

# Designing Non-blocking Broadcast with Collective Offload on InfiniBand Clusters : A Case Study with HPL

**Krishna Kandalla** <sup>(1)</sup>, Hari Subramoni <sup>(1)</sup>,  
Jerome Vienne<sup>(1)</sup>, Siddhesh Raikar<sup>(1)</sup> ,  
Karen Tomko<sup>(2)</sup>, Sayantan Sur<sup>(1)</sup>,  
and Dhabaleswar. K. Panda<sup>(1)</sup>

<sup>(1)</sup>Computer Science & Engineering Department,  
The Ohio State University

<sup>(2)</sup> The Ohio Supercomputer Center

# Outline

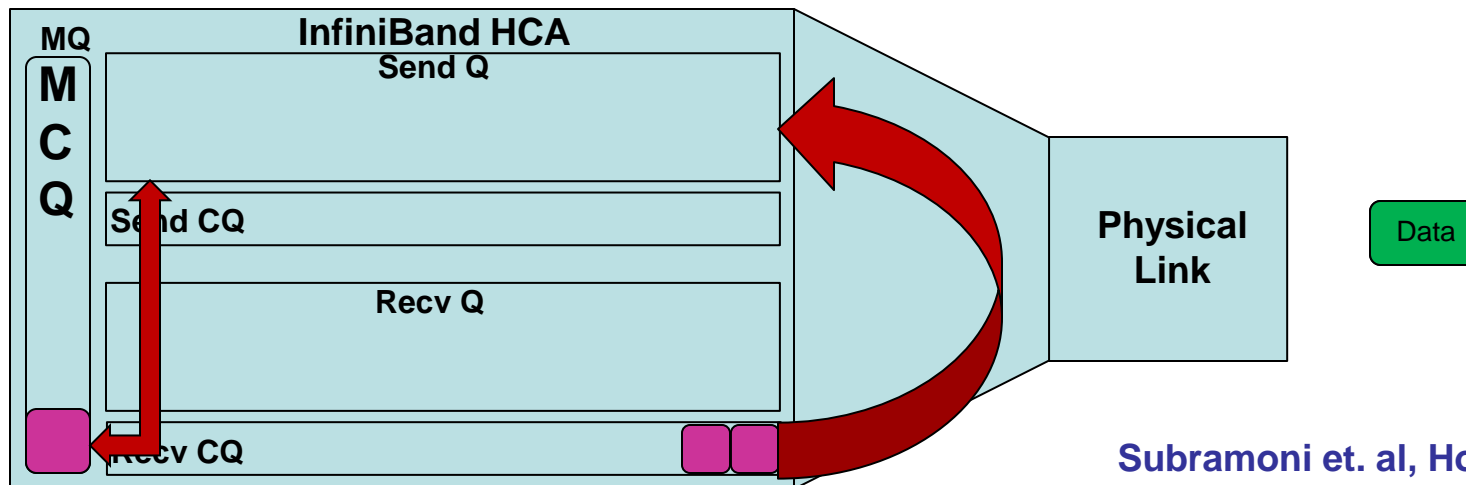
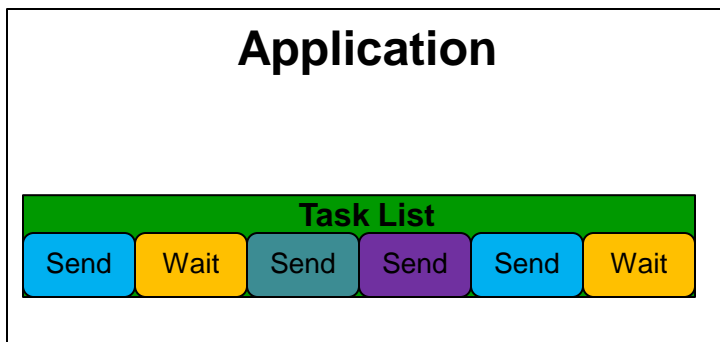
- Introduction
- Motivation and Problem Statement
- Designing MPI\_Ibcast with Collective Offload
- Experimental Evaluation
  - Micro-Benchmark evaluations
  - Experiments with HPL
  - Impact of System Noise
- Conclusions and Future work

# Introduction

- MPI-2.2 defines blocking collective operations – limits performance and scalability of applications
- Host-based blocking collectives are prone to noise
- MPI-3 will support non-blocking collectives
- ConnectX-2 adapters from Mellanox can be used to design non-blocking collectives

# Overview of InfiniBand Collective Offload

- Applications can offload task-lists to the NIC
- A CQE gets created on the MCQ after execution

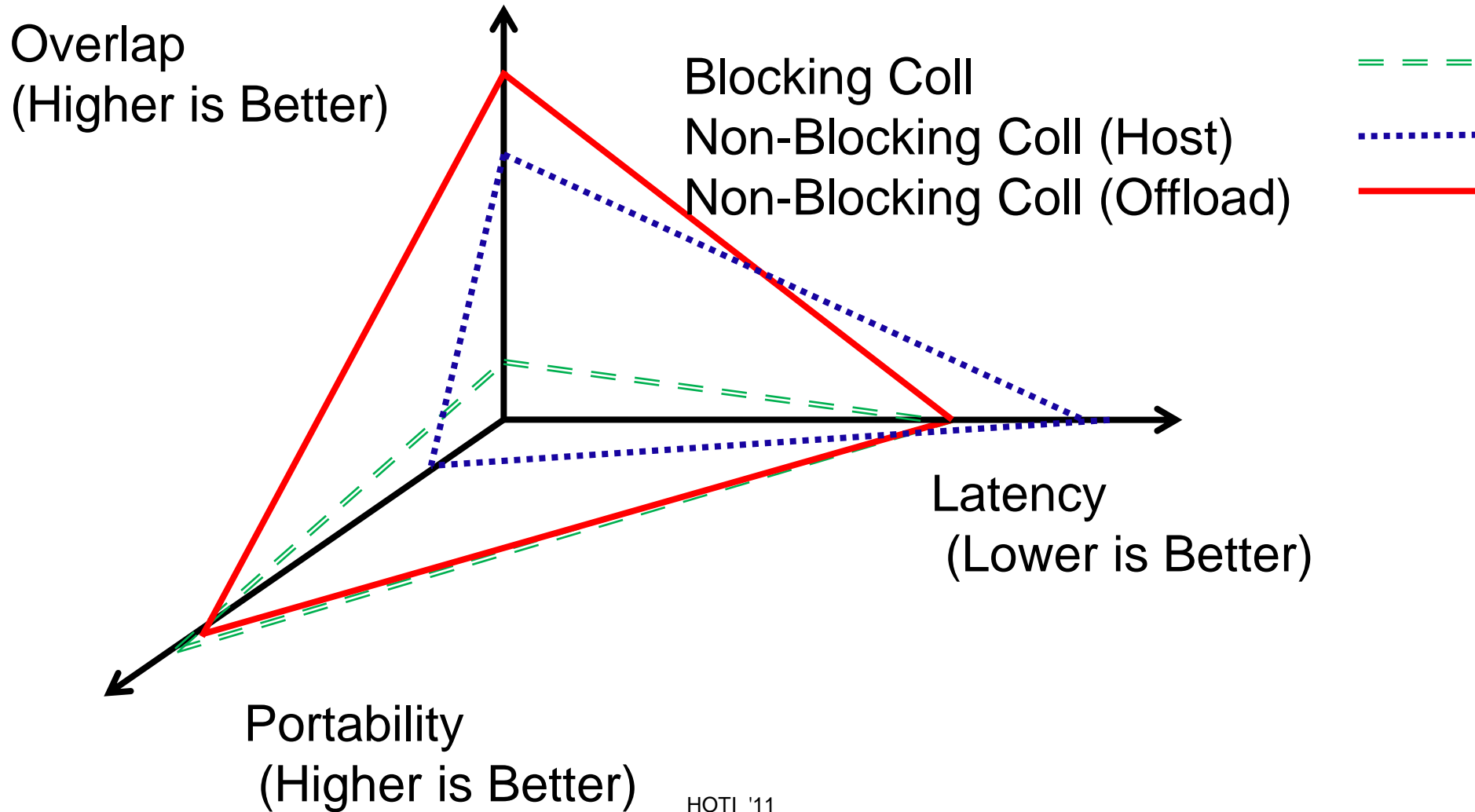


Subramoni et. al, Hot Interconnects 2010

# Outline

- Introduction
- Motivation and Problem Statement
- Designing MPI\_Ibcast with Collective Offload
- Experimental Evaluation
- Conclusions and Future work

# Design Space of Collective Algorithms



# Design Space of Collective Algorithms

Overlap  
(Higher is Better)

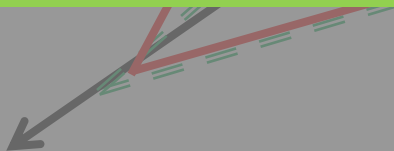


Blocking Coll

Non-Blocking Coll (Host)



- **Challenge: progress the collective schedules in an asynchronous manner *with*:**
  - **performance portability**
  - **minimal host processor intervention**
  - **acceptable communication latency**



Portability  
(Higher is Better)

# Related Work

- Hemmert et. al, studied offload capabilities with Portals
- Hoefler et. al, proposed host-based non-blocking solutions (libNBC)
- Graham et. al, reported early experiences with CORE-Direct (CAC '09, CCGrid '10)
- Kandalla et. al, proposed MPI\_lalltoall and studied benefits with P3DFFT (ISC '11)
- Venkata et. al, proposed network-offload based MPI\_Bcast and MPI\_lbroadcast with 64 processes (CAC '11)

# Problem Statement

- Can we design network-offload-based MPI\_Ibcast?
- Will network offload solutions offer better communication/computation overlap for collectives?
- Can network offload improve application throughput?
- Are network-based designs more resilient to noise?
- Can we leverage our proposed MPI\_Ibcast to improve the efficiency of popular benchmarks like HPL?

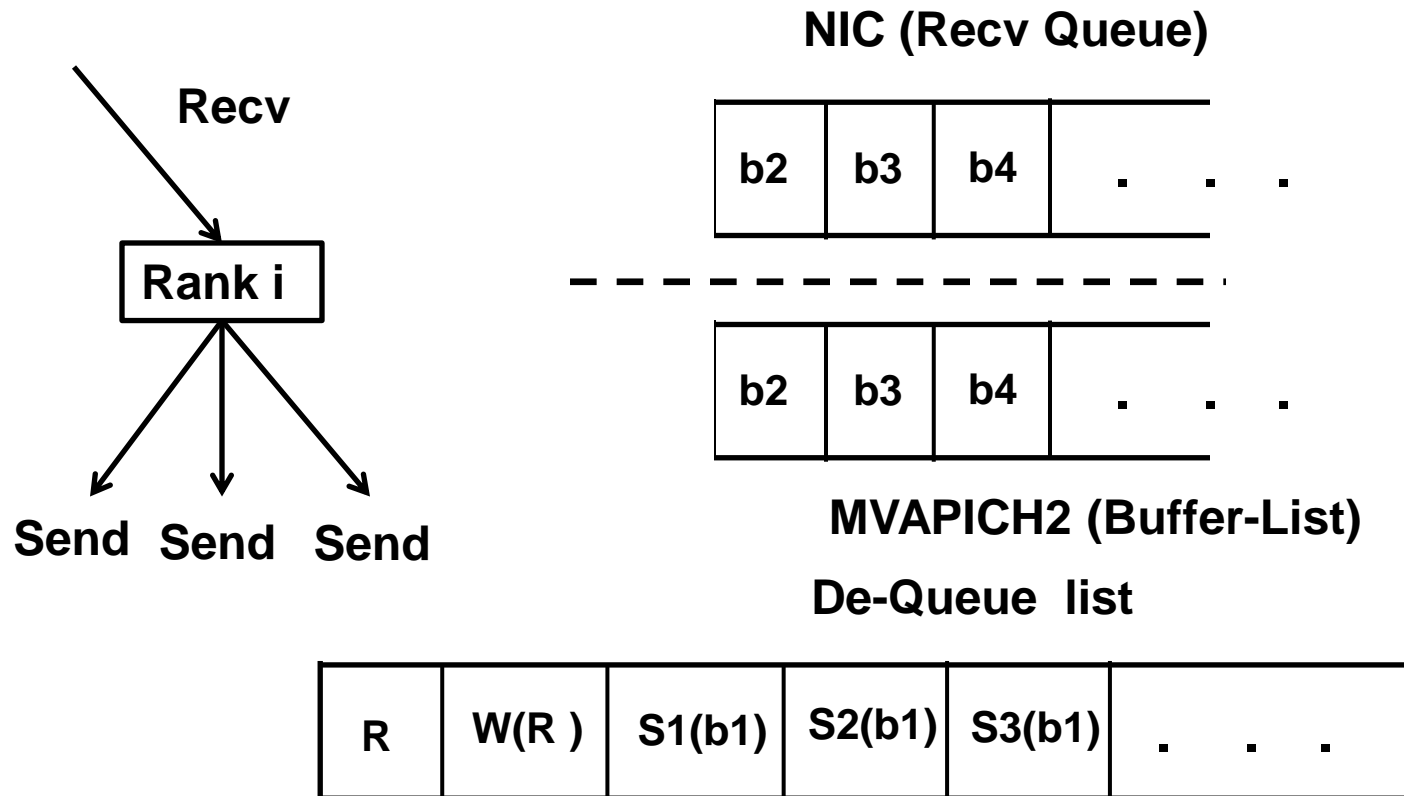
## MVAPICH/MVAPICH2 Software

- High Performance MPI Library for IB, 10GigE/iWARP and RoCE
  - MVAPICH (MPI-1) and MVAPICH2 (MPI-2.2)
  - Used by more than 1,650 organizations in 63 countries
  - More than 76,000 downloads from OSU site directly
  - Empowering many TOP500 clusters
    - 7<sup>th</sup> ranked 111,104-core cluster (Pleiades) at NASA
    - 17<sup>th</sup> ranked 62,976-core cluster (Ranger) at TACC
  - Available with software stacks of many IB, HSE and server vendors including Open Fabrics Enterprise Distribution (OFED) and Linux Distros (RedHat and SuSE)
  - <http://mvapich.cse.ohio-state.edu>

# Outline

- Introduction
- Motivation and Problem Statement
- Designing MPI\_Ibcast with Collective Offload
- Experimental Evaluation
- Conclusions and Future work

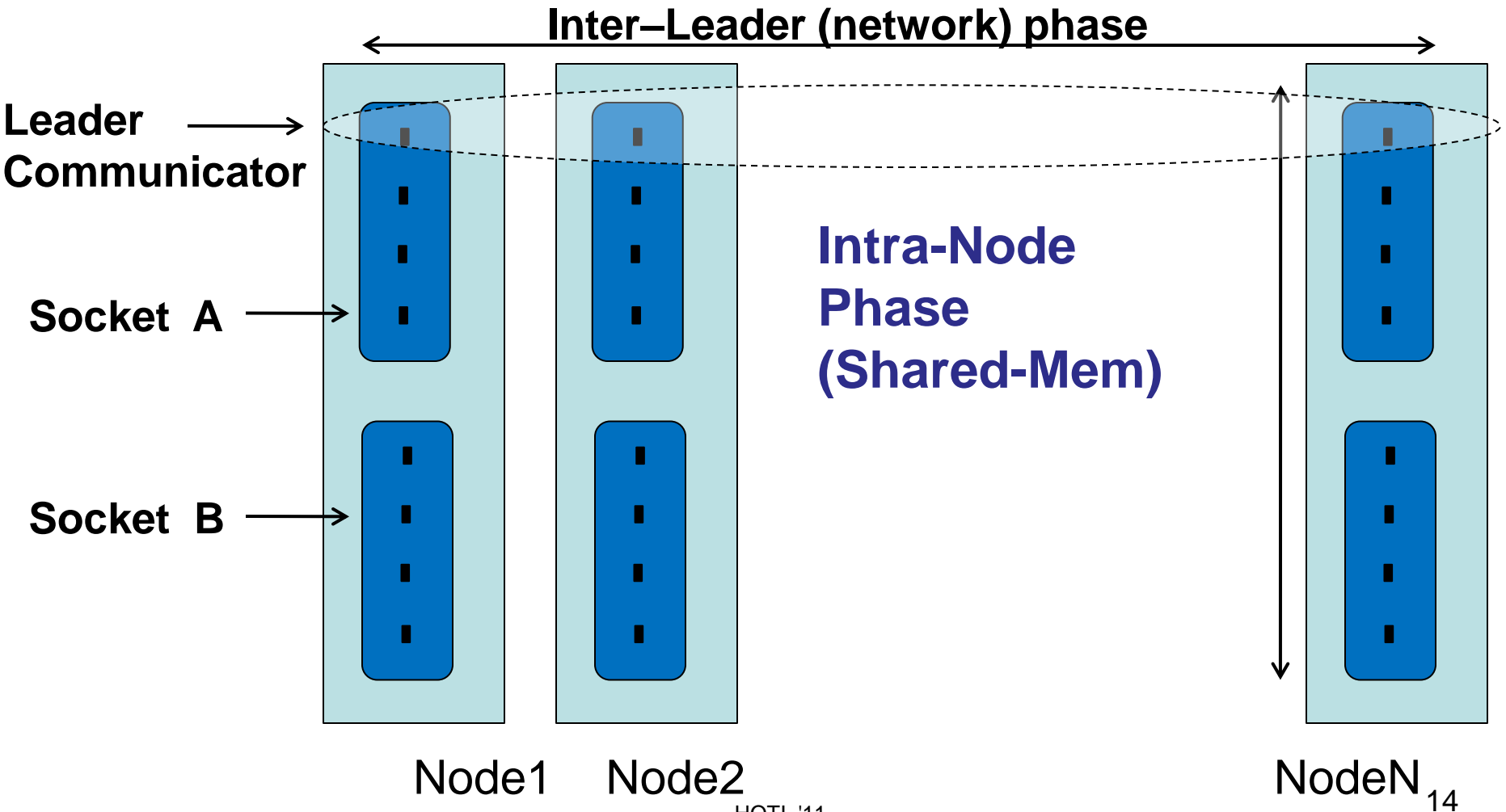
# MPI\_Ibcast (small messages)



# MPI\_Ibcast (large messages)

- Use separate large message QP
- Dynamically register user-buffers
- Use InfiniBand's RNR for flow-control
- Network offload-based scatter-allgather algorithms
  - Scatter : Binomial Tree  
(Special case of K-nomial with  $K=2$ )
  - Allgather : Recursive-Doubling or Ring
- Use send/recv/wait tasks to implement the above algorithms. Flow control guaranteed by the network

# Multi-Core Aware Shared-Memory Based Collective Algorithms in MVAPICH2



# Design Choices for MPI\_Ibcast

- Flat-Offload (FO) : Offload the entire broadcast operation
- Two-Level-Host (TOH):

Inter-leader through offload channel

Intra-node through shared-memory (during MPI\_Wait)

(Better for latency sensitive scenarios)

- Two-Level-Offload (TOO):

Inter-leader through offload channel

Intra-node through offload loopback

(Can offer better overlap, if latency is not too critical)

# Outline

- Introduction
- Motivation and Problem Statement
- Designing MPI\_Ibcast with Collective Offload
- Experimental Evaluation
  - Micro-Benchmark evaluations
  - Experiments with HPL
  - Impact of System Noise
- Conclusions and Future work

# Experimental Setup

- Intel Xeon E5640 (2.53 GHz), 12 GB memory per node
- MT26428 QDR ConnectX-2 with PCI-Ex interfaces, 171-port Mellanox QDR switch, OFED 1.5.1
- RHEL 5.4, 2.6.18-164.e15 kernel version
- MVAPICH2 (v1.6)

# Micro-Benchmark Evaluations

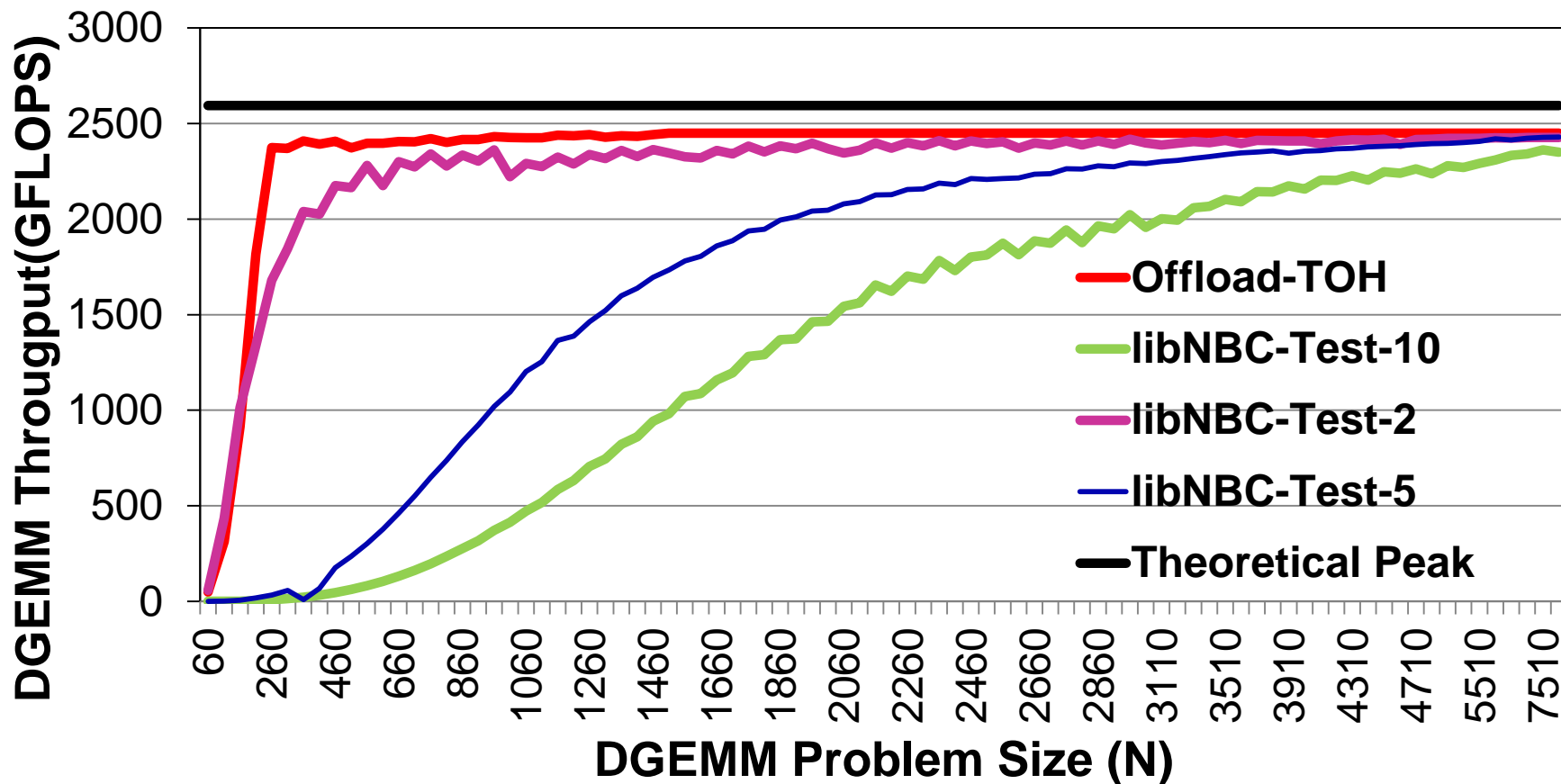
## Throughput Benchmark

```
start_throughput_timer()  
  MPI_Ibcast(..)  
  CBLAS_DGEMM()  
  MPI_Wait(..)  
end_throughput_timer()
```

## Impact of Noise

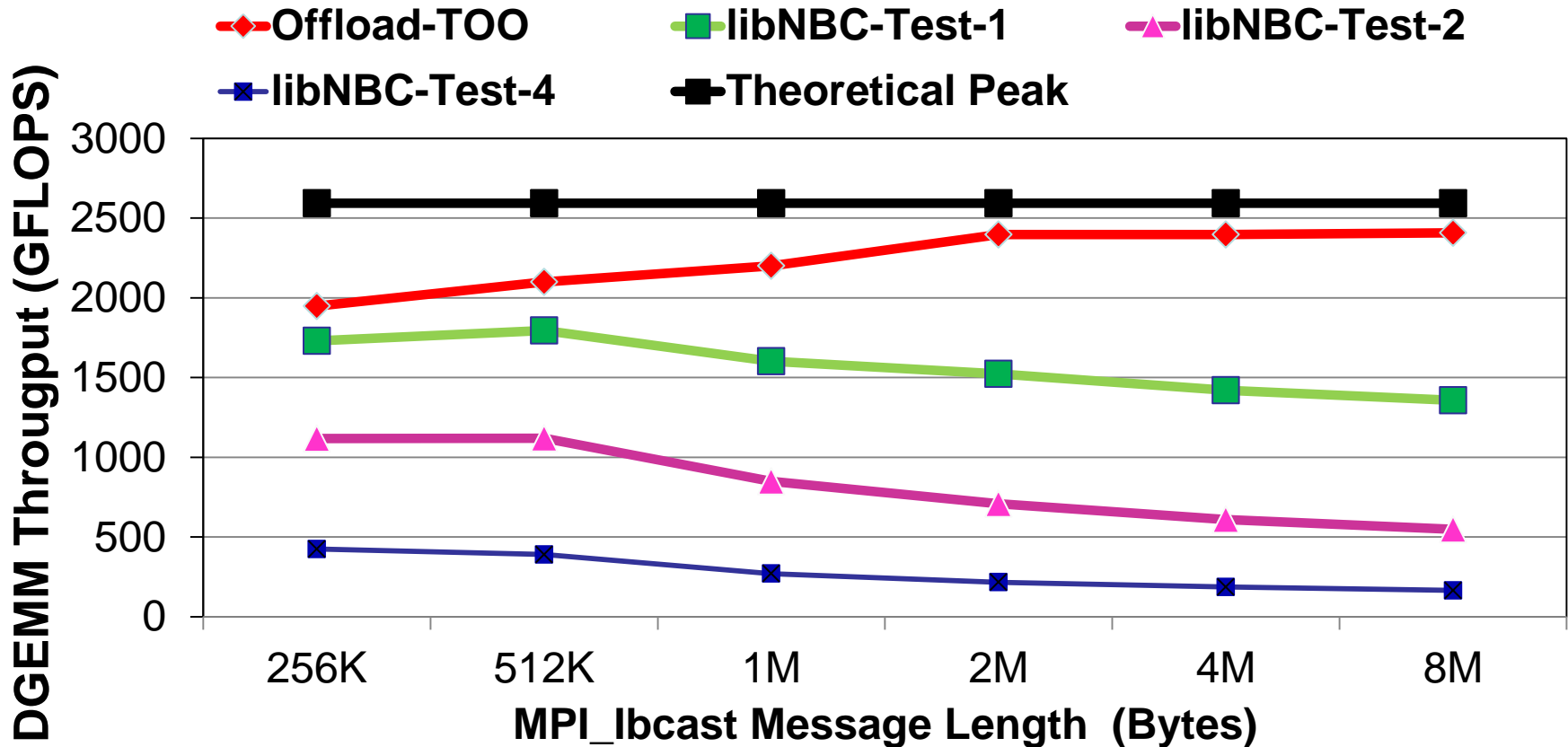
- Daemon process on each core --  
mat\_mul
- Noise Freq ( 20Hz to 1KHz )
- Noise Duration ( 50 to 250 usec )
- Measure throughput of  
CBLAS\_DGEMM
  - Offload MPI\_Ibcast
  - Host-based MPI\_Ibcast

# Communication/Computation Overlap



**CBLAS-DGEMM overlapped with Offload-Ibcast delivers better throughput when compared to Host-Based Ibcast with 256 processes**

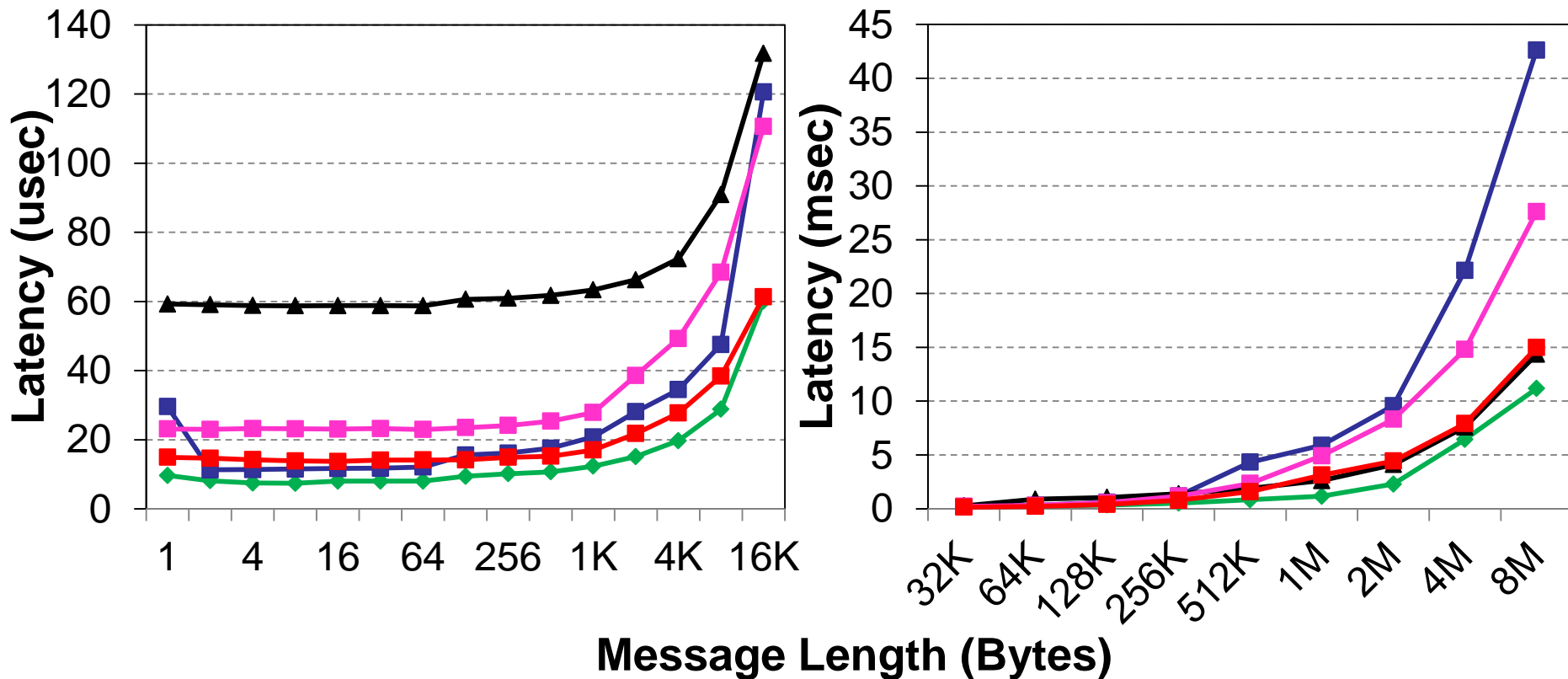
# Communication/Computation Overlap



**Bcast-Offload delivers near perfect communication/computation overlap for all messages in a portable manner with 256 Processes**

# Latency Comparison

◆ MV2-Bcast-Default     ■ MV2-Bcast-Loop-back     ▲ libNBC  
■ Offload-TOO     ■ Offload-TOH



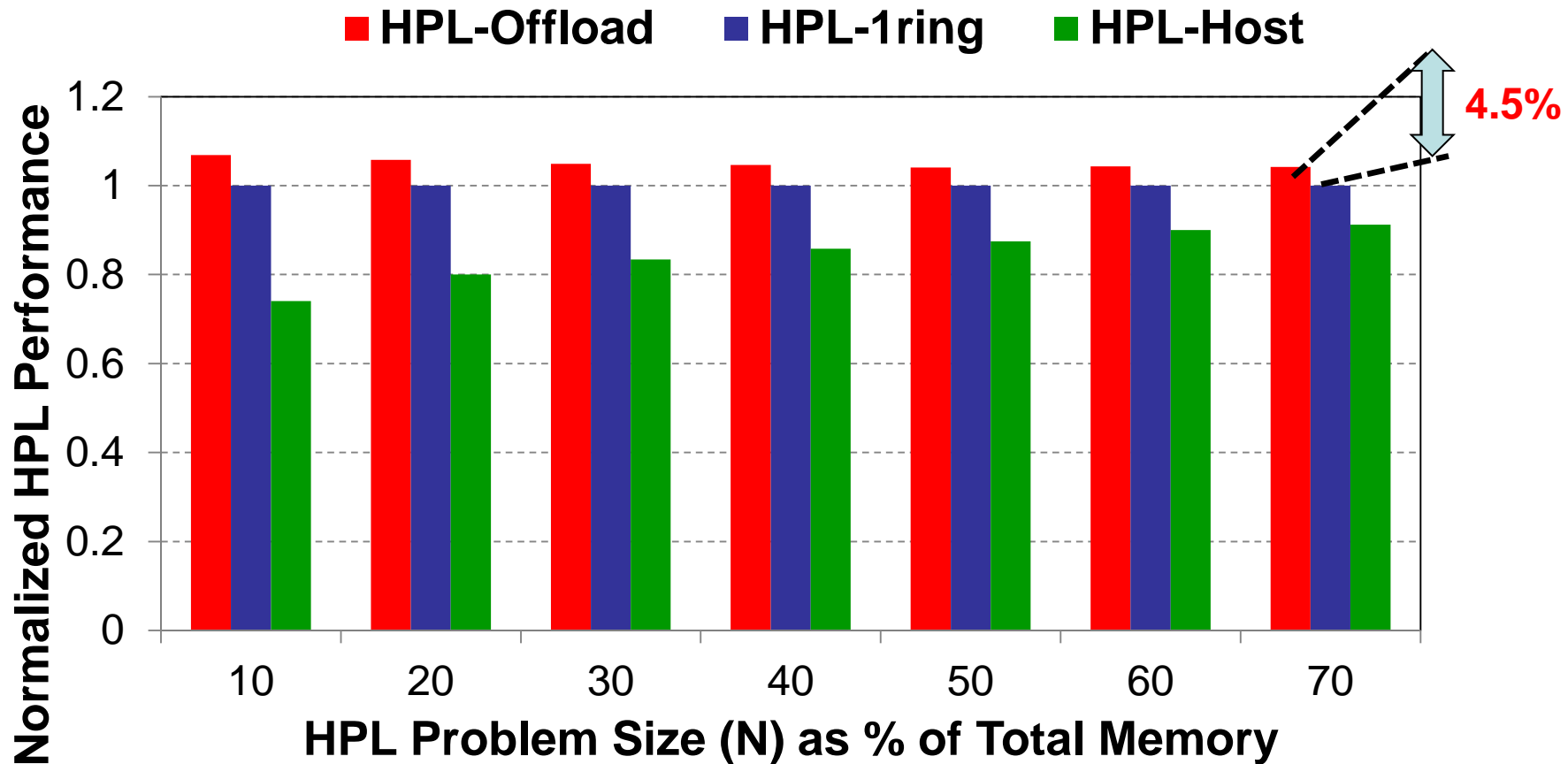
**Bcast Latency Comparison with 256 Processes**

**Bcast-Offload delivers good overlap, without sacrificing on communication latency!**

# Outline

- Introduction
- Motivation and Problem Statement
- Designing MPI\_Ibcast with Collective Offload
- Experimental Evaluation
  - Micro-Benchmark evaluations
  - Experiments with HPL
  - Impact of System Noise
- Conclusions and Future work

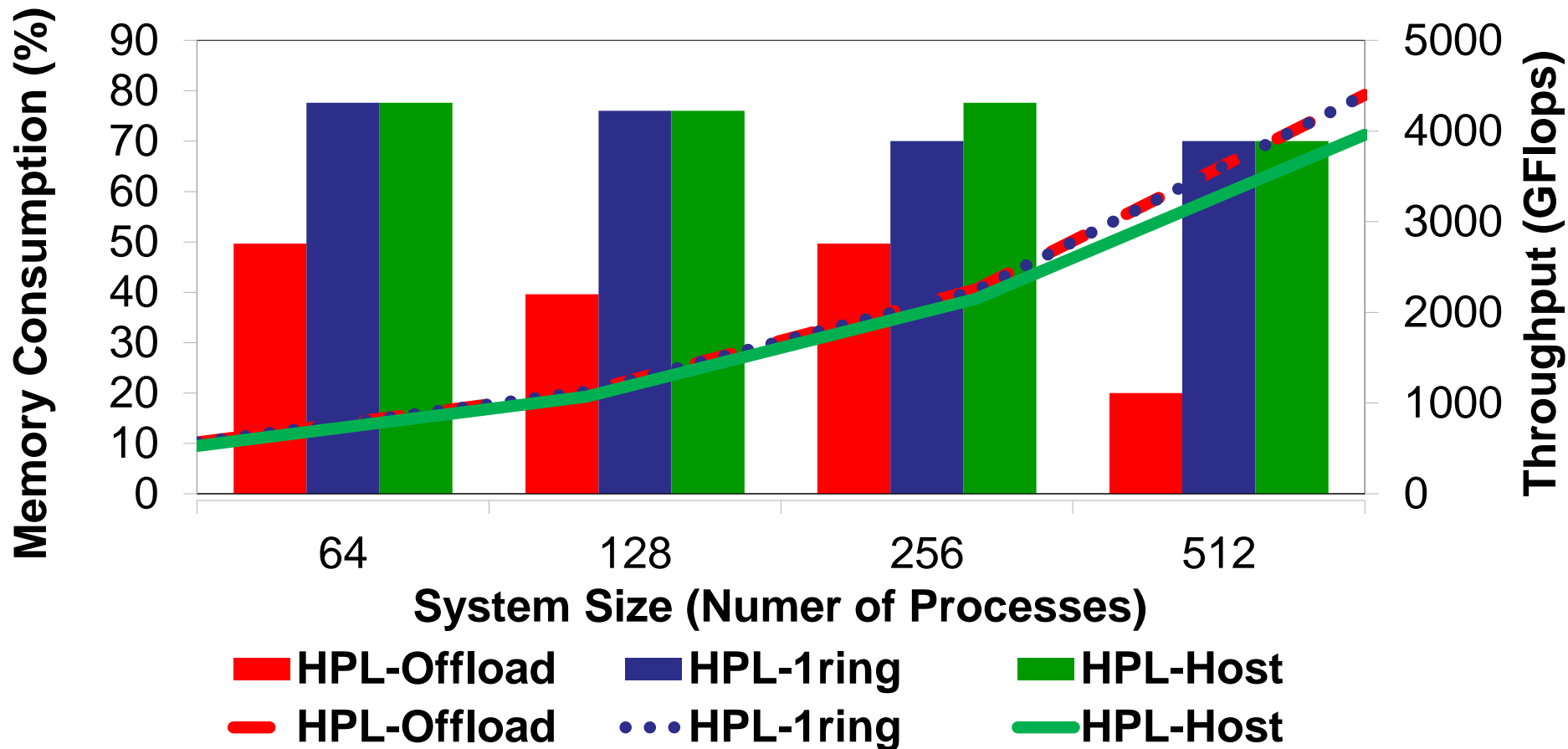
# HPL Performance



## HPL Performance Comparison with 512 Processes

**HPL-Offload consistently offers higher throughput than HPL-1ring and HPL-Host. Improves peak throughput by up to 4.5 % for large problem sizes**

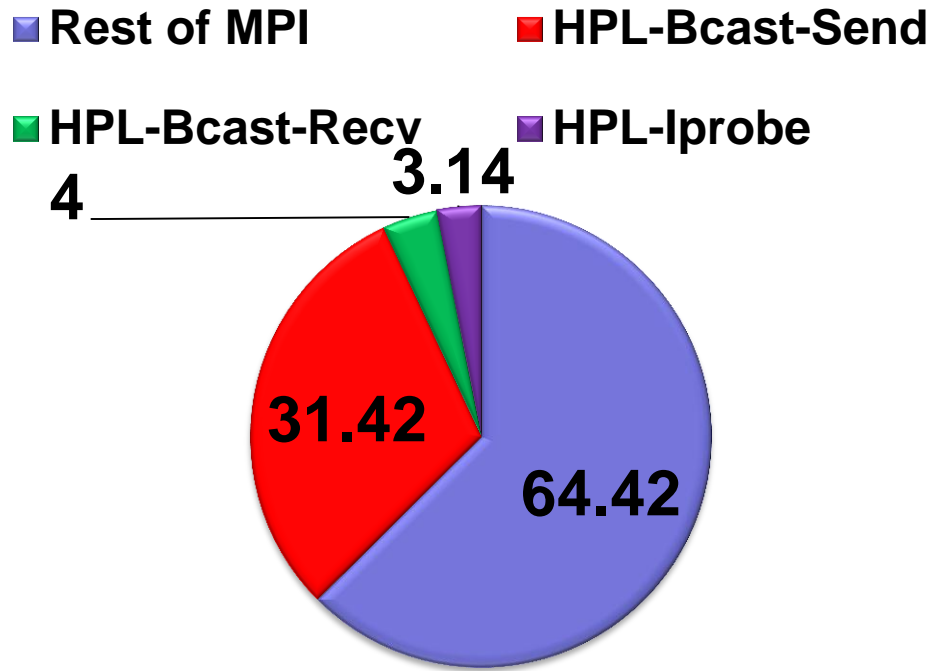
# HPL Performance and Problem Size



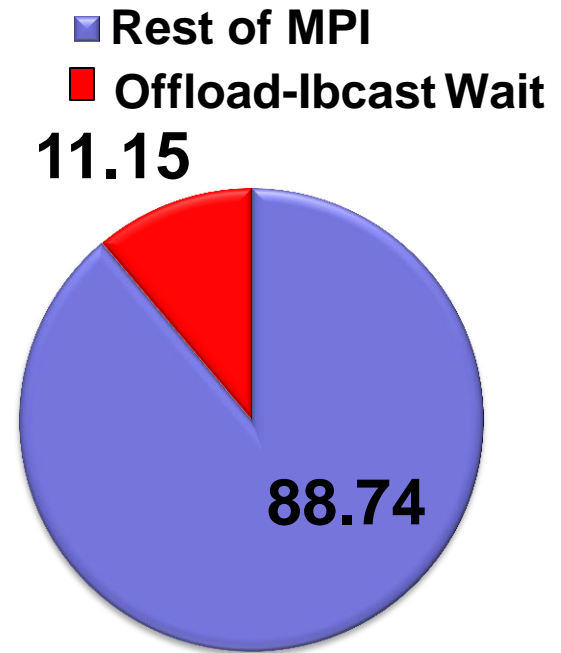
HPL Peak Performance and Corresponding Problem Sizes

**HPL-Offload surpasses the peak throughput of HPL-1ring with significantly smaller problem sizes and run-times!**

# HPL Performance Analysis



**MPI Time % Comparison with HPL-1-ring Algorithm**



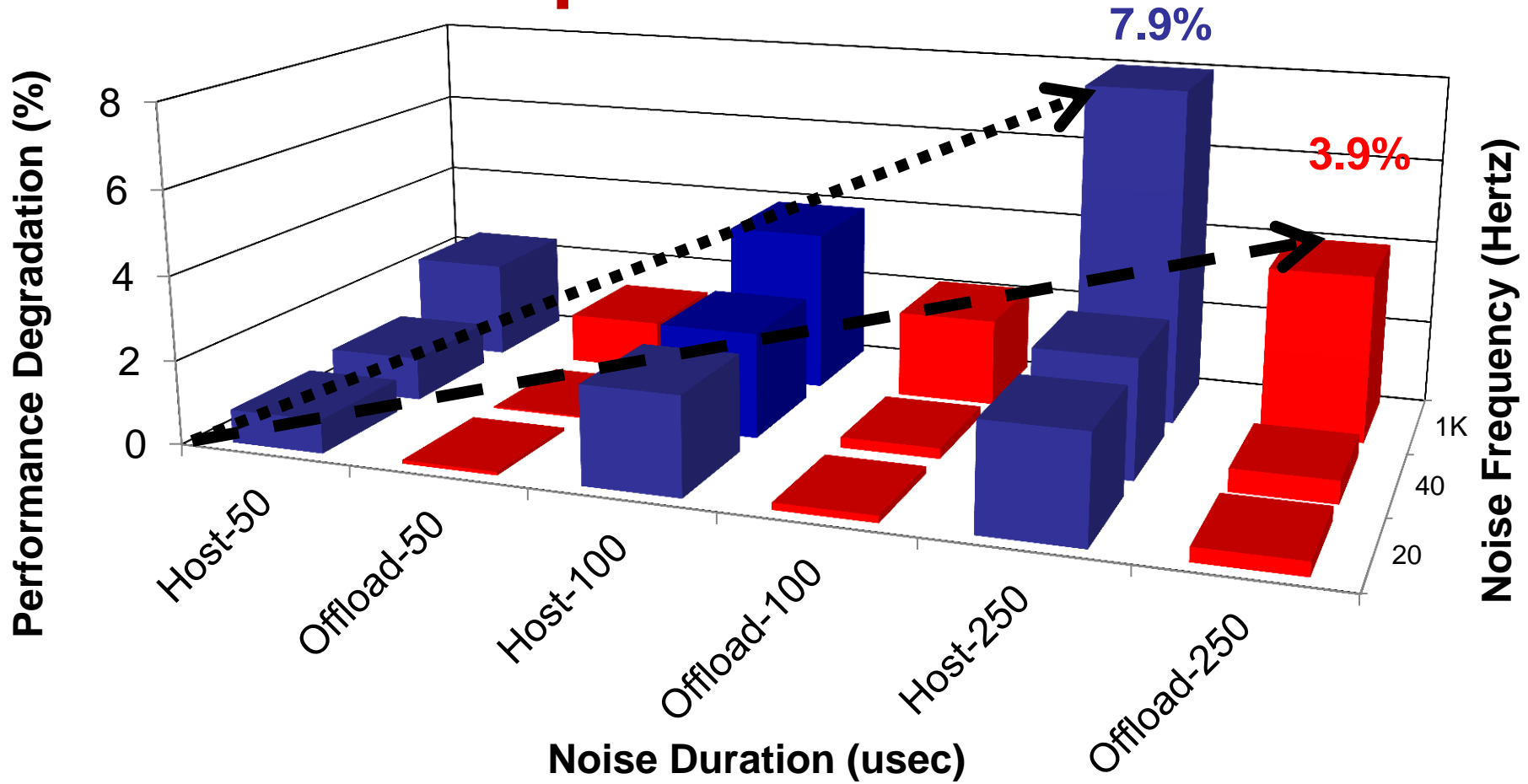
**MPI Time % Comparison with HPL-Offload-lbroadcast Algorithm**

HPL's Run-time analysis with 256 Processes and N=177,000  
**HPL with Offload-lbroadcast has lower MPI-level overheads than HPL's 1-ring Broadcast Algorithm**

# Outline

- Introduction
- Motivation and Problem Statement
- Designing MPI\_Ibcast with Collective Offload
- Experimental Evaluation
  - Micro-Benchmark evaluations
  - Experiments with HPL
  - **Impact of System Noise**
- Conclusions and Future work

# Impact of Noise



DGEMM Throughput degradation due to System Noise

Host-based Throughput drops by about 7.9%  
**Offload-Throughput drops by only about 3.9%**

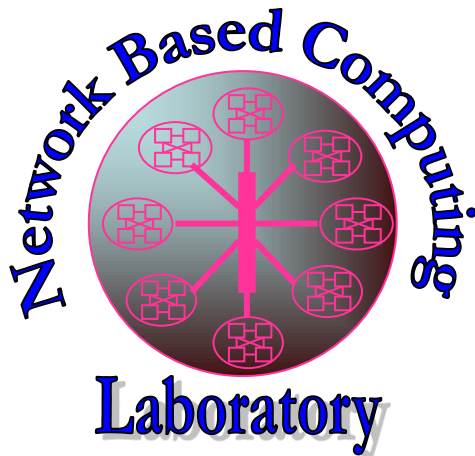
# Conclusions and Future Work

- Proposed MPI\_Ibcast shows near perfect overlap
- MPI\_Ibcast based on CORE-Direct improves HPL's peak throughput by upto 4.5% with 512 processes
- HPL-Offload achieves peak throughput with significantly smaller problem sizes and shorter run-times
- MPI\_Ibcast based on Offload is more resilient to noise

## *Future work*

- Extend Offload-based techniques for other MPI collectives and study their benefits with real applications
- Support for Offload-based collectives will be available in future MVAPICH2 releases

# Thank you!



<http://mvapich.cse.ohio-state.edu>

(1){kandalla, subramon, viennej, pai, surs, panda}@cse.ohio-state.edu

(2)ktomko@osc.edu

(1)Network-Based Computing Laboratory, Ohio State University

(2)The Ohio Supercomputer Center