Accelerating Renderscript applications using OpenCL SPIR

Lukáš Kuklínek,
Codeplay Software

January 21, 2015

Codeplay Software Ltd.

- Incorporated in 1999
- Based in Edinburgh, Scotland
- Compilers, optimisation, programming models
- Partner in three FP7 research projects:
  - Peppher, LPGPU, CARP
- Contributing member of Khronos group since 2006

About the CARP project

Framework Programme 7 project

- Imperial College London
- ARM Ltd.
- RWHT Aachen University
- University of Twente
- École Normale Supérieure
- Realeyes
- Codeplay Software Ltd.
- Rightware

RenderScript and SPIR overview

RenderScript

- Compute API by Google for Android
- Java host API
- Kernel language based on C99
- The way data are passed to a kernel differs from OpenCL

SPIR

- Khronos Group standard
- Standard Portable Intermediate Representation
- Encoding of OpenCL kernels in terms of LLVM IR

Transformation example

$$out_i = a \cdot in_i^2$$

```
float a;

float scale_by_a(float x          ) {
   return a * x;
}

float kernel square(float  in                        )
{

   return      scale_by_a(in * in);
}
```

Transformation example

$$out_i = a \cdot in_i^2$$

```
float a;

float scale_by_a(float x          ) {
   return a * x;
}

float kernel square(float  in                          )
{
   int i = get_global_id(0);
   return     scale_by_a(in * in);
}
```

Transformation example

$$out_i = a \cdot in_i^2$$

```
float a;

float scale_by_a(float x           ) {
   return a * x;
}

float kernel square(float *inbuf                          )
{
   int i = get_global_id(0);
   return     scale_by_a(inbuf[i] * inbuf[i]);
}
```

Transformation example

$$out_i = a \cdot in_i^2$$

```
float a;

float scale_by_a(float x          ) {
   return a * x;
}

void  kernel square(float *inbuf, float *outbuf          )
{
   int i = get_global_id(0);
   outbuf[i] = scale_by_a(inbuf[i] * inbuf[i]);
}
```

Transformation example

$$out_i = a \cdot in_i^2$$

```
float a;

float scale_by_a(float x          ) {
    return a * x;
}

void  kernel square(float *inbuf, float *outbuf          )
{
    int i = get_global_id(0);
    outbuf[i] = scale_by_a(inbuf[i] * inbuf[i]);
}
```

Transformation example

$$out_i = a \cdot in_i^2$$

```
float a;

float scale_by_a(float x, float g_a) {
    return g_a * x;
}

void  kernel square(float *inbuf, float *outbuf            )
{
    int i = get_global_id(0);
    outbuf[i] = scale_by_a(inbuf[i] * inbuf[i]);
}
```

Transformation example

$$out_i = a \cdot in_i^2$$

```
float a;

float scale_by_a(float x, float g_a) {
    return g_a * x;
}

void  kernel square(float *inbuf, float *outbuf, float g_a)
{
    int i = get_global_id(0);
    outbuf[i] = scale_by_a(inbuf[i] * inbuf[i], g_a);
}
```

Transformation example

$$out_i = a \cdot in_i^2$$

```
float scale_by_a(float x, float g_a) {
    return g_a * x;
}

void kernel square(float *inbuf, float *outbuf, float g_a)
{
    int i = get_global_id(0);
    outbuf[i] = scale_by_a(inbuf[i] * inbuf[i], g_a);
}
```

Address space inference

- In order to generate valid SPIR we have to annotate pointers with address space qualifiers, e.g. global, private, etc.
- RenderScript source or IR does not provide any address space information.
- Address spaces are inferred using a Hindley-Milner style unification-based algorithm from the IR.
- The result is a set of equality constraints on the address space annotations present in the source. We use the handy `EquivalenceClasses` data structure to represent the constraints.
- The set of computed constraints shall be minimal (i.e. as general as possible) such that any assignment that satisfies the constraints shall yield a type-correct LLVM module.

Address space inference example

Compute a weighted maximum of `%buf`, returning a pointer to the maximal element found.

```
define i32*   @weightedmax(i32*   %buf, i32*   %weights, i32 %len) {
entry:
  br label %loop
loop:
  %i          = phi i32 [ 0, %entry ], [ %inxt, %loop ]
  %max        = phi i32 [ 0, %entry ], [ %maxnxt, %loop ]
  %maxptr     = phi i32*    [ %buf, %entry ], [ %maxptrnxt, %loop ]
  %eltptr     = getelementptr i32*   %buf, i32 %i
  %wgtptr     = getelementptr i32*   %weights, i32 %i
  %elt        = load i32*   %eltptr
  %wgt        = load i32*   %wgtptr
  %welt       = mul i32 %elt, %wgt
  %gt         = icmp ugt i32 %welt, %max
  %maxnxt     = select i1 %gt, i32 %welt, i32 %max
  %maxptrnxt  = select i1 %gt, i32*   %eltptr, i32*   %maxptr
  %inxt       = add i32 %i, 1
  %continue   = icmp ult i32 %i, %len
  br i1 %continue, label %loop, label %endloop
endloop:
  ret i32*   %maxptrnxt
}
```

Address space inference example

Compute a weighted maximum of %buf, returning a pointer to the maximal element found.

```
define i32 AS(A)* @weightedmax(i32 AS(A)* %buf, i32 AS(B)* %weights, i32 %len) {
entry:
  br label %loop
loop:
  %i          = phi i32 [ 0, %entry ], [ %inxt, %loop ]
  %max        = phi i32 [ 0, %entry ], [ %maxnxt, %loop ]
  %maxptr     = phi i32 AS(A)* [ %buf, %entry ], [ %maxptrnxt, %loop ]
  %eltptr     = getelementptr i32 AS(A)* %buf, i32 %i
  %wgtptr     = getelementptr i32 AS(B)* %weights, i32 %i
  %elt        = load i32 AS(A)* %eltptr
  %wgt        = load i32 AS(B)* %wgtptr
  %welt       = mul i32 %elt, %wgt
  %gt         = icmp ugt i32 %welt, %max
  %maxnxt     = select i1 %gt, i32 %welt, i32 %max
  %maxptrnxt = select i1 %gt, i32 AS(A)* %eltptr, i32 AS(A)* %maxptr
  %inxt       = add i32 %i, 1
  %continue   = icmp ult i32 %i, %len
  br i1 %continue, label %loop, label %endloop
endloop:
  ret i32 AS(A)* %maxptrnxt
}
```

Function instantiation

- We have to instantiate the kernels to operate on particular address spaces. The instantiation is driven by the address space inference.
- For now, any kernel pointer agument is allocated into the global address space.
- Address spaces of function arguments are determined by the call sites. Function operating on different address spaces are generated on demand.
- Any pointers that remain polymorphic in address space are assigned *private* AS by default.

Function Instantiation

- Sometimes a function is called from multiple contexts so it is called with arguments operating on multiple different address spaces.
- Results in multiple copies of the function being generated.

Example:

```
void nonneg (        float * p) { *p = max(*p, 0.0); }


void kernel foo(       float *ptr) {
  float x = random();
  nonneg (&x);     // &x   :         float *
  nonneg (ptr);    // ptr  :         float *
}
```

Function Instantiation

- Sometimes a function is called from multiple contexts so it is called with arguments operating on multiple different address spaces.
- Results in multiple copies of the function being generated.

Example:

```
void nonneg (???     float * p) { *p = max(*p, 0.0); }


void kernel foo(global float *ptr) {
  float x = random();
  nonneg (&x);     // &x  : private float *
  nonneg (ptr);    // ptr : global  float *
}
```

Function Instantiation

- Sometimes a function is called from multiple contexts so it is called with arguments operating on multiple different address spaces.
- Results in multiple copies of the function being generated.

Example:

```
void nonneg0(private float * p) { *p = max(*p, 0.0); }
void nonneg1(global  float * p) { *p = max(*p, 0.0); }
```

```
void kernel foo(global float *ptr) {
  float x = random();
  nonneg (&x);     // &x  : private float *
  nonneg (ptr);    // ptr : global  float *
}
```

Function Instantiation

- Sometimes a function is called from multiple contexts so it is called with arguments operating on multiple different address spaces.
- Results in multiple copies of the function being generated.

Example:

```
void nonneg0(private float * p) { *p = max(*p, 0.0); }
void nonneg1(global  float * p) { *p = max(*p, 0.0); }

void kernel foo(global float *ptr) {
  float x = random();
  nonneg0(&x);      // &x  : private float *
  nonneg1(ptr);     // ptr : global  float *
}
```

Host API & the future

Host API implementation:

- Replace parts of RenderScript runtime with our implementation
- Forward RenderScript API calls to OpenCL API calls

In progress:

- Full stack integration
- Gather performance data
- Experiment with kernel fusion

Summary

- Part of the CARP FP7 project
- RenderScript vs. OpenCL impedance mismatch
  - Scalar inputs
  - `get_global_id` vs arguments
  - Semantics of global variables
- Address space reconstruction
- Runtime support
- In progress: experiment with kernel fusion, evaluate performance