computers

MDPI

Article

Scheduling and Tuning for Low Energy in Heterogeneous and Configurable Multicore Systems †

Mohamad Hammam Alsafrjalani [1,*] and Ann Gordon-Ross [2]

[1] Department of Electrical and Computer Engineering, University of Miami, Coral Gables, FL 33146, USA
[2] Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32608, USA; Ann@ece.ufl.edu
* Correspondence: Alsafrjalani@miami.edu
† This paper is an extended version of our paper published in Proceedings of the 12th IEEE International Conference on Embedded and Ubiquitous Computing, Milano, Italy, 26–28 Aug. 2014.

Received: 8 March 2018; Accepted: 11 April 2018; Published: 14 April 2018

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 incur minimal runtime overhead. Results reveal energy savings potential and insights on energy trade-offs in heterogeneous, configurable systems.

Keywords: heterogeneous cores; configurable caches; energy optimizations; design space subsetting; embedded systems; multicore architectures; scheduling

1. Introduction

Reducing energy is a key design goal for all computing domains including low-power embedded devices and high performance computing centers. To reduce energy, hardware resources, such as the number of cores and the core's configuration, must closely match, or be specialized to, the applications' hardware requirements such that both dynamic and idle energy are minimized. If hardware resources are excessive or very large (redundant core(s), a large cache memory, high clock frequency, etc.) idle energy of the unused resource is wasted. Alternatively, if the hardware resources are too small or scarce, applications require longer periods of time to execute and expend excessive dynamic energy.

Heterogeneous systems and configurable systems provide coarse- and fine-grained hardware specialization, respectively, that meet the applications' hardware requirements. Heterogeneous multicore systems such as the ARM big.LITTLE [1] or OMAP3530 [2] provide coarse-grained hardware specialization via disparate, fixed 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 trade-off) constitutes the application's best core. 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 [3] (e.g., energy savings in our work). Alternatively, configurable systems have cores with runtime configurable parameters, which increase the design space and thus potential adherence to design goals based on varying, unknown application requirements. However, to exploit adherence to the design goal, heterogeneous multicore systems and configurable systems impose profiling and tuning challenges, respectively.

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, dynamic profiling introduces three challenges: (1) runtime profiling incurs profiling overhead (e.g., performance/dynamic energy) while profiling the applications and expend idle energy by the other cores; (2) this overhead is tightly coupled to the degree of heterogeneity (i.e., higher heterogeneity typically requires more profiling/sampling); and (3) 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. Alternative to profiling, tuning is adjusting the hardware resources parameters to obtain a best configuration that most closely adheres to the application's hardware requirements. However, configurable systems impose large tuning overhead when executing/evaluating applications in inappropriate, non-best configurations [4], 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 configurations).

Dynamic profiling and tuning overhead can be alleviated if the system comprises a small degree of heterogeneous cores with a small set of tunable configurations, respectively. The key challenges are then determining a set of cores and the cores' constituent configurations. If the cores' heterogeneity is of a small degree, such that one application profiling is required to determine the best core with the configuration(s) that most closely adhere(s) to the application's hardware requirements, then the profiling overhead can be alleviated. Furthermore, if the number of configurations evaluated during tuning is narrowed to the most promising best or near-best configuration, then tuning overhead is alleviated. Since applications with similar execution requirements belonged to similar application domains, and have similar, but not necessarily the same, best configurations [5], 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. If these subsets are offered in a multicore system, such as each core is offering a distinct subset, then the multicore system offers adherence to disparate domains of applications, with a small degree of heterogeneity. To determine the application domain, and hence the core, only a single profiling execution of the application is required to determine the application's domain, and hence the application's best core. Furthermore, once the core is determined, tuning evaluates only the configurations of the subsets of that domain/core, and tuning overhead is alleviated.

In addition to alleviating dynamic profiling and tuning overhead, a key challenge is to reduce the idle energy expended by idle cores. If the best core for a domain remains idle due to the lack of applications of that domain, the cores could expend precious energy—a growing concern in recent technology [6]. One method to reduce idle energy is to power-gate idle cores (e.g., [7]). However, to avoid unpredictable performance behavior, power-gating an entire core requires a priori knowledge of applications' schedule, or a hardware-based oracle to monitor application memory access patterns to predict the performance of power-gating the core. Alternatively, idle energy can be saved if the cores are kept busy executing applications of other domains, when available.

Based on these observations, 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. Each core offers a distinct subset of configuration (heterogeneous), and can be tuned/specialized to a configuration of this subset (configurable).

Whereas this fundamental architecture and approach is applicable to any configurable parameters such as issue window (e.g., [8]) and reorder buffer (e.g., [9]), we focused on configurable caches due to the cache's large contribution to system energy consumption [10] and configurable caches' energy savings potential [8–11]. The key contribution of our architecture is providing close adherence to application's hardware requirements with low profiling and tuning overhead.

To exploit the system's ability to adhere to design goals, we propose an associated scheduling and tuning (SaT) algorithm. SaT dynamically profiles the applications to determine the applications' domains, and schedules the applications to the best core that provides the most suitable configuration. Once the application is scheduled to a core, SaT tunes that core's parameter to the best configuration that adheres to the application's hardware requirement. 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. Furthermore, SaT balances dynamic and idle energy by determining whether or not to halt or execute the application on an idle, non-best core. If executing an application on a non-best, idle core expends less dynamic energy than idle energy expended by leaving the core idle, SaT schedules the application to that non-best idle core.

We compared our system to a base system, prior work, and performance-centric system, all of which had a priori knowledge of the applications. Our results revealed our system can save up to 31.6% and 22.6% energy, as compared to the base system and prior work, respectively. Furthermore, our performance analysis revealed that SaT can outperform prior work and performance-centric systems in 50% of the cases, and that our system provided energy-delay product (EDP) within 10.9% of a performance-centric system.

The remaining of the paper is organized as follows: Section 2 discusses heterogeneous and configurable systems as hardware specialization approaches, and scheduling and tuning algorithms used with these systems, respectively. We describe our heterogeneous and configurable system design approach in Section 3, and our associated scheduling and tuning (SaT) algorithm in Section 4. Sections 5 and 6 provide our experiment setup and evaluation methodology, respectively. We discuss the results and analysis in Section 7 and conclude our work in Section 8.

2. Related Work

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

2.1. Hardware Specialization

Kumar et al. [12] 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 applications' specific requirements and the applications' offline profiling. Silva et al. [13] 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 (e.g., [8,9]), dynamic voltage and frequency scaling (e.g., [14,15]), caches (e.g., [10,11,16]), and data paths (e.g., [17]). Zhang et al. [10] designed a

configurable cache that has configurable size, line size, and associativity. Shutting-down/enabling cache ways reduced/increased the cache size, logical concatenation of cache ways configured the cache associativity, and fetching additional physical cache lines for larger, logical line sizes enabled line size configurability.

Furthermore, Viana et al. [5] demonstrated that configurable cores provide best and near-best configurations. When the design goals are relaxed, Ref. [5] demonstrated that the near-best configurations could substitute the best configurations. Using this finding, Alsafrjalani et al. [11] designed a configuration subsetting exploration method that determined and grouped best and near-best configurations based on applications' domains. The authors demonstrated that applications that belonged to the same domain shared the best and near-best subsetted configurations. 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. [3] 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.

Whereas these prior works saved dynamic energy by specializing the hardware resources to the application's hardware requirements, these works did not consider idle energy consumption of idle, non-busy components. To reduce idle energy consumption in on-chip static random access memory (SRAM), Powell et al. [18] architected a power gating technique that gate the supply voltage to unused SRAM cells to reduce idle energy. To reduce idle energy consumption of cache lines, Kaxiras et al. [19] turned off cache lines that spent pre-determined, idle number of cycles. Similarly, Zhou et al. [20] turned off only the data portion of the cache lines (not the tags), once the lines spent an idle period of time—dynamically adjusted. Alternative to turning off the cache lines, Flautner et al. [21] used a reduced-power technique that kept the cache lines turned on, at a reduced-power state. Whereas these prior works reduced the cache's energy consumption, these works did not evaluate a system level idle to dynamic energy trade-off evaluation.

To save idle energy, Jeong et al. [6] leveraged the power gating approach for power gating the entire core. On a cache memory miss, the core's status was saved and the core was put into a power-gated state while the data was fetched from lower-level memory. A hardware switch and oracle determined the cycles required for the data to be fetched from lower-level memory back to the core. A few cycles before the data was ready for the core, and the switch switched the core to active mode. Kahng et al. [7] extended the works of [6] to cores with out-of-order execution. Kahng's et al.'s work saved the status of the core's register and pipeline states on a cache miss and power-gated the core while the data was being fetched from lower-level memory. A few cycles before the data was available, the core's register and pipeline states were restored, and the core was put into an active mode.

Whereas the power gating techniques provided efficient solutions to reducing idle energy, these techniques relied on a pre-determined applications schedule, or a hardware oracle to monitor applications' memory access patterns and adjust the core's status. However, these contributions could be complementary to our approach, and we plan to incorporate power-gating policies into our SaT algorithm in future work.

2.2. Application Scheduling and Core Tuning

Luo et al. [22] 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. [23] 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. [24] 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. [25] used a finer grained approach that partitioned the application into threads and scheduled these threads to cores for increased performance. Das et al. [26] scheduled applications closer to the network controller, in order to alleviate inter-application interference and contention for data access in network-on-chip systems. Whereas these works provided novel mechanisms, these works did not consider energy-performance trade-offs with systems of high degree of heterogeneity or fine-grained configurations. Shelepov et al. [27] proposed a heterogeneity-aware scheduler that dynamically maps applications to the best cores. However, the scheduler required an offline application architectural signature, in order to estimate the performance of an application on a core during runtime.

Alternatively to scheduling applications to disparate heterogeneous cores, researchers also tuned configurable hardware to adhere to the disparate applications' requirements. Prior work [10,28–30] leveraged configurable caches to reduce energy and/or increase performance; however, many of these works required designers' efforts to determine the best configuration. Wang et al. [28] profiled the applications offline and stored the best configuration information in a lookup table to be used during runtime. To meet real-time deadlines, the system looked up the application's performance information in the lookup table and tuned the cache to the highest performance configuration.

To alleviate designer efforts, Chen et al. [31] and Gordon-Ross et al. [4] used 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 [4].

To reduce tuning overhead, Gordon-Ross et al. [32] 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). To extended single-core tuning to many-core systems, Rawlins et al. [33] 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. However, these works did not consider systems with heterogeneous and configurable cores. Alternatively, our work performs scheduling and tuning for our novel heterogeneous and configurable systems.

3. Heterogeneous, Configurable Multicore System Architecture

3.1. Overview

To design a multicore system that provides coarse- and fine- grained adherences, the cores of the system must comprise coarse- and fine-grained configurations. However, to limit redundant

configurations and reduce core hardware area, we map configurations to the cores based on the hardware area requirements. This section details our configuration selection and mapping methodology using an illustrative example of a quad-core system.

3.2. Selecting Coarse- and Fine-Grained Configurations

Since our approach focuses on configurable caches, we define coarse-grained configurations as configurations that have the highest impact on performance, and fine-grained configurations as configurations with lower impact on performance, as compared to disparate configurations. Prior work [10] demonstrated that cache size has the highest impact on performance, followed by line size and associativity. Given the energy-performance trade-off (higher performance requires higher energy), performance-impact ordering is highly probably applicable to energy as well; cache size has the highest impact on energy. Therefore, our coarse-grained configurations are those of distinct cache sizes, and fine-grained configurations are configurations with similar cache size but different line size and associativity.

Table 1 depicts the complete cache configuration design space, which corresponds to our sample architecture and the requirements of our embedded system's 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 select subsets of the complete design space using our subset selection methodology.

Table 1. Cache configuration design space.

	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}

Figure 1 depicts the design space subsetting for multi-domain subsets. The subset selection methodology takes in as inputs the number of domains, d, subset size, s, and the complete design space, C, and produces subsetted design space comprised of n subsets S *(note that $n = d$)*. For each d, the methodology selects the best subset S, comprised of s configurations out of C using a design space exploration methodology adapted from [11].

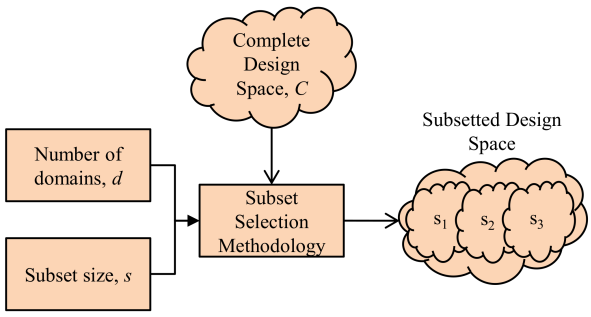

Figure 1. Design space subsetting for multi-domain subsets.

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.

Subsequently, we set the number of domains d to three, which corresponds to domains of applications with low, mid-range, and large cache hardware requirements; we use Table 1 as our complete design space, (C = 18); and we set the subset size, s, to four since a four-configuration subset is large enough to represent the complete design space of size 18 [5,11].

The subset selection methodology creates all of the configuration subsets, evaluate these subsets using application kernels and select the highest quality subset for each domain. First, the subset selection methodology creates all combinations of possible subsets out of the complete design space (3040, given C = 18 and s = 4). Once created, the methodology evaluates the quality of each subset using d sets of applications. These applications are kernels that represent common tasks in the anticipated applications—34 in our case (Section 5). To create d sets of kernel applications, the methodology classifies these applications hierarchically, using Euclidian distance of the applications' cache miss-rate c_{18}. The output of the classification is d classes of applications, grouped based on the similarity of the applications cache miss-rate—three classes in our case.

Finally, to select the highest quality subset for each domain, the methodology exhaustively evaluates all of the subsets with the applications of that domain, and select the subset that most closely adhere to the design constraint, as shown in Algorithm 1. The algorithm takes in as its inputs n configuration subsets, d, and kernel application-set per domain k_d, and returns the best subset per domain. For each domain, the algorithm executes all of the kernel application-set of that domain with each subset and calculates the average energy consumption of this execution. Once all of the subsets are evaluated with for this domain, the algorithm selects the subset that resulted in the lowest average energy consumption as the best subset for that domain.

We note that for much larger design spaces the design space subsetting for multi-domain subsets process could be prohibitively lengthy, and a design space exploration heuristic could be employed to speed up the process (e.g., [34]). For example, a design space that considers 18 cache memory sizes/line sizes/associativity, 24 pipeline depths (PD), 16 issue window (IW) sizes, and 10 voltage-frequency (VF) values will have 69,120 configurations, and 9.50×10^{17} subsets. The configurations could be subsetted independently (e.g., PD) or jointly (e.g., PD $\times$ VF) before they are used in our subset selection methodology.

Algorithm 1. Subset evaluation and selection.

Input: n configuration subsets: Sn;
Number of domains: d
Kernel application-set per domain: k_d
Output: Best subset per domain: S_1-S_d
For all ***d***
For all ***n***
Execute each application in k_d on each configuration in S_n
Calculate average energy consumption
End for
Select subset with lowest energy
End for
Return (S_1-S_d)
End

Once the methodology selects the constituent configurations of subsets S_1, S_2, and S_3, we map these subsets to a quad-core system, such that 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. For our setup, the algorithm selected subsets S_1, S_2, and S_3, with the configuration sets $\{c_1, c_2, c_7, c_{13}\}$, $\{c_3, c_9, c_{14}, c_{15}\}$, and $\{c_5, c_{11}, c_{12}, c_{18}\}$, respectively.

3.3. Mapping Subsets to Cores

To avoid hardware overhead, redundant configurations must not be mapped multiples times; however, best configurations must be distributed amongst the cores to alleviate possible performance bottleneck(s). Given n subsets of size s, mapping each subset to a core will require r cores, where $r \in [1, n]$. If $r = 1$, then all subsets are mapped into one core and that core becomes a performance bottleneck. Alternatively, if $r = n$, then each subset is mapped to a dedicated core and the potential bottleneck is alleviated. However, dedicating each subset to an individual core can result in redundant configurations and wasted hardware area, since the subsets could comprise similar configurations. For instance, if $c_{14} \in S_1$ and S_2, and S_3, then only one instance of c_{14} is mapped to the core with the cache size of c_{14}.

Furthermore, to reduce area overhead, the logical grouping of configurations (subsetting) must be disjoined from the physical implementation of these subsets on actual hardware. If distinct subsets contain configurations of similar coarse-granularity (e.g., same cache size), then these subsets will duplicate the hardware area requirements (e.g., three cores with the same cache size). To reduce this redundancy, configurations of similar coarse-granularity (e.g., cache size) are mapped to the core of the same cache size.

Figure 2 depicts our three subsets mapped to a quad core system. 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. 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 (c_{18} is the best-performance cache on average over all of our experimental applications, and thus minimizes the profiling overhead); 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.

Figure 3 depicts our sample quad-core heterogeneous, configurable multicore architecture based on our subset selection and mapping process. Each core has private, dedicated L1 data and instruction caches. Since cache size has the largest impact on energy consumption [31], and to limit the cores' design spaces, the cores' caches have disparate, fixed cache sizes and the caches' line sizes and associativities are configurable. To alleviate the hardware requirement for the cache tuner, we use a global hardware tuner (Section 3.4).

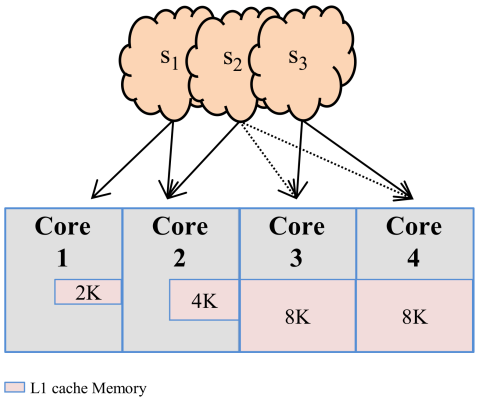

Figure 2. Mapping configurations of subsets S_1, S_2, and S_3 to a quad core system.

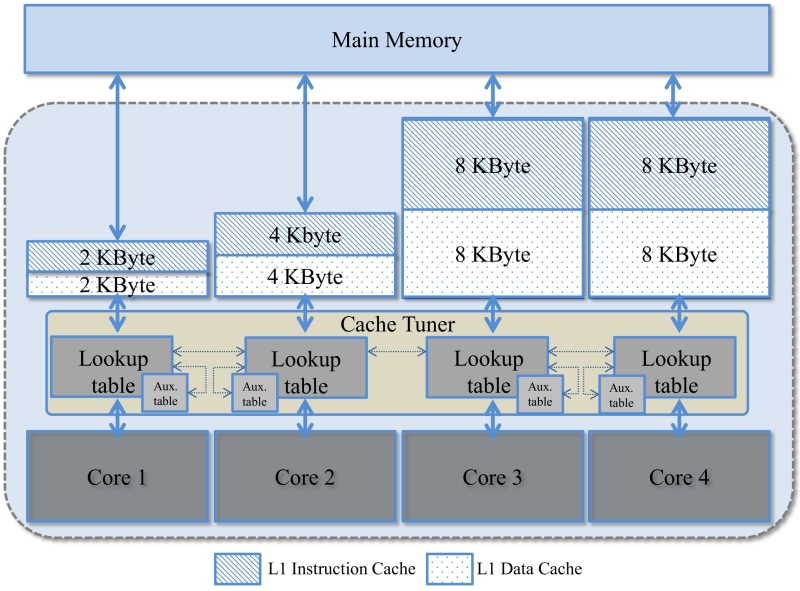

Figure 3. Sample heterogeneous, configurable multicore system architecture.

Our experiments show that since domain-specific subsets typically contain at least one configuration with a cache size of 8 K, 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. For instance, given a system comprising cores with configurable cache memory, PD, IW, and VF, mapping of the configuration could be based on single or joint configuration types. Furthermore, since cores with higher VF could benefit from larger cache memory, (e.g., ARM big.LITTLE [1]), the cache memory size-VF pairing (e.g., larger cache size paired with higher VF range) provides the system's heterogeneity; however, the cache line-size and associativity provides the system's configurability.

3.4. Hardware Support Requirements

To alleviate the hardware requirement for the cache tuner, we use a global hardware tuner that requires only the control logic bits, which are negligible as compared to the size of the caches (Equations (1) and (2)). The tuner reads in the application's tuning information stored in the tuner's lookup table and adjusts the caches' line size and associativity based on our SaT algorithm (Section 4). To enable portability to any operating system scheduling, we integrate SaT's finite state machine (FSM) into the global hardware tuner, which is triggered by the operating system's scheduling invocation.

The tuner stores the application profiling information in lookup tables, along with the application's identification number (ID), execution status (i.e., ready, executing, terminating, etc.), arrival time, etc. 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 [1], and is used to calculate the application's energy and performance (Section 4.2), 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 been executed on.

The storage requirement for each application profile is:

$$m = ceiling|\log_2(a \times r \times c \times e \times t)| \quad (1)$$

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.

To limit area overhead, this complete tuning information is only stored for an application during scheduling and tuning. After the best configuration is determined, only that configuration is retained in a smaller auxiliary lookup table. To enable scalability to an arbitrary number of applications, we utilize a least recently used replacement policy to govern the auxiliary table's information. Thus, given a applications, the total tuning information storage requirement M in bits is:

$$M = \log_2(a) \times log_2(c) \times m, \quad (2)$$

which imposes only 4.7% area overhead in embedded processors such as the MIPS M4k [35].

4. Application Scheduling and Tuning (SaT) Algorithm

4.1. Overview

Figure 4 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. To save dynamic energy, SaT schedules the application to the best core, and, to reduce idle energy, SaT schedules the application to an idle non-best core, if the best core is busy.

SaT is invoked when an application is placed in the ready queue by the operating system. 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 (e.g., all cores are busy) 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.

We note that, to avoid starvation, we can employ a preemption mechanism into SaT that ensures fair resource sharing amongst applications. However, since our study does not aim for hard real-time systems, we plan to incorporate such techniques in future work.

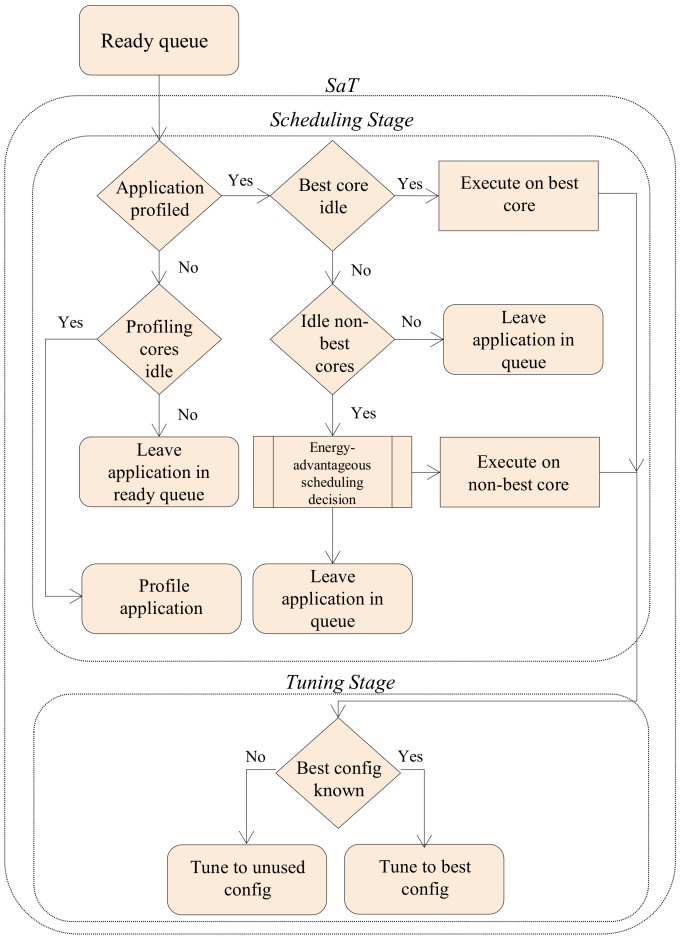

Figure 4. Scheduling and tuning (SaT) algorithm for heterogeneous, configurable multicore systems.

4.2. 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., cores 3 or 4 in our sample architecture (Section 3.3)), removes the application from the queue, profiles the application, and updates the application's profiling information in the lookup table. 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 [4], 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 idle-to-dynamic 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. Essentially, the non-best core executes the application with more dynamic energy; however, waiting for the best core to become available expends idle energy by the idle, non-best core. SaT evaluates if the worse-than-best dynamic energy is more advantageous than expending idle energy.

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

$$Th = E_{dyn}(a_2, c_1) + E_{dyn}(a_1, c_1) + E_{idl}(a_2, c_2) > E_{dyn}(a_1, c_2), \quad (3)$$

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 [24]. Additionally, this scheduling enables SaT to populate the energy consumption and performance history for additional core configurations, which enhances future scheduling decisions.

4.3. 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 [5,34].

5. Experimental Setup

We evaluated SaT using our proposed architecture (Figure 3) with 34 embedded applications: sixteen (complete suite) from the EEMBC automotive application suite [36], six from Mediabench [37], and twelve from Motorola's Powerstone applications [29], which represent a diversity of application requirements [3,10,11].

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 1000 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 [38,39], we invoked SaT every 2000 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 [3], and SaT's efficacy can be evaluated with L1 caches, our experimental architecture's (Figure 3) 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 [32].

In order to directly compare to previous research [11], Figure 5 depicts our cache hierarchy energy model (similar to [19]) and we determined the dynamic energy using CACTI [40] 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., 0.032 um) [22]), this technology gives pessimistic energy savings for SaT: larger technology present lower idle-energy advantages of SaT. We estimated the idle and static energies each as 10% of the dynamic energy [32] and the CPU stall energy as 20% of the active energy [11].

$$
\begin{aligned}
\textbf{\textit{E(total)}} &= \textit{E(sta)} + \textit{E(dyn)} \\
\textbf{\textit{E(dyn)}} &= \textit{cache-hits} \times \textit{E(hit)} + \textit{cache-misses} \times \textit{E(miss)} \\
\textbf{\textit{E(miss)}} &= \textit{E(off chip access)} + \textit{miss-cycles} \times \textit{E(CPU stall)} + \textit{E(cache fill)} \\
\textbf{\textit{Miss Cycles}} &= \textit{cache-misses} \times \textit{miss-latency} + (\textit{cache-misses} \times (\textit{line size}/16)) \times \textit{memory band width)} \\
\textbf{\textit{E(sta)}} &= \textit{total-cycles} \times \textit{E(static-per-cycle)} \\
\textbf{\textit{E(static-per-cycle))}} &= \textit{E(per-Kbyte)} \times \textit{cache-size-in-Kbytes} \\
\textbf{\textit{E(per Kbyte)}} &= (\textit{E(dyn of base cache)} \times 10\%) / (\textit{base cache size in Kbytes})
\end{aligned}
$$

Figure 5. Cache hierarchy energy model for the level one instruction and data caches.

6. Evaluation Methodology

We compared SaT with prior configurable cache research [5,11] and scheduling algorithms [38,39] using three systems, denoted as system-1, system-2, and system-3. All systems had the same configurable heterogeneous multicore architecture (Figure 3), and provided the same per-core configurations mapping (Figure 2), 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 first-available-core policy [38,39].

System-1 was modeled similarly to [11] 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 awaits 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 [11], 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 1000 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 trade-offs 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.

Since the energy savings trades off performance [1], we measured the performance of system-1, -2, and -3, and to obtain insight on the performance traded for the energy savings potential, and calculated the energy-delay product (EDP) for these systems. Since system-1 and system-2 represent energy-conservative and performance-centric systems, respectively, we compared SaT's performance and energy-delay product to these systems.

To measure the performance of the system, we measured the execution time required for the system to process the applications queue in our experiment (Section 5). The shorter the execution time, the better the performance. Since the cores executed different applications at the same time, the cores' execution times differed, and we designated the longest execution time as the system's execution time. We used the total energy (idle + dynamic), and the longest execution time for the energy and performance in the EDP calculations.

7. Results and Analysis

Given the extensive analysis of our work, this section presents the energy, performance, and energy-delay product (EDP) results and analysis based on our experiment setup and evaluation methodology.

7.1. Energy Analysis

Figure 6a,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 correspond to less/more energy consumption than the base system.

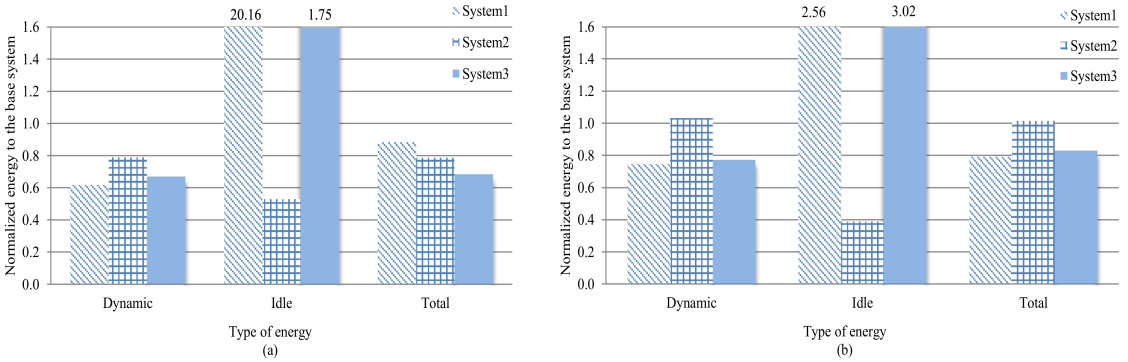

Figure 6. Energy of all systems normalized to the energy of the base system for the (**a**) data cache and (**b**) instruction cache.

Compared to the base system for the data and instruction caches respectively, system-1 (prior work) 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 cores' subsets, 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 could increase the system's total energy consumption, and still does not eliminate the potential for bottlenecks (i.e., depending on the applications). As a result, systems with an increased number of cores are more likely to waste more idle energy than to save dynamic energy.

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 [5]), 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.

As compared to the base system, for data and instruction caches respectively, system-3 (SaT) reduced the dynamic energy by 33.0% and 22.7%, and the total energy by 31.6% and17.0%. However, SaT increased the idle energy for the instruction and data caches, respectively, by 74.9% and 202.4%, as compared to the base system. SaT was able to save dynamic energy by scheduling applications to the best cores, at the expense of more idle energy. However, since SaT performed scheduling decisions using Equation (3), the total energy of SaT was still less than the base system's. This result suggests that SaT saves energy in a system if the savings in dynamic energy makes up for the expended idle energy.

To obtain further insight on the benefits of trading off dynamic or idle energy expenditure, we compared SaT to system-1 (prior work) and system-2 (performance centric), since these systems attempted to reduce dynamic energy and idle energy, respectively. We normalized the total energy of SaT to the total energy of system-1 and system-2.

Figure 7a,b depicts the total energy of system-3 (SaT) normalized to the total energy of system-1 and system-2, for data and instruction caches, respectively. SaT saved 22.6% and 13.0% more energy, as compared to system-1 and system-2, respectively for data caches. SaT saved 18.2% more energy as compared to system-2 for instruction caches; however, it increased the total energy by only 4.8%, as compared to system-1. These results provide the following insights: (1) SaT outperformed prior work and performance-centric in energy savings for data cache without requiring a priori knowledge of applications; (2) the energy savings superseded the profiling and tuning overhead of SaT for data caches, and SaT can save energy for systems with disparate data access patterns; (3) SaT increased the total energy for the instruction caches, as compared to prior work due to the instruction cache's higher requirement variations; and (4) a priori knowledge of the applications' domains and best configurations only provided minor energy improvements (4.8%) in prior work, compared to SaT. Unlike prior work, SaT dynamically profiled the applications and tuned the hardware, which broadens SaT's applicability and usability to any general purpose system.

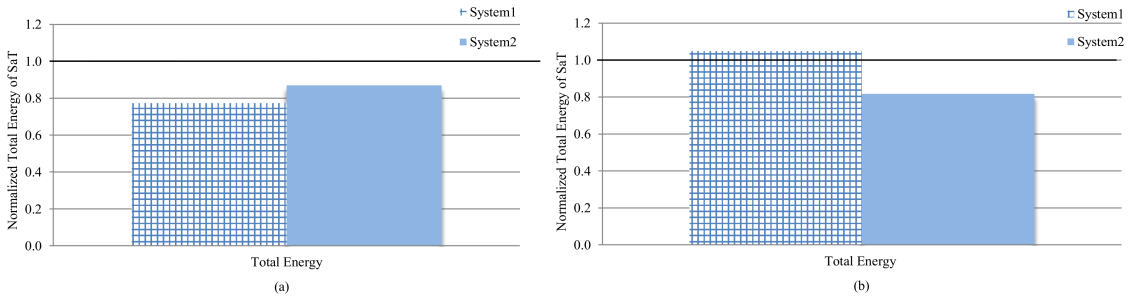

Figure 7. The total energy of system-3 (SaT) normalized to the total energy of system-1 and system-2, for (**a**) data and (**b**) instruction caches.

Although SaT increased the instruction cache energy as compared to system-1 (prior work), SaT outperformed system-1 with respect to the data cache energy savings, and SaT 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.

7.2. Performance Analysis

Figure 8a,b depict the performance for system-3 (SaT), normalized to the performance of system-1 (prior work) and system-2 (performance-centric) for the data and instruction caches, respectively. Values below/above 1 correspond to enhanced/degraded performance as compared to the base system.

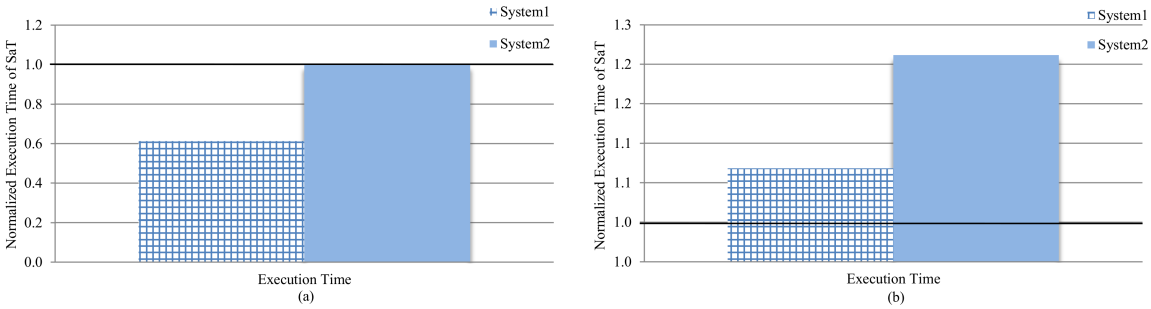

Figure 8. Execution time of system-3 (SaT) normalized to the execution time of system-1 and system-2, for the (**a**) data cache and (**b**) instruction cache.

SaT executed the applications 38.7% faster and 6.8% slower, as compared to system-1, for the data and instruction caches, respectively. Similarly, SaT executed the applications as fast as system-2 for the data cache and 21.2% slower than system-2 for the instruction cache. Since SaT prioritized energy to core utilization (performance), SaT exhibited closer behavior to system-1 than to system-2.

System-1 always scheduled applications to the best core, and applications must be halted until the best core was idle. Furthermore, since applications that belonged to the same domain contended for the same best core, the execution time was exacerbated based on the number of applications halted until the best core was idle: a queue that comprises applications of mostly a single domain will result in more performance degradation, as compared to a queue that comprises applications that belong to multiple domains.

Alternatively, system-2 was performance centric and aimed to maximize throughput and core utilization. In addition, system-2 alleviated the best-core bottleneck by scheduling applications to idle, non-best cores, using first available core policy, and, hence, outperformed SaT. This outcome was expected since SaT prioritized energy savings to performance and/or core utilization; SaT's decision to schedule applications to idle, non-busy core(s) was based on the energy saving advantage using Equation (3). Since Equation (3) used energy-advantage calculation to make scheduling decisions, to make SaT's performance comparable to a performance-centric system (e.g., system-2), a key modification is to allow the scheduling decisions based not only on energy, but also performance. A quality-of-service policy (e.g., [41]) can be incorporated into the decision-making process to guarantee adherence to performance constraints within a prefixed threshold.

Furthermore, the variation between data and instruction caches performance of SaT, compared to system-2 (i.e., only instruction cache performance was lower) can be explained by the fact that unknown configurations can affect instruction cache more than data caches [42]. Since SaT profiled the applications during runtime, SaT executed the applications on different cache configurations with unknown performance expectations, and the unknown configurations could degrade performance for the instruction cache more than data cache. One method to alleviate unknown performance impact while tuning is to use phase-based tuning [43], and/or incorporating quality-of-service threshold policy [40] into the decision-making of SaT, in order to make SaT suitable for performance centric systems. We plan to investigate these possibilities in future work.

7.3. Energy-Delay Product Analysis

Figure 9 depicts the energy-delay product (EDP) for system-3 (SaT), normalized to the EDP of system-1 (prior work) and system-2 (performance-centric) for (a) data and (b) instruction caches. Values

below/above 1 correspond to reduced/increased EDP as compared to the base system. The results revealed that, for the data cache, SaT outperformed system-1 and system-2 by 52.5% and 12.9%, respectively, and thus SaT met or outperformed system-1 and system-2 in terms of total energy saved (Section 7.1), performance (Section 7.2), and EDP. As for instruction cache, SaT under- and outperformed as compared to system-1 and system-2, respectively. SaT outperformed system-2 by 1%, which suggests that the energy savings compensated for the performance degradation, as compared to system-2. However, SaT underperformed system-1 by 10.9%. Even though SaT saved 18.2% energy, as compared to system-2, this energy saving was not enough to offset the 21.2% performance degradation.

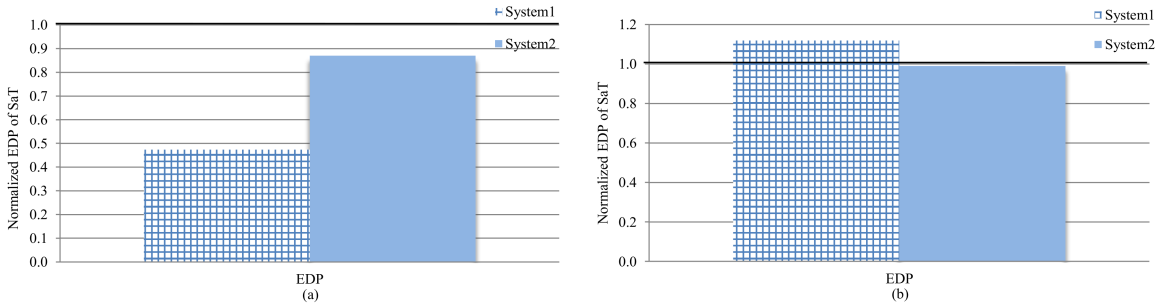

Figure 9. Energy-Delay-Product (EDP) system-3 (SaT) normalized to EDP of system-1 and system-2, for the (**a**) data cache and (**b**) instruction cache.

This result suggests that SaT is amenable to the majority of design goals such as low-energy, high-performance, or commercial-off-the-shelf systems. However, for systems with hard real-time constraints, SaT must utilize a policy that prioritizes performance as well as energy. We reiterate that, in general, SaT achieved substantial energy savings as compared to the base configuration and prior work, and, for future work, we intend to explore techniques for improving SaT's performance such as phase-based tuning [43], and/or incorporating quality-of-service monitoring policy [41] into the decision-making of SaT, in order to make SaT suitable for performance centric systems.

8. Conclusions

Heterogeneous and configurable multicore systems provide hardware specialization to meet disparate application hardware requirements. Multicore systems with a high degree of heterogeneity or highly configurable parameters provide a fine-grained hardware specialization at the expense of higher profiling, tuning overhead, and/or designer effort. To minimize profiling and tuning overhead and alleviate designer efforts, while providing fine-grained hardware specializations, we propose, to the best of our knowledge, the first heterogeneous and configurable multicore system with application-domain specific configuration subsets and an associated scheduling and tuning (SaT) algorithm.

To evaluate our system, we used a base, energy-conservative (representing prior work), and performance-centric systems, all of which had a priori knowledge of the applications. Our results revealed that our system saved up to 31.6% of energy, as compared to the base systems. Additionally, as much as 22.6% of energy compared to prior work, and with only 4.8% of profiling and tuning overhead. Furthermore, our results also revealed that our system outperformed prior work and performance-centric systems in 50% of the cases, and that our system provided EDP within 10.9% of a performance-centric system, which had a priori knowledge of the applications and the application's best core/configuration.

Future work includes integrating additional energy savings techniques into SaT, such as dynamic core shutdown and dynamic voltage and frequency scaling. To extend SaT capabilities to hard real-time systems, we plan to incorporate performance-and-energy based scheduling policies into SaT's

decision-making. Finally, we also intend to study different subset-to-core mapping methodologies with an increased number of cores to gain insight into the best per-core subset distributions and potential bottlenecks.

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

Author Contributions: Mohamad Hammam Alsafrjalani and Ann Gordon-Ross conceived of and designed the experiment. Mohamad Hammam Alsafrjalani performed the experiments and analyzed the results. Both authors wrote the article and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. ARM Ltd. big.LITTLE Technology. White Paper. Available online: http://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf (accessed on 1 February 2018).
2. Texas Instruments. OMAP3530 Applications Processors Datasheet. Available online: http://www.ti.com/lit/ds/sprt656/sprt656.pdf (accessed on 1 February 2018).
3. Adegbija, T.; Gordon-Ross, A. Exploring the Tradeoffs of Configurability and Heterogeneity in Multicore Embedded Systems. In Proceedings of the International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM'13), Porto, Portugal, 29 September–3 October 2013.
4. Gordon-Ross, A.; Vahid, F. A Self-Tuning Configurable Cache. In Proceedings of the 2007 44th ACM/IEEE Design Automation Conference, San Diego, CA, USA, 4–8 June 2007; pp. 234–237.
5. Viana, P.; Gordon-Ross, A.; Keogh, E.; Barros, E.; Vahid, F. Configurable cache subsetting for fast cache tuning. In Proceedings of the 2006 43rd ACM/IEEE Design Automation Conference, San Francisco, CA, USA, 24–28 July 2006; pp. 695–700.
6. Jeong, K.; Kahng, A.B.; Kang, S.; Rosing, T.S.; Strong, R. MAPG: memory access power gating. In Proceedings of the Design, Automation and Test in Europe, Dresden, Germany, 12–16 March 2012; pp. 1054–1059.
7. Kahng, A.B.; Kang, S.; Rosing, T.S.; Strong, R. Many-Core Token-Based Adaptive Power Gating. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2013**, *32*, 1288–1292. [CrossRef]
8. Adegbija, T.; Gordon-Ross, A. Phase-Based Dynamic Instruction Window Optimization for Embedded Systems. In Proceedings of the 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Pittsburgh, PA, USA, 11–13 July 2016; pp. 397–402.
9. Kora, Y.; Yamaguchi, K.; Ando, H. MLP-aware dynamic instruction window resizing for adaptively exploiting both ILP and MLP. In Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, Davis, CA, USA, 7–11 December 2013.
10. Zhang, C.; Vahid, F.; Najjar, W. A highly configurable cache architecture for embedded systems. In Proceedings of the 30th Annual International Symposium on Computer Architecture, San Diego, CA, USA, 9–11 June 2003; pp. 136–146.
11. Alsafrjalani, M.H.; Gordon-Ross, A.; Viana, P. Minimum Effort Design Space Subsetting for Configurable Caches. In Proceedings of the 2014 12th IEEE International Conference on Embedded and Ubiquitous Computing, Milano, Italy, 26–28 August 2014; pp. 65–72.
12. Kumar, R.; Tullsen, D.M.; Jouppi, N.P.; Ranganathan, P. Heterogeneous chip multiprocessors. *Computer* **2005**, *38*, 32–38. [CrossRef]
13. De Abreu Silva, B.; Cuminato, L.A.; Bonato, V. Reducing the overall cache miss rate using different cache sizes for Heterogeneous Multi-core Processors. In Proceedings of the 2012 International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, 5–7 December 2012; pp. 1–6.
14. Semeraro, G.; Magklis, G.; Balasubramonian, D.; Albonesi, S.; Dwarkadas, H.; Scott, M. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In Proceedings of the Eighth International Symposium on High-Performance Computer Architecture, Cambridge, MA, USA, 2–6 February 2002; pp. 29–40.
15. Silvano, C.; Palermo, G.; Xydis, S.; Stamelakos, I. Voltage island management in near threshold manycore architectures to mitigate dark silicon. In Proceedings of the 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 24–28 March 2014; pp. 1–6.

16. Alsafrjalani, M.H.; Ross, A.G. Dynamic Scheduling for Reduced Energy in Configuration-Subsetted Heterogeneous Multicore Systems. In Proceedings of the 2014 12th IEEE International Conference on Embedded and Ubiquitous Computing, Milano, Italy, 26–28 August 2014; pp. 17–24.
17. Kayiran, O.; Jog, A.; Pattnaik, A.; Ausavarungnirun, R.; Tang, X.; Kandemir, M.T.; Loh, G.H.; Mutlu, O.; Das, C.R. μC-States: Fine-grained GPU datapath power management. In Proceedings of the 2016 International Conference on Parallel Architecture and Compilation Techniques (PACT), Haifa, Israel, 11–15 September 2016; pp. 17–30.
18. Powell, M.; Yang, S.-H.; Falsafi, B.; Roy, K.; Vijaykumar, T.N. Gated-V_{dd}: A circuit technique to reduce leakage in deep-submicron cache memories. In Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISLPED'00), Rapallo, Italy, 26–27 July 2000; pp. 90–95.
19. Kaxiras, S.; Zhigang, H.; Martonosi, M. Cache decay: Exploiting generational behavior to reduce cache leakage power. In Proceedings of the Proceedings 28th Annual International Symposium on Computer Architecture, Goteborg, Sweden, 30 June–4 July 2001; pp. 240–251.
20. Zhou, H.; Toburen, M.C.; Rotenberg, E.; Conte, T.M. Adaptive mode control: A static-power-efficient cache design. In Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques, Barcelona, Spain, 8–12 September 2001; pp. 61–70.
21. Flautner, K.; Kim, N.; Martin, S.; Blaauw, D.; Mudge, T. Drowsy caches: Simple techniques for reducing leakage power. In Proceedings of the 29th Annual International Symposium on Computer Architecture, Anchorage, AK, USA, 25–29 May 2002; pp. 148–157.
22. Luo, J.; Jha, N. Battery-aware static scheduling for distributed real-time embedded systems. In Proceedings of the Design Automation Conference, Las Vegas, NV, USA, 22 June 2001; pp. 444–449.
23. Kim, K.; Kim, D.; Park, C. Real-time scheduling in heterogeneous dual-core architectures. In Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS'06), Minneapolis, MN, USA, 12–15 July 2006; p. 6.
24. Van Craeynest, K.; Jaleel, A.; Eeckhout, L.; Narvaez, P.; Emer, J. Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In Proceedings of the 2012 39th Annual International Symposium on Computer Architecture (ISCA), Portland, OR, USA, 9–13 June 2012; pp. 213–224.
25. Joao, J.A.; Aater Suleman, M.; Mutlu, O.; Patt, Y.N. Utility-based acceleration of multithreaded applications on asymmetric CMPs. *ACM SIGARCH Comput. Arch. News* **2013**, *41*, 154–164. [CrossRef]
26. Das, R.; Ausavarungnirun, R.; Mutlu, O.; Kumar, A.; Azimi, M. Application-to-core mapping policies to reduce memory interference in multi-core systems. In Proceedings of the 2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), Minneapolis, MN, USA, 19–23 September 2012; pp. 455–456.
27. Shelepov, D.; Alcaide, J.C.S.; Jeffery, S.; Fedorova, A.; Perez, N.; Huang, Z.F.; Blagodurov, S.; Kumar, V. HASS: A scheduler for heterogeneous multicore systems. *ACM SIGOPS Oper. Syst. Rev.* **2009**, *43*, 66–75. [CrossRef]
28. Wang, W.; Mishra, P.; Gordon-Ross, A. SACR: Scheduling-Aware Cache Reconfiguration for Real-Time Embedded Systems. In Proceedings of the 2009 22nd International Conference on VLSI Design, New Delhi, India, 5–9 January 2009.
29. Malik, A.; Moyer, B.; Cermak, D. A low power unified cache architecture providing power and performance flexibility. In Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISLPED'00), Rapallo, Italy, 26–27 July 2000; pp. 241–243.
30. Srikantaiah, S.; Kultursay, E.; Zhang, T.; Kandemir, M.; Irwin, M.J.; Xie, Y. MorphCache: A Reconfigurable Adaptive Multi-level Cache hierarchy. In Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture, San Antonio, TX, USA, 12–16 February 2011; pp. 231–242.
31. Chen, L.; Zou, X.; Lei, J.; Liu, Z. Dynamically Reconfigurable Cache for Low-Power Embedded System. In Proceedings of the Third International Conference on Natural Computation (ICNC 2007), Haikou, China, 24–27 August 2007; pp. 180–184.
32. Gordon-Ross, A.; Viana, P.; Vahid, F.; Najjar, W.; Barros, E. A One-Shot Configurable-Cache Tuner for Improved Energy and Performance. In Proceedings of the 2007 Design, Automation & Test in Europe Conference & Exhibition, Nice, France, 16–20 April 2007; pp. 1–6.
33. Rawlins, M.; Gordon-Ross, A. An application classification guided cache tuning heuristic for multi-core architectures. In Proceedings of the 17th Asia and South Pacific Design Automation Conference, Sydney, NSW, Australia, 30 January–2 February 2012; pp. 23–28.

34. Palermo, G.; Silvano, C.; Zaccaria, V. ReSPIR: A response surface-based Pareto iterative refinement for application-specific design space exploration. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2009**, *28*, 1816–1829. [CrossRef]
35. Adegbija, T.; Gordon-Ross, A.; Rawlins, M. Analysis of cache tuner architectural layouts for multicore embedded systems. In Proceedings of the 2014 IEEE 33rd International Performance Computing and Communications Conference (IPCCC), Austin, TX, USA, 5–7 December 2014.
36. EEMBC. The Embedded Microprocessor Benchmark Consortium. Available online: https://www.eembc.org/benchmark/automotive_sl.php (accessed on 15 December 2017).
37. Mediabench Consortium. Available online: http://euler.slu.edu/~fritts/mediabench/ (accessed on 15 December 2017).
38. Mauerer, W. Process Management and Scheduling. In *Professional Linux Kernel Architecture*, 1st ed.; Wrox: Norwood, MA, USA, 2008; Chapter 2; p. 37.
39. Silberschatz, A. Processes. In *Operating System Concept*, 9th ed.; Wiley: Hoboken, NJ, USA, 2012; Chapter 3; p. 111.
40. Reinman, G.; Jouppi, N.P. *CACTI2.0: An Integrated Cache Timing and Power Model*; COMPAQ Western Research Laboratory: Palo Alto, CA, USA, 1999.
41. Alsafrjalani, M.H.; Gordon-Ross, A. Quality of service-aware, scalable cache tuning algorithm in consumer-based embedded devices. In Proceedings of the 2016 International Great Lakes Symposium on VLSI (GLSVLSI), Boston, MA, USA, 18–20 May 2016; pp. 357–360.
42. Alsafrjalani, M.H.; Gordon-Ross, A. Instruction set architecture impact on design space subsetting for configurable systems. In Proceedings of the 2017 3rd IEEE International Conference on Control Science and Systems Engineering (ICCSSE), Beijing, China, 17–19 August 2017; pp. 720–723.
43. Adegbija, T.; Gordon-Ross, A. Energy-efficient phase-based cache tuning for multimedia applications in embedded systems. In Proceedings of the 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 10–13 January 2014; pp. 89–94.