Propel with OpenCL
A Deep Dive Workshop to Create, Debug, Analyze and Optimize OpenCL Applications using Intel® Tools

Anita Banerjee, Julia Fedorova, Uri Levy, Alexandr Kurylev, Sonal Sharma, Michael Stoner, Robert Ioffe
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. All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Intel processors, chipsets, and desktop boards 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.

Any 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 product plans in this presentation do not constitute Intel plan of record product roadmaps. Please contact your Intel representative to obtain Intel's current plan of record product roadmaps.

Performance claims: 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. For more information go to http://www.Intel.com/performance

Intel, Intel Inside, the Intel logo, Centrino, Intel Core, Intel Atom, Pentium, and Ultrabook are trademarks of Intel Corporation in the United States and other countries.
Agenda

• Intel® Graphics Overview
• The Intel® OpenCL™ Code Builder
• Intel® VTune™ Amplifier 2015
• Optimization Techniques and Examples
• OpenCL™ 2.0 Overview
• Summary / Questions
Agenda

• Intel® Graphics Overview ---- Presenter: Julia Fedorova
  • Intel® OpenCL™ Code Builder
  • Intel® VTune™ Amplifier 2015
  • Optimization Techniques and Examples
  • OpenCL™ 2.0 Overview
  • Summary / Questions
Intel® Processor Graphics?

- Intel® Processor Graphics: 3D Rendering, Media, and Compute
- Discrete class performance but... integrated on-die for true heterogeneous computing, SoC power efficiency, and a fully connected system architecture
- Some products are near TFLOPS performance
- Highly threaded, data parallel compute engine

Intel Processor Graphics is a key Compute Resource
Example Chip Level Architecture: Intel® Core™ M

Many different processor products, with different processor graphics configs

Multiple CPU cores, shared LLC, system agent

Multiple clock domains, target power where it’s needed
EU: The Execution Unit

1. Gen8: Seven hardware threads per EU
2. 128 general purpose registers per thread
   - 4K registers/thread or 28K/EU
   - Each register: 32 bytes wide
     - 8 x 32b floats, 8 x 32b integers
     - 16 x 16b half-floats, 16 x 16b shorts
3. Thread Arbiter picks instructions to run from runnable thread(s)
   - Each cycle: can co-issue multiple instructions, from up to four different threads
   - Dispatches instruction to appropriate functional unit

The fine grain threaded nature of the EUs ensures continuous streams of ready to execute instructions, while also enabling latency hiding of longer operations such as memory scatter/gather, sampler requests, or other system communication.
Subslice: An Array of 8 EU’s

Each: Subslice

① Eight Execution Units

② Local Thread Dispatcher & Inst $\$

③ Texture/Image Sampler Unit:
  • Includes dedicated L1 & L2 caches
  • Dedicated logic for dynamic texture decompression, texel filtering, texel addressing modes
  • 64 Bytes/cycle read bandwidth

④ Data Port:
  • General purpose load/store memory unit
  • Memory request coalescence
  • 64 Bytes/cycle read & write bandwidth
Slice: 3x Subslices

Each Slice: 3 x 8 = 24 EU’s
- 3 x 8 x 7 = 168 HW threads/slice

① Dedicated interface for every sampler & data port

② Level-3 (L3) Data Cache:
- Typically 384 KB / slice, though allocations are app reconfigurable
- 64 byte cachelines
- Monolithic, but distributed cache
- 64 bytes/cycle read & write

③ Shared Local Memory:
- 64 KB / subslice
- More highly banked than rest of L3

④ Hardware Barriers, 32bit atomics
Product Configuration Examples

12 EUs

24 EUs

48 EUs
OpenCL™ Execution Model maps to Intel® Graphics Architecture

OpenCL* Kernels run on an Execution Unit (EU)

Each EU is a Multi-Threaded SIMD Processor

Up to 7 threads per EU

- 128 x 8 x 32-bit registers per thread

Up to 8, 16, or 32 OpenCL* work items per thread (compiler-controlled)

- “SIMD8”, “SIMD16”, “SIMD32”
- SIMD8 → More Registers
- SIMD16 and SIMD32 → Better Efficiency
OpenCL™ Execution Model maps to Intel® Graphics Architecture
Memory Hierarchy and Sharing

- Intel® Processor Graphics has full performance access to system memory
- “Zero Copy” CPU & Graphics data sharing
- Shared Virtual Memory – new in Gen8

Facilitated by OpenCL™ 2.0 Shared Virtual Memory:
- Coarse & fine grained SVM
- CPU & GPU atomics as synchronization primitives
- System SVM as soon as OSVs are ready
Agenda

- Intel® Iris™ Graphics Overview
- The Intel® OpenCL™ Code Builder ---- Presenter: Uri Levy
- Intel® VTune™ Amplifier 2015
- Optimization Techniques and Examples
- OpenCL™ 2.0 Overview
- Summary / Questions
### OpenCL™ Code Builder

A comprehensive developers’ tool-chain for OpenCL™ and Intel® Graphics compute

**Supports Each State of the OpenCL™ Code Development**

<table>
<thead>
<tr>
<th>Getting Started</th>
<th>Build</th>
<th>Debug</th>
<th>Tune</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify opportunities for accelerations</td>
<td>Quickly build OpenCL code</td>
<td>Debug your code – both functional and performance</td>
<td>Tune the application</td>
</tr>
<tr>
<td>- Code Sample</td>
<td>- Kernel development framework</td>
<td>- API level debugging</td>
<td>- Application level profiling</td>
</tr>
<tr>
<td>- Case Studies</td>
<td>- OpenCL 2.0 development environment</td>
<td>- API calls and commands tracing and analysis</td>
<td>- Platform level timeline view</td>
</tr>
<tr>
<td>- Identify performance bottlenecks in applications</td>
<td>- Pre-defined projects</td>
<td>- Step-by-step kernel debugging (preview)</td>
<td>- Platform performance counters</td>
</tr>
<tr>
<td>- Redesign algorithms</td>
<td>- IDE integration – auto compilation, syntax highlighting</td>
<td>- Kernel statistics views</td>
<td>- Correlate CPU &amp; GPU activities</td>
</tr>
<tr>
<td>* This stage is largely manual</td>
<td>- Offline compilation with errors reports</td>
<td>- Memory debug</td>
<td>- Kernel hot-spots analysis</td>
</tr>
</tbody>
</table>

**Carry-on performance optimizations in each step of the development**
OpenCL™ Code Builder

What Is New in Version 2015?

**OpenCL™ Code Builder fully integrated in Intel’s development suites**
- Available with Intel® Integrated Native Developer Experience (Intel® INDE)
- Available with Intel® Media Server Studio
- Advanced new editing and debugging features with Microsoft* Visual Studio plug-in
- Free Windows & Android development with Intel INDE starter edition

**Commercial OpenCL 1.2 Linux* driver for Intel® Graphics**
- Available with Intel® Media Server Studio

**OpenCL 2.0 on 5th Generation Intel® Core™ Processors**
- Fine-grained shared virtual memory (SVM) support
- Support Intel® HD Graphics 5500/6000 and Intel® Iris™ Graphics 6100
- Available with Intel® Integrated Native Developer Experience (Intel® INDE)

Where did The Intel® SDK for OpenCL™ Applications go?

- Intel® SDK for OpenCL™ Applications is now available as OpenCL™ Code Builder
- All SDK’s capabilities are now integrated into Intel’s suites for developers through the OpenCL™ Code Builder.
- With new suite support, developer now gets OpenCL Code Builder, profiling features, and interoperable products in a single place.
- A standalone Intel® Code Builder for OpenCL™ API is available for support configurations that are not available with the integrated suites
  - E.g Ubuntu* for CPU, Intel® Xeon Phi™ coprocessor, and more
Intel's Portfolio of Tools for OpenCL™ Development

Maximize the power of the platform with OpenCL™ and Intel® Graphics Compute

- Build high-performance applications
- Optimizing tasks with standard APIs and best available compute engines
- Tap into a comprehensive developers' tool-chain, IDE integration, and more.

Intel® INDE
- For mobile and PC client applications
- Create & Debug with OpenCL™ Code Builder
- Analyze with INDE Analyze capabilities
- Supports:
  - OpenCL 1.2 & 2.0
  - Windows*, Android*
  - Intel® Core™ & Atom™
  - Intel® Graphics Compute

Intel® Media Server Studio
- For enterprise media solution
- Create & Debug with OpenCL™ Code Builder
- Analyze with Intel® VTune™ Amplifier XE
- Supports:
  - OpenCL 1.2 & 2.0
  - Linux*, Windows*
  - Intel® Xeon® E3 & Core™ i7
  - Intel® Graphics Compute

Intel continues to support Intel Xeon® E5 & E7, and Intel Xeon Phi through a standalone Intel® Code Builder for OpenCL™ API

Don't leave performance on the platform!
## Which Suite To Download?

<table>
<thead>
<tr>
<th>Tool suite</th>
<th>Intel® INDE</th>
<th>Intel® Media Server Studio</th>
<th>Intel® Code Builder for OpenCL™ API for Linux*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Supported devices</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel® Graphics (GPU)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Intel® processors (CPU)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Intel® Xeon Phi™ coprocessors</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>Target OS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows*</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Android*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux*</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>Host OS (Development environment)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows*</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Android*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux*</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>IDE Integration</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft Visual Studio*</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Eclipse*</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Standalone UI</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Select the tool that best fit your target applications and OS matrix:
- PC & Mobile applications → Use Intel® INDE
- Enterprise media applications → Use Intel® Media Server Studio
- HPC apps → Use Intel® Code Builder for OpenCL™ APIs
Tools for OpenCL™ Development

The Development Flow

When I CREATE my OpenCL Code

When I first ANALYSE my Application

When I TUNE my entire system

Don’t leave performance on the table!
Get optimized code faster - use performance tools during each step of the development
OpenCL™ Code Builder
Development Tools Features
## OpenCL™ Code Builder - Features and Support Matrix

**Development Environment**

<table>
<thead>
<tr>
<th>VISUAL STUDIO*</th>
<th>ECLIPSE*</th>
<th>STANDALONE</th>
<th>FEATURES:</th>
<th>PREVIEW FEATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>OpenCL™ 1.2 support</td>
<td>Create &amp; Build</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>OpenCL™ 2.0 support with Intel® Core™ M and 5th Gen Intel® Core™ Processors</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>OpenCL™ 2.0 development environment on previous CPU generations</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>Kernel Development Framework</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>New OpenCL Project wizard</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>Syntax highlighting</td>
<td>Debug</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>Code auto completion</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>Offline compilation</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>SPIR* 1.2 generation and consumption</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>Remote development for Android*</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>API-level debugging</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>Image and memory view</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>API calls tracing</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>Step-by-step debugging for CPU kernels</td>
<td>Analyze</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>Step-by-step debugging for GPU kernels</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>API calls and memory command analysis</td>
<td></td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>Kernel occupancy and latency analysis</td>
<td></td>
</tr>
</tbody>
</table>

*VISUAL STUDIO®, ECLIPSE®, STANDALONE® are trademarks of Microsoft Corporation, Eclipse Foundation, and Oracle Corporation, respectively.

SPIR® (Standard Portable Intermediate Representation) is a trademark of the Khronos Group Inc.
Code Editing and Compiling

- IDE integration (Visual Studio and Eclipse)
- Offline compilation and binary generation of OpenCL™ kernels
- Syntax checking and compile error reports.
- Project wizards
- Offline build for the Android* target.
Kernel Development Framework

- Create and build OpenCL Kernels on standalone environment
- Assign variables to the kernel and check its correctness
- Show the input and output values
- Analyze kernel’s performance with “What-if” analysis on work group sizes
- Remote development on Android devices
API Level Debugging

- Seamless debugging of OpenCL™ API calls, objects, and queues
- Enables monitoring and understanding the OpenCL environment of an application throughout execution
- OpenCL API calls tracing
- Images and memory objects view
Objects Tree View
Explore all OpenCL Objects in memory and their properties

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

Image View
Show the visualized content of OpenCL Images Objects

Date View
Show the content of OpenCL Memory Objects (Buffers + Images)

Properties View
View the properties of the selected OpenCL objects
Kernel Level Debugging on the CPU

- Step into Kernels running on the CPU
- Supports existing debugging capabilities
  - Breakpoints
  - Memory view
  - Watch variables – including OpenCL types like float4, int4, etc.
- Call stack
- Auto and local variables views
Code Builder – Application Analysis

- **Host level analysis**
  - Identify performance bottlenecks in the API calls
  - Optimize the host code to reduce API execution time and kernels run time

- **Kernel level analysis**
  - Optimize the kernel code to get better utilization and reduce the latency
  - Measure compute metrics (latency, and utilization)
Demo Session
Introduction with OpenCL™ Code Builder
A Walkthrough
Create, Build and Analyze my OpenCL kernel with **Kernel Development Framework**

Debug my OpenCL host application and Kernel with **OpenCL® Debugger**

Analyze and Optimize your OpenCL application and kernel code with **OpenCL® Code Analyzer**
Create, Build and Analyze my OpenCL kernel with **Kernel Development Framework**

Debug my OpenCL host application and Kernel with **OpenCL® Debugger**

Analyze and Optimize your OpenCL application and kernel code with **OpenCL® Code Analyzer**
Create, Build and Analyze your OpenCL Kernel with Kernel Development Framework

Our case study: **Sobel Filter Kernel**

- Edge detection algorithm
- Discrete differentiation operator, computing an approximation of the gradient of the image intensity function

\[
G_x = \begin{bmatrix}
+1 & 0 & -1 \\
+2 & 0 & -2 \\
+1 & 0 & -1
\end{bmatrix} \ast A \quad \text{and} \quad G_y = \begin{bmatrix}
+1 & +2 & +1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{bmatrix} \ast A
\]

Source Image

horizontal and vertical derivative approximations

\[
G = \sqrt{G_x^2 + G_y^2}
\]
Create, Build and Analyze my OpenCL kernel with **Kernel Development Framework**

Debug my OpenCL host application and Kernel with **OpenCL® Debugger**

Analyze and Optimize your OpenCL application and kernel code with **OpenCL® Code Analyzer**
Debug OpenCL host applications and kernel code with OpenCL Debugger

Our case study: Sobel Filter App
Create, Build and Analyze my OpenCL kernel with Kernel Development Framework

Debug my OpenCL host application and Kernel with OpenCL® Debugger

Analyze and Optimize your OpenCL application and kernel code with OpenCL® Code Analyzer
Analyze OpenCL host applications and kernel code with OpenCL Code Analyzer

1st case study: Host level optimization on trivial “Hello World” application
- Serial execution of 4 basic compute workloads (OpenCL kernels)
  - Non optimized host code
  - 4 optimization step
1st optimization – Wrong API Usage
Avoid redundant usage of API calls

cBuildProgram calls takes ~40% of total execution time
1st optimization – Wrong API Usage
Avoid redundant usage of API calls

- Inefficient "clCreateBuffer" calls.
  The host program includes 10 calls to "clCreateBuffer" where "fbuf" includes "CL_MEM_COPY_HOST_PTR".
- 4 redundant calls to "clCreateCommandQueue".
  The host program includes 5 calls to "clCreateCommandQueue" with the same arguments.
- 4 redundant calls to "clCreateContextFromType".
  The host program includes 5 calls to "clCreateContextFromType" that refer to the same device, "Device [1] (Intel(R) HD Graphics 4000)".
- "clEnqueueReadBuffer" calls.
  The host program includes 5 calls to "clEnqueueReadBuffer".

The work-group dimensions are defined as "column" work-group.
The host program includes 1 call to "clEnqueueNDRangeKernel" for kernel "Kernel [1] (Add)" where the work-group dimensions are defined as "column" work-group.

Consider using the same OpenCL context instead of recreate it.
2nd optimization - Memory Access Patterns

Row memory access VS column memory access

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

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

<table>
<thead>
<tr>
<th>LocalWorkSize[2] = {1,16}</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x,y} =</td>
</tr>
<tr>
<td>y * width + x =</td>
</tr>
<tr>
<td>0 0 1 1 2 2 3 3 ... 0 15</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>LocalWorkSize[2] = {16,1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x,y} =</td>
</tr>
<tr>
<td>y * width + x =</td>
</tr>
<tr>
<td>0 0 1 1 2 2 3 3 ... 15</td>
</tr>
</tbody>
</table>

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).
2nd optimization - Memory Access Patterns
Row memory access VS column memory access

Row memory access:
- 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 come from a single L3 cache line (16 x sizeof(int) = 64 bytes).

Column memory access:
- 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.

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

When reading from memory, best to reorganize the work-group to read in lines instead of columns.
3rd optimization - Host to Device Transfers

- **clCreateBuffer**
  - CL_MEM_USE_HOST_PTR flag enables the application to share its memory allocation with the OpenCL™ runtime implementation, and avoid memory copies of the buffer.

```c
clCreateBuffer(..., CL_MEM_USE_HOST_PTR, ...);
...
clEnqueueReadBuffer(...);
```
3rd optimization - Host to Device Transfers

buffer created by malloc()

buffer created by clCreateBuffer(..., CL_MEM_COPY_HOST_PTR, ...)

There are two ways to ensure zero-copy paths 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 swap memory yourself and share the pointer with the OpenCL™ runtime by using the "CL_MEM_USE_HOST_PTR" flag.

When possible, use "clEnqueueMapBuffer" and "clEnqueueUnmapMemObject" instead of call to "clEnqueueReadBuffer" or "clEnqueueWriteBuffer".
Host level optimization – Summary

On Intel® Iris™ Graphics 5100

1st Optimization:
Avoid redundant usage of API calls

2nd Optimization:
Row memory access VS column memory access

3rd Optimization:
Host to Device Transfers

- 1st Optimization: 2568 ms
- 2nd Optimization: 1330 ms (X2.35)
- 3rd Optimization: 1187 ms (X2)

- 1st Optimization: 968 ms (X1.2)
Analyze OpenCL host applications and kernel code with OpenCL Code Analyzer

2nd case study: Kernel level analysis with:

- **Occupancy View** – how much the GPU is busy
  
  Higher is better

- **Latency View** - execution time of each kernel instruction (especially memory)
  
  Lower is better
Download, Learn, Code, Optimize

- Free download at: intel.com/software/opencl
- Follow us: @IntelOpenCL
- Contact as through our forum:

Try related products
- Native client development with Intel® Integrated Native Developer Experience (Intel® INDE)
- Performance tuning with the Intel® VTune™ Amplifier XE
- Media performance with the Intel® Media SDK

What is available online?
- Free Downloads
- Code Samples
- Documentation
- Tech Articles
- Reviews
- Forums and Support
- Webinars
Agenda

• Intel® Iris™ Graphics Overview
• Intel® OpenCL™ Code Builder
• Intel® VTune™ Amplifier 2015 ---- Presenter: Alexandr Kurylev
• Optimization Techniques and Examples
• OpenCL™ 2.0 Overview
• Summary / Questions
What We Are Going To Talk About

• Introduction into GPU Analysis for OpenCL* applications with Intel® VTune™ Analyzer XE
• New features in recent releases
• Case study: an OpenCL* kernel optimization
• Summary / Questions
OpenCL kernel and data transfers

GPU hardware metrics attributed to kernels

GPU hardware metrics

OpenCL queue

GPU software queue

GPU Analysis Features Overview

- Intel VTune Amplifier is a powerful performance debugging tool with mature GPU profiling capabilities.
- Shows host and GPU activities correlated.
- OpenCL kernel queue graphical view.
- Allows to see kernels and their characteristics.
- Goes down to hardware level by showing both GPU and CPU hardware metrics.
Metric Presets For Intel® Graphics Analysis

- “Overview”: useful for Graphics and Compute and indeed, provides an overview
- “Compute Basic”: details about compute as Occupancy, IPC, FPUs Active etc.
- “Compute Extended”: Memory Access and Coalescence metrics
Architecture Diagram

Shows GPU blocks & CPU utilization and Uncore bandwidth while specific GPU task was running.
Case Study: Gaussian Blur Kernel

- Convolves the image with Gaussian function
  \[ G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}} \]
- Reduces noise and details
- Good representative of image processing kernels

The sample matrix, produced by sampling the Gaussian filter kernel ($\sigma = 0.840896$) at the midpoints of each pixel and then normalizing:

0.0000067 0.0002292 0.00019117 0.00039771 0.00019117 0.0002292 0.0000067
0.0002292 0.00078634 0.00655965 0.0130373 0.00655965 0.00708633 0.0002292
0.00019117 0.00655965 0.05472157 0.11098164 0.05472157 0.0655965 0.00019117
0.00039771 0.0130373 0.11098164 0.22508352 0.11098164 0.0130373 0.00039771
0.00019117 0.00655965 0.05472157 0.11098164 0.05472157 0.0655965 0.00019117
0.0002292 0.00078634 0.00655965 0.0130373 0.00655965 0.00708633 0.0002292
0.0000067 0.0002292 0.00019117 0.00039771 0.00019117 0.0002292 0.0000067

Source: http://en.wikipedia.org/wiki/Gaussian_blur
Naïve implementation

- Uses Sampler
- Processes one pixel per a work-item
What Can We Learn From The Tool?

* Code source by Intel

**EU Stalled ~ 25% → EUs are waiting 25% of the time**
Optimization Considerations

25% stalls due to Sampler accesses

However:

- regular access pattern allows using plane buffers instead of images (memory buffers are faster to access than Sampler)
- can use Gaussian Blur’s separability property
  - make two kernels (instead of one): horizontal and vertical pass
  - access image data from linear buffers

\[
G(x, y) = \frac{1}{2\pi \sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}
\]
Gaussian Blur: Two Passes

Taking advantage of Gaussian Blur’s separable property

```c
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;
}

_kernel void gaussian_blur_hor_1(__global read_only uchar4* src,  
                               __global read_only float* table,  
                               const int blur_radius,  
                               __global write_only 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]);
        }  
        dst_val += src_val * table[k];
    }  
    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[w]);
        }  
        else if (w >= image_height)
        {
            src_val = _unpack_uchar4(src[image_width*(image_height-1) + x]);
        }  
        dst_val += src_val * table[k];
    }  
    dst[y*image_width + x] = _pack_float4(dst_val);
}
```

Memory Access Pattern

Read/Write from/to memory by 4 bytes
Optimizes memory access and makes it coalesced

64 bytes == 1 cache line (!)
Two Passes. Is more speed-up possible?

- EU↔L3 memory bandwidth is far from its peak value (43 + 28 = 71 vs 128 GB/s for 2 slices @ 1 GHz)
- Process more pixels in one work item

Use available memory bandwidth to speed up the kernels
Gaussian Blur Optimization Steps

Gaussian Blur Performance Data

Hardware metrics guide optimization of GPU bound code
What To Optimize For

<table>
<thead>
<tr>
<th>Optimization Target</th>
<th>Metric(s) to Watch</th>
<th>Traps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Throughput</td>
<td>Untyped Read/Write $\rightarrow$ for 128 GB/s at 1 GHz per 1 slice</td>
<td>Non-coalesced accesses might consume additional bandwidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Occupancy</td>
<td>Kernels with too small work for working item</td>
<td>More EU threads is not good when the working item is doing too small work</td>
</tr>
<tr>
<td></td>
<td>EU Idle $\rightarrow$ 0 %</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EU Stalled $\rightarrow$ 0 %</td>
<td></td>
</tr>
<tr>
<td>Compute Throughput</td>
<td>EU Active $\rightarrow$ 100 %</td>
<td>Redundant calculation might elevate EUActive</td>
</tr>
<tr>
<td></td>
<td>EU Stalled $\rightarrow$ 0 %</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EU Idle $\rightarrow$ 0 %</td>
<td></td>
</tr>
</tbody>
</table>
## Operating Systems Support

<table>
<thead>
<tr>
<th>Feature</th>
<th>Window*</th>
<th>Linux*</th>
<th>Android*</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU usage, per engine</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPU usage items attributed to OpenCL</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GPU Hardware Metrics</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Media Server Studio CPU-side APIs support</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>OpenCL 1.2 support</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>OpenCL 2.0 basic support</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GPU Architecture Diagram</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary

Use VTune to analyze OpenCL* applications running on Intel® Graphics

- Watch for hot kernels and possible inefficient CPU↔GPU interactions

Optimize Hottest OpenCL* Kernels using Intel Graphics Hardware Metrics

Watch for

- Memory Access Pattern
- Occupancy
- EU utilization

VTune helps use full potential of Intel Iris Graphics with your OpenCL* application
Agenda

- Intel® Iris™ Graphics Overview
- Intel® OpenCL™ Code Builder
- Intel® VTune™ Amplifier 2015
- Optimization Techniques and Examples ---- Presenter: Anita Banerjee
- OpenCL™ 2.0 Overview
- Summary / Questions
Optimization Techniques and Examples

- Host Side Optimizations
- Memory Matters
  - Host to Device
  - Device Access
- Compute Characteristics
  - Maximizing Gflops
- Device Side Optimizations
OpenCL* Host Side Optimizations

- Pre-compile kernels if possible
  - Compile once and save binary – load at app start

- Enqueue multiple commands in queue
  - Use in-order queues
  - No need to wait or “clFinish()” on every kernel
  - Allows OpenCL runtime to minimize overhead

- Use null for LWS
  - Let driver to choose the best LWS for you if you are not sure
Optimizing Host to Device Transfers

Host (CPU) and Device (GPU) share the same physical memory

For OpenCL* buffers:
- No transfer needed (zero copy)!
- Allocate memory aligned to a cache line (64 bytes) and multiple of 4KB (page size)
- Create buffer with system memory pointer and CL_MEM_USE_HOST_PTR
- Use `clEnqueueMapBuffer()` to get pointer to access data from CPU
- Use `clEnqueueUnmapMemObject()` to give the pointer back to GPU before using at kernels.

For OpenCL* images:
- Use cl_khr_image2d_from_buffer Ext.
Adjacent work items should ideally read/store adjacent memory locations

\[ x = \text{data}[\text{get\_global\_id}(0)] \]
- One cache line, full bandwidth

Especially avoid the work items reading/storing skipping memory or vertically

\[ x = \text{data}[\text{get\_global\_id}(0) \times 2] \]
- Strided, half bandwidth

\[ x = \text{data}[\text{get\_global\_id}(0) \times 16] \]
- Very strided, worst-case
__global and __constant Memory

Global memory access performance depends of size and alignment.

Best: Load/Store 16 bytes of data at a time, starting from a cache line aligned address

OK: Load/Store at least 4 bytes of data at a time, starting from a 4 bytes aligned address
### Local Memory

<table>
<thead>
<tr>
<th></th>
<th>L3$</th>
<th>SLM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>1 cache line</td>
<td>16 banks</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>Full bandwidth</td>
<td>Full bandwidth</td>
</tr>
<tr>
<td>Example</td>
<td>data[get_global_id(0)];</td>
<td>data[get_local_id(0)];</td>
</tr>
<tr>
<td></td>
<td>data[get_global_id(0) + 1];</td>
<td>data[get_local_id(0) + 1];</td>
</tr>
<tr>
<td></td>
<td>data[get_global_id(0) * 2];</td>
<td>data[get_local_id(0) * 2];</td>
</tr>
<tr>
<td></td>
<td>data[get_global_id(0) * 16];</td>
<td>data[get_local_id(0) * 16];</td>
</tr>
<tr>
<td></td>
<td>data[get_global_id(0) * 17];</td>
<td>data[get_local_id(0) * 17];</td>
</tr>
</tbody>
</table>

**When picking a memory type, consider access patterns!**
private Memory

Compiler can *usually* allocate Private Memory in the Register File

- Even if Private Memory is dynamically indexed
- Good Performance

Fallback: Private Memory allocated in Global Memory

- Accesses are very strided
- Bad Performance
Intel® HD Graphics Memory Hierarchy

- Intel® HD Graphics Register File
- Shared Local memory (SLM)
- Main memory
- Private
- Local
- Global/Constant

Latency:
- Minimal latency
- Low latency
- Longest latency
Use Built-in Functions

\[ dp = V0.x \times V1.x + V0.y \times V1.y + V0.z \times V1.z + V0.w + V1.w; \]

\[ dp = \text{dot}(V0, V1); \]

\[ C = \sqrt{A \times A + B \times B}; \]

\[ C = \text{hypot}(A, B); \]

\[ cl = \text{fmin}(\text{fmax}(X, \text{minVal}), \text{maxVal}); \]

\[ cl = \text{clamp}(X, \text{minVal}, \text{maxVal}); \]

Allow the Compiler to Optimize Better
Trade Accuracy vs. Speed

- Use `mad()`/`fma()`: Either explicitly with built-ins or via `-cl-mad-enable` build option
- Use native_* versions of trigonometry functions or compile with `-cl-fast-relaxed-math` build option
- Floats processing is about ~2x of throughput than int – HD5200 and older

```c
a[id] += a[id] * b[id];
c[id] += a[id] * b[id];
a[id] = sin(a[id]);
c[id] = fma(a[id], b[id], c[id]);
a[id] = native_sin(a[id]);
```
Avoid Byte/Short Load and Stores

__kernel void k(global uchar* a, global uchar* b)
{
    int gid = get_global_id(0);
    a[gid] *= b[gid];
    . . .
}

__kernel void k(global uint4* a, global uint4* b)
{
    int gid = get_global_id(0);
    a[gid] *= b[gid];
    . . .
}

Avoid byte or short loads. Load and store in greater chunk
Agenda

• Intel® Iris™ Graphics Overview
• Intel® OpenCL™ Code Builder
• Intel® VTune™ Amplifier 2015
• Optimization Techniques and Examples
• OpenCL™ 2.0 Overview ---- Presenter: Sonal Sharma & Michael Stoner
• Summary / Questions
Shared Virtual Memory (Pre-history)

Builds upon “shared physical memory” (SPM) feature

- SPM established with OpenCL 1.0 => CL_MEM_USE_HOST_PTR flag
- Supported on Intel 3rd Gen processors with HD Graphics
- Eliminated buffer copy costs, aka “zero-copy” buffers*
- Buffer must have 4k byte alignment and size divisible by 64

SPM available since 2011, but many OpenCL apps still not using it...

* See “Getting the Most from OpenCL™ 1.2: How to Increase Performance by Minimizing Buffer Copies on Intel® Processor Graphics”
Shared Virtual Memory

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

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

**Coarse-grain buffers** *(Intel 5\textsuperscript{th} 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-mapped state: Only GPU can access buffer*
3 types of SVM

Coarse-grain buffers (Intel 5\textsuperscript{th} 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
3 types of SVM

**Fine-grain buffers (Intel 5th Gen Processors w/ HD Graphics 5500+)**

- SVM buffers can be accessed from either CPU or GPU at any time
- Can use *atomics* to avoid ‘race’ conditions
  
  Check if device supports (CL_DEVICE_SVM_FINE_GRAIN_BUFFER & CL_DEVICE_SVM_ATOMICS flags)
3 types of SVM

Fine-grain system memory (Future Intel Processors)
- CPU & GPU can share anything allocated from the C-runtime ‘heap’ (i.e. `malloc/new`)
- Ideal end-state – requires convergence of OS, H/W, and API support
Shared Virtual Memory – API Basics

Allows host-allocated VM pointers to be de-referenced directly on the GPU

- No need for clCreateBuffer() => cl_mem object to encapsulate a buffer
- Use clSVMAlloc() to allocate memory. To create Fine-grained buffer, use flag CL_MEM_SVM_FINE_GRAIN_BUFFER

```c
__kernel void svmBasic
    (global float *inputFloats)
{
    ...
    float x = *inputFloats;
    ...
}
```

```c
// Allocate SVM
float* inputFloats = (float*) clSVMAlloc(
    context,
    CL_MEM_READ_ONLY,  // input buffer
    size* sizeof(float), // size (bytes)
    0);                  // alignment
```
Two ways to pass SVM pointer to a kernel

- Use `clSetKernelArgSVMPointer()` to pass the pointer directly
- If this buffer contains pointers to additional SVM regions, use `clSetKernelExecInfo()` with `CL_KERNEL_EXEC_INFO_SVM_PTRS` flag

```c
// inputElements is an SVM allocation
err = clSetKernelArgSVMPointer(kernel, 0, inputElements);

// inputElements contains pointers to inputFloats
err = clSetKernelExecInfo(kernel,
                          CL_KERNEL_EXEC_INFO_SVM_PTRS,
                          sizeof(inputFloats),
                          &inputFloats
                       );
```
Nested Parallelism

• “Device-side enqueue”
  - OpenCL kernels can launch ‘child’ kernels on the device without returning control to the CPU host

• Enables flexible work scheduling entirely on the GPU
  - Recursive algorithms (e.g. quickSort, Sierpinski’s Carpet, etc.)

• Also, meets competitive challenge with CUDA’s implementation

• Device-side enqueue Building blocks:
  - Host side API: creating a default device queue from the host
  - Block Syntax: simplifies device side enqueue
  - Device side API: enqueue_kernel
Sierpiński Carpet

The **Sierpiński carpet** is a plane fractal first described by [Wacław Sierpiński](https://en.wikipedia.org/wiki/Wacław_Sierpiński) in 1916.

Start with a white square.

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

Paint the central sub-square black.

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

And so on …


Sierpiński Carpet Kernel in OpenCL 2.0

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

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

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

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

Easy to translate recursive algorithm to implementation
Sierpiński Carpet - Result

2187x2187 image: $8^6 = 299592$ enqueue_kernel calls!
Conference Session on OpenCL 2.0

Achieving Performance with OpenCL 2.0 on Intel Processor Graphics

Presented By: Robert Ioffe, Sonal Sharma and Michael Stoner

Date and Time: May 13th 2015, 10:40am
Agenda

• Intel® Iris™ Graphics Overview
• Intel® OpenCL™ Code Builder
• Intel® VTune™ Amplifier 2015
• Optimization Techniques and Examples
• OCL 2.0 Overview
• Summary / Questions
Additional Resources