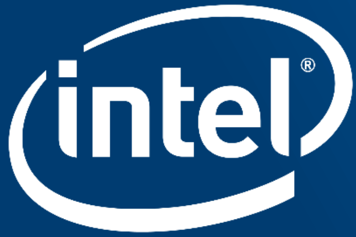


# Debugging And Optimizing OpenCL* Applications

## Best Practices and Tools

Yuval Eshkol (yuval.eshkol@intel.com)

# Legal Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information. The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

- Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, 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.
- All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.
- All products, platforms, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice. All dates specified are target dates, are provided for planning purposes only and are subject to change.
- This document contains information on products in the design phase of development. Do not finalize a design with this information. Revised information will be published when the product is available. Verify with your local sales office that you have the latest datasheet before finalizing a design.
- Code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.
- Intel, Intel Inside, Intel Atom and Intel Core are trademarks of Intel Corporation in the U.S. and other countries.
- Other names and brands may be claimed as the property of others.
- Copyright © 2015-2016, Intel Corporation. All rights reserved.
- OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos



Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Objectives

- Understand the architecture characteristics relevant to compute applications on Intel® Processor Graphics
- Learn techniques for optimizing OpenCL* applications for Intel® Processor Graphics
- Introduce with Intel® tools for development, debugging and optimizing OpenCL* applications

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Agenda

- Intel® Processor Graphics introduction
- Optimization techniques for OpenCL* applications
- Develop OpenCL* applications with Intel® SDK for OpenCL™ Applications
- Debug  OpenCL* applications with Intel® SDK for OpenCL™ Applications
- Optimize OpenCL* application with Intel tools
  - Intel® VTune™ Amplifier XE
  - Intel® SDK for OpenCL* Applications

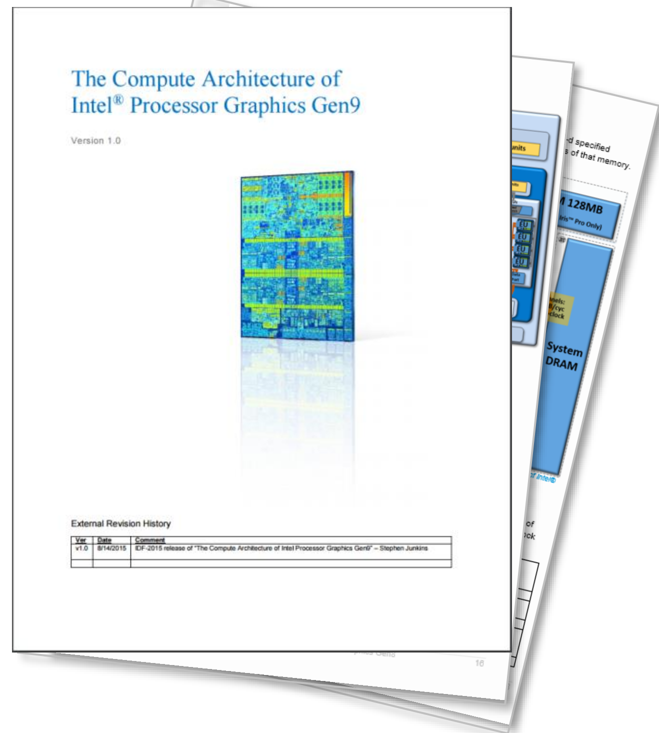

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Processor Graphics

## Introduction

# Intel® Processor Graphics Architecture

- Today, our focus is on Intel® Iris™ Graphics and Intel® HD Graphics in 6th Generation Intel® Core™ Processors
  - Or, Intel Processor Graphics Gen9
- For more details, see our whitepaper, The Compute Architecture of Intel Processor Graphics Gen7.5/Gen8.0/Gen9.0
- https://software.intel.com/en-us/articles/intel-graphics-developers-guides





Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

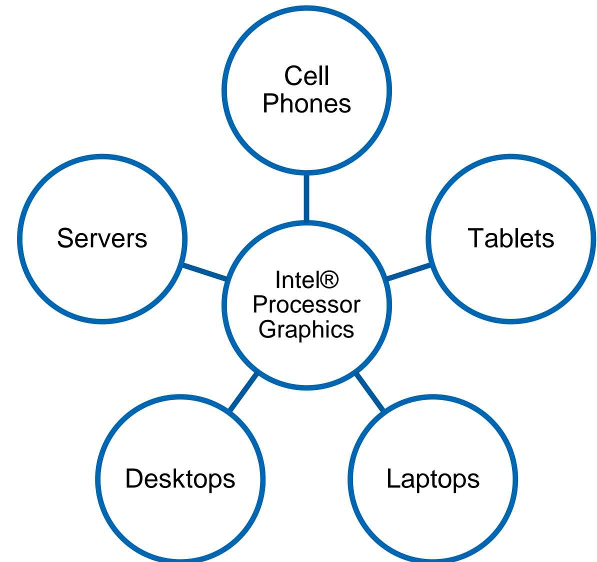
# Intel® Processor Graphics Architecture

- Outstanding rendering and media performance
- High-throughput general purpose compute capabilities
- High bandwidth memory hierarchy
- Deep integration with on-die CPUs and other SoC devices

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Processor Graphics Architecture

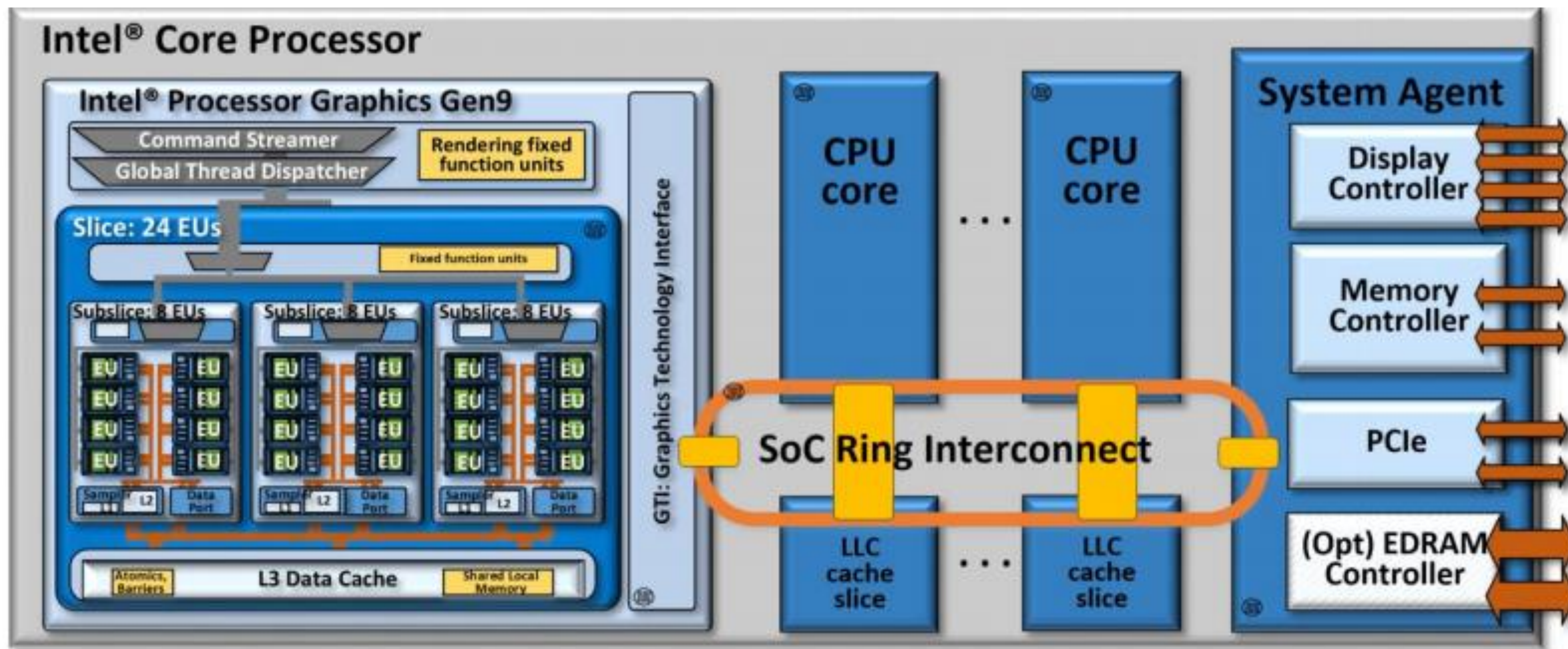
- Modular architecture
- Scalability for a range of products





Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# GEN9 Core Processor

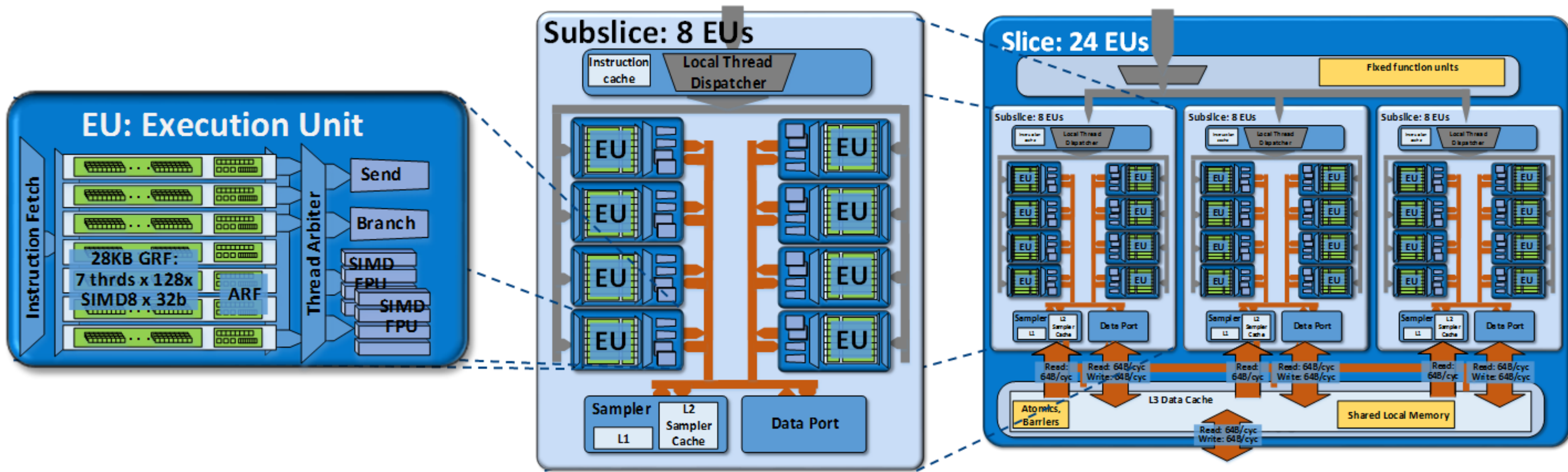



*An Intel® Core™ i7 processor 6700K SoC and its ring interconnect architecture.*

Optimization Notice


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Graphics Architecture Building Blocks




- Modular architecture, which enables scalability across a wide range of target products

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Graphics Architecture Building Blocks

A potential product design composed of three slices, each with three sub-slices, for a total of 72 EUs

Intel® Processor Graphics Gen9

Command Streamer

Global Thread Dispatcher

Rendering fixed function units

Slice: 24 EUs

Fixed function units

Subslice: 8 EUs

L3 Data Cache

Atomics, Barriers

Shared Local Memory

L3 fabric

Each Slice: Read: 64B/cyc Write: 64B/cyc

GTI: Graphics Technology Interface

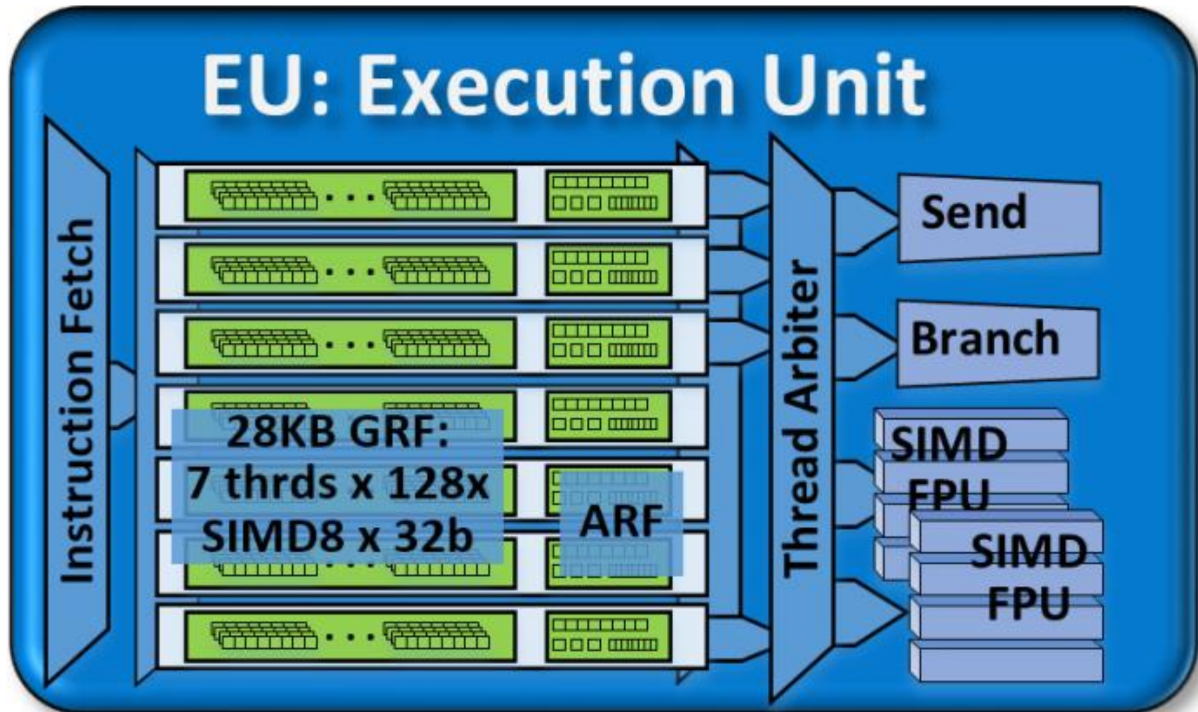
Read: 64B/cyc Write: 64B/cyc

*Intel® Core™ i7-6970HQ Processor with Intel® Iris™ Pro Graphics 580*


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Graphics Architecture Building Blocks - EU

- 7 threads with
  - 128 GRF of 32 bytes, Accessible as SIMD-8 32-bit
- Can co-issue up to 4 instruction processing units including:
  - 2 FPUs
  - Branch unit
  - Message send unit

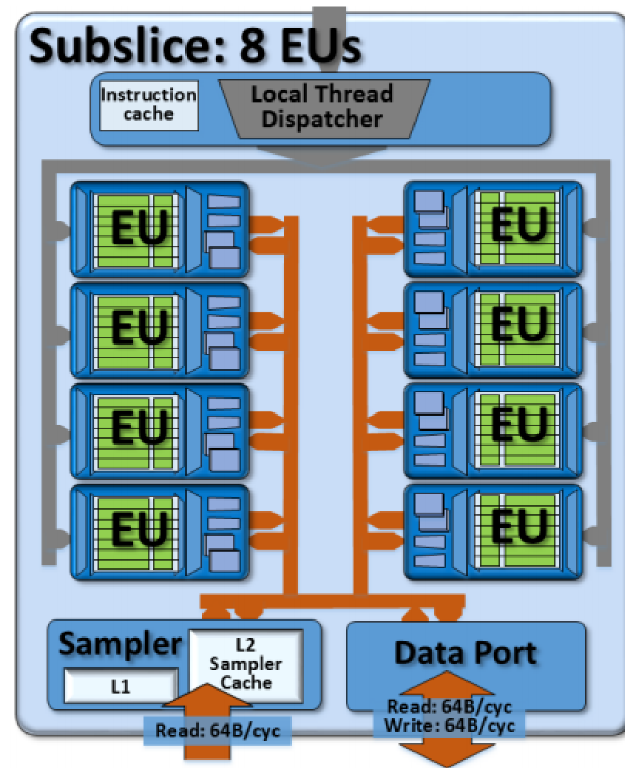




Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Graphics Architecture Building Blocks - Subslice

- 8 EUs (can be changed for scalability) X 7 threads
  - Dedicated hardware resources and register files for 56 simultaneous threads
- Local thread dispatcher unit
- Supporting instruction caches
- Sampler
  - Read-only memory fetch unit includes 2-level caches
- Data port
  - A memory load/store unit

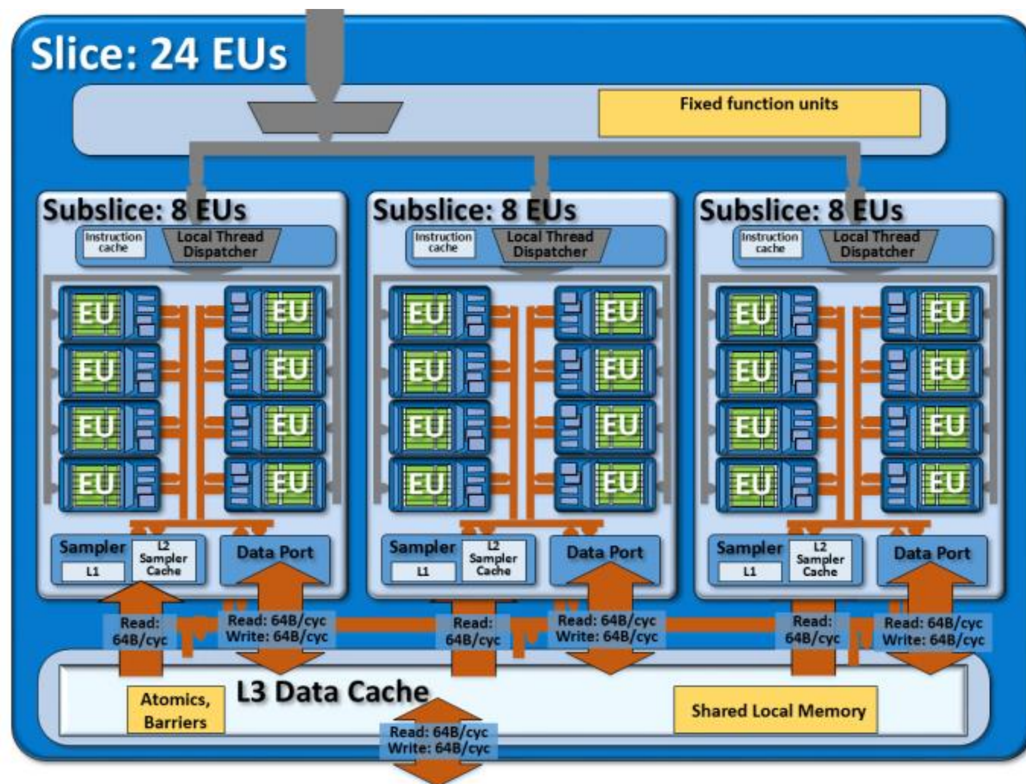




Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Graphics Architecture Building Blocks - Slice

- 3 sub-slices for a total of 24 EUs



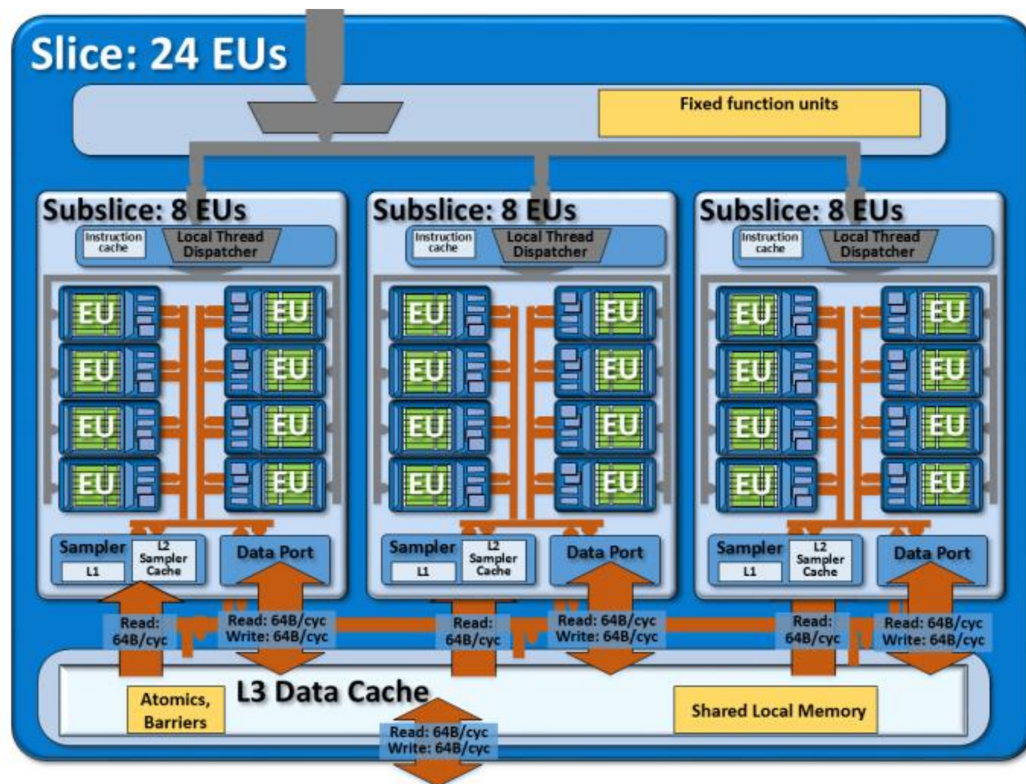


Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Graphics Architecture Building Blocks - Slice

- Banked L3 cache
  - Cachelines are 64 bytes each
- Smaller highly-banked shared local memory
  - For sharing among EU hardware threads within the same subslice
  - Same latency as L3 data cache
  - Can yield full bandwidth for access patterns that may not be 64-byte aligned

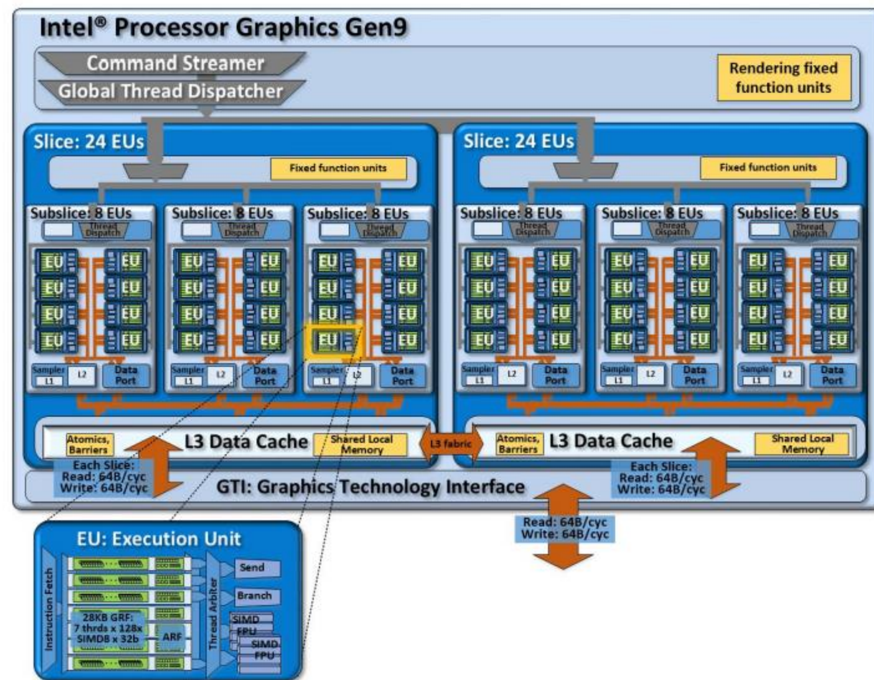



Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Graphics Architecture Building Blocks - Product

- SoC product instantiates a single slice or groups of slices.
- Additional front end logic
  - Manage command submission
- Fixed-function logic
  - Support 3D rendering, and media pipelines
- Graphics technology interface (GTI)
  - Interfaces to the rest of the SoC components (memory, CPU, …)

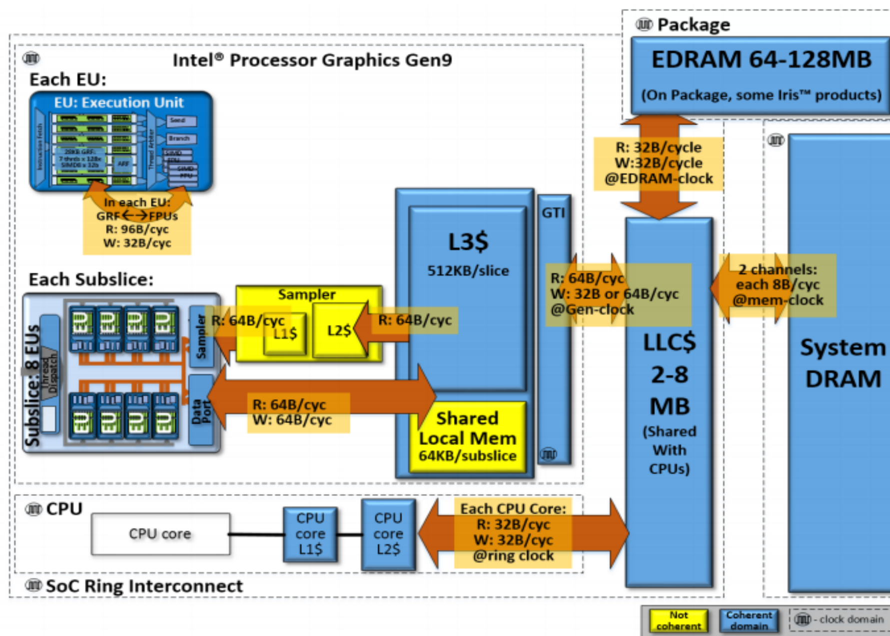



*The Intel® Core™ i7 processor 6700K with Intel® HD Graphics 530.*


Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Graphics Architecture - Memory

- Share DRAM physical memory with the CPU
  - Zero copy
- Shared memory coherency and consistency




*SoC chip level memory hierarchy and its theoretical peak bandwidths for Intel processor graphics gen9.*


Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

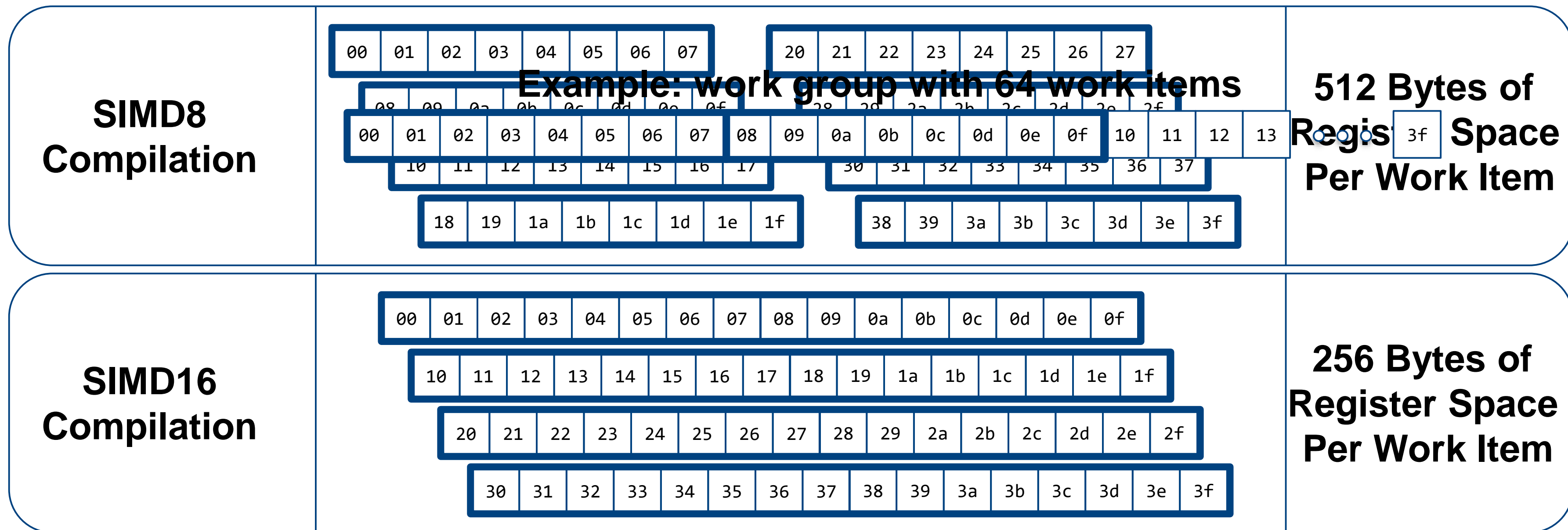
# How OpenCL* Maps To Intel® Processor Graphics

# Executing OpenCL* Kernels

- OpenCL* work items map to SIMD lanes of a hardware thread
- Compiler may decide to compile a kernel SIMD32, SIMD16, or SIMD8
  - A compiler heuristic will choose a SIMD width that best maximizes register footprint within a single hardware thread and avoid needs for register spill/fill.
  - Typically short kernels that need less than 128 bytes of private memory will compile to SIMD32

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Executing OpenCL* Kernels

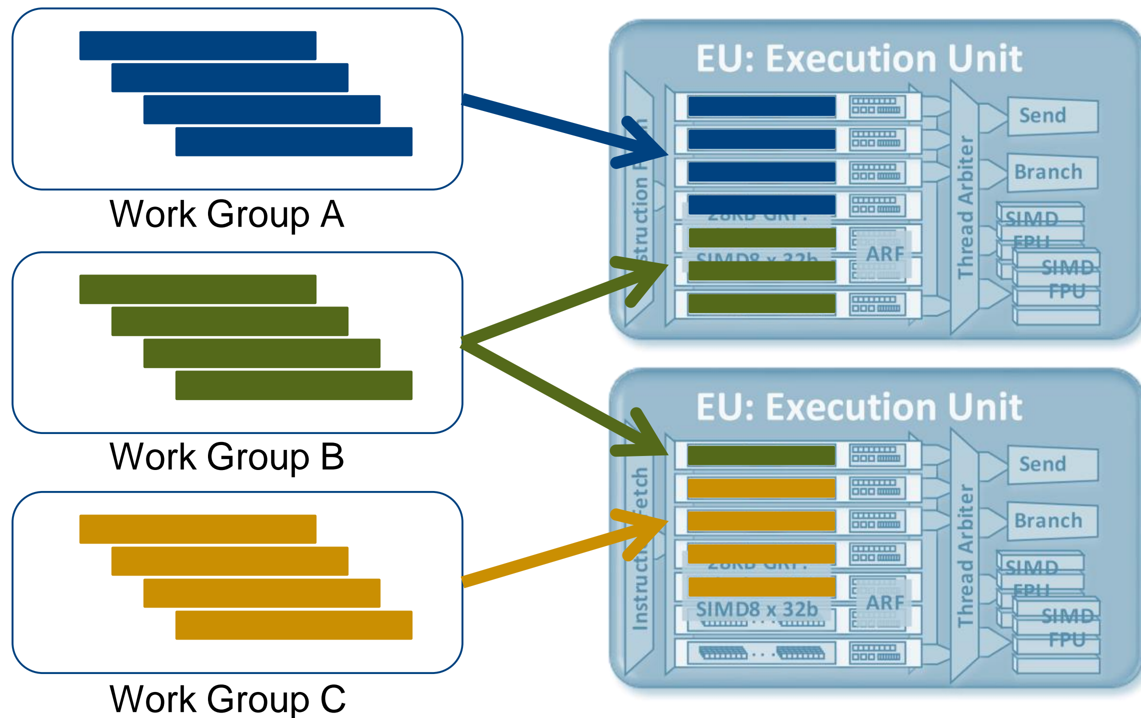



***Compiler can trade register space for IPC!***

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Executing OpenCL* Kernels

- Example: SIMD16 compile, 64 work items per work group




***Workgroups may span EU threads!***

***Workgroups may span EUs!***

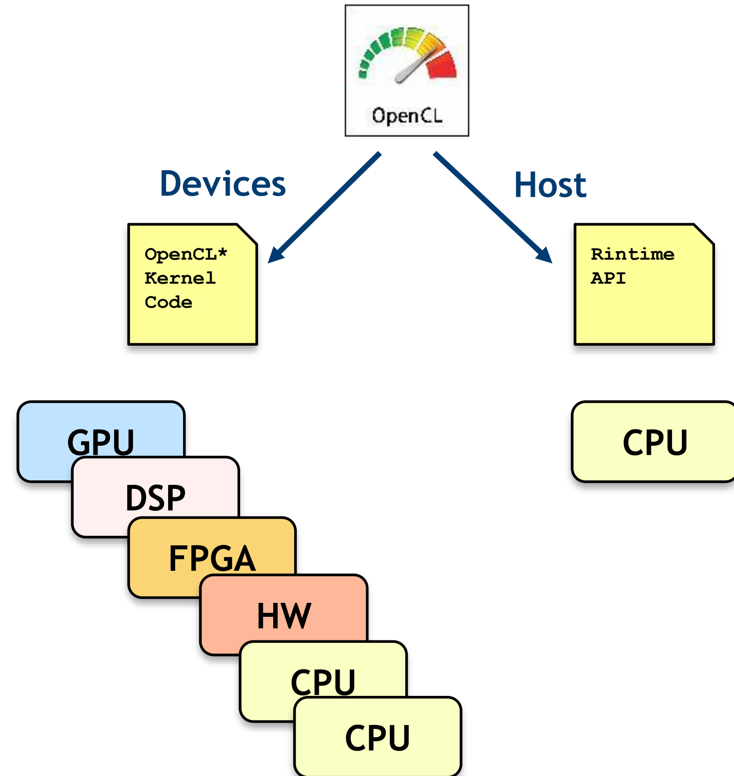

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Optimize OpenCL* applications

Best Practices

# Two Levels of Optimizations for OpenCL*

- Application level optimization
  - Optimization is ~vendor agnostic, tools are ~similar
  - Many API level tricks are ~portable
- Kernel level optimization side
  - Entirely vendor-specific (and so are tools)
  - Kernels optimizations are generally less portable





Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Optimization Factors

- Optimize host API calls
- Reduce host <> device memory traffic and bandwidth
- Optimizing memory access
- Maximizing occupancy and computation
- Kernel algorithm optimization

OpenCL* Developer Guide for Intel® Processor Graphics:

**https://software.intel.com/en-us/iocl_opg**

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Optimization Factors

- **Optimize host API calls**
- Reduce Host <> Device memory traffic and bandwidth
- Optimizing memory access
- Maximizing occupancy
- Maximizing computation
- Kernel algorithm optimization

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intro to Host-Side API Optimization

- "Wall-clock" time: wrap OpenCL* API calls with timestamps + printfs
  - Most "device" OpenCL* APIs (like clEnqueueNDRangeKernel) just put a call into a queue and immediately return
    - To measure actual execution time, need to synchronize on completion
- Better solution with profiling events
  - OpenCL* profiling info from events associated with all queued commands:
    - Time spent in the command queue, driver and actual hardware exec
  - Enable queue for profiling with CL_QUEUE_PROFILING_ENABLE
  - Wait for the command completion before querying the event stats
- Best solution
  - Use Intel® Code Builder to get API profiling report and optimization tips

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code Builder ANALYZE

# Avoid Redundant Usage of API Calls

Regular application (c/c++)

```
main ()
{
    foo();
    …
    foo();
    …
    …
    bar();
    …
    …
    bar();
    …
    foo();
}
```

```
foo ()
{
    for (…)
    {
        …
    }
}

bar()
{
    for (…)
    {
        …
    }
}
```

OpenCL Application

```
// New OpenCL implementation
foo ()
{
    // OpenCL initialization
    // Run kernel
    …
    // Release OpenCL objects
}

bar()
{
    // OpenCL initialization
    // Run kernel
    …
    // Release OpenCL objects
}
```

clGetPlatformIDs
clGetDeviceIDs
clCreateContextFromType
clCreateCommandQueue
clCreateProgramWithSource
clBuildProgram
clCreateBuffer

clCreateKernel
SetKernelArguments
clEnqueueNDRangeKernel

clReleaseMemObject
clReleaseProgram
clReleaseCommandQueue
clReleaseContext
clReleaseDevice

OpenCL Application (optimized)

```
// New OpenCL implementation
// OpenCL initialization
foo ()
{
    // Run kernel
    …
}

bar()
{
    // Run kernel
    …
}
// Release OpenCL objects
```


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code snippets provided in this presentation are for illustrative purposes only. Intel disclaims any and all implied or express warranties associated with the code snippets, and any and all use of such code snippets is at the sole discretion and exclusive risk of the user.

# Reusing Compilation Results

- Reusing compilation results is typically faster than recreating the program from the source
  - Check it for your specific program and device
- Cache the resulting binaries after the first OpenCL* compilation and reuse them by calling clCreateProgramWithBinary
  - To retrieve binaries generated from CreateProgramWithSource and clBuildProgram:
    - Call clGetProgramInfo with the CL_PROGRAM_BINARIES parameter
- A better way – Pre compile your program offline with Intel® Code Builder for OpenCL™ API and save intermediate binaries


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Code Builder API Calls Report

Code Builder ANALYZE

Session Info | Host Profiling | Kernels Overview

Api Calls: [ Data Table ] Graphical View

| Api Name | Count | # Errors | Total Duration (µs) | Avg Duration (µs) | | |
|---|---|---|---|---|---|---|
| + clBuildProgram | 5 | 0 | 1800529.69 | 360105.938 | | |
| + clCreateBuffer | 15 | 0 | 605422.278 | 40361.485 | | |
| + clCreateCommandQueue | 5 | 0 | 42899.852 | 8579.97 | | |
| + clCreateContextFromType | 5 | 0 | 73049.299 | 14609.86 | | |
| + clCreateKernel | 5 | 0 | 1173.758 | 234.752 | | |
| + clCreateProgramWithSource | 5 | 0 | 491.837 | 98.367 | | |
| + clEnqueueNDRangeKernel | 5 | 0 | 560.809 | 112.162 | 94.837 | 124.396 |
| + clEnqueueReadBuffer | 5 | 0 | 175029.128 | 35005.826 | 30261.938 | 43323.538 |
| + clFinish | 10 | 0 | 308955.786 | 30895.579 | 0.821 | 123338.613 |
| + clGetDeviceIDs | 5 | 0 | 10.264 | 2.053 | 1.232 | 2.874 |
| + clGetPlatformIDs | 2 | 0 | 47.213 | 23.607 | 0.411 | 46.803 |
| + clGetPlatformInfo | 10 | 0 | 16.422 | 1.642 | 0.411 | 5.337 |
| + clReleaseCommandQueue | 5 | 0 | 19446.86 | 3889.372 | 348.556 | 9850.7 |
| + clReleaseContext | 5 | 0 | 15139.796 | 3027.959 | 2651.322 | 3382.919 |
| + clReleaseDevice | 5 | 0 | 15.19 | 3.038 | 2.053 | 3.695 |
| + clReleaseKernel | 5 | 0 | 27.917 | 5.583 | 4.927 | 6.569 |
| + clReleaseMemObject | 15 | 0 | 256074.68 | 17071.645 | 14939.038 | 22415.946 |

4

**inefficient "clCreateBuffer" calls.**
The host program includes 10 calls to "clCreateBuffer" where "flags" includes "CL_MEM_COPY_HOST_PTR".

**4 redundant calls to "clCreateContextFromType".**
The host program includes 5 calls to "clCreateContextFromType" with the same arguments.

**4 redundant calls to "clCreateCommandQueue".**
The host program includes 5 calls to "clCreateCommandQueue" that refer to the same device: "Device [1] (Intel(R) HD Graphics 4400)".

**"clEnqueueReadBuffer" calls.**
The host program includes 5 calls to "clEnqueueReadBuffer".


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Code Builder API Calls Report

Session Info | Host Profiling | Kernels Overview

Api Calls: [ Data Table ] Graphical View

| Api Name | Count | # Errors | Total Duration (µs) | Avg Duration (µs) | | |
|---|---|---|---|---|---|---|
| + clBuildProgram | 5 | 0 | 1800529.69 | 360105.938 | | |
| + clCreateBuffer | 15 | 0 | 605422.278 | 40361.485 | | |
| clCreateCommandQueue | 5 | 0 | 42899.852 | 8579.97 | | |
| clCreateContextFromType | 5 | 0 | 73049.299 | 14609.86 | | |
| + clCreateKernel | 5 | 0 | 1173.758 | 234.752 | | |
| + clCreateProgramWithSource | 5 | 0 | 491.837 | 98.367 | | |
| + clEnqueueNDRangeKernel | 5 | 0 | 560.809 | 112.162 | 94.837 | 124.396 |
| + clEnqueueReadBuffer | 5 | 0 | 175029.128 | 35005.826 | 30261.938 | 43323.538 |
| + clFinish | 10 | 0 | 308955.786 | 30895.579 | 0.821 | 123338.613 |
| + clGetDeviceIDs | 5 | 0 | 10.264 | 2.053 | 1.232 | 2.874 |
| + clGetPlatformIDs | 2 | 0 | 47.213 | 23.607 | 0.411 | 46.803 |
| + clGetPlatformInfo | 10 | 0 | 16.422 | 1.642 | 0.411 | 5.337 |
| + clReleaseCommandQueue | 5 | 0 | 19446.86 | 3889.372 | 348.556 | 9850.7 |
| + clReleaseContext | 5 | 0 | 15139.796 | 3027.959 | 2651.322 | 3382.919 |
| + clReleaseDevice | 5 | 0 | 15.19 | 3.038 | 2.053 | 3.695 |
| + clReleaseKernel | 5 | 0 | 27.917 | 5.583 | 4.927 | 6.569 |
| + clReleaseMemObject | 15 | 0 | 256074.68 | 17071.645 | 14939.038 | 22415.946 |

**inefficient "clCreateBuffer" calls.**
The host program includes 10 calls to "clCreateBuffer" where "flags" includes "CL_MEM_COPY_HOST_PTR".

**4 redundant calls to "clCreateContextFromType".**
The host program includes 5 calls to "clCreateContextFromType" with the same arguments.

**4 redundant calls to "clCreateCommandQueue".**
The host program includes 5 calls to "clCreateCommandQueue" that refer to the same device: "Device [1] (Intel(R) HD Graphics 4400)".

**"clEnqueueReadBuffer" calls.**
The host program includes 5 calls to "clEnqueueReadBuffer".

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Code Builder API Calls Report

Session Info | Host Profiling | Kernels Overview

Api Calls: [ Data Table ] Graphical View

| Api Name | Count | # Errors | Total Duration (µs) | Avg Duration (µs) | | |
|---|---|---|---|---|---|---|
| + clBuildProgram | 5 | 0 | 1800529.69 | 360105.938 | | |
| + clCreateBuffer | 15 | 0 | 605422.278 | 40361.485 | | |
| clCreateCommandQueue | 5 | 0 | 42899.852 | 8579.97 | | |
| clCreateContextFromType | 5 | 0 | 73049.299 | 14609.86 | | |
| + clCreateKernel | 5 | 0 | 1173.758 | 234.752 | | |
| + clCreateProgramWithSource | 5 | 0 | 491.837 | 98.367 | | |
| + clEnqueueNDRangeKernel | 5 | 0 | 560.809 | 112.162 | | |
| + clEnqueueReadBuffer | 5 | 0 | 175029.128 | 35005.826 | 30261.938 | 43323.538 |
| + clFinish | 10 | 0 | 308955.786 | 30895.579 | 0.821 | 123338.613 |
| + clGetDeviceIDs | 5 | 0 | 10.264 | 2.053 | 1.232 | 2.874 |
| + clGetPlatformIDs | 2 | 0 | 47.213 | 23.607 | 0.411 | 46.803 |
| + clGetPlatformInfo | 10 | 0 | 16.422 | 1.642 | 0.411 | 5.337 |
| + clReleaseCommandQueue | 5 | 0 | 19446.86 | 3889.372 | 348.556 | 9850.7 |
| + clReleaseContext | 5 | 0 | 15139.796 | 3027.959 | 2651.322 | 3382.919 |
| + clReleaseDevice | 5 | 0 | 15.19 | 3.038 | 2.053 | 3.695 |
| + clReleaseKernel | 5 | 0 | 27.917 | 5.583 | 4.927 | 6.569 |
| + clReleaseMemObject | 15 | 0 | 256074.68 | 17071.645 | 14939.038 | 22415.946 |

4

**inefficient "clCreateBuffer" calls.**
The host program includes 10 calls to "clCreateBuffer" where "flags" includes "CL_MEM_COPY_HOST_PTR".

**4 redundant calls to "clCreateContextFromType".**
The host program includes 5 calls to "clCreateContextFromType" with the same arguments.

**4 redundant calls to "clCreateCommandQueue".**
The host program includes 5 calls to "clCreateCommandQueue" that refer to the same device: "Device [1] (Intel(R) HD Graphics 4400)".

**"clEnqueueReadBuffer" calls.**
The host program includes 5 calls to "clEnqueueReadBuffer".

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Optimization Factors

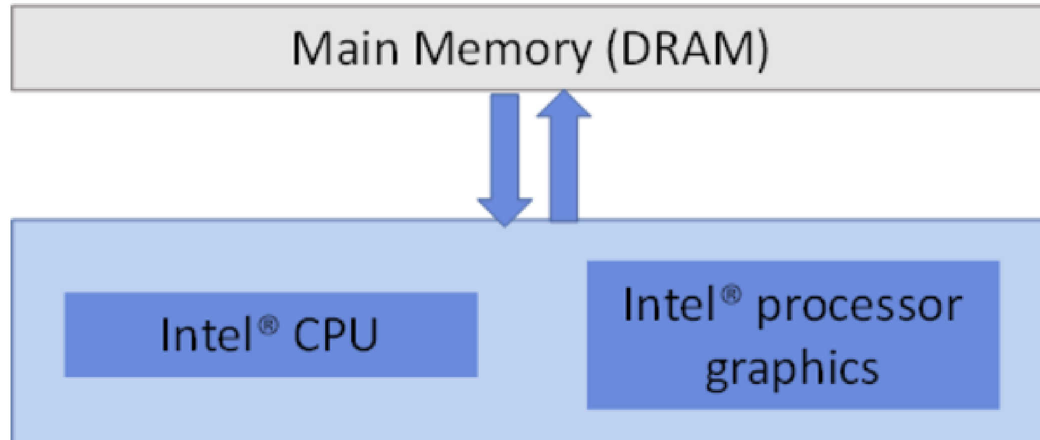
- Optimize host API calls
- **Reduce Host <> Device memory traffic and bandwidth**
- Optimizing memory access
- Maximizing occupancy
- Maximizing computation
- Kernel algorithm optimization

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code Builder ANALYZE

# Zero Copy

- The key hardware feature that enables zero copy is the fact that the CPU and GPU have shared *physical* memory
- Memory shared between the CPU and GPU can be efficiently accessed by both devices





Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Zero Copy

- Always improves performance
- To create zero copy buffers, do one of the following:
  - Use CL_MEM_ALLOC_HOST_PTR
    - Let the runtime handle creating a zero copy allocation buffer for you
  - Use CL_MEM_USE_HOST_PTR with:
    - Buffer allocated at a 4096 byte boundary (aligned to a page and cache line boundary)
    - Total size that is a multiple of 4096 byte (page size)

```
int *pbuf = (int *)_aligned_malloc(sizeof(int) * 1024, 4096);
cl_mem myZeroCopyCLMemObj =
clCreateBuffer(ctx,…CL_MEM_USE_HOST_PTR…);
```

**https://software.intel.com/en-us/articles/getting-the-most-from-opencl-12-how-to-increase-performance-by-minimizing-buffer-copies-on-intel-processor-graphics**

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code snippets provided in this presentation are for illustrative purposes only. Intel disclaims any and all implied or express warranties associated with the code snippets, and any and all use of such code snippets is at the sole discretion and exclusive risk of the user.

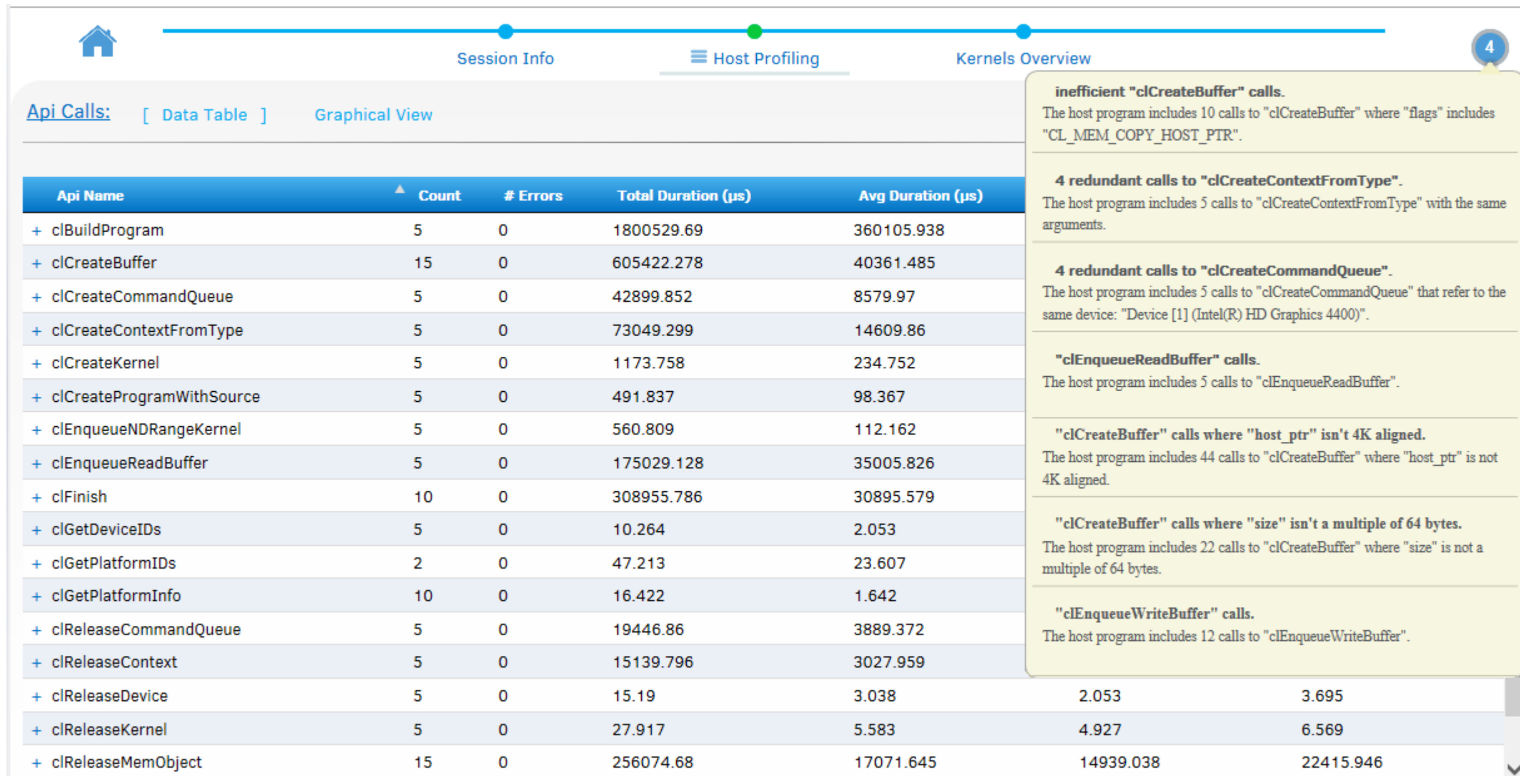
# Zero Copy - Accessing the Buffer on the Host

- Use clEnqueueMapBuffer() and clEnqueueUnmapMemObject()
- Don’t use:
  - clEnqueueReadBuffer()
  - clEnqueueWriteBuffer()

* This behavior may not be the same on all platforms.

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Code Builder API Calls Report




| Api Name | Count | # Errors | Total Duration (µs) | Avg Duration (µs) | | |
|---|---|---|---|---|---|---|
| + clBuildProgram | 5 | 0 | 1800529.69 | 360105.938 | | |
| + clCreateBuffer | 15 | 0 | 605422.278 | 40361.485 | | |
| + clCreateCommandQueue | 5 | 0 | 42899.852 | 8579.97 | | |
| + clCreateContextFromType | 5 | 0 | 73049.299 | 14609.86 | | |
| + clCreateKernel | 5 | 0 | 1173.758 | 234.752 | | |
| + clCreateProgramWithSource | 5 | 0 | 491.837 | 98.367 | | |
| + clEnqueueNDRangeKernel | 5 | 0 | 560.809 | 112.162 | | |
| + clEnqueueReadBuffer | 5 | 0 | 175029.128 | 35005.826 | | |
| + clFinish | 10 | 0 | 308955.786 | 30895.579 | | |
| + clGetDeviceIDs | 5 | 0 | 10.264 | 2.053 | | |
| + clGetPlatformIDs | 2 | 0 | 47.213 | 23.607 | | |
| + clGetPlatformInfo | 10 | 0 | 16.422 | 1.642 | | |
| + clReleaseCommandQueue | 5 | 0 | 19446.86 | 3889.372 | | |
| + clReleaseContext | 5 | 0 | 15139.796 | 3027.959 | | |
| + clReleaseDevice | 5 | 0 | 15.19 | 3.038 | 2.053 | 3.695 |
| + clReleaseKernel | 5 | 0 | 27.917 | 5.583 | 4.927 | 6.569 |
| + clReleaseMemObject | 15 | 0 | 256074.68 | 17071.645 | 14939.038 | 22415.946 |

Optimization Notice


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Code Builder API Calls Report

Code Builder ANALYZE

Session Info | Host Profiling | Kernels Overview

Api Calls: [ Data Table ] Gra...

| Api Name | | | | | | |
|---|---|---|---|---|---|---|
| + clBuildProgram | | | | | | |
| + clCreateBuffer | | | | | | |
| + clCreateCommandQueue | | | | | | |
| + clCreateContextFromType | | | | | | |
| + clCreateKernel | | | | | | |
| + clCreateProgramWithSource | | | | | | |
| + clEnqueueNDRangeKernel | | | | | | |
| + clEnqueueReadBu | | | | | | |
| + clFinish | | | | 30895.579 | | |
| + clGetDeviceIDs | | | | 2.053 | | |
| + clGetPlatformIDs | | | | 23.607 | | |
| + clGetPlatformInfo | 10 | 0 | 16.422 | 1.642 | | |
| + clReleaseCommandQueue | 5 | 0 | 19446.86 | 3889.372 | | |
| + clReleaseContext | 5 | 0 | 15139.796 | 3027.959 | | |
| + clReleaseDevice | 5 | 0 | 15.19 | 3.038 | 2.053 | 3.695 |
| + clReleaseKernel | 5 | 0 | 27.917 | 5.583 | 4.927 | 6.569 |
| + clReleaseMemObject | 15 | 0 | 256074.68 | 17071.645 | 14939.038 | 22415.946 |

There are two ways to ensure zero-copy path on memory objects mapping. Allocate memory with "CL_MEM_ALLOC_HOST_PTR", this method ensures that the memory is efficiently mirrored on the host Another way is to allocate properly aligned and sized memory yourself and share the pointer with the OpenCL framework by using the "CL_MEM_USE_HOST_PTR" flag.

For best results, align memory address to host memory page (4K bytes).

**inefficient "clCreateBuffer" calls.**
The host program includes 10 calls to "clCreateBuffer" where "flags" includes "CL_MEM_COPY_HOST_PTR".

**4 redundant calls to "clCreateContextFromType".**
The host program includes 5 calls to "clCreateContextFromType" with the same arguments.

**4 redundant calls to "clCreateCommandQueue".**
The host program includes 5 calls to "clCreateCommandQueue" that refer to the same device: "Device [1] (Intel(R) HD Graphics 4400)".

**"clEnqueueReadBuffer" calls.**
The host program includes 5 calls to "clEnqueueReadBuffer".

**"clCreateBuffer" calls where "host_ptr" isn't 4K aligned.**
The host program includes 44 calls to "clCreateBuffer" where "host_ptr" is not 4K aligned.

**"clCreateBuffer" calls where "size" isn't a multiple of 64 bytes.**
The host program includes 22 calls to "clCreateBuffer" where "size" is not a multiple of 64 bytes.

**"clEnqueueWriteBuffer" calls.**
The host program includes 12 calls to "clEnqueueWriteBuffer".


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Shared Virtual Memory (SVM)

- Supported from OpenCL* 2.0
- Enables the host and device to seamlessly share pointers and complex pointer-containing data-structures
  - Linked lists or trees
- Tight Host-Kernel synchronization using atomics
  - Just like two distinct cores in a CPU

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Shared Virtual Memory (SVM)

- Basically a productivity feature
  - Targeted to fulfill the needs of developers for tighter host-device synchronization beyond enqueuing commands and synchronizing through events
- Also a very important performance feature
  - Go to “**GPU daemon – Road to Zero Cost Submission**“

    by Michal Mrozek and Zbigniew Zdanowicz (Intel) on THURSDAY 21st APRIL 16:30 – 17:00

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Shared Virtual Memory (SVM)

- Requires dedicated hardware coherency support
  - Such as  enabled in Intel Core Processors with Intel® Graphics Gen8/Gen9 compute architecture
- There are different levels of SVM support depending on OpenCL* platform hardware capabilities
  - Tradeoff between productivity and portability
- Not all OpenCL* platforms support all SVM features
- OpenCL* 2.0 specification defines a minimum level of required SVM support
  - Other features are optional

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

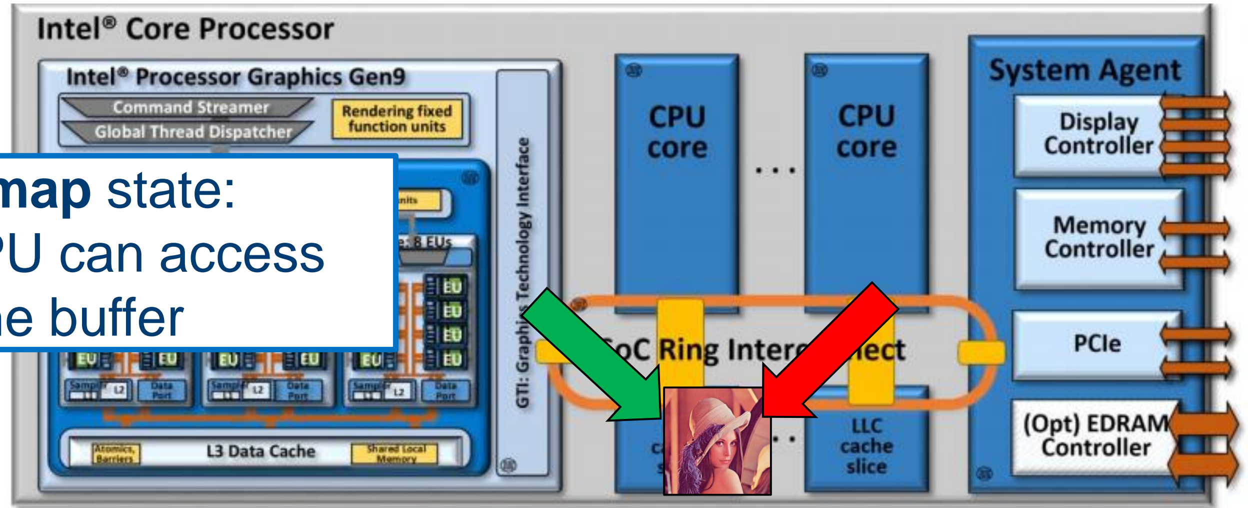
# 3 types of SVM

- Coarse-grain buffers (Intel 5th Gen Processors w/ HD Graphics 5300)
  - SVM buffers are mapped to either CPU or GPU at any given time
  - Access is controlled by clEnqueueSVMMap/Unmap commands
- Fine-grain buffers (Intel 5th Gen Processors w/ HD Graphics 5500+)
  - SVM buffers can be accessed from either CPU or GPU at any time
  - Use atomics to control access (if CPU & GPU may try to modify the same memory location)
  - Check CL_DEVICE_SVM_FINE_GRAIN_BUFFER for fine-grained buffer SVM support, CL_DEVICE_SVM_ATOMICS is for atomics support
- Fine-grain system memory (Future Intel Processors)
  - CPU & GPU can share anything allocated from the C-runtime 'heap' (i.e. malloc/new)


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# 3 types of SVM

- Coarse-grain buffers (Intel 5th Gen Processors w/ HD Graphics 5300)
  - SVM buffers are mapped to either CPU or GPU at any given time
  - Access is controlled by clEnqueueMap/Unmap commands



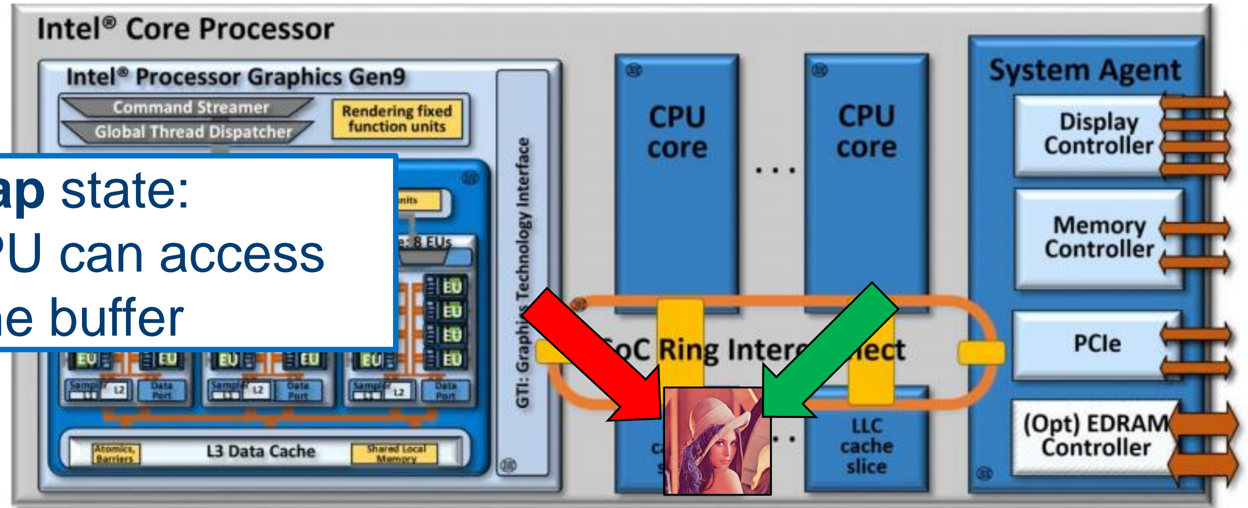

**Un-map** state:
Only GPU can access the buffer



Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# 3 types of SVM

- Coarse-grain buffers (Intel 5th Gen Processors w/ HD Graphics 5300)
  - SVM buffers are mapped to either CPU or GPU at any given time
  - Access is controlled by clEnqueueMap/Unmap commands

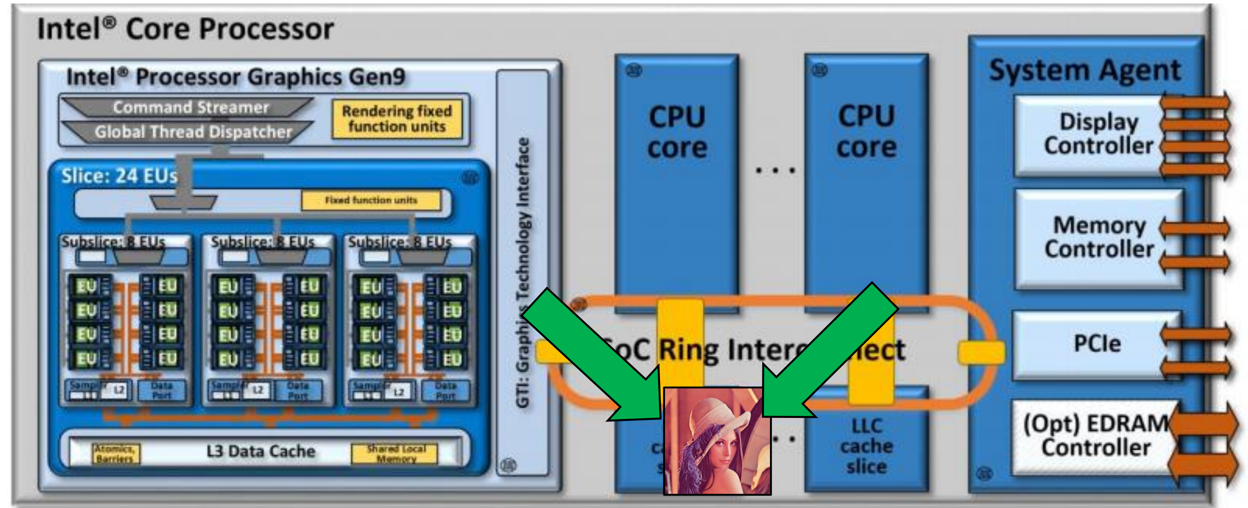




Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# 3 types of SVM

- Fine-grain buffers (Intel 5th Gen Processors w/ HD Graphics 5500+)
  - SVM buffers can be accessed from both CPU and GPU at any time
  - Can use atomics to avoid ‘race’ conditions






Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# 3 types of SVM

- Fine-grain system memory (Future Intel Processors)

Refer to **OpenCL* 2.0 Shared Virtual Memory Overview** for more information

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code Builder ANALYZE

# SVM in VTune Amplifier XE Views

**GPU OpenCL Info**

| | |
|---|---|
| Version: | OpenCL C 2.0 |
| Max Compute Units: | 24 |
| Max Work Group Size: | 256 |
| Local Memory: | 64 KB |
| SVM Capabilities: | Fine-grained buffer with atomics |

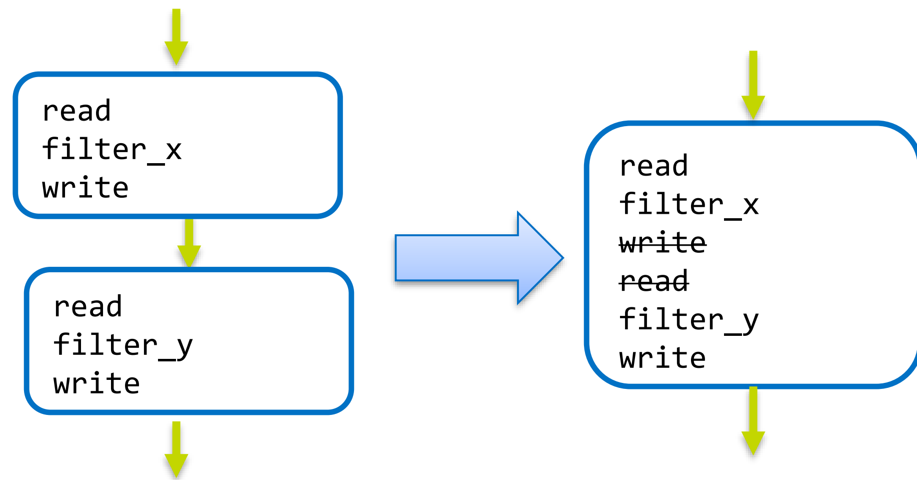
Grouping: Computing Task Purpose / Computing Task (GPU) / Instance

| Computing Task Purpose / Computing Task (GPU) / Instance | Work Size | | Computing Task | | | | |
|---|---|---|---|---|---|---|---|
| | Global | Local | Total Time | Average Time | Instance Count | SIMD Width | SVM Usage Type |
| Compute | | | 499.664s | 0.038s | 13,005 | | |
| ReadWriteCopy_NoAlignPartWrite | 2097152 | 256 | 133.550s | 0.157s | 849 | 32 | |
| ReadWriteCopy_NoAlignPartWrite | 2097152 | 256 | 61.267s | 0.072s | 850 | 32 | Fine-Grained Buffer |
| ReadWriteCopy | 2097152 | 256 | 47.392s | 0.056s | 850 | 32 | Fine-Grained Buffer |
| ReadWriteCopyUnRoll | 2097152 | 256 | 34.491s | 0.041s | 850 | 32 | |
| ReadOnly | 2097152 | 256 | 34.422s | 0.020s | 1,700 | 32 | |
| ReadWriteCopy | 2097152 | 256 | 32.639s | 0.038s | 850 | 32 | Coarse-Grained Buffer |
| ReadWriteCopy | 2097152 | 256 | 31.976s | 0.038s | 850 | 32 | |

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Avoiding Needless Synchronization

- Merging kernels reduces memory traffic
  - But mind instruction cache size
- Continue executing kernels until you really need to read the results
  - Use in-order queue and blocking call to clEnqueueMapXXX
- Merging multiple kernels in a pipeline




Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Optimization Factors

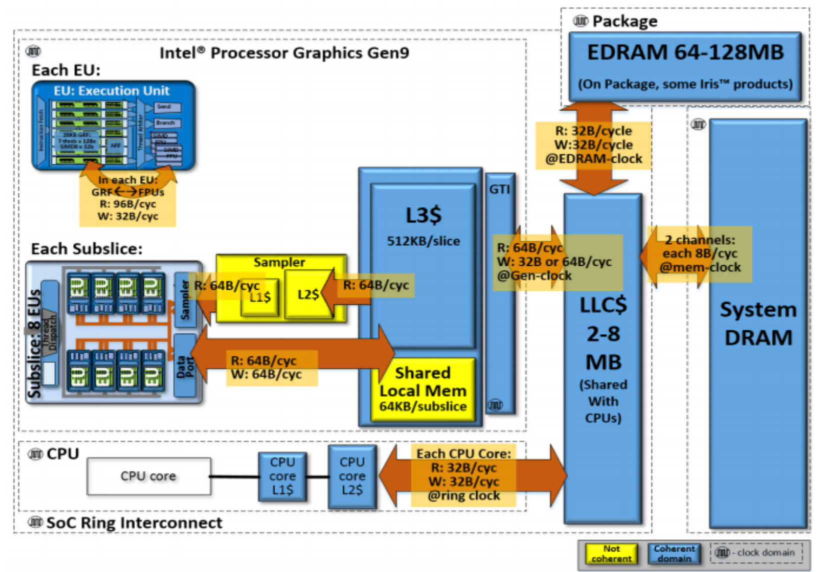
- Optimize host API calls
- Reduce Host <> Device memory traffic and bandwidth
- **Optimizing memory access**
- Maximizing occupancy
- Maximizing computation
- Kernel algorithm optimization

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code Builder ANALYZE

# Optimizing Memory Access

- Accesses to **global** and **constant** memory go through
  - L3 cache: GPU-specific, Cache line is 64 bytes
  - LLC: CPU and GPU shared

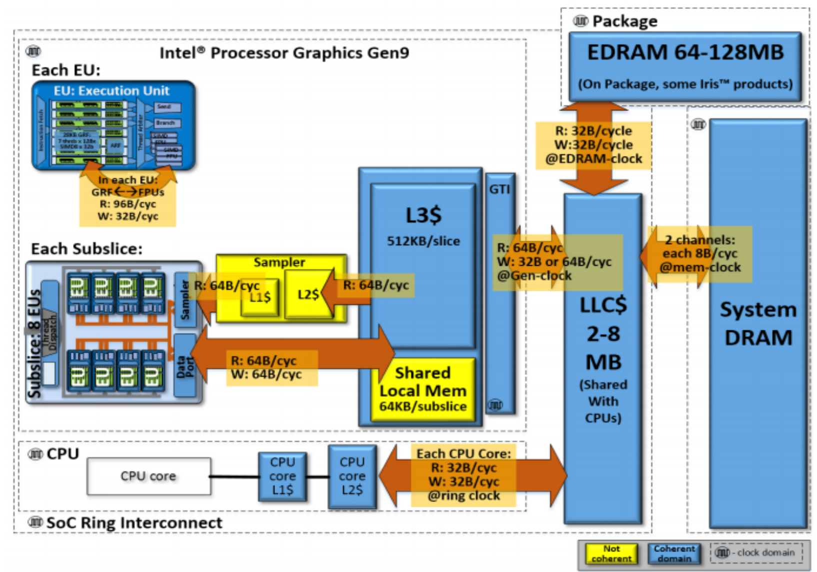




Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
intel

# Optimizing Memory Access

- **Local** memory is allocated directly from the L3 cache
  - Divided into 16 banks at a 32-bit granularity

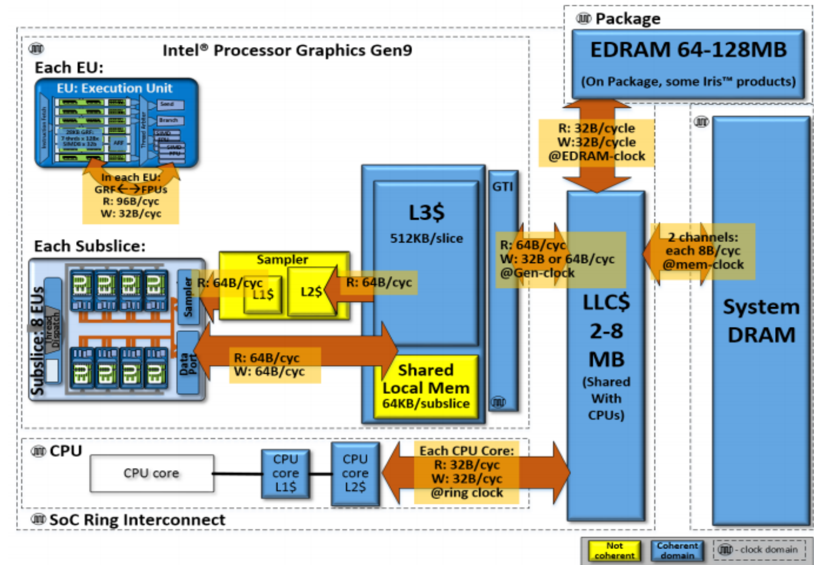



Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code Builder ANALYZE

# Optimizing Memory Access

- **private** memory that is allocated to registers is very efficient to access
- **Private** memory that spills from registers do the same as **Global** memory
  - The performance in this  can be very poor
  - There is no locality for __private memory accesses
  - each work-item accesses a unique cache line for every access to __private memory
  -




Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

intel

# Global Memory and Constant Memory

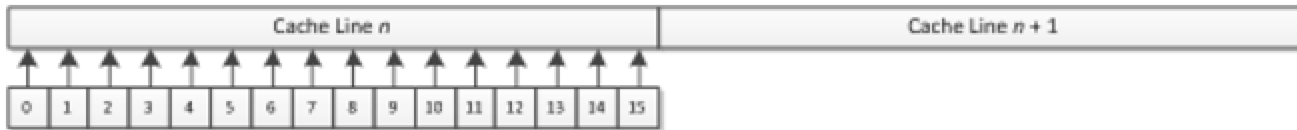
- Global, and constant memory bandwidth is determined by the number of the accessed L3 cache lines.
  - If two L3 cache lines are accessed from different work items in the same hardware thread, memory bandwidth is ½ of the memory bandwidth in case when only one L3 cache line is accessed
- Affected by two factors
  - The access pattern function of the work-item global id(s)
  - The work-group dimensions

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

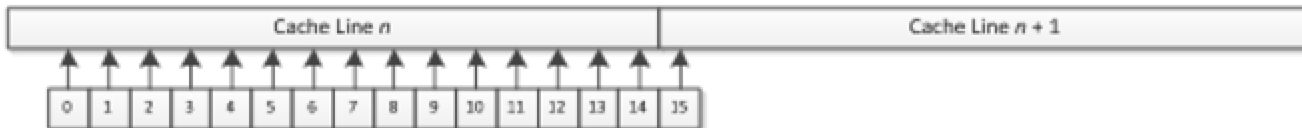
# Global Memory and Constant Memory

**Access pattern** function example:
Workgroup = <16,1,1>

```
x = myArray[ get_global_id(0) ];
```




```
x = myArray[ get_global_id(0) + 1 ]
```




Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

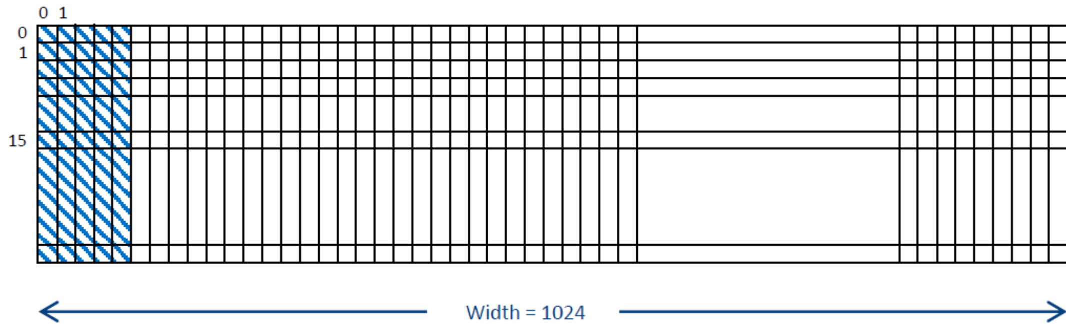
# Global Memory and Constant Memory

Host

```
size_t localWorkSize[2] = {1, 16};
clEnqueueNDRangeKernel(…, localWorkSize,…);
```

Kernel

```
const int id = y * width + x;
local = buffer[id];
```

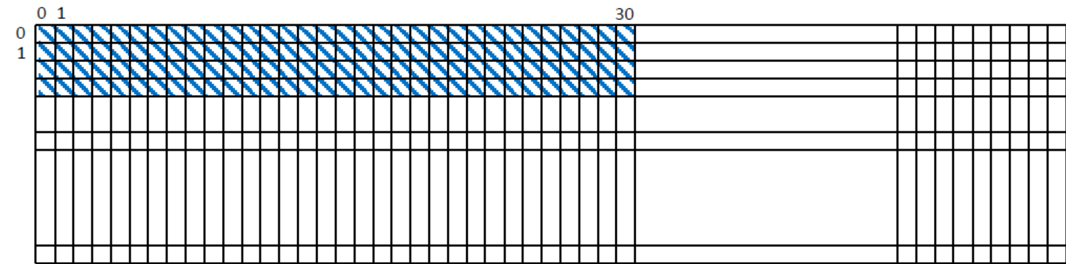



LocalWorkSize[2] = {1,16}

| | | | | |
|---|---|---|---|---|
| {x,y}= | 0,0 | 0,1 | . . . | 0,15 |
| y * width + x = | 0 | 1024 | . . . | 15360 |

Y is different for every work-item in the work-group; every read operation comes from a different cache line for every work-item in the work-group

Width = 1024




LocalWorkSize[2] = {16,1}

| | | | | |
|---|---|---|---|---|
| {x,y}= | 0,0 | 1,0 | | 15,0 |
| y * width + x = | 0 | 1 | . . . | 15 |

Y is constant for all work-items in the work-group. Id increases monotonically across the entire work-group, which means that the read operations comes from a single L3 cache line (16 x sizeof(int) = 64 bytes).

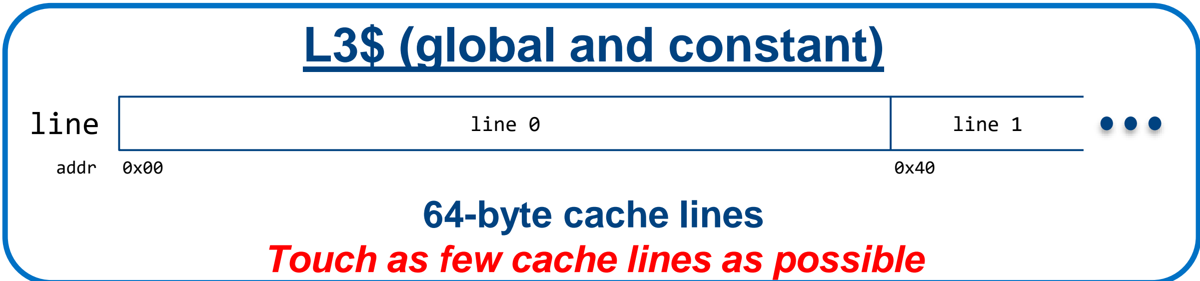

Optimization Notice

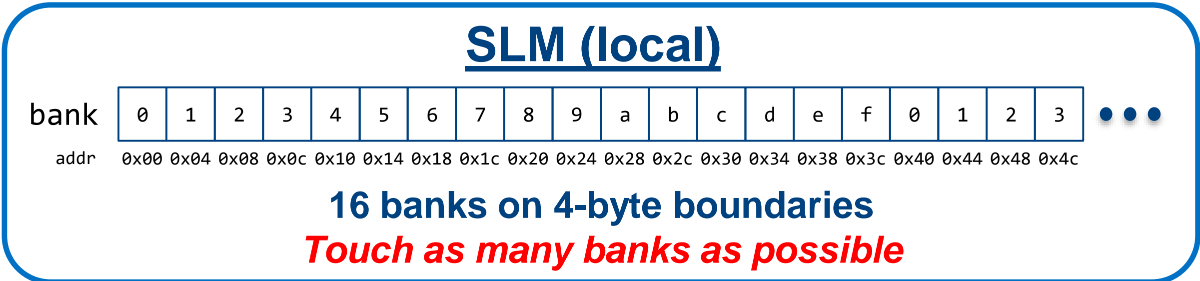
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code snippets provided in this presentation are for illustrative purposes only. Intel disclaims any and all implied or express warranties associated with the code snippets, and any and all use of such code snippets is at the sole discretion and exclusive risk of the user.

# Local Memory

- Highly-banked
  - More important to minimize bank conflicts than to minimize the number of L3 cache lines accesses
- Local memory accesses have latencies similar to L3$ hits
  - Using only local memory as a cache is often not productive
- But, local memory and L3$ are organized differently





Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Private Memory

- Each work item in an OpenCL* kernel has access to up to 512 bytes of register space
- Bandwidth to registers is faster than any memory
- Loading and processing blocks of pixels in registers is very efficient!
  - Example: non-separable convolution (filter2D) in OpenCV*

```
float sum[PX_PER_WI_X] = { 0.0f };
float k[KERNEL_SIZE_X];
float d[PX_PER_WI_X + KERNEL_SIZE_X];
er kernel in k, input data in d

onvolution
 px < PX_PER_WI_X; ++px)
    for (sx = 0; sx < KERNEL_SIZE_X; ++sx)
        sum[px]= mad(k[sx], d[px + sx], sum[px]);
```

Achieved up to 5.7X!

***Use available registers (up to 512 bytes) instead of memory, where possible!***

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Optimization Factors

- Optimize host API calls
- Reduce Host <> Device memory traffic and bandwidth
- Optimizing memory access
- **Maximizing occupancy**
- Maximizing computation
- Kernel algorithm optimization

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Maximizing Occupancy

- Occupancy is a measure of utilization
- The goal is to keep a sufficient number of work-groups active
  - If one is stalled, another can run on its hardware resource.

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Maximizing Occupancy

- Two primary things to consider:
  - Launch enough work items to keep GPU units busy
    - Compiler may pack up to 32 work items per thread (with SIMD-32).
  - Let the kernel do enough work
    - In short kernels: use short vector data types and compute multiple pixels to better amortize thread launch cost
    - Use Vload and Vstore

```
__global uchar* src, dst;
p = src[src_idx]     * B2Y +
    src[src_idx + 1] * G2Y +
    src[src_idx + 2] * R2Y;
dst[dst_idx] = p;
```

1 pixel per work item

```
__global uchar* src_ptr, dst_ptr;
uchar16 src = vload16(0, src_ptr);
uchar4 c0 = src.s048c;
uchar4 c1 = src.s159d;
uchar4 c2 = src.s26ae;
uchar4 Y = c0 * B2Y +
           c1 * G2Y +
           c2 * R2Y;
vstore4(Y, 0, dst_ptr);
```

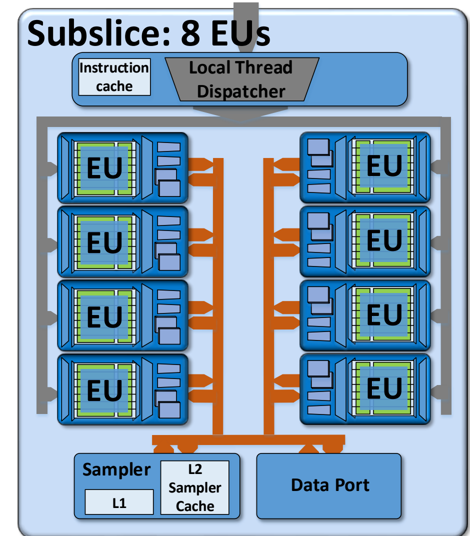
4 pixels per work item

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Code snippets provided in this presentation are for illustrative purposes only. Intel disclaims any and all implied or express warranties associated with the code snippets, and any and all use of such code snippets is at the sole discretion and exclusive risk of the user.

Code Builder ANALYZE

# Maximizing Occupancy

- More subtle occupancy issues (when using barriers or local memory):
  - Sub-slices will not run partial workgroups
    - Can be a limiting factor for very large work groups
  - Sub-slices will not run more than 16 (32 on Gen9) work groups
    - Can be a limiting factor for very small work groups
  - Shared Local Memory (SLM) – 64KB SLM per sub-slice
    - Can be a limiting factor for kernels that use a lot of local memory
- General advice when using barriers or local memory
  - Experiment with workgroup sizes of 64, 128, or 256
  - Use less than 64 bytes of local memory per work item






Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

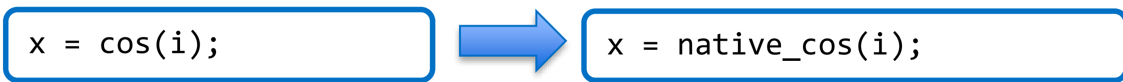
# Optimization Factors

- Optimize host API calls
- Reduce Host <> Device memory traffic and bandwidth
- Optimizing memory access
- Maximizing occupancy
- **Maximizing computation**
- Kernel algorithm optimization

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Maximizing Compute Performance

- Prefer float over int, if possible
- Trade accuracy for speed, where appropriate
  - Use native_* and built-ins (or use -cl-fast-relaxed-math)
  - Compiler optimization options that enable optimizations for floating-point arithmetic for the whole OpenCL* program:
    - For example: -cl-mad-enable, -cl-fast-relaxed-math




***Used to speedup OpenCV* SURF and HOG!***

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Using OpenCL* 2.0 Workgroup Functions

- The OpenCL* 2.0 standard offers new workgroup built in functions
  - Parallel Primitive – popular parallel primitives (scan, reduction)
    - Operations available: add, min, max
    - Allows reductions and scans without exposed local memory or barriers
  - Broadcast – Transmit data from one work item to all work items within the workgroup
  - Predicate – evaluate a predicate for all work items in a workgroup (any, all)

- Convenient
  - much simpler to use
- Performance efficient
  - Device-specific implementation optimized for the hardware

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Using OpenCL* 2.0 Workgroup Functions

- Predicate workgroup functions
  - work_group_reduce_<op> -
  - work_group_scan_exclusive_<op>
  - work_group_scan_inclusive_<op>
    - Operations available: add, min, max

```
__kernel void foo(int *p)
{
    ...
    int prefix_sum_val = work_group_scan_inclusive_add(p[get_local_id(0)]);
   …
}
```

P= [3 1 7 0 4 1 6 3]

prefix_sum_val = [3 4 11 11 15 16 22 25]

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code snippets provided in this presentation are for illustrative purposes only. Intel disclaims any and all implied or express warranties associated with the code snippets, and any and all use of such code snippets is at the sole discretion and exclusive risk of the user.

Code Builder ANALYZE

# Example: work_group_reduce_add

```
__local float smem[256];
unsigned int id = get_local_id(0);
float smem[id] = sum = input;

if (id < 128) smem[id] = sum = sum + smem[id + 128]; barrier(CLK_LOCAL_MEM_FENCE);
if (id <  64) smem[id] = sum = sum + smem[id +  64]; barrier(CLK_LOCAL_MEM_FENCE);
if (id <  32) smem[id] = sum = sum + smem[id +  32]; barrier(CLK_LOCAL_MEM_FENCE);
if (id <  16) smem[id] = sum = sum + smem[id +  16]; barrier(CLK_LOCAL_MEM_FENCE);
if (id <   8) smem[id] = sum = sum + smem[i
if (id <   4) smem[id] = sum = sum + smem[i
if (id <   2) smem[id] = sum = sum + smem[i
if (id <   1) smem[id] = sum = sum + smem[i
sum = smem[0];
```

- ✓ *No exposed local memory or barriers*
- ✓ *Code written independent of workgroup size*
- ✓ *Intel optimized for Processor Graphics*

```
sum = work_group_reduce_add(input);
```

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code snippets provided in this presentation are for illustrative purposes only. Intel disclaims any and all implied or express warranties associated with the code snippets, and any and all use of such code snippets is at the sole discretion and exclusive risk of the user.

intel

# Using OpenCL* 2.0 Workgroup Functions

- Predicate workgroup functions
  - work_group_all()
  - work_group_any()

```
__kernel void foo (int *in, int *out)
{
    …
    int gid = get_global_id(0);
    int result = work_group_all(in[gid] < in[gid+1])
    …
}
```

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Code snippets provided in this presentation are for illustrative purposes only. Intel disclaims any and all implied or express warranties associated with the code snippets, and any and all use of such code snippets is at the sole discretion and exclusive risk of the user.

# Develop OpenCL* applications with Intel® SDK for OpenCL™ Applications

# OpenCL* is Great!

- However…
  - Development is not trivial
  - Debugging of parallel processing applications is difficult
  - Optimization of OpenCL* applications is platform-dependent and very challenging

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® SDK for OpenCL™ Applications

**The OpenCL* development environment for Intel® based platforms**

Available for free

# Intel® Code Builder for OpenCL™ API

**Comprehensive development environment for the build, debug, and analysis of an OpenCL* applications**

- BUILD AND CREATE
  - JumpStart Kit
    - A wizard for creating an OpenCL* project
  - Kernel Development Framework (KDF)
    - Stand alone environment for write, compile and run kernels

Building

Debugging

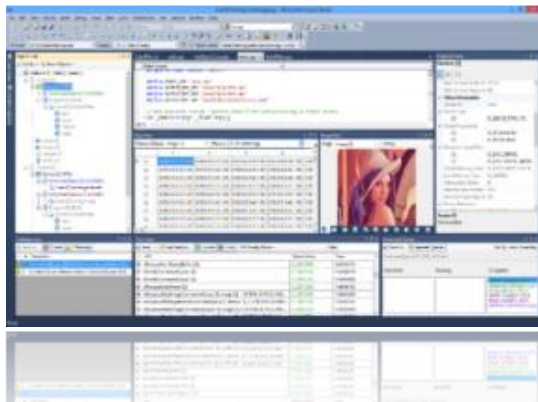
Performance Analysis

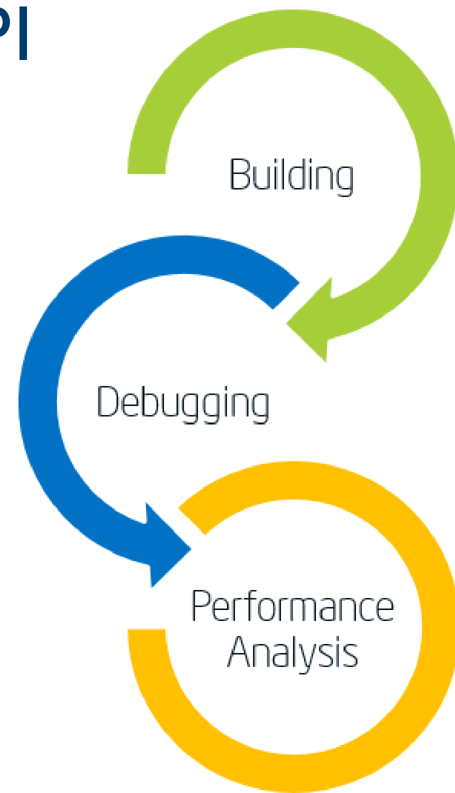
Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Code Builder for OpenCL™ API

**Comprehensive development environment for the build, debug, and analysis of an OpenCL* applications**

- DEBUG
  - Seamless debugging tool for OpenCL* applications
    - OpenCL* API debugger for host side debugging
    - OpenCL* Kernel Debugger for device side debugging



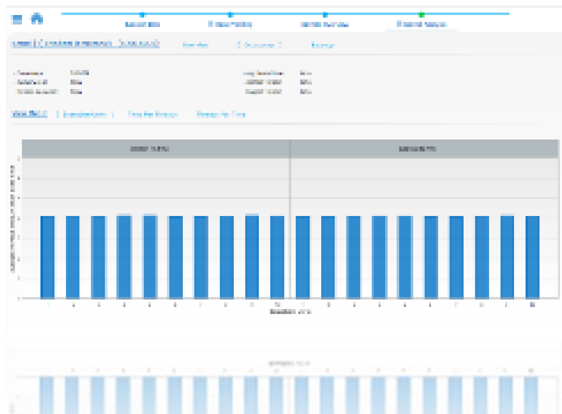


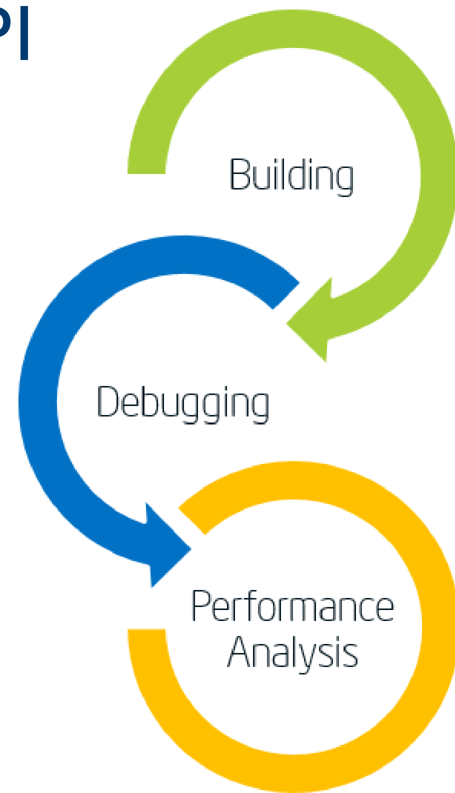

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Code Builder for OpenCL™ API

**Comprehensive development environment for the build, debug, and analysis of an OpenCL* applications**

- ANALYZE
  - Easy and simple performance debugging tool
  - Collect performance data from both the host side and the kernel side





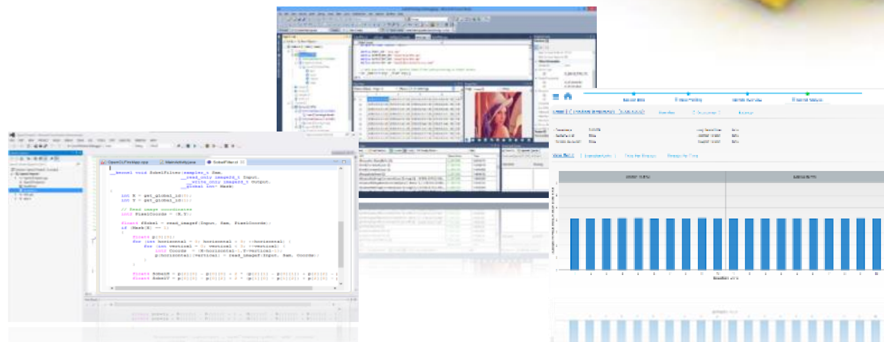


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® Code Builder for OpenCL™ API

**Comprehensive development environment for the build, debug, and analysis of an OpenCL* applications**

- Integration
  - A single framework for all the functionality that the developer needs
  - Smooth path between all components
  - IDE native integration




Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Build and create

# Create new OpenCL* Project

- Create OpenCL* project  with Jump-Start wizard
  - Very simple wizard for creating new OpenCL* project
  - Intended for developers that write an OpenCL* application from scratch
  - Plug in for Visual Studio*


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Create new OpenCL* Project

- In a few clicks you can generate:
  - An empty project ready for you to implement host and kernel code
  - Full host + kernel code project ready for build

New Project

Recent
Installed
Templates
Visual C++
Windows Store
ATL
CLR
General
MFC
Test
Win32
OpenCL
Windows Phone
LightSwitch
Other Languages
Other Project Types
Samples
Online

.NET Framework 4.5 Sort by: Default

Search Installed Templates (Ctrl+E)

| | |
|---|---|
| CodeBuilder Project for Android | Visual C++ |
| CodeBuilder Project for Windows | Visual C++ |
| Empty OpenCL Project for Windows | Visual C++ |

Type: Visual C++
An OpenCL Project

Name:
Location:
Solution: Create new solution
Solution name: <Enter_name>

Code Builder wizard for OpenCL API

intel WELCOME TO CODE BUILDER WIZARD FOR OPENCL API

Basic Settings
Advanced Settings

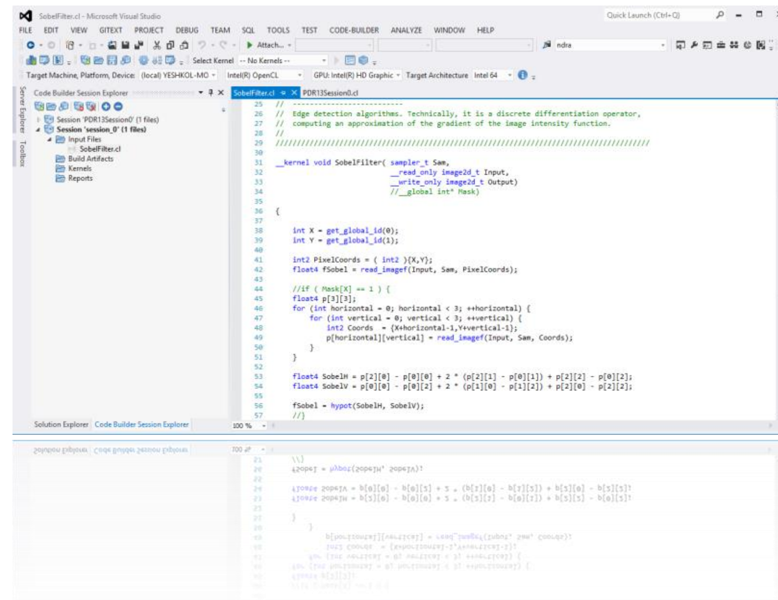
Platform Name: Intel
Device Type: GPU
Create Kernel With Buffers
Create Kernel With Images
Build Options:
Use explicit Local work Group Size

< Previous | Next > | Finish | Cancel


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Kernel Development Framework (KDF)

- Standalone environment for kernel development
- Syntax checking and auto-completion for OpenCL* C language
- Offline compilation and binary generation of OpenCL* kernels
- Compilation error reports





Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Kernel Development Framework

FILE EDIT VIEW GITEXT PROJECT BUILD DEBUG TEAM SQL TOOLS TEST CODE-BUILDER ANALYZE WINDOW HELP

Local Windows Debugger | Auto | Debug | Win32 | local_index

Select Kernel | Sobel_v1_uchar

Target Machine, Platform, Device: (local) YESHKOL-MO | Intel(R) OpenCL | GPU: Intel(R) HD Graphic | Target Architecture | Intel 64

Server Explorer | Toolbox

C:\Users\yeshkol\D...el_v2_uchar16.meta

```
54  __kernel void Sobel_v2_uchar16 (__global uchar16* pSrcImage, __global uchar16*
55  {
56      uint dstYStride = get_global_size(0);
57      uint dstIndex   = get_global_id(1) * dstYStride + get_global_id(0);
58      uint srcYStride = dstYStride + 2;
59      uint srcIndex   = get_global_id(1) * srcYStride + get_global_id(0) + 1;
60
61      uint a; uint16 b; uint c;
62      uint d; uint16 e; uint f;
63      uint g; uint16 h; uint i;
64
65      // Read data in
66      a = ((__global uchar*)(pSrcImage+srcIndex))[-1]; b = convert_uint16(pSrcIma
67      srcIndex += srcYStride;
68      d = ((__global uchar*)(pSrcImage+srcIndex))[-1]; e = convert_uint16(pSrcIma
69      srcIndex += srcYStride;
70      g = ((__global uchar*)(pSrcImage+srcIndex))[-1]; h = convert_uint16(pSrcIma
71
72      uint16 xVal, yVal;
73
74      xVal = (uint16)(a, b.s0123, b.s456789ab, b.scde) -   (uint16)(b.s123, b.
75           2*(uint16)(d, e.s0123, e.s456789ab, e.scde) - 2*(uint16)(e.s123, e.
76             (uint16)(g, h.s0123, h.s456789ab, h.scde) -   (uint16)(h.s123, h.
77
78      yVal = (uint16)(a, b.s0123, b.s456789ab, b.scde) + 2*b + (uint16)(b.s123, b
79             (uint16)(g, h.s0123, h.s456789ab, h.scde) - 2*h - (uint16)(h.s123, h
80
81      // Write data out
82      pDstImage[dstIndex] = convert_uchar16(min((float16)255.0f, sqrt(convert_flo
83  }
```

| Line | Offset | | Op | Exec | Dst | Src0 | Src1 |
|---|---|---|---|---|---|---|---|
| 16 | 0xa0 | (W) | mov | (8\|M8) | r35.0<1>:d | acc0.0<8;8,1>:d | |
| 17 | 0xb0 | | mov | (8\|M8) | r54.0<1>:d | r35.0<8;8,1>:d | {Compa |
| 18 | 0xb8 | | mul | (8\|M0) | acc0.0<1>:d | r51.0<8;8,1>:d | r2.0 |
| 19 | 0xc0 | | mach | (8\|M0) | null<1>:d | r51.0<8;8,1>:d | r2.0 |
| 20 | 0xc8 | | mov | (8\|M0) | r55.0<1>:d | acc0.0<8;8,1>:d | |
| 21 | 0xd8 | | add | (16\|M0) | r59.0<1>:d | r53.0<8;8,1>:d | r57.( |
| 22 | 0xe0 | | add | (16\|M0) | r63.0<1>:d | r59.0<8;8,1>:d | 1:d |
| 23 | 0xe8 | | add | (16\|M0) | r68.0<1>:d | r63.0<8;8,1>:d | r67.( |
| 24 | 0xf0 | | shl | (16\|M0) | r70.0<1>:ud | r63.0<8;8,1>:ud | 0x4: |
| 25 | 0xf8 | | shl | (16\|M0) | r81.0<1>:ud | r68.0<8;8,1>:ud | 0x4: |
| 26 | 0x100 | | add | (16\|M0) | r85.0<1>:ud | r70.0<8;8,1>:ud | 0xC: |
| 27 | 0x108 | | add | (16\|M0) | r79.0<1>:ud | r70.0<8;8,1>:ud | 0xFF |
| 28 | [illegible]18 | | add | (16\|M0) | r91.0<1>:ud | r81.0<8;8,1>:ud | 0xC: |
| 29 | 0x1[illegible] | | add | (16\|M0) | r95.0<1>:d | r85.0<8;8,1>:d | -12: |
| 30 | 0x130 | | add | (16\|M0) | r87.0<1>:ud | r81.0<8;8,1>:ud | 0xFF |
| 31 | 0x140 | | add | (16\|M0) | r103.0<1>:d | r91.0<8;8,1>:d | -12: |
| 32 | 0x150 | | add | (16\|M0) | r74.0<1>:d | r68.0<8;8,1>:d | r67.( |
| 33 | 0x158 | | send | (16\|M0) | r33:uw | r95 | 0xC |
| 34 | 0x168 | | add | (16\|M0) | r93.0<1>:ud | r81.0<8;8,1>:ud | 0x10 |
| 35 | 0x170 | | send | (16\|M0) | r113:uw | r87 | 0xA |
| 36 | 0x180 | | send | (16\|M0) | r105:uw | r79 | 0xA |
| 37 | 0x190 | | add | (16\|M0) | r83.0<1>:ud | r70.0<8;8,1>:ud | 0x10 |
| 38 | 0x19[illegible] | | send | (16\|M0) | r25:uw | r103 | 0xC |
| 39 | [illegible] | | shl | (16\|M0) | r89.0<1>:ud | r74.0<8;8,1>:ud | 0x4: |
| [illegible] | 0x1b0 | | send | (16\|M0) | r5:uw | r93 | 0xA |
| 41 | 0x1c0 | | send | (16\|M0) | r111:uw | r83 | 0xA |
| 42 | 0x1d0 | | add | (16\|M0) | r99.0<1>:ud | r89.0<8;8,1>:ud | 0xC: |
| 43 | 0x1d8 | | add | (16\|M0) | r97.0<1>:ud | r89.0<8;8,1>:ud | 0xFF |
| 44 | 0x1e8 | | add | (16\|M0) | r101.0<1>:ud | r89.0<8;8,1>:ud | 0x10 |
| 45 | 0x1f0 | | add | (16\|M0) | r109.0<1>:d | r99.0<8;8,1>:d | -12: |
| 46 | 0x200 | | send | (16\|M0) | r121:uw | r97 | 0xA |

## Gen assembly <> OpenCL*-C Line mapping


Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

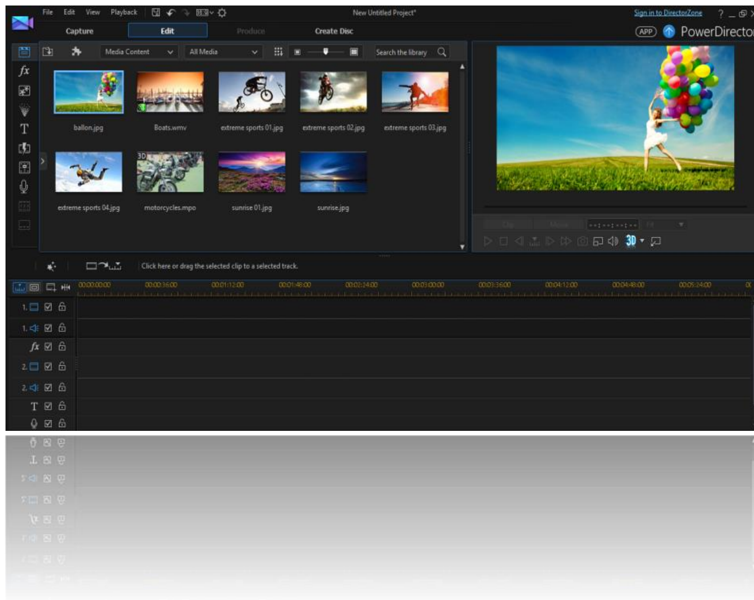
Code snippets provided in this presentation are for illustrative purposes only. Intel disclaims any and all implied or express warranties associated with the code snippets, and any and all use of such code snippets is at the sole discretion and exclusive risk of the user.

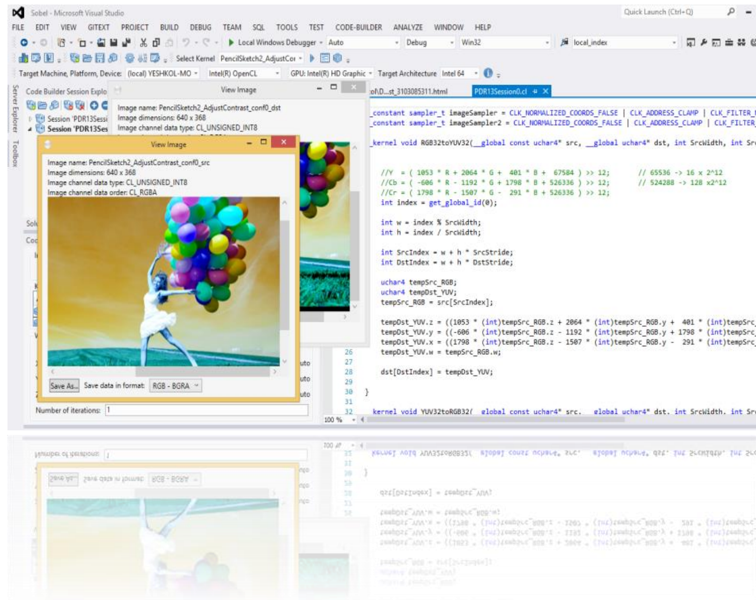
# Kernel Development Framework

- Run and review the results
  - Assign variables to the kernel and check its correctness
  - Show the input and output values
- Capture kernel session from exiting OpenCL* application
  - Store the kernels code with its inputs (buffer or images)
- Coming soon:
  - Generate host code from session
  - Validate kernel outputs versus a reference

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Capture & Reply Kernel Sessions

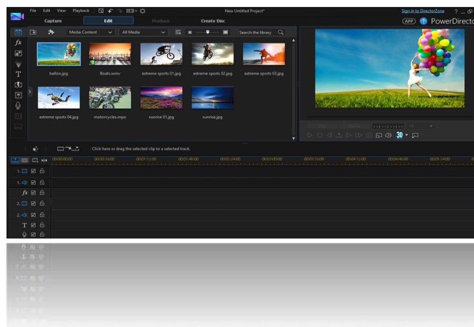


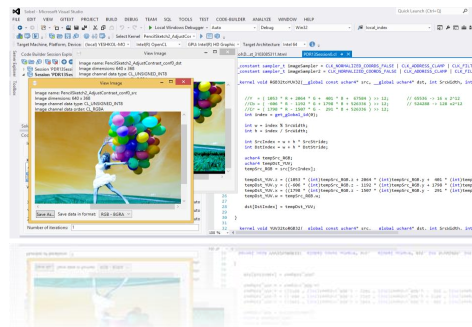


Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Capture & Reply Kernel Sessions

- Very useful when kernel inputs are not available
  - Created in run-time
  - Output of a previous kernel
- Very useful when the application requires user interaction to execute the kernel
- Eliminates the need to run the application for any kernel change






Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# DEMO – Kernel Development

- Sobel Kernel
  - Edge detection algorithm
  - Discrete differentiation operator, computing an approximation of the gradient of the image intensity function
  - https://en.wikipedia.org/wiki/Sobel_operator

Source Image

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

horizontal and vertical derivative approximations

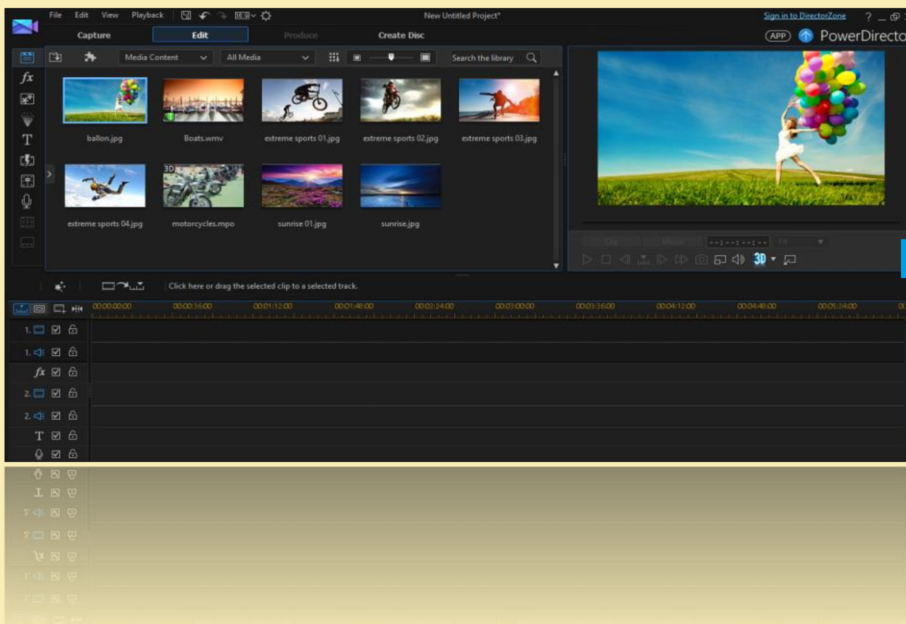
$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

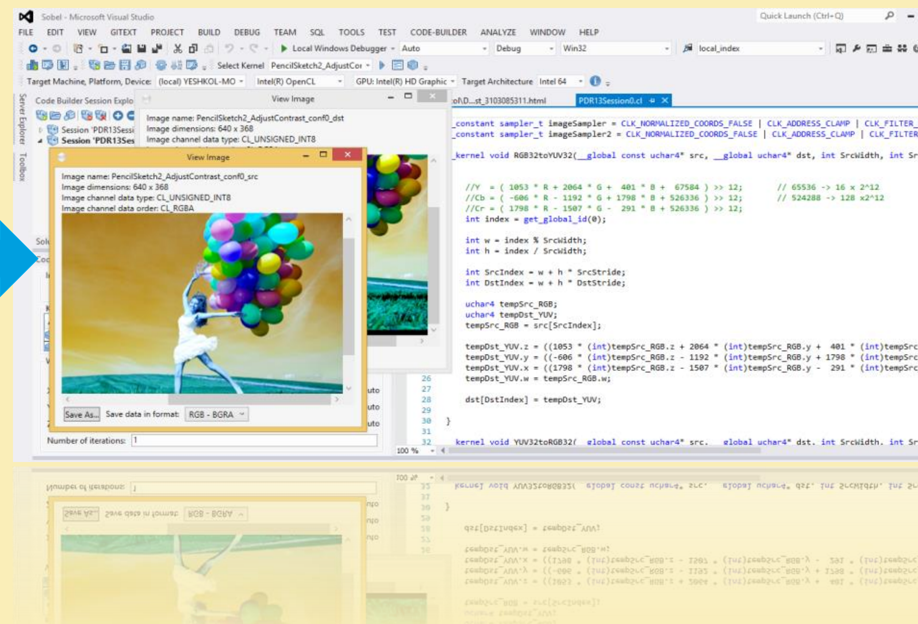
Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# DEMO – Session Generation

- Generate session from CyberLink Power Director*
  A High Performance Video Editing suite





Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Debug

# Host Level Debugging

- Seamless debugging of OpenCL* API calls, objects, and queues
- Enables monitoring and understanding the OpenCL* environment of an application execution
- OpenCL* API call tracing
- Images and memory objects view
- Extension to the Visual Studio* debugger

Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

**Objects Tree View**
Explore all OpenCL* objects in memory and their properties

**Date View**
Show the content of OpenCL* memory objects (buffers + images)

**Image View**
Show the visualized content of OpenCL* image objects

**Commands Queue View**
Examine commands queue status and their commands' state

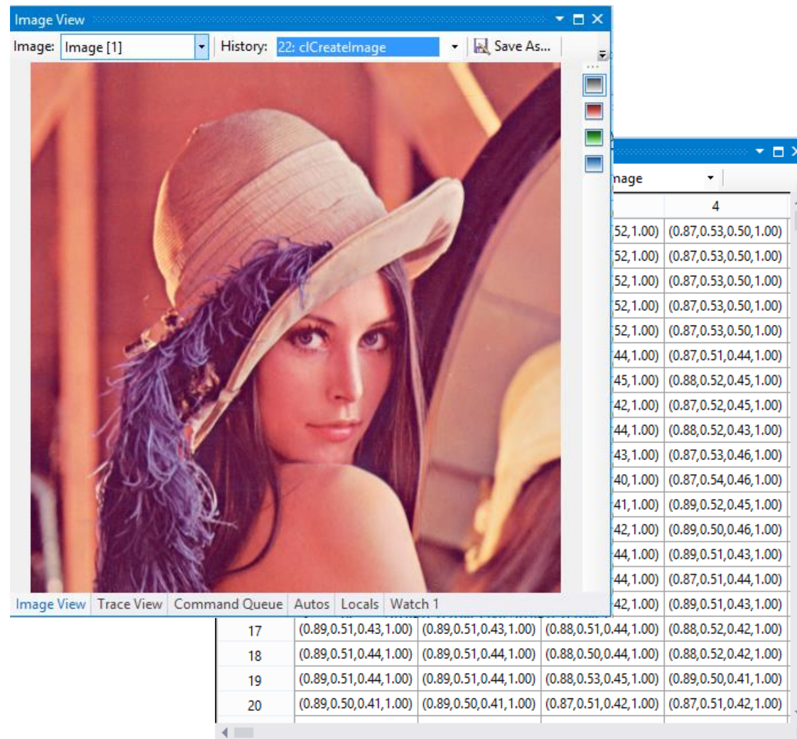
**Problems View**
Look for hints for potential error or warnings during execution

**Trace View**
Trace application's OpenCL* API calls and their return values

**Properties View**
View the properties of the selected OpenCL* objects

# Host Level Debugging

- Image view
  - Show the visualized content of OpenCL* Image objects when hitting the break point
  - Option to see the image content in different stages of the program
  - Channel filter
- Data view
  - Show the content of OpenCL* memory objects (buffers + images)



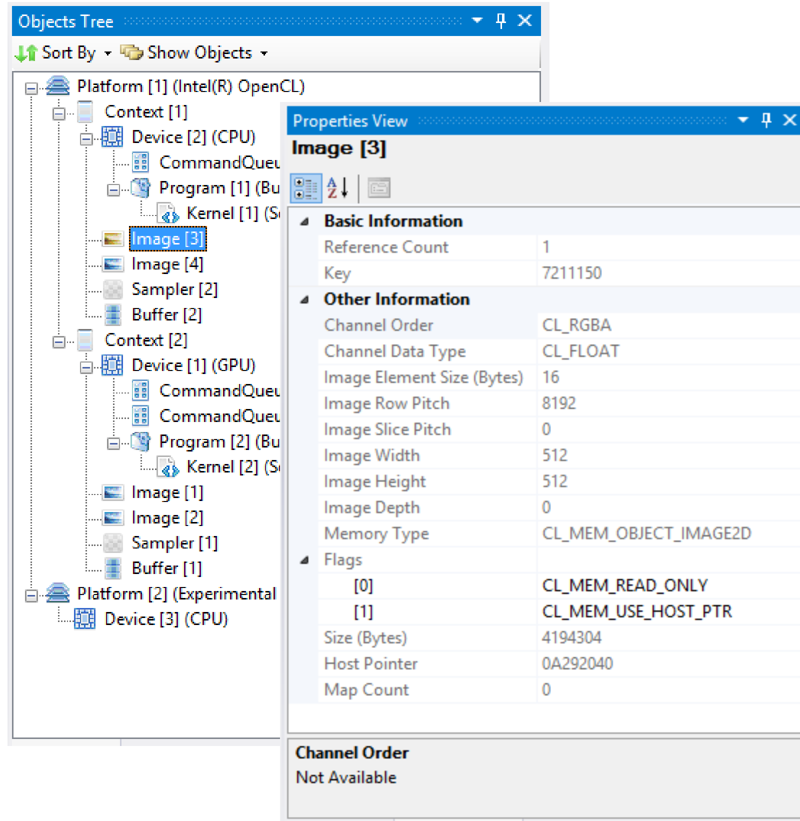


Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Host Level Debugging

- Object tree view
  - Hierarchical view of all OpenCL* objects in memory
  - Filter objects by type

- Properties view
  - View the properties of the selected OpenCL* objects





Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Host Level Debugging

- Problems view
  - Look for hints for potential error or warnings during execution
  - Filter for errors/warning

Problems view

0 Errors | 3 Warnings

| | # | Decription |
|---|---|---|
| ⚠ | 1 | Kernel [2] (SobelFilter)'s argument number 1 is not initialized yet |
| ⚠ | 2 | Kernel [2] (SobelFilter)'s argument number 2 is not initialized yet |
| ⚠ | 3 | Kernel [2] (SobelFilter)'s argument number 3 is not initialized yet |

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Host Level Debugging

- Trace view
  - Show for any executed OpenCL* API:
    - Name and arguments
    - Error code
    - Return value
    - Execution time
  - Filter by errors/success

Trace View

Save... | Load Session... | Success | Errors | API Display Mode | Filter:

| | API | Return Value | Error Code | Time |
|---|---|---|---|---|
| 25 | clCreateImage(Context [1], CL_MEM_READ_ONLY... | Image [3] | CL_SUCCESS | 10:03:40:720 |
| 24 | clCreateSampler(Context [2], CL_FALSE, CL_ADDR... | Sampler [1] | CL_SUCCESS | 10:03:40:712 |
| 23 | clCreateImage(Context [2], CL_MEM_WRITE_ONL... | Image [2] | CL_SUCCESS | 10:03:40:610 |
| 22 | clCreateImage(Context [2], CL_MEM_READ_ONLY... | Image [1] | CL_SUCCESS | 10:03:40:481 |
| 21 | clGetDeviceInfo(Device [2] (CPU), CL_DEVICE_ME... | CL_SUCCESS | CL_SUCCESS | 10:03:37:856 |

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Host Level Debugging

- Commands queue view
  - Examine commands queue status and their commands' state
  - Help understand the commands flow thru the various queues during the application

Command Queue

Unify Queues

Sort By Time: Ascending

CommandQueue [1] (CPU, In Order)

| Submitted | Running | Completed |
|---|---|---|
| NDRANGE_KERNEL(1) | WRITE_IMAGE(0) | |

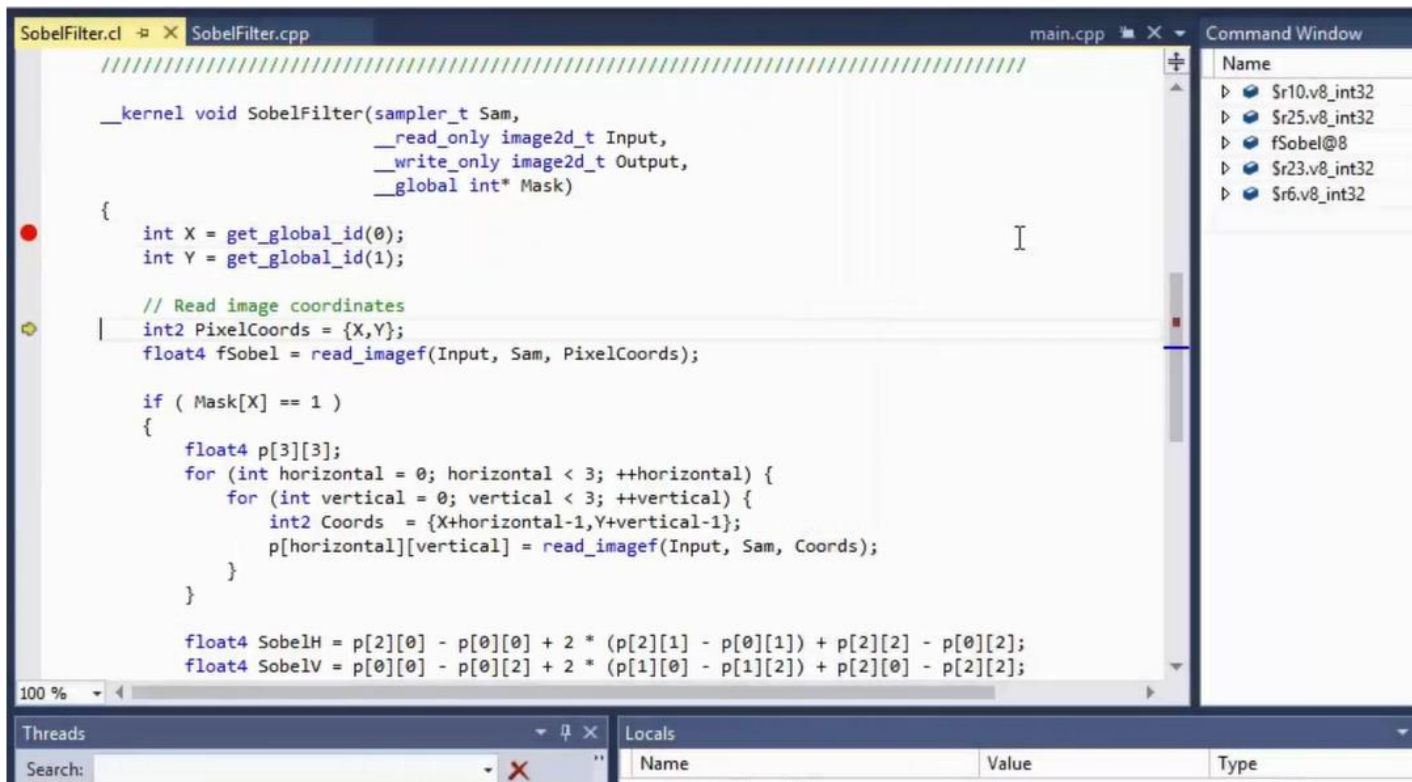
Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Kernel Level Debugging on the GPU

- Enables source and assembly level debugging on GPU
- Provide all the conveniences of the modern debugger
  - Step-in, break and continue, show variables, switch between threads, etc.
- Enhanced for the specifics of OpenCL
  - Ability to view the content of vector variables like float4, uchar16 etc.
- Remote debugging only (host vs. target)
- GDB based
- Microsoft Visual Studio* 2015 integration
- Supported on Gen9 and above (Beta version) on Windows

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Kernel Level Debugging on the GPU





Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Kernel Level Debugging on the GPU

Disassembly

Address: SobelFilter

Viewing Options

```
0x00000000000920e0  (W)        mul    (1|M0)               r18.0<1>:uw
0x00000000000920f0  (W)        add    (1|M0)               a0.0<1>:uw
0x0000000000092100  (W)        mov    (1|M0)               r14.0<1>:d
0x0000000000092110  (W)        mul    (1|M0)               r12.0<1>:q
0x0000000000092120             mov    (8|M0)               r13.0<1>:d
0x0000000000092130             mov    (8|M0)               r6.0<1>:q
0x0000000000092140             add    (8|M0)               r15.0<1>:q
0x0000000000092150  (W)        mov    (1|M0)               r17.0<1>:q
0x0000000000092160             add    (8|M0)               r19.0<1>:q
0x0000000000092170             mov    (8|M0)               r8.0<1>:d
0x0000000000092180             sends  (8|M0)               null:ud
0x0000000000092190  (W)        mov    (1|M0)               r5.14<1>:w
0x00000000000921a0  (W)        cmp    (8|M0)   [(eq)f0.0]  null<1
0x00000000000921b0  (W)        mov    (1|M0)               r9.0<1>:w
0x00000000000921c0  (W&f0.0)   sel    (1|M0)               r10.0<2>:b
0x00000000000921d0  (W)        mov    (1|M0)               r11.0<1>:d
0x00000000000921e0  (W)        add    (1|M0)               r14.0<1>:d
0x00000000000921f0  (W)        mul    (1|M0)               r13.0<1>:uw
0x0000000000092200  (W)        add    (1|M0)               a0.0<1>:uw
0x0000000000092210  (W)        mov    (1|M0)               r6.0<1>:d
0x0000000000092220  (W)        mul    (1|M0)               r7.0<1>:q
0x0000000000092230             mov    (8|M0)               r12.0<1>:d
```

Disassembly | Diagnostic Tools


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code Builder DEBUG

# Kernel Level Debugging on the GPU

Command Window | Watch 1

| Name | Value | Type |
|---|---|---|
| $r10.v8_int32 | [8] | v8i32 |
| $r25.v8_int32 | [8] | v8i32 |
| fSobel@8 | [8] | float4 [8] |
| $r23.v8_int32 | [8] | v8i32 |
| $r6.v8_int32 | [8] | v8i32 |
| X@8 | [8] | int [8] |
| [0] | 0 | int |
| [1] | 1 | int |
| [2] | 0 | int |
| [3] | 1 | int |
| [4] | 0 | int |
| [5] | 0 | int |
| [6] | 0 | int |
| [7] | 0 | int |
| Y@8 | [8] | int [8] |
| [0] | 0 | int |
| [1] | 0 | int |
| [2] | 1 | int |
| [3] | 1 | int |
| [4] | 0 | int |
| [5] | 0 | int |
| [6] | 0 | int |
| [7] | 0 | int |

Locals

| Name | Value | Type |
|---|---|---|
| __ocl_dbg_gid0 | 2 | unsigned long |
| __ocl_dbg_gid1 | 2 | unsigned long |
| __ocl_dbg_gid2 | 0 | unsigned long |
| __ocl_dbg_lid0 | 0 | unsigned long |
| __ocl_dbg_lid1 | 0 | unsigned long |
| __ocl_dbg_lid2 | 0 | unsigned long |
| __ocl_dbg_grid0 | 1 | unsigned long |
| __ocl_dbg_grid1 | 1 | unsigned long |
| __ocl_dbg_grid2 | 0 | unsigned long |
| X | 2 | int |
| Y | 2 | int |
| PixelCoords | [2] | int2 |
| fSobel | [4] | float4 |
| Sam | 1 | opencl_sampler_t |
| Input | 0x201000000000000 | struct opencl_image2d_t * |
| Mask | 0xa28c062000 | int * |

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code Builder DEBUG

# Kernel Level Debugging on the GPU

## Step by Step source level debugging of the GPU EU threads

Switch between GPU EU threads

Step by step kernel source-level debugger

Local and Global variables view

Assembly level debugging

Global memory view

Inspect GPU register values

```
__kernel void SobelFilter(sampler_t Sam,
                          __read_only image2d_t Input,
                          __write_only image2d_t Output,
                          __global int* Mask)
{
    int X = get_global_id(0);
    int Y = get_global_id(1);

    // Read image coordinates
    int2 PixelCoords = {X,Y};
    float4 fSobel = read_imagef(Input, Sam, PixelCoords);

    if ( Mask[X] == 1 )
    {
        float4 p[3][3];
        for (int horizontal = 0; horizontal < 3; ++horizontal) {
            for (int vertical = 0; vertical < 3; ++vertical) {
                int2 Coords  = {X+horizontal-1,Y+vertical-1};
                p[horizontal][vertical] = read_imagef(Input, Sam, Coords);
            }
        }

        float4 SobelH = p[2][0] - p[0][0] + 2 * (p[2][1] - p[0][1]) + p[2][2] - p[0][2];
        float4 SobelV = p[0][0] - p[0][2] + 2 * (p[1][0] - p[1][2]) + p[2][0] - p[2][2];
```

Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Analyze

# Performance Analysis with Code Builder

- 2 ways for performance analysis with the Code Builder
  - Kernel level analysis only with the Kernel Development Framework
  - Full application analysis (host + kernels)

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

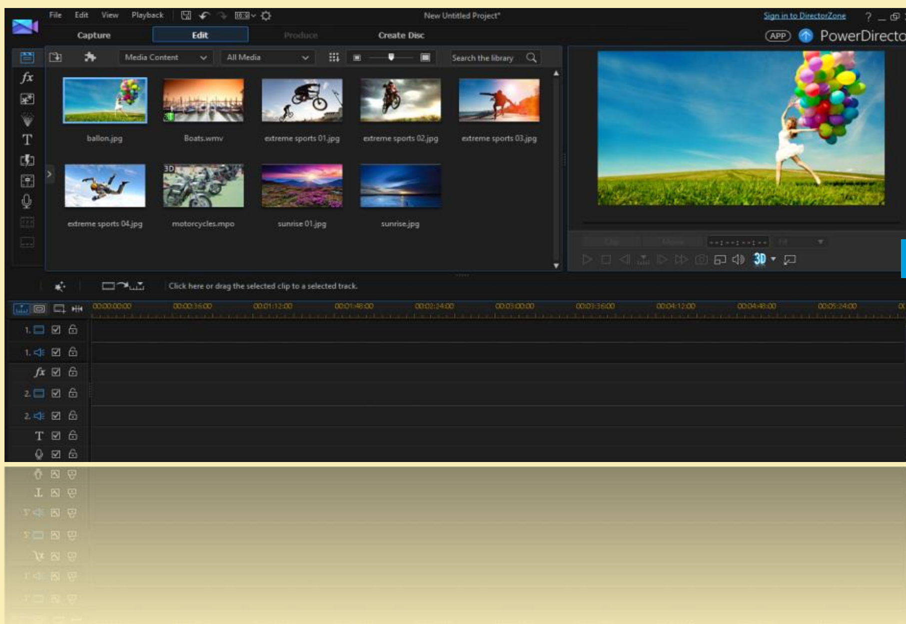
# Code Builder – kernel Analysis with KDF

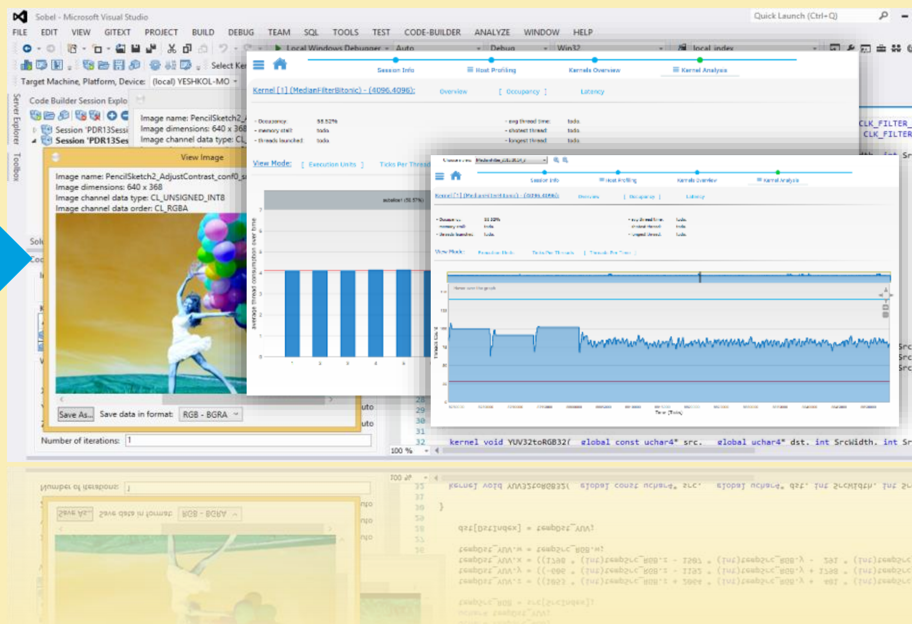
- Kernel Development Framework enable a standalone environment for performance analysis of kernels
  - Enables What-if analysis
  - Provides a lot of performance data:
    - Throughput
    - Memory bandwidth
    - GPU utilization
    - Occupancy
    - Latency for memory operation

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# DEMO – Performance Analysis with KDF

- Generate session from CyberLink Power Director*
  A High Performance Video Editing suite





Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# DEMO – Case Study

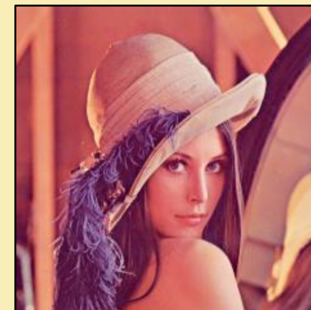
- Optimization of Sobel Kernel
  - Uchar -> Uchar16
  - Int -> float

Source Image

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

horizontal and vertical derivative approximations

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$



Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Code Builder - Full Application Analysis

- Guided performance debugging and source level analysis capabilities
  - a “wizard-like” profiling tool with runtime hints and drill down analysis (host to kernel)
  - Command line tool
  - Fully integrated to Visual Studio*

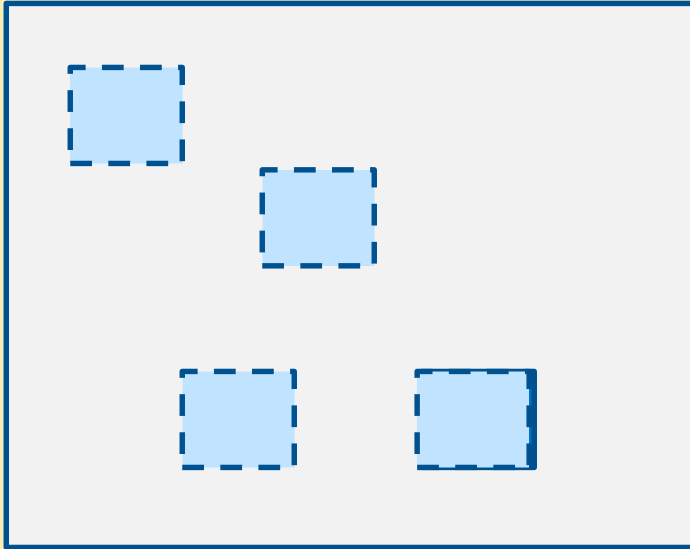
Session Info | Host Profiling | Kernels Overview | Kernel Analysis

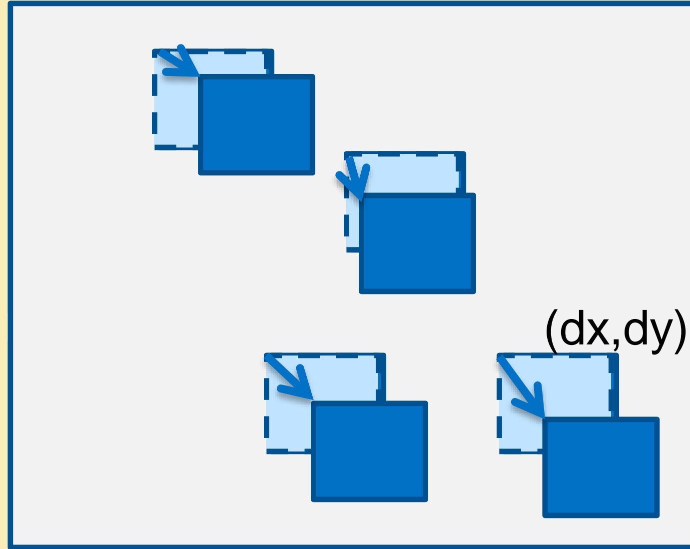
Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# DEMO – Full Application Analysis with Code Builder

Target application: Optical Flow (OpenCV* implementation)

Given a set of points in an image > find those same points in another image







Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# DEMO – Full Application Analysis with Code Builder

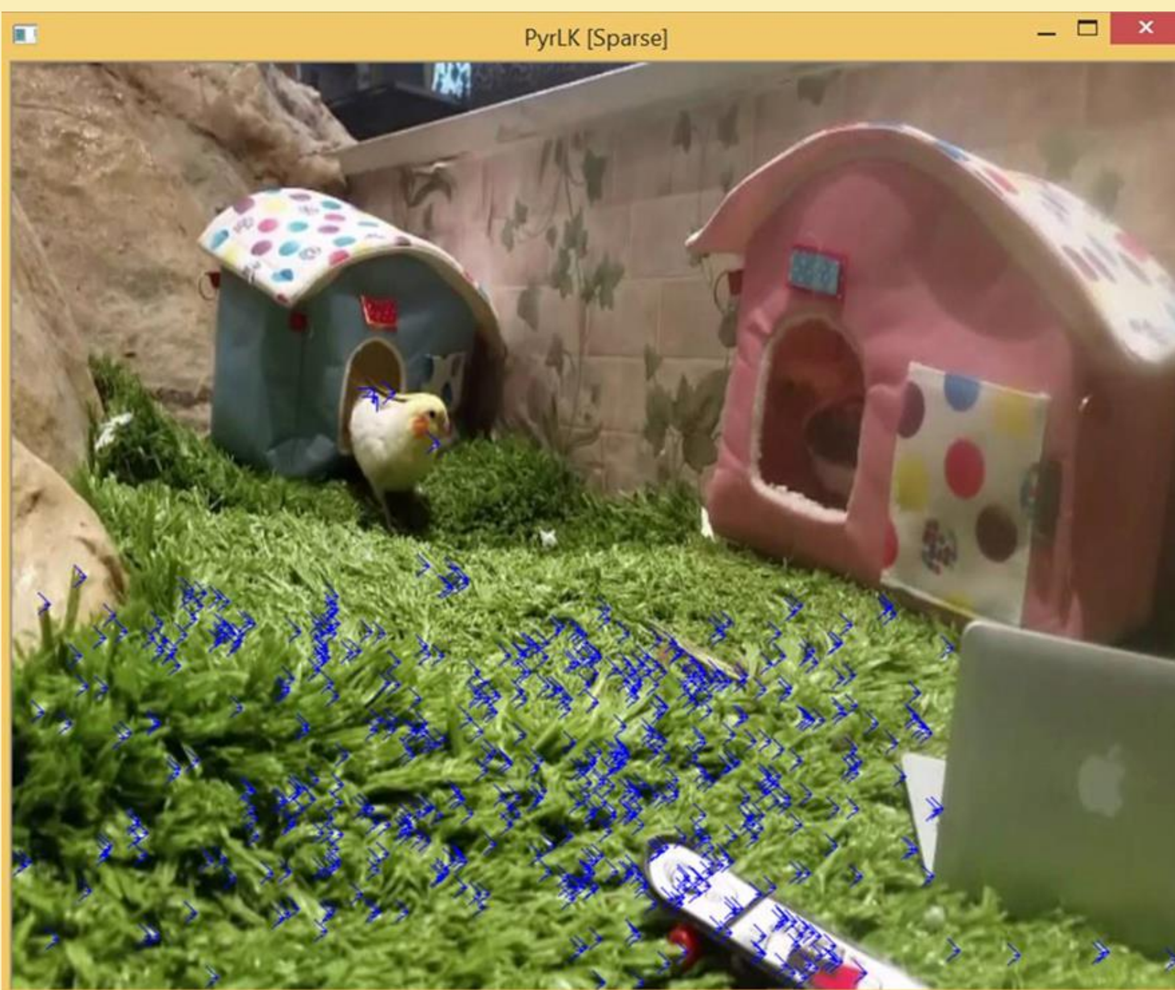
Target application: Optical Flow (OpenCV* implementation)

Given a set of points in an image > find those same points in another image

- Can be used to:
  - Find an object from one image in another
  - Determine how an object/camera moved
  - Resolve depth from a single camera.
  - More..
- Use a lot of OpenCL* kernels

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Code Builder
ANALYZE






Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

# Intel® VTune Amplifier XE

Code Builder ANALYZE

Process / Thread / Module / Function / Call Stack

| Process / Thread / Module / Function / Call Stack | CPU Time by Utilization (Idle, Poor, Ok, Ideal, Over) | Instructions Retired | Ov. an. | CPI Rate | CPU Freq.. | PID | TID | GPU Time by GPU Engine (Video Codec, Render and GPGPU) | Mo. |
|---|---|---|---|---|---|---|---|---|---|
| IEXPLORE.EXE | 10.326s | 6,321,000,000 | 0.119s | 3.668 | 1.125 | 0x137c | 0 | 12.542s | |
| sqlservr.exe | 4.140s | 6,631,800,000 | 0.031s | 1.506 | 1.209 | 0x81c | 0 | | |
| ntoskrnl.exe | 2.355s | 547,400,000 | 0s | 9.138 | 1.065 | 0 | 0 | | |
| Pid 0x4 | 2.255s | 1,408,400,000 | 0s | 3.328 | 1.042 | 0x4 | 0 | | |
| dwm.exe | 1.504s | 487,200,000 | 0.015s | 5.977 | 0.970 | 0x1514 | 0 | 0.660s | |
| Selected 1 row(s): | 10.326s | 6,321,000,000 | 0.119s | 3.668 | 1.125 | | | 12.542s | |

Processes

CPU

GPU

1s 2s 2.74s 4s 5s 6s 7s 8s 9s 10s 11s 12s 13s 14s 15

Thread

Thread (0x16b8)

Thread (0xe90)

Thread (0x1334)

Thread (0x15a8)

GPU Software Queue

GPU Usage

CPU Time

Overhead and Spin Time

3.4%

CPU Time

283.5%

Render and GPGPU

GPU Software Queue

GPU Software Queue

Video Codec

Render and GPGPU

GPU Usage

GPU Engines Usage

Video Codec

Render and GPGPU

CPU Time

CPU Time

No filters are applied. | Any Process | ...ad | Any Module | Utilization: Any Utilization

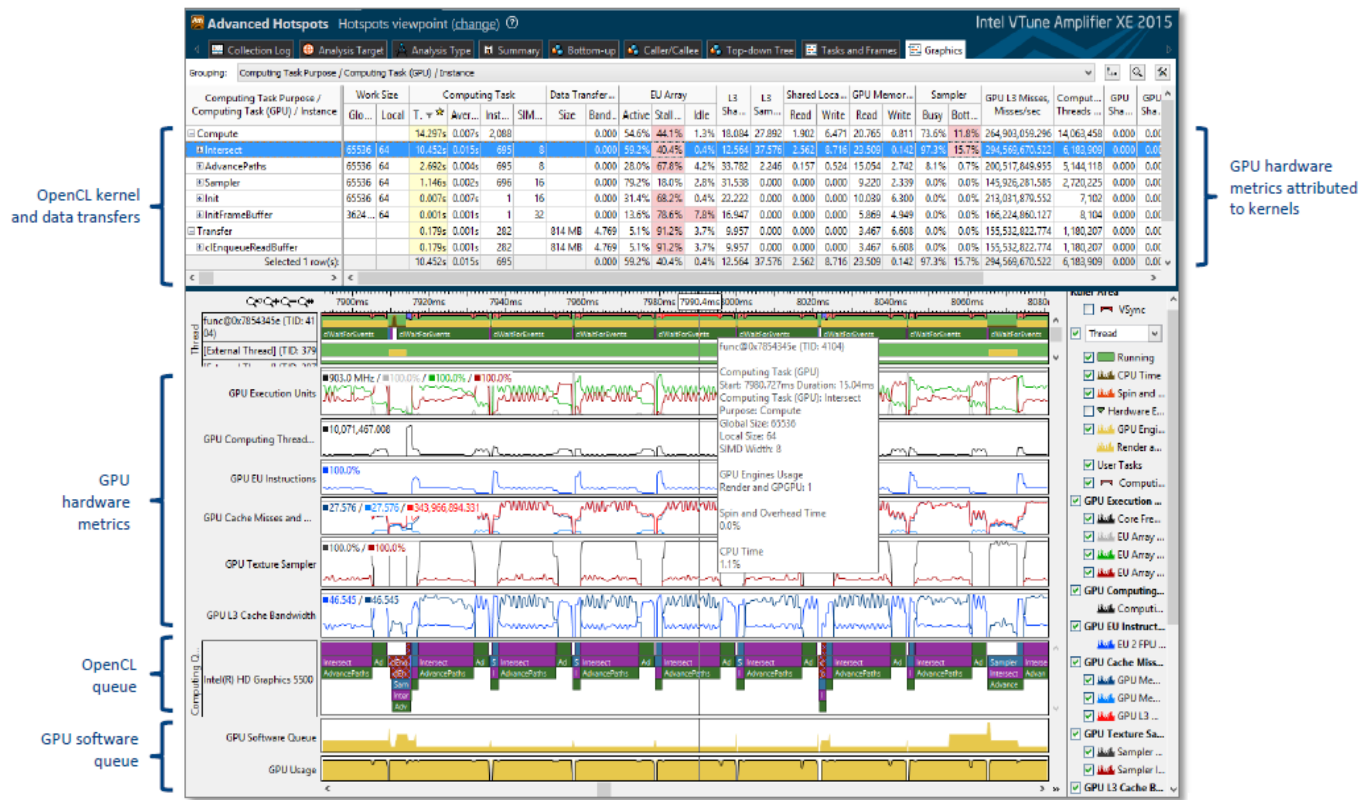
Call Stack Mode: Only user functions | Inli... | ... Mode: Functions only

VTune analysis of Microsoft* HTML5 Fish Bowl application http://ie.microsoft.com/testdrive/performance/fishbowl/

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Code Builder ANALYZE

# OpenCL™ Command Queue View




Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice

Grouping: Computing Task Purpose / Computing Task (GPU) / Instance

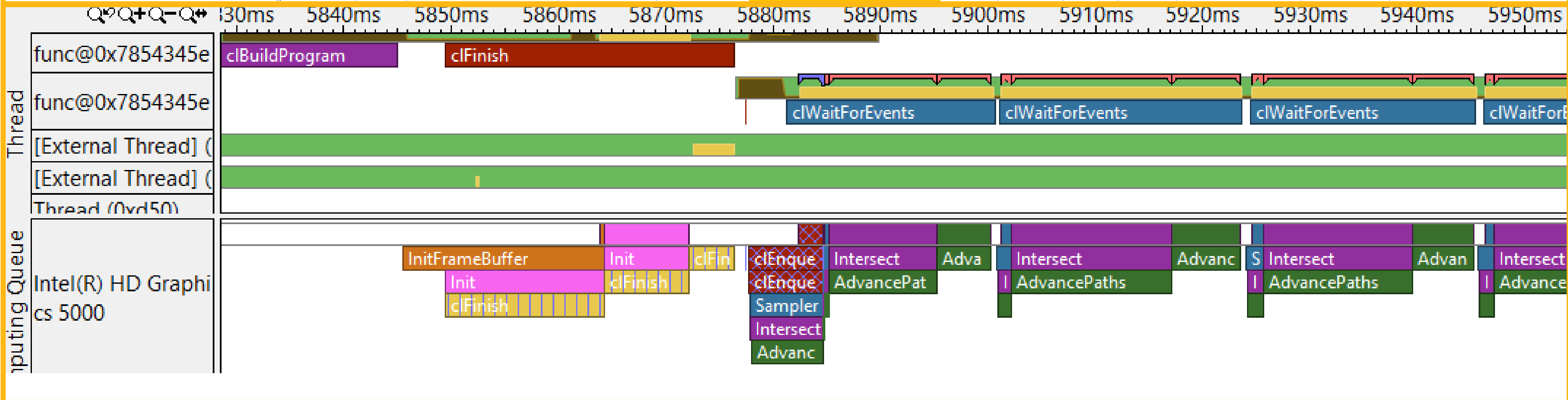
| Computing Task Purpose / Computing Task (GPU) / Insta... | Work Size | | Computing Task | | | EU Array | | | GPU Memor... | | Sampler | | GPU L3 | Comput... | GPU Time by |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Global | Loc.. | Total Time | Average ... | Instance... | Acti... | Stalled | Idle | Read | Write | Busy | Bott... | Misses, Miss... | Threads... | Render and GPGPU |
| ⊟Compute | | | 7.982s | 0.006s | 1,352 | 0.412 | 0.492 | 0.096 | 45.715 | 1.526 | 0.540 | 0.056 | 2,201,656,420 | 9,158,403 | 7.982s |
| ⊞Intersect | 65536 | 64 | 4.970s | 0.011s | 450 | 0.513 | 0.373 | 0.113 | 52.469 | 0.255 | 0.875 | 0.092 | 1,580,764,630 | 3,908,391 | 4.970s |
| ⊞AdvancePaths | 65536 | 64 | 2.347s | 0.005s | 450 | 0.156 | 0.811 | 0.034 | 36.354 | 3.449 | 0.036 | 0.001 | 503,007,946 | 3,453,242 | 2.347s |
| ⊞Sampler | 65536 | [illegible] | 0.656s | 0.001s | 450 | 0.599 | 0.207 | 0.194 | [illegible] | [illegible] | 0.002 | 0.000 | 115,744,896 | 1,770,202 | 0.656s |
| ⊞Init | 65536 | 64 | [illegible] | 0.008s | 1 | 0.277 | 0.701 | [illegible] | 35.002 | [illegible] | 0.000 | 0.000 | 2,138,948 | 26,568 | 0.008s |
| ⊞InitFrameBuffer | [illegible] | 64 | 0.000s | 0.000s | 1 | 0.000 | 0.000 | 0.000 | [illegible] | 0.000 | 0.000 | 0.000 | 0 | 0 | 0.000s |
| ⊟Transfer | | | 0.165s | 0.001s | 156 | 0.008 | 0.830 | 0.162 | [illegible] | 6.494 | 0.005 | 0.000 | 19,172,743 | 1,564,660 | 0.165s |
| ⊞clEnqueueReadBuffer | | | 0.165s | 0.001s | 156 | 0.008 | 0.830 | 0.162 | 6.908 | 6.494 | 0.005 | 0.000 | 19,172,743 | 1,564,660 | 0.165s |

Kernels

Work size

Timing

HW metrics




Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice

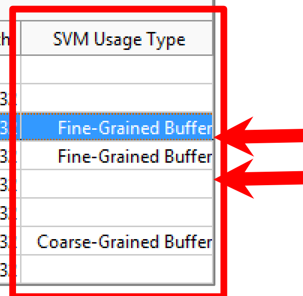
# SVM Usage Info

**GPU OpenCL Info**

| | |
|---|---|
| Version: | OpenCL C 2.0 |
| Max Compute Units: | 24 |
| Max Work Group Size: | 256 |
| Local Memory: | 64 KB |
| SVM Capabilities: | Fine-grained buffer with atomics |

Grouping: Computing Task Purpose / Computing Task (GPU) / Instance

| Computing Task Purpose / Computing Task (GPU) / Instance | Work Size | | Computing Task | | | | |
|---|---|---|---|---|---|---|---|
| | Global | Local | Total Time | Average Time | Instance Count | SIMD Width | SVM Usage Type |
| ⊟ Compute | | | 499.664s | 0.038s | 13,005 | | |
| ⊞ ReadWriteCopy_NoAlignPartWrite | 2097152 | 256 | 133.550s | 0.157s | 849 | 3 | |
| ⊞ ReadWriteCopy_NoAlignPartWrite | 2097152 | 256 | 61.267s | 0.072s | 850 | 3 | Fine-Grained Buffer |
| ⊞ ReadWriteCopy | 2097152 | 256 | 47.392s | 0.056s | 850 | 3 | Fine-Grained Buffer |
| ⊞ ReadWriteCopyUnRoll | 2097152 | 256 | 34.491s | 0.041s | 850 | 3 | |
| ⊞ ReadOnly | 2097152 | 256 | 34.422s | 0.020s | 1,700 | 3 | |
| ⊞ ReadWriteCopy | 2097152 | 256 | 32.639s | 0.038s | 850 | 3 | Coarse-Grained Buffer |
| ⊞ ReadWriteCopy | 2097152 | 256 | 31.976s | 0.038s | 850 | 3 | |

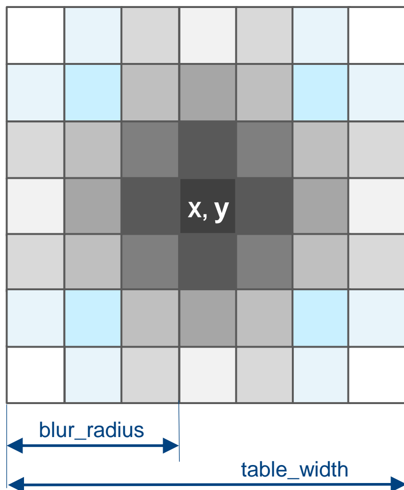


Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Gaussian Blur

## Naïve implementation

- Uses Sampler
- Process one pixel per a work-item




```
const sampler_t samplerA = CLK_FILTER_NEAREST ;

__kernel void gaussian_blur_naive(read_only image2d_t src,
                                  __global float* table,
                                  const int blur_radius,
                                  write_only image2d_t dst)
{
    float4 dst_val = { 0, 0, 0, 0}, src_val  = { 0, 0, 0, 0};
    int i, k, h, w ;

    int x = get_global_id(0);
    int y = get_global_id(1);

    int table_width = blur_radius*2 + 1;

    for (i = 0; i < table_width; ++i)
    {
        w = i - blur_radius;
        for (k = 0; k < table_width; ++k)
        {
            h = k - blur_radius;
            src_val = read_imagef(src, samplerA, (int2)(x + w, y + h));

            dst_val += src_val * table[i*table_width + k];
        }
    }
    write_imagef(dst, (int2)(x, y), dst_val);
}
```

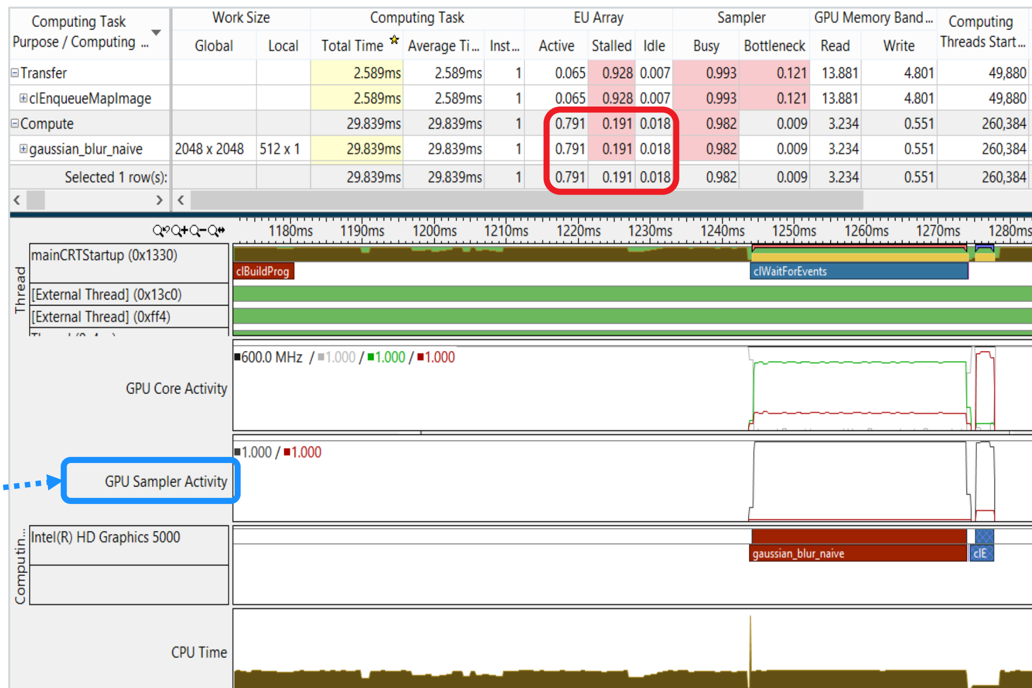
* Code source by Intel

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# What Can We Learn from VTune ?

```
const sampler_t samplerA = CLK_FILTER_NEAREST ;

__kernel void gaussian_blur_naive(read_only image2d_t src,
                                  __global float* table,
                                  const int blur_radius,
                                  write_only image2d_t dst)
{
    float4 dst_val = { 0, 0, 0, 0}, src_val  = { 0, 0, 0, 0};
    int i, k, h, w ;

    int x = get_global_id(0);
    int y = get_global_id(1);

    int table_width = blur_radius*2 + 1;

    for (i = 0; i < table_width; ++i)
    {
        w = i - blur_radius;
        for (k = 0; k < table_width; ++k)
        {
            h = k - blur_radius;
            src_val = read_imagef(src, samplerA, (int2)(x + w, y + h));

            dst_val += src_val * table[i*table_width + k];
        }
    }
    write_imagef(dst, (int2)(x, y), dst_val);
}
```

* Code source by Intel



| Computing Task Purpose / Computing ... | Work Size Global | Work Size Local | Computing Task Total Time | Average Ti... | Inst... | EU Array Active | EU Array Stalled | EU Array Idle | Sampler Busy | Sampler Bottleneck | GPU Memory Band... Read | GPU Memory Band... Write | Computing Threads Start... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Transfer | | | 2.589ms | 2.589ms | 1 | 0.065 | 0.928 | 0.007 | 0.993 | 0.121 | 13.881 | 4.801 | 49,880 |
| clEnqueueMapImage | | | 2.589ms | 2.589ms | 1 | 0.065 | 0.928 | 0.007 | 0.993 | 0.121 | 13.881 | 4.801 | 49,880 |
| Compute | | | 29.839ms | 29.839ms | 1 | 0.791 | 0.191 | 0.018 | 0.982 | 0.009 | 3.234 | 0.551 | 260,384 |
| gaussian_blur_naive | 2048 x 2048 | 512 x 1 | 29.839ms | 29.839ms | 1 | 0.791 | 0.191 | 0.018 | 0.982 | 0.009 | 3.234 | 0.551 | 260,384 |
| Selected 1 row(s): | | | 29.839ms | 29.839ms | 1 | 0.791 | 0.191 | 0.018 | 0.982 | 0.009 | 3.234 | 0.551 | 260,384 |

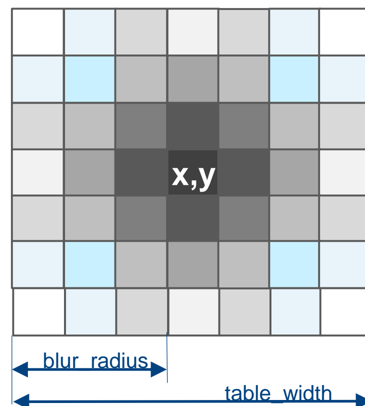


EUStalled ~ 0.2 => EUs are waiting 20% of the time

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

**Optimization Notice**

# Gaussian Blur: Can We Do Faster ?

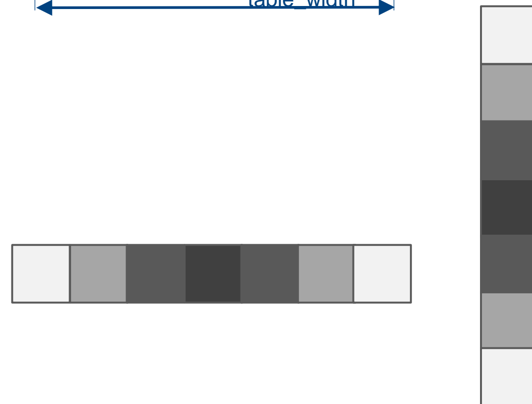



## Use memory buffers instead of images

- Memory buffers are faster to access then Sampler

## Take advantage of Gaussian Blur's separability property

- Two kernels (instead of one):
  - Horizontal pass
  - Vertical pass



Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Code given here just for a reference.
It is not recommended to use it after the Webinar

# Gaussian Blur: Two Passes

```
float4 _unpack_uchar4(uchar4 src)
{
    private uchar4 temp = src;
    float4 res;
    res.x = (float)temp.x;
    res.y = (float)temp.y;
    res.z = (float)temp.z;
    res.w = (float)temp.w;
    return res;
}
```

```
uchar4 _pack_float4(float4 src)
{
    uchar4 res;
    res.x = (uchar)src.x;
    res.y = (uchar)src.y;
    res.z = (uchar)src.z;
    res.w = (uchar)src.w;
    return res;
}
```

```
__kernel void gaussian_blur_hor_1(__global uchar4* src,
                                  __global float* table,
                                  const int blur_radius,
                                  __global  uchar4* dst)
{
    float4 dst_val = { 0, 0, 0, 0}, src_val  = { 0, 0, 0, 0};
    int x = get_global_id(0);
    int y = get_global_id(1);
    int image_width = get_global_size(0);

    for (int k = 0; k < blur_radius*2 + 1; ++k)
    {
        int w = x + k - blur_radius;
        if ( w >= 0 && w < image_width )
        {
            src_val = _unpack_uchar4(src[image_width*y + w]);
        }
        else if (w < 0)
        {
            src_val = _unpack_uchar4(src[image_width*y]);
        }
        else if (w >= image_width)
        {
            src_val = _unpack_uchar4(src[image_width*y + image_width-1]);
        }
        float4 mult = (float4)table[k];
        dst_val  += src_val * mult;
    }
    dst[y*image_width + x] = _pack_float4(dst_val);
}
```

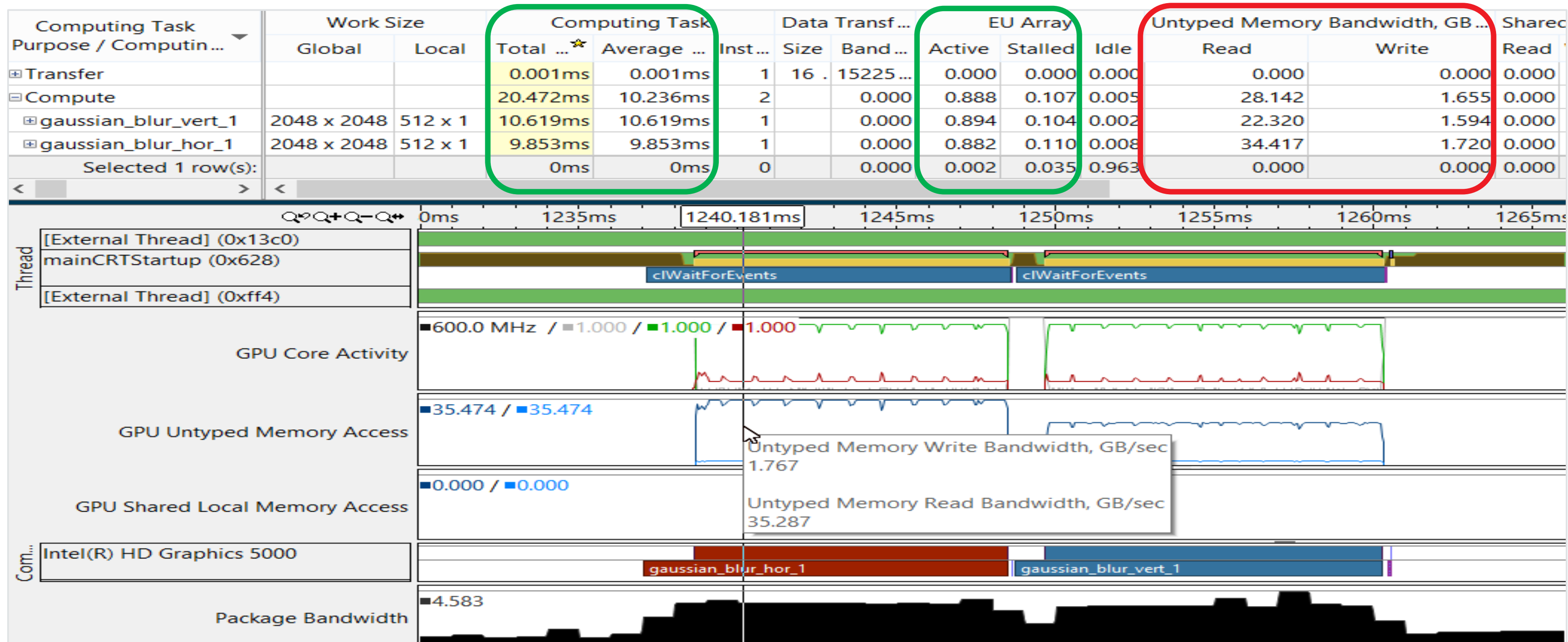
```
__kernel void gaussian_blur_vert_1(__global uchar4* src,
                                   __global float* table,
                                   const int blur_radius,
                                   __global uchar4* dst)
{
    float4 dst_val = { 0, 0, 0, 0}, src_val  = { 0, 0, 0, 0};
    int x = get_global_id(0);
    int y = get_global_id(1);
    int image_width  = get_global_size(0);
    int image_height = get_global_size(1);

    for (int k = 0; k < blur_radius*2 + 1; ++k)
    {
        int w = y + k - blur_radius;

        if ( w >= 0 && w < image_height )
        {
            src_val = _unpack_uchar4(src[image_width*w + x]);
        }
        else if (w < 0)
        {
            src_val = _unpack_uchar4(src[ x]);
        }
        else if (w >= image_height)
        {
            src_val = _unpack_uchar4(src[(image_width)*(image_height-1) + x]);
        }
        float4 mult = (float4)table[k];
        dst_val  += src_val * mult;
    }
    dst[y*image_width + x] = _pack_float4(dst_val);
}
```

* Code source by Intel

**Two passes give 21 ms of device time instead of 30 ms!**
**EUActive increased from 0.8 to 0.9**


Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Gaussian Blur: Two Passes




EU <-> L3 memory bandwidth is far from its peak value (~37 Gb/s vs. 150 Gb/s )

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

# Gaussian Blur Optimization Steps

| Kernel | Time, ms | EUActive | EUStalled | EUIdle | L3 Reads, Gb/s | L3 Writes, Gb/s |
|---|---|---|---|---|---|---|
| Naïve | 30 | 0.78 | 0.21 | 0.014 | N/A | N/A |
| Hor Pass Simple<br>1 pixel per work-item | 9.9 | 0.89 | 0.10 | 0.019 | 30 | 1.5 |
| Vert Pass Simple<br>1 pixel per work-item | 11 | 0.85 | 0.091 | 0.059 | 20 | 1.5 |
| Total* | 21.8 | | | | | |
| Hor Pass<br>4 pixels per work-item | 5.6 | 0.89 | 0.094 | 0.015 | 54 | 3.0 |
| Vert Pass<br>4 pixels per work-item | 5.8 | 0.68 | 0.68 | 0.006 | 25 | 2.9 |
| Total* | 13.5 | | | | | |
| Hor Pass<br>8 pixels per work-item | 4.6 | 0.93 | 0.064 | 0.006 | 68 | 7.3 |
| Vert Pass<br>8 pixels per work-item | 4.9 | 0.96 | 0.038 | 0.003 | 53 | 6.7 |
| Total* | 10.8 | | | | | |

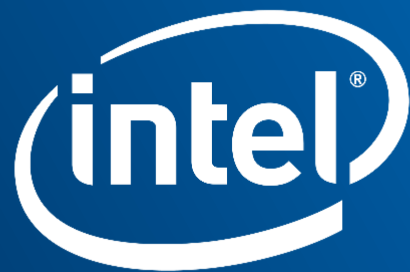
An increase of EUActive and L bandwidth

Sum of kernels duration + time between the kernels

Performance data where collected on Intel® 4th Generation Intel® Core™ Processor with Intel® HD Graphics 5000

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice


intel