Reorder Buffer: An Energy-Efficient Multithreading Architecture for Hardware MIMD Ray Traversal

SAMSUNG Advanced Institute of Technology

Won-Jong Lee, Youngsam Shin, Seok Joong Hwang, Seok Kang, Jeong-Joon Yoo, Soojung Ryu
Background: Mobile ray tracing H/W revisit

- Early desktop CPU/GPU (`00~09)
  - Packet tracing [Gunther 07][Overbeck 08][Benthin 09]

- HW Specialization (`02~06)
  - SarrCor [Schmitter `02], RPU, D-RPU [Woop `05, `06]
  - Not commercialized

- Modern GPUs and MICs (`10~)
  - OptiX [Steven `10], Embree [Wald `14]
  - Professional graphics

Mobile GPU and H/W revisit (`13~present)

- SGRT [Lee `13], GR6500 [McCombe `14], RayCore [Nah `14]
- Targeted for real-time applications (Game, UX, AR/VR)
Key features of modern ray tracing H/W are..

- MIMD Traversal Architecture ↔ Branch Divergence
  - Independent, single ray based, parallel processing
  - Better parallelism than SIMD for incoherent rays
    [Lee `13] [Kopta `10] [Nah `14] [Keely `14]

- Ray Scheduling & Multithreading ↔ Memory Divergence
  - Schedule the rays to increase memory locality
    [Aila `10] [Moon `11] [McCombe `14] [Kopta `14] [Keely `14]
  - Hide memory latencies caused by miss penalties
    [Nah `11] [Lee `13] [Kwon `13][Nah `14]
Background: Low Power Consumption

MIMD traversal pipeline consumes energy of 58.16% in the ray tracing logic (except DRAM).

- Baseline T&I unit (SGRT [Lee et al 2013]) ASIC ver.
- Gate-level simulation and evaluation with Synopsys PrimTime PX

Energy consumption ratio (%)

- Traversal: 58.16%
- Intersection: 27.19%
- L1 cache: 7.59%
- L2 cache: 6.26%
- Etc.: 0.81%
Traversals (or Intersection) pipeline

Consists of input buffer, cache, pipeline and feedback.
Classical problem: Cache miss causes pipeline stall
Previous hardware multithreading (latency hiding) for ray tracing

Ray Accumulation Unit (RAU) [Nah `11 – T&I Engne] [Lee `13 - SGRT]
- Prevents pipeline stall by storing rays that induce a cache miss to the buffer and processing other threads

Retry [Kwon `13][Nah `14 - RayCore]
- Invalidates cache-missed rays and feeds them to the hardware without any pipeline stalls being incurred
- A.K.A “Looping for next chance”
PROBLEM
The RAU and the Retry deal with rays inefficiently
Ray Accumulation Unit

Let’s make RAU

Input Buffer

CACHE

Traversal or Intersection Pipeline
Ray Accumulation Unit

We need a non-blocking cache to successive service

- A.K.A “Lockup-free”

```
Input Buffer
```

```
Non-Blocking CACHE
```

```
Pipeline latch
logic
Traversals or Intersection Pipeline
```

Input Buffer and non-blocking cache diagram with rays and data flow.
Ray Accumulation Unit

Accumulation buffer & control logic

- For storing the missed rays

Ray Accumulation Unit
- Control
- Buffer

Input Buffer

Non-Blocking CACHE

Pipeline latch

Traversal or Intersection Pipeline

rays

data
Ray Accumulation Unit

Buffer is configured in two dimensionally to group rays that have the same address.
When the cache miss is occurred..

stored in same row (0x1 (R0) == 0x1 (R3))
When the cache miss is complete....

the cache data is copied to the corresponding row of RAU and the row can be ready..
Problem #1: What if a row is full and the ray with the same address is arrived at input buffer?

Capacity miss even if there’s a room
Problem#2: RAU needs a relatively big SRAM buffer (32-64KB)

Ray payloads (org, dir,..), data, address.. etc

- Ray Accumulation Unit
- Control
- Buffer

- Input Buffer

- Non-Blocking CACHE

- Pipeline latch
- logic

- Traversal or Intersection Pipeline

- Cache hit ray
- Cache missed ray
Let’s talk about alternative! – Retry
Retry method

- No needs for additional buffer
- Utilize existing resources
  - NB cache
  - Feedback
  - Bypassing logic
When the cache miss is occurred...

- The ray is just invalidated and fed to the pipeline without stall.
- Does nothing in the pipeline

\[(R0, 0x1, \text{False})\]
When the cache-missed-ray arrives to the end of the pipeline...

- Retry to access cache!

![Diagram showing the pipeline and cache interaction](image-url)
Problem: Bypassing causes higher energy consumption by R2R transfer and switching

Longer DRAM latency
→ more bypassing iteration
→ more cache re-access
→ more energy consumption

Input Buffer

Pipeline latch

Traversal or Intersection Pipeline

(R0, 0x1, True)

Cache hit ray

Cache missed ray

Non-Blocking CACHE

Retry
Goal & Approach

Goal: Efficient hardware multithreading for MIMD traversal with minimal cost and energy consumption

Approach:
- Eliminate dedicated buffers and avoid bypassing
- Utilize existing buffer with minimal modification
Now, we propose a new method to resolve previous problems
Eliminate dedicated buffers

Ray Accumulation Unit
- Control
- Buffer

Input Buffer

Non-Blocking CACHE

Traversals or Intersection Pipeline

Pipeline latch

logic

rays

data

rays
Avoid bypassing

- Input Buffer
- Non-Blocking CACHE
- Pipeline latch
- Logic
- Traversal or Intersection Pipeline
Approach

How can we retaining the cache missed ray without any further resources, bypassing and pipeline stall

→ Input Buffer with small extension
  : Valid (1bit) + Ready (1bit) + Address (26bits)
Reorder Buffer - Configuration

Utilize input buffer for latency hiding

- valid
- ready
- addr
- rays

Reorder Buffer

Non-Blocking CACHE

Pipeline latch

logic

Traversal or Intersection Pipeline

data
Reorder Buffer - Configuration

- Type of ray, 1: newly arrived & not yet accessed the cache
- 0: accessed the cache and missed

Non-Blocking CACHE

Pipeline latch

Traversal or Intersection Pipeline

valid  ready  addr  rays

Reorder Buffer
Reorder Buffer - Configuration

Ray is ready? 0: cache missed and waiting for miss complete

1: cache miss is complete, so the ray is ready to go

valid  ready  addr  rays

Non-Blocking CACHE

Traversals or Intersection Pipeline

Ray is ready? 0: cache missed and waiting for miss complete

1: cache miss is complete, so the ray is ready to go
Reorder Buffer - Configuration

- Cache address references data, and used for searching

Table:

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>addr</th>
<th>rays</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- Reorder Buffer
- Non-Blocking CACHE
- Pipeline latch
- Traversal or Intersection Pipeline
- Logic
Reorder Buffer

**Example!**

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>addr</th>
<th>rays</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
(R3, 0x4)
(R2, 0x3)
(R1, 0x2)
(R0, 0x1)
```

![Diagram of Reorder Buffer](image)

Non-Blocking CACHE

Traversal or Intersection Pipeline
Reorder Buffer

Example!

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>addr</th>
<th>rays</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(R3, 0x4)
(R2, 0x3)
(R1, 0x2)
(R0, 0x1)

Non-Blocking CACHE

Traversal or Intersection Pipeline
R0 enters the buffer...

- Set valid $\leftarrow 1$, ready $\leftarrow$ null

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>addr</th>
<th>rays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>0x1</td>
<td>R0</td>
</tr>
</tbody>
</table>

(R3, 0x4)
(R2, 0x3)
(R1, 0x2)

Traversal or Intersection Pipeline

Non-Blocking CACHE
R0 requesting data @ 0x1

R1 enters the buffer

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>addr</th>
<th>rays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>0x2</td>
<td>R1</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0x1</td>
<td>R0</td>
</tr>
</tbody>
</table>

(R3, 0x4)
(R2, 0x3)
R0 missed the cache, so it is retained

- Set R0’s valid ← 0, ready ← 0
- R1 requesting data@0x2..

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>addr</th>
<th>rays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>0x3</td>
<td>R2</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0x2</td>
<td>R1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x1</td>
<td>R0</td>
</tr>
</tbody>
</table>

R0 Retained

Non-Blocking CACHE

Traversal or Intersection Pipeline

Pipeline latch

logic

request

R0 Miss

(R3, 0x4)

Request
R1 missed the cache and is retained

- Set R1’s valid ← 0, ready ← 0
- R2 requesting data@0x3..

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>addr</th>
<th>rays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>0x4</td>
<td>R3</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0x3</td>
<td>R2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x2</td>
<td>R1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x1</td>
<td>R0</td>
</tr>
</tbody>
</table>

R1 Retained

Reorder Buffer

Non-Blocking CACHE

Request

Traversal or Intersection Pipeline

Pipeline latch
R2 hits the cache, so immediately dispatched

- R3 requesting data@0x4..

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>addr</th>
<th>rays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0x4</td>
<td>R3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x2</td>
<td>R1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x1</td>
<td>R0</td>
</tr>
</tbody>
</table>

R2 Dispatched

Traversal or Intersection Pipeline

Non-Blocking CACHE

R2 Hit

Request

R2 Dispatched

(R2 + cache data)
Optimization

- When do the retained rays re-access the cache?
- How we prioritize the rays to fetch from buffer - the retained rays or the newly arrived rays?

→ Valid & Ready bits
Selection Priority

- Check the valid & ready bits:

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>addr</th>
<th>rays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>0x6</td>
<td>R11</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0xA</td>
<td>R10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x1</td>
<td>R4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x2</td>
<td>R1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x1</td>
<td>R0</td>
</tr>
</tbody>
</table>

Input Buffer

Non-Blocking CACHE

Traversed or Intersection Pipeline
Selection Priority

Check the valid & ready bits:

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>addr</th>
<th>rays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>0x6</td>
<td>R11</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0xA</td>
<td>R10</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0x1</td>
<td>R4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x2</td>
<td>R1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0x1</td>
<td>R0</td>
</tr>
</tbody>
</table>

Non-Blocking CACHE

Arrive

data

Input Buffer

cache event at 0x1 (miss complete)

Selection Priority

1. Invalid & Ready

: Data has been arrived to the cache

Traversal or Intersection Pipeline
Selection Priority

- Check the valid & ready bits:

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>addr</th>
<th>rays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>0x6</td>
<td>R11</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0xA</td>
<td>R10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x1</td>
<td>R4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x2</td>
<td>R1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x1</td>
<td>R0</td>
</tr>
</tbody>
</table>

Selection Priority

1. Invalid & Ready
   - Data has been arrived at the cache

2. Valid - New ray might hit the cache
What if the address of the newly arrived ray would be the same with the one of the retained rays?

→ The corresponding data was already requested and under delivery from external memory.

→ Do not re-access the cache!
### Redundancy control

R12 is newly arrived and referencing data@0x5, and R4 has the same address.

#### Memory Table

<table>
<thead>
<tr>
<th>Valid</th>
<th>Ready</th>
<th>Addr</th>
<th>Rays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>0x5</td>
<td>R12</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0x6</td>
<td>R11</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0xA</td>
<td>R10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x5</td>
<td>R4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x2</td>
<td>R1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x1</td>
<td>R0</td>
</tr>
</tbody>
</table>

#### Diagram

- **Reorder Buffer**
- **Non-Blocking CACHE**
- **Traversal or Intersection Pipeline**
- **Pipeline latch**

The diagram illustrates the flow of data through the reorder buffer and the pipeline latch, indicating the traversal or intersection process.
Redundancy control

Update valid bit & counter to avoid unnecessary cache access and increase ray coherency (R4, R12)

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>addr</th>
<th>rays</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0x5</td>
<td>R12</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0x6</td>
<td>R11</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0xA</td>
<td>R10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x5</td>
<td>R4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x2</td>
<td>R1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x1</td>
<td>R0</td>
</tr>
</tbody>
</table>

(R12, 0x5)
Advantages

- Cost-effective and energy-efficient
  - Minimal extension of existing H/W
  - No bypass

- Reordering* in input buffer
  - Multithreading
  - Increase ray coherency
EVALUATION
Experimental setup – baseline architecture

T&I Units, SGRT [Lee et al 2013] without any multithreading.
Experimental setup – test scenes
All scenes are rendered by diffused path tracing
Experimental Setup - Simulation

Comparison: different multithreading scheme..

- Baseline, RAU, Retry, Reorder

Cycle accurate, performance simulation

- Based on SGRT [Lee et al 2013]

Energy simulation of memory elements

- CACTI 6.5 [Muralimanohar 2007]

Configuration of MIMD Traversal H/W

: Clock (500MHz), Latency L1(1) / L2(20) / DRAM (10, 100, 200, 300)

All scenes are rendered by diffused path tracing
**Pipeline utilization** with varying the DRAM latency

- **Execution**: effective running time in the pipeline
- **Retry**: bypass time of the invalidated ray in the pipeline
- **Stall**: stall time of the pipeline

![Breakdown of the cycles spent](image)

<table>
<thead>
<tr>
<th>DRAM Latency &amp; Multithreading Architecture</th>
<th>Baseline</th>
<th>RAU</th>
<th>Retry</th>
<th>Reorder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakdown of the cycles spent</td>
<td>10</td>
<td>100</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>100</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>100</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>100</td>
<td>200</td>
<td>300</td>
</tr>
</tbody>
</table>
At latency 10: RAU, Retry, and Reorder recorded similar utilization level

![Diagram showing breakdown of cycles spent at latency 10 for Baseline, RAU, Retry, and Reorder.](image)

- **Baseline**: 47.2% Execution, 52.8% Stall
- **RAU**: 69.7% Execution, 30.3% Stall
- **Retry**: 72.5% Execution, 18.4% Stall
- **Reorder**: 77.0% Execution, 23.0% Stall

**DRAM Latency & Multithreading Architecture**
At latency 10: RAU, Retry, and Reorder recorded similar utilization level.

But, RAU stalls pipeline a bit more, and Retry consumes the time for pipeline bypassing.

Breakdown of the cycles spent:

- Baseline: 52.8% Execution, 47.2% Stall
- RAU: 69.7% Execution, 30.3% Stall
- Retry: 72.5% Execution, 18.4% Stall
- Reorder: 77.0% Execution, 23.0% Stall
At latency 100: Retry, and Reorder recorded similar utilization level

- RAU stalls pipeline more, and Retry retries a bit more
At latency 200: Retry, and Reorder recorded similar utilization level

RAU spent more time for stall than execution and Retry retries caching a bit more
At latency 300: RAU spent most stall time and Retry spent the 21% of the time for pipeline iteration

- Stall time $\rightarrow$ lower performance
- Retry time $\rightarrow$ higher energy consumption

![Diagram showing breakdown of cycles spent]
As the DRAM latency is increased, RAU drops sharply and Retry consumes more retry times.

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>RAU</th>
<th>Retry</th>
<th>Reorder</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execution</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>47.2</td>
<td>52.8</td>
<td>69.7</td>
<td>30.3</td>
</tr>
<tr>
<td>100</td>
<td>27.7</td>
<td>72.3</td>
<td>52.1</td>
<td>30.3</td>
</tr>
<tr>
<td>200</td>
<td>18.1</td>
<td>81.9</td>
<td>63.4</td>
<td>30.3</td>
</tr>
<tr>
<td>300</td>
<td>13.3</td>
<td>86.7</td>
<td>72.6</td>
<td>30.3</td>
</tr>
<tr>
<td><strong>Retry</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9.1</td>
<td>18.4</td>
<td>65.2</td>
<td>35.2</td>
</tr>
<tr>
<td>100</td>
<td>13.4</td>
<td>21.4</td>
<td>54.0</td>
<td>43.8</td>
</tr>
<tr>
<td>200</td>
<td>22.1</td>
<td>22.1</td>
<td>27.4</td>
<td>44.8</td>
</tr>
<tr>
<td>300</td>
<td>21.1</td>
<td>21.1</td>
<td>27.4</td>
<td>44.8</td>
</tr>
<tr>
<td><strong>Stall</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>30.3</td>
<td>30.3</td>
<td>22.1</td>
<td>44.8</td>
</tr>
<tr>
<td>100</td>
<td>47.9</td>
<td>47.9</td>
<td>22.1</td>
<td>44.8</td>
</tr>
<tr>
<td>200</td>
<td>63.4</td>
<td>63.4</td>
<td>22.1</td>
<td>44.8</td>
</tr>
<tr>
<td>300</td>
<td>72.6</td>
<td>72.6</td>
<td>22.1</td>
<td>44.8</td>
</tr>
</tbody>
</table>

**DRAM Latency & Multithreading Architecture**
In high complex scene, retry times are increased sharply (latency is more than 200)

- Longer DRAM latency caused to IST bottleneck
  - increasing TRV retries

<table>
<thead>
<tr>
<th>Breakdown of the cycles spent</th>
<th>Baseline (10, 100, 200, 300)</th>
<th>RAU (10, 100, 200, 300)</th>
<th>Retry (10, 100, 200, 300)</th>
<th>Reorder (10, 100, 200, 300)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution</td>
<td>10.6, 5.0, 7.5, 7.0</td>
<td>20.1, 13.5, 6.3, 4.4</td>
<td>21.0, 16.2, 17.3, 16.3</td>
<td>21.2, 16.2, 17.9, 16.9</td>
</tr>
<tr>
<td>Retry</td>
<td>89.4, 95.0, 92.5, 93.0</td>
<td>79.9, 86.5, 93.7, 95.6</td>
<td>66.9, 68.0, 72.2, 77.8</td>
<td>83.8, 82.1, 83.1, 83.1</td>
</tr>
</tbody>
</table>

**DRAM Latency & Multithreading Architecture**
**Performance with varying the DRAM latency**

- Reorder Buffer slightly outperforms Retry due to the better cache utilization with ray-reordering (1~5%)
- RAU drops sharply by lower cache & pipeline utilization

![Graph showing performance with varying DRAM latency](image)

- Baseline
- RAU
- Retry
- Reorder
**On-chip memory energy** (registers, buffer, SRAM, L1/L2 cache) consumption with varying the DRAM latency

Retry records the highest consumption because the bypass makes the number of accesses to the pipeline register and the SRAM buffer increases.

![Graph showing energy consumption with varying DRAM latency](image-url)
**Off-chip memory energy** (DRAM) consumption with varying the DRAM latency

- It is purely proportional to the miss ratio of the L2 cache.
- Reorder Buffer could reduce the power consumption up to 27.5% by the better cache utilization (vs. RAU)

![Graph](image.png)

*Off-chip Energy Consumption (Joules)*

*DRAM Latency*
**On-chip memory energy** consumption with varying the DRAM latency (complex scene)

- Retry ratio sharply increases in more than 200 cycles, which causes 3.4 times more energy to be consumed when the latency moves from 100 to 200.

![On-chip Memory Energy Consumption Graph](image)

- **On-chip Memory Energy Consumption (Joules)**
  - Baseline
  - RAU
  - Retry
  - Reorder

- **DRAM Latency**
  - 10
  - 100
  - 200
  - 300
It is the relative number to the baseline arch.
Relative Performance / Energy

Retry vs. Reorder...

Relative Performance / Energy (MRPS/Joules)

DRAM Latency

Baseline
RAU
Retry
Reorder
Reorder Buffer achieves up to 1.52x better efficiency.

Retry records a similar performance with the Reorder Buffer, but, its bypassing feature consumes much more on-chip energy.
Relative Performance / Energy

RAU vs. Reorder...

Baseline | RAU | Retry | Reorder
--- | --- | --- | ---

Relative Performance / Energy (MRPS/Joules)

DRAM Latency

0 10 100 200 300
Reorder achieves up to 4.7x better efficiency.

RAU records much lower throughputs with more pipeline stalls. Further, it consumes slightly more off-chip energy for lower cache locality.
SUMMARY
Reorder Buffer

- New hardware multithreading for MIMD ray traversal with minimal cost and energy consumption. (Does not need additional buffer and pipeline bypassing)

- Reschedule the order of the input rays in buffer to latency hiding and increase ray coherency (based on the cache hit/miss)
  → Up to 11.7% better cache utilization and 4.7x better efficiency.
Ray tracing and mobile

Ray tracing provides a potential rendering technique for future mobile applications that require photorealistic graphics...
Thank you