Online Mechanism for Reliability and Power-Efficiency Management of a Dynamically Reconfigurable Core

Sudarshan Srinivasan, Israel Koren, Sandip Kundu
ECE Department, University of Massachusetts at Amherst, MA, USA
{ssrinivasan, koren, kundu}@ecs.umass.edu

Abstract—**Previous studies have shown that the best way to achieve high throughput/Watt of a single threaded application is by running it on an asymmetric multicore processor (AMP). AMPs feature cores that are tuned for specific workload characteristics. To increase efficiency, the core that offers the best power-performance trade-off for the executing thread is chosen. To reduce the overhead of thread migration, we have previously proposed a morphable core that can morph into multiple core types. In this study, apart from power-performance efficiency, we also consider the reliability of the different core types as indicated by their vulnerability to soft-errors. We show that the best core type for power-efficiency may not be the best for reliability. Accordingly, we develop a multi-objective thread migration strategy to determine the best core type considering power efficiency and reliability. To support runtime decision making, we have developed online estimators for reliability and power efficiency based on performance monitoring counters. In keeping with the existing literature, we use the architectural vulnerability factor (AVF) as the metric for reliability and instructions-per-second^2/Watt as the metric for power efficiency. For the multi-objective optimization we use a Cobb-Douglas production function. Our results indicate that the proposed runtime mechanism for reliability and power-efficiency improves, on the average, the throughput/Watt of applications by 24% and reduces the Soft-Error Rate (SER) by 12% compared to the best static execution.**

I. Introduction

As the focus in the microprocessor industry has shifted from high performance computing to energy efficient computing, Asymmetric Multi-core Processors (AMPs) proved to offer a viable path forward. AMPs consist of a set of diverse cores where each core is specialized for power and performance characteristics to suit specific application needs [1], [2]. Resource needs of a typical workload change during the course of its execution. Thus, to remain power efficient, an application needs to hop from one core type to another. There has been a number of prior publications to address dynamic core hopping for maximizing power efficiency of an AMP [3], [4], [5], [6]. There are two lines of research in this direction. The first assumes that the cores in an AMP are distinct, executing multiple concurrent workloads. The optimization here focuses on thread swapping. In the second line of research, it is assumed that each core can dynamically reconfigure into different core types requiring power gating or powering on of various units or resource banks. The issues here are: (*i*) what is the best core type to switch into, and (*ii*) what is the best instruction granularity for switching.

In this paper, we follow the second approach and consider dynamic core reconfiguration from the perspectives of *both* throughtput/Watt and vulnerability to soft-errors. Recent research has shown that diversity in resource needs within an application can be catered to by having distinct core types which address different processor bottlenecks [3]. These core types are non-monotonic where each core has different performance and power characteristics. It was further shown that more opportunities for power savings exist at fine grain instruction granularity but this relies on online core reconfiguration to minimize switching overhead [4]. Our target core features four core configuration modes (CCMs) which were chosen specifically because of their demonstrated performance/Watt benefits. Each CCM represents a partial reconfiguration of the core involving (*i*) resizing resources through power-gating, and/or (*ii*) changing voltage and frequency.

Power efficiency and vulnerability to soft-error often lead to a trade-off in core reconfiguration. For example, a workload that exhibits frequent cache misses achieves a higher power efficiency under lower voltage and frequency conditions that lead to a reduced power without a decrease in performance as the performance bottleneck is the result of cache misses and not low frequency. Even though this may improve power efficiency, it also leads to *greater vulnerability* to soft-error due to the lower voltage. Recent literature has shown that reduced feature sizes and aggressive power management lead to increased soft error rate (SER) [7], [8]. Other studies report adverse impact of dynamic voltage and frequency scaling (DVFS) on SER [9], [10], [11], [12]. This paper is the first to investigate dynamic core reconfiguration with non-monotonic core types for optimizing two objectives simultaneously: improving throughput/Watt and reducing vulnerability to soft errors. Given that a thread is running on a certain core type, if by switching to a different core type the power efficiency can be improved without increasing the soft-error vulnerability, we should always do so. Similarly, if switching to another core configuration mode can reduce the soft-error vulnerability without decreasing the power efficiency, we should always do so. This approach ultimately leads to choices where we cannot improve one of these objectives without sacrificing the other one. We use a Cobb-Douglas production function [13] for trading-off competing optimization goals.

Measuring power efficiency and soft-error vulnerability requires quantitative metrics. For power efficiency we use IPS^2/Watt, where IPS represents the number of instructions executed per second. Soft error vulnerability is measured in terms of Architecture Vulnerability Factor (AVF) introduced in [14]. The AVF of a processor depends on the utilization of processor structures during runtime. Prior research has shown that AVF varies at instruction granularities within applications

[15]. AVF sensitivity to microarchitectural resource sizes was studied in [16]. These studies demonstrate that AVF varies from workload to workload and for a given workload, it varies from one CCM to another. Thus, in a dynamic core reconfiguration, there is a need to measure AVF online and take proactive actions, which is a one of the contributions of this paper.

We summarize the contributions of our work below:

- We present a runtime mechanism for dynamically switching among non-monotonic core types, at fine grain granularity for simultaneously balancing throughput/Watt and SER.
- We present a runtime estimation mechanism for estimating the throughput/Watt and AVF of all CCMs. This mechanism estimates throughput/Watt and AVF in each of the CCMs that are micro-architecturally different and run at different voltage/frequency, using the performance monitoring counters (PMCs) of the host CCM.
- We present a comparative study of the proposed runtime scheme against several other alternatives to demonstrate its efficiency.

II. RelatedWork

In this section we summarise selected prior research works on asymmetric multicores, reconfigurable architectures, thread scheduling in AMPs and run time AVF computations which are relevant to our proposed approach.

A. Asymmetric Multicore Processors (AMPs)

Prior research on AMPs has shown the need for having the right set of core types that can improve the performance and lower the energy of both single-threaded and multi-threaded applications [1], [6], [17], [18]. Kumar *et al.* proposed having a right mix of heterogeneous cores such that each program phase of an application is mapped to a core that achieves the largest power reduction and performance improvement [1]. Navada *et al.* targeted accelerating single-threaded workloads by performing complete design space exploration and finding a set of heterogeneous cores that would maximize performance/Watt [3]. Suleman *et al.* proposed AMP architectures consisting of a big core and several small cores where the big core is used to eliminate serial bottlenecks [19]. The above AMP architectures are designed to switch at coarse grain instruction granularity or granularity of application phase change. The penalty of thread switching is high in these architectures, and consequently, they are unable to take advantage of power reduction opportunities at fine instruction granularity.

B. Reconfigurable Architectures

Previous research has explored core reconfiguration at run time to better adapt to changing demands of applications and improve performance/Watt [4], [5], [6], [20]. Fusing of several smaller cores to a bigger Out-of-order (OOO) core at run time is explored in [6], [20]. A reconfigurable architecture that can support fine grain switching to improve performance/Watt of applications has been recently proposed in [4], [5]. Lukefahr *et al.* have proposed an architecture in which, an OOO core is turned into an InOrder (InO) core at run time at fine grain instruction slices [4]. The above reconfigurable core can only switch between two extreme architectures (OOO and InO). Switching between only these extreme architectures does not cater to the demands of all applications. Srinivasan *et al.* proposed a dynamic heterogeneous architecture that reconfigures among multiple CCMs, with different voltage and frequency [21]. It was shown that DVFS combined with resource scaling, provides improved performance/Watt when switching at fine granularity. All the above works have considered switching between different CCMs for improving performance or performance/Watt. The need for dynamic micro-architecture reconfiguration to balance power, performance and reliability in heterogeneous cores has also been discussed in [22], [23]. However, these papers do not present a complete on-line solution for dynamic reconfiguration at fine granularity as presented in this work.

C. Thread-to-Core Mapping in AMPs

Prior research has employed performance counter (PMC) based online thread to core mapping schemes. In these schemes, the power and performance of threads on different AMP core types are estimated using PMC statistics which are collected on the host core [2], [4], [17]. In this work we follow this approach for online power/performance computation but ours differs in the following ways. Unlike previous works which estimate performance/power on the same core or consider only two core modes (OOO and InO), we allow a larger number of AMP CCMs, which are architecturally different and differ in voltage/frequency. Our online scheme estimates power and performance in all the CCMs using the PMCs of the current CCM unlike prior reconfigurable architectures that consider only performance when making online decisions [4].

D. Runtime AVF Computation

The probability that a soft error will lead to a user visible error is computed based on the processor's AVF. The soft error rate (SER) of a system is computed as the product of the raw SER and individual components' AVFs [14]. Prior works have shown how AVF may be estimated during runtime using a set of PMCs [14], [24]. Biswas *et al.* proposed quantized AVF estimation to track AVF variations at fine grain granularity using a few PMCs. They then showed that the AVF of processor components such as ROB, LSQ and IQ, vary significantly during the course of program execution [15]. Soundararajan *et al.* studied the frequency and voltage impact on SER when applying DVFS [9]. They concluded that using only performance/Watt goals to choose an operating point for DVFS will not lead to improved system reliability. Prior works have developed online estimation schemes for computing AVF for a single core. In this work we develop linear regression based expressions for estimating the AVFs of all the CCMs which are architecturally different and run at different voltage/frequency using the PMCs of the host CCM.

III. Proposed Architecture

As the cores in an AMP are power constrained, a core cannot be designed such that it operates at the highest frequency and at the same time has large micro-architectural features.

TABLE I. CORE PARAMETERS

Core Configuration Mode	F (GHz) /V (Volt)	Buffer size (IQ,LSQ,ROB)	Width (fetch,issue)
CCMA	1.6/0.8	36,128,128	4,4
CCMN	2/1	24,64,64	2,2
CCML	1.4/0.8	48,128,256	4,4
CCMS	1.2/0.7	12,16,16	1,1

For example, to support applications with high ILP, the size of ILP extracting buffers should be increased or the width of the pipeline should be resized. These changes may require additional circuits which in turn may require reducing the core frequency due to the thermal dissipation limit (TDP). Thus, designing AMP cores requires careful balancing of micro-architectural parameters.

Prior works have performed AMP core sizing to relieve processor performance bottlenecks using design space exploration [3], [21]. They found that if there is only one core type, it would resemble an existing commercial super-scalar OOO core, where the core parameters are chosen to strike a balance between achieving sufficient ILP and the frequency. The alternate cores are accelerator core types that would be designed to relieve bottlenecks that arise in the base core. In contrast, we consider a single core that dynamically morphs into four core configuration modes (CCMs), shown in Table I, as in [21]. As the objective of this work is to develop an online scheme to effectively assign threads to different core types to strike a balance between throughput/Watt and reliability, the CCMs in Table I are only representatives of how diverse the CCMs could be. Our approach can be extended to other flavors of CCMs.

We observe from Table I that along with an average OOO (CCMA) mode which is our baseline core, the design choices include: (*i*) a narrow core mode (CCMN) that has smaller fetch/issue width but higher frequency, targeting application phases with low ILP and accelerating sequential phases; (*ii*) a larger window core mode (CCML) that has increased window size and targets applications with window bottleneck; and (*iii*) a small core (CCMS) that caters to low performance phases at fine grain instruction granularity (as observed in [4]) and also benefits benchmarks which have high branch mis-prediction rates. For our reconfigurable architecture, we *do not change the cache content* to avoid costly cache migration overhead. We have fixed the D-cache size to 64KB and the L2 cache size to 2MB.

A. Dynamic Reconfigurable Core

A high-level view of our reconfigurable micro-architecture is shown in Figure 1. As the IPC varies at fine granularity, the usage of various processor resources, such as ROB, LSQ and IQ, also varies. To aid in fine grain switching between different CCMs, we have a single processor core whose resources are banked; they can be turned on or off and the frequency can be raised or lowered to configure the core to each of CCMs shown in Table I. The four CCMs have different buffer sizes, varying fetch and issue widths and run at different frequencies. However, they all have the same cache size to enable fast switching thus taking every available opportunity for energy saving or performance improvement. For the proposed architecture to seamlessly move between four different core configurations, we have implemented banked structures for ROB, LSQ and IQ. Each bank unit in a buffer can be independently powered on/off. The size of the banks for ROB, LSQ and IQ need to be designed carefully. Too small a bank might result in significant overhead in terms of layout area and design cost. Prior research has shown that having larger bank sizes might lead to increased energy consumption whereas bank sizes of 8, 16 or 32 have limited energy consumption [25]. Based on these considerations, we set the bank size for ROB and LSQ to be 8 and for IQ to be 16.

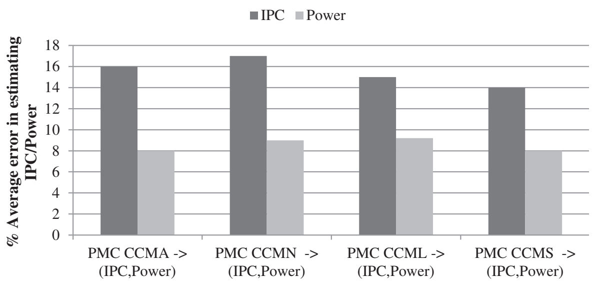

Fig. 2. Average error in estimating power/IPC across all CCMs using the PMCs of the host CCM. For example, PMC CCMA ⇒ (IPC,Power) denotes the average error in estimating power and IPC on all other CCMs using the PMCs of CCMA.

IV. ONLINE MANAGEMENT OF THE RECONFIGURABLE ARCHITECTURE

We have implemented a run time management scheme that uses PMCs to select the core to reconfigure into. To strike a trade-off between power and performance in choosing the right CCM, we compute the metric IPS^2/Watt of all the CCMs. Unlike the commonly used energy efficiency metric IPS/Watt, the IPS^2/Watt assigns a higher weight to performance and will more likely avoid switching to a CCM that reduces power considerably but also lowers significantly the performance [26]. To compute IPS^2/Watt online, we use PMCs to estimate the power and performance for all the CCMs. Likewise, we use PMCs to predict the AVF for all the CCMs, which would allow us to compute the SER. A key novelty in this work is that we need to estimate IPS^2/Watt and AVF for all other CCMs, using the PMCs of the host CCM, unlilke previous works that compute AVF or power/IPC only on the same core.

A. Power and Performance Estimation using PMCs

We compute power and performance on-line for all the CCMs using a few PMCs that have high correlation to power. These PMCs have been identified in [2], [27] and include: IPC counter, cache activity counters (L1 and L2 cache misses/hits), branch misprediction counter and instructions fetched counters to track the power dissipated due to wrong path speculative instructions. We also monitor counters related to individual types of instruction executed (e.g., integer, floating-point, load, store and branch) which would help to estimate the power of the resources used by these instructions. Finally, we consider stall counters that track the buffer stalls due to lack of free entries in LSQ, IQ and ROB. These counter values are collected at fine grain instruction granularity of 2K instruction. Linear regression expressions are then derived to estimate power and performance for each of the CCMs. To derive the linear expressions we chose a set of 8 training workloads from SPEC2006 benchmark [28] whose application phases have affinity to one of the CCMs. The training workloads include

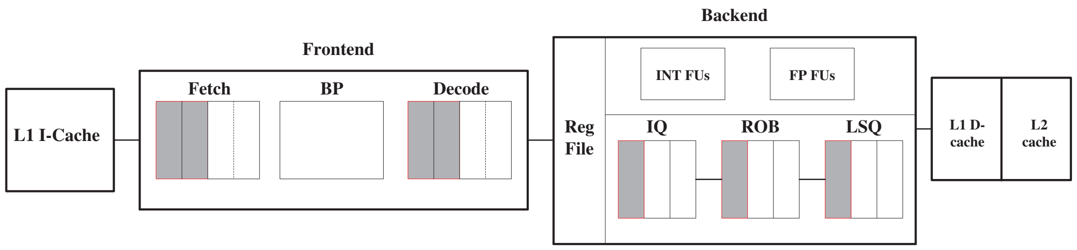

Fig. 1. High-level view of the reconfigurable architecture. The shaded units are reconfigured during run time.

TABLE II. ACCURACY OF PERFORMANCE AND POWER ESTIMATION ACROSS ALL CCMS

Core configuration mode	R^2 coefficient (IPC)	R^2 coefficient (power)
PMC CCMA $\Rightarrow$ IPC/Power	0.84	0.91
PMC CCMN $\Rightarrow$ IPC/Power	0.79	0.94
PMC CCML $\Rightarrow$ IPC/Power	0.83	0.90
PMC CCMS $\Rightarrow$ IPC/Power	0.85	0.92

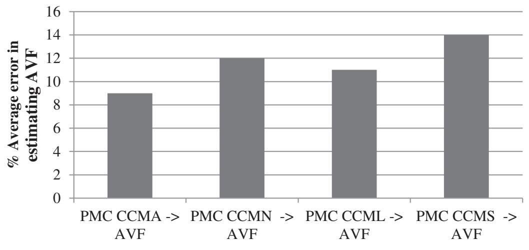

Fig. 3. Average error in estimating AVF in all the CCMs using the PMCs of the host CCM. E.g., PMC CCMA $\Rightarrow$ AVF denotes the average error in estimating the AVF for all the other CCMs using the PMCs of CCMA.

TABLE III. ACCURACY OF AVF ESTIMATION ACROSS ALL CCMS

Core configurational Mode	R^2 coefficient
PMC CCMA $\Rightarrow$ AVF	0.92
PMC CCMN $\Rightarrow$ AVF	0.84
PMC CCML $\Rightarrow$ AVF	0.85
PMC CCMS $\Rightarrow$ AVF	0.81

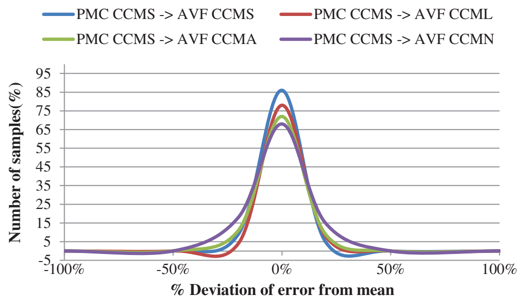

Fig. 4. Distribution of error when using the PMCs of CCMS to compute the AVF for all other CCMs.

sjeng, h264ref, soplex, omnetpp, bzip2, namd, gobmk and *hmmer*. The correlation coefficient (R^2) provides a statistical measure of goodness of fit of the regression line. Higher R^2 values indicate a high correlation between the estimated value and the selected PMCs. We obtained an average R^2 value of 0.91 and 0.82, for the power and performance, respectively, as shown in Table II. Table II shows the average R^2 obtained when estimating power and IPC across different CCMs using the PMCs of the host CCM. The accuracy in estimating power/performance is shown in Figure 2, for a mix of 17 workloads combining SPEC2000 and SPEC2006 benchmarks [28]. In this figure we show the average error in estimating power and performance for each of other CCMs using the PMC of the host CCM. The average error in estimating performance and power is 16% and 8%, respectively, across all CCMs. While this error may be considered high, what we are concerned about is making the right decision about which CCM to run on. It has been shown that the accuracy of decisions that is based on the relative values of the estimations, is much higher than the estimations themselves [29].

B. AVF Estimation using PMCs

As was proposed in [15] we use PMCs to track the AVF of the three buffers (ROB, LSQ and IQ) in each of CCMs using the PMCs of the host CCM. The counters used to estimate AVF include store buffer utilization, ROB utilization, branch miss-prediction, IQ utilization, ROB empty cycles and stores flushed before DTLB response. Linear regression expressions were then derived for estimating the AVF in each of the CCMs. The same set of training workloads were chosen as mentioned earlier for power/performance estimation to derive trained expressions to compute AVF online. Due to high correlation of the above mentioned counters with AVF, an average R^2 value of 0.86, considering all CCMs, is obtained as shown in Table III. Table III shows the average R^2 obtained when estimating AVF across different CCMs using the PMCs of the host CCM. The accuracy in estimating the AVF across different CCMs is shown in Figure 3, for a mix of SPEC2000 and SPEC2006 benchmarks [28]. We observe that the average error in estimating AVF is 11%, across all CCMs. The instantaneous error during run time may be higher for some time intervals. To this end we analyzed the temporal distribution of the error, shown in Figure 4. For the sake of brevity, we show in Figure 3 only the error distribution for CCMS which had the worst case average error. Figure 4 shows that for 75% of sample points the error is within $\pm$ 15% from the mean. This indicates that the error at the time of decision making is not high.

C. Metrics to Achieve Trade-off between Throughput/Watt and Reliability

Our on-line scheme consists of estimating the performance/Watt and SER online using PMCs as discussed previously. IPS^2/Watt of all the CCMs are computed online using the PMCs of the current CCM. The effective SER of each CCM is then computed as shown below [14]:

Effective_SER = $AVF \times$ Raw_SER

Since not all soft errors affect the output, the *Effective*_SER is derived from the Raw_SER by multiplying it by the AVF of the processor. The Raw_SER is the total expected bit flip rate in the processor due to soft errors. The AVF is a function of the number of processor state bits that are required for architecturally correct execution [14]. These bits are called Architecturally Correct Execution (*ACE*) bits. Bits that are not critical to program execution are termed as *unACE* bits and they include, for example, discarded bits due to mis-speculation.

The Raw_SER dpends on the voltage and frequency. Scaling of voltage has an exponential relationship with soft errors, where lower voltage leads to increased error rate [30], [31]. Previous studies have shown that SER has a linear relationship with frequency [31], [32]. We therefore, express the RAW_SER as a function of both voltage and frequency as shown below [31]:

Raw_SER(f,v) = $(f/f_{max}) \times e^{-c_0(v-v_{max})}$ * *Raw_SER*$_0$

where f_{max} and v_{max} are the maximum voltage and frequency values for the core, respectively, and c_0 is a constant coefficent [31]. *Raw_SER*$_0$ is the SER computed at maximum voltage and frequency. The decision to switch between the CCMs is made by computing a reliability power efficiency (RPE) metric online. The proposed RPE metric follows the Cobb-Douglas function [13] to trade off between two simultaneous objectives, namely, IPS^2/Watt and SER.

$$RPE = (\text{IPS}^2/\text{Watt})^a \times (\text{Effective\_SER})^{-b}$$

The exponents *a* and *b* are weights that control RPE, where *a* and *b* are positive numbers such that *a*+*b*=1. Normalization of IPS^2/Watt and Effective_SER is necessary, as RPE is a dimensionless metric. The calculated IPS^2/Watt and Effective_SER are normalized to their values for the baseline CCM (CCMA). Sensitivity analysis was performed to determine common values for *a* and *b* across all CCMs. Individually we analyzed each CCM and found out the values of *a* and *b* that provides the maximum value for the RPE function. We then computed a weighted average taking into account workload occupancies in each of the CCMs to arrive at common weights for *a* and *b* across CCMs that maximizes the function RPE. Figure 5 shows the RPE value against *a*. It can be observed that RPE is maximized at values *a* = 0.6 and *b* = 0.4. The values of the weights could also be set based on the designers requirements as discussed in the result section.

For runtime management, the RPE metric is computed online for each CCM every 2K instructions interval. We call this scheme, PMC_RPE. Please note that the RPE computation on current CCM and other CCMs is based on PMCs that are available only for the current execution mode. A decision to switch configuration is based on maximizing RPE across CCMs. To prevent too frequent switching from one CCM to another, the potential gain in RPE from switching core modes must exceed a certain threshold. The target threshold was found to be 4% based on a sensitivity study.

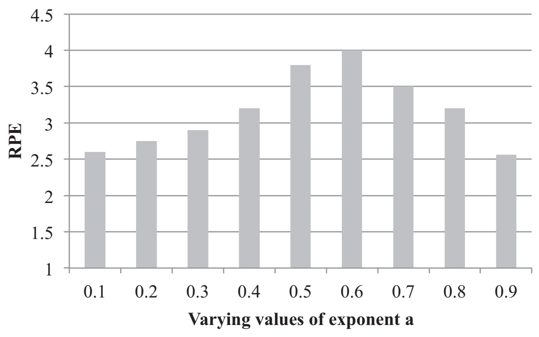

Fig. 5. Sensitivity study showing the RPE as a function of the exponent a.

V. Experimental Setup

A. Simulator and Benchmarks

We evaluate our proposed scheme using the Gem5 cycle accurate simulator [33] integrated with McPAT power model [34]. A modified Gem5 simulator was used to collect AVF information and power values online. SPEC2006 and SPEC2000 benchmarks were used for our experiments which were cross compiled for Alpha ISA with -O2 optimization. We ran our experiments for 4 billion instructions after skipping the first 2 billion.

B. Reconfiguration Overhead

In our proposed scheme, the core reconfiguration takes place at fine granularity of 2K instructions. To check the viability of reconfiguration at fine granularity, we must evaluate the overhead involved in core reconfiguration. The CCMs operate at different voltage/frequency and each time there is a CCM switch, we must take into account the penalty for changing the voltage and frequency. Kim *et al.* proposed an on-chip regulator that can support voltage scaling at the order of hundreds of processor cycles [35]. We assume therefore, a DVFS latency of 200 cycles as estimated in [35]. We also estimate the overheads associated with power-gating/power-up of banked buffer resources when switching from one CCM to another. Power-gating the banks does not consume any dynamic power and the static power consumption in these banked resources is small. Power-gating a particular bank takes tens of cycles [25]. The bank that was chosen to be turned off is the one that has the least entries. If the bank selected to be turned off is not empty, the entries in the bank need to be vacated before turning off.

An on-chip controller is used for the run time PMC collection and for making core reconfiguration decisions. The controller periodically collects the PMC values for power, performance and AVF computation. It contains a multiply and accumulate unit (MAC) unit to perform these computations and we assume that the MAC unit is pipelined and can perform 1 MAC operation per cycle. We have included the overhead for these MAC operations. Once the mode switch decision is made, the controller needs to set the voltage and frequency registers with new values and initiate a CCM switch which incurs an additional overhead. Taking into account all the above overheads, the total overhead is conservatively estimated to be 500 cycles with the actual overhead calculated in run time taking into account draining of a bank resources. We also show in the results section, the impact of a higher switching overhead.

VI. Results

A. Throughput/Watt and SER results

We evaluated the benefits of the proposed scheme using our 4-CCM architecture and compared the obtained throughput/Watt and SER to those of the baseline core, the CCMA.

Figure 6 shows the improvements in IPS/Watt across all benchmarks using our PMC_RPE scheme. We observe that on average, an IPS/Watt benefit of 24% is achieved. Benchmarks which are highly control bound (e.g., *astar, sjeng*) incur more stalls due to branch mispredictions. During periods of high misprediction, these benchmarks deliver the best throughput/Watt in the CCMS mode, as higher frequency or buffer resources do not address this bottleneck. Similarly, memory intensive benchmarks (e.g., *mcf, soplex, libquantum*) contain low ILP phases due to significant data L2/TLB misses. These phases achieve better throughput/Watt in CCMN or CCMS. As benchmarks like *mcf* have independent loads, the CCMN mode with reduced resources sizes and increased frequency helps to improve performance. On the other hand, compute bound benchmarks (e.g., *bzip2, hmmer* and *h264ref*) have high IPC resulting in window bottleneck and thus prefer to execute on CCML as the performance of these benchmarks is limited by buffer resources and issue width.

The decrease in Effective_SER compared to running in the baseline CCM across all benchmarks, is shown in Figure 7. An Effective_SER reduction of 12% is achieved compared to the baseline. Figure 8 shows the time spent in each of the CCM, demonstating that the CCMs are well utilized across all benchmarks.

There is trade-off between throughput/Watt improvement and Effective_SER reduction as explained below. Memory intensive workloads, such as *libquantum*, that have a high L2 miss rate and numerous parallel loads show high occupancy in CCMS and CCMN. When run on the baseline CCMA, *libquantum* has enough memory level parallelism to use the buffer resources but occupancy of instructions in the shadow of L2 misses increases as well. This would lead to increased SER (resulting from increased AVF) when operated on the baseline CCM. If these sections of the program can be run on CCMS, it can provide a reduced SER (lower AVF) and a higher throughput/Watt compared to CCMA. For *libquantum* we obtain 12% SER reduction and 26% improvement in throughput/Watt compared to the baseline core (CCMA) as shown in Figures 7 and 6.

Another memory intensive benchmark (*mcf*) shows a different behavior from *libquantum* as it experiences a significant number of branch miss-predictions dependent L2 misses. Branch miss-predicts instructions are not part of ACE bits and thus do not affect AVF. Thus, benchmarks like mcf use CCMN to accelerate independent loads with increased frequency, but obtain lower SER reduction compared to *libquantum*, as the occupancy of ACE bits is still limited by the miss-predicted branches which do not affect AVF. For *mcf* we obtain 5.3% reduction in SER and 34% improvement in throughput/Watt

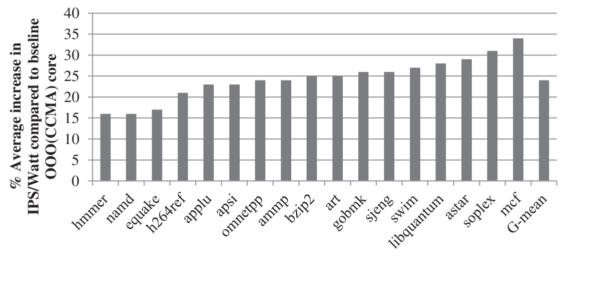

Fig. 6. % Average increase in IPS/Watt compared to the baseline OOO(CCMA) using PMC_RPE scheme

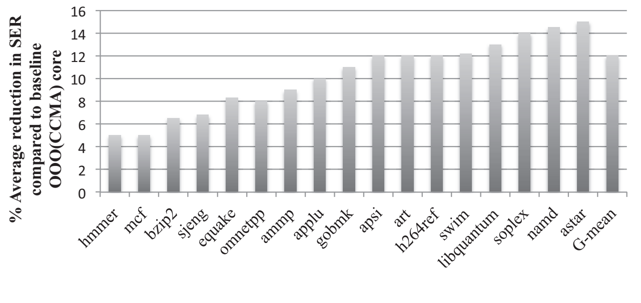

Fig. 7. % Average decrease in Effective_SER compared to baseline OOO(CCMA) while switching based on PMC_RPE scheme

compared to the baseline as shown in Figures 7 and 6. Compute bound applications such as *hmmer* have high ILP and would prefer the CCML mode. However, higher ILP in compute bound applications could cause the processor to bring more instructions into the pipeline, resulting in an increase in the number of ACE bits and thus increasing AVF. CCML operates at a lower frequency but a similar voltage compared to the baseline CCMA. Voltage and frequency scaling can have contrasting effect on Effective_SER as explained previously for each of the CCMs. Thus, switching into a new CCM can increase AVF over the baseline but might not increase RAW_SER. For *hmmer* we obtain 5% reduction in SER and 16% improvement in throughput/Watt. Figure 9 shows the overall average improvement in IPS/Watt, performance increase (6%) and Effective_SER decrease compared to the baseline.

We also experimented with different weight factors for the RPE function above and below our chosen values, to compare the improvement in throughput/Watt and average SER reduction for different weight factors. Figure 10 compares three different weight factors for the RPE function. Changing the values of *a* and *b* to 0.7 and 0.3, respectively, results in an increase in IPS/Watt by 4% and a decrease in Effective_SER by 6% over the previous values of *a*=0.6 and *b*=0.4. Increasing the weight of SER to *b*=0.6, results in a 3% further SER reduction but an 8% lower IPS/Watt increase compared to the chosen weight factors. Based on the importance of a particular metric to the designer, the weight factors can be modified to achieve the required IPS/Watt increase and SER reduction.

B. Comparison to Alternative Switching Schemes

We compare our PMC_RPE scheme with three other schemes, namely, oracular scheme (Oracular), PMC-based

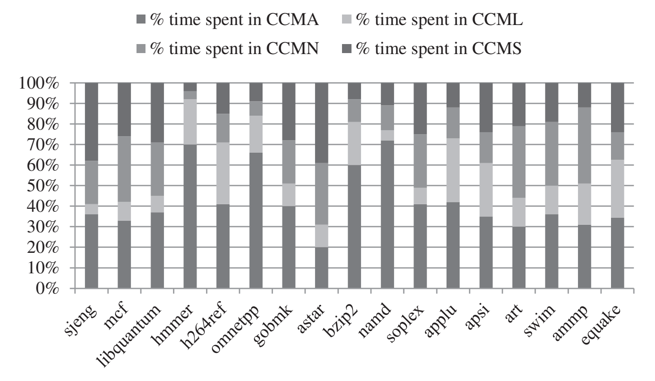

Fig. 8. % of Time spent in different CCMs

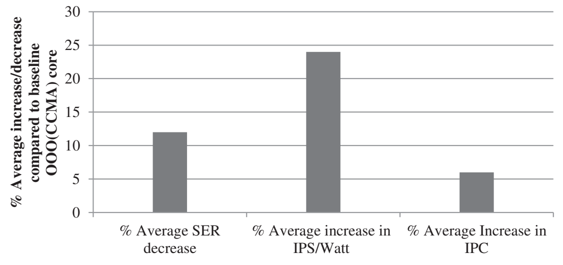

Fig. 9. Increase in throughput/Watt, performance and decrease in Effective_SER while switching based on PMC_RPE scheme compared to the baseline OOO(CCMA).

coarse-grain switching scheme (CoarseGrain_RPE) and PMC-based fine-grain switching scheme (PMC_IPS^2/Watt). The oracular scheme is implemented such that, every 2K instruction an oracle determines what is the best CCM to switch into for the next 2K instructions. This scheme provides an upper-bound as it can not be implemented in practice. As seen from Figure 11, the oracular scheme achieves a 11% higher average IPS/Watt and 4% higher average SER reduction compared to our fine grain (PMC_RPE) scheme. The CoarseGrain_RPE scheme makes switching decision using the RPE metric at coarse grain granularity of 1M instructions and it can not take advantage of power benefits at fine granularity. AVF also varies at fine granularity but these changes are not captureed by this scheme. CoarseGrain_RPE achieves 8% lower IPS/Watt compared to the PMC_RPE scheme. The PMC_IPS^2/Watt scheme would try to switch into a CCM that provides higher power efficiency without loosing much in performance. This scheme does not consider SER while switching. Switching based on IPS^2/Watt provides 7% more IPS/Watt than the PMC_RPE scheme as its only goal is to find a CCM that improves throughput/Watt. We also obtain 6% reduction in Effective_SER compared to the PMC_RPE scheme.

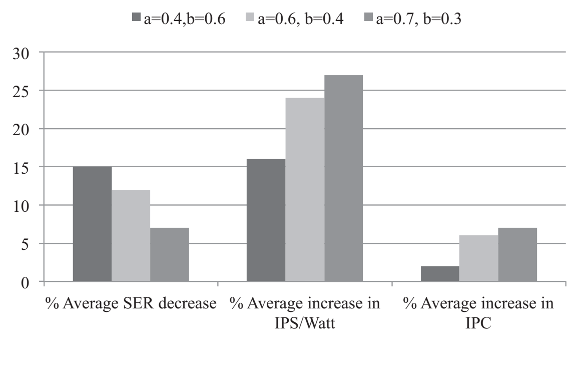

Fig. 10. Variation in Throughput/Watt, performance and Effective_SER while switching based on PMC_RPE scheme.

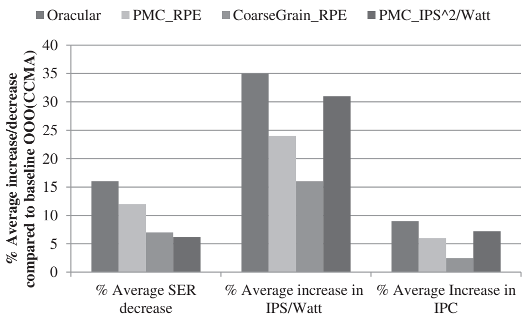

Fig. 11. Comparison to various other switching scheme

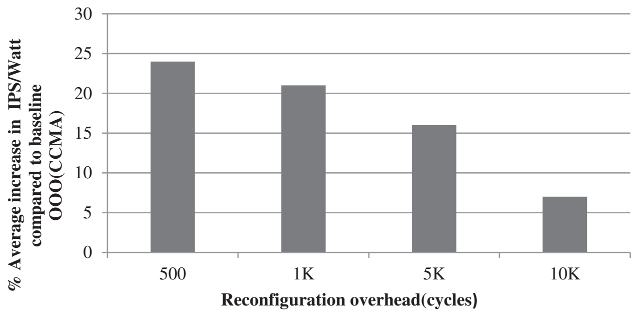

Fig. 12. Reconfiguration overhead and its impact on throughput/Watt gains.

C. Reconfiguration Overhead

We demonstrate the impact of increasing reconfiguration overhead on IPS/Watt in Figure 12. The overhead of mode reconfiguration is highly design dependent and we show the impact of a higher overhead on the throughput/Watt. As mentioned earlier, the average mode reconfiguration overhead was assumed to be 500 cycles but the actual overhead may vary taking into account draining of banked resources during a CCM switch. Figure 12 shows how an increase in reconfiguration overhead would affect the IPS/Watt using our PMC_RPE scheme. At an overhead of 1K cycles, the IPS/Watt decreases by 3% compared to its value for 500 cycles. As the overhead further increases to 5K cycles, an IPS/Watt gain of 16% is achieved with a 7% reduction compared to its value for an overhead of 500 cycles. We also counted the number of switches taken by our reconfigurable architecture. We observed that for our selected instruction granularity of 2K instructions, we obtain 9200 switches per 100M instructions, i.e, our probability of making a mode switch is 18.4% every 2K instructions.

VII. Conclusion

In this paper we have presented an online management scheme for a reconfigurable architecture that strives to achieve a balance between power efficiency and reliability. The target reconfigurable architecture features four non-monotonic core configurations with varied micro-architectural resources, voltage and frequency. SER and throughput/Watt change rapidly during the course of program execution. The proposed runtime management scheme estimates on-line the power, performance and AVF based on a few performance counters to determine which is the best CCM to run on for improving both power efficiency and AVF. Our results indicate that having diverse non-monotonic core types can increase the throughput/Watt of application by 24% while also providing a 12% reduction in SER compared to static execution on the baseline core.

References

[1] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *PACT*, 2006.

[2] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu, "Scalable thread scheduling in asymmetric multicores for power efficiency," in *SBAC-PAD*, 2012.

[3] 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 *PACT*, 2013.

[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 *MICRO*, 2012.

[5] 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 *MICRO*, 2012.

[6] 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.

[7] A. Dixit and A. Wood, "The impact of new technology on soft error rates," in *IRPS*, 2011.

[8] S. Borkar, T. Karnik, and V. De, "Design and reliability challenges in nanometer technologies," in *DAC*, 2004.

[9] N. Soundararajan, N. Vijaykrishnan, and A. Sivasubramaniam, "Impact of dynamic voltage and frequency scaling on the architectural vulnerability of gals architectures," in *ISPLED*, 2008.

[10] R. Vadlamani, J. Zhao, W. Burleson, and R. Tessier, "Multicore soft error rate stabilization using adaptive dual modular redundancy," in *DATE*, 2010.

[11] F. Firouzi, A. Azarpeyvand, M. E. Salehi, and S. M. Fakhraie, "Adaptive fault-tolerant dvfs with dynamic online avf prediction," in *Microelectronics Reliability*, 2012.

[12] X. Xu, K. Teramoto, A. Morales, and H. Huang, "Dual: Reliability-aware power management in data centers," in *CCGrid*, May 2013.

[13] P. H.Douglas, "Cobb, charles w and douglas, paul h," in *The American Economic Review*, 1928.

[14] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *MICRO*, 2003.

[15] A. Biswas, N. Soundararajan, S. S. Mukherjee, and S. Gurumurthi, "Quantized avf: A means of capturing vulnerability variations over small windows of time," in *IEEE Workshop on Silicon Errors in Logic-System Effects*, 2009.

[16] A. Nair, S. Eyerman, L. Eeckhout, and L. John, "A first-order mechanistic model for architectural vulnerability factor," in *ISCA*, 2012.

[17] 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.

[18] K. Rangan, M. Powell, G.-Y. Wei, and D. Brooks, "Achieving uniform performance and maximizing throughput in the presence of heterogeneity," in *HPCA*, 2011.

[19] 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.

[20] 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.

[21] S. Srinivasan, I. Koren, R. Rodrigues, S. Kundu *et al.*, "A runtime support mechanism for fast mode switching of a self-morphing core for power efficiency," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*. ACM, 2014, pp. 491–492.

[22] T. Ramirez, E. Herrero, N. Axelos, J. Carretero, N. Foutris, D. Sanchez, and X. Vera, "Mitigating lower layer failures with adaptive system reconfiguration," in *MIXDES*, 2012.

[23] T. Sato and T. Funaki, "Dependability, power, and performance trade-off on a multicore processor," in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, 2008.

[24] K. R. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic prediction of architectural vulnerability from microarchitectural state," in *ISCA*, 2007.

[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] E. Grochowski and M. Annavaram, "Energy per instruction trends in intel microprocessors," *Technology@ Intel Magazine*, vol. 4, no. 3, 2006.

[27] B. Su, J. Gu, L. Shen, W. Huang, J. Greathouse, and Z. Wang, "Ppep: Online performance, power, and energy prediction framework and dvfs space exploration," in *MICRO*, 2014.

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

[29] A. Annamalai, R. Rodrigues, I. Koren, and S. Kundu, "An opportunistic prediction-based thread scheduling to maximize throughput/watt in amps," in *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*. IEEE Press, 2013, pp. 63–72.

[30] D. Zhu, R. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," in *ICCAD*, 2004.

[31] F. Firouzi, M. E. Salehi, F. Wang, and S. M. Fakhraie, "An accurate model for soft error rate estimation considering dynamic voltage and frequency scaling effects," in *Microelectronics Reliability*, 2011.

[32] F. Irom and F. Farmanesh, "Frequency dependence of single-event upset in advanced commercial powerpc microprocessors," in *Nuclear Science, IEEE Transactions on*, 2004.

[33] 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.

[34] 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 *MICRO*, 2009.

[35] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *HPCA*. IEEE, 2008, pp. 123–134.