High-Performance GMRES Mixed-Precision (HPG-MxP) Benchmark with Kokkos-Kernels backend

EXASCALE COMPUTING PROJECT

Ichitaro Yamazaki, Jennifer Loe, Christian Glusa, Siva Rajamanickam (Sandia National Labs)

Piotr Luszczek, and Jack Dongarra (University of Tennessee, Knoxville)

Kokkos User Group (KUG) Meeting

Albuquerque, New Mexico, 2023-12-12

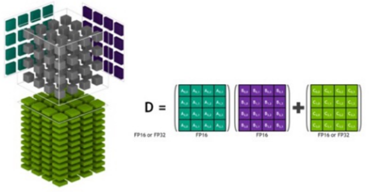
National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

Goal & Motivations

- HPG-MxP is designed to
  - capture typical performance of **“real” applications**
    - Consist of kernels found commonly in these applications
  - allow the use of **mixed precision arithmetic**
  - It’s a mix of HPCG and HPL-MxP
- Some current & emerging HP computers can perform lower precision arithmetic at higher performance
  - Some emerging accelerators may not support double precision
- Lower precision reduces the data transfer volume and may improve application performance
  - Application performance is often limited by communication (latency or bandwidth)
- Such benchmark may be of interests to many decilplines

		GPU Peak Performance (Tflop/s)		
System	GPU	FP64	FP32	FP16
Frontier (ORNL)	AMD MI250X	26.5	26.5	191.0
Fugaku (Riken)	Fujitsu A64 FX	3.4	6.7	13.5
Summit (ORNL)	NVIDIA V100	7.5	19.5	N/A
Perlmutter (NERSC)	NVIIDIA A100	9.7	19.5	312.0
Sierra (LLNL)	AMD MI100	11.5	23.1	184.0
Selena (NVIDIA)	AMD MI250X	26.5	26.5	191.0



NVIDIA Sparse Tensor Core

HP GMRES Mixed-Precision (HPG-MxP)

Input: $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^{n\times 1}$, initial guess $x_0 \in R^{n\times 1}$, relative residual tolerance $rTol$
Output: approximate solution x_m

```
r = b − Ax,
γ = ‖r‖₂
while not converged do
    v₁ = r₀/γ, and h₁,₁ = 0
    for j = 1 : m do
        // GMG preconditioner M, followed by SpMV
        w_j = A M v_j
        // CGS2 orthogonalization
        w_j = w_j − V_j t_j with t_j = V_j^T w_j
        h_{1:j,j} = t_j
        w_j = w_j − V_j t_j with t_j = V_j^T w_j
        h_{1:j,j} = h_{1:j,j} + t_j
        h_{j+1,j} = ‖w_j‖₂
        v_{j+1} = w_j / h_{j+1,j}
    end for
    d̂ = arg min_{y∈ℝ^m} ‖γe₁ − H_{1:m+1,1:m} y‖₂
    x = x + V_m d̂
    r = b − Ax
    γ = ‖r‖₂
end while
```

GMRES in lower precision

Refinement in double precision

	Dense Problem Compute Intensive	Sparse Problem Compute/Comm pattern in "Real" Appls
Uniform Precision	HPL	HPCG
Mixed Precision	HPL-MxP	HPG-MxP

- Mixed-precision Iterative refinement to solve a sparse linear system
  - Lower-precision may be used to solve the sparse linear system
    - GMRES to provide the robustness
      - GMG preconditioner + GS smoother
    - Typically dominates benchmark time
  - Iterative refinement to obtain the double-precision solution
  - How much speedup can we get using lower-precision for communication-bound operations?
  - Benchmark result is penalized if lower-precision increases the iteration count

Mixed-precision GMRES – Iterative Refinement

for solving sparse non-symmetric linear system

- Generalized Minimum Residual (GMRES)
  - A popular Krylov method for solving a non-symmetric system
  - It computes an approximate solution minimizes the residual norm in the computed Krylov projection subspace
- Mixed-precision variant
  - is also a well-established algorithm
  - Growing interests, with lots of numerical theories and performance studies, in recent years

1) *P. Amestoy, A. Buttari, N. Higham, J. L'Excellent, T. Mary, and B. Vieuble. Five-precision GMRES- based iterative refinement. 2021.*

2) *P. Amestoy, A. Buttari, N. Higham, J. L'Excellent, T. Mary, and B. Vieuble. Combining sparse approximate factorizations with mixed precision iterative refinement. Technical report, The University of Manchester, 2022.*

3) *E. Carson and N. Higham. Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions. SIAM J. Sci. Comput., 40(2):A817–A847, 2018.*

4) *S. Gratton, E. Simon, D. Titley-Pe´loquin, and P. Toint. Exploiting variable precision in GMRES. ArXiv, abs/1907.10550, 2019*

5) *N. Lindquist, P. Luszczek, and J. Dongarra. Improving the Performance of the GMRES Method using Mixed-Precision Techniques. in Smoky Mountains Conference Proceedings, 2020.*

6) *J. Loe, C. A. Glusa, I. Yamazaki, E. G. Boman, and S. Rajaman- ickam. Experimental evaluation of multiprecision strategies for GMRES on gpus. In 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pages 469–478, 2021.*

7) ***K. Turner and H. Walker. Efficient high accuracy solutions with GMRES(m). SIAM J. Sci. Stat. Comput., 13(3):815–825, 1992.***

8) *Etc. etc.*

Also, mixed-precigion MG:

1) *S. McCormick, J. Benzaken, and R. Tamstorf. Algebraic error analysis for mixed-precision multigrid solvers. SIAM J. Sci. Comp., 43(5):S392–S419, 2021.*

HPG-MxP: Main tasks

Same kernels as HPCG, except for CGS2

1. Sparse Matrix Vector Multiply (SpMV)
   - **Point-to-point neighborhood communication** (halo exchange)
     - Exchange 1, n_x, or n_x^2 elements with 7 ~ 26 neighbors
   - Local **SpMV** with 27-pts stencil
     - 54nm Flops / restart
2. Orthogonalization based on Classical Gram Schmidt with reorthogonalization (CGS2)
   - **Blas-2 dense matrix-vector dot-product**, local atomic and **global reduce**
     - Total of 2n(1+m)m Flops / Restart
   - **Blas-2 dense matrix-vector update**, embarrassingly parallel
     - Total of 2n(1+m)m Flops / Restart

Mixture of sparse and dense operations, commonly found in real applications

- With m = 40, about same number of flops for GMG and CGS2

3. Geometric Multi Grid (GMG)
   - Four level with 2^3x smaller coarse grid
   - One forward-sweep of Gauss-Seidel (GS) as pre & post smoother
     - **Halo-exchange**, Local **SpTRSV**
     - Total of 2*(54*73)/64 nm Flops / Restart
   - Residual vector computation
     - **Halo-exchange**, Local **SpMV**
     - Total of 2*(54*73)/64 nm Flops / Restart
   - Restriction & Prolongation operators
     - No communication, Local **SpMV** with a rectangular matrix,
       e.g., one nonzero per row
   - One forward sweep of GS at the final coarse level.
     - **Halo-exchange**, Local **SpTRSV**
     - Total 81 / 512 nm Flops / Restart

HPG-MxP reference implementation

- The reference implementation (solver & benchmark suite) is available
  - https://github.com/iyamazaki/hpcg
  - It is meant to be optimized by participants
- It reuses many of HPCG components
- It is based on C++ template
  - To make it easier to use various precision

```
template<class SparseMatrix_type, class SparseMatrix_type2, class CGData_type, class CGData_type2, class Vector_type>
int GMRES_IR(const SparseMatrix_type & A, const SparseMatrix_type2 & A_lo,
             CGData_type & data, CGData_type2 & data_lo, const Vector_type & b_hi, Vector_type & x_hi,
             const int restart_length, const int max_iter, const typename SparseMatrix_type::scalar_type tolerance,
             int & niters, typename SparseMatrix_type::scalar_type & normr, typename SparseMatrix_type::scalar_type & normr0,
             double * times, bool doPreconditioning);
```

- It also provides CUDA/HIP backends
  - It uses GPUs to generate basis vectors,
    while the tiny least square problem is solved redundantly on each CPU.
  - It uses MPI for data exchange, while solely rely on vendor libraries for the GPU computation
    - **CuBLAS** and **CuSparse** on NVIDIA and **RocBLAS** and **RocSparse** on AMD
    - GS uses general SpMV & SpTRSV
  - Minimum CUDA/HIP code
    - Gather for MPI P2P communication
    - If the vector needs to be casted, then it is done on a CPU (no mixed-precision kernels)

Performance studies of reference implementation : Experimental setups to motivate Kokkos-Kernels backend

- Test machines
  - Summit (OLCF)
    - Each node with 2×22-core Power9 CPUs and six NVIDIA V100 GPUs
  - Frontier (OLCF)
    - Each node with 1×64-core AMD EPYC CPUs and four AMD MI250X GPUs
  - Perlmutter (NERSC)
    - Each node with 1×64-core AMD EPYC CPUs and four NVIDIA A100 GPUs
- Weak-scaling
  - a fixed problem size per MPI (one MPI / CPU core or GPU)
- Using single-precision for GMRES iterations
  - 2.0x reduction in dense matrix storage
  - 1.6x reduction in sparse matrix storage
- Performance of the reference implementation
  - Meant to motivate interests

name	value
Solver parameters	
restart cycle, m	30
GMG levels	3
GS sweeps	1
Step 1 (Validation)	
problem size (n_x, n_y, n_z)	(80,80,80)
convergence tol	10^{-9}
# of MPI procs	4
Step 2 (Benchmark)	
# of iterations	300
# of minimum solves	10
minimum time	30 minutes (disabled)

Some of the parameter values are selected for convenience.

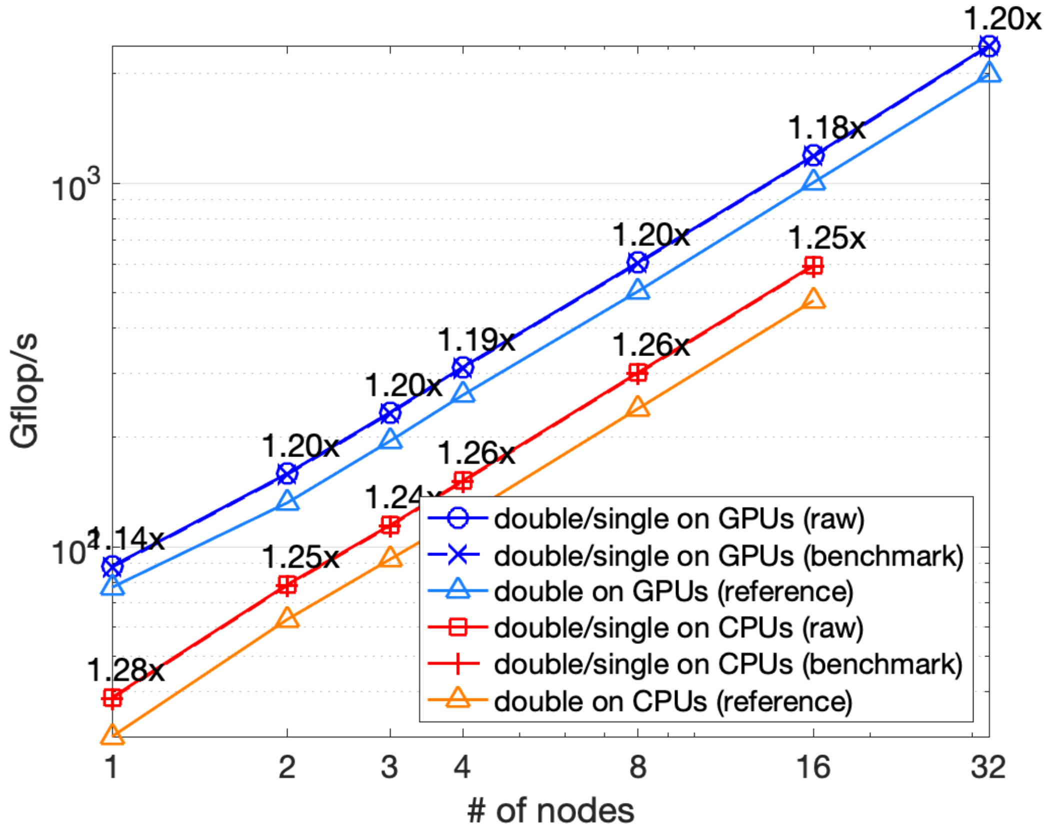
EXASCALE COMPUTING PROJECT

Performance of reference implementation on Summit
IBM Power9 CPUs + NVIDIA V100 GPUs

- Speedup of 1.2x using a non-optimized reference
- Most of the solver time is spent in SpTRSV for GMG
  - It has limited parallelism, and
    its performance may be more dominated by latency
  - it is harder to get speedup using lower precision
  - Reference implementation uses CuSparse SpMV & SpTRSV
    (no coloring)

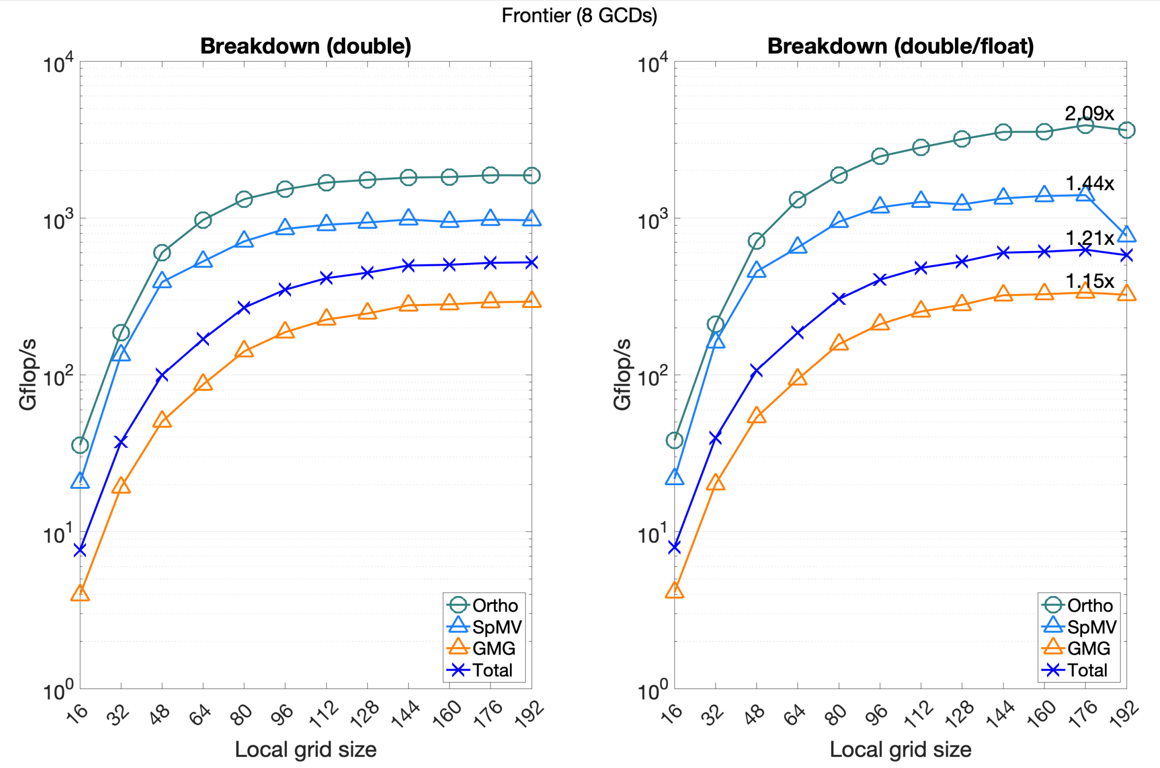
	Time in seconds with GPUs				TFlop/s with GPUs			
	GMG	SpMV	CGS2	**Total**	**GMG**	SpMV	CGS2	**Total**
Uniform	**51.5**	3.8	2.5	**60.2**	**0.30**	1.20	4.13	**0.50**
Mixed	**44.5**	2.4	1.8	**50.1**	**0.35**	1.87	5.73	**0.61**
Speedup	**1.16**	1.56	1.39	**1.20**	**1.15**	1.56	1.39	**1.20**

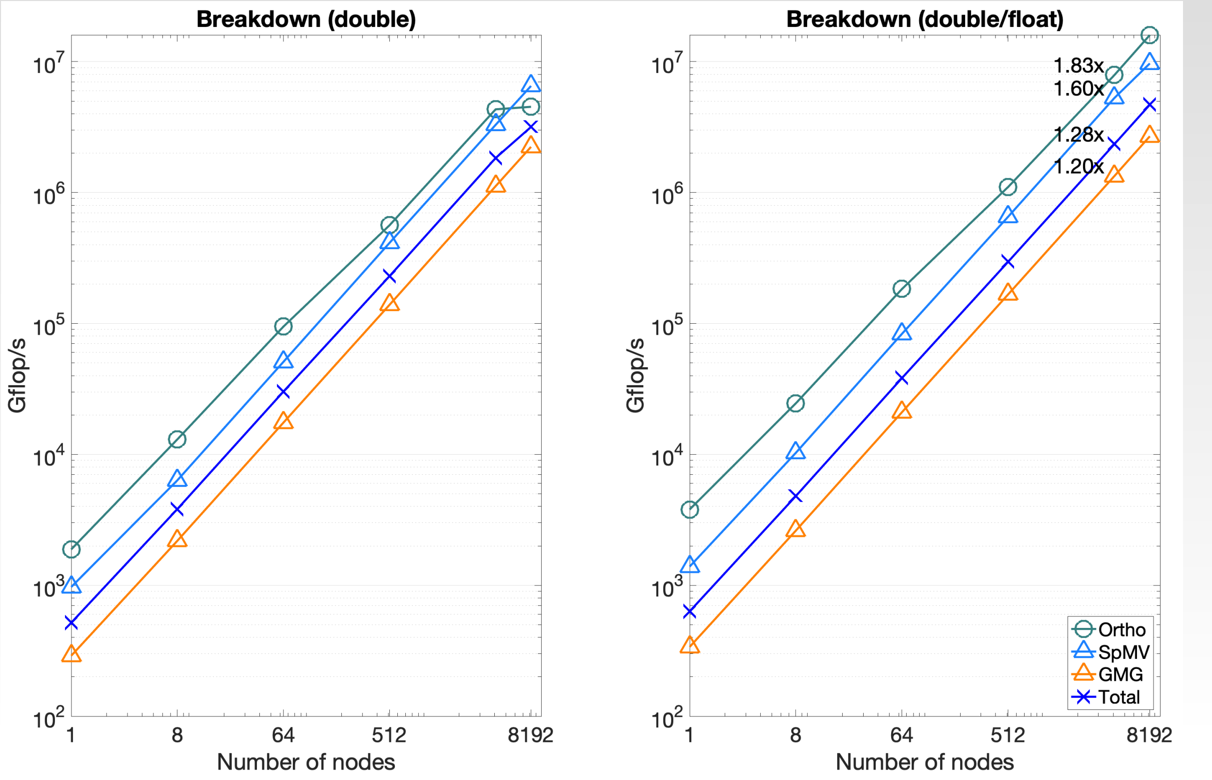
Performance on 8 Summit nodes with GPUs
(about same total # of flops for GMG or CGS2)



EXASCALE COMPUTING PROJECT

Performance of reference implementation on Frontier (at larger scale)
AMD EPYC CPUs + AMD MI250X GPUs





- Speedups, similar to those on Summit
  - AMD MI250X GPU on Frontier has same peak compute performance using double and single
    - Peak 530-1600 Gflop/s for Ortho and 450-1100 Gflop/s for SpMV (800 GB/s)
    - 14 PFlop/s HPCG on Frontier
  - SpMV (with P2P) seems to scale better than Ortho (all-reduce)

EXASCALE COMPUTING PROJECT

HPG-MxP with Kokkos-Kernels backend (as an "optimized" implementation)

- Kokkos-Kernels backends
  - Optimized version of GS (e.g., coloring) [Devici'16, Kelley'22]
    - TPL to vendor BLAS/Sparse kernels
    - Reference spends most of time in GS
    - KK GS may get closer to SpMV on big enough matrix?
  - Mixed-precision operations, including FP16 or BF16
    - Reference explicitly type-casts (on host) before uniform-precision kernels
  - Portable implementation of LA ops on different node architectures
    - FP16/BF16 supports on host.

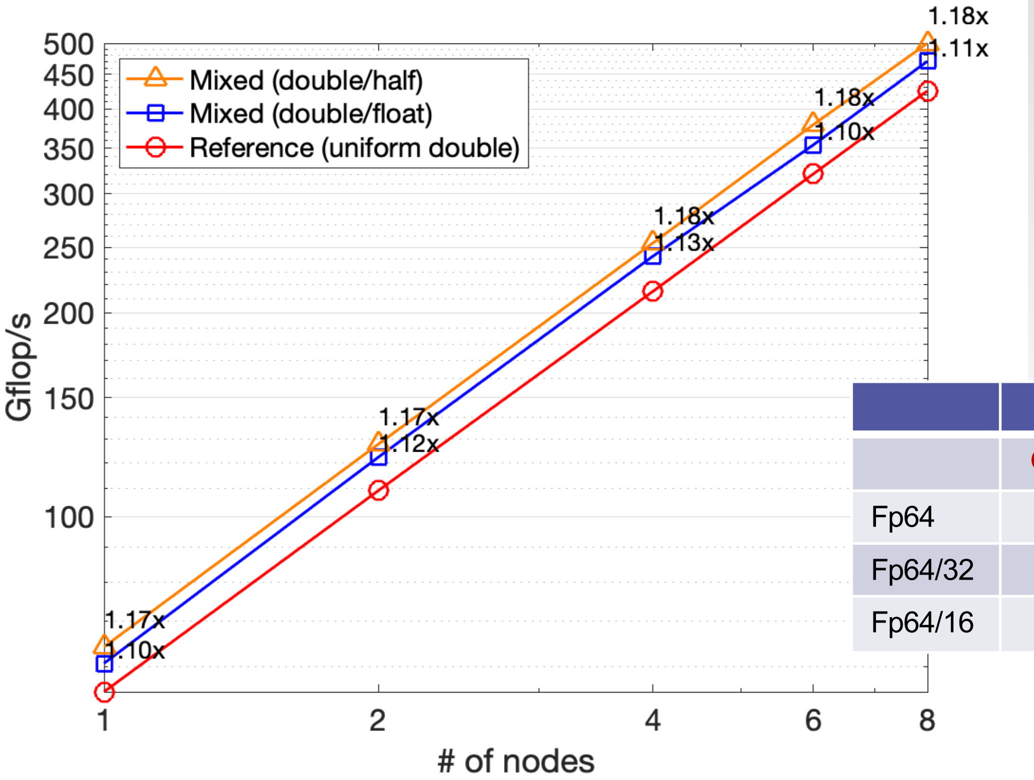
```
template <class ExecutionSpace, class AViewType, class XViewType,
          class YViewType>
void gemv(const ExecutionSpace& space, const char trans[],
          typename AViewType::const_value_type& alpha, const AViewType& A,
          const XViewType& x, typename YViewType::const_value_type& beta,
          const YViewType& y) {
```

		GPU Peak Performance (Tflop/s)		
System	GPU	FP64	FP32	FP16
Crusher (ORNL)	AMD MI250X	26.5	26.5	191.0
Fugaku (Riken)	Fujitsu A64 FX	3.4	6.7	13.5
Perlmutter (NERSC)	NVIIDIA A100	9.7	19.5	312.0
Sierra (LLNL)	AMD MI100	11.5	23.1	184.0
Selena (NVIDIA)	AMD MI250X	26.5	26.5	191.0

	fp64	fp32	fp16	bf16
maximum	$1.80 \cdot 10^{308}$	$3.40 \cdot 10^{38}$	$6.55 \cdot 10^{4}$	$3.39 \cdot 10^{38}$
minimum	$2.22 \cdot 10^{-308}$	$1.17 \cdot 10^{-38}$	$5.96 \cdot 10^{-8}$	$1.18 \cdot 10^{-38}$
epsilon	$2.22 \cdot 10^{-16}$	$1.19 \cdot 10^{-7}$	$9.77 \cdot 10^{-4}$	$7.81 \cdot 10^{-3}$

Special thanks to Kokkos-Kernels team,
Brian Kelley, Evan Harvey, and Vinh Dang

Performance (time / iteration) using FP16 on Perlmutter (Four NVIDIA A100 GPUs / node) using mixed-precision FP16/FP32 KK interface (FP16 for GMRES iterations, but accumulations are stored in FP32)



- using FP16 did help reducing the iteration time
  - Performance & speedup still dominated by GS
  - We used clustered GS from Kokkos-Kernels
  - Iteration count went up, and benchmark results were 25x slower using FP16

	Time in milliseconds / iter				GFlop/s				Speedup			
	GMG	SpMV	CGS	Total	GMG	SpMV	CGS	Total	GMG	SpMV	CGS	Total
Fp64	8.80	0.67	3.88	13.4	42.3	162.1	64.9	55.1	--	--	--	--
Fp64/32	8.11	0.47	3.55	12.1	45.8	232.6	71.0	60.7	1.08	1.43	1.09	1.10
Fp64/16	8.02	0.40	3.04	11.5	46.4	274.4	82.7	64.2	1.10	1.69	1.27	1.17

Performance on one Perlmutter node with GPUs (about same total # of flops for GMG or CGS2)

Final remarks

- Kokkos-Kernels backend for HPG-MxP
  - Optimized GS, mixed-precision interface, and portable kernels
  - Looking at performance on current HPC system (Frontier, Perlmutter, and maybe Aurora)
    - GS performance and speedups
    - Potential of BF16/FP16 precision
      - Store just matrices in FP16 for GS

Thank you !!

EXASCALE COMPUTING PROJECT

Acknowledgments

- This work was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. Sandia National Laboratories is a multimssion laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

Discussion