

SMIX: Schedulable Instruction Set Architecture Extension Interface for Multi-Operand Operators

Shufan He, Hanmo Wang, Kefa Chen, Xuyin Chen, Xianhua Liu and Chun Yang

School of Computer Science

Peking University

Beijing, China

{sofan_he, wanghanmo, 2301213205, chinlisten}@stu.pku.edu.cn, {liuxianhua, yangchun}@pku.edu.cn

Abstract—Integrating domain-specific operators into processor cores is essential for performance scaling. However, multi-operand operators often face a semantic gap with conventional ISAs, which are limited in operand capacity and scheduling flexibility. This paper presents SMIX, a schedulable instruction set extension interface for multi-operand operators. SMIX decouples execution into three stages: out-of-order input filling, computation, and out-of-order result picking. By employing explicit encoding and counter-based dependency management, SMIX enables both efficient static scheduling by compilers and dynamic out-of-order execution in hardware. Experimental results demonstrate the high schedulability of SMIX, where static scheduling provides an average 12% performance gain on the Rocket core and dynamic out-of-order scheduling contributes an additional 9.2% speedup on the BOOM core, all while maintaining minimal hardware overhead.

Index Terms—Instruction Set Extension, Domain-specific Architecture, Software-Hardware Co-design.

I. INTRODUCTION

The diminishing returns of traditional technology scaling have shifted the computing paradigm toward Domain-Specific Architectures (DSAs). By tailoring hardware to specific workloads, DSAs achieve orders-of-magnitude gains in efficiency. Central to this specialization is the integration of custom operators directly into processor cores. In particular, multi-operand operators (e.g., MIMO subgraphs) offer massive potential for enhancing throughput and reducing energy consumption compared to standard scalar operations [1].

However, a significant semantic gap exists between these complex operators and conventional Instruction Set Architectures (ISAs), which are typically restricted to a "two-input, one-output" format [2]. While frameworks like RoCC [3] and SCAIE-V [4] provide extensible interfaces for custom instructions, they are not inherently optimized for the high operand count of modern domain-specific kernels. Recent work [5]–[8], attempts to bridge this gap by implicitly binding subsequent instructions to provide operands. Unfortunately, such designs often introduce strict sequential dependencies. In modern out-of-order (OoO) superscalar processors, these constraints severely limit the hardware's ability to rename

and schedule instructions, thereby throttling the overall performance gains of the extension.

To address these challenges, this paper proposes SMIX, a schedulable ISA extension interface specifically designed for multi-operand operators. SMIX decouples the execution process into three distinct, explicitly encoded stages: out-of-order input filling, computation, and out-of-order output picking. By employing a multi-group architecture, SMIX allows the compiler and hardware to exploit parallelism across multiple operator instances. SMIX introduces a group-independent pseudo-instruction layer to facilitate static compiler scheduling and a counter-based dependency tracking mechanism to enable efficient dynamic issue in OoO cores.

II. SMIX INSTRUCTION SEMANTICS

SMIX bridges the gap between fixed-length ISAs and multi-operand operators using decoupled semantics and a **Grouped Custom Register File (CRF)**. The CRF organizes input (I_{reg}) and output (O_{reg}) registers into N independent groups (G_{gid}), enabling inter-operator parallelism.

The interface defines four primary two-input, one-output instructions:

- **fill** $rs1, rs2, groupid, indexin$: Moves $GPR_{rs1,rs2}$ to input registers $I_{indexin \times 2, +1}$ in $G_{groupid}$.
- **pick** $rd, groupid, indexout$: Moves $O_{indexout}$ from $G_{groupid}$ to GPR_{rd} .
- **fillpick** $rd, rs1, rs2, groupid, indexin, indexout$: Atomically executes **fill** and **pick** to optimize register port usage.
- **exec** $rd, rs1, rs2, groupid, indexin, indexout$: Performs a **fill**, triggers the functional unit for $G_{groupid}$, and then executes a **pick**.

By explicitly encoding *groupid* and operand indices, SMIX creates a clean dependency graph for both compiler and hardware scheduling.

III. STATIC SCHEDULING FOR SMIX

SMIX preserves ILP by decoupling logical operations from hardware binding through a two-phase flow (Fig. 1):

- 1) **Group-agnostic phase**: High-level pseudo-instructions allow the scheduler to optimize order based solely on true data dependencies.

This work was supported by the National Key R&D Program of China (Grant no. 2022YFB4500500). Chun Yang and Xianhua Liu are the Corresponding authors of this paper.

2) **Group-aware phase:** A greedy allocation pass maps pseudo-instructions to physical groups, lowering them into concrete `fill`, `exec`, and `fillpick` sequences.

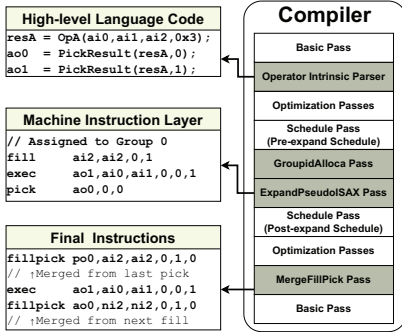


Fig. 1. The SMIX Compiler Infrastructure

This approach prevents premature hardware resource constraints from limiting ILP during optimization.

IV. DYNAMIC SCHEDULING FOR SMIX

To support out-of-order (OoO) execution, SMIX employs a **counter-based issue logic** (Fig. 2) to track dependencies between variable numbers of `fill` and `exec` instructions.

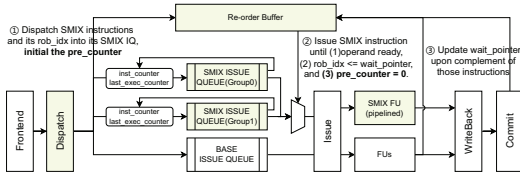


Fig. 2. Processor architecture supporting out-of-order execution of both SMIX and other instructions

Two global counters, `inst_counter` and `last_exec_counter`, track dependencies. At dispatch, `exec` instructions inherit the `inst_counter` into their local `pre_counter`, while other SMIX instructions take `last_exec_counter + 1`. Instructions issue only when their `pre_counter` hits zero and GPR operands are ready. This mechanism enables inter-group OoO execution and simplifies branch recovery; resetting counters to the current `inst_counter` eliminates the need for complex state snapshots.

V. EVALUATION

A. Experimental Methodology

We evaluate SMIX on three RISC-V cores: the 4-stage **VexRiscv** (RV32I), the in-order **Rocket** (RV64GC), and the out-of-order **BOOM** (RV64GC). Five kernels spanning linear algebra (GEMV), deep learning (Conv1d), imaging (YUV2RGB), cryptography (Rijndael), and bioinformatics (Smith-Waterman) serve as benchmarks. Each core integrates SMIX functional units (FUs) with two register groups and pipelined execution logic.

B. Low Implementation Overhead

SMIX achieves high efficiency by reusing standard 2-in/1-out pipeline stages and avoiding complex state snapshots. Integrating the out-of-order SMIX interface into BOOM requires only **218 lines** of Chisel code, while the in-order version on Rocket needs only **112 lines**. Hardware-wise, SMIX adds less than **1%** LUTs/FFs to Rocket and approximately **2.5%** to BOOM on FPGA.

C. Static and Dynamic Scheduling Performance

SMIX achieves significant performance gains by exploiting scheduling flexibility.

Static Scheduling: On the in-order Rocket core under `-O3` optimization, SMIX’s explicit encoding allows the compiler to optimize group allocation and overlap data movement. This leads to an average **12%** performance improvement compared to configurations without scheduling.

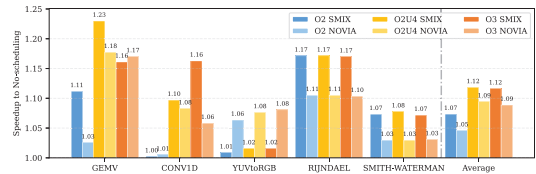


Fig. 3. Speedup of SMIX over No-scheduling configuration on Rocket under varying optimization levels

Dynamic Scheduling: On the out-of-order BOOM core, SMIX’s counter-based issue logic enables instructions to bypass sequential constraints across different groups. This dynamic mechanism contributes an additional **9.2%** speedup over in-order execution.

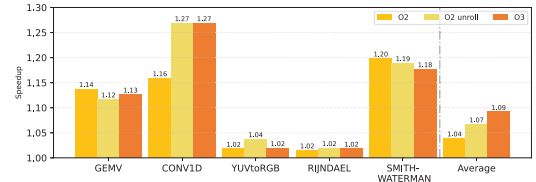


Fig. 4. Speedup of Out-of-Order SMIX execution over In-Order execution on BOOM

These results confirm that SMIX’s design effectively bridges the gap between complex operator semantics and hardware execution. By enabling both the compiler and the hardware to extract more parallelism, SMIX ensures that the high potential of multi-operand operators is fully realized through superior instruction-level scheduling.

VI. CONCLUSION

SMIX provides a schedulable interface for multi-operand operators by decoupling execution into explicitly encoded stages. Evaluations demonstrate significant performance gains through both static and dynamic scheduling. With minimal hardware overhead, SMIX effectively bridges the semantic gap for domain-specific acceleration in modern processors.

REFERENCES

- [1] N. Pothineni, A. Kumar, and K. Paul, "Application specific datapath extension with distributed i/o functional units," in *20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07)*, 2007, pp. 551–558.
- [2] F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, J. C. Hoe, and J. M. F. Moura, "Spiral: Extreme performance portability," *Proceedings of the IEEE*, vol. 106, no. 11, pp. 1935–1968, 2018.
- [3] K. Asanović, R. Avižienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, P. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," EECS Department, University of California, Berkeley, Technical Report UCB/EECS-2016-17, April 2016. [Online]. Available: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html>
- [4] M. Damian, J. Oppermann, C. Spang, and A. Koch, "Scaic-v: an open-source scalable interface for isa extensions for risc-v processors," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 169–174. [Online]. Available: <https://doi.org/10.1145/3489517.3530432>
- [5] D. Trilla, J.-D. Wellman, A. Buyuktosunoglu, and P. Bose, "Novia: A framework for discovering non-conventional inline accelerators," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 507–521. [Online]. Available: <https://doi.org/10.1145/3466752.3480094>
- [6] R. Jayaseelan, H. Liu, and T. Mitra, "Exploiting forwarding to improve data bandwidth of instruction-set extensions," in *Proceedings of the 43rd annual Design Automation Conference*, ser. DAC '06. Association for Computing Machinery, pp. 43–48. [Online]. Available: <https://dl.acm.org/doi/10.1145/1146909.1146924>
- [7] A. K. Verma, P. Brisk, and P. Ienne, "Fast, quasi-optimal, and pipelined instruction-set extensions," in *2008 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '08. IEEE, pp. 334–339. [Online]. Available: <http://ieeexplore.ieee.org/document/4483970/>
- [8] J. Cong, Y. Fan, G. Han, A. Jagannathan, G. Reinman, and Z. Zhang, "Instruction set extension with shadow registers for configurable processors," in *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, ser. FPGA '05. Association for Computing Machinery, pp. 99–106. [Online]. Available: <https://dl.acm.org/doi/10.1145/1046192.1046206>