APPENDIX A

OpenCL on Xeon Phi

Open Computing Language (OpenCL) standardizes language and *application programming interfaces* (APIs) for programming heterogeneous parallel computing systems, such as hosts containing Xeon Phi coprocessors. This is an open standard maintained by the industry consortium, Khronos Group, and adopted by various leading companies to make the language-based applications portable across devices.[1]

The Intel *Software Development Kit* (SDK) for OpenCL Applications XE 2013 provides support for compiling and running OpenCL 1.2-compliant applications to execute on hosts containing Xeon Phi coprocessors. Xeon Phi support is available for Linux OS only. The OpenCL tool provides another way of programming and harvesting the power of the Xeon Phi coprocessor in addition to those provided by the various programming models of Intel C/C++/Fortran compilers for Xeon Phi. The SDK contains OpenCL runtime, development tools, and relevant documentation to get you started on developing applications for Xeon Phi using the OpenCL standard. You can also use Intel VTune Amplifier XE to profile the applications built with OpenCL development tools.

A single runtime supports both Intel Xeon processors and Intel Xeon Phi coprocessors. The package supports multiple Xeon Phi coprocessors.

This appendix introduces the types of tools available for building OpenCL code for Xeon Phi. It assumes that you are already familiar with the OpenCL programming concepts.

Installation

There are two packages provided for installation purposes:

- The runtime-only package, containing the runtime and the OpenCL compiler.
- The full package, which includes the runtime package and SDKs with the necessary header files for C/C++ development with OpenCL, examples, and documentation.

You can download these packages from `http://software.intel.com/en-us/vcsource/tools/opencl-sdk-xe`. The final download dialog will also provide a public key Intel-x-y-z.pub that you need to download as part of the package.

Install the package by the following steps:

1. First import the public key Intel-x-y-z.pub where x, y, z will have various dashed symbols that are provided by your download package. For example, the version of the package I downloaded had a public key `Intel-E901-172E-EF96-900F-B8E1-4184-D7BE-0E73-xxxx-xxxx.pub`. To import the public key as a root use the command:

   ```
   rpm --import <package public key>
   ```

[1] `http://www.khronos.org/opencl/`

2. Verify integrity of the Redhat Package Manager (rpm) packages by running `rpm --checksig`

```
Command_prompt> rpm --checksig *.rpm
opencl-1.2-base-3.0.67279-1.x86_64.rpm: rsa sha1 (md5) pgp md5 OK
opencl-1.2-devel-3.0.67279-1.x86_64.rpm: rsa sha1 (md5) pgp md5 OK
opencl-1.2-intel-cpu-3.0.67279-1.x86_64.rpm: rsa sha1 (md5) pgp md5 OK
opencl-1.2-intel-devel-3.0.67279-1.x86_64.rpm: rsa sha1 (md5) pgp md5 OK
opencl-1.2-intel-mic-3.0.67279-1.x86_64.rpm: rsa sha1 (md5) pgp md5 OK
```

3. Install the OpenCL package by executing install-cpu+mic.sh as follows:

```
Command_prompt > sudo ./install-cpu+mic.sh
In case of failure please consult README file
Preparing...              ########################################### [100%]
   1:opencl-1.2-base       ########################################### [ 20%]
   2:opencl-1.2-intel-cpu  ########################################### [ 40%]
   3:opencl-1.2-intel-mic  ########################################### [ 60%]
   4:opencl-1.2-intel-devel ########################################### [ 80%]
   5:opencl-1.2-devel      ########################################### [100%]
Done.
```

The rpm packages installed are as follows:

`opencl-base`: Allows applications to select between different OpenCL implementations at runtime using the OpenCL installable client driver (ICD). The ICD allows the coexistence of OpenCL implementation from multiple vendors.

`opencl-intel-cpu`: Enables Intel Xeon and Intel Core processors as an OpenCL device. The package contains the compiler and runtime for Intel processors.

`opencl-intel-mic`: Enables an Intel Xeon Phi coprocessor as an OpenCL device. The MPSS must be available for this package to work.

`opencl-devel`: Contains OpenCL C header files for the development of opencl applications.

`opencl-intel-devel` : Provides Intel SDK for opencl which includes Kernel Builder which enables full offline OpenCL language compilation and analysis of OpenCl kernels.

There are other ways to install the package. Refer to readme.txt that comes with the package for a more customized installation process.

Building and Running OpenCL Application

The SDK provides the following tools to help you build and execute OpenCL applications on Xeon Phi.

Once you install OpenCL SDK on your system, the library `libOpenCL.so` for building and running OpenCL application is copied to the /usr/lib64 folder and the base OpenCL header files are available in the /usr/include/CL system-level folder. To build an OpenCL file, you need to invoke the compiler as follows:

1. Set C++ compiler path and environment.
2. Invoke the C++ compiler as

```
Command_prompt > icc test.cpp –lOpenCL –otest.out
```

where test.cpp contains OpenCL calls and includes the header file CL/cl.h to declare the OpenCL APIs.

The test.cpp will include the OpenCL kernel calls that need to run on the Xeon Phi coprocessors. The SDK itself comes with a separate KernelBuilder64 tool that allows you to check your kernel code for OpenCL syntax errors and test its correctness, look at generated *low-level virtual machine* (LLVM) and assembly code, and analyze kernel performance. Please refer to the users guide for details of the Kernel Builder tool.[2]

Intel OpenCL implementation supports shared context between the host Xeon processor and multiple Xeon Phi coprocessors. This implementation allows the sharing of memory objects and events between these heterogeneous devices for the development of workloads that run on both devices. You can also create separate contexts for the Xeon host and Xeon Phi devices to handle them separately by creating separate OpenCL buffers for each Xeon Phi and Xeon device.

Performance Optimization

Although OpenCL codes can be portable across platforms, measures should be taken to optimize OpenCL for the individual hardware it is targeted to run on. For example, Xeon Phi has certain features, such as its 512-bit vector unit and 240 hardware threads, that must be utilized to obtain best performance.

The execution order of work items within a *work group* (WG) and the execution order of WGs are implementation-specific. When launching the kernel for execution, the host code defines the global work dimensions (NDRange) and may also set the partitioning of the NDRange to the WG size. The OpenCL basic parallelism enables the kernel to execute on multiple work items. The Xeon Phi architecture supports SIMD parallelization, which enables the processing of multiple work items with SIMD as necessary to achieve high performance on Xeon Phi with OpenCL. The vectorization unit packs work items for dimension 0 of the NDRange, the innermost loop to utilize SIMD units on Xeon Phi.

On Xeon Phi, each WG is assigned one thread that loops over all the work items within the WG, with SIMD processing thus providing parallelism within the WG level. The parallelism between the WGs is implemented by threading.

Guidelines for optimizing OpenCL on Xeon Phi include the following:[3]

- *Parallelization*: It is recommended that you have at least 960 (a multiple of number of threads) WGs per NDRange with longer execution duration per WG. This allows the OpenCL runtime to utilize the 240 threads with load balancing. The long execution duration (100K clock cycles) for WGs helps reduce context switching overhead.
- *Efficient vectorization*: The Intel OpenCL compiler implicitly vectorizes the WG routines. In order for the compiler to perform its vectorization, do not vectorize the kernels. This requires extra work for the compiler to scalarize the code for later implicit vectorization. Use a workgroup size that is a multiple of 32, as that will allow the compiler to efficiently vectorize the code. The performance benefit may be lower for kernels with complicated control flows.
- *Data alignment*: For efficient data access and vectorization, use a local size that is a multiple of 32.
- *Unit stride access*: Try writing your kernel so that the innermost loop of the WG accesses memory in unit stride for low latency memory access.
- *Local shared memory*: Avoid using shared local memory on Xeon Phi architecture as this results in unnecessary data movement between the cache and GDDR memory to simulate the shared local memory model of GPU architectures.
- *Data prefetching*: Some Xeon Phi applications may benefit from using `prefetch` built into the kernel code. This allows you to reduce memory access latency in cases where hardware prefetch does not kick in.
- *Tiling/blocking*: Make use of Xeon Phi's cache hierarchy by maximizing intra-WG data reuse by tiling/blocking of the code.
- *Array of Structures/Structure of Arrays (AOS/SOA)*: Use AOS for sparse random access and SOA for access to improve cache locality and usage.

[2]Intel® SDK for OpenCL* Applications XE 2013 User's Guide for Linux* OS. Document Number 328882-002US, Intel Corporation.
[3]Intel SDK for OpenCL Applications XE 2013, Optimization Guide for Linux OS. Document Number 326542-002US, Intel Corporation.