Dynamic Scheduling for Reduced Energy in Configuration-Subsetted Heterogeneous Multicore Systems

Mohamad Hammam Alsafrjalani and Ann Gordon-Ross*
Department of Electrical and Computer Engineering University of Florida (UF), Gainesville, FL, USA
*Also with the NSF Center for High Performance Reconfigurable Computing (CHREC) at UF
E-mail: mha8@ufl.edu, ann@ece.ufl.edu

***Abstract*—Heterogeneous and configurable multicore systems provide hardware specialization to meet disparate application hardware requirements. However, effective multicore system specialization can require a priori knowledge of the applications, application profiling information, and/or dynamic hardware tuning to schedule and execute applications on the most energy efficient cores. Furthermore, even though highly disparate core heterogeneity and/or highly configurable parameters with numerous potential parameter values result in more fine-grained specialization and higher energy savings potential, these large design spaces are challenging to efficiently explore. To address these challenges, we propose a novel configuration-subsetted heterogeneous and configurable multicore system, wherein each core offers a small subset of the design space, and propose a novel scheduling and tuning (SaT) algorithm to efficiently exploit the energy savings potential of this system. Our proposed architecture and algorithm require no a priori application knowledge or profiling, and incurs minimal runtime overhead. Results reveal energy savings potential and insights on energy tradeoffs in heterogeneous, configurable systems.**

Keywords-Heterogeneous cores, configurable caches, energy optimizations, design space subsetting, embedded systems, multicore architectures, scheduling.

I.INTRODUCTION

Reducing energy consumption is a key design goal in all computing domains and devices. Since an application's hardware requirements significantly impact the system's energy consumption, hardware resources can be specialized to meet these requirements for energy efficiency. Hardware *specialization* can be achieved using multicore systems, wherein each core can have different hardware parameter values, such as voltage, clock frequency, cache size/associativity, etc. The specific configuration of these parameters' values that most closely adheres to the application's requirements while achieving design goals (e.g., lowest energy consumption, highest performance, or a tradeoff) constitutes the application's *best* configuration.

Heterogeneous multicore systems, such as the ARM big.LITTLE [4] or OMAP3530 [25], have specialized/configurable cores to meet different application requirements, and thus have excellent energy savings potential. However, determining the core's specific configurations and scheduling applications to the core with the application's best configuration requires accurately identifying the application's requirements, which can be done using application profiling information (e.g., level one (L1) cache miss rate, pipeline stalls, cycles per instruction (CPI), etc.).

Application profiling can be done statically during design time or dynamically during runtime. Static profiling requires a priori knowledge of the applications, but can be leveraged to determine the best core configurations based on these requirements, thus offering greater energy savings potential at the expense of an inflexible, static system. However, due to this application-specific specialization, this method is only suitable for static, known applications. Dynamic profiling increases system flexibility, which is necessary for general purpose systems, by profiling unknown applications during runtime to determine the application's best configuration. However, since the applications are not known a priori, the cores' configurations must be generally suitable for any application, and thus may not closely adhere to each application's specific requirements, which decreases the energy savings potential. Additionally, runtime profiling incurs profiling overhead (e.g., performance/energy) while profiling the applications.

Even though the specialized cores are heterogeneous, offering different configurations for different application requirements, these configurations are *fixed*, thus the total number of different configurations (e.g., the design space) is very small, which limits potential adherence to design goals [1] (e.g., energy savings in our work). Alternatively, heterogeneous, configurable multicore systems have cores with *runtime* configurable parameters, which increases the design space and thus potential adherence to design goals based on varying, unknown application requirements.

Parameter configurability in heterogeneous, configurable multicore systems increases the design goal adherence potential, but increases the runtime overhead, since after scheduling an application to a core, a tuning algorithm must tune the core's configurable parameters to the application's best configuration. If the cores have disparate design spaces, the application should be scheduled only to the core that offers the application's best configuration, which could force applications to stall if the core with the best configuration is not available. If all cores offer the same design space, scheduling is simplified since applications can be scheduled to any core, then that core can be tuned. However, core tuning is challenging [11] and can introduce large tuning overhead when executing/evaluating applications in inappropriate, non-best configurations [11], especially for highly configurable cores with many parameters and parameter values (e.g., NM where N is the number of cores and M is the number of core configuration).

Whereas this vastly increased design space increases potential design goal adherence, offering the same design space on all cores is not necessary. Prior work showed that applications with similar execution requirements belonged to similar application domains, and had similar, but not necessarily the same, best configurations [3]. Based on these similarities, the design space can be subsetted to a small fraction of the complete design space, while still offering best, or near-best, configurations for each application.

Based on these observations, we conjecture that the cores' collective design space can be reduced to offer a much smaller subset of the complete design space, wherein the subset can be specialized to contain different configurations that are amenable

to different application domains. During runtime, application profiling determines the application's domain, and the application is scheduled to the core that offers a configuration for that domain.

In this work, we propose, to the best of our knowledge, the first heterogeneous, configurable multicore system architecture with domain-specific core configuration subsets and an associated scheduling and tuning (SaT) algorithm. Whereas this fundamental architecture and approach is applicable to any configurable parameters, we focused on configurable caches due to the cache's large contribution to system energy consumption [28] and configurable caches' energy savings potential [3][6][10][28]. SaT's key contribution is the ability to save energy with no designer effort in a highly configurable system without any a priori knowledge of the applications.

II.RELATED WORK

Much prior work has focused on hardware specialization using heterogeneous multicore systems and configurable cores, for example, and various application scheduling and tuning algorithms have been proposed to harness the benefits afforded by these specialization methods. In this section, we discuss selected hardware specialization methods that relate most closely to our proposed work, in addition to state-of-the-art tuning algorithms.

A. *Hardware Specialization*

Kumar et al. [16][17] used a four-core heterogeneous multicore system consisting of cores from the same processor family, but each core contained different, fixed parameters, such as issue-width, branch prediction, etc. An oracle dynamically scheduled applications to cores for reduced energy based on the application's specific requirements and energy/performance that were evaluated offline. Silva et al. [7] used heterogeneous multicore systems with different cache sizes and statically scheduled applications to cores for reduced cache miss rates and increased performance. However, these prior works focused on general purpose desktop computing, which is less energy constrained than embedded systems.

To complement heterogeneous multicore systems, much research has focused on configurable cores with configurable parameters, such as issue-width [9], dynamic clock rate [2], dynamic voltage and frequency scaling [13], and caches [3][10][28]. Even though these works have shown good design goal adherence in isolation, these works did not consider amalgamating heterogeneous and configurable cores into a holistic system.

To evaluate the potential benefits of combining these specialization methods, Adegbija et al. [1] evaluated and compared heterogeneous multicore systems to configurable multicore systems for embedded systems, and showed that this combination maximized energy savings. However, that work used an exhaustive search to find the best combination of heterogeneous cores and core configurations to reduce energy consumption, which incurs significant tuning overhead for systems with a large design space.

B. *Application Scheduling and Core Tuning*

Luo et al. [18] studied static application scheduling in heterogeneous multicore systems for reduced energy consumption in battery-operated embedded systems. The proposed method profiled the battery's discharging characteristics to determine an application's best core. Using a system with a processor core and digital signal processing (DSP) core, Kim et al. [15] modified a static-priority-based scheduling algorithm. Typically, static-priority scheduling algorithms cause high priority applications to block all accesses to the DSP while executing on the processor, even if the application is not currently using the DSP. The authors proposed a modified algorithm that allowed lower priority applications to execute on the DSP when the DSP was idle using a remote procedure call. Van Craeynest et al. [26] dynamically scheduled applications in a heterogeneous multicore system for increased performance using a method that estimated the performance change for executing an application on a different core based on the application's performance on the current core. Since the method estimated performance, there was no profiling overhead. Instead of scheduling the entire application to a core, Joao et al. [12] used a finer grained approach that partitioned the application into threads and scheduled these threads to cores for increased performance.

Alternatively to scheduling applications to disparate heterogeneous cores, researchers also tuned configurable hardware to adhere to disparate application requirements. Prior work [5][19][28] leveraged configurable caches to reduce energy and/or increase performance, however, many of these works required designer effort to determine the best configuration. Chen et al. [6] and Gordon-Ross et al. [10] alleviated designer effort using tuning algorithms that leveraged specialized cache hardware, called organizers/tuners/orchestrators, to automatically search the design space and dynamically tune the configurable cache to determine the best configuration. Even though these methods required no designer effort, during tuning, the application executed in inappropriate, non-best configurations, which could impose significant tuning overhead [10].

To reduce tuning overhead, Gordon-Ross et al. [11] presented non-intrusive oracle hardware that ran in parallel with the cache to evaluate all possible cache configurations simultaneously and determine the application's best configuration. However, even though this oracle eliminated cache tuning overhead, the oracle hardware imposed significant energy overhead, and thus was only feasible for systems with very persistent applications

While prior work motivated and demonstrated the potential for tuning to reduce energy consumption, most prior work tuned only a single core and did not evaluate tuning benefits for multicore systems, or the additional tuning overhead incurred when considering intra-core dependencies (e.g., shared data). Rawlins et al. [22] applied single core cache tuning concepts to multicore systems, and considered intra-core dependencies introduced by a single instruction multiple data (SIMD) model. The authors determined that cores with similar cache miss rates also had similar best cache configurations. Thus, to reduce tuning overhead, the cores were grouped based on the cores' cache miss rate similarity, and only one core from each group was tuned and that core's best configuration was conveyed to all other cores in the same group.

III.HETEROGENEOUS, CONFIGURABLE MULTICORE SYSTEM ARCHITECTURE

A. *Architecture*

Figure 1 depicts our sample quad-core heterogeneous, configurable multicore architecture that we evaluated in our experiments. Each core has private, dedicated L1 data and instruction caches. Since cache size has the largest impact on energy consumption [28], and to limit the cores' design spaces, the cores' caches have disparate, fixed cache sizes and the caches' line sizes and associativities are configurable.

	16B	32B	64B
2K_1W	c_1	c_7	c_{13}
4K_1W	c_2	c_8	c_{14}
4K_2W	c_3	c_9	c_{15}
8K_1W	c_4	c_{10}	c_{16}
8K_2W	c_5	c_{11}	c_{17}
8K_4W	c_6	c_{12}	c_{18}

Table 1: Cache configuration design space.

Table 1 depicts our cache configuration design space, which corresponds to our sample architecture and the requirements of our embedded system experimental applications (Section 5). Columns represent the line sizes in bytes (B) and rows represent the sizes in Kbytes (K) and associativity (W). Each column-row intersection denotes a unique configuration c_n. We use c_{18} for application profiling, since c_{18} is the best-performance cache on average over all of our experimental applications, and thus minimizes the profiling overhead.

The cores' subsets contain configurations from Table 1, and the union of the cores' subsets' configurations is specialized to meet different domain-specific application requirements. Based on application profiling and our prior evaluations, we determined that small domain-specific configuration subsets attained nearly the same energy savings as the complete design space. Since these evaluations showed that three domain-specific subsets were sufficient to meet disparate application requirements, we consider three domain-specific subsets. Each domain subset meets a given range of application profiling information (e.g., cache miss rate ranges), and during runtime, SaT (Section 4) profiles the applications and uses the profiling information to determine the applications' domains. Since the cores' subsets are specialized to meet application domain-specific requirements, the domain dictates the applications' best subset, and hence, best cores.

Given the union of these domain-specific configuration subsets, we grouped the configurations based on the configurations' cache sizes (i.e., three given Table 1), and mapped each group to the corresponding core with the same cache size (i.e., cores one through three in Figure 1). Thus, the core's mapped configurations comprise that core's subset, which restricts the core's configuration design space. Any core with c_{18} can be used as a profiling core, however, if c_{18} is not part of any domain-specific subset, c_{18} can be easily included in at least one of the subsets. Our experiments show that since domain-specific subsets typically contain at least one configuration with a cache size of 8K, the overhead to include c_{18} in any subset without c_{18} requires only a few additional control bits. Since this mapping only requires three of the four cores, the fourth core's subset replicates the core with the largest cache size (i.e., 8 Kbyte). Even though the fourth core could replicate any of the other cores' subsets, replicating the largest cache size is pessimistic with respect to energy savings, and provides a second profiling core.

Even though our work evaluates this specific system architecture and configuration design space, and three application domains, our fundamental methodologies are generally applicable to any arbitrary number of cores, configurations, and application domains, and increasing any of these parameters would increase the potential energy savings.

B. *Software Support*

We integrate SaT into the operating system's scheduler, which enqueues applications into a ready queue for SaT to schedule. SaT stores the application profiling information in the process (i.e., application) control block (PCB) [24], along with the application's identification number (ID), execution status (i.e., ready, executing, terminating, etc.), arrival time, etc.

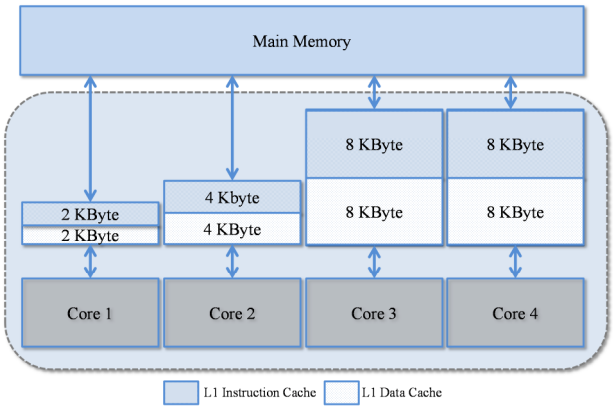


Figure 1: Sample heterogeneous, configurable multicore system architecture.

The application profiling information contains the application's cache statistics, such as the L1 cache miss rate, which is obtained from the core's hardware counters [4], and is used to calculate the application's energy and performance (Section 5), and to determine the application's best core. The profiling information also stores the application's best core and best configuration after these have been determined. Since during scheduling, the application's best core may not be available (Section 4.2), to facilitate scheduling to the best alternative core, the profiling information also stores a history of the energy consumption and performance for all of the prior cores/configurations that the application has executed on.

The storage requirement for each application profile is:

$$\boldsymbol{m = ceiling|\log_2(a * r * c * e * t)|} \qquad (1)$$

bits, where a, r, and c represent the number of anticipated applications that will execute on the system, the number of cores, and total number of configurations across all cores, respectively, and e and t represent the energy and performance in number of cycles, respectively, for each application execution per core/configuration. Since most embedded systems typically run a fixed set of persistent applications, m requires limited number of bits, which can be stored in main memory and requires no additional hardware storage unit.

IV. APPLICATION SCHEDULING AND TUNING (SAT) ALGORITHM

A. *Overview*

Algorithm 1 depicts SaT's operational flow, which has two stages: *scheduling*, which determines the application's best core, effectively determining the application's best cache size, and *tuning*, which configures the core to the application's best cache line size and associativity.

Since SaT is part of the operating system's scheduler, SaT is invoked at a predetermined time interval [24]. On each invocation, SaT processes applications in the ready queue in first come first served (FCFS) order, and attempts to schedule the application such that the total energy is minimized. Since dynamic energy is the primary energy contributor, SaT first attempts to schedule the application to the application's best core/configuration. If the best core is busy and there are idle, non-best cores, to reduce wasted idle energy, SaT evaluates the energy

advantage for scheduling the application to a non-best core as compared to leaving the application in the ready queue to wait for the application's best core, thus wasting idle energy as idle cores are unused.

If SaT is unable to schedule an application or determines that it is energy advantageous for an application to wait for the application's best core, the application remains in the ready queue and SaT attempts to schedule the next application in the ready queue. If SaT successfully schedules an application, the application's profiling information is updated when the application terminates.

B. SCHEDULING STAGE

In the scheduling stage, SaT checks the application's profiling information. If there is no profiling information, the application is executing for the first time and SaT schedules the application to any arbitrary idle profiling core (i.e., core 3 or 4 in our sample architecture (Section 3.1)), removes the application from the queue, profiles the application, and updates the application's PCB. If no profiling core is idle, SaT leaves the application in the ready queue, since attempting to schedule the application without any profiling information may force the application to execute with an extreme configuration. Extreme configurations are configurations that are so ill-suited to the application's requirements that the configuration causes a significant increase in the energy consumption [10], and thus should be avoided.

If there is profiling information for the application, the best core is known. If the best core is idle, SaT schedules the application to this core and removes the application from the ready queue.

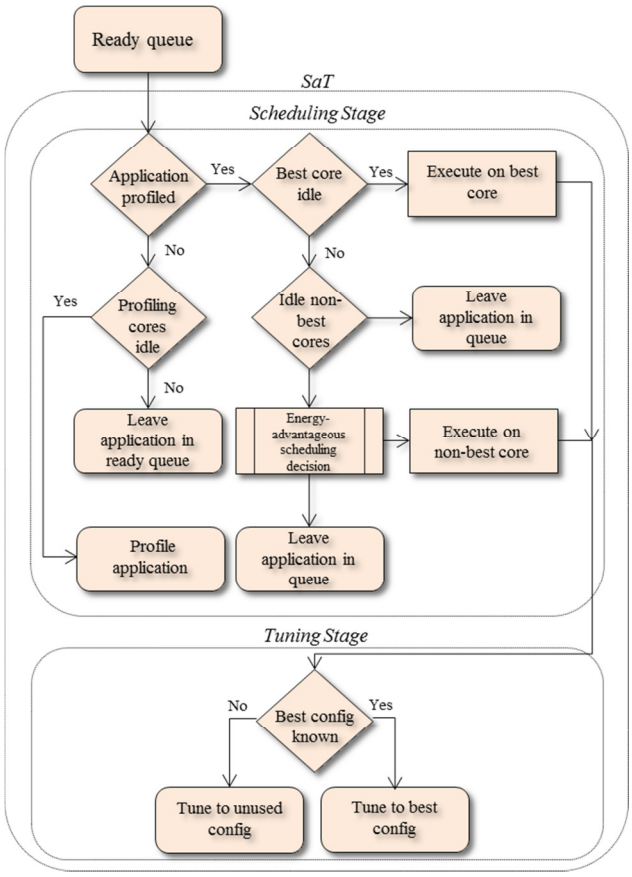
If the application's best core is not idle, but other non-best cores are idle, SaT evaluates an energy-advantageous scheduling decision using the application's profiling information. This evaluation determines if it is energy-advantageous to schedule the application to a non-best core or leave the application in the ready queue to wait for the best core to be idle.

The energy-advantageous scheduling decision is a Boolean value *Th* calculated by:

$$\boldsymbol{Th = E_{dyn}(a_2, c_1) + E_{dyn}(a_1, c_1) + E_{idl}(a_2, c_2) > E_{dyn}(a_1, c_2)} \quad (2)$$

where a_1 is the application being scheduled, a_2 is the application executing on a_1's best core c_1, c_2 is an idle, non-best core, $E_{dyn}(a_x, c_x)$ is the dynamic energy expended by executing a_x on c_x, and $E_{idl}(a_2, c_2)$ is the idle energy expended by idle c_2 while waiting for c_1 to finish executing a_2. Essentially, if the dynamic energy $E_{dyn}(a_1, c_2)$ to execute the application being scheduled a_1 on an idle, non-best core c_2 is less than the idle energy expended $E_{idl}(a_2, c_2)$ by the idle, non-best core c_2 plus the dynamic energy $E_{dyn}(a_2, c_1)$ of the busy, best core c_1 to both complete execution of a_2 and execute a_1, a_1 is scheduled to the idle, non-best core c_2. Otherwise, SaT leaves a_1 in the ready queue.

Th can only be calculated if both applications a_1 and a_2 have previously executed in all configurations on cores c_1 and c_2 (i.e., the applications' best configurations' are known). If any core configuration energy consumption is unknown, SaT optimistically assumes *Th* is true and schedules a_1 to execute on c_2, and removes the application from the ready queue, which promotes throughput since the application may have to wait for a long period of time, and models prior scheduling algorithms [26]. Additionally, this scheduling enables SaT to populate the energy consumption and performance history for additional core configurations, which enhances future scheduling decisions.



Algorithm 1: Scheduling and tuning (SaT) algorithm for heterogeneous, configurable multicore system.

C. TUNING STAGE

Once an application is scheduled to execute on a core, either best or non-best, SaT enters the tuning stage. If the application's profiling information contains energy consumptions for all of the core's configurations, SaT directly tunes the core to the application's best (i.e., lowest energy) configuration for that core.

If there is any core configuration with unknown energy consumption, then the application's best configuration on that core is not yet known and SaT must execute the application with one of the unknown configurations. Since all configurations must be executed, SaT arbitrarily chooses an unknown configuration from the core's subset, and tunes the core to that configuration, and updates the application's profiling information with this configuration's energy consumption. We note that since the core subsets are small (four configurations in our experiments), this exhaustive exploration is feasible, however, for advanced systems with larger subsetted design spaces per core, search heuristics can also be used [27].

V. EXPERIMENTAL SETUP

We evaluated SaT using our proposed architecture (Figure 1) with 34 embedded applications: sixteen (complete suite) from the EEMBC Automotive application suite [8], six from Mediabench I [21], and twelve from Motorola's Powerstone applications [19], which represent a diversity of application requirements [26].

Since embedded system applications are typically persistent, we replicated the applications in the ready queue. Each application is identified with an ID from one to 34, and we generated a series of

E(total) = *E(sta) + E(dyn)*

E(dyn) = *cache_hits * E(hit) + cache_misses * E(miss)*

E(miss) = *E(off_chip_access) + miss_cycles * E(CPU_stall) E(cache_fill)*

Miss Cycles = *cache_misses * miss_latency + (cache_misses * (line_size/16)) * memory_band_width)*

E(sta) = *total_cycles * E(static_per_cycle)*

E(static_per_cycle)) = *E(per_Kbyte) * cache_size_in_Kbytes*

E(per_Kbyte) = *(E(dyn_of_base_cache) * 10%) / (base_cache_size_in_Kbytes)*

Figure 2: Cache hierarchy energy model for the L1 instruction and data caches.

1,000 IDs using a discrete uniform distribution. We modeled the application arrival times using a normal distribution centered at the mean and within one standard deviation of the average execution time of all applications using the base configuration.

To model common operating system schedulers [20][24], we invoked SaT every 2,000 cycles, which represents less than 1% of the average execution time of the applications using the base configuration.

Since many embedded systems do not have level two caches [1], and SaT's efficacy can be evaluated with L1 caches, our experimental architecture's (Figure 1) private, separate L1 data and instruction caches can be tuned independently and simultaneously. We used SimpleScalar to obtain cache accesses/hits/misses, and obtained off-chip access energy from a standard low-power Samsung memory. We estimated that a fetch from main memory took forty times longer than an L1 cache fetch, and the memory bandwidth was 50% of the miss penalty [11].

In order to directly compare to previous research [3][26], Figure 2 depicts our cache hierarchy energy model (similar to [3]) and we determined the dynamic energy using CACTI [23] for 0.18 um technology. Even though 0.18 um technology is a large technology, many embedded systems do not require cutting edge technologies. Furthermore, since SaT reduces the idle energy and the idle energy constitutes a larger percentage of the total energy as the technology size decreases (over 30% of the total energy in smaller technologies (e.g., .032 um) [14]), this technology gives pessimistic energy savings for SaT. We estimated the idle and static energies each as 10% of the dynamic energy [11] and the CPU stall energy as 20% of the active energy [3].

VI.EVALUATION METHODOLOGY

We compared SaT with prior configurable cache research [3][27] and scheduling algorithms [20][24] using three systems, denoted as system-1, system-2, and system-3. All systems had the same configurable heterogeneous multicore architecture (Figure 1), but used different scheduling algorithms. We compared the systems' energy consumptions by normalizing the energy consumption to a base system with all four cores configured to c_{18} that scheduled applications using round robin [20][24].

System-1 was modeled similarly to [3] and provided insights on the significance of wasted idle energy, and served as a near-optimal system for comparison purposes. System-1 assumed a priori knowledge of the applications' domains (i.e., no profiling overhead) and best configurations (i.e., no tuning overhead). System-1 only scheduled an application to the application's best core using the best configuration, and left the application in the ready queue if the application's best core was not idle, even if other, non-best cores were idle and wasting idle energy.

Alternatively, instead of requiring an application to wait for the application's best core to be available, the application can be scheduled to a non-best core, if available, which trades off saved idle energy for increased dynamic energy. System-2 modeled this performance-centric system, which maximizes throughput and core utilization. Similarly to system-1, system-2 had a priori knowledge of the applications' domains and best configurations. However, the overall energy implications of this performance-centric system are unclear. If there is an idle, non-best core, and the idle core's wasted energy while the application waits in the ready queue for the application's best core to be available is greater than the dynamic energy for executing the application on a non-best core, then system-2 consumes less energy than system-1. However, prior works have shown that non-best configurations, and thus non-best cores, can significantly increase the energy consumption [3], thus if the dynamic energy for executing the application with a non-best core is greater than the wasted idle energy expended while the application waits in the ready queue, then system-1 consumes less energy than system-2. Since this evaluation is highly dependent on the actual applications' best cores/configurations and the applications' arrival orders, our experiments consider myriad applications and the results were averaged over 1,000 application arrivals to capture an average case.

Finally, system-3 evaluated SaT's ability to achieve energy savings without any designer effort or a priori knowledge of the applications' domains or best core/configuration, and to give insights on idle and dynamic energy tradeoffs for different scheduling decisions. System-3 also provided insights on the significance of profiling and tuning overhead, which determines the feasibility of using SaT in general purpose and/or constrained embedded systems. SaT imposes profiling overhead while profiling the applications using the base configuration, which is not necessarily the best configuration and may incur large dynamic energy overhead, and since not all cores offer the base configuration, profiling can force applications to wait in the ready queue for a profiling core to be idle, and thus incurs idle energy overhead. Additionally, SaT imposes tuning overhead when exhaustively executing the applications on all core configurations, which forces applications to execute using non-best configurations, and thus incurs dynamic energy overhead. Designer effort and a priori knowledge of the applications' best configurations enables SaT to directly execute the applications with the applications' best configurations, thereby eliminating profiling and tuning overhead. We evaluated SaT's profiling and tuning overhead by computing the energy difference between executing system-3 with and without a priori knowledge the of applications' domains and best configurations, and normalized this energy difference to the base system.

VII.RESULTS AND ANALYSIS

Figure 3 (a) and (b) depict the dynamic, idle, and total energy consumptions for system-1, -2, and -3 (system-3 results include profiling and tuning overhead) normalized to the energy consumption of the base system for the data and instruction caches, respectively. Values below/above 1 corresponds to less/more energy consumption that the base system.

Compared to the base system for the data and instruction caches respectively, system-1 reduced the dynamic energy consumption by 38.2% and 25.4%, but increased the idle energy consumption by 1916.4% and 155.6%, resulting in total energy savings of 11.6% and 20.8%. The idle energy increase suggests that the applications' best cores were not equally distributed across the

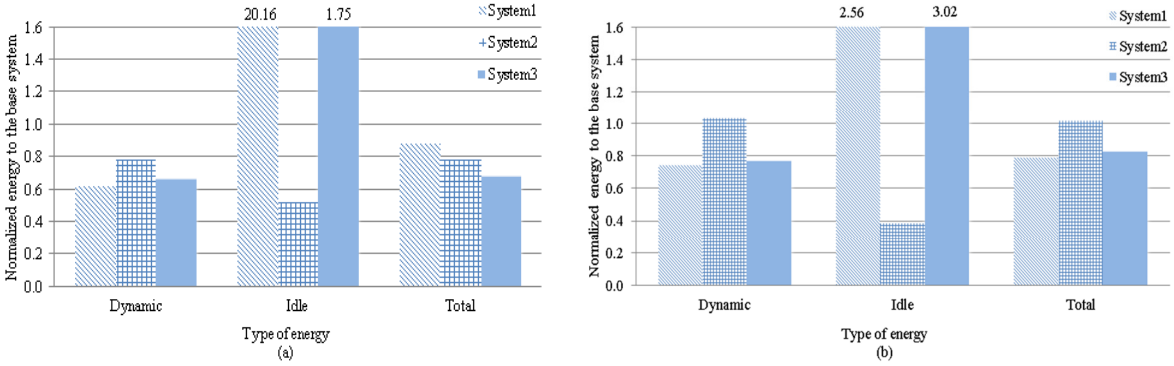


Figure 3: Energy of all systems normalized to the energy of the base system for the (a) data cache and (b) instruction cache.

cores' susbsets, which caused core bottlenecks while applications waited indefinitely for the applications' best cores to be available. Alleviating this bottleneck is difficult since a balanced distribution of the applications' best configurations across the cores is highly dependent on the actual applications that are executing. An alternate solution is to increase the number of cores, however, this solution will increase the system's total energy consumption, and still does not eliminate the potential for bottlenecks. As a result, systems with increased number of cores are more likely to waste more idle energy than to save dynamic energy. We expect that dynamic core shutdown could alleviate these idle expenditures, and is the focus of our future work.

Since system-2 was performance-centric and scheduled applications to any available core, best or non-best, system-2's cores were less likely to be idle and thus consumed less idle energy. System-2 reduced the idle energy compared to the base system and system-1 by 47.0% and 97.4%, respectively, for the data cache, and by 61.1% and 84.8% for the instruction cache, respectively. Compared to system-1, system-2 increased the dynamic energy by 27.9% and 38.3% for the data and instruction caches, respectively, which is expected since system-2 did not guarantee that applications executed on/with the applications' best cores/configurations. Compared to the base system, system-2 decreased the dynamic energy for the data cache by 21.0%, but increased the dynamic energy for the instruction cache by 3.19%. For the data and instruction caches respectively, system-2 decreased the total energy by 21.4% and increased the total energy by 1.5% as compared to the base system, and decreased the total energy by 11.2% and increased the total energy by 28.2% as compared to system-1.

The increases in dynamic and total energies for the instruction cache, as compared to the decreases in dynamic and total energies for the data cache, are attributed to the fact that system-2 ran more applications with extreme instruction cache configurations. Since our analysis showed that instruction caches tended to exhibit less miss rate and cache requirement variation as compared to data caches across different applications (prior work also showed that instruction cache subsets can be smaller than data cache subsets [27]), executing in a non-best instruction cache configuration causes a larger energy consumption increase due to the likelihood that a non-best instruction cache configuration is an extreme configuration. The increase in the instruction cache's total energy consumption with respect to system-1 suggests that the idle energy savings is not large enough to compensate for the increased dynamic energy consumption. Avoiding extreme configurations can reduce the dynamic energy increase, and thus the idle energy savings would reduce the total energy. We conjecture that process migration and process preemption can alleviate this increased dynamic energy for extreme configurations by migrating the process to a core with a different configuration subset, or by returning the application to the ready queue until the application's best core is available, However, both process migration and preemption incur performance overhead due to saving the process's context and requires hardware support to store and restore the process's context, which is beyond the scope of this paper and is part of our future work.

For the data and instruction caches respectively, system-3 increased the dynamic energy by 8.4% and 3.6% as compared to system-1, decreased the dynamic energy by 33.0% and 22.7% as compared to the base system, and decreased the dynamic energy by 15.2% and 25.1% as compared to system-2. System-3 had higher dynamic energy compared to system-1, since unlike system-1, system-3 did not always schedule applications to the application's best core.

For the data and instruction caches respectively, system-3 increased the idle energy by 74.9% and 202.4% as compared to the base system, and by 229.9% and 678.4% as compared to system-2, since system-3, unlike the base system and system-2, left applications in the ready queue until the applications' best cores were available based on Equation (2). System-3 consumed 91.3% less idle energy than system-1 for the data cache, but consumed 18.3% more energy for the instruction cache. Our analysis of system-1 revealed that the data cache's idle energy was much larger than the instruction cache's idle energy and thus system-3 was able to reduce more idle energy for the data cache as compared to the instruction cache. For the data cache, system-3 reduced the total energy by 31.6%, 22.6%, and 13.0% as compared to the base system, system-1, and system-2, respectively. For the instruction cache, system-3 reduced the total energy by 17.0% and 18.3% as compared to the base system and system-2, respectively, but increased the total energy by 4.8% as compared to system-1. System-3 decreased and increased the total energy for the data and instruction caches, respectively, as compared to system-1 due to the instruction cache's higher requirement variations. However, since our experiments considered 0.18um technology (Section 5), we surmise that SaT will have larger idle energy savings with smaller technologies.

Although system-3 increased the instruction cache energy as compared to system-1, system-3 outperformed system-1 with

respect to the data cache energy savings, and system-3 outperformed the base system, system-1, and system-2 with respect to both the instruction and data cache energy savings. Since system-3 outperformed system-1, which prioritized energy savings, and system-1, which was performance-centric, in 75% of the cases, system-3 can save energy in performance-centric systems and systems with low energy constraints. However, since the energy savings trades off performance [1], complete evaluation of the performance tradeoffs between system-1, -2, and -3 requires additional comparisons of the energy and performance, which is part of our future work.

Finally, analysis of system-3's profiling overhead revealed that a priori knowledge of the applications' domains and best configurations only provided minor energy improvements, and increased the energy savings for the data and instruction caches by 1.8% and 0.9%, respectively. The small overhead is attributed to application persistence such that the overhead is amortized over multiple executions. Since most embedded systems repeatedly execute persistent applications, SaT is amenable to general purpose systems. SaT's most significant contribution is appreciable energy savings with no designer effort with respect to application profiling, which broadens SaT's applicability and usability to any general purpose system.

VIII.CONCLUSION AND FUTURE WORK

Heterogeneous and configurable multicore systems provide hardware specialization to meet disparate application hardware requirements. However, heterogeneous multicore systems have small, fixed design spaces and require static or dynamic application profiling to select the best cores that most closely adhere to application hardware requirements. Configurable multicore systems provide flexible and larger design spaces at the cost of profiling and tuning overhead and designer effort. To adhere to application hardware requirements while minimizing profiling and tuning overhead, we propose, to the best of our knowledge, the first heterogeneous, configurable multicore system with application-domain specific configuration subsets and an associated scheduling and tuning (SaT) algorithm. Our results revealed average energy savings of 31.6% and 17.0% for the data and instruction caches, respectively, as compared to a base system, with only 1.8% and 0.9% profiling and tuning overhead.

Future work includes integrating additional energy savings techniques into SaT, such as dynamic core shutdown and dynamic voltage and frequency scaling. We also plan to study different subset-to-core mapping methodologies with increased number of cores to gain insight on the best per-core subset distributions and potential bottlenecks. Finally, to gain insight on the performance and energy tradeoffs, we will instrument our future experiments to collect performance statistics and perform energy-delay product analysis.

ACKNOWLEDGEMENT

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] Adegbija, T., A. Gordon-Ross. "Exploring the Tradeoffs of Configurability and Heterogeneity in Multicore Embedded Systems, " Int. Con. on Mobile Ubiquitous Comp., Systems, Services and Technologies (UBICOMM'13), Sept. 2013.

[2] Albonesi, D.H., "Dynamic IPC/clock rate optimization," Proc. of Int. Sym. on Computer Architecture, Jul. 1998

[3] Alsafrjalani, M. H., A. Gordon-Ross, and P. Viana. "Minimum Effort Design Space Subsetting for Configurable Caches, " Embedded and ubiquitous computing Design (EUC), May 2014

[4] ARM Ltd., big.LITTLE Technology, White Paper: http://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf

[5] Bahar, R.I., Albera, G., Manne, S., "Power and performance tradeoffs using various caching strategies," Proc. of Int. Sym. on Low Power Electronics and Design, 1998

[6] Chen, L., Zou, X., Lei,J., Liu, Z., "Dynamically Reconfigurable Cache for Low-Power Embedded System," 3rd Int. Conf. on Natural Computation, Aug. 2007.

[7] de Abreu Silva, B., Cuminato, L.A., Bonato, V., "Reducing the overall cache miss rate using different cache sizes for Heterogeneous Multi-core Processors," Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on , vol., no., pp.1,6, 5-7 Dec. 2012

[8] EEMBC. The Embedded Microprocessor Benchmark Consortium http://www.eembc.org/benchmark/automotive_sl.php, Sept. 2013

[9] Folegnani, D.; Gonzalez, A., "Energy-effective issue logic," Proc. on Inter Sym. on, Computer Architecture 2001

[10] Gordon-Ross, A., Vahid, F. "A Self-Tuning Configurable Cache" IEEE Design Automation Conference, Jul. 2007

[11] Gordon-Ross, A., Viana, P., Vahid, F., Najjar W., Barros, E. "A One-Shot Configurable-Cache Tuner for Improved Energy and Performance" IEEE/ACM Design, Automation and Test in Europe, Apr. 2007

[12] Joao, José A., Aater Suleman, M., Mutlu, O., Patt, N., "Utility-based acceleration of multithreaded applications on asymmetric CMPs. SIGARCH, " Comp. Arch. News 41, 3, Jun. 2013

[13] Ishihara, T., Yasuura, H., "Voltage scheduling problem for dynamically variable voltage processors," Proc. on Int. Sym. on Low Power Electronics and Design, Aug. 1998.

[14] Kahng, A.B., Seokhyeong Kang, Rosing, T.S., Strong, R., "Many-Core Token-Based Adaptive Power Gating," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.32, no.8

[15] Kim, K., Kim, D., Park, C., "Real-time scheduling in heterogeneous dual-core architectures, " International Conference on Parallel and Distributed Systems, 2006

[16] Kumar, R., Farkas, K.I., Jouppi, N.P., Ranganathan, P., Tullsen, D.M., "Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction," Microarchitecture, Int. Symp on, Proceedings. 36th pp.81,92, 3-5 Dec. 2003

[17] Kumar, R., Tullsen, D., N. Jouppi, N., Ranganathan, P., "Heterogeneous chip multiprocessors, " Computer, vol. 38, Nov. 2005

[18] J. Luo and N. Jha, "Battery-aware static scheduling for distributed real-time embedded systems, " Design Automation Conference, 2001, pp. 444-449.

[19] Malik, A., Moyer, B., Cermak, D., "A low power unified cache architecture providing power and performance

flexibility," Proc. of the 2000 Int. Symp. on Low Power Electronics and Design

[20] Mauerer, W., "Process Management and Scheduling", Professional Linux Kernel Architecture, 1st Ed., Wrox, Oct., 2008, ch 2, pp 37.

[21] Mediabench Suite, http://euler.slu.edu/~fritts/mediabench/

[22] Rawlins, M., Gordon-Ross, A., "An Application Classification guided Cache Tuning Heuristic for Multi-core Architectures," Trans. on IEEE Computers 2012

[23] Reinman, G., and N.P. Jouppi, COMPAQ Western Research Lab: CACTI2.0: An Integrated Cache Timing and Power Model, 1999.

[24] Silberschatz, A., "Processes", Operating System Concept, 9th Ed., Wiley, Dec. 2012, ch 3, pp 111

[25] Texas Instruments, OMAP3530 Applications Processors Datasheet: http://www.ti.com/lit/ds/sprt656/sprt656.pdf

[26] Van Craeynest, K., Jaleel, A., Eeckhout, L., Narvaez, P., Emer, J., "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," Computer Architecture (ISCA), 2012 39th Annual International Symposium on , vol., no., pp.213,224, 9-13 Jun. 2012

[27] Viana, P., Gordon-Ross, A., Keogh, E., Barros, E., Vahid, F., "Configurable cache subsetting for fast cache tuning," Design Automation Conference, 2006

[28] Zhang, C., Vahid, F., Najjar, W., "A highly configurable cache architecture for embedded systems," Proc. of Int. Sym. on Computer Architecture, Jun 2003