# Performance Analysis of Lattice QCD on GPUs in APGAS Programming Model

Koichi Shirahata[1], Jun Doi[2], Mikio Takeuchi[2]

1: Tokyo Institute of Technology

2: IBM Research - Tokyo

# Programming Models for Exascale Computing

- GPU-based heterogeneous clusters
  - e.g.) TSUBAME 2.5 (3 GPUs x 1408 nodes)
  - Acceleration using GPUs
    - High peak performance/memory bandwidth

- Programming models for GPU-based clusters
  - Massage passing (e.g. MPI)
    - High tuning efficiency
    - High programming cost
  - APGAS (Asynchronous Partitioned Global Address Space)
    - Abstract distributed memory and deep memory hierarchy
    - X10: an instance of APGAS programming languages

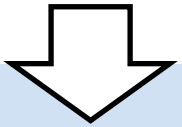
→ Highly scalable and productive computing on GPUs using APGAS

# Problem Statement

- How much do GPUs accelerate applications using APGAS?
  - Tradeoff between performance and productivity
    - Performance
      - The abstraction of memory hierarchy may limit performance
      - Scalability on multi-GPU
    - Productivity
      - Can we use GPUs with little programming cost?

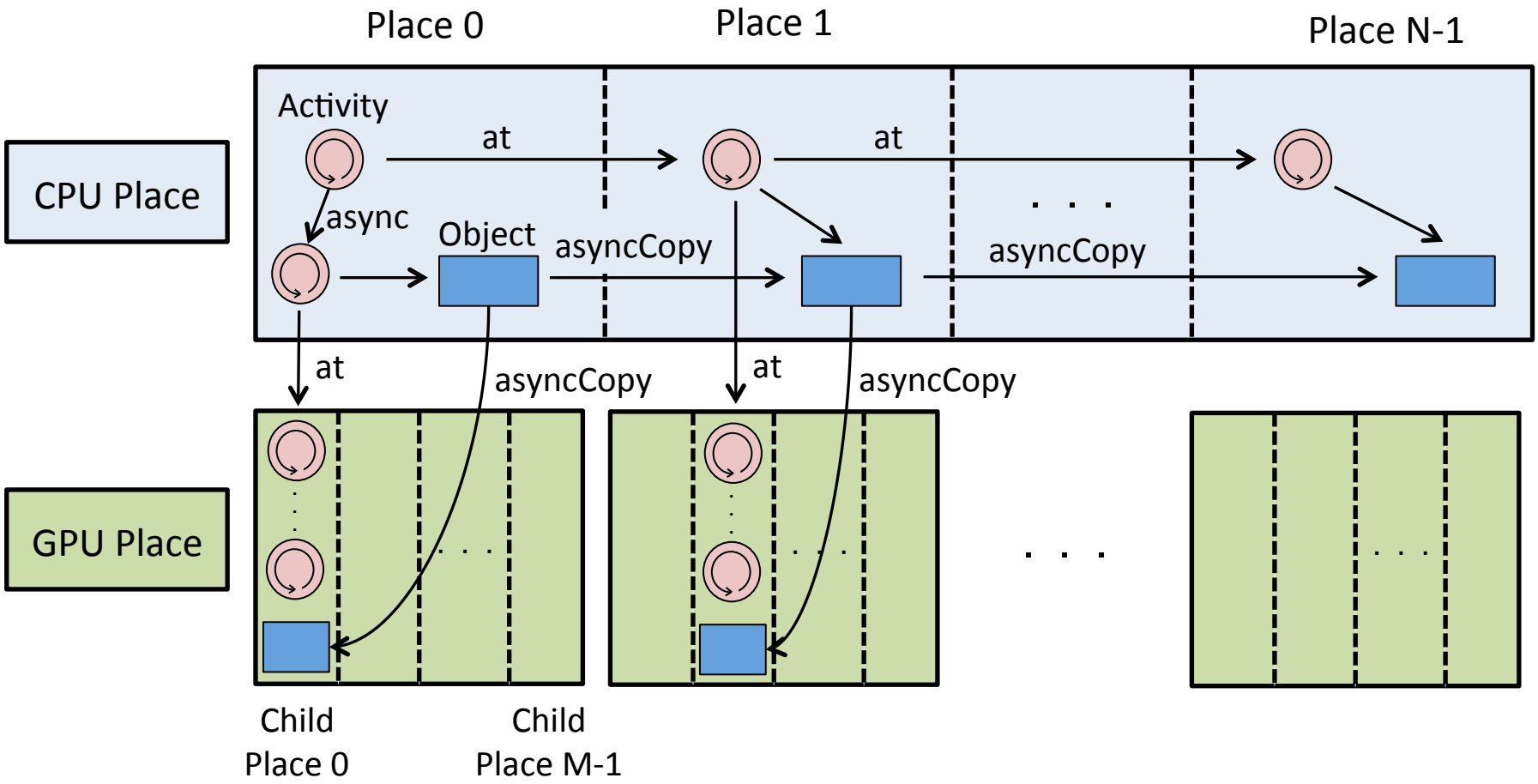
# Goal and Contributions

- Goal
  - Scalable and productive computing on GPUs
- Approach
  - Performance analysis of lattice QCD in X10 CUDA
    - Implement lattice QCD in X10 CUDA
    - Comparative performance analysis of X10 on GPUs



- Confirm acceleration using GPUs in X10
  - 19.4x speedup from X10 by using X10 CUDA on 32 nodes of TSUBAME 2.5

# The APGAS Model using GPUs

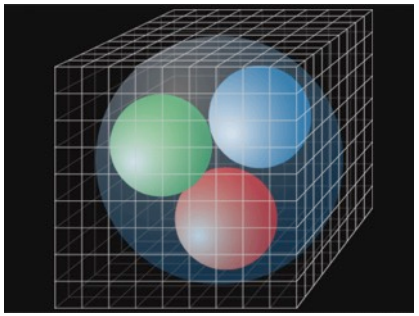


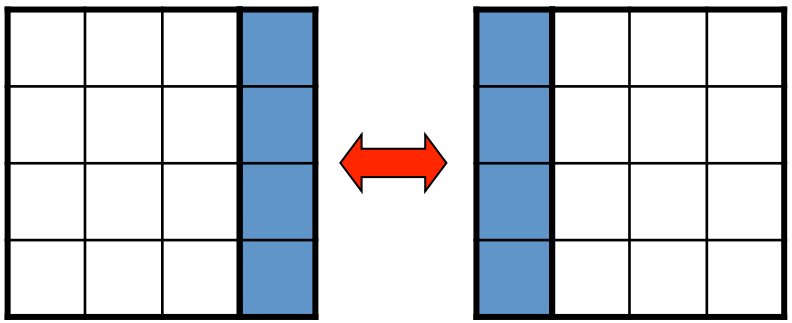

X10 provides CUDA support for GPUs [1]

[1] Cunningham, D. et al. "GPU programming in a high level language: compiling X10 to CUDA." Proceedings of the 2011 ACM SIGPLAN X10 Workshop. ACM, 2011.

# Lattice QCD

- Lattice QCD
  - Simulation of Quantum ChromoDynamics (QCD) of quarks and gluons on 4D grid of points in space and time
  - A grand challenge in high-performance computing
    - Requires high memory/network bandwidth and computational power
- Computing lattice QCD
  - Monte-Carlo simulations on 4D grid
  - Dominated solving linear equations of matrix-vector multiplication using iterative methods (etc. CG method)
  - Parallelizable by dividing 4D grid into partial grids for each place
    - Boundary exchanges are required between places in each direction



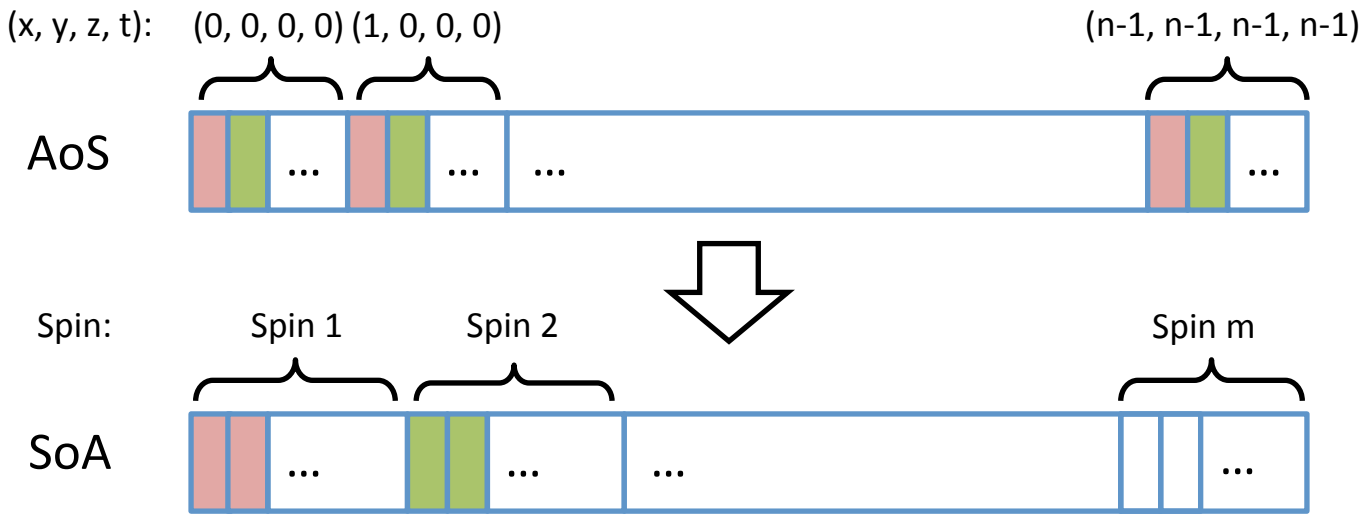


# Implementation of Lattice QCD in X10 CUDA

- We extend our lattice QCD in X10 into X10 CUDA
  - Porting whole solvers into CUDA kernels
    - Wilson-Dirac operator, BLAS level 1 operations
    - Avoid waste memory copy overheads between CPU and GPU, except boundary data transfers
  - Implement boundary data transfer among GPUs
    - Add memory copy operations
      - (1) GPU → CPU, (2) CPU → CPU, (3) CPU → GPU
  - Optimizations
    - Data layout transformation
    - Communication overlapping

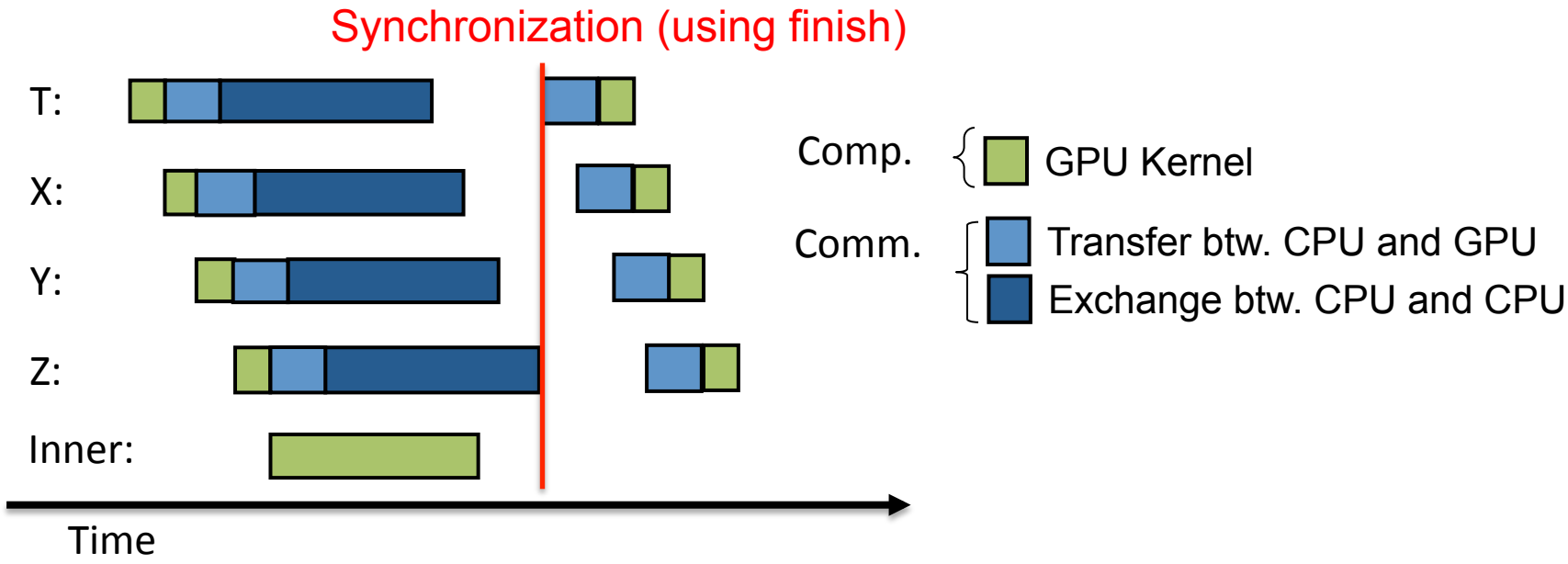
# Data Layout Transformation

- Two types of data layouts for quarks and gluons
  - AoS (Array of Structure)
    - Non-contiguous data
    - Used in our original CPU implementations
  - SoA (Structure of Array)
    - Contiguous data
    - Suitable for vectorization
- We translate from AoS to SoA
  - GPU is suitable for coalesced memory access



# Communication Optimizations in X10 CUDA

- Two communication optimizations
  - multi-dimensional partitioning
  - Communication overlapping
    - Overlap memory copy between GPU and CPU in addition to between CPU and CPU
    - Overlapping domain is limited by finish-based synchronization



# Experimental Setup

- Performance comparison with other implementations
  - X10 C++, MPI C, MPI CUDA
  - Use one GPU per node
- Measurements
  - Weak scaling
  - Strong scaling
- Configuration
  - Measure average iteration time of one convergence of CG method
    - Typically 300 – 500 iterations
  - Problem size
    - (x, y, z, t) = (24, 24, 24, 96) (unless specified)
    - Fit on one Tesla K20X GPU

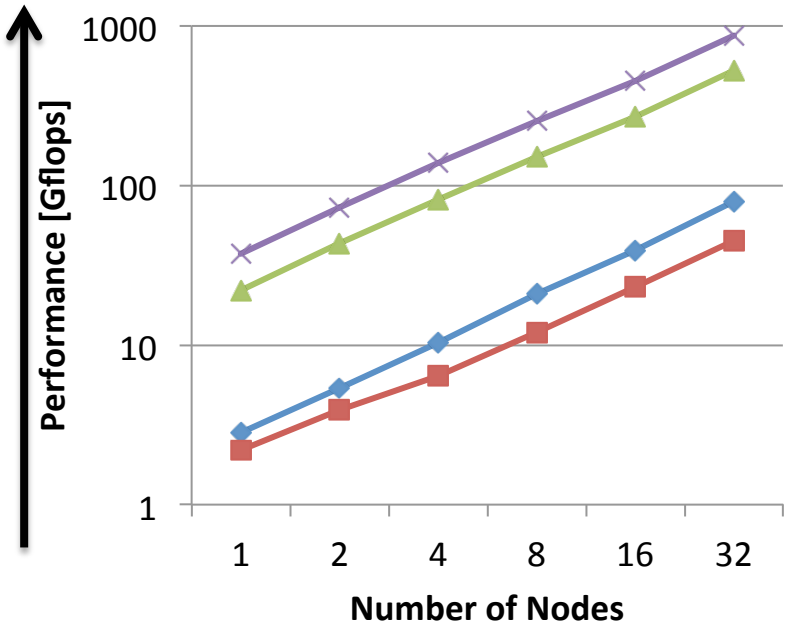
# Experimental environments

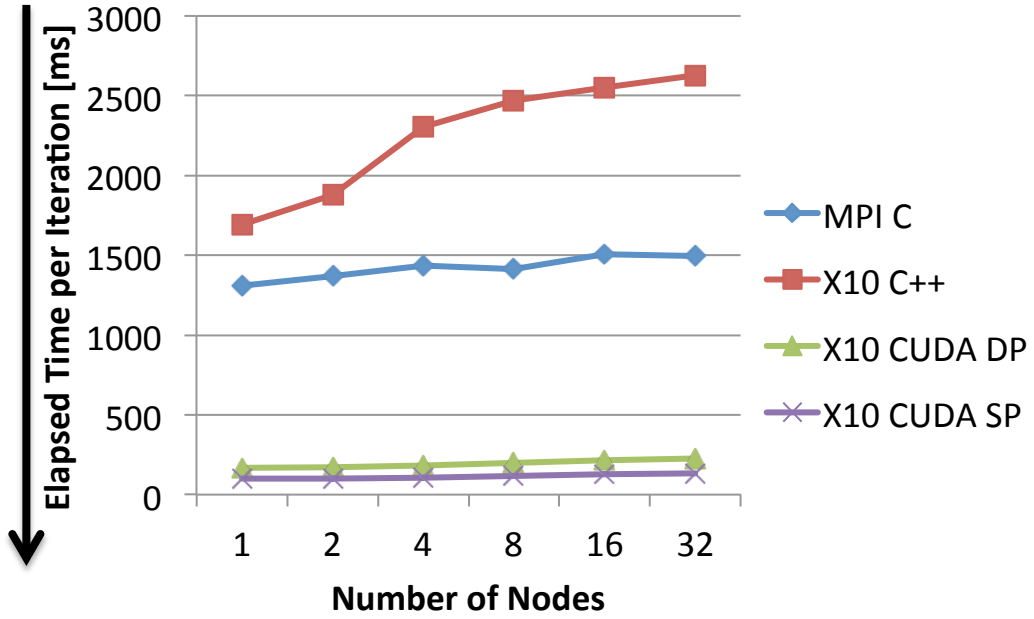
- TSUBAME2.5 supercomputer (unless specified)
  - Use up to 32 nodes (32 GPUs)
  - CPU-GPU: PCI-E 2.0 x16 (8 GB/sec)
  - Internode: QDR IB dual rail (10 GB/sec)
- Setup
  - 1 place per node
  - 1 GPU per place (X10 CUDA)
  - 12 threads per place (X10 C++, MPI C)
- Software
  - X10 version 2.4.3.2
  - CUDA version 6.0
  - OpenMPI 1.6.5

| | 2 CPUs / node | 3 GPUs / node |
|---|---|---|
| Model | Intel® Xeon® X5670 | Tesla K20X |
| # Cores | 6 | 2688 |
| Frequency | 2.93 GHz | 0.732 GHz |
| Memory | 54 GB | 6 GB |
| Memory BW | 32 GB/sec | 250 GB/sec |
| Compiler | gcc 4.3.4 | Nvcc 6.0 |

# Comparison of Weak Scaling with CPU-based Implementations

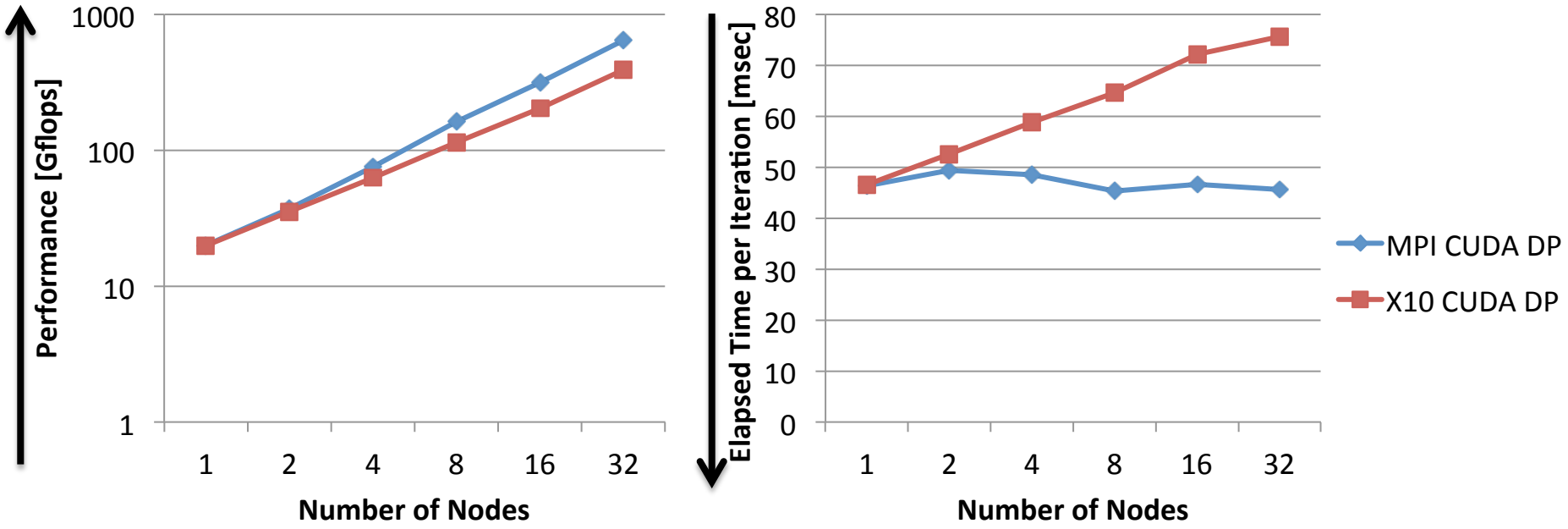
- X10 CUDA exhibits good weak scaling
  - 19.4x and 11.0x faster than X10 C++ and MPI C each on 32 nodes
  - X10 CUDA does not incur communicational penalty with large amount of data on GPUs



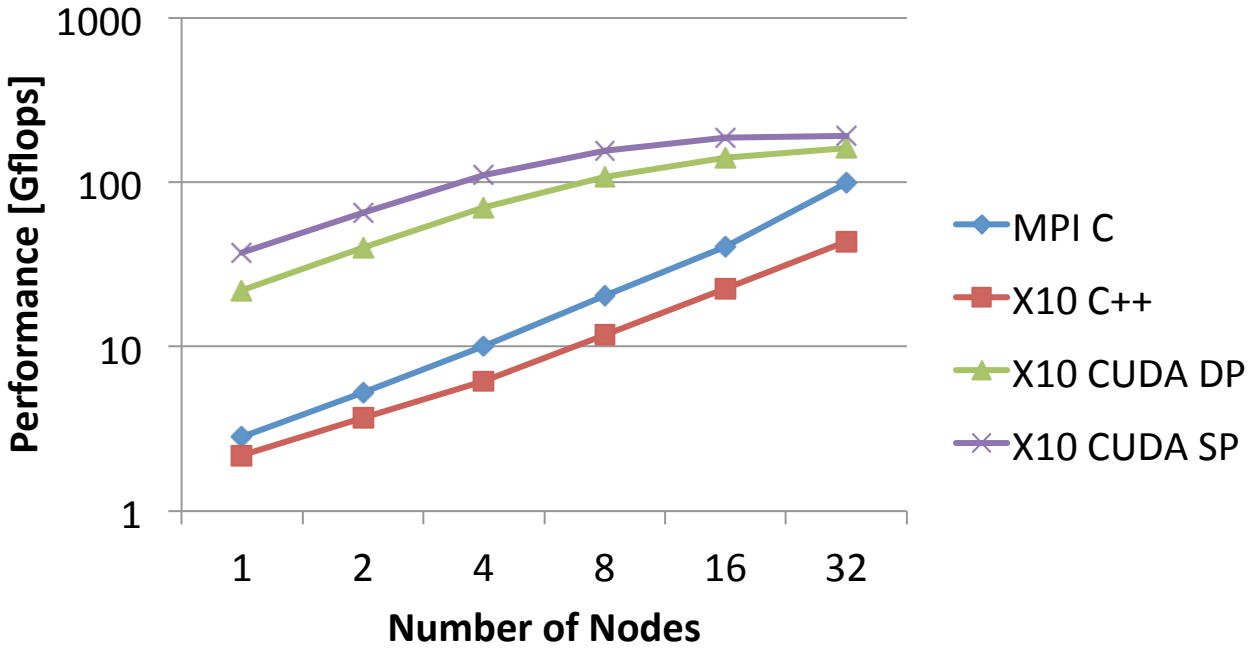
# Comparison of Weak Scaling with MPI CUDA

- Comparison on TSUBAME-KFC
  - X10 2.5.1, CUDA 5.5, OpenMPI 1.7.2
  - Problem Size: 24 x 24 x 24 per node
- X10 CUDA performs similar scalability with MPI CUDA
- Increase performance gap as using larger number of nodes



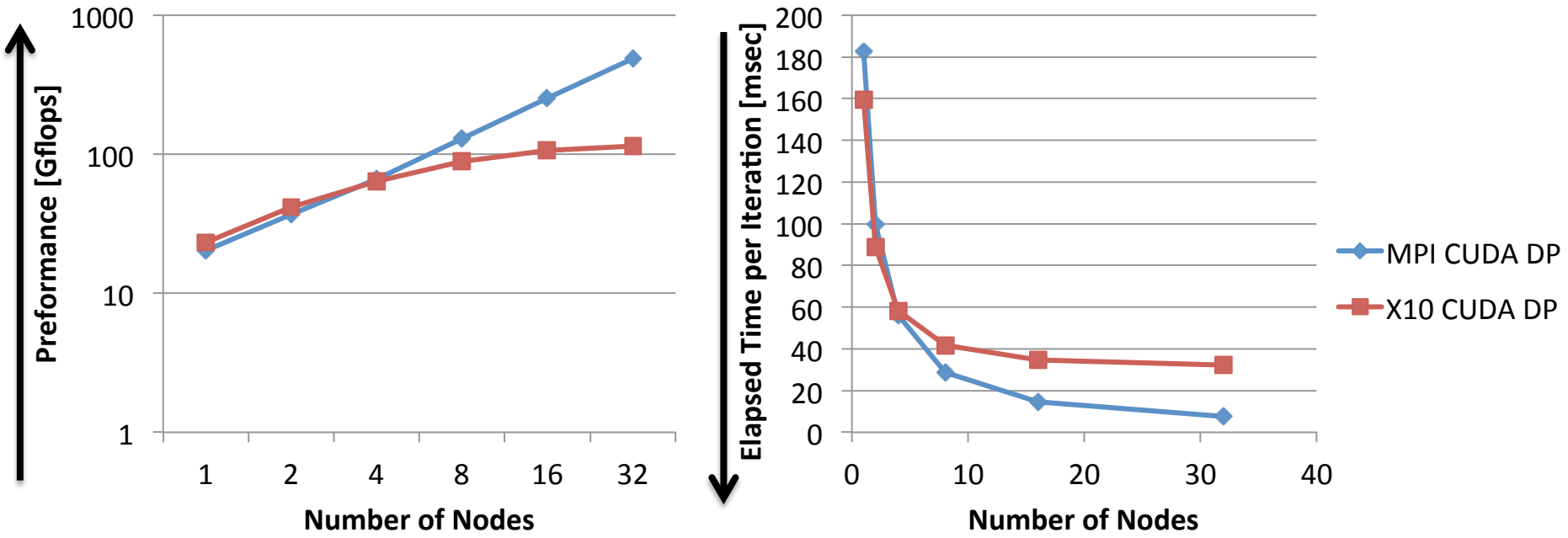
# Strong Scaling Comparing with CPU-based Implementations

- X10 CUDA outperforms both X10 C++ and MPI C
  - 8.27x and 4.57x faster each on 16 Places
  - Scalability of X10 CUDA gets poorer as the number of nodes increases



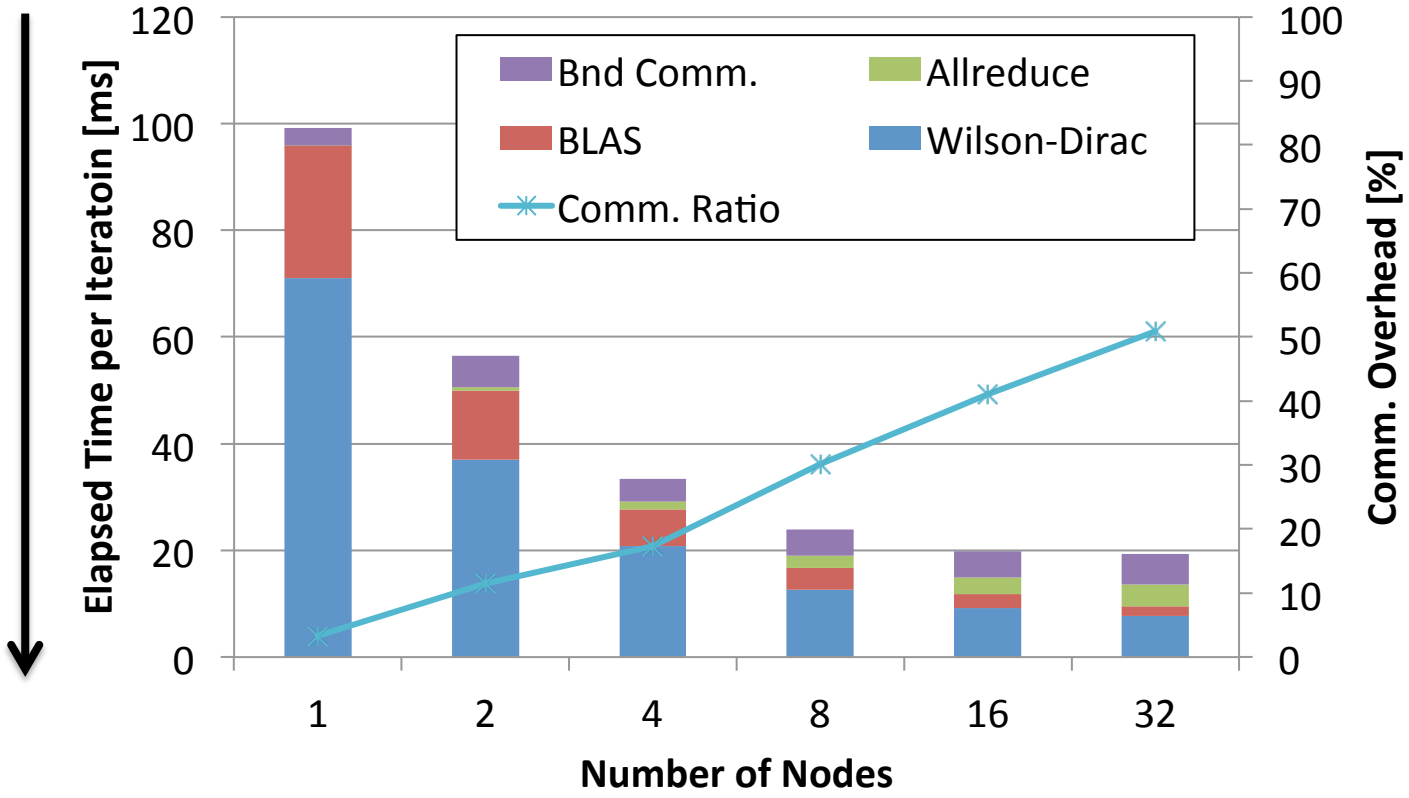
# Comparison of Strong Scaling with MPI CUDA

- Comparison on TSUBAME-KFC
  - X10 2.5.1, CUDA 5.5, OpenMPI 1.7.2
- X10 CUDA exhibits comparative performance up to 4 nodes
- X10 CUDA suffers heavy overheads on over 8 nodes



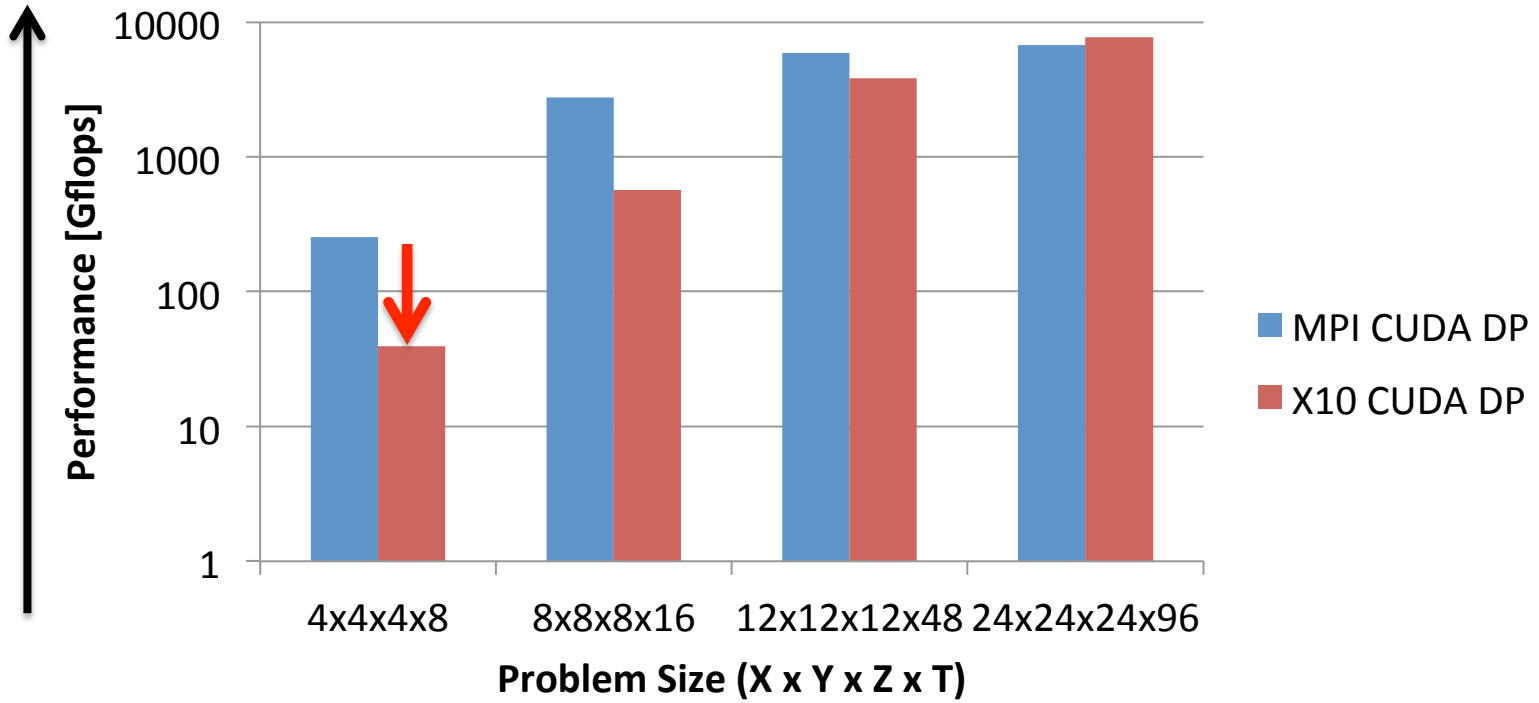
# Performance Breakdown of Lattice QCD in X10 CUDA

- Communication overhead increases
  - Both boundary communication and MPI Allreduce
- Computation parts also do not scale linearly



# Comparison with MPI CUDA using Different Problem Sizes

- Comparison on TSUBAME-KFC
  - X10 2.5.1, CUDA 5.5
- X10 CUDA suffers heavy overhead on small problem sizes
  - 6.02x slower on 4 x 4 x 4 x 8
  - We consider X10 CUDA suffers constant overhead



# Comparison of Productivity with MPI CUDA

- Lines of code
  - X10 CUDA version contains 1.92x larger lines of code compared with MPI CUDA in total
    - Since currently X10 CUDA cannot call device functions inside CUDA kernels

| | MPI CUDA | X10 CUDA |
|---|---|---|
| Total | 4667 | 8942 |
| Wilson Dirac | 1590 | 6512 |

- Compiling time
  - X10 CUDA takes 11.3x longer time to compile

| | MPI CUDA | X10 CUDA |
|---|---|---|
| Compiling Time [sec] | 15.19 | 171.50 |

# Pros/Cons of X10 CUDA from Our Study

- Advantages of X10 CUDA
  - Straightforward porting from X10 to X10 CUDA
    - Simply porting computation kernels into CUDA
    - inserting memory copy operations between CPU and GPU
  - X10 CUDA exhibits good weak scaling
- Drawbacks of current version of X10 CUDA
  - Limitations of programmability
    - X10 CUDA cannot call a function inside a kernel
  - Limitations of performance
    - finish-based synchronization incurs overhead
    - X10 CUDA does not support creating CUDA streams

# Related Work

- High performance large-scale lattice QCD computation
  - Peta-scale lattice QCD on a Blue Gene/Q supercomputer [Doi et al. 2012]
    - Fully overlapping communication and applying node-mapping optimization for BG/Q
- Lattice QCD using many-core accelerators
  - QUDA: A QCD library on GPUs [Clark et al. 2010]
    - Invokes multiple CUDA streams for overlapping
  - Lattice QCD on Intel Xeon Phi [Joo et al. 2013]
- PGAS language extension for multi-node GPU clusters
  - XcalableMP extension for GPU [Lee et al. 2012]
    - Demonstrated their N-body implementation scales well

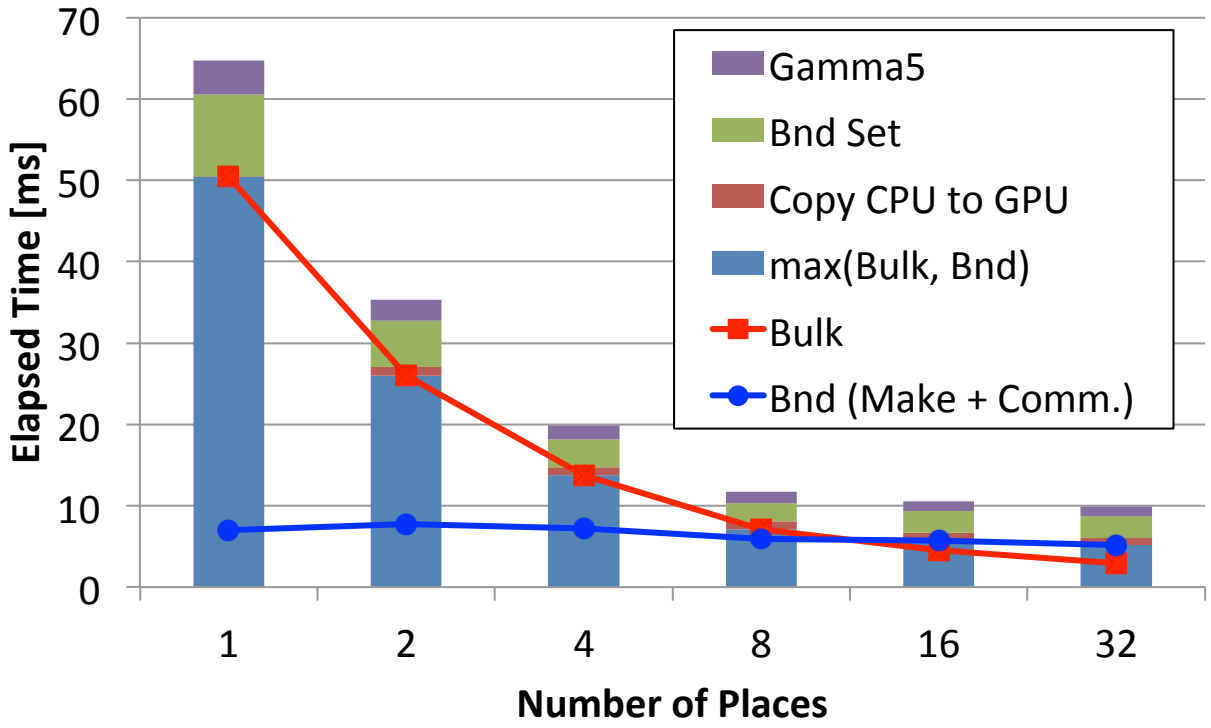
# Conclusion

- Conclusion
  - GPUs accelerate lattice QCD significantly in APGAS programming model
    - X10 CUDA exhibits good scalability in weak scaling
    - 19.4x speedup from X10 by using X10 CUDA on 32 nodes of TSUBAME 2.5
  - We reveal limitations in current X10 CUDA
    - Performance overheads in strong scalability
    - Increase of lines of code
- Future work
  - Performance improvement in strong scaling
    - More detailed analysis of overheads in X10 CUDA

- Backup

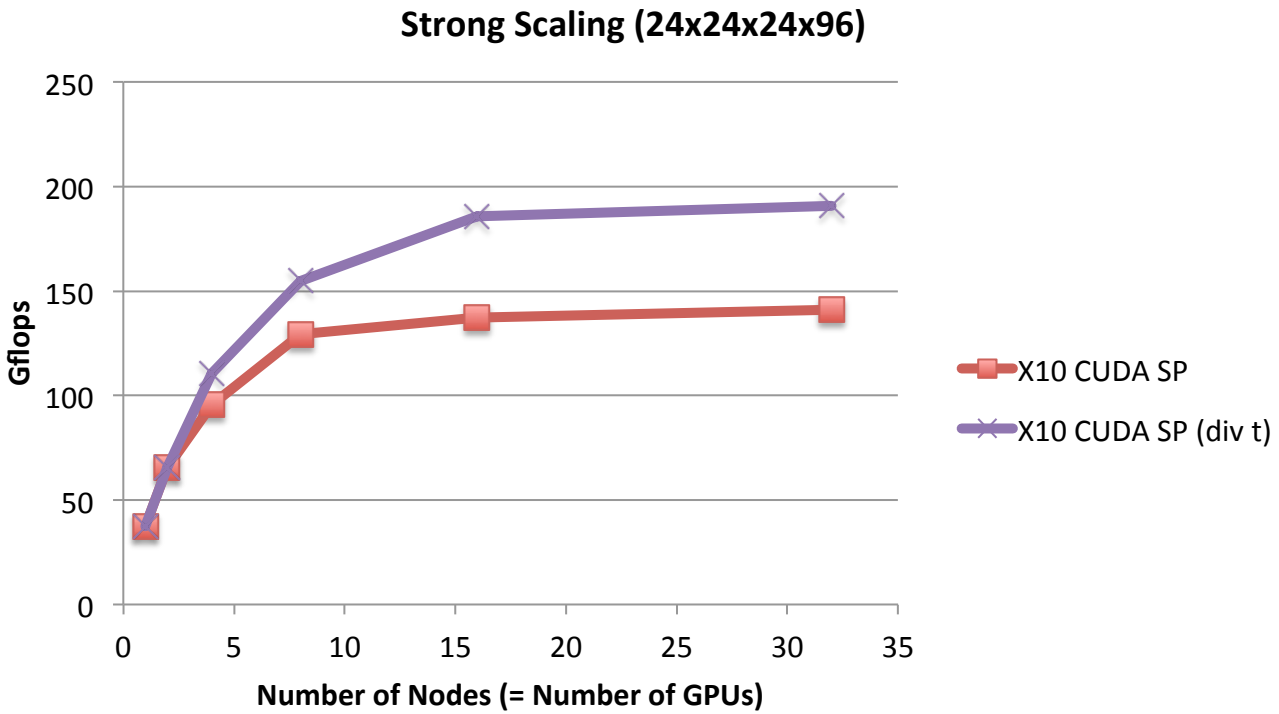
# Breakdown of Wilson-Dirac Computation in X10 CUDA

- Communication becomes dominant when using more than 16 places
  - A cause of the limit of strong scaling
- Possible ways to improve the scalability
  - Applying one-to-one synchronization
  - Improving communication and synchronization operations themselves in X10



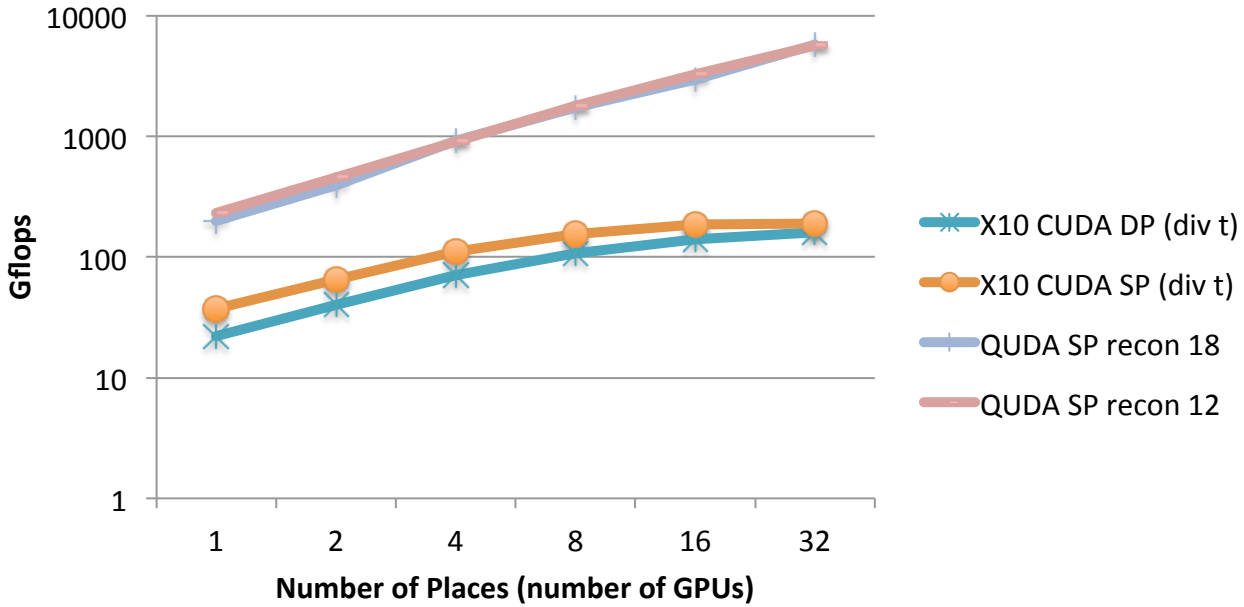
# Comparison with Different Dimensions of Lattice Partitioning

- Comparison between 4D and 1D partitioning
  - 1D partitioning exhibits better strong scalability
  - 1.35x better on 32 GPUs
  - Still saturate using over 16 GPUs



# Comparison with QUDA

- QUDA [1]: A QCD library on GPUs
  - Highly optimized for GPUs
- QUDA exhibits better strong scalability
  - Currently 30.4x slower in X10 CUDA




[1]: Clark, Michael A., et al. "Solving Lattice QCD systems of equations using mixed precision solvers on GPUs." Computer Physics Communications 181.9 (2010): 1517-1528.