OpenCL
A Super(ficial) Introduction

Radu Velea

Intel Romania

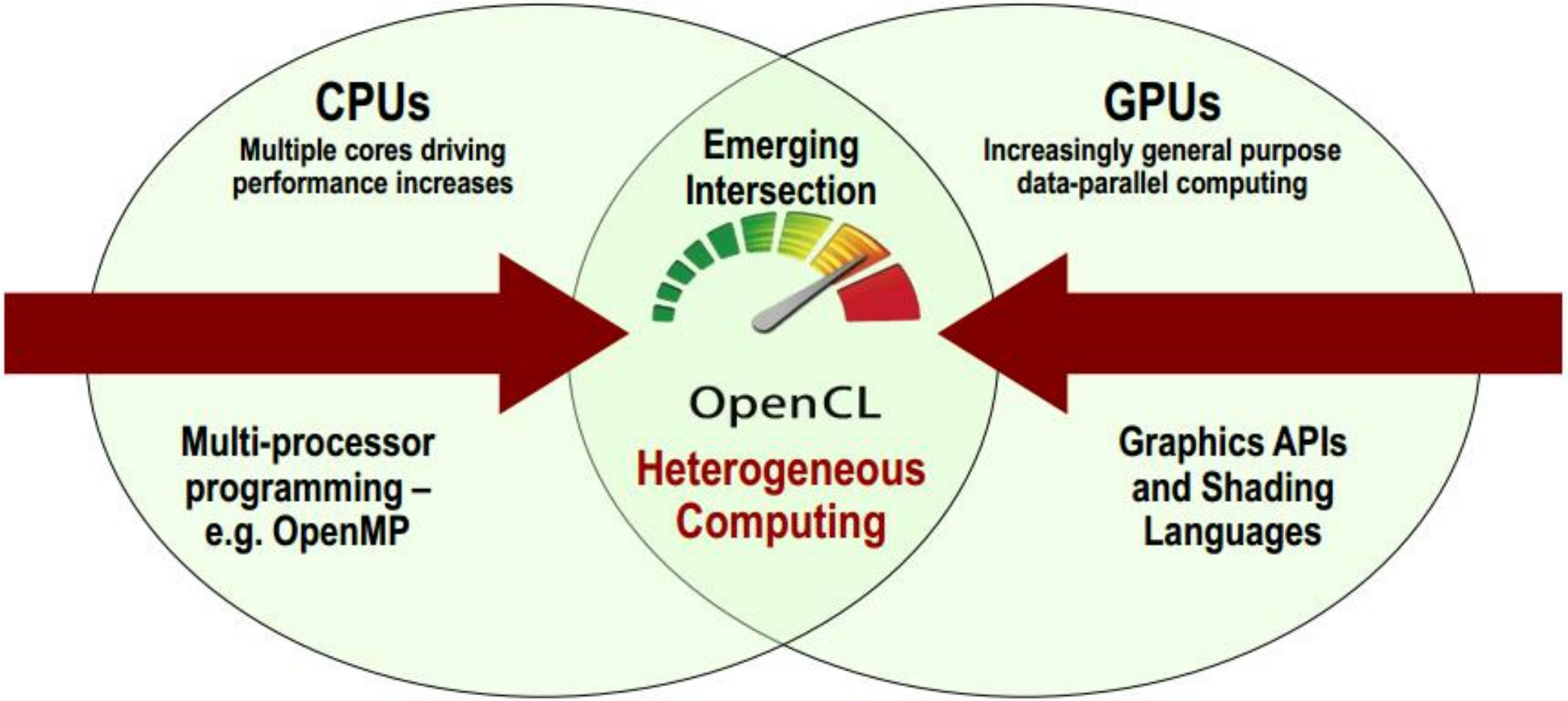
Preliminaries – Disclosures

- The views expressed in this tutorial are those of the people delivering the tutorial.
- We are not speaking for our employers.
- We are not speaking for Khronos.

Agenda

- **Introduction**
- Importance
- Current Support State
- Building OpenCL
- Running OpenCL
- Debugging OpenCL
- OpenCL Performance
- A Look Into the Future

Motivation


CPUs
Multiple cores driving performance increases
Emerging Intersection
GPUs
Increasingly general purpose data-parallel computing
OpenCL
Heterogeneous Computing
Multi-processor programming – e.g. OpenMP
Graphics APIs and Shading Languages

History

- Thus in 2008 OpenCL appeared as the “brainchild” of several tech corporations
- It is developed by the Khronos Group*
- Releases:
  - OpenCL 1.0 – 2008
  - OpenCL 1.1 – 2010
  - OpenCL 1.2 – 2011
  - OpenCL 2.0 – 2013

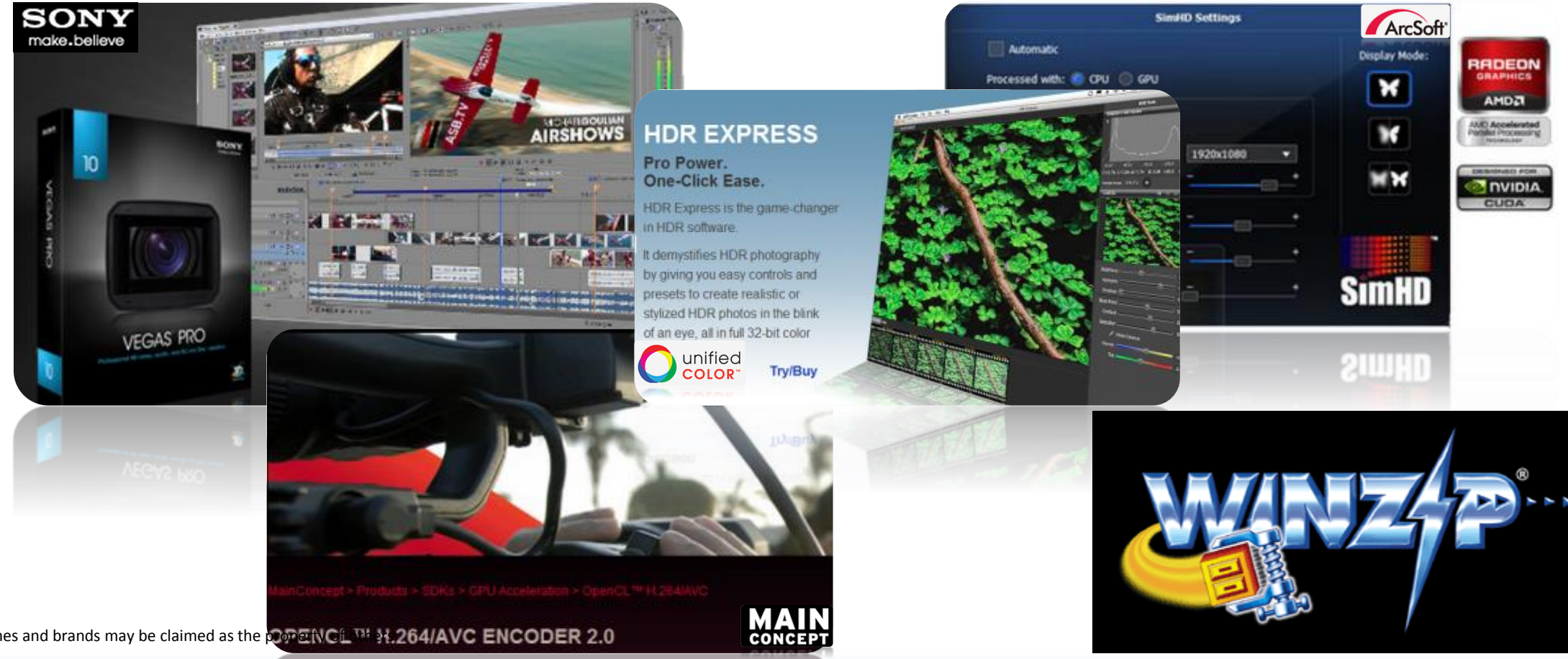
Agenda

- Introduction
- **Importance**
- Current Support State
- Building OpenCL
- Running OpenCL
- Debugging OpenCL
- OpenCL Performance
- A Look Into the Future

Enabling Heterogeneous Computing

- Advantages of using OpenCL
  - It’s free!
  - Portable language
  - Continuous development
  - Works with OpenGL
  - Can be used on embedded platforms as well as in Web applications
  - Both open source and proprietary implementations
- Disadvantages
  - Not as mature as other technologies
  - Hard to get top performance on all platforms

Usages – Client – Video/Image Encoding



*Other names and brands may be claimed as the property of others.

Usages – Computer Aided Engineering

*Other names and brands may be claimed as the property of others

Usages – HPC

- Medical imaging
- Weather forecasting
- Molecular modeling
- Cryptanalysis
- Bitcoin mining

Agenda

- Introduction
- Importance
- **Current Support State**
- Building OpenCL
- Running OpenCL
- Debugging OpenCL
- OpenCL Performance
- A Look Into the Future

Operating Systems

- OpenCL is supported by most modern operating systems
  - Microsoft Windows
  - Linux
  - Mac OS/OS X
  - Chrome OS
  - Tizen
  - Near Future: Android

Hardware

- All major hardware vendors support OpenCL
  - Intel
  - AMD/ATI
  - NVIDIA
  - QUALCOMM
  - ARM
  - Texas Instruments
  - Imagination Technologies
  - IBM

Virtualization and Other Software

- OpenCL can run in virtual and distributed environments, making it an important asset for GPGPU computing in the cloud
- Virtual solutions that can run OpenCL are supported by VMware and MOSIX (Virtual OpenCL)

- OpenCL support can be compiled into other frameworks such as OpenCV or Intel Integrated Performance Primitives (IPP)

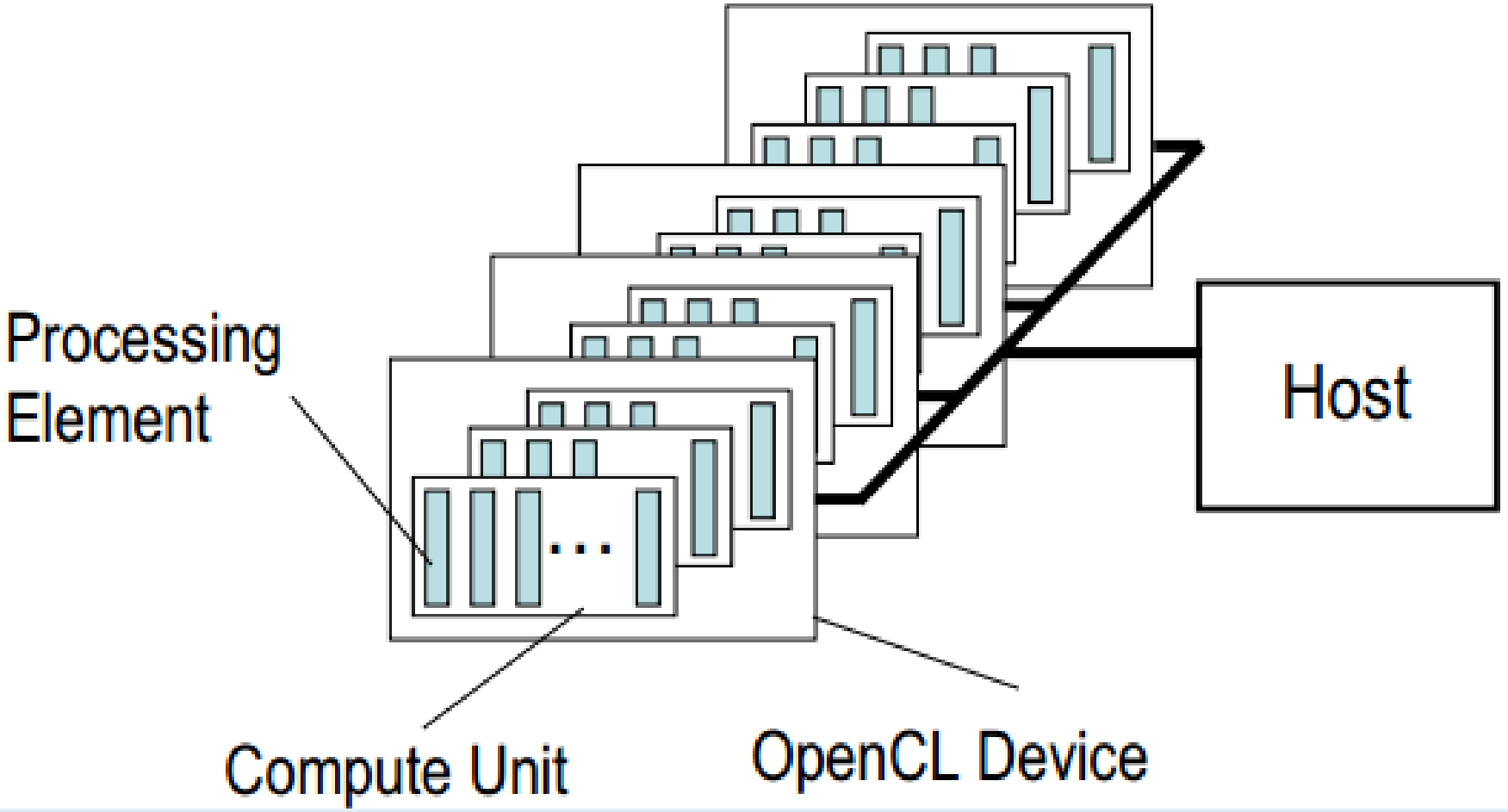
Agenda

- Introduction
- Importance
- Current Support State
- **Building OpenCL**
- Running OpenCL
- Debugging OpenCL
- OpenCL Performance
- A Look Into the Future

OpenCL Platform Model

- An OpenCL application is composed of **Host** and **Device** code
- Host code sets-up the OpenCL environment and selects the hardware on which the application will run as well as its parameters – extensions, optimizations, synchronization, etc.
- Host code support is implemented in C, but has bindings in other languages such as C++, Java, C#, Python
- Device code or kernel code runs on the computing end of the machine and it’s very similar to a C99 function

OpenCL Platform Abstraction



Building an OpenCL “Hello World”

```
const char *msg = "Hello World!";
// Get Platform Object
clGetPlatformIDs(1, &platform, NULL);
// Get Device Object
clGetDeviceIDs(platform, CL_DEVICE_TYPE_CPU, 1, &device, NULL);
// Create OpenCL Context
context = clCreateContext(NULL, 1, &device, NULL, NULL, NULL);
// Create OpenCL Program
program = clCreateProgram(context, 1, &kernel_code, NULL, NULL);
// Build Program
clBuildProgram(program, 1, &device, NULL, NULL, NULL);
// Get Kernel Reference
kernel = clCreateKernel(program, "hello_world", NULL);
// Create Command Queue
queue = clCreateCommandQueue(context, device, 0, NULL);
// Create Memory Object
memobj = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
strlen(msg) + 1, msg, NULL);
// Set Kernel Argument
clSetKernelArg(kernel, 0, sizeof(memobj), (void*)&memobj);
// Schedule the Kernel for Execution
clEnqueueNDRangeKernel(queue, kernel, 1, NULL, {4, 0, 0}, NULL, 0, NULL,
NULL);
__kernel void hello_world(__global char *msg) {
      int global_id = get_global_id(0);
      printf("%s from %d!\n", msg, global_id);
}
```

Platforms and Devices

- Platform is a synonym for vendor OpenCL implementation. You can have multiple corresponding to a single hardware device (ex: multiple OpenCL drivers installed)
- Each platform has 1 or more devices. Devices are handles for the underlying hardware (CPU, GPU, etc.)
- Device type is important when compiling OpenCL kernel code
- Programs can be created from source or loaded as precompiled binary. They can contain 1 or more kernels and auxiliary functions

Context

- Contexts are used to define the kernel execution environment:
  - Memory management
  - OpenCL object scope (ex: events, kernel objects, queues, programs, etc.)
- Contexts can be made to include one or more devices (especially useful for hybrid implementations)
- With OpenCL 1.2, contexts can partition devices using **clCreateSubDevices**
- For managing objects, OpenCL uses a reference counting model with **clRetain…** and **clRelease…** functions

Command Queues

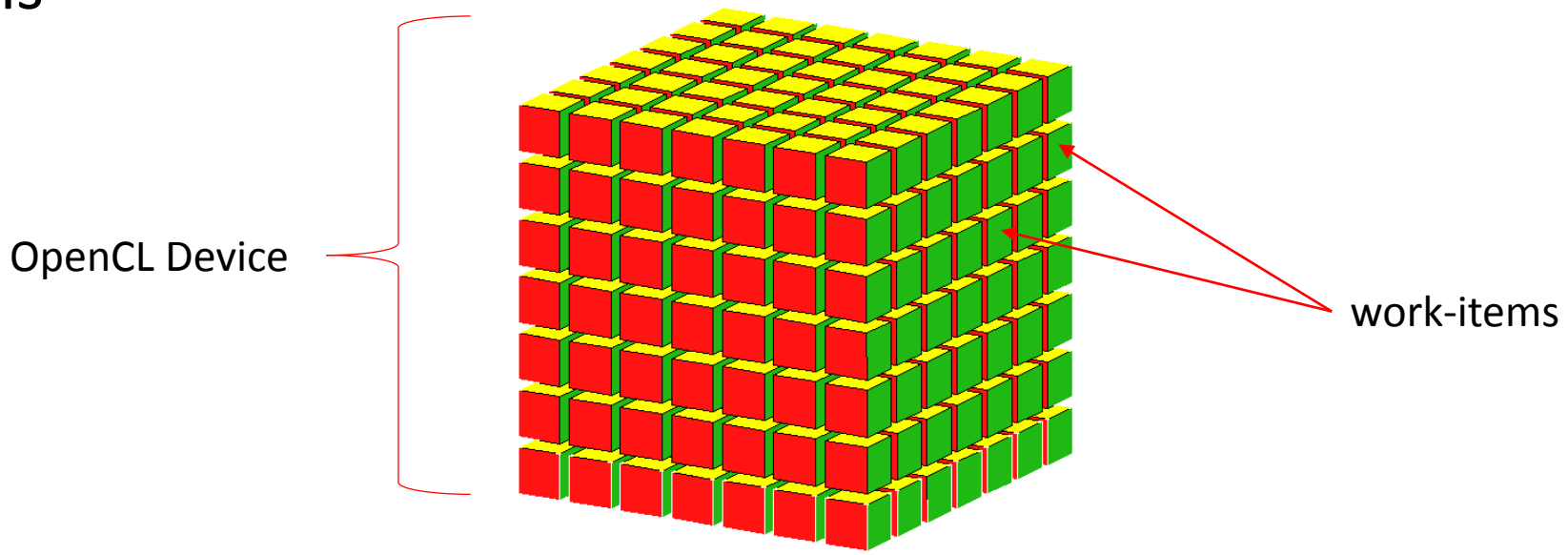
- Queues are used to submit commands to the device:
  - Memory operations
  - Kernel execution
  - Synchronization: waiting for events, barriers
- They support in-order and out-of order execution
- Each command queue points to a single device within a context, but a context can have many command queues and a device can be used by multiple queues
- Command queues are vital when it comes to synchronization
- Command queue functions will start with **clEnqueue…**

Agenda

- Introduction
- Importance
- Current Support State
- Building OpenCL
- **Running OpenCL**
- Debugging OpenCL
- OpenCL Performance
- A Look Into the Future

Work-Items

- Work-items are an abstraction used for device compute-cores
- The kernels is a piece of code designed to run in parallel on a group of work-items



clEnqueueNDRangeKernel

- cl_command_queue **command_queue**,
- cl_kernel **kernel**,
- cl_uint **work_dim**,
- const size_t ***global_work_offset**,
- const size_t ***global_work_size**,
- const size_t ***local_work_size**,
- cl_uint **num_events_in_wait_list**,
- const cl_event ***event_wait_list**,
- cl_event ***event**

Kernel Execution

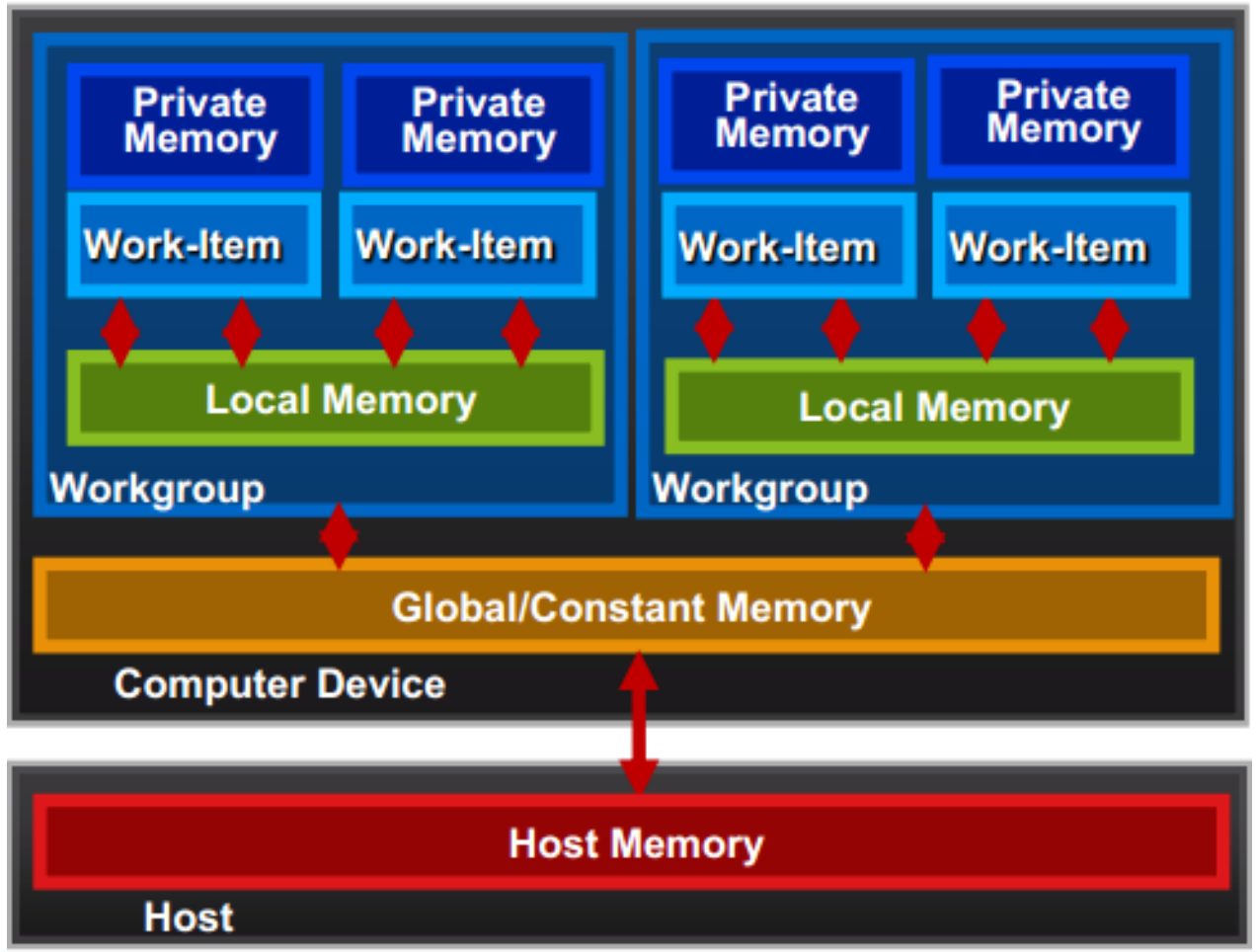
- Kernel functions are designed to replace and parallelize loops in serial programs
- The programmer can set the dimension of the problem when he/she queues the kernel for execution
- Besides the original layout (**global work size**), kernels can be grouped in smaller units called **local work groups**
- Kernel functions have some special API not part of standard C:
  - get_global_id, get_local_id, get_global_size, get_work_dim, get_local_size, get_num_groups, get_group_id

OpenCL C99

- The following section lists most of the differences between C standard and __kernel functions
  - Recursion is not supported
  - Built-in data types: vectors, half floats
  - Built-in mathematical and image processing functions
  - Built-in kernels
  - Cannot dynamically allocate memory inside a kernel – no malloc/free (has to be done from host code)
  - Synchronization functions
  - No function pointers
  - No bit fields and variable size arrays/structures

Kernel Memory Model

- Relaxed consistency
- Memory transfers can take up most of the time



Optimal Execution Model

- A big challenge for anyone writing portable applications for OpenCL is getting the maximum performance on all platforms
- Things to take into consideration when writing portable code:
  - Problem size vs. global/local work sizes
  - Memory transfers between host and device:
    - Bandwidth
    - Unified memory access
  - Platform support – using CPU instead of GPU
  - Device extensions: dx/gl sharing, kernel atomic operations
- Useful information can be obtained using **clGetDeviceInfo**

Agenda

- Introduction
- Importance
- Current Support State
- Building OpenCL
- Running OpenCL
- **Debugging OpenCL**
- OpenCL Performance
- A Look Into the Future

Host Code

- Host code can be debugged using regular debuggers
- User can get program build information using **clGetProgramBuildInfo**
- Popular IDEs have plug-ins to support OpenCL developers:
  - Visual Studio
  - Eclipse
  - NetBeans

Printf

- “The best debugging tools are your intuition and printf” (where supported)
- Printf output is flushed after the kernel invocation – there is no synchronization with respect to work-item order
- Differences from normal printf:
  - Vector printing
  - Does not return the number of characters printed (returns 0 on success)
  - “%s” can only be used for literal strings

OpenCL Debuggers

- gDEBugger/AMD CodeXL
  - Real time OpenCL and OpenGL API level debugging
  - Static kernel analysis
    - Build messages
  - Online OpenCL kernel debugging
    - Object vizualization
  - Works on both Linux and Windows
  - GPU profiling
- Intel® SDK for OpenCL Applications - Kernel Builder
  - Syntax checker
  - Cross hardware compilation support
  - Assembly language view
  - Can run kernel code on a device without user having to write host code

Agenda

- Introduction
- Importance
- Current Support State
- Building OpenCL
- Running OpenCL
- Debugging OpenCL
- **OpenCL Performance**
- A Look Into the Future

OpenCL Profiling

- Users can attach events to queued commands
- Command queues have be created with CL_QUEUE_PROFILING_ENABLE flag set
- These events can later be interrogated via **clGetEventProfilingInfo**:
  - Queue time (CL_PROFILING_COMMAND_QUEUED)
  - Submit time (CL_PROFILING_COMMAND_SUBMIT)
  - Start (CL_PROFILING_COMMAND_START)
  - Stop (CL_PROFILING_COMMAND_END)
- Devices running OpenCL programs can be profiled using profiling software such as VTune

OpenCL vs. CUDA

- OpenCL is more recent than CUDA but already has support from all major vendors
- CUDA only works on Nvidia
- OpenCL performance may vary across vendor implementations, whereas CUDA tends to be more uniform
- CUDA benefits from better debugging and profiling tools
- Performance comparisons between the 2 APIs usually comes down to hardware and driver performance rather then implementation (there is no neutral ground)

Agenda

- Introduction
- Importance
- Current Support State
- Building OpenCL
- Running OpenCL
- Debugging OpenCL
- OpenCL Performance
- **A Look Into the Future**

OpenCL 2.0

- In late 2013 OpenCL 2.0 appeared and brought a new dimension to heterogeneous computing
- Important features include:
  - Shared virtual memory
  - Nested parallelism
  - Generic address space
  - Improved image support
  - C11 atomics for inter-work-item synchronization
  - Android support – enables OpenCL libraries to be discovered and loaded in Android
  - Pipes for kernels
  - Industry support – growing enthusiasm and adoption rate based on passed successes

Vision

- OpenCL drives progress in areas such as Perceptual Computing and Augmented Reality
- Mobile support can enable faster data processing algorithms to be implemented at a reduced cost
- OpenCL has been included in Cloud technologies by Amazon and Adobe and this tendency is likely to grows
- Hybrid implementations can tackle a wider range of problems, surpassing GPGPUs and CPU-offload programs in performance and efficiency

Conclusions

- The “Age of OpenCL” is just beginning
- OpenCL has the potential to bring a new dimension in the field of computing as s whole rather than just HPC or GPGPU
- Samples and documentation is freely available making the initial learning curve less steep
- Continuous development and integration with other APIs will ensure even greater success in the future

References

- https://software.intel.com/en-us/articles/tutorial-opencl-introduction-for-hpc-programmers
- http://developer.amd.com/tools-and-sdks/opencl-zone/opencl-tools-sdks/introductory-tutorial-to-opencl/
- http://www.winzip.com/win/en/index.htm
- http://www.winzip.com/win/en/cl-features.html
- http://www.arcsoft.com/Corporate/press-news-li/615/0/2011/ArcSoft-Optimizes-Several-of-its-Best-Selling-Applications-for-New-AMD-A-Series-APU
- http://www.sonycreativesoftware.com/news/newsletter?eid=599
- http://unifiedcolor.com/content/unified-color-technologies-launches-hdr-expose-industry-leading-technology-provides-photogra
- http://www.mainconcept.com/eu/products/sdks/gpu-acceleration-sdk/opencltm-h264avc.html
- http://www.indigorenderer.com/indigo3
- http://support.apple.com/kb/ht4664

References

- http://gpumodeling.blogspot.com/2010/04/eyeon-software-unveils-fusion-61-with.html
- http://wiki.blender.org/index.php/OpenCL
- http://www.khronos.org/conformance/adopters/conformant-products#opencl
- https://developer.tizen.org/sites/default/files/page/tizen-opencl-add-on-sdk-developer-guide-v2.2.1_0.pdf
- http://blogs.vmware.com/performance/2011/10/gpgpu-computing-in-a-vm.html
- http://www.mosix.cs.huji.ac.il/txt_vcl.html
- http://ken-soft.com/2009/06/27/3d-cube-engine-java/
- http://www.khronos.org/registry/cl/sdk/1.0/docs/man/xhtml/workItemFunctions.html
- http://www.cmsoft.com.br/index.php?option=com_content&view=category&layout=blog&id=85&Itemid=136
- http://developer.amd.com/tools-and-sdks/opencl-zone/opencl-tools-sdks/codexl/

References

- https://software.intel.com/en-us/articles/intel-sdk-for-opencl-applications-performance-debugging-intro
- https://software.intel.com/en-us/intel-vtune-amplifier-xe
- http://streamcomputing.eu/blog/2011-06-22/opencl-vs-cuda-misconceptions/
- http://scicomp.stackexchange.com/questions/10354/cuda-vs-opencl-as-of-late-2013
- https://www.khronos.org/news/press/khronos-finalizes-opencl-2.0-specification-for-heterogeneous-computing
- http://iwocl.org/2014/04/amd-opencl-acceleration-for-adobe-apps/
- http://parallelis.com/2014-will-be-the-opencl-year/
- http://www.altera.com/products/software/opencl/opencl-index.html

Notes

Legal Notice

- Intel and the Intel logo, are trademarks of Intel Corporation in the U.S. and/or other countries.

- THE MATERIALS ARE PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDED BUT NOT LIMITED TO THE WARRANTIES OF MECHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN ACTION OF CONTRACT, TORT, OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE MATERIALS OR THE USE OR OTHER DEALINGS IN THE MATERIALS

- Licensed under Creative Commons - Attribution-NonCommercial 4.0 International http://creativecommons.org/licenses/by-nc/4.0/