C-DAC Four Days Technology Workshop

*ON*

**Hybrid Computing – Coprocessors/Accelerators Power-Aware Computing – Performance of Applications Kernels**

**hyPACK-2013**
**(Mode-4 : GPUs)**

# Lecture Topic: An Overview of OpenCL on Xeon Phi

*Venue : CMSD, UoHYD ; Date : October 15-18, 2013*

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## OpenCL tries to Standardize Parallel Programming

**Background & Challenging Objectives :**

- ❖ OpenGL: Open Graphics Library
  - ➢ Widely supported application programming interface (API) for graphics ONLY
- ❖ OpenCL: "CL" Stands for Computing Language
  - ➢ providing an API library
  - ➢ Modifies C and C++ parallel programming
  - ➢ New Initiatives for other programming languages(Fortran)

**Aim:** to standardize general purpose parallel programming for any application

**Source :** Intel, NVIDIA, Khronos AMD, References

## The OpenCL Standard

❖ **Challenging Objectives :**

- ➢ OpenCL C is a restricted version of the C99 language with extension appropriate for executing data-parallel code on a variety of heterogeneous devices.

- ➢ Aimed for full support for the IEEE 754 formats

- ➢ Programming language, well suited to the capabilities of current heterogeneous platforms

**Source :** Intel, NVIDIA, Khronos AMD, References

## The OpenCL Standard

❖ **Challenging Objectives :**

➢ The model set forth by OpenCL creates portable, vendor- and device-independent programs that are capable of being accelerated on many different platforms.

- The OpenCL API is C wit h a C++ Wrapper API that is defined in terms of the C-API.
- There are third-party bindings for many languages, including Java, Python, and .NET
- The code that executes on an OpenCL device, which in general is not the same device as the host-CPU, is written in the OpenCL C language.

**Source :** Intel, NVIDIA, Khronos AMD, References

## OpenCL Design Requirements

- **Use all computational resources in system**
  - Program GPUs, CPUs and other processors as peers
  - Support both data- and task- parallel compute models
- **Efficient c-based parallel programming model**
  - Abstract the specified of underlying hardware
- **Abstraction is low-level, high-performance but device-portable**
  - Approachable –but primarily targeted at expert developers
  - Ecosystem foundation – no middleware or "convenience" functions
- **Implementation on a range of embedded, desktop, and server systems**
  - HPC desktop, and handheld profiles in on specification
- **Drive future hardware requirements**
  - Floating point precision requirements
  - Application to both consumer and HPC applications

**Source :** Intel, NVIDIA, Khronos AMD, References

## OpenCL Design Requirements

- ❖ **Efficient c-based parallel programming model**
  - ➢ Abstract the specified of underlying hardware
- ❖ **Abstraction is low-level, high-performance but device-portable**
  - ➢ Approachable –but primarily targeted at expert developers
  - ➢ Ecosystem foundation – no middleware or “convenience” functions

**Source :** Intel, NVIDIA, Khronos AMD, References

## Conceptual Foundations of OpenCL

**An Application for a heterogeneous platform must carry out the following steps.**

- ❖ Discover the completion that make-up the heterogeneous system
- ❖ Probe the characteristics of these components, so that the software can adapt to specific features of different hardware elements
- ❖ Create the blocks of instructions (Kernels) that will run on the platform

**Source :** Intel, NVIDIA, Khronos AMD, References

## Conceptual Foundations of OpenCL

**An Application for a heterogeneous platform must carry out the following steps.**

❖ Set up and manipulate memory objects involved in the computation.

❖ Execute the kernels in the right order and on the right components of the system

❖ Collect the final results

- Above steps are accomplished through a series of APIs inside OpenCL plus a programming environment for the kernels

**Source :** Intel, NVIDIA, Khronos AMD, References

## Conceptual Foundations of OpenCL

**An Application for a heterogeneous platform must carry out the following steps.**

- ❖ Discover the completion that make-up the heterogeneous system
- ❖ Probe the characteristics of these components, so that the software can adapt to specific features of different hardware elements
- ❖ Create the blocks of instructions (Kernels) that will run on the platform

**Source :** Intel, NVIDIA, Khronos AMD, References

## Conceptual Foundations of OpenCL

**An Application for a heterogeneous platform must carry out the following steps.**

- ❖ Set up and manipulate memory objects involved in the computation.
- ❖ Execute the kernels in the right order and on the right components of the system
- ❖ Collect the final results
  - Above steps are accomplished through a series of APIs inside OpenCL plus a programming environment for the kernels

**Source :** Intel, NVIDIA, Khronos AMD, References

## The OpenCL Specification – Models

❖ The OpenCL specification is defined in four parts, called models, that can be summarized as follows.

- Platform Model
- Execution Model
- Memory Model
- Programming Model

**Source :** NVIDIA, Khronos AMD, References

## The OpenCL Specification – Models

❖ **OpenCL Software Stack**

- **Platform Layer**
  - ➢ Query and select computer devices in the system
  - ➢ Initialize a compute device(s)
  - ➢ Create compute contexts and work-queues
- **Runtime**
  - ➢ Resource management
  - ➢ Execute compute kernels
- **Compiler**
  - ➢ A subset of ISO C99 with appropriate language additions
  - ➢ Compile and build compute program executable
  - ➢ Online or offline

**Source :** Intel, NVIDIA, Khronos AMD, References

## The OpenCL Specification – Models

❖ The OpenCL specification is defined in four parts, called models, that can be summarized as follows.

➢ Platform Model

- High Level description of the heterogeneous system

➢ Execution Model

- An abstract representation of how stream of instructions execute on the heterogeneous system

**Source :** Intel, NVIDIA, Khronos AMD, References

## The OpenCL Specification – Models

- The OpenCL specification is defined in four parts, called models, that can be summarized as follows.
  - Memory Models
    - The Collection of memory regions within OpenCL and how they interact during at OpenCL computation
  - Programming Model
    - The high-level abstractions a programmer uses when designing algorithms to implement an application

**Source :** Intel, VIDIA, Khronos AMD, References

## The OpenCL Specification

❖ **Platform model :**

- Specifies that there is one processor coordinating the execution (***the host***) and one or more processors capable of executing OpenCL C Code (***the devices***).
- It defines an abstract hardware model that is used by programmers when writing OpenCL functions (Called ***Kernels***) that execute on the devices.
- The platform model defines the relation between the host an device.
  - i.e., OpenCL implementation executing on a host x86 GPU, which is using a GPU device as an accelerator

**Source :** Intel, NVIDIA, Khronos AMD, References

## The OpenCL Specification

❖ **Platform model :**

- Platforms can be thought of a vendor – specific implementations of the OpenCL API.
- The platform model also presents an abstract device architecture that programmers target writing OpenCL C code.
- Vendors map this abstraction architecture to the physical hardware.

**Source :** Intel, NVIDIA, Khronos AMD, References

## OpenCL PLATFROM AND DEVICES

**Host-Device Interaction**

❖ Platform Model

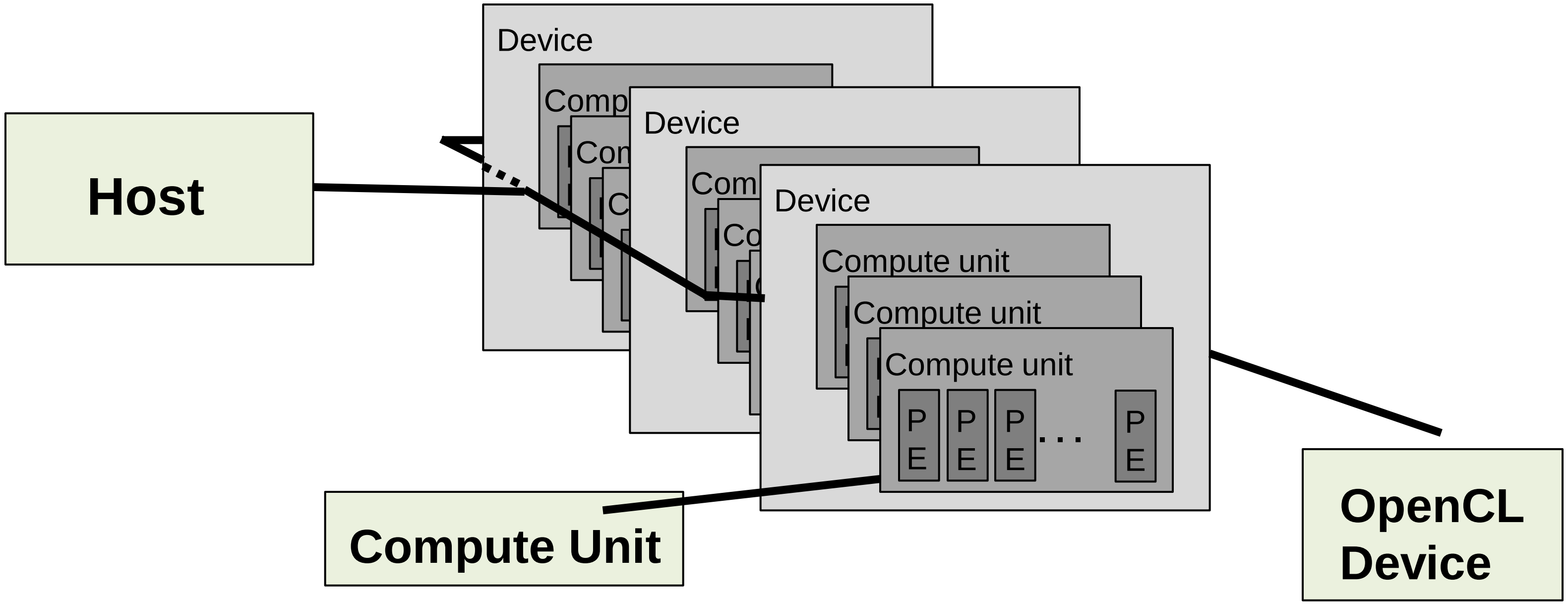
- Provides an abstract hardware model for devices
- Present an abstract device architecture that programmers target when writing OpenCL C code.
- Vendor-specific implementation of the OpenCL API.

❖ Platform Model

- Defines a device as an array of compute units
  - Compute units are further divided into processing elements
  - OpenCL device schedule execution of instructions.

**Source :** Intel, NVIDIA, Khronos AMD, References

## OpenCL Platform Model

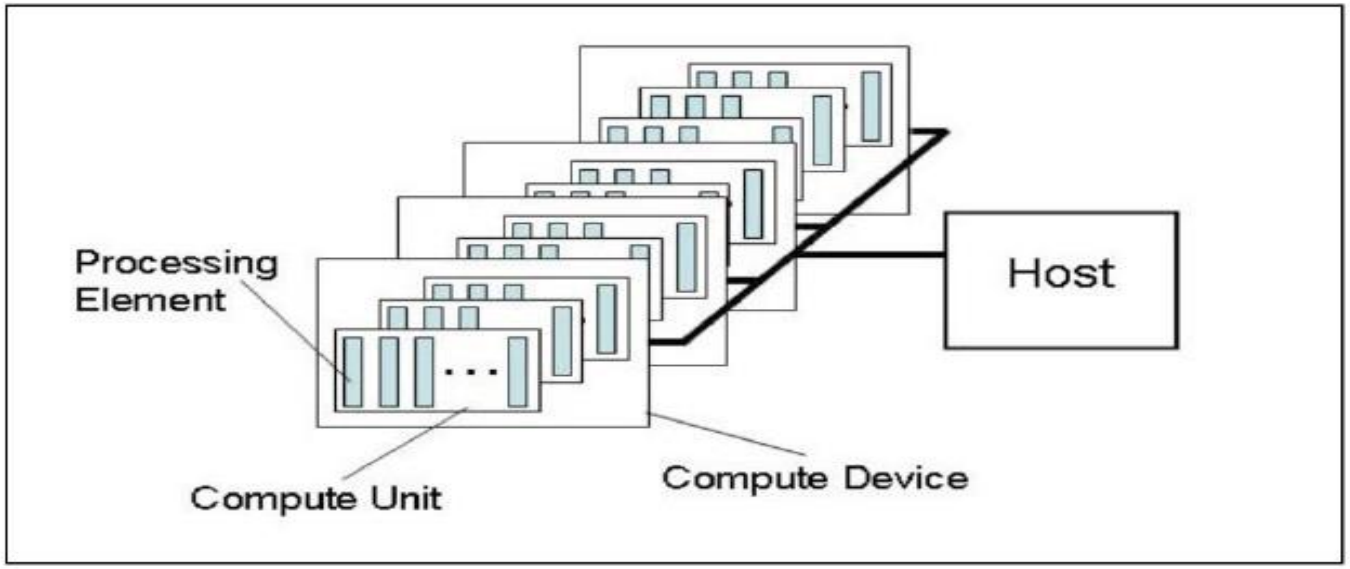


**The platform model defines an abstract architecture for devices.**

- The host is connected to one or more devices
- Device is where the stream of instructions (or kernels) execute (an OpenCL device is often referred to as a **compute device**
- A device can be a CPU, GPU, DSP, or any other processor provided by Hardware and supported by the OpenCL Vendor

**Source :** Intel, NVIDIA, Khronos AMD, References

## OpenCL Platform Model



- ❖ One Host + one or more compute Devices
  - ➢ Each compute Device is connected to one or more Compute Units.
    - Each compute Unit is further divided into one or more Processing Elements

**Source :** Intel, NVIDIA, Khronos AMD, References

## OpenCL PLATFROM Model

**How to discover available platforms for a given system ?**

```
cl_int
ClGetPlatformIds(cl_unit num_entries,
                 cl_platform_Id *platforms,
                 cl_unit *num_platforms)
```

- ❖ Platform Model
  - Defines  a device as an array of compute units
    - Compute units are further divided into processing elements
    - OpenCL device schedule execution of instructions.

**Source :** NVIDIA,  Khronos   AMD, References

## OpenCL PLATFORM Model

**How to discover available platforms for a given system.**

❖ Application calls **`ClGetPlatformIds() twice`**

- The **first** call passes an **unsigned int** pointer as the **`num_platforms`** argument and NULL is passes as the **platform** argument.
  - The programmer can then allocate space to hold the platform information.
- The **second** call, a **`cl_platform_id`** pointer is passed to the implementation with enough space allocated for **`num_entries`** platforms.

**Source :** NVIDIA, Khronos AMD, References

## OpenCL PLATFROM AND DEVICES

**After platforms have been discovered, How to determine which implementation (vendor) the platform was defined by ?**

The **`ClGetPlatformInfo()`** call determines implementation

The **`clGetDeviceIDs()`** call works very similar to **`ClGetPlatformId()`**

**How to use `device_type` argument ?**

| | | |
|---|---|---|
| GPUs | : | **`cl_DEVICE_TYPE_GPU`** |
| CPUs | : | **`cl_DEVICE_TYPE_CPU`** |
| All devices | : | **`cl_DEVICE_TYPE_ALL`** & other options |

**`Cl_GetDeviceinfo()`** is called to retrieve information such as name, type, and vendor from each device.

**Source :** Inttel, NVIDIA, Khronos AMD, References

## OpenCL PLATFROM Model

**After platforms have been discovered, How to determine which implementation (vendor) the platform was defined by ?**

The **`clGetDeviceIDs()`**

```
cl_int
clGetDeviceIDs(cl_platform_id platform,
                  cl_DEVICE_TYPE_GPU device_type,
                  cl_unit num_entries,
                  cl_device_id *devices,
                  cl_uint *num_devices)
```

**Source :** Intel, NVIDIA, Khronos AMD, References

# OpenCL PLATFORM Model

**How to get printed information about the OpenCL, supported platforms and devices in a system ?**

**`CLinfo` prorgam in the AMD APP SDK**

Uses **`clGetplatforminfo()`** and **`clGetDeviceInfo()`**

Hardware details such as memory size and bas widths are available using the commands

$ **`./CLinfo`** program gives complete information

**Source :** Intel, NVIDIA, Khronos AMD, References

## OpenCL PLATFROM AND DEVICES

```
$ ./CLinfo
```

```
Number of platforms :     1
Platform Profiles :       FULL_PROFILE
Platform Version :        OpenCL 1.1 AMD SDK –v2.4
Platform Name :           AMD Accelerated Parallel Processing
Platform Vendor :         Advanced Micro  Devices, Inc.
Number of Devices :       2
Device Type :             CL_DEVICE_TYPE_GPU
Name :                    Cypress
Max Compute Units :       20
Address bits              32
```

## OpenCL PLATFROM AND DEVICES

```
$ ./CLinfo
```

```
Max Memory Allocation:      268435456
Global Memory size :        1073741824
Constant buffer size :      65536
Local Memory type :         Scratchpad
Local Memory size :         32768
Device endianess :          little
Device Type :               CL_DEVICE_TYPE_CPU
Max Compute units :         16
Name :                      AMD Phenom™ 11 X4 945
                            Processor
```

**Source :** NVIDIA, Khronos AMD, References

## The OpenCL Specification

- **Execution model :**
  - Defines
    - How the OpenCL environment is configured on the host
    - How kernels are executed on device
  - This includes
    - Setting up an OpenCL context on the host,
    - Providing mechanism for host-device interaction, &
    - defining a concurrency model used for kernel execution on device
    - The host sets up a kernel for the GPU to run and instantiates it with some special degree of parallelism.

**Source :** NVIDIA, Khronos AMD, References

## The OpenCL Execution Model

❖ **Execution Model**

- Application consists of **two** distinct parts
- **The host program**
  - Runs on the host
  - OpenCL does not define the details of how the host progrma works, only how it interacts with objects defined in OpenCL
- **A Collection of Kernels**
  - The Kernel execute on the OpenCL device

**Source :** NVIDIA, Khronos AMD, References

## OpenCL Implementation Steps

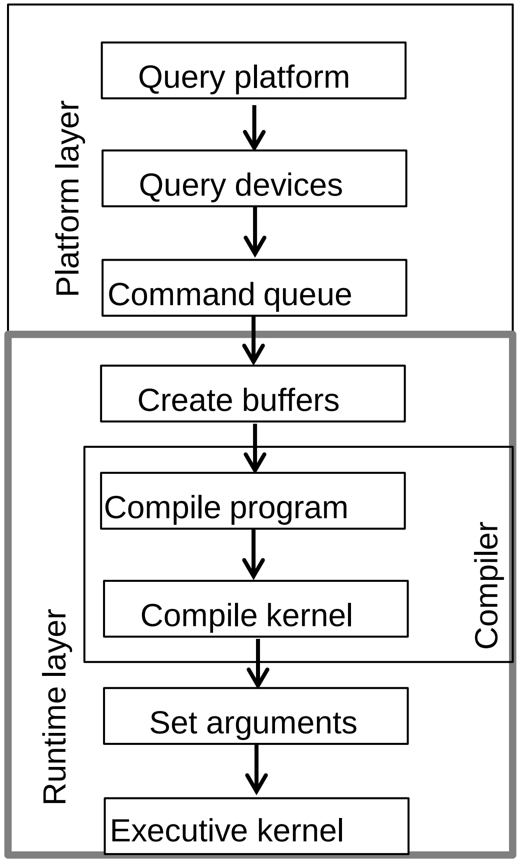


Figure 2 Programming steps to writing a complete OpenCL applications

## OpenCL Important Steps – Implementation

***Step 1 : Discover and initialize the platforms***

***Step 2 : Discover and initialize the devices***

***Step 3 : Create context***

***Step 4 : Create a command queue***

***Step 5 : Create device buffers***

***Step 6 : Write host data device buffers***

***Step 7 : Create and compile the program***

***Step 8 : Create the kernel***

***Step 9 : Set the kernel arguments***

***Step 10 : Configure the work -items structure***

***Step 11 : Enqueue the kernel for execution***

***Step 12 : Read the output buffer back to the host***

***Step 13 : Release OpenCL resources***

## OpenCL Important Steps – Implementation

***Step 1*** **:** *Discover and initialize the platforms*

***Step 2*** **:** *Discover and initialize the devices*

***Step 3*** **:** *Create context*

***Step 4*** **:** *Create a command queue*

***Step 5*** **:** *Create device buffers*

***Step 6*** **:** *Write host data device buffers*

The OpenCL specification in four parts, called models.

- **Platform Model**
- **Execution Model**
- **Memory Model**
- **Programming Model**

## OpenCL Important Steps – Implementation

***Step 7*** ***:*** *Create and compile the program*

***Step 8*** ***:*** *Create the kernel*

***Step 9*** *: Set the kernel arguments*

***Step 10 :*** *Configure the work -items structure*

***Step 11 :*** *Enqueue the kernel for execution*

***Step 12 :*** *Read the output buffer back to the host*

***Step 13 :*** *Release OpenCL resources*

The OpenCL specification in four parts, called models.

- **Platform Model**
- **Execution Model**
- **Memory Model**
- **Programming Model**

## OpenCL Important Steps – Implementation

- Create an OpenCL context on the first available device
- Create a command –queue on the first available device
- Load a kernel file (hello-world.cl) and build it into a program object
- Create a kernel object for the kernel function hello_world()
- Query the kernel for execution
- Read the results of the kernel back into the result buffer

## OpenCL Important Steps – Implementation

```
_kernel void hello_kernel(_global *, *, )
{
      int gid = get_global_id(0);
     ………
 }

int main (int argc,  char** argv)
{
// Create an OpencL context on first available platform

// Create an command-queue on the first device
//  available on the created context
```

## The OpenCL Execution Model

❖ **Execution Model - Kernels**

➢ **A Collection of Kernels**

- Execute on the OpenCL device
- Do the real work of an OpenCL application
- Simple functions transform **input** memory objects into **output** memory objects

**Execution Model - Kernels**

➢ **OpenCL defines two types of Kernels**

- **OpenCL** Kernels & **Native** Kernels

Source : Khronous, & References

## The OpenCL Execution Model

- ❖ **Execution Model : Defines how the kernels execute**
  - ➢ **Several Steps Exist.**
    - **FIRST :** How an individual kernel runs on an OpenCL device ?
    - **Second:** How the host defines the **context** for kenrel execution
    - **THIRD:** How the kernels are enqueued for execution

**Source :** Intel, NVIDIA, Khronos AMD, References

## The OpenCL Execution Model

- **Execution Model - Kernels**
  - **OpenCL Kernels**
    - Written in OpenCL C programming language and compiled with the OpenCL Compiler
    - All OpenCL implementations must support OpenCL Kernels
  - **Native Kernels**
    - Functions created outside of **OpenCL** and accessed within **OpenCL** through a function pointer. ( An Optional functionality within in **OpenCL** exist )

**Source :** NVIDIA, Khronos AMD, References

## The OpenCL Execution Model

- The OpenCL Execution Environment defines the following how the kernel execute
  - Contexts
  - Command Queues
  - Events
  - Memory Objects (Buffers -large array /images
    - Buffers (allocate buffer & return memory object)
    - Image (2D & 3D)
  - Flush & Finish

**Source : Intel** NVIDIA, Khronos AMD, References

## The OpenCL on Intel Xeon Phi

### Mapping :OpenCL constructs to Intel Xeon Phi coprocessor

- Conceptually, at initialization time, the OpenCL driver creates 240 SW threads and pins them to the HW threads (for a 60-core configuration).
- Then, following a **`clEnqueueNDRange()`** call, the driver schedules the work groups (WG) of the current NDRange on the 240 threads.
- A WG is the smallest task being scheduled on the threads. So calling **`clEnqueueNDRange()`** with less than 240 WGs, leaves the coprocessor underutilized

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping :OpenCL constructs to Intel Xeon Phi coprocessor**

- The OpenCL compiler implicitly vectorizes the WG routine based on dimension zero loop, i.e., the dimension zero loop is unrolled by the vector size.

```
__Kernel ABC(…)
for(int i = 0; i < get_local_size(2); i++)
for(int j = 0; j < get_local_size(1); j++)
for(int k = 0; k < get_local_size(0); k += VECTOR_SIZE)
   Vector_Kernel_Body;
```

The vector size of Intel Xeon Phi coprocessor is 16,

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping :OpenCL constructs to Intel Xeon Phi coprocessor**

- While the OpenCL specification provides various ways to express parallelism and concurrency, some of them will not map well to Intel Xeon Phi coprocessor. Most importantly, design your application to exploit its parallelism

**Multi-threading**

- To get good utilization of the 240 HW threads, it's best to have more than 1000 WGs per NDRange. Having 180–240 WGs per NDRange will provide basic threads utilization; however, the execution may suffer from poor load-balancing and high invocation overhead.

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping : OpenCL constructs to Intel Xeon Phi coprocessor**

**Multi-threading**

- **Recommendation**: Have at least 1000 WGs per NDRange to optimally utilize the Intel Xeon Phi coprocessor HW threads. Applications with NDRange of 100 WGs or less will suffer from serious under-utilization of threads

- Single WG execution duration also impacts the threading efficiency. Lightweight WGs are also not recommended, as these may suffer from relatively high overheads.

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping : OpenCL constructs to Intel Xeon Phi coprocessor**

**Vectorization :**

- OpenCL on Intel Xeon Phi coprocessor includes an implicit vectorization module. The OpenCL compiler automatically vectorizes the implicit WG loop over the work items in dimension zero (see example above).
- The vectorization width is currently 16, regardless of the data type used in the kernel. In future implementations, we may vectorize even 32 elements. As OpenCL work items are guaranteed to be independent, the OpenCL vectorizer needs no feasibility analysis to apply vectorization.
- Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping : OpenCL constructs to Intel Xeon Phi coprocessor**

**Vectorization :**

- Note that the vectorized kernel is only used if the local size of dimension zero is greater than or equal to 16. Otherwise, the OpenCL runtime runs scalar kernel for each of the work items. If the WG size at dimension zero is not divisible by 16, then the end of the WG needs to be executed by scalar code. This isn't an issue for large WGs, e.g., 1024 items at dimension zero, but is for WGs of size 31 on dimension zero.

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping : OpenCL constructs to Intel Xeon Phi coprocessor**

**Vectorization :**

- **Recommendation 1:** Don't manually vectorize kernels, as the OpenCL compiler is going to scalarize your code to prepare it for implicit vectorization.

- **Recommendation 2:** Avoid using a WG size that is not divisible by 32 (16 will work for now).

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping : OpenCL constructs to Intel Xeon Phi coprocessor**

**Vectorization :**

- **Work-Item-ID non-uniform control flow**

  Understand the difference between uniform and nonuniform control flow in the context of vectorization

- **Data Alignment :** For various reasons, memory access that is vector-size-aligned is faster than unaligned memory access. In the Intel Xeon Phi coprocessor, OpenCL buffers are guaranteed to start on a vector-size-aligned address

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping : OpenCL constructs to Intel Xeon Phi coprocessor**

**Vectorization :**

- **Recommendation 1:** Don’t use NDrange offset. If you have to use an offset, then make it a multiple of 32, or at least a multiple of 16.

- **Recommendation 2:** Use local size that is a multiple of 32, or at least of 16..

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping : OpenCL constructs to Intel Xeon Phi coprocessor**

**Algorithm Design :**

- **Intra WG data reuse**

  Designing your application to maximize the amount of data reuse from the caches is the first memory optimization to apply.

- **Data Data access pattern :** Consecutive data access usually allows the best memory system performance

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping : OpenCL constructs to Intel Xeon Phi coprocessor**

**Algorithm Design :**

- **Data layout : Pure SOA (Structure-of-Arrays)** data layout results in simple and efficient vector loads and stores

- With **AOS (Array-of-Structures)** data layout, the generated vectorized kernel needs to load and store data via gather and scatter instructions, which are less efficient than simple vector load and store.

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping : OpenCL constructs to Intel Xeon Phi coprocessor**

**Algorithm Design :**

- **Data Prefetching :** With the Intel Xeon Phi coprocessor being an in-order machine, data prefetching is an essential way to bring data closer to the cores, in parallel with other computations. Loads and stores are executed serially, with parallelism.
- Manual prefetching can be inserted by the programmer into the OpenCL kernel, via the prefetch built-in.
- Automatic SW prefetches to the L1 and L2 are inserted by the OpenCL compiler for data accessed in future iterations,

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping : OpenCL constructs to Intel Xeon Phi coprocessor**

**Summary :**While designing your OpenCL application for Intel Xeon Phi coprocessor, you should pay careful attention to the following aspects:

- Include enough work groups within each NDRange—a minimum of 1000 is recommended.
- Avoid lightweight work groups. Don’t hesitate using the maximum local size allowed (currently 1024). Keep the WG size a multiple of 32.
- Avoid ID(0) dependent control flow. This allows efficient implicit vectorization.

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## The OpenCL on Intel Xeon Phi

**Mapping : OpenCL constructs to Intel Xeon Phi coprocessor**

**Summary :**While designing your OpenCL application for Intel Xeon Phi coprocessor, you should pay careful attention to the following aspects:

- Prefer consecutive data access.
- Data layout preferences: AOS for sparse random access; pure SOA or AOSOA(32) otherwise.
- Exploit data reuse through the caches within the WG—tiling/blocking.
- If auto-prefetching didn't kick in, use the PREFETCH built-in to bring the global data to the cache 500–1000 cycles before use.
- Don't use local memory. Avoid using barriers.

Source : www.intel.com : The Intel SDK for OpenCL Applications XE The OpenCL 1.1 Quick Reference Guide

## References

1. Randi J. Rost, OpenGL – shading Language, Second Edition, Addison Wesley 2006
2. GPGPU Reference http://www.gpgpu.org
3. NVIDIA http://www.nvidia.com
4. NVIDIA tesla http://www.nvidia.com/object/tesla_computing_solutions.html
5. RAPIDMIND http://www.rapidmind.net
6. Peak Stream - Parallel Processing (Acquired by Google in 2007) http:/www.google.com
7. guru3d.com http://www.guru3d.com/news/sandra-2009-gets-gpgpu-support/
   ATI & AMD http://ati.amd.com/products/radeon9600/radeon9600pro/index.html
8. AMD http:www.amd.com
9. AMD Stream Processors http://ati.amd.com/products/streamprocessor/specs.html
10. RAPIDMIND & AMD http://www.rapidmind.net/News-Aug4-08-SIGGRAPH.php
11. General-purpose computing on graphics processing units (GPGPU) http://en.wikipedia.org/wiki/GPGPU
12. Khronous Group, OpenGL 3, December 2008 URL : http://www.khronos.org/opencl
13. *OpenCL - The open standard for parallel programming of heterogeneous systems* URL : http://www.khronos.org/opencl
14. Programming the GPU and a brief intro to the OPENGL shading language – Marcel Cohan & VVR Talk
15. David B Kirk, Wen-mei W. Hwu nvidia corporation, 2010, Elsevier, Morgan Kaufmann Publishers, 2011
16. Benedict R Gaster, Lee Howes, David R Kaeli, Perhadd Mistry Dana Schaa, Heterogeneous Computing with OpenCL, Elsevier, Moran Kaufmann Publishers, 2011
17. The OpenCL 1.2 Specification (Document Revision 15) Last Released November 15, 2011 Editor : Aaftab Munshi Khronos OpenCL Working Group
18. The OpenCL 1.1 Quick Reference card

## References

19. http://developer.amd.com/sdks/AMDAPPSDK/downloads/Pages/default.aspx AMD APP SDK with OpenCL 1.2 Support
20. http://developer.amd.com/sdks/AMDAPPSDK/downloads/Pages/default.aspx#oneAMD-APP-SDKv2.7 (Linux) with OpenCL 1.2 Support
21. http://icl.cs.utk.edu/magma/software/ MAGMA OpenCL
22. http://developer.amd.com/zones/OpenCLZone/pages/GettingStarted.aspx Getting Started with OpenCL
23. http://developer.amd.com/openclforum AMD Developer OpenCL FORUM
24. http://developer.amd.com/zones/OpenCLZone/programming/pages/benchmarkingopenclperformance.aspx AMD Developer Central - Programming in OpenCL - Benchmarks performance
25. http://developer.amd.com/sdks/AMDAPPSDK/assets/opencl-1.2.pdf OpenCL 1.2 (pdf file)
26. http://developer.amd.com/zones/opensource/pages/ocl-emu.aspx AMD OpenCL Emulator-Debugger
27. http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf The OpenCL 1.2 Specification (Document Revision 15) Last Released November 15, 201 Editor : Aaftab Munshi <I> Khronos OpenCL Working Group
28. http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/ OpenCL1.1 Reference Pages
29. The Intel SDK for OpenCL Applications XE – Optimization Guide includes many more details.

**Source :** Intel, NVIDIA, Khronos AMD, References

## Thank you

***Any Questions ?***

**Source :** NVIDIA,  Khronos   AMD, References