OAK RIDGE National Laboratory

Automatic IRIS Code Generation in OpenARC

Seyong Lee

IRIS Miniworkshop
Oak Ridge National Laboratory
Janunary 4, 2022

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

U.S. DEPARTMENT OF ENERGY

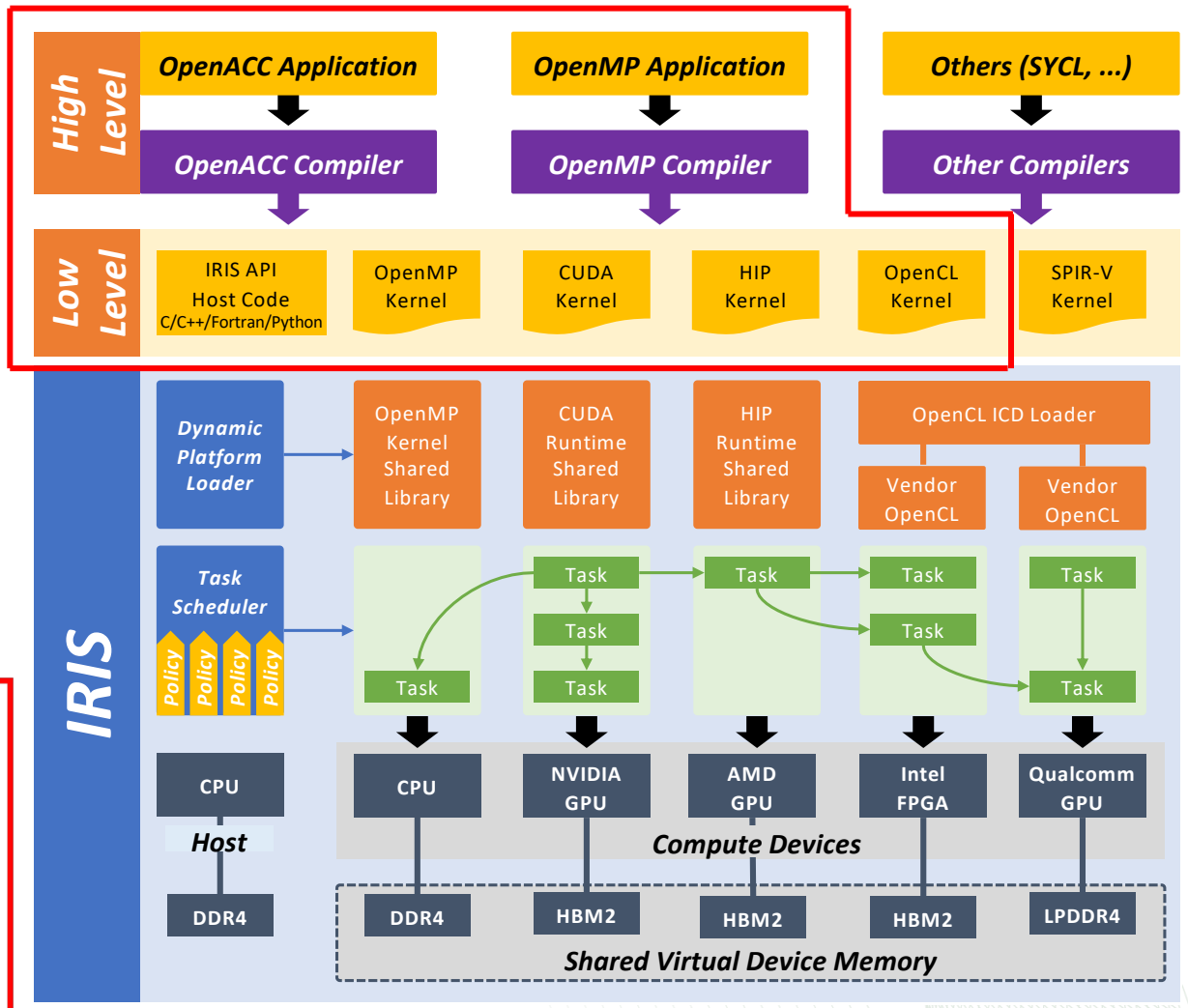
ORNL is managed by UT-Battelle
for the US Department of Energy

https://csmd.ornl.gov/project/openarc-open-accelerator-research-compiler/

The IRIS Architecture

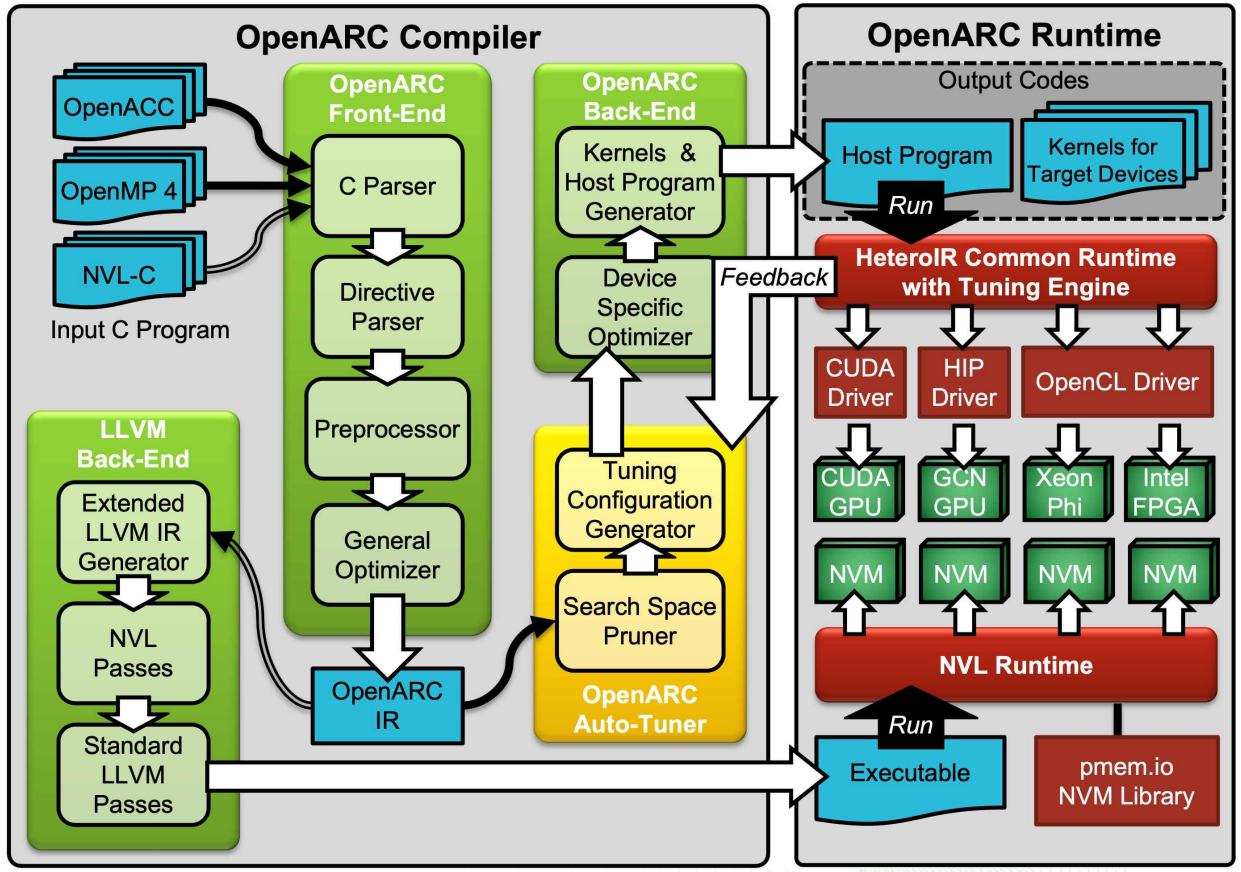
- Platform Model
  - A single-node system equipped with host CPUs and multiple compute devices (GPUs, FPGAs, Xeon Phis, and multicore CPUs)
- Memory Model
  - Host memory + shared virtual device memory
  - All compute devices share the device memory
- Execution Model
  - DAG-style task parallel execution across all available compute devices
- Programming Model
  - Low-level C/C++/Fortran/Python IRIS API host code + OpenCL/SPIR-V/CUDA/HIP/OpenMP kernels
  - High-level OpenACC, OpenMP4, SYCL* (* planned)

Automatic high-level-to-low-level translation by OpenARC



OpenARC: Open Accelerator Research Compiler

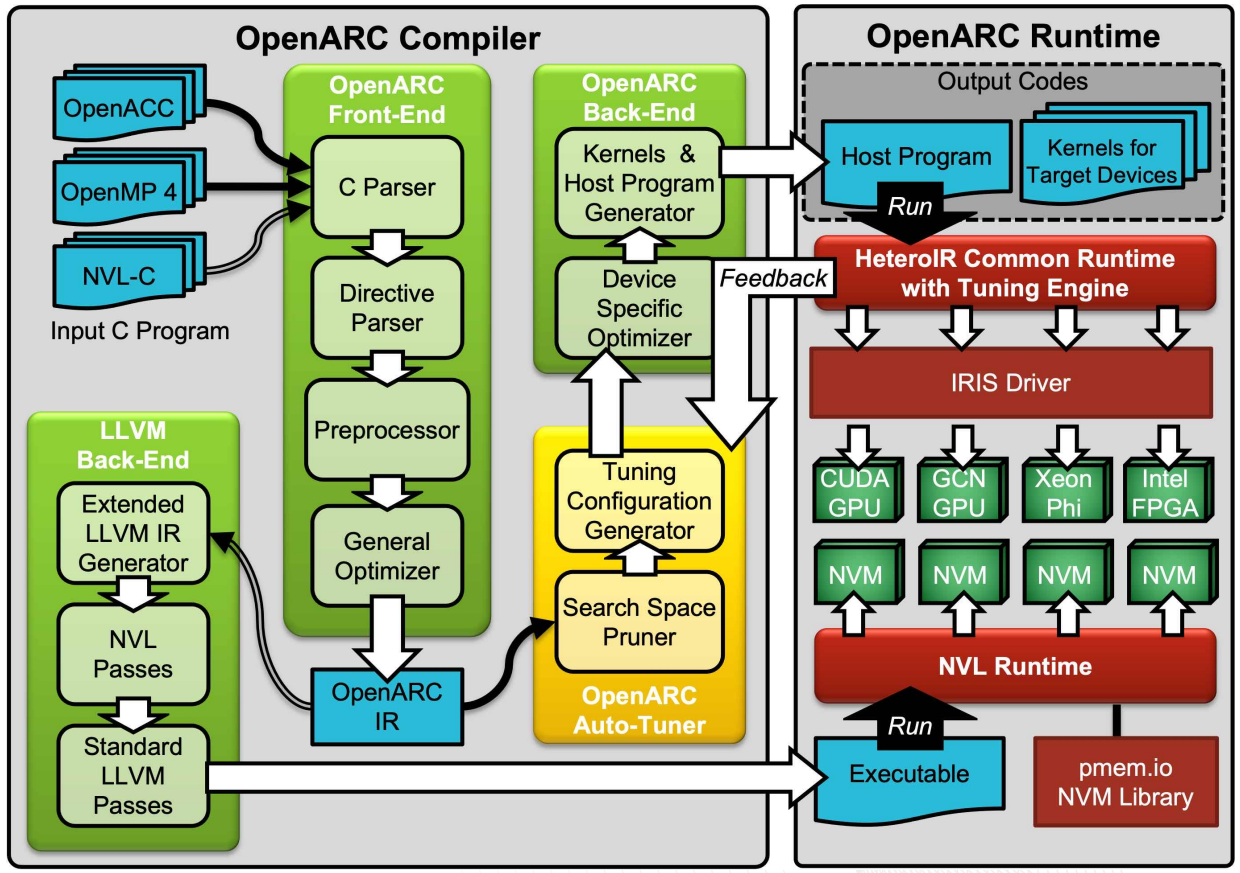
- Open-sourced, high-level intermediate representation (HLIR)-based, extensible compiler framework.
  - Perform source-to-source translation from OpenACC/OpenMP4+ C to CUDA/OpenCL/HIP.
    - Also support source-to-source translation and optimization between OpenMP and OpenACC.
  - Supported target architectures: NVIDIA/AMD GPUs, Intel Xeon Phis, Intel FPGAs, and Multicore CPUs (via LLVM)
- Provide a common runtime abstraction (HeteroIR) for various back-ends.
  - Target-specific drivers (e.g., CUDA driver) implement HeteroIR APIs using low-level backend programming models (e.g., CUDA).
- Can be used as a research framework for various study on directive-based accelerator computing.
  - OpenARC's HLIR is easy to understand, access, and transform the input program.
  - Equipped with various advanced analysis/transformation passes.



S. Lee and J.S. Vetter, "OpenARC: Open Accelerator Research Compiler for Directive-Based, Efficient Heterogeneous Computing," in *ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC). Vancouver: ACM, 2014*

Automatic IRIS Code Generation in OpenARC

- OpenARC takes input OpenACC/OpenMP4 programs and generates output IRIS programs consisting of the following codes:
  - HeteroIR-based host program
    - Indirectly calls IRIS APIs using the IRIS Driver.
  - Target-specific kernels:
    - CUDA kernels for NVIDIA GPUs
    - OpenCL kernels for general OpenCL devices
    - OpenCL kernels for Intel FPGAs
    - HIP kernels for AMD GPUs
    - OpenMP kernels for general CPUs (partially implemented)
    - DSP kernels for Qualcomm Hexagon DSPs (planned)



S. Lee and J.S. Vetter, "OpenARC: Open Accelerator Research Compiler for Directive-Based, Efficient Heterogeneous Computing," in *ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC). Vancouver: ACM, 2014*

HeteroIR: OpenARC Runtime API

- HeteroIR: High-Level, Architecture-Independent Intermediate Representation
  - Used as an intermediate language to map high-level programming models (e.g., OpenACC) to diverse heterogeneous devices.
- Primary Constructs in HeteroIR (Partial List)

HI_init (devnum)	Initialize a selected device.
HI_reset ()	Deinitialize the device.
HI_malloc (devptr, size, flags)	Allocate memory on the device (devptr).
HI_malloc1D(hostptr, devptr, size, asyncID, flags)	Allocate device memory (devptr) corresponding to the host memory (hostptr).
HI_memcpy (dst, src, size, trtype)	Perform synchronous data transfers.
HI_memcpy_async(dst, src, size, trtype, asyncID, waits)	Perform asynchronous data transfers.
HI_get_device_address (hptr, dptr, asyncID, tid)	Check if the host address has a valid mapping on the device. Return the device address if mapping is present.
HI_free (hostptr, asyncID)	Free the device memory corresponding to the host memory pointed by hostptr.
HI_register_kernel_arg (kernel name, parameter, …)	Attach the argument to the corresponding kernel.
HI_kernel_call (kernel name, grid size, block size, …)	Launch the kernel with the specified grid size on the specified queue.
HI_set_async (asyncID)	Map the specified async ID to a device queue if the mapping does not exist.
HI_wait (asyncID)	Wait until all actions on the specified queue are finished.
HI_wait_all ()	Wait for all queues to finish the actions queued on them.

Example Translation: VecAdd

```
22
23 #pragma acc data copyin(A[0:size], B[0:size]) copyout(C[0:size])
24     {
25 #pragma acc kernels loop independent gang worker
26         for (i = 0; i < size; i++) {
27             C[i] = A[i] + B[i];
28         }
29     }
30
```

Input OpenACC Code

Example Translation: Generated Output IRIS Codes

```
100 ...
101 HI_enter_subregion(NULL);
102 gpuBytes=(sizeof (float)*size);
103 HI_malloc1D(A, ((void * *)( & gpu__A)), gpuBytes, DEFAULT_QUEUE, HI_MEM_READ_WRITE);
104 HI_memcpy(gpu__A, A, gpuBytes, HI_MemcpyHostToDevice, 0);
105 gpuBytes=(sizeof (float)*size);
106 HI_malloc1D(B, ((void * *)( & gpu__B)), gpuBytes, DEFAULT_QUEUE, HI_MEM_READ_WRITE);
107 HI_memcpy(gpu__B, B, gpuBytes, HI_MemcpyHostToDevice, 0);
108 gpuBytes=(sizeof (float)*size);
109 HI_malloc1D(C, ((void * *)( & gpu__C)), gpuBytes, DEFAULT_QUEUE, HI_MEM_READ_WRITE);
110 HI_exit_subregion(NULL);
111 size_t dimGrid_main_kernel0[3];
112 dimGrid_main_kernel0[0]=((int)ceil((((float)size)/64.0F)));
113 dimGrid_main_kernel0[1]=1;
114 dimGrid_main_kernel0[2]=1;
115 size_t dimBlock_main_kernel0[3];
116 dimBlock_main_kernel0[0]=64;
117 dimBlock_main_kernel0[1]=1;
118 dimBlock_main_kernel0[2]=1;
119 gpuNumBlocks=((int)ceil((((float)size)/64.0F)));
120 gpuNumThreads=64;
121 totalGpuNumThreads=(((int)ceil((((float)size)/64.0F)))*64);
122 {
123 HI_enter_subregion(NULL);
124 HI_register_kernel_numargs("main_kernel0",4);
125 HI_register_kernel_arg("main_kernel0",0,sizeof(void*),( & gpu__A),1);
126 HI_register_kernel_arg("main_kernel0",1,sizeof(void*),( & gpu__B),1);
127 HI_register_kernel_arg("main_kernel0",2,sizeof(void*),( & gpu__C),1);
128 HI_register_kernel_arg("main_kernel0",3,sizeof (int),( & size),0);
129 HI_kernel_call("main_kernel0",dimGrid_main_kernel0,dimBlock_main_kernel0,DEFAULT_QUEUE,0,NULL);
130 HI_synchronize(0);
131 gpuNumBlocks=((int)ceil((((float)size)/64.0F)));
132 HI_exit_subregion(NULL);
133 }
134 HI_enter_subregion(NULL);
135 gpuBytes=(sizeof (float)*size);
136 HI_memcpy(C, gpu__C, gpuBytes, HI_MemcpyDeviceToHost, 0);
137 HI_exit_subregion(NULL);
138 HI_free(C, DEFAULT_QUEUE);
139 HI_free(B, DEFAULT_QUEUE);
140 HI_free(A, DEFAULT_QUEUE);
141 ...
```

Output Host Code

```
25
26 extern "C" __global__ void main_kernel0(float * A, float * B, float * C, int size)
27 {
28 int lwpriv__i;
29 lwpriv__i=(threadIdx.x+(blockIdx.x*blockDim.x));
30 if (lwpriv__i<size)
31 {
32 C[lwpriv__i]=(A[lwpriv__i]+B[lwpriv__i]);
33 }
34 }
35
```

CUDA Kernel

```
26
27 __kernel void  main_kernel0(__global float * A, __global float * B,
28 __global float * C, int size)
29 {
30 int lwpriv__i;
31 lwpriv__i=get_global_id(0);
32 if (lwpriv__i<size)
33 {
34 C[lwpriv__i]=(A[lwpriv__i]+B[lwpriv__i]);
35 }
36 }
37
```

OpenCL Kernel

```
26
27 __kernel void __attribute__((reqd_work_group_size(64, 1, 1))) main_kernel0(
28 __global float * A, __global float * B, __global float * C, int size)
29 {
30 int lwpriv__i;
31 lwpriv__i=get_global_id(0);
32 if (lwpriv__i<size)
33 {
34 C[lwpriv__i]=(A[lwpriv__i]+B[lwpriv__i]);
35 }
36 }
37
```

FPGA-OpenCL Kernel

```
27
28 extern "C" __global__ void main_kernel0(float * A, float * B, float * C, int size)
29 {
30 int lwpriv__i;
31 lwpriv__i=(threadIdx.x+(blockIdx.x*blockDim.x));
32 if (lwpriv__i<size)
33 {
34 C[lwpriv__i]=(A[lwpriv__i]+B[lwpriv__i]);
35 }
36 }
37
```

HIP Kernel

Output Kernel Codes

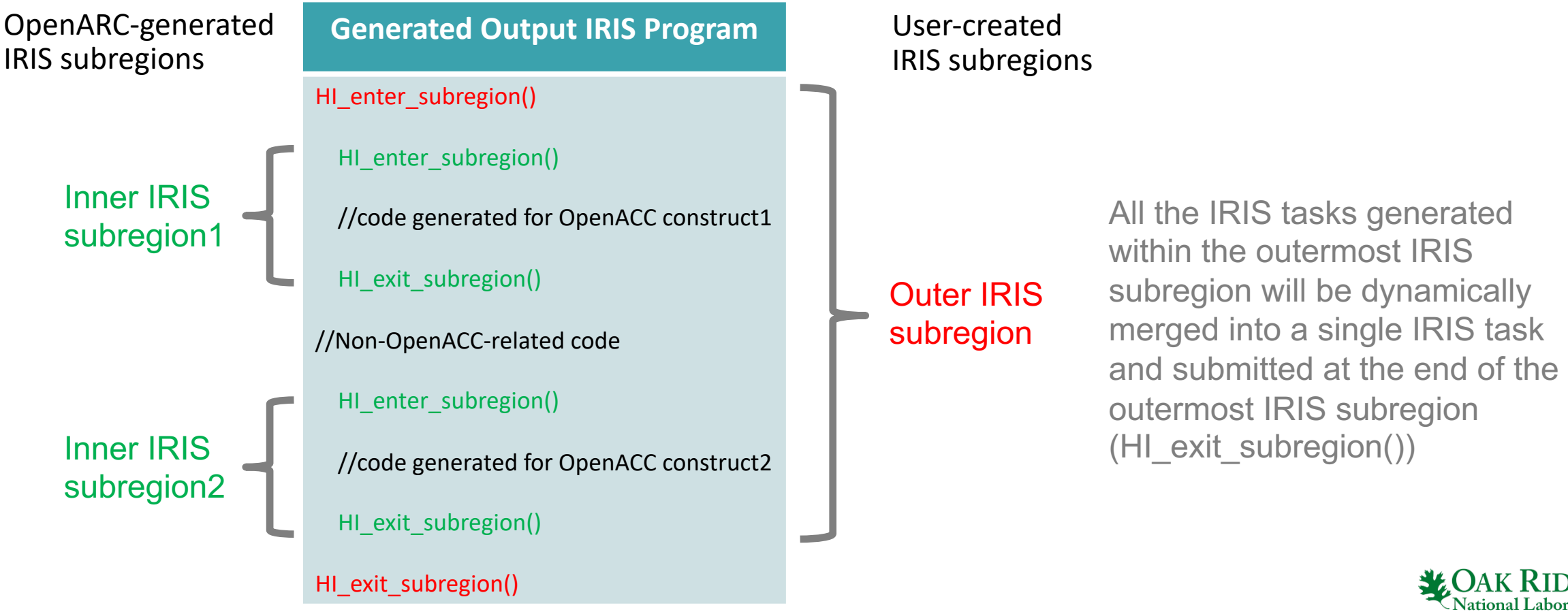
OAK RIDGE National Laboratory

Optimizations to Reduce IRIS Task Overheads

- Naïve implementation of the OpenARC-IRIS system would generate IRIS tasks per memory transfer and compute kernel, which may suffer from additional overheads caused by too many IRIS tasks.
- Developed an optimized IRIS-task-generation pass, which automatically merges IRIS tasks belonging to the same OpenACC construct.
- Provide optional runtime APIs to automatically merge multiple IRIS tasks across different OpenACC constructs if the user guarantees its safety.
  - HI_enter_subregion(…);
  - HI_exit_subregion(…);

Optimizations to Reduce IRIS Task Overheads (Cont.)

- Provide optional runtime APIs to automatically merge multiple IRIS tasks across different OpenACC constructs if the user guarantees its safety.



Task Scheduling in OpenARC-IRIS System

- Input OpenACC provides a way to set a target device type and/or device number using environment variables/directives/runtime APIs.
- If a target device is specified in the input OpenACC program, OpenARC-IRIS runtime submits the generated IRIS tasks to the target device.
- If a target device is not specified, the underlying IRIS runtime decides how to schedule the generated tasks.
  - Environment variable, **BRISBANE_ARCHS** is used to decide the order to search available device types.
    - Default search order: openmp:cuda:hip:levelzero:hexagon:opencl
    - If multiple device types are available, the first found device type is used.

Task Scheduling in OpenARC-IRIS System (Cont.)

- Environment variable, **OPENARCRT_BRISBANE_POLICY** is used to decide which IRIS policy to use when scheduling the generated tasks.
  - **none**: do not use any IRIS policy and use OpenACC device type and number when choosing a target device (*default*)
  - **brisbane_roundrobin**: submit tasks to devices in a round-robin manner.
  - **brisbane_data**: submit tasks to devices in a way to minimize memory transfers among devices
  - **brisbane_profile**: submit tasks to devices based on the profiled performance
  - **brisbane_random**: submit tasks to devices randomly
  - **brisbane_any**: submit each task to a device with minimal pending tasks
  - **brisbane_all**: submit each task to all devices but only the first available device executes the task.

How to Build OpenARC Compiler and Runtime for IRIS Backend

- Build OpenARC (Refer to **README.md** in the OpenARC repository)
  - Assume IRIS is installed and environments are correctly set (e.g., LD_LIBRARY_PATH, LIBRARY_PATH, and CPATH)
  - Set up an environment variable, **openarc** to the root directory of the OpenARC repository.

```
export openarc=[OpenARC-root-path]
```

  - Set up an environment variable, **OPENARC_ARCH** to 6 to use IRIS as a backend.

```
export OPENARC_ARCH=6
```

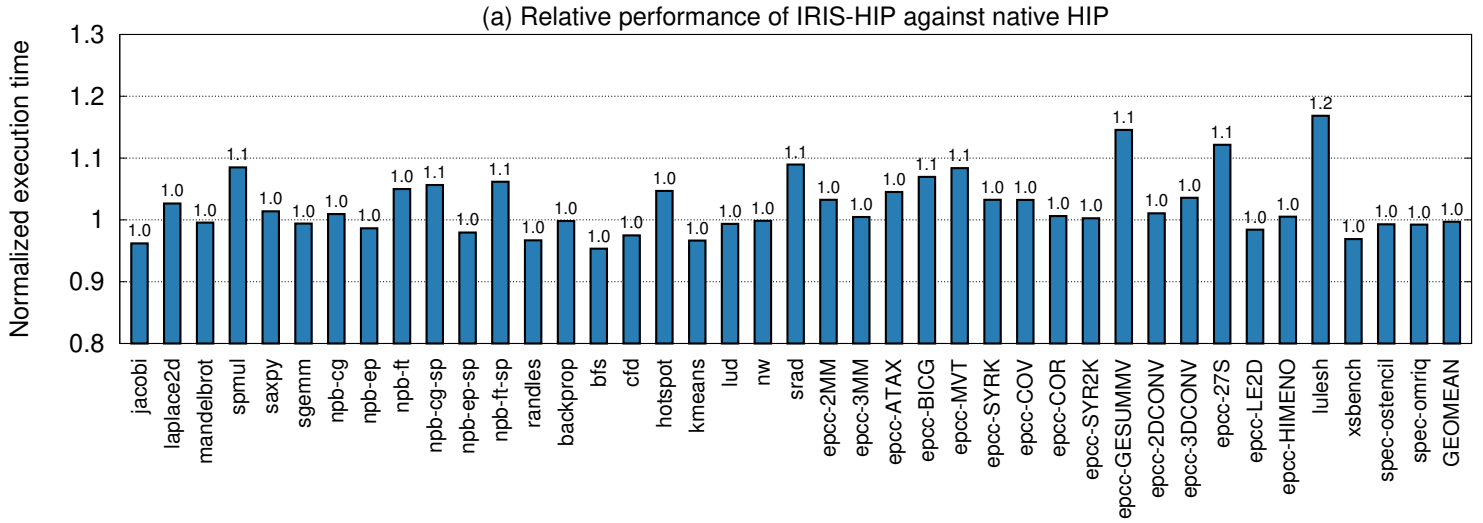
  - Run the make command to build both OpenARC compiler and runtime.

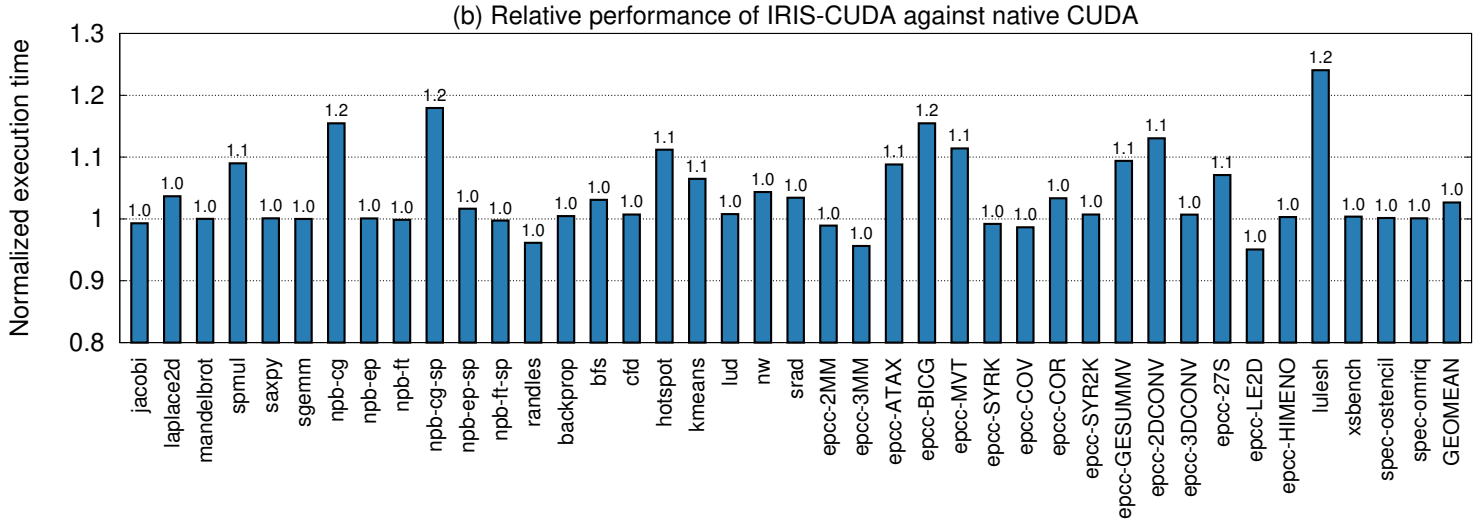
```
$ make purge #needed only when OpenARC was previously built for other targets.
$ make
$ make install TARGET_SYSTEM=[install-path] #for optional install
```

OpenARC-IRIS Example Compilation and Execution

```
& bisBuilder brisbase libresilience.a libompteiper.a Timer libeclert.a
make[1]: Leaving directory `/tmp/openarc/openarcrt'
make[1]: Entering directory `/tmp/openarc/openarcrt'
g++ -DPRINT_LOG=0 -DOMP=1  -DOPENARC_ARCH=6 -O3 -fPIC  -fopenmp -o openacc.o openacc.cpp -c
g++ -DPRINT_LOG=0 -DOMP=1  -DOPENARC_ARCH=6 -O3 -fPIC  -fopenmp -o openaccrt.o openaccrt.cpp -c
g++ -DPRINT_LOG=0 -DOMP=1  -DOPENARC_ARCH=6 -O3 -fPIC  -fopenmp -o brisbanedriver.o brisbanedriver.cpp -c
ar rs ./libopenaccrtomp_brisbane.a openacc.o openaccrt.o brisbanedriver.o
ar: creating ./libopenaccrtomp_brisbane.a
make[1]: Leaving directory `/tmp/openarc/openarcrt'
make[1]: Entering directory `/tmp/openarc/openarcrt'
rm -f *.o *~
make[1]: Leaving directory `/tmp/openarc/openarcrt'
make[1]: Entering directory `/tmp/openarc/openarcrt'
g++ -DPRINT_LOG=0 -DOMP=1  -DOPENARC_ARCH=6 -O3 -fPIC -g -D_OPENARC_PROFILE_=1  -fopenmp -o openacc.o openacc.cpp -c
g++ -DPRINT_LOG=0 -DOMP=1  -DOPENARC_ARCH=6 -O3 -fPIC -g -D_OPENARC_PROFILE_=1  -fopenmp -o openaccrt.o openaccrt.cpp
 -c
g++ -DPRINT_LOG=0 -DOMP=1  -DOPENARC_ARCH=6 -O3 -fPIC -g -D_OPENARC_PROFILE_=1  -fopenmp -o brisbanedriver.o brisbane
driver.cpp -c
ar rs ./libopenaccrtomp_brisbanepf.a openacc.o openaccrt.o brisbanedriver.o
ar: creating ./libopenaccrtomp_brisbanepf.a
make[1]: Leaving directory `/tmp/openarc/openarcrt'
make[1]: Entering directory `/tmp/openarc/openarcrt'
rm -f *.o *~
make[1]: Leaving directory `/tmp/openarc/openarcrt'
make[1]: Entering directory `/tmp/openarc/openarcrt'
g++ -DPRINT_LOG=0 -DOMP=0  -DOPENARC_ARCH=6 -O3 -fPIC  -o openacc.o openacc.cpp -c
g++ -DPRINT_LOG=0 -DOMP=0  -DOPENARC_ARCH=6 -O3 -fPIC  -o openaccrt.o openaccrt.cpp -c
g++ -DPRINT_LOG=0 -DOMP=0  -DOPENARC_ARCH=6 -O3 -fPIC  -o brisbanedriver.o brisbanedriver.cpp -c
```

Performance Comparison of OpenARC-Generated IRIS Programs against the Native GPU Versions





[Target System]
CPU: AMD EPYC 7702
GPU: AMD MI60
Backend Runtime: AMD HIP 3.8
Backend Compiler: GCC 9.1

[Target System]
CPU: IBM Power9
GPU: NVIDIA V100
Backend Runtime: CUDA 10.1
Backend Compiler: IBM XL C++ 16.1

OAK RIDGE National Laboratory

Current Status and Future Work

- Not-all HeteroIR APIs are implemented in the IRIS driver yet.
  - Need to implement some asynchronous HeteroIR APIs such as low-level asynchronous event management APIs (high-level asynchronous memory transfer/kernel launch APIs are supported now).
- Some device-specific optimizations are not yet available when targeting the IRIS backends.
  - Example: Pipeline transformation for Intel FPGAs, which is enabled only when targeting a specific device type directly without using IRIS for now.
- Plan to implement automatic OpenMP kernel and DSP kernel generation passes in the OpenARC compiler.