FTF FREESCALE TECHNOLOGY FORUM 2014

GPU Compute: An Introduction and Forward Facing Look

FTF-SDS-F0194

Rick Tewell | Senior Graphics Architect

APR.2014

freescale™

External Use

Session Introduction

- Today's GPUs are extremely parallelized math processing engines. They are capable of bringing tremendous co-processing capabilities to general purpose applications far beyond graphics.
- In this session you will be presented with how to harness and leverage these increasingly important computational capabilities.
- I am the Senior Graphics Architect and lead of the Freescale Graphics Technology Engineering Center (GTEC).
- This session should run around one hour – hope to have time for a few questions at the end.

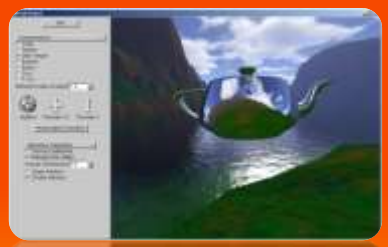
GTEC

An Organized Team Of Customer Centric Graphics Technology Experts Focused On Strategy, Architecture, Software And Hardware *Solutions* At The IC *And* System Level For Both Traditional Graphics *And* GPU Compute Applications

Functional Teams

GPU South

- Kernel Level Drivers
- API Level Libraries (GLES, VG, etc.)
- Validate New SW from GPU IP Provider
- Debugging / Test
- Customer Support



Graphics

- Graphics “Above” the API Level
- Benchmarking
- Testing
- Demonstration Applications
- Customer Support / Education

Compute

- OpenCL
- Compute Shaders
- Benchmarking
- Testing
- Customer Support / Education
- Demonstration Applications

GTEC

Functional Teams

Studio Solutions

- Conceptual Art
- 2D / 3D artwork creation
- Artwork Optimization
- Technical Art
- Tutorials
- Demos
- Benchmarks

3rd Party Ecosystem

- Graphics Engines (2D / 3D)
- Debuggers
- Performance Analysis
- HMI Tooling
- OS Vendors

Session Objectives

- After completing this session you will be able to:
  - Describe the benefits of GPU compute
  - Install and run a sample GPU compute program using OpenCL
  - Find additional resources to go deeper and further with GPU compute

Agenda

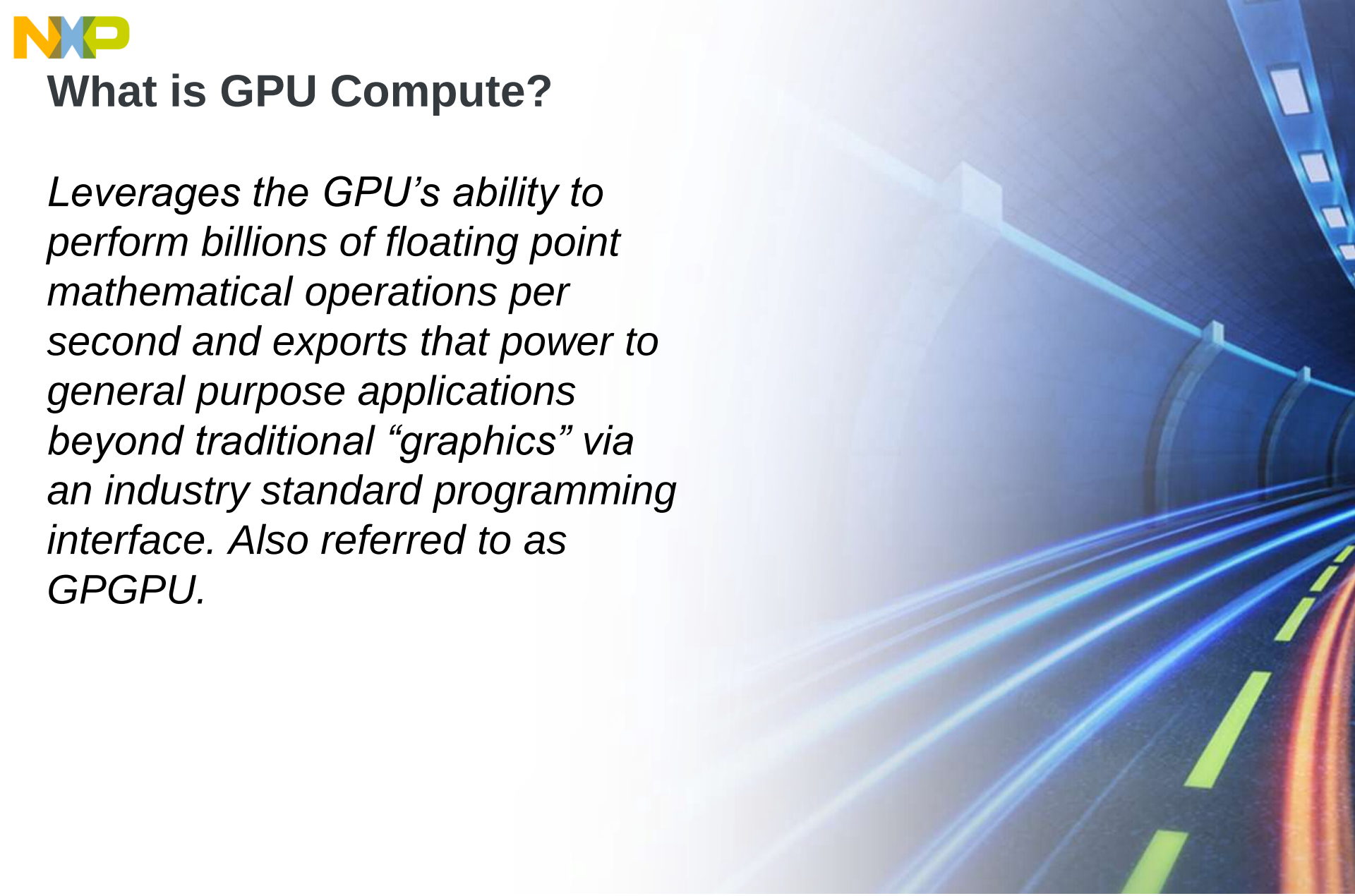
- What is GPU Compute?
- i.MX 6 GPU Compute Capabilities
- How to use GPU Compute - OpenCL Introduction
- OpenCL SDK
- Future of GPU Compute
- Resources

Agenda

- What is GPU Compute?
- i.MX 6 GPU Compute Capabilities
- How to use GPU Compute - OpenCL Introduction
- OpenCL SDK
- Future of GPU Compute
- Resources

What is GPU Compute?

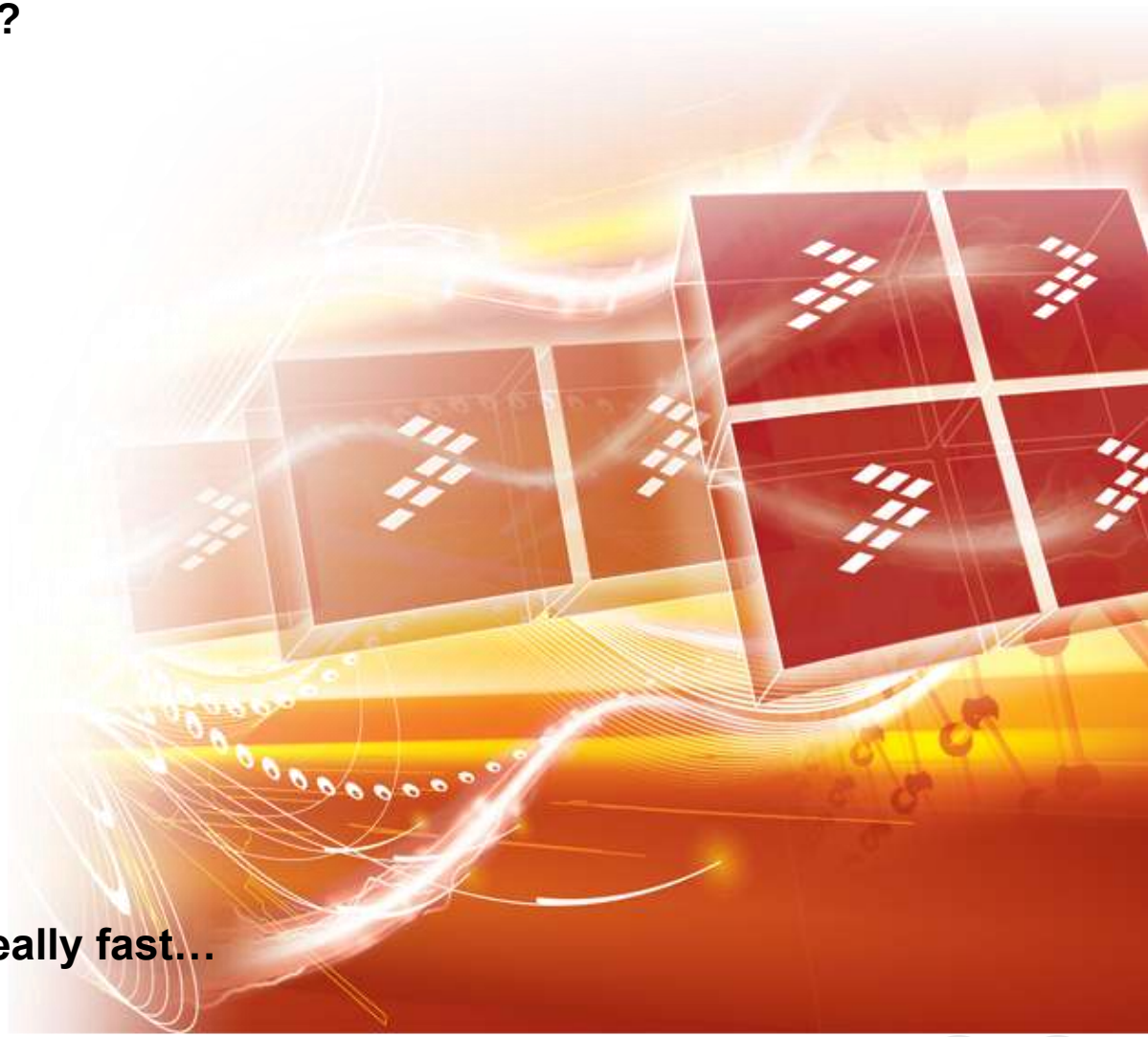
Leverages the GPU's ability to perform billions of floating point mathematical operations per second and exports that power to general purpose applications beyond traditional "graphics" via an industry standard programming interface. Also referred to as GPGPU.



What is GPU Compute?

What can it do mathematically?

- Dense Linear Algebra
- Sparse Linear Algebra
- Spectral Methods
- N-Body Methods
- Structured Grids
- Unstructured Grids
- Map-Reduce & Monte Carlo
- Combinational Logic
- Graph Traversal
- Dynamic Programming
- Backtracking
- Probabilistic Graphical Models
- Finite State Machines
- more…

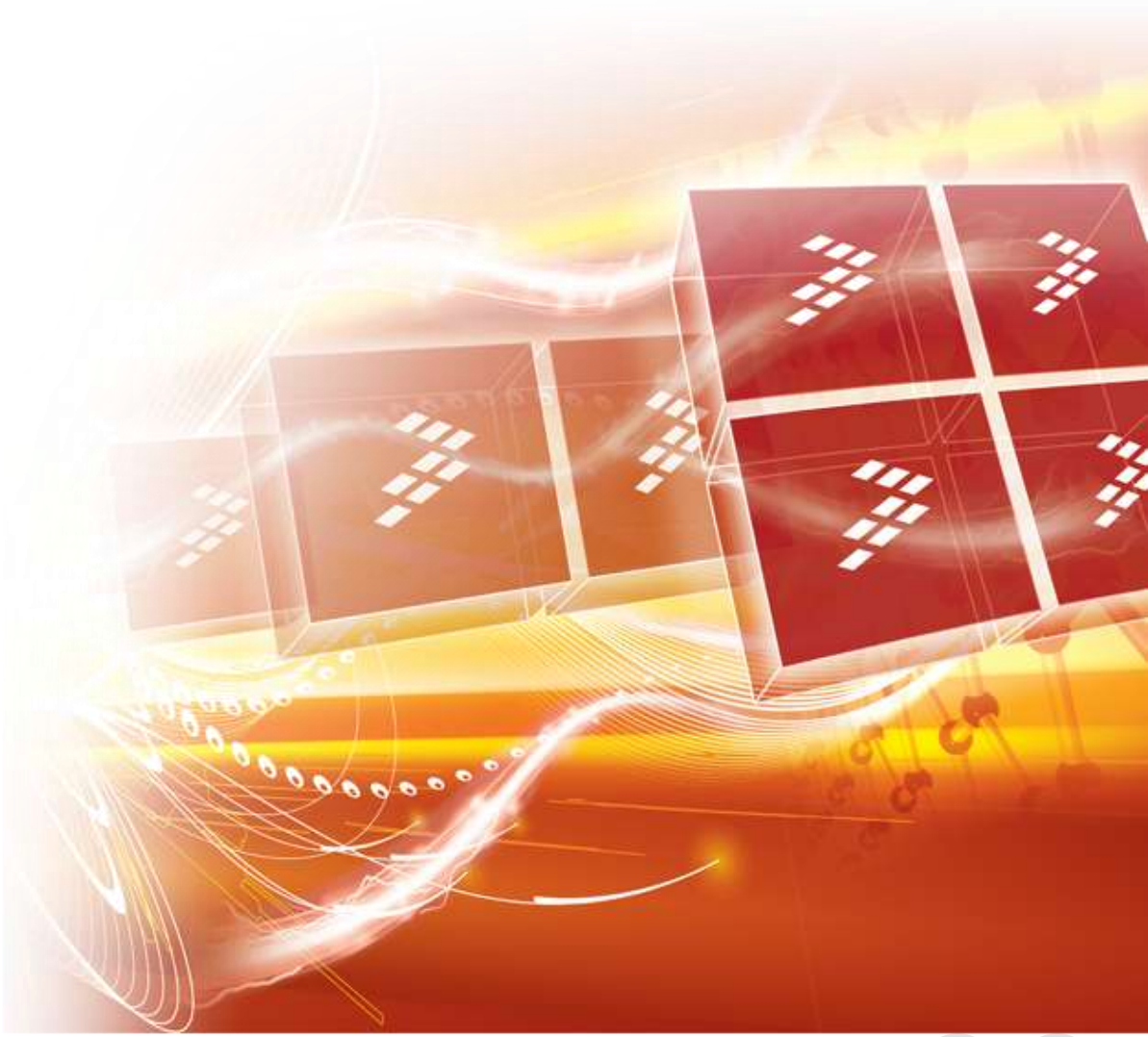


It can do these things really, really fast…

What is GPU Compute?

Everyday use

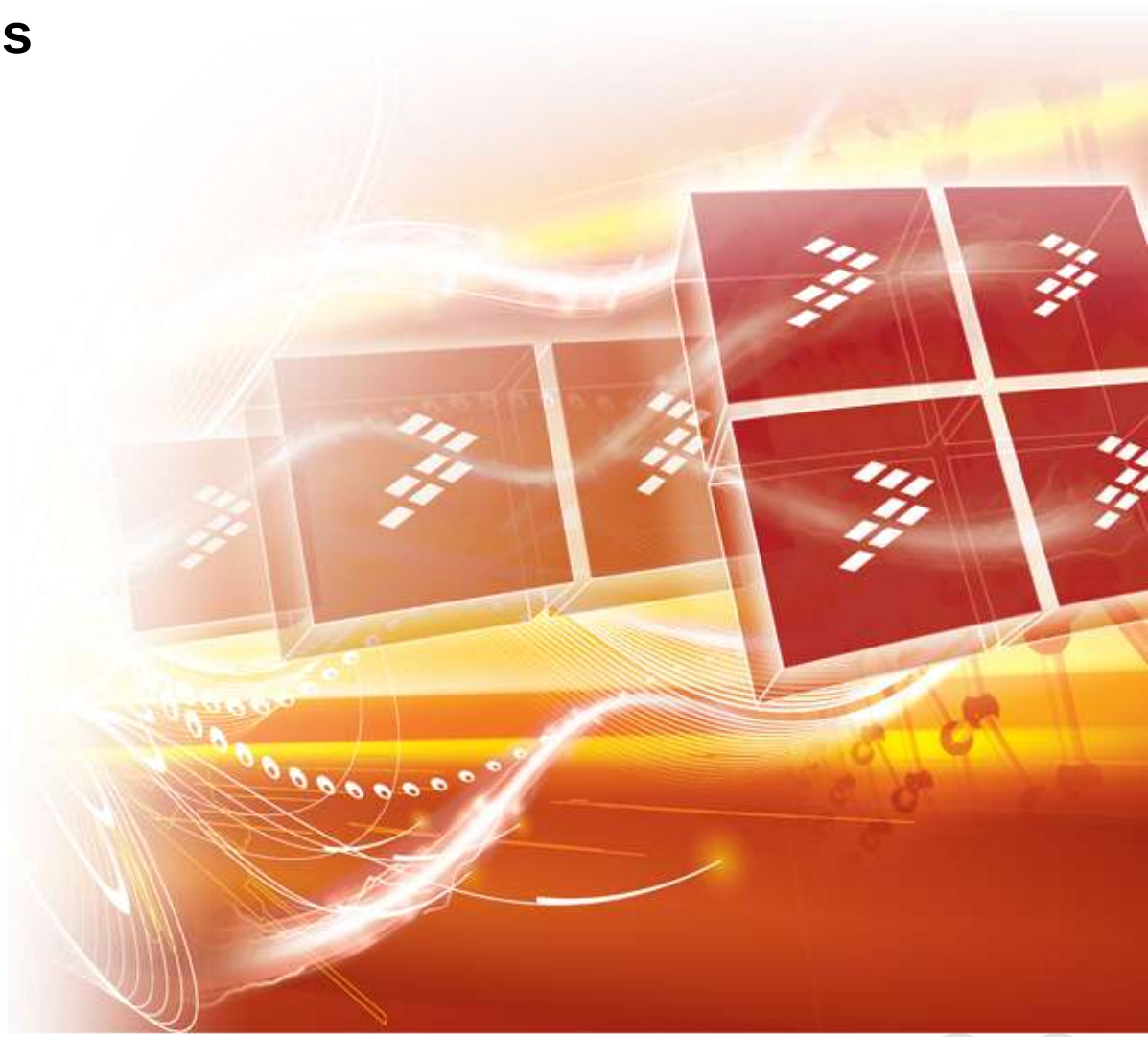
- Gesture Recognition
- Motion Processing
- Computer Vision
- Terrain Generation
- Augmented Reality
- Cryptography
- Speech Recognition
- much more…bounded only by imagination



What is GPU Compute?

Automotive Applications

- Lane Departure
- Pedestrian Detection
- Road Sign Detection
- Blind Spot Detection
- Driver Monitoring
- Collision Avoidance
- more…



What is GPU Compute?

- GPGPU stands for General Purpose Graphics Processing Unit.
- Algorithms well-suited to GPGPU implementation are those that exhibit two properties:
  - *Data parallel*
  - *Throughput intensive*
- Data parallel means that a processor can execute the same operation on different data elements simultaneously.
- Throughput intensive means that the algorithm is going to process lots of data elements, so there will be plenty to operate on in parallel.
- Pixel-based applications such as computer vision and video and image processing are very well suited to GPGPU technology, and for this reason, many of the commercial software packages in these areas now include GPGPU acceleration.

What is GPU Compute?

- CPUs devote a lot of resources (primarily chip area) to make single streams of instructions run fast, including caching to hide memory latency and complex instruction-stream processing (pipelining, out-of-order execution and speculative execution).

- GPUs, on the other hand, use the chip area for hundreds of individual processing elements that execute a single instruction stream on many data elements simultaneously

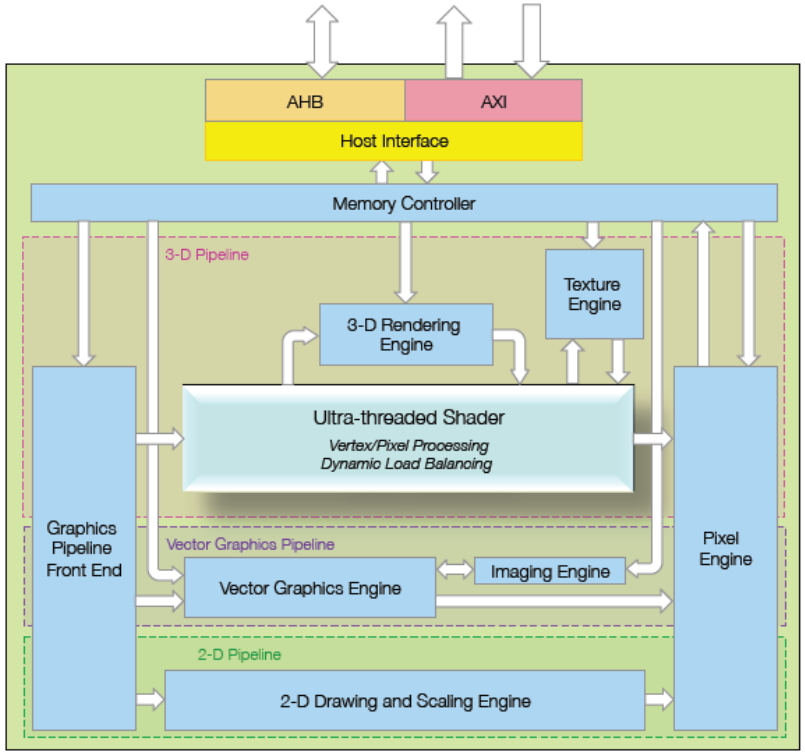
- GPUs can run certain algorithms anywhere from 10 to 100 (or even more) times faster than CPUs

- Gaining this speed-up requires that algorithms are coded to reflect the GPU architecture, and programming for the GPU differs significantly from traditional CPUs. incorporating GPU acceleration into pre-existing codes is more difficult than just moving from one CPU family to another

Agenda

- What is GPU Compute?
- i.MX 6 GPU Compute Capabilities
- How to use GPU Compute - OpenCL Introduction
- OpenCL SDK
- Future of GPU Compute
- Resources

i.MX 6 GPU Compute Capabilities



Host interface

- Connects external AXI and AHB buses to the GCCORE memory controller
- Manages events

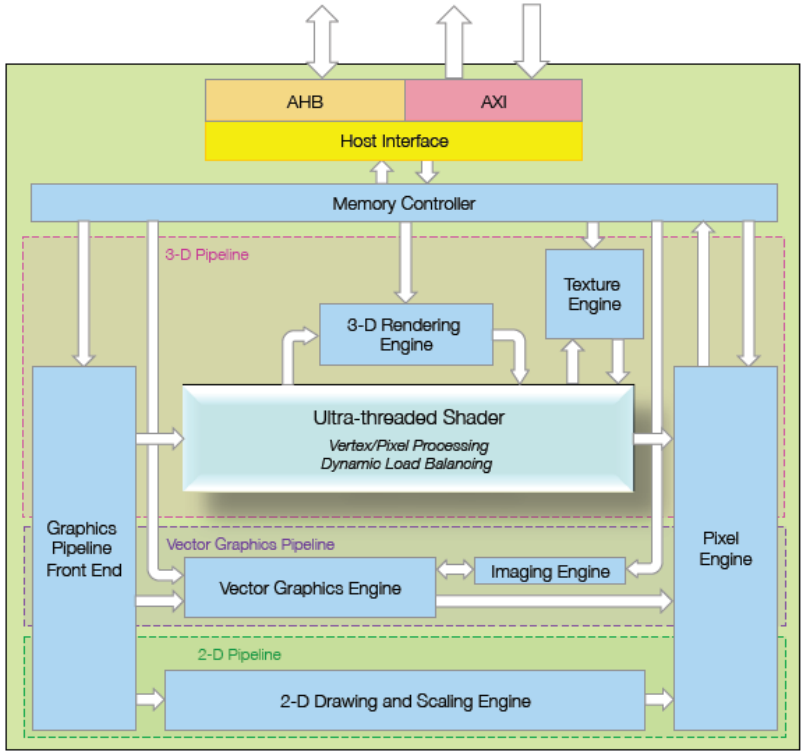
Memory Controller

- Internal memory controller that connects blocks to the host interface for memory requests
- Round robin arbitration

Graphics Pipeline Front End

- Inserts high level primitives and commands into the graphics pipeline
- Contains vertex cache and index buffers

i.MX 6 GPU Compute Capabilities



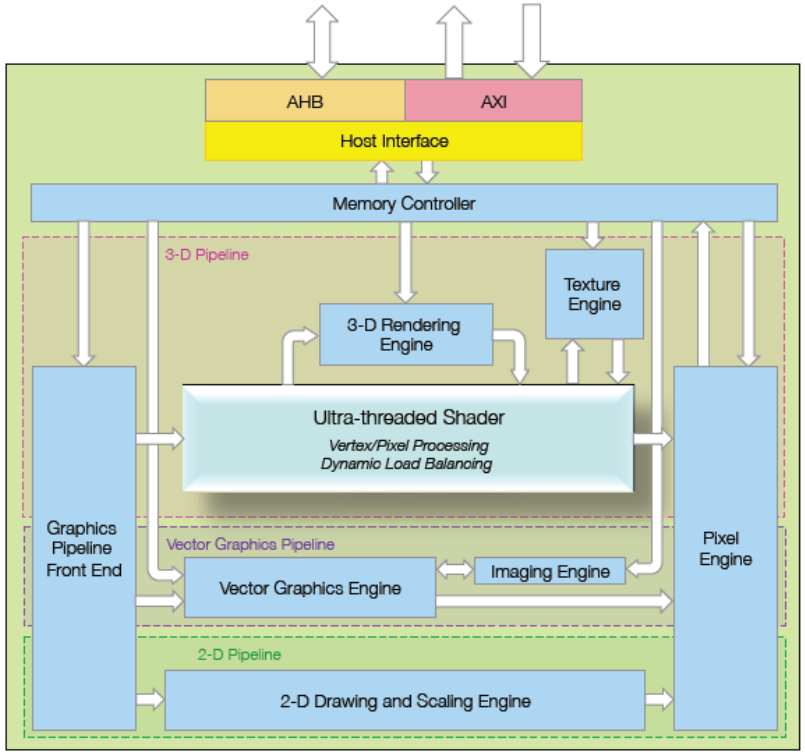
Ultra-threaded Shader

- SIMD processor combines vertex shader and pixel shader
- When used as a vertex shader, performs geometry transformations and lighting computations on vertices
- When used as a pixel shader, applies texels to pixels

3D Rendering Engine

- Converts high level primitives into triangles and lines
- Computes slopes of color attributes and texture coordinates
- Performs clipping
- Computes pixel coordinates and colors for each pixel

i.MX 6 GPU Compute Capabilities



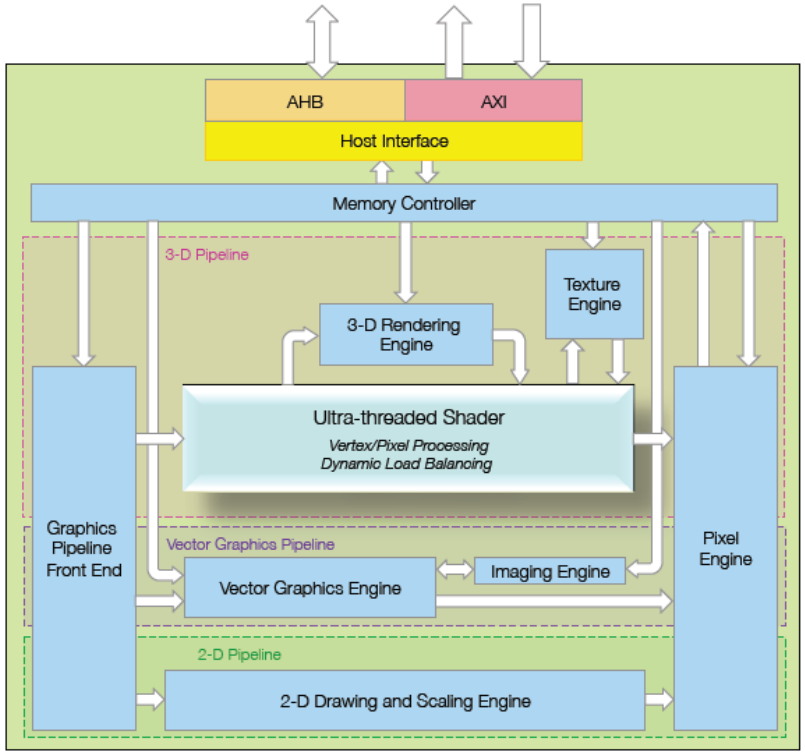
Texture Engine

- Retrieves texture information from memory upon request by the vertex and pixel shader, and then interpolates, filters, and transfers the computed value to the shader

Pixel Engine

- Performs blending, depth, and stencil computations, as well as de-tiling (resolve) and MSAA (Multi Sample Anti Alias)

i.MX 6 GPU Compute Capabilities



Vector Graphics Engine

- Performs tessellation of OpenVG path primitives
- Rasterizes tessellated primitives and rectangles

Imaging Engine

- Generates paint and images
- Performance Gaussian blur or other programmable filters

2D Drawing and Scaling Engine

- Draws 2D graphics primitives
- Perform high quality separable 1, 3, 5, 7, or 9-tap scaling in two passes or 1, 3, or 5-tap filter in one pass
- Performs YUV conversion

i.MX 6 GPU Compute Capabilities

High Performance OpenGL/OpenCL GPU Core

- Vivante GC2000 @ 528MHz (600 MHz shader)
- 88M triangles / sec (176MT/s @ 50% cull)
- 4 Shader Cores: 21.6 GFLOPS
- 1056 M Pixels/s

i.MX 6 GPU Compute Capabilities

- 4x SIMD cores or shader units
- 512 general purpose 128b registers for each of the cores
- Maximum number of instructions for kernels is 512
- OpenCL embedded profile capabilities
- 1-cycle throughput for all shader instructions
- L1 cache of 4KB
- Uniform registers:
  - 168 for vertex shader
  - 64 for fragment shader
- Vec4 / 32-bit float pipeline/core

i.MX 6 GPU Compute Capabilities

GC2000 ALU

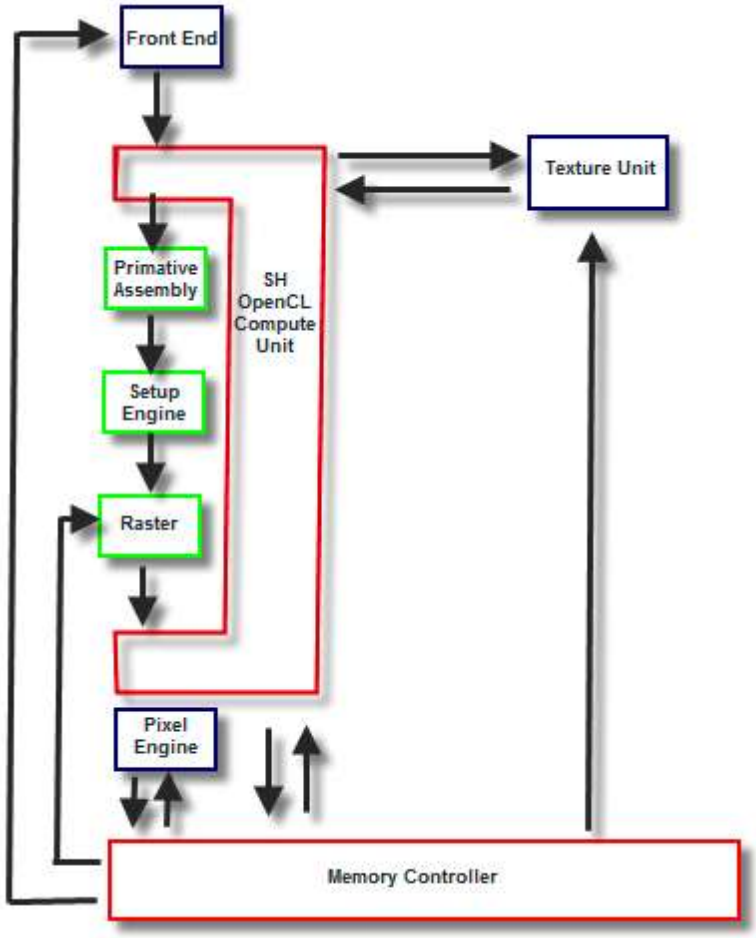
Float	MUL	32	32	32	32
	ADD	32	32	32	32
INT	MUL	32			
	ADD	32			
	ALU				

i.MX 6 GPU Compute Capabilities

Native – HW accelerated math functions

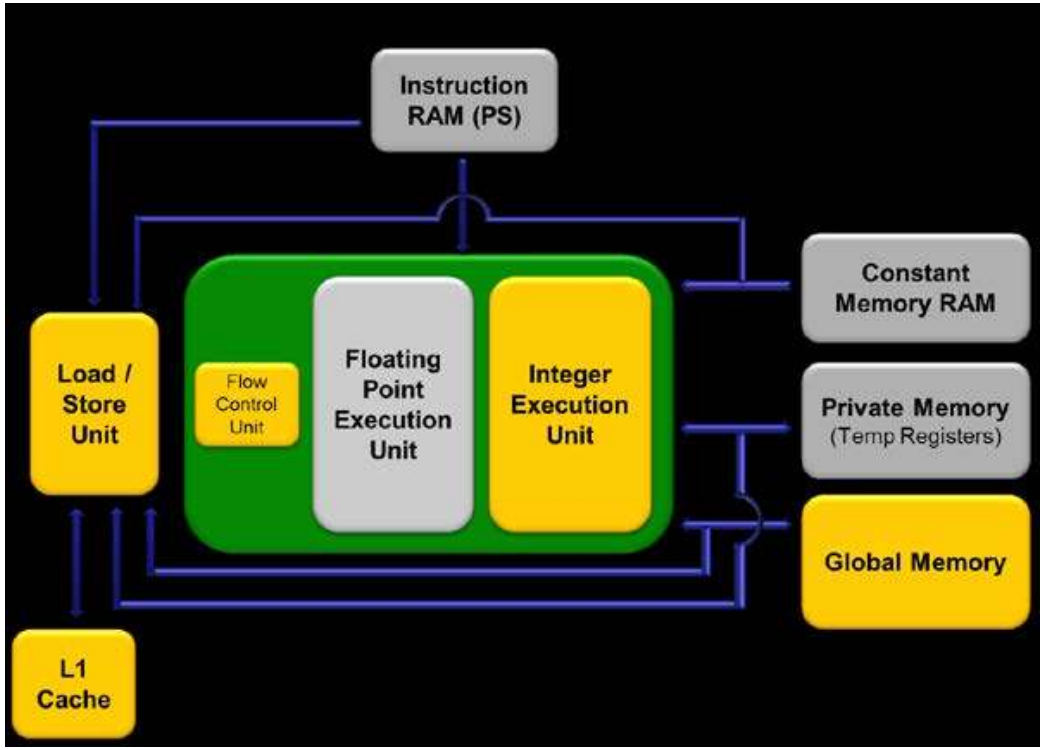
- native_cos
- native_divide
- native_exp
- native_exp2
- native_exp10
- native_log
- native_log2
- native_log10
- native_powr
- native_recip
- native_rsqrt
- native_sin
- native_sqrt
- native_tan

i.MX 6 GPU Compute Capabilities



- The front end passes instructions and constant data as State Loads to the OpenCL Compute Unit (Shader) block. State Loads program instructions and constant data and work groups initiate execution on the instructions and the constants loaded.
- Threads from a work-group will be grouped into internal “thread-groups.” All the threads in a thread-group execute in parallel. Barrier instruction is supported to enforce synchronization within a work-group.

i.MX 6 GPU Compute Capabilities



Agenda

- What is GPU Compute?
- i.MX 6 GPU Compute Capabilities
- How to use GPU Compute - OpenCL Introduction
- OpenCL SDK
- Future of GPU Compute
- Resources

How to use GPU Compute - OpenCL Introduction

- Open Computing Language (OpenCL) is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors.
- OpenCL includes a language (based on C99) for writing kernels (functions that execute on OpenCL devices), plus application programming interfaces (APIs) that are used to define and then control the platforms.
- OpenCL provides parallel computing using task-based and data-based parallelism.
- OpenCL is an open standard maintained by the non-profit technology consortium Khronos Group.

How to use GPU Compute - OpenCL Introduction

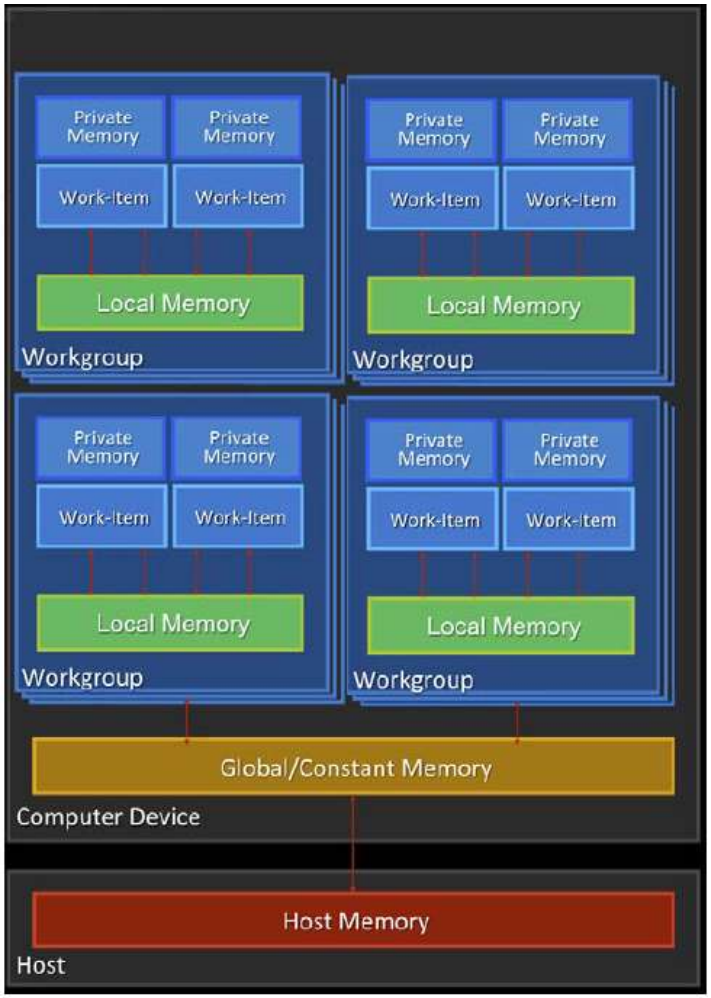
- Once a developer finds an appropriate compute device to execute a program on and creates a context, the OpenCL runtime API takes over and controls the rest of the execution.

- Users can then create kernels to run on the devices, create and control command queues that feed data to kernels, allocate memory at various levels (described below), define the scope of kernel execution and how it can be run, and define synchronization points between the kernels.

Kernel = essentially a ‘C’ function that does “work”

How to use GPU Compute - OpenCL Introduction

- When a kernel is submitted, an index space is defined.
- An instance of the kernel is executed for each index in the index space.
  (the kernel instance = **work item**)
- Each work item has an associated global ID.
- Work items are organized into **work-groups.** Each work group will have a unique local ID.
- The Index space supported in OpenCL is called an **NDRange.**



How to use GPU Compute - OpenCL Introduction

- OpenCL is run in a 'data parallel' programming model where the kernels are run once for each item in an 'index space'

- The dimensionality of the data being processed (e.g., 1, 2, or 3 dimension arrays; called NDRange or N-dimensional range).

- A array of data (e.g., a 2D array of pixels), the kernel is run once on each data element (e.g., pixel) in the array.

How to use GPU Compute - OpenCL Introduction

- To further exploit the parallelism of work groups, OpenCL defines multiple memory types that are restricted based on the kernel's scope:

OpenCL Memory Types	Definition	Mapping to GC2000 GPGPU in i.MX 6Q
Private Memory	Visible only to the individual work-item; not visible to any other work-item	Registers, System Memory*
Local Memory	Visible to all work-items within a work-group	System Memory*
Global Memory	Visible to all work-items (allows read, write, and map)	System Memory*
Constant Memory	Visible to all work-items (Read-only Global Memory)	Constant Registers, System Memory*

*System Memory is loaded to L1 cache and the GPGPU reads from the cache.

How to use GPU Compute - OpenCL Introduction

- Kernels are controlled via command queues, which contain three types of commands:
  - **Kernel Execution:** Dispatches a kernel to the correct processing element for execution.
  - **Memory Control:** Controls data flow or mapping between memory objects (including mapping objects to the host memory system).
  - **Synchronization:** Controls execution by adding blocks and synchronization signals.

How to use GPU Compute - OpenCL Introduction

Hardware Features	GC2000
Compute Units (Shader Cores)	4
Local Storage	1 KB
Instruction Memory	512
Texture Samplers	8
L1 Cache	4 KB
Temporary Registers	64
Maximum count of global work items each dimension	64K
Maximum count of work-items each dimension per work-group	1K
Maximum stack size for nested functions	4
Preferred work-group size	16

How to use GPU Compute - OpenCL Introduction

- Each of the shader cores function as a CU. The cores are a native Vec4 ISA, thus the preferred vector width for all primitives is four (4).

OpenCL Queries	GC2000
CL_DEVICE_MAX_COMPUTE_UNITS	4
CL_DEVICE_MAX_WORK_GROUP_SIZE	1024
CL_DEVICE_LOCAL_MEM_SIZE	1KB
CL_DEVICE_MAX_SAMPLERS	8
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT	4

How to use GPU Compute - OpenCL Introduction

OpenCL Queries	GC2000
CL_DEVICE_IMAGE2D_MAX_WIDTH/HEIGHT	8192
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS	3
CL_DEVICE_MAX_WORK_ITEM_SIZES[0/1/2]	1024
CL_DEVICE_MAX_PARAMETER_SIZE	256
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE	64
CL_DEVICE_MAX_READ_IMAGE_ARGS	8
CL_DEVICE_MAX_WRITE_IMAGE_ARGS	8

How to use GPU Compute - OpenCL Introduction

Architectural Optimizations:

- **Vector math inputs in multiples of 4.**
  - As mentioned previously, the GC2000 in i.MX 6Q is a Vec4 floating point SIMD engine, so vector math always prefers 4 inputs (or a multiple of 4) for maximum math throughput.

- **Use full 32 bit native registers for math.**
  - Both integer and floating point math is natively 32 bit. 8 and 16bit primitives will still use 32 bit registers, so there is no gain (for the math computation) in going with lower sizes.

How to use GPU Compute - OpenCL Introduction

Use floating point instead of integer formats

- 1x 32-bit integer pipeline (supports 32-bit INT formats in hardware, 8-bit/16-bit done in software)
- 4x 32-bit Floating Point (FP) pipeline (supports 16-bit and 32-bit FP formats in hardware)

- To maximize OpenCL compute efficiency, it is better to convert integer formats to floating point to utilize the four (4) parallel FP math units.

How to use GPU Compute - OpenCL Introduction

- **Keep global work size at 64K (maximum) per dimension**
  - Since global IDs for work items are 16 bits, it is necessary to keep the global work size within 64K (65,536 or 216) per dimension.

- **Branching should be avoided if possible**
  - Branch penalties depend on the percentage of work items that go down the branch paths.

How to use GPU Compute - OpenCL Introduction

```
__kernel void gaussian_filter   (       __global uchar *input,
                                        __global uchar *output,
                                        int width,
                                        int height
                                )
{
        int x = get_global_id (0);
        int y = get_global_id (1);
        int id = (y * width) + x;

        float sum = 0;

        float kernel_weights[3][3] =    {
                                                {1.0/16.0, 2.0/16.0, 1.0/16.0},
                                                {2.0/16.0, 4.0/16.0, 2.0/16.0},
                                                {1.0/16.0, 2.0/16.0, 1.0/16.0}
                                        };

        if ((x == 0) || (y == 0) || (x == (width -1)) || (y == (height -1)))
        {
                output[id] = input[id];
                return;
        }

        sum = input[id - width - 1] * kernel_weights[0][0];
        sum += input[id - width] * kernel_weights[0][1];
        sum += input[id - width + 1] * kernel_weights[0][2];

        sum += input[id - 1] * kernel_weights[1][0];
        sum += input[id] * kernel_weights[1][1];
        sum += input[id + 1] * kernel_weights[1][2];

        sum += input[id + width - 1] * kernel_weights[2][0];
        sum += input[id + width] * kernel_weights[2][1];
        sum += input[id + width + 1] * kernel_weights[2][2];

        output[id] = sum;

        return;
}
```

How to use GPU Compute - OpenCL Introduction

```
__kernel void dilate_op (__global uchar *input,
                         __global uchar *output,
                         int width,
                         int height
                )
{
        int i;
        int x = get_global_id (0);
        int y = get_global_id (1);
        int id = (y * width) + x;
        float max = 0;
        float values[5];

        if ((x == 0) || (y == 0) || (x == (width -1)) || (y == (height -1)))
        {
                output[id] = input[id];
                return;
        }

        values[0] = input[id - width];
        values[1] = input[id - 1];
        values[2] = input[id];
        values[3] = input[id + 1];
        values[4] = input[id + width];

        if (values[0] > max) max = values[0];
        if (values[1] > max) max = values[1];
        if (values[2] > max) max = values[2];
        if (values[3] > max) max = values[3];
        if (values[4] > max) max = values[4];

        output[id] = max;

        return;
}
```

Agenda

- What is GPU Compute?
- i.MX 6 GPU Compute Capabilities
- How to use GPU Compute - OpenCL Introduction
- OpenCL SDK
- Future of GPU Compute
- Resources

OpenCL SDK

i.MX6 Community

– https://community.freescale.com/community/imx

i.MX6 Graphics SDK

- http://freescale.com/imx6

Click on i.MX6D

Software & Tools tabs

Software Development Tools-> Snippets, Boot Code, Headers, Monitors, etc.-> IMX6_GPU_SDK

i.MX6 Graphics Vivante Tools

- http://freescale.com/imx6

Click on i.MX6D

Software & Tools tabs

Software Development Tools->IDE – Debug, Compile, and Build Tools -> IMX_6D_Q_VIVANTE_VDK_145_TOOLS

OpenCL Partner

Au-Zone Technologies Inc.

Au-Zone Technologies Inc.

GPU Compute on the i.MX6
Some Working Examples

Application Space

Embedded CL

- Intelligent IoT end points
- Advanced / Remote M2M sensors
- Embedded Signal Processing
- Multi-sensor fusion + analytics
- Acceleration of embedded CV

Embedded CV

- Machine Vision / Asset monitoring
  - QA, automation, equipment/process
- Healthcare Monitoring
  - Elder-care, patient care, staff hygiene
- Advanced Surveillance
- Building management
  - Occupancy, flow, HVAC
- Driver / Operator Assist
  - Consumer and commercial/industrial
- Marketing
  - Realtime audience analytics

Embedded CL Tips

- Reduce memory I/O as much as possible.
- Vivante offers direct mapping of textures through an OpenGL extension.
- An IPU buffer can be used for direct mapping of video frames into OpenGL.
- Limited to YUV source data, need to be aware GPU natively uses RGBA.
- No equivalent for output data. OpenCL should only output reduced data if possible.
- Vivante native math functions provide enough accuracy for image processing and are much faster.
- Use asynchronous mode to ensure the GPU is always busy processing kernels while the CPU deals with juggling a stream of input and output buffers.
- Local memory is very limited on embedded GPU (size and speed) and generally offers no benefit compared to global memory.
- In nearly all cases of image processing the Kernel compile option -cl-fast-relaxed-math will provide noticeable performance improvements with no visible degradation.

Object Tracking

Method / Technique

- Background Subtraction
- Tracks moving objects
- Blob extraction
- OpenVG overlay graphics

Implementation

- 2 Vivante Direct textures for IPU flip flop.
- Each pixel is tested for motion using native_abs(current_pixel – previous_pixel);
- If result is greater than our threshold set the binary motion map to 1 otherwise 0.
- Motion map is then segmented into regions and each region is processed through a workgroup to report if enough pixels are set to 1.
- We download to the host the results of the motion map pixel test and generate OpenVG rects for each group of connected segments.

Test Results

- 30fps at HD vs < 6fps w/o GPU

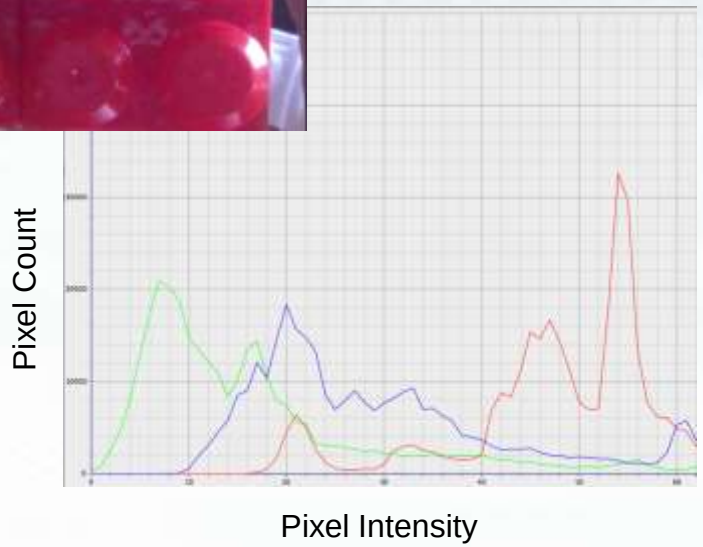
Lessons

- GPU was able to deal with increasing resolutions very well compared to CPU implementation.
- Solution could be further optimized by leveraging OpenCL-OpenGL interoperability to push blob rects directly to shader as VBO.

Colour Histogram

Method / Technique

- USB or CSI connected camera to capture real time video stream
- Creation of histogram overlay
- Output results HDMI and RTSP
- Histograms can be used for image enhancement.
- By equalizing the histogram the image will make better use of the available colourspace.
- Useful technique for dealing with low light conditions.



Implementation

```
/* each workgroup iterates local histogram segment.
   results are summed in a second pass. */ for (int x = origin.x; x <
origin.x + width; x++) {
        for (int y = origin.y; y < origin.y + height; y++) {
                vec3 colour = read_imagei(img, …, vec2(x, y)).rgb;
                local_histogram[colour.r]++;
                local_histogram[colour.g]++;
                local_histogram[colour.b]++;
        }
}
```

Lessons

- Lack of atomics requires second pass of the results on the host side. Kernel calculates local segment histogram and host sums the results.

More Information

http://www.au-zone.com

http://www.embeddedcv.com

http://www.embeddedcl.com

Agenda

- What is GPU Compute?
- i.MX 6 GPU Compute Capabilities
- How to use GPU Compute - OpenCL Introduction
- OpenCL SDK
- Future of GPU Compute
- Resources

Future of GPU Compute

OpenGL ES 3.1 – Compute Shaders – compute for the “masses”

OpenCL 2.0

- **Shared Virtual Memory**
  - Host and device kernels can directly share complex, pointer-containing data structures such as trees and linked lists, providing significant programming flexibility and eliminating costly data transfers between host and devices.

- **Dynamic Parallelism**
  - Device kernels can enqueue kernels to the same device with no host interaction, enabling flexible work scheduling paradigms and avoiding the need to transfer execution control and data between the device and host, often significantly offloading host processor bottlenecks.

- **Generic Address Space**
  - Functions can be written without specifying a named address space for arguments, especially useful for those arguments that are declared to be a pointer to a type, eliminating the need for multiple functions to be written for each named address space used in an application.

Future of GPU Compute

OpenCL 2.0

- **C11 Atomics**
  - A subset of C11 atomics and synchronization operations to enable assignments in one work item to be visible to other work items in a work group, across work groups executing on a device or for sharing data between the OpenCL device and host.

- **Pipes**
  - Pipes are memory objects that store data organized as a FIFO. OpenCL 2.0 provides built-in functions for kernels to read and write pipes, providing straight forward programming that can be highly optimized by implementers.

Agenda

- What is GPU Compute?
- i.MX 6 GPU Compute Capabilities
- How to use GPU Compute - OpenCL Introduction
- OpenCL SDK
- Future of GPU Compute
- Resources

Resources

i.MX6 Community

- https://community.freescale.com/community/imx

Khronos OpenCL Resources

OpenCL Specification

- http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf

OpenCL Registry

- http://www.khronos.org/registry/cl/

OpenCL Developer Forums

- http://www.khronos.org/message_boards/

OpenCL Quick Reference Card

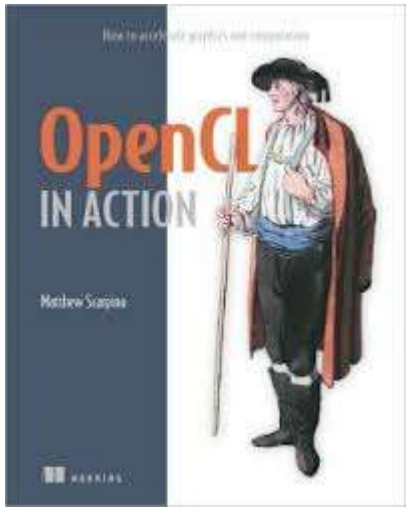
- http://www.khronos.org/files/opencl-1-1-quick-reference-card.pdf

OpenCL Online Man pages

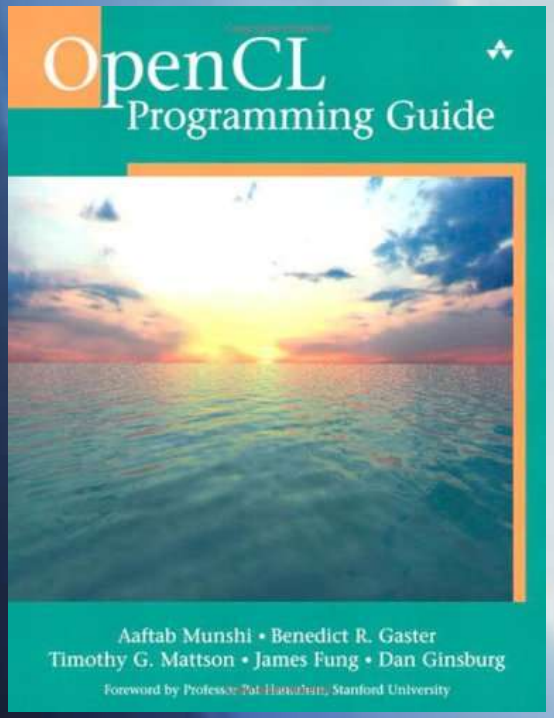
- http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/

OpenCL In Action

- By Matthew Scarpino
- ISBN-10: 1617290173 | ISBN-13: 978-1617290176
- Publishing Date: November 2011
- Manning Publications http://www.manning.com/scarpino2/
- Amazon http://www.amazon.com/OpenCL-Action-Accelerate-Graphics-Computations/dp/1617290173



Now Available...



OpenCL Programming Guide

By Aaftab Munshi, Benedict Gaster, Timothy G. Mattson, James Fung, Dan Ginsburg

ISBN-10: 0321749642 | ISBN-13: 978-0321749642
Publishing Date: July 2011

Addison-Wesley Professional
http://www.informit.com/title/0321749642

Available in Print, major eBook formats, and Safari Books Online.

ORDER NOW from InformIT or your local book store or online reseller.

Discount code: OPENCL35

www.Freescale.com