第47卷第10期 中国科学技术大学学报 Vol.47,No.10
2017年10月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Oct. 2017

文章编号:0253-2778(2017)10-0808-09

基于OpenCL的加速鲁棒特征算法并行实现

郭 景,陈贤富

(中国科学技术大学信息科学技术学院,安徽合肥 230027)

摘要:加速鲁棒特征算法(speed up robust features, SURF)的时间复杂度大,传统串行计算的方法,实时性难以保证.针对上述问题,提出一种基于OpenCL架构的SURF并行实现方法.首先对算法中的积分图的计算、Hessian响应图、特征点主方向、特征点描述等步骤实施数据并行和任务并行处理,并给出详细的算法流程.接着从OpenCL架构的数据传输、内存访问以及负载均衡等方面优化算法性能.实验结果表明,该算法对不同分辨率的图片均实现了10倍以上的加速比,一些高分辨率的图片甚至可以达到39.5倍,并且算法适用于多种通用计算平台.

关键词:加速鲁棒特征;开放运算语言;图像处理器;并行计算

中图分类号:TP391 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2017.10.002

引用格式:郭景,陈贤富. 基于OpenCL的加速鲁棒特征算法并行实现[J]. 中国科学技术大学学报,2017,47(10):808-816.

GUO Jing, CHEN Xianfu. Parallel implementation of surf algorithm based on OpenCL[J]. Journal of University of Science and Technology of China, 2017,47(10):808-816.

Parallel implementation of surf algorithm based on OpenCL

GUO Jing, CHEN Xianfu

(*School of Information Science and Technology, University of Science and Technology of China, Hefei* 230027, *China*)

Abstract: SURF algorithm has high computational complexity and can not meet the real-time requirement. To solve these problems, a parallel SURF algorithm based on OpenCL was presented. Firstly, data parallelism and task parallelism model were applied to the calculations of the integral images, Hessian detector, orientation and descriptor, and the detailed algorithm process was given. Secondly, the performance of the parallel algorithm was optimized from data transmission, memory access and load balancing. The experimental results show that the algorithm can achieve more than 10 times speedup for images with different resolution, and some high-resolution images can even reach up to 39.5 times. Furthermore, it can be applied to a variety of general purpose computing platforms.

Key words: SURF; OpenCL; GPU; parallel computing

0 引言

图像特征匹配算法广泛应用于全景图像拼接[1]、机器人视觉定位[2]等领域.目前比较流行的特征匹配算法有SIFT、SURF和BRISK.SURF算法是SIFT算法的改进,在保证准确性的情况下,SURF算法速度比SIFT快3~5倍[3].BRISK算法基本可以满足实时性的要求,但是它的准确性不尽

收稿日期:2017-04-01;**修回日期**:2017-08-05

作者简介:郭景,男,1992年生,硕士生.研究方向:图像匹配、目标跟踪.E-mail:guojing6@mail.ustc.edu.cn

通讯作者:陈贤富,博士/副教授.E-mail:xfchen@ustc.edu.cn

如人意[4].凭借较为突出的综合性能,SURF 算法得到了广泛的应用.

随着实际应用中需要处理的图片分辨率不断提高,现有的 SURF 算法越来越无法满足实时性要求,提高该算法性能的要求日益迫切.与此同时,随着 FPGA、GPU 的计算速度迅猛上升,越来越多的复杂算法被移植到了异构计算平台.研究人员针对 SURF 算法的异构并行计算也做了一些研究.文献[5-6]提出基于 FPGA 加速的 SURF 算法,但是 FPGA 的专用性,决定了该算法无法应用在通用的计算平台上.文献[7-9]提出了基于 CUDA 的 SURF 算法加速,但是 CUDA 架构只能用于 NVIDIA 的 GPU,同样限制了该算法的应用范围,并且实验结果未见生成特征点数的描述.已有的一些基于 OpenCL 架构的工作[10]多是从整体上对算法进行粗粒度并行化,没有深入算法的细节步骤,加速效果仍有较大发挥空间.

本文提出的基于 OpenCL 架构的 SURF 算法并行设计,并行化过程尽可能地保留了原有算法的数据流结构,主要针对 SURF 算法中时间复杂度较高的步骤(积分图的计算、Hessian 响应图、特征点主方向计算及特征点描述等)细致地进行数据并行化和任务并行化处理;然后从通用并行计算架构 OpenCL 的数据传输、内存访问以及负载均衡等方面考虑,对算法性能进一步优化.

1　OpenCL 架构

目前流行的异构计算架构有 OpenCL 和 CUDA.CUDA 是 NVIDIA 推出的计算平台,只适用于该厂商生产的 GPU,而 OpenCL 具有通用性,支持绝大多数 GPU,甚至包括移动计算平台上的 GPU,因此使用 OpenCL 架构的算法可以很方便地在不同异构平台之间移植.这也是本文选用 OpenCL 架构的重要原因之一.

OpenCL 架构最初由苹果公司开发,并在 AMD、IBM 等技术团队合作下逐渐完善.它实现异构并行计算的方法可以抽象为 4 个模型,分别是平台模型、执行模型、内存模型和编程模型[11].

1.1　平台模型

OpenCL 的平台模型定义了使用 OpenCL 异构平台的高层表示.OpenCL 总是包括一个宿主机,实际应用中一般是 CPU.宿主机与一个或多个 OpenCL 设备连接,设备一般指 GPU.OpenCL 的设备进一步划分为计算单元(CU),计算单元更进一步划分为多个处理单元(PE).

1.2　执行模型

OpenCL 应用分为两个部分:一个宿主机程序和多个内核(kernel).宿主机程序在由宿主机串行执行,它定义了与 OpenCL 对象之间的交互方法,并且实现对设备的控制.内核程序在 OpenCL 设备上并行执行,它由宿主机程序定义,当宿主机发出提交内核命令后,OpenCL 运行时系统会在设备上创建一个 N 维工作索引空间,对应这个索引空间中的每个点分别执行内核的实例.每个实例称为工作项,多个工作项组成工作组.

1.3　内存模型

OpenCL 的内存模型将内核用到的内存分为 4 种类型:全局内存、常量内存、局部内存和私有内存.它们的作用范围不同,并且读写速度相差巨大.从左到右内存的作用范围依次递减,读写速度递增.各类内存的有效作用范围及其在异构平台的位置如图 1 所示.

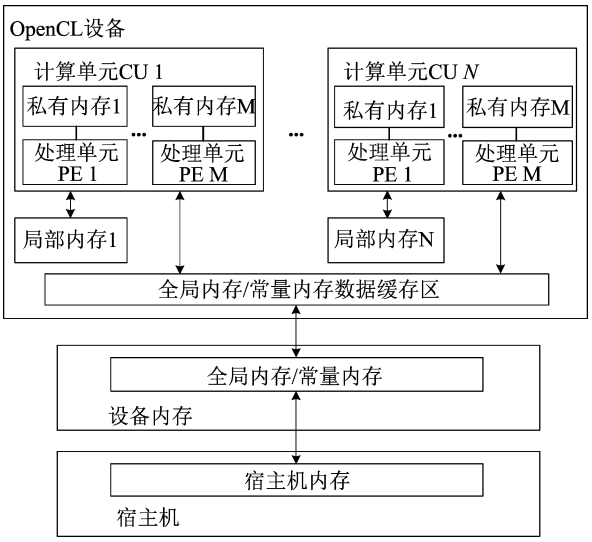


图 1　OpenCL 内存结构

Fig.1　Memory structure of OpenCL

1.4　编程模型

OpenCL 的编程模型支持数据并行模型和任务并行模型.

数据并行模型指的是同一段指令并发作用在内存对象的不同数据元素上.OpenCL 提供了层次结构的数据并行,即工作组中工作项的数据并行,再加上工作组层次的数据并行,因此开发人员需要显式地规定参与计算的所有工作项节点数,同时还要规定每个节点所属的工作组.

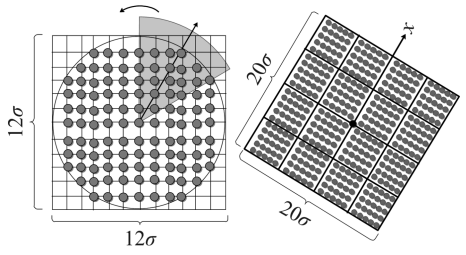
OpenCL 架构为数据并行化提供了最大的便利，但是它也同时支持任务并行模型.如果算法中的某个步骤与其他步骤完全独立，可以利用一个乱序队列同时执行多个步骤的内核以实现任务并行.

2 SURF 特征匹配算法

2006 年 Bay 等[3]提出了 SURF 特征匹配算法，在保证检测的准确性前提下，极大地降低了算法的时间复杂度.SURF 算法通过 Hessian 矩阵检测图像金字塔中不同尺度图像的特征点.为了保证特征描述符具有旋转不变性，计算以特征点为中心，半径为 6σ 的圆形邻域内的 Haar 特征值；接着按逆时针方向旋转一个 60°的扇形区域，如图 2(a)所示，当扇形内像素点的 Haar 特征值总和最大时，该扇形的方向即作为特征点主方向；最后以特征点主方向为 x 轴，构造边长为 20σ 的正方形区域，并等分成 16 块子区域，如图 2(b)所示，计算每块子区域内的 Haar 特征值，并构造一个 4 维向量如下：

$$v_{\text{subregion}}=\left(\sum \mathrm{d}x,\sum \mathrm{d}y,\sum|\mathrm{d}x|,\sum|\mathrm{d}y|\right) \tag{1}$$

式中，$\mathrm{d}x$、$\mathrm{d}y$ 为 x 和 y 方向的 haar 特征值.16 块子区域的 4 维向量共同组成一个 64 维的特征点描述符.



(a)圆形邻域 (b)方形邻域

图 2 特征点邻域示意图

Fig.2 Neighborhood of feature point

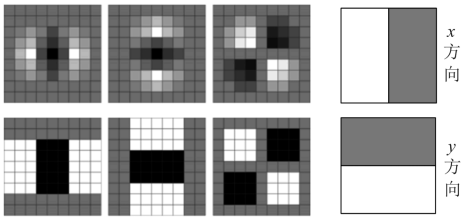
由于 SURF 算法引入了尺度空间理论，要将原图像与可变尺度的 2 维高斯滤波模板进行卷积，形成图像金字塔.此时 Hessian 矩阵可以表示为

$$\boldsymbol{H}(x,\sigma)=\begin{bmatrix}L_{xx}(x,\sigma) & L_{xy}(x,\sigma)\\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma)\end{bmatrix} \tag{2}$$

为了加快图像的卷积过程，SURF 算法采用方框滤波器近似式(2)中的高斯二阶微分，对应的方框滤波模板如图 3(a).使用方框滤波处理后，图像的 Hessian 响应值计算近似为

$$\det(\boldsymbol{H}_{\text{approx}})=D_{xx}D_{yy}-(0.9D_{xy})^2 \tag{3}$$

式中，D_{xx}、D_{yy} 和 D_{xy} 为各方向的方框滤波的结果.SURF 算法中还大量使用 Haar 特征，Haar 特征模板如图 3(b)所示.



(a)方框滤波 (b)Haar特征模板

图 3 模板

Fig.3 Box filters and Haar wavelet filters

此外，如果每次方框滤波和 Haar 特征的计算都要统计众多像素点的和，那么庞大的计算量无疑会严重降低算法的运行速度.为此，SURF 算法引入了积分图作为图像的中间表达形式，定义如下：

$$I_\Sigma(x,y)=\sum_{i=0}^{i\leqslant x}\sum_{j=0}^{j\leqslant y} i(x,y) \tag{4}$$

式中，$I_\Sigma(x,y)$ 为积分图；$i(x,y)$ 为像素灰度值.所有的方框区域内像素和的计算转成 3 次积分图的加减运算，加快了计算的速度.

$$\Sigma=I_\Sigma(D)-I_\Sigma(C)-I_\Sigma(B)+I_\Sigma(A) \tag{5}$$

其中 A、B、C 和 D 是方框的四个顶点坐标.

3 SURF 算法并行实现

实际应用中，SURF 算法的程序设计可以分为以下 4 个步骤：①计算积分图；②计算尺度空间中多个尺度的 Hessian 响应图，并通过三维线性插值定位亚像素级特征点；③计算特征点主方向；④构建 64 维特征描述符.

由上述算法介绍可知，SURF 算法的各步骤均具有良好的并行特性.本文基于 OpenCL 架构的编程模型，充分地利用各步骤的并行特性，对算法细致地进行数据并行化和任务化并行处理.SURF 算法并行实现的系统如图 4 所示.由图 4 可以看出，算法的核心步骤都是在 GPU 上并行实现，而 CPU 主要完成图像读取、图像预处理以及对 GPU 的控制.

3.1 积分图计算

从式(4)不易直接看出积分图计算的并行特性，但是通过改变积分图累加的次序，可以从以下 2 个步骤对积分图进行并行计算.

(Ⅰ)对图像的每一行的像素灰度值进行累加，行与行之间相互独立，中间结果表示为

$$I_\Sigma(x,y)=I_\Sigma(x-1,y)+i(x,y) \tag{6}$$

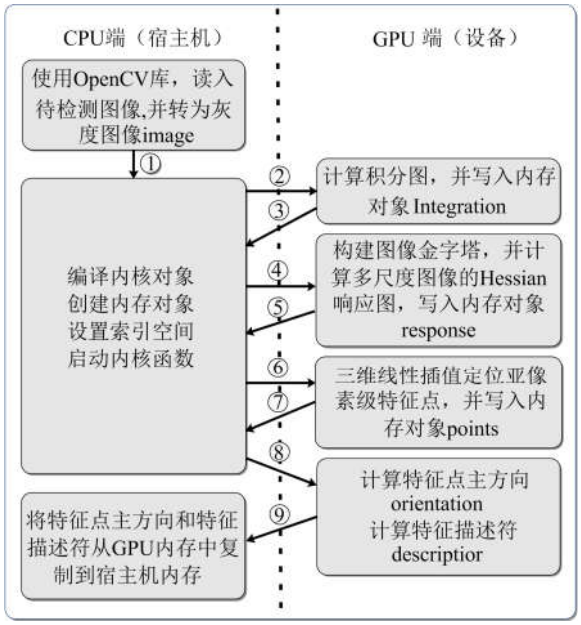


图 4　并行实现系统框图

Fig.4　System chart of parallel implementation

（Ⅱ）接着对步骤（Ⅰ）的中间结果 $I_\Sigma(x,y)$ 的每一列独立进行累加，得到图像最后的积分图表示为

$$I_\Sigma(x,y)=I_\Sigma(x,y)+I_\Sigma(x,y-1) \quad (7)$$

假设输入图像经过预处理后，得到 $n*m$ 的灰度图像，其中 $n>=m$.使用 OpenCL 架构分配 n 个工作项，构成一维索引空间.每个工作项负责步骤（Ⅰ）或步骤（Ⅱ）的累加计算.首先由工作项 ID 获得每行的起始位置，对图像的每一行同时进行累加，结果写入积分图对象，然后同步全局内存对象，等待索引空间的前 m 个工作项完成步骤（Ⅰ）；接着再由工作项 ID 获得每一列的起始位置，对积分图内存对象的每一列进行累加.伪代码如下：

```
输入：原始图像数组 image、图像长度 m、图像宽度 n
输出：积分图内存对象 integration
1. __kernel buildIntegralImage(...) {
2. int index=get_global_id(0);
3. if(index<m) {//仅 index<m 的节点参与步骤(1)
4.    for(int i=0;i<n;i++) {
5.   步骤(1)输入图像每行累加，结果写入 integration;}
6. }
7. 全局内存对象同步 barrier，等待完成步骤 1;
8. for( i=0;i<m;i++) {
9.   步骤(2)中间结果 integration 的每列累加;}
10. }
```

3.2　多尺度 Hessian 响应图计算

多尺度 Hessian 响应图的构建，首先要利用积分图计算图像金字塔每一层每个像素点的方框滤波值，得到近似的高斯二阶微分 D_{xx}、D_{yy} 和 D_{xy}；然后根据式(3)计算得到多尺度 Hessian 响应图.这个过程存在 2 级并行.

（Ⅰ）图像级并行.图像金字塔中的每一层都会形成一个尺度的 Hessian 响应图，层与层之间的计算相互独立，可以并行实现.

（Ⅱ）像素级并行.同一层图像的各像素点的 Hessian 响应值的计算既相互独立，也可以并行.

假设图像金字塔总共有 N 层，那么可以循环启动 N 个 kernel，每个 kernel 对应图像金字塔中一层图像的 Hessian 响应图的计算.另外，每层图像中还存在像素级并行，因此为每个 kernel 分配与其对应的图像大小一致的二维索引空间，即每个工作项完成图像中一个像素点的 Hessian 响应值的计算，具体过程如图 5 所示.伪代码如下.这样的设计虽然能够实现充分利用算法的并行特性，但是也存在一些问题，将在下文算法优化部分介绍.

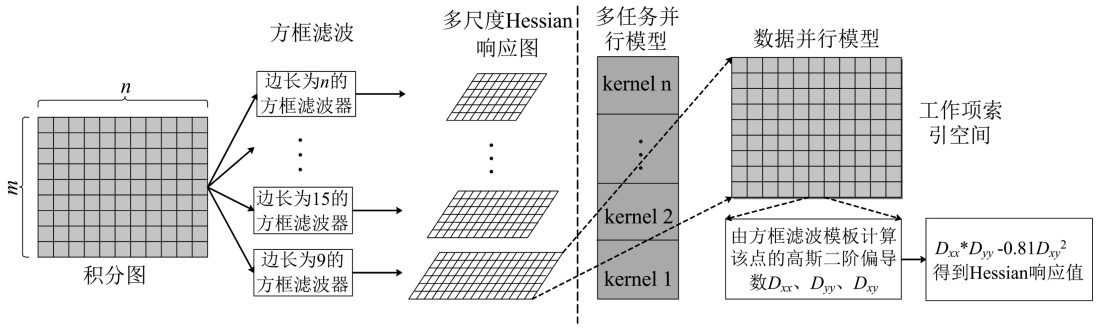


图 5　Hessian 响应图计算并行处理

Fig.5　Parallel implementation of calculation of Hessian response map

```
输入：积分图内存对象 integration、方框滤波器边长 filterSize
输出：Hessian 响应图内存对象 responses
1. __kernel buildResponseLayer(...) {
2. 计算工作项的二维索引(index0,index1);
```

3. 由 filterSize 计算方框滤波的参数 border 和 lobe；
4. 结合积分图计算该点 x 方向方框滤波值 D_{xx}；
5. 结合积分图计算该点 y 方向方框滤波值 D_{yy}；
6. 结合积分图计算该点 xy 方向方框滤波值 D_{xy}；
7. Hessian 响应值$=D_{xx}*D_{yy}-0.81D_{xy}*D_{xy}$；
8. 根据二维索引，结果写入 responses 的对应位置；
9. }

由上述过程得到多个尺度的 Hessian 响应图.将连续的 4 个 Hessian 响应图合成一组，根据组内各点与其立体邻域的 Hessian 响应值定位特征点位置，再由三维线性插值法确定亚像素级特征点位置.

3.3 特征点主方向计算

确定特征点的位置后，接着计算特征点主方向.由第 2 节的算法介绍可以看出，特征点主方向的计算包含 3 级的并行.

(Ⅰ)特征点级并行.由 3.2 节确定了多个特征点，特征点之间相互独立，每个特征点的主方向计算可以并行实现.

(Ⅱ)像素级并行.以特征点为中心的圆形邻域内的所有像素点的 Haar 响应值计算相互独立.

(Ⅲ)扇形级并行.扇形旋转过程中的各个位置的 Haar 特征值累加相互独立.

实际的算法实现过程，无法穷举图 2(a)中扇形旋转过程所有位置的情况，只能令扇形每次旋转一定的角度.旋转角度越小，越接近扇形旋转过程，但是相应的计算量就越大.为了确定旋转角度的大小，对图 2(a)进行特征点匹配，调整旋转角度得到的匹配点的个数，如表 1 所示.

表 1 不同的旋转角度对应的匹配点数

Tab.1 Matching points corresponding to different rotation angles

旋转角度/(°)	3	6	9	18	30	40
匹配点数/个	68	68	65	61	55	43

由表 1 可以看出，随着旋转角度的增大，能够匹配的特征点数越少.当旋转角度为 9°时，已经能取得比较好的近似效果.故本文算法令扇形每次旋转角度为 9°，那么将整个圆分为 40 份，对每个位置编号为 0～39.另外，从图 2(a)可以看出，当图像缩小 σ 倍后，圆形区域内的所有像素点都将包含在一个 12 * 12 的矩形区域内；因此，为每个特征点的计算分配一个 12 * 12 的局部二维索引.假设特征点的个数为 N，那么所有特征点的计算，则需要 12 * 12 * N 个工作项.在逻辑上，将所有工作项分为 N 个 work-group，每个 work-group 的工作项数目都是 12 * 12.这样安排的好处就是每个 work-group ID 即为特征点编号，每个 work-group 内的局部二维索引即对应图 2(a)矩形区域内各像素点位置.全局工作空间的映射如图 6 所示.首先计算圆形邻域内的每个像素点的 Haar 特征值，并使用同步函数，等待所有的工作项都完成 Haar 特征值计算.接着前 40 个工作项分别计算编号 0～39 位置的 60°扇形内 Haar 特征值总和，最后比较得出特征值的最大方向作为特征点主方向.伪代码如下.

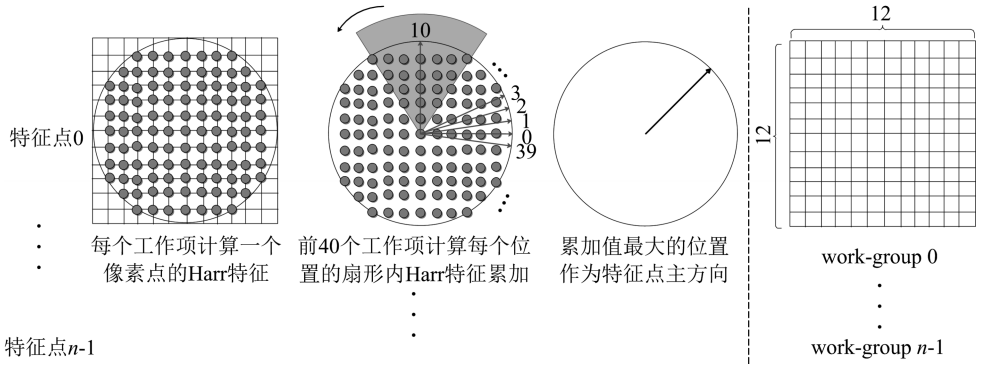


图 6 特征点主方向生成过程

Fig.6 Generating process of principal direction of feature point

输入：特征点信息内存对象 featurePoints
输出：特征点主方向内存对象 orientations
1. __kernel calculateOrientation(...) {
2.由 featurePoints 获取特征点的坐标及尺度；
3.获得局部二维索引(i,j)及全局索引 index；
4. if ($i^2+j^2<36$) { //判断该点是否在圆形邻域内
5. 由 Haar 特征模板计算该点 x 方向 Haar 特征值；
6. 由 Haar 特征模板计算该点 y 方向 Haar 特征值；

7.　计算该像素点相对于 x 轴的角度；}
8. 同步 barrier，等待所有工作项的 Haar 特征值计算完毕；
9. if(index＜40) {//前 40 个工作项参与扇形内特征值累加
10.　for(遍历圆形邻域内所有像素点) {
11.　　if(由像素点的角度判断点是否在该位置扇形内) {
12.　　分别累加 x 和 y 方向的 Haar 特征值；}
13.　}
14. }
15. 同步等待 0～39 位置扇形内 Haar 特征值累加；
16. 比较 0～39 位置的累加和，选出最大值；
17. 对应扇形方向即为主方向，结果写入 orientations；
18. }

3.4　特征描述符计算

由第 2 节算法介绍可以看出，特征描述符利用特征点主方向上构造的一个方形区域作为其邻域，并划分成 16 个方形子区域.这个计算过程存在 2 级的并行.

(Ⅰ)特征点级并行.与特征点主方向类似，每个点特征描述符计算独立.

(Ⅱ)子区域级并行.每个方形子区域独立计算产生 4 维矢量.

特征描述符计算的难点在于以特征点主方向作为 x 轴建立的正方形邻域是倾斜的，如图 2(b)所示，因此，OpenCL 架构的二维索引不能直接对应每个像素点的坐标.本文算法采用坐标变换，将二维索引变换到旋转后的图像坐标.假设正方形以原点为中心，从位置 1 旋转 θ 到位置 2，P_1 和 P_2 分别为旋转前后对应的点，如图 7 所示.

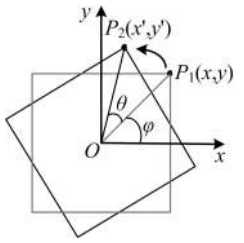


图 7　方形旋转变化

Fig.7　Schematic diagram of square rotation

P_1、P_2 的坐标可以表示为

$$\left.\begin{aligned}x&=\rho\cos\varphi\\y&=\rho\sin\varphi\end{aligned}\right\},\quad\left.\begin{aligned}x'&=\rho\cos(\varphi+\theta)\\y'&=\rho\sin(\varphi+\theta)\end{aligned}\right\}.$$

式中，ρ 表示到原点的距离.由上面两式联立可以得出，方形旋转前后坐标的映射关系为

$$\left.\begin{aligned}x'&=x\cos\theta-y\sin\theta\\y'&=x\sin\theta+y\cos\theta\end{aligned}\right\}\tag{8}$$

另外一个难点是，每个点的 Haar 特征模板也是随着主方向倾斜，无法利用积分图快速计算 Haar 特征值，也只能通过坐标变换，对每个像素点进行累加.为此，宿主机为每个特征点的计算分配一个 work-group，每个 work-group 包含 4 * 4 的局部二维索引，即每个工作项独立计算一个方形子区域，形成 4 维向量.另外，每个工作项对应的子区域位置，可以通过将其二维索引($index_0$，$index_1$)经过

$$\left.\begin{aligned}x&=5index_0-8\\y&=5index_1-8\end{aligned}\right\}\tag{9}$$

线性变换，接着按式(8)旋转变换到子区域中心点坐标.具体的过程如图 8 所示.伪代码如下.

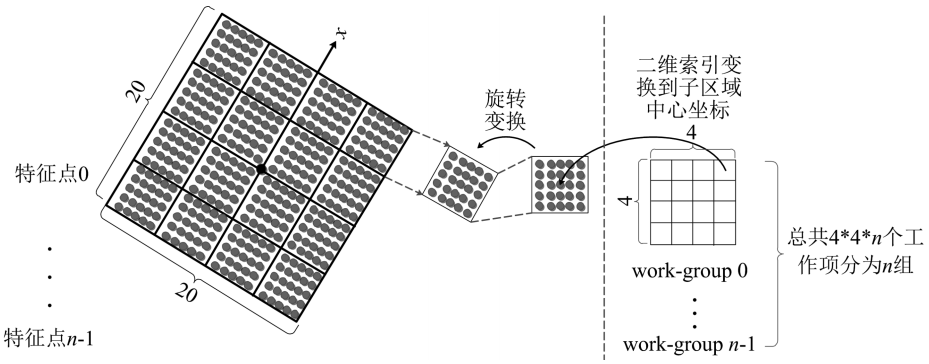


图 8　特征点描述符生成过程

Fig.8　Generating process feature point descriptors

输入：特征点信息内存对象 featurePoints、特征点主方向内存对象 orientations
输出：特征描述符内存对象 descriptor
1. __kernel calculateDescriptor(...) {
2. 由 featurePoints 得到的特征点坐标及尺度；
3. 由 orientations 得到对应特征点的旋转角度；
4. 将二维索引变换到对应子区域中心坐标(x,y)；
5. for(int i=0;i＜5;i＋＋) {//遍历方形子区域内的所有点

```
6.  for(int j=0;j<5;j++){
7.  将点(x+i,y+j)变换到旋转后子区域坐标;
8.  计算该点X方向Haar特征dx,并累加;
9.  计算该点Y方向Haar特征dy,并累加;
10.  }
11.}
12.由式(1)得到对应的4维向量;
13.结果写入myDescriptor的对应位置;
14.}
```

4 算法优化

第3节虽然实现了SURF算法并行处理,但是仍然存在一些影响GPU并行计算速度的问题.对算法进行优化,可以显著地提高GPU的加速比.本文算法采用分步优化的策略,主要从以下4个方面进行优化.

(Ⅰ)数据传输优化.积分图对象、Hessian响应图对象占用的内存较大,并且在算法的多个kernel函数中被用做输入变量.因为在不同kernel之间频繁传递大块数据非常耗时,优化的办法就是将它们设置为全局内存对象,并将大块数据保存在GPU端内存,这样可以避免地增加主机和设备之间交换数据的传输时间.

(Ⅱ)内存访问优化.由于积分图、Hessian响应图等被设置为全局内存对象,但是在所有内存类型中全局内存对象的读写速度较慢,考虑到数据在程序中的访问具有局部性,可以仅将kernel函数需要的部分数据从全局内存对象复制到局部内存对象,在数据使用结束后,将局部内存对象复制回全局内存对象.

(Ⅲ)函数选择优化.内核中使用数学函数,比如取最大值、最小值、三角函数等,通过直接调用OpenCL的内建函数,可以提高算法性能.另外,现在的GPU架构中,所有线程被分为32或64为一组被调度的,而一组中的线程一旦遇到if else语句,那么不同分支路径会被串行执行[11],因此算法应尽量少用if else,如果必须要用,也最好用?:语句来替代.

(Ⅳ)负载平衡.由图5可以看出,每个尺度的Hessian响应图大小相差较大.假设算法包含一个12个尺度图像的图像金字塔,那么顶层和底层的图像大小相差10倍.这意味着如果同时启动多个kernel,一些kernel可能很快地就完成计算,并停下来等待.这就造成了负载的不均衡.为了实现负载均衡,可以考虑将顶层的几个计算量较小的kernel合并成一个,使得不同kernel之间计算量相当.

5 实验结果及分析

本文在Intel i5-4596 CPU、Intel HD Graphics 4600 GPU、8G内存的测试平台上,实现了12个尺度的SURF算法;测试平台的开发环境为VS2012.为了提高算法的可移植性,使用的OpenCL版本为OpenCL 1.1.

从算法的介绍可以看出,积分图计算、多尺度Hessian响应图构建以及特征点定位的计算量主要和输入图像的分辨率有关,而特征点主方向、特征描述符的计算量主要与特征点数有关.为了方便比较本文算法的加速效果,将前者放在一起统称为特征点提取过程,后者统称为特征点描述过程.

5.1 特征点提取过程

为了验证特征点提取过程的加速效果,实验分别使用了串行算法和本文并行算法,测试了5张不同分辨率的图片,得到特征点提取过程各步骤消耗时间,取10次实验的平均值,如表2所示.

表2 特征点提取过程的各个步骤消耗时间(单位:ms)

Tab.2 The consumption time of each step in feature points extraction process(ms)

分辨率	串行算法			本文并行方法		
	积分图计算	Hessian响应图构建	特征点定位	积分图计算	Hessian响应图构建	特征点定位
176*144	0.63	57.57	8.43	0.53	6.00	1.87
320*240	1.21	178.80	23.14	0.96	15.89	3.04
640*480	5.99	671.35	91.34	2.86	54.31	10.34
800*480	6.01	847.52	113.41	4.03	69.92	10.41
1920*1080	32.71	4641.08	612.67	25.76	70.87	23.67

由表2可以看出,随着图像分辨率的提高,各步骤所消耗的时间都有所增加,其中串行算法的消耗时间随分辨率上升的趋势尤为显著.此外,特征点提取过程中的Hessian响应图构建最为耗时,而积分

图计算的耗时最小，相比其他两项几乎都可以忽略不计.

由表 2 中并行算法和串行算法的耗时对比，可以得到各步骤的加速比，如图 9 所示.积分图计算的加速效果最不明显，主要原因是单一工作项的计算量过小，因此大多数时间都消耗在 kernel 函数的启动以及数据的传递.另外，Hessian 响应图计算的加速效果最好，这是因为这个步骤的计算量较大，而且算法的并行化程度高.从整体来看，图像分辨率越高、计算过程越复杂，算法加速效果愈佳.

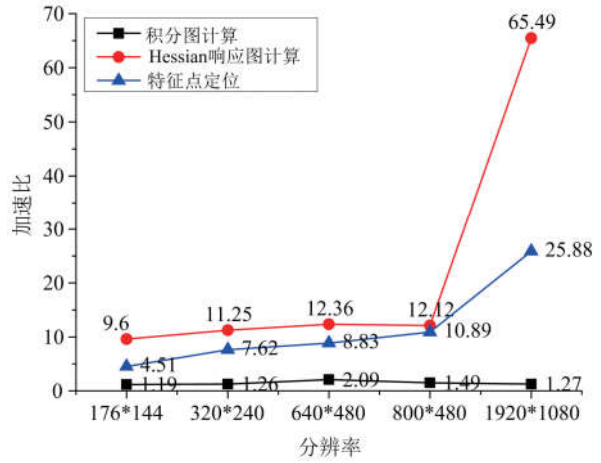


图 9　特征点提取过程

Fig.9　The process of feature points extraction

5.2　特征点描述过程

为了验证特征点描述过程的加速效果，实验选取同一张图片，通过不断调整算法的参数，使得结果显示多种特征点个数的情况.特征描述过程各步骤所消耗时间，取 10 次实验的平均值，如表 3 所示.

表 3　特征描述过程各个步骤消耗时间(单位:ms)

Tab.3　The consumption time of each step in feature extraction process(ms)

特征点个数	串行计算		本文并行方法	
	计算主方向	计算特征描述符	计算主方向	计算特征描述符
11	0.81	6.82	0.50	0.70
50	3.58	33.21	0.85	2.54
207	15.14	136.99	2.67	10.80
502	36.59	328.91	6.18	24.65
1022	71.95	702.56	18.67	67.96

由表 3 可知，在串行算法中，特征描述符的计算量大约是特征点主方向计算量的 10 倍左右.随着特征点个数的增大，两个步骤的耗时呈现出线性递增的趋势.

通过计算表 3 中串行算法和并行算法的耗时，可以得出特征点描述过程的加速比，如图 10 所示.图 10 也说明了 5.1 节的结论，算法越复杂，加速效果越好.另外，两个步骤的加速比曲线呈现出相似的趋势，先是随着特征点个数的增加，加速比迅速增大.当特征点个数增加到一定数量，加速比却缓慢下降了.这是因为随着特征点数增加，并行程度的增大，GPU 的并行计算能力得到充分发挥，但是当特征点增加到一定数量，GPU 内部资源已不能满足并行计算要求，获取 GPU 资源则通过需要排队竞争，导致加速比慢慢地下降.

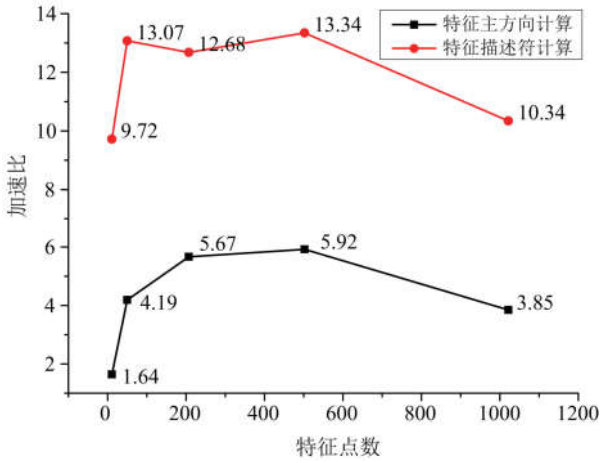


图 10　特征描述过程

Fig.10　The process of feature description

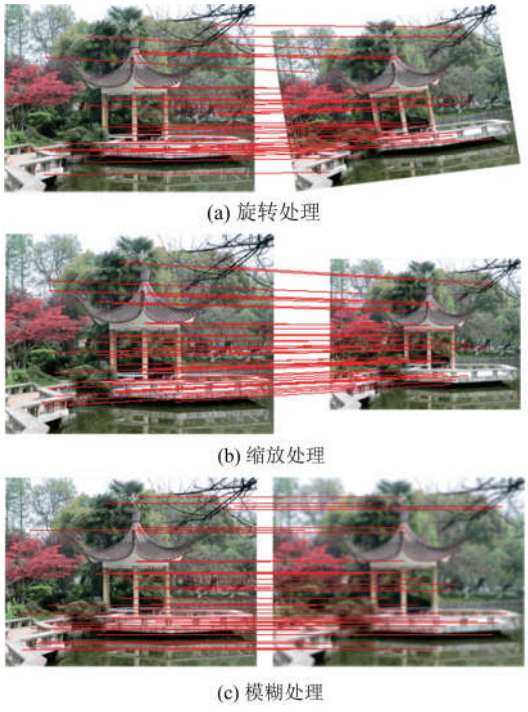
5.3　实验效果及对比

为了说明本文算法的正确性，对算法生成的特征描述符进行匹配.原图像分别与经过旋转、缩放和模糊处理后的图像进行匹配，效果如图 11 所示，原图像拍摄的校园一角.由表 2、表 3 可知，以生成 207 个特征点为例，本文并行算法和串行算法的总耗时以及加速比，如表 4 所示.

表 4　本文算法加速比

Tab.4　The speedup ratio of the algorithm in this paper

分辨率	串行算法时间/ms	本文算法时间/ms	加速比
176 * 144	218.76	21.87	10.00
320 * 240	356.28	34.36	10.36
640 * 480	922.81	82.98	11.12
800 * 480	1122.07	100.83	11.13
1920 * 1080	5442.59	137.77	39.50



(a) 旋转处理

(b) 缩放处理

(c) 模糊处理

图 11 不同条件下的匹配效果

Fig.11 Matching results under different conditions

6 结论

本文基于 OpenCL 架构提出了一种 SURF 算法并行实现方法，对 SURF 算法中的计算积分图、计算 Hessian 响应图、确定特征点主方向、构建特征描述符等步骤进行并行化分割，并给出了详细的算法流程.实验结果表明，算法对旋转、缩放和模糊等干扰均有良好的处理效果.由于算法保留了原有串行算法的数据流结构，在准确性上也与原算法相同，但是相比串行算法，它对不同分辨率的图片进行处理均能实现 10 倍以上的加速比，高分辨率的图片甚至可以达到 39.5 倍.另外，算法还能适用于绝大多数通用计算平台.

参考文献(References)

[1] 赵向阳，杜利民. 一种全自动稳健的图像拼接融合算法[J]. 中国图象图形学报：A 辑，2004，9(4)：417-422.
ZHAO Xiangyang，DU Limin. An automatic robust mosaic algorithm[J]. Journal of Image and Graphics，2004，9(4)：417-422.

[2] WANG Y T，FENG Y C. Data association and map management for robot SLAM using local invariant features [C]// International Conference on Mechatronics and Automation. Takamatsu，Japan：IEEE，2013：1102-1107.

[3] BAY H，TUYTELAARS T，VAN GOOL L. Surf：Speeded up robust features[J]. Computer Vision & Image Understanding，2006，110(3)：404-417.

[4] LEUTENEGGER S，CHLI M，SIEGWART R Y. BRISK：Binary robust invariant scalable key points[C]// Proceedings of the International Conference on Computer Vision. Washington：IEEE，2011：2548-2555.

[5] 赵春阳，赵怀慈. SURF 算法并行优化及硬件实现[J]. 计算机辅助设计与图形学学报，2015，27(2)：256-263.
ZHAO Chunyang，ZHAO Huaici. Parallel optimized method and hardware implementation of SURF algorithm[J]. Journal of Computer-Aided Design & Computer Graphics，2015，27(2)：256-263.

[6] SVAB J，KRAJNIK T，FAIGL J，et al. FPGA based speeded up robust features[C]// Proceedings of the International Conference on Technologies for Practical Robot Applications. Woburn，USA：IEEE，2009：35-41.

[7] 刘金硕，曾秋梅，邹斌，等. 快速鲁棒特征算法的 CUDA 加速优化[J]. 计算机科学，2014，41(4)：24-27.
LIU Jinshuo，ZENG Qiumei，ZOU Bin，et al. Speed-up robust feature image registration algorithm based on CUDA[J]. Computer Science，2014，41(4)：24-27.

[8] 徐晶，曾苗祥，许炜. 基于 GPU 的图片特征提取与检测[J]. 计算机科学，2014，41(7)：157-161.
XU Jing，ZENG Miaoxiang，XU Wei. GPU based image feature extraction and detection[J]. Computer Science，2014，41(7)：157-161.

[9] SINHA S N，FRAHM J M，POLLEFEYS M，et al. GPU-based video feature tracking and matching[R]. MSR-TR-2016-19，Workshop on Edge Computing Using New Commodity Architectures，2006.

[10] CAO J，XIE X F，LIANG J，et al. GPU accelerated target tracking method[J]. Advances in Multimedia，Software Engineering and Computing，2012，1：251-257.

[11] Khronos OpenCL Working Group. The OpenCL Specification Version1.2 [EB/OL]. [2017-10-18] https://www.khronos.org/registry/OpenCL/specs/opencl-1.2.pdf.