OpenMP

SC'20 Booth Talk Series

OpenMP in Exascale Numerical Libraries

Hartwig Anzt*, Mark Gates, Piotr Luszczek, and Stanimire Tomov
University of Tennessee
*Karlsruhe Institute of Technology

Scope: Libraries and Platforms

- Numerical libraries
  - Ginkgo
    - Sparse storage formats, iterative solvers, preconditioners, multigrid, SpMv
    - Variants: reference, OpenMP, accelerators
  - MAGMA
    - Dense: linear and eigenvalue/SVD
    - Sparse: iterative, preconditioners, storage
    - Mixed precision: 16-, 32-, and 64-bit solve
    - Variants: CUDA, clMagma, micMagma
  - PLASMA
    - Dense: linear, least-squares, EIG/SVD
    - Tile matrix layout and OpenMP 4 tasking
    - Variants: POSIX, WinThreads, OpenMP 4
  - SLATE
    - Distributed memory, multicore and GPUs
    - Flexible tile storage with affinity tracking

- ECP hardware/software
  - NVIDIA CUDA 10 and 11
    - Summit, Perlmutter
  - AMD HIP and rocM
    - Perlmutter, Frontier, Cray CCE
  - Intel DPC++
    - Aurora
- CI/CD systems
  - Cloud VMs and bare metal systems
    - Any modern x86-64
    - POWER
    - ARM

General Portability Approach

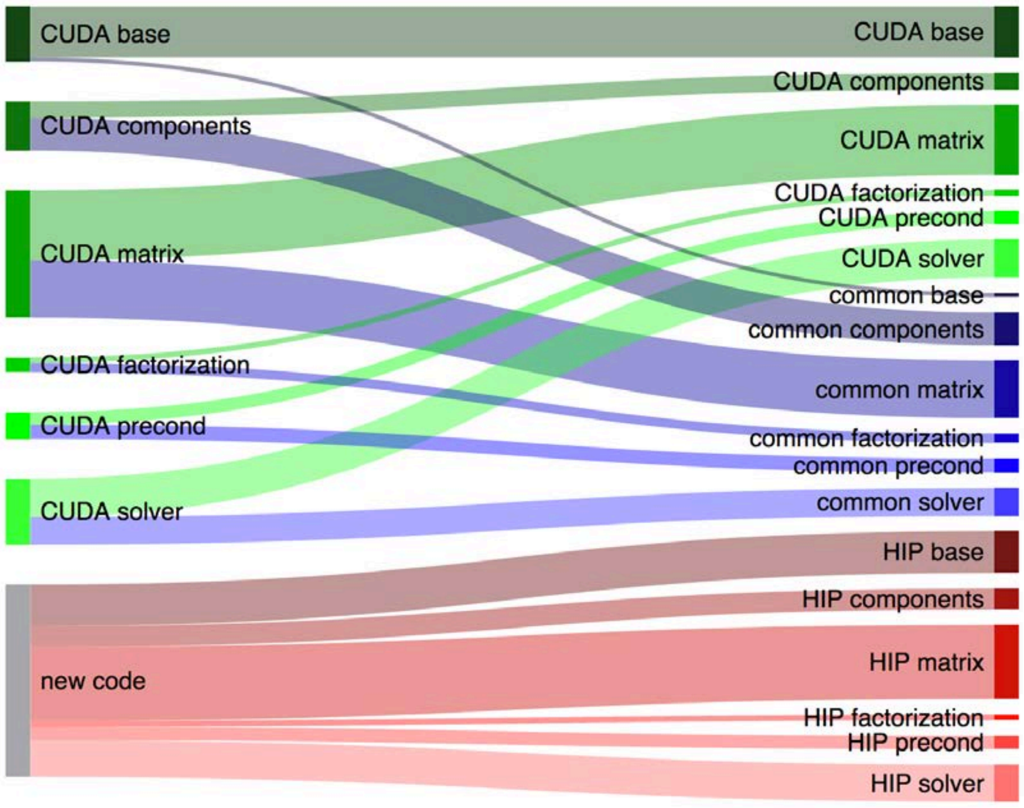
- Common interfaces
  - Reuse or emulate established interfaces (optimized by vendors)
    - DGEMM(), cuDgemm(), hipDgemm(), rocDgemm(), mkl_dgemm()
- Abstractions
  - Well defined and practical objects' structure for user data
  - Focus on user experience
    - Object hierarchy for matrix, vector, execution policy (host or device)
- Generic algorithms
  - Programming against generic types
  - Testing on concrete types

Ginkgo's OpenMP Backend

- Sparse calculations are loop-heavy
  - `parallel for` is ubiquitous throughout the code base
- Atomics are important for memory utilization
  - Specific clauses allow further optimization: `atomic read`, `write`, and `update`
- Reductions allow efficient parallelization of row/column operations
  - Standard reductions are used with arithmetic (matrix values) and logical operators (matrix structure)
  - Declared reduction for matrix-specific sparse operations

Ginkgo Kernels: from CUDA to HIP

- Kernels
  - Mostly shared between CUDA and HIP
  - Considered common
  - Abstracting away non-portable features
    - Cooperative groups
- Backend-specific optimizations
  - Always added for new backend
  - Must be maintained independently



CUDA only

CUDA & HIP

Ginkgo Porting Remarks and Challenges

- OpenMP remains a portable and efficient target
- CUDA and HIP are now relevant alternatives for Ginkgo functionality
- Similarities of HIP/CUDA syntax allow for significant sharing
  - Even for low-level implementations
- (In case of Ginkgo) compiling HIP to target NVIDIA devices has moderate effect on performance
- Comparable performance of CUDA/HIP across Ginkgo functionality

Use Case: MAGMA

MAGMA's OpenMP Use

- MAGMA is a library with many hybrid algorithms
  - Both host processor and the accelerator is needed throughout the algorithm
  - Host processor must use parallelism
    - If possible then use optimized libraries: BLAS or LAPACK
    - The remaining code must use manual coding
      - OpenMP reduces developer effort
      - Some portions are numerically sensitive (eigenvalue calculations)

MAGMA: Dense and Sparse Linear Algebra

- MAGMA’s functionality scope exposed issues in HIP stack
  - Filed with AMD and fixed with HIP/rocM 3.5
- Automated approach necessary due to a large code base
  - Hippify’ing tools have their limits for MAGMA code base
  - Remaining issues fixed with automation (again: see code size and its changes)
- Performance results
  - Still work in progress across MAGMA, HIP, and rocM libraries
  - No clear winner or guidelines emerge yet

Performance Results: DGEMM on MI25 & MI50

hipMAGMA 1.0

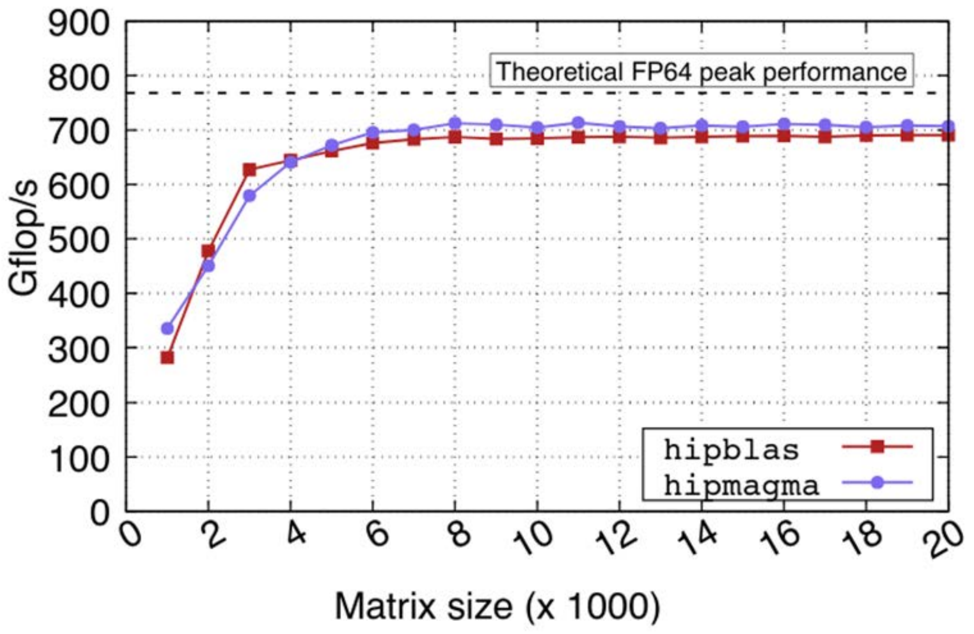


Fig. 3. Performance of the GEMM operation in double precision on the Mi25 GPU. Results are shown for square sizes using ROCm 3.5.

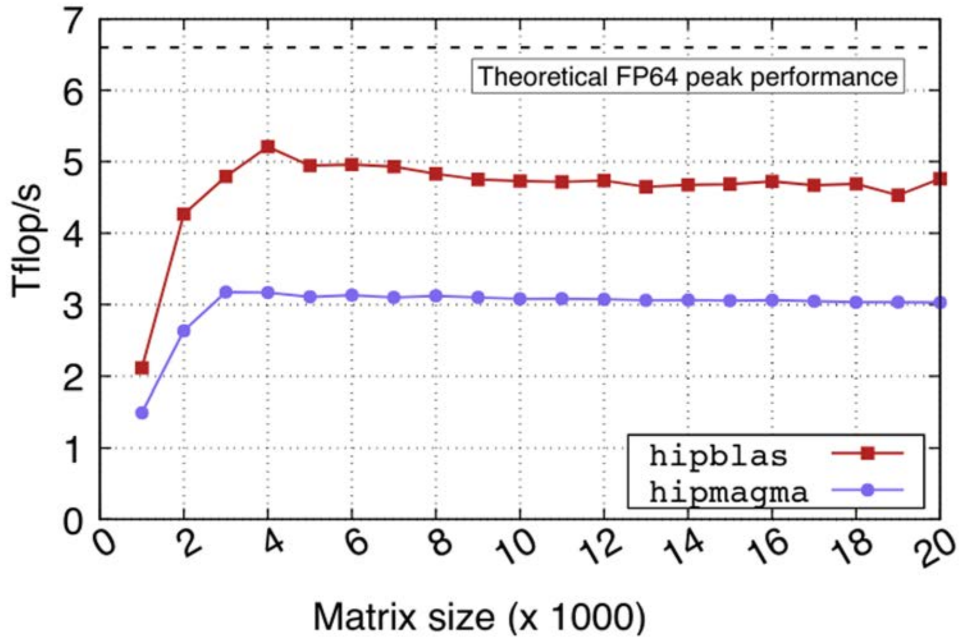
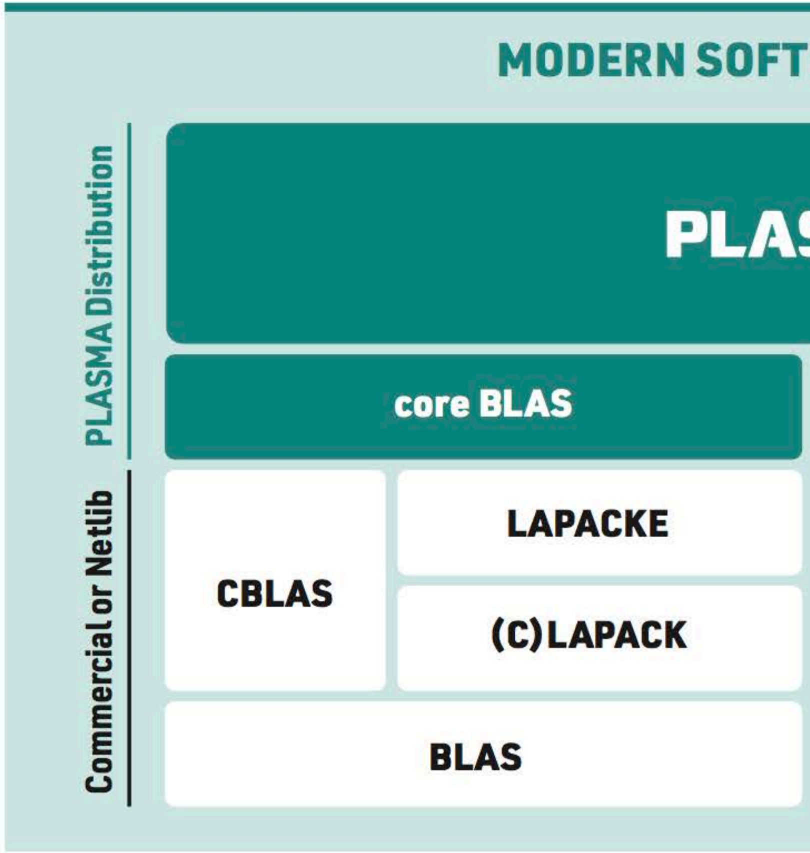


Fig. 2. Performance of the GEMM operation in double precision on the Mi50 GPU. Results are shown for square sizes using ROCm 3.5.

Use Case: PLASMA

PLASMA with OpenMP Tasking



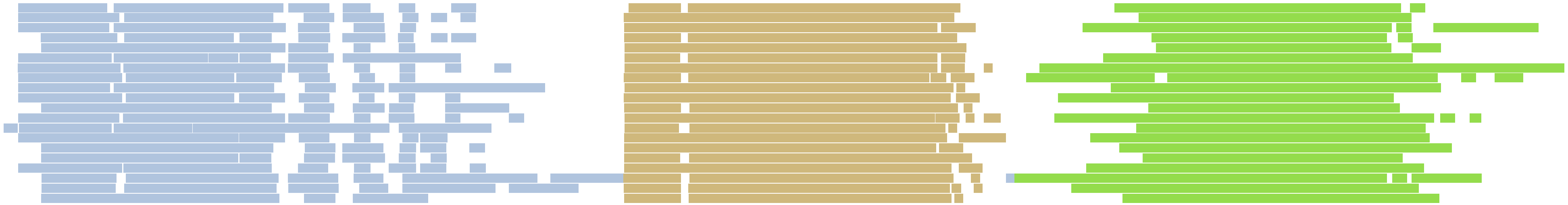
Multicore

- task
- taskloop

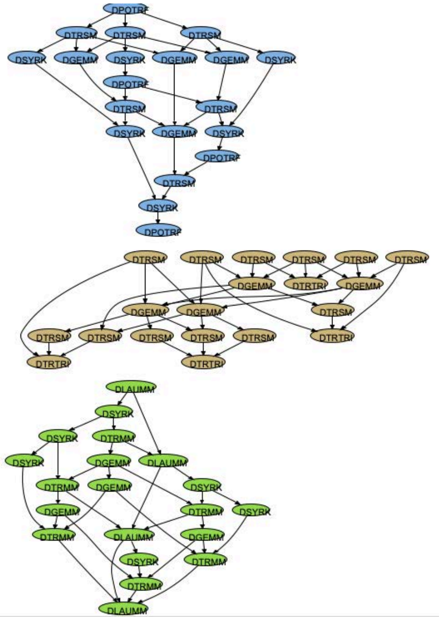
Accelerators

- Data transfers
- Offload kernels

PLASMA OpenMP Cholesky Inversion

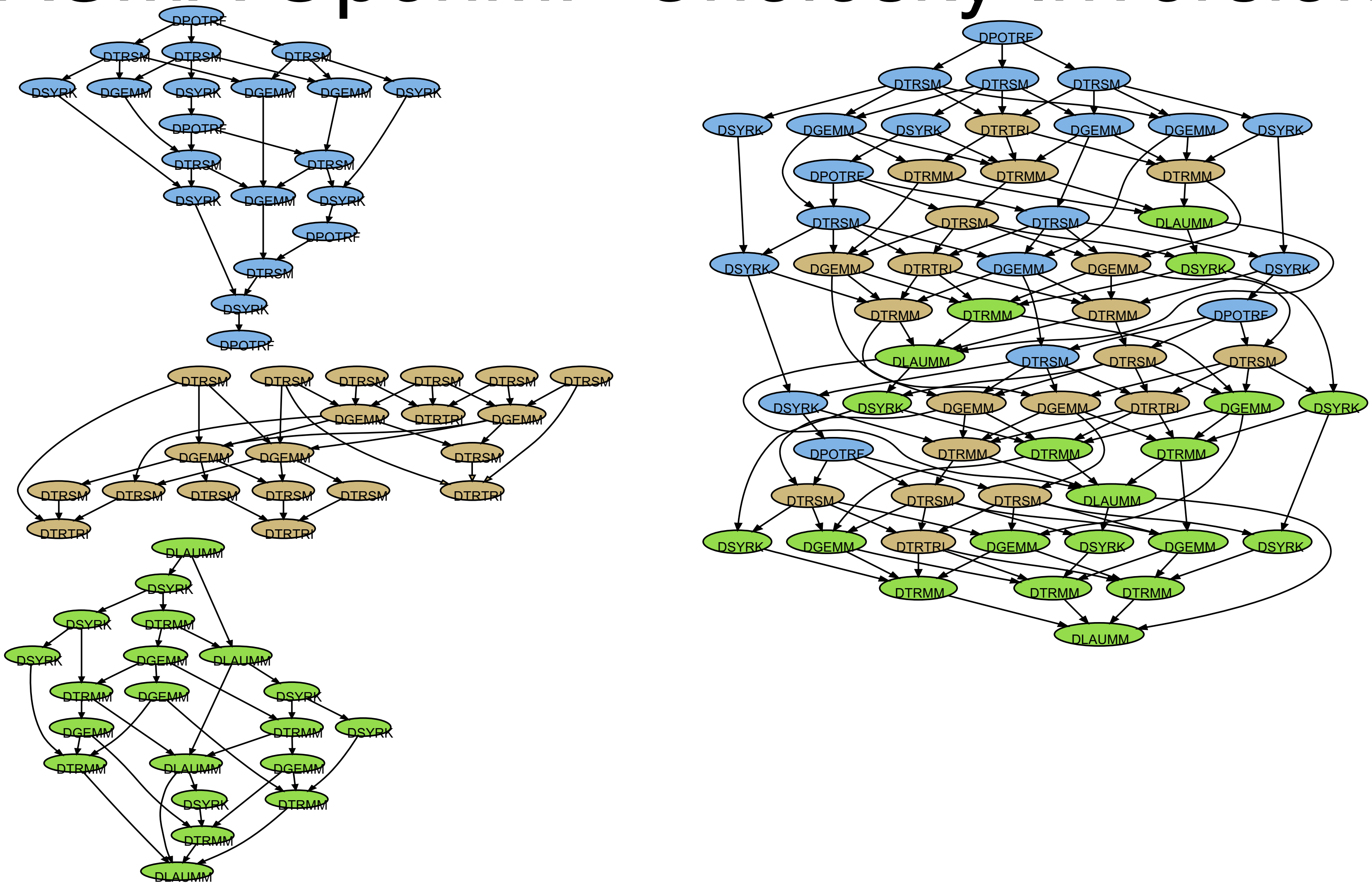


```
plasma_dpotrf(uplo, n, pA, lda);
plasma_dlauum(uplo, n, pA, lda);
plasma_dtrtri(uplo, diag, n, pA, lda);
```



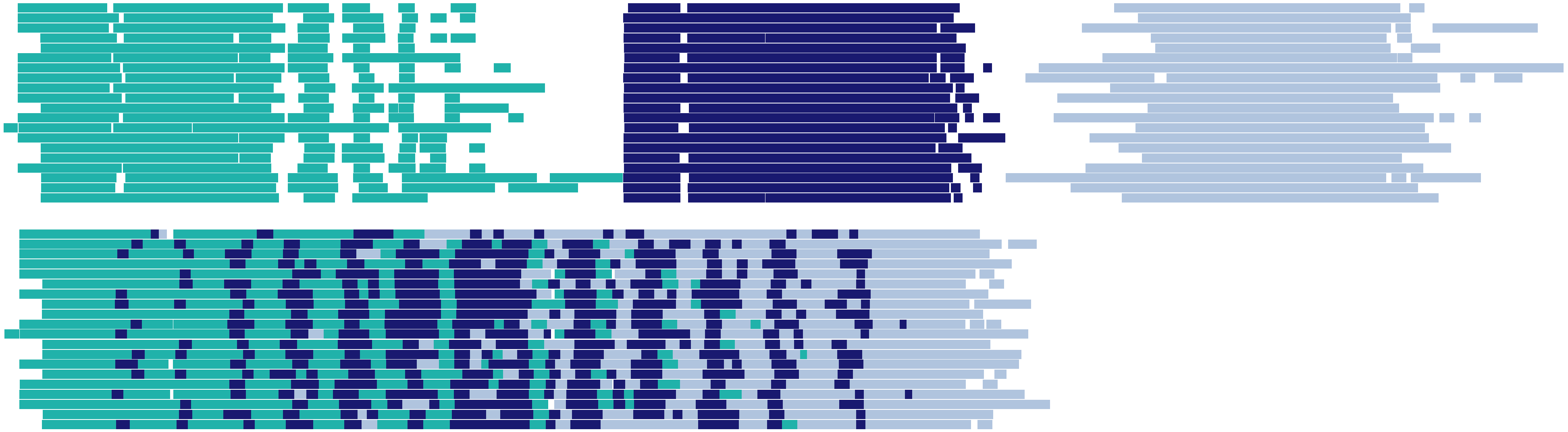
PLASMA Cholesky inversion using OpenMP
Intel Xeon E5-2650 v3 (Haswell) 2.3GHz 20 cores
tiles of size 224 x 224, matrix of size 13 x 13 tiles (2912 x 2912)

PLASMA OpenMP Cholesky Inversion DAG



PLASMA OpenMP Cholesky Inversion

PLASMA Cholesky inversion using OpenMP
Intel Xeon E5-2650 v3 (Haswell) 2.3GHz 20 cores
tiles of size 224 x 224, matrix of size 13 x 13 tiles (2912 x 2912)



Use Case: SLATE

SLATE: dense and low-rank algorithms

- Large scope in terms of hardware
  - Multicore
    - ARM, POWER, x86
  - GPUs
    - CUDA, HIP, DPC++
  - Distributed memory
    - MPI (multithreaded)
- Large algorithmic scope
  - Dense algorithms: linear, least-squares, eigenvalue, SVD
  - Matrix types and storages: rectangular, square, triangular, trapezoidal
  - Low-rank algorithms (ACA, low-rank tiles, …)
- OpenMP coordinates MPI, cores, and devices

OpenMP®

SC'20 Booth Talk Series

openmp.org OpenMP API specs, forum, reference guides, and more

link.openmp.org/sc20 Videos and PDFs of OpenMP SC'20 presentations