HYDRA: a multi-core RISC-V with cryptographically useful modes of operation

Ben Marshall[1] Dan Page[2] Thinh Pham[2] Max Whale[2]

[1] PQShield Ltd, Oxford, UK.
ben.marshall@pqshield.com
[2] Department of Computer Science, University of Bristol.
{daniel.page,th.pham,mw17440}@bristol.ac.uk

June 16, 2022

Motivation

- Computing parallelism:
  - Data Level (DLP), Instruction Level (ILP), Thread Level (TLP)
  - DLP via Single Instruction Multiple Data (SIMD) extension
  - The trend from single- toward multi-core processor designs [5]
- Composable lightweight processor [13]
  - Flexible multi-core configuration.
  - Exploit the advantages of DLP and TLP.
- Case study in Cryptography:
  - Performance (i.e., speed up cryptographic software)
  - Security/Reliability (i.e., Fault Attack/Detection).
- The implementation is available: https://github.com/scarv/hydra, under an open-source license

Composable System Design Concept

- Consider a set of p atomic processor cores:

$$P = \{P_0, P_1, \ldots, P_{p-1}\}$$

- Define a composable configuration:

$$C^M(P) = \{\{\overline{P_0}, \ldots, P_{n-1}\}, \{P_n\}, \ldots, \{P_{p-1}\}\}.$$

- The system will be reconfigured as
  - A single n-composed system operating in M mode.

    $C(P)_0^M = \{\overline{P_0}, \ldots, P_{n-1}\}$; $\overline{P_0}$ denotes primary core
  - Multiple single conventional cores

    $C(P)_1 = \{P_n\}; \quad \ldots; \quad C(P)_{p-n} = \{P_{p-1}\}$

- Generic composable computing concept
  - Given two $n \times b$-bit operands represented by b-bit elements:

$$x = \{x_{n-1} \parallel x_{\ldots} \parallel x_0\}; \qquad y = \{y_{n-1} \parallel y_{\ldots} \parallel y_0\};$$

  - And each b-bit processor, p_i, supports $\mathfrak{I}_{\oslash}$ instruction; s_i denotes status flags.

$$\{s_i, r_i\} \leftarrow \mathfrak{I}_{\oslash}^{s_i}(x_i, y_i)$$

  - Compute $\oslash$ operation:

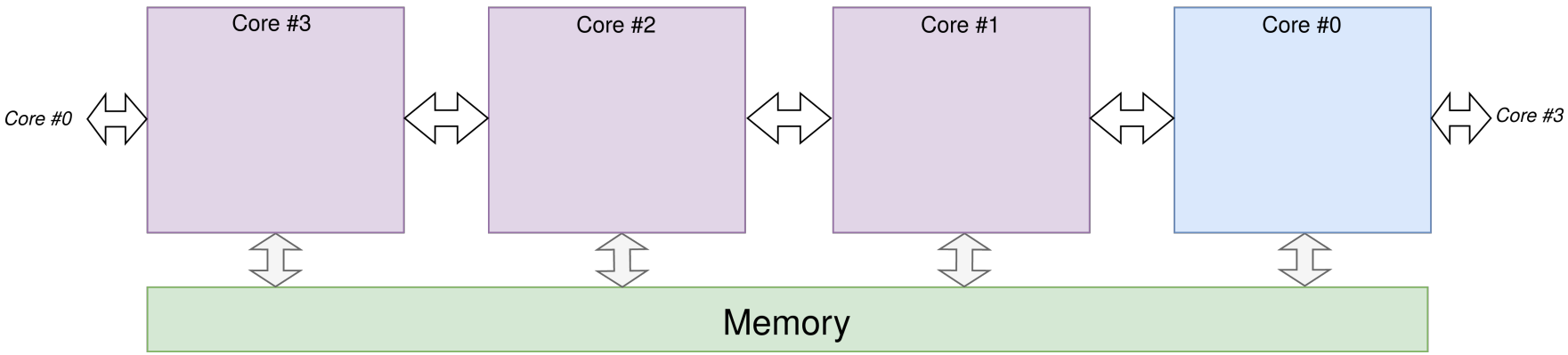
$$r = \{r_{n-1} \parallel r_{\ldots} \parallel r_0\} = x \oslash y = \{x_{n-1} \parallel x_{\ldots} \parallel x_0\} \oslash \{y_{n-1} \parallel y_{\ldots} \parallel y_0\};$$

  - needs:

$$\mathfrak{I}_{\oslash}^{s_0}(x_0, y_0) \longleftrightarrow \mathfrak{I}_{\oslash}^{s_1}(x_1, y_1) \ldots \longleftrightarrow \mathfrak{I}_{\oslash}^{s_{n-1}}(x_{n-1}, y_{n-1})$$

- Three composed modes depend on operation, $\oslash$, and operands (i.e., $\parallel$)

  SIMD Compute Mode, Wide Data-path Mode and SIMD Redundant Mode:



SIMD Compute Mode

- Consider operands are sets of n b-bit elements,
  $\|$: independent elements, denoted $\{\ ;\ \}$

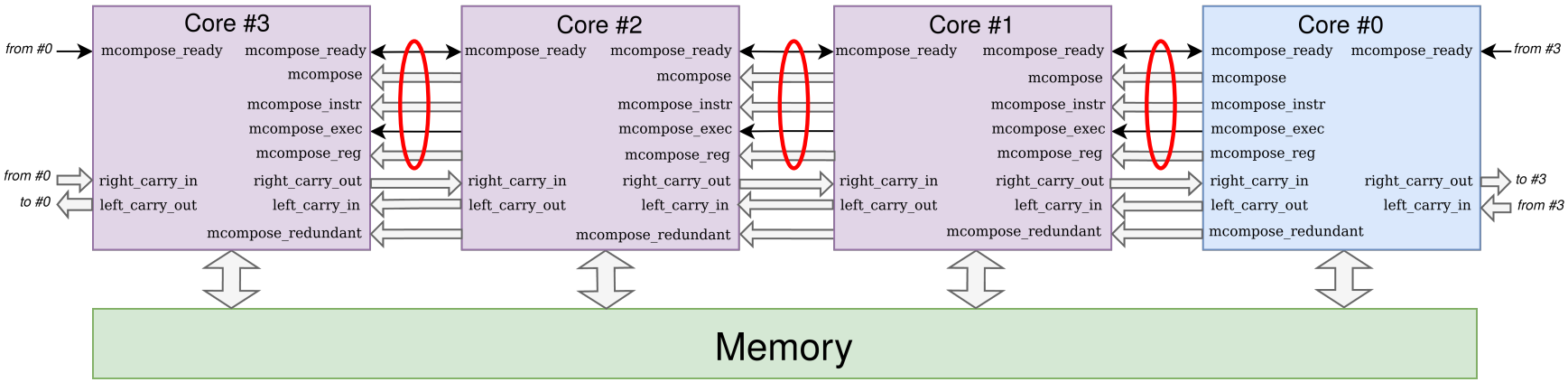
$$x = \{x_{n-1};\ x_{...};\ x_0\}; \qquad y = \{y_{n-1};\ y_{...};\ y_0\};$$

- To compute $\oslash$ operation:

$$r = \{r_{n-1};\ r_{...};\ r_0\} = \{x_{n-1};\ x_{...};\ x_0\} \oslash \{y_{n-1};\ y_{...};\ y_0\};$$

- Each p_i atomic processor core in the composed system.

$$r_i \leftarrow \Im_{\oslash}(x_i, y_i)$$



Wide Data-path Mode

- Consider operands are $n \times b$-bit values,
  $\|$: bit-order dependent elements, denoted { _ }

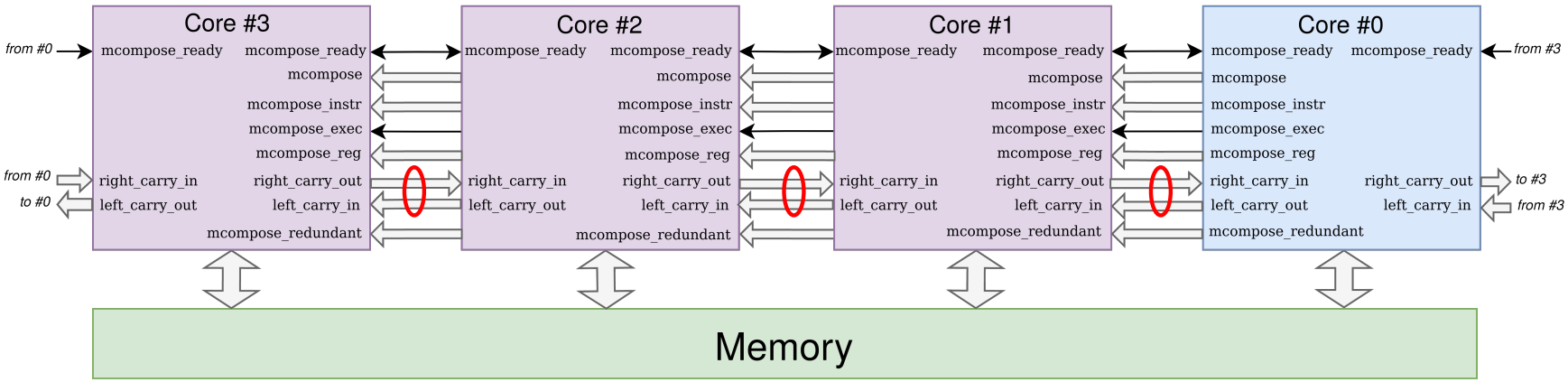
$$x = \{x_{n-1}\_x_{\ldots}\_x_0\}; \qquad y = \{y_{n-1}\_y_{\ldots}\_y_0\};$$

- To compute $\oslash$ operation:

$$r = \{r_{n-1}\_r_{\ldots}\_r_0\} = \{x_{n-1}\_x_{\ldots}\_x_0\} \oslash \{y_{n-1}\_y_{\ldots}\_y_0\};$$

- Each p_i atomic processor core in the composed system.
  $s_i \equiv c_i$: using carrier flags

$$\{c_i, r_i\} \leftarrow \mathfrak{I}_{\oslash}^{c_{i-1}}(x_i, y_i)$$



SIMD Redundant Mode

- Detect if a fault happens on: $\underline{x} \oslash \underline{y}$
- Redundantly extend scalar operands to sets of n same value elements,

$$x = \{\underline{x};\ \ldots;\ \underline{x}\}; \qquad y = \{\underline{y};\ \ldots;\ \underline{y}\};$$

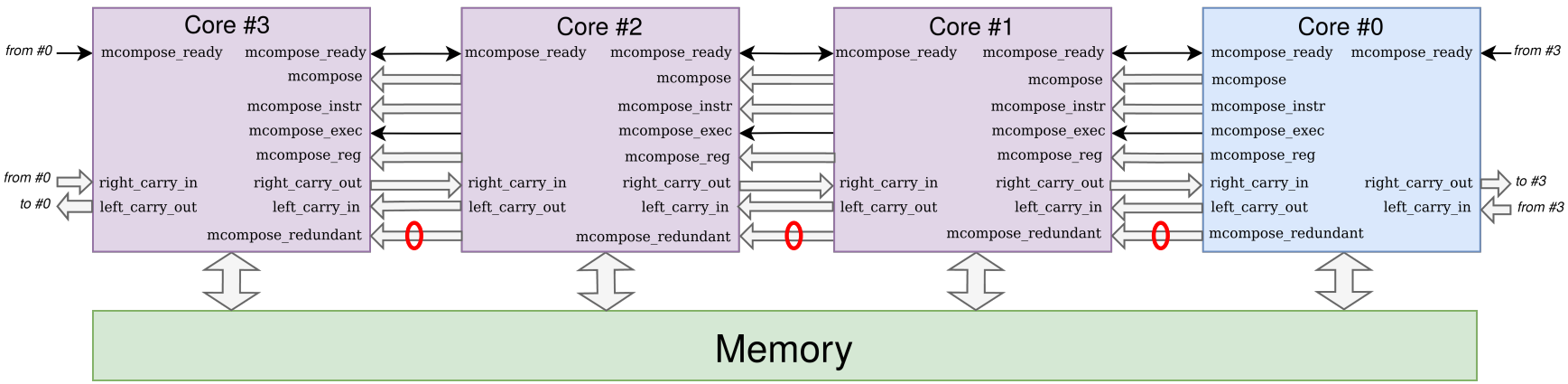
  { ; }: independent elements

- Synchronise : $s_0, s_1, \ldots, s_{n-1}$; Compute $\oslash$: $r = x \oslash y$;
- Each p_i atomic processor core in the composed system.

$$r_i \leftarrow \mathfrak{I}_{\oslash}(x_i, y_i)$$

- To detect a fault, check:

$$\mathfrak{I}_{\oslash}^{s_0}(x_0, y_0) == \mathfrak{I}_{\oslash}^{s_1}(x_1, y_1) \ldots == \mathfrak{I}_{\oslash}^{s_{n-1}}(x_{n-1}, y_{n-1})$$



Hardware Implementation

- A proof of concept on an FPGA platform:
  - Modified PicoRV32[1]: an area-optimised, non-pipelined core
  - Interconnection to connect 4 cores to a shared memory
  - Xilinx Kintex-7 (xc7k160tfbg676) FPGA on SASEBO-GIII [8]
  - Clock speed of 50 MHz
- Hardware overheads of 4-core conventional v.s. composable alternative

	LUTs	FFs	Longest delay	
Original core	1425 (1.00×)	968 (1.00×)	–	–
Primary core	1881 (1.32×)	1109 (1.15×)	–	–
Secondary core	1897 (1.33×)	1059 (1.09×)	–	–
Conventional system	10964 (1.00×)	4181 (1.00×)	5.324	(1.00×)
Composable system	12522 (1.14×)	4517 (1.08×)	7.096	(1.33×)

[1] https://github.com/cliffordwolf/picorv32

Software

- Software Adaptation:

  An example of composed version of multi-precision addition

```
//Composed multi-precision addition
//input:  int *a_addr, int *b_addr,
//        int n_bytes, int n_cores
//output: int *r_addr
mp_add_com:
csrwi mcompose_mode, mcompose_wide
csrw  mcompose_reg,  n_cores
slli bytes_per_word, n_cores, 2

li   carry, 0
add  addr_end, n_bytes, a_addr
mp_add_comp_loop:
beq  a_addr,  addr_end, mp_add_end
lw   a_value, 0(a_addr)
lw   b_value, 0(b_addr)
add  r_value, a_value,  carry
sltu carry,   r_value,  a_value
add  r_value, r_value,  b_value
sltu t_carry, r_value,  b_value
or   carry,   carry,    t_carry
sw   r_value, 0(r_addr)
add  a_addr,  a_addr,   bytes_per_word
add  b_addr,  b_addr,   bytes_per_word
add  r_addr,  r_addr,   bytes_per_word
j    mp_add_comp_loop
mp_add_end:

csrw mcompose_reg, zero
ret
```

- Cryptographic software benchmarks:
  - Multi-precision algorithms: addition, multiplication, and modular exponentiation (modExp)
  - ChaCha20 stream cipher
  - AES-128 encryption

Performance Speed-up Results

- Speed up multi-precision algorithms using the wide data-path mode

1024-bit Operations	Metric	Single Core	Composed Systems	
			2 Cores	4 Cores
Addition	Instructions	431 (1.00×)	228 (1.89×)	124 (3.48×)
	Cycles	2023 (1.00×)	1225 (1.65×)	843 (2.40×)
Multiplication	Instructions	15822 (1.00×)	4083 (3.88×)	1091 (14.50×)
	Cycles	179395 (1.00×)	72490 (2.47×)	32086 (5.59×)
ModExp	Instructions	57395054 (1.00×)	15019653 (3.82×)	4144180 (13.85×)
	Cycles	594838395 (1.00×)	238293765 (2.50×)	106391562 (5.59×)

- Speed up ChaCha20 encryption in using the SIMD compute mode

Message size	Metric	OpenSSL	Single Core	Composed Systems		128 bit Vector [15]	
				2 Cores	4 Cores		
64 bytes	Instructions	2825 (1.00×)	1765 (1.60×)	1199 (2.36×)	659 (4.29×)	607	(4.65×)
	Cycles	25073 (1.00×)	17603 (1.42×)	11905 (2.11×)	7609 (3.30×)		
256 bytes	Instructions	11015 (1.00×)	6919 (1.59×)	4637 (2.38×)	2537 (4.34×)	2332	(4.72×)
	Cycles	98969 (1.00×)	69524 (1.42×)	46435 (2.13×)	29659 (3.34×)		
1024 bytes	Instructions	43775 (1.00×)	27535 (1.59×)	18389 (2.38×)	10049 (4.36×)	9232	(4.74×)
	Cycles	394553 (1.00×)	277208 (1.42×)	184555 (2.14×)	117859 (3.35×)		

Fault Detection Results

- Consider fault injection attacks on AES block cipher
  - Use the simulation-based to inject faults
  - 2 types of fault injections: control flow and data flow
  - Perform 100 times per experiment
  - AES encryption results are checked against the correct value,

    If failed, attacker takes advantage of the faults

- AES encryption against control flow and data fault injections

	Control flow fault case				Data fault case			
Implementations	Passed	Failed	Broken	Detected	Passed	Failed	Broken	Detected
Unprotected AES	30	62	8	–	42	29	29	–
Protected AES	12	0	2	87	46	0	0	54

Concluding Remarks

- A proof-of-concept multi-core processor implemented on a FPGA platform
  - Present a Composable Lightweight Processor to support cryptographic workloads
  - General-purpose benefits of a multi-core processor
  - Flexible composed modes to address pertinent efficiency or security challenges
- Furture Works
  - Explore and address system-level challenges
  - Address obvious drawbacks, e.g., the increased critical delay in Wide Data-path
  - Consider other base-cores, e.g., a pipelined processor instead of PicoRV32
- Open-Source: https://github.com/scarv/hydra

References

[1] *Advanced Encryption Standard (AES)*. National Institute of Standards and Technology, Federal Information Processing Standard (FIPS) 197. 2001. URL: http://csrc.nist.gov.

[2] J.A. Ambrose, S. Parameswaran, and A. Ignjatovic. "MUTE-AES: A multiprocessor architecture to prevent power analysis based side channel attack of the AES algorithm". In: *IEEE/ACM International Conference on Computer-Aided Design*. 2008, pp. 678–684.

[3] A. Barenghi et al. "Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures". In: *Proceedings of the IEEE* 100.11 (2012), pp. 3056–3076.

[4] Daniel J. Bernstein. "ChaCha, a variant of Salsa20". In: *Workshop Record of SASC: The State of the Art of Stream Ciphers*. 2008.

[5] G. Blake, R.G. Dreslinski, and T. Mudge. "A Survey of Multicore Processors". In: *IEEE Signal Processing Magazine* 26.6 (2009), pp. 26–37.

[6] J.W. Bos, T. Kleinjung, and D. Page. *Efficient Modular Multiplication*. Cryptology ePrint Archive, Report 2021/1151. 2021. URL: https://ia.cr/2021/1151.

[7] Y. Cheng et al. "Efficient Multiple-ISA Embedded Processor Core Design Based on RISC-V". In: *Workshop on Computer Architecture Research with RISC-V (CARRV)*. 2020.

[8] Y. Hori et al. "SASEBO-GIII: A hardware security evaluation board equipped with a 28-nm FPGA". In: *IEEE Global Conference on Consumer Electronics*. https://doi.org/10.1109/GCCE.2012.6379944. 2012, pp. 657–660.

[9] M. Joye and M. Tunstall, eds. *Fault Analysis in Cryptography*. Information Security and Cryptography. Springer, 2012.

[10] D. Karaklajić, J.-M. Schmidt, and I. Verbauwhede. "Hardware Designer's Guide to Fault Attacks". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21.12 (2013), pp. 2295–2306.

[11] A. Karatsuba and Y. Ofman. "Multiplication of Many-Digital Numbers by Automatic Computers". In: *Physics-Doklady* 7 (1963), pp. 595–596.

[12] C. Kaya Koc, T. Acar, and B. S. Kaliski. "Analyzing and comparing Montgomery multiplication algorithms". In: *IEEE Micro* 16.3 (June 1996), pp. 26–33. ISSN: 0272-1732.

[13] C. Kim et al. "Composable Lightweight Processors". In: *International Symposium on Microarchitecture (MICRO)*. 2007, pp. 381–394.

[14] R.B. Lee, X. Yang, and Z. Shi. "Validating Word-Oriented Processors for Bit and Multi-word Operations". In: *Annual Computer Security Applications Conference (ACSAC)*. 2004, pp. 473–488.

References (cont.)

[15] B. Marshall, D. Page, and T. H. Pham. "A lightweight ISE for ChaCha on RISC-V". In: *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 2021, pp. 25–32.

[16] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC, 1996.

[17] E. Rescorla. *The Transport Layer Security (TLS) protocol version 1.3*. Internet Engineering Task Force Request for Comments (RFC) 8446. 2018. URL: http://tools.ietf.org/html/rfc8446.

[18] R.L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the ACM (CACM)* 21.2 (1978), pp. 120–126.

[19] M. Tunstall, D. Mukhopadhyay, and S. Ali. "Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault". In: *Workshop on Information Security Theory and Practice (WISTP)*. LNCS 6633. Springer-Verlag, 2011, pp. 224–233.

[20] B. Yuce et al. "A Secure Exception Mode for Fault-Attack-Resistant Processing". In: *IEEE Transactions on Dependable and Secure Computing* 16.3 (2019), pp. 388–401.

[21] B. Yuce et al. "FAME: Fault-Attack Aware Microprocessor Extensions for Hardware Fault Detection and Software Fault Response". In: *Hardware and Architectural Support for Security and Privacy (HASP)*. 2016, 8:1–8:8.