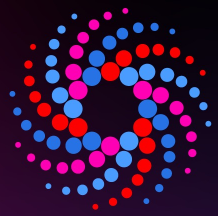


SC22
Dallas, TX | hpc accelerates.

LaRIS: Targeting Portability and Productivity for LAPACK Codes on Extreme Heterogeneous Systems by Using IRIS

Mohammad Alaul Haque Monil, Narasinga Rao Miniskar, Frank Y. Liu, Jeffrey S. Vetter, Pedro Valero-Lara

Computer Science and Mathematics Division, Oak Ridge National Laboratory

Presenter: Mohammad Alaul Haque Monil

RSDHA Workshop, 2022

Introduction

- Heterogeneous systems come in variety
  - Ensuring portability and utilization are challenging
- Tuned vendor libraries for common kernels are not portable
  - Linear algebra kernels
- Making tiled linear algebra algorithm portable is a challenge
- There are solutions that provides tight coupling between algorithm and kernels
  - Not productive; changing source for every library support
- A separation between algorithm development and vendor library kernels is needed

What's Missing?

- Portability
  - Raja
  - Kokkos
  - OpenMP offload
- Heterogeneity
  - StarPU
    - Does not support hip/rocBLAS
    - Tightly coupled
  - MAGMA (CPU + GPU*)
    - hipMAGMA (same)
- Linear Algebra
  - PLASMA (CPU)
  - libFLAME (CPU)
  - Intel MKL (CPU)
  - ATLAS (CPU)
  - cuBLAS/cuSolver (GPU)
  - Hip/roc BLAS/Solver (GPU)
  - Others (Chameleon, etc.)

- Missing?
  - A solution that combines portability and multi-device heterogeneity for LAPACK codes

Research Question

Can we separate linear algebra algorithms from kernels to make seamless portability and utilization possible for different heterogeneous systems?

This Research

- A proof of concept - focusing on **portability and heterogeneity**
- LaRIS – LAPACK on IRIS runtime
- A test case -- considering LU factorization

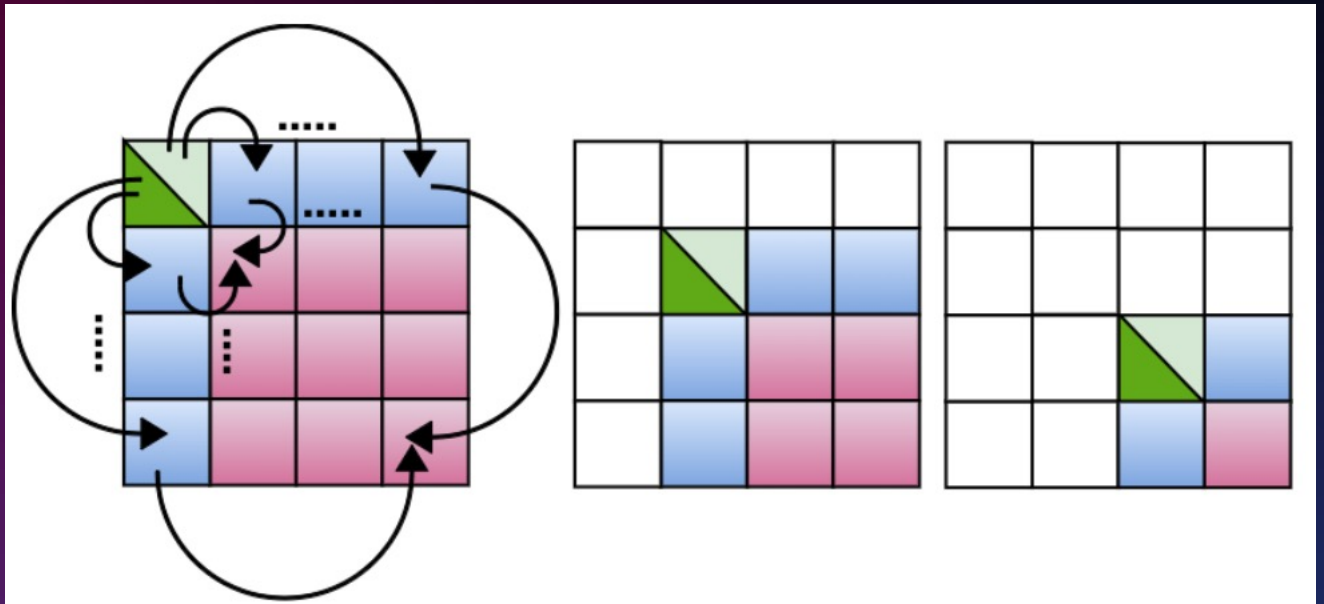
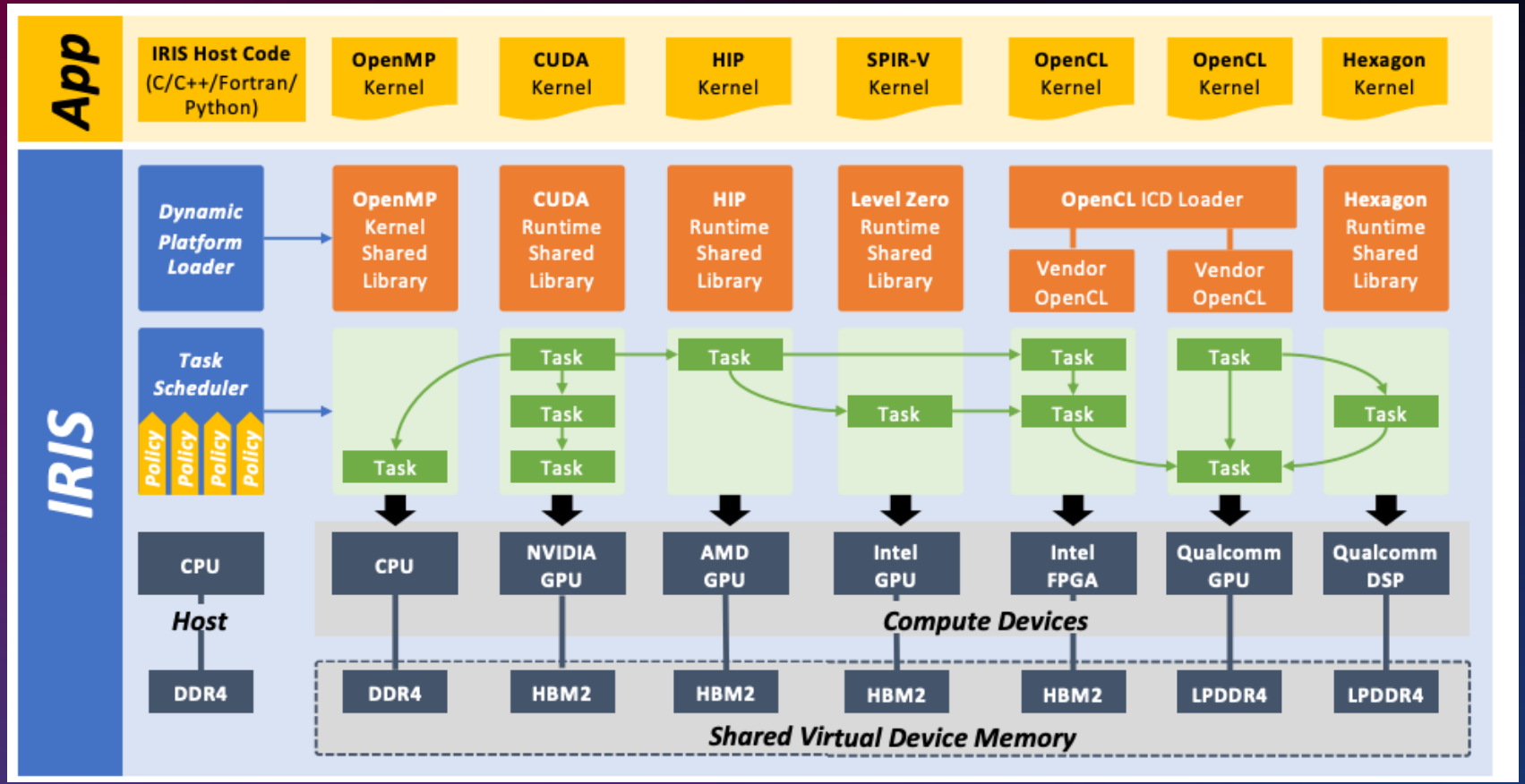


Figure 2: Tiled LU decomposition scheme [20].

[20] P. Valero-Lara, S. Catalan, X. Martorell, T. Usui, and J. Labarta. slass: ´ A fully automatic auto-tuned linear algebra library based on openmp extensions implemented in ompss (lass library). J. Parallel Distributed Comput., 138:153–171, 2020

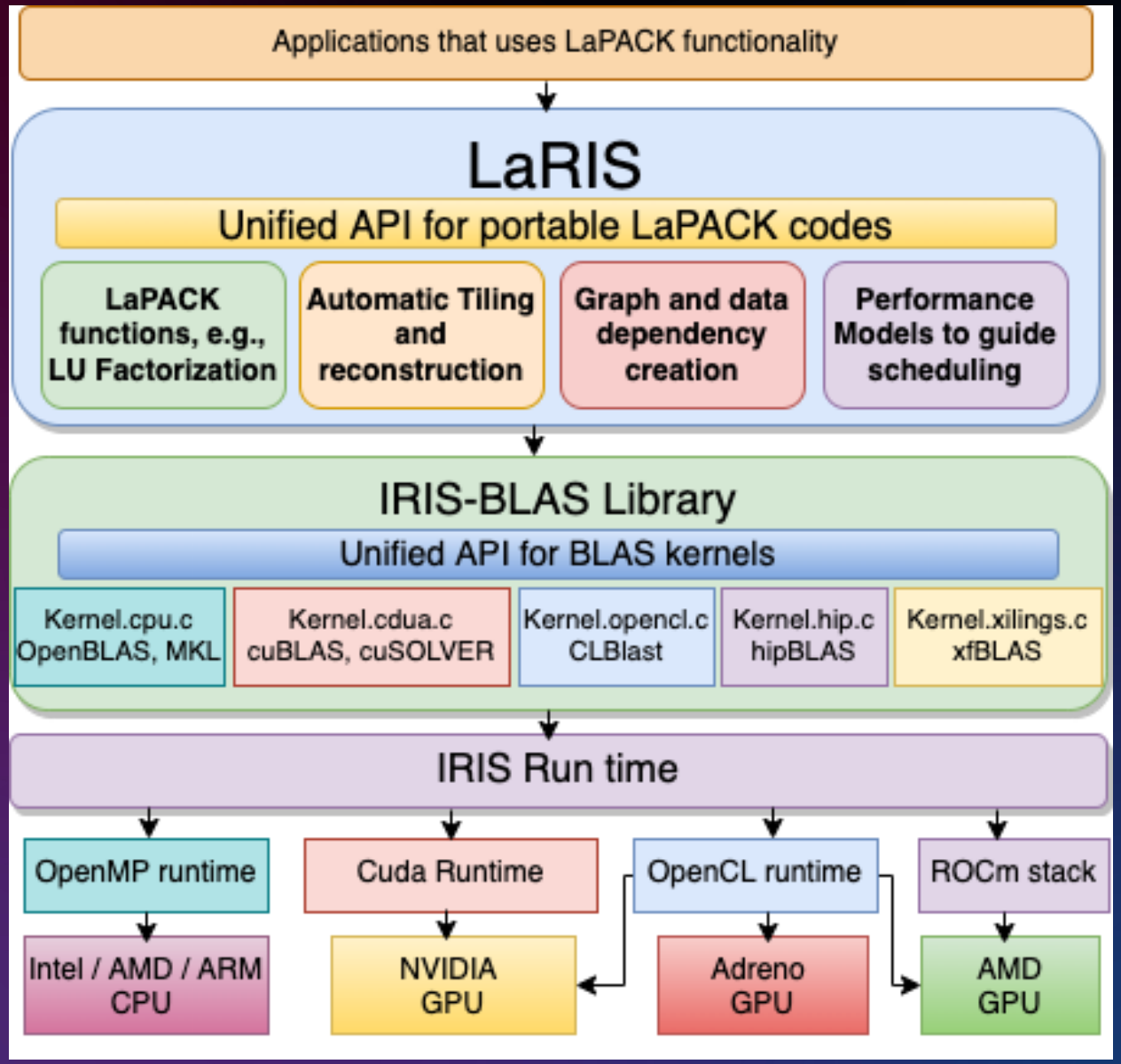
IRIS Runtime

- IRIS Runtime
  - Heterogeneous runtime
  - Supports CPU, GPU, FPGA and DSP
  - Orchestrate computation dynamically
- IRIS-BLAS
  - BLAS library API for different vendor libraries
  - Supports
    - hip/rocBLAS
    - cuBLAS
    - OpenBLAS
    - MKL
    - Under development



LaRIS Software Stack

- LaRIS: Unified API for portable LaPACK codes
  - Targets Four capabilities
  - Algorithm links vendor tuned kernels at runtime
    - Facilitated by IRIS runtime and IRIS-BLAS



LaRIS: LU Factorization

- Three types of kernels
  - GETRF - Red
  - TRSM - Yellow
  - GEMM – Blue
- Graph expressed with dependency
- Dynamic scheduling
  - IRIS_ANY
- Automatic tiling capability

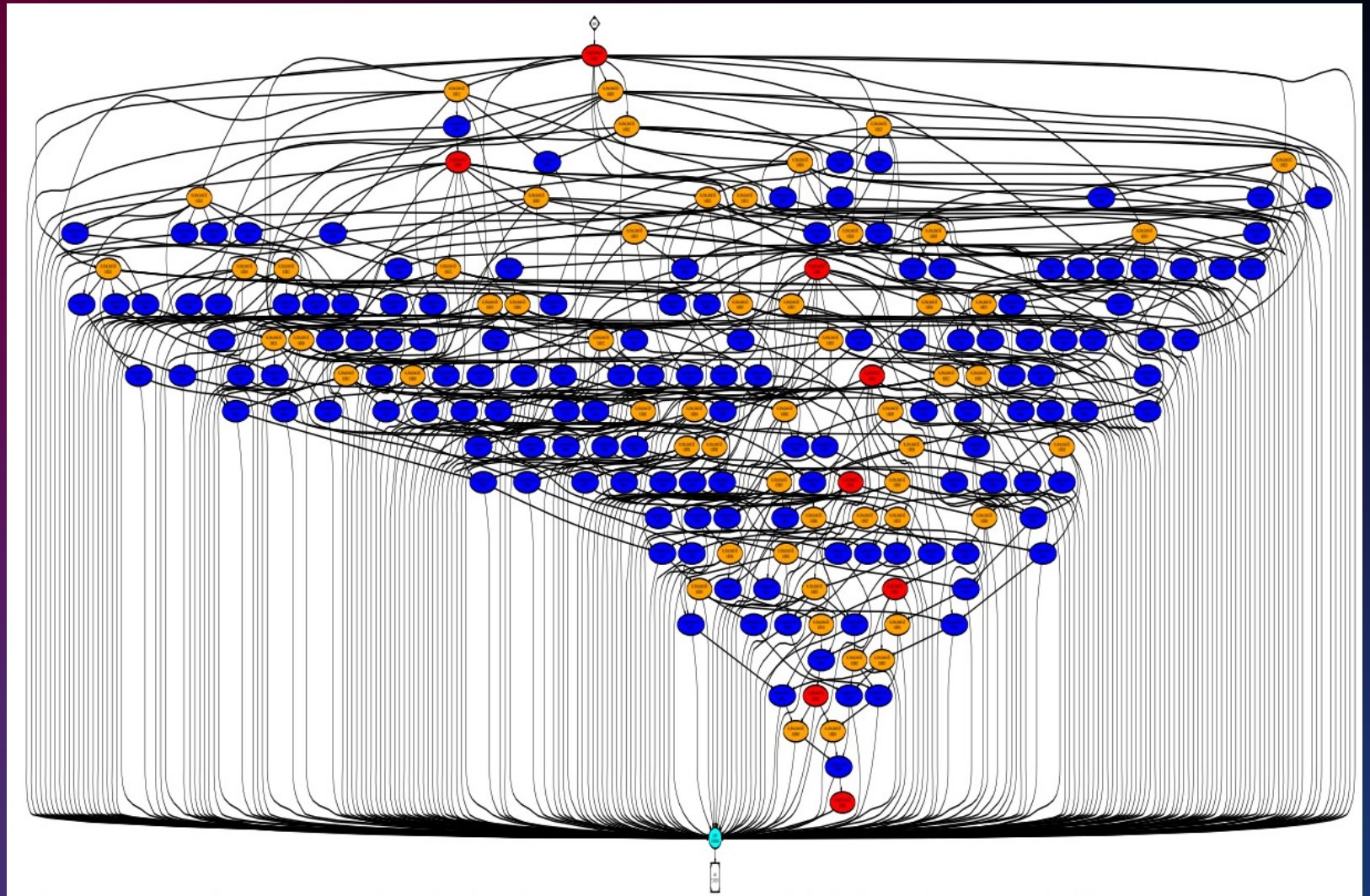


Figure 4: DAG of an 8×8 LU factorization that creates 204 tasks and their dependencies. Red ellipses are GETRF, orange ellipses are TRSM, and blue ellipses are GEMM.

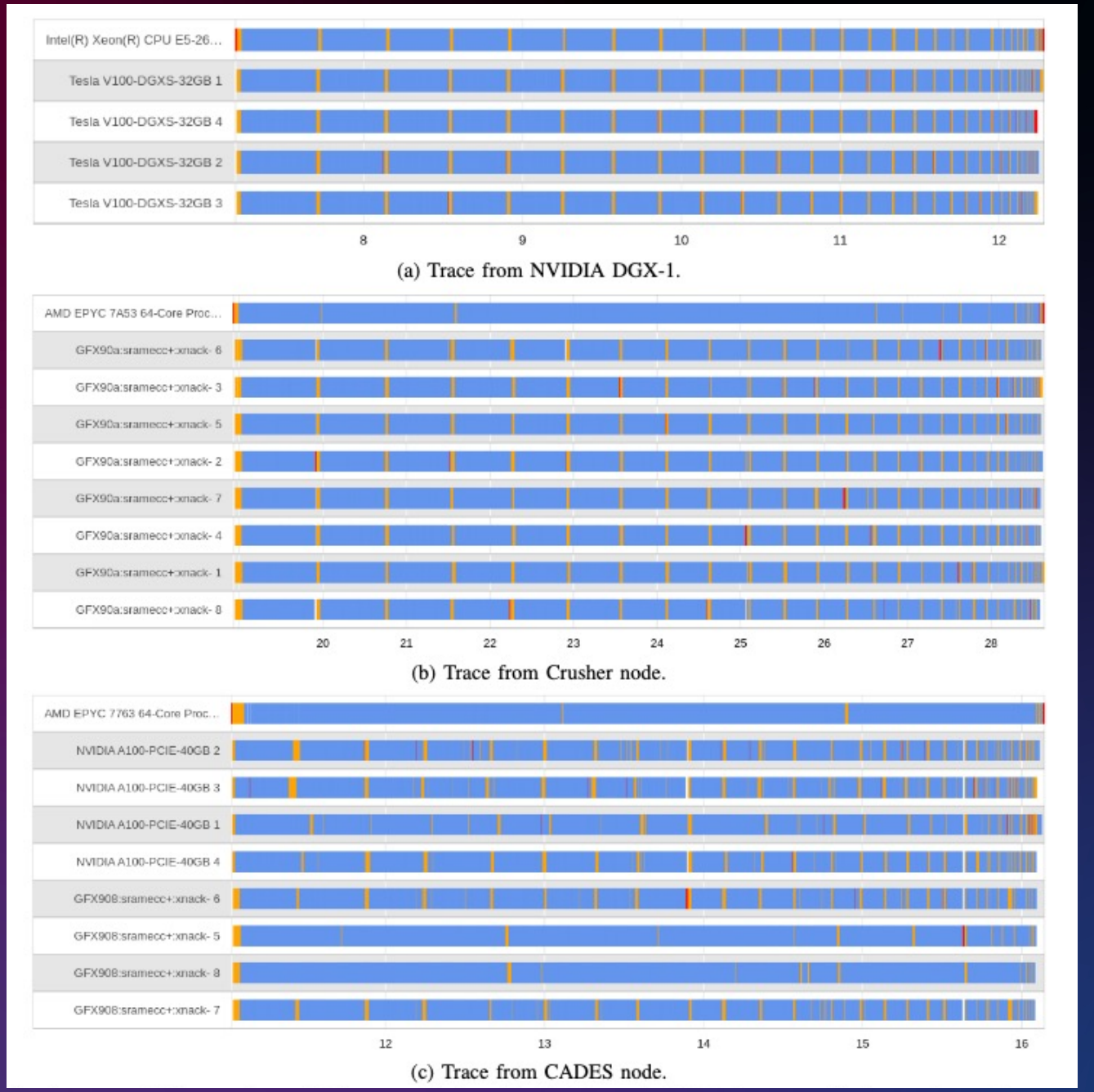
LaRIS: Systems Used

Table I: Heterogeneous systems used in this research

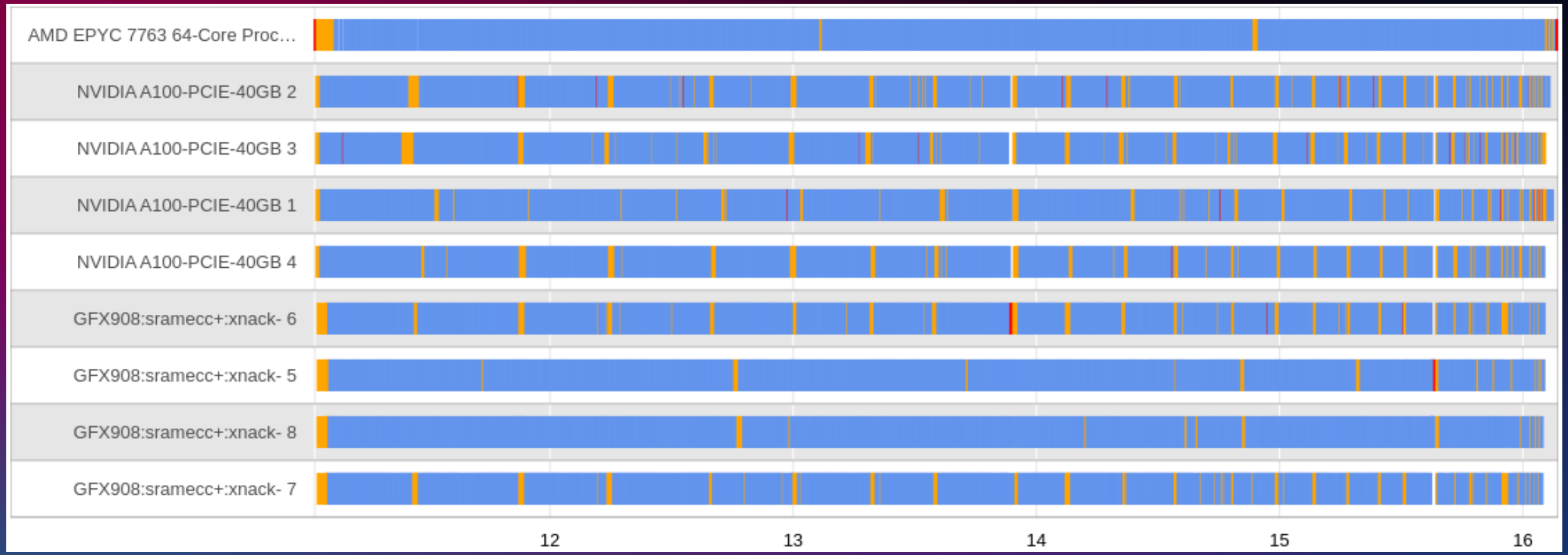
Nodes	NVIDIA DGX-1	Frontier-like node	CADES node
Facility	ExCL at ORNL	Crusher (early access) at ORNL	CADES at ORNL
GPUs	Total 4 GPUs 4× NVIDIA V100	Total 8 GPUs 4× AMD MI250X (each contains two)	Total 8 GPUs 4× NVIDIA A100 4× AMD MI100
CPU	Intel Xeon E5-2698, 20 cores	AMD EPYC 7A53, 64 cores	AMD EPYC 7763, 128 cores
Compiler	GNU-9.4.0	GNU-8.5.0	GNU-8.5.0
CUDA and ROCm versions	CUDA-11.7	CUDA-11.7 and ROCm-5.1.0	CUDA-11.7 and ROCm-5.1.2
Math libraries for GPUs	cuBLAS and cuSOLVER	hipBLAS	cuBLAS, cuSOLVER and hipBLAS
Math libraries for CPU	OpenBLAS-0.3.20 and MKL-2022.1.0	OpenBLAS-0.3.17 and MKL-2020.4.304	OpenBLAS-0.3.20 and MKL-2020.4.304

LaRIS: Portability

- 16384 x 16384 matrix
- 32 x 32 tile = 11440 tasks
- Portability
  - CPUs (AMD and Intel)
  - GPUs
    - NVIDIA (A100 and V100)
    - AMD (MI100 and MI250X)
- Colors
  - GETRF - Red
  - TRSM - Yellow
  - GEMM – Blue

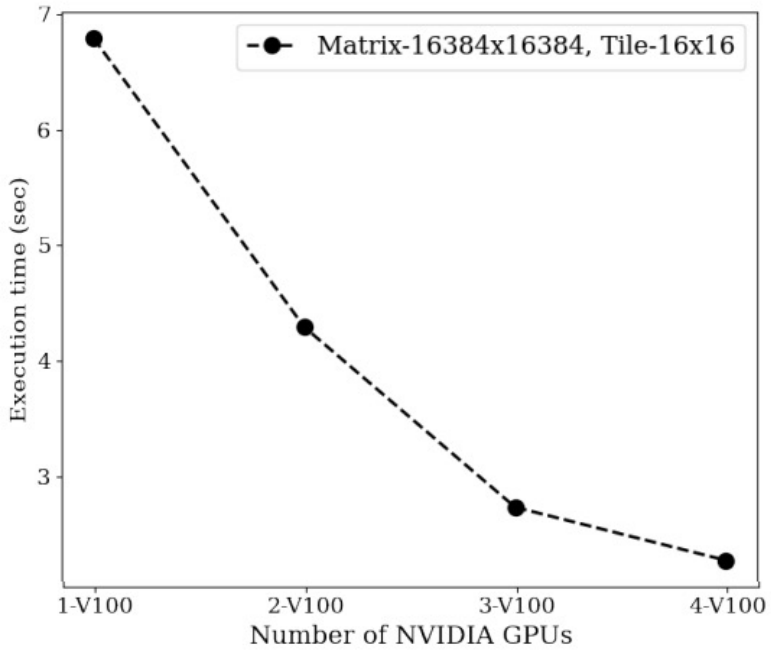


LaRIS: Zoomed-in Trace of the Cades Node

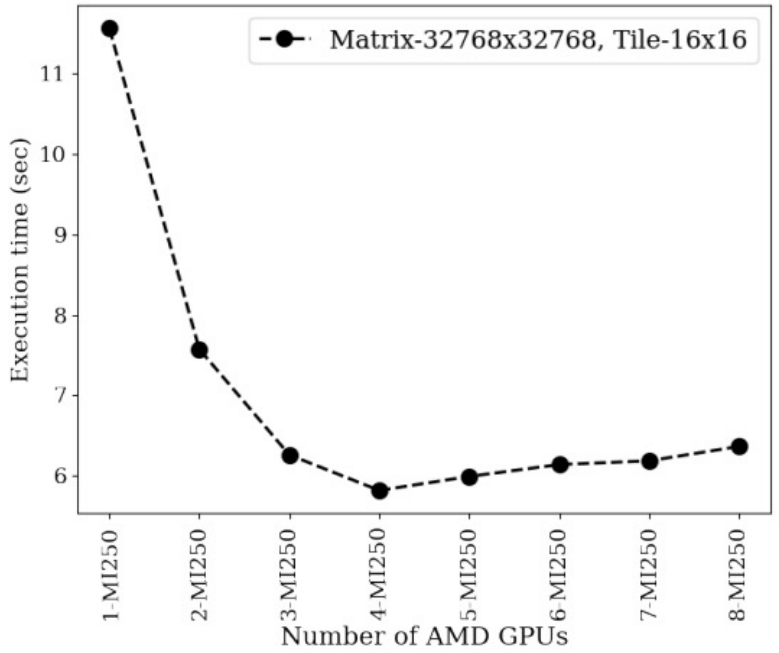


- Main Message: Trace shows the utilization of all the heterogeneous processors without much gap in the timeline

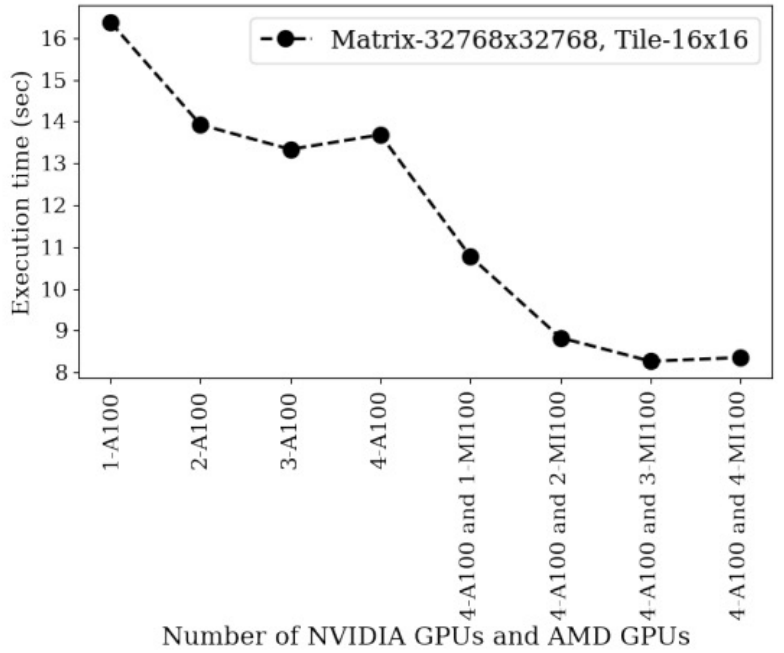
LaRIS: Strong Scaling



(a) NVIDIA DGX-1 with four V100 GPUs.



(b) Crusher node with eight MI250X GPUs.

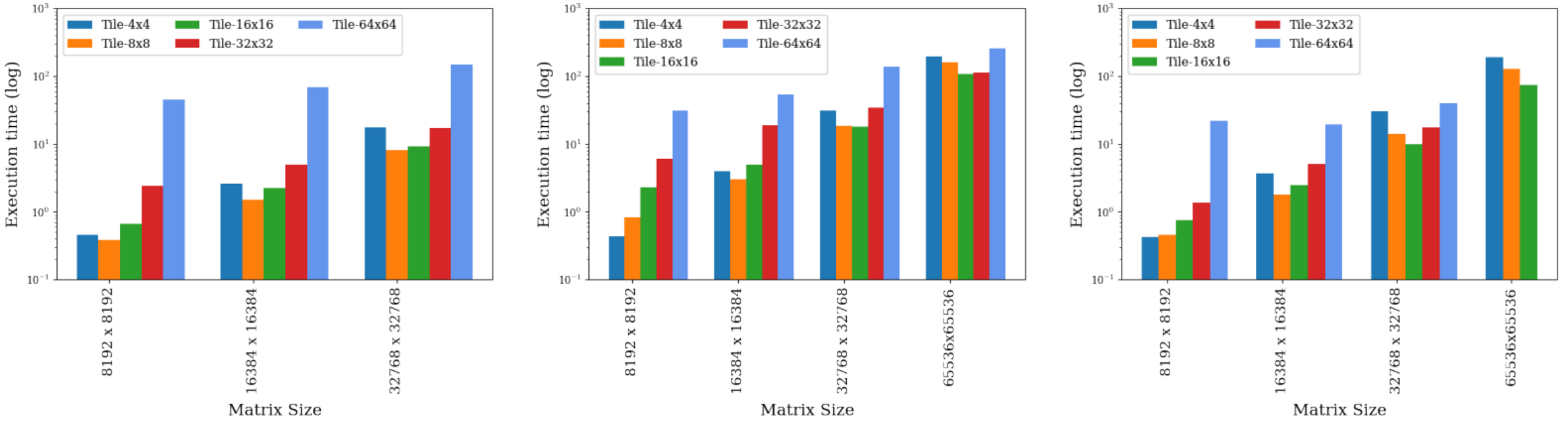


(c) CADES node with four NVIDIA A100 GPUs and four AMD MI100 GPUs.

Figure 6: Strong scaling on different machines.

- Main Message: Increased number of processors provide better performance, however, reaches a saturation. Why?
- Tradeoff between computation capability and overhead

LaRIS: Decomposition vs Matrix Sizes



(a) NVIDIA DGX-1. (b) Crusher node. (c) CADES node.

Figure 7: Finding the best tile configuration for different matrix sizes when all the processing elements are used.

- Main Message: Different system has different best tile decomposition that depends on Matrix sizes.
- Trade-off between parallelism vs overhead

64 x 64 ~ Total 90,000 task and kernel calls

Table II: Best tile configuration for different matrix sizes

Matrix	DGX-1	Crusher node	CADES node
$8,192 \times 8,192$	8×8	4×4	4×4
$16,384 \times 16,384$	8×8	8×8	8×8
$32,768 \times 32,768$	8×8	16×16	16×16
$65,536 \times 65,536$	NA	16×16	16×16

Discussion and Future Opportunities

- This work focuses on portability and heterogeneity
  - Kernel level performance is ensured by the tuned vendor library
  - Optimization opportunities at runtime level
- Reduction of runtime overhead
  - Memory transfer optimization
  - Scheduling algorithm improvement
- Fresh from the Oven
  - We are working on both of these optimization and preliminary result is promising (unpublished)
  - We are working towards a public release as MatRIS
    - Combines LaRIS and IRIS-BLAS on top of IRIS

Conclusion

- LaRIS separates LAPACK algorithms from vendor specific libraries
- Seamless portability is obtained in LaRIS through IRIS runtime and IRIS-BLAS
- IRIS runtime schedular can utilize all the processors in a heterogeneous system to compute LU factorization
- Portability and heterogeneity is demonstrated as a proof of concept, however, there is scope for performance improvement.

Acknowledgments

This research was supported by DOE BLUESTONE
and ECP PROTEAS-TUNE project

Thank you. Question?

```
int laris_gemm_graph(
    iris_graph graph,
    iris_task T,
    int step, int tile_i, int tile_j,
    int M, int N, int K,
    TYPE ALPHA, iris_mem d_A, TYPE *h_A, int LDA,
    iris_mem d_B, TYPE *h_B, int LDB,
    TYPE BETA, iris_mem d_C, TYPE *h_C, int LDC )
{
  // Memory communication -> input
  iris_task_h2d(T, d_A, 0, M*K*sizeof(TYPE), h_A);
  iris_task_h2d(T, d_B, 0, K*N*sizeof(TYPE), h_B);
  iris_task_h2d(T, d_C, 0, M*N*sizeof(TYPE), h_C);
  // IRIS-BLAS call
  iris_core_nodt_gemm( T, IRIS_BLAS_COL_MAJOR,
        IRIS_BLAS_NO_TRANS, IRIS_BLAS_NO_TRANS,
        M, N, K, ALPHA, d_A, LDA,
        d_B, LDB,BETA, d_C, LDC);
  // Memory communication -> output
  iris_task_d2h(T, d_C, 0, M*N*sizeof(TYPE), h_C);
  iris_graph_task( graph, T, iris_any, NULL );
  return 0;
}
```

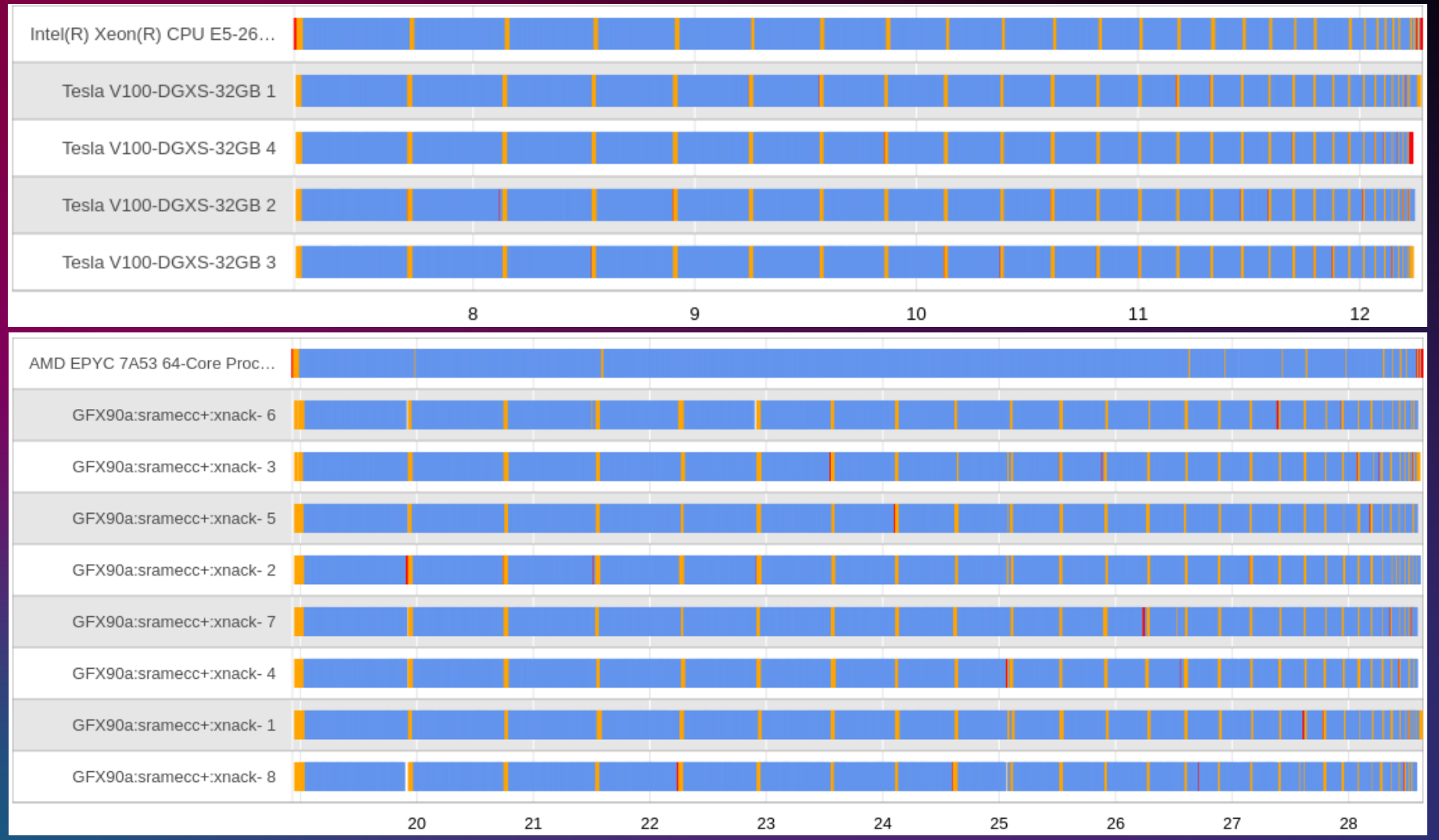
Listing 2: LaRIS's GEMM code.

```
// Creation of the IRIS graph
iris_graph graph; iris_graph_create(&graph);
iris_task getrf_tasks[TILE_NUM*TILE_NUM],
    trsm_tasks[TILE_NUM*TILE_NUM],
    gemm_tasks[TILE_NUM*TILE_NUM*TILE_NUM];
// Tiled LU factorization
for ( int step = 0; step < TILE_NUM; step++ ) {
  // GETRF tasks
  iris_task_create(
    &getrf_tasks[(step*TILE_NUM)+step]);
  // Dependencies for getrf tasks
  if ( step > 0 ){
    // Encapsulate trsm tasks into the graph
    laris_trsm_top_graph( graph, trsm_tasks[(step*
    TILE_NUM)+
          tile_jj], step, step, tile_jj,
          TILE_SIZE, TILE_SIZE,
          IRIS_Ctile[step*TILE_NUM+step],
          A_tile[step][step], TILE_SIZE,
          IRIS_Ctile[step*TILE_NUM+tile_jj],
          A_tile[step][tile_jj], TILE_SIZE );
  }
  for ( int tile_ii = 1 + step; tile_ii < TILE_NUM;
        tile_ii++ ) {
    // trsm tasks
    iris_task_create(
        &trsm_tasks[(tile_ii*TILE_NUM)+step] );
    // Dependencies for trsm tasks
    if ( step == 0 ) {
      iris_task trsm_depend_tasks[] = {
        getrf_tasks[ ( step * TILE_NUM ) + step ]};
        iris_task_depend( trsm_tasks[ ( tile_ii *
        TILE_NUM ) + step ], 1, trsm_depend_tasks );
    } else {
      iris_task trsm_depend_tasks[] = {
        getrf_tasks[(step*TILE_NUM)+step],
        gemm_tasks[((step-1)*TILE_NUM*TILE_NUM)+
        (tile_ii*TILE_NUM)+step] };
      iris_task_depend(
        trsm_tasks[ ( tile_ii * TILE_NUM ) + step ],
        2, trsm_depend_tasks );
    }
```

```
    // Encapsulate trsm tasks into the graph
    laris_trsm_top_graph( graph, trsm_tasks[(step*
    TILE_NUM)+
          tile_jj], step, step, tile_jj,
          TILE_SIZE, TILE_SIZE,
          IRIS_Ctile[step*TILE_NUM+step],
          A_tile[step][step], TILE_SIZE,
          IRIS_Ctile[step*TILE_NUM+tile_jj],
          A_tile[step][tile_jj], TILE_SIZE );
  }
  for ( int tile_ii = 1 + step; tile_ii < TILE_NUM;
        tile_ii++ ) {
    // trsm tasks
    iris_task_create(
        &trsm_tasks[(tile_ii*TILE_NUM)+step] );
    // Dependencies for trsm tasks
    if ( step == 0 ) {
      iris_task trsm_depend_tasks[] = {
        getrf_tasks[ ( step * TILE_NUM ) + step ]};
        iris_task_depend( trsm_tasks[ ( tile_ii *
        TILE_NUM ) + step ], 1, trsm_depend_tasks );
    } else {
      iris_task trsm_depend_tasks[] = {
        getrf_tasks[(step*TILE_NUM)+step],
        gemm_tasks[((step-1)*TILE_NUM*TILE_NUM)+
        (tile_ii*TILE_NUM)+step] };
      iris_task_depend(
        trsm_tasks[ ( tile_ii * TILE_NUM ) + step ],
        2, trsm_depend_tasks );
    }
    // Encapsulate trsm tasks into the graph
    laris_trsm_left_graph( graph,
        trsm_tasks[(tile_ii*TILE_NUM)+step],
        step, tile_ii, step, TILE_SIZE, TILE_SIZE,
        IRIS_Ctile[step * TILE_NUM + step],
        A_tile[step][step], TILE_SIZE,
        IRIS_Ctile[tile_ii * TILE_NUM + step],
        A_tile[tile_ii][step], TILE_SIZE );
    }
    for ( int tile_i = step + 1; tile_i < TILE_NUM;
          tile_i++ ) {
      for ( int tile_j = step + 1; tile_j < TILE_NUM
      ;
            tile_j++ ) {
        // gemm tasks
        iris_task_create( &gemm_tasks[(step*TILE_NUM
      *TILE_NUM) +
            ((tile_i*TILE_NUM)+tile_j)] );
        // Dependencies for gemm tasks
        if ( step == 0 ) {
          iris_task gemm_depend_tasks[] = {
            trsm_tasks[(tile_i*TILE_NUM)+step],
            trsm_tasks[(step*TILE_NUM)+tile_j] };
          iris_task_depend(
            gemm_tasks[(step*TILE_NUM*TILE_NUM)+((
      tile_i*
            TILE_NUM)+tile_j)], 2, gemm_depend_tasks
       );
        }
        else {
          iris_task gemm_depend_tasks[] = {
              trsm_tasks[(tile_i*TILE_NUM)+step],
      trsm_tasks[(step*TILE_NUM)+tile_j],
              gemm_tasks[((step-1)*TILE_NUM*
      TILE_NUM)+
              ((tile_i*TILE_NUM)+tile_j)] };
          iris_task_depend(
              gemm_tasks[(step*TILE_NUM*TILE_NUM)+
              ((tile_i*TILE_NUM)+tile_j)], 3,
              gemm_depend_tasks );
        }
        // Encapsulate gemm tasks into the graph
        laris_gemm_graph( graph,
            gemm_tasks[(step*TILE_NUM*TILE_NUM)+
            ((tile_i*TILE_NUM)+tile_j)], step,
      tile_i,
            tile_j, TILE_SIZE, TILE_SIZE, TILE_SIZE,
       -1.0,
            IRIS_Ctile[tile_i*TILE_NUM+step],
            A_tile[tile_i][step], TILE_SIZE,
            IRIS_Ctile[step*TILE_NUM+tile_j],
            A_tile[step][ tile_j], TILE_SIZE, 1.0,
            IRIS_Ctile[tile_i*TILE_NUM+tile_j],
            A_tile[tile_i][tile_j], TILE_SIZE );
      }
    }
}
iris_graph_submit(graph, iris_default, 1);
```

Listing 1: LaRIS's tiled LU factorization (GETRF) code.

LaRIS: Crusher and DGX-1



LaRIS: Zoomed in Graph