How to Login ?

1. You will need a server login. If you do not have any Download the form and submit it to the Software Lab.
2. Use the command (**ssh <nis-username>@10.5.18.114 -X**)

How to submit a job?

To submit a job

1. create a jobscript file as given in the example(s).
2. Use the following command to submit the job:

   ```
   qsub <jobscript>
   Your job <jobid> ("<jobscript>") has been submitted
   ```

   The above message will be displayed, if the job submission is successful
3. Use the following command to check status of the job:

   ```
   qstat -f
   ```

   If the above command is not showing anything then the job is finished.
4. You can check the result by looking at the STDOUT and STDERR files in your home directory in the format given below

   STDIN.o<jobid>
   STDIN.e<jobid>

C Program Example

C Program file: cprogram.c

```
#include <iostream>
using namespace std;

int main(int argc, char** argv)
{
      cout << "You have entered " << argc << " arguments:" << "\n";
      for (int i = 0; i < argc; ++i)
        cout << argv[i] << "\n";
      return 0;
}
```

Compile(with gcc) it to get **cprogram.o**

job file: cprogram.pbs

```
#PBS -N testcpp
#PBS -l walltime=00:05:00,mem=400mb,nodes=1:ppn=1
#PBS -V

# User Directives
./cprogram.o 345
#End of script
```

Tensorflow Example

tensorflow program: tensorflow_test.py

```
import tensorflow as tf
# Creates a graph.
a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
c = tf.matmul(a, b)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
```

job file: tensorflow_test.pbs

```
#!/bin/sh
# Torque script to submit CUDA C/C++ programs.
# Torque directives
#PBS -N testpy
#PBS -q gpu
#PBS -l walltime=00:05:00,mem=400mb,nodes=1:ppn=1
#PBS -V
# User Directives
python tensorflow_test.py
#End of script
```

CUDA Program Example

CUDA Program file: cuda_test.cu

```
#include <stdio.h>

const int N = 16;
const int blocksize = 16;

__global__
void hello(char *a, int *b)
{
	a[threadIdx.x] += b[threadIdx.x];
}

int main()
{
	char a[N] = "Hello \0\0\0\0\0\0";
	int b[N] = {15, 10, 6, 0, -11, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

	char *ad;
	int *bd;
	const int csize = N*sizeof(char);
	const int isize = N*sizeof(int);

	printf("%s", a);

	cudaMalloc( (void**)&ad, csize );
	cudaMalloc( (void**)&bd, isize );
	cudaMemcpy( ad, a, csize, cudaMemcpyHostToDevice );
	cudaMemcpy( bd, b, isize, cudaMemcpyHostToDevice );

	dim3 dimBlock( blocksize, 1 );
	dim3 dimGrid( 1, 1 );
	hello<<<dimGrid, dimBlock>>>(ad, bd);
	cudaMemcpy( a, ad, csize, cudaMemcpyDeviceToHost );
	cudaFree( ad );
	cudaFree( bd );

	printf("%s\n", a);
	return EXIT_SUCCESS;
}
```

Compile (with nvcc) it to get **cuda_test.o**

job file: cprogram.pbs

```
#PBS -N OpenCLTest
#PBS -l walltime=00:10:00,nodes=1:ppn:2
cuda_test.o
```

OPENCL Program Example

OPENCL Program file: opencl_test.c

```
#include <stdio.h>
#include <stdlib.h>

#ifdef __APPLE__
#include <OpenCL/opencl.h>
#else
#include <CL/cl.h>
#endif

#define MAX_SOURCE_SIZE (0x100000)

int main(void) {
    // Create the two input vectors
    int i;
    const int LIST_SIZE = 1024;
    int *A = (int*)malloc(sizeof(int)*LIST_SIZE);
    int *B = (int*)malloc(sizeof(int)*LIST_SIZE);
for(i = 0; i < LIST_SIZE; i++) {
    A[i] = i;
    B[i] = LIST_SIZE - i;
}

// Load the kernel source code into the array source_str
FILE *fp;
char *source_str;
size_t source_size;

fp = fopen("vector_add_kernel.cl", "r");
if (!fp) {
    fprintf(stderr, "Failed to load kernel.\n");
    exit(1);
}
source_str = (char*)malloc(MAX_SOURCE_SIZE);
source_size = fread( source_str, 1, MAX_SOURCE_SIZE, fp);
fclose( fp );

// Get platform and device information
cl_platform_id platform_id = NULL;
cl_device_id device_id = NULL;
cl_uint ret_num_devices;
cl_uint ret_num_platforms;
cl_int ret = clGetPlatformIDs(1, &platform_id, &ret_num_platforms);
ret = clGetDeviceIDs( platform_id, CL_DEVICE_TYPE_ALL, 1,
        &device_id, &ret_num_devices);

// Create an OpenCL context
cl_context context = clCreateContext( NULL, 1, &device_id, NULL, NULL, &ret);

// Create a command queue
cl_command_queue command_queue = clCreateCommandQueue(context, device_id, 0, &ret);

// Create memory buffers on the device for each vector
cl_mem a_mem_obj = clCreateBuffer(context, CL_MEM_READ_ONLY,
        LIST_SIZE * sizeof(int), NULL, &ret);
cl_mem b_mem_obj = clCreateBuffer(context, CL_MEM_READ_ONLY,
        LIST_SIZE * sizeof(int), NULL, &ret);
cl_mem c_mem_obj = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
        LIST_SIZE * sizeof(int), NULL, &ret);

// Copy the lists A and B to their respective memory buffers
ret = clEnqueueWriteBuffer(command_queue, a_mem_obj, CL_TRUE, 0,
        LIST_SIZE * sizeof(int), A, 0, NULL, NULL);
ret = clEnqueueWriteBuffer(command_queue, b_mem_obj, CL_TRUE, 0,
        LIST_SIZE * sizeof(int), B, 0, NULL, NULL);

// Create a program from the kernel source
cl_program program = clCreateProgramWithSource(context, 1,
        (const char **)&source_str, (const size_t *)&source_size, &ret);

// Build the program
ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);

// Create the OpenCL kernel
cl_kernel kernel = clCreateKernel(program, "vector_add", &ret);

// Set the arguments of the kernel
ret = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *)&a_mem_obj);
ret = clSetKernelArg(kernel, 1, sizeof(cl_mem), (void *)&b_mem_obj);
ret = clSetKernelArg(kernel, 2, sizeof(cl_mem), (void *)&c_mem_obj);

// Execute the OpenCL kernel on the list
```

```
    size_t global_item_size = LIST_SIZE; // Process the entire lists
    size_t local_item_size = 64; // Process in groups of 64
    ret = clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL,
            &global_item_size, &local_item_size, 0, NULL, NULL);

    // Read the memory buffer C on the device to the local variable C
    int *C = (int*)malloc(sizeof(int)*LIST_SIZE);
    ret = clEnqueueReadBuffer(command_queue, c_mem_obj, CL_TRUE, 0,
            LIST_SIZE * sizeof(int), C, 0, NULL, NULL);
    // Display the result to the screen
    for(i = 0; i < LIST_SIZE; i++)
        printf("%d + %d = %d\n", A[i], B[i], C[i]);

    // Clean up
    ret = clFlush(command_queue);
    ret = clFinish(command_queue);
    ret = clReleaseKernel(kernel);
    ret = clReleaseProgram(program);
    ret = clReleaseMemObject(a_mem_obj);
    ret = clReleaseMemObject(b_mem_obj);
    ret = clReleaseMemObject(c_mem_obj);
    ret = clReleaseCommandQueue(command_queue);
    ret = clReleaseContext(context);
    free(A);
    free(B);
    free(C);
    return 0;
}
```

Kernel File : vector_add_kernel.cl

```
-----------------------------------------
__kernel void vector_add(__global int *A, __global int *B, __global int *C) {
    // Get the index of the current element
    int i = get_global_id(0);
    C[i] = A[i] + B[i];
}
```

Compile (with gcc) it to get **opencl_test.o**

job file: opencl_test.pbs

```
----------------------------------
#PBS -N OpenCLTest
#PBS -l walltime=00:10:00,nodes=1:ppn:2
```

./opencl_test.o