中图法分类号：TP312 文献标志码：A 文章编号：1006-8961(2011)02-0152-09
论文索引信息：杨鑫，许端清，赵磊. 基于多核架构的大图像实时浏览技术 [J]. 中国图象图形学报,2011,16(2):152-160

基于多核架构的大图像实时浏览技术

杨鑫，许端清，赵磊
（浙江大学计算机学院,杭州 310013）

摘 要：近年来,随着数据获取设备的不断提高,越来越多的高精度大图像出现在医学、遥感、气象、考古等领域中。这些大图像给使用者带来更多细节信息的同时,也给计算机交互式的显示带来巨大的挑战。设计了一个可扩展的并行工作管道,根据所提出的基于预测的数据管理方法,可以使图像显示和数据导入同步并行的进行;然后,我们基于 CUDA(compute unified device architecture)架构,设计针对大图像的放大缩小算法,利用 GPU 快速的对所显示的图片进行放大缩小处理。我们的算法不需要任何预处理操作,对大图像的浏览操作获得很好的实时性和稳定性,大大提高了浏览的效率。

关键词：多核；大图像；实时

Real-time viewing of large images based on multi-core

Yang Xin, Xu Duanqing, Zhao Lei
(*College of Computer Science, Zhejiang University, Hangzhou* 310013 *China*)

Abstract: In recent years, since data acquisition equipments have been improved, a growing number of high-precision large images appear in the medical imaging, remote sensing, meteorology, archeology and other fields. These images not only provide users with detailed information, but also bring out tremendous challenge to display them interactively. We designed a scalable parallel pipeline, according to our algorithm based on proposed method of data management. Then, we design the algorithm to zoom the large image based on CUDA(compute unified device architecture) framework interactively. Our algorithm does not require any pre-processing operation, can obtain excellent real-time performance and stability, and improve the efficiency of the browser greatly.

Keywords: multi-core; large image; real-time

0 引 言

近年来,随着数据获取设备的不断提高,越来越多的高精度大图像出现在医学、遥感、气象、考古等领域中[1]。这些大图像给使用者带来更多细节信息的同时,也给计算机交互式的显示带来了巨大的挑战。例如,我们使用 Canon 5D 相机在敦煌莫高窟 285 洞窟拍摄的“飞天”佛像全景图数据大小达到了 15 G,远远超过了计算机内存和显存的容量限制,导致了效率低下的 out-of-core 数据[2-3]访问,图片的实时浏览出现了巨大的问题。

以往对大图像浏览的研究大多利用数据访问的时间相关性和空间相关性,通过缓存最近访问的图像数据,最大限度地降低数据从硬盘到内存的延迟时间[4]。但是这种方法有一个明显的缺点,当使用者浏览到一块新的图像数据块时,其还没有在缓存中记录,这样使用者就要等待数据从硬盘调度,从而

收稿日期:2009-07-28;修回日期:2009-09-14
基金项目:国家科技支撑计划项目(2007BAH11B05);浙江省科技计划项目(2008C13079);国家高技术研究发展计划课题项目(863)(2006AA10Z234)。
第一作者简介:杨鑫(1984—),男。浙江大学计算机学院计算机应用专业博士研究生,主要研究方向为图形图像处理、并行计算。E-mail:xinyang@zju.edu.cn。

出现了明显的时间延迟,影响了浏览的实时性。

随着计算机硬件计算能力呈指数速度的增长,尤其是多核技术的出现,使数据的并行处理能力有了巨大的提高。例如,一个标准的 3 GHz duo-core Opteron 处理器已经达到 24 Gflops,一台 PlayStation 3 的 CELL 处理器有 180 Gflops,Inter 公司已经推出了达到 TeraFlop 性能的具有 80 个处理核的处理器[5]。随着 CPU 计算能力的增强,GPU 也变得越来越快,Nvidia 公司推出的 GTX285 图形卡已经可以达到 500 Gflops 的性能,其基于 GT200 架构的图形卡里包含了多达 240 个处理器,通过 SIMT 编程模型实现高性能计算[6];另外,图形卡的可编程性也越来越灵活,Nvidia 公司的 CUDA[7] 编程架构可以使编程者灵活而且充分的利用 GPU 的强大并行计算能力进行算法设计。

图像处理本身就是十分适合并行处理的算法,因此很适合利用这些新的硬件性能[8]。为了充分挖掘硬件的并行计算能力,以实现大图像的实时浏览,我们必须重新设计数据结构和算法:首先,我们设计了一个可扩展的并行工作管道,根据我们所提出的基于预测的数据管理方法,使图像显示和数据导入同步并行地进行;然后,我们基于 CUDA 架构,设计了图像的放大缩小算法,利用 GPU 快速的对所显示的图片进行放大缩小处理。(创新之处)我们的算法对大图像的浏览操作获得了很好的实时性和稳定性,相对于多数算法,我们的算法通过我们所设计的并行工作管道和基于预测的数据管理技术,不需要任何预处理操作,大大提高了浏览的效率;我们通过使用 GPU 所提供的大量多线程并行运算能力,实时的对所浏览图像进行放大或缩小操作;我们的算法具有很好的稳定性,一方面,我们的算法不会因为发生数据等待而在某一时刻出现性能突降的情况,另一方面,我们的算法不受图像大小的影响,对不同大小的图片都取得了相近的性能;同时,我们的算法对处理核的数量也表现出了良好的可扩展性。

虽然本文的算法主要针对大图像数据,但是也可以很容易拓展到其他涉及 out-of-core 数据的应用中。

1　相关工作

已经有不少算法被提出来处理超过内存容量限制的大数据[9-10]。Quicktime VR[11] 将纹理全景图切割成垂直的纹理段来满足 cache 的访问特性。Pharr 等人[12] 根据存储器的页机制将纹理切割成页大小的纹理块,以实现纹理的按需调度。使用这一方法,他们的光线跟踪系统的绘制速度能够比以前提高许多倍。SGI 的 ImageVision 库[13] 也使用了类似的将图像切割成页的机制。Clip-map 方法[14] 尽管不需要将图像切割成块数据,但却过多依赖高级硬件的支持。David Cline 等人[15] 设计了一个 cache 的层次结构,来缓解纹理数据的传输压力。Binns 等人[16] 通过将整个图像数据复制到每个渲染节点上,提出了一个基于集群的图像浏览系统,每个节点都按需从硬盘里读取原始的图像数据。Juxtaview[17] 是第一个使用数据流的基于集群的大图像浏览系统,但是由于其复杂的分布式共享内存协议,它并不能获得图像浏览的实时性。Jiang 等人[18] 基于数据流提出了一个层次存储结构,然而他们并不能很好地预测即将使用的图像数据,致使算法在某一时刻仍然会出现数据访问的延迟。在对图像浏览时所可能遇到的对图像的放大缩小操作,大多数算法都采用了经典的金字塔结构[19],预先生成若干不同分辨率的图片。David Cline 等人[15] 使用了一个四叉树结构来表示不同分辨率的图片,通过树的父子关系来控制数据的流动。

然而,这些算法存在两个明显的不足:首先,这些算法都需要进行大量的预处理操作,这些操作是十分耗时的,大大降低了操作效率;其次,这些算法都是通过设计一些存储层次结构,来缓存已访问的数据,利用数据使用的时间相关性和空间相关性,然而一旦需要访问新的图像数据,就不得不从硬盘调度,产生了明显的 I/O 延迟,使图像的实时浏览性能突然下降。为此,我们提出一种基于预测的数据管理方法,通过预测使用者即将可能浏览的图像数据,利用多核架构下的并行工作管道,将这些预测数据快速的提前预取到内存中,从而避免了数据访问中可能产生的延迟;同时,我们通过使用 GPU 的 CUDA 解决方案实时地对图像数据块进行放大或者缩小处理,从而有效地避免了费时的预处理操作。

2　算法描述

2.1　按需读取

虚拟内存技术可能诱使人们想要把数据全部一次性读进来,这种想法忽视了两个重要的方面:首

先,这些数据并不会同时被使用,而读取这些用不到的数据却浪费了大量的硬盘读取时间和带宽;其次,内存的容量远远小于硬盘,并没有足够的空间来存储我们所需要的大规模数据。另外,一次虚拟存储操作会产生两次 I/O 操作,一次是从硬盘文件读取数据,一次是进行页面置换操作,开销是很大的。这里,我们采用内存映射技术,将数据直接映射到进程的地址空间中去,如同整个文件加载到了内存中一样,使访问速度明显加快。另一方面,为了减少无用数据的读取,我们采用按需读取的载入方式。这里,如何预测出想要读取的数据是一个难点问题,我们将在下面详细给出我们的方法。

还有一个比较重要的问题就是我们一次应该读取多少数据。为了使数据从硬盘到内存的载入速度加快,一种极端的做法可能是一次只读取一个像素。这样做尽管减少了进行一次数据传输的时间,但却大大增加了传输的次数,产生了更大的传输花费。这里,我们主要从 3 方面考虑设计数据读取块的大小:1)由于 CPU 最终要通过其片上 cache 读取数据,为了增加其 cache 命中率,我们应尽量使每次读取的数据块大小不超过 cache 的容量;2)尽管图像是以按行方式存储的,但由于图像浏览时向任何方向移动的概率都是相同的,如果以行为单位作为数据块的形状,可能会出现一行数据中只有前半行被访问而后半行被浪费的情况,因此我们所选取的数据块形状应该是等方性的,为了操作上的简单,我们选取正方形作为我们的数据块形状;3)为了加快内存访问的速度,数据块的大小和尺寸应该均为 2 的幂次方。

需要指出的是,与大多数算法不同,我们的算法不需要预先对数据文件进行分块处理和逻辑组织,而是在需要读取时通过使用内存映射技术根据图像块在整个图像文件的相对位置关系直接读取,从而实现真正的“按需读取”。

2.2 基于预测的数据管理技术

在有关 out-of-core 数据的各种算法中,经常需要预测出即将可能使用的数据,从而提前将其从硬盘载入内存,避免可能出现的等待延迟,同时置换出淘汰的数据。然而,如何准确地预测出下一步可能需要使用的数据,至今也没有效果较好的通用算法。多数算法都是根据数据访问的时间相关性和空间相关性,假定当前访问的数据在接下来几次操作中仍然可能会被访问,利用存储层次结构和缓存机制,保存这些刚刚被访问的数据。然而,当操作需要访问新的数据块,即前几次操作中未使用过的数据块时,就不得不从硬盘读取,从而产生明显的等待延迟,影响了图像浏览的实时性。这里,我们根据图像浏览的特点,提出了一种基于预测的数据管理方法,有效地避免了访问数据时可能出现的等待延迟。

下面我们详细给出基于预测的数据管理算法。我们设立了一个缓存池,用来存储当前活跃的数据块以及我们预测出的即将可能访问的数据块,如图 1 所示,我们的缓存池中包含有 8×3 个数据块,初始状态下,这 24 个数据块对应着图像文件的一块连续区域。假设当前浏览区域是 A 图像块,而且当前的浏览方向是从右向左横向浏览,对于数据块 A 来说,除了数据块 E、D、F,其他所有在缓存池中的数据块都是图像文件中与图像 A 相邻的区域,由于这些数据已经处于内存中,而且大小均不超过 cache 的容量,使得访问这些数据几乎没有延迟。由于缓存池的大小是不变的,随着浏览操作的进行,我们必须要淘汰一些可能不再使用的数据而导入一些新的数据。目前,多数算法都采用操作系统中虚拟内存机制里一些页面置换算法所采用的方法,比如 LRU、FIFO 等方法。这里,我们选择数据块 E、D、F 作为淘汰的数据块,置换成我们根据浏览行进方向所预测新的数据。

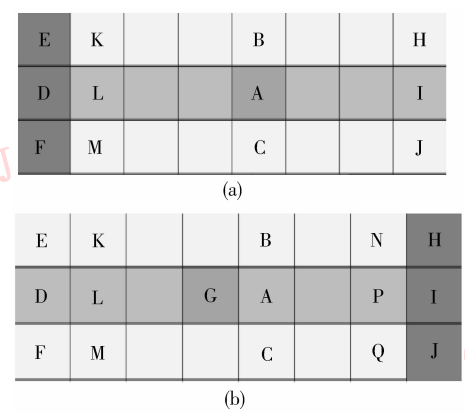

图 1 基于预测的图像浏览过程

Fig. 1 View large image according to proposed method

根据浏览方向的特点,我们在进行图像浏览时,在某一方向上的操作具有持续性的概率很大,也就是说,假如我们在对图像进行横向浏览时,很可能在接下来的若干步操作中仍然是横向浏览,很少会出现一步横向浏览一步纵向浏览的交叉操作,因此按照我们的推断,数据块 K、M、L 在图像中相邻的左侧

区域将是最有可能即将被访问的区域,于是我们将数据块 E 置换成在图像文件中与数据块 K 左相邻的图像数据,数据块 D、F 也进行类似的处理。如图 1 所示,当我们的浏览区域由数据块 A 行进到数据块 G 时,所选的数据块淘汰区域也相应的按照同样的方向行进。数据块 E、D、F 已经位于缓存池的最左端,因此我们转到缓存池的最右端,将数据块 H、I、J 作为数据块淘汰区域,同时导入所预测的数据。需要注意的是,经过预测,数据块 H 置换的是数据块 E 在图像文件中的相邻左侧图像块,而数据块 N 虽然在缓存池中是与数据块 H 相邻的,但是在图像文件中,它们却是没有任何相对位置关系的,类似的,数据块 I、J 也进行同样的处理。

如图 2 所示,当图像的浏览方向发生改变,即从横向浏览变成纵向浏览时,浏览区域将从数据块 G 移到数据块 O,而需要预测的区域则大大增多,数据块 E 应置换成在图像文件中数据块 F 下方的数据,数据块 K、R、S、B、T、N、H 也要做相应的更新。尽管我们采用了一些策略,如渐进式更新,甚至是随着浏览的进行,舍弃一些数据块的更新,但是经过测试,我们发现这种方法仍然会给传输带来了很大的负担,因此我们将在满足使用的前提下尽量使所预测的数据量达到最小。如图 3 所示,我们进一步优化了我们的缓存池,图 3(a)—(d)给出了从数据块 A→B→D→C的图像浏览过程。与图 1 所示的缓存池不同的是,优化过后的缓存池中并不都是有效的数据块,如图 3(a)中的数据块 H、D、L。如图 4 所示,当图像浏览区域由数据块 A 移到数据块 J 时,我们只更新数据块 E、F、G,而数据块 H、D、L 并不作处理。这里所说的无效数据块,只是不再进行数据的更新,但是由于里面仍然存有以前曾经浏览过的图像数据,因此仍起到了数据缓存的作用。通过进行这样的优化,我们大大减少了数据的量,节省了传输时间。为了防止过多频繁而无效的数据读取操作,我们设立了一个阈值,通常设为数据块的长度或者宽度,只有当图像浏览操作在某一方向上连续移动的位移绝对值超过这个阈值,我们才进行数据块的预测和置换更新工作。

正如前面所说的图像浏览的特点,即在某一方向上的操作具有持续性的概率很大,我们在设计缓存池时也充分考虑到了这一特性。我们的缓存池是一个长方形的 2 维结构,但是我们将根据当前图像浏览的方向性,动态的调整缓存池的长宽,也就是

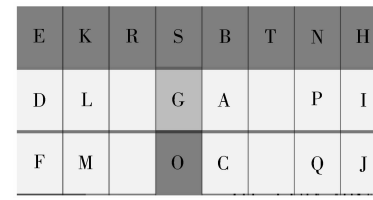

图 2　图像纵向浏览过程

Fig. 2　View large image as vertical

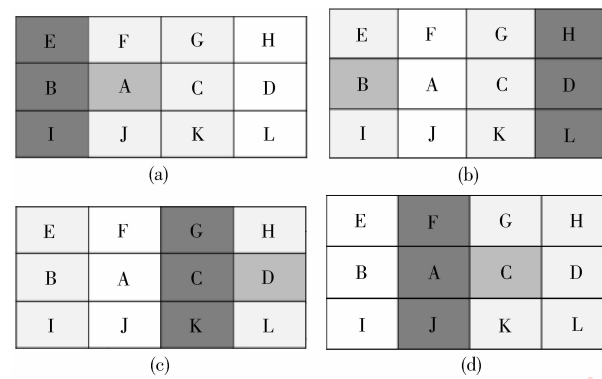

图 3　优化后的从数据块 A→B→D→C 的图像浏览过程

Fig. 3　Optimized image browsing from the data blocks A→ B→ D→ C

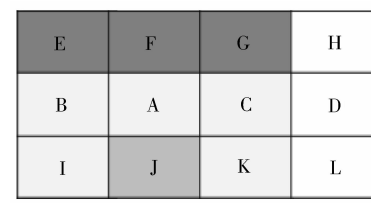

图 4　优化后的纵向浏览过程

Fig. 4　Optimized image browsing as vertical

说,当图像做横向浏览时,我们的缓存池的长度会大于宽度,而当图像做纵向浏览时,我们的缓存池的宽度则会大于长度。当然,缓存池的长宽变换并不是一步完成的,而是在并行工作管道中,随着图像浏览方向的变化,逐渐地分几步操作完成的,渐进式的完成长宽变换。

2.3　可扩展的并行工作管道

如上所述,基于预测的数据管理方法尽管可以有效的将浏览即将可能需要的数据块提前载入内存,但却需要在短时间内进行大量的硬盘读取操作,这对于通常的单线程操作是一个巨大的挑战。但是,随着多核技术的出现,程序的并行执行能力得到了革命性的增强,这同时也让数据的并行快速读取

有了可能。一开始设计并行工作管道时,我们本打算采用 GPU 多线程强大的并行计算能力,但经过试验我们发现,由于 GPU 主要是通过发射大量的线程,使某些线程在等待数据的同时,其他线程可以继续执行计算,从而充分利用计算单元,隐藏访存延迟[6],而我们的算法是一个数据密集型的操作,不是计算密集型的操作,因此 GPU 的工作机制并不适合我们的算法。我们采用的是 CPU 的多核技术,而且由于我们的操作都是对不同的数据进行同样的操作,因此我们还使用了 SIMD(single instruction multiple data)技术来进一步扩大程序执行的并行度。

我们的并行工作管道主要是并行地完成两个任务:图像显示和数据读取。通过使用 Intel 的 VTune 工具[20],我们发现,程序执行的热点在于数据的读取上。因此,如果有 n 个可以并行执行的线程,我们只使其中的一个线程去处理图像的显示操作,而其余 $n-1$ 个线程均进行数据的读取操作。如图 5 所示,显示操作与读取操作并行执行,尽管显示操作的执行时间要少于读取操作的执行时间,但是前面已经说过,只有当图像浏览操作在某一方向上连续移动的位移超过某一阈值时,我们才进行数据块的预测和置换更新工作,也就是说,显示操作的执行次数要多于读取操作的执行次数,这样就平衡了两种操作的速度差异,避免了显示操作因数据等待而可能出现的停滞。当需要进行读取操作时,即达到阈值要求时,显示操作线程将向读取操作线程组发送一个执行信号,读取操作线程组收到这个信号后,组内各个线程开始并行地读取数据,当某一线程完成其读取任务后,它将查看执行信号是否依然有效,如果仍然有效,则此线程将继续进行新的数据读取工作,从而保证所有线程均一直处于工作状态。当所有的读取任务执行完毕后,执行信号将会被设置成无效状态。我们的并行工作管道是可扩展的,它不针对某个特定的处理核数或者线程数,只要发现存在可用的空闲线程,其将会都被加入到数据读取的线程组中。

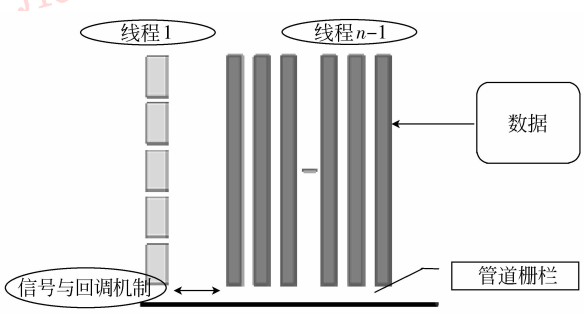

图 5 并行工作管道

Fig. 5 Parallel pipeline

2.4 对图像的放大缩小操作

在图像浏览过程中,对浏览区域进行放大或者缩小是一种基本的操作,尤其对高精度的大图像来说,一些细节信息必须要通过放大操作来观察。传统的方法大多需要经过预处理以对欲浏览图像生成一系列不同分辨率的图片,这种方法显然是耗时而且低效的。由于对图像的放大缩小处理都是对某一局部区域几个像素的计算,与多数像素都是无关的,因此很适合使用大量的线程进行并行处理。

我们使用 Nvidia 公司推出的 CUDA 解决方案来实时的生成浏览区域所对应的一系列放大缩小图片。Nvidia 公司的 GTX285 图形卡有 240 个处理器,高达 141 GB/s 的吞吐能力,这为程序的并行执行提供了很好的支持。在 CUDA 架构中,并行执行的程序称为核(kernal),以网格(grid)的形式执行,一个网格中包含有大量的块(block),大量的线程以块作为组织单位,每个块里包含有一定数量的线程,只有在同一个块内的线程才能进行通信,不同的块里的线程是无法直接进行通信的。块里的线程以 warp 作为执行单位,warp 宽度为 32,同一个 warp 内的线程执行同一条指令,half-warp 是存储操作的基本单位。CUDA 架构提供了容量较大的可供数据存储的全局存储区(global memory),但是在非对齐模式下访存花费可高达 500 个时钟周期;同时还有一种可供 block 内线程通讯和存储使用的共享存储区(shared memory),在不存在 bank conflicts 的情况下,访存花费仅为 1 个时钟周期,但容量仅为 16 kB。很明显,对图像进行放大缩小操作将会有大量的访存操作,如果使用全局存储区将会有明显的延迟,但是共享存储区的容量很小,也无法满足我们对图像数据的存储要求。因此,我们选择 CUDA 架构提供的纹理单元作为存储载体,尽管纹理单元也属于全局存储区的一部分,但是它具有缓存功能,对数据量较大时的随机数据访问或者非对齐访问有良好的加速效果。

下面,我们以缩小操作为例,详细说明我们的算法。我们根据浏览区域的图像分辨率来决定创建线程的数量,例如如果分辨率是 1 024 × 768,则我们将创建 1 024 × 768 个线程,每个线程用来计算所生成

图片中的一个像素,这个像素是对原图像中几个邻近像素根据权值计算后的结果。随着图像的逐渐缩小,所采样的邻近像素的间隔将会越来越大,意味着邻近像素间所包含的像素越来越多,由于 GPU 的存储能力十分有限,这种情况下我们不能把所有的像素数据都导入到 GPU 中去。另一方面,根据我们的测试,假设几个邻近像素分别为 $q_1,q_2,\cdots,q_n$,其权值分别为 $w_1,w_2,\cdots,w_n$,令 $w_1,\cdots,w_{n-1}$ 均为 0,w_n 为 1,根据这样的权值计算出来的结果尽管从图像质量上不如以前,但是从图像浏览的角度上看,是不影响其最终的观看效果的。因此,我们可以根据放大或者缩小的倍数,当达到一定倍数以后,根据一定的步长,将一些像素点权值设置为 1,另一些像素点权值设置为 0,只导入权值为 1 的像素数据进入 GPU。需要注意的是,当放大到一定倍数时,图像的浏览区域只是整幅大图像的一小块区域,其数据量大小已经满足 GPU 的存储要求,而这时候图像的细节信息对于浏览来说也是很必要的,所以这时我们将会向 GPU 导入所有像素信息,以生成高质量的图片。

在处理各种图片的过程中,我们也遇到了一些每个像素只占 3 个字节的图片,即只有 R、G、B 3 个分量,这种存储方式对于纹理单元的访问是很不友好的,因为它会导致非合并访问(uncoalesced access)的访存行为,使执行效率降低。在这种情况下,我们将以 4 个像素为一组,按组访问,然后将其放入共享存储区中,再按照上面所讲的方法进行再处理。需要注意的是,这种方式下访问的顺序很重要,同一个 half-warp 内的线程的访存地址一定要连续,以达到从纹理单元到共享存储区的合并访问(coalesced access)。

3　实验结果

我们的算法已经成功地实现在一台配有一颗 Intel Xeon 3.7 GHz 的 4 核 CPU,一颗 Nvidia GTX285(1 G 显存)的 PC 上。我们选择了在敦煌莫高窟 66 号洞窟拍摄的一幅大小为 4.5 G,分辨率为 45 000 × 25 000 的五台山全景图作为我们的测试图片,图 6 所示的是此全景图的一个局部图。我们所生成的图片,即使用者可见的浏览窗口,其分辨率为 1 024 × 1 024。我们的浏览操作的步长设为每步移动 50 个像素长,最多可以放大或缩小 50 倍,初始状态时,我们的浏览区域显示的是没经过任何处理的原始图像的一小块区域。为了更好的显示出我们的算法所获得的效果,我们对比了文献[18]的大图像浏览算法,同时我们也给出了从硬盘读取数据进行图片显示所需的时间,之所以给出这个时间是为了说明以往多数算法的一个致命的缺陷,即当读取缓存中没有的新数据时,进程要被悬挂,等待数据从硬盘调度,而我们的算法通过在并行工作管道中使用基于预测的数据管理方法,成功地避免了这种情况的发生。

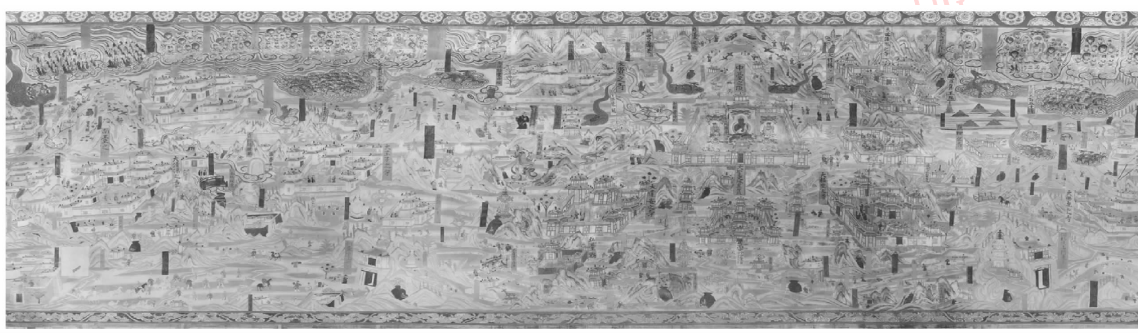

图 6　五台山全景图

Fig. 6　Wutaishan panorama

实验结果中的延迟时间是指完成一次浏览行进操作或者放大缩小操作平均所需要的时间。由表 1 可以看出,相对于文献[18]方法,我们的方法提高了近 50%,更重要的是,我们的方法不存在某一时刻的性能突降,而且也不需要耗时的预处理操作。为了方便比较,我们每 5 帧图像取一个平均延迟时间,如图 7 所示,文献[18]方法的延迟时间周期性的发生突变,原因在于发生突变的时刻缓存中没有所需的数据,而我们的算法所产生的延迟时间则一直比较平稳,没有大的抖动,在某些时刻,会有一些

比较小的抖动,这是由于图像浏览的行进方向突然发生改变,使在并行工作管道里的数据传输量突然增多,而且缓存池结构也要进行一些动态的调整。我们在图 8 中对图像浏览中放大缩小操作所产生的延迟时间与文献[18]方法作了比较,同样的,我们每 3 帧图像取一个平均延迟时间,多数时刻我们的方法所产的延迟时间是文献[18]方法的一倍,尽管我们所产生的延迟时间较多,但是对于图像浏览来说,这样的延迟时间仍然是可以满足实时性的要求的,而且我们的方法有两个很大的优势:首先,我们不需要进行耗时的预处理操作,只有在对 GPU 设置工作环境时需要大约 172 ms 左右的时间,而这样短的时间是可以忽略不计的;其次,文献[18]方法当切换到一个新的分辨率图像时要重新导入数据,产生剧烈的性能下降,而我们的方法所产生的性能则一直比较平稳,只是在某些时刻会产生一些峰值,产生这些峰值的原因是由于当我们的图像浏览缩小到一定倍数时,我们不会把所有的图像数据都导入到 GPU,而是根据取样要求选择一部分数据导入 GPU,而对这些数据的选择过程会增加一些寻址花费,也可能会产生访存冲突,因此增加了生成图片的时间。

我们的算法之所以能达到实时浏览的性能,很大程度上依赖于并行工作管道的并行工作能力,图 9给出了配有不同数量的处理核的并行工作管道每次处理图像浏览操作所产生的不同的延迟时间。可以看出,我们的并行工作管道是可扩展的,在一定范围内,随着处理核数目的增多,完成一次浏览行进操作平均所需要的时间逐渐减少,但是由于目前的硬盘的读写速率仅为 35 MB/s 左右,带宽也仅为 100 MB/s 左右,因此当处理核增加到一定数量时,必然会出现数据读取的瓶颈,因此当处理核数量增加到 32 时,所获得的效果并不理想。

表 1 几种方法完成一次浏览行进操作平均所需要的时间

Tab.1 Latency when performing a panning

	硬盘读取	文献[18]方法	本文方法
延迟时间/ms	1 468	16	10

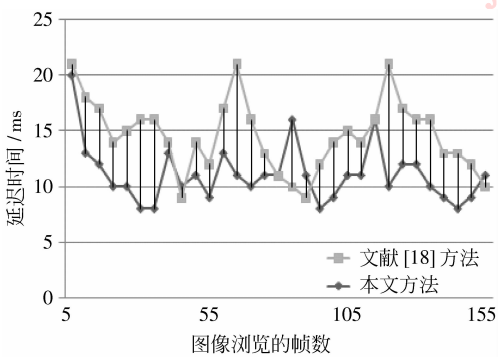

图 7 图像浏览行进操作所需的延迟时间

Fig. 7 Latency between frames when performing panning

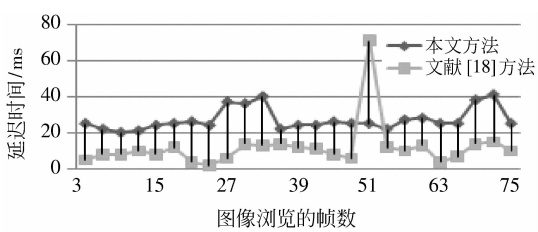

图 8 图像浏览进行放大缩小操作所需的延迟时间

Fig. 8 Latency between frames when performing zooming

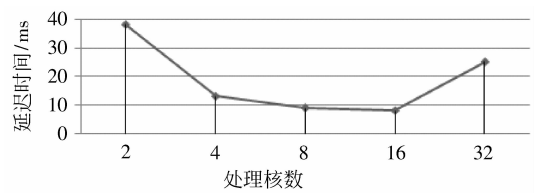

图 9 不同处理核数所产生的延迟时间

Fig. 9 Latency with number of cores

为了更好地测试我们算法的稳定性,我们同时也测试了另一幅在敦煌莫高窟 285 洞窟拍摄的全景图,其大小为 11.5 G,分辨率为 58 000 × 45 000。正如图 10 所示的那样,我们的算法对两幅测试图像都表现出了大致一样的延迟时间,说明我们的算法在对图像作行进浏览时不受图像大小的影响。但是正

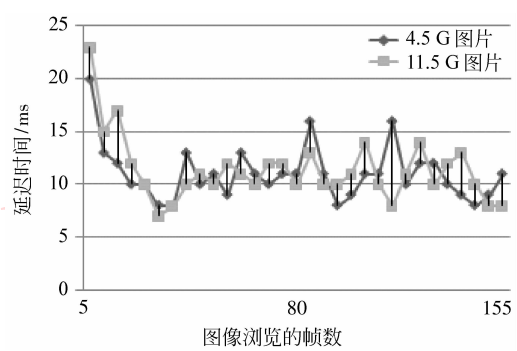

图 10 两幅图像进行行进浏览操作所产生的延迟时间对比

Fig. 10 Latency between frames when performing zooming for different size of pictures

如图 11 所示,在对图像进行缩小操作时,如上面所说,我们不会把所有的图像数据都导入 GPU,而是根据取样要求选择一部分数据导入 GPU,而对这些数据的选择过程会增加一些寻址花费,也可能会产生访存冲突,而图像越大这些花费也就越大,访存的冲突可能性也越大,因此 11.5 G 的图片比 4.5 G 的图片在某些时刻产生了更多的延迟时间。

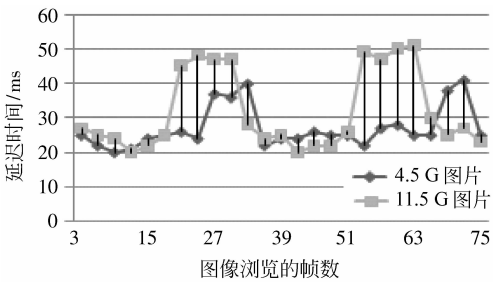

图 11　两幅图像进行放大缩小操作所产生的延迟时间对比

Fig. 11　Latency between frames when performing zooming for different size of pictures

4　结　论

大图像的浏览操作获得了很好的实时性和稳定性,相对于多数算法,我们的算法通过所设计的并行工作管道和基于预测的数据管理技术,不需要任何预处理操作,大大提高了浏览的效率;我们通过使用 GPU 所提供的大量多线程并行运算能力,实时地对所浏览图像进行放大或缩小操作;算法具有很好的稳定性,一方面,不会因为发生数据等待而在某一时刻出现性能突降的情况,另一方面,不受图像大小的影响,对不同大小的图片都取得了相近的性能;同时,对处理核的数量也表现了良好的可扩展性。

尽管算法主要针对大图像数据,但是也可以很容易拓展到其他涉及 out-of-core 数据的应用中,比如在图形绘制中,复杂模型常常具有大量的点面数据和纹理数据[21],我们的并行工作管道和基于预测的数据管理方法可以用到对这类问题的处理上。

下一步我们打算将图像的解压缩加入到我们的并行工作管道中,通过大量的工作线程并行的对图像进行解压缩。由于图像压缩后使数据的存储和传输更加有效[22-23],因此将会进一步增强图像浏览的实时效果。另外,我们将继续优化缓存池结构,改进基于预测的数据管理方法,当前我们的方法消耗了过多的带宽,由于 Nvidia 公司最新推出的CUDA 2.2 版本中支持在 GPU 里直接操纵 CPU 内的数据[24],因此我们可以省去数据从 CPU 到 GPU 的导入过程,这大大加快了算法的处理效率。同时,我们也希望我们的算法能在 Inter 公司即将推出的 Larrabee 处理器[25]中获得出色的性能。

参考文献(References)

[1] Pan Lei, Gu Lixu, Xu Jianrong. Implementation of medical image segmentation in CUDA [C]//Proceedings of the 5th International Conference on Information Technology and Application in Biomedicine. Birmingham, UK: IEEE CS Press, 2008: 82-85.

[2] Brian Budge, Tony Bernardin, Jeff A Stuart, et al. Out-of-core data management for path tracing on Hybrid resources [J]. Computer Graphics Forum, 2009, 28: 385-396.

[3] Rutt B, Kumar V S, Pan T, et al. Distributed out-of-core preprocessing of very large microscopy images for efficient querying [C]//IEEE International conference on Cluster Computing. Boston, Massachusetts, USA: IEEE CS Press, 2005: 1-10.

[4] Darwyn Peachey. Texture on demand [DB/OL]. (1990-8-01) [2009-8-01]. http://graphics.pixel.com/library/TOD.

[5] Intel Company. Intel Technology [EB/OL]. (2009-3-01) [2009-7-01]. http://www.intel.com/go/terascale/.

[6] Lindholm E, Nickolls J, Oberman S, et al. A unified graphics and computing architecture [C]//IEEE in Micro. Lake Como, Italy: IEEE CS Press, 2008, 28: 39-55.

[7] Nickolls J, Buck I, Garland M, et al. Scalable parallel programming with cuda [C]//ACM SIGGRAPH 2008 Classes. Los Angeles, California, USA: ACM Press, 2008, 6: 40-53.

[8] Yang Zhiyi, Zhu Yating, Pu Yong. Parallel image processing based on CUDA [C]//Proceedings of 2008 International Conference on Computer Science and Software Engineering. Wuhan, China: IEEE CS Press, 2008, 3: 198-201.

[9] Andrew C Beer, Maneesh Agrawala, Navin Chaddha. Rendering from compressed textures [C]//Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. New York, NY, USA: ACM Press, 1996: 373-378.

[10] Tao Ni, Greg S Schmidt, Oliver G Staadt, et al. A survey of large high-resolution display technologies, techniques, and applications [C]// Proceedings of the IEEE Conference on Virtual Reality. Alexandria, Virginia, USA: IEEE CS Press, 2006: 223-236.

[11] Shenchang Eric Chen. Quicktime VR-an image-based approach to virtual environment navigation [C]//SIGGRAPH 95

Conference Proceedings. New York, NY, USA: ACM Press, 1995: 29-38.

[12] Matt Pharr, Craig Kolb, Reid Gershbein, et al. Rendering complex scenes with memory-coherent ray tracing [C]// SIGGRAPH 97 Conference Proceedings. New York, NY, USA: ACM Press, 1997: 101-108.

[13] Jacke Neider, CS Tillman. Image Vision Library Programming Guide[M]. CA, USA: Silicon Graphics, 1992: 007-020.

[14] Montrym John S, Daniel R Baum, David L Dignam, et al. A real-time graphics system [C]// SIGGRAPH 97 Conference Proceedings. New York, NY, USA: ACM Press, 1997: 293-301.

[15] David Cline, Parris K Egbert. Interactive display of very large textures[C]//Proceedings of the Conference on Visualization. Durham, NC: IEEE CS Press, 1998: 343-350.

[16] Binns J, Papka M E, Stevens R. Development of cluster-based image viewer [C]//High Performance Distrubuted Computing Conference. Edinburgh, Scotland, UK: IEEE CS Press, 2002: 450-454.

[17] Krishnaprasad N, Vishwanath V, Venkataraman S, et al. View-a tool for interactive visualization of large imagery on scalable tiled displays[C]//Proceedings of IEEE Cluster. San Diego, California, USA: IEEE CS Press, 2004: 411-420.

[18] Jiang Zhongding, Luo Xuan, Mao Yandong, et al. Interactive browsing of large images on multi-projector display wall system [C]//Proceedings of 12th International Conference on Human-Computer Interaction. Beijing, China: Springer-Verlag, 2007: 827-836.

[19] Williams L. Pyramidal parametrics [C]//Proceedings of SIGGRAPH83. New York, NY, USA: ACM Press, 1983: 1-11.

[20] IntelCompany. IntelTechnology [EB/OL]. (2008-6-01)[2009-4-01]. http://www.intel.com/cd/software/products/apac/zho/275878.htm.

[21] Zhou Kun, Hou Qiming, Ren Zhong, et al. RenderAnts: interactive REYES rendering on GPUs[J]. ACM Transactions on Graphics, 2009, 28: 5-15.

[22] Stefan Lietsch, Paul Hermann Lensing. GPU-supported image compression for remote visualization- realization and benchmarking [C]// Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 2008: 658-668.

[23] Stefan Lietsch, Oliver Marquardt. A CUDA-supported approach to remote rendering[C]//Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 2007: 724-733.

[24] Nvidia Company. NvidiaTechnology [EB/OL]. (2008-7-01) [2009-5-01]. http://forums.nvidia.com/index.php?showtopic=97333.

[25] Seiler L, Carmean D, Sprangle E, et al. A many-core x86 architecture for visual computing [J]. ACM Transactions on Graphics, 2008, 27: 26-35.