Improving Performance of OpenCL™ Workloads on Intel® Processors with Profiling Tools

An Introduction…

Michael R. Carroll
Intel Corporation

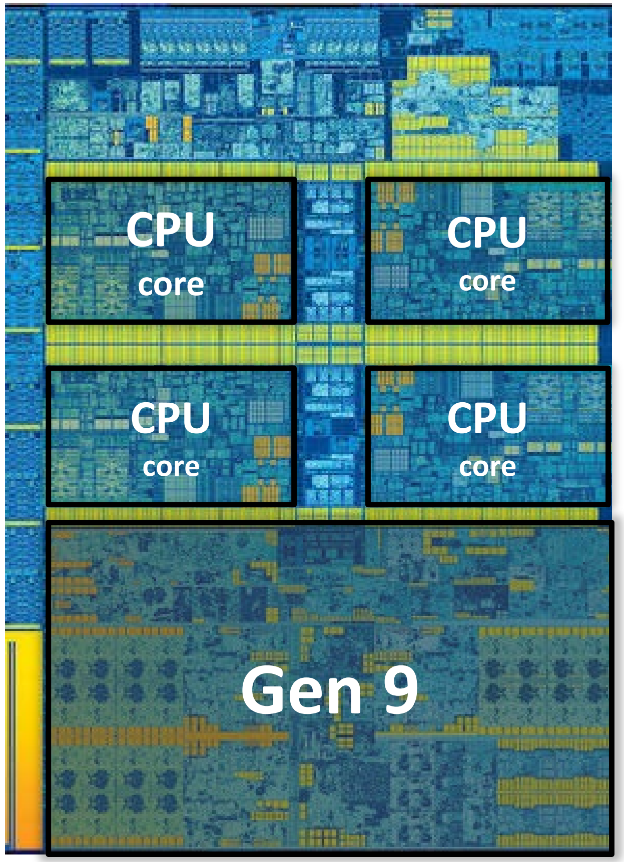
IWOCL – May, 2018 – Oxford, UK

Agenda

- Value Proposition for Tuning and Profiling through OpenCL™
- Generalized Recommendations for Tuning
- Tooling Ecosystem
- Instrumentation results against example source transformations
- Follow up

Why performance tune for OpenCL™?/1

- OpenCL™ code is not performance portable.
- OpenCL™ compilers employ an even wider variety asm transformations beyond classic x86 ecosystem. Low level tuning may not be suitable for many developers.
- Some applications targeting lower power/small form factor OpenCL™ [co]processors… become realtime with tuning. Tuning can make or break the usability of your app!



6th Generation Intel® Core™ i7 (Skylake) Processor

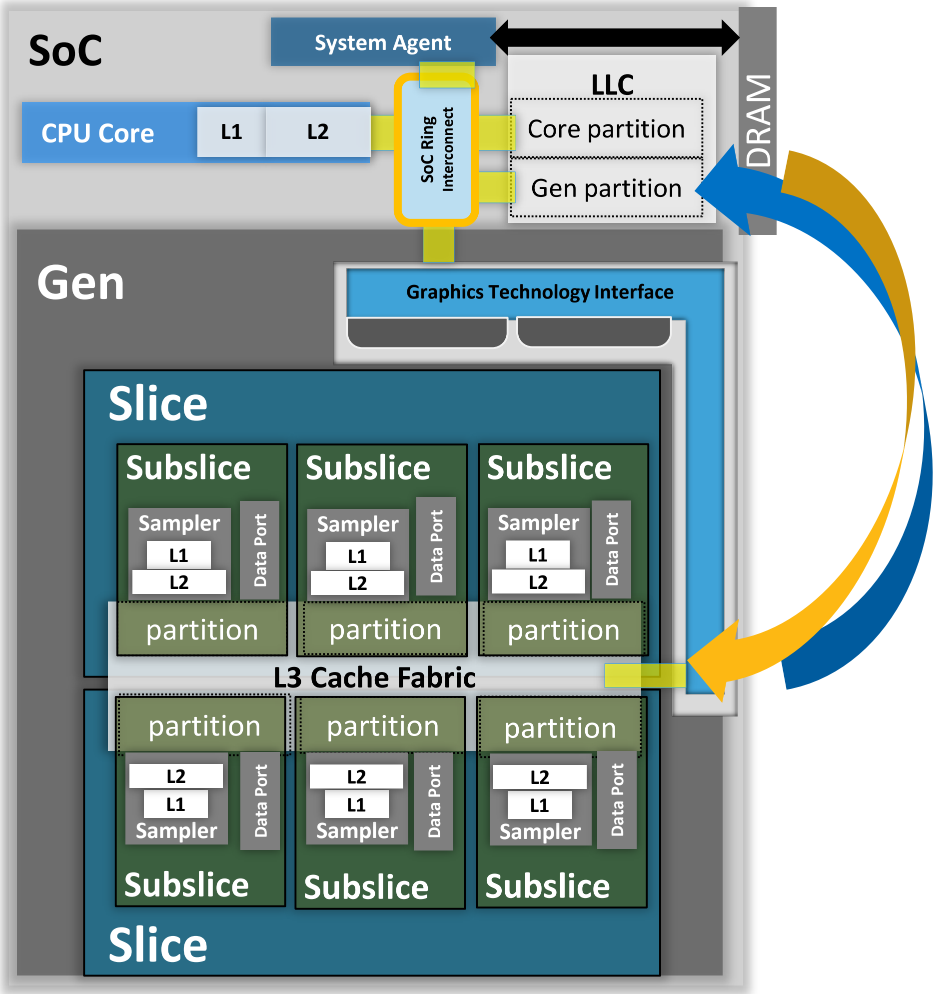
Why performance tune for OpenCL™?/2

- Developer’s maximum impact comes from addressing obvious *performance bottlenecks* low hanging high value fruit…
- Vendors provide advisories and heuristics for how to develop for our devices.
- Instrumented binaries and sampled performance counters allow developers to review hotspots and architectural metrics. Apply best practices and verify their effect with the aid of tools.
- Before we walk through details of profiling… let’s identify some generalized best practices.

Generalized Primary Recommendations/1

Reduce memory bandwidth overhead. Be conscious of your interdevice and intradevice memory access.

- clEnqueueWriteBuffer or clEnqueueMapBuffer? The first can imply memcpy(…) (slow), the other implies LLC style cache access (fast). Pick the right one for your use case.
- On efficient memory access… We'll revisit in the subgroups instrumentation example.



Generalized Primary Recommendations/2

Let the system flush your OpenCL™ queue.

- *n* single OpenCL™ API calls entering ring 0 may be more expensive than enqueueing the *n* calls and having the runtime manage it.
- When you flush… you can still execute on the host!

Generalized Secondary Recommendations/4

Restrict!

- Just like classic x86… leverage restrict qualifier on pointers where applicable. Restrict allows compilers to be more aggressive.
- Consider incorporating restrict into development expectations from the start.

```
__kernel void foo( __constant float* restrict a,
                   __constant float* restrict b,
                   __global float* restrict result)
{

…


}
```

Generalized Secondary Recommendations/5

Appeal to builtins

- Porting code from classic C or C++ won’t leverage builtins of OpenCL-C.
- We have vector types and related built ins, and math intrinsics in OpenCL-C that aren’t provided out of the box many other places. Leverage the legacy of gfx compute!

Generalized Secondary Recommendations/6

Float v int? 16b vs 32b vs 64b?

- Hardware with gfx legacy may benefit from using floats over ints for the same work.
- Write kernels with type defs upfront to play with operand types to see which gives best performance.
- Gen9 has more compute b/w for floating point operations.

Generalized Secondary Recommendations/7

Avoid JIT compiling your kernels

On some devices, JITing kernels may be significantly expensive due to the nature of getting execution code situated to the device.

- We're primarily looking at Gen9 topology here, however FPGA's may have a heavy load time.
- For example, Intel® offers the ioc64 command line compiler tool within Intel® SDK for OpenCL™ Applications

```
clCreateProgramWithBinary(…)
```

x64 Native Tools Command Prompt for VS 2017

```
C:\Intel\OpenCL\sdk\bin\x64>ioc64 -output=kernel.bin -input=kernel.cl
```

Generalized Secondary Recommendations/8

Kernel Compiler options

- Are you OK with a fuzzy epsilon and error propagation? Try building with relaxed math toggles. Relaxed math can improve performance. A full roster of standard options is available in the Khronos documentation. See the options definitions under:
  - https://www.khronos.org/registry/OpenCL/sdk/2.0/docs/man/xhtml/clBuildProgram.html

x64 Native Tools Command Prompt for VS 2017

```
C:\Intel\OpenCL\sdk\bin\x64>ioc64 -output=kernel.bin -bo="-cl-fast-relaxed-math" -input=kernel.cl
```

Generalized Secondary Recommendations/9

Check out OpenCL™ extensions

- Vendors are incentivized to load die area with features. OpenCL™ is a great tool to get to the special function features of a device via extensions. Extension source transformations may bring you better performance.
- Extensions are registered and described at https://www.khronos.org/registry/OpenCL/

Intel Extension Examples

cl_intel_accelerator
cl_intel_advanced_motion_estimation
cl_intel_d3d11_nv12_media_sharing
cl_intel_device_partition_by_names
cl_intel_device_side_avc_motion_estimation
cl_intel_driver_diagnostics
cl_intel_dx9_media_sharing
cl_intel_egl_image_yuv
cl_intel_media_block_io
cl_intel_motion_estimation
cl_intel_packed_yuv
cl_intel_planar_yuv
cl_intel_required_subgroup_size
cl_intel_simultaneous_sharing
cl_intel_subgroups
cl_intel_subgroups_short
cl_intel_thread_local_exec
cl_intel_va_api_media_sharing

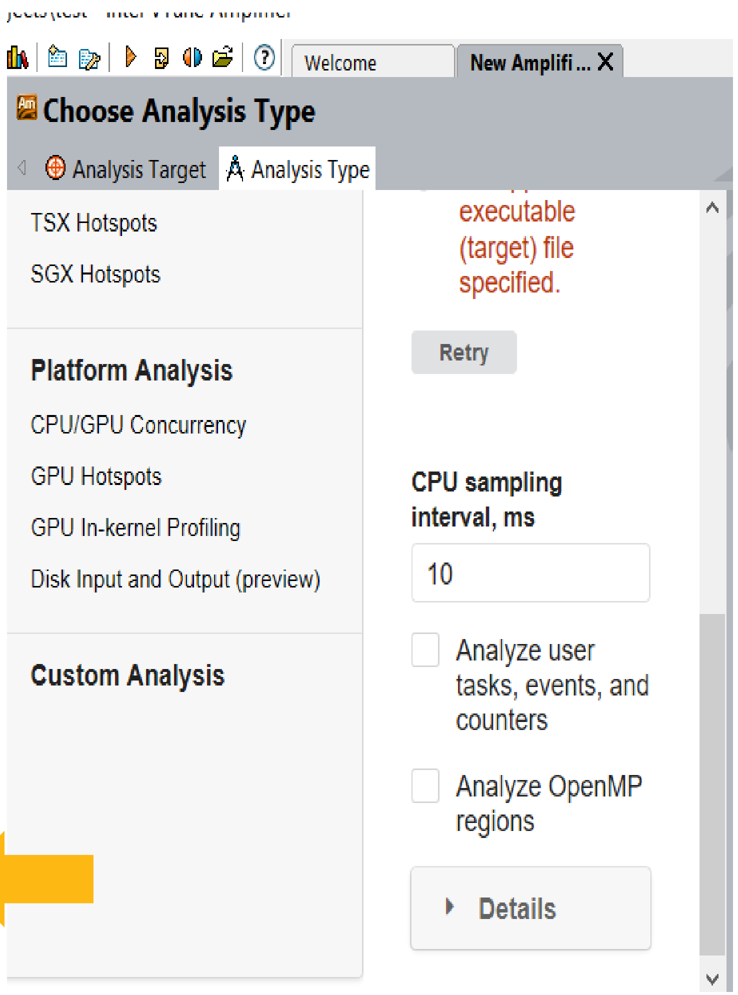
Typical Profiler setup for Instrumentation tools

- Privileged HW registers are preprogrammed to report an event count for an event manifest supplied by the user. Event definitions are provided on a per uArch basis.
- Tools can correlate counted events and timers with debugging symbols. In some cases, some profiling modes induce time multiplexing… so give yourself a decently long runtime.
- Tools may allow for remote connections. Remote connections limit impact to system under test. The may instrument for a subsection of program functionality or time slice.

Example Profiler setup for Instrumentation tools

- Our Vtune™ Example visualizes EUs behavior, command queues, data transfer, OpenCL™ API calls, low-level hardware metrics and much more. It includes in-kernel profiling to identify source-level hotspots in kernel programs.

- Don't forget to turn OpenCL ™ queue profiling on:
  - `cl_queue_properties qprops[] = { CL_QUEUE_PROPERTIES, CL_QUEUE_PROFILING_ENABLE, 0 };`



Example System Setup for our case studies

Centos 7.2 updated to patched Linux 4.4 kernel for i915 driver

OpenCL™ implementation provided via Intel® SRB5.0 package

Intel® Vtune™ Amplifier XE 2018 Update 2 (standalone)

Intel® Core™ i7-6770HW SkullCanyon NUC w/ Intel® Iris® Pro Graphics 580 (Skylake)

4C8T (SMT) 2.6Ghz base 3.6GHz Turbo

L1: 128KB I + 128KB D / core, L2: 1MB / core, L3: 6MB / chip, 16GB DDR4 SODIMM

Integrated: Intel® Iris® Pro Graphics 580: Gen9 u-Arch 350MHz base 950MHz turbo

- Fixed function available for general purpose compute:
  - Texture Sampler
  - Video Motion Estimation
- Tier: GT4e 9 subslices, 72 EUs, 7 threads per execution unit.

SEE BACKUP SLIDES FOR MORE INFO

Profiling case 1) subgroups transformation

cl_intel_subgroups allows for more targeted granularity for novel Intel® h/w. It's a Khronos registered extension that was demo'd in IWOCL'17. We've run this app through default *GPU hotspots* Vtune™ profiling.

Before Transformation: GPU hotspots summary feedback

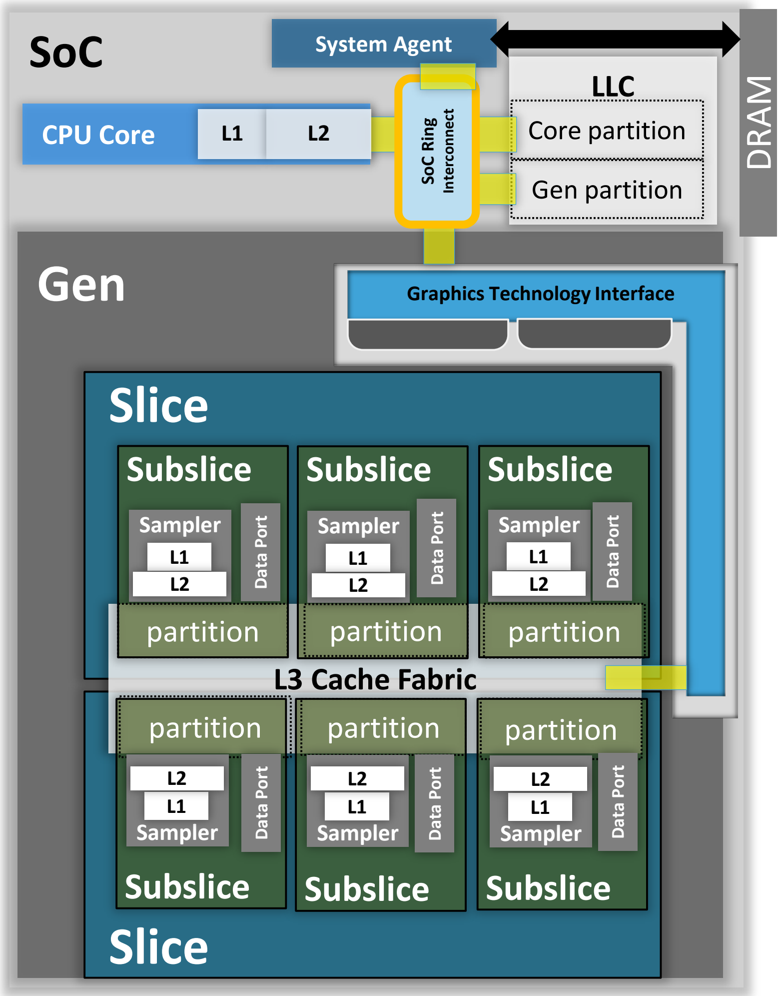
Elapsed Time: 15.520s

GPU Usage: 94.6%

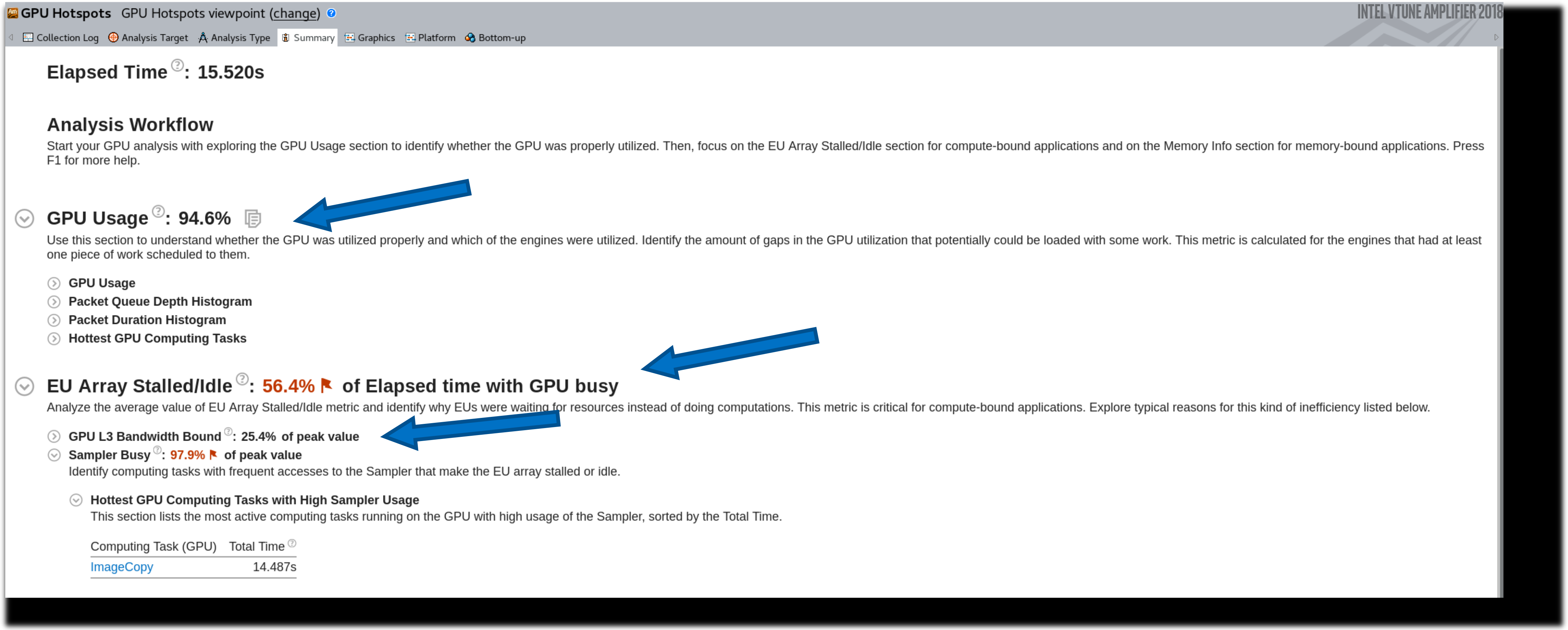
EU Array Stalled/Idle: 56.4% of Elapsed time with GPU busy

GPU L3 Bandwidth Bound: 25.4% of peak value

Sampler Busy: 97.9% of peak value



‘subgroups’ before… ex: GUI summary page



Example Source Transformation: *cl_intel_subgroups*

Example host source reads bmp from disk, allocates space for input and output image. The kernel performs simple copy to output. To demo, a basic time measurement is taken around 10K loops of the NDRange kernel execution.

```
cl::ImageFormat format( CL_R, CL_UNSIGNED_INT8 );
cl::Image2D outputImage(…); //Mapped
cl::Image2D inputImage(…); //Mapped
kernel.setArg( 0, outputImage );
kernel.setArg( 1, inputImage );
cl::NDRange globalSize( imageWidth / 4, imageHeight );
cl::NDRange localSize( 32, 1 );
for( unsigned i = 0; i < cmd.iterations.getValue(); i++ )
    {
        queue.enqueueNDRangeKernel(…);
    }
queue.finish();
```

HOST SIDE!

Case 1) kernel before *cl_intel_subgroups* usage

```
__kernel void ImageCopy( write_only image2d_t dstImage, read_only image2d_t
srcImage )
{

    int work_group_pixel_offset =
            get_group_id(0) * get_enqueued_local_size(0) * 4;
    int work_item_pixel_offset =
            work_group_pixel_offset + get_local_id(0) * 4;
    for( uint pixel = 0; pixel < 4; pixel++ )
    {
        int2    coord = (int2)(
            work_item_pixel_offset + pixel,
            get_global_id(1) );

        uint4   color = read_imageui( srcImage, coord );

        write_imageui( dstImage, coord, color );
    }
}
```

BEFORE!

Case 1) kernel after *cl_intel_subgroups* transformation

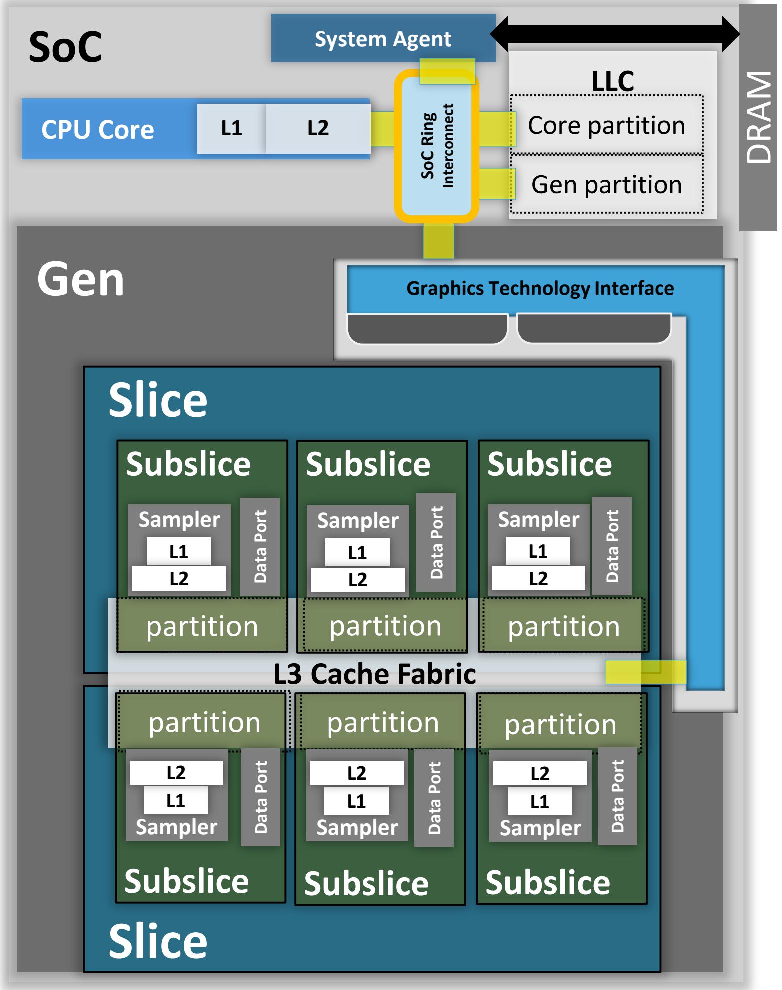
```
__kernel void ImageCopy( write_only image2d_t dstImage, read_only
image2d_t srcImage )
{
    int work_group_byte_offset =
            get_group_id(0) * get_enqueued_local_size(0) * 4;
    int sub_group_byte_offset =
            work_group_byte_offset +
            get_sub_group_id() * get_max_sub_group_size() * 4;

    int2    coord = (int2)(
                sub_group_byte_offset,
                get_global_id(1) );

    uint    color = intel_sub_group_block_read( srcImage, coord );

    intel_sub_group_block_write( dstImage, coord, color );
}
```

AFTER!

Profiling case 1) subgroups transformation

We've reduced our L3 Bandwidth dependency and our reliance on the sampler. This speaks to the notion that the sampler should be used in spots, and L3 access can be more costly vs EUs execution.

SUBGROUPS	Before	After
Whole Program Elapsed Time	15.5s	13.8s
GPU Usage	94%	94%
EU Array Stalled/Idle	56%	79%
GPU L3 Bandwidth Bound	25%	14%
Sampler Busy	98%	0.5%



Next steps….

For more gains with subgroups:

- Dividing the global NDRange space by another factor of four: `cl::NDRange globalSize( imageWidth / 4, imageHeight / 4 );`
- use a 4 wide version of `intel_sub_group_block_write4( dstImage, coord, color );`

To continue the introduction, let’s demonstrate profiling another OpenCL™ extension, *cl_intel_planar_yuv* by way of YUV->RGB colorspace conversion.

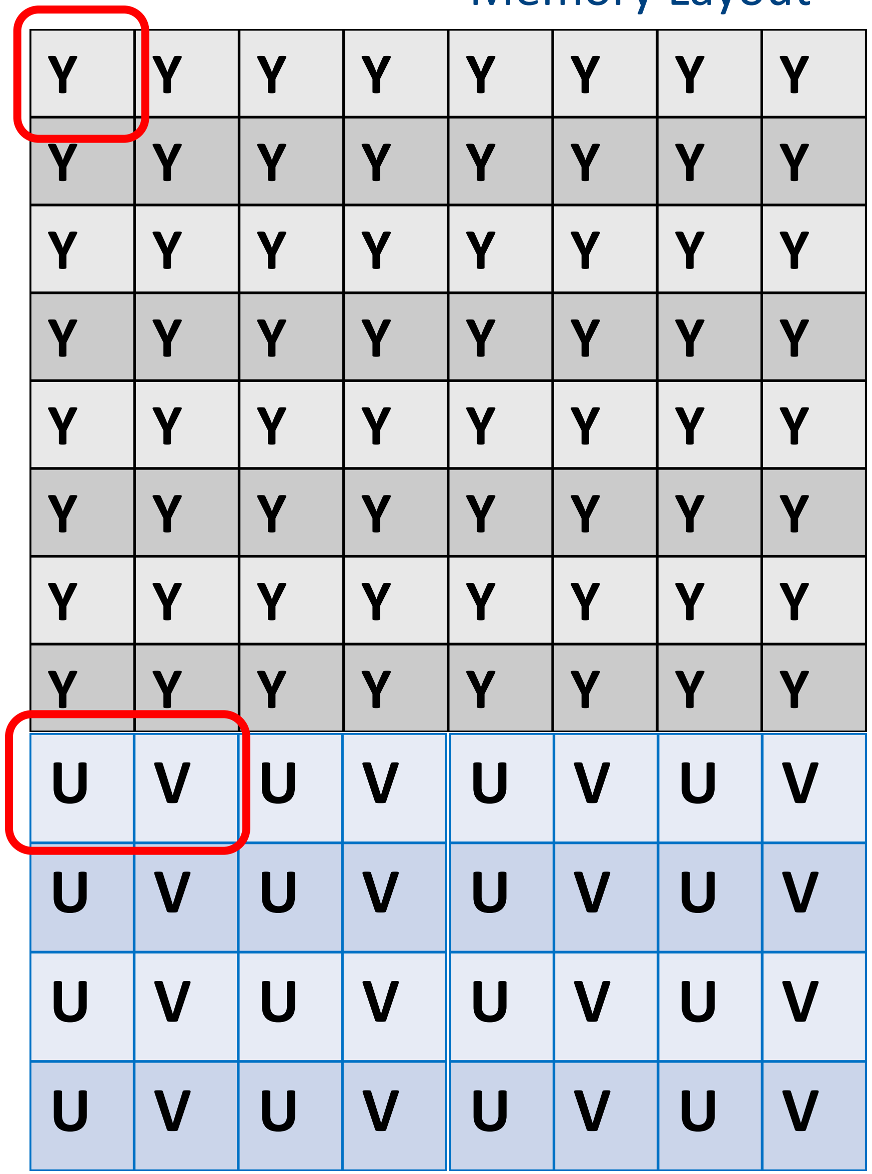
Case 2) *cl_intel_planar_yuv*… concept

Media pipelines often involve conversion from a Luma + Chroma format, like YUV 4:2:0, to RGB – 8bit. This presents an memory access challenge due to bad memory localities. Such a layout is represented by the example to the right.

The red overlays represent luma and chroma image data that we may want to sample together.

This is where the *cl_intel_planar_uv* extension comes in.

Memory Layout



Case 2) Before format converstion yuv to rgb

The extension sampling capability allows for kernels to more easily interpolate between YUV planes to obtain the values. Our example kernel samples the three values, converts to RGB, ships image data to host, then writes to disk.

BEFORE!

```
__kernel void nv12toRGB2Plane(read_only image2d_t nv12YImg, read_only image2d_t nv12UVImg, __global uchar *
out_rgb)
{
    int x = get_global_id(0);
    int y = get_global_id(1);
    int w = get_global_size(0);
    float4 sample_y = read_imagef(nv12YImg, sampler, (int2)(x,y));
    float4 sample_uv = read_imagef(nv12UVImg, sampler, (int2)(x/2,y/2));
    float4 sample = (float4)(sample_uv.y, sample_y.x, sample_uv.x, 0.0f);
    float3 frgb = (float3)(
        (sample.y + 1.402f * (sample.x - 0.5f)),
        (sample.y - 0.34414f * (sample.z - 0.5f) - 0.71414f * (sample.x - 0.5f)),
        (sample.y + 1.77200f * (sample.z - 0.5f))
    ) * 255.0f;
    uchar3 rgb = convert_uchar3(frgb);
    int idx = (x + y * w )* 3;
    out_rgb[idx]     = rgb.x;
    out_rgb[idx + 1] = rgb.y;
    out_rgb[idx + 2] = rgb.z;
}
```

Sample

Write out

Convert

Case 2) after format converstion yuv to rgb

One sample into the image is needed; it's an advantage for this OpenCL h/w.

Sample

Convert

Write out

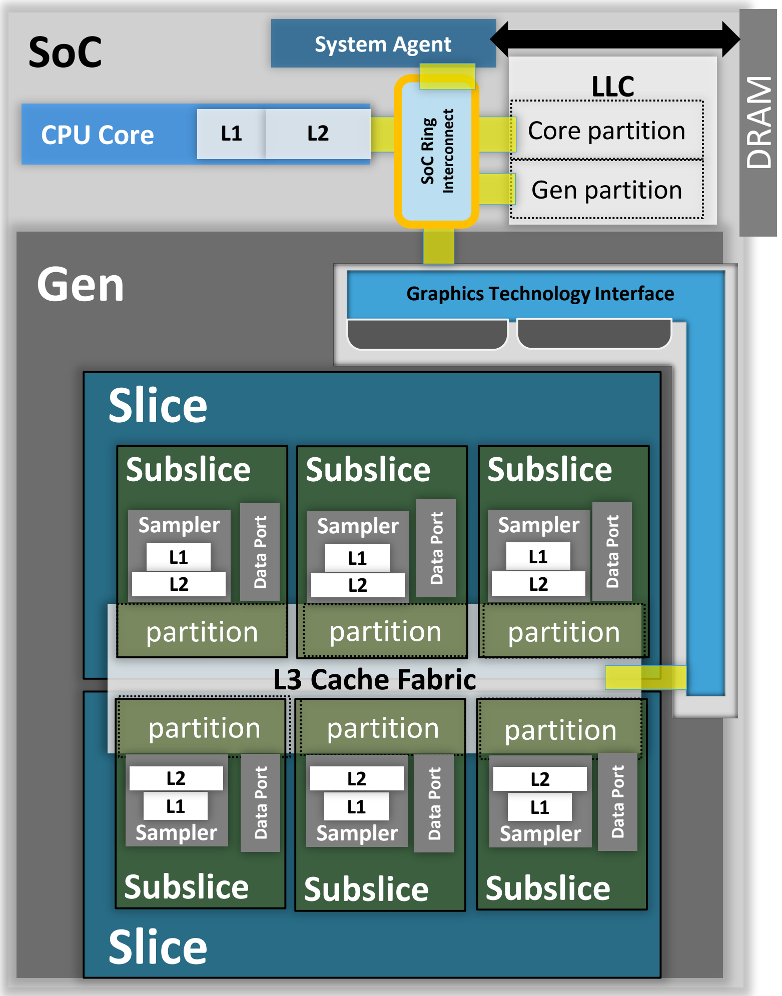
```
__kernel void nv12toRGB(read_only image2d_t nv12YImg, __global uchar * out_rgb)
{
    int x = get_global_id(0);
    int y = get_global_id(1);
    int w = get_global_size(0);
    float4 sample = read_imagef(nv12YImg, sampler, (int2)(x,y));
    float3 frgb =(float3)(
        (yuv.y + 1.402f * (yuv.x - 0.5f)),
        (yuv.y - 0.34414f * (yuv.z - 0.5f) - 0.71414f * (yuv.x - 0.5f)),
        (yuv.y + 1.77200f * (yuv.z - 0.5f))
    ) * 255.0f;
    uchar3 rgb = convert_uchar3(frgb);
    int idx = (x + y * w )* 3;
    out_rgb[idx]     = rgb.x;
    out_rgb[idx + 1] = rgb.y;
    out_rgb[idx + 2] = rgb.z;
}
```

AFTER!

Profiling case 2) format conversion yuv->rgb

cl_intel_planar_yuv alleviates sampler bottleneck decreasing overall execution time.

PLANARUV	Before	After
Whole Program Elapsed Time	9.1s	8.8s
GPU Usage	71%	76%
EU Array Stalled/Idle	64.2%	64.9%
GPU L3 Bandwidth Bound	10%	11%
Sampler Busy	51%	53%
Sampler is Bottleneck	1.2%	0.7%



Summary & Call to Action

- Even domain algorithm experts can attack low hanging fruit with the help of profilers.
- Metrics can be derived with the help of summary tools or scratchpad math.
- Not all OpenCL™ compute resources are optimal for the same task on varying hardware. Texture samplers maybe scarce resources most suitable for use for specific (spatial) access patterns.
- Please provide feedback on OpenCL™ forums for what’s useful out of performance tuning workflows. Share your challenges encountered when profiling. All levels of experience and expertise are welcome and provide mutual benefit. Link: https://software.intel.com/en-us/forums/opencl

Thank you

My email: michael DOT r DOT carroll AT intel DOT com

See backup slides for links and references to related assets.

Legal Notice and Disclaimers

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

No computer system can be absolutely secure.

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

Intel, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

Legal Disclaimer and Optimization Notice

Backup

References:

For more in depth guidance on OpenCL ™ implementations and development on Intel ® hardware:

Intel® SDK for OpenCL™ Applications - https://software.intel.com/en-us/intel-opencl

Intel® FPGA SDK for OpenCL™ - https://www.altera.com/products/design-software/embedded-software-developers/opencl/

Linux getting started guide - https://software.intel.com/en-us/articles/sdk-for-opencl-gsg

Classic: "SRB5.0" Intel® graphics OpenCL™ implementation - https://software.intel.com/en-us/articles/opencl-drivers

New Implementation: "NEO" open source Intel graphics OpenCL™ implementation - http://01.org/compute-runtime

References:

For more guidance on *Intel® VTune™ Amplifier XE* suite on Intel hardware:

- Landing page: https://software.intel.com/en-us/intel-vtune-amplifier-xe
- OpenCL™ Developer Guide for Intel® Processor Graphics - https://software.intel.com/en-us/iocl_opg
    - Developer Guide performance checklist! - https://software.intel.com/en-us/node/540452
- *Intel® VTune™ Amplifier XE: Getting started with OpenCL* performance analysis on Intel® HD Graphics 2014,* Julia Fedorova. https://software.intel.com/en-us/articles/intel-vtune-amplifier-xe-getting-started-with-opencl-performance-analysis-on-intel-hd-graphics
- *Propel with OpenCL - A Deep Dive Workshop to Create, Debug, Analyze and Optimize OpenCL Applications using Intel Tools,* IWOCL '15 *Banerjee, Fedorova, Levy, Kurylev, Sharma,* Stoner, Ioffe
- *The Compute Architecture of Intel Processor Graphics "Gen9"* - https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

Vtune™ bring up/1

Windows 10:

- For CPU target:
  - Execute your application through the *Intel® VTune™ Amplifier XE* GUI front end or from the command line interface. SSE2 Intel® Pentium™ 4 or newer processor.
- For GPU target:
  - 5th generation Intel® Core™ processor or newer (Broadwell) with Gen9 graphics.
  - The Intel® OpenCL™ runtime is included in the Windows graphics driver package available from system vendor or from https://downloadcenter.intel.com/.
  - Developers can look to install Intel® SDK for OpenCL™ Applications package for headers to build for windows target.
  - Execute your application through the *Intel® VTune™ Amplifier XE* GUI front end or from the command line interface.

Vtune™ bring up/2

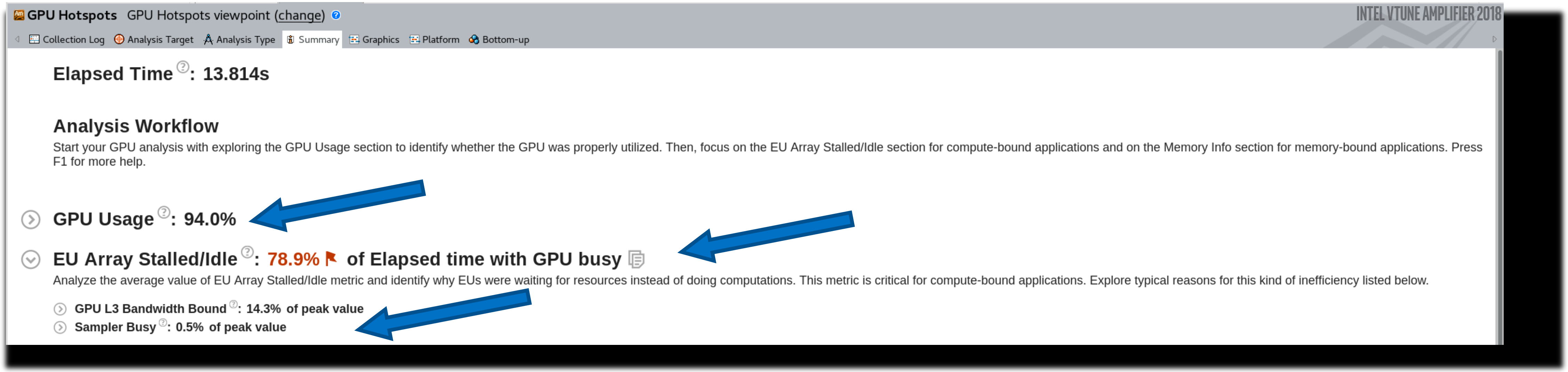
Linux:

- For CPU target:
  - Execute your application through the *Intel® VTune™ Amplifier XE* GUI front end or from the command line interface. SSE2 instruction set, Intel® Pentium™ 4 or newer processor.
- For GPU target:
  - 5th generation Intel® Core™ processor or newer (Broadwell).
  - The Intel® OpenCL™ runtime is either:
    - From the "SRB5.0" package
    - Or the Intel® compute runtime "NEO" package.
  - Install Intel® SDK for OpenCL™ Applications package for headers and libraries to build and for instrumentation.
  - Centos 7.3 or newer
    - Linux 4.4 w/ Intel® provided kernel patches (SRB5.0) or Linux 4.14 (NEO)
    - kernels need CONFIG_DRM_I915_LOW_LEVEL_TRACEPOINTS=y and CONFIG_EXPERT=y turned on.
    - Libmd dependency from here https://github.com/intel/metrics-discovery.
    - Turn profiling enablement on during command queue setup in src via command queue properties.
    - May function with other Linux distros provided correct Linux kernel. (Ubuntu)
    - See the getting started guide for more info.
      - https://software.intel.com/en-us/articles/sdk-for-opencl-gsg
- Forums: https://software.intel.com/en-us/forums/opencl

Misc References:

- Extension registry:
  - https://www.khronos.org/registry/OpenCL/
- GPU Metrics reference (definitions for composite metrics):
  - https://software.intel.com/en-us/vtune-amplifier-help-gpu-metrics-reference
- See https://ark.intel.com to look up capabilities for various Intel processors

‘subgroups’ after gui example…



Acknowledgements

Adam Herr

Jeffrey McAllister

Ben Ashbaugh

Julia Fedorova

Jon Webb (SCEA)

intel®