

文章编号:1007-2780(2016)07-0714-07

基于 GPU+CPU 的 CANNY 算子快速实现

唐 斌^{*}, 龙 文²

(1. 贵州财经大学 信息学院, 贵州 贵阳 550025;

2. 贵州财经大学 贵州省经济系统仿真重点实验室, 贵州 贵阳 550025)

摘要:本文提出一种基于 GPU+CPU 的快速实现 Canny 算子的方法。首先将算子分为串行和并行两部分, 高斯滤波、梯度幅值和方向计算、非极大值抑制和双阈值处理在 GPU 中完成, 将二维高斯滤波分解为水平方向上和垂直方向上的两次一维滤波从而降低计算的复杂度; 然后使用 CUDA 编程完成多线程并行计算以加快计算速度; 最后使用共享存储器隐藏线程访问全局存储的延迟; 在 CPU 中则使用队列 FIFO 完成边缘连接。仿真测试结果表明: 对分辨率为 $1\ 024 \times 1\ 024$ 的 8 位图像的处理时间为 122 ms, 相对应单独使用 CPU 而言, 加速比最高可达 5.39 倍, 因此本文方法充分利用了 GPU 的并行性的特征和 CPU 的串行处理能力。

关 键 词:CANNY; CUDA; GPU; 加速

中图分类号:TP333 **文献标识码:**A **doi:**10.3788/YJYXS20163107.0714

Fast Canny algorithm based on GPU + CPU

TANG Bin^{1*}, LONG Wen²

(1. *School of Information, Guizhou University of Finance and Economics, Guiyang 550025, China;*

2. *Guizhou Key Laboratory of Economics System Simulation, Guizhou University of Finance and Economics, Guiyang 550025, China*)

Abstract: This paper presents a fast method for Canny algorithm based on GPU + CPU. The Canny algorithm is divided into two parts: Gauss filtering, gradient computations, non maximum suppression and double thresholding are processed by GPU. The fast method convert two-dimensional Gaussian filter to two separable convolutions to reduce the computation complexity. Then, multiple threads execute kernel in parallel to speed up the computation in the CUDA program. Finally, threads access shared memory instead of global memory to hide the latencies of global memory. In addition, FIFO is used to connect components in CPU. The simulation results show that the processing time of the 8-bit images with the resolution $1\ 024 \times 1\ 024$ is 122 ms, which is 5.39 times faster than CPU. Therefore, this method takes full advantage of the parallelism of GPU and the serial processing capability of CPU.

收稿日期:2015-12-21; **修订日期:**2016-03-03.

基金项目:国家自然科学基金(No. 61463009)

Supported by National Natural Science Foundation of China(No. 61463009)

* 通信联系人, E-mail: tangbin54@163.com

Key words: Canny; CUDA; GPU; acceleration

1 引 言

John F. Canny 在 1986 年提出最佳多级边缘检测 Canny 算子^[1],由于其检测效果较好,因此在边缘检测处理中应用广泛。CANNY 算子需要高斯滤波、梯度幅值和方向计算、非极大值抑制和双阈值处理,如果直接使用 CPU 检测分辨率高的图像边缘时,计算工作量大、消耗时间较长。考虑到图像中各个像素点的数据结构规则,运算程序相同,适合并行运算。GPU(Graphics Processing Unit)可以提供多个计算资源并行运算以加快程序的计算速度^[2],使用 GPU 加速处理的技术广泛应用在分子动力学模拟、医疗成像、空间建模和图像处理等密集计算领域,因此可以利用 GPU 进行加速处理 CANNY 算子。钮圣斌等在其文中使用 GPU 加速 canny 算子的计算^[3];Luo Y C 等则在英伟达的 GPU 上实现了 CANNY 算子的计算,相对使用 OPENCV 而言,最大加速比可达 3.403^[4];文献[3]和[4]均采用的是在 GPU 上完成 CANNY 算子整个处理过程,但并没有优化充分利用高斯函数可分离性的性质降低高斯滤波的计算量,而且在 GPU 中实现真边缘的连接,需要向 CPU 返回真边缘以完成线程的同步,传输时间大于连接时间。因此本文提出一种基于 GPU 和 CPU 的快速实现 Canny 算子的方法,利用 CUDA 编程在 GPU 上完成数值计算部分,在 CPU 上完成边缘的连接。

2 CANNY 算子

2.1 CANNY 算子的准则

Canny 算子满足最优检测器的 3 条准则^[5]。

(1)信噪比检测准则:要求尽量测出图像中所有的真实边缘点和抑制所有的伪边缘点,提供一个最大的信噪比。

(2)精确定位准则:要求检测到的边缘点位置与真实边缘点位置之间的距离差值最小。

(3)单边缘响应准则:要求降低单边缘多响应的概率,检测到的边缘点与真实边缘点具备一对一特征从而抑制伪边缘的多响应。

2.2 CANNY 算子的流程

使用 CANNY 算子检测图像边缘的流程分为以下 4 步:

(1)高斯滤波:待处理的图像可能在图像的获取和传输的过程中受到噪声污染,在边缘检测前需要使用高斯滤波器对图像滤波,高斯滤波的实质是线性低通滤波,主要用来抑制图像中的高频信息(噪声等)和保留图像的低频信息从而平滑图像。

(2)梯度幅值和方向计算:根据像素点的梯度幅值和方向检测图像的边缘。CANNY 算子利用一阶有限差分在 2×2 邻域中计算梯度和方向,像素点 $P(i, j)$ 的梯度幅值和方向计算过程如下所示:

X 方向偏导数:

$$G_x(i, j) = (P[i, j+1] - P[i, j] + P[i+1, j+1] - P[i+1, j]) / 2 \quad (1)$$

Y 方向偏导数:

$$G_y(i, j) = (P[i, j] - P[i+1, j] + P[i, j+1] - P[i+1, j+1]) / 2 \quad (2)$$

梯度幅值:

$$A(i, j) = \sqrt{G_x^2(i, j) + G_y^2(i, j)} \quad (3)$$

方向:

$$\theta(i, j) = \arctan \frac{G_y(i, j)}{G_x(i, j)} \quad (4)$$

(3)非最大值抑制:在第二步中获取全局梯度后还要进一步筛选局部真实边缘点。筛选方法是将梯度方向分为 0° 、 45° 、 90° 、 135° 等 4 个方向。像素点 $P(i, j)$ 的梯度幅值与其梯度方向上两个像素点的梯度幅值做比较,如果其值最大,则保留像素点 $P(i, j)$ 的梯度值,否则将其梯度幅值设置为零。

(4)双阈值处理和边缘连接:经过非极大值处理后的图像还需要进行双阈值处理确定真边缘和消除假边缘,使用高阈值 T_H 筛选出真边缘图像 $S(i, j)$;使用低阈值 T_L 删除图像中的假边缘像素点;而介于高阈值和低阈值的像素点则构建成弱边缘图像 $W(i, j)$ 。若图像 $W(i, j)$ 中的某个弱边缘像素其 8 邻域中存在任意一个真边缘像素,将该弱边缘像素补充到真边缘图像 $S(i, j)$ 中完成边缘的连接。为检测 $W(i, j)$ 的像素是否可以补

充到真边缘图像 $S(i, j)$ 中,需要使用递归函数遍历弱边缘图像 $S(i, j)$ 中每个像素。

3 CANNY 算法在 GPU 中的加速

3.1 优化高斯滤波算法

由于处理的图像为二维函数,平滑图像时采用的是二维高斯函数^[6],其表达式如下所示:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)} = G(x)G(y), \quad (5)$$

式中: x 和 y 分别表示的是邻域中某像素点与中心像素点在水平方向和垂直方向的距离。高斯滤波算法就是利用高斯核函数与原始图像做卷积运算^[7]。高斯滤波后中心像素点的值为中心像素点及其邻域像素点的加权平均值。高斯滤波具体过程如下所示:

$$I(i, j) = \sum_{s=-a}^a \sum_{t=-b}^b G(s, t) p(i+s, j+t), \quad (6)$$

式(6)中 $I(i, j)$ 为滤波后的中心像素点值, $p(i, j)$ 为中心像素点值, i 和 j 为中心像素点的坐标,由于绝大多数的图像处理都采用滑动窗口形式计算,通常可以先将各个距离的权系数计算出来作为模板使用,使用模板与原始图像做卷积完成高斯滤波。系数 a 和 b 与高斯模板大小相关, $2a+1$ 和 $2b+1$ 表示的都是模板的大小。

由于高斯滤波的加权系数具有关于旋转对称的特征,任意方向上的权系数关于中心点对称。根据高斯函数的二维空间表达式,二维空间的高斯滤波可以分解成水平方向的和垂直方向的两次独立的一维高斯滤波^[8]。其具体计算过程如下所示:

$$\text{temp}(i, j) = \sum_{s=-a}^a G(s) p(i+s, j), \quad (7)$$

$$I(i, j) = \sum_{s=-a}^a G(s) \cdot \text{temp}(i, j+s). \quad (8)$$

首先根据公式(7)计算出所有像素点在水平方向上的高斯滤波,然后再使用公式(8)对水平放上计算的结果做垂直上的高斯滤波。二维滤波分解为两次一维滤波后,除边界点外,每个像素点水平方向一维高斯滤波值 $\text{temp}(i, j)$ 被重复利用 $2a+1$ 次。原有计算复杂度 $O((2a+1) \times (2a+1) \times M \times N)$ ($M \times N$ 为图像分辨率)降低到 $O((2a+1) \times M \times N) + O((2a+1) \times M \times N)$, 计算复杂度下降。

3.2 使用多个线程加快计算速度

NVIDIA 公司的 GPU 属于 SIMD 体系结构, GPU 具有全局存储器、共享存储器和多个 SM (Streaming Multiprocessor),每个 SM 包含 8 个 SP (Stream Processor),SP 的实质是一个全流水线化的单发射微处理器,其包含两个逻辑运算部件和一个浮点数运算部件,每个 SP 执行一个线程 (Thread)。多个线程组成一个线程块 (Block),等价于每个 SM 执行一个线程块,因此 GPU 可以同时执行多个线程。CUDA (Compute Unified Device Architecture, 计算统一架构) 是 NVIDIA 公司在 2007 年提出的一种新开发环境,在 CUDA 中可以使用 C 语言完成多线程程序的编程。其异构编程模型如图 1 所示:一个完整的程序分为串行和并行两部分,串行部分通常在主机 (Host) 上运行,可并行部分则在设备 (Device) 中运行以充分利用 GPU 的并行性实现计算加速^[9]。

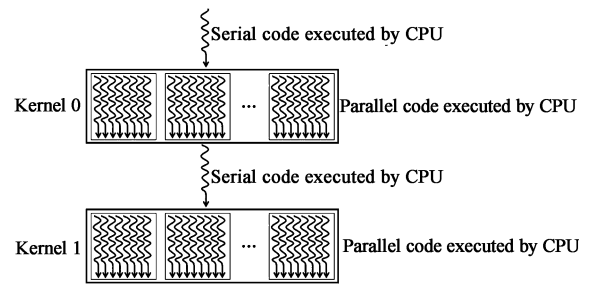


图 1 CUDA 编程模型

Fig. 1 CUDA programming model

本文在 GPU 中加速 CANNY 算子的高斯滤波、梯度幅值和方向计算、非极大值抑制和双阈值处理,处理的流程如图 2 所示:首先使用 `cudaMemcpy()` 函数将待处理图像像素矩阵值从主机 (Host) 传输到 GPU 的全局存储器,任务 1、任务 2、任务 3 分别是高斯滤波、梯度幅值和方向计算、非极大值抑制和双阈值处理。三个任务依序在 GPU 中并行执行,由于上个任务执行完后才能开启下个任务的执行,需要使用 CUDA 提供的 `syncthreads()` 函数实现任务之间的栅栏同步^[2]。最后通过 `cudaMemcpy()` 函数将 GPU 处理结果从全局存储器返回主机。在 GPU 中所有任务使用相同的 n^2 个线程块并行完成,每个线程块中包含 m^2 个线程,通常情况下所有线程的程序相同但处理像素点不同,线程与像素点属于一对一关系。

由于图像可以采用二维矩阵表示,因此每个像素点对应的线程 ID 号也采用二维坐标(Row,Col)表示,坐标值由以下公式获取:

$$\text{Row} = \text{blockIdx.y} \times m^2 + \text{threadIdx.y}, (9)$$

$$\text{Col} = \text{blockIdx.x} \times m^2 + \text{threadIdx.x}. (10)$$

上述表达式中 blockIdx.x 和 blockIdx.y 表示的是线程块的坐标,它们的乘积等于 n^2 。每个块中的线程可以用坐标 threadIdx.x 和 threadIdx.y 表示,它们的乘积等于 m^2 。

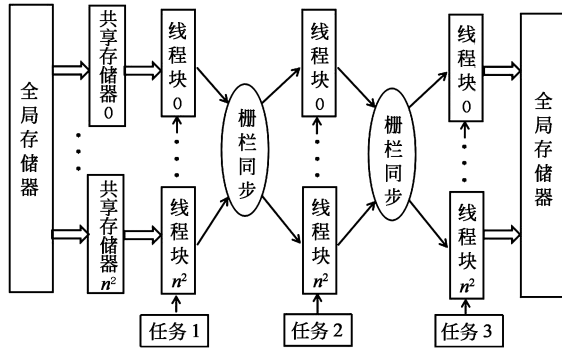


图2 CANNY算子加速流程

Fig. 2 Acceleration flow of Canny algorithm

3.3 使用共享存储器(shared memory)隐藏全局存储器(global memory)访存延迟

由于 SP 中没有使用缓冲技术,各个线程需要从全局存储器中读写像素值,但全局存储器的访存时间较长,此时会导致部分线程处于等待操作状态,因此降低了线程的利用效率。从 CANNY 算子处理流程可知:相邻像素点在做计算时,所需要的数据存在交集;同时上个任务执行完后的结果需要提供给下个任务使用。因此可以将像素值存储在访存时间较短的共享存储器中,减少线程对全局存储器的访问次数,从而隐藏存储器访存延迟。共享存储器的作用类似于 CPU 中的 CACHE。

基本块中线程的数量为 m^2 ,共享存储器可存储 m^2 个像素值。需要注意的是共享存储器只供基本块中的线程访问,不支持跨块访问。在 3 个任务处理过程中,基本块中边界点的计算均需要跨块访问,为防止跨块访问,本文将基本块扩展成扩展块。如果非极大值抑制的基本块大小为 $m \times m$,那么其扩展块大小为 $(m+2) \times (m+2)$;梯度幅值和方向计算的基本块大小则为 $(m+2) \times (m+$

2),其扩展块大小为 $(m+4) \times (m+4)$;在高斯滤波时,由于模板大小为 $a \times a$,其基本块大小为其 $(m+4) \times (m+4)$,其扩展块大小应为 $(m+2a+4) \times (m+2a+4)$ 。基本块与扩展块如下图所 3:基本块设置为 $m \times m$,扩展块大小设置为 $(m+2a+4) \times (m+2a+4)$ 。扩展块中基本块中的线程主要完成高斯滤波、梯度幅值和方向计算、非极大值抑制和双阈值处理,剩余部分的线程负责为边界像素点提供数据。采用共享存储器作为扩展块后,访问全局存储器的次数减少为 $(m+2a+4)^2 \times M \times N / m^2 + M \times N$,而不使用共享存储器时需要访问全局存储器次数为 $(2a+1)^2 \times M \times N + 12 \times M \times N$,因此该方法能显著减少访问全局存储器的次数。

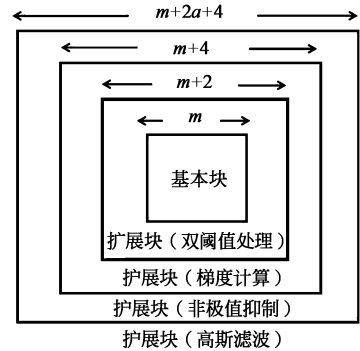


图3 共享存储器分配示意图

Fig. 3 Allocate shared memory

4 边缘连接在 CPU 中的实现

4.1 边缘连接

如果直接在 GPU 上完成边缘点的连接,由于 GPU 不支持块与块之间的通信^[4],在多个线程完成某个边缘点连接时,需要向 CPU 返回处理结果完成线程与线程之间的同步,导致同步消耗的时间大于边缘连接的时间;或是使用原子操作同步,但是由于边缘连接像素位置可能不是列连续,随机写全局存储存在大的延迟,不适合在 GPU 中做边缘连接处理。因此本系统采用队列 FIFO(First In First Output)在 CPU 中完成强边缘图像 $S(i,j)$ 中所有像素边缘的连接^[10]。在强边缘图像 $S(i,j)$ 中取出第一个像素 P 压入队列 Queue,同时以 P 点为起点,在弱边缘图像 $W(i,j)$ 中搜索 P 点的 8 邻域。如果存在弱边缘像素,将它们补充到强边缘图像 $S(i,j)$ 中,同时也压入

到队列的尾部。然后从队列的首部取出已经归类于强边缘的弱若边缘 Q 点。再以 Q 点为起点,继续使在弱边缘图像 $W(i,j)$ 中搜索 Q 点的 8 邻域像素;如果不周围不存在弱边缘像素,则直接从队

列中取出其他像素,重复上述过程直至队列为空。然后将 $S(i,j)$ 中的第二个像素压入队列,重复上述过程直至搜索完 $S(i,j)$ 的每个像素。图 4 给出了起点 $P(3,2)$ 的边缘连接过程。

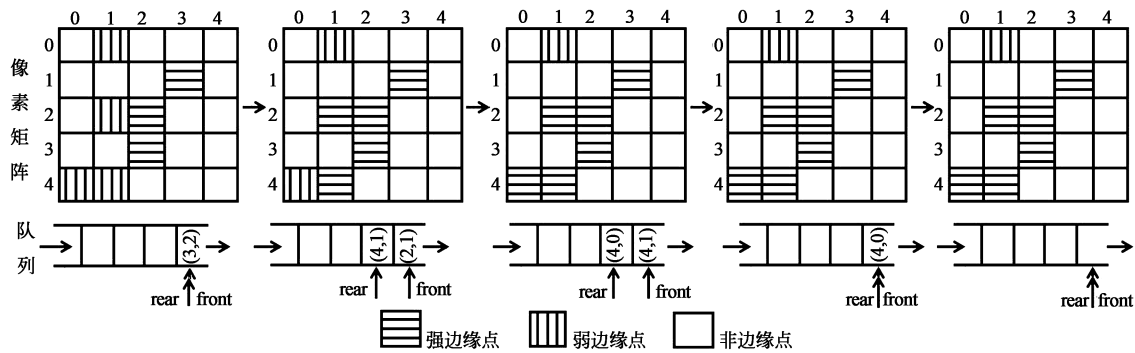


图 4 边缘连接示例图

Fig. 4 Example of connecting edges

边缘连接伪代码如下所示:

```
For P  $S(i,j)$ {
    将 P 点坐标存入 Queue;
    Queue 首地址指向 P;
    While(Queue 不为空){
        从 Queue 队首获取强边缘像素点 Q
        坐标 X 和 Y;
        Queue 首地址指向下一个强边缘
        像素;
```

```
For Q (Q 点的 8 邻域像素) //在弱边缘 W
(i,j)中检测 Q 的 8 邻域像素;
If  $Q > T_L$ {
     $Q \leq 0$ ;
    Q 点存入强边缘图像  $S(i,j)$ ;
    将 Q 点坐标存入 Queue;
```

Queue 尾地址指向 Q 点;

5 实验结果及分析

本文最后对上述所提出的方法进行实验测试,测试图片选用 Lena,图片格式是 BMP,BMP 格式较为简单,其文件由档头、信息头、调色板和位图数据组成。位图数据没有压缩原始数据。存储的是原始 RGB 像素值,使用 C 语言程序直接读出各个像素点的 RGB 值,然后转换为 8 位的灰度图像。本文所使用的实验环境如表 1 所示。

表 1 实验环境

Tab. 1 Experimental environment

实验设备	主频	开发环境	语言
Intel core i5(4 核)	1.7 GHz	Visualstudio 2012	C 语言
NVIDIA GEFORCE 820M	1.25 GHz	CUDA 6.5 Visualstudio 2012	C 语言

NVIDIA GEFORCE 820M 中存在 2 个 SM,全局存储器容量为 1 GB,每个线程块的共享存储器容量为 48 KB。由于需要计算梯度的幅值和方向,需要在每个线程块中开辟 3 个扩展块,一个用于保存梯度方向,其余两个用于存储上

个任务的处理结果和本次任务处理结果。因此每个扩展块的容量不能超过 16 KB。本文基本块的大小选择为 16×16 ,包含 16×16 个线程。线程块数量为 $M \times N / 256$ 。最后测试的数据如表 2 所示。

表 2 实验结果

Tab.2 Result of system simulation

图像 分辨率	高斯模板 大小	完全 CPU 运算时间/ms	GPU+GPU 混合运算时间/ms	加速比
256×256	3×3	10	5	2.00
512×512	3×3	41	22	1.86
1 024×1 024	3×3	149	82	1.81
512×512	9×9	133	29	4.56
1 024×1 024	9×9	480	113	4.24
1 024×1 024	11×11	658	122	5.39
2 048×2 048	11×11	2 831	734	3.85

从上表数据可以看出:图像分辨率越高,高斯模板尺寸越大,加速的效果越明显。

6 结 论

本文通过使用 CUDA 编程实现在 GPU 和 CPU 中实现了 Canny 算子的快速计算。通过对 Canny 算子中高斯滤波的优化和使用共享存储器

减少访问全局存储器,在 GPU 实现高斯滤波、梯度幅值和方向计算、非极大值抑制和双阈值处理,同时在 CPU 使用队列 FIFO 实现边缘连接。相比在 CPU 上完成 Canny 算子,最大加速比可达 5.39 倍。本设计充分利用了 GPU 的并行计算能力和 CPU 的串行处理能力,具有较好的灵活性和较快的处理速度,对图像的快速处理具有一定的参考价值。

参 考 文 献:

[1] CANNY J. A computational approach to edge detection [J]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986, PAMI-8(6): 679-698.

[2] NVIDIA. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide: Version 3.2* [M]. Santa Clara: NVIDIA Corporation, 2010.

[3] LUO Y C, DURAISWAMI R. Canny edge detection on NVIDIA CUDA [C]. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, Anchorage, AK: IEEE, 2008: 1-8.

[4] 钮圣斌,王盛,杨晶晶,等.完全基于边缘信息的快速图像分割算法[J].*计算机辅助设计与图形学学报*,2012,24(11):1410-1419.

NIU S X, WANG S, YANG J J, *et al.* A fast image segmentation algorithm fully based on edge information [J]. *Journal of Computer-Aided Design & Computer Graphics*, 2012, 24(11): 1410-1419. (in Chinese)

[5] 石柱名,魏庆涛,孟繁盛.基于 Canny 算子的图像边缘检测算法[J].*现代电子技术*,2015,38(12):92-93,97.

SHI G M, WEI Q T, MENG F S. Image edge detection algorithm based on Canny operator [J]. *Modern Electronics Technique*, 2015, 38(12): 92-93, 97. (in Chinese)

[6] 刘久文,潘峰,李军.结合图像边缘检测和最小误差替换的隐写方案[J].*液晶与显示*,2015,30(1):151-156.

LIU J W, PAN F, LI J. steganography based on edge detection and minimum error replacement [J]. *Chinese Journal of Liquid Crystals and Displays*, 2015, 30(1): 151-156. (in Chinese)

[7] 许佳佳.结合 Harris 与 SIFT 算子的图像快速配准算法[J].*中国光学*,2015,8(4):574-581.

XU J J. Fast image registration method based on Harris and SIFT algorithm [J]. *Chinese Optics*, 2015, 8(4): 574-581. (in Chinese)

[8] 丁怡心,廖勇毅.高斯模糊算法优化及实现[J].*现代计算机*,2010(8):76-77,100.

DING Y X, LIAO Y Y. Optimization and Implementation of Gaussian blur algorithm [J]. *Modern Computer*, 2010

(8): 76-77, 100. (in Chinese)

[9] 王新华,王晓坤. 十亿像素瞬态成像系统实时图像拼接[J]. 中国光学,2015,8(5):785-793.
WANG X H, WANG X K. Real time image mosaic of the transient gigapixel imaging system [J]. *Chinese Optics*, 2015, 8(5): 785-793. (in Chinese)

[10] 刘谷,安虹,李小强,等. 图广度优先搜索算法面向图形处理器的优化方法研究[J]. 小型微型计算机系统,2014,35(5):1074-1079.
LIU G, AN H, LI X Q, *et al.* Study on optimizing techniques of breadth-first search algorithm on graphic processing unit [J]. *Journal of Chinese Computer Systems*, 2014, 35(5): 1074-1079.

作者简介:唐斌(1981—),男,湖南邵阳人,硕士,讲师,主要从事电子系统设计自动化方面、图像处理研究。E-mail:tangbin54@163.com

龙文(1977—),男,湖南邵阳人,博士,教授,主要从事进化计算研究。E-mail:382635426@qq.com