

# HSA AND THE MODERN GPU

Lee Howes AMD Heterogeneous System Software



# HSA AND THE MODERN GPU

In this brief talk we will cover three topics:

- Changes to the shader core and memory system
- Changes to the use of pointers
- Architected definitions to use these new features
- We'll look both at how the hardware is becoming more flexible, and how the changes will benefit OpenCL implementations.



# THE HD7970 AND GRAPHICS CORE NEXT



# **GPU EXECUTION AS WAS**

We often view GPU programming as a set of independent threads, more reasonably known as "work items" in OpenCL:

```
kernel void blah(global float *input, global float *output) {
   output[get_global_id(0)] = input[get_global_id(0)];
}
```

Which we flatten to an intermediate language known as AMD IL: model

Note that AMD IL contains short vector instructions

mov r255, r1021.xyz0 mov r255, r255.x000 mov r256, I9.xxxx ishl r255.x , r255.xxxx, r256.xxxx iadd r253.x\_\_\_, r2.xxxx, r255.xxxx mov r255, r1022.xyz0 mov r255. r255.x000 ishl r255.x , r255.xxxx, r256.xxxx iadd r254.x , r1.xxxx, r255.xxxx mov r1010.x . r254.xxxx uav\_raw\_load\_id(11)\_cached r1011.x\_\_\_, r1010.xxxx mov r254.x\_\_\_\_, r1011.xxxx mov r1011.x , r254.xxxx mov r1010.x , r253.xxxx uav\_raw\_store\_id(11) mem.x\_\_\_, r1010.xxxx, r1011.xxxx ret



• The GPU hardware of course does not execute those work items as threads

- The reality is that high-end GPUs follow a SIMD architecture
  - Each work item describes a lane of execution
  - Multiple work items execute together in SIMD fashion with a single program counter
  - Some clever automated stack management to handle divergent control flow across the vector

The GPU hardware of course does not execute those work items as threads

The reality is that high-end GPUs follow a SIMD architecture

- Each work item describes a lane of execution
- Multiple work items execute together in SIMD fashion with a single program counter
- Some clever automated stack management to handle divergent control flow across the vector

mov r255, r1021.xyz0 mov r255. r255.x000 mov r256. 19.xxxx ishl r255.x , r255.xxxx, r256.xxxx iadd r253.x , r2.xxxx, r255.xxxx mov r255, r1022.xyz0 mov r255. r255.x000 ishl r255.x\_\_\_, r255.xxxx, r256.xxxx iadd r254.x\_\_\_, r1.xxxx, r255.xxxx mov r1010.x , r254.xxxx uav raw load id(11) cached r1011.x , r1010.xxxx mov r254.x . r1011.xxxx <u>mov r1011.x</u>, r254.xxxx mov r1010.x . r253.xxxx uav raw store id(11) mem.x , r1010.xxxx, r1011.xxxx ret

mov r255, r1021.xyz0 mov r255. r255.x000 mov r256. 19.xxxx ishl r255.x\_\_\_, r255.xxxx, r256.xxxx iadd r253.x , r2.xxxx, r255.xxxx mov r255, r1022.xyz0 mov r255. r255.x000 ishl r255.x\_\_\_, r255.xxxx, r256.xxxx iadd r254.x\_\_\_, r1.xxxx, r255.xxxx mov r1010.x , r254.xxxx uav raw load id(11) cached r1011.x , r1010.xxxx mov r254.x . r1011.xxxx mov r1011.x . r254.xxxx mov r1010.x . r253.xxxx uav raw store id(11) mem.x , r1010.xxxx, r1011.xxxx ret

mov r255, r1021.xyz0 mov r255. r255.x000 mov r256. 19.xxxx ishl r255.x , r255.xxxx, r256.xxxx iadd r253.x , r2.xxxx, r255.xxxx mov r255, r1022.xyz0 mov r255. r255.x000 ishl r255.x\_\_\_, r255.xxxx, r256.xxxx iadd r254.x\_\_\_, r1.xxxx, r255.xxxx mov r1010.x , r254.xxxx uav raw load id(11) cached r1011.x , r1010.xxxx mov r254.x\_\_\_, r1011.xxxx mov r1011.x . r254.xxxx mov r1010.x . r253.xxxx uav raw store id(11) mem.x , r1010.xxxx, r1011.xxxx ret



• The GPU hardware of course does not execute those work items as threads

#### • The reality is that high-end GPUs follow a SIMD architecture

- Each work item describes a lane of execution
- Multiple work items execute together in SIMD fashion with a single program counter
- Some clever automated stack management to handle divergent control flow across the vector

mov r255, r1021.xyz0	mov r255, r1021.xyz0	mov r255, r1021.xyz0
mov r255, r255.x000	mov r255, r255.x000	mov r255, r255.x000
mov r256, I9.xxxx	mov r256, 19.xxxx	mov r256, I9.xxxx
ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx
iadd r253.x, r2.xxxx, r255.xxxx	iadd r253.x, r2.xxxx, r255.xxxx	iadd r253.x, r2.xxxx, r255.xxxx
mov r255, r1022.xyz0	mov r255, r1022.xyz0	mov r255, r1022.xyz0
mov r255, r255.x000	mov r255, r255.x000	mov r255, r255.x000
ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx
iadd r254.x, r1.xxxx, r255.xxxx	iadd r254.x, r1.xxxx, r255.xxxx	iadd r254.x, r1.xxxx, r255.xxxx
mov r1010.x, r254.xxxx	mov r1010.x, r254.xxxx	mov r1010.x, r254.xxxx
uav_raw_load_id(11)_cached r1011.x, r1010.xxxx	uav_raw_load_id(11)_cached r1011.x, r1010.xxxx	uav_raw_load_id(11)_cached r1011.x, r1010.xxxx
mov r254.x, r1011.xxxx	mov r254.x, r1011.xxxx	mov r254.x, r1011.xxxx
mov r1011.x, r254.xxxx	mov r1011.x, r254.xxxx	mov r1011.x, r254.xxxx
mov r1010.x, r253.xxxx	mov r1010.x, r253.xxxx	mov r1010.x, r253.xxxx
uav_raw_store_id(11) mem.x, r1010.xxxx, r1011.xxxx	uav_raw_store_id(11)	uav_raw_store_id(11) mem.x, r1010.xxxx, r1011.xxxx
ret	ret	ret

• The GPU hardware of course does not execute those work items as threads

#### • The reality is that high-end GPUs follow a SIMD architecture

- Each work item describes a lane of execution
- Multiple work items execute together in SIMD fashion with a single program counter
- Some clever automated stack management to handle divergent control flow across the vector

mov r255 r1021 vvz0	mov r255 r1021 vvz0	mov r255 r1021 vvz0
mov r255, r255.x000	mov r255, r255.x000	mov r255, r255.x000
mov r256, 19.xxxx	mov r256, 19.xxxx	MOV 1256, 19.XXXX
ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx
iadd r253.x, r2.xxxx, r255.xxxx	iadd r253.x, r2.xxxx, r255.xxxx	iadd r253.x, r2.xxxx, r255.xxxx
mov r255, r1022.xyz0	mov r255, r1022.xyz0	mov r255, r1022.xyz0
mov r255, r255.x000	mov r255, r255.x000	mov r255, r255.x000
ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx
iadd r254.x, r1.xxxx, r255.xxxx	iadd r254.x, r1.xxxx, r255.xxxx	iadd r254.x, r1.xxxx, r255.xxxx
mov r1010.x, r254.xxxx	mov r1010.x, r254.xxxx	mov r1010.x, r254.xxxx
uav_raw_load_id(11)_cached r1011.x, r1010.xxxx	uav_raw_load_id(11)_cached r1011.x, r1010.xxxx	uav_raw_load_id(11)_cached r1011.x, r1010.xxxx
mov r254.x, r1011.xxxx	mov r254.x, r1011.xxxx	mov r254.x, r1011.xxxx
mov r1011.x, r254.xxxx	mov r1011.x, r254.xxxx	mov r1011.x, r254.xxxx
mov r1010.x, r253.xxxx	mov r1010.x, r253.xxxx	mov r1010.x, r253.xxxx
uav_raw_store_id(11) mem.x, r1010.xxxx, r1011.xxxx	uav_raw_store_id(11)	uav_raw_store_id(11)
ret	ret	ret

8 | IEEE Workshop: Data Parallelism for Multi-Core Chips and GPU | 27 October 2012 | Public

• The GPU hardware of course does not execute those work items as threads

The reality is that high-end GPUs follow a SIMD architecture

- Each work item describes a lane of execution
- Multiple work items execute together in SIMD fashion with a single program counter
- Some clever automated stack management to handle divergent control flow across the vector

mov r255, r1021.xyz0	mov r255, r1021.xyz0	mov r255, r1021.xyz0
mov r256, I9.xxxx	mov r256, I9.xxxx	mov r256, 19.xxxx
ISNI 1255.X, 1255.XXXX, 1256.XXXX	ISNI 1255.X, 1255.XXXX, 1256.XXXX	ISNI 1255.X, 1255.XXXX, 1256.XXXX
iadd r253.x, r2.xxxx, r255.xxxx	iadd r253.x, r2.xxxx, r255.xxxx	iadd r253.x, r2.xxxx, r255.xxxx
mov r255, r1022.xyz0	mov r255, r1022.xyz0	mov r255, r1022.xyz0
mov r255, r255.x000	mov r255, r255.x000	mov r255, r255.x000
ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx
iadd r254.x, r1.xxxx, r255.xxxx	iadd r254.x, r1.xxxx, r255.xxxx	iadd r254.x, r1.xxxx, r255.xxxx
mov r1010.x, r254.xxxx	mov r1010.x, r254.xxxx	mov r1010.x, r254.xxxx
uav_raw_load_id(11)_cached r1011.x, r1010.xxxx	uav_raw_load_id(11)_cached r1011.x, r1010.xxxx	uav_raw_load_id(11)_cached r1011.x, r1010.xxxx
mov r254.x, r1011.xxxx	mov r254.x, r1011.xxxx	mov r254.x, r1011.xxxx
mov r1011.x, r254.xxxx	mov r1011.x, r254.xxxx	mov r1011.x, r254.xxxx
mov r1010.x, r253.xxxx	mov r1010.x, r253.xxxx	mov r1010.x, r253.xxxx
uav_raw_store_id(11) mem.x, r1010.xxxx, r1011.xxxx	uav_raw_store_id(11)	uav_raw_store_id(11) mem.x, r1010.xxxx, <u>r1011.xxxx</u>
ret	ret	ret

• The GPU hardware of course does not execute those work items as threads

The reality is that high-end GPUs follow a SIMD architecture

- Each work item describes a lane of execution
- Multiple work items execute together in SIMD fashion with a single program counter
- Some clever automated stack management to handle divergent control flow across the vector

mov r255, r1021.xyz0	mov r255, r1021.xyz0	mov r255, r1021.xyz0
mov r255, r255.x000	mov r255, r255.x000	mov r255, r255.x000
mov r256_19 xxxx	mov r256_19 xxxx	mov r256_19 xxxx
ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx
iauu 1200.x, 12.xxxx, 1200.xxxx	adu 1203.x, 12.xxxx, 1203.xxxx	lauu 1200.x, 12.xxxx, 1200.xxxx
mov r255, r1022.xyz0	mov r255, r1022.xyz0	mov r255, r1022.xyz0
mov r255, r255.x000	mov r255, r255.x000	mov r255, r255.x000
ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx	ishl r255.x, r255.xxxx, r256.xxxx
iadd r254.x, r1.xxxx, r255.xxxx	iadd r254.x, r1.xxxx, r255.xxxx	iadd r254.x, r1.xxxx, r255.xxxx
mov r1010.x, r254.xxxx	mov r1010.x, r254.xxxx	mov r1010.x, r254.xxxx
uav_raw_load_id(11)_cached r1011.x, r1010.xxxx	uav_raw_load_id(11)_cached r1011.x, r1010.xxxx	uav_raw_load_id(11)_cached r1011.x, r1010.xxxx
mov r254.x, r1011.xxxx	mov r254.x, r1011.xxxx	mov r254.x, r1011.xxxx
mov r1011.x, r254.xxxx	mov r1011.x, r254.xxxx	mov r1011.x, r254.xxxx
mov r1010.x, r253.xxxx	mov r1010.x, r253.xxxx	mov r1010.x, r253.xxxx
uav_raw_store_id(11) mem.x, r1010.xxxx, r1011.xxxx	uav_raw_store_id(11)	uav_raw_store_id(11)
ret	ret	ret



10 | IEEE Workshop: Data Parallelism for Multi-Core Chips and GPU | 27 October 2012 | Public

# IT WAS NEVER QUITE THAT SIMPLE

The HD6970 architecture and its predecessors were combined multicore SIMD/VLIW machines

- Data-parallel through hardware vectorization
- Instruction parallel through both multiple cores and VLIW units
- The HD6970 issued a 4-way VLIW instruction per work item
  - Architecturally you could view that as a 4-way VLIW instruction issue per SIMD lane
  - Alternatively you could view it as a 4-way VLIW issue of SIMD instructions



• The IL we saw earlier ends up compiling to something like this:

: ----- Disassembly -----00 ALU: ADDR(32) CNT(9) KCACHE0(CB1:0-15) KCACHE1(CB0:0-15) 0 w: LSHL , R0.x, 2 1 z: ADD\_INT \_\_\_\_, KC0[0].x, PV0.w 2 v: LSHR R0.v, PV1.z, 2 3 x: MULLO\_INT R1.x, R1.x, KC1[1].x y: MULLO\_INT \_\_\_\_, R1.x, KC1[1].x z: MULLO\_INT \_\_\_\_, R1.x, KC1[1].x w: MULLO\_INT \_\_\_\_, R1.x, KC1[1].x 01 TEX: ADDR(48) CNT(1) 4 VFETCH R2.x , R0.y, fc153 FETCH\_TYPE(NO\_INDEX\_OFFSET) 02 ALU: ADDR(41) CNT(7) KCACHE0(CB0:0-15) KCACHE1(CB1:0-15) 5 w: ADD\_INT \_\_\_\_, R0.x, R1.x 6 z: ADD INT \_\_\_\_, PV5.w, KC0[6].x 7 y: LSHL , PV6.z, 2 8 x: ADD\_INT \_\_\_\_, KC1[1].x, PV7.y 9 x: LSHR R0.x, PV8.x, 2 03 MEM\_RAT\_CACHELESS\_STORE\_DWORD\_\_NI: RAT(11)[R0].x\_\_\_, R2, ARRAY\_SIZE(4) MARK\_VPM 04 END END OF PROGRAM



The IL we saw earlier ends up compiling to something like this:

· ----- Disassembly -----00 ALU: ADDR(32) CNT(9) KCACHE0(CB1:0-15) KCACHE1(CB0:0-15) <u>0 w: LSHL</u> , R0.x, 2 1 z: ADD\_INT \_\_\_\_, KC0[0].x, PV0.w 2 y: LSHR R0.y, PV1.z, 2 3 x: MULLO\_INT R1.x, R1.x, KC1[1].x y: MULLO\_INT \_\_\_\_, R1.x, KC1[1].x z: MULLO\_INT \_\_\_\_, R1.x, KC1[1].x w: MULLO\_INT \_\_\_\_, R1.x, KC1[1].x 01 TEX: ADDR(48) CNT(1) 4 VFETCH R2.x , R0.y, fc153 FETCH\_TYPE(NO\_INDEX\_OFFSET) 02 ALU: ADDR(41) CNT(7) KCACHE0(CB0:0-15) KCACHE1(CB1:0-15) 5 w: ADD\_INT \_\_\_\_, R0.x, R1.x 6 z: ADD\_INT \_\_\_\_, PV5.w, KC0[6].x 7 y: LSHL , PV6.z, 2 8 x: ADD\_INT \_\_\_\_, KC1[1].x, PV7.y 9 x: LSHR R0.x, PV8.x, 2 03 MEM RAT CACHELESS STORE DWORD NI: RAT(11)[R0].x , R2, ARRAY SIZE(4) MARK VPM 04 END END OF PROGRAM

Clause header Work executed by the shared scalar unit

The IL we saw earlier ends up compiling to something like this:

Clause header · ----- Disassembly -----Work executed by the 00 ALU: ADDR(32) CNT(9) KCACHE0(CB1:0-15) KCACHE1(CB0:0-15) shared scalar unit 0 w: LSHL , R0.x, 2 1 z: ADD\_INT \_\_\_\_, KC0[0].x, PV0.w 2 y: LSHR R0.y, PV1.z, 2 3 x: MULLO\_INT R1.x, R1.x, KC1[1].x y: MULLO\_INT \_\_\_\_, R1.x, KC1[1].x z: MULLO\_INT \_\_\_\_, R1.x, KC1[1].x w: MULLO INT , R1.x, KC1[1].x 01 TEX: ADDR(48) CNT(1) 4 VFETCH R2.x , R0.y, fc153 FETCH TYPE(NO INDEX OFFSET) Clause body 02 ALU: ADDR(41) CNT(7) KCACHE0(CB0:0-15) KCACHE1(CB1:0-15) Units of work dispatched 5 w: ADD\_INT \_\_\_\_, R0.x, R1.x by the shared scalar unit 6 z: ADD\_INT \_\_\_\_, PV5.w, KC0[6].x 7 y: LSHL , PV6.z, 2 8 x: ADD\_INT \_\_\_\_, KC1[1].x, PV7.y 9 x: LSHR R0.x, PV8.x, 2 03 MEM\_RAT\_CACHELESS\_STORE\_DWORD\_NI: RAT(11)[R0].x\_\_\_, R2, ARRAY\_SIZE(4) MARK\_VPM 04 END END OF PROGRAM

• The IL we saw earlier ends up compiling to something like this:

· Disassembly	Clause fieadel
00 ALU: ADDR(32) CNT(9) KCACHE0(CB1:0-15) KCACHE1(CB0:0-15)	Work executed by
0 w: LSHL, R0.x, 2	shared scalar unit
1 z: ADD_INT, KC0[0].x, PV0.w	
2 y: LSHR R0.y, PV1.z, 2	VLIW instruction
3 x: MULLO_INT R1.x, R1.x, KC1[1].x	Compiler-generate
y: MULLO_INT, R1.x, KC1[1].x	Complier-generate
z: MULLO_INT, R1.x, KC1[1].x	parallelism for the
w: MULLO_INT, R1.x, KC1[1].x	Each instruction (x
01 TEX: ADDR(48) CNT(1)	across the vector
4 VFETCH R2.x, R0.y, fc153	
FETCH_TYPE(NO_INDEX_OFFSET)	Clause bodv
02 ALU: ADDR(41) CNT(7) KCACHE0(CB0:0-15) KCACHE1(CB1:0-15)	Units of work dispa
5 w: ADD_INT, R0.x, R1.x	
6 z: ADD_INT, PV5.w, KC0[6].x	by the shared scal
7 y: LSHL, PV6.z, 2	
8 x: ADD_INT, KC1[1].x, PV7.y	
9 x: LSHR R0.x, PV8.x, 2	
03 MEM_RAT_CACHELESS_STORE_DWORDNI: RAT(11)[R0].x, R2	, ARRAY_SIZE(4) MARK VPM
04 END	
END OF PROGRAM	

Clause header ork executed by the ared scalar unit

#### IW instruction packet

mpiler-generated instruction level allelism for the VLIW unit. ch instruction (x, y, z, w) executed oss the vector.

#### ause body

its of work dispatched the shared scalar unit

• The IL we saw earlier ends up compiling to something like this:



Clause header Work executed by the shared scalar unit

#### VLIW instruction packet

Compiler-generated instruction level parallelism for the VLIW unit. across the vector.

#### Clause body Units of work dispatched by the shared scalar unit

03 MEM\_RAT\_CACHELESS\_STORE\_DWORD\_\_NI: RAT(11)[R0].x\_\_\_, R2, ARRAY\_SIZE(4) MARK\_VPM END OF PROGRAM

Notice the poor occupancy of VLIW slots



### WHY DID WE SEE INEFFICIENCY?

The architecture was well suited to graphics workloads:

- VLIW was easily filled by the vector-heavy graphics kernels
- Minimal control flow meant that the monolithic, shared thread scheduler was relatively efficient

• Unfortunately, workloads change with time.

So how did we change the architecture to improve the situation?

Brand new – but at this level it doesn't look too different



Brand new – but at this level it doesn't look too different

- Two command processors
  - Capable of processing two command queues concurrently

	AMD Radeon HD7970							
Asy	/nchronous Compute Engine / Command Processor	Asynchronous Compute Engine / Command Processor						
SC cache SC cache I cache I cache	SC       SIMD Core       L1       LDS	LDS L1 SIMD Core SC LDS L1 SIMD Core SC						
SC cache SC cache I cache I cache	SC       SIMD Core       L1       LDS         SC       SIMD Core       L1       LDS	LDS L1 SIMD Core SC LDS L1 SIMD Core SC						
	Level 2 cache GDDR5 Memory System							

Brand new – but at this level it doesn't look too different

- Two command processors
  - Capable of processing two command queues concurrently
- Full read/write L1 data caches
- SIMD cores grouped in fours
  - Scalar data and instruction cache per cluster
  - L1, LDS and scalar processor per core

• Up to 32 cores / compute units

	AMD Radeon HD7970																
	Asynchronous Compute Engine							A	synch	nron	ous Compute	e Eng	ine				
		/0	ommand Pro	cess	or				/(	Com	mand Proces	sor					
		sc	SIMD Coro	11				[		11	SIMD Coro	sc					
che	ЭС	SC	SIMD Core	11						11	SIMD Core	SC	-	SC			
cad	acl	SC	SIMD Core	11						11	SIMD Core	SC	act	cac			
SC	-	SC	SIMD Core	11			e				SIMD Core	SC	le	he			
		SC	SIMD Core	11			fac			11	SIMD Core	SC		$\exists$			
che	Ъ	SC	SIMD Core	11			Iter			11	SIMD Core	SC	-	SC			
cad	acl	SC	SIMD Core	11			ni yror			11	SIMD Core	SC	act	cac			
SC	-	SC	SIMD Core	11				nor			11	SIMD Core	SC	le	he		
		SC	SIMD Core				mei			11	SIMD Core	SC					
he	e	SC	SIMD Core				E T				SIMD Core	SC	-	SC			
cac	ach	SC	SIMD Core	11			/Lit			11	SIMD Core	SC	act	cac			
SC	-	SC	SIMD Core				X	2				SIMD Core	SC	le	he		
		SC	SIMD Core				eac				SIMD Core	SC		$\square$			
he	e	SC	SIMD Core			3	æ	æ	8	~			11	SIMD Core	SC	-	SC
cac	ach	SC	SIMD Core	11						11	SIMD Core	SC	ach	cac			
SC	-	SC	SIMD Core	11						11	SIMD Core	SC	le	he			
		30	SIND COLE		103				LD3		SIMD COLE	30					
	level 2 cache																
				G	DDR5	Me	mo	rv	Syste	m							
				0	DUNJ	IVIC		' Y	Syste								

Brand new – but at this level it doesn't look too different

- Two command processors
  - Capable of processing two command queues concurrently
- Full read/write L1 data caches
- SIMD cores grouped in fours
  - Scalar data and instruction cache per cluster
  - L1, LDS and scalar processor per core

• Up to 32 cores / compute units

	AMD Radeon HD7970																
	Asynchronous Compute Engine							A	synch	nron	ous Compute	e Eng	ine				
		/ C	ommand Pro	cesso	or				/ (	Com	mand Proces	sor					
	SC SIMD Core 11 LDS					1		11	SIMD Core	SC							
che	Je	SC	SIMD Core	11	LDS				LDS	11	SIMD Core	SC	-0	SC			
cac	cacl	SC	SIMD Core	11					LDS	11	SIMD Core	SC	ach	cac			
SC	-	SC	SIMD Core	e L1 LDS		LDS	11	SIMD Core	SC	le	he						
		SC	SIMD Core	11			fac			11	SIMD Core	SC					
he	e	SC	SIMD Core	11	LDS		ter		LDS	11	SIMD Core	SC	-	SC			
cac	ach	SC	SIMD Core		LDS		emory int	ory in		LDS		SIMD Core	SC	ac	cac		
SC	-	SC	SIMD Core	11	LDS				Lo	(noi		LDS		SIMD Core	SC	ne	che
		SC	SIMD Core	LI	LDS				LDS	LI	SIMD Core	SC					
he	e	SC	SIMD Core	LI	LDS		E		LDS	LI	SIMD Core	SC	-	SC			
cac	ach	SC	SIMD Core	L1	LDS		ad/Write	ad/Write		LDS	L1	SIMD Core	SC	cac	Ca		
S		SC	SIMD Core	L1	LDS				ad/W	3		LDS	L1	SIMD Core	SC	he	che
0,		SC	SIMD Core	L1	LDS						LDS	L1	SIMD Core	SC			
e	0.	SC	SIMD Core	L1	LDS		Re		LDS	L1	SIMD Core	SC	_	S			
ach	che	SC	SIMD Core	L1	LDS					LDS	L1	SIMD Core	SC	ca	CC		
CC	Ca	SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC	che	ach			
S		SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC		P			
	Level 2 cache																
				GE	DDR5	Me	emo	ry	Syste	m							

The SIMD unit on the HD6970 architecture had a branch control but full scalar execution was performed globally



 The SIMD unit on the HD6970 architecture had a branch control but full scalar execution was performed globally



• On the HD7970 we have a full scalar processor and the L1 cache and LDS have been doubled in size





- On the HD7970 we have a full scalar processor and the L1 cache and LDS have been doubled in size
- Then let us consider the VLIW ALUs





- Remember we could view the architecture two ways:
  - An array of VLIW units





- Remember we could view the architecture two ways:
  - An array of VLIW units
  - A VLIW cluster of vector units





- Now that we have a scalar processor we can dynamically schedule instructions rather than relying on the compiler
- No VLIW!



- The heart of Graphics Core Next:
  - A scalar processor with four 16-wide vector units
  - Each lane of the vector, and hence each IL work item, is now scalar

Simpler and more efficient

Instructions for both sets of execution units // inline v cmp gt f32

S а S v s v m

s m

- No VLIW
  - Fewer compiler-induced bubbles in the instruction schedule
- Full support for exceptions, function calls and recursion

<pre>float fn0(float a,float b) {</pre>
if(a>b)
return((a-b)*a);
else
return(( $b-a$ )*b);
}

/Registers	r0	contains	"a″,	r1	contains	"b"
			•			

v_cmp_gt_f32	r0,r1	//a > b, establish VCC
s_mov_b64	s0,exec	//Save current exec mask
s_and_b64	exec,vcc,exec	c //Do "if"
s_cbranch_vccz	label0	//Branch if all lanes fail
v_sub_f32	r2,r0,r1	//result = a - b
v_mul_f32	r2,r2,r0	<pre>//result=result * a</pre>

ndn2_b64	<pre>exec,s0,exec</pre>	//Do "else"(s0 & !exec)
oranch_execz	label1	//Branch if all lanes fail
ub_f32	r2,r1,r0	//result = b - a
ul_f32	r2,r2,r1	<pre>//result = result * b</pre>
ov b64	exec,s0	//Restore exec mask



Simpler and more efficient

Instructions for both sets of execution units //value is returned in r2 inline
u cmp ct f32 = r0 r1

- No VLIW
  - Fewer compiler-induced bubbles in the instruction schedule
- Full support for exceptions, function calls and recursion

<pre>float fn0(float a,float b) {</pre>
if(a>b)
return((a-b)*a);
else
return((b-a)*b);
}

	v_cmp_gt_f32	r0,r1	//a > b, establish VCC
	s_mov_b64	sU,exec	//Save current exec mask
n the	s_and_b64	exec,vcc,exec	//Do "if"
	s_cbranch_vccz	label0	//Branch if all lanes fail
	v_sub_f32	r2,r0,r1	//result = a - b
alls	v_mul_f32	r2,r2,r0	//result=result * a
	:		
	s_andn2_b64	exec,s0,exec	//Do "else"(s0 & !exec)
	s_andn2_b64 s_cbranch_execz	<pre>exec,s0,exec label1</pre>	<pre>//Do ``else"(s0 &amp; !exec) //Branch if all lanes fail</pre>
	<pre>s_andn2_b64 s_cbranch_execz v_sub_f32</pre>	<pre>exec,s0,exec label1 r2,r1,r0</pre>	<pre>//Do "else"(s0 &amp; !exec) //Branch if all lanes fail //result = b - a</pre>
I	<pre>s_andn2_b64 s_cbranch_execz v_sub_f32 v_mul_f32</pre>	<pre>exec,s0,exec label1 r2,r1,r0 r2,r2,r1</pre>	<pre>//Do "else"(s0 &amp; !exec) //Branch if all lanes fail //result = b - a //result = result * b</pre>
l	<pre>s_andn2_b64 s_cbranch_execz v_sub_f32 v_mul_f32</pre>	<pre>exec,s0,exec label1 r2,r1,r0 r2,r2,r1</pre>	<pre>//Do "else"(s0 &amp; !exec) //Branch if all lanes fail //result = b - a //result = result * b</pre>
l	<pre>s_andn2_b64 s_cbranch_execz v_sub_f32 v_mul_f32 s_mov_b64</pre>	<pre>exec,s0,exec label1 r2,r1,r0 r2,r2,r1 exec,s0</pre>	<pre>//Do "else"(s0 &amp; !exec) //Branch if all lanes fail //result = b - a //result = result * b //Restore exec mask</pre>
l	<pre>s_andn2_b64 s_cbranch_execz v_sub_f32 v_mul_f32 s_mov_b64</pre>	<pre>exec,s0,exec label1 r2,r1,r0 r2,r2,r1 exec,s0</pre>	<pre>//Do "else"(s0 &amp; !exec) //Branch if all lanes fail //result = b - a //result = result * b //Restore exec mask</pre>
l	<pre>s_andn2_b64 s_cbranch_execz v_sub_f32 v_mul_f32 s_mov_b64</pre>	<pre>exec,s0,exec label1 r2,r1,r0 r2,r2,r1 exec,s0</pre>	<pre>//Do "else"(s0 &amp; !exec) //Branch if all lanes fail //result = b - a //result = result * b //Restore exec mask</pre>

AMD

//Registers r0 contains "a", r1 contains "b"

Simpler and more efficient

 Instructions for both sets of execution units //value is returned in r2 inline

> s\_andn2\_ s\_cbranc v\_sub\_f3 v mul f3

s mov be

- No VLIW
  - Fewer compiler-induced bubbles in the instruction schedule
- Full support for exceptions, function calls and recursion

<pre>float fn0(float a,float b) {</pre>	)
if(a>b)	
return((a-b)*a);	
else	
return((b-a)*b);	
}	

//value is returned in r2			
v cmp gt f32	r0,r1	//a > b, establish VCC	
s_mov_b64	s0,exec	//Save current exec mask	
s_and_b64	exec,vcc,exec	//Do "if"	
s_cbranch_vccz	label0	//Branch if all lanes fai	
v_sub_f32	r2,r0,r1	//result = a - b	
v_mul_f32	r2,r2,r0	<pre>//result=result * a</pre>	

//Registers r0 contains "a", r1 contains "b"

b64 h_execz 2 2	<pre>exec,s0,exec label1 r2,r1,r0 r2,r2,r1</pre>	<pre>//Do "else"(s0 &amp; !exec) //Branch if all lanes fail //result = b - a //result = result * b</pre>
4	exec,s0	//Restore exec mask



31 | IEEE Workshop: Data Parallelism for Multi-Core Chips and GPU | 27 October 2012 | Public

Simpler and more efficient

Instructions for both sets of execution units //value is returned in r2 inline v cmp gt f32

- No VLIW
  - Fewer compiler-induced bubbles in the instruction schedule
- Full support for exceptions, function calls and recursion

//Registers r0 contains "a	", r1 contains "b"
----------------------------	--------------------

v_cmp_gt_f32	r0,r1	//a > b, establish VCC
s_mov_b64	s0,exec	//Save current exec mask
s_and_b64	exec,vcc,exe	c //Do "if"
s_cbranch_vccz	label0	//Branch if all lanes fail
v_sub_f32	r2,r0,r1	//result = a - b
v_mul_f32	r2,r2,r0	<pre>//result=result * a</pre>

float fn0(float a,float b)	s_andn2_b64	exec,s0,exec	<pre>//Do "else"(s0 &amp; !exec) //Branch if all lanes fail</pre>
{ if(a>b)	v_sub_f32 v_mul_f32	r2,r1,r0 r2,r2,r1	<pre>//result = b - a //result = result * b</pre>
return((a-b)*a); else	s_mov_b64	exec,s0	//Restore exec mask
<b>return((b-a)*b);</b> }			

# FAMILIAR?

If we add the frontend of the core...





33 | IEEE Workshop: Data Parallelism for Multi-Core Chips and GPU | 27 October 2012 | Public

# FAMILIAR?



34 | IEEE Workshop: Data Parallelism for Multi-Core Chips and GPU | 27 October 2012 | Public

# FAMILIAR?



# SHARED VIRTUAL MEMORY




#### TAKING THE MEMORY SYSTEM CHANGES FURTHER AFIELD

- GCN-based devices are more flexible
  - Start to look like CPUs with few obvious shortcomings
- The read/write cache is a good start at improving the memory system
  - Improves efficiency on the device
  - Provides a buffer for imperfectly written code
- We needed to go a step further on an SoC
  - Memory in those caches should be the same memory used by the "host" CPU
  - In the long run, the CPU and GPU become peers, rather than having a host/slave relationship







We can store x86 virtual pointers here





Data stored here is addressed in the same way as that on the CPU





#### **USE CASES FOR THIS ARE FAIRLY OBVIOUS**

Pointer chasing algorithms with mixed GPU/CPU use

Algorithms that construct data on the CPU, use it on the GPU

Allows for more fine-grained data use without explicit copies

Covers cases where explicit copies are difficult:

- Picture OS allocated data that the OpenCL runtime doesn't know about

However, that wasn't quite enough to achieve our goals...



• We need a global view of the GPU, not just of the shader cores

	AMD Radeon HD7970													
	Asy	/nchi	ronous Comp	oute	Engin	е		A	synchror	ous Comput	e Eng	ine		
	/ Command Processor							/ Command Processor						
		66		14		a r				CINAD Com				
he	e	SC	SIMD Core	LI	LDS				LDS LI	SIMD Core	SC	-	SC	
cac	ach	SC	SIMD Core	L1	LDS		0)		LDS L1	SIMD Core	SC	cac	Ca	
S	- C	SC	SIMD Core	L1	LDS				LDS L1	SIMD Core	SC	he	che	
0,		SC	SIMD Core	L1	LDS		ace		LDS L1	SIMD Core	SC			
e	a	SC	SIMD Core	L1	LDS		Write memory interf		LDS L1	SIMD Core	SC	_	S	
ach	ç	SC	SIMD Core	L1	LDS				LDS L1	SIMD Core	SC	cache I cache	C	
S	Ca	SC	SIMD Core	L1	LDS				LDS L1	SIMD Core	SC		ich	
S		SC	SIMD Core	L1	LDS				LDS L1	SIMD Core	SC		P	
e		SC	SIMD Core	L1	LDS				LDS L1	SIMD Core	SC		S	
ach	che	SC	SIMD Core	L1	LDS				LDS L1	SIMD Core	SC		Cci	
U U	ca	SC	SIMD Core	L1	LDS				LDS L1	SIMD Core	SC		ach	
Š		SC	SIMD Core	L1	LDS		/pe		LDS L1	SIMD Core	SC		ē	
e		SC	SIMD Core	L1	LDS	1	Rea		LDS L1	SIMD Core	SC		S	
ach	che	SC	SIMD Core	L1	LDS	1			LDS L1	SIMD Core	SC	Са	Cc	
<sup>3</sup>	ca	SC	SIMD Core	L1	LDS				LDS L1	SIMD Core	SC	che	ach	
S		SC	SIMD Core	L1	LDS				LDS L1	SIMD Core	SC	10	Ie	
					_	2 6				- 16- 	10			
					Le	ve	2 ca	acł	ne					
				G	DDR5	Me	emo	ry	System					
L														

• We need a global view of the GPU, not just of the shader cores

Of course, we need to see the data here too

	AMD Radeon HD7970												
As	ynchronous Compute Engine		Asynchronous Compute Engine										
	/ Command Processor		/ Command Processor										
	SC SIMD Care 11 LDS												
he	SC SIMD Core L1 LDS		LDS L1 SIMD Core SC _ S										
cac	SC SIMD Core L1 LDS		LDS L1 SIMD Core SC G										
SC	SC SIMD Core L1 LDS	a	LDS L1 SIMD Core SC B										
	SC SIMD Core LI LDS	face	LDS L1 SIMD Core SC										
e le	SC SIMD Core L1 LDS	tert	LDS L1 SIMD Core SC _ G										
cacl	SC SIMD Core L1 LDS	in	LDS L1 SIMD Core SC C C										
<u> </u>	SC SIMD Core L1 LDS	ory	LDS L1 SIMD Core SC 핥 다										
0,	SC SIMD Core L1 LDS	em	LDS L1 SIMD Core SC										
و م	SC SIMD Core L1 LDS	Ĕ	LDS L1 SIMD Core SC _ v										
act act	SC SIMD Core L1 LDS	ite	LDS L1 SIMD Core SC S										
CC	SC SIMD Core L1 LDS	Ň	LDS L1 SIMD Core SC 물 드										
S	SC SIMD Core L1 LDS	ad/	LDS L1 SIMD Core SC										
e a	SC SIMD Core L1 LDS	Re	LDS L1 SIMD Core SC _ v										
ach	SC SIMD Core L1 LDS		LDS L1 SIMD Core SC 🔒 🔒										
CC	SC SIMD Core L1 LDS		LDS L1 SIMD Core SC 물 음										
S –	SC SIMD Core L1 LDS		LDS L1 SIMD Core SC										
	Leve	el 2 c	ache										
	GDDR5 M	lemo	bry System										



• We need a global view of the GPU, not just of the shader cores

Most importantly! We need to be able to compute on the same data here.

Of course, we need to see the data here too

AMD Radeon HD7970														
Asynchronous Compute Engine / Command Processor								Asynchronous Compute Engine / Command Processor						
SC cache SC cache	I cache I cache	SC SC SC SC SC SC SC SC	SIMD Core SIMD Core SIMD Core SIMD Core SIMD Core SIMD Core SIMD Core SIMD Core	L1 L1 L1 L1 L1 L1 L1 L1 L1 L1	LDS LDS LDS LDS LDS LDS LDS LDS LDS		nemory interface	LDS L1 SIMD Core SC LDS L1 SIMD Core SC						
SC cache	I cache	SC SC SC	SIMD Core SIMD Core SIMD Core	L1 L1 L1	LDS LDS LDS		d/Write r	LDS L1 SIMD Core SC LDS L1 SIMD Core SC LDS L1 SIMD Core SC						
SC cache	I cache	SC SC SC SC	SIMD Core SIMD Core SIMD Core SIMD Core	L1 L1 L1 L1	LDS LDS LDS LDS		Rea	LDS L1 SIMD Core SC LDS L1 SIMD Core SC LDS L1 SIMD Core SC LDS L1 SIMD Core SC						
					Le	vel	2 ca	ache						
				GE	DDR5	Me	emo	ry System						

same data here.



AMD

We need a global view of the GPU, not just of the shader cores

## TODAY'S COMMAND AND DISPATCH FLOW





50 | IEEE Workshop: Data Parallelism for Multi-Core Chips and GPU | 27 October 2012 | Public





AMD



Queues are in user-mode virtual memory

AMD





- Application codes to the hardware
- User mode queuing
- Hardware scheduling
- Low dispatch times

- No required APIs
- No Soft Queues
- No User Mode Drivers
- No Kernel Mode Transitions

AMD

No Overhead!



# ARCHITECTED ACCESS



## HETEROGENEOUS SYSTEM ARCHITECTURE – AN OPEN PLATFORM

- Open Architecture, published specifications
  - HSAIL virtual ISA
  - HSA memory model
  - Architected Queuing Language
- HSA system architecture
  - Inviting partners to join us, in all areas
  - Hardware companies
  - Operating Systems
  - Tools and Middleware
  - Applications
- HSA Foundation being formed



AMD

## ARCHITECTED INTERFACES

- Standardize interfaces to features of the system
  - The compute cores
  - The memory hierarchy
  - Work dispatch
- Standardize access to the device
  - Memory backed queues
  - User space data structures

				1	AMD Radeon HD7970												
	Asy	/nch	ronous Comp	oute	Engin	e		Asynchronous Compute Engine									
		/ C	ommand Pro	cess	or			/ Command Processor									
										1.1							
he	e	SC	SIMD Core	LI	LDS				LDS	LI	SIMD Core	SC	-	SC			
ac	ach	SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC	cac	ca			
S	0	SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC	he	ch			
S		SC	SIMD Core	L1	LDS		ace		LDS	L1	SIMD Core	SC		D			
e	2	SC	SIMD Core	L1	LDS		erf		LDS	L1	SIMD Core	SC		S			
ach	che	SC	SIMD Core	L1	LDS		int		LDS	L1	SIMD Core	SC	Ca	Cci			
C	Ca	SC	SIMD Core	L1	LDS		2		LDS	L1	SIMD Core	SC	che	ach			
S		SC	SIMD Core	L1	LDS		om		LDS	L1	SIMD Core	SC	10	ē			
e		SC	SIMD Core	L1	LDS		me		LDS	L1	SIMD Core	SC		S			
ach	che	SC	SIMD Core	L1	LDS		Write		LDS	L1	SIMD Core	SC	cache	Cc			
C C	ca	SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC		ach			
S		SC	SIMD Core	L1	LDS		/pe		LDS	L1	SIMD Core	SC		le			
a		SC	SIMD Core	L1	LDS		Rea		LDS	L1	SIMD Core	SC		S			
ach	che	SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC	l ca	Co			
C	cag	SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC	ich	ach			
SC	-	SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC	CD	le			
	_	_		_	_	_	_			-							
					Le	ve	2 c	acl	he								
				G	DDR5	M	emo	ry	Syste	m							
						_	_	_									



## HSA INTERMEDIATE LAYER - HSAIL

•HSAIL is a virtual ISA for parallel programs

- Finalized to ISA by a runtime compiler or "Finalizer"
- Explicitly parallel
  - Designed for data parallel programming
- Support for exceptions, virtual functions, and other high level language features
- Syscall methods
  - GPU code can call directly to system services, IO, printf, etc

Debugging support

			AMD Ra	deon	H	D7970					
	Asy	nchronous Comp	oute Engine		Asynchronous Compute Engine						
		/ command Pro	1003301		/ Command Processor						_
e		SC SIMD Core	L1 LDS			LDS L	1	SIMD Core	SC		S
ach	I cache	SC SIMD Core	L1 LDS			LDS L	1	SIMD Core	SC	l ca	Cc
U U		SC SIMD Core	L1 LDS			LDS L	1	SIMD Core	SC	Che	ach
S		SC SIMD Core	L1 LDS	Ce		LDS L	1	SIMD Core	SC		e
e	-	SC SIMD Core	L1 LDS	erfa		LDS L	1	SIMD Core	SC		S
ach	cache	SC SIMD Core	L1 LDS	inte		LDS L	1	SIMD Core	SC	Ca	Cc
U U		SC SIMD Core	L1 LDS	Dr.V		LDS L	1	SIMD Core	SC	che I cache	ach
S		SC SIMD Core	L1 LDS	Du		LDS L	1	SIMD Core	SC		æ
e	cache	SC SIMD Core	L1 LDS	me		LDS L	1	SIMD Core	SC		S
ach		SC SIMD Core	L1 LDS	Write		LDS L	1	SIMD Core	SC		00
C		SC SIMD Core	L1 LDS			LDS L	1	SIMD Core	SC		ach
S		SC SIMD Core	L1 LDS	/pe		LDS L	1	SIMD Core	SC		e
e	-	SC SIMD Core	L1 LDS	Re		LDS L	1	SIMD Core	SC	_	S
ach	che	SC SIMD Core	L1 LDS			LDS L	1	SIMD Core	SC	Ca	CC
C	Ca	SC SIMD Core	L1 LDS			LDS L	1	SIMD Core	SC	che	ach
S		SC SIMD Core	L1 LDS			LDS L	1	SIMD Core	SC		e
										-	
			Leve	el 2 ca	acł	ne					
			GDDR5 M	lemo	ry	System					

AMD

## HSA MEMORY MODEL

- Designed to be compatible with C++11, Java and .NET Memory Models
- Relaxed consistency memory model for parallel compute performance
- Loads and stores can be re-ordered by the finalizer
- Visibility controlled by:
  - Load.Acquire
  - Store.Release
  - Barriers

		A	AMD F	Radec	on F	ID797	0				
As	ynchronous Comp / Command Pro	ute cess	Engin or	e	Asynchronous Compute Engine / Command Processor						
SC cache SC cache SC cache SC cache   I cache I cache I cache I cache	SC SIMD Core SC SIMD Core	11 11 11 11 11 11 11 11 11 11 11 11 11	LDS LDS LDS LDS LDS LDS LDS LDS LDS LDS	Read/Write memory interface		LDS LDS LDS LDS LDS LDS LDS LDS LDS LDS	L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L	SIMD Core SIMD Core	SC   SC	I cache I cache I cache I cache	SC cache SC cache SC cache SC cache
			Le	vel 2	cad	che					
		G	DDR5	Mem	ory	/ Syste	m				



## ARCHITECTED QUEUING LANGUAGE

- Defines dispatch characteristics in a small packet in memory
  - Platform neutral work offload
- Designed to be interpreted by the device
  - Firmware implementations
  - Or directly implemented in hardware

	AMD Radeon HD7970													
Γ	Asy	nchi / C	ronous Comp ommand Pro	ute	Asynchronous Compute Engine									
	, command riocessor								,				_	_
e	che	SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC		S
ach		SC	SIMD Core	L1	LDS	1			LDS	L1	SIMD Core	SC	Ca	Cci
U U	Ca	SC	SIMD Core	L1	LDS			erface	LDS	L1	SIMD Core	SC	che	ach
S	-	SC	SIMD Core	L1	LDS		ace		LDS	L1	SIMD Core	SC		Ð
e		SC	SIMD Core	L1	LDS		erfa		LDS	L1	SIMD Core	SC		S
ach	che	SC	SIMD Core	L1	LDS	1	inte		LDS	L1	SIMD Core	SC	I cache	Cci
U U	l ca	SC	SIMD Core	L1	LDS		Z		LDS	L1	SIMD Core	SC		ach
S		SC	SIMD Core	L1	LDS		memo		LDS	L1	SIMD Core	SC		Ð
e	che	SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC		S
ach		SC	SIMD Core	L1	LDS		ite		LDS	L1	SIMD Core	SC	ca	Cc
U U	Ca	SC	SIMD Core	L1	LDS		Vr		LDS	L1	SIMD Core	SC	che	ach
S		SC	SIMD Core	L1	LDS		/pe	in the second	LDS	L1	SIMD Core	SC		P
e		SC	SIMD Core	L1	LDS		Rea		LDS	L1	SIMD Core	SC		S
ach	che	SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC	ca	Cci
U U	ca	SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC	che	ach
S		SC	SIMD Core	L1	LDS				LDS	L1	SIMD Core	SC		e
													-	
					Le	ve	el 2 c	ac	he					
				G	DDR5	м	emo	ry	Syste	m				

- User space memory allows queues to span devices
- Standardized packet format (AQL) enables flexible and portable use
- Single consumer, multiple producer of work
  - Enables support for task queuing runtimes and device->self enqueue



- User space memory allows queues to span devices
- Standardized packet format (AQL) enables flexible and portable use
- Single consumer, multiple producer of work
  - Enables support for task queuing runtimes and device->self enqueue



- User space memory allows queues to span devices
- Standardized packet format (AQL) enables flexible and portable use
- Single consumer, multiple producer of work
  - Enables support for task queuing runtimes and device->self enqueue



- User space memory allows queues to span devices
- Standardized packet format (AQL) enables flexible and portable use
- Single consumer, multiple producer of work
  - Enables support for task queuing runtimes and device->self enqueue



- User space memory allows queues to span devices
- Standardized packet format (AQL) enables flexible and portable use
- Single consumer, multiple producer of work
  - Enables support for task queuing runtimes and device->self enqueue



- User space memory allows queues to span devices
- Standardized packet format (AQL) enables flexible and portable use
- Single consumer, multiple producer of work
  - Enables support for task queuing runtimes and device->self enqueue



- User space memory allows queues to span devices
- Standardized packet format (AQL) enables flexible and portable use
- Single consumer, multiple producer of work
  - Enables support for task queuing runtimes and device->self enqueue



- User space memory allows queues to span devices
- Standardized packet format (AQL) enables flexible and portable use
- Single consumer, multiple producer of work
  - Enables support for task queuing runtimes and device->self enqueue



# **Driver Stack**

# **HSA Software Stack**



## **OPENCL™ AND HSA**

- ■HSA is an optimized platform architecture for OpenCL<sup>™</sup>
  - Not an alternative to OpenCL<sup>™</sup>
- OpenCL<sup>™</sup> on HSA will benefit from
  - Avoidance of wasteful copies
  - Low latency dispatch
  - Improved memory model
  - Pointers shared between CPU and GPU
- HSA also exposes a lower level programming interface, for those that want the ultimate in control and performance
  - Optimized libraries may choose the lower level interface





# QUESTIONS




## **Disclaimer & Attribution**

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

AMD

© 2011 Advanced Micro Devices, Inc.