Exploring Heterogeneity within a Core for Improved Power Efficiency

Sudarshan Srinivasan, Nithesh Kurella, Israel Koren, *Fellow, IEEE,* and Sandip Kundu, *Fellow, IEEE*

Abstract—Asymmetric multi-core processors (AMPs) comprise cores with different sizes of micro-architectural resources yielding very different performance and energy characteristics. Since the computational demands of workloads vary from one task to the other, AMPs can often provide a higher power efficiency than symmetric multi-cores. Furthermore, as the computational demands of a task change during its course of execution, reassigning the task from one core to another, where it can run more efficiently, can further improve the overall power efficiency. However, too frequent re-assignments of tasks to cores may result in high overhead. To greatly reduce this overhead, we propose a morphable core architecture that can dynamically adapt its resource sizes, operating frequency and voltage to assume one of four possible core configurations. Such a morphable architecture allows more frequent task to core configuration re-assignments for a better match between the current needs of the task and the available resources. To make the online morphing decisions we have developed a runtime analysis scheme that uses hardware performance counters. Our results indicate that the proposed morphable architecture controlled by the runtime management scheme, can improve the throughput/Watt of applications by 31% over executing on a static out-of-order core while the previously proposed big/little morphable architecture achieves only a 17% improvement.

Index Terms—Asymmetric multi-core processors; Hardware Performance Counters; Morphable core; Design space exploration; Online morphing.

1 Introduction

As the focus of the microprocessor industry has shifted from high performance computing to energy efficient computing, Asymmetric Multi-core Processors (AMPs) have emerged as a viable stride forward. AMPs consist of cores that have the same ISA but different power and performance characteristics to better match various application behaviors [1], [2]. Commercial AMP architectures, such as ARM's big.LITTLE, consist of a high performance Out-of-Order (OOO) core and a smaller energy efficient In-order (InO) core [3]. AMPs offer opportunities to achieve higher energy efficiency by dynamically migrating an application from one core type to another based on its current resource needs. The key challenges are: *(i)* how to dynamically determine which is the best core for the application to run on and, *(ii)* How often to allow such migrations, given that they may entail considerable overhead.

Traditional thread migration in AMPs is facilitated by *sampling* the application on each core for both performance and performance/Watt and then assigning it to the core that best suits its current needs [1]. The most important drawback of this approach is that it involves a substantial overhead that increases with the number of core types. Consequently, task migrations can only be performed at *coarse grain* instruction granularity [6], thus missing opportunities at fine grain granularity. Recent research has shown that considerable energy savings opportunities exist at fine instruction granularities [4]. To take advantage of such opportunities, Lukefahr *et al.*, [4] have proposed a composite core architecture where an OOO core dynamically morphs into an InO core during runtime. The use of a dynamically morphable core raises the question whether allowing such a core to morph into more than two distinct modes can further improve its power efficiency.

Navada *et al.* explored AMP cores and showed that applications can exhibit diverse program phase behavior where each program phase can exhibit one (or more) of common processor performance bottlenecks arising from cache misses, limited execution resources or execution width, large degree of instruction dependencies, or inherently low instruction level parallelism [5]. Consequently, they proposed using a diverse set of cores designed to address these bottlenecks, and showed that the use of these cores can improve the performance and provide higher performance/Watt.

Lukefahr *et al.* [4] and Khuaib *et al.* [6] proposed dynamic morphing between a big OOO core and a small InO core during run time. Such designs eliminate the overhead associated with transferring the state of a workload from one core to another upon a switch. This allows morphing to take place at fine grain instruction granularities (∼1000 instructions) which reportedly results in significant energy savings with performance loss up to 5% [4]. We observe that resource requirements of applications can be quite diverse and consequently, resource bottlenecks or excesses can vary considerably. Thus, morphing between just two core modes may not fully exploit power and performance improvement opportunities.

To test the potential benefits of having three or more

• *Authors are with Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA*

distinct morphable core configurations, we analyzed the IPC/Watt of three OOO cores differing in execution widths and core resources (which are scaled appropriately for the chosen width), for workloads from the SPEC 2006 suite [7]. We call these cores the 4-way, 2-way and 1-way cores where 4, 2, and 1 indicate the execution width. Figure 1 shows that there are workloads that achieve the highest IPC/Watt when run on the 4-way core while for other workloads a 2-way or even a 1-way core can provide the highest performance/Watt. The latter workloads do not need the same amount of resources as those that prefer the 4-way core. Hence, for such workloads energy savings can be achieved by running them on the reduced fetch width core with reduced resource sizes, resulting in better performance/Watt.

Figure 2 shows the performance/Watt for the *sjeng* benchmark from the SPEC 2006 suite [7] at fine instruction granularity, for the three OOO cores mentioned above. Although when executing on a single core type, *sjeng* prefers the 1-way core (see Figure 1), there are time periods when a 4-way or a 2-way core will yield a higher performance/Watt. Such a temporal variation in demand for resources motivates studying a morphable architecture that can morph between three (or more) core modes as it may achieve a higher performance/power.

In this paper we perform a core design space exploration to select a set of core architectures that are fundamentally different from the big/little architecture. The architectures of the cores can differ in fetch width, issue width, buffer sizes (e.g., ReOrder Buffer (ROB), Load Store Queue (LSQ) and Instruction Queue (IQ)), clock frequency and operating voltage. We then use the selected core architectures to define the distinct core modes of the proposed morphable architecture. Our self-morphable core will dynamically morph into any one of these execution modes and will, this way, mimic a high diversity asymmetric multicore processor.

A morphable core architecture that can switch between different core modes needs an effective online mechanism to determine the most efficient core mode for the current phase of the application. Furthermore, since many performance and power improvement opportunities exist at a fine instruction granularity, the morphing decision and the morphing process must be done fast. To this end, we propose an online decision mechanism to identify the most power efficient core mode for the current execution based on hardware performance monitoring counters (PMCs).

The contributions of this paper are:

• We conduct design space exploration to identify the best core modes for the morphable architecture. The number of modes is determined based on the law of diminishing marginal utility. The design space exploration has resulted in four distinct core modes appropriate for fine grain switching.

• We study the throughput/Watt of the proposed morphable architecture against previous publications and show 17% improvement in throughput/Watt and 9% improvement in performance over morphable architectures with only Big/Little cores. We also improve the throughput/watt of applications by 31% over executing on a static out-of-order core.

• We propose a simple on-line management solution based on online estimation of throughput/Watt of the application on all core modes. A counter selection algorithm has been used to determine the appropriate number of counters which will allow us to estimate the power and performance in each of the core modes.

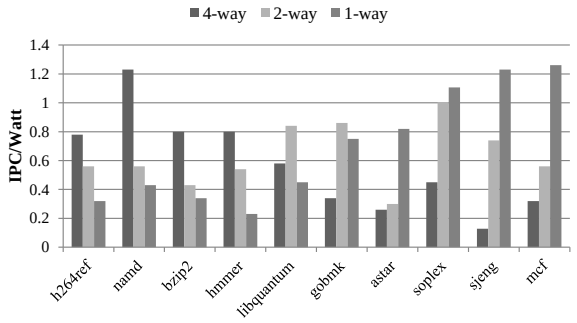

Fig. 1. IPC/Watt for SPEC benchmarks [7] running on OOO cores differing in fetch/execution/retire widths and core resources.

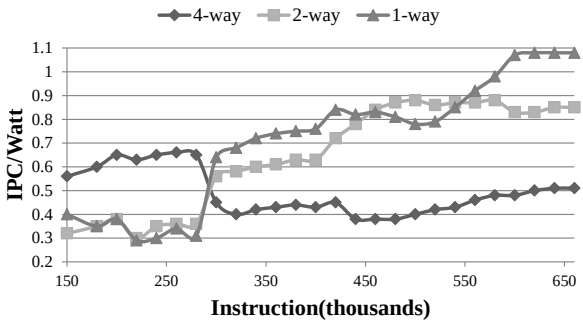

Fig. 2. IPC/Watt over a period of execution for the benchmark *sjeng* [7] sampled every few thousand of committed instructions.

2 Related Work

In this section we summarize selected previous research related to asymmetric multicores and morphable cores. This review is not meant to be a comprehensive survey of related work. We only include those that are relevant to the principles of our proposed scheme.

2.1 Asymmetric Multicore Processors (AMP)

Previous research on AMPs has explored their benefits to both single and multi-threaded programs in terms of performance and power. Annavaram *et al.* advocate using a big core for serial sections of a program, while parallel sections are run on moderately resourced cores [8]. This improves the overall performance of a multi-threaded program within a given power budget. Suleman *et al.* presented an architecture consisting of a big

core and several small cores [9]. The big core was used as an accelerator to eliminate serial bottlenecks.

Kumar *et al.* proposed using a mix of cores with different power and performance characteristics, so that every application phase is assigned to a core that achieves the best power-efficiency [1]. For example, if a certain program phase exhibits low IPC, it can be assigned to a small core for greater power efficiency. In such a case, boosting the frequency of the smaller core (within the power dissipation constraints) may achieve *both* higher performance and power efficiency. Core design space exploration studies for performance/power have been described in [10], [11]. Recently, Navada *et al.* considered accelerating single threaded workloads by performing complete design space exploration and identifying a set of heterogeneous cores that would maximize performance [5]. Their core design space exploration was done at coarse grain instruction granularity to determine the best set of core types for an AMP. In this work we try to unearth new core types for which switching could be performed at fine grain.

The above described AMP architectures make workload to core assignments online at the granularity of an application phase change. If these AMP designs are made to switch at fine grain instruction granularity, the penalty of thread switching would be high, thus negating the expected power benefits.

2.2 Morphable Asymmetric Cores

There are several publications advocating morphing of a core at runtime to adapt to changing workload needs and improve performance and/or power efficiency.

Some researchers have considered fusing several small cores into a large OOO core on demand [12], [13]. This approach introduces additional latencies in the pipeline due to the layout and logic constraints.

Recently, Khubaib *et al.* proposed a morphable architecture to support fine grain thread switching to improve performance/Watt [6]. A traditional OOO core is morphed into an highly threaded in-order SMT core when Thread-level Parallelism (TLP) is available. Their main observation was that a highly threaded in-order core can achieve a higher energy efficiency than a more powerful OOO core. Lukefahr *et al.* proposed a morphed architecture where an OOO core is morphed into an InO by provisioning two execution back-ends in the same core where one back-end engine is used in the OOO mode and the other in the InO mode [4]. With their dual back-end architecture they can switch between OOO and InO modes at fine grain instruction slices.

These morphable architectures focus only on morphing between two extreme architectures while we explore, in this paper, morphing into a larger number of core configurations (or modes). Such a morphable architecture is more likely to match the demands of various workloads by addressing a more *diverse set of bottlenecks*.

A less aggressive form of core morphing has been discussed in [14], [15]. These configurable architectures dynamically adjust the cache and storage buffers such as ROB, LSQ and IQ to the application demands. The resource adaptations are done at coarse grain granularities by adjusting the size of each buffer independently. The proposed configurable architectures do not consider varying the execution width or changing the frequency and voltage, as we do in this work.

2.3 Thread to Core Assignment in AMPs

Several prior papers have employed regression-based analysis for thread to core assignment in AMPs [16], [17]. Such schemes characterize the behavior of the workloads offline and this information is then distilled into a set of runtime signatures that are used for online assignment of threads to cores. In contrast, online learning schemes offer a more practical approach to thread to core mapping. These schemes learn the characteristics of the workloads online and make informed assignment decisions.

These online learning schemes primarily employ phase classification and sampling techniques [1], [18]. Whenever a stable phase change is detected, the new phase is sampled on all the core-types in the AMP. The sampling poses a significant overhead and hence such schemes can only be employed at coarse grain instruction granularities.

Estimation-based thread assignment schemes offer improvements over the offline profiling and online learning schemes. In such schemes, the performance and/or power behavior of a particular workload on another core in the AMP is estimated using event statistics such as cache misses and pipeline stalls gathered on the host core [4], [19], [20]. Thus, these estimation-based schemes overcome the shortcomings of the above two approaches.

The estimation-based schemes proposed so far considered only two core modes. In contrast, we allow a larger number of architectural modes (each operating at its own frequency/voltage) and we develop an online mechanism which can estimate the power and performance for all the available core modes to determine (at a fine instruction granularity) the one that can provide the highest power efficiency.

3 Proposed Architecture

AMPs may contain multiple core types with each core type specialized for specific workload characteristics. As the multicore is constrained by the Thermal Dissipation Power (TDP) limit of the package, the cores cannot feature the largest possible size for all micro-architectural structures and yet operate at the highest possible frequency.

Consequently, there are always trade-offs in core design. For example, to support a higher degree of instruction level parallelism (ILP), the pipeline width should be increased. Such an increase would, in turn, limit the allowed core frequency due to the TDP constraint. Thus, a core specialized for high ILP may not meet the needs

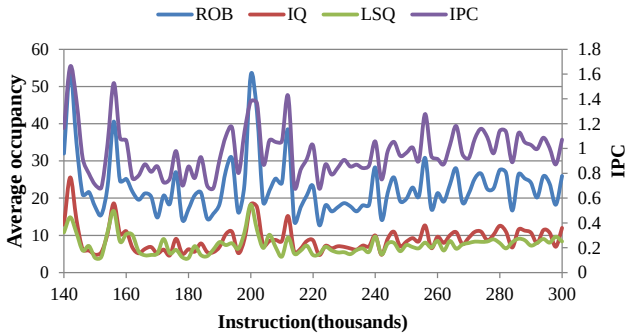

Fig. 3. Temporal changes in IPC and resource occupancy for the benchmark *sjeng* as a function of instructions committed.

TABLE 1
Core Design Parameters

Core Parameter	Range of Values
Fetch, Issue Width	1,2,3,4,5
ROB size	8,16,32,64,96,128,192,256,384
Issue Queue size	12,24,36,48,64
LSQ Size	8,16,32,64,96,128,192
Clock Period	0.4ns-1ns (steps of 0.1ns)

of a workload with extensive sequential instruction dependency whose performance can only be improved by increasing the frequency. Therefore, designing the right mix of cores for an AMP that caters to the demands of diverse workloads requires careful balancing.

A carefully selected mix of cores that can address different resource demands can also benefit the execution of a single benchmark. For example, Figure 3 shows the IPC variations (between 0.6 to 1.6) observed in the course of executing the SPEC benchmark *sjeng*. The figure shows that as the IPC varies, the usage of several core resources also varies. We further observe that variations in ROB, LSQ and IQ occupancy happen at a *small instruction granularity*. Therefore, our proposed *self-morphable core* should be capable of dynamically reconfiguring to adapt to the current demands of the executing workload and should allow such reconfiguration to be done at a fine-grain instruction granularity. To this end, we do not vary cache sizes upon core reconfiguration to avoid costly migrations of cache content. The selected fixed cache sizes were determined experimentally on a set of benchmarks to be: 64KB for the I-cache and the D-cache and 2MB for the L2 cache.

3.1 Design Space Exploration

We wish to identify the distinct core modes that should be supported by our morphable core. Table 1 shows the range of core parameters for our core selection. Clock frequency is varied within the common range of superscalars' speed. The search would determine the core modes that can provide the best performance/Watt for fine grain instruction slices. If we allow the core parameters to assume all their possible values shown in Table 1, the resulting design space exploration would require experimenting with 11,025 combinations. However, core sizing for improved performance/power in [17] has shown that increasing the size of one resource without a commensurate increase in other resources yields limited benefits. Thus, certain parameter combinations such as (ROB=8, IQ=64, LSQ=192, Width=4) are not acceptable design candidates as a small 64-entry ROB cannot support large IQ and LSQ and would not result in any benefits. By performing design space pruning, we have reduced the number of core design combinations that need to be analyzed to 300.

The remaining 300 combinations were analyzed exhaustively with the objective of achieving the highest possible $IPS^2/Watt$ by allowing switching between core modes every 2000 instructions. The decision to switch modes is based on the metric $IPS^2/Watt$ that assigns higher significance to performance than to power. Unlike the IPS/Watt metric (used, for example, in [21], [22]), using $IPS^2/Watt$ is more likely to avoid switching to a core mode that reduces the power greatly but also lowers considerably the performance. The reason for choosing 2K as our fine grain instruction interval will be explained in the next section. After each 2K retired instructions interval we computed the $IPS^2/Watt$ for every core configuration out of the 300 candidates. A minor increase in the $IPS^2/Watt$ would not justify a core mode switch but, more importantly, would not justify adding a new core mode. Therefore, we had to choose a threshold for the minimum improvement that will justify an additional core mode. Each mode switch involves an overhead (explained in Section 5.3) and thus we need to determine the preferred number of modes as a large number of modes may result in high morphing overhead while too few modes may not benefit all benchmarks and thus achieve a lower $IPS^2/Watt$. Our experiments revealed that selecting a $IPS^2/Watt$ threshold of less than 20% yields 10 core types but the additional $IPS^2/Watt$ improvement achieved by most of these core types is limited. Increasing the threshold to 20% reduces the number of core types to four with a higher $IPS^2/Watt$ improvement for most benchmarks. We have also observed that the $IPS^2/Watt$ improvement and the details of the core types are not very sensitive to small variations (from 20%) in the threshold. A further increase in the threshold to 30% (and higher) resulted in fewer core combinations but the majority of benchmarks did not sufficiently benefit from morphing. We have, therefore, decided to use a threshold of 20% and have as a result, four core modes.

3.1.1 Power Unconstrained Core Selection

The architectural parameters of the selected four core modes that will best accommodate the diverse application phase behavior (of the SPEC benchmarks) are shown in Table 2. This set includes an Average Core (AC) which targets most of the application phases and a Larger Window (LW) core that has a bigger window

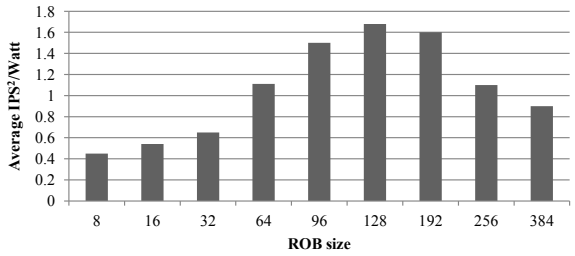

Fig. 4. IPS^2/Watt as a function of ROB size for the AC core mode (power unconstrained).

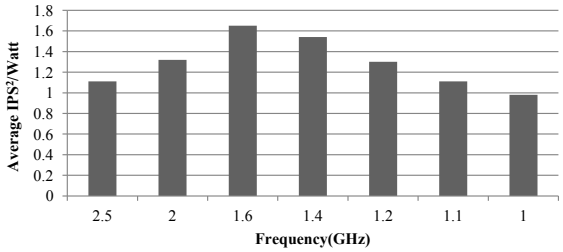

Fig. 5. IPS^2/Watt as a function of frequency for the AC core mode (power unconstrained).

size and targets application phases which have a window bottleneck. In addition, the set includes a Narrow Core (NC) which targets application phases with low ILP and accelerates sequential execution using a higher frequency. The fourth mode is a Small core (SM) with a lower frequency that caters to low performance phases that exist at fine grain granularity.

To illustrate our core mode selection process, we show the impact of changes in the ROB size of the AC mode in Figure 4 for a subset of benchmarks that spent significant amount of time in the AC mode. It can be observed that ROB=128 (the chosen size for the AC mode), offers the best IPS^2/Watt. Figure 5 illustrates our process for determining the frequency. We observe that at a frequency of 1.6GHz the IPS^2/Watt is the highest.

3.1.2 Power Constrained Core Selection

In the previous section we searched for the best core modes without restricting the overall peak power. Peak power dissipation is important for processor design since the thermal budget of processor, cooling cost, power supply cost and packaging cost depend on the processor's peak power dissipation [23]. We now repeat the search (for the preferred core modes) but with a limit on the power budget. We considered processor peak power dissipation limits similar to those in prior works [5], [24], i.e., 2W and 1.5W. For a peak power constraint of 2W, we obtained four somewhat different core modes, shown in Table 3. Further reducing the power budget to 1.5W, reduces the number of preferred core modes to three with the narrow core excluded as shown in Table 4.

TABLE 2
Core parameters for a power unconstrained design

Core Mode	F (GHz) /V	Buffer size (IQ,LSQ,ROB)	Width (fetch,issue)	Average Power (W)
AC	1.6/0.8	36,128,128	4,4	2.2
NC	2/1	24,64,64	2,2	1.7
LW	1.4/0.8	48,128,256	4,4	2.4
SM	1.2/0.7	12,16,16	1,1	0.82

TABLE 3
Core parameters for a power constrained design (2W)

Core Mode	Freq (GHz)	Buffer size (IQ,LSQ,ROB)	Width (fetch,issue)	Average Power (W)
AC_2W	1.6	36,128,96	4,4	1.6
NC_2W	2	24,64,64	2,2	1.7
LW_2W	1.2	48,192,128	3,3	1.9
SM_2W	1.2	12,16,16	1,1	0.82

TABLE 4
Core parameters for a power constrained design (1.5W)

Core Mode	Freq (GHz)	Buffer size (IQ,LSQ,ROB)	Width (fetch,issue)	Average Power (W)
AC_1.5W	1.4	36,64,64	3,3	1.32
LW_1.5W	1	24,128,128	3,3	1.4
SM_1.5W	1.2	12,16,16	1,1	0.82

3.2 Dynamic Morphing

In the proposed mechanism, all core modes are derived from a single OOO processor core with banked resources, where each bank can be turned on or off and the frequency can be raised or lowered to configure the core to the modes described in Table 2. The buffers that are dynamically resized are the ROB, LSQ and IQ. The fetch width and issue width are also dynamically resized. Decoding units are subsequently powered on/off when the fetch and issue width is resized.

Our baseline execution mode is an average OOO core (AC) that will be dynamically morphed into three other modes, namely, a smaller core (SM), a narrow core (NC) or a larger window (LW) core during runtime. Although each of the four modes has a distinct combination of buffer sizes, fetch and issue width and frequency, they all have the same cache size. This allows us to resize resources while leaving the contents of the cache intact, which in turn allows fine grain switching with low overhead to take advantage of every opportunity for power savings or performance enhancement.

Switching from one mode to another is determined by estimating the power and performance in all other modes based on performance counters' values in the currently executing mode. We reconfigure into another core mode only when the reconfiguration is predicted to result in a sufficiently higher IPS^2/Watt. An in-depth description of our runtime switching mechanism is provided in Section 4.

3.3 Adaptive Buffer Re-sizing

In the proposed morphable core, the ROB, IQ and LSQ are implemented as banked structures where each bank

can be independently powered on/off. The bank size for ROB, LSQ and IQ needs to be determined carefully. A too small a bank may result in larger resizing overhead in terms of layout area and design cost. It has also been shown that a too big bank size causes a significant increase in energy consumption whereas bank sizes of 8, 16 or 32 have only small differences in energy consumption [25]. Therefore, the bank size for the ROB and LSQ has been set to 16 and for the IQ it was set to 8.

4 Runtime Morphing Management

The proposed dynamic morphing/reconfiguration relies on online estimators to select the best core mode for the current needs of the executing application. The previously proposed morphing in [4] has determined the core to morph into by computing the performance (online) without taking power into account. Also, prior configurable architectures did not provide an effective online management scheme for run time morphing decisions [14], [15]. We design an online power and performance estimation scheme that is fast and sufficiently accurate to support the morphing decision process. The key challenge here is the fact that while the program is executing on the current core mode, we need to estimate the IPS^2/Watt for all four core modes.

To estimate power and performance on-line for computing the IPS^2/Watt metric we need to select an appropriate set of hardware performance counters. We start with a large number of counters that have good correlation with power and identify a smaller set of counters that can be used to estimate power and performance online in each of core modes at a sufficient accuracy. Linear regression is then used to derive expressions for estimating the performance and power in the other core modes using hardware performance monitoring counters (PMCs) in the current core mode. Prior works [19], [26], [27] that derived expressions for estimating power and/or performance have considered only a big/little architecture and without changes in voltage/frequency. In this work we show how accurately we could estimate power and performance in each of the cores modes which are architecturally different and are running at different voltages and frequencies.

4.1 Power and Performance Estimations based on PMCs

We use PMCs to estimate the power and performance of each of the different core modes and based on these values and the known frequencies of all the core modes, we compute the metric IPS^2/Watt. The PMCs chosen for our study are listed below.

1) ***IPC***: The longer the processor takes to execute an application, the more power it dissipates.
2) ***Cache activity***: Cache misses at any level in the hierarchy directly impact the performance and in turn, the power consumption. Therefore, the number of hits and misses at both Level 1 ($L1h$, $L1m$) and Level 2 ($L2m$, $L2h$) caches are important when estimating the power and performance.
3) ***Branch activity***: Branch mispredictions cause considerable loss in performance and power. Therefore, we track the number of *Branch mispredictions (Bmp)* and the total number of *Fetched instructions (Fi)*.
4) ***Instructions committed***: Each instruction type (in the ISA) utilizes a separate set of resources. Thus, hardware counters which count the number of Integer (INT), Floating-point (FP), Load (L), Store (St) and Branch (Br) instructions committed serve to estimate the power consumption.
5) ***Buffer-full stalls***: The performance of a processor suffers when the pipeline stalls due to lack of entries in the ROB, LSQ, IQ or RAT.

4.1.1 Shortlisting Performance Counters

Our goal is to find the smallest set of counters that would allow us to estimate the power and performance on each of the core modes as monitoring fewer counters reduces the required hardware overhead. PMC values are available only for the currently executing mode but we need to estimate the power and performance for the other three core modes as well. For example, if the application is currently running on the AC mode, we need to estimate the power of this configuration and the power and performance for the other three configurations, namely the NC, LW and SM modes using the PMCs of the current AC mode.

To select the PMCs that exhibit the highest correlation with the required estimates (of power and performance) and then obtain the corresponding expressions (using linear regression) we select a training set of eight SPEC2006 benchmarks [7] that includes *sjeng, h264ref, soplex, omnetpp, bzip2, namd, gobmk, hmmer* where each of these benchmarks has application phases that prefer one of the four different core modes.

The values of the above listed 14 counters were tracked at fine grain instruction granularity, i.e., after every 2K instructions committed, during the execution of the benchmark. To select a suitable subset of counters, we use an iterative algorithm based on the least squared error. The algorithm seeks to minimize the sum of squares of the differences between the estimated and actual power values. The correlation coefficient R^2 provides a statistical measure of how close the data are to the fitted regression line. Starting with a set of counters, we iterate through all remaining counters to determine which among them is the best to add to the existing set. The candidate counter that yields the highest correlation coefficient is selected.

Figure 6 shows the value of the coefficient R^2 when the PMCs of the average core mode (AC) are used to estimate the power and IPC on the other three core modes. The figure shows that four to five counters are sufficient as the R^2 value saturates afterwards. Note also that estimating the power in the same core yields

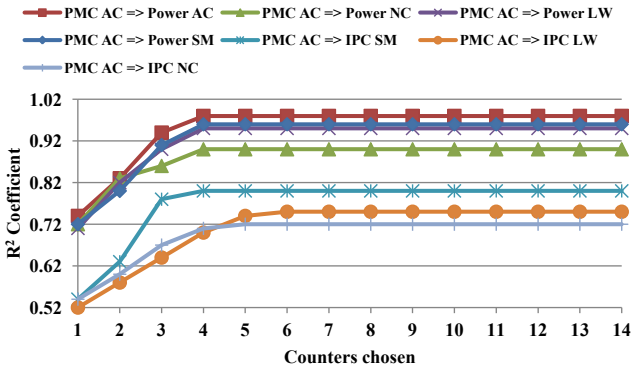

Fig. 6. R^2 coefficient as a function of the number of chosen PMCs. PMC AC => Power NC denotes using the performance counters of the average core mode to estimate the power on the narrow core mode.

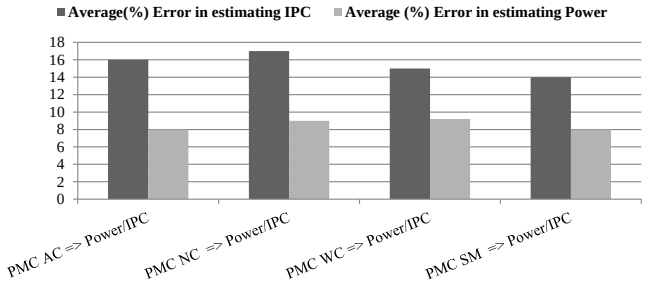

Fig. 7. Average error in estimating power and IPC in all core modes using the PMCs of the current mode. E.g., PMC AC ⇒ Power/IPC denotes the average error in estimating power and IPC in all other core modes using the PMCs of the average core (AC) mode.

a higher R^2 value than in other core modes indicating higher estimation accuracy. For IPC estimation we need a minimum of four counters to estimate the IPC on the SM and NC core modes but R^2 saturates at five counters for the LW core. Similar analysis was carried out for the estimations on the other three core modes.

After selecting the most suitable counters, linear regression was used to derive expressions for the performance and power estimation. Table 5 shows the expressions obtained for estimating the power and IPC for each of other three modes and the power of the AC mode using the values of the PMCs monitored while executing in the AC mode. For the sake of brevity, we show only the expressions when using the PMCs of the AC mode. Similar expressions have been obtained for the other three cases.

4.1.2 Accuracy of Power/Performance Estimation

The accuracy of the power and performance estimations is shown in Figure 7. The estimation error study was conduced for a set of 17 workloads that consists of a mix of SPEC2000 and SPEC2006 benchmarks [7], [28]. We observe that the average error in estimating power

TABLE 5
Power (P) and performance (IPC) estimation for the other three modes using the performance counters values in the AC mode.

Estimated Param	Expression
AC ⇒ Power AC	$1.40{\cdot}10^{-2}{\cdot}\,L1h + 13.81 \cdot IPC$ $+2.95{\cdot}10^{-2}{\cdot}\,St - 1.18{\cdot}10^{-2}{\cdot}\,Bmp - 0.29$
AC ⇒ Power NC	$-1.30 \cdot Bmp - 0.85 \cdot L1m$ $+0.41 \cdot Br + 2.30{\cdot}10^{-2}{\cdot}\,St$ +0.46
AC ⇒ Power LW	$-0.34 \cdot L2m - 1.04 \cdot L$ -0.56 $\cdot Bmp - 1.40{\cdot}10^{-2}{\cdot}L1h$ +0.1
AC ⇒ Power SM	$-3.10 \cdot L1m + 6.67{\cdot}10^{-3}{\cdot}\,IPC$ -4.20 $\times\, 10^{-2} \cdot Bmp + 0.27$
AC ⇒ IPC SM	$0.21 \cdot L1h + 0.91 \cdot IPC$ $+0.11 \cdot L - 0.12 \cdot Bmp$ +4.46
AC ⇒ IPC LW	$0.12 \cdot IPC - 1.81 \cdot L1h$ $+0.31 \cdot St - 1.23 \cdot L1m$ +0.29
AC ⇒ IPC NC	$1.12{\cdot}10^{-1}{\cdot}\,Br + 1.81 \cdot IPC$ $+3.9{\cdot}10^{-2}{\cdot}\,St - 1.18{\cdot}10^{-2}{\cdot}\,L2h + 0.38$

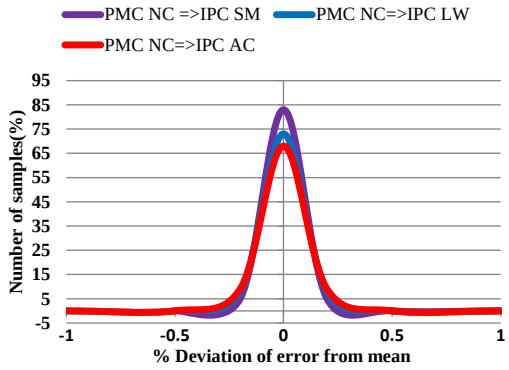

Fig. 8. Estimation error distribution when using the PMCs of the NC mode to estimate the IPC for the remaining three core modes.

is 8% which is significantly lower than the 16% average error in estimating IPC. Although the average estimation error is reasonably low, the actual estimation error may be considerably higher at some time instances and this may cause wrong morphing decisions. Therefore, we analyzed the temporal distribution of errors and the results are shown in Figure 8. This figure depicts the error in IPC estimation for the other three core modes using the PMCs of the NC core mode. We observe that the deviation of the errors from the mean is low and the majority of sample points (80% of them) lie between ± 10% from the mean. This demonstrates that the average error is a sufficiently good indicator for the instantaneous estimation error. In our experiments we have observed very few decision errors.

4.2 Morphing Controller

To enable morphing between different core modes, we need an on-chip controller that governs all the required changes in the core configuration upon morphing. We envision this controller to be a variant of similar controllers for core morphing [4]. It obtains periodically

PMC values from the current core configuration and estimates the IPS^2/Watt for the current and alternative core modes. The estimation is based on the expressions described previously. We assume that the controller includes a multiply and accumulate (MAC) unit that is pipelined and capable of completing 1 MAC operation per cycle, and is power gated when not in use. The IPS^2/Watt estimates are used to determine the best mode to morph into. If the estimated IPS^2/Watt for one of the modes is sufficiently higher than for the current mode, a mode transition is initiated. Mode switches incur some overhead especially if a change in voltage/frequency is required as the PLL must be relocked to the new operating conditions. The overheads imposed by the controller are discussed in the next section. The controller sets the voltage and frequency by placing values in the Voltage Control Register (VCR) and the Frequency Control Register (FCR). The Voltage Regulator Module (VRM) reads the VCR and sets up the new voltage. Similarly, the FCR controls the frequency division within the PLL. Finally, the controller also features a Configuration Control Register (CCR) that directs which units should be powered on and which shall remain off. We would like to emphasize that the morphing controller for the big/little architecture in [4] guarantees the resulting performance to be within 95% of running on the big core, whereas we do not provide such performance guarantees. Still, our proposed morphing scheme achieves significant performance improvements as shown in the results section.

5 Experimental Setup

5.1 Simulator and Benchmarks

To evaluate our proposed morphable core architecture we have used Gem5 as a cycle accurate simulator with integrated McPAT modeling framework to compute the power of the core and L1 caches [29], [30]. We ran experiments using 17 benchmarks from the SPEC2006 and SPEC2000 benchmarks suites [7], [28]. The benchmarks were cross compiled using gcc for Alpha ISA with -O2 optimization. In the simulation experiments we executed 4 billion instructions of each benchmark after skipping the first two billion.

5.2 Determining the Window Size

Power and performance estimates are calculated after a fixed number of committed instructions referred to as *window*. To prevent switching too frequently (e.g., after every window) we wait until the particular phase of the currently executing application has stabilized. To this end, we wait for a fixed number of windows to elapse before making a decision to switch modes. We call this number of windows *history depth*. A decision to switch modes is then made based on the most frequently recommended core mode during the windows in this period. This way, short periods of transient behavior of the executing application will not result in a core mode switch.

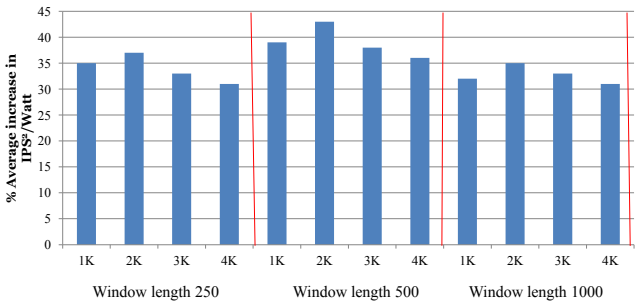

Fig. 9. Percentage increase in IPS^2/Watt over the AC core mode for several values of window size and history depth.

We denote by n the total number of retired instructions during this period where n = *history depth* $\times$ *window length*. For example, if for the past n committed instructions, moving from the average core to the narrow core mode was the most frequent recommendation, we conclude that the application has entered a phase where the narrow core may provide a higher IPS^2/Watt and we switch from the average to the narrow core mode. We have conducted a sensitivity study to quantify the impact of the *window length* and *history depth* on the achieved results. The *window size* and *history depth* combination that yields the highest IPS^2/Watt for the entire program execution would be the best choice. The window length was varied from 250 to 1000 instructions in steps of 250. Within a particular window, the history depth was varied from 1 to 10. For example, a window length of 500 and history depth of 4 means that we make a reconfiguration decision at the end of every 2K instruction (500 $\times$ 4). Figure 9 shows the achieved increase in the average IPS^2/Watt when switching to the preferred core mode from the current AC mode for the SPEC benchmarks. Based on this figure, a window length of 500 and history depth of four provide the largest improvement in IPS^2/Watt. Thus, in all our remaining experiments, a reconfiguration decision is made at the end of every 2K instructions.

5.3 Morphing overhead

In the proposed scheme, the voltage and frequency may change at a fine instruction granularity. The potential overhead of frequent voltage and frequency scaling must be taken into account. Traditionally, DVFS has been applied at coarse instruction granularity, of the order of millions of processor cycles, due to the high overhead that is involved in scaling voltage and frequency using an off-chip regulator [31]. Recently, Kim *et al.* has proposed the use of an on-chip regulator which reduces the time needed for scaling voltage to tens of nanoseconds or hundreds of processor cycles [32]. Using an on chip regulator, a low overhead (hundreds of cycles) DVFS can be performed at a fine grain instruction interval. A hardware-based fine grain DVFS mechanism that uses

an on chip regulator was implemented by Eyerman *et al.* where DVFS was performed upon individual off-chip memory accesses [33]. We assume that such an on chip regulator has been included in the processor design. The authors of [32] have estimated the DVFS latency to be 200 cycles. In our experiments we have used this 200 cycles DVFS latency that constitutes a major component of the overall core morphing overhead. As the latter is design dependent, the result section includes analysis of the impact of higher overheads on the core performance.

Overheads associated with power-gating/power-up of banks of ROB, LSQ, IQ and partial powering on/off of fetch and decode units are also taken into account. When power gating individual units/banks, no dynamic energy is consumed and the static energy consumed by these idle units is low. Power-gating/power-on of all the blocks simultaneously may lead to a sudden power surge and therefore, we assume staggered power gating where only a single bank is gated in a given clock cycle. Powering off a single bank is expected to take tens of clock cycles [27]. The bank selected to be turned off is the one with the smallest number of used entries. If the selected bank is not empty we must wait until all its entries are vacated before switching it off.

Calculating IPS^2/Watt using the PMC-based performance and power estimates involves a computational overhead. To compute IPS^2/Watt, 7 expressions (shown in Table 5) must be evaluated online, which require four MAC operations per expression. The resulting computation overhead is about 30 clock cycles. Once the mode switch decision is made, the controller needs to set the voltage and frequency registers with new values and initiate a mode switch which incurs an additional overhead. Taking into account all the individual overheads we, conservatively, estimate the total overhead to be 500 cycles. As the frequency of core reconfiguration is not high (as will be shown in the next section), even a higher morphing overhead will have a negligible impact.

6 Evaluation

6.1 Power Efficiency

As noted in Section 1, applications exhibit diverse phase behavior and the core mode on which an application runs most efficiently changes during the course of execution. The decision to switch mode is based on the metric IPS^2/Watt that assigns higher significance to performance than to power. To avoid frequent switching, the expected increase in IPS^2/Watt must be at least 5% to trigger reconfiguration. A lower switching threshold will result in more frequent switching for insignificant gains in IPS^2/Watt.

Figure 10 shows the percentage occupancy in each of the four core modes in the unconstrained power case. The figure demonstrates the diversity in the use of four core modes by the different benchmarks and also shows that each of the four modes is highly utilized (more than 40% of the time) in some of the benchmarks. The

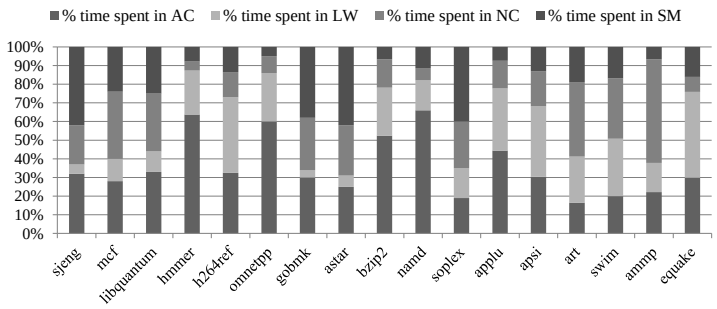

Fig. 10. Tenancy of core modes for the unconstrained power core.

morphable architecture presented in [4] consists of two core modes (OOO and InO) and only benefits applications that have memory intensive phases or phases with high branch mis-prediction rates, as these phases are mapped to the power efficient InO core. Compute intensive benchmarks do not benefit as much from the two-mode morphable architecture as they have very few phases with low performance that could be mapped to an InO core. Our proposed morphable architecture caters to more diverse application phases due to the four distinct core modes that relieve diverse resource bottlenecks.

The percentage improvement in IPS^2/Watt for the SPEC benchmarks executing on our proposed morphable architecture when compared to executing completely on the AC core mode is shown in Figure 11. On an average (using geometric mean), we achieve an IPS^2/Watt improvement of 37% compared to the baseline architecture of the AC core. Benchmarks which are memory intensive or have high branch mis-prediction rates, such as *mcf*, *soplex*, and *art*, achieve larger IPS^2/Watt improvements since they can be mapped to an energy efficient core mode. Compute intensive benchmarks, such as *hmmer, bzip2,* and *h264ref*, also take advantage of the proposed morphable architecture. We observe on average of 34% improvement in IPS^2/Watt for the compute intensive benchmarks compared to the 38% improvement for memory intensive ones. The morphable core should allow a wide variety of applications to run effectively on different core modes. We ran benchmarks from the MiBench and Mediabench [34] suites to test our morphable core capabilities on applications apart from SPEC benchmarks. As seen in Figure 12, we obtain on average 15% IPS^2/Watt improvement demonstrating the benefits of the morphable cores for a wide range of applications. Note however, that the MiBench and Mediabench applications achieve a lower benefit as they do not have as diverse program phases as the SPEC benchmarks.

6.2 Comparison to Other Switching Schemes

We compare our PMC-based fine-grain core mode switching scheme, referred to as *FineGrain_PMC*, to three other switching schemes, namely: (i) Sampling based

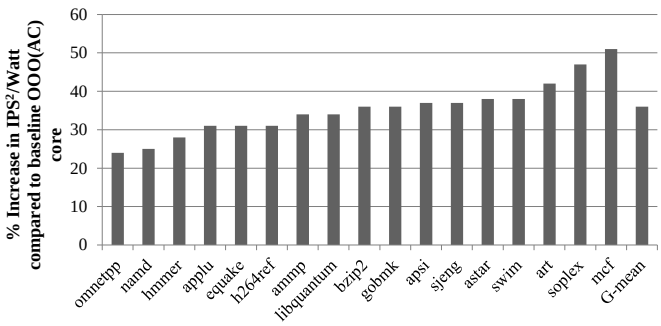

Fig. 11. $\mathrm{IPS}^2/\mathrm{Watt}$ improvement of the proposed morphing scheme compared to execution on AC mode for SPEC benchmarks.

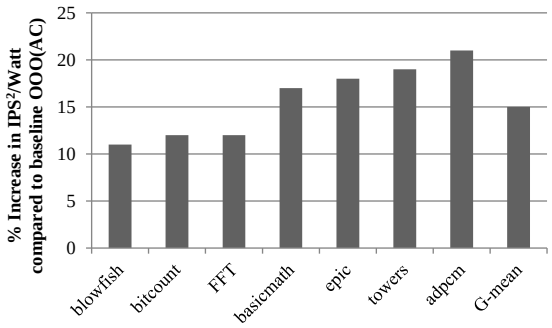

Fig. 12. $\mathrm{IPS}^2/\mathrm{Watt}$ improvement of the proposed morphing scheme compared to execution on AC mode for Mediabench/Mibench benchmarks.

switching within a morphable architecture, referred to as *CoarseGrain_sampling*; (ii) Oracular scheme referred to as *Oracular_Switch*; and (iii) PMC-based switching at coarse grain granularity, referred to as *CoarseGrain_PMC*.

To this end, we have implemented the morphable architecture presented in this paper with morphing decisions made based on sampling. The parameters used for this implementation include a switching interval of 1M instructions and a sampling interval of 10K instructions [5]. We have also implemented the oracular scheme where an oracle determines, every 2K instruction, which is the best core mode for the next interval of 2K instructions. The third implemented scheme is a PMC-based one making switching decisions at a coarse grain granularity of 1M instructions.

Figure 13 compares the IPS^2/Watt and the energy savings obtained for the four switching schemes. The *CoarseGrain_sampling* yields 14% less energy savings than the *FineGrain_PMC*. The reason for lower energy savings for the sampling-based scheme is twofold. First, sampling is wasteful when the program is already running on the best available core. Second, sampling is performed at a coarse grain level thus missing opportunities available at finer granularity. Thus, our PMC-based runtime decision mechanism helps in making the right morphing decisions yielding higher energy savings. We also compare the IPS^2/Watt of the coarse grain and fine grain PMC-based schemes. *FineGrain_PMC* yields an 11%

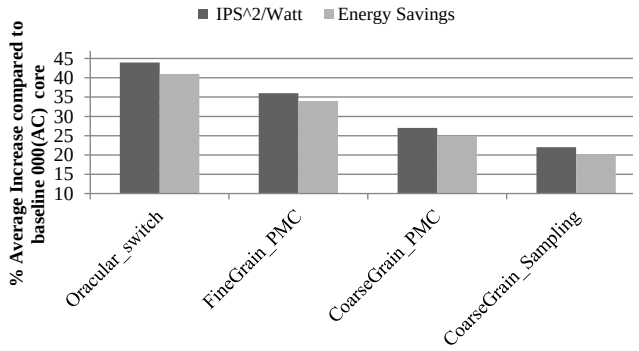

Fig. 13. Comparing the IPS^2/Watt and Energy saving of four morphing schemes.

higher IPS^2/Watt compared to *CoarseGrain_PMC*. This scheme also does much better than *CoarseGrain_sampling* due to a smaller performance overhead in PMC schemes compared to sampling based ones. The oracular scheme achieves a higher IPS^2/Watt, by 10%, than our *FineGrain_PMC* scheme. As the oracular scheme cannot be implemented in practice, it provides an upper-bound for the maximum IPS^2/Watt that could potentially be achieved by our approach.

Figure 14 compares the IPC improvements over the baseline average core for the four switching schemes. The IPC value obtained for the power budget of 2W and 1.5W is normalized to that of the corresponding average core (AC) mode obtained with 2W and 1.5W power constraint, respectively. For the unconstrained case, we observe a 9% improvement in IPC over the baseline (AC) core mode using the *FineGrain_PMC* scheme compared to the 3% achieved by the sampling-based scheme. The oracular scheme shows an upper bound of 12% IPC improvement. For a 2W power budget, a 7% improvement in IPC is achieved by the *FineGrain_PMC* scheme compared to 2.5% for the sampling-based scheme. These results show that our morphing scheme improves performance although its main goal is to improve the performance/Watt.

Figure 15 shows the reduced performance improvement experienced by the different morphing schemes for increasing values of the core morphing overhead. Although our initial estimated cost (overhead) of morphing is 500 cycles, the actual overhead is calculated during the simulation accounting for draining of banked resources. As mentioned previously, the morphing overhead is design dependent and thus it is important to estimate the impact of a higher overhead. Figure 15 shows that as the overhead increases from 500 to 1K cycles, the performance drops by 3.5% for our *FineGrain_PMC* scheme. Higher increases in the morphing overhead result in larger performance losses indicating that switching at fine granularity must have a fast switching mechanism.

Figure 16 shows the number of switches in our 4-mode morphable architecture at various instruction granularities. As expected, a greater number of reconfigurations

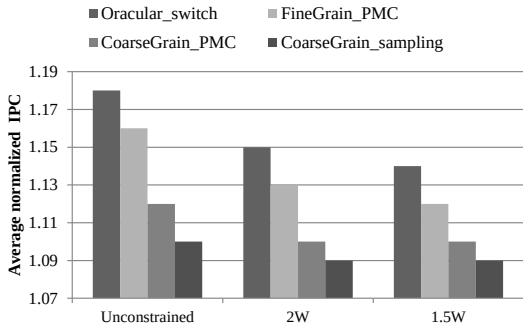

Fig. 14. IPC comparison for power constrained and unconstrained cores for various switching schemes.

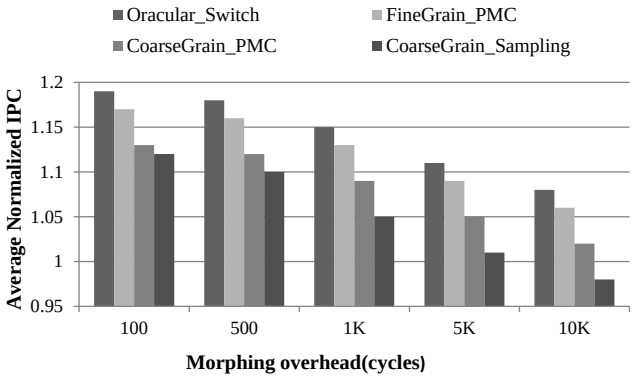

Fig. 15. The impact of morphing overhead on IPC.

takes place at lower instruction granularities, thus yielding higher IPS^2/Watt when compared to coarse grain switching at instruction granularity of 10K and above. For our selected 2K instruction interval the number of switches on average is 12500 in 100M instructions, i.e., after every 2K instructions we have a probability of 25% to perform a mode switch.

Figure 17 shows the impact of power constraints on the IPS^2/Watt improvements and the energy savings achieved by our morphing scheme. For the unconstrained power case, we obtain a 37% IPS^2/Watt improvement and 33% energy savings (compared to the

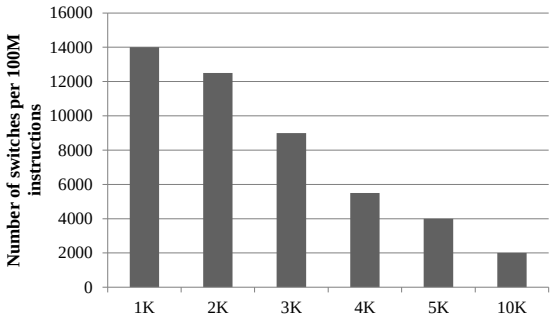

Fig. 16. Number of switches per 100 million instructions for a range of instruction granularities for the power unconstrained 4-mode morphing scheme.

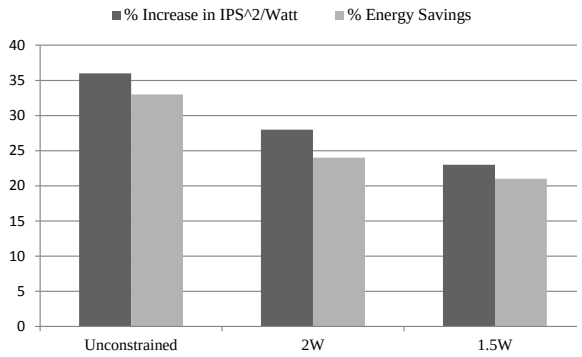

Fig. 17. IPS^2/Watt and energy savings for the power constrained and unconstrained cases compared to execution on the average OOO(AC) core.

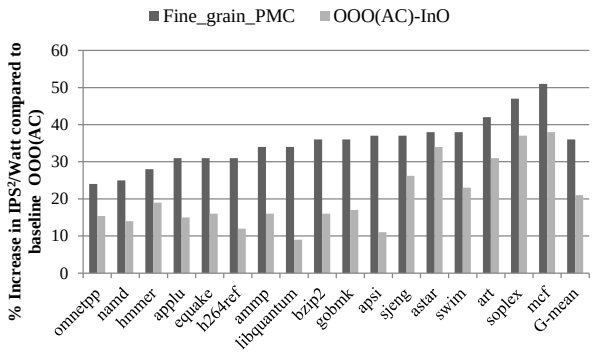

Fig. 18. Comparison of the IPS^2/Watt improvement (over execution on the AC mode) between our *Fine-Grain_PMC* morphable core and the 2-mode morphable core (AC,InO).

average core mode), while for the 2W power budget case, the IPS^2/Watt improvement is only 27% and the energy savings drop down to 24%.

6.3 Comparison of the 4-mode Morphable Core to the Big/Little architecture

To compare our 4-mode morphable core to the previously proposed 2-mode architectures (OOO and InO) [4], [6] we analyzed an OOO/InO morphable architecture similar to the one proposed in [4] but differs in that, both core modes (OOO and InO) share common front and back ends and as a result, architectural states do not need to be transferred. Whenever a decision is made to switch from OOO to InO the fetch width is reduced, half the decoders are powered off, some of the functional units are shut down (e.g., INT ALUs reduced from 4 to 2, FP ALUs reduced from 2 to 1) and the ROB and RAT are powered off. Turning off structures (while moving to InO mode) by clock gating was employed in [6]. When a mode switch happens, the pipeline is drained and several units are powered on/off depending on the core mode we are morphing into, and then instruction execution starts in the new mode.

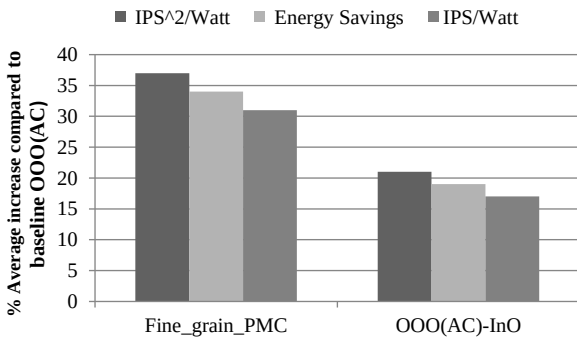

Fig. 19. Comparing the IPS^2/Watt, IPS/Watt and energy savings of the *FineGrain_PMC* and the 2-mode (AC,InO) morphable core.

The selected architectural parameters for the OOO core are those of the AC mode in Table II. The InO core has a fetch/issue width of 2, IQ with 36 entries, and cache sizes and frequency identical to those of the AC core. Note that the InO core that we have modeled is somewhat different from a regular InO core in terms of power efficiency as a regular InO core might have smaller cache sizes, a shorter pipeline and no Load/Store queue. The instruction granularity at which core switching decisions are made was set to 2K instructions after performing a sensitivity analysis similar to that shown in Figure 10. The decision to morph is based on the estimated IPS^2/Watt using a PMC-based estimation mechanism. The IPS^2/Watt improvement achieved by the 2-mode morphable core (AC,InO) (over executing on the AC mode) is compared to that achieved by our 4-mode core in Figure 18. On average, the 2-mode core achieves a 21% increase in IPS^2/Watt versus the 37% achieved by our 4-mode core. Figure 19 compares the average increase in IPS^2/Watt, IPS/Watt and energy savings achieved by our 4-mode *FineGrain_PMC* with those obtained when using the 2-mode (OOO(AC),InO) scheme. On average, the 4-mode scheme achieves a 12% higher IPS/Watt and a 14% higher energy saving compared to the 2-mode (OOO(AC),InO) scheme.

The goal of the next experiment is to determine whether including an InO mode is necessary or it can be replaced by our SM mode that is still an OOO core but has a width of 1. We compare two schemes: one has two core modes, OOO (AC) and OOO (SM) while other has three core modes, namely, OOO (AC), OOO (SM) and InO. We have observed that the 3-mode morphing scheme that includes an InO mode provides an additional 6% IPS^2/Watt improvement. We conclude that the inclusion of the InO core does not sufficiently improve IPS^2/Watt to justify the increased design complexity of supporting the two very different core architecture styles. We have, therefore, excluded the InO mode to keep the micro-architecture simple.

6.4 Benchmark Analysis

In this section we focus on the characteristics of different benchmarks and try to understand why some benchmarks prefer one mode of the morphable architecture over the others. The characteristics that we study include branch mis-predictions, occupancy of buffers (LSQ, IQ, ROB), L2 cache misses and IPC.

Benchmarks with high branch mis-prediction rates have a low ILP and are not expected to benefit from a higher frequency. Such benchmarks would therefore, prefer the small core mode (SM) that runs at reduced frequency and has small resource sizes. To illustrate this we show in Figure 20 the temporal behavior of the benchmark *astar*, that has high branch mis-prediction rates, and compare its performance while running on the SM and AC core modes. During this period of program execution *astar* exhibits a high branch mis-prediction rate and as a result, the IPC difference between the AC and SM modes is small but executing in the SM mode improves the IPS^2/Watt.

Memory-bound applications, e.g., *libquantum, mcf* and *xalancbmk*, experience a large number of L2 misses and generate many parallel loads. Thus, these benchmarks prefer running in the narrow (NC) mode which has a higher frequency and reduced buffer sizes. The higher frequency helps when many independent loads are invoked. A similar observation was reported in [5]. Figure 21 shows a portion of the behavior, as a function of time, of the benchmark *mcf* and compares its IPC when running on the NC and AC modes. The section of program shown in the Figure 21 has a high L2 miss rate. When the L2 miss rate is high, the NC core mode provides a higher IPC than the AC mode since its higher frequency helps in issuing independent loads. The performance difference between the two core modes is large for high L2 miss rates. The IPS^2/Watt is improved by up to 7% by running in the NC rather than the AC mode.

Compute-bound applications, like *bzip2, hmmer* and *h264ref*, have high IPC and their performance is limited by the issue width and buffer resources and not by L2 cache misses or branch miss predictions. Therefore, these benchmarks tend to prefer the Large Window (LW) core mode. Figure 22 compares the execution of the *bzip2* benchmark in the LW and AC modes. The number of times when one the buffers, ROB or LSQ or IQ, became full while running in the AC mode is also shown in the figure. We observe that providing larger resources alleviates the problem of buffers getting full and improves the IPC. Improved IPC and reduced frequency while running in the LW mode, compared to the AC mode, provides an IPS^2/Watt improvement of up to 6%.

7 Conclusions

We have presented in this paper a morphable core design that can assume one of four different core modes.

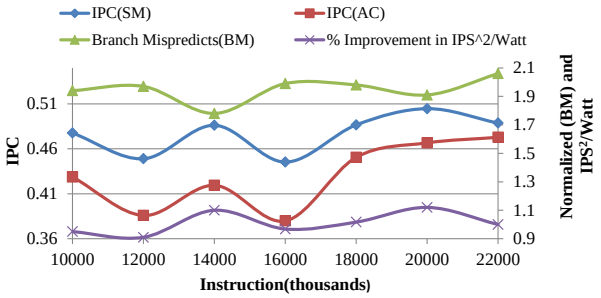

Fig. 20. Analysis of the benchmark *astar* at a fine instruction granularity: comparing its execution in the SM and AC modes.

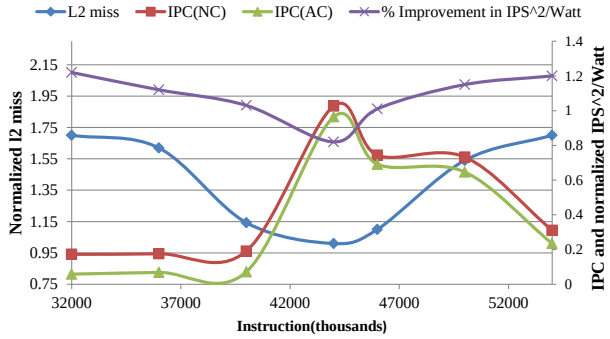

Fig. 21. Analysis of the benchmark *mcf* at a fine instruction granularity: comparing its execution in the NC and AC modes.

Apart from the baseline average core mode, the additional core modes are suited to address most common performance bottlenecks found in the considered benchmarks. Based on a small number of performance counters, a novel runtime mechanism estimates the performance and power across all core modes and uses this information to determine the core mode that offers the best power efficiency. The cache was not resized across core modes to support fast switching from one mode to another enabling fine-grain morphing. We have shown that the proposed four-mode morphing offers higher power efficiency than the two-mode morphing considered earlier. It was also shown that fine-grain switching between core modes outperforms switching at a large instruction granularity which misses power saving opportunities. Our results indicate that the four-mode morphable core achieves an IPS^2/Watt gain of 37% compared to a standard OOO core and 16% higher energy efficiency compared to big/little morphable architectures. Importantly, unlike previous self-morphing schemes that only improve throughput/power but not performance, we also improve the performance by 9%.

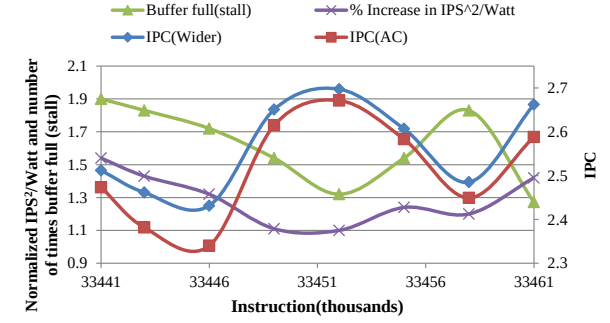

Fig. 22. Analysis of the benchmark *bzip2* at a fine instruction granularity: comparing its execution in the LW and AC modes.

8 Acknowledgment

This research was supported in part by grants (numbers 0903191 and 1201834) from the National Science Foundation. The authors would like to thank the anonymous reviewers for their suggestions and valuable feedback.

References

[1] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-isa heterogeneous multi-core architectures: The potential for processor power reduction," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, 2003, pp. 81–92.

[2] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, 2006, pp. 23–92.

[3] P. Greenhalgh, "Big.little processing with arm cortex-a15 & cortex-a7," *ARM White paper*, 2011.

[4] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. Dreslinski, T. F. Wenisch, and S. Mahlke, "Composite cores: Pushing heterogeneity into a core," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 317–328.

[5] S. Navada, N. K. Choudhary, S. V. Wadhavkar, and E. Rotenberg, "A unified view of non-monotonic core selection and application steering in heterogeneous chip multiprocessors," in *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, 2013, pp. 133–144.

[6] M. A. Khubaib, Suleman, M. Hashemi, C. Wilkerson, and Y. N. Patt, "Morphcore: An energy-efficient microarchitecture for high performance ilp and high throughput tlp," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, 2012, pp. 305–316.

[7] S. Bird *et al.*, "Performance characterization of spec cpu benchmarks on intel's core microarchitecture based processor." in *SPEC Benchmark Workshop*, January 2007.

[8] M. Annavaram, E. Grochowski, and J. Shen, "Mitigating amdahl's law through epi throttling," in *Computer Architecture, 2005. ISCA'05. Proceedings. 32nd International Symposium on*, 2005, pp. 298–309.

[9] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt, "Accelerating critical section execution with asymmetric multi-core architectures," *SIGPLAN Not.*, vol. 44, no. 3, pp. 253–264, Mar. 2009.

[10] M. Monchiero, R. Canal, and A. González, "Design space exploration for multicore architectures: A power/performance/thermal view," in *Proc. of 28th ICS*. ACM, 2006, pp. 177–186.

[11] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron, "Cmp design space exploration subject to physical constraints," in *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, 2006, pp. 17–28.

[12] C. Kim, S. Sethumadhavan, M. S. Govindan, N. Ranganathan, D. Gulati, D. Burger, and S. W. Keckler, "Composable lightweight processors," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 381–394.

[13] M. Pricopi and T. Mitra, "Bahurupi: A polymorphic heterogeneous multi-core architecture," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 22:1–22:21, Jan. 2012.

[14] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. Scottt, "Integrating adaptive on-chip storage structures for reduced dynamic power," in *Parallel Architectures and Compilation Techniques, 2002. Proceedings. 2002 International Conference on*, 2002, pp. 141–152.
[15] D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources," in *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, 2001, pp. 90–101.
[16] D. Shelepov *et al.*, "Hass: a scheduler for heterogeneous multicore systems," *SIGOPS Oper. Syst. Rev.*, vol. 43, pp. 66–75, April 2009.
[17] O. Khan and S. Kundu, "A model to exploit power-performance efficiency in superscalar processors via structure resizing," in *GSLVLSI*, 2010, pp. 215–220.
[18] M. Becchi and P. Crowley, "Dynamic thread assignment on heterogeneous multiprocessor architectures," in *Proceedings of the 3rd conference on Computing frontiers*, ser. CF '06, 2006, pp. 29–40.
[19] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu, "Scalable thread scheduling in asymmetric multicores for power efficiency," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*, oct. 2012, pp. 59 –66.
[20] D. Koufaty *et al.*, "Bias scheduling in heterogeneous multi-core architectures," in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys '10, 2010, pp. 125–138.
[21] E. Grochowski, R. Ronen, J. Shen, and P. Wang, "Best of both latency and throughput," in *Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference ons*, oct. 2004, pp. 236–243.
[22] E. Grochowski and M. Annavaram, "Energy per instruction trends in intel microprocessors," *Technology@ Intel Magazine*, vol. 4, no. 3, pp. 1–8, 2006.
[23] V. Kontorinis, A. Shayan, D. M. Tullsen, and R. Kumar, "Reducing peak power with a table-driven adaptive processor core," in *Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture*. ACM, 2009, pp. 189–200.
[24] A. Venkat and D. M. Tullsen, "Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor," in *MICRO*, 2014, pp. 121–132.
[25] H. Wang, I. Koren, and C. Krishna, "Utilization-based resource partitioning for power-performance efficiency in smt processors," in *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, july 2011, pp. 1150–1163.
[26] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, 2012, pp. 213–224.
[27] S. Srinivasan, L. Zhao, R. Illikkal, and R. Iyer, "Efficient interaction between OS and architecture in heterogeneous platforms," *SIGOPS Oper. Syst. Rev.*, vol. 45, pp. 62–72, February 2011.
[28] SPEC2000, "The Standard Performance Evaluation Corporation (Spec CPI2000 suite)."
[29] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.
[30] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, 2009, pp. 469–480.
[31] J. Park, D. Shin, N. Chang, and M. Pedram, "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, pp. 419–424.
[32] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, 2008, pp. 123–134.
[33] S. Eyerman and L. Eeckhout, "Fine-grained dvfs using on-chip regulators," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 1, pp. 1:1–1:24, Feb. 2011.
[34] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems," in *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, 1997, pp. 330–335.

Sudarshan Srinivasan received the Masters degree in Electrical and Computer Engineering from the University of Massachusetts, Amherst in 2011, where he is currently working towards the Doctor of Philosophy degree in Electrical and Computer Engineering with emphasis on power/performance evaluation in Multicore processors. He was a research intern at the Department of Informatics and Mathematics, University of Passau, Germany and at the Institute for Research in Electronic and Embedded system (IRSEEM) Rouen, France. He previously interned with the Performance/Power modeling teams at AMD and Samsung Austin R&D.

Nithesh Kurella is currently a student in the Department of Electrical and Computer Engineering's Masters program at the University of Massachusetts Amherst. He received his B.E degree from the Visvesvaraya Technological University, Bangalore. His research interests include Computer Architecture and Fault Tolerance Systems. He interned at Performance/power modeling team at Samsung Austin R&D previously.

Israel Koren (M'76 - SM'87 - F'91) is currently a Professor of Electrical and Computer Engineering at the University of Massachusetts, Amherst. He has been a consultant to numerous companies including IBM, Analog Devices, Intel, AMD and National Semiconductors. His research interests include Fault-Tolerant systems, Computer Architecture, VLSI yield and reliability, Secure Cryptographic systems, and Computer Arithmetic. He publishes extensively and has over 250 publications in refereed journals and conferences. He is an Associate Editor of the VLSI Design Journal, and Sustainable Computing: Informatics and Systems. He served as General Chair, Program Chair and Program Committee member for numerous conferences. He is the author of the textbook "Computer Arithmetic Algorithms," 2nd Edition, A.K. Peters, 2002 and a co-author of "Fault Tolerant Systems," Morgan-Kaufman, 2007.

Sandip Kundu is a Professor of Electrical and Computer Engineering at the University of Massachusetts, Amherst. Previously, he was a Principal Engineer at Intel Corporation and Research Staff Member at IBM Corporation. He has published more than 200 papers in VLSI design and CAD, holds 12 patents, and has co-authored multiple books. He served as the Technical Program Chair of ICCD in 2000, co-Program Chair of ATS in 2011, ISVLSI in 2012 and 2014, DFT in 2014. Prof. Kundu is a Fellow of the IEEE and has been a distinguished visitor of the IEEE Computer Society.