文章编号:1000-5862(2020)05-0462-10

基于 OpenCL 的图像灰度化并行算法研究

肖 汉[1,2],郭宝云[3*],李彩林[3],肖诗洋[4]

(1.郑州师范学院信息科学与技术学院,河南 郑州 450044;2.郑州大学信息工程学院,河南 郑州 450001;
3.山东理工大学建筑工程学院,山东 淄博 255000;4.东北林业大学土木工程学院,黑龙江 哈尔滨 150040)

摘要:随着图像数据量的增加,传统单核处理器或多处理器结构的计算方式已无法满足图像灰度化实时处理需求.该文利用图像处理器(GPU)在异构并行计算的优势,提出了基于开放式计算语言(OpenCL)的图像灰度化并行算法.通过分析加权平均图像灰度化数据处理的并行性,对任务进行了层次化分解,设计了2级并行的并行算法并映射到"CPU+GPU"异构计算平台上.实验结果显示:图像灰度化并行算法在OpenCL架构下NVIDIA GPU计算平台上相比串行算法、多核CPU并行算法和CUDA并行算法的性能分别获得了27.04倍、4.96倍和1.21倍的加速比.该文提出的并行优化方法的有效性和性能可移植性得到了验证.

关键词:图像灰度化;加权平均;图形处理器;开放式计算语言;并行算法

中图分类号:TP 311 **文献标志码**:A **DOI**:10.16357/j.cnki.issn1000-5862.2020.05.03

0 引言

由于受到种种条件限制和随机干扰的影响,人们往往不能直接从图像中获得所需信息,必须对彩色图像进行灰度化预处理,才能使图像所包含的信息能更好地被提取和使用[1-2].随着大数据时代的到来,图像的灰度化等预处理数据量正在呈指数量级激增,使得传统的单核处理器或多核处理器的处理方式难以满足图像数据处理的实时性要求,高性能计算技术的发展为大数据处理提供了计算加速的平台[3-5].CPU+图形处理器(Graphic Processing Unit,GPU)的异构计算体系结构以高性能和低功耗成为主流的计算模式,为众多领域大规模计算提供良好的计算平台[6].一般而言,图像灰度化处理有以下几种方法:最大值法、平均法和加权平均法等.灰度图像的灰度值可取自彩色图像3种分量R、G、B中的最大值或平均值.加权平均法是分别赋予R、G、B不同的权值,然后取其加权后的平均值作为灰度图像的灰度值.

面对复杂多样的异构架构,各种为异构系统设计的并行编程模型相继被提出,如NVIDIA提出的计算统一设备架构(Compute Unified Device Architecture,CUDA)、微软提出的C++加速大规模并行性模型(C++ Accelerated Massive Parallelism,C++ AMP)、PGI等提出的OpenACC、Intel提出的基于至强融核的MIC架构等[7-10].然而,这些并行编程模型均只能在特定的异构系统中运用,造成了不同的异构系统须应用不同的编程模型,软件移植的开销和系统设计的难度大大增加[11].面向异构计算平台的开放计算语言(Open Computing Language,OpenCL),并行计算规范,允许让同样的代码运行在支持该规范的任何平台上,包括GPU、Cell处理器、CPU和FPGA等,这大大降低了软件对硬件平台的依赖性.基于OpenCL的图像灰度化并行算法是一种高效的图像灰度化方法,有着较高的性价比和平台移植性,极具研究意义和应用价值[12-13].

该算法解决了在异构计算设备上对于大规模图

收稿日期:2020-02-19

基金项目:国家自然科学基金(41701525,41601496),山东省自然科学基金(ZR2017LD002)和山东省重点研发计划(2018GGX106002)资助项目.

作者简介:肖 汉(1970-),男,湖北武汉人,教授,博士后,主要从事大规模并行算法研究与设计、遥感大数据并行处理的研究.E-mail:xiaohan70@163.com

通信作者:郭宝云(1986-),女,安徽滁州人,讲师,博士,主要从事数字摄影测量与计算机视觉研究.E-mail:guobaoyun@sdut.edu.cn

像灰度化的快速处理问题. 实验数据表明,图像灰度化并行算法与其串行算法精度保持了一致,并取得了明显的性能提升. 本文提出了一种基于"CPU + GPU"异构计算平台的图像灰度化并行算法,并采用可移植性较强的 OpenCL 架构实现了该算法. 在各种 GPU 设备上该算法表现出良好的可移植性.

1　相关研究

数字图像学已经被广泛运用在生物、医药、机械、建筑等现代工业中. 图像灰度化也是轮廓提取、图像识别等热点问题的图像预处理的基础步骤. Nafchi Hossein Ziaei 等[14]通过相关性估算颜色对灰度转换的 3 个全局线性加权参数提出了一种高效的图像灰度化方法. Viswanathan Sowmya 等[15]通过有效结合色度信息,提出了一种改进的性能良好的彩色图像灰度化算法. Rostami Mohamad Javad[16]将逻辑图用于灰度图像的加密,该算法适用于通信和并行处理. 崔宣等[17]利用图像灰度转化运算设计了基于图像快速并行细化的焊缝识别方法. Tan Fenglin 等[18]提出了一种基于灰度直方图短时能量的高动态工业 X-射线图像自动开窗算法,实现了去除 16 位灰度级图像的冗余信息. 杨洪余等[19]针对高分辨率灰度图像提出了一种利用 GPU 并行处理图像中值滤波算法,获得了 17.74 倍加速比. Saleem Sajid 等[20]通过对比分析多传感器灰度图像中具有非线性强度变化的特征,提出具有 BRIEF 描述符的 Harris 检测器,优于多传感器灰度图像上的其他检测器描述符组合. Freitas Pedro Garcia 等[21]提出了一种基于 CUDA 的半色调的水印方法,可以将彩色图像嵌入到二进制黑白图像中,具有较高的视觉质量和性能. Kiermaier Michael 等[22]提出了扩展的双重 Kerdock 码 κ_6^*,该码是一种具有参数的非线性二值灰度图像的 Z_4 线性码,性能比任何二进制线性码都更好. 姜涛等[23]采用SP16HP-G220DSP设计实现了图像灰度变换并行处理系统,满足了机器人系统对图像处理实时性的需求,但该算法系统的成本高、可移植性差. 班志华等[24]使用 CUDA 与 DotNet 混合编程技术,实现了图像灰度处理并行算法的动态链接库,提高了算法实时性,但缺乏对并行算法关键技术的分析. 占正峰等[25]提出了基于 CUDA 的加权平均法图像灰度化并行算法,该算法充分利用 GPU 的并行处理能力,获得了 9.87 倍加速比,但实验数据量较小,未能体现处理大像幅图像的效果.

综上所述,目前的研究在一定程度上提高了图像灰度化算法的执行速度,但是改进的体系结构的方式不具有普遍性和通用性,具有平台局限性,且加速效果均不明显. OpenCL 作为一个通用的与平台无关的异构计算标准,能够充分利用 GPU 强大的并行计算能力以及与 CPU 的协同计算. 本文设计了一种基于 OpenCL 的加权平均法图像灰度化并行算法,在不同计算平台上算法实现了性能移植,算法的并行性进一步提高,有效缩短了系统执行时间.

2　算法的研究与分析

2.1　OpenCL 执行模型

OpenCL 的执行模型规定了核心的执行方式. 一个 kernel 在被主机创建之前,必须预先创建一个标识索引的 N 维($1 \leqslant N \leqslant 3$)工作空间 ,如图 1 所示. 每个工作组均有一个唯一的工作组 ID,OpenCL 工作项通过全局维度索引范围来标识自身,每个工作项在该维度上的全局 ID 被定义为工作项在相应维度上的索引[26]. 工作项在一个工作组内被分配了一个唯一的局部 ID. OpenCL 对工作空间提供了全局索引和较小粒度的工作组空间. N 维工作组的维度需与 N 维工作空间的维度保持一致,工作组 ID、工作项全局 ID 和局部 ID 的维度均为 N,并且整个工作空间在每个维度上必须能等分成若干个工作区间[27-28].

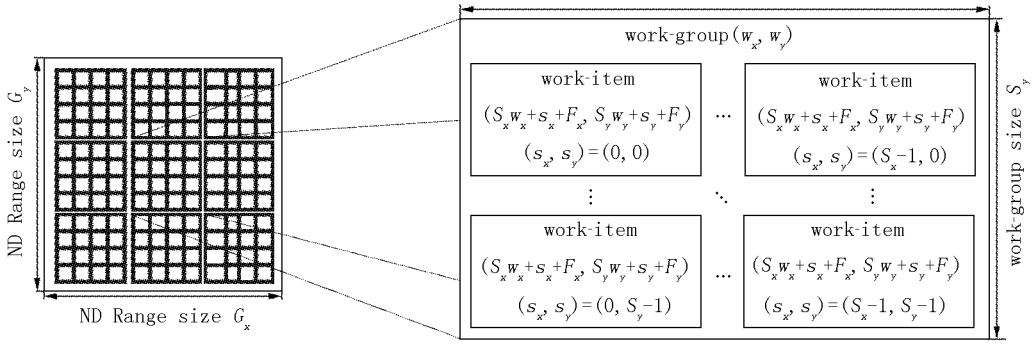


图 1　OpenCL 2 维索引空间

2.2 算法原理

将彩色图像转换成灰度图像的过程是图像灰度化处理过程. 像素的颜色是由其在彩色图像中的 R、G、B 这 3 种分量共同决定,分量的取值范围为 0 ~ 255. 而灰度图像是一种 R、G、B 分量大小相同的特殊彩色图像,其每个像素点的变化范围为 256 种[29]. 像素的灰度值是指在灰度图像中颜色的亮暗程度,在灰度图像中每个像素点不再用 R、G、B 这 3 种颜色分类表示,而是统一用一个灰度值表示. 灰度图像的层次可以用灰阶来体现,灰阶越大,图像逼真度越高. 在数字图像处理中,通常预先将各种格式的图像变换为相应的灰度图像,以利于图像的后续计算量的减少[30].

设彩色图像 C 的宽度为 w_{idth},高度为 h_{eight},则图像大小为 $T = w_{idth} \times h_{eight}$. 加权平均法图像灰度化算法取像素 i 位置的 3 个分量分别乘以各自的权值,然后再相加,得到像素 i 位置经过灰度化后的值. 彩色图像 C 经过图像灰度化处理后的图像 C' 定义为

$$C'_i = \sum_{i=0}^{T-1} (0.299 \times R_i + 0.587 \times G_i + 0.114 \times B_i),$$

其中 C'_i 表示灰度图像像素 i 位置的亮度值,R_i、G_i、B_i 分别表示彩色图像像素 i 位置的红色、绿色和蓝色分量值. 加权平均法转换的灰度图像能较好地反映原图像的亮度信息[31].

2.3 算法可并行性分析

在许多图像工程实际的应用领域中,图像灰度化往往是一个重要的处理环节,它具有密集的处理过程且时间消耗较多的特点. 然而,图像灰度化是一种逐像素处理的计算过程并具备天然的并行性,在 GPU 上进行并行处理非常适合.

在图像灰度化算法处理流程中,对于图像的每个像素点值的提取和各彩色分量加权处理均相互独立,在处理过程中无逻辑冲突,是数据并行模式的典型应用. 而且,这 2 个处理过程是整个算法运算流程中计算量最大、耗时最多的部分,并行化确有其必要. 其他处理过程计算量较小、执行次数也较少,考虑到并行化操作带来的数据 I/O 时间消耗,不适于对其并行化. 因此,仅将像素点值的提取和彩色分量加权映射到核函数. 根据分析可知,每个像素的转换均可独立完成,各自计算间无依赖性,可以把遍历像素点的转换计算转化为并行计算.

3 加权平均法图像灰度化算法的异构计算平台实现

3.1 并行算法描述

根据 OpenCL 架构的特点,设 GPU 的单指令多线程(Single Instruction Multiple Thread,SIMT)编程模型中的 1 个工作组中启用 k 个工作项,在 N 维工作空间 NDRange 中启动 $w_g = \lceil T/k \rceil$ 个工作组,每个工作项只负责处理一个像素的转换,然后把结果存储到对应的位置.

设图像像幅为 T,若用 CPU 实现,加权平均法图像灰度化串行算法是通过遍历整个图像像素数据进行计算,则时间复杂度是 $O(T)$. 然而,采用 GPU 启动的众多工作项进行并行计算,一个像素点的灰度化计算处理由相应的一个工作项负责,若系统启动 w 个工作项,则算法的时间复杂度则降为 $O(T/w)$.

算法 1 SIMT 模型上的加权平均法图像灰度化并行算法.

输入:彩色图像的像素矩阵 $\boldsymbol{s}_{rcImageData}$,

输出:灰度化后的图像像素矩阵 $\boldsymbol{d}_{esImageData}$,

Begin

In CPU:

Input color image array $\boldsymbol{s}_{rcImageData}$

In GPU:

for $i = 0$ to $w_g - 1$ par-do

for $j = 0$ to $k - 1$ par-do

$b_{aseAddress} \leftarrow i \times k + j \times 3$

$\boldsymbol{d}_{esImageData}(i) \leftarrow \boldsymbol{s}_{rcImageData}(b_{aseAddress}) \times 0.114 + \boldsymbol{s}_{rcImageData}(b_{aseAddress} + 1) \times 0.587 + \boldsymbol{s}_{rcImageData} \cdot (b_{aseAddress} + 2) \times 0.299$

end for

end for

In CPU:

Output grayscale image array $\boldsymbol{d}_{esImageData}$

End.

3.2 算法执行模式

OpenCL 架构下的加权平均法图像灰度化算法并行处理模式如图 2 所示.

实现加权平均法图像灰度化的并行计算主要步骤如下:

(i)初始化平台. 查询计算平台信息和相应平台下支持的设备列表,并选定 1 个 GPU 作为加速设备;

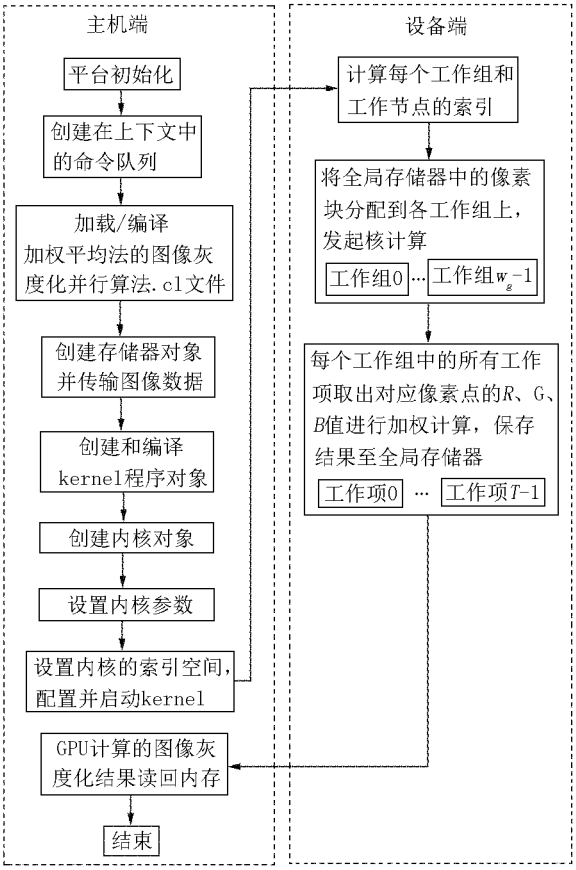


图 2　加权平均法图像灰度化并行算法执行模式

(ii)在选择的设备上创建命令队列,用以宿主机向相应设备发送命令;

(iii)加权平均法图像灰度化并行算法为了具有较好的可移植性,系统在读入源程序后,根据上下文中的设备特性采用运行时编译构建程序对象;

(iv)设置存储器对象并进行数据拷贝. buffer 存储器对象在全局存储器中首先创建,然后在命令队列中加入访问存储器的任务,最后将图像数据隐式地通过 clCreateBuffer 函数从 host 端拷贝到设备端的全局存储器中;

(v)内核对象创建. 在设备上创建一个内核对象,并将彩色图像和灰度图像的像素矩阵作为 kernel 的参数;

(vi)为内核对象设置传递的参数;

(vii)创建 kernel 函数,调度 kernel 执行,启动 GPU 设备进行并行计算;

(viii)把计算的图像灰度化像素矩阵结果传回宿主机端.

3.3　并行算法设计

(i)工作组级并行. 根据 OpenCL 执行模型的特点,充分发挥 GPU 进行大规模并行计算的能力,将彩色图像 C 进行互不重叠性分块,将整个图像矩阵进行划分为子图像块来处理. 子图像块由工作组进行数据处理,即将相应的子图像块通过工作组根据取得的 2 维位置坐标($g_{et\_group\_id}(0)$,$g_{et\_group\_id}(1)$)调度到计算单元中,如图 3 所示. 一个子图像块作为基本处理单位,工作空间中启动的众多工作组可以用来对大量的子图像块进行灰度化计算. 系统中的 GPU 研究平台为 Tesla K20,由于 GPU 中的计算单元有 15 个,32 个子图像块可以同时被一个计算单元处理. 所以,GPU 可以并行处理 480 个子图像块.

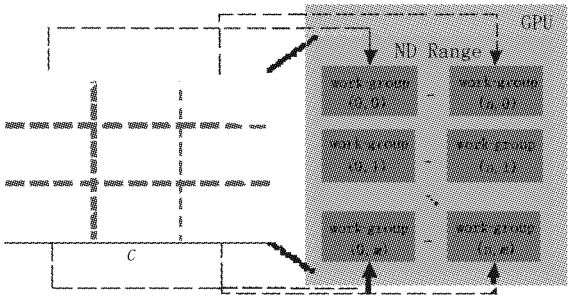


图 3　图像与工作组的分配关系

(ii)工作项级并行. 工作项负责每个子图像块中相应像素点的数据处理,即每个工作组中的每个工作项按照获得的 2 维 ID 索引位置($w_{orkItemx}$,$w_{orkItemy}$),将相应的数据元组$\langle r_i, g_i, b_i\rangle$($0 \leqslant i \leqslant T-1$)调度到处理单元中做转换运算. 具体工作项和像素点之间的位置映射关系如下所示:

$$w_{orkItemx} = d_{et\_local\_id}(1) + d_{et\_group\_id}(1) \cdot g_{et\_local\_size}(1),$$

$$w_{orkItemy} = d_{et\_local\_id}(0) + d_{et\_group\_id}(0) \cdot g_{et\_local\_size}(0).$$

加权平均法图像灰度化并行算法按照工作组的粗粒度并行和工作项的细粒度并行 2 种方式来进行并行计算. 处于同一 N 维工作空间 NDRange 内的不同工作组之间的并行属于粗粒度并行,在同一工作组内的工作项之间的并行属于细粒度并行,工作组内的多个工作项之间需要数据交换和通信. 因此,同一个工作组中的工作项可以通过 $w_{ork\_group\_barrier}(\cdot)$ 进行同步处理.

(iii)GPU 内核函数配置. 根据 OpenCL 编程模型,在 GPU 上实现的加权平均法图像灰度化计算过程以内核函数定义,核函数由工作空间中的不同工作组和工作项并行执行. GPU 中每个工作组中最大工作项数为 1 024,工作空间中工作组在 x、y、z 方向上最大尺寸为($2^{31}-1$)、65 535、65 535,工作组中最大工作项在 x、y、z 方向上最大尺寸为 1 024、1 024、64. 加权平均法图像灰度化并行算法将每个工作组内工作项在 x 方向和 y 方向上的长度为子图像块的长度$\sqrt{k}$,工作项在 x 方向和 y 方向上的总长度分别为 H_{SIZE}和 W_{SIZE},核函数的工作空间配置为

$$H_{SIZE}=(h_{eight}+\sqrt{k}-1)/\sqrt{k}\cdot\sqrt{k},$$

$$W_{SIZE}=(w_{idth}+\sqrt{k}-1)/\sqrt{k}\cdot\sqrt{k},$$

$$\text{size\_t } l_{ocalWorkSize}[\,]=\{\sqrt{k},\sqrt{k}\},$$

$$\text{size\_t } g_{lobalWorkSize}[\,]=\{H_{SIZE},W_{SIZE}\},$$

$$c_{lEngueueNDRangeKernel}\begin{pmatrix}c_{qCommandQueue},c_{kKernel},2,0,\\ g_{lobalWorkSize},l_{ocalWorkSize},0,\\ \text{Null},\&\ G_{PUExecution}\end{pmatrix}.$$

存储器访问的性能会受到不同的工作组布局的影响. 在具有足够多的工作组数量后,就可以避免因 warp 块填充不足而造成的计算资源浪费. 每个工作组中工作项的数量一般为 64 的倍数. 根据上述分析,在加权平均法图像灰度化并行算法中各参数取值为

$$k=256,W_{SIZE}\leqslant((2^{31}-1)\times2^{10}),$$

$$H_{SIZE}\leqslant((2^{16}-1)\times2^{10}).$$

3.4 性能移植优化

GPU 架构日益多样化,但统一的层次式架构模式为性能移植提供了可能. 图像灰度化并行算法通过对硬件平台关键性能参数的抽取,建立了性能参数向系统传递的机制,从而实现了不同 GPU 硬件平台间的性能移植. 2 种主要 GPU 计算平台的关键性能参数如表 1 所示.

表 1 不同 GPU 计算平台的关键性能参数

GPU	AMD Radeon HD6950	NVIDIA Tesla K20
work-group 大小	512×512×64	1 024×1 024×64
work-group 数量	$65\,535^3$	$(2^{31}-1)\times65\,535^2$
计算单元 CU 数量	22	13
每组 CU 包含处理单元数量	64	192
处理每个子图像块的 work-item 数量	512	1 024

在性能参数传递机制构建方面,利用 OpenCL 系统编译的特点,通过宏定义预先将关键性能参数定义在参数文件中. 然后,利用 OpenCL 内核编译器进行预编译,生成可加载的二进制文件. 最后调用 clGetProgramInfo 函数得到并行线程执行(Parallel Thread eXecution, PTX)文件,并将其保存到磁盘上. 系统通过调用 clCreateProgramWithBinary 直接读取缓存的 PTX 性能参数文件并生成内核. 这样,就可在不修改系统的前提下,实现了图像灰度化并行算法在 AMD Radeon HD6950 和 NVIDIA Tesla K20 2 个不同 GPU 计算平台上的性能移植.

4 实验结果及分析

4.1 测试环境和数据记录

(i)硬件平台.

平台 1:CPU 为 Intel i7 7800X 3.5GHz(六核心),系统内存为 32.0 GB. GPU 为 NVIDIA Tesla K20,含有 2 496 颗 CUDA 核心,其核心频率为 706 MHz,单精度浮点运算速度为 3.52 TFLOP/s,208 GB/s 的显存带宽,384 bits 的显存位宽,5 GB GDDR5 显存.

平台 2:CPU 为 Intel i7 7800X 3.5GHz(六核心),系统内存为 32.0 GB. GPU 为 AMD Radeon HD6950,含有 1 408 颗流处理单元,其核心频率为 810 MHz,单精度浮点运算速度 2.7 TFLOP/s,GDDR IO 频率为 5 000 MHz, 512 MB 显存,256 bits 的显存位宽.

(ii)软件平台. Microsoft Windows 10 64bits 操作系统;Matlab R2005;Microsoft Visual Studio 2019 集成环境;CUDA 环境为 Toolkit 9.0,支持 OpenCL 1.1.

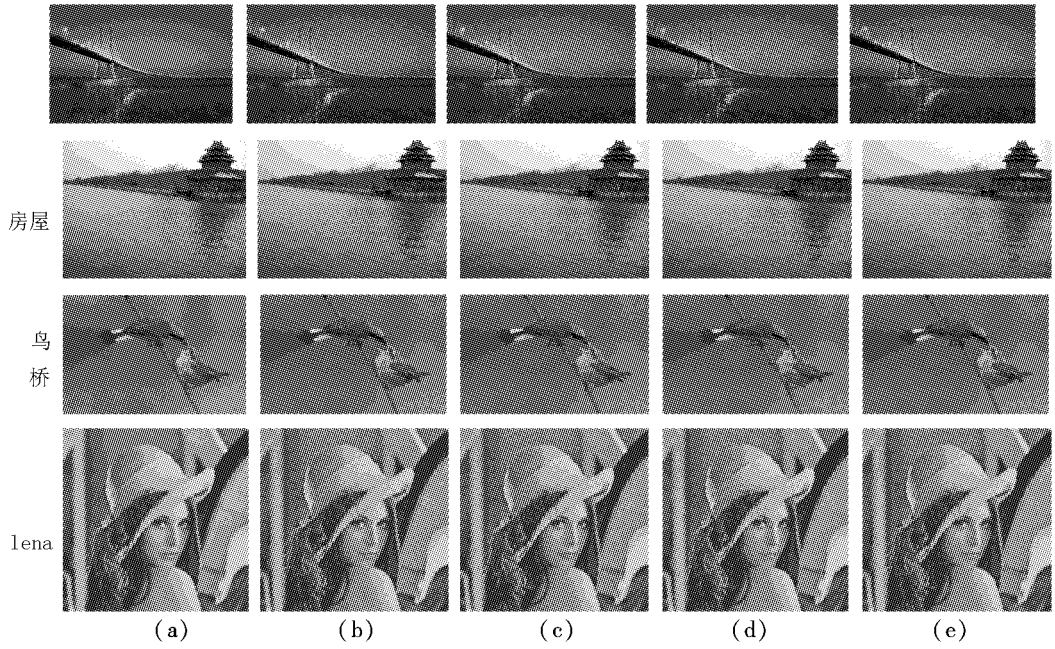
为了方便开展多组数据的对比实验,需要事先对原始图像进行预处理,经过裁剪获得图像大小分别为 512×288、640×480、1 024×1 280、2 048×1 536、3 200×2 000、4 000×3 000 和 8 767×7 842 共 7 组实验数据.

本文算法对桥、房屋、鸟和人物场景共 4 种彩色图像进行灰度化处理,效果如图 4 所示,在图 4 中的(b)、(c)、(d)和(e)分别为用串行系统、OpenMP 系统、CUDA 系统和 OpenCL 系统灰度化后的图像.

以经过预处理的 7 幅不同大小的图像进行加权平均法图像灰度化对比实验,分别用基于 CPU 的串行系统、基于 OpenMP 的并行系统、基于 CUDA 的并行系统、基于 OpenCL 的并行系统进行运行,并记录系统运行时间,结果如表 2 所示.

为了直观地进行并行算法在各种计算架构下的效率对比,算法的加速效果采用加速比来衡量. 加速比是指基于 CPU 单线程串行算法运算时间与并行算法运算时间之比. 基于 OpenMP 的并行算法运算时间与基于 NVIDIA GPU 的 OpenCL 并行算法运算时间之比被定义为相对加速比 1. 基于 CUDA 的并行算法运算时间与基于 NVIDIA GPU 的 OpenCL 并行算法运算时间之比被定义为相对加速比 2. 将 CPU 串行算法与在相应并行计算架构下并行算法运算时间的比较来反映加速比,结果如表 3 所示. 将基于 NVIDIA GPU 的 OpenCL 加速的算法与基于

OpenMP 的算法的运算时间相比来反映相对加速比 1,基于 NVIDIA GPU 的 OpenCL 加速的算法与基于 CUDA 的算法的运算时间相比来反映相对加速比 2.



注:(a)彩色图像,(b)串行处理灰度图像,(c)OpenMP 灰度图像,(d)CUDA 灰度图像,(e)OpenCL 灰度图像.

图 4　不同计算平台下加权平均法图像灰度化结果

表 2　不同计算平台下加权平均法图像灰度化算法的运算时间

图像尺寸	串行时间/ms		并行时间/ms				
	文献[24]	本文	OpenMP	文献[24]	CUDA	OpenCL (AMD)	OpenCL (NVIDIA)
512×288		1.07	0.31		0.28	0.20	0.27
640×480	492.30	2.92	0.70	15.80	0.57	0.34	0.51
1 024×1 280	2 385.40	17.29	3.97	30.90	1.30	1.18	1.16
2 048×1 536	5 724.30	30.54	6.53	51.80	2.20	1.98	1.86
3 200×2 000		48.15	9.59		3.07	2.73	2.61
4 000×3 000	19 782.50	124.25	23.71	194.70	6.81	5.97	5.74
8 767×7 842		187.65	34.43		8.40	8.07	6.94

表 3　图像灰度化并行算法性能比较

图像尺寸	加速比				相对加速比 1	相对加速比 2
	OpenMP	CUDA	OpenCL (AMD)	OpenCL (NVIDIA)		
512×288	3.45	3.82	5.35	3.96	1.15	1.04
656×613	4.17	5.12	8.59	5.73	1.37	1.12
1 920×1 200	4.36	13.30	14.65	14.91	3.42	1.12
2 560×1 600	4.68	13.88	15.42	16.42	3.51	1.18
3 200×2 000	5.02	15.68	17.64	18.45	3.67	1.18
6 580×5 432	5.24	18.25	20.81	21.65	4.13	1.19
8 767×7 842	5.45	22.34	23.25	27.04	4.96	1.21

4.2　验证算法系统的正确性

缩短图像处理时间是进行图像灰度化并行处理的目的,以获得更高图像灰度化速度.但若以损失图像质量为前提,则没有了并行化处理的作用.下面以 2 560 × 1 600 的原始图像为例进行灰度化效果分析.

(i)宏观层面结果一致性.

如图 4 中的(b)、(c)、(d)和(e)所示,当处理的像幅大小相同时,图像灰度化串行系统和各种并行系统的运行结果相同.无肉眼可辨的差异.

(ii)微观层面结果一致性.

从微观层面上看,将串行系统处理得到的图像灰度值与各种并行系统处理得到的图像灰度值用直方图表示,对比如图 5 中的(a)、(b)、(c)和(d)所示. 由图 5 可见,图像灰度化串行处理和并行处理的图像结果保持一致.

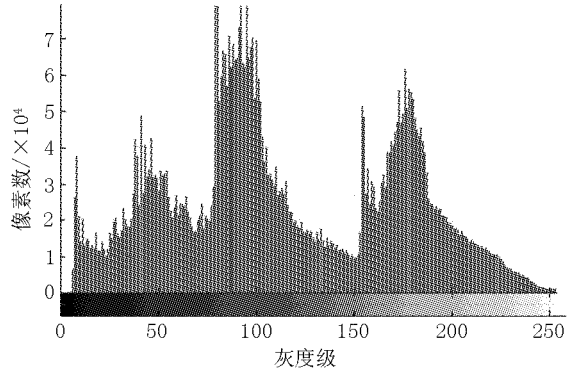


图 5 串/并行算法灰度直方图对比

4.3 并行算法性能分析

(i)不同架构下图像灰度化算法运算时间分析. 在 3 种计算平台上对不同像幅大小的实验图像进行图像灰度化处理,将 CPU 串行运行时间和在 3 种不同并行计算架构下的运行时间进行对比,如图 6 所示. 当图像像幅大小相同时,在不同并行计算平台上执行时间相对 CPU 串行执行时间均有不同程度的缩减,即均获得了加速效果. 如对于像幅大小为 8 767 ×7 842 的图像灰度化串行运算时间为 187.65 ms,在 OpenMP 计算平台下运算时间缩短为 34.43 ms,而在 CUDA 架构和 OpenCL 架构下的并行计算平台上运算时间则分别大幅缩减为 8.40 ms 和 6.94 ms.

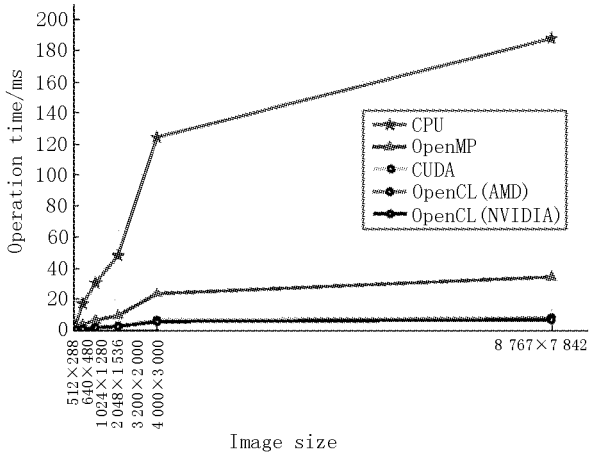


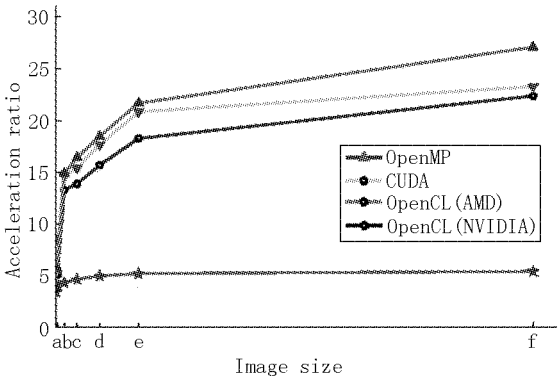
图 6 不同计算平台下加权平均法图像灰度化算法运算时间对比

在相同像幅大小下,利用 OpenMP 并行化之后的图像灰度化算法的运行时间比传统的串行算法运行时间有明显的减少. 并且在相同的线程数下随着图像规模的增大,运行时间也越来越长,基本符合线性增长的趋势.

当图像像幅较小时,在基于 OpenCL 的图像灰度化并行算法中,由于内核调度的线程工作量不足,在系统调度方面的时间消耗较大. 没有展现出 GPU 并行计算的优势,kernel 处理的时间与串行处理的时间差别不明显;随着像幅的增大,线程的工作量逐渐增加,在系统调度方面的时间消耗占比下降,提高了 GPU 的并行度,加速比提升.

下面将本文算法的整体运算时间与文献[24]进行对比. 文献[24]采用 CUDA 与 DotNet 混合编程技术实现了图像灰度处理. 根据文中提供的测试数据,当图像大小为 640 ×480、1 024 ×1 280、2 048 ×1 536和 4 000 ×3 000 时的串行算法运算时间,均分别远大于本文串行算法运算时间. 同时,文献[24]中的基于 GPU 加速的图像灰度算法处理时间也均远大于本文的基于 OpenCL 加速的图像灰度化并行算法的处理时间. 因此,本文并行算法相比文献[24]中的算法取得了更好的加速效果.

(ii)不同架构下加权平均法图像灰度化并行算法性能分析. 在 2 种对比平台上将 7 幅不同像幅大小的图像进行图像灰度化处理,将 CPU 串行运行时间和在 4 种不同并行计算架构下的运行时间进行对比. 当图像像幅大小相同时,在不同并行计算平台上执行时间相对 CPU 串行执行时间均有不同程度的缩减,即均获得了加速效果,结果如图 7 所示.



注:a. 512 × 288;b. 640 × 480;c. 1 024 × 1 280;d. 2 048 × 1 536;e. 3 200 × 2 000;f. 4 000 × 3 000;g. 8 767 × 7 842.

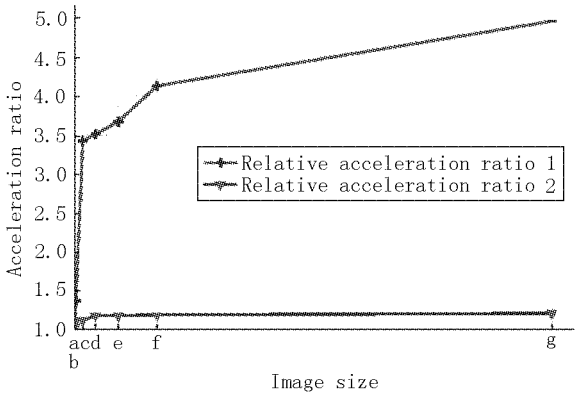
图 7 不同计算平台下加权平均法图像灰度化并行算法的加速比对比

由图 7 可见,多核 CPU 能够提高图像灰度化的运算速度,但是由于核心数有限,获得的加速比较小且增幅缓慢. 而 GPU 加速比随着像幅大小的扩大而增大,GPU 的良好性能来自于其所拥有的大量计算单元以及高吞吐量提供多点计算. 只有提供的数据量足够,充分利用 GPU 丰富的计算资源,才能充分

发挥其性能,这就是加速比随着计算规模增加而提高的原因.

由图 7 可看出,GPU 的加速效果随着像幅的增加越发明显,如对于像幅为 8 767 ×7 842 的图像灰度化运算,加速比达到了 27 倍. 然而,随着像幅大小的逐步增加,并行优化处理的性能提升倍数并不是线性增长,而是呈现出缓慢上升的趋势. 这主要是由于当像幅较大时,受总线带宽的限制,设备存储器与主机存储器间大量的数据通信耗时较多,出现了明显的通信瓶颈现象. GPU 并行计算所带来的优势被主机存储器与设备存储器之间的传输时间极大地抵消了,加速比数值增幅减缓,显示出系统整体性能的提升受到制约.

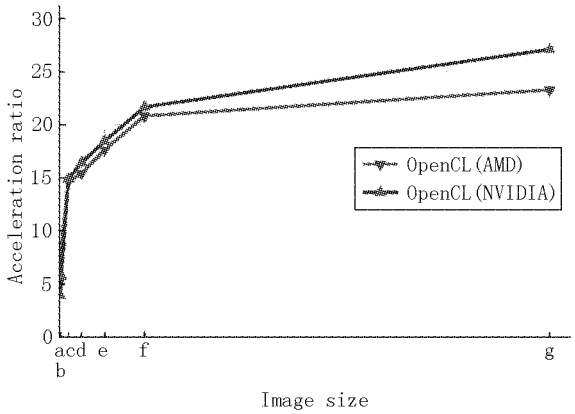
图 8 表明,优化后的 OpenCL 并行算法性能相比 CUDA 并行算法快,相对加速比 2 表明最高提高了 1.21 倍. 同时,反映出随着图像像幅的增大,2 类并行算法性能的差距有拉大的趋势. 由此可见,优化后的 OpenCL 并行算法与 CUDA 并行算法性能接近,当图像像幅较大时,OpenCL 并行算法有更大性能优势. 相对加速比 1 显示出基于 OpenCL 的加权平均法图像灰度化并行算法性能,与基于 OpenMP 平台的算法性能相比有较大的提高,最大取得了 4.96倍加速比.



注:a. 512 × 288; b. 640 × 480; c. 1 024 × 1 280; d. 2 048 × 1 536; e. 3 200 × 2 000; f. 4 000 × 3 000; g. 8 767 × 7 842.

图 8　相对加速比趋势图

(iii)基于 OpenCL 的加权平均法图像灰度化并行算法跨平台性分析. 算法源码不仅要在不同的运算平台上成功地编译和运行,而且还要求算法保持相当的性能,才能达到算法源码级别的可移植性要求. 图 9 表明,本文实现的基于 OpenCL 的加权平均法图像灰度化并行算法在 2 种不同 GPU 计算架构下处理不同像幅大小的图像均获得了可观的加速比:在平台 1 上,得到最大 27.04 倍的性能加速;在平台 2 上,得到了最大 23.25 倍的性能加速. 这不仅说明了基于 OpenCL 的加权平均法图像灰度化优化算法的有效性,而且说明了并行算法在多种异构并行计算环境下具有最大程度的兼容性和可移植性.



注:a. 512 × 288; b. 640 × 480; c. 1 024 × 1 280; d. 2 048 × 1 536; e. 3 200 × 2 000; f. 4 000 × 3 000; g. 8 767 × 7 842.

图 9　OpenCL 加速比趋势图

5　结束语

GPU 是众核处理器发展的重要方向,能够提供较高的峰值性能,但如何利用其多级并行性的结构特点、充分发挥其峰值性能,这就对并行算法的设计提出了较高的要求.

本文结合图像灰度化处理过程中体现出来的大数据量、计算密集等特征,设计实现了一种基于 OpenCL 加速的加权平均法图像灰度化并行算法. 本文分别从算法分析与设计、算法执行模式以及算法优化等方面进行了阐述,并通过实验验证相比传统的并行计算平台算法在性能上有较大的提高,且保证了处理质量. 实验结果表明,与基于 CPU 单线程串行算法、基于 OpenMP 多线程并行算法和基于 CUDA 并行算法的性能相比,基于 OpenCL 的加权平均法图像灰度化并行算法分别获得了 27.04 倍、4.96 倍和 1.21 倍的性能加速比,不仅实现了算法的高速运算,而且实现了在异质 GPU 并行计算平台间的性能移植. 本文提出的优化方法充分利用了 GPU 大量的并行计算单元和超强的计算能力,为海量图像数据高效处理提供了一条崭新的途径.

下一步的工作还有性能提升的空间:图像像素点灰度化之间没有任何依赖性,因此,考虑在 GPU 集群上采用 MPI 和 OpenCL 相结合的技术,由 MPI 完成图像块之间的并行,由每个节点上的 GPU 完成图像块内的并行,通过 GPU 集群可以使更大的像幅图像灰度化速度更快.

6 参考文献

[1] Schindler S, Bruchmann M, Bublatzky F, et al. Modulation of face- and emotion-selective ERPs by the three most common types of face image manipulations [J]. Social Cognitive and Affective Neuroscience, 2019, 14(5): 493-503.

[2] Li Dahua, Zhao Hui, Zhao Xiangfei, et al. Cucumber detection based on texture and color in greenhouse [J]. International Journal of Pattern Recognition and Artificial Intelligence, 2017, 31(8): 1-17.

[3] Howard S R, Shrestha M, Schramme J, et al. Honeybees prefer novel insect-pollinated flower shapes over bird-pollinated flower shapes [J]. Current Zoology, 2019, 65(4): 457-465.

[4] Gotz M, Cavallaro G, Geraud T, et al. Parallel computation of component trees on distributed memory machines [J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(11): 2582-2598.

[5] Rostami M J, Shahba A, Saryazdi S, et al. A novel parallel image encryption with chaotic windows based on logistic map [J]. Computers and Electrical Engineering, 2017, 62(8): 384-400.

[6] Luo Gaofeng, Zhou Rigui, Liu Xingao, et al. Fuzzy matching based on gray-scale difference for quantum images [J]. International Journal of Theoretical Physics, 2018, 57(8): 2447-2460.

[7] Dhivya R, Prakash R. Edge detection of satellite image using fuzzy logic [J]. Cluster Computing: The Journal of Networks Software Tools and Applications, 2019, 22(5): 11891-11898.

[8] Sreenilayam Sithara P, Panarin Yuri P, Vij Jagdish K, et al. Spontaneous helix formation in non-chiral bent-core liquid crystals with fast linear electro-optic effect [J]. Nature Communications, 2016, 7(5): S1-S11.

[9] Lowes S, Leaver A, Cox K, et al. Evolving imaging techniques for staging axillary lymph nodes in breast cancer [J]. Clinical Radiology, 2018, 73(4): 396-409.

[10] Jahid Tarik, Karmouni Hicham, Hmimid Abdeslam, et al. Fast computation of Charlier moments and its inverses using Clenshaw's recurrence formula for image analysis [J]. Multimedia Tools and Applications, 2019, 78(9): 12183-12201.

[11] Liu Zhao, Jing Hui, Han Xue, et al. Shear wave elastography combined with the thyroid imaging reporting and data system for malignancy risk stratification in thyroid nodules [J]. Oncotarget, 2017, 8(26): 43406-43416.

[12] Yadav, Gyan Singh, Ojha, Aparajita. Secure data hiding scheme using shape generation algorithm: a key based approach [J]. Multimedia Tools and Applications, 2018, 77(13): 16319-16345.

[13] Jin Changzhu, Nam Kweon-Ho, Paeng Dong-Guk. Asymmetric pulsation of rat carotid artery bifurcation in three-dimension observed by ultrasound imaging [J]. International Journal of Cardiovascular Imaging, 2016, 32(10): 1499-1508.

[14] Nafchi H Z, Shahkolaei A, Hedjam R, et al. CorrC2G: color to gray conversion by correlation [J]. IEEE Signal Processing Letters, 2017, 24(11): 1651-1655.

[15] Viswanathan S, Divakaran G, Soman K P. Exploring the significance of using perceptually relevant image decolorization method for scene classification [J]. Journal of Electronic Imaging, 2017, 26(6): 21-31.

[16] Rostami M J, Shahba A, Saryazdi S, et al. A novel parallel image encryption with chaotic windows based on logistic map [J]. Computers and Electrical Engineering, 2017, 62(8): 384-400.

[17] 崔宣, 丁杨, 高文, 等. 快速并行细化算法在焊缝识别中的实现 [J]. 西华大学学报: 自然科学版, 2014, 33(6): 11-15.

[18] Tan Fenglin, Li Yanxiong, Guan Mingyu. Automatic windowing for highly dynamic industrial X-ray image based on short-term energy of gray histogram [J]. Journal of Nondestructive Evaluation, 2017, 36(3): 539-552.

[19] 杨洪余, 李成明, 王小平, 等. CPU/GPU 异构环境下图像协同并行处理模型 [J]. 集成技术, 2017, 6(5): 8-18.

[20] Saleem S, Bais A, Sablatnig R, et al. Feature points for multisensor images [J]. Computers and Electrical Engineering, 2017, 62(8): 511-523.

[21] Freitas P G, Farias M C Q, Araujo A P F. Hiding color watermarks in halftone images using maximum similarity binary patterns [J]. Signal Processing: Image Communication, 2016, 48(10): 1-11.

[22] Kiermaier M, Wassermann A, Zwanzger J. New upper bounds on binary linear codes and a Z(4)-code with a better-than-linear gray image [J]. IEEE Transactions on Information Theory, 2016, 62(12): 6768-6771.

[23] 姜涛, 张云伟, 何芳. 基于流处理器的图像灰度变换并行处理研究 [J]. 电子技术应用, 2011, 37(2): 116-119.

[24] 班志华, 陈华, 刘田田, 等. CUDA 与 Dot Net 混合编程在图像灰度处理中的应用 [J]. 电脑知识与技术, 2011, 7(11): 2647-2648, 2651.

[25] 占正锋, 李戈, 张学贺, 等. 基于 CUDA 的图像预处理并行化研究 [J]. 机械与电子, 2014(7): 64-67.

[26] Jaksic Zoran, Cadenelli Nicola, Buchaca Prats David, et al. A highly parameterizable framework for conditional re-

stricted boltzmann machine based workloads accelerated with FPGAs and OpenCL [J]. Future Generation Computer Systems: The International Journal of Escience, 2020, 104(5):201-211.

[27] Singh Dhirendra Pratap, Joshi Ishan, Choudhary Jaytrilok. Survey of GPU based sorting algorithms [J]. International Journal of Parallel Programming, 2018, 46 (6): 1017-1034.

[28] Maier Georg, Pfaff Florian, Wagner Matthias, et al. Real-time multitarget tracking for sensor-based sorting a new implementation of the auction algorithm for graphics processing units [J]. Journal of Real: Time Image Processing, 2019, 16(6):2261-2272.

[29] Hajmohammadi S, Nooshabadi S, Archer G E, et al. Parallel hybrid bispectrum-multi-frame blind deconvolution image reconstruction technique [J]. Journal of Real: Time Image Processing, 2019, 16(4):919-929.

[30] Funasaka S, Nakano K, Ito Y. Fully parallelized LZW decompression for CUDA-enabled GPUs [J]. IEICE Transactions on Information and Systems, 2016, E99D (12): 2986-2994.

[31] D'Ambra P, Filippone S. A parallel generalized relaxation method for high-performance image segmentation on GPUs [J]. Journal of Computational and Applied Mathematics, 2016, 293(2):35-44.

The Study on Image Gray-Scale Parallel Algorithm Based on OpenCL

XIAO Han[1,2], GUO Baoyun[3*], LI Cailin[3], XIAO Shiyang[4]

(1. School of Information Science and Technology, Zhengzhou Normal University, Zhengzhou Henan 450044, China;
2. School of Information Engineering, Zhengzhou University, Zhengzhou Henan 450001, China;
3. School of Civil and Architectural Engineering, Shandong University of Technology, Zibo Shandong 255000, China;
4. School of Civil Engineering, Northeast Forestry University, Harbin Heilongjiang 150040, China)

Abstract: With the increase of image data amount, the computing model of the traditional single-core processor or multi-processor structure can't meet the real-time processing requirements of image gray-scale. In this paper, the parallel algorithm of image gray-scale based on Open Computing Language (OpenCL) is proposed by using the advantages of Graphic Processing Unit (GPU) in heterogeneous parallel computing. By analyzing the parallelism of weighted average image gray-scale algorithm data processing, the task is decomposed hierarchically. Two levels parallel algorithm is designed and mapped onto the CPU + GPU heterogeneous computing platform. The experimental results show that compared with the performance of the serial algorithm, multi-core CPU parallel algorithm and parallel algorithm based on Compute Unified Device Architecture (CUDA), the image gray-scale parallel algorithm obtains 27.04 times, 4.96 times and 1.21 times speedup in the NVIDIA GPU computing platform under the OpenCL architecture respectively. The validity and performance portability of the proposed parallel optimization method are verified.

Key words: image gray-scale; weighted average; Graphic Processing Unit (GPU); Open Computing Language (OpenCL); parallel algorithm

(责任编辑:冉小晓)