A Runtime Support Mechanism for Fast Mode Switching of a Self-Morphing Core for Power Efficiency

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

Rance Rodrigues
NVIDIA
Beaverton, Oregon, USA
rance.rodrigues@gmail.com

ABSTRACT

Asymmetric multicore processors (AMPs) consist of cores executing the same ISA, but differing in microarchitectural resources, performance, and power consumption. As the computational bottleneck of a workload shifts from one resource to the next, during its course of execution, reassigning it to the core where it runs most efficiently can improve the overall energy efficiency. Simulation studies show that the performance bottlenecks can shift frequently, often within a few thousands cycles. With frequent core hooping, the overhead of thread migration becomes significant. To mitigate this overhead, we propose a morphable core that can assume one of four possible configurations to address the dominant performance bottlenecks, while retaining the same cache and registers. This way the architectural state remains intact while the morphable core is reconfigured in resources and frequency. We then implement a runtime scheme to decide the best configuration to run on and switch configuration as necessary. Simulation results indicate that on the average, the proposed scheme results in performance/watt improvement of 41%.

Categories and Subject Descriptors

C.1.3 [**Processor Architectures**]: Adaptable architectures

Keywords

Asymmetric multi-core processors; Hardware Performance Counters; Core Morphing

1. INTRODUCTION

Previous research on Asymmetric Multi-core Processors (AMPs) has identified multiple opportunities for exploiting the processors' asymmetry. A key to AMP efficiency is having a mix of core types with different power and performance characteristics so that a program phase of an application is mapped to a core which achieves the best energy efficiency. [2].

Most AMP design references feature a high performance Out-of Order (OOO) core combined with a smaller low performance and energy efficient In-Order (InO) core. ARM's big.Little^{TM}is an example of such a design [1]. However, a workload's performance on a particular core type depends on larger set of processor resources since workloads are diverse in nature. Thus, a larger set of diverse cores that could address the shifting resource needs during various program phases of an application is needed.

Navada *et al.* explored non-monotonic AMP cores that are specifically designed to address bottlenecks that result in poor performance [4]. Their set of AMP cores consist of a narrow core which suits application with low ILP, a large window core for application phases which have window bottlenecks (reorder buffer, issue queue), a wider core for phases that have width bottlenecks (fetch and issue width) and finally, a normal super-scalar OOO core to target most common scenarios. Core configurations similar to these four core types form the base for our work in this paper.

Recent research has shown that power savings opportunities can often be found in much shorter time scales, typically measured in thousands of instructions[3]. To support such fine grain switching, self-morphing core architectures where an OOO core dynamically morphs into an InO core during run time have been proposed [3]. Morphing was considered only between two extreme architectures, i.e., a wide issue OOO core and an InO core. However, as discussed previously, the resource bottlenecks or excesses can be quite diverse thus missing out on important power performance optimization opportunities. In this paper we present a novel morphing architecture in which the core dynamically switches between four OOO core execution modes of varying fetch and issue width, buffer sizes (IQ, LSQ, ROB) and clock frequency.

2. SELF MORPHING ARCHITECTURE

Our self morphing architecture consist of a single 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 the four core modes shown in Table I. Our baseline execution mode is an average OOO core (AC) that will be dynamically morphed into one out of the other three modes, namely, wider core (WC), narrow core(NC) and larger window core (LW) during run time. The four modes have dif-

PACT'14, August 24–27, 2014, Edmonton, AB, Canada.
ACM 978-1-4503-2809-8/14/08 .
http://dx.doi.org/10.1145/2628071.2628124

Table 1: Architecture parameters of the four core modes.

CoreType	ClockPeriod(ns)	Buffer size (IQ,LSQ,ROB)	Width (fetch,issue)
AC	0.6	32,128,128	3,4
NC	0.5	32,64,64	2,2
LW	0.7	48,128,384	4,4
WC	0.7	32,128,128	6,6

ferent buffer sizes, varying fetch and issue widths and also run at different frequencies. Decoding units are also subsequently powered on/off when fetch and issue width are resized. Still, all four modes have the same cache sizes with I-cache size of 64KB, D-cache size of 64KB and the L2 cache size of 2MB. This allows us to resize resources dynamically leaving the content of the cache intact, which in turn allows fine grain switching.

3. RUNTIME SUPPORT FOR SELF MORPHING

The computational resource requirements of an application change during its execution and are not available beforehand. Hence, an online mechanism that is computationally fast and reasonably accurate, is required in guiding the application to the right core configuration at run time. The decision to select the most appropriate core configuration for the current application phase is made by computing the throughput/watt for each of the configurations and selecting the one that provides the highest throughput/watt (measured in IPS^2/Watt). To calculate the throughput/watt, we use a hardware performance counters based linear regression model to estimate the power and performance on each of the different configurations. The counters used in this work consist of IPC, Cache activity, Branch activity, Instructions committed and Buffer-full stalls. Our results indicate that we can predit power and performance in each of the core modes with reasonable accurancy with average errors of 15% and 8% for predicitng power and IPC, respectively.

4. MORPHING OVERHEAD

Switching between one morph core to another is based on the IPS^2/Watt metric. The decision to morph from one core mode to another is taken if the IPS^2/Watt computed for one of the other three core modes is greater by at least 5% than that for the currently used core mode. When a core mode switch happens, we drain the core pipeline and resize the core buffers and frequency (depending on the core we are morphing into) and start executing instructions in the newly morphed core. Fine grain DVFS with on-chip regulator is employed to switch voltage/frequency at fine granularity with an overhead of 200 cycles. The overhead of banking for ROB, IQ and LSQ is taken to be 10 cycles to power off a particular bank. Once the bank to be turned off is selected, we wait until the existing instructions in this bank has committed before switching off that bank. On average, this morphing overhead has been estimated to be 500 cycles.

5. RESULTS

In this section we evaluate our proposed self morphing scheme. The core parameters considered are listed in Table I. We use gem5 as our cycle accurate simulator with integrated McPAT modeling framework to compute the power of the cores and caches. We evaluate our proposed scheme using the SPEC benchmark suite. The benchmarks were compiled using gcc for Alpha ISA with -O2 optimization. All our evaluations were carried out by running each benchmark for 2 billion instructions after skipping the first 2 billion.

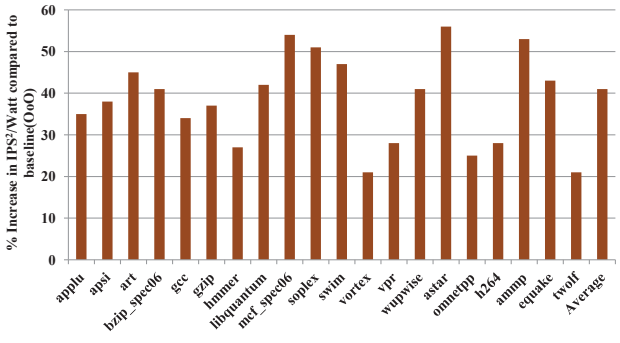

Figure 1: % Improvement in IPS^2/Watt of proposed self-morphing scheme with 4 modes

The percentage improvement in IPS^2/Watt of our proposed self-morphing scheme with four core types when compared to the baseline where applications are run completely on the Average core type (AC) is shown in Figure 1. On average we achieve IPS^2/Watt benefits of 41% and performance improvement of 10% compared to the baseline.

6. CONCLUSION

In this paper, we presented a self-morphing core that can morph into one of four different core types. The selected core types are suited to address most common performance bottlenecks found in SPEC benchmarks. Based on a small number of performance counters, a novel co-designed runtime system predicts the performance and power across all core modes based on statistics obtained from the current core type (whichever it may be) and uses this information to morph into the core type that offers the best energy efficiency. Our results indicate that we can achieve an average throughput/watt gain of 41% over executing on a standard core.

7. REFERENCES

[1] P. Greenhalgh. Big.little processing with arm cortex-a15 and cortex-a7.

[2] R. Kumar et al. 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*, pages 81–92. IEEE, 2003.

[3] A. Lukefahr et al. Composite cores: Pushing heterogeneity into a core. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2012.

[4] S. Navada et al. 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*, pages 133–144. IEEE Press, 2013.