BZL
BARCELONA
ZETTASCALE LAB

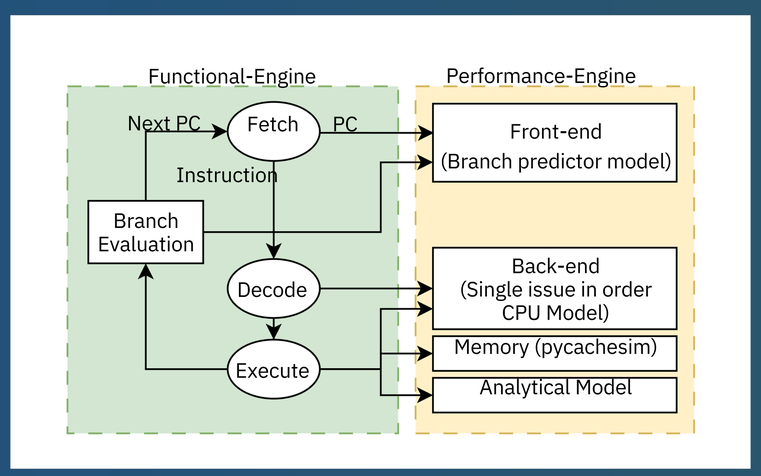
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Driving RISC-V® Innovation for HPC

RISCV-PySim: A Modular and Flexible Python-Based RISC-V Simulator

Carlos Rojas Morales[1,2], Víctor Asanza[1], Julian Pavon[1,2], Ivan Vargas Valdivieso[1,2],
Erick Brandon Cureño Contreras[3], and Adrian Cristal[1]
[1]Barcelona Supercomputing Center (BSC)
[2]Universitat Politècnica de Catalunya (UPC)
[3]Centro de Investigación en Computación, Instituto Politécnico Nacional (CIC-IPN)
E-mail: carlos.rojas@bsc.es

1. Introduction: Domain-Specific Hardware Accelerators (DSHAs) offer significant gains in performance and energy efficiency over general-purpose Central Processing Units (CPUs). This work presents RISCV-PySim, a Python-based RISC-V simulator designed to integrate and evaluate custom instructions, aiding DSHA development. RISCV-PySim simplifies the verification of new instructions, reducing the overhead typically associated with traditional simulators. Additionaly, Its modular design enables easy integration with different performance models, making it highly adaptable to specific design needs.



Algorithm 1: Systolic Array 8x8

```
for i ← A.N; i+ = 8 do
    for k ← B.M; k+ = 8 do
        bufferA ← block(A_{i→i+7:k→k+7})
        for m ← B.N; m+ = 8 do
            bufferB ← block(B_{k→k+7:m→m+7})
            bufferC = multiply()
            block(C_{i→i+7:m→m+7}) += bufferC)
        end
    end
end
```

2. Design flow: The simulator is based on a Finite State Machine (FSM) with two decoupled modules.

Functional Engine, executes instructions atomically to generate execution traces. It includes:

- Fetch: Retrieves instructions and controls flow via the Program Counter (PC).
- Decode: Decodes instructions to extract operands and operations.
- Branch Evaluation (BE): Determines the next PC by evaluating branches.
- Execute Stage: Performs the operations using functions from the Decode stage.

Performance Engine, estimates system performance by configuring architectural parameters. It includes:

- Front-end: Models branch prediction.
- Back-end: Simulates a single-issue, in-order CPU.
- Memory: Uses pycachesim to model memory access latency.
- Analytical Model: Calculates performance based on latencies, mispredictions, and Level 1 Data Cache (L1-D) accesses.

3. Evaluations: To evaluate RISCV-PySim, we used a General Matrix Multiply (GEMM) benchmark using scalar, vector and a DSHA based on a systolic array architecture. A simplified in-order CPU model based on the 64-bit Sargantana processor (7-stage pipeline, RV64G and RVV 0.7.1 support) was used and compared against the gem5 simulator.

- Three custom instructions were added to work with a systolic array accelerator (Algorithm 1), significantly boosting performance: The vector implementation achieved a 1.68X speed-up over the scalar version, while the systolic array with custom instructions achieved a 9.63X speed-up.
- Integrating these custom extensions into RISCV-PySim only required 67 lines of code, compared to modifying 12 files and about 300 lines in gem5.

Acknowledgment
This publication is promoted by the Barcelona Zettascale Laboratory, backed by the Ministry for Digital Transformation and of Public Services, within the framework of the Recovery, Transformation, and Resilience Plan – funded by the European Union – NextGenerationEU.

Get more information at bzl.es

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA
MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y LA FUNCIÓN PÚBLICA
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia

Este proyecto está impulsado por el Ministerio para la Transformación Digital y de la Función Pública, en el marco del Fondo de Resiliencia y Recuperación -y la Unión Europea- NextGenerationEU.