Julia with intelligent runtime for Heterogeneous Computing

Narasinga Rao Miniskar, Valero-Lara Pedro, William Godoy, Keita Teranishi, Jeffrey S. Vetter

OAK RIDGE National Laboratory

U.S. DEPARTMENT OF ENERGY

Abstract

- Julia is a high-performance, high-level programming language leverages dynamic typing and LLVM Just-in-time compiler for speed

  Challenge: How to target Julia kernels for heterogeneous devices at runtime?
- IRIS is a heterogeneous runtime for dynamic discovery and simultaneous execution, support diverse devices: CPUs, GPUs, FPGAs, DSPs

  Challenge: Complex task programming APIs for applications in C/C++/Fortran
- The integration of Julia and IRIS makes a powerful combination and provide benefits of both worlds and also provide simplified programming for heterogeneous computing
- Supports multiple programming models:

  + Core Julia written kernels written for CUDA, HIP, Threads

  + Kernel Abstraction and JACC (Julia vendor provided programming)

  + Native C++ programming (CUDA, HIP, OpenMP, Xilinx HLS)
- Results: + AXPY: comparison of different kernel writing approaches

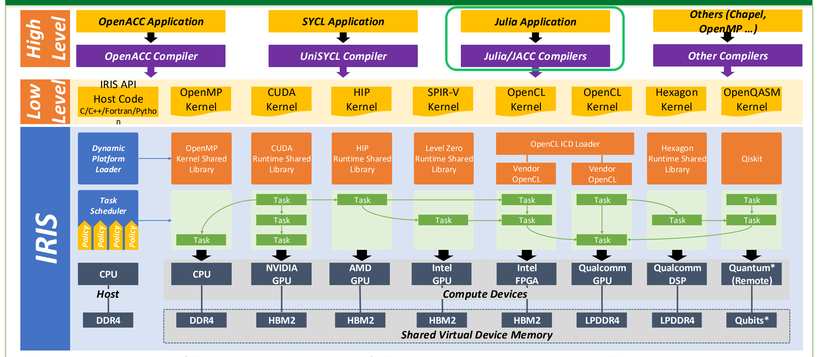
  + Tiled heterogeneous math library using vendor kernels (DGEMM example)

Background: Julia: High-level Programming Language

- High level model: Equivalent to MATLAB/Python, with macros and modules
- LLVM-based JIT: Lowered and type inferred, converted to LLVM IR and compiled to optimized machine code at runtime

- Existing task based programming package: Dagger
- Julia and Dagger lacks automatic data orchestration across multiple devices
- Missing automatic On-device multi-stream synchronization
- Lacks simultaneous use of heterogeneous devices (GPUs of Nvidia/AMD, FPGAs) support
- Lacks simplified tasking and graph creation APIs
- Need simplified tasking APIs for different variants of kernels (data parallel,

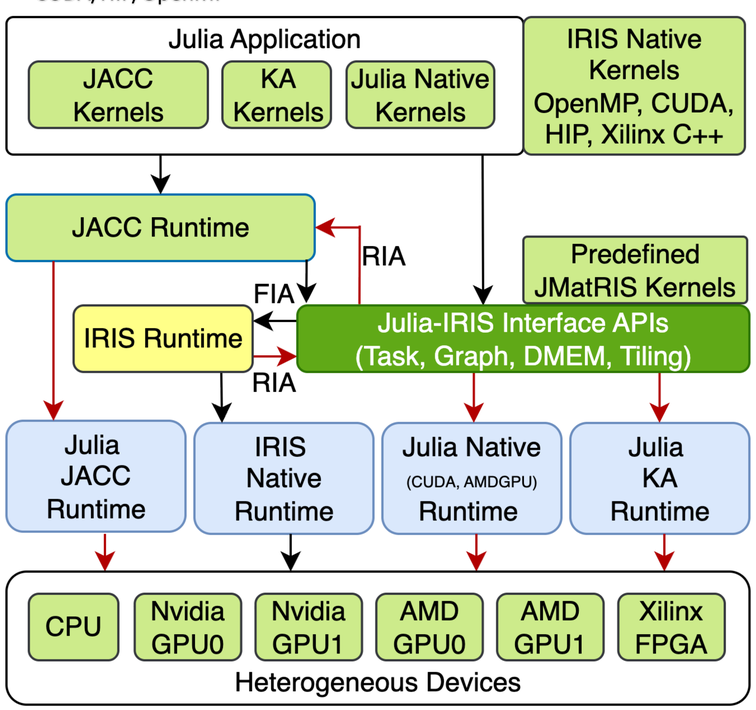
Background: IRIS: Intelligent Runtime System



- Heterogeneous (Dynamic discovery) Platform, memory and Task model
- C/C++/Fortran applications to call IRIS task based programming model APIs
- Writing C/C++/Fortran applications with IRIS APIs requires expertise
- High-level programming models with IRIS as backend: OpenACC, SYCL
- Lacks support for JIT and Datatype agnostic Kernel execution

Proposed: Julius: Julia with IRIS Integration

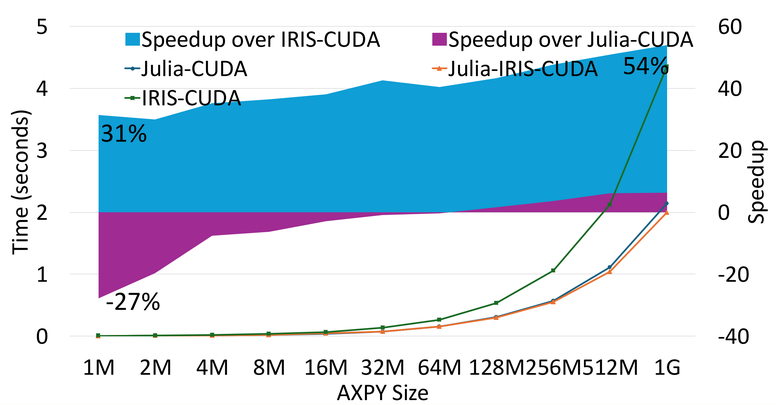
- Julia with IRIS integration have two kind of APIs
  - Forward Interface APIs (Creating/submitting Tasks/Graphs, DMEM objects)
  - Reverse Interface APIs (Call back Julia APIs to execute Julia kernels on device in Julia environment)
- Support kernels written either in
  - Native Julia code kernels written using Threads, CUDA, HIP packages
  - Kernel Abstration / JACC (Write kernel once and deploy everywhere)
  - Native IRIS kernels written in native programming models CUDA/HIP/OpenMP



Heterogeneous Kernel Example: AXPY $Z_i = A*X_i + Y_i$

- Julius can automatically create IRIS heterogeneous memory handlers and can identify the data flow dependencies of kernels and tasks
- Direct call on Julia is better in performance but Julius with scheduling overhead, etc, is on par for reasonable size input of AXPY kernel
- Kernel is type invariant, which can be called for any data type FP64, FP32, INT32, etc.,
- Compared to IRIS native calls, Julius with Julia/JACC kernel have given high performance due to optimized runtime JIT compilation

```
# JACC AXPY kernel; Write once and deploy it on any backend
function axpy(i, Z, A, X, Y)
   @inbounds  Z[i] = A*X[i] + Y[i]
end
...
Julius.init()
X, Y, Z = ones(8), ones(8), zeros(8)
Julius.@jparallel_for  gws:[size(X)], wait:true axpy(Z, A, X, Y)
Julius.finalize()
```



JMATRIS: Julius based Heterogeneous Math Library

- Tiling-2D, 1D and 3D macros provided to simplify load distribution of BLAS

```
Ct = Tiling2D.@doit data=C order=:row tile=[Mt, Nt] flat=true
Tiling2D.@doit data=A order=:row tile=[Mt, Kt] name=gA begin
  Tiling2D.@doit data=B order=:col tile=[Kt, Nt] name=gB begin
      ctile = Ct[gA.index, gB.index]
      for (atile, btile) in zip(gA.data, gB.data)
          task0 = @jtask  ...  Julius.gemm(atile, btile, ctile, ...)
          Julius.add_task(graph, task0, policy)
      end
  end
end
```

- Performance JMATRIS.DGEMM is on par with MATRIS.DGEMM
- Tiling APIs provide split of big array of tiles and can provide iterators and groups
- Enables users to write any kind of tiled algorithms
- 62 TFLOPS on 4 Nvidia A100 GPUs for 32k x 32k Matrix size DGEMM
- JMATRIS uses vendor library BLAS and LAPACK functions for device level GEMM calls

- IRIS: https://github.com/ORNL/iris

- "IRIS: A portable runtime system exploiting multiple heterogeneous programming systems." Kim, Jungwon, Seyong Lee, Beau Johnston, and Jeffrey S. Vetter. In 2021 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1-8. IEEE, 2021.
- "Matris: Multi-level math library abstraction for heterogeneity and performance portability using iris runtime." Monil, Mohammad Alaul Haque, Narasinga Rao Miniskar, Keita Teranishi, Jeffrey S. Vetter, and Pedro Valero-Lara. In Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, 2023