CUDA 并行技术与数字图像几何变换①

覃方涛　房　斌　(重庆大学 计算机学院模式识别研究所 重庆 400030)

摘　要： CUDA 是 GPU 通过并发执行多个线程以实现大规模快速并行计算能力的技术，它能使对 GPU 编程变得更容易。介绍了 CUDA 基本特性及主要编程模型，在此基础上,提出并实现了基于 NVIDIA CUDA 技术的图像快速几何变换。采用位置偏移增量代替原变换算法中大量乘法运算，并把 CUDA 技术的快速并行计算能力应用到数字图像几何变换中,解决了基于 CPU 的传统图像几何变换运算效率低下的问题。实验结果证明使用 CUDA 技术，随着处理图像尺寸的增加，对数字图像几何变换处理效率最高能够提高到近 100 倍。

关键词： CUDA; 并行化; 数字图像; GPU 编程; 位置偏移增量

Computer Unified Device Architecture Parallel Technology for Image Geometric Transform

QIN Fang-Tao, FANG Bin

(Chongqing University Institute of Computer Science of Pattern Recognition, Chong qing 400030, China)

Abstract: CUDA is a type of technology that performs general purposes fast and parallel computation by running tens of thousands of threads concurrently. It makes users develop general GPU programs easily. This paper analyzes the distinct features of CUDA and summarizes the general program mode of CUDA. Furthermore, it proposes and implements an image geometric transform process by the CUDA. By using a position incremental offset method, instead of a large number of multiplications of the original algorithm, and by applying the rapid parallel computing ability of CUDA to transform the geometric image, solve the problem of the inefficient image transform based on CPU. The experimental results show that, as the image size increase, application of CUDA on image geometric transform can reach to more than 100x.

Keywords: CUDA; parallelization; digital image; GPU programming; position offset incrementa

1　引言

在某些如车牌识别系统等对实时性要求较高的有大量图像处理工作的系统开发中，对从摄像设备中获取的原始图像的预处理所耗费的时间占了很大比重，如何最大限度地减少图像预处理所耗费的时间成为提高这类实时系统性能的关键。图像几何变换是预处理的一个重要方面，并行化是解决这类问题的一个思路。近年来计算机图形处理器(Graphic Processing Unit,GPU)高速发展，极大地提高了计算机图形处理的速度和图形质量,同时也具有高度的可程序化能力，这使得通过 GPU 实现数字图像几何变换并行化成为可能。由于显示芯片通常具有相当高的内存带宽，以及大量的执行单元，绘制流水线的高速度和并行性为图形处理以外的通用计算提供了良好的运行平台,这使得基于 GPU 的通用计算成为近几年人们关注的一个研究热点。 CUDA(Computer Unified Device Architecture)是 NVIDIA 的 GPU 编程模型，它的编程模式是单程序多重数据(Single Program Multiple Data,SPMD) ,即多个并发的线程执行单一的程序来处理多重数据[1]。本文提出的基于 CUDA 的图像并行几何变换并不仅是将传统的基于 CPU 的图像几何变换算法简单机械移植到具有并行计算能力的 GPU 中。通过对传统的数字图像几何变换的分析,尽量减少原算法中的冗余乘法运算，然后利用 CUDA 技术并

① 收稿时间:2010-01-19;收到修改稿时间:2010-03-03

发执行的优点,并发执行多个线程对图像的像素点进行同步几何变换，因此大幅提高了变换效率。

2　CUDA 技术概况

2.1 CUDA 的硬件模型

CUDA 的硬件结构模型如图 1，包含多个 SIMD 多处理器，每个多处理器具有 4 种类型的芯片内存储器：本地寄存器，并行数据高速缓存，只读常量高速缓存，只读纹理高速缓存，后三种存储器可以通过主机读或写，并永久存在于相同应用程序的内核启动中。这个模型有以下几个特点[1,2]：

1) 全局读写存储器：GPU 能够从任意位置和存储器中获取数据，也可以将数据写入任何存储器中，这几乎和 CPU 一样具有灵活性。

2) 线程共享存储器：它可以使得在同一个处理器中的线程快速获取数据，避免从全局变量存储器中存取。共享存储器的访问速度几乎寄存器一样快，从这里获取数据仅仅需要 4 个时钟周期，然而全局变量存储器的访问花费为 400-600 个时钟周期。

3) 线程同步：处在同一个线程组的线程组能够进行同步操作，因此他们可以通迅和协作以解决一些复杂的问题。

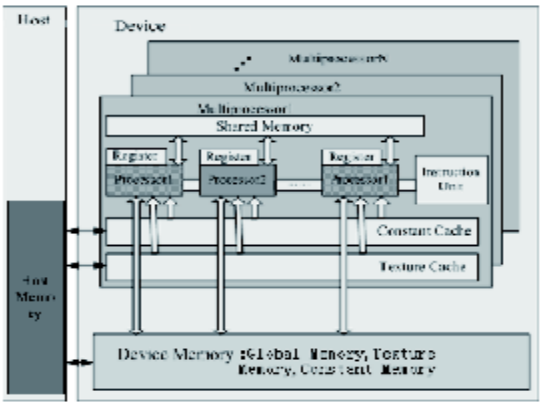

图 1　CUDA 的硬件模型

2.2 CUDA 的编程模型

从 CUDA 体系结构[3,4]的组成来说，它主要扩展了驱动程序和函数库，软件栈(包含驱动，运行时库，一些 API 和 NVCC 编译器)，如图 2 所示。整个程序被 NVCC 统一编译，然后 GPU 和 CPU 再各自编译各自的代码。

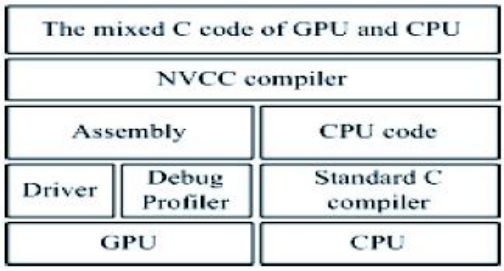

图 2　CUDA 编译程序

CUDA 模型隐藏了低层硬件细节,为了实现数据并行计算，我们可以用如下编程模型[5,6]。它主要有以下几步：

1) 在 CPU 上初始化数据;

2) 将数据传到 GPU ;

3) 运行 kernel 处理数据;

4) 把结果传回 CPU。

3　适于SPMD计算的图像变换改进算法

图像几何变换主要包括图像的平移，缩放，旋转及扭曲等。平移及缩放变换处理较为简单，只需要做加法处理即能得到最终结果，因此在变换算法上没有大的提升空间。图像的扭曲根据各情情况其变换过程较复杂，本文不作详细讨论，但其中常见如平等四边形扭曲等，其变换规律与图像的旋转较为类似。图像的几何变换可以用一个线性方程组来描述：

$$\begin{cases} x' = a_0 x + a_1 y + a_2 \\ y' = b_0 x + b_1 y + b_2 \end{cases} \tag{1}$$

图像的几何变换过程实质就是一个确定线性方程组系数值的过程。由于图像的像素点的位置是离散的，经过式(1)变换以后新的位置为 $P'(x', y')$ 不一定正好落在一个像素上,即所谓的空穴现象[7]。采用逆向变换的方法可以解决这一问题。若变换过程 P 采用直接求式(1)逆变换方法处理新图像的每一个像素点，每个点将会进行 4 次乘法运算，那么对于变换之后的大小为 $M * N$ 的图像，将会产生 $4 * M * N$ 次乘法运算。考虑采用位置增量来代替大量乘法运算，以进一步提高变换效率。将一大小为 $M * N$ 图像任意点 (x, y) 以点 (x_0, y_0) 为中心旋转 q ，变换后的位置为：

$$\begin{cases} x' = (x - x_0)\cos q - (y - y_0)\sin q \\ y' = (y - y_0)\cos q + (x - x_0)\sin q \end{cases} \tag{2}$$

为了避免出现提到的所谓空穴点，采用逆变换方式，因此将式(2)做逆变换得：

$$\begin{cases} x = x'\cos q + y'\sin q + x_0 \\ y = y'\cos q - x'\sin q + y_0 \end{cases} \tag{3}$$

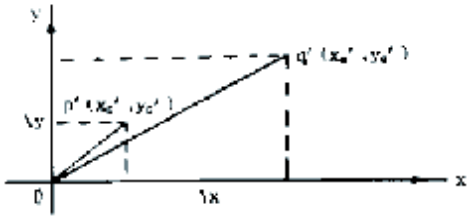

图 3　任意两点位置关系

如图 3 所示，设 p' 和 q' 分别表示变换后新图像中两点，其位置关系可用式(4)表示：

$$\begin{cases} x_{q'} = x_{p'} + \Delta x \\ y_{q'} = y_{p'} + \Delta y \end{cases} \tag{4}$$

将式(4)代入式(3)，最终得到：

$$\begin{cases} x_q = x_p + \Delta x\cos q + \Delta y\sin q \\ y_q = y_p + \Delta y\cos q - \Delta x\sin q \end{cases} \tag{5}$$

设 p' 和 q' 为同一行的两个像素点，则 $\Delta y = 0$。因此在同一行中任意一点 q' 与行首点 X_0 在原图像的位置偏移量[8]，Δx，Δy 为：

$$\Delta x = \begin{cases} 0 * cosq & X_{q'} = X_0 \\ 1 * cosq & X_{q'} = X_1 \\ \mathbf{L} & \\ (n-1) * cosq & X_{q'} = X_n \end{cases} \tag{6}$$

$$\Delta y = \begin{cases} 0 * sinq & X_{q'} = X_0 \\ -1 * sinq & X_{q'} = X_1 \\ \mathbf{L} & \\ -(n-1) * sinq & X_{q'} = X_n \end{cases} \tag{7}$$

由上式可以看出，任意同一行中点 $q'_n, n = 1,2,...,N$ 的 X 方向及 Y 方向上的位置偏移量与其对应的坐标具有线性增量关系。同一列像素点位置偏移关系类似。这样，只要计算旋转图像中原点在原图中的位置，其余像素点均可通过上式求出其位置偏移量，进而计算出在原图像中的位置。由此，对于变换之后的大小为的 $M * N$ 图像，只需要计算原点位置时进行 4 次乘法操作，其余点均可以加法操作代替乘法操作。

由采用位置偏移量的方法将图像变换需要 $4 * M * N$ 次乘法运算变成只需要 $2 * M * N$ 次加法运算，在效率上很大提高。至此，$M * N$ 次加法运算成为图像变换算法中最耗时的部分。注意到每个像素点在变换过程中都是执行相同的加法操作指令，且操作的数据具有线性关系，非常符合并行化的特点，可以进行并行运算。设 X 为输入图像，Y 为输出图像，P 为图像变换过程，|表示并行执行，则图像变换并行化可以表示为：

$$Y = P(X) = P_1(X) \,|\, P_2(X) \,|\, \mathbf{L} \,|\, P_n(X)$$

本文所讨论的并行化基于 CUDA 技术，它的编程模式为 SPMD[9]，其核心就是一条指令能够同时完成多条相同的操作。典型的执行模式如图所示，操作符 P 表示各种指令运算。

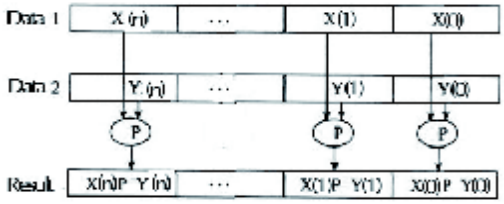

图 4　SPMD 执行模式

设要计算在同一行的 10 个点 $P_0, P_1, ..., P_9$ 的位置，以 $ADDX$ 指令代表对像素点完成 X 方向的加法操作，$ADDY$ 指令完成 Y 方向的加法操作，行首点的位置为 (x_0, y_0)，图 5 表示的过程可以将同时完成计算 $P_0, P_1, ..., P_9$ 的新位置。

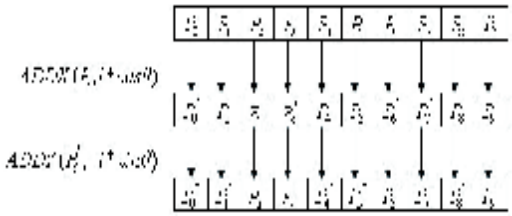

图 5　并行处理示意图

若不采用并行计算，完成上面 10 个像素点的位置计算需要进行 20 次加法运算，而通过将算法分解为并行计算，只需要 2 个加法运算过程即可完成。显然，随着处理像素的增加，通过 CUDA 技术来现实具有大量像素点的数字图像的几何变换的效率会成倍提升。

4　CUDA并行算法描述

根据上面的研究，实现并行几何变换，先求得图像四个顶点变换后新位置进而得出新图像大小，再由式(6)、(7)求出新图像中像素点在原图中的位置偏移，各像素点由各个对应线程同时并发处理。整个并行变换过程可以分为以下几步：

4.1 并行程序初始化

初始化并行环境，开辟显存空间将所要处理的图像信息如像素矩阵及矩阵的宽度和高度送入显存。设置线程数目[10]，设每个线程处理 N 个像素点，每个线程块的线程数为，每个网格含有的线程块数为，因为 NVIDIA 9800GT 显卡一个线程块最多允许的线程数为 512，因此<512;

1)bool InitCUDA();　/*初始化设备*/

2)cudaMalloc((void **)&sDataCopy, sizeof(char)*sWidth*sHight); /*在GPU设备中分配内存*/

3)cudaMemcpy(sDataCopy, sData, sizeof(char)*sWidth*sHight, cudaMemcpyHostToDevice); /*将数据数据由内存复制到显存中*/

4)DeviceTran<<<dimGrid, dimBlock>>>(sDataCopy, pCopy, dimsSize, dimnSize, dimPoint, dimSinCos); /*调用内核*/

第句(1)呼叫函数 InitCUDA 以初始化系统支持 CUDA 的设备，第(2)、(3)句先取得一块适当大小的显卡内存，再把数据复制进去。第(4)句呼叫执行 CUDA 函数。

4.2 计算辅助变量值

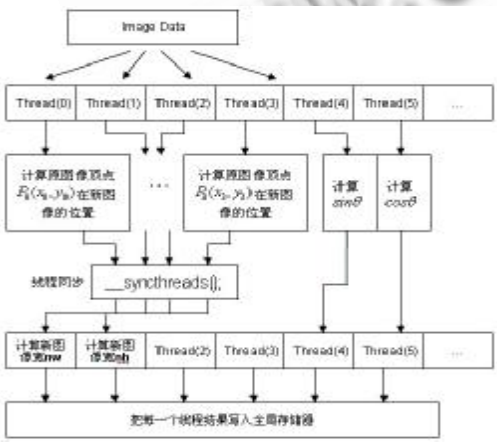

图 6　CUDA 程序运行过程

由式(3)计算四个顶点新位置(x_i', y_i')，新图像的宽度 nw 和高度 nh，以及累加单位值$\cos q$ 、$\sin q$ 等。nw 和 nh 需要先计算出四个顶点的位置，因此四顶点位置可由 4 个 thread 并行计算，同步之后，由两个线程计算出宽度与高度，其执行过程如图 6 所示。

4.3 GPU 完成并行化核心过程

对旋转后大小为 M*N 的新图像各像素点做位置逆运算。计算公式为式(5)，可以看出对于每一行像素点在原图像中的位置偏移仅为$n*\cos q, n=0,1,...,N$，因此，为进一步提高效率可先计算出各行或各列第一个像素点在原图像的位置$P(x,y)$，再作累加操作计算各行或各列像素点在原图中的位置。CUDA 内核关键伪代码为：

```
for (int i=bidx*blockDim.x+ tidx; i<dimnSize.x; i+=wstep){
for (int j=bidy*blockDim.y + tidy; j<dimnSize.y; j+=hstep)  {
(1)temp_x = dimPoint.x + i*dimSinCos.y - j*dimSinCos.x; /*计算新图像在源图中的 x 坐标*/
(2)temp_y = dimPoint.y + i*dimSinCos.x + j*dimSinCos.y; /*计算新图像在源图中的 y 坐标*/
(3)if(temp_x>=0 && temp_x<dimsSize.x && temp_y>=0&& temp_y<dimsSize.y) /*如果坐标没有越界*/
{
(4)nData[dimnSize.x * j+i]
= sData[temp_y * dimsSize.x + temp_x]; /*没有越界则赋灰度值*/
}
Else /*若越界则直接置为白点 */
(5)nData[dimnSize.x * j+i] = 255;
}
}
}
```

i，j，为当前线程在线程块(thread block)的横轴方向和纵轴方向的索引值。由于一个线程可能不止处理一个像素点，所以出现 for 语句。第(1)和第(2)句正是计算当前像素点对应在原图像中的位置。若计算出的位置没有越界，则赋对应像素点的值(第(4)句)，否则，置为白色。其并行处理过程如图 7。

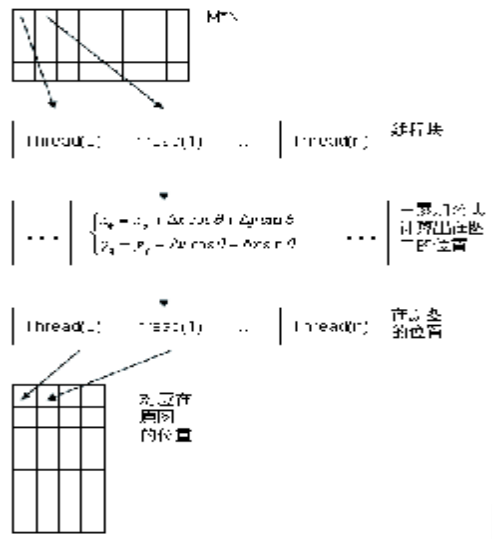

图 7　CUDA 程序运行过程

4.4 数据回传

在显示芯片执行程序不能有回传值，计算完成以后，把结果从显示芯片复制到主内存上。

```
*nData = (char *)malloc(sizeof(char)* new_Width * new_Hight); /*在内存中分配新空间以保存变换后的图像*/
cudaMemcpy(nData, pCopy,
sizeof(char)*new_Width*new_Hight,
cudaMemcpyDeviceToHost); /*将变换后的图像从显存中复制到内存*/
```

4.5 实验结果和分析

实验在 NVIDIA GeForce 9800gt,intel core2 duo E7400 的硬件条件下进行。本文分别实现在 CPU 代码和 CUDA 代码，并且比较了他们在各自的执行时间,比较结果如表 1。

在 CPU 与 CUDA 上的执行时间关系曲线如图 8。可以看出,随着处理图像尺寸逐步增加,CUDA 较 CPU 的相对运行时间开销明显大幅偏少，其并行计算优势显现，效率成倍提升。在输入图像为 8000*5000 时，CPU的计算所花时间为CUDA计算所花时间近40倍，甚至当输入图像为 11200*7000 时，CPU 所花时间为 CUDA 的 105 倍。这是因为在图像很小的时候，运算量并不大,加上 CUDA 数据在宿主与内核之间的传送时间开销，因此还不能充分发挥出其并行计算优势。

表 1　相同图像在 CPU 与 CUDA 上的执行时间

序号	输入图像	CPU（ms）	CUDA(ms)
1	640*400	313	47
2	1120*700	375	47
3	1600*1000	750	62
4	2400*1500	1687	94
5	3200*2000	3016	125
6	4800*3000	6813	203
7	6400*4000	12110	328
8	8000*5000	18969	500
9	11200*7000	96828	921

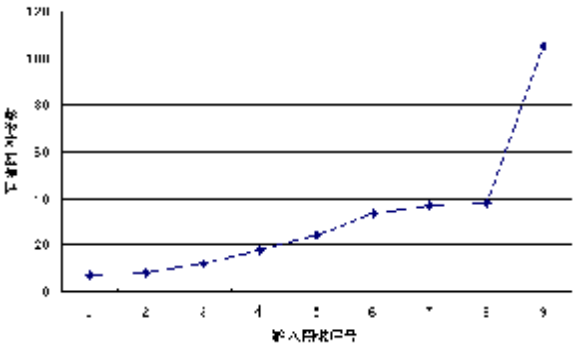

图 8　图像在 CPU 与 CUDA 上耗时比

5　结语

本文通过对基于CUDA技术的数字图像几何形变方法的实现，验证了 CUDA 技术在数字图像处理领域的高效性。采用 CUDA 进行并行计算，我们需要主意两点。一是每一线程处理数据分配问题。如果能找到更合理的数据数据分配算法，图像的处理效率会进一步得到提高。二是宿主设备的数据存取带宽。如何进一步提高数据的存取速度值得进一步观注。

显然 CUDA 为我们提供了一种超大规模并行计算方法，从硬件价格上来说比要实现同等价格所需要 CPU 的价格便宜很多。从实际效果来看，CUDA 非常适合数字图像处理领域这样需要进行大量数据运算以及对实时性要求较高的领域，其效

(下转第 116 页)

(上接第 172 页)

率较传统 CPU 计算而言不言而喻。除了图像的几何变换外，图像的边缘检测算法等都非常适合用 CUDA 技术来实现。

参考文献

1 NVIDIA Corporation. CUDA Programming Guide 1.0, http://www.nvidia.com, 2009,(8):11.

2 Yang ZY, Zhu YT, Pu Y. Parallel Image Processing Based on CUDA. Wuhan, China: 2008 International Conference on Computer Science and Software Engineering, 2008:198－201.

3 NVIDIA CUDA. http://forums.nvidia.com, 2009, (8):13.

4 Halfhil TR. Parallel Processing With CUDA. Microprocessor Report, Scottsdale, Arizona, Jan 28, 2008.

5 Owens JD, Houston M, Luebke D, et al. GPU Computing, Proc. of the IEEE, May 2008,96(5):879－897.

6 Nickolls J, et al. Scalable Parallel Programming with CUDA. ACM Queue, 2008,6(2):40－53.

7 李伟青.图像旋转的快速显示技术.计算机应用研究，1994,(3):13－14.

8 张发存,王馨梅,张毅坤.数字图像几何变换的数据并行方法研究.计算机工程, 2005,31(11):159－161,196.

9 刘振安,章守信,刘胜璞.并行图像处理算法的设计与实现.测控技术,2003,22(5):5－6.

10 Boyer M, Skadron K, Weimer W. Automated Dynamic Analysis of CUDA Programs. STMCS 2008, Boston, Massachusetts, Apr 06, 2008.