CROW
A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability

Hasan Hassan
Minesh Patel  Jeremie S. Kim  A. Giray Yaglikci  Nandita Vijaykumar
Nika Mansouiri Ghiasri  Saugata Ghose  Onur Mutlu

ETH Zürich  Carnegie Mellon University
Summary
Summary

Challenges of DRAM scaling:

- High access latency → bottleneck for improving system performance/energy
Summary

Challenges of DRAM scaling:

• **High access latency** → bottleneck for improving system performance/energy
• **Refresh overhead** → reduces performance and consume high energy
Summary

Challenges of DRAM scaling:

• **High access latency** → bottleneck for improving system performance/energy
• **Refresh overhead** → reduces performance and consume high energy
• **Exposure to vulnerabilities** (e.g., RowHammer)
Summary

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

Copy-Row DRAM (CROW)
Summary

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

Copy-Row DRAM (CROW)

- Introduces *copy rows* into a subarray
Summary

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

**Copy-Row DRAM (CROW)**

- Introduces copy rows into a subarray
- The benefits of a copy row:
  - Efficiently duplicating data from regular row to a copy row
Summary

Challenges of DRAM scaling:

• High access latency → bottleneck for improving system performance/energy
• Refresh overhead → reduces performance and consume high energy
• Exposure to vulnerabilities (e.g., RowHammer)

Copy-Row DRAM (CROW)

• Introduces copy rows into a subarray
• The benefits of a copy row:
  • Efficiently duplicating data from regular row to a copy row
  • Quick access to a duplicated row
Summary

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

**Copy-Row DRAM (CROW)**

- Introduces *copy rows* into a subarray
- The benefits of a *copy row*:
  - Efficiently duplicating data from regular row to a *copy row*
  - Quick access to a duplicated row
  - Remapping a regular row to a *copy row*
Summary

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

**Copy-Row DRAM (CROW)**

- Introduces copy rows into a subarray
- The benefits of a copy row:
  - Efficiently duplicating data from regular row to a copy row
  - Quick access to a duplicated row
  - Remapping a regular row to a copy row

CROW is a flexible substrate with many use cases:

- CROW-cache & CROW-ref  **(20% speedup** and consumes **22% less DRAM energy)**
Summary

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

Copy-Row DRAM (CROW)

- Introduces *copy rows* into a subarray
- The benefits of a *copy row*:
  - Efficiently duplicating data from regular row to a *copy row*
  - Quick access to a duplicated row
  - Remapping a regular row to a *copy row*

CROW is a flexible substrate with many use cases:

- CROW-cache & CROW-ref  **(20% speedup and consumes 22% less DRAM energy)**
- Mitigating RowHammer
Summary

Challenges of DRAM scaling:
• **High access latency** → bottleneck for improving system performance/energy
• **Refresh overhead** → reduces performance and consume high energy
• **Exposure to vulnerabilities** (e.g., RowHammer)

Copy-Row DRAM (CROW)
• Introduces *copy rows* into a subarray
• The benefits of a *copy row*:
  • Efficiently duplicating data from regular row to a *copy row*
  • Quick access to a duplicated row
  • Remapping a regular row to a *copy row*

CROW is a flexible substrate with many use cases:
• CROW-cache & CROW-ref  *(20% speedup* and consumes *22% less DRAM energy)*
• Mitigating RowHammer
• We hope CROW enables many other use cases going forward
Summary

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

Copy-Row DRAM (CROW)

- Introduces *copy rows* into a subarray
- The benefits of a *copy row*:
  - Efficiently duplicating data from regular row to a *copy row*
  - Quick access to a duplicated row
  - Remapping a regular row to a *copy row*

CROW is a flexible substrate with many use cases:

- **CROW-cache & CROW-ref** *(20% speedup and consumes 22% less DRAM energy)*
- Mitigating RowHammer
- We hope CROW enables many other use cases going forward

Source code available in July: [github.com/CMU-SAFARI/CROW](http://github.com/CMU-SAFARI/CROW)
Outline

1. DRAM Operation Basics

2. The CROW Substrate
   - CROW-cache: Reducing DRAM Latency
   - CROW-ref: Reducing DRAM Refresh
   - Mitigating RowHammer

3. Evaluation

4. Conclusion
Outline

1. DRAM Operation Basics
2. The CROW Substrate
   - CROW-cache: Reducing DRAM Latency
   - CROW-ref: Reducing DRAM Refresh
   - Mitigating RowHammer
3. Evaluation
4. Conclusion
DRAM Organization

Memory Bus

Memory Controller

CPU
DRAM Organization

CPU

Memory Controller

Memory Bus

DRAM Subarray
DRAM Organization

Memory Bus

Memory Controller

CPU

DRAM Subarray

DRAM Cell
DRAM Organization

CPU

Memory Controller

Memory Bus

DRAM Subarray

DRAM Cell

Sense Amplifier
DRAM Organization

Memory Bus

Memory Controller

CPU

DRAM Subarray

DRAM Cell

DRAM Row

Sense Amplifier
Accessing DRAM

DRAM Subarray

DRAM Cell

DRAM Row

Sense Amplifier
Accessing DRAM

DRAM Subarray

DRAM Cell

DRAM Row

Sense Amplifier
Accessing DRAM

DRAM Subarray

Activate

DRAM Cell

DRAM Row

Sense Amplifier
Accessing DRAM

Activate

DRAM Subarray

DRAM Cell

DRAM Row

Sense Amplifier
Accessing DRAM

DRAM Subarray

Activate

DRAM Cell

DRAM Row

Sense Amplifier
Accessing DRAM

DRAM Subarray

Activate

Read

DRAM Cell

DRAM Row

Sense Amplifier
Accessing DRAM

- **Activate**
- **Precharge**
- **Read**

**DRAM Subarray**
- **DRAM Cell**
- **DRAM Row**
- **Sense Amplifier**
# Outline

1. DRAM Operation Basics

2. The CROW Substrate
   - CROW-cache: Reducing DRAM Latency
   - CROW-ref: Reducing DRAM Refresh
   - Mitigating RowHammer

3. Evaluation

4. Conclusion
Challenges of DRAM Scaling
Challenges of DRAM Scaling

1. access latency

DRAM
Challenges of DRAM Scaling

1. access latency
2. refresh overhead
Challenges of DRAM Scaling

1. access latency
2. refresh overhead
3. exposure to vulnerabilities
Our Goal

We want a substrate that enables the duplication and remapping of data within a subarray.
The Components of CROW
The Components of CROW
The Components of CROW

DRAM Subarray

regular rows

copy rows

regular row decoder

DRAM

CROW decoder

SA

SA

SA

SA

SA

SA
The Components of CROW

- **DRAM Subarray**
  - Regular rows
  - Copy rows

- **CROW-table**

- **Memory Controller**

---

SAFARI
CROW Operation 1: Row Copy

DRAM Subarray

regular rows

CROW decoder

copy rows

Memory Controller

SAFARI
CROW Operation 1: Row Copy
CROW Operation 1: Row Copy

DRAM Subarray

regular rows

CROW decoder

copy rows

DRAM

ACT-c (copy)

Memory Controller
CROW Operation 1: Row Copy

The diagram illustrates the process of CROW (Column Write On Row) Operation 1: Row Copy. It shows a DRAM Subarray with rows labeled as regular rows and copy rows. The Memory Controller initiates the ACT-c (copy) operation, which is highlighted in the diagram.
CROW Operation 1: Row Copy

DRAM Subarray

CROW decoder

regular rows

copy rows

DRAM

ACT-c (copy)

Memory Controller

SA
SA
SA
SA
SA
SA
Row Copy: Steps

source row:

destination row:

Sense Amplifier
Row Copy: Steps

1. Activation of the source row

source row: [Diagram showing a red circle] Destination row: [Diagram showing an open circle]

Sense Amplifier

SAFARI
Row Copy: Steps

source row:

destination row:

1. Activation of the source row

2. Charge sharing

Sense Amplifier
Row Copy: Steps

source row:

destination row:

1. Activation of the source row
2. Charge sharing
3. Beginning of restoration

Sense Amplifier
Row Copy: Steps

1. Activation of the source row
2. Charge sharing
3. Beginning of restoration
4. Activation of the destination row

source row:

destination row:

Sense Amplifier
Row Copy: Steps

**source row:**

1. Activation of the source row

2. Charge sharing

3. Beginning of restoration

**destination row:**

4. Activation of the destination row

5. Restoration of both rows to source data

Sense Amplifier
Row Copy: Steps

1. Activation of the source row
2. Charge sharing
3. Beginning of restoration

Enables quickly copying a regular row into a copy row

source row: destination row:

Enables quickly copying a regular row into a copy row

Amplifier
CROW Operation 2: Two-Row Activation

DRAM Subarray

regular rows

CROW decoder

copy rows

Memory Controller
CROW Operation 2: Two-Row Activation
CROW Operation 2: Two-Row Activation

DRAM Subarray

- regular rows
- copy rows

ACT-t (two row)

Memory Controller

SAFARI
Two-Row Activation: Steps

- Both charged or discharged

<table>
<thead>
<tr>
<th>Sense Amplifier</th>
</tr>
</thead>
</table>
Two-Row Activation: Steps

1. Activation of two rows

both charged or discharged
Two-Row Activation: Steps

both charged or discharged

1. Activation of two rows
2. Charge sharing
Two-Row Activation: Steps

1. Activation of two rows
2. Charge sharing \textbf{fast}

\textit{Sense Amplifier}

both charged or discharged
Two-Row Activation: Steps

1. Activation of two rows
2. Charge sharing  fast
3. Restoration

both charged or discharged

Sense Amplifier
Two-Row Activation: Steps

1. Activation of two rows
2. Charge sharing  fast

Enables fast access to data that is duplicated across a regular row and a copy row.
Outline

1. DRAM Operation Basics
2. The CROW Substrate
   - CROW-cache: Reducing DRAM Latency
   - CROW-ref: Reducing DRAM Refresh
   - Mitigating RowHammer
3. Evaluation
4. Conclusion
CROW-cache

**Problem:** High access latency
CROW-cache

**Problem:** High access latency

**Key idea:** Use copy rows to enable low-latency access to most-recently-activated regular rows in a subarray
CROW-cache

**Problem:** High access latency

**Key idea:** Use copy rows to enable low-latency access to most-recently-activated regular rows in a subarray

CROW-cache combines:

- **row copy** → copy a newly activated regular row into a copy row
- **two-row activation** → activate the regular row and copy row together on the next access
CROW-cache

**Problem:** High access latency

**Key idea:** Use copy rows to enable low-latency access to most-recently-activated regular rows in a subarray

CROW-cache combines:

- **row copy** → copy a newly activated regular row into a copy row
- **two-row activation** → activate the regular row and copy row together on the next access

Reduces activation latency by **38%**
CROW-cache Operation
CROW-cache Operation

DRAM Subarray

regular rows

copy rows
CROW-cache Operation

DRAM Subarray

regular rows

copy rows

CROW-table

Memory Controller
CROW-cache Operation

- Memory Controller
- DRAM Subarray
  - regular rows
  - copy rows
- CROW-table
- Request Queue
CROW-cache Operation

DRAM Subarray

regular rows

copy rows

CROW-table

Request Queue
load row X
CROW-cache Operation

DRAM Subarray

regular rows

copy rows

Request Queue
load row X

1 CROW-table miss
CROW-cache Operation

DRAM Subarray

Memory Controller

CROW-table

<table>
<thead>
<tr>
<th>copy row 0</th>
<th>row X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Request Queue

1. CROW-table miss
2. Allocate a copy row

load row X
CROW-cache Operation

Memory Controller

DRAM

ACT-c

DRAM Subarray

regular rows

copy rows

CROW-table

<table>
<thead>
<tr>
<th>copy row 0</th>
<th>row X</th>
</tr>
</thead>
</table>

Request Queue

load row X

1. CROW-table miss
2. Allocate a copy row
3. Issue ACT-c (copy)
CROW-cache Operation

Request Queue

- load row X
  - [bank conflict]

1. CROW-table miss
2. Allocate a copy row
3. Issue ACT-c (copy)

Memory Controller

DRAM Subarray

- regular rows
- copy rows

CROW-table

<table>
<thead>
<tr>
<th>copy row 0</th>
<th>row X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CROW decoder

copy rows

regular row

DRAM Subarray

regular row decoder
CROW-cache Operation

Request Queue

load row X

[bank conflict]
load row X

1. CROW-table miss
2. Allocate a copy row
3. Issue ACT-c (copy)
CROW-cache Operation

Request Queue

load row X

[bank conflict]

load row X

1. CROW-table miss
2. Allocate a copy row
3. Issue ACT-c (copy)

1. CROW-table hit
CROW-cache Operation

- **DRAM Subarray**
  - **regular rows**
  - **copy rows**

- **CROW-table**
  - copy row 0
  - row X

- **Request Queue**
  - load row X
  - [bank conflict]
  - load row X

1. CROW-table miss
2. Allocate a copy row
3. Issue ACT-c (copy)

1. CROW-table hit
2. Issue ACT-t (two row)
CROW-cache Operation

Request Queue
- load row X
  - [bank conflict]
  - load row X

1. CROW-table miss
2. Allocate a copy row

Second activation of row X is faster
Outline

1. DRAM Operation Basics
2. The CROW Substrate
   - CROW-cache: Reducing DRAM Latency
   - CROW-ref: Reducing DRAM Refresh
   - Mitigating RowHammer
3. Evaluation
4. Conclusion
Problem: Refresh has high overheads. Weak rows lead to high refresh rate

- weak row: at least one of the row’s cells cannot retain data correctly when refresh rate is decreased
CROW-ref

Problem: Refresh has high overheads. Weak rows lead to high refresh rate
  • weak row: at least one of the row’s cells cannot retain data correctly when refresh rate is decreased

Key idea: Safely reduce refresh rate by remapping a weak regular row to a strong copy row
CROW-ref

**Problem:** Refresh has high overheads. Weak rows lead to high refresh rate

- **weak row:** at least one of the row’s cells cannot retain data correctly when refresh rate is decreased

**Key idea:** Safely reduce refresh rate by remapping a weak regular row to a strong copy row

**CROW-ref uses:**

- **row copy** → copy a weak regular row to a strong copy row
CROW-ref

**Problem:** Refresh has high overheads. Weak rows lead to high refresh rate
  • **weak row:** at least one of the row’s cells cannot retain data correctly when refresh rate is decreased

**Key idea:** Safely reduce refresh rate by remapping a weak regular row to a strong copy row

CROW-ref uses:
  • **row copy** → copy a weak regular row to a strong copy row

CROW-ref **eliminates more than half of the refresh** requests
CROW-ref Operation
CROW-ref Operation

Retention Time Profiler

1. Perform retention time profiling
CROW-ref Operation

1. Perform retention time profiling
2. Remap weak rows to strong copy rows

Retention Time Profiler

strong
strong
weak
strong
strong
strong

SA
SA
SA
SA
SA
SA
CROW-ref Operation

1. Perform retention time profiling
2. Remap weak rows to strong copy rows
3. On ACT, check the CROW-table

Retention Time Profiler

strong
strong
weak
strong
strong
strong
**CROW-ref Operation**

1. Perform retention time profiling
2. Remap weak rows to strong copy rows
3. On ACT, check the CROW-table
4. If remapped, activate a copy row

**Retention Time Profiler**

- Weak
- Strong

Diagram showing the transition of retention time profiling with strong and weak rows.
CROW-ref Operation

1. Perform retention time profiling
2. Remap weak rows to strong copy rows
3. On ACT, check the CROW-table
4. If remapped, activate a copy row

How many weak rows exist in a DRAM chip?
Identifying Weak Rows

Weak cells are rare \cite{Liu+2013}
Identifying Weak Rows

Weak cells are rare [Liu+, ISCA'13]
weak cell: retention < 256ms
Identifying Weak Rows

Weak cells are rare \cite{Liu+13}

**weak cell: retention < 256ms**

\( \sim 1000/2^{38} \) (32 GiB) failing cells
Identifying Weak Rows

Weak cells are rare [Liu+, ISCA’13]
weak cell: retention < 256ms
~1000/2^{38} (32 GiB) failing cells

Probability

Weak rows in a subarray
Identifying Weak Rows

Weak cells are rare [Liu+, ISCA'13]

weak cell: retention < 256ms

~1000/2^{38} (32 GiB) failing cells

![Bar chart showing probability of weak rows in a subarray](image)
Identifying Weak Rows

Weak cells are rare [Liu+, ISCA’13]

weak cell: retention < 256ms

~1000/2^{38} (32 GiB) failing cells

DRAM Retention Time Profiler

• REAPER [Patel+, ISCA’17]
• PARBOR [Khan+, DSN’16]
• AVATAR [Qureshi+, DSN’15]
Identifying Weak Rows

Weak cells are rare [Liu+, ISCA’13]
weak cell: retention < 256ms
~1000/2^{38} (32 GiB) failing cells

DRAM Retention Time Profiler
- REAPER [Patel+, ISCA’17]
- PARBOR [Khan+, DSN’16]
- AVATAR [Qureshi+, DSN’15]
- At system boot or during runtime

Probability of weak rows in a subarray

SAFARI
Identifying Weak Rows

Weak cells are rare \([Liu+, ISCA'13]\)

weak cell: retention < 256ms

~1000/2^{38} (32 GiB) failing cells

DRAM Retention Time Profiler
- REAPER \([Patel+, ISCA'17]\)
- PARBOD \([Khan+, DSN'16]\)

A few copy rows are sufficient to halve the refresh rate
Outline

1. DRAM Operation Basics
2. The CROW Substrate
   - CROW-cache: Reducing DRAM Latency
   - CROW-ref: Reducing DRAM Refresh
   - Mitigating RowHammer
3. Evaluation
4. Conclusion
Mitigating RowHammer
Mitigating RowHammer
Mitigating RowHammer

activate

SA
SA
SA
SA
SA
SA
Mitigating RowHammer
Mitigating RowHammer

Key idea: remap victim rows to copy rows
Outline

1. DRAM Operation Basics
2. The CROW Substrate
   - CROW-cache: Reducing DRAM Latency
   - CROW-ref: Reducing DRAM Refresh
   - Mitigating RowHammer
3. Evaluation
4. Conclusion
Methodology

• Simulator
  • DRAM Simulator (Ramulator [Kim+, CAL’15])
    https://github.com/CMU-SAFAI/ramulator

Source code available in July:
github.com/CMU-SAFAI/CROW
Methodology

• **Simulator**
  - DRAM Simulator (Ramulator [Kim+, CAL’15])
    https://github.com/CMU-SAFARI/ramulator

• **Workloads**
  - 44 single-core workloads
    - SPEC CPU2006, TPC, STREAM, MediaBench
  - 160 multi-programmed four-core workloads
    - By randomly choosing from single-core workloads
  - Execute at least 200 million representative instructions per core

Source code available in July:
github.com/CMU-SAFARI/CROW
Methodology

• Simulator
  • DRAM Simulator (Ramulator [Kim+, CAL’15])
    
    https://github.com/CMU-SAFAI/ramulator

• Workloads
  • 44 single-core workloads
    • SPEC CPU2006, TPC, STREAM, MediaBench
  • 160 multi-programmed four-core workloads
    • By randomly choosing from single-core workloads
    • Execute at least 200 million representative instructions per core

• System Parameters
  • 1/4 core system with 8 MiB LLC
  • LPDDR4 main memory
  • 8 copy rows per 512-row subarray

Source code available in July: github.com/CMU-SAFAI/CROW
CROW-cache Results
CROW-cache Results

* with 8 copy rows and a 64Gb DRAM chip (sensitivity in paper)
CROW-cache Results

* with 8 copy rows and a 64Gb DRAM chip (sensitivity in paper)
CROW-cache Results

CROW-cache improves single-/four-core performance and energy

- **Speedup**
  - Single-core: 7.5%
  - Four-core: 7.1%

- **Normalized DRAM Energy**
  - Single-core: 8.2%
  - Four-core: 6.9%
CROW-ref Results
CROW-ref Results

* with 8 copy rows and a 64Gb DRAM chip (sensitivity in paper)
CROW-ref Results

* with 8 copy rows and a 64Gb DRAM chip (sensitivity in paper)
CROW-ref significantly reduces the performance and energy overhead of DRAM refresh.
Combining CROW-cache and CROW-ref
Combining CROW-cache and CROW-ref

- Speedup for different configurations:
  - CROW-(cache+ref)
  - Ideal CROW-cache + no refresh

- Graph comparison for single-core and four-core scenarios:
  - Single-core: 17%
  - Four-core: 20%
Combining CROW-cache and CROW-ref

<table>
<thead>
<tr>
<th>Speedup</th>
<th>CROW-(cache+ref)</th>
<th>Ideal CROW-cache + no refresh</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-core</td>
<td>17%</td>
<td>20%</td>
</tr>
<tr>
<td>four-core</td>
<td>17%</td>
<td>20%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Normalized DRAM Energy</th>
<th>CROW-(cache+ref)</th>
<th>Ideal CROW-cache + no refresh</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-core</td>
<td>23%</td>
<td>22%</td>
</tr>
<tr>
<td>four-core</td>
<td>23%</td>
<td>22%</td>
</tr>
</tbody>
</table>
Combining CROW-cache and CROW-ref

CROW-(cache+ref) provides more performance and DRAM energy benefits than each mechanism alone.
Hardware Overhead

For 8 copy rows and 16 GiB DRAM:

• 0.5% DRAM chip area
• 1.6% DRAM capacity
• 11.3 KiB memory controller storage
Hardware Overhead

For 8 copy rows and 16 GiB DRAM:

- 0.5% DRAM chip area
- 1.6% DRAM capacity
- 11.3 KiB memory controller storage

CROW is a low-cost substrate
Other Results in the Paper
Other Results in the Paper

• Performance and energy sensitivity to:
  • Number of copy-rows per subarray
  • DRAM chip density
  • Last-level cache capacity

• CROW-cache with prefetching

• CROW-cache compared to other in-DRAM caching mechanisms:
  • TL-DRAM [Lee+, HPCA’13]
  • SALP [Kim+, ISCA’12]
Outline

1. DRAM Operation Basics
2. The CROW Substrate
   - CROW-cache: Reducing DRAM Latency
   - CROW-ref: Reducing DRAM Refresh
   - Mitigating RowHammer
3. Evaluation
4. Conclusion
Conclusion

Source code available in July:
github.com/CMU-SAFAI/CROW
Conclusion

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

Source code available in July:
[github.com/CMU-SAFARI/CROW](github.com/CMU-SAFARI/CROW)
Conclusion

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

**Copy-Row DRAM (CROW)**

- Introduces copy rows into a subarray

Source code available in July: 
github.com/CMU-SAFARI/CROW
Conclusion

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

Copy-Row DRAM (CROW)

- Introduces **copy rows** into a subarray
- The benefits of a **copy row**:
  - Efficiently duplicating data from regular row to a **copy row**
  - Quick access to a duplicated row
  - Remapping a regular row to a **copy row**

Source code available in July: [github.com/CMU-SAFAI/CROW](github.com/CMU-SAFAI/CROW)
Conclusion

Challenges of DRAM scaling:
- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

Copy-Row DRAM (CROW)
- Introduces copy rows into a subarray
- The benefits of a copy row:
  - Efficiently duplicating data from regular row to a copy row
  - Quick access to a duplicated row
  - Remapping a regular row to a copy row

CROW is a flexible substrate with many use cases:
- CROW-cache & CROW-ref  *(20% speedup and consumes 22% less DRAM energy)*

Source code available in July: [github.com/CMU-SAFARI/CROW](https://github.com/CMU-SAFARI/CROW)
Conclusion

Challenges of DRAM scaling:

- **High access latency** → bottleneck for improving system performance/energy
- **Refresh overhead** → reduces performance and consume high energy
- **Exposure to vulnerabilities** (e.g., RowHammer)

**Copy-Row DRAM (CROW)**

- Introduces *copy rows* into a subarray
- The benefits of a *copy row*:
  - Efficiently duplicating data from regular row to a *copy row*
  - Quick access to a duplicated row
  - Remapping a regular row to a *copy row*

CROW is a flexible substrate with many use cases:

- CROW-cache & CROW-ref *(20% speedup and consumes 22% less DRAM energy)*
- Mitigating RowHammer

Source code available in July: [github.com/CMU-SAFARI/CROW]
Conclusion

Challenges of DRAM scaling:

• **High access latency** → bottleneck for improving system performance/energy
• **Refresh overhead** → reduces performance and consume high energy
• **Exposure to vulnerabilities** (e.g., RowHammer)

**Copy-Row DRAM (CROW)**

• Introduces **copy rows** into a subarray
• The benefits of a **copy row**:
  • Efficiently duplicating data from regular row to a **copy row**
  • Quick access to a duplicated row
  • Remapping a regular row to a **copy row**

CROW is a flexible substrate with many use cases:

• **CROW-cache & CROW-ref**  (**20% speedup** and consumes **22% less DRAM energy**)
• Mitigating RowHammer
• We hope CROW enables many other use cases going forward
CROW
A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability

Hasan Hassan
Minesh Patel  Jeremie S. Kim  A. Giray Yaglikci  Nandita Vijaykumar
Nika Mansouri Ghiasi  Saugata Ghose  Onur Mutlu

ETH Zürich
Carnegie Mellon University

SAFARI
Backup Slides
Latency Reduction with MRA

(a) tRCD (18 ns)

(b) tRAS (42 ns), tWR (18 ns), and restoration (24 ns)
Mitigating RowHammer
Mitigating RowHammer
Mitigating RowHammer

activate
Mitigating RowHammer
Mitigating RowHammer

Key idea: remap victim rows to copy rows
CROW-cache Performance
CROW-cache Performance

![Graph showing CROW-cache performance with speedup on the y-axis and various benchmarks like leslie3d, tpch2, zeus, lbm, mcf, stream-cp, libq, h264-dec, and AVERAGE (1-core) with four-core and single-core performance benchmarks.]

- Speedup values range from 0.90 to 1.20.
- Benchmarks include leslie3d, tpch2, zeus, lbm, mcf, stream-cp, libq, h264-dec, and AVERAGE (1-core).
- The graph compares performance across single-core and four-core configurations.
CROW-cache Performance

![Bar chart showing CROW-cache Performance across different benchmarks and core counts. The chart displays speedup for CROW-1 compared to Ideal CROW-cache (100% Hit Rate) with 6.6% and 0.7% improvements in single-core and four-core scenarios respectively. Benchmarks include leslie3d, tpch2, zeus, lbm, mcf, stream-cp, libq, h264-dec, and HHHH.](chart.png)
CROW-cache Performance

<table>
<thead>
<tr>
<th>Test</th>
<th>CROW-1</th>
<th>CROW-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leslie3d</td>
<td>1.10</td>
<td>1.12</td>
</tr>
<tr>
<td>tpc2</td>
<td>1.05</td>
<td>1.08</td>
</tr>
<tr>
<td>zeps</td>
<td>1.12</td>
<td>1.15</td>
</tr>
<tr>
<td>lbm</td>
<td>1.08</td>
<td>1.10</td>
</tr>
<tr>
<td>mcf</td>
<td>1.15</td>
<td>1.18</td>
</tr>
<tr>
<td>stream-cp</td>
<td>1.12</td>
<td>1.14</td>
</tr>
<tr>
<td>libq</td>
<td>1.07</td>
<td>1.10</td>
</tr>
<tr>
<td>h264-dec</td>
<td>1.10</td>
<td>1.12</td>
</tr>
<tr>
<td>AVERAGE (1-core)</td>
<td>1.13</td>
<td>1.15</td>
</tr>
<tr>
<td>HHHH</td>
<td>1.15</td>
<td>1.18</td>
</tr>
</tbody>
</table>

Speedup

- Single-core: 7.5%
- Four-core: 7.1%
CROW-cache Performance

- leslie3d
- tpch2
- zeus
- lbm
- mcf
- stream-cp
- libq
- h264-dec
- AVERAGE (1-core)
- HHHH

Speedup

CROW-1
CROW-64
CROW-8
CROW-128

single-core

four-core
CROW-cache Performance

- CROW-1
- CROW-64
- CROW-8
- CROW-128
- Ideal CROW-cache (100% Hit Rate)

<table>
<thead>
<tr>
<th>Application</th>
<th>single-core</th>
<th>four-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>leslie3d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tpch2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zeus</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lbm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mcf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stream-cp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>libq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h264-dec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVERAGE (1-core)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHHH</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Speedup**
CROW-ref Performance

Speedup vs. Bandwidth for single-core and four-core configurations.

- Bandwidth options: 8 Gbit, 16 Gbit, 32 Gbit, 64 Gbit

- Performance categories: mcf, milc, ibm, stream.cp, cactus, tpch17, leslie3d, jp2-enc, libq, zeus, AVERAGE, HHHH

- Chart shows speedup for different benchmarks and core counts.
CROW-ref Performance

<table>
<thead>
<tr>
<th></th>
<th>8 Gbit</th>
<th>16 Gbit</th>
<th>32 Gbit</th>
<th>64 Gbit</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcf</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>milc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ibm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stream_cp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cactus</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tpch17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>leslie3d</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jp2-enc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>libq</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zeus</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVERAGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHHH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Speedup

- single-core: 7.1%
- four-core: 11.9%
CROW-ref Energy Savings

Normalized DRAM Energy

- 8 Gbit
- 16 Gbit
- 32 Gbit
- 64 Gbit

single-core

four-core
CROW-ref Energy Savings

Normalized DRAM Energy

- 8 Gbit
- 16 Gbit
- 32 Gbit
- 64 Gbit

CROW-ref Energy Savings:
- 17.2% for single-core
- 7.8% for four-core
Speedup - CROW-cache

Single-core
Speedup - CROW-cache

Four-core
Energy – CROW-cache
Comparison to TL-DRAM and SALP

Normalized DRAM Energy vs. Speedup

Chip Area Overhead vs. Speedup
Slide on RLTL
Speedup – CROW-ref

![Graph showing speedup for different benchmarks and memory sizes (8 Gbit, 16 Gbit, 32 Gbit, 64 Gbit). The graph compares the speedup of mcf, milc, ibm, gems, cactus, tpcch17, sphinx3, leslie3d, soplex, jp2-enc, libq, wcount, jp2-dec, zeusmp, and 1-core HHHH with varying memory sizes.]
Energy – CROW-ref
CROW-cache + ref

(a) Single-core workloads

(b) Four-core workloads
CROW-table Organization

Figure 4: Organization of the CROW-table.
tRCD vs tRAS

Figure 6: Normalized tRCD latency as a function of normalized tRAS latency for different number of simultaneously activated DRAM rows.
MRA Area Overhead

Figure 7: Power consumption and area overhead of MRA.
DRAM Charge over Time

<table>
<thead>
<tr>
<th>charge</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 0</td>
<td></td>
</tr>
<tr>
<td>Data 1</td>
<td></td>
</tr>
</tbody>
</table>

- Cell
- Sense-Amplifier
DRAM Charge over Time

Cell

Sense Amplifier

Charge

Time

Ready to Access Charge Level

Data 1

Data 0

Sense-Amplifier
DRAM Charge over Time

Cell

Sense Amplifier

Charge Level

Ready to Access

Data 0

Data 1

time

Sensing

ACT
DRAM Charge over Time

Cell

Sense Amplifier

Data 0

Data 1

Ready to Access Charge Level

charge

Sensing

Restore

time

ACT

SAFARI
DRAM Charge over Time

- Cell
- Sense Amplifier
- Ready to Access
- Data 0
- Data 1
- Charge Level
- Sensing
- Restore
- ACT
- R/W

Ready to Access Charge Level
DRAM Charge over Time

- Cell
- Sense Amplifier
- Charge Level
- Ready to Access
- Data 0
- Data 1
- Sensing
- Restore
- tRCD
- ACT
- R/W

Graph showing the charge over time of a DRAM cell with sensing and restore processes.
DRAM Charge over Time

- Cell
- Sense Amplifier

Charge over time:
- Data 0
- Data 1

Events:
- Ready to Access
- Ready to Precharge
- Ready to Access Charge Level

Time milestones:
- Sensing
- Restore

Timing:
- ACT
- tRCD
- R/W
DRAM Charge over Time

- Cell
- Sense Amplifier
- Ready to Access
- Ready to Precharge
- Ready to Access Charge Level
- Charge Level
- Data 0
- Data 1
- Sensing
- Restore
- Precharge
- time
- ACT
- tRCD
- R/W
- PRE
DRAM Charge over Time

- **Cell**
- **Sense Amplifier**

- **Ready to Access**
- **Ready to Precharge**
- **Ready to Access Charge Level**

- **Data 0**
- **Data 1**

- **tRCD**
- **tRAS**

- **ACT**
- **R/W**
- **PRE**

Graph showing the charge level over time for DRAM operations.