TOWARDS ASYNCHRONOUS AND MPI-INTEROPERABLE ACTIVE MESSAGES

Xin Zhao, Darius Buntinas, Judicael Zounmevo, James Dinan, David Goodell, Pavan Balaji, Rajeev Thakur, Ahmad Afsahi, William Gropp

University of Illinois at Urbana-Champaign, Argonne National Laboratory, Queen’s University
Motivation

• Many new important large-scale applications
  • Bioinformatics, social network analysis
Motivation

• Many new important large-scale applications
  • Bioinformatics, social network analysis

• Different from “traditional” applications
  • Many small messages sent to random targets
  • Communication pattern is irregular
  • Computation is data-driven
Motivation

• Many new important large-scale applications
  • Bioinformatics, social network analysis
• Different from “traditional” applications
  • Many small messages sent to random nodes
  • Communication pattern is irregular
  • Computation is data-driven
• Approaches for “traditional” applications are not well-suited
while any rank’s queue is not empty:
  for i in ranks:
    out_queue[i] ← empty
  for vertex v in my queue:
    if color(v) is white:
      color(v) ← black
      for vertex w in neighbors(v):
        append w to out_queue[owner(w)]

for i in ranks: start receiving in_queue[i] from rank i
for j in ranks: start sending out_queue[j] to rank j
synchronize and finish communications
Active Messages (AM)

- Proposed by von Eicken et al for Split-C in 1992
- Sender explicitly sends message
- Upon message’s arrival, message handler is triggered
- Receiver is not explicitly involved
- A suitable paradigm for data-driven applications
  - Data is sent immediately
  - Communication is asynchronous
Breadth-First Search in Active Messages

while queue is not empty:
  new_queue ← empty
begin AM epoch
  for vertex v in queue:
    for vertex w in neighbors(v)
      send AM to owner(w)
  end AM epoch
end AM epoch
queue ← new_queue

AM_handler (vertex v)
if color(v) is white:
  color(v) ← black
  insert v to new_queue
Motivation

• Most applications are MPI-based, but we don’t want to rewrite the entire application
  • An incremental approach is needed to modify the application
Motivation

• Most applications are MPI-based, but we don’t want to rewrite the entire application
  • An incremental approach is needed to modify the application

• This paper tackles the challenge of supporting Active Messages within MPI framework, so that both traditional MPI communication and Active Messages can be simultaneously utilized by applications
API DESIGN
Challenges

• Active Messages should work correctly with other MPI messages
• Memory consistency semantics
• Consistency between two different active messages
Challenges

• Active Messages should work correctly with other MPI messages
• Memory consistency semantics
• Consistency between two different active messages
• **Leverage MPI One-sided (RMA) Interface, which already address those challenges**
MPI RMA Interface

• “One-sided” communication
  • *Origin process* specifies all communication parameters
  • *Target process* exposes memory ("window") accessed by other processes
• Three basic operations: Put / Get / Accumulate
• Accumulate: simple updates on target process

![Diagram](image-url)
API Design

• Extend MPI_Accumulate to support user function
  • User-defined function
    • *MPI_User_function (void *invec, void *inoutvec, int *len, MPI_Datatype *dtype)*
API Design

• Extend MPI_Accumulate to support user function
  • User-defined function
    •  
  
• Operation creation
  •  


MPI_User_function (void *invec, void *inoutvec, int *len, MPI_Datatype *dtype)

MPI_Op_create (MPI_User_function *user_fn, int commute, MPI_Op *user_op)
API Design

• Extend MPI_Accumulate to support user function
  • User-defined function
    • `MPI_User_function (void *invec, void *inoutvec, int *len, MPI_Datatype *dtype)`
  • Operation creation
    • `MPI_Op_create (MPI_User_function *user_fn, int commute, MPI_Op *user_op)`
  • Operation registration
    • `MPIX_Op_register (MPI_Op user_op, int id, MPI_Win win)`
    • Collective call on the window
Example

void func0 (void *in, void *inout, int *len, MPI_Datatype *dtype);

MPI_Win_create (…, &win);
MPI_Op_create (func0, …, &user_op);
MPIX_Op_register (user_op, 0, win);

MPI_Win_lock (…, 1, …, win);
MPI_Accumulate (…, 1, …, user_op, win);
MPI_Win_unlock (1, win);

MPIX_Op_deregister (user_op, win);
MPI_Win_free (&win);

rank 0

void func1 (void *in, void *inout, int *len, MPI_Datatype *dtype);

MPI_Win_create (…, &win);
MPI_Op_create (func1, …, &user_op);
MPIX_Op_register (user_op, 0, win);

MPIX_Op_register (user_op, 0, win);

MPIW_Accumulate (…, 1, …, user_op, win);
MPI_Win_unlock (1, win);

MPIX_Op_deregister (user_op, win);
MPI_Win_free (&win);

rank 1

func0 and func1 have the same functionality

func1 is triggered at rank1

/* func1 is triggered at rank1 */
ASYNCHRONOUS PROCESSING
Asynchronous execution models

• “NON-ASYNC”
  • No asynchronous processing, messages are processed by single thread
  • Disadvantage: messages cannot be processed upon arrival
Asynchronous execution models

• “NON-ASYNC”
  • No asynchronous processing, messages are processed by single thread
  • Disadvantage: messages cannot be processed upon arrival

• “THREAD-ASYNC”
  • Asynchronous processing is provided by thread on top of MPI library
  • Disadvantage: “active polling” for intra-node messages
Asynchronous execution models

• “NON-ASYNC”
  • No asynchronous processing, messages are processed by single thread
  • Disadvantage: messages cannot be processed upon arrival

• “THREAD-ASYNC”
  • Asynchronous processing is provided by thread on top of MPI library
  • Disadvantage: “active polling” for intra-node messages

• “INTEGRATED-ASYNC”
  • Asynchronous processing is supported internally from MPI library
Inter-node Communication

- Thread-based approach
  - Spawn thread in TCP network module
  - Block polling for AM
  - Separate sockets for MPI messages and AM
Intra-node Communication

• “Origin computation”
  • During window creation, processes on the same node allocate a shared-memory region
  • Origin process directly fetches data from target process, completes computation locally and writes data back to target process
Integrated Idea

NODE 0

rank 0
(target)

rank 1
(origin)

Shared-memory

Network

NODE 1

rank 2
(origin)
Integrated Idea

Network

Integrated Idea

NODE 0
NODE 1

rank 1
(origin)

rank 0
(target)

rank 2
(origin)

Shared-memory

Network
Integrated Idea

NODE 0

NODE 1

Shared-memory

Network

rank 0 (target)

rank 1 (origin)

rank 2 (origin)
Integrated Idea

NODE 0
NODE 1

rank 1 (origin)  rank 0 (target)  rank 2 (origin)

Shared-memory
Network
Integrated Idea

NODE 0

Shared-memory

NODE 1

Network

rank 1 (origin)

rank 0 (target)

rank 2 (origin)
Integrated Idea

NODE 0

rank 0 (target)

rank 1 (origin)

NODE 1

rank 2 (origin)

Shared-memory

Network
PERFORMANCE
Stencil Benchmark

Number of processes: 2x2 to 20x20, 8 processes per node
(“Fusion” Cluster at ANL: 320 nodes, 8 cores per node, QDR Infiniband)
Graph500 Benchmark

- Breadth-First Search (BFS)
- Large number of small messages to random targets
- Optimization: origin process accumulates local data and sends only one message to each target process

(a) Small problem size (#vertices = $2^{15}$)
(b) Large problem size (#vertices = $2^{20}$)

(TEPS: Traversed Edges Per Second, higher is better)
Conclusion

• Why to support Active Messages in MPI
• Leverage MPI RMA Interface
• API Design
• Asynchronous processing
  • Intra-node messages
  • Inter-node messages
• Stencil benchmark and Graph500 benchmark
QUESTIONS
BACKUP SLIDES
Message Passing Models

• Two-sided communication
  • Explicit sends and receives

• One-sided communication
  • Explicit sends, implicit receives
  • Simple operations on remote process

• Active Messages
  • Explicit sends, implicit receives
  • User-defined operations on remote process
Active Message Models

- Libraries and runtime systems
  - Low level: GASNet, IBM’s LAPI, DCMF and PAMI, AMMPI
  - High level: Charm++, X10
  - Middle: AM++
MPI RMA Interface

- Two synchronization modes
Inter-node Communication

• Separate sockets for Active Messages
Inter-node Communication

• Separate sockets for Active Messages

main thread

AM thread

origin_sc → rma_send_sc

rma_send_sc → rma_recv_sc

rma_send_sc → rma_resp_sc

rma_recv_sc → rma_resp_sc

rma_recv_sc → origin_sc

rma_resp_sc → rma_recv_sc

rma_resp_sc → origin_sc