GPU THREADING MODEL

OPENCL 2.0 UNIVERSITY TOOLKIT

Zhongliang Chen and Yash Ukidave,
Northeastern University Computer Architecture Research Lab
with
Perhaad Mistry and Dana Schaa, AMD

INSTRUCTOR NOTES

AMD

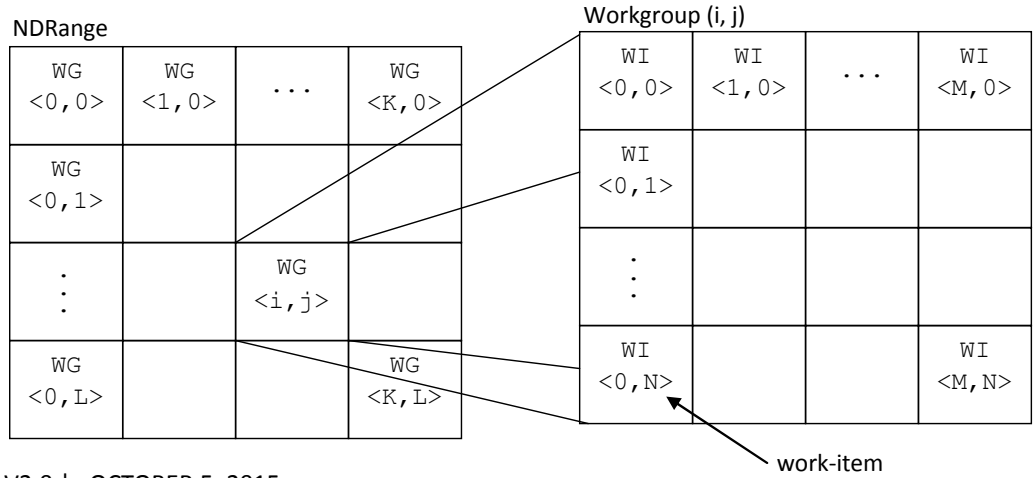
- This section describes how work-groups are scheduled for execution on the compute units of devices
- We cover effects of divergence of work-items within a group and its negative effect on performance
- Reasons for discussing wavefronts/warps because even though they are not part of the OpenCL specification
  - Serve as another hierarchy of threads and their implicit synchronization enables interesting implementations of algorithms on GPUs
  - Implicit synchronization and write combining property in local memory used to implement warp voting
  - Use of predication for divergent work-items even though all threads in a warp are issued in lockstep

TOPICS

- Wavefronts and warps
- Thread scheduling on GPUs
- Predication
- Warp voting and synchronization
- Pitfalls of wavefront/warp specific implementations

WORK GROUPS TO HW THREADS

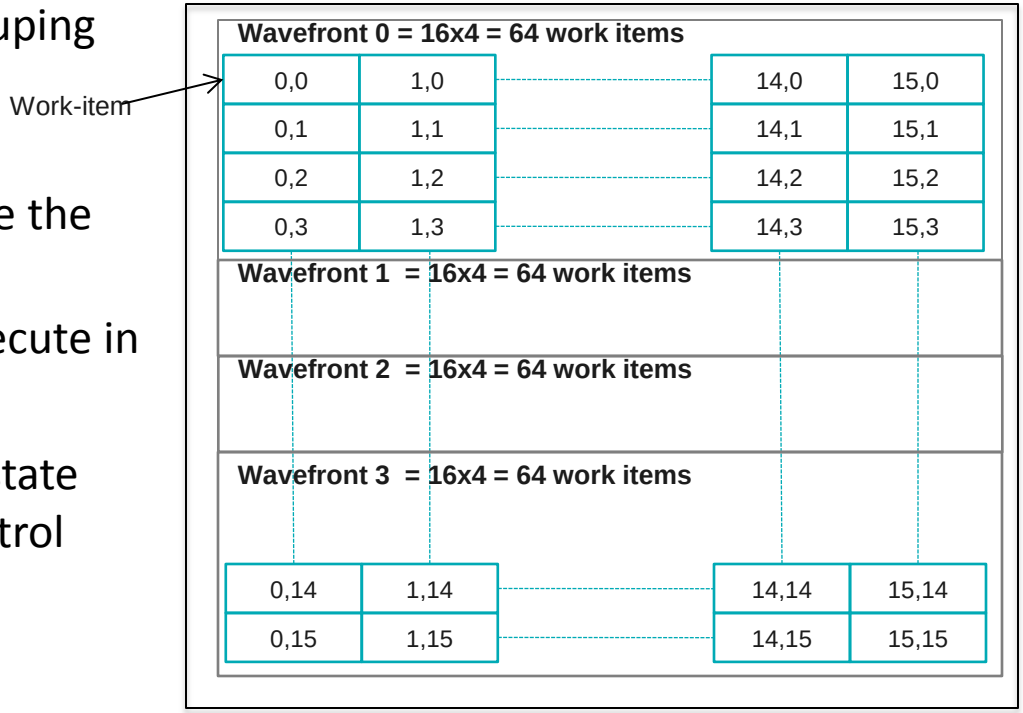
- OpenCL kernels are structured into work-groups that map to device compute units
- Compute units on GPUs consist of SIMT processing elements
- Work-groups automatically get broken down into hardware schedulable groups of threads for the SIMT hardware
  - This “schedulable group” is known as a wavefront (AMD) or a warp (NVIDIA)



WORK-ITEM SCHEDULING

- Hardware creates wavefronts by grouping work-items of a work-group
  - Along the X dimension first
- All work-items in a wavefront execute the same instruction
  - Work-items within a wavefront execute in lockstep
- Work-items have their own register state and are free to execute different control paths
  - Work-item masking used by HW
  - Predication can be set by compiler

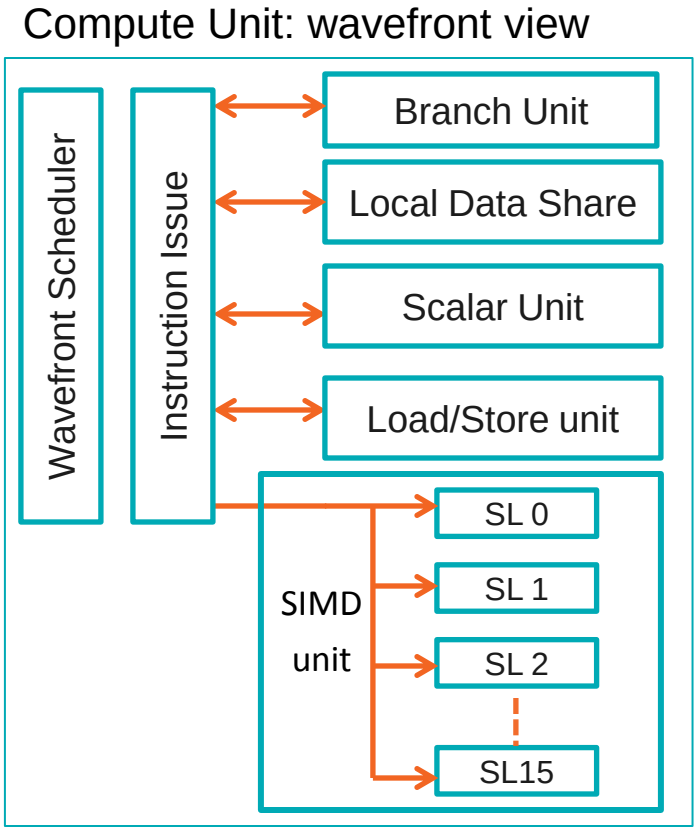
Example 16x16 Workgroup



Grouping of work-group into wavefronts

WAVEFRONT SCHEDULING

- Wavefront size is 64 work-items
  - Vector instructions performed with one lane per work-item
  - Scalar instructions are performed once for entire wavefront
  - Vector load/store instructions supply one address per work-item
- A SIMD lane (SL) executes one vector instruction
  - 16 stream cores execute 16 vector instructions on each cycle
- A quarter of the wavefront (16 work-items) is issued on each cycle to the SIMD
  - The entire wavefront is issued over four consecutive cycles
  - The entire wavefront issues to the other units in a single cycle



WAVEFRONT SCHEDULING

AMD

- In the case of Read-After-Write (RAW) hazard, one wavefront will stall for four extra cycles
  - If another wavefront is available it can be scheduled to hide latency
  - After eight total cycles have elapsed, the ALU result from the first wavefront is ready, so the first wavefront can continue execution
- Two wavefronts (128 work-items) completely hide a RAW latency
  - The first wavefront executes for four cycles
  - Another wavefront is scheduled for the next four cycles
  - The first wavefront can then run again
- Note that two wavefronts are needed just to hide RAW latency, the latency to global memory is much greater
  - During this time, the compute unit can process other independent wavefronts, if they are available

GPU OCCUPANCY TRADEOFFS

AMD

- Local memory and registers are persistent within compute unit once a work-group is scheduled
  - Traditional context switching is not used
  - Allows for no-overhead wavefront interleaving
- Number of active wavefronts supported per compute unit is limited
  - Decided by
    - local memory required per work-group
    - register usage per work-item
- Number of active wavefronts possible on a compute unit can be expressed using a metric called *occupancy*
- Larger numbers of active wavefronts allow for better latency hiding on both AMD and NVIDIA GPUs

DIVERGENT CONTROL FLOW

AMD

- Although work-items have unique program counters, in practice they are executed in lockstep on SIMD hardware
- Work-items can execute different path from other work-items in the wavefront using masking or predication
- For correctness, lock-step execution is transparent to the programmer
- In practice, branching should be limited to a wavefront granularity whenever possible to fully utilize SIMD execution units

PREDICATION AND CONTROL FLOW

AMD

- How do we handle work-items going down different execution paths when the same instruction is issued for all the work-items in a wavefront ?
- Predication is a method for mitigating the costs associated with conditional branches
  - Beneficial in case of branches to short sections of code
  - Based on fact that executing an instruction and squashing its result may be as efficient as executing a conditional
  - Compilers may replace “switch” or “if then else” statements by using branch predication

PREDICATION FOR GPUS

AMD

- Predicate is a condition code that is set to true or false based on a conditional
- Both cases of conditional flow get scheduled for execution
  - Instructions with a true predicate are committed
  - Instructions with a false predicate do not write results or read operands
- Removes branches
  - Benefits performance only for very short conditionals

```
__kernel void test()
{
    int  tid= get_local_id(0);

    if( tid %2 == 0)
        Do_Some_Work();
    else
        Do_Other_Work();
}
```

Predicate = True for work-items 0,2,4….
Predicate = False for work-items 1,3,5….
Predicates switched for the else condition

DIVERGENT CONTROL FLOW

- **Case 1:** All **odd** work-items will execute if conditional while all **even** work-items execute the else conditional. The if and else block need to be issued for each wavefront
- **Case 2:** All work-items of the first wavefront will execute the if case while other wavefronts will execute the else case. In this case only one out of if or else is issued for each wavefront

Case 1

```
int tid = get_local_id(0)
if ( tid % 2 == 0) // even work-Items
    DoSomeWork()
else  // odd work-Items
    DoSomeWork2()
```

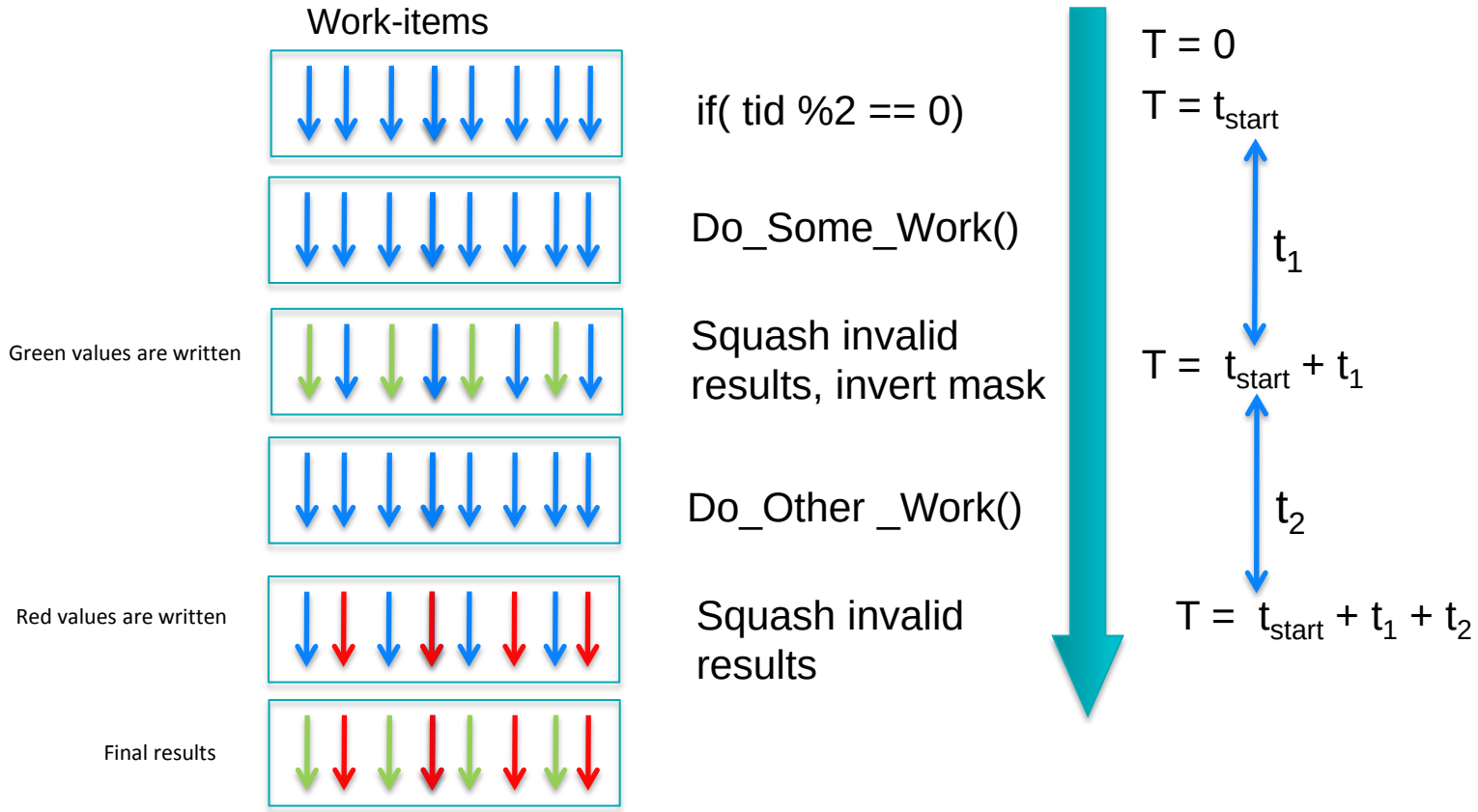
With divergence

Case 2

```
int tid = get_local_id(0)
if ( tid / 64 == 0)  // first wavefront
    DoSomeWork()
else if (tid /64 == 1) // second wavefront
    DoSomeWork2()
```

Without divergence

EFFECT OF PREDICATION ON PERFORMANCE



SUMMARY

AMD

- Divergence within a work-group should be restricted to a wavefront/warp granularity for performance
- A tradeoff between schemes to avoid divergence and simple code which can quickly be predicated
  - Branches are usually highly biased and localized which leads to short predicated blocks
- The number of wavefronts active at any point in time should be maximized to allow latency hiding
  - Number of active wavefronts is determined by the requirements of resources like registers and local memory
- Wavefront specific implementations can enable more optimized implementations and enables more algorithms to GPUs
  - Maintaining performance and correctness may be hard due to the different wavefront sizes on AMD and NVIDIA hardware