NETWORK-BASED COMPUTING LABORATORY

Designing Scalable Graph500 Benchmark with Hybrid MPI+OpenSHMEM Programming Models

Jithin Jose[1], Sreeram Potluri[1], Karen Tomko[2] and Dhabaleswar K. (DK) Panda[1]

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

[2]Ohio Supercomputer Center,
Columbus, Ohio, USA

OHIO STATE

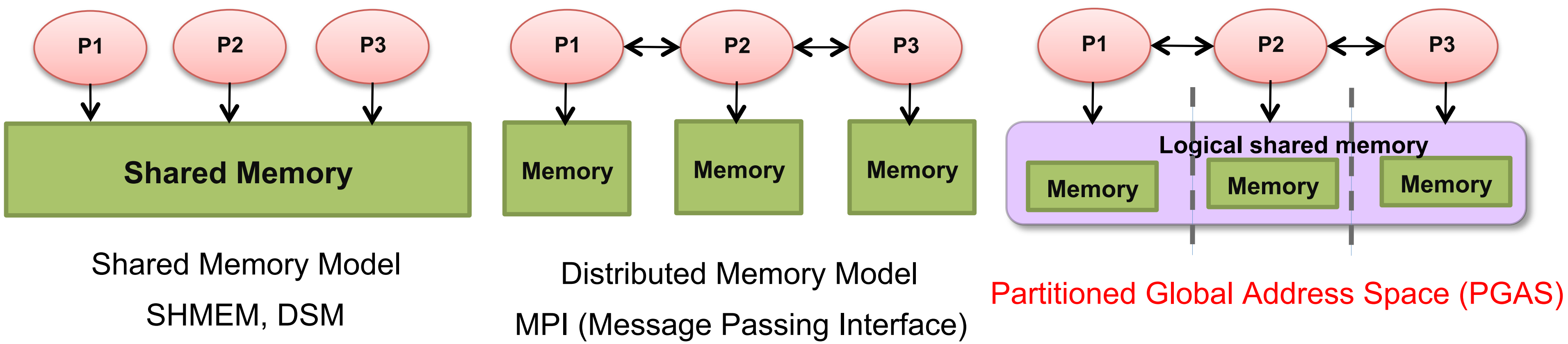
Outline

- Introduction
- Problem Statement
- Graph500 Benchmark
- Design Details
- Performance Evaluation
- Conclusion & Future Work

Introduction

- MPI - the de-facto programming model for scientific parallel applications
- Offers attractive features for High Performance Computing (HPC) applications
  - Non blocking, One sided, etc.
- MPI Libraries (such as MVAPICH2, OpenMPI, IntelMPI) have been optimized to the hilt
- Emerging Partitioned Global Address Space (PGAS) models -Unified Parallel C (UPC), OpenSHMEM

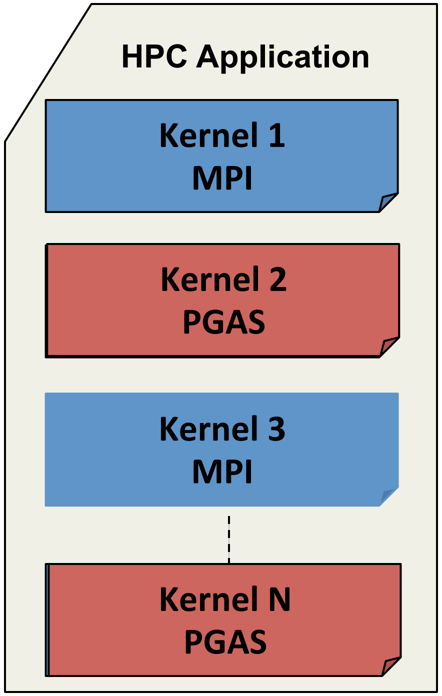
Partitioned Global Address Space (PGAS) Models



- PGAS Models
  - Shared memory abstraction over distributed systems
  - Global view of data, One sided operations, better programmability
  - Suited for irregular and dynamic applications
- OpenSHMEM, Unified Parallel C (UPC)
  - Popular PGAS models
- *Will applications be re-written entirely in PGAS model?*

Hybrid (MPI+PGAS) Programming for Exascale Systems

- Application sub-kernels can be re-written in MPI/PGAS based on communication characteristics
- Benefits:
  - Best of Distributed Computing Model
  - Best of Shared Memory Computing Model
- Exascale Roadmap*:
  - "Hybrid Programming is a practical way to program exascale systems"



* The International Exascale Software Roadmap, Dongarra, J., Beckman, P. et al., Volume 25, Number 1, 2011, International Journal of High Performance Computer Applications, ISSN 1094-3420

Introduction to Graph500

- Graph500 Benchmark
  - Represents data intensive and irregular applications that use graph algorithm-based processing methods
  - Bioinformatics and life sciences, social networking, data mining, and security/intelligence rely on graph algorithmic methods
  - Exhibits highly irregular and dynamic communication pattern
  - Earlier research have indicated scalability limitations of the MPI-based Graph500 implementations

Problem Statement

- *Can a high performance and scalable Graph500 benchmark be designed using MPI and PGAS models?*
- *How much performance gain can we expect?*
- *What will be the strong and weak scalability characteristics of such a design?*

Outline

- Introduction
- Problem Statement
- Graph500 Benchmark
- Design Details
- Performance Evaluation
- Conclusion & Future Work

Graph500 Benchmark – The Algorithm

- Breadth First Search (BFS) Traversal
- Uses 'Level Synchronized BFS Traversal Algorithm
  - Each process maintains – '*CurrQueue*' and '*NewQueue*'
  - Vertices in *CurrQueue* are traversed and newly discovered vertices are sent to their owner processes
  - Owner process receives edge information
    - if not visted; updates parent information and adds to *NewQueue*
  - Queues are swapped at end of each level
  - Initially the 'root' vertex is added to *currQueue*
  - Terminates when queues are empty
- Size of graph represented by SCALE and Edge Factor (EF)
  - #Vertices = 2**SCALE, #Edges = #Vertices * EF

MPI-based Graph500 Benchmark

- MPI_ISend/MPI_Test-MPI_IRecv for transferring vertices
- Implicit barrier using zero length message
- MPI-AllReduce to count number *newqueue* elements
- Major Bottlenecks:
  - Overhead in send-recv communication model
    - More CPU cycles consumed, despite using non-blocking operations
    - Most of the time spent in MPI-Test
  - Implicit Linear Barrier
    - Linear barrier causes significant overheads
- Other MPI Implementations
  - MPI-CSR, MPI-CSC, MPI-OneSided

Outline

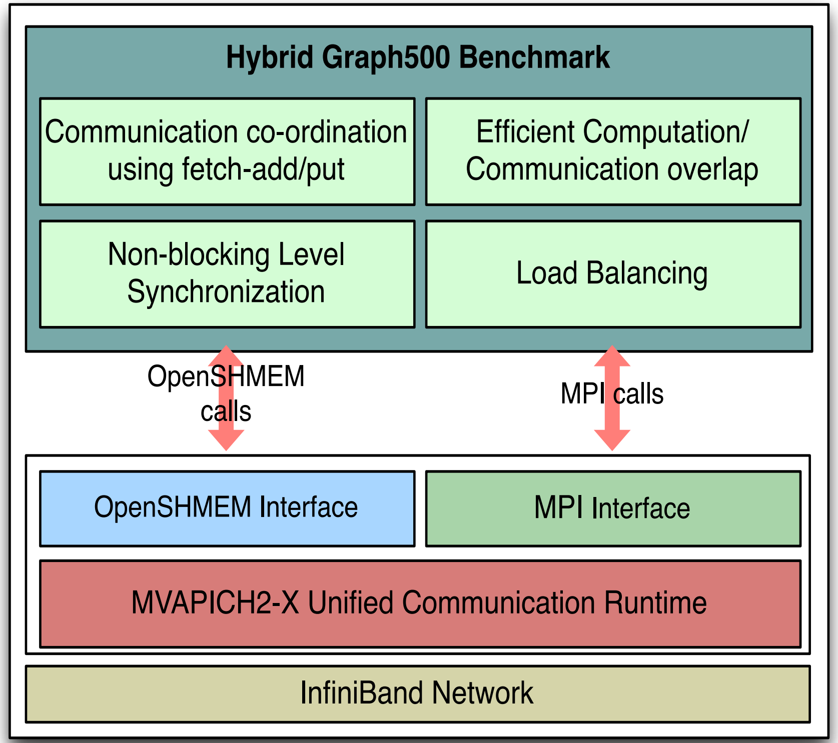
- Introduction
- Problem Statement
- Graph500 Benchmark
- Design Details
- Performance Evaluation
- Conclusion & Future Work

Design Challenges for Hybrid Graph500

- Co-ordination between sender and receiver processes and between multiple sender processes
  - How to synchronize, while using one-sided communication?
- Memory scalability
  - Size of receive buffer
- Synchronization at the end of each level
  - Barrier operations simply limit computation-communication overlap
- Load imbalance

Detailed Design

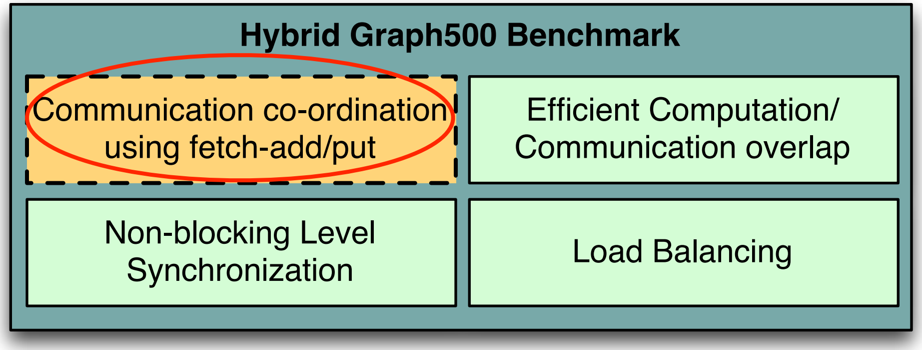
- Communication and co-ordination using one-sided routines and fetch-add atomic operations
- Buffer structure for efficient computation-communication overlap
- Level synchronization using non-blocking barrier
- Load Balancing



MVAPICH2/MVAPICH2-X Software

- High Performance open-source MPI Library for InfiniBand, 10Gig/iWARP and RDMA over Converged Enhanced Ethernet (RoCE)
  - MVAPICH (MPI-1) ,MVAPICH2 (MPI-3.0), Available since 2002
  - **MVAPICH2-X (MPI + PGAS), Available since 2012**
  - Used by more than 2,000 organizations (HPC Centers, Industry and Universities) in 70 countries
  - More than 173,000 downloads from OSU site directly
  - Empowering many TOP500 (Jun '13) clusters
    - 6th ranked 462,462-core cluster (Stampede) at TACC
    - 19th ranked 125,980-core cluster (Pleiades) at NASA
    - 21st ranked 73,278-core cluster (Tsubame 2.0) at Tokyo Institute of Technology
    - and many others
  - Available with software stacks of many IB, HSE and server vendors including Linux Distros (RedHat and SuSE)
  - http://mvapich.cse.ohio-state.edu

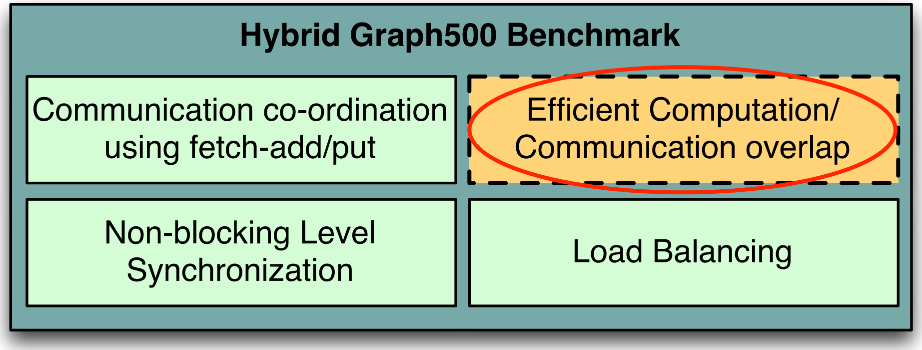
Communication and Co-ordination



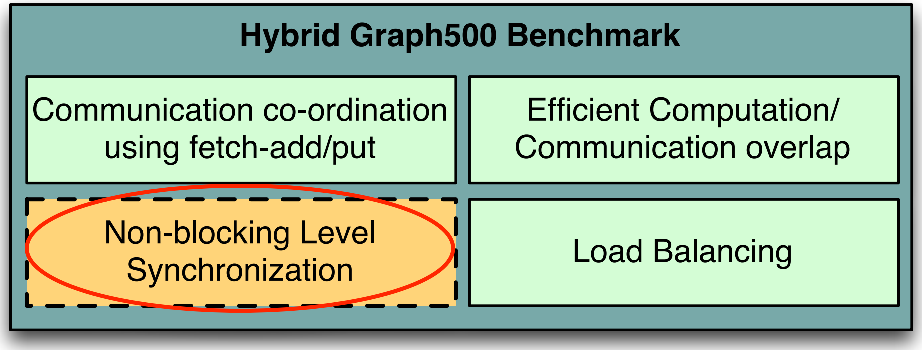
- Vertices transferred using OpenSHMEM shmem_put routine
- Receive buffers are globally shared
- Receive buffer size depends on number of local edges that the process owns and connectivity
  - Size is independent of system scale
- Atomic fetch-add operation for co-ordinating between sender and receiver, and between multiple senders
  - Receive buffer indices are globally shared

Buffer Structure for better Overlap

- Receiver process shall know if the data has arrived
- Buffer structure helps to identify incoming data
- Receive process ensures arrival of complete data
- packet by checking tail marker and can then process immediately



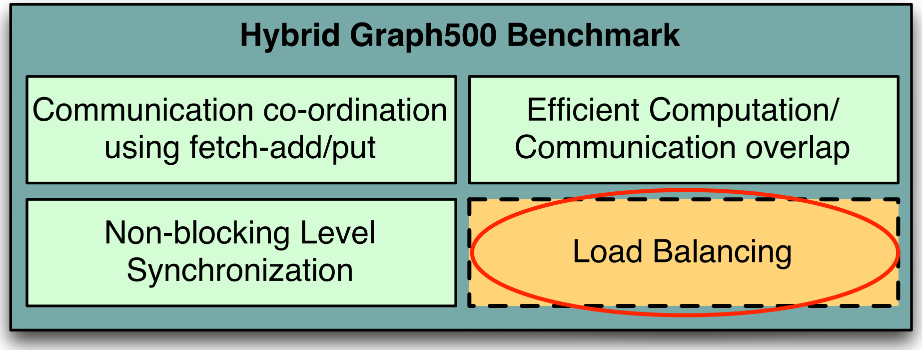
Level synchronization using non-blocking barrier



- MPI-3 non-blocking barrier for level synchronization
- Process enters the barrier and still can continue to receive and process incoming vertices
- Offers better computation/communication overlap

Intra-node Load Balancing

- Overloaded process exposes work
- Idle process takes up shared work and processes it, and puts back for post-processing
- Uses ‘shmem_ptr’ routine in OpenSHMEM to access shared memory data



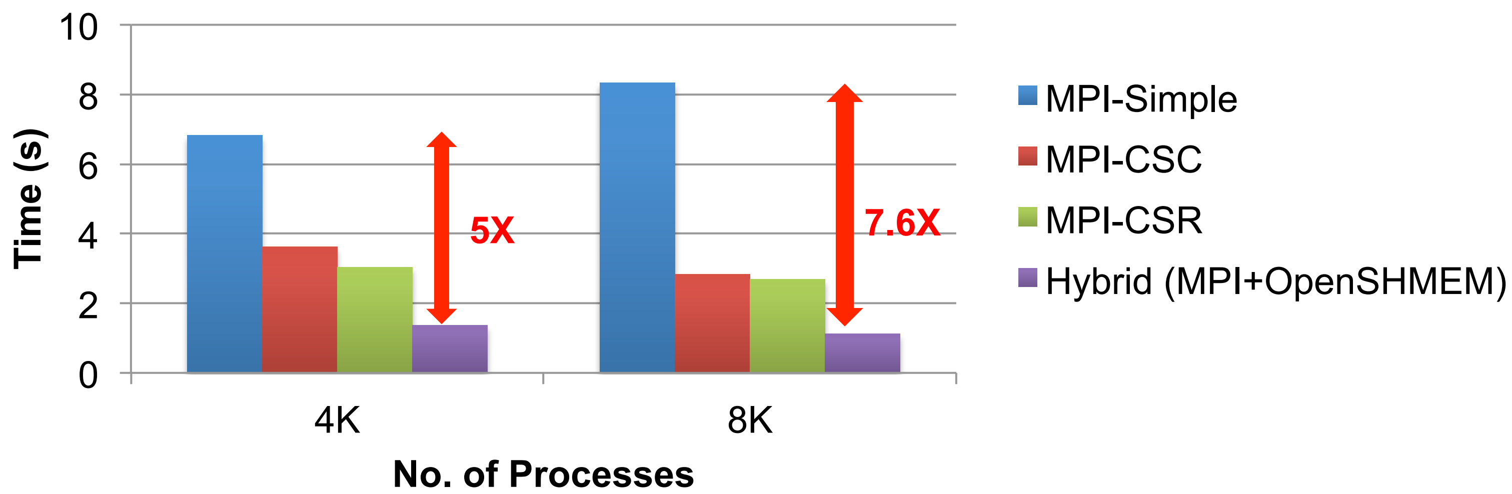
Outline

- Introduction
- Problem Statement
- Graph500 Benchmark
- Design Details
- Performance Evaluation
- Conclusion & Future Work

Experiment Setup

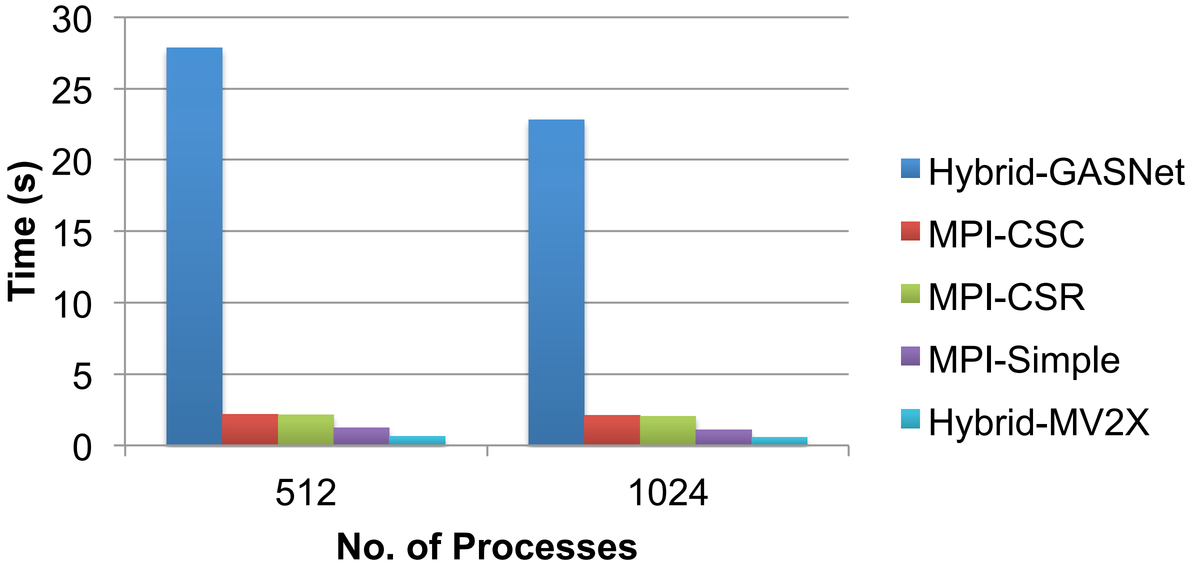
- Cluster A (TACC Stampede)
  - Intel Sandybridge series of processors using Xeon dual 8 core sockets (2.70GHz) with 32GB RAM
  - Each node is equipped with FDR ConnectX HCAs (54 Gbps data rate) with PCI-Ex Gen3 interfaces
- Cluster B
  - Xeon Dual quad-core processor (2.67GHz) with 12GB RAM
  - Each node is equipped with QDR ConnectX HCAs (32Gbps data rate) with PCI-Ex Gen2 interfaces
- Software Stacks
  - Graph500 v2.1.4
  - MVAPICH2-X OpenSHMEM (v1.9a2) and OpenSHMEM over GASNet (v1.20.0) and

Graph500 - BFS Traversal Time



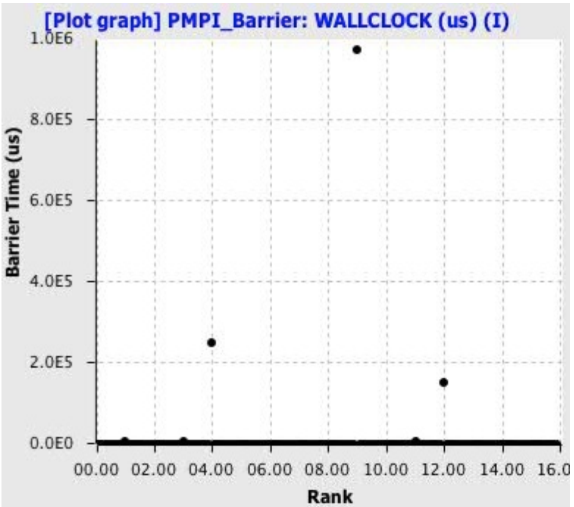
- Hybrid design performs better than MPI implementations
- 4,096 processes
  - **2.2X** improvement over MPI-CSR
  - **5X** improvement over MPI-Simple
- 8,192 processes
  - **7.6X** improvement over MPI-Simple (Same communication characteristics)
  - **2.4X** improvement over MPI-CSR

Unified Runtime vs. Separate Runtimes

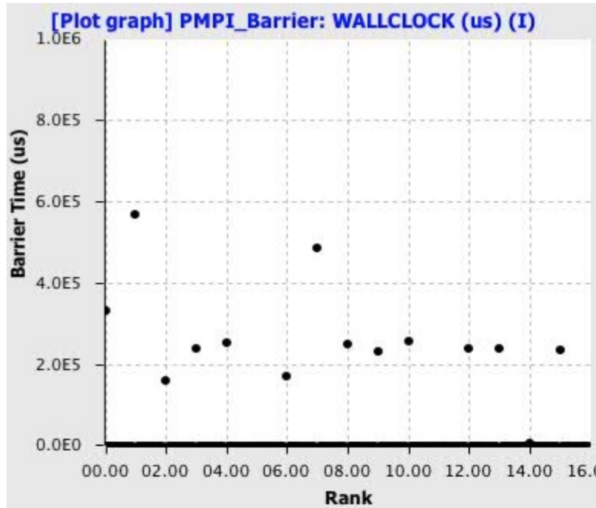


- Hybrid-GASNet uses separate runtimes for MPI and OpenSHMEM
  - Significant performance degradation due to lack of efficient atomic operations, and overhead due to separate runtimes
- For 1,024 processes
  - BFS time for Hybrid- GASNet: 22.8 sec
  - BFS time for Hybrid MV2X: 0.58 sec

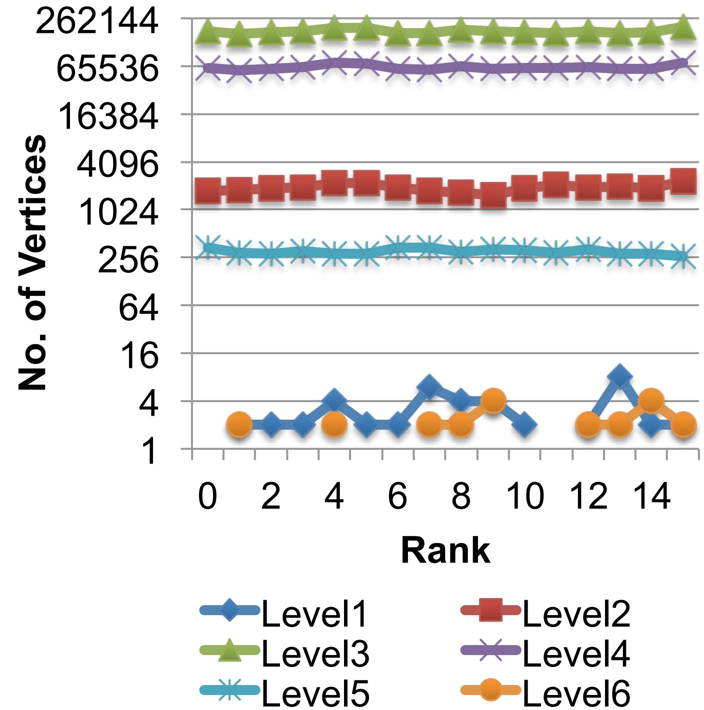
Load Balancing



Without Load Balancing



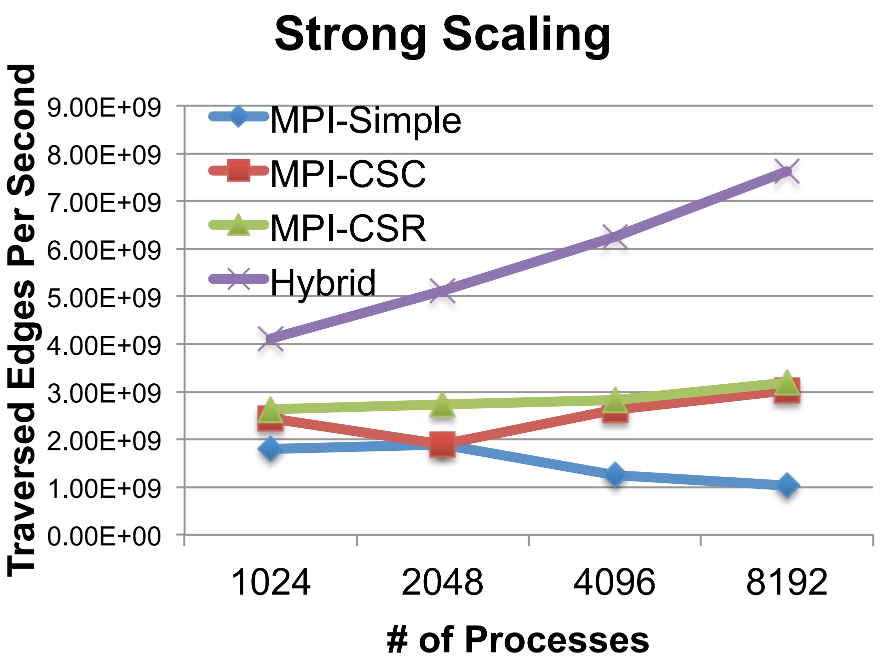
With Load Balancing

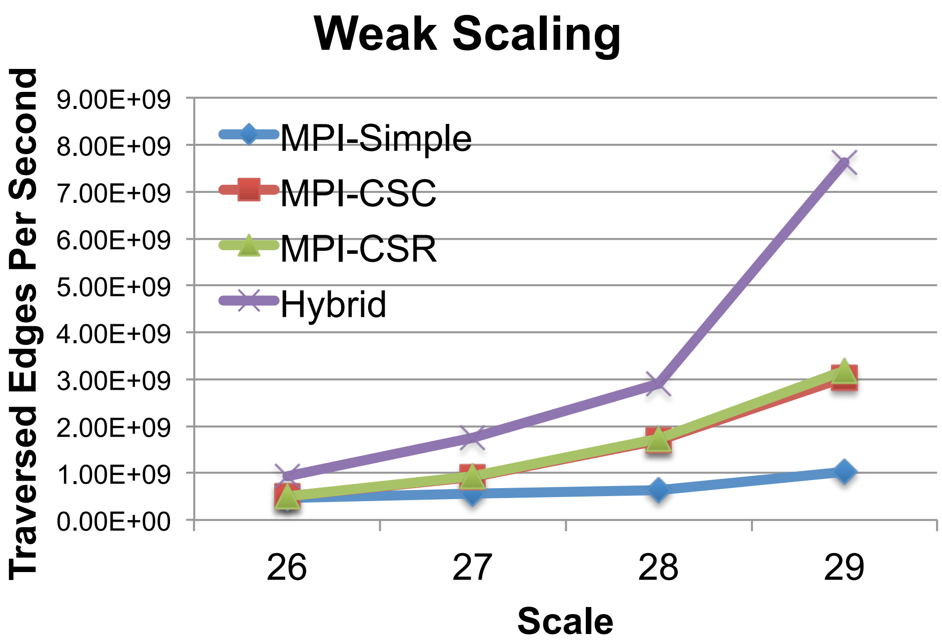


Vertex Distribution

- Evaluations using HPC Toolkit indicate that load is being balanced within node
- Load balancing limited within a node
  - Need for post processing
  - Higher cost for moving data
- Amount of work almost equal at each rank!

Scalability Analysis





- Strong Scaling

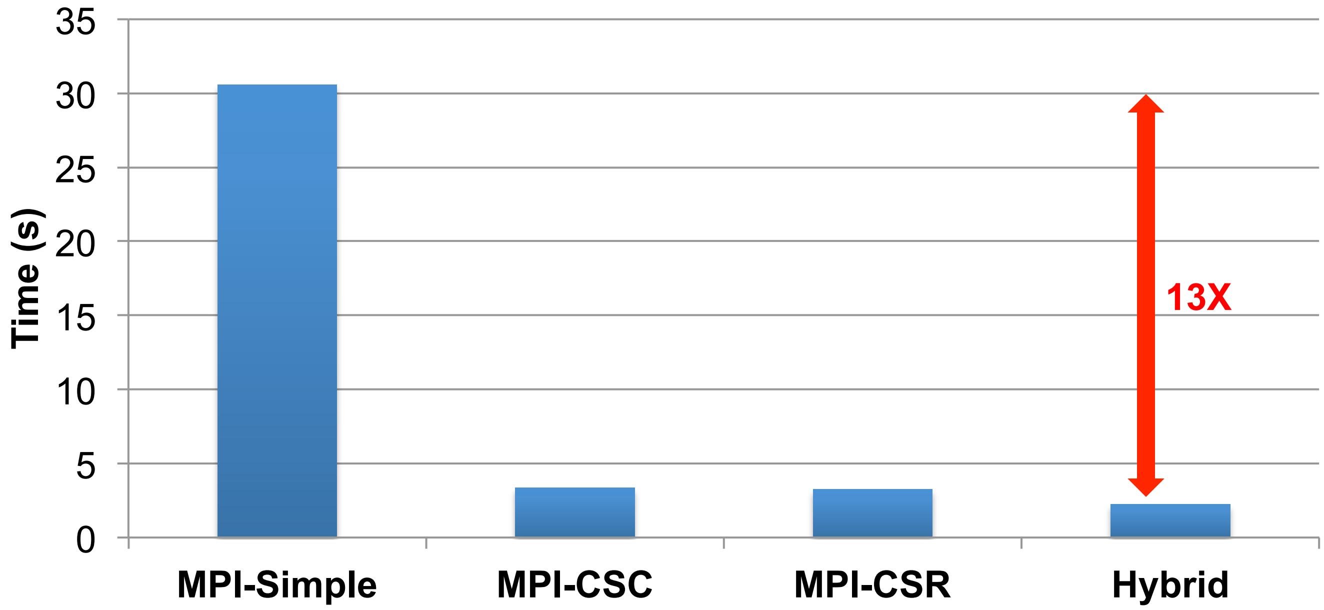
  Graph500 Problem Scale = 29

- Weak Scaling

  Graph500 Problem Scale = 26 per 1,024 processes

- Results indicate good scalability characteristics

Performance at 16K processes



- Graph Size - Scale = 29, EdgeFactor = 16
- Time for BFS Traversal
  - MPI Simple – **30.5s**
  - MPI CSR (best performing MPI version) – **3.25s**
  - Hybrid (MPI+OpenSHMEM) – **2.24s**
  - **13X** improvement over MPI Simple (same communication characteristics)

Outline

- Introduction
- Problem Statement
- Graph500 Benchmark
- Design Details
- Performance Evaluation
- Conclusion & Future Work

Conclusion & Future Work

- Presented a scalable design of Graph500 benchmark using hybrid MPI+OpenSHMEM
- Identified critical bottlenecks in the MPI-based implementation
- Not intended to compare programming models, but demonstrate the benefits of hybrid model
- Performance Highlights
  - At 16,384 cores, Hybrid design achieves **13 X** improvement over MPI-Simple and **2.4X** improvement over MPI-CSR
  - Exhitbits good scalability characteristics
  - Significant performance improvement over using separate runtimes
- Plan to improve load-balancing scheme, considering inter-node
- Plan to evaluate our design at larger scales and also consider real-world applications

Thank You!

{jose, potluri, panda}@cse.ohio-state.edu
{ktomko}@osc.edu

Network-Based Computing Laboratory

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

MVAPICH Web Page

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