Hierarchical-matrix Linear Solver on GPU clusters

with MAGMA variable-size batched kernel

Ichitaro Yamazaki*, Ahmad Abdelfattah*, Akihiro Ida†,
Satoshi Ohshima‡, Stanimire Tomov*, Rio Yokota♯, Jack Dongarra*

*The University of Tennessee, Knoxville, USA
†The University of Tokyo, Japan
‡Kyushu University, Japan
♯Tokyo Institute of Tehcnology, Japan

GPU Technology Conference
San Jose, CA, 03/26/2018

Boundary Equation Method: from integral equation to linear equations

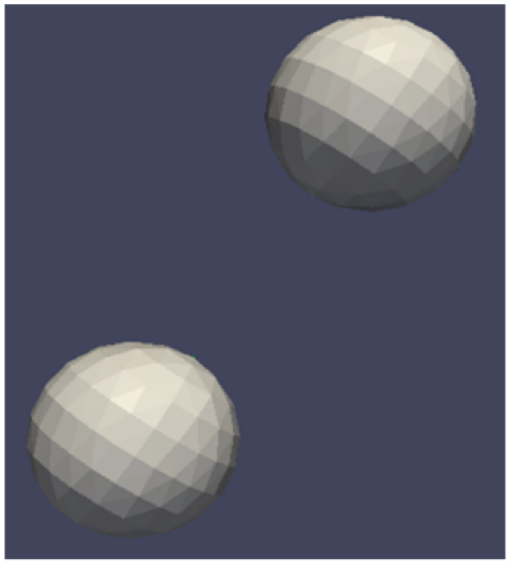
- many scientific and engineering applications
  (e.g., acoustics, electro magnetics, and fracture and fluid mechanics)
- numerical solution of integral equation

$$\int_{\Omega} K(\mathbf{x}, \mathbf{y})\mathbf{u}(\mathbf{y})\mathrm{d}\mathbf{y} = \mathbf{f}$$

  $\rightarrow$ solution of dense linear system

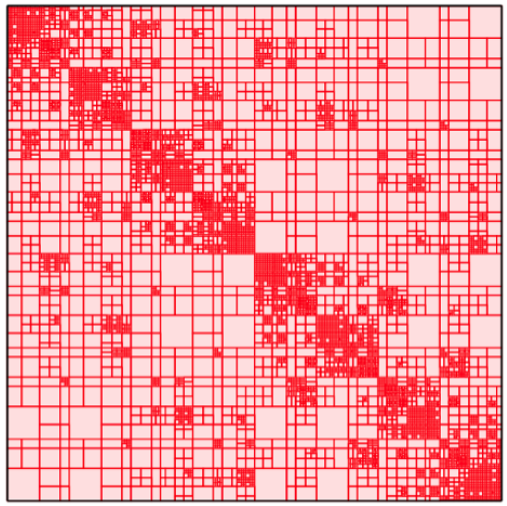
$$A\phi = \mathbf{b}$$

- sizes of problems limited by cost of solving the linear system



HACApK: dense linear solver

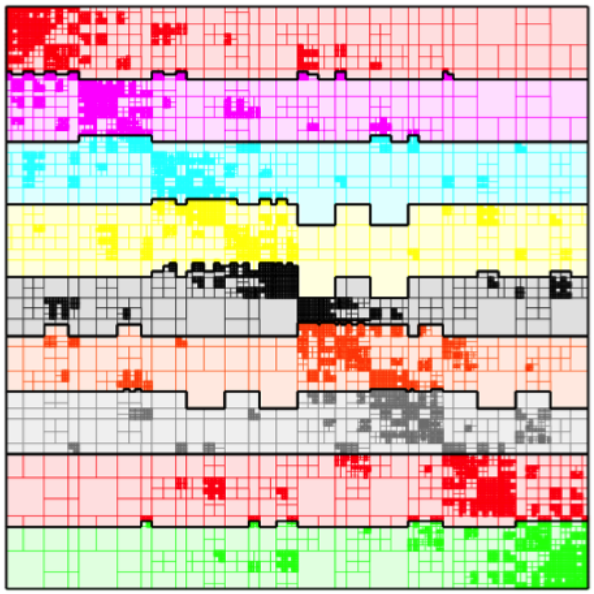
- solves dense linear system of equations
  e.g., for BEM (ppohBEM).
- reduces computational and storage costs by compressing the matrix into $\mathcal{H}$-matrix
  ▷ reordered/partitioned using geometry of problem
- uses Krylov solver like BiCGStab for computing the solution
- is available at
  `http://ppopenhpc.cc.u-tokyo.ac.jp`



→ this talk focuses on utilizing GPUs

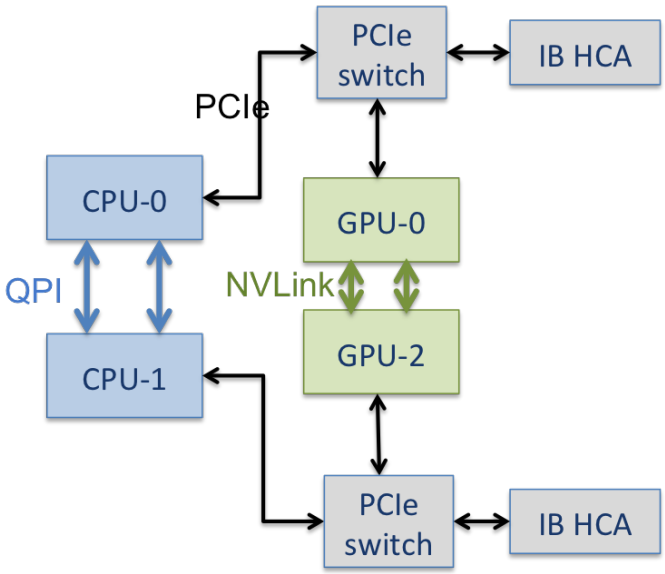
BiCGStab with $\mathcal{H}$-matrix on distributed-memory computer

1: $\mathbf{t} := A\mathbf{x}$
2: $\mathbf{r} := \mathbf{b} - \mathbf{t}$; $\mathbf{r}_0 := \mathbf{r}$, $\gamma := \|\mathbf{r}_0\|_2$
3: **for** $iter = 1, 2, \ldots, maxiters$ **do**
4: $\quad \mathbf{p} := \mathbf{r} + \beta \cdot (\mathbf{p} - \zeta \cdot \mathbf{v})$
5: $\quad \mathbf{v} := A\mathbf{p}$, followed by Allgatherv
6: $\quad \alpha = (\mathbf{r}_0, \mathbf{r})/(\mathbf{r}_0, \mathbf{v})$
7: $\quad \mathbf{v} := \mathbf{r} - \alpha \cdot \mathbf{v}$
8: $\quad \mathbf{t} := A\mathbf{v}$, followed by Allgatherv
9: $\quad \zeta = (\mathbf{t}, \mathbf{v})/(\mathbf{t}, \mathbf{t})$
10: $\quad \mathbf{x} := \mathbf{x} + \alpha\mathbf{p} + \zeta\mathbf{v}$
11: $\quad \mathbf{r} := \mathbf{v} - \zeta\mathbf{t}$
12: $\quad \beta = \alpha/\zeta \cdot (\mathbf{r}_0, \mathbf{r})/\gamma$
13: $\quad \gamma = \|\mathbf{r}\|$
14: **end for**

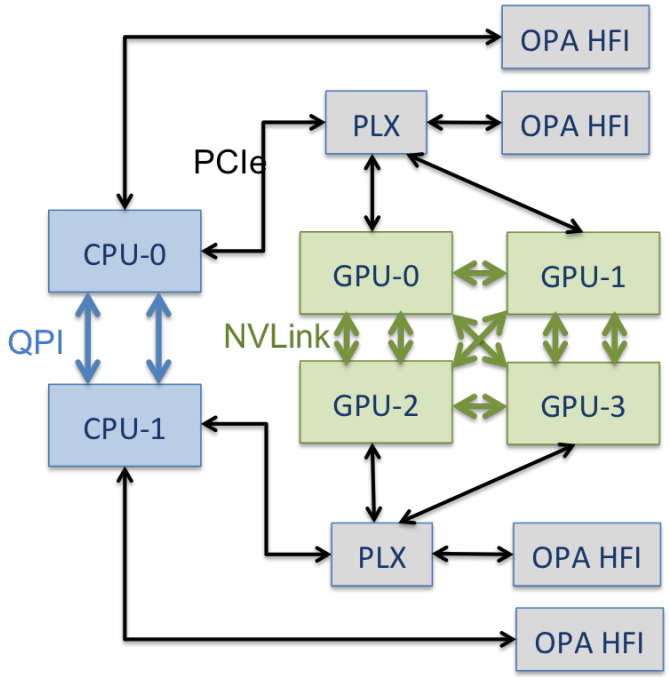


- `HiMV` is dominant in iteration time, and parallelized
  ▷ 1D block row, but with $\mathcal{H}$-blocks and non-disjoint rows
- vector operations are insignificant, and redanduntly computed
  ▷ avoid `all-reduces` for five dot-products per iter
- `MPI_Allgatherv` after each `HiMV`
- OpenMP threads may be used to parallelize local matrix/vector

GPU testbeds



Reedbush-H at University of Tokyo

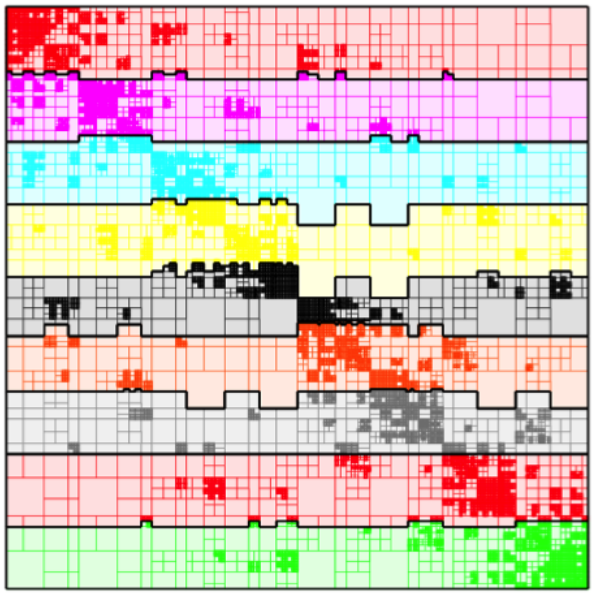


Tsubame-3 at Tokyo Tech

- **Reedbush-H**: two 18-core Intel Xeon CPUs and two NVIDIA P100 GPUs per node, connected with 2×56Gb/s InifiniBand
- **Tsubame-3**: two 14-core Intel Xeon CPUs and four NVIDIA P100 GPUs per node, connected with 4×100 Gb/s Omni-Path

BiCGStab with $\mathcal{H}$-matrix on GPU cluster

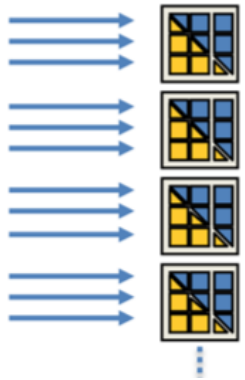
1: $\mathbf{t} := A\mathbf{x}$
2: $\mathbf{r} := \mathbf{b} - \mathbf{t}$; $\mathbf{r}_0 := \mathbf{r}$, $\gamma := \|\mathbf{r}_0\|_2$
3: **for** $iter = 1, 2, \ldots, maxiters$ **do**
4: $\quad \mathbf{p} := \mathbf{r} + \beta \cdot (\mathbf{p} - \zeta \cdot \mathbf{v})$
5: $\quad \mathbf{v} := A\mathbf{p}$, followed by Allgatherv
6: $\quad \alpha = (\mathbf{r}_0, \mathbf{r})/(\mathbf{r}_0, \mathbf{v})$
7: $\quad \mathbf{v} := \mathbf{r} - \alpha \cdot \mathbf{v}$
8: $\quad \mathbf{t} := A\mathbf{v}$, followed by Allgatherv
9: $\quad \zeta = (\mathbf{t}, \mathbf{v})/(\mathbf{t}, \mathbf{t})$
10: $\quad \mathbf{x} := \mathbf{x} + \alpha\mathbf{p} + \zeta\mathbf{v}$
11: $\quad \mathbf{r} := \mathbf{v} - \zeta\mathbf{t}$
12: $\quad \beta = \alpha/\zeta \cdot (\mathbf{r}_0, \mathbf{r})/\gamma$
13: $\quad \gamma = \|\mathbf{r}\|$
14: **end for**



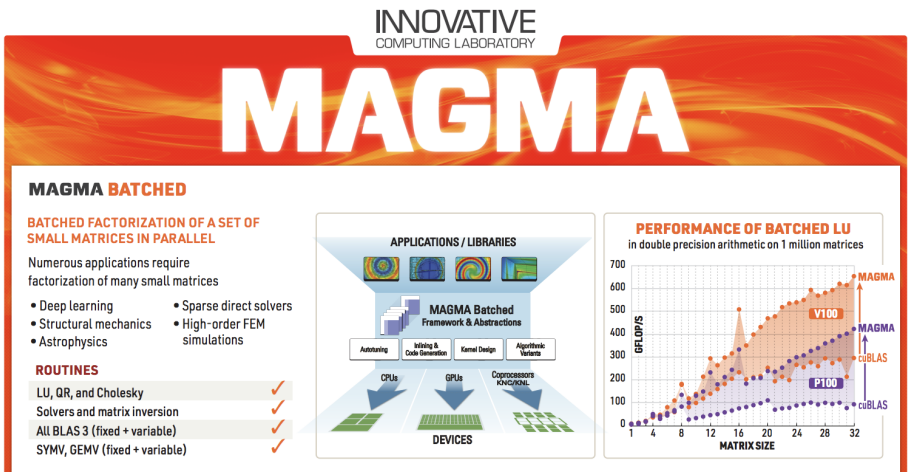
- all the operations are on GPUs (CPUs schedules tasks)
  - ▷ CPU-GPU data copy before/after MPI call
  - ▷ vector-operations using CUBLAS
  - ▷ `HiMV` using batched kernel !!

    fine-grained irregular computation + global communication

batched GPU kernels from MAGMA



- many small same operations in parallel
- hardware parallelism through data parallilization
- motizated by application needs
  (e.g., deep learning, structural mechanics, high-order FEM, astrophysics, sparse/dense solvers)
- **MAGMA**: http://www.icl.utk.edu/magma
  LU, QR, Cholesky (fixed), all BLAS-3 (fixed or variable), and
  `SYMV` and `GEMV` (fixed or variable)



http://www.icl.utk.edu/files/print/2017/magma-sc17.pdf (SC'17 handout)

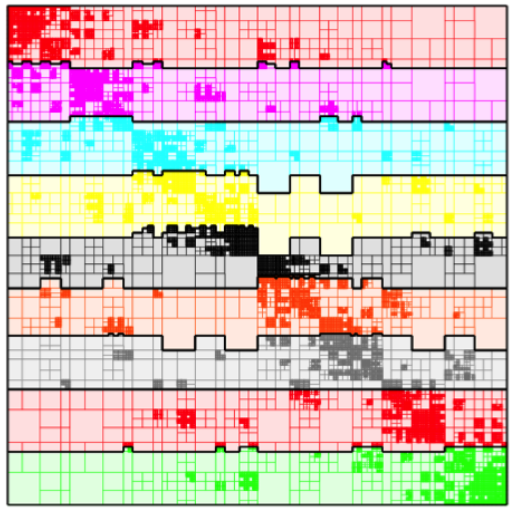
interface to variable-size batched DGEMV kernel

```
magmablas_dgemv_vbatched(
    magma_trans_t   trans,
    magma_int_t   * m,
    magma_int_t   * n,
    double          alpha,
    magmaDouble_ptr dA_array[], magma_int_t* ldda,
    magmaDouble_ptr dx_array[], magma_int_t* incx,
    magmaDouble_ptr dy_array[], magma_int_t* incy,
    magma_int_t     batchCount,
    magma_queue_t   queue)
```

- matrices/vectors as arrays of size `batchCount` on GPU (i.e., `dA`, `dx`, `dy`)
  ▷ maximum `batchCount` is 65,536
- variable matrix sizes as arrays on GPU (e.g, `m`, `n`, `lda`)
- same operations (i.e., `trans` and `alpha`)
- layered interface (e.g., `magmablas_dgemv_vbatched_nocheck`)

integration of variable-size batched kernel into HiMV

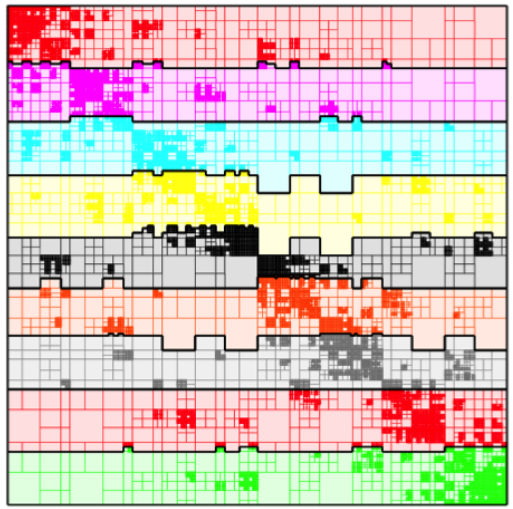
for $k = 1, 2, \ldots, n_\ell$ **do**
  if dense block **then**
    // multiply with dense $B^{(k)}$
    $\mathbf{y}^{(k)} := B^{(k)}\mathbf{x}^{(k)}$
  else
    // multiply with compressed $U^{(k)}V^{(k)}$
    $\mathbf{t}^{(k)} := V^{(k)}\mathbf{x}^{(k)}$
    $\mathbf{y}^{(k)} := U^{(k)}\mathbf{t}^{(k)}$
  end if
end for



- variable-size batched kernel to perform a batch of `dgemv` in parallel
  ▷ group `dgemv`s into multiple batches (e.g., of fixed batch count)
- `HiMV` with many small `dgemv` with dense or compressed blocks
  ▷ flat `for`-loop without hierarchical recursion
  → effective integration of batched kernel

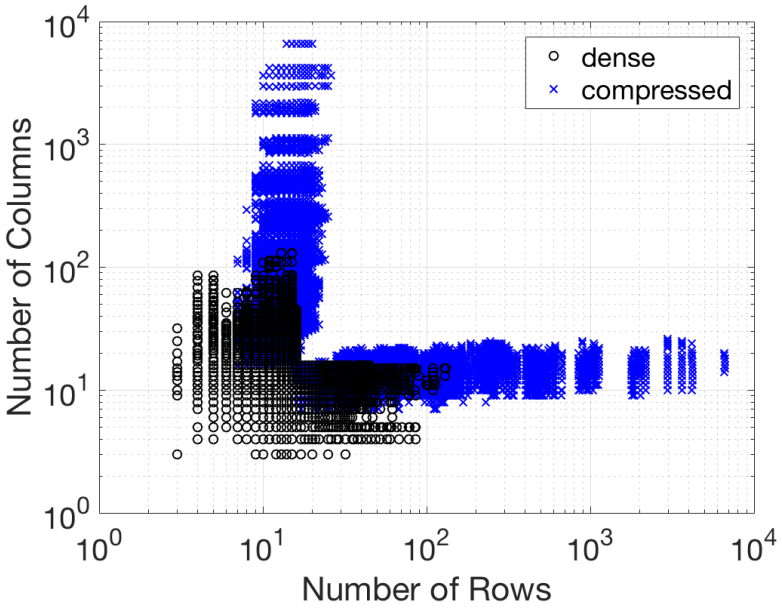
integration of variable-size batched kernel into HiMV

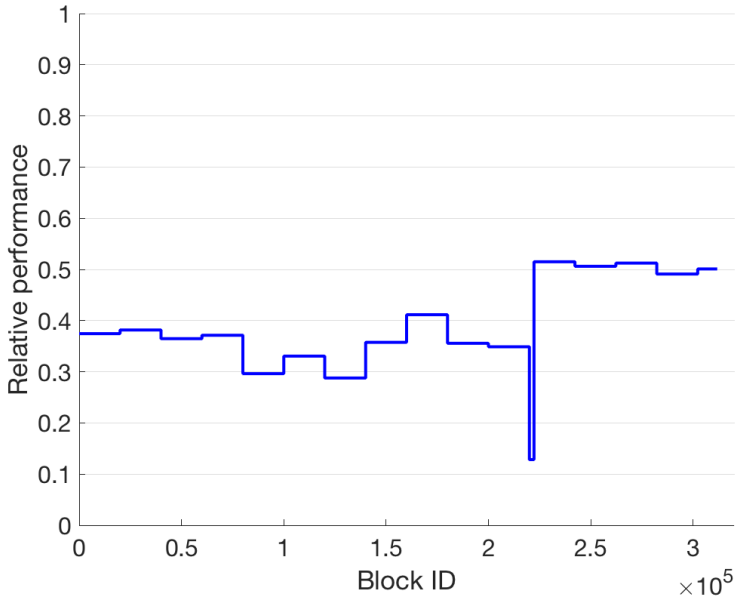
for $k = 1, 2, \ldots, n_\ell$ **do**
  if dense block **then**
    // multiply with dense $B^{(k)}$
    $\mathbf{y}^{(k)} := B^{(k)}\mathbf{x}^{(k)}$
  else
    // multiply with compressed $U^{(k)}V^{(k)}$
    $\mathbf{t}^{(k)} := V^{(k)}\mathbf{x}^{(k)}$
    $\mathbf{y}^{(k)} := U^{(k)}\mathbf{t}^{(k)}$
  end if
end for



- two data conflicts:
  - ▷ output $\mathbf{y}^{(k)}$ may overlap
    → NVIDIA's atomic-add on $\mathbf{y}$
  - ▷ multiply with $U^{(k)}$ depends on $\mathbf{t}^{(k)}$ from that with $V^{(k)}$
    → 1) batches of $B^{(k)}$ and $V^{(k)}$, and then
    2) batches of $U^{(k)}$
    on same stream, or on multiple streams with events

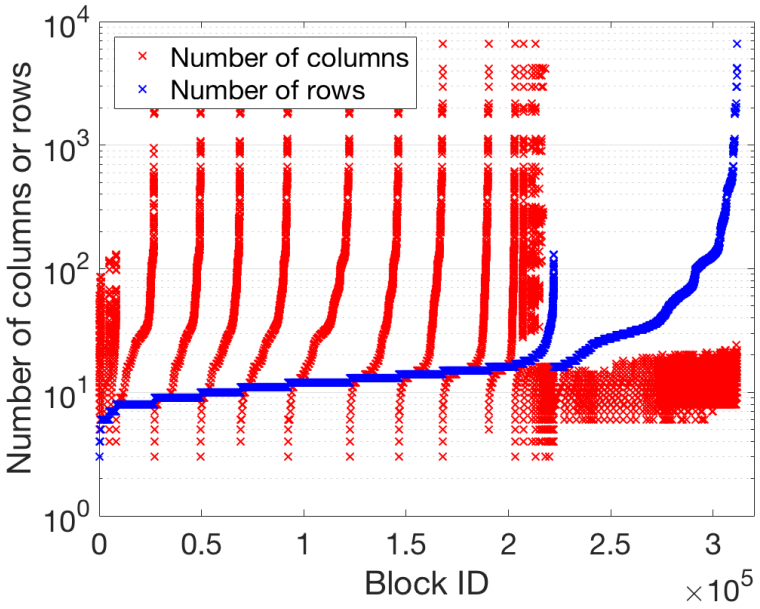
performance of batched kernel for HiMV

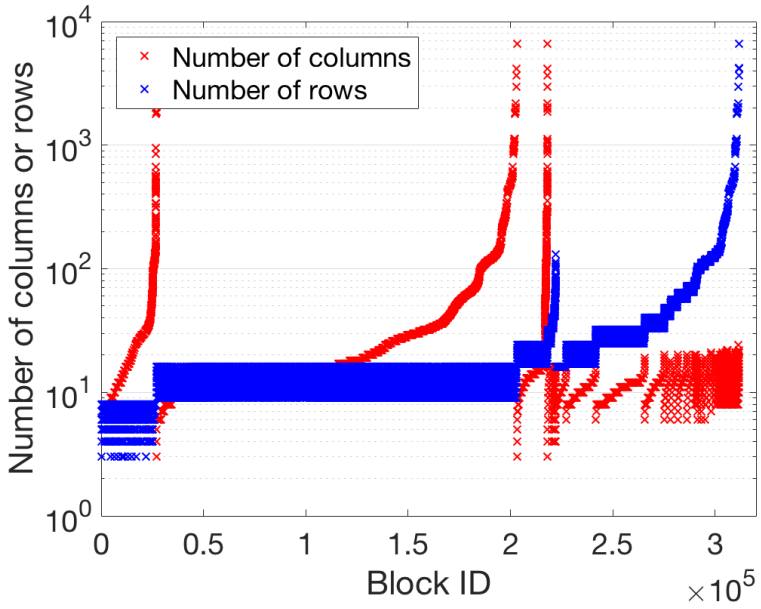




- a wide range of block sizes
  - ▷ diagonal blocks : dense & square
  - ▷ off-diagonal blocks: dense/compress & tall-skinny/short-wide
- overhead with variable sizes, e.g.,
  to accomodate largest block, smaller blocks have thread blocks with no work
- lower variable-size performance (Gflop/s)

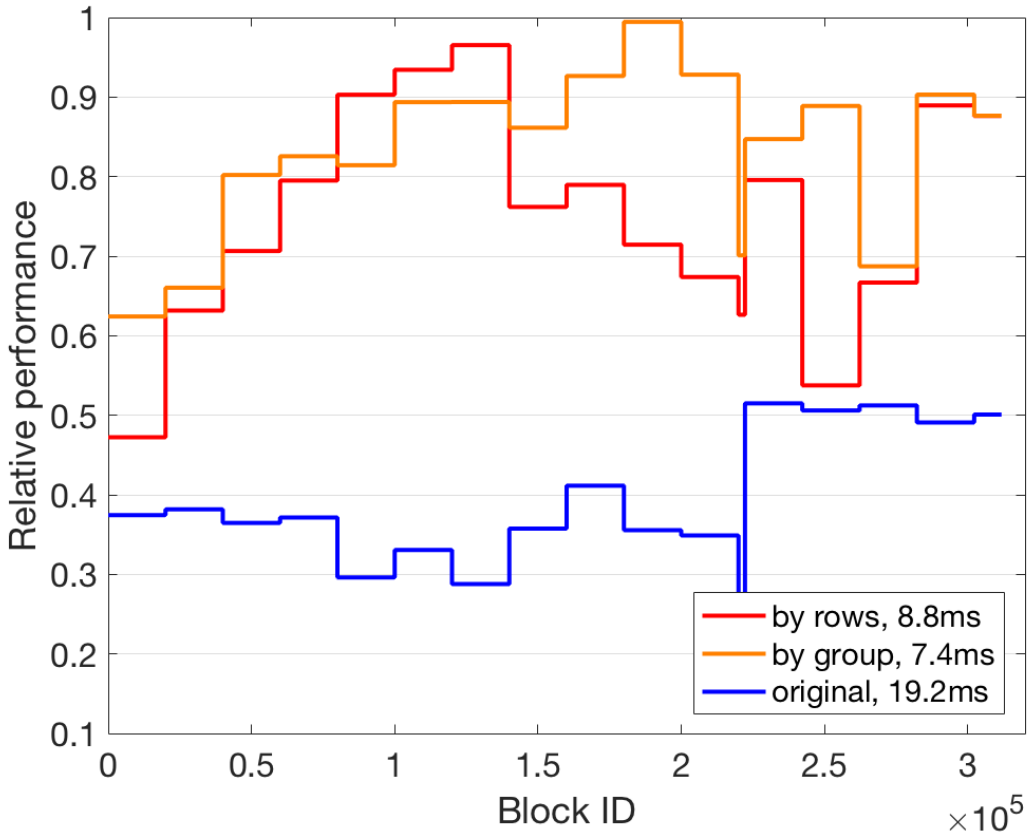
performance of batched kernel for HiMV





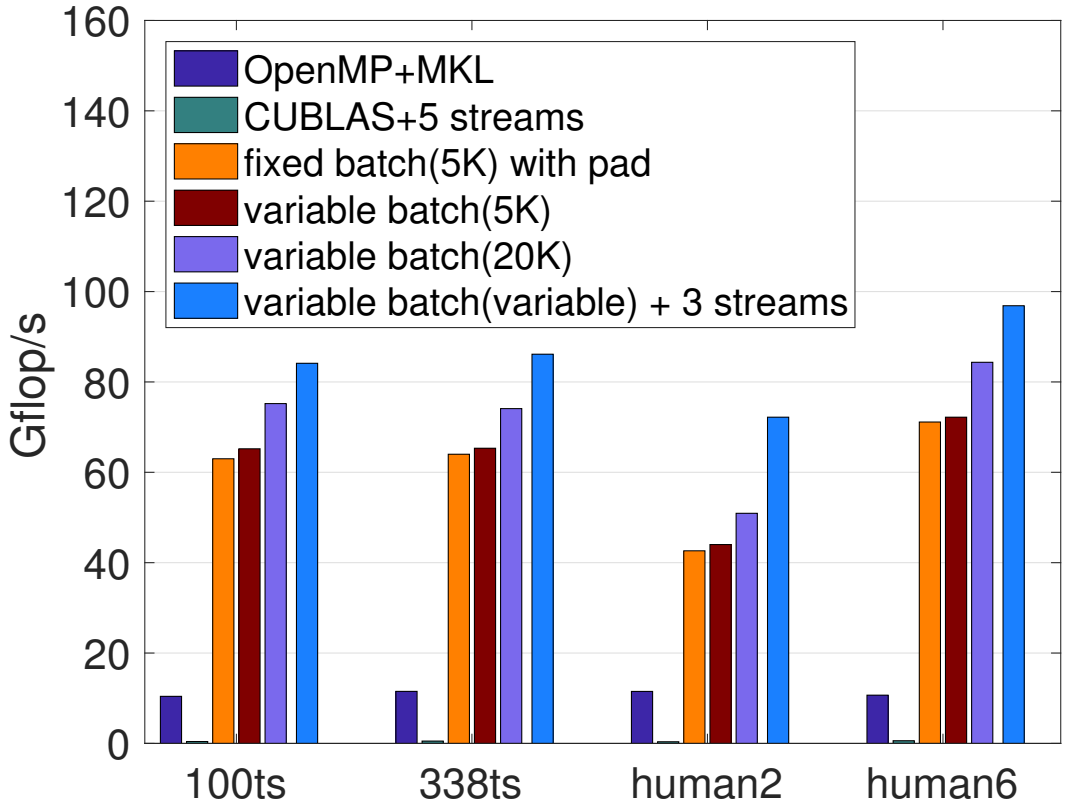
- sort blocks to reduce overhead associated with variable-size blocks
  - ▷ sort by numbers of rows in blocks
  - ▷ group by number of rows, sort by numbers of columns within group

performance of batched kernel for `HiMV`



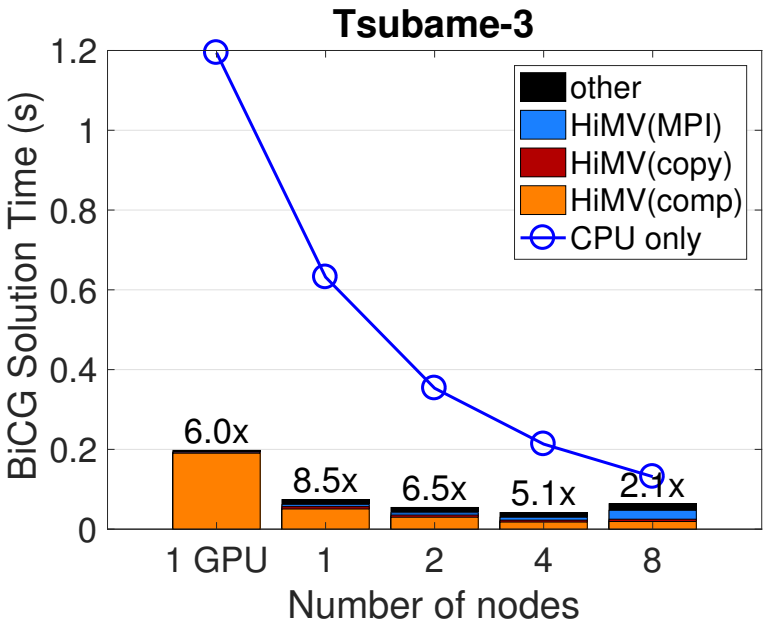
- appropriate sorting scheme improves performance
  - ▷ up to 2.5× speedups

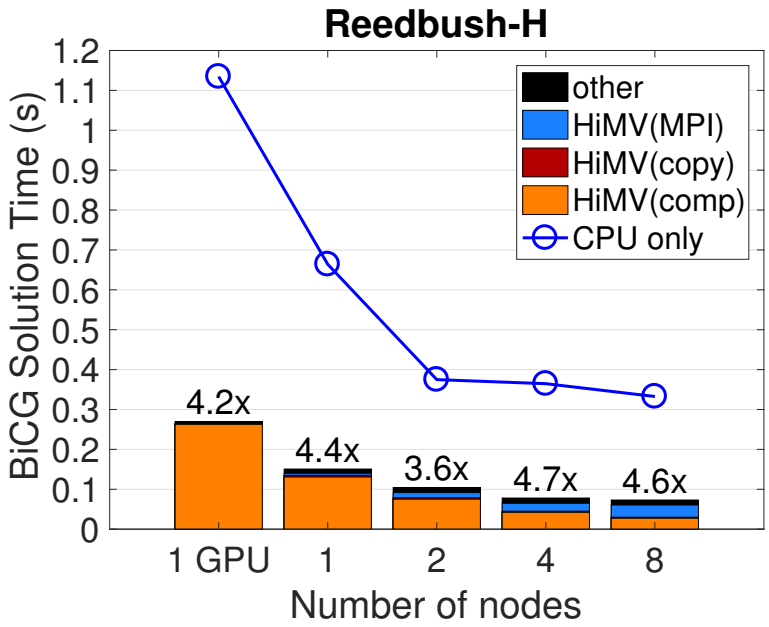
performance (Gflop/s) of different `HiMV` implementations



- obtained higher performance using variable-size GPU kernel compared to fixed-size (wasted ops with zeros or limited batch count)
- last three rows: variable batch counts to reduce overhead
  - ▷ specific range of block sizes in each batch

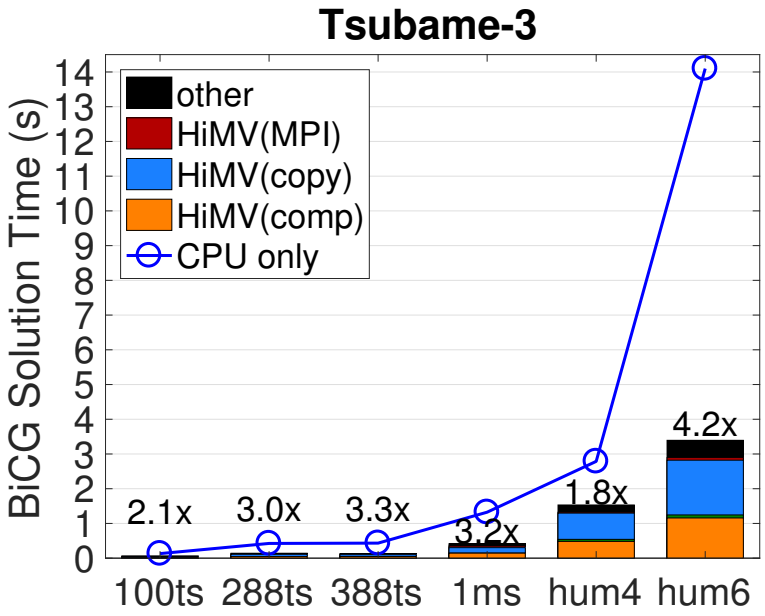
BiCGStab performance with GPUs (strong scaling)

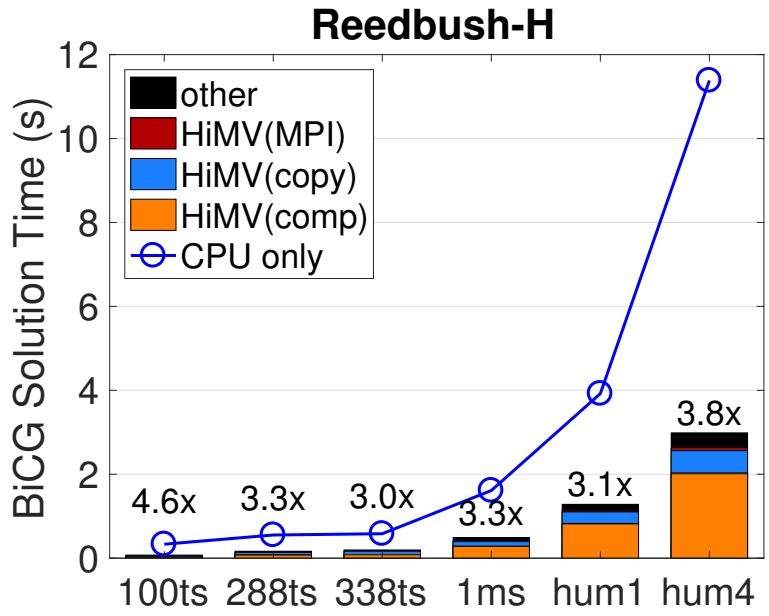




- CPU runs (one process/socket with threads) vs. GPU runs (one process/GPU)
- 2.1× speedup on 8 nodes of Tsubame-3
- 4.6× speedup on 8 nodes of Reedbush-H

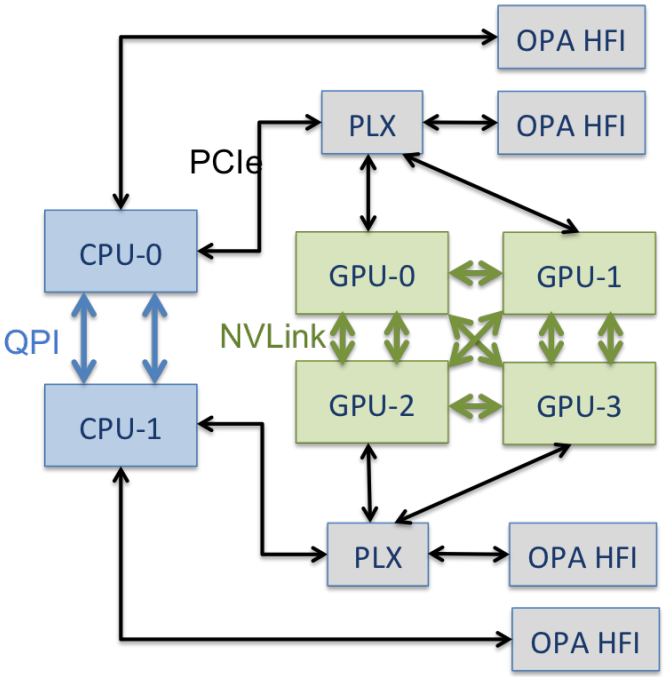
BiCGStab performance with GPUs on 8 nodes



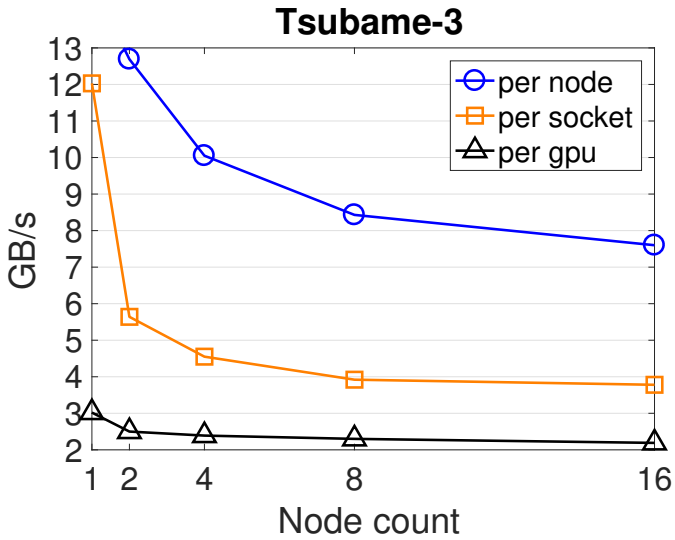


- CPU runs (one process/socket with threads)
  vs. GPU runs (one process/GPU)
- upto 4.2× speedup on 8 nodes of Tsubame-3
  ▷ 6.0× on one node
- upto 4.6× speedup on 8 nodes of Reedbush-H
  ▷ 4.2× on one node
- communication starts to become significant
  ▷ 46% or 43% on Tsubame-3 or Reedbush-H

BiCGStab with multiple GPUs per process

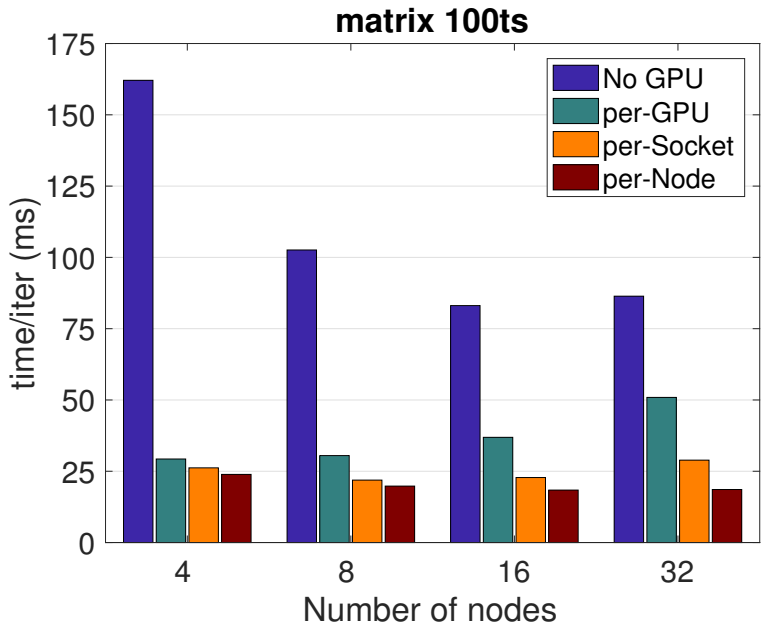


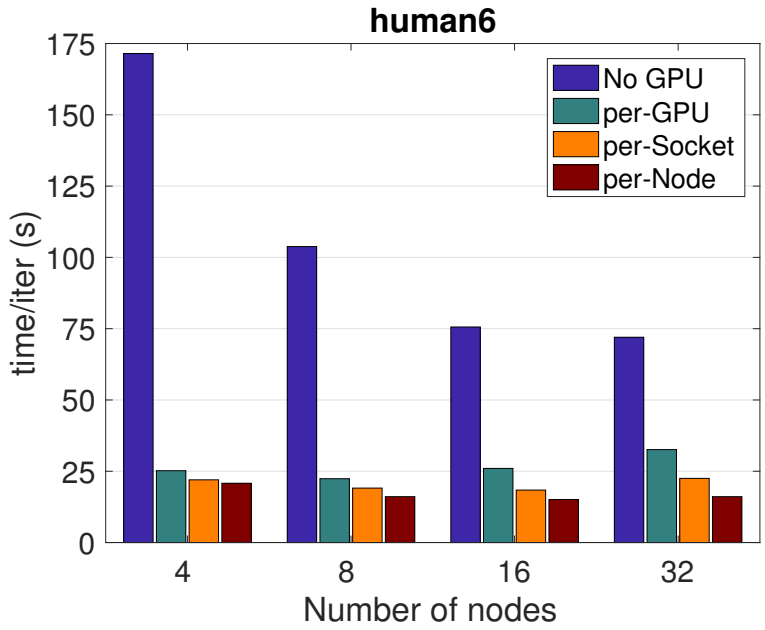
Tsubame-3 at Tokyo Tech



- each process with multiple GPUs to lower inter-node communication by reducing number of processes
- use NVLink for data transfer among local GPUs

BiCGStab performance with multiple GPUs per process





- on large number of nodes, inter-GPU comm may be reduced by multi-GPU implementation with careful communication scheme

hiding communication on GPU cluster

	100ts			338ts			hum4		
n_g	block	pipe1	pipe2	block	pipe1	pipe2	block	pipe1	pipe2
1	18.4	19.2	19.3	−−	−−	−−	−−	−−	−−
4	10.2	10.0	9.4	34.6	33.8	31.6	34.3	32.1	30.4
8	9.1	8.9	7.6	31.5	30.1	25.6	30.5	28.1	24.3
16	8.8	8.6	6.7	30.2	28.6	22.0	30.1	28.0	21.1
32	10.3	10.1	6.8	31.7	30.8	21.8	30.9	28.3	19.7

- hide `all-gatherv` for `HiMV` behind vector operations
- `pipe1` aims to hide CPU-GPU vector copy
- `pipe1` aims to hide MPI communication

  ▷ upto 1.57× speedup

Conclusion

- used MAGMA's variable-size batched kernel to utilized GPUs on node
- considered underlying hardware to improve inter-GPU communication
- more details in IPDPS'18 paper
  - ▷ some GPU-aware MPI results on Reedubush-H

Current Work

- GPU acceleration of other parts
  - ▷ generation/compression of the matrix?
- scalability improvement
  - ▷ load balancing?
- factorization-based solver

Thank you!!