基于 OpenCL 的 Viola-Jones 人脸检测算法性能优化研究

贾海鹏 张云泉 袁 良 李士刚
(中国科学院计算技术研究所计算机体系结构国家重点实验室 北京 100190)

摘 要 Viola-Jones 人脸检测算法是最为成功的可实用的人脸检测算法之一. 然而,随着该算法所在领域数据处理规模的不断扩大,现有算法的性能已经越来越无法满足日益增长的交互性与实时性要求. 使用 GPU 计算平台提升该算法性能,以满足日益增长的实时性要求已经成为研究热点. 然而,该算法在对 GPU 的实现和优化中,存在线程间负载不均衡的非规则特性,如果仅使用传统的优化方法,则难以在 GPU 计算平台上达到较高性能. 针对此种情况,该文构建了针对此类算法的并行优化框架,通过 Uberkernel、粗粒度并行、Persistent Thread、线程与数据的动态映射、全局及本地队列等优化方法的应用,突破了负载不均衡非规则特性导致的性能瓶颈,大幅提高了人脸检测算法在 GPU 计算平台上的性能. 同时,该文通过对不同 GPU 计算平台关键性能参数的定义、抽取和传递,实现了该算法在不同 GPU 计算平台间的性能移植. 实验结果表明,与 OpenCV2.4 中经过高度优化的 CPU 版本在 Intel Xeon X5550 CPU 上的性能相比,优化后的算法在 AMD HD7970 和 NVIDIA GTX680 两个不同 GPU 计算平台上分别达到了 11.24～20.27 和 9.24～17.62 倍的加速比,不仅实现了高性能,而且实现了在不同 GPU 计算平台间的性能移植.

关键词 OpenCL;负载不均衡;任务队列;线程与任务动态映射;性能移植
中图法分类号 TP302 **DOI 号** 10.11897/SP.J.1016.2016.01775

Research of Viola-Jones Face Detection Algorithm Performance Optimization Based on OpenCL

JIA Hai-Peng ZHANG Yun-Quan YUAN Liang LI Shi-Gang
(*Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)

Abstract Viola-Jones face detection algorithm is one of the most successful and functional face detection algorithm. However, with the continuous expansion of the processing data, the performance of existing algorithm has become increasingly unable to meet its growing interactive and real-time requirements. Improving the algorithm performance through using GPU computing platform, so as to meet the growing requirements of real-time has become a hot topic. However, face detection algorithm exposes irregular feature of workload imbalance among threads when ported to GPUs. It is hard to obtain high performance if only using the conventional optimization methods. In this paper, we present an OpenCL-implementation of Viola-Jones face detection algorithm with high performance on GPUs through five main techniques: kernel merge, coarse-grained parallelism, persistent threads, dynamic mapping between thread and task and global queues. Furthermore, this paper also achieves performance portability between different GPU computing platforms by defining, extracting and delivering key performance parameters of hardware. We also demonstrate the high performance of our implementation by comparing it with a

收稿日期:2015-03-18;在线出版日期:2015-07-23. 本课题得到国家自然科学基金(61133005,61272136,61521092,61402441)资助.
贾海鹏,男,1983 年生,博士,助理研究员,中国计算机学会(CCF)会员,主要研究方向为高性能计算、众核编程方法、面向众核平台的关键优化技术研究. E-mail: jiahaipeng@ict.ac.cn.com. **张云泉**,男,1973 年生,博士,研究员,中国计算机学会(CCF)会员,主要研究领域为高性能计算及并行数值软件、并行计算模型. **袁 良**,男,1984 年生,博士,助理研究员,中国计算机学会(CCF)会员,主要研究方向为并行计算模型. **李士刚**,男,1986 年生,博士,助理研究员,中国计算机学会(CCF)会员,主要研究方向为大规模并行算法研究与设计.

well-optimized CPU version from OpenCV library. Experiment results show that the performance speedup reaches up to 11.24—20.27 times and 9.24—17.62 times on AMD HD7970 and NVIDIA GTX680 GPU respectively, not only achieves high performance but also achieves performance portability among different GPU computing platforms.

Keywords OpenCL; workload imbalance; task queue; dynamic mapping; performance portability

1 引 言

Viola-Jones 人脸检测算法由 Viola 和 Jones[1] 于 2001 年提出,是最为成功且在一定程度上满足实时要求的可实用人脸检测算法,并在执法监控、安防系统和娱乐等领域有着广泛应用.然而,随着该算法所在领域需要同时处理的数据规模的不断扩大以及图片分辨率的不断提高,现有算法的性能已经越来越无法满足其日益增长的交互性与实时性要求,提高该算法性能的需求日益迫切.与此同时,随着 GPU 通用计算的发展,特别是 GPU 计算能力与可编程性的不断增强,GPU 受到了越来越多应用开发人员的青睐,利用 GPU 计算平台实现对应用程序的加速已成为提高程序性能的主要模式.使用 GPU 计算平台提升人脸检测算法的性能,以满足日益增长的实时性要求已经成为研究热点.本文将对 Viola-Jones 人脸检测算法在 GPU 计算平台上实现和优化的关键技术和方法进行深入研究.同时,针对当前 GPU 架构日益多样化以及人脸检测算法应用场景不确定的特点,本文进行该算法在不同 GPU 计算平台间的性能移植研究.

GPU 计算平台具有大规模细粒度并行的硬件架构特征,这对于 Viola-Jones 人脸检测算法来说是一把双刃剑:一方面,Viola-Jones 人脸检测算法具有良好的并行性,非常适合 GPU 大规模并行的架构特点;另一方面,在 Viola-Jones 人脸检测算法的 GPU 实现中,线程间会产生严重的负载不均衡现象,它将导致算法性能的急剧降低,这无疑是 GPU 这种细粒度并行处理器的梦魇.不幸的是,传统 GPU 编程一般采用静态编程模式,即在 GPU kernel 启动之前,线程和数据的映射就已经确定,线程被 GPU 硬件静态顺序调度执行.这种编程模式虽然对任务规则的数据级并行算法具有良好的适用性,但是对于像 Viola-Jones 人脸检测算法这样具有线程间负载不均衡非规则特性的算法,该编程模式会产生严重的性能瓶颈,严重制约算法性能.因此,要提高 Viola-Jones 人脸检测算法在 GPU 计算平台上的性能,必须研究新的优化方法和策略,突破由于线程间负载不均衡导致的性能瓶颈,实现算法到硬件架构的良好映射,从而有效地提高算法在 GPU 计算平台上的性能.

对此,本文提出了一个面向 GPU 计算平台,针对具有负载不均衡特性算法的性能优化框架.通过 Uberkernel、粗粒度并行、Persistent Thread、线程与数据的动态映射、全局及本地队列等优化方法的应用,突破 Viola-Jones 人脸检测算法在 GPU 计算平台的实现中由于负载不均衡导致的性能瓶颈,从而完成了该算法在 GPU 计算平台上的实现和优化.本文提出的并行优化框架不仅大大提高了人脸检测算法的性能,而且也能够在一定程度上指导类似算法在 GPU 上的移植和优化,具有一定的普适性.同时,本文还针对当前 GPU 架构日益多样化以及人脸检测算法应用场景不确定的特点,总结不同 GPU 平台架构的异同,通过关键性能参数的定义、抽取和传递,实现了人脸检测算法在不同 GPU 计算平台间的性能移植.

实验结果表明:优化后的算法相对于 OpenCV① 库中高度优化的 CPU 版本在 Intel Xeon X5550 上的性能,在 AMD HD7970 和 NVIDIA GTX680 两个 GPU 计算平台上,分别达到了 11.24～20.27 和 9.24～17.62 倍的加速比.它不仅实现了高性能,而且实现了不同 GPU 计算平台间的性能移植.

值得注意的是,本文工作是在已有工作[2] 的基础上进行的改进和提升.相对于之前工作,本文主要工作体现在整体优化框架的提出、Uberkernel 调度、本地队列分层次优化、新的线程与任务动态映射机制 4 个方面.与前期工作相比,最终获得了 22.3%～51.6%的性能提升.

本文的主要贡献如下:

(1)研究和实现了 Viola-Jones 人脸检测算法在 GPU 计算平台上的关键优化技术和方法.

① OpenCV: http://www.opencv.org.cn

(2)提出了一个面向 GPU 计算平台,针对具有负载不均衡特征算法的并行优化框架.

(3)实现了人脸检测算法在 NVIDIA GTX680 和 AMD HD7970 两个不同 GPU 计算平台上的高性能与性能移植.

本文第 2 节为相关工作介绍,详细讨论 Viola-Jones 人脸检测算法在 GPU 计算平台上实现和优化的相关工作;在第 3 节简要介绍 GPU 架构以及 Viola-Jones 人脸检测算法之后;第 4 节详细讨论该算法在 GPU 计算平台上实现和优化的关键技术和方法;并在第 5 节给出性能评估结果;最后,第 6 节进行总结.

2　相关工作

近年来,关于 Viola-Jones 人脸检测算法的 GPU 移植和优化已有许多工作,代表性的工作主要有:David 等人[3]通过优化图像积分图算法在 GPU 上的性能来提高人脸检测算法的整体性能,但是对人脸检测部分在 GPU 上的实现和优化工作较少,更没有指出如何解决算法中由于负载不平衡问题而导致的性能瓶颈. Kong 等人[4]通过使用 share-memory 存储待检测窗口,通过数据本地化减小对访存带宽的依赖,但其主要工作集中在检测窗口的并行处理上,并没有对算法并行性进行充分挖掘,更没有解决算法中由于负载不平衡问题而导致的性能瓶颈. Sharma 等人[5]将 Viola-Jones 人脸检测算法移植到 GPU,通过建立图像金字塔机制实现对不同大小的缩放图像的统一处理,从而在一定程度上缓解了图像间负载不均衡的问题,但对于 Thread 级和 work-group 级的负载不均衡现象没有进行进一步研究. Ghorayeb 等人[6]充分分析了人脸检测算法的并行性,提出该算法具有三级并行性:特征值计算并行、检测窗口并行和缩放图像并行,并通过对这三级并行的性能优化,使人脸检测算法的性能达到了在 FPGA 上的性能,但在负载不均衡的处理上,仅仅通过在工作量大的一级分类器中,将负责每个探测窗口的线程数目提高两倍的方法来解决. 这种方法不仅简单粗暴而且非常低效,只能在较低的程度上缓解负载不均衡问题,但不能有效解决. Jia 等人[2]认为层次式队列机制是解决负载不均衡的优先选择,并构建了本地和全局两级队列解决人脸检测算法 Thread 级和 work-group 级负载不均衡问题. 然而该工作的全局队列实现较为简单,没有实现线程和数据的动态映射机制,导致全局队列的任务调度实现方式不够高效,同时也没有解决全局队列的高效访问的问题.

总之,虽然上述工作在对 Viola-Jones 人脸检测算法的 GPU 移植方面取得了很大的进展,但在并行性发掘以及负载不均衡导致的性能瓶颈的处理上,这些工作的研究依然不够深入,更没有给出一个统一、有效的解决方案.

与此同时,近年来也有许多工作通过其他算法的 GPU 移植,提出了克服 GPU 计算平台上线程间负载不均衡的方法和策略. 如 Tzeng 等人[7]通过光纤追踪算法在 GPU 上的移植和优化,构建了非规则算法在 GPU 实现中的作业管理机制,并提出有效的内存管理和动态调度是负载不均衡应用在 GPU 上达到高性能的关键因素. Merrill 等人[8]和 Aila 等人[9]根据以上方法,分别完成了光线追踪和广度优先图遍历算法在 GPU 上的实现和优化,并取得了很好的加速比. Nasre 等人[10]通过图算法在众核计算平台上的实现和优化,根据其算法特性,得出在众核编程环境下,Topology-driven 比 Data-driven 更能降低负载不均衡,从而会大幅提升程序性能的结论. Cederman 等人[11]和 Chatterjee 等人[12]通过将改进的任务窃取算法应用到全局队列中,以动态任务调度的方式大大缓解了 work-group 间的负载不均衡现象,从而获得了较高的性能加速,但该方法实现过于复杂. Yan 等人[13]通过引进邻接同步的定义和算法,通过消除全局同步以及降低线程间的任务依赖达到解决负载不均衡的目的,从而提高了程序性能. Burtscher 等人[14]提出了针对具有条件分支和访存不规则特征的算法在 GPU 上实现和优化的方法和策略. Nasre 等人[15]针对非规则算法在 GPU 计算平台上实现和优化中,原子操作可能会导致的性能瓶颈,提出了 atom-free 编程方法,有效地解决了原子操作导致的性能瓶颈. 虽然,这些工作针对负载不均衡算法在 GPU 上的实现和优化都提出了不同的编程和优化方法,也都取得了很好的加速效果,但是这些方法都没有在 Viola-Jones 人脸检测算法上验证其有效性. 然而,这些工作提出的方法和思想,对于 Viola-Jones 人脸检测算法的 GPU 移植具有很好的借鉴意义.

本文将从上述的已有研究工作出发,结合已经提出的优化策略方法,根据 Viola-Jones 人脸检测算法的算法特点和 GPU 硬件架构特征,实现该算法在 GPU 计算平台上的高性能移植. 同时,通过抽象影响性能的关键参数,本文实现了该算法在不同 GPU 计算平台上的性能移植.

3 背景介绍

本节将详细描述 GPU 硬件架构特征和 Viola-Jones 人脸检测的算法特点.

3.1 GPU 架构

随着 GPU 通用计算应用的推广及其应用领域的不断扩大,GPU 架构的发展非常迅速.主流芯片厂商根据实际计算需求,不断发展自己的 GPU 架构来满足日益增长的市场需求和性能需求.NVIDIA 已经发布了如 Fermi[16]、Kepler[17]和 Maxwell[18]架构的 GPU,AMD 也发布了 Cypress、Cayman、GCN 架构[19]的 GPU.这些 GPU 都具有不同的架构特点,对应的优化策略也有差异.这种 GPU 架构的日益多样性,对算法移植特别是性能移植提出了新的挑战.

幸运的是,虽然不同的 GPU 架构在优化方法选择以及优化细节上会有不同,但是从整体上看,GPU 架构又具有很好的统一性,即都是大规模细粒度并行处理器,且具有层次式的架构特点,这主要体现在 4 个方面:(1)在计算单元的组织上,GPU 由多个计算单元(Compute Unit,CU)组成,每个 CU 又由多个处理部件(Processing Unit,PE)组成;(2)从内存组织上,从只能被一个线程访问的私有内存,到能够被 work-group 内所有线程共享的本地内存,再到可被所有线程访问的全局内存;(3)在线程调度上,GPU 一般都采用静态顺序调度模式,线程和数据映射在 GPU kernel 启动之前就已经确定,线程以 work-group 为单位被硬件顺序调度;(4)在编程方式上,都采用了 Host + Device 的编程模式,即 GPU 程序依然由 CPU 端控制,CPU 设定开启线程的数目后,将计算任务发送到 GPU 端执行,线程组织也都采用了 Grid-Block-Thread 的层次式组织方式.GPU 这种统一的架构特点和线程组织调度方式,使性能移植成为可能.Jia 等人[20]给出了在不同 GPU 计算平台上优化策略的选择及应用.

本文将以 Viola-Jones 人脸检测算法为例,研究该算法在不同 GPU 计算平台间性能移植的方法.

3.2 Viola-Jones 人脸检测算法

Viola-Jones 人脸检测算法由剑桥大学的 Viola 和 Jones[1]于 2001 年最早提出,是最为成功且满足实时要求的可实用人脸检测算法.Viola-Jones 人脸检测算法使用 Haar 特征值进行目标检测:通过 Adaboost算法生成级联分类器,直接对图像的一小块区域进行特征匹配,从而判断该区域内是否有人脸存在.该算法包括训练和检测两部分:训练部分使用 Adaboost 算法从预先收集的正负样本中提取特征值进行计算,最终生成一个级联分类器;检测部分使用级联分类器,通过检测窗口的移动和缩放,对图像上的人脸进行检测.由于训练部分可脱机执行,对实时要求并不高.因此,本文只讨论算法的检测部分在 GPU 上的实现和优化.

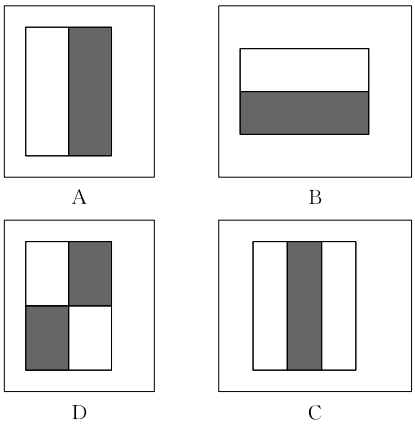


图 1 Haar 特征模板[1]

如图 1 所示,Viola-Jones 人脸检测算法使用 Haar 特征值计算特征.Haar 特征值共使用了 3 种类型 4 种形式的特征模板,每种特征模板内包含黑色和白色两种矩形,该模板的特征值定义为白色矩形像素和减去黑色矩形像素和的差值.

图 1 所示的特征模板称为“特征原型”,“特征原型”在图像窗口中平移伸缩而得到的特征称为“矩形特征”,“矩形特征”的值称为“特征值”.在检测过程中,特征模板可以以任意尺寸放置在图像窗口上的任意位置.从而形成各种形态,每一种形态称为一个特征.这样,通过改变特征模板的大小和在图像中的位置,可在图像窗口中穷举出海量特征.

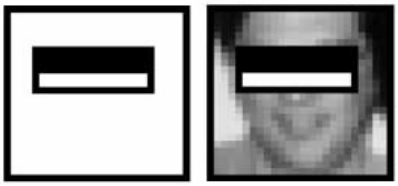


图 2 人脸检测中的 Haar 特征[1]

图 2 显示了人脸检测中的 Haar 特征值在样本窗口中的位置.对于一个 24×24 的图像样本而言,矩阵特征的数目可达 160000,这是非常庞大的计算量.而且随着图像尺寸的不断增大,特征数目也会快速增长,这就对计算能力提出了严峻挑战.同时,如果每次 Haar 特征值的计算,都要统计矩形中所有像素的和,那么庞大的计算量无疑会降低检测的速

度.为此,Viola 与 Jones 引入了积分图作为输入图像的一种中间表达形式.在完成图像的积分图运算后,通过图像积分图来完成 Haar 特征值的计算.因此,积分图生成算法的性能对于人脸检测算法的性能也至关重要.然而,这超出了本文的讨论范围,关于积分图生成算法的 GPU 移植,已有大量的研究工作,并且已经非常成熟.感兴趣的读者可以参考文献[3].

图 3 显示了 Viola-Jones 人脸检测算法的主要流程,Viola-Jones 人脸检测算法通过定义检测窗口遍历整幅图像,使用 Adaboost 方法构建的级联分类器检测检测窗口中的人脸.值得注意的是,通过使用级联分类器可明显加快检测速度.检测流程具体如下:首先,为了检测出图像中包含的不同大小的所有人脸,需要按照一定大小的缩放因子缩放图像;其次,使用级联分类器对检测窗口进行检测.级联分类器将检测窗口交给第一级分类器,如果检测窗口检测为未包含人脸,则分类器将此检测窗口丢弃并停止检测,如果当前探测窗口被判为疑似包含人脸,则进入下一级分类器继续检测.最后,只有通过所有分类器检测的窗口才被判定为包含人脸.在级联分类器中,分类器的复杂度是逐层增加的,层数越深,检测要求越高,与之相对应的计算量也越大.

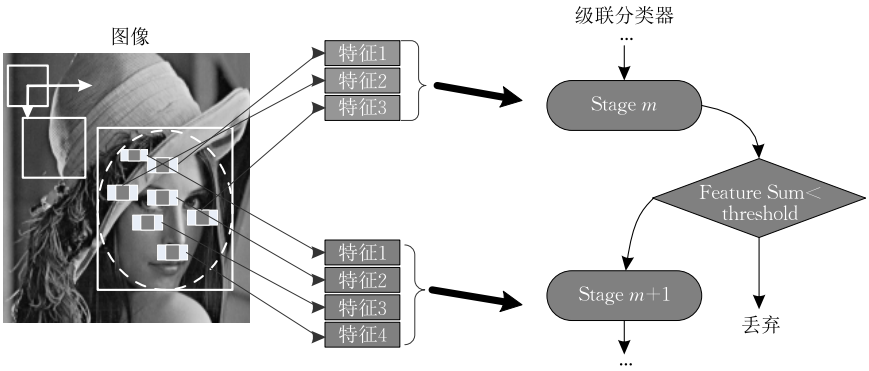


图 3　Viola-Jones 人脸检测算法

4　人脸检测算法的 GPU 实现与优化

4.1　并行性分析

通过 3.2 节的算法介绍,我们可以分析得出 Viola-Jones 人脸检测算法具有良好的并行性,具体来说,该算法包含 3 级并行性:

(1)特征值级并行:每个特征值的计算是相互独立的,可以并行执行;

(2)窗口级并行:每个检测窗口的检测是相互独立的,可以并行检测;

(3)图像级并行:为了检测大小不同的人脸,需要将图像进行放缩,对每一幅缩放图像的检测也是相互独立的,可以并行执行.

由此可见,一方面 Viola-Jones 人脸检测算法的计算量非常大,但另一方面,该算法又具备良好的并行性.这种算法特性非常适合 GPU 计算平台大规模并行的架构特点.针对以上分析,本文实现了 Viola-Jones 人脸检测算法在 GPU 计算平台上的 naïve 版本.

4.2　naïve 实现与负载不均衡

Viola-Jones 人脸检测算法在 GPU 计算平台上的 naïve 版本采用最基本、最简单的并行化策略,即仅并行化检测窗口的检测,具体过程为:(1)每次只处理一幅缩放图像,在 CPU 端循环启动 OpenCL kernel 以处理所有的缩放图像;(2)对于每幅缩放图像,每个线程负责一个检测窗口,多个检测窗口可进行并行检测;(3)使用 LDS 完成数据本地化,实现 work-group 内线程的数据共享,减少对访存带宽的依赖;(4)在线程组织上,采取有多少检测窗口,就开启多少线程的策略,每个 work-group 大小为 256,并采取二维组织形式:16×16.

然而,与 CPU 版本的性能相比,naïve 版本的性能并没有提升,反而有所下降.这主要有两方面的原因:一是没有充分发掘并行性,naïve 版本仅仅开发了三级并行性中的一级,即检测窗口并行;二是在 naïve 实现版本中,线程间存在严重的负载不均衡现象,这是 GPU 这种大规模细粒度并行处理器的梦魇.这使得 GPU 计算资源远未得到充分利用,从而导致性能的严重降低.其中第 2 个原因是最主要、最关键的因素.

Viola-Jones 人脸检测算法的串行实现中,级联分类器之所以能够提高检测速度,是因为在一般的输入图像中,大部分区域都不包含人脸.通过前面的几级简单分类器就可以直接滤去这些区域,只对少

量的极可能包含人脸的区域使用更为复杂的分类器进行检测.在这个检测过程中,随着弱分类器数量的增加,通过 Adaboost 构建的强分类器的检测表现也会不断提高.但这种方法也导致了该算法在 GPU 的实现和优化中,会导致严重的负载不均衡现象,大大限制了其在 GPU 上的性能.图 4 说明了这种负载不均衡现象的产生过程.

如图 4 所示,假设有 9 个检测窗口,共开启 9 个线程,每个线程负责一个窗口的检测.当使用级联分类器进行检测时:窗口(0,0)首先被 Stage 0 分类器检测为肯定不是人脸而被丢弃,在此后的检测过程中,线程(0,0)将一直处于空闲状态,其他线程继续工作;接着在下一级检测中,窗口(1,1)被 Stage 1 分类器也检测为不是人脸,因此线程(1,1)在随后的检测中,也将处于空闲状态;更为不幸的是,随着检测的进行,越来越多的窗口被检测为不是人脸而被丢弃,也就是说越来越多的线程处于空闲状态;直到最后一个阶段,只有线程(1,2)处于忙碌状态,而其他线程都处于空闲状态.更为严重的是,根据级联分类器的定义,检测越靠后,分类器级数越高,相对应的计算量就越大.因此,实际的负载不均衡现象远比图 4 描述的严重得多.

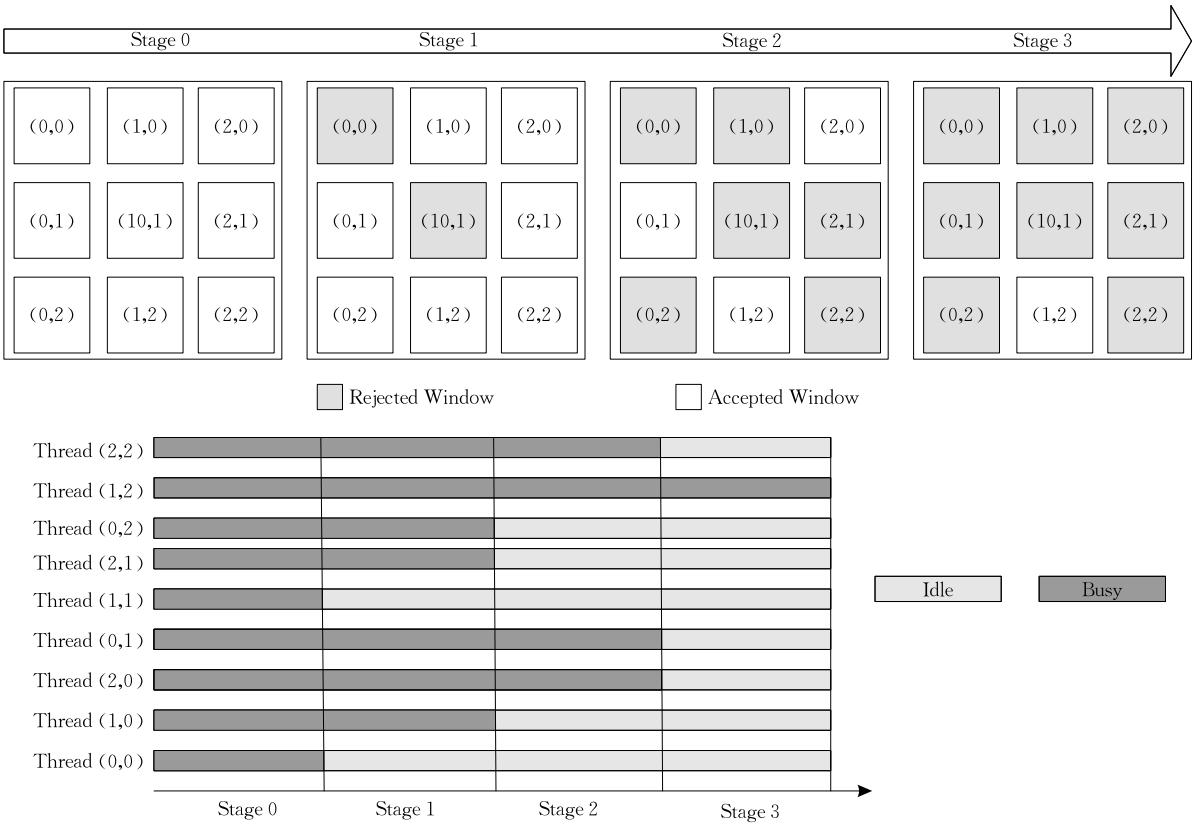


图 4 线程间负载不均衡现象

由此可见,级联分类器方法虽然在串行算法中可大大减少工作量,提高检测速度,但在 GPU 实现中却会导致严重的线程间负载不均衡现象,成为性能瓶颈,严重制约算法性能.特别是在大多数图像中,人脸区域可能只占很少一部分.如果对该算法的 GPU 实现不进行改进,也就意味着在实际 GPU 程序中,只有一小部分线程会一直处于工作状态,直到检测结束.而绝大部分线程可能很快就退出并在 CPU 端等待全局同步了.这显然是我们不愿意看到的结果.

此外,为了检测图像中大小不同的人脸,需要将图像按照一定的缩放因子进行放缩.naïve 实现在 CPU 端循环处理这些缩放图像.这样处理不仅增加了 GPU kernel 的启动和同步开销,而且当缩放图像过小(如只有几个甚至一个检测窗口时)而不能充分利用 GPU 的计算资源时,就不能充分利用 GPU 强大的计算能力,从而造成资源浪费.

因此,在实际图片的人脸检测中,Viola-Jones 人脸检测算法存在三级负载不均衡:

(1) Thread 级.线程负责的检测窗口中的图像越接近于人脸,该线程的工作量越大;否则,工作量越小.

(2) Work-group 级. 当一个 work-group 处理的图像区域包含人脸时，工作量巨大；否则，工作量可能会很小.

(3) 图像级. 当图像不断放缩而不能充分利用 GPU 计算资源时，就会导致图像级的负载不均衡.

本文下面的内容将着重讨论如何解决这些负载不均衡的问题. 幸运的是在解决负载不均衡问题的同时，并行性没有充分发掘的问题也一并得到了解决.

4.3　GPU 优化

如上节分析，Viola-Jones 人脸检测算法虽然具有很好的并行性，但在 GPU 移植中存在负载不均衡的非规则特性. 这种非规则特性是 GPU 计算平台的梦魇：一方面，GPU 具有大规模细粒度并行的架构特点，负载不均衡会导致 GPU 计算资源利用率的降低；另一方面，现代 GPU 的线程调度采用静态调度策略，如不进行针对性优化，无法自动处理负载不均衡问题. 更为严重的是，传统的 GPU 编程和优化方法并没有涉及对负载不均衡现象的处理和优化. 因此，负载不均衡现象将会成为人脸检测算法在 GPU 计算平台上的性能瓶颈，仅仅使用传统 GPU 编程和优化方法(即计算、访存和数据本地化优化)无法克服该瓶颈，将严重制约算法的性能. 对此，本文针对 Viola-Jones 人脸检测算法的特性，结合 GPU 的架构特征，提出了一种并行优化框架，以突破由于负载不均衡导致的性能瓶颈. 图 5 显示了该优化框架的整体架构.

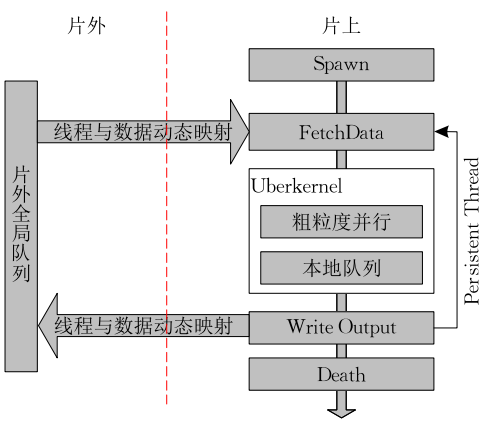


图 5　并行优化框架

并行优化框架主要由 6 个部分组成：粗粒度并行、Persistent Thread、Uberkernel、线程与数据的动态映射、全局及本地队列. 这 6 个组成部分相互协同，共同解决由于负载不均衡导致的性能瓶颈问题：

(1) Uberkernel. 将执行人脸检测算法主要计算部分的多个 kernel 合并为一个 Uberkernel，统一负责人脸检测. 这样一方面可以减少 kernel 的启动和全局同步开销，另一方面也可解决图像级负载不均衡问题.

(2) Persistent Thread 与粗粒度并行. Persistent Thread 和粗粒度并行共同定义了 Uberkernel 的线程组织和运行方式，作为解决线程间负载不均衡问题的基础. 粗粒度并行通过重新定义线程组织方式，提升 GPU 并行粒度：由 thread 变为 warp (wavefront，AMD GPU)，在一定程度上缓和了负载不均衡对性能的影响. Persistent Thread 重新定义了 GPU thread 的运行方式，使 thread 的生命周期和 OpenCL kernel 的生命周期相同，可循环处理多个任务.

(3) 动态映射与全局队列. 在线程和任务的映射上，本文不采用传统 GPU 编程中的静态映射方式，而是构建线程与任务的动态映射机制，根据线程的任务负载情况，实现线程和任务的动态映射. 同时构建以此为任务调度策略的全局队列，解决 work-group 间负载不均衡的问题.

(4) 本地队列. 构建位于共享内存(LDS，AMD GPU)的本地队列，work-group 内的所有线程协同工作，解决 work-group 内的负载不均衡问题.

本节将对以上关键优化方法和技术进行详细讨论和介绍.

4.3.1　Uberkernel

如 3.2 节所述，为了检测图像中不同大小的人脸，我们需要按照一定的缩放因子对图像进行缩放，直到缩放图像和检测窗口为同等大小为止，这样就形成了一个缩放图像集. 如果一次只处理一幅图像，并通过在 CPU 端通过多次启动 GPU kernel 循环处理这些缩放图像，不仅会增加 GPU kernel 的启动和全局同步开销，而且当缩放图像过小而不能充分利用 GPU 的计算资源时，会导致 GPU 计算资源的极大浪费，由此会产生图像级负载不均衡.

为此，我们引入了 Uberkernel 机制，其核心是通过 kernel 合并，一次处理多幅甚至所有图像. 这样仅通过一次或几次 OpenCL kernel 的启动就全部处理完所有的缩放图像. Uberkernel 的具体流程如图 6 所示.

在 GPU 的 Global Memory 中设立一个统一的地址空间，将所有图片按照缩放比例，顺序放入统一的地址空间中. 同时，OpenCL kernel 将对这些缩放图片按照统一的方式进行处理. 当图片太大，统一的

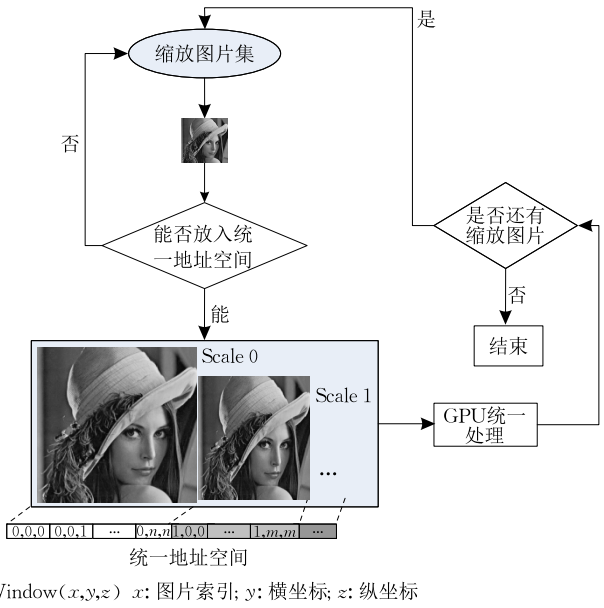


图 6 统一地址空间

地址空间不能全部容纳所有缩放图片时，则按照统一地址空间最大化利用的原则(尽可能填满统一地址空间)对缩放图片进行分组，然后循环处理每组图片. 该机制尽可能的保证每个 OpenCL kernel 的工作量足以充分利用 GPU 所有的计算资源，很好地解决了图像级负载不均衡问题.

4.3.2 粗粒度并行与 Persistent Thread

在 Uberkernel 的线程组织及运行方式上，本文引入组粒度并行与 Persistent Thread.

传统 GPU 编程是大规模细粒度并行，即一次开启大量线程，并以单个软件线程(thread)作为并行粒度. 这种编程方式无疑是线程间存在负载不均衡特性算法的噩梦. 为此，本文在 Viola-Jones 人脸检测算法的 GPU 实现中，采用硬件线程(NVIDIA GPU 为 warp，含 32 个 thread，AMD GPU 为 wavefront，含 64 个 thread)作为并行粒度. 同时，硬件线程内多个 thread 协同工作，共同处理分配的检测窗口，其协同工作方式在 4.3.5 节本地队列中会详细讨论. 粗粒度并行的实现方式较为简单：一个 work-group 只包含一个 warp 或者 wavefront，work-group 将作为全局队列任务分配的单位.

采用粗粒度并行编程方式的优势主要有 3 个：

(1) 移除本地同步操作. wavefront/warp 是 GPU 最基本的执行和调度单元. 当一个 work-group 内只包含一个 wavefront/warp 时，可移除本地同步操作，减少本地同步开销，在一定程度上提高程序性能.

(2) 一个 work-group 只包含一个 wavefront/warp，我们可以将 work-group 看成是一个与其他 work-group 执行相互独立的多指令多数据(Mutiple Instruction Multiple Data，MIMD)线程. 即保证了 SIMD 执行方式的有效性，又提供了 MIMD 式的工作粒度.

(3) 减轻 work-group 内线程间的负载不均衡. 在粗粒度并行模式下，任务的分配以 work-group 为单位，work-group 内的所有线程协同处理所分配的计算任务. 这种工作模式结合我们下面即将讨论的本地队列，可大大减小 work-group 内线程的负载不均衡现象.

当然，粗粒度编程方式也存在一个劣势：由于 GPU 硬件资源的限制，每个 CU 上同时运行的 work-group 数目是有限制的. 因此，当 work-group 包含的线程数目较少时，可能会导致 CU 上同时运行的线程不足，从而不能有效地隐藏访存延迟. 幸运的是，由于人脸检测算法有较大的计算密度，这个劣势可以消除. 在实际实现中，结合数据本地化，为每个 CU 部署 8～12 个 work-group 即可有效地隐藏访存延迟.

在运行方式上，传统 GPU 线程的生命周期一般为五个过程：启动、获取操作数据、处理数据、写回处理结果、退出，其运行及调度方式都是静态的且由 GPU 硬件控制. 这种运行方式显然对于解决负载不均衡的问题是非常不利的. 因此，本文引入了 Persistent Thread，其生命周期和 OpenCL kernel 的生命周期相同，并可循环处理多个任务：线程在将一次数据处理的结果写回后，不是立即退出，而是判断是否还有别的任务需要处理，如果有，线程将继续获取任务进行处理，如没有才退出. 这样在 Viola-Jones 人脸检测算法中，可为每个线程分配多个检测窗口，线程将循环处理这些检测窗口，直到将分配给它的窗口全部处理完毕. 同时，线程与检测窗口的映射将根据线程的任务负载情况采用动态映射方式，从而最大限度的保证了线程间的负载均衡. 这将在 4.3.3 节线程与任务的动态映射中进行详细讨论.

4.3.3 线程与任务的动态映射

在线程与任务的映射方面，传统 GPU 编程采用静态编程模式，即在 GPU kernel 启动之前，线程和任务的映射就已经确定，线程由 GPU 硬件顺序调度执行，每个线程处理的任务和任务数都是固定的. 这种编程模式虽然很好的满足了规则的数据并行应用，但对于具有线程间负载不均衡特征的人脸检测算法，无法解决其存在的负载不均衡问题. 本文

在 Persistent Thread 的基础上，引入 GPU 动态编程模式，该模式具有以下 3 个特征：

(1) 固定开启线程数目. 根据目标 GPU 计算平台的 CU 数量，确定开启的线程数目. 在人脸检测算法中，共开启 $8\times N$(AMD GPU)或者 $12\times N$(NVIDIA GPU)个 work-group，其中 N 为 CU 数目. 每个 work-group 包含 64(AMD GPU)或 32(NVIDIA GPU)个线程.

(2) 每个线程分配多个计算任务. 结合 Persistent Thread 编程方式，每个线程循环处理多个任务.

(3) 在线程的任务分配方面，在 GPU kernel 启动之后，根据线程的实际任务负载情况确定线程与任务的映射关系.

(4) 该策略将作为全局队列(4.3.4 节)的任务调度机制，根据粗粒度并行的定义，以 work-group 为单位进行任务分配.

图 7 显示了线程与任务动态映射的过程：首先每个 work-group 的第 0 号线程作为标记线程，访问位于全局内存上的原子变量 G，在获得原子变量访问权后，对原子变量加 N(N 为 work-group 每次处理的窗口数目，在 AMD GPU 上 N 为 64，在 NVIDIA GPU 上 N 为 32)；然后判断 G 是否小于检测窗口总数 Total$+N$，如果小于，则获取该 work-group 要处理的检测窗口，否则对应 work-group 退出；最后将待处理的 N 个检测窗口返回给对应的 work-group. work-group 以 4.3.5 节介绍的本地队列机制处理

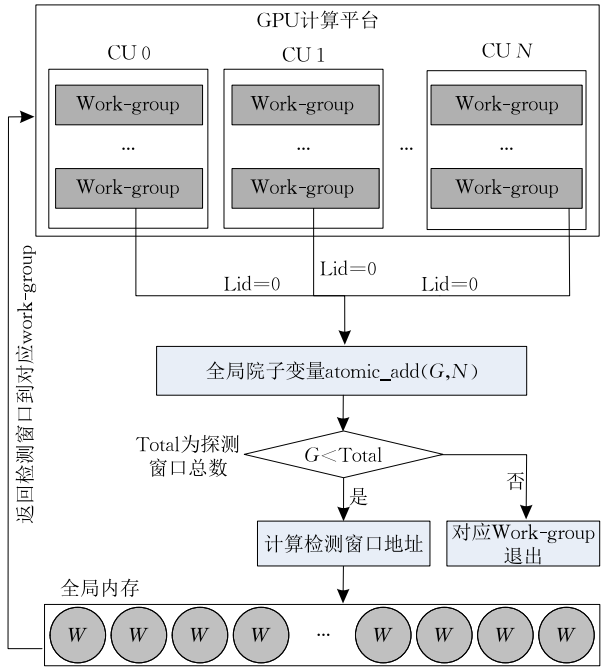


图 7　线程与任务动态映射示意图

完这些检测窗口后，再重复以上操作，直到所有检测窗口都处理完毕为止. 注意，所有检测窗口以队列形式存储在全局内存上，全局内存的构建组织方式将在 4.3.4 节详细讨论.

图 8 为线程与任务动态映射的伪代码.

```
G：位于全局内存的原子变量，记录被处理的窗口数
N：每个 work-group 一次处理的窗口数
1.    __kernel facedection(…){
2.      int lid=get_local_id(0);
3.      __local int local_index;
4.      while(1){
5.        if(lid==0) local_index=atom_add(G,N);
6.        Barrier(CLK_LOCAL_MEM_FENCE);
7.        if(G<Total+N){//Total 为检测窗口总数
8.          int index=com_data_add(local_index);
9.          fetch_data_and_compute(index);
10.       }
11.     }
12.   }
```

图 8　动态映射伪代码

动态编程模式根据线程的实际任务负载情况进行任务分配，即当 work-group 处理的窗口包含人脸而导致计算量过大时，该 work-group 处理的窗口数目就会变少；反之，当 work-group 处理的窗口计算量小时，该 work-group 就会处理更多的窗口. 因此，动态编程模式在一定程度上解决了 work-group 间负载不均衡的问题.

当然，动态编程模式会因原子变量访问导致额外的开销. 但一方面，由于 work-group 的调度执行存在一定的时间间隔，所以这个开销会非常小；另一方面相对于负载不均衡导致的性能瓶颈，这个开销几乎可以忽略不计. 因此，动态编程方式会大幅提高人脸检测算法在 GPU 上的性能.

4.3.4　全局队列

线程与任务的动态映射机制能够很好的解决 work-group 间负载不均衡问题的前提是：位于全局内存上的待检测窗口必须被很好的组织，能够及时响应访存请求. 因此，本文引入了全局队列.

全局队列的作用是以队列的形式组织好待检测窗口，在线程与负载动态映射机制下，能方便的建立起线程与数据的动态映射关系. 结合前面讨论的 Uberkernel 和 Persistent Thread，全局任务队列的工作流程如下：首先将 Uberkernel 中所有的探测窗口都加入到该队列中；其次以 32(NVIDA GPU)或者 64(AMD GPU)为单位将探测窗口分成若干任务组，并将其作为任务调度单位；最后，根据线程与任务动态映射机制，以 work-group 为分配单元，

根据线程的实际任务负载情况，完成任务的动态分配.

全局队列除了任务与负载动态映射机制外，没有使用更加复杂的任务调度方式.这里有两方面的原因：一方面，全局队列位于全局内存上，其访存和原子操作的开销都非常昂贵.复杂的任务调度方式不仅难以实现，而且可能会产生昂贵的调度开销，在性能的提升上得不偿失.另一方面，线程与数据动态映射机制已经决定了全局任务队列的任务调度方式，且这种任务调度方式足够解决 work-group 间的负载不均衡现象.

4.3.5 本地队列

线程与任务的动态映射及全局队列机制的引入，较好地解决了 work-group 间负载不均衡的问题，而 work-group 内线程间负载不均衡的问题由位于片上本地内存(共享内存，NVIDIA GPU)的本地队列解决.

图 9 详细地显示了本地队列及其任务处理过程.work-group 将其负责处理的待检测窗口组织为位于片上本地内存的队列，然后使用级联分类器进行检测.每级分类器将会检测队列中的所有窗口，通过本级检测的窗口将会重新进入队列，等待下一级分类器的进一步检测；否则，该窗口将会被丢弃.其检测过程分为两个阶段：

(1)单独处理阶段.因为 0～2 级分类器特征数目较少，可以很快完成计算，故令每个线程单独负责处理一个窗口.在这个过程中，当使用 1～2 级分类器进行检测时，会有线程处于空闲状态，但这个时间太短，不足以引起性能瓶颈，反而能减少协同处理开销.

(2)协同处理阶段.随着分类器级数的增大，特征数目和计算量会急剧增长.同时，经过 0～2 级分类器的检测，队列中的待检测窗口数目也会减少.此时进入协同处理阶段，即一次从队列中取出 M(在 AMD GPU 上 M 为 4，Nvidia GPU 上 M 为 2)个检测窗口，由 work-group 内所有线程协同处理(通过检测的窗口返回队列)，直到队列中的所有窗口都通过本级分类器检测为止；然后，进入下一级分类器继续检测；最后，只有经过最后一级分类器检测的检测窗口中才包含人脸.

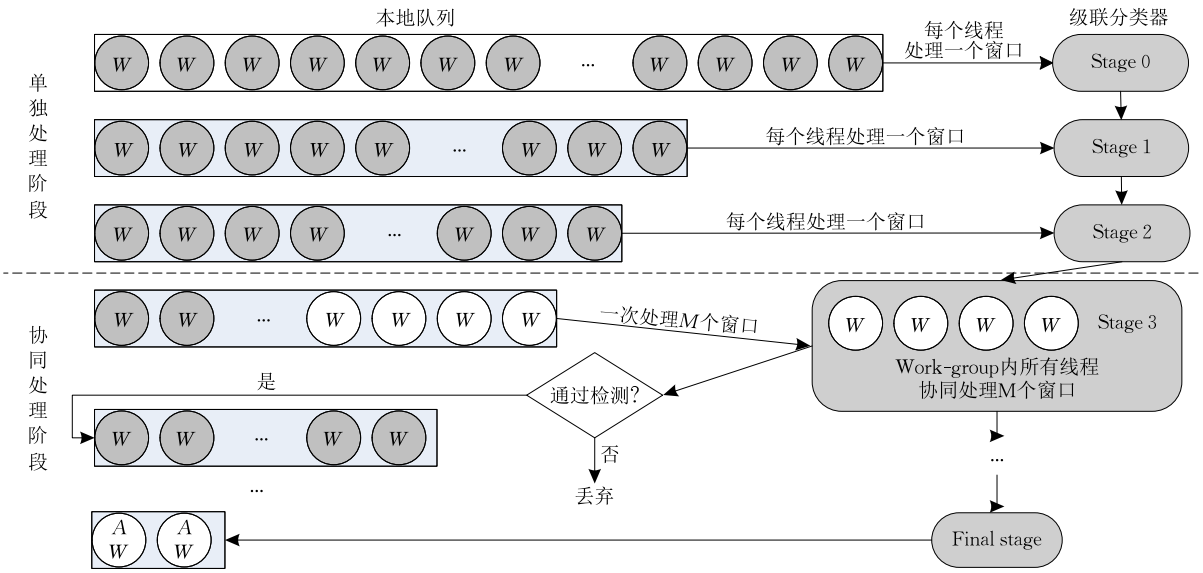


图 9 本地队列

由此可见，本地队列基本上解决了 work-group 内部线程间负载不均衡的问题.

4.3.6 其他优化方法

除以上优化方法外，Viola-Jones 人脸检测算法还使用了 GPU 传统优化方法：

(1)开发 ILP.开发 ILP 主要有两种方式：一是循环展开；二是调整代码顺序，使相同指令类型(GPU 的指令类型可分为 3 种：读内存指令、计算指令以及写内存指令.在执行过程中，GPU 会将相互独立、相同类型且相邻的指令打包在一起并行执行)的代码打包在一起，编写对编译器友好的代码.

(2)指令选择优化.因为人脸检测函数较高的计算密度，选择高吞吐量的指令对性能的提升就变得尤为重要.在该算法的实现中，共采取了两种指令优化：一是使用位运算指令代替乘法和除法指令；二是用 mad24、MUl24 指令代替乘加指令.

(3)减少动态指令. 主要方法是减少条件分支. 因为人脸检测算法中存在着大量的条件判断语句，因此使用?:语句代替 if…else…语句是减少动态指令的主要方式.

4.3.7　性能移植优化

如前文所所述，虽然 GPU 架构日益多样化，但是统一的层次式架构模式为性能移植提供了可能. 只要完成了关键性能参数的抽取，并建立完善的性能参数传递机制，就可实现不同 GPU 硬件平台间的性能移植.

两个 GPU 计算平台的关键性能参数抽取如表 1 所示. 在性能参数传递机制构建方面，利用 OpenCL 程序运行时编译的特点是通过宏定义将性能参数在编译时传入 OpenCL kernel，根据目标平台，传入相应的性能参数. 根据以上方法，本文在不修改代码的前提下，最终实现了 Viola-Jones 人脸检测算法在 AMD Radeon HD7970 和 NVIDIA GTX680 两个不同 GPU 计算平台上的性能移植.

表 1　不同 GPU 计算平台的关键性能参数

GPU	AMD Radeon HD7970	NVIDIA GTX 680
work-group 大小	64	32
work-group 数目	28×8	14×12
全局队列任务发送单元	64	32
本地队列协同处理单元	4	2
处理每个检测窗口的线程数目	16	16
循环展开次数	4	2

5　性能评估

5.1　测试平台搭建

本文选取 AMD HD7970 和 NVIDIA GTX680 两个不同架构的 GPU 作为性能测试平台，选取 Intel(R) Xeon(R) X5550 Quad Core CPU(2.66 GHz 8 MB L3 Cache)作为 CPU 计算平台. 选取两个不同计算平台的目的是验证性能可移植性. 两个 GPU 计算平台的主要性能参数如表 2 所示.

表 2　GPU 计算平台性能参数

GPU	Clock Rate/GHz	PE	CU	Peak Per/GFlops	Memory/GB	Peak BW/(GB/s)	Register/CU/K	LDS/CU/K
AMD Radeon HD7970	0.925	2048	32	3790	3.0	264	16	64
NVIDIA GTX 680	1.006	1536	8	3090	6.0	192	16	48

Viola-Jones 人脸检测算法的 CPU 版本来自 OpenCV2.4，该版本已深度优化，在 CPU 平台上具有较高的性能. 因此，选择该版本，可提高性能对比的可信性. 值得注意的是 CPU 版本在编译时，添加“-o3”选项，以充分利用 CPU 的计算资源，提高 CPU 串行版本的性能. 检测图片来源于 CMU 人脸检测项目所用的数据库[21]，该数据库是 CMU 人脸检测项目的专用数据库，提供了用于大量评估算法准确性的人脸正面图片，非常具有代表性.

在实际测试中，我们选择了 50 张图像共包含 427 个人脸来评估我们实现的人脸检测算法 GPU 版本的正确性和性能. 图 10 列举了其中 3 幅不同大小、不同背景、不同人脸数目的图片. 其中图片 1 的大小为 256×337，包含人脸数目为 1；图片 2 大小为 696×510，包含人脸数目为 12；图片 3 大小为 1280×1024，包含人脸数目为 56.

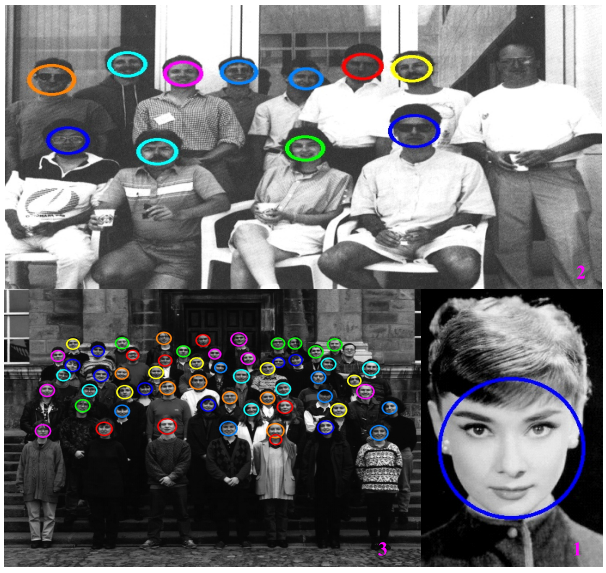


图 10　检测图像

5.2　正确性验证

本文选取 OpenCV2.4 库中实现的 Viola-Jones 人脸检测算法作为进行 GPU 移植和优化的基准 CPU 程序，OpenCL 程序的各方面参数与该基准程序保持一致. 同时，检测模型也直接使用 OpenCV2.4 自带的检测模型. 因此，本文的正确性验证只和该 CPU 串行代码的运行结果进行比较. 值得注意的是，通过改进算法及检测模型来提高检测的准确率并不是本文关注的内容，本文主要关注的是相同算法实现在 GPU 计算平台上的性能提升.

图 10 显示了本文实现的 OpenCL 版本对 3 幅不同图像的检测结果. 从中我们可以看出:并不是所有的人脸都被检测出来. 这主要是因为这些未检测出的人脸并不是标准正面像. 同时也可以发现许多没有人脸的区域也被检测为人脸,如图像 3. 这是因为这些区域在一定程度上与人脸相似. 从以上分析可以看出:OpenCL 版本并没有达到 100%的人脸检测精确度. 但通过和 OpenCV 库中的 CPU 串行版本的运行结果相比较,我们发现人脸检测精度没有实现 100%与串行算法实现及检测模型有关,而与 OpenCL 实现无关.

表 3 CPU 版本与 OpenCL 版本人脸检测数目(单位:个)

图片编号	CPU 版本		OpenCL 版本	
	group 前	group 后	group 前	group 后
1	24	1	24	1
2	621	11	621	11
3	2896	58	2896	58

表 3 显示了CPU串行版本和本文实现的 OpenCL 版本针对不同图片的人脸检测数目. 从中可以看出,本文实现的 OpenCL 版本在人脸检测数目上与 CPU 串行版本完全一致,这就证明了人脸检测算法 OpenCL 版本实现的正确性. 而人脸检测的准确率可通过改进算法实现或者使用更精确的检测模型来提高,但这并不是本文关注的内容.

5.3 性能分析

5.3.1 整体性能分析

图 11 和图 12 分别显示了优化后的 Viola-Jones 人脸检测算法在 AMD HD7970 和 NVIDIA GTX680 两个不同 GPU 计算平台上的性能及相对于 CPU 版本的性能提升. 值得注意的是,OpenCL 版本的性能测定包含了 OpenCL 程序运行的所有时间,包括 OpenCL 初始化时间、CPU 和 GPU 间的数据传输时间以及 kernel 的运行时间.

从中我们可以看出,与人脸检测的 CPU 版本的性能相比,本文实现的 OpenCL 版本在两个 GPU 计算平台上处理不同的图片都达到了可观的加速比:在 AMD HD7970 GPU 计算平台上,实现了 11.24～20.27 的性能加速;在 NVIDIA GTX680 计算平台上,实现了 9.24～ 17.62 的性能加速. 由此可以看到,NVIDIA GTX 680 GPU 与 AMD HD7970 GPU 相比,在性能提升方面略有差距. 这主要是因为前者在峰值计算性能和峰值访存带宽这两个主要的性能参数方面要弱于后者.

图 11 和图 12 不仅说明了我们优化框架的有效性,有效地解决了算法负载不均衡的问题;而且说明了虽然两个 GPU 计算平台的架构不同,计算单元的组织也不相同,但是二者都采用了层次式架构,优化技术和方法也大致相同,只要对性能参数进行精心抽取和定义(如 4.3.7 节中的每个 work-group 一次处理的窗口数 N 以及 work-group 内线程处理一次协同处理的窗口数 M 等),是完全可以实现不同 GPU 计算平台间性能移植的.

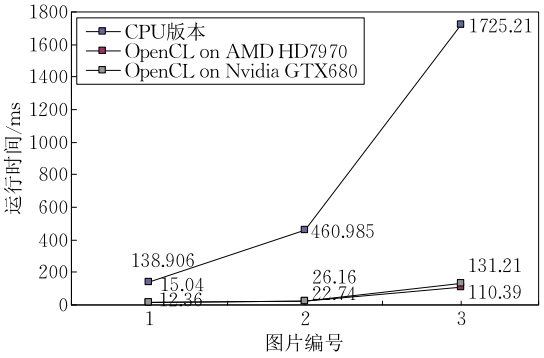


图 11 OpenCL 版本与 CPU 版本性能对比

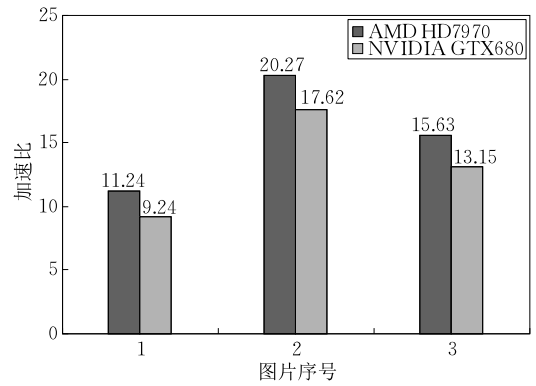


图 12 在两个 GPU 上的性能加速

同时,我们可以看到,对于不同图像,GPU 的加速效果也是不同的. 主要有两方面的原因:第一,GPU 是大规模并行处理器,理论上,图片规模越大,计算量越大,越能充分利用 GPU 计算平台强大的计算能力,加速效果也就越好. 第二,图像的背景和人脸数目不同,算法的总体计算量也不同. 同时,算法在处理不同图片时的负载不均衡的程度也不尽相同,从而进一步影响了 GPU 计算平台对性能的提升效果.

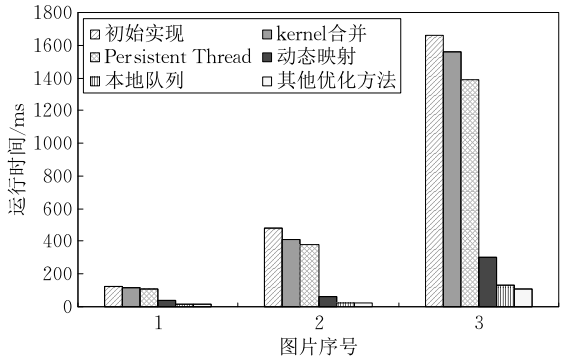
5.3.2 不同优化方法对性能的影响

图 13 显示了以 Naïve 版本为基准,采用不同优化方法后,在两个 GPU 计算平台上所带来的性能提升.

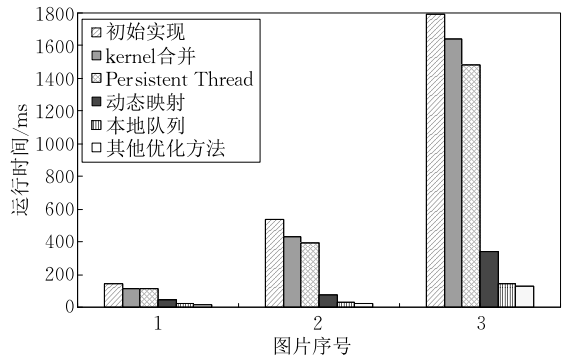
从图 13 中,我们可以看出如下 6 点:

(1)在两个 GPU 计算平台上,不同优化方法带

来的性能提升虽有差异，但趋势大致相同. 这不仅说明了两个 GPU 计算平台在整体架构设计上的统一性，而且说明了在两个 GPU 计算平台上，线程间负载不均衡都是最严重的性能瓶颈，这些都是导致性能降低的主要因素.



(a) AMD HD 7970 GPU



(b) NVIDIA GTX680 GPU

图 13　不同优化方法在两个 GPU 计算平台上的性能提升

(2) 线程与任务的动态映射带来的性能提升最大，注意这里的线程与任务动态映射实际指的是动态映射＋全局队列，因为两者是一个统一整体，没有必要分开说明. 从性能图中可以看出，采用传统 GPU 静态编程模式，work-group 间的负载不均衡会导致性能的极大降低，这是因为图片中绝大部分区域没有人脸，大量的线程可能运行很短的时间就会退出等待，只有少量线程仍在运行. 而线程和任务的动态映射很好地解决了这个问题，因为线程和任务动态映射的核心是根据线程的实际任务负载情况，动态进行任务的分配. 这样就可以让执行大任务的线程，执行的任务数量少一些；执行小任务量的线程，执行的任务数量多一些. 这就从根本上解决了 work-group 间负载不均衡的问题.

(3) 本地队列也有效地提升了算法性能. 这说明了 3 点：第一，work-group 内同样存在着线程间负载不均衡问题，同样会影响程序性能；第二，work-group 内线程间负载不均衡对性能的影响没有 work-group 间的负载不均衡对性能的影响大，这是因为一个 work-group 处理的图像区域毕竟有限，线程间的计算量差距并不会太大；第三，本地队列很好地解决了 work-group 内线程间的负载不均衡问题.

(4) 传统优化方法带来的性能提升并不明显. 这就说明了负载不均衡是人脸检测算法在 GPU 计算平台上的性能瓶颈，传统意义上引起性能下降的因素在该算法上体现得并不明显. 但同时也说明了，只要解决负载不均衡问题，传统优化方法依然后改善 GPU 计算资源的利用率，同样可以带来性能提升.

(5) 虽然 Persistent Thread 和 kernel 合并对性能的提升有限，但是两者是整个优化框架的基础. 对性能影响最大的线程与任务的动态映射机制构建的基础就是 Persistent Thread 和 kernel 合并. 粗粒度并行虽然无法测试对性能的具体影响，但同样作为优化框架的基础，同本地队列一起，克服 work-group 内线程间的负载不均衡问题，从而在一定程度上提升了算法性能.

(6) 通过该算法的优化，说明了只要经过精心优化，具有负载不均衡特征的非规则算法在 GPU 计算平台上也能够取得非常好的加速比. 因此，GPU 不仅对规则的数据级并行能够取得很好的加速效果，对于不规则的任务级并行，只要优化方法得当，也会取得相当可观的性能加速效果. 这无疑会大大扩展 GPU 计算平台的应用场景.

5.3.3　与以往工作的比较

虽然有很多关于 Viola-Jones 人脸检测算法的 GPU 移植工作，但很少有发布出来的代码或者库. 而针对不同图片，人脸检测算法的性能又具有很大的差异性. 因此，在与以往工作的对比上，本文只选取了 OpenCV2.4 库中该算法的 OpenCL 实现版本. 但考虑到 OpenCV 在计算机视觉领域应用的广泛性，这个对比也在一定程度上体现了我们工作的有效性.

图 14 显示了本文实现与 OpenCV2.4 中人脸检测算法的 OpenCL 版本的性能对比. 从图中可以看出，相对于 OpenCV 库的实现，本文实现在两个 GPU 计算平台上，针对不同图片都取得了较大的性能提升. 具体为：在 AMD HD7970 计算平台上，取得了 27.1%～37.9%的性能提升；在 NVIDIA GTX 680 计算平台上取得了 20.6%～31.7%的性能提升.

本文实现性能提升的主要原因是 work-group 间负载不均衡问题的解决：OpenCV 库的 OpenCL

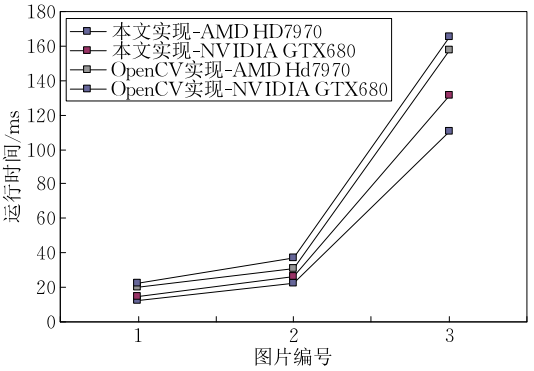


图 14 本文实现与 OpenCV 的性能对比

实现版本中,全局队列依旧采用静态调度的方式,没有很好地解决全局队列的任务调度问题,work-group 间的负载不均衡问题依然没有得到改善. 而本文实现了线程与任务的动态映射机制,全局队列可以根据线程的实际任务负载情况进行任务分配,很好地解决了 work-group 间的负载不均衡问题.

总之,本文提出的优化框架很好的解决了 Viola-Jones 人脸检测算法在 GPU 的实现和优化中,由于负载不均衡导致的性能瓶颈,取得了可观的性能加速比. 该框架的 6 个组成部分,既各司其职,又相互协作,共同提升了人脸检测算法在 GPU 计算平台上的性能. 同时,该框架不仅适用于人脸检测算法,而且对于其他具有类似特征的算法,也具有很好的指导意义和参考价值.

6 结束语

本文详细地讨论了 Viola-Jones 人脸检测算法在 GPU 计算平台上实现和优化的关键方法和技术. 由于人脸检测算法的 GPU 实现存在线程间负载不均衡的非规则特性,导致仅使用传统优化方法无法有效地提升性能. 本文构建了一个针对此类算法的并行优化框架通过 Uberkernel、粗粒度并行、Persistent Thread、线程与任务动态映射、本地/全局队列等优化方法的使用突破了由于线程间负载不均衡导致的性能瓶颈. 实验结果表明,与 OpenCV2.4 中经过高度优化的 CPU 版本在 Intel Xeon X5550 CPU 上的性能相比,优化后的算法在 AMD HD7970 和 NVIDIA GTX680 两个不同 GPU 计算平台上分别达到了 11.24～20.27 和 9.24～17.62 倍的加速比,不仅实现了高性能,而且实现了在不同 GPU 计算平台间的性能移植. 本文采用的优化方法,对其他具有线程间负载不均衡特性的算法在 GPU 计算平台上的实现和优化也具有很好的指导意义和参考价值.

致 谢 感谢王伟俨对本文前期工作的支持;感谢 AMD 公司对本文工作的支持;感谢课题组内其他成员的帮助和支持!

参 考 文 献

[1] Viola P, Jones M. Robust real-time object detection//Proceedings of the 2nd International Workshop on Statistical and Computation Theories of Vision-Modeling, Learning, Computing and Sampling. Vancouver, Canada, 2001: 34-47

[2] Jia Hai-Peng, Zhang Yun-Quan, Wang Wei-Yan, Xu Jian-Liang. Accelerating viola-jones face detection algorithm on GPUs//Proceedings of the 14th IEEE International Conference on High Performance Computing and Communications (HPCC-2012). Livepool, UK, 2012: 396-403

[3] David O, Fernández C, Saeta J R, et al. Real-time GPU-based face detection in HD video sequences//Proceedings of the 2011 IEEE International Conference on Computer Vision Workshops. Barcelona, Spain, 2011: 530-537

[4] Kong Jian-Gang, Deng Yang-Dong. GPU accelerated face detection//Proceedings of the 2010 International Conference on Intelligent Control and Information Processing. Dalian, China, 2010: 584-588

[5] Sharma B, Thota R, Vydyanathan N, et al. Towards a robust, real-time face processing system using CUDA-enabled GPUs//Proceedings of the 2009 IEEE International Conference on High Performance Computing. Kochi, India, 2009: 368-377

[6] Ghorayeb H, Steux B, Laurgeau C. Boosted algorithms for visual object detection on graphics processing units//Proceedings of the 7th Asian Conference on Computer Vision. Hyderabad, India, 2006: 254-263

[7] Tzeng S, Patney A, Owens J D. Task management for irregular-parallel workloads on the GPU//Proceedings of the Conference on High Performance Graphics. Ville, Switzerland, 2010: 29-37

[8] Merrill D, Gariand M, Grimshaw A. Scalable GPU graph traversal//Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Parallel Pragramming. New York, USA, 2012: 117-128

[9] Aila T, Laine S. Understanding the efficiency of ray traversal on GPUs//Proceedings of the Conference on High Performance Graphics. New York, USA, 2009: 145-150

[10] Nasre R, Burtscher M, Pingali K. Data-driven versus topology-driven irregular computations on GPUs//Proceedings of the IEEE 27th International Symposium on Parallel & Distributed Processing. Boston, USA, 2013: 463-474

[11] Cederman D, Tsigas P. On dynamic load balancing on graphics processors//Proceedings of the 23rd ACM Symposium on Graphic Hardware. Ville, Switzerland, 2008: 57-64

[12] Chatterjee S, Grossman M, Sbirlea A, Sarkar V. Dynamic task parallelism with a GPU work-stealing runtime system//Proceedings of the 24th International Workshop on Languages and Compilers for Parallel Computing. Fort Collins, USA, 2011: 203-217

[13] Yan Shen-Gen, Long Guo-Ping, Zhang Yun-Quan. StreamScan: Fast scan algorithms for GPUs without global barrier synchronization//Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. Guangzhou, China, 2013: 229-238

[14] Burtscher M, Nasre R, Pingali K. A quantitative study of irregular programs on GPUs//Proceedings of the 2012 IEEE International Symposium on Workload Characterization. La Jolla, USA, 2012: 141-151

[15] Nasre R, Burtscher M, Pingali K. Atomic-free Irregular Computations on GPUs//Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units. New York, USA, 2013: 96-107

[16] NVIDIA Corporation. NVIDIA's Next Generation CUDA Compute Architecture: Fermi, 2009

[17] NVIDIA Corporation. NVIDIA Kepler GK110 Architecture Whitepaper, 2012

[18] NVIDIA Corporation. NVIDIA GeForce GTX 750 Ti: Featuring first-generation maxwell GPU technology, designed for extreme performance per watt, 2014

[19] AMD Corporation. Accelerated Parallel Processing OpenCLTM, 2014

[20] Jia Hai-Peng, Zhang Yun-Quan, Long Guo-Ping, et al. GPURoofline: A model for guiding performance optimizations on GPUs//Proceedings of the International European Conference on Parallel and Distributed Computing. Rhodes Island, Greece, 2012: 920-932

[21] Sim T, Baker S, Bsat M. The CMU pose, illuminatin, and expression database. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2003, 25(12): 1615-1618

JIA Hai-Peng, born in 1983, Ph. D., assistant professor. His research interests include high performance computing, the method of programming and optimization for many-core computing platform.

ZHANG Yun-Quan, born in 1973, Ph. D., professor. His research interests include high performance computing, parallel numerical software and parallel computing model.

LIANG Yuan, born in 1984, Ph. D., assistant professor. His research interest is parallel computing model.

LI Shi-Gang, born in 1986, Ph. D., assistant professor. His research interests include the research and design of massively parallel algorithm.

Background

Viola-Jones face detection algorithm is widely used in many application domains, such as law enforcement, surveillance, security systems and entertainment. With the development of real-time requirement in these applications, it is necessary to improve the performance of this algorithms, so that it can meet the real-time constrains. On the other hand, GPUs are widely used in various domains as standard computing accelerators because of their increasing computing power and programmability. In this paper, we leverage the compute capabilities of modern GPUs to accelerate Viola-Jones face detection algorithm.

However, it is very challenging to port and optimize Viola-Jones face detection algorithm to GPU computing platforms because of load imbalance problem. Modern GPUs might appear poorly suite for face detection algorithm, despite with high computational throughput. Furthermore, face detection algorithm also represents a class of algorithms which are hard to obtain high performance on GPUs because of their irregular work-loads. Unfortunately, traditional GPU programming wisdom usually guides us on how to efficiently write GPU programs for data level parallelism tasks with regular input and output. While how to write task-level parallelism programs with irregular workloads on GPUs effectively not only have not much material to reference but also are hard to implement.

This paper addresses these problems by presenting an optimization framework, which includes six techniques: warp size work granularity, persistent threads, Uberkernel, the dynamic mapping between thread and task, local queues and global queues. Among them, warp size work granularity can emulate task-level parallelism on GPU without loss in efficiency. Persistent threads can cycle work multiple times through a kernel. Uberkernel can eliminate the overheads of switching kernels and global synchronization. And on these basises, local queue and global queue alleviate the imbalanced computation.

In this paper, we first explore the parallelization of the Viola-Jones face detection algorithm. And then use these six optimization techniques discussed above to implement an OpenCL version which can achieve high performance on both NVIDIA and AMD GPU. Furthermore, we also achieve performance portability among different GUs.

This work is supported by the General Program of National Natural Science Foundation of China (Nos. 61133005, 61272136, 61521092, and 61402441).