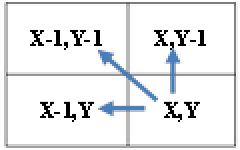
Wavefront Parallel Processing on GPUs with an Application to Video Encoding Algorithms

Biju George, Ben Ashbaugh

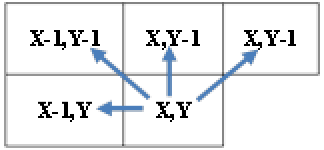
IWOCL 2017

Wavefront Parallel Processing (WPP)

- Efficient Parallel Processing technique for problems characterized by specific patterns of data dependencies across an n-dimension grid
- Patterns referred to as Wavefront Dependency Patterns



45° dependencies



26° dependencies

- Key observations:
  - Data dependencies satisfied by ordered traversals along diagonals a.k.a *wavefronts*
  - Independent computations in a wavefront

0,0 [0]	1,0 [1]	2,0 [2]	3,0 [3]	4,0 [4]	5,0 [5]	6,0 [6]
0,1 [1]	1,1 [2]	2, 1 [3]	3, 1 [4]	4, 1 [5]	5, 1 [6]	6,1 [7]
0,2 [2]	1,2 [3]	2, 2 [4]	3, 2 [5]	4, 2 [6]	5, 2 [7]	6,2 [8]
0,3 [3]	1,3 [4]	2, 3 [5]	3, 3 [6]	4, 3 [7]	5, 3 [8]	6,3 [9]
0,4 [4]	1,4 [5]	2, 4 [6]	3, 4 [7]	4, 4 [8]	5, 4 [9]	6,4 [10]
0,5 [5]	1,5 [6]	2, 5 [7]	3, 5 [8]	4, 5 [9]	5, 5 [10]	6,5 [11]

45° wavefront traversal

0,0 [0]	1,0 [1]	2,0 [2]	3,0 [3]	4,0 [4]	5,0 [5]	6,0 [6]
0,1 [2]	1, 1 [3]	2, 1 [4]	3, 1 [5]	4, 1 [6]	5, 1 [7]	6,1 [8]
0,2 [4]	1, 2 [5]	2, 2 [6]	3, 2 [7]	4, 2 [8]	5, 2 [9]	6,2 [10]
0,3 [6]	1, 3 [7]	2, 3 [8]	3, 3 [9]	4, 3 [10]	5, 3 [11]	6,3 [12]
0,4 [8]	1, 4 [9]	2, 4 [10]	3, 4 [11]	4, 4 [12]	5, 4 [13]	6,4 [14]
0,5 [10]	1, 5 [11]	2, 5 [12]	3, 5 [13]	4, 5 [14]	5, 5 [15]	6,5 [16]

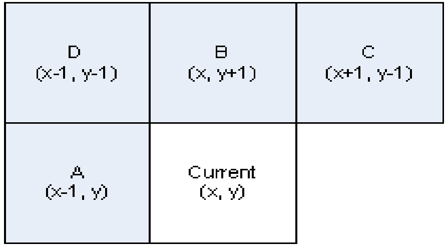
26° wavefront traversal

Applications Of WPP

- Scientific algorithms based on dynamic programming – Smith-Waterman for genome sequencing
  - Large grids & less computation at grid points
- Modern video encoding algorithms – AVC & HEVC
  - Small grids & much computation at grid points
- Image analysis – Morphological Reconstruction
  - Large grids & dynamic dependencies at grid points

-	-	A	T	C	G	A	A
-	0	0	0	0	0	0	0
C	0	0	0	5	1	0	0
A	0	5	1	1	2	5	5
T	0	1	10	6	2	1	2
A	0	5	6	7	3	7	6
C	0	1	2	11	7	3	4

Smith-Waterman sequence alignment



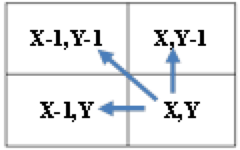
Differential encoding of motion vectors in video encoding

Video Encoding Algorithms

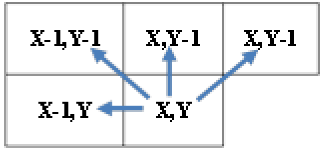
- Video encoding algorithms exploit temporal (inter frame) and spatial (intra frame) similarities across and within frames
- Modern video encoding algorithms employ block based video motion estimation (VME) to do this
- Dominant compute intensive component
- Critical to efficiently extract parallelism for performance
- Exhibits 26˚ and 45˚ wavefront patterns for *“Predicted Motion Vector”* (PMV) and *“Most Probable Mode”* (MPM)

[4]

Inter and Intra frame motion estimation

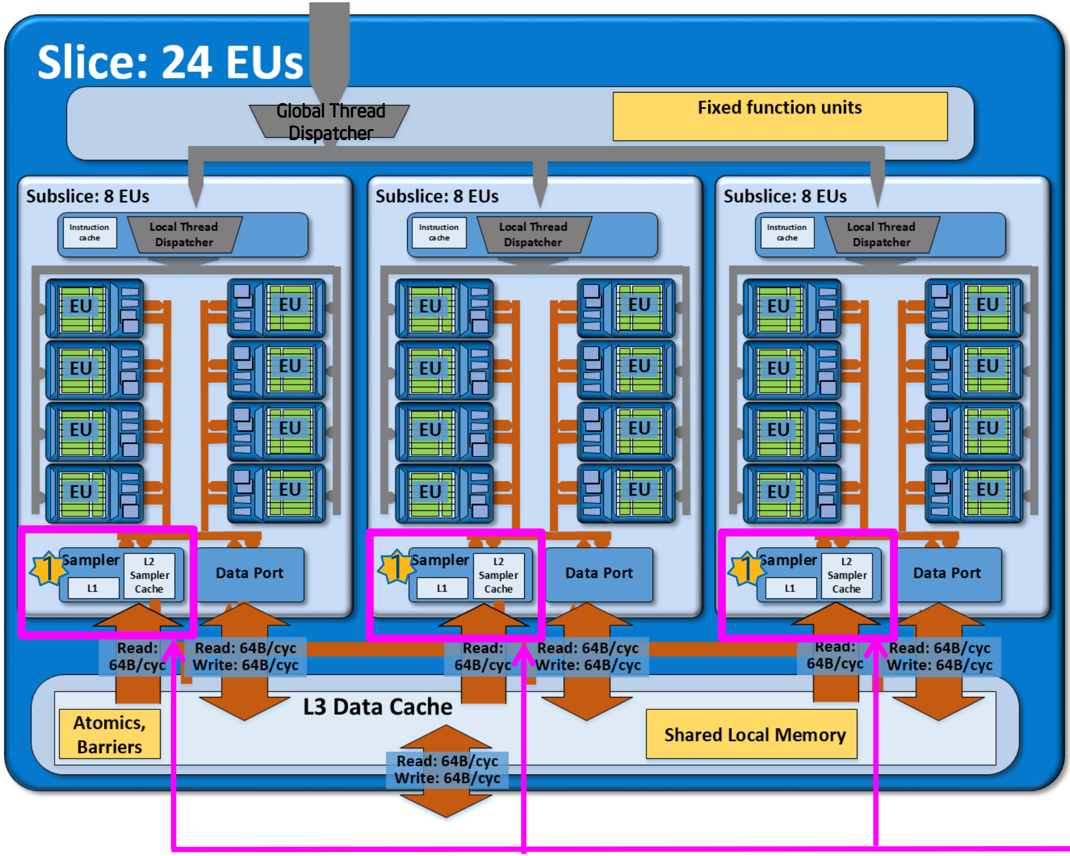


45° dependencies



26° dependencies

Intel® Graphics 530 GPU Architecture



Sub-slice

- SLM shared by sub-slice EU threads
- Barrier synchronization in sub-slice
- One VME engine

Programmable VME Engine

- Optimized for memory bandwidth
- Provides configurable raw compute
- Smarts in the hands of the programmer

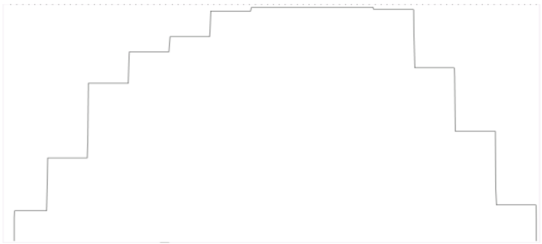
1 Powerful Video Motion Estimation (VME) Engine in Sampler

OpenCL SW Interface

- Device-side VME vendor extension - exposes programmable VME functionality in GPU
- Set of built-in functions **callable from user written OpenCL kernels** – maps closely with exposed HW interface
    - Essentially provides a very low-level motion estimation library with a underlying HW implementation – think of it as Inter Performance Primitives (IPP).
- Subgroups functions for block API

Challenges with WPP on GPUs with OpenCL

- Challenges with synchronization between WGs
  - GPU schedulers not particularly designed to handle dependencies across work-groups (WGs)
  - OpenCL spec allows launch order of WGs to be implementation specific
  - Non-preemptable nature of WGs
- Challenges with expanding and contracting parallelism
  - Not having enough compute to saturate machine
  - Idle polling

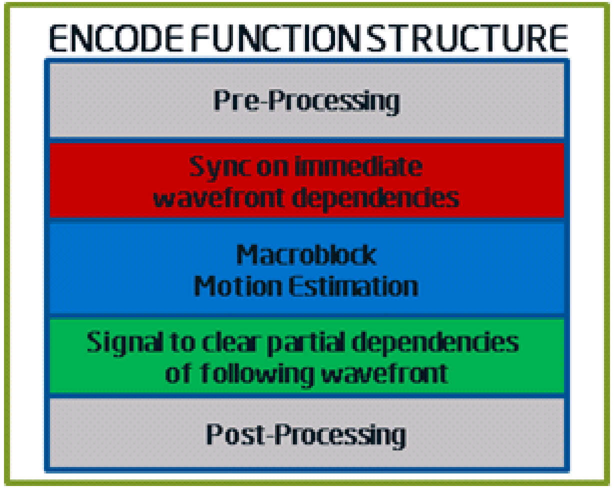


Plot of parallelism as wavefront progresses

IMPLEMENTED SOLUTIONS

Implemented Solutions

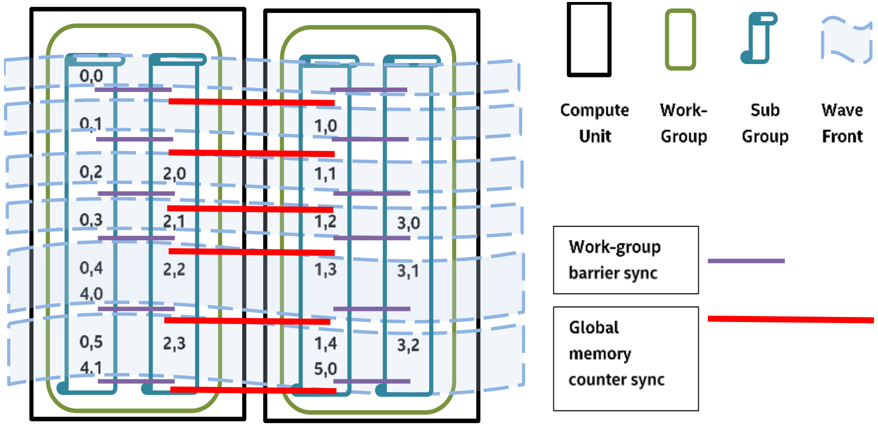
- Four WPP OpenCL solutions implemented and evaluated on Intel® Processor Graphics (Intel® Graphics 530)
  - Same basic encode kernel structure
    - Uses custom global memory barriers
    - No data dependencies for pre and post-processing stages
    - Per Amdahl's law the Motion Estimation stage is the performance critical part
    - Pre and post-processing part move work out of the performance critical part
  - Up to 9 forward reference frames searched per MB.
  - VME operations leveraged through Intel OpenCL device-side VME extensions



- Major considerations
  - Maximally utilize achievable parallelism
  - Efficient synchronization

Persistent Threads with Distributed Wavefront Sweep

- Subgroups active for entire kernel duration – w/a launch order issues
- One work-group runs on a compute unit
- Maximal launch of subgroups
- Work-queues process ordered wavefronts
- Subgroups with work-group sync efficiently using barriers
- Inter work-group sync using global memory counter
- Efficient sync



0,0 [0]	1,0 [1]	2,0 [2]	3,0 [3]	4,0 [4]	5,0 [5]	6,0 [6]
0,1 [1]	1,1 [2]	2, 1 [3]	3, 1 [4]	4, 1 [5]	5, 1 [6]	6,1 [7]
0,2 [2]	1,2 [3]	2, 2 [4]	3, 2 [5]	4, 2 [6]	5, 2 [7]	6,2 [8]
0,3 [3]	1,3 [4]	2, 3 [5]	3, 3 [6]	4, 3 [7]	5, 3 [8]	6,3 [9]
0,4 [4]	1,4 [5]	2, 4 [6]	3, 4 [7]	4, 4 [8]	5, 4 [9]	6,4 [10]
0,5 [5]	1,5 [6]	2, 5 [7]	3, 5 [8]	4, 5 [9]	5, 5 [10]	6,5 [11]

45˚ wavefront data distribution

Persistent Threads with Distributed Wavefront Sweep

```
void poll(__global atomic_int* counter, int threshold) {
    int entry = threshold - 1;
    // Only one representative work-item from representative subgroup
    // needs to poll.
    if (get_sub_group_local_id()==0&&  get_sub_group_id() == 0) {
        while (entry != threshold) {
            entry = atomic_load_explicit(
                counter, memory_order_acquire, memory_scope_device);
        }
    }
   work_group_barrier(CLK_LOCAL_MEM_FENCE);
}

void signal(__global atomic_int* counter) {
    // Only one representative work-item from representative subgroup
    // needs to signal.
    if (get_sub_group_local_id() == 0) {
        atomic_fetch_add_explicit(
            counter, 1, memory_order_acq_rel, memory_scope_device);
    }
}
```

Synchronization functions

- OpenCL 2.0 memory model atomics needed to guarantee correctness
  - Polls with acquire semantics
  - Signal with acquire-release semantics
  - Writes from subgroups signaling a counter update need to be visible in subgroups polling for the same counter update
- Major drawback is inability to extract partial parallelism across wavefronts.

Persistent Threads with Cyclic Computation

- Data partitioned into unit intervals along an axis and assigned to persistent threads in round-robin
- Only works if no forward dependency across intervals
- For 45° wavefront, cyclic distribution along x or y axis
- For 26° wavefront, cyclic distribution only along y because of top-right dependency
- Persistent threads - one subgroup per WG

WG0	WG1	WG2	WG3	WG0	WG1	WG2
0,0 [0]	1,0 [1]	2,0 [2]	3,0 [3]	4,0 [4]	5,0 [5]	6,0 [6]
0,1 [1]	1,1 [2]	2, 1 [3]	3, 1 [4]	4, 1 [5]	5, 1 [6]	6,1 [7]
0,2 [2]	1,2 [3]	2, 2 [4]	3, 2 [5]	4, 2 [6]	5, 2 [7]	6,2 [8]
0,3 [3]	1,3 [4]	2, 3 [5]	3, 3 [6]	4, 3 [7]	5, 3 [8]	6,3 [9]
0,4 [4]	1,4 [5]	2, 4 [6]	3, 4 [7]	4, 4 [8]	5, 4 [9]	6,4 [10]
0,5 [5]	1,5 [6]	2, 5 [7]	3, 5 [8]	4, 5 [9]	5, 5 [10]	6,5 [11]

1-step intervals

45° wavefront data distribution

WG0	0,0 [0]	1,0 [1]	2,0 [2]	3,0 [3]	4,0 [4]	5,0 [5]	6,0 [6]
WG1	0,1 [2]	1, 1 [3]	2, 1 [4]	3, 1 [5]	4, 1 [6]	5, 1 [7]	6,1 [8]
WG2	0,2 [4]	1, 2 [5]	2, 2 [6]	3, 2 [7]	4, 2 [8]	5, 2 [9]	6,2 [10]
WG3	0,3 [6]	1, 3 [7]	2, 3 [8]	3, 3 [9]	4, 3 [10]	5, 3 [11]	6,3 [12]
WG0	0,4 [8]	1, 4 [9]	2, 4 [10]	3, 4 [11]	4, 4 [12]	5, 4 [13]	6,4 [14]
WG1	0,5 [10]	1, 5 [11]	2, 5 [12]	3, 5 [13]	4, 5 [14]	5, 5 [15]	6,5 [16]

2-step intervals

26° wavefront data distribution

Persistent Threads with Cyclic Computation

```
int2 mbid = { get_group_id(0), 0 };
do {
  int2 imgsize = get_image_dim(srcimg);
  int2 framembsize = (imgsize + (int2)(15, 15)) / 16;
   preprocess(…);
   if (mbid.x > 0) {
       poll(scoreboard + mbid.x - 1, mbid.y + 1);
   }
   if (!skip_block) {
       block_motion_estimate_process(...);
   }
   signal(scoreboard + mbid.x);
   postprocess(…); mbid.y += 1;
   // Cycle computation in interval
   if (mbid.y == framembsize.y) {
       mbid.y = 0; mbid.x += get_num_groups(0);
   }
 } while (mbid.x < framembsize.x);
```

Poll neighbor interval counter

Signal current interval counter

Cyclic computation of wavefronts

- Subgroups process intervals
- Synch using global memory counters – one per WG
- Similar pair of sync functions
- Enables overlapped partial execution of multiple wavefronts
- Drawback is not having enough threads to saturate GPU for lower resolutions

Distributed Computation of Wavefronts

- Similar to cyclic computation approach
- Proposed extension for OpenCL runtime to fill GPU deterministically using pre-defined pattern
  - *reqd_launch_pattern(pattern)*
  - ‘native’ or ‘raster’, or ‘custom’ patterns
- Eliminates cycling step
- Enables WG-level pre-emption if in an environment with context switch latency requirements

WG0	WG1	WG2	WG3	WG4	WG5	WG6
0,0 [0]	1,0 [1]	2,0 [2]	3,0 [3]	4,0 [4]	5,0 [5]	6,0 [6]
0,1 [1]	1,1 [2]	2, 1 [3]	3, 1 [4]	4, 1 [5]	5, 1 [6]	6,1 [7]
0,2 [2]	1,2 [3]	2, 2 [4]	3, 2 [5]	4, 2 [6]	5, 2 [7]	6,2 [8]
0,3 [3]	1,3 [4]	2, 3 [5]	3, 3 [6]	4, 3 [7]	5, 3 [8]	6,3 [9]
0,4 [4]	1,4 [5]	2, 4 [6]	3, 4 [7]	4, 4 [8]	5, 4 [9]	6,4 [10]
0,5 [5]	1,5 [6]	2, 5 [7]	3, 5 [8]	4, 5 [9]	5, 5 [10]	6,5 [11]

1-step wavefronts

45˚ wavefront data distribution

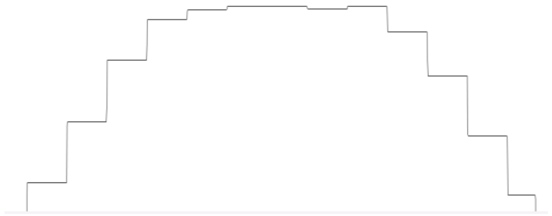
WG0	0,0 [0]	1,0 [1]	2,0 [2]	3,0 [3]	4,0 [4]	5,0 [5]	6,0 [6]
WG1	0,1 [2]	1, 1 [3]	2, 1 [4]	3, 1 [5]	4, 1 [6]	5, 1 [7]	6,1 [8]
WG2	0,2 [4]	1, 2 [5]	2, 2 [6]	3, 2 [7]	4, 2 [8]	5, 2 [9]	6,2 [10]
WG3	0,3 [6]	1, 3 [7]	2, 3 [8]	3, 3 [9]	4, 3 [10]	5, 3 [11]	6,3 [12]
WG4	0,4 [8]	1, 4 [9]	2, 4 [10]	3, 4 [11]	4, 4 [12]	5, 4 [13]	6,4 [14]
WG5	0,5 [10]	1, 5 [11]	2, 5 [12]	3, 5 [13]	4, 5 [14]	5, 5 [15]	6,5 [16]

2-step wavefronts

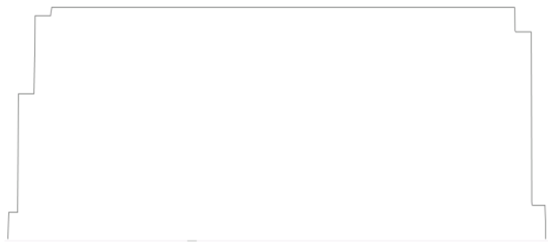
26˚ wavefront data distribution

Persistent Threads with Cyclic Computation of Multiple Independent Wavefronts

- Enhancement of basic cyclic computation to address key drawbacks
  - Unable to saturate GPU for smaller frames
  - Lesser parallelism during wavefront expansion/contraction phases
- Process multiple independent wavefronts
  - from independent encode streams, or
  - from independent slices within same stream
  - we chose 3 wavefronts from different streams



Plot of parallelism as single wavefront progresses



Plot of parallelism with multiple wavefronts

Persistent Threads with Cyclic Computation of Multiple Independent Wavefronts

```
int2 mbid = { 0, get_sub_group_id() };
int2 imgsize = get_image_dim(src0img);
int2 framembsize = (imgsize + (int2)(15, 15)) / 16;
do {
   preprocess(…);
   if (mbid.y > 0) {
      uint threshold = mbid.x + wavefront_step_size;
      threshold = (threshold>framembsize.x)? framembsize.x: threshold;
      poll(scoreboard + mbid.y - 1, threshold, mbid);
   }
   if (get_group_id(0) == 0) {
      if (!skip_block[0])  block_motion_estimate_process(..., src0img, ...)
   } else if (get_group_id(0) == 1) {
      if (!skip_block[1])  block_motion_estimate_process(..., src1img, ...)
   } else if (get_group_id(0) == 2) {
      if (!skip_block[2])   block_motion_estimate_process(..., src2img, ...)
   }
   signal(scoreboard + mbid.y);
   postprocess(…); mbid.x += 1;
   if (mbid.x == framembsize.x) {
      mbid.x = 0;  mbid.y += get_num_sub_groups();
   }
} while (mbid.y < framembsize.y);
```

Poll wavefront neighbor thread counter

Signal wavefront current thread counter

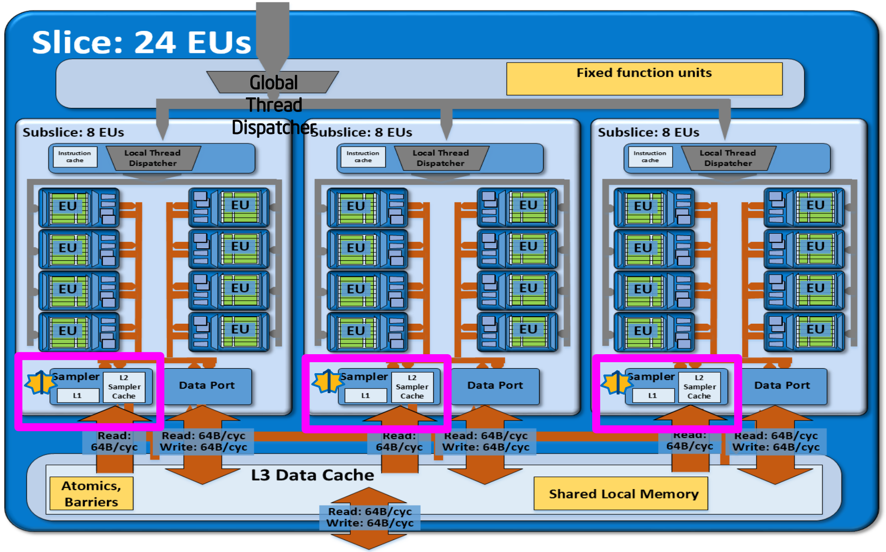
Cyclic computation of multiple independent wavefronts

- Scaled version of basic cyclic approach
  - One persistent WG processing intervals from one set of independent wavefronts
  - Three WGs
  - Maximal launch of subgroups in WGs
- Difference global counters across WGs for sync
- Other benefits
  - Better L1/L2 sampler cache locality

PERFORMANCE EVALUATIONS

Performance Evaluation – Experiment Setup

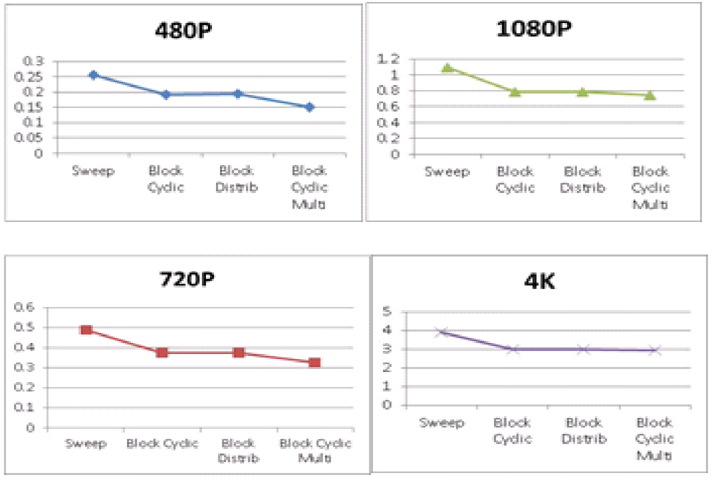
- Key performance metrics
  - GPU execution time per frame
    - Overall performance
  - VME engine busyness
    - Parallelism extracted
  - Count of atomic operations
    - Efficiency of sync



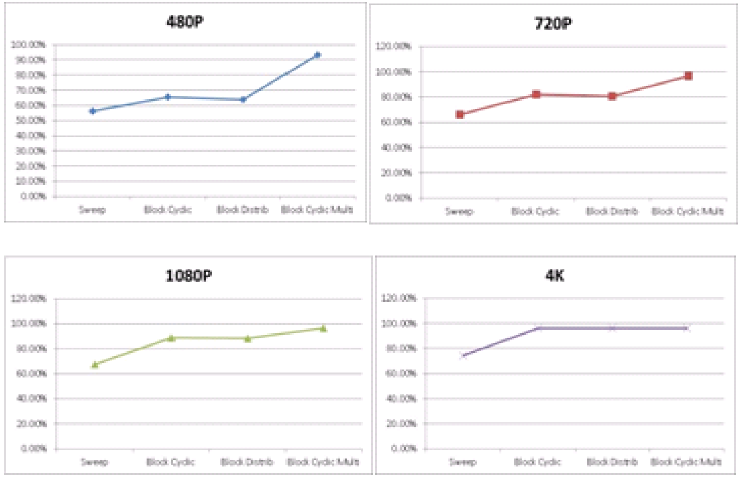
Intel® Graphics 530 Architecture

- Test sequences
  - 480p (858x480), 72-p (1280x720), 1080p (1920x1080), 4k (3840x2160)
    - 15 planar YUV frames
- Force max workgroup size to be 896

Performance Evaluation – Key Observations



GPU execution times comparison

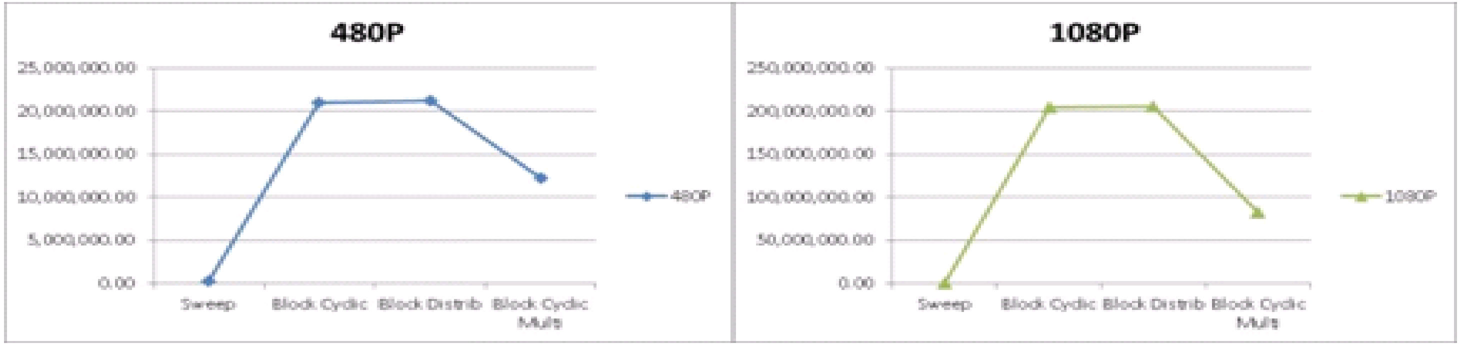


GPU media samples busyness comparison

- Distributed wavefront sweep performed poorly despite most efficient sync
  - Low sampler utilization
  - Extracting parallelism more important
- Cyclic & Distributed computation solutions performed identically

- Multiple independent wavefront solution performed best specially for lower resolutions
  - For 480p 21% over basic cyclic solution; sampler utilization up to 96% from 65%
  - For 4K no noticeable improvement over basic

Performance Evaluation – Key Observations



GPU atomic operation comparison

- Distributed wavefront sweep performed most efficient sync as expected
- Cyclic & Distributed computation solutions had quite of but of idle polling and bandwidth utilization
- Multiple independent wavefront solution performed well
  - Lesser threads per independent wavefront; but enough to keep sampler busy
  - Ergo lesser idle polling load per set of global sych counters

SUMMARY

Summary

- Background and Challenges with WPP on GPUs
- Evaluated 4 WPP solutions for video encoding on Intel® Processor Graphics
  - Cost of sync is not as significant when compared to the efficiency of extracting parallelism
  - Efficiency of sync improved by running just as many threads to keep the VME engine busy
  - Cyclic computation with multiple independent wavefronts solution performed best overall particularly for 720p resolutions and below
  - In cases where only one encode stream is available basic cyclic computation solution is recommended unless multi-slice is an option

Acknowledgements

We would like to acknowledge the valuable inputs and advices of our colleagues Jay Patel, Daniel Rhee, Raghukrishnan Embar, and Saimanohara Alapati.

References

1. https://software.intel.com/sites/default/files/managed/43/c1/cl_intel_device_side_avc_motion_estimation.txt
2. https://software.intel.com/sites/default/files/managed/c6/f8/cl_intel_device_side_avc_vme_programmers_manual.pdf
3. Junkins, Stephen. 2015. The Compute Architecture of Intel® Processor Graphics Gen9. Retrieved from: https://software.intel.com/en-us/file/the-compute-architecture-of-intel-processor-graphics-gen9-v1d0pdf
4. Image in slide 4 created with permission from video content in https://www.youtube.com/user/Faraoni7Prod

Legal Notice and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Legal Disclaimer and Optimization Notice