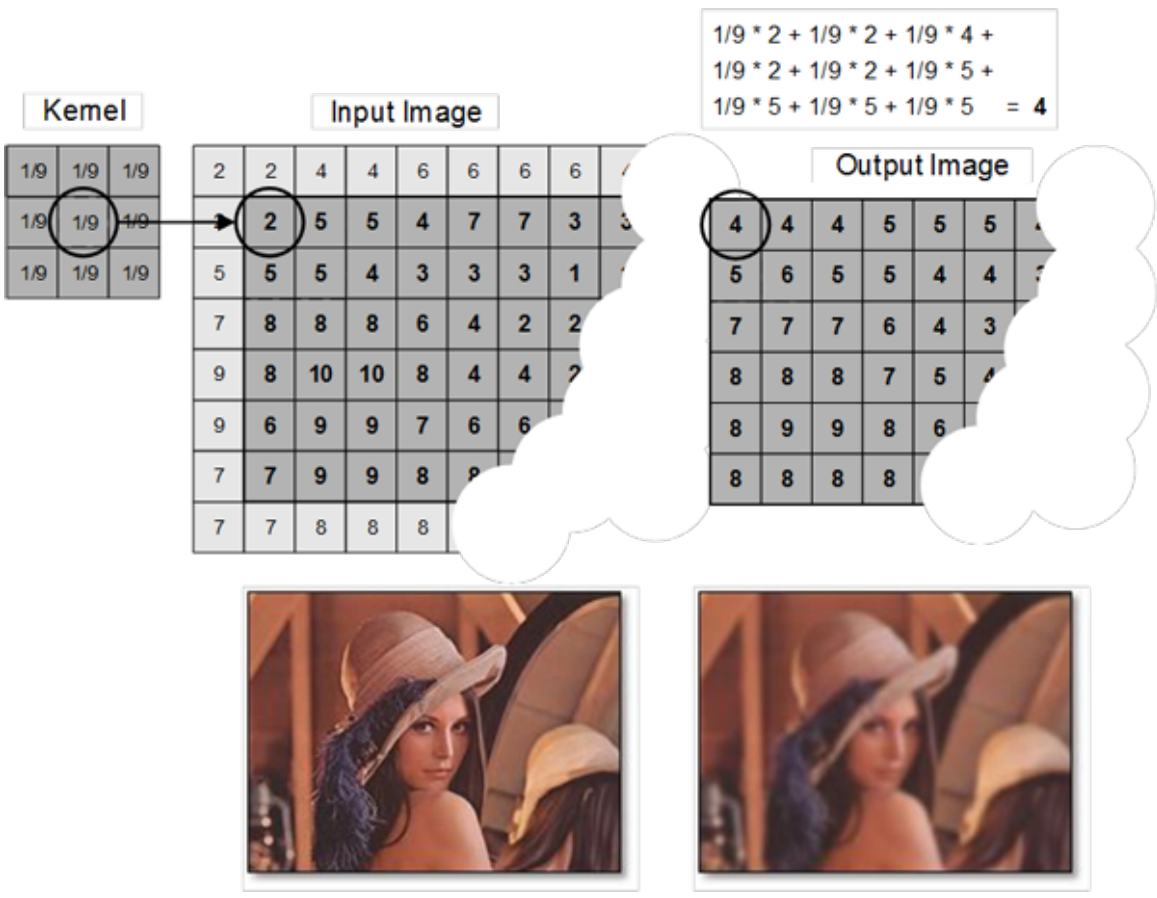
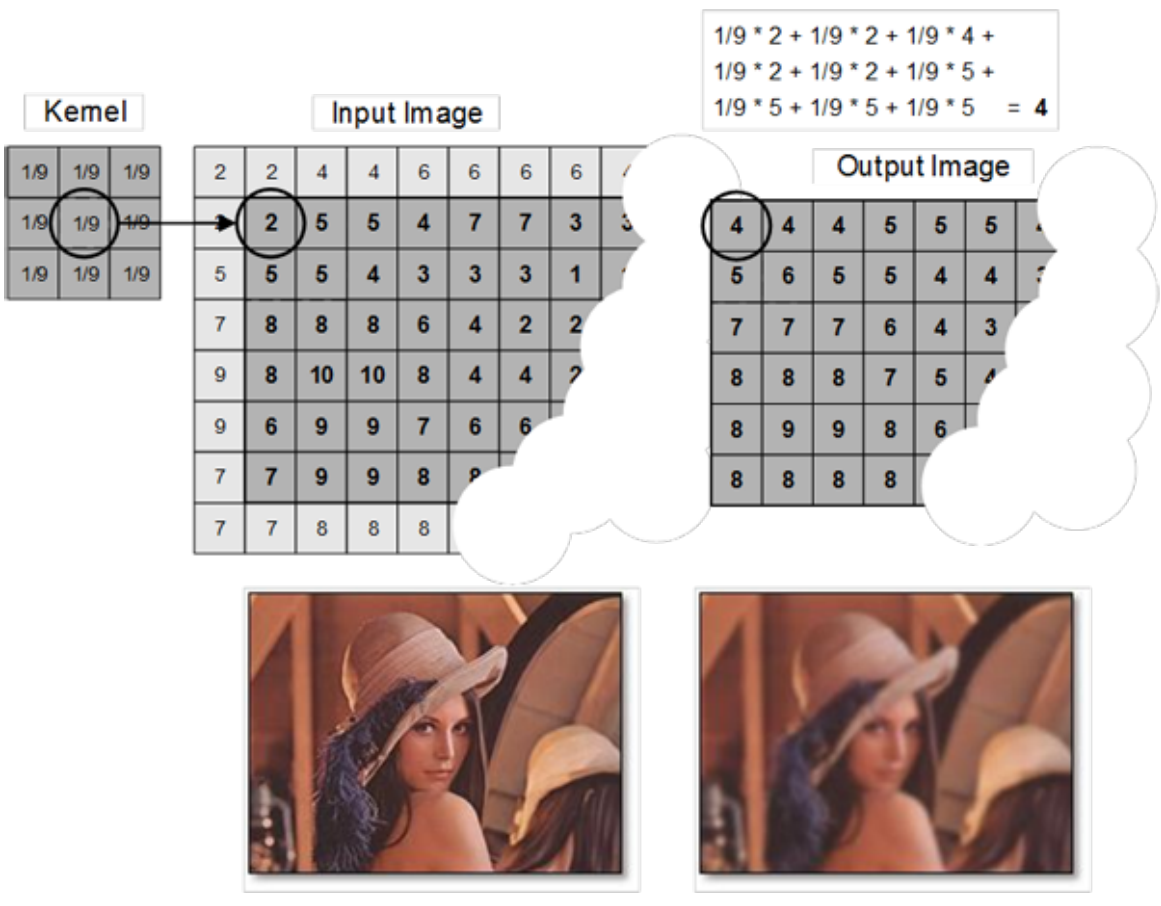
异构计算案例研究：图像卷积滤波

由 技术编辑archive1 于 星期二, 2015-09-15 17:18 发表



在先前发表的一篇文章中，我为PowerVR Rogue GPU上如何编写OpenCL内核作了简单的介绍；这篇文章也为下一步奠定了基础：实用案例研究，解析如何使用OpenCL来编写图像卷积内核。

许多图像处理任务，比如模糊、锐化和边界检测可以通过图像与数字矩阵（或内核）的卷积来实现。下图显示了一个3X3的内核，实现了一个平滑滤波器，将图片中每个像素的值替换为包括其自身在内的周边像素的平均值。



内核卷积通常需要图像边界之外的像素值。有很多方法可以用来处理图像边界，比如为卷积扩充最近边界的像素值，或者是排除掉输出图像中需要输入图像边界之外像素值的部分像素，这种方法将降低输出图像的大小。

下表上半部分显示了该滤波器算法的伪代码，下半部分是C代码实现。在C程序中，假设每个像素表示为四个分别代表R、G、B、A的8位数组成的32位整数，宏MUL4执行四个独立乘法，每个乘法对应这四个8位值中的一个。

卷积滤波伪代码

for each image row in input image:
for each pixel in image row:

set accumulator to zero

```
for each kernel row in kernel:
for each element in kernel row:
multiply element value corresponding to pixel value
add result to accumulator
set output image pixel to accumulator
```

卷积滤波的**C**程序

```
void blur(int src[Gx][],int dst[Gx][], char *weight)
{ int x, y, pixel, acc;

for (y=1; y for (x=1; x

acc = 0;

for (j=-1; j<=1; j++) {
for (i=-1; i<=1; i++) {
pixel = MUL4(src[y+j][x+i], weight[j+1][i+1], 16);
acc += pixel;
} }
dst[y][x] = acc;
}
}
}
```

使用上一篇文章介绍的方法，从串行控制流中（外部的两层嵌套循环）抽取计算内核，并且遵循这里的编程指导，将生成下面的OpenCL内核。

```
__attribute__((reqd_work_group_size(8, 4, 1)))
__kernel void blur(image2d_t src, image2d_t dst, sampler_t s, float *weight)
{
int x = get_global_id(0);
int y = get_global_id(1);
float4 pixel = 0.0f;
for (j=-1; j<=1; j++) {
for (i=-1; i<=1; i++)
pixel += read_imagef(src, (int2)(x+i,y+j), s) * weight[j+1][i+1];
}
write_imagef(dst, (int2)(x,y), pixel/9.f);
}
```

表达式

```
__attribute__((reqd_work_group_size(8, 4, 1)))
```

在编译时将工作组的大小设置为32（8X4）。这将宿主程序限制成将内核排队成一个8X4配置的2D范围，但在编译时提升了内核的性能。

函数声明

```
__kernel void blur(image2d_t src, image2d_t dst, sampler_t s, …
```

指明了OpenCl图像的参数以及从系统内存访问图像数据的采样器。为实现前面例子中边界像素的行为，宿主程序需要将采样器配置为clamp-to-nearest-border（这里没有显示）

表达式

```
float4 pixel = 0.0f;
```

定义了4个32位浮点值组成的一个向量。线程使用浮点算法来执行卷积，相较于整数或者是字符数据，浮点算法的吞吐率更高。

表达式

```
read_imagef(src, s, (int2)(x+i,y+j))
```

指明TPU将系统内存的一个像素采样到本地内存中，然后将R、G、B、A的值卷积为四个32位浮点值，再次它们放置在一个宽度为4的向量中。该卷积由硬件高效执行，仅需要多处理器发射一条指令即可。

表达式

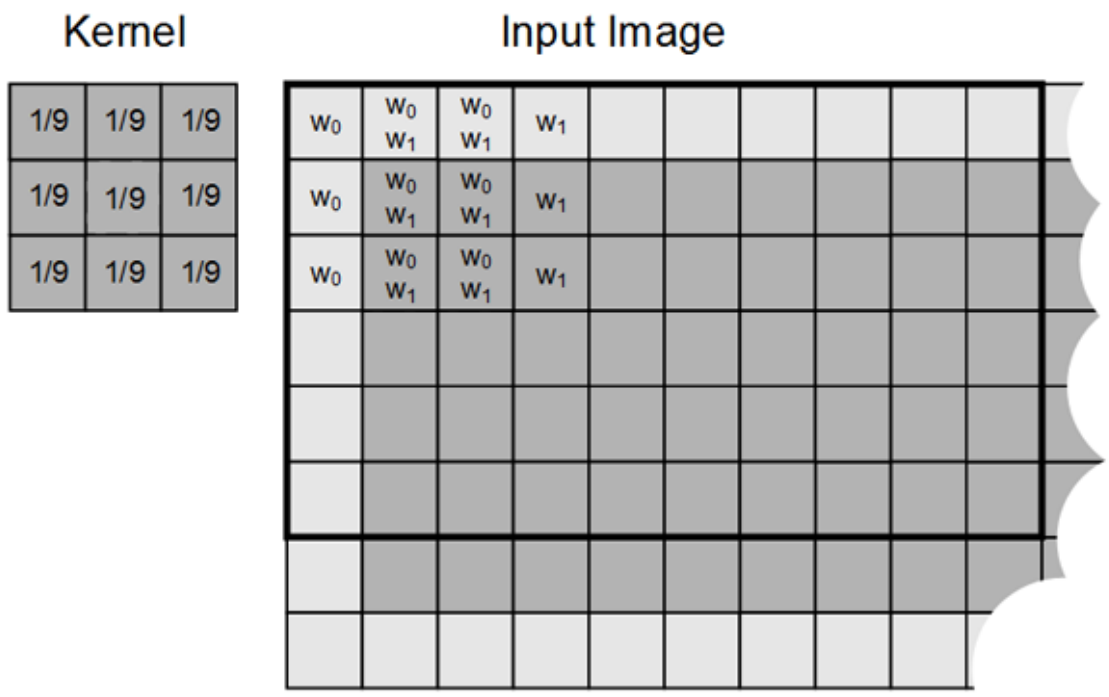
```
write_imagef(dst, (int2)(x,y), pixel/9.f);
```

将一个（归一化）的输出像素值写回到系统内存中。

将常的数据缓存到通用存储中

在上一部分的示例中，所有的工作项之间相互独立运算，每个工作项独立地采样9个输入像素并计算一个输出像素。整体来说，内核的计算密集度较低（即乘累加运算与内存采样运算的比值较低），因此导致性能较低。

对于大小为32的工作组，需要执行总计288（9X32）次采样运算。然而，如下图所示，近邻的工作项使用输入图像的6个重叠像素。



通用存储是片上快速存储器，可用于优化对常用数据的访问，也可用于存储工作组中工作项间的共享数据，从而降低一个工作组内采样运算的数量。典型的编程模式通过读取工作组中的每个工作项来将系统内存中的数据搬运到通用存储中。

- 从全局内存中加载数据到本地内存。
- 同步该工作组的所有工作项，确保所有工作项阻塞直到所有内存读取结束。
- 处理本地内存中的数据。
- 同步工作组中的所有工作项，确保所有工作项已经将结果写入本地内存中。
- 将结果写回全局内存。

下面程序是上一个程序的重新调优，重写后，利用本地内存来降低系统内存采样操作的数量。

```
__attribute__((reqd_work_group_size(8, 4, 1)))
__kernel void blur (image2d_t src, image2d_t dst, sampler_t s, float *weight)
{
int gid = (int2)(get_group(id(0)*8, get_group_id(1)*4);
int lid = (int2)(get_local_id(0), get_local_id(1));
float4 pixel = 0.0f;

__local float4 rgb[10][6];
prefetch_texture_samples_8x4(src, sampler, rgb, gid, lid);

for (j=-1; j<=1; j++) {
for (i=-1; i<=1; i++)
pixel += rgb[lid.x+1+i][lid.y+1+i]) * weight[j+1][i+1]);
}
write_imagef(dst, (int2)(x, y), pixel/9.f);
}

void prefetch_texture_samples_8x4(image2d_t src, sampler_t s, __local float4 rgb [10][6], int2 gid, int2 lid)
{
if (lid == 0) {
// work-item 1 fetches all 60 rgb samples
for (int i=-1; i<9; i++) {
for (int j=-1; j<5; j++)
rgb[i+1][j+1] = read_imagef(src, s, gid+(int2)(i, j));
}
}
barrier(CLK_LOCAL_MEM_FENCE);
}
```

表达式

```
__local float4 rgb[10][6];
```

定义了一个本地阵列，分配在通用存储中。

内核首先调用了函数

```
void prefetch_texture_samples_8x4( …
```

在该函数中，工作组中的所有工作项首先一起测试它们的本地ID，工作项0从内存中采样数据到通用存储；所有的工作项在barrier处同步。这种同步操作是必要的，以防止其它工作项试图从通用存储中读取未初始化的数据。在主内核中，read_imagef调用被替换为从本地内存阵列rgb中读取数据。

优化后的程序，每个工作组执行总计60次采样操作，所有都发生在初始化阶段，而对应的原程序则需要执行288次内联采样操作。这降低了内存带宽需求并有效提升了性能。

预取函数可以进一步改进，原来每个工作项顺次读取60个采样，可以替换为30个工作项每个依次取回两个采样。下面示例显示了这种方法的实现方式。

```
inline void prefetch_8x4_optimized(image2d_t src, sampler_t s, __local float4 rgb[10][6])
{
// Coord of wi0 in NRDange
int2 wi0Coord = (int2)(get_group_id(0)*8, get_group_id(1)*4);
```

```
// 2D to 1D address (from 8x4 to 32x1)
int flatLocal = get_local_id(1)*8 + get_local_id(0);

// Only first 30 work-items load, each loads 2 values in sequence
if (flatLocal < 30)
{
/* Convert from flatLocal 1D id to 2D, 10x3 */
int i = flatLocal % 10; // Width
int j = flatLocal / 10; // Height

/* 30 work iteams reads 10x3 values,
* values 0-9, 10-19, 20-29 from 10x6 - top half
*/
rgb[j][i] = read_imagef(src, s, (int2)(wi0Coord.x + i - 1, wi0Coord.y + j - 1));

/* 30 work iteams reads 10x3 values,
* values 30-39, 40-49, 50-59 from 10x6 - bottom half
*/
rgb[j + 3][i] = read_imagef(src, s, (int2)(wi0Coord.x + i - 1, wi0Coord.y + j + 3 - 1));
}
barrier(CLK_LOCAL_MEM_FENCE);
}
```

最好情形下，每个时钟周期工作项都可以从通用存储中取回数据。但实际上，需要满足一系列条件才能达到最高效率。

计算机视觉将是我们异构计算系列文章中下一个要关注的焦点：

扩展阅读

下面是文章列表，可以助你导航到异构计算系列中发表的每一篇文章：

- PowerVR异构计算完全汇编
- 移动系统的异构计算入门
- PowerVR Rogue GPU上编写OpenCL内核快速指南
- 提升异构软件的性能和功耗表现
- 面向安卓的PowerVR图像框架
- 异构计算案例研究：图像卷积滤波
- 深入解剖：使用PowerVR实现计算机视觉
- 深入解剖：计算机视觉中的硬件IP
- 深入解剖：PowerVR中的OpenCL脸部识别
- PowerVR图像框架中的零拷贝流支持
- PowerVR图像框架照相演示
- 衡量GPU计算性能
- Imagination在移动计算方面的精巧高效方法

原文链接：
http://blog.imgtec.com/powervr/heterogeneous-compute-case-study-image-co...

欢迎扫码加入移动GPU开发交流群！

相关文章

- 【视频】GoogLeNet和AlexNet模型处理图像演示
- 32位超前进位加法器的设计
- 计算机视觉随谈
- 基于曲率的图像处理
- 网友 loogon 问：复杂图像处理及显示等应用怎样和互联网结合起来呢
- 【干货】图像卷积与滤波的一些知识点

- Android 图像处理(一)：Shader
- 虚拟机上可运行多少软件?
- 资深攻城狮解读5个被误解的CPUGPU概念
- “行动”是唯一的出路