UPC Queues for Scalable Graph Traversals: Design and Evaluation on InfiniBand Clusters

Jithin Jose, Sreeram Potluri, Miao Luo,

Sayantan Sur & D. K. Panda

Network-Based Computing Laboratory

Department of Computer Science and Engineering

The Ohio State University, USA

Outline

- Introduction
- Motivation
- Problem Statement
- UPC Queues
- Redesigning Applications using UPC Queues
- Performance Evaluation
- Conclusion & Future Work

Introduction

- PGAS languages getting more & more popular
  - Ease of programmability
  - Control of data layout
  - Shared memory abstraction on distributed memory systems
- UPC – one of the most popular PGAS language
- Graphs – ubiquitous model in analytical workloads
- Graph Benchmarks
  - Graph500 (http://www.graph500.org)
  - Unbalanced Tree Search (UTS)
- We focus on “UPC for Graph Algorithms / Irregular Applications”

Outline

- Introduction
- Motivation
- Problem Statement
- UPC Queues
- Redesigning Applications using UPC Queues
- Performance Evaluation
- Conclusion & Future Work

Motivation

- Graphs – powerful representations of relations, process dynamics
  - Used in a variety of Scientific & Engineering fields
  - Basic graph algorithms are key components in many real-life applications
- Irregular Communication Characteristics
  - Uses load balancing - work stealing, work sharing
  - Producer-Consumer relationship exists
- How to express producer-consumer relationships in UPC?

Expressing Producer-Consumer Relationships in UPC

- UPC Locks
  - upc_lock()/upc_unlock() to provide mutual exclusion
  - Easy to use
  - Lock contention
  - Each transaction translates to 3 messages over network
- Replicating Resources
  - Consumers keep dedicated receive buffers for each producer
  - Better performance than locks
  - Polling overhead, Increased memory requirement O(N)

# UPC Threads	4	16	64	256
UPC Locks	24	135	610	2610
Replicating Resources	6	31	138	610

Average Transaction Time (us)

Outline

- Introduction
- Motivation
- Problem Statement
- UPC Queues
- Redesigning Applications using UPC Queues
- Performance Evaluation
- Conclusion & Future Work

Problem Statement

- What are the challenges involved in implementing producer-consumer relationships in UPC?
- How can these be addressed?
- How to redesign applications using new schemes?
- What would be the impact on performance?

Outline

- Introduction
- Motivation
- Problem Statement
- UPC Queues
- Redesigning Applications using UPC Queues
- Performance Evaluation
- Conclusion & Future Work

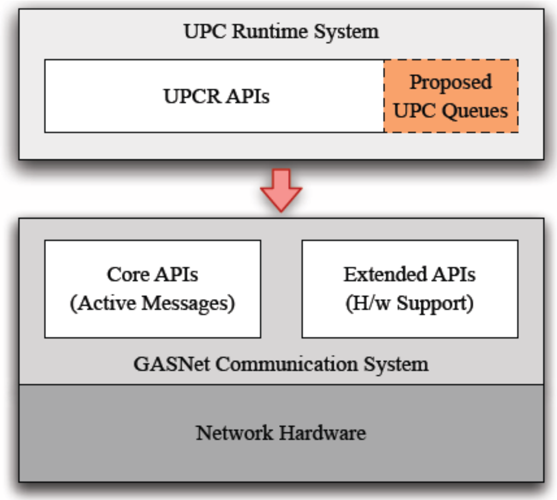
Overview of UPC Queues

- Provides an easy way to express producer-consumer relationships
- Producer just puts data onto the consumer queue
- Better programmability
- Optimized network utilization
- Suits well for irregular applications

UPC Queues – Operations

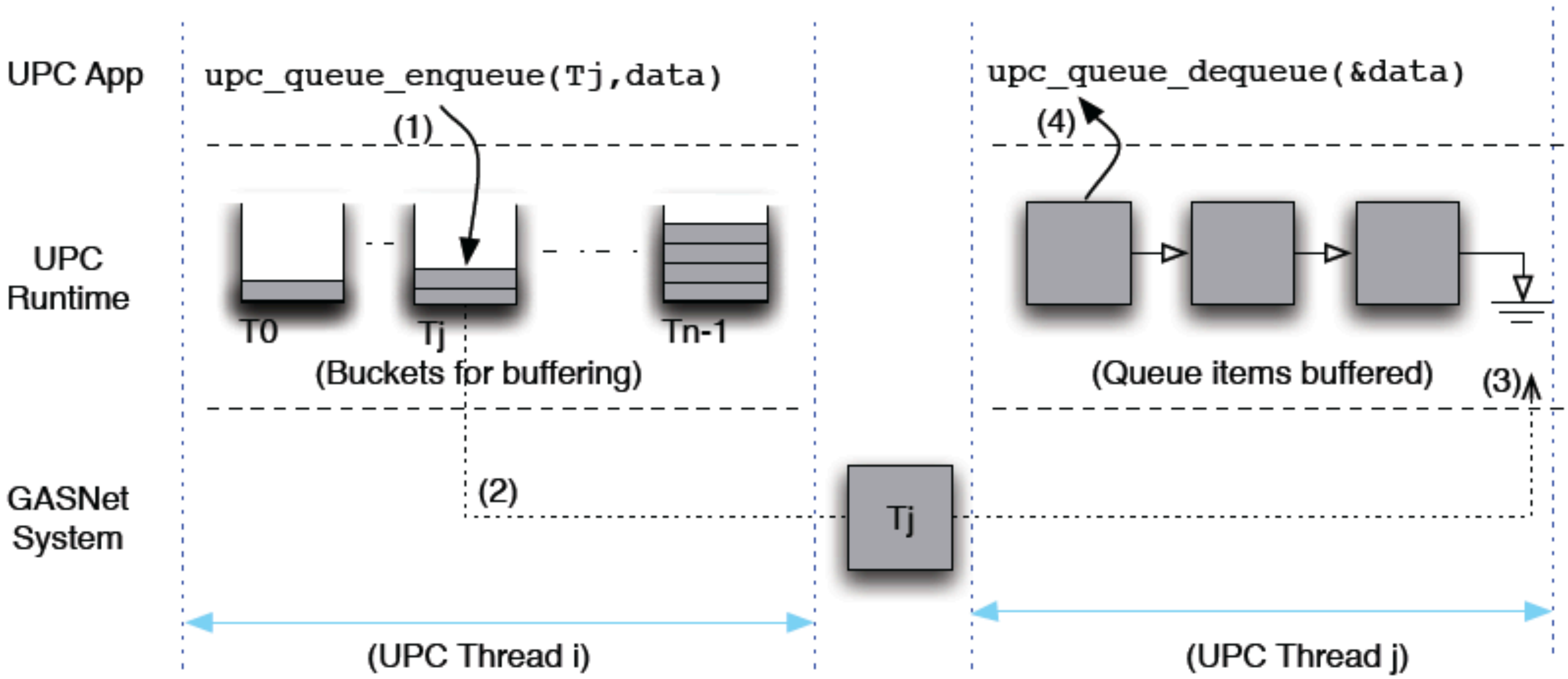
- `upc_queue_create()`
  - Collective call, initializes queue
  - Input arguments to enable/disable coalescing, configure bucket size
  - Returns a handle to be used in subsequent queue operations
- `upc_queue_enqueue()/upc_queue_dequeue()`
  - Enqueues/dequeues queue item
  - Buffer/Send queue item based on coalescing option
- `upc_queue_flush()`
  - Used only if coalescing is enabled
  - Flushes out local buffers
- `upc_queue_destroy()`
  - Collective call
  - Releases any resources allocated for the queue

UPC Queues - Design



- Key design characteristics
  - Programmability
  - Scalability
  - Low latency
  - Portability
- Implemented in UPC Runtime (UPCR) layer
- Coalescing for better network utilization (optional)
- Buckets for ‘true’ consumers
- Uses Active Messages for sending queue items
  - Implemented over ‘medium’ active message
- Can be used with any network conduit

UPC Queues – Operation



(1) Enqueue Operation – (Buffering)
(2) Sending out the queue item over Active Message
(3) Buffering at Receiver Side
(4) Dequeue Operation – dequeues from buffer

NETWORK-BASED COMPUTING LABORATORY

Outline

- Introduction
- Motivation
- Problem Statement
- UPC Queues
- Redesigning Applications using UPC Queues
- Performance Evaluation
- Conclusion & Future Work

Redesigning Graph Benchmarks using UPC Queues

- Graph500
- Unbalanced Tree Search

Graph500

- Set of benchmarks to evaluate scalability of supercomputing clusters
  - Data Intensive & Irregular communication pattern
  - http://www.graph500.org/
  - Announced at ISC'10 & first ranking appeared at SC'10
- 3 Comprehensive benchmarks
  - Search, Optimization & Edge Oriented
  - Sequential, OpenMP, XMT and MPI implementations available
- Developed UPC version of Concurrent Search benchmark
  - First UPC implementation (to the best of our knowledge)
  - Based on Graph500 Specification v1.2

UPC Implementation

- Concurrent Search kernel
  - Breadth First Search traversal
  - Graph generated in Compressed Sparse Row (CSR) format
  - CSR is distributed among UPC threads
- Visited vertices need to be given to `Owner' UPC threads for traversing successor vertices
- Level synchronization
- Design Evaluations
  - Replication of Resources (replicate resource)
  - UPC Queues (queues)

Unbalanced Tree Search (UTS)

- Exhaustive Search on an Tree with dynamic load balancing
  - http://barista.cse.ohio-state.edu/wiki/index.php/UTS
  - UPC, Shmem, MPI, OpenMP, Pthreads, Chapel, X10 versions available
- Tree constructed on the fly
- Variation in the sizes of subtrees at different nodes
  - Load balancing required
- Work-stealing and work-sharing versions available

UTS Enhancement

- Used 'uts_upc_enhanced' benchmark as reference
  - Idle UPC threads request for work
  - Request made using a shared resource protected using upc_lock()
  - Response by updating a shared resource
- New design using Queues
  - Uses Queues for requests/response
- Design Evaluations
  - Release Version 1.1 of 'uts_upc_enhanced' (base version)
  - New implementation using Queues (queue)

Outline

- Introduction
- Motivation
- Problem Statement
- UPC Queues
- Redesigning Applications using UPC Queues
- Performance Evaluation
- Conclusion & Future Work

Performance Evaluation

- Experimental Setup
- Microbenchmark Evaluations
- Evaluation using Graph Benchmarks
  - Graph500
  - Unbalanced Tree Search (UTS)

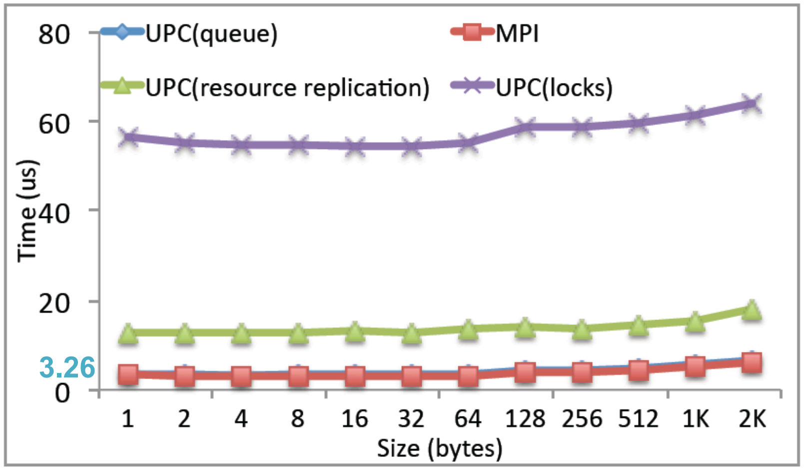
Experimental Platform

- Intel Westmere Cluster
  - 144 Compute nodes
  - Each node has 8 processor cores on 2 Intel Xeon 2.67 GHz Quad-core CPUs
  - 12 GB main memory, 160 GB hard disk
  - MT26428 QDR ConnectX HCAs (36Gbps)
  - Red Hat Enterprise Linux Server 5.4 (Tikanga)
- Berkeley UPC 2.12.2
  - GASNet-IBV
  - GASNet-UCR
- MVAPICH (v1.2) library used in microbenchmark evaluations

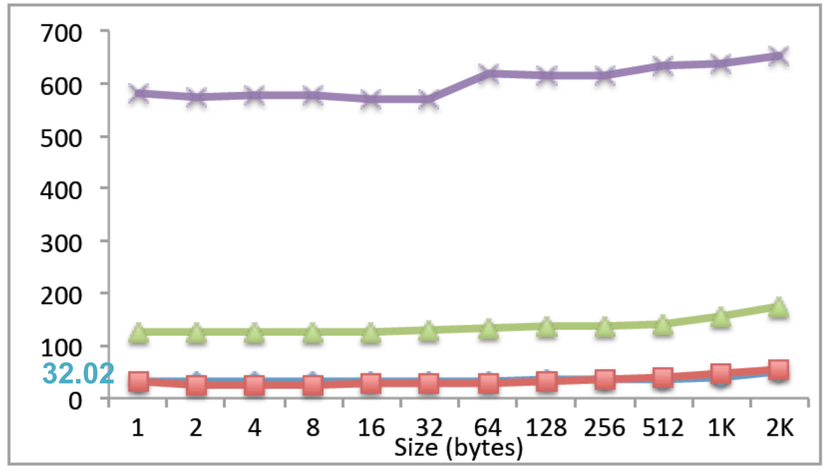
Unified Communication Runtime (UCR)

- 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/)
- UCR provides interfaces for Active Messages as well as one-sided put/get operations
- Multi-end point design
  - M. Luo, J. Jose, S. Sur and D. K. Panda, **Multi-threaded UPC Runtime with Network Endpoints: Design Alternatives and Evaluation on Multi-core Architectures**, (HiPC'11)
- UCR in Data Center domain
  - J. Jose, H. Subramoni, M. Luo, S. Sur, D. K. Panda, et al., **Memcached Design on High Performance RDMA Capable Interconnects**, (ICPP'11)

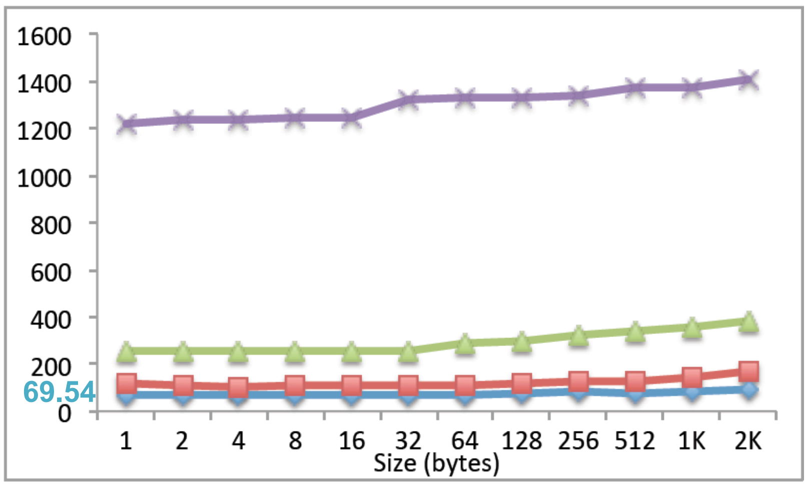
Performance Analysis - Latency



(Latency - 8 procs)



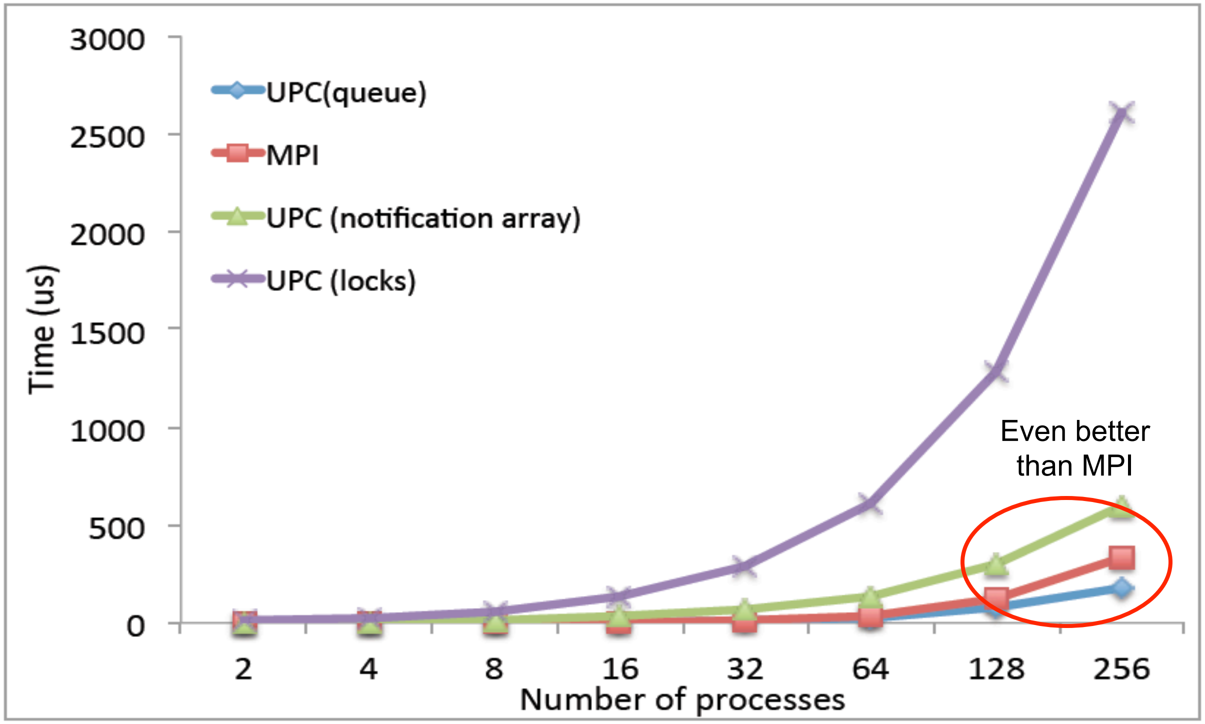
(Latency - 64 procs)



(Latency - 128 procs)

- UPC Queues perform better than other schemes
- For 128 process run
  - 54% improvement over replication of resources
  - 16% improvement than MPI

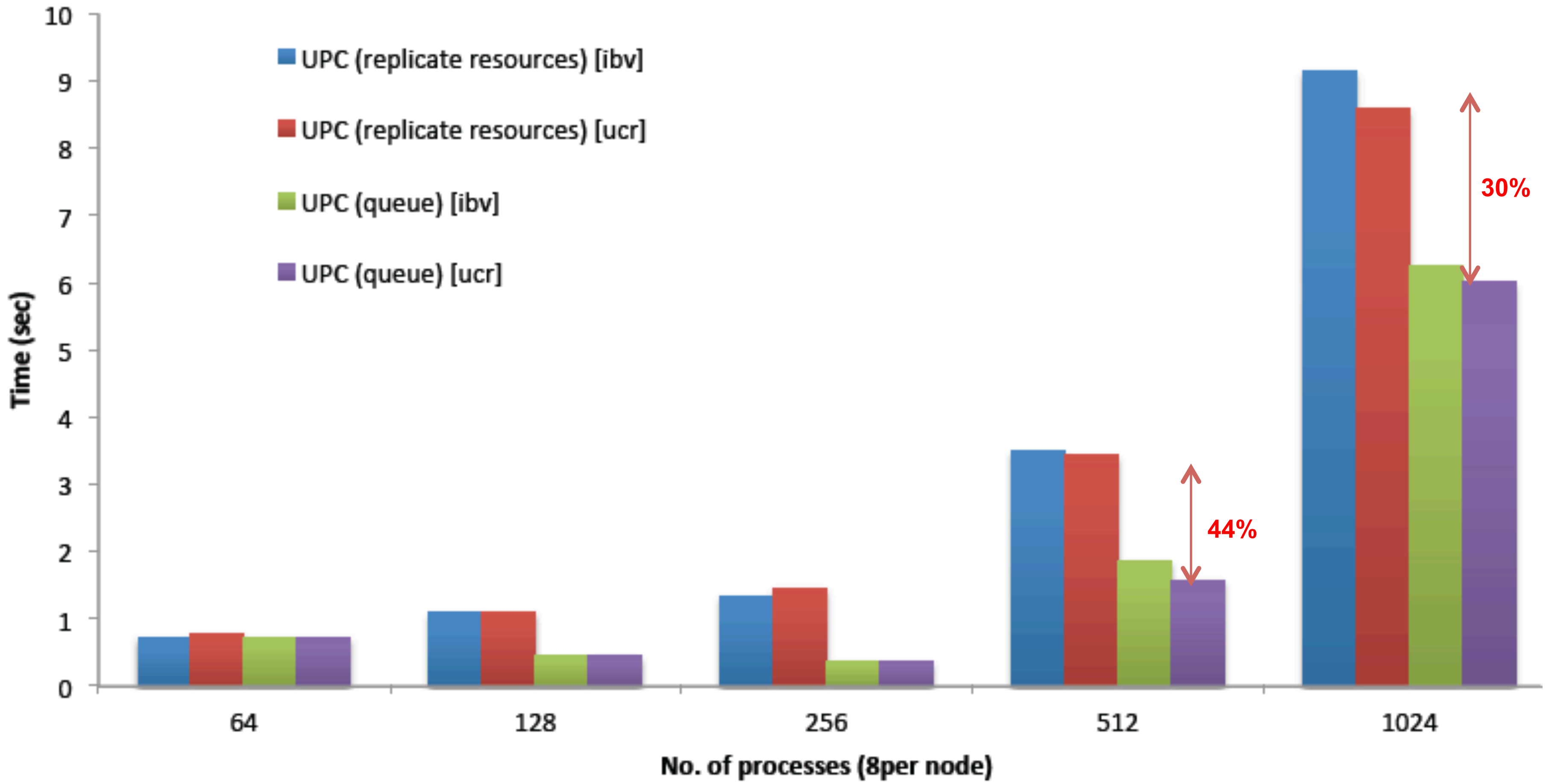
Performance Analysis - Scalabilty



(Scalability Analysis - 128 byte queue item)

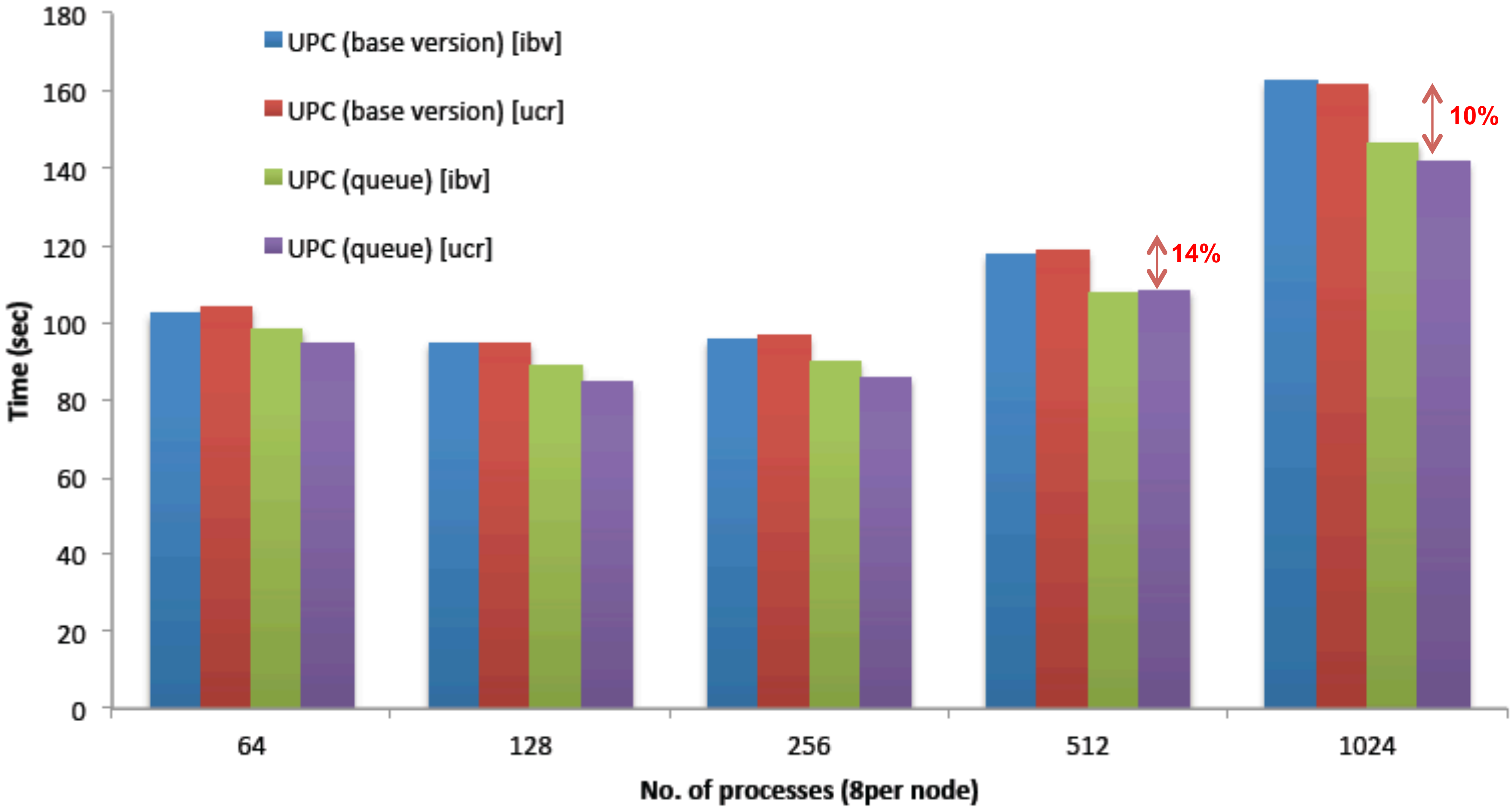
- UPC Queues scales well
- For 256 process run
  - 55% improvement over resource replication
  - 23% improvement over MPI

Graph500 Results



- Workload – Scale:24, Edge Factor:16 (16 million vertices, 256 million edges)
- 44% Improvement over base version for 512 UPC-Threads
- 30% Improvement over base version for 1024 UPC-Threads

Unbalanced Tree Search



- Workload - T1WL (270 billion nodes)
- 14% Improvement over base version for 512 UPC-Threads
- 10% Improvement over base version for 1024 UPC-Threads

Outline

- Introduction
- Motivation
- Problem Statement
- UPC Queues
- Redesigning Applications using UPC Queues
- Performance Evaluation
- Conclusion & Future Work

Conclusion & Future Work

- Introduced UPC Queues concept
  - Suits for producer-consumer relationship implementations
  - Least overhead & Highly Scalable
  - Easy to use API's
- Performance Improvements
  - Graph500 - 44% and 30% for 512 & 1024 UPC-thread runs respectively
  - UTS - 14% and 10% for 512 & 1024 UPC-thread runs respectively
- In this work, we accentuate on the concept, not the API syntax
  - Queue API's can be molded to match UPC style
  - Use of efficient compiler translation techniques possible

Thank You!

{jose, potluri, luom, 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/