arm

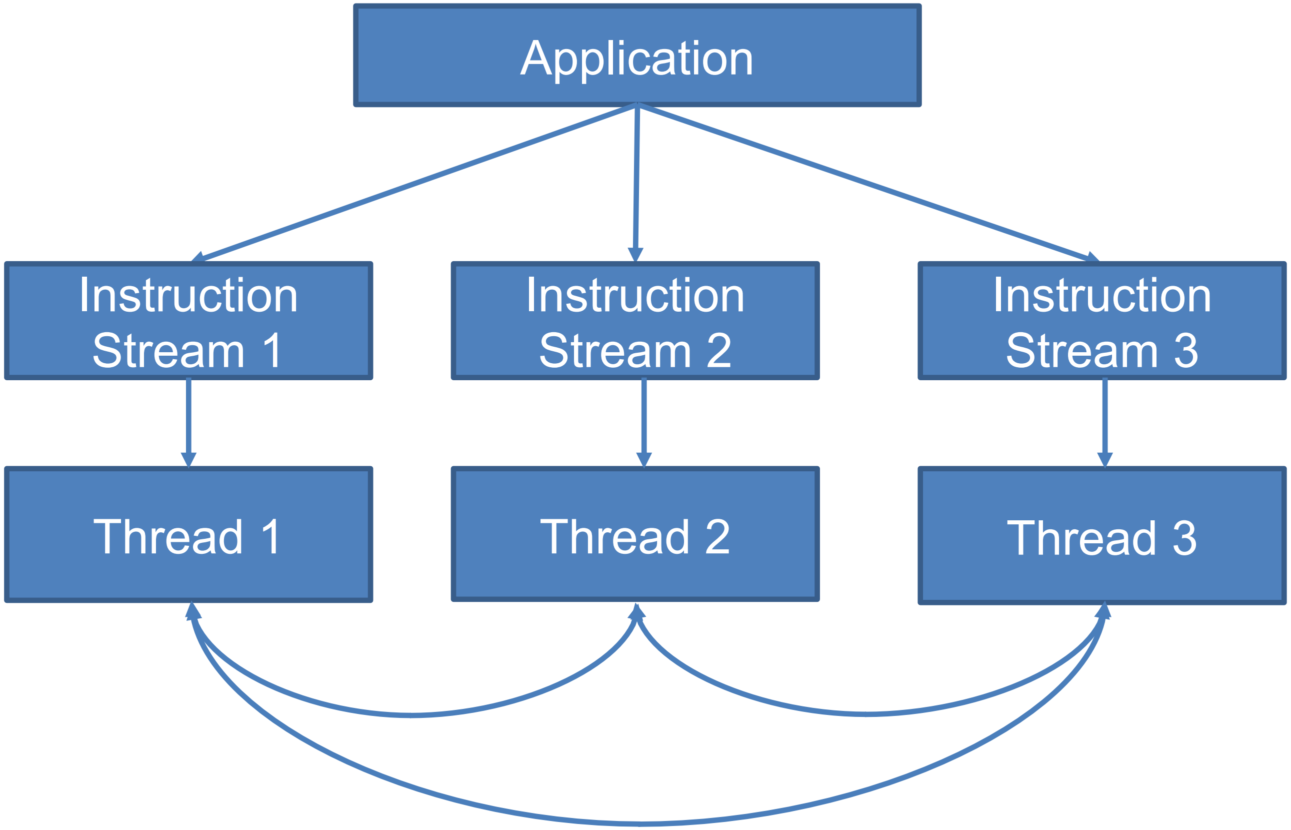
Art class for Dragons: Supporting GPU compilation without metadata hacks!

Neil Hickey

Overview

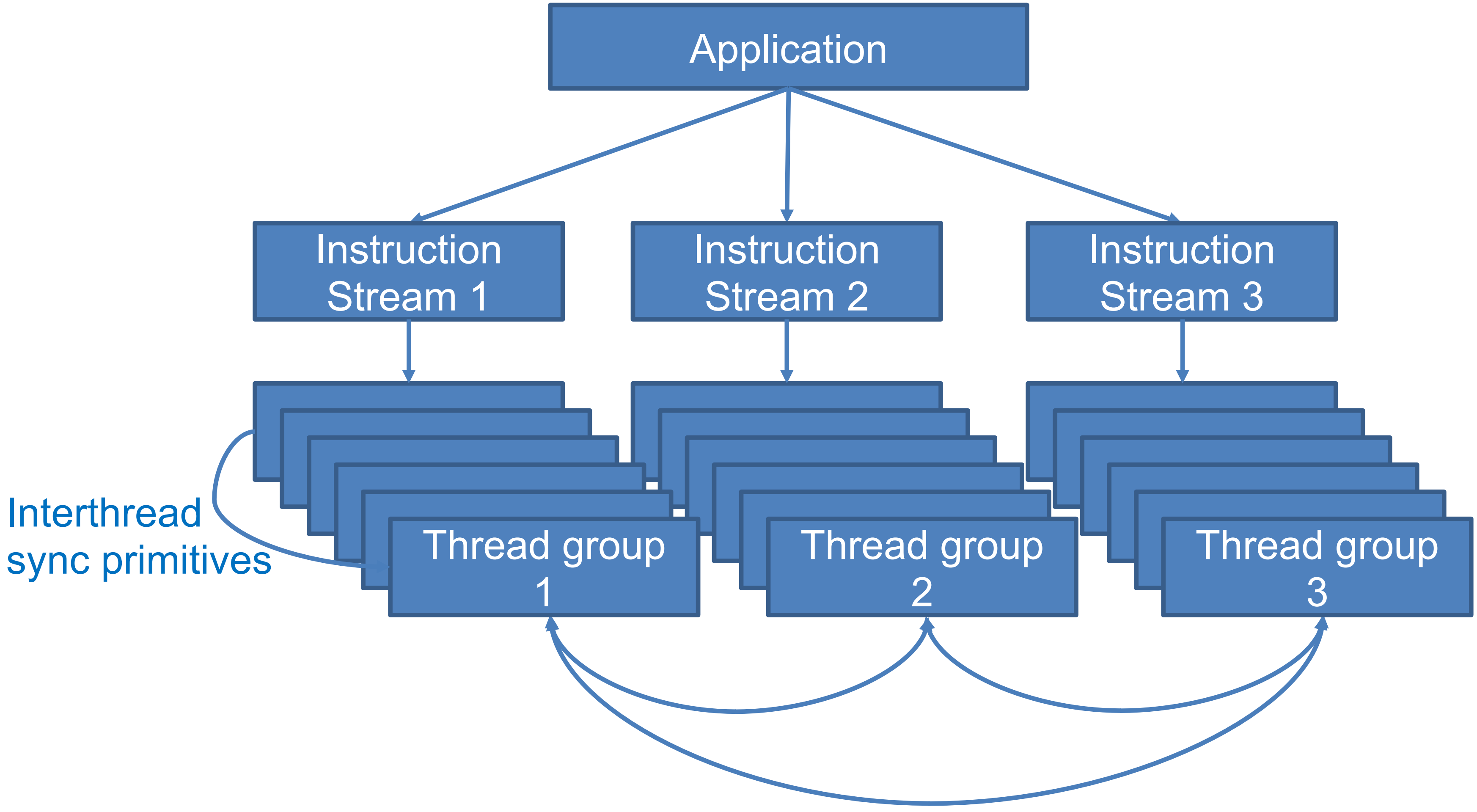
- Execution model, Compute and Graphics on GPU
- Introduction to Vulkan semantics and mapping to LLVM IR
- Other tools for GPU execution
- Better solutions needed

GPUs vs CPUs



Synchronization primitives

GPUs vs CPUs



Inter-thread group sync primitives

GPU programming languages

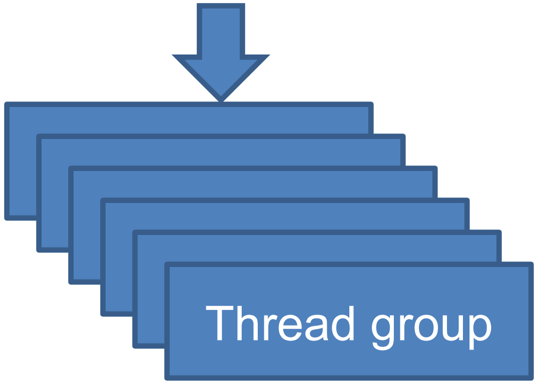
- OpenCL
- GLSL/ESSL
- Vulkan GLSL

OpenCL – massively parallel compute

```
void matmul(float *A, float *B, float *C,
              unsigned int dim) {
  unsigned int I = 0, J = 0, K = 0;
  for (I = 0; I < dim; ++I) {
    for (J = 0; J < dim; ++J) {
      C[I*dim+J] = 0;
      for (K = 0; K < dim; ++K) {
        C[I*dim+J]+=A[I*dim+K] * B[K*dim+J];
      }
}
```

OpenCL – massively parallel compute

```
kernel void matmul(float *A, float *B,
                   float *C,
                   unsigned int dim) {
  unsigned int K = 0;
  unsigned int I = get_global_id(0);
  unsigned int J = get_global_id(1);
  C[I*dim+J] = 0;
  for (K = 0; K < dim; ++K) {
    C[I*dim+J]+=A[I*dim+K] * B[K*dim+J];
  }
}
```



Vulkan – massively parallel graphics

0.7843265

Pi or something

0.1686

0.31419

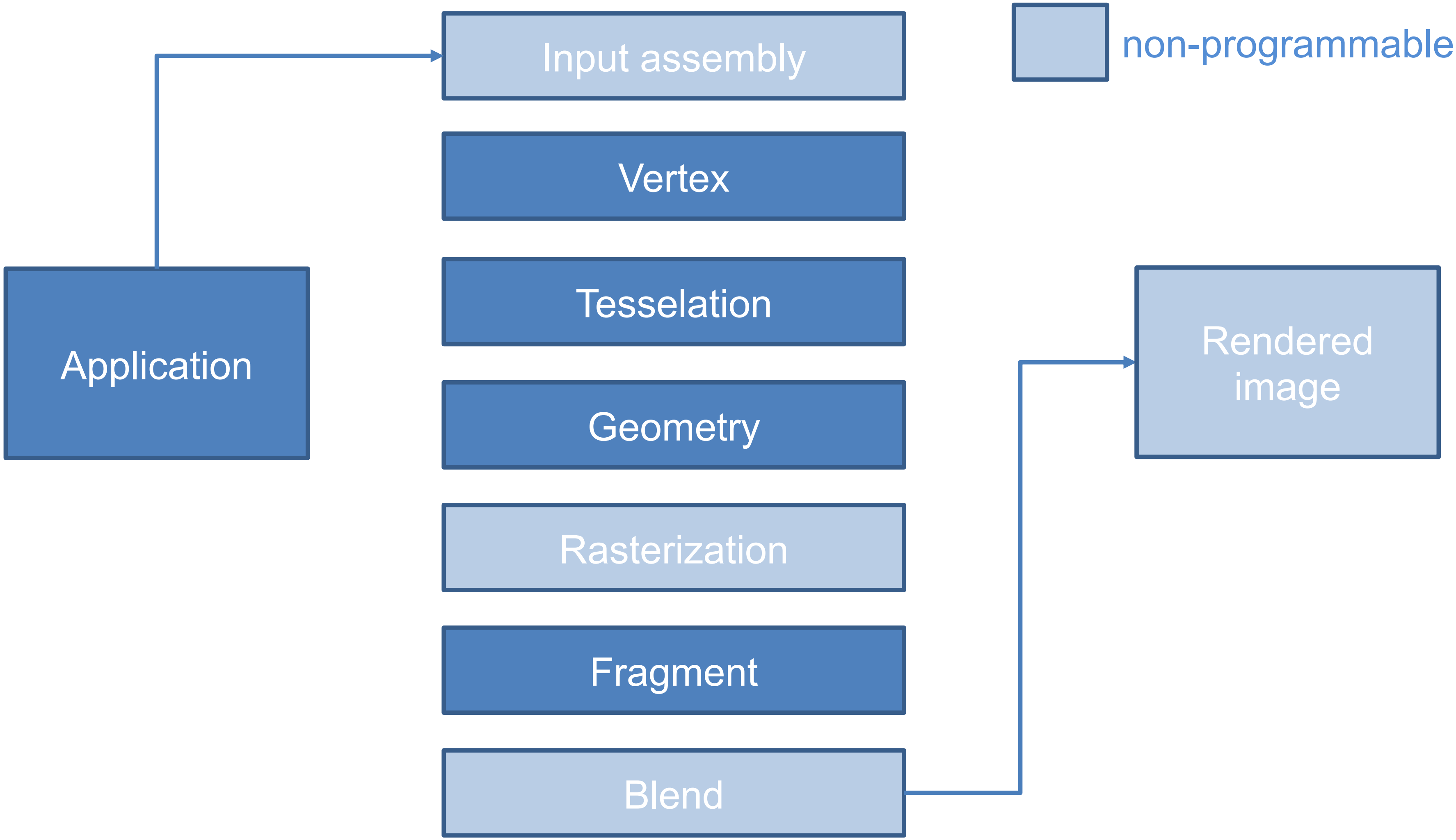
0.82

MAGIC

Graphics pipeline

■ programmable

□ non-programmable



Application

Input assembly

Vertex

Tesselation

Geometry

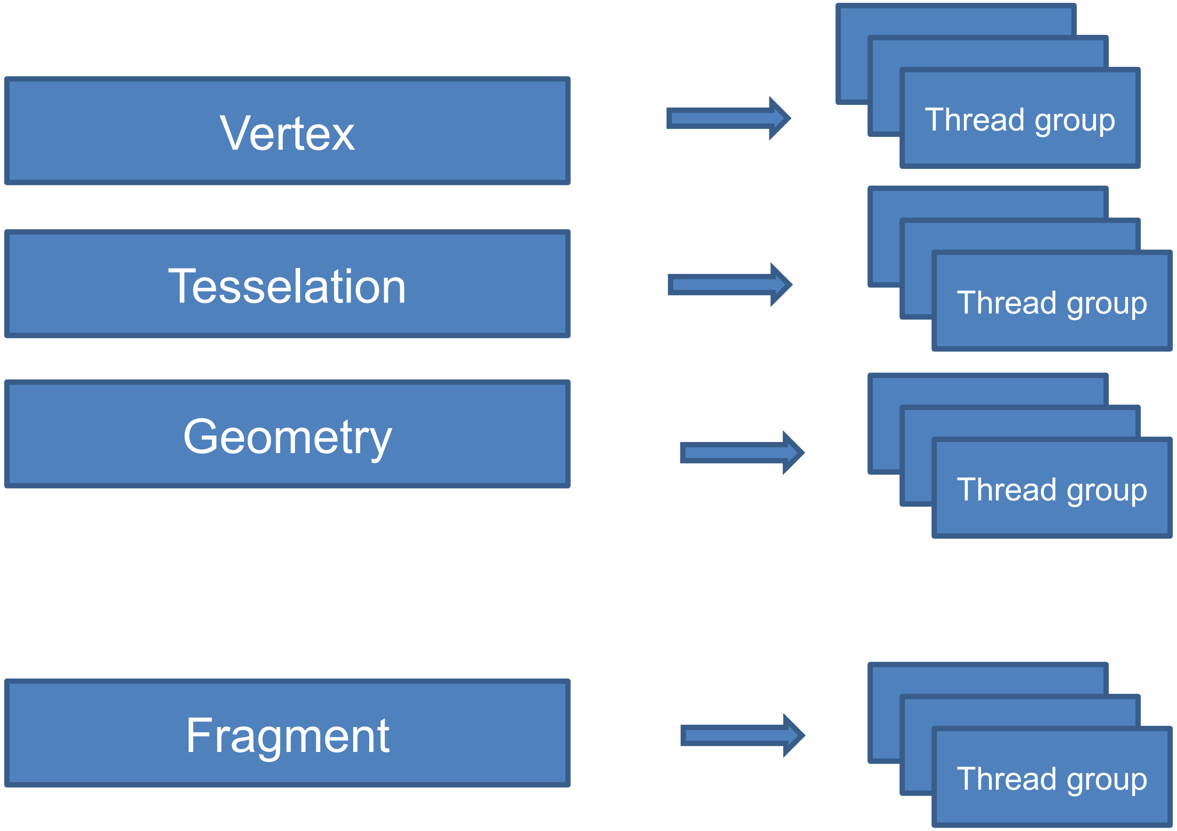
Rasterization

Fragment

Blend

Rendered image

Graphics pipeline



Vertex

Thread group

Tesselation

Thread group

Geometry

Thread group

Fragment

Thread group

Vulkan shader language

- Similar concepts to OpenCL
- Strongly correlated to how GPUs work
- Targeting massively parallel devices

Vulkan shader language

- Native types – images, samplers, vector, matrix
- Aware of its neighbours – derivatives/ subgroup operations
- Multiple types of memory regions (address spaces)

Vulkan GLSL

```
#version 310 es

layout (location = 0) in vec4 pos;

void main(void)
{
    gl_Position = pos;
}
```

Representing in LLVM

```
@pos = external addrspace(5) global <4
x float> !0

@0 = external addrspace(6) global { <4
x float>, float }


!0 = !{i32 0}
```

Vulkan GLSL(2)

```
#version 310 es

layout (location = 0) in vec4 pos;

void main(void)
{
    if (gl_InstanceIndex = 1)
        gl_Position = pos;
    else
        gl_Position = vec4(0.0);
}
```

Representing in LLVM(2)

```
@gl_InstanceIndex = external
addrspace(5) global i32 !0

!0 = !{i32 40}
```

Layout

Specify structure layout

- Memory layout
- Descriptor set and binding
- Structure offset

Memory Layout

- shared – not valid in Vulkan
- packed – not valid in Vulkan
- std140
- std430

Memory Layout

```
layout(std140,binding=1) uniform BL
{
  vec4 arp[7];
  int arg;
} nm;

@nm = external addrspace(7) global { [7
x <4 x float>], i32 }, !spirv.Block !2

!2 = !{{ { i32, i64, i64 }, i64 } { {
i32, i64, i64 } { i32 16, i64 0, i64 0
}, i64 64 }}
```

Memory offset

```
layout(location=0, component=1) in
float in_f1[2];
layout(location=2, component=0) flat in
int in_f;

@in_f1 = external addrspace(5) global
[2 x float], !0
@in_f = external addrspace(5) global
i32, !1

!0 = !{{ i32, i32, i32 } { i32 1, i32
0, i32 0 }}
```

Usage of address spaces

Concept	SPIR-V SC	AS used in DXIL	AS used in LLVM-Translator	AS used in NVVM
generic	Generic	--	4	0
private	Function	0	0	5
gl private	Private	0	0	3
local	WorkGroup	0	3	3
global	CrossWorkgroup	3	1	1
constant	UniformConstant	2	2	4

😢

Additional address spaces added for graphics

- In
- Out
- StorageBuffer
- PushConstant
- Uniform
- AtomicCounter

Challenges for optimizations

- No common meaning for address spaces
- Optimisations have to be conservative
- Special optimisations need to be written to handle generic (InferAddressSpace)

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!

감사합니다

धन्यवाद

arm