Implications of a PIM Architectural Model for MPI

Keith Underwood

Sandia National Laboratories

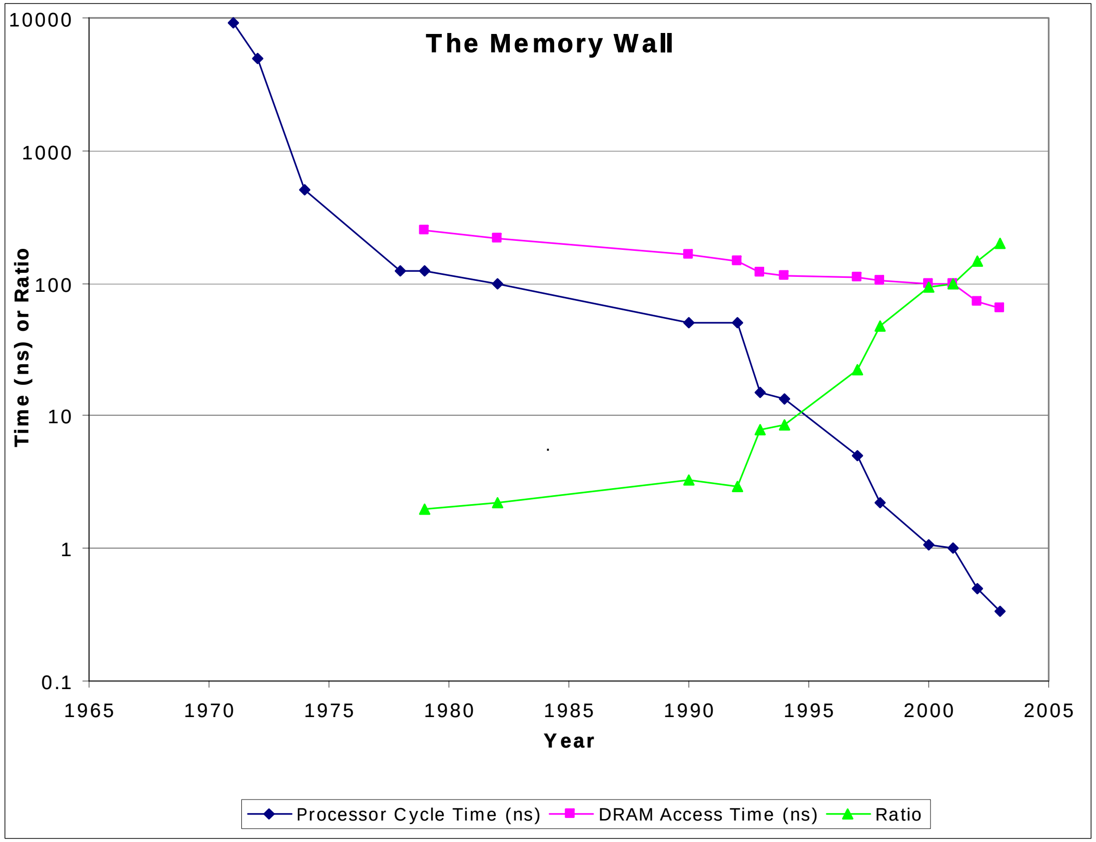
Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

Overview

- Growing use of COTS components in HPC (price pressure)
- Increasing gap between processor & memory speed
- Novel architectures on the horizon, but a huge investment exists in current parallel codes
- Next-next-Generation must…
  - … be high volume (commodity)
  - … break the memory wall
  - … support current applications

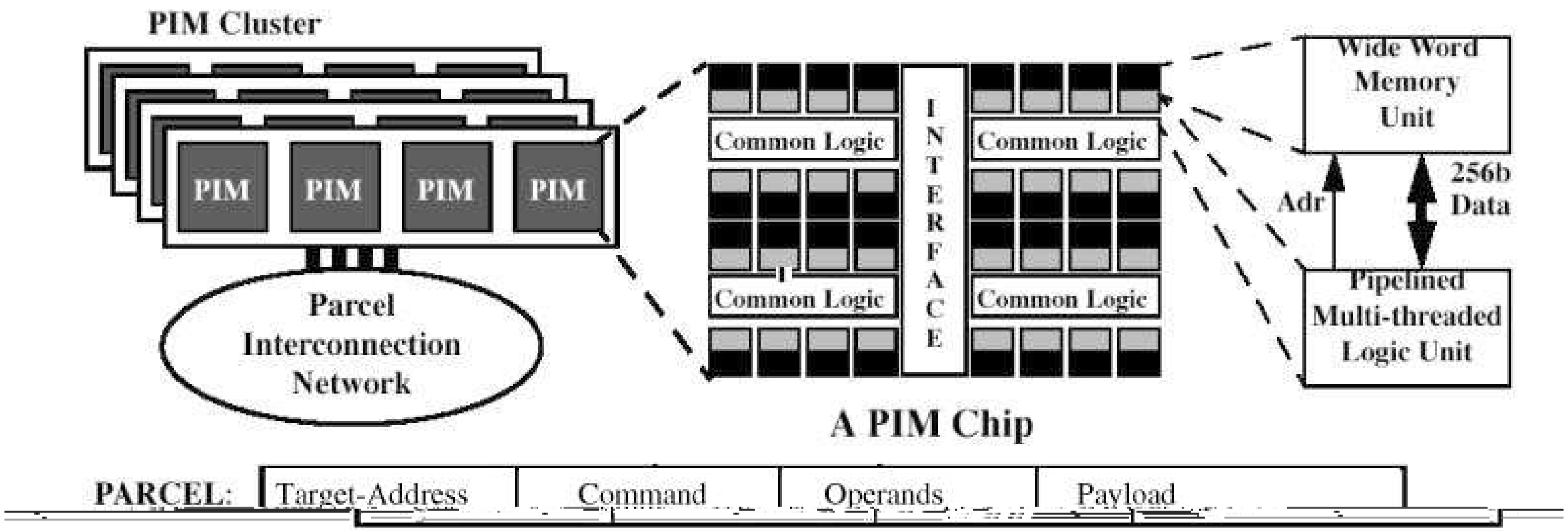
The Memory Wall

- Processor clock speed has grown dramatically
- Main memory access (DRAM) speed has grown much more slowly
- Caches and prefetching help, but problem is fundamental



PIM Systems

- Merge processing and memory on same die
- Very low latency to memory (< 10 ns)
- Simple, small, cheap multithreaded processor
- PIMs + interconnect = All-PIM supercomputer



PIM Programming Model

- Pervasive Multithreading
  - Hardware support for thread creation
  - Low-cost synchronization
  - Makes threading CHEAP!
- Traveling threads
  - Threads can migrate from PIM to PIM as required to access memory
  - Hardware support for thread state packaging and continuation on remote node
  - Send the thread to the data, not the data to the thread

MPI For PIM Implementation

- Subset of MPI 1.2
  - No collectives (other than MPI_Barrier)
  - MPI_COMM_WORLD only
  - Basic datatypes only
- Explore PIM-specific issues
  - Multi-threading
  - Thread migration

PIM Effects

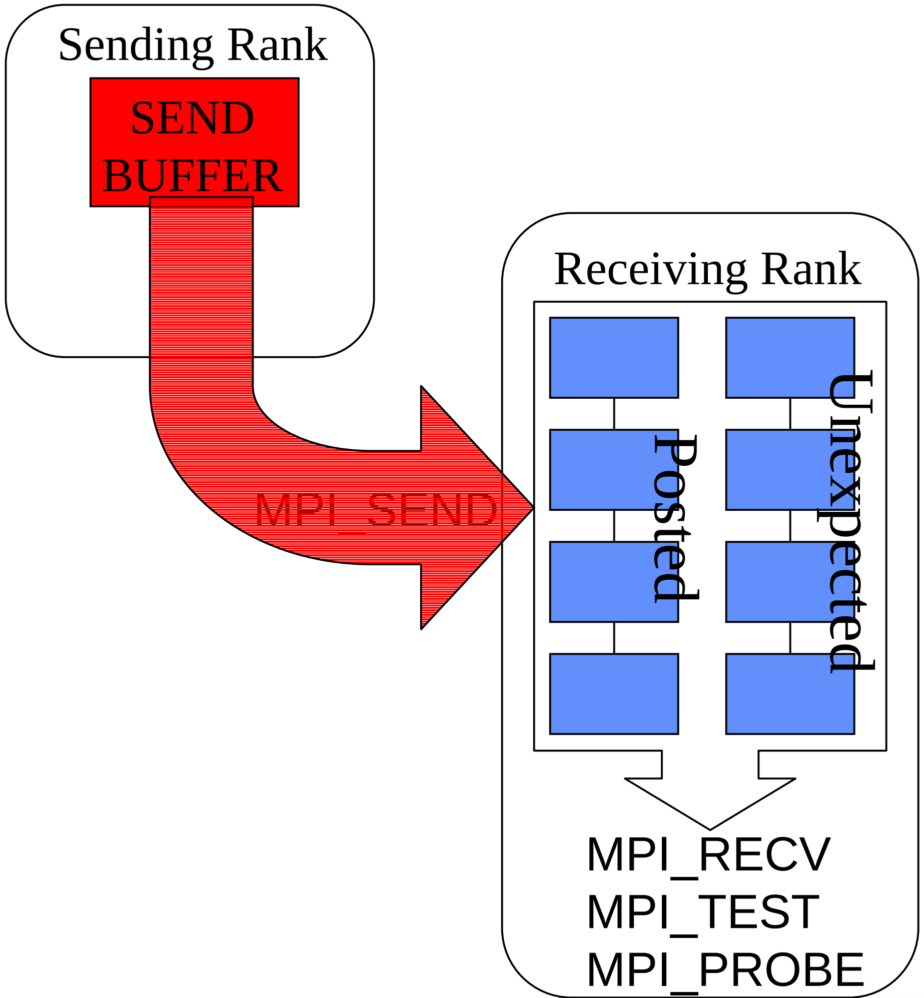
- Avoid “juggling”
  - Each Request assigned a thread
  - No central thread which must iterate through each active and pending request
- Hide latency
  - High latency tasks (i.e. memcpy()) divided between many threads
- Simplified implementation
  - Traveling thread sends data + execution
  - Receiving node does not have to interpret incoming message. Message can “look after itself”

Related Work

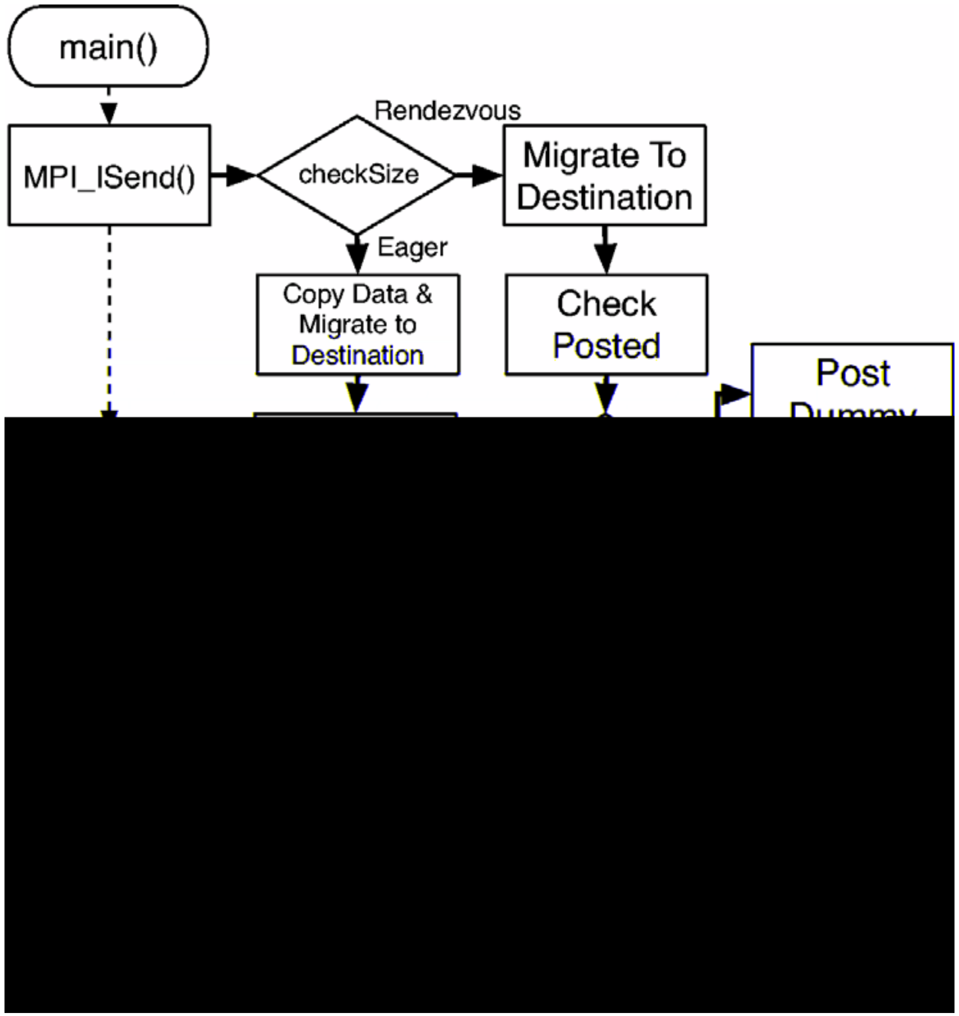
- Similar to MPI on Active messages
  – MPI-FM, MPI-LAPI, MPI-AM
- But, MPI-PIM does not require polling
  – Saves processor cycles
  – Maintains MPI progress rule
- Similar to RDMA MPIs
  – Infiniband, MVICH, T3D
- Incoming messages do not require processing by MPI library
- Traveling threads can "process themselves"

Basic Operation

- A thread is spawned to perform the MPI function
  - A “send thread” migrates to deliver data
  - Recv/Test threads check local state
- Posted and Unexpected Queues coordinate between threads



Isend() Implementation

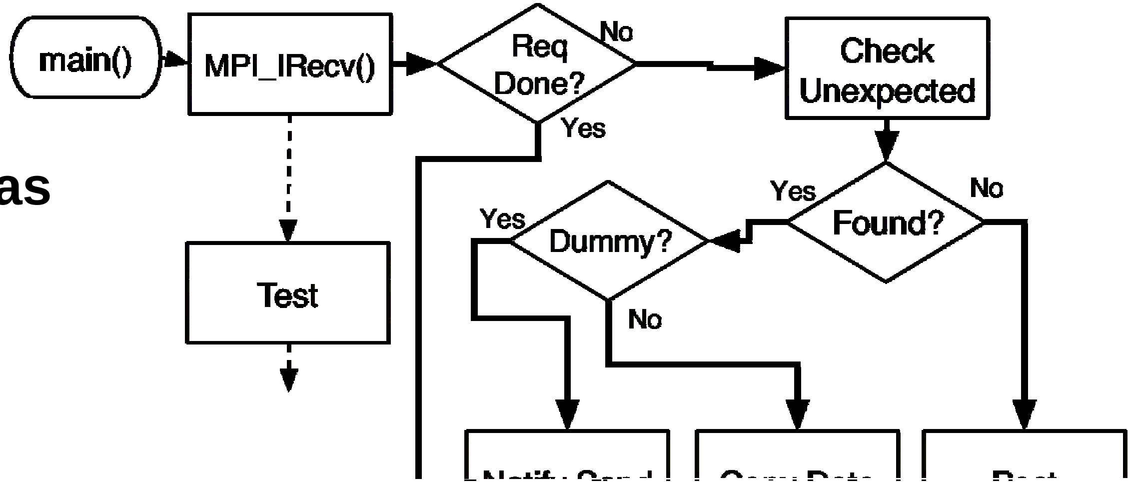


[redacted]

- **Thread spawned to perform send**
- **Rendezvous**
  - **Migrate to destination**
  - **Find posted buffer, or wait for one**
  - **Retrieve data & deliver**
- **Eager**
  - **Copy data, migrate**
  - **Deliver to posted, or post to unexpected**

IRecv()/Probe()

- IRecv()
  - Check if request has been completed
  - Check unexpected queue
- Probe()
  - Check unexpected Queue
- Ordering semantics preserved by locking

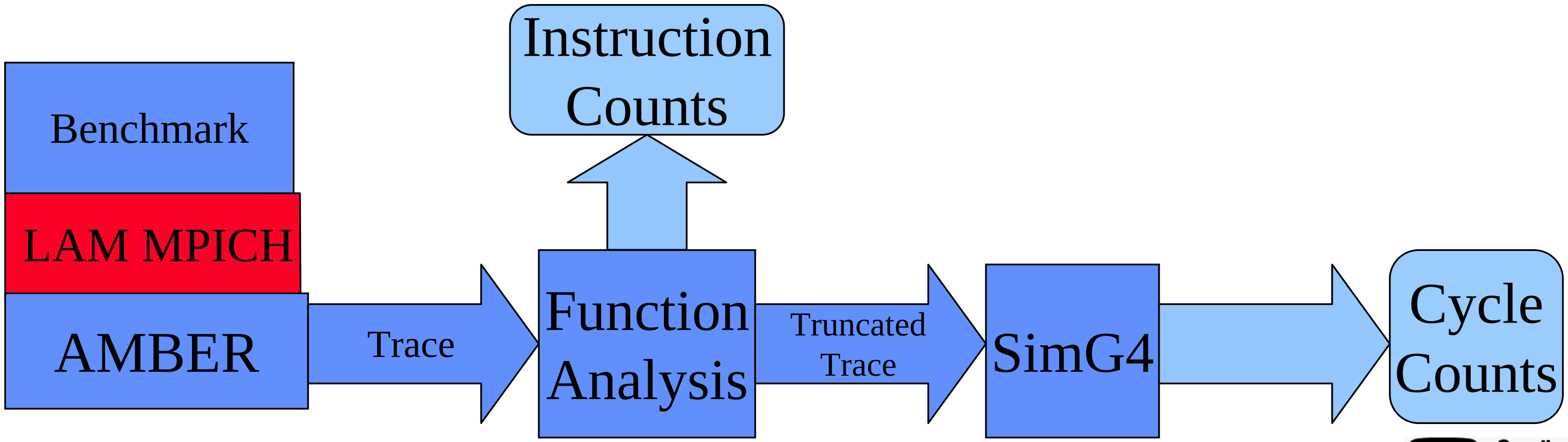


Test Methodology

- Compare MPI for PIM against LAM MPI and MPICH
  - LAM/MPICH traced on PowerPC, simulated on cycle accurate PowerPC simulator
  - MPI for PIM executed on PIM Simulators
- Focus on MPI overhead
  - OS & Network effects removed
  - Features not supported by MPI for PIM removed (error checking, user data types, etc.)

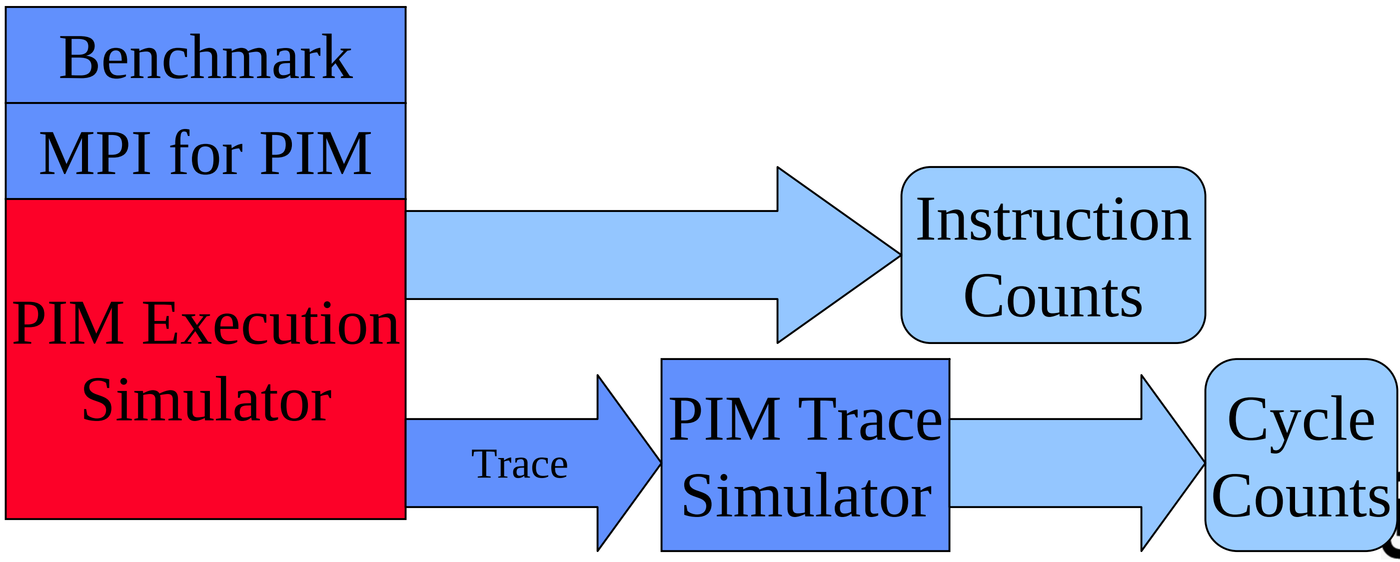
LAM MPICH

- Amber captures trace
- Function Analysis
  - Functions which concern features not present in MPI for PIM are removed (user data types, communicator lookup)
  - Functions categorized (setup, queue handling, etc…)
- SimG4: Cycle accurate G4 simulator from Motorolla



MPI for PIM

- PIM Execution Simulator
  - MPI for PIM code annotated to categorize code
  - SimpleScalar with PIM-specific extensions for threading and synchronization
- PIM Trace Simulator
  - More detailed simulation



Processor Comparison

- G4 (MPC7400)
  - Superscalar OOO processor with vector unit
  - Traditional memory hierarchy
- PIM
  - Interwoven 1-wide multithreaded processor
  - Traveling thread execution model

	G4	PIM
Memory Latency, Open Page	20	4
Memory Latency, Closed Page	44	11
L2 Latency	6	N/A
Pipelines	7	1
Pipeline Depth	4 (Integer)	4 (Interwoven)

mpi-bw Benchmark

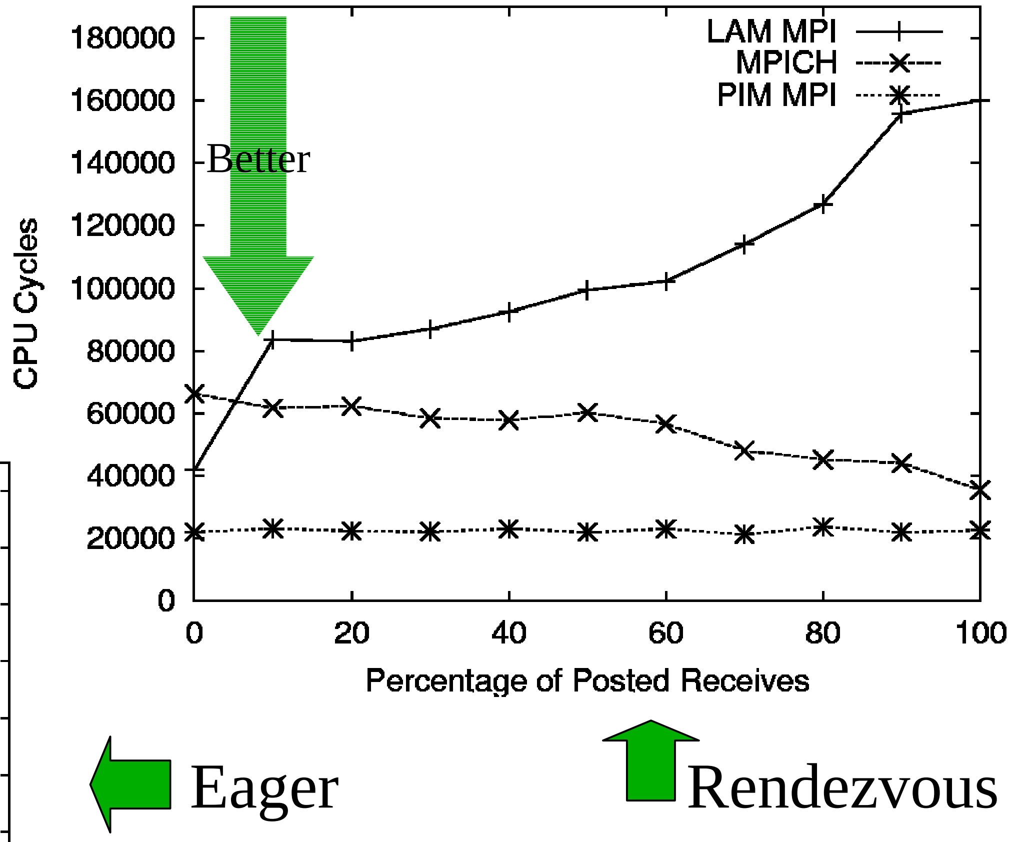
- Mpi-bw written to consider the effect of posted vs. unexpected messages
- Exercises a common subset of MPI

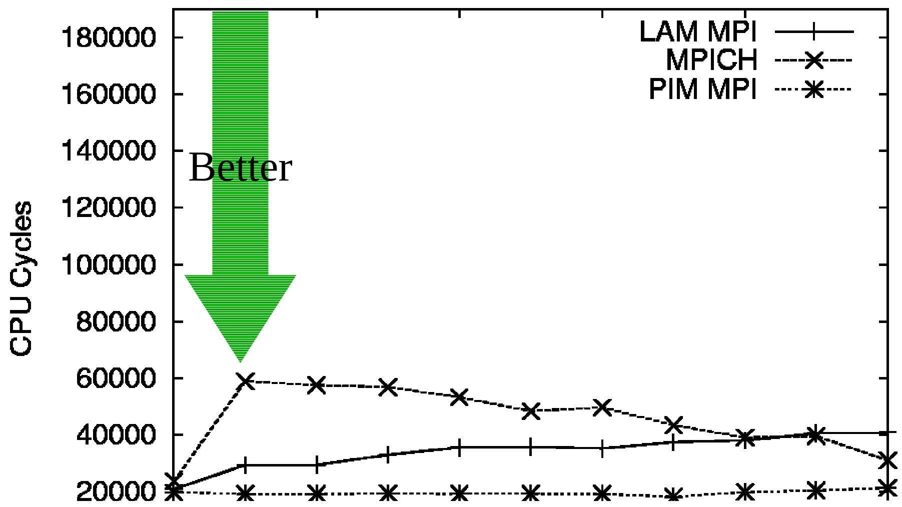
```
for (post=0; post<=10; post++) {
  if (rank == 0) {
   for (i = 0; i < post; i++)
     MPI_Irecv(rmsg_buf[i], ...
   MPI_Barrier(MPI_COMM_WORLD);
   for (i = 0; i < 10; i++)
     MPI_Send(smsg_buf[i], ...
   for (i = post; i < 10; i++) {
     MPI_Probe(1, 0, ...
     MPI_Recv(rmsg_buf[i], ...
   }
  }
}
```

Abbreviated mpi-bw

MPI Performance: Overhead

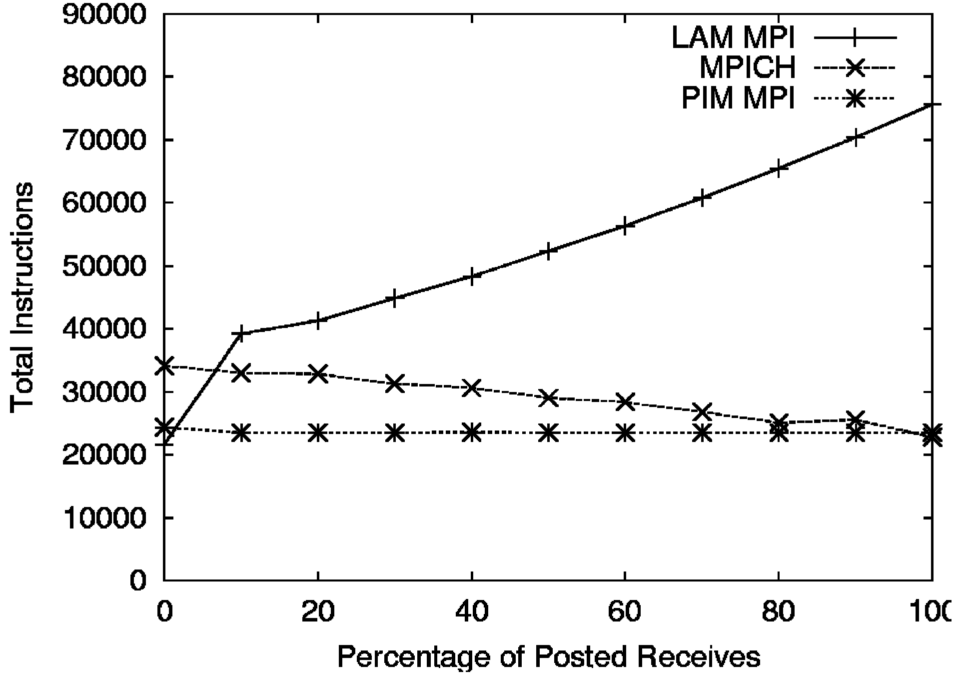
- PIM MPI requires fewer cycles for MPI overhead
- More consistent over % posted

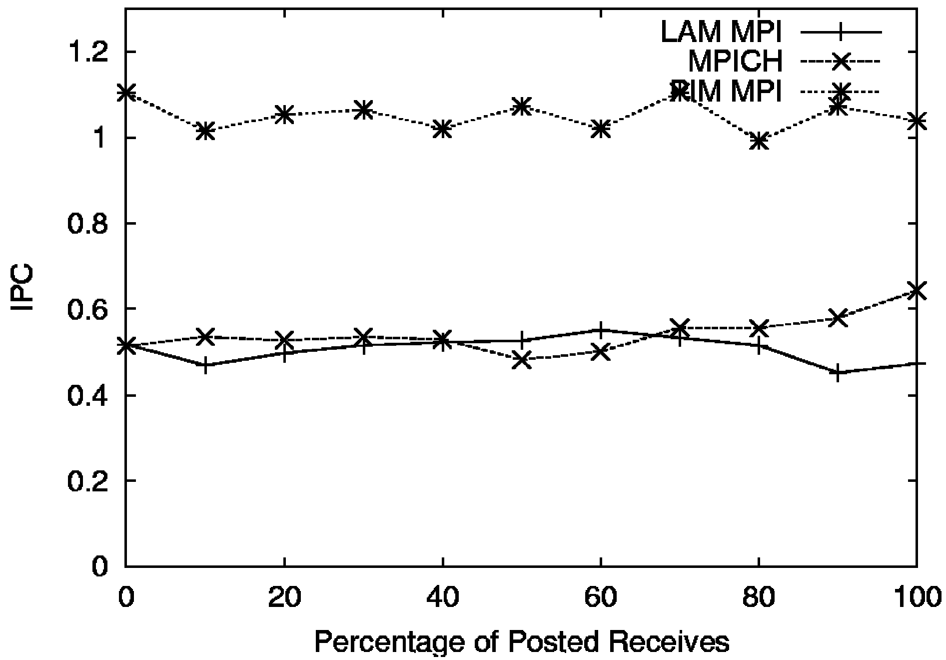




Performance: Overhead

- **MPI-PIM requires fewer instructions**
- **MPI-PIM IPC much higher**

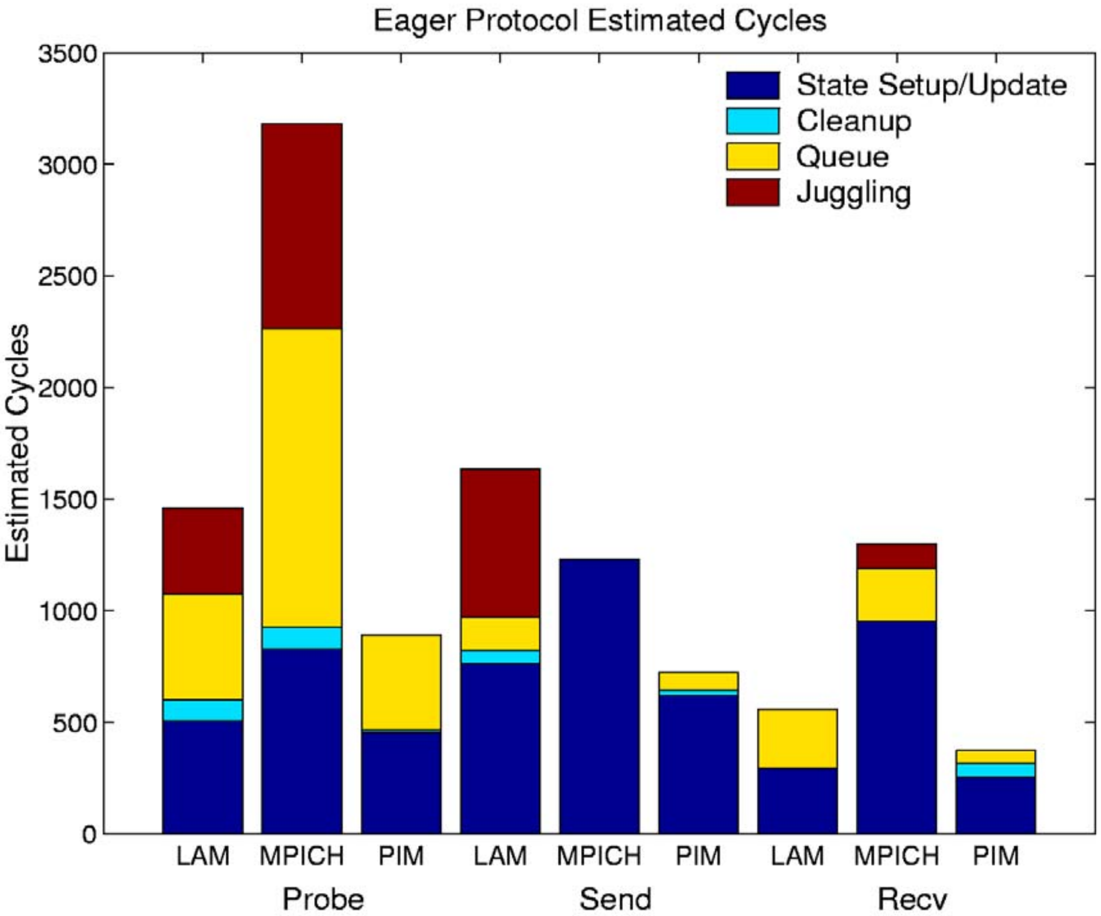




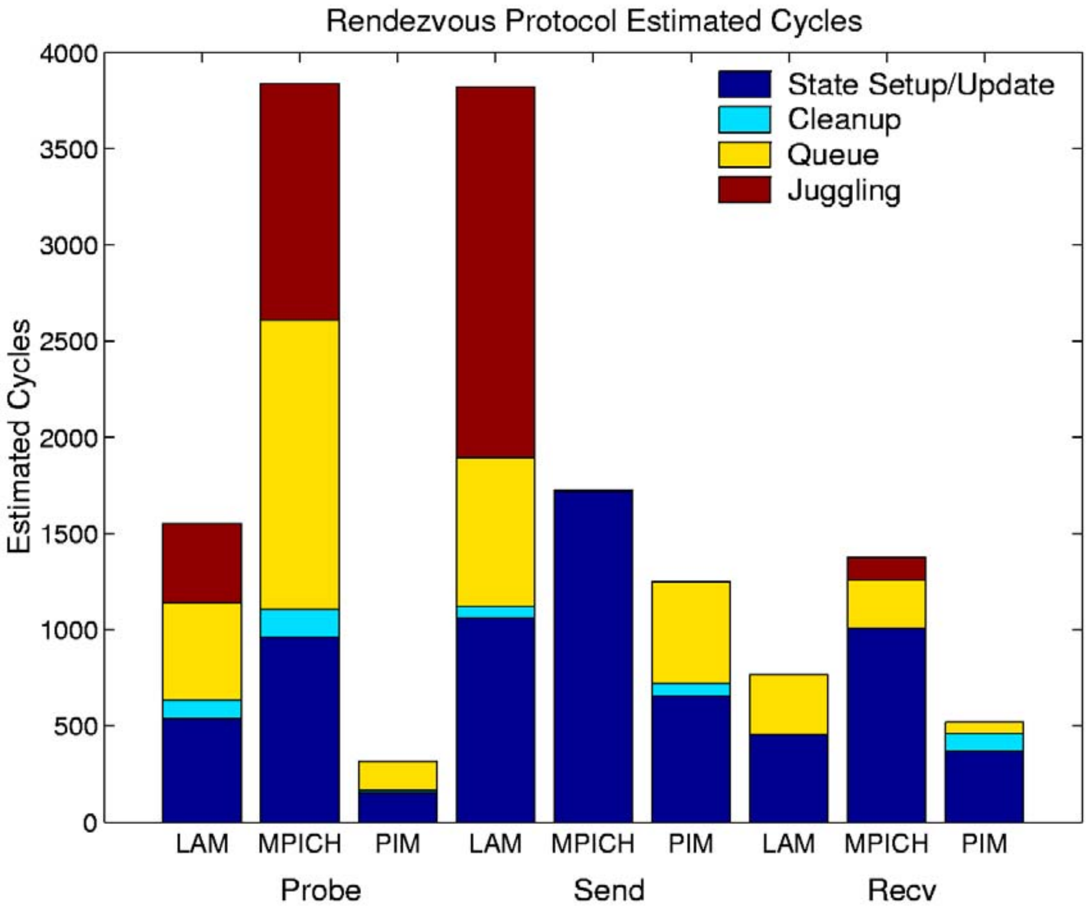
Sandia National Laboratories

Function Analysis

- **Function Categories**
  - **Setup: State initialization and Update**
  - **Cleanup: Resource deallocation**
  - **Queue: inserting, interating through lists**
  - **Juggling: MPI context switches**
- **PIM removes "Juggling"**



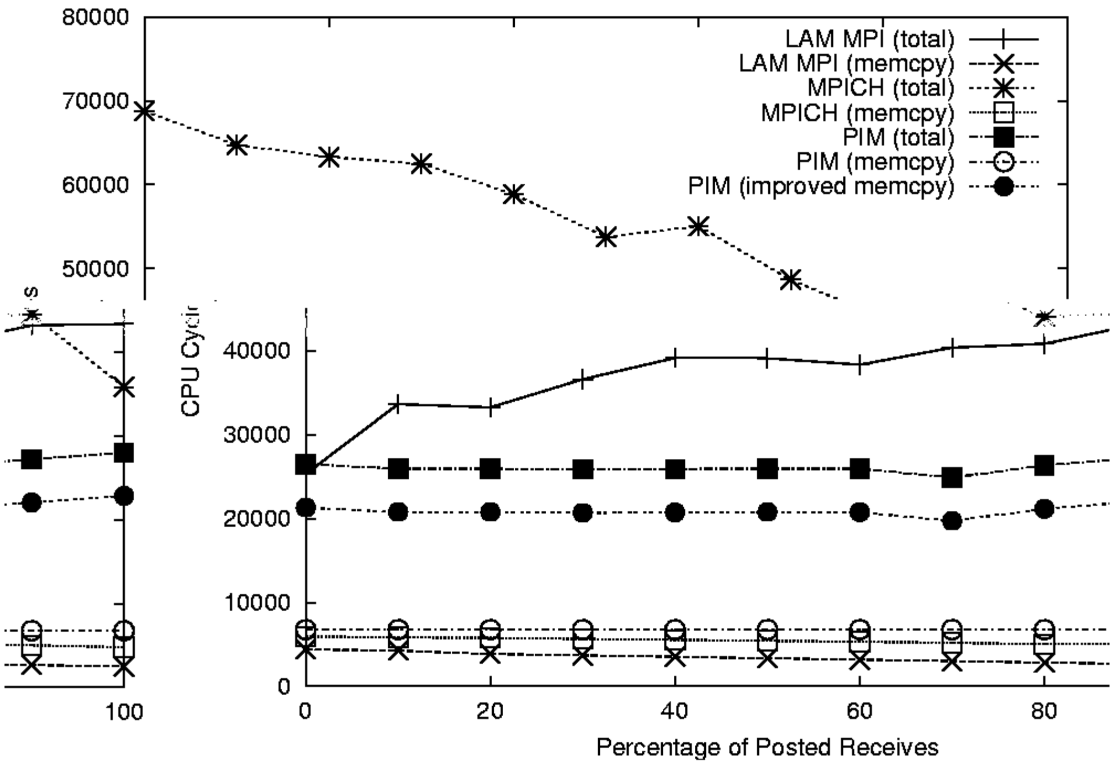
Overhead: PIM Benifits



- **PIM avoids juggling**
- **Higher IPC**
  - **Faster memory access**
  - **Fewer branch penalties**
- **Traveling thread decreases state setup**

Other Benefits

- Massive on-chip bandwidth allows fast memcpy()
  - Closer to memory
  - Can copy entire open row at a time
- Reduce negative effects of unexpected messages



Conclusion

- MPI maps onto PIM effectively
- Performance improved by
  - Multithreading
  - Traveling threads
  - Fast memory copy
- Implementation simplified

Future Work

- Expand to full MPI implementation
- More detailed simulation
- Optimized collectives
- One-sided communication

Sandia National Laboratories

Acknowledgments

- Arun Rodriques and Rich Murphy
  - University of Notre Dame students
  - The guys who did the work
- Peter Kogge and Jay Brockman
  - University of Notre Dame faculty
  - The guys who provided the students