Achieving Performance with OpenCL 2.0 on Intel® Processor Graphics

Robert Ioffe, Sonal Sharma, Michael Stoner

Legal

One more short message from our lawyers ☺
FTC Disclaimer

Software and workloads used in performance tests may have been optimized for performance only on Intel® microprocessors. Performance tests are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Acknowledgements

Thanks to **Stephen Junkins**, **Aaron Kunze**, **Allen Hux**, **Adam Lake**, **Michał Mrozek**, **Ayal Zaks, Deepti Joshi, Ben Ashbaugh** and **Yuri Kulakov** for input and feedback on this presentation and GPU-Quicksort sample!

Agenda

Shared Virtual Memory in OpenCL 2.0

- Crowd Simulation algorithm
- Border pixel processing
- Cyberlink PowerDirector usage

Device Side Enqueue and Work-group Scan Functions in OpenCL 2.0

- Usage and Benefits
- Sierpinski Carpet Example
- GPU-Quicksort Example

Shared Virtual Memory

Allows de-referencing of host-allocated virtual memory pointers directly on the GPU

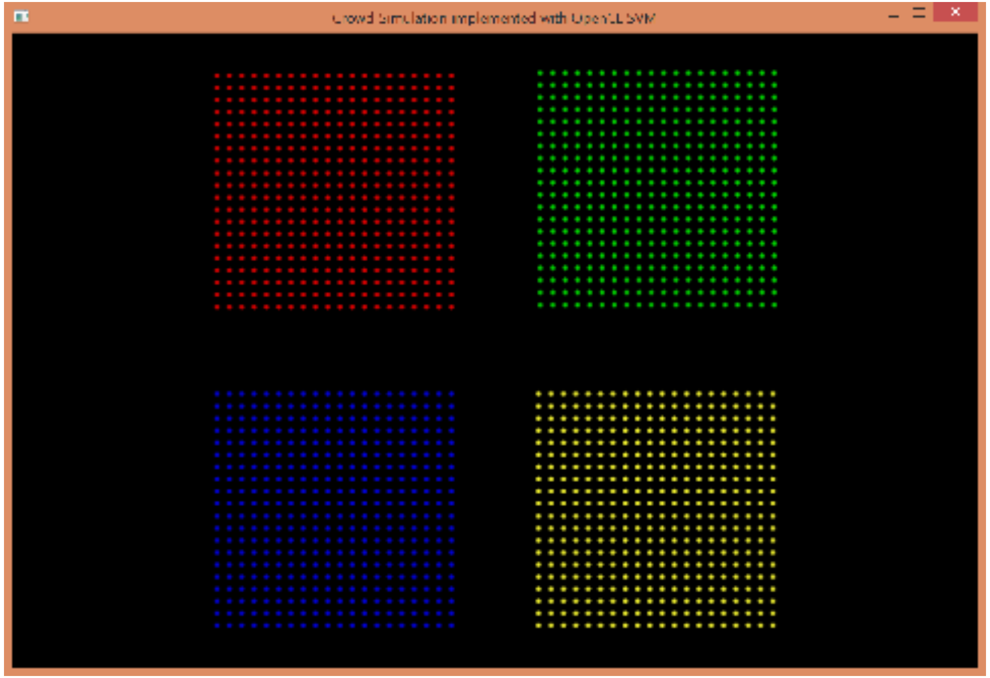
Enables GPU offload of pointer-oriented algorithms (e.g. using trees or linked lists)

```
//
//        Array of Element                         Array of floats
//           structures
//
//    ||===================||
//    ||    ............   ||                     ||===========||
//    ||    ............   ||<-----+              ||  .........||
//    ||===================||      |              ||   float   ||
//    ||  float* internal--||------+              ||   float   ||
//    ||  float* external--||-------------------->||   float   ||
//    ||  float value <----||------+              ||  .........||
//    ||===================||      |              ||  .........||
//    ||    ............   ||      |              ||   float   ||
//    ||    ............   ||      |              ||   float   ||
//    ||===================||      |              ||   float   ||
//    ||===================||      |              ||   float   ||
//    ||  float* internal--||------+              ||   float   ||
//    ||  float* external--||-------------------->||   float   ||
//    ||  float value      ||                     ||   float   ||
//    ||===================||                     ||   float   ||
//    ||    ............   ||                     ||  .........||
//    ||    ............   ||                     ||===========||
//    ||===================||
//
```

SVM buffer   SVM buffer

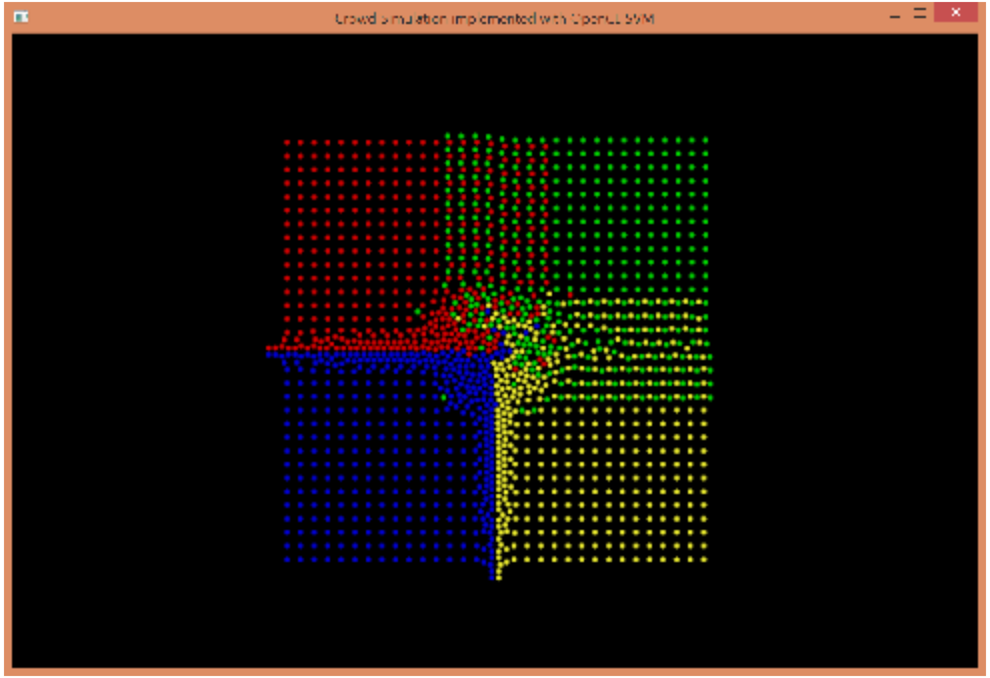
Crowd Simulation Example

What type of algorithm can actually benefit from SVM?



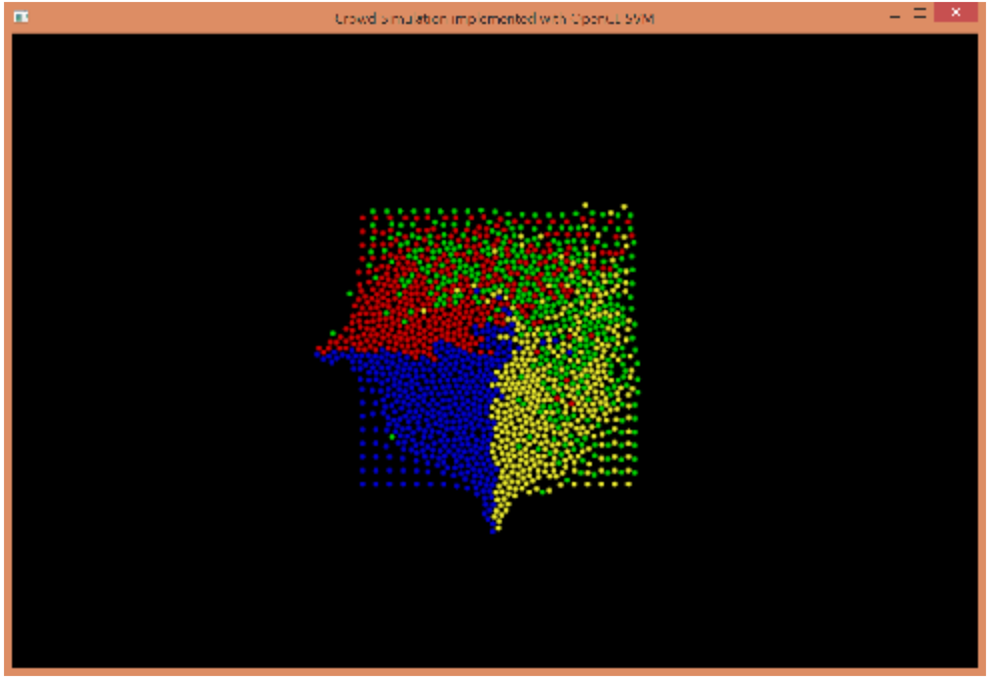
Crowd Simulation Example

What type of algorithm can actually benefit from SVM?



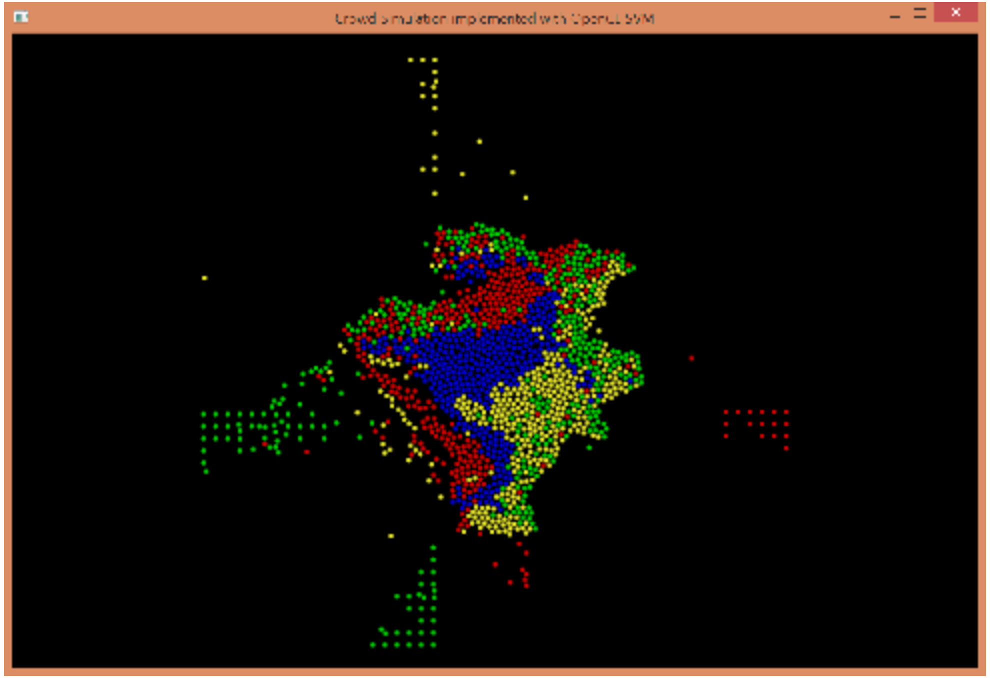
Crowd Simulation Example

What type of algorithm can actually benefit from SVM?



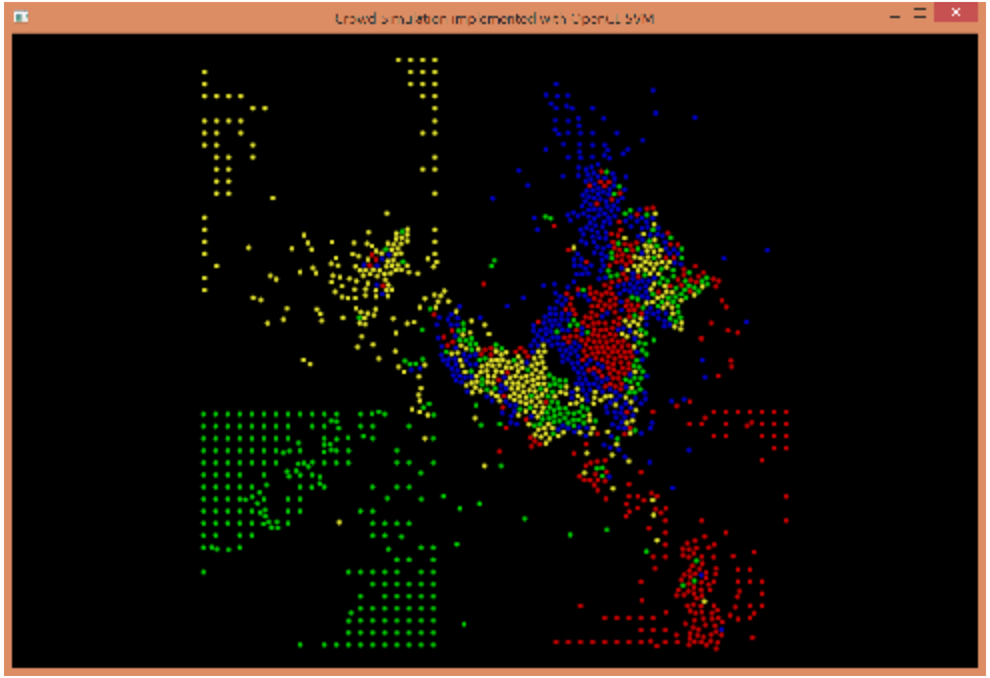
Crowd Simulation Example

What type of algorithm can actually benefit from SVM?



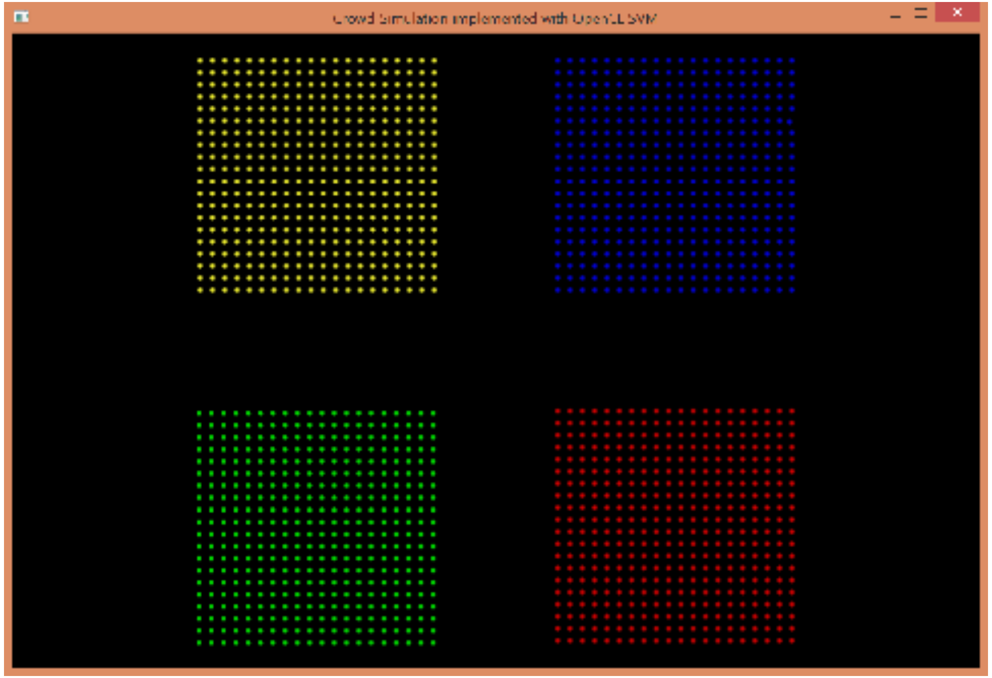
Crowd Simulation Example

What type of algorithm can actually benefit from SVM?



Crowd Simulation Example

What type of algorithm can actually benefit from SVM?



CrowdSim – SVM Illustration

RVOSimulator.cpp   oclobject.cpp   main.cpp   CrowdSim.cl*

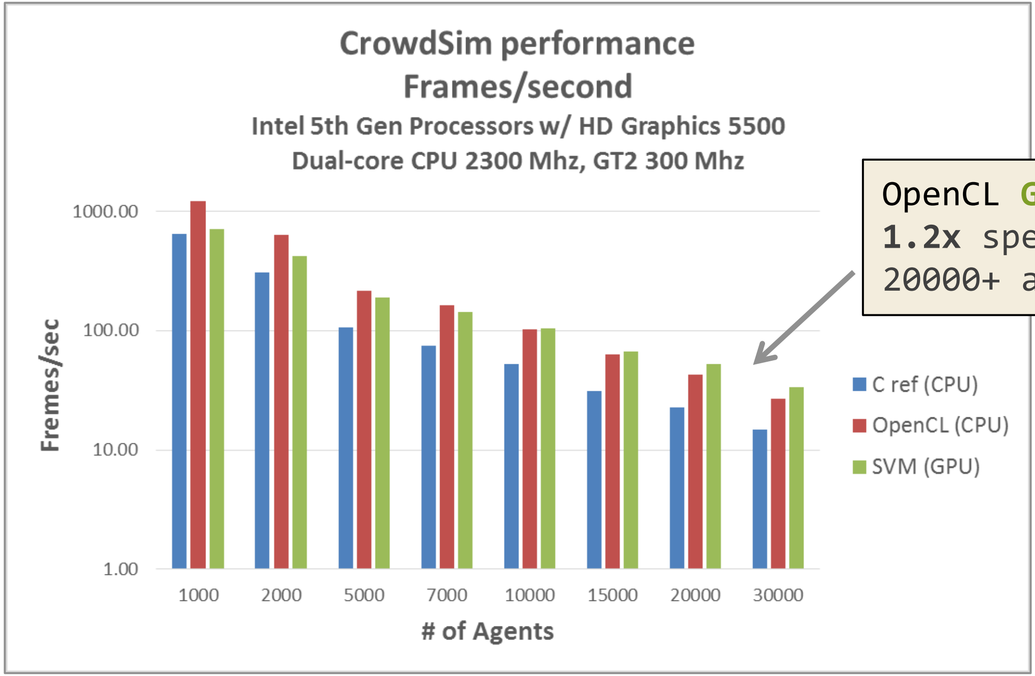
```
/* Create agent ORCA lines. */
for (uint i = 0; i < agent->numAgentNeighbors_; ++i) {
    const __global Agent *const other = agent->agentNeighbors_[i].second;
    const Vector2 relativePosition = other->position_ - position_;
    const Vector2 relativeVelocity = velocity_ - other->velocity_;
    const float distSq = absSq(relativePosition);
    const float combinedRadius = radius_ + other->radius_;
    const float combinedRadiusSq = sqr(combinedRadius);

    if (distSq > combinedRadiusSq) {
        /* No collision. */
        const Vector2 w = relativeVelocity - invTimeHorizon * relativePosition;
        /* Vector from cutoff center to relative velocity. */
        const float wLengthSq = absSq(w);

        const float dotProduct1 = dot(w, relativePosition);

        if (dotProduct1 < 0.0f && sqr(dotProduct1) > combinedRadiusSq * wLengthSq) {
            /* Project on cut-off circle. */
            const float wLength = sqrt(wLengthSq);
            const Vector2 unitW = w / wLength;

            line.direction = (Vector2)(unitW.y, -unitW.x);
            u = (combinedRadius * invTimeHorizon - wLength) * unitW;
        }
    }
    else {
        /* Collision. Project on cut-off circle of time timeStep. */
        const float invTimeStep = 1.0f / timeStep;

        /* Vector from cutoff center to relative velocity. */
        const Vector2 w = relativeVelocity - invTimeStep * relativePosition

        const float wLength = length(w);
        const Vector2 unitW = w / wLength;

        line.direction = (Vector2)(unitW.y, -unitW.x);
        u = (combinedRadius * invTimeStep - wLength) * unitW;
    }

    line.point = velocity_ + 0.5f * u;
    agent->orcaLines_[agent->numOrcaLines_++] = line;
}
```

'orcaLines_' – pointer to SVM

- ➢ Pointer contained in "Agent" struct
- ➢ No need for `clCreateBuffer`(), but must use `clSetKernelExecInfo`()

```
__kernel
void computeNewVelocity(__global PAgent* agents, …)
{
 __global Agent* agent =

agents[get_global_id(0)].value;
...

 agent->orcaLines_[agent->numOrcaLines_++] = line;
...
```

Crowd Simulation Performance Gains



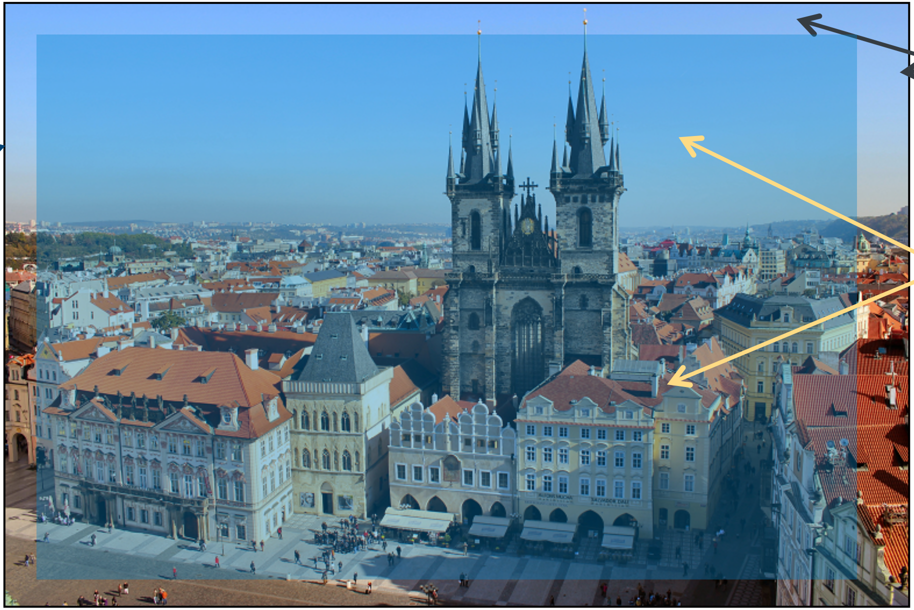
OpenCL GPU vs CPU
1.2x speedup for
20000+ agents

Further gains possible with heterogeneous mode, issuing OpenCL threads concurrently on GPU and CPU

SVM ISV usage – Cyberlink Photo Director

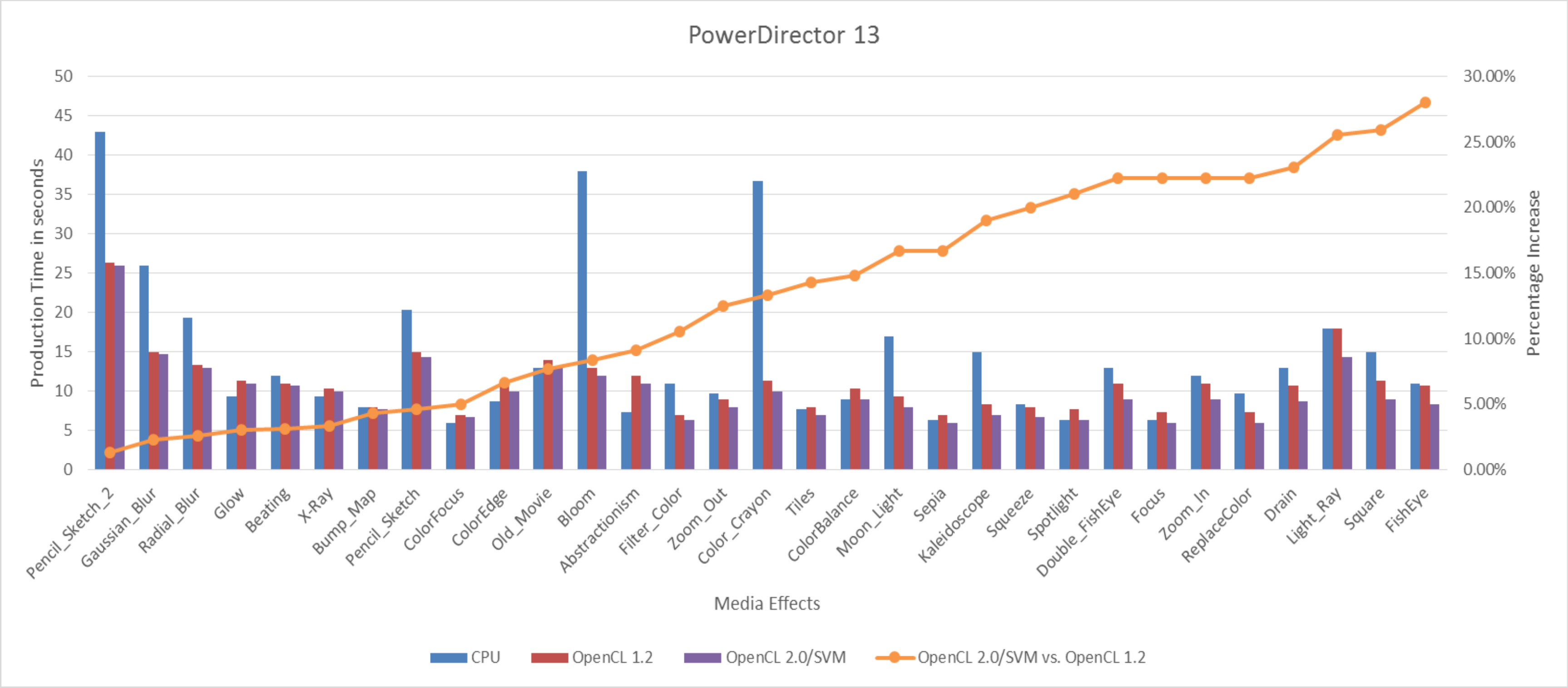
CPU/GPU border pixel processing strategy

- Remove border bounds-checks from GPU kernel, process border pixels on the CPU in parallel



5% gain vs. serial execution

Cyberlink PowerDirector 13 Effects Performance



* Permission to share Information from Cyberlink Corp.

SVM Summary

Coarse-grain SVM available on Intel 5th Generation Processors with HD Graphics 5300+

- Supports virtual-memory pointer access from GPU kernels
- No longer need to marshal buffers into ‘cl_mem’ constructs
- No alignment or size restrictions to achieve zero-copy buffer sharing

Fine-grain SVM available in Intel 5th Generation Processors w/ HD Graphics 5500+

SVM samples available on Intel® Developer Zone

- *SVM Basic* sample
- *CrowdSim* coming soon!

Part II:

Device Side Enqueue and Work-group Scan Functions in OpenCL 2.0

Device Side Enqueue

Device kernels can enqueue kernels to the same device with no host interaction, enabling flexible work scheduling paradigms and avoiding the need to transfer execution control and data between the device and host, often significantly offloading host processor bottlenecks*

Introduced to OpenCL 2.0 to express recursive and iterative algorithms

We are going to use Sierpiński Carpet as a simple example to show all the building blocks of the device side enqueue

***Khronos Finalizes OpenCL 2.0 Specification for Heterogeneous Computing**
https://www.khronos.org/news/press/khronos-finalizes-opencl-2.0-specification-for-heterogeneous-computing

Sierpiński Carpet

The **Sierpiński carpet** is a plane fractal first described by Wacław Sierpiński in 1916.

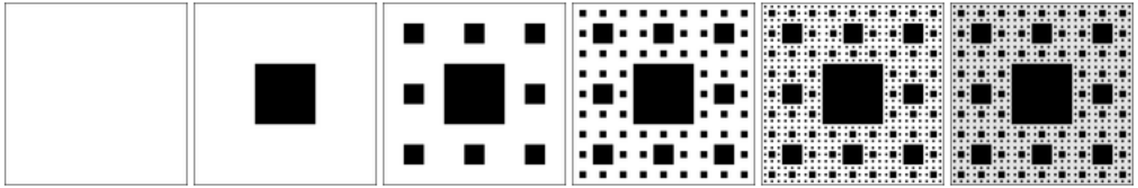
Start with a white square.

Divide the square into 9 sub-squares in a 3-by-3 grid

Paint the central sub-square black.

Apply the same procedure recursively to the remaining 8 sub-squares

And so on …



See http://en.wikipedia.org/wiki/Sierpinski_carpet for more info

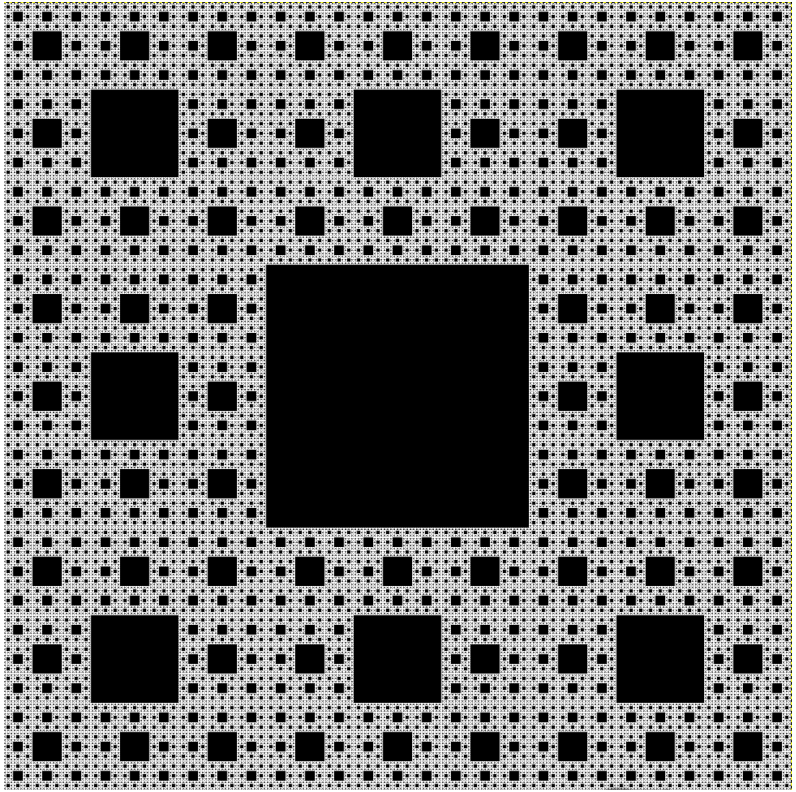
* Sierpiński Carpet image sequence above from http://en.wikipedia.org/wiki/Sierpinski_carpet

Sierpiński Carpet Kernel in OpenCL 2.0

```
__kernel void sierpinski(__global char* src, int width, int offsetx, int offsety)
{
        int x = get_global_id(0);
        int y = get_global_id(1);
        queue_t q = get_default_queue();

        int one_third = get_global_size(0) / 3;
        int two_thirds = 2 * one_third;

        if (x >= one_third && x < two_thirds && y >= one_third && y < two_thirds)
        {
                src[(y+offsety)*width+(x+offsetx)] = BLACK;
        }
        else
        {
                src[(y+offsety)*width+(x+offsetx)] = WHITE;

                if (one_third > 1 && x % one_third == 0 && y % one_third == 0)
                {
                        const size_t  grid[2] = {one_third, one_third};
                        enqueue_kernel(q, 0, ndrange_2D(grid), ^{ sierpinski(src, width, x+offsetx, y+offsety); });
                }
        }
}
```

Easy to translate recursive algorithm to implementation

Sierpiński Carpet - Result



2187x2187 image: 8↑6 = 299592 enqueue_kernel calls!

How Recursive Version Compares to Iterative?

Recursive: 2050 ms ☹

Iterative: 11 ms

Solution: combine the two, start w/ recursive, switch to iterative for 243x243 tiles

Mixed version: 10.45 ms ☺ - speedup of 1.05X for 2187 by 2187 image

Speedup improves for larger image sizes: 1.15X for 6561 by 6561 image

1.23X for 19683 by 19683 image

`enqueue_kernel` will improve performance when used properly!

GPU-Quicksort - Overview

Invented by Daniel Cederman and Phillipas Tsigas

- Student and Professor pair
- At Chalmers University of Technology
- Invented in 2007, written in CUDA
- Improves on the work of Shubhabrata Sengupta, Mark Harris, Yao Zhang, and John D. Owens

First phase:

- workgroups work on different parts of the same sequence
- Each workgroup partitions a block assigned to it around the pivot
- The partitioned blocks are merged
- The last workgroup writes one or more pivot values between the sequences
- We repeat the first phase until each subsequence is short enough to be sorted by one workgroup

Second phase:

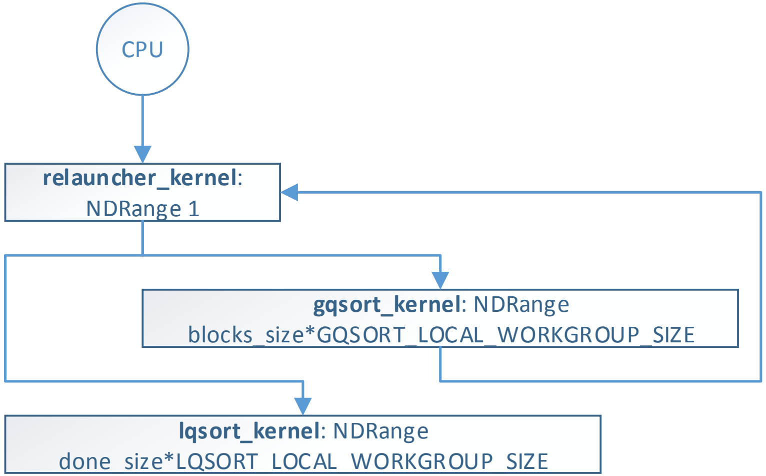
- Each workgroup is assigned its own subsequence of <= QUICKSORT_BLOCK_SIZE (e.g. 1536) elements
- Use explicit work stack to simulate quicksort recursive calls within the kernel
- Use Bitonic sort when the number of elements in subsequence is <= SORT_THRESHOLD (e.g. 512)

* Photos of Daniel Cederman and Prof. Phillipas Tsigas from their research group's home page

GPU-Quicksort in OpenCL 2.0

Switch to work group scan functions

- **work_group_scan_exclusive_add** in gqsort_kernel
- **work_group_scan_exclusive_add** and **work_group_scan_inclusive_add** in lqsort_kernel
- Performance gain of **8%** compared to Blelloch algorithm in 1.2
- Gain in code conciseness, maintainability and clarity
  - code size reduced almost **3X**



Take advantage of `enqueue_kernel` function

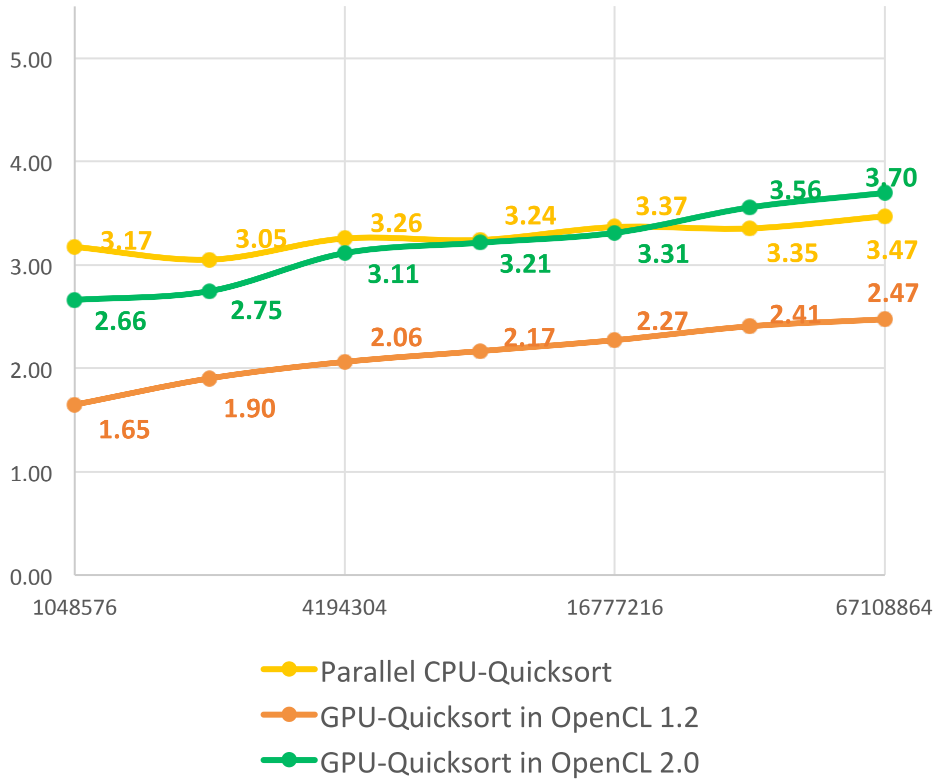
- Move the logic that calculates block and parent records, sorts the records after each gqsort_kernel run, and launches either gqsort_kernel or lqsort_kernel to GPU
- Dramatically simplify launch from CPU: just **relauncher_kernel**
- Simplified gqsort_kernel and lqsort_kernel CPU wrapper code: we use **blocks** on GPU

```
if (work_size != 0) {
    // Calculate block size, parents and blocks
    // ...

    enqueue_kernel(q, CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
        ndrange_1D(GQSORT_LOCAL_THREADCOUNT * blocks_size, GQSORT_LOCAL_THREADCOUNT),
        ^{ gqsort_kernel(d, dn, blocks, parents, result, work, done, done_size, MAXSEQ, 0); });
} else {
    enqueue_kernel(q, CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
        ndrange_1D(LQSORT_LOCAL_THREADCOUNT * done_size, LQSORT_LOCAL_THREADCOUNT),
        ^{ lqsort_kernel(d, dn, done); });
}
```

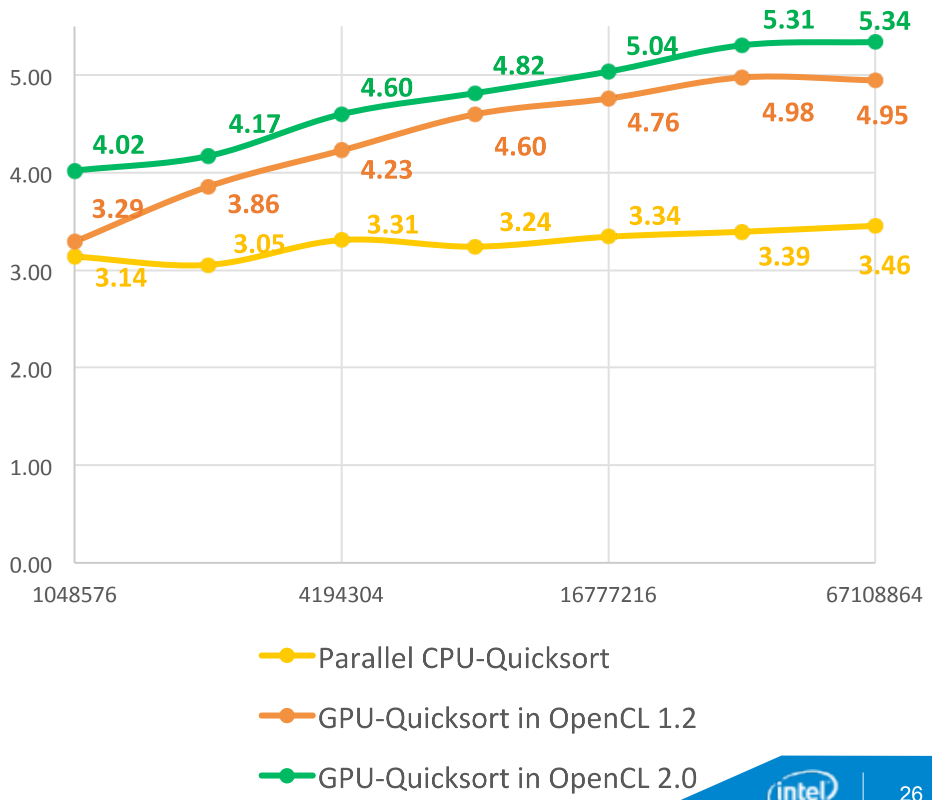
GPU-Quicksort is 44% to 62% faster on Intel HD Graphics 5500 when implemented in OpenCL 2.0 vs OpenCL 1.2!

GPU-Quicksort in OpenCL 2.0 Performance

HD Graphics 5500: Speedup vs std::sort
X axis - input size, Y axis - speedup



HD Graphics 6000: Speedup vs std::sort
X axis - input size, Y axis - speedup



Conclusions

Device side enqueue is a powerful addition to the OpenCL programmer toolbox

- Provides the ability to port recursive and iterative algorithms to OpenCL
- Might dramatically improve performance when used properly
- Syntactically comparable to CUDA's dynamic parallelism feature

`enqueue_kernel` and more available in Intel's OpenCL 2.0 driver

GPU-Quicksort for OpenCL 1.2 runs very well on Intel® Processor Graphics

GPU-Quicksort for OpenCL 2.0 runs even better

- Due to optimized work group scan functions
- Due to enqueue_kernel functions, which avoid round trips to the CPU

Intel HD Graphics 5500 with Intel's OpenCL 2.0 driver is a powerful platform for writing high performance algorithms!

Key Takeaways

SVM

- Use it for all your shared pointy data structure needs
- Coarse grain and Fine grain flavors w/ atomics are available on Intel's 5th Generation Processors

Device side enqueue

- Use it to implement and port high-performance recursive and iterative algorithms
- Avoid round-trips to the host

New work group functions

- Simplify you code for scan, reduce and other common group ops
- Use high-performance implementations optimized for Intel hardware

GPU-Quicksort Bibliography

"My Early Days at Elliots" by *Tony Hoare* http://www.cs.man.ac.uk/CCS/res/res48.htm

Partition by C.A.R.Hoare. Communications of the ACM, Volume 4 Issue 7, July 1961 http://dl.acm.org/citation.cfm?doid=366622.366642

Quicksort by C.A.R.Hoare. Communications of the ACM, Volume 4 Issue 7, July 1961 http://dl.acm.org/citation.cfm?doid=366622.366644

"Implementing Quicksort programs" by Robert Sedgewick. Communications of the ACM, Volume 21 Issue 10, Oct. 1978 http://dl.acm.org/citation.cfm?doid=359619.359631

Sorting algorithms/Quicksort http://rosettacode.org/wiki/Sorting_algorithms/Quicksort

"Scan Primitives for GPU Computing" by Shubhabrata Sengupta, Mark Harris, Yao Zhang, and John D. Owens, Graphics Hardware 2007, pages 97--106, August 2007. http://www.idav.ucdavis.edu/publications/print_pub?pub_id=915

Cederman, D. and Tsigas, P. 2009. GPU-Quicksort: A practical Quicksort algorithm for graphics processors. ACM J. Exp. Algor. 14, Article 1.4 (July 2009), 24 pages http://dl.acm.org/citation.cfm?id=1564500

Quicksort at Wikipedia, http://en.wikipedia.org/wiki/Quicksort

GPU-Quicksort in OpenCL 2.0, https://software.intel.com/en-us/articles/gpu-quicksort-in-opencl-20-using-nested-parallelism-and-work-group-scan-functions

Sierpiński Carpet in OpenCL 2.0, https://software.intel.com/en-us/articles/sierpinski-carpet-in-opencl-20