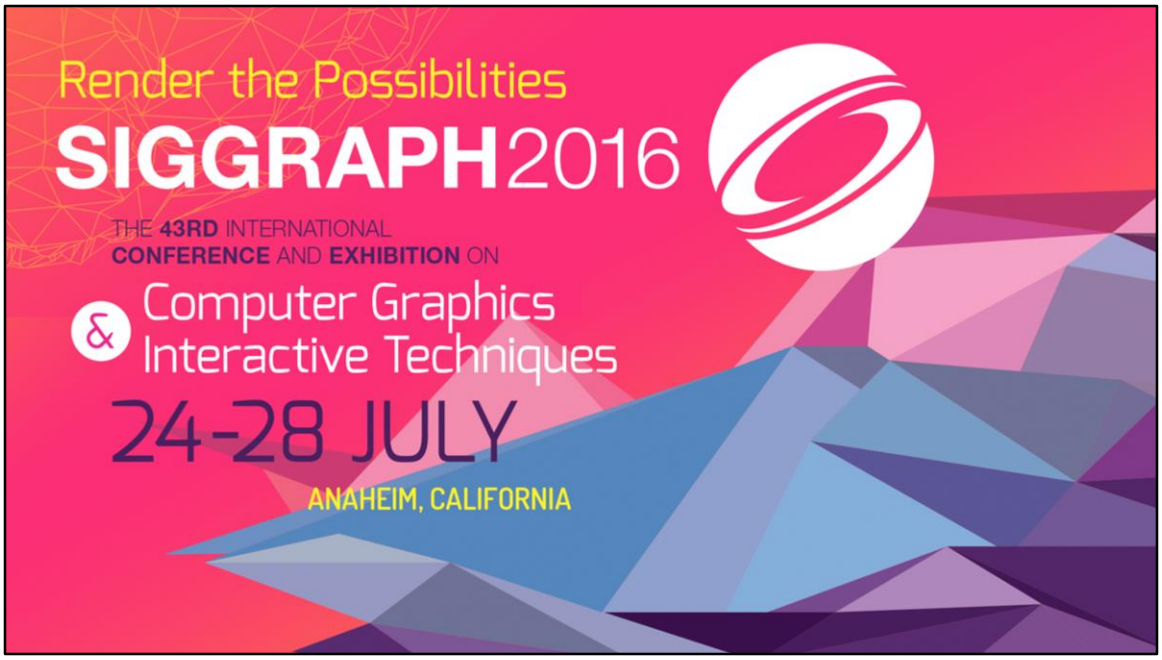
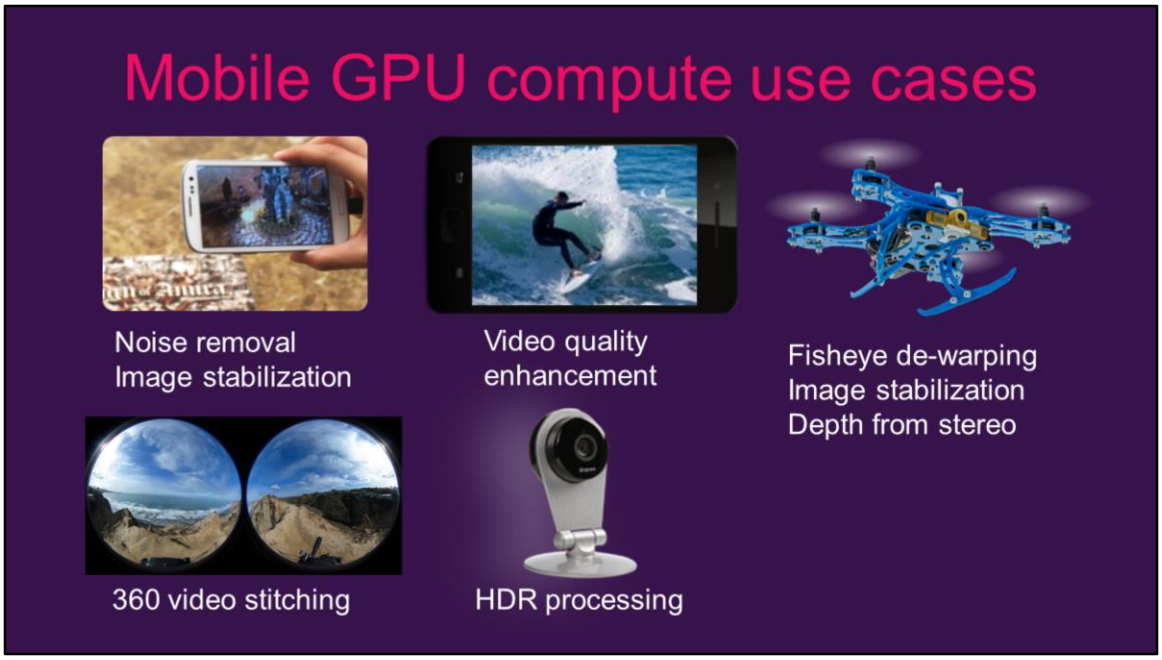

Render the Possibilities
SIGGRAPH2016
THE 43RD INTERNATIONAL
CONFERENCE AND EXHIBITION ON
&
Computer Graphics
Interactive Techniques
24-28 JULY
ANAHEIM, CALIFORNIA


Render the Possibilities
SIGGRAPH2016
THE 43RD INTERNATIONAL
CONFERENCE AND EXHIBITION ON
Computer Graphics
& Interactive Techniques
QUALCOMM®
Video Pre-processing on Mobile GPU –
A Case Study on Performance and Power
Tuning
Jay Yun
Qualcomm Technologies, Inc

Compute on mobile GPU

- Uses standard programming API.
- Used by IHVs to accelerate certain camera, video & vision solutions
  - When using GPU yields best result.
- Used by OEMs to accelerate native apps.
- Application performance benefits from tuning to specific hardware.

IHV means Independent Hardware Vendor. Example is Qualcomm Technologies Inc. that makes Snapdragon™ processors.
OEM means Original Equipment Manufacturer. Examples are smartphone manufacturers.
Tuning applications to specific hardware is not unique to mobile GPUs; even for discrete GPUs, architecture specific tuning is often required



For smartphone market, camera applications were the first to adapt GPU Compute (a.k.a., GPGPU), and for many years remained as the dominant use case for mobile GPU Compute.

Primary use cases for camera have been noise reduction such as spatial or temporal denoising, chroma aberration correction, radial noise reduction as well as lens shading correction.

Image stabilization for camera and drone applications is one of the most widely used non-rendering GPU application. OpenGL ES is the main API in such case. Image stabilization may include rolling shutter effect removal.

Video post-processing is also a key use case where the algorithm tries to remove artifacts or enhance details after scaling as well as improving color for better user experience. Many smartphone OEMs provide their own applications in order to provide differentiating image quality enhancement features.

Recently with popularity of VR, many 360 cameras are gaining traction. These cameras either apply stitching algorithm at runtime or use post-processing on smartphone or PC after capture. In some cases, stitching is combined with dewarping during playback.

HDR means High Dynamic Range. This is an overloaded term; in this context, it applies to enhancing dynamic range during video capture for security camera

products. Security cameras do not have ability to control the lighting conditions, but need to be able to record objects that are in shadow clearly, for obvious reasons.

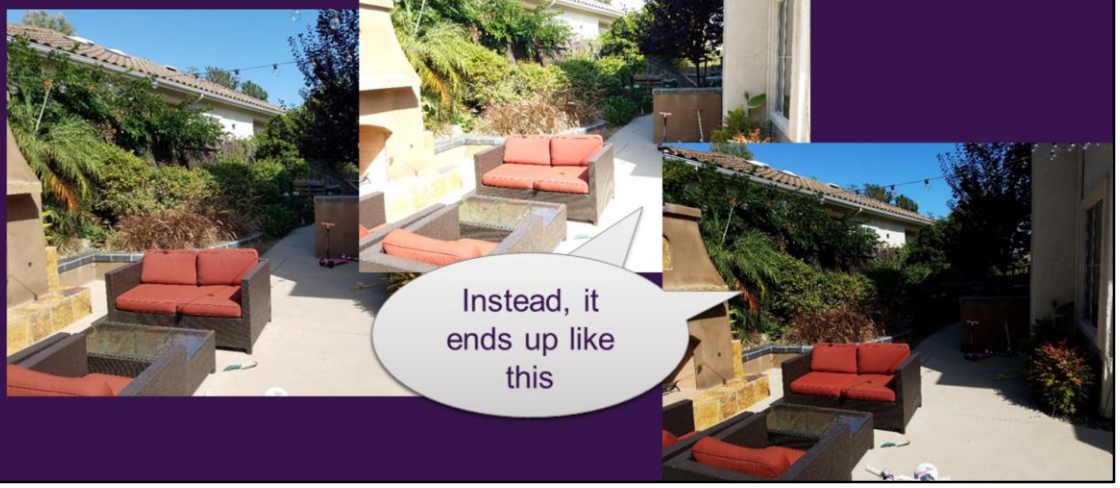
Focus of this talk

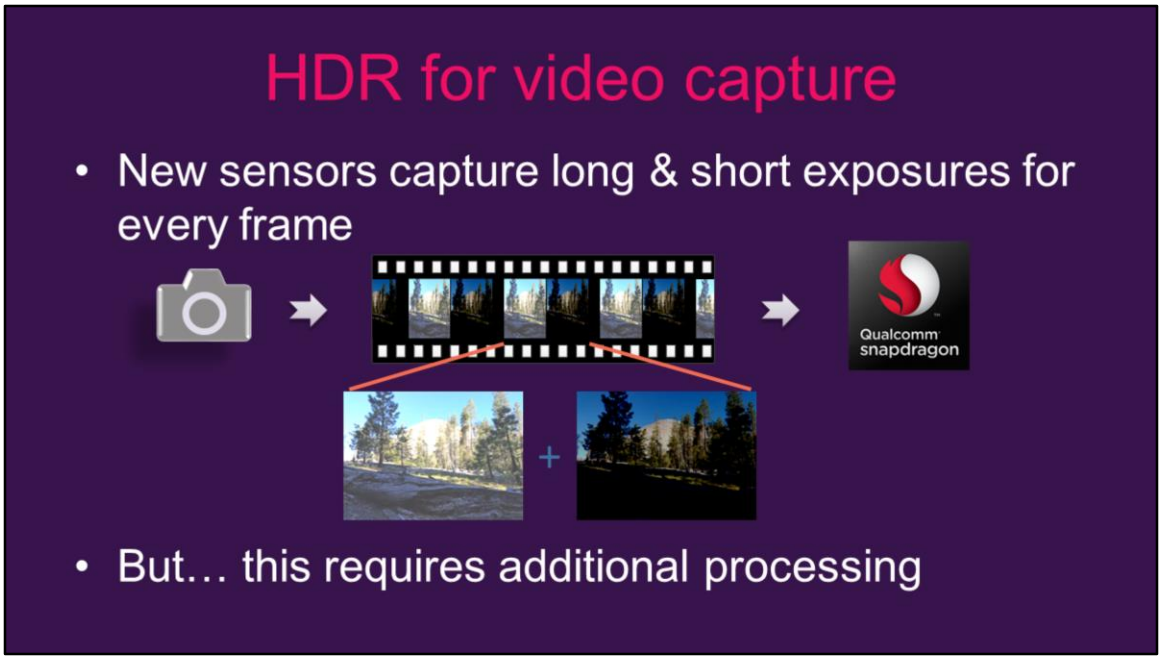
- Compute on mobile GPU is gaining traction, what can we do to help make it pervasive?
- Performance optimization is one problem we need to address.
- In this talk, we will select one use case & show key optimization techniques.

HDR problem in photography

Want to take a picture like this

HDR problem in photography





Typically, raw data from camera sensor is streamed directly to an ISP (Image Signal Processor) which is a hardware unit that performs series of image processing and color conversion to produce visually appealing images.
For new sensors that support “HDR” features, the raw data needs to be pre-processed prior to being streamed to ISP, in order to combine the long and short exposure frames.
There are many variations on sensor’s HDR features, depending on sensor manufactures. Sensor manufactures keep advancing this technology in order to yield better solution that reduces motion artifacts, etc.
In future ISP hardware, HDR processing could possibly become built-in, which removes the requirement of the additional “software” stage.

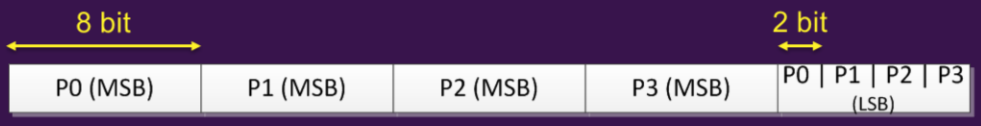
Goals for optimization

- Note, most mobile use cases are memory bound
- Merge stages to reduce intermediate data traffic
- Vectorize and coalesce as much as possible
- Find ideal workgroup size
  - Improves cache performance
  - Enhances latency hiding and overall GPU utilization
- Use 16 bit data and lower-precision math

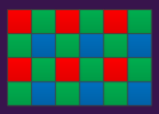
Some of these techniques are standard optimization techniques that benefit many GPU Compute use cases.

Data format challenges

- 10 bits/pix, tightly packed in “MIPI RAW10” format



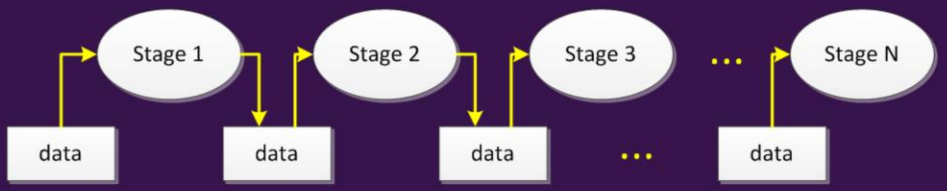
- Bayer pattern
  - Each pixel has 1 color component
  - Alternating pattern across row & column



- This is not a recognizable GPU texture format

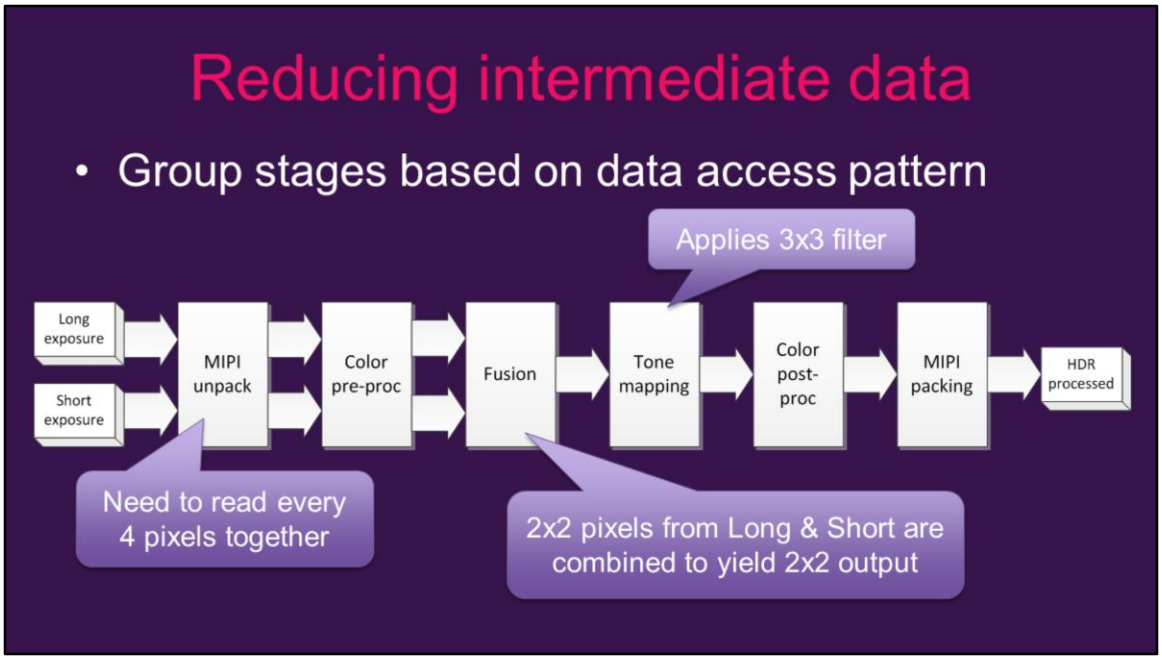
Complexity challenges

- Multiple stage algorithm: a lot of intermediate data

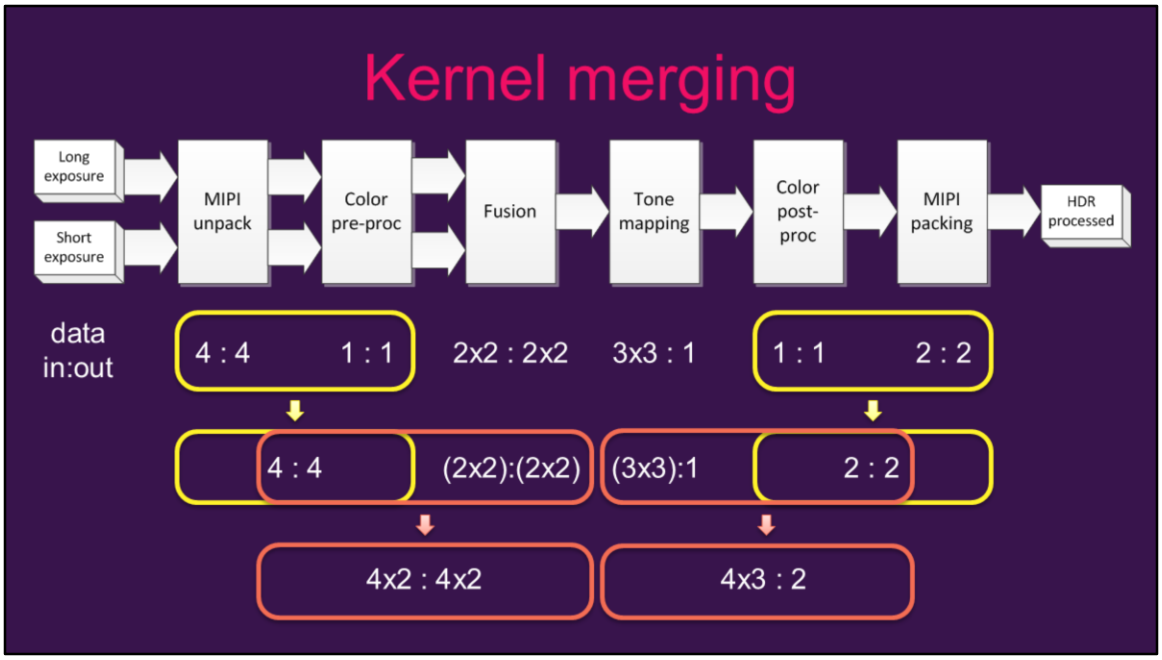


- Perf req't: UHD 30fps for Snapdragon™ 650
- Thermally constrained, so run at nominal clock

UHD is 3840 × 2160, aka "4K"
Running at nominal clock simply means to run at default mode which allows the SoC to dynamically adjust clock frequency and voltage of the hardware modules in order to yield best performance/watt outcome. Typically, this means running the clock at much lower than peak level.



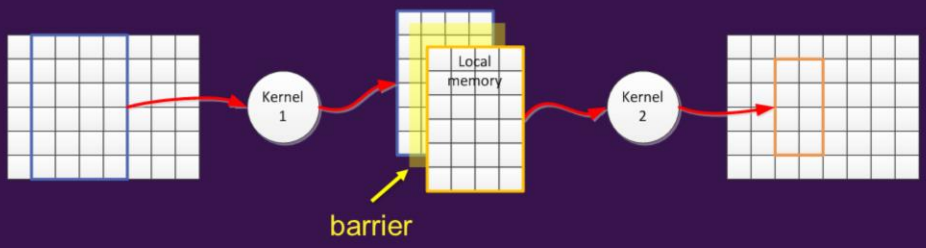
The purpose of this page is to show that for each stage, the developer needs to identify data packing requirements for input and output.



Shown here is the most natural—and easy—way of combining kernels, by paying attention to data grouping requirements. This allows kernels to be combined without requiring additional usage of registers, meaning the register usage of the combined kernel will not be more than the registers used by either of the original kernels.

Uber kernel – merge everything

- Store intermediate data in local memory



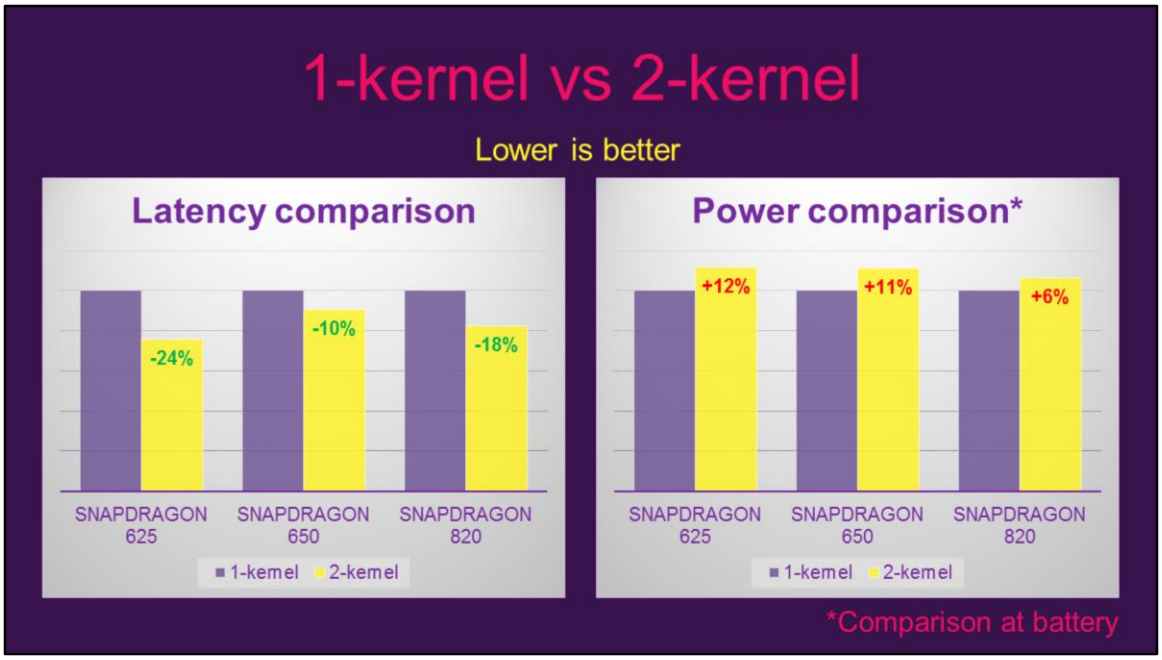
  - Use barrier to synchronize between two stages
  - Eliminates data traffic to DDR but may increase register pressure

Barrier synchronization is required to ensure that all processing from the first stage is completed before starting the next stage.
This synchronization often comes with a hidden cost, which is the latency required to reach the synchronization point across all work items working on the first stage.

Workgroup (WG) size tuning

- Profile performance using different x, y sizes
  - Cache hit rate is related to WG size.
- local_work_size=NULL may not be the best option
  - The driver attempts to pick a reasonable workgroup size but this will most likely not be the best size.
- Larger workgroups are better at hiding memory latency but…
  - if too large, they may lead to cache thrashing

For 2D workgroup, its shape (e.g., width vs height) is as important as the total size.
Upper limit for the workgroup size of a particular kernel is determined by a number of factors including register usage (which is related to the complexity of the kernel) and presence of barrier instructions.
"local_work_size=NULL" is an OpenCL feature. For OpenGL ES, the work group size needs to be specified in the compute shader.
If the application needs to run on multiple devices, it is important to try different devices as well.



Here, we are showing comparisons of two solutions, “1-kernel” which is the “uber” kernel that combines all stages into a single kernel using local memory, and “2-kernel” which does not use local memory, and requires writing intermediate data to DDR.

Latency comparison chart is showing the performance of these two solutions, normalized to the 1-kernel case. Performance measurement includes memory reads and writes and any software overhead for launching GPU kernels.

Power comparison chart is showing the power consumption at battery, normalized to the 1-kernel case.

These two charts show that the 1-kernel case is more power efficient but has lower performance compared to the 2-kernel case, likely due to having lower parallelism from higher register usage.

Summary

- Memory bandwidth reduction is key to good performance and power efficiency
- Use local memory to combine kernels, to reduce power consumption
- Pay attention to WG tuning
- Acknowledgement
  - Xujie Zhang, Vijay Ganugapati