

# **A Performance Comparison of Myrinet Protocol Stacks**

**Ron Brightwell, Bill Lawry,  
Mike Levenhagen, and Rolf Riesen**  
Sandia National Labs

**Arthur B. Maccabe**  
University of New Mexico

# Outline

---

- Objective
- COMB benchmark suite
- Hardware
- Software
- Results and analysis
- Summary

# Objective

---

- Point of this paper was to demonstrate the utility of the COMB benchmark suite in analyzing different protocol stacks
- Actual performance data is really secondary



# Communication Offload MPI Benchmark(COMB)

---

- **Measures the ability of an MPI implementation to overlap computation and communication**
- **Ability to overlap related to**
  - **Quality of MPI implementation**
  - **Capabilities of the underlying network layers**
- **Provides insight into the relationship between network performance and host CPU performance**
- **Needed to quantify the benefit of “application offload” in Portals**

# COMB Design Goals

---

- **Quantify effectiveness of offloading MPI functionality to programmable NICs and Portals hardware**
- **Accurately measure**
  - CPU availability
  - Bandwidth
- **Portable**

## Previous Work

---

- **Overhead**
  - **Netperf**
    - **Two processes per node**
    - **Assumes process driving communication relinquishes the CPU**
  - **Portability and accuracy issues**
- **Overlap**
  - **Pallas MPI benchmark's "exploit CPU method"**
  - **Various others, but no definitive metric for measuring overlap with respect to overall performance**

# COMB Approach

---

- **Two nodes**
- **One process per node**
  - **Node 0**
    - **Facilitate communication**
    - **I/O**
  - **Node 1**
    - **Simulate computation**
    - **Communication**
    - **Timing**
- **Use MPI for portability**

## COMB Approach (cont'd)

---

- Time a specified amount of work with no communication
- Time same amount of work with communication
- Ratio is CPU availability (1 – overhead)

# COMB Methods

---

- **Poll**
  - Measures sustained maximum bandwidth
  - Perform communication throughout work
  - Allow for maximum possible overlap
- **Post-Work-Wait (PWW)**
  - Time all MPI calls and do work
  - Tests for overlap under practical restrictions on MPI calls

# Poll Method

---

```
pre-post asynchronous receive(s)
read current time
for ( i = 0; i < work/poll factor; i++ ) {
    for ( j = 0; j < poll factor; j++ ) {
        /* nothing */
    }
    if (asynchronous receive is complete ) {
        start asynchronous reply(s)
        pre-post asynchronous receive(s)
    }
}
read current time
```

## Poll Method (cont'd)

---

- All receives are posted before sends
- Ping-pong messages flow in both directions
- Each process polls for message arrival
- Replacement messages are propagated as soon as previous batch completes
- Bandwidth and CPU availability are computed after a pre-determined amount of “computation”
- Polling interval can be adjusted to demonstrate tradeoff between bandwidth and availability
- Never blocks waiting for message arrival
- Accurate measure of availability

# PWW Method

---

```
read current time
pre-post asynchronous send(s) & receive(s)
read current time
for ( i = 0; i < work interval; i++ ) {
    /* nothing */
}
read current time
wait for asynchronous send(s) & receive(s)
read current time
```

## PWW Method (cont'd)

---

- Time work independent of messaging
- Collects wall clock times for different phases
  - Non-blocking post phase
  - Work phase
  - Wait phase
- Worker process waits for current batch of messages

# Hardware

---

- **Compaq DS10L**
- **617 MHz Alpha EV67 processor**
- **256 MB RAM**
- **Myrinet LANai-9, 64-bit 33 MHz**
- **Mesh64 switch**

## Platform - Software

---

- RedHat 6.2 Alpha
- Linux 2.2.18
- Portals 3.1 data movement interface
- MPICH 1.2.0 over Portals 3.1 direct device

# What is Portals?

---

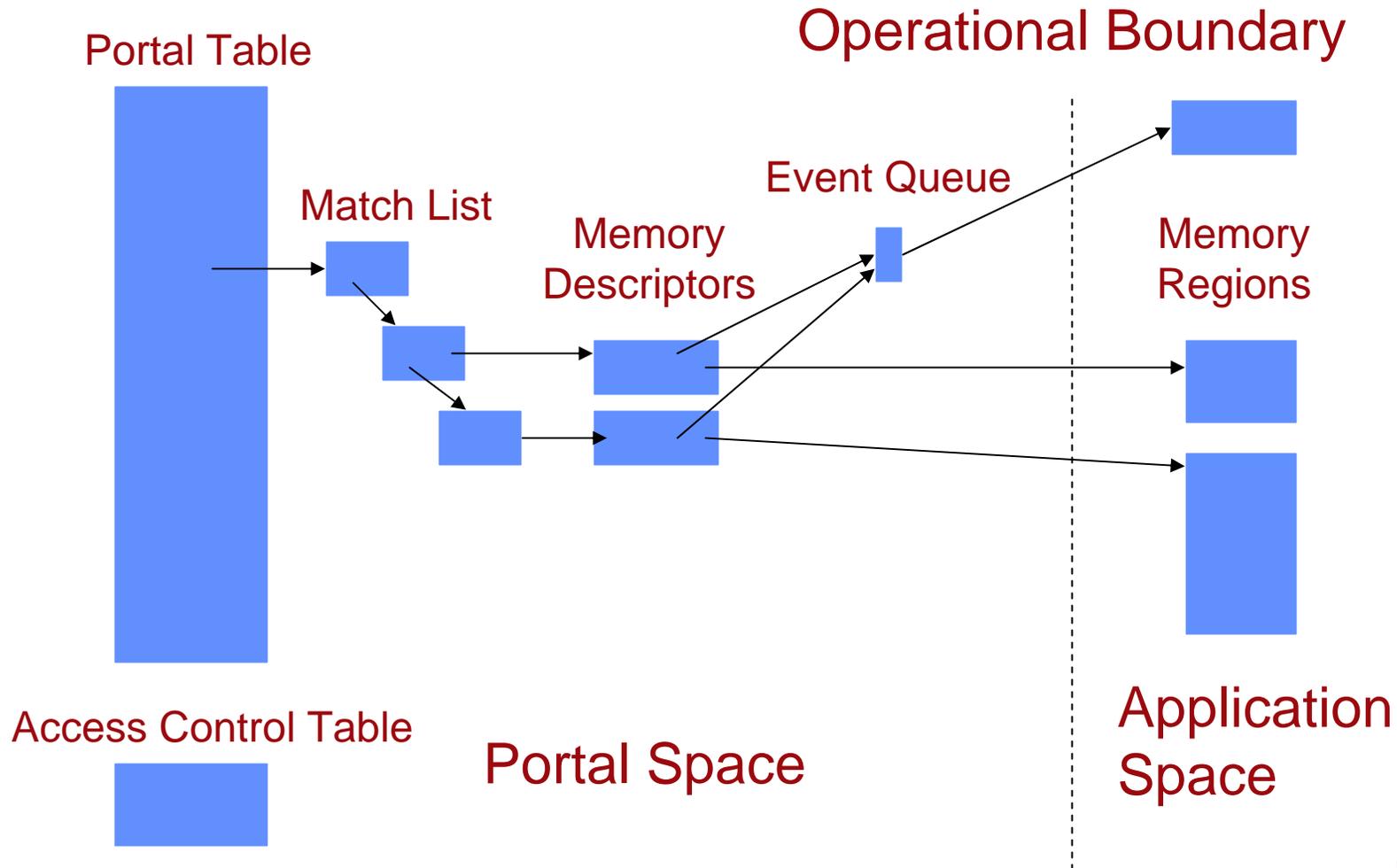
- **A data movement layer**
  - Data movement is fundamental to more than just parallel applications
  - Runtime systems, I/O systems, parallel debuggers
- **A programming interface**
  - User-level or kernel-level
- **Not a programming model**
- **Not a wire protocol**

# Portals 3.x Features

---

- **Best effort, in-order delivery**
- **Well-defined transport failure semantics**
- **Expected messages**
- **One-sided operations**
  - **Put and Get**
- **Zero-copy message passing**
  - **Increased bandwidth**
- **OS-bypass implementation**
  - **Reduced latency**
- **Application-bypass semantic**
  - **No polling, no threads**
  - **No host CPU utilization**
  - **Reduced software complexity**

# Portal Addressing



# What Makes Portals Different?

---

- Provides elementary building blocks for supporting higher-level protocols well
- Allows structures to be placed in user-space, kernel-space, or NIC-space
- Allows for OS-bypass implementations
- Receiver-managed offset allows for efficient and scalable buffering of unexpected messages
- Supports multiple protocols within a process
- Runtime system independent
- Well-defined failure semantics
- Supports application-bypass semantic

# Supported Higher-Level Systems

---

- High-level message passing libraries
  - MPICH
  - MPI/Pro, ChaMPIon/Pro
  - RPC
  - InterComm
  - Intel NX
  - nCUBE Vertex
  - Initial MPI-2 one-sided specification from March 1996
  - Cray SHMEM variant
- Cplanted™ Runtime system
  - Distributed server library
  - Dynamic process creation library
- File systems
  - ASCI/Red fyod
  - Cplanted™ ENFS
  - Lustre
- In progress
  - MPI-2 one-sided (MSTI)
  - TotalView channel implementation (Sandia)

# Portals Over RMPP

---

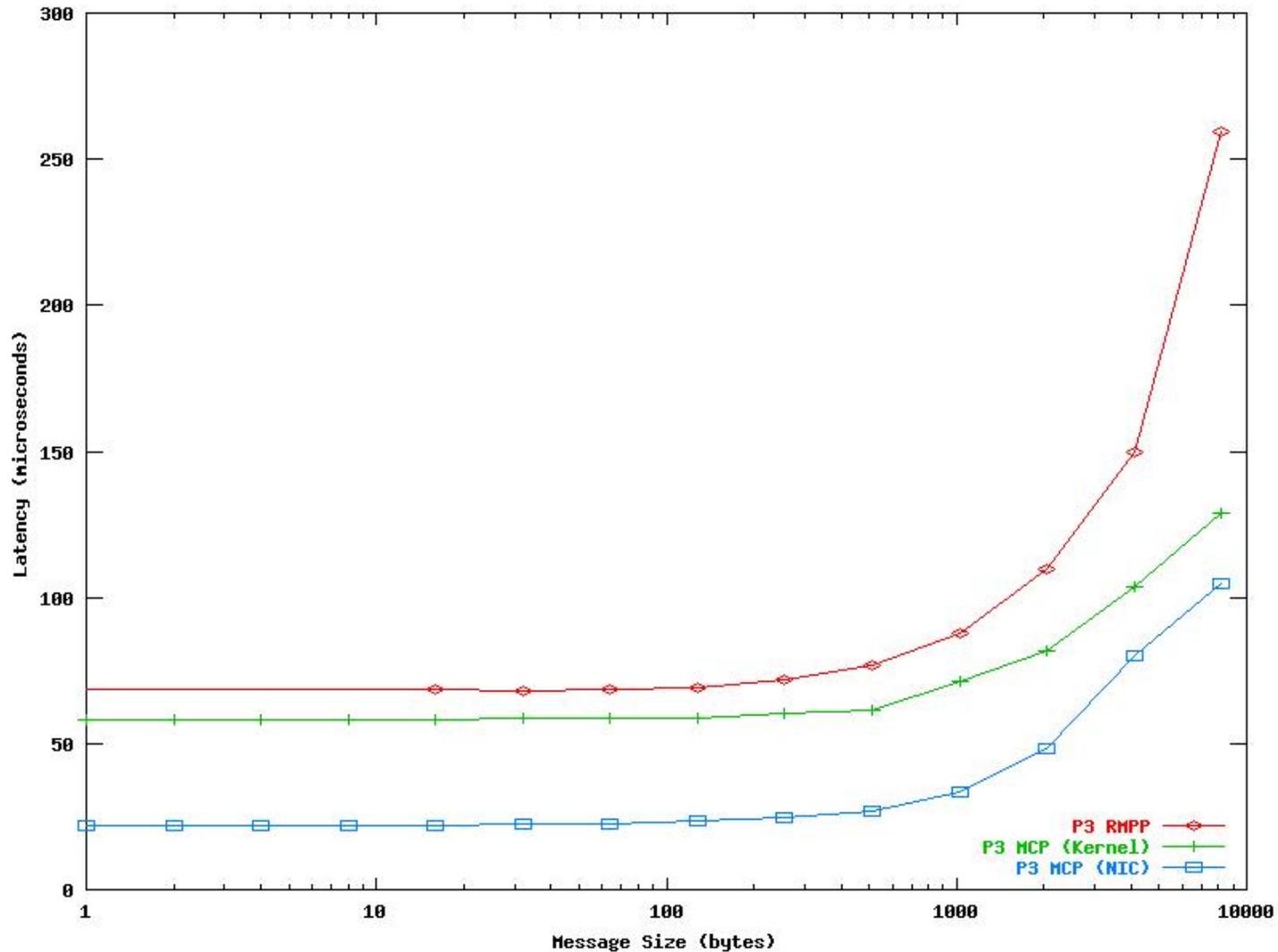
- **Reliable Message Passing Protocol (RMPP)**
  - Linux kernel module that works with any Linux network device
  - Provides flow control and reliability
  - Interrupt driven
  - Copies from NIC to kernel to user
- **Works with a Portals kernel module**
  - All Portals processing occurs in the kernel
  - No OS bypass

# Portals Myrinet Control Program

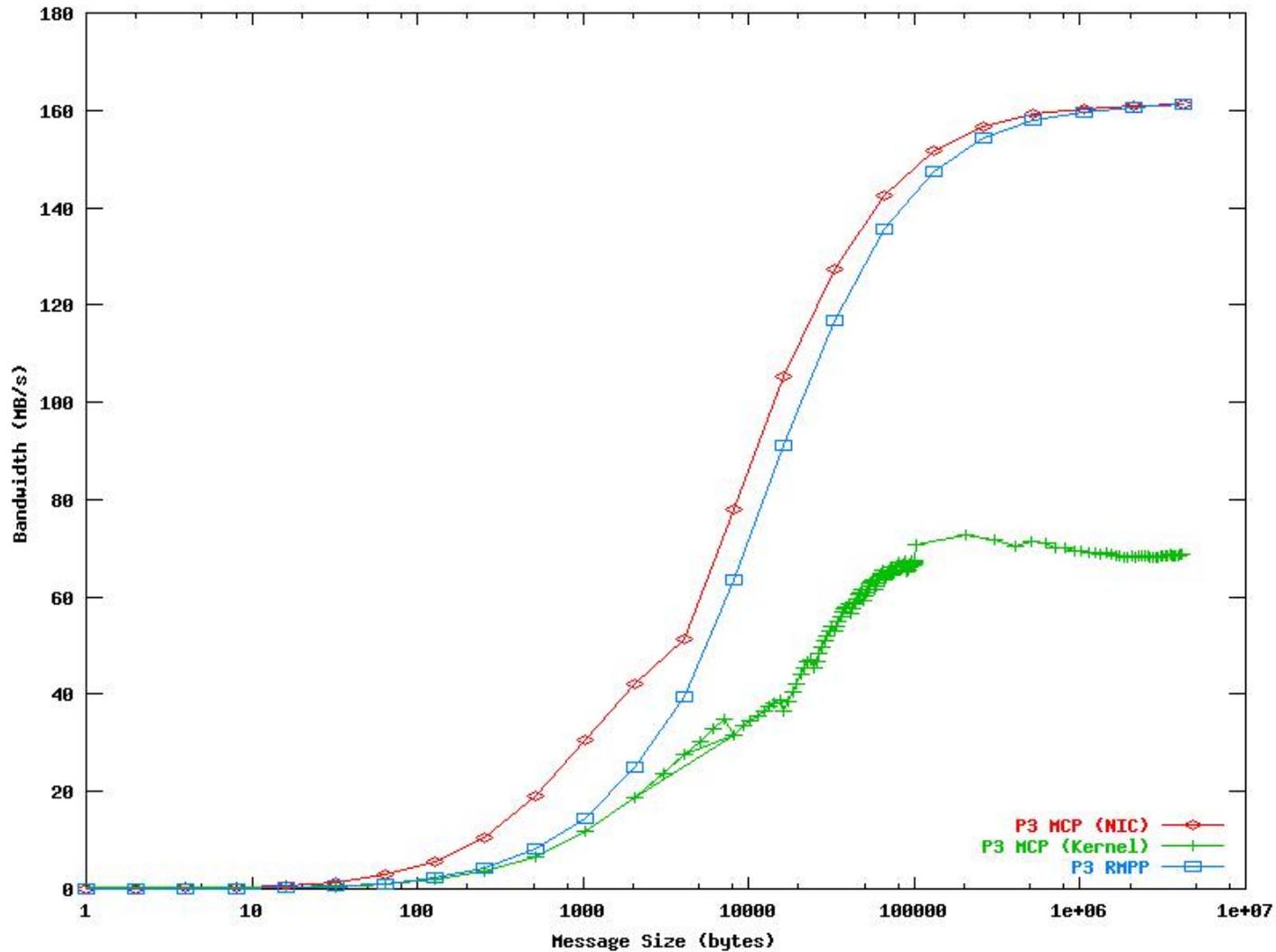
---

- **Runs on the Myrinet NIC processor**
- **Portals processing can occur on the NIC or in the kernel**
- **OS bypass**
- **Limitations**
  - **Reliability and flow control protocol is not very scalable**
  - **Limited to using physically contiguous 4 MB memory regions**

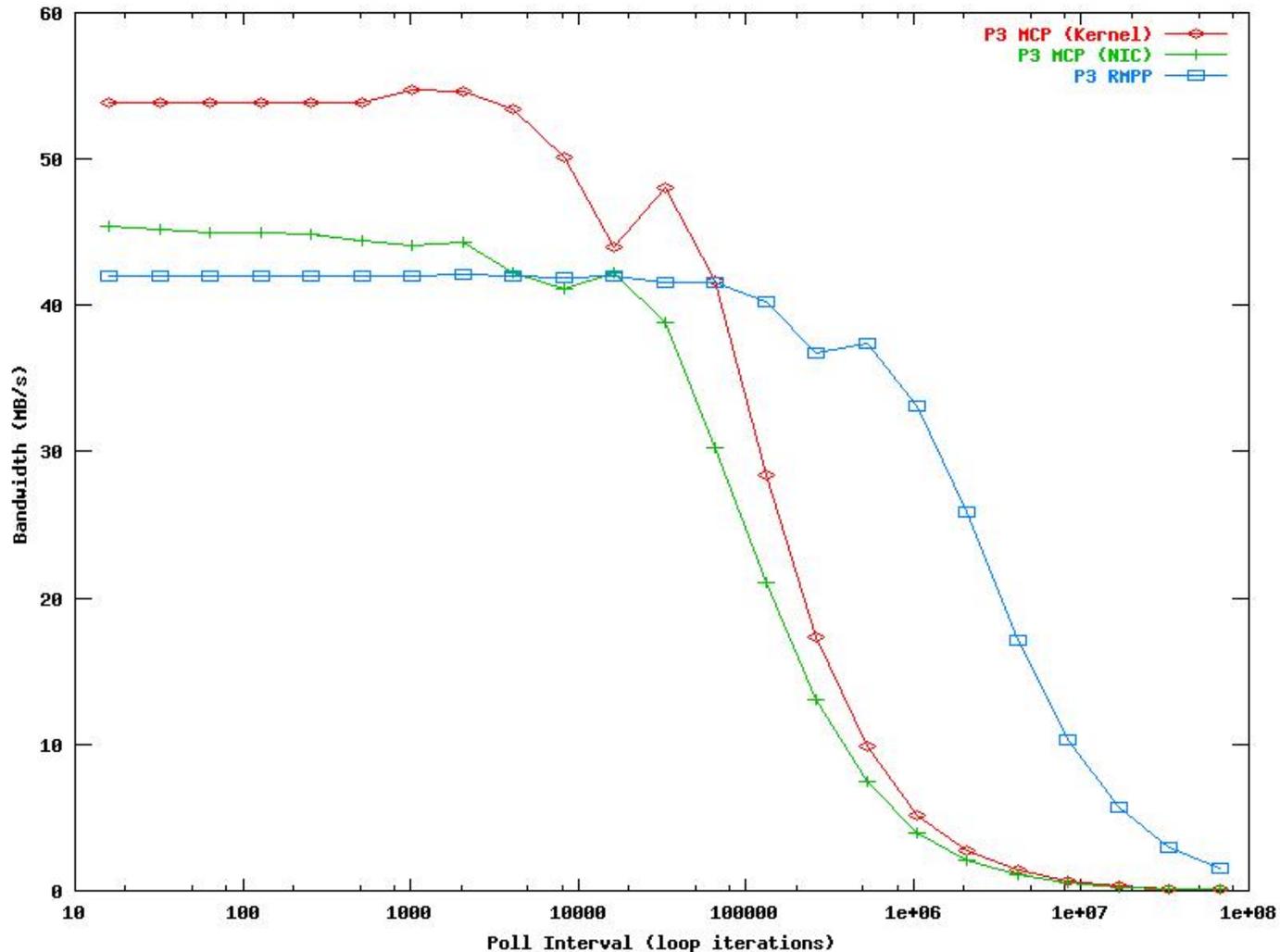
# Ping-pong Latency



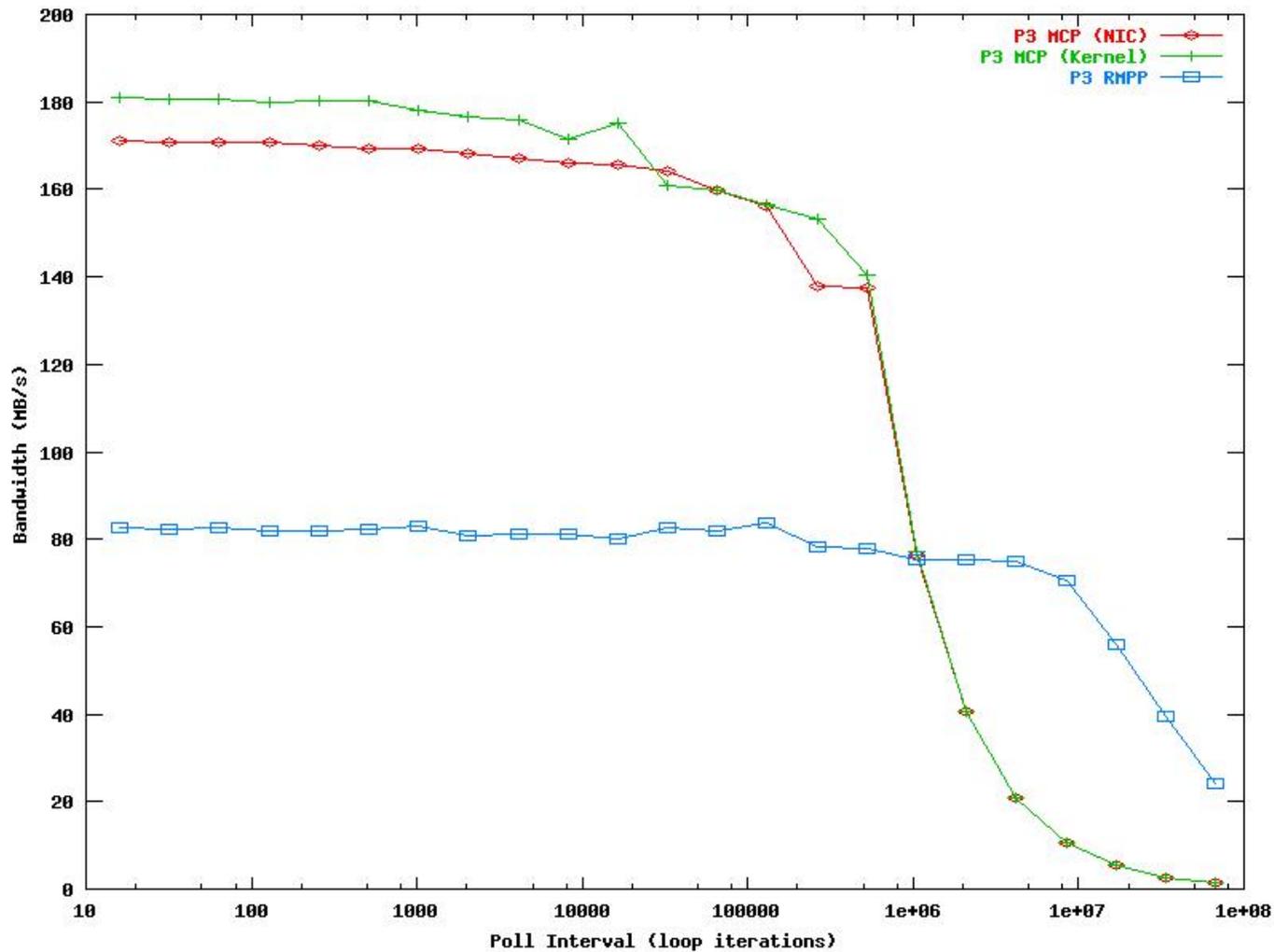
# Ping-pong Bandwidth



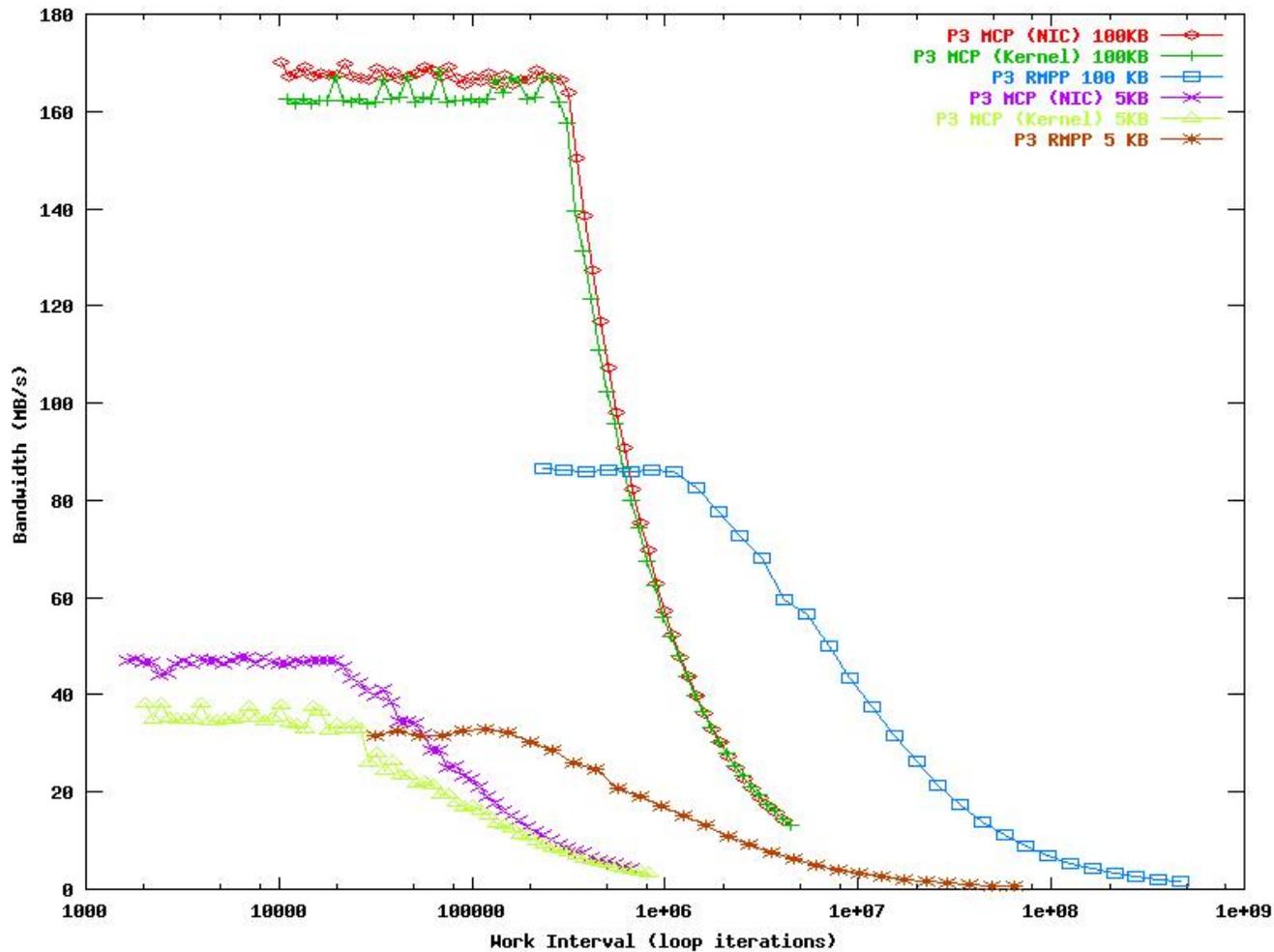
# Poll Method: Bandwidth (5KB)



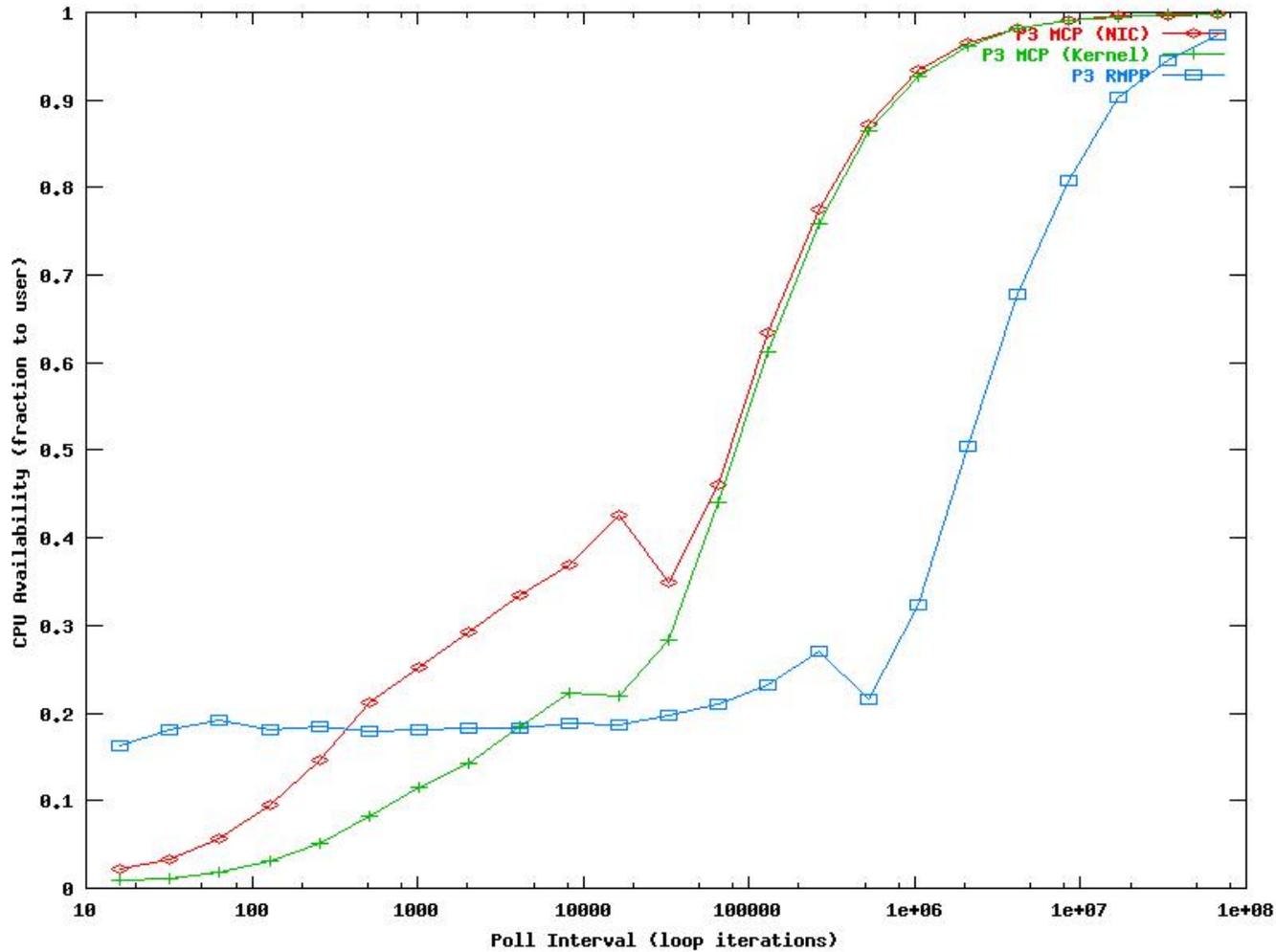
# Poll Method: Bandwidth (100KB)



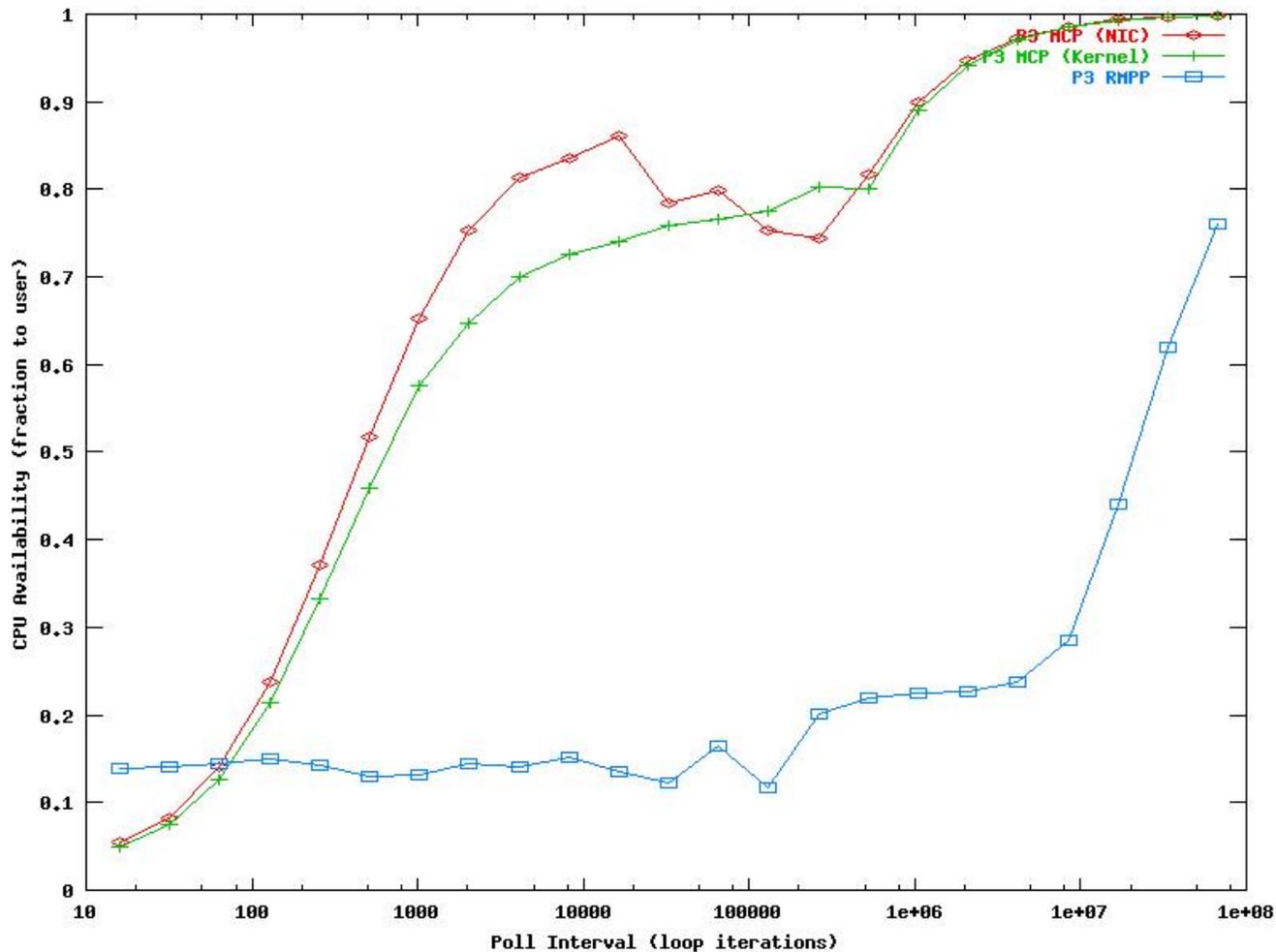
# PWW Method: Bandwidth (5 KB & 100 KB)



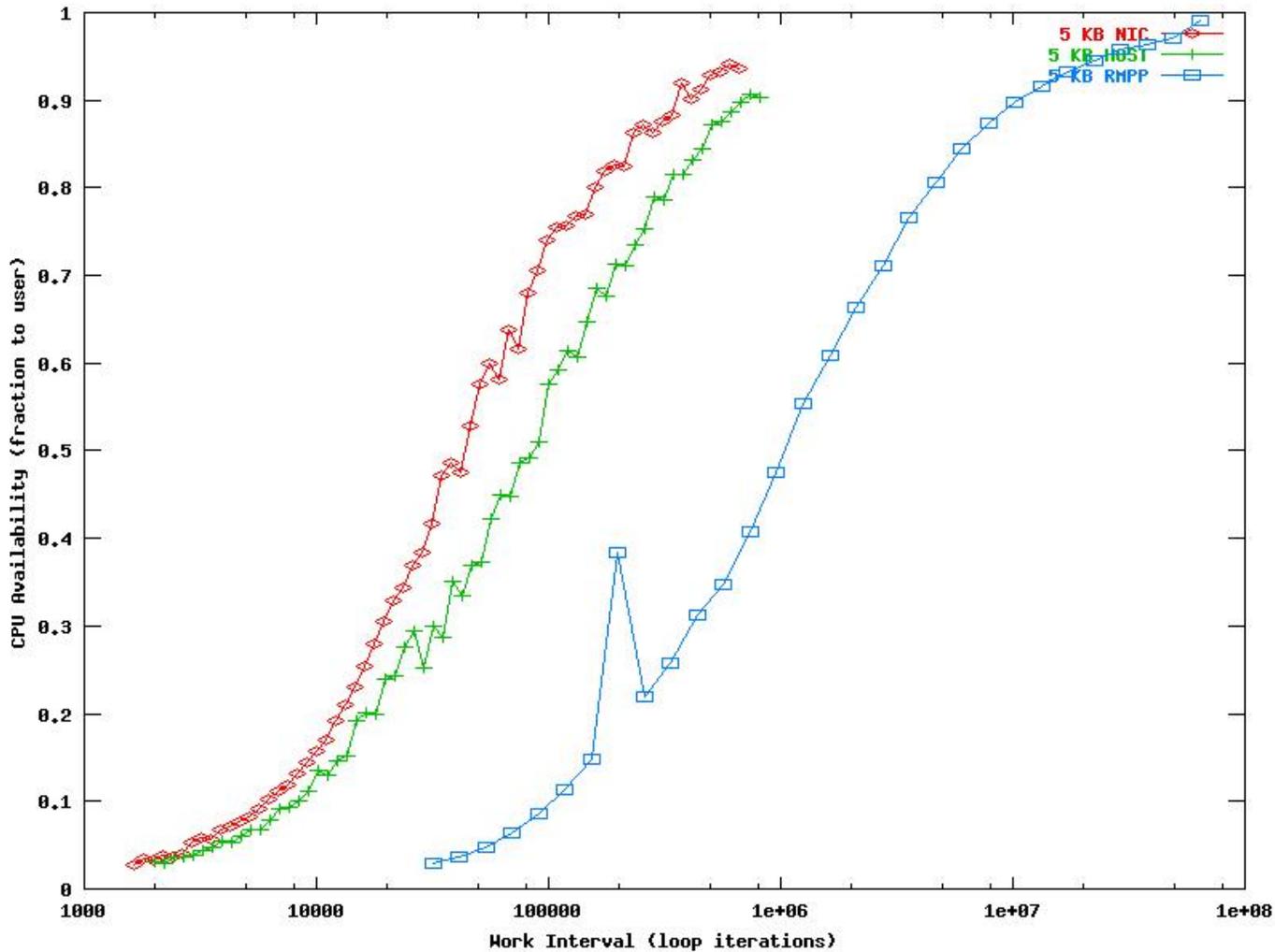
# Poll Method: CPU Availability (5 KB)



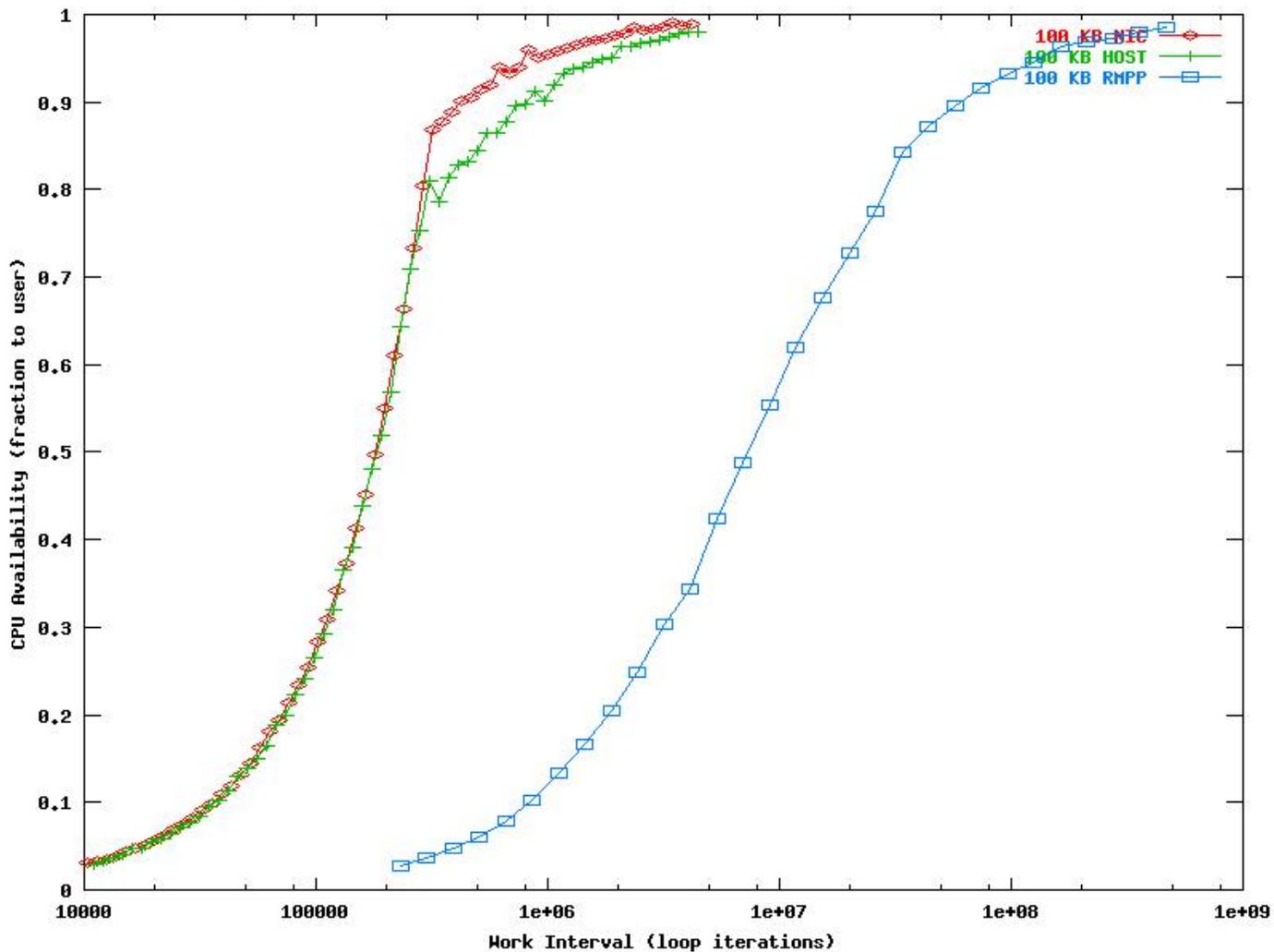
# Poll Method: CPU Availability (100 KB)



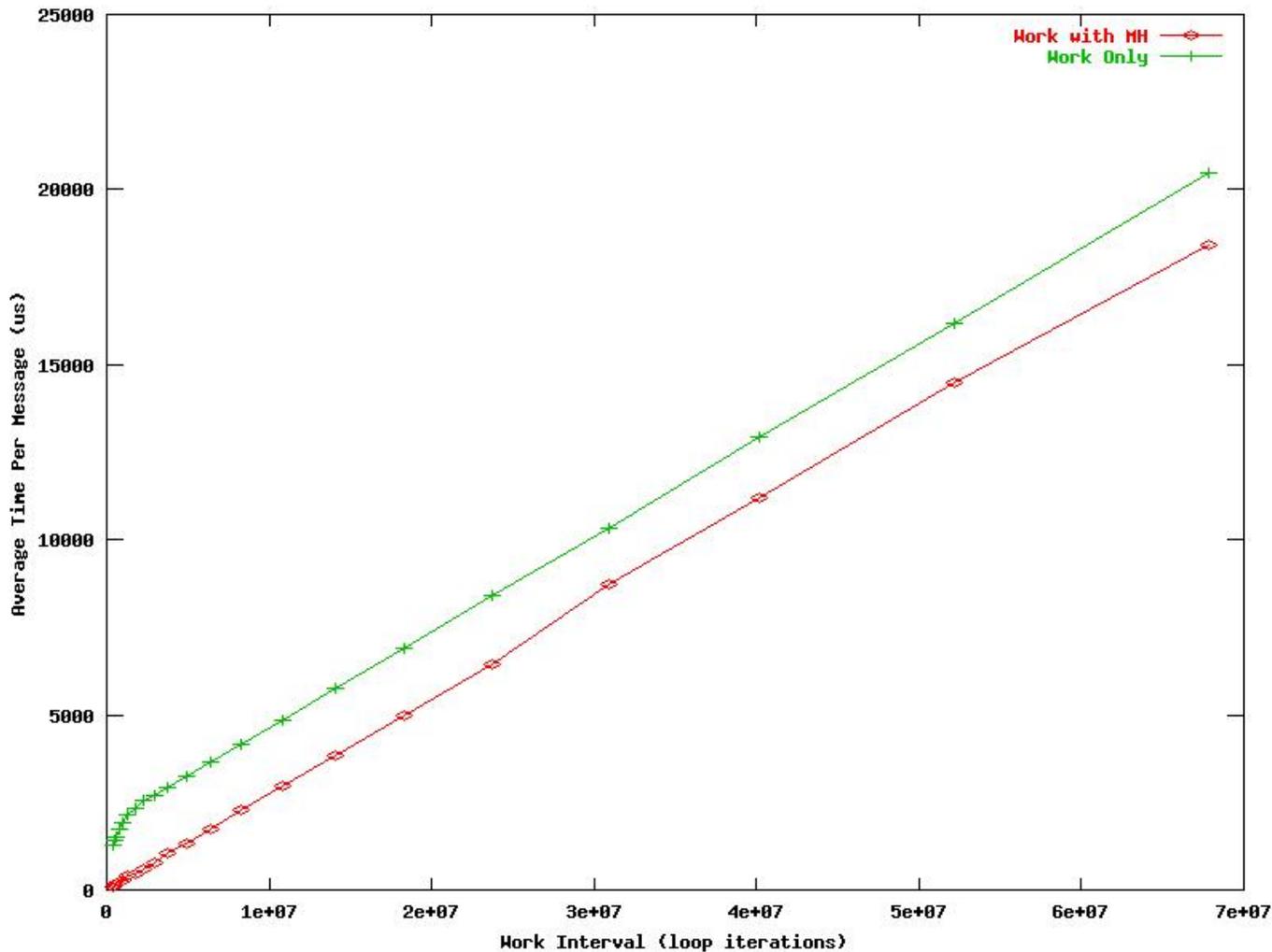
# PWW Method: CPU Availability (5 KB)



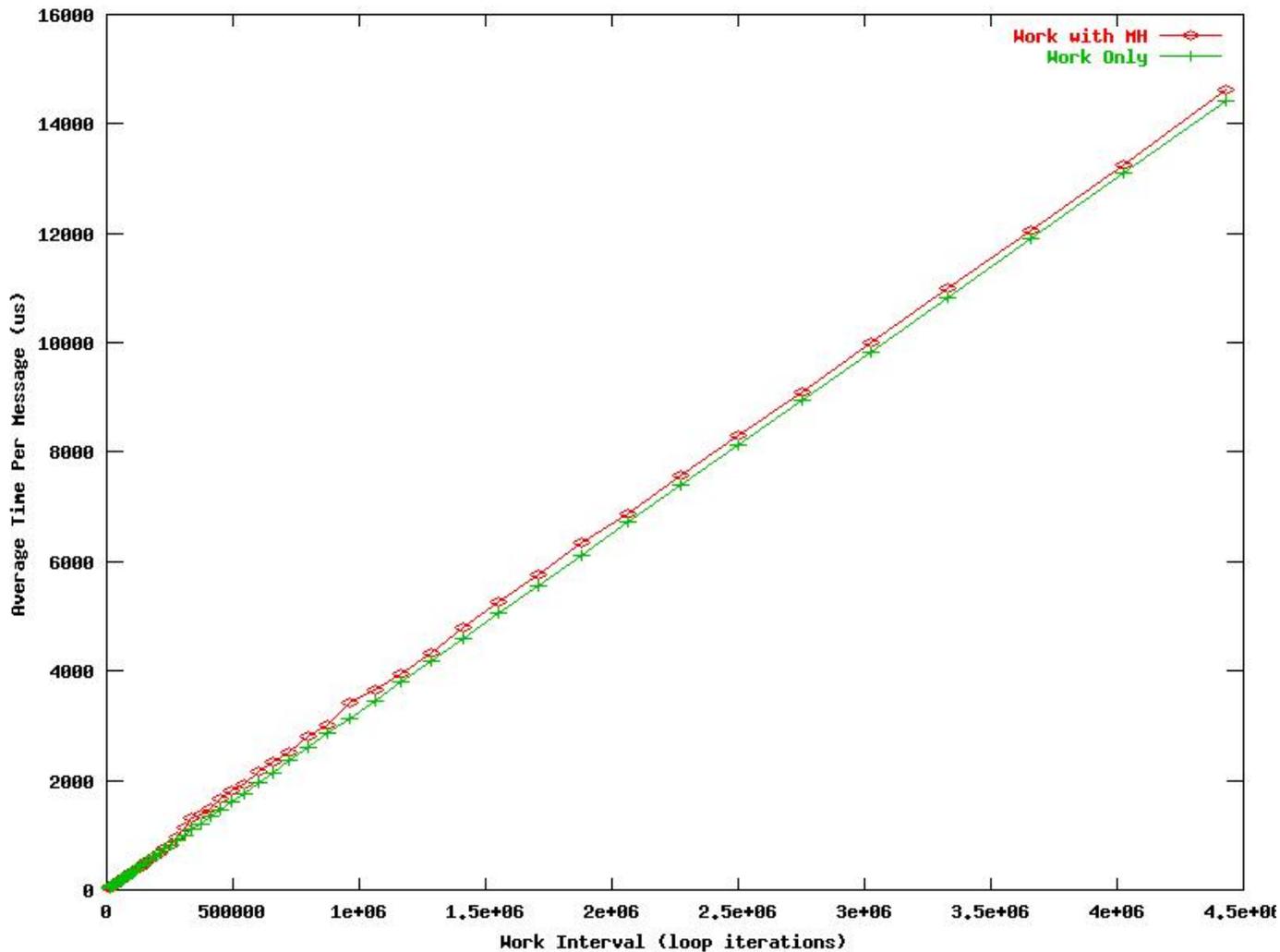
# PWW Method: CPU Availability (100 KB)



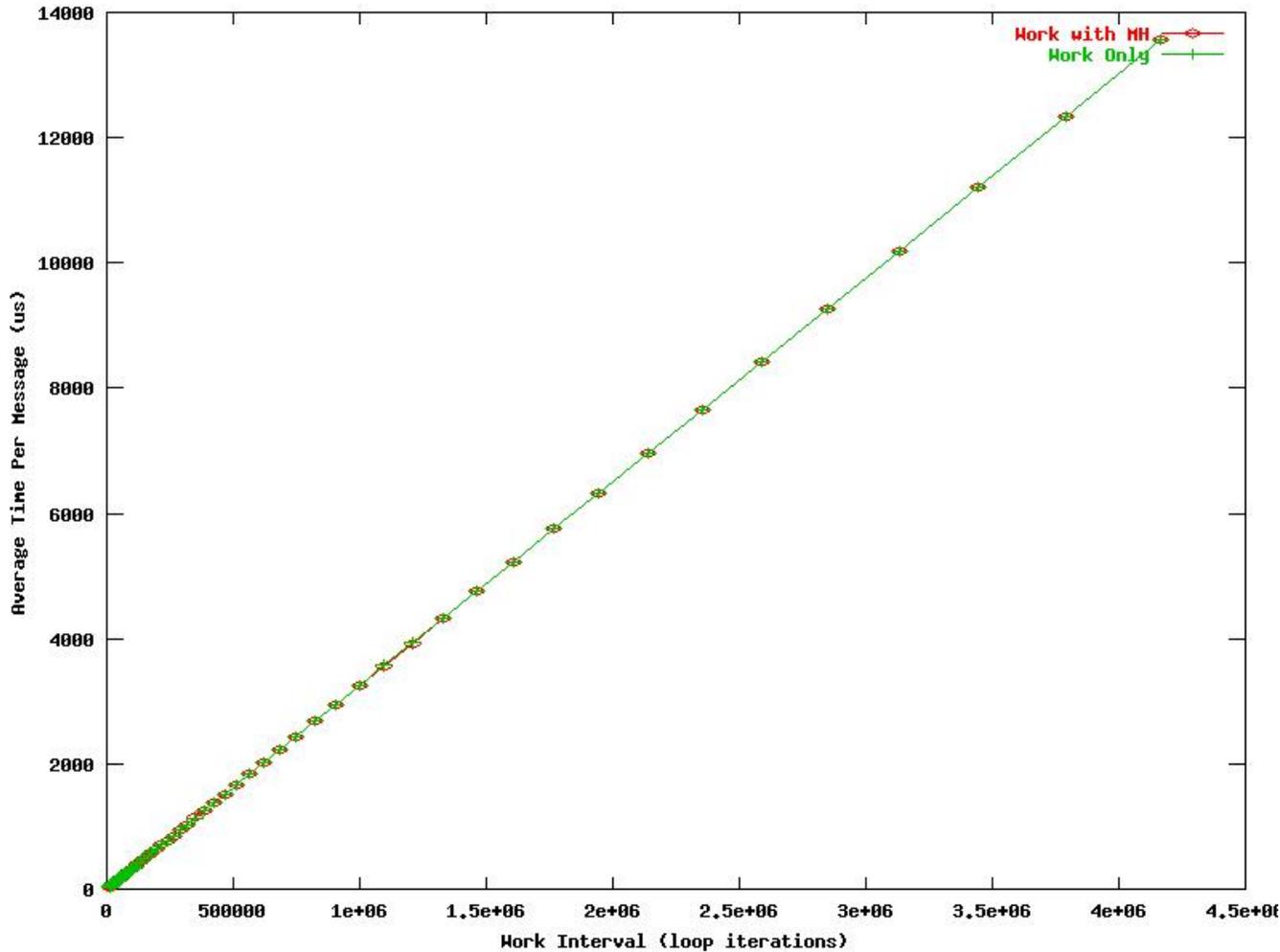
# PWW Method: CPU Overhead (RMPP)



# PWW Method: CPU Overhead (MCP Kernel)



# PWW Method: CPU Overhead (MCP NIC)



# Summary

---

- **COMB measures the ability of an MPI implementation to overlap computation and communication**
- **COMB provides more insight into the relationship between network performance and host CPU performance**
- **Helps to quantify the benefit of “application offload”**

# Acknowledgments

---

- **Arthur B. Maccabe, Bill Lawry**
  - COMB design and implementation
- **Rolf Riesen**
  - RMPP module
- **Mike Levenhagen**
  - Portals MCP
- **Ron Brightwell, Arthur B. Maccabe, Rolf Riesen**
  - Portals API

# Application-Bypass Results

