Efficient MPI Support for Advanced Hybrid Programming Models

Torsten Hoefler, Greg Bronevetsky, Brian Barrett,
Bronis R. de Supinski, and Andrew Lumsdaine

EuroMPI 2010, Stuttgart, Germany, Sep. 13th 2010

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Threaded/Hybrid MPI Programming

- Hybrid Programming gains importance
  - Reduce surface-to-volume (less comm.)
  - Will be necessary at Peta- and Exascale!
- MPI supports hybrid programming
  - Offers thread levels:
    - single, serial, funneled, multiple
  - Thread_multiple becomes more common
    - E.g., codes using OpenMP tasks

MPI Messaging Details

- MPI_Probe to receive messages of unknown size
  - MPI_Probe(…, status)
  - size = get_count(status)*size_of(datatype)
  - buffer = malloc(size)
  - MPI_Recv(buffer, …)
- MPI_Probe peeks in matching queue
  - Does not change it → stateful object

Multithreaded MPI Messaging

- Two threads, A and B perform probe, malloc, receive sequence
  - $A_P \rightarrow A_M \rightarrow A_R \rightarrow B_P \rightarrow B_M \rightarrow B_R$
- Possible ordering
  - $A_P \rightarrow B_P \rightarrow B_M \rightarrow B_R \rightarrow A_M \rightarrow A_R$
  - Wrong matching!
  - Thread A's message was "stolen" by B
  - Access to queue needs mutual exclusion ☹

“Obvious” Solution 1

- Separate threads with “channels”
  - Needs t*p threads or communicators
    - Not scalable
  - Threads cannot “share” messages
    - Not flexible for load-balancing (master/worker)
  - Problems with libraries
    - Each needs t*p tags or communicators
- This solution is impractical!

“Obvious” Solution 2

- Lock each P,M,R sequence
  - Unnecessary synchronization
  - This sequence might be slow (malloc)
    - Only one thread can perform it
  - Observation:
    - E.g., (tag,src)=(4,5) and (5,5) do not “conflict”

Solution 3 – 2d Locking

- Lock each (src,tag) pair
  - Requires 2d lock matrix
    - Should be sparse!

```
lock (src, tag)
P,M,R (e.g., irecv)
unlock(src,tag)
```

  - Wildcards (ANY_SRC, ANY_TAG) acquire locks for whole row/column or matrix
  - Minimizes lock overhead

Solution 3 is incorrect ☹

- Can lead to deadlocks
  - A correct MPI code (threads A+B):

```
A:                              A:
  send(..., 1, 1, comm)           probe/recv(0, 2, comm)
  recv(..., 1, 1, comm)         B:
  send(..., 1, 2, comm)           probe/recv(0,ANY_TAG,comm)
  ...                             send(..., 0, 1, comm)
```

  - Thread A enters locks (0,2), B is waiting forever (deadlock)

Updated Solution 3

- Obvious fix: don't block, poll ☹
  - Only needed if code uses wildcards
  - Several variants:

Scenario	any_src	any_tag	Specific	Strategy
1	-	-	x	simple 2d, blocking
2	-	x	-	simple 1d, blocking
3	-	x	x	2d lock, polling
4	x	-	-	simple 1d, blocking
5	x	-	x	2d lock, polling
6	x	x	-	2d lock, polling
7	x	x	x	2d lock, polling

Solution 4 - Matching Outside MPI

- Helper thread calls MPI_Probe
  - Receives all incoming messages
  - Full matching logic on top of that
    - Replicating MPI logic (thread safe)
- Allows blocking on MPI calls
  - High overhead though

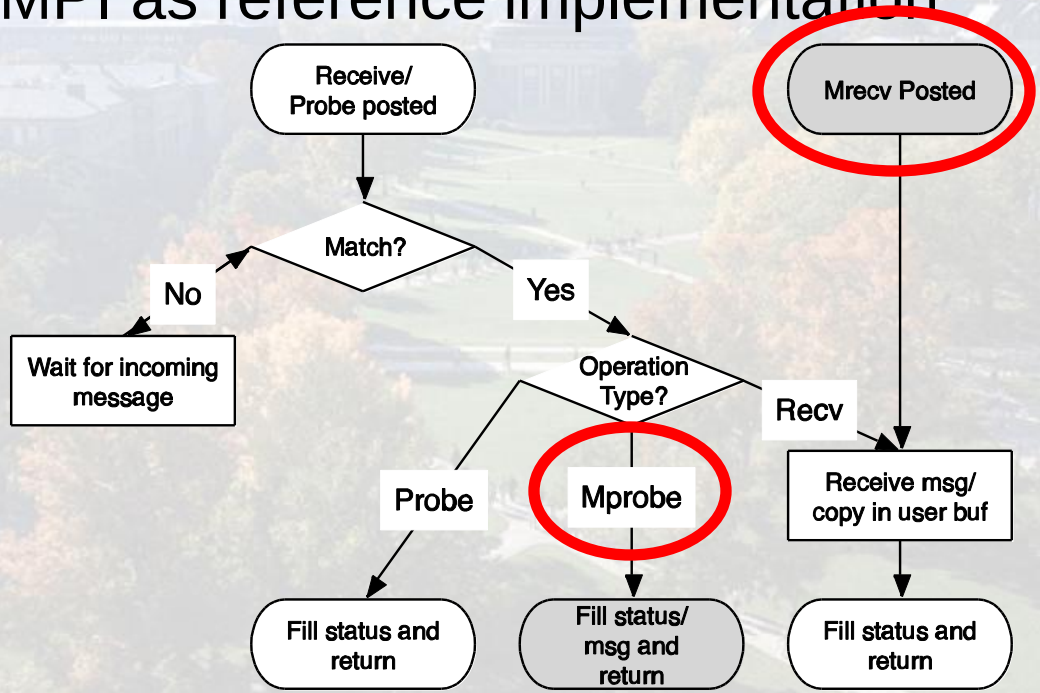
Fixing the MPI Standard?

- Avoid state in the library
  - Return handle, remove message from queue

```
MPI_Message msg; MPI_Status status;
/* Match a message */
MPI_Mprobe(MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
                          &msg, &status);
/* Allocate memory to receive the message */
int count; MPI_get_count(&status, MPI_BYTE, &count);
char* buffer = malloc(count);
/* Receive this message. */
MPI_Mrecv(buffer, count, MPI_BYTE, &msg, MPI_STATUS_IGNORE);
```

Implementation

- Open MPI as reference implementation



- Low-level matching (e.g., MX) will need FW support

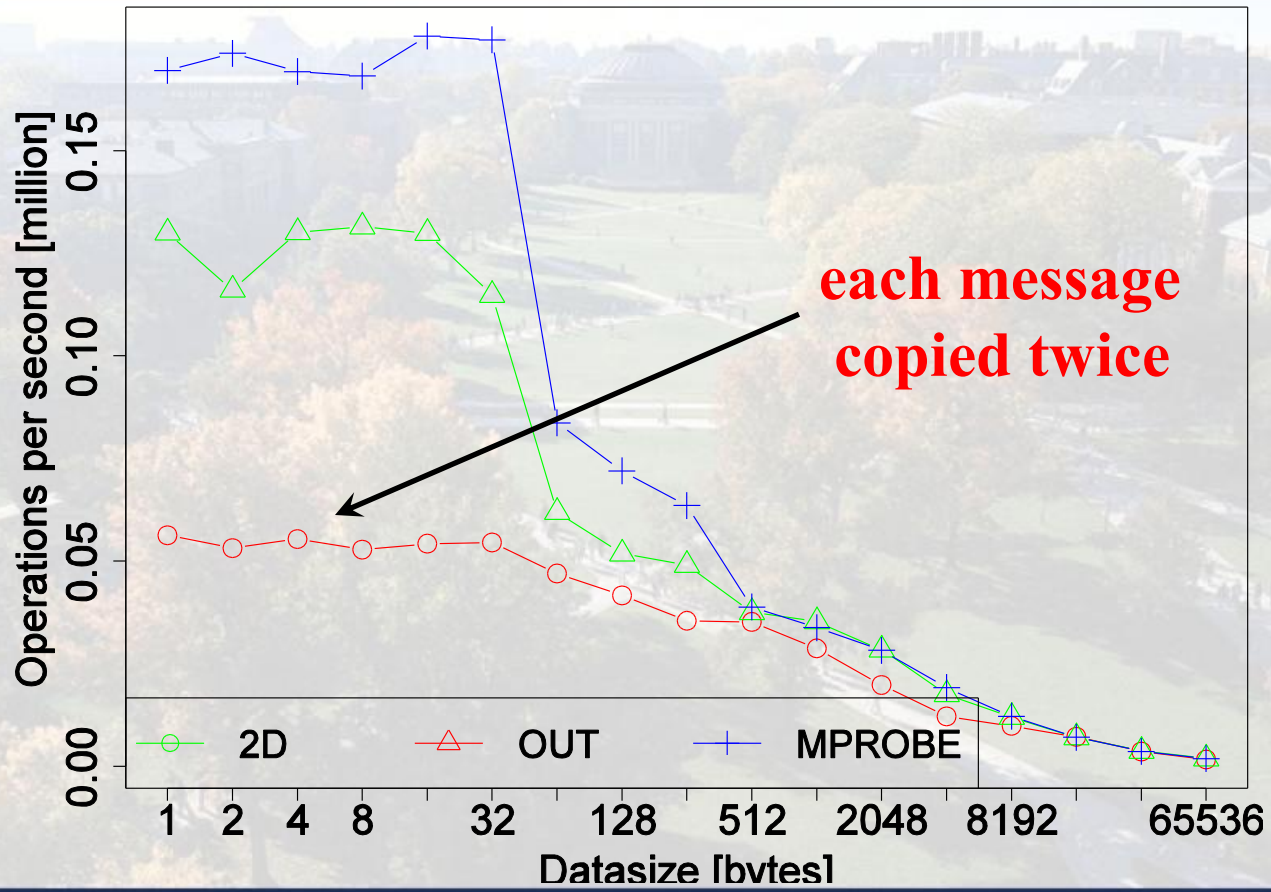
Test System

- Sif at Indiana University
  - Eight core 1.86 GHz Xeon
  - Myrinet 10G (MX)
  - Open MPI rev. 22973 + mprobe patch
    - `--enable-mpi-thread-multiple`
    - Using MPI_THREAD_MULTIPLE with TCP BTL

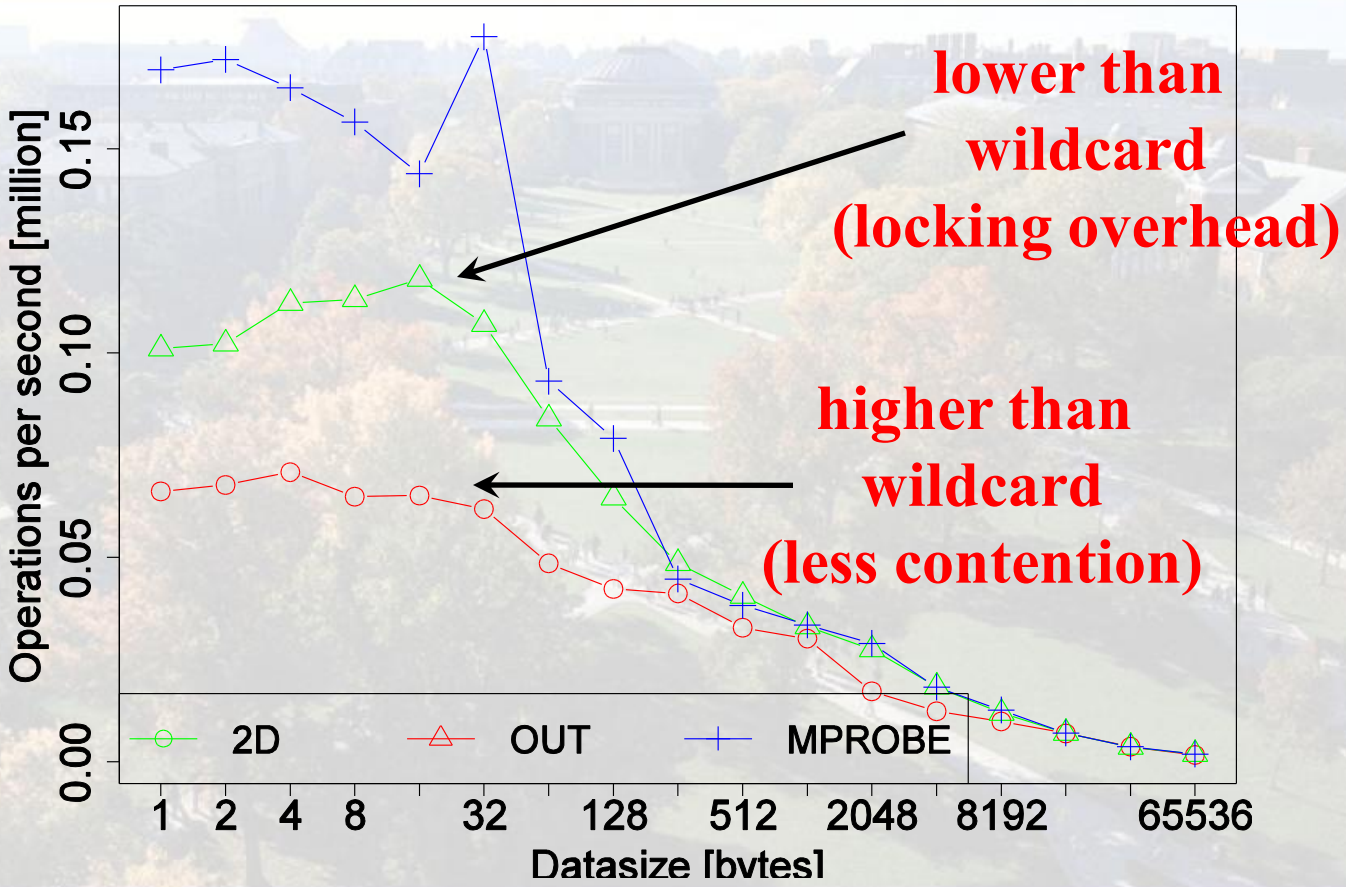
Benchmarks

- Receive Message Rate
  - MT receive (j processes send to j threads)
    - 2d locking (2D)
    - Outside MPI matching (OUT)
    - Mprobe reference (MPROBE)
- Threaded Roundtrip Time
  - Send n RTT messages between threads
  - Report average latency

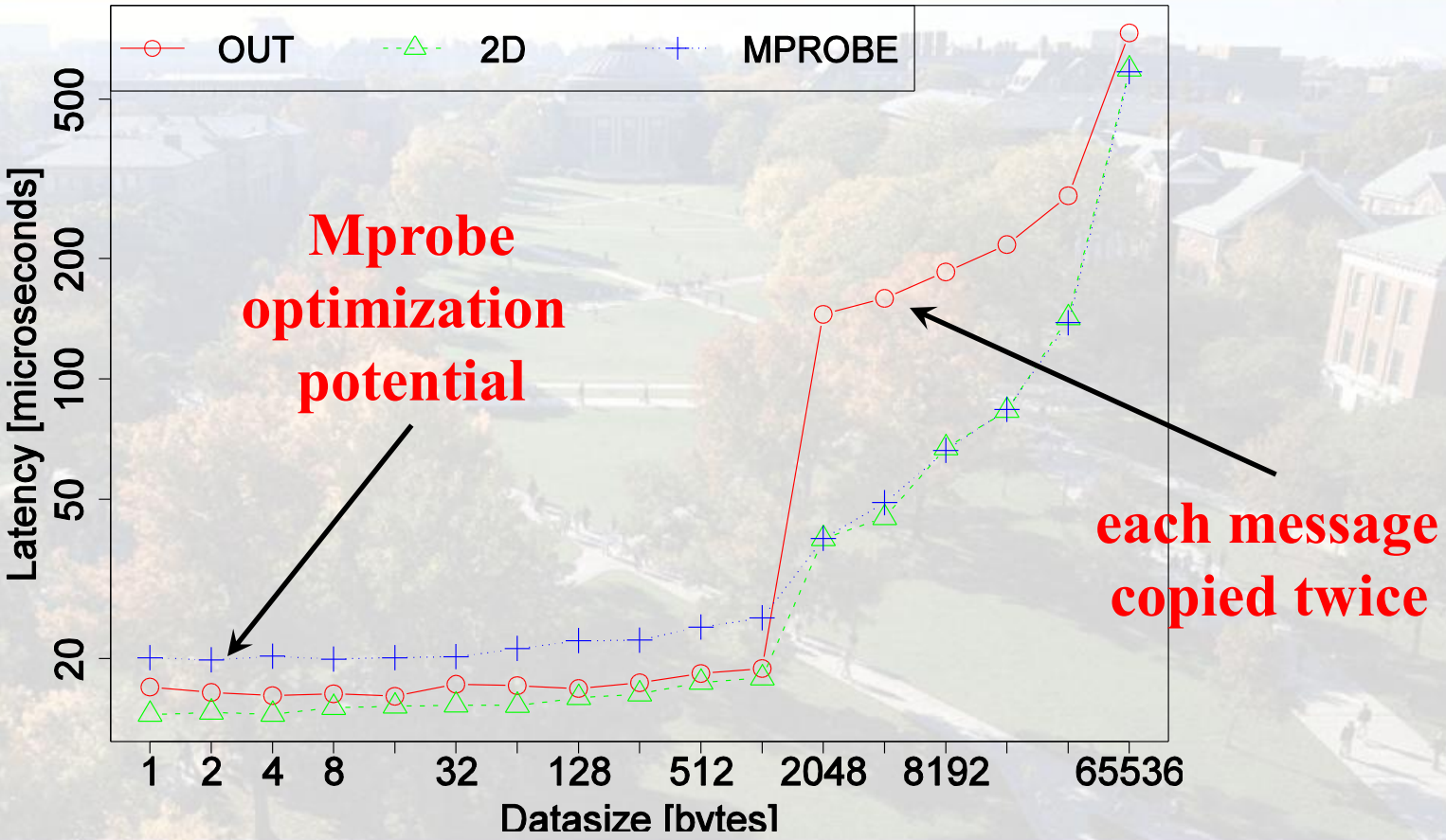
ANY_SRC, ANY_TAG Receive



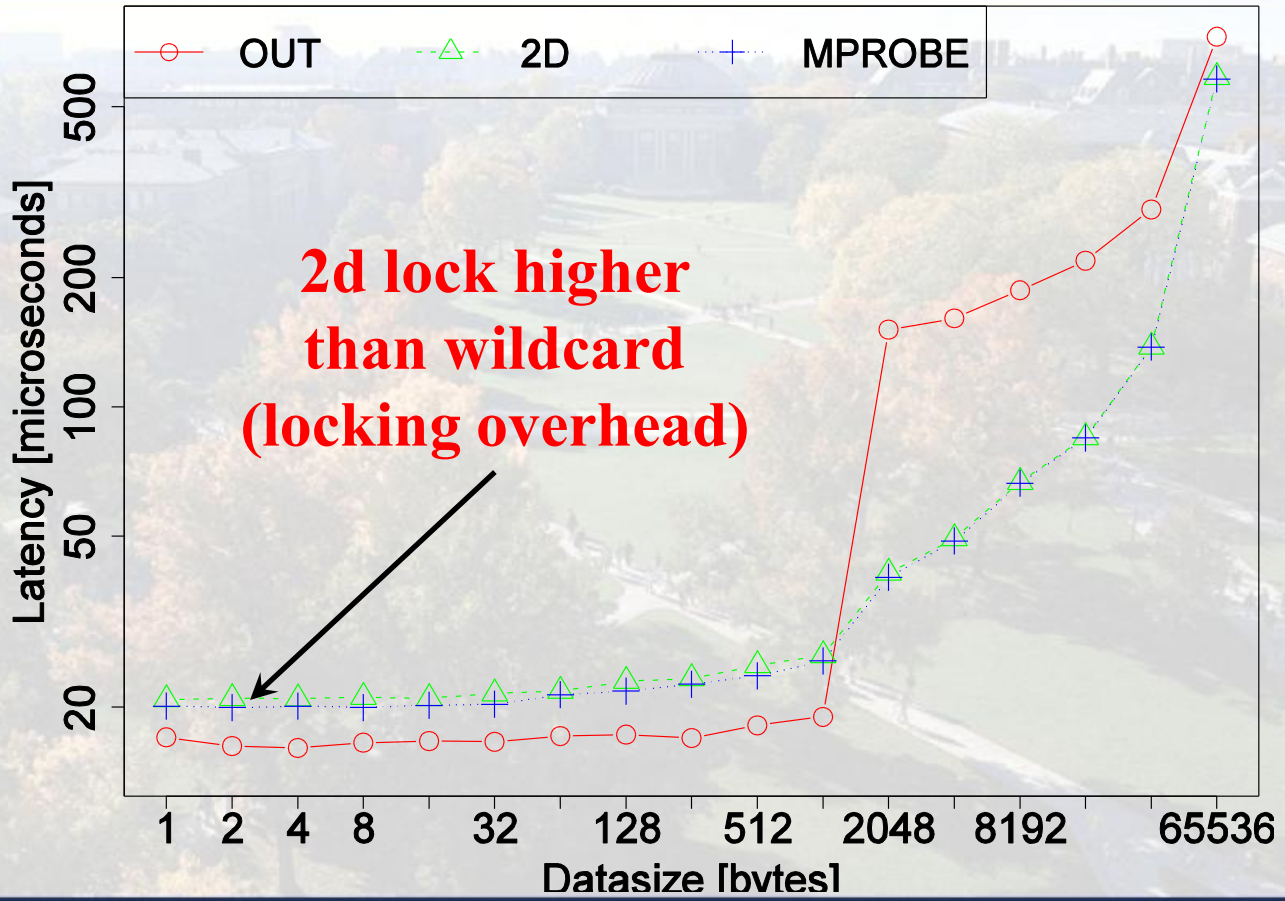
Directed Receive



ANY_SRC, ANY_TAG Latency



Directed Latency



Conclusions

- MPI_Probe is not thread-safe
  - Arguably a bug in MPI-2.2
- Obvious solutions do not help
  - Resource exhaustion
- Complex solutions are tricky
  - Too complex for average MPI user
- Change to standard to add stateless interface
  - Mprobe proposal under consideration for MPI-3
  - Encouraging initial performance results!