Seeds of SEED: H-CRAM: In-memory Homomorphic Search Accelerator using Spintronic Computational RAM

Hüsrev Cılasun, Salonik Resch, Zamshed I. Chowdhury, Masoud Zabihi, Zhengyang Zhao, Thomas Peterson, Jian-Ping Wang, Sachin S. Sapatnekar, Ulya R. Karpuzcu

University of Minnesota

This work was supported in part by NSF grant no. SPX-1725420.

Introduction

- Secure data retrieval
    - Sensitive databases, bioinformatics, biomarker search
    - Encryption/decryption is a vulnerability

Introduction

- Secure data retrieval
  - Sensitive databases, bioinformatics, biomarker search
  - Encryption/decryption is a vulnerability
- Homomorphic Encryption
  - Computation in encrypted domain
  - General-purpose computation is **prohibitively costly**
    - Encrypted computation accumulates noise on cyphertext
    - Bootstrapping: noise reduction without decryption

Introduction

- Secure data retrieval
  - Sensitive databases, bioinformatics, biomarker search
  - Encryption/decryption is a vulnerability
- Homomorphic Encryption
  - Computation in encrypted domain
  - General-purpose computation is **prohibitively costly**
    - Encrypted computation accumulates noise on cyphertext
    - Bootstrapping: noise reduction without decryption
- **SCAM**: **S**ecure **C**ontent **A**ddressable **M**emory
  - Encrypted matching can avoid bootstrapping
  - Reduces to parallelized long addition
  - Data size blowup

Introduction

- Secure data retrieval
    - Sensitive databases, bioinformatics, biomarker search
    - Encryption/decryption is a vulnerability
- Homomorphic Encryption
    - Computation in encrypted domain
    - General-purpose computation is **prohibitively costly**
        - Encrypted computation accumulates noise on cyphertext
        - Bootstrapping: noise reduction without decryption
- **SCAM**: **S**ecure **C**ontent **A**ddressable **M**emory
    - Encrypted matching can avoid bootstrapping
    - Reduces to parallelized long addition
    - Data size blowup
- **Spintronic CRAM**: Fuses compute and memory
    - True in-memory processing semantics
    - Enables massive parallelism

Introduction

- Secure data retrieval
  - Sensitive databases, bioinformatics, biomarker search
  - Encryption/decryption is a vulnerability
- Homomorphic Encryption
  - Computation in encrypted domain
  - General-purpose computation is **prohibitively costly**
    - Encrypted computation accumulates noise on cyphertext
    - Bootstrapping: noise reduction without decryption
- **SCAM**: **S**ecure **C**ontent **A**ddressable **M**emory
  - Encrypted matching can avoid bootstrapping
  - Reduces to parallelized long addition
  - Data size blowup
- **Spintronic CRAM**: Fuses compute and memory
  - True in-memory processing semantics
  - Enables massive parallelism
- **H-CRAM**: SCAM in CRAM to accelerate homomorphic search

SCAM Fundamentals

- Plaintext comparison of *w*-bit *x* and *y*:

$$f(x, y) = \prod_{i=1}^{w} \overline{x_i \oplus y_i} \tag{1}$$

SCAM Fundamentals

- Plaintext comparison of *w*-bit *x* and *y*:

$$f(x, y) = \prod_{i=1}^{w} \overline{x_i \oplus y_i} \qquad (1)$$

- Homomorphic equivalent is cyphertext subtraction:

$$Hom\widehat{XOR}\text{-}OR(x, y) = \sum_{i=0}^{w} c_{x_i} - c_{y_i} \qquad (2)$$

SCAM Fundamentals

- Plaintext comparison of *w*-bit *x* and *y*:

$$f(x,y) = \prod_{i=1}^{w} \overline{x_i \oplus y_i} \qquad (1)$$

- Homomorphic equivalent is cyphertext subtraction:

$$Hom\widehat{XOR}\text{-}OR(x,y) = \sum_{i=0}^{w} c_{x_i} - c_{y_i} \qquad (2)$$

- Cyphertext expansion: **6.8KB per bit!** (128-bit security)

SCAM Fundamentals

- Plaintext comparison of w-bit x and y:

$$f(x,y) = \prod_{i=1}^{w} \overline{x_i \oplus y_i} \tag{1}$$

- Homomorphic equivalent is cyphertext subtraction:

$$Hom\widehat{XOR}\text{-}OR(x,y) = \sum_{i=0}^{w} c_{x_i} - c_{y_i} \tag{2}$$

- Cyphertext expansion: **6.8KB per bit!** (128-bit security)
- Costly in CPU $\longrightarrow$ **ASIC acceleration**

SCAM Fundamentals

- Plaintext comparison of w-bit x and y:

$$f(x, y) = \prod_{i=1}^{w} \overline{x_i \oplus y_i} \tag{1}$$

- Homomorphic equivalent is cyphertext subtraction:

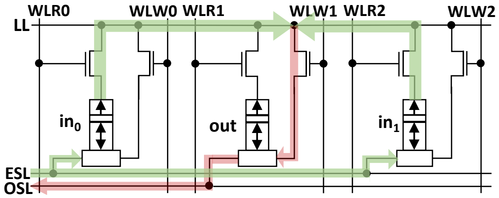
$$Hom\widehat{XOR}\text{-}OR(x, y) = \sum_{i=0}^{w} c_{x_i} - c_{y_i} \tag{2}$$

- Cyphertext expansion: **6.8KB per bit!** (128-bit security)
- Costly in CPU $\longrightarrow$ **ASIC acceleration**
- Suffers from memory access overhead

CRAM Basics

2Transistor-1Magnet CRAM cells

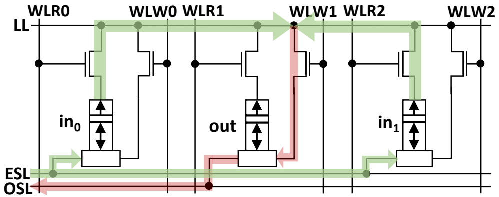
- Logic Line (LL)
- Even/Odd Select Lines (E/OSL)
- Wordline Read/Write (WLR/W)



CRAM Basics

2Transistor-1Magnet CRAM cells

- Logic Line (LL)
- Even/Odd Select Lines (E/OSL)
- Wordline Read/Write (WLR/W)

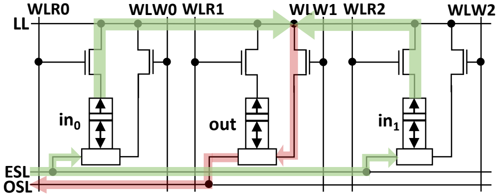


- Cell states Parallel/Antiparallel↔low/high resistance (∼0/1)
- **Memory mode:** Standard read/write operations

CRAM Basics

2Transistor-1Magnet CRAM cells

- Logic Line (LL)
- Even/Odd Select Lines (E/OSL)
- Wordline Read/Write (WLR/W)

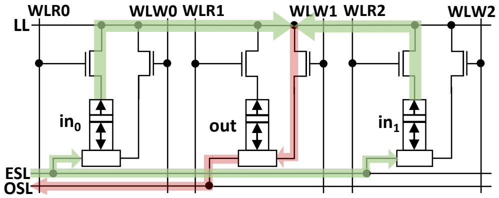


- Cell states Parallel/Antiparallel↔low/high resistance (∼0/1)
- **Memory mode:** Standard read/write operations
- **Compute mode:** Logic gate formation between cells

CRAM Basics

2Transistor-1Magnet CRAM cells

- Logic Line (LL)
- Even/Odd Select Lines (E/OSL)
- Wordline Read/Write (WLR/W)

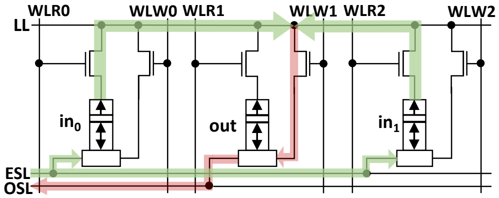


- Cell states Parallel/Antiparallel↔low/high resistance (∼0/1)
- **Memory mode:** Standard read/write operations
- **Compute mode:** Logic gate formation between cells
  - A resistive network is established using control lines

CRAM Basics

2Transistor-1Magnet CRAM cells

- Logic Line (LL)
- Even/Odd Select Lines (E/OSL)
- Wordline Read/Write (WLR/W)

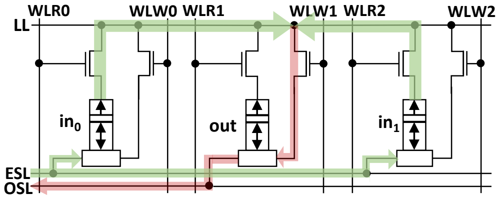


- Cell states Parallel/Antiparallel↔low/high resistance (∼0/1)
- **Memory mode:** Standard read/write operations
- **Compute mode:** Logic gate formation between cells
  - A resistive network is established using control lines
  - in_0&in_1 currents get combined, pass through *out* (preset to 1)

CRAM Basics

2Transistor-1Magnet CRAM cells

- Logic Line (LL)
- Even/Odd Select Lines (E/OSL)
- Wordline Read/Write (WLR/W)

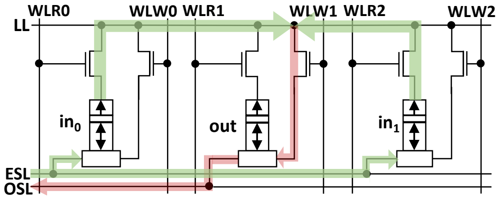


- Cell states Parallel/Antiparallel↔low/high resistance (∼0/1)
- **Memory mode:** Standard read/write operations
- **Compute mode:** Logic gate formation between cells
  - A resistive network is established using control lines
  - in_0&in_1 currents get combined, pass through *out* (preset to 1)
    - *out* remains unchanged if $in_0 in_1 = 00, 01, 10$
    - *out* reset to 0 if $in_0 in_1 = 11$

CRAM Basics

2Transistor-1Magnet CRAM cells

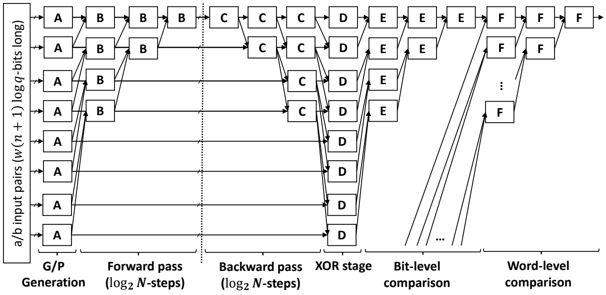
- Logic Line (LL)
- Even/Odd Select Lines (E/OSL)
- Wordline Read/Write (WLR/W)



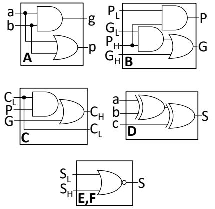
- Cell states Parallel/Antiparallel↔low/high resistance (∼0/1)
- **Memory mode:** Standard read/write operations
- **Compute mode:** Logic gate formation between cells
  - A resistive network is established using control lines
  - *in_0&in_1* currents get combined, pass through *out* (preset to 1)
    - *out* remains unchanged if $in_0 in_1 = 00, 01, 10$
    - *out* reset to 0 if $in_0 in_1 = 11$
  - **Practically universal NAND gate!**

H-CRAM: SCAM in CRAM

- RCA: $O(n)$, CLA and others: $O(n \log n)$ ($O(\log n)$ depth)
- CLA adder based SCAM logic flow

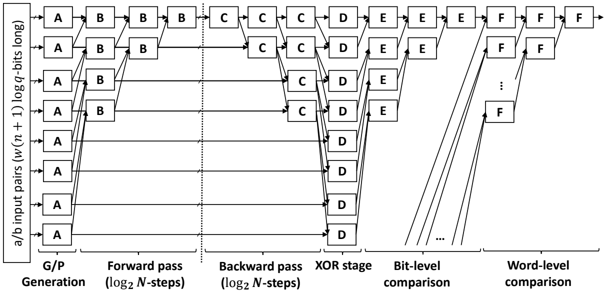


a/b input pairs ($w(n+1)\log q$-bits long)
A
B
C
D
E
F
G/P Generation
Forward pass ($\log_2 N$-steps)
Backward pass ($\log_2 N$-steps)
XOR stage
Bit-level comparison
Word-level comparison

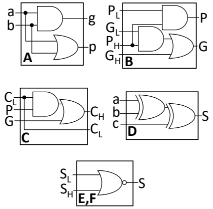


H-CRAM: SCAM in CRAM

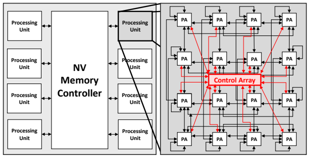
- RCA: $O(n)$, CLA and others: $O(n \log n)$ ($O(\log n)$ depth)
- CLA adder based SCAM logic flow

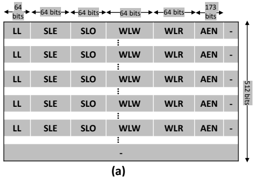


a/b input pairs ($w(n+1)\log q$-bits long)
A
B
C
D
E
F
G/P Generation
Forward pass ($\log_2 N$-steps)
Backward pass ($\log_2 N$-steps)
XOR stage
Bit-level comparison
Word-level comparison



- Mapped to **Processing** and **Control** arrays
- De Bruijn graph topology for inter-array connectivity

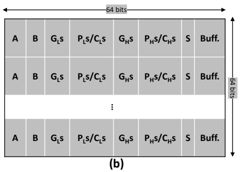




64 bits
64 bits
64 bits
64 bits
64 bits
173 bits
512 bits

LL	SLE	SLO	WLW	WLR	AEN	-
LL	SLE	SLO	WLW	WLR	AEN	-
LL	SLE	SLO	WLW	WLR	AEN	-
LL	SLE	SLO	WLW	WLR	AEN	-
LL	SLE	SLO	WLW	WLR	AEN	-
			⋮			
			-			

(a)

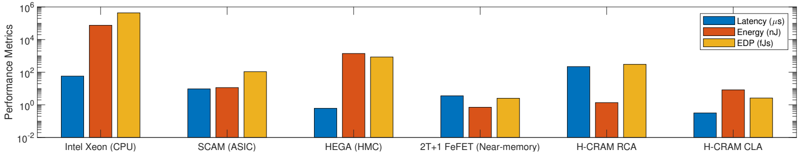


64 bits
64 bits

A	B	G_Ls	P_Ls/C_Ls	G_Hs	P_Hs/C_Hs	S	Buff.
A	B	G_Ls	P_Ls/C_Ls	G_Hs	P_Hs/C_Hs	S	Buff.
			⋮				
A	B	G_Ls	P_Ls/C_Ls	G_Hs	P_Hs/C_Hs	S	Buff.

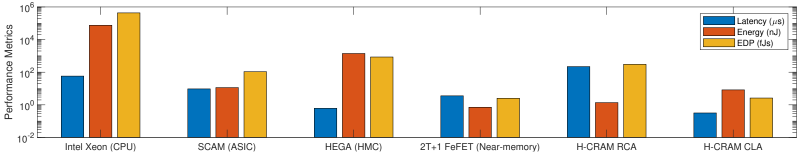
(b)

Results

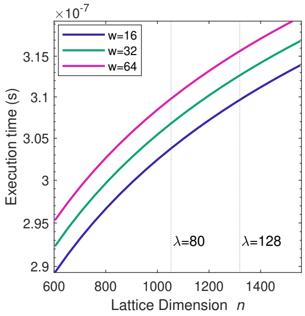


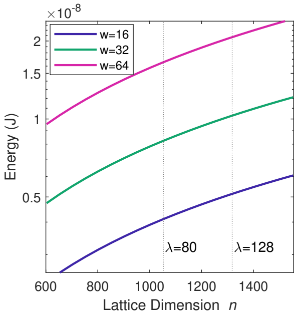
H-CRAM CLA is **2**$\times$ **faster** vs. HEGA, $\sim$**EDP** vs. 2T+1 FeFET

Results



H-CRAM CLA is **2**$\times$ **faster** vs. HEGA, $\sim$**EDP** vs. 2T+1 FeFET





H-CRAM scales efficiently with word size!

Vision

- H-CRAM enables
  - Multi-grain parallelism
  - Memory access overhead elimination
  - True processing in-memory semantics, tight coupling of memory&compute

Vision

- H-CRAM enables
  - Multi-grain parallelism
  - Memory access overhead elimination
  - True processing in-memory semantics, tight coupling of memory&compute
- Efficient acceleration of long addition in SCAM

Vision

- H-CRAM enables
  - Multi-grain parallelism
  - Memory access overhead elimination
  - True processing in-memory semantics, tight coupling of memory&compute
- Efficient acceleration of long addition in SCAM
- Future work: general purpose homomorphic computing
  - How to extend H-CRAM to more complex homomorphic computation without bootstrapping?

Thank you!
Questions?