AMD
Smarter Choice

CAL Kernel Programming

Module Overview

AMD Smarter Choice

- CAL Kernel Programming APIs
- Overview of AMD IL
- Overview of AMD GPU ISA

CAL Kernel Programming APIs

AMD Smarter Choice

- AMD Brook+
- DirectX High Level Shading Language (HLSL)
- AMD Intermediate Language (IL)
- AMD GPU-specific Assembly Instruction Set (ISA)

AMD Brook+

AMD Smarter Choice

- 'C' with Streaming Extensions
- Includes
  - Kernel programming interface
  - Runtime implementation
    - Transparent to application programmer
    - Supports multiple runtimes
  - Runtime API
    - Simple and small API for basic operations
    - Provides only high-level control over stream data

DirectX High Level Shading Language (HLSL)

AMD
Smarter Choice

- 'C' for GPU Programming for 3D applications
- Includes
  - Kernel Programming Interface
  - Runtime provided by associated runtime library (DirectX)
  - Front-end compiler provided by Microsoft (`fxc`)
  - Back-end code generation and optimization done by GPU vendors
- Original BrookGPU included as DirectX runtime
  - Generate HLSL for Brook kernels
  - Invoke DirectX runtime calls for stream management and kernel invocation, etc

AMD Intermediate Language

AMD Smarter Choice

- Psuedo-assembly interface
- Interface is *Architecture Independent*
- Includes graphics-ish commands as well
- Evolves with GPU evolution
- Specifications available with the SDK installation

AMD GPU ISA

AMD Smarter Choice

- True GPU assembly
- Interface and Implementation is *Architecture Dependent*
  - Exposes the GPU architecture completely
  - Provides insight into GPU architecture and expected performance
- No optimizations done by the assembler on specified ISA
- Not expected to use directly for application development
  - Very useful for performance profiling
  - Useful for debugging programs

AMD
Smarter Choice

Overview of AMD IL

AMD IL Interface

AMD Smarter Choice

- An IL program consists of
  - Versioning information, declarations, etc
  - Registers (operands, temporaries and results)
  - Instructions
- IL registers are 4-component vectors
- IL instructions that accept vectors by default perform vector instructions
  - Current and future GPUs have superscalar units so scalar instructions are encouraged

  ⇒Use masking operators when vector instructions are not needed

  ⇒The compiler will optimize the generated code appropriately

AMD IL – Data Types

AMD Smarter Choice

- IL is a type-less language
  - Registers do not have specific types
  - Can be used to store 32-bit integers, floats, and 64-bit double precision values
  - Types are defined by the instructions, e.g.

    `IMUL`, `UMUL`, `MUL` and `DMUL`

    correspond to multiplication of

    signed integers, unsigned integers, floats and doubles respectively

AMD IL – Instruction Syntax

AMD Smarter Choice

```
<instr>[_<ctrl>][_<ctrl(val)>]
[<dst>[_<mod>][.<write-mask>]]
[, <src>[_<mod>][.<swizzle-mask>]]...
```

Items within `<>` are replaced by specific words or phrases. For example, `<instr>` could be replaced by `mov`.

When more than one item of the same type exists, for example `<src>`, a zero-based number is appended to the item name. For example, for 2-input instructions the sources (inputs) would be labeled `src0` and `src1`.

Items within square brackets, `[]`, are optional.

Items within braces, `{}`, represent a choice-grouping. For example, `{x|y|z|w}` means to choose `x`, `y`, `z` or `w`.

An ellipsis, `...`, represents a list of alternatives having the same format as the first alternative in the list.

Important ALU Instructions

AMD Smarter Choice

`ADD` – Addition of two registers

```
add dst, src0, src1
```

dst = src0 + src1

`MUL` – Multiplication of two registers

```
mul dst, src0, src1
```

dst = src0 * src1

`MAD` – Multiplication of two registers and addition with third

- More efficient as it performs 2 vector instructions in 1 cycle

```
mad[_ieee] dst, src0, src1, src2
```

dst = src0 * src1 + src2

Important ALU Instructions

`DIV` – Division of two registers

```
div_zeroop(op) dst, src0, src1
```

dst = src0 / src1

`zeroop()` decides value of output if divisor is zero, e.g.

`fltmax, zero, infinity, inf_else_max`

`RCP` – Reciprocal

```
rcp_zeroop(op) dst, src0
```

dst = 1/src0.w

`SQRT` – Square Root

```
sqrt dst, src0
```

dst = sqrt(src0.w)

IL Registers

IL provides registers for various tasks

Register	Description	Need to declare	Example
`r#`	Temporary	No	`mad r0, r0, r0, r0`
`cb#[#]`	Const buffer	Yes	`mov o0, cb0[0]`
`v0`	Position	Yes	`mul o0, v0.xyxx, cb0[0]`
`o#`	Output	Yes	`mov o0, v0.xyxx`

AMD Smarter Choice

Reading Inputs - Samplers

- Handle used to "Sample" or read from input buffers (textures in graphics)
  - specify the buffer from which to read
  - the address within this buffer – 2-tuple for 2D buffers
  - use sampling operator

C

```
// Buffer A, Address (x, y), Operator [][]
float temp = A[x][y];
```

Brook+

```
// Buffer A, Address vector reg.xy, Operator []
float templ = A[reg.xy];
```

IL

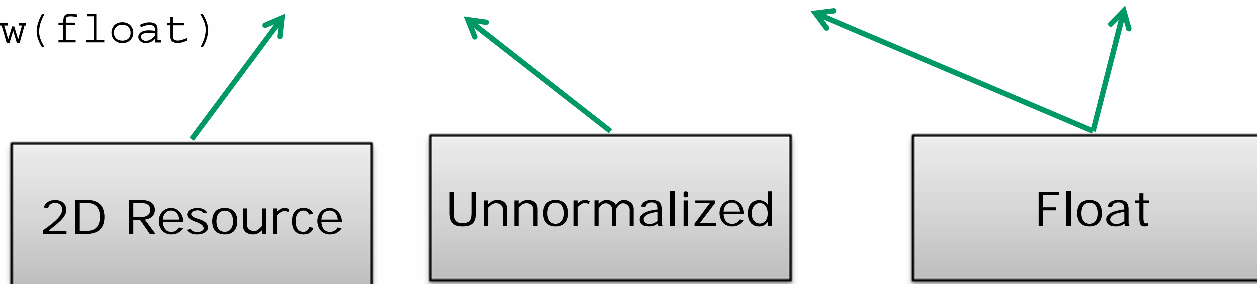
```
// Buffer resource 0, Address vector v0.xy, Operator
// sample_resource(#)_sampler(#)
sample_resource(0)_sampler(0) r0, v0.xy
```

Reading Inputs - Samplers

- Both the sample operator and output register are type-less
  – The data format and sampling mode needs to be specified in the kernel for each sampler

e.g.

```
dcl_resource_id(0)_type(2d,unnorm)_fmtx(float)_fmty(float)_fmt
  z(float)_fmtw(float)
```



Type of resource (1d vs 2d) specified

Type of address (normalized vs unnormalized) specified

– Normalized uses coordinates in [0..1] range

– Unnormalized uses coordinates in [0..n-1] range

Register Modifiers, Masks and Swizzles

AMD Smarter Choice

```
<instr>[_<ctrl>][_<ctrl(val)>]
[<dst>[_<mod>][.<write-mask>]]
[, <src>[_<mod>][.<swizzle-mask>]]...
```

Write Mask

Destination Modifier

Source Modifier

Swizzle Mask

Source Modifiers

AMD Smarter Choice

Modifier	Description	Example
`_invert`	Invert components (1- x)	`add r0, r1_invert, r2`
`_bias`	Elements are biased (x – 0.5)	`add r0, r1, r2_bias`
`_x2`	Multiply elements by 2.0	`add r0, r1, r2_x2`
`_bx2`	Signed scaling. Combined bias and x2 modifiers.	`add r0, r1, r2_bx2`
`_sign`	Signs Elements Elements less then 0 become -1 Elements equal to 0 become 0 Elements greater then 0 become 1	`mov r0, r1_sign`
`_divcomp(`*`type`*`)`	Performs division based on *type*. *type*: y, z, w, unknown	`texld_stage(0) r0, vT0_divcomp(y)`
`_abs`	Takes the absolute value of elements.	`mov r0, r1_abs`
`_neg(`*`comp`*`)`	Provides per element negate	`mov r0, r1_neg(xw)`

Destination Modifiers

Modifier	Description	Example

`_x2`
`_x4`
`_x8`
`_d2`
`_d4`
`_d8`
	Shift scale modifiers	`add_x2 r0, r1, r2`

`_sat`	Saturate or clamp result to [0,1]	`add_sat r0, r1, r2` `add_x2_sat r0, r1, r2`

Write Masks

- Mask a destination element using specified flag

```
reg.{x|_|0|1}{y|_|0|1}{z|_|0|1}{w|_|0|1}
```

  - Letter (`x`, `y`, `z`, `w`) => element to be written
  - Underscore (_) => element *not* to be written
  - Zero (`0`) or One (`1`) => element forced to zero or one
  - Omitting => underscore

E.g.

```
mov r0.x_zw, r1

mov r0.x_01, r1
```

Swizzle-mask

AMD Smarter Choice

- Manipulate the default positions of elements in a register to be used as a source/destination

`reg.{x|y|z|w|0|1}{y|z|w|x|0|1}{z|w|x|y|0|1}{w|x|y|z|0|1}`

  - Swizzle mask is position dependent
  - Letter (`x`, `y`, `z`, `w`) => element to be read/written
  - Zero (`0`) or One (`1`) => element forced to zero or one
  - Omitting => default value

E.g.

```
mov r0, r1.yz ; same as mov r0, r1.yzzw

mov r0, r1.yzx1 ; force r0.w to be 1
```

Comments

Semi-colon used for comment

e.g.

```
; this is a comment

mov r0, r1 ; this is another comment
```

AMD
Smarter Choice

Overview of AMD GPU ISA

AMD GPU ISA

AMD Smarter Choice

- Reviewing assembly is handy for performance profiling and debugging
- CAL provides interfaces to dump dis-assembly from `CALimage`

```
calclDisassembleImage

void calclDisassembleImage(const CALimage image,
CALLogFunction logfunc);
```

- Disassembles the CAL image

```
calclDisassembleObject

void calclDisassembleObject(const CALobject* obj,
CALLogFunction logfunc);
```

- Disassembles the CAL object

AMD GPU ISA

AMD Smarter Choice

- Disassembler
  - Both outputs the disassembled ISA on a line-by-line basis
  - Uses application specified log function

```
typedef void (*CALLogFunction)(const char* msg);
void disasm(const char *msg) {
  fprintf(stderr, "%s\n", msg);
}
calclDisassembleImage(image, disasm);
calclDisassembleObject(object, disasm);
```

- Command line utilities – `amudisasm`, `amuasm`
  - Dis-assemble a given binary image
  - Assemble a given text GPU ISA kernel

AMD IL to GPU ISA

AMD Smarter Choice

INPUT: AMD IL Kernel

OUTPUT: R600 ISA

GPU ShaderAnalyzer - IL

File Edit Help

Source Code

Function

```
1 // Enter your shader in this window
2 il_ps_2_0
3 dcl_input_interp(linear) v0
4 dcl_output_generic o0
5 dcl_cb cb0[1]
6 mul o0, v0.xy, cb0[0]
7 ret_dyn
8 end
9
```

Compile

HLSL Compiler

Macro Definitions

Bool Constants

Object Code

Format: Radeon HD 2900 (R600) Assembly

```
; -------------- PS Disassembly --------------------
00 ALU: ADDR(32) CNT(4) KCACHE0(CB0:0-15)
      0  x: MUL   R0.x,  R0.x,  KC0[0].x
         y: MUL   R0.y,  R0.y,  KC0[0].y
         z: MUL   R0.z,  R0.y,  KC0[0].z
         w: MUL   R0.w,  R0.y,  KC0[0].w
01 EXP_DONE: PIX0, R0
END_OF_PROGRAM
```

Compiler Statistics (Using Catalyst 7.12)

Name	GPR	Min	Max	Avg	Est Cycles(Bi)	ALU:TEX(Bi)	Est Cycles(Tri)	ALU:TEX(Tri)	Est Cycles(Aniso)	ALU:TEX(Aniso)	BottleNeck(Bi)	BottleNeck(Tri)	BottleNeck(Aniso)	Pixel\Clock(Bi)	Pixel\Clock(Tri)	Pixel\Clock(Aniso)	Scrat
Radeon 9700	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
Radeon x800	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
Radeon x850	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
Radeon x1800	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
Radeon x1300	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
Radeon x1600	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
Radeon x1900	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
Radeon HD 2900	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	EXP	EXP	EXP	16.00	16.00	16.00	
Radeon HD 2400	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	EXP	EXP	EXP	4.00	4.00	4.00	
Radeon HD 2600	2	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	EXP	EXP	EXP	4.00	4.00	4.00	
Radeon HD 3870	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	EXP	EXP	EXP	16.00	16.00	16.00	

D3D Assembly Statistics

Performance statistics for different GPUs

AMD GPU ISA - Program Structure

AMD Smarter Choice

- ISA Program consists of set of Instruction Clauses
  - ALU Clauses
  - TEX Clauses
- ISA Maps directly to the underlying hardware
  - Computational Core consists of 5-way ALU units
  - ALU Clause consists of one or more ALU Instruction Groups
  - ALU Instruction Group consists of up-to 5 ALU instructions
    - Represented by x, y, z, w, t
- TEX Clauses perform memory read operations
  - Texture mapping in 3D graphics
  - Additional Flags are used for compute-specific features

AMD GPU ISA - Program Example

AMD Smarter Choice

```
kernel void
test(int N,
        float A<>,
        float B<>,
        out float C<>)
{
        float a = A + 5.0f;
        float b = B + 5.0f;
        C = a + b;
}
```

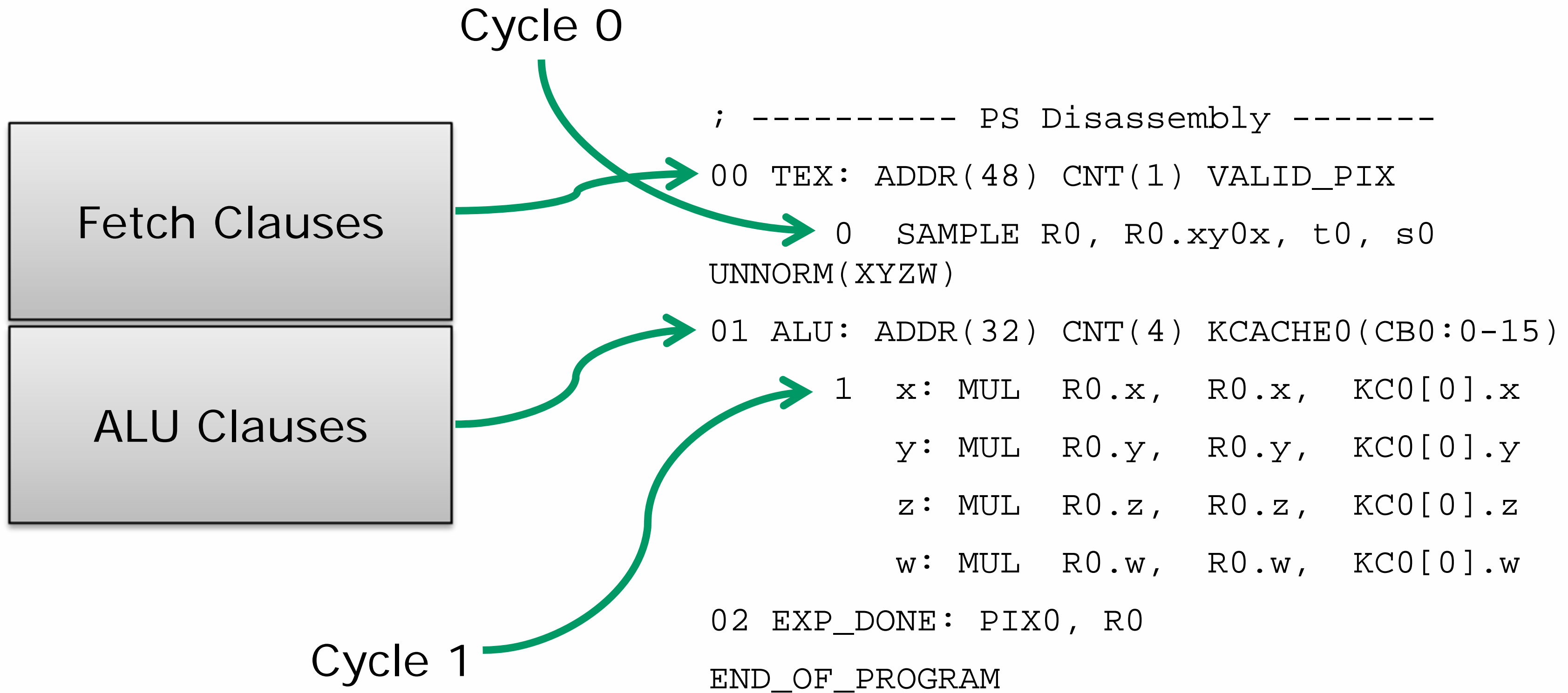
```
00 TEX: ADDR(48) CNT(2) VALID_PIX
    0  SAMPLE R1.x___, R0.xyxx, t0, s0 UNNORM(XYZW)
    1  SAMPLE R0.x___, R0.xyxx, t1, s0  UNNORM(XYZW)
01 ALU: ADDR(32) CNT(5)
    2  y: MOV          R0.y,  0.0f
       z: ADD          ____,  R0.x,  (0x40A00000, 5.0f).x
       w: ADD          ____,  R1.x,  (0x40A00000, 5.0f).x
    3  x: ADD          R0.x,  PV2.w,  PV2.z
02 EXP_DONE: PIX0, R0.xyyy
END_OF_PROGRAM
```

AMD GPU ISA

AMD Smarter Choice

- Reveals Information on
  - Actual number of ALU cycles in kernel
  - Data Dependencies
  - ALU utilization
- Does not reveal Information on
  - Total number of cycles for execution
    - Memory access latencies are not accounted for

AMD GPU ISA - Program Structure



Important Tokens

AMD Smarter Choice

`R#` - 128-bit GPRs

`KCACHE#` - GPU Constant Cache

`SAMPLE` – Memory Read Operation

`F_TO_I` – Floating point to Integer conversion

`LOOP_DX10` – DX10-style **`for`** loop

Q&A and Recap

AMD
Smarter Choice

- CAL Kernel Programming Interfaces
  - Brook+
  - HLSL
  - AMD IL
  - AMD GPU ISA