



OpenCL初探:环境搭建

2015年01月09日 标签：OpenCL

前言

笔者的机器是支持Nvidia的，而所参考书籍及资料大多却是AMD。作为一枚新手，初次搭建cuda环境还是花了点时间的，故此记录下折腾的过程，方便自己以及其他OpenCL学习者参考吧。

笔者硬件信息及开发工具

- 机器: Lenovo IdeaPad Y470
- 系统：Windows 7 旗舰版 64位 SP1
- 处理器: 英特尔 第二代酷睿 i5-2430M @ 2.40GHz 双核
- 内存: 4 GB (DDR3 1333MHz)
- 主显卡: Nvidia GeForce GT 550M (2 GB / 联想)
- 开发工具: Visual Studio 2010

安装 CUDA

笔者选择的是最新版**CUDA6.5** 64-bit,下载地址:

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

下载完毕之后，点击安装，有两点需得提醒下：

- 记得选择自定义，为了避免漏装了些组件而导致后面出错，笔者全部安装了

你有1条系统消息



- 下图中的路径可以更改，但请对这些路径有所印象，后续还得从中寻些所需的头文件和库



btw:上图是从网上借的，路径里显示的是v5.5代表的是5.5版本，

因为我们安装的是6.5故应为v6.5,其余一样。

提醒：记得更新显卡驱动为最新。

你有1条系统消息

安装完成后,在CUDA Samples中找到deviceQuery.exe,默认路径
C:\Program Files\NVIDIA Corporation\CUDA Samples\v6.5\bin
\win64\Release,将其拖拽cmd中并回车,显示如下则安装成功：

```
CUDA Device Query (Runtime API) version (CUDART static linking)
Detected 1 CUDA Capable device(s)
Device 0: "GeForce GT 550M"
  CUDA Driver Version / Runtime Version      6.5 / 6.5
  CUDA Capability Major/Minor version number: 2.1
  Total amount of global memory: 2048 MBytes (2147483648 bytes)
  ( 2) Multiprocessors, ( 48) CUDA Cores/MP: 96 CUDA Cores
  GPU Clock rate: 1480 Mhz (1.48 GHz)
  Memory Clock rate: 900 Mhz
  Memory Bus Width: 128-bit
  L2 Cache Size: 131072 bytes
  Maximum Texture Dimension Size (x,y,z)  1D=(65536), 2D=(65536, 65535), 3D=(2048, 2048, 2048)
  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
  Total amount of constant memory: 65536 bytes
  Total amount of shared memory per block: 49152 bytes
  Total number of registers available per block: 32768
  Warp size: 32
  Maximum number of threads per multiprocessor: 1536
  Maximum number of threads per block: 1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size (x,y,z): (65535, 65535, 65535)
  Maximum memory pitch: 2147483647 bytes
  Texture alignment: 512 bytes
  Concurrent copy and kernel execution: Yes with 1 copy engine(s)
  Run time limit on kernels: Yes
  Integrated GPU sharing Host Memory: No
  Support host page-locked memory mapping: Yes
  Alignment requirement for Surfaces: Yes
  Device has ECC support: Disabled
  CUDA Device Driver Mode (TCC or WDDM): WDDM (Windows Display Driver Model)
  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 = 6.5, CUDA Runtime Version = 6.5, NumDevs = 1, Device0 = GeForce GT 550M
Result = PASS
```

需提醒:笔者的机子是集成显卡+独立显卡的,第一次运行此程序的时候报错"no cuda-capable device is detected",打开双显卡切换按钮即可。

第一个OpenCL测试案例

- 新建文件夹OpenCL_inc,然后将CUDA Toolkit安装路径下的include文件夹中的头文件及CL文件夹拷贝进来(默认路径

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA
\v6.5\include)

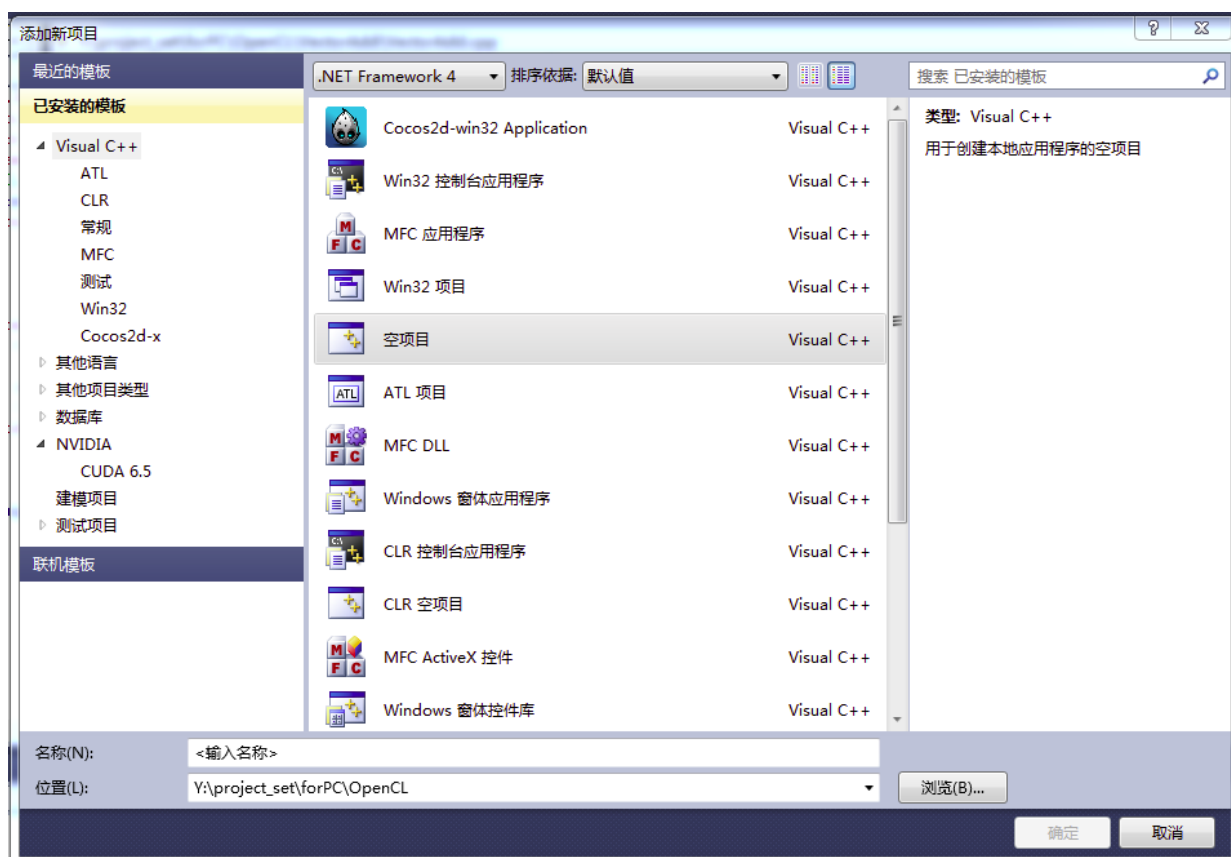
你有1条系统消息

- 新建文件夹**OpenCL_lib** , 然后将**CUDA Toolkit**安装路径下的**lib\Win32**(记住不是lib\x64)文件夹中的**OpenCL.lib**拷贝进来 (默认路径C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v6.5\lib\Win32) ; 接着将nvidia显卡驱动中的动态链接库**OpenCL.dll**和**OpenCL64.dll**拷贝进来 (默认路径C:\Program Files\NVIDIA Corporation\OpenCL)

所有文件准备齐全 , 以备后用 , 如下图所示 :

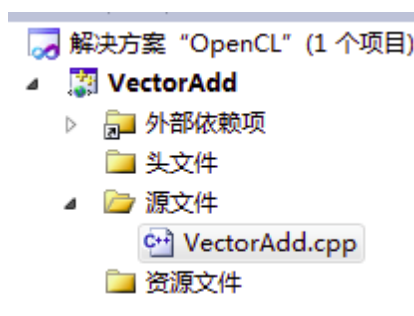
	OpenCL_inc	2015/1/9 9:51	文件夹
	OpenCL_lib	2015/1/9 10:23	文件夹

- 打开**vs**,创空项目,然后将上述库文件和头文件文件夹置于根目录下

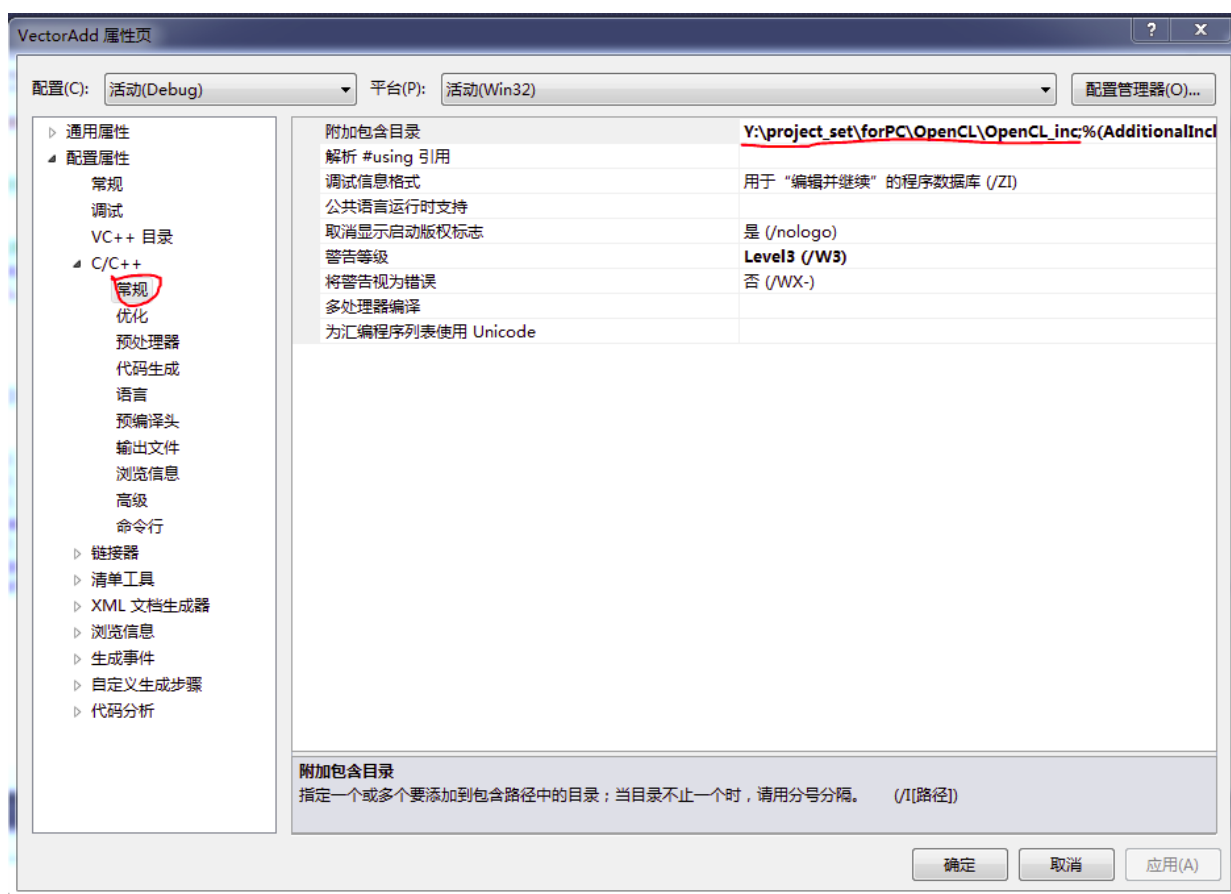


- 添加源文件 VectorAdd.cpp，复制文章末尾处测试代码

你有1条系统消息

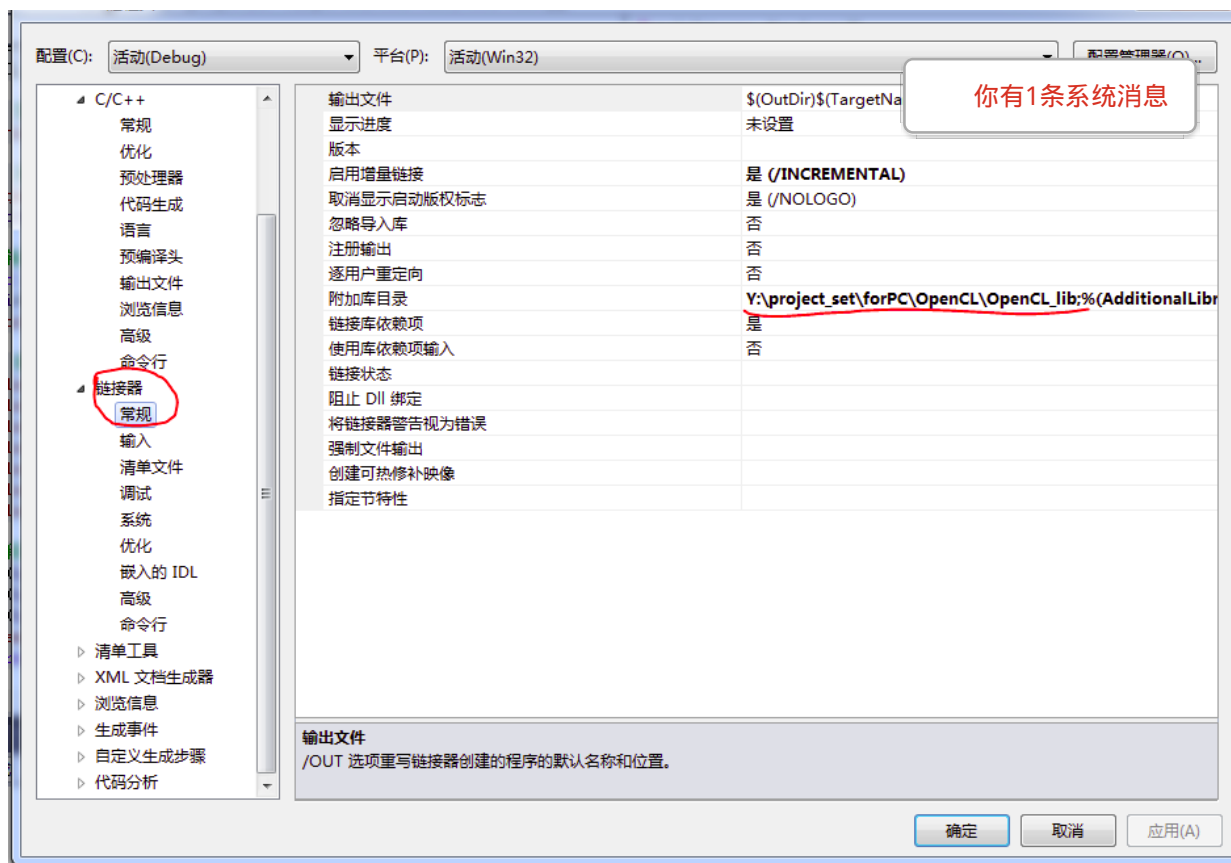


- 添加头文件 光标置于项目上，右键然后选择属性。在如下对话框中选择C/C++->常规，然后在右侧的编辑框内输入需添加的头文件，笔者头文件路径是Y:\projectset\forPC\OpenCL\OpenCLinc\OpenCLinc;

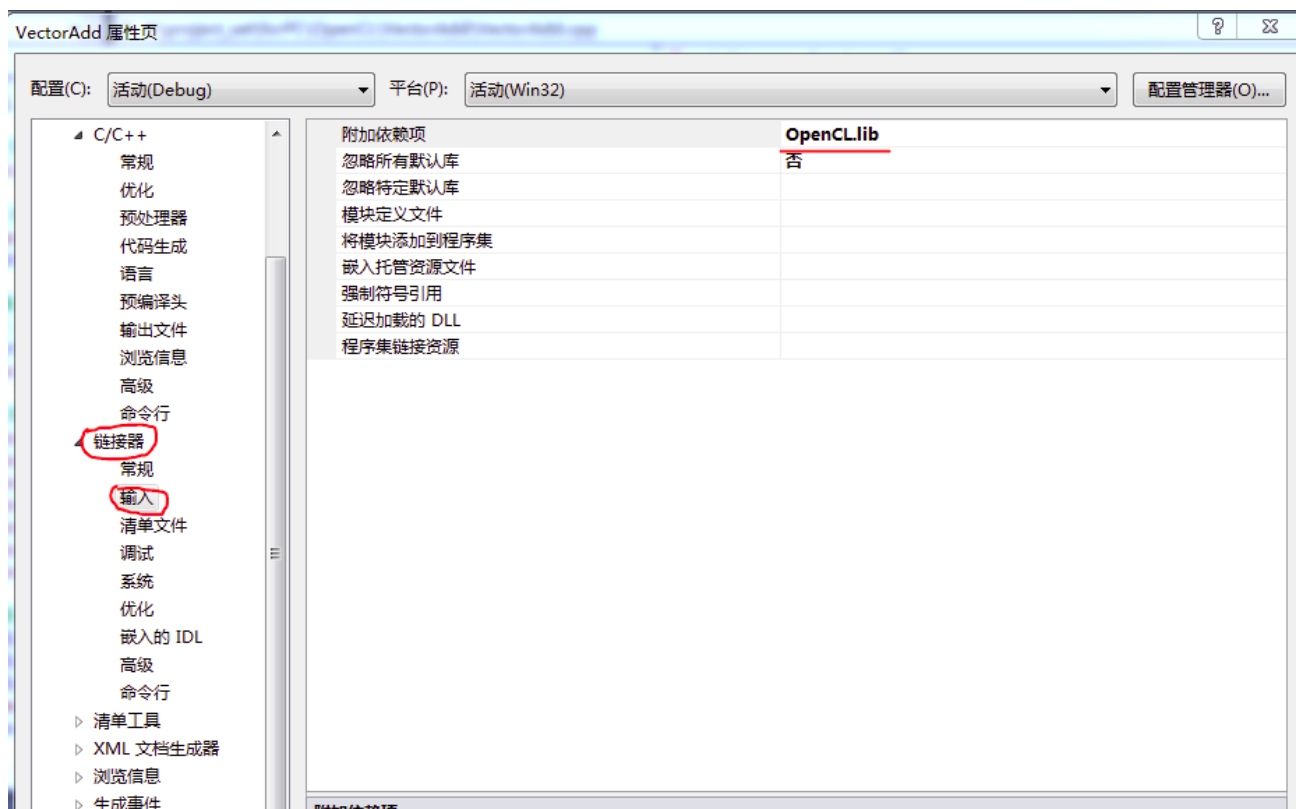


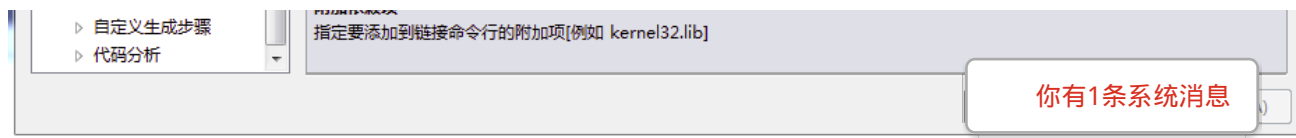
- 添加库文件 依旧在上述属性对话框中操作，首先打开链接器，然后选择常规，添加库文件所在路径，笔者库文件路径是Y:\projectset\forPC\OpenCL\OpenCLlib;





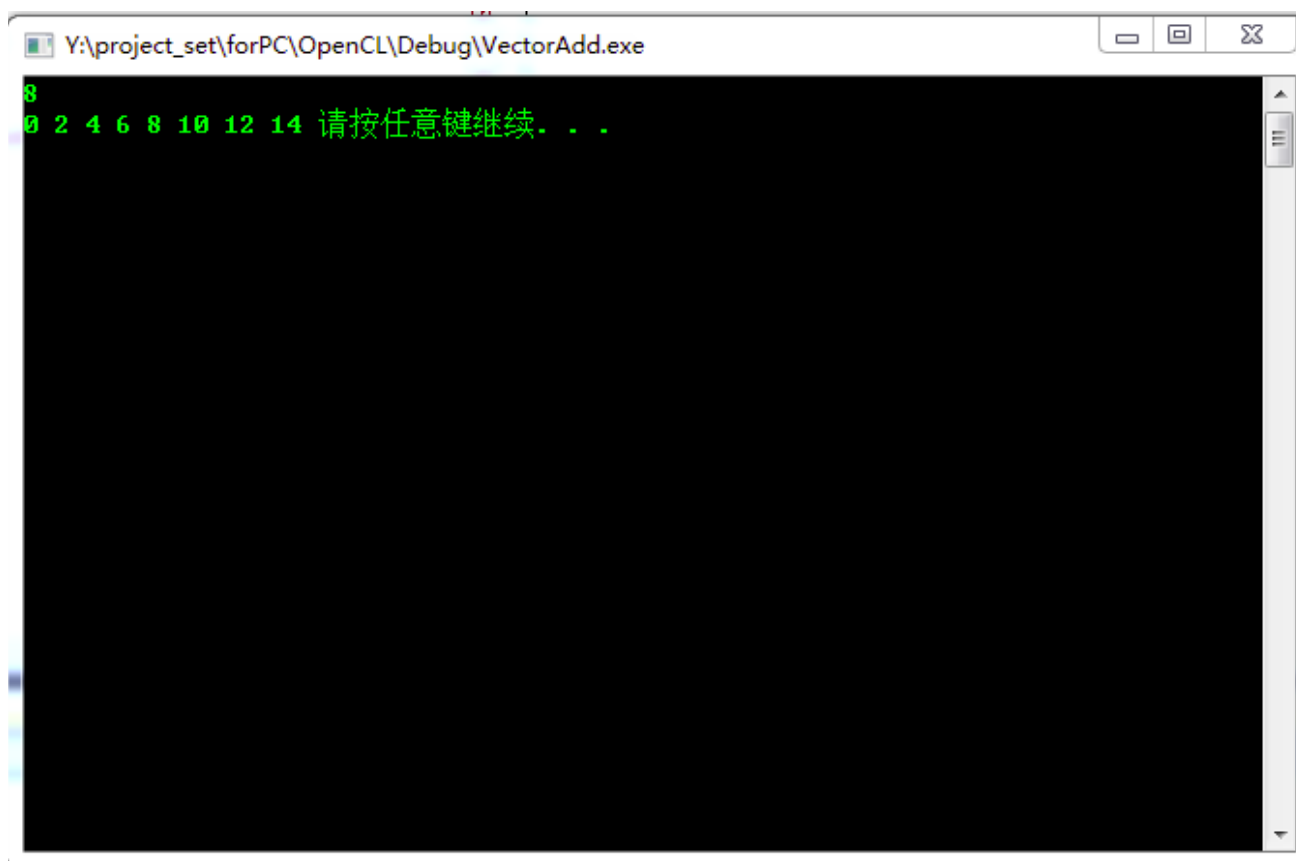
然后，打开链接器，然后选择输入，在附加依赖项中输入
OpenCL.lib





运行项目

运行项目，显示如下,恭喜配置成功：



测试代码

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <CL/opencl.h>
// OpenCL kernel. Each work item takes care of one element of c
const char *kernelSource =
```

你有1条系统消息

```
"__kernel void vecAdd( __global float *a,      \n"          __global float *b,      \n"          __global float *c,      \n"          const unsigned int n)      \n" {\n"    //Get our global thread ID      \n"    int id = get_global_id(0);      \n"    \n"    //Make sure we do not go out of bounds      \n"    if (id < n)      \n"        c[id] = a[id] + b[id];      \n" }\n"\n";\n\nint main( int argc, char* argv[] )\n{\n    // 向量长度\n    int n = 8;\n    // 输入向量\n    int *h_a;\n    int *h_b;\n    // 输出向量\n    int *h_c;\n    // 设备输入缓冲区\n    cl_mem d_a;\n    cl_mem d_b;\n    // 设备输出缓冲区\n    cl_mem d_c;\n    cl_platform_id cpPlatform;    // OpenCL 平台\n    cl_device_id device_id;    // device ID
```


你有1条系统消息

```
cl_context context;          // context
cl_command_queue queue;      // command queue
cl_program program;          // program
cl_kernel kernel;            // kernel
// ( 每个向量的字节数 )
size_t bytes = n*sizeof(int);
// ( 为每个向量分配内存 )
h_a = (int*)malloc(bytes);
h_b = (int*)malloc(bytes);
h_c = (int*)malloc(bytes);
// ( 初始化向量 )
int i;
for( i = 0; i < n; i++ )
{
    h_a[i] = i;
    h_b[i] = i;
}
size_t globalSize, localSize;
cl_int err;
// ( 每个工作组的工作节点数目 )
localSize = 2;
// ( 所有的工作节点 )
globalSize = (size_t)ceil(n/(float)localSize)*localSize;
printf("%d\n",globalSize);
// ( 获得平台ID )
err = clGetPlatformIDs(1, &cpPlatform, NULL);
// ( 获得设备ID , 与平台有关 )
err = clGetDeviceIDs(cpPlatform, CL_DEVICE_TYPE_CPU, 1,
&device_id, NULL);
```

你有1条系统消息

```
// ( 根据设备ID , 得到上下文 )
context = clCreateContext(0, 1, &device_id, NULL, NU
// ( 根据上下文 , 在设备上创建命令队列 )
queue = clCreateCommandQueue(context, device_id, 0, &err);
// ( 根据OpenCL源程序创建计算程序 )
program = clCreateProgramWithSource(context, 1,
    (const char **) & kernelSource, NULL, &err);
// ( 创建可执行程序 )
clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
// ( 在上面创建的程序中创建内核程序 )
kernel = clCreateKernel(program, "vecAdd", &err);
// ( 分配设备缓冲 )
d_a = clCreateBuffer(context, CL_MEM_READ_ONLY, bytes, NULL,
NULL);
d_b = clCreateBuffer(context, CL_MEM_READ_ONLY, bytes, NULL,
NULL);
d_c = clCreateBuffer(context, CL_MEM_WRITE_ONLY, bytes, NULL,
NULL);
// ( 将向量信息写入设备缓冲 )
err = clEnqueueWriteBuffer(queue, d_a, CL_TRUE, 0,
    bytes, h_a, 0, NULL, NULL);
err = clEnqueueWriteBuffer(queue, d_b, CL_TRUE, 0,
    bytes, h_b, 0, NULL, NULL);
// ( 设置计算内核的参数 )
err = clSetKernelArg(kernel, 0, sizeof(cl_mem), &d_a);
err = clSetKernelArg(kernel, 1, sizeof(cl_mem), &d_b);
err = clSetKernelArg(kernel, 2, sizeof(cl_mem), &d_c);
err = clSetKernelArg(kernel, 3, sizeof(int), &n);
// ( 在数据集的范围内执行内核 ) Execute the kernel over the
```

你有1条系统消息

```
entire range of the data set

err = clEnqueueNDRangeKernel(queue, kernel, 1, NU
&globalSize, &localSize,
    0, NULL, NULL);

// ( 在读出结果之前 , 等待命令队列执行完毕 ) Wait for the
command queue to get serviced before reading back results
clFinish(queue);

// ( 从设备缓冲区读出结果 ) Read the results from the device
clEnqueueReadBuffer(queue, d_c, CL_TRUE, 0,
    bytes, h_c, 0, NULL, NULL );

// ( 输出读出的结果 )
float sum = 0;
for(i=0; i<n; i++)
    printf("%d ",h_c[i]);

// ( 释放资源 )
clReleaseMemObject(d_a);
clReleaseMemObject(d_b);
clReleaseMemObject(d_c);
clReleaseProgram(program);
clReleaseKernel(kernel);
clReleaseCommandQueue(queue);
clReleaseContext(context);

// ( 释放内存 )
free(h_a);
free(h_b);
free(h_c);
system("pause");
return 0;
}
```

以上代码实现向量相加，来源于Let it be!。原文中代码有错，笔者已修改。

你有1条系统消息



zhaxin

分享此博文

一柄键盘，一把吉他，添一笔墨色 ...

📍 厦门

喜欢

0条评论

最新 最早 最热

还没有评论，沙发等你来抢

微笑的夏草冬虫

帐号管理



说点什么吧...

☒ 分享到:

发布

Gameeer正在使用多说

前一篇

OpenCL初探:向量相加 (C)

你有1条系统消息

后一篇

读《OpenCL异构计算 (第二版)》