Exploring Heterogeneous-ISA Core Architectures for High-Performance and Energy-Efficient Mobile SoCs

Wooseok Lee[1], Dam Sunwoo[2], Christopher D. Emmons[2],
Andreas Gerstlauer[1], and Lizy K. John[1]
[1]The University of Texas at Austin, [2]ARM Research

ABSTRACT

In this paper, we explore opportunities to increase energy efficiency by providing cores with restricted functionality, but without necessarily impacting performance. We aim to achieve this by removing support for complex but less frequently executed instructions since instruction mixes used by real-world workloads are often heavily biased. We investigate which instructions are worthwhile to remove by analyzing a subset of instructions in the ARM ISA and their corresponding logic burden in the microarchitecture. We propose a heterogeneous-ISA system to achieve energy efficiency without performance degradation using a system architecture that combines both full- and reduced-ISA cores. Results show that by providing the flexibility of heterogeneous-ISA cores, the proposed system can improve energy efficiency by 12% on average and up to 15% for applications that do not require NEON support, all without performance overhead.

1. INTRODUCTION

Current mobile systems-on-chips (SoCs) take advantage of heterogeneity at the system level by switching between high-performance and energy-efficient cores [1]. These heterogeneous systems match application demands with core types to maximize energy efficiency. In such systems, the heterogeneity lies in the microarchitecture. In bigger cores, more transistors are spent on components that improve performance, such as branch predictors and out-of-order processing capabilities. In smaller cores, reducing the amount of performance-relevant resources can, however, be detrimental to some workloads.

An alternate approach is to implement energy-efficient cores by restricting functionality instead of giving up performance. Specifically, by reducing resources that have less impact on performance, power dissipation can be alleviated without losing significant performance. The benefit of implementing certain features in the Instruction Set Architecture (ISA) is highly dependent on workloads. Not all instructions are frequently used by every workload. If a particular workload favors specific instructions that are not directly supported by the hardware, performance dramatically decreases. By contrast, there is little performance degradation if those instructions are not frequently used.

In this paper, we explore opportunities for systems comprised of heterogeneous reduced-ISA cores to improve energy efficiency. We identify candidate instructions that are complex to implement in state-of-the-art ARM-based systems. We further evaluate performance degradations if candidate instructions are removed. If performance degradation is large, it is not worthwhile to consider removing them despite the potential power benefit. We evaluate system performance with a reduced instruction set running several benchmarks. Results show that some subsets of instructions are critical to performance while others are not essential since in most cases, their usage frequency is low enough to not impact performance significantly.

We propose a heterogeneous-ISA architecture to obtain energy benefits while maintaining performance across a wide range of workloads. In our proposed system architecture, reduced-ISA cores remove hardware support for complex instructions, which increases energy efficiency but requires trapping and emulating of non-supported instructions. When unsupported instructions are infrequent, a workload runs on the reduced-ISA core to reduce energy. By contrast, when unsupported instructions are prevalent, the workload is migrated to a traditional full-ISA core. This dynamic core switching allows the system to avoid the performance degradation and energy inefficiency of software-emulated instructions. As long as they are not performance-critical, a compiler can thereby optimize binaries to remove unsupported instructions and thus maximize residency on the reduced-ISA core. Our results show that workloads without performance critical instructions spend most of their execution time on reduced-ISA cores, achieving energy savings of up to 15%. Workloads with frequent use of unsupported instructions execute exclusively on full-ISA cores with no change in performance or energy consumption. On average, 12% energy savings at little to no performance cost are observed across a variety of benchmarks, where applications migrate between reduced- and full-ISA cores depending on dynamically varying instruction usage.

2. REDUCED-ISA CORE DESIGN

We first identify the instructions that greatly impact logic complexity for reduced-ISA core design and evaluate the performance impact after removing them. As a case study, we select the ARM V7 ISA, and one of the performance-oriented ARM processors, Cortex-A15, as a baseline. We find that the overall number of instructions in an ISA is not as crucial for logic reduction as would be expected. Our study instead focuses on the specific semantics required by particular instructions that contribute to large logic within the microarchitecture. Since the processor is unaware of the instruction until decoded, we regard the fetch stage as an irrelevant block. We analyze the logic burdens of various instructions and provide the instruction distribution of both Android applications and SPEC benchmarks. The details of our ISA analyses can be found in [12]. Based on these analyses, we identify four sets of candidate instructions for removal as follows:

- NEON instruction set extensions including floating-point (FP) and SIMD operations.
- Load and store multiple (LDM/STM) instructions.

GLSVLSI '17, May 10-12, 2017, Banff, AB, Canada
 ISBN 978-1-4503-4972-7/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3060403.3060408

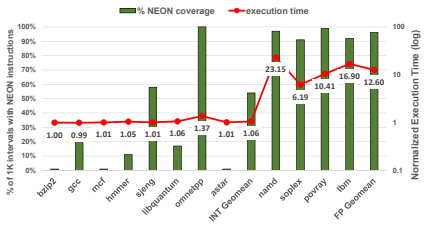


Figure 1: Performance of ISA without NEON instructions.

Figure 2: Performance of ISA without conditional instructions.

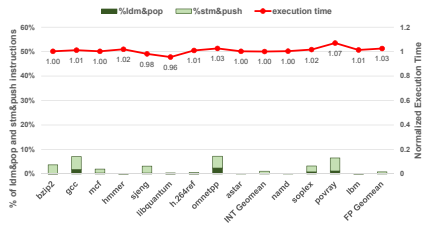


Figure 3: Performance of ISA without Load/Store Multiple instructions.

- Predicated instructions [2].
- DSP-like instructions (QADD, SSAT, etc.).

Reduced-ISA Performance Impact: We define reduced instruction sets as case studies and observe the performance overhead of each with SPEC CPU2006 benchmarks (bzip2, gcc, mcf, hmmer, sjeng, libquantum, omnetpp, astar, namd, soplex, povray, and lbm). We modify LLVM to generate binaries in which candidate instructions are removed. The execution time was measured on an Arndale board [3].

NEON instructions are a large source of logic to sustain performance. In other words, the performance loss from removing NEON instructions is also large if they are heavily used. To measure the performance impact of NEON instructions, we compare the execution time of SPEC benchmarks with and without NEON instructions. Note that since we could not fully remove all NEON code from some of the pre-compiled libraries, there still exist a negligible number of NEON instructions. Figure 1 shows the detailed performance loss and original NEON instruction usage for each benchmark along with NEON coverage.

For floating-point benchmarks, the performance slowdown is significant, ranging from about 6 to 23 times. When removing floating-point support in the ISA, there is no other choice for the compiler than to resort to soft-float emulation. By contrast, for integer benchmarks, which sporadically use NEON instructions, the performance degradation is less than 7% with the exception of *omnetpp*. Since *omnetpp* contains a large number of NEON instructions, the 37% performance degradation is understandable. These profiling result match prior work [5].

Figure 2 shows results for a reduced ISA that excludes predicated instructions. Avoiding to use of conditional instructions is done by disabling the *SimplifyCFG* optimization pass in LLVM. When removing this optimization pass, the compiler also loses the chance to generate optimal code. Thus, the slowdown is not solely from removing conditional instructions. However, since we observed that only a very small number of instructions are changed compared to full-ISA binaries, we regard such effects as negligible.

Overall, the performance loss is less than 5% except for *hmmer*. Removing conditional instructions results in an increased amount of branches in the code. Results indicate that the branch predictor of a modern high-performance mobile processor is good enough to handle the removal of predicated instructions. In case of *hmmer*, further analysis showed that the single hot loop in which most of the execution time is spent shows significantly worse branch predictor performance, causing the severe performance degradation.

Load and store multiple instructions provide a way to reduce a number of consecutive load/store operations. However, since modern instruction prefetch/fetch and out-of-order logic can hide such an increase in instruction count, the penalty of replacing them with separate load/store instructions is somewhat mitigated. Figure 3 shows the detailed performance with and without load/store multiple instructions. The performance loss is negligible across benchmarks.

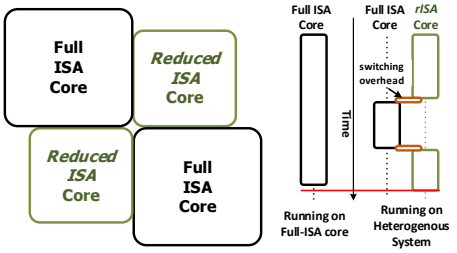


Figure 4: Full- and reduced-ISA core heterogeneous system.

We find that current compilers hardly use DSP-like instructions for most types of workloads. Some of them are never used by the LLVM compiler back-end in the first place. Consequently, it is meaningless to conduct performance comparison with or without these DSP-like instructions. However, despite the fact that we skipped the performance evaluation, a reduced ISA without DSP-like instructions to reduce the logic burden is still valid.

3. HETEROGENEOUS-ISA SYSTEM

We propose a heterogeneous system that contains a combination of both full- and reduced-ISA cores (Figure 4). Traditional full-ISA cores execute applications with no change in performance or energy consumption. By contrast, by cutting down on the instructions supported by the underlying microarchitecture, the logic complexity of reduced-ISA cores is decreased. This allows such cores to achieve lower power consumption. At the same time, unsupported instructions need to be trapped and software-emulated. A compiler can create binaries that are optimized for execution on the reduced-ISA core. Nevertheless, if removed instructions are critical, as is the case with NEON instructions for floating-point workloads, performance suffers severely. In the remainder of the paper, we define *rISA* as the reduced ISA with all the previously mentioned instructions removed.

For workloads with unsupported instructions that are sensitive to performance, a heterogeneous-ISA system can alleviate the performance degradation. When running an application in which performance-sensitive instructions are not prevalent, it is better to run it on reduced-ISA cores. However, when the application is in a phase where it includes performance-sensitive instructions, the application switches to the full-ISA core where it does not lose any performance. The core switching overhead depends on the granularity and subsequent frequency of switching. We present evaluation results for varying core switching granularities in Section 4.

Dynamically monitoring the instruction streams and switching cores based on performance is the best way to maximize energy savings. In this approach, the process scheduler can dynamically map applications to an appropriate ISA core based on the information from hardware performance counters, such as the number of NEON instructions executed during the current scheduling period. As mentioned above, we assume that unsupported instructions are executed using emulation after undefined instruction exceptions

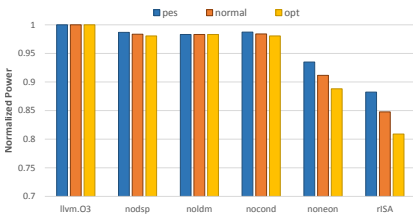


Figure 5: Power estimation of reduced ISA (SPEC CPU2006 INT).

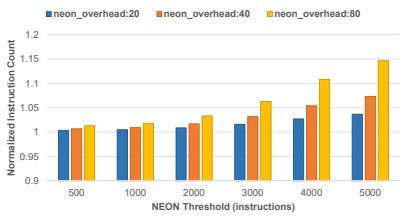


Figure 6: Runlength of SPEC CPU2006 bzip2 with various NEON overheads.

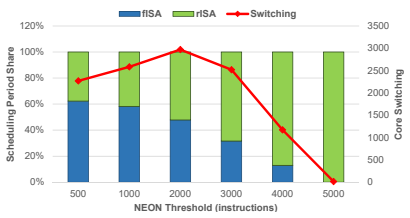


Figure 7: Core residency of SPEC CPU2006 bzip2.

in the reduced-ISA core. This will only be acceptable for infrequent use of the software-emulated instructions. Once the number of software-emulated instructions exceeds a certain threshold within a certain time period, the application should be migrated to a full-ISA core to avoid further performance degradation. On a full-ISA core, on the other hand, the same instructions are also monitored. If the instructions are not used during a certain time period, the application is migrated back to the reduced-ISA core.

For optimal energy and performance balance, we implement a dynamic core scheduling strategy. At the end of each scheduling period, the scheduler determines which core the application should run on based on NEON instruction counts. If there are more NEON instructions than a predetermined threshold, it is better to run the application on a full-ISA core to reduce performance loss. In the opposite case, it is better to run the application on a reduced-ISA core for energy efficiency. Core switching happens when there are consecutive periods where NEON instructions are above or below a pre-defined *neon_threshold*. In this paper, we assume that there are two cores, full- and reduced-ISA, respectively, and no other programs are running. The detailed algorithm can be found in [12].

4. EXPERIMENTS AND RESULTS

Power Estimation: We use McPAT [10] to estimate power consumption. We estimate the logic reduction effects resulting from the removal of the candidate instructions by adjusting numbers under certain assumptions about physical and implementation details. The power estimation details can be found in [12].

As accurately quantifying power consumption is challenging, we give a range of power estimates: *opt* (optimistic) indicates that the effect of the logic reduction is assumed to be relatively large while *pes* (pessimistic) means the opposite (Figure 5). The power reduction for removing most instruction types is about 2% each, except for NEON instructions, which show about 9% improvement. Due to the orthogonality of removed instructions, we assume that the logic related to each instruction is independent, and, thus, those power reductions are additive. Our rISA core shows about 15% power reduction compared to the full-ISA one.

Performance Evaluation: To observe the performance degradation based on the core switching scenario, we evaluate SPEC CPU2006 integer benchmarks using QEMU [4] while collecting a trace of all NEON instructions. We focus on NEON instructions as other removed instructions have little impact on performance as previously shown. We simulate core switching using the previously shown scheduling algorithm that accounts for the number of NEON instruction in each scheduling period. Since switching is not free, each core switching is assumed to incur a fixed instruction overhead. Furthermore, since emulating NEON instructions leads to additional instructions, every time a NEON instruction is executed, an associated instruction penalty is added to the total instruction counts.

Reducing the emulation penalty is crucial for removing instructions. Ideally, it is desirable for an application to entirely run on the reduced-ISA core. However, we find that the penalty to emulate NEON instructions is critical to performance (Figure 6). We estimate the NEON instruction penalty in increments of 20 instructions corresponding to the maximum 20x performance degradation of FP benchmarks. An increase in the penalty per NEON instruction from 20 to 80 directly affects overall performance.

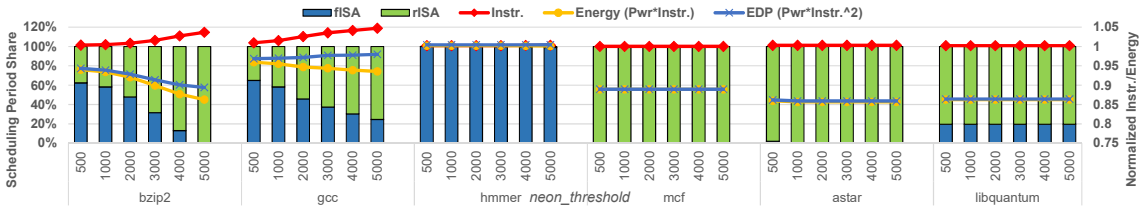
Figure 7 details the fraction of scheduling periods *bzip2* is running on each core and the corresponding number of core switches. The higher the NEON threshold for scheduling, the longer the *bzip2* can run on the reduced-ISA core. This, however, results in more NEON instructions requiring emulation, and, thus, an increase in instruction counts. However, the increase is relatively small. In addition, the number of core switches is low. As such the switching overhead is not crucial for overall performance, and a large switching overhead does not result in significant performance loss. In other words, lowering the penalty allows more NEON instructions to run on the reduced-ISA core.

System Evaluation: Based on the performance and energy evaluation results, we estimate the overall efficiency of our heterogeneous system. The estimated average power of each benchmark is multiplied by the total number of instructions and the square of total instructions to compute energy and energy delay product (EDP), respectively, assuming a constant CPI per benchmark. Furthermore, the computed energy is normalized to the energy and EDP for the full-ISA core in order to observe how much energy savings our system can achieve. These experiments are conducted with a 20-instruction NEON penalty and a core switching overhead of 3000 instructions.

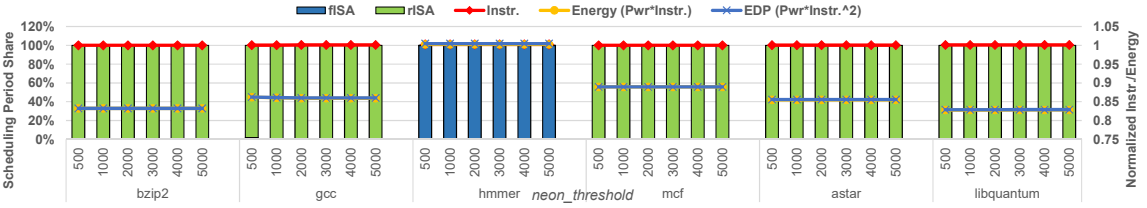
Figure 8(a) shows the estimated performance and energy for SPEC CPU2006 benchmarks. Among the integer benchmarks, we exclude *omnetpp* and *sjeng*; given their high portion of NEON instructions. It is evident that they will run mostly on the full-ISA core. Since dynamic scheduling determines which core to run on next, other applications show diverse scheduling behavior. The *hmmer* benchmark runs on the full-ISA core most of the time due to having more NEON instructions than the threshold for each period, resulting in no performance degradation with no energy benefit.

At the opposite end of the spectrum, *mcf*, *astar*, and *libquantum* run most of the time on the reduced-ISA core with negligible performance degradation, but significant energy benefits. Interestingly, *bzip2* and *gcc* show unbiased behavior. In these workloads, due to the broad range of NEON instructions that vary between 500 and 5000 per period, our scheduling algorithm switches cores frequently depending on the NEON threshold. This aggravates the performance loss with higher NEON penalties and thresholds, thus reducing energy efficiency.

Compiler Optimizations: To optimize energy efficiency, it is better to run applications on the reduced-ISA core as much as possible without losing any performance. This suggests that reducing the amount of NEON instructions is



(a) SPEC CPU2006 integer benchmarks with NEON instructions. Compiler statically removes LDM/STM, DSP-like, and conditional instructions



(b) SPEC CPU2006 integer benchmarks. Frequently used NEON load/store instructions are replaced to integer load/store instructions

Figure 8: Performance evaluation and energy estimation.

beneficial not only to maximize residency of workloads on reduced-ISA cores, but also to reduce performance degradation. Thus, we further limit the generation of NEON instructions by modifying the LLVM back-end. Since we observe that vector load/store instructions are frequently used in instruction optimizations, blocking such optimizations leads to less NEON instructions. We measure the performance of the two binaries on full-ISA cores on the real board and confirm that performance differences of blocking such optimizations are negligible.

Figure 8(b) shows the performance and estimated energy of the modified binaries. The amount of time *bzip2* and *gcc* run on the full-ISA core is reduced to zero. This leads to a total energy efficiency equivalent to running solely on the reduced-ISA core. The *hmmer* benchmark still incorporates quite a few NEON instructions, causing it to stay on the full ISA core. Experimental results show that energy savings of up to 15% are achieved for benchmarks that have negligible NEON instructions. On average, about 12% energy savings are achieved across all evaluated benchmarks. Interestingly, with the manipulation of the compiler, core switches are completely avoided, further increasing energy efficiency. This suggests that if a system has control over the compiler, it is possible to prefer and only use the limited set of instructions available on the energy-efficient reduced-ISA core.

5. RELATED WORKS

Kumar et al. first suggested the possibility of increasing energy efficiency by proposing a single-ISA heterogeneous system [9]. ARM's big.LITTLE architecture [1] subsequently implemented this approach for mobile systems. All of these approaches, however, only focused on the high-performance components in the microarchitecture.

Recently, several notable approaches investigated heterogeneous ISA architectures. Blem et al. revisited the debate about RISC versus CISC, comparing x86 and ARM ISAs [6]. Their research quantifies each ISA and argues that there is not much difference between x86 and ARM ISAs. DeVuyst et al. [8] and Venkat et al. [7] proposed a way of harnessing heterogeneous ISAs. However, their research only focused on the diversities of each ISA and argued the benefits when finding the right ISA depending on the workloads. Several prior works [11, 5] investigated OS and software support for task migration between cores with overlapping ISAs. However, their work does not discuss detailed trade-offs in designing actual architectures for such heterogeneous-ISA systems.

6. SUMMARY AND CONCLUSIONS

A reduced ISA core opens up a chance to run high-performance tasks with better energy efficiency. In this paper, we explore and demonstrate the potential with a case study of a heterogeneous system that includes both full- and reduced-ISA cores. Results show that certain complex instructions can be removed with little performance overhead. We argue that in a heterogeneous system that effectively migrates applications to match ISA requirements, significant benefits in energy efficiency can be obtained. Providing heterogeneity in functionality rather than performance can improve energy efficiency with virtually no performance degradation. Results show that our proposed system can improve energy by up to 15% and by 12% on average, all with little to no performance overhead.

7. ACKNOWLEDGEMENTS

This research was supported in part by SRC grant 012-HJ-2317. We would like to thank the anonymous reviewers for their many insightful comments and suggestions.

8. REFERENCES

[1] ARM big.LITTLE Processing. http://www.arm.com/products/processors/technologies/biglittleprocessing.php.
[2] ARMv8 Instruction Set Overview. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.genc010197a/index.html.
[3] Arndale Board. http://www.arndaleboard.org.
[4] Qemu. http://www.qemu.org.
[5] A. Aminot et al. FPU Speedup Estimation for Task Placement Optimization on Asymmetric Multicore Designs. In *MCSoC*, 2015.
[6] E. Blem et al. Power Strugles: Revisiting the RISC vs. CISC Debate on Contemporary ARM and x86 Architectures. In *HPCA*, 2013.
[7] A. V. et al. Harnessing ISA Diversity: Design of a Heterogeneous-ISA Chip Multiprocessor. In *ISCA*, 2014.
[8] M. D. et al. Execution Migration in a Heterogeneous-ISA Chip Multiprocessor. In *ASPLOS*, 2012.
[9] R. K. et al. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In *Micro*, 2003.
[10] S. Li et al. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Micro*, 2009.
[11] T. Li et al. Operating System Support for Overlapping-ISA Heterogeneous Multi-core Architectures. In *HPCA*, 2010.
[12] W. Lee et al. Exploring Opportunities for Heterogeneous-ISA Core Architectures in High-Performance Mobile SoCs. Technical Report UT-CERC-17-01, The University of Texas At Austin, 2017.