GPU-based LU Factorization and Solve on Batches of Matrices with Band Structure

Ahmad Abdelfattah, Stan Tomov, Piotr Luszczek, Hartwig Anzt, and Jack Dongarra

14th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Heterogeneous Systems

November 13th, 2023

ICL INNOVATIVE COMPUTING LABORATORY

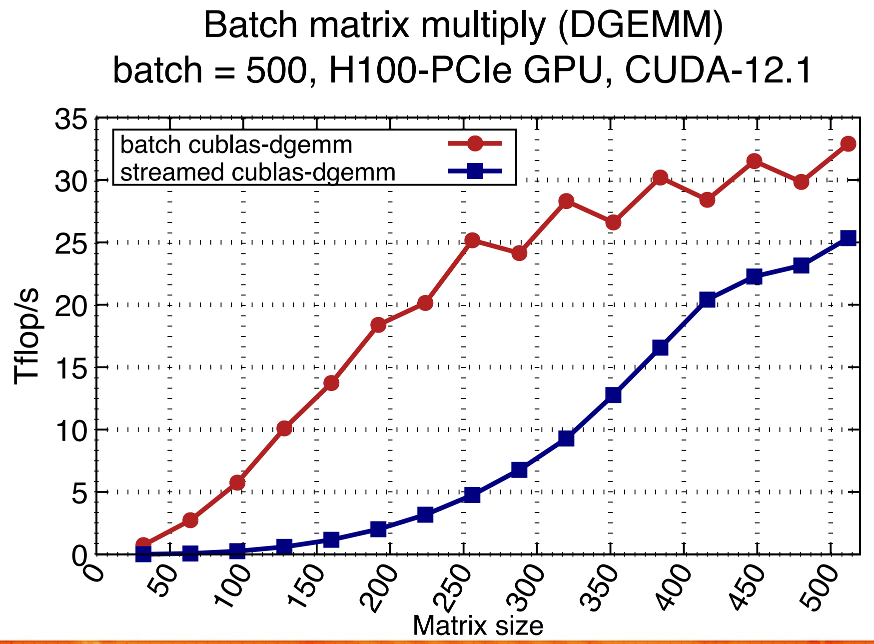
THE UNIVERSITY OF TENNESSEE KNOXVILLE

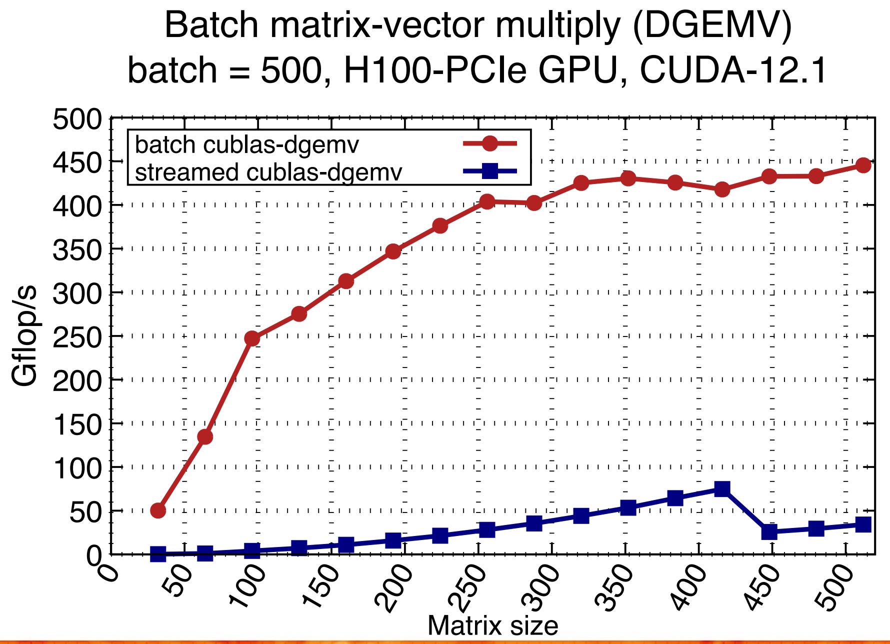
Batch Dense Linear Algebra

- Apply a BLAS/LAPACK operation on a batch of small matrices
  - Very active research topic since 2015
  - Standardization efforts, vendor support, wide adoption into applications
    - AI/ML, sparse direct solvers, hierarchical matrices, … etc
  - Main advantage → performance over traditional approaches

Batch Dense Linear Algebra

- Apply a BLAS/LAPACK operation on a batch of small matrices
  - Very active research topic since 2015
  - Standardization efforts, vendor support, wide adoption into applications
    - AI/ML, sparse direct solvers, hierarchical matrices, … etc
  - Main advantage → performance over traditional approaches





INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Goal of This Work

- Direct solve `Ax = B`, for batches of `A`'s and `B`'s
  - `A` is a band matrix
  - `B` is a dense matrix of right hand side(s)
  - Not supported by the vendors (e.g. cuBLAS or rocBLAS)
- LU factorization and solve on banded matrices
  - Partial pivoting is used for numerical stability
  - Reusable factors
- Standard Solution
  - No restriction on the dimensions, bandwidths, or #RHS
- Part of the ECP batch sparse LA effort
  - Combustion simulation, gyrokinetics, plasma fusion, … etc

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

LAPACK Convention

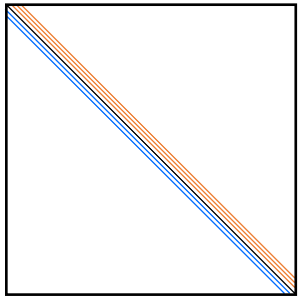
- For x = {s, d, c, z}
  - xGBTRF: LU factorization of a band matrix with partial pivoting
  - xGBTRS: Forward and backward triangular solves given the L/U factors
  - xGBSV: Factorize & solve
- E.g., LU Factorization with partial pivoting: P×A = L×U
  - `SUBROUTINE DGBTRF( m, n, kl, ku, AB, ldab, ipiv, info )`
  - `(m, n, ldab): matrix size and leading dimension`
  - `(kl, ku)    : Lower/upper bandwidths`
  - `AB          : pointer to the matrix`
  - `ipiv        : pivot vector`
  - `info        : return code`

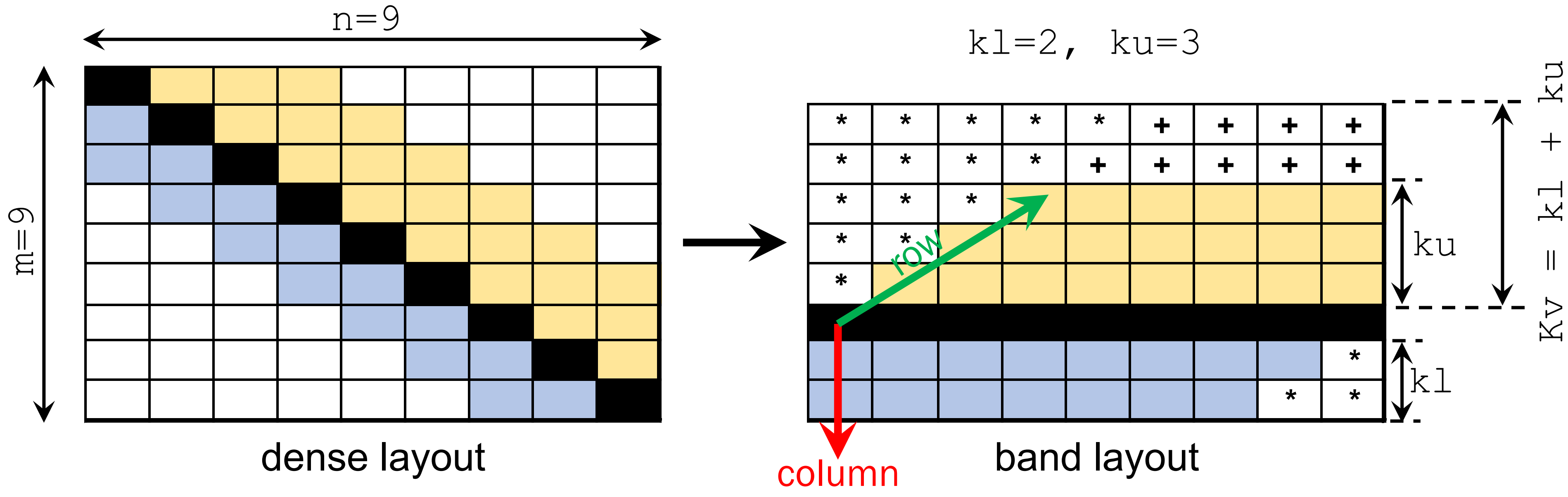
INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Band Matrix Layout

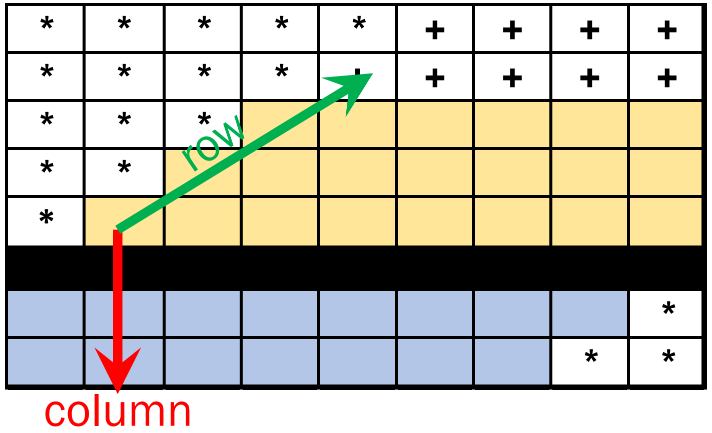
- LAPACK band storage
  - Still column-major, but store non-zeros only (column-wise)
  - Needs an extra space in the upper factor due to pivoting (`kl × n`)
  - The L factor is not stored in its “final” form (only right-looking row interchanges)
    - L is a product of permutations and unit lower triangular matrices
  - Reduces storage by `n × (m – [kv + kl + 1])` elements, where `kv = kl + ku`





Batch Band LU Design: Fully fused

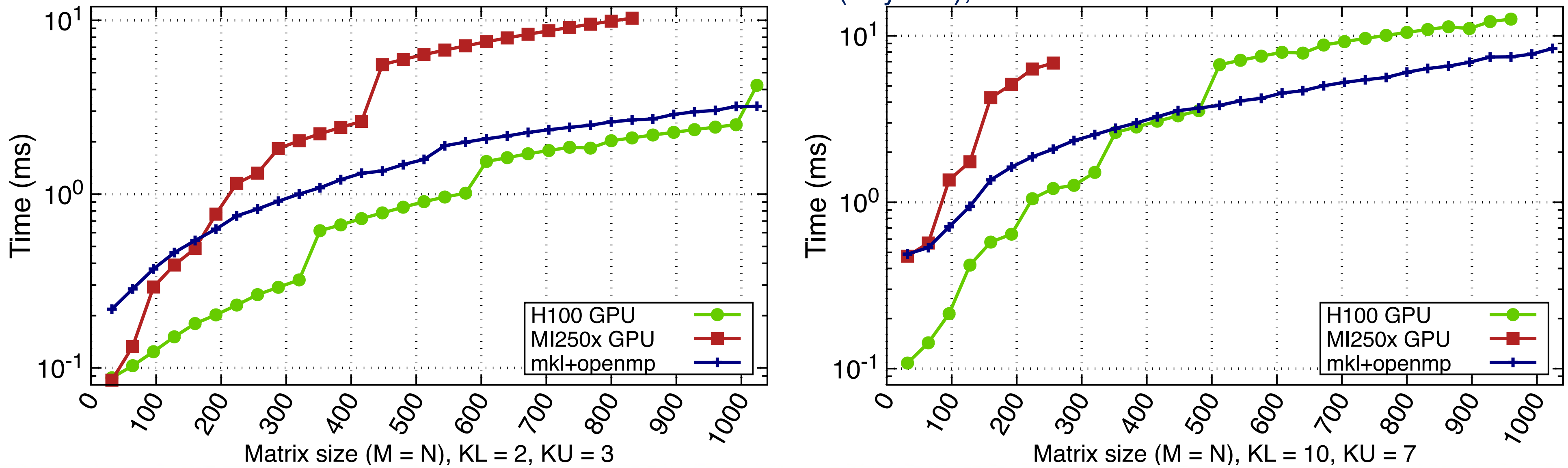
- Why? Optimal memory traffic
- Cache the entire matrix into shared memory or the register file
- Register file is faster, but there are challenges
  - Dense layout is friendly for contiguous access (one thread per row). This is not true for band layout.
  - Thread ownership for band layout needs to be altered
- Shared memory blocking
  - Unblocked band LU factorization



Batch Band LU Design: Fully fused

- Lower is better
- Smaller shared memory → lower occupancy
- Certain drops in occupancy cause jumps in execution time
- Shared memory becomes a bottleneck for larger sizes
- Fused kernels are not always the best option (despite the optimal memory traffic)

Batch band LU factorization (DGBTRF) -- Batch = 1000,
H100-PCIe GPU (CUDA-12.1), MI250x GPU (ROCM-5.5.1),
Intel Xeon Gold 6140 CPU (Skylake), MKL-2023.0.1



ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Batch Band LU Design: Sliding window

- We don't need to cache the whole matrix
- For a given column `j` ∈ `{0, 1, …, min(m,n)-1}`,
  and pivot location `jp`,

→ we can calculate the last column index affected by the factorization

  - `ju = max(ju, min(j+ku+jp, n-1))`

- Sliding window kernel
  - One thread-block per matrix
  - A kernel call factorizes `nb` columns, and accounts for the largest value of `ju`
  - Need (⌈N/nb⌉) iterations
  - Still ideal memory traffic
  - Relaxed shared memory requirements

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Batch Band LU Design: Sliding window

- A sliding window = factor window + update window
- Factor window
  - fixed width = `nb` (tuning parameter)
- Update window
  - maximum width = `kv + 1`
- Shared memory requirements
  - `(kv + nb + 1) * (kv + kl + 1)` elements
  - No longer dependent on N
  - Constant regardless of the matrix original size
  - Controllable through `nb`

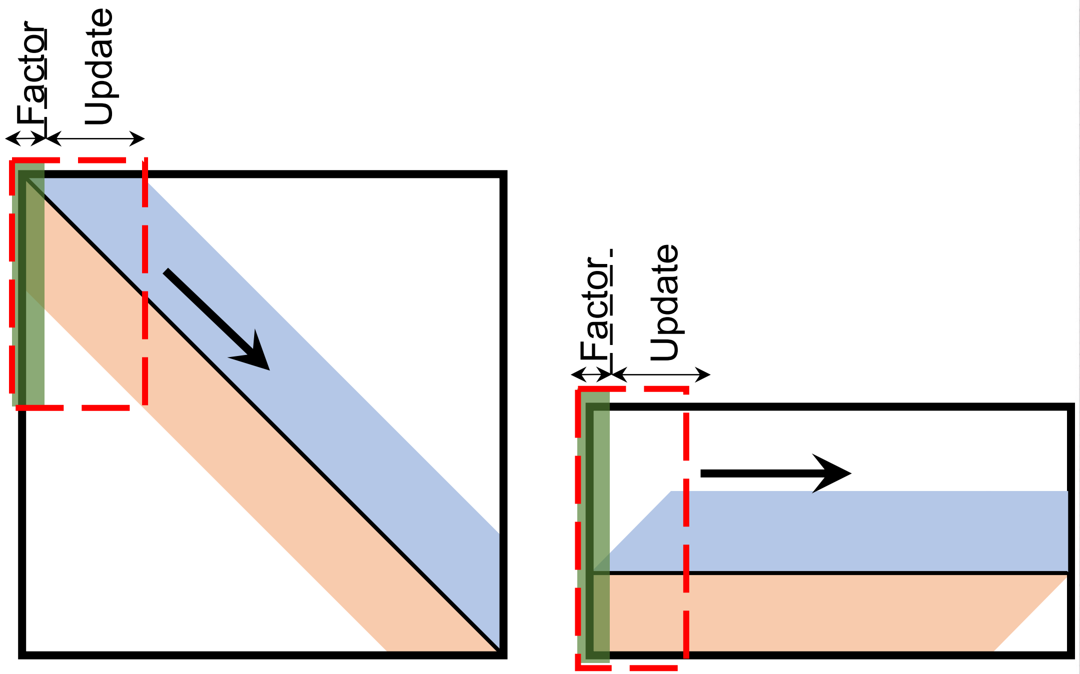


Illustration of the sliding window kernel

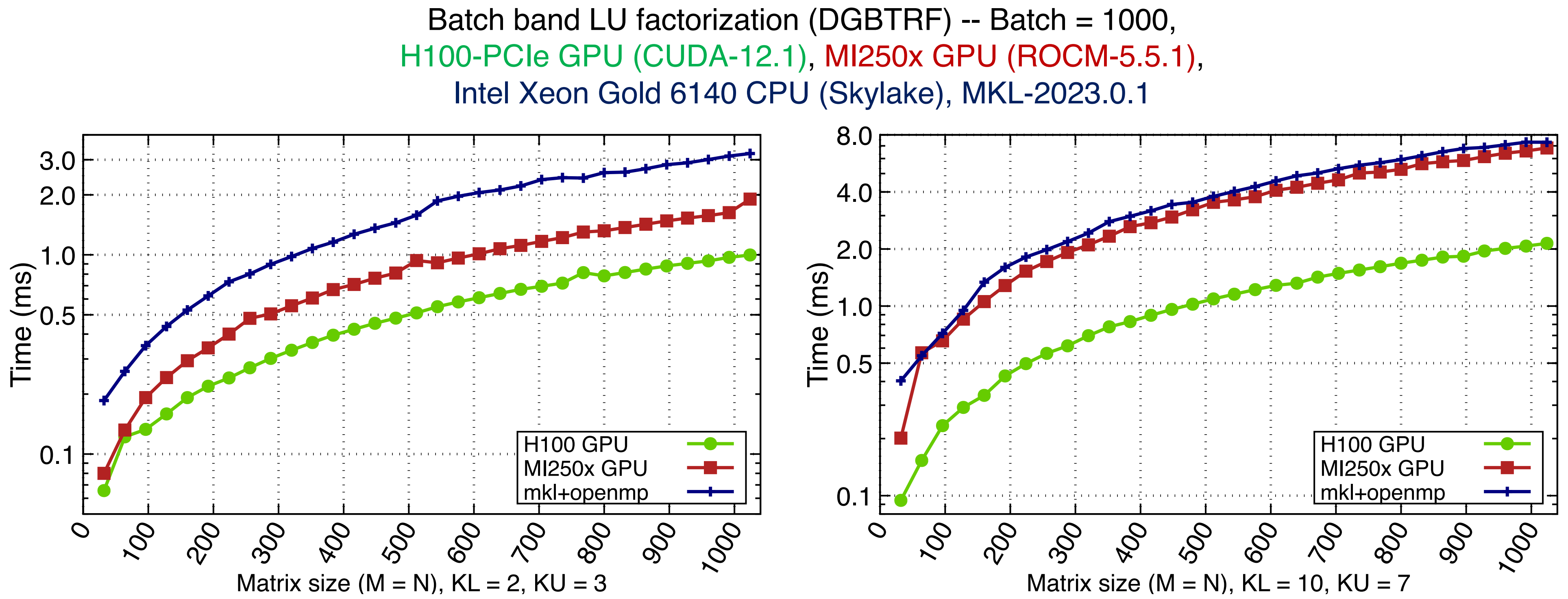
Batch Band LU Design: A Fallback Design

- The sliding window design covers a wide range of band sizes, but still can run out of shared memory
- Fallback design
  - Unblocked factorization (GBTF2) implemented using memory bound BLAS kernels
  - Not expected to deliver a good performance
  - Future plan: use L3 BLAS
- The overall picture
  1) If the matrices are very small (up to 32x32), use the fully fused code
  2) else if shared memory capacity permits, use the sliding window factorization
  3) Else, launch the fallback design

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

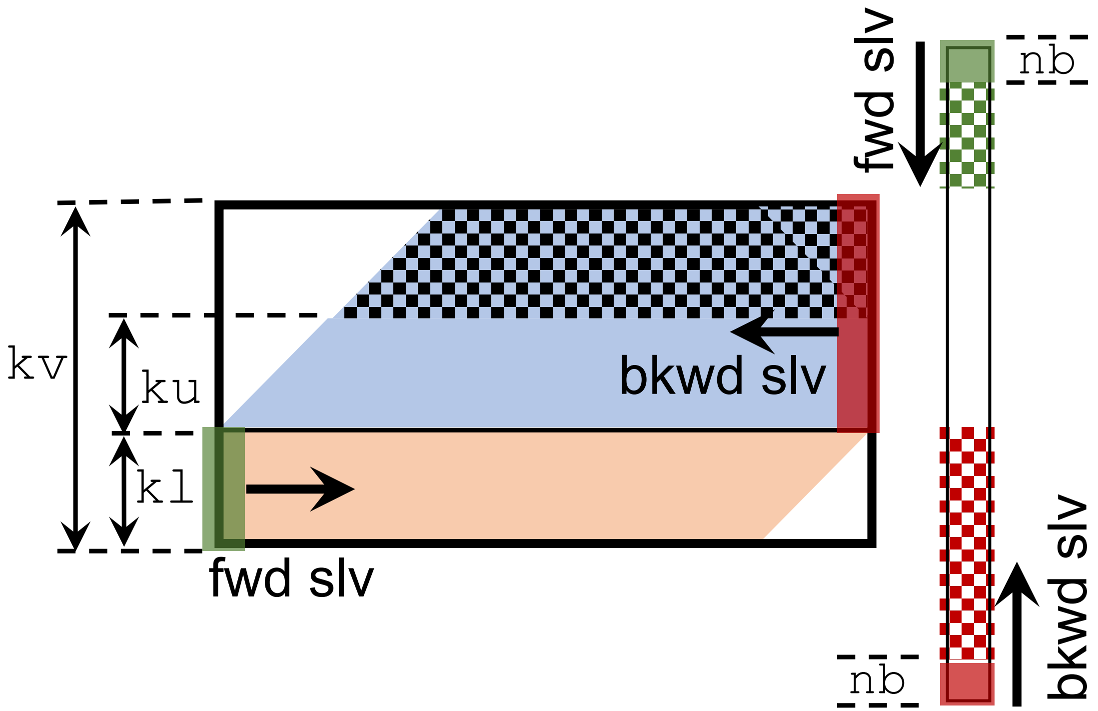
Band LU Final Performance



	H100-PCIe GPU			MI250x GPU		
(kl, ku)	min.	max.	avg.	min.	max.	avg.
(2, 3)	2.13×	3.43×	3.07×	1.67×	2.32×	1.88×
(10, 7)	3.07×	4.27×	3.56×	0.96×	2.01×	1.16×

Batch Band Triangular Solve

- Two designs
  1) A reference implementation as a fallback design (column-wise)
  2) A blocked version similar to the sliding window technique

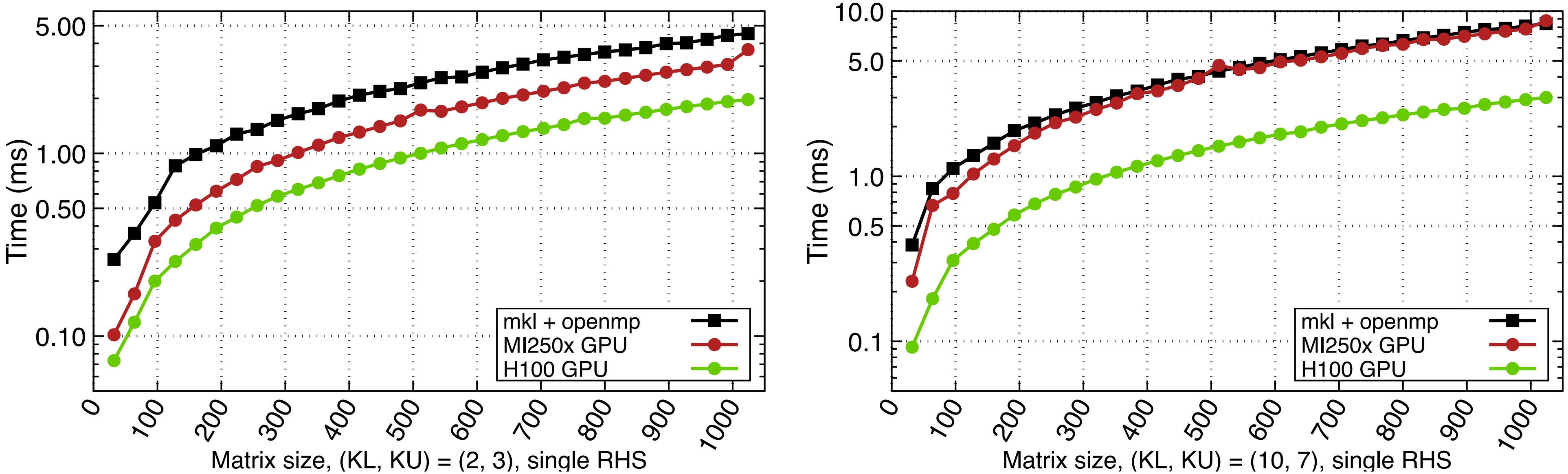


Overall Picture for Factorization and Solve

- GBSV
  a) Fused approach (one kernel for factorization & solve)
  b) Standard approach
     - Factorization
       1) Fused
       2) Sliding window
       3) Ref. implementation
     - Triangular solve
       1) Blocked
       2) Ref. implementation

Final Performance Results (single RHS)

Batch band LU factorization & solve (DGBSV) – RHS = 1, Batch = 1000,
H100-PCIe GPU (CUDA-12.1), MI250x GPU (ROCM-5.5.1),
Intel Xeon Gold 6140 CPU (Skylake), MKL-2023.0.1



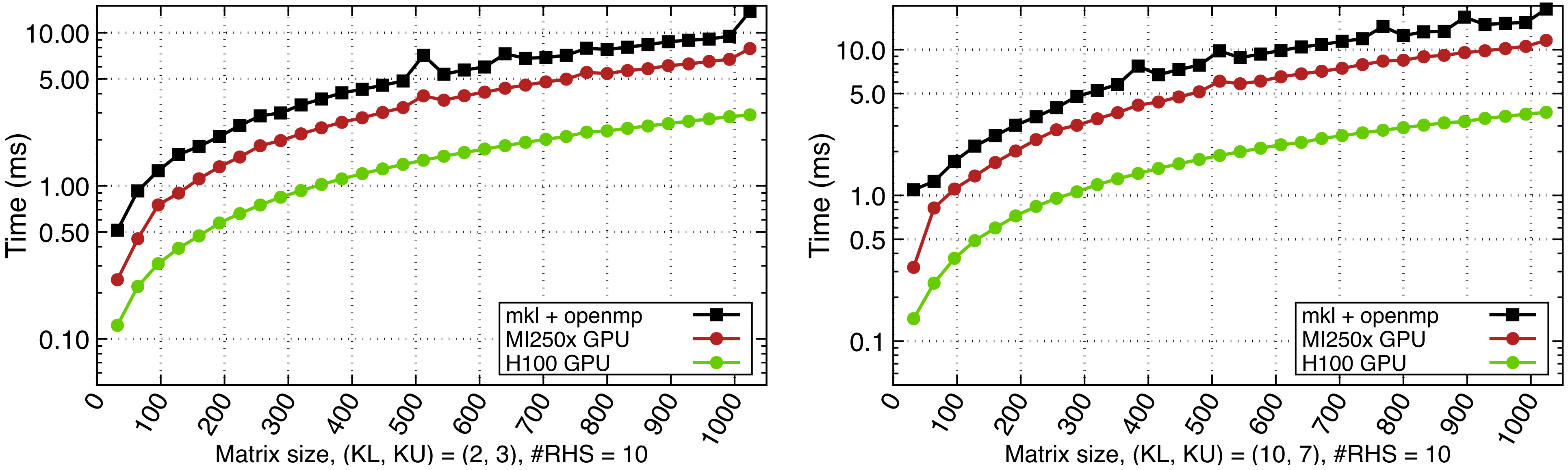
	H100-PCIe GPU			MI250x GPU		
(kl, ku)	min.	max.	avg.	min.	max.	avg.
(2, 3)	2.23×	3.58×	2.54×	1.22×	2.58×	1.59×
(10, 7)	2.79×	4.65×	3.03×	0.92×	1.66×	1.11×

INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Final Performance Results (RHS = 10)

Batch band LU factorization & solve (DGBSV) – RHS = 10, Batch = 1000,
H100-PCIe GPU (CUDA-12.1), MI250x GPU (ROCM-5.5.1),
Intel Xeon Gold 6140 CPU (Skylake), MKL-2023.0.1



	H100-PCIe GPU			MI250x GPU		
(kl, ku)	min.	max.	avg.	min.	max.	avg.
(2, 3)	3.33×	4.85×	3.69×	1.40×	2.11×	1.57×
(10, 7)	4.12×	7.67×	4.64×	1.42×	3.41×	1.61×

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Conclusion and Lessons Learned

- First try for batch band LU factorization & solve on GPUs
  - Supports any size and band structure
  - Fully compliant with LAPACK's specifications
- Chances of parallelism are limited
  - Mostly across the batch
  - CPUs are tough competitors
- Shared memory capacity is a bottleneck
  - Maybe rethink storage in the register file
- Performance tuning is not straightforward

Future Work

- Efficient use of the register file
- Single matrix factorization
  - Challenging to rival the CPU performance
- Robust performance tuning
- Support for Intel GPUs
- Support for different sizes and/or different bandwidths?

Code is available

- https://bitbucket.org/icl/magma
- Lined up for MAGMA 2.8.0

Thank You

INNOVATIVE
COMPUTING LABORATORY

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation’s exascale computing imperative.