An Opportunistic Prediction-based Thread Scheduling to Maximize Throughput/Watt in AMPs

Arunachalam Annamalai, Rance Rodrigues, Israel Koren and Sandip Kundu
Department of Electrical and Computer Engineering, University of Massachusetts at Amherst
Email: {annamalai, rodrigues, koren, kundu}@ecs.umass.edu

***Abstract*—The importance of dynamic thread scheduling is increasing with the emergence of Asymmetric Multicore Processors (AMPs). Since the computing needs of a thread often vary during its execution, a fixed thread-to-core assignment is sub-optimal. Reassigning threads to cores (thread swapping) when the threads start a new phase with different computational needs, can significantly improve the energy efficiency of AMPs. Although identifying phase changes in the threads is not difficult, determining the appropriate thread-to-core assignment is a challenge. Furthermore, the problem of thread reassignment is aggravated by the multiple power states that may be available in the cores. To this end, we propose a novel technique to dynamically assess the program phase needs and determine whether swapping threads between core-types and/or changing the voltage/frequency levels (DVFS) of the cores will result in higher throughput/Watt. This is achieved by predicting the expected throughput/Watt of the current program phase at different voltage/frequency levels on all the available core-types in the AMP. We show that the benefits from thread swapping and DVFS are orthogonal, demonstrating the potential of the proposed scheme to achieve significant benefits by seamlessly combining the two. We illustrate our approach using a dual-core High-Performance (HP)/Low-Power (LP) AMP with two power states and demonstrate significant throughput/Watt improvement over different baselines.**

***Index Terms*—Asymmetric Multicore Processor (AMP); dynamic thread scheduling; Hardware Performance Counters (HPCs); phase detection; throughput/Watt prediction**

I. Introduction

Advancements in technology has resulted in improved transistor performance and the ability to pack more transistors into a smaller area. The increased device density and rising frequency led, unfortunately, to a power density problem in processor ICs which paved the way for the multicore era [1]. Most current multicore processors consist of many symmetric cores (SMP) with more modest computational capabilities that are suited for thread level parallelism (TLP). But, performance suffers whenever sequential applications with high instruction level parallelism (ILP) are encountered [2].

Asymmetric Multicore Processors (AMP) with the capability to cater to the diverse needs of workloads were introduced as a potential solution to this conundrum. They have been shown to outperform their SMP counterparts within a given area and power budgets [3]–[6]. Often, these cores are of two types: big and small. The big cores provide higher performance while the smaller ones are more power efficient. However, the benefits of AMPs are highly dependent on a proper thread-to-core assignment and a non-optimal assignment may even make them worse than SMPs.

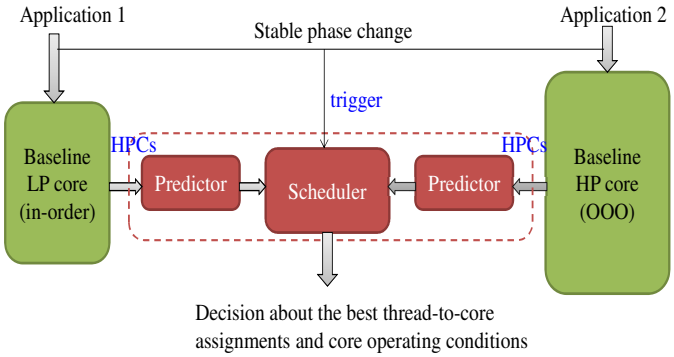

Fig. 1. High-level view of the proposed thread scheduling scheme.

Further, the current multicores widely employ Dynamic Voltage and Frequency Scaling (DVFS) to minimize energy consumption [7] and/or maximize performance [8], [9]. For example, a memory bound application typically does not have sufficient ILP to keep the core busy while waiting for the long-latency memory accesses to complete. Reducing the voltage and/or clock frequency of the core in such a case results in a cubic power reduction without greatly impacting the overall performance [7]. Intel's Turbo Boost technology enhances the performance of a high-performing core through dynamic voltage and frequency boosting when the other cores are inactive [8]. Therefore, there is a need to adapt the operating conditions (voltage/frequency levels) of the cores to match the time-varying program behavior to increase throughput/Watt.

To this end, we propose a novel prediction-based dynamic scheme that holistically addresses the above two requirements of determining the best thread-to-core assignment and core operating conditions to suit the current program phase. By opportunistically making thread scheduling decisions only when a new phase is encountered, we keep the associated overheads at bay. The central idea of our proposal is the online prediction of the expected throughput/Watt of the current phase on all the other core-types in the AMP and at different voltage/frequency levels, while it is being executed on the current core. Thus, by examining all the alternatives, our scheme makes an informed decision about the thread-to-core mapping and the core operating conditions. The prediction is made possible by employing hardware performance counters (HPCs) of the current core. A relationship is established between the values of these counters in the core executing the application and the expected throughput/Watt of this application if it would run on the other cores in the AMP and at different voltage/frequency

978 -1-4799-1021-2/13/$31.00 ©2013 IEEE

levels. By making a decision about the best core-type and the operating conditions, our scheme can closely track phase changes in the applications. The available compute resources are then matched to the threads' requirements resulting in increased throughput/Watt.

To illustrate our approach, we consider a dual-core AMP comprising of a 4-way issue out-of-order high performance core (HP core), and a 2-way issue in-order low power core (LP core). Our choice of the asymmetric cores is in line with recent studies [10]–[12]. The two baseline cores operate at different frequencies which is the reason for choosing throughput/Watt as the optimization metric. Figure 1 presents a high-level view of the proposed approach. Whenever a new stable phase is encountered by any of the threads, the proposed scheme determines the best thread assignment based on the predicted throughput/Watt. We compare the proposed scheme against a static baseline (the same dual core AMP with no thread swapping or DVFS capabilities), a baseline with *swap-only* capability, a baseline with *DVFS-only* capability and a greedy oracular scheduler. Our results indicate that the proposed scheme achieves significant throughput/Watt and throughput benefits over the non-oracular baselines.

The key contributions of this paper are:

1) A novel prediction-based dynamic thread scheduling scheme for AMPs to determine the best thread-to-core assignments and core operating conditions at runtime.
2) A mechanism to accurately predict the expected throughput/Watt of the current program phase if it would run on other core-types in the AMP and at different voltage/frequency levels.
3) An efficient heuristic to determine the optimal number and choice of performance counters to estimate throughput/Watt.

II. Related Work

The prior thread scheduling schemes can be broadly classified into those that employ offline profiling, online learning via sampling and online estimation.

Khan *et al.* [13] propose regression analysis along with phase classification to identify thread to core affinity. Shelepov *et al.* [14] profile applications offline to determine architectural signatures based on cache misses. In [15], Chen *et al.* use multi-dimensional curve fitting to determine the optimal thread to core assignment for AMPs. All these approaches rely on offline profiling and are not practical, since they require knowledge of the workloads that will be run on the multicore.

Online learning-based schemes offer a more practical solution to the AMP scheduling problem. Kumar *et al.* [3] proposed an AMP consisting of cores of various sizes. Whenever a new program phase is detected, sampling is initiated and the core which provides the best power efficiency is chosen. A similar approach was proposed by Becchi *et al.* [10], where an optimal thread scheduling was determined by forcing a thread swap between baseline cores. Although these schemes are a practical alternative, it is clear that with an increase in the number of core types in the system, the number of samples for each phase detected will be large resulting in a significant overhead.

Online estimation-based schemes [12], [16], [17] are an improvement over the learning schemes since they avoid sampling and the associated overhead. Here, based on the current characteristics of a workload being executed, its performance on other core types is estimated. Saez *et al.* [12] propose a comprehensive scheduler for AMPs by estimating the performance on each core-type based on last level cache miss rate. It is, however, unclear whether using L2 misses alone is sufficient to make thread to core assignment decisions such that performance/power is optimized. The work closest to ours is that proposed by Srinivasan *et al.* [16], Koufaty *et al.* [11] and Rodrigues *et al.* [18]. In [16], Srinivasan *et al.* estimate the performance of the thread currently running on one core type, on another core, using a closed form expression. These expressions were developed for specific cores and a general approach was not provided. Koufaty *et al.* [11] determine thread to core mapping in an AMP, using program to core bias which is estimated online using the number of external and internal stalls. In both of these papers, the objective is only performance. Extending the above techniques to improve throughput/Watt is not straightforward. Rodrigues *et al.* [18] presented an estimation-based thread scheduling scheme using HPCs to improve performance/Watt. However, they only considered cores operating at the same voltage/frequency which simplifies the scheduling problem. Further, their scheme suffers from a high estimation error (average IPC/Watt estimation error of about 34%). Annamalai *et al.* [4], [5] considered thread scheduling in an AMP by using predetermined rules but the cores that they consider are very specific and the extension to other types of cores is unclear.

Most of the earlier scheduling schemes focus mainly on performance and, they do not take into account the multiple voltage/frequency levels that may be available within the cores. To the best of our knowledge, our proposed prediction-based scheme is the first of its kind which makes decision about both the thread-to-core assignments and core operating conditions with the objective of maximizing the overall throughput/Watt.

III. Methodology

To illustrate our approach (detailed in the next two sections), we selected a dual-core AMP consisting of two core types at the two ends of the power/performance spectrum - a low-power core (LP) and a high-performance core (HP). This is one of the worst cases for a scheme for predicting the throughput/Watt on the HP core based on the activities observed in the LP core and vice versa. Furthermore, the considered dual-core AMP would stress the importance of swapping threads between the cores or dynamically changing voltage/frequency levels to adapt to the time-varying program characteristics. For example, a thread currently being executed on the HP core may have entered a low-ILP program phase, during which it is better to either run this thread on the LP core or reduce the voltage/frequency of the HP core to save power.

TABLE I
CHOSEN CORE PARAMETERS

Param	LP	HP	Param	LP	HP
Issue	2	4	Type	In-order	OOO
INT/FP REG	64/64	96/80	LSQ	NA	32
INT/FP ISQ	NA	36/24	ROB	NA	128
L1(I/D)	32K	32K	L2	512K	2M

TABLE II
EXECUTION UNIT SPECIFICATIONS FOR THE CORES. (P - PIPELINED, NP - NOT PIPELINED, PP - PARTIALLY PIPELINED)

Core	FP DIV	FP MUL	FP ALU
LP	1 unit, 60 cyc, NP	1 unit, 4 cyc, PP	1 unit, 5 cyc, P
HP	1 unit, 21 cyc, P	1 unit, 5 cyc, P	2 units, 3 cyc, P
	INT DIV	INT MUL	INT ALU
LP	1 unit, 207 cyc, NP	1 unit, 10 cyc, P	2 unit, 1 cyc, P
HP	1 unit, 23 cyc, P	1 unit, 8 cyc, P	4 units, 1 cyc, P

The list of core parameters and execution latencies used for both the core types are shown in Tables I and II, respectively. Most of the core parameters and latencies were taken from [19]. It can be seen from Table I that the two cores are significantly different. Similar to the latest Intel and AMD processors [8], [9], the two baseline cores can operate either in normal or boost mode and, the corresponding voltage and frequency levels in the two modes are shown in Table III.

We used SESC as our architectural performance simulator [20] and employed Wattch [21] and CACTI [22] to calculate power with modifications to account for static power. For our experiments, we have selected 38 benchmarks from MiBench [23], SPEC suite [24] and Mediabench [25] suites.

IV. DYNAMIC THREAD SCHEDULING

The proposed dynamic thread scheduling scheme strives to maximize throughput/Watt of the applications by determining the program phase to core affinity and the best core operating conditions at runtime. As the knowledge about the computational needs of different program phases are generally unavailable beforehand, there is a need to determine them online. To keep the swapping and DVFS overheads at bay, a good thread scheduling scheme should consider reassigning threads and/or changing voltage/frequency levels only when a thread has moved to a new and stable phase. Therefore, there is a need to detect stable phase changes in a program even before determining the best thread-to-core affinity or the appropriate power state (voltage/frequency levels). The program phase detection mechanism should ignore short-lived unstable phases that do not warrant thread reassignment or change in core operating conditions. We describe our phase change detection mechanism in the next subsection.

A. Phase detection mechanism

A number of phase classification mechanisms have been proposed in the literature [26], [27]. After certain modifications, we adopt the phase classification scheme based on Instruction Type Vectors (ITVs) proposed by Khan *et al.* [13] owing to its simplicity. In their scheme, ITVs are formulated using hardware counters that count the number of committed instructions of certain types (9 in [13]) during a specified interval. A fixed number n of committed instructions constitute the above interval, with the value of n to be determined. The appropriate instruction counter is incremented whenever an instruction is retired. After the commit of n instructions, the resulting 9-element vector is captured and compared to the ITV of the previously identified phase. If the sum of differences between the instruction types of the previously encountered and currently executing phase is greater than a threshold, Δ (another parameter to be determined), then this is potentially a new phase. The scheme qualifies a newly detected phase as stable only when at least m (the last phase classification parameter that should be determined) consecutive intervals have their ITV differences smaller than Δ. All previously detected stable phases are stored in a phase table. Since we are interested only in phase change detection, rather than classification, we do away with the phase table in our implementation. In addition, due to the nature of the considered baseline cores (LP and HP), further classifying integer and floating-point instructions as ALU, multiply or divide does not offer any significant benefit. Therefore, we reduce the ITV from 9 to 5 elements corresponding to floating-point, integer, load, store and branch instructions. Our modified phase detection mechanism is shown in Figure 2. Khan *et al.* determined the phase classification parameters (n, m, and Δ) by experimentation. Since the baseline core configurations and the benchmarks that we consider are very different from those in [13], we have redone the experiments, details of which could be found in [28]. Based on those experiments, we have set the phase detection parameters to (i) interval length n = 100K instructions, (ii) threshold Δ = 12.5% and, (iii) m = 4 such that maximum throughput/Watt benefits were achieved when compared to the static baseline heterogeneous configuration.

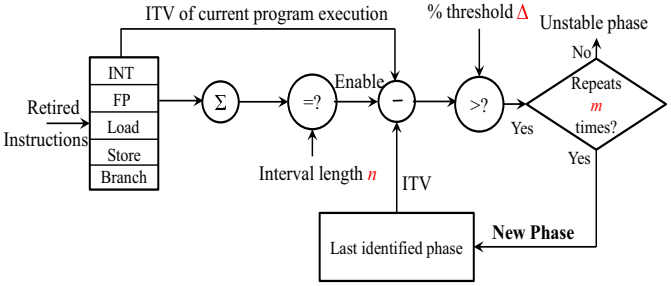

Fig. 2. Our phase detection mechanism.

TABLE III
VOLTAGE/FREQUENCY LEVELS OF THE CORES.

Core-type	Normal	Boost
LP	0.81 V / 1 GHz	0.9 V / 1.6 GHz
HP	1.1 V / 2 GHz	1.3 V / 3 GHz

B. Determining program affinity online by estimating the expected throughput/Watt

After establishing the phase detection mechanism, we need to determine online the affinity of the current program phase to the core-types and voltage/frequency levels in the AMP. The objective that our scheme tries to maximize is throughput/Watt

(Instructions per second (IPS)/Watt) which is the product of performance/Watt (IPC/Watt) and frequency. Since the frequency of each power state is known beforehand, the proposed scheme tries to predict the expected IPC/Watt of the current phase at different operating conditions on both the core-types. Hardware performance counters (HPCs) have been observed to reveal significant information about the characteristics of the thread currently being executed [29], [30]. We therefore, decided to develop a scheme to predict IPC/Watt of an executing application on the host core, as well as on other cores in the AMP at all the available voltage/frequency levels using HPCs. To do so, we need to first identify a set of counters that could be used for estimation and then choose a small subset that would have the highest correlation with the IPC/Watt.

The HPCs explored by us can be grouped as follows:

- **Instructions per Cycle (IPC)**: Power consumption of the processor is dependent on its activity and the IPC counter provides a good measure of program activity.
- **Fetch counters**: The IPC metric considers only the retired instructions, but in a processor, many instructions are executed speculatively and then flushed from the pipeline. To account for these, we considered *# Fetched instructions (F)* and, *Branch mispredictions (BMP)*.
- **Miss/Hit counters**: Cache hits/misses play a significant role in performance or power consumption of a core. In this regard, the following event counters: *L1 hit (L1h), L1 miss (L1m), L2 hit (L2h), L2 miss (L2m)* and, *TLB miss (TLBm)* are considered.
- **Retired instructions counters**: Performance or power consumption can vary significantly depending on the type of the retired instructions (integer (INT), floating-point (FP), Load (Ld), Store (St), Branch (Br)). Hence, we considered the corresponding retired instructions counters.
- **Stalls**: The activity of the processor will be low when it experiences data or resource conflicts frequently. We consider stalls due to reservation stations, re-order buffer (ROB), load/store queues (LSQ), register renaming and RAT (Register Alias Table). We refer to this counter as *Stalls (S)*.

In all, we examined 14 different HPCs. It is to be noted that none of the above counter values would change by changing the voltage/frequency levels of the corresponding core.

1) Performance/Power Modeling: Power estimation on the same core has been done before by using 3 to 4 counters [29], [30]. However, it is not straightforward to estimate the metrics on the other core by employing the counters of the host core. Our intent is to use the least number of counters to predict IPC/Watt at a reasonably high precision. The objective of this is not to save hardware, but, to minimize the number of counters that need to be monitored simultaneously.

We next discuss the total number of predictions required for the considered 2-core (LP and HP) AMP with two power states each (normal and boost). As power cannot be extracted at runtime, there is a need to estimate IPC/Watt of the current phase even on the same core at the current operating condition (besides at alternate condition). This results in total of 4 predictions (2 for each core and 2 for each operating mode) within the same core. Further, to make thread swapping decisions, we need to predict the expected IPC/Watt of a thread running on HP (LP) core, on LP (HP) core at both operating modes. This accounts for another 4 predictions thereby increasing the total number of predictions required to 8.

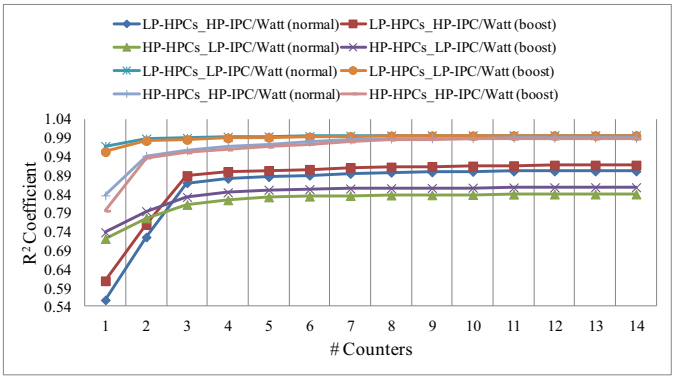

Fig. 3. Variation in the R^2 coefficient with increasing number of HPCs while predicting IPC/Watt at both operating modes on both the core-types. The HPCs of the first core-type in the legend name is used to estimate the IPC/Watt on the second core-type in the legend name at the operating mode mentioned within the parenthesis. For example, legend LP-HPCs_HP-IPC/Watt (normal) corresponds to IPC/Watt prediction on the HP core in the normal mode using the HPCs of the LP core.

Our objective is to identify the smallest set of counters from the list of 14 that could be used to estimate IPC/Watt. Once the right set of counters is chosen, we could employ multi-dimensional curve fitting and regression analysis to obtain expressions for IPC/Watt using the selected counters. To perform this analysis, we identified 12 representative benchmarks from the set of 38, such that they include: integer intensive (*intStress,bzip2,gzip*), floating-point intensive (*fpStress,equake,ammp*), load/store intensive (*gcc,whetstone,swim*) and, branch intensive (*mcf,twolf,art*) benchmarks. These 12 benchmarks were run on both the cores at the two operating modes for 5 billion instructions, after skipping the initial 5 billion. The value of the 14 performance counters along with the observed IPC/Watt were sampled periodically after the commit of every 100K instructions. All the obtained counter values were normalized with respect to the number of instructions (100K) in the interval. This normalization allows us to use the same IPC/Watt expressions obtained during the training phase for a different instruction length (n) while making runtime thread scheduling decisions.

To accomplish the task of making the right choice of HPCs, we devised an efficient heuristic that searches the counter space iteratively. During each iteration, our counter selection algorithm picks a new counter that best fits IPC/Watt along with the set of counters already chosen in previous iterations. We tried only linear models for curve-fitting and the best fit is qualified by the R^2 coefficient. During the initial few iterations, the value of the R^2 coefficient increases steeply as more counters are added, but it tends to saturate later. The best set of counters is around the region where the R^2 coefficient tends to saturate.

Figure 3 shows the value of the R^2 coefficient obtained dur-

TABLE IV
TRAINED EXPRESSIONS FOR IPC/WATT PREDICTION IN THE NORMAL MODE ON BOTH THE CORE-TYPES.

HPCs of/Prediction on	Expression
LP/HP	-1.2 × BMP - 0.1 × L1m + 0.04 × Br + 3.6 × 10^{-4} × S + 0.05
HP/LP	-0.28 × L1m - 0.04 × Ld + -0.5 × BMP + 0.1 × TLBm + 0.08
LP/LP	0.2 × IPC - 8 × 10^{-4} × S + 0.04
HP/HP	0.02 × IPC - 0.01 × L1m + 0.04

ing each iteration of the algorithm while estimating IPC/Watt for all the 8 cases (4 IPC/Watt predictions on the same core and another 4 predictions on the other core). It is evident that a reasonably high value of the R^2 coefficient is achieved for the same core predictions (the top 4 curves) and it saturates after 2 counters. Consequently, we used only two counters for IPC/Watt estimation on the same core. However, the value of the R^2 coefficient achieved while predicting the IPC/Watt on the other core (the bottom 4 curves) by using the HPCs of the host core is significantly lower than that on the same core (the top 4 curves). Further, the curves tend to saturate only after the third iteration indicating that 3 or more counters of the host core may be needed to adequately predict the IPC/Watt on the other core. Being an heuristic, our counter selection algorithm can only provide hints regarding the probable set of counters to explore. Therefore, we tried the set of 3 and 4 counters chosen by our algorithm around the saturating region of the curve and analyzed the increase in estimation accuracy. We observed that by using 4 counters, instead of 3, the average percentage error went down by 3.6%. We found this to be a good trade-off to make and went ahead with 4 counters for predicting the IPC/Watt on the other core. Based on these experiments, we obtained expressions for IPC/Watt prediction corresponding to each of the 8 cases. For the sake of brevity, we show only 4 out of the 8 expressions that correspond to IPC/Watt prediction in the normal mode in Table IV.

2) Evaluating the accuracy of IPC/Watt prediction: After deriving the expressions for estimating the IPC/Watt, we evaluated the accuracy of our prediction using all the 38 workloads, although only 12 were used during the training phase. The absolute average percentage error in IPC/Watt estimation for each of the 8 estimations is shown in Figure 4. Due to better quality of fit (higher value of R^2 coefficient), a much higher accuracy was achieved for estimating IPC/Watt on the same core when compared to the other core. The maximum average percentage error was less than 5% for IPC/Watt estimation on the same core. In contrast, the maximum average error was about 16.4% when predicting the IPC/Watt on LP core in normal mode using the HPCs of HP core.

From Figure 4 we make a key observation that the scheme faces higher imprecision while predicting the IPC/Watt on LP core using the HPCs of the HP core. The main suspect for this is the difference in L2 cache sizes of the two cores. The HP core has a 2 MB L2 cache while it is only 512 KB in the LP core. Hence, it is quite possible that the scheme overestimates the IPC/Watt on the LP core during memory intensive phases. Overall, our scheme achieves adequate accuracy in predicting

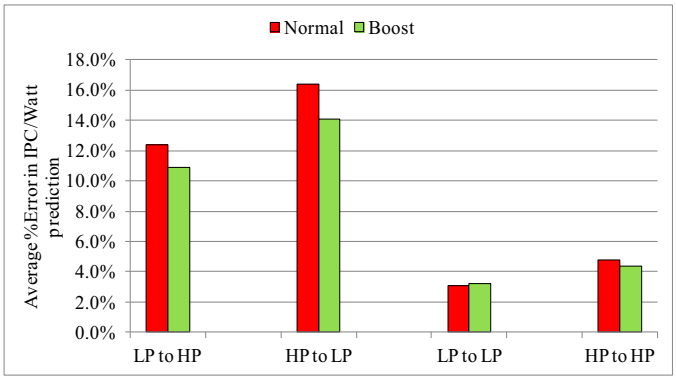

Fig. 4. Average percentage error in IPC/Watt estimation. The HPCs of the first core-type in the legend name is used to estimate the IPC/Watt on the second core-type in the legend name. For example, legend "LP to HP" corresponds to IPC/Watt prediction on the HP core using the HPCs of the LP core.

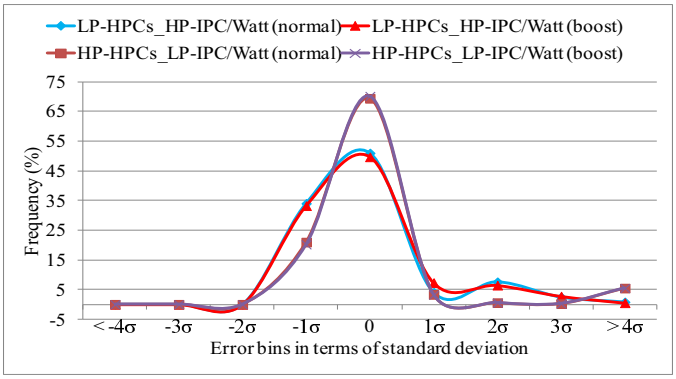

Fig. 5. Distribution of error in estimating IPC/Watt on the other core using HPCs of the host core. Horizontal axis indicates the number of standard deviations by which the observations are off from the mean.

the IPC/Watt both on the same core and on the other core at different operating conditions.

Although the overall average percentage error is low, the scheme could still end up making wrong thread scheduling decisions if the prediction error at the time of decision making is high. Hence, we also analyzed the distribution of the error in IPC/Watt prediction. For the sake of brevity, only the error distribution of the worst case, predicting the IPC/Watt on the other core using the HPCs of the host core, is shown in Figure 5. It is evident from the figure that most of the sample points are contained within +/- 1σ, reflecting the high accuracy of our prediction scheme. About 89% (94%) of the samples corresponding to IPC/Watt estimation on the HP (LP) core using HPCs of the LP (HP) core fall within this range. Therefore, we can expect our prediction scheme to make good thread scheduling decisions most of the time.

C. The complete thread scheduling framework

Having devised a phase detection mechanism and a scheme to predict the expected throughput/Watt at both the operating modes on the two considered core-types, we still need a way to seamlessly govern these two autonomous mechanisms. We assume a software layer called Microvisor to coordinate the predictions and thread scheduling decisions whenever a new phase is detected for either of the threads. A similar layer

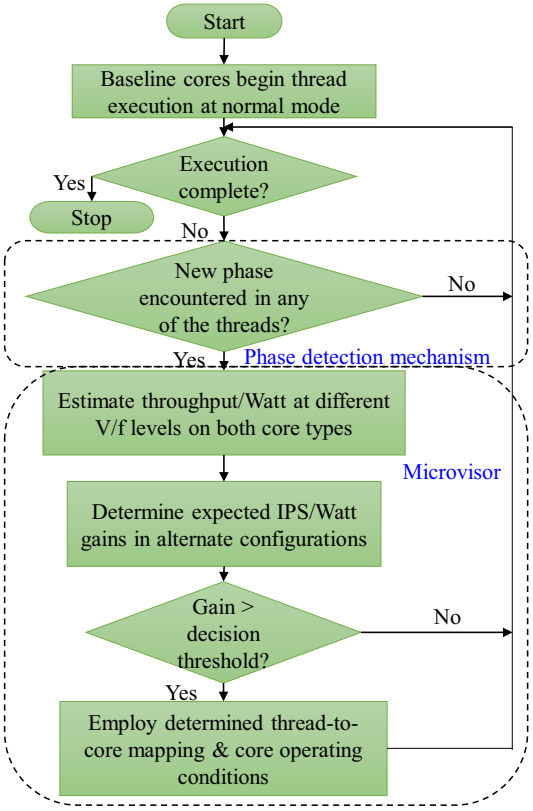

Fig. 6. Our complete thread scheduling framework.

called Millicode [31] was developed by IBM and has been used by Khan *et al.* in [27].

Figure 6 shows the flowchart of our thread scheduling framework. The two threads begin their execution in the normal mode with a random thread-to-core assignment. The phase detection mechanism in each of the cores monitors the ITV of the current execution phase. Whenever a stable phase change is detected for one of the threads, the Microvisor is invoked to predict the expected throughput/Watt of the current execution phases of both the threads at different operating modes on the two core-types. This prediction is done using the current values of the chosen HPCs of the corresponding host cores. Based on the above predictions, the Microvisor calculates the projected geometric throughput/Watt gain (weighted or harmonic metric could also be used) in moving to each of the possible new states (combinations of different thread-to-core assignment and voltage/frequency levels) over the current one. If the maximum geometric speedup is greater than 5% (called decision threshold to account for the overheads; detailed sensitivity study was conducted to determine this value), the corresponding new thread-to-core assignment and core operating conditions are opted for. Else, the current thread-to-core mapping and core operating conditions are maintained.

D. Overheads of the proposed scheme

Thread swapping between cores incurs an overhead due to context switch and cold cache misses. Rodrigues *et al.* [4] have estimated this overhead to be 400 cycles. To be conservative, we assume an overhead to refill about one-fourth of the L1 caches in the new core-type upon a thread swap. Considering the current memory access latencies and the processor-memory bus width [32], we estimated this to be about 13.8 μs. Hence, we assume an overhead of 14 μs for every thread swap. Changing the voltage and frequency of the cores at runtime incurs an even higher overhead. The processor needs to be halted while the PLL relocks to the new frequency. The PLL relock time in the latest Intel processors is 5 μs. In addition to the PLL relock time, while scaling up the voltage/frequency, the processor operates at the lower frequency until the voltage has risen to the new value [33]. This performance under-driven loss of the processor during the voltage transition time should also be considered under DVFS overheads. Based on the DVFS overhead expressions deduced by Park *et al.* [33], the performance under-driven loss comes to about 25 μs for our considered voltage/frequency levels. As indicated in Section V-A, we observed only about 39 reconfigurations (dynamic thread swapping and voltage/frequency changing) on an average for a program execution of 5 billion instructions which keeps the swap and DVFS overheads at bay.

TABLE V
OVERHEADS ASSOCIATED WITH THE PROPOSED SCHEME

Type	Overhead
Thread swap	14 μs
Voltage/frequency downscaling	5 μs
Voltage/frequency upscaling	30 μs
Microvisor invocation	500 cycles

The final source of overhead is associated with the invocation of the Microvisor whenever a new stable phase is detected for any one of the threads. On an average, the Microvisor was invoked 450 times per run; but the associated overhead is relatively small as it involves collecting the counter values from both the cores, evaluating the throughput/Watt expressions, calculating the projected gains and determining the final thread-to-core assignment and core operating conditions. This can be assumed to be at most a few hundred clock cycles and we observed this to have negligible impact on our results. In our experiments, we have assumed an overhead of 500 cycles for each Microvisor invocation. Table V presents the summary of the overheads considered in our experiments.

V. EVALUATION

We now evaluate the benefits of the proposed prediction-based thread scheduling scheme. A large number of multiprogrammed workloads were run on the considered dual-core AMP with two operating modes until both the threads completed 5 billion instructions (after the skipping the initial 5 billion). We compare the proposed thread scheduling schemes against the following baselines:

- **Static**: This is the baseline heterogeneous AMP with a *static* thread-to-core assignment. The fixed assignment is based on oracular knowledge of the best assignment over the entire run of the workloads and as such is not practical. The cores operate in normal mode and this baseline lacks the capability to change the voltage/frequency levels of the cores at runtime.
- **Swap-only**: In this baseline, the executing threads are swapped dynamically between the baseline cores, if deemed

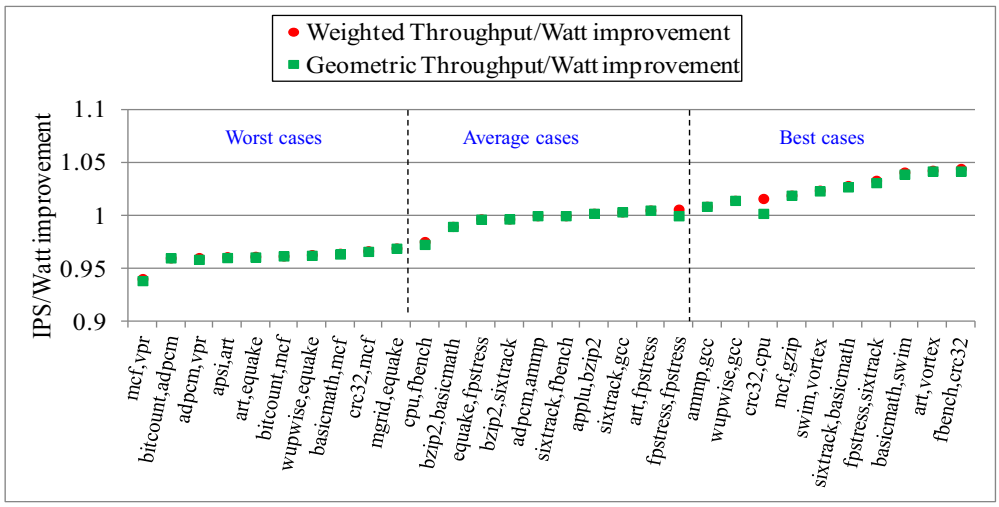

Fig. 7. Weighted/Geometric speedup in throughput/Watt due to the proposed scheme over *Greedy Oracle* scheme for different multiprogrammed workloads.

beneficial. Dynamic thread swapping is done using our prediction mechanism, i.e., by predicting the expected throughput/Watt of the current program phase on the alternate core-type. However, the baseline does not support DVFS. Even in this baseline, the initial thread-to-core assignment is made based on oracular knowledge.

• **DVFS-only**: This baseline has the capability to dynamically boost the voltage/frequency levels of the cores to adapt to the time-varying program behavior. But, dynamic thread swapping is not supported. As in the above two baselines, the threads begin their execution with an oracular best thread-to-core assignment. It should be noted that the trigger for both *Swap-only* and *DVFS-only* baselines is phase detection.

• **Greedy Oracle**: To illustrate the accuracy of our online prediction mechanism, we compare our scheme against a greedy oracle. This baseline supports both thread swapping and DVFS at runtime. The trigger is once again based on phase change detection. However, the thread scheduling decisions are based on oracular knowledge available at that instant in time, i.e., it makes 100% accurate throughput/Watt predictions. Further, we completely discount all the overheads for this baseline to explore the theoretical maximum gains that could be achieved.

A. Throughput/Watt analysis

To illustrate the benefits of our approach, we compare the throughput/Watt achieved using our scheme over the different baselines. We employ weighted and geometric speedup metrics for comparison. The considered geometric metric accounts for the fairness of the system. We first define the following terms:

$S_0 = (IPS/Watt_{thread0})_{proposed}/(IPS/Watt_{thread0})_{baseline}$
$S_1 = (IPS/Watt_{thread1})_{proposed}/(IPS/Watt_{thread1})_{baseline}$

The considered speedups are:

1) Weighted: $Speedup_{weighted} = (S_0 + S_1)/2$
2) Geometric: $Speedup_{geometric} = \sqrt[2]{S_0 \times S_1}$

From the pool of all 38 benchmarks, 120 random combinations of two benchmarks were chosen and run on the dual-core using the proposed scheme as well as each of the baseline schemes. Although all the baseline configurations start with a best initial thread-to-core assignment, a random initial assignment is considered for the proposed scheme. The hope is that our scheme will detect the best assignment shortly after the programs begin to run.

As even the baseline configurations (*swap-only* and *DVFS-only*) employ our prediction mechanism, we need to evaluate the prediction accuracy at the time of decision making. Hence, we first compare the proposed scheme against the *Greedy Oracle*. Figure 7 shows the weighted/geometric throughput/Watt speedup of our scheme over the oracle. For the sake of clarity, only 30 combinations (out of the 120) are shown in the figure. The shown 30 combinations were carefully chosen to include the 10 worse results (out of the 120), the 10 best results and 10 that showed average throughput/Watt benefits. As expected, for most cases the speedup is less than 1 as we are comparing against an oracle. It is promising to observe that even under such scenarios, we fall short of the oracle only by about 6% in the worst case. The average percentage error in throughput/Watt prediction at the time of decision making, corresponding to that combination ({*mcf,vpr*}) is a reasonable 17%. The average weighted/geometric throughput/Watt improvement achieved by our scheme relative to the oracular scheme is 0.99 considering all the 120 combinations. This strongly indicates that our scheme closely follows the oracle and makes the right thread scheduling decisions most of the time. The average speedup of 0.99 even without considering the overheads for the oracular scheme infers that the number of thread swapping and voltage/frequency boosting done using our scheme is minimal. Considering all the 120 combinations, we observed an average percentage error of only about 11.3% in throughput/Watt prediction while making thread scheduling decisions. Further, there were only about 39 thread swaps and voltage/frequency changes on an average for an execution of 5 billion instructions. It is interesting to note that the proposed scheme does better than the oracular scheme in a few rare cases. The reason for this is that sometimes by taking a wrong decision (as is done by the proposed scheme), the opportunities that come up later, as compared to the case where always the

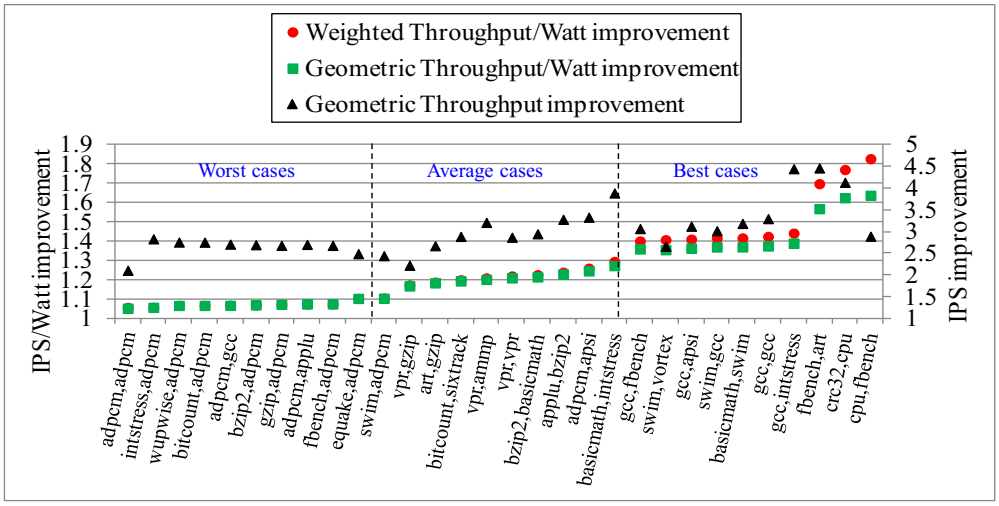

Fig. 8. Weighted/Geometric improvement in throughput/Watt and Geometric speedup in throughput due to the proposed scheme over the static baseline heterogeneous configuration for different multiprogrammed workloads.

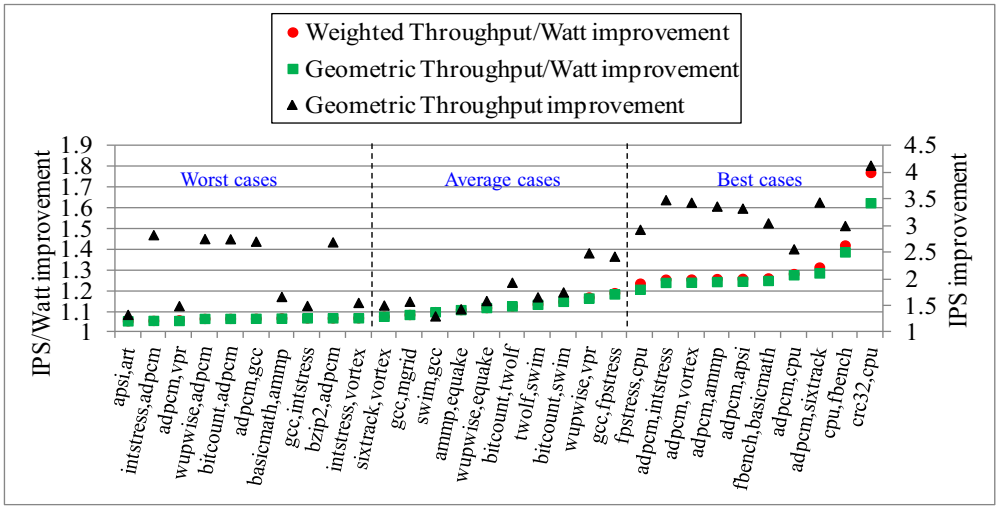

Fig. 9. Weighted/Geometric improvement in throughput/Watt and Geometric speedup in throughput due to the proposed scheme over the *swap-only* baseline for different multiprogrammed workloads.

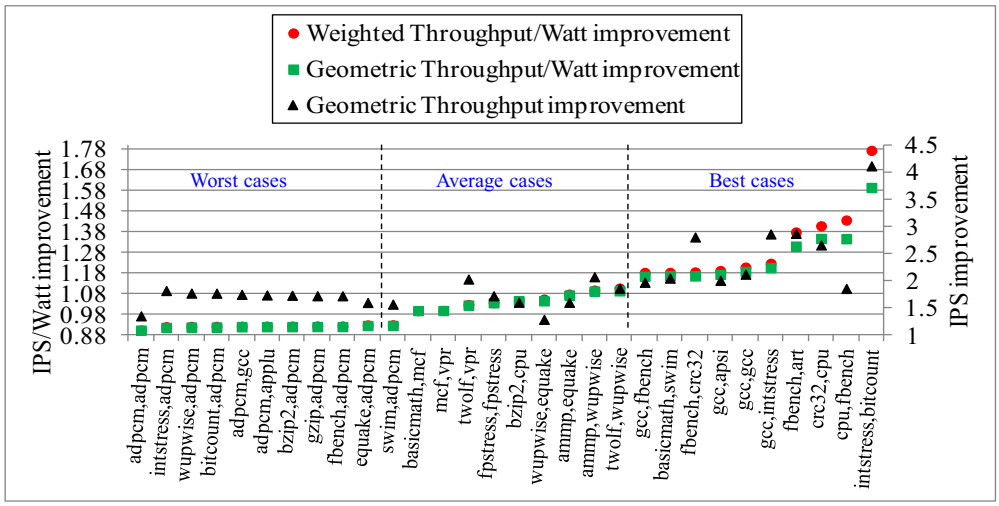

Fig. 10. Weighted/Geometric improvement in throughput/Watt and Geometric speedup in throughput due to the proposed scheme over the *DVFS-only* baseline for different multiprogrammed workloads.

right (greedy) decision is made, are different. These additional opportunities at times may provide better benefits.

We next compare the proposed scheme against the non-oracular baselines. In addition to the performance/power metric (throughput/Watt), we also show the stand-alone performance impact by calculating the geometric throughput improvement achieved using our scheme.

a) vs. Static: The *static* baseline lacks the capability to adapt to the time-varying behavior of the workload and hence, it never takes advantage of program phases. Though a thread may have affinity for a certain core or an operating mode over the entire run, there may be periods where this thread would be more affine to another core or a power state in the AMP. By taking advantage of program phases and adapting to the thread needs, the proposed scheme achieves significant throughput and throughput/Watt benefits over this baseline (see Figure 8). Of the 120 combinations, we did not find any case where this baseline performed better than the proposed scheme. Even for the worst case, our scheme was better than the *static* baseline by 5.5% (5.2%) when considering weighted (geometric) throughput/Watt improvement. On an average, considering all the 120 combinations, our scheme achieved a 27.2% (25%) weighted (geometric) improvement in throughput/Watt over this baseline. Furthermore, by opportunistically opting for the boost mode and efficiently making use of the HP core for high-compute intensive/high-ILP program phases, our proposed scheme resulted in much higher throughput improvement of about 190%, on an average, over the *static* baseline. These results stress the need for a dynamic scheme that can adapt the available core resources and operating modes to the program phase behavior.

b) vs. Swap-only: This is a dynamic baseline that can swap threads between cores at runtime. As mentioned before, the trigger and thread scheduling mechanisms are similar to that of the proposed scheme. As could be seen from Figure 9, a substantial increase in throughput/Watt is achieved using our scheme even over the *swap-only* baseline. Again, we did not encounter any combination where the *swap-only* scheme performed better than ours. We achieved a throughput/Watt improvement of about 5.3% even for the worst case. Using the proposed scheme, there was on average 13.7% (13%) weighted (geometric) improvement in IPS/Watt and about 91% geometric improvement in IPS over the *swap-only* baseline.

We analyzed the benchmark combinations at the right end of Figure 9 for which we achieve maximum IPS/Watt improvement over the *swap-only* baseline. It is interesting to note that most of them are either compute-memory intensive benchmark combinations (e.g., {*fbench*,*basicmath*}) or both are compute intensive benchmark combinations (e.g., {*adpcm,cpu*}, {*adpcm,sixtrack*}). In the case of {*fbench*,*basicmath*}, *fbench* is memory intensive with about 58% load/store instructions while *basicmath* is compute intensive. For such compute-memory intensive benchmark combinations, besides determining the best thread-to-core assignments, our scheme makes use of DVFS to good extent. During high-IPC/high-ILP phases of compute intensive benchmark, our scheme pushes the HP core to boost mode resulting in much faster execution and hence, better IPS/Watt. This results in a higher IPS speedup for these combinations over *swap-only* baseline (speedup of about 3 for {*fbench,basicmath*}). For cases when both the threads go through high compute intensive phases at the same time (e.g., {*adpcm,cpu*}), our scheme pushes the HP core to boost mode, clearing the conflict for better resources (HP core) quickly. During this time, the performance of the thread executing on non-affine core is improved by opting for the boost mode within the LP core. Once the conflict clears up, the latter thread is migrated to HP core. These cases clearly substantiate the need for dynamically changing the voltage/frequency of the cores besides swapping threads.

c) vs. DVFS-only: This is another baseline with some capability to adjust to the program behavior. Here, the voltage and frequency of the cores are chosen so as to maximize throughput/Watt. As is evident from Figure 10, the proposed approach achieves reasonable benefits over this baseline as well. In contrast to the previous two baselines, there were few benchmark combinations (e.g., {*adpcm,adpcm*}, {*bitcount,adpcm*}) out of the 120, for which our scheme performed worse than *DVFS-only* scheme. In the worst case, we observed a IPS/Watt degradation of about 9.5% using our scheme. We analyzed these cases and observed a probable reason for such a behavior. For few rare cases, our scheme ends up making wrong thread scheduling decisions due to errors in throughput/Watt prediction. As a result of this, our scheme performed few non-beneficial thread swaps and opted for boosting the voltage and frequency of the cores at a much later stage of the program execution. Since it is an opportunistic scheme that looks for a thread scheduling opportunity only upon a phase change, a wrong thread scheduling decision made, is retained for the entire phase, magnifying its impact. However, these worst case scenarios were infrequent and there were only 10 out of 120 combinations that resulted in degradation of more than 3%. We also analyzed the cases for which our scheme performs much better than the *DVFS-only* scheme. We observed that most of such cases were for symmetric workload combinations (both the threads having affinity for the same core-type, e.g., {*gcc,gcc*}, {*intStress,bitcount*} - both are integer intensive). By swapping threads, our scheme efficiently shares the affine resource (preferred core-type) while one of the threads is forced to execute on the non-affine core throughout its execution in *DVFS-only* scheme. Hence, there is a definite need for a scheme to support thread swapping besides DVFS.

Furthermore, we analyzed the best 10 cases for which our scheme achieves maximum IPS/Watt speedup over *swap-only* (see Figure 9) and *DVFS-only* (see Figure 10) baselines. We noticed that there were only 2 benchmarks combinations ({*crc32,cpu*} and {*crc32,cpu*}) that were in common between the two. This is very encouraging for our proposed scheme as it shows that the benefits of dynamic thread swapping and DVFS are mostly non-overlapping. As a result, different kinds of benchmark combinations could benefit from either of them, indicating the potential benefits of schemes (like the

one proposed in this paper) that seamlessly combine the two approaches.

VI. Conclusions

We have presented an opportunistic prediction-based thread scheduling scheme to maximize throughput/Watt in AMPs. The key idea of the proposed approach is the online prediction of the expected throughput/Watt of the current program phase at different operating conditions on all the available core-types. We leverage the use of performance counters which are available in almost all processors for such a prediction. To illustrate our approach, we considered a dual-core AMP comprising of a high performance HP core and a power efficient LP core with two operating modes. We presented a counter selection heuristic to determine the smallest subset of HPCs to adequately predict throughput/Watt on various core configurations. Approximate expressions based on the values of the chosen HPCs were formulated to assist in determining the best thread-to-core mapping and core operating conditions to suit the current program phase.

We compared our proposed scheme to a static baseline with best thread to core assignment, a baseline with *swap-only* capability, a baseline with *DVFS-only* capability, and an oracular scheme that makes 100% accurate predictions. Our results indicate that the proposed scheme can achieve significant weighted throughput/Watt benefits of about 27.2%, 13.7% and 8% on an average, over the static baseline, the baseline with *swap-only* capability and the baseline with *DVFS-only* capability, respectively. Further, the proposed scheme falls short of the oracular scheme by only about 6% in the worst case.

VII. Acknowledgement

This research was supported in part by grants 0903191 and 1201834 from the National Science Foundation.

References

[1] J. Held *et al.*, "White Paper From a Few Cores to Many: A Tera-scale Computing Research Review," 2006.

[2] M. Pericas *et al.*, "A Flexible Heterogeneous Multi-Core Architecture," in *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques, 2007*.

[3] R. Kumar *et al.*, "Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction," in *Proceedings of 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003*.

[4] R. Rodrigues *et al.*, "Performance Per Watt Benefits of Dynamic Core Morphing in Asymmetric Multicores," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2011*.

[5] A. Annamalai *et al.*, "Dynamic Thread Scheduling in Asymmetric Multicores to Maximize Performance-per-Watt," in *26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, 2012*.

[6] P. Greenhalgh, "Big.little processing with arm cortex-a15 and cortex-a7," 2011.

[7] G. Keramidas *et al.*, "Interval-based models for run-time DVFS orchestration in superscalar processors," in *7th ACM International Conference on Computing Frontiers, 2010*.

[8] "Intel Corporation. Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors, White Paper," 2008.

[9] D. Foley *et al.*, "AMD's "LLANO" FUSION APU, Hot Chips 2011, Paper: HC23.19.930."

[10] M. Becchi *et al.*, "Dynamic thread assignment on heterogeneous multiprocessor architectures," in *Proceedings of the 3rd conference on Computing frontiers, 2006*.

[11] D. Koufaty *et al.*, "Bias scheduling in heterogeneous multi-core architectures," in *Proceedings of the 5th European conference on Computer systems, 2010*.

[12] J. C. Saez *et al.*, "A comprehensive scheduler for asymmetric multicore systems," in *Proceedings of the 5th European conference on Computer systems, 2010*.

[13] O. Khan *et al.*, "A self-adaptive scheduler for asymmetric multi-cores," in *Proceedings of the 20th symposium on Great lakes symposium on VLSI, 2010*.

[14] D. Shelepov *et al.*, "HASS: a scheduler for heterogeneous multicore systems," *SIGOPS Oper. Syst. Rev.*, vol. 43.

[15] J. Chen *et al.*, "Efficient program scheduling for heterogeneous multi-core processors," in *Proceedings of the 46th Annual Design Automation Conference, 2009*.

[16] S. Srinivasan *et al.*, "Efficient interaction between OS and architecture in heterogeneous platforms," *SIGOPS Oper. Syst. Rev.*

[17] V. Craeynest *et al.*, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," *SIGARCH Comput. Archit. News*.

[18] R. Rodrigues *et al.*, "Scalable thread scheduling in asymmetric multicores for power efficiency," in *IEEE 24th International Symposium on Computer Architecture and High Performance Computing, 2012*.

[19] A. Fog, "The microarchitecture of Intel, AMD and VIA CPU," Copenhagen University College of Engineering, Tech. Rep.

[20] J. Renau, "SESC: SuperESCalar Simulator," 2005.

[21] D. Brooks *et al.*, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proceedings of the 27th International Symposium on Computer Architecture, 2000*.

[22] P. Shivakumar *et al.*, "CACTI 3.0: An Integrated Cache Timing, Power, and Area Model," Tech. Rep., 2001.

[23] M. Guthaus *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE International Workshop on Workload Characterization, 2001*.

[24] SPEC2000, "The Standard Performance Evaluation Corporation (Spec CPI2000 suite)."

[25] C. Lee *et al.*, "MediaBench: a tool for evaluating and synthesizing multimedia and communicatons systems," in *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture, 1997*.

[26] A. S. Dhodapkar *et al.*, "Comparing Program Phase Detection Techniques," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, 2003*.

[27] O. Khan *et al.*, "Microvisor: A Runtime Architecture for Thermal Management in Chip Multiprocessors," *T. HiPEAC*, vol. 4, 2011.

[28] R. Rodrigues *et al.*, "Improving performance per watt of asymmetric multi-core processors via online program phase classification and adaptive core morphing," *ACM Trans. Des. Autom. Electron. Syst.*

[29] G. Contreras *et al.*, "Power prediction for Intel XScale reg; processors using performance monitoring unit events," in *Proceedings of the International Symposium on Low Power Electronics and Design, 2005*, aug. 2005, pp. 221 – 226.

[30] K. Singh *et al.*, "Real time power estimation and thread scheduling via performance counters," *SIGARCH Comput. Archit. News*.

[31] L. C. Heller *et al.*, "Millicode in an IBM zSeries processor," *IBM Journal of Research and Development*, 2004.

[32] "The Nehalem Preview: Intel Does It Again. http://www.anandtech.com/show/2542/5."

[33] J. Park *et al.*, "Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design, 2010*.