A Minimum Out-of-Order Core

Mehdi Alipour, Stefanos Kaxiras, and David Black-Schaffer
Uppsala University, Sweden
first.last@it.uu.se

Rakesh Kumar
Norwegian University of Science and Technology
rakesh.kumar@ntnu.no

I. Introduction

Mobile computing has grown drastically over the past decade. There are already more than 2 billion mobile devices in circulation worldwide [1] and the number of mobile subscribers is expected to cross the 6 billion mark in coming years [2]. A high-quality user experience and longer battery life are the key challenges in building competent mobile devices including smartphone, tablets, and other handheld devices. To meet the heavy processing demands of high-quality user experience, these devices generally deploy out-of-order (OoO) processing cores. However, the high energy requirements of OoO execution directly limit the battery life. This work investigates a *minimum out-of-order* core that approaches the performance of full OoO execution, however, only at a fraction of its energy cost.

In pursuit of performance, an OoO core exploits an application's inherent instruction level parallelism (ILP) by executing independent instructions out of the program order. However, it employs a number of large, complex, and power hungry hardware structures such as instruction queue (IQ), physical register files (PRF), and load/store queues (LSQ) etc. to expose the ILP. Instruction queue, which is typically implemented as a content addressable memory (CAM), is arguably the most complex and power hungry structure of an OoO core. It comprises a complex web of wires and combinational circuits to broadcast the results (or physical register ids) to make waiting instructions ready for execution and select among ready instructions based on set priorities. Furthermore, to a first order, the complexity and power consumption of IQ increases quadratically with issue width and linearly with the number of entries.

In this work, we study the behaviour of SPECcpu2006 workloads and discover that not all instructions need to pass through these complex OoO structures to expose ILP. In particular, as shown in Figure 1, we found that in a typcial mobile core, with 128 ROB entries and 4-wide issue, about 28% of the instructions are ready for execution when they are dispatched to the IQ, we call these *Ready at Dispatch (R@D)* instructions. Also, a further 15% of instructions become ready within 2 cycles after dispatch, called *Almost Ready at Dispatch AR@D* instructions. Based on these results, we make a **critical observation** that a complex IQ brings little or no ILP benefits for these instructions as they can execute in program order with minimal stalls. Our **second key observation** is that the values/results passed among R@D and/or AR@D instructions are seldom read by any other instruction. Therefore, the results can be directly forwarded to such consumer instructions via bypass network, without being written to the PRF, hence saving energy and reducing PRF port contention.

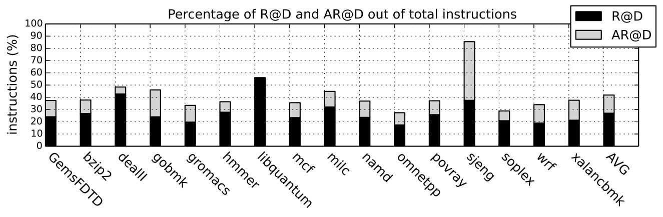

Fig. 1. Percentage of instructions that are either ready for execution at dispatch (R@D) or become ready within few cycles after dispatch (AR@D).

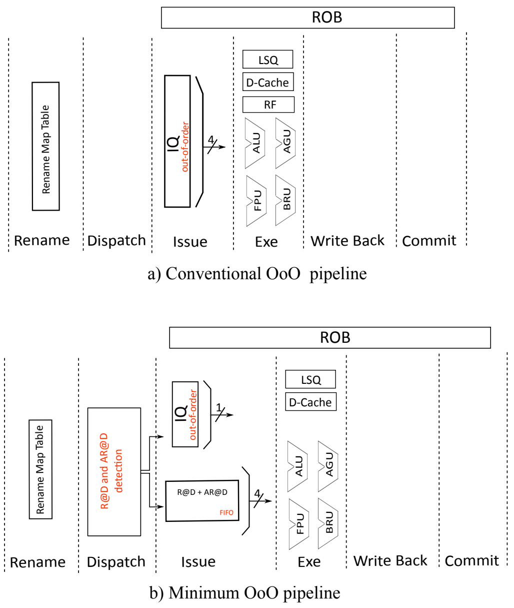

Fig. 2. The pipeline view of, a) Conventional OoO Core, and b) Minimum OoO Core

II. The Minimum Out-of-Order Core

To exploit the observations made in Section I, we introduce a minimum out-of-order core, as shown in Figure 2, that reduces the energy consumption significantly by executing about 43% of instructions in program order. The proposed design introduces a first-in first-out instruction queue (FIFO-IQ) logically in parallel with the regular IQ. The dispatch stage identifies R@D and AR@D instructions and dispatches them to the FIFO-IQ for in-order execution. As these instructions are mostly ready for execution, the FIFO-IQ stalls only very rarely. The rest of the instructions are dispatched to the regular IQ similar to a full OoO core. Furthermore, as the regular IQ now needs to handle significantly fewer instructions, its complexity and power consumption can be lowered by reducing both the number of entries and its issue width. Nonetheless, the overall issue width stays the same as FIFO-IQ also supplies instructions. The proposed core also tracks the consumers of R@D and AR@D instructions and their distance from the producers. If the results can be forwarded to the all consumers from the bypass network, the destination register of the producers is *cleared* and the results are not written to PRF. Also, a producer instruction is committed only when its consumer is also ready to commit.

By reducing IQ complexity and power consumption along with fewer PRF writes, the proposed core design reduce the overall energy consumption significantly while still delivering similar performance as a full OoO core.

III. Related Work

A previously proposed technique, called FXA [3], also aims to reduce energy consumption of an OoO core by executing some of the instructions in program order. However, FXA uses a brute force approach of first trying to execute all instructions in program order and then moving only those instructions for OoO execution that could not be executed in-order due to operand unavailability. Our design splits the instruction stream upfront for in-order and OoO execution. FXA *inserts* a 4-stage in-order pipeline within a regular OoO pipeline to support in-order execution and requires functional unit replication. Our design, in contrast, shares the same functional units among in-order and OoO execution. To reduce the area overhead, FXA replicates only integer functional units; therefore, the floating point operations are always executed by OoO engine. Our design, however, has no such limitation. FXA potentially increases PRF reads as operands first need to be read for in-order execution and then again for OoO execution if in-order execution of an instruction fails. A potential solution is to pass the operands from in-order to OoO execution engine, however it requires, 64-bit buses to run between these two engines and saving the ready operands in the IQ; all of which leads to more energy overhead.

References

[1] Developers, G. Google i/o keynote. https://www.youtube.com/watch?v=Y2VF8tmLFHw, May 2017.

[2] Heuveldop, N. Erricson mobility report. https://www.ericsson.com/assets/local/mobility-report/documents/2017/ericsson-mobility-report-june-2017.pdf, 2017.

[3] Shioya, R., Goshima, M., and Ando, H. A front-end execution architecture for high energy efficiency. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture* (2014), pp. 419–431.