CPACT – The Conditional Parameter Adjustment Cache Tuner for Dual-Core Architectures

Marisha Rawlins and Ann Gordon-Ross*
Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611, USA
mrawlins@ufl.edu & ann@ece.ufl.edu
**Also with the NSF Center for High-Performance Reconfigurable Computing*

***Abstract* – Cache tuning reveals substantial energy savings for single-core architectures, but has yet to be explored for multi-core architectures. In this paper we explore level one (L1) data cache tuning in a heterogeneous dual-core system where each data cache can have a different configuration. We show that L1 data cache tuning in a dual-core system achieves 25% average energy savings, which is comparable to single-core data cache tuning. We present the dual-core tuning heuristic CPACT, which finds cache configurations within 1% of the optimal configuration while searching only 1% of the design space. Finally, we provide valuable insights on core-interactions and data coherence revealed when tuning the multithreaded SPLASH-2 benchmarks.**

Keywords-cache tuning; multi-core; energy savings; low power; embedded systems.

I. INTRODUCTION

Multi-core architectures are becoming a popular method to increase system performance via exploited parallelism without increasing the processor frequency and/or energy. To further improve these systems, multi-core optimizations focus on improving system performance [1][3][4] or energy efficiency [12][13][16][20][21]. In particular, fine-grained multi-core architectures allow applications to be decomposed into smaller subtasks executing on several processors. Application decomposition enables fine-grained energy management techniques by allowing individual devices (e.g., processors, caches, pipelines, etc.), to be shut down or placed in a lower power operating mode. Additionally, considering both homogeneous and heterogeneous processors increases energy reduction potential since the smaller subtasks can be highly specialized to a particular processor.

In this paper, we focus on cache tuning for multi-core systems due to the memory hierarchy's large contribution to overall energy consumption [15] and previous single-core successes [9][10][15][27][28]. Cache tuning enables application-specific optimizations by selecting the lowest energy configuration (e.g., specific values for cache parameters such as cache size, line size, and associativity) that satisfy the application's runtime behavior, since inappropriate configurations can waste energy. For example, caches larger than an application's working set, waste energy by fetching from an unnecessarily large cache, while caches much smaller than the working set waste energy during frequent cache miss stalls. Similarly, an application's locality affects the optimal line size and associativity. Previous single-core cache tuning revealed level one (L1) average energy savings of 40% [27] and two level cache average energy savings of 53% and 62% for systems with a separate [9] and unified [10] level two (L2) cache, respectively. Given these previous single-core successes, we believe cache tuning in multi-core systems may yield similar results if the additional multi-core-specific challenges are addressed.

One of the main multi-core cache tuning challenges is runtime cache configuration design space exploration. Runtime cache tuning requires a hardware configurable cache such as the highly configurable cache in [28] or Motorola's M*Core [15], and a design space exploration methodology. Runtime design space exploration executes the application for a period of time with each potential configuration until the optimal or near optimal configuration is determined. Unfortunately, this process incurs energy and performance penalties while executing inferior configurations. Additionally, large design spaces (dozens for L1 tuning [27] and thousands for two-level tuning [9][10]) makes exhaustive design space exploration infeasible, even for single-core systems. However, cache tuning heuristics significantly reduce the number of configurations explored. Previous single-core heuristics searched as little as 0.2% of the design space and achieved energy savings within 1% of the optimal configuration [10].

Whereas these single-core heuristics were highly effective, developing cache tuning heuristics was challenging given large design spaces, cache interdependencies, and minimizing system intrusion (e.g., certain exploration orders can reduce cache flushing [27]). Multi-core systems not only exacerbate these challenges, but also introduce new challenges. For example, a linear increase in the number of heterogeneous cores with different cache configurations exponentially increases the design space. New challenges include core interdependencies across applications with a high degree of sharing and resource contention, both of which affect cache behavior (e.g., increased cache miss rates) and processor stalls.

In this paper, we quantify data cache tuning energy savings in a heterogeneous dual-core system with highly configurable caches able to configure the total size, line size, and associativity. Our results and methodologies for the data cache in a dual-core system provide valuable insights into core-interactions and data coherence considerations given a manageable design space, and may be leveraged for larger systems. Results reveal that dual-core cache tuning energy savings are comparable to single-core cache tuning energy savings. To exploit these energy savings, we develop the **C**onditional **P**arameter **A**djustment **C**ache **T**uner (CPACT) – a heuristic to determine the optimal or near optimal (within 1%

of the optimal) lowest energy cache configuration by examining only 1% of the design space.

The remainder of this paper is organized as follows. In Section II we present related work in multi-core performance and energy optimizations and single-core cache tuning. Section III discusses dual-core cache tuning and CPACT. Section IV-A describes the benchmarks and experimental setup used for our experiments. Sections IV-B and IV-C present experimental results for the optimal cache and CPACT, respectively. Section IV-D provides valuable insights into core-interactions and data coherence for dual-core systems.

II. RELATED WORK

A. Multi-core Optimizations

Most previous work in multi-core optimizations focused on improving cache performance by reducing off-chip accesses and resource contention using methods such as cooperative caching [4], proximity aware caches [3], scheduling heuristics [1], cache partitioning (for maximizing throughput [14], minimizing miss rate [24], or maximizing speedup and fairness [26]), or cooperative cache partitioning [5]. Some performance-centric techniques (e.g., scheduling) can also target energy consumption. Merkel et al. [16] reduced the energy-delay product using specialized chip frequencies to minimize delay and by co-scheduling tasks to minimize resource contention. Park et al. [20] proposed three synchronization-aware algorithms to estimate thread-dependent slowdowns and achieved energy savings using different processor power modes. Seo et al. [21] balanced the task loads of multiple cores to optimize power consumption and adjusted the number of active cores, resulting in energy savings as high as 26%.

Although multi-core optimizations were typically applied to homogeneous architectures, heterogeneous architectures were introduced to improve energy and performance using application-specific processor optimizations. Kumar et al. [13] introduced a single-ISA heterogeneous multi-core architecture where system software evaluated resource requirements and selected the best core for good performance while minimizing energy during runtime. The proposed system achieved more than a 3x reduction in energy with only an 18% performance penalty. In [12], Kumar et al. used a hill-climbing search heuristic to design the single-ISA heterogeneous architecture used in [13], which produced performance improvements as high as 40%, was within 5% of the optimal (best performance), and searched 14% of the design space.

B. Single-core Cache Tuning

Hardware configurable caches enable cache tuning by varying cache parameters during runtime. Motorola M*Core's [15] *way designation* allowed cache ways to be configured as instruction only, data only, unified, or shut down. Zhang et al.'s [28] highly configurable L1 cache offered configurable cache size via way shut down, a configurable line size by fetching multiple physical cache lines to increase the logical line size, and configurable associativity via way concatenation (Fig. 1).

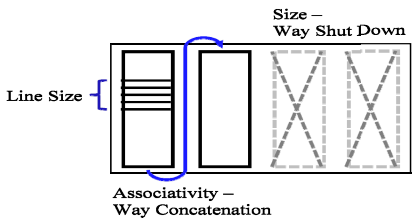

Fig. 1. Highly configurable cache architecture proposed by Zhang et al. [28]

The application designer can either determine optimal cache parameters during design time or the cache parameters can be determined dynamically during runtime. Design time cache tuning can require significant effort in terms of simulation setup and run time and may be inappropriate for highly dynamic applications. Alternatively, runtime cache tuning requires no application designer effort and is appropriate for highly dynamic applications because runtime cache tuning can react to actual input stimulus, environmental conditions, and application phase changes [7][11].

Runtime cache tuning determines the optimal cache configuration by executing the application in each explored cache configuration for a period of time. During this design space exploration, several inferior, non-optimal cache configurations may be explored before the optimal or near optimal configuration is determined. Since previous work shows that these inferior configurations can consume an exorbitant amount of energy and introduce performance overheads as compared to executing in a base cache configuration [8], it is imperative to reduce the number of inferior configurations explored. Tuning heuristics reduce the number of inferior configurations explored and find the optimal or near optimal configuration while searching only a fraction of the design space.

Zhang et al. [27] introduced an L1 cache tuning heuristic that searched configurations in order of their impact on energy, resulting in 40% average energy savings, which was within 7% of the optimal, by searching only 5 of 18 possible configurations. Zhang's impact-ordered, single-core L1 cache tuning heuristic tunes the cache size followed by the line size and finally the associativity. During cache tuning, in order to isolate the effects of a single cache parameter, one cache parameter is explored while the other parameters are held constant. For example, during size tuning, the line size and associativity are held constant at their smallest values while the cache size is explored from smallest to largest, which minimizes cache flushing and misses incurred while switching configurations. The heuristic increases the cache size as long the size increase results in a decrease in energy consumption. This process continues until there is an increase in energy consumption or the largest cache size is reached, wherein the cache size is then fixed at the lowest energy size. The line size and associativity are tuned in a similar way.

TCaT [9], which extended Zhang's impact ordering heuristic to two levels of cache, interleaved the L1 and L2 exploration, and included a final associativity adjustment. TCaT searched 6.5% of over 400 possible configurations, found configurations within 3% of the optimal, and achieved 53% energy savings on average. ACE-AWT [10] increased the L2's configurability using way designation, searched 0.2% of over 18,000 possible configurations, found configurations within 1% of the optimal cache configuration, and achieved

62% energy savings on average. However, these works did not consider multi-core systems.

III. RUNTIME DUAL-CORE DATA CACHE TUNING

Results in Section IV-B will show that dual-core data cache tuning can achieve an average energy savings of 25%, with energy savings as high as 50% (Fig. **5** (a)) for the optimal configurations. To evaluate a naïve adaptation of a state-of-the-art single-core cache tuning heuristic to a dual-core system, we applied Zhang's et al.'s [27] single-core L1 tuning heuristic to both processors in the dual-core system and found that Zhang's heuristic was not sufficient for dual-core cache tuning. Zhang's heuristic resulted in configurations that increased the energy consumption by as much as 26% as compared to the optimal dual-core configuration and achieved less than 1% average energy savings as compared to a base dual-core configuration. Since these results revealed that existing heuristics are not appropriate for runtime dual-core cache tuning, we developed a dual-core cache tuning heuristic to achieve energy savings by efficiently exploring a fraction of the design space.

Fig. 2 depicts our dual-core cache tuning heuristic – the **C**onditional **P**arameter **A**djustment **C**ache **T**uner (CPACT) – which determines the final cache configuration (optimal or near optimal) during runtime with no application designer effort. CPACT consists of an initial and a size adjustment step that determine initial parameter values and *conditions* that govern final tuning *actions* that perform additional parameter value adjustments to hone in on the optimal configuration. Conditions are critical to reducing the number of configurations explored as the governed actions are only performed when additional parameter value adjustment is highly likely to improve the final configuration's energy consumption. By minimizing the number of configurations explored, CPACT also minimizes the energy and performance overhead penalties incurred during design space exploration when the application is executed in inferior configurations.

Prior to execution, CPACT initializes each processor's data cache to an 8 KB direct-mapped cache with a 16 byte line size – the smallest configuration. CPACT's initial step applies impact-ordered cache tuning on both processors' data caches simultaneously, which reduces the total *tuning time* (design exploration time) and number of dual-core configurations explored as compared to sequentially exploring each core's cache while holding the other core's cache fixed. Note that this reduction in tuning time will become more pronounced as the number of cores in the system increases. While holding the line size and associativity constant at their smallest values, CPACT increases the size from 8 KB to 64 KB, in increments of powers of two, until there is no decrease in energy. CPACT similarly tunes the line size and then associativity.

Since the cache size has the largest impact on energy consumption and the cache size can be greatly affected by the actual line size and associativity values, CPACT performs a size adjustment step after the initial step tunes the associativity (i.e., tune the cache size, line size, associativity, and then the cache size again). Typically, CPACT's initial step chooses a larger size than the optimal size to compensate for the small line size and associativity, therefore the additional size adjustment step begins with the size determined during the initial step and decreases the size until the smallest size is reached or there is no decrease in the energy consumption.

Finally, CPACT conditionally adjusts the line size and associativity again. Similar to the initial step's line size and associativity exploration, these adjustments increase the parameter values until the largest size is reached or there is no decrease in energy consumption. Condition 1 evaluates the energy results of the size adjustment step. If the size adjustment step decreased the energy consumption, action 1 performs a final adjustment of the line size followed by the associativity. Alternatively, if the size adjustment step increased the energy consumption, action 2 evaluates one additional configuration with the cache size halved and the line size doubled from the configuration found during CPACT's initial impact ordered tuning step.

Condition 2 evaluates action 2's energy results. If action 2 decreased the energy consumption, action 3 performs a final adjustment on the line size and associativity. If condition 2 determines that action 2 increased the energy consumption, action 4 halts cache tuning and no additional parameter adjustment is necessary.

Once the final configuration is determined, the benchmark is executed in that final configuration for the remainder of the benchmark's execution. However, CPACT can be combined with phase classification and configuration techniques ([6][11][22]) to tune for each execution phase.

In our experiments, we model a dual-core system where each L1 data cache uses Zhang's highly configurable cache [28]. The physical total size of each data cache is 64 KB, however, the data cache is composed of 32, 2 KB banks that can be shut down and/or concatenated to vary the size and

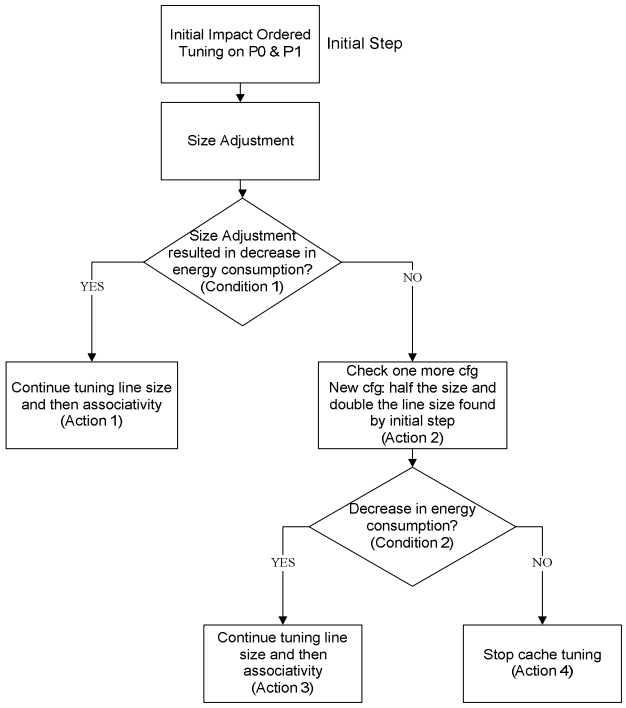

Fig. 2. The Conditional Parameter Adjustment Cache Tuner (CPACT)

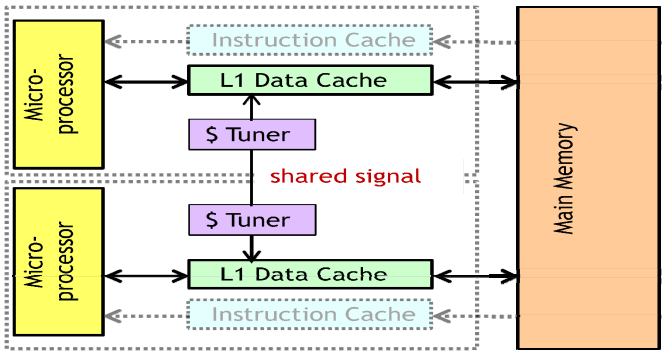

Fig. 3. Architectural layout of dual-core system

```
total energy = energy_P0 + energy_P1
for each processor (P0 and P1):
energy = dynamic_energy + static_energy + fill_energy +
         writeback_energy + CPU_stall_energy
dynamic_energy = dL1_accesses * dL1_access_energy
static_energy = ((dL1_misses * miss_latency_cycles) +
         (dL1_hits * hit_latency_cycles) +
         (dL1_writebacks * writeback_latency_cycles)) *
         dL1_static_energy
fill_energy = dL1_misses * (linesize / wordsize) *
         mem_read_energy_perword
writeback_energy = dL1_writebacks * (linesize / wordsize)
         * mem_write_energy_perword
CPU_stall_energy = ((dL1_misses * miss_latency_cycles) +
         (dL1_writebacks * writeback_latency_cycles)) *
         CPU_idle_energy
```

Fig. 4. Energy model used for the dual-core system

associativity, thus creating different configurations [28]. To enable runtime cache tuning, each data cache is coupled with a private hardware cache tuner that executes the CPACT algorithm. A shared signal between these tuners enables a tuner to signal the other tuner when the initial tuning step has completed (Fig. **3**). This signal is critical to the quality of the configured cache as our experimental results revealed that without coordinating the start of the size adjustment step, CPACT does not find the optimal configuration for some benchmarks. Previous work showed that the hardware overhead of a similar-purposed cache tuner is minimal [27].

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

We determined CPACT's energy savings using 11 SPLASH-2 multithreaded benchmarks (2 SPLASH-2 benchmarks were not evaluated due to the benchmarks' long execution times) [25] on the SESC simulator [19] for a dual-core system. For each processor, we varied the L1 data cache size from 8 KB to 64 KB, the line size from 16 bytes to 64 bytes, and the associativity from direct-mapped to 4-way [25] resulting in a design space of 1,296 possible cache configurations. In SESC, the L1 instruction cache and L2 unified cache were fixed at 64 KB, 4-way set associative cache with a 64 byte line size and 256 KB, 4-way set associative cache with a 64 byte line size, respectively. SESC has a 4 instruction issue width and allows out of order execution. The integer multiplication and division functional units and the floating point multiplication and division functional units have latencies of 4, 12, 2, and 10 cycles, respectively. All other functional units have a single cycle latency.

Fig**.** 4 depicts the dual-core energy model (adapted from a single-core model [27]) used to calculate the data cache hierarchy's and processor's (henceforth referred to as the *system*) energy consumption given a particular cache configuration. Our energy model considers the dynamic access energy and static energy for the L1 data (*dL1*) cache, main memory fetch energy for a cache miss (*fill_energy)*, main memory write energy for a cache write back (*mem_write_energy_perword*), and processor stall energy during a cache miss (*CPU_stall_energy)* for each processor. We gathered *dL1_misses*, *dL1_hits*, and *dL1_writebacks* using SESC. CACTIv6.5 [18] provided the dynamic cache energy dissipation and main memory energy per word for 90nm technology. We assumed static energy per clock cycle for each data cache configuration as 25% [2] of that configuration's dynamic access energy and the processor idle energy (*CPU_idle_energy*) as 25% of the processor' active energy [23] of the MIPS32 M14K processor [17]. We assumed a 2 cycle hit latency and a 30, 60, and 90 cycle miss penalty for reading and writing off chip memory for the 16, 32, and 64 byte line sizes, respectively.

To calculate energy savings for a given configuration, we normalize the system's energy consumption with the given configuration to that of a system where each processor's cache is set to a ***base configuration*** of a 64 KB, 4-way set associative L1 cache with a 64 byte line size. Our base configuration is set to the largest size since caches not specialized for specific applications are typically set to the largest configuration to maximize performance. We also evaluated the system's performance (total number of cycles needed for execution) normalized to the base system's performance.

We evaluated CPACT's energy savings and performance impacts with respect to several evaluation aspects. First, we calculated CPACT's energy savings and performance impacts when executing the entire benchmark in the optimal configuration as compared to the fixed base configuration. Next, we quantified the ineffectiveness of the naïve adaptation of Zhang's et al.'s [27] single-core L1 tuning heuristic (i.e., CPACT's initial step's impact-ordered tuning for dual-core systems). Finally, we evaluated CPACT's final configuration's effectiveness at improving energy savings beyond the impact-ordered tuning heuristic via the size adjustment step and full conditional parameter adjustment options. Thus, we calculated the percentage difference between the energy consumed when executing the benchmark in the optimal configuration and executing the benchmark in the three configurations determined at different *evaluation points* during CPACT's execution – after the initial step, after the size adjustment step, and at the final configuration (summarized in Table **I**).

To simulate run-time cache tuning in SESC, we defined a tuning interval of 500,000 cycles (i.e., the data cache configuration is changed after every 500,000 cycles until the CPACT algorithm is complete). The SPLASH-2 benchmark suite consists of computational kernels and complete applications [25]. Our tuning interval is enough cycles to execute a small SPLASH-2 kernel, but since the benchmarks' execution times varied, we normalized the benchmarks' execution times by looping the slower benchmarks several times. CPACT changed the data cache configuration of each processor at the beginning of each tuning interval.

While CPACT explores the cache configurations, the benchmarks are executed in several inferior configurations, each for one tuning interval. Once the final configuration is determined, the benchmark is executed in that final configuration for the remainder of the benchmark's execution. Each benchmark is run to completion for every configuration explored by CPACT, and the energy and performance in total cycles are calculated using our energy model (Fig. 4). The *final_energy_per_cycle* for each benchmark for each configuration is calculated. We use the *final_energy_per_cycle* to determine the energy consumed during each tuning interval and during execution in the final configuration. CPACT's final reported energy therefore includes the energy consumed by all cache configurations explored as well as the energy consumed while executing in the final configuration.

The reported execution time included the stall cycles (100 cycles) incurred between tuning intervals. During these stall cycles, energy consumption was calculated, the next cache configuration was chosen, the cache configuration was changed, and the cache contents were flushed (if necessary). Therefore, CPACT's performance overhead compared to executing the entire benchmark in the optimal configuration is defined as the *(number of configurations explored – 1) * number of tuning stall cycles* where the *number of configurations explored* is reported in Table I. Section IV-C shows that the average overhead is 3%. CPACT's final reported energy also includes processor stall energy (*CPU_idle_energy*) consumed during these stall cycles. CPACT's energy overhead is defined as the processor stall energy consumed during the tuning stall cycles and the additional energy consumed when inferior configurations are executed during exploration. Section IV-C shows that this overhead is no more than 2% compared to executing the entire benchmark in the optimal configuration.

B. Maximum Attainable Energy Savings

Fig. 5 (a) and (b) depict the energy savings and performance, respectively, for the optimal energy configuration for a single- and a dual-core system normalized to the system's respective base configurations. Data cache tuning in the dual-core system achieved an average energy savings of 25%, with energy savings as high as 50% for *radiosity*. The average energy savings for the dual-core system is comparable to single-core energy savings (25%) across the same benchmarks (Fig. 5 (a)). One motivation for switching from single- to dual-core systems is that dual-core systems can be more energy efficient because the application is decomposed across both processors with each processor consuming a smaller amount of energy. Our results showed that the dual-core system consumed less energy than the single-core system by as much as 10% for *ocean-non*, and by applying dual-core cache tuning, energy consumption is reduced even further.

With respect to performance, cache tuning in the dual-core system imposed an average performance penalty of 5%, with a maximum performance penalty of 14% for *water-spatial,* which also achieved 41% energy savings (for comparison, [13] showed an average performance penalty of 18% for their proposed heterogeneous multi-core system). Another motivation for switching from single- to dual-core systems is to improve the performance without increasing processor frequency. As expected, all benchmarks in our experiments required fewer cycles to complete execution on the dual-core system as compared to the single-core system. On average, the optimal dual-core cache configuration required only 61% of the number of cycles needed to complete execution on a single-core base system (Fig. 5 (b) dual-core (optimal) % of cycles), or a 1.64x speedup.

C. CPACT Results and Analysis

CPACT achieved an average of 24% energy savings as compared to the base configuration (Fig. 5 (a)) and explored at most 13 out of 1,296 possible configurations, equivalent to 1% of the design space. CPACT found the optimal configuration for 10 out of 11 benchmarks and resulted in a configuration within 1% of the optimal configuration for the remaining benchmark.

Although CPACT found 10 of 11 benchmarks' optimal cache configurations, the energy savings achieved by CPACT can be lower than the energy savings achieved by executing the entire benchmark in the optimal configuration. This discrepancy is due to the energy overhead of executing the benchmark in inferior, much higher energy configurations, and processor stall energy incurred while configurations are changed during design space exploration. Results showed that the energy overhead during design space exploration was small, with the largest overhead decreasing energy savings by only 2% for *radix* as compared to executing the entire benchmark in the optimal configuration. However, even with the energy consumption penalty, using CPACT, *radix* still achieved 30% energy savings compared to the base configuration (Fig. 5 (a)).

Executing benchmarks in inferior configurations and stalling execution between tuning intervals also incurred a performance penalty. On average, CPACT incurred an 8% performance penalty, a 3% increase in performance penalty compared to executing only in the optimal configuration (Fig. 5 (b)). Additionally, on average, CPACT required only 62% of the number of cycles needed to complete execution on a single-core base system (Fig. 5 (b) CPACT % of cycles), or a 1.60x speedup. This shows that the overhead of CPACT is very small and that, even with the added stall cycles, CPACT achieves speedups comparable to dual-core optimal speedups.

Table I depicts the percentage difference (*% DIFF*) between the energy consumed when executing the benchmark in the optimal configuration and executing the benchmark in the three configurations chosen by CPACT at the different evaluation points. Table I also reports the total number of configurations explored by CPACT (*# CFG EXP*).

For most benchmarks, CPACT's initial step was unable to determine the optimal cache size. Note that CPACT's initial step is equivalent to applying Zhang's original tuning heuristic [27]. The configuration found by the initial step achieves an average of 1% energy savings compared to the base cache, and can consume as much as 26% more energy when compared to the dual-core optimal configuration (Table I). To determine the final configuration, CPACT explored 11 configurations on average (*# CFGS EXP*), while after the initial step, only 8

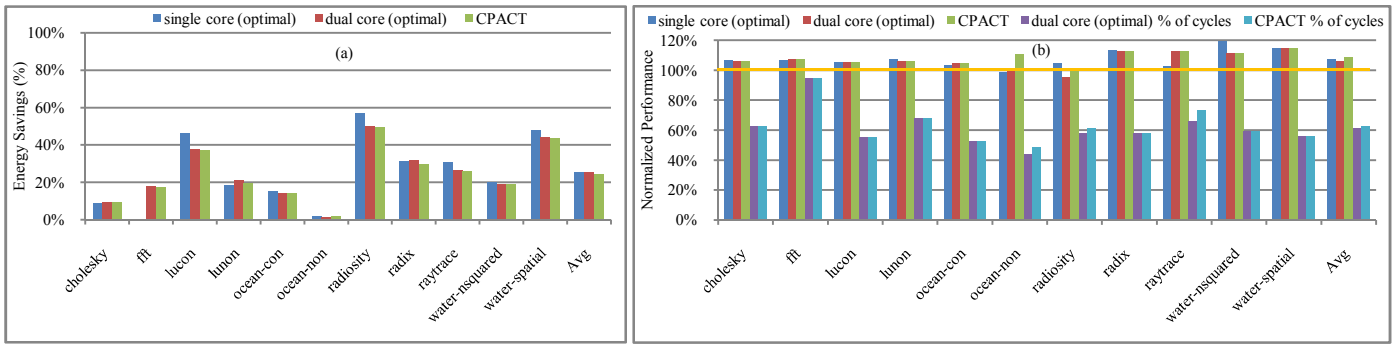

Fig. 5. (a) Energy savings for the optimal (lowest energy) cache for single- and dual-core systems and CPACT's final configuration applied to a dual-core system as compared to their respective base configurations. (b) Performance (measured in cycles) normalized to their respective base configurations for the optimal cache for single- and dual-core systems, CPACT's performance, and percentage of execution cycles needed by the dual-core system compared to the execution cycles needed by the single-core system (% of cycles), for the dual-core optimal configuration and CPACT.

configurations were explored on average. This means that, on average, to find the final configuration, CPACT required 300 additional stall cycles between tuning intervals as compared to applying just the initial step. However, we point out that: 1) 300 cycles is very small compared to the SPLASH-2 benchmarks, which execute for up to 80 million cycles; 2) the performance penalty of CPACT is only 3% compared to executing in the optimal dual-core configuration; and 3) the energy savings for the initial step are not enough to justify the reduced performance overhead.

During the first cache size tuning step of CPACT's initial step, the optimal cache size was bypassed since the line size and associativity were fixed at 16 bytes and direct-mapped, respectively, which did not necessarily reflect the optimal line size and associativity. For example, for *lunon* the 64 KB direct-mapped cache with a 16 byte line size consumed less energy than the 32 KB direct-mapped cache with a 16 byte line size for both processors, therefore 64 KB was chosen as the size. However, the 64 KB 2-way cache with a 32 byte line size configuration found after the initial step consumed more energy than the optimal configuration (32 KB 4-way cache with a 64 byte line size). The optimal configuration was not found by the initial step since the remaining configurations with a 32 KB cache size were not explored. However, including the size adjustment step was beneficial for 6 of the 11 benchmarks, resulting in configurations that were as much as 22% closer to the optimal configuration.

Although the size adjustment step had a large impact on the

Table I

PERCENTAGE DIFFERENCE BETWEEN THE ENERGY CONSUMED BY THE OPTIMAL CONFIGURATION AND CPACT AFTER THREE EVALUATION POINTS (*% DIFF*), AND NUMBER OF CONFIGURATIONS EXPLORED BY CPACT (*# CFGS EXP*).

benchmark	CPACT			
	after initial step	after size adjustment	final configuration	
	% diff	% diff	% diff	# cfgs exp
cholesky	11%	8%	0%	12
fft	6%	0%	0%	12
lucon	15%	15%	0%	9
lunon	26%	17%	0%	13
ocean-con	0%	0%	0%	8
ocean-non	2%	2%	0%	13
radiosity	22%	10%	1%	12
radix	1%	0%	0%	11
raytrace	22%	0%	0%	13
water-nsquared	0%	0%	0%	9
water-spatial	0%	0%	0%	9
Avg	10%	5%	0%	11

results, final line size and associativity adjustment was required to find the optimal (or near optimal) configuration for the benchmarks. *Radiosity, lucon,* and *lunon* were particularly difficult benchmarks to tune. *Radiosity* is a graphics program with unstructured access patterns to irregular data structures, which reduced spatial locality and complicates the analysis of the impact of cache parameters and runtime cache tuning [25]. Unlike the graphics programs with irregular data structures, both LU (Lower triangular matrix, Upper triangular matrix) Decomposition benchmarks (*lucon* and *lunon*) are decomposed into regular blocks [25]. When the cache size increases from 8 KB to 64 KB, the miss rate remains nearly constant, which makes cache tuning difficult, but larger sizes (128 KB and 256 KB) cause a 90% decrease in the miss rate due to the size of these applications' working sets. However, CPACT did not explore these larger sizes as they would not have been chosen as the optimal size due to the large size's high dynamic energy per access and static energy per cycle.

CPACT's final line size and associativity adjustments address both of these issues and produced final configurations much closer to the optimal for all benchmarks. If the size adjustment step decreased the energy consumption (condition 1, Fig. 2), a final adjustment of the line size followed by the associativity improved energy savings (action 1, Fig. 2). This applies to benchmarks such as *radiosity* and results in the final configuration being within 1% of the optimal configuration.

Alternatively, if the size adjustment step increased the energy consumption, one additional configuration with the cache size halved and the line size doubled from the configuration found after CPACT's initial step is explored (action 2, Fig. 2). If action 2 decreased the energy consumption, a final adjustment of the line size and associativity is required for improved energy savings (condition 2, action 3, Fig. 2). This applies to benchmarks such as *lucon* where the cache size chosen after the initial step is double that of the optimal cache size, but additional energy savings are not revealed when the size is tuned again in the size adjustment step given the line size and associativity chosen after the initial step is complete.

However, if the additional configuration with the cache size halved and the line size doubled from the configuration found after CPACT's initial step increases energy consumption (i.e., condition 2 determines that action 2 increased the energy consumption in Fig. 2), cache tuning is halted and no additional

parameter adjustment is necessary. This applies to the benchmarks such as *raytrace* where the configuration chosen after the size adjustment step was the optimal configuration, thus eliminating unnecessary configuration exploration for these benchmarks.

D. Cross-Core Data Decomposition Analysis

We evaluated the benchmark's optimal configurations to determine the effects of core-interactions and data coherence. Results revealed that the benchmarks typically required the same data cache configuration for both processors due to the benchmark's SIMD (single instruction multiple data)-like data decomposition across the processors where each processor executes the same functions on different data sets. However, three benchmarks *fft, radiosity,* and *raytrace*, required heterogeneous configurations with different associativities and/or line sizes.

Evaluating the benchmarks that required heterogeneous configurations revealed that core-interactions did affect the data cache configurations. In these benchmarks, one processor (arbitrarily referred to as P0) does significantly more work than the other processor (arbitrarily referred to as P1). For example, P0 may perform tasks that would not be necessary in a single-core system [25], such as building large data structures and distributing the data to reduce communication during execution, and data aggregation at the end of execution.

With respect to data coherence, we modified SESC to identify cache misses due to data sharing (coherence misses) and observed that for *fft, radiosity,* and *raytrace*, coherence misses attributed to as much as 17% of the overall cache misses for a 64 KB cache, even though the benchmark suite consists of multithreaded applications that are optimized to reduce inter-processor communication and data sharing [25]. As expected, the number of coherence misses increases as the cache size increases since larger caches are more likely to store data needed by another processor. Increasing the cache size to 128 KB and 256 KB increased the percentage of coherence misses for the three benchmarks (as high as 29% for *raytrace*) and revealed additional benchmarks with coherence misses (as high as 11% for *water-nsquared*). However, the 128 KB and 256 KB cache sizes are not explored by CPACT since these large cache sizes are never optimal configurations due to their high dynamic and static energy consumptions.

Although the coherence misses increase with the cache size, benchmarks with significant coherence misses do not necessarily choose a small cache size as the optimal configuration. Analysis revealed that the optimal cache size is dictated by the size of the working set and is not affected by data sharing and coherence misses for the cache sizes used in our experiments. Both *fft's* and *raytrace's* optimal sizes were 32 KB even though the number of coherence misses in the 32 KB cache was 5x larger than the 8 KB cache. As the cache size increased, the increase in coherence misses was offset by the decrease in total cache misses due to the ability to store more of the working set in the larger cache.

In summary, our analysis has revealed that core-interactions have a much greater effect on the cache configurations than data coherence does. However, data coherence does have an effect on cache configurations and we observed that the three benchmarks that required heterogeneous configurations were the benchmarks with data sharing among the processors while benchmarks with no data sharing chose the same configuration for both processors.

Although the SPLASH-2 benchmark suite uses data decomposition [25], the SPLASH-2 benchmarks can be classified into two types based on our core-interaction analysis of the dual-core system: 1) benchmarks with very little or no data sharing where coherence misses make up less than 1% of total misses and 2) benchmarks with data sharing. Some of our preliminary work shows that this classification is true for 4-core systems. Furthermore, for the benchmarks that exhibit data sharing and coherence in both the dual- and 4-core systems, one processor, the coordinating processor, performs pre- and post-execution processing while the other processors, the working processors, perform SIMD-like work. By extending our core-interaction analysis to systems with more than two cores, we project that this trend will continue for 4-, 8-, and 16-core systems.

We predict that, for benchmarks with no data sharing, CPACT may only need to be applied to a single processor while the other processors remain in a fixed configuration. Once the single processor is tuned, the remaining processors can be set to CPACT's final configuration without requiring tuning. For benchmarks that exhibit data sharing, cache tuning may only need to be done for two processors: the coordinating processor and one working processor, wherein the one tuned working processer can convey the optimal configuration to the other working processors without having to perform cache tuning on every processor. Reducing the number of cores that need tuning reduces the energy and performance overhead of cache tuning, which will become very important as the number of cores in the system increases.

Finally, previous work showed that instruction cache tuning can save as much energy as data cache tuning [8][9], therefore we expect to see an average of 25% energy savings gained when the instruction caches are tuned as compared to a system using a base instruction cache configuration. Therefore, similarly to data cache tuning, CPACT may only need to be applied to a single processor after which the remaining processors can be set to CPACT's final configuration since the instruction cache does not exhibit sharing. Our future work will explore what coordination mechanisms will be required between data and instruction cache tuning, such as whether or not the instruction caches can be tuned simultaneously with the data caches or will the caches need to be tuned sequentially.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we present a **C**onditional **P**arameter **A**djustment **C**ache **T**uner (CPACT) that achieves average data cache energy savings of 24%. CPACT finds the optimal configuration for 10 out of 11 benchmarks and a final configuration within 1% of the optimal configuration for the remaining benchmark by searching, at most, 13 out of 1,296 configurations. Benchmark data decomposition analysis revealed that core-interactions have a greater effect on the optimal configurations than data coherence. Future work

includes extending CPACT to *n*-core systems and incorporating instruction cache and level two cache tuning.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation (CNS-0953447). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] J. H. Anderson, J. M . Calandrino, and U. C. Devi, "Real-time scheduling on multicore platforms," in Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, April 2006.

[2] J. Butts, and G. Sohi, "A static power model for architects," in Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture (MICRO 33).

[3] J. A. Brown, R. Kumar, and D. Tullsen, "Proximity-aware directory-based coherence for multi-core processor architectures," in Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures, June 2007.

[4] J. Chang, and G. Sohi, "Cooperative caching for chip multiprocessors," in Proceedings of the 33rd Annual international Symposium on Computer Architecture, June 2006.

[5] J. Chang and G. Sohi, "Cooperative cache partitioning for chip multiprocessors," in Proceedings of the 21st Annual international Conference on Supercomputing, June 2007.

[6] A. Dhodapkar, and J. Smith, "Comparing program phase detection techniques," MICRO 2003.

[7] L. Eeckhout, H. Vandierendonck, K. De Bosschere, "Workload design: selecting representative program-input pairs," in Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2002.

[8] A. Gordon-Ross and F. Vahid, "A self-tuning configurable cache," in Proceedings of the 44th annual Design Automation Conference, DAC '07.

[9] A. Gordon-Ross, F. Vahid, and N. Dutt, "Automatic tuning of two-level caches to embedded applications," in Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1, February 2004.

[10] A. Gordon-Ross, F. Vahid, and N. Dutt, "Fast configurable-cache tuning with a unified second-level cache," in Proceedings of the 2005 international Symposium on Low Power Electronics and Design, ISLPED '05.

[11] A. Gordon-Ross, J. Lau, and B. Calder, "Phase-based cache reconfiguration for a highly-configurable two-level cache hierarchy," in Proceedings of the 18th ACM Great Lakes Symposium on VLSI, GLSVLSI '08.

[12] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in Proceedings of the 15th international Conference on Parallel Architectures and Compilation Techniques, PACT '06.

[13] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance," in Proceedings of the International Symposium on Computer Architecture, June 2005.

[14] C. Liu, A. Sivasubramaniam, and M. Kandemir, "Organizing the last line of defense before hitting the memory wall for CMPs," in Proceedings of the 10th international Symposium on High Performance Computer Architecture, February 2004.

[15] A. Malik, W. Moyer, and D. Cermak, "A low power unified cache architecture providing power and performance flexibility," in Proceedings of the International Symposium on Low Power Electronics and Design, ISLPED 2000.

[16] A. Merkel, and F. Bellosa, "Memory-aware scheduling for energy efficiency on multicore processors," in Proceedings of the 2008 Conference on Power Aware Computing and Systems.

[17] MIPS32 M14K http://www.mips.com/products/cores/32-64-bit-cores/mips32-m14k/

[18] N. Muralimanohar N. P. Jouppi, "Cacti6.0: A tool to model large caches," COMPAQ Western Research Lab, 2009.

[19] P. M. Ortego and P. Sack, "SESC: SuperESCalar Simulator," http://iacoma.cs.uiuc.edu/~paulsack/sescdoc/ Dec. 2004.

[20] S. Park, W. Jiang, Y. Zhou, and S. Adve, "Managing energy-performance tradeoffs for multithreaded applications on multiprocessor architectures," *SIGMETRICS Perform. Eval.* Rev. 35, 1.

[21] E. Seo, J. Jeong, S. Park, and J. Lee, "Energy efficient scheduling of real-time tasks on multicore processors," *IEEE Trans. Parallel Distrib. Syst*. 19, 11, November 2008.

[22] X. Shen, Y. Zhong, and C. Ding, "Locality phase prediction," in Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, 2004.

[23] A. Shrivastava, E. Earlie, N. Dutt, and A. Nicolau, "Aggregating processor free time for energy reduction," in Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, CODES+ISSS '05.

[24] G. E. Suh, L. Rudolph, and S. Devadas, "Dynamic partitioning of shared cache memory," J. Supercomput. 28, 1 (Apr. 2004), 7-26.

[25] S. C. Woo, M. Ohara, et al., "The splash-2 programs: Characterization and methodological considerations," in *Proceedings of the International Symposium on Computer Architecture*, pp. 24–36, June 1995.

[26] T. Y. Yeh and G. Reinman, "Fast and fair: data-stream quality of service," in Proceedings of the 2005 international Conference on Compilers, Architectures and Synthesis For Embedded Systems, CASES '05.

[27] C. Zhang, F. Vahid, and R. Lysecky, "A self-tuning cache architecture for embedded systems," ACM Trans. Embed. Comput. Syst. 3, 2 (May. 2004), 407-425.

[28] C. Zhang, F. Vahid, and W. Najjar, "A highly-configurable cache architecture for embedded systems," in the Proceedings of the 30th Annual International Symposium on Computer Architecture, June 2000.