第 30 卷第 10 期
2010 年 10 月

计算机应用
Journal of Computer Applications

Vol. 30 No. 10
Oct. 2010

文章编号:1001 - 9081(2010)10 - 2774 - 03

# GPU 加速的二值图连通域标记并行算法

覃方涛,房　斌
(重庆大学 计算机学院, 重庆 400030)
(fantot@ gmail. com)

**摘　要**:结合 NVIDIA 公司统一计算设备架构(CUDA)下的图形处理器(GPU)并行结构和硬件特点,提出了一种新的二值图像连通域标记并行算法,高速有效地标识出了二值图的连通域位置及大小,大幅缩减了标记时间耗费。该算法通过搜索邻域内最小标号值的像素点对连通域进行标记,各像素点处理顺序不分先后并且不相互依赖,因此可以并行执行。算法效率不受连通域形状及数量的影响,具有很好的鲁棒性。实验结果表明,该并行算法充分发挥了 GPU 并行处理能力,在处理高分辨率与多连通域图像时效率为一般 CPU 标记算法的 300 倍,比 OpenCV 的优化函数(CPU)效率高近 17 倍。

**关键词**:GPU 加速; 连通域标记; 并行化; 统一计算设备架构; 8 邻域
**中图分类号**: TP751　　**文献标志码**:A

# GPU accelerated parallel labeling algorithm of connected-domains in binary images

QIN Fang-tao, FANG Bin
(*College of Computer Science, Chongqing University, Chongqing* 400030, *China*)

**Abstract:** In combination of NVIDIA's Graphics Processing Unit (GPU) parallel architecture and hardware features under Compute Unified Device Architecture (CUDA) architecture, a new parallel labeling algorithm of connected domain was proposed for binary images. It effectively located the connected domain of the binary image and recorded its size at high speed, and significantly reduced the marking time. It recognized the connected domain through searching the minimum labeled pixel value in neighborhood. Because the processing sequence of each pixel is not in particular order and independent from each other, it can be dealt in parallel. The calculation efficiency of the algorithm is independent of the shapes and the quantity of the connected regions, and the algorithm has good robustness. The experimental results show that the algorithm fully plays the parallel processing capability of GPU, and can get a more than 300 times speedup than general algorithm based on CPU and 17 times speedup than OpenCV function (CPU) in processing high-resolution images and multi-connected-domain images.

**Key words:** GPU acceleration; connected-domain labeling; parallelization; Compute Unified Device Architecture (CUDA); 8-neighborhood

## 0　引言

在目标自动识别过程中,首先对获取的图像进行阈值分割及提取,得到二值图像。通常得到的二值图像会包含多个连通区域,系统需要对这些连通域进行标记,再利用目标的几何特性对目标进行自动识别。这一过程中二值图的连通域标记耗时占目标识别的大部分,因此连通域标记算法效率成为影响整个识别系统实时性能的关键。目前连通标记算法主要划分为两类:一类是局部邻域算法[1],其基本思想是从局部到整体,要逐个标记连通成分,对每个连通域都要先确定一个"种子点",再向周围邻域扩展地填入标记;另一类则是"分而治之"算法[2],其基本思想是从整体到局部,先确定不同的连通成分,再对每一个成分用区域填充方法填入标记。两类方法都要对图像做多趟扫描,效率不高。有学者提出一种算法将待处理的图像分为 $N$ 列,对 $N$ 列子图像同时进行连通域标记,最后进行连通域合并处理。由于子图像内仍然是串行处理方式,这并非真正意义上的并行算法,因此处理效率提升有限。

另一方面,随着 GPU 技术的快速发展[3-4],当前的 GPU 已经具有很强的并行计算能力。同时,随着 NVIDIA 公司的统一计算设备架构(Compute Unified Device Architecture, CUDA)[5-7]的推出,使得 GPU 具有更好的可编程性,已经在诸如物理系统模拟、金融建模以及地球表面测绘等通用大规模计算领域有着广泛的应用。如何充分利用 GPU 的并行计算特点实现一些复杂运算的快速求解,已经成为当今的热点问题之一。本文结合 GPU 硬件结构的特点,对传统连通域标记算法[9-12]进行改造,提出一种新的连通域标记并行算法并基于 GPU 并行实现,大幅提高连通域标记的执行效率,进而提升目标自动识别系统的实时性能。

## 1　适合 GPU 的连通域标记并行算法

本算法采用 8 邻域连通,基本思想是先将输入图像的所有像素点进行唯一整数标号,然后扫描各像素点邻域内所有像素点标号,比较当前像素点的标号值与邻域内其他像素点的标号值大小,若当前像素点的标号值在邻域内为最小,则不改变当前像素点的标号,否则将邻域内最小标号值赋予当前

**收稿日期**:2010 - 04 - 07;**修回日期**:2010 - 06 - 02。

**作者简介**:覃方涛(1981 - ),男,四川达州人,硕士研究生,主要研究方向:模式识别;　房斌(1967 - ),男,四川成都人,教授,博士,主要研究方向:模式识别、图像处理、生物特征鉴定、生物医学图像分析、文本处理。

像素点,并标识当前像素点状态为扩展点,设置当前像素点指针指向连通域顶点并修改连通域顶点记录的连通域宽度与高度。重复这一扫描过程直到图像中没有任何像素点需要变换标号时结束。

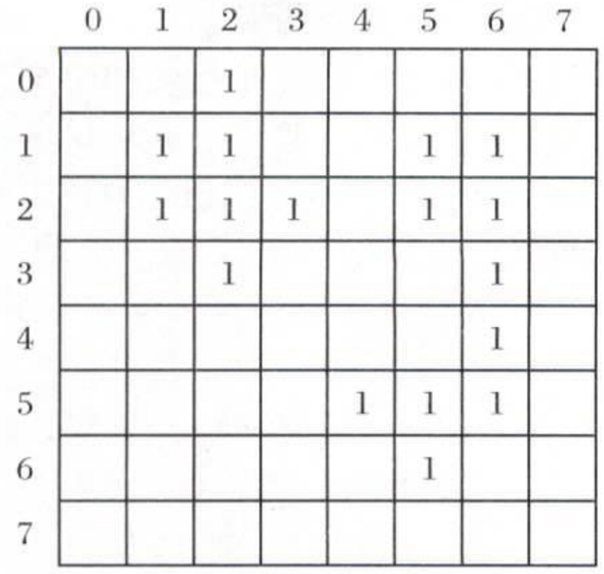



图 1　初始二值图

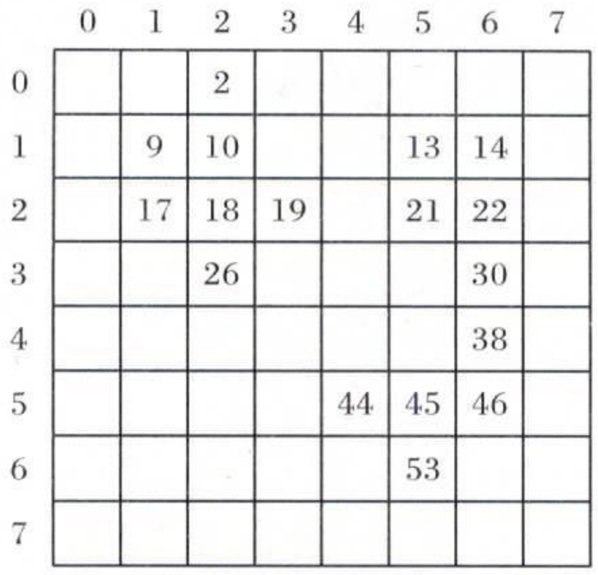



图 2　像素点标号

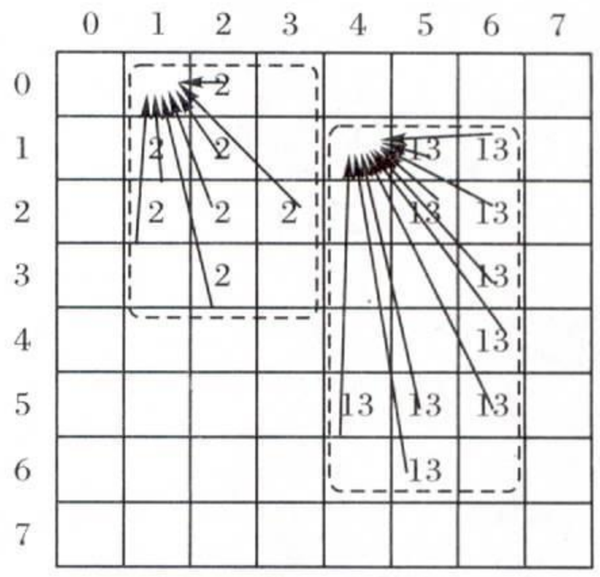



图 3　处理过程示意图

在对像素点进行标号时,采用从左至右、从上至下的顺序,从小到大依次进行,以保证位置较高像素点其标号值较小。为了保证各像素点标号值的唯一性及独立性,像素点标号值只与其本身位置相关,因此可将图像矩阵映射为一维数组,将像素点在数组中的位置作为其标号值(图 2)。各像素点在算法执行过程中始终处于两种状态之一:连通域顶点或连通域内像素点。若判断当前像素点为第二类状态,则只需要调整连通域的高度和宽度即可,对于某些情况特殊的像素点则需调整邻域矩阵的顶点位置。针对本算法的特点设计数据结构 NPixel 如下。

```
Typedef struct NPixel
{ Int code;                          //像素标号
  Int xwidth;                        //连通域宽度或者像素点横坐标
  Int yhight;                        //连通域高度或者像素点纵坐标
  Int flag;                          //像素点类型标识
} NPixel;
```

算法的具体过程描述如下。

1)根据输入图像分配新标记空间。令输入图像为 $IMG_s$,新分配空间为 *Data*。设输入图像大小为 $M \times N \times$ sizeof(char),则新分配空间大小为 $M \times N \times$ sizeof(NPixel)。

2)初始化 *Data*。对 $IMG_s$ 每一像素点,若该点为白点,即 $IMG_s(x,y)$ 值为 1 或者 255 时,则对 $Data(x,y)$ 进行标号,对其成员变量赋初值:

$$\begin{cases} Data(x,y) -> code = y \times N + x \\ Data(x,y) -> xwidth = 0 \\ Data(x,y) -> yheight = 0 \\ Data(x,y) -> flag = 0 \end{cases}$$

若该点值为 0 即为背景点时,不对该点进行标号,令 $Data(x,y) -> code$ 值为 -1。

3)置全局变量 *Completed* 为 1。对 *Data* 中的每个点像素 *P*,比较该点的领域内最小标号点 $P_0$ 的标号是否小于当前点 *P* 的标号。若为真则置 *Completed* 为 0,并将最小标号赋予当前点;设置当前像素点为普通点:$Data(x,y) -> flag = 1$。判断 $P_0$ 点的状态,若 $P_0 -> flag$ 值为 0 即 $P_0$ 为连通域顶点,则将 *P* 指向 $P_0$,即 $P -> xwidth = P_0.x$;$P -> yheight = P_0.y$;若 $P_0 -> flag$ 值为 1 即 $P_0$ 为普通点时,将 $P_0$ 的指针赋予 *P*,即 $P -> xwidth = P_0 -> xwidth$,$P -> yheight = P_0 -> yheight$。此时,*P* 与 $P_0$ 同时指向连通域的顶点。

4)修正连通域顶点位置及连通域大小。判断当时像素点 *P* 与连通域顶点的位置关系,若点 *P* 在当前连通域顶点的左边,即 $P.x < P -> xwidth$,此时需要修正连通域顶点位置并将原顶点标识为普通点,即:

$Data(P.x, P -> yheight) -> flag = 0$;

$Data(P.xwidth, P -> yheight) -> flag = 1$;

并将新的连通域大小赋予新的顶点:

$Data(P.x, P -> yheight) -> xwidth = Data(P -> xwidth, P -> yheight) -> xwidth + abs(P -> xwidth - P.x)$;

$Data(P.x, P -> yheight) -> yheight = MAX(Data(P -> xwidth, P -> yheight) -> yheight, ABS(P -> yheight - P.y))$;

若点 *P* 在当前连通域顶点的右边,则只需要修改连通域的高度及宽度。本算法不需要考虑当前点 *P* 与连通域顶点位置的上下关系,这是因为本算法的像素标号方法保证了处于上方的像素点标号值必然小于下方的标号值。

5)同步,保证所有像素点处理完成一次,操作代码:_syncthreads()。

6)若 *Completed* 为 0 则返回 3),为 1 则算法结束,标记完成。

## 2　GPU 内核配置及实现

本算法运行过程中各像素点计算顺序没有先后,处理结果相互不影响,实现了并行化,可以采用 SIMD 模式,利用支持 CUDA 的 GPU 进行并行计算。在 GPU 中进行计算,需要为计算配置一个处理内核,以便对要处理像素的进行划分,设每个线程处理 *N* 个像素点,每个线程块的线程数为 *thread_n*,每个网格含有的线程块数为 *block_n*,显卡一个线程块最多支持的线程数 512,因此 $thread\_n < 512$。根据 GPU 的寻址特点,*block_n* 取值为 64 的整数倍时,性能更优。对于算法中关键的数据结构 NPixel,在 GPU 中可以用内置向量类型代替(如 int4)。实际应用中,性能会受到多方面因素的影响,如线程切换、访问延迟等,因此应该根据实验结果进行取值。图 4 给出了 GPU 处理内核并行处理过程。

## 3　实验结果与分析

本文所采用的实验平台为 Intel core2 duo E7400,主频为 2.8 GHz,系统内存为 2 GB,显卡采用的是 NVIDIA GeForce 9800GT,显卡内存为 512 MB,显卡的核心频率为 600 MHz。操作系统为 Windows XP,GPU 实现基于 CUDA SDK2.0,实验数据为不同分辨率的 8 位 $M \times N$ 二值灰度图像,$GridDim = M$,$BlockDim = 512$,其中 *GridDim* 为网络线程块数,*BlockDim* 为线程块内线程数。图 5 为两组二值图采用基于 GPU 加速的连通域标记并行算法处理结果示意图。表 1 列出了在 GPU 上执行本算法和 CPU 上执行传统算法进行连通域标记的实

验结果。

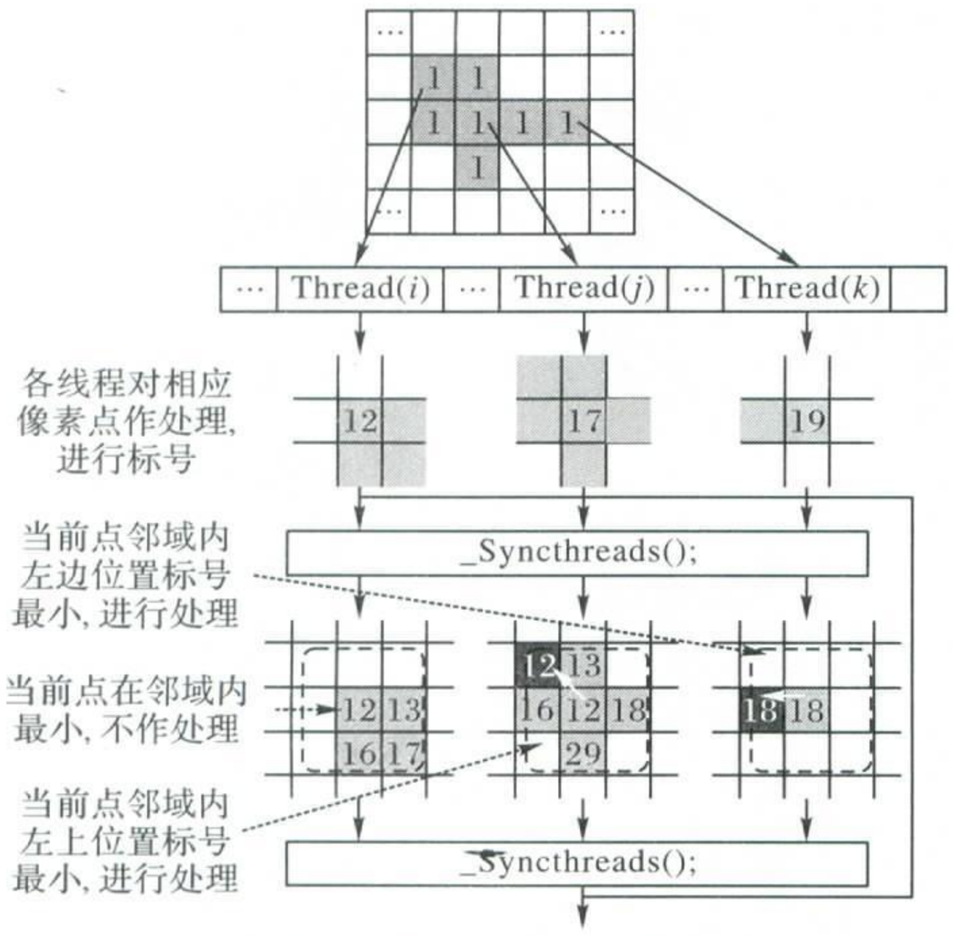
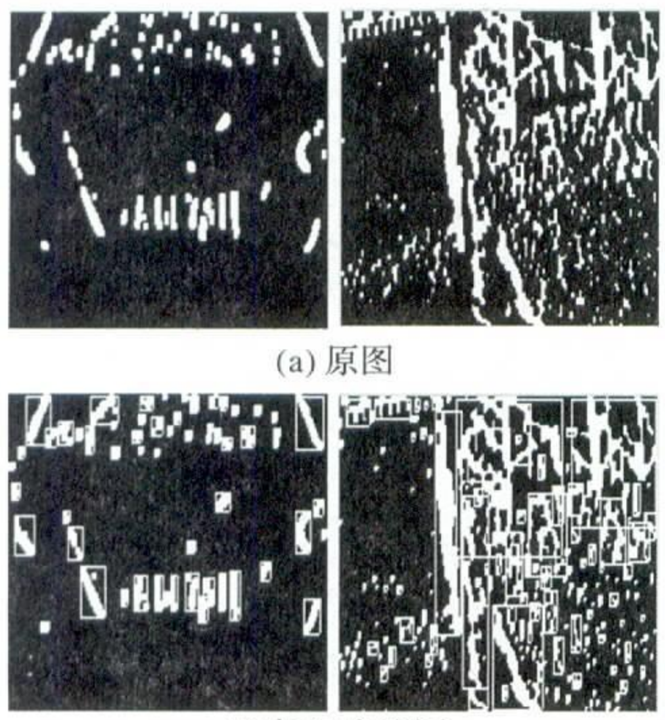



图 4 GPU 并行处理示意图



(a) 原图

(b) 标记完成图

图 5 GPU 并行连通域标记实验效果

**表 1 类图像连通域标记需要的时间**

| 图序号 | 分辨率 | 连通域个数 | 标记时间/ms | | |
|---|---|---|---|---|---|
| | | | 一般标记算法(CPU) | OpenCV 标记函数(CPU) | GPU 加速并行算法 |
| 1 | 600 ×399 | 143 | 10 | 1 | 6 |
| 2 | 600 ×399 | 492 | 15 | 7 | 6 |
| 3 | 1 503 ×1 000 | 709 | 375 | 28 | 9 |
| 4 | 1 503 ×1 000 | 8 235 | 1 972 | 152 | 11 |
| 5 | 4 256 ×2 832 | 4 115 | 7 631 | 189 | 28 |
| 6 | 4 256 ×2 832 | 81 619 | 17 551 | 751 | 47 |

从表 1 可以看出:

1)GPU 加速并行连通域标记算法具有明显的加速效果。特别是在处理 4 256 ×2 832 分辨率图像时,与一般连通域标记算法相比较,最高加速比达 370,与 OpenCV 提供的边界函数连通域标记比较加速比也达到近 16。这是因为 GPU 的高度并行架构,能够同时并行处理多个像素点;另一方面也因为本文提出的并行算法具有充分的并行性,能够充分发挥 GPU 的架构特点,有效提高了运行效率。

2)GPU 在处理低分辨率图像的加速效果不明显,随着图像分辨率的提高,加速优势逐步体现。在处理分辨率为 600 ×399且连通域数量较少的图像时,一般标记算法仅耗时 10 ms,OpenCV 函数更是只需 1 ms,而采用 GPU 加速算法却耗时6 ms,优势并不明显。这是因为,分辨率较低时,运算量并不大,基于 CPU 的方法也能够快速完成。而 GPU 的线程间的切换时间、系统调度开销等相对来讲占用了较多的执行时间。

3)GPU 并行方法运行效率受图像分辨率大小及图像内连通域多少影响不大。理论上同时并行处理 1 个像素点与 100 个像素点耗时相同,而图像中包含连通域的多少对于本并行算法也几乎没有影响,因为本算法所耗费时间取决于图像中此类连通域,即此连通域内存在着两点,此两点"距离"在所有连通域中最大。在没有出现极端情况,"距离"等于该连通域外接矩形高度或宽度的最大值。但对基于 CPU 的方法而言,分辨率大小与连通域多少都对运行效率有较大影响。实验数据也证明这点。在 1 503 ×1 000 同一分辨率条件下,3 号图连通域 709 个,4 号图连通域 8 235 个,CPU 方法耗时比为 4 ~5,但 GPU 处理耗时仅差 2 ms。

## 4 结语

二值图连通域标记是很多目标自动识别系统必要的基本操作,且为较耗时部分,标记效率对整个系统的实时性能有很大影响。本文针对传统串行连通域标记算法效率低下的缺点,提出适合于 SIMD 处理模式,并在 GPU 上实现的图像连通域标记并行新算法。该算法运算规则简单,易于实现。本方法特别适用于处理图像分辨率大、连通区域较多的二值图连通域标记。在 GPU 实现过程中要注意配置合适的内核数据参数,合理的分块参数更能提高处理效率。从价格上看,实现同等计算量的 GPU 价格比 CPU 价格上更经济。因此,对一些需要大规模数据运算,对系统的实时性又要求较高的应用,考虑将传统的串行算法进行改造以 GPU 方式实现是一条新的思路。

**参考文献:**

[1] 章德伟, 蒲晓蓉, 章毅. 基于 Max-tree 的连通区域标记新算法[J]. 计算机应用研究, 2006, 23(8): 168 -170.

[2] 徐利华, 陈早生. 二值图像中的游程编码区域标记[J]. 光电工程, 2004, 31(6): 63 -65.

[3] 宋晓丽, 王庆. 基于 GPGPU 的数字图像并行化预处理[J]. 计算机测量与控制, 2009, 17 (6): 1169 -1171.

[4] OWENS J D, HOUSTON M, LUEBKE D. GPU computing [J]. Proceedings of the IEEE, 2008, 96(5): 879 -897.

[5] NVIDIA Corporation. CUDA programming guide 2.0 [S/OL]. [2010 -01 -03]. http://www.nvidia.com.

[6] NVIDIA CUDA [EB/OL]. [2010 -01 -03]. http://forums.nvidia.com.

[7] HALFHIL T R. Parallel processing with CUDA [J/OL]. Microprocessor Report, 2008, 22(1): 1 -8 (2008 -01 -28) [2010 -01 -08]. http://www.nvidia.com/docs/IO/47906/220401_Reprint.pdf.

[8] NICKOLLS J. Scalable parallel programming with CUDA [J]. ACM Queue, 2008, 6(2): 40 -53.

[9] HIRSCHBERG D S, CHANDRA A K, SARWATE D V. Computing connected components on parallel computers [J]. Communications of the ACM, 1979, 22(8): 461 -464.

[10] HE LIFENG, CHAO YUYAN, SUZUKI K, *et al*. Fast connected-component labeling [J]. Pattern Recognition, 2009, 42(9): 1977 -1987.

[11] HU QINGMAO, QIAN GUOYU, NOWINSKI W L. Fast connected-component labelling in three-dimensional binary images based on iterative recursion [J]. Computer Vision and Image Understanding, 2005, 99(3): 414 -434.

[12] SUZUKI K, HORIBA I, SUGIE N. Linear-time connected-component labeling based on sequential local operations [J]. Computer Vision and Image Understanding, 2003, 89(1): 1 -23.