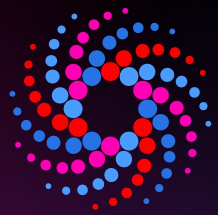


SC22
Dallas, TX | hpc accelerates.

High-Performance GMRES Multi-Precision Benchmark

Design, Performance, and Challenges

Ichitaro Yamazaki, Jennifer Loe, Christian Glusa, Siva Rajamanickam (Sandia National Labs)
Piotr Luszczek, and Jack Dongarra (University of Tennessee, Knoxville)

13th IEEE International Workshop on
Performance Modeling, Benchmarking, and Simulation of High Performance Computer Systems (PMBS)

Goal & Motivations

- New benchmark is designed to
  - capture typical performance of “real” applications
  - allow the use of mixed precision arithmetic
- Some current & emerging HP computers provide higher performance for lower precision arithmetic
  - 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)

		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

Goal & Motivations

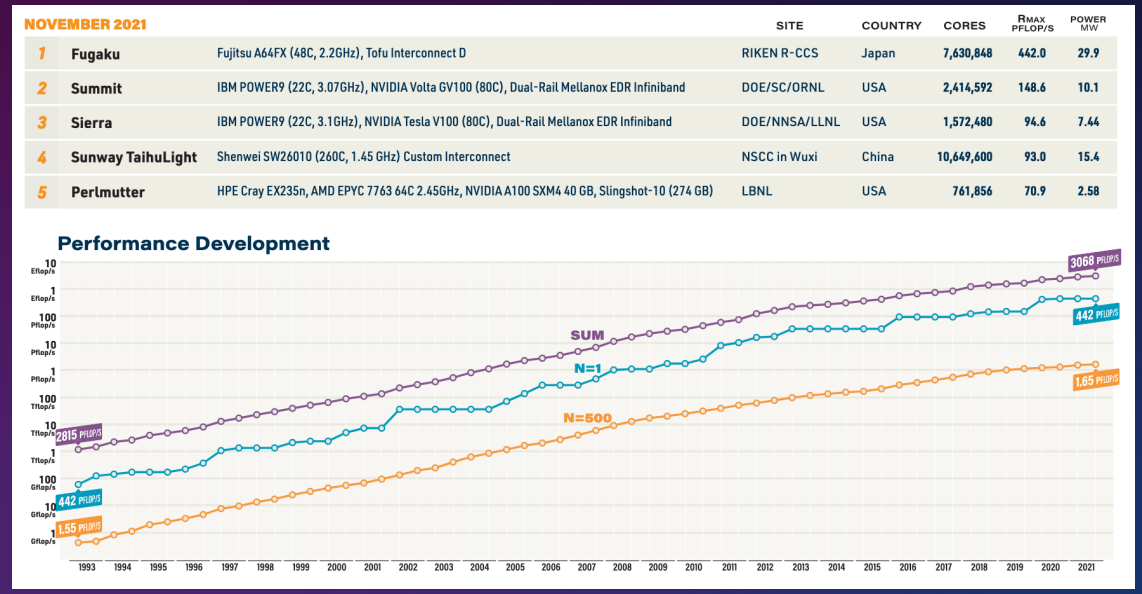
- Growing interests in utilizing lower-precision for “real” applications
  - ECP xSDK multi-precision project funded by US DOE
- New benchmark could have wide impacts
  - Capture the computers capabilities for applications by allowing mixed-precision operations
    - Algorithmic & software efforts to utilize lower-precision
  - Motivate hardware vendors to design future HP computers that can obtain high application performance, with mixed-precision arithmetic

		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

Related HP Benchmark 1/3 : High Performance Linpack (HPL)

- HPL measures performance of solving dense linear system in double precision
  - It is based on exact dense LU factorization
- Its performance is dominated by dense matrix-matrix multiply, with a proper implementation.
- Its performance is close to the double-precision **peak compute** performance of the target machine.
- It is used to rank HP computers for Top500 list, providing historical data
- It is also used to stress-test new systems

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



NOVEMBER 2021			SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	Fugaku	Fujitsu A64FX (48C, 2.2GHz), Tofu Interconnect D	RIKEN R-CCS	Japan	7,630,848	442.0	29.9
2	Summit	IBM POWER9 (22C, 3.07GHz), NVIDIA Volta GV100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/SC/ORNL	USA	2,414,592	148.6	10.1
3	Sierra	IBM POWER9 (22C, 3.1GHz), NVIDIA Tesla V100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/NNSA/LLNL	USA	1,572,480	94.6	7.44
4	Sunway TaihuLight	Shenwei SW26010 (260C, 1.45 GHz) Custom Interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
5	Perlmutter	HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10 (274 GB)	LBNL	USA	761,856	70.9	2.58

Image source : top500.org

Related HP Benchmark 2/3 : High Performance Conjugate Gradient (HPCG)

- HPCG is designed to reflect the **application performance**
- It solves a sparse linear system using CG with GMG preconditioner (GS smoother).
  - Composed of computation & communication tasks common in real applications
  - Its performance is more limited by communication latency or bandwidth.
- It is meant to motivate the future HP computers that can achieve high application performance

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

HPCG Rank	HPL Rank	System	HPCG (Pflop/s)	HPL (Pflop/s)
1	1	Fugaku (Riken)	16.0	442.0
2	2	Summit (ORNL)	2.9	148.6
3	5	Perlmutter (NERSC)	1.9	70.8
4	3	Sierra (LLNL)	1.8	94.6
5	6	Selena (NVIDIA)	1.6	63.5

Data source : top500.org (Nov, 2021)

Related HP Benchmark 3/3 : HPL - Accelerator Introspection (HPL-AI)

- HPL-AI solves the same **dense** linear system as HPL, but allows the use of **lower-precision**
- It uses lower-precision for the **compute-intensive** LU factorization (no pivoting), which dominated benchmark time.
- **Iterative refinement** is used to obtain the solution with double precision accuracy.
- It achieves much higher performance than HPL on machines that provide lower-precision arithmetic at higher performance
  - It measures the computer's capability to perform compute intensive tasks

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

HPCG Rank	HPL Rank	System	HPL-AI (Pflop/s)	HPL (Pflop/s)
1	1	Fugaku (Riken)	2.00	0.44
2	2	Summit (ORNL)	1.41	0.15
3	6	Selena (NVIDIA)	0.63	0.06
3	5	Perlmutter (NERSC)	0.59	0.07
5	8	Juwels BM (FZJ)	0.47	0.04

Data source : top500.org (Nov, 2021)

New HP Benchmark : HP GMRES mixed-precision (HPGMP)

- The new benchmark
  - performs computation & communication common in real applications (like HPCG)
  - allows lower-precision (like HPL-AI)
- Iterative refinement for solving a sparse linear system
  - Lower-precision may be used to solve the linear system
    - Sparse iterative solver, which typically dominates benchmark time
  - Double-precision is used to update the solution and to compute the new residual vector
- We think this new benchmark could have wide impacts
  - As discussed previously in moov

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

Initialization for solving $Ax = b$:
$x = 0$ and $r = b - Ax$
While not converged
1. Approximately solve (potentially in mixed lower precision)
   $Ae = r$
   for correction $e \approx A^{-1}r$
2. Update approximate solution (in double precision)
   $x = x + e$ and $r = b - Ax$

Majority of time spent in Step 1.

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.

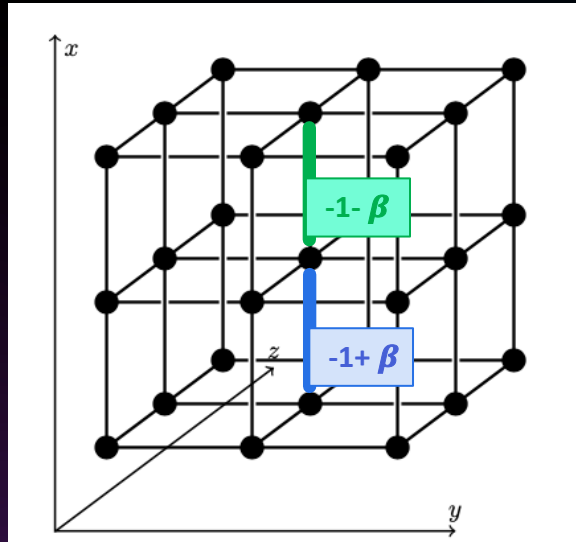
8) *Etc. etc.*

Also, mixed-precigion MG:

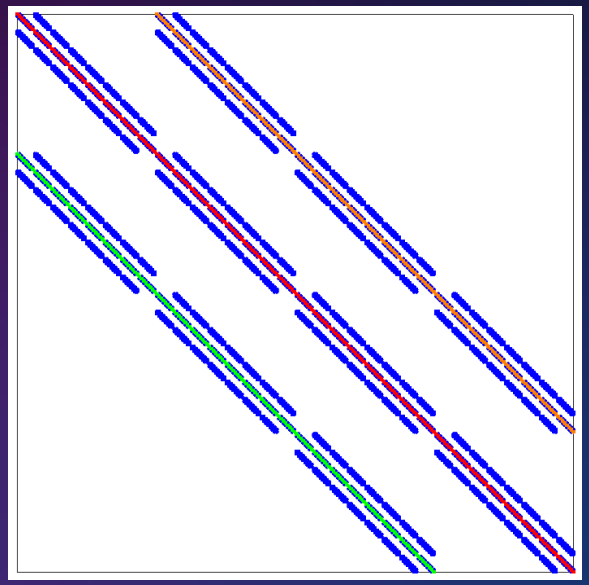
1) *S. McCormick, J. Benzaken, and R. Tamstorf. Algebraic error analysis for mixed-precision multigrid solvers. SIAM J. Sci. Comp., 43(5):S392–S419, 2021.*

HPGMP: Problem description

- Regular 3D grid with 27-points stencil
  - Same as in HPCG
  - Parameterized for non-symmetric numerical values
    - Represents finite difference discretization of an advection-diffusion problem
  - Right-hand-side vector $b = A*ones$, and initial approximate solution x=zeros.
- MPI processes are arranged into a 3D process grid (p_x, p_y, p_z)
- Participant specifies the dimension of the local subdomain (n_x, n_y, n_z) on each MPI
  - the global matrix size is ($n_x p_x$, $n_y p_y$, $n_z p_z$)



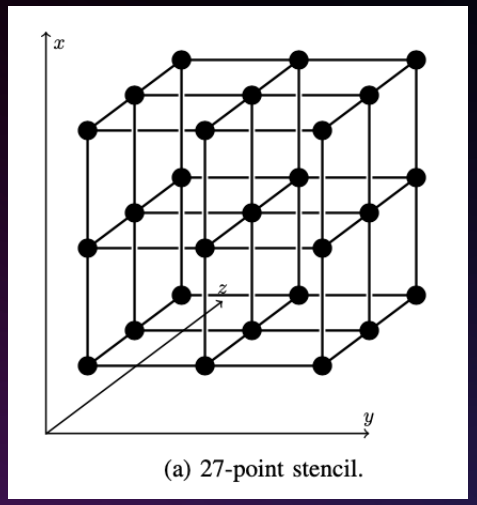
(a) 27-point stencil.



HPGMP: Main task 1/3

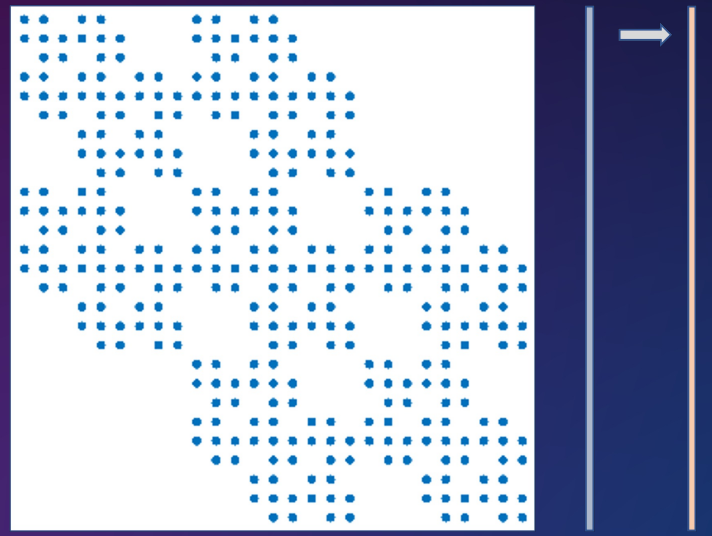
for GMRES, potentially in lower/mixed precision

Same components as HPCG, except for CGS2



(a) 27-point stencil.

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



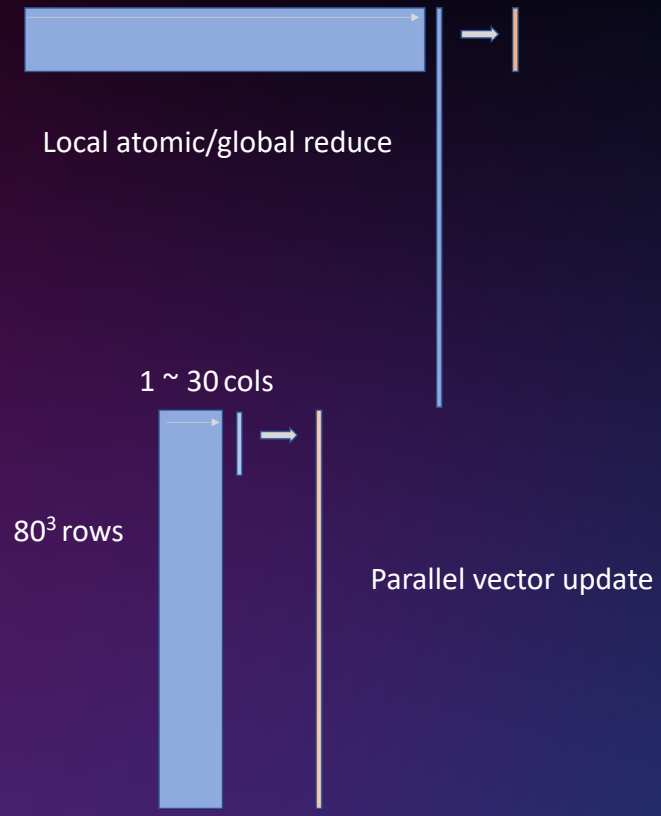
Irregular access to input vector entries, but update output vector entries in parallel

HPGMP: Main task 2/3

for GMRES, potentially in lower/mixed precision

Same components as HPCG, except for CGS2

2. Orthogonalization based on Classical Gram Schmidt with reorthogonalization (CGS2) based on
   - **Blas-2 matrix-vector multiply (GEMV)**
     - **dot-product**, local **atomic** multiply-add, followed by **global reduce**
       - Total of 2n(1+m)m Flops / Restart
     - **vector update**, embarrassingly parallel
       - Total of 2n(1+m)m Flops / Restart
   - **Blas-1 vector operation**
     - **Vector norm & Vector scale**

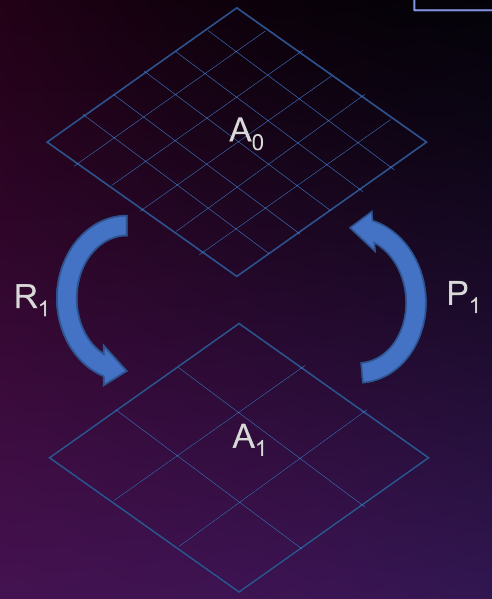


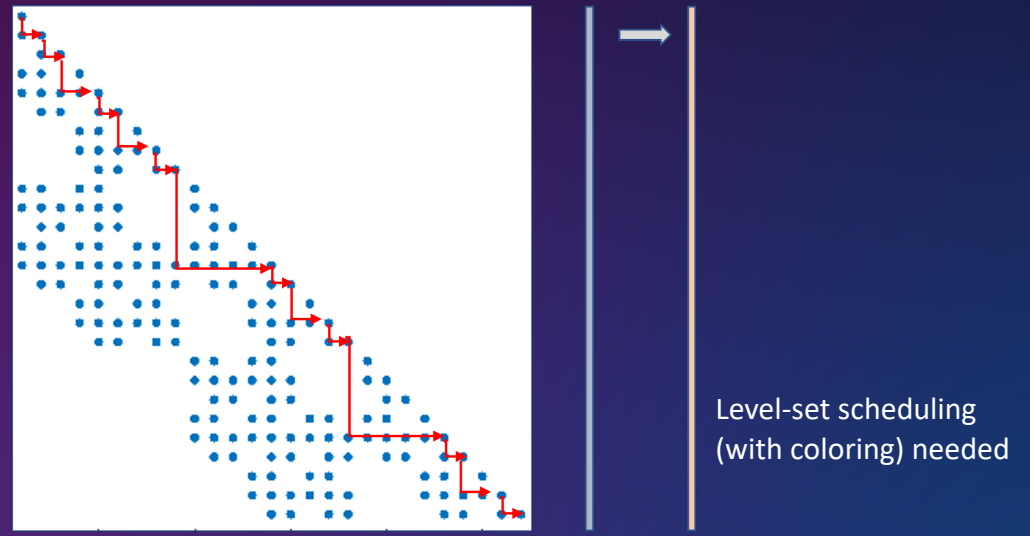
HPGMP: Main task 3/3

for GMRES, potentially in lower/mixed precision

Same components as HPCG, except for CGS2

3. Geometric Multi Grid (GMG)
   - 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





Level-set scheduling (with coloring) needed

HPGMP: Other tasks
for iterative refinement

- A small least-square problem needs to be solved
  (31-by-30 on CPU, redundantly by all MPIs)
- Approximate solution needs to be updated in double precision (BLAS-1 AXPY)
  - New residual vector needs to be also computed in double precision (SpMV)

HPGMP: Main tasks

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
   reorthogonalization (CGS
   - **Blas-2 dense vector dot**
     - Total of 2n(1+m)m Flops / Restart
   - **Blas-2 vector update**, embarrassingly parallel
     - Total of 2n(1+m)m Flops / Restart

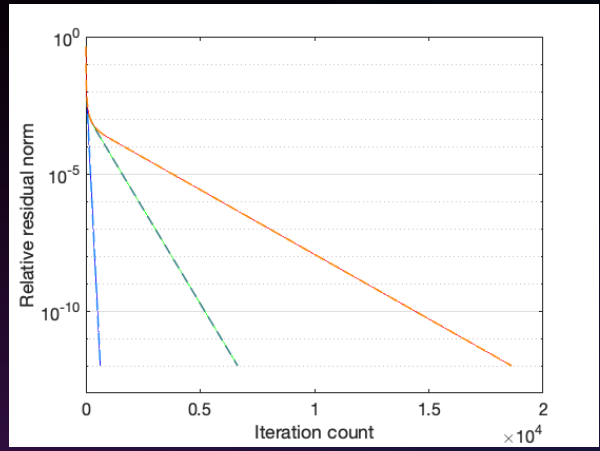
Mixture of sparse and dense operations, commonly found in real applications

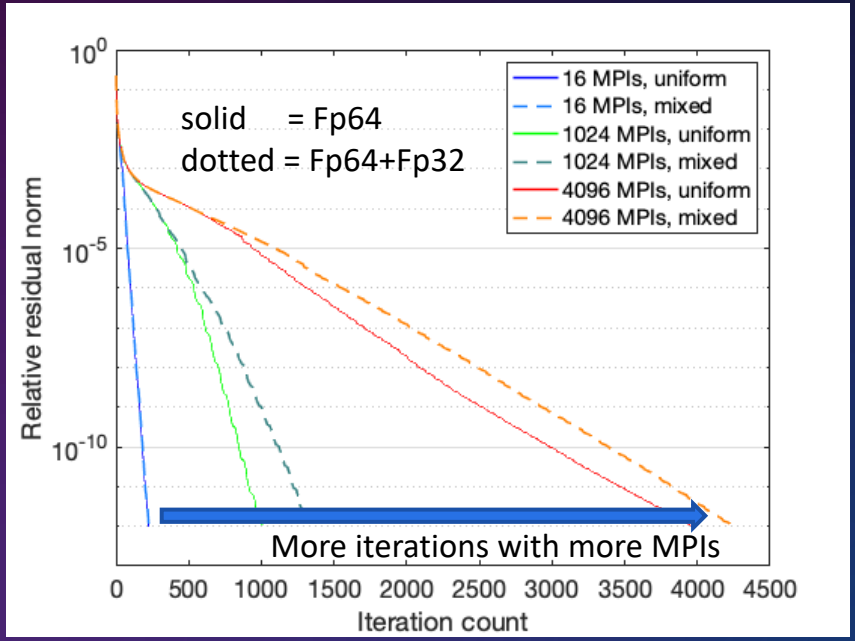
- With m = 30, about same number of flops for GMG and CGS2

3. Geometric Multi Grid (GMG)
   - 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
   - Prolongation operators
     - unication, Local **SpMV** with a
       ar 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

Q1 : How much flexibility to allow in term of precision?

- Current benchmark specification allows any precision for the GMRES iteration.
- The benchmark is **not** meant to be a **robust** or **scalable** solver.
  - It is designed to capture application performance
  - Iteration count increases with MPI count
  - It may converge slower, or faster, using lower precision
- We need verification and validation!!
  - The solver should achieve the double precision accuracy
  - If the solver requires more iterations, then the benchmark results should be appropriately penalized





HPGMP benchmark : two steps

1. **Verification step**:
   - Run both reference & optimized solver
     - Using a fixed problem size on a fixed # of MPI processes
     - To reach double-precision accuracy
   - Record # of iteration needed by both
     - Failure if optimized code did not converge
     - Compute penalty factor
       i_p = min(1.0, # of optimized iterations / # of reference iteration)
2. **Benchmark step**:
   - Run optimized solver for a fixed number of iterations
     - Using user-specified problem size and # of MPI processes
     - Until reaching a minimum # of solves or time
   - Compute benchmark Gflop/s
     - **I_p x (# of Gflops / Optimized benchmark time)**

- The benchmark is designed
  - To allow the use of mixed-precision
  - To penalize if the lower-precision results in the loss of accuracy (convergence rate
- Note: we cannot run to double precision accuracy for the large benchmark runs (iteration count increases with # of MPIs)

HPGMP 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
- 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** for CGS2, **CuSparse** SpMV & SpTRSV for GS and restriction/prolongation, and
      CUDA library for memory management
  - No custom CUDA/HIP code
    - MPI message communication is through CPUs
    - If the vector needs to be casted, then it is done on a CPU

```
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);
```

HPGMP: Allowed optimizations

Similar to HPCG

- Hardware specific optimization are allowed
  - Data structures, communication schemes, etc.
- Matrix may be permuted for GS smoother to expose parallelism
  - If the permutation increases the iteration count, benchmark performance is penalized (validation step)
- Algorithm changes are **not** allowed
  - *s*-step (communication-avoiding), pipelined, or randomized variant of GMRES
  - Low-synchronous/single-reduce orthogonalization
  - Iterative-variant of GS smoother
- Knowledge of matrix structure **cannot** be used
  - The matrix should be treated as a general matrix for SpMV
- **Any precision(s) may be used for the sparse solver**
  - **Need to pass the verification, and will be penalized on any increase in iteration count**
- Matrix scaling is **not** allowed
  - The matrix may not be scaled to fit in the numerical range of lower precision
  - It can be used to improve the conditioning of the matrix

Performance studies of reference implementation : Experimental setups

- OLCF machines
  - Summit
    - Each node with 2×22-core Power9 CPUs and six **NVIDIA V100** GPUs
  - Spock
    - Each node with 1×64-core **AMD EPYC** 7662 CPU and four **AMD MI100** GPUs
  - Crusher
    - Each node with 1×64-core **AMD EPYC** 7A53 CPU and four **AMD MI250X** GPUs
- Weak-scaling
  - a fixed problem size per MPI (one MPI / CPU core or GPU)
- Using single-precision for GMRES iterations
  - 1.6x reduction in sparse matrix storage
- Performance of the **reference** implementation
  - Meant to motivate interests

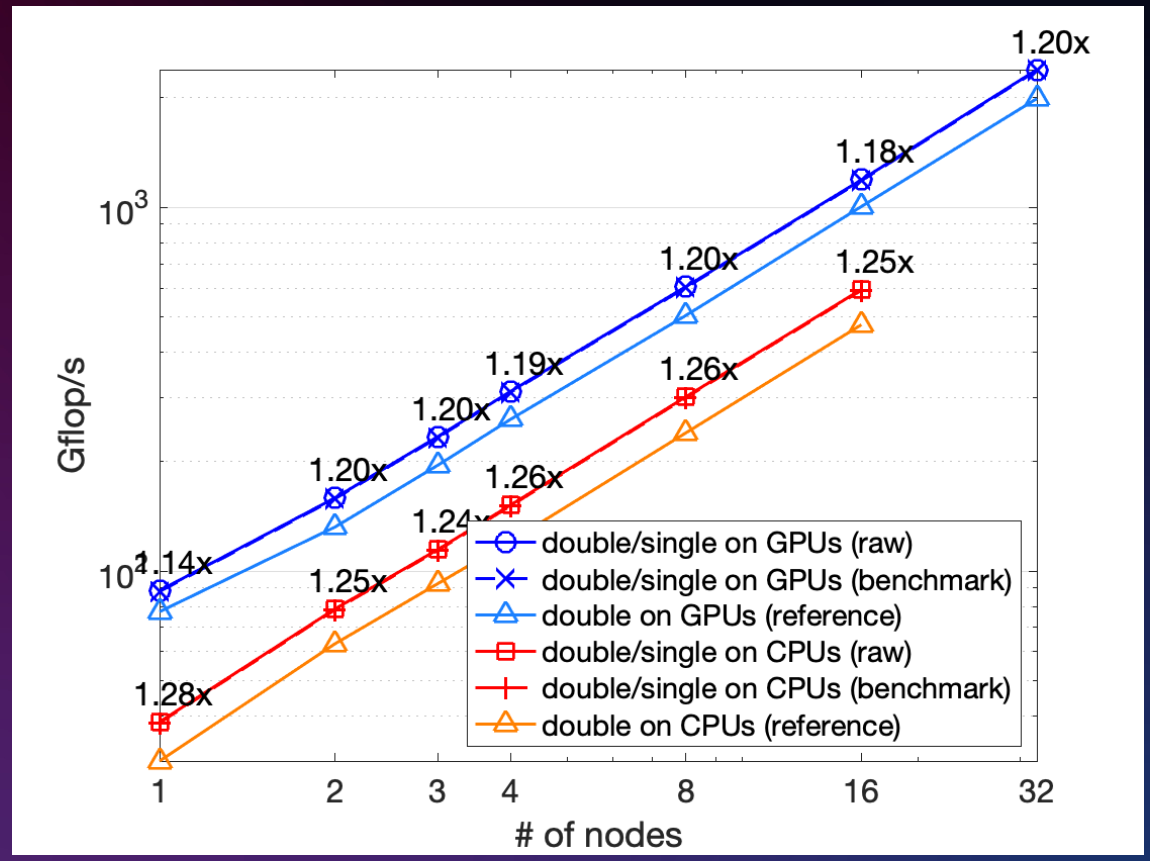
name	value
Solver parameters	
restart cycle, m	30
GMG levels	3
GS sweeps	1
Step 1 (Validation)	
problem size (n_x, n_y, n_z)	(80,80,80)
convergence tol	10^{-9}
# of MPI procs	4
Step 2 (Benchmark)	
# of iterations	300
# of minimum solves	10
minimum time	30 minutes (disabled)

Some of the parameter values are selected for convenience.

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

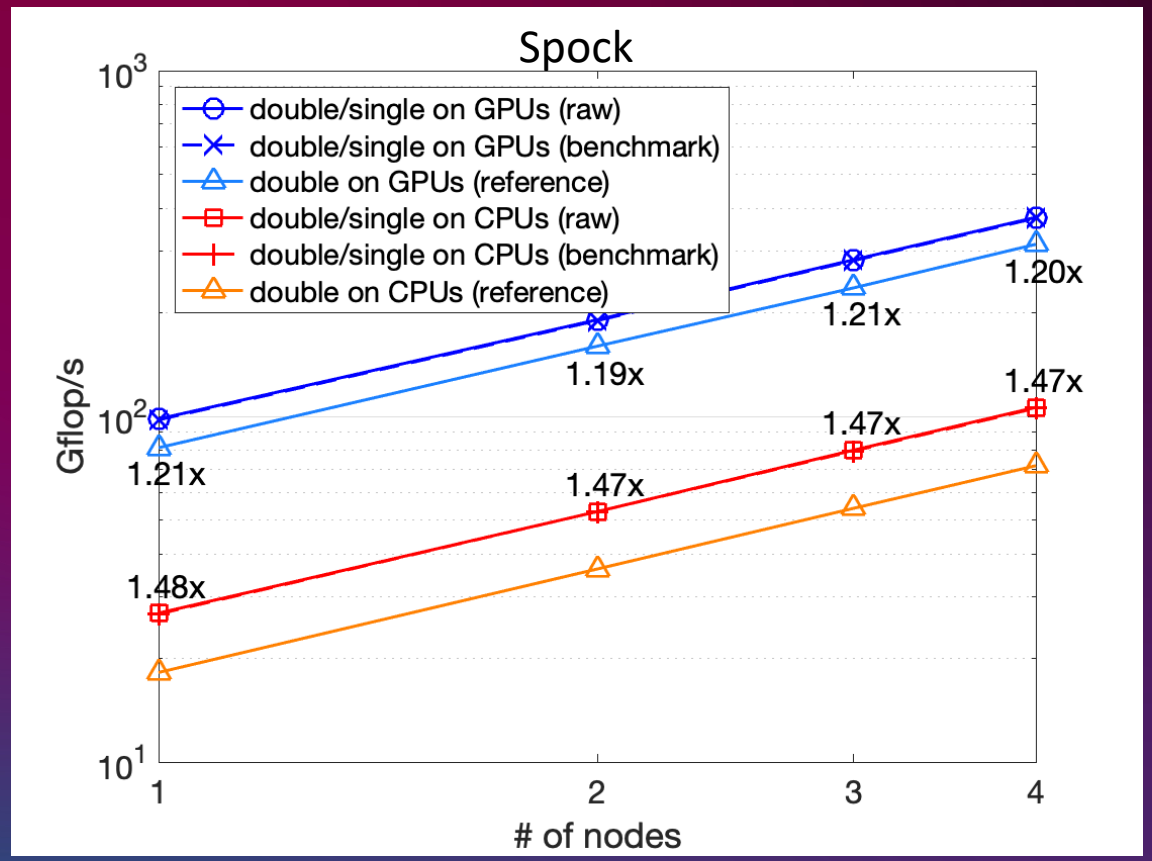
- Speedup of 1.2x using a non-optimized reference
- Most of the solver time is spent in SpTRSV
  - It has limited parallelism, and its performance may be more dominated by latency
  - it is harder to get speedup using lower precision
  - Reference implementation uses CuSparse SpMV & SpTRSV (no coloring)

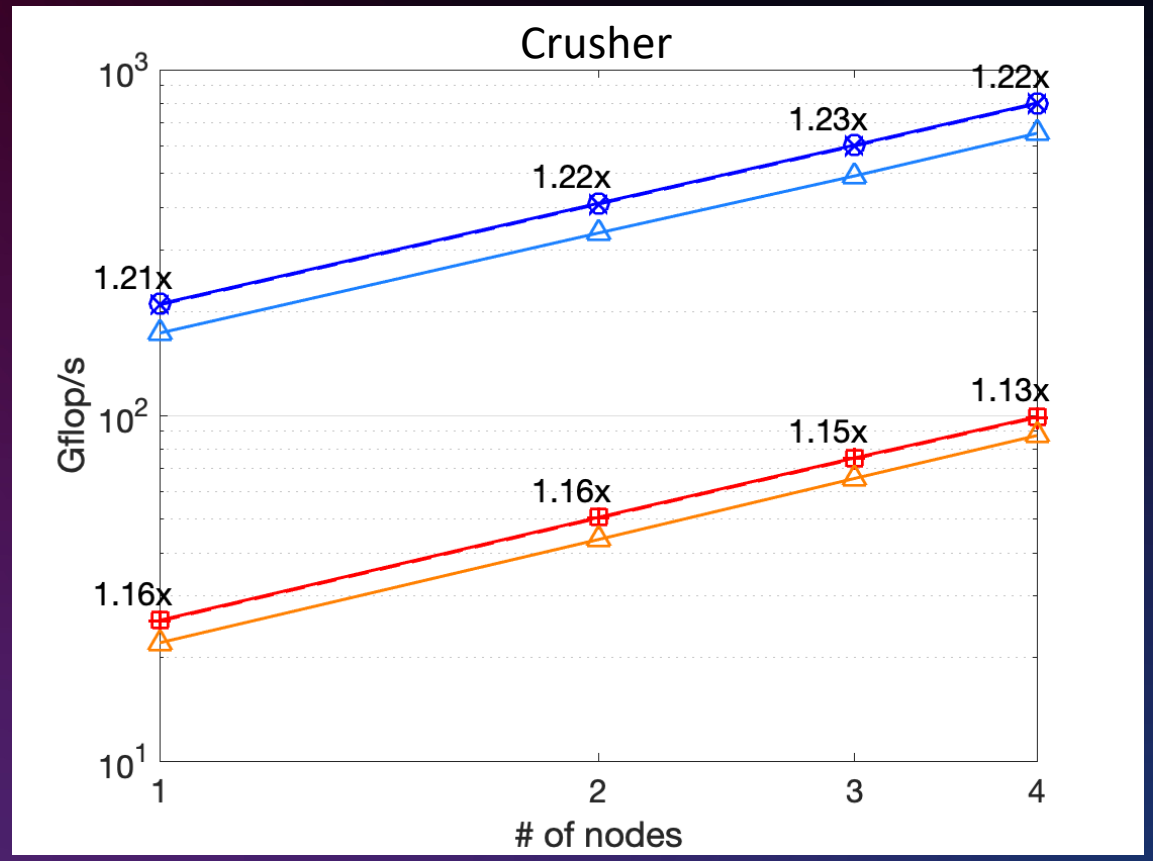
	Time in seconds with GPUs				TFlop/s with GPUs			
	GMG	SpMV	CGS2	Total	GMG	SpMV	CGS2	Total
Uniform	51.5	3.8	2.5	60.2	0.30	1.20	4.13	0.50
Mixed	44.5	2.4	1.8	50.1	0.35	1.87	5.73	0.61
Speedup	1.16	1.56	1.39	1.20	1.15	1.56	1.39	1.20



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

Performance of reference implementation on Spock & Crusher
AMD EPYC CPUs + AMD MI100/250X GPUs

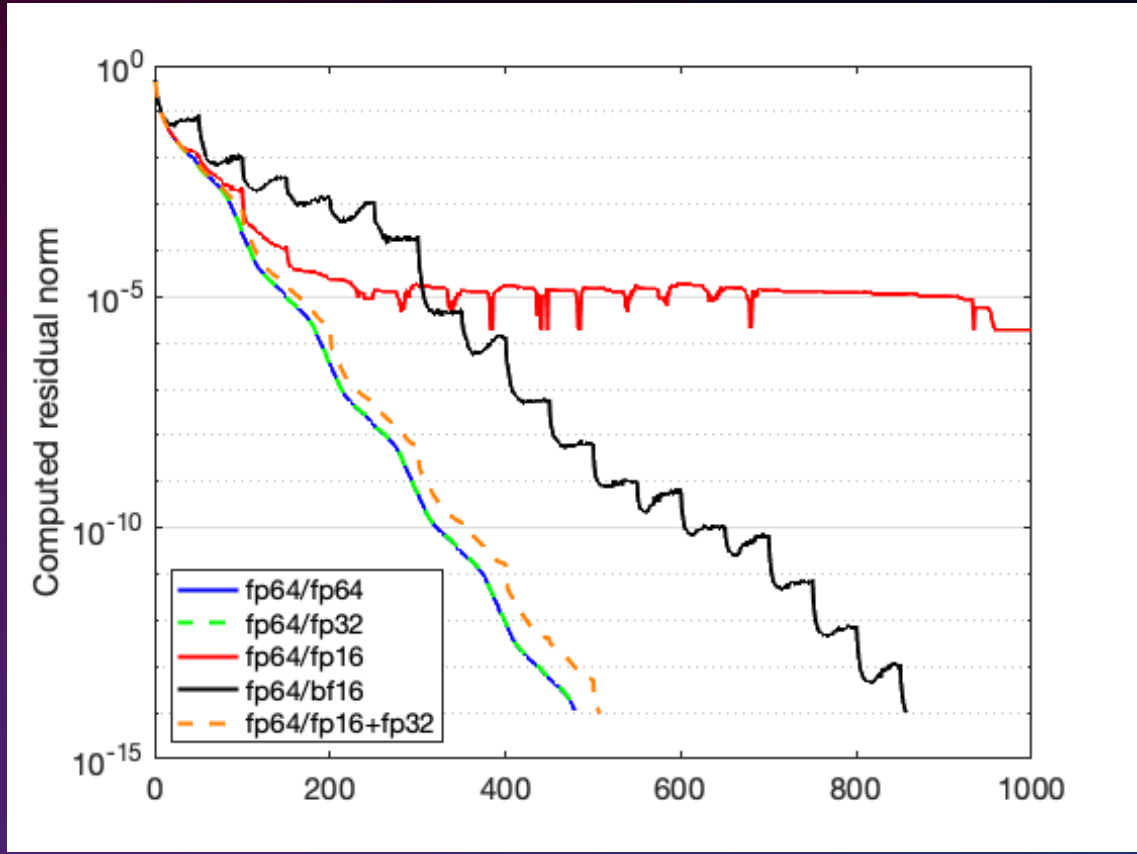




- Speedups, similar to those on Summit
  - AMD MI250X GPUs on Crusher have same peak performance using double and single

Potential of 16-bit float precisions

- Current specification allows any precisions
  - Though, reference implementation is templated with two precision
  - As long as it passes verifications
  - May be penalized for the increased number of iterations
- GMRES **may** converge, using FP16 for majority of iterations
  - Need careful accumulation for dot-products, e.g., into single precision
  - Under investigation !!



Numerical studies using plain Kokkos-Kernel GMRES on NERSC Perlmutter (NVIDIA A100 GPU)

Final remarks

- Proposing a new benchmarks
  - Captures the performance of applications
  - Allows the use of mixed lower precision arithmetic
- Reference implementation is publicly available
  - Finalizing benchmark framework
    - Validation, Choice of smoothers, etc.
  - Running on current top-ranked HP computers, at larger-scale, and beyond
  - Working on Kokkos/Kokkos-Kernels backend, for numerical & performance tests as “optimized” version with more mixed-precision (some results in the paper)
  - Comparing with application performance
  - Etc. etc.

Open for feedbacks & collaborations !!

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.