CUDA / OPENCL - MPI

(Some of my notes on CUDA / OPENCL - MPI for Discovery Cluster Users – email n.roy@neu.edu if you have questions or need more clarifications. Nilay K. Roy, Ph.D)

CUDA will work with several MPI implementations

LAM-MPI, MVAPICH2, OpenMPI, CRAY, IBM Platform MPI and MPICH. Note Discovery Cluster supports only the last two MPI versions.

If you are combining MPI and CUDA, you often need to send GPU buffers instead of host buffers. Without CUDA - MPI, you need to stage GPU buffers through host memory, using cudaMemcpy as below.

```c
//MPI rank 0
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_CHAR,1,100,MPI_COMM_WORLD);

//MPI rank 1
MPI_Recv(r_buf_h,size,MPI_CHAR,0,100,MPI_COMM_WORLD, &status);
cudaMemcpy(r_buf_d,r_buf_h,size,cudaMemcpyHostToDevice);
```

With a CUDA- MPI library this is not necessary; the GPU buffers can be directly passed to MPI as below.

```c
//MPI rank 0
MPI_Send(s_buf_d,size,MPI_CHAR,1,100,MPI_COMM_WORLD);

//MPI rank n-1
MPI_Recv(r_buf_d,size,MPI_CHAR,0,100,MPI_COMM_WORLD, &status);
```

A CUDA- MPI implementation must handle buffers differently depending on whether it resides in host or device memory. An MPI implementation could offer different APIs for host and device buffers, or it could add an additional argument indicating where the passed buffer lives. Fortunately, neither of these approaches is necessary because of the Unified Virtual Addressing (UVA) feature introduced in CUDA 4.0 (on Compute Capability 2.0 and later GPUs). With UVA the host memory and the memory of all GPUs in a system (a single node) are combined into one large (virtual) address space.

![No UVA: Multiple Memory Spaces vs UVA: Single Address Space](image)
With UVA the location of a buffer can be determined based on the MSBs of its address, so there is no need to change the API of MPI.

CUDA - MPI not only makes it easier to work with a CUDA+MPI application, it also makes the application run more efficiently as all operations that are required to carry out the message transfer can be pipelined, and acceleration technologies like GPUDirect can be utilized by the MPI library transparently to the user.

NVIDIA GPUDirect technologies provide high-bandwidth, low-latency communications with NVIDIA GPUs. GPUDirect is an umbrella name used to refer to several specific technologies. In the context of MPI the GPUDirect technologies cover all kinds of inter-rank communication: intranode, inter-node, and RDMA inter-node communication.

The newest GPUDirect feature, introduced with CUDA 5.0, is support for Remote Direct Memory Access (RDMA), with which buffers can be directly sent from the GPU memory to a network adapter without staging through host memory.

Another variant is GPUDirect for Peer-to-Peer (P2P) transfers, which was introduced with CUDA 4.0 and can accelerate intra-node communication. Buffers can be directly copied between the memories of two GPUs in the same system with GPUDirect P2P.

Host memory allocated with malloc is usually pageable, that is, the memory pages associated with the memory can be moved around by the kernel, for example to the swap partition on the hard drive. Memory paging has an impact on copying data by DMA and RDMA. DMA and RDMA transfers work independently of the CPU and thus also independently of the OS kernel, so memory pages must not be moved by the kernel while they are being copied. Inhibiting the movement of memory pages is called memory “pinning”. So memory that cannot be moved by the kernel and thus can be used in DMA and RDMA transfers is called pinned memory. As a
side note, pinned memory can also be used to speed up host-to-device and device-to-host transfer in general.

GPUDirect for accelerated communication with network and storage devices was the first GPUDirect technology, introduced with CUDA 3.1. This feature allows the network fabric driver and the CUDA driver to share a common pinned buffer in order to avoid an unnecessary `memcpy` within host memory between the intermediate pinned buffers of the CUDA driver and the network fabric buffer.

To explain how these acceleration techniques and the necessary intermediate buffers affect communication with MPI consider a simple application with two MPI Ranks. MPI Rank 0 sends a GPU buffer to MPI Rank 1 and MPI Rank 1 receives the message from MPI Rank 0 into a GPU Buffer. So MPI Rank 0 will execute an `MPI_Send`.

```c
MPI_Send(s_buf_d,size,MPI_CHAR,1,100,MPI_COMM_WORLD);
```

MPI Rank 1 will execute an `MPI_Recv`.

```c
MPI_Recv(r_buf_d,size,MPI_CHAR,0,100,MPI_COMM_WORLD,&stat);
```

Depending on the MPI implementation, the message size, the chosen protocol and other factors, the details might differ but the conclusions remain valid.

- A green box is a GPU Buffer
- A blue box is a regular pageable host buffer
- A yellow box is a pinned CUDA buffer in host memory
- A red box is a pinned network Fabric buffer in host memory
- A green arrow is a DMA transfer over the PCI-E bus
- A blue arrow is a regular `memcpy` within host memory
- A red arrow is a RDMA network message.

If GPUDirect RDMA is available the buffer can be directly moved to the network without touching the host memory at all. So the data is directly moved from the buffer in the device memory of MPI Rank 0 to the device memory of MPI Rank 1 with a PCI-E DMA → RDMA
PCI-E DMA sequence as indicated in the picture below by the red arrow with a green outline line.

Depending on the size of the buffer this might be done in chunks and so that we get multiple incarnations of the PCI-E DMA → RDMA → PCI-E DMA sequence.

If MPI rank 0 and MPI rank 1 are running on the same host and using GPUs on the same PCI-E bus GPUDirect P2P can be utilized to achieve a similar result.

If no variant of GPUDirect is available, for example if the network adapter does not support GPUDirect, the situation is a little bit more complicated. The buffer needs to be first moved to the pinned CUDA driver buffer and from there to the pinned buffer of the network fabric in the host memory of MPI Rank 0. After that it can be sent over the network. On the receiving MPI Rank 1 these steps need to be carried out in reverse.

Although this involves multiple memory transfers, the execution time for many of them can be hidden by executing the PCI-E DMA transfers, the host memory copies and the network transfers in a pipelined fashion as shown below.
If GPUDirect accelerated communication with network and storage devices is available, the memory copy between the pinned CUDA buffer and the network fabric buffer can be eliminated.

In contrast, if a non-CUDA-aware MPI implementation is used, the programmer has to take care of staging the data through host memory, by executing the following sequence of calls.

MPI Rank 0 will execute a `cudaMemcpy` from device to host followed by `MPI_Send`.

```c
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_CHAR,1,100,MPI_COMM_WORLD);
```

MPI Rank 1 will execute `MPI_Recv` followed by a `cudaMemcpy` from device to host.

```c
MPI_Recv(r_buf_h,size,MPI_CHAR,0,100,MPI_COMM_WORLD,&stat);
cudaMemcpy(r_buf_d,r_buf_h,size,cudaMemcpyHostToDevice);
```

This will not only introduce an additional memory copy within each node's host memory, but it will also stall the pipeline after the first `cudaMemcpy` and after the `MPI_Recv` on MPI rank 1, so the execution time will be much longer as visualized in the diagram below.

A user of a non-CUDA-MPI library could implement a more efficient pipeline using CUDA streams and asynchronous memory copies to speed up the communication. A CUDA-MPI implementation can more efficiently exploit the underlying protocol and can automatically utilize the GPUDirect acceleration technologies.
AN EXAMPLE CUDA – MPI RUN on the Discovery Cluster:

Below are the two source files:

1) mpi.c

```c
[roy@discovery2 cuda-mpi12]$ pwd
/home/roy/cuda-mpi12
[roy@discovery2 cuda-mpi12]$ cat mpi.c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <mpi.h>

#define MAX NODES 100
#define BUFF_LEN 256

extern void enumCudaDevices(char *buff);

int main(int argc, char *argv[])
{
    int i, myrank, nprocs;
    char pName[MAX_PROCESSOR_NAME], buff[BUFF_LEN];

    MPI_Init(&argc, &argv);
    MPI_Comm size(MPI_COMM_WORLD, nprocs);
    MPI_Comm rank(MPI_COMM_WORLD, myrank);
    MPI_Get_processor_name(pName, &i);

    sprintf(buff, "-%s %d", pName, myrank);
    // Find local CUDA devices
    enumCudaDevices(buff);
    // Collect and print the list of CUDA devices from all MPI processes
    if (myrank == 0)
    {
        char devList[MAX NODES][BUFF_LEN];
        MPI_Gather(buff, BUFF_LEN, MPI_CHAR, devList, BUFF_LEN, MPI_CHAR, 0, MPI_COMM_WORLD);
        for (i = 0; i < nprocs; i++)
        {
            printf("%s", devList + i);
        }
    }
    else
    {
        MPI_Gather(buff, BUFF_LEN, MPI_CHAR, NULL, 0, MPI_CHAR, 0, MPI_COMM_WORLD);
        MPI_Finalize();
    }
    return 0;
}
```

2) cuda.cu

```c
[roy@discovery2 cuda-mpi12]$ cat cuda.cu
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <cuda.h>

#define MAX NODES 100
#define BUFF_LEN 256

extern "C" void enumCudaDevices(char *buff);

void enumCudaDevices(char *buff)
{
    char countBuff[BUFF_LEN];
    int i, devCount;

    cudaGetDeviceCount(&devCount);
    sprintf(countBuff, "%d", devCount);
    strncat(countBuff, buff, BUFF_LEN);
    for (i = 0; i < devCount; i++)
    {
        cudaGetDeviceProp devProp;
        cudaGetDeviceProperties(&devProp, i);
        sprintf(buff, "%s", i, devProp.name);
        strncat(buff, countBuff, BUFF_LEN);
    }
}
```

Modules required to run this, and the submit script are shown below:

```
[nroy@discovery2 cuda-mpi2]$ module list
Currently Loaded Module files:
  1) gnu-4.4-compilers  2) fftw-3.3.3   3) platform-mpi   4) cuda-6.0

[nroy@discovery2 cuda-mpi2]$ cat bsubmit.bash
#!/bin/sh
#BSUB -J NKR_CUDA_MPI_SAMPLE_1
#BSUB -o output_file
#BSUB -e error_file
#BSUB -n 5
#BSUB -q par-gpu
#BSUB -R 'span[ptile=1]'
#BSUB -cwd /home/nroy/cuda-mpi2

work=/home/nroy/cuda-mpi2

cd $work

mpirun -np 5 -prot TCP -lsf sample-cuda-mpi
```

```
[nroy@discovery2 cuda-mpi2]$ module whatis cuda-6.0
cuda-6.0 : loads the modules environment for cuda-6.0 compilers, libraries, and include files.

Needs the following modules to be loaded as prerequisites

module load gnu-4.4-compilers
module load fftw-3.3.3
module load platform-mpi

Put these module load commands in your .bashrc file that is found in your /home/<user-id> directory.
```

Submit and run is shown below:

```
[nroy@discovery2 cuda-mpi2]$ bsub < bsubmit.bash
[nroy@discovery2 cuda-mpi2]$ bjobs

<table>
<thead>
<tr>
<th>JOBID</th>
<th>USER</th>
<th>STAT</th>
<th>QUEUE</th>
<th>FROM_HOST</th>
<th>EXEC_HOST</th>
<th>JOB_NAME</th>
<th>SUBMIT_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>200976</td>
<td>nroy</td>
<td>RUN</td>
<td>par-gpu</td>
<td>discovery2</td>
<td>compute-2-129</td>
<td>1*SAMPLE_1 Jun 19 13:37</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1*compute-2-129</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1*compute-2-150</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1*compute-2-152</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1*compute-2-153</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
[nroy@discovery2 cuda-mpi2]$ bjobs
```

Compilation is shown below – this is a simple example but the recommended method is to use a “makefile”.

```
[nroy@discovery2 cuda-mpi2]$ nvcc -c cuda.cu -o cuda.o
[nroy@discovery2 cuda-mpi2]$ mpicc -c mpi.c -o mpi.o
[nroy@discovery2 cuda-mpi2]$ mpicc mpi.o cuda.o -L/shared/apps/cuda6.0/lib64 -lcudart -o sample-cuda-mpi
```

Output from run is shown below:

```
[nroy@discovery2 cuda-mpi2]$ cat error_file
[nroy@discovery2 cuda-mpi2]$ cat output_file
```

Sender: LSF System <lsfadmin@compute-2-128>
Subject: Job 200976: <NKR_CUDA_MPI_SAMPLE_1> in cluster <mghpcc_cluster1> Done

Job <NKR_CUDA_MPI_SAMPLE_1> was submitted from host <discovery2> by user <nroy> in cluster <mghpcc_cluster1>. Job was executed on host(s) <1*compute-2-129>, in queue <par-gpu>, as user <nroy> in cluster <mghpcc_cluster1>.

<1*compute-2-150>
<1*compute-2-152>
<1*compute-2-153>

</home/nroy> was used as the home directory,
</home/nroy/cuda-mpi2> was used as the working directory.
Started at Thu Jun 19 05:32:26 2014
Results reported at Thu Jun 19 05:32:46 2014

Your job looked like:

------------------------------------------------------------
# LSATCH: User input
#!/bin/sh
#BSUB -J NKR_CUDA_MPI_SAMPLE_1
#BSUB -o output_file
#BSUB -e error_file
#BSUB -n 5
#BSUB -q par-gpu
#BSUB -R "span[ptile=1]"
#BSUB -cwd /home/nroy/cuda_mpi2

work=/home/nroy/cuda_mpi2

cd $work

mpirun -np 5 -prot-TCP -lsf sample-cuda-mpi

------------------------------------------------------------

Successfully completed.

Resource usage summary:

CPU time : 8.49 sec.
Max Memory : 10 MB
Average Memory : 7.50 MB
Total Requested Memory : -
Delta Memory : -
(Delta: the difference between total requested memory and actual max usage.)
Max Swap : 51 MB

Max Processes : 2
Max Threads : 2

The output (if any) follows:

Host 0 -- ip 10.100.8.168 -- ranks 0
Host 1 -- ip 10.100.8.169 -- ranks 1
Host 2 -- ip 10.100.8.190 -- ranks 2
Host 3 -- ip 10.100.8.192 -- ranks 3
Host 4 -- ip 10.100.8.193 -- ranks 4

host | 0  1  2  3  4
======|==========================
0 : SHM TCP TCP TCP TCP
1 : TCP SHM TCP TCP TCP
2 : TCP SHM TCP TCP TCP
3 : TCP SHM TCP SHM TCP
4 : TCP TCP TCP TCP SHM

Prot - All Intra-node communication is: SHM
Prot - All Inter-node communication is: TCP

compute-2-128 0 1 0:Tesla K20m
compute-2-129 1 1 0:Tesla K20m
compute-2-150 2 1 0:Tesla K20m
compute-2-152 3 1 0:Tesla K20m
compute-2-153 4 1 0:Tesla K20m

PS:

Read file <error_file> for stderr output of this job.

[nroy@discovery2 cuda_mpi2]$
What did this code do?
It used MPI threads to enumerate GPU’s on each of the five GPU servers in the par-gpu queues. Now we can get each thread to work on the GPU they enumerate. In this way one achieves MPI parallelization across nodes where each node has one or more GPU’s and one can use this GPU to do work.

**OPENCL with MPI:**

Now let us look at an example with OPENCL and MPI. The example files run on the Discovery Cluster from my account is [here](http://nuweb12.neu.edu/re/wp-content/uploads/2014/06/nilay_roy-mpi-opencl-example.tar.gz). You will need the following modules loaded to compile and run it.

```bash
[nroy@discovery2 bin]$ pwd
/home/nroy/opencl_example/het-comp-MPI-OpenCL/het-comp-MPI-OpenCL/bin
[nroy@discovery2 bin]$ ls -la
total 527
drwxrwx-- 2 nroy nroy 165 Jun 25 16:03 .
drwxrwx-- 7 nroy nroy 164 Jun 25 16:04 ..
-rw-rw-r-- 1 nroy nroy 119058 Jun 25 15:44 hybridPrefixSum
-rw-rw-r-- 1 nroy nroy 119058 Jun 25 15:44 hybridPrefixSum

[nroy@discovery2 bin]$ module list
Currently Loaded Modulefiles:
1) gnu-4.4-compilers   2) fftw-3.3.3          3) platform -mpi        4) cuda-6.0            5) AMD-APP-SDK-v2.9
[nroy@discovery2 bin]$ module list
```

The submit script is all set to use the “par-gpu” queue that submits to the 32 nodes with a NVIDIA Kepler K20 m GPU each.

The LSF submit script is shown below:

```bash
[nroy@discovery2 bin]$ cat mpi_opencl_job1.bash
#!/bin/sh
#BSUB -J MPI_OPENCL_JOB.1
#BSUB -o output_file
#BSUB -e error_file
#BSUB -n 64
#BSUB -q par-gpu
#BSUB -R "span[ptile=32]
#BSUB -cwd /home/nroy/opencl_example/het-comp-MPI-OpenCL/het-comp-MPI-OpenCL/bin

######## THIS IS A TEMPLATE FILE FOR TCP ENABLED MPI RUNS ON THE DISCOVERY CLUSTER ########

###### #BSUB -n has a value equal to the given value for the -np option ####
# prefix for next run is entered below

# file staging code is entered below

#### Enter your working directory below - this is the string returned from issuing the command
#### "pwd"
#### IF you stage your files this is your run directory in the high speed scratch space mounted
#### across all compute nodes
work=/home/nroy/opencl_example/het-comp-MPI-OpenCL/het-comp-MPI-OpenCL/bin
```
Configure the “Makefile and “make.inc” file and in the same directory enter make to compile the code. Then put the right paths in the submit file above and run using “bsub < name_of_LSF_submit_script”.

The output is shown below and we shall briefly look at the MPI / OPENCL hybrid code following that. The “error_file” is empty as expected.

```
[nroy@discovery2 bin]$ cat output_file
Sender: LSF System <lsfadmin@compute-2-138>
Subject: Job 202322: <MPI_OPENCL_JOB.1> in cluster <mghpcc_cluster1> Done

Job <MPI_OPENCL_JOB.1> was submitted from host <discovery2> by user <nroy> in cluster <mghpcc_cluster1>.
Job was executed on host(s) <32*compute-2-138>, in queue <par-gpu>, as user <nroy> in cluster <mghpcc_cluster1>.
<32*compute-2-139> <32*compute-2-139>
</home/nroy> was used as the home directory.
</home/nroy/opencl_example/hetr-comp-MPI-OpenCL/hetr-comp-MPI-OpenCL/bin> was used as the working directory.
Started at Wed Jun 25 07:59:09 2014
Results reported at Wed Jun 25 08:00:17 2014

Your job looked like:
```

```
```
#BSUB -n has a value equal to the given value for the -np option

# prefix for next run is entered below

# file staging code is entered below

#### Enter your working directory below - this is the string returned from issuing the command
#### "pwd"
#### IF you stage your files this is your run directory in the high speed scratch space mounted across all compute nodes
work=/home/nroy/opencl_example/hetr-comp-MPI-OpenCL/hetr-comp-MPI-OpenCL/bin

cd $work
tempfile1=hostlistrun
tempfile2=hostlist-tcp
echo $LSB_MCPU_HOSTS > $tempfile1
declare -a hosts
read -a hosts < ${tempfile1}
for ((i=0; i<${#hosts[@]}; i += 2)) ;
do
HOST=${hosts[$i]}
CORE=${hosts[(($i+1))]};
echo $HOST:$CORE >> $tempfile2
done

mpirun -np 64 -prot -TCP -hostfile $work/$tempfile2 ./hybridPrefixSum 10000

# any clean up tasks and file migration code is entered below

(... more ...)

Successfully completed.

Resource usage summary:

| CPU time    | 185.29 sec. |
| Max Memory  | 6 MB        |
| Average Memory | 5.20 MB |
| Total Requested Memory | - |
| Delta Memory | - |
| Max Swap    | 750 MB      |
| Max Processes | 38 |
| Max Threads | 8 |

The output (if any) follows:

./hybridPrefixSum:/usr/lib64/libOpenCL.so.1: no version information available (required by ./hybridPrefixSum)
./hybridPrefixSum:/usr/lib64/libOpenCL.so.1: no version information available (required by ./hybridPrefixSum)
./hybridPrefixSum:/usr/lib64/libOpenCL.so.1: no version information available (required by ./hybridPrefixSum)
./hybridPrefixSum:/usr/lib64/libOpenCL.so.1: no version information available (required by ./hybridPrefixSum)
./hybridPrefixSum:/usr/lib64/libOpenCL.so.1: no version information available (required by ./hybridPrefixSum)
./hybridPrefixSum:/usr/lib64/libOpenCL.so.1: no version information available (required by ./hybridPrefixSum)
./hybridPrefixSum:/usr/lib64/libOpenCL.so.1: no version information available (required by ./hybridPrefixSum)
./hybridPrefixSum:/usr/lib64/libOpenCL.so.1: no version information available (required by ./hybridPrefixSum)
./hybridPrefixSum:/usr/lib64/libOpenCL.so.1: no version information available (required by ./hybridPrefixSum)
./hybridPrefixSum:/usr/lib64/libOpenCL.so.1: no version information available (required by ./hybridPrefixSum)
Host 0 -- ip 10.100.8.178 -- ranks 0 - 31
Host 1 -- ip 10.100.8.179 -- ranks 32 - 63

host | 0    1
-----|======
0 : SHM  TCP
1 : TCP  SHM

Prot - All Intra-node communication is: SHM
Prot - All Inter-node communication is: TCP

Input array to calculate prefix sum value on : process 1
3 6 5 3 5 6 2 9 1

Input array to calculate prefix sum value on : process 2
3 4 9 4 1 9 5 7 4 1

Input array to calculate prefix sum value on : process 3
6 7 6 9 7 6 0 9 0 7

Input array to calculate prefix sum value on : process 4
7 1 6 2 0 4 8 2 2 3

Input array to calculate prefix sum value on : process 5
2 2 0 7 1 5 6 3 2 1

Input array to calculate prefix sum value on : process 6
3 8 1 6 4 9 2 7 6

Input array to calculate prefix sum value on : process 7
9 3 0 3 5 7 6 4 1

Input array to calculate prefix sum value on : process 8
5 6 0 4 6 0 9 4 6

Input array to calculate prefix sum value on : process 9
2 1 9 0 1 2 8 3 2 9

Input array to calculate prefix sum value on : process 10
8 3 0 2 5 2 9 9 4 2

Input array to calculate prefix sum value on : process 11
7 0 6 7 2 2 6 9 6 2

Input array to calculate prefix sum value on : process 12
5 1 3 1 2 0 1 7 4 2
Input array to calculate prefix sum value on : process 13
3 8 9 8 7 3 2 3 0 3

Input array to calculate prefix sum value on : process 14
9 1 3 9 6 1 1 4 9

Input array to calculate prefix sum value on : process 15
0 7 2 0 2 1 5 8 6 5

Input array to calculate prefix sum value on : process 16
6 3 0 5 5 9 4 6 8 0

Input array to calculate prefix sum value on : process 17
6 0 2 5 1 6 2 0 9 0

Input array to calculate prefix sum value on : process 18
0 2 4 8 5 2 5 9 5 3

Input array to calculate prefix sum value on : process 19
2 7 8 1 1 1 3 5 5 2

Input array to calculate prefix sum value on : process 20
8 8 9 5 2 4 4 3 7 5

Input array to calculate prefix sum value on : process 21
6 4 3 5 8 4 9 2 0

Input array to calculate prefix sum value on : process 22
1 3 7 3 9 4 0 8 9

Input array to calculate prefix sum value on : process 23
9 5 0 3 1 7 3 1 2 0

Input array to calculate prefix sum value on : process 24
3 5 3 4 9 2 2 9 3

Input array to calculate prefix sum value on : process 25
3 8 9 2 6 7 9 1 1

Input array to calculate prefix sum value on : process 26
9 9 1 6 7 0 0 2 6 4

Input array to calculate prefix sum value on : process 27
0 3 0 2 7 5 4 4 4 5

Input array to calculate prefix sum value on : process 28
1 5 1 5 9 2 0 8 9 6

Input array to calculate prefix sum value on : process 29
7 4 8 3 8 7 2 7 9

Input array to calculate prefix sum value on : process 30
2 9 2 3 0 7 1 0 9 4

Input array to calculate prefix sum value on : process 31
4 1 8 9 7 2 2 0 1 9

Input array to calculate prefix sum value on : process 32
6 0 2 1 2 3 7 8 5 3

Input array to calculate prefix sum value on : process 33
2 9 4 6 7 5 6 0 3 4

Input array to calculate prefix sum value on : process 34
4 3 7 7 7 3 7 4 4 0

Input array to calculate prefix sum value on : process 35
1 9 3 0 3 9 4 7 2 9
Input array to calculate prefix sum value on: process 36
1 8 1 6 5 4 0 1 8 3

Input array to calculate prefix sum value on: process 37
0 1 8 3 0 7 4 8 8 2

Input array to calculate prefix sum value on: process 38
3 4 5 6 2 5 1 1 2 1

Input array to calculate prefix sum value on: process 39
8 0 3 2 3 5 0 1 6 6

Input array to calculate prefix sum value on: process 40
7 2 5 8 8 5 7 3 7 0

Input array to calculate prefix sum value on: process 41
1 7 2 6 6 0 1 6 3 7

Input array to calculate prefix sum value on: process 42
2 6 0 3 6 3 8 6 4 1

Input array to calculate prefix sum value on: process 43
5 1 5 1 0 1 6 6 9 6

Input array to calculate prefix sum value on: process 44
0 4 1 4 5 9 6 9 2 8

Input array to calculate prefix sum value on: process 45
9 1 4 5 4 8 8 5 4 1

Input array to calculate prefix sum value on: process 46
7 0 5 9 5 3 8 5 5 6

Input array to calculate prefix sum value on: process 47
3 7 4 2 4 8 3 7 6 9

Input array to calculate prefix sum value on: process 48
2 1 8 8 1 5 7 6 3 2

Input array to calculate prefix sum value on: process 49
4 6 9 3 0 4 1 9 7 9

Input array to calculate prefix sum value on: process 50
4 2 7 0 9 7 4 9 2 5

Input array to calculate prefix sum value on: process 51
2 5 3 9 6 1 3 1 1 3

Input array to calculate prefix sum value on: process 52
0 9 3 3 2 5 3 2 0 0

Input array to calculate prefix sum value on: process 53
6 3 9 3 0 2 0 4 0 7

Input array to calculate prefix sum value on: process 54
5 7 6 5 3 6 3 9 7 3

Input array to calculate prefix sum value on: process 55
3 4 4 0 4 2 2 3 5 4

Input array to calculate prefix sum value on: process 56
4 3 0 0 7 6 9 5 7 3

Input array to calculate prefix sum value on: process 57
5 6 5 8 1 5 3 6 2

Input array to calculate prefix sum value on: process 58
2 4 6 6 8 1 1 8 5 2
Input array to calculate prefix sum value on : process 59
5 2 0 2 0 3 1 5 6 2

Input array to calculate prefix sum value on : process 60
0 1 6 0 6 0 1 8 4 0

Input array to calculate prefix sum value on : process 61
9 4 2 8 7 5 5 4 4

Input array to calculate prefix sum value on : process 62
2 0 9 9 6 5 2 5 7 2

Input array to calculate prefix sum value on : process 63
5 6 4 1 2 8 5 8 1

Processor Name : compute-2-138
Device Name : Tesla K20m
Output prefix sum value : process 1
3 9 16 21 24 29 35 37 46 47

Processor Name : compute-2-138
Device Name : Tesla K20m
Output prefix sum value : process 2
3 7 16 20 21 20 35 42 46 47

Processor Name : compute-2-138
Device Name : Tesla K20m
Output prefix sum value : process 3
6 13 19 28 35 41 41 50 50 57

Processor Name : compute-2-138
Device Name : Tesla K20m
Output prefix sum value : process 4
7 8 14 16 16 20 28 30 32 35

Processor Name : compute-2-138
Device Name : Tesla K20m
Output prefix sum value : process 5
2 4 4 11 12 17 23 26 28 29

Processor Name : compute-2-138
Device Name : Tesla K20m
Output prefix sum value : process 6
3 11 12 13 19 23 32 34 41 47

Processor Name : compute-2-138
Device Name : Tesla K20m
Output prefix sum value : process 7
9 17 20 20 23 28 35 41 45 46

Processor Name : compute-2-138
Device Name : Tesla K20m
Output prefix sum value : process 8
5 11 11 15 21 21 27 36 40 46

Processor Name : compute-2-138
Device Name : Tesla K20m
Output prefix sum value : process 9
2 3 12 12 13 15 23 26 28 37

Processor Name : compute-2-138
Device Name : Tesla K20m
Output prefix sum value : process 10
8 11 11 13 18 20 29 38 42 44

Processor Name : compute-2-138
Device Name : Tesla K20m
Output prefix sum value : process 11
7 7 13 20 22 24 30 39 45 47
Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 12
5 6 9 10 12 12 13 20 24 26

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 13
3 11 20 28 35 38 40 43 43 46

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 14
9 10 13 22 31 37 38 39 43 52

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 15
0 7 9 9 11 12 17 25 31 36

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 16
6 9 9 14 19 28 32 38 46 46

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 17
6 6 8 13 14 20 22 22 31 31

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 18
0 2 6 14 19 21 26 35 40 43

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 19
2 9 17 18 19 23 28 33 35

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 20
8 16 25 30 32 36 40 43 50 55

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 21
6 10 13 18 23 31 35 44 46 46

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 22
1 4 11 14 17 26 30 30 38 47

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 23
9 14 14 17 18 25 28 29 31 31

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 24
3 8 13 16 20 29 31 33 42 45

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 25
3 11 14 23 25 31 38 47 48 49
Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 26
9 18 19 25 32 32 32 34 40 44

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 27
0 3 3 5 12 17 21 25 29 34

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 28
1 6 7 12 21 23 23 31 40 46

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 29
7 11 15 23 26 34 41 43 50 59

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 30
2 11 13 16 16 23 24 24 33 37

Processor Name: compute-2-138
Device Name: Tesla K20m
Output prefix sum value: process 31
4 5 13 22 29 31 33 33 34 43

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 32
6 8 9 11 14 21 29 34 37

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 33
2 11 15 21 28 33 39 39 42 46

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 34
4 7 14 21 28 31 38 42 46 46

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 35
1 10 13 16 25 29 36 38 47

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 36
1 9 10 16 21 25 25 26 34 37

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 37
0 1 9 12 12 19 23 31 39 41

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 38
3 7 12 18 20 25 26 27 29 30

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 39
8 8 11 13 16 21 22 28 34
Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 40
7 9 14 22 30 35 42 45 52 52

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 41
1 8 10 16 22 22 23 29 32 39

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 42
2 8 11 17 20 28 34 38 39

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 43
5 6 11 12 12 13 19 25 34 40

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 44
0 4 5 9 14 23 29 38 40 48

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 45
9 10 14 19 23 31 39 44 48 49

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 46
7 7 12 21 26 29 37 42 47 53

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 47
3 10 14 16 20 28 31 38 44 53

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 48
2 3 11 19 20 25 32 38 41 43

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 49
4 10 19 22 26 27 8 6 43 52

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 50
4 6 13 13 22 29 33 42 44 49

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 51
2 7 10 19 25 26 29 30 31 34

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 52
0 9 12 15 17 22 25 27 27 27

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 53
6 9 18 21 21 23 27 27 33 34
Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 54
5 12 18 23 26 32 35 44 51 54

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 55
3 7 11 15 17 19 22 27 31

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 56
4 7 7 14 20 29 34 41 44

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 57
5 11 16 21 29 30 35 38 44 46

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 58
2 6 12 18 26 27 28 36 41 43

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 59
5 7 9 9 12 13 18 24 26

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 60
0 1 7 7 13 14 22 26 26

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 61
9 13 15 23 30 35 40 45 49 53

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 62
2 2 11 20 26 31 33 38 45 47

Processor Name: compute-2-139
Device Name: Tesla K20m
Output prefix sum value: process 63
5 11 15 16 17 19 27 32 40 41

PS:
Read file <error_file> for stderr output of this job.

[nroy@discovery2 bin]$ 

The code does the following: given total number of arrays binding to the number of processes spawned in the MPI program, each MPI process rank reads the input array and OpenCL kernel performs prefix sum of an array on each GPU associated with MPI process rank. The input to the problem is given as arguments in the command line. Each Process prints the final prefix sum of an array. This is the basis for extending the problem to do DOT product multiplication, VECTOR multiplication, VECTOR MATRIX multiplication and MATRIX MULTIPLICATION, DIAGONALIZATION, and EIGEN MATRIX extraction to name a few implementations.
The source code is shown below:

```cpp
#include<mpiOcl.h>

// define relative kernel path with respective package bin dir so that
// at the time of executable running, the kernel can be found at src dir.
#define KERNEL_SOURCE_PATH "/PrefixSum_kernel.cl"

// @brief       Subroutine to fill input array with random input value.
// @param[out]  hInArray Input array, to be filled with initial random values.
// @param[in]   width   Length of input vector.
// @return      On successful execution this subroutine return void.
void fillInArray(cl_int *hInArray, size_t length)
{
    for(size_t count=0; count< length; count++)
    hInArray[count] = rand()%10;
}

int main(int argc, char* argv[])
{
    exeEnvParams exeEnvParamList;
    cl_int status = CL_SUCCESS;

    exeEnvParamList.maxNumOfPlatforms = exeEnvParamList.maxNumOfDevices =
    exeEnvParamList.deviceType = CL_DEVICE_TYPE_ALL;
    exeEnvParamList.myProcID = myProcRank;
    exeEnvParamList.kernelBuildSpec = BUILD_USER_DEF_KERNEL;
    strcpy(exeEnvParamList.pathToSrc,KERNEL_SOURCE_PATH);
    setExeEnv(&exeEnvParamList);

    cl_device_id attachedDevWithQueue;
    status = clGetCommandQueueInfo( exeEnvParamList.queue, CL_QUEUE_DEVICE,
    sizeof(cl_device_id)&attachedDevWithQueue,NULL);
    // get device specific information
    cl_device_type attachedDevTypeWithQueue;
    status = clGetDeviceInfo(attachedDevWithQueue,CL_DEVICE_NAME,sizeof(char) * deviceNameLen,deviceName,NULL);

    // create kernel handle
    cl_kernel prefixSum_kernel;
    prefixSum_kernel = clCreateKernel( exeEnvParamList.hProgram, "prefixSum_kernel", &status);
    STATUSCHKMSG("kernel handle");

    // create input array
    cl_mem dInArray = clCreateBuffer(exeEnvParamList.context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, length *
    sizeof(cl_int), (void*)hInArray, &status);
    STATUSCHKMSG("memory allocation ");

    // create output buffer
    cl_mem dOutArray = clCreateBuffer(exeEnvParamList.context, CL_MEM_WRITE_ONLY | CL_MEM_COPY_HOST_PTR, length *
    sizeof(cl_int),(void*)hOutArray, &status);
    STATUSCHKMSG("o/p memory allocation ");

    // create space for row and col argument
    cl_int hLength = length;
    cl_mem dLength = clCreateBuffer(exeEnvParamList.context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
    sizeof(cl_int),(void*)&hLength, &status);
    STATUSCHKMSG("scaler memory setting");
```
// create kernel argument
// the input array
status = clSetKernelArg(prefixSum_kernel, 0, sizeof(cl_mem), (void*) &dInArray);
STATUSCHKMSG("in arg setting");
// output array
status = clSetKernelArg(prefixSum_kernel, 1, sizeof(cl_mem), (void*) &dOutArray);
STATUSCHKMSG("out arg set");
// scalar value to be multiplied
status = clSetKernelArg(prefixSum_kernel, 2, sizeof(cl_mem), (void*) &dLength);
STATUSCHKMSG("scalar value argument");

// enqueue a kernel run call
size_t globalThreads[] = {NUMTHREAD};       // number of global items in work dimension
size_t localThreads[] = {GROUP_SIZE};  // number of work item per group
status = clEnqueueNDRangeKernel(exeEnvParamList.queue, prefixSum_kernel, 1, NULL, globalThreads, localThreads, 0, NULL, NULL);
STATUSCHKMSG("kernel enqueue");
status = clFinish(exeEnvParamList.queue);
STATUSCHKMSG("clFinish");

// read output result
status = clEnqueueReadBuffer(exeEnvParamList.queue, dOutArray, CL_TRUE, 0, length * sizeof(cl_int), hOutArray, 0, NULL, &events[0]);
STATUSCHKMSG("read output");
// wait for read buffer to complete the read of output produce by kernel
status = clWaitForEvents(1, &events[0]);
STATUSCHKMSG("read event not completed");
cReleaseEvent(events[0]);
cReleaseKernel(prefixSum_kernel);
cReleaseProgram(exeEnvParamList.hProgram);
cReleaseCommandQueue(exeEnvParamList.queue);
cReleaseContext(exeEnvParamList.context);
}
// end of open_cl_routine

// please modify "ARRAY_LENGTH" as per user requirement. "ARRAY_LENGTH" specify the input array length,
// on which parallel prefix sum operation operation will be performed.
#define ARRAY_LENGTH 1000

int main(int argc, char *argv[])
{
    int root = 0, myrank, numprocs, source, dest, iproc, count;
    int destTag, sourceTag;
    MPI::Status status;

    int hInArrayLen = ARRAY_LENGTH;
    int *hInArray = new int[hInArrayLen];
    int *hOutArray = new int[hInArrayLen];
    char procName[BUFFER_SIZE];
    int procNameLen = BUFFER_SIZE;
    char deviceName[BUFFER_SIZE];
    char deviceNameLen = BUFFER_SIZE;

    //......MPI Initialization......*/

    MPI::Init(argc, argv);
    numprocs = MPI::COMM_WORLD.Get_size();
    myrank = MPI::COMM_WORLD.Get_rank();

    if(myrank == 0){
        for(iproc = 0; iproc < numprocs; iproc++)
        {
            dest = iproc;
            destTag = 0;
            fillInArray(&(cl_int*) hInArray, size_t hInArrayLen);
            cout<<"hIn array to calculate prefix sum value on : process "]<<dest<<endl;
            for(size_t count = 0; count < 10; count++)
            {
                cout<<""<<hInArray[count];
            }
        }
    }

    //......MPI Initialization......*/
Instead of using the NVIDIA CUDA SDK we have used the OPENCL SDK. It is best to compare and contrast speeds before deciding on one of the other methods to exploit GPU cores. KRONOS that maintains OPENCL recommends it to CUDA as it (OPENCL) is NVIDIA device independent. OPENCL code will run on any GPU including INTEL XEON PHI’s. On the other hand NVIDIA has removed the OPENCL SDK and profiler from CUDA SDK and claims it provides customized optimized routines like CUDA BLAS for the developer. Although there is OPENCL BLAS library available too.

The ultimate choice is yours.

You can also use OpenCL applications on multi-core CPU’s in addition to GPU’s. Several parameters affect OpenCL application performance. They include thread scheduling, instruction-level parallelism, datatransfer, data locality, and compiler auto vectorization. OpenCL programmers can explicitly set workgroup size, or let the OpenCL implementation decide it. OpenCL assumes a distributed memory system for its target, a system where communication between host and compute devices is performed explicitly by a system net-work, such as PCI-Express. But, the assumption of discrete memory systems is not true when we use CPUs as compute devices for kernel execution. The host and the compute devices share the same memory system resources such as last-level cache, on-chip interconnection, memory controllers, and
DRAMs. The drawback of disjoint memory address space is that it requires explicit data transfer between the host and compute devices for kernel execution. In common OpenCL applications, the data should be transferred back and forth in order to be processed by the host or device, which becomes unnecessary when we only use the host for computation. OpenCL also provides the programmer many types of memory object allocation flags. OpenCL also provides different APIs for data transfer between the host and compute devices. However unlike OpenMP it does not provide CPU affinity. An OpenCL workitem is a logical thread, which is not tightly coupled with a physical thread even though most parallel programming languages provide this feature. The reason for the lack of this functionality is that the OpenCL design philosophy emphasizes portability over efficiency.

All our compute nodes support OpenCL including the GPU nodes. The same modules shown and used above for the CUDA/OPENCL MPI examples are needed.

Below is example OPENCL code and output on a node that has no GPU to detect CPU. Makefile is similar to the above examples as are the linker flags. Compare the output to one that has a GPU shown later below.

[Below are the modules and OPENCL source code for CPU detection]

```c
#include <iostream>
#include <CL/cl.h>

int main(void)
{
  // define host and device data structures
  cl_platform_id *platform;
  cl_device_id *devices;
  cl_uint num_devices;
  // Identify a platform
  cl_uint num_entries = 2; // maximum number of platforms we are interested in detecting
  cl_uint num_platforms = 0; // number of platforms actually found during run-time
  cl_int err = clGetPlatformIDs(num_entries, &platform, &num_platforms);
  if(err < 0) {
    std::cout << "Couldn't find any platforms\n";
    exit(1);
  }
  std::cout << "Detected " << num_platforms << " platforms\n";
  std::cout << "Using OpenCL to get more information about the platform:\n";
  char pform_name[40];        // platform name
  char pform_vendor[40];        // platform vendor
  char pform_version[40];       // platform version
  char pform_profile[40];       // platform profile
  char pform_extensions[4096];  // platform extensions
  clGetPlatformInfo(platform, CL_PLATFORM_NAME, sizeof(pform_name), &pform_name, NULL);
  clGetPlatformInfo(platform, CL_PLATFORM_VENDOR, sizeof(pform_vendor), &pform_vendor, NULL);
  clGetPlatformInfo(platform, CL_PLATFORM_VERSION, sizeof(pform_version), &pform_version, NULL);
  clGetPlatformInfo(platform, CL_PLATFORM_PROFILE, sizeof(pform_profile), &pform_profile, NULL);
  clGetPlatformInfo(platform, CL_PLATFORM_EXTENSIONS, sizeof(pform_extensions), &pform_extensions, NULL);

  std::cout << "CL_PLATFORM_NAME — " << pform_name << std::endl;
  std::cout << "CL_PLATFORM_VENDOR — " << pform_vendor << std::endl;
  std::cout << "CL_PLATFORM_VERSION — " << pform_version << std::endl;
  std::cout << "CL_PLATFORM_PROFILE — " << pform_profile << std::endl;
}
```

std::cout << "CL_PLATFORM_EXTENSIONS --- " << pform_extensions << std::endl;

// Determine number of connected devices
clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 1, NULL, &num_devices);

// Access connected devices
devices = (cl_device_id*) malloc(sizeof(cl_device_id) * num_devices);
clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, num_devices, devices, NULL);

// Extension data
char ext_data[4096];
char name_data[48];
cl_uint deviceAvl;
cl_uint deviceAdd;
cl_ulong globalMem;
cl_ulong blockSize;
cl_ulong sharedMem;
cl_uint witemDims;
cl_ulong xyzDims[3];
char deviceExt[4096];

// Obtain specifications for each connected device
for(unsigned int i=0; i<num_devices; i++) {
    clGetDeviceInfo(devices[i], CL_DEVICE_NAME                      , sizeof(name_data), name_data  , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_AVAILABLE                 , sizeof(ext_data) , &deviceAvl , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_ADDRESS_BITS              , sizeof(ext_data) , &deviceAdd , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_MAX_CLOCK_FREQUENCY       , sizeof(ext_data) , &frequency , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_GLOBAL_MEM_SIZE           , sizeof(ext_data) , &globalMem , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_MAX_WORK_GROUP_SIZE       , sizeof(ext_data) , &blockSize , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_LOCAL_MEM_SIZE            , sizeof(ext_data) , &sharedMem , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS  , sizeof(ext_data) , &witemDims , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_MAX_WORK_ITEM_SIZES       , sizeof(ext_data) , &xyzDims[0], NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_EXTENSIONS                , sizeof(deviceExt), deviceExt  , NULL);
}
freed(devices);
return 0;

// Obtain specifications for each connected device
for(unsigned int i=0; i<num_devices; i++) {
    clGetDeviceInfo(devices[i], CL_DEVICE_NAME                      , sizeof(name_data), name_data  , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_AVAILABLE                 , sizeof(ext_data) , &deviceAvl , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_ADDRESS_BITS              , sizeof(ext_data) , &deviceAdd , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_MAX_CLOCK_FREQUENCY       , sizeof(ext_data) , &frequency , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_GLOBAL_MEM_SIZE           , sizeof(ext_data) , &globalMem , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_MAX_WORK_GROUP_SIZE       , sizeof(ext_data) , &blockSize , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_LOCAL_MEM_SIZE            , sizeof(ext_data) , &sharedMem , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS  , sizeof(ext_data) , &witemDims , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_MAX_WORK_ITEM_SIZES       , sizeof(ext_data) , &xyzDims[0], NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_EXTENSIONS                , sizeof(deviceExt), deviceExt  , NULL);
}
freed(devices);
return 0;

// Obtain specifications for each connected device
for(unsigned int i=0; i<num_devices; i++) {
    clGetDeviceInfo(devices[i], CL_DEVICE_NAME                      , sizeof(name_data), name_data  , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_AVAILABLE                 , sizeof(ext_data) , &deviceAvl , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_ADDRESS_BITS              , sizeof(ext_data) , &deviceAdd , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_MAX_CLOCK_FREQUENCY       , sizeof(ext_data) , &frequency , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_GLOBAL_MEM_SIZE           , sizeof(ext_data) , &globalMem , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_MAX_WORK_GROUP_SIZE       , sizeof(ext_data) , &blockSize , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_LOCAL_MEM_SIZE            , sizeof(ext_data) , &sharedMem , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS  , sizeof(ext_data) , &witemDims , NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_MAX_WORK_ITEM_SIZES       , sizeof(ext_data) , &xyzDims[0], NULL);
    clGetDeviceInfo(devices[i], CL_DEVICE_EXTENSIONS                , sizeof(deviceExt), deviceExt  , NULL);
}
freed(devices);
return 0;

[Below is a run showing output from a node that has no GPU]

jroy@compute-0-142 openclcpuget$ /home/nroy/opencl_example/get_cpu_opencl/bin/openclcpuget
Detected 1 platforms
Using OpenCL to get more information about the platform:

CL_PLATFORM_NAME       Intel(R) OpenCL
CL_PLATFORM_VENDOR     Intel(R) Corporation
CL_PLATFORM_VERSION    OpenCL 1.2 LINUX
CL_PLATFORM_PROFILE    FULL_PROFILE
CL_PLATFORM_EXTENSIONS---cl_khr_3d_image_writes cl_khr_depth_images cl_khr_global_int32 base atoms cl_khr_global_int32 extended_atomics cl_khr_int32 base atoms cl_khr_local_int32 extended_atomics cl_khr_byte_addressable_store cl_khr_spir cl_intel_exec_by_local_thread cl_khr_depth_images cl_khr_3d_image_writes cl_khr_fp64
Running OpenCL code to get device specifications:

------------------------------------------------------------------------
CL_DEVICE_NAME                           Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz
CL_DEVICE_AVAILABLE                1
CL_DEVICE_ADDRESS_BITS             64 bits
CL_DEVICE_MAX_CLOCK_FREQUENCY      2.8 MHz
CL_DEVICE_GLOBAL_MEM_SIZE          135.281 GB
CL_DEVICE_MAX_WORK_GROUP_SIZE      8192 work items
CL_DEVICE_LOCAL_MEM_SIZE           32768 Bytes
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS 3
CL_DEVICE_MAX_WORK_ITEM_SIZES : X  8192 work items
     : Y  8192 work items
     : Z  8192 work items
CL_DEVICE_EXTENSIONS               cl_khr_icd cl_khr_global_int32_base_atomics cl_khr_global_int32_extended_atomics cl_khr_local_int32_base_atomics cl_khr_local_int32_extended_atomics cl_khr_byte_addressable_store cl_khr_spir cl_intel_exec_by_local_thread cl_khr_depth_images cl_khr_3d_image_writes cl_khr_fp64

[Below is a run showing output from a node that has a GPU]

USING CUDA – 6.5:
At the time of writing this document CUDA 6.5 was not available. However the GPU nodes now have the latest NVIDIA driver (“NVIDIA-Linux-x86_64-340.58”) and the CUDA 6.5 SDK (“module whatis cuda-6.5”). CUDA 6.5 has the following advantages over 6.0:

1) CUDA 6.5 takes the next step, enabling CUDA on 64-bit ARM platforms. The heritage of ARM64 is in low-power, scale-out data centers and microservers, while GPUs are built for ultra-fast compute performance. When we combine the two, we have a compelling solution for HPC. ARM64 provides power efficiency, system configurability,
and a large, open ecosystem. GPUs bring to the table high-throughput, power-efficient compute performance, a large HPC ecosystem, and hundreds of CUDA-accelerated applications. For HPC applications, ARM64 CPUs can offload the heavy lifting of computational tasks to GPUs. CUDA and GPUs make ARM64 competitive in HPC from day one.

2) Users of cuFFT often need to transform input data before performing an FFT, or transform output data afterwards. Before CUDA 6.5, doing this required running additional CUDA kernels to load, transform, and store the data. These transform kernels increase the bandwidth used by applications, and that’s where cuFFT device callbacks come in.

![cuFFT 6.5 let you specify CUDA device callback functions that re-direct or manipulate the data as it is loaded before processing the FFT, and/or before it is stored after the FFT. This means cuFFT can transform the input and output data without extra bandwidth usage above what the FFT itself uses.

3) CUDA 6.5 adds improved support for CUDA Fortran in the cuda-gdb debugger, the nvpprof command line profiler, cuda-memcheck, and the NVIDIA Visual Profiler. This includes debugging support for FORTRAN arrays (in Linux only), improved source-to-assembly code correlation, and better documentation. CUDA Fortran tools support is a beta feature in CUDA 6.5, and requires PGI compiler version 14.4 or higher.

4) CUDA 6.5 provides new CUDA occupancy calculator and occupancy-based launch configuration API interfaces. These functions help set execution configurations with reasonable occupancy. “cuda occupancy.h” includes stand-alone implementations of both the occupancy calculator and the occupancy-based launch configuration functions, so applications can use them without depending on the entire CUDA software stack.
5) CUDA 6.5 expands host compiler support to include Microsoft Visual Studio 2013 for Windows.

6) The core math libraries in CUDA 6.5 introduce significant performance improvements for many double precision functions, notably sqrt(), rsqrt(), hypot(), log(), log2(). These optimizations can result in real performance improvements in applications.

7) CUDA 6.5 (on Linux and Mac OS) now includes static library versions of the cuBLAS, cuSPARSE, cuFFT, cuRAND, and NPP libraries. This can reduce the number of dynamic library dependencies you need to include with your deployed applications. These new static libraries depend on a common thread abstraction layer library cuLIBOS (libculibos.a) distributed as part of the CUDA toolkit.

8) nvprune is a new binary utility which prunes host object files and libraries to only contain device code for the specified target architectures. For example, the following command line prunes libcublas_static.a to only contain sm_35 code and remove all other targets contained by the library.

   nvprune -arch sm_35 libcublas_static.a -o libcublas_static35.a

   Software developers may find nvprune useful for reducing the GPU object file sizes in their apps, especially if they use third-party or NVIDIA libraries.

9) The CUDA Multi-Process Service (MPS) transparently enables cooperative multiprocess CUDA applications, typically MPI jobs, to run kernels from multiple processes concurrently on individual GPUS. CUDA 6 introduced MPS, and CUDA 6.5 significantly improves MPS performance: reducing launch latency from 7 to 5 microseconds, and reducing launch and synchronize latency from 35 to 15 microseconds.

10) NVIDIA driver Xid error reporting reports general GPU errors via the operating system’s kernel or event logs. The messages can indicate hardware problems, NVIDIA software problems, or user application problems. CUDA 6.5 improves Xid error 13 reporting on Linux to give more detail and show the type of the Xid 13 error cause. Here is an example error message from earlier CUDA versions.

   GPU at 0000:07:00: GPU-b850f46d-d5ea-c752-ddf3-c4453e44d3f7
   Xid (0000:07:00): 13, 0003 0000 0000 0000a1c0 000002bc 00000003 00000000

   The same error message could be displayed for a number of causes. CUDA 6.5 differentiates Xid 13 errors based on the type of cause, as shown here.

   Xid (0000:07:00): 13, Graphics SM Warp Exception on (GPC 0, TPC 4): Stack Error
   Xid (0000:07:00): 13, Graphics Exception: ESR - 0x506648=0x2000d, 0x506650=0x0, 0x506644=0x8eef2, 0x50664c=0x7f
   Xid (0000:07:00): 13, Graphics Exception: ChID 0002, Class 0000a1c0, Offset 000002bc, Data 00000003

11) Support for Block Sparse Row (BSR) format matrices in cuSparse, and Application Replay mode in the NVIDIA Visual Profiler that enables faster analysis of complex scenarios involving multiple hardware counters.

Contact us at ITS – Research Computing if you have further questions or any issues.

Nilay K Roy, Ph.D

-----------------------------------------------------------------
Nilay Roy, PhD Computational Physics, MS Computer Science
Assistant Director - Research Computing, Information Technology Services
Northeastern University, 221-177, 360 Huntington Avenue, Boston, MA 02115
Email: n.roy@neu.edu (C) 508.226.2261 (Preferred) / (O) 617.373.6048
Northeastern Research Computing Website: http://www.northeastern.edu/rc
-----------------------------------------------------------------