CUDA 架构下的快速图像去噪

李 军,李艳辉,陈双平
LI Jun,LI Yan-hui,CHEN Shuang-ping

暨南大学 珠海学院 计算机系,广东 珠海 519070
Department of Computer,School of Zhuhai,Jinan University,Zhuhai,Guangdong 519070,China
E-mail:tlljyh@jnu.edu.cn

LI Jun,LI Yan-hui,Chen Shuang-ping.Fast image denoising with CUDA.Computer Engineering and Applications,2009,45(11):183-185.

Abstract: The huge computation load is necessary commonly for the image processing,the research on the fast algorithm of image denoising is very important for it is often used in the field.GPU(Graphic Processing Unit) is powerful for Data-Parallel Processing but it is idle for the most of time.CUDA(Compute Unified Device Architecture) is a simple and easy to use for the general purpose computation.A fast algorithm of image denoising is proposed based on the CUDA,it uses the power of GPU to accelerate the process of image denoising and reduce the time significantly.

Key words: Graphic Processing Unit(GPU);image denoising;Compute Unified Device Architecture(CUDA);data-parallel Processing

摘 要:图像处理通常需要较大的计算量,其中图像去噪是经常使用的一种预处理算法,研究其快速算法具有重要意义。图形处理器具有强大的并行计算能力,但大部分时间处于闲置状态。统一计算设备架构提供了一种简单易用的开发环境,可利用图形处理器进行通用计算。提出了基于统一计算设备架构的快速图像去噪算法,可以利用 GPU 的计算能力,加快去噪过程,显著地减少计算时间。

关键词:图形处理器;图像去噪;统一计算设备架构;并行数据处理
DOI:10.3778/j.issn.1002-8331.2009.11.055 **文章编号**:1002-8331(2009)11-0183-03 **文献标识码**:A **中图分类号**:TP391

图像去噪是图像处理领域的一项关键技术,在图像处理领域的有着广泛用途。图像处理的本质是大规模矩阵运算,特别适合并行处理。但 CPU 通用计算很难利用该特性。GPU 在并行数据运算上具有强大的计算能力,特别适合做运算符相同而运算数据不同的运算[1-2],当执行具有高运算密度的多数据元素时,内存访问的延迟可以被忽略[1]。用 GPU 进行图像去噪可以加快图像去噪的速度,该理念可以拓展到其他图像处理领域。运用 NVIDIA 统一计算设备架构 CUDA 在 GPU 上进行 KNN 滤波器图像去噪,提高了 KNN 滤波器图像去噪的运行效率,计算效果令人满意。相对 CPU 而言,运算速度取得重大提高。

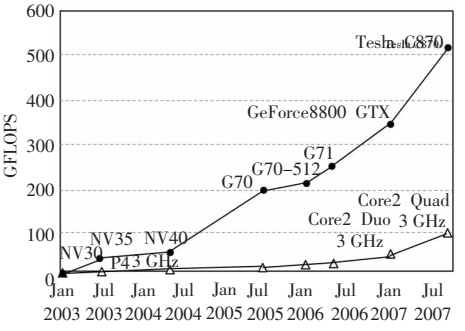


图 1 CPU 与 GPU 浮点性能的比较

1 关键技术介绍

1.1 GPU 简介

由于高性能图形计算需求的推动,GPU 技术发展极为迅速,其计算性能的发展速度远超过 CPU 计算性能的发展速度,见图 1。从图 1 可知,一般通用 CPU 的计算性能只有 10~20GLFOPS,而 GPU 的计算性能已经达到数百 GLFOPS[2]。

2007 年 5 月,NVIDIA 推出了面向高性能计算的 Tesla C870 GPU,专为计算程序所设计。该 GPU 产品包含了主频为 1.35 GHz 的 128 个并行处理单元,1.5 GB 的专用存储器,其峰值计算性能达到 518GFLOPS,支持 IEEE 754 单精度浮点运算,和 CPU 浮点单元一样支持各种高级的浮点操作,GPU 内存访问带宽高达 76.8 GB/s,远高于常规的 CPU 内存访问带宽[3]。NVIDIA Tesla GPU 计算解决方案在设计上与现行的 IT 基础架构做到了无缝衔接。

GPU 强大的计算能力,通常是为 3D 图形计算而设计,主要用于游戏领域,在这类应用之外,GPU 大部分时间处于空闲,造

基金项目:国家自然科学基金(the National Natural Science Foundation of China under Grant No.60802026)。
作者简介:李军(1973-),男,博士,主要研究方向:计算机系统结构;李艳辉(1979-),女,硕士,主要研究方向:图像处理、CAD;陈双平(1976-),男,博士,主要研究方向:智能计算。
收稿日期:2008-10-09 **修回日期**:2009-01-12

成资源的极大浪费。CUDA 架构的提出,使得利用 GPU 进行通用计算变得简单易行。此外,通过将多个 GPU 组成机群,其计算能力可以同超级计算机媲美[4]。

1.2 CUDA 软件开发环境

2007 年 7 月,随着一系列 NVIDIA Tesla GPU 计算解决方案的发布,NVIDIA 公司宣布提供 NVIDIA CUDA 1.0 版本的 C 语言编译器和软件开发套件(SDK),还支持许多 C++的特性。CUDA 代表"计算统一设备架构",是 NVIDIA 公司专为计算而发展的一套应用软件开发环境与工具软件。结合 GPU 运算技术和 CUDA 软件开发环境,为对运算能力要求极高的计算密集型应用程序提供了极具弹性的大型并行计算平台[5]。此外,公司还发布了一款可让 MATLAB 程序运用标准 GPU 函数库加速应用程序运算的插件套件模板。CUDA 软件堆栈包括一个硬件驱动程 CUDA Driver,一个应用程序编程接口 Application 和它的 Runtime,还有 CUDA 的函数库 CUDA Libraries,如图 2 所示[2]。

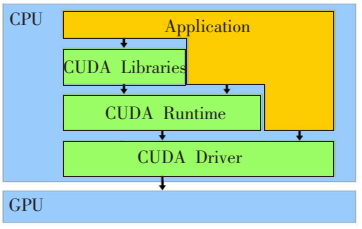


图 2 统一计算设备架构 CUDA 的软件堆栈

CUDA 使得熟悉 C 语言的编程人员专注于开发并行程序而不是处理图形 API。为了简化开发,CUDA 允许程序员将 CPU 和 GPU 的代码混合记录到一个程序文件中。NVCC 作为编译器驱动程序负责从主机代码中分离出设备代码,如图 3 所示。与常规的 GPU 编程不同,CUDA 软件使得 GPU 编程和在 CPU 上编程一样方便,函数是在主机 CPU 上执行还是在 GPU 上执行,由函数的声明和函数所属的库在编译时决定,并不需要在程序执行时进行相应的处理或进行特别的编程说明。CUDA 程序把要处理的数据细分成更小的区块,然后并行处理。采用 CUDA 技术的 GPU 既可作为灵活的线程处理器来运行,由数千个计算程序来调用线程,协作解决复杂的问题,也可作为流处理器运行在具体的应用程序中,其中的各个线程并不进行信息交流[6-7]。

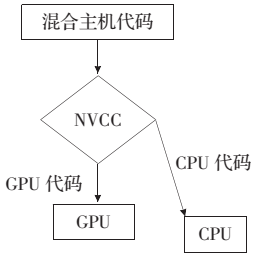


图 3 主机代码分离

目前,NVIDIA 仅对基于 Tesla 架构的 GPU 开放 CUDA 功能,包括 G80 以后的显卡和最近的 Quadro 显卡和 Tesla 显卡。本文实验使用的显卡是 GeForce 8400M GS,该显卡该有 16 个统一渲染单元,计算兼容性数值为 1.1,支持更多的 CUDA 原子函数。

2 GPU 上 KNN 算法的设计与分析

KNN 滤波器对乘性噪声的处理效果非常好,对椒盐噪声的处理效果最差,对高斯噪声的处理效果在三者中居中。选择 KNN 作为算法平台,不仅是因为 KNN 滤波器具有实用性,更重要的是 KNN 滤波器的运算量较大,有利于检测 GPU 的并行运算能力,发挥与 CPU 的比较优势。另外,略去浮点运算的误差,则 KNN 滤波器在 GPU 和 CPU 上的去噪效果是相同的。

2.1 KNN 程序参数设置

KNN 滤波函数的表达式如式(1)所示。

$$KNN_{h,r}(x)=\frac{1}{C(x)}\int_{\Omega(x)}u(y)e^{\frac{|y-x|^2}{r^2}}e^{\frac{|u(y)-u(x)|^2}{h^2}}\mathrm{d}y \quad (1)$$

$\Omega(p)$的大小与图像的大小有关,p 为一个特定的像素点,通常来说 $\Omega(p)$取 5×5 或 7×7 去噪效果较好。实验程序中取 7×7 来进行计算,又由于程序中的 *window Radius* 的值与 $\Omega(p)$邻域的边长 N 是相关的,即 N=2×*window Radius*+1,所以在实验程序中取 *window Radius*=3,即使得 N=7。

2.2 GPU 平台 KNN 设计

2.2.1 元素拷贝内核函数设计

内核函数的工作原理是:被设计成并行数据处理的内核 kernal 的 CUDA 程序,发送到设备上的栅格 Grid 上执行,Grid 里包含若干个块 Block,每个块 block 则在一个多处理器上运行,每个 Block 执行若干线程。一个具体的配置构成内核程序的执行环境,它在程序启动的时候被指定。

元素拷贝内核函数的声明如下:

```
__global__ void Copy(TColor *dst,int imageW,int imageH)
```

被 __global__ 限定词声明的函数将作为一个内核 kernal 函数,在主机上调用,在 GPU 上执行。NVCC 会将这一部分分流到 GPU 上进行并行数据计算。

Copy 内核函数将图像的各像素并行送入 GPU,Copy 内核函数的调用情况如下:

```
int iDivUp(int a,int b){return((a % b)! =0)?(a/b+1):(a/b);}
dim3 grid(iDivUp(imageW,BLOCKDIM_X),
          iDivUp(imageH,BLOCKDIM_Y));
          dim3 thread(BLOCKDIM_X,BLOCKDIM_Y);
Copy<<<grid,threads>>>(d_dst,imageW,imageH);
```

其中,grid.*x**grid.*y* 就等于被发送的块数量,即发送了 iDivUp(imageW,BLOCKDIM_X)*iDivUp(imageH,BLOCKDIM_Y)个 block;thread.*x** thread.*y* 等于每个块的线程数量,即每个块(block)内有 BLOCKDIM_X*BLOCKDIM_Y=8×8=64 个 thread。

2.2.2 KNN 内核函数设计

KNN 内核函数的工作原理与 Copy 内核函数基本一致,只是 Copy 内核函数除了复制像素到全局内存外,不做其他处理;KNN 内核函数需按照 KNN 滤波函数的要求对像素进行去噪处理后,才能把某一像素点去噪后生成的新的像素值写入全局内存。

KNN 滤波函数的声明如下:

```
__global__void KNN(TColor*dst,int imageW,int imageH,float Noise,
float lerpC);
```

KNN 的调用情况如下:

```
int iDivUp(int a,int b){return((a % b)! =0)?(a/b+1):(a/b);}
dim3 grid(iDivUp(imageW,BLOCKDIM_X),
          iDivUp(imageH,BLOCKDIM_Y));
dim3 thread(BLOCKDIM_X,BLOCKDIM_Y);
KNN <<<grid,thread >>>(d_dst,imageW,imageH,1.0f/(knnNoise*
knnNoise),lerpC);
```

2.2.3　BMP 图片加载主机代码设计

BMP 图片的加载代码为主机代码，在 CPU 上运行。读入 24 深度 BMP 图像的关键是要读入 BMP 图像的两个结构体文件 BMPHeader 和 BMPInfoHeader，这里面包含了对图像处理需要的信息，比如 BMPInfoHeader.width 是图像的宽度，而 BMP-InfoHeader.height 则是图像的高度。

BMPHeader 结构体文件的声明如下：

```
typedef struct{
  short type;
  int size;
  short reserved1;
  short reserved2;
  int offset;
}BMPHeader;
```

BMPInfoHeader 结构体文件的声明如下：

```
typedef struct{
  int width;
  int height;
  ……
  int yPelsPerMeter;
  int clrImportant;
}BMPInfoHeader;
```

读入 BMP 图像的关键代码设计如下，确定指针位置后分三次分别读入 RGB 三色值。

```
fseek(fd,hdr.offset-sizeof(hdr)-sizeof(infoHdr),SEEK_CUR);
for(y=0;y<infoHdr.height;y++){
  for(x=0;x<infoHdr.width;x++){
    (*dst)[(y*infoHdr.width+x)].z=fgetc(fd);
    (*dst)[(y*infoHdr.width+x)].y=fgetc(fd);
    (*dst)[(y*infoHdr.width+x)].x=fgetc(fd);
  }
  for(x=0;x<(4-(3*infoHdr.width)%4)%4;x++)
    fgetc(fd);
}
```

2.3　CPU 平台 KNN 设计

为了在实验中与 GPU 上 KNN 滤波运算进行对比，设计了 CPU 上的 KNN 算法。对每一个 7×7 的领域使用式(1)的 KNN 的滤波函数对中心点的像素进行去噪，这一过程由一个叫 process 的函数完成。为了测量 KNN 在 CPU 上每秒处理的像素个数使用了头文件 time.h，begin 和 end 测试的仅仅只是 KNN 运算结束需要花费的时间。

测量 KNN 在 CPU 上运行速度的过程如下：

```
long finish,start;
start=clock();
for(k=0;k<frame;k++)
for(i=3;i<row-3;i++)
for(j=3;j<col-3;j++)
  process(data,i,j);
finish=clock();
```

测试时间结束后，打印输出 CPU 每秒处理的像素个数：

```
printf("Time Per Frame=%.1lf\n",(double)(finish-start)/ CLOCKS_PER_SEC)
```

3　去噪实验与分析

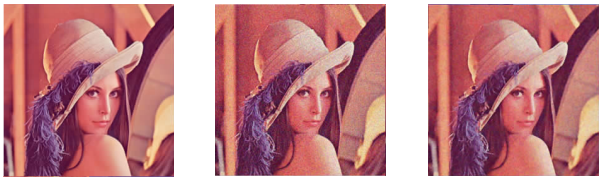
3.1　实验运算平台搭建

硬件平台：GPU 为 NVIDIA GeForce 8400M GS；CPU 为 Intel Core2 Duo 1.5 GHz；内存为 2.0 GB。

软件平台：操作系统为 Microsoft Windows XP Pro SP2；编程环境：Microsoft Visual Studio 2005 Visual C++、Microsoft Visual C++ 6.0；支持软件：NVIDIA CUDA Toolkit 2.0beta、NVIDIA CUDA SDK 2.0beta、NVIDIA CUDA Driver 2.0beta；环境设置：CUDA VS2005 Wizard；辅助工具：Matlab 6.5。

3.2　实验图像预处理

实验研究 KNN 滤波器去噪算法在 GPU 上运行的效果，将 Lena 图像调整为多种分辨率，然后施加高斯噪声，强度参数为 0.02。之所以选择高斯噪声源，一方面它分布最广，另一方面 KNN 算法对高斯噪声图像的效果折中，不失研究的一般性。因为在 CPU 和 GPU 上执行的是同样的算法，它们的去噪效果是等同的，所以这里只比较时间复杂性。图 4 中上面的图像是 Lena 原图，左下方为施加高斯噪声后的图像，右下方为 KNN算法滤波后的图像。各分辨率的图像都要进行这样的处理。



(a)Lena 原图　(b)施加高斯噪声后的图像　(c)KNN 算法滤波后的图像

图 4　Lena 图像去噪效果

3.3　对比实验

用 CUDA 规范进行扩展 C 语言的编程，将 6 张 24 位深度的噪声密度为 0.02 的高斯噪声图像作为输入数据依次输入，其中 KNN 滤波器图像去噪部分将被 NVCC 分流到 GPU 上进行并行数据计算。每次将对应的图像循环去噪 24 次来加大数据量，以准确测算去噪速度，然后去掉最大值和最小值，取平均值。CPU 中进行相应的数据统计，处理结果见表 1 所示。

表 1　去噪速度对比

图像编号	1	2	3	4	5	6
图像大小/像素	500×100	700×300	900×500	1 100×700	1 300×900	1 500×1 100
CPU 耗时/ms	6.07	22.61	47.35	80.13	120.64	169.71
GPU 耗时($\times10^3$ms)	1.81	7.90	17.09	29.34	44.65	63.36
加速比	3 354	2 862	2 771	2 731	2 701	2 678

从表 1 可见，最小的图像也有 500×100=50 000 个数据元素(像素)，最大的图像则有 1 500×1 100=1 650 000 个数据元素，平均数据量达到 72.7 万个像素。总体来看，GPU 比 CPU 的计算速度快三个数量级。随着数据规模的增大，加速比有一定的降低，但是趋于稳定。如果 GPU 内有更多的处理单元，该速度还能更快。需要指出的是，实验中用到的显卡，是支持 CUDA 的低档显卡。所以，将 GPU 用于图像处理显示出极大的潜力。

4　结论

过去的图像去噪及其他图像处理研究，主要集中在对算法的研究和改进来提高大规模图像处理的运行速度。本文运用 NVIDIA 统一计算设备架构 CUDA 在 GPU 上进行 KNN 滤波器图像去噪运算，说明了在 CUDA 架构下进行图像处理的一般方

(下转 222 页)