Late Breaking Results: A RISC-V ISA Extension for Chaining in Scalar Processors

Luca Colagrande
Integrated Systems Laboratory (IIS)
ETH Zurich
Zurich, Switzerland
colluca@iis.ee.ethz.ch

Jayanth Jonnalagadda
D-ITET
ETH Zurich
Zurich, Switzerland
jjonnalagadd@student.ethz.ch

Luca Benini
Integrated Systems Laboratory (IIS)
ETH Zurich
Zurich, Switzerland
lbenini@iis.ee.ethz.ch

***Abstract*—Modern general-purpose accelerators integrate a large number of programmable area- and energy-efficient processing elements (PEs), to deliver high performance while meeting stringent power delivery and thermal dissipation constraints. In this context, PEs are often implemented by scalar in-order cores, which are highly sensitive to pipeline stalls. Traditional software techniques, such as loop unrolling, mitigate the issue at the cost of increased register pressure, limiting flexibility. We propose scalar chaining, a novel hardware-software solution, to address this issue without incurring the drawbacks of traditional software-only techniques. We demonstrate our solution on register-limited stencil codes, achieving >93% FPU utilizations and a 4% speedup and 10% higher energy efficiency, on average, over highly-optimized baselines. Our implementation is fully open source and performance experiments are reproducible using free software.** [1]

***Index Terms*—RISC-V, in-order, hazard, energy efficiency**

I. Introduction

As power delivery and thermal dissipation challenge the development of ever more powerful high-performance computers [1], designing energy-efficient architectures has become of paramount importance [2]. To this end, architectures featuring many energy-efficient in-order cores with shallow pipelines are favoured over multicores hosting a few power- and resource-hungry deeply-pipelined out-of-order processors [3].

While in-order cores are more area- and energy-efficient, they are also more sensitive to pipeline stalls as a consequence of data hazards, leading to reduced performance. Consider the RISC-V code in Fig. 1a, implementing the vector operation $\mathbf{a} = b*(\mathbf{c}+\mathbf{d})$ on the state-of-the-art scalar in-order Snitch processor [4]. To improve FPU utilization, we take advantage of the Stream Semantic Register (SSR) Instruction Set Architecture (ISA) extension of Snitch: registers `ft0` and `ft1` implicitly read the next value from the $\mathbf{c}$ and $\mathbf{d}$ arrays, respectively, and `ft2` implicitly writes to the next element in the $\mathbf{a}$ array. In an in-order pipelined processor, the Read After Write (RAW) dependency between the `fadd` and `fmul` instructions causes the latter to stall until the result from the prior is available, wasting useful cycles, three in the case of Snitch, equal to the number of pipeline stages in its Floating Point Unit (FPU).

Loop unrolling and software pipelining techniques can be adopted to eliminate wasted cycles [5], as illustrated in Fig. 1b. The downside of this approach is that it increases register pressure, limiting the effectiveness of this technique to small codes, to avoid register spills [6]. Vector and out-of-order processors with renaming support mitigate this issue by implementing large physical Register Files (RFs), at the cost of much increased area and energy consumption.

```
fadd.d ft3, ft0, ft1
fmul.d ft2, ft3, %[b]
addi %[i], %[i], 1
bneq %[i], %[len], -12
```
(a)

```
fadd.d ft3, ft0, ft1
fadd.d ft4, ft0, ft1
fadd.d ft5, ft0, ft1
fadd.d ft6, ft0, ft1
fmul.d ft2, ft3, %[b]
fmul.d ft2, ft4, %[b]
fmul.d ft2, ft5, %[b]
fmul.d ft2, ft6, %[b]
addi %[i], %[i], 4
bneq %[i], %[len], -36
```
(b)

```
1  li   %[mask], 8
2  csrs 0x7C3, %[mask]
3  fadd.d ft3, ft0, ft1
4  fadd.d ft3, ft0, ft1
5  fadd.d ft3, ft0, ft1
6  fadd.d ft3, ft0, ft1
7  fmul.d ft2, ft3, %[b]
8
9  fmul.d ft2, ft3, %[b]
10 fmul.d ft2, ft3, %[b]
11 fmul.d ft2, ft3, %[b]
12 addi %[i], %[i], 4
13 bneq %[i], %[len], -36
14 csrs 0x7C3, x0
```
(c)

Fig. 1: (a) Baseline and (b) unrolling-optimized code variants. (c) A sample execution trace of the chaining-optimized code. The colored issue slots are marked for reference in Fig. 2.

In this work, we propose an alternative solution that enables the advantages of loop unrolling without increasing register pressure. By utilizing the pipeline registers within the processor's functional units to store intermediate results, our approach is well-suited for integration into highly area- and energy-efficient cores. While we demonstrate our implementation on the Snitch processor with its ISA extensions, the solution is not restricted to this particular design.

II. Implementation

The code in Fig. 1b uses three additional architectural registers (`ft4`-`ft6`) to store the intermediate results of the `fadd` instructions, until they are consumed by the respective `fmul` instructions. Our first insight is that the FPU already provides enough storage for these intermediate results in its three pipeline stages, which led to unrolling the code by four in the first place. By properly balancing the production and consumption rate of intermediate results by the `fadd` and `fmul` instructions, we can avoid allocating additional physical registers. This dataflow, which conceptually involves *chaining* the `fadd` and `fmul` instructions, is illustrated in Fig. 2.

As pipeline registers are not directly addressable, we need to extend the ISA to implement the desired dataflow. We

[1] https://github.com/colluca/snitch_cluster/tree/chaining

978-3-9826741-0-0/DATE25/© 2025 EDAA

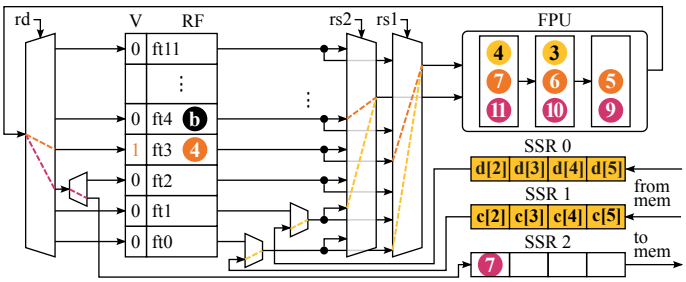

Fig. 2: Block diagram of Snitch's FP subsystem, illustrating the dataflow associated to the trace in Fig. 1c. Elements in different colors represent snapshots at different moments in time, particularly at the respectively-colored issue slots in Fig. 1c. Numbered tokens represent the outputs of the instructions at the respectively-numbered issue slots in Fig. 1c.

integrate a custom Control and Status Register (CSR) (at address `0x7c3`) hosting a 32-bit mask (one bit per architectural register) to dynamically enable and disable chaining on the selected architectural Floating-Point (FP) registers. Enabling chaining alters instruction semantics: writeback to a chaining-enabled register is no longer enforced to be carried out atomically with operation execution. There is thus no notion of Write After Write (WAW) dependency between successive `fadd` instructions in the code in Fig. 1c. Instead, chaining-enabled registers are assigned *FIFO semantics*, i.e. writes and reads to the register respectively imply push and pop operations from a logical First-In First-Out Queue (FIFO), which is physically implemented by concatenating the chaining-enabled architectural register (`ft3` in our example) and the functional unit's pipeline registers. Conversely, the traditional unrolling approach in Fig. 1b implements this logical FIFO in the RF, at the cost of precious architectural registers. Thus, chaining benefits are increased for functional units with deeper pipelines.

Finally, we add a valid bit per architectural register to implement the backpressure mechanism avoiding that elements in the logical FIFO are overwritten before they are consumed in the occurrence of a stall, as illustrated by the orange issue slot in Fig. 1c.

III. Results

We implement a Snitch cluster [4] with one compute core in GlobalFoundries' 12LP+ FinFET technology using Fusion Compiler 2023.12, with a target clock frequency of 1 GHz. Our extensions introduce negligible overheads, <2% cell area increase and maximum frequency degradation, comparable to synthesis process variability margins.

All experiments are conducted in cycle-accurate RTL simulations using QuestaSim 2023.4. Switching activities are extracted from post-layout simulations, and used for power estimation in PrimeTime 2022.03, assuming typical operating conditions of 25 °C and 0.8 V supply voltage.

We demonstrate the benefits of chaining on a set of stencil codes implemented in [7]. We observe that the `box3d1r` and `j3d27pt` stencils are register-limited, and that, by applying chaining, enough registers can be freed to fully store the stencil coefficients in the RF.

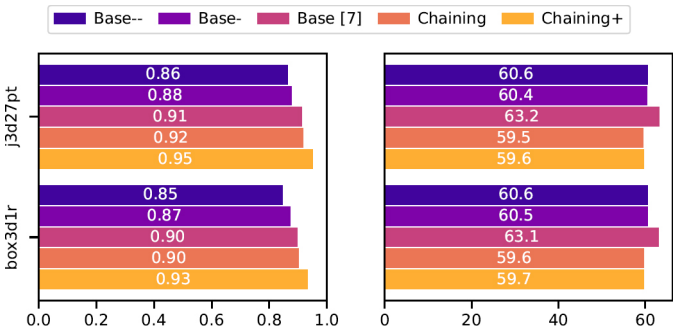

Fig. 3: FPU utilization (left) and power consumption [mW] (right) for all code variants.

Fig. 3 shows the results of our evaluation. By moving the stencil coefficients to the RF, we can get rid of repeated coefficient loads from memory. In the implementation from [7], these are mapped to an SSR; thus, we do not directly benefit from eliminating issue cycles spent on explicit load instructions. However, we observe a 7% geomean improvement in energy efficiency, as repeated accesses to the L1 memory are avoided. For comparison, the `Base--` implementation does not use an SSR to load stencil coefficients.

Additionally, by freeing the SSR previously employed in loading coefficients, chaining enables mapping writeback of stencil computation results to the freed SSR, eliminating explicit store instructions. We implement this optimization in `Chaining+` and `Base-`, which uses the SSR freed in `Base--`. Overall, chaining results in a 4% geomean speedup and a 10% geomean energy efficiency improvement over the highly-optimized baselines in [7], and 8% and 9% gains respectively over the direct comparison point `Base-`, reaching near-ideal FPU utilizations above 93%.

IV. Conclusion

We presented a novel hardware and software solution to hide functional unit latencies in scalar in-order processors, without incurring increased register pressure, as with traditional software-only techniques. We demonstrated our solution on a set of highly-optimized stencil codes, achieving a 4% speedup and 10% higher energy efficiency, on average, and >93% FPU utilizations. Our solution is lightweight and thus suited for integration into highly area- and energy-efficient cores.

References

[1] O. Villa *et al.*, "Scaling the power wall: A path to exascale," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[2] H. Esmaeilzadeh *et al.*, "Dark silicon and the end of multicore scaling," in *2011 38th Annual International Symposium on Computer Architecture*.

[3] S. Huang *et al.*, "On the energy efficiency of graphics processing units for scientific computing," in *2009 IEEE International Symposium on Parallel & Distributed Processing*.

[4] F. Zaruba *et al.*, "Snitch: A tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads," *IEEE Transactions on Computers*, vol. 70, no. 11, 2021.

[5] J. M. Cardoso *et al.*, "Chapter 5 - source code transformations and optimizations," in *Embedded Computing for High Performance*, 2017.

[6] J. Makino, *Principles of High-Performance Processor Design*, 2021.

[7] P. Scheffler *et al.*, "Saris: Accelerating stencil computations on energy-efficient risc-v compute clusters with indirect stream registers," in *DAC '24: Proceedings of the 61st ACM/IEEE Design Automation Conference*.