OAK RIDGE National Laboratory

IRIS-BLAS: Towards a Performance Portable and Heterogeneous BLAS Library

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

Computer Science and Mathematics Division (CSMD)

21 Dec 2022

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

ORNL is managed by UT-Battelle
for the US Department of Energy

miniskarnr@ornl.gov

Outline

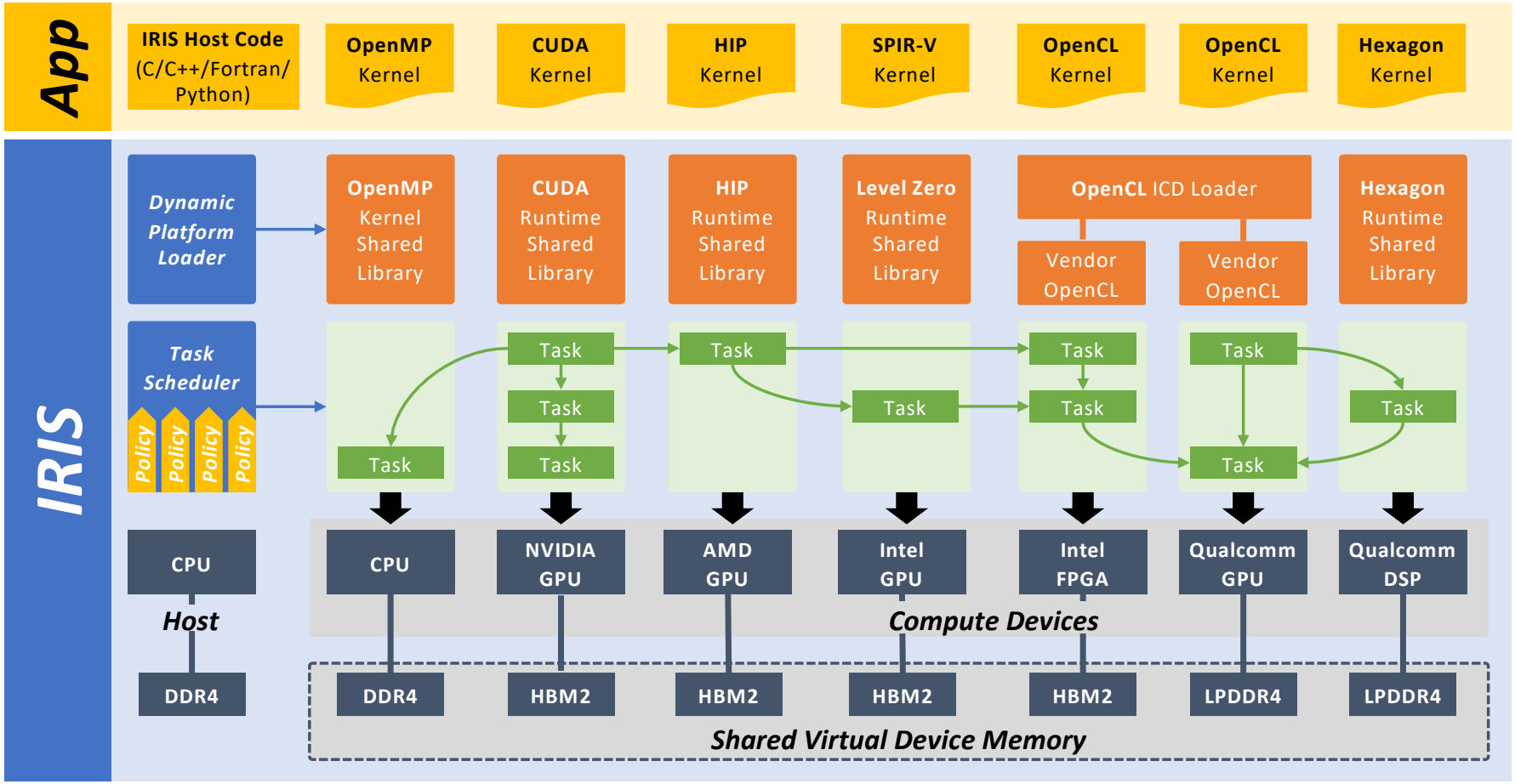
- Motivation for IRIS-BLAS: Why Heterogenous BLAS Library?
- Background: IRIS Runtime Framework
- IRIS-BLAS Software Architecture
- Experimental Results
- Future direction

Motivation for IRIS-BLAS

- Computer architectures are becoming more specialized and heterogeneous
- Future systems are truly heterogenous with GPUs, FPGAs, CPUs, ASICs
  - Thanks to Chiplet based designs
- Heterogenous BLAS Library
  - Not only for performance and portability
  - Driven by run-time system with intelligent resource mapping and schedule
  - Load distribution (Tiled algorithms)
  - Scalable and easily extensible for future heterogenous architectures and languages
- State of the art:
  - BLIS, BLASX, MAGMA: Heterogenous with static mapping / schedduling
  - StarPU: Limited support of compute units(Lacks AMD HIP, Xilinx, etc. support)

Background: IRIS Runtime Framework

https://github.com/ORNL/iris

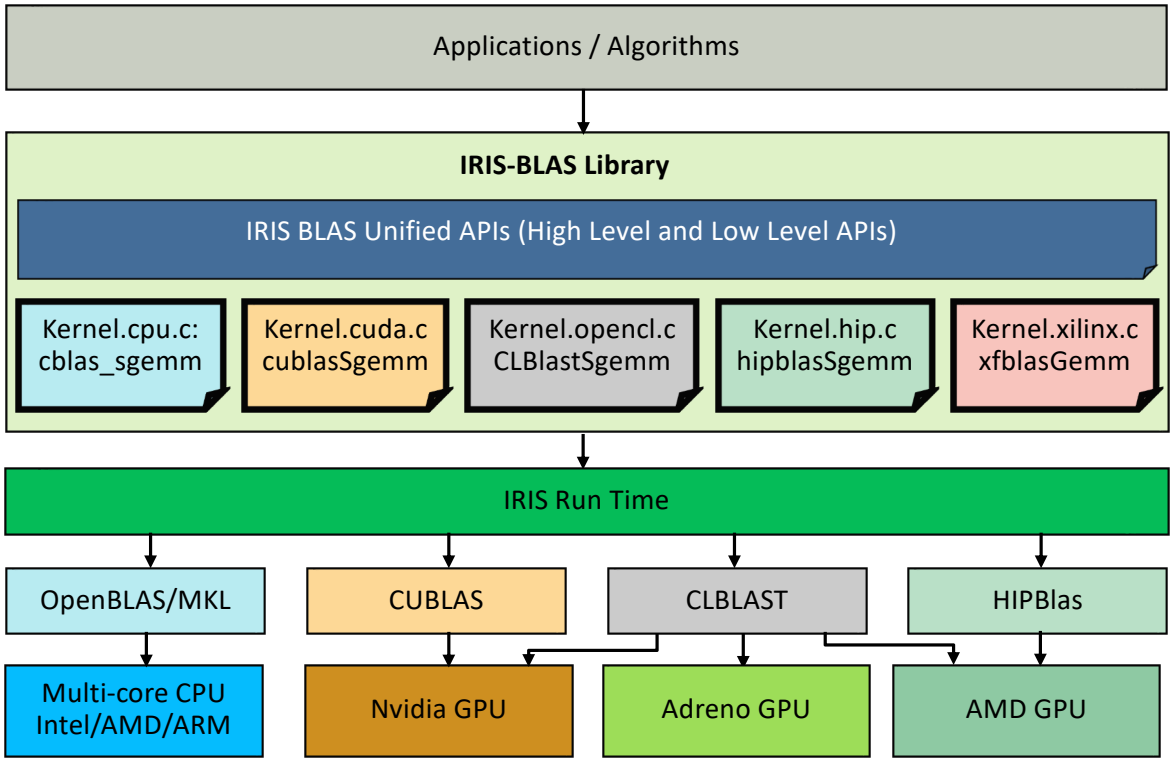


Kim, Jungwon, Seyong Lee, Beau Johnston, and Jeffrey S. Vetter. "IRIS: A portable runtime system exploiting multiple heterogeneous programming systems." In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1-8. IEEE, 2021.

Comparison of IRIS with StarPU

Feature	IRIS (C++ library)	StarPU (C library)
Architectures	**CPU, GPUs (Nvidia & AMD), FPGAs (Intel & Xilinx), Qualcomm SoCs**	CPU, GPUs (Nvidia & AMD), Xilinx FPGAs
Programming Models	**OpenMP, CUDA, HIP, OpenCL, IntelCL, XilinxCL, HexagonC**	OpenMP, CUDA, OpenCL, HIP (Recent support)
BLAS Libraries	**OpenBLAS, Intel MKL, ATLAS, cuBLAS, hipBLAS, rocBLAS, CLBLAST**	OpenBLAS, Intel MKL, cuBLAS, ATLAS
CUDA Streams	Not supported	**Supported**
New architecture support	**Application code intact; Either add a new kernel or use existing OpenCL/OpenACC/etc. kernels**	Needs to extend the application codelet along with kernel programming
Application interfaces	C/C++/Fortran/Python	C/C++/Fortran (Python & Java in recent)

IRIS-BLAS Heterogenous BLAS Library

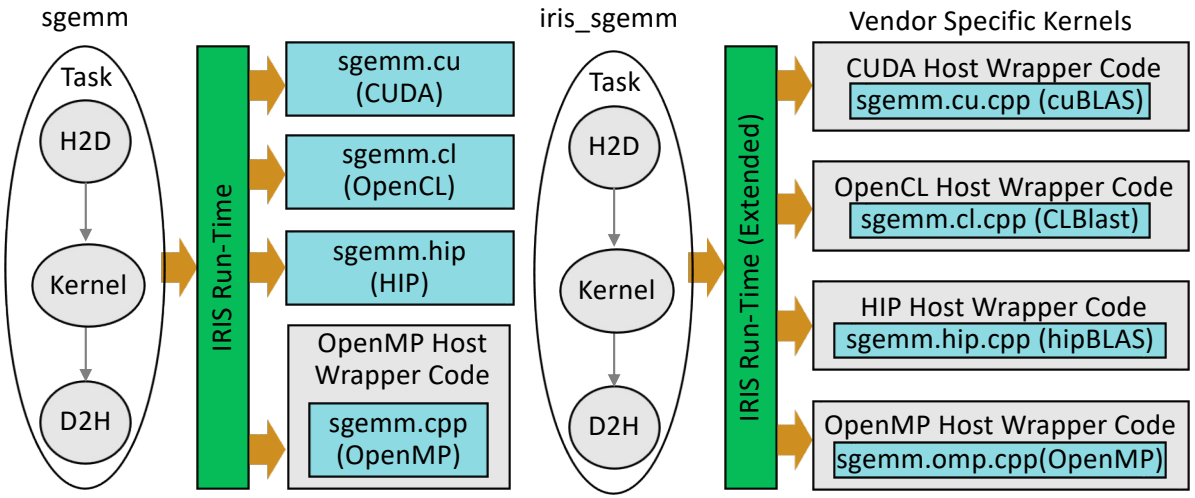


IRIS BLAS APIs

Example Usage in Application

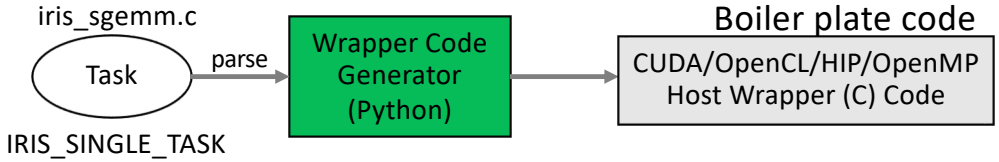
```
target_dev = TARGET == 0 ? brisbane_cpu :
             TARGET == 1 ? brisbane_gpu :
             brisbane_fpga;

init_irisblas(argc, argv);
set_irisblas_target(target_dev);

int nbytes;
float *A, *B, *C;
nbytes = SIZE * SIZE * sizeof(float);
A = (float*) malloc(nbytes);
B = (float*) malloc(nbytes);
C = (float*) malloc(nbytes);

for (int i = 0; i < SIZE * SIZE; i++) {
  A[i] = (float)i+1;
  B[i] = (float)((i+1) * 10);
  C[i] = (float)0;
}

iris_sgemm(A, B, C, SIZE);

int print_size = (SIZE > 8) ? 8: SIZE;
for(int i=0; i<print_size; i++) {
    for (int j = 0; j < print_size; j++) {
        printf("%10.1lf ", C[i*SIZE+j]);
    }
    printf("\n");
}

free(A); free(B); free(C);

finalize_irisblas();
```

How? IRIS Extensions for Vendor Specific Libraries



a) IRIS Task with native CUDA / OpenCL / HIP / OpenMP Run-times

b) IRIS Extension to support vendor specific kernels.



c) IRIS Extension to support vendor specific kernels.

- Vendor specific libraries are heavily optimized for specific compute units
- Helps in fast development of Heterogenous kernels
- Vendor specific kernels require host wrapper code
- Automated IRIS build to generate the wrapper code based on the IRIS task specification signature

- Other Extensions
  - Multi-vendor Multi-GPU support (Task specific kernel parameters memory instead of global memory)
  - Enabled Device to Device (Peer access) data transfer for NVidia CUDA and AMD HIP devices

IRIS-BLAS Software APIs for SGEMM

Two level SGEMM APIs (Top: High Level, Bottom: Low Level)

```
int iris_core_sgemm( int target_dev, IRISBlasType major,
             IRISBlasType a_trans, IRISBlasType b_trans,
             int M, int N, int K, float alpha,
             float *A, int lda,
             float *B, int ldb, float beta,
             float *C, int ldc ) {
  IRIS_SINGLE_TASK( task0, "iris_sgemm_kernel",
    target_dev, 1,
    NULL_OFFSET, GWS(M), NULL_LWS,
    PARAM(IRISBlasType, major),
    PARAM(IRISBlasType, a_trans), ...
    IN_TASK(A, float*, float, A, sizeof(float)*M*N),
    PARAM(int, lda), ...
    IN_OUT_TASK(C, float*, float, C, sizeof(float)*M*K),
    PARAM(int, ldc) );
  return IRIS_SUCCESS;
}
int iris_task_sgemm( iris_task task0, IRISBlasType major,
             IRISBlasType a_trans, IRISBlasType b_trans,
             int M, int N, int K, float alpha,
             iris_mem IRIS_VAR(A), int lda,
             iris_mem IRIS_VAR(B), int ldb, float beta,
             iris_mem IRIS_VAR(C), int ldc ) {
  IRIS_TASK_NO_DT( task0, "iris_sgemm_kernel", 1,
    NULL_OFFSET, GWS(M), NULL_LWS,
    PARAM(IRISBlasType, major),
    PARAM(IRISBlasType, a_trans), ...
    IN_TASK(A, float*, float, A, sizeof(float)*M*N),
    PARAM(int, lda), ...
    IN_OUT_TASK(C, float*, float, C, sizeof(float)*M*K),
    PARAM(int, ldc) );
    return IRIS_SUCCESS;
}
```

cuBLAS SGEMM core kernel

```
int iris_sgemm_kernel( int devno,
  IRISBlasType major,
  IRISBlasType a_trans, IRISBlasType b_trans,
  int32_t M, int32_t N, int32_t K,
  float alpha, CUdeviceptr A, int32_t lda,
               CUdeviceptr B, int32_t ldb,
  float beta, CUdeviceptr C, int32_t ldc ) {
    cublasStatus_t status =
    cublasSgemm( GetCUBlasHandle(devno),
                 GetCUBlasMap(b_trans),
                 ↪  GetCUBlasMap(a_trans),
                 N, M, K,
                 &alpha, B, ldb,
                         A, lda,
                  &beta, C, ldc );
  if (status != cudaSuccess) return IRIS_ERROR;
  return IRIS_SUCCESS;
}
```

OpenBLAS/MKL SGEMM core kernel

```
int iris_sgemm_kernel( int devno,
  IRISBlasType major,
  IRISBlasType a_trans, IRISBlasType b_trans,
  int32_t M, int32_t N, int32_t K,
  float alpha, float *A, int32_t lda,
               float *B, int32_t ldb,
  float beta,  float *C, int32_t ldc ) {
  cblas_sgemm(GetOpenBlasMap(major),
        GetOpenBlasMap(a_trans), GetOpenBlasMap(b_trans),
        M, N, K, alpha, A, lda, B, ldb, beta, C, ldc);
  return IRIS_SUCCESS;
}
```

- IRIS_SINGLE_TASK for high-level API, takes host addresses for A, B and C parameters
- IRIS_TASK_NO_DT for low-level API takes IRIS Memory objects for A, B, and C parameters for data transfer optimizations

Application Interface

C/C++ application with IRIS BLAS Sgemm call

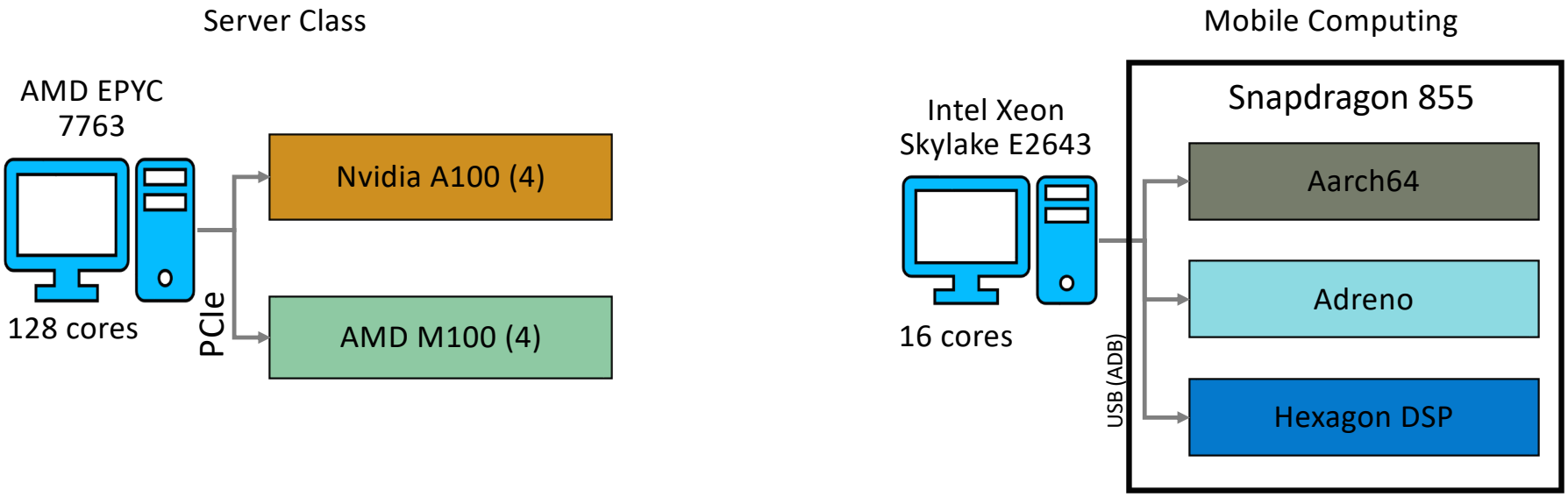
```c
#include "irisblas.h"
int call_iris_blas_sgemm(int N) {
  float *A = (float*) malloc(N*N*N*sizeof(float));
  float *B = (float*) malloc(N*N*N*sizeof(float));
  float *C = (float*) malloc(N*N*N*sizeof(float));
  init_matrix(A, N); init_matrix(B, N);
  iris_core_sgemm(iris_any, IRISBLAS_ROW_MAJOR,
        IRISBLAS_NO_TRANS, IRISBLAS_NO_TRANS,
        N, N, N, 1.0, A, N, B, N, 1.0, C, N);
  print_matrix(A, N);
  print_matrix(B, N);
  print_matrix(C, N);
  free(A); free(B); free(C);
  return IRIS_SUCCESS;
}
```

Python application with IRIS BLAS Sgemm call

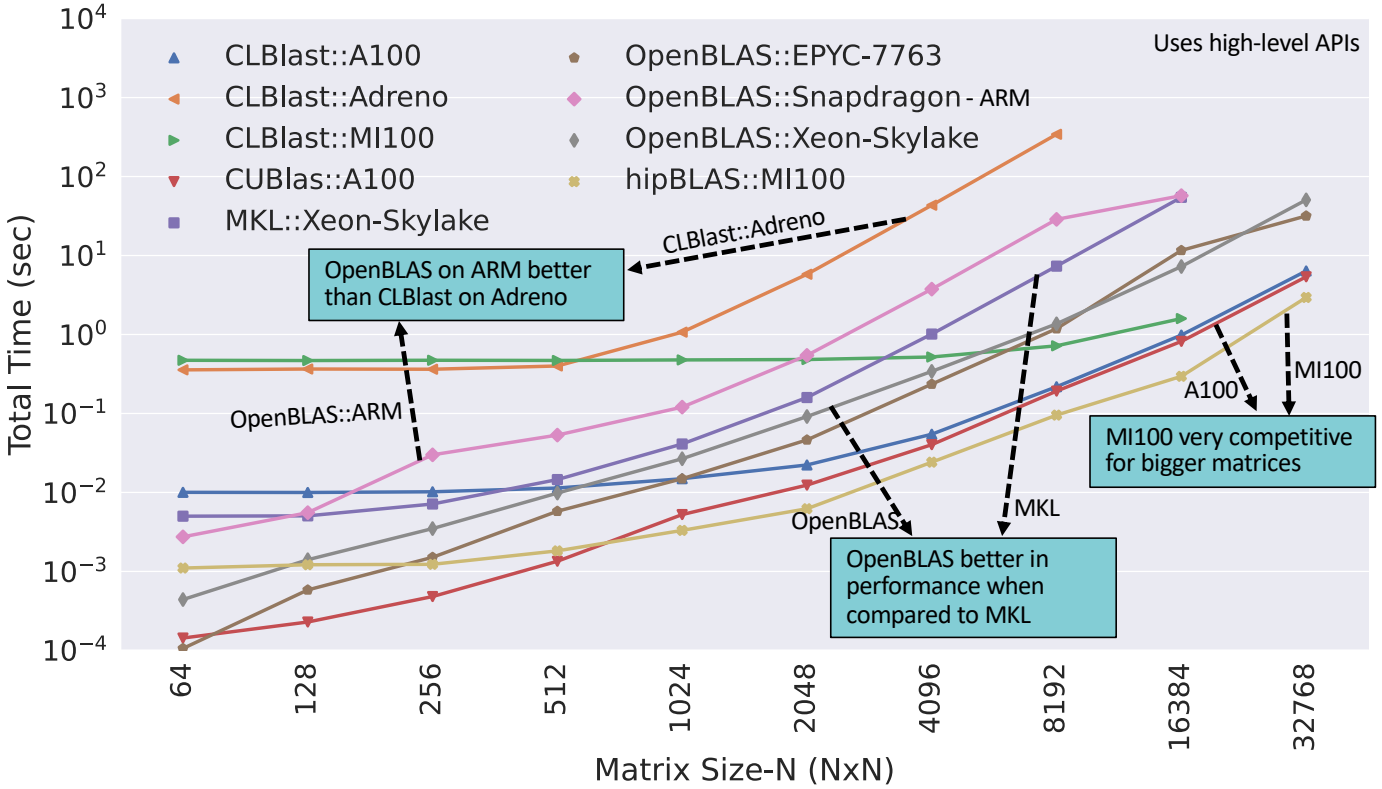
```python
import irisblas
def call_iris_blas_sgemm():
    x = np.array([[1.0, 2.0], [3.0, 4.0]], np.float)
    y = np.array([[1.0, 2.0], [3.0, 4.0]], np.float)
    z = np.array([[0.0, 0.0], [0.0, 0.0]], np.float)
    irisblas = IRISBLAS()
    irisblas.call(irisblas.iris_core_sgemm,
            iris.iris_gpu,
            irisblas.IRIS_BLAS_ROW_MAJOR,
            irisblas.IRIS_BLAS_NO_TRANS,
            irisblas.IRIS_BLAS_NO_TRANS,
            x[0].size, y[0].size, y[1].size,
            np.float(1.0), x, x[0].size,
                           y, y[0].size,
            np.float(1.0), z, z[0].size)
    print('x=',x)
    print('y=',y)
    print('z=',z)
    irisblas.finalize()
```

- C/C++ APIs
  - IRIS-BLAS High-Level APIs same as OpenBLAS APIs except name change of function and constants
  - IRIS-BLAS Low-Level APIs require programmer to create IRIS Memory objects for A, B, C matrices and pass them as parameters
- Python APIs for Heterogenous IRIS-BLAS
  - Takes Numpy array objects as parameters
  - Supports both high-level and low-level APIs

Experimental Setup



IRIS-BLAS SGEMM Performance on CPUs and GPUs

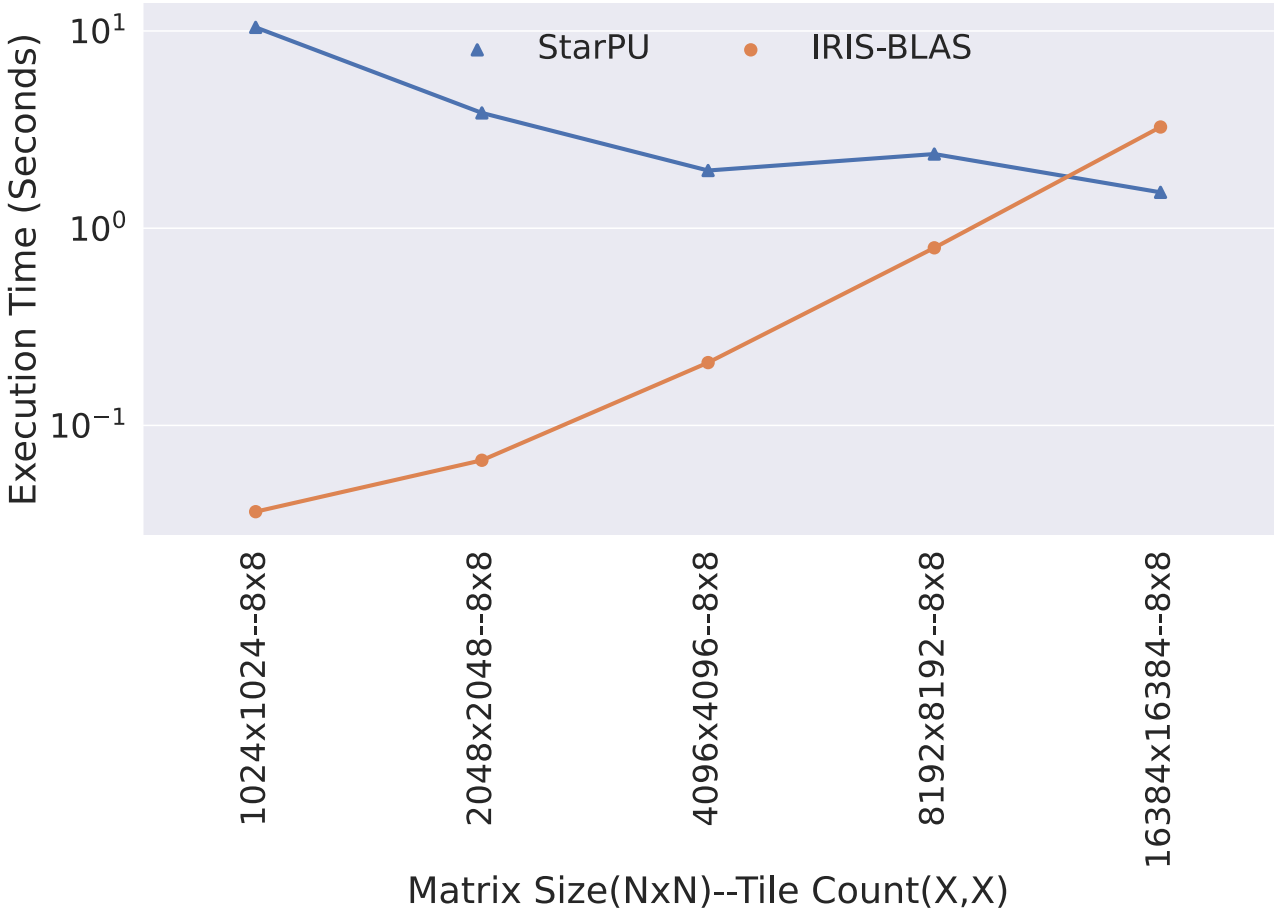


GPUs:
- Nvidia A100
- AMD MI100
- Snapdragon 855 Adreno

CPUs:
- AMD EPYC-7763
- Intel Xeon Skylake
- Snapdragon 855 ARM

- Offloads vendor or open-source optimized BLAS kernels to the target hardware at run-time
- Opens door for contemporary and upcoming heterogenous systems
- Objective of IRIS-BLAS: Portability, Performance, Heterogenous targets, Map to target at run-time

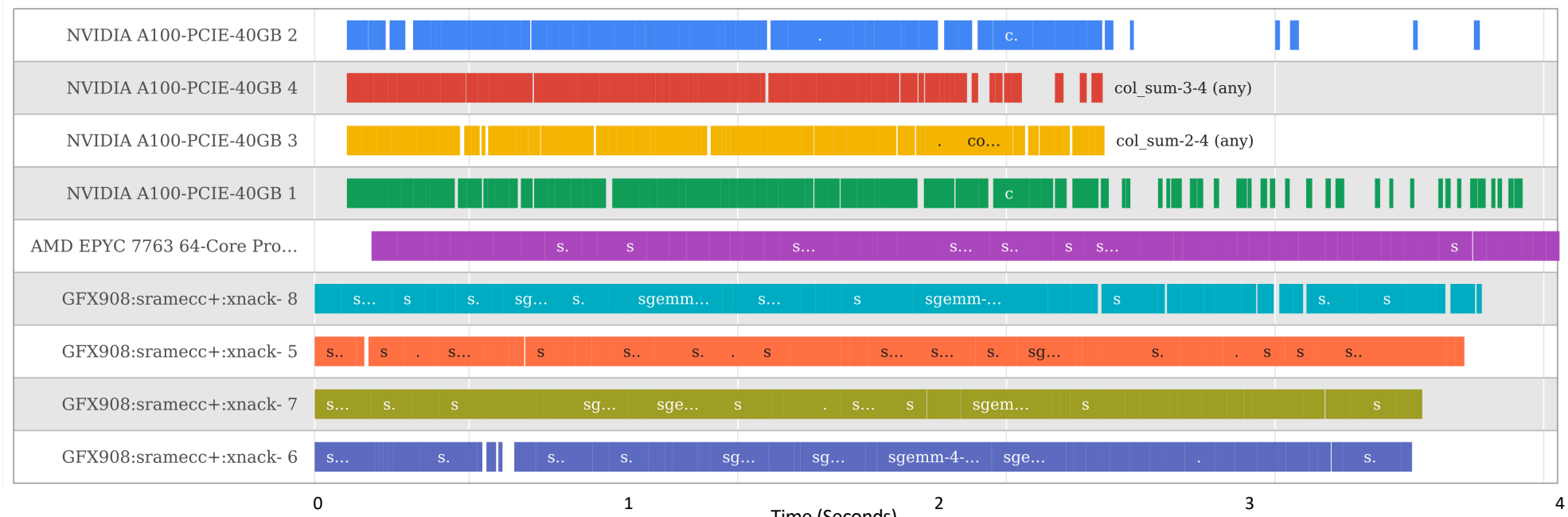
Comparison of IRIS-BLAS with StarPU-BLAS



- Usecase of IRIS-BLAS in Tiled SGEMM and comparison with StarPU
- System: 4 NVidia-A100
- Fixed Tiles Count: 8x8
- StarPU is important for big matrices
- Inferior results for bigger matrices
  - IRIS framework lacks CUDA streaming, rudimentary memory management
  - Address these issues in IRIS

Tiled SGEMM IRIS-BLAS on Extreme Heterogenous Platform

4x Nvidia GPUs, 4x AMD GPUs and 1x AMX EPYC 64-core



Extreme heterogenous task scheduling Gantt chart Test case: 16384 x 16384 SGEMM with 8x8 count of tiles

- First time extreme heterogenous systems usage with portability and run-time mapping
- StarPU doesn't have support for AMD GPUs using HIP support
- Uses IRIS-ANY (First come first serve resource allocation) dynamic task scheduler
- Most of the computational units are being used most of the time

Current Status of IRIS-BLAS

- Supported IRIS-BLAS Library functions
  - GEMM (SGEMM, DGEMM)
  - GEMV (SGEMV, DGEMV)
  - AXPY (SAXPY, DAXPY)
  - TRSM (STRSM, DTRSM)
  - GEAM (SGEAM, DGEAM)
- Renamed to MatRIS
  - Core kernels: IRIS-BLAS + LAPACK + DSP
  - Algorithms: LU Factorization*, Tiled BLAS operations
- Current use cases
  - HPC LU Factorization algorithms
  - Atomic Force Microscopy Analysis
- Efforts to make it open source soon

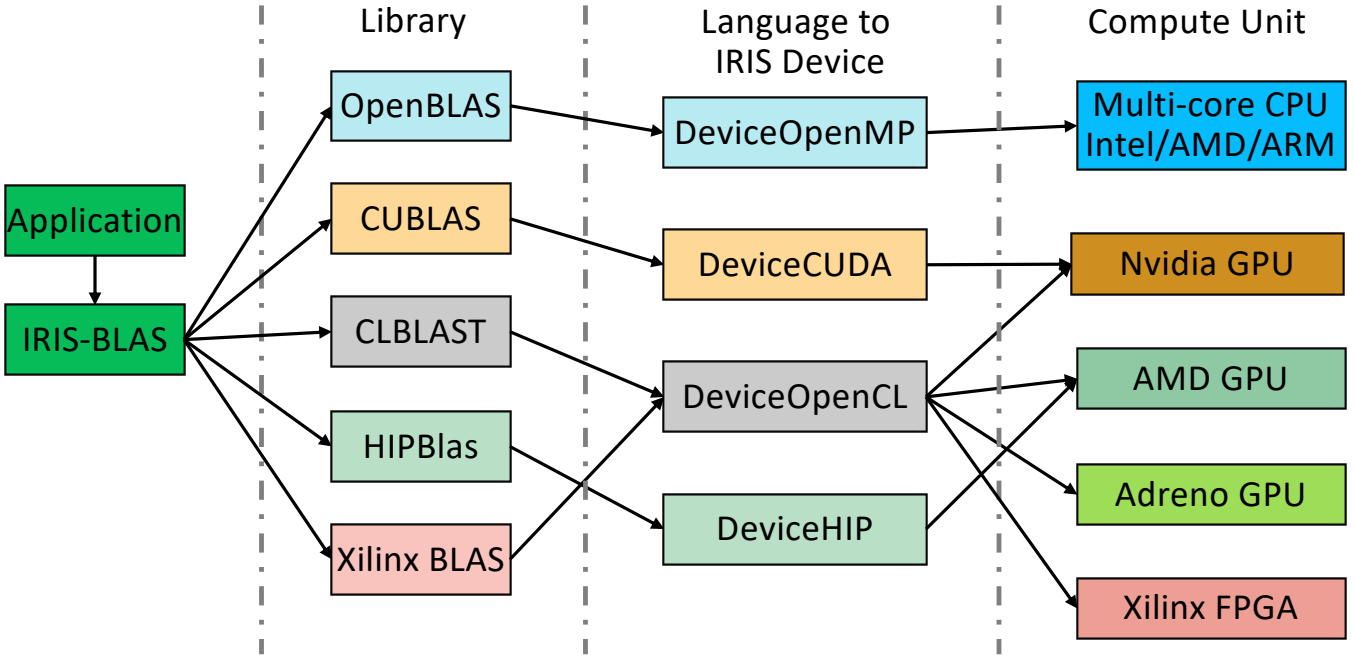
* LaRIS: Targeting Portability and Productivity for LaPACK Codes on Extreme Heterogeneous Systems using IRIS, SC 2022 RSDHA Workshop, Monil Mohammad, Narasinga Rao Miniskar, Pedro Valero Lara, Jeffrey Vetter

Conclusion and Future Work

- Presented a novel heterogenous and portable BLAS library
- Supports large number of current HPC architectures, programming models and BLAS libraries
- Experimented with extremely heterogenous system with Nvidia and AMD GPUs
- Future work:
  - Extend to support all BLAS functions
  - Architectures support
    - FPGAs (Xilinx and Intel)
    - Hexagon DSP
  - Experiment MatRIS algorithms with different IRIS task schedulers
- 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

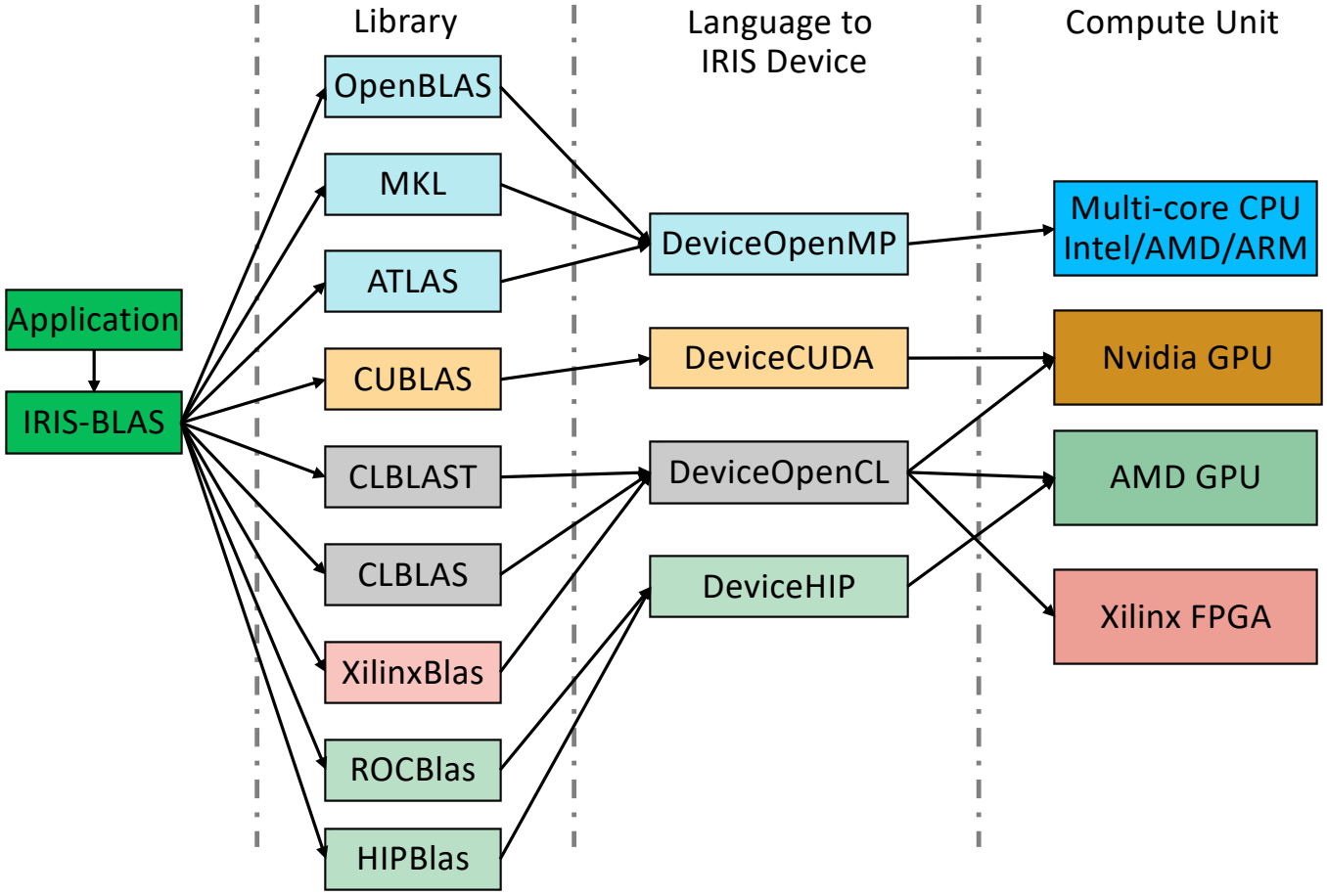
Thank you

IRIS Framework Library -> Language -> Compute Unit



- Current support
  - Run-time selection of language and compute unit
- Future Work
  - Compile time selection of library
  - Run-time selection of library

IRIS Framework Library -> Language -> Compute Unit



- Current support
  - Run-time selection of language and compute unit
- Future Work
  - Compile time selection of library
  - Run-time selection of library

Thank You