Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

Vivek Seshadri[1,5] Donghyuk Lee[2,5] Thomas Mullins[3,5] Hasan Hassan[4]
Amirali Boroumand[5] Jeremie Kim[4,5] Michael A. Kozuch[3] Onur Mutlu[4,5]
Phillip B. Gibbons[5] Todd C. Mowry[5]

[1]Microsoft Research India [2]NVIDIA Research [3] Intel [4]ETH Zürich [5]Carnegie Mellon University

Presented at MICRO 2017

Moritz Knüsel

Seminar on Computer Architecture

Executive Summary

- **Problem: Bulk bitwise operations**
  - Problematic when used on large in-memory bitvectors
  - **Limited** by available memory bandwidth

Executive Summary

- **Problem: Bulk bitwise operations**
  - Problematic when used on large in-memory bitvectors
  - **Limited** by available memory bandwidth
- The proposal: Ambit
  - Perform bulk bitwise operation **in DRAM**
  - **Activate multiple DRAM rows** to compute AND/OR
  - Use **existing inverters** to compute NOT
  - AND, OR and NOT are sufficient to compute all bitwise operations (NAND,XOR,...)
  - **less than 1%** area overhead

Executive Summary

- **Problem: Bulk bitwise operations**
  - Problematic when used on large in-memory bitvectors
  - **Limited** by available memory bandwidth
- The proposal: Ambit
  - Perform bulk bitwise operation **in DRAM**
  - **Activate multiple DRAM rows** to compute AND/OR
  - Use **existing inverters** to compute NOT
  - AND, OR and NOT are sufficient to compute all bitwise operations (NAND,XOR,...)
  - **less than 1%** area overhead
- Results compared to state-of-the-art:
  - **32x** performance improvement, **35x** energy reduction averaged across 7 bulk bitwise operations
  - **3x-7x** performance improvement for real-world data intensive workloads

Background, Problem & Goal

Background

- Applications that rely on bulk bitwise operations include:

Background

- Applications that rely on bulk bitwise operations include:
  - Bitmap indices in databases

Background

- Applications that rely on bulk bitwise operations include:
  - Bitmap indices in databases
  - Set operations

Background

- Applications that rely on bulk bitwise operations include:
  - Bitmap indices in databases
  - Set operations
  - DNA sequencing

Background

- Applications that rely on bulk bitwise operations include:
    - Bitmap indices in databases
    - Set operations
    - DNA sequencing
    - Encryption

Background

- Applications that rely on bulk bitwise operations include:
  - Bitmap indices in databases
  - Set operations
  - DNA sequencing
  - Encryption
  - ...

Background

- Applications that rely on bulk bitwise operations include:
  - Bitmap indices in databases
  - Set operations
  - DNA sequencing
  - Encryption
  - ...
- Bulk bitwise operations become problematic if

Background

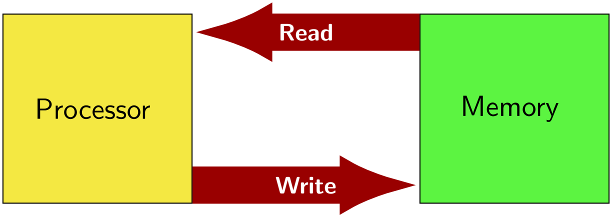
- Applications that rely on bulk bitwise operations include:
  - Bitmap indices in databases
  - Set operations
  - DNA sequencing
  - Encryption
  - ...
- Bulk bitwise operations become problematic if
  - The bitvectors involved are very large

Background

- Applications that rely on bulk bitwise operations include:
  - Bitmap indices in databases
  - Set operations
  - DNA sequencing
  - Encryption
  - ...
- Bulk bitwise operations become problematic if
  - The bitvectors involved are very large
  - They cannot be combined with other processing

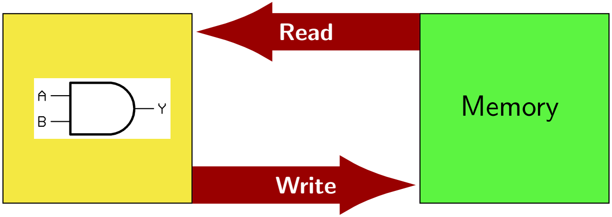
Problem

Memory bandwidth is a bottleneck



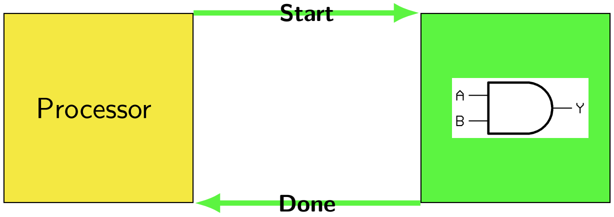
Problem

Memory bandwidth is a bottleneck



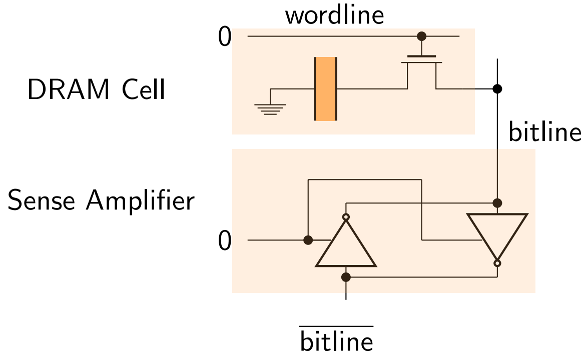
Goal

Idea: Perform bitwise operations directly in DRAM

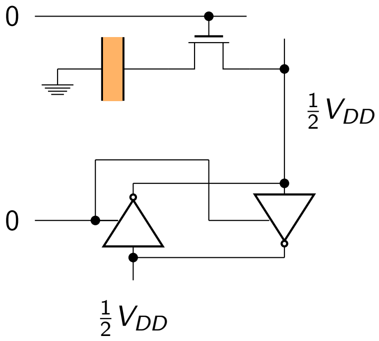


Mechanism

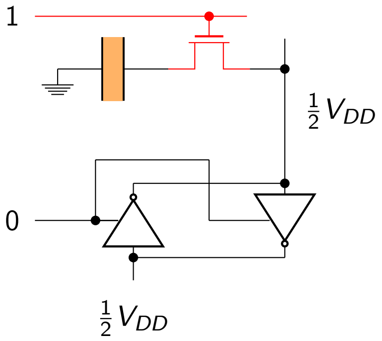
Background on DRAM



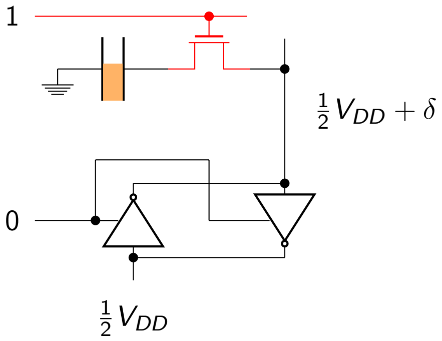
Precharged State



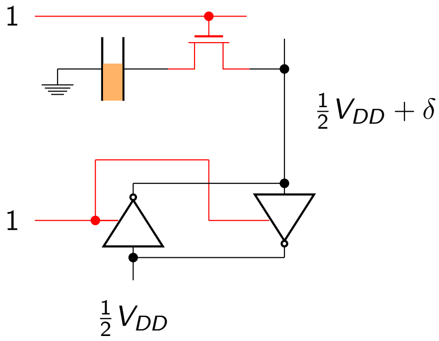
Wordline Enable



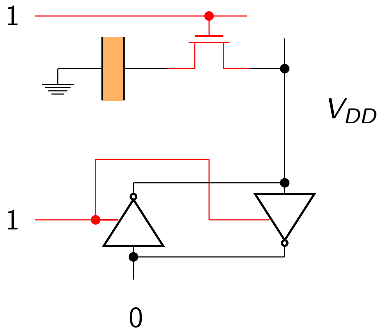
Charge Sharing Phase



Sense Amplifier Enable



Activated State



Ambit-AND-OR

Ambit-AND-OR

- Ambit-AND-OR relies on analog charge sharing.

Ambit-AND-OR

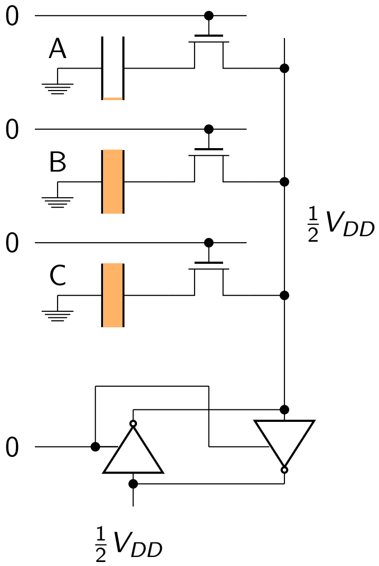
- Ambit-AND-OR relies on analog charge sharing.

- The final state of the bitline depends mostly on the deviation of the bitline after charge sharing

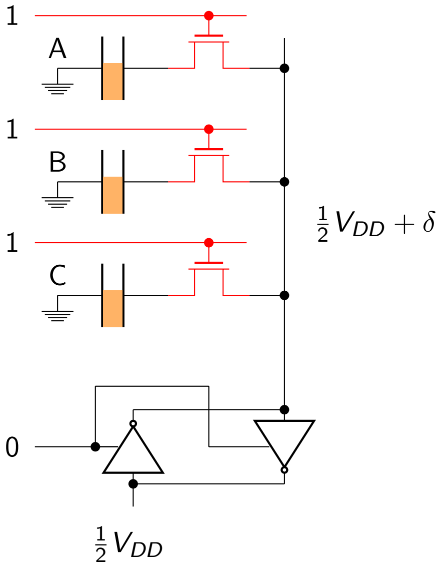
Ambit-AND-OR

- Ambit-AND-OR relies on analog charge sharing.
- The final state of the bitline depends mostly on the deviation of the bitline after charge sharing
- **Key observation**: By activating three rows at once, the bitwise majority of the three rows is computed

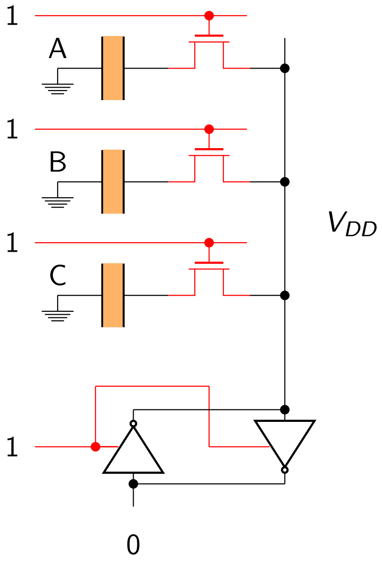
Initial State



Triple Row Activation (**TRA**)



After Sense Amplification



Ambit-AND-OR

C	A	B	Bitwise Majority
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Ambit-AND-OR

C	A	B	Bitwise Majority
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Ambit-AND-OR

C	A	B		Bitwise Majority
0	0	0		0
0	0	1	A AND B	0
0	1	0		0
0	1	1		1
1	0	0		0
1	0	1	A OR B	1
1	1	0		1
1	1	1		1

Problems with naïve TRA

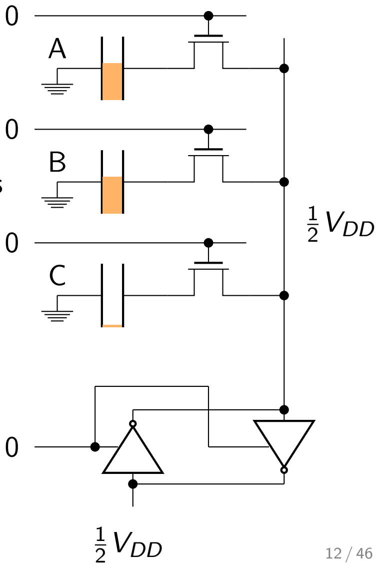
- Source Rows are overwritten during TRA

Problems with naïve TRA

- Source Rows are overwritten during TRA
- Capacitor discharging between refreshes could affect the correctness

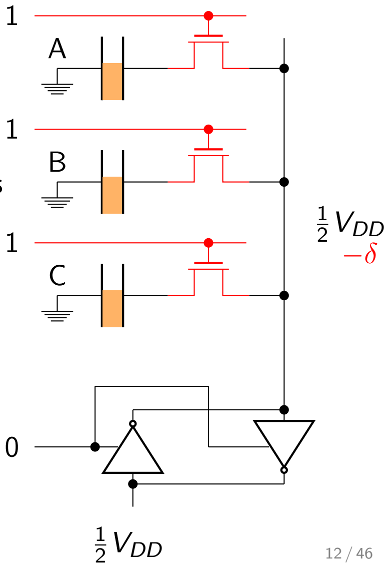
Problems with naïve TRA

- Source Rows are overwritten during TRA
- Capacitor discharging between refreshes could affect the correctness



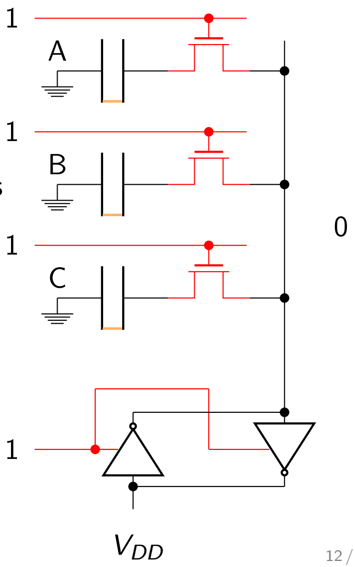
Problems with naïve TRA

- Source Rows are overwritten during TRA
- Capacitor discharging between refreshes could affect the correctness



Problems with naïve TRA

- Source Rows are overwritten during TRA
- Capacitor discharging between refreshes could affect the correctness



Problems with naïve TRA

- Source Rows are overwritten during TRA
- Capacitor discharging between refreshes could affect the correctness
- Solution: Use three designated rows T0, T1, T2
  - Copy source rows to two designated rows
  - Initialize the third row to all 0 or all 1
  - Do a TRA on the designated rows
  - Copy result to destination

Problems with naïve TRA

- Source Rows are overwritten during TRA
- Capacitor discharging between refreshes could affect the correctness
- Solution: Use three designated rows T0, T1, T2
  - Copy source rows to two designated rows
  - Initialize the third row to all 0 or all 1
  - Do a TRA on the designated rows
  - Copy result to destination
- Source rows are no longer directly involved in TRA

Problems with naïve TRA

- Source Rows are overwritten during TRA
- Capacitor discharging between refreshes could affect the correctness
- Solution: Use three designated rows T0, T1, T2
  - Copy source rows to two designated rows
  - Initialize the third row to all 0 or all 1
  - Do a TRA on the designated rows
  - Copy result to destination
- Source rows are no longer directly involved in TRA
- All rows in a TRA have been refreshed recently

Problems with naïve TRA

- A lot of **copying and initialization** is needed for designated rows

Problems with naïve TRA

- A lot of **copying and initialization** is needed for designated rows
- Ambit relies on **RowClone** for copying data and initializing rows

  Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry

  RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

Problems with naïve TRA

- Another problem is additional complexity in the memory bus and the address decoder

Problems with naïve TRA

- Another problem is additional complexity in the memory bus and the address decoder
- How can we transmit a command to activate three rows without sending and decoding 3 addresses?

Problems with naïve TRA

- Another problem is additional complexity in the memory bus and the address decoder
- How can we transmit a command to activate three rows without sending and decoding 3 addresses?
- The solution: We use a **reserved address** to communicate a TRA on the designated rows

Problems with naïve TRA

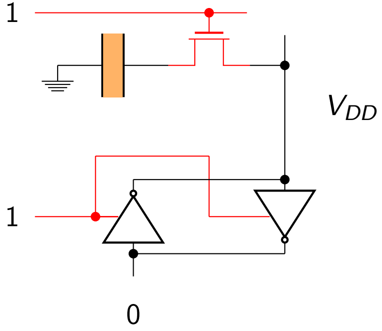
- Another problem is additional complexity in the memory bus and the address decoder
- How can we transmit a command to activate three rows without sending and decoding 3 addresses?
- The solution: We use a **reserved address** to communicate a TRA on the designated rows
- Also, we can split up the row decoder into two parts, which leads to a simpler design and better performance

Mechanism

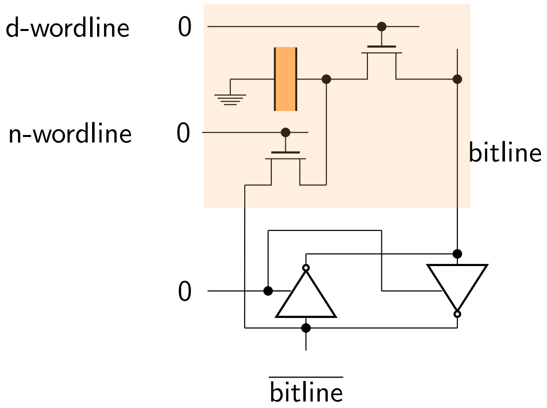
Ambit-NOT

Ambit-NOT

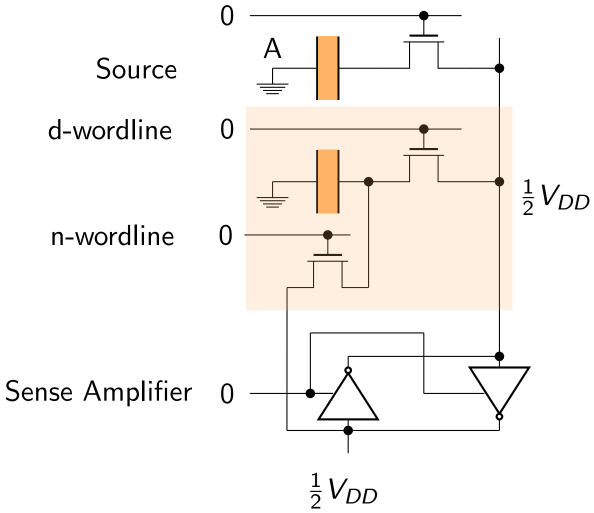
Notice that, during normal DRAM operation, the voltage level of $\overline{\text{bitline}}$ is the negation of the value in the cell.



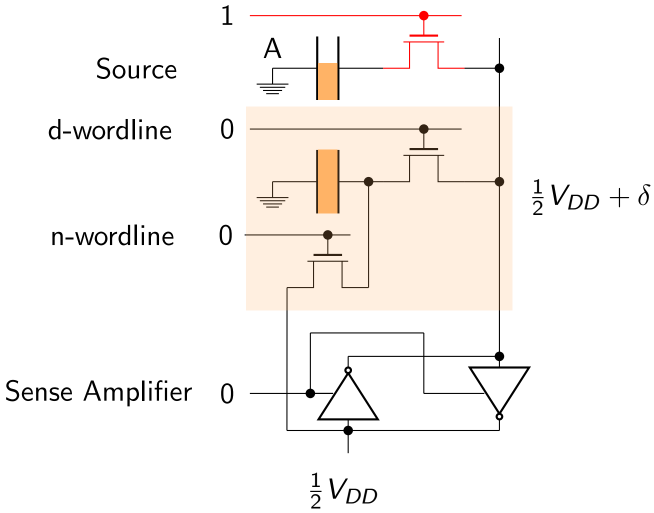
Dual Contact Cell (DCC)



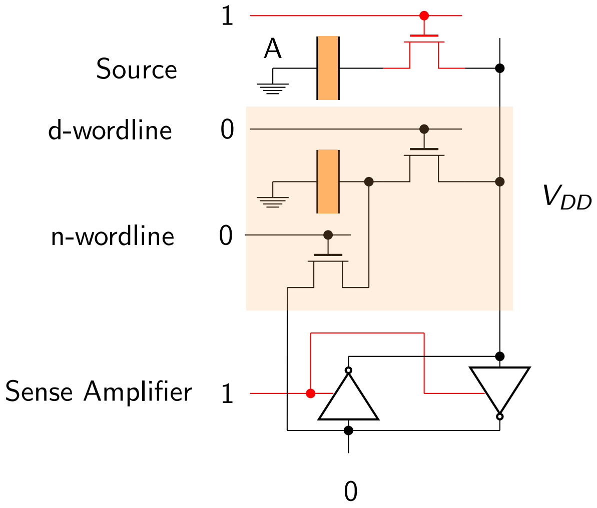
Initial State



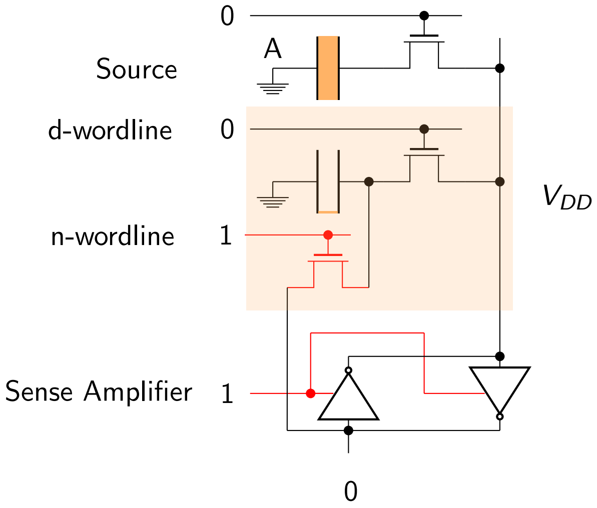
After Charge Sharing



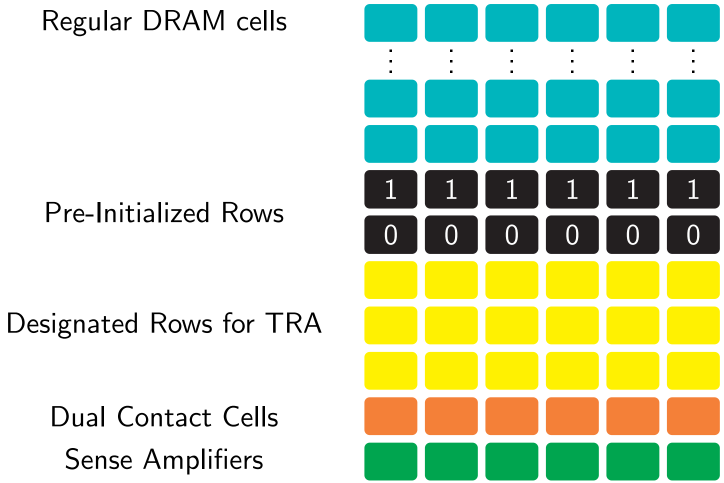
Activated Source Row



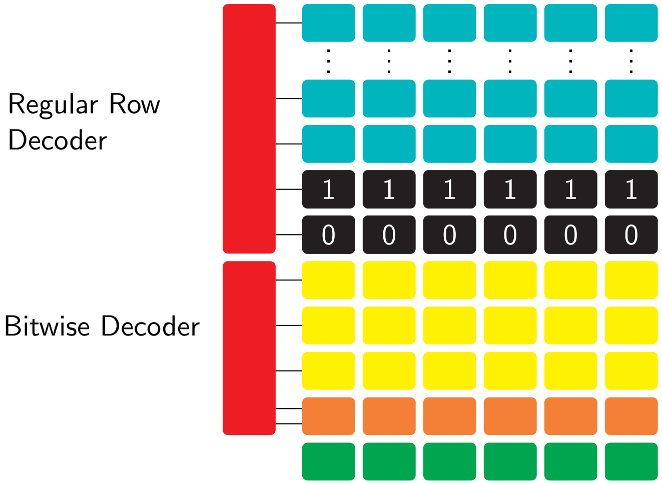
Activated n-Wordline



Implementation

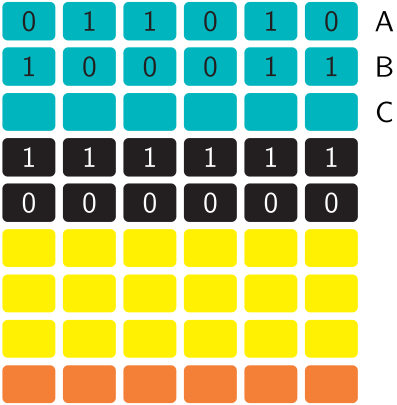


Implementation



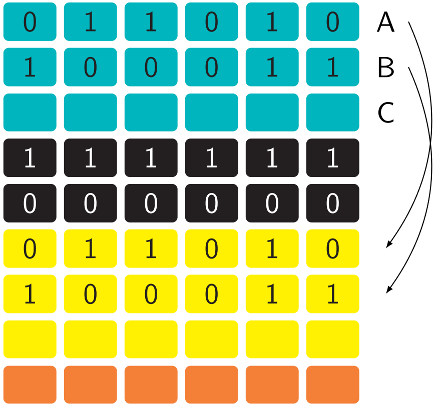
Computing C = A XOR B

- A XOR B = (A OR B) AND NOT (A AND B)



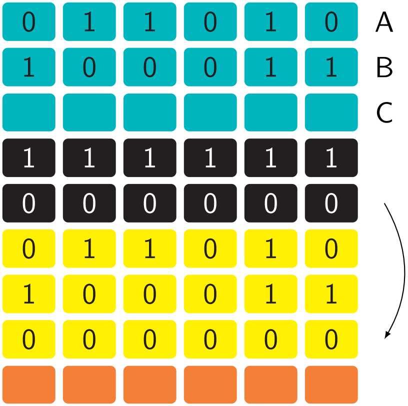
Computing C = A XOR B

- A XOR B = (A OR B) AND NOT (A AND B)
- Copy A to T0 and B to T1



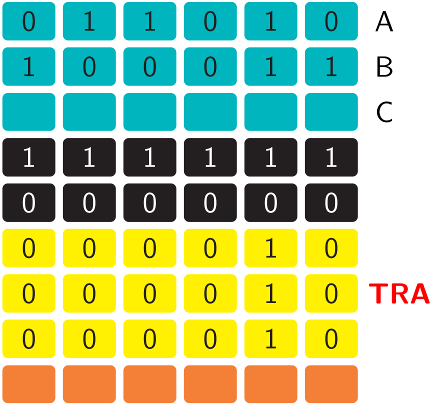
Computing C = A XOR B

- A XOR B = (A OR B) AND NOT (A AND B)
- Copy A to T0 and B to T1
- Initialize T2 to 0 to compute A AND B



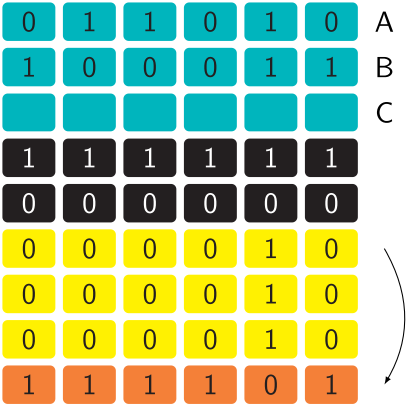
Computing C = A XOR B

- A XOR B = (A OR B) AND NOT (A AND B)
- Copy A to T0 and B to T1
- Initialize T2 to 0 to compute A AND B



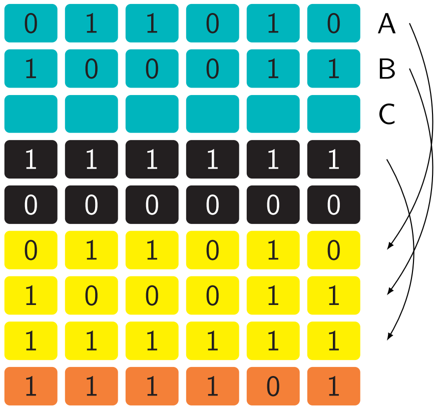
Computing C = A XOR B

- A XOR B = (A OR B) AND NOT (A AND B)
- Copy A to T0 and B to T1
- Initialize T2 to 0 to compute A AND B
- Store the negation of T0 in the DCC row



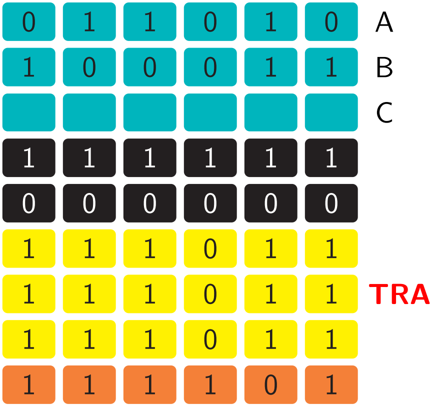
Computing C = A XOR B

- A XOR B = (A OR B) AND NOT (A AND B)
- Copy A to T0 and B to T1
- Initialize T2 to 0 to compute A AND B
- Store the negation of T0 in the DCC row
- Prepare T0, T1 and T2 to compute A OR B



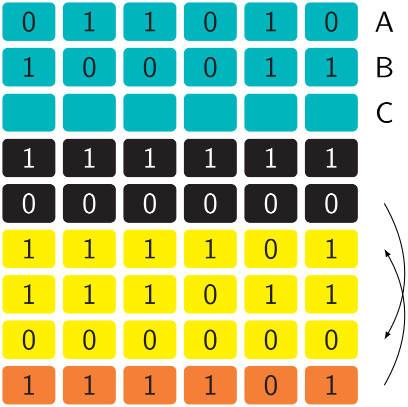
Computing C = A XOR B

- A XOR B = (A OR B) AND NOT (A AND B)
- Copy A to T0 and B to T1
- Initialize T2 to 0 to compute A AND B
- Store the negation of T0 in the DCC row
- Prepare T0, T1 and T2 to compute A OR B



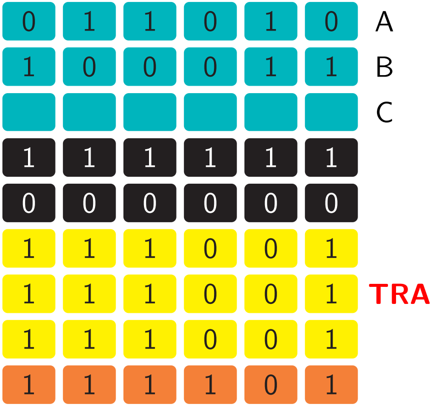
Computing C = A XOR B

- A XOR B = (A OR B) AND NOT (A AND B)
- Copy A to T0 and B to T1
- Initialize T2 to 0 to compute A AND B
- Store the negation of T0 in the DCC row
- Prepare T0, T1 and T2 to compute A OR B
- Copy the DCC row to T0 and set T2 to 0



Computing C = A XOR B

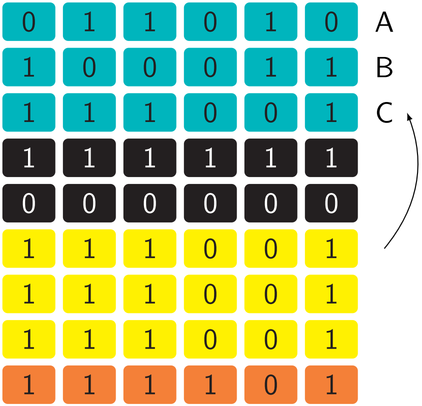
- A XOR B = (A OR B) AND NOT (A AND B)
- Copy A to T0 and B to T1
- Initialize T2 to 0 to compute A AND B
- Store the negation of T0 in the DCC row
- Prepare T0, T1 and T2 to compute A OR B
- Copy the DCC row to T0 and set T2 to 0
- Do a TRA and copy the result to C



0	1	1	0	1	0	A
1	0	0	0	1	1	B
						C
1	1	1	1	1	1	
0	0	0	0	0	0	
1	1	1	0	0	1	
1	1	1	0	0	1	**TRA**
1	1	1	0	0	1	
1	1	1	1	0	1	

Computing C = A XOR B

- A XOR B = (A OR B) AND NOT (A AND B)
- Copy A to T0 and B to T1
- Initialize T2 to 0 to compute A AND B
- Store the negation of T0 in the DCC row
- Prepare T0, T1 and T2 to compute A OR B
- Copy the DCC row to T0 and set T2 to 0
- Do a TRA and copy the result to C



Integrating Ambit with the System

- As a PCIe device
    - Simple approach, similar to other accelerators

Integrating Ambit with the System

- As a PCIe device
  - Simple approach, similar to other accelerators
- Directly plugged onto the memory bus
  - Makes sense since Ambit uses the same interface as regular DRAM
  - However, this requires additional support such as a new CPU instruction

Integrating Ambit with the System

Proposal: New CPU instruction

bb*op* dst src1 [src2] size

Integrating Ambit with the System

Proposal: New CPU instruction

bb*op* dst src1 [src2] size

- Size is required to be a multiple of the row size

Integrating Ambit with the System

Proposal: New CPU instruction

bb*op* dst src1 [src2] size

- Size is required to be a multiple of the row size
- Source and destination need to be row-aligned

Integrating Ambit with the System

Proposal: New CPU instruction

bb*op* dst src1 [src2] size

- Size is required to be a multiple of the row size
- Source and destination need to be row-aligned
- The CPU checks the constraints. If they are met, Ambit is used to complete the operation. Otherwise, the CPU does the operation normally.

Integrating Ambit with the System

Other system considerations:

Integrating Ambit with the System

Other system considerations:

- API/Driver support

Integrating Ambit with the System

Other system considerations:

- API/Driver support
  - Rows involved in bulk bitwise operations need to be in the same subarray so we can use **fast copying mechanisms**

Integrating Ambit with the System

Other system considerations:

- API/Driver support
  - Rows involved in bulk bitwise operations need to be in the same subarray so we can use **fast copying mechanisms**
  - Applications need a way to specify which parts of memory are likely to be involved in bulk bitwise operations

Integrating Ambit with the System

Other system considerations:

- API/Driver support
  - Rows involved in bulk bitwise operations need to be in the same subarray so we can use **fast copying mechanisms**
  - Applications need a way to specify which parts of memory are likely to be involved in bulk bitwise operations
- Cache coherence

Integrating Ambit with the System

Other system considerations:

- API/Driver support
  - Rows involved in bulk bitwise operations need to be in the same subarray so we can use **fast copying mechanisms**
  - Applications need a way to specify which parts of memory are likely to be involved in bulk bitwise operations
- Cache coherence
  - Ambit changes the contents of memory directly

Integrating Ambit with the System

Other system considerations:

- API/Driver support
  - Rows involved in bulk bitwise operations need to be in the same subarray so we can use **fast copying mechanisms**
  - Applications need a way to specify which parts of memory are likely to be involved in bulk bitwise operations
- Cache coherence
  - Ambit changes the contents of memory directly
  - Existing **DMA techniques** can be utilised

Integrating Ambit with the System

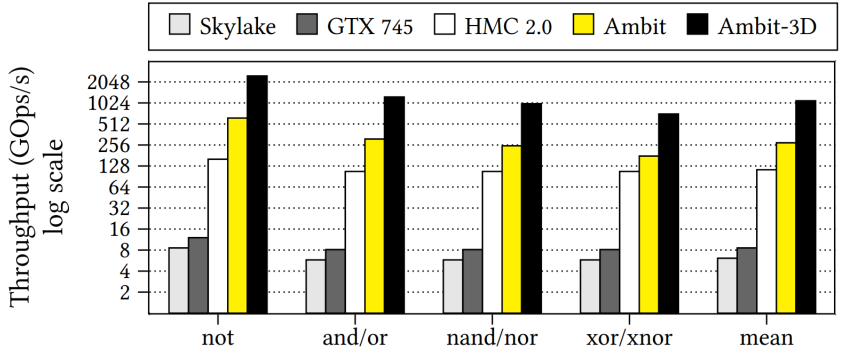
Other system considerations:

- API/Driver support
  - Rows involved in bulk bitwise operations need to be in the same subarray so we can use **fast copying mechanisms**
  - Applications need a way to specify which parts of memory are likely to be involved in bulk bitwise operations
- Cache coherence
  - Ambit changes the contents of memory directly
  - Existing **DMA techniques** can be utilised
  - Alternatively, the bb*op* instruction could implicitly manage the caches as well

Key Results: Methodology & Evaluation

Ambit Throughput & Energy

Microbenchmarks with 32MB input vectors



Ambit Throughput & Energy

- Energy is estimated for DDR3-1333 using the Rambus power model
- Energy numbers include only the DRAM and channel energy, and not the energy consumed by the processor

DRAM & Channel Energy (nJ/KB)

Design	not	and/or	nand/nor	xor/xnor
DDR3	93.7	137.9	137.9	137.9
Ambit	1.6	3.2	4.0	5.5
Energy reduction	**59.5x**	**43.9x**	**35.1x**	**25.1x**

Methodology

Evaluations were carried out using the Gem5 full-system simulator. Major simulation parameters:

Processor	x86, 8-wide, out-of-order, 4 Ghz 64-entry instruction queue
L1 cache	32 KB D-cache, 32 KB I-cache, LRU policy
L2 cache	2 MB, LRU policy, 64 B cache line size
Memory Controller	8 KB row size, FR-FCFS scheduling
Main memory	DDR4-2400, 1-channel, 1-rank, 16 banks

Workloads:

- Set Operations - Comparing to bitvectors and red-black trees
- Bitmap Indices
- Bitweaving - Column scans using bulk bitwise operations

Workload: Sets

- Sets with limited domain may be represented as bitvectors or trees

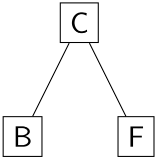
Workload: Sets

- Sets with limited domain may be represented as bitvectors or trees
- Example: Subsets of $\{A, B, C, D, E, F\}$

Workload: Sets

- Sets with limited domain may be represented as bitvectors or trees
- Example: Subsets of $\{A, B, C, D, E, F\}$
- The set $\{B, C, F\}$ may be represented by:

a tree

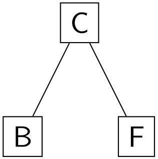


or a bitvector 011001

Workload: Sets

- Sets with limited domain may be represented as bitvectors or trees
- Example: Subsets of $\{A, B, C, D, E, F\}$
- The set $\{B, C, F\}$ may be represented by:

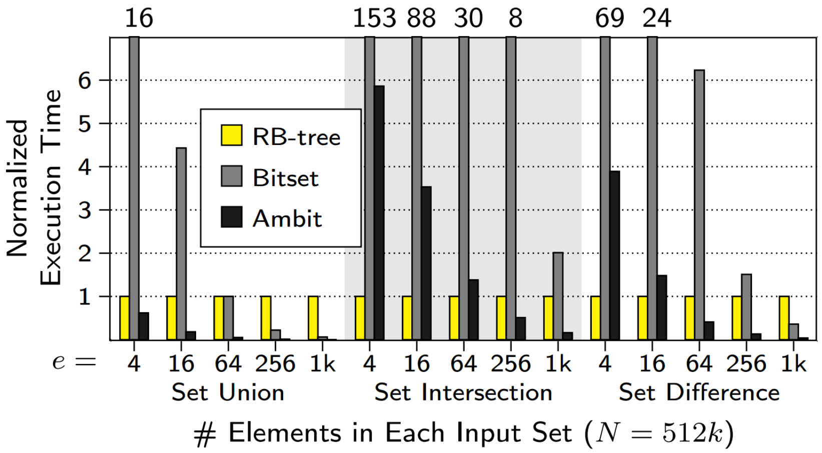
a tree



or a bitvector 011001

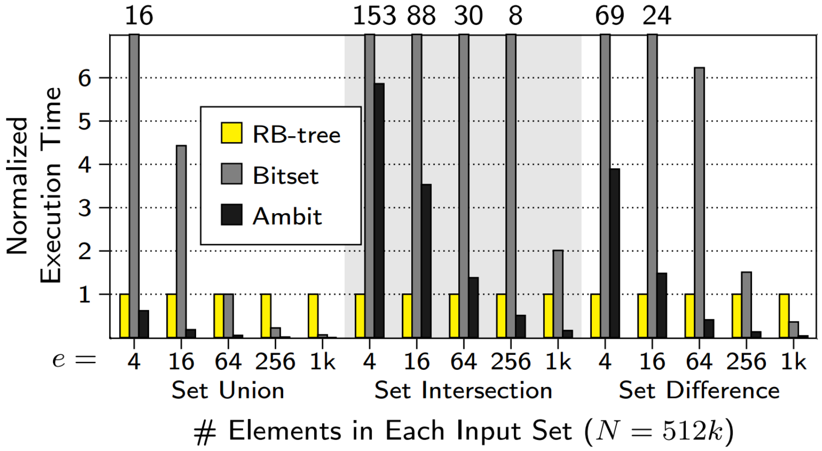
- Set operations on trees **scale with the number of elements in the set**, whereas bitvectors also have to **process elements that are not in the set**

Performance of Set Operations



Performance of Set Operations

- Ambit shifts the balance heavily in favor of bitvectors



Workload: Bitmap Index

- Using bitsets to build database indices

Workload: Bitmap Index

- Using bitsets to build database indices
- As an example, consider this table:

USER_ID	REGION	INCOME_LEVEL
101	east	bracket_1
102	central	bracket_1
103	west	bracket_2
104	east	bracket_2

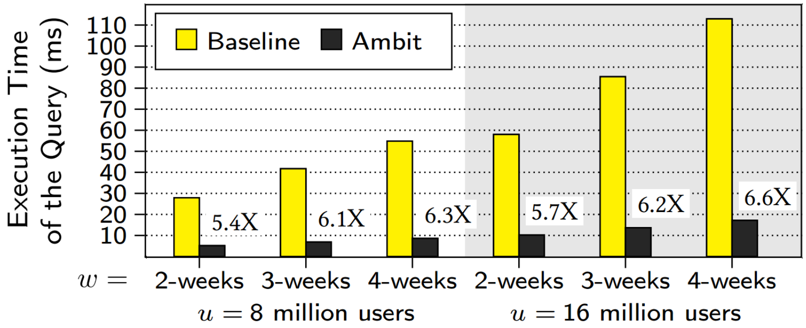
Workload: Bitmap Index

- A bitmap for the REGION and INCOME_LEVEL columns might look like this:

REGION			INCOME	
east	central	west	bracket_1	bracket_2
1	0	0	1	0
0	1	0	1	0
0	0	1	0	1
1	0	0	0	1

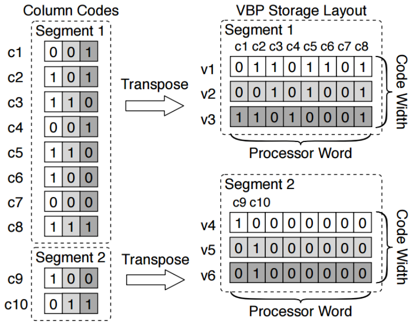
Bitmap Index Performance

Running an analytic query using a bitmap index.
Each query takes $\mathrm{O}(w)$ bulk bitwise operations, each of which takes $\mathrm{O}(u)$ time.



Workload: Bitweaving

- Used to speed up predicate evaluation in databases
- Tables are stored columnwise, but at the bitlevel

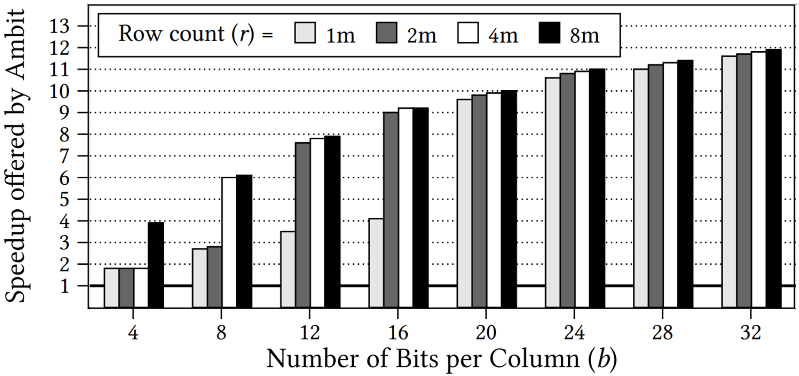


- For details on how it works, refer to **BitWeaving: Fast Scans for Main Memory Data Processing** by Yinan Li and Jignesh M. Patel

Bitweaving Performance

- Evaluated Query:

```
select count(*) from T where c1 <= val <= c2
```



Executive Summary

- **Problem: Bulk bitwise operations**
  - Problematic when used on large in-memory bitvectors
  - **Limited** by available memory bandwidth

Executive Summary

- **Problem: Bulk bitwise operations**
  - Problematic when used on large in-memory bitvectors
  - **Limited** by available memory bandwidth
- The proposal: Ambit
  - Perform bulk bitwise operation **in DRAM**
  - **Activate multiple DRAM rows** to compute AND/OR
  - Use **existing inverters** to compute NOT
  - AND, OR and NOT are sufficient to compute all bitwise operations (NAND,XOR,...)
  - **less than 1%** area overhead

Executive Summary

- **Problem: Bulk bitwise operations**
    - Problematic when used on large in-memory bitvectors
    - **Limited** by available memory bandwidth
- The proposal: Ambit
    - Perform bulk bitwise operation **in DRAM**
    - **Activate multiple DRAM rows** to compute AND/OR
    - Use **existing inverters** to compute NOT
    - AND, OR and NOT are sufficient to compute all bitwise operations (NAND,XOR,...)
    - **less than 1%** area overhead
- Results compared to state-of-the-art:
    - **32x** performance improvement, **35x** energy reduction averaged across 7 bulk bitwise operations
    - **3x-7x** performance improvement for real-world data intensive workloads

Strengths

- Addresses a problem with often used operations

Strengths

- Addresses a problem with often used operations
- Simple but novel mechanism

Strengths

- Addresses a problem with often used operations
- Simple but novel mechanism
- Paper discusses many low level- and implementation issues

Strengths

- Addresses a problem with often used operations
- Simple but novel mechanism
- Paper discusses many low level- and implementation issues
- Low hardware overhead

Strengths

- Addresses a problem with often used operations
- Simple but novel mechanism
- Paper discusses many low level- and implementation issues
- Low hardware overhead
- Well written

Strengths

- Addresses a problem with often used operations
- Simple but novel mechanism
- Paper discusses many low level- and implementation issues
- Low hardware overhead
- Well written
- 43 citations over the last year

Weaknesses

- Doesn't work well with ECC memory (ECC scheme must satisfy ECC(A *op* B) = ECC(A) *op* ECC(B)

Weaknesses

- Doesn't work well with ECC memory (ECC scheme must satisfy ECC(A *op* B) = ECC(A) *op* ECC(B)
- Vectors whose length is a multiple of the row size are a problem

Weaknesses

- Doesn't work well with ECC memory (ECC scheme must satisfy ECC(A *op* B) = ECC(A) *op* ECC(B)
- Vectors whose length is a multiple of the row size are a problem
- Multicore performance was not evaluated/considered

Weaknesses

- Doesn't work well with ECC memory (ECC scheme must satisfy ECC(A *op* B) = ECC(A) *op* ECC(B)
- Vectors whose length is a multiple of the row size are a problem
- Multicore performance was not evaluated/considered

  How are concurrent Ambit requests to the same subarray scheduled?

Weaknesses

- Doesn't work well with ECC memory (ECC scheme must satisfy ECC(A *op* B) = ECC(A) *op* ECC(B)
- Vectors whose length is a multiple of the row size are a problem
- Multicore performance was not evaluated/considered

  How are concurrent Ambit requests to the same subarray scheduled?

  How much slowdown occurs if a thread accesses data on a subarray which does a lot of bulk bitwise operations?

Key Takeaways

- A novel method to accelerate bulk bitwise operations

Key Takeaways

- A novel method to accelerate bulk bitwise operations
- Mechanism is simple but effective, low overhead

Key Takeaways

- A novel method to accelerate bulk bitwise operations
- Mechanism is simple but effective, low overhead
- Big savings in real world applications (**3x-7x** speedup)

Key Takeaways

- A novel method to accelerate bulk bitwise operations
- Mechanism is simple but effective, low overhead
- Big savings in real world applications (**3x-7x** speedup)
- Potential for further work

Related Work

- Add more processing capabilites to DRAM

Related Work

- Add more processing capabilites to DRAM
- **DRISA: A DRAM-based Reconfigurable In-Situ Accelerator** (2017)
  Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, Yuan Xie

Related Work

- Add more processing capabilites to DRAM
- **DRISA: A DRAM-based Reconfigurable In-Situ Accelerator** (2017)
  Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, Yuan Xie
    - Adds more mechanism for data movement, such as bit shifting

Related Work

- Add more processing capabilites to DRAM
- **DRISA: A DRAM-based Reconfigurable In-Situ Accelerator** (2017)
  Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, Yuan Xie
  - Adds more mechanism for data movement, such as bit shifting
- **DrAcc: a DRAM based Accelerator for Accurate CNN Inference** (2018)
  Quan Deng, Lei Jiang, Youtao Zhang, Minxuan Zhang, Jun Yang

Related Work

- Add more processing capabilites to DRAM
- **DRISA: A DRAM-based Reconfigurable In-Situ Accelerator** (2017)
  Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, Yuan Xie
  - Adds more mechanism for data movement, such as bit shifting
- **DrAcc: a DRAM based Accelerator for Accurate CNN Inference** (2018)
  Quan Deng, Lei Jiang, Youtao Zhang, Minxuan Zhang, Jun Yang
  - Builds an in-DRAM carry-lookahead adder to accelerate CNN Inference

Related Work

- Processing in other kinds of memory

Related Work

- Processing in other kinds of memory
- In the cache:

Related Work

- Processing in other kinds of memory
- In the cache:
  - **Compute Caches** (2017)
    Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, Reetuparna Das

Related Work

- Processing in other kinds of memory
- In the cache:
  - **Compute Caches** (2017)
    Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, Reetuparna Das
  - **Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks** (2018)
    Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaauw, Reetuparna Das

Related Work

- Processing in other kinds of memory
- In the cache:
  - **Compute Caches** (2017)
    Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, Reetuparna Das
  - **Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks** (2018)
    Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaauw, Reetuparna Das
- In non-volatile memory:

Related Work

- Processing in other kinds of memory
- In the cache:
  - **Compute Caches** (2017)
    Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, Reetuparna Das
  - **Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks** (2018)
    Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaauw, Reetuparna Das
- In non-volatile memory:
  - **Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories** (2016)
    Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, Yuan Xie

Questions?

Discussion Starters

How could we better support bitvectors whose length is not a multiple of the row size?

Masking

- A bitmask could be used to preserve part of a row

Masking

- A bitmask could be used to preserve part of a row
- Say we only want to do an operation on half rows

Masking

- A bitmask could be used to preserve part of a row
- Say we only want to do an operation on half rows
    - Initialize a row M with 00...011...1

Masking

- A bitmask could be used to preserve part of a row
- Say we only want to do an operation on half rows
  - Initialize a row M with 00...011...1
  - Compute C = ((A *op* B) AND NOT M) OR (M AND C)

Masking

- A bitmask could be used to preserve part of a row
- Say we only want to do an operation on half rows
  - Initialize a row M with 00...011...1
  - Compute C = ((A *op* B) AND NOT M) OR (M AND C)
- Creating a mask might be expensive, but they can be reused

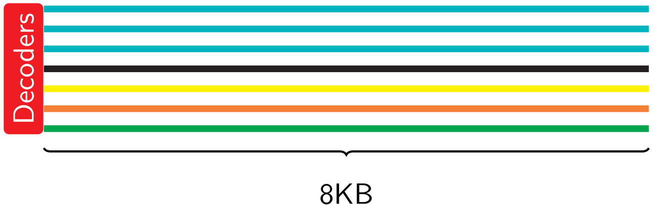
Masking

- A bitmask could be used to preserve part of a row
- Say we only want to do an operation on half rows
  - Initialize a row M with 00...011...1
  - Compute C = ((A *op* B) AND NOT M) OR (M AND C)
- Creating a mask might be expensive, but they can be reused
- DRISA proposes a shifting mechanism, which could speed up mask initialization

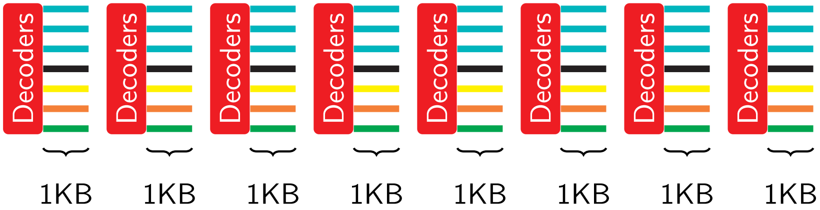
Masking

- A bitmask could be used to preserve part of a row
- Say we only want to do an operation on half rows
  - Initialize a row M with 00...011...1
  - Compute C = ((A *op* B) AND NOT M) OR (M AND C)
- Creating a mask might be expensive, but they can be reused
- DRISA proposes a shifting mechanism, which could speed up mask initialization
- But: Even if mask creation is cheap, it still requires **4 additional operations** to mask off a single useful operation

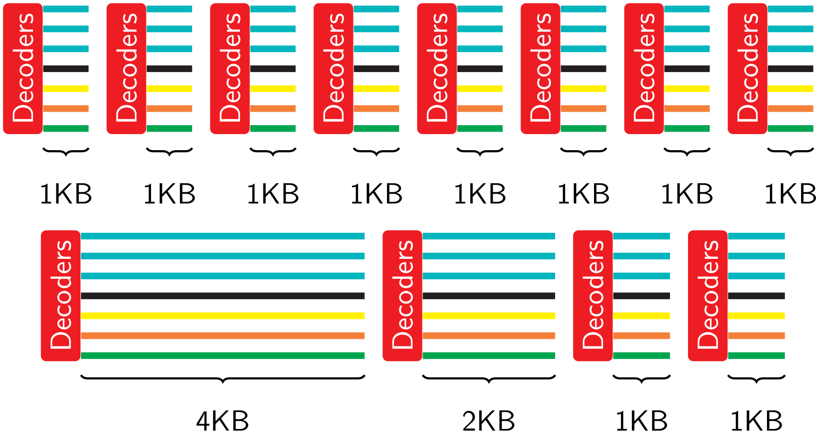
Add a few smaller subarrays



Add a few smaller subarrays



Add a few smaller subarrays



Discussion Starters

What kind of changes thoughout the system might be necessary to make Ambit useful for applications?

- The internal mapping of the DRAM needs to be exposed to the rest of the system

- The internal mapping of the DRAM needs to be exposed to the rest of the system
- Applications need a way to place bitvectors in the right subarrays

- The internal mapping of the DRAM needs to be exposed to the rest of the system
- Applications need a way to place bitvectors in the right subarrays
- Applications need to deal with vectors with bad lengths