Optim.* 10, 3, Article 10 (September 2013), 25 pages.

DOI: <http://dx.doi.org/10.1145/2499901>

1. INTRODUCTION

Physical properties of fabricated transistors dictate changes in computer architecture. Silicon scaling rules applied in the past [Dennard et al. 1974] cannot be sustained for low operating voltages, due to circuit sensitivity to manufacturing variations and operating point changes [Gonzalez et al. 1997]. As a result, various engineering solutions are required to accommodate the increase in transistor density within a highly constrained power budget [Borkar 1999]. These solutions need to take into account the different usage models of a general-purpose processor, optimizing for latency and

Authors' addresses: E. Shifer and S. Weiss (corresponding author), School of Electrical Engineering, Tel-Aviv University, Tel Aviv 69978, Israel; email: weiss@eng.tau.ac.il.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2013 ACM 1544-3566/2013/09-ART10 \$15.00

DOI: <http://dx.doi.org/10.1145/2499901>

throughput across and within applications [Grochowski et al. 2004], without exceeding the allocated power budget.

Current solutions to the silicon scaling challenge have limitations. SMT and CMP [Eggers et al. 1997; Burns and Gaudiot 2001, 2002] provide thread scalability within the computation core and processor, respectively, and respond to dynamic computation demand by running additional threads on the CPU, at a significant silicon area and power cost. Dynamic Voltage and Frequency Scaling (DVFS) [Chandrakasan et al. 1992; Brooks and Martonosi 2001; Rotem et al. 2012] adjusts the processor's performance to thermal dissipation and energy constraints, but has limited value when the processor already operates at low voltage limits, and incurs a delay until the clock is restabilized and the voltage swing completes.

Recent developments in processors targeting mobile hand-held markets indicate potential transition to asymmetric multicore architecture [ARM Cortex 2011; Nvidia Tegra 2013; Marvell Tri-Core 2013; TI OMAP 5 2011]. While a promising technique for optimizing battery-life and supplying user-level responsiveness for low form-factor devices, asymmetric multicore architecture has several limitations that may hinder its applicability for more demanding usage models. Besides the basic difficulty of identifying the desired thread migrations between different core types at a given point in time, if multiple threads compete for the same type of core the migration needs to be managed by firmware or the Operating System (OS), increasing the time constant of the migration. In addition, context migration between cores incurs a delay and power penalty that increases with the number of swapped threads and can degrade the migration benefit.

In this work we present a proposal for an Asymmetric Clustered Core (ACC), constructed of tightly coupled execution pipelines that share some parts of the core and exhibit fundamentally distinct optimization points. By supporting multiple modes of operation, an ACC offers dynamic selection among multiple points on a wide-range power-performance curve. ACCs conceptually resemble asymmetric multicores. There is, however, a fundamental difference. Asymmetric multicores [Balakrishnan et al. 2005] offer "big" cores for high performance and "small" cores for reduced power dissipation. On the other hand, in our ACC proposal all cores are identical and the asymmetry is within the core itself. While asymmetric multicores expose both "big" and "small" cores to the operating system, in ACCs only a single type of execution pipeline is exposed, enabling hardware-managed, fast mode transitions.

An ACC is complementary to DVFS and enables hierarchical partitioning of power management between a global power control unit and a local core. Hierarchical power management assigns the DVFS algorithm to the operating system and to the central power management control unit. Shorter time scale adaptation decisions are done locally at the ACC based on the assigned power budget.

Compared to asymmetric multicores, the ACC provides low-overhead mode switching at the expense of additional core complexity. Switching pipelines in an ACC requires a pipeline flush and memory fence. Data migration and cache warm-up are not needed because internal caches are shared. Thread scalability is achieved within the cores by adding multiple throughput-oriented and energy-efficient execution pipelines within the cores. Processor-level integration of ACCs has several benefits compared to asymmetric multicores, due to the similarity of the cores in the former.

1.1. Contributions

This article makes the following contributions.

- (1) We propose a novel ACC architecture that enables dynamic switching, requiring very low latency from one mode of operation to another. Multiple modes of operation are supported exhibiting a wide range of power-performance trade-offs.

- (2) We discuss core-level and chip-level functional and layout integration considerations for ACC and asymmetric multicore architectures, presenting area comparisons between the two options based on a specific chip-level integration scheme.
- (3) We describe a hierarchical power management control-partitioning scheme: (a) the operating system and a central power management control unit on the global level, and (b) an ACC on the local level. The latter performs fine-grain power mode transitions according to given directives and hints.
- (4) We propose a fine-grain, adaptive switch heuristic for single-thread workload which is based on training periods and real-time information.
- (5) We provide a single-thread power-performance analysis of the ACC for the proposed switch heuristic scheme.
- (6) We demonstrate the relative power management efficiency of ACCs and asymmetric multicores.

1.2. Overview

The rest of the article is comprised of the following sections. Section 2 discusses the ACC microarchitecture. Section 3 describes the methodology used in this article to study the scalar performance characteristics of the ACC. Section 4 describes mode switch considerations and mode switch heuristics. Section 5 presents the ACC evaluation results. Section 6 presents a comparison between ACC and asymmetric multicores. Section 7 discusses related work. Section 8 summarizes the work and discusses future research.

2. ASYMMETRIC CLUSTERED CORE

In this section we describe the microarchitecture of the proposed asymmetric clustered core and discuss thread scalability properties and trade-offs related to design complexity and optimization challenges.

2.1. ACC Microarchitecture

An ACC consists of several functional clusters that may be combined to operate together and form a functional CPU core. Different cluster combinations effectively create different CPU core types, each with its own characteristic power-performance curves. The asymmetric nature of the core is achieved by partitioning its back-end flow control to two different clusters—one implementing a complex flow-control pipeline, and the other implementing a simpler flow-control pipeline. Figure 1 describes the general structure of an ACC. Both Pipe A and Pipe B are out-of-order pipelines that are substantially different from one another in some of the pipeline fundamental characteristics. Specifically, the instruction issue, retirement widths, and the depth of the reorder-buffer are substantially different in the two pipelines. Another difference is that Pipe B is single-threaded while Pipe A is multithreaded. In Figure 1 the context cache holds architectural state copies of gated execution pipelines, the partial ALU includes mostly integer and address calculation units, the vector ALU holds wide SIMD execution units, and the system interface connects the core to the on-die interconnect fabric and external cache hierarchies.

The ACC's potential benefit over a core that is dedicated to either Pipe A or Pipe B is dependent on the workload and the desired system optimization points. The sharing of the memory subsystem by Pipe A and Pipe B may lead to degradation in the load-to-use latency of memory operands and hurt the performance of both pipelines. This may be minimized by allowing only one of the pipeline clusters to be functional at a given time, at the cost of hurting thread scalability properties. Furthermore, adding a second pipeline cluster to an already existing core baseline incurs area and power penalty. While the power penalty may be partially mitigated by power gating the

Fig. 1. Asymmetric clustered core.

unused pipeline, the area penalty should be compared against other alternatives. Such a potential alternative for comparison is the integration of asymmetric multicores around a chip-level interconnect subsystem. Section 6.2 presents chip-level area consequences of a specific multicore integration scheme.

There are many partition and optimization options of the different core clusters. One optimization example is the potential clock gating of parts of the underutilized front-end and ALU blocks when the core operates in the simpler Pipe A mode. Sleep transistors [Tschanz et al. 2003] may be used to dynamically reduce leakage power of gated blocks with fast time constants of entering and exiting idle modes. Another possible partition option would share the cache hierarchy at the L2 cache, better optimizing for load latency, at the cost of area and migration time overheads. The analysis given in the subsequent sections assumes Pipe A and Pipe B core parameters according to Table I. The core parameters and cache hierarchy definition are based on Intel internal simulation tools and sensitivity analysis to accommodate core reuse across segments with desired targets of power-performance, cost, and scalability.

2.2. Thread Scalability

The streamlined Pipe A of the ACC (refer to Figure 1) may be exploited for parallel compute scalability. While a complicated out-of-order core incurs high area and power cost for every physical thread added in SMT constellation, a simple in-order pipeline or shallow out-of-order pipeline can scale to a higher level of parallel threads in the SMT constellation with less area and power overhead. The number of threads visible to the OS is the SMT thread count of Pipe A. Any of these threads may be moved from Pipe A to the single-threaded Pipe B without OS involvement. If, for example,

Table I. ACC Example - Distinct Pipe A and Pipe B

Core	Pipe A	Pipe B
Core Type	out-of-order	out-of-order
Issue Width	3-way	4-way
ROB	16 entries	128 entries
LD Buffer	16 entries	48 entries
ST Buffer	12 entries	32 entries
L1 I-Cache	32KB; 4-way	32KB; 4-way
L1 D-Cache	32KB; 4-way	32KB; 4-way
L2 Cache	256KB; 16-way	256KB; 16-way
L3 Cache	1MB; 32-way	1MB; 32-way
Memory Load Latency	70ns	70ns

Table II. Core Width and Thread Count Scaling Factors for Pipe A and pipe B Shown in Table I

Group	Width scaling factor	Thread scaling factor	Relative area	Hardware blocks
1	$O(d^2)$	$O(t)$	4%	Renamed Register Files
2	$O(d^2)$	$O(\log_2 t)$	1%	Branch Predictor
3	$O(d^2)$	$O(1)$	18%	TLBs, Instruction Queue, Load Store Queue, Int/FP ALUs, Reorder logic
4	$O(d)$	$O(t \log_2 t)$	2%	Remap Table
5	$O(d)$	$O(t)$	1%	Fetch Logic
6	$O(d)$	$O(\log_2 t)$	8%	L1 Instruction cache, decode logic
7	$O(d)$	$O(1)$	52%	L1 Data cache, routing overhead
8	$O(1)$	$O(1)$	14%	Miscellaneous logic

‘d’ is core width. ‘t’ is thread count.

Pipe A is a fully populated 4-way SMT, moving one thread from Pipe A to Pipe B would leave three threads on Pipe A running concurrently with one thread on Pipe B. Concurrent operation of the complex Pipe B and simple Pipe A supports thread migration by hardware without requiring OS involvement.

Burns and Gaudiot [2002] discuss the diminishing returns and technical challenges of adding multiple threads to a wide out-of-order core. The impact of adding an SMT thread is spread across most of the functional blocks of the core, as additional physical threads supported within a core require increase in the width or depth of the main flow control and execution units, as well as size and throughput increases in the local caches and TLBs. The additional hardware stresses several architectural timing paths within the core, resulting in transistor size increases, additional pipeline stages, and additional power consumption within the core.

To maintain a balanced microarchitecture implementation, different parts of a core need to scale in a specific manner to accommodate changes in machine width, depth, and thread count. For example, the core commit logic size is proportional to the square of the core width, to handle instruction retirement dependencies. The same commit logic is proportional in size to the core thread count, to handle per-thread retirement windows. Typical area scaling factors based on silicon studies and grouped according to machine width and thread count sensitivity are given in Table II. These scaling factors may be utilized for illustrating the general scalability characteristics of a generic out-of-order core.

Figure 2 illustrates core width and thread count scalability based on Table II and ACC block sharing between execution pipelines. This figure demonstrates the area growth

Fig. 2. Core width and thread count scalability. In the three superscalar versions, each SMT thread runs on the specified superscalar width. ACC consists of a single-threaded 4-way superscalar Pipe B and a multithreaded 3-way superscalar Pipe A.

Fig. 3. ACC layout partition.

difference between narrow and wide cores as a function of the thread count, and the area scaling of the ACC. The ACC architecture has better area scaling properties relative to a wide SMT. This aspect can be exploited to provide throughput scalability through the simple pipeline while maintaining high scalar performance through the complex pipeline. In this regard, the tightly coupled nature of the ACC opens opportunities for fine-grain critical section acceleration in multithreaded environments.

2.3. ACC Core Design and Trade-Offs

Figure 3 describes a schematic partition of an ACC with a single complex execution pipeline and two simple execution pipelines. For illustration purposes the simple execution pipelines are drawn as distinct pipelines and not as a merged simple SMT pipeline. The simpler Pipe A clusters are positioned closer to the front-end and memory subsystems, because their performance is more sensitive to additional pipeline stages. The positioning of interface blocks and additional arbitration stages were taken into account for timing estimates without accounting for a full integration overhead, which would be based on chip netlist data. The complex Pipe B cluster incurs one-cycle load-latency degradation relative to Pipe A load latency. Cluster architectural allocation and scheduler timing paths are unchanged by the proposed partition, due to the inclusion of Ld/St buffers within the pipeline clusters, and to the proximity of the pipeline clusters to the front-end cluster dispatcher and Vector ALU write-back bus.

Commercial high-performance cores are typically a result of several generations of consecutive improvements. These cores are characterized by a high level of design

complexity and are carefully optimized for power and performance. The ACC architecture adds complexity by introducing the simple pipeline blocks and connections. While it is not our intention to cover in this article the specific challenges relevant to the design and validation of an ACC, we will describe the main functional and power-performance challenges associated with such a design.

The cluster boundaries of the complex and simple pipelines, as demonstrated in Figure 1, align with the boundaries of a typical core functional block. The level of behavioral similarity between the distinct pipelines defines the complexity scope of handling two types of internal pipelines at the front-end block boundary. If the simple and complex pipelines have the same out-of-order flow control implementation, with different instruction issue widths and trackers sizes, their interface signals and protocol with the front-end block will have a high level of similarity. If the simple pipeline is an in-order pipeline and the complex pipeline is an out-of-order pipeline, a design implementation may opt to implement the in-order interface signals and protocol as a special case of the out-of-order pipeline. Another point to consider is the similarity level between the simple and complex pipeline micro-ops. For example, the simple pipeline may support a datapath that is half the width of the complex pipeline, requiring the microcode to double-pump operations for wide arithmetic operations. The shared Vector-ALU in Figure 1 limits the potential divergence between complex and simple pipeline decoders and microcode.

The interface signals and protocol with the memory subsystem is another challenging point for the ACC. Since the load and store buffers are encapsulated within the respective pipeline clusters, the interface signals and protocol with the memory subsystem can have high similarity levels. Memory reordering operations of an out-of-order execution engine take place within the pipeline cluster, enabling a simplified memory reference and completion protocol with the memory subsystem block. This may come at the expense of performance degradation, due to a higher level of decoupling between load and store issue queues and the memory subsystem fill-buffer, data cache, and TLBs.

The shared front-end, memory subsystem, and Vector-ALU clusters are designed to support high-throughput requirements of the ACC when the big pipeline is activated, or when multiple simple pipelines are activated in a multithreaded constellation. Enabling low power scalability of shared clusters when only the simple pipeline is operating is a fundamental aspect of the ACC architecture and presents several design requirements. Static clock gating is needed for underutilized parts like a decoder block, a surplus SIMD adder, or a secondary memory load port. Sleep transistors may be implemented to reduce leakage power of the complex cluster, gated blocks and arrays, at the cost of additional unit capacitance.

The complex and simple pipelines may be designed to be mutually exclusive. Doing so simplifies the operation and validation of the front-end and memory subsystem blocks. However, to enable thread scalability, there is a need to allow concurrent operation of the complex and simple pipelines. As long as the interface signals and protocols between the pipeline clusters, the front-end, and the memory subsystem blocks enjoy high similarity levels, we expect most of the design and validation challenges to be higher than the scope of a corresponding SMT core, but still within the scope of a reasonable design target.

3. QUANTITATIVE ANALYSIS METHODOLOGY

The EPI (Energy Per Instruction) of a processor is the processor average power divided by the rate of instructions committed per second. We define *energy efficiency* = $1/EPI$. The quantitative analysis approach taken in this article is comprised of relative energy-efficiency estimates based on silicon data and modeled performance of scalar workloads.

An approximated range of the relative energy efficiency is given for the ACC between the two operation modes described in Section 2. With this estimate and the power-performance heuristics described in Section 4, simple mode switch decision rules may be integrated into a performance simulation model.

For the remainder of this article, we will refer to execution of the simpler, energy-optimized Pipe A as executing the core in “small mode.” The execution of the more elaborate, performance-optimized Pipe B will be referred to as executing the core in “big mode.”

3.1. Relative Energy-Efficiency Estimates

There are many factors that affect the relative energy efficiency of the two modes of operations of the proposed asymmetric clustered core. The two asymmetric pipelines have distinct power-performance curves due to differences in fundamental microarchitectural attributes—for example, micro-ops issue width and reorder-buffer depth. Hardware floorplan considerations as well as power and clock distribution schemes impact the achieved energy saving of the small mode configuration relative to the big mode configuration. The behavior of the task being executed results in different power dissipation in various parts of the processor, depending on which execution mode takes place.

We provide a relative energy-efficiency estimate scaled to a 22nm process for the two distinct execution pipelines of the ACC shown in Table I. Our estimate is based on real silicon data of two representative Intel processors correlated with internal power and performance model. For a given task with a specific number of instructions that need to be committed by the processor, the EPI would be proportional to the processor average power multiplied by the time duration of the task. The processor average power equals the sum of average dynamic power consumed when transistors change their logic state and the average leakage power. The following equation provides the dynamic portion of EPI scaled from a baseline lithographic process to a target lithographic process.

$$\begin{aligned} EPI_{Dynamic|Scaled} &\propto L_s * V_s^2 * F_s * P_{Dynamic} * \left(\frac{T}{F_s} \right) \\ &= L_s * V_s^2 * P_{Dynamic} * T \end{aligned}$$

Here $EPI_{Dynamic|Scaled}$ is Energy Per Instruction invested in transistors’ logic state changes scaled to the target lithographic process, L_s is the Lithographic pattern-length ratio, V_s is the Voltage scaling factor, F_s is the Frequency scaling factor, T is the Time duration to complete a given task estimated on baseline silicon process, and $P_{Dynamic}$ is the average CPU dynamic power dissipation during workload execution estimated on the baseline lithographic process.

We make an assumption which is relevant for computation-intensive workloads, that the processor’s average power is proportional to the processor’s thermal budget limit. While workloads that are characterized by long idle periods are bound to be optimized by power management schemes in a manner that increases the gap between peak power and average power values, computation-intensive workloads tend to utilize more of the available thermal budget of the processor. Another assumption we make is that the dynamic power consumption of the nonclustered parts of the ACC, like the shared front-end and ALU blocks, the cache hierarchy and the memory subsystem, are designed to scale linearly with data and command throughput of the two modes of operation of the clustered core. The correlation between the processor’s average power and thermal budget limit is affected by power dissipation variances through application phases, suboptimal thermal capping algorithms, and the actual power-performance optimization scheme used. The correlation between nonclustered dynamic power to data and command throughput is affected by differences in microarchitectural

Table III. Processor Parameters

	Intel Pentium III	Intel Core2 Duo E6600
SPEC CPU2000 Score	450	2560
TDP[Watt]	29	65
Vcc[V]	1.75	1.3
Freq[GHz]	1	2.4
Lithography[nm]	180	65
L2 cache [KB]	256	4096
Front-side bus freq [MHz]	133	1066
Area [mm^2]	106	143

scaling characteristics, clock gating/performance trade-offs, inherent overhead of clock delivery, and maintenance logic of system units like on-die firmware, memory, and I/O controllers. To account for these differences, a 20% charge for additional energy overhead is included for the ACC small mode calculation.

The processor leakage power is dependent on temperature, total transistors' width, threshold voltage, gate insulator thickness, and dielectric constant. As the silicon process scales, the relative part of leakage power out of the total processor power significantly increases due to the reduction in threshold voltage and gate insulator thickness. High-k dielectric gate insulator material, which is used in our target 22nm silicon process, usage of low-leakage transistor devices where applicable, and dynamic power gating with sleep transistors are a few methods for mitigating the increase in leakage power. For the purposes of dynamic power scaling, the typical leakage power of the baseline silicon process is subtracted from the estimated processor average power.

The scaled average power of big mode configuration is based on the scaled big mode dynamic power and the typical leakage power of the target silicon process. The scaled average power of small mode configuration is based on the scaled small mode dynamic power and leakage power taken from the big mode calculation. The core leakage power in the small mode configuration was reduced to account for potential power gating and smaller transistor devices. Smaller transistor devices in the small mode pipeline cluster are attributed to the decrease in the long interconnect count relative to the big mode configuration. Note that the adjustment of the small mode leakage power was done only on the core power portion, since L3 cache and I/O leakage power are not affected by the operating mode of the ACC.

The silicon data used for relative energy-efficiency estimate is taken from processor data sheets and published benchmark results. The processors used as references for the ACC small mode and big mode of operation are Intel Pentium III and Intel Core2 Duo processors, respectively [INTEL Pentium 2013; INTEL Core2 Duo 2013]. Although not identical in every aspect of the internal pipeline and characteristics of the hardware blocks, these two cores sufficiently resemble the asymmetric core parameters defined in Table I. The time delay performance numbers are taken from published SPEC CPU2000 benchmark results for the aforesaid two processors [SPEC 2013]. SPEC CPU2000 is a compute-intensive workload with a recorded history that spans several generations of processors (including the Intel Pentium III and Intel Core2 Duo processors). Hence it may be considered suitable for the type of analysis carried out in this work. Internal power partitioning and performance-sensitivity models were used to account for microarchitectural differences between the two reference processors.

The resulting EPI ratio between the preceding two processors scaled to the 22nm process is 3.8. In the following parts of this work we use a range of 3–5 for the EPI ratio between big mode and small mode configurations of the ACC.

Table IV. Simulated Benchmarks

Benchmark	Category
mcf	Combinatorial Optimization
ammmp	Computational Chemistry
vpr	FPGA Circuit Placement and Routing
gzip	Compression
bzip2	Compression
crafty	Game Playing: Chess
gap	Group Theory, Interpreter
art	Image Recognition/Neural Networks
equake	Seismic Wave Propagation Simulation
mesa	3-D Graphics Library

3.2. Simulation Methodology

Performance modeling is done using PTLsim [Yourst 2007], a microprocessor simulator and virtual machine for the x86 and x86-64 instructions set. PTLsim models a superscalar out-of-order x86-64 compatible processor, a complete cache hierarchy, and a memory subsystem. The simulation tool can run natively on x86/x86-64 platforms and switch between simulation mode and native mode in a way that is transparent to the executing user code. The PTLsim simulator was modified to simulate an ACC having two distinct execution pipes and an ability to dynamically switch between the pipes following a pipeline flush and memory fence. Switch overhead on machine mode change is modified by inserting a switch delay to the mode switch operation. To account for full pipeline flush we included memory fences. Memory fences may have long latencies if the store buffer is full and cache misses occur or on write-combine operations. However, we found (based on Intel internal performance model) typical memory fence latency to be 20-30 core cycles due to store buffer average occupancy at a pipeline flush and due to data locality. This latency was included as part of the switch delay parameter.

The modified PTLsim model was checked with 10 benchmarks from the SPEC CPU2000 [SPEC 2013], suitable for validating the modified PTLsim model against published results. The simulated benchmarks are listed in Table IV. Using the ability of PTLsim to run in native mode, actual simulation was triggered in each benchmark with statistics gathered after 20 million cycles to account for warm-up time. For each simulation 100 million instructions were executed.

4. ACC MODE SWITCHING

The given two distinct execution pipes of the ACC manifest two execution modes. In order to efficiently utilize the ACC big and small operation modes, decision rules for switching between the modes are needed. These decision rules are based on power and performance metrics with different weights given to either power or performance depending on diverse factors like application type, usage model, thermal dissipation, and battery status.

Several potential decision rules for switching between the cores may be considered. One potential metric strives to optimize application power efficiency, at the cost of a bounded performance loss, by minimizing the energy spent per instruction (EPI), under a constraint of allowed performance degradation. Another metric treats the allocated power budget for computation as a constant, and tries to adjust the energy allocated per instruction to the level of parallelism of the running program (IPS—Instructions Per Second) in order to converge to the given power target: $EPI \times IPS$ [Grochowski et al. 2004]. Energy delay [Gonzalez and Horowitz 1996] or energy delay² is another metric

Table V. Core States and Mode Residency

Core State	Big Mode Residency	Peak Power (W)
CS4	100%	5.4
CS3	75%	4.2
CS2	50%	3.1
CS1	25%	2.0
CS0	0%	0.8

Peak power estimates based on Intel internal power model.

that strives to optimize the application's power efficiency, but also aims at giving weight to the execution performance.

Having two different modes of operation for the core resembles in some ways other techniques that optimize for performance and power consumption, such as dynamically gating-off parts of core that do not contribute much to the ongoing computation, or scaling the core operating voltage and frequency. These different optimization techniques have different performance, power, and mode switch latencies and may be grouped together for achieving a wide range of potential core operating points.

4.1. Power Budget Allocation in ACCs

Efficient allocation of the power budget to the different system components is an essential task in CPU optimization. This task is challenging in a single-threaded environment, given the dynamic nature of the requirements of an executing application and the complex interaction with other system components like memory and I/O interfaces. The inherent complexity of power budget allocation increases substantially in a multithreaded environment, due to the difficulty in predicting the power needs of running threads and the time constants involved in adjusting the system to the requirements of the threads.

Intel Software Developer's Manual [2012] provides a software interface mechanism to enforce power consumption limits through machine-specific registers termed Running Average Power Limit (RAPL). Thermal limits and averaging window sizes that represent characteristics like platform thermal constraints and type of cooling solution are programmed into RAPL registers. Hierarchical partitioning of these registers is supported and may be used to control power dissipation limits of the package, DRAM cards, and two power planes, namely PP0 (refers to processor cores) and PP1 (may refer in client platforms to system devices and shared L3 cache). On-die firmware may make use of these registers when translating power state directives of the operating system into an actual voltage and frequency working point. The on-die firmware takes into account the available real-time power budget for worst-case thermal capping calculation.

In the context of ACCs, the RAPL mechanism may be extended for efficient utilization of the dynamic power-performance range of the cores. The algorithm presented in David et al. [2010] utilizes hard power limits for guaranteed thermal capping of server memory bandwidth, and soft power limits for predicting actual power demand based on recent workload behavior. This algorithm can be used to smooth the effect of processor thermal limiting in a multithreaded environment and take advantage of application phases. Table V provides asymmetric core state (CS: Core State) definitions with worst-case peak power limits for different big mode residencies. These power limits approximately match the core parameters defined in Table I for a processor built in a 22nm silicon process and running at 2 GHz with 0.9v voltage supply.

The RAPL algorithm takes as its input a basic clock tick interval for measuring power data, which we will assume to be 1 msec. The following equation defines a power

hard limit for a single core. Note that real-time power values may be either directly measured by power monitor circuits or indirectly estimated by activity counters.

$$CorePwrBudget_{hardlimit} = \frac{N * PwrLimit - \sum_{i=1}^{N-1} ProcessorPwr_i}{CoreNum}$$

Here

N is the fixed averaging window size,

$PwrLimit$ is the processor power limit,

$ProcessorPwr_i$ is the measured processor power at tick interval i , and

$CoreNum$ is the number of processor ACCs.

The following equation defines a soft power limit for a single core.

$$CorePwrBudget_{softlimit} = \frac{N * PwrLimit}{M * CoreNum} - \frac{\sum_{i=M}^{N-1} CorePwr_i}{M}$$

Here

M is the sliding averaging window size, $M < N$ and

$CorePwr_i$ is the measured ACC power at tick interval i .

The hard limit averaging window N is typically programmed to a value of hundreds of milliseconds and above to account for warm-up and cooling changes in the platform temperature. The soft limit averaging window M is programmed to a few milliseconds to track actual changes in application power-performance behavior. In addition to the hard and soft power limit measurements, the RAPL algorithm also updates a lookup table that captures the history of the actual core power consumption at each of the defined core states.

$$CAP^{new}[CS] = \alpha * CorePwr + (1 - \alpha) * CAP^{curr}[CS]$$

Here

$CAP[CS]$ is the specific core state entry of the core average power table, and

α is the weight factor to account for historic average power measurements.

In each measurement interval the RAPL algorithm looks up the corresponding core state of a CAP entry that satisfies the measured $CorePwrBudget_{softlimit}$. It then checks if the worst-case power (from Table V) of the selected core exceeds the measured $CorePwrBudget_{hardlimit}$, in which case the core state is demoted to ensure the viability of the thermal hard power constraint. In most cases the core state will be based on $CorePwrBudget_{softlimit}$, since the statistical distribution of actual average power consumption falls below the worst-case thermal capping power calculation. The resulting core state serves as an input parameter for fine-grain switch heuristics described in Section 4.2.

Note that the mode residency states of a core, shown in Table V, may be extended to cover a wider power range by taking into account potential DVFS transitions. Integration of on-die voltage regulators can facilitate this potential direction by mitigating the issue of handling multiple power-rails and enabling per-core low-latency DVFS control, at the cost of reduced efficiency of power delivery and load-line transient effects. The potential interaction between DVFS and mode transitions in ACCs is left as a future research topic.

4.2. Mode Switch Heuristics

Switch heuristic policies are determined at various levels. The operating system cooperates with on-die firmware code to manage the platform power dissipation and maintain fairness and required quality of service between multiple threads. Compilers and software profiling tools may employ switch algorithms through macros and directives, exploiting wide-scope visibility of the application. A core can track statistics of currently running physical threads which were allocated to it. The hardware may monitor runtime characteristics at finer granularity than software but has narrower visibility of the running application. Given ACC thread scalability properties, there is also a need to provide selection rules that cover bids from multiple threads for a populated or nonpopulated complex pipeline. Note that within the scope of this work we consider only single-thread mode selection algorithms.

For a given decision metric and workload, aimed at optimizing a specific thread on a specific core, there is an ideal time partitioning between the two execution modes of the ACC, taking into account also the switch overhead. Common methods for dynamic hardware configuration approximate an ideal mode change, utilizing the periodic or bursting nature of many workloads. Methods for identifying different program phases were proposed in the past. For example, Dhodapkar and Smith [2002] present an algorithm that tracks phase changes in hardware by monitoring the memory working-set in a specific time window. Training periods are used to characterize an optimal hardware setting for each phase, and history of past training periods is kept to reduce the overhead of new training periods.

The mode switch algorithm used in this work takes as input a core state directive, which is based on the RAPL definition given in Section 4.1 and limits the allowed big mode residency of the core. An additional input is a decision metric method of either energy delay² or EPI minimization under a fixed performance loss constraint. The core makes progress in execution intervals, initiates a training period at the start of each interval, and samples committed instructions to select one of three potential operating modes for the current execution interval: operate in small mode, operate in big mode under the RAPL residency constraint, or implement fine-grain mode switches under the RAPL residency constraint.

In the fine-grain operation mode, the core attempts to make mode transitions based on short phase transitions in the application. The fine-grain mode change is driven by actual machine utilization and a few temporal microarchitectural hints, such as cache miss rate, memory data return indications, and internal buffers' utilization indicators. When switching to a mode, the core starts in an unused state and can move to a utilized state based on the actual instruction commit rate over time. Changes in core utilization are identified and drive, along with additional microarchitectural indications, another mode switch. Figure 4 describes the utilization and state machines associated with dynamic switching of modes.

The machine utilization metrics and threshold values affect the probability of mode switches. These metrics are derived from a lookup table that translates the provided RAPL core state to transition functions and hardware threshold values. Figure 5 specifies heuristics used for mode transitions. The transition algorithm tracks machine utilization status and tries to schedule the actual mode transition on a demand load miss accompanied by a core stalled condition. Additional runtime statistics that track core units occupancies may be added to the mode transition function. Table VI lists utilization metrics and threshold values used to drive state transitions of the core illustrated in Figure 4. Policy A indicates a directive that favors higher big mode residency. Policy B indicates a directive that favors low mode residency.

Fig. 4. State machines that implement dynamic switching between modes. All four state machine combinations occur during normal run of the core.

```

{Input: run-time parameters and directives}
{Output: ACC mode}
{initialization;}
if big mode then
    if (utilized counter ≤ big mode utilized threshold) and demand_miss_happened
        core_stalled and not load_completion then
            machine mode ← small mode
        end if
    else
        if (utilized counter ≤ small mode utilized threshold) and demand_miss_happened
            core_stalled and not load_completion then
                machine mode ← big mode
            end if
        end if
    end if

```

Fig. 5. Algorithm for machine mode transitions. Following are the definitions of the microarchitectural events used in the algorithm: *demand_miss_happened*—load operation missed the L1 data cache in the last few cycles; *core_stalled*—retirement stalled in the last few cycles because of unresolved data dependencies; *load_completion*—early indication that a memory load operation is about to complete in a few cycles.

5. ACC MODE SWITCHING RESULTS

Figure 6 presents the runtime results of the ACC, using the SPEC CPU2000 benchmarks, for a range of energy-efficiency ratios with the energy-delay² optimization directive given in Section 4. Differences in application phases for the given switch directives drive the execution mode changes, as illustrated in Figure 7, for a 9-million-cycle window of the SPEC2000 gzip application. Figure 8 presents the big and small mode

Table VI. Utilization Metrics and Threshold Values.

Metric	Policy A	Policy A	Policy B	Policy B
	$CS > CS2$	$CS > CS2$	$CS \leq CS2$	$CS \leq CS2$
	Big mode	Small mode	Big mode	Small mode
Width utilized	80%	80%	80%	50%
Width not utilized	20%	20%	50%	20%
Threshold utilized	6	8	8	6
Threshold not utilized	2	4	4	2

Fig. 6. Runtime results of the ACC.

Fig. 7. Gzip application runtime parallelism and mode residencies.

residencies relative to the big mode runtime for the same energy-delay² optimization directive. The results are given under the assumption that mode switch overhead incurs a pipeline flush and a memory fence.

A sensitivity analysis of interest in the context of the ACC is the effect of different switch times between modes. For example, higher switch time can be attributed to internal context save which is accompanied by power gating of an unused pipeline cluster. Table VII lists values of mode switch latencies. For comparison purposes, we

Fig. 8. Mode residency of the ACC.

Table VII. ACC Mode Switch Latencies—Sensitivity Analysis

Asymmetric Clustered Core	Switch Latency	Switch operation
Highly-coupled	20ns	pipeline flush; memory fence
Slightly-coupled	100ns	internal context save to enable power-gating

will term an ACC with fast mode switch time as a highly coupled ACC and an ACC with slower mode switch time slightly coupled ACC.

Figure 9 and Figure 10 present energy and performance curves as a function of the characteristic relative energy efficiency of the ACC. The graphs are given for highly coupled and slightly coupled ACC with the energy-delay² optimization directive. As observed from the graphs, for the given switch heuristic an ACC with higher relative energy-efficiency ratio between the simple pipeline and the complex pipeline uses smaller amount of energy at the expense of higher performance loss. Assuming a given performance gap between ACC simple and big pipelines, increasing the energy consumption gap between ACC simple and big pipelines results in a higher tendency for using the more energy-efficient simple pipeline.

In addition, Figure 9 and Figure 10 demonstrate small energy and performance sensitivity to ACC mode switch latency. For ACC with low relative energy-efficiency characteristic, the energy saving and performance loss differences between highly coupled and slightly coupled ACC were 1.3% and 0.6%, respectively. For ACC with high relative energy-efficiency characteristic, the energy saving and performance loss between highly coupled and slightly coupled ACC were 2.9% and 1.4%, respectively.

Providing a power-performance directive of energy delay² can lead to substantial performance degradation, as may be seen from Figure 10 due to the significant difference in the energy-efficiency between the two execution pipes. Another power-performance directive described in Section 4 aims to optimize for energy efficiency at a bounded value of allowed performance degradation. Figure 11 presents energy consumption for different ACC mode switch latencies at an allowed performance loss of 10%. An increase in mode switch latency for the given switch heuristics results in up to 1.2% energy saving decrease for small relative energy-efficiency characteristic and up to 2.5% energy saving decrease for high relative energy-efficiency characteristic.

Fig. 9. ACC energy.

Fig. 10. ACC performance.

Fig. 11. ACC energy consumption under a constraint of 10% allowed performance loss.

Table VIII. Core Area Size Estimates Scaled to 22nm Process

Core type	L2 size	Area
Big core	256KB	5.5 mm^2
Small core	No L2	0.96 mm^2
ACC	256KB	6.88 mm^2
Asymmetric Multicore	512KB	8.4 mm^2

6. ACC AND ASYMMETRIC MULTICORE COMPARISON

In this section we discuss and compare characteristics of the proposed ACC and asymmetric multicores, including thread scalability, layout and chip-level integration considerations, and mode switch overhead differences implications.

6.1. Thread Scalability Comparison

A central aspect of the ACC architecture is the low-cost transition of a thread between execution pipelines. To enable low-overhead transitions, the number of physical threads exposed to the operating system is limited to the SMT width of Pipe A. Acceleration of a thread by switching to Pipe B is done locally without OS involvement. While allowing quality-of-service and performance directives from the operating system, limiting the number of physical threads to the SMT width of Pipe A enables fast mode transitions, triggered by hardware signals and counters.

In the asymmetric multicore architecture full population of big and small cores, managed by OS scheduling, has to be achieved for reaching optimal peak performance. This becomes more challenging when the logical thread count increases and their quality of service and performance requirements diversify. Exposing a subset of symmetric cores to the operating system in an asymmetric multicore architecture simplifies the SW requirements at the cost of wasted silicon area. Platform thermal limit [Esmaeilzadeh et al. 2011] may be another factor which drives the partial exposure of physical threads to the operating system. It should be noted that ARM Cortex [2011] amd Nvidia Tegra [2013] asymmetric multicore chips designed for energy-efficient, low form-factor products expose a symmetric group of cores, which streamlines power management. For ACCs with multiple simple pipelines, the relative area cost of an unexposed complex pipeline decreases.

6.2. Layout and Chip-Level Integration Considerations

Table VIII shows the area size of different core types, scaled to 22nm silicon process. The data are based on the core parameters given in Table I and silicon die analysis and block area scaling factors given in Section 2.2. The big core area size is relatively small compared to current high-end commercial cores for client and server markets, because it is based on direct scaling from 65nm Intel Core2 Duo processors [INTEL Core2 Duo 2013] to the 22nm process, leaving aside features and capabilities incorporated since into Intel's core product line (for example, hyperthreading and AVX architecture).

The area increase over a baseline big core of ACC with two simple execution pipelines and a single complex execution pipeline that match the parameters given in Table I is approximately 25%. An equivalent asymmetric multicore managed by an OS has an area overhead of approximately 50% over the baseline big core. The smaller area overhead of the ACC is due to the sharing of resources like the L2 cache between the complex and simple pipelines. The asymmetric multicore area utilization is favorable relative to the ACC if all of the cores are utilized with operating system scheme involvement, providing higher thread count for peak performance.

(a) ACC

(b) asymmetric multicore

Fig. 12. Full chip layout partitioning - 32 physical threads, without OS involvement.

In the layout of a multicore chip the increase in core size has a global impact. Figure 12(a) describes the full chip layout of 16 integrated ACCs that match the core layout presented in Figure 3. Full chip interconnect fabric between the different cores adds approximately 15% to the total chip area, under the assumption that part of the interconnect may be routed above the L3 cache. The chip interconnect illustrated in Figure 12(a) is a two-dimensional mesh with 5-port routers (designated R in the diagram) connecting between the cores and the fabric. The L3 cache is shared between the different cores and its size is set to four times the aggregate size of the L2 caches to match the parameters given in Table I. The cores share a router port with their locally attached L3 cache slice.

For comparison, Figure 12(b) presents a schematic layout option of an asymmetric multicore chip managed without OS involvement with 16 big cores and 32 small cores. In this diagram, the AC-Big block indicates a “big” core, and the 4xAC-Small block indicates a group of four “small” cores. The specific group count of four small cores is given to avoid “white-space” inefficiencies in the chip integration and provide balanced port bandwidth on the chip interconnect fabric. The chip-level area of the presented

Fig. 13. Area partitioning of chip components for ACC and asymmetric multicore chips.

asymmetric multicore option is found to be approximately 14% higher than the corresponding area of the ACC option. Figure 13 shows the area partitioning of major chip components for the ACC and asymmetric multicore chips.

An increase in the router count affects the chip-level power consumption. The interconnect power consumption is dependent on various parameters—frequency and cross-section bandwidth of the interconnect, aggregate throughput requirements of attached cores, interconnect routing scheme, usage of virtual channels, number of routing elements and their internal design. Howard et al. [2010] provide a power breakdown for a two-dimensional mesh interconnect of a 48-tile research chip, indicating a 10% chip power consumption by highly optimized chip routers and interconnect fabric, for high interconnect bandwidth scenarios. Assuming an equivalent power consumption ratio for our proposed chip and proportional ratio between interconnect power consumption and routing elements count, the additional routers and interconnect of the asymmetric multicore option will result in additional 5% chip-level power consumption relative to the ACC chip.

There are many options for local grouping of cores with various numbers of interconnect routers and chip-level interconnect schemes. For example, the asymmetric multicore small cores and big cores may be grouped in bundles to reduce interconnect routers count, at the expense of an additional interconnect router bandwidth requirement. In addition, asymmetric multicores provide higher flexibility in determining the ratio of complex to simple execution pipelines. While we don't make an attempt to optimize for chip-level interconnect within the scope of this article, we conclude that the chip comparison examples given in this section represent a reasonably balanced integration solution in terms of area and routers bandwidth, and it demonstrates the advantage of resource sharing in the ACC. This advantage exists only for a fixed ratio of simple to complex pipelines and under the assumption of no OS involvement, with thread count limited to the number of simple pipelines or small cores.

6.3. ACC and Asymmetric Multicores: Performance and Energy Comparison

Differences in performance and energy characteristics between ACC and asymmetric multicores are attributed to differences in mode switch latencies and differences in interconnect structure. The shorter mode switch latencies of ACC compared to asymmetric multicores effectively increase the power/performance dynamic range of the former architecture. The reduced core count of ACC architecture compared to asymmetric multicores leads to a smaller interconnect fabric with lower power consumption.

The benefit of a wider power/performance dynamic range is dependent on workload and usage model. For example, consider a power management scheme that strives

Table IX. ACC and Asymmetric Cores Mode Switch Latencies.

Core Type	Switch Latency	Switch operation
ACC	20ns	pipeline flush; memory fence
Asymmetric Multicore	10us	core migration, including L2 cache migration.

Fig. 14. ACC and asymmetric multicore: energy and performance comparison.

to optimize user experience through better responsiveness under multiple system constraints.

For comparison between ACC and asymmetric multicores we will consider the power and performance effects of ACC with fast mode switch time and asymmetric multicores with a slower mode switch time. Table IX lists values of mode switch latencies for ACC and asymmetric multicores.

Figure 14 presents both energy and performance curves for ACC and asymmetric multicore, with the energy-delay² optimization directive. The resulting energy saving and performance loss follow a similar pattern relative to the one presented in Figure 9 and Figure 10. The significant mode switch latency difference between ACC and asymmetric multicore leads to higher differences in energy saving and performance losses. For the given switch mode heuristic, a highly coupled ACC exhibited lower energy consumption of 5–11% at an about 3–6% additional cost of performance degradation relative to asymmetric multicore.

7. RELATED WORK

The ability to reconfigure a CPU at runtime and form different computation entities according to dynamic workload demand was suggested in several works. Clustered core architecture was presented in Farkas et al. [1997] as a method for reducing the clock cycle time as a result of better partitioning of internal hardware blocks. A method for predicting runtime variations in thread parallelism and switching between in-order and out-of-order operation modes for the purpose of saving power was presented in Ghiasi and Grunwald [2000]. Kumar et al. [2004] investigated possible sharing of floating point units, crossbar ports, instruction and data caches between adjacent cores of a chip multiprocessor. The potential morphing of heterogeneous cores with floating point and integer units that exhibit different performance characteristics was described in Das et al. [2010]. Another form of core morphing was presented in Ipek et al. [2007] suggesting to group independent cores into a larger CPU as needed at runtime by

applications. MorphCore [Khubaib et al. 2012] begins with a conventional out-of-order core, which may be used for high-performance applications, but may be also morphed into an in-order SMT configuration for the purpose of high throughput. Another recent work, Composite Cores [Lukefahr et al. 2012], suggests an architecture in which big and little microengines share a substantial proportion of the architectural state, thus reducing switching time. Although this proposal shares some similarity with our work at the conceptual level, the two approaches are very different in terms of the design details.

The benefits of heterogeneous computing were investigated in several papers. For example, Kumar et al. [2003] and Kumar [2005] show power, performance, and area implications when tuning the selection of four types of Alpha cores to a workload ILP (Instruction-Level Parallelism). A theoretical analytical model describing the potential performance benefit for a given power envelope of ACCMP (Asymmetric Cluster Core MultiProcessor) as opposed to SCMP (Symmetric Cluster Core MultiProcessor) was given in Morad et al. [2006]. The analytic model was based on an empirical observation that core performance is roughly proportional to the square root of its area. The proposed ACCMP architecture was based on distinct cores of different sizes grouped in clusters.

A scheme that combines a narrow out-of-order core with a wide in-order core and performs binary translation for efficient utilization of the wide in-order core was described in Wu et al. [2011]. Another work that combined hardware and software mechanisms to efficiently accelerate critical-section-intensive workloads on an asymmetric multi-core architecture was described in Suleman et al. [2009] with a reported acceleration of 23% relative to a nonimproved asymmetric multicore architecture and 34% performance improvement relative to a symmetric CMP architecture.

While asymmetric architectures present new opportunities for saving power and chip area, utilizing such chips involves scheduling challenges at the operating system level [Li et al. 2007]. Li et al. [2010] propose algorithms for sharing heterogeneous cores among applications. These algorithms were implemented in the Linux 2.6.24 kernel and their performance evaluated by running the OS on a heterogeneous multicore processor. The paper reports performance improvements for a set of applications. Saez et al. [2010] propose a comprehensive scheduler for asymmetric multicore processors. The scheduler was implemented in the OpenSolaris operating system. The evaluation reported in the paper shows that the proposed scheduler utilizes asymmetric cores in an efficient way for a range of applications.

8. SUMMARY AND CONCLUSIONS

ACC is an alternative for asymmetric multicores, providing a wide dynamic range of power-performance characteristics and enabling efficient multithreading scalability. The ACC provides an opportunity for efficient core-level optimization at the expense of core complexity and loss of flexibility in setting a distinct pipeline ratio. Under an assumption of no operating system involvement, for a ratio of two between simple and complex pipelines and for a specific core configuration provided in this work, a comparable asymmetric multicore incurs a 14% increase in total chip area, accompanied by additional chip-level power dissipation of 5%. The asymmetric multicore gains in area efficiency when peak power is managed by an operating system that is aware of the distinct core type and power-performance characteristics.

The ACC architecture is characterized by an ability to perform low-latency mode switches, locally within the core, according to directives and hints received from software and firmware. The benefit of having a highly coupled ACC relative to a partially decoupled ACC or a fully decoupled asymmetric core was demonstrated by the ability of the highly coupled ACC to utilize better power directives from software and on-die firmware. The higher utilization of power directives was manifested by achieving lower

energy values for different directives such as energy delay² or minimal energy with a constraint of allowed performance loss. For the directive of energy-delay² optimization, a highly coupled core exhibited lower energy consumption of 5–11% at an about 3–6% additional cost of performance degradation. For the directive of minimal energy consumption at a constraint of allowed performance loss, the highly coupled ACC was able to utilize the directive for energy saving beyond the given performance loss limit, while the decoupled ACC achieved an energy saving level that was lower than the given performance loss limit, due to the high overhead of changing modes. The RAPL algorithm described in this work can utilize additional energy credits returned by a highly coupled ACC for efficient thermal capping and power budgeting in a multithreaded environment.

As a future research topic, the multithreading scalability characteristics of ACCs relative to asymmetric multicores should be explored. One of the most challenging tasks when operating in a multithreaded environment is the runtime allocation of threads among distinct core types. Our expectation in this context is that the ability of the RAPL algorithm to allocate energy credits across application phases and among multiple threads, and the ability of the ACC to make local mode switch decisions based on directives derived from the RAPL energy budget will provide more efficient scaling relative to asymmetric multicores.

ACKNOWLEDGMENT

We thank the reviewers for their constructive suggestions.

REFERENCES

- ARM CORTEX. 2011. Big. LITTLE processing with arm cortex-a15 & cortex-a7. <http://www.arm.com/files/downloads/big.LITTLE.Final.pdf>.
- BALAKRISHNAN, S., RAJWAR, R., UPTON, M., AND LAI, K. 2005. The impact of performance asymmetry in emerging multicore architectures. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*. 506–517.
- BORKAR, S. 1999. Design challenges of technology scaling. *IEEE Micro* 19, 4, 23–29.
- BROOKS, D. AND MARTONOSI, M. 2001. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*. 171–182.
- BURNS, J. AND GAUDIOT, J.-L. 2001. Area and system clock effects on SMT/CMP processors. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*. 211–218.
- BURNS, J. AND GAUDIOT, J.-L. 2002. SMT layout overhead and scalability. *IEEE Trans. Parallel Distrib. Syst.* 13, 2, 142–155.
- CHANDRAKASAN, A., POTKONJAK, M., RABAAY, J., AND BRODERSEN, R. 1992. HYPER-LP: A system for power minimization using architectural transformations. In *Proceedings of the International Conference on Computer-Aided Design*. 300–303.
- DAS, A., RODRIGUES, R., KOREN, I., AND KUNDU, S. 2010. A study on performance benefits of core morphing in an asymmetric multicore processor. In *Proceedings of the International Conference on Computer Design*. 17–22.
- DAVID, H., GORBATOV, E., HANEPUTTE, U. R., KHANNA, R., AND LE, C. 2010. RAPL: Memory power estimation and capping. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 189–194.
- DENNARD, R., GAENSSLEN, F., RIDEOUT, V., BASSOUS, E., AND LEBLANC, A. 1974. Design of ion-implanted mosfet's with very small physical dimensions. *Proc. IEEE* 9, 5, 256–268.
- DHODAPKAR, A. AND SMITH, J. 2002. Managing multi-configuration hardware via dynamic working set analysis. In *Proceedings of the Annual International Symposium Computer Architecture*. 233–244.
- EGGERS, S., EMER, J., LEHY, H., LO, J., STAMM, R., AND TULLSEN, D. 1997. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro* 17, 5, 12–19.
- ESMAELIZADEH, H., BLEM, E., ST. AMANT, R., SANKARALINGAM, K., AND BURGER, D. 2011. Dark silicon and the end of multicore scaling. In *Proceedings of the Annual International Symposium on Computer Architecture*. 365–376.

- FARKAS, K., CHOW, P., JOUPPI, N., AND VRANESIC, Z. 1997. The multicluster architecture: Reducing cycle time through partitioning. In *Proceedings of the Annual International Symposium on Microarchitecture*. 149–159.
- GHIASI, S., CASMIRA, J., AND GRUNWALD, D. 2000. Using ipc variations in workloads with externally specified rates to reduce power consumption. In *Proceedings of the Workshop on Complexity Effective Design*.
- GONZALEZ, R., GORDON, B., AND HOROWITZ, M. 1997. Supply and threshold voltage scaling for low power CMOS. *IEEE J. Solid State Circ.* 32, 8, 1210–1216.
- GONZALEZ, R. AND HOROWITZ, M. 1996. Energy dissipation in general purpose microprocessors. *IEEE J. Solid State Circ.* 31, 9, 1277–1284.
- GROCHOWSKI, E., RONEN, R., SHEN, J., AND WANG, H. 2004. Best of both latency and throughput. In *Proceedings of the IEEE International Conference on Computer Design*. 236–243.
- HOWARD, J., DIGHE, S., HOSKOTE, Y., VANGAL, S., FINAN, D., RUHL, G., JENKINS, D., WILSON, H., BORKAR, N., SCHROM, G., PAILET, F., JAIN, S., JACOB, T., YADA, S., MARELLA, S., SALIHUNDAM, P., ERRAGUNTLA, V., KONOW, M., RIEPEN, M., DROEGE, G., LINDEMANN, J., GRIES, M., APEL, T., HENRISS, K., LUND-LARSEN, T., STEIBL, S., BORKAR, S., DE, V., VAN DER WIJNGAART, R., AND MATTSON, T. 2010. A 48-core ia-32 messagepassing processor with dvfs in 45nm CMOS. In *Proceedings of the International Solid-State Circuits Conference*.
- INTEL CORE2 DUO. 2013. Intel core2 duo processor e6600. <http://ark.intel.com/products/27250/Intel-Core2-Duo-Processor-E6600-4M-Cache-2-40-GHz-1066-MHz-FSB>.
- INTEL PENTIUM. 2013. Intel pentium iii processor. <http://ark.intel.com/products/27528/Intel-Pentium-III-Processor-1-00-GHz-256K-Cache-100-MHz-FSB>.
- INTEL SOFTWARE DEVELOPER'S MANUAL. 2012. Intel 64 and ia-32 architectures software developers manual volume 3b. <http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.html>.
- IPEK, E., KIRMAN, M., KIRMAN, N., AND MARTINEZ, J. F. 2007. Core fusion: Accommodating software diversity in chip multiprocessors. In *Proceedings of the Annual International Symposium on Computer Architecture*. 186–197.
- KHUBAIB, SULEMAN, M. A., HASHEMI, M., WILKERSON, C., AND PATT, Y. N. 2012. MorphCore: An energy-efficient microarchitecture for high performance ILP and high throughput TLP. In *Proceedings of the 45th Annual International Symposium on Microarchitecture*. 305–316.
- KUMAR, R. 2005. Heterogeneous chip multiprocessors. *IEEE Comput.* 38, 11.
- KUMAR, R., FARKAS, K., JOUPPI, N., RANGANATHAN, P., AND TULLSEN, D. 2003. Single-ISA heterogeneous multicore architectures: The potential for processor power reduction. In *Proceedings of the Annual International Symposium on Microarchitecture (MICRO'03)*. 81–92.
- KUMAR, R., JOUPPI, N. P., AND TULLSEN, D. M. 2004. Conjoined-core chip multiprocessing. In *Proceedings of the Annual International Symposium on Microarchitecture*. 195–206.
- LI, T., BAUMBERGER, D., KOUFATY, D. A., AND HAHN, S. 2007. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In *Proceedings of the Conference on Supercomputing*. 53:1–53:11.
- LI, T., BRETT, P., KNAUERHASE, R., KOUFATY, D., REDDY, D., AND HAHN, S. 2010. Operating system support for overlapping-isa heterogeneous multi-core architectures. In *Proceedings of the International Symposium on High Performance Computer Architecture*. 1–12.
- LUKEFAHR, A., PADMANABHA, S., DAS, R., SLEIMAN, F. M., DRESLINSKI, R., WENISCH, T. F., AND MAHLKE, S. 2012. Composite cores: Pushing heterogeneity into a core. In *Proceedings of the 45th Annual International Symposium on Microarchitecture*. 317–328.
- MARVELL TRI-CORE. 2010. Marvell raises technology bar again with worlds first 1.5 GHz Tri-core processor delivering dual stream 1080p 3d video for smartphones and tablets. http://www.marvell.com/company/press-kit/assets/Marvell_ARMADA_628_Release_FINAL3.pdf.
- MORAD, T., WEISER, U., KOLODNY, A., VALERO, M., AND AYGUADE, E. 2006. Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors. *Comput. Archit. Lett.* 5, 1, 14–17.
- NVIDIA TEGRA. 2013. A multi-core CPU architecture for low power and high performance. <http://www.nvidia.com/content/PDF/tegra-white-papers/tegra-whitepaper-0911b.pdf>.
- ROTEM, E., NAVEH, A., ANANTHAKRISHNAN, A., WEISSMANN, E., AND RAJWAN, D. 2012. Power-management architecture of the Intel microarchitecture code-named sandy bridge. *IEEE Micro* 32, 2, 20–27.
- SAEZ, J. C., PRIETO, M., FEDOROVA, A., AND BLAGODUROV, S. 2010. A comprehensive scheduler for asymmetric multicore systems. In *Proceedings of the European Conference on Computer Systems*. 139–152.
- SPEC CPU. 2000. <http://www.spec.org/osg/cpu2000/>.
- SULEMAN, M. A., MUTLU, O., QURESHI, M. K., AND PATT, Y. N. 2009. Accelerating critical section execution with asymmetric multi-core architectures. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*. 253–264.

- TI OMAP 5. 2011. Omap 5 mobile applications platform. http://focus.ti.com/pdfs/wtbu/OMAP5_2011-7-13.pdf.
- TSCHANZ, J., NARENDRA, S., YE, Y., BLOECHEL, B., BORKAR, S., AND DE, V. 2003. Dynamic sleep transistor and body bias for active leakage power control of microprocessors. *IEEE J. Solid State Circ.* 38, 11.
- WU, Y., HU, S., BORIN, E., AND WANG, C. 2011. A HW/SW co-designed heterogeneous multi-core virtual machine for energy-efficient general purpose computing. In *Proceedings of the Annual International Symposium on Code Generation and Optimization*. 236–245.
- YOURST, M. 2007. Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator. In *Proceedings of the International Symposium on Performance Analysis of Systems Software*. 23–34.

Received January 2013; revised May 2013; accepted June 2013