

# CUDA 加速实现图像边缘检测

## 一、要求

Sobel 常用于图像的边缘检测，计算公式如下：

$$Gx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \quad Gy = \begin{bmatrix} -1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

$$G = \sqrt{Gx^2 + Gy^2}$$

其中  $A$  是二维图像， $G$  为检测到的梯度强度。

请用 GPU 实现，并与使用 OpenMP 的方法比较计算时间。自选图像，要求图像像素数大于 2,000,000。

## 二、调试与结果

### 2.1 配置过程

在 Visual Studio 2019 集成开发环境下，建立 CUDA 工程，并结合 OpenCV 库进行实验，所选图像的像素数为  $3943 \times 2628 = 10,362,204$ ，电脑的 GPU 型号为 NVIDIA GeForce GTX 1650 Max-Q Design。

为了在 CUDA 工程中使用 OpenMP 进行对比，需要配置项目属性。点击属性中的 CUDA C/C++，在 Host 里选择 Additional Compiler Options 进行编辑，在编辑框中添加 /openmp，点击确定，如图 1 所示。

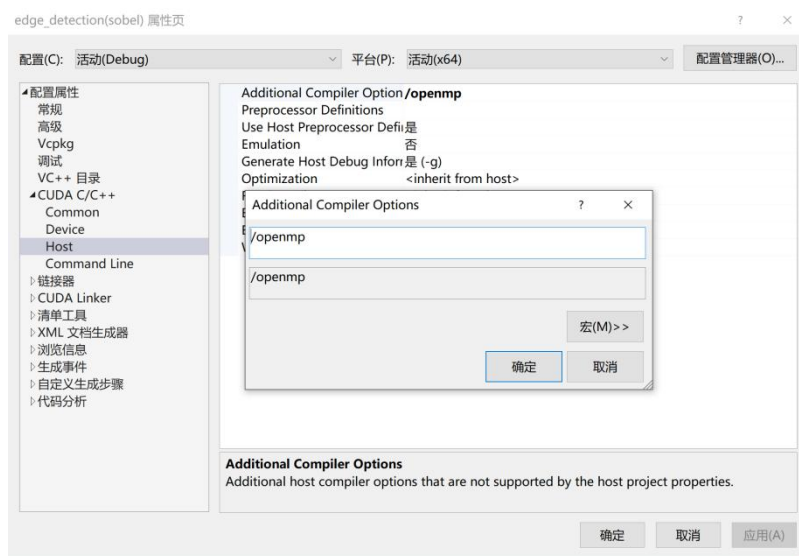


图 1 配置 OpenMP

### 2.2 CUDA 编程处理图像的流程

- (1) 使用 `cudaMalloc()`函数创建 GPU 内存;
- (2) 使用 `cudaMemcpy()`函数和 `cudaMemcpyHostToDevice` 参数将图像数据从 CPU 内存中拷贝到 GPU 的内存中;
- (3) 根据所使用的 GPU 的相关信息定义 `threadsPerBlock` 和 `blocksPerGrid`;
- (4) 执行 `kernel` 核函数;
- (5) 使用 `cudaMemcpy()`函数和 `cudaMemcpyDeviceToHost` 参数将图像数据从 GPU 内存中拷贝到 CPU 的内存中;
- (6) 使用 `cudaFree()`函数释放 GPU 内存。

### 2.3 注意事项

- (1) `threadsPerBlock` 不能超过一个线程块中包含的最大线程数目, 如这里定义 `threadsPerBlock(32, 32)`, 总线程数为 1024, 刚好为本 GPU 允许的最大线程数目。
- (2) 为了在 `kernel` 核函数中使用 `sqrt()`函数, 其中的形参至少是 `float` 型, 不能为 `int` 类型, 这样才能调用 CUDA 的 `math` 库;
- (3) 如果使用 CPU 计时方式, 一定要加同步函数 `cudaDeviceSynchronize()`;
- (4) 正常情况下, 前几次执行核函数的时间会慢一些。这是因为 GPU 在首次计算时需要 `warm up`, 所以前几次核函数的执行时间是不精确的, 可以采用循环 100 次然后求平均的方法获得实际执行时间;
- (5) 应当在 `Debug` 模式下运行代码, 因为 `Release` 模式下编译器会对代码进行优化, 部分语句并行执行以提升运行速度。

### 2.4 运行结果

- (1) 使用 CUDA 进行边缘检测结果如图 2 所示, 可以看到, 原始图片的边缘信息已经被较好地检测出来。
- (2) 进行三次实验, 对比 CUDA 和 OpenMP 带来的加速效果, 结果如表 1 所示。计算可知, 本例中使用 CUDA 带来的加速比 OpenMP 快约 13 倍。



图 1 原始图像



图 2 CUDA 边缘检测结果

表 1 CUDA 与 OpenMP 加速效果对比

	无加速	CUDA 加速	OpenMP 加速
第一次 (s)	0.201	0.00528	0.063
第二次 (s)	0.179	0.00536	0.078
第三次 (s)	0.193	0.00576	0.079

```

Microsoft Visual Studio 调试控制台
the information for the device : 0
name:NVIDIA GeForce GTX 1650 with Max-Q Design
the memory information for the device : 0
total global memory:4294639616
total constant memory:65536
threads in warps:32
max threads per block:1024
max threads dims:1024 1024 64
max grid dims:2147483647 65535 65535

CPU run time = 0.201seconds
OpenMP run time = 0.063seconds
CUDA run time = 0.00528seconds
  
```

图 2 第一次运行结果