Batched Sparse Linear Algebra Interfaces, Solvers, Libraries, and Preconditioners

Piotr Luszczek, Ahmad Abdelfattah, Stanimire Tomov

University of Tenessee

Existing Batched Interfaces from 3 Vendors

- AMD's ROCm interface with rocBLAS and rocSOLVER
  - rocBLAS: Level 1, 2, and 3 BLAS
  - rocSOLVER: potf2, potrf, getf2, getrf, geqr2, geqrf, gerq2, gerqf, geql2, geqlf, gelq2, gelqf, gebd2, gebrd, sytd2, hetd2, sytrd, hatred, sygs2, hegst, trtri, getri, getrs, gesv, potri, potrs, posv, gels, syev, heev, syevd, heevd, sygv, hegv, sygvd, hegvd, gesvd, getf2_npvt, getrf_npvt, getri_npvt, getri_outofplace, getri_nvpt_outofplace
- Intel's oneMKL, part of oneAPI
  - BLAS: AXPY, COPY, GEMV, GEMM, SYRK, TRSM, DGMM
  - GEQRF, GETRF, GETRI, GETRS, ORGQR, POTRF, POTRS, UNGQR
- NVIDIA's Batched Functionality in CUDA
  - cuBlas: gemm, trsm, getrf, geqrf, gels
  - cuSolver: potrf, potrs, gesv, syevj, sparse csrQRslv
  - cuSparse: SpMM

Some Applications and Use Cases

- Pele: chemical kinetics
  - Matrix size: 30..200
  - Batch count: 10+
  - Condition number: 10^{15} (reduced with scaling to ≈100)
- ExaStar: FLASH, Nuclear kinetics
  - Matrix size: 100..200
  - Batch count: 10+
  - Condition number: high (for stiff problems)
- WDMApp / XGC code: plasma simulation
  - Matrix size: +1000 (5 point stencil, bandwidth: 40)
  - Batch count: 16+
  - Condition number: manageable
- ExaSGD: electric grid simulation via stochastic grid dynamics
  - Batching in Schur complement and for arrowhead sparse matrices

Proposed Interface Design: Batched Band LU Factorization

- dgbtrf_batched
  - int use_pivoting, // 0 - no pivoting; 1 - row partial pivoting enabled
  - int m, // number of matrix rows
  - int n, // number of matrix columns
  - int kl, // size of lower bandwidth
  - int ku, // size of upper bandwidth
  - double **dev_a_arr, // array of pointers to the linear systems' matrices
  - int *ldda, // leading dimensions of the linear system matrices
  - int **ipiv_arr, // array of pointers to pivoting vectors
  - int *info_arr, // array of status information for each factorization
  - int batchCount, // the total size of the batch
  - DeviceContext device_context // where, when and how to run

Proposed Interface Design: Batched Band LU Solve

- dgbtrs_batched
  - int use_pivoting, // 0 - no pivoting; 1 - row pivoting enabled
  - enum Transpose trans, // no transpose, transpose, or conjugate transpose
  - int m, // number of matrix rows
  - int n, // number of matrix columns
  - int kl, // size of lower bandwidth
  - int ku, // size of upper bandwidth
  - double **dev_a_arr, // array of pointers to the linear systems' matrices
  - int *ldda, // leading dimensions of the linear systems' matrices
  - int **ipiv_arr, // array of pointers to pivoting vectors
  - double **dev_b_arr, // array of pointers to the right hand sides
  - int *lddb, // leading dimensions of the right hand sides
  - int *info_arr, // array of status information for each factorization
  - int batchCount, // the total size of the batch
  - DeviceContext context // where, when and how to run

Proposed Interface Design: Batched Band LU 1-step Solve

- dgbsv_batched(
  - int use_pivoting, // 0 - no pivoting; 1 - row pivoting enabled
  - int m, // number of matrix rows
  - int n, // number of matrix columns
  - int kl, // size of lower bandwidth
  - int ku, // size of upper bandwidth
  - double **dev_a_arr, // array of pointers to the linear systems' matrices
  - int *ldda, // leading dimensions of the linear system matrices
  - int **ipiv_arr, // array of pointers to pivoting vectors
  - double **dev_b_arr, // array of pointers to the right hand sides
  - int *lddb, // leading dimensions of the right hand sides
  - int *info_array, // array of status information for each factorization
  - int batchCount, // the total size of the batch
  - DeviceContext context // where, when and how to run

Proposed Batched Direct Solver Interface

- direct_solve_batched<T, RealT>
  - AlgorithmInfo solver, // symbolic,numerical, allocation, triangular solve, solver:LU,QR,Cholesky
  - int M, int N, // matrix dimensions: rows and columns
  - int NNZ, // number of non-zero entries
  - int NRHS, // number of right-hand-sides
  - SparseMatrixStorage *A, // array of sparse matrix storage for the linear system matrices
  - <T> **Xptr, // array of pointers to dense solution storage
  - int *ldX, // array of leading dimensions of X
  - <T> **RHSptr, // array of pointers to dense RHS storage
  - int *ldRHS, // array of leading dimensions of RHS
  - <RealT> **ReqbPtr, // array of pointers to diagonal row scaling vectors, each of size M
  - <RealT> **CeqbPtr, // array of pointers to diagonal col. scaling vectors , each of size N
  - int **RpivPtr, // array of pointers to row permutation vectors , each of size M
  - int **CpivPtr, // array of pointers to column permutation vectors , each of size N
  - SparseMatrixStorage *F, // array of sparse matrix storage for the factored matrices
  - int batchCount, // number of matrices in the batch
  - DeviceContext context // device context including queues, events, dependencies

Proposed Batched Sparse Iterative Solver Interface

- iterative_solver_batched<T>
  - AlgorithmInfo solver_info, // solver to use: CG, BiCGStab, GMRES, IDR, … and its parameters: maximum iteration count, tolerance for convergence stopping criterion
  - int M, int N, // matrix dimensions: rows and columns
  - int NNZ, // number of non-zero entries
  - int NRHS, // number of right-hand-sides
  - SparseMatrixStorage *A, // array of sparse matrix storage for the linear system matrices
  - <T> **Xptr, // array of pointers to solution storage
  - int ldX, // leading dimension of X
  - <T> **RHSptr, // array of pointers to RHS storage
  - int *ldRHS, // leading dimension of RHS
  - <T> **X0ptr, // array of pointers to initial guess storage
  - int *ldX0, // leading dimension of initial guess storage
  - int BatchCount, // number of matrices in the batch storage: CSR, CSC
  - DeviceContext context // device context including queues, events, dependencies

Proposed Batched Sparse Iterative Solver C++ Interface

- template <typename OperatorType, typename VectorType, typename PrecOperatorType, typename KrylovHandleType> static int batchedSolveIterative
  - const OperatorType& A, // third-order sparse tensor of size b by n by n
  - const VectorType& B, // second-order dense tensor (the right-hand side) of size b by n
  - const VectorType& X, // second-order dense tensor (the initial solution) of size b by n
  - KrylovHandleType* handle, // Krylov subspace algorithm and its parameters: stoping criterion, max. iterations, etc.; and output: achieved convergence, final iteration count, etc.
  - const PrecOperatorType& P, // left preconditioner
  - DeviceContext context

Proposed Batched Sparse Iterative C++ Extended Interface

- template <scalar_t ValueType, typename RuntimeReturn, typename RuntimeContext, multi_vector_t VectorType, solver_lin_op_t<VectorType> MatrixType, preconditioner_t<MatrixType, VectorType> PrecType, typename StopperType, typename LoggerType> RuntimeReturn batched_iterative_solver
  - const MatrixType* A, // array of sparse matrix storage for the linear system matrices
  - const VectorType* b, // array of pointers to RHS vectors
  - VectorType* x, // array of pointers to solution vectors
  - PrecType* preconditioner,// array of sparse matrix storage for the preconditioner matrices
  - StopperType& stop, // stopping criteria object
  - LoggerType& logger, // logger object for reporting iteration progress
  - DeviceContext& context // device context including queues, events, dependencies

Initial Sparse Batched Kernels in Libraries

- Ginkgo
  - Sparse batched CG, BiCGStab, GMRES, IDR, preconditioning
- hypre
  - Using sparse batched solvers in approximate inverse and smoothing
- KokkosKernels
  - Batched sparse GMRES
- MAGMA
  - Band solvers and factorizations
    - Bandwidth and pivot sequence may differ across batch
- PETSc
  - Simultaneous function evaluations
  - Extending and porting MATKAIJ representation
- SUNDIALS
  - Simultaneous sparse solves
- SuperLU
  - Batched direct factorization and triangular solves

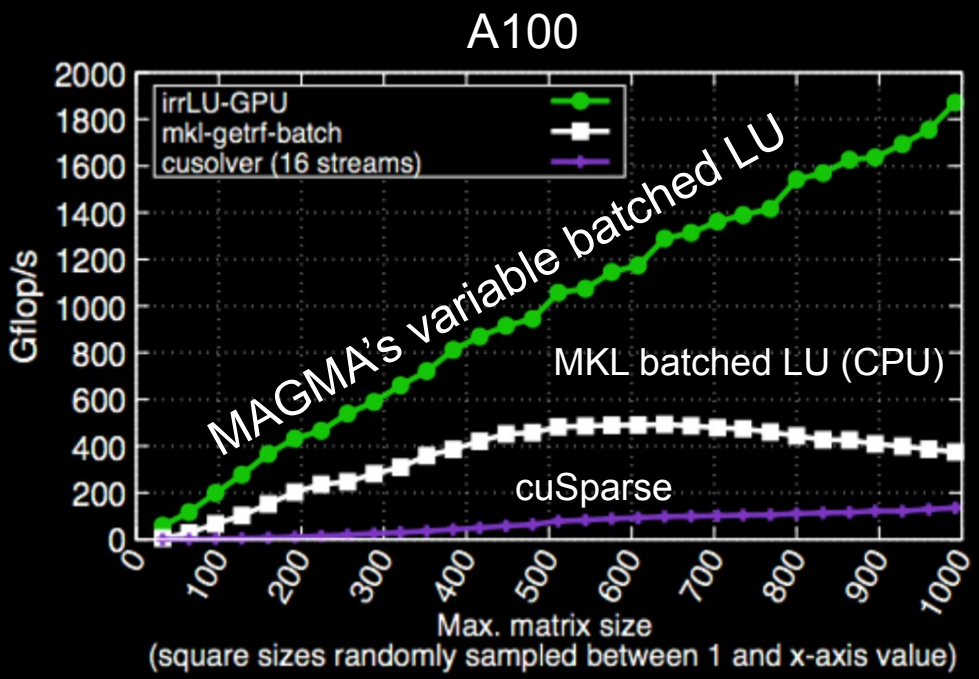
Benchmarking

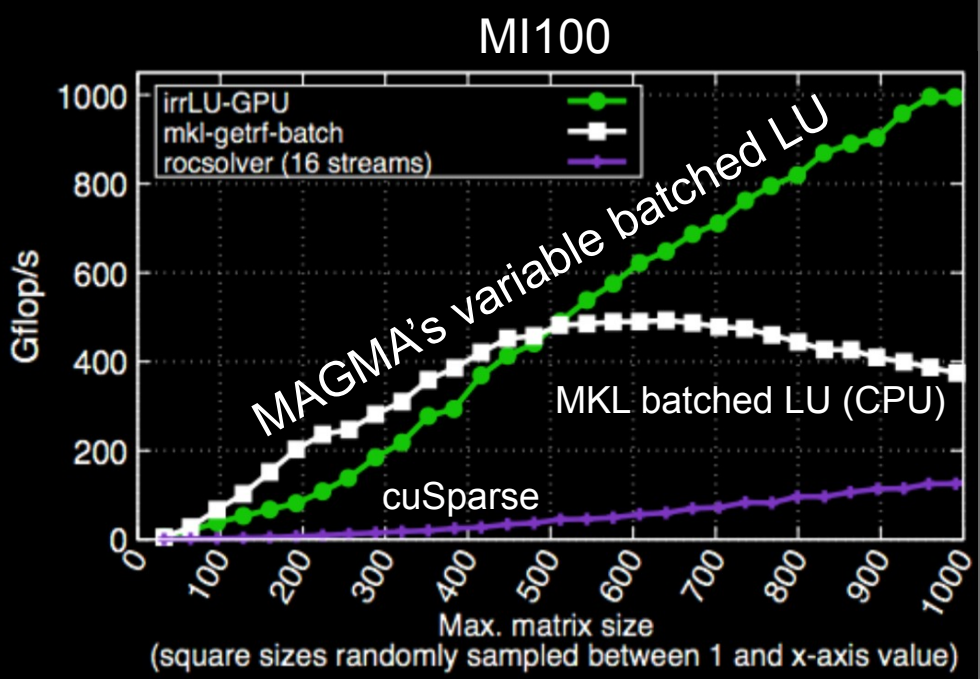
- Data collections
  - Matrices and matrix batches
  - Derived from the initial application portfolio
    - Dodecane
    - Drm19
    - Gri12
    - Gri30
    - Isooctane
    - Lidryer

} Contributed by Cody Balos

- Benchmark scenarios
  - Sorting:
    - By amount of work (e.g. bandwidth size)
    - By sparsity pattern (e.g. value of NNZ, fill-in, or use superset of adjacency)
    - By condition number
  - Maximum variability

Early Performance Assesment (Gflop/s, higher is better)

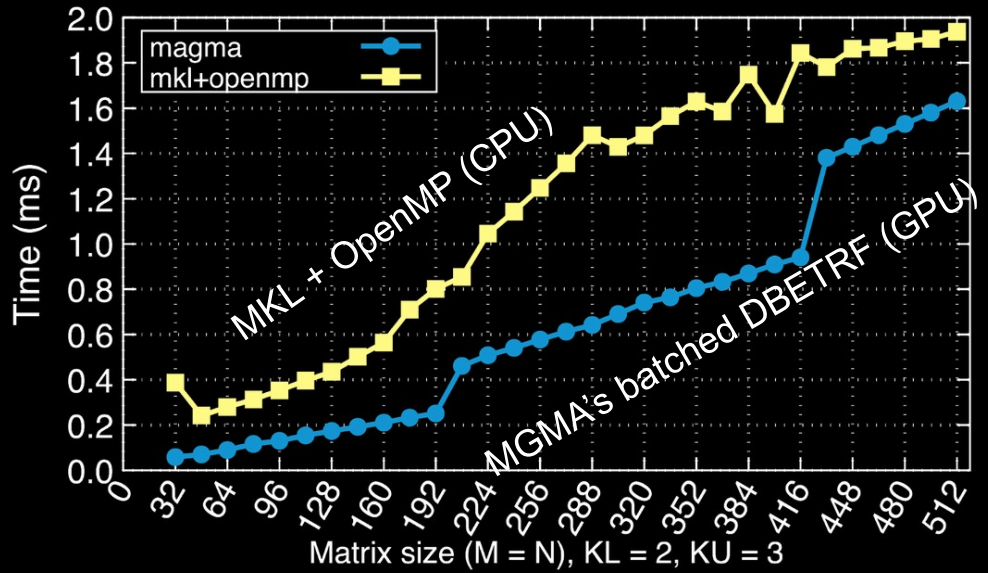


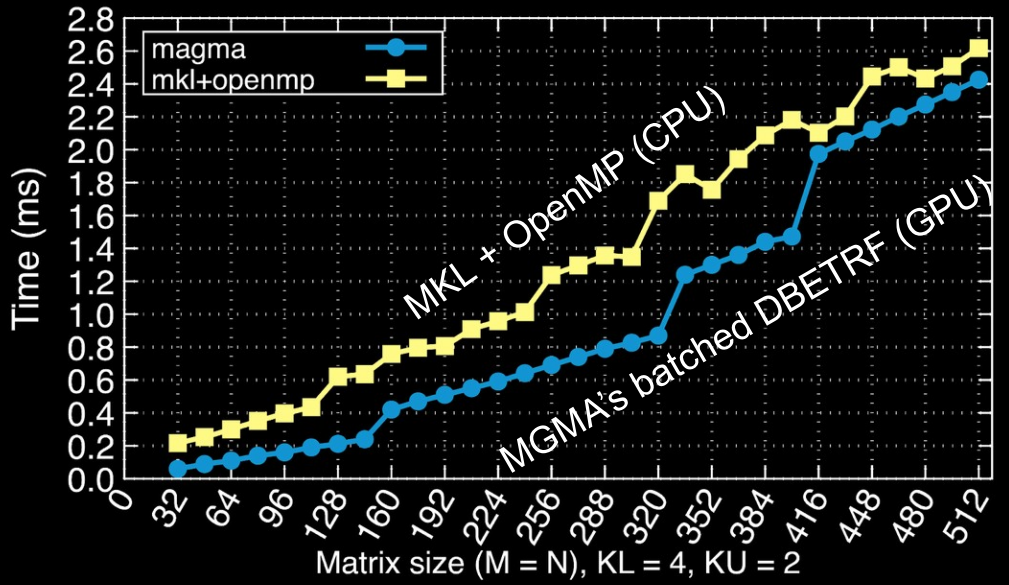


Credit: Ahmad Abdelfattah and Stanimire Tomov

Early Performance Comparison: LU (time, lower is better)

NVIDIA Ampere A100





Credit: Ahmad Abdelfattah and Stanimire Tomov