AMD
Smarter Choice

Advanced R7xx Features

Compute Shaders

AMD
Smarter Choice

- More general approach for GPU Compute
  - Removes graphics-centric terminology and ideas
  - Exposes GPU as an array of parallel processing elements
  - Removes graphics pipeline from the picture (no PS, GS, VS)
- Disconnects output domain from execution domain
  - Read anywhere, write anywhere (Global Buffer)
  - Linear memory format
  - Gives more control to the kernel writer on thread execution and corresponding optimizations

Compute Terminology

AMD
Smarter Choice

- Thread – Single invocation of a kernel
- Group – Set of threads that can share data and run together on a single SIMD. Multiple groups can run on a single SIMD if registers allow
- Wavefront – Group of 64 threads running concurrently on a SIMD (16 SPs * 4 cycles)
- Neighborhood - Group of 4 threads in the same Wavefront having consecutive thread IDs (Tid)

Using Compute Mode in IL

- Header

  `il_cs_2_0` (Instead of `il_ps_2_0`)
- Number of threads per group

  `dcl_num_thread_per_group 64`
- New Indexing Values – No more vPos/vWinCoord
  - `vTid` – ID of thread within a group
  - `vaTid` – ID of thread within a domain
  - `vTgroupid` – ID of group within a domain
    - e.g.

      Group ID (`l0.x = 6`) Upper 26 bits in `vaTid0` in above case

      `ishr r0.x, vaTid0.x, l0.x`

      Tid within a group (`l1.w = 0x3F`) Lower 6 bits in `vaTid0`

      `and r0.y, vaTid0.x, l1.w`

Using Compute Mode in CAL

- New Entry Points

  `calCtxRunProgramGrid`

  – Routine to launch kernel in *Compute Mode*

  – Exposed as a CAL extension

- New Domain specification mechanism

  `CALprogramGrid`

  – Specifies various parameters for kernel launch

```
struct {
  CALfunc     func;      /* CALfunc to execute */
  CALdomain3D gridBlock; /* size of a block of data */
  CALdomain3D gridSize;  /* size of 'blocks' to execute */
  CALuint     flags;       /* misc grid flags */
} CALprogramGrid;
```

Using Compute Mode in CAL

AMD Smarter Choice

```
CALprogramGrid pg;
pg.func             = func;
pg.flags            = 0;
pg.gridBlock.width  = 64; // same as the value in the
                          // kernel for block size
pg.gridBlock.height = 1;
pg.gridBlock.depth  = 1;
pg.gridSize.width   = (1024 * 1024 + 63) / pg.gridBlock.width;
pg.gridSize.height  = 1;
pg.gridSize.depth   = 1;

// Get the function ptr for CAL Extension
calExtGetProc((CALextproc*)&calCtxRunProgramGrid,
  CAL_EXT_COMPUTE_SHADER, "calCtxRunProgramGrid");

// Launch the kernel in compute mode
calCtxRunProgramGrid(&event, *ctx, &pg);
```

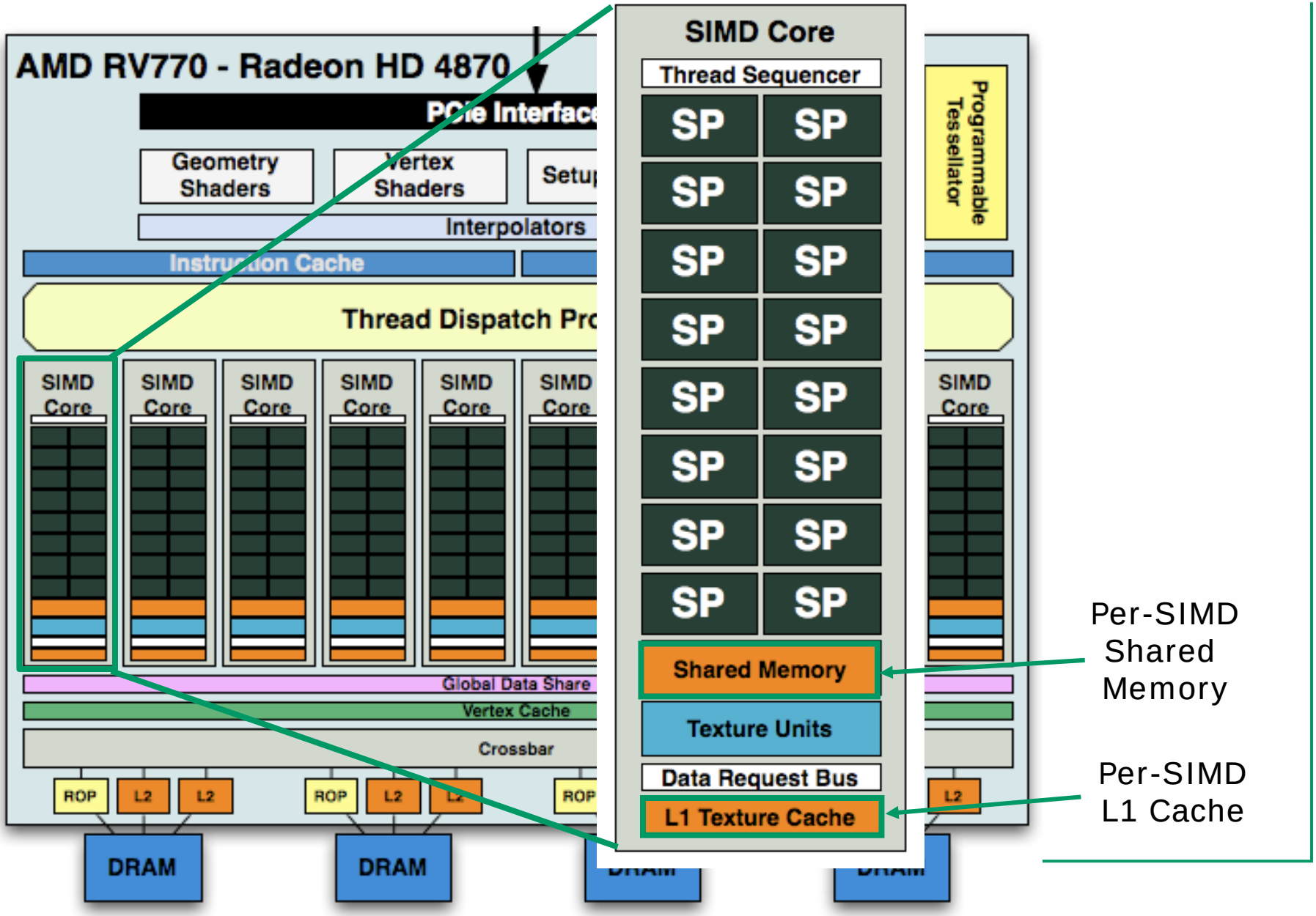
Using Compute Mode

AMD Smarter Choice

- Key Items to Remember
  - Output resources are required to be Global Buffers (only 1 supported).
  - Cache characteristics will be different from ‘regular kernels’ due to different execution order, e.g. for 8 MRT MMM algorithm implemented using CAL,
    - PS 8 MRT - 393 Gflops
    - PS MemExport - 393 Gflops
    - CS MemExport - 222 Gflops
  - R7xx supports only linear *thread dispatch*
    - True 3D grid blocks available with future hardware only
    - For R7xx, `gridBlock.width == dcl_num_thread_per_group`

R7xx - 2008

AMD
Smarter Choice



Data Sharing

AMD
Smarter Choice

- Local Data Share (LDS)
  - 16kb On-chip memory per SIMD shared between threads in a block
  - Write local, read global system
  - Share between all threads in a block
  - Synchronization required
- Shared Registers (SR) – Globally shared registers
  - Registers that are global to a SIMD
  - Sharing between all wavefronts in a SIMD
  - Column sharing on the SIMD
  - Persistent registers
  - Atomic read, modify, write in same instruction guaranteed

Using LDS in IL

AMD Smarter Choice

- Size of LDS memory to be used in a shader in dwords

  `dcl_lds_size_per_thread n`

  `n <=64` and a factor of 4.
- LDS Memory sharing

  `dcl_lds_sharing_mode mode`

  `where mode` can be
  - `_wavefrontRel =>` Relative, i.e. each wavefront has its private LDS memory
  - `_wavefrontAbs =>` Absolute, i.e. all wavefronts share the same piece of LDS memory

Using LDS in IL

AMD Smarter Choice

- Reading LDS Memory

  `read_lds (_neighborExch)(_sharingMode) dst, src0.xy`

  LDS location is given by `src0.xy`, where `src0.x = Tid`, `src0.y = offset`

  `dst` can be any register

  Options Flags

  – `_sharingMode(rel)` or `_sharingMode(abs)` for relative or absolute sharing mode.

  – `_neighborExch` If specified, the output of LDS will be exchanged with its neighboring threads such that

    - first thread gets all values from x-channels
    - second thread gets all values from y-channels, and so on.

    This flag is useful for applications like FFT matrix transpose.

Using LDS in IL

AMD Smarter Choice

- Writing LDS Memory

  `write_lds (offset) (_sharingMode) dst, src`

  `src` can be any register

  Location is fixed to (`Tid`, `offset`)

  `dst` must be of type `IL_REGTYPE_GENERIC_MEM`. This is only used to provide write mask

  Options Flags

  – `_sharingMode(rel)` or `_sharingMode(abs)` for relative or absolute sharing mode.

  – `_lOffset(n)`

    - If not specified, offset = 0
    - n must be a value of multiples of 4 in the range of [0, 60] and smaller than declared `lds_size_per_thread`

Synchronization

AMD Smarter Choice

- Fence
  - Synchronization mechanism for threads within a group
  - No thread in the group should pass that point until all threads reach the point
  - Disallow compiler optimizations to occur around that point
  - The `fence` instruction has four flags - One of the flags must be present and they can exist in any order
    - `_lds` is for LDS accesses
    - `_threads` is for thread synchronization
    - `_memory` is for non-lds memory accesses
    - `_shared` is for SR accesses

Q&A and Recap

AMD
Smarter Choice

- RV770 New Features
  - Compute Shaders
  - Data Sharing Mechanisms