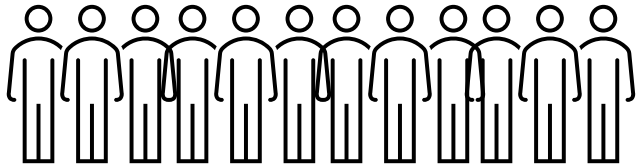
Supporting CUDA for an extended RISC-V GPU architecture

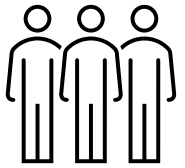
Reporter: Ruobing Han

Date: 05/30/2021

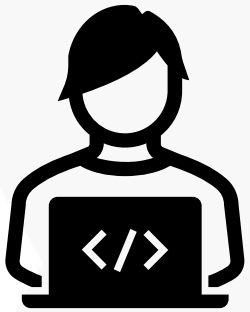
co-author: Blaise Tine, Jaewon Lee, Jaewoong Sim, Hyesoon Kim

Motivation





High-level programmers

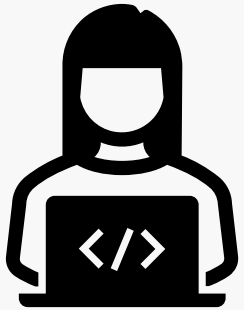


NVIDIA CUDA

SYCL™

OpenCL™

Low-level programmers

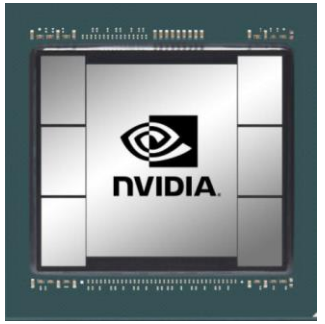


SiFive

CloudBEAR

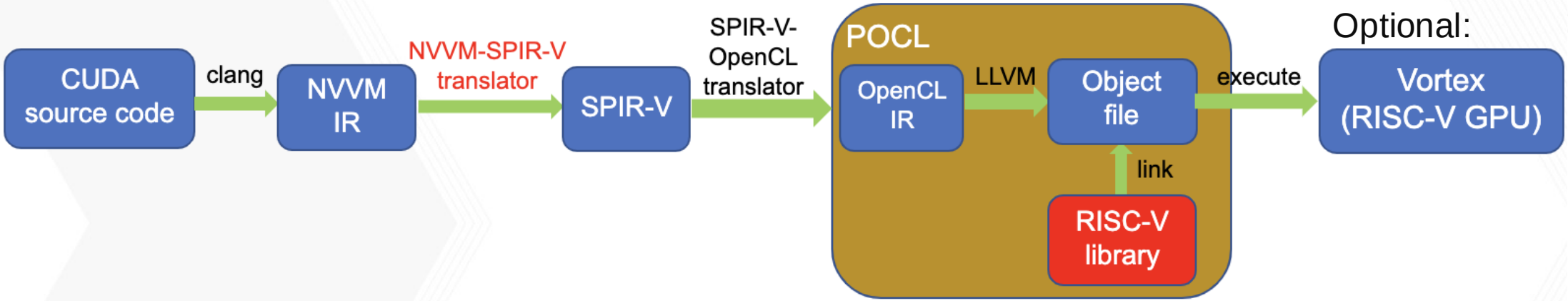
Alibaba.com

codasip



Georgia Tech
CREATING THE NEXT

Pipeline for running CUDA on the RISC-V GPU



Features:

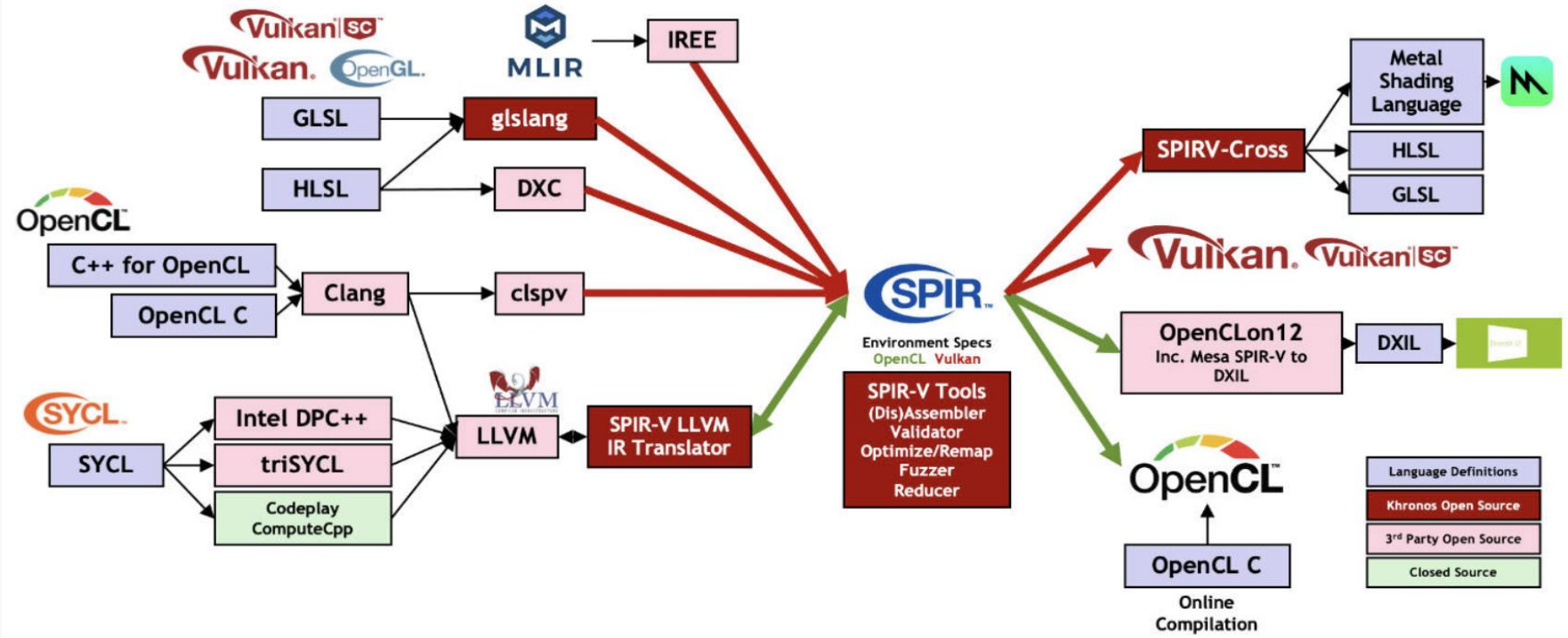
- Lightweight
- Scalable
- Open-Source

Reference: "Bringing OpenCL to Commodity RISC-V CPUs" CARRV 2021

Background: code migration

- HIPIFY: Only for CUDA->HIP
- SYCL: Re-implement CUDA programs by SYCL libraries

Background: SPIR

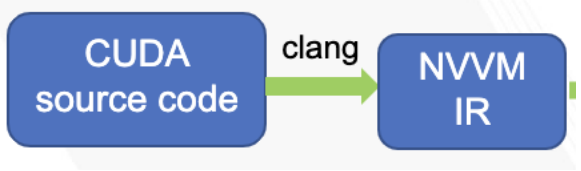


The SPIR-V ecosystem includes a rich variety of language front-ends, tools and run-times

source: https://www.khronos.org/spir/

Georgia Tech
CREATING THE NEXT

OVERVIEW OF THE PIPELINE: CUDA->NVVM IR



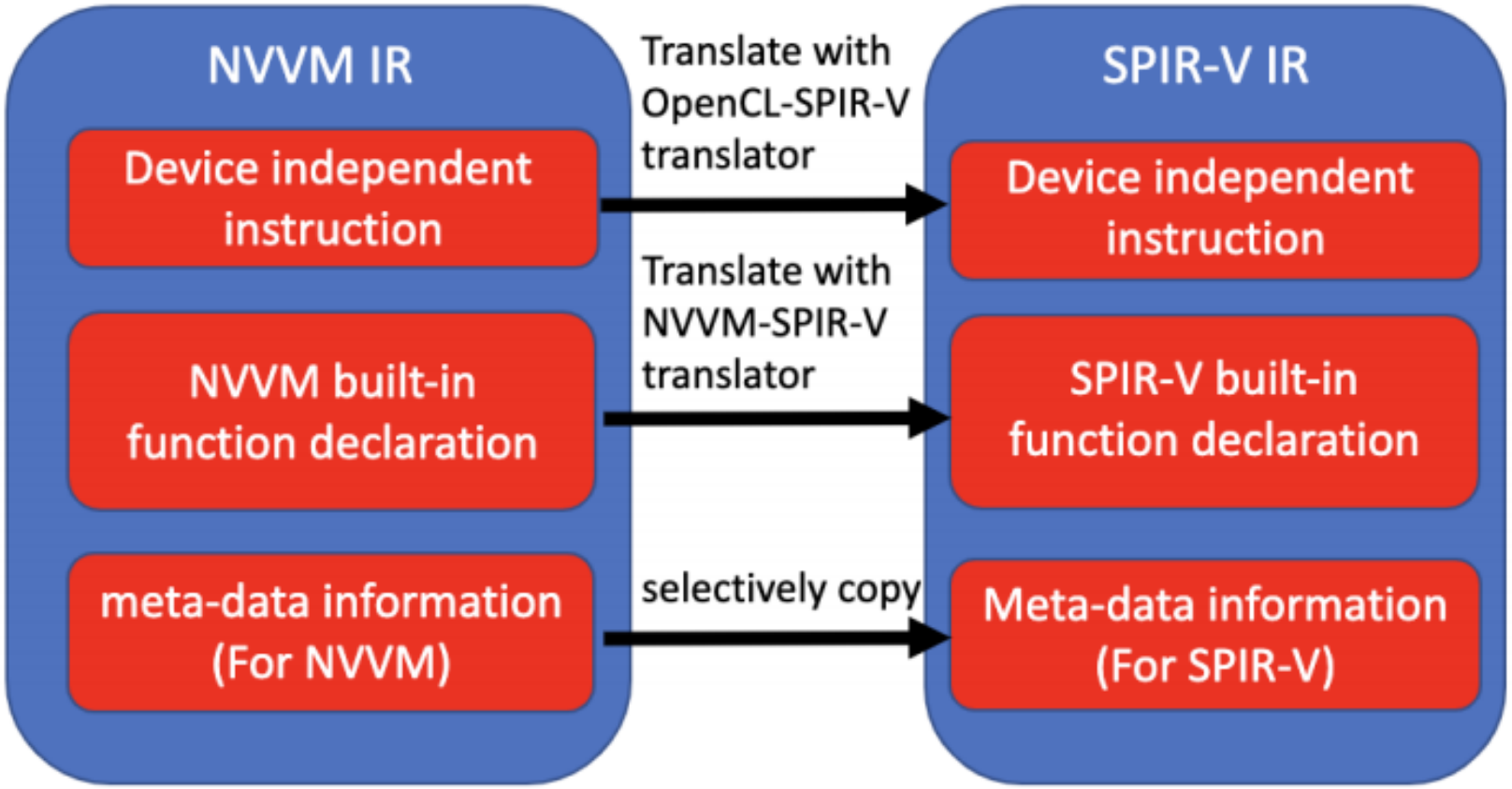

```
1   __global__ void vecadd (int *a, int *b, int *c) {
2     int gid = threadIdx.x;
3     c[gid] = a[gid] + b[gid];
4   }
```

Code 2: NVVM VectorAdd IR

```
1   target triple = "nvptx64-nvidia-cuda"
2
3   define dso_local void @vecadd(
4       i32* nocapture readonly %a,
5       i32* nocapture readonly %b,
6       i32* nocapture %c) {
7   entry:
8     ; see Fig. 2(a) for deatil code
9   }
10  declare i32 @llvm.nvvm.read.ptx.sreg.tid.x()
11
12  !nvvm.annotations = !{!3}
13  !3 = !{void (i32*, i32*, i32*) *@vecadd,
14          !"kernel", i32 1}
```

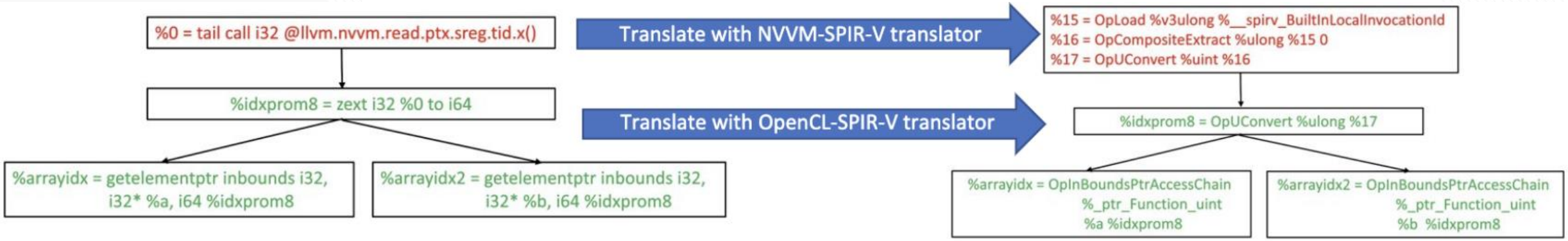
Georgia Tech
CREATING THE NEXT

OVERVIEW OF THE PIPELINE: NVVM IR->SPIR-V IR



Github link:

Example for NVVM-SPIR-V translation



For device independent instructions (load, store, binaryOp…): use OpenCL-SPIR-V translator
For NVVM special built-in function (llvm.nvvm.read.ptx.sreg.tid.x()...): use NVVM-SPIR-V translator

OVERVIEW OF THE PIPELINE: SPIR-V->OpenCL IR

```
%9 = OpTypeFunction %void
                    %_ptr_Function_uint
                    %_ptr_Function_uint
                    %_ptr_Function_uint
%__spirv_BuiltInLocalInvocationId = OpVariable %_ptr_Input_v3ulong Input
%10 = OpFunction %void None %9
...
%15 = OpLoad %v3ulong %__spirv_BuiltInLocalInvocationId
%16 = OpCompositeExtract %ulong %15 0
%17 = OpUConvert %uint %16
 %idxprom8 = OpUConvert %ulong %17
 %arrayidx = OpInBoundsPtrAccessChain %_ptr_Function_uint %a %idxprom8
%20 = OpLoad %uint %arrayidx Aligned 4
%arrayidx2 = OpInBoundsPtrAccessChain %_ptr_Function_uint %b %idxprom8
%22 = OpLoad %uint %arrayidx2 Aligned 4
%add = OpIAdd %uint %22 %20
%arrayidx4 = OpInBoundsPtrAccessChain %_ptr_Function_uint %c %idxprom8
OpStore %arrayidx4 %add Aligned 4
OpReturn
OpFunctionEnd
```

```
define spir_kernel void @_Z6vecaddPiS_S_(i32* %a, i32* %b, i32* %c) {
entry:
  %0 = call spir_func i64 @_Z12get_local_idj(i32 0) #1
  %1 = trunc i64 %0 to i32
  %idxprom8 = zext i32 %1 to i64
  %arrayidx = getelementptr inbounds i32, i32* %a, i64 %idxprom8
  %2 = load i32, i32* %arrayidx, align 4
  %arrayidx2 = getelementptr inbounds i32, i32* %b, i64 %idxprom8
  %3 = load i32, i32* %arrayidx2, align 4
  %add = add i32 %3, %2
  %arrayidx4 = getelementptr inbounds i32, i32* %c, i64 %idxprom8
  store i32 %add, i32* %arrayidx4, align 4
  ret void
}

declare spir_func i64 @_Z12get_local_idj(i32) #1
```

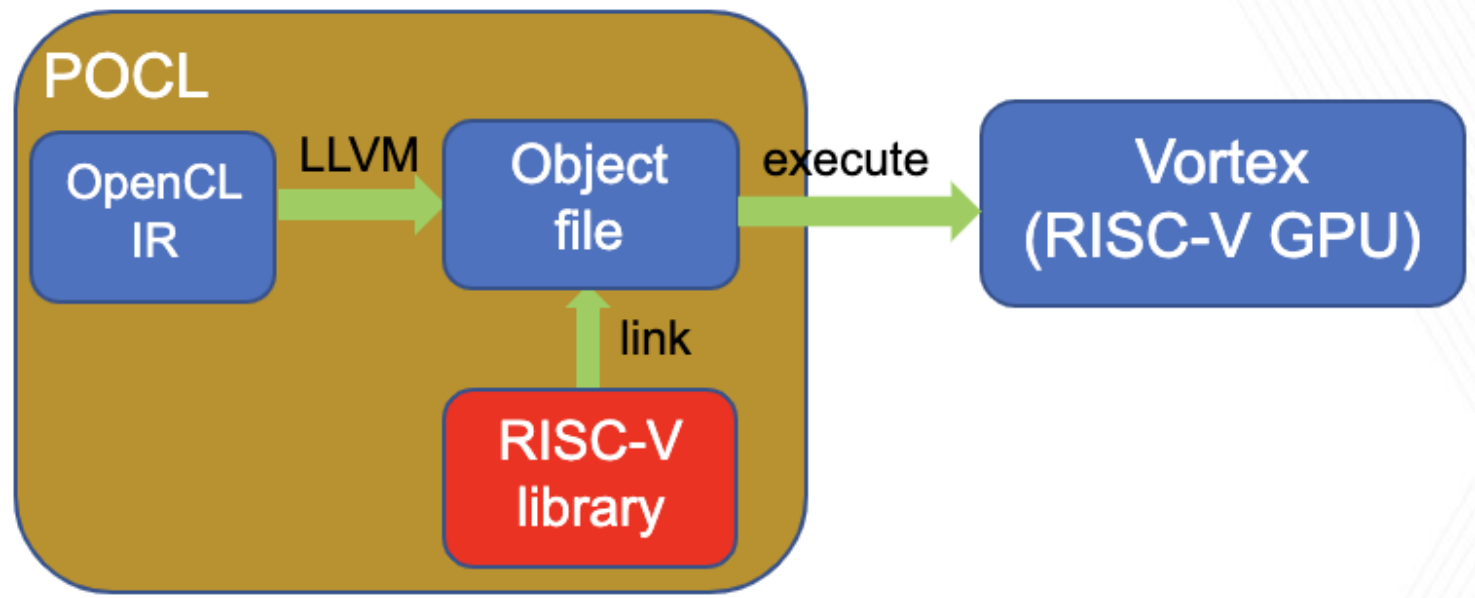
SPIR-V

OpenCL IR

SPIRV-LLVM-Translator: https://github.com/KhronosGroup/SPIRV-LLVM-Translator

Georgia Tech
CREATING THE NEXT

OVERVIEW OF THE PIPELINE: OpenCL->Object file



POCL for Vortex: https://github.com/vortexgpgpu/pocl
Vortex: https://github.com/vortexgpgpu/vortex

Experiments

application	feature	support?
b+tree	-	yes
bfs	-	yes
cfd	double3 type	yes
huffman	atomic	yes
pathfinder	memory hierachy	yes
gaussian	-	yes
hotspot	-	yes
hotspot3D	-	yes
lud	memory hierachy	yes
nw	-	yes
streamcluster	-	yes
particlefilter	d2i	on going
backprop	__log2f	on going
lavaMD	d2i	on going
kmeans	texture	no
hybrid sort	texture	no
leukocyte	texture	no

Table 2: Translating applications in Rodinia benchmark

Version:

Vortex-pocl: commit log 6272e2
Vortex: v0.2.2
NVPTX-SPIR-V translator: v0.1.0

Analysis:

- Support built-in function is the key to support general programs
- Well support for primitive instructions
- For new features, need to modify both translators and RISC-V library for Vortex

Rodinia: https://github.com/yuhc/gpu-rodinia

Georgia Tech
CREATING THE NEXT

Support feature (NVPTX-SPIR-V translator: version v0.1.0, Vortex: version v0.2.0)

- Memory hierarchy
- Synchronization
- Some mathematics operations (square root, log, absolute value)

function name	detail
llvm.nvvm.read.ptx.sreg.ctaid	get the block index
llvm.nvvm.read.ptx.sreg.ntid	get the block dimension
llvm.nvvm.read.ptx.sreg.tid	get the thread index
llvm.nvvm.barrie	synchronize threads within a block
llvm.sqrt	calculte the square root
llvm.fabs	calculate the absoulte value
llvm.nvvm.d2i	narrowing convertions
llvm.fma	fused multiply–add

Conclusion

- We propose and implement a lightweight, scalable pipeline for running CUDA source code on an open-source RISC-V GPU
- We build a translator to convert from NVVM IR to SPIR-V IR
- We can execute most of sample in a widely used benchmark

contact information:vortexgpu@cc.gatech.edu