Multi-threaded UPC Runtime with Network Endpoints: Design Alternatives and Evaluation on Multi-core Architectures

Miao Luo, **Jithin Jose,**

Sayantan Sur & D. K. Panda

Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University, USA

Outline

- Introduction
- Problem Statement
- UPC Runtime Design Choices
- Multi-endpoint Design
- Performance Evaluations
- Conclusion & Future Work

Outline

- Introduction
- Problem Statement
- Runtime Design Choices
- Multi-endpoint Design
- Performance Evaluations
- Conclusion & Future Work

Introduction

- Partitioned Global Address Space (PGAS) is an emerging parallel programming model:
  - Shared memory abstraction on distributed memory machines
  - User can control data layout and work distribution to take advantage of locality
  - High-productivity and better applicability with multi-core and network architecture
- Unified Parallel C (UPC) is one of the most popular PGAS languages:
  - Based on parallel extensions to the C language
  - Ease of programmability
  - Suitable for multi-core and accelerator clusters

UPC Runtime Choice – Pthreads vs Processes

- Thread-based runtime
  - Low-latency intra-node communication
  - Low-level load balancing schemes
  - Criticized for poor network performance
- Process-based runtime
  - Good inter-node communication due to independent network context
  - Need kernel/shared memory schemes for intra node communication
- Runtime design choice has an impact on:
  - Performance, Portability, Interoperability, Support for irregular parallelism

Outline

- Introduction
- Problem Statement
- Existing Runtime Designs
- Multi-endpoint Design
- Performance Evaluations
- Conclusion & Future Work

Problem Statement

- *With the advent of multi-cores, should the UPC runtime itself be multi-threaded?*
- *How it will affect the performance and productivity aspects?*
- *Can it provide implicit load balancing?*

Outline

- Introduction
- Problem Statement
- Existing Runtime Designs
- Multi-endpoint Design
- Performance Evaluations
- Conclusion & Future Work

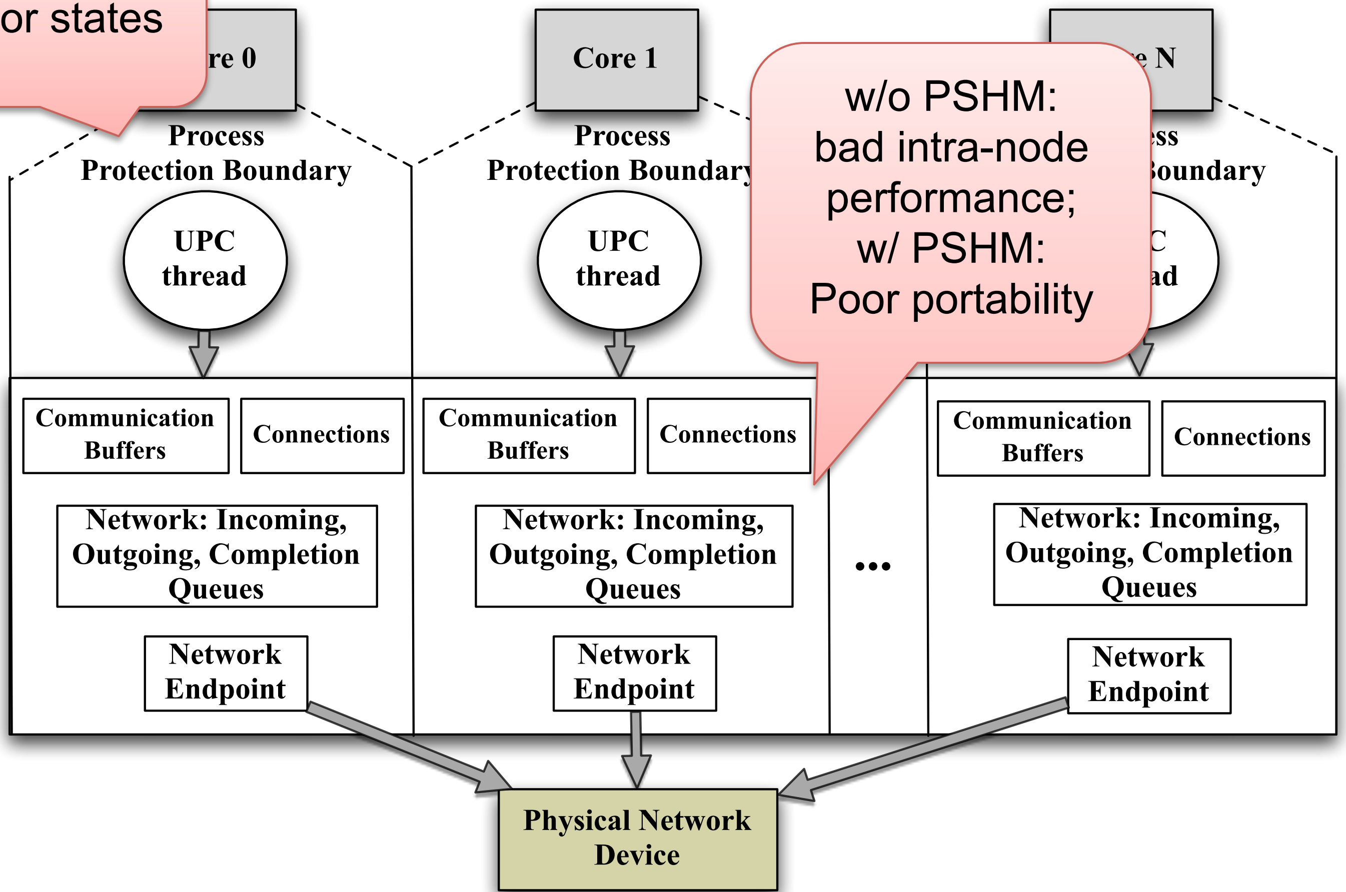
Existing UPC Runtimes and Multithreaded Runtime Design Choices

- Process Based Runtime
- Process Based Runtime with intra-node communication optimizations (PSHM)
- Multi-threaded Runtime – Global lock
- Multi-threaded Runtime – Fine-grained lock

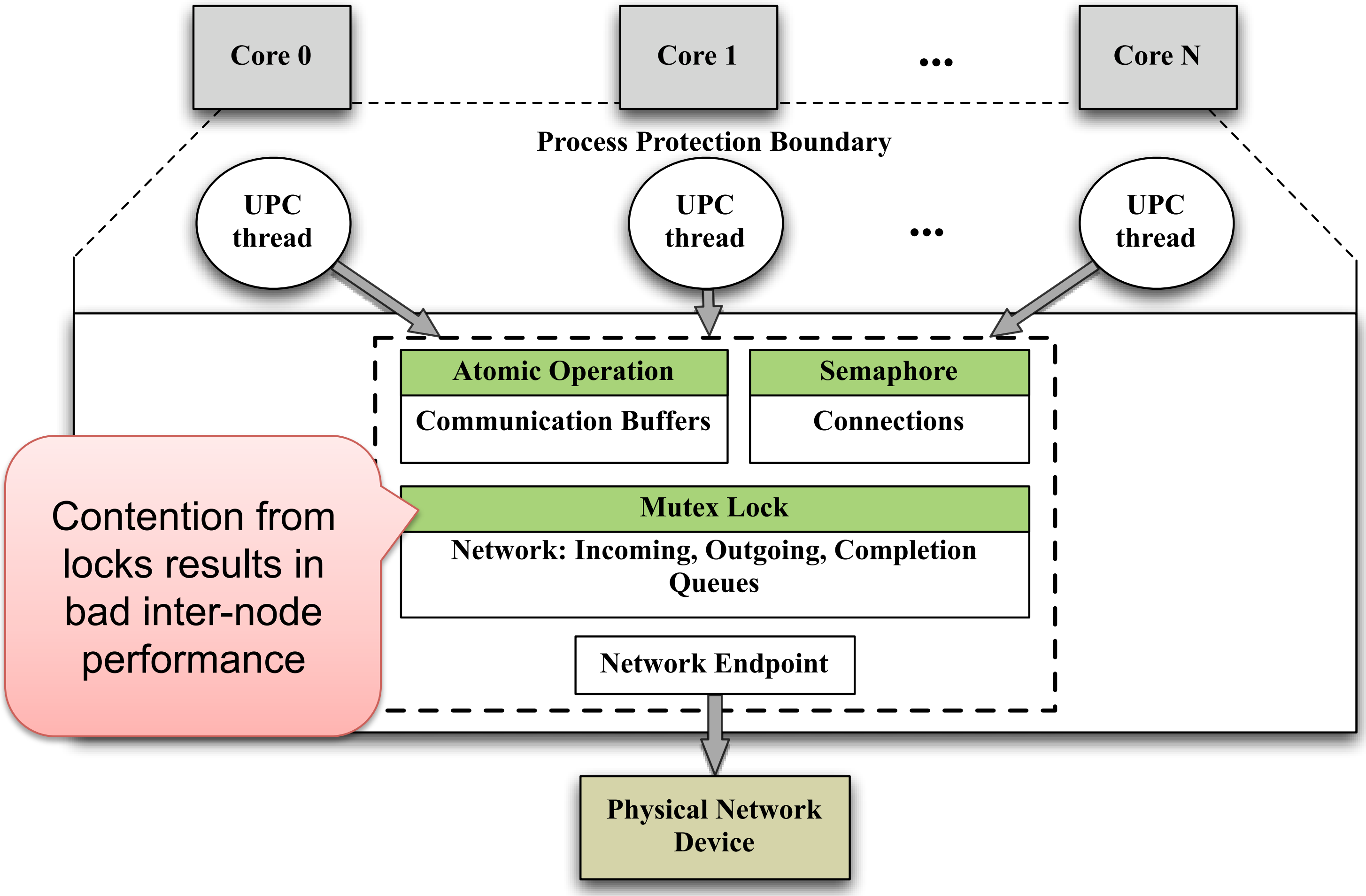
Process based Runtime

Difficult to share processor states

w/o PSHM: bad intra-node performance; w/ PSHM: Poor portability



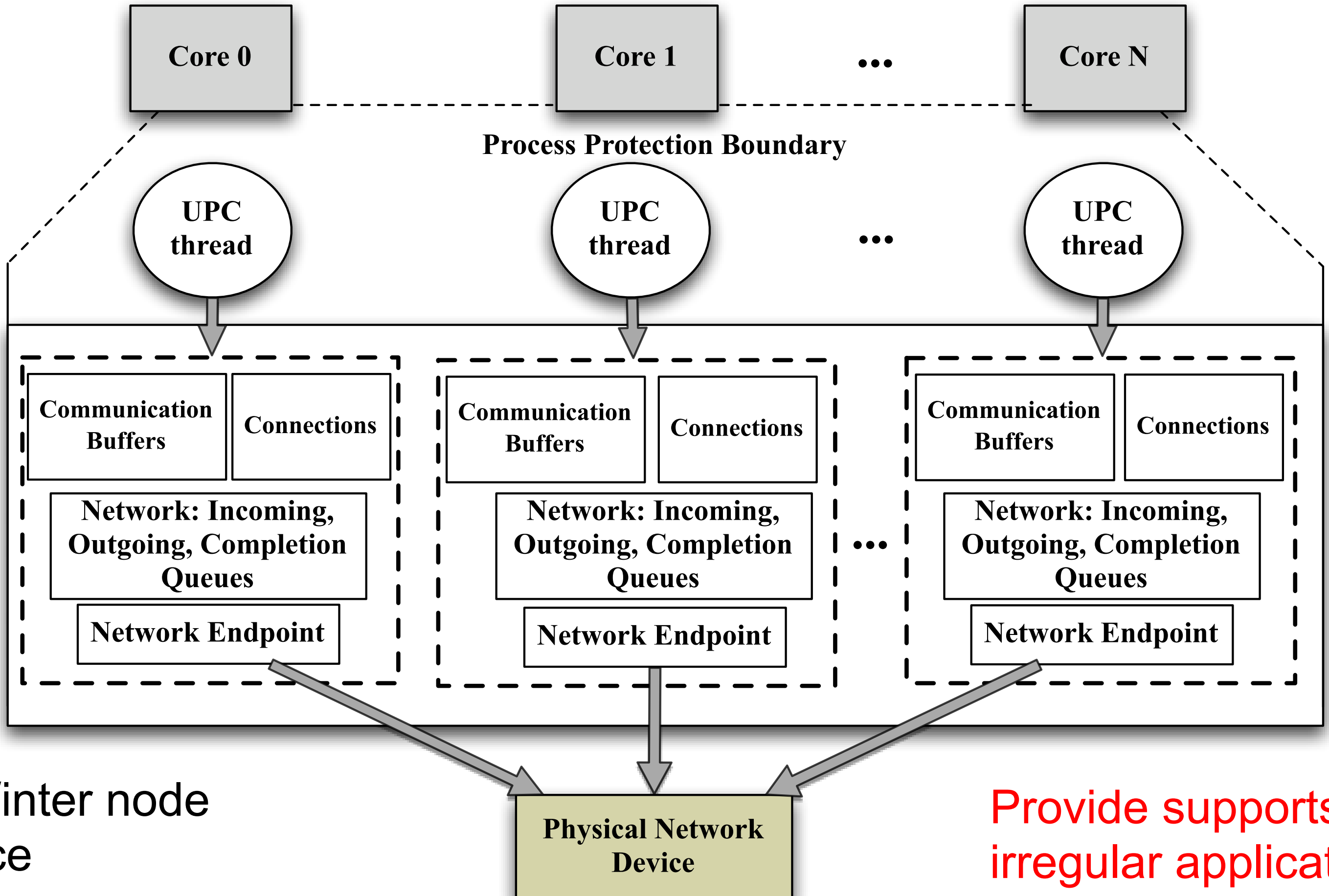
Multi-threaded Runtime with Single Network Endpoint



Outline

- Introduction
- Problem Statement
- Existing Runtime Designs
- Multi-endpoint Design
- Performance Evaluations
- Conclusion & Future Work

Multi-threaded Runtime with Multiple Network Endpoint

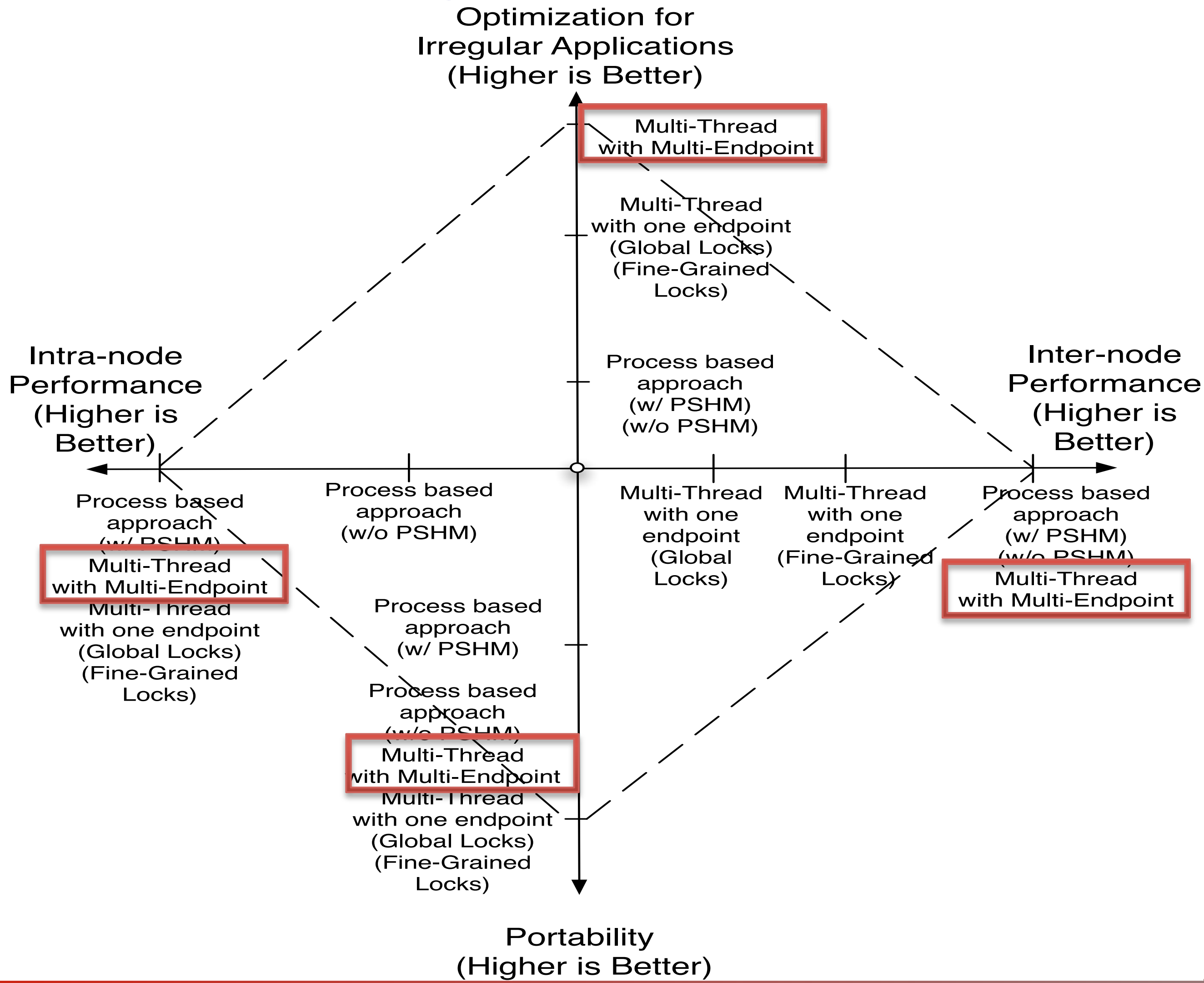


Good intra/inter node performance

Good portability

Provide supports to irregular applications through work stealing and helper threads

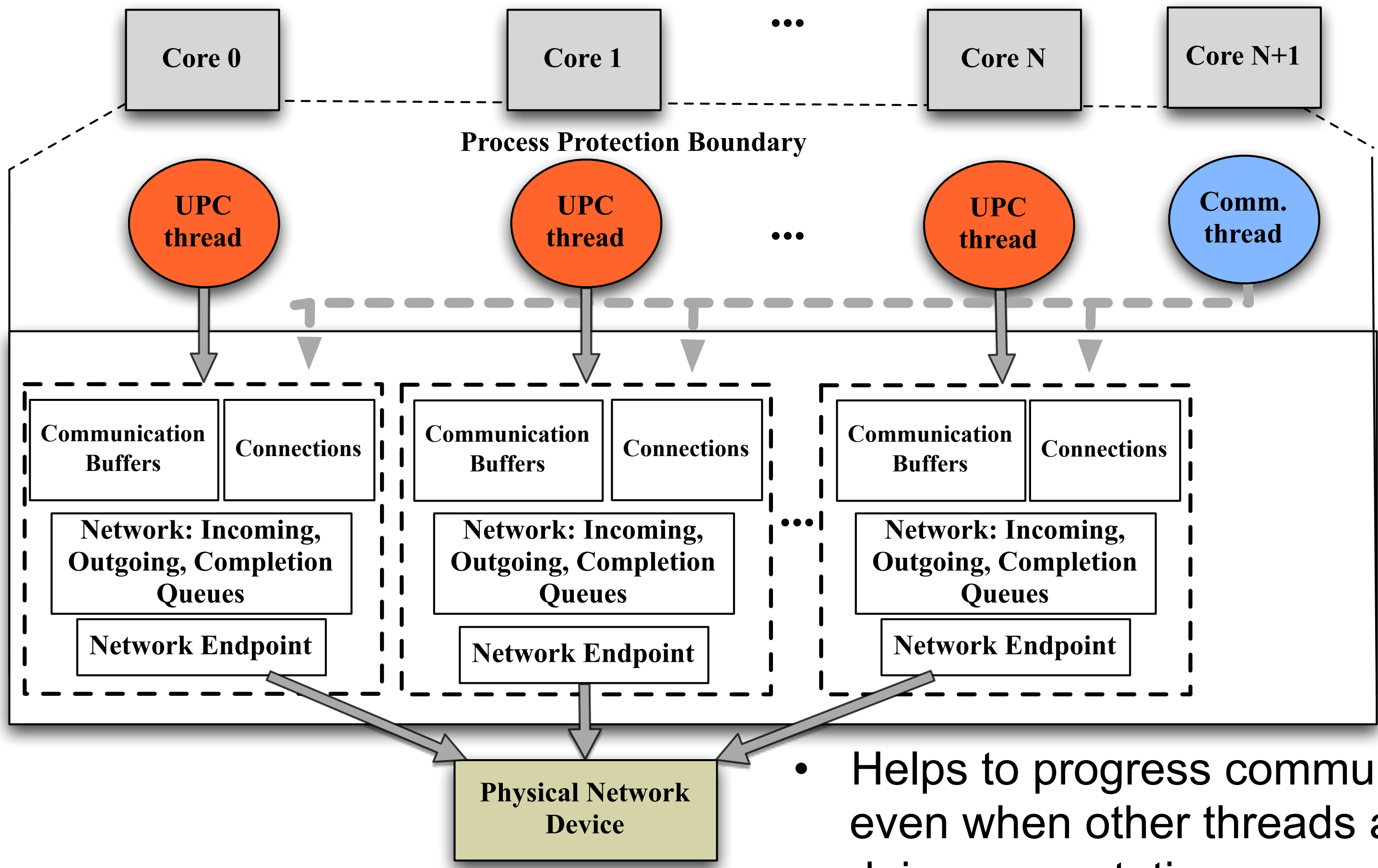
Multi-Dimensional Comparison of Design Alternatives



Load Balancing For Irregular Applications – Helper Threads and Work Stealing

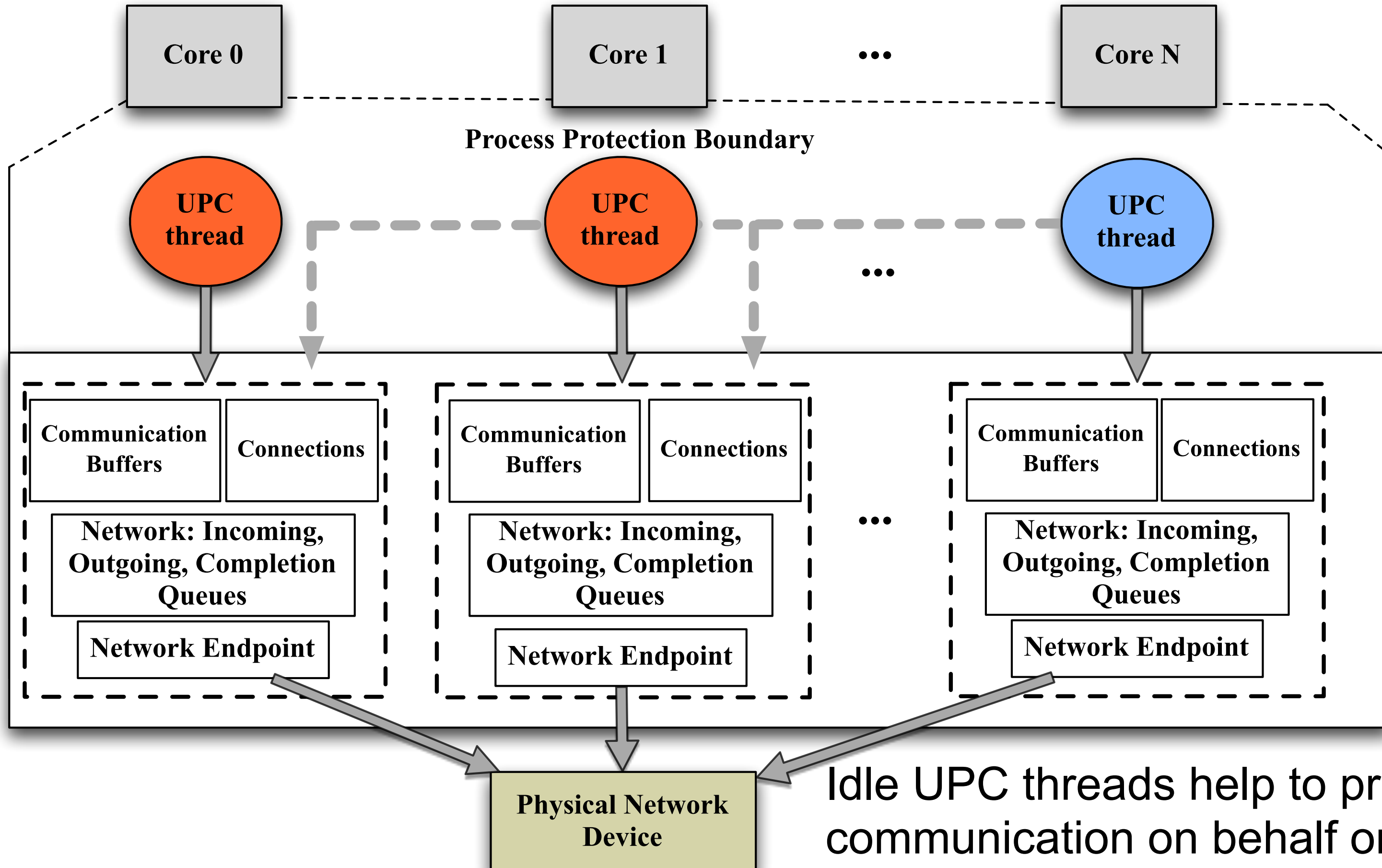
- Irregular applications are hard to express
- Handling irregularity in the application hurts programmer productivity
- Compiler based approaches: may be limited by lack of dynamic characteristics
- We present runtime level load balancing schemes, and provide transparent optimization to irregular applications:
  - Application independent
  - Can provide generic load balancing, even for applications that don't have specific application-level optimizations

Dedicated Communication Threads (Helper Thread)



- Helps to progress communication, even when other threads are busy doing computation
- Helper thread is not visible at UPC application level

Work Stealing for Efficient Asynchronous Remote Methods



Idle UPC threads help to progress communication on behalf on busy threads

Unified Communication Runtime (UCR)

- Aims to unify communication runtimes of different parallel programming models
  - J. Jose, M. Luo, S. Sur and D. K. Panda, **Unifying UPC and MPI Runtimes: Experience with MVAPICH**, (PGAS'10)
- Design of UCR evolved from MVAPICH/MVAPICH2 software stacks (http://mvapich.cse.ohio-state.edu/)
  - Used by more than 1,810 organizations in 65 countries
- UCR provides interfaces for Active Messages as well as one-sided put/get operations
- Support for Scalable Graph Traversals
  - J. Jose, S. Potluri, M. Luo, S. Sur, D. K. Panda, **UPC Queues for Scalable Graph Traversals – Design and Evaluation on InfiniBand Clusters -**(PGAS''11)
- UCR in Cloud Computing domain
  - J. Jose, H. Subramoni, M. Luo, S. Sur, D. K. Panda, et al., **Memcached Design on High Performance RDMA Capable Interconnects**, (ICPP'11)
  - J. Huang, X. Ouyang, J. Jose, D.K. Panda et al, **High Performance Design of Hbase with RDMA over InfiniBand – (**IPDPS'12)

Outline

- Introduction
- Problem Statement
- Existing Runtime Designs
- Multi-endpoint Design
- Performance Evaluations
- Conclusion & Future Work

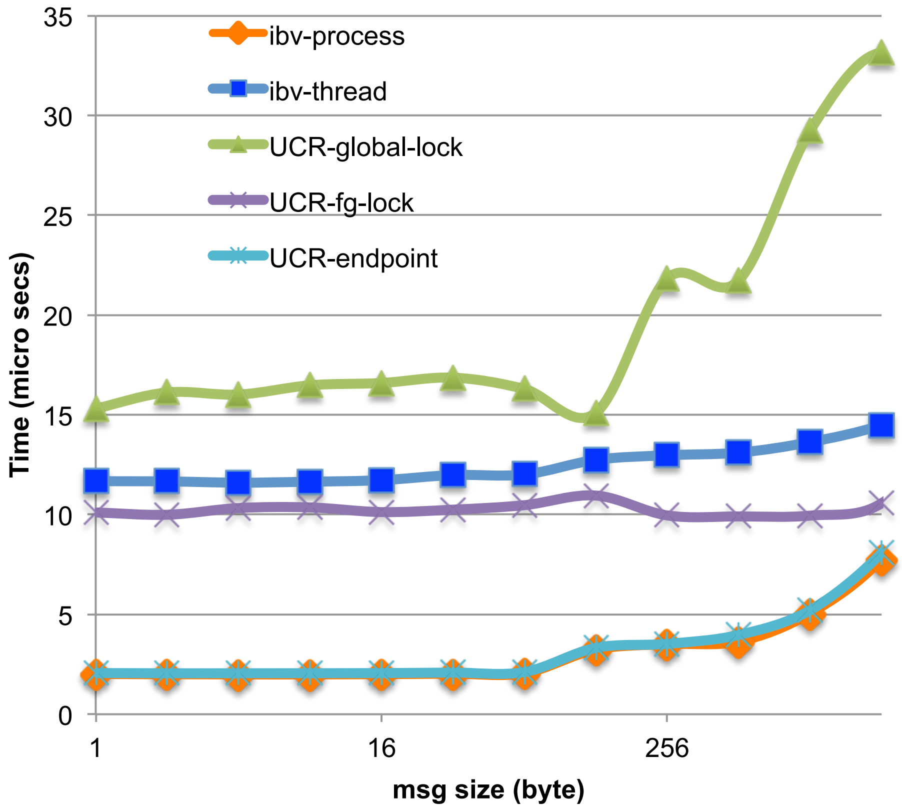
Experimental Platform

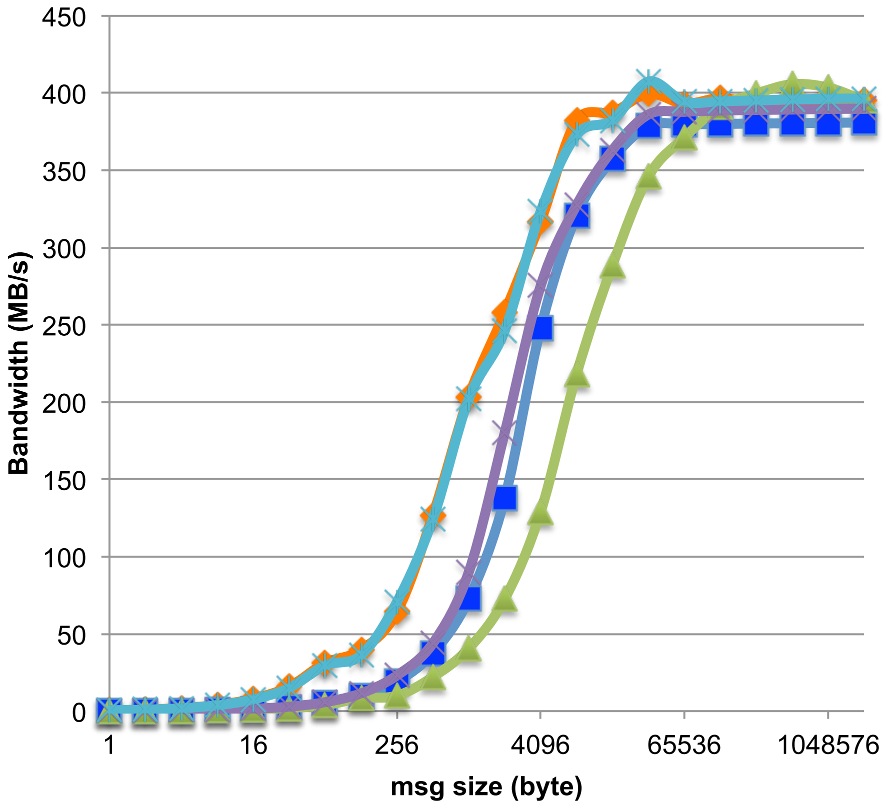
- Intel Westmere cluster
  - 1,280 cores where each node has eight Intel Xeon EE5630 processors, organized into two sockets of four cores each clocked at 2.53 GHz.
  - Mellanox ConnectX QDR HCAs (32 Gbps data rate)
  - L1 cache is 32K, L2 is 256 K and shared L3 (among cores in one socket) is 12 M.
  - Each node has 12 GB of main memory
  - Red Hat Enterprise Linux Server Release 5.4

Performance Evaluation

- Evaluated existing UPC runtime choices and design alternatives:
  - ibv-process: the process based runtime from Berkeley UPC GASNet IBV-conduit
  - ibv-thread: the multi-threaded runtime with single endpoint from Berkeley UPC GASNet IBV-conduit
  - UCR-global-lock: Multi-threaded runtime with single endpoint (global locks to achieve thread safety)
  - UCR-fg-lock: Multi-threaded runtime with single endpoint (fine grained locks to achieve thread safety)
  - UCR-endpoint: Multi-threaded runtime with multiple endpoints.
- Evaluation based on Latency, Bandwidth, Message Rate, Load balancing
- Berkeley UPC version 2.12.1 with PSHM (sysv) enabled is used for ibv-process and ibv-thread

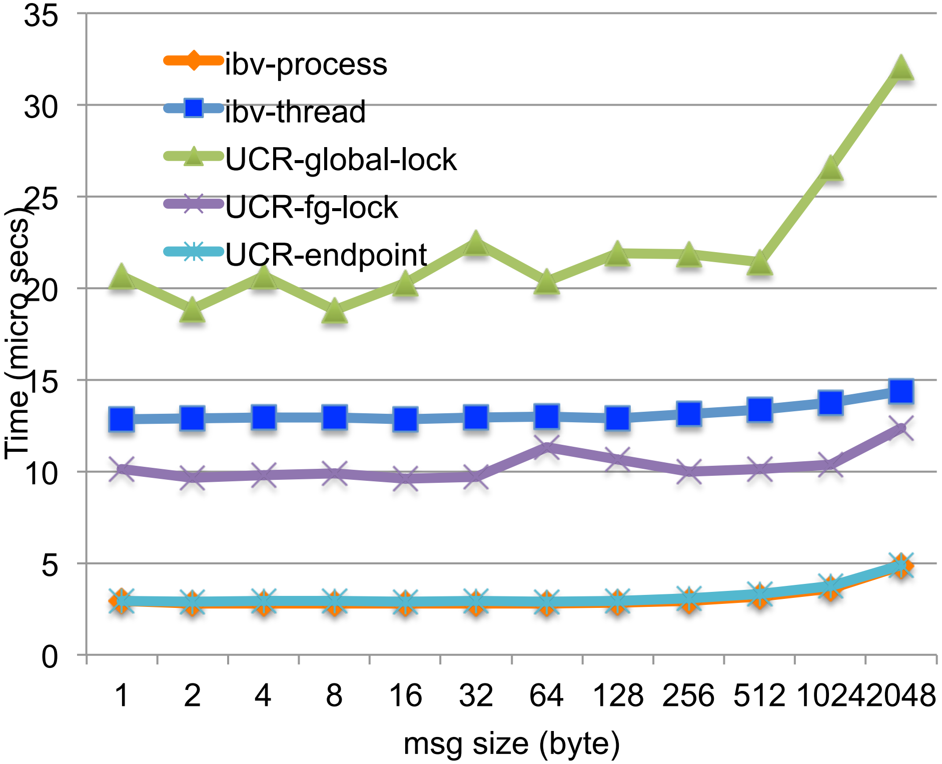
Micro-benchmark Performance upc_memput

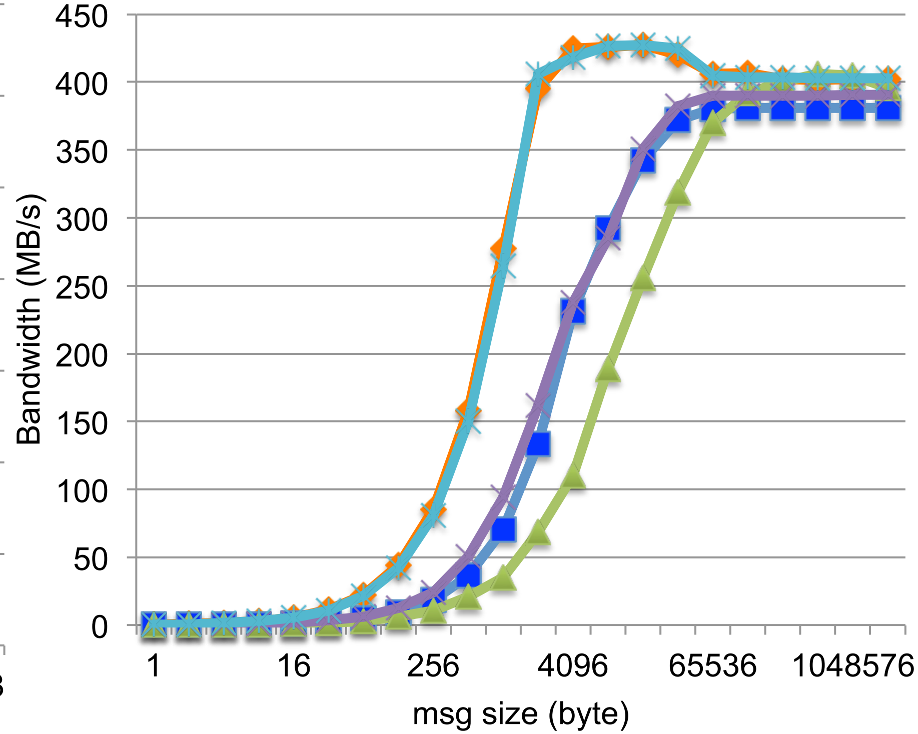




- UPC memput operation latency and bandwidth micro benchmark
- Latency reduced by 80% compared to single endpoint multi-thread design.
- 2X improvement in bandwidth for middle range message size

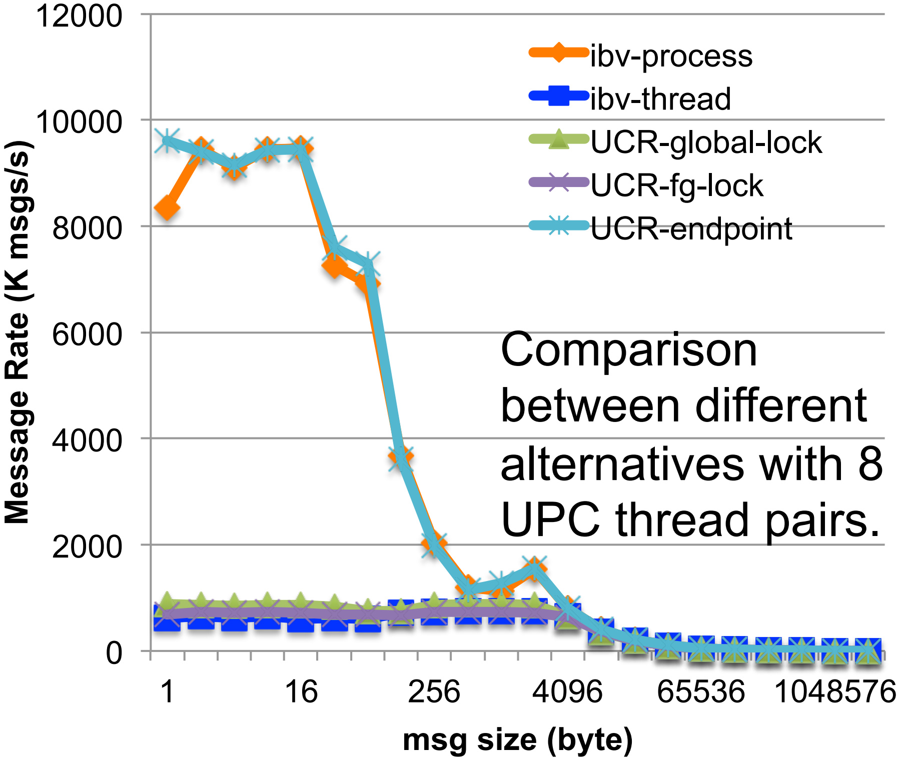
Micro-benchmark Performance upc_memget

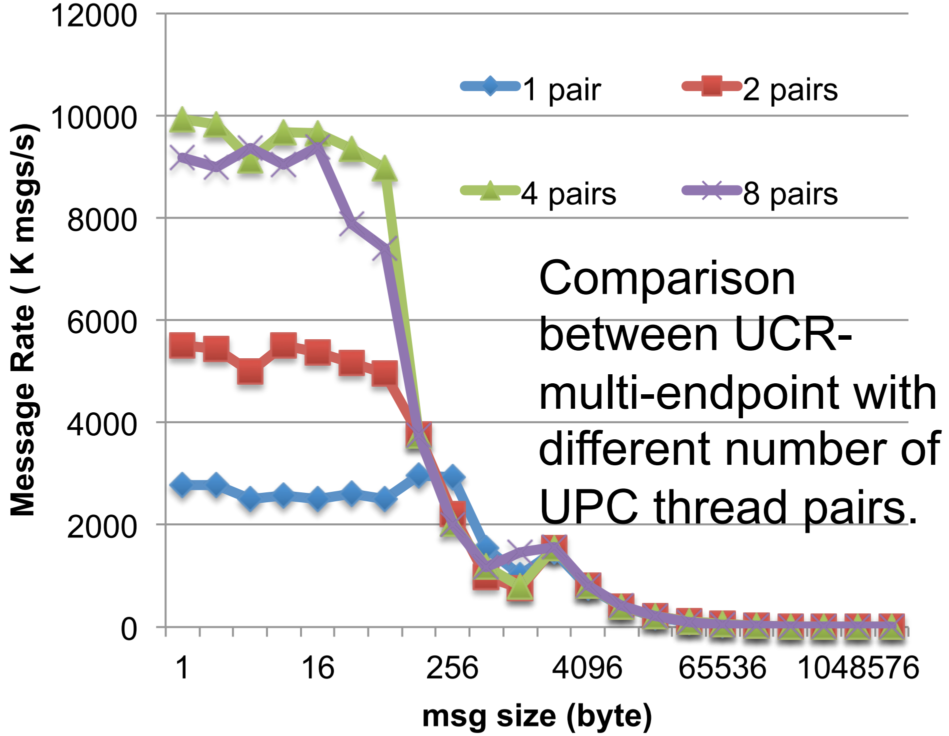




- Similar for UPC memget micro benchmark
- Latency is reduced by 76% from single endpoint multi-thread design
- Bandwidth is doubled for middle range message sizes

Message Rate Evaluation





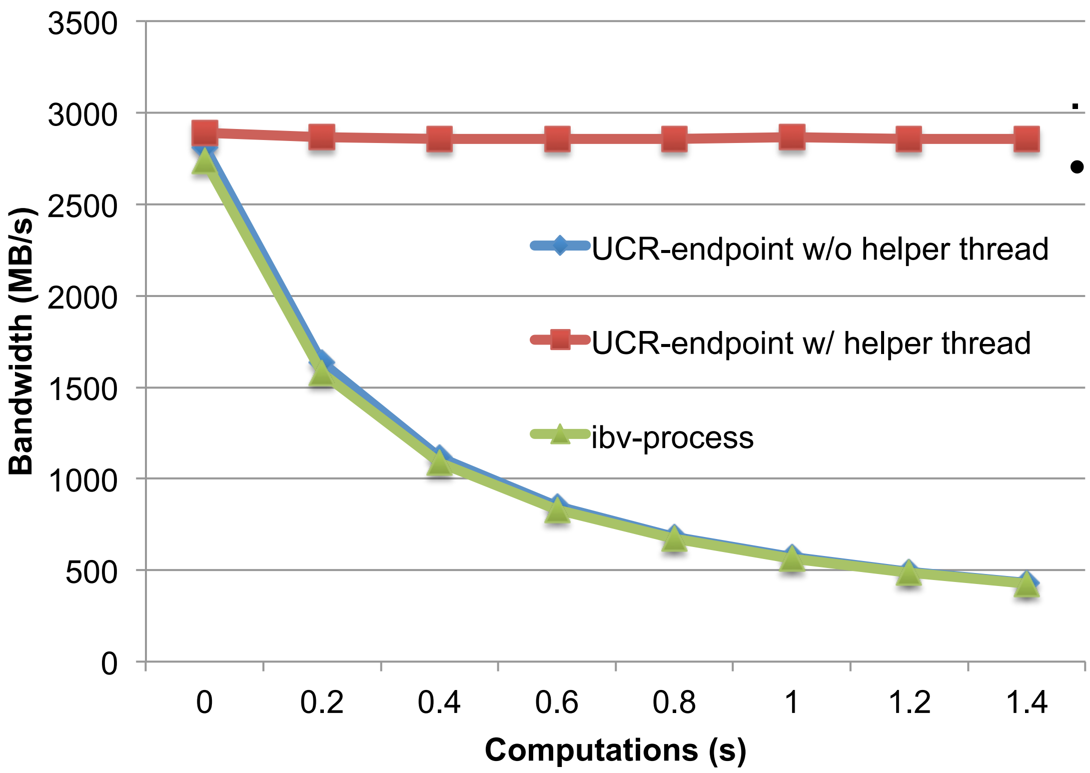
- Left: Multi-threaded with single endpoint can only achieve one-eighth as compared to ibv-process and UCR-endpoint
- Right: Message rate of small messages is dependent on number of endpoints; Most concurrency in the network adapter is already utilized by four pairs

Load Balancing Evaluation for Multi-endpoint Design: Helper Thread

- Benchmark Description:
  - 14 UPC threads are grouped into seven pairs on two nodes
  - Senders send 1MB message to peers and wait for acknowledgement
  - Receivers perform a defined amount of computation before polling network

Load Balancing Evaluation for Multi-endpoint Design: Helper Thread

- Bandwidth Results with Computation on Receiver Side:

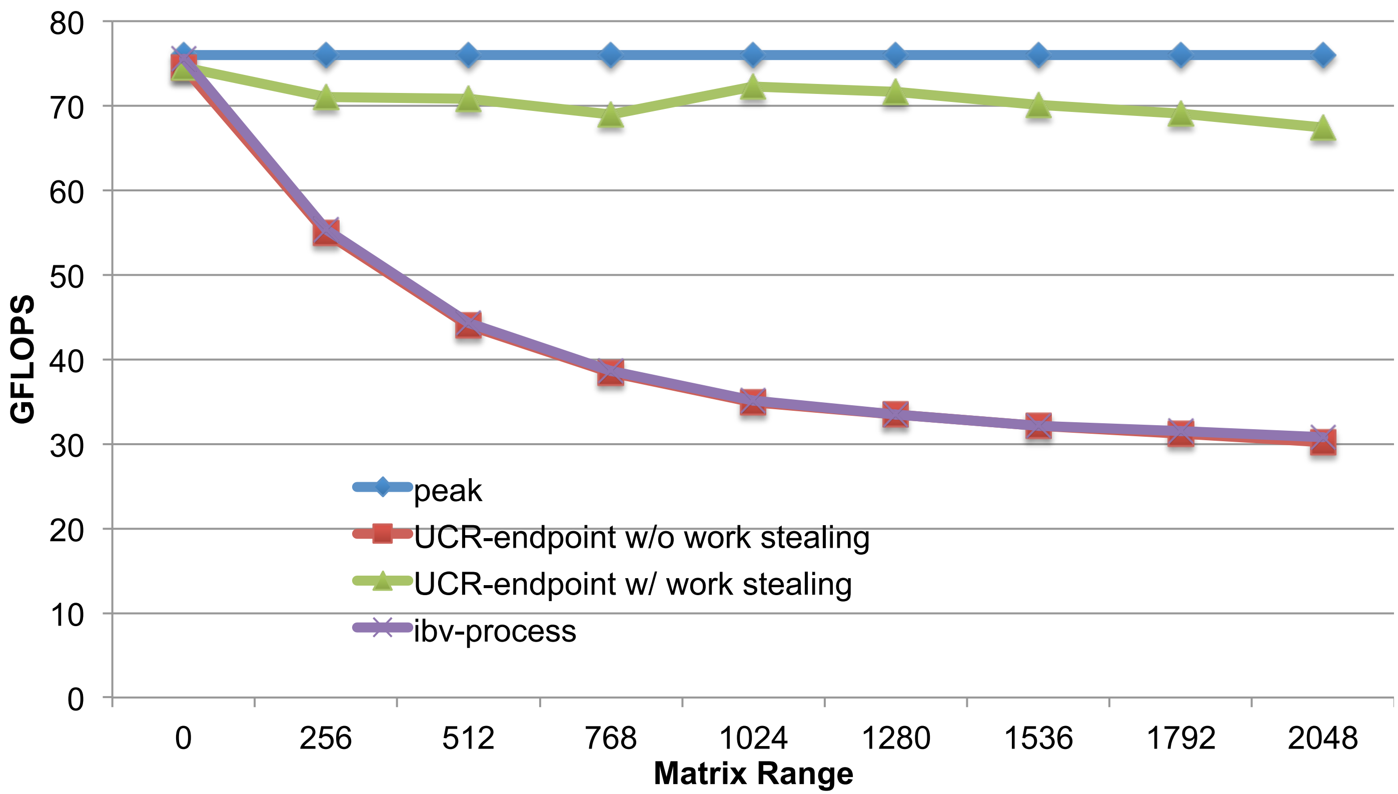


- Helper threads can keep bandwidth fully utilized while UPC threads are busy computation

Load Balancing Evaluation for Multi-endpoint Design: Work Stealing

- Work stealing benchmark:
  - 16 UPC threads are grouped into eight pairs on two nodes.
  - Computation is represented by DGEMM
  - Senders send varying computation to peers and wait for acknowledgement
  - The average workload is matrix size equals to 2,000
  - As matrix range x increases, three UPC threads have workload as 2,000 – x; another three UPC threads receive 2,000 + x workload; the left two UCP threads will keep getting requests of 2,000
  - Receivers reply back once they finish corresponding computations

Load Balancing Evaluation for Multi-endpoint Design: Work Stealing



- Without work stealing, receivers with light workload become idle and CPU cycles are wasted
- With work stealing, idle threads consume workload for busy threads: GFLOPS is kept close to peak value

Outline

- Introduction
- Problem Statement
- Existing Runtime Designs
- Multi-endpoint Design
- Performance Evaluations
- Conclusion & Future Work

Conclusion

- Explored multiple design alternatives for UPC runtime implementation on multi-core architectures
- Designed a new multi-threaded runtime with multiple network endpoint design in UCR
- Significant performance improvements over available multi-threaded runtime
  - 80% lower latency as compared to existing multi-threaded designs
  - 2X improvement on bandwidth for medium size messages
- Efficient load balancing using ‘Helper Thread’ and ‘Work stealing’ techniques
  - 90% of peak efficiency
  - 1.3 times better than existing multi-threaded Runtime design

Future Work

- Application level evaluations using Irregular applications such as Graph500, Barnes Hut, etc
- UPC collectives using multi-threaded design with multiple endpoints
- Multi-threaded, multi-endpoint support to hybrid applications of MPI and UPC

Thank You!

{luom, jose, surs, panda}@cse.ohio-state.edu

Network-Based Computing Laboratory

http://nowlab.cse.ohio-state.edu/

MVAPICH Web Page

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