Active Access: A Mechanism for High-Performance Distributed Data-Centric Computations

MACIEJ BESTA, TORSTEN HOEFLER
REMOTE MEMORY ACCESS (RMA) PROGRAMMING
REMOTE MEMORY ACCESS (RMA) PROGRAMMING

Process p

Memory A
REMOTE MEMORY ACCESS (RMA) PROGRAMMING
REMOTE MEMORY ACCESS (RMA) PROGRAMMING

Process p
Memory
A

Process q
Memory
B

Cray
BlueWaters
REMOTE MEMORY ACCESS (RMA) PROGRAMMING

Process p
Memory A

Process q
Memory B

Cray BlueWaters
REMOTE MEMORY ACCESS (RMA) PROGRAMMING

Process p

Memory A

Process q

Memory B

Cray BlueWaters
REMOTE MEMORY ACCESS (RMA) PROGRAMMING

Process p
Memory

Process q
Memory

Cray
BlueWaters
REMOTE MEMORY ACCESS (RMA) PROGRAMMING

Process p
Memory

Process q
Memory

Cray
BlueWaters

A put

A
B
REMOTE MEMORY ACCESS (RMA) PROGRAMMING

Process p
Memory
A
B

A put

get B

A
B

Process q
Memory

Cray
BlueWaters
REMOTE MEMORY ACCESS (RMA) PROGRAMMING

Process p

Memory

A

B

Process q

Memory

A

B

A put

get B

flush

Cray

BlueWaters
REMOTE MEMORY ACCESS (RMA) PROGRAMMING

Process p

Memory

A

B

Process q

Memory

A

B

put

get

flush

Cray
BlueWaters
REMOTE MEMORY ACCESS PROGRAMMING

- Implemented in hardware in NICs in the majority of HPC networks (RDMA)
REMOTE MEMORY ACCESS PROGRAMMING

- Implemented in hardware in NICs in the majority of HPC networks (RDMA)
REMOTE MEMORY ACCESS PROGRAMMING

- Implemented in hardware in NICs in the majority of HPC networks (RDMA)
REMOTE MEMORY ACCESS PROGRAMMING

- Implemented in hardware in NICs in the majority of HPC networks (RDMA)
REMOTE MEMORY ACCESS PROGRAMMING

- Implemented in hardware in NICs in the majority of HPC networks (RDMA)
REMOTE MEMORY ACCESS PROGRAMMING

- Supported by many HPC libraries and languages
REMOTE MEMORY ACCESS PROGRAMMING

- Supported by many HPC libraries and languages
REMOTE MEMORY ACCESS PROGRAMMING

- Supported by many HPC libraries and languages
REMOTE MEMORY ACCESS PROGRAMMING

- Enables significant speedups over message passing in many types of applications, e.g.:

REMOTE MEMORY ACCESS PROGRAMMING

- Enables significant speedups over message passing in many types of applications, e.g.:
  - Speedup of ~1.5 for communication patterns in irregular workloads

REMOTE MEMORY ACCESS PROGRAMMING

- Enables significant speedups over message passing in many types of applications, e.g.:
  - Speedup of ~1.5 for communication patterns in irregular workloads
  - Speedup of ~1.4-2 in physics computations

RMA vs. Message Passing

RMA:

Process p
Memory

Process q
Memory

A put

flush
RMA vs. Message Passing

RMA:

Process p
Memory

Process q
Memory

Message Passing:

put
flush
RMA vs. Message Passing

RMA:
- Process p
  - Memory
- Process q
  - Memory

Message Passing:
- Process p
  - Memory
  - Message
- Process q
  - Memory
RMA vs. Message Passing

- Communication in RMA is one-sided

RMA:

Message Passing:
RMA vs. Message Passing

- Communication in RMA is one-sided

RMA:

Message Passing:
RMA vs. Message Passing

- Communication in RMA is one-sided

RMA:
- Process p puts memory A to memory of Process q
- No active participation, direct access to memory

Message Passing:
- Process p sends message A to Process q
- Process q receives message A and puts it into memory
RMA vs. Message Passing

- Communication in RMA is one-sided

RMA:
- Process p
  - Memory
  - \texttt{put} to Memory

Message Passing:
- Process p
  - Memory
  - \texttt{send} to Process q
  - \texttt{message} to Process q
  - \texttt{receive} from Process q
  - \texttt{queueing}

Process q
- Memory
  - \texttt{put} from Process p
  - \texttt{flush}
  - \texttt{queueing}
  - \texttt{receive}
  - \texttt{queueing}

No active participation, direct access to memory.

Explicit receive, possible queueing.
REMOTE MEMORY ACCESS PROGRAMMING

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal?

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? NO!
REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

No hash collision:

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

No hash collision:

\[ \Rightarrow 1 \text{ remote atomic} \]

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

No hash collision:

⇒ 1 remote atomic

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

No hash collision:

- 1 remote atomic
- Up to 5x speedup over MP [1]

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

No hash collision:
- 1 remote atomic
- Up to 5x speedup over MP [1]

A hash collision:

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

No hash collision:
- 1 remote atomic
- Up to 5x speedup over MP [1]

A hash collision:
- 4 remote atomics + 2 remote puts

---

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

```
No hash collision:
- 1 remote atomic
- Up to 5x speedup over MP [1]
```

```
A hash collision:
- 4 remote atomics + 2 remote puts
```

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO**
- Consider an insert in a distributed hashtable...

No hash collision:
- 1 remote atomic
- Up to 5x speedup over MP [1]

A hash collision:
- 4 remote atomics + 2 remote puts
- Significant performance drops

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

No hash collision:
- 1 remote atomic
- Up to 5x speedup over MP [1]

A hash collision:
- 4 remote atomics + 2 remote puts
- Significant performance drops

---

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

No hash collision:
- 1 remote atomic
- Up to 5x speedup over MP [1]

A hash collision:
- 4 remote atomics + 2 remote puts
- Significant performance drops

---

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

No hash collision:
- 1 remote atomic
- Up to 5x speedup over MP [1]

A hash collision:
- 4 remote atomastics + 2 remote puts
- Significant performance drops

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

![Diagram showing Proc p and Proc q with active access](Diagram)

No hash collision:
- 1 remote atomic
- Up to 5x speedup over MP [1]

A hash collision:
- 4 remote atomics + 2 remote puts
- Significant performance drops

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal? **NO!**
- Consider an insert in a distributed hashtable...

No hash collision:
- 1 remote atomic
- Up to 5x speedup over MP [1]

A hash collision:
- 4 remote atomics + 2 remote puts
- Significant performance drops

REMOTE MEMORY ACCESS PROGRAMMING

 Is it ideal?
 Consider an insert in a distributed hashtable...

No hash collision:

⇒ 1 remote atomic
⇒ Up to 5x speedup over MP [1]

A hash collision:

⇒ 4 remote atomics + 2 remote puts
⇒ Significant performance drops

Local execution; triggered by an active access. In RMA?

Proc p

Active access

Proc q

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal?
- Consider an insert in a distributed hashtable...

How to enable it?

No hash collision:
- 1 remote atomic
- Up to 5x speedup over MP [1]

A hash collision:
- 4 remote atomics + 2 remote puts
- Significant performance drops

Local execution; triggered by an active access. In RMA?

Use “active” semantics

REMOTE MEMORY ACCESS PROGRAMMING

- Is it ideal?
- Consider an insert in a distributed hashtable...

How to enable it?

No hash collision:
- 1 remote atomic
- Up to 5x speedup over MP [1]

Use and extend I/O MMUs and their paging capabilities

Use “active” semantics

Local execution; triggered by an active access. In RMA?

USE SEMANTICS FROM ACTIVE MESSAGES (AM) [1]

USE SEMANTICS FROM ACTIVE MESSAGES (AM) [1]

USE SEMANTICS FROM ACTIVE MESSAGES (AM) [1]

USE SEMANTICS FROM ACTIVE MESSAGES (AM) [1]

Use semantics from Active Messages (AM) [1]

USE SEMANTICS FROM ACTIVE MESSAGES (AM) [1]

USE SEMANTICS FROM ACTIVE MESSAGES (AM) [1]

USE SEMANTICS FROM ACTIVE MESSAGES (AM) [1]

USE SEMANTICS FROM ACTIVE MESSAGES (AM) [1]

USE SEMANTICS FROM ACTIVE MESSAGES (AM) [1]

**USE SEMANTICS FROM ACTIVE MESSAGES (AM) [1]**

USE SEMANTICS FROM ACTIVE MESSAGES (AM) [1]

Use semantics from Active Messages (AM) [1]

AM++ [2]
GASNet [3]

We need active puts/gets:
- Invoke a handler upon accessing a given page
- Preserve one-sided RMA behavior

USE SEMANTICS FROM ACTIVE MESSAGES (AM) [1]

We use it in syntax & semantics to enable the “active” behavior

We need active puts/gets:
- Invoke a handler upon accessing a given page
- Preserve one-sided RMA behavior

USE INPUT/OUTPUT MEMORY MANAGEMENT UNITS
Use Input/Output Memory Management Units

Main memory

MMU

CPU
USE INPUT/OUTPUT MEMORY MANAGEMENT UNITS

Main memory

MMU

Virtual addresses

CPU
USE INPUT/OUTPUT MEMORY MANAGEMENT UNITS

Main memory

Physical addresses

MMU

Virtual addresses

CPU
USE INPUT/OUTPUT MEMORY MANAGEMENT UNITS

Main memory

- Physical addresses
  - MMU
  - TLB
  - Virtual addresses
  - CPU
**Use Input/Output Memory Management Units**

Main memory

- IOMMU
- Physical addresses
- I/O devices

- MMU
- Virtual addresses
- CPU

- TLB
Use Input/Output Memory Management Units

Main memory

IOMMU

Device addresses

I/O devices

Physical addresses

MMU

Virtual addresses

CPU

TLB
USE INPUT/OUTPUT MEMORY MANAGEMENT UNITS

Main memory

Physical addresses

IOMMU

Device addresses

I/O devices

Physical addresses

MMU

Virtual addresses

CPU

TLB


**USE INPUT/OUTPUT MEMORY MANAGEMENT UNITS**

Main memory

- Physical addresses
- Device addresses
- Virtual addresses

IOMMU

- IOTLB

I/O devices

MMU

- TLB

CPU

- Physical addresses

- Physical addresses

- Virtual addresses
USE INPUT/OUTPUT MEMORY MANAGEMENT UNITS

Main memory

- Physical addresses
- Device addresses
- I/O devices

IOMMU

- IOTLB

MMU

- Virtual addresses
- TLB
- CPU

- Physical addresses

CPU

I/O devices

Virtual addresses

Device addresses

Physical addresses

Main memory
USE INPUT/OUTPUT MEMORY MANAGEMENT UNITS

Main memory

- Physical addresses
  - IOMMU
    - Device addresses
      - I/O devices
    - IOTLB
  - MMU
    - Virtual addresses
      - TLB
    - CPU

Physical addresses

Virtual addresses

Device addresses

Physical addresses
Use Input/Output Memory Management Units

Main memory

- Physical addresses

IOMMU

- Device addresses

I/O devices

MMU

- Virtual addresses

TLB

CPU

- Physical addresses

CPU

Virtual addresses

IOTLB

Device addresses

Physical addresses

I/O devices
USE INPUT/OUTPUT MEMORY MANAGEMENT UNITS

Main memory

- Physical addresses
- Device addresses
- I/O devices

IOMMU

- Virtual addresses
- IOTLB

MMU

- Physical addresses
- Virtual addresses
- TLB

CPU

- Physical addresses
- CPU
We propose it as a way to implement the “active” behavior.
IOMMU S AND RMA
IOMMUs AND RMA
IOMMUs AND RMA
IOMMUs AND RMA
IOMMUs and RMA

NIC

IOMMU

MSI

CPU

SMT cores
IOMMUs AND RMA

NIC → IOMMU → Main memory → CPU

IOMMU

MSI

SMT cores

CPU
IOMMUs AND RMA

NIC

Main memory

Remapping structures

IOMMU

MSI

CPU

SMT cores
IOMMUs AND RMA

NIC

Main memory

Remapping structures

Dev-to-PT

IOMMU

MSI

CPU

SMT cores
IOMMUS AND RMA

IOMMU

NIC

Main memory

Remapping structures

Dev-to-PT

PT

MSI

CPU

SMT cores
IOMMUs AND RMA

Main memory

Remapping structures

Dev-to-PT

PT

IOMMU

NIC

CPU

SMT cores

MSI
IOMMUS AND RMA

1. An RDMA packet

NIC → IOMMU → CPU

Main memory

Remapping structures:
- Dev-to-PT
- PT
IOMMU'S AND RMA

1. An RDMA packet
2. PCIe packets

NIC → IOMMU → Main memory

Remapping structures:
- Dev-to-PT
- PT

MSI

CPU

SMT cores
IOMMUs AND RMA

1. An RDMA packet
2. PCIe packets
3. IOMMU

Main memory

Remapping structures
- Dev-to-PT
- PT

MSI

CPU

SMT cores
IOMMUs and RMA

1. An RDMA packet
2. PCIe packets
3. IOMMU
4. Dev-to-PT cache

Remapping structures:
- Dev-to-PT
- PT

MSI

CPU

SMT cores

Main memory
IOMMUs AND RMA

1. An RDMA packet
2. PCIe packets
3. IOMMU
4. Dev-to-PT cache
5. Remapping structures

NIC

Main memory

CPU

SMT cores

MSI
IOMMUs AND RMA

An RDMA packet

PCIe packets

NIC

IOMMU

Dev-to-PT cache

IOTLB

Dev-to-PT

Remapping structures

PT

Main memory

MSI

CPU

SMT cores

1

2

3

4

5

6

7
IOMMU S AND RMA

1. An RDMA packet
2. PCIe packets
3. IOMMU
4. Dev-to-PT cache
5. Dev-to-PT
6. IOTLB
7. PT

Main memory

Remapping structures

CPU

SMT cores

MSI
IOMMUs and RMA

1. An RDMA packet
2. PCIe packets
3. IOMMU
4. Dev-to-PT cache
5. Remapping structures
6. IOTLB
7. PT

Main memory

PCIe packets

MSI

SMT cores

CPU
IOMMUs and RMA

1. An RDMA packet
2. PCIe packets
3. IOMMU
4. Dev-to-PT cache
5. Remapping structures
6. IOTLB
7. PT
8. W, R

Main memory

MSI

CPU

SMT cores
IOMMUs and RMA

1. An RDMA packet
2. PCIe packets
3. IOMMU
4. Dev-to-PT cache
5. Remapping structures
6. IOTLB
7. PT
8. System-wide fault log

MSI

NIC

Main memory

Dev-to-PT

CPU

SMT cores
IOMMU S AND RMA

An RDMA packet

NIC

PCIe packets

Main memory

Dev-to-PT cache

IOMMU

Dev-to-PT

IOTLB

Remapping structures

PT

System-wide fault log

Fault entry

Fault entry

SMT cores

CPU

MSI
IOMMUs and RMA

1. An RDMA packet
2. PCIe packets
3. NIC
4. Dev-to-PT cache
5. Remapping structures
6. IOTLB
7. PT
8. W
9. System-wide fault log
10. Fault entry → ... → Fault entry

Main memory

IOMMU

MSI

CPU

SMT cores
IOMMUs and RMA

1. An RDMA packet
2. PCIe packets
3. NIC
4. Dev-to-PT cache
5. Remapping structures
6. IOTLB
7. PT
8. W
9. System-wide fault log
10. MSI

Main memory

NIC → IOMMU → Dev-to-PT cache → IOTLB → Main memory

Main memory → Remapping structures → Main memory

Remapping structures → Dev-to-PT

Dev-to-PT → PT

System-wide fault log

Fault entry → ... → Fault entry

CPU

SMT cores
IOMMUs and RMA

1. An RDMA packet
2. PCIe packets
3. NIC
4. IOMMU
5. Dev-to-PT
6. IOTLB
7. PT
8. Remapping structures
9. System-wide fault log
10. MSI
11. CPU

Main memory

Fault entry → … → Fault entry

PCIe packets

Dev-to-PT cache

SMT cores
IOMMUs and RMA

1. An RDMA packet
2. PCIe packets
3. NIC
4. Dev-to-PT cache
5. Remapping structures
6. IOTLB
7. PT
8. W/R
9. System-wide fault log
10. MSI
11. CPU

Main memory

- An RDMA packet
- PCIe packets
- Remapping structures
- System-wide fault log
- User handlers
IOMMUs and RMA

1. An RDMA packet
2. PCIe packets
3. IOMMU
4. Dev-to-PT cache
5. Remapping structures
6. IOTLB
7. PT
8. W R
9. System-wide fault log
10. MSI
11. CPU
12. User handlers

NIC

Main memory

Dev-to-PT

Fault entry → … → Fault entry

Handler A

SMT cores

Fault entry

Handler A
We could use it somehow. But...
IOMMUs and RMA

We could use it somehow. But...

1. An RDMA packet
2. PCIe packets
3. NIC
4. IOMMU
5. Remapping structures
6. IOTLB
7. PT
8. Dev-to-PT
9. System-wide fault log
10. MSI
11. CPU
12. User handlers

No parallelism (single log)... BAD

We could use it somehow. But...
We could use it somehow. But…

IOMMUs AND RMA

1. An RDMA packet
2. PCIe packets
3. IOMMU
4. Dev-to-PT cache
5. Remapping structures
6. IOTLB
7. PT
8. W R
9. System-wide fault log
10. MSI
11. SMT cores
12. User handlers

NIC

Main memory

No parallelism (single log)… BAD

No multiplexing (single log)… BAD

We could use it somehow. But…
IOMMUs AND RMA

1. An RDMA packet
2. PCIe packets
3. NIC
4. Dev-to-PT cache
5. Remapping structures
6. IOTLB
7. PT
8. W/R
9. System-wide fault log
10. MSI
11. CPU
12. User handlers

We could use it somehow. But...

No parallelism (single log)... BAD

No multiplexing (single log)... BAD

Data is discarded... Extremely BAD

Fault entry → ... → Fault entry

Extremely BAD
**ACTIVE PUTS**

- An RDMA packet
- PCIe packets
- Main memory
- IOMMU
  - Dev-to-PT cache
  - IOTLB
- System-wide fault log
  - Fault entry → … → Fault entry
- User handlers
  - Handler A
- SMT cores
  - MSI
  - CPU
- Remapping structures
  - Dev-to-PT
  - PT

---

**Diagram Description**

- NIC
- PCIe packets
- Main memory
- IOMMU
  - Dev-to-PT cache
  - IOTLB
- System-wide fault log
  - Fault entry → … → Fault entry
- User handlers
  - Handler A
- SMT cores
  - MSI
  - CPU
**ACTIVE PUTS**

- An RDMA packet
- PCIe packets
- NIC

**IOMMU**
- Dev-to-PT cache
- IOTLB

**Main memory**

**Remapping structures**
- Dev-to-PT
- PT

**System-wide fault log**
- Fault entry $\rightarrow \cdots \rightarrow$ Fault entry

**Access log (private for each process)**

**User handlers**
- Handler A
- $\cdots$

**CPU**

**MSI**

$\text{spcl.inf.ethz.ch}$

$\text{@spcl_eth}$
ACTIVE PUTS

An RDMA packet

NIC

PCIe packets

Main memory

IOMMU

Dev-to-PT cache

IOTLB

MSI

CPU

SMT cores

Remapping structures

Dev-to-PT

PT

Access log (private for each process)

Fault entry

Fault entry...

Fault entry

System-wide fault log

Fault entry

Fault entry...

Fault entry

User handlers

Handler A

...
ACTIVE PUTS

An RDMA packet

NIC

PCIe packets

Main memory

Dev-to-PT cache

IOMMU

IOTLB

MSI

CPU

SMT cores

Remapping structures

Dev-to-PT

PT

System-wide fault log

Fault entry

Fault entry

Fault entry

Fault entry

Access log (private for each process)

Request data

Request data

User handlers

Handler A

...
ACTIVE PUTS

An RDMA packet

NIC

PCIe packets

IOMMU

Dev-to-PT cache

IOTLB

Main memory

MSI

CPU

SMT cores

Remapping structures

Dev-to-PT

PT

System-wide fault log

Fault entry → ... → Fault entry

Access log (private for each process)

Fault entry → ... → Fault entry

Request data

Request data

User handlers

Handler A

...
ACTIVE PUTS

An RDMA packet

IOMMU

Dev-to-PT cache

IOTLB

Access log table

CPU

SMT cores

System-wide fault log

Fault entry

Fault entry

Fault entry

Access log (private for each process)

Request data

Request data

Data can be reused

Main memory

Remapping structures

Dev-to-PT

PT

NIC

PCle packets

PCIe packets

Main memory
**ACTIVE PUTS**

- An RDMA packet
- PCIe packets
- Main memory
  - Remapping structures
    - Dev-to-PT
    - PT
  - IOMMU
    - Dev-to-PT cache
    - IOTLB
- Access log table
- MSI
- CPU
  - SMT cores
- System-wide fault log
  - Fault entry → ... → Fault entry
- Access log (private for each process)
  - Fault entry → ... → Fault entry
  - Request data
- User handlers
  - Handler A
  - ...
ACTIVE PUTS

An RDMA packet

NIC

PCIe packets

Main memory

IOMMU

IOTLB

Dev-to-PT cache

Access log table

MSI

CPU

SMT cores

Remapping structures

Dev-to-PT

PT

Stores addresses of each access log

System-wide fault log

Fault entry

Fault entry

User handlers

Handler A

Access log (private for each process)

Fault entry

Fault entry

Request data

Request data

Data can be reused
**ACTIVE PUTS**

- An RDMA packet
  - NIC
  - PCIe packets
  - PCIe packets
- Main memory
  - IOMMU
    - Dev-to-PT cache
    - IOTLB
    - Access log table
- System-wide fault log
  - Fault entry → ... → Fault entry
  - Access log (private for each process)
  - Fault entry → ... → Fault entry
  - Request data
  - Request data
- User handlers
  - Handler A
  - Data can be reused
- CPU
  - SMT cores
- Stores addresses of each access log
  - MSI
  - MSI
ACTIVE PUTS

An RDMA packet

NIC

PCIe packets

Main memory

IOMMU

Dev-to-PT cache

IOTLB

Access log table

MSI

CPU

SMT cores

Remapping structures

Dev-to-PT

PT

Access log (private for each process)

Fault entry

Request data

System-wide fault log

Fault entry

Fault entry

User handlers

Handler A

Data can be reused

Stores addresses of each access log

Decide on keeping/discardign the entry/data
ACTIVE PUTS

An RDMA packet

NIC

PCIe packets

Main memory

IOMMU

Dev-to-PT cache

IOTLB

Access log table

MSI

CPU

SMT cores

Remapping structures

Dev-to-PT

PT

Access log table

Fault entry → ... → Fault entry

System-wide fault log

Access log (private for each process)

Fault entry → ... → Fault entry

Request data

Request data

User handlers

Handler A

... Data can be reused

Stores addresses of each access log

Decide on keeping/discarding the entry/data

Stores addresses of each access log

Data can be reused

Decide on keeping/discarding the entry/data

Stores addresses of each access log

Data can be reused
**ACTIVE PUTS**

Stores addresses of each access log

Decide on keeping/discard the entry/data

Maps each page to an access log

Data can be reused
**ACTIVE PUTS**

- Stores addresses of each access log
- Decides on keeping/discard the entry/data
- Enables data-centric programming
- Data can be reused

- Spcl.inf.ethz.ch
- @spcl_eth
- ETH Zürich

**Diagram Description**

- **An RDMA packet** is received by the NIC, which forwards it to the IOMMU.
- **PCIe packets** are handled by the IOTLB, which maps them to the appropriate access log table.
- The IOMMU manages the remapping structures for both the Dev-to-PT and PT domains.
- **Main memory** communicates with the IOMMU through the Access log table, which stores addresses of each access log.
- **CPU** utilizes the System-wide fault log for fault entries, which can be monitored for access logs.
- **User handlers** (Handler A) are notified of any faults and can request data from the Access log (private for each process).
- **Access log table** is used to access the data stored in the system, allowing for efficient data reuse.
ACTIVE PUTS
ACTIVE PUTS

Process p

Process q

IOMMU

CPU

Main memory
ACTIVE PUTS

Process p

Process q

IOMMU

Accessed page

CPU

Access log

Main memory
ACTIVE PUTS

Process p

Process q

IOMMU

Accessed page
W = 0
WL = 1
WLD = 1

Access log

Main memory

CPU
ACTIVE PUTS

Do not modify the page

Process q

IOMMU

Accessed page

W = 0
WL = 1
WLD = 1

Access log

Main memory

Process p

CPU
**ACTIVE PUTS**

Do not modify the page

Log both the entry and the data of an incoming put

Accessed page

W = 0
WL = 1
WLD = 1

Access log

Main memory

Process q

IOMMU

CPU

Process p
**Active Puts**

Do not modify the page

Log both the entry and the data of an incoming put

1. **Put(X)**

Process p → IOMMU → Process q

Accessed page:
- W = 0
- WL = 1
- WLD = 1

Access log

Main memory

CPU
Active Puts

Process p

1. Put(X)

Process q

2. attempt to write(X)

IOMMU

Access log

Accessed page

W = 0
WL = 1
WLD = 1

Main memory

CPU

Do not modify the page

Log both the entry and the data of an incoming put

Accessed page
ACTIVE PUTS

Process p

1. Put(X)

Process q

2. Attempt to write(X)

IOMMU

3. Page fault! (W = 0)

Accessed page

Access log

CPU

Main memory

Do not modify the page

Log both the entry and the data of an incoming put

Accessed page

W = 0
WL = 1
WLD = 1

Attempt to write(X)
**ACTIVE PUTS**

Process P

1. **Put(X)**

2. Attempt to write(X)

3. Page fault! (W = 0)

4. Move(X)

**Do not modify the page**

**Log both the entry and the data of an incoming put**

**Access log**

Main memory

Accessed page

W = 0
WL = 1
WLD = 1

CPU

Process q

IOMMU
ACTIVE Puts

Process p

1. Put(X)

Process q

2. Attempt to write(X)

IOMMU

Page fault!

(W = 0)

Accessed page

W = 0
WL = 1
WLD = 1

Access log

X

Do not modify the page

Log both the entry and the data of an incoming put

CPU

Main memory
ACTIVE PUTS

Log both the entry and the data of an incoming put

Do not modify the page

1. Put(X)

Process p

Process q

2. Attempt to write(X)

Accessed page

3. Page fault! (W = 0)

W = 0
WL = 1
WLD = 1

Access log

4. Move(X)

Accessed page

X

5. Process(X)

Main memory

IOMMU

CPU
**ACTIVE GETS**

- An RDMA packet
- NIC
- PCIe packets
- IOMMU
  - Dev-to-PT cache
  - IOTLB
  - Access log table
- MSI
- CPU
  - SMT cores
- Main memory
- Remapping structures
  - Dev-to-PT
    - PT
    - IUID
  - Access log table (private for each process)
  - System-wide fault log
    - Fault entry → ... → Fault entry
    - Request data → ... → Request data
- User handlers
  - Handler A
**ACTIVE GETS**

- **An RDMA packet**
  - NIC
  - PCIe packets

- **Main memory**
  - IOMMU
    - Dev-to-PT cache
    - IOTLB
  - Access log table

- **MSI**
  - CPU
    - SMT cores

- **Remapping structures**
  - Dev-to-PT
    - PT
  - User handlers
    - Handler A

- **System-wide fault log**
  - Fault entry → ... → Fault entry
  - Access log (private for each process)
    - Fault entry → ... → Fault entry
    - Request data → Request data
ACTIVE GETS
ACTIVE GETS

Process p

Process q

IOMMU

CPU

Main memory
ACTIVE GETS

Process p

Process q

IOMMU

Accessed page

CPU

Access log

Main memory
ACTIVE GETS

Process p

Process q

IOMMU

Accessed page

R = 1
RL = 1
RLD = 1

Access log

Main memory

CPU
ACTIVE GETS

Enable reading from the page

Process q

- IOMMU
- Accessed page
- R = 1
- RL = 1
- RLD = 1

Access log

CPU

Main memory

Process p
ACTIVE GETS

Enable reading from the page

Log both the entry and the data accessed by a get

Process q

Accessed page

R = 1
RL = 1
RLD = 1

Access log

Main memory

Process p

IOMMU

CPU

Main memory

Accessed page

R = 1
RL = 1
RLD = 1

Access log

Main memory

Process q
ACTIVE GETS

Enable reading from the page

Log both the entry and the data accessed by a get

Process p

1 Get(X)

Process q

IOMMU

Accessed page

R = 1
RL = 1
RLD = 1

Access log

CPU

Main memory
**ACTIVE GETS**

Enable reading from the page

Log both the entry and the data accessed by a get

Process p

1. Get(X)

Process q

IOMMU

Accessed page

R = 1
RL = 1
RLD = 1

Access log

CPU

Main memory
ACTIVE GETS

Enable reading from the page

Log both the entry and the data accessed by a get

1. Get(X)
2. Read(X)

Process p

IOMMU

CPU

Process q

 Accessed page

R = 1
RL = 1
RLD = 1

Main memory

Access log
**ACTIVE GETS**

Enable reading from the page

Log both the entry and the data accessed by a get

Process p

1. Get(X)

IOMMU

2. Read(X)

Accessed page

R = 1
RL = 1
RLD = 1

Access log

Main memory

Process q

CPU

Main memory

Active Gets
ACTIVE GETS

Enable reading from the page

Log both the entry and the data accessed by a get

1. Get(X)
2. Read(X)
3. Copy(X)

Process p

IOMMU

Accessed page

R = 1
RL = 1
RLD = 1

Access log

Main memory

CPU

Process q

X

Copy(X)
**ACTIVE GETS**

Enable reading from the page

Log both the entry and the data accessed by a get

1. Process p \( \text{Get}(X) \)
2. IOMMU \( \text{Read}(X) \)
3. Copy(X)
4. Process(X)

Process q

Accessed page

\( R = 1 \)
\( RL = 1 \)
\( RLD = 1 \)

Access log

Main memory

X
ACTIVE GETS

Enable reading from the page

Log both the entry and the data accessed by a get

Process p

Get(X)

IOMMU

Read(X)

Copy(X)

Process(X)

Access log

X

Process q

Accessed page

R = 1
RL = 1
RLD = 1

Sounds like we can reuse most of the existing stuff!
INTERACTIONS WITH THE CPU

An RDMA packet

NIC

PCIe packets

Dev-to-PT cache

IOTLB

Access log table

Main memory

Remapping structures

Dev-to-PT

PT

Access log table

System-wide fault log

Fault entry → … → Fault entry

Access log (private for each process)

Request data

Fault entry → … → Fault entry

Request data

User handlers

Handler A

SMT cores

MSI

CPU
INTERACTIONS WITH THE CPU

IOMMU

Dev-to-PT cache

IOTLB

Access log table

MSI

CPU

SMT cores

...
INTERACTIONS WITH THE CPU

IOMMU

- Dev-to-PT cache
- IOTLB

Access log table

MSI

CPU

SMT cores...
INTERACTIONS WITH THE CPU

- Interrupts

IOMMU

Dev-to-PT cache

IOTLB

Access log table

MSI

CPU

SMT cores
INTERACTIONS WITH THE CPU

- Interrupts
- Polling
INTERACTIONS WITH THE CPU

- Interrupts
- Polling
- Direct notifications via scratchpads
INTERACTIONS WITH THE CPU

- Interrupts
- Polling
- Direct notifications via scratchpads
INTERACTIONS WITH THE CPU

- Interrupts
- Polling
- Direct notifications via scratchpads
**INTERACTIONS WITH THE CPU**

- Interrupts
- Polling
- Direct notifications via scratchpads

Diagram:

- IOMMU
  - Dev-to-PT cache
  - IOTLB
  - Access log table

- MSI

- CPU
  - SMT cores
  - Scratchpad memory
  - Var
  - Handler A

Symbol: +
**INTERACTIONS WITH THE CPU**

- Interrupts
- Polling
- Direct notifications via scratchpads

**Diagram**

- IOMMU
  - Dev-to-PT cache
  - IOTLB
  - Access log table

- MSI

- CPU
  - SMT cores
  - Scratchpad memory
    - Var
    - Handler A
INTERACTIONS WITH THE CPU

- Interrupts
- Polling
- Direct notifications via scratchpads
INTERACTIONS WITH THE CPU

- Interrupts
- Polling
- Direct notifications via scratchpads

Are we done?
INTERACTIONS WITH THE CPU

- Interrupts
- Polling
- Direct notifications via scratchpads

Are we done?

Well…
CONSISTENCY
CONSISTENCY

- A weak consistency model [1]

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
CONSISTENCY

- A weak consistency model [1]
  - Consistency on-demand

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
CONSISTENCY

- A weak consistency model [1]
  - Consistency on-demand
  - active_flush(int target_id)

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
CONSISTENCY

- A weak consistency model [1]
  - Consistency on-demand
- active_flush(int target_id)
  - Enforces the completion of active accesses issued by the calling process and targeted at target_id

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
CONSISTENCY

- A weak consistency model [1]
  - Consistency on-demand

- active_flush(int target_id)
  - Enforces the completion of active accesses issued by the calling process and targeted at target_id
  - Implemented with an active get issued at a special flushing page

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
CONSISTENCY

- A weak consistency model [1]
  - Consistency on-demand
  - active_flush(int target_id)
    - Enforces the completion of active accesses issued by the calling process and targeted at target_id
    - Implemented with an active get issued at a special flushing page

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
CONSISTENCY

- A weak consistency model [1]
  - Consistency on-demand
  - active_flush(int target_id)
    - Enforces the completion of active accesses issued by the calling process and targeted at target_id
    - Implemented with an active get issued at a special flushing page

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
**CONSISTENCY**

- A weak consistency model [1]
  - Consistency on-demand
- active_flush(int target_id)
  - Enforces the completion of active accesses issued by the calling process and targeted at target_id
  - Implemented with an active get issued at a special flushing page

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
CONSISTENCY

- A weak consistency model [1]
  - Consistency on-demand
- active_flush(int target_id)
  - Enforces the completion of active accesses issued by the calling process and targeted at target_id
  - Implemented with an active get issued at a special flushing page

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
**CONSISTENCY**

- A weak consistency model [1]
  - Consistency on-demand
  - active_flush(int target_id)
    - Enforces the completion of active accesses issued by the calling process and targeted at target_id
    - Implemented with an active get issued at a special *flushing page*

---

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
**CONSISTENCY**

- A weak consistency model [1]
  - Consistency on-demand
  - `active_flush(int target_id)`
    - Enforces the completion of active accesses issued by the calling process and targeted at `target_id`
    - Implemented with an active get issued at a special *flushing page*

---

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
**CONSISTENCY**

- A weak consistency model [1]
  - Consistency on-demand
  - active_flush(int target_id)
    - Enforces the completion of active accesses issued by the calling process and targeted at target_id
    - Implemented with an active get issued at a special *flushing page*

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
CONSISTENCY

- A weak consistency model [1]
  - Consistency on-demand
- active_flush(int target_id)
  - Enforces the completion of active accesses issued by the calling process and targeted at target_id
  - Implemented with an active get issued at a special flushing page

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
CONSISTENCY

- A weak consistency model [1]
  - Consistency on-demand
  - active_flush(int target_id)
    - Enforces the completion of active accesses issued by the calling process and targeted at target_id
    - Implemented with an active get issued at a special flushing page

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
**CONSISTENCY**

- A weak consistency model [1]
  - Consistency on-demand
  - `active_flush(int target_id)`
    - Enforces the completion of active accesses issued by the calling process and targeted at `target_id`
    - Implemented with an active get issued at a special *flushing page*

---

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
**CONSISTENCY**

- A weak consistency model [1]
  - Consistency on-demand
  - active_flush(int target_id)
    - Enforces the completion of active accesses issued by the calling process and targeted at target_id
    - Implemented with an active get issued at a special *flushing page*

---

[1] K. Gharachorloo et al. Memory consistency and event ordering in scalable shared-memory multiprocessors. ISCA '90
CONSISTENCY

- A weak consistency model [1]
  - Consistency on-demand
  - active_flush(int target_id)
    - Enforces the completion of active accesses issued by the calling process and targeted at target_id
    - Implemented with an active get issued at a special flushing page

CONSISTENCY

IOMMU

- Dev-to-PT cache
- IOTLB

Access log table

MSI

CPU

- SMT cores
- Scratchpad memory
- Handler A
- Hyper thread
**CONSISTENCY**

- IOMMU
  - Dev-to-PT cache
  - IOTLB
  - Access log table
  - Flushing buffer

- MSI

- CPU
  - SMT cores
  - Scratchpad memory
  - Handler A
  - Hyper thread
CONSISTENCY

IOMMU

- Dev-to-PT cache
- IOTLB
- Access log table
- Flushing buffer

MSI

CPU

- SMT cores
- Scratchpad memory
- Handler A
- Hyper thread

Contains the addresses of flushing pages
**CONSISTENCY**

- **IOMMU**
  - Dev-to-PT cache
  - IOTLB
  - Access log table
  - Flushing buffer

- **MSI**

- **CPU**
  - SMT cores
  - Scratchpad memory
  - Handler A
  - Hyperthread

**Contains the addresses of flushing pages**

**Maps flushing pages to IUIDs and access logs**
**CONSISTENCY**

**IOMMU**
- Dev-to-PT cache
- Access log table
- Flushing buffer

**CPU**
- SMT cores
- Scratchpad memory
- Handler A
- Hyper thread

**MSI**

Contains the addresses of flushing pages

Maps flushing pages to IUIDs and access logs
Let’s summarize…
Let’s summarize…

Active Messages
Let's summarize...

**Active Messages**

We need active processes:
- Invoke a handler upon accessing a given page
- Preserve one-sided DMA behavior

**IOMMUs**

**Use semantics from Active Messages (AM) [1]**

**Use input/output memory management units**
Let’s summarize…

**Active Messages**

- **USE SEMANTICS FROM ACTIVE MESSAGES (AM)**
  - IBM
  - Myrics [2]
  - GASNet [3]

  We need active puts/gets:
  - Invoke a handler upon accessing a given page
  - Preserve one-sided RMA behavior

**IOMMUs**

- **USE INPUT/OUTPUT MEMORY MANAGEMENT UNITS**
  - Physical addresses
  - IOMMU
  - Physical addresses
  - MMU
  - Device addresses
  - IOTLB
  - Virtual addresses
  - TLB
  - I/O devices
  - CPU

**Active Puts/Gets**

- **ACTIVE PUTS**
  - Access $k$ index
  - Access $k$ entry
  - Do not hold cache
  - Process 1
  - Process 0
  - Access tag
  - Access tag

- **ACTIVE GETS**
  - Read line
  - IOMMU
  - Process 1
  - Process 0
  - Access tag
  - Access tag
Let’s summarize...

**Active Messages**

**IOMMUs**

**Consistency**

- A weak consistency model [1]
  - Consistency on-demand
  - active_flush(int target_id)
  - Ensures the completion of active accesses issued by the calling process and targeted at target_id
  - Implemented with an active get issued at a special flushing page

**Active Puts/Gets**
Let’s summarize...

**Active Messages**

- **USE SEMANTICS FROM ACTIVE MESSAGES (AM)**
  - IBM
  - GasNet

  We need active puts/gets:
  - Invoke a handler upon accessing a given page
  - Preserve one-sided RMA behavior

**IOMMUs**

- **USE INPUT/OUTPUT MEMORY MANAGEMENT UNITS**
  - Main memory
  - Physical addresses
  - IOMU
  - Device addresses
  - I/O devices
  - CPU
  - AMD
  - IBM
  - ARM
  - Sun Solarflare
  - Intel

**Consistency**

- A weak consistency model [1]
- Consistency on-demand
- active_flush(int target_id)
- Enforces the completion of active accesses issued by the calling process and targeted at target_id
- Implemented with an active get issued at a special flushing page

**Active Puts/Gets**

- **ACTIVE PUTS**
  - Process a
  - IOMU
  - Access log
  - Page fault
  - HW

- **ACTIVE PUTS**
  - Process p
  - Attempt to active(s)
  - Page fault
  - Access log

How can we use it?
ACTIVE ACCESS USE-CASES
DISTRIBUTED HASHTABLE
ACTIVE ACCESS USE-CASES
DISTRIBUTED HASHTABLE

- Used to construct key-value stores (e.g., Memcached [1])

ACTIVE ACCESS USE-CASES
DISTRIBUTED HASHTABLE

- Used to construct key-value stores (e.g., Memcached [1])

Local volume 0 (at process 0)  Local volume 1 (at process 1)  Local volume N-1 (at process N-1)

**ACTIVE ACCESS USE-CASES**

**DISTRIBUTED HASHTABLE**

- Used to construct key-value stores (e.g., Memcached [1])

---

ACTIVE ACCESS USE-CASES

DISTRIBUTED HASH TABLE

- Used to construct key-value stores (e.g., Memcached [1])

Local volume 0 (at process 0)
- Table of elements
  - Overflow heap

Local volume 1 (at process 1)
- Table of elements
  - Overflow heap

Local volume N-1 (at process N-1)
- Table of elements
  - Overflow heap

ACTIVE ACCESS USE-CASES

DISTRIBUTED HASHTABLE: INSERTS (RMA)
ACTIVE ACCESS USE-CASES
DISTRIBUTED HASHTABLE: INSERTS (RMA)
ACTIVE ACCESS USE-CASES
DISTRIBUTED HASHTABLE: INSERTS (RMA)
ACTIVE ACCESS USE-CASES
DISTRIBUTED HASHTABLE: INSERTS (RMA)
ACTIVE ACCESS USE-CASES

DISTRIBUTED HASHTABLE: INSERTS (RMA)
ACTIVE ACCESS USE-CASES
DISTRIBUTED HASHTABLE: INSERTS (RMA)

Proc p

CAS (insert attempt)

FAD (get and increment ptr to the next free cell)

PUT (insert element)

FAD & CAS & PUT (update ptrs)

Proc q

Table of elements

Overflow heap
**ACTIVE ACCESS USE-CASES**

**DISTRIBUTED HASHTABLE: INSERTS (AA)**

Proc p

CAS (insert attempt)

FAD (get and increment ptr to the next free cell)

PUT (insert element)

FAD & CAS & PUT (update_ptrs)

Overflow heap

Proc q

Table of elements
ACTIVE ACCESS USE-CASES
DISTRIBUTED HASHTABLE: INSERTS (AA)

Proc p

FAD (get and increment ptr to the next free cell)
PUT (insert element)
FAD & CAS & PUT (update ptrs)

Proc q

Table of elements
Overflow heap
ACTIVE ACCESS USE-CASES
DISTRIBUTED HASH TABLE: INSERTS (AA)

**Proc p**

PUT (intercepted by the IOMMU)

FAD (get and increment ptr to the next free cell)

PUT (insert element)

FAD & CAS & PUT (update ptrs)

**Proc q**

Table of elements

Overflow heap
ACTIVE ACCESS USE-CASES
DISTRIBUTED HASHTABLE: INSERTS (AA)

Proc p

PUT (intercepted by the IOMMU)

FAD (get and increment ptr to the next free cell)

PUT (insert element)

FAD & CAS & PUT (update ptrs)

Table of elements

Overflow heap

All other accesses become local

Proc q
ACTIVE ACCESS USE-CASES
DISTRIBUTED HASHTABLE: INSERTS (AA)

Proc p

PUT (intercepted by the IOMMU)

Proc q

Table of elements

Overflow heap
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)

Machine 0  Machine 1  Machine N-1
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)

Machine 0
Proc 0
  MMU
  Memory
  NIC

Machine 1
Proc 1
  MMU
  Memory
  NIC

Machine N-1
Proc N-1
  MMU
  Memory
  NIC
**ACTIVE ACCESS USE-CASES**

**VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)**

Machine 0

- Proc 0
- MMU
- Memory
- IOMMU
- NIC

Machine 1

- Proc 1
- MMU
- Memory
- IOMMU
- NIC

Machine N-1

- Proc N-1
- MMU
- Memory
- IOMMU
- NIC
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)

Machine 0
  Proc 0
  MMU
  Memory
  IOMMU
  NIC

Machine 1
  Proc 1
  MMU
  Memory
  IOMMU
  NIC

Machine N-1
  Proc N-1
  MMU
  Memory
  IOMMU
  NIC

Local memory protection

Memory
V-GAS
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)

Machine 0
- Proc 0
- MMU
- Memory
- IOMMU
- NIC

Machine 1
- Proc 1
- MMU
- Memory
- IOMMU
- NIC

Machine N-1
- Proc N-1
- MMU
- Memory
- IOMMU
- NIC

Local memory protection
V-GAS
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)

Remote memory protection

Machine 0
Proc 0
MMU
Memory
IOMMU
NIC

Machine 1
Proc 1
MMU
Memory
IOMMU
NIC

Machine N-1
Proc N-1
MMU
Memory
IOMMU
NIC

Local memory protection
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)

Remote memory protection

Machine 0
Proc 0
MMU
Memory
IOMMU
NIC

Machine 1
Proc 1
MMU
Memory
IOMMU
NIC

Machine N-1
Proc N-1
MMU
Memory
IOMMU
NIC

Local memory protection

V-GAS

Remote memory protection

Local memory protection
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)

Remote memory protection

Machine 0
Proc 0
MMU
Memory
IOMMU
NIC

Machine 1
Proc 1
MMU
Memory
IOMMU
NIC

Machine N-1
Proc N-1
MMU
Memory
IOMMU
NIC

Local memory protection

V-GAS

Fetch data (used for logging, fault-tolerance, etc...)

PERFORMANCE

- Evaluation on CSCS Monte Rosa
  - 1,496 computing Cray XE6 nodes
  - 47,872 schedulable cores
  - 46TB memory
- 3 microbenchmarks
- 4 use-cases
PERFORMANCE: MICROBENCHMARKS
RAW DATA TRANSFER
PERFORMANCE: MICROBENCHMARKS
RAW DATA TRANSFER

- Workload simulated with [1]:
PERFORMANCE: MICROBENCHMARKS
RAW DATA TRANSFER

- Workload simulated with [1]:

PERFORMANCE: MICROBENCHMARKS
RAW DATA TRANSFER

- Workload simulated with [1]:

- Data generated with:

**Performance: Microbenchmarks**

**Raw Data Transfer**

- Workload simulated with [1]:

- Data generated with:
  - PktGen [2]

---


**PERFORMANCE: MICROBENCHMARKS RAW DATA TRANSFER**

- Workload simulated with [1]:
  ![gem5](image)

- Data generated with:
  - PktGen [2]
  - Netmap [3]

**Performance: Microbenchmarks**

**Raw Data Transfer**

- Workload simulated with [1]:
  ![gem5](image)

- Data generated with:
  - PktGen [2]
  - Netmap [3]

![Graph showing bandwidth vs. packet size](image)

---

PERFORMANCE: LARGE-SCALE CODES
COMPARISON TARGETS
PERFORMANCE: LARGE-SCALE CODES
COMPARISON TARGETS

Active Access
PERFORMANCE: LARGE-SCALE CODES
COMPARISON TARGETS

Active Access

AA-Int
PERFORMANCE: LARGE-SCALE CODES
COMPARISON TARGETS

Active Access

AA-Int  AA-Poll
**Performance: Large-Scale Codes Comparison Targets**

<table>
<thead>
<tr>
<th>Active Access</th>
<th>AA-Poll</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA-Int</td>
<td>AA-SP</td>
</tr>
</tbody>
</table>
PERFORMANCE: LARGE-SCALE CODES
COMPARISON TARGETS

Active Access
AA-Int
AA-Poll
AA-SP

RMA
PERFORMANCE: LARGE-SCALE CODES
COMPARISON TARGETS

Active Access
AA-Int
AA-Poll
AA-SP

RMA
DMAPP
PERFORMANCE: LARGE-SCALE CODES
COMPARISON TARGETS

Active Access
AA-Int
AA-Poll
AA-SP

RMA
DMAPP
PERFORMANCE: LARGE-SCALE CODES
COMPARISON TARGETS

Active Access
AA-Int
AA-Poll
AA-SP

RMA
DMAPP

IBM Cell
PERFORMANCE: LARGE-SCALE CODES
COMPARISON TARGETS

Active Access

AA-Int

AA-Poll

AA-SP

RMA

DMAPP

IBM Cell

Mellanox Technologies

InfiniBand

RoCE
**Performance: Large-Scale Codes Comparison Targets**

- Active Access
  - AA-Int
  - AA-Poll
  - AA-SP

- RMA
- DMAPP

- IBM Cell
- Mellanox RoCE
Performance: Large-Scale Codes
Comparison Targets

Active Access
AA-Int
AA-Poll
AA-SP

RMA
DMAPP
Cell
RoCE

Active Messages
AM

IBM
Cray
Mellanox
InfiniBand
RoCE
Performance: Large-Scale Codes Comparison Targets

- Active Access
  - AA-Int
  - AA-Poll
  - AA-SP

- Active Messages
  - AM
  - AM-Exp

- RMA
- DMAPP
- Cell
- RoCE
PERFORMANCE: LARGE-SCALE CODES
COMPARISON TARGETS

Active Access
  AA-Int
  AA-Poll
  AA-SP

Active Messages
  AM
  AM-Exp
  AM-Onload

RMA
DMAPP
IBM
Cell
Mellanox
RoCE
InfiniBand
**Performance: Large-Scale Codes Comparison Targets**

- Active Access
  - AA-Poll
  - AA-Int
  - AA-SP

- Active Messages
  - AM
  - AM-Onload
  - AM-Exp
  - AM-Ints

- RMA

- DMAPP

- IBM Cell

- Mellanox Technologies

- InfiniBand

- RoCE
PERFORMANCE: LARGE-SCALE CODES
COMPARISON TARGETS

Active Access
AA-Poll
AA-Int
AA-SP

RMA
DMAPP
Cell
RoCE

Active Messages
AM
AM-Onload
AM-Exp
AM-Ints

IBM
DCMF
LAPI
PAMI
**Performance: Large-Scale Codes**

**Comparison Targets**

- Active Access
  - AA-Poll
  - AA-Int
  - AA-SP

- RMA
  - DMAPP
  - InfiniBand
  - RoCE

- Active Messages
  - AM
  - AM-Onload
  - AM-Exp
  - AM-Ints

- IBM
  - DCMF
  - LAPI
  - PAMI
  - MX

- Myricom
**Performance: Large-Scale Codes Comparison Targets**

Active Access
- AA-Poll
- AA-Int
- AA-SP

Active Messages
- AM
- AM-Onload
- AM-Exp
- AM-Ints

**IBM**
- DCMF
- LAPI
- PAMI

**Myricom**
- MX

**RMA**
- DMAPP
- Cell

**Cray**
- RoCE

**Mellanox Technologies**
**Performance: Large-Scale Codes**

**Distributed Hashtable**

Collisions: 5%

Collisions: 25%

![Graph showing performance impact with varying collisions](image)
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (AA):

  Proc p
  \[\rightarrow\] GET
  IOMMU
  \[\rightarrow\] Log the GET

  Proc q
  \[\rightarrow\] Fetch the logs
  \[\rightarrow\] reply the GET

p is modified
CONCLUSIONS
CONCLUSIONS

Active Access
CONCLUSIONS

Active Access

Alleviates RMA’s problems with AMs while preserving one-sided semantics
CONCLUSIONS

Active Access

Alleviates RMA’s problems with AMs while preserving one-sided semantics

Uses commodity & common IOMMUs
CONCLUSIONS

Active Access

Alleviates RMA’s problems with AMs while preserving one-sided semantics

Uses commodity & common IOMMUs

Extends paging capabilities in a distributed environment
CONCLUSIONS

Active Access

Alleviates RMA’s problems with AMs while preserving one-sided semantics

Uses commodity & common IOMMUs

Extends paging capabilities in a distributed environment

Data-centric programming
**CONCLUSIONS**

**Active Access**

Alleviates RMA’s problems with AMs while preserving one-sided semantics

**Uses commodity & common IOMMUs**

Extends paging capabilities in a distributed environment

**Data-centric programming**

Addresses of pages guide the execution of handlers
# CONCLUSIONS

**Active Access**

- Alleviates RMA’s problems with AMs while preserving one-sided semantics
- Extends paging capabilities in a distributed environment

**Data-centric programming**

- Uses commodity & common IOMMUs
- Addresses of pages guide the execution of handlers
- Hashtables, logging schemes, counters, V-GAS, checkpointing...
CONCLUSIONS

Active Access

- Uses commodity & common IOMMUs
- Extends paging capabilities in a distributed environment
- Alleviates RMA’s problems with AMs while preserving one-sided semantics

Data-centric programming

- Addresses of pages guide the execution of handlers
- Hashtables, logging schemes, counters, V-GAS, checkpointing...

Performance
CONCLUSIONS

Active Access

- Alleviates RMA’s problems with AMs while preserving one-sided semantics

Data-centric programming

- Addresses of pages guide the execution of handlers

Uses commodity & common IOMMUs

- Extends paging capabilities in a distributed environment

Performance

- Accelerates various distributed codes

- Hashtables, logging schemes, counters, V-GAS, checkpointing...
Thank you for your attention
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging – a popular mechanism for fault-tolerance.
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging – a popular mechanism for fault-tolerance.
- Remote communication (puts/gets) is logged.
 ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging – a popular mechanism for fault-tolerance.
- Remote communication (puts/gets) is logged.
- Upon a process crash, it is restored and uses the logs to replay its previous actions.
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging – a popular mechanism for fault-tolerance.
- Remote communication (puts/gets) is logged.
- Upon a process crash, it is restored and uses the logs to replay its previous actions.
- Logs are stored in volatile memories.
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA
ACTIVE ACCESS USE-CASES

ACCELERATING LOGGING FOR RMA

- Logging puts:
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging puts:

```
Proc p
```
```
Proc q
```
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging puts:
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging puts:

```
 Proc p
   ^
    |   PUT   |
    |         |
    |         |
    v
   Proc q
```

q is modified
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging puts:

\[\text{Proc } p \rightarrow \text{Proc } q\]

\[\text{PUT} \quad \text{Log the PUT}\]

\[q \text{ is modified}\]
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging puts:

```
Proc p
```
```
Proc q
```

PUT

Log the PUT

q is modified
ACTIVE ACCESS USE-CASES

ACCELERATING LOGGING FOR RMA

- Logging puts:

  Proc p
  \[\text{PUT} \downarrow\]
  \[\text{Log the PUT}\]

  Proc q
  \[\text{q is modified}\]

  Fetch the logs
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging puts:

  - Proc p
  - PUT
  - Log the PUT
  - Fetch the logs

  - Proc q
  - Reply the PUT
  - q is modified
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (naive):

```
Proc p
```
```
Proc q
```
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (naive):

```
Proc p

GET

Proc q
```
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (naive):

\[ \text{Proc } p \quad \text{GET} \quad \text{Proc } q \]

\( p \) is modified
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (naive):

  p is modified

  Proc p

  GET

  Log the GET

  Proc q
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (naive):

  ![Diagram showing the process of logging in a system](image)
Active Access Use-Cases
Accelerating Logging for RMA

- Logging gets (naive):

  Proc p
  \[\text{p is modified}\]

  \[\text{Log the GET}\]

  \[\text{Attempt to reply the GET}\]

  Proc q

  GET
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (naive):

  p is modified

  Proc p

  Log the GET

  Attempt to reply the GET

  FAIL!

  Proc q

  GET
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (traditional) [1]:

ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (traditional) [1]:

ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (traditional) [1]:

ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (traditional) [1]:

ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (traditional) [1]:

ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (traditional) [1]:

ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (AA):

  p is modified
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (AA):

  ![Diagram]

  - p is modified
  - Proc \( p \) and Proc \( q \)
  - GET
  - IOMMU
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (AA):

```
Proc p ----> GET ----> Proc q

p is modified

IOMMU

Log the GET
```
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (AA):

  Proc \( p \)

  \( p \) is modified

  Proc \( q \)

  GET

  IOMMU

  Log the GET
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (AA):

  Proc p
  \[\text{GET}\]
  IOMMU
  \[\text{Log the GET}\]
  \[\text{Fetch the logs}\]
  Proc q

p is modified
Active Access Use-Cases
Accelerating Logging for RMA

- Logging gets (AA):

  ![Diagram]

  - Proc p
  - Proc q
  - GET
  - IOMMU
  - Log the GET
  - Fetch the logs
  - reply the GET

  p is modified
Performance: Large-Scale Codes Fault Tolerance Scheme

Logging gets:

```
Logged gets/second
```

```
Processes
```

```
No-FT
AA-Poll
RMA
AM
```

Sorting time:

```
Latency [s]
```

```
Processes
```

```
No-FT
AA-Poll
RMA
AM
```