Instructions for setting up OpenCL Lab

This document describes the procedure to setup OpenCL Lab for Linux and Windows machine. Specifically if you have limited no. of graphics cards and a no. of users will be having the lab sessions simultaneously then we can use putty, an ssh client, for accessing the server. Detail is given below.

A- If you have a single Linux machine with GPU

For running OpenCL program you can use this single machine with GPU in LAN with other clients running windows. The idea is to access the Linux machine having OpenCl devices from several other clients running windows. Follow the following steps (all the commands and outputs are highlighted in green):

1. On the machine with Linux install all the necessary software including AMD APP SDK and the required driver. All installation must be done by root. Make sure you are connected to internet on this machine. If your OS is 64 bit download the 64 bit version of AMD APP SDK. On Linux you just need to unzip the downloaded package like this:

   # tar -xvzf AMD-APP-SDK-v2.4-lnx32.tgz , this will unzip the package in the home of the root that is /root. If one wants to unzip the package in /, do this:

   # cd / and then type **tar -xvzf AMD-APP-SDK-v2.4-lnx32.tgz** this will unzip the package in /

   You should also make sure that your Graphics card is detected. For that you need to type **lspci** at the command, which will list all the PCI devices in the system. Use sh command to install driver file. Drivers can be downloaded from
   http://support.amd.com/us/gpudownload/Pages/index.aspx

2. After installation go to the terminal and type:

   a. export AMDAPPSDKROOT=*<location where sdk is extracted>* e.g if you extracted the SDK in (step 1) at /root/AMD-APP-SDK-v2.4-lnx32 , then the command that you have to type is:

   # AMDAPPSDKROOT=/root/ AMD-APP-SDK-v2.4-lnx32

   Cross check if the variable is set properly or not by using the following Command:

# echo $AMDAPPSDKROOT , this should output the following:
/root/ AMD-APP-SDK-v2.4-lnx32
If you are getting this output all is set well, else repeat this step.

4. Repeat the step 3 for another variable: **AMDAPPSDKSAMPLESROOT**

5. Set the library path: LD_LIBERARY_PATH:

a. – For 32-bit systems:

# export LD_LIBRARY_PATH=$AMDAPPSDKROOT/lib/x86:$LD_LIBRARY_PATH

b. – For 64-bit systems:

# export LD_LIBRARY_PATH=$AMDAPPSDKROOT/lib/x86_64:$LD_LIBRARY_PATH

As in step 3 use **echo $LD_LIBERARY_PATH** to cross check if the variable is set correctly or not. This command should return the following:
AMDAPPSDKROOT/lib/x86

Note: If this variable is not set properly, the **libOpenCL.so** may not get included, and you will get errors. Also Ideally you should make the exports permanent by writing the exports in files such as .bashrc, .profile or bash_profile. You might find these files in /etc. Simply open any of these files and copy paste above commands starting with exports (without # of course!). The net effect will be this that each time you restart your machine the settings of exports will remain intact; you will not do exports again.

6. **Register the OpenCL ICD.** The **OpenCL™ ICD** (Installable Client Driver) is a means of allowing multiple OpenCL™ implementations to co-exist and applications to select between them at runtime. Download this file here: http://developer.amd.com/Downloads/icd-registration.tgz and unzip in / by using the following command:
   i. In the command shell, type "cd /"
   ii. Type "**ls**", you will see a directory named "etc". The icd-registration.tgz needs to be unzipped to that directory.
   iii. Now copy icd-registration.tgz to "**cd /**"
   iv. Unzip by typing "**tar xzf icd-registration.tgz**"
   v. Now you should be able to find an "**OpenCL**" directory in "/etc". Check etc/OpenCL/vendors/, make sure there are "**amdocl32.icd**" and "**amdocl64.icd**" there.

7. Test the SDK samples since your installation is OK. Go to AMD-APP-SDK-v2.4-lnx32 by typing the following command **: # cd AMD-APP-SDK-v2.4-lnx32** assuming you unzipped the package in /root.
    i. Now go to the "samples" by typing **cd samples,** and type **make.** This will start building all the sample projects (you will see the progress on the screen).
    ii. The corresponding binaries are generated in the folder bin. Go to bin folder and **ls** to see the list of all the binary files of the projects. To run any sample simply type **./<nameof the project binary>**
    iii. Every sample project has instructions for executing the binaries, and optional flags. For example if you go to matrix multiplication sample, you will find a pdf file inside the doc folder. This gives you various options, such as you can get the execution time by typing **./matrixmultiplication -t**
    iv. An important utility is provided for querying the devices and getting info about opencl platform installed on the system. This tool can be helpful to check if your system is detecting all the opencl devices or not. This file **clinfo** is present in bin folder inside the AMD APP Package you extracted. Go to that file by typing cd command and type **./clinfo.** This will give the following output:

```
Number
of platforms:                      1
Platform Profile:                       FULL_PROFILE
Platform Version:                       OpenCL 1.1 ATI-Stream-v2.2
(302)
Platform Name:                          ATI Stream
Platform Vendor:                        Advanced Micro Devices,
Inc.
Platform Extensions:               cl_khr_icd cl_amd_event_callback


Platform Name:                          ATI Stream
Number
of devices:                        2
Device Type:                            CL_DEVICE_TYPE_CPU
Device ID:                              4098
Max compute units:                      6
Max work items dimensions:              3
Max work items[0]:                    1024
Max work items[1]:                    1024
Max work items[2]:                    1024
Max work group size:                    1024
Preferred vector width char:                  16
Preferred vector width short:                 8
Preferred vector width int:             4
Preferred vector width long:                  2
Preferred vector width float:                 4
Preferred vector width double:          0
Max clock frequency:                    800Mhz
Address bits:                           64
Max memory allocation:                  1073741824
```

```
Image support:                     No
Max size of kernel argument:                    4096
Alignment (bits) of base address:        1024
Minimum alignment (bytes) for any datatype:    128
Single precision floating point capability
Denorms:                                Yes
Quiet NaNs:                             Yes
Round to nearest even:                  Yes
Round to zero:                          Yes
Round to +ve and infinity:                    Yes
IEEE754-2008 fused multiply-add:              No
Cache type:                               Read/Write
Cache line size:                          64
Cache size:                               65536
Global memory size:                       3221225472
Constant buffer size:                     65536
Max number of constant args:                    8
Local memory type:                        Global
Local memory size:                        32768
Profiling timer resolution:               1
Device endianess:                         Little
Available:                                Yes
Compiler available:                       Yes
Execution capabilities:
Execute OpenCL kernels:                 Yes
Execute native function:                Yes
Queue properties:
Out-of-Order:                    No
Profiling :                             Yes
Platform ID:                      0x7f47e30e2b20
Name:                    AMD Phenom(tm) II X6 1055T Processor
Vendor:                            AuthenticAMD
Driver version:                    2.0
Profile:                           FULL_PROFILE
Version:                  OpenCL 1.1 ATI-Stream-v2.2 (302)
Extensions:                               cl_amd_fp64
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_int64_base_atomics cl_khr_int64_extended_atomics
cl_khr_byte_addressable_store cl_khr_gl_sharing
cl_ext_device_fission
cl_amd_device_attribute_query cl_amd_printf
Device Type:                              CL_DEVICE_TYPE_GPU
Device ID:                                4098
Max compute units:                        20
Max work items dimensions:                3
Max work items[0]:                      256
Max work items[1]:                      256
Max work items[2]:                      256
Max work group size:                      256
Preferred vector width char:                    16
Preferred vector width short:                   8
Preferred vector width int:               4
Preferred vector width long:                    2
Preferred vector width float:                   4
Preferred vector width double:            0
```

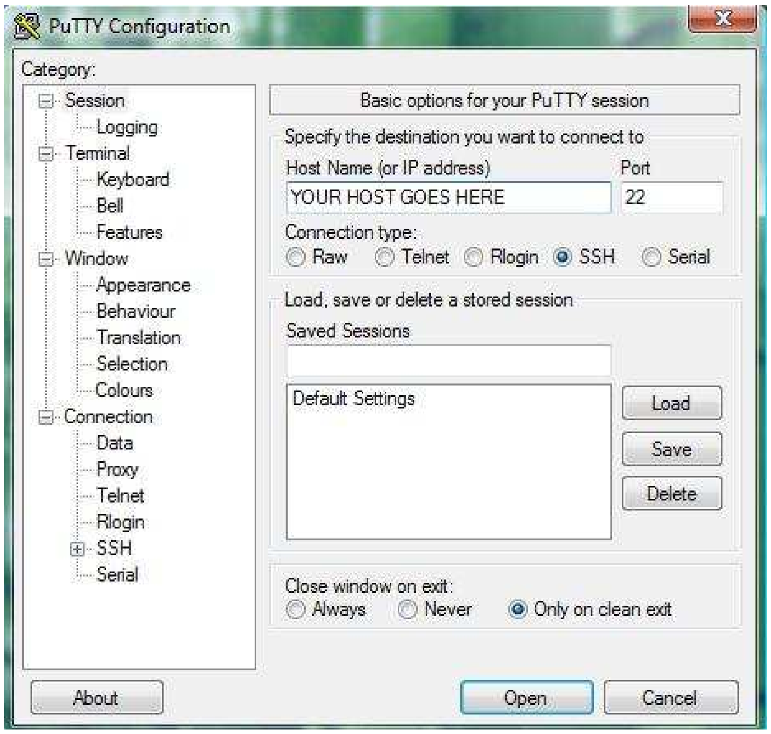
```
Max clock frequency:                    850Mhz
Address bits:                           32
Max memory allocation:                  134217728
Image support:                    Yes
Max number of images read arguments:    128
Max number of images write arguments:   8
Max image 2D width:               8192
Max image 2D height:              8192
Max image 3D width:               2048
Max image 3D height:   2048
Max image 3D depth:               2048
Max samplers within kernel:       16
Max size of kernel argument:                  1024
Alignment (bits) of base address:       32768
Minimum alignment (bytes) for any datatype:   128
Single precision floating point capability
Denorms:                              No
Quiet NaNs:                           Yes
Round to nearest even:                Yes
Round to zero:                        Yes
Round to +ve and infinity:                  Yes
IEEE754-2008 fused multiply-add:            Yes
Cache type:                             None
Cache line size:                        0
Cache size:                             0
Global memory size:                     536870912
Constant buffer size:                   65536
Max number of constant args:                  8
Local memory type:                      Scratchpad
Local memory size:                      32768
Profiling timer resolution:             1
Device endianess:                       Little
Available:                              Yes
Compiler available:                     Yes
Execution capabilities:
Execute OpenCL kernels:               Yes
Execute native function:              No
Queue properties:
Out-of-Order:                   No
Profiling :                           Yes
Platform ID:                            0x7f47e30e2b20
Name:                                   Cypress
Vendor:                           Advanced Micro Devices, Inc.
Driver version:                   CAL 1.4.736
Profile:                          FULL_PROFILE
Version:                          OpenCL 1.1 ATI-Stream-v2.2 (302)
Extensions:                             cl_amd_fp64
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_3d_image_writes cl_khr_byte_addressable_store
cl_khr_gl_sharing
cl_amd_device_attribute_query cl_amd_printf cl_amd_media_ops


Passed!
```

This shows we have one CPU device and one GPU device

Connecting in LAN

Now you want to connect this machine with several other machines running windows. The idea is to access this Linux machine having OpenCl device through these clients running windows. This can be done by following these steps:

8. Download putty (an ssh client) on each client running windows from here: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
9. Double click the downloaded file . A new window will open as shown below:



Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): YOUR HOST GOES HERE

Port: 22

Connection type: Raw [Telnet] Rlogin [✓ SSH] Serial

Load, save or delete a stored session

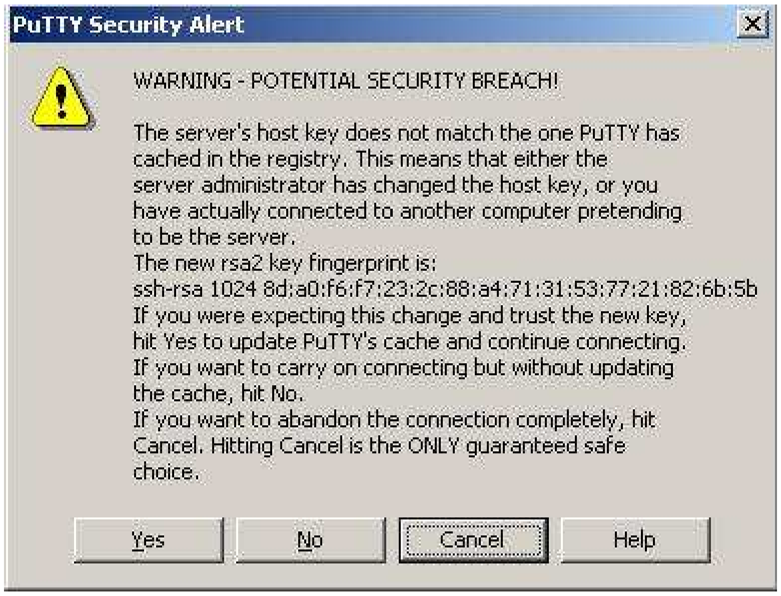
Saved Sessions

Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

10. Put the IP address of the Linux machine (the server) in the host name in putty. Leave the port to 22
11. Select the connection type as SSH and click **open**

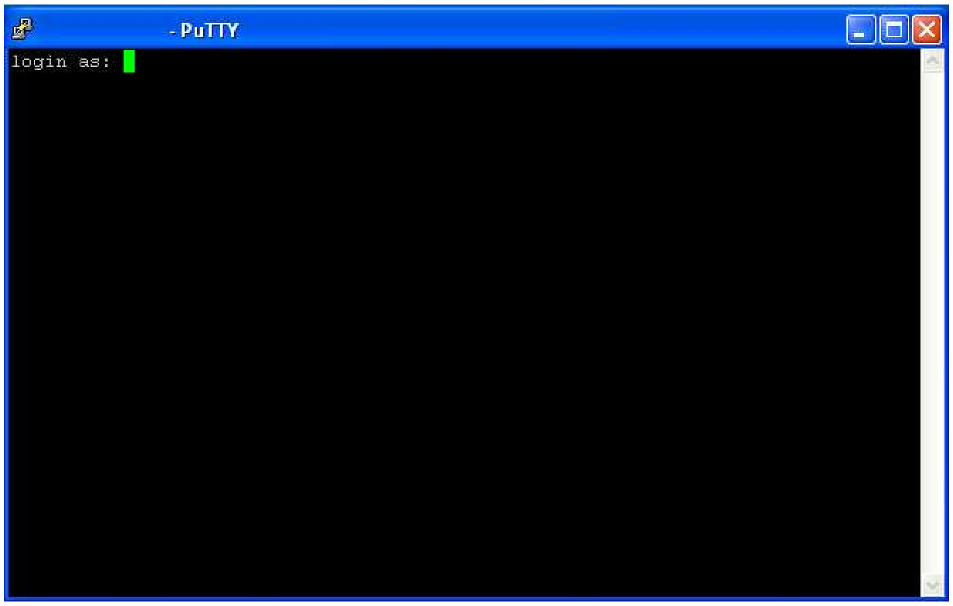
12. This will open a new warning window, simply ignore it and click yes. The warning window appears like this:


PuTTY Security Alert

WARNING - POTENTIAL SECURITY BREACH!

The server's host key does not match the one PuTTY has cached in the registry. This means that either the server administrator has changed the host key, or you have actually connected to another computer pretending to be the server.
The new rsa2 key fingerprint is:
ssh-rsa 1024 8d:a0:f6:f7:23:2c:88:a4:71:31:53:77:21:82:6b:5b
If you were expecting this change and trust the new key, hit Yes to update PuTTY's cache and continue connecting.
If you want to carry on connecting but without updating the cache, hit No.
If you want to abandon the connection completely, hit Cancel. Hitting Cancel is the ONLY guaranteed safe choice.

13. Now the work is done. You will get a terminal like this:



```
login as:
```

14. Now you need to enter your login id and password for this client machine assigned by the Linux machine. Whatever will be types on this terminal will be actually executed on the Linux machine. Thus multiple user will be having access to the same OpenCL device.
15. **Creating User ID.** This will be done on Linux machine. Login as root. And type following command for adding users: **# useradd user1** where user1 is the user ID of the first user. Similarly create all the required users for your lab.

16. Now you can assign each user a password as well. For this you can use the command **passwd user1**, this will set the password for the user1.
17. Now to test your users, log in from any of the clients by starting the putty and entering login id and password. See if you are able to successfully login and write and build simple c programs.
18. Try to run sample programs and **clinfo** as described in step no. 7 above.

Known issues

19. If you are getting some **permission denied** message it means the user is not having the rights to access the AMD APP SDK installed on Linux machine. For that you need to assign the permission to each user (to access the AMD APP package you extracted in step no. 1 above) by using **chown** command on the Linux machine.
20. On the client machines, you should also check the variables we exported on the Linux machine, by using echo command as described in step no 2 to 5. If the echo is not returning proper paths, you should set it using export command as described in step 2 to 5 above.

Running your own OpenCl program without using SDK utilities

21. So now you know how to run programs in the SDK. In the SDK you will find a sample **Template** and **TemplateC**, which teaches how to compile your own openCl programs and what all libraries are actually needed if one is not using SDK utilities (yes samples in SDK uses some special utilities meant for the SDK).
22. Go to the terminal and type vim first.cpp or vi first.cpp. This will open a new text file where you will be writing your OpenCL code. As a demonstration simply copy and paste the following program in the new .cpp file:

Note that we have not included any SDK specific utility in the following code

```
#include <iostream>

using namespace std;

#define __NO_STD_VECTOR // Use cl::vector and cl::string and
#define __NO_STD_STRING // not STL versions, more on this later
#include <CL/cl.h>


#define DATA_SIZE (1024*1240)

const char *KernelSource = "\n"                        \
  "__kernel void square(                    \n" \
  "   __global float* input,                \n" \
  "   __global float* output,               \n" \
  "   const unsigned int count)             \n" \
  "{                                        \n" \
  "   int i = get_global_id(0);             \n" \
  "   if(i < count)                         \n" \
  "       output[i] = input[i] * input[i];  \n" \
  "}                                        \n" \
  "\n";

int main(int argc, char* argv[])
{
  int devType=CL_DEVICE_TYPE_GPU;

  if(argc > 1) {
    devType = CL_DEVICE_TYPE_CPU;
    cout << "Using: CL_DEVICE_TYPE_CPU" << endl;
  } else {
    cout << "Using: CL_DEVICE_TYPE_GPU" << endl;
  }

  cl_int err;     // error code returned from api calls

  size_t global;  // global domain size for our calculation
  size_t local;   // local domain size for our calculation

  cl_platform_id cpPlatform; // OpenCL platform
  cl_device_id device_id;    // compute device id
  cl_context context;        // compute context
  cl_command_queue commands; // compute command queue
  cl_program program;        // compute program
  cl_kernel kernel;          // compute kernel

  // Connect to a compute device
  err = clGetPlatformIDs(1, &cpPlatform, NULL);
  if (err != CL_SUCCESS) {
    cerr << "Error: Failed to find a platform!" << endl;
    return EXIT_FAILURE;
  }

  // Get a device of the appropriate type
  err = clGetDeviceIDs(cpPlatform, devType, 1, &device_id, NULL);
  if (err != CL_SUCCESS) {
    cerr << "Error: Failed to create a device group!" << endl;
    return EXIT_FAILURE;
  }

  // Create a compute context
  context = clCreateContext(0, 1, &device_id, NULL, NULL, &err);
  if (!context) {
    cerr << "Error: Failed to create a compute context!" << endl;
    return EXIT_FAILURE;
  }

  // Create a command commands
  commands = clCreateCommandQueue(context, device_id, 0, &err);
```

```
    if (!commands) {
      cerr << "Error: Failed to create a command commands!" << endl;
      return EXIT_FAILURE;
    }

    // Create the compute program from the source buffer
    program = clCreateProgramWithSource(context, 1,
                                        (const char **) &KernelSource,
                                        NULL, &err);
    if (!program) {
      cerr << "Error: Failed to create compute program!" << endl;
      return EXIT_FAILURE;
    }

    // Build the program executable
    err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
    if (err != CL_SUCCESS) {
      size_t len;
      char buffer[2048];

      cerr << "Error: Failed to build program executable!" << endl;
      clGetProgramBuildInfo(program, device_id, CL_PROGRAM_BUILD_LOG,
                            sizeof(buffer), buffer, &len);
      cerr << buffer << endl;
      exit(1);
    }

    // Create the compute kernel in the program
    kernel = clCreateKernel(program, "square", &err);
    if (!kernel || err != CL_SUCCESS) {
      cerr << "Error: Failed to create compute kernel!" << endl;
      exit(1);
    }

    // create data for the run
    float* data = new float[DATA_SIZE];    // original data set given to device
    float* results = new float[DATA_SIZE]; // results returned from device
    unsigned int correct;                  // number of correct results returned
    cl_mem input;                          // device memory used for the input
  array
    cl_mem output;                         // device memory used for the output
  array

    // Fill the vector with random float values
    unsigned int count = DATA_SIZE;
    for(int i = 0; i < count; i++)
      data[i] = rand() / (float)RAND_MAX;

    // Create the device memory vectors
    //
    input = clCreateBuffer(context,  CL_MEM_READ_ONLY,
                           sizeof(float) * count, NULL, NULL);
    output = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
                            sizeof(float) * count, NULL, NULL);
    if (!input || !output) {
      cerr << "Error: Failed to allocate device memory!" << endl;
      exit(1);
    }

    // Transfer the input vector into device memory
    err = clEnqueueWriteBuffer(commands, input,
                              CL_TRUE, 0, sizeof(float) * count,
                              data, 0, NULL, NULL);
    if (err != CL_SUCCESS) {
      cerr << "Error: Failed to write to source array!" << endl;
      exit(1);
    }

    // Set the arguments to the compute kernel
    err = 0;
    err  = clSetKernelArg(kernel, 0, sizeof(cl_mem), &input);
    err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &output);
    err |= clSetKernelArg(kernel, 2, sizeof(unsigned int), &count);
    if (err != CL_SUCCESS) {
      cerr << "Error: Failed to set kernel arguments! " << err << endl;
      exit(1);
    }

    // Get the maximum work group size for executing the kernel on the device
    err = clGetKernelWorkGroupInfo(kernel, device_id,
                                   CL_KERNEL_WORK_GROUP_SIZE,
                                   sizeof(local), &local, NULL);
    if (err != CL_SUCCESS) {
      cerr << "Error: Failed to retrieve kernel work group info! "
           <<  err << endl;
      exit(1);
    }

  // Execute the kernel over the vector using the
  // maximum number of work group items for this device
    global = count;
    err = clEnqueueNDRangeKernel(commands, kernel,
                                1, NULL, &global, &local,
                                0, NULL, NULL);
    if (err) {
      cerr << "Error: Failed to execute kernel!" << endl;
      return EXIT_FAILURE;
    }

    // Wait for all commands to complete
    clFinish(commands);

    // Read back the results from the device to verify the output
    //
    err = clEnqueueReadBuffer( commands, output,
                              CL_TRUE, 0, sizeof(float) * count,
                              results, 0, NULL, NULL );
    if (err != CL_SUCCESS) {
      cerr << "Error: Failed to read output array! " <<  err << endl;
      exit(1);
    }

    // Validate our results
    //
    correct = 0;
    for(int i = 0; i < count; i++) {
      if(results[i] == data[i] * data[i])
        correct++;
    }

    // Print a brief summary detailing the results
    cout << "Computed " << correct << "/" << count << " correct values" <<
  endl;
    cout << "Computed " << 100.f * (float)correct/(float)count
         << "% correct values" << endl;

    // Shutdown and cleanup
    delete [] data; delete [] results;

    clReleaseMemObject(input);
    clReleaseMemObject(output);
    clReleaseProgram(program);
    clReleaseKernel(kernel);
    clReleaseCommandQueue(commands);
    clReleaseContext(context);

    return 0;
  }
```

You can also download this code from this link:
http://www.codeproject.com/KB/showcase/Portable-Parallelism.aspx

Now to compile and run this code, type the following commands:

```
# OCL_HOME=../ati-stream-sdk-v2.4-lnx32
# g++ -I $OCL_HOME/include -L $OCL_HOME/lib/x86 first.cpp -l OpenCL
```

The first command sets the OCL_HOME to the SDK where you have extracted it in step 1, while the second command is for compiling.

23. Type **./a.out** to see the output
24. If you get correct output, without any error, it means you have successfully run you first OpenCL program. You can test this program from any of the clients running windows. If you have followed the instructions carefully, hopefully your programs will be compiled and execute successfully

And your lab is now ready for OpenCL programming !!

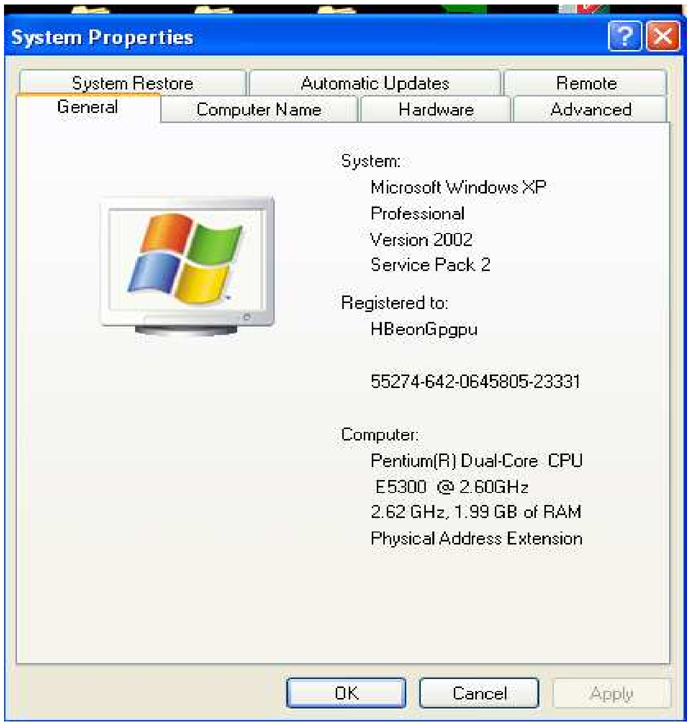
B- You have only windows machines

In the previous section we described the procedure for setting up lab in case we have a single machine with GPU running Linux. In this section we will see what can be done in case we have all the systems running Windows -7.

1. Download and install the AMD APP SDK, and the necessary drivers. Drivers can be downloaded from http://support.amd.com/us/gpudownload/Pages/index.aspx
2. You will not need any ICD registration, as we did in case of Linux in the previous section.
3. Make sure that you are fulfilling all the pre-requisites for an issue less installation. **Windows XP service pack 2 is not supported**, so it will not work as intended.
4. It is best you have windows 7 systems and MS Visual studio 2008.
5. Additional information regarding installation can be seen here: http://developer.amd.com/sdks/AMDAPPSDK/assets/AMD_APP_SDK_Installation_Notes.pdf

   After the installation you will have to set some environmental variable. On an XP SP 3 machine follow this:

   a. Right click on My Computer and click properties. This will display the following window.



System Properties

System Restore | Automatic Updates | Remote
General | Computer Name | Hardware | Advanced

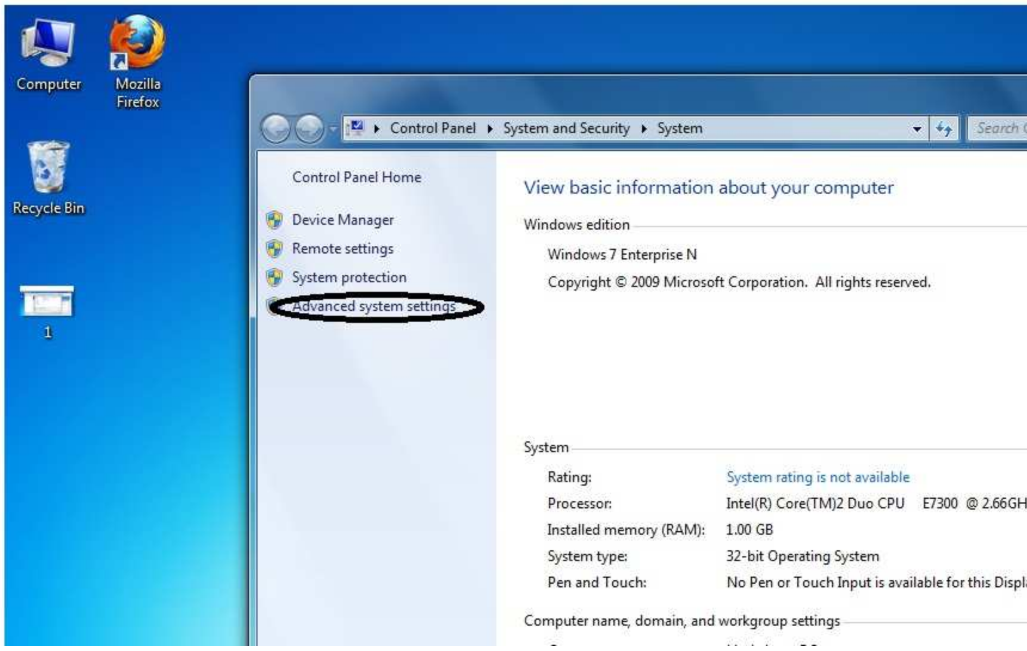
System:
Microsoft Windows XP
Professional
Version 2002
Service Pack 2

Registered to:
HBeonGpgpu

55274-642-0645805-23331

Computer:
Pentium(R) Dual-Core CPU
E5300 @ 2.60GHz
2.62 GHz, 1.99 GB of RAM
Physical Address Extension

on window 7 this will appear like this:



Control Panel Home

Device Manager
Remote settings
System protection
[✓ Advanced system settings]

View basic information about your computer

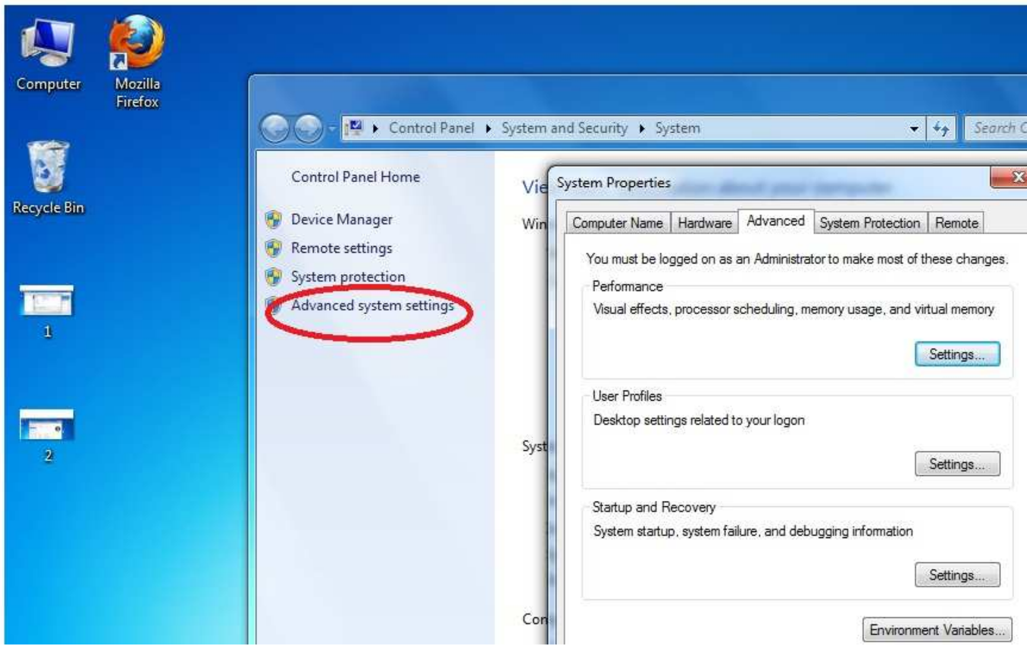
Windows edition

Windows 7 Enterprise N
Copyright © 2009 Microsoft Corporation. All rights reserved.

System

Rating:	System rating is not available
Processor:	Intel(R) Core(TM)2 Duo CPU E7300 @ 2.66GH
Installed memory (RAM):	1.00 GB
System type:	32-bit Operating System
Pen and Touch:	No Pen or Touch Input is available for this Displ

Computer name, domain, and workgroup settings



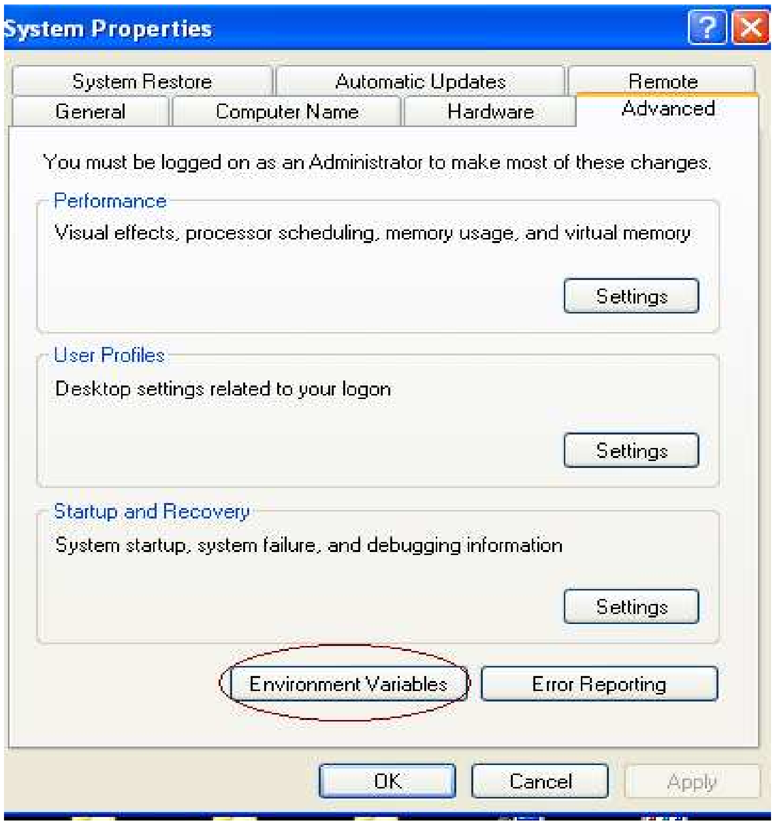
You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information

b. Now click Advance tab. You will see the following window:



You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

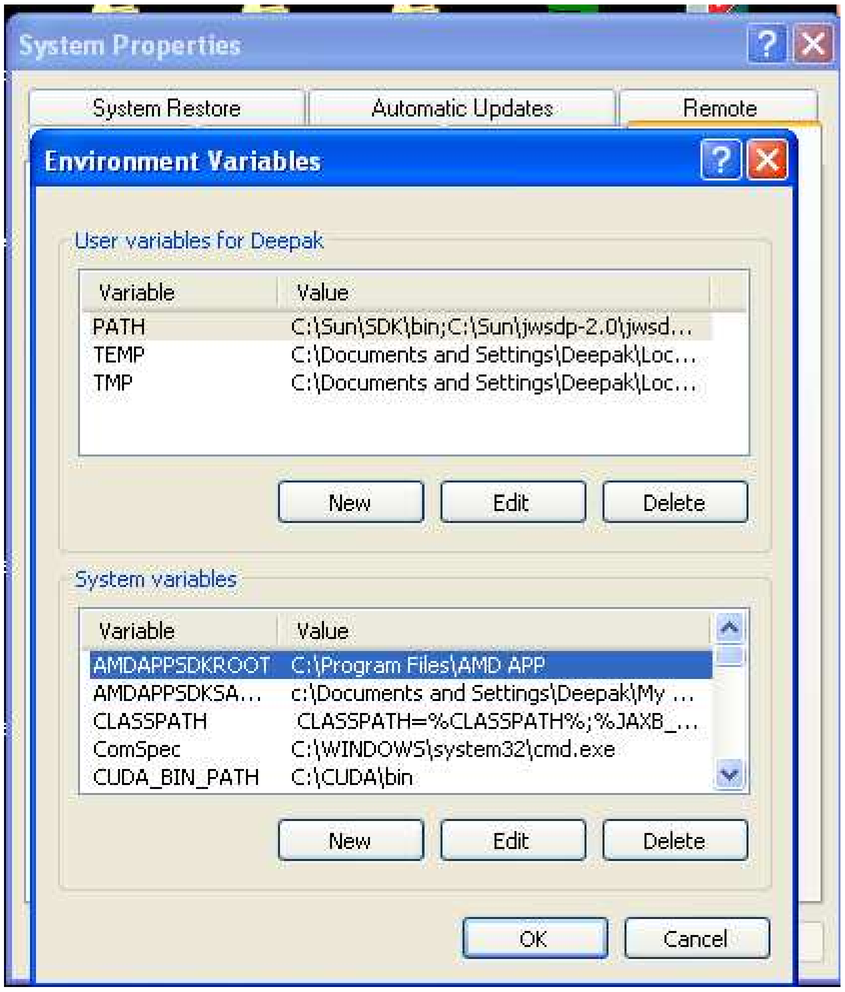
User Profiles

Desktop settings related to your logon

Startup and Recovery

System startup, system failure, and debugging information

c. Click the Environmental Variable Tab. This will display the following window:



System Properties

System Restore | Automatic Updates | Remote

Environment Variables

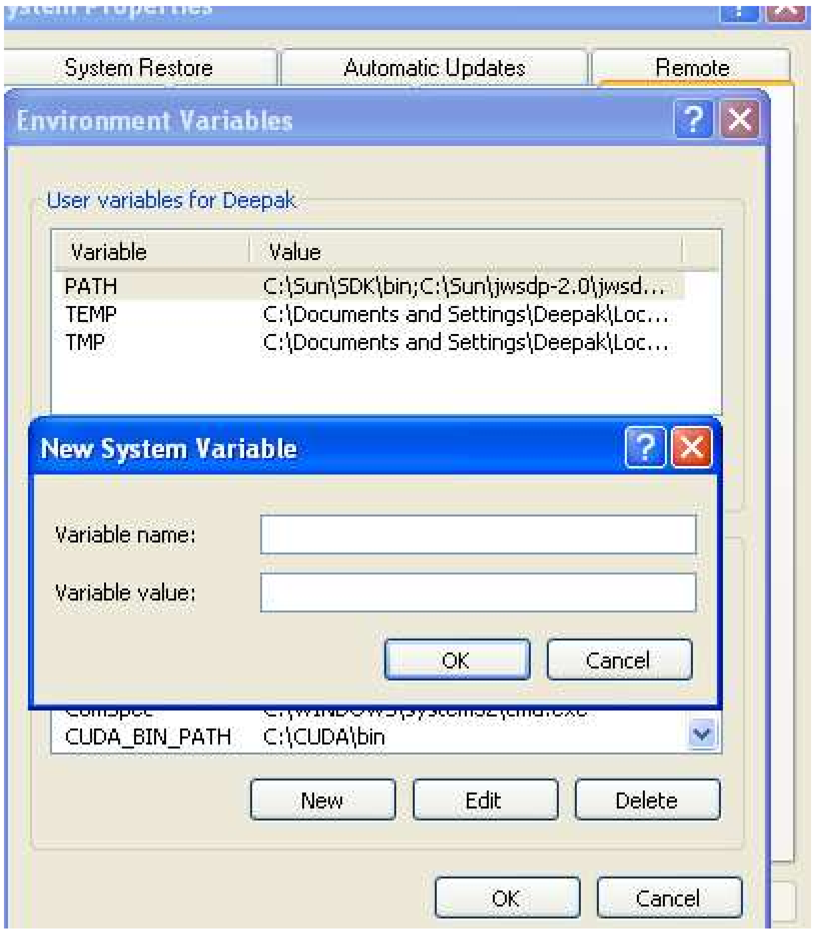
User variables for Deepak

Variable	Value
PATH	C:\Sun\SDK\bin;C:\Sun\jwsdp-2.0\jwsd...
TEMP	C:\Documents and Settings\Deepak\Loc...
TMP	C:\Documents and Settings\Deepak\Loc...

System variables

Variable	Value
AMDAPPSDKROOT	C:\Program Files\AMD APP
AMDAPPSDKSA...	c:\Documents and Settings\Deepak\My ...
CLASSPATH	CLASSPATH=%CLASSPATH%;%JAXB_...
ComSpec	C:\WINDOWS\system32\cmd.exe
CUDA_BIN_PATH	C:\CUDA\bin

d. You can see there are two editable fields: User Variable, and System Variable. Now you have to first set two variables, namely AMDAPPSDKROOT and AMDAPPSDKSAMPLESROOT. **In your case you will not see the two variables in System variables, which is what you need to set now. For this click New. You will see a window like this:**



Variable	Value
PATH	C:\Sun\SDK\bin;C:\Sun\jwsdp-2.0\jwsd...
TEMP	C:\Documents and Settings\Deepak\Loc...
TMP	C:\Documents and Settings\Deepak\Loc...

Variable name:

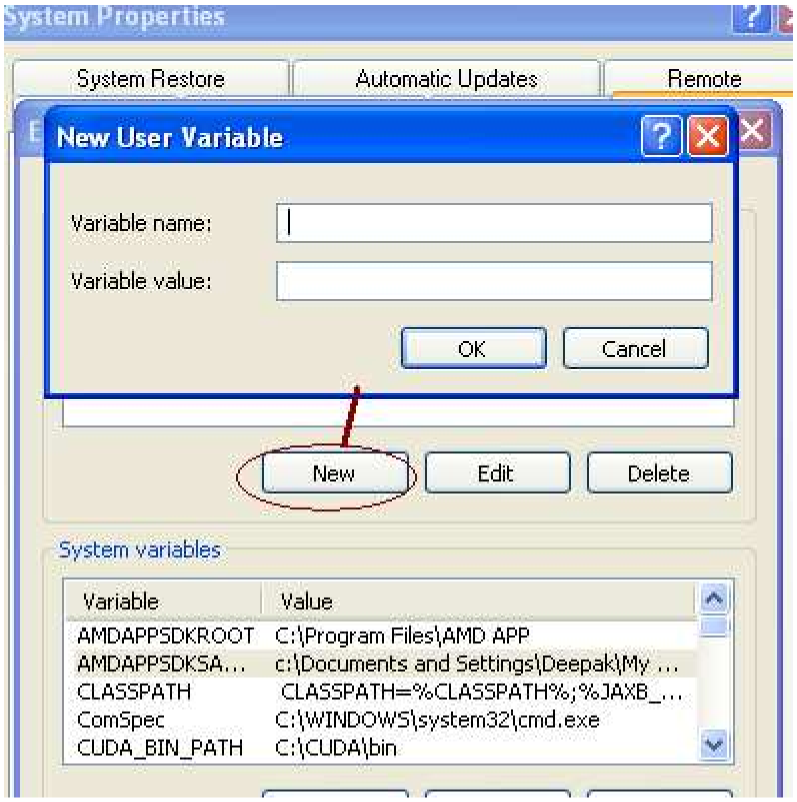
Variable value:

CUDA_BIN_PATH	C:\CUDA\bin

Enter the variable name AMDAPPSDKROOT, and then the variable value as **C:\Program Files\AMD APP** (your installation path) and click OK.

e. Repeat the step-d for the variable AMDAPPSDKSAMPLESROOT. This time however you have to put the following variable value: **c:\Documents and Settings\username\My Documents\AMD APP**. Basically this is the path of your SDK samples. By default it will be My Document. If you have not done any mistakes you will find the variable name and value in the system variable section.

f. Now the System Variables are set. You should now set the PATH variable.

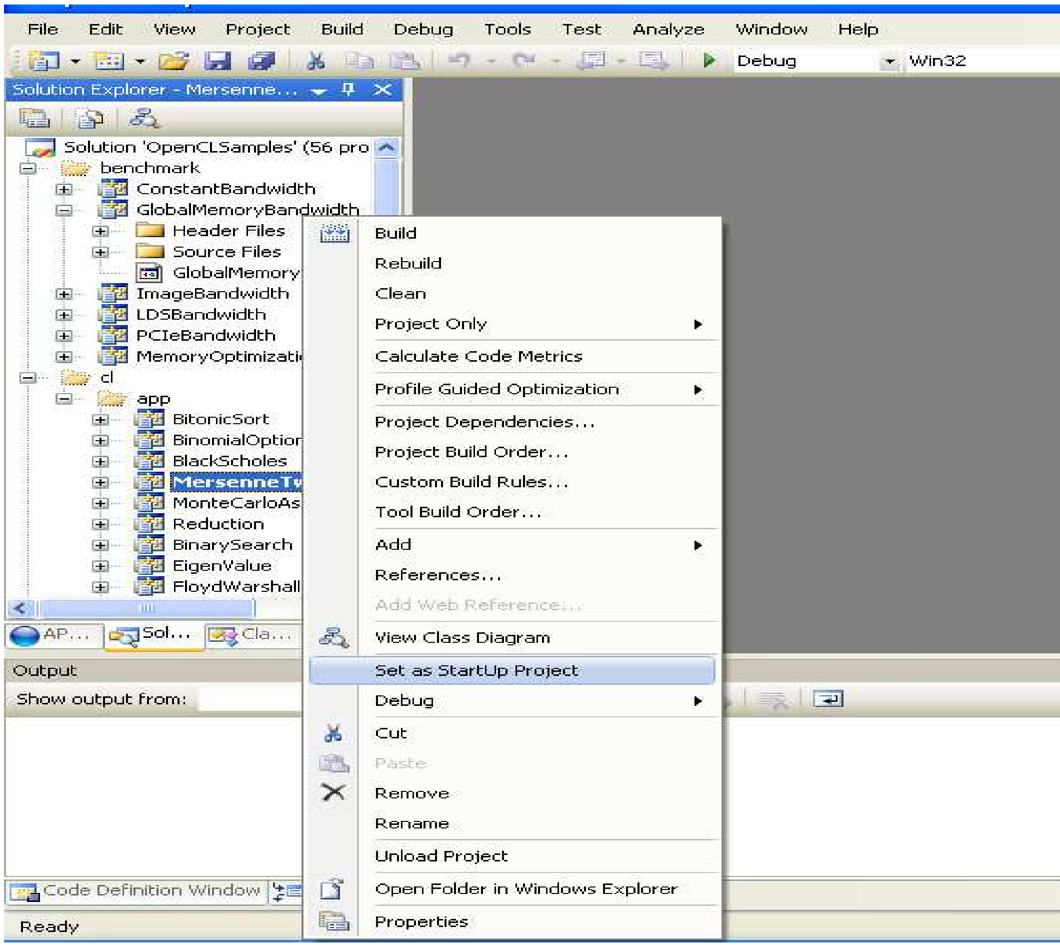


Variable	Value
AMDAPPSDKROOT	C:\Program Files\AMD APP
AMDAPPSDKSA...	c:\Documents and Settings\Deepak\My ...
CLASSPATH	CLASSPATH=%CLASSPATH%;%JAXB_...
ComSpec	C:\WINDOWS\system32\cmd.exe
CUDA_BIN_PATH	C:\CUDA\bin

Click New in user variable section. And set the variable as PATH and variable value as **;$(AMDAPPSDKROOT)\bin\x86 (for 32-bit systems), or ;$(AMDAPPSDKROOT)\bin\x86_64 (for 64-bit systems).** Note that ';' used before the values because variable values are separated by ';'. In the same Variable value you also need to put the following lines:
;$(AMDAPPSDKSAMPLESROOT)\bin\x86 (for 32-bit systems) or ;$(AMDAPPSDKSAMPLESROOT)\bin\x86_64 (for 64-bit systems).
So for a 32 bit system you will enter exactly the following in the variable value:
xyz;$(AMDAPPSDKROOT)\bin\x86;$(AMDAPPSDKSAMPLESROOT)\bin\x86
Where, xyz is some already set path variable value.

g. Now that you have successfully set the variables. Its time for testing the SDK samples. Go to your \AMD APP\samples\opencl folder in My Document. And double click OpenCLSamples MSVS solution. This will launch the Visual Studio and will start loading all the projects. Once the loading is over you will see a list of projects in the solution explorer.

You can build the entire project or select any of the projects and set to startup project as shown below:



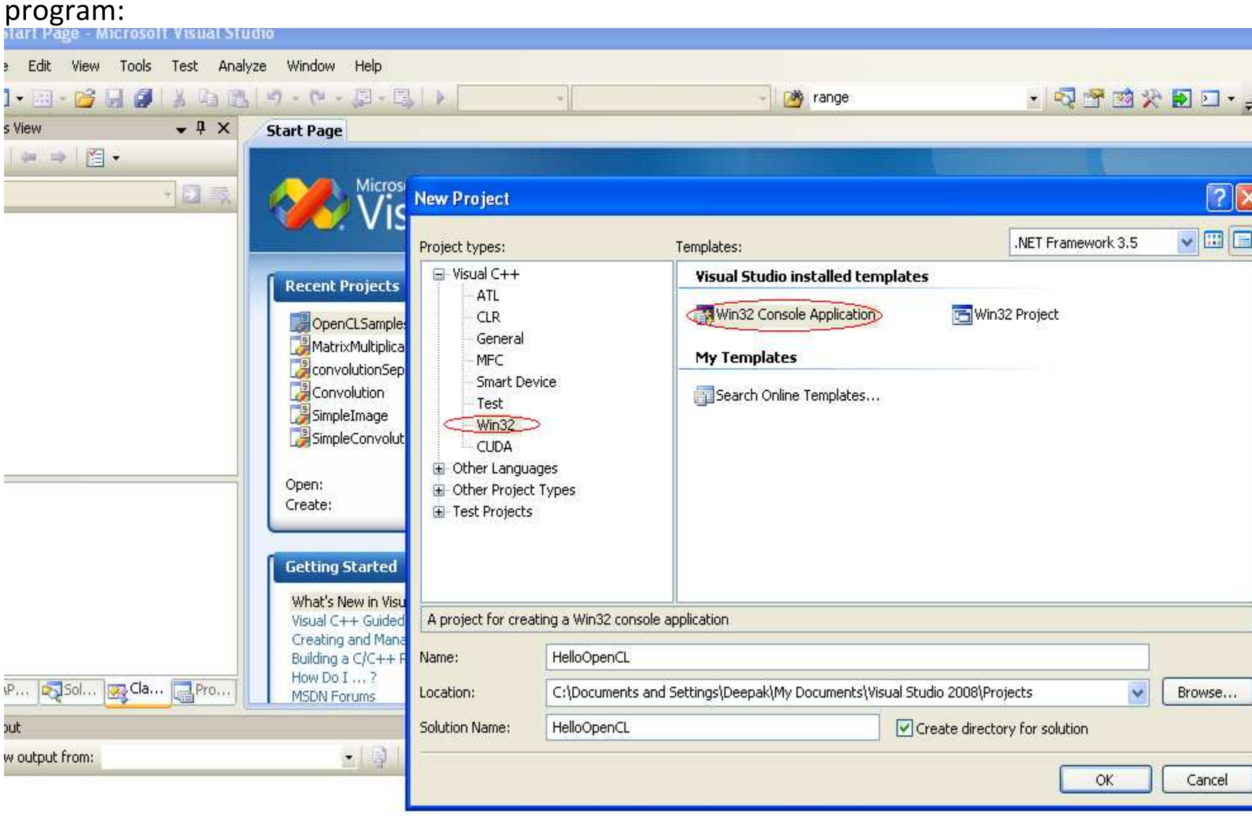
As you can see we have selected Mersenne Twister as an startup project, which you can compile and run.

h. You can also run the executable through command line. Click Stat and then type in run : cmd , this will open the dos command prompt. To execute any particular binary of you OpenCL SDK samples, go to the folder where the binaries are present, some this like this: cd C:\Documents and Settings\Deepak\My Documents\AMD APP\samples\opencl\bin\x86. By typing dir you can list all the files present in this folder (\bin\x86). Simply type the name of the executable and hit enter. For example to execute matrix multiplication simply type: **MatrixMultiplication**.

7. Every sample project has instructions for executing the binaries, and optional flags. For example if you go to matrix multiplication sample, you will find a pdf file inside the doc folder. This gives you various options, such as you can get the execution time by typing **./matrixmultiplication -t**
8. An important utility is provided for querying the devices and getting info about opencl platform installed on the system. This tool can be helpful to check if your system is detecting all the opencl devices or not. This file **clinfo** is present in **C:\Program Files\AMD APP\bin\x86**. Go to that file by typing cd command and type **clinfo** or double click the file in the above folder**.** This will display the information about the openCl system you have. See step 7(iv) in Linux section above.

Running your own OpenCL Code

9. Now we will see how we compile and run our own openCl program, without using the SDK utilities. Follow the following step to write a new OpenCl program:



A project for creating a Win32 console application

Name: HelloOpenCL

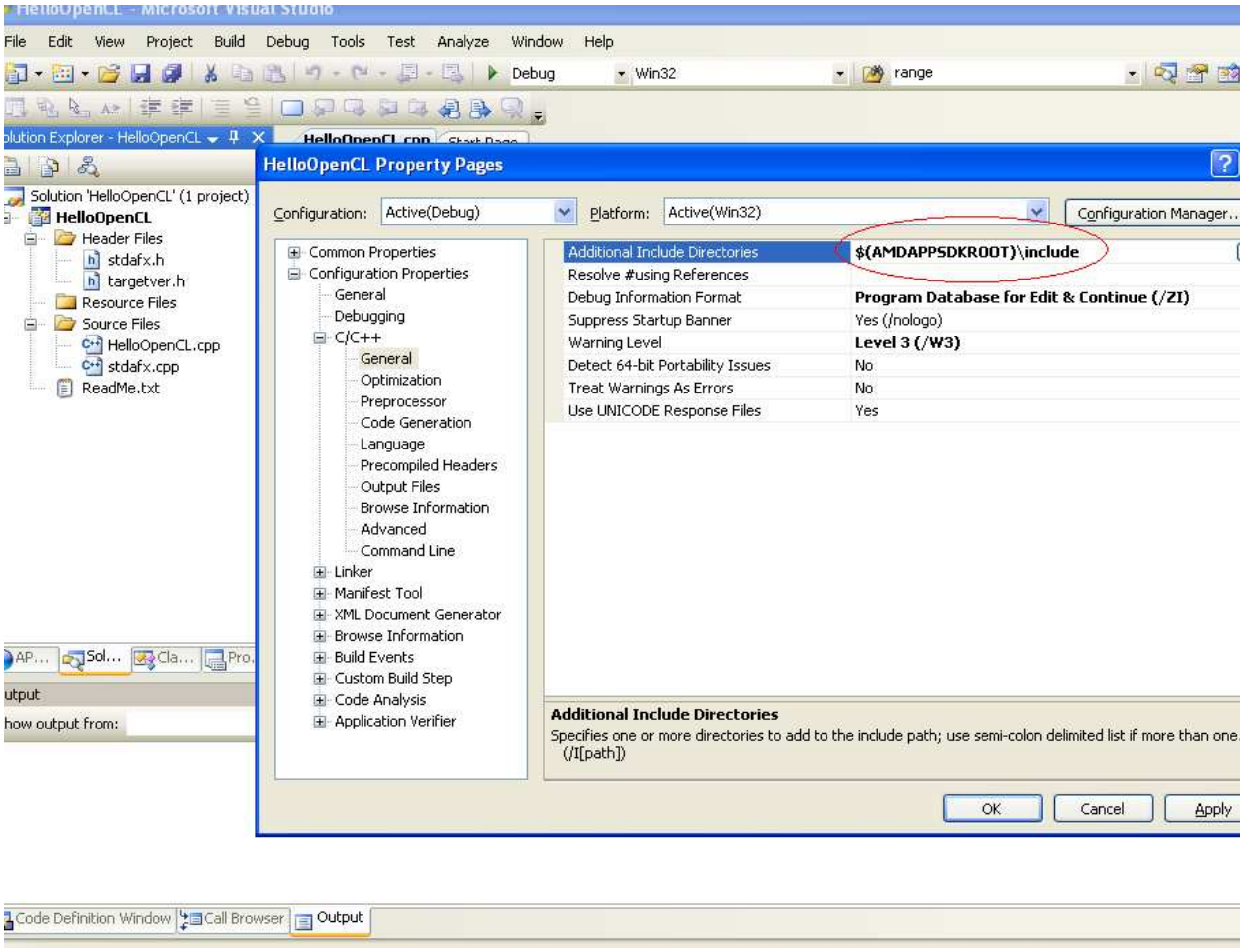
Location: C:\Documents and Settings\Deepak\My Documents\Visual Studio 2008\Projects

Solution Name: HelloOpenCL

✓ Create directory for solution

10. Click ok and Finish
11. Right click on project and go to properties

12. Go to C/C++ -> General and in **additional include directories** type: $(AMDAPPSDKROOT)\include

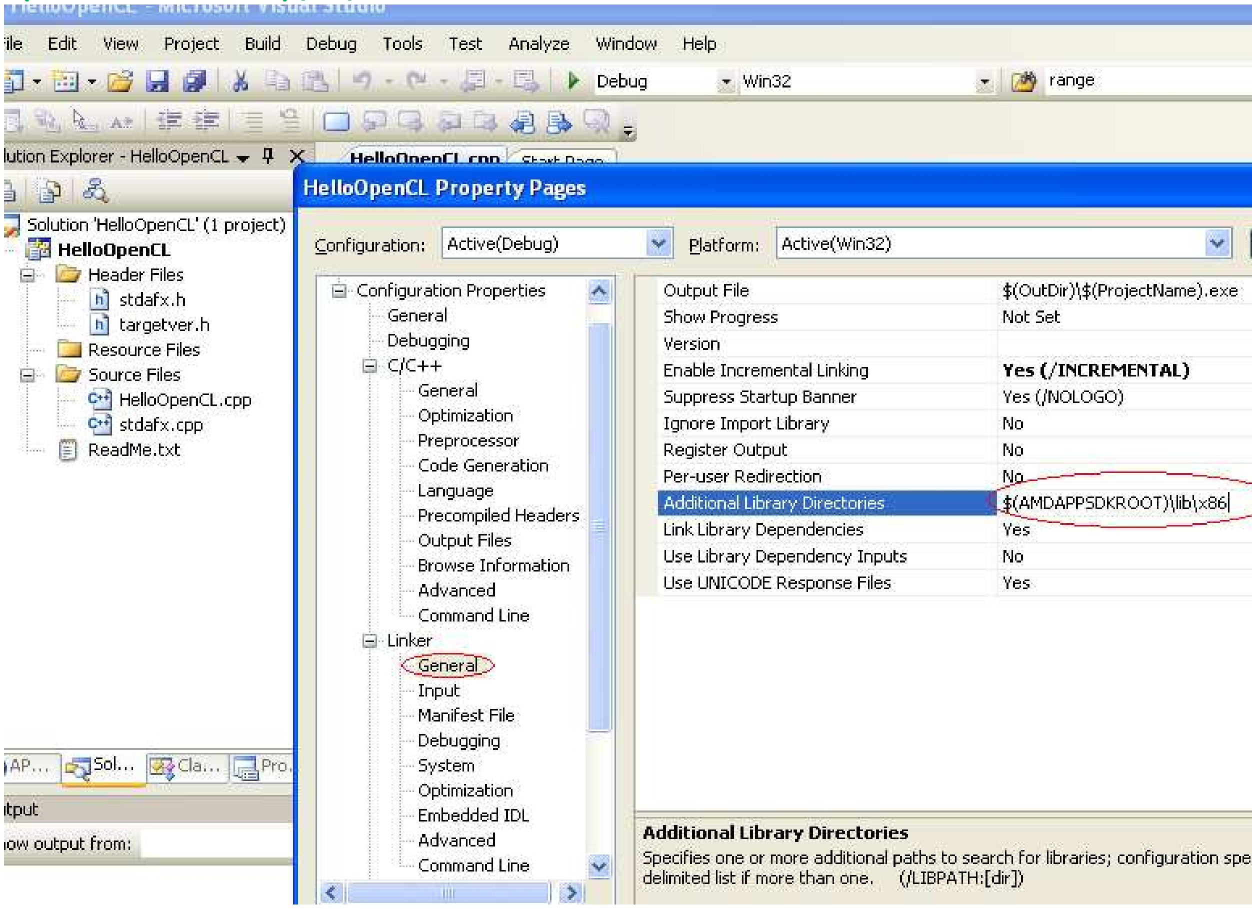


Configuration: Active(Debug)
Platform: Active(Win32)

Additional Include Directories	$(AMDAPPSDKROOT)\include
Resolve #using References	
Debug Information Format	Program Database for Edit & Continue (/ZI)
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level 3 (/W3)
Detect 64-bit Portability Issues	No
Treat Warnings As Errors	No
Use UNICODE Response Files	Yes

Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])

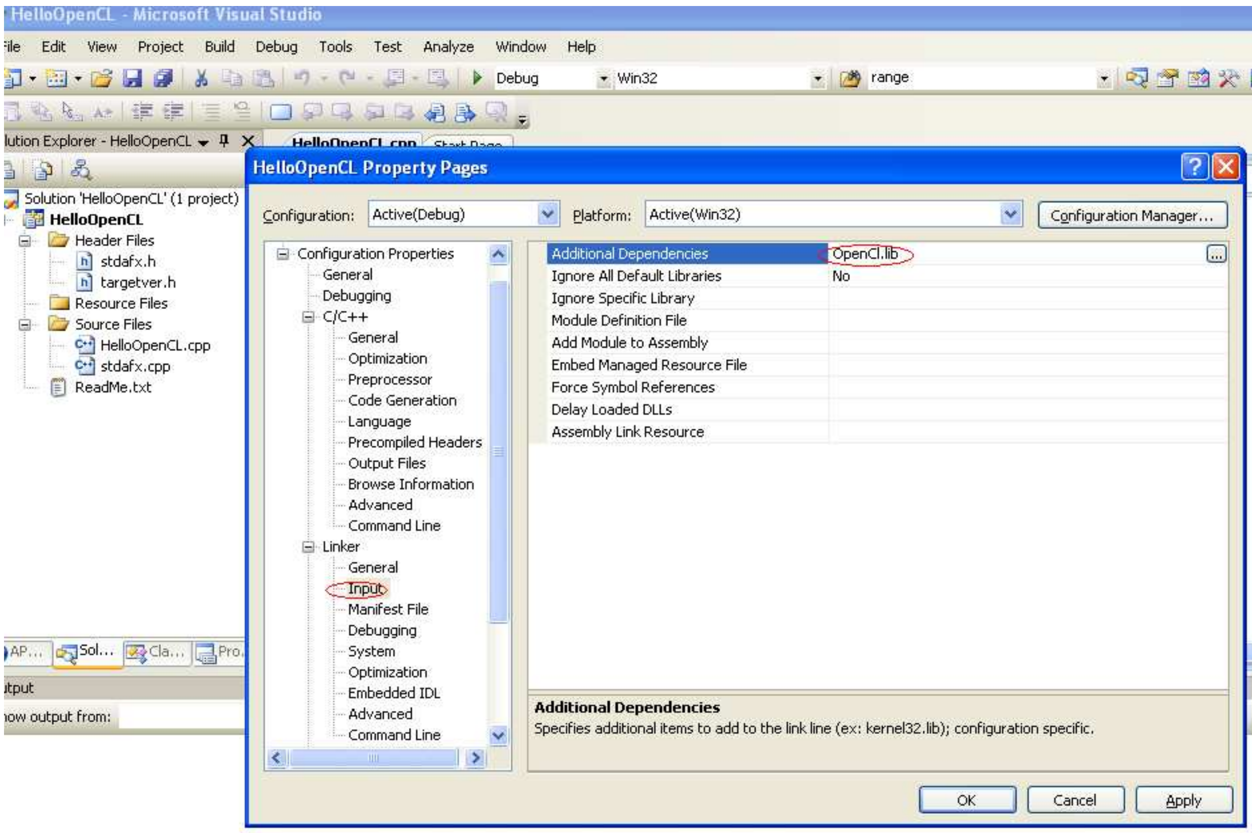
13. Go to the Linker -> General tab and in **additional include directories** type **$(AMDAPPSDKROOT)\lib\x86**



Output File	$(OutDir)\$(ProjectName).exe
Show Progress	Not Set
Version	
Enable Incremental Linking	Yes (/INCREMENTAL)
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	$(AMDAPPSDKROOT)\lib\x86
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Use UNICODE Response Files	Yes

Additional Library Directories
Specifies one or more additional paths to search for libraries; configuration spe
delimited list if more than one. (/LIBPATH:[dir])

14. In the linker Input page type **OpenCl.lib** in Additional Dependencies List:



Configuration: Active(Debug) Platform: Active(Win32)

Additional Dependencies	OpenCl.lib
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

Apply these settings and now you can run your code straight away. Test the source code given in the Linux section to see if everything is working fine. Also in the SDK look at Template and TemplateC samples, which will provide you use full pointers.

Npte: What will you do If you have all the machines running Winodws 7 and have only one graphics card? **<u>In such a case you should install SDK on all the machines</u>** and use **<u>Putty for Dos</u>** to connect and run programs on the machine having graphics card. Basically the client machines (with no GPU), will be connected to the machine with GPU (I,e server),and you will be running and compiling your code on the server from the clients using the putty client. Also since all the clients will be running the SDK, so the server can be used only for timing or testing purpose as needed. This can be another alternative in case server fails due to some reason.