Symmetric dense matrix tridiagonalization on a GPU cluster

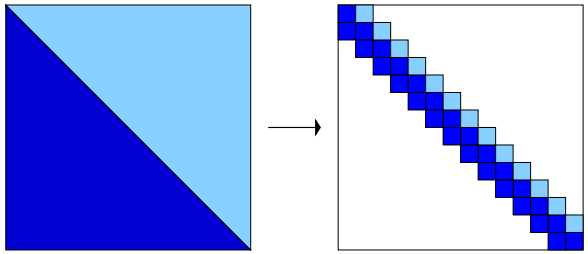
Ichitaro Yamazaki, Tim Dong, Stan Tomov, Jack Dongarra

Inovative Computing Lab.
University of Tennessee, Knoxville

Accelerators and Hybrid Exascale Systems (AsHES) Workshop
Boston, Massachusetts, 05/20/2013

Introduction

- Objective: reduces a matrix A into a tridiagonal matrix T,



where A is dense $(a_{ij} \neq 0)$ and symmetric $(A = A^T)$, through

$$Q^T A Q = T,$$

where Q is orthogonal.

- Motivation: often a bottleneck in solving symmetric dense eigenvalue problem:

$$Av = \lambda v \qquad \text{or} \qquad Av = \lambda Bv.$$

  - arise in many scientific and engineering simulations: e.g.,

    - electronics calculation, quantum physics, image processing, web analysis, etc.

MAGMA (Matrix Algebra on GPU and Multicore Architecture)

multi-GPU SYTRD integrated in

> dense and sparse eigensolvers

8GPU results are from R. Solcá of ETH.

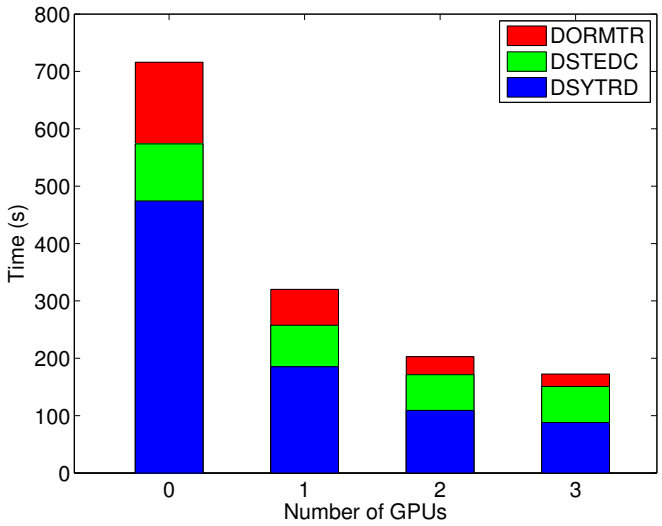
> electronic structure calculation

(e.g., Exciting, Elk, Abinit, QuantumEspress).

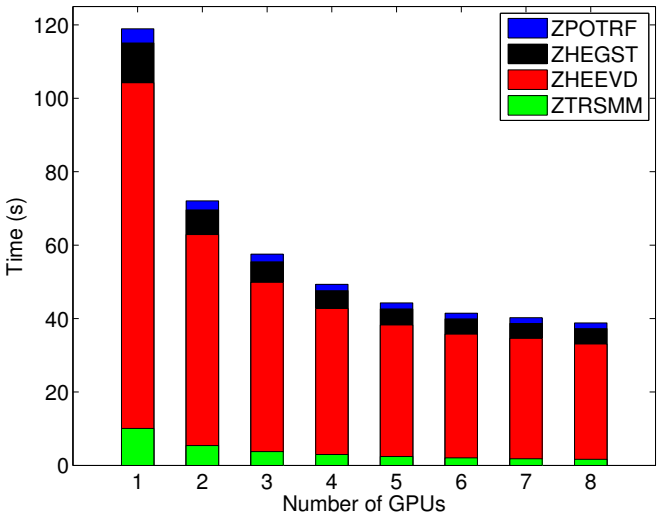
> optimized GPU kernels

with 1DBC on multiple GPUs.

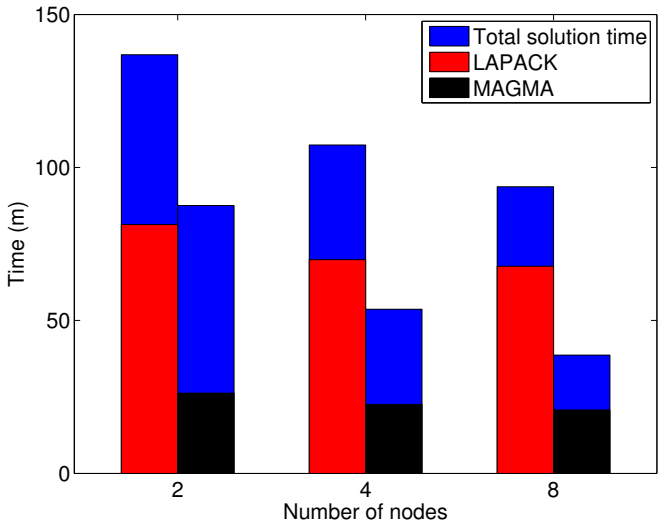
- multi-GPU dense symmetric solver -



- multi-GPU dense symmetric generalized solver -



- distributed sparse Lanczos solver -



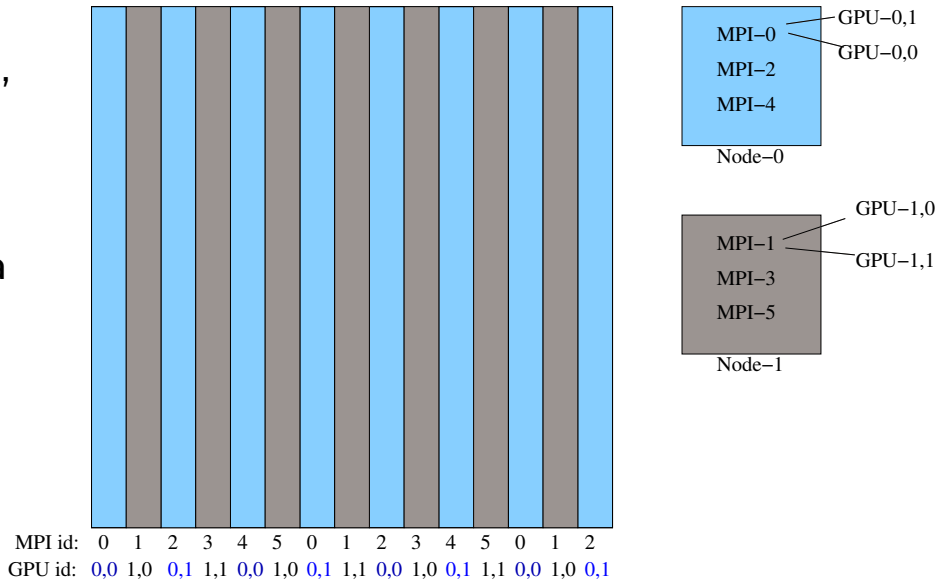
Extension to distributed GPUs

Motivation: solving larger-scale problems

- not easy to develop an out-of-GPU-memory version
  - whole trailing submatrix accessed to reduce each column
  - problem size limitted by GPU memory
- weak-scaling studies on tens of GPUs or nodes.

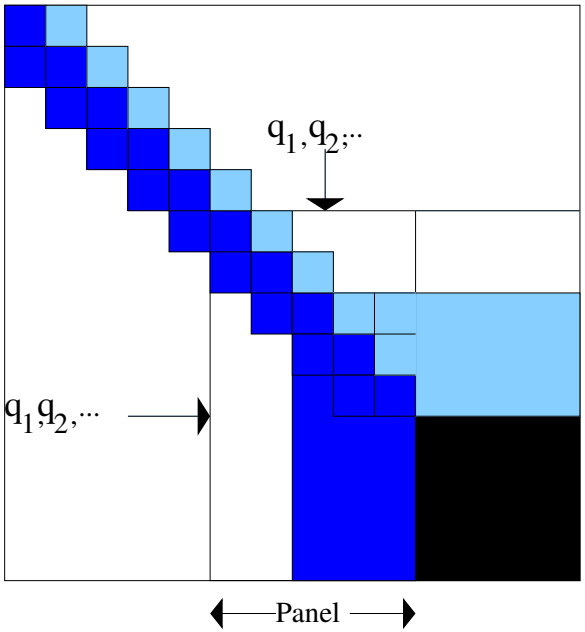
Our first step: ScaLAPACK with GPUs

- any number of MPIs/GPUs per node, but one MPI dispatch GPU kernels.
  - larger GPU kernel and smaller communication
- 1DBC and MPI mapped to cores in a round-robing among nodes.
  - our GPU-kernels recycled
- same optimization techniques (e.g., static schedule, overlapping CPU with GPU and MPI-comm of vectors).



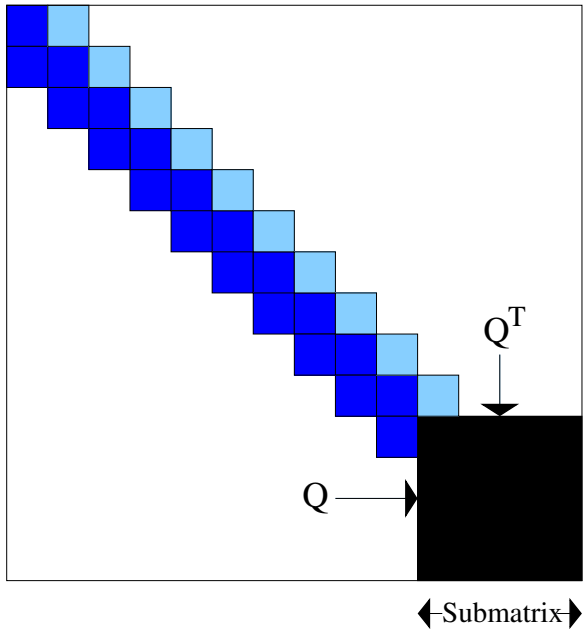
"Blocked" tridiagonalization algorithm

step 1:
panel factorization



for each column in the panel
- generate q_j to reduce the column
- accumurate it into a blocked update Q

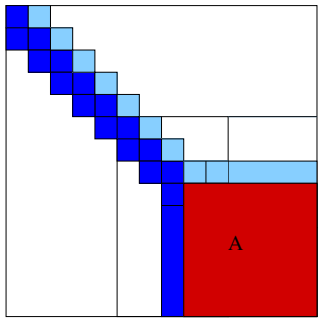
step 2:
trailing submatrix update



- reduce communication by a block update

computational kernels in tridiagonalization: total of $\frac{4}{3}n^3 + O(n^2)$ flops

1. **panel factorization:** tridiagonal reduction of a panel
   - **SYMV (bandwidth-limited BLAS-2)** about 50% of flops
     multiply with whole trailing submatrix $\widehat{A}$ to reduce each column



$$w_j := \widehat{A}v_j \quad \rightarrow \quad \frac{2\widehat{n}^2 \text{ flops}}{(\widehat{n}^2+\widehat{n})/2+\widehat{n} \text{ data}} \approx \frac{4 \text{ flops}}{\text{data}} \text{ (per call)}$$

$$\rightarrow \quad \text{total of } \frac{2}{3}n^3 \text{ flops}$$

$\longrightarrow$ exploit greater memory bandwidth of GPUs

2. **trailing submatrix update:** blocked orthogonal update of trailing submatrix:
   - **SYR2K (more data-parallel BLAS-3)** about 50% of flops
     symmetric rank-k update with each panel

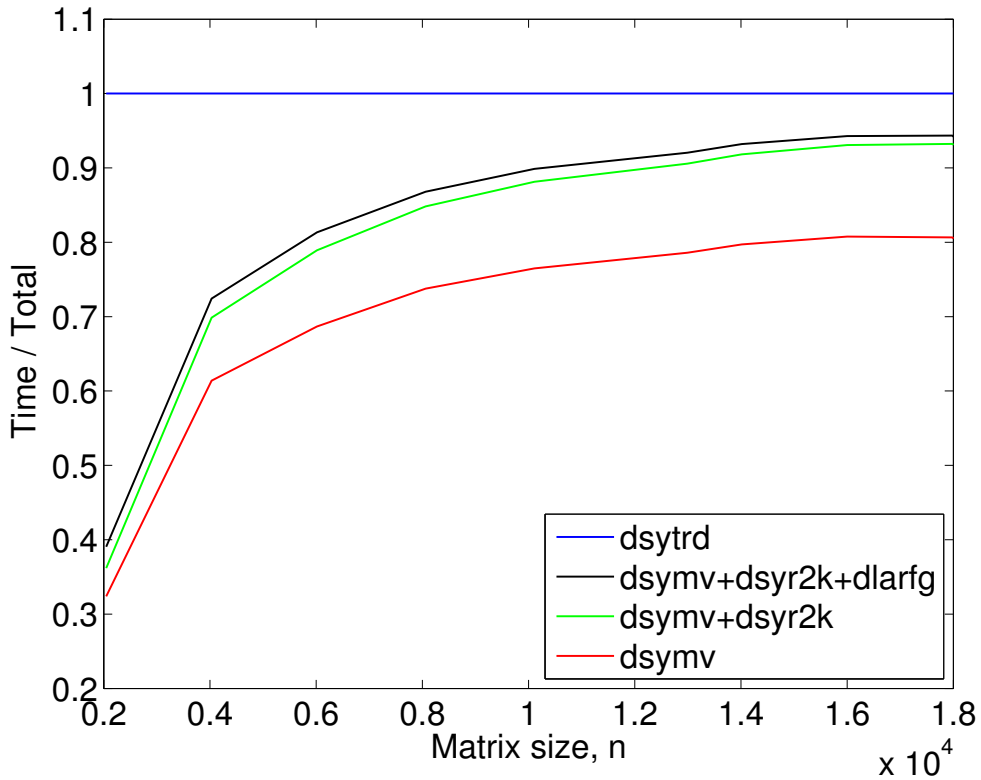
$$A := A - VW^T - WV^T \quad \rightarrow \quad \frac{2\widehat{n}^2 k \text{ flops}}{(\widehat{n}^2+\widehat{n})/2+\widehat{n}k \text{ data}} \approx \frac{4k \text{ flops}}{\text{data}} \text{ (per call, } k = 32, 64)$$

$$\rightarrow \quad \text{total of } \frac{2}{3}n^3 \text{ flops}$$

$\longrightarrow$ exploit larger computational throughput of GPUs

breakdown of reduction time using one GPU

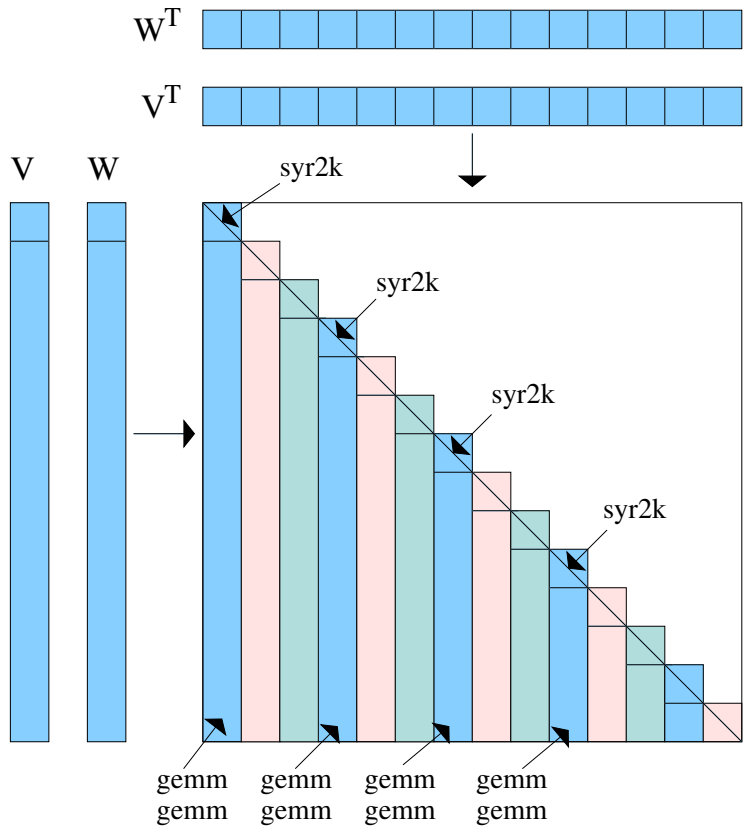
Keeneland: (two 8-core Intel SandyBridge CPUs + three NVIDIA M2090 GPUs)/node



- reduction time dominated by dsymv and dsyr2k (up to 90%), especially BLAS-2 dsymv (up to 80%).

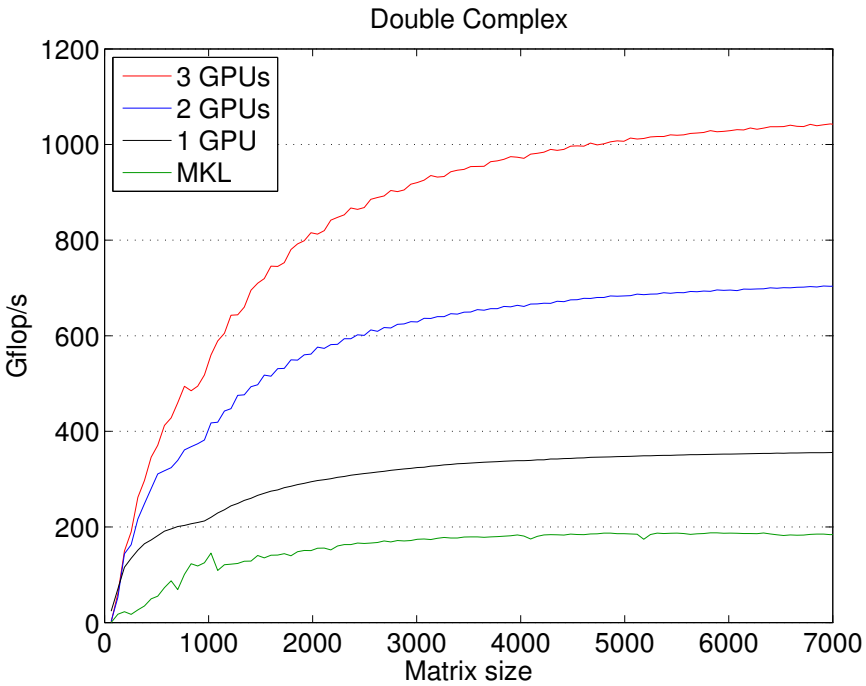
multi-GPU kernel 1: BLAS-3 SYR2K symmetric rank-k updates

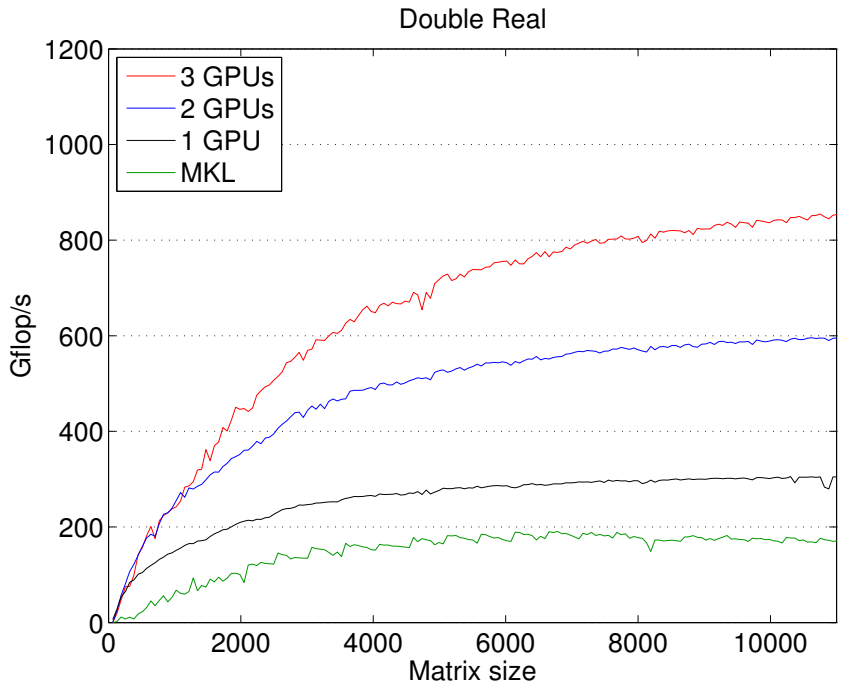
a sequence of "small" CUBLAS calls on streams to exploit data parallelism where A is statically distributed among GPUs in 1DBC



- each GPU updates block columns of its local submatrix
  - SYR2K(V_I, W_I) on diagonal block, and
    GEMM(V_I^T, $W_{I:N}$) and GEMM(W_I^T, $V_{I:N}$) on off-diagonal
  - multiple GPU streams cyclically on block columns

performance of SYR2K on multiple GPUs (Keeneland: three NVIDIA M2090 GPUs/node).



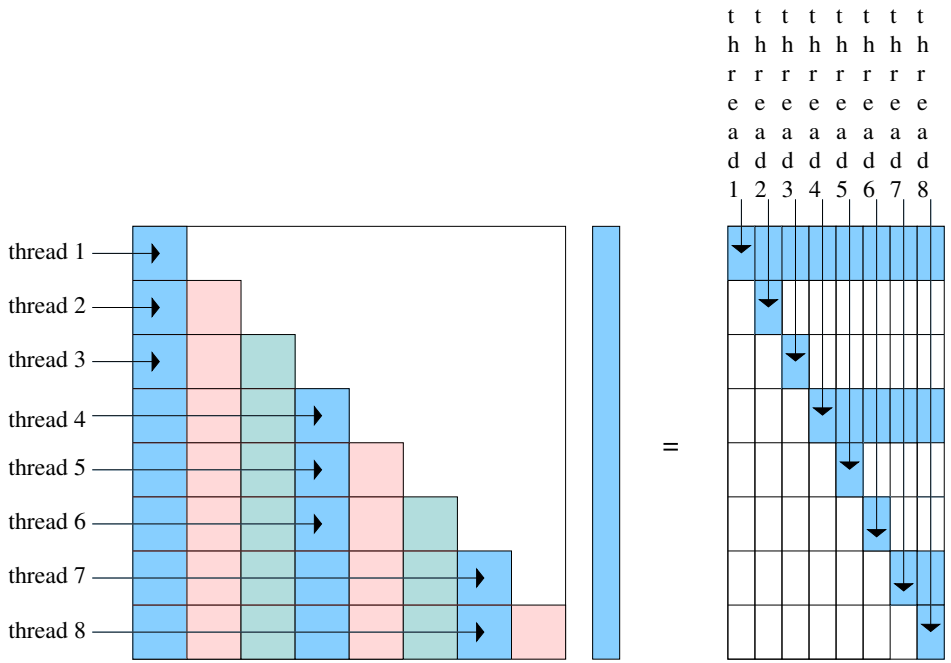


high data-parallelism

- peak double precision performance = 665Gflop/s $\rightarrow$ 40% of the peak
- 430 and 380 Gflops by zgemm and dgemm $\rightarrow$ 75% of gemm
- only about 15% of the reduction time is spent here

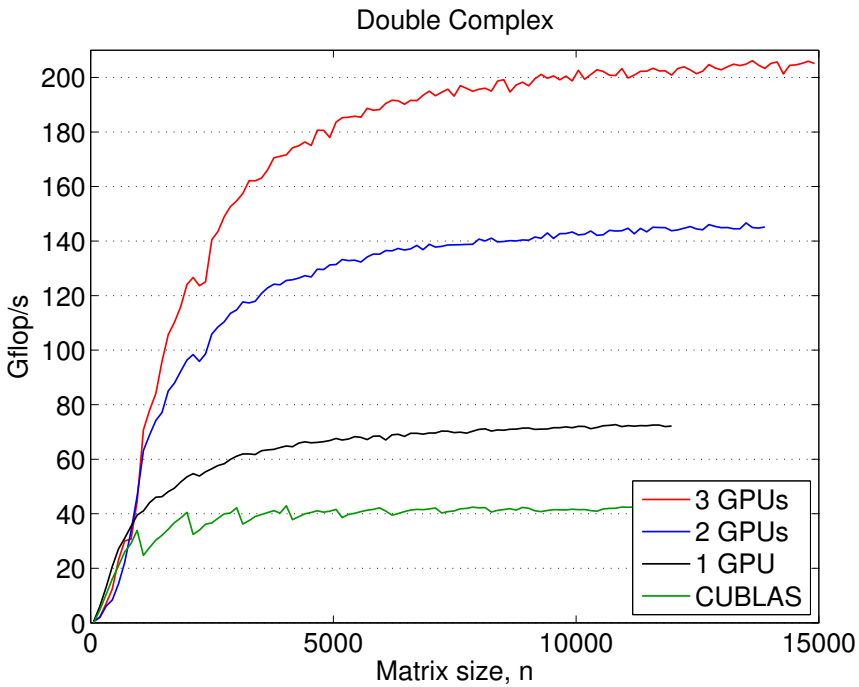
multi-GPU kernel 2: BLAS-2 SYMV symmetric matrix-vector multiplication

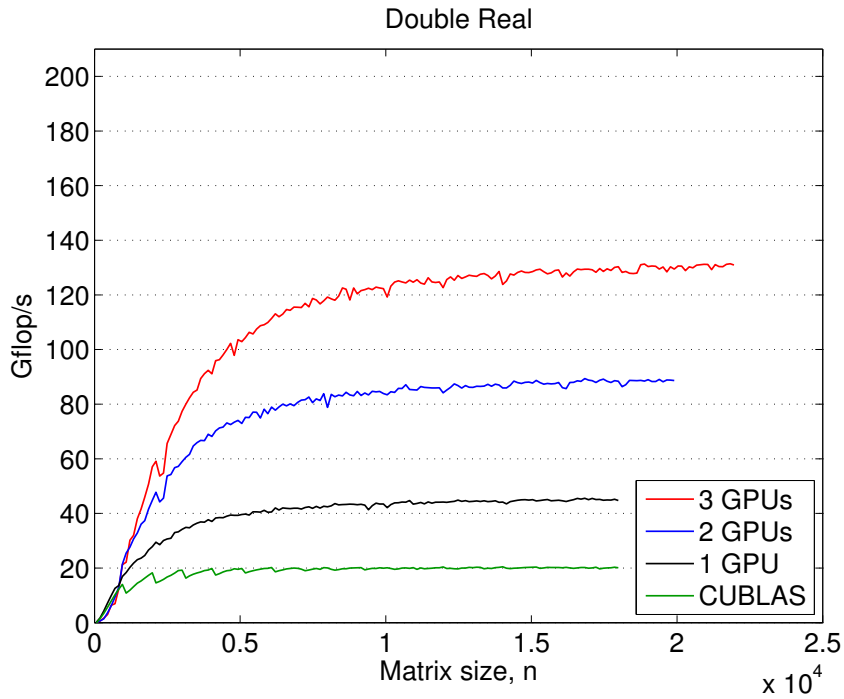
a specialized CUDA kernel to minimize data traffic for multiply with local matrix
(an extension of our SC'11 paper).



- each GPU thread processes a block row
- as each block A_{ij} is read, we multiply with A_{ij} and A_{ij}^T
  - each thread writes its partial result to its own workspace
  - another kernels is launched to sum the partial results
  - A is read only once from the GPU memory

performance of SYMV on multiple GPUs (Keeneland: three NVIDIA M2090 GPUs/node).





bandwidth-limited operation:

- zhemv performs twice more operations $\rightarrow$ twice the Gflop/s
- 120GB/sec with ECC on $\rightarrow$ peaks are 120 and 60 Gflop/s for zhemv and dsymv
  $\rightarrow$ 65% – 75% of this peak
  $\rightarrow$ 15% – 20% of gemm
- up to 80% of the reduction time is spent here

putting them together: multi-GPU tridiagonalization

1 statically distribute A in 1DBC

2 for each panel

- 2.1 panel factorization
  - for each column in the panel
    - 2.1.1 update a_j with previous v and w on CPUs
    - 2.1.2 compute Householder reflector on CPUs
    - 2.1.3 broadcast reflector to all GPUs
    - 2.1.4 SYMV of local submatrices on GPUs in parallel
    - 2.1.5 copy vector v_j back to CPU
    - 2.1.6 compute w_j on CPUs
- 2.2 trailing submatrices update
  - 2.2.1 broadcast V and W to GPUs
  - 2.2.2 SYR2K of local submatrix on GPUs in parallel

for each GPU call, communicate:

- local matrix ($\frac{n^2+n}{2\times \text{gpus}}$ data) from GPU memory
- vector(s) (n or nk data) between CPU and GPU (non-blocking MPI + GPU streams)

hybrid CPU-GPU computing with asynch. MPI and GPU communication

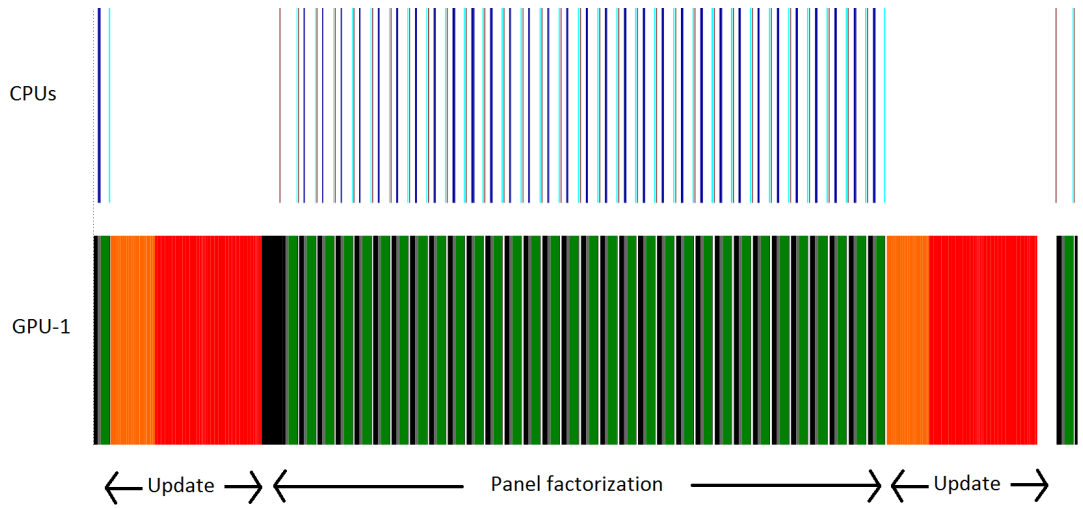
- SYR2K: hide communication of V or W behind panel factorization
  using asynch MPI send and dedicated GPU stream

- SYMV: overlap CPU and GPU computation (and hide CPU-GPU communication)
  for multiplying with updated $\widehat{A}$ (left-look within panel, right-look between panels),

$$(\widehat{A} - \widehat{V}\widehat{W}^T - \widehat{W}\widehat{V}^T)v_j,$$

where $\widehat{A}$ is $\widehat{n} \times \widehat{n}$, while $\widehat{V}$ and $\widehat{W}$ are $\widehat{n} \times (j-1)$.

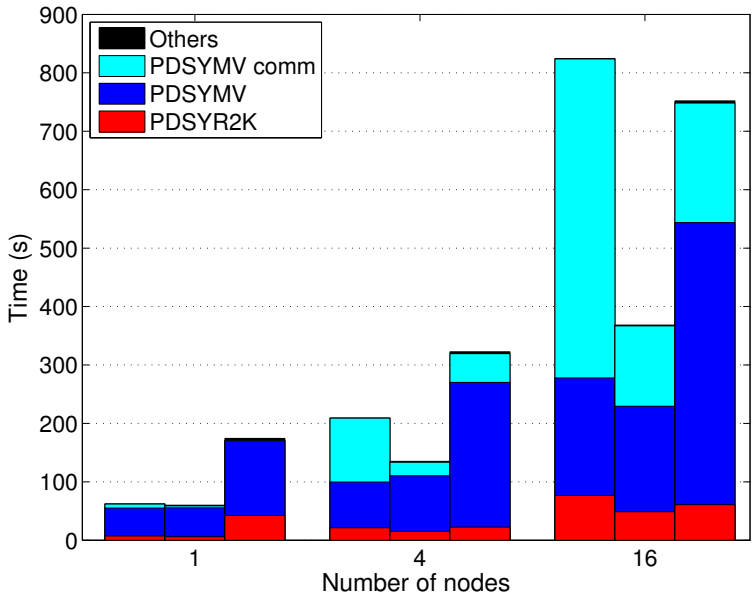
- $\widehat{A}v_j$ on GPUs, while $(\widehat{V}\widehat{W}^T + \widehat{W}\widehat{V}^T)v_j$ (and updating next column) on CPUs.



- a piece of a trace on 3 GPUs -

Performance on Keeneland: ScaLAPACK

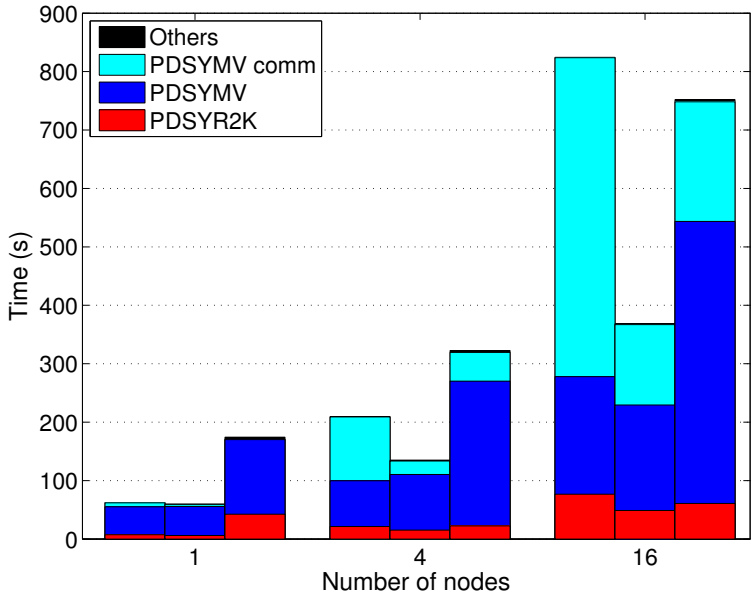
- Keeneland: two 8-core 2.6GHz Intel Xeon processors (SandyBridge) and three NVIDIA Tesla M2090 GPUs.
- weak-scaling studies: matrix size $= 11,520 \times (\text{number of nodes})^{\frac{1}{2}}$.

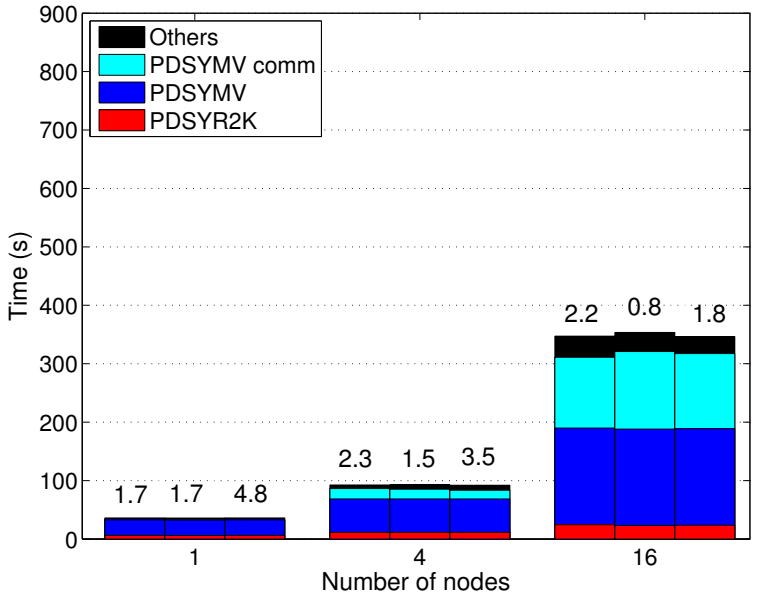


- hybrid-programming performed well

  - 1MPI/socket obtained best computation and communication performance

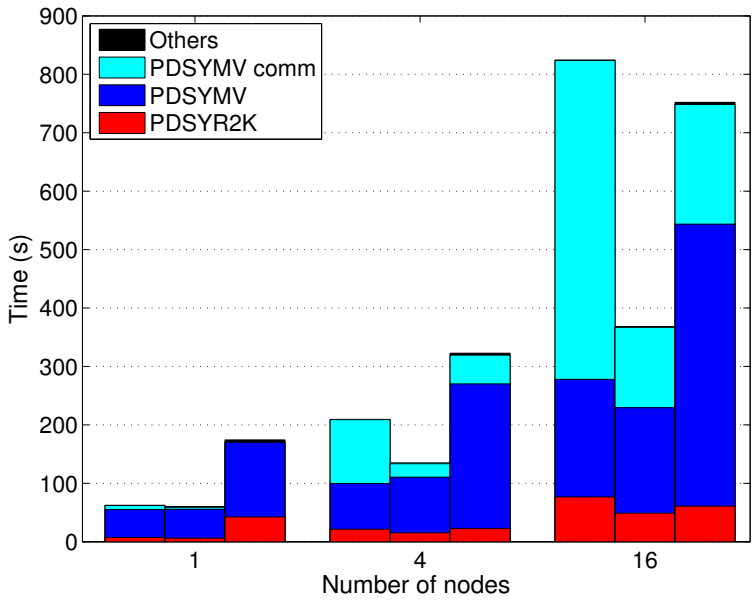
Performance on Keeneland: ScaLAPACK+1GPU/node

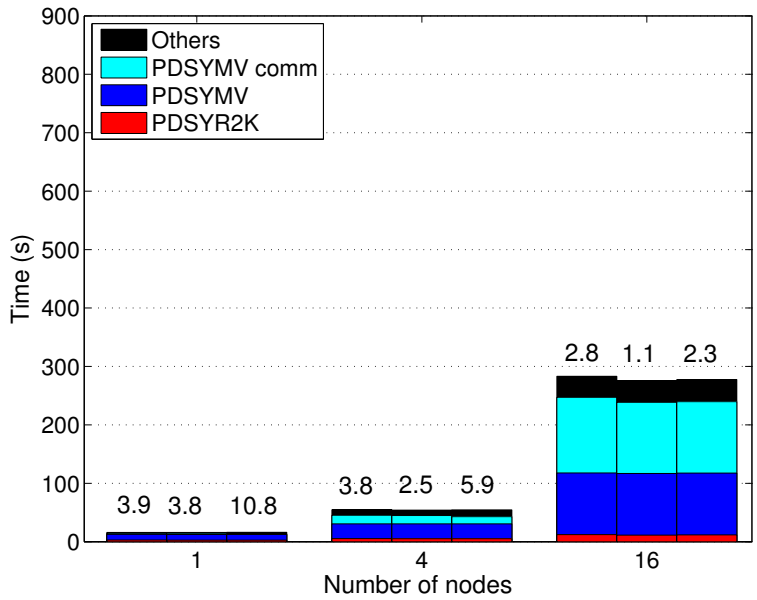




- GPU-extension got reasonable speedups
  - significant speedups over 1MPI/core or 1MPI/node.
  - similar performance as 1MPI/socket.

Performance on Keeneland: ScaLAPACK+3GPUs/node

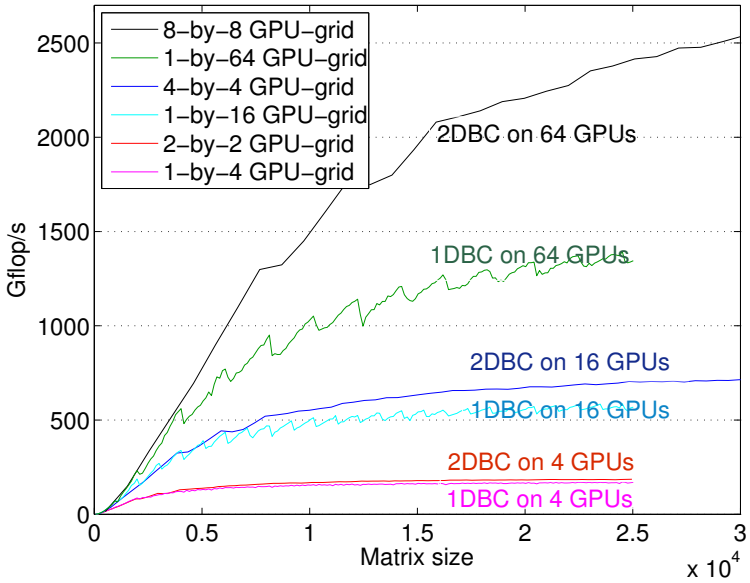


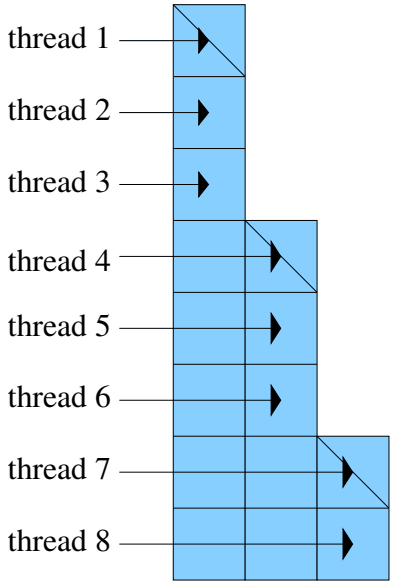


- GPU-extension got reasonable speedups
- it is difficult to scale to large-scale,
  - non-blocking MPI-GPU comm. to reduce waiting time
    $\rightarrow$ MPI communication starts fo dominate.
  - GPU kernel to reduce computation time
    $\rightarrow$ GPU efficiency goes down on many nodes

  $\uparrow$ both of which may be addressed by 2DBC.

Performance of SYMV in 2DBC vs. 1DBC

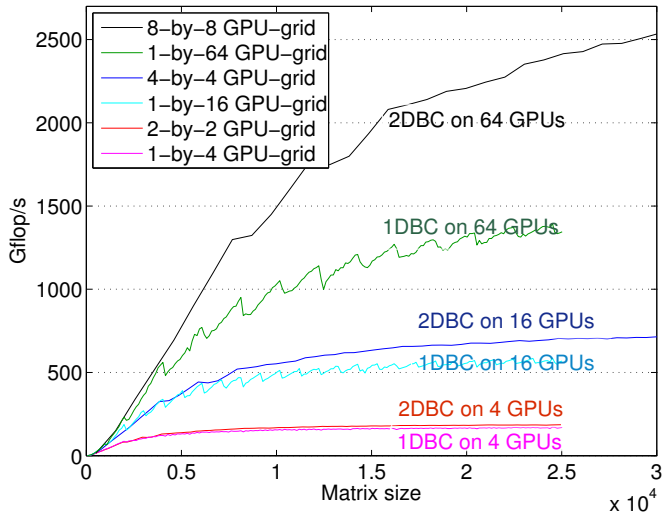




- local submatrix is closer to square with 2DBC
  - GPU-kernel performs better on a square matrix
- it is being integrated into ScaLAPACK.

Final remarks

- ok speedups on a small number of nodes
- challenge to scale to more nodes
  - MPI communication starts to dominate
  - GPU kernel does not scale well



Current and future work:

- 2DBC (or any other suggestion?) to obtain higher-performance
  - to reduce communication and to improve GPU utilization
  - larger-scale studies and distributed Kepler?
- runtime system? two-stage on distributed GPUs?