Journal of Computer Applications
计算机应用,2014,34(11):3121-3125
ISSN 1001-9081
CODEN JYIIDU
2014-11-10
http://www.joca.cn

文章编号:1001-9081(2014)11-3121-05
doi:10.11772/j.issn.1001-9081.2014.11.3121

基于 Chan-Vese 模型的面向多核 CPU 和 GPU 的人脸轮廓提取并行算法

王丽娜*,史晓华
(北京航空航天大学 计算机学院,北京 100191)
(*通信作者电子邮箱 binglina.wang@163.com)

摘　要:针对人脸轮廓提取中 Chan-Vese 模型计算量大、分割速度缓慢等问题,采用开放计算语言(OpenCL)并行编程模型,提出了一种基于图形处理器(GPU)和多核 CPU 加速的并行算法。该算法首先将模型的框架进行重构,消除模型中的数据依赖关系;然后,利用开放计算语言对算法进行并行化以及相应的优化。实验结果表明,与单线程算法相比,在 NVIDIA GTX660 和 AMD FX-8530 下达到了较高的加速比。

关键词:Chan-Vese 模型;并行;开放计算语言;人脸轮廓提取
中图分类号:TP311.1　　**文献标志码**:A

Parallel face contour extraction algorithm based on Chan-Vese model on multi-core CPU and GPU

WANG Lina*, SHI Xiaohua
(*School of Computer Science and Engineering, Beihang University, Beijing* 100191, *China*)

Abstract: Aiming at the problem that Chan-Vese model of face contour extraction has a large amount of calculation and slow segmentation, an parallel algorithm was proposed based on Graphic Processing Unit (GPU) and multi-core CPU with Open Computing Language (OpenCL) parallel programming model. Firstly, by reconstructing the computing architecture, the algorithm could eliminate data dependence of the model; secondly, using OpenCL to parallel and optimize the algorithm. The result shows that, compared with the single-thread method, the proposed algorithm gets a higher average acceleration ratios under NVIDIA GTX660 and AMD FX-8530.

Key words: Chan-Vese model; parallel; Open Computing Language (OpenCL); face contour extraction

0　引言

人脸识别是模式识别、图像处理、计算机视觉、神经网络等领域研究的热点,有十分广泛的应用前景。作为人脸特征的重要组成部分,人脸轮廓的提取是人脸特征检测和人脸识别等人脸图像分析的重要前提[1]。

相对于其他轮廓提取的分割模型,Chan-Vese 模型[2]一旦初始化便可以自动检测待分割物体的内部轮廓,曲线的初始化与梯度、位置都无关,在很大程度上可以减少噪声干扰,同时适用于物体边缘模糊甚至不连续的情况。基于 Chan-Vese 模型的人脸轮廓提取方法,虽然可以精确地检测出任意图像中的人脸轮廓,包括正面轮廓、侧面轮廓等,但也存在计算量大、分割速度缓慢等问题。文献[3-4]对 Chan-Vese 模型进行了改进,在一定程度上提高了算法的效率。但现有的改进算法都是基于串行化的设计,而随着多核处理器的发展,尤其是以图形处理器(Graphic Processing Unit, GPU)为代表的流处理器的快速发展,CPU-GPU 异构计算已经成为并行计算的主流之一,这为数据级并行的计算提供了良好的并行平台。

本文利用 Chan-Vese 的改进模型——基于全变分 G-Norm 的 Chan-Vese 分割模型[3,5-6](Chan-Vese segmentation Model with total variation G-norm, CVG Model)和基于 OpenCL 的并行编程模型,提出了 Chan-Vese 模型人脸轮廓提取的并行化方法,该方法可以在精确提取人脸轮廓的同时,有效地提高算法的效率,减少图像的分割时间。

1　CVG 模型

本文使用的是基于原对偶混合梯度[7](Primal Dual Hybrid Gradient,PDHG)的 CVG 模型,该模型由 Bresson 等[5]提出,流程如图 1 所示。图 1 中,*solve*()、*solve*1() 和 *solve*2() 对应的计算过程如下:

solve(): $\boldsymbol{v}[i] = \boldsymbol{u}[i] - 0.0042 + 0.012 * \boldsymbol{fr}[i]$
　　// $\boldsymbol{fr}$ 是常量矩阵,代表输入图像
*solve*1(): $RHS\_p1 = \boldsymbol{p}1[i] + A * \boldsymbol{ux}[i]$　　//A 是个常量
　$RHS\_p2 = \boldsymbol{p}2[i] + A * \boldsymbol{uy}[i]$
　$NRHS = sqrt(RHS\_p1 * RHS\_p1 + RHS\_p2 * RHS\_p2)$
　$\boldsymbol{p}1[i] = RHS\_p1/(\max(NRHS,1))$
　$\boldsymbol{p}2[i] = RHS\_p2/(\max(NRHS,1))$
*solve*2(): $\boldsymbol{u}[i] = B * \boldsymbol{u}[i] + C * (\boldsymbol{v}[i] - D * (\boldsymbol{ux}[i] + \boldsymbol{uy}[i]))$
　　//B、C、D 是常量

其中:

$\boldsymbol{u}$ 的初值为将原始输入图像从 RGB 色彩空间转换为

收稿日期:2014-06-05;**修回日期**:2014-08-28。
基金项目:国家自然科学基金资助项目(61073010, 61272166);国家科技重大专项(核高基)(2012ZX01039-004)。
作者简介:王丽娜(1990-),女,山西晋城人,硕士研究生,主要研究方向:并行计算;　史晓华(1942-),男,江苏宜兴人,副教授,博士,CCF 会员,主要研究方向:编译、并行计算、微处理器体系结构、嵌入式系统。

计算机应用 www.joca.cn

YCBCR 色彩空间后缩放至 1/6 大小后的像素的灰度值矩阵，***p***1 和 ***p***2 为初始值是 0 的矩阵。变量 *W* 和 *H* 的值分别代表缩放后输入图像的宽度和高度。最外层循环次数 *k* 代表了数学运算的迭代次数。在每一次的迭代中，都将上一次迭代产生的输出数据作为本次迭代过程的输入数据，比如 ***u***、***p***1 和 ***p***2。

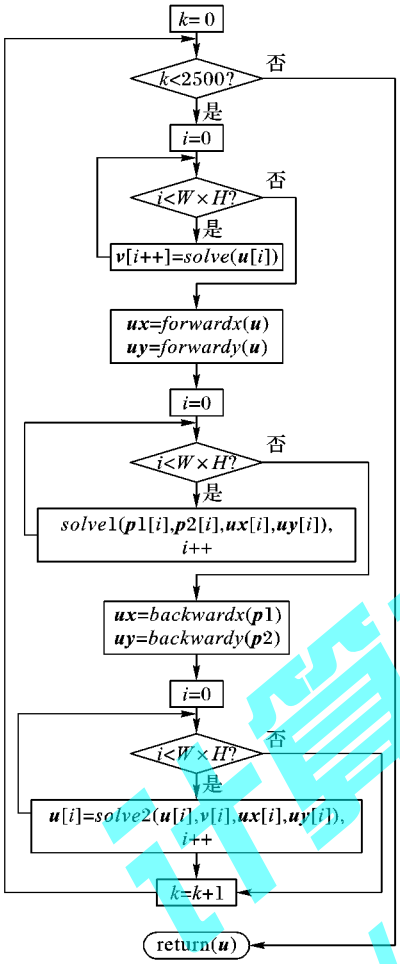


图 1 CVG 算法的流程

矩阵 ***ux*** 和 ***uy*** 是用来保存 ***u*** 在 *X* 方向和 *Y* 方向产生的前向差分，***ux*** 和 ***uy*** 的后向差分被用来求解 ***p***1 和 ***p***2，经过每一次循环后，其值都会被更新。之后，再计算 ***p***1 和 ***p***2 在 *X* 和 *Y* 方向上的前向差分，并将值分别存在 ***ux*** 和 ***uy*** 中。最后，再将 ***u***、***v***、***ux*** 和 ***uy*** 的值更新。

forwardx( ) 和 *backwardx*( ) 分别用来计算 ***u*** 在 *X* 方向的前向差分和后向差分，*forwardy*( ) 和 *backwardy*( ) 分别用来计算 ***u*** 在 *Y* 方向的前向差分和后向差分，4 个函数的流程如图 2 所示，每个函数均分为两步进行计算，图 2(a) 为 *forwardx*( ) 的第一步计算过程，图 2(b) 为 4 个函数的第二步计算过程。对于图 2(a) *forwardx*( ) 的阴影部分，替换成图 2(c)、(d) 和 (e) 的流程，则分别对应为 *forwardy*( )、*backwardx*( ) 和 *backwardy*( ) 第一步的相应计算流程。

对于 *forwardx*( )，先计算当前像素和与它在列 (*X* 方向) 上邻接的像素之间的差分，然后再将中间的行或列的差分值赋值给外层的 4 个边界。其他的函数 *forwardy*( )、*backwardx*( ) 和 *backwardy*( )，除了差值的计算方法不同之外，其计算过程与 *forwardx*( ) 类似。

2 基于 OpenCL 加速的 CVG 算法

程序中的可并行点一般包括循环迭代间细粒度可并行性的挖掘和循环体与过程调用点之间的粗粒度可并行性挖掘。而循环体内数据的数据依赖关系往往决定了程序并行化的程度。因此，要实现程序的并行优化，就需要解决数据依赖关系的分析、变换程序、数据分布和重分布等问题。

而从图 1 可看出 CVG 的循环体数据依赖性较强，不容易实现程序的并行。

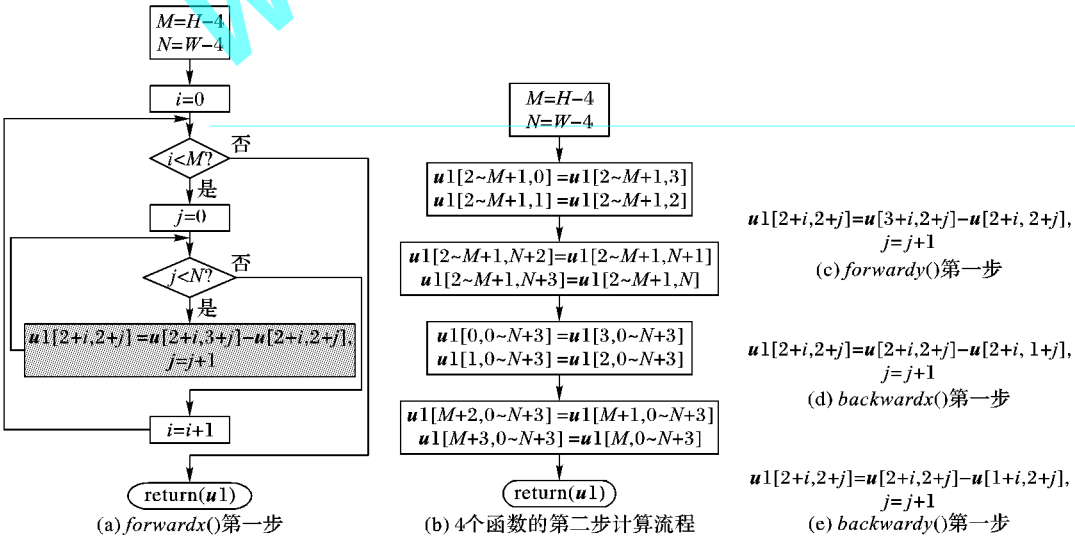
首先，在求前向差分和后向差分 *forwardx*( )、*forwardy*( )、*backwardx*( ) 和 *backwardy*( ) 中，每个像素点的计算结果都依赖于其他像素点的计算值。

其次，CVG 的循环体内有非常复杂的相关性，每一次的最外层迭代 (2 500 次) 都依赖于前一次迭代计算结果，比如 ***u***。通过简单的循环变换是无法消除循环迭代间所携带的数据相关性的。

最后，在循环体内的计算中，存在非理想的嵌套循环，各循环之间有强数据依赖性。在不同的循环体之间，***p***1、***p***2 的计算依赖于 ***ux*** 和 ***uy***，计算完成后，***p***1、***p***2 又决定了 ***ux*** 和 ***uy***。这种复杂的嵌套循环导致程序难以实现并行。

2.1 前向差分和后向差分的构建

为了使 CVG 主循环更容易并行化，就必须重建 CVG 的



(a) *forwardx*()第一步　(b) 4个函数的第二步计算流程　(c) *forwardy*()第一步　(d) *backwardx*()第一步　(e) *backwardy*()第一步

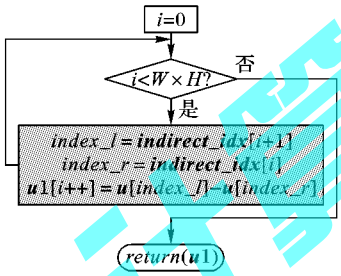
图 2 *X* 和 *Y* 方向的前后向差分计算过程

计算机应用 www.joca.cn

内层循环。首先需要消除前向差分和后向差分的数据依赖关系。

图 3 为重建后的前向差分和后向差分的流程,通过引入一个映射到输入图像的间接索引矩阵,从而消除了内存之间的拷贝。对于输入图像,通过利用内部边界像素的间接索引来重新计算外部边界区域的像素,从而消除了不必要的内存的拷贝。图 3(a)为重建后的 *forwardx*( ),将图 3(a) 的阴影部分替换成图 3(b)、图 3(c) 和图 3(d),则为重建后的 *forwardy*( )、*backwardx*( ) 和 *backwardy*( ) 计算流程,图 3(e) 为大小为 110 × 84 的图像的间接索引矩阵。

重建后的 *forwardx*( )、*forwardy*( )、*backwardx*( ) 和 *backwardy*( ) 通过利用间接索引矩阵 ***indirect_idx***(如图 3 所示),原图像分辨率为 480 × 640,缩放后输入图像大小为 (480/6 +4) × (640/6 +4) = 84 × 110,间接索引矩阵的计算只与输入图像大小有关,如图3 中,对于84 × 110 的输入图像,矩阵的第一个值 ***indirect_idx***[0,0] = 255,与 ***indirect_idx***[3,3] = 255 = 3 × 84 + 3 的值一致(图3(e) 所示),其余边界像素的索引值也可以通过类似的计算得到。而内部像素的索引值是不改变的,如 ***indirect_idx***[4,3] = 4 × 84 + 3 = 399。其他的函数 *forwardy*( )、*backwardx*( ) 和 *backwardy*( ),除了差值的计算方法不同之外,其计算过程和 *forwardx*( ) 相类似。



(a) 重建后的 *forwardx*()

index_l = ***indirect_idx_y***[*i*]
index_r = ***indirect_idx***[*i*]
u1[*i*++] = ***u***[*index_l*]−***u***[*index_r*]

(b) 重建后的 *forwardy*()

index_l = ***indirect_idx***[*i*]
index_r = ***indirect_idx***[*i*]−1
u1[*i*++] = ***u***[*index_l*]−***u***[*index_r*]

(c) 重建后的 *backwardx*()

index_l = ***indirect_idx_y***[*i*]−*W*
index_r = ***indirect_idx***[*i*]−*W*
u1[*i*++] = ***u***[*index_l*]−***u***[*index_r*]

(d) 重建后的 *backwardy*()

255	254	254	255	…	332	333	333	332
171	170	170	171	…	248	249	249	248
171	170	170	171	…	248	249	249	248
255	254	254	255	…	332	333	333	332
339	338	338	339	…	416	417	417	416
⋮	⋮	⋮	⋮		⋮	⋮	⋮	⋮
8823	8822	8822	8823	…	8900	8901	8901	8900
8907	8906	8906	8907	…	8984	8985	8985	8984
8991	8990	8990	8991	…	9068	9069	9069	9068
8991	8990	8990	8991	…	9068	9069	9069	9068
8907	8906	8906	8907	…	8984	8985	8985	8984

(e) 84 × 110图像的间接索引矩阵

图 3　重建后前后向差分以及间接索引矩阵

图 4 为使用图 3 所示的前向和后向差分计算过程的 CVG 算法流程。从图 4 可看出,与单线程算法流程图 1 相比,CVG 的内部循环体已经减少,尽管这两个循环体之间仍存在数据依赖,但比起单线程算法而言,更容易并行。

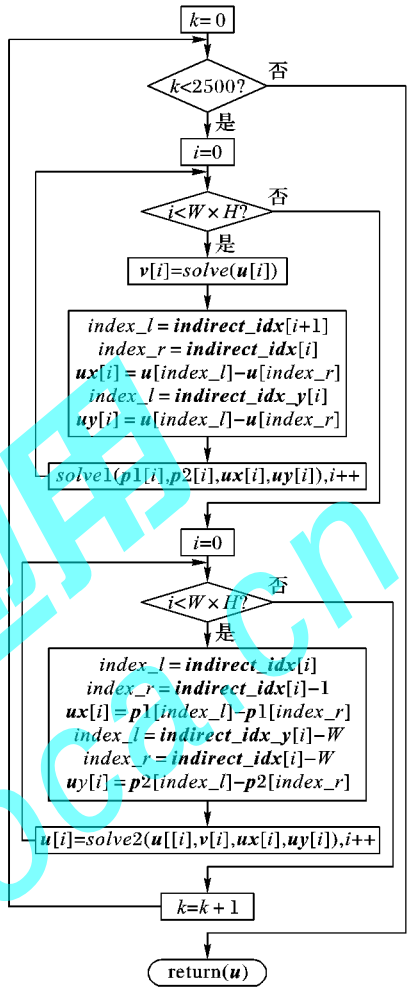


图 4　CVG 的优化算法流程

2.2　CVG 的构建

一个循环只有不存在循环迭代的数据依赖性时,才可以被不加同步地并行执行。但由于 CVG 的最外层循环需要依赖于上次循环输出的数据 ***u***,为了确保并行化时线程访问内存一致,就需要在外层循环结束时对所有数据并行同步。

而循环体可以不需要同步完全并行计算的充要条件就是循环执行的过程中不会依赖于不同的循环实例的执行顺序。但是从图 4 可看出,CVG 内部的两个循环需要顺序执行,第 2 个循环体的 ***p***1 和 ***p***2 需要乱序访问第一个循环体产生的输出。为了可以将 CVG 并行,就必须在两个循环体间进行同步的操作。这样,当不同的线程分别访问 ***p***1 和 ***p***2 时,就可以保证内存的一致性。

图 5 是用 OpenCL 实现的 CVG 的并行化流程图。对于输入图像的每一个像素点,分别单独地用一个线程来进行计算。实验利用 OpenCL 的内置原子操作函数 *atomic_inc*( ) 在 CVG 中设置同步点,引用一个整型数据 *lock* 作为全局锁,用来记录到达该同步点的线程数目。当一个线程未到达同步点时,其他所有已达到同步点的线程暂时停止计算,进入阻塞状态。只有当所有线程都到达同步点时,每个线程才会继续相应的计算。

计算机应用 www.joca.cn

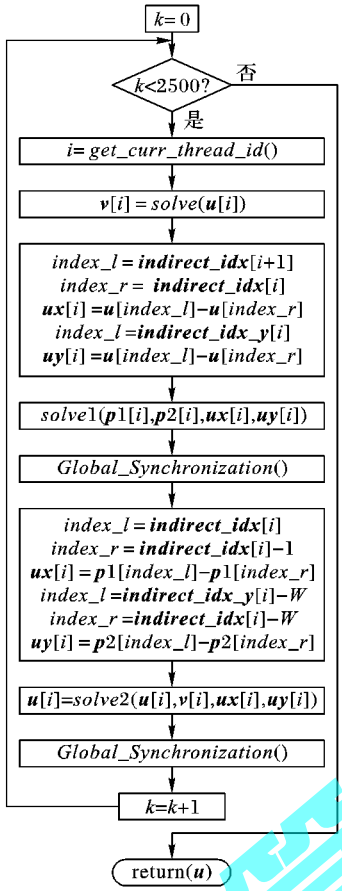


图 5 CVG 的 OpenCL 算法流程

2.3 GPU 上全局锁的优化

由于全局锁 *lock* 的值是线程到达同步点时更新的，因此，如果同一时刻有多个线程到达同步点时，则每个线程需要串行地更新全局锁。从图 7 来看，全局锁的设置在一定程度上提高了算法的性能，在 NVIDIA GTX660 和 AMD FX8530 上，平均加速比分别达到 4.81 和 4.47，但是如果可以在减少对全局锁 *lock* 操作的线程同时依然可以保证数据一致性的话，算法的性能应该会有进一步的提升。

基于上述的讨论，考虑到 GPU 在执行 OpenCL 时，是利用单指令多线程（Single Instruction Multiple Thread，SIMT）执行模型，即一个 warp（线程束）的所有线程共用同一套指令，因此，可以通过设置一个分支，将所有线程的全局锁改为工作组（workgroup）的全局锁，而同一个 workgroup 内线程则用 OpenCL 内置函数来进行同步。而当 OpenCL 内核函数中的指令存在不同的分支时，将会产生控制流指令，而控制流指令可能会引起一个 warp 中的线程分别跳转到不同的分支，甚至导致所有线程串行执行的情况，影响指令吞吐量。

文献[8]设计了有 32 个分支的条件语句，如下所示（其中 *gid* 代表了线程号）：

```
if(gid == 0) {repeat128(t1/ = t2; t2% = t1;)}
else if(gid == 1) {repeat128(t1/ = t2; t2% = t1;)}
...
else if(gid == 31){repeat128(t1/ = t2; t2% = t1;)}
```

从执行结果来看，在同样的计算下，有 32 个分支时执行时间几乎是没有分支情况下的 32 倍。这是因为一个 warp 中的 32 个线程分别会跳到不同的分支上进行处理（即 warp 内的线程执行的指令指针指向不同的位置），则 GPU 的流多处理器（Stream Multiprocessor，SM）就需要把每一个分支的指令发射到标量流处理器（Stream Processor，SP）上再由 SP 根据线程逻辑决定需不需要执行，执行时间也将是执行多个分支所用时间之和。

虽然在 SIMT 编程模型中使用分支会导致性能的下降[8]，但在本实验中，由于一个 workgroup 只有一个线程会执行该分支来更新全局锁 *lock* 的值，所以执行全局锁操作的时间也就是该线程执行该分支所用的时间。尽管之前所有线程都对全局锁更新时，不存在分支，但是由于同一时刻只能有一个线程操作全局锁，所以各线程仍是串行执行的。因此，对 workgroup 进行上锁操作，执行时间就从所有线程的数量级降低到 workgroup 级别上，因此 workgroup 同步锁的设置使得 CVG 的并行化性能有了明显提高。GPU 上利用 workgroup 全局锁同步化后的算法如下所示：

```
index = get_local_id();
if(it_is_the_first_thread_in_the_workgroup(index))
  lock();
barrier(CLK_LOCAL_MEM_FENCE);
```

对于同一个 workgroup 的所有线程，仅给第一个线程设置全局锁（if 程序块），当此线程更新全局锁 *lock* 后，再调用 *barrier*（*CLK_LOCAL_MEM_FENCE*）来保证该 workgroup 的所有线程都同时到达同步点。

3 实验与分析

实验选取的 GPU 和 CPU 分别为：NVIDIA GTX660 和 AMD FX8530 4 GHz 8 核 CPU，内存为 DDR3 1 866 MHz 4 GB，显存为 2 GB DDR5，OpenCL 版本为 AMD APP SDK V2.7 OpenCL1.2 和 CUDA 4.2.9 OpenCL1.2。

实验选用 GTAV[9] 人脸库作为所用数据集，以满足大样本量的数据要求。GTAV 人脸数据库共有 1 188 幅分辨率为 240×320 的人脸图像，由 44 人在不同光照、表情下得到。实验将数据集照片分辨率归一化设置为 480×640 进行处理，对 GTAV 人脸库的大多数图像进行检测并提取了较好的轮廓曲线。

图 6 即为人物 ID01 的 9 张图像所得到的人脸轮廓提取的示意图，依次分别为：ID01_001.jpg ~ ID01_009.jpg。其中，不规则实体线代表的是检测所得到的轮廓线，椭圆实体线代表由轮廓线进一步生成的较为圆滑的椭圆曲线。

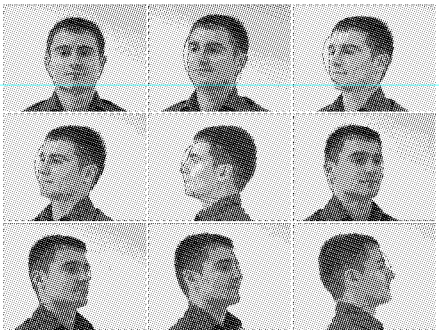


图 6 人脸轮廓示意图

由图 6 可看出，无论对于正面人脸或者侧面人脸，CVG 算法都能够正确地检测出人脸，基本上去除了颈部区域，并且能够获得较为精确的人脸轮廓。

图 7 为 ID01_001.jpg ~ ID01_003.jpg 的 CV 单线程和 OpenCL 下多线程的性能示意图。从图 7 可看出：在 NVIDIA

计算机应用 www.joca.cn

GTX660 和 AMD FX-8530 上,并行化后的 CVG 速度比在 AMD FX-8530 上运行的单线程的 CVG 平均要快 10.95 和4.47 倍。在 NVIDIA GTX660 上,相对于对 CVG 的所有线程设置全局锁的加速比 4.81,只对 workgroup 设置全局锁的加速比可以达到 10.95。然而由于 CPU 的体系结构与 GPU 不同,所以这个优化针对 CPU 是无效的。

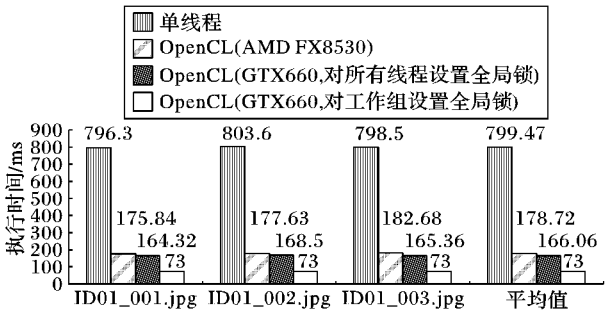


图 7　CVG 时间性能

在上述针对 GPU 的 CVG 优化中,设置了一个 2 维的全局工作组,大小为[W,H],对应的局部工作组大小为[W/N,H/M]。为了充分利用 NVIDIA GTX660 的 5 个计算单元以及 960 个流处理器,$N \times M$ 的值至少要比 5 大。而对于 AMD FX8530,将全局工作组设置为一维的,大小为 32,同 GPU 一样,为了利用 CPU 8 核的优势,将局部工作组设置为 4,每一个线程分别处理多个像素点。

图 8(a)为输入图像 ID01_001.jpg 在 AMD FX8530 八核的测试时间作为基准,当 workgroup 的个数增加时,在理想状态下其时间变化的一个线性曲线。如在 AMD FX8530 上,当只有一个工作组时,所需时间是 870.40 ms,随着工作组的增加,即全局工作组/局部工作组 = *NumofGroup* 的递增,在 *NumofGroup* 达到 8 之前,所需时间应为 870.40/*NumofGroup*,而当 *NumofGroup* 增到 8 之后,时间应该在 870.40/8 = 108.80 ms处浮动。这是由于工作组会被分到不同的计算单元去执行,当其达到 8 时,会充分利用 AMD FX8530 的 8 个核。

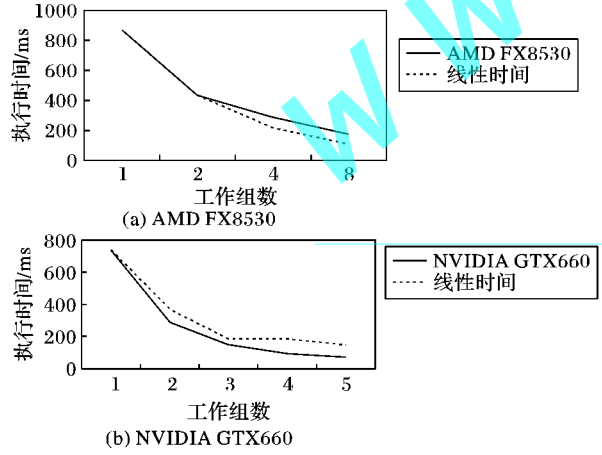


图 8　CVG 的可扩展性性能

从图 8(a)中能够看出,在 AMD FX8530 上:当 workgroup 的个数为 2 时,CVG 的执行时间与理论值一致;当 workgroup 的个数继续增加时,CVG 的执行时间都要比理论值情况下要高,但是理论值与实际值之间的差距变化却很小。因此,当 CPU 的计算单元(核数)越多,CVG 算法的性能应该还会有所提高。

输入图像 ID01_001.jpg 在 NVIDIA GTX660 上执行的情况如图 8(b)所示。相比 AMD FX8530,NVIDIA GTX660 有 5 个计算单元,因此将 workgroup 分别从 1 增至 5。与 CPU 出现的现象不同,CVG 的实际执行时间要比理论值要少,这可能是由于当只有一个工作组时,会有一些计算资源的瓶颈问题,如同步化、存储器带宽等。随着 workgroup 的增加,CVG 的执行时间也逐渐下降,因此,当 GPU 的硬件允许有更多的计算单元时,CVG 的性能还会有很大的提升。

4　结语

本文提出了基于 Chan-Vese 模型改进的 CVG 模型的并行优化算法,详细讨论了在多核 CPU 和 GPU 下的性能。随着计算单元个数的增加,算法加速比会逐渐提高,与单线程的 CVG 相比,实验结果表明,在 NVIDIA GTX660 和 AMD FX8530 上的平均加速比可达到 10.95 和 4.47。

参考文献:

[1] SOBOTTKA K, PITAS I. Segmentation and tracking of faces in color images [C]// Proceedings of the Second International Conference on Automatic Face and Gesture Recognition. Piscataway: IEEE Press, 1996: 236 – 241.

[2] CHAN T F, VESE L A. Active contours without edges[J]. IEEE Transactions on Image Processing, 2001, 10(2): 266 – 277.

[3] CHAN T F, ESEDOGLU S, NIKOLOVA M. Algorithms for finding global minimizers of image segmentation and denoising models[J]. SIAM Journal of Applied Mathematics, 2006, 66(5): 1632 – 1648.

[4] SONG B, CHAN T. A fast algorithm for level set based optimization [EB/OL]. [2010-10-10]. ftp://ftp.math.ucla.edu/pub/camreport/cam02-68.pdf.

[5] BRESSON X, ESEDOGLU S, VANDERHEYNST P, *et al.* Fast global minimization of the active contour/snake model[J]. Journal of Mathematical Imaging and Vision, 2007, 28(2): 151 – 167.

[6] THEOCHARIDES T, VIJAYKRISHNAN N, IRWIN M J. A parallel architecture for hardware face detection[C]// Proceedings of the 2006 IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures. Piscataway: IEEE Press, 2006: 452 – 453.

[7] ZHU M, CHAN T F. An efficient primal-dual hybrid gradient algorithm for total variation image restoration[EB/OL]. [2010-10-10]. ftp://ftp.math.ucla.edu/pub/camreport/cam08-34.pdf.

[8] YAN X, SHI X, WANG L, YANG H. An OpenCL micro-benchmark suite for GPUs and CPUs[C]// Proceedings of the 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies. Piscataway: IEEE Press, 2012: 53 – 58.

[9] TARRÉS F, RAMA A. GTAV face database[DB/OL]. [2013-10-10]. http://gps-tsc.upc.es/GTAV/ResearchAreas/UPCFaceDatabase/GTAVFaceDatabase.htm.

[10] ANAND M, SINGH V, KAUSHIK S. Fast evolution of curve based on Chan-Vese model for specific images[J]. International Journal of Engineering, 2012, 1(9): 55 – 59.

[11] WANG X, HUANG D, XU H. An efficient local Chan-Vese model for image segmentation[J]. Pattern Recognition, 2010, 43(3): 603 – 618.

[12] YE J, BRESSON X, GOLDSTEIN T, *et al.* A fast variational method for surface reconstruction from sets of scattered points[EB/OL]. [2013-10-10]. ftp://ftp.math.ucla.edu/pub/camreport/cam10-01.pdf.

[13] BARBOSA D, DIETENBECK T, SCHAERER J, *et al.* B-spline explicit active surfaces: an efficient framework for real-time 3-D region-based segmentation[J]. IEEE Transactions on Image Processing, 2012, 21(1): 241 – 251.

计算机应用 www.joca.cn