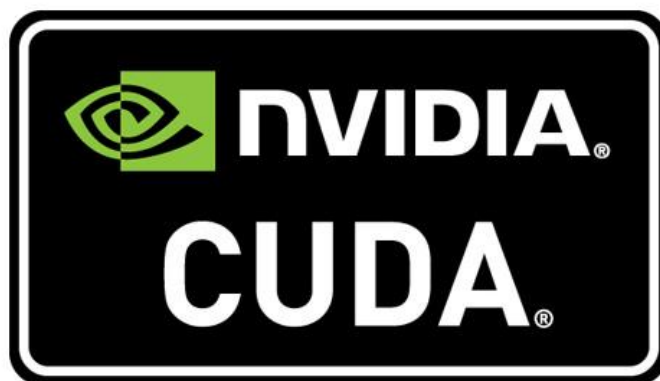


# **APPLIED PARALLEL PROGRAMMING ON GPU**

## **Lab. Tutorial**



**CUDA Installation and Introduction to CUDA**

## Contents

1. What is CUDA? .....	3
2. Installation of CUDA on Windows.....	3
2.1. Requirements .....	3
2.2. Installing CUDA Software .....	4
2.3. Verify the Installation .....	5
2.4. Working with Visual Studio - CUDA 5.0 or newer .....	8
3. A Simple Example.....	12
4. Lab. Exercises .....	14
5. Assignment 1: Matrix Multiplication .....	15
6. References .....	15

## 1. What is CUDA?

CUDA™ is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

## 2. Installation of CUDA on Windows

### 2.1. Requirements

#### a. CUDA-Capable GPU

Check your GPU from Device Manager. If it's an NVIDIA card that is listed in <https://developer.nvidia.com/cuda-gpus>, it means that your GPU is CUDA-capable. **However**, NVIDIA doesn't support GPUs with CUDA compute capability less than 2.0 anymore so if you can't see the GPU in the list, you should check <https://developer.nvidia.com/cuda-legacy-gpus> or search on the internet. You may have a CUDA-capable GPU with compute capability less than 2.0!

#### b. Microsoft Windows XP, Vista, 7, 8 or 10

#### c. NVIDIA CUDA Toolkit

The NVIDIA CUDA Toolkit contains the driver and tools needed to create, build and run a CUDA application as well as libraries, header files, CUDA samples source code, and other resources.

Toolkit is available free at

<https://developer.nvidia.com/cuda-downloads>

CUDA CC \	< 6.5	6.5	7.5	8.0	9.X	10.X
Default	1.0	2.0	2.0	2.0	3.0	3.5
Support	1.0	1.1	2.0 – 5.X	2.0 – 6.X	3.0 – 7.X	3.0 – 7.X

#### d. Microsoft Visual Studio 2008-2017 or Microsoft Visual Studio Express (Community)

Express (Community) version is available free at

<https://www.visualstudio.com/en-US/products/visual-studio-express-vs>

CUDA	< 6.5	6.5	8.0	9.X	10.X
VS	< 2012	2008 - 2013	2008 - 2015	2010 - 2017	2012-2017

## 2.2. Installing CUDA Software

Install the CUDA Toolkit by executing the Toolkit installer and following the on-screen prompts.

**Note:** The driver and toolkit must be installed for CUDA to function. If you have not installed a stand-alone driver, install the driver from the NVIDIA CUDA Toolkit.

You can choose what to install from the following packages:

### 1. CUDA Driver

The CUDA Driver installation can be done silently or by using a GUI. A silent installation of the driver is done by enabling that feature when choosing what to install.

- Silent: Only the display driver will be installed.
- GUI: A window will appear after the CUDA Toolkit installation if you allowed it at the last dialog with the full driver installation UI. You can choose which features you wish to install.

### 2. CUDA Toolkit

The CUDA Toolkit installation defaults to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v#.#, where #.# is version number 3.2 or higher. This directory contains the following:

Bin\

the compiler executables and runtime libraries

Include\

the header files needed to compile CUDA programs

Lib\

the library files needed to link CUDA programs

Doc\

the CUDA C Programming Guide, CUDA C Best Practices Guide, documentation for the CUDA libraries, and other CUDA Toolkit-related documentation

**Note:** CUDA Toolkit versions 3.1 and earlier installed into C:\CUDA by default, requiring prior CUDA Toolkit versions to be uninstalled before the installation of new

versions. Beginning with CUDA Toolkit 3.2, multiple CUDA Toolkit versions can be installed simultaneously.

### 3. CUDA Samples

The CUDA Samples contain source code for many example problems and templates with Microsoft Visual Studio projects.

For Windows XP, the samples can be found:

C:\Documents and Settings\All Users\Application Data\NVIDIA Corporation\CUDA Samples\v5.0

For Windows Vista, Windows 7, and Windows Server 2008, the samples can be found:

C:\ProgramData\NVIDIA Corporation\CUDA Samples\v5.0

**Note:** The NVIDIA CUDA Toolkit installer only installs Visual Studio project templates for toolkit version 5.0 and higher. Installing NVIDIA® Nsight™, Visual Studio Edition will install Visual Studio project templates for toolkit versions earlier than CUDA 5.0.

## 2.3. Verify the Installation

Before continuing, it is important to verify that the CUDA programs can find and communicate correctly with the CUDA-capable hardware. To do this, you need to compile and run some of the included sample programs.

### Running the Compiled Examples

The version of the CUDA Toolkit can be checked by running `nvcc -V` in a Command Prompt window. You can display a Command Prompt window by going to:

Start > All Programs > Accessories > Command Prompt

CUDA Samples include sample programs in both source and compiled form. To verify a correct configuration of the hardware and software, it is highly recommended that you run the `deviceQuery` program located here:

Windows XP:

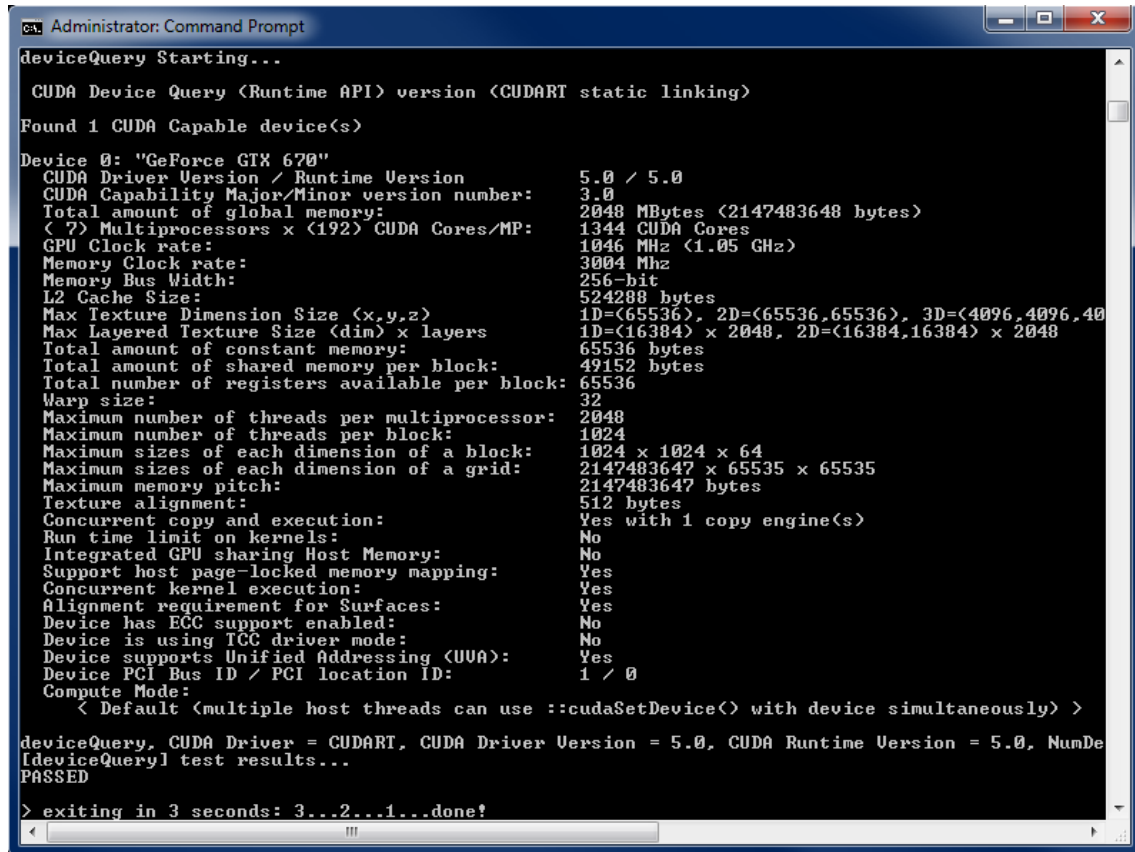
C:\Documents and Settings\All Users\Application Data\NVIDIA Corporation\CUDA Samples\v5.5\C\bin\win32\Release

Windows Vista, Windows 7, Windows 8, Windows Server 2003, and Windows Server 2008:

C:\ProgramData\NVIDIA Corporation\CUDA Samples\v5.5\C\bin\win32\Release

This assumes that you used the default installation directory structure. (On 64-bit versions of Windows, the directory name ends with \win64\Release.) If CUDA is installed and configured correctly, the output should look similar to Figure 1.

Figure 1. Valid Results from Sample CUDA Device Query Program



```

Administrator: Command Prompt
deviceQuery Starting...

  CUDA Device Query (Runtime API) version (CUDART static linking)

Found 1 CUDA Capable device(s)

Device 0: "GeForce GTX 670"
  CUDA Driver Version / Runtime Version      5.0 / 5.0
  CUDA Capability Major/Minor version number: 3.0
  Total amount of global memory:              2048 MBytes (2147483648 bytes)
  < ? > Multiprocessors x <192> CUDA Cores/MP: 1344 CUDA Cores
  GPU Clock rate:                            1046 MHz (1.05 GHz)
  Memory Clock rate:                         3004 Mhz
  Memory Bus Width:                          256-bit
  L2 Cache Size:                             524288 bytes
  Max Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536,65536), 3D=(4096,4096,4096)
  Max Layered Texture Size (dim) x layers    1D=(16384) x 2048, 2D=(16384,16384) x 2048
  Total amount of constant memory:            65536 bytes
  Total amount of shared memory per block:    49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                 32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:        1024
  Maximum sizes of each dimension of a block: 1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:  2147483647 x 65535 x 65535
  Maximum memory pitch:                      2147483647 bytes
  Texture alignment:                         512 bytes
  Concurrent copy and execution:              Yes with 1 copy engine(s)
  Run time limit on kernels:                  No
  Integrated GPU sharing Host Memory:          No
  Support host page-locked memory mapping:    Yes
  Concurrent kernel execution:                Yes
  Alignment requirement for Surfaces:         Yes
  Device has ECC support enabled:              No
  Device is using ICC driver mode:             No
  Device supports Unified Addressing (UVA):    Yes
  Device PCI Bus ID / PCI location ID:        1 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 5.0, CUDA Runtime Version = 5.0, NumDe
[deviceQuery] test results...
PASSED

> exiting in 3 seconds: 3...2...1...done!
  
```

The exact appearance and the output lines might be different on your system. The important outcomes are that a device was found, that the device(s) match what is installed in your system, and that the test passed.

If a CUDA-capable device and the CUDA Driver are installed but deviceQuery reports that no CUDA-capable devices are present, ensure the device and driver are properly installed.

Running the bandwidthTest program, located in the same directory as deviceQuery above, ensures that the system and the CUDA-capable device are able to communicate correctly. The output should resemble Figure 2.

Figure 2. Valid Results from Sample CUDA Bandwidth Test Program

The device name (second line) and the bandwidth numbers vary from system to system. The important items are the second line, which confirms a CUDA device was found, and the second-to-last line, which confirms that all necessary tests passed.

If the tests do not pass, make sure you do have a CUDA-capable NVIDIA GPU on your system and make sure it is properly installed.

To see a graphical representation of what CUDA can do, run the sample Particles executable in:

For Windows XP:

c:\Documents and Settings\All Users\Application Data\CUDA  
Samples\v5.5\C\bin\win32\Release

(or ... \win64\Release on 64-bit Windows)

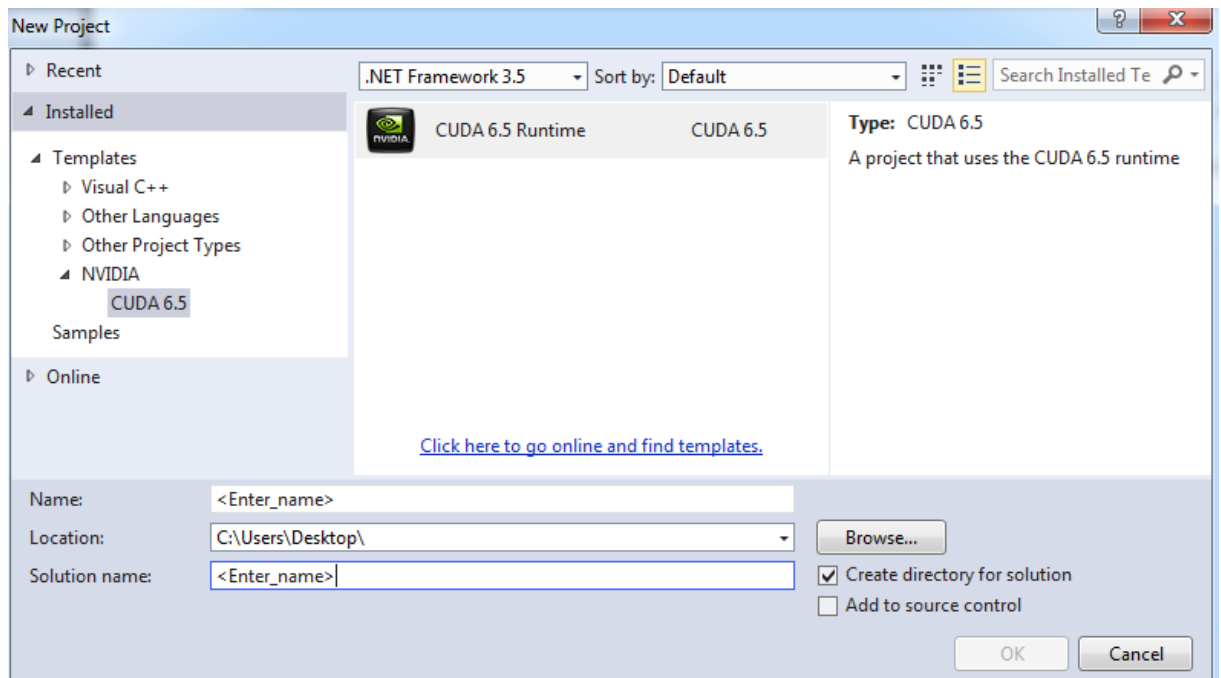
For Windows Vista, Windows 7, Windows 8, Windows Server 2003, and Windows Server 2008:

C:\ProgramData\NVIDIA Corporation\CUDA Samples\v5.5\C\bin\win32\Release

(or ... \win64\Release on 64-bit Windows)

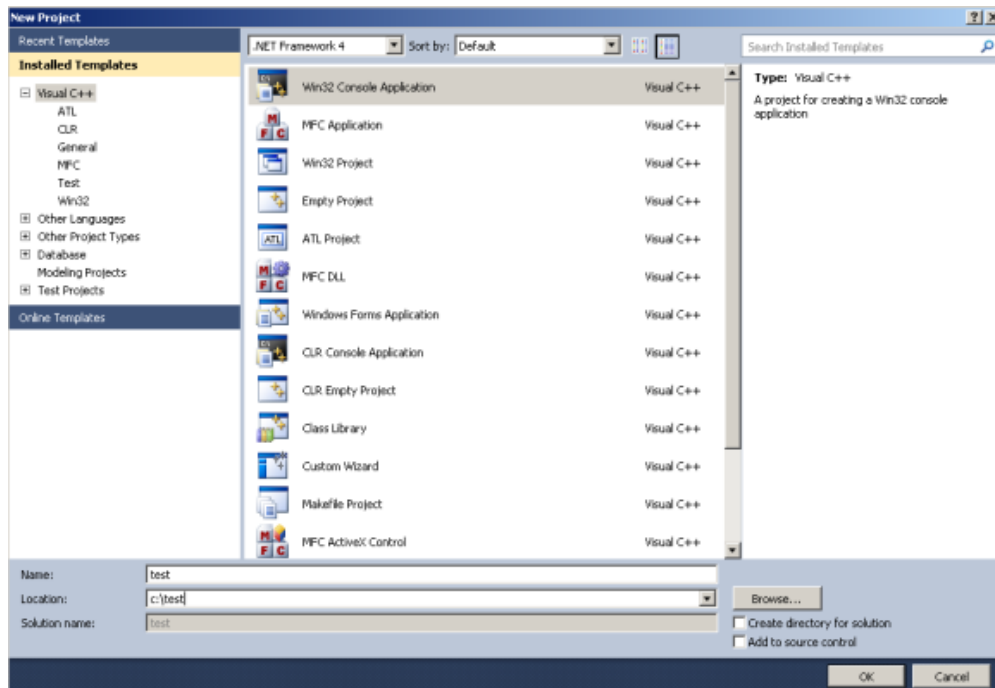
## 2.4. Working with Visual Studio - CUDA 5.0 or newer

Simply create a new CUDA project (under the NVIDIA group)



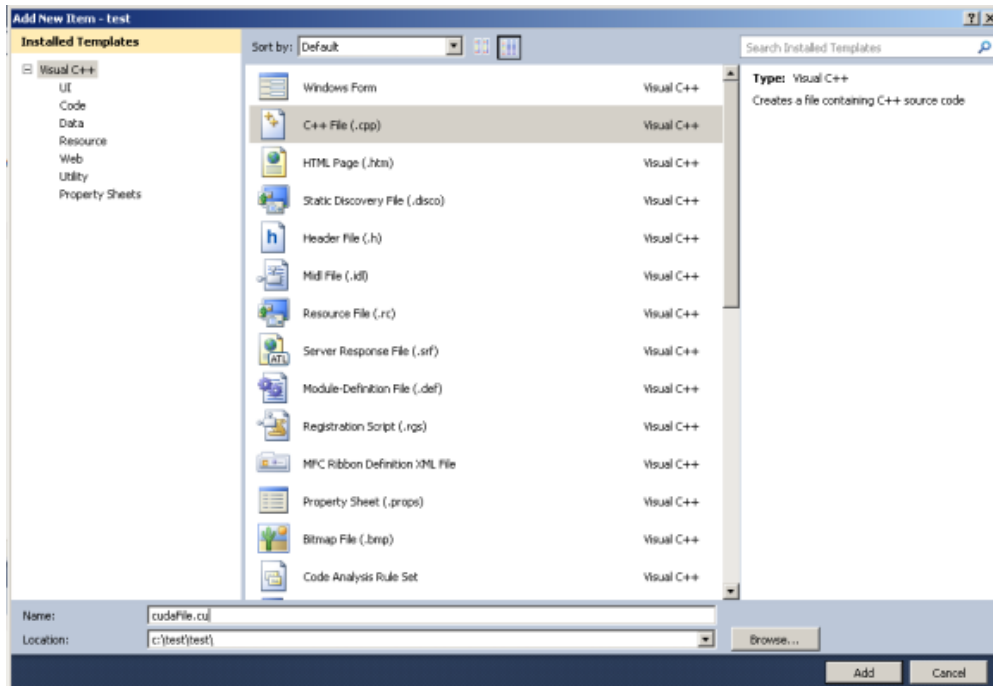
**If not possible or you can't see the NVIDIA group:**

1. Set up a visual studio console project





2. Create a .cu file and add it to your project.
3. Check the build customizations project->build customization



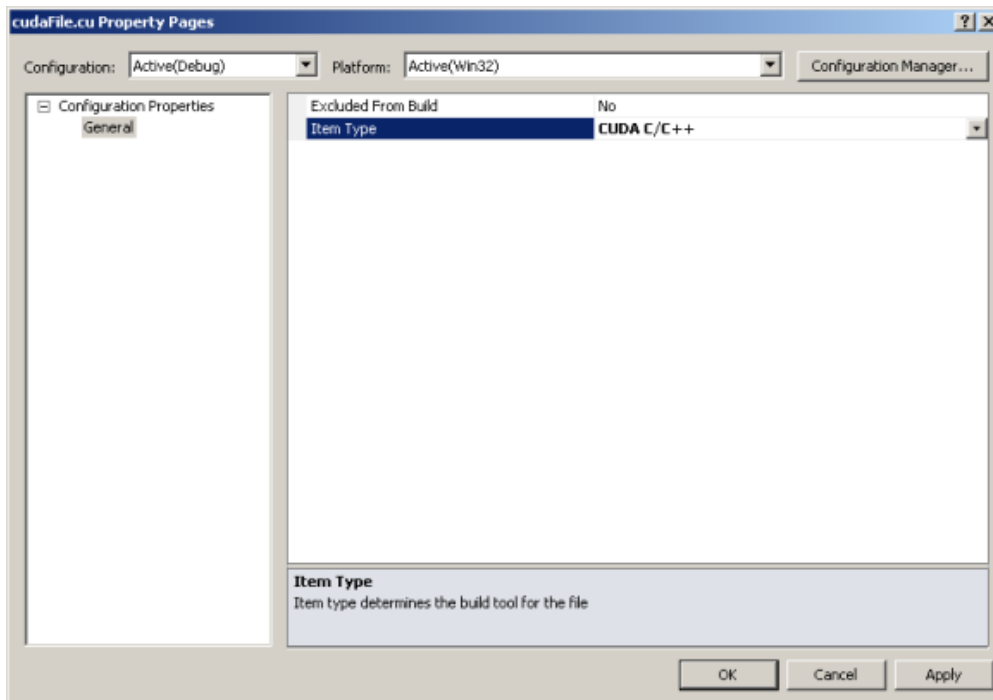
Select cuda5.5 build customization. If the cuda toolkit is properly installed then it should have the cuda build customization option. However if it is not there, please copy the .targets, .props from the location

"C:\Program Files\NVIDIA GPU Computing

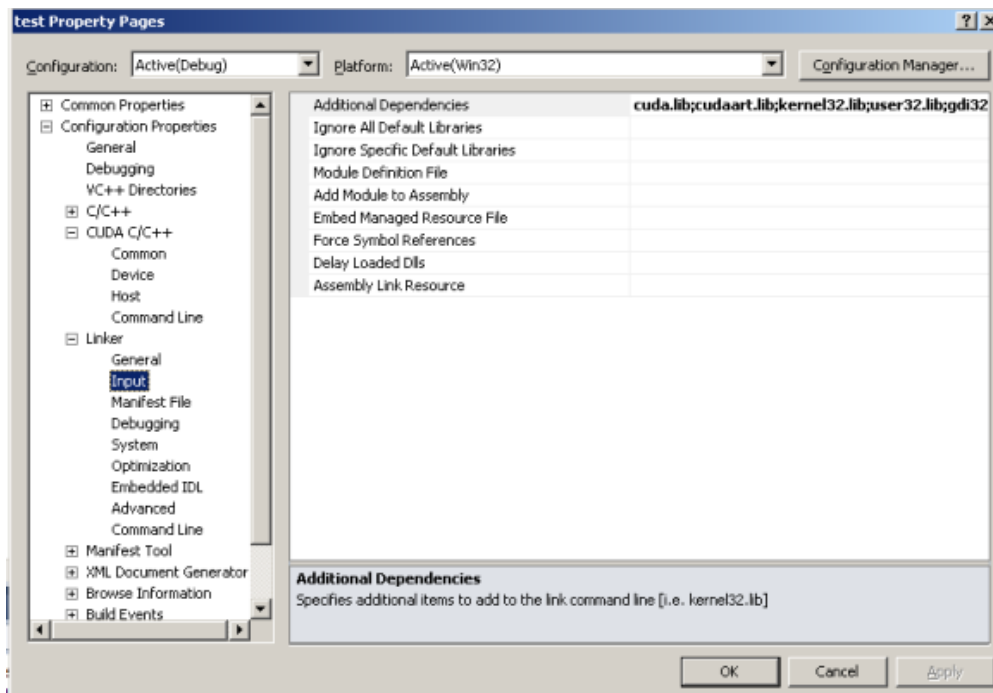
Toolkit\CUDA\v4.2\extras\visual\_studio\_integration\MSBuildExtensions" to

""C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v5.5\BuildCustomizations"

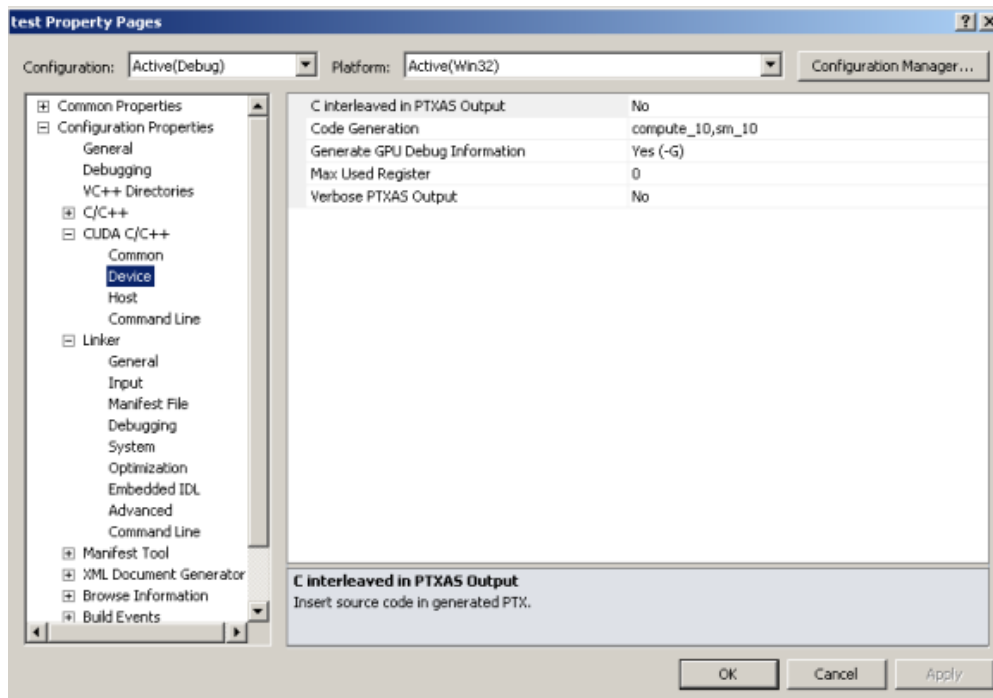
4. Go the property page for the file "cudaFile.cu" and change the item type to Cuda C/C++



5. Add cudart.lib, and cuda.lib in the linker input.



You should be able to compile now your empty project. In case there is any error check the project property->CUDA C/C++->Device and ensure the code generation is compute\_10,sm\_10



### 3. A Simple Example

In this section, we will develop a simple CUDA program which computes the square of each number in a single vector.

Create a new CUDA project, write element by element summation code and run the project:

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

// Kernel that executes on the CUDA device
__global__ void array_summation(float *a, int N)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N) a[idx] = a[idx] + a[idx];
}

// main routine that executes on the host
int main(void)
{
    float *a_h, *a_d; // Pointer to host & device arrays
    const int N = 20; // Number of elements in arrays
    size_t size = N * sizeof(float);
    a_h = (float *)malloc(size); // Allocate array on host
    cudaMalloc((void **) &a_d, size); // Allocate array on device
    // Initialize host array and copy it to CUDA device
    for (int i=0; i<N; i++) a_h[i] = (float)i;
    cudaMemcpy(a_d, a_h, size, cudaMemcpyHostToDevice);
    // Do calculation on device:
    int block_size = 4;
    int n_blocks = N/block_size + (N%block_size == 0 ? 0:1);
    array_summation <<< n_blocks, block_size >>> (a_d, N);
    // Retrieve result from device and store it in host array
    cudaMemcpy(a_h, a_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
    // Print results
    for (int i=0; i<N; i++) printf("%d %f\n", i, a_h[i]);
    // Cleanup
    free(a_h); cudaFree(a_d);
}
```

(If you want to get rid of warnings, you may add “cuda\_runtime.h” and “device\_launch\_parameters.h”)

### **If you can't create a CUDA project:**

1. Start Visual Studio.
2. File-->New Project -->Visual C++ ---> Win32 -->Win32 Console Application
3. Choose a creative project name as, example1
4. Choose Console Application in Application Settings
6. At the moment, you may either write or copy the code
7. Rename example1.cpp--->example1.cu
8. Project Name-->right-click-->Build Customizations-->CUDA X.X
9. Add: "cudart.lib cuda.lib" (for each configuration such as Debug and Release).
10. Run the project

### **Output:**

```
0 0.000000
1 2.000000
2 4.000000
3 6.000000
4 8.000000
5 10.000000
6 12.000000
7 14.000000
8 16.000000
9 18.000000
10 20.000000
11 22.000000
12 24.000000
13 26.000000
14 28.000000
15 30.000000
16 32.000000
17 34.000000
18 36.000000
19 38.000000
```

## 4. Lab. Exercises

In the lab. session, you will implement the following exercises. You will introduce your results as a lab. document by uploading the document to ODTUclass.

In your document:

1. Introduce the problem definition of the exercise.
2. Introduce your algorithm (your implementation strategy).
3. Discuss your results, that is the cons and pros of your solution, by considering the memory, time and environment constraints.

If you cannot complete the exercises in the lab. session, you may upload your document after the lab. Moreover, please upload your codes to the lab. assignment.

### Exercise 1: Vector Product

In the example, we have implemented an element-by-element summation algorithm. In this exercise, you will implement a simple CUDA program that calculates the inner product of two vectors. (You may use a loop on CPU for summation after multiplication)

For example, if  $\bar{x} \in \mathbb{R}^N$  and  $\bar{y} \in \mathbb{R}^N$  are two N-element vectors, such as,  $\bar{x} = [x_1, \dots, x_i, \dots, x_N]$  and  $\bar{y} = [y_1, \dots, y_i, \dots, y_N]$ , the inner product of two-vectors is a scalar number  $z$ ,

$$z = \langle \bar{x}, \bar{y} \rangle = \sum_{i=1}^N x_i y_i$$

### Exercise 2: Matrix Summation

Following the simple example in Section 4, you will implement a simple CUDA program, which computes the summation of the elements of two matrices.

For example, if  $X \in \mathbb{R}^{M \times N}$  and  $Y \in \mathbb{R}^{M \times N}$  are two M by N-element matrices, the element wise summation of the matrices is a matrix  $Z \in \mathbb{R}^{M \times N}$ ,

$$Z_{i,j} = X_{i,j} + Y_{i,j},$$

where  $Z_{i,j}$  is the element in the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the matrix  $Z$ .

### Exercise 3: Matrix - Vector Multiplication

Following the simple example in Section 4, you will implement a simple CUDA program, which computes the matrix - vector multiplication.

For example, if  $X \in \mathbb{R}^{M \times N}$  is a matrix and  $\bar{y} \in \mathbb{R}^N$  is a vector, their multiplication is a vector  $\bar{z} \in \mathbb{R}^M$ , such that,

$$z_i = \sum_{j=1}^N X_{i,j} y_j,$$

where  $\bar{z} = [z_1, \dots, z_i, \dots, z_M]$ .

## 5. Assignment 1: Matrix Multiplication

In the assignment, you will implement a matrix multiplication algorithm with CUDA. **You are required to introduce your assignment with an assignment document.**

In the assignment, you may study on the matrix multiplication implementation provided by CUDA SDK, or on the other implementations that you can find on internet.

There are several matrix multiplication algorithms, strategies and implementations. You are required to implement **your own algorithm and strategy**. You should also indicate the implementations etc. that you used in your assignment in the references part.

**You need to have a look at the “Assignment and Report Guide” document on ODTUclass for how you should submit your code and grading details!**

## 6. References

1. Lecture notes of the course MMI-713, access with ODTUclass.
2. CUDA Documentation <https://docs.nvidia.com/cuda/index.html>