Microprocessors and Microsystems 39 (2015) 271–285

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Cross-architecture prediction based scheduling for energy efficient execution on single-ISA heterogeneous chip-multiprocessors

Ying Zhang [a,*], Lide Duan [b], Bin Li [c], Lu Peng [a], Srinivasan Sadagopan [b]

[a] Division of Electrical & Computer Engineering, School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, LA 70803, United States
[b] Department of Electrical and Computer Engineering, The University of Texas at San Antonio, San Antonio, TX 78249, United States
[c] Department of Experimental Statistics, Louisiana State University, Baton Rouge, LA 70803, United States

ARTICLE INFO

Article history:
Available online 8 May 2015

Keywords:
Parallel processors
Heterogeneous systems
Energy efficiency
Modeling techniques

ABSTRACT

In recent years, single-ISA heterogeneous chip multiprocessors (CMP) consisting of big high-performance cores and small power-saving cores on the same die have been proposed for the exploration of high energy-efficiency. On such heterogeneous platforms, an appropriate runtime scheduling policy lies at the heart of program executions to benefit from the processor heterogeneity. To date, most prior works addressing this problem concentrate on the performance enhancement; however, they lack detailed justification of the runtime energy consumption and do not result in the most energy-efficient execution all the time. In this work, we pay attention to reducing the energy consumption for workloads running on heterogeneous CMPs and propose a scheduling algorithm based on dynamic execution behaviors to exploit better energy-efficiency. Our strategy is capable of significantly reducing the energy consumption while delivering comparable performance to a recently proposed heterogeneous scheduler (MLP-ratio), thus improving the energy-efficiency impressively.

1. Introduction

The near-cubic rise of power consumption with the increasing of core frequency has driven the processor development into the chip multi-processor (CMP) era in the past decade. By effectively exploring the thread-level parallelism, contemporary CMP computer systems stand as an attractive platform to manage the complex multi-threaded and multi-program execution scenarios. However, the ever increasing power crisis makes the energy-efficiency a first-order concern in addition to raw performance in a wide spectrum of computing platforms ranging from data centers that aim to reduce the utility cost to smartphones where a long battery life is among the most important goals, thus necessitating further exploitation of the energy-efficiency in future computing environments. Considering the diversity of program characteristics, conventional CMPs which consist of multiple identical cores might not be the perfect candidate for the exploitation. In this situation, computer architects propose the "heterogeneous CMP", where processor cores with disparate architectures are integrated on the same silicon, as an alternative design paradigm of traditional CMPs. It is widely acknowledged that heterogeneous chip multi-processors can effectively improve the energy-efficiency compared to homogeneous CMPs [8,17].

A heterogeneous CMP can be implemented in various manners, while the single Instruction-Set Architecture (ISA) organization is a representative design paradigm. A single-ISA heterogeneous CMP is usually composed of big cores equipped with complex out-of-order issue logic and sufficient computing resources and small cores on which the executions are driven by simple in-order pipelines. Put it another way, big cores provide high performance by consuming more power while the small cores reduce power dissipation at the expense of slower execution. To more effectively utilize the core heterogeneity and leverage their respective advantages, an appropriate job scheduler that is responsible for dynamic program-to-core assignment is in high demand. Ideally, the scheduler should be able to "foresee" the execution results (e.g., the system throughput or overall energy-efficiency) of all possible task distributions prior to a scheduling point, in order to select the most promising job assignments for the next execution period. Nevertheless, the diversities of program execution behaviors and architectural disparities among integrated cores make estimating the execution behavior of a program running on a core with different architecture a challenging problem. To work around this issue and effectively identify the optimal

* Corresponding author.
E-mail addresses: ying.esz.zhang@gmail.com (Y. Zhang), lide.duan@utsa.edu (L. Duan), bli@lsu.edu (B. Li), lpeng@lsu.edu (L. Peng), sadagopan.Srinivasan@amd.com (S. Sadagopan).

http://dx.doi.org/10.1016/j.micpro.2015.04.008
0141-9331/

program-to-core assignment, in this work we employ a rigorous statistical technique to generate a set of "signatures" with respect to common performance metrics, and use them to guide the scheduling decision at runtime.

On the other hand, in prior works targeting heterogeneity-aware scheduling, the program relative performance between big and small cores is widely adopted as the heuristic to guide the scheduling [5,8,17]. Specifically, programs that gain impressive performance boost on faster cores are selected to run on big cores; on the contrary, programs demonstrating mild performance improvement on big cores are chosen to execute on small cores. When applications with such distinctive features are concurrently running on a heterogeneous CMP system, a good scheduler will be capable of making the right decision and significantly improve the overall performance. Correspondingly, workloads falling into this category are considered as scheduling-sensitive [8].

While those proposed scheduling strategies are capable of improving performance, they do not necessarily lead to the most energy-efficient execution all the time, especially when the programs to be scheduled are scheduling-insensitive due to similar relative performance. We use the co-execution of programs *bzip2* and *vpr* on a dual-core heterogeneous CMP as an example to justify this argument. As listed in Table 1, these two programs demonstrate fairly similar performance ratio between big and small cores. Therefore, by employing an existing heterogeneity-aware scheduler based on relative performance, it is likely that (1) they are randomly mapped to different cores since the scheduler recognizes this workload as scheduling-insensitive, or (2) *bzip2* is assigned to the big core as it demonstrates slightly higher performance gain than its co-runner. In Fig. 1 we illustrate the energy consumption and energy-delay product (ED) for two possible scheduling, namely *bzip2*_B + *vpr*_S and *vpr*_B + *bzip2*_S, where the former one indicates that *bzip2* is running on the big core and *vpr* is on the small core. Similarly, the latter notation corresponds to the opposite scheduling decision. As can be observed, the second scheduling (i.e., *vpr*_B + *bzip2*_S) turns to be more energy-efficient than its alternative due to the significant power reduction on the big core. Two implications can be noticed from this example. First, scheduling aiming to minimize the energy consumption and ED does not always reach consensus with the performance-oriented scheduling in a heterogeneous system. Second, for scheduling-insensitive workloads where programs have fairly close speedup on the big core, there is still plenty of headroom for the energy efficiency optimization.

- Drawing upon these observations, we consider that an appropriate scheduling policy targeting on energy minimization is of great significance in emerging heterogeneous systems. Therefore in this work, we propose a rule-set guided scheduling strategy to minimize the energy consumption for workloads running on heterogeneous CMPs. Meanwhile, our scheduler is able to deliver comparable performance to the optimal existing heterogeneous scheduler, thus achieving higher energy efficiency than previous schemes. We employ an advanced statistical tool to facilitate the development of our algorithm. The tool is able to generate a set of "IF-ELSE" conditions with regard to common performance metrics on involved cores. Each condition is expressed as an inequality such as "$X_i \leqslant (\text{or} \geqslant) N$", where X_i is an easily measured performance metric and N is a certain value. The scheduler then dynamically makes decisions for program assignment by comparing the runtime execution behaviors with the selected rules at each scheduling interval. When the conditions on both cores are satisfied, the scheduler predicts that a job swap will be more energy-saving than the current mapping, thus switching the programs on the big and small cores accordingly. Otherwise the current scheduling is maintained for the next period. In general, compared to prior schemes, our rule-set based scheduling strategy has the following advantages.
- It is capable of identifying the most energy-efficient execution patterns on heterogeneous CMPs. Instead of exclusively concentrating on performance gain, it pays attention to the energy saving and effectively increases the overall energy efficiency by scheduling intervals that might be inappropriately handled or ignored in throughput-targeted scheduling.
- It is a rigorous approach as the selective rules are generated by an advanced statistical tool. This means that the scheduling is guided by factors that are essentially influential to the energy efficiency rather than arbitrary variables.
- It is easy to be implemented in practice since no extra hardware is required. The performance counters available in most commercial processors can be utilized to monitor the execution behaviors for scheduling.

Table 1
Power and performance ratio for *bzip2* and *vpr*.

Program	Big core power (W)	Small core power (W)	Performance ratio
bzip2	24.64	9.18	2.51
vpr	18.97	10.32	2.48

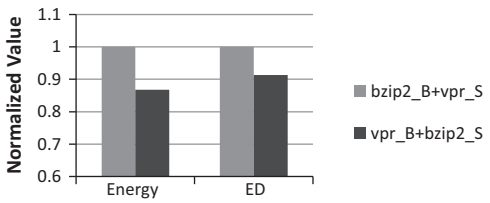


Fig. 1. Energy and ED for *bzip2* + *vpr* with different mappings.

2. Related work

Within past years, several researchers have authored outstanding studies in the heterogeneous architecture field. Kumar et al. [17] propose one of the earliest single-ISA heterogeneous multiprocessor and discuss its potential for power reduction. The sampling-based scheduling algorithm that can be applied to a realistic multi-processor for energy-efficient execution is also proposed. In [18], the performance for multithreaded workload executing on a single-ISA heterogeneous processor is analyzed in detail. By adopting a similar sampling-based assignment policy, the system can capture the intra-thread diversity and schedule the jobs for the maximal throughput. Becchi and Crowley evaluate a set of static and dynamic scheduling policies designed for heterogeneous platform in [5]. The authors show that dynamic job scheduling largely outperforms the static assignment by delivering higher throughput. Hao et al. [11] describe a scheduling policy using hardware counters and evaluate it on a real multiprocessor system running Linux. They argue that the last level cache access latency is a good metric to guide the scheduling on heterogeneous platform. In [16], Koufaty et al. introduce the bios scheduling which is similar to the policies based on memory intensity. Balakrishnan et al. quanti-tatively analyze the impact of performance asymmetry between cores on the application scalability and predictability [4]. Saez et al. present a series of works that target the performance enhancement on asymmetric CMP platforms [25–27]. They propose an algorithm named HASS [27] to guide the job assignment on single-ISA heterogeneous systems for the maximum performance. They also develop the CAMP scheduler to explore both efficiency and TLP [25,26]. Radojkovic et al. [24]

consider a scenario with massive multithreaded processors where an exhaustively search for the optimal task assignment is unfeasible due to the substantial possibilities. Therefore, they introduce a statistical approach to seek the best work distribution. Li et al. [22,23] implement a scheduler composed of fast-core-first assignment and migration on a performance-asymmetric CMP architecture. Lakshminarayna, on the other hand, propose an age based approach that maps the thread with longer remaining execution time to a faster core [19]. More recently, Craeynest et al. present a heterogeneous scheduler via performance impact estimation (PIE) [8]. Their evaluation results demonstrate that the PIE scheduling policy outperforms prior schemes based on program memory intensity. The authors also show that memory-level parallelism [9] ratios of programs provide good estimation for relative performance and can be employed to guide the runtime scheduling. A similar strategy through the prediction of CPI across core types is proposed by Srinivasan et al. [33].

Studies addressing energy minimization on heterogeneous platform can also be found in literature. Saad et al. [29] and Goraczko et al. [13] respectively propose the software partitioning approach to reduce the energy consumption on heterogeneous embedded systems. In [7], Chen and John present a scheduler based on weighted Euclidean distances to improve the energy efficiency on heterogeneous CMPs. Sharifi et al. [30] takes temperature into account and introduce a joint solution for thermal and energy management. Grant and Afsahi [14] introduce a scheduling mechanism to save energy on asymmetric multiprocessors for scientific applications. In their proposed algorithm, one core is reserved for running the operating system at adjustable frequencies while other processors are executing the user threads at full speed. In [12], Heath et al. design a heterogeneous server cluster which demonstrates remarkable energy efficiency improvement over traditional homogeneous clusters. Singh et al. [31] propose a prediction based approach for power estimation and scheduling on traditional homogeneous CMPs, in order to improve the energy efficiency.

3. Heterogeneous architecture

The heterogeneous platform considered in this study is a single-ISA CMP containing a number of big and small cores. Fig. 2 illustrates its architectural overview. As can be seen, each core is equipped with private L1 caches, connecting to the shared L2 cache via an interconnection. The main memory stands as the lowest level in the memory hierarchy and communicates with the shared cache through a memory controller. Note that although our study is conducted on CMPs with shared last-level caches (LLC), the rule-set guided scheduling approach can also be adapted to systems without shared LLC and effectively increase the energy efficiency.

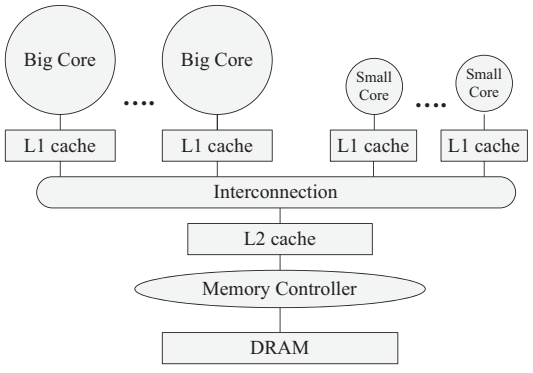


Fig. 2. Architectural overview of the considered heterogeneous CMP.

4. Statistical tools

As described in Section 1, the proposed scheduling scheme is built on the measurements of common performance metrics. This introduces two challenging problems to our study. First, we should identify the important factors which impose relatively large impact on the overall energy consumption. Second, we need to quantitatively formulate the scheduling condition with regard to the selective performance metrics. Taking these into consideration, we employ an advanced statistical tool to facilitate the rule extractions.

4.1. Patient Rule Induction Method (PRIM)

PRIM is an advanced statistical model [10] whose objective is to find a region in the input space (com-posed of configuration parameters in this work) that gives relatively low values for the output response, e.g. the FIT value. The selected region (or "box") is described in an interpretable form involving a set of "rules" depicted as $B = \cap_{j=1}^{p}(x_j \in s_j)$, where x_j represents the jth input variable and s_j is a subset of all possible values of the jth variable. In other words, the identified region B is the intersection of p subsets, each of which is from one of the p input variables.

Fig. 3 illustrates the construction of the "optimal" region, which is composed of two phases: (1) patient successive top-down peeling process; (2) bottom-up recursive pasting process. The top-down peeling starts from the entire space (box B) that covers all the data. In each iteration, a small subbox b within the current box B is removed; we calculate the output mean for the elements remaining in $B-b = \{x \in B \ \& \ x \notin b\}$, performing this operation in each dimension (i.e., try removing a different subbox from each input variable); finally we choose the one which yields the smallest output mean value for the next box $B-b$. This procedure is applied iteratively until the proportion of the data points remaining in the current box (termed the support) is below a preset threshold β. Note that for a categorical variable, an eligible subbox b contains only one element of the possible values of the variable in the current box B.

The pasting algorithm works inversely from the peeling results and the final box can be improved by readjusting its boundaries. The reason for including this step is that we only look one step ahead in each peeling iteration, thus the box boundary is determined without knowledge of later iterations. This implies that we may peel too much from the input space and eliminate many design options unintentionally. In specific, the pasting phase works as follows. Starting with the peeling solution, the current box B is iteratively enlarged by pasting onto it a small subbox that minimizes the output mean in the new larger box. We iteratively apply this process and successively enlarge the current box, until the addition of the next subbox causes the output mean to increase.

According to the above description, it is straightforward to derive that the first peel stage introduces at most $n \times \sum_{j=1}^{p} C_j$ comparisons, where n denotes the number of observations, p is the number of input variables, and C_j indicates the amount of values that the jth variable can take. Each peeling iteration conducts a gradually decreasing number of operations since there are fewer samples left after a peeling iteration. PRIM performs approximately $-\log(n)/\log(1-\alpha)$ peeling steps where α denotes the percentage of points removed at each iteration.

PRIM outstrips many widely adopted strategies including greedy methods by providing much more stable solutions (hyper-boxes). For instance, a binary tree partitions the data quickly because of its binary splits, while with PRIM only a small portion of data is removed at each iteration. As a consequence, in case where the training data is slightly changed, a tree structure

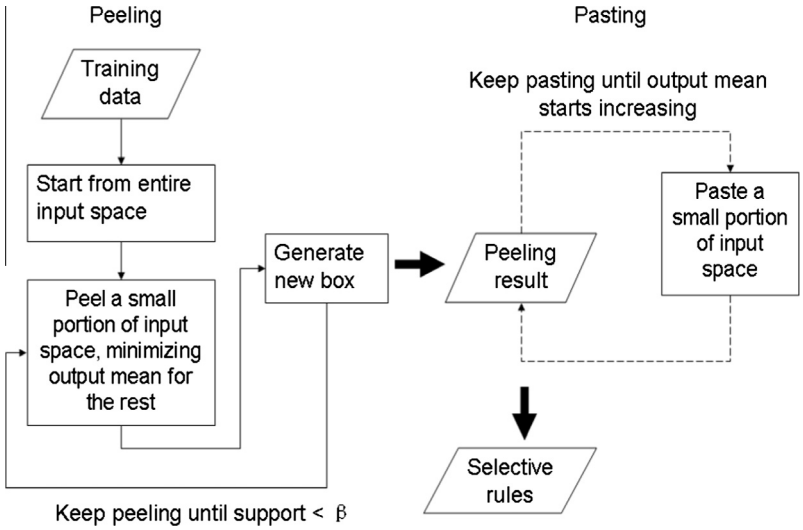


Fig. 3. PRIM training procedure, including peeling and pasting.

may change drastically but the PRIM solution is less affected. On the other hand, PRIM is also capable of identifying the optimal region even if it is inconsecutive. In this situation, PRIM will generate a sequence of hyper-boxes instead of only one. Namely, the PRIM algorithm can be repeated on the remaining dataset after getting the first hyper-box. By doing so, the disconnected subspace can also be covered. In this work, we found that the leading box often covers most of the points with the small response values, thus we only identify the first hyper-box.

4.2. Classification and Regression Tree (CART)

Although the described statistical technique PRIM is able to build a rigorous correlation between multiple input variables and a response, the accuracy of the model depends on features of the applications in the training set. Let us assume that the execution behaviors of a few intervals significantly deviate from those of other training instances while their response values are identical. In this scenario, a single universal PRIM model may not be capable of capturing all those runtime variations. This is because that the PRIM algorithm is prone to build a model that fits the majority situations in the training instances. As a result, the established model might ignore those samples appearing less frequently. Considering the diversity of program characteristics, this limitation might significantly decrease the prediction accuracy when the model is applied to different program phases or applications that demonstrate completely distinct execution behaviors to training samples.

In order to figure out this problem, we propose to partition the entire data set into several categories, each of which contains instances demonstrating similar characteristics. If we train a PRIM model for each data subset and generate a group of rule sets correspondingly, the obtained rules are supposed to be more robust and be effective to handle different execution scenarios. To achieve this goal, we employ another statistical tool named Classification and Regression Tree (CART) [6] for the data segmentation. CART has been in use for about 25 years and remains a popular data analysis tool. It provides an alternative to linear and additive models for regression problems. The CART models are fitted by a recursive partitioning whereby a dataset is successively split into increasingly homogenous subsets until the information gained by additional splits is not outweighed by the additional complexity due to the tree's growth. Trees are adept at capturing non-additive behavior, e.g. interactions among input variables are routinely and automatically handled. Further, regression tree analysis can easily handle a mix of numeric and categorical input variables.

5. Rule-set guided scheduling

While most modern operating systems support con-current execution of multiple programs running on different cores, it is not necessary to keep all processor cores active during the entire execution. That is, a portion of cores might be idle (or in power-saving mode) while others are running in order to reduce the total energy consumption. This implies two scheduling circumstances that need to be carefully considered on a heterogeneous CMP platform; namely, (1) choosing appropriate cores to execute the programs while making other cores idle and (2) identifying a suitable task-to-core mapping when all cores need to be utilized. In this section, we will present how the rule-set guided scheduling strategy would be applied in these two scenarios in detail. Moreover, the scheduling policy should be sufficiently scalable as heterogeneous CMPs might be configured in different manners. Therefore, we also discuss the effectiveness of the proposed strategy on heterogeneous platforms with different configurations.

5.1. Scheduling in presence of idle cores

It is fairly common that a portion of integrated cores on a CMP are idled at runtime for the sake of power saving. For instance, assume a single-thread program is to be executed on a heterogeneous chip multi-processor similar to the Big.Little platform from ARM which consists of a powerful big core and a slow small core [1]. In this situation, it makes no sense to enable both cores since one core is sufficient to run the program at any instant during the execution. Considering the representativeness of this scenario in practice, we demonstrate the single-program scheduling on a dual-core system to exemplify the implementation of the rule-set guided strategy in presence of idle cores.

For a scheduling interval, we must identify whether to run the program on the big core or the small core. Clearly, an oracle scheduler will examine these two cases during runtime (at each scheduling point) and choose the most suitable core for execution to achieve the optimal energy efficiency. However, dynamically determining the optimal schedule for a program at runtime is a challenging problem. To overcome this conundrum, we employ

Patient Rule Induction Method (PRIM) to generate some selective rules on a number of performance measurements. In a scheduling interval, if the measured performance counters conform to these rules, the scheduler will map the program to the appropriate core accordingly.

More specifically, the PRIM model training is composed of the following steps. First, we select a number of typical programs for extracting the rules. For each of them, we respectively affinitize it to the big core and the small core for execution and collect a set of easily measured performance metrics along with the energy consumption for every interval, whose length is set to a reasonable value for the study. By doing so, we can obtain the following information from the two types of cores:

$$\text{Information from big core} :< X_b^1, X_b^2, X_b^3, \ldots\ldots, X_b^m, E_b >$$

$$\text{Information from small core} :< X_s^1, X_s^2, X_s^3, \ldots\ldots, X_s^n, E_s >$$

In the tuple listed above, the X variables denote the measured performance counters such as the number of cache misses and the number of branch mispredictions. The subscript of each variable indicates the corresponding platform (i.e., b = big core, s = small core). The variable E represents the energy consumption of this interval. In this example, we measure m performance counters on the big core and n counters on the small core. Second, we compare the energy consumption for each interval under these two assignments and set a Boolean flag based on the comparison result. The flag is then used as the output of a training instance. Finally, we feed the training samples measured from all selected programs to establish PRIM models and extract the rules.

It should be noted that separate models should be established for big and small cores. This is because the program is running on either the big core or the small core at any interval, requiring two groups of conditions to respectively guide the big-to-small and small-to-big migration. Let us first focus on the big-core model that is used to manage the big-to-small migration. We assume that E_s is smaller than E_b for a specific interval. With this assumption, the training sample corresponding to this interval is $<X_b^1, X_b^2, X_b^3, \ldots, X_b^m, flag>$ where the flag is set to 1, indicating that the program should be assigned to the small core for energy saving. In contrast, if E_s is greater than E_b, a flag of 0 will be set. This means that running the program on the big core is more preferable (i.e., it is unnecessary to transfer the job to the small core). We train a model for the small core to govern the small-to-big migration in a similar fashion. In specific, training instances in a form of $<X_s^1, X_s^2, X_s^3, \ldots, X_s^n, flag>$ are fed into the PRIM tool. Note that there are two approaches to measure the runtime energy consumption in practice:

(1) If the processor provides hardware counter to report the power usage, we just need compare the energy consumption between the aforementioned two cases. Then we set the flag based on the comparison result. Some recently released processors such as Intel Sandy Bridge architectures and later products support dynamic power measurement by using a model-specific register (MSR) [3].
(2) In case that there is no dynamic energy reporting function on the chip, we can have an accurate estimation on runtime energy via multiplying the average power and the execution time. The dynamic power of the chip can be estimated from performance counters through another predictive model [15]. Specifically, the chip power can be added up by each component's power derived from their accessing rates, a scaling factor, and the maximal component power, plus idle power. The access rate of a component can be read and calculated from performance counters; the maximal power of each component and the scaling factors are generated and tuned by running a set of stress benchmarks.

Recall that PRIM rules identify the input space subregion that has the highest response values. Therefore, the generated rules quantify the situations that a program migration from the big core to the small core (or the other way around) is needed to achieve better energy efficiency.

The selective PRIM rules are then engaged by the operating system to guide the scheduling of the program between two cores. Assume the program is randomly mapped to a core (either the big one or the small one) at initial. At a scheduling point, the performance measurements are compared with the extracted PRIM rules corresponding to the current used core. If conditions are satisfied, the model predicts that transferring the job to the other core will lead to better energy efficiency; otherwise the present scheduling is preserved. The scheduler then makes the assignment based on the prediction result and continues the execution to the next scheduling point. Note that the rule-set guided scheduling is sufficiently flexible to manage the program execution for optimizing different metrics. For instance, by changing the objective during the model construction, this approach can be easily applied to guide the scheduling in a system where performance maximization is the prime concern. Nevertheless, our concentration in this paper is energy minimization.

5.2. Scheduling without idle cores

When the number of concurrent programs is increasing, all integrated cores on a CMP might be utilized to maximally exploit the processor computation capability. In this situation, the scheduling problem is essentially to identify the task-to-core mapping which results in the minimal energy consumption. Without loss of generality, we consider a scenario where two programs (A and B) run on a dual-core CMP consisting of one big core and one small core. For a scheduling interval, we need to compare the total energy consumption of the following two cases: (1) A on the big core and B on the small core; and (2) B on the big core and A on the small core. Between these two scheduling, we should choose the one with the lower energy consumption. Similarly, we adopt the PRIM tool to generate a set of rules to guide the scheduling.

The training procedure is fairly close to that described in the previous subsection. The most significant difference lies in that a unified model regarding to the performance metrics from both the big and small cores is built, meaning that conditions on big and small cores are checked simultaneously at a scheduling point. This is because that both the big and small cores are utilized to run programs, thus the execution behaviors from both sides should be monitored in order to evaluate whether a job swap leads to less energy consumption. The specific training process is as follows. First, we randomly select a certain number of program pairs. For each program pair (A and B), we assume that A runs on the big core and B runs on the small core. For each interval, we can obtain the following information by executing A and B on the big and small cores, respectively:

$$\text{Program A} :< X_b^1, X_b^2, X_b^3, \ldots\ldots, X_b^m >$$

$$\text{Program B} :< X_s^1, X_s^2, X_s^3, \ldots\ldots, X_s^n >$$

Similarly, the variables X denote the measured performance counters. Second, we compare the energy consumption of this schedule with its counterpart (re-running B on the big core and A on the small core), setting a Boolean variable (flag) to one if

swapping these two programs will generate lower energy. Consequently, we can form a PRIM training sample by combining the above information:

$$< X_b^1, X_b^2, X_b^3, \ldots\ldots, X_b^m, X_s^1, X_s^2, X_s^3, \ldots\ldots, X_s^n, flag >$$

For each training instance, the inputs are the $m + n$ performance counters from both cores while the output is a flag indicating if these two programs need to be switched in the next interval. We then feed all instances into PRIM to generate the conditions.

Fig. 4 illustrates how the rule set interacts with the OS and makes decision for program assignment at runtime. The two programs are first executed on two cores (one big and one small) for an interval, respectively. At a scheduling point, the performance measurements of the current interval are compared with the extracted PRIM rules. If conditions on both cores are satisfied, the model predicts that swapping the two programs will lead to better energy efficiency; otherwise the present scheduling is preserved. The scheduler then makes the assignment based on the prediction result and continues the execution to the next scheduling point.

As described in Section 4, the effectiveness of the rule guided scheduling is largely determined by the features of the programs in the training set. In case where the programs for validation demonstrate significantly different execution behaviors from the training programs, the derived rules may not be effective in identifying the swapping cases. In this situation, the model accuracy can be further improved by preprocessing the training data. Instead of training a single PRIM model, we can build a number of different PRIM models according to the similarity of different training samples. Specifically, we use the CART mechanism to partition the input space into a few subregions. The points belonging to each individual subregion are similar in terms of energy efficiency. After that, we build a separate PRIM model for each of these subregions. Consequently, we will have a group of rule sets. When making predictions during runtime, we first identify which subregion the current input sample locates in, then use the corresponding rule set to determine if a program switch is needed. In practice, the number of subregions does not need to be large. Our experiments show that partitioning the input space into 4 subregions (and also training 4 PRIM models accordingly) can result in prediction accuracy within only 5% difference from the oracle scheduler. This approach is termed Hierarchical PRIM (or H-PRIM).

5.3. Algorithm scalability

Our approach is sufficiently scalable to be adopted by a system with more than 2 cores. In this subsection, we consider two generalized heterogeneous platforms and show that how the rule-set based schedulers lead to energy-efficient execution on these architectures.

We first assume a CMP with an equivalent number of big and small cores while the core count of each processor type is n. In this scenario, the optimal energy efficiency can be achieved by performing n iterations of parallel pair comparison. The scheduling process is illustrated in Fig. 5. As shown in the figure, in the first iteration, a big core with the index i ($i \in [0, n-1]$) is compared with the small core whose index is $(n + i\%n)$. All n pairs of comparisons are performed in parallel. In the second iteration, the big core i will form a group with the small core $(n + (1 + i)\%n)$ and make comparison correspondingly. Similarly, the comparison will be conducted between the big core i and the small core $(n + (n - 1 + i)\%n)$ in the nth iteration. Note that the mod operations are involved to emulate the rotational comparisons. We prove that this method will lead to the optimal scheduling as follows.

Since we have n big cores and n small cores, as well as $2n$ jobs running on them, the optimal schedule is a situation that n jobs suitable running on the big cores for low energy consumption (we label the jobs as "1"s) will be assigned to n big cores and the left n jobs, denoted as "0"s, will be allocated on n small cores. We claim that all "1" programs will be assigned to big cores and all "0" jobs will be allocated on small cores after n iterations, even though we are unaware of the program classification at the beginning, i.e., whether a program belongs to "1" category or "0" category. During each of the n iterations, we have n parallel comparisons between big and small cores. For each comparison, we seek better energy efficiency for two programs running on a big-small core pair. Therefore, we have four possible situations before the comparison:

(1) a "1" job running on a big core compared with a "0" job running on a small core;
(2) a "0" job running on a big core compared with a "1" job running on a small core;
(3) a "1" job running on a big core compared with another "1" job running on a small core;
(4) a "0" job running on a big core compared with another "0" job running on a small core.

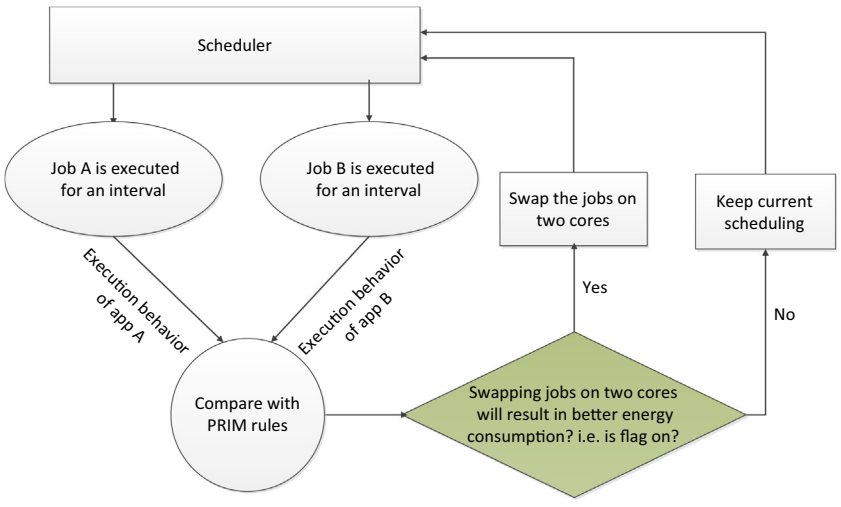


Fig. 4. PRIM rules guided scheduling for dual-program execution.

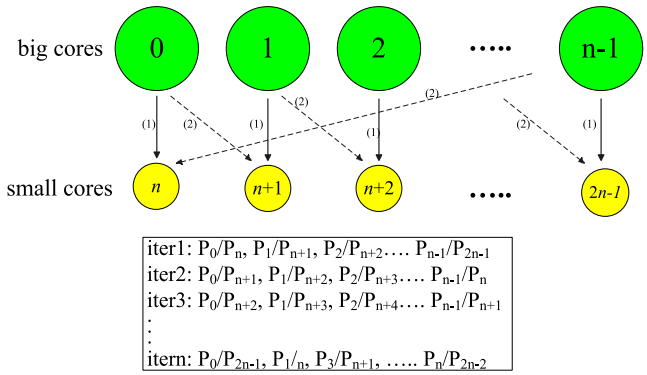


Fig. 5. Pair-wise comparison illustration for $2n$-program scheduling on an nB + nS platform.

For the first two cases, it will generate an ideal situation that a "1" job will be assigned on a big core. For the third case, a "1" job will also be allocated on a big core, no matter which "1" job is selected. Similarly, a "0" job will be set on a big core for the fourth case. However, there must be a "1" job running on a small core at this point, considering that the number of "1" jobs is equal to the total number of big cores. This implies an opportunity for this "1" job running on a small core to be compared with a "0" job executed on a big core in a future iteration, since we have n iterations of parallel comparisons. Thus, any case (4) comparison will fall into case (2) comparison eventually. Based on this analysis, we conclude that all "1" jobs will finally go to big cores, meaning that the optimal schedule is achieved after n iterations.

Our algorithm can be further generalized to guide the scheduling on a heterogeneous CMP with non-equivalent number of big and small cores. Let us assume there are m big cores and n small cores. Therefore, there should be a total of m jobs with label "1" and n jobs with label "0". Without loss of generality, we assume that m is greater than n. In this situation, the PRIM-based approach is capable of reaching the desired scheduling status by performing $\lceil m/n \rceil$ rounds of parallel comparisons described in above as shown in Fig. 5. In case that m is less than n, the algorithm is similar but requires $\lceil n/m \rceil$ rounds of parallel comparisons.

Fig. 6 illustrates the scheduling procedure on such a heterogeneous CMP. As can be noted, the parallel comparisons are conducted within a window whose size is equal to n (i.e., the smaller one between m and n). By doing so, we are able to perform n iterations of parallel comparisons between n big and n small cores. Note that the total number of "0" jobs is n and total number "1" jobs is m. According to the analysis described earlier (where m is equal to n), after each round of parallel pair comparisons between n big cores and n small cores, all of the n big cores will have "1" jobs running on them. Therefore, after rounds of parallel comparisons, all big cores will have "1" jobs. Meanwhile, all "0" jobs are scheduled running on the small cores.

It is important to notice that this approach introduces fairly light overhead to the program execution. First, the model training is conducted offline and therefore has no impact on the dynamic execution. Second and more importantly, the pair-wise comparisons which are performed at each scheduling interval can be completed in reasonable time due to the parallel operation in each round. Specifically, although the total number of comparisons to reach the desired scheduling is approximately $O(mn)$, all comparisons can actually be finished in $O(m)$ time, where m is the larger core count (i.e., $m \geqslant n$ on a CMP with m big cores and n small cores, or the other way around). Note that traditional heterogeneity-aware scheduling policies based on relative performance estimation involve a sorting process in order to identify the programs suitable to run on big cores (or small cores). Assume a quicksort algorithm is employed for the operation. This introduces $O(n\log n)$ comparisons where n is the total number of programs. Therefore, our rule-set based scheduling policy raises no additional overhead compared to state-of-the-art strategies.

Also, the algorithms discussed in this work are built on an assumption that two types of cores are integrated on the die. This is reasonable considering that most commercial heterogeneous chip multi-processors including ARM Big.Little [1] and Nvidia Tegra 3 [2] are composed of two families of cores for good tradeoff between the design complexity and energy-efficiency.

6. Methodology

6.1. Simulation environment

We use a modified SESC simulator [28] to conduct the experiments in this work. The simulator is configured to contain a number of big and small cores, whose architectural parameters are listed in

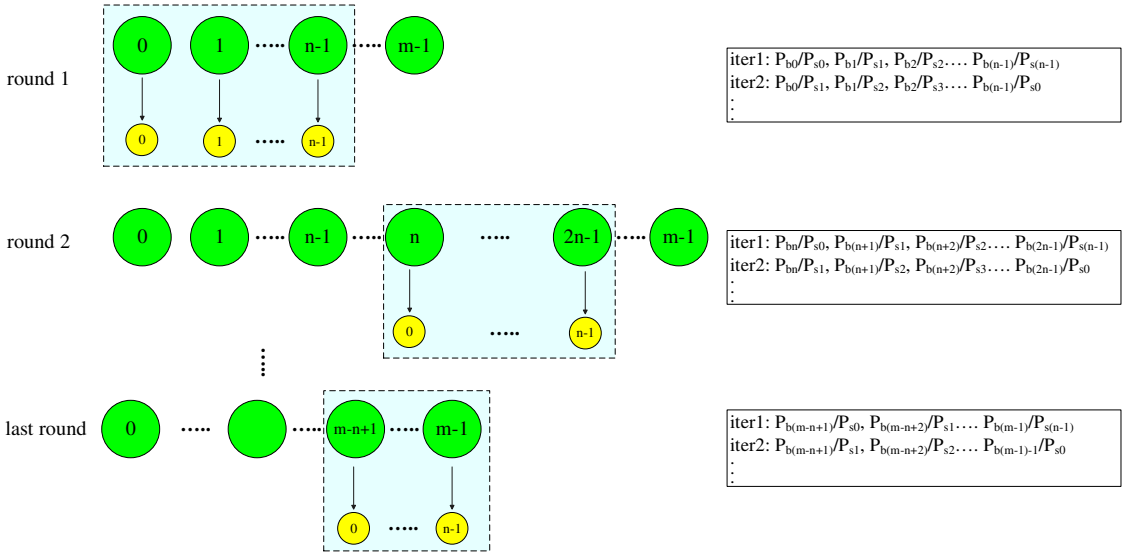


Fig. 6. Scheduling procedure on a heterogeneous CMP with m big cores and n small cores ($m > n$). Big cores are denoted as P_{bi} ($i = 0, 1, \ldots, m-1$); Small cores are denoted as P_{sj} ($j = 0, 1, \ldots, n-1$).

Table 2
Architectural parameters of system components.

Component	Parameter	Value
Big core	Pipeline type	Out-of-order
	Processor width	4
	ALU/FPU	4/2
	ROB/RF	120/160
	L1I cache size	32 KB
	L1D cache size	32 KB
	L1 associativity	4
	BTB entries	2048
Small core	Pipeline type	In-order
	Processor width	2
	ALU/FPU	2/1
	L1I cache size	16 KB
	L1D cache size	16 KB
	L1 associativity	2
	BTB entries	1024
Other parameters	L2 cache size	4 MB
	L2 associativity	8
	Cache block size	32B
	Branch Predictor	Hybrid
	Frequency	3G

Table 2. McPAT v0.8 [21] is used for dynamic and leakage power estimation. We select 26 programs from SPEC 2000 and SPEC 2006 with the ref input size for the study. In the multi-program simulation, we form 220 workloads composed of individual programs. Note that we do not use other programs from the suites for two reasons: (1) our current cross compiler is only capable of compiling programs implemented with C/C++. Many remaining programs are written in Fortran, thus it is difficult to co-compile them with C/C++ applications; (2) we pay much attention to scheduling-insensitive programs, which are usually not carefully examined in performance-oriented scheduling studies, to demonstrate and exploit the opportunity of energy optimization.

Each program is simulated 1 billion instructions after fast-forwarding the initial 2 billion. For the single-program study, we use 19 programs for training and use the left 7 programs for validation. For the multi-program study, we choose 180 out of the 220 program combinations for PRIM model training and use the remaining ones to evaluate the effectiveness. Recall that the training procedure is conducted offline. This takes about 3 s on a Dell Precision T7500 workstation equipped with an Intel E5530 CPU. In addition, for the scheduling in absence of idle cores, we always launch as many programs as cores. We stop the simulation when the slowest application in the workload completes 1 billion instructions. The faster applications are not repeating. By doing so, we guarantee that the same amount of work is always performed when different scheduling policies are engaged, i.e., each application in the workload executes 1 billion instructions after the initial 2 billion. This makes the comparison of total energy consumption from run to run rational. Note that once faster programs complete, the scheduling problem deprecates to the situation with idle cores.

The scheduling interval is set to 2.5 ms in this study. As shown in prior works [8], this granularity is small enough to capture the variations in program execution behaviors and assist the scheduler to make energy-efficient assignments more precisely. We do account the migration overhead due to architectural state retrieving and set it to 150 μs [20]. The additional energy dissipation due to the migration is also appropriately modeled. For instance, the energy consumed by cache re-warming can be calculated from the corresponding cache access times. The time overhead of the scheduler is ignorable because making a scheduling decision only requires reading the performance counters from the big and small cores and comparing them with the corresponding rules. We compare performance, energy consumption, and ED product resulted from different schedulers to assess the effectiveness. Note that since each workload executes the same amount of instructions under different scheduling policies, comparing the total energy consumption is equivalent to comparing the energy-per-instruction (EPI). We thus use EPI as the metric for interpretation in later sections.

6.2. Scheduling algorithms for comparison

In this subsection, we introduce the scheduling strategies that are implemented for comparison.

- **Static scheduling:** This is the baseline scheduler implemented for the comparison. The programs are pinned to processor cores and execute till completion. For the single-program investigation, this means two specific approaches: static-big where the program is mapped to the big core; and static-small where the program goes to the small core. For the multi-program evaluation, we run all possible task-to-core mappings and choose the most energy-saving one as the baseline for comparison.
- **Round-robin (R-R):** With this policy, the programs running on the big and small cores are swapped every 5 intervals. The scheduler does not take into account the program difference and runtime execution behaviors, but blindly swapping the jobs at a preset frequency. We set the swap period to 5 intervals because such a setting leads to the optimal energy-efficiency for round-robin scheme. Specifically, we ran a sensitivity study by sweeping the swapping frequency from 1 interval to 10 intervals and the results show that setting the frequency to 5 delivers the best energy-efficiency for most workloads used in this study. Therefore, we fix the swapping frequency to 5, which essentially implies that we are comparing the "optimal" round-robin scheme against other schedulers.
- **Sample-Optimize-Symbios (SOS):** The SOS scheduler is originally proposed for the simultaneous multi-threading execution [32]. Many heterogeneous scheduling algorithms presented in prior works also fall into this category [5,17]. With this scheduling policy, the execution proceeds in a pattern consisting of three steps. First, at a scheduling point, the programs are executed on each type of core for an interval. This is called the "sampling phase" since the energy consumption of each assignment is available after this process. Second, the most energy-efficient scheduling is identified, thus this step is termed the "optimization phase". Finally, the execution will experience the "symbios phase" during which all programs are running N intervals with the optimal mapping. In this study, N is set to 10. Note that this strategy is also called "sampling" in a few prior works.
- **MLP-ratio:** This scheme is introduced in a recent work [6] aiming to improve the system throughput on heterogeneous CMPs. Although it does not focus on energy saving, it stands as one of the best heterogeneity-aware schedulers to date, thus deserving a comparison with our strategy. Note that the optimal scheduler proposed in [6] takes both the instruction-level parallelism (ILP) and the memory-level parallelism (MLP) into consideration. Nevertheless, the authors demonstrate that the algorithm based on only MLP-ratio delivers fairly close performance to their optimal scheduler. Considering the complexity to calculate the ILP on the fly, we implement a scheduling scheme based on only MLP estimation for the comparison due to its simplicity. In the MLP-ratio scheduler, the memory-level parallelism (MLP) ratios of all programs between the big and small cores are evaluated. Programs with higher MLP ratios are placed on the big cores while those with lower ratios are assigned to small cores.

- **PRIM:** At a scheduling point, the performance metrics collected from a pair of big and small cores are compared with the selective PRIM rules. If the conditions for both big and small cores are satisfied, the jobs on two cores are swapped; otherwise the current assignment is maintained. In case where the number of cores (programs) is greater than or equal to four, the optimal scheduling is achieved through a few steps of suboptimal assignments as described in Section 5.3.
- **Hierarchical PRIM (H-PRIM):** Instead of training a single PRIM model, we use CART to partition the training data into 4 categories according to the performance measurements and train a PRIM model for each subset. At a scheduling point, we first identify to which subset a pair of program executions belongs. We then compare the corresponding PRIM rules with the execution behaviors of these two programs and make scheduling accordingly.
- **Oracle:** In this scheduling policy, we assume that the scheduler knows the energy consumption of each program mapping in apriori and performs the optimal scheduling based on that information. Note that the "oracle" scheduler is implemented from the energy perspective. That is, at the end of each scheduling interval, we compare the possible program-core mappings for the next interval and pick the one which consumes the least energy.

7. Results

In this section, we perform a detailed evaluation of the rule-set guided scheduling algorithm by comparing it with a set of existing schemes.

7.1. Results of single-thread execution

7.1.1. Selective rules

We start the result demonstration by analyzing the extracted rules. By training PRIM models, we generate two sets of rules respectively for the big core and small core.

Rule set:
Big Core Rules:
L1D.nMiss > 37,510 && L1D.writeHit < 191,275 && nStall.SmallReg > 213,400
Small Core Rules:
L1D.nMiss < 45,600 && L2.nAccess > 32,200 && BR.misp < 27,733

As we mentioned in previous section, a matching between the observed execution behaviors and the corresponding rule sets implies that transferring the job to a different type of core is more energy-saving. Specifically, if the program is currently running on a big core and we observe that its cache access and pipeline stall statistics satisfy the big core conditions listed above, it should be moved to a small core for the execution in next interval. The two inequalities related to L1 data cache (*L1D.nAccess* and *L1D.writeHit*) indicates that the execution in the past interval issues considerable memory requests that go to the L1D, however many accesses are missed in this level of cache. The third condition shows that the pipeline is frequently stalled due to the shortage of free physical registers (*nStall.SmallReg*). Jointly, these three conditions indicate that the program may not be able to effectively utilize the computation resource on the big core and would be more suitable to run on a small core for better energy efficiency. Note that all the counter values are normalized to those in one million instructions (e.g., *L1D.nMiss* is actually *L1D.nMiss*/*MInst*). On the other hand, the rules corresponding to the small core imply that the program can achieve high speedup on the big core and result in better energy efficiency after migration. For instance, the relatively low miss rate in the L1 data cache and infrequent branch mispredictions means that the program is able to fully exploit the computing resource on the big core and more efficiently utilize the energy (i.e., executing with a lower EPI).

7.1.2. Effectiveness comparison

In this subsection, we compare the effectiveness of different scheduling policies on reducing the energy consumption. Fig. 7 demonstrates the comparison of energy for all selected programs running on a dual-core heterogeneous CMP when different strategies are engaged. Note that the results under all schemes are normalized to that corresponding to the big core execution. As can be observed, the selected programs manifest distinctive variation on the energy consumption. For the static schemes, applications including *equake*, *lbm*, *mcf*, and *milc* are more appropriate to run on the small core while benchmarks such as *wupwise*, *dealII* and *h264* are suitable candidates to be placed on the big core. This corroborates the conclusions drawn by few prior works that program features such as memory-intensity, computation-level and memory-level parallelism impact their relatively energy consumption on different types of cores [5,17,33], which further justifies the opportunities for intelligent scheduling on heterogeneous systems.

The round-robin scheme does not involve true scheduling intelligence either, since it just blindly transfers the program to a different core at a preset frequency. As a consequence, it coincidently results in lower energy consumption than the static scheme for some benchmarks while performing even worse for programs including crafty and eon. The SOS scheme is able to identify the correct task-to-core mapping via online sampling, thus leading to lower energy consumption than both static and R-R for many benchmarks. However, it suffers from two intrinsic drawbacks. First, frequent sampling introduces noticeable overhead which may prolong the execution time and consume extra energy that mitigates the benefit. Second, this scheduler assumes a continuum of program characteristics in the symbios stage (i.e., the following *N* intervals after sampling), which might not be true as the execution behaviors usually vary across different phases. This may cause inefficient executions in many intervals and thus raise the energy consumption. The PRIM rule set guided policy works the most closely to the oracle scheduler because it eliminates the unnecessary sampling over-head and transfers the job to the energy-saving core when necessary. In general, the round-robin, SOS, PRIM and oracle schedulers are able to reduce the energy consumption respectively by 3.4%, 12.8%, 20.1% and 22.7% compared to the execution on big core.

The unnecessary context switch is an important cause of inefficient execution on heterogeneous CMPs when non-ideal schedulers are employed. Those context switches introduce substantial overhead due to architectural state retrieving and cache re-warming to the execution; furthermore, they transfer programs to the inappropriate cores for execution, which may adversely increases the energy consumption. Taking this into consideration, we collect the number of context switches during the execution for each application when different schedulers are used. Table 3 lists the recorded statistics. The round-robin scheme causes much more switches than all other policies since it continually moves a job every 5 intervals (recall the experimental set up described in Section 6.2). The SOS scheme makes a job migration decision based on the sampling result and thus usually moves jobs at a much lower frequency, which in turn significantly reduces the switch times. The PRIM and the oracle scheduler generally lead to comparable context switches as SOS does; however, they capture the migration opportunities more precisely and make scheduling

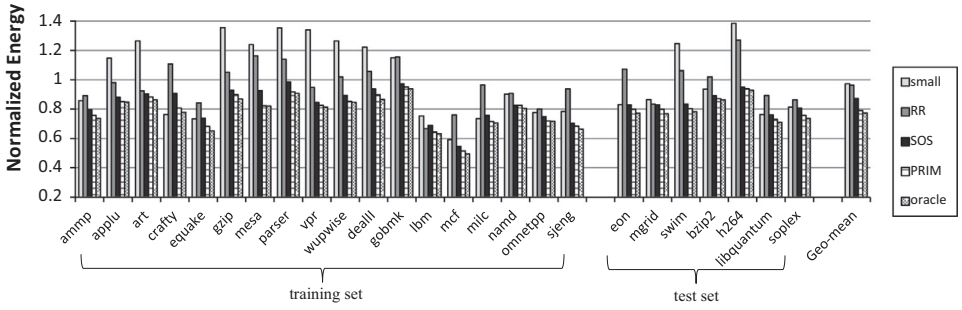


Fig. 7. Normalized energy consumption for single-programs executing on a dual-core CMP with different schedulers.

decisions at finer-granularity, thus appearing to be more energy-saving compared to SOS. Also, since the PRIM scheduler does not require sampling, it offers better performance than the SOS scheme. Fig. 8 shows the average performance of each program normalized to the big core's execution. Not surprisingly, the PRIM scheduler results in only 20% longer execution time than the static-big policy, delivering better performance than both RR and SOS which respectively prolong the running time by 69% and 29%.

Although the 20% performance overhead may seem relatively high considering that the energy-efficiency improvement is 20.1%, this is reasonable because we are reporting the geometric mean value for the program collection, which means that a few samples with relatively high performance degradation may bias the average result and show large average loss. For example, programs *h264* and *art*, whose performance is fairly sensitive to the core type, show large performance degradation compared the execution on the big core. On the other hand, for most programs such as *ammp*, *crafty*, *mcf*, *milc* and *namd*, the PRIM based approach leads to impressive energy saving with low performance degradation. As we will show in the following section, for multi-program workloads, PRIM (and the enhanced version H-PRIM) guided scheduling leads to remarkable energy saving with less than 5% performance degradation. Therefore, we believe it is safe to claim that the proposed PRIM rule guided scheduling is able to significantly reduce the energy consumption with acceptable performance loss.

Table 3
Number of Context Switches.

Benchmark		RR	SOS	PRIM	Oracle
Training	*ammp*	1084	201	194	189
	applu	486	110	127	140
	art	466	107	89	83
	crafty	384	98	93	101
	equake	874	162	177	182
	gzip	658	149	106	96
	mesa	336	58	69	71
	parser	516	193	102	123
	vpr	464	122	97	103
	wupwise	298	44	58	72
	dealII	354	102	118	105
	gobmk	464	121	102	99
	lbm	734	101	121	115
	mcf	1268	406	387	367
	milc	1208	104	97	89
	namd	358	76	64	58
	omnetpp	382	152	125	104
	sjeng	496	101	95	98
Test	*eon*	360	89	103	99
	mgrid	959	120	98	88
	swim	980	103	105	95
	bzip2	1072	82	105	102
	h264	304	98	75	72
	libquantum	322	60	73	70
	soplex	496	93	120	114

7.2. Results of multi-program execution

In this subsection, we demonstrate the evaluation results for the second circumstance, where each workload contains as many programs as cores. Again, we analyze the extracted PRIM rules at first.

7.2.1. Selective rules

By training a PRIM model, we can generate the following rule sets to guide the scheduling for two programs running on a pair of big and small cores:

Rule set:
Big Core Rules:
L1D.nMiss < 13,430 && nStall.SmallIQ > 256,949 && nFetch > 4,879,317
Small Core Rules:
BP.nMiss > 9674 && iLoad.count > 198,169 && iALU.count < 400,026 && L1D.nMiss < 17,701

At a scheduling point, the scheduler compares the performance metrics collected from a pair of big and small cores. If the measurements on both sides are satisfied with these rules, the scheduler predicts that swapping the two programs will decrease the total energy consumption.

Since our prediction is made at each scheduling interval, minimizing the energy consumption essentially translates to lowering down the total power of that period. Also, the big cores always consume much larger power than the small ones, thus dominating the total power consumption all the time. We present these two statements to assist the interpretation of the PRIM rules. Note that we will analyze the correlation between execution behaviors and power consumption including both dynamic power and leakage power. Let us focus on the big core rules at first. As can be seen, the big core rules suggest that a program with a low L1 cache miss rate (*L1D.nMiss*), a high instruction fetch rate (*nFetch*), and substantial internal stalls (e.g., stalls due to small instruction issue queue, or *nStall.smallIQ*) should be exchanged to the small core for execution. It is straightforward to understand the metrics related to L1 cache and instruction fetch. A Low L1 cache miss rate and high fetch speed indicates that the current program on the big core is executed at relatively high speed without suffering from frequent cache misses. However, from the power perspective, this implies large dynamic power on many function units due to high activities. As for the second condition, a large *nStall.smallIQ* value implies persistent high utilization on the issue queue. This also increases dynamic power consumption on this component because of

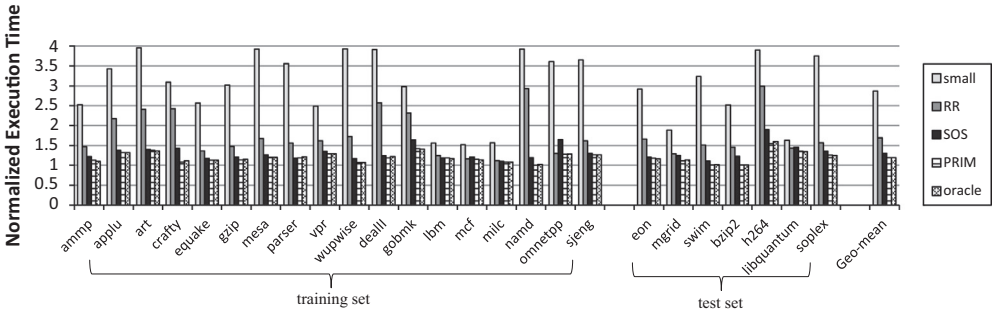


Fig. 8. Normalized execution time for single-programs executing on a dual-core CMP with different schedulers.

frequent operations such as checking the operands status. On the other hand, components including IQ and integer ALU tend to become the hotspot on die. As a consequence, the leakage power on these units is rapidly increasing since it is positively related to the temperature. In one word, the big core rules outline the features of intervals which tend to consume both high dynamic and leakage power. For the purpose of energy saving, these intervals should be migrated to the small core for execution.

We now shift our concentration to the small core rules. Recall that the total power consumption is dominated by the big core. Therefore, the rules for the small core essentially characterize the execution phases that are not likely to result in extreme high power on a big core. Specifically, the first and the third conditions respectively set constraints for the occurrences of branch mispredictions (*BP.nMiss*) and number of integer ALU instructions (*iALU.count*). A branch misprediction will lead to a pipeline flush and lower down the execution speed. Fewer number of ALU instructions can alleviate the utilization on ALUs, reducing the dynamic power and cooling down the component accordingly. These two conditions jointly reduce the power consumed by the core running this program. On the other hand, the rule sets require that the number of load instructions (*iLoad.count*) should be no smaller than a certain value while the misses in L1 data cache (*L1D.nMiss*) cannot go beyond a threshold. These two conditions imply that this program potentially issues a large amount of memory requests, but most of them can be served by the L1 cache. Nevertheless, the stress on L1 cache will not significantly increase the total power consumption since the L1 cache consumes relatively small power com-pared to other components. In general, the intervals filtered by the small core rule set tend to result in moderate power if executed on the big core, thus reducing the total power consumption.

7.2.2. Hierarchical PRIM

As described in Section 4.2, CART is able to partition the entire data set into several subsets, each of which contains similar samples. Therefore, if we train an individual PRIM model for each subset, the effectiveness of our strategy is expected to be increased due to the similarity of instances within the same subset. Taking this into account, we use CART to perform a data segmentation operation in prior to the PRIM model training. Fig. 9 demonstrates the segmentation result for all training instances. As can be seen, the entire data set are partitioned into 4 categories, as represented by the 4 leaf nodes on the generated tree. Each branch represents a condition on the performance metrics on a big or small core and is expressed in a form of "$X_i \geqslant$ (or $\leqslant$) M", where X_i denotes a performance metric and M denotes a value to segment the data set. Specifically in the tree shown in Fig. 9, *X126* indicates the number of branch mispredictions on the big core (*BP.nMiss*) and *X232* corresponds to the *iALU.count* metric which records the number of integer ALU instructions on the small core. *X244* tracks the number

Fig. 9. Data segmentation result from CART.

of fetched instruction on the small core (*nFetch*). The value at each leaf node is the average of CART response for that partition.

We train a PRIM model for each data segment and list their respective rule sets in Table 4. Note that for the fourth subset (i.e., the rightmost leaf node in Fig. 9), more than 95% of the included samples maintain a flag "1", meaning that the majority of this partition are candidates for job swap. Consequently, we do not train an extra PRIM model for this subset and directly use its branch conditions to guide the scheduling. During the execution, this tree is accessed at each scheduling point in order to classify a program pair into an appropriate subset. The access starts from the root node of the tree. If the condition is satisfied after the variable comparison (*X126*, or *BP.nMiss*), the access will proceed to the left child; otherwise it goes to the right child. This process is performed again on the child node and the program pair will be classified to a specific subset thereafter. The corresponding PRIM rules are then compared with the execution behaviors and make job assignments accordingly. In particular, if a program pair falls into the rightmost subset, the scheduler will immediately swap the two jobs for execution in the next interval.

7.2.3. Effectiveness comparison

In this subsection, we compare the effectiveness of different scheduling policies on reducing the energy consumption and improving energy efficiency. To demonstrate the scalability of our proposed algorithm, we run four-program workloads on two types of heterogeneous architectures: a platform with an identical number of big and small cores (2B + 2S) and a system with non-equivalent numbers of big and small cores (3B + 1S).

Fig. 10(a) illustrates the energy reduction for these workloads on the first platform (2B + 2S) when distinct schedulers are engaged. Note that all results are compared with those corresponding to the static scheduling case. The workloads are sorted in ascending order according to the degree of energy saving. As can be observed, our PRIM and H-PRIM strategies always outperform other scheduling algorithms with respect to energy consumption.

Table 4
PRIM rules for each data subset.

Segment	Big core rules	Small core rules	Performance metrics description
Subset 1	*LDSTUnit.util* < 0.27 && *nStall.noCachePort* < 10,785	*L1I.nMiss* > 3791	*LDSTUnit.util*: the utilization of load/store unit *nStall.noCachePort*: cumulative stall cycles due to cache port contention *L1I.nMiss*: number of misses in L1I cache
Subset 2	*IQ.avgFree* < 3 && *L1I.nHit* > 817,392	*avgBranchPenalty* > 73 && *iComplex.count* < 3921	*IQ.avgFree*: the average number of free entries in IQ, indicating the IQ utilization *L1I.nHit*: number of hits in L1I cache *avgBranchPenalty*: average penalty (in cycles) of branch misprediction *iComplex.count*: number of integer complex instructions, such as division
Subset 3	*nStall.noCachePort* < 44,750 && *nStall.smallREG* > 851,493	*L1I.avgMissLat* < 513	*nStall.smallREG*: cumulative stall cycles due to available registers *L1I.avgMissLat*: average penalty (in cycles) of a miss in L1I cache
Subset 4	N/A		

This is because that the rule-set guided scheduler is capable of effectively identifying the most appropriate program assignment at runtime to minimize the energy consumption. Furthermore, the total energy consumed by H-PRIM is fairly close to the oracle case (i.e., the minimum), since the data segmentation increase the accuracy of identifying the candidate intervals. Other schedulers suffer from distinctive drawbacks which adversely impact their effectiveness. The previous subsection explains the disadvantage of the R-R and SOS scheduler. The MLP-ratio algorithm, on the other hand, aims to improve the system performance. As we demonstrated in Section 1, this scheduler can increase the total energy consumption for some intervals, thus trailing our strategies in saving the total energy.

Fig. 10(b) shows the performance improvement when the workloads are respectively running with these schedulers. Note that the workloads are sorted according to the performance gain in this figure. Also recall that the oracle scheduler is the optimal with respect to the energy consumption instead of performance. As can be seen, the MLP-ratio strategy always leads to better performance (i.e., positive value) compared to the baseline. This does not go beyond our expectation because the goal of this scheduler is to enhance the overall performance, thus the programs are assigned in a manner to maximize the execution speed. On the other hand, both performance improvement and degradation (i.e., negative value) are observed in other scheduling policies. The performance loss mainly stem from two sources, namely migration overhead and slower execution in certain intervals. Our scheduler eliminates unnecessary job swaps during the execution compared to round-robin and SOS, thus delivering better performance.

Fig. 10(c) demonstrates the average performance gain, energy saving and ED reduction for all workloads. From the performance respective, MLP-ratio stands as the optimal by accelerating the execution by 5.7% while PRIM and H-PRIM respectively enhance the performance by 3.9% and 4.1%. Note that the MLP-ratio scheduler is more effective for scheduling-sensitive workloads [8]. However, the performance gains for applications demonstrating less sensitivity to program assignment are fairly modest. Therefore in general, our schedulers lead to comparable performance to MLP-ratio on average. For energy saving, the PRIM and H-PRIM algorithms are able to reduce the energy consumption by 11.8% and 14.8% compared to 16.3% delivered by the oracle scheduler. Finally, for the ED metric, the PRIM and H-PRIM algorithms respectively reduce its value by 15.3% and 17.9% while the oracle scheduler can decrease the product by 19.1%. The SOS and MLP-ratio policies lead to less impressive savings. In other words, our best scheduler H-PRIM outperforms the MLP-ratio policy, which is one of the optimal state-of-the-art heterogeneous schedulers by 7.8% and 5.7%, respectively on energy and ED. We also collect the number of context switches with different

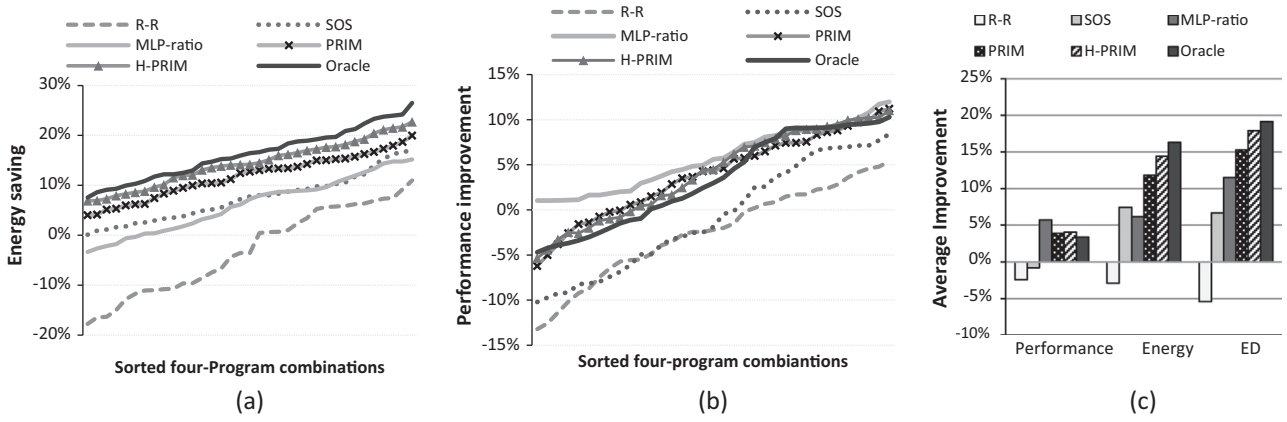


Fig. 10. Evaluation results of four-program workloads running on a 2B + 2S platform: (a) energy saving, (b) performance improvement, (c) average improvement for energy, performance and ED.

scheduling policies and observe similar trend as shown in Table 3. We do not list the complete results due to space limitation.

Fig. 11 demonstrate the results for quad-program workloads running on a CMP consisting of three big cores and a small core (3B + 1S). As can be seen, the general trends of the curves are similar to those shown in the 2B + 2S case. On average, the PRIM and H-PRIM scheduler is capable of saving energy by 11.5% and 13.9%. For the energy-delay product, these two schemes decrease the value by 14.9% and 17.3%. This implies that compared to MLP-ratio, the H-PRIM policy reduces the total energy and ED by 8.1% and 5.5%, respectively.

We also evaluate the effectiveness of our strategy with other configurations. Fig. 12 shows the average improvement when two-program workloads are running on a CMP with a big and a small core. Not-surprisingly, H-PRIM surpasses other policies by improving the energy-efficiency most closely to the oracle scheduler. More results (e.g., 1B + 3S, etc.) are not shown due to space limitation. Nevertheless, our evaluation results demonstrate that the proposed rule-set guided scheduling policy is more effective in optimizing the energy-efficiency of single-ISA heterogeneous platforms compared to existing schedulers.

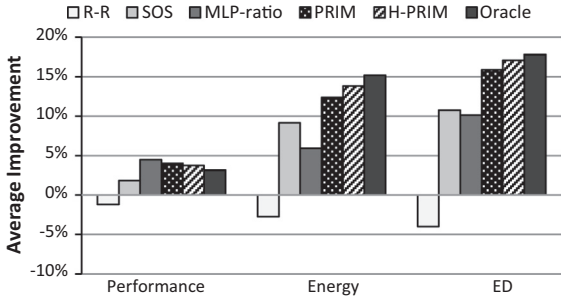


Fig. 12. Average improvement for performance, energy and ED for two-program workloads running on a 1B + 1S platform.

7.3. Pareto analysis

The energy efficiency depends on the application scheduling on the system, which is essentially determined by a variety of program execution behaviors. In other words, making a scheduling decision can also be solved as a multi-cause problem. Pareto analysis is a statistical technique handling multi-cause problems and has been extensively used in many research domains. In this subsection, we concentrate on a single-program execution to exemplify the usage of Pareto analysis in the scheduling study and compare the result with PRIM.

Fig. 13 shows the Pareto frontiers together with our PRIM results for a SPEC 2000 benchmark *eon* when it is executed on a small core. In a multi-objective optimization problem such as minimizing both energy and delay in our case, a Pareto optimization changes the parameters to improve at least one metric (e.g. reduce energy) without negatively impacting any other metric (e.g. increasing the delay). A Pareto frontier is the one when no further Pareto optimization can be implemented. In the figure below, all the points including those small dots on the background represent 1000 possible designs, each of which corresponds to an execution interval. The Pareto frontiers are those stars scattered on the boundary at the bottom. The upward triangles, which are clustered

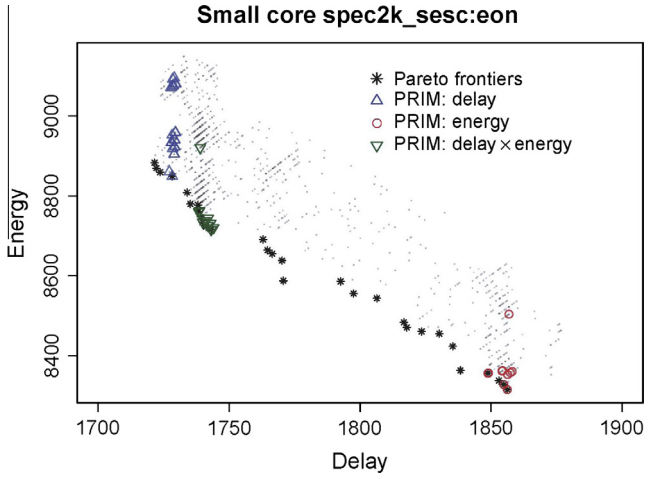


Fig. 13. Pareto analysis along with PRIM illustration for *eon* running on a small core.

on the left, are the designs by the PRIM rule which minimizes the delay. The circles clustered at lower right corner are the designs identified by the PRIM rule for energy minimization. The downward triangles which mostly coincide with the Pareto frontiers are the designs selected by the PRIM rule which minimizes the product of energy and delay. As can be observed from the figure, PRIM successfully identifies the Pareto frontiers by a set of simple rules, which further confirms the effectiveness and accuracy of the PRIM-based approach. Therefore, making scheduling decision

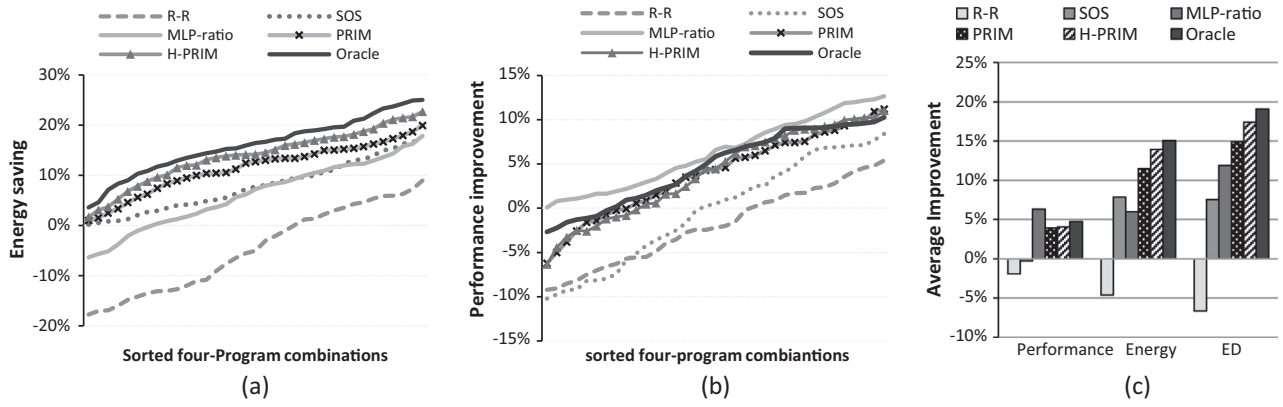


Fig. 11. Evaluation results of four-program workloads running on a 3B + 1S platform: (a) energy saving, (b) performance improvement, (c) average improvement for energy, performance and ED.

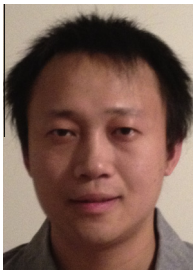
based on the PRIM rule set is able to deliver fairly high energy-efficiency.

8. Conclusion

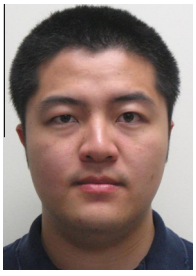
In this paper, we propose a scheduling strategy for energy-efficient execution on single-ISA heterogeneous chip-multiprocessors. We demonstrate that performance-oriented scheduling may lead to executions that are not sufficiently energy-efficient. Due to this limitation, we concentrate on energy saving and introduce a rule-set guided scheduling to exploit the optimal energy efficiency on heterogeneous CMPs. We employ advanced statistical tools including PRIM and CART to facilitate the development of our algorithm. The evaluation results show that our proposed algorithm impressively outperform existing scheduling schemes by minimizing the energy consumption, thus delivering better energy efficiency.

References

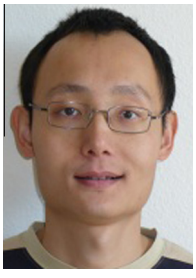
[1] ARM Corporation, Big.Little Processing. <http://www.arm.com/products/processors/technologies/bigLITTLEprocessing.php>.
[2] Nvidia Corporation, Tegra 3 super chip processors. <http://www.nvidia.com/object/tegra-3-processor.html>.
[3] Intel 64 and IA-32 Architectures Software Developers Manual Volume 3: System Programming Guide, October 2011.
[4] S. Balakrishnan, R. Rajwar, M. Upton, K. Lai, The impact of performance asymmetry in emerging multicore architectures, in: ISCA, June 2005.
[5] M. Becchi, P. Crowley, Dynamic thread assignment on heterogeneous multiprocessor architectures, in: CF, May 2006.
[6] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Wadsworth Press, 1984.
[7] J. Chen, L.K. John, Efficient program scheduling for heterogeneous multi-core processors, in: DAC, June 2009.
[8] K.V. Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, J. Emer, Scheduling heterogeneous multi-cores through performance impact estimation, in: ISCA, June 2012.
[9] S. Eyerman, L. Eeckhout, A memory-level parallelism aware fetch policy for SMT processors, in: HPCA, February 2007.
[10] J. Friedman, N. Fisher, Bump hunting in high-dimensional data, Stat. Comput. 9 (1999).
[11] S. Hao, Q. Liu, L. Zhang, J. Wang, Process scheduling on heterogeneous multi-core architecture with hardware support, in: NAS, June 2011.
[12] T. Heath, B. Diniz, E.V. Carrera, W. Meria Jr., R. Bianchini, Energy conservation in heterogeneous server clusters, in: PPoPP, June 2005.
[13] M. Goraczko et al., Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems, in: DAC, June 2008.
[14] R. Grant, A. Afsahi, Power-performance efficiency of asymmetric multiprocessors for multi-threaded scientific applications, in: The 2nd Workshop on High-Performance, Power-Aware Computing, with IPDPS, April 2006.
[15] S. Kaxiras, M. Martonosi, Computer Architecture Techniques for Power Efficiency, Morgan and Claypool Publishers, 2008.
[16] D. Koufaty, D. Reddy, S. Hahn, Bias scheduling in heterogeneous multi-core architectures, in: EuroSys, April 2010.
[17] R. Kumar, K.I. Farkas, N.P. Jouppi, P. Ranganathan, D.M. Tullsen, Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction, in: MICRO, December 2003.
[18] R. Kumar, D.M. Tullsen, P. Ranganathan, N.P. Jouppi, K.I. Farkas, Single-ISA heterogeneous multi-core architectures for multithreaded workload performance, in: ISCA, June 2004.
[19] N.B. Lakshminarayna, J. Lee, H. Kim, Age based scheduling for asymmetric multiprocessors, in: SC, November 2009.
[20] C. Li, C. Ding, K. Shen, Quantifying the cost of context switch, in: ExpCS, June 2007.
[21] S. Li et al., McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures, in: MICRO'09.
[22] T. Li, D. Baumberger, D.A. Koufaty, S. Hahn, Efficient operating system scheduling for performance-asymmetric multi-core architectures, in: SC, November 2007.
[23] T. Li, P. Brett, R. Knauerhase, D. Koufaty, D. Reddy, S. Hahn, Operating system support for overlapping-ISA heterogeneous multi-core architectures, in: HPCA, January 2010.
[24] P. Radojkovic et al., Optimal task assignment in multithreaded processors: a statistical approach, in: ASPLOS, March 2012.
[25] J.C. Saez, A. Fedorova, M. Prieto, S. Blagodurov, A comprehensive scheduler for asymmetric multicore systems, in: EuroSys, April 2010.
[26] J.C. Saez, A. Fedorova, D. Koufaty, M. Prieto, Leveraging core specialization via OS scheduling to improve performance on asymmetric multicore systems, in: ACM Transactions on Computer Systems, April 2012.
[27] J.C. Saez, D. Shelepov, A. Fedorova, M. Prieto, Leveraging workload diversity through OS scheduling to maximize performance on single-ISA heterogeneous multicore systems, J. Parallel Distr. Comput. 71 (7) (2011).
[28] J. Renau et al., SESC Simulator.
[29] E. Saad, M. Awadalla, M. Shalan, A. Elewi, Energy-aware task partitioning on heterogeneous multiprocessor platforms, in: The International Journal of Computer Science Issues (IJCSI), vol. 9, 2012.
[30] S. Sharifi, A.K. Coskun, T.S. Rosing, Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs, in: ASPDAC, January 2010.
[31] K. Singh, M. Bhadauria, S.A. Mckee, Prediction-based power estimation and scheduling for CMPs, in: ICS (Extended Abstract and Poster Presentation), June 2009.
[32] A. Snavely, D.M. Tullsen, Symbiotic job scheduling for a simultaneous multithreading processor, in: ASPLOS, November 2000.
[33] S. Srinivasan, R. Iyer, L. Zhao, R. Illikkal, HeteroScouts: hardware assist for OS scheduling in heterogeneous CMPs, in: Poster session of the ACM SIGMETRICS, June 2011.



Ying Zhang received his Bachelor's and Master's degree in Electronics and Information Engineering from Huazhong University of Science and Technology, China, and the Ph.D. degree in Electrical and Computer Engineering from Louisiana State University. He is currently working as a performance architect in Intel, responsible for modeling and optimizing the performance of Intel's server processors. He has wide research interests in the computer architecture area including heterogeneous system design and optimization, GPU performance characterization, and hard-error reliable processor design.



Lide Duan is currently an Assistant Professor in the Department of Electrical and Computer Engineering at The University of Texas at San Antonio. Prior to joining UTSA, he worked as a senior CPU design engineer at AMD, working on future x86 based high performance and low power CPU microarchitecture design and performance modeling. He received a PhD in Computer Engineering from Louisiana State University in 2011. His PhD research focused on soft error reliability analysis and prediction for processors at computer architecture level. He also received a bachelor's degree in Computer Science from Shanghai Jiao Tong University, China, in 2006.



Bin Li received his Bachelor's degree in Biophysics from Fudan University, China. He obtained his Master's degree in Biometrics (08/2002) and Ph.D. degree in Statistics (08/2006) from The Ohio State University. He is an Associate Professor with the Experimental Statistics department at Louisiana State University. His research interests include statistical learning & data mining, statistical modeling on massive and complex data, and Bayesian statistics. He received the Ransom Marian Whitney Research Award in 2006 and a Student Paper Competition Award from ASA on Bayesian Statistical Science in 2005. He is a member of the Institute of Mathematical Statistics (IMS) and American Statistical Association (ASA).

Lu Peng is currently an Associate Professor with the Division of Electrical and Computer Engineering at Louisiana State University. He received the Bachelor's and Master's degrees in Computer Science and Engineering from Shanghai Jiao Tong University, China. He obtained his Ph.D. degree in Computer Engineering from the University of Florida in Gainesville in April 2005. His research focus on memory hierarchy system, reliability, power efficiency and other issues in CPU design. He also has interests in Network Processors. He received an ORAU Ralph E. Powe Junior Faculty Enhancement Awards in 2007 and a Best Paper Award from IEEE International Conference on Computer Design in 2001. He is a member of the ACM and the IEEE Computer Society.

Srinivasan Sadagopan is a Senior Member of Technical Staff in the Client Performance Architecture and Modeling Group at Advanced Micro Devices. His research interests include computer architecture, workload characterization, performance evaluation, power management and projection. He has a PhD in Computer Engineering from the University of Maryland at College Park. He is a member of the ACM SIGARCH.