IDF14

Intel® Processor Graphics: Optimizing Computer Vision and More

Aaron Kunze – GPU Compute Architect, Intel Corporation

GVCS002

intel®

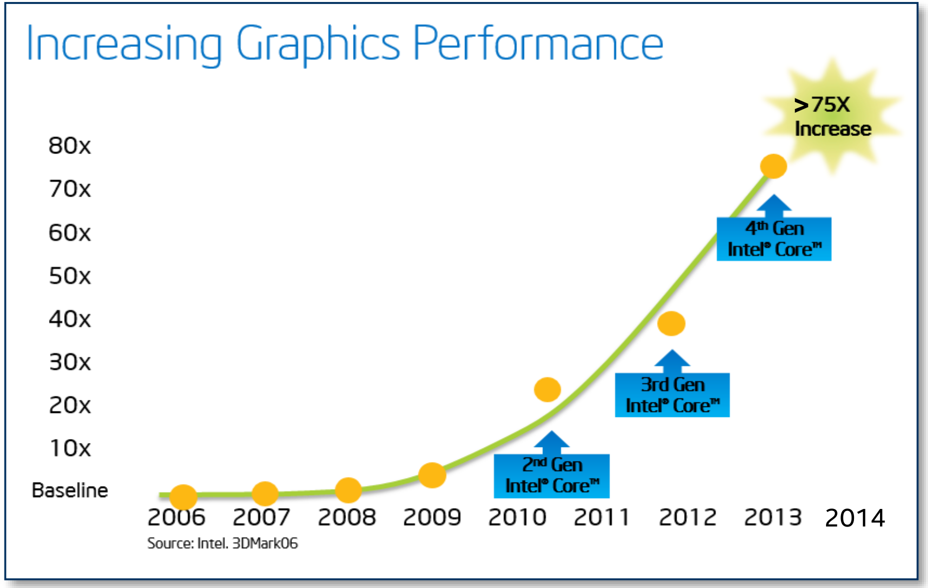
Agenda

- Intel® Graphics Introduction
- OpenCV 3.0 on Intel Graphics
- OpenCL™ Applications on Intel® Graphics Architecture
- Optimization Techniques
  - Maximizing Occupancy
  - Optimizing Memory Access
  - Using Registers
  - Maximizing Computation
- Intel® VTune™ Amplifier XE 2013 Support for OpenCL Applications
- New OpenCL 2.0 Features
- Summary / Questions

Agenda

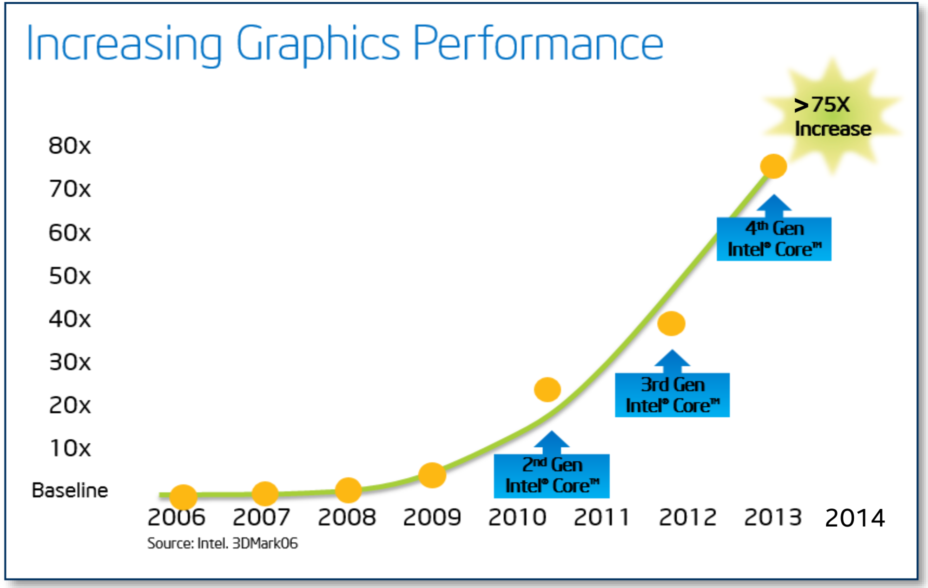
- Intel® Graphics Introduction
- OpenCV 3.0 on Intel Graphics
- OpenCL™ Applications on Intel® Graphics Architecture
- Optimization Techniques
  - Maximizing Occupancy
  - Optimizing Memory Access
  - Using Registers
  - Maximizing Computation
- Intel® VTune™ Amplifier XE 2013 Support for OpenCL Applications
- New OpenCL 2.0 Features
- Summary / Questions

Intel® Graphics

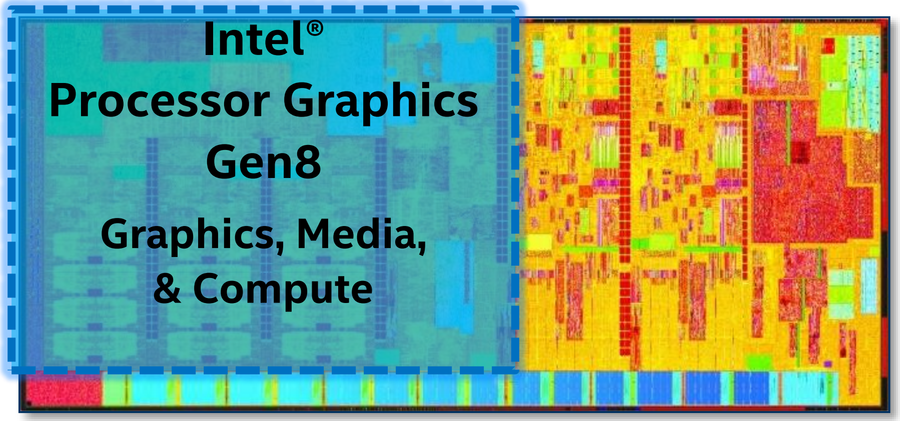


IDF14

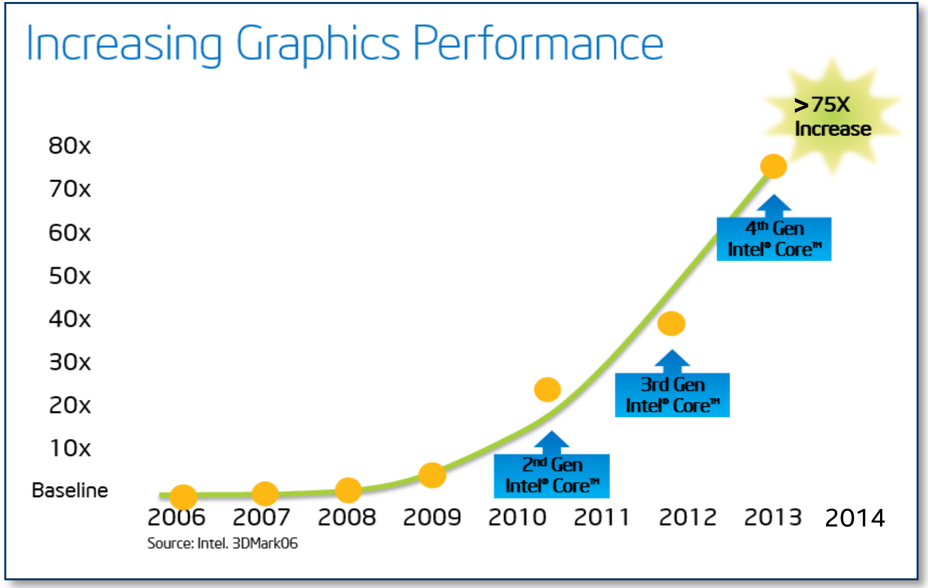
Intel® Graphics



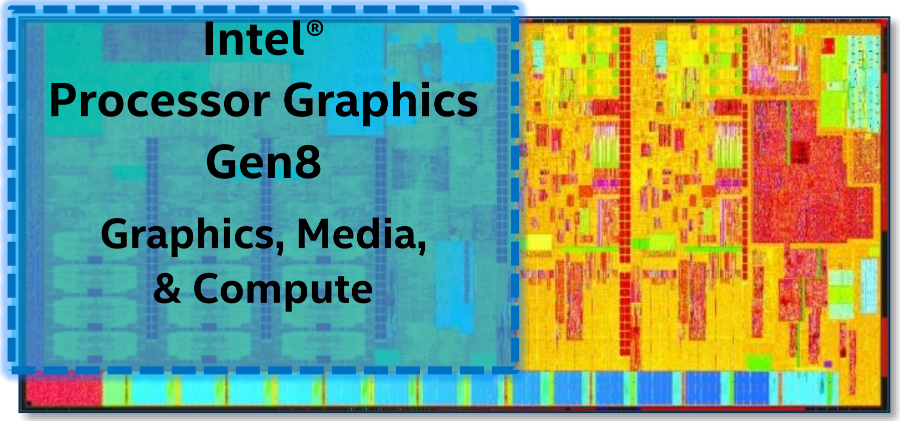
Intel® Core™ M:



Intel® Graphics



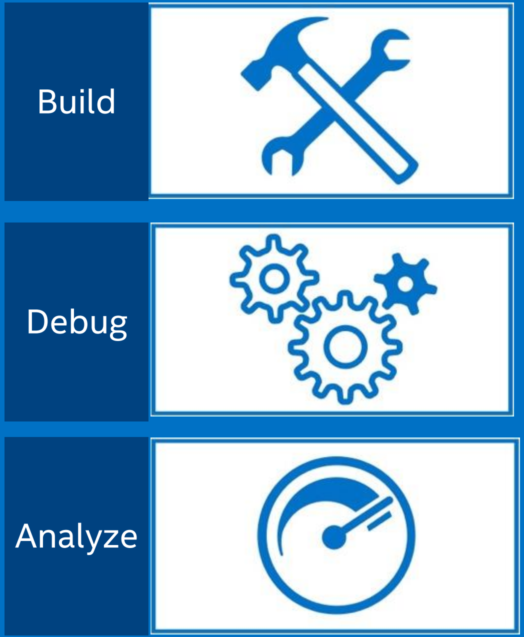
Intel® Core™ M:



Intel® Processor Graphics is a key compute resource

Intel® SDK for OpenCL™ Applications 2014 R2

The first industry SDK to provide developer tools for OpenCL™ 2.0



Supports latest standards and processors

- New Intel® Core™ M Processors for tablets and 2-in-1s
- OpenCL™ 2.0 and Shared Virtual Memory
- SPIR* 1.2

Easy-to-use development environment

- Build, Debug, Analyze
- Develop OpenCL 2.0 code that runs everywhere[1]

Remote execution on Android* devices

More analysis? Upgrade to Intel® VTune™ Amplifier XE

[1]Runs on previous generations of Intel Core Processors with CPU based OpenCL 2.0 runtime.

Download: intel.com/software/OpenCL

OpenCL™ Applications on Intel® Architecture: Success Stories

"OpenCL lets us write one line of code that will run on lots of different types of hardware"

*Eric Berdahl, Senior Engineering Manager, Adobe**
Adobe Optimizes with OpenCL™ and Intel® Graphics: http://www.youtube.com/watch?v=IXdhhud5iH4

"The Intel Iris Pro graphics and the Intel Core i7 processor are … allowing me to do all of this while the graphics and video are never stopping"

Dave Helmly, Solution Consulting Pro Video/Audio, Adobe
Adobe Premiere Pro demonstration: http://www.youtube.com/watch?v=u0J57J6Hppg

"We are very pleased that Intel is fully supporting OpenCL. We think there is a bright future for this technology."

Michael Bryant, Director of Marketing, Sony Creative Software
Vegas* Software Family by Sony* Optimized with OpenCL and Intel® Processor Graphics
http://www.youtube.com/watch?v=_KHVOCwTdno

OpenCL™ Applications on Intel® Architecture: Success Stories

“OpenCL lets us write one line of code that will run on lots of different types of hardware”

*Eric Berdahl, Senior Engineering Manager, Adobe**
Adobe Optimizes with OpenCL™ and Intel® Graphics: http://www.youtube.com/watch?v=IXdhhud5iH4

“The Intel Iris Pro graphics and the Intel Core i7 processor are … allowing me to do all of this while the graphics and video are never stopping”

Dave Helmly, Solution Consulting Pro Video/Audio, Adobe
Adobe Premiere Pro demonstration: http://www.youtube.com/watch?v=u0J57J6Hppg

“We are very pleased that Intel is fully supporting OpenCL. We think there is a bright future for this technology.”

Michael Bryant, Director of Marketing, Sony Creative Software
Vegas* Software Family by Sony* Optimized with OpenCL and Intel® Processor Graphics
http://www.youtube.com/watch?v=_KHVOCwTdno

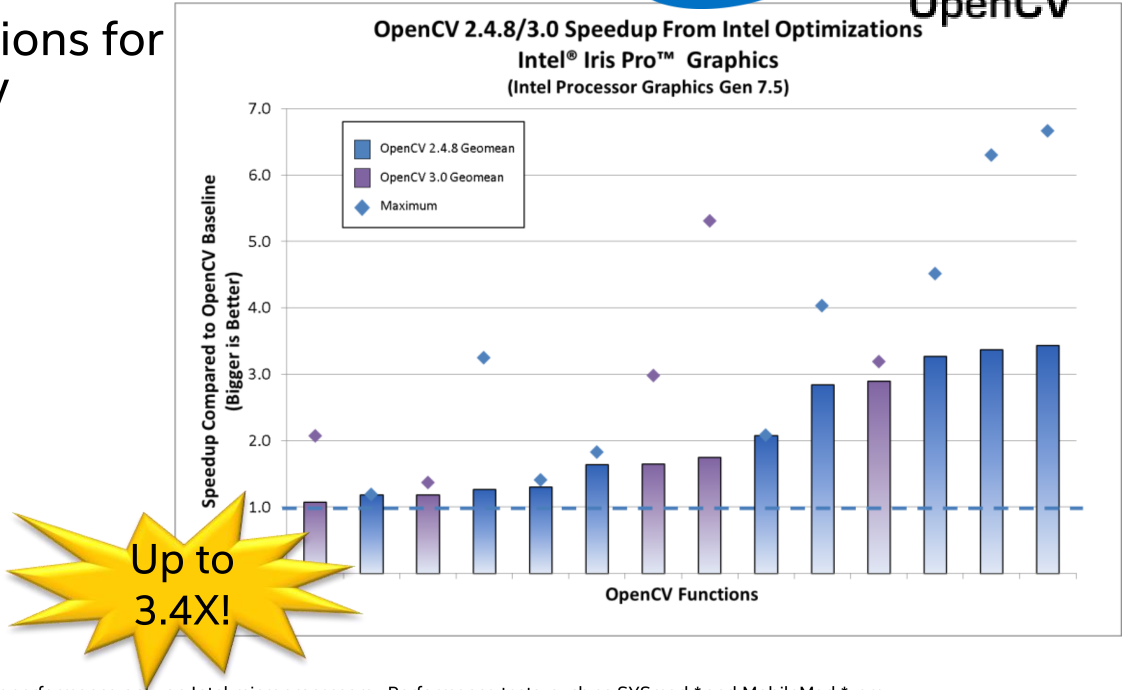
Our customers report on benefits like productivity, performance, and use of open standard

Agenda

- Intel® Graphics Introduction
- OpenCV 3.0 on Intel Graphics
- OpenCL™ Applications on Intel® Graphics Architecture
- Optimization Techniques
  - Maximizing Occupancy
  - Optimizing Memory Access
  - Using Registers
  - Maximizing Computation
- Intel® VTune™ Amplifier XE 2013 Support for OpenCL Applications
- New OpenCL 2.0 Features
- Summary / Questions

OpenCV Optimization for Intel® Graphics

- OpenCV: the leading open-source computer vision library
- Intel is contributing optimizations for the OpenCL™ code in OpenCV
- Intel optimizations delivering substantial performance improvements!
  - Example optimizations described throughout this talk

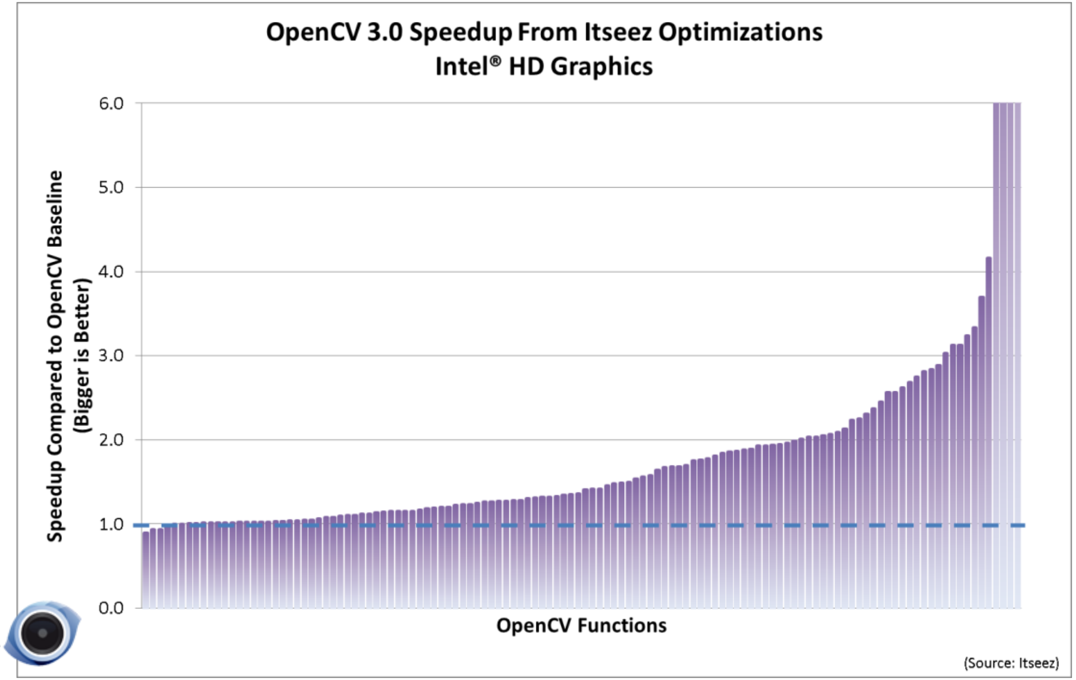


Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors 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

OpenCV Optimization for Intel® Graphics

Intel is working with Itseez* to further optimize OpenCL™ code in OpenCV 3.0

- Itseez is the official maintainer of OpenCV
- Itseez is using optimization BKMs described in this talk!



OpenCV

intel

itseez — Vision that works!

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

OpenCV 3.0 for Intel® Graphics

OpenCV 3.0 architecture further improves support for Intel® Graphics (Alpha release available!)

- "Transparent API" enables same code to use CPU or OpenCL™ devices
  - Little or no code changes from existing OpenCV code
  - Code uses efficient CPU fallback on platforms without OpenCL

```
cv::UMat inMat, outMat;
vidInput >> inMat;
cv::cvtColor(inMat, outMat, cv::COLOR_RGB2GRAY);
vidOutput << outMat;
```

- APIs operate asynchronously
- Improved use of shared physical memory for integrated GPU performance

OpenCV 3.0 for Intel® Graphics

OpenCV 3.0 architecture further improves support for Intel® Graphics (Alpha release available!)

- "Transparent API" enables same code to use CPU or OpenCL™ devices
  - Little or no code changes from existing OpenCV code
  - Code uses efficient CPU fallback on platforms without OpenCL

```
cv::UMat inMat, outMat;
vidInput >> inMat;
cv::cvtColor(inMat, outMat, cv::COLOR_RGB2GRAY);
vidOutput << outMat;
```

- APIs operate asynchronously
- Improved use of shared physical memory for integrated GPU performance

OpenCV 3.0 improves use of OpenCL on Intel® Graphics!

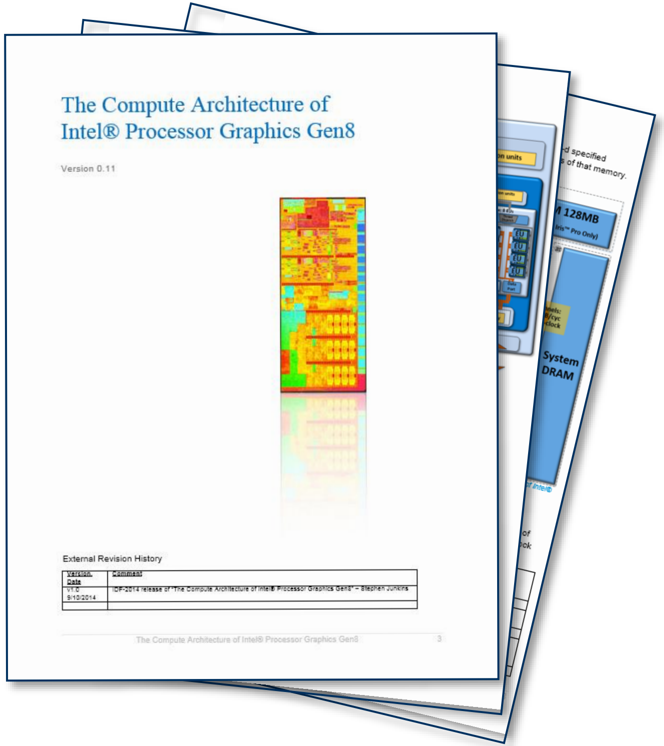
Agenda

- Intel® Graphics Introduction
- OpenCV 3.0 on Intel Graphics
- OpenCL™ Applications on Intel® Graphics Architecture
- Optimization Techniques
  - Maximizing Occupancy
  - Optimizing Memory Access
  - Using Registers
  - Maximizing Computation
- Intel® VTune™ Amplifier XE 2013 Support for OpenCL Applications
- New OpenCL 2.0 Features
- Summary / Questions

Intel® Processor Graphics Architecture

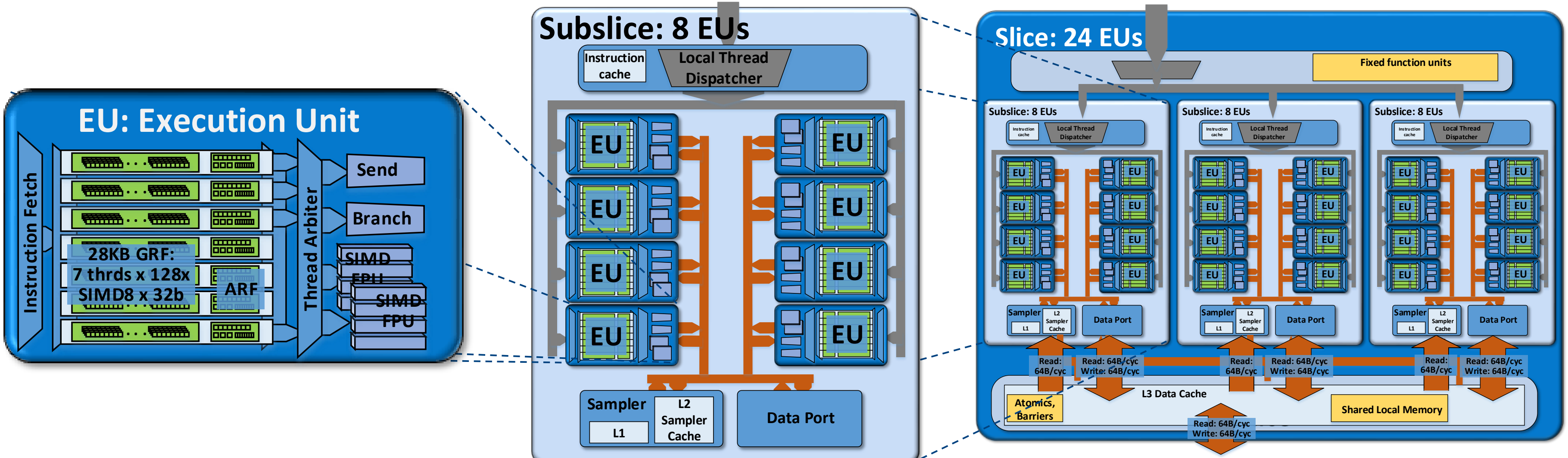
- Today, our focus is on Intel® Iris™ Graphics and Intel® HD Graphics in 4th Generation Intel® Core™ Processors and Intel Core M Processors
  - Or, Intel Processor Graphics Gen7.5 and Gen8.0
- For more details, see our whitepaper, The Compute Architecture of Intel Processor Graphics Gen7.5/Gen8.0
  - https://software.intel.com/en-us/articles/intel-graphics-developers-guides

New!



IDF14

Intel® Graphics Architecture Building Blocks



Executing OpenCL™ Kernels

- OpenCL™ work items map to SIMD lanes
- Compiler can choose between SIMD8, SIMD16, and SIMD32

Executing OpenCL™ Kernels

- OpenCL™ work items map to SIMD lanes
- Compiler can choose between SIMD8, SIMD16, and SIMD32

Example: work group with 64 work items

00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	10	11	12	13	○ ○ ○	3f

Executing OpenCL™ Kernels

- OpenCL™ work items map to SIMD lanes
- Compiler can choose between SIMD8, SIMD16, and SIMD32

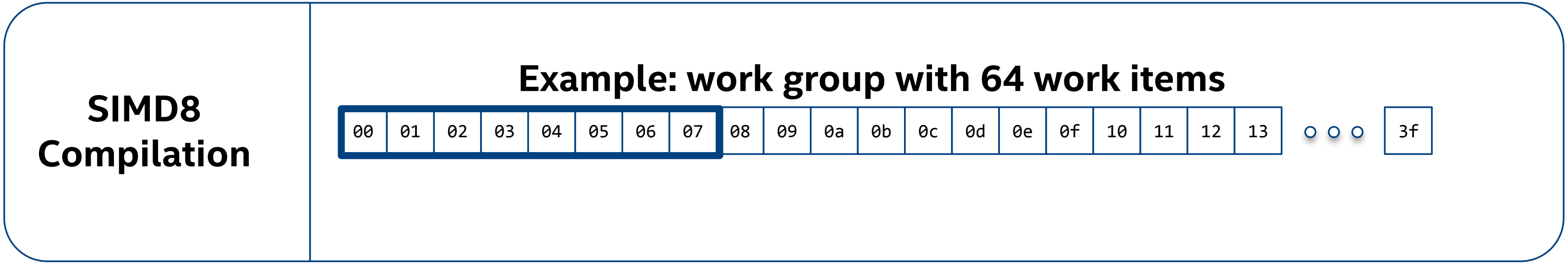
SIMD8 Compilation

Example: work group with 64 work items

00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	10	11	12	13	…	3f

Executing OpenCL™ Kernels

- OpenCL™ work items map to SIMD lanes
- Compiler can choose between SIMD8, SIMD16, and SIMD32



Executing OpenCL™ Kernels

- OpenCL™ work items map to SIMD lanes
- Compiler can choose between SIMD8, SIMD16, and SIMD32

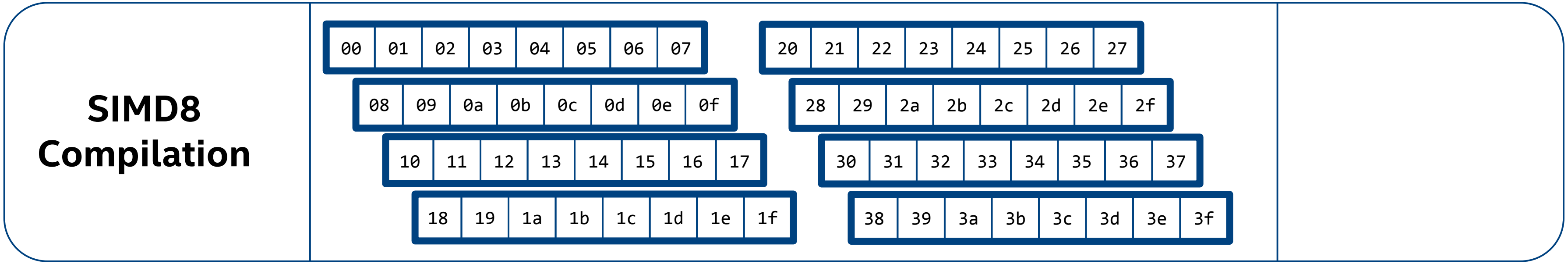
SIMD8 Compilation

Example: work group with 64 work items

00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	10	11	12	13	…	3f

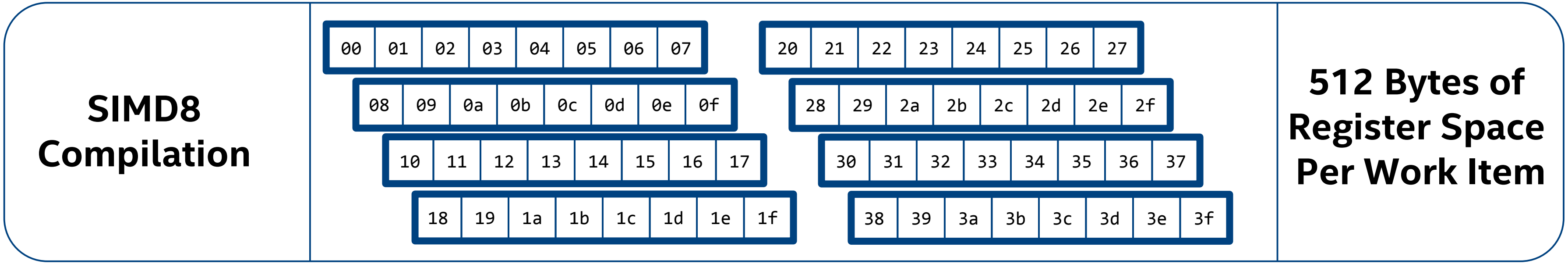
Executing OpenCL™ Kernels

- OpenCL™ work items map to SIMD lanes
- Compiler can choose between SIMD8, SIMD16, and SIMD32



Executing OpenCL™ Kernels

- OpenCL™ work items map to SIMD lanes
- Compiler can choose between SIMD8, SIMD16, and SIMD32



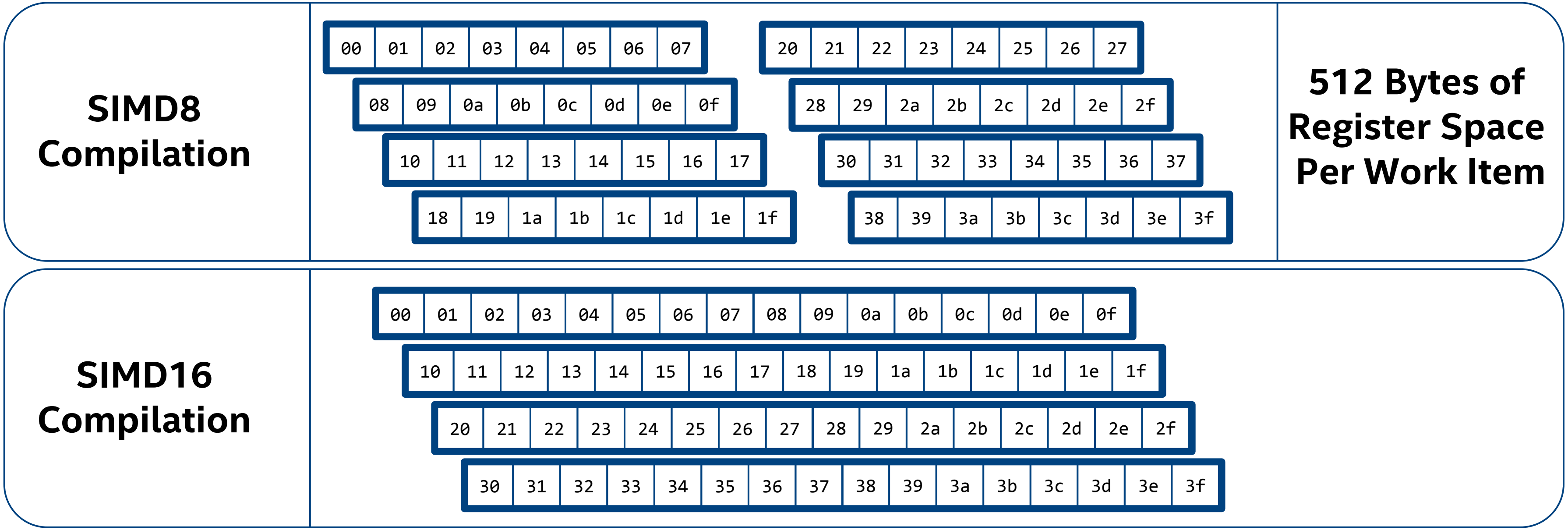
Executing OpenCL™ Kernels

- OpenCL™ work items map to SIMD lanes
- Compiler can choose between SIMD8, SIMD16, and SIMD32

SIMD8 Compilation	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f	20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f	**512 Bytes of Register Space Per Work Item**
SIMD16 Compilation			

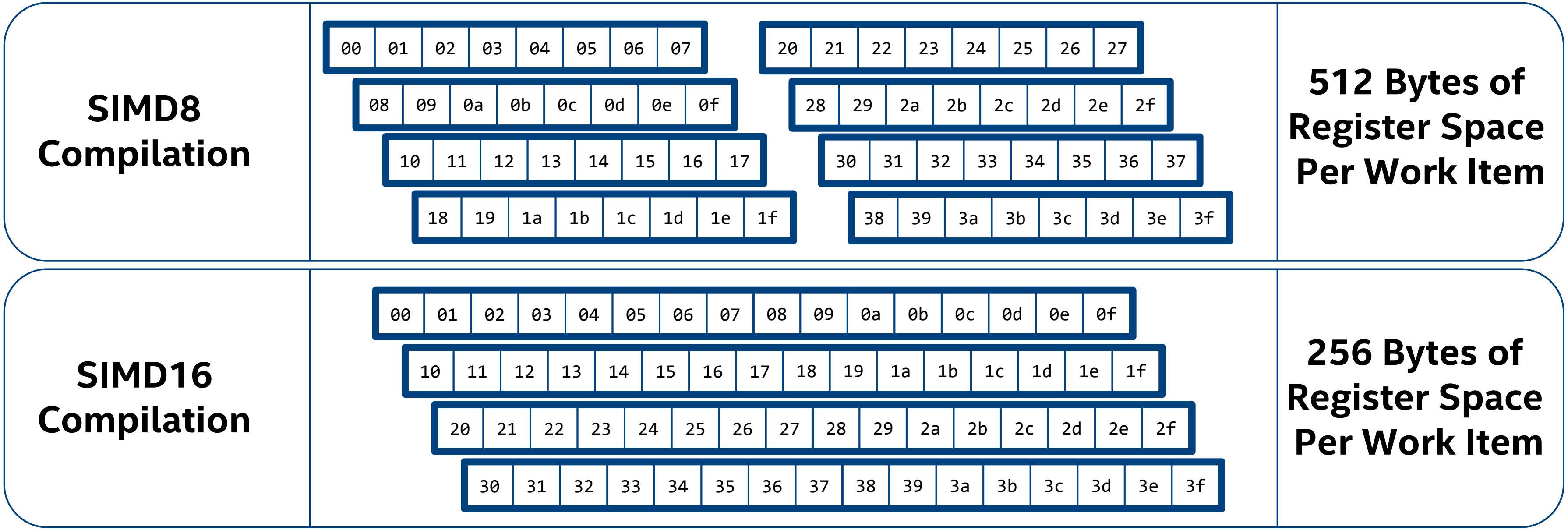
Executing OpenCL™ Kernels

- OpenCL™ work items map to SIMD lanes
- Compiler can choose between SIMD8, SIMD16, and SIMD32



Executing OpenCL™ Kernels

- OpenCL™ work items map to SIMD lanes
- Compiler can choose between SIMD8, SIMD16, and SIMD32

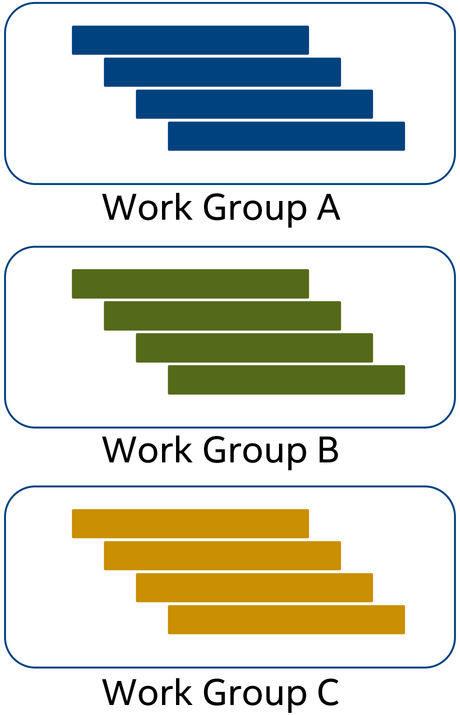


Executing OpenCL™ Kernels

- OpenCL™ work items map to SIMD lanes
- Compiler can choose between SIMD8, SIMD16, and SIMD32

Executing OpenCL™ Kernels

Example: SIMD16 compile, 64 work items per work group

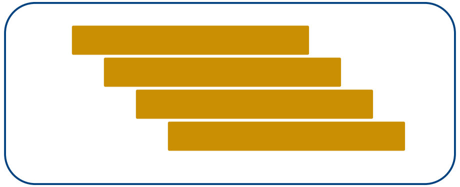


Executing OpenCL™ Kernels

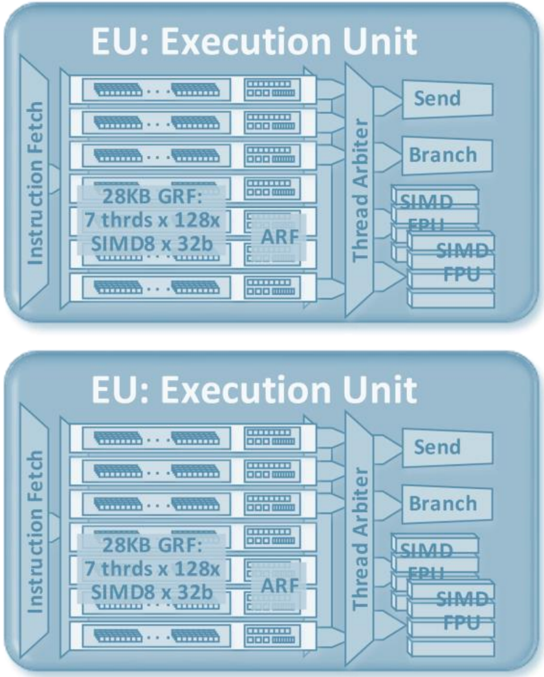
Example: SIMD16 compile, 64 work items per work group

Work Group A

Work Group B



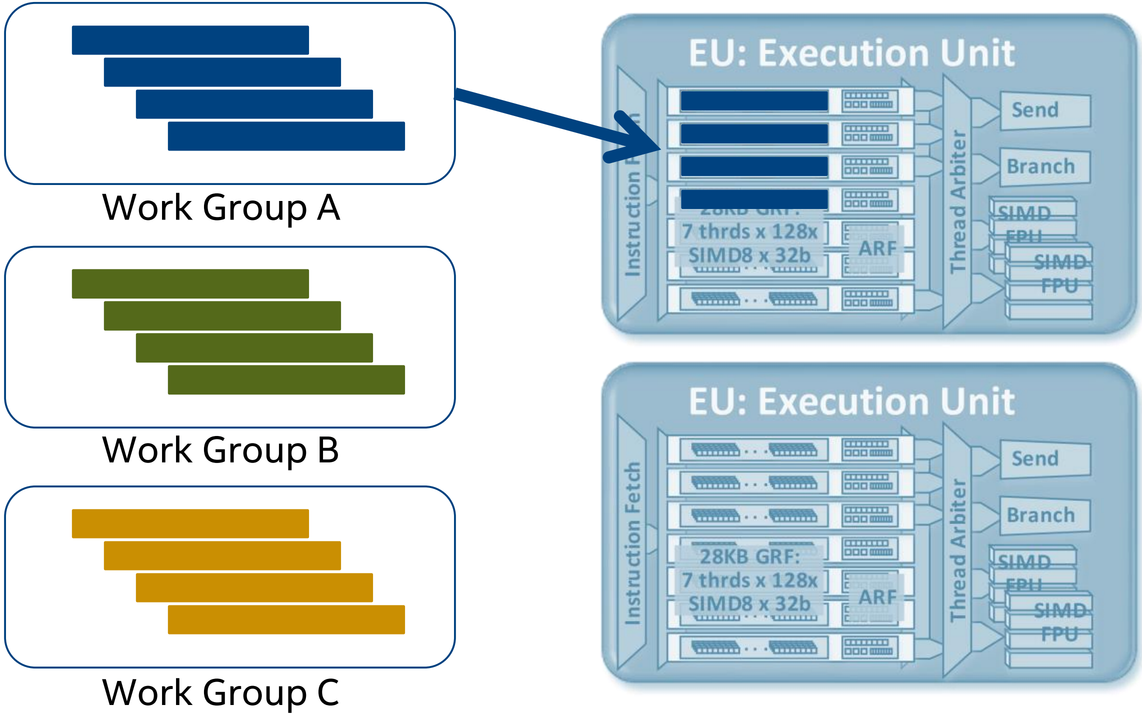
Work Group C



IDF14

Executing OpenCL™ Kernels

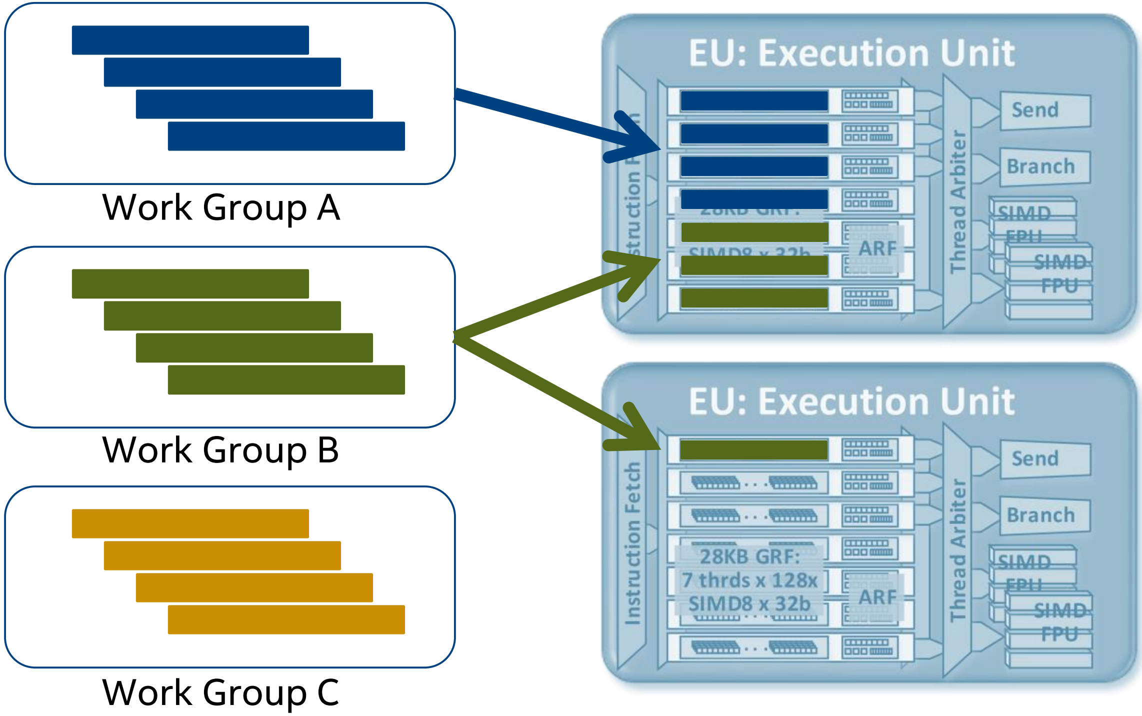
Example: SIMD16 compile, 64 work items per work group



IDF14

Executing OpenCL™ Kernels

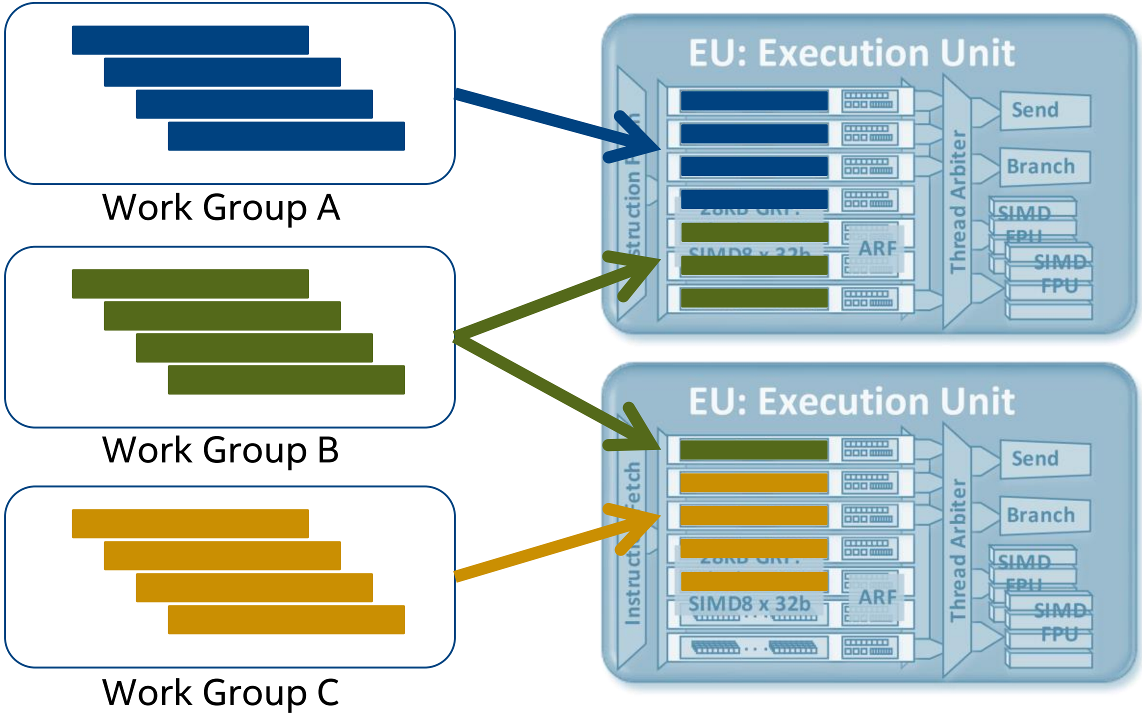
Example: SIMD16 compile, 64 work items per work group



IDF14

Executing OpenCL™ Kernels

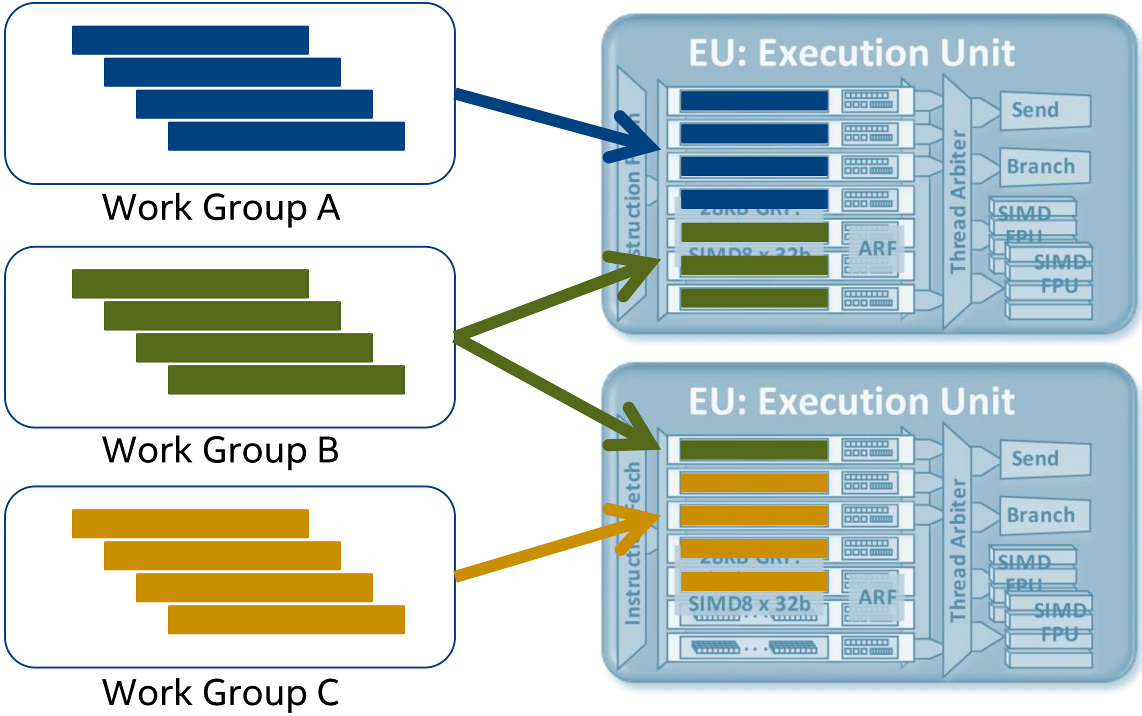
Example: SIMD16 compile, 64 work items per work group



IDF14

Executing OpenCL™ Kernels

Example: SIMD16 compile, 64 work items per work group



Workgroups may span EU threads!

Workgroups may span EUs!

If using local memory or barriers, workgroups will not span sub slices!
(8 EUs in Processor Graphics Gen8)

IDF14

Agenda

- Intel® Graphics Introduction
- OpenCV 3.0 on Intel Graphics
- OpenCL™ Applications on Intel® Graphics Architecture
- Optimization Techniques
  - Maximizing Occupancy
  - Optimizing Memory Access
  - Using Registers
  - Maximizing Computation
- Intel® VTune™ Amplifier XE 2013 Support for OpenCL Applications
- New OpenCL 2.0 Features
- Summary / Questions

Maximizing Occupancy

- Occupancy is a measure of EU thread utilization
- Two primary things to consider:
  - Launch enough work items to keep EU threads busy
  - In short kernels: use short vector data types and compute multiple pixels to better amortize thread launch cost

Maximizing Occupancy

- Occupancy is a measure of EU thread utilization
- Two primary things to consider:
  - Launch enough work items to keep EU threads busy
  - In short kernels: use short vector data types and compute multiple pixels to better amortize thread launch cost
    - For example, color conversion:

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

Before:
One pixel per work item

Maximizing Occupancy

- Occupancy is a measure of EU thread utilization
- Two primary things to consider:
  - Launch enough work items to keep EU threads busy
  - In short kernels: use short vector data types and compute multiple pixels to better amortize thread launch cost
    - For example, color conversion:

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

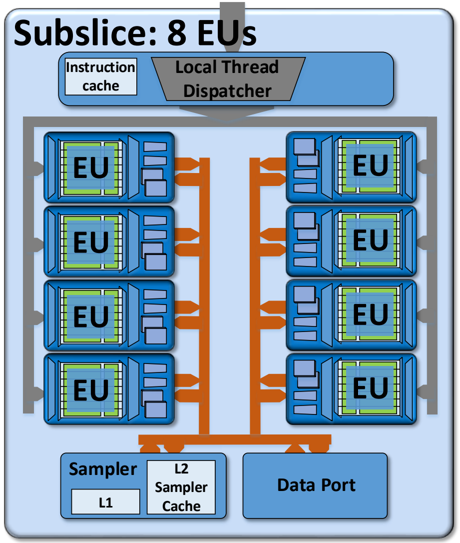
Before:
One pixel per work item

```
__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);
```

After:
Four pixels per work item

Occupancy Constraints

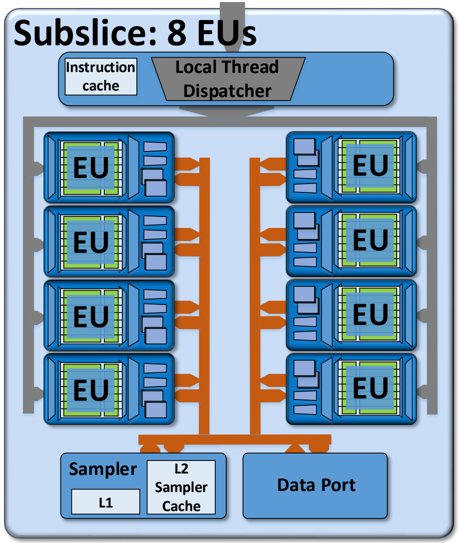
- More subtle occupancy issues (*when using barriers or local memory*):
  -
  -
  -



IDF14

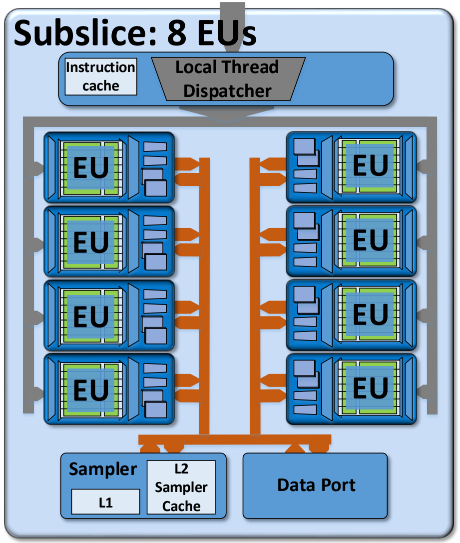
Occupancy Constraints

- 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
    -
    -



Occupancy Constraints

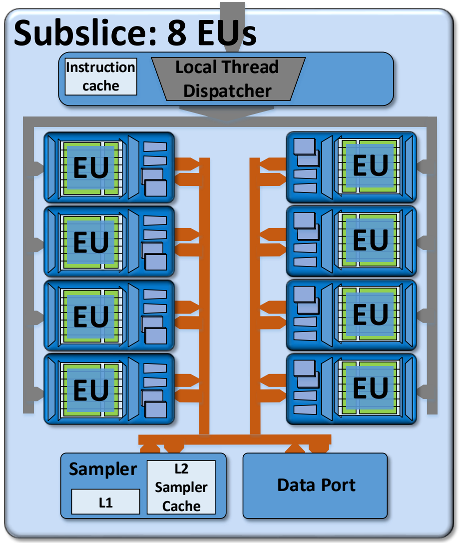
- More subtle occupancy issues (*when using barriers or local memory*):
  - Sub slices will not run partial workgroups
    - Can be a limiting factor for very large work groups
  - Sub slices will not run more than 16 work groups
    - Can be limiting factor for very small work groups
    -



IDF14

Occupancy Constraints

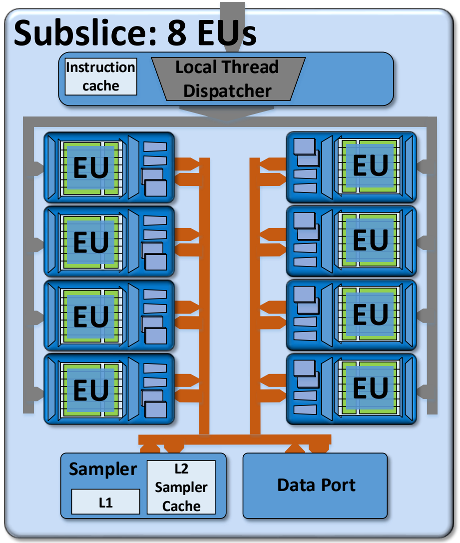
- More subtle occupancy issues (*when using barriers or local memory*):
  - Sub slices will not run partial workgroups
    - Can be a limiting factor for very large work groups
  - Sub slices will not run more than 16 work groups
    - Can be limiting factor for very small work groups
  - Shared Local Memory (SLM) – 64KB SLM per sub slice
    - Can be a limiting factor for kernels that use a lot of local memory



IDF14

Occupancy Constraints

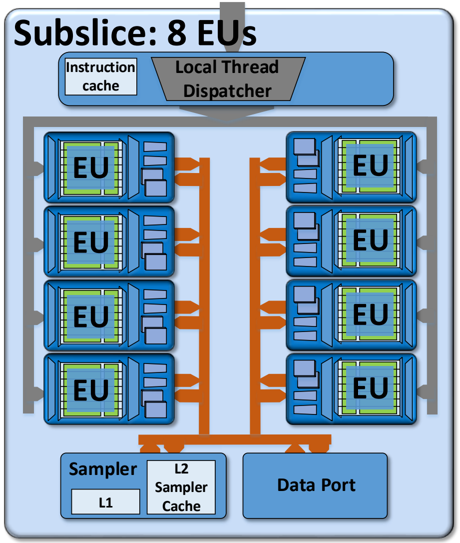
- More subtle occupancy issues (*when using barriers or local memory*):
  - Sub slices will not run partial workgroups
    - Can be a limiting factor for very large work groups
  - Sub slices will not run more than 16 work groups
    - Can be limiting factor for very small work groups
  - Shared Local 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



IDF14

Occupancy Constraints

- 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 work groups
    - Can be 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



Maximize occupancy to keep EUs busy!

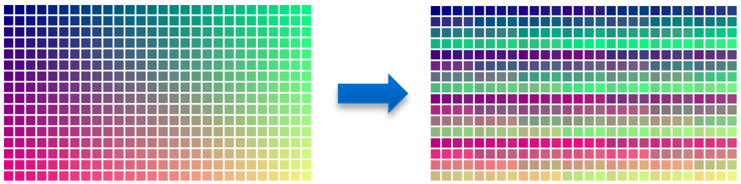
Optimizing Host to Device Transfers

- Host (CPU) and Device (GPU) share the same physical memory
- For buffers allocated through the OpenCL™ runtime:
  - Let the OpenCL runtime allocate system memory
    - Create buffer with system memory pointer and **`CL_MEM_ALLOC_HOST_PTR`**
  - ***OR***, Use pre-allocated system memory
    - Create buffer with system memory pointer and **`CL_MEM_USE_HOST_PTR`**
    - Allocate system memory aligned to a page (4096 bytes) (e.g., use _aligned_malloc or memalign to allocate)
    - Allocate a multiple of cache line size (64 bytes)
    - No transfer needed (zero copy)!
  - Use **`clEnqueueMapBuffer()`** to access data
    - No transfer needed (zero copy)!
  - OpenCV 3.0 changes make excellent use of this feature!

IDF14

Optimizing Host to Device Transfers

- For images allocated through the OpenCL™ runtime:
  - OpenCL images are tiled by default (transfer required)

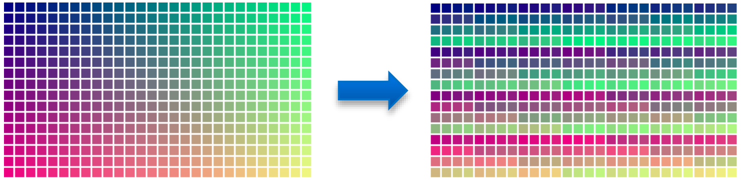


Default Image Creation Path

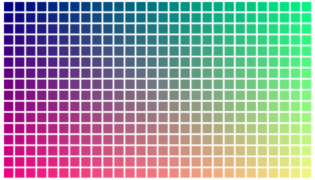
Optimizing Host to Device Transfers

- For images allocated through the OpenCL™ runtime:
  - OpenCL images are tiled by default (transfer required)
  - Or, convert a linear buffer to an image without copy!
  - Core feature in OpenCL 2.0, **`cl_khr_image2d_from_buffer`** extension in OpenCL 1.2

New!



Default Image Creation Path



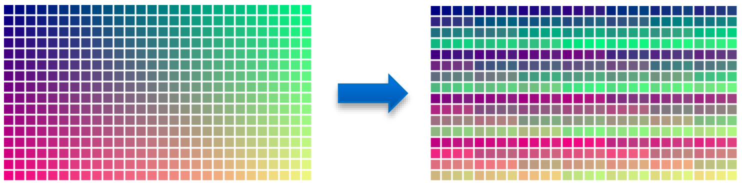
Convert a Buffer to an Image (No Copy!)

Optimizing Host to Device Transfers

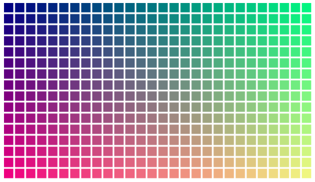
- For images allocated through the OpenCL™ runtime:
  - OpenCL images are tiled by default (transfer required)
  - Or, convert a linear buffer to an image without copy!
  - Core feature in OpenCL 2.0, **`cl_khr_image2d_from_buffer`** extension in OpenCL 1.2

New!

- Texture sampler great for cases that need linear interpolation
- Used in some flavors of OpenCV resize and Pyramid Lucas-Kanade
  - Up to 2X performance!



Default Image Creation Path



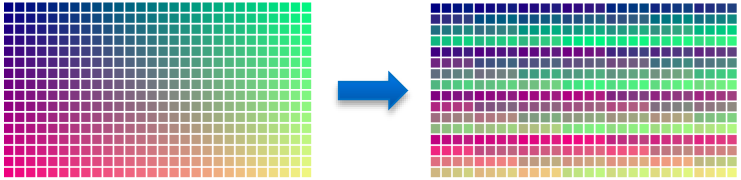
Convert a Buffer to an Image (No Copy!)

Optimizing Host to Device Transfers

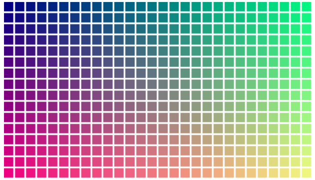
- For images allocated through the OpenCL™ runtime:
  - OpenCL images are tiled by default (transfer required)
  - Or, convert a linear buffer to an image without copy!
  - Core feature in OpenCL 2.0, **`cl_khr_image2d_from_buffer`** extension in OpenCL 1.2

New!

- Texture sampler great for cases that need linear interpolation
- Used in some flavors of OpenCV resize and Pyramid Lucas-Kanade
  - Up to 2X performance!



Default Image Creation Path



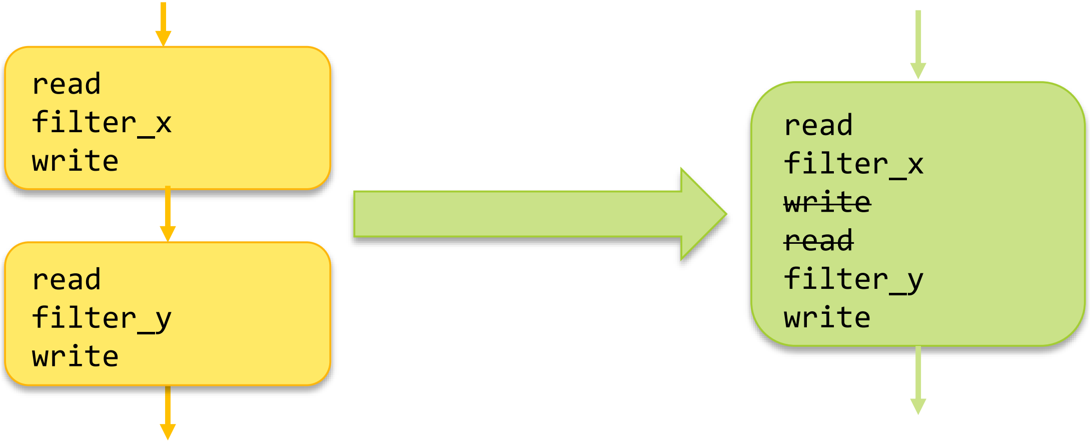
Convert a Buffer to an Image (No Copy!)

Take advantage of shared physical memory for buffers and images!

IDF14

Optimizing Memory Accesses

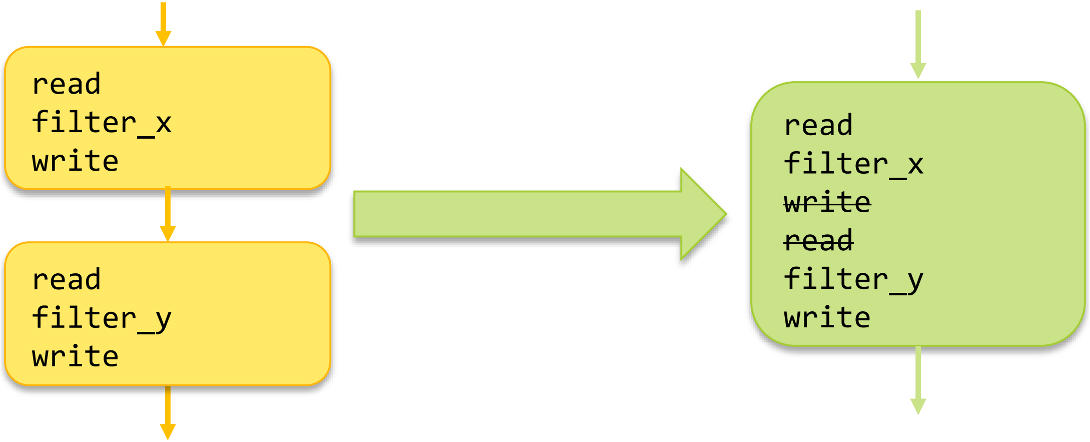
- Merging kernels reduces memory traffic
  - Computer vision algorithms often form pipelines
  - Merging multiple kernels in a pipeline can reduce trips to memory
    - Also reduces runtime overhead!



  - But mind instruction cache size (2K – 4K instructions)!
  - New read/write images in OpenCL™ 2.0 standard can help merge kernels

Optimizing Memory Accesses

- Merging kernels reduces memory traffic
  - Computer vision algorithms often form pipelines
  - Merging multiple kernels in a pipeline can reduce trips to memory
    - Also reduces runtime overhead!



  - But mind instruction cache size (2K – 4K instructions)!
  - New read/write images in OpenCL™ 2.0 standard can help merge kernels

New!

Used to speedup OpenCV separable filters!

Memory Access Patterns

- Local memory accesses have latencies similar to L3$ hits
  - Just using local memory as a cache is often not productive
- But, local memory and L3$ are organized differently

Memory Access Patterns

- Local memory accesses have latencies similar to L3$ hits
  - Just using local memory as a cache is often not productive
- But, local memory and L3$ are organized differently

L3$ (global and constant)

line	line 0	line 1	• • •
addr	0x00	0x40	

64-byte cache lines

Touch as few cache lines as possible

SLM (local)

bank	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	0	1	2	3	• • •
addr	0x00	0x04	0x08	0x0c	0x10	0x14	0x18	0x1c	0x20	0x24	0x28	0x2c	0x30	0x34	0x38	0x3c	0x40	0x44	0x48	0x4c	

16 banks on 4-byte boundaries

Touch as many banks as possible

Memory Access Patterns

- Examples

	L3$	SLM	
`data[get_global_id(0)];`	1 cache line Full bandwidth	16 banks Full bandwidth	`data[get_local_id(0)];`
`data[get_global_id(0) + 1];`	2 cache lines Half bandwidth	16 banks Full bandwidth	`data[get_local_id(0) + 1];`
`data[get_global_id(0) * 2];`	2 cache line Half bandwidth	8 banks Half bandwidth	`data[get_local_id(0) * 2];`
`data[get_global_id(0) * 16];`	16 cache lines Worst case!	1 bank Worst case!	`data[get_local_id(0) * 16];`
`data[get_global_id(0) * 17];`	16 cache line Worst case!	16 banks Full bandwidth	`data[get_local_id(0) * 17];`

Memory Access Patterns

- Examples

	L3$	SLM	
`data[get_global_id(0)];`	1 cache line Full bandwidth	16 banks Full bandwidth	`data[get_local_id(0)];`
`data[get_global_id(0) + 1];`	2 cache lines Half bandwidth	16 banks Full bandwidth	`data[get_local_id(0) + 1];`
`data[get_global_id(0) * 2];`	2 cache line Half bandwidth	8 banks Half bandwidth	`data[get_local_id(0) * 2];`
`data[get_global_id(0) * 16];`	16 cache lines Worst case!	1 bank Worst case!	`data[get_local_id(0) * 16];`
`data[get_global_id(0) * 17];`	16 cache line Worst case!	16 banks Full bandwidth	`data[get_local_id(0) * 17];`

When picking a memory type, consider access patterns!

Registers Vs. Memory

- Each work item in an OpenCL™ kernel has access to up to 512 bytes of register space
- Bandwidth to registers faster than any memory
- Loading and processing blocks of pixels in registers is very efficient!

Registers Vs. Memory

- Each work item in an OpenCL™ kernel has access to up to 512 bytes of register space
- Bandwidth to registers 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];
// Load filter kernel in k, input data in d
...
// Compute convolution
for (px = 0; px < PX_PER_WI_X; ++px)
    for (sx = 0; sx < KERNEL_SIZE_X; ++sx)
        sum[px]= mad(k[sx], d[px + sx], sum[px]);
```

Registers Vs. Memory

- Each work item in an OpenCL™ kernel has access to up to 512 bytes of register space
- Bandwidth to registers 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];
// Load filter kernel in k, input data in d
...
// Compute convolution
for (px = 0; px < PX_PER_WI_X; ++px)
    for (sx = 0; sx < KERNEL_SIZE_X; ++sx)
        sum[px]= mad(k[sx], d[px + sx], sum[px]);
```

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

Maximizing Compute Performance

- Avoid **`long`** and **`size_t`** data types
- Using **`short`** data types may improve performance
- Trade accuracy for speed, where appropriate
  - Use “native” built-ins (or use **`-cl-fast-relaxed-math`**)
  - Use **`mad()`** / **`fma()`** (or use **`-cl-mad-enable`**)

```
x = cos(i);
```

→

```
x = native_cos(i);
```

Maximizing Compute Performance

- Avoid **`long`** and **`size_t`** data types
- Using **`short`** data types may improve performance
- Trade accuracy for speed, where appropriate
  - Use “native” built-ins (or use **`-cl-fast-relaxed-math`**)
  - Use **`mad()`** / **`fma()`** (or use **`-cl-mad-enable`**)

```
x = cos(i);
```

```
x = native_cos(i);
```

Used to speedup OpenCV SURF and HOG!

Maximizing Compute Performance

- The OpenCL™ 2.0 standard offers new workgroup reductions and scans
  - Operations available: add, min, max
  - Allows reductions and scans without exposed local memory or barriers
  - Device-specific implementation very efficient for the hardware

Maximizing Compute Performance

- The OpenCL™ 2.0 standard offers new workgroup reductions and scans
  - Operations available: add, min, max
  - Allows reductions and scans without exposed local memory or barriers
  - Device-specific implementation very efficient for the hardware

```
__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[id +   8]; barrier(CLK_LOCAL_MEM_FENCE);
if (id <   4) smem[id] = sum = sum + smem[id +   4]; barrier(CLK_LOCAL_MEM_FENCE);
if (id <   2) smem[id] = sum = sum + smem[id +   2]; barrier(CLK_LOCAL_MEM_FENCE);
if (id <   1) smem[id] = sum = sum + smem[id +   1]; barrier(CLK_LOCAL_MEM_FENCE);
sum = smem[0];
```

Before

Maximizing Compute Performance

- The OpenCL™ 2.0 standard offers new workgroup reductions and scans
  - Operations available: add, min, max
  - Allows reductions and scans without exposed local memory or barriers
  - Device-specific implementation very efficient for the hardware

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

After

New!

Maximizing Compute Performance

- The OpenCL™ 2.0 standard offers new workgroup reductions and scans
  - Operations available: add, min, max
  - Allows reductions and scans without exposed local memory or barriers
  - Device-specific implementation very efficient for the hardware

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

After

New!

✓ *No exposed local memory or barriers*

✓ *Code written independent of workgroup size*

✓ *Intel optimized for Processor Graphics*

Maximizing Compute Performance

- The OpenCL™ 2.0 standard offers new workgroup reductions and scans
  - Operations available: add, min, max
  - Allows reductions and scans without exposed local memory or barriers
  - Device-specific implementation very efficient for the hardware

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

After

New!

✓ *No exposed local memory or barriers*

✓ *Code written independent of workgroup size*

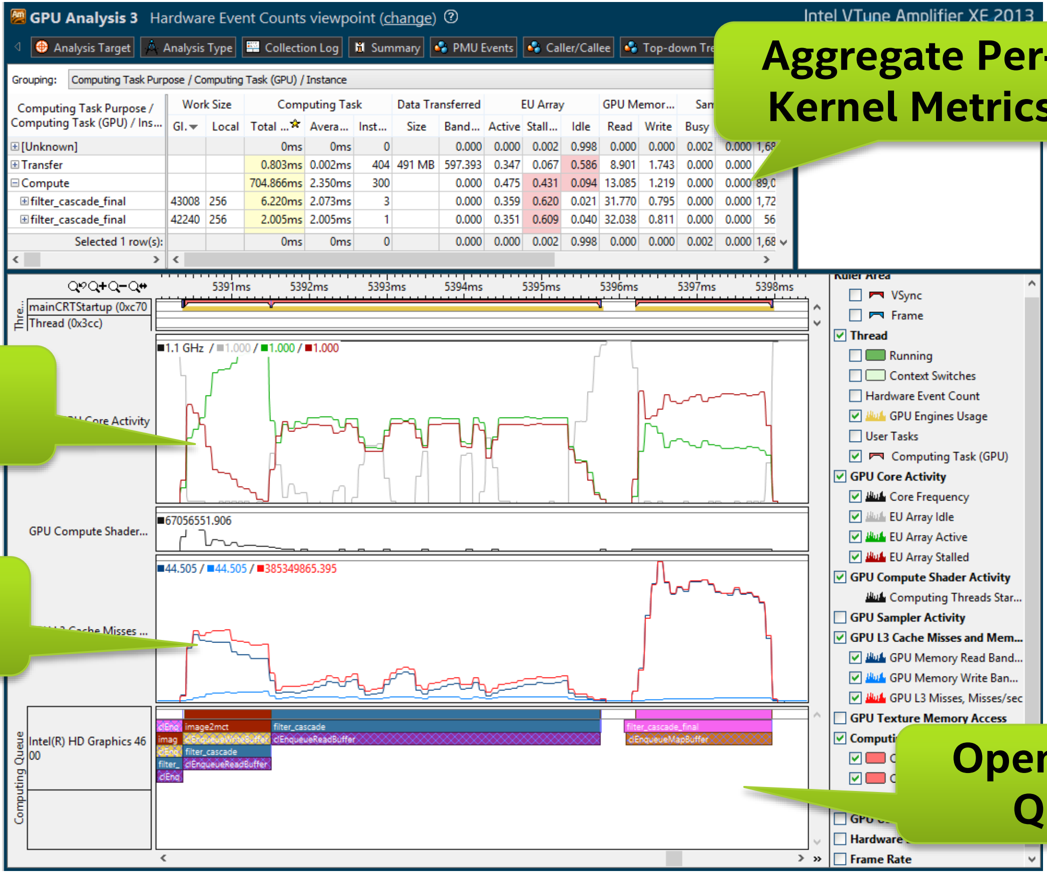
✓ *Intel optimized for Processor Graphics*

Use work group reductions to clean up and speed up code!

Agenda

- Intel® Graphics Introduction
- OpenCV 3.0 on Intel Graphics
- OpenCL™ Applications on Intel® Graphics Architecture
- Optimization Techniques
  - Maximizing Occupancy
  - Optimizing Memory Access
  - Using Registers
  - Maximizing Computation
- Intel® VTune™ Amplifier XE 2013 Support for OpenCL Applications
- New OpenCL 2.0 Features
- Summary / Questions

Intel® VTune™ Amplifier XE 2015



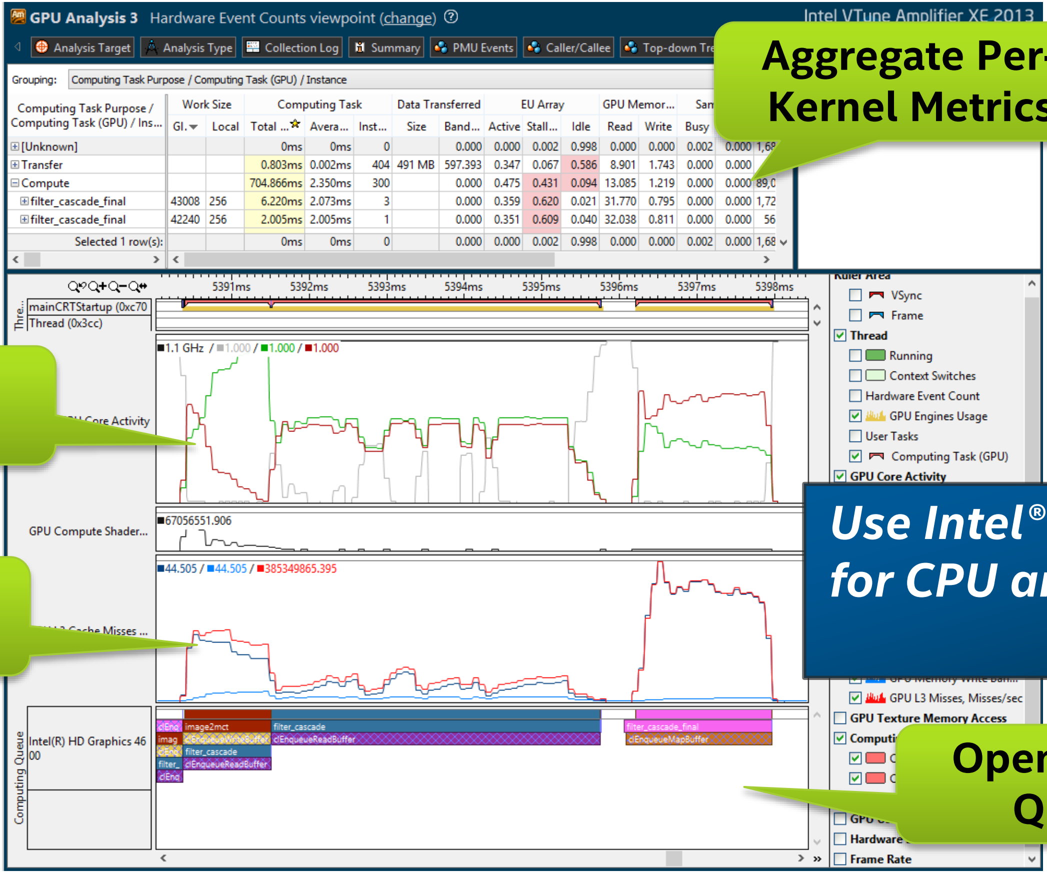
Aggregate Per-Kernel Metrics

EU Activity

Memory Traffic

OpenCL™ Command Queue Profile

Intel® VTune™ Amplifier XE 2015



Aggregate Per-Kernel Metrics

EU Activity

Memory Traffic

Use Intel® Vtune™ Amplifier XE for CPU and GPU performance analysis!

OpenCL™ Command Queue Profile

IDF14

Agenda

- Intel® Graphics Introduction
- OpenCV 3.0 on Intel Graphics
- OpenCL™ Applications on Intel® Graphics Architecture
- Optimization Techniques
  - Maximizing Occupancy
  - Optimizing Memory Access
  - Using Registers
  - Maximizing Computation
- Intel® VTune™ Amplifier XE 2013 Support for OpenCL Applications
- New OpenCL 2.0 Features
- Summary / Questions

Exciting New Features in OpenCL™ 2.0 Standard

- Intel® Processor Graphics Gen8 supports the OpenCL™ 2.0 standard, including

OpenCL

New!

Go to intel.com/software/opencl to learn more!

Exciting New Features in OpenCL™ 2.0 Standard

- Intel® Processor Graphics Gen8 supports the OpenCL™ 2.0 standard, including
  - Shared Virtual Memory (coarse- and fine-grained buffer-based SVM)
    - Share pointer-rich data structures (no more marshaling!)
    - Hardware-supported byte-level coherency

OpenCL

New!

Go to intel.com/software/opencl to learn more!

Exciting New Features in OpenCL™ 2.0 Standard

- Intel® Processor Graphics Gen8 supports the OpenCL™ 2.0 standard, including
  - Shared Virtual Memory (coarse- and fine-grained buffer-based SVM)
    - Share pointer-rich data structures (no more marshaling!)
    - Hardware-supported byte-level coherency
  - Nested Parallelism
    - Kernels can enqueue more kernels
    - Great for divide and conquer algorithms

New!

Go to intel.com/software/opencl to learn more!

Exciting New Features in OpenCL™ 2.0 Standard

- Intel® Processor Graphics Gen8 supports the OpenCL™ 2.0 standard, including
  - Shared Virtual Memory (coarse- and fine-grained buffer-based SVM)
    - Share pointer-rich data structures (no more marshaling!)
    - Hardware-supported byte-level coherency
  - Nested Parallelism
    - Kernels can enqueue more kernels
    - Great for divide and conquer algorithms
  - Non-uniform work-group sizes
    - No longer necessary for work group size to evenly divide NDRange size
    - Less conditional code, better SIMD usage, better memory access patterns

OpenCL

New!

Go to intel.com/software/opencl to learn more!

Exciting New Features in OpenCL™ 2.0 Standard

- Intel® Processor Graphics Gen8 supports the OpenCL™ 2.0 standard, including
  - Shared Virtual Memory (coarse- and fine-grained buffer-based SVM)
    - Share pointer-rich data structures (no more marshaling!)
    - Hardware-supported byte-level coherency
  - Nested Parallelism
    - Kernels can enqueue more kernels
    - Great for divide and conquer algorithms
  - Non-uniform work-group sizes
    - No longer necessary for work group size to evenly divide NDRange size
    - Less conditional code, better SIMD usage, better memory access patterns
  - Generic address space
    - In many cases, __local, __global, and __constant can be inferred by the compiler
    - Write generic functions that operate on any address space

New!

Go to intel.com/software/opencl to learn more!

IDF14

Agenda

- Intel® Graphics Introduction
- OpenCV 3.0 on Intel Graphics
- OpenCL™ Applications on Intel® Graphics Architecture
- Optimization Techniques
  - Maximizing Occupancy
  - Optimizing Memory Access
  - Using Registers
  - Maximizing Computation
- Intel® VTune™ Amplifier XE 2013 Support for OpenCL Applications
- New OpenCL 2.0 Features
- Summary / Questions

Summary

Summary

- OpenCL™ applications make excellent use of Intel® Graphics architecture in a standard programming model

Summary

- OpenCL™ applications make excellent use of Intel® Graphics architecture in a standard programming model
- OpenCV uses OpenCL to take advantage of Intel Graphics

Summary

- OpenCL™ applications make excellent use of Intel® Graphics architecture in a standard programming model
- OpenCV uses OpenCL to take advantage of Intel Graphics
- Following optimization advice for Intel Graphics can provide dramatic performance improvements
  - Maximize occupancy
  - Optimize memory accesses
  - Use registers
  - Optimize compute

Summary

- OpenCL™ applications make excellent use of Intel® Graphics architecture in a standard programming model
- OpenCV uses OpenCL to take advantage of Intel Graphics
- Following optimization advice for Intel Graphics can provide dramatic performance improvements
  - Maximize occupancy
  - Optimize memory accesses
  - Use registers
  - Optimize compute
- Use Intel® VTune™ Amplifier to analyze your code and guide optimizations

Acknowledgements

- This presentation would not have been possible without material and review comments from many people – Thank you!
- Ben Ashbaugh, Murali Sundaresan, Stephen Junkins, Deepti Joshi, Tom Craver, Brijender Bharti, Michal Mrozek, Pavan Lanka, Adam Lake, Arnon Peleg, Mostafa Hagog, Raun Krisch, Berna Adalier, Allen Hux, Robert Ioffe, Mike MacPherson, Dan Petre, Konstantin Rodyushkin, Mikhail Letavin, Maxim Shevtsov, Alexander Batushin, Tim Bauer, Ron Miller, Julia Fedorova, Alexander Kurylev, Vitaly Slobodskoy, Scott Janus

Download, Learn, Code, Optimize

- Free download of Intel® SDK for OpenCL™ Applications at: intel.com/software/opencl
- Follow us: @IntelOpenCL
- Contact as through our forum:

http://software.intel.com/en-us/forums/intel-opencl-sdk

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

A PDF of this presentation is available from our Technical Session Catalog: www.intel.com/idfsessionsSF. This URL is also printed on the top of Session Agenda Pages in the Pocket Guide.

Intel
SDK for
OpenCL™
Applications

OpenCL

What is available online?

- ✓ Free Downloads
- ✓ Code Samples
- ✓ Documentation
- ✓ Tech Articles
- ✓ Reviews
- ✓ Forums and Support
- ✓ Webinars

Legal Disclaimer

Legal Disclaimer

- Iris™ Graphics: Iris™ graphics is available on select systems. Consult your system manufacturer.
- 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.

Risk Factors

The above statements and any others in this document that refer to plans and expectations for the second quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Words such as “anticipates,” “expects,” “intends,” “plans,” “believes,” “seeks,” “estimates,” “may,” “will,” “should” and their variations identify forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Many factors could affect Intel’s actual results, and variances from Intel’s current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be important factors that could cause actual results to differ materially from the company’s expectations. Demand for Intel's products is highly variable and, in recent years, Intel has experienced declining orders in the traditional PC market segment. Demand could be different from Intel's expectations due to factors including changes in business and economic conditions; consumer confidence or income levels; customer acceptance of Intel’s and competitors’ products; competitive and pricing pressures, including actions taken by competitors; supply constraints and other disruptions affecting customers; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Intel operates in highly competitive industries and its operations have high costs that are either fixed or difficult to reduce in the short term. Intel's gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; segment product mix; the timing and execution of the manufacturing ramp and associated costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; and product manufacturing quality/yields. Variations in gross margin may also be caused by the timing of Intel product introductions and related expenses, including marketing expenses, and Intel's ability to respond quickly to technological developments and to introduce new products or incorporate new features into existing products, which may result in restructuring and asset impairment charges. Intel's results could be affected by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Intel’s results could be affected by the timing of closing of acquisitions, divestitures and other significant transactions. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust, disclosure and other issues, such as the litigation and regulatory matters described in Intel's SEC filings. An unfavorable ruling could include monetary damages or an injunction prohibiting Intel from manufacturing or selling one or more products, precluding particular business practices, impacting Intel’s ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. A detailed discussion of these and other factors that could affect Intel’s results is included in Intel’s SEC filings, including the company’s most recent reports on Form 10-Q, Form 10-K and earnings release.

Rev. 4/15/14