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
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

A

Process q

Memory

A

put

A

B

Cray
BlueWaters
REMOTE MEMORY ACCESS (RMA) PROGRAMMING

Process p

Memory

A

B

Process q

Memory

A

B

A put

get

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

Put

get

flush

Process q
Memory

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

A

Process q

Memory

A put

flush
RMA vs. Message Passing

RMA:

Process p

Memory

Process q

Memory

Message Passing:
RMA vs. Message Passing

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

Message Passing:
- Process p
  - Memory
- Process q
  - Memory

A Message
A put
flush
A message
Communication in RMA is one-sided

**RMA vs. Message Passing**

- RMA:
  - Process p
  - Memory
  - A
  - put
  - flush

- Message Passing:
  - Process p
  - Memory
  - A
  - message
  - A
**RMA vs. Message Passing**

- Communication in RMA is one-sided

**RMA:**
- Process p
  - Memory
  - put
- Process q
  - Memory
  - put
  - flush

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

- Communication in RMA is one-sided

**RMA:**
- Process p
- Memory
- process put

**Process q**
- Memory
- no active participation, direct access to memory

**RMA:**
- process put

**Message Passing:**
- Process p
- Memory
- process message

**Process q**
- Memory
RMA vs. Message Passing

Communication in RMA is one-sided

RMA:
- Process p
  - put
  - flush

Message Passing:
- Process p
  - send
  - explicit receive, possible queueing

no active participation, direct access to memory
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?
- Consider an insert in a distributed hashtable...

How to enable it?

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

2. A hash collision:
   - More atomics & puts
   - Significant performance drops

Use and extend I/O MMUs and their paging capabilities

Use “active” semantics

Local execution; triggered by an active access? in RMA?

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

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

We propose it as a way to implement the “active” behavior.
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

Handler A...

Fault entry → ... → Fault entry
ACTIVE PUTS

An RDMA packet

NIC

PCIe packets

Main memory

IOMMU

Dev-to-PT cache

IOTLB

Access log table

Stores addresses of each access log

MSI

CPU

SMT cores

Remapping structures

Dev-to-PT

Maps each page to an access log

Access log (private for each process)

Enables data-centric programming

Fault entry

Fault entry

Request data

System-wide fault log

Decide on keeping/discarding the entry/data

User handlers

Handler A

... Data can be reused
**ACTIVE PUTS**

Process p

1. Put(X)

Process q

2. Attempt to write(X)

3. Page fault! (W = 0)

Accessed page

W = 0
WL = 1
WLD = 1

Access log

Log both the entry and the data of an incoming put

4. Move(X)

5. Process(X)

CPU

Main memory

Do not modify the page

Accessed page

X

Attempt to write(X)

Page fault!
ACTIVE GETS

Enable reading from the page

Log both the entry and the data accessed by a get

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

An RDMA packet

NIC

IOMMU

Dev-to-PT cache

IOTLB

Access log table

PCIe packets

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)

Fault entry → … → Fault entry

Request data

Request data

User handlers

SMT cores

CPU

MSI

spcl.inf.ethz.ch
@spcl_eth
ETH Zürich
INTERACTIONS WITH THE CPU

- Interrupts
- Polling
- Direct notifications via scratchpad memory

Are we done?

Well…
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
- Packet tag buffer
- Access log table
- Flushing buffer

IOTLB

Contains the addresses of flushing pages

MSI

CPU

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

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

**Active Messages**

**IOMMUs**

**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**

**How can we use it?**
**ACTIVE ACCESS USE-CASES**

**DISTRIBUTED HASH TABLE**

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

---

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

Proc p

PUT (in CAS (first attempt) MMU)

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

Overflow heap

Table of elements

Proc q

All other accesses become local
ACTIVE ACCESS USE-CASES
VIRTUAL GLOBAL ADDRESS SPACE (V-GAS)

Remote memory protection

Machine 0  Machine 1  Machine N-1

Proc 0  Proc 0  Proc N-1

Machine 0  Machine 1  Machine N-1

Proc 0  Proc 1  Proc N-1

Local memory protection

MMU  MMU  MMU

Memory  Memory  Memory

IOMMU  IOMMU  IOMMU

NIC  NIC  NIC

V-GAS

Memory protection

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

Remote memory protection
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**

- Workload simulated with [1]:

  ![gem5](image)

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

---


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

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

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

### RMA
- DMAPP
- Cell

### IBM
- DCMF
- LAPI
- PAMI

### Myricom
- MX

### Mellanox Technologies
- AM++
- GASNet

---

*Images and logos*
**Performance: Large-Scale Codes**

**Distributed Hashtable**

Collisions: 5%

Collisions: 25%
CONCLUSIONS

Active Access

- Uses commodity & common IOMMUs
  - Extends paging capabilities in a distributed environment

Data-centric programming

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

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

- 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

- Logging puts:

  ![Diagram](image)
  
  Proc p
  
  PUT
  
  Log the PUT
  
  Fetch the logs
  
  Reply the PUT
  
  Proc q
  
  q is modified
ACTIVE ACCESS USE-CASES
ACCELERATING LOGGING FOR RMA

- Logging gets (naive):

  ![Diagram showing the sequence of events for naive logging]

  - Proc p
  - GET
  - Log the GET
  - Attempt to reply the GET
  - FAIL!
  - Proc q

  p is modified
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

  Proc p

  GET

  IOMMU

  Log the GET

  Fetch the logs

  reply the GET

  Proc q
ACTIVE ACCESS USE-CASES
INCREMENTAL CHECKPOINTING FOR RMA

barrier

compute

compute

compute

compute

global rollback

Proc 1 ...

Proc k ...

Proc 1 ...

Proc k ...
COORDINATED CHECKPOINTING (MP)

Node 1

Proc 1

... Barrier

Proc k

compute

compute

... Barrier

compute

compute

Node N

Proc 1

... Barrier

Proc k

compute

compute

compute

compute

global rollback
COORDINATED CHECKPOINTING (MP)
**PERFORMANCE: LARGE-SCALE CODES**

**FAULT TOLERANCE SCHEME**

Logging gets:

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

Sorting time:

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