模糊连接图像分割 CUDA 并行算法的改进

李 栋, 黄绍辉*, 黄晓阳, 王连生, 王博亮

(厦门大学信息科学与技术学院计算机科学系 厦门 361005)
(hsh@xmu.edu.cn)

摘 要: 已有的模糊连接并行算法 CUDA-kFOE 未考虑线程块边缘点同时更新所引发的竞争问题, 导致计算结果出现少量误差. 由于医学图像处理对精度的要求很高, 为了解决边缘点计算误差的问题, 基于CUDA-kFOE提出一种修正迭代算法. 首先分析了 CUDA-kFOE 算法在线程块边缘产生竞争的原因; 然后讨论了边缘点亲和力的所有可能的传递路径, 以及由此造成的出错情况; 最后提出二次迭代修正算法, 将第一次迭代得到的所有边缘点转入第二次的修正迭代步骤, 从而修正第一次迭代中错误的亲和力值. 采用 3 组不同规格的 CT 序列对肝脏血管进行分割实验, 并选用 3 个不同的种子点进行算法验证, 结果表明, 文中算法的计算结果与串行版本一致, 解决了 CUDA-kFOE 算法的计算误差问题.

关键词: 肝脏血管分割; 模糊连接; CUDA; 块边缘点竞争
中图法分类号: TP391.9

An Improved Fuzzy Connected Image Segmentation Method Base on CUDA

Li Dong, Huang Shaohui*, Huang Xiaoyang, Wang Liansheng, and Wang Boliang

(*Computer Science Department, School of Information Science and Engineering, Xiamen University, Xiamen* 361005)

Abstract: A paralleled CUDA version of kFOE(CUDA-kFOE)was proposed to segment medical images. CUDA-kFOE achieves fast segmentation when processing large image datasets. However, it cannot precisely handle the competition of edge points when update operations happen by multiple threads simultaneously, thus an iterative correction method to improve CUDA-kFOE was proposed. By analyzing all the pathways of marginal voxels affinity and their consequently caused results, a two iteration correction scheme is employed to achieve the accurate calculation. In these two iterations, the resulted marginal voxels from the first iteration are used as the correction input of the second iteration, therefore, the values of affinity are corrected in the second iteration. Experiments are conducted on three CT image sequences of liver vessels with small, medium, and large size. By choosing three different seed points, final results are not only comparable to the sequential implementation of fuzzy connected image segmentation algorithm on CPU, but achieve more precise calculation compared with CUDA-kFOE.

Key words: liver vessel segmentation; fuzzy connected method; CUDA; block edge points competition

收稿日期: 2015-02-05; 修回日期: 2015-05-21. 基金项目: 国家自然科学基金(61001144, 61102137, 61301010, 61327001).
李 栋(1990—), 男, 硕士研究生, 主要研究方向为医学图像处理、并行计算; 黄绍辉(1976—), 男, 博士, 副教授, 硕士生导师, 论文通讯作者, 主要研究方向为医学图像处理; 黄晓阳(1978—), 男, 博士, 讲师, 主要研究方向为医学图像处理; 王连生(1979—), 男, 博士, 讲师, CCF 会员, 主要研究方向为心电功能成像研究与稀疏性算法; 王博亮(1945—), 男, 教授, 博士生导师, 主要研究方向为医学图像处理.

1 模糊连接

模糊连接分割在 1996 年由 Udupa 等[1]首次提出，该算法通过比较种子点与目标区域和背景区域的连通度大小来分割目标和背景.

1.1 基本概念及定义

定义 X 是一个任意的参考集，一个模糊子集 A 是 X 中的有序对

$$A=\{X,\mu_A(X)|x\in X\};$$

其中 $\mu_A: X\to[0,1]$. μ_A 是 X 中 A 的成员函数.

一个在 X 中的二元模糊关系 ρ 是 $X\times X$ 的模糊子集，ρ 称之为在 X 中的相似关系

$$\rho=\{(x,y),\mu_\rho(x,y)|x,y\in X\};$$

其中，$\mu_\rho: X\times X\to[0,1]$[2].

ρ 具有自反性. 如果 $\forall x\in X$，$\mu_\rho(x,x)=1$;

ρ 具有对称性. 如果

$\forall x,y\in X$，$\mu_\rho(x,y)=\mu_\rho(y,x)$;

ρ 具有传递性. 如果

$$\forall x,z\in X,\mu_\rho(x,z)=\max_{y\in X}\left[\min\left(\mu_\rho(x,y),\mu_\rho(y,z)\right)\right].$$

1.2 基本概念及定义

由于容积效应的存在，CT 影像边界较为模糊，因此常规的区域生长算法难以取得较好的效果，模糊连接算法可视为区域生长算法的改进. 在 CT 影像中，各个组织具有渐变的组成. 如肝内血管的 CT 值从管道中心到边界逐渐变小，直到过渡到肝实质的部分. 这个渐变导致血管的边界很模糊，难以通过阈值化将其与肝实质直接分割. 从模糊数学的角度则比较容易描述边界附近的点的归属，因为可以通过模糊函数的取值来刻画一个点分别属于血管和肝实质的隶属度；此外，相同的 CT 值未必属于同一个组织，因此还需要能够刻画点和点之间连通性的量. 综上，一个能用于图像分割的模糊关系需要量化两方面的信息：1) 与种子点的相似程度；2) 与种子点的连通程度. 下面给出符合这两点要求的模糊关系定义.

令 $I=(\boldsymbol{C},k)$ 是一个场景，其中 $\boldsymbol{C}$ 表示 CT 图像上所有点(体素)的集合，它通常是一个三维矩阵；k 表示密度函数，它能对 $\boldsymbol{C}$ 中所有体素进行计算，并将其映射到指定的取值区间[l, h]. 如果 k 是一种模糊关系，并且 k 是自反的、对称的，则称 k 是在 $\boldsymbol{C}$ 上的模糊相似关系.

为量化 $\boldsymbol{C}$ 中任意 2 个体素 c 和 d 之间的连接强度，定义 k 的隶属函数 $\mu_k(c,d)$，其值域在[0, 1]. $\mu_k(c,d)$ 又称为 c 和 d 的亲和力. 亲和力越大，表明 c 和 d 同属于一个组织的可能性越大. $\mu_k(c,d)$ 的具体形式为

$$\mu_k(c,d)=\mu_\alpha(c,d)\sqrt{g_1(f(c),f(d))g_2(f(c),f(d))} \quad (1)$$

式(1)中，$\mu_\alpha(c,d)$ 是表征连通程度的函数，通常取 6-邻域，

$$\mu_\alpha(c,d)=\begin{cases}1, & \text{if } \sqrt{\sum_i(c_i-d_i)^2}\leqslant 1\\ 0, & \text{otherwise.}\end{cases};$$

g_1，g_2 是 $\frac{f(c)+f(d)}{2}$ 和 $\frac{|f(c)-f(d)|}{2}$ 的高斯函数，g_1 的均值和方差通过种子点附近目标物体的计算得到，在迭代过程中是一个常量；g_2 是一个 0 均值高斯函数[3]. 实际计算过程中，算法将从种子点开始，利用式(1)计算 6-邻域的体素的连接强度，并按照广度优先的原则不断将种子点的影响外扩，得到 $\boldsymbol{C}$ 中所有体素与种子点的连接强度图(称为模糊场景)，最后将模糊场景二值化，即可得到分割结果.

1.3 基于模糊连接的图像分割算法

模糊连接的图像分割算法[1]简称 kFOE，它使用动态规划方法的 Dijkstra 算法来寻找从目标点 o 到各个像素点的最佳路径. kFOE 算法步骤如下:

输入. $I=(\boldsymbol{C},f)$，任意的 $o\in\boldsymbol{C}$，模糊度 k.

输出. 矩阵 $\boldsymbol{C}$ 中的目标区域 O.

辅助数据. 一个代表连接场景的三维矩阵 $\boldsymbol{C}_{Ko}=(\boldsymbol{C},f_{Ko})$ 和一个队列包含待处理体素 Q.

begin:

Step1. 设置所有 $\boldsymbol{C}_{Ko}$ 中除了目标点 o 以外的体素为 0, o 点体素为 1.

Step2. 将目标点 o 加入队列 Q.

Step3. 当队列 Q 不为空时:

Step3.1. 移出一个在 Q 内的体素 c，找到

$$f_{Ko}(c)_{\max}=\max_{d\in\boldsymbol{C}_{Ko}}\left[\min\left(f_o(d),\mu_k(c,d)\right)\right];$$

Step3.2. 若 $f_{Ko}(c)_{\max}>f_{Ko}$，设置

$$f_{Ko}(c)=f_{Ko}(c)_{\max}.$$

end.

2 CUDA 架构及执行模型

CUDA 的基本思想是对于不同 CUDA 设备，

使所有计算线程能够在逻辑上并发的运行. 实际上，根据设备的不同，任务又被细化分以 Block 为单位的线程块，并被 GPU 自动分配到 SM(stream multiprocessor)处理器上进行并行处理. 图 1 中说明了 CUDA 将分块从软件层次映射到硬件次上[4]，任意一个 SM 处理器中的 Block 是独立并行运行的，也就是不允许不同的线程块之间执行栅栏同步[5]，不同的线程块不用相互等待，便能以任何顺序执行.

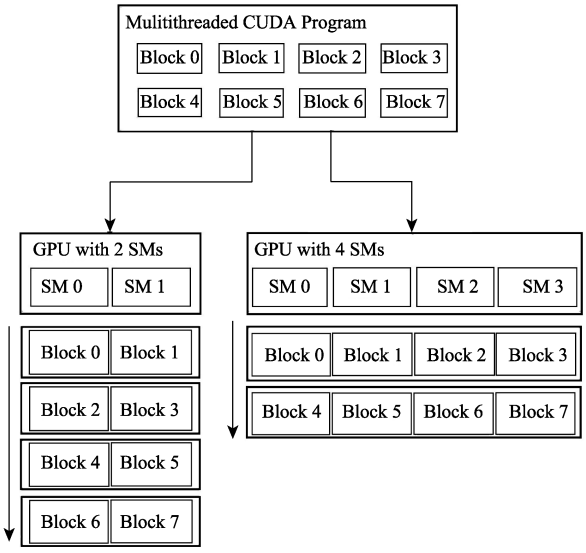


图 1　自动可测性

3　基于 CUDA 的并行模糊连接算法及改进

3.1　CUDA-kFOE 算法

由 Zhuge 等[6-7]提出的 CUDA-kFOE 算法已经将模糊连接并行化，文中将其分为 2 步:

1) 亲和力计算. 亲和力计算是对体素对 (c,d) 之间的计算，主要将式(1)代入得到，并将计算得到的亲和力 $\mu_k(c,d)$ 保存在 GPU 特定的内存中.

2) 更新模糊连接. 对体素的模糊连接算法其本质是单源 Dijkstra 算法，算法中采用 Harish 等[8]提出的 CUDA_SSSP 算法，在 CUDA 计算能力大于 1.1 的设备中，使用原子操作解决了多线程访问同一地址冲突的问题，基本上实现了其并行化.

CUDA-kFOE 算法步骤如下:

输入. 图像 I，阈值 k, I 中目标 O 的种子点.

输出. 目标区域 O 保存的图像 $\boldsymbol{C}_{Ko}$.

辅助数据结构. 2 个用于判定是否收敛的数组 C_1，C_2.

begin:

Step1. 设置图像 $\boldsymbol{C}_{Ko}$ 和收敛数组 C_1 所有的像素点为 0，除了目标 O 的种子点为 1.

Step2. 运行 AFFINITY-KERNEL 计算图像 I 中所有点的亲和度 μ_k.

Step3. 当数组 C_1 不全为 0 时:

Step3.1. 把 C_2 的所有像素点设置为 0.

Step3.2. 运行 TRACKING-KERNEL 更新模糊关系.

Step3.3. 将 C_2 拷贝给 C_1.

end.

AFFINITY-KERNEL

begin:

Step1. 计算每个线程对应的索引 t.

Step2. 对每个线程 t 所对应的像素点 d 的 6-邻域 c，通过式(1)计算亲和度 $\mu_k(c,d)$，将亲和度 $\mu_k(c,d)$ 写入 GPU 内存中.

end.

TRACKING-KERNEL

begin:

Step1. 计算每个线程对应的索引 t.

Step2. 对于每个像素点 c: 若像素点 c 坐标对应的 C_1 为 1: 对像素点 e，若满足亲和度 $\mu_k(c,e)>0$，

取 $f_{\min}=\min\{f(c),\mu_k(c,e)\}$.

若 $f_{\min}>f(e)$, then $f(e)=f_{\min}$，$C_2(e)=1$.

end.

3.2　CUDA-kFOE 执行模型及不足

由 CUDA 程序的执行模型可以看到，本文将相应的线程号(*threadIdx*)一一对应图像中的体素，线程块号(*blockidx*)对应由图像经过网格分割图像后得到的块，如图 2 所示.

但由于块与块之间缺少通信，导致 2 个相邻块的边缘处会发生不可预见的干扰，从而造成产生结果上的误差. 如图 3 所示，其中红色表示 C_1 的值为 TRUE 的点，意味着要进行计算并更新的点. 在计算能力为 2.1 的 CUDA 设备中，一个 SM 最多可以分配 1 536 个线程(即可以分配 6 个包含 256 个线程的线程块，或者 3 个包含 512 个线程的线程块等情况). 根据最优化分配 SM 内寄存器，并通过 NVIDIA[9]计算得出线程块内线程数为 256 个时效率最佳，在三维情况下，对应块划分为 16×16×1. 为了方便说明，下面采用二维的情况来阐述其块与块之间相互影响的原因. 如果采用的是计算能力 2.1 的 GPU，将线程块分为 3×3，即一个 SM 内有 8 个线程块并行参与运算. 当 SM 处理到(0, 1), (1, 1),…,(7, 1) 8 个块时，由于块(1, 1)与块(2, 1)之间边缘产生同时要更新的点，此时不同线程块的不同线程计算模糊度会产生竞争现象，从而导致出现不正确的结果.

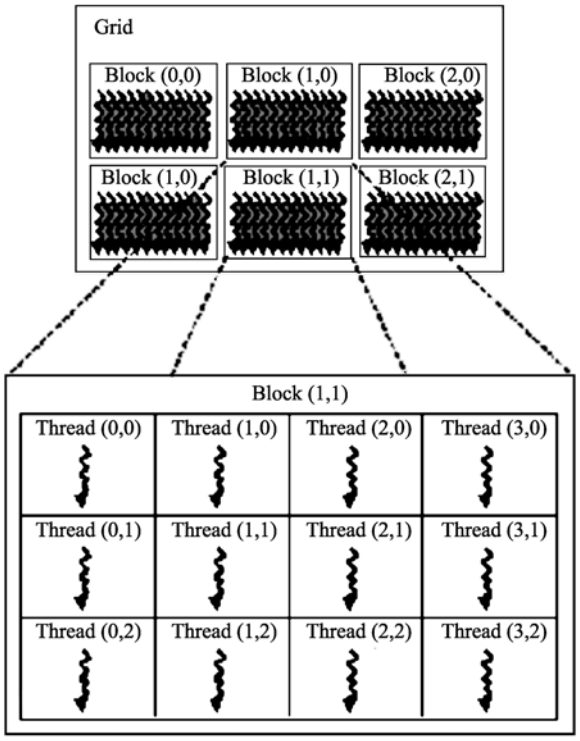


图 2 线程块的网格组织形式

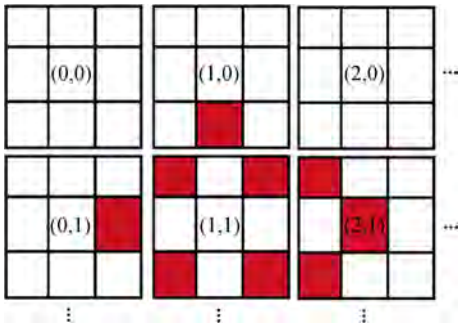


图 3 块边缘之间处理情况

3.3 改进 CUDA-kFOE 算法

CUDA-kFOE 算法有 2 个不足之处：一是在第 1 步计算模糊度并将其保存在 GPU 显存中，需要大量的额外空间，如果在显存容量有限的显卡中就很难执行；二是由于块之间边缘处的相互干扰，造成在块边缘处的计算出现误差．为了解决误差问题，需要分析误差来源．考虑到算法起点是单一种子点，并且以广度优先的顺序计算模糊场景，因此计算过程可以看成是以种子点为根的树状结构的生成过程．如果以红色表示需要向外传播的亲和力值，则当红色传递到块边缘后，因为需要跨块传播亲和力值就会引发竞争的问题．由于在传播过程中种子点是呈树状向外传播的，因此块与块之间的红色不会出现回路，否则与树状传播相矛盾．下面考虑竞争最激烈，也就是最坏的情况．如图 4 所示，红色表示需更新其邻居的点，蓝色表示被更新的点；像素点 1, (2, 4), 3, 5 分别位于不同的线程块中，其中像素点 1, 2, 3 为 $C_1(c)$ 等于 1 的点，像素点 4, 5 为被更新的邻域点．像素点 5 受到像素点 2, 3 更新时，由于线程块的运行是并行无序的，在判断 $f_{\min} > f(e)$ 时，像素点影响的情况根据线程块的完成顺序有以下 6 种情况:

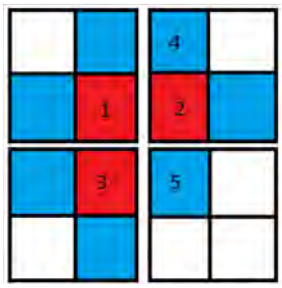


图 4 边缘像素点处理情况

1) $2\rightarrow5$，$3\rightarrow5$；
2) $3\rightarrow5$，$2\rightarrow5$；
3) $1\rightarrow3$，$1\rightarrow2$，$3\rightarrow5$，$2\rightarrow5$；
4) $1\rightarrow3$，$1\rightarrow2$，$2\rightarrow5$，$3\rightarrow5$；
5) $2\rightarrow1$，$2\rightarrow5$，$1\rightarrow3$，$3\rightarrow5$；
6) $3\rightarrow1$，$3\rightarrow5$，$1\rightarrow2$，$2\rightarrow5$．

情况 1 和情况 2 的先后顺序并不会影响其结果，因为更新像素点 5 时只会选取最大的亲和度值，所以不会因为线程块不同步而产生误差；情况 3 和情况 4 中，如果像素点 1 对像素点 2, 3 不产生数值上的影响，则效果等同于情况 1 和情况 2，同样不影响结果；如果像素点 1 对像素点 2, 3 的其中 1 个或 2 个产生数值上的影响，则像素点 5 的值可能就受到像素点 2, 3 值更改的影响．此时，若先执行 $2\rightarrow5$，$3\rightarrow5$ 或 $3\rightarrow5$，$2\rightarrow5$ 的操作，像素点 1 的影响将不能传递到像素点 5，继而像素点 5 就不能得到正确的值．如果在此基础上做一次修正迭代，受像素点 1 影响的像素点 2, 3 将能对错误值像素点 5 进行修正，因此二次迭代可以解决情况 3 和情况 4；情况 5 和情况 6 中，像素点的传播路径跨越 3 个线程块，进而对像素点 5 产生影响，参照情况 3 和情况 4 的解决思路，可以采用 3 次修正迭代进行消除误差．

通过以上分析可以得出，在理论上需要进行 3 次修正迭代，才能完全消除线程块边缘点竞争造成的影响．不过经过实验得出，2 次迭代足以消除这种线程块间传递带来的误差．因此，本文的算法将 CUDA-kFOE 中的两步骤合并到一步，同时也减少了在迭代运算中主机端与设备端交换数据的时间，算法步骤如下:

输入. 图像 I, 阈值 k, I 中目标 O 的种子点.

输出. 目标区域 O 保存的图像 $\boldsymbol{C}_{Ko}$.

辅助数据结构. 2 个用于判定是否收敛的数组 C_1，C_2.

begin:

Step1. 设置图像 C_{Ko} 和收敛数组 C_1 所有的像素点为 0, 除了目标 O 的种子点为 1.

Step2. 当 C_1 不全为 0 时:

Step2.1. 把 C_2 的所有像素点设置为 0.

Step2.2. 运行 PARALLEL-kFOE-KERNEL.

Step2.3. 将 C_2 的值拷贝到 C_1.

end.

PARALLEL-kFOE-KERNEL

begin:

Step1. 对于 $C_1(c)$ 为 1 的点.

Step2. 对像素点 c 的 6-邻域:

Step2.1. 计算 $\mu_k = FuzzyAffinity(e,c)$.

Step2.2. 取 $f_{\min} = \min(O(c), \mu_k)$.

Step2.3. 若 $f_{\min} > O(e)$ then

Step2.3.1. $f(e) = f_{\min}$，$C_2(e) = 1$，同步不同方向线程.

Step2.3.2. 所有线程同步，进入修正迭代，重复计算 $C_1(c) = 1$ 且位于线程块边缘的点.

end.

本文提出的改进的 CUDA-kFOE 算法是一个迭代算法, 第 1 次迭代时, 只有一个种子点进行运算, 计算种子点 6-邻域的亲和力, 并更新邻域点的亲和力值. 随着迭代次数的增加, 越来越多的点进行运算, 直到所有进程都不参与迭代更新时, 运算结束. 在 PARALLEL-kFOE-KERNEL 的 Step6 中, 因为同一个体素会被多个线程同时访问, 为了避免由此而产生的冲突, 采用原子运算函数(*atomicExch*), 这样线程会有序的进行访问. 改进的 KERNEL 函数使用了 2 次修正迭代, 对第 1 次迭代中线程块边缘处部分未计算正确的体素, 将在修正迭代时进入运算并修正其亲和力值, 解决了亲和力跨块传递的问题.

4　实验对比

本文实验环境为 I5-3470, GT610(计算能力 2.1, 显存容量 1 GB GDDR3, 单精度浮点运算能力约为 133 G Flops); 运行环境为 Windows 7, Visual Studio 2010, CUDA Version 6.0. 此外, 采用 3 组 CT 数据(大小分别为 512×512×155, 512×512×215, 512×512×175)进行比较, 横向比较同一套 CT 数据不同点, 对比算法为 CUDA-kFOE. 比较标准为 CPU 版本的结果, 实验数据均采用 float 类型数据. 每个种子点运算 3 次取平均值. 实验结果由表 1~3 所示.

计算完模糊场景之后, 还需要一个阈值将模糊场景进行二值化处理, 才能生成最后的处理结果. 图 5~7 所示为 3 套测试数据的分割结果.

表 1　数据 1 结果对比

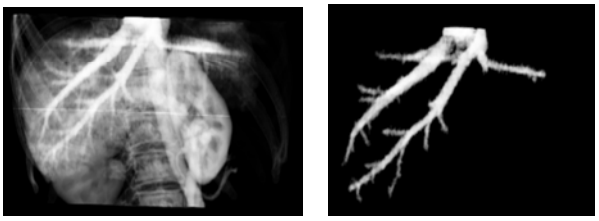
种子点	串行算法运行时间/s	CUDA-kFOE 算法			本文算法		
		运行时间/s	加速比	误差点数	运行时间/s	加速比	误差点数
(197, 258, 123)	281.6	63.8	4.41	762	72.3	3.89	0
(163, 187, 86)	283.8	60.6	4.68	616	71.0	4.00	0
(196, 188, 110)	247.9	56.2	4.41	956	66.0	3.76	0

表 2　数据 2 结果对比

种子点	串行算法运行时间/s	CUDA-kFOE 算法			本文算法		
		运行时间/s	加速比	误差点数	运行时间/s	加速比	误差点数
(189,244, 180)	245.5	73.3	3.47	722	81.8	3.11	0.7
(144, 239, 147)	340.1	60.6	5.61	1147	84.2	4.04	0.7
(107, 223, 117)	386.7	73.4	5.27	526	86.8	4.46	0

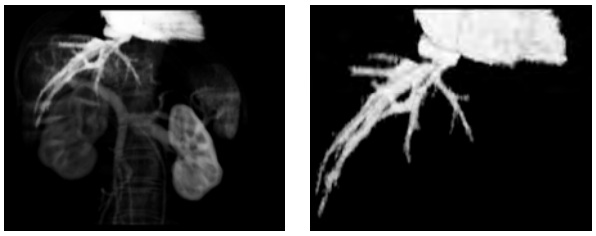
表 3　数据 3 结果对比

种子点	串行算法运行时间/s	CUDA-kFOE 算法			本文算法		
		运行时间/s	加速比	误差点数	运行时间/s	加速比	误差点数
(183, 279, 76)	288.8	63.9	4.52	3560	74.8	3.86	2
(167, 257, 107)	267.4	63.5	4.21	1147	74.0	3.61	0
(158, 247, 148)	230.5	61.9	3.72	526	70.8	3.26	0



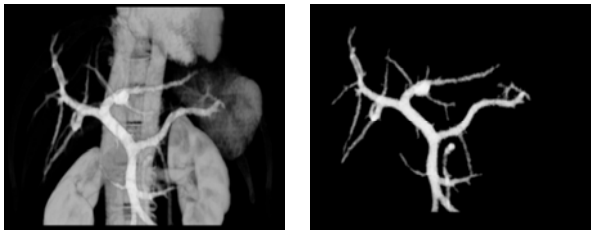
a. 未经过阈值处理 b. 经过阈值 0.5 的处理

图 5 数据 1 测试结果



a. 未经过阈值处理 b. 经过阈值 0.5 的处理

图 6 数据 2 测试结果



a. 未经过阈值处理 b. 经过阈值 0.574 的处理

图 7 数据 3 测试结果

5 结 语

经过实验比对，经过迭代修正后的实验数据结果与CPU串行版本的模糊连接结果几乎完全相同. 虽然在速度上对比未迭代修正的原 CUDA-kFOE 算法略微落后，但是仍然极大地提升了原 CPU 算法的速度. 在医学图像处理中，精度比速度更为重要，因此本文算法更适合作为并行化版本的模糊连接算法. 此外，在模糊连接算法中，最后的分割结果与初始的种子点位置以及二值化处理的阈值密切相关. 目前，这 2 个参数都是由手工调整的，今后将进一步探索种子点和阈值的自动获取方法，完成算法的全自动处理.

参考文献(References):

[1] Udupa J K, Samarasekera S. Fuzzy connectedness and object definition: theory, algorithms, and applications in image segmentation[J]. Graphical models and image processing, 1996, 58(3): 246-261

[2] Udupa J K, Saha P K. Fuzzy connectedness and image segmentation[J]. Proceedings of the IEEE, 2003, 91(10): 1649-1669

[3] Saha P K, Udupa J K, Odhner D. Scale-based fuzzy connected image segmentation: theory, algorithms, and validation[J]. Computer Vision and Image Understanding, 2000, 77(2): 145-174

[4] CUDA C programming guide version 6.0[OL]. [2015-02-05]. http://docs.nvidia.com/cuda/cuda-c-programming-guide

[5] Kirk D, Hwu W M. Programming massively parallel processors: a hands-on approach[M]. 2nd ed. Boston: Newnes, 2013: 59-93

[6] Zhuge Y, Cao Y, Miller R W. GPU accelerated fuzzy connected image segmentation by using CUDA[C] //Proceedings of Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2009. Minneapolis: Los Alamitos: IEEE Computer Society Press, 2009: 6341-6344

[7] Zhuge Y, Cao Y, Udupa J K, *et al*. Parallel fuzzy connected image segmentation on GPU[J]. Medical Physics, 2011, 38(7): 4365-4371

[8] Harish P, Narayanan P J. Accelerating large graph algorithms on the GPU using CUDA[M] //Lecture Notes in Computer Science. Heidelberg: Springer, 2007, 4873: 197-208

[9] NAIDIA CUDA occupancy calculator[OL]. [2015-02-05]. developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls