CPU/GPU 异构环境下图像协同并行处理模型

杨洪余[1,2] 李成明[2] 王小平[1] 姜青山[2]

[1](五邑大学 江门 529020)
[2](深圳先进技术研究院 深圳 518055)

摘 要 随着 GPU 通用计算能力的不断发展，一些新的更高效的处理技术应用到图像处理领域。目前已有一些图像处理算法移植到 GPU 中且取得了不错的加速效果，但这些算法没有充分利用 CPU/GPU 组成的异构系统中各处理单元的计算能力。文章在研究 GPU 编程模型和并行算法设计的基础上，提出了 CPU/GPU 异构环境下图像协同并行处理模型。该模型充分考虑异构系统中各处理单元的计算能力，通过图像中值滤波算法，验证了 CPU/GPU 环境下协同并行处理模型在高分辨率灰度图像处理中的有效性。实验结果表明，该模型在 CPU/GPU 异构环境下通用性较好，容易扩展到其他图像处理算法。

关键词 GPU；图像处理；协同并行处理；模型；通用性
中图分类号 TG 156 **文献标志码** A

Image Cooperative Parallel Processing Model in CPU / GPU Heterogeneous Environment

YANG Hongyu[1,2] LI Chengming[2] WANG Xiaoping[1] JIANG Qingshan[2]

[1](*Wuyi University*, *Jiangmen* 529020, *China*)
[2](*Shenzhen Institutes of Advanced Technology*, *Shenzhen* 518055, *China*)

Abstract In recent years, with the continuous development of GPU general computing power, more efficient processing technologies have been for image processing. At present, some image processing algorithms have been transplanted to GPU and have good effect in acceleration. However these algorithms do not make full use of the computing power of each processing unit in a hybrid systems composed by CPU/GPU. Based on GPU programming model and parallel algorithm design, we proposed image collaborative parallel processing model at CPU/GPU heterogeneous environment in this paper. The effectiveness of this model in high resolution grayscale image processing was verified by the image median filtering algorithm, which was based on the computing power of each processing unit in heterogeneous system. The experimental results show that the model performs well in CPU/GPU heterogeneous environment and is easy to execute other image processing algorithms.

Keywords GPU; image processing; collaborative parallel processing; model; generality

收稿日期：2017-01-16 **修回日期**：2017-04-23
基金项目：深圳市基础研究(JCYJ20150630114942277)；深圳市 2016 年技术攻关(JSGG20160229123657040)；2015 年广东省省级科技计划(2015A080804019)
作者简介：杨洪余，硕士研究生，研究方向为数据挖掘与 GPU 并行图像处理；李成明(通讯作者)，博士，助理研究员，研究方向为互联网技术，E-mail：cm.li@siat.ac.cn；王小平，副教授，硕士生导师，研究方向为生物多特征识别；姜青山，研究员，博士生导师，研究方向为数据挖掘。

1　引　言

随着摩尔定律的不断发展，图形处理器(Graphics Processing Unit，GPU)中集成的晶体管数目已经超过了中央处理器(Central Processing Unit，CPU)中集成的晶体管数目，GPU 的通用计算能力正在不断地被挖掘和应用。目前异构多核系统(一个多核处理器加 GPU)被广泛应用在计算机系统中。由于这些系统中的 GPU 具有高效的并行计算能力、高速内存带宽和友好的并行编程模型而被广泛应用于计算密集型程序。目前无论在个人电脑，还是超级计算机或者 GPU 集群中[1,2]，GPU 都作为主要的加速器件负责计算任务。罗力等[3]提出基于 CPU/GPU 集群求解偏微分方程的可扩展混合算法，将算法中的可并行部分转移到 GPU 上，利用 GPU 的并行计算能力加速算法的执行。何文婷等[4]提出 Hadoop+编程框架，将 GPU 与大数据处理框架 Hadoop 结合起来，允许用户将数据处理过程中的计算密集型任务转移到 GPU 上。许栋等[5]将 GPU 应用于浅水波运动数值模拟中，利用 GPU 的并行计算能力为单机快速实现洪水预测提供了支撑。由此可见，利用 GPU 加速计算任务已成为主流，越来越多的高性能计算机采用由 CPU 与 GPU 组成的异构系统作为主要的计算单元。

而随着图像获取技术的不断改进，图像的分辨率和质量也不断地提高，但由于数字图像处理算法的复杂度高、计算量大等，导致 CPU 平台上现有的图像处理工具已经无法满足图像处理的需求。这样容易造成图像数据的堆积，无法实时处理图像，影响图像处理的效率，最终使整个图像处理的时间增加。如何快速有效地处理图像成为了一个急需解决的问题。

本文主要根据 GPU 上的编程模型，结合通用的并行算法设计过程，提出 CPU/GPU 协同并行图像处理模型。该模型简单、易理解、容易实现。针对高分辨率灰度图像，通过图像中值滤波算法[6]进行了验证。该研究将为更多的图像处理算法移植到 CPU/GPU 异构环境下提供了理论基础。

2　GPU 并行图像处理及编程模型

GPU 在图像处理中具有易于并行化的优势，因此已有不少专家学者成功地将 GPU 的并行计算能力应用于图像处理中，并取得不错的加速效果。图像匹配技术在城市安防中起着重要的作用，因此，快速有效地进行图像匹配在实践中有着重要的意义。厉旭杰[7]将 GPU 应用于图像匹配技术中，加快了图像匹配的速度，减少了匹配时间。边缘检测是图像处理和计算机视觉，尤其是特征提取中的一个研究领域，唐斌和尤文[8]将 GPU 应用于图像边缘检测中，加快了图像边缘检测的速度，同时保留了图像的重要结构属性。反投影滤波算法凭借其可实现感兴趣区域重建的优点，近年来逐步应用在锥束 CT 中。伍绍佳等[9]将 GPU 应用于反投影滤波算法中，优化并加快了锥束 CT 的重建过程，减少了重建时间。压缩感知通过少量测量值就可以高概率重构原始信号，但重构过程需要大量的运算。苗壮等[10]将 GPU 应用于压缩感知图像恢复算法中，利用 GPU 的高速并行计算能力实现了 3 种高效的压缩感知图像恢复算法。

上述那些移植到 GPU 中的算法只是一些特定的算法，并没有形成通用的移植模型，难以推广到其他数字图像处理算法中。针对这些问题，王平等[11]提出了 GPU 并行处理模型，该模型从 GPU 体系结构上考虑图像并行处理，但并没有考虑到系统中其他处理单元的作用，造成资源的浪费。为充分利用系统中各处理单元的计算能力，Luk 等[12]提出了 Qilin 模型，通过训练阶段测量执行时间和传输时间，建立 CPU 和 GPU 的线性执行速率模型。但由于训练阶段对处理单元性能评估的不

准确性，导致线性预计的执行速率不能反映处理单元的实际执行速率，容易造成负载不均衡。

计算统一架构（Compute Unified Device Architecture，CUDA）编程模型是一种将 GPU 作为数据并行计算设备的软硬件体系[13]，如图 1 所示。CUDA 程序由主机端和设备端两部分代码组成。其中，主机端代码在 CPU 上运行；设备端代码又称为 Kernel 函数，在 GPU 上运行。在 CUDA 程序中，一个 Kernel 函数对应一个 Grid，每个 Grid 根据待处理的并行数据的需要配置不同的 Block 数量和 Thread 数量。Grid 和 Block 可以根据问题的需要配置成逻辑上的一维、二维或三维，便于数据映射到 GPU 内部处理单元上。编程模型中 Kernel 函数的执行是异步的，即在主机端调用 Kernel 函数时，该 Kernel 函数经过延迟后执行，执行完毕后通知主机端。对于异构系统中每一个 GPU，都需要一个多核 CPU 中的一个特定核对应的线程控制。对于现有异构多核系统来说，系统中 CPU 核心数远大于 GPU 的个数，因此可利用多线程技术，将一部分数据转移到 CPU 剩余核上进行处理，使系统中的所有处理单元同时处理数据，充分利用系统资源，提高处理数据的效率。从 CUDA 编程模型可以看出，CUDA 包含两个并行逻辑层，分别是 Block 层和 Thread 层。在执行时，Block 层映射到流多处理器（Streaming Multiprocessors，SM），Thread 层映射到流处理器（Streaming Processor，SP）。如何在实际应用程序中高效地开发这两个层次的并行，是 CUDA 编程与优化的关键技术之一。

针对现有的 GPU 并行处理模型的缺点，根据 CUDA 编程模型，结合通用的并行算法设计过程，本文提出 CPU/GPU 异构环境下的图像协同并行处理模型。

3 CPU/GPU 协同并行图像处理模型

3.1 图像处理并行化

在异构系统中，图像处理算法并行化设

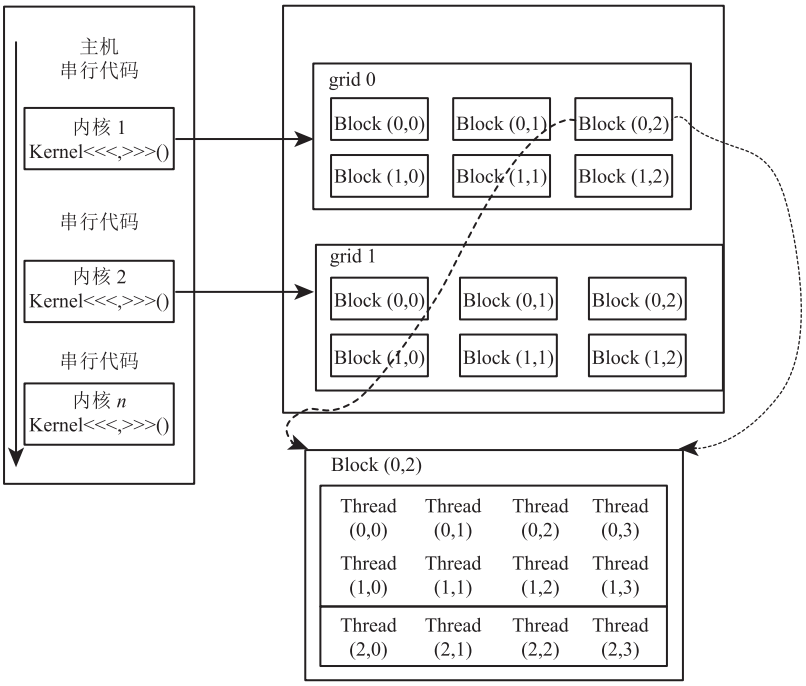


图 1 CUDA 编程模型[13]

Fig.1 CUDA programming model[13]

计遵从著名的划分通讯组合映射(Partitioning Communication Agglomeration Mapping，PCAM)设计方法学[14]。PCAM 设计方法学把并行算法设计分成 4 个阶段：(1)划分(Partitioning)，分解成小的任务，开拓并行性；(2)通讯(Communication)，确定诸任务间的数据交换，监测划分的合理性；(3)组合(Agglomeration)，依据任务的局部性，组合成更大的任务；(4)映射(Mapping)，将每个任务分配到相应的处理器上，提高并行算法的性能，在映射时要注意算法中的负载均衡问题。但异构系统中图像处理算法并行化又有自己的特点。基于 PCAM 设计方法学和 CUDA 编程模型，本文提出异构系统中图像处理算法的并行化设计方法，主要包括以下步骤：算法并行化分析、任务划分、并行粒度划分、映射到 GPU 中、负载均衡和程序的优化，具体如图 2 所示。

3.1.1 算法并行化分析

常用并行算法的设计策略主要有：串行算法的直接并行化、从问题描述开始设计并行算法、借用已有算法求解新问题[15]。其中，串行算法的直接并行化，即发掘和利用现有串行算法中的并行性，直接将串行算法改造为并行算法。由串行算法直接并行化是并行算法设计中最常用的方法之一，通常许多数值串行算法可以比较容易地转化为有效的数值并行算法。从问题描述开始设计并行算法就是从问题描述的本身出发，不考虑相应的串行算法，设计一个全新的并行算法。采用已有算法求解新问题，即找出求解问题和某个已解决问题之间的联系，改造或利用已知算法应用到求解问题上。本文是对已有的数字图像处理算法进行并行化加速，因此主要采用串行算法的直接并行化这一并行算法设计策略。

3.1.2 任务划分

并行计算中的计算任务可以分成多个部分，以便多个控制流同时处理。根据算法分析中对输出数据是否可独立并行处理，判断是否属于数据划分。

3.1.3 并行粒度划分

从 CUDA 编程模型可以看出，GPU 上的计算可看成是一个处理器上的网格计算。该网格包含两个层次，即线程块和线程块中的线程。在 GPU 具体执行中，线程块和线程分别被调度成粗粒度和细粒度并行。并行粒度的划分就是将图像并行算法中要处理的数据划分到线程块和块的线程中，从而实现线程粗、细粒度并行执行，提高 GPU 的利用率，进而使 GPU 加快数据处理。在图像处理并行算法中，常用的粗粒度划分数据方法[16]有：按行划分、按列划分和棋盘阵列划分。其中，按行划分或按列划分，就是每一个线程块分别处理图像中的一行或一列，这种划分方式只需要 blockId.x 和 threadId.x 这两个索引坐标就可对相应的像素数据进行定位。Kernel 函数配置时只需要把 gridDim.x 和 blockDim.x 设置为相应的数据，其余维数设置为 1 即可。棋盘阵列划分方式是采用小矩形块的方式划分图像，将整个图像划分成若干个矩形块，每一个小矩形块由对应的线程块进行处理。棋盘阵列划分方式需要 blockId.x、blockId.y、threadId.x、threadId.y 这 4 个索引坐标即可对相应的像素数据进行定位，Kernel 函数配置时只需要把 gridDim.x、gridDim.y 和 blockDim.x、blockDim.y 设置为相应的数据，其余维数设置为 1 即可。确定粗粒度划分的方式后，就可以根据

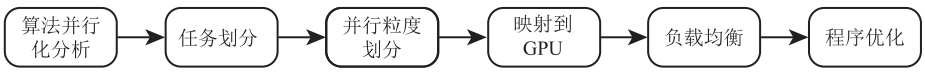


图 2　CPU/GPU 协同并行图像处理过程

Fig. 2　Collaborative parallel image processing using CPU/GPU

待处理的图像数据大小计算出线程块的个数。由GPU 线程调度的特点可知，一个线程块中的线程数量应为 32 的整数倍，且不应超过 GPU 硬件的限制，按照这样的方式组织的线程可以提高 GPU 的资源利用率。

3.1.4 映射到 GPU

根据上一步中的并行粒度划分方式，将图像处理并行算法映射到 GPU 的众核架构上，通过单指令多线程(Single Instruction Multiple Threads)并行方式实现数据的并行处理。主要步骤为传入→运算→传出，具体如下：

(1) 在 GPU 上开辟显存空间，将图像数据从电脑主存中通过总线传输到 GPU 显存中；

(2) 在主机中设置执行配置，即线程块的维数和线程的维数，然后启动核函数；

(3) 等待 Kernel 函数执行结果，并将结果数据从显存中拷回主机内存中。

3.1.5 负载均衡

负载均衡是指通过调整计算任务在各个处理单元上的分配，使相应的计算任务在对应的处理单元上的完成时间近似相等，以充分发挥系统内处理单元的计算能力，提高并行计算的效率。在 CPU/GPU 组成的异构系统中，这意味着 CPU 和 GPU 近似同时结束计算。Zhong 等[17]详细讨论了在 CPU/GPU 组成的异构系统中，由于资源访问冲突造成 CPU 处理单元性能下降的问题。为解决负载均衡，CPU/GPU 协同并行图像处理模型中采用根据计算能力分配数据的负载均衡方案。其具体方法是分别测量未优化的 CPU 版本执行时间和未优化的 GPU 版本执行时间，根据它们的执行时间比率，确定理想的数据分配比例。同时，CPU/GPU 协同并行图像处理模型考虑到 GPU 的高效并行计算能力，CPU 线程处理的数据受操作系统调度等问题的影响，应将分配到 CPU 线程上的数据减少一定的比例。具体算法如下：设总数据量为 S，处理同一任务所需要的 CPU 和 GPU 测量时间分别为 T_{CPU} 和 T_{GPU}，CPU 和 GPU 处理速度分别为 V_{CPU}和 V_{GPU}，分配到 CPU 上的数据为 S_{CPU}，分配到 GPU 上的数据为 S_{GPU}，考虑到影响因素时分配到 CPU 上的数据为 $S_{R\text{-}CPU}$。令 CPU 和 GPU 执行时间相等可得如下公式[18]：

$$S_{CPU}+S_{GPU}=S \tag{1}$$

$$\frac{S_{CPU}}{V_{CPU}}=\frac{S_{GPU}}{V_{GPU}} \tag{2}$$

$$\alpha=\frac{T_{CPU}}{T_{GPU}} \tag{3}$$

由以上公式可得

$$S_{CPU}=\frac{S}{1+\alpha} \tag{4}$$

$$S_{R\text{-}CPU}=S_{CPU}\times\beta \tag{5}$$

其中，β 为调节因子，$0.5\leqslant\beta<1$，本文实验取 $\beta=0.9$。在以上公式中，(1)式表示 CPU 和 GPU 要处理的数据总量为 S；(2)式等号左边表示分配给 CPU 上的数据处理时间，等号右边表示分配给 GPU 上的数据处理时间；(3)式表示当处理数据大小为 S 时 CPU 与 GPU 的执行时间的比例；(4)式表示不考虑 CPU 线程调度等因素时应分配给 CPU 上的数据；(5)式表示当考虑到 CPU 线程受操作系统调度时造成的部分开销时应分配给 CPU 上要处理的数据。

3.1.6 程序优化

程序优化的主要作用是在保证程序运行正确的情况下，加快程序的运行速度。对于本模型来说，程序优化主要是指 GPU 代码部分的优化。对于 GPU 代码部分程序的优化[19]，主要考虑的是访存优化，减少分支处理，提高计算密集度。这是由 GPU 的体系结构决定的。由于 GPU 大量晶体管用于计算，只有极少量的缓存甚至没有缓存，对分支处理没有 CPU 那么复杂的预测机制，容易浪费大量的 GPU 执行时间，造成性能下降。因此本模型的优化方法主要有：发生数据

复用时，尽可能地使用共享存储器；访问全局内存时尽量做到合并访存；在程序中尽量避免分支处理；在涉及到图像插值运算时，考虑使用纹理存储。

3.2　协同并行图像处理模型的验证

为验证模型的有效性，本文采用图像滤波算法中常用的中值滤波算法进行验证。验证过程严格按照 3.1 节提出的步骤进行，即算法并行化分析、任务划分、并行粒度划分、映射到 GPU 中、负载均衡和程序优化这 6 个步骤，具体步骤如下：

(1) 中值滤波算法并行化分析。图像中值滤波算法的核心思想是提取图像中每个像素周围的若干个像素点进行排序，得到的中间值即为本像素的滤波结果。由此可以看出，图像处理结果中各个像素之间的操作没有相关性，是典型的输出数据并行化，因此该算法可以并行化。

(2) 任务划分。根据第一步算法分析可知，中值滤波属于数据划分的方式。

(3) 并行粒度划分。本文实验采用棋盘阵列划分方式，其中选择的每个小矩形大小为 16×16，这样每个小矩形中共有 256 个像素，在程序映射到硬件上时恰好满足 32 的倍数，提高了 GPU 的资源利用率。

(4) 映射到 GPU。本验证算法包括以下步骤和部分核心代码：

① cudaMalloc((void**)&dev_input,ImageSize);//分配输入图像的显存，cudaMalloc((void**)&dev_output,ImageSize);//分配输出图像的显存；

② cudaMemcpy(dev_input,input_image,ImageSize,cudaMemcpyHostToDevice);//将图像数据从主存中传输到 GPU 显存中；

③Dim3 threadPerBlock(16,16,1);//设置线程块的维数，Dim3 numBlocks((width+15)/16,(height+15)/16,1);//设置网格的维数；

④ Kernel<<<numBlocks,threadPerBlock>>>(dev_input,dev_output,width,height);//根据设置的参数启动核函数；

⑤cudaMemcpy(out_put,dev_output,ImageSize,cudaMemcpyDeviceToHost);//等待核函数中图像处理结束，把数据拷贝到主机内存中；

⑥cudaFree(dev_input);//释放分配的输入图像显存，cudaFree(dev_output);//释放分配的输出图像的显存。

(5) 负载均衡。为充分利用系统中各处理单元的计算能力，提高数据处理效率，采用公式 (5) 中的方法进行数据分配。

(6) 程序优化。采用 16×16 棋盘阵列划分方式划分图像数据时，每一个块中的像素数据会被其自身及周围的线程使用，存在着数据复用现象。这时可以考虑在线程块中使用共享存储器来优化该部分复用数据，减少全局内存的读写，提高计算访存比。

4　实验结果及分析

为验证 CPU/GPU 协同并行处理模型的有效性，实验采用 8 位灰度 bmp 图片格式进行验证。对于不是 bmp 格式的图片，实验采用 MATLAB 软件中相应的函数进行格式转换，以满足 8 位灰度 bmp 图片格式的要求。

4.1　实验软硬件环境

本文实验所采用的软硬件环境如表 1 所示。

表 1　实验软硬件环境

Table 1　Experimental software and hardware environment

软硬件名称	参数描述
CPU	Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz
GPU	NVIDIA Quadro K620
CPU 代码编译器	Visual Studio 2013
GPU 代码编译器	NVCC8.0
操作系统	Windows10 专业版
内存	主存 16 GB，显存 2 GB

4.2 实验评价指标与结果

在 GPU 并行计算中，常用的评价指标为加速比，其定义为 CPU 串行算法运行时间与 GPU 并行算法运行时间之比[20]，加速比越大代表程序的运行速度越快。本文实验同样采用加速比作为评价指标。为验证本文模型的有效性，对不同类型的图像数据，使用 CPU 串行算法[20]、GPU 并行处理[11]、Qilin[12]模型和 CPU/GPU 协同并行模型分别进行图像处理，通过对处理时间进行计时的结果做比较分析。

本实验的数据有三个组成部分。其中第一组数据为系统随机产生的 6 张图片，随机产生的照片为算法根据像素的两个坐标值随机产生位于这两个值之间的数据并对 256 取余得到该点的像素值。这 6 张图片的分辨率从小到大依次为 256×256、512×512、1 024×1 024、2 048×2 048、4 096×4 096、8 192×8 192，其对应的图片数据如图 3 所示。

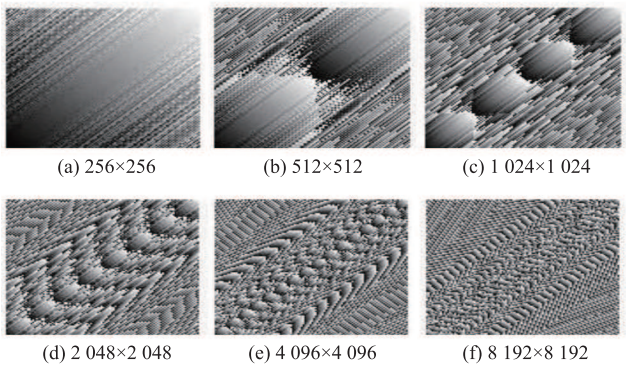


图 3 随机产生的原始图像

Fig. 3 Randomly generated raw image

首先，采用第一组数据作为示例数据，实验结果如表 2 和图 4 所示。从表 2 和图 4 可以看出，在 GPU 参与的情况下，图像处理的速度明显加快，尤其是随着图像分辨率的提高，效果更明显。这是因为 GPU 有大量并行线程可以并行处理图像像素，且各个像素的处理是互不影响的关系。

在图像分辨率较小时，根据计算能力分配一部分数据在 CPU 上运行时，可以发现 CPU/GPU 协同并行运算的加速比小于 GPU 并行处理的加速比。这说明在数据规模较小时，GPU 的并行计算能力没有完全发挥出来。在图像分辨率较大时，根据计算能力将一部分数据分配到 CPU 上，可以发现 CPU/GPU 协同并行运算的加速比要大于 GPU 并行处理的加速比。这说明在数据量较大时，GPU 的计算能力可以比较充分地发挥出来，按照 CPU/GPU 的计算能力分配数据可以达到预期的效果。Qilin 模型在训练阶段对 GPU 性能估计的不足，使得采用线性预计的执行速率不能反映 GPU 的实际执行速率，因此分配过多的数据给 CPU 处理，从而导致负载不均衡。从表 2 可以看出，本文提出的模型比 Qilin 模型效果好。

表 2 多种算法运行时间对比

Table 2 Comparison of multiple algorithms running time

图像大小	算法运行时间（ms）			
	CPU 串行	GPU 并行	Qilin 模型	CPU/GPU
256	22.32	1.65	2.01	1.98
512	90.10	5.54	5.89	5.85
1 024	366.84	18.89	20.07	19.28
2 048	1 654.59	72.84	70.89	69.51
4 096	7 208.61	282.13	277.05	273.49
8 192	29 628.10	1 123.24	1 103.13	1 087.47

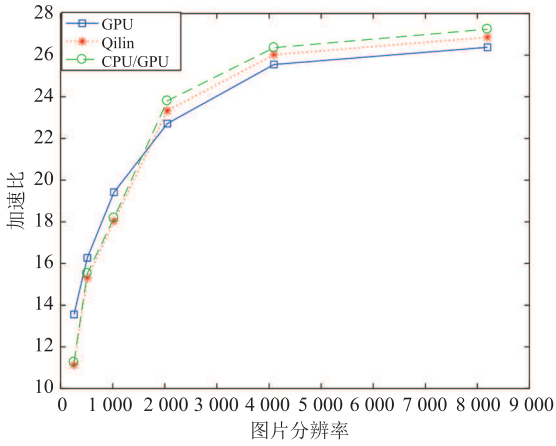


图 4 多种算法加速比曲线对比图

Fig. 4 Comparison of speedup curves of multiple algorithms

为了验证上述实验结论的正确性，我们采用分辨率比较大的图像数据来验证该结论。由表 2 和图 4 可知，当图像分辨率大于 1 024×1 024 时，CPU/GPU 协同并行运算加速比大于 GPU 并行处理的加速比。为此本文采用第二组数据来验证该结论。

第二组数据来自图像质量参考评价数据库 LIVEMD[21]，从该数据库中随机选择 15 张分辨率均为 1 280×720，位深为 24 位的彩色 bmp 图片。为满足实验要求，将彩色图转换为 8 位灰度 bmp 图进行实验，即图片分辨率为 1 280×720 的灰度 bmp 图像，结果如图 5 所示。

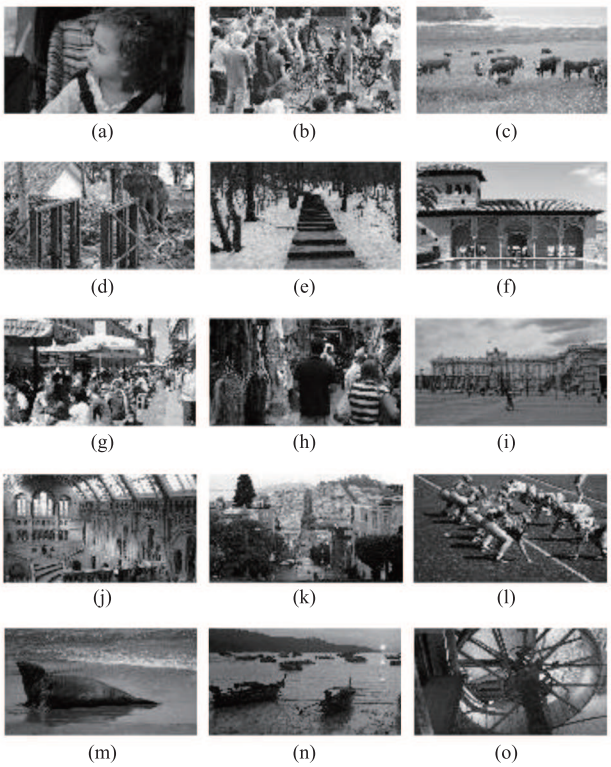


图 5　LIVEMD 参考图像

Fig. 5　Reference images in LIVEMD

将上述 15 张图片分别采用 CPU 串行算法、GPU 并行处理、Qilin 模型、CPU/GPU 协同并行模型进行图像处理，对处理时间进行计时，实验结果如表 3 和图 6(a)所示。从表 3 和图 6(a)可以看到，当灰度图像分辨率为 1 280×720 时，CPU/GPU 协同并行运行的时间与 GPU 单独运行时间相差不大，但 CPU/GPU 协同并行运行的时间有时会大于 GPU 并行处理的运行时间。造成这一现象的主要原因是灰色图像分辨率不够大，导致 GPU 的计算能力未能正确评估。

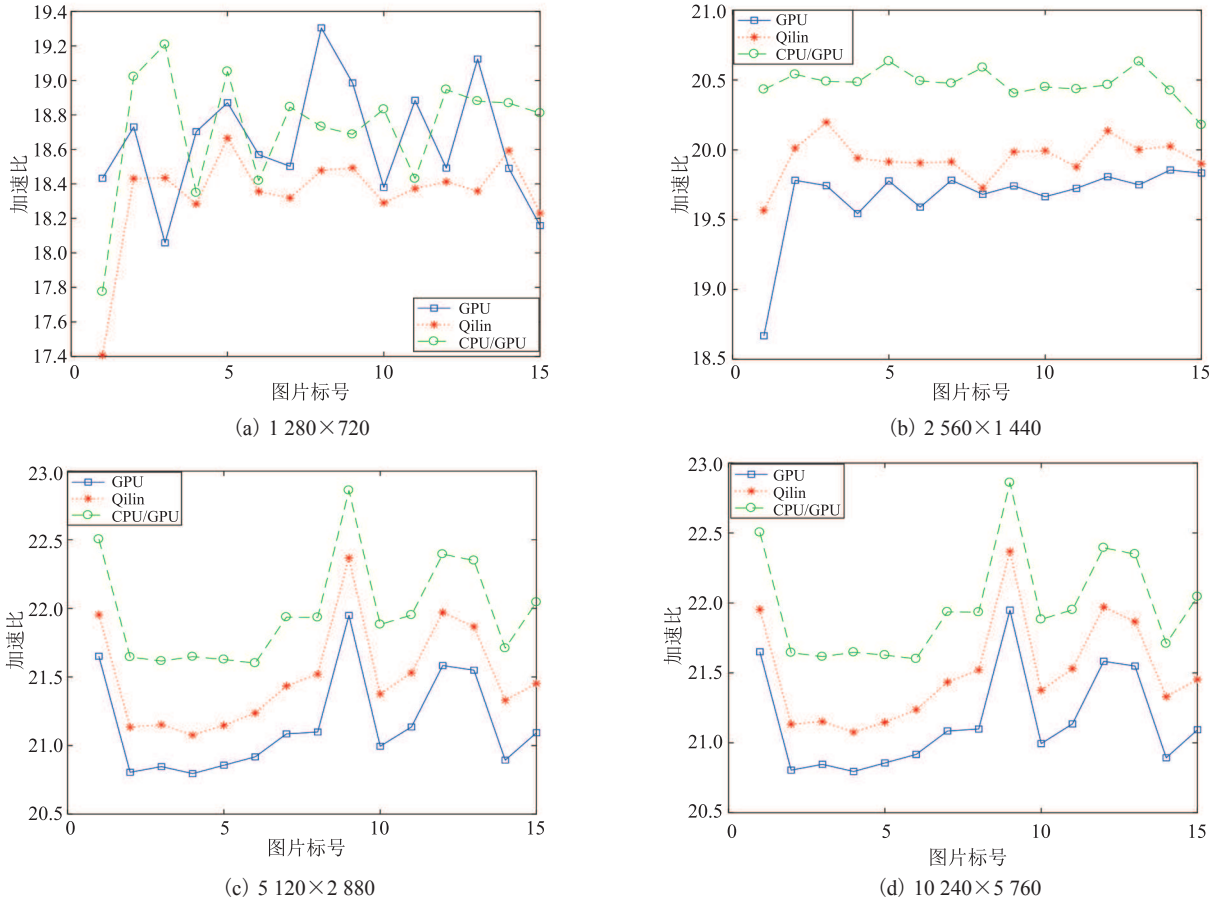
表 3　多种算法运行时间对比

Table 3　Comparison of multiple algorithms running time

图像	算法运行时间（ms）			
	CPU 串行	GPU 并行	Qilin 模型	CPU/GPU
1	310.37	15.91	17.14	16.82
2	310.28	16.57	16.84	16.32
3	310.78	17.21	16.86	16.18
4	310.85	16.62	16.64	16.94
5	309.69	16.41	16.59	16.26
6	309.14	16.65	16.86	16.78
7	309.35	16.70	16.87	16.40
8	309.68	16.03	16.74	16.51
9	309.18	16.31	18.11	17.74
10	309.16	16.82	16.91	16.57
11	309.34	16.80	17.13	17.02
12	309.69	16.82	17.03	16.37
13	308.80	16.15	16.85	16.33
14	309.23	16.73	16.58	16.39
15	309.45	17.11	17.03	16.42

为此，我们将上述随机选择的 15 张图片用图像插值算法中的最近邻插值算法进行图像放大，具体操作采用 MATLAB 软件中的 imresize 函数分别将图片扩大 2 倍、4 倍和 8 倍，之后进行实验，结果如图 6(b)～6(d)所示。

从图 6(b)～6(d)可看出，当图像分辨率大于或等于 2 560×1 440 时，CPU/GPU 协同并行运算加速比大于 GPU 并行处理的加速比，且 Qilin 模型的加速比大于 GPU 并行处理的加速比，但小于 CPU/GPU 协同并行运算加速比。考虑到通常的图片分辨率比较小，需要使用插值算法才能将图像扩大到相应的分辨率，这种情况可能对真实的图像处理有差异。因此，需要采用较大的真实图像来做实验验证，此时采用第三组数据进行实验。

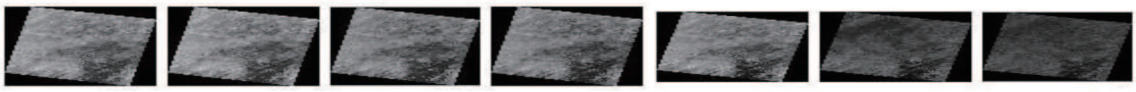


(a) 1 280×720 (b) 2 560×1 440

(c) 5 120×2 880 (d) 10 240×5 760

图 6 多种算法加速比曲线对比图

Fig. 6 Comparison of speedup curves of multiple algorithms



(a) (b) (c) (d) (e) (f) (g)

图 7 卫星遥感图像

Fig. 7 Satellite remote sensing image

第三组实验数据来自中国科学院计算机网络信息中心地理空间数据云平台[22]。具体数据来源于卫星标号为 Landsat8 OLI-TIRS，行政范围为广东省深圳市南山区，时间范围为 2016 年 11 月 1 日到 2016 年 12 月 25 日，数据标识为 LC81220442016326LGN00，图像分辨率大小为 7 531×7 711 的 TIFF 格式图片。采用 MATLAB 中的转换函数，首先利用 mat2gray 函数实现图像的归一化操作，然后再转换为 0～255 的 8 位 bmp 图片格式，结果如图 7 所示。相应的算法加速比如图 8 所示。

从图 8 可以看出，当灰度图像分辨率为 7 531×7 711 时，CPU/GPU 协同并行处理的加速比大于 GPU 并行处理的加速比，且 Qilin 模型的加速比大于 GPU 并行处理的加速比，但小于 CPU/GPU 协同并行运算加速比。

通过前面几组实验，验证了根据计算能力分配数据的处理结果差异。当图像分辨率较低时，

CPU/GPU 协同并行图像处理没有明显的优势，这时仅需利用 GPU 并行处理图像。当图像分辨率较高时，CPU/GPU 协同进行图像处理具有明显的优势，也验证了根据计算能力分配数据的负载均衡方案相比 Qilin 模型线性预计的负载均衡方案更有效。

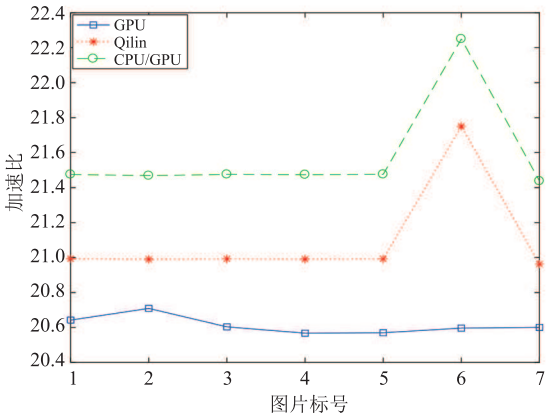


图 8　多种算法在卫星遥感图像数据集加速比曲线对比图

Fig. 8　Comparison of multiple algorithms in the acceleration curve of satellite remote sensing image data

5　结　论

本文结合并行算法设计流程及 CUDA 编程模型，提出了 CPU/GPU 协同并行图像处理模型。该模型充分利用异构系统中各处理单元的计算能力，并通过图像中值滤波算法，验证了 CPU/GPU 协同并行处理高分辨率灰色图像的优势及模型的有效性。该模型简单易懂、通用性强，且负载均衡代价少，为更多的图像处理算法移植到 CPU/GPU 异构环境下提供了一定的理论基础。

参 考 文 献

[1] Expósito RR, Taboada GL, Ramos S, et al. General-purpose computation on GPUs for high performance cloud computing [J]. Concurrency & Computation Practice & Experience, 2013, 25(12): 1628-1642.

[2] Showerman M, Enos J, Steffen C, et al. EcoG: a power-efficient GPU cluster architecture for scientific computing [J]. Computing in Science & Engineering, 2011, 13(2): 83-87.

[3] 罗力, 杨超, 赵宇波, 等. CPU/GPU 集群上求解偏微分方程的可扩展混合算法 [J]. 集成技术, 2012, 1(1): 84-88.

[4] 何文婷, 崔慧敏, 冯晓兵. 异构大数据编程环境 Hadoop+ [J]. 集成技术, 2016, 5(3): 60-71.

[5] 许栋, 徐彬, PAyet D, 等. 基于 GPU 并行计算的浅水波运动数值模拟 [J]. 计算力学学报, 2016, 33(1): 113-120.

[6] Wikipedia. Median_filter [EB/OL]. [2016-12-15]. https://en.wikipedia.org/wiki/Median_filter.

[7] 厉旭杰. GPU 加速的图像匹配技术 [J]. 计算机工程与应用, 2012, 48(2): 173-176.

[8] 唐斌, 龙文. 基于 GPU+CPU 的 CANNY 算子快速实现 [J]. 液晶与显示, 2016, 31(7): 714-720.

[9] 伍绍佳, 陈皓, 廖丽, 等. BPF 重建算法的 CUDA 并行实现 [J]. 集成技术, 2014, 3(5): 61-68.

[10] 苗壮, 隋冬, 李棚艳, 等. 采用 GPU 加速的压缩感知图像恢复算法 [J]. 微电子学与计算机, 2016, 33(12): 125-129.

[11] 王平, 全吉成, 王宏伟. GPU 图像并行处理设计模型研究 [J]. 舰船电子工程, 2016, 36(11): 74-76.

[12] Luk CK, Hong S, Kim H. Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping [C] // IEEE/ACM International Symposium on Microarchitecture, 2009: 45-55.

[13] NVIDIA. CUDA Programming Guide [DB/OL]. [2016-12-15]. http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[14] 陈国良. 并行计算: 结构•算法•编程 [M]. 北京: 高等教育出版社, 2003.

[15] 罗贵章, 陈忠伟. 并行算法综述 [J]. 计算机光盘软

件与应用, 2013(15): 51-52.

[16] 肖汉. 基于 CPU+GPU 的影像匹配高效能异构并行计算研究 [D]. 武汉: 武汉大学, 2011.

[17] Zhong Z, Rychkov V, Lastovetsky A. Data partitioning on multicore and multi-GPU platforms using functional performance models [J]. IEEE Transactions on Computers, 2015, 64(9): 2506-2518.

[18] Huang W, Yu LC, Ye MJ, et al. A CPU-GPGPU scheduler based on data transmission bandwidth of workload [C] // Proceedings of the 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies, 2012: 610-613.

[19] 李繁. 基于 GPU 的高性能并行优化算法研究 [D]. 大连: 大连理工大学, 2014.

[20] 黄文慧. 图像处理并行编程方法的研究与应用 [D]. 广州: 华南理工大学, 2012.

[21] Jayaraman D, Mittal A, Moorthy AK, et al. Objective quality assessment of multiply distorted images [C] // Signals, Systems and Computers, 2012: 1693-1697.

[22] 中国科学院计算机网络信息中心. 地理空间数据云 [EB/OL]. [2016-12-19]. http://www.gscloud.cn.