May 11, 2015

Hierarchical DAG scheduling for Hybrid Distributed Systems

Wei Wu, Mathieu Faverge, Aurelien Bouteiller, George Bosilca

ICL UT
INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Outline

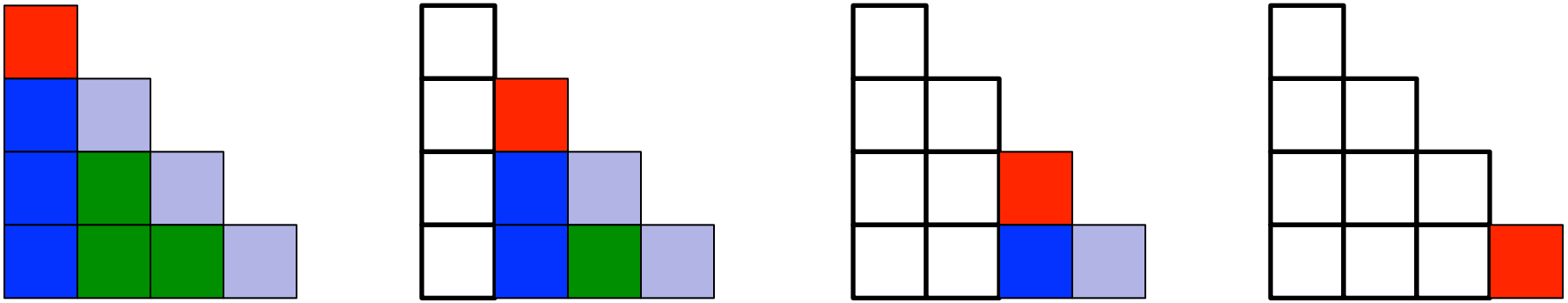
- Introduction & Motivation
- Hierarchical DAG Implementation
- Hierarchical DAG Support in PaRSEC
  - Multiple CUDA Streams
  - CPU/GPU Load Balance
  - Data Coherence
- Performance Evaluation
  - Tile Size Tuning
  - Shared Memory
  - Distributed Memory
- Conclusions

Introduction

- Different optimal task granularity on GPU and CPU
- “Hierarchical DAG”
  - Adapt the granularity of tasks dynamically according to execution location.
- Linear algebra application (matrix factorization).
  - Tiled algorithm. Tile size is a critical tuning parameter
  - “Hierarchical DAG” allows different tile sizes coexist during execution

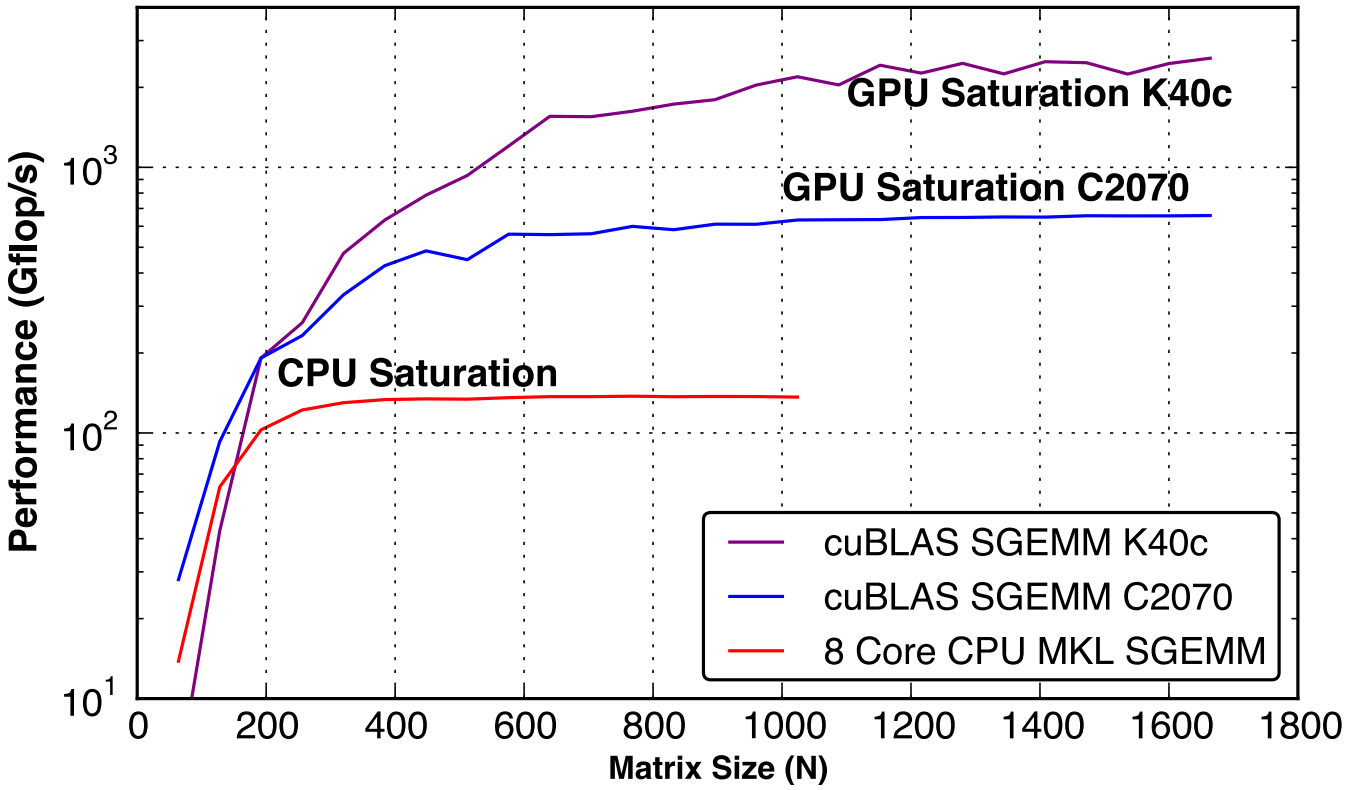
Background Matrix Factorization

- Diagonal matrix: Cholesky
- Square matrix: QR



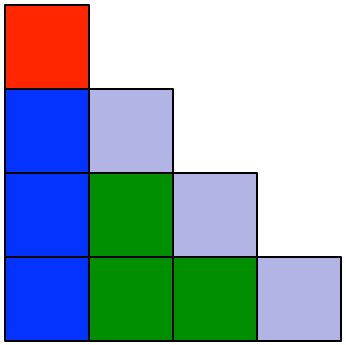
Example of Cholesky factorization on 4 × 4 matrix

Motivation



Impact of the problem granularity on the performance of the CPU and GPU SGEMM compute kernel when applied to a single tile of size N.

Motivation (cont'd)

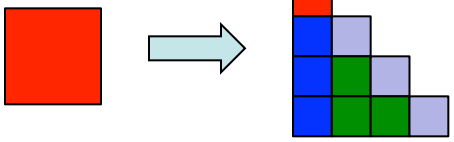


Traditional

Unique tile size for each task
- Big tile size: not enough parallelism
- Small tile size: less GPU occupancy

Hierarchical

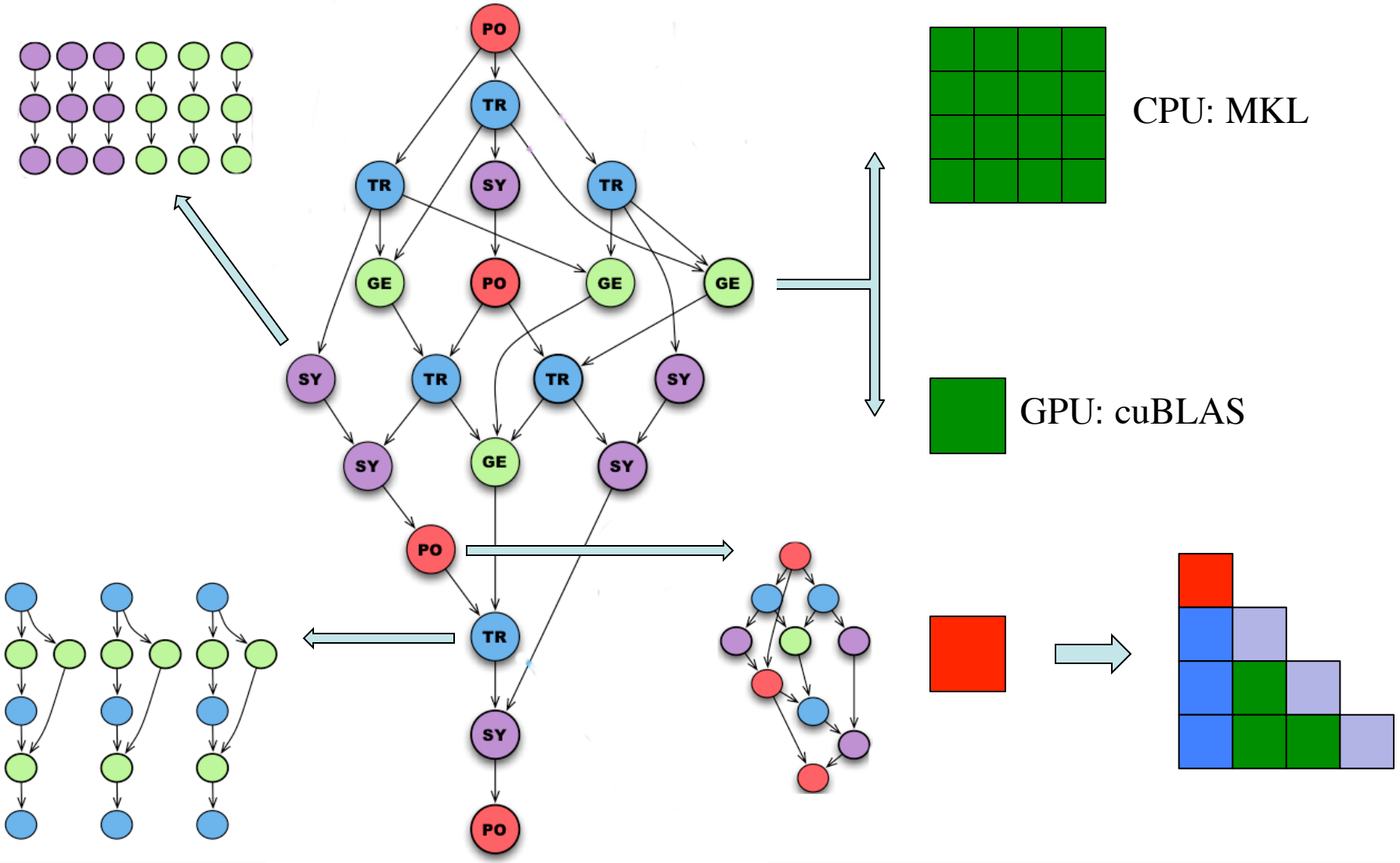
Big tile size on GPU
for better GPU occupancy



Small tile size on CPU
to expose more parallelism

POTRF TRSM SYRK GEMM

Hierarchical DAG Implementation

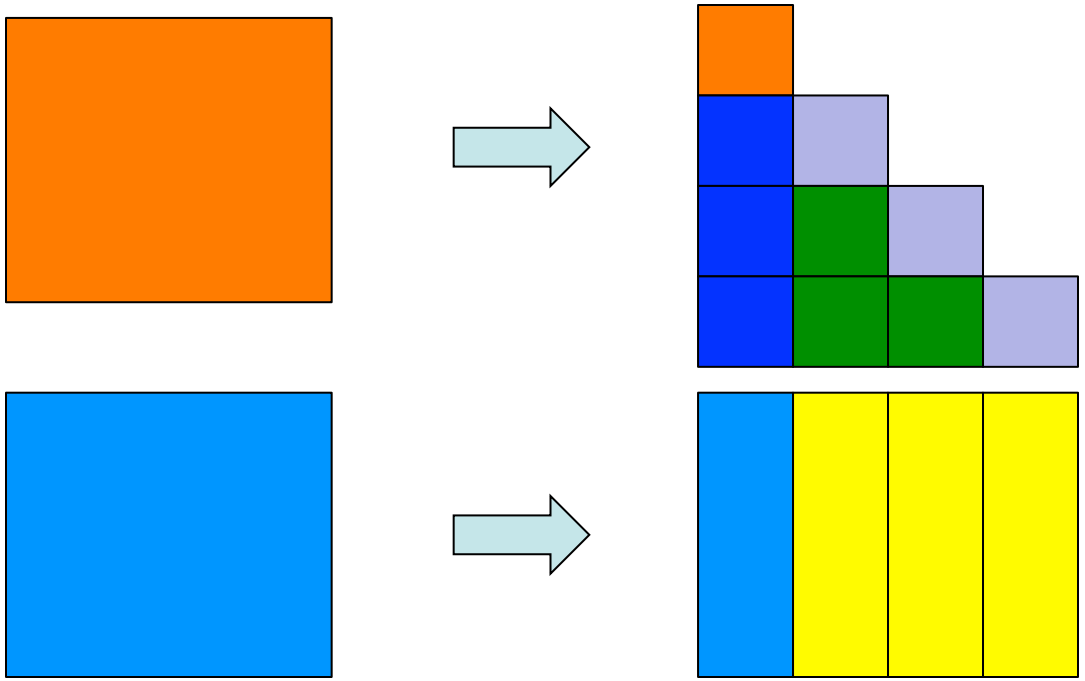


Hierarchical DAG Implementation (cont'd)

- Coarse grain DAG
  - Creation operation is collective across the entire distributed memory domain.
  - Unique global identifier
- Fine grain DAG
  - Creation operation is dynamic, depending on data locality and work load of devices.
  - No global identifier, visible only to local node

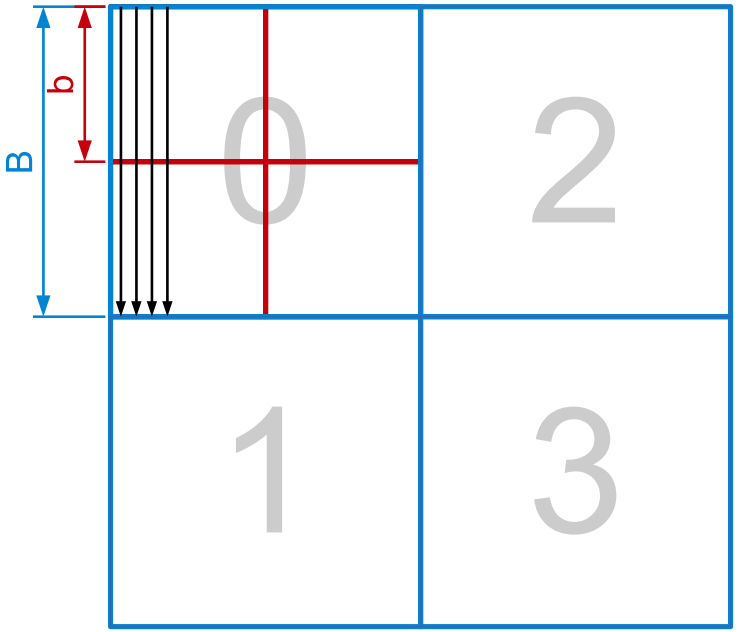
Hierarchical DAG LA Applications

- Generic solution can be applied to any algorithm that tasks can be split.
  - Cholesky factorization: fine grain tasks operating on small square tiles.
  - QR factorization: fine grain tasks operating on small rectangle tiles.



Hybrid Data Layout

- Different data layout: tile and LAPACK
- For sub-tiles in the fine grain DAG (red), the data layout is the same as the LAPACK layout with interleaved data, while tile layout (blue) is used for large tiles and permits a much more efficient data transfer to/from the accelerators



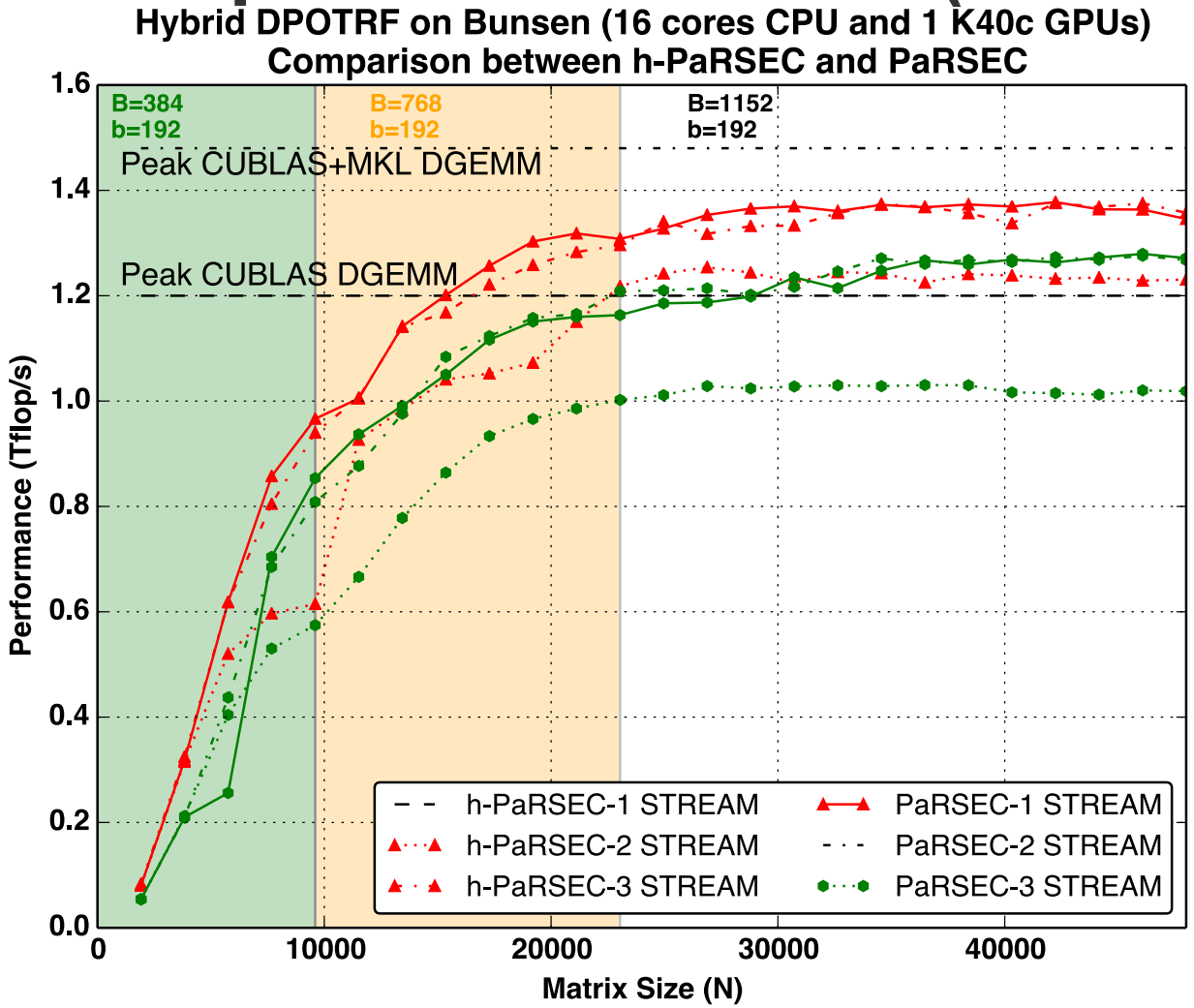
Hierarchical DAG Support in PaRSEC

- Tasks can be executed on either CPU or GPUs if both implementations are provided.
- Tasks will be split dynamically to provide more parallelism if current parallelism is not sufficient.
- Multiple CUDA streams for better GPU occupancy.
- Multi-level GPU task queues for dynamic task execution order.
- Dynamic CPU/GPU load balance
- CPU/GPU data coherence protocol for minimal data movement.

Multiple CUDA Streams

- Define GPU task as three stages:
  - Move data into GPU
  - Kernel execution
  - Move data out of GPU
- Data move in/out
  - Each direction takes one CUDA stream.
  - Data prefetch
- Kernel execution: multiple CUDA streams depending on GPU generations

Multiple CUDA Stream (cont'd)



Performance difference between hierarchical DAG and the standard version on POTRF with a varying number of CUDA streams

Multi-level GPU Task Queues

- Each stage of GPU task contain a queue.
- Out of order task execution
  - Task priority: promote the execution of tasks which have higher priority
  - Data availability: promote the execution of tasks for which a copy of the data is already available
- CUDA events track the completion of each stage.

CPU/GPU Dynamic Load Balance

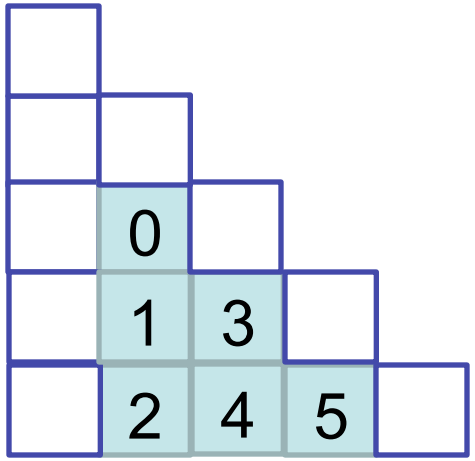
- Task weight: the inverse of the peak performance of the kernel on the device
- Coarse grain DAG
  - Workload: task will be assigned to the device which would have the lowest workload if this task runs on it.
  - Locality: if a task whose R&W data is already on a device, then later tasks which use the same R&W data will stay on this device. This does not apply to CPU.
- Fine grain DAG
  - Job stealing

CPU/GPU Load Balance (cont'd)

K40 1/1200 | 0 | 1 | 2 | 4 | 5 | 1/1200

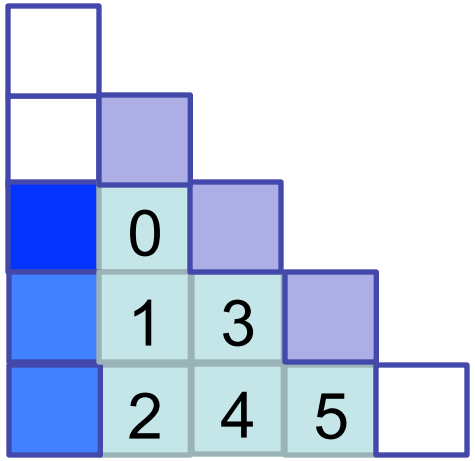
C2070 1/300 | 3 | 1/300

CPU 1/19

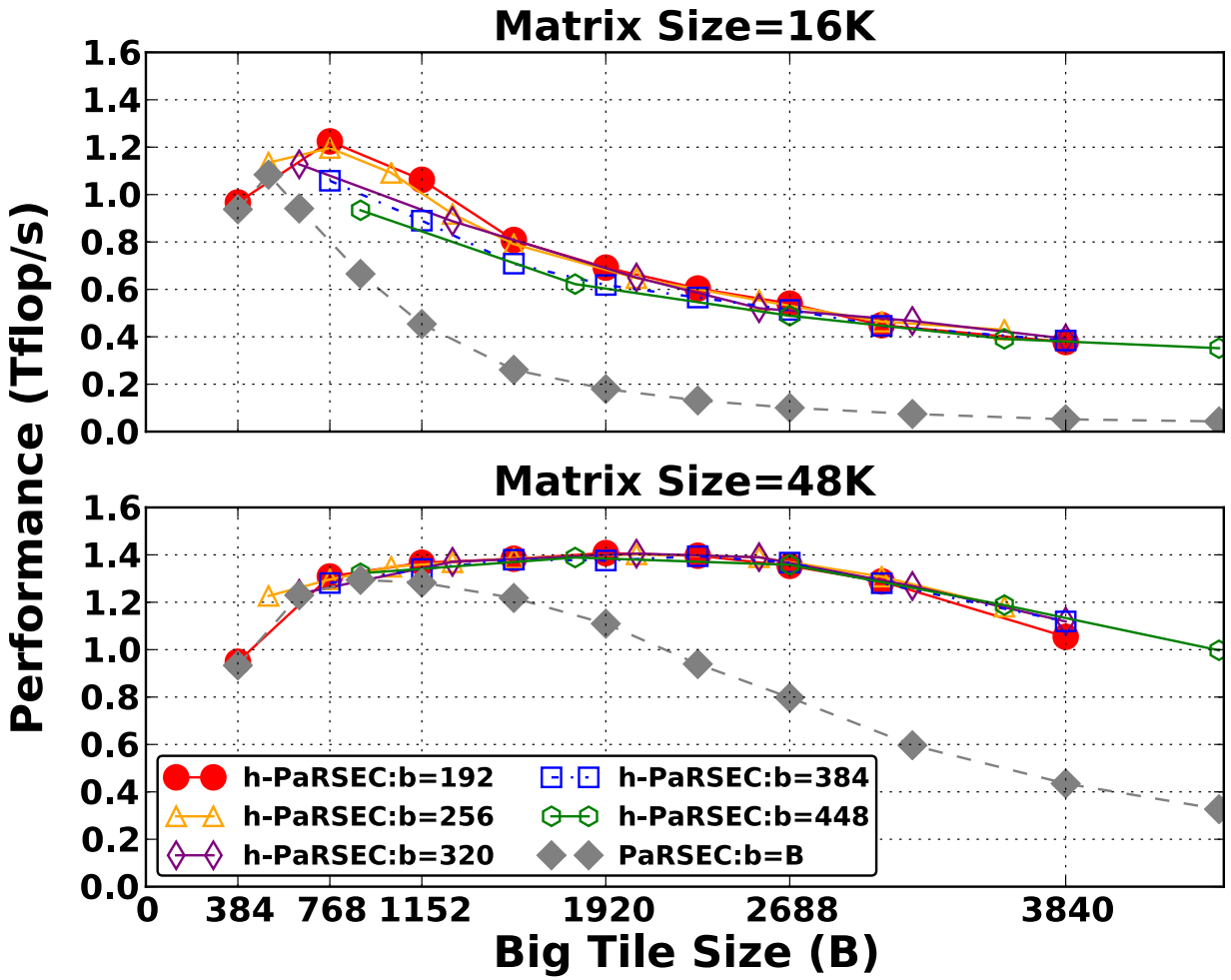


Data Coherence between CPU & GPU

- Modified MOESI protocol
  - Read Only, Write Only, Read Write
  - Owner of data
  - Number of readers
  - Location of data on each device
- Out of Core GPU Operations
  - Write LRU and Read Only LRU
  - Clean up the Write LRU when GPU dedicated thread is idle.
  - Improve Load Balance.

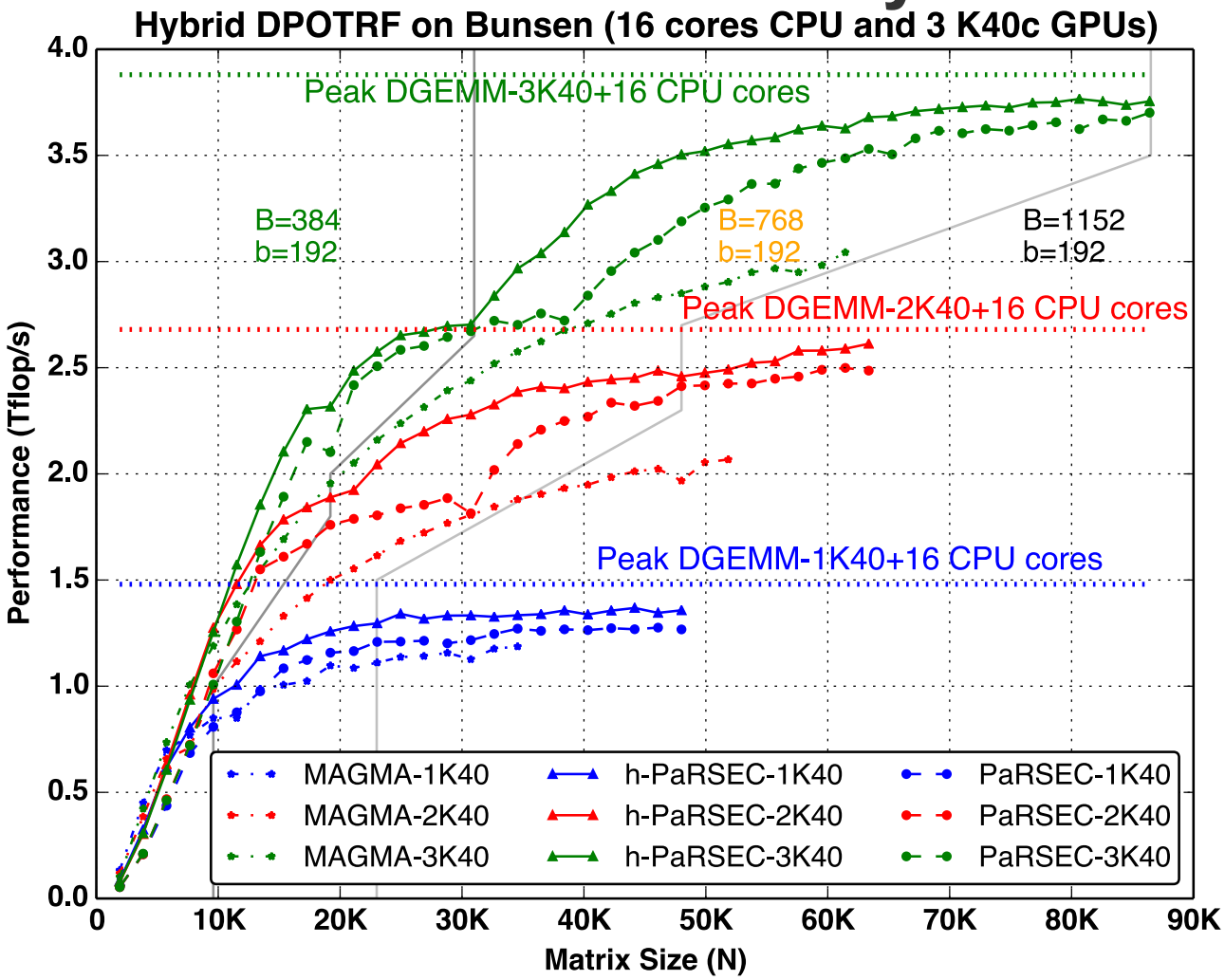


Performance Evaluation
Tile Size Tuning



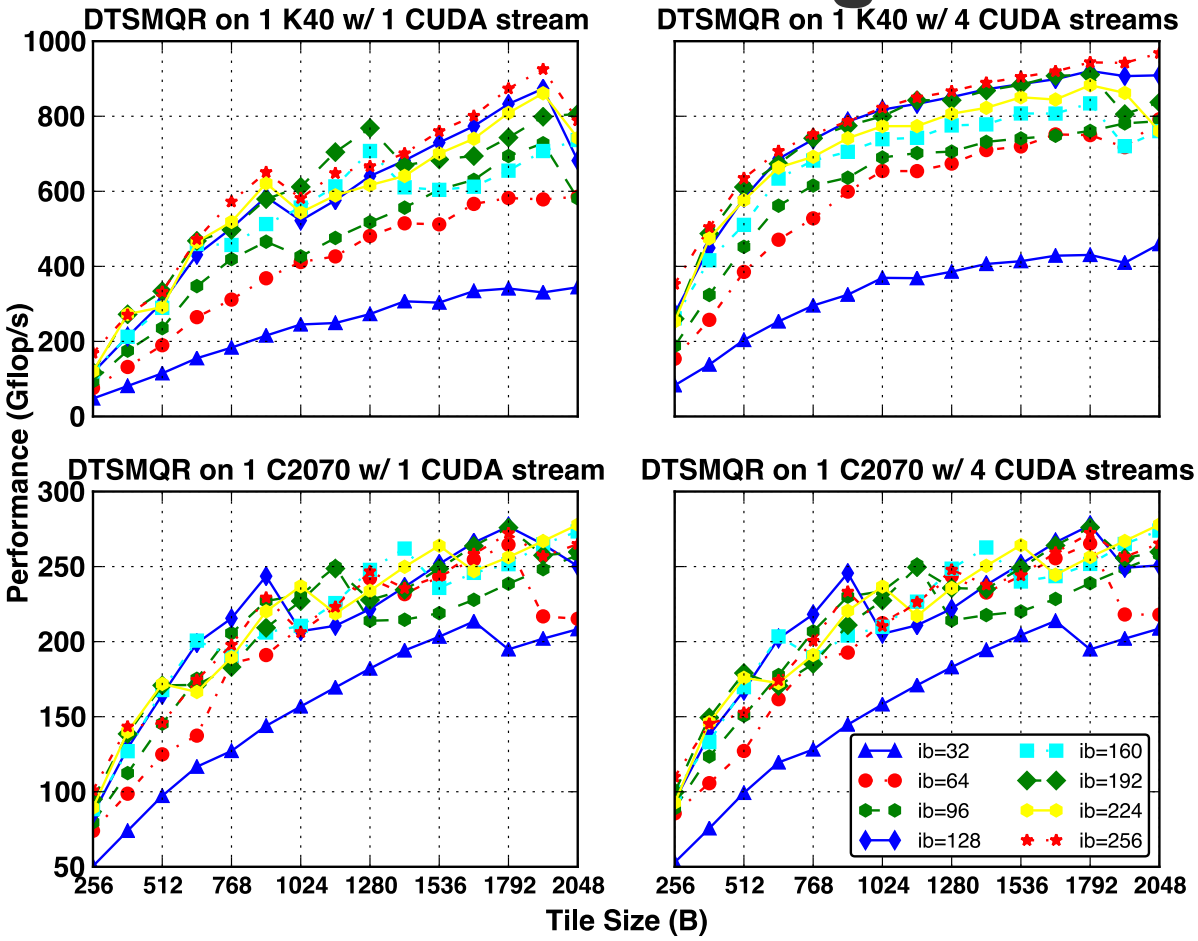
Performance for different tile size parameters (DPOTRF, using 1 GPU on Bunsen)

Performance Evaluation
Shared Memory



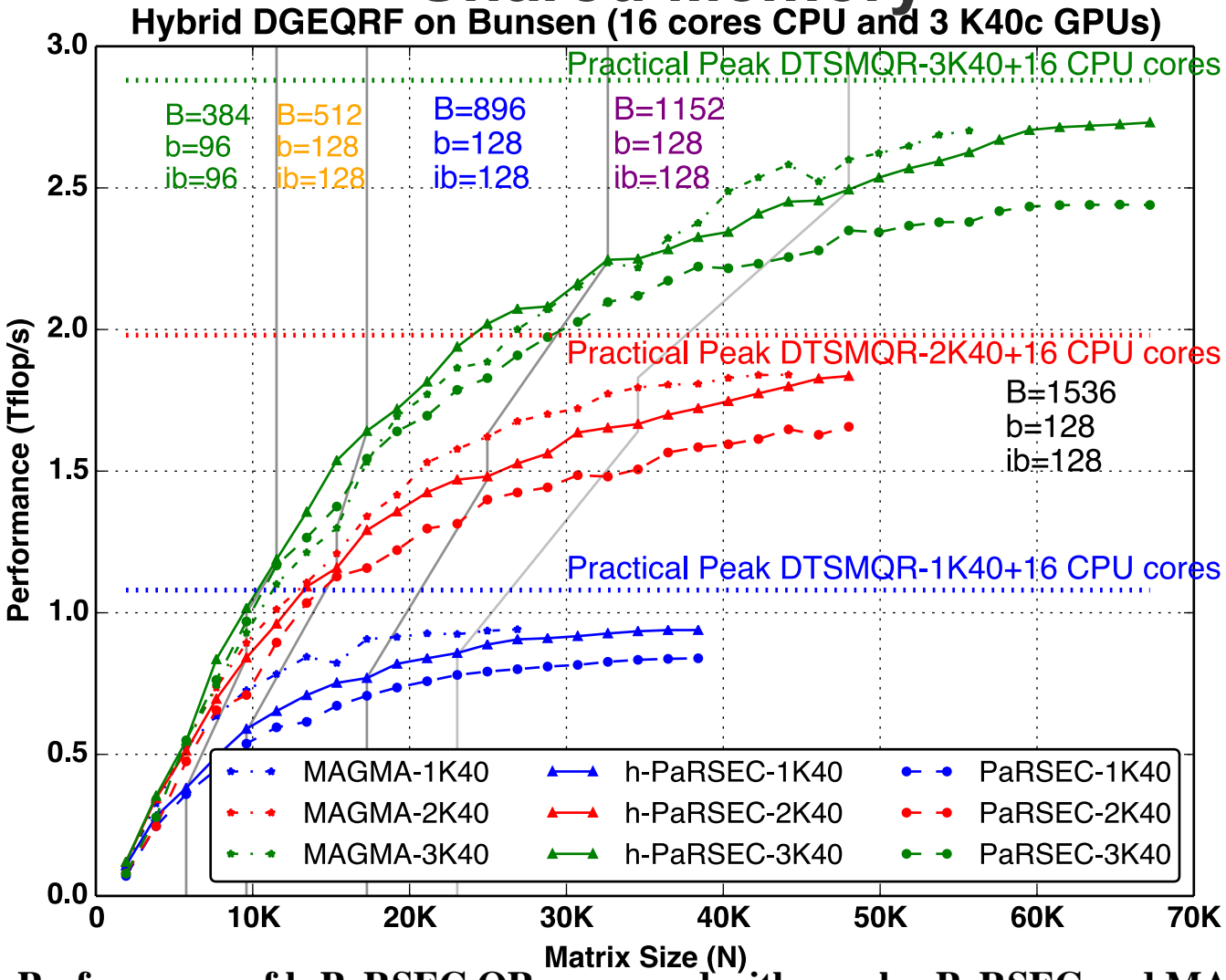
Performance of h-PaRSEC Cholesky compared with regular PaRSEC and MAGMA

Performance Evaluation
Tile Size Tuning-QR



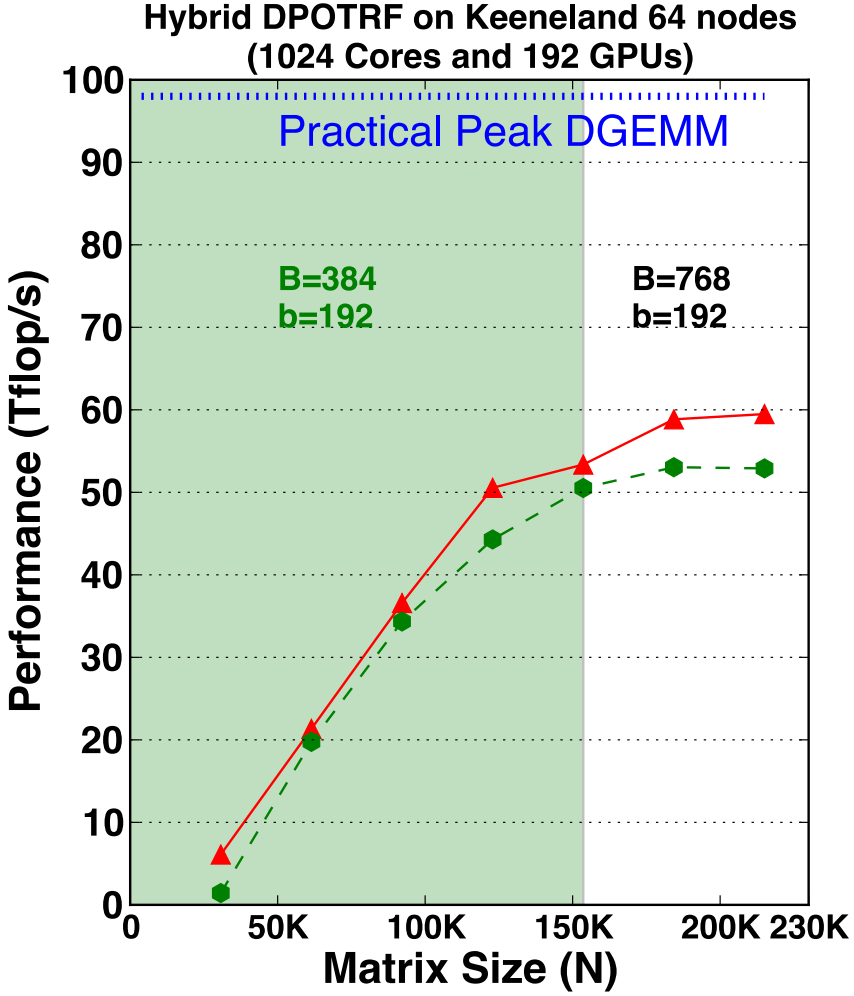
Performance of DTSMQR kernel on Fermi C2070 and Kepler K40 with 1 and 4 CUDA streams.

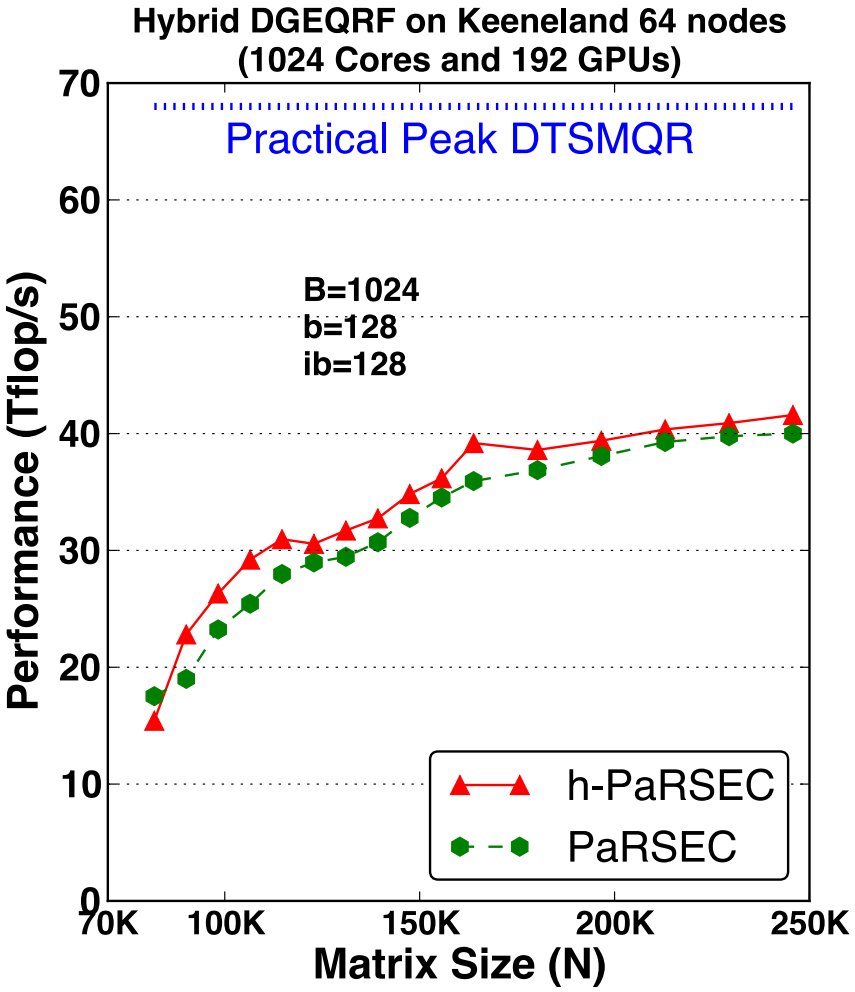
Performance Evaluation Shared Memory



Performance of h-PaRSEC QR compared with regular PaRSEC and MAGMA

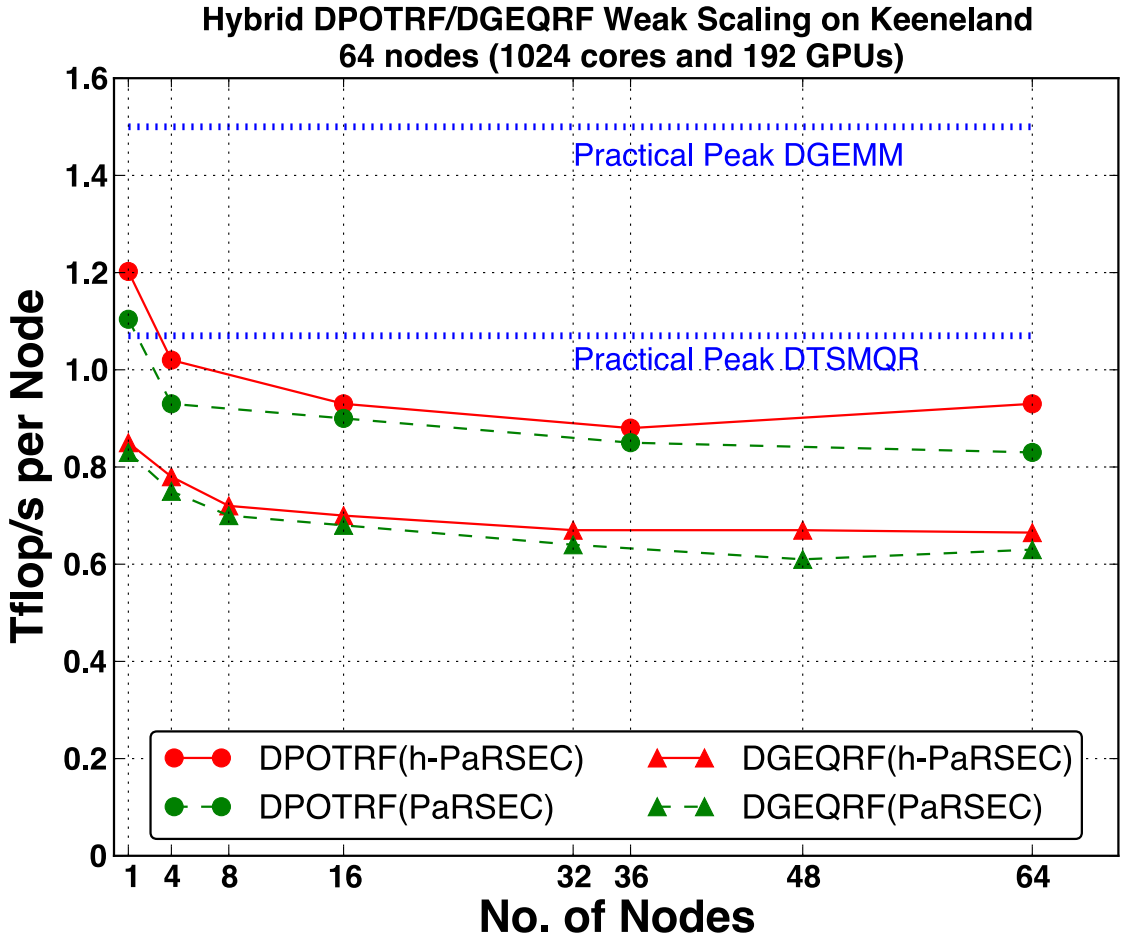
Performance Evaluation
Distributed Memory – Performance Scale





DPOTRF and DGEQRF performance with varying problem size; the dotted line is the DGEMM/DTSMQR peak performance (on 1 node, multiplied by 64)

Performance Evaluation
Distributed Memory – Weak Scale



Weak Scalability: DPOTRF and DGEQRF performance as a function of the number of nodes, with a problem size scaled accordingly (Keeneland, 3 M2090 GPUs and 16 cores per node)

Conclusions

- Proposed a “hierarchical DAG” approach that is able to adjust granularity of tasks depending on the type of resources where it will be executed
- From various performance evaluation, it is demonstrated that “hierarchical DAG” approach is able to improves the asymptotic performance for large matrices by employing the appropriate task grain on accelerators, while retaining a suitable amount of parallelism for CPU computations. It also enhances the scalability of the underlying algorithms.