GPU Server Guide @ Robotics Lab

Account Creation:

To get an account for accessing GPU server, please send your following information at ahkhan@kfupm.edu.sa

Required Login Name:
Full Name:
Mobile Number:
Email:

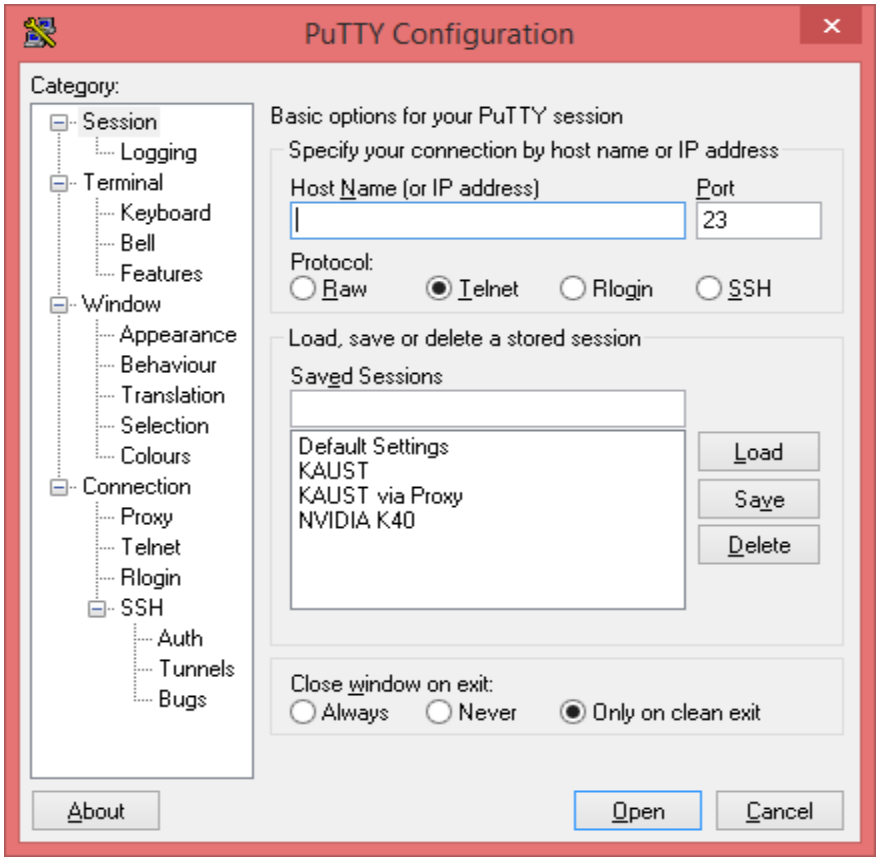
Note: You will be provided access to GPU server for only the current Term. If you need access for long duration then send the expected work duration with the approval from your advisor.

Login to Server:

For login to server, you need a ssh client such as “Putty”. It is freely available on internet, you can download putty from http://www.putty.org/

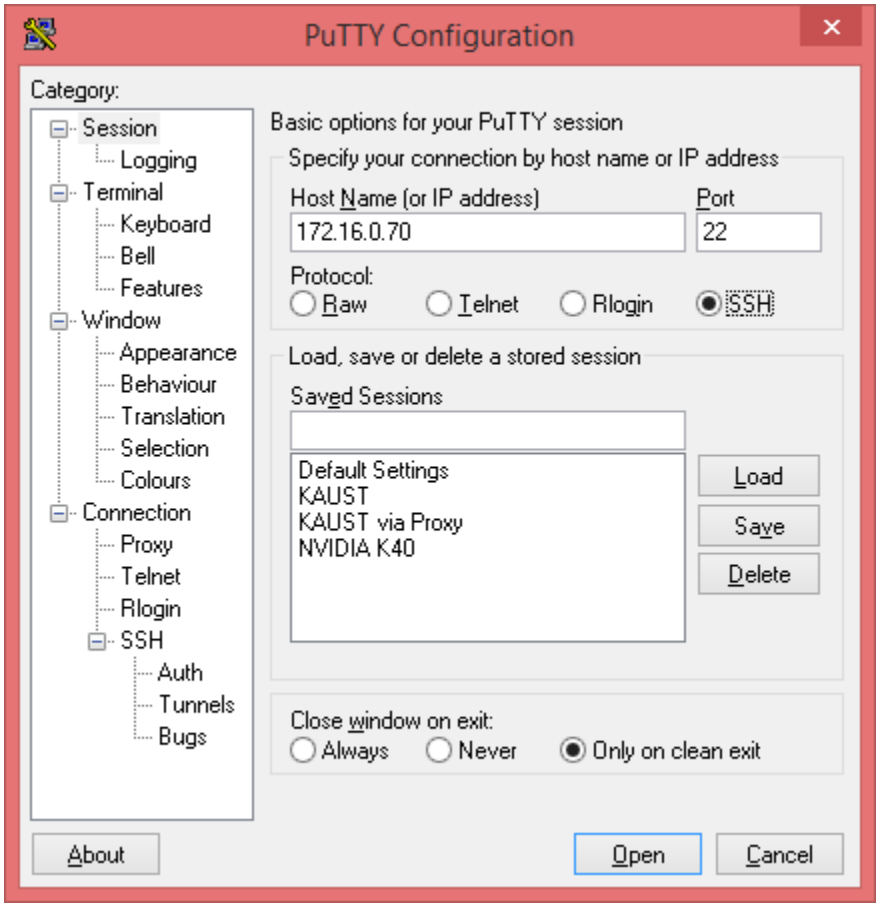
Steps:

1. Open Putty

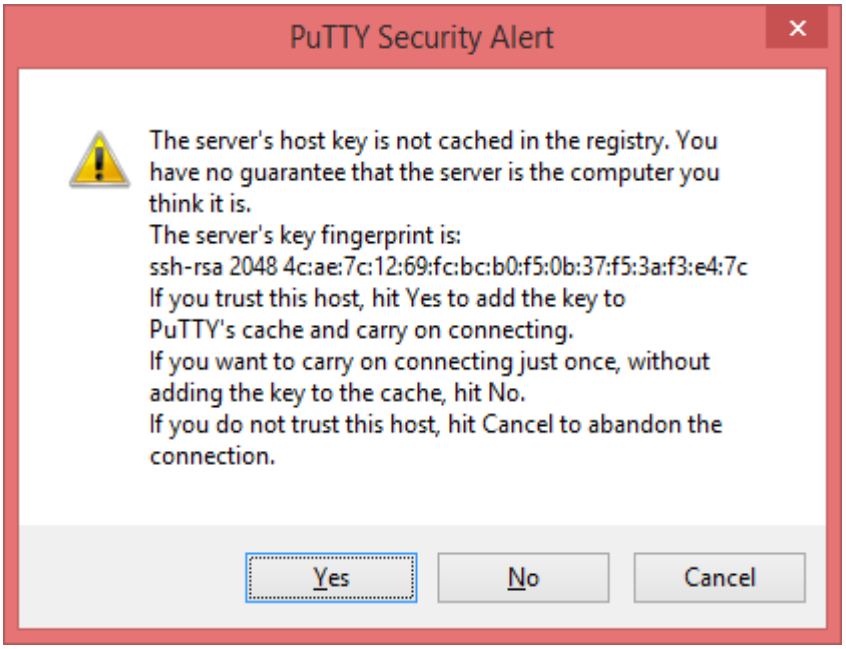


Prepared By Ayaz ul Hassan Khan

2. Enter IP address of GPU Server: 172.16.0.70 and Select “SSH” in Protocol. Click Open

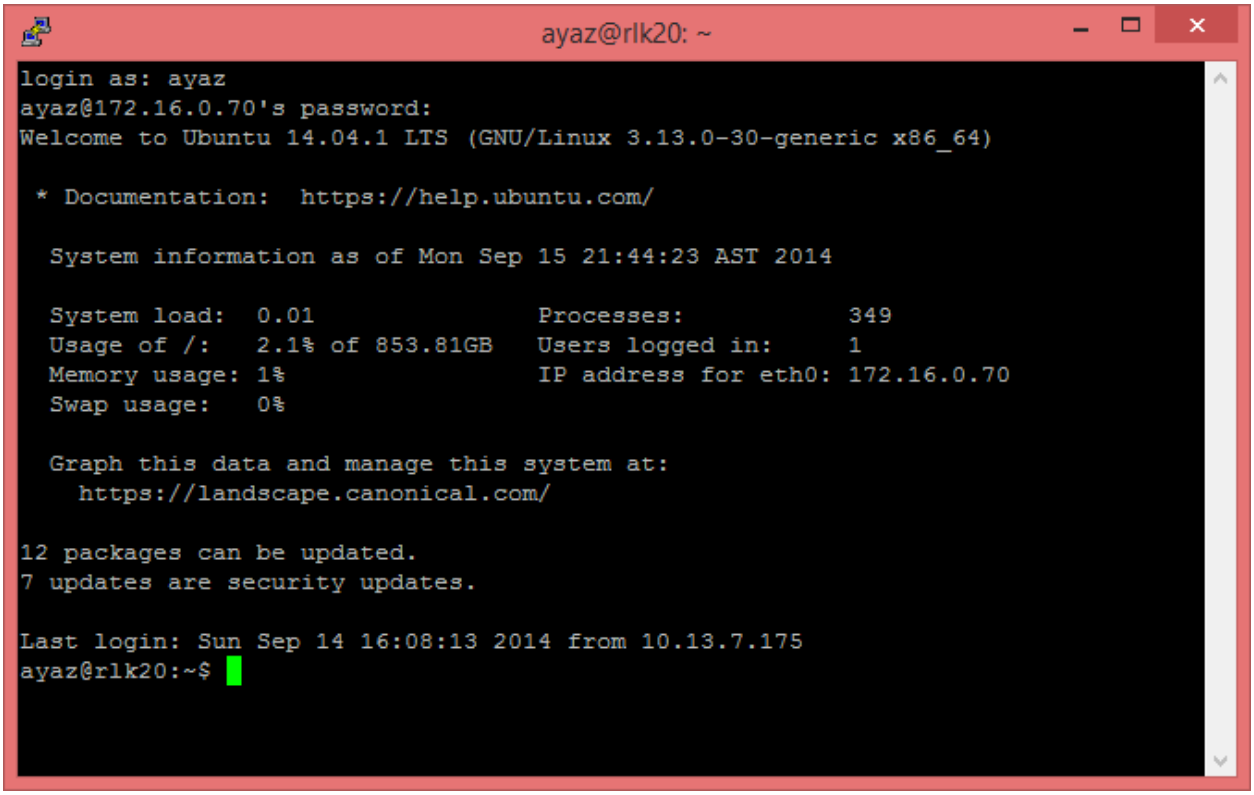


3. Click Yes on Putty Security Alert. This is only for the first login.



4. Enter User name and Password.

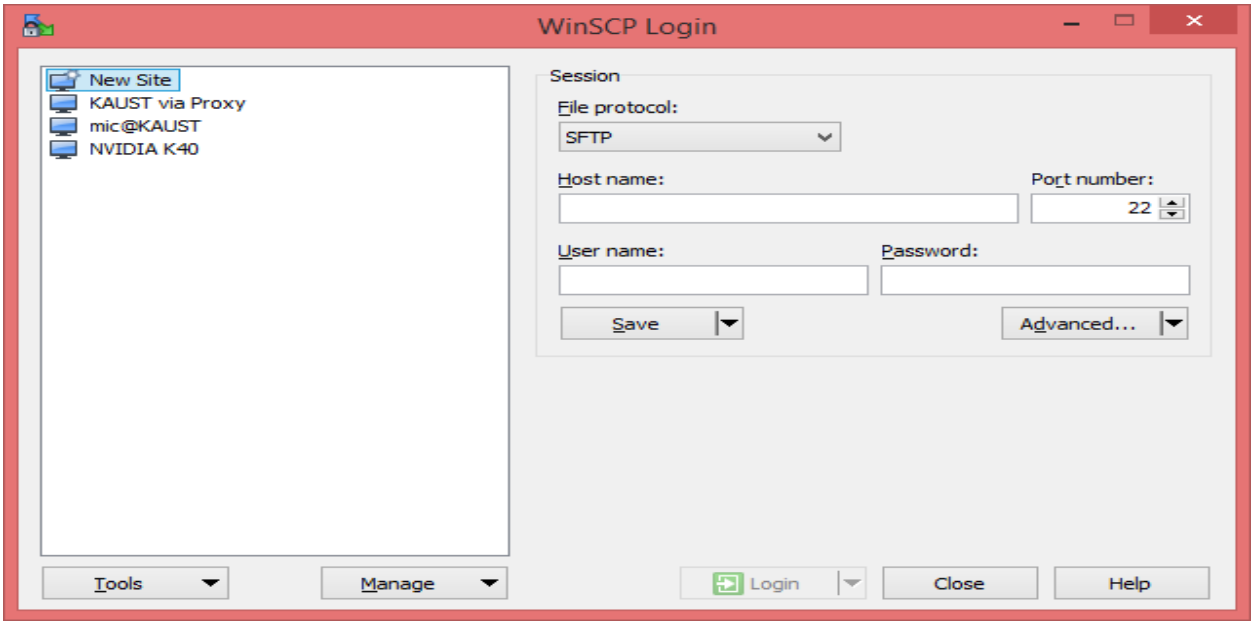
5. You are not logged in to the system.



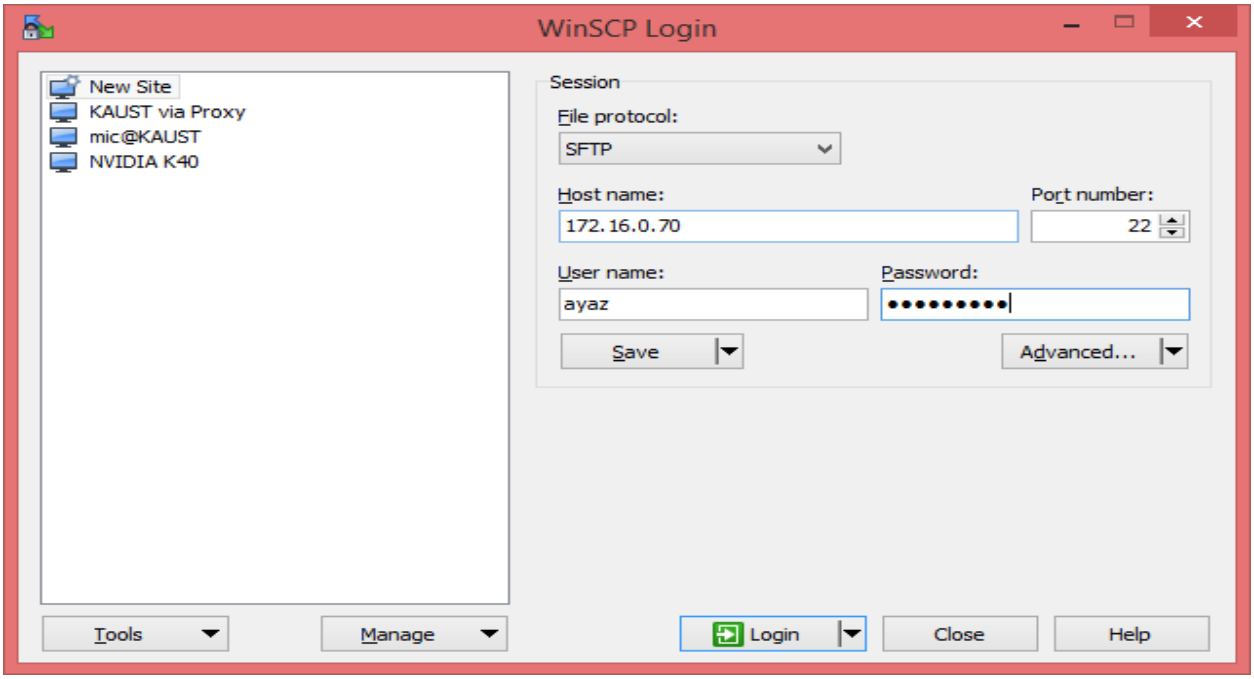
Transfer Files:

For login to server, you need a scp client such as “WinSCP”. It is freely available on internet, you can download WinSCP from http://winscp.net/

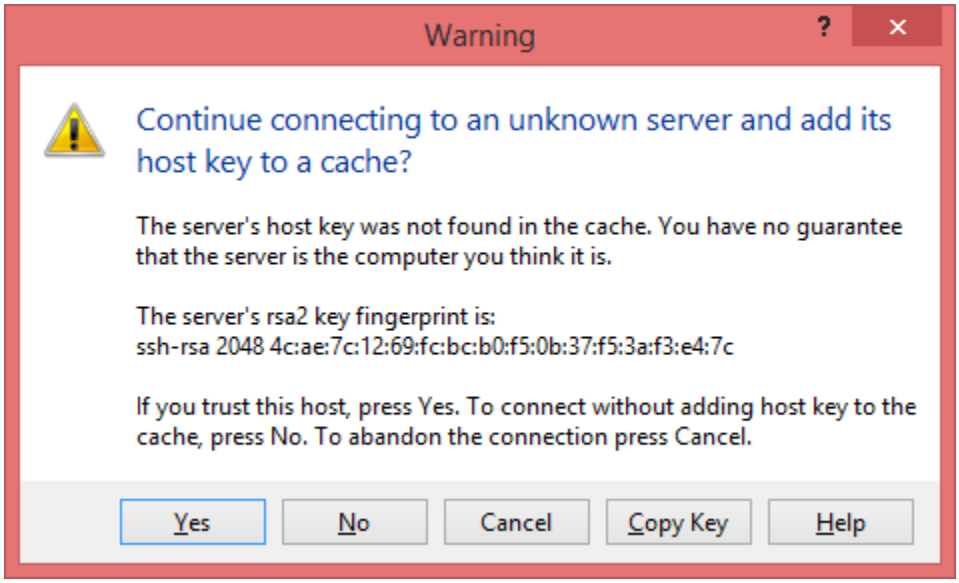
1. Open WinSCP



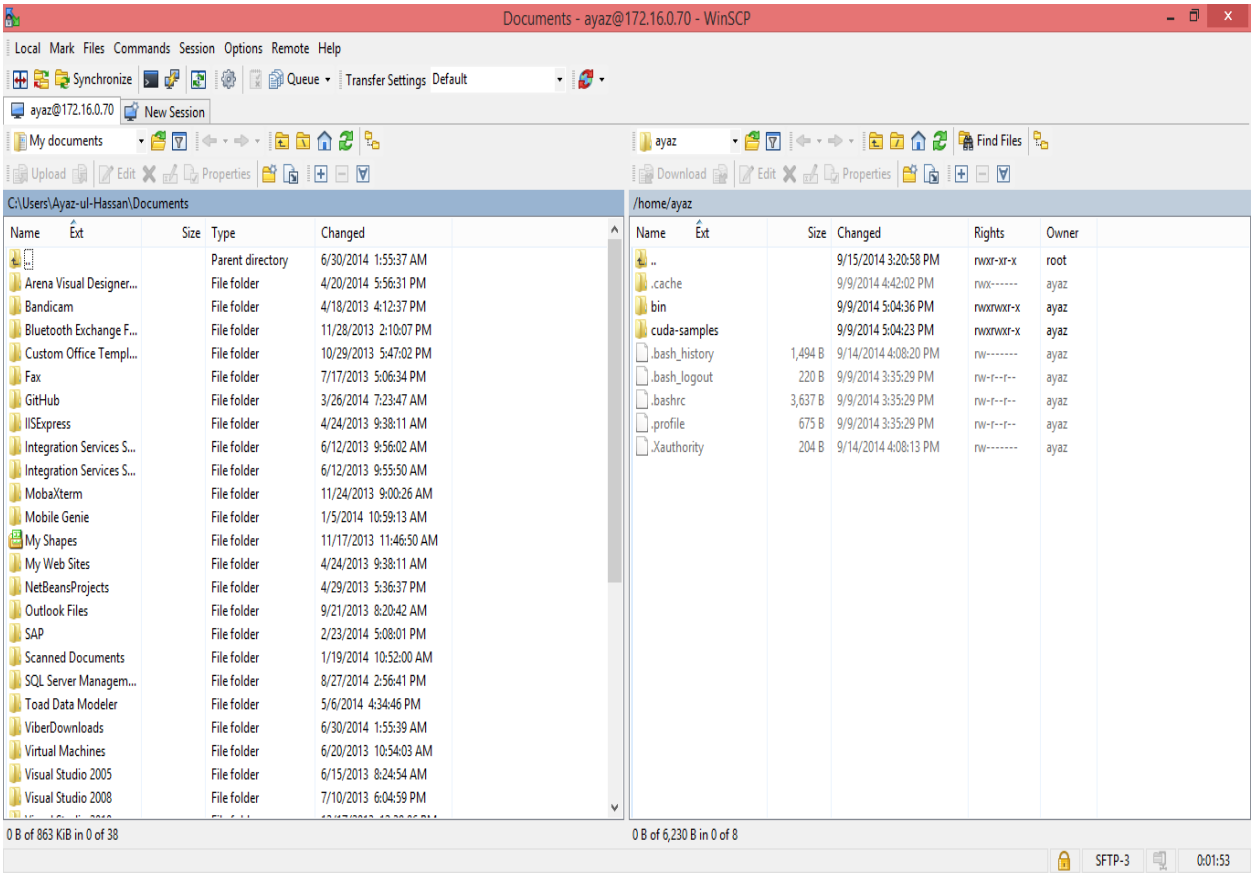
2. Enter IP Address of GPU Server: 172.16.0.70 as Host name, Enter your user name and password. Click Login



3. Click Yes on Warning for host key. This is only for the first login.



4. You can now transfer files from server to local machine or local machine to server by drag and drop the files/folders from left to right or right to left respectively.



CUDA Example:

Kernel File: kernel.cu

```
__global__ void matrix_scale(float *C, float const* __restrict__ A, int scale, int N)
{
   int tid = threadIdx.x;
   int bid = blockIdx.x;
   int ij = bid * BLOCKSIZE + tid;
   int i = (ij / N) * MERGE_LEVEL;
   int j = (ij % N) * SKEW_LEVEL;
   for (int m = 0; m < MERGE_LEVEL; m++)
      for (int n = 0; n < SKEW_LEVEL; n++)
         C[((i + m)) * N + ((j + n))] = scale * A[((i + m)) * N + ((j + n))];
}
```

Main File: main.cu

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<time.h>
#include<cuda.h>

void checkCudaError(const char *msg) {
   cudaError_t err = cudaGetLastError();
   if (cudaSuccess != err) {
      printf("%s(%i) : CUDA error : %s : (%d) %s\n", __FILE__, __LINE__, msg, (int) err,
cudaGetErrorString(err));
      exit(-1);
   }
}
#include "params.h"
#include "kernel.cu"

int main(int argc, char *argv[]) {
   int N = 1024;
   int GPU = 0;
   if (argc > 1)N = atoi(argv[1]);
   if (argc > 2)GPU = atoi(argv[2]);
   cudaSetDevice(GPU);
   float *A, *C;
   int memsize = N * N * sizeof (float);
   cudaMallocManaged(&A, memsize);
   cudaMallocManaged(&C, memsize);
   A[0] = 1;
```

```
    dim3 threads(BLOCKSIZE, 1);
    dim3 grid(N * N / BLOCKSIZE / MERGE_LEVEL / SKEW_LEVEL, 1);

    float time;
    cudaEvent_t start, stop; // variables to record time of kernel start and stop

    // pre-requisite to collect timings at kernel start and stop events
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    //record the time at kernel start
    cudaEventRecord(start, 0);

    matrix_scale << <grid, threads >> >(C, A, 3.0, N);
    cudaDeviceSynchronize();

    //record the time at kernel stop
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);

    //calculate the time using start and stop timings
    cudaEventElapsedTime(&time, start, stop);

    printf("kernel execution time = %f sec\n", time * 1e-3);

    printf("A[0] = %f, C[0] = %f\n", A[0], C[0]);
    printf("End of Program\n");
    cudaFree(A);
    cudaFree(C);
    cudaThreadExit();
}
```

Compiling and Running the Example:

Steps:

1. Goto the source directory containing kernel file, main file, other headers, and Makefile:

```
cd test_program/
```

2. To compile the program, use the Make utility (Makefile is provided in the example package):

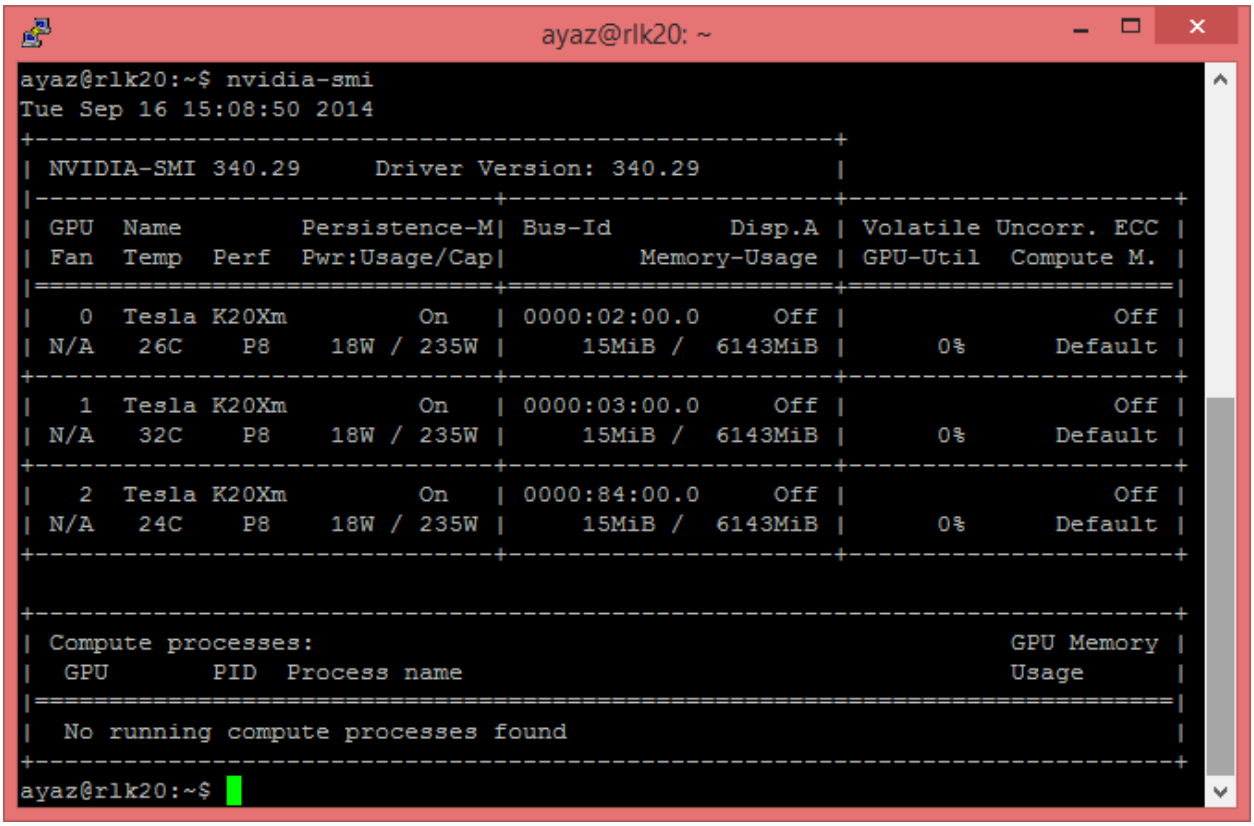
```
make
```

3. To run the program, execute following command:

```
./main
```

Note: Example code with Makefile can be downloaded from the following link:
https://dl.dropboxusercontent.com/u/13524969/test_program.tgz

Before running your CUDA program, make sure that no one else is using GPUs at the same time so there should not be any conflict among different cuda kernels. You can check this by running following command: nvidia-smi



To run the kernel on a particular GPU device, you need to use following API function:

cudaSetDevice(GPU);

Here, GPU is the ID of GPU to be used. It can be 0, 1, or 2.

For any help regarding GPU Server and CUDA:
Contact Person: Ayaz ul Hassan Khan
Email: ahkhan@kfupm.edu.sa
Robotics Lab
Available Hours: UT 3:00 PM – 5:00 PM