Tiling Framework for Heterogeneous Computing of Matrix based Tiled Algorithms

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

Computer Science and Mathematics Division

Sawtooth Project 25th Feb 2023

ExHET 2023 Workshop (PPoPP)

miniskarnr@ornl.gov ; {monilm ; velerolarap ; liufy ; veter}@ornl.gov

About Paper

- Proposed Tiling framework with a tiled data structure for heterogenous memory mapping and parameterization to a heterogeneous task specification API
- Integrated into MatRIS (Math kernels library using IRIS runtime)
- Benefits
  - Tiling algorithms for heterogeneous computation
  - Improved programmability of tiled algorithms (Dense GEMM, LAPACK)
    - Less number of lines to write tiling algorithms
  - Improves performance by 20% when compared to traditional methods
    - Enabled to utilize 2D and 3D data transfer APIs (Ex: cuMemcpy2D)
    - No need of tile to flat and flat to tile conversion of data

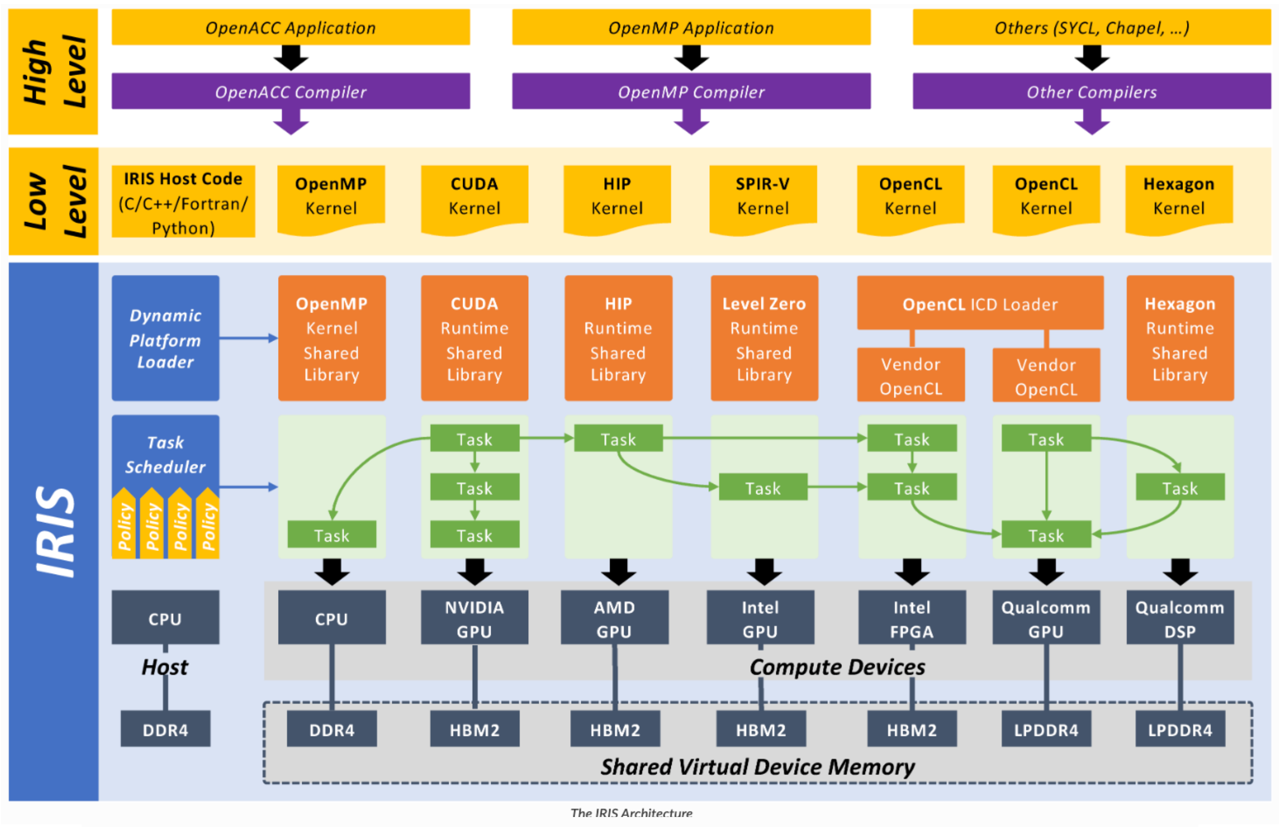
OAK RIDGE National Laboratory

Outline

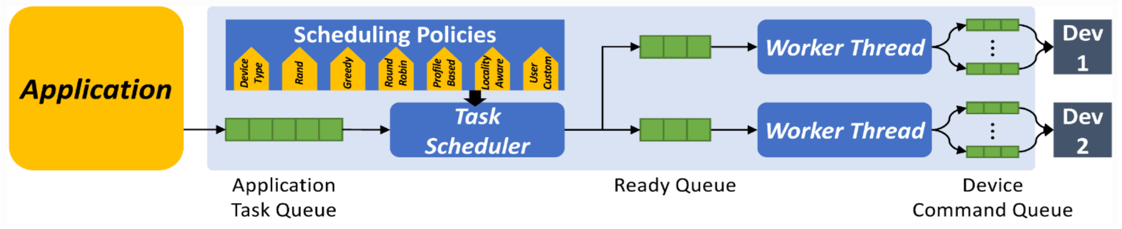
- IRIS Run-time and MatRIS library
- State of the art Tiling approaches
- Proposed Tiling Framework
- Example Tiled Matrix Addition, Multiplication and LU Factorization Algorithms
- Experimental Results
- Conclusion & Future work

IRIS Runtime Framework

https://github.com/ORNL/iris
https://code.ornl.gov/brisbane/iris.git



The IRIS Architecture



- Heterogenous Platform Model
- Task Programming Model
  - Create Synchronous / Asynchronous tasks
  - Add dependencies
  - Ability to create task graphs
- Memory Model
  - Handles heterogenous memory address spaces for application data objects
  - Automatic data movement
  - No need of writing H2D and D2H data transfers
  - Heterogenous data transfer policies
- Execution Model
  - Supports dynamic and static task mapping policies
  - Customization of policies

Kim, Jungwon, Seyong Lee, Beau Johnston, and Jeffrey S. Vetter. "IRIS: A portable runtime system exploiting multiple heterogeneous programming systems." In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1-8. IEEE, 2021.

https://iris-programming.github.io/

IRIS Host and Kernels for SAXPY

Host Code: C++ C Python Fortran

```
int main(int argc, char** argv) {
  size_t SIZE;
  float *X, *Y, *Z;
  float A = 10;
  int ERROR = 0;
  iris::Platform platform;
  platform.init(&argc, &argv, 1);

  SIZE = argc > 1 ? atol(argv[1]) : 8;

  X = (float*) malloc(SIZE * sizeof(float));
  Y = (float*) malloc(SIZE * sizeof(float));
  Z = (float*) malloc(SIZE * sizeof(float));

  for (int i = 0; i < SIZE; i++) {
    X[i] = i; Y[i] = i;
  }

  iris::DMem mem_X(X, SIZE * sizeof(float));
  iris::DMem mem_Y(Y, SIZE * sizeof(float));
  iris::DMem mem_Z(Z, SIZE * sizeof(float));

  iris::Task task;
  void* params0[4] = { &mem_Z, &A, &mem_X, &mem_Y };
  int pinfo0[4] = { iris_w, sizeof(A), iris_r, iris_r };
  task.kernel("saxpy", 1, NULL, &SIZE, NULL, 4, params0, pinfo0);
  task.flush_out(mem_Z);
  task.submit(1, NULL, 1);

  for (int i = 0; i < SIZE; i++) {
    if (Z[i] != A * X[i] + Y[i]) ERROR++;
  }
  free(X);  free(Y); free(Z);

  platform.finalize();
  return 0;
}
```

Kernels

CUDA HIP OpenCL OpenMP Hexagon

```
static void saxpy(float* Z, float A, float* X, float* Y) {
  size_t i;
#pragma omp parallel for shared(Z, A, X, Y) private(i)
  IRIS_OPENMP_KERNEL_BEGIN(i)
  Z[i] = A * X[i] + Y[i];
  IRIS_OPENMP_KERNEL_END
}
```

CUDA HIP OpenCL OpenMP Hexagon

```
extern "C" __global__ void saxpy(float* Z, float A, float* X, float* Y) {
  size_t id = blockIdx.x * blockDim.x + threadIdx.x;
  Z[id] = A * X[id] + Y[id];
}
```

https://iris-programming.github.io/

IRIS Host and Kernels for SAXPY

Host Code: C++ C Python Fortran

```python
import iris
import numpy as np

def saxpy(SIZE=1024):
    iris.init()
    x = np.arange(SIZE, dtype=np.float32)
    y = np.arange(SIZE, dtype=np.float32)
    z = np.zeros(SIZE, dtype=np.float32)

    #SAXPY: Z = AX + Y
    mem_x = iris.dmem(x)
    mem_y = iris.dmem(y)
    mem_z = iris.dmem(z)

    A = 10.0
    task = iris.task("saxpy", 1, [], [SIZE], [], [
        (mem_z, iris.iris_w),
        A,
        (mem_x, iris.iris_r),
        (mem_y, iris.iris_r)
        ])
    task.submit(iris.iris_gpu)
    iris.finalize()
```

Kernels

CUDA HIP OpenCL OpenMP Hexagon

```c
static void saxpy(float* Z, float A, float* X, float* Y) {
  size_t i;
#pragma omp parallel for shared(Z, A, X, Y) private(i)
  IRIS_OPENMP_KERNEL_BEGIN(i)
  Z[i] = A * X[i] + Y[i];
  IRIS_OPENMP_KERNEL_END
}
```

CUDA HIP OpenCL OpenMP Hexagon

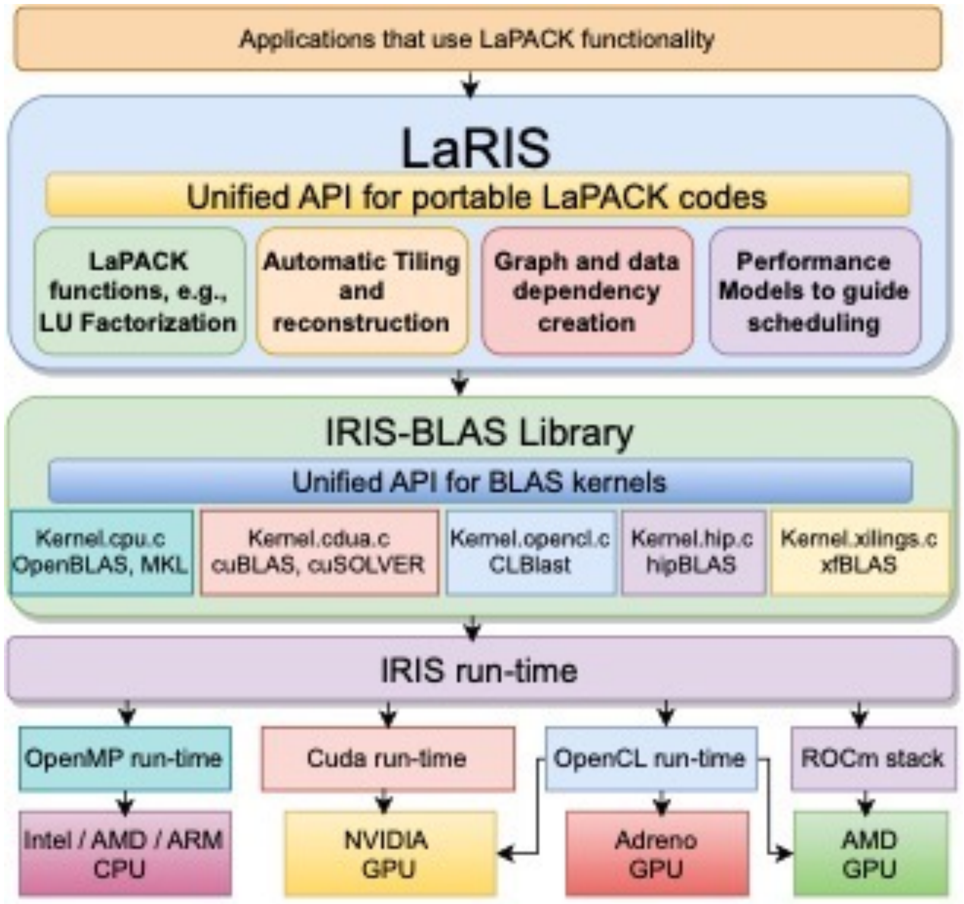
```c
extern "C" __global__ void saxpy(float* Z, float A, float* X, float* Y) {
  size_t id = blockIdx.x * blockDim.x + threadIdx.x;
  Z[id] = A * X[id] + Y[id];
}
```

OAK RIDGE National Laboratory

MatRIS = IRIS-BLAS + LaRIS

- Heterogenous Math Kernels
- IRIS-BLAS
  - cuBLAS for Nvidia GPU
  - hipBLAS for AMD GPU
  - clBLAST for OpenCL GPUs
  - OpenBLAS/MKL for AMD / Intel CPUs
- LaRIS: LaPACK functions
  - Configurable tiles
  - Task Graphs with dependencies
  - Performance models and scheduling policies
- Heterogenous platforms
  - CPU + Nvidia GPUs + AMD GPUs

https://code.ornl.gov/brisbane/matris



Applications that use LaPACK functionality
LaRIS
Unified API for portable LaPACK codes
LaPACK functions, e.g., LU Factorization
Automatic Tiling and reconstruction
Graph and data dependency creation
Performance Models to guide scheduling
IRIS-BLAS Library
Unified API for BLAS kernels
Kernel.cpu.c OpenBLAS, MKL
Kernel.cdua.c cuBLAS, cuSOLVER
Kernel.opencl.c CLBlast
Kernel.hip.c hipBLAS
Kernel.xilings.c xfBLAS
IRIS run-time
OpenMP run-time
Cuda run-time
OpenCL run-time
ROCm stack
Intel / AMD / ARM CPU
NVIDIA GPU
Adreno GPU
AMD GPU

1. LaRIS: Targeting Portability and Productivity for LaPACK Codes on Extreme Heterogeneous Systems using IRIS, SC 2022 RSDHA Workshop, Monil Mohammad, Narasinga Rao Miniskar, Pedro Valero Lara, Frank Liu, Jeffrey Vetter
2. IRIS-BLAS: Towards a performance portable and heterogeneous BLAS Library, HiPC 2022, Narasinga Rao Miniskar, Monil Mohammad, Pedro Valero Lara, Frank Liu, Jeffrey Vetter

State Of the Art Tiling

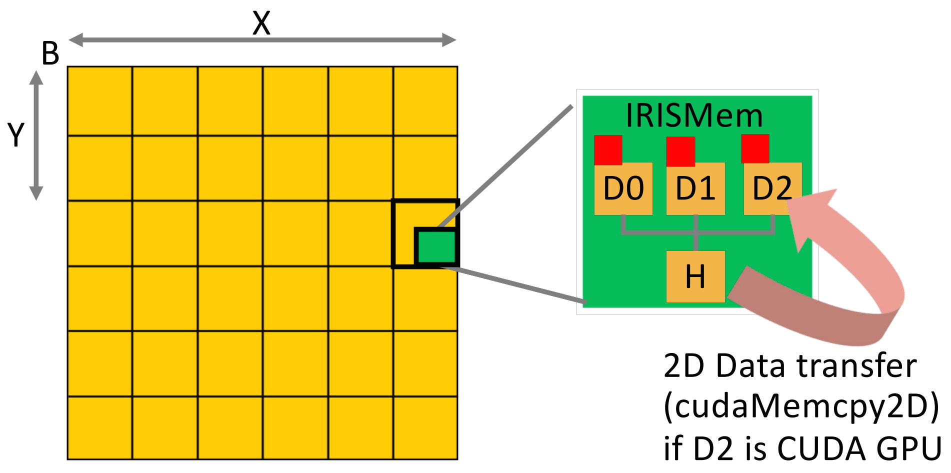
- Traditional: Uses rudimentary for loop iterators, loop indexes and index increments
  - Error prune and requires lot of code
- SOA DSL and frameworks for homogenous compute units
  - Halide, Tiramisu, Triton
- Challenges of tiling for heterogenous computing algorithms
  - Along with tiling utilities,
  - Manage heterogenous memories for tiles
  - Create tasks using tiled inputs and outputs
  - Ease of programmability
  - Use device specific tile data transfer APIs
- Requirement: A tiling framework to handle the above challenges

OAK RIDGE National Laboratory

Tiling Framework

- Tiling Specification
  - Size of the matrix in all dimensions, size of tile, stride between tiles
- Binding to IRIS memory
  - Required for heterogeneous computing
  - Enables tiling data structure to map the tiles to heterogenous memory locations
  - Tiling object manages IRIS memory object for each tile
- Tiling iterators for regularized access of tiles
  - Sequence iterators
    - Row-major iterator
    - Column-major iterator
    - Row-Column/Right-Down Tree wise iterator
    - 3D Iterators (ZYX, ZXY, YXZ, YZX, XYZ, XZY)
  - Index based iterators
    - For random access of tiles or
    - interleaved access of tiles

Mapping of tile to IRIS Memory object

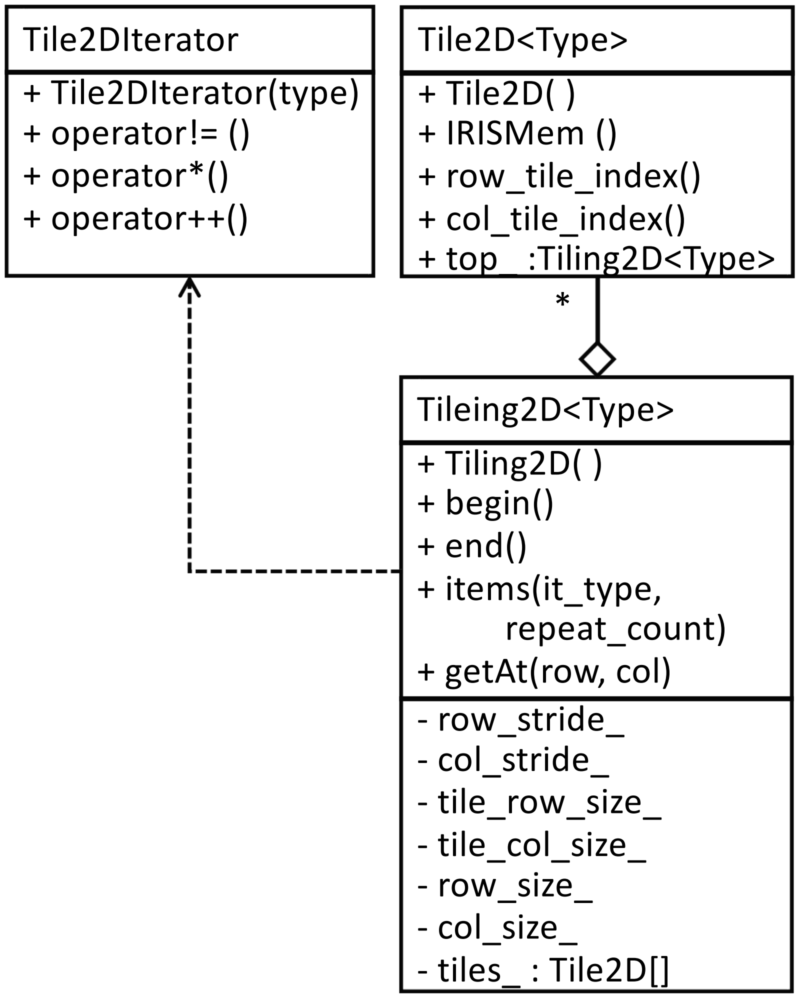


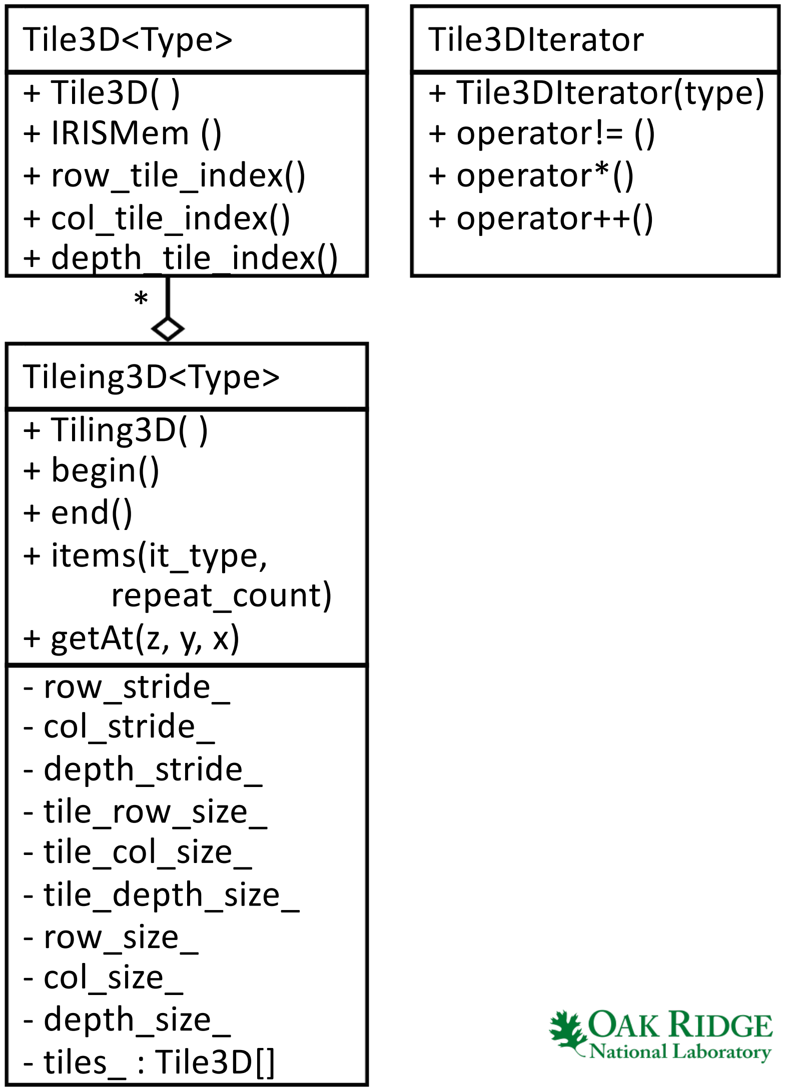
Matrix with Tiles: Tile2D[]

IRISMem Host (H) memory points to Base (B) start address of matrix with 2D offset (X, Y)

- IRIS Memory object maps host memory to device memory and synchronizes data validity and data transfers
- Map 2D/3D tile to IRIS Mem objects

Tiled Data Structure in MatRIS





OAK RIDGE National Laboratory

Tiling iterators

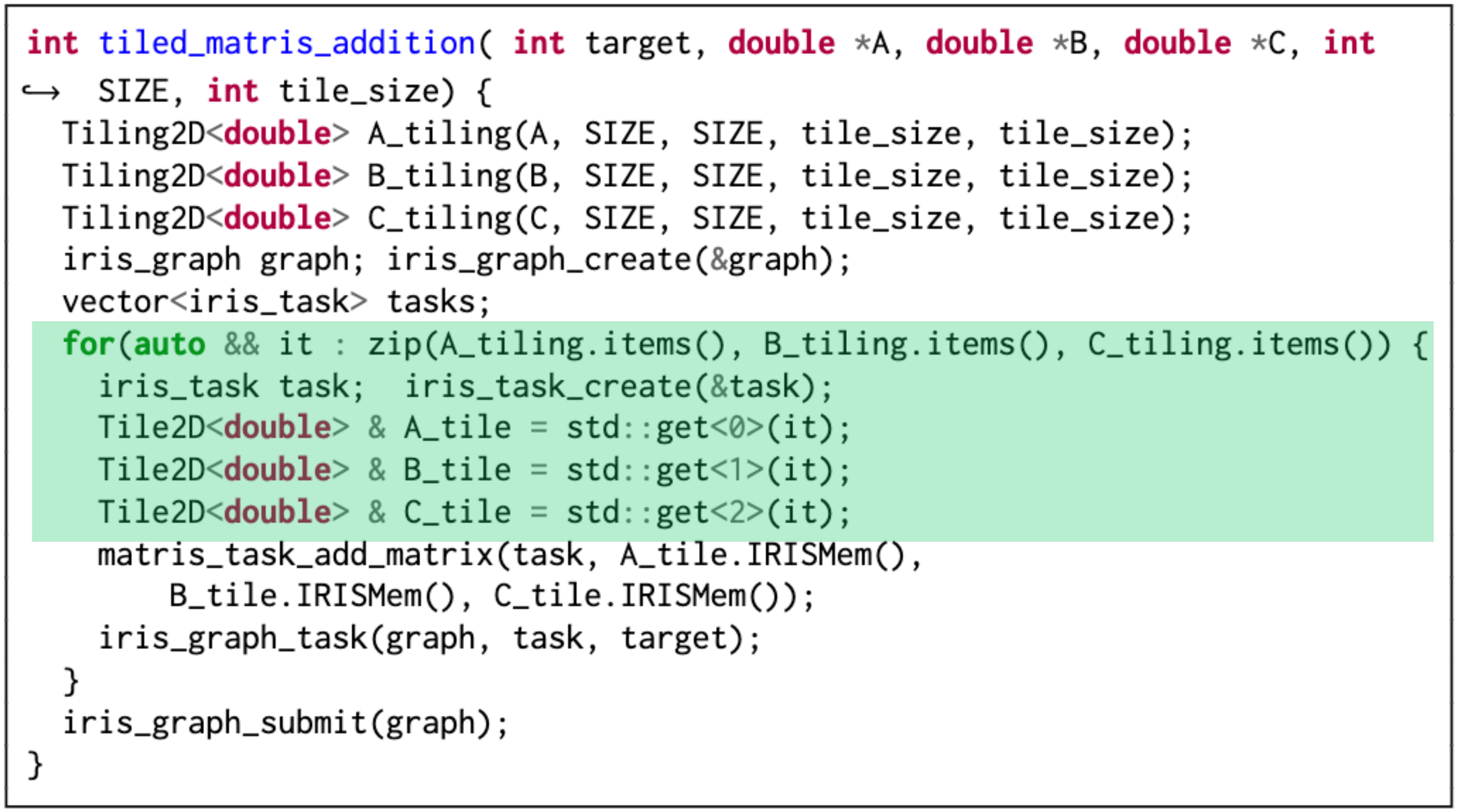
Sequence Iterators:

```
for( auto & a_tile : A_tiling.items( ))  {                    // Default: TILE2D_ROW_MAJOR
 ....  // C++ Range based row-major iterator
}
for( auto & a_tile : A_tiling.items(TILE2D_COL_MAJOR))  {
 ....  // C++ Range based column-najor iterator
}
for( auto & Z : zip(A_tiling.items(), B_tiling.items())  {
  auto & a_tile = std::get<0>(Z);
  auto & b_tile = std::get<1>(Z);
  .... // Multiple C++ Range based iterators
}
```

Index based Iterators:

```
for (size_t i=0; i<A_tiling.row_tiles_count(); i++) {
  ... // Index based iterators
}
```

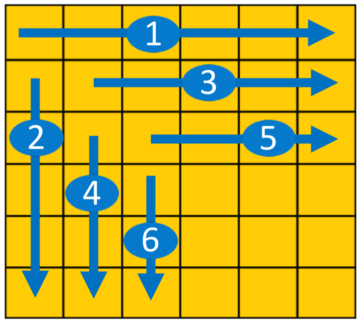
Tiled Matrix Addition with Sequence Iterator



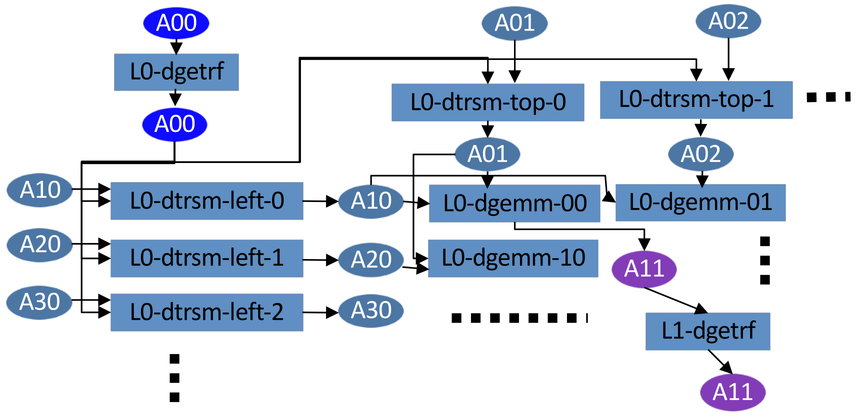
```
int tiled_matris_addition( int target, double *A, double *B, double *C, int
↪  SIZE, int tile_size) {
  Tiling2D<double> A_tiling(A, SIZE, SIZE, tile_size, tile_size);
  Tiling2D<double> B_tiling(B, SIZE, SIZE, tile_size, tile_size);
  Tiling2D<double> C_tiling(C, SIZE, SIZE, tile_size, tile_size);
  iris_graph graph; iris_graph_create(&graph);
  vector<iris_task> tasks;
  for(auto && it : zip(A_tiling.items(), B_tiling.items(), C_tiling.items()) {
    iris_task task;  iris_task_create(&task);
    Tile2D<double> & A_tile = std::get<0>(it);
    Tile2D<double> & B_tile = std::get<1>(it);
    Tile2D<double> & C_tile = std::get<2>(it);
    matris_task_add_matrix(task, A_tile.IRISMem(),
        B_tile.IRISMem(), C_tile.IRISMem());
    iris_graph_task(graph, task, target);
  }
  iris_graph_submit(graph);
}
```

Tiled LU Factorization

- Right-Down /Row-Column Tree wise iterator

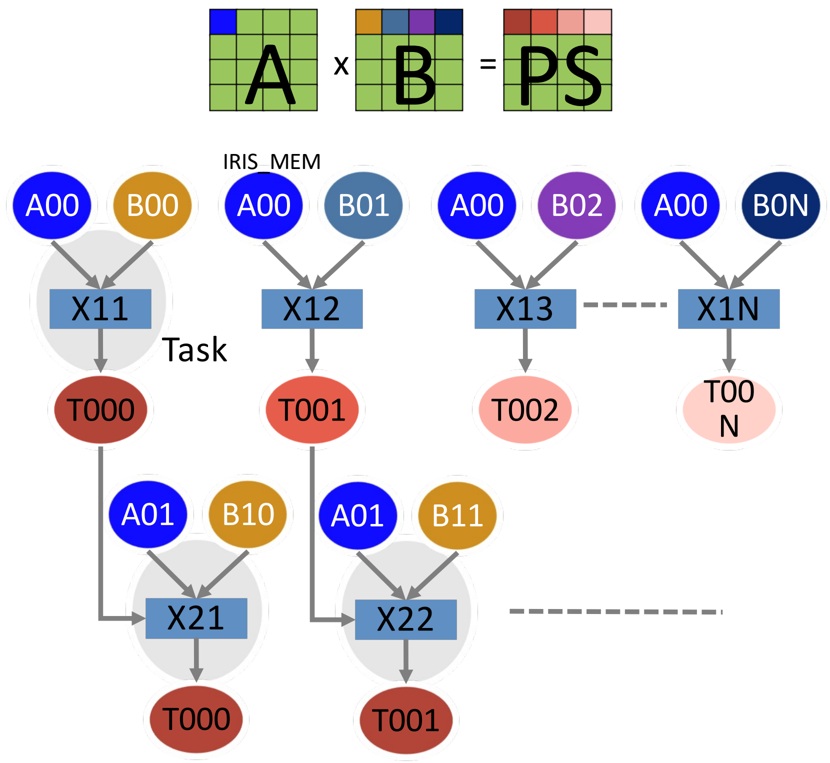


- Task Graph



```
size_t step = 0;
Tiling2D<DTYPE> A_tiling(A, M, N, TILE_SIZE, TILE_SIZE);
for(auto & a_tile : A_tiling.items(TILE2D_RIGHT_DOWN_TREE_WISE)) {
    if (a_tile.row_tile_index() == a_tile.col_tile_index()) {
        // First check whether GEMM has to be applied and proceed with GetRF
        step = a_tile.row_tile_index();
        if (step != 0) for(auto & gemm_tile : A_tiling.items(step, step)) {
            // Do GEMM: Iterate over each GEMM tile of previous step
            size_t tile_jj = gemm_tile.row_tile_index();
            size_t tile_ii = gemm_tile.col_tile_index();
            Tile2D<DTYPE> & left_trsm_tile = A_tiling.GetAt(step-1, tile_ii);
            Tile2D<DTYPE> & top_trsm_tile = A_tiling.GetAt(tile_jj, step-1);
            iris_task task; iris_task_create_name("gemm", &task);
            CALL_GEMM(left_trsm_tile.IRISMem(), top_trsm_tile.IRISMem(),
                    gemm_tile.IRISMem());
            ADD_TASK_DEPENDENCIES(gemm_tile, ...);
        }
        // Now Do GETRF: GETRF of new step
        iris_task_create_name("getrf", &getrf_tasks[step]);
        CALL_GETRF(getrf_tasks[step], a_tile.IRISMem());
        ADD_TASK_DEPENDENCIES(getrf_tasks[steps], ...);
    }
    else if (a_tile.row_tile_index() == step) {
        // Do LEFT TRSM (Assuming it is transformed)
        size_t tile_ii = a_tile.col_tile_index();
        Tile2D<DTYPE> & getrf_tile = A_tiling.GetAt(step, step);
        iris_task_create_name("trsm-left", &left_trsm_tasks[tile_ii]);
        CALL_TRSM_LEFT(left_trsm_tasks[tile_ii], getrf_tile.IRISMem(), a_tile.IRISMem());
        ADD_TASK_DEPENDENCIES(left_trsm_tasks[tile_ii], ...);
    }
    else if (a_tile.col_tile_index() == step) {
        // Do TOP TRSM (Assuming it is transformed)
        size_t tile_jj = a_tile.row_tile_index();
        Tile2D<DTYPE> & getrf_tile = A_tiling.GetAt(step, step);
        iris_task_create_name("trsm-top", &top_trsm_tasks[tile_jj]);
        CALL_TRSM_TOP(top_trsm_tasks[tile_jj], getrf_tile.IRISMem(), a_tile.IRISMem());
        ADD_TASK_DEPENDENCIES(top_trsm_tasks[tile_jj], ...);
    }
}
```

Tiled Matrix Multiplication

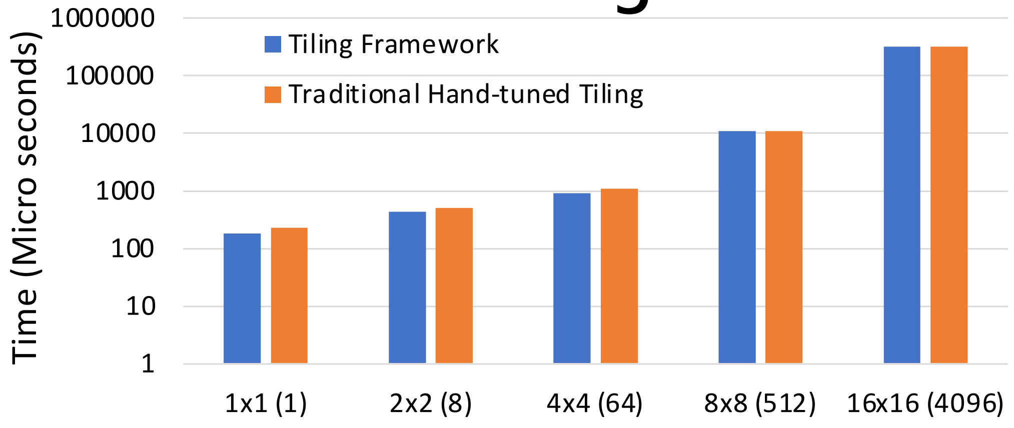


Algorithm	Traditional Tiling Lines	Proposed Tiling Lines	Line reduction
Tiled GEMM	230	50	78%
Tiled LU Factorization	125	68	45%

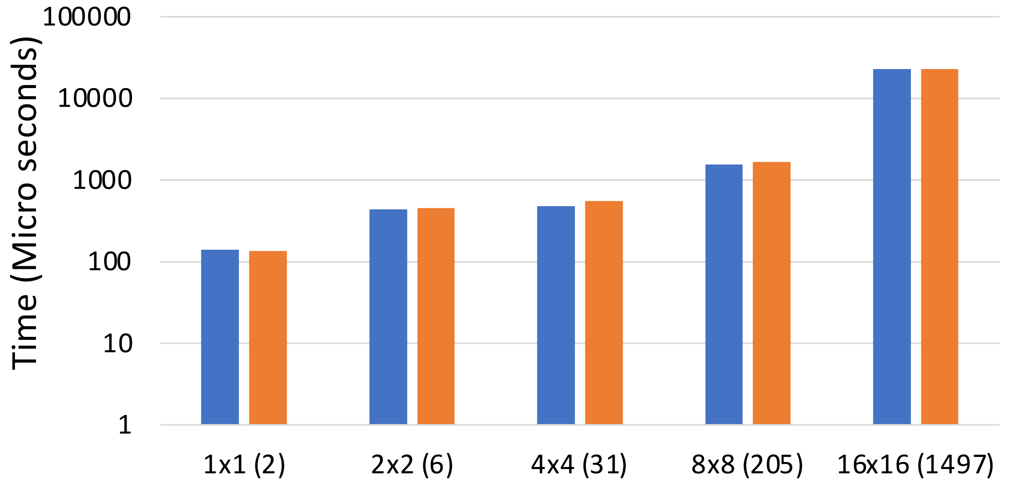
Index based iterator

```
int tiled_matris_multiplication( int target, double *A, double *B, double *C,
  int SIZE, int tile_size) {
    Tiling2D<DTYPE> A_tiling(A, SIZE, SIZE, tile_size, tile_size);
    Tiling2D<DTYPE> B_tiling(B, SIZE, SIZE, tile_size, tile_size);
    Tiling2D<DTYPE> C_tiling(C, SIZE, SIZE, tile_size, tile_size);
    iris_graph graph; iris_graph_create(&graph);
    for(size_t i=0; i<A_tiling.row_tiles_count(); i++) {
        for(size_t j=0; j<B_tiling.col_tiles_count(); j++) {
            Tile2D<DTYPE> & c_tile = C_tiling.getAt(i, j);
            iris_task prev_task = NULL;
            for(size_t k=0; k<B_tiling.row_tiles_count(); k++) {
                Tile2D<DTYPE> & a_tile = A_tiling.getAt(i, k);
                Tile2D<DTYPE> & b_tile = B_tiling.getAt(k, j);
                iris_task task;  iris_task_create(&task);
                matris_task_dgemm
                    (task, MATRIS_ROW_MAJOR, MATRIS_NO_TRANS, MATRIS_NO_TRANS,
                     a_tile.row_tile_size(), b_tile.row_tile_size(),
                     b_tile.col_tile_size(),
                     1.0f, a_tile.IRISMem(), tile_size,
                     b_tile.IRISMem(), tile_size,
                     1.0f, c_tile.IRISMem(), tile_size);
                if (prev_task != NULL) {
                    iris_task gemm_depend_tasks[] =  { prev_task };
                    iris_task_depend(task, 1, gemm_depend_tasks);
                }
                iris_graph_task(graph, task, target_dev);
                prev_task = task;
            }
            iris_task_dmem_flush_out(prev_task, c_tile.IRISMem());
        }
    }
    iris_graph_submit(graph);
}
```

Overhead of Tiling framework



(a) Varying X-axis Tile Count: M x M (# DGEMM Tasks)

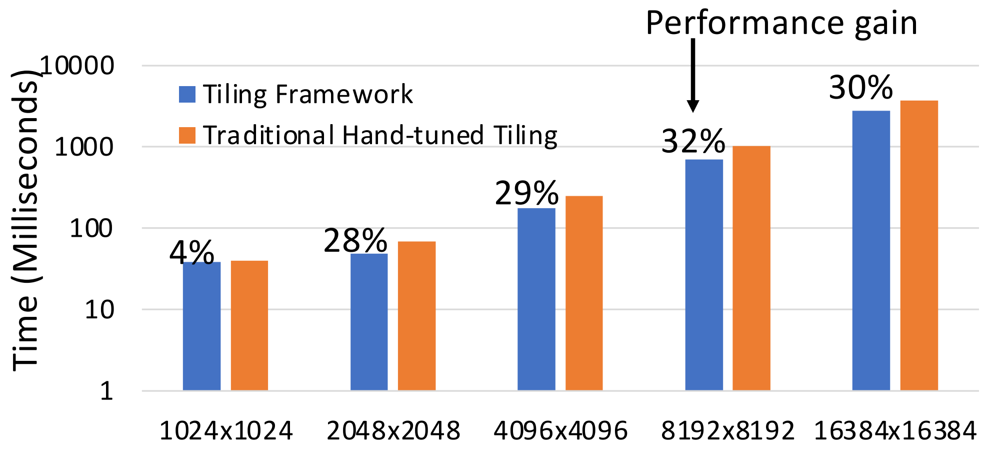


(b) Varying X-axis Tile Count: M x M (# LU Factorization Tasks)

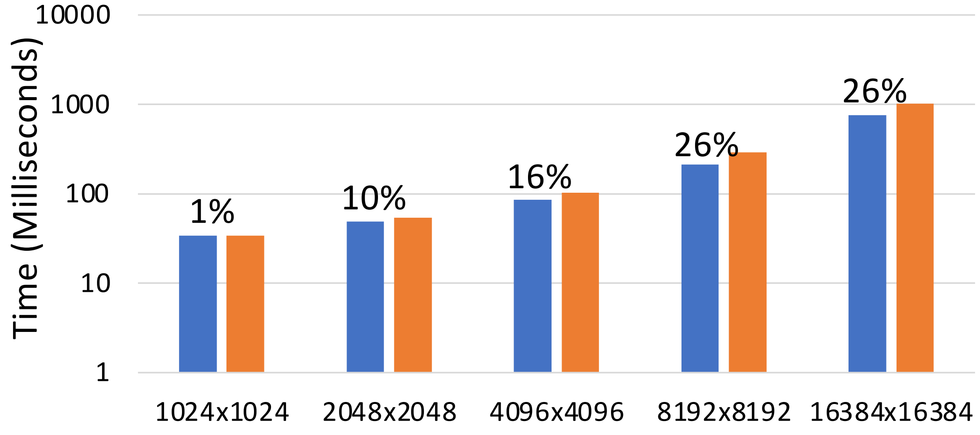
- Overhead: Time taken to create heterogeneous tasks and task graph
- Compared with traditional hand-tuned tiling approach
- varying x-axis (tile count) leads to a varying number of tasks in the task graph
- Average overhead when compared with traditional approach is ~90 µs
- Our tiling framework overhead for an increasing number of tasks is still negligible

OAK RIDGE
National Laboratory

Performance Enhancement with Native Tile Data Transfer APIs



(a) DGEMM Matrix Size: N x N



(b) LU Factorization Matrix Size: N x N

- Tiling framework enables native 2D/3D tile data transfer APIs (i.e., cuMemcpy2D) to transfer the data from a host tile to device memory through the IRIS run-time
- Varied the matrix size, as shown on the x-axis, and measured the execution time of a tiled algorithm's task graph
- Traditional, hand-tuned approach introduces a memory copy operation for flattening the tile to a continuous host memory location
- ~20% performance uplift compared with the traditional approach

OAK RIDGE National Laboratory

Conclusion and Future Work

- Proposed a novel framework for tiled algorithms for heterogenous computing
- Tiling framework
  - Creates tiles and binds tile to IRIS memory object
  - Handles the heterogeneous memory objects
  - Enables to create heterogeneous tasks for heterogeneous computing
- Benefits
  - Simplifies writing the tiled algorithms
  - Performance efficient when compared against the traditional, handwritten tiling
  - Exploit the architecture-native tile data transfer APIs
  - Gains are ~20% when compared to traditional approaches
- Future work
  - Domain specific language to further simplify the tiling algorithms
- MatRIS: https://code.ornl.gov/brisbane/matris (Need XCAMS id: https://xcams.ornl.gov/)
- IRIS: https://code.ornl.gov/brisbane/iris (Need XCAMS id)
- For any help, contact: miniskarnr@ornl.gov ;{monilm ; velerolarap ; veter}@ornl.gov

OAK RIDGE National Laboratory