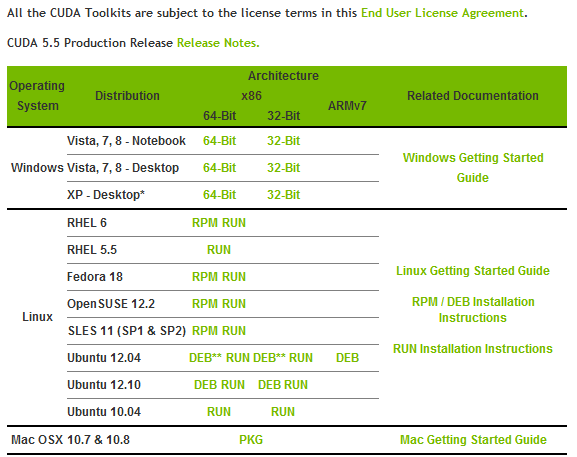
**VS2010 CUDA 5.5 Win7 64位配置以及项目创建配置**

**一.安装CUDA5.5以及配置VS助手**

1、安装之前必须确认自己电脑的GPU支持CUDA。在设备管理器中找到显示适配器（Display adapters)，找到自己电脑的显卡型号，如果包含在<http://www.nvidia.com/object/cuda_gpus.html>的列表中，说明支持CUDA。

安装CUDA之前最好检查一下自己电脑的显卡驱动版本，版本过老的话，需要更新。有时候安装完毕CUDA之后，运行6中的deviceQuery程序时会出错，请优先考虑显卡驱动的问题。

2、下载NVIDIA CUDA Toolkit （[http://www.nvidia.com/content/cuda/cuda-downloads.html](https://developer.nvidia.com/cuda-downloads)），本人下载的是desktop版本win64



CUDA5.0以上已经将ToolKit和SDK等整合在了一起，因此只需下载一个安装文件即可。

3、首先确认自己已经安装Visual Studio 2010 后再安装助手Visual Assist X。这里Visual Assist X 对于使用CUDA 不是必须的，但为了使程序编写更为方便，这里推荐安装。

4、CUDA的默认安装目录为：C:\Program Files\NVIDIA GPU Computing Toolkit\ 这里推荐使用默认的安装路径，不需更改。

5、安装完成后，在系统变量环境里面会发现新添加了两个环境变量

CUDA\_PATH

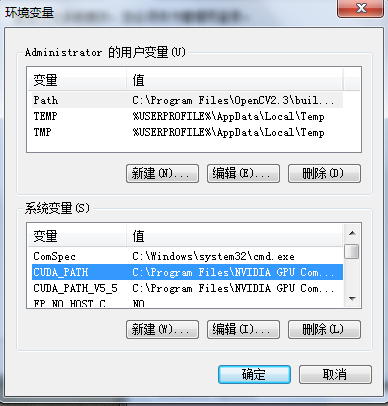
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v5.5

CUDA\_PATH\_V5\_5

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v5.5

且在系统变量的path也会自动添加

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v5.5\bin;C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v5.5\libnvvp;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;



6、此时CUDA已经安装成功。可以在DOS命令下，测试运行显示GPU硬件配置和cuda属性例子程序deviceQuery。路径在如下位置：

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

Dos下的命令提示符

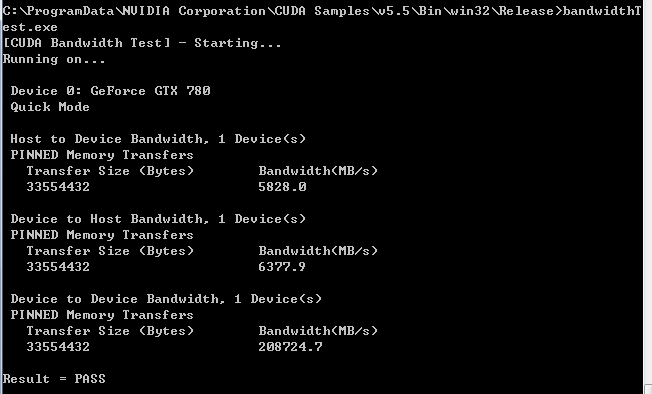
dir查看文件夹下的目录

dir /a:h 查看当前目录的隐藏文件，因为ProgramData是隐藏文件。

cd .. 返回上一级文件夹， cd file 进入file 文件夹



运行bandwidthTest示例程序，结果如下：

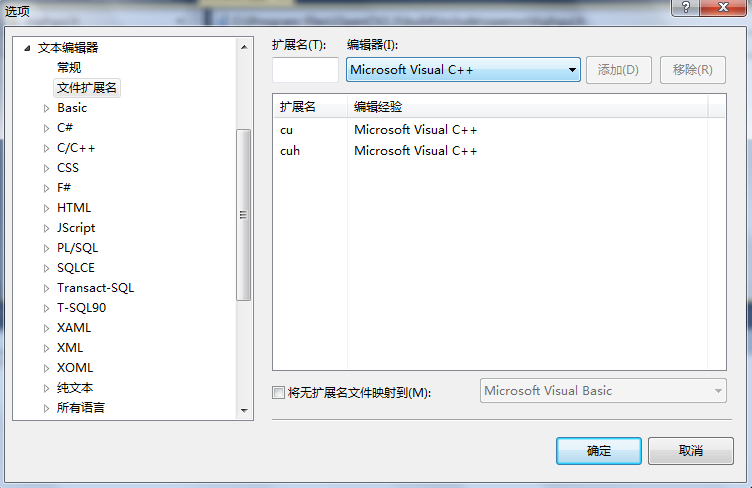


双击桌面上的图标，运行cuda自带的例子。



7、CUDA C/C++关键字及函数高亮显示

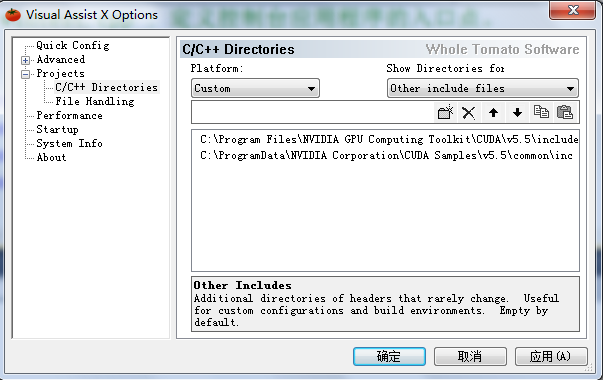
这个设置是让Visual Studio2010 在编辑.cu 文件时，把.cu 文件里的C/C++语法高亮。设置方法： 在Visual Studio 2010 的菜单依次选“ Tools|Options|Text Editor|File Extension（工具|选项|文本编辑器|文件扩展名）”，在该窗口中将“Editor（编辑器）”下拉框选择“Microsoft Visual C++”，在“Extension（扩展名）”文本框中输入cu 点击“Add（添加）”按钮，重复工作把cuh 添加为Visual C++类型，添加完成后点击“OK（确定）”按钮，如下图所示：



为了让CUDA 的关键字，如\_\_device\_\_、dim3 等的文字高亮，需按如下步骤设置：  
将C:\ProgramData\NVIDIA Corporation\CUDA Samples\v5.5\doc\syntax\_highlighting\visual\_studio\_8 目录下的usertype.dat 文件复制到C:\Program Files\Microsoft Visual Studio10.0\Common7\IDE\ 目录下（ 对x64 位Win7 系统为X:\Program Files(X86)\Microsoft Visual Studio 10.0\Common7\IDE\）。重启Visual Studio 2010 后打开.cu 文件，CUDA 的关键字应该变成蓝色了。

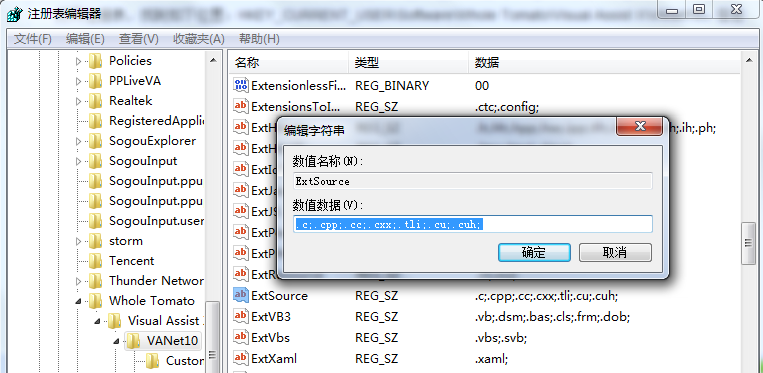
8、CUDA 函数高亮，及CUDA 函数输入代码提示

实现这个功能需要使用Visual Assist X，如果没有安装支持Visual Studio 2010的Visual Assist X，这部分功能无法实现。这里只是为使编写代码更加方便，对CUDA 程序的开发无实质性障碍。  
如果已经安装Visual Assist X，可以通过以下两步实现需要的功能。  
（1）使Visual Assist X 支持CUDA 函数高亮和代码完成  
在Visual Studio 2010 的菜单里依次选择：“ VAssistX|Visual assist X Options|Projects|C/C++Directories”，在该界面的“Platform”下拉框中选择Custom，在“Show Directories for”下拉框中选择Other include files， 然后在下面的输入框里，新建、添加如下路径，如图：  
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v5.5\include  
C:\ProgramData\NVIDIA Corporation\CUDA Samples\v5.5\common\inc



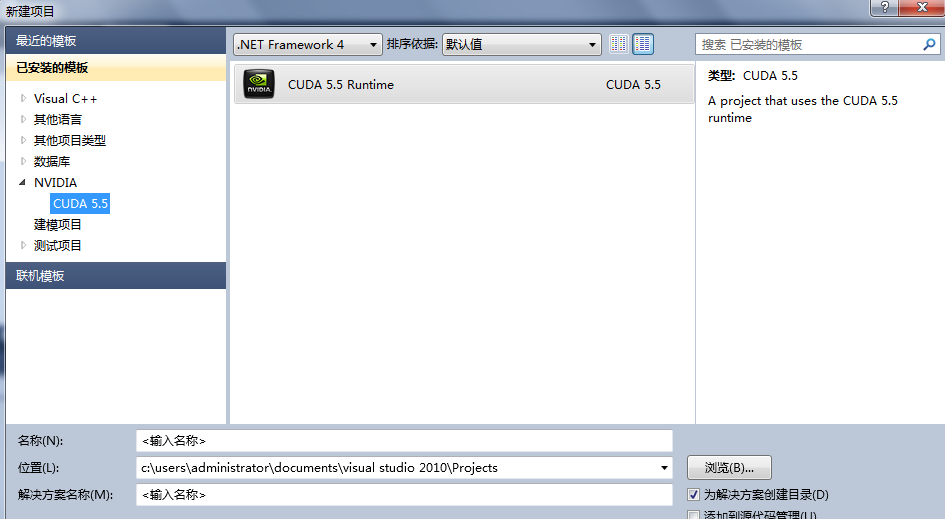
（2）使Visual Assist X 实现.cu 文件高亮和代码完成功能，需要编辑注册表。在修改注册表时，为避免带来不必要的错误请先关闭Visual Studio 2010。使用Win+R组合键打开“运行”窗口，键入入regedit 命令（register edit 的缩写）打开注册表，找到如下位置：HKEY\_CURRENT\_USER\Software\Whole Tomato\Visual Assist X\VANet10。在右边找到ExtSource 项目，鼠标右键选修改，在原有文字后添加如下文字：.cu;.cuh; 确定后关闭注册表。重新打开Visual Studio 2010，Visual Assist X 便开始支持.cu 及.cuh 文件的语法高亮及代码完成。此时.cu 文件的CUDA 函数是高亮的，使用函数名符号就会自动提示函数全称，参数类型等信息。

修改注册表时，请把vs关闭，否则修改不成功。

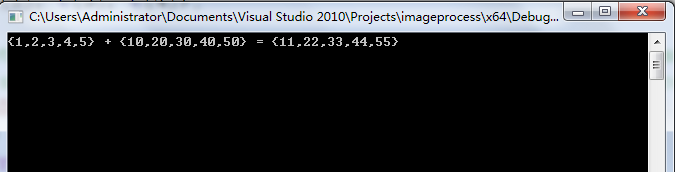


**二.创建cu文件测试示例**

在Visual Studio 2010 菜单选择“file|new|project（文件|新建|工程）”，在打开的新建项目窗口的“已安装的模板”一栏中选择“NVIDIA|CUDA”，类型选择为“CUDA 5.5 Runtime”，见下图。



在“名称”中输入工程名后，点击确定。cuda自动提供的kernel.cu 并行加法示例进行编译运行。结果如下：



在此基础上可以方便的进行cuda的代码测试。

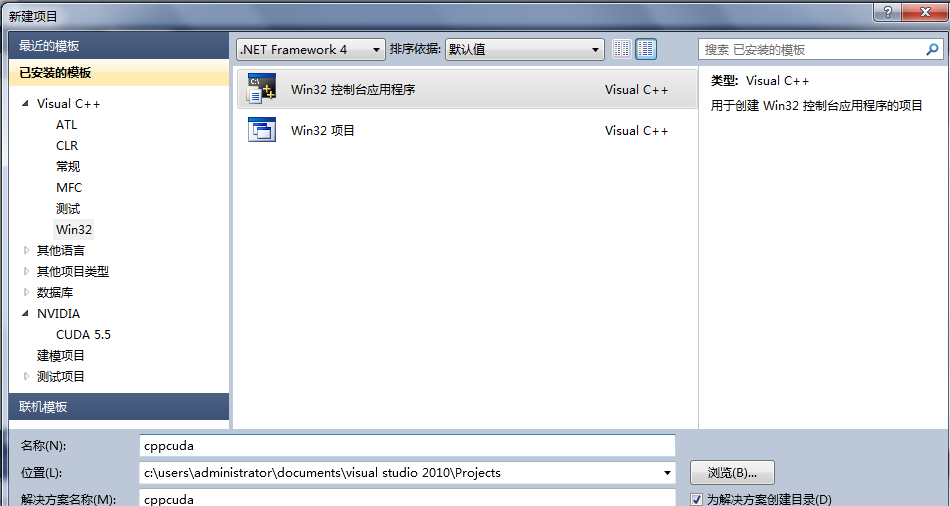
**三.在已有的C++项目中添加cu文件**

我们往往需要在已建成的vc项目中，加入gpu的并行算法，以提高算法的效率。在此介绍如何在已有的项目中添加cu文件和配置过程。在cuda sample中也提供了具体的参考示例，路径如下：

C:\ProgramData\NVIDIA Corporation\CUDA Samples\v5.5\0\_Simple\cppIntegration

运行cppIntegration\_vs2010.sln，查看其配置。

1) 新建一个Win32控制台项目cppcuda；



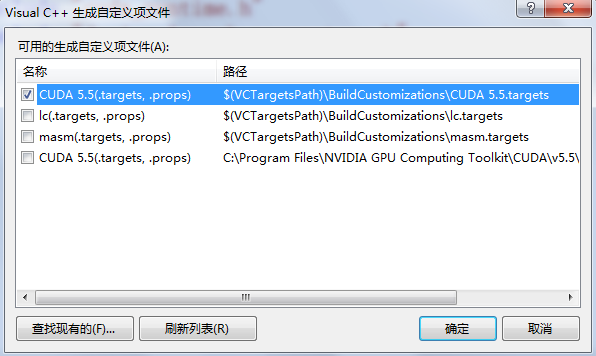
2) 然后在项目中添加空白cu文件：在源文件处点击右键，添加》新建项

选择的文件类型是CPP文件，在写名称的时候可以直接写为.cu文件，也可以写成.cpp文件之后再进行重命名。此处写为add\_kernel.cu.

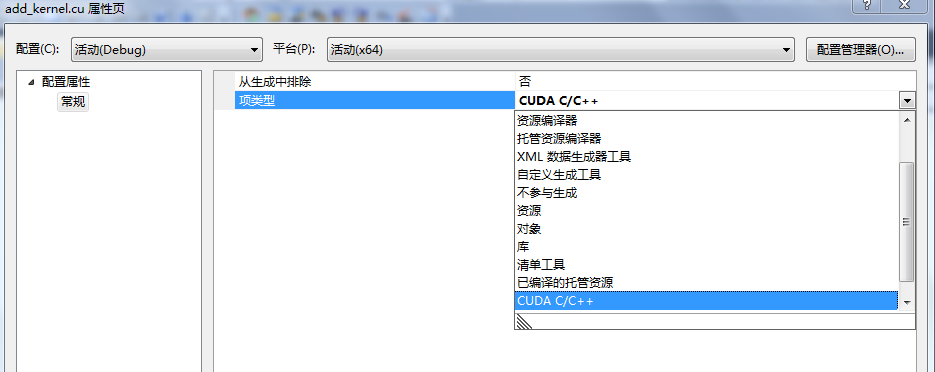
3)  在项目名称右键选择 生成自定义（B）…：



选择CUDA 5.5.(targets,.props)。

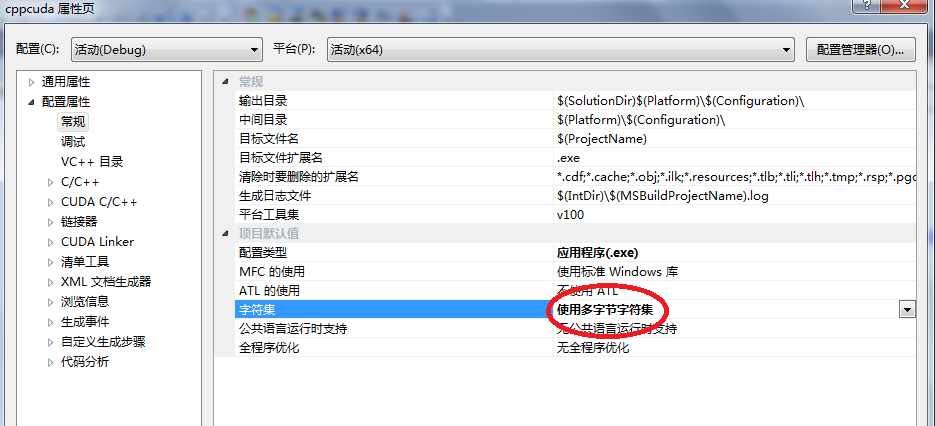


4) 在add\_kernel.cu文件处点击右键，选择属性》配置属性》项类型 下拉菜单中选择 CUDA C/C++



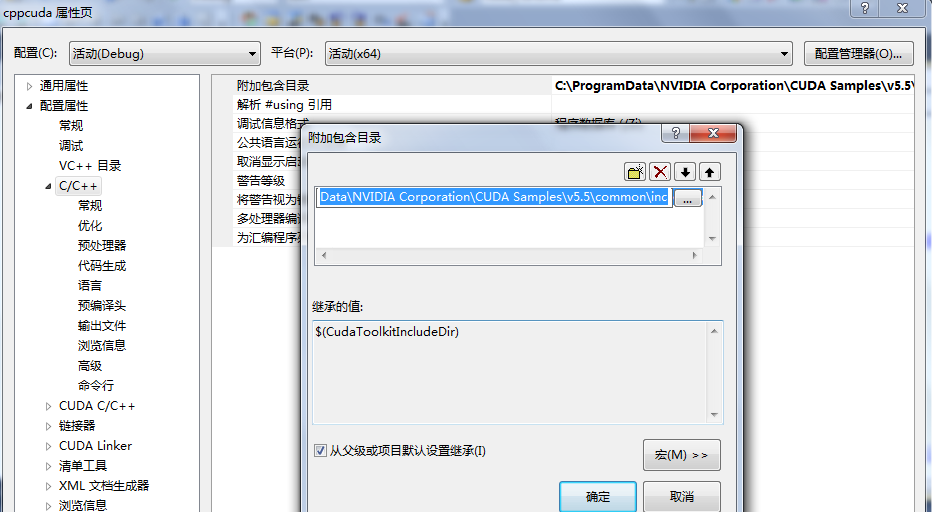
5)在项目》cppcuda属性页中，设置如下红色圆圈内的参数

更改 配置属性》常规》字符集：

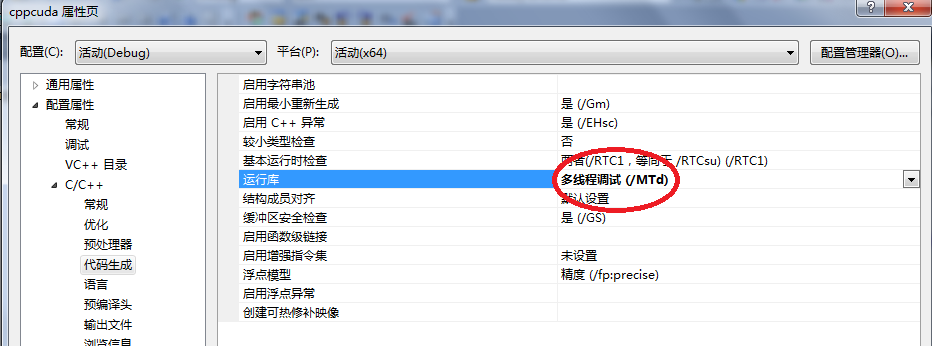


在 C/C++》常规》附加包含目录 中添加绝对路径：

C:\ProgramData\NVIDIA Corporation\CUDA Samples\v5.5\common\inc

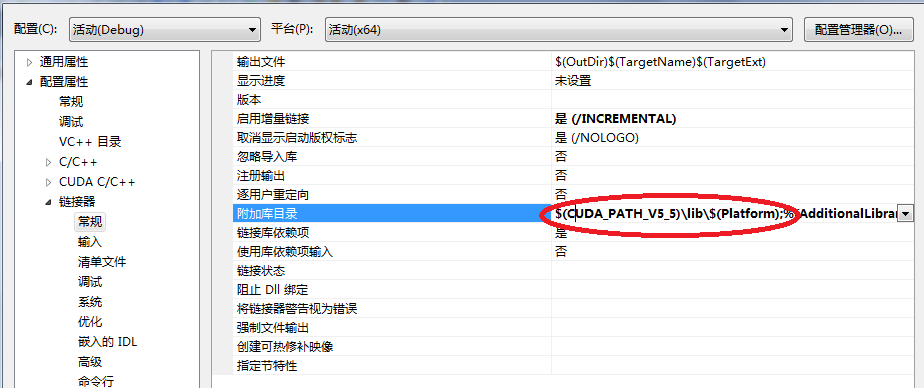


C/C++》代码生成》运行库 改为 多线程调试



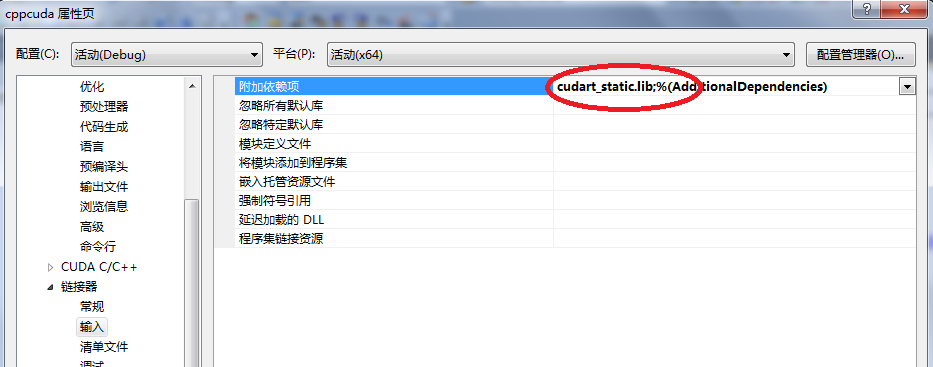
链接器》常规》附加库目录中添加

$(CUDA\_PATH\_V5\_5)\lib\$(Platform)



链接器》输入》附加依赖项 中输入

cudart\_static.lib



6)以上配置完成之后，可以在空白的addcuda.cu开头添加如下两行代码

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

7) 在cpp 与cu文件建立连接的函数定义必须用extern "C"形式开头，如

extern "C" double addnum(int \*c, const int \*a, const int \*b, unsigned int size);

addnum这个函数具体在cu文件实现并行加法的功能，在cpp文件中调用。

8) 下面给出cppcuda具体的代码

**如cppcuda.cpp：**

#include "stdafx.h"

extern "C" double addnum(int \*c, const int \*a, const int \*b, unsigned int size);

int \_tmain(int argc, \_TCHAR\* argv[])

{

const int arraySize = 5;

const int a[arraySize] = { 1, 2, 3, 4, 5 };

const int b[arraySize] = { 10, 20, 30, 40, 50 };

int c[arraySize] = { 0 };

addnum(c, a, b, arraySize);

return 0;

}

**如add\_kernel.cu：**

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <stdio.h>

cudaError\_t addWithCuda(int \*c, const int \*a, const int \*b, unsigned int size);

\_\_global\_\_ void addKernel(int \*c, const int \*a, const int \*b)

{

int i = threadIdx.x;

c[i] = a[i] + b[i];

}

extern "C" double addnum(int \*c, const int \*a, const int \*b, unsigned int size)

{

cudaError\_t cudaStatus = addWithCuda(c, a, b, size);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "addWithCuda failed!");

return 1;

}

}

cudaError\_t addWithCuda(int \*c, const int \*a, const int \*b, unsigned int size)

{

int \*dev\_a = 0;

int \*dev\_b = 0;

int \*dev\_c = 0;

cudaError\_t cudaStatus;

// Choose which GPU to run on, change this on a multi-GPU system.

cudaStatus = cudaSetDevice(0);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaSetDevice failed! Do you have a CUDA-capable GPU installed?");

goto Error;

}

// Allocate GPU buffers for three vectors (two input, one output) .

cudaStatus = cudaMalloc((void\*\*)&dev\_c, size \* sizeof(int));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

cudaStatus = cudaMalloc((void\*\*)&dev\_a, size \* sizeof(int));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

cudaStatus = cudaMalloc((void\*\*)&dev\_b, size \* sizeof(int));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

// Copy input vectors from host memory to GPU buffers.

cudaStatus = cudaMemcpy(dev\_a, a, size \* sizeof(int), cudaMemcpyHostToDevice);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy failed!");

goto Error;

}

cudaStatus = cudaMemcpy(dev\_b, b, size \* sizeof(int), cudaMemcpyHostToDevice);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy failed!");

goto Error;

}

// Launch a kernel on the GPU with one thread for each element.

addKernel<<<1, size>>>(dev\_c, dev\_a, dev\_b);

// Check for any errors launching the kernel

cudaStatus = cudaGetLastError();

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "addKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));

goto Error;

}

// cudaDeviceSynchronize waits for the kernel to finish, and returns

// any errors encountered during the launch.

cudaStatus = cudaDeviceSynchronize();

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaDeviceSynchronize returned error code %d after launching addKernel!\n", cudaStatus);

goto Error;

}

// Copy output vector from GPU buffer to host memory.

cudaStatus = cudaMemcpy(c, dev\_c, size \* sizeof(int), cudaMemcpyDeviceToHost);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy failed!");

goto Error;

}

Error:

cudaFree(dev\_c);

cudaFree(dev\_a);

cudaFree(dev\_b);

return cudaStatus;

}