The **SnuCL** Manual

Version 0.8

October 2011

Center for Manycore Programming

School of Computer Science and Engineering

Seoul National University, Seoul 151-744, Korea

http://aces.snu.ac.kr

1. Introduction

SnuCL is an OpenCL framework and freely available, open-source software developed at Seoul National University. It 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 GPU or a set of CPU cores 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. SnuCL achieves both high performance and ease of programming in a heterogeneous cluster environment.

2. Installation

2.1. Supported Platforms

SnuCL builds on 32-bit, 64-bit flavors of Linux. SnuCL runs on the heterogeneous CPU/GPU clusters. The following processors are supported by SnuCL and become OpenCL compute devices.

- x86 CPUs, ARM CPUs, PowerPC CPUs, and NVIDIA GPUs

SnuCL has been tested on a cluster with the following nodes:

- Host node
  - Intel® Xeon® Processor X5680
  - Red Hat Enterprise Linux Server 5.5
  - OpenMPI 1.4.1
- Compute node
  - Intel® Xeon® Processor X5660
  - NVIDIA GeForce GTX 480
  - Red Hat Enterprise Linux Server 5.5
  - OpenMPI 1.4.1
  - CUDA Toolkit 4.0

2.2. Installing SnuCL

Download the SnuCL framework source code from http://snucl.snu.ac.kr. The package includes the SnuCL runtime, source-to-source translators, and all libraries required by the framework.

Put the gzipped tarball in your work directory. Then, untar it and configure shell environment variables for SnuCL.

```
user@computer:~/$ tar zxvf SnuCL.0.8.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 the SnuCL distribution

```
user@computer:~/$ cd SnuCL/build
user@computer:~/SnuCL/build$ ./install.sh
```

2.3. Verifying the Installation

You should now test running a SnuCL application and check it runs correctly. First, you should edit *snucl_nodes* in the directory $SNUCLROOT/bin. The file specifies the nodes' hostnames in the cluster. (See section 3.2)

The *sample* example is started by entering the following commands:

```
user@computer:~/$ cd SnuCL/apps/sample
user@computer:~/SnuCL/apps/sample$ make
user@computer:~/SnuCL/apps/sample$ snuclrun 1 ./bin/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$
```

3. Using SnuCL

3.1. Building OpenCL applications using SnuCL

SnuCL provides an illusion of a single system for the user for heterogeneous CPU/GPU clusters. 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 heterogeneous system to run on the cluster without any modification.

You only need to link your applications with SnuCL libraries to run your OpenCL applications in the cluster environment. You may use the Makefile template in the directory $SNUCLROOT/apps/sample.

```
EXECUTABLE := <program name>
CCFILES := <source files>
include $(SNUCLROOT)/common.mk
```

3.2. Running OpenCL applications using SnuCL

SnuCL provides a script named *snuclrun* to run an OpenCL application on the cluster. You can use *snuclrun* as follows:

snuclrun <# of compute nodes> <program> [<program arguments>]

snuclrun uses a hostfile, $SNUCLROOT/bin/*snucl_nodes*, which specifies the nodes' hostnames in the cluster. *snucl_nodes* follows the hostfile format in OpenMPI.

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

4. API Functions Supported

SnuCL follows the OpenCL specification version 1.0. The current SnuCL implementation provides the following OpenCL APIs (functions crossed out are not supported by SnuCL).

- Platform
  - clGetPlatformIDs()
  - clGetPlatformInfo()
- Device
  - clGetDeviceIDs()
  - clGetDeviceInfo()
- Context
  - clCreateContext()
  - clCreateContextFromType()
  - clRetainContext()
  - clReleaseContext()
  - ~~clGetContextInfo()~~
- Command Queue
  - clCreateCommandQueue()
  - clRetainCommandQueue()
  - clReleaseCommandQueue()
  - ~~clGetCommandQueueInfo()~~
  - clSetCommandQueueProperty()
- Memory Object

  - clCreateBuffer()
  - ~~clCreateImage2D()~~
  - ~~clCreateImage3D()~~
  - clRetainMemObject()
  - clReleaseMemObject()
  - ~~clGetSupportedImageFormats()~~
  - ~~clGetMemObjectInfo()~~
  - ~~clGetImageInfo()~~
- Sampler
  - ~~clCreateSampler()~~
  - ~~clRetainSampler()~~
  - ~~clReleaseSampler()~~
  - ~~clGetSamplerInfo()~~
- Program Object
  - clCreateProgramWithSource()
  - ~~clCreateProgramWithBinary()~~
  - clRetainProgram()
  - clReleaseProgram()
  - clBuildProgram()
  - clUnloadCompiler()
  - ~~clGetProgramInfo()~~
  - ~~clGetProgramBuildInfo()~~
- Kernel Object
  - clCreateKernel()

    - ~~clCreateKernelsInProgram()~~
    - clRetainKernel()
    - clReleaseKernel()
    - clSetKernelArg()
    - ~~clGetKernelInfo()~~
    - ~~clGetKernelWorkGroupInfo()~~
- Event Object
    - clWaitForEvents()
    - ~~clGetEventInfo()~~
    - clRetainEvent()
    - clReleaseEvent()
- Profiling
    - ~~clGetEventProfilingInfo()~~
- Flush and Finish
    - clFlush()
    - clFinish()
- Enqueued Commands
    - clEnqueueReadBuffer()
    - clEnqueueWiteBuffer()
    - clEnqueueCopyBuffer()
    - ~~clEnqueueReadImage()~~
    - ~~clEnqueueWriteImage()~~
    - ~~clEnqueueCopyImage()~~
    - ~~clEnqueueCopyImageToBuffer()~~

- ~~clEnqueueCopyBufferToImage()~~
- ~~clEnqueueMapBuffer()~~
- ~~clEnqueueMapImage()~~
- ~~clEnqueueUnmapMemObject()~~
- clEnqueueNDRangeKernel()
- clEnqueueTask()
- ~~clEnqueueNativeKernel()~~
- clEnqueueMarker()
- clEnqueueWaitForEvents()
- clEnqueueBarrier()

5. Collective Communication Extensions

SnuCL provides collective communication operations for manipulating OpenCL buffer objects. These extensions to OpenCL are similar to MPI collective communication operations. The following table lists each collective communication operation and its MPI equivalent.

OpenCL	**MPI Equivalent**
clEnqueueBroadcastBuffer	MPI_Bcast
clEnqueueScatterBuffer	MPI_Scatter
clEnqueueGatherBuffer	MPI_Gather
clEnqueueAllGatherBuffer	MPI_Allgather
clEnqueueAlltoAllBuffer	MPI_Alltoall
clEnqueueReduceBuffer	MPI_Reduce
clEnqueueAllReduceBuffer	MPI_Allreduce
clEnqueueReduceScatterBuffer	MPI_Reduce_scatter
clEnqueueScanBuffer	MPI_Scan

The function

```
cl_int clEnqueueBroadcastBuffer( cl_command_queue * cmd_queue_list,
                                 cl_mem src_buffer,
                                 cl_uint num_dst_buffers,
                                 cl_mem * dst_buffer_list,
                                 size_t src_offset,
                                 size_t * dst_offset_list,
                                 size_t cb,
                                 cl_uint num_events_in_wait_list,
                                 const cl_event * event_wait_list,
                                 cl_event * event)
```

enqueues commands to broadcast a buffer object identified by *src_buffer* to all buffer objects in the list identified by *dst_buffer_list*.

cmd_queue_list refers to the command-queues that are associated with the compute devices where the destination buffers identified by *dst_buffer_list* are located.

num_dst_buffers refers to the number of destination buffers identified by *dst_buffer_list*.

src_offset refers to the offset where to begin copying data from *src_buffer*.

dst_offset_list refers to the offsets where to begin copying data into *dst_buffer_list*.

cb refers to the size in bytes to copy.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed.

event returns an event object that identifies this particular broadcast command and can be used to query or queue a wait for this particular command to complete.

The function

```
cl_int clEnqueueScatterBuffer( cl_command_queue * cmd_queue_list,
                               cl_mem src_buffer,
                               cl_uint num_dst_buffers,
                               cl_mem * dst_buffer_list,
                               size_t src_offset,
                               size_t * dst_offset_list,
                               size_t cb,
                               cl_uint num_events_in_wait_list,
                               const cl_event * event_wait_list,
                               cl_event * event)
```

enqueues commands to distribute a buffer object identified by *src_buffer* to each buffer object in the list identified by *dst_buffer_list*.

cmd_queue_list refers to the command-queues that are associated with the compute devices where the destination buffers identified by *dst_buffer_list* are located.

num_dst_buffers refers to the number of destination buffers identified by *dst_buffer_list*.

src_offset refers to the offset where to begin copying data from *src_buffer*.

dst_offset_list refers to the offsets where to begin copying data into *dst_buffer_list*.

cb refers to the size in bytes to copy.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed.

event returns an event object that identifies this particular broadcast command and can be used to query or queue a wait for this particular command to complete.

The function

```
cl_int clEnqueueGatherBuffer( cl_command_queue cmd_queue,
                              cl_uint num_src_buffers,
                              cl_mem * src_buffer_list,
                              cl_mem dst_buffer,
                              size_t * src_offset_list,
                              size_t dst_offset,
                              size_t cb,
                              cl_uint num_events_in_wait_list,
                              const cl_event * event_wait_list,
                              cl_event * event)
```

enqueues commands to gather distinct buffer objects in the list identified by *dst_buffer_list* to a buffer object identified by *src_buffer*

cmd_queue refers to the command-queue that is associated with the compute device where the destination buffer identified by *dst_buffer* are located.

num_src_buffers refers to the number of source buffers identified by src*_buffer_list*.

src_offset_list refers to the offsets where to begin copying data from *dst_buffer_list*.

dst_offset refers to the offset where to begin copying data into dst*_buffer*.

cb refers to the size in bytes to copy.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed.

event returns an event object that identifies this particular broadcast command and can be used to query or queue a wait for this particular command to complete.

The function

```
cl_int clEnqueueAllGatherBuffer( cl_command_queue * cmd_queue_list,
                                 cl_uint num_buffers,
                                 cl_mem * src_buffer_list,
                                 cl_mem * dst_buffer_list,
                                 size_t * src_offset_list,
                                 size_t * dst_offset_list,
                                 size_t cb,
                                 cl_uint num_events_in_wait_list,
                                 const cl_event * event_wait_list,
                                 cl_event * event)
```

enqueues commands to gather data from all buffer objects in the list identified by src*_buffer_list* and distibute it to all buffer objects in the list identified by *dst_buffer_list.*

cmd_queue_list refers to the command-queues that are associated with the compute devices where the destination buffers identified by *dst_buffer_list* are located.

num_buffers refers to the number of buffers identified by src*_buffer_list* and *dst_buffer_list*.

src_offset_list refers to the offsets where to begin copying data from *src_buffer_list*.

dst_offset_list refers to the offsets where to begin copying data into *dst_buffer_list*.

cb refers to the size in bytes to copy.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed.

event returns an event object that identifies this particular broadcast command and can be used to query or queue a wait for this particular command to complete.

The function

```
cl_int clEnqueueAlltoAllBuffer(cl_command_queue * cmd_queue_list,
                               cl_uint num_buffers,
                               cl_mem * src_buffer_list,
                               cl_mem * dst_buffer_list,
                               size_t * src_offset_list,
                               size_t * dst_offset_list,
                               size_t cb,
                               cl_uint num_events_in_wait_list,
                               const cl_event * event_wait_list,
                               cl_event * event)
```

enqueues commands to distribute data from all buffer objects in the list identified by src*_buffer_list* to all buffer objects in the list identified by *dst_buffer_list* in order by index.

cmd_queue_list refers to the command-queues that are associated with the compute devices where the destination buffers identified by *dst_buffer_list* are located.

num_buffers refers to the number of buffers identified by src*_buffer_list* and *dst_buffer_list*.

src_offset_list refers to the offsets where to begin copying data from *src_buffer_list*.

dst_offset_list refers to the offsets where to begin copying data into *dst_buffer_list*.

cb refers to the size in bytes to copy.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed.

event returns an event object that identifies this particular broadcast command and can be used to query or queue a wait for this particular command to complete.

The function

```
cl_int clEnqueueReduceBuffer( cl_command_queue cmd_queue,
                              cl_uint num_src_buffers,
                              cl_mem * src_buffer_list,
                              cl_mem dst_buffer,
                              size_t * src_offset_list,
                              size_t dst_offset,
                              size_t cb,
                              cl_channel_type datatype,
                              cl_uint num_events_in_wait_list,
                              const cl_event * event_wait_list,
                              cl_event * event)
```

enqueues commands to reduce values on all buffer objects in the list identified by src*_buffer_list* and copy the value to the buffer object identified by *dst_buffer.*

cmd_queue refers to the command-queue that is associated with the compute device where the destination buffer identified by *dst_buffer* are located.

num_src_buffers refers to the number of source buffers identified by src*_buffer_list*.

src_offset_list refers to the offsets where to begin copying data from *dst_buffer_list*.

dst_offset refers to the offset where to begin copying data into dst*_buffer*.

cb refers to the size in bytes to copy.

datatype refers to the following built-in types: int, uint, float, and double.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed.

event returns an event object that identifies this particular broadcast command and can be used to query or queue a wait for this particular command to complete.

The function

cl_int **clEnqueueAllReduceBuffer**(cl_command_queue * *cmd_queue_list*,
cl_uint *num_buffers*,
cl_mem * *src_buffer_list*,
cl_mem * *dst_buffer_list*,
size_t * *src_offset_list*,
size_t * *dst_offset_list*,
size_t *cb*,
cl_channel_type *datatype*,
cl_uint *num_events_in_wait_list*,
const cl_event * *event_wait_list*,
cl_event * *event*)

enqueues commands to reduce values on all buffer objects in the list identified by src*_buffer_list* and copy the value to all buffer objects in the list identified by *dst_buffer_list.*

cmd_queue_list refers to the command-queues that are associated with the compute devices where the destination buffers identified by *dst_buffer_list* are located.

num_buffers refers to the number of buffers identified by src*_buffer_list* and *dst_buffer_list*.

src_offset_list refers to the offsets where to begin copying data from *src_buffer_list*.

dst_offset_list refers to the offsets where to begin copying data into *dst_buffer_list*.

cb refers to the size in bytes to copy.

datatype refers to the following built-in types: int, uint, float, and double.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed.

event returns an event object that identifies this particular broadcast command and can be used to query or queue a wait for this particular command to complete.

The function

```
cl_int clEnqueueReduceScatterBuffer( cl_command_queue * cmd_queue_list,
                                     cl_uint num_buffers,
                                     cl_mem * src_buffer_list,
                                     cl_mem * dst_buffer_list,
                                     size_t * src_offset_list,
                                     size_t * dst_offset_list,
                                     size_t cb,
                                     cl_channel_type datatype,
                                     cl_uint num_events_in_wait_list,
                                     const cl_event * event_wait_list,
                                     cl_event * event)
```

enqueues commands to combine values from all buffer objects in the list identified by src*_buffer_list* and scatter the results to all buffer objects in the list identified by *dst_buffer_list.*

cmd_queue_list refers to the command-queues that are associated with the compute devices where the destination buffers identified by *dst_buffer_list* are located.

num_buffers refers to the number of buffers identified by src*_buffer_list* and *dst_buffer_list*.

src_offset_list refers to the offsets where to begin copying data from *src_buffer_list*.

dst_offset_list refers to the offsets where to begin copying data into *dst_buffer_list*.

cb refers to the size in bytes to copy.

datatype refers to the following built-in types: int, uint, float, and double.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed.

event returns an event object that identifies this particular broadcast command and can be used to query or queue a wait for this particular command to complete.

The function

cl_int **clEnqueueScanBuffer**(cl_command_queue * *cmd_queue_list*,
cl_uint *num_buffers*,
cl_mem * *src_buffer_list*,
cl_mem * *dst_buffer_list*,
size_t * *src_offset_list*,
size_t * *dst_offset_list*,
size_t *cb*,
cl_channel_type *datatype*,
cl_uint *num_events_in_wait_list*,
const cl_event * *event_wait_list*,
cl_event * *event*)

enqueues commands to perform an inclusive prefix reduction on data distributed across the buffer objects in the list identified by src*_buffer_list* and copy the results to all buffer objects in the list identified by *dst_buffer_list.*

cmd_queue_list refers to the command-queues that are associated with the compute devices where the destination buffers identified by *dst_buffer_list* are located.

num_buffers refers to the number of buffers identified by src*_buffer_list* and *dst_buffer_list*.

src_offset_list refers to the offsets where to begin copying data from *src_buffer_list*.

dst_offset_list refers to the offsets where to begin copying data into *dst_buffer_list*.

cb refers to the size in bytes to copy.

datatype refers to the following built-in types: int, uint, float, and double.

event_wait_list and *num_events_in_wait_list* specify events that need to complete before this particular command can be executed.

event returns an event object that identifies this particular broadcast command and can be used to query or queue a wait for this particular command to complete.