SnuCL User Manual

Center for Manycore Programming
Department of Computer Science and Engineering
Seoul National University, Seoul 151-744, Korea
`http://aces.snu.ac.kr`

Release 1.3.2 December 2013

Contents

Chapter 1

Introduction

SnuCL is an OpenCL framework and freely available, open-source software developed at Seoul National University.

SnuCL naturally extends the original OpenCL semantics to the heterogeneous cluster environment. The target cluster consists of a single host node and multiple compute nodes. They are connected by an interconnection network, such as Gigabit and InfiniBand switches. The host node contains multiple CPU cores and each compute node consists of multiple CPU cores and multiple GPUs. For such clusters, SnuCL provides an illusion of a single heterogeneous system for the programmer. A set of CPU cores, a GPU, or an Intel Xeon Phi coprocessor becomes an OpenCL compute device. SnuCL allows the application to utilize compute devices in a compute node as if they were in the host node.

In addition, SnuCL integrates multiple OpenCL platforms from different vendor implementations into a single platform. It enables OpenCL applications to share objects (buffers, events, etc.) between compute devices of different vendors.

As a result, SnuCL achieves both high performance and ease of programming in both a single system (i.e., a single operating system instance) and a heterogeneous cluster system (i.e., a cluster with many operating system instances).

SnuCL provides three OpenCL platforms:

- **SnuCL CPU** provides a set of CPU cores as an OpenCL compute device.
- **SnuCL Single** is an OpenCL platform that integrates all ICD compatible OpenCL implementations installed in a single system.
- **SnuCL Cluster** is an OpenCL platform for a heterogeneous cluster. It integrates all ICD compatible OpenCL implementations in the compute nodes and provides an illusion of a single heterogeneous system for the programmer.

Chapter 2

Installation

SnuCL platforms can be installed separately. To install SnuCL CPU, see Section 2.1. To install SnuCL Single, see Section 2.2. To install SnuCL Cluster, see Section 2.3.

2.1 Installing SnuCL CPU

Prerequisite The OpenCL ICD loader (i.e., `libOpenCL.so`) should be installed in the system. If there are other ICD compatible OpenCL implementations in the system, they may provide the OpenCL ICD loader. Or you can download the source code of the official OpenCL 1.2 ICD loader from `http://www.khronos.org/registry/cl/`.

Installing Download the SnuCL source code from `http://snucl.snu.ac.kr`. The package includes the OpenCL ICD driver, the SnuCL CPU runtime, and the source-to-source translator for a CPU device. Then, untar it and configure shell environment variables for SnuCL.

```
user@computer:~/$ tar zxvf snucl.1.3.tar.gz
user@computer:~/$ export SNUCLROOT=$HOME/snucl
user@computer:~/$ export PATH=$PATH:$SNUCLROOT/bin
user@computer:~/$ export
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SNUCLROOT/lib
```

Build SnuCL CPU using a script named `build_cpu.sh`. It may require a long time to build the LLVM compiler infrastructure and all OpenCL built-in functions.

```
user@computer:~/$ cd snucl/build
```

```
user@computer:~/snucl/build$ ./build_cpu.sh
```

As a result, the following files will be created:

- `snucl/lib/libsnucl_cpu.so`: the SnuCL CPU runtime
- `snucl/bin/clang`, `snucl/bin/snuclc-merger`, `snucl/lib/libSnuCLTranslator.a`, `snucl/lib/libSnuCLTranslator.so`, `snucl/lib/libsnucl-builtins-lnx??.a`: components of the source-to-source translator

Then, register SnuCL CPU to the OpenCL ICD loader. This may require the root permission.

```
user@computer:~/$ echo libsnucl_cpu.so >
  /etc/OpenCL/vendors/snucl_cpu.icd
```

An example run Test running an OpenCL application and check it runs correctly. The example run is started on the target system by entering the following commands:

```
user@computer:~/$ cd snucl/apps/sample
user@computer:~/snucl/apps/sample$ make cpu
user@computer:~/snucl/apps/sample$ ./sample
[ 0] 100
[ 1] 110
[ 2] 120
[ 3] 130
[ 4] 140
[ 5] 150
[ 6] 160
[ 7] 170
[ 8] 180
[ 9] 190
[10] 200
[11] 210
[12] 220
[13] 230
[14] 240
[15] 250
[16] 260
[17] 270
[18] 280
```

```
[19] 290
[20] 300
[21] 310
[22] 320
[23] 330
[24] 340
[25] 350
[26] 360
[27] 370
[28] 380
[29] 390
[30] 400
[31] 410
user@computer:~/snucl/apps/sample$
```

2.2 Installing SnuCL Single

Prerequisite There should be one or more ICD compatible OpenCL implementations installed in the system, e.g., Intel SDK for OpenCL Applications, AMD Accelerated Parallel Processing SDK, and NVIDIA OpenCL SDK. They serves the OpenCL programming environment for each compute device.

Since SnuCL CPU follows the OpenCL ICD extension, it can also be integrated into SnuCL Single. For example, OpenCL applications can use both multicore CPUs and NVIDIA GPUs at the same time by integrating SnuCL CPU and NVIDIA OpenCL SDK.

In addition, the OpenCL ICD loader (i.e., `libOpenCL.so`) should be installed in the system. The ICD compatible OpenCL implementations may provide the OpenCL ICD loader. Or you can download the source code of the official OpenCL 1.2 ICD loader from `http://www.khronos.org/registry/cl/`.

Installing Download the SnuCL source code from `http://snucl.snu.ac.kr`. The package includes the OpenCL ICD driver and the SnuCL Single runtime. Then, untar it and configure shell environment variables for SnuCL.

```
user@computer:~/$ tar zxvf snucl.1.3.tar.gz
user@computer:~/$ export SNUCLROOT=$HOME/snucl
user@computer:~/$ export PATH=$PATH:$SNUCLROOT/bin
user@computer:~/$ export
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SNUCLROOT/lib
```

Build SnuCL Single using a script named `build_single.sh`.

```
user@computer:~/$ cd snucl/build
user@computer:~/snucl/build$ ./build_single.sh
```

As a result, the `snucl/lib/libsnucl_single.so` file will be created.

Then, register SnuCL Single to the OpenCL ICD loader. This may require the root permission.

```
user@computer:~/$ echo libsnucl_single.so >
   /etc/OpenCL/vendors/snucl_single.icd
```

An example run Test running an OpenCL application and check it runs correctly. The example run is started on the target system by entering the following commands:

```
user@computer:~/$ cd snucl/apps/sample
user@computer:~/snucl/apps/sample$ make single
user@computer:~/snucl/apps/sample$ ./sample
[ 0] 100
[ 1] 110
[ 2] 120
[ 3] 130
[ 4] 140
[ 5] 150
[ 6] 160
[ 7] 170
[ 8] 180
[ 9] 190
[10] 200
[11] 210
[12] 220
[13] 230
[14] 240
[15] 250
[16] 260
[17] 270
[18] 280
[19] 290
[20] 300
[21] 310
```

```
[22] 320
[23] 330
[24] 340
[25] 350
[26] 360
[27] 370
[28] 380
[29] 390
[30] 400
[31] 410
user@computer:~/snucl/apps/sample$
```

2.3 Installing SnuCL Cluster

Prerequisite There should be one or more ICD compatible OpenCL implementations installed in the compute nodes, e.g., Intel SDK for OpenCL Applications, AMD Accelerated Parallel Processing SDK, and NVIDIA OpenCL SDK. They serves the OpenCL programming environment for each compute device.

Since SnuCL CPU follows the OpenCL ICD extension, it can also be integrated into SnuCL Cluster. For example, OpenCL applications can use both multicore CPUs and NVIDIA GPUs in the compute nodes by integrating SnuCL CPU and NVIDIA OpenCL SDK.

An MPI implementation (e.g., Open MPI) should be installed in both the host node and the compute nodes. Additionally, you must have an account on all the nodes. You must be able to ssh between the host node and the compute nodes without using a password.

Installing SnuCL Cluster should be installed in all the nodes to run OpenCL applications on the cluster. You can install SnuCL Cluster on the local hard drive of each node, or on a shared filesystem such as NFS.

Download the SnuCL source code from `http://snucl.snu.ac.kr`. The package includes the SnuCL Cluster runtime. Put the gzipped tarball in your work directory and untar it.

```
user@computer:~/$ tar zxvf snucl.1.3.tar.gz
```

Then, configure shell environment variables for SnuCL. Add the following configuration in your shell startup scripts (e.g., `.bashrc`, `.cshrc`, `.profile`, etc.)

```
export SNUCLROOT=$HOME/snucl
export PATH=$PATH:$SNUCLROOT/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SNUCLROOT/lib
```

Build SnuCL Cluster using a script named `build_cluster.sh`.

```
user@computer:~/$ cd snucl/build
user@computer:~/snucl/build$ ./build_cluster.sh
```

As a result, the `snucl/lib/libsnucl_cluster.so` file (i.e., the SnuCL Cluster runtime) will be created.

An example run Test running an OpenCL application and check it runs correctly. First, you should edit the `snucl/bin/snucl_nodes` file on the host node. The file specifies the nodes' hostname in the cluster. (See Chapter 3). The example run is started on the target system by entering the following commands on the host node:

```
user@computer:~/$ cd snucl/apps/sample
user@computer:~/snucl/apps/sample$ make cluster
user@computer:~/snucl/apps/sample$ snuclrun 1 ./sample
[ 0] 100
[ 1] 110
[ 2] 120
[ 3] 130
[ 4] 140
[ 5] 150
[ 6] 160
[ 7] 170
[ 8] 180
[ 9] 190
[10] 200
[11] 210
[12] 220
[13] 230
[14] 240
[15] 250
[16] 260
[17] 270
[18] 280
[19] 290
[20] 300
```

```
[21] 310
[22] 320
[23] 330
[24] 340
[25] 350
[26] 360
[27] 370
[28] 380
[29] 390
[30] 400
[31] 410
user@computer:~/snucl/apps/sample$
```

Chapter 3

Using SnuCL

3.1 Writing OpenCL applications using SnuCL

SnuCL follows the OpenCL 1.2 core specification. SnuCL provides a unified OpenCL platform for compute devices of different vendors and in different nodes. OpenCL applications written for a single system and a single OpenCL vendor implementation can run on a heterogeneous cluster and on compute devices of different vendors, without any modification.

If SnuCL and other OpenCL implementations are installed in the same system, the `clGetPlatformID` function returns multiple platform IDs and OpenCL applications should choose the proper one. You may use the `clGetPlatformInfo` function to check the name (`CL_PLATFORM_NAME`) and vendor (`CL_PLATFORM_VENDOR`) of platforms. The name and vendor of the SnuCL platforms are as follows:

Table 3.1: The name and vendor of the SnuCL platforms

Platform	`CL_PLATFORM_NAME`	`CL_PLATFORM_VERSION`
SnuCL CPU	“`SnuCL CPU`”	“`Seoul National University`”
SnuCL Single	“`SnuCL Single`”	“`Seoul National University`”
SnuCL Cluster	“`SnuCL Cluster`”	“`Seoul National University`”

3.2 Building OpenCL applications using SnuCL

To use SnuCL CPU or SnuCL Single, you only need to link your applications with the OpenCL ICD driver and set the include path as follows:

```
user@computer:~$ gcc -o your_program your_source.c
   -I$(SNUCLROOT)/inc -L$(SNUCLROOT)/lib -lOpenCL
```

To build an application for the cluster environment, you should use the compiler for MPI programs (i.e., `mpicc` and `mpic++`) and link the application with the SnuCL Cluster runtime as follows:

```
user@computer:~$ mpicc -o your_program your_source.c
   -I$(SNUCLROOT)/inc -L$(SNUCLROOT)/lib -lsnucl_cluster
```

3.3 Running OpenCL applications using SnuCL

To run an OpenCL application on a single node, just execute the application. To run an OpenCL application on a cluster, you can use a script named `snuclrun` as follows:

```
snuclrun <number of compute nodes> <program> [<program
   arguments>]
```

`snuclrun` uses a hostfile, `snucl/bin/snucl_nodes`, which specifies the nodes' hostname in the cluster. `snucl_nodes` follows the hostfile format in the installed MPI implementation. The first node becomes the host node and the other nodes become the compute nodes.

```
hostnode slots=1 max_slots=1
computenode1 slots=1 max_slots=1
computenode2 slots=1 max_slots=1
..
```

Chapter 4

Understanding SnuCL

4.1 What SnuCL Cluster does with your OpenCL applications

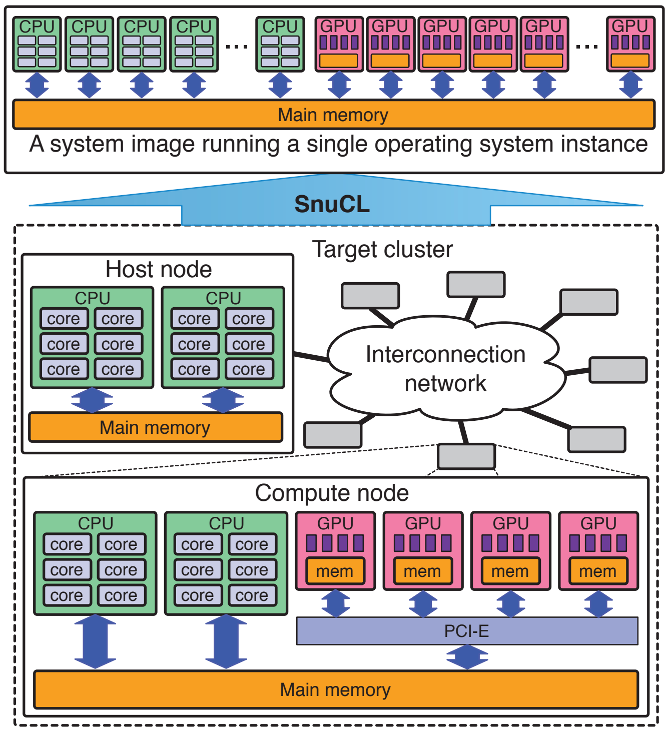

Figure 4.1: Our Approach.

SnuCL provides a system image running a single operating system instance for heterogeneous CPU/GPU cluster to the user. It allows the application to utilize compute devices in a compute node as if they were in the host node. The user can launch a kernel to a compute device or manipulate a memory object in a remote node using only OpenCL API functions. This enables OpenCL applications written for a single node to run on the cluster without any modification. That is, with SnuCL, an OpenCL application becomes portable not only between heterogeneous computing devices in a single node, but also between those in the entire cluster environment.

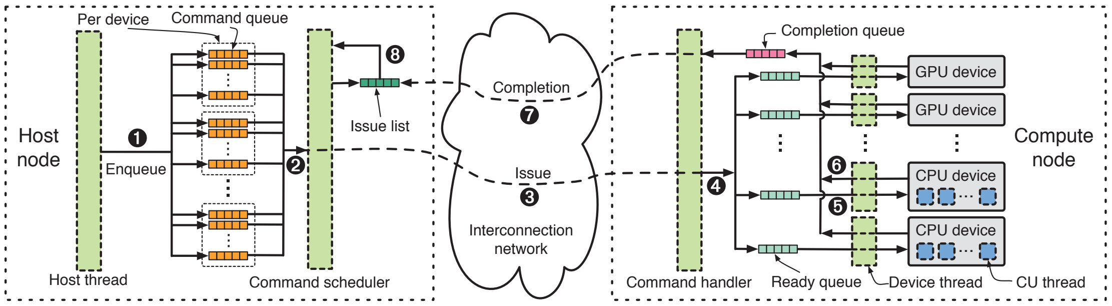

Figure 4.2: The organization of the SnuCL runtime.

This figure shows the organization of the SnuCL runtime. It consists of two different parts for the host node and a compute node.

The runtime for the host node runs two threads: *host thread* and *command scheduler*. When a user launches an OpenCL application in the host node, the host thread in the host node executes the host program in the application. The host thread and command scheduler share the OpenCL command-queues. A compute device may have one or more command-queues. The host thread enqueues commands to the command-queues (① in the figure). The command scheduler schedules the enqueued commands across compute devices in the cluster one by one (②).

When the command scheduler in the host node dequeues a command from a command-queue, the command scheduler *issues* the command by sending a *command message* (③) to the target compute node that contains the target compute device associated with the command-queue. A command message contains the information required to execute the original command. To identify each OpenCL object, the runtime assigns a unique ID to each OpenCL object, such as contexts, compute devices, buffers (memory objects), programs, kernels, events, etc. The command message contains these IDs.

After the command scheduler sends the command message to the target compute node, it calls a non-blocking receive communication API function to wait the completion message from the target node. The command scheduler encapsulates the receive request in the command event object and adds the event object in the *issue list.* The issue list contains event objects associated with the commands that have been issued but have not completed yet.

The runtime for a compute node runs a *command handler thread.* The command handler receives command messages from the host node and executes them across compute devices in the compute node. It creates a command object and an associated event object from the message. After extracting the target device information from the message, the command handler enqueues the command object to the *ready-queue* of the target device (④). Each compute device has a single ready-queue. The ready-queue contains commands that are issued but not launched to the associated compute device yet.

The runtime for a compute node runs a *device thread* for each compute device in the node. The device thread dequeues a command from its ready-queue and executes the command (⑤). When the compute device completes executing the command, the device thread updates the status of the associated event to *completed*, and then inserts the event to the *completion queue* in the compute node (⑥). The command handler in each compute node repeats handling commands and checking the completion queue in turn. When the completion queue is not empty, the command handler dequeues the event from the completion queue and sends a completion message to the host node (⑦).

The command scheduler in the host node repeats scheduling commands and checking the event objects in the issue list in turn until the OpenCL application terminates. If the receive request encapsulated in an event object in the issue list completes, the command scheduler removes the event from the issue list and updates the status of the dequeued event from *issued* to *completed* (⑧).

Chapter 5

Known Issues

The following is a list of known issues in the current SnuCL implementation.

- OpenCL API functions added in OpenCL 1.2 (`clEnqueueReadBufferRect`, `clEnqueueWriteBufferRect`, `clEnqueueCopyBufferRect`, `clCompileProgram`, and `clLinkProgram`) are not supported for NVIDIA GPUs because NVIDIA OpenCL SDK only supports OpenCL 1.1.
- Compute devices can't be partitioned into sub-devices.
- If a buffer and its sub-buffer are written by different devices, the memory consistency between the buffer and the sub-buffer is not guaranteed.
- The `CL_MEM_USE_HOST_PTR` and `CL_MEM_ALLOC_HOST_PTR` flags don't work for SnuCL Single and SnuCL Cluster.