Hardware/Software Co-design of Thread-enabled RISC-V-based Multicores

* binh@vlsilab.ee.uec.ac.jp

Binh Kieu-Do-Nguyen*, Khai-Duy Nguyen, Tuan-Kiet Dang
Cong-Kha Pham, and Trong-Thuc Hoang

国立大学法人 電気通信大学 The University of Electro-Communications

Pham Laboratory
Integrated circuit design laboratory

Introduction

Since its inception in 2011, RISC-V has experienced tremendous growth, resulting in numerous free and open-source processors based on its instruction set. RISC-V is increasingly used in various devices like IoT, wearables, and AI, creating a demand for multicore platforms built with RISC-V cores. While combining different RISC-V cores for specific applications seems ideal, it's challenging because most open cores lack features needed for efficient multicore design. This research proposes a hardware/software co-design solution to overcome these bottlenecks, significantly improving performance of multicore system.

Hardware design and Evaluation

Multiple level cache system:

- L2 Cache → External data cache.
- Shared Memory → Internal shared data/Thread info.
- Local Memory (LMEM) → Tightly coupled local memory.

Local switch: Allows LMEM's access without delay.

- From RISC-V
- From tightly coupled accelerator (Accel.).
- From System Bus (SBUS).

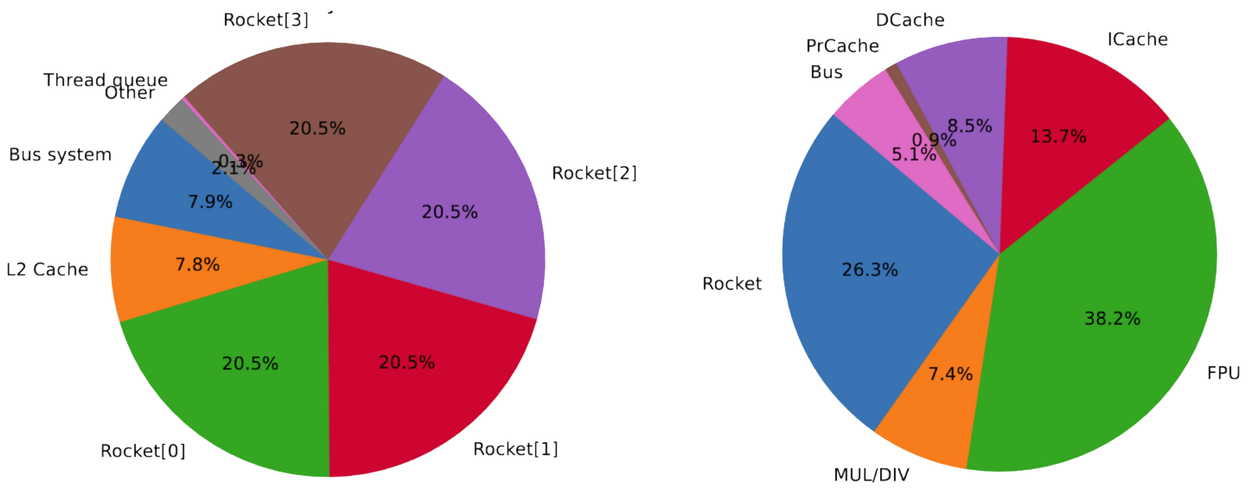

Figure 2 Resource overhead (LUTs)

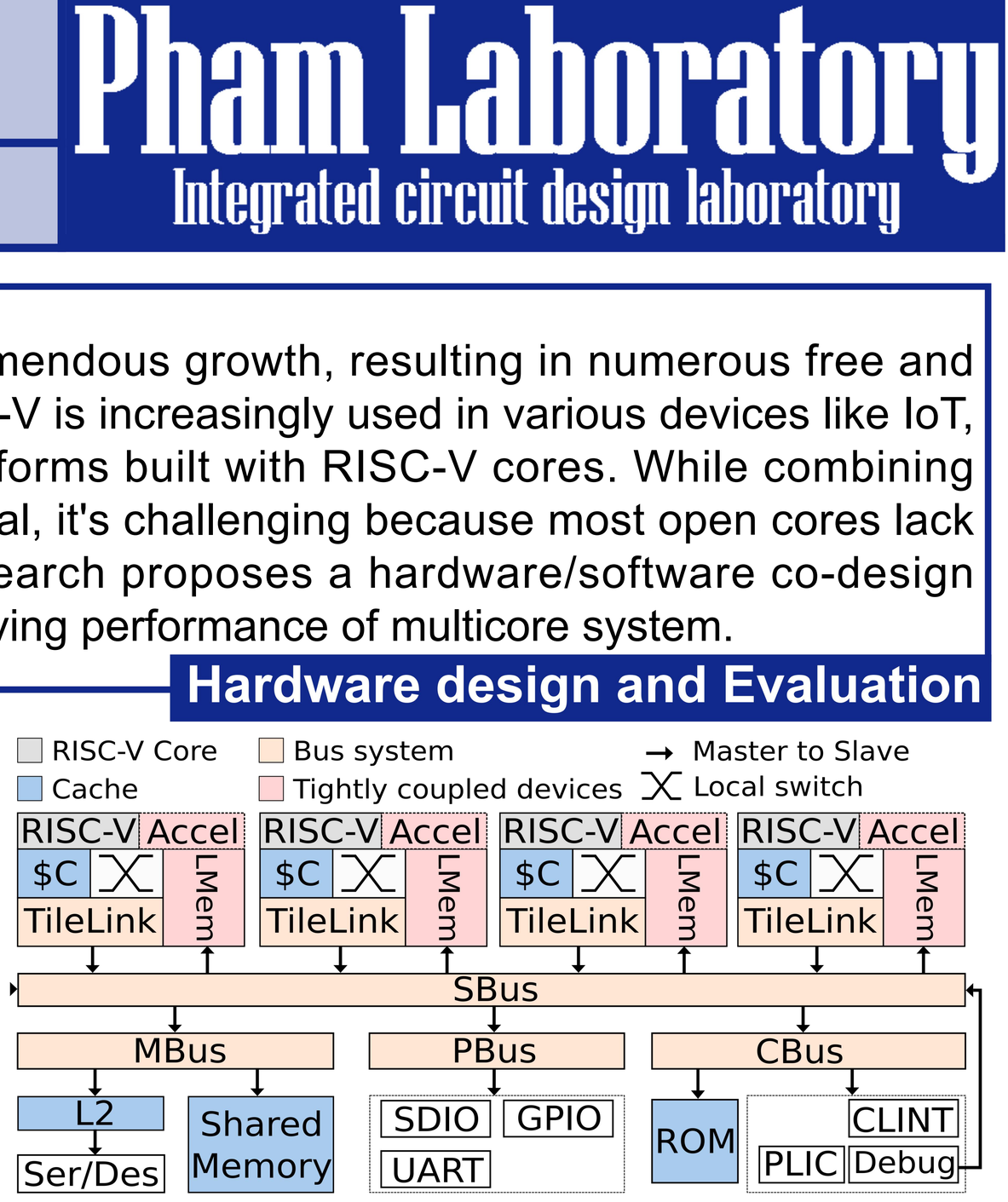

Figure 1 Proposed Thread-enabled Multicore RISC-V System

Resource/Overhead	LUTs	Regs
System	55,642	27,596
|– RISC-V core*	11,291	5,568
|–– Local switch	104	118
|– Shared memory	143	186

* Resource consumption for only one Rocket core (RV32-IMAFC)

Software design and Evaluation

Algorithm 1 Strassen's algorithm (Divide-and-Conquer matrix multiplication)

Input: matrices A and B of size p.
Output: C is the result with the appropriate size.

1: Partition matrix A of size p into:

$$A = \begin{Bmatrix} a_{0,0} & \cdots & a_{0,p/2-1} & \cdots & a_{0,p-1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{p/2-1,0} & \cdots & a_{p/2-1,p/2-1} & \cdots & a_{p/2-1,p-1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{p-1,0} & \cdots & a_{p-1,p/2-1} & \cdots & a_{p-1,p-1} \end{Bmatrix} = \begin{Bmatrix} A_0 & A_1 \\ A_2 & A_3 \end{Bmatrix}$$

2: Partition matrix B of size p into:

$$B = \begin{Bmatrix} b_{0,0} & \cdots & b_{0,p/2-1} & \cdots & b_{0,p-1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ b_{p/2-1,0} & \cdots & b_{p/2-1,p/2-1} & \cdots & b_{p/2-1,p-1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ b_{p-1,0} & \cdots & b_{p-1,p/2-1} & \cdots & b_{p-1,p-1} \end{Bmatrix} = \begin{Bmatrix} B_0 & B_1 \\ B_2 & B_3 \end{Bmatrix}$$

3: Partition matrix C of size p into:

$$C = \begin{Bmatrix} A_0 \times B_0 + A_1 \times B_2 & A_0 \times B_1 + A_1 \times B_3 \\ A_2 \times B_0 + A_3 \times B_2 & A_2 \times B_1 + A_3 \times B_3 \end{Bmatrix} = \begin{Bmatrix} P_0 + P_1 & P_2 + P_3 \\ P_4 + P_5 & P_6 + P_7 \end{Bmatrix}$$

4: Parallel multiplication
5: $MatMul(P_0, A_0, B_0)$
6: $MatMul(P_1, A_1, B_2)$
7: $MatMul(P_2, A_0, B_1)$
8: $MatMul(P_3, A_1, B_3)$
9: $MatMul(P_4, A_2, B_0)$
10: $MatMul(P_5, A_3, B_2)$
11: $MatMul(P_6, A_2, B_1)$
12: $MatMul(P_7, A_3, B_3)$
13: Parallel addition
14: $MatAdd(C_0, P_0, P_1)$
15: $MatAdd(C_1, P_2, P_3)$
16: $MatAdd(C_2, P_4, P_5)$
17: $MatAdd(C_3, P_6, P_7)$

- Thread info/shared data are stored in Shared Memory.
- Maximize Local Memory Utilization.
- Reduce shared resource usage (cache, bus).
- Reduce Local Memory replacement.

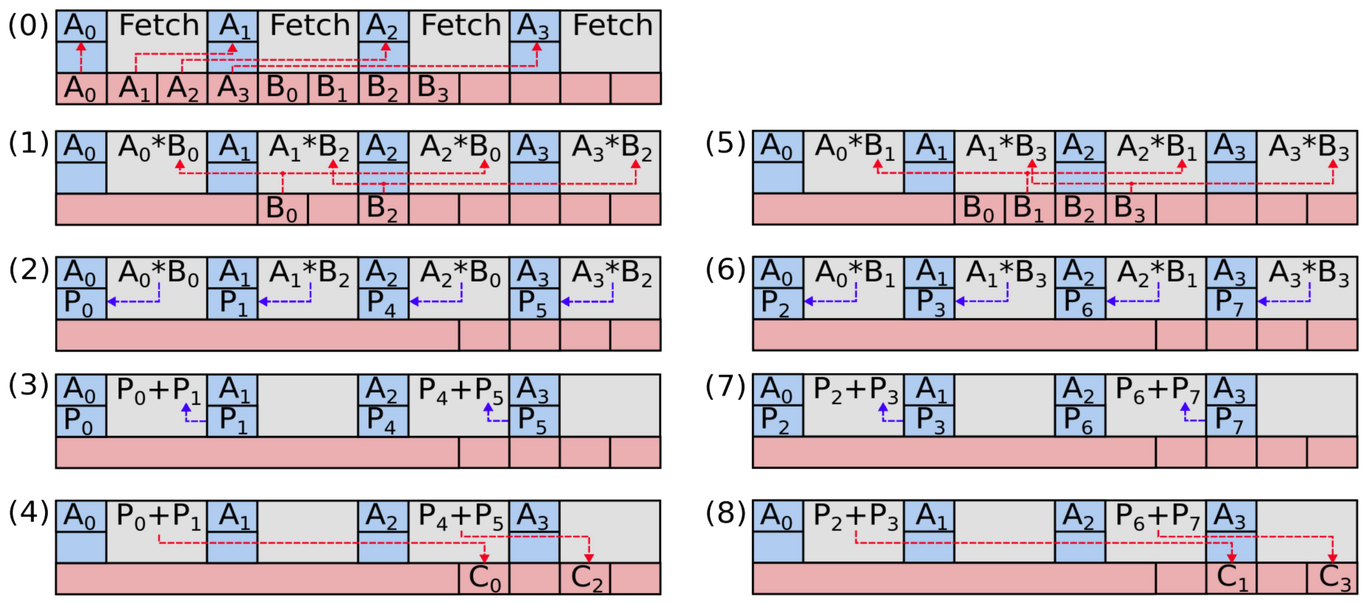

Figure 3 Dataflow for Strassen's Matrix Multiplication Algorithm

Size	Iterative					
	Instruction count	Time (cycles)		CPI		Speed up over single-thread
		Single-thread	Multi-thread	Single-thread	Multi-thread	
8	6,444	32,324	7,578	5.02	1.17	4.27
16	50,271	247,612	64,662	4.92	1.28	3.83
32	496,942	2,036,081	485,057	4.09	0.98	4.19
64	3,953,166	15,225,460	4,134,107	3.85	1.05	3.68
Size	Strassen					
	Instruction count	Time (cycles)		CPI		Speed up over
		Single-thread	Multi-thread	Single-thread	Multi-thread	Single-thread / Iterative (multi)
8	6,290	40,940	7,395	6.51	1.17	5.53 / 1.02
16	53,271	247,612	51,186	4.65	0.96	4.83 / 1.26
32	406,451	2,036,081	380,012	5.01	0.93	5.35 / 1.27
64	3,966,614	17,820,396	3,202,560	4.49	0.81	5.56 / 1.29

[1] Yuta Nojiri, et al.: A Design of Multithreaded RISC-V Processor for Real-Time System, CANDARW, Nov. 2023.
[2] Abdallah Cheikh, et al.: Klessydra-T: Designing Vector Coprocessors for Multi-Threaded Edge-Computing Cores, IEEE Micro, Jan. 2021.