

虚拟化环境下多 GPU 并行计算研究

闵 芳¹, 张志先², 张玉洁²

(1 南京航空航天大学 金城学院, 江苏 南京 210016; 2 南京航空航天大学 自动化学院, 江苏 南京 210016)

摘 要: 针对大规模计算任务场景, 提出在虚拟化环境下多 GPU 并行计算的实现方案, 使用多线程或流处理的方式实现多 GPU 并行计算, 并分析 GPU 多层次存储结构、传输等方面内容, 采用经典的蒙特卡罗方法这一具有代表性的科学计算实例进行实验验证。

关键词: GPU 通用计算; 虚拟化; 并行计算; 蒙特卡罗

中图分类号: TP319

文献标识码: A

文章编号: 1000-7180(2016)03-0069-07

Multi-GPU Parallel Computing under Virtualization

MIN Fang¹, ZHANG Zhi-xian², ZHANG Yu-jie²

(1 JinCheng College, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China;

2 College of Computer Science and Technology, Nanjing University of
Aeronautics and Astronautics, Nanjing 210016, China)

Abstract: The solution of multi-GPU parallel computing under virtualization is proposed for the large-scale computing tasks. The multi-threads and flow mode are supported to implement the multi GPU cooperative computing. The multi-level storage structure for GPU, transmission and other aspects are analyzed to accelerate the program. We make experimental verification of representative scientific computing examples such as Monte Carlo method. Experimental results show that with the increase of the calculation scale, it can achieve a speed-up ratio close to the number of GPU.

Key words: general-purpose computation on GPU; virtualization; parallel computing; monte carlo

1 引言

目前 CUDA 中没有针对多 GPU 的 API, 且现有的 GPU 虚拟化解决方案大多针对单块 GPU 设备, 在多 GPU 环境下, 应用程序使用多 GPU 进行加速的研究较少. 为了能够更加充分地利用多块 GPU 设备智能地扩展应用性能, 加快工作流程, 本文根据 CUDA 规范, 将多线程技术引入到 GPU, 实现了在虚拟化环境下多线程控制多 GPU 的并行计算; 对于在主机端只有单个主线程的情况, 实现了在虚拟化环境下基于流的多 GPU 并行计算, 使得单个主线程也可以访问到所有 GPU 设备, 并且通过流的方式, 使得传输和计算并行执行^[1-2]. 在虚拟化多 GPU 环境下, 为了优化程序的设计, 充分利用 GPU 强大的计

算能力, 本文分析了 GPU 多层次存储结构, 并优化存储器间的传输带宽.

2 GPU 虚拟化

如今 I/O 外设复杂多样, 对 I/O 的虚拟化一直都是 VMM (Virtual Machine Monitor, 虚拟机监视器) 面临的主要瓶颈. GPU 属于 I/O 设备中较为特殊的设备, 学术界纷纷致力于 GPU 虚拟化技术的开发^[3].

在通用计算方面, 主要是基于 CUDA 的解决方案, 例如 vCUDA^[4]、Gvim^[5]、gVirtus^[6]、rCUDA^[7] 等等. vCUDA 允许在虚拟机中运行的 CUDA 程序使用硬件 GPU, 从而达到加速的效果. GVim 基于 XEN 系统, 允许虚拟机通过虚拟机和特权域之间的通讯

通道访问 GPU 设备,从而为虚拟机访问 GPU 提供了支持. gVirtuS 和 vCUDA、GVim 使用类似的机制,即都是使用虚拟机中前端和特权域后端的前后端通讯方式,虚拟机前端截获 CUDA API 调用,然后转发到特权域后端执行,这是比较常见的虚拟化方式. rCUDA 是一个远程执行 CUDA 的框架,服务器与客户端之间的通信直接采用 socket 方式而不是 RPC 方式,在效率上相对 vCUDA 有所提升.

3 GPU 多层次存储技术与优化设计

3.1 GPU 存储层次分析

为了优化应用程序的性能,CUDA 使用了多层次存储器技术,不同类型的存储器面向不同应用中的数据结构,对其合理利用可以极大地改善程序的运行效率,减少数据传输时间^[8]. GPU 设备内存拥有复杂的存储层次,可以分为三层:每个线程内部都拥有自己的一块局部存储器(Local memory)和寄存器(Registers);在每一个 Block 中,存在着一块共享存储器(Shared Memory),提供该 Block 内的所有线程共用;能够被一个应用程序中所有线程访问的全局存储器(Global Memory). 另外,板载显存上还存在着可被所有线程访问的只读存储器:常量存储器(Constant Memory)和纹理存储器(Texture Memory)^[9].

GPU 片上含有大量的寄存器资源,根据硬件不同,每个流多处理器(SM)可供所有线程使用的寄存器空间大小也不同,它为大规模线程的并发提供了零开销切换. 每个线程的每个变量都会占用一个寄存器,当线程中的寄存器不够用时,会自动溢出到片外局部存储器中. 在线程执行过程中由于增加了大量的全局访存事物,会使机器速度大幅度减慢,降低应用性能. 共享存储器是一种可受用户限制的一级缓存,属于片上存储器,速度仅次于寄存器,支持在一个 Block 中临时计算结果的重用或共享. 当线程束中的所有线程同时访问相同地址的存储体时,共享内存会触发一个广播机制到线程束的每个线程中,对数据重复利用或者线程之间有共享数据时性能会得到很大的提升. 常量内存是为多个线程只读类型的广播操作而优化的内存,通过广播机制,可以快速地数据分配到线程束的多个线程中. 纹理存储器通常用来做局部优化,即它希望数据提供给连续的线程进行操作. 全局存储器是一个 CPU 和 GPU 都可以对其进行写操作的存储. 全局存储器属于片外存储器,对其访问需要消耗大量的时间,存储速度

非常慢^[10].

3.2 异构平台中传输带宽优化设计

尽管 GPU 拥有很强的计算能力,但有限的传输带宽在性能优化方面仍然是一大瓶颈. 通常情况下,在可分页内存中数据从主机端拷贝到设备端,拷贝速度将受限于 PCI-E 传输速度和系统前端总线速度相对较低的一方. 在某些系统中,这些总线在带宽上存在着巨大的差异. 因此当在 GPU 和主机间复制数据时,这种差异会使页锁定主机内存的性能比标准可分页内存的性能要高大约 2 倍. 即使 PCI-E 的速度与前端总线的速度相等,由于可分页内存需要更多一次由 CPU 参与的复制操作,因此会带来额外的开销. 合理分配使用页锁定内存能够提高传输带宽,但是由于页锁定内存不能交换到磁盘上,在应用程序中使用时都需要分配物理内存,故与可换页内存相比,系统内存将会更快地耗尽,所以使用完毕后,应立即释放. 对于使用多 GPU 的应用程序,如果支持统一虚拟寻址(Unified Virtual Addressing, UVA),指定所有页锁定内存为可共享的是一个非常好的方式.

一般情况下 CPU 只能访问主机内存, GPU 只能访问设备内存. 零拷贝内存也是页锁定内存,它被映射到 GPU 地址空间,在内核程序中可以直接对其读取或写入. 采用零拷贝内存和写结合策略可以有效提高带宽的利用率. 对于两个输入缓冲区,运行时可将内存分配为“合并式写入(Write-Combined)”内存,虽然这种方式不能改变应用程序本身的性能,但却可以提升 GPU 读取内存时的性能. 对于计算密集型程序,计算密度很大,采用全局内存合并方法隐藏 PCI-E 延迟. 采用零拷贝内存,将内核传输操作分解为更小的块,通过流水线的方式执行,减少整体时间.

4 虚拟化环境下多 GPU 并行计算的实现方案

4.1 系统架构

本文在虚拟化环境下多 GPU 并行计算的系统架构如图 1 所示,分为客户端和服务端. 客户端即为虚拟机端,服务端即为可以访问到物理 GPU 设备的特权域或物理机. 实现方案的框架分为横向和纵向,横向分为应用层和系统层,纵向分为物理机和虚拟机. 客户端位于虚拟机应用层,服务端位于物理机应用层,系统层中存在有若干物理 GPU.

在客户端编写 CUDA 程序,采用动态库拦截的方式转移到服务端,服务端调用物理 GPU 进行运

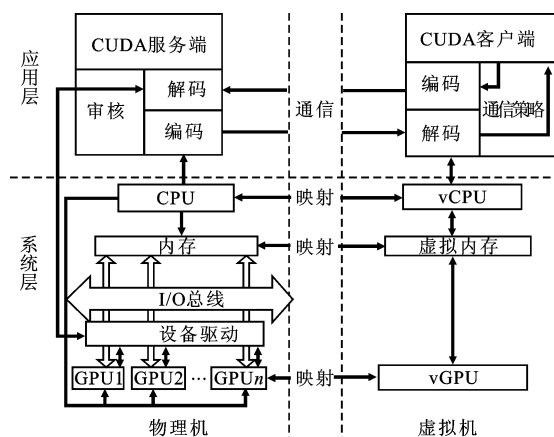


图1 虚拟化环境下多GPU系统计算实现方案

算,最后将结果返回给虚拟机。依据现有的多GPU并行计算的方法,将其引入到虚拟化环境中,使得CUDA程序在虚拟化环境下仍然可以调用多块GPU设备进行运算。如图1所示,在虚拟机中配置了vCPU(虚拟CPU),虚拟内存和vGPU(虚拟GPU),vCPU和虚拟内存都与物理CPU和内存相对应,通过物理CPU和内存映射,开辟虚拟的CPU和内存,而vGPU只是在虚拟机中配置所需真实GPU的逻辑映像,在虚拟机中并不能直接使用。当一个CUDA程序需要多块GPU设备运行时,可以使用单线程控制多个GPU设备,也可使用多线程控制多个GPU设备。其中,使用多线程控制多GPU的方案需在虚拟机端开设多个线程分别映射到物理CPU,使用物理CPU线程控制GPU,一个线程控制一块GPU设备;当使用单个线程时,通过轮转控制GPU设备。在虚拟化环境下进行多GPU并行运算时,涉及到数据分解、数据计算和数据合并这三个步骤。

数据分解是将程序分解为多个小规模的程序,根据每块GPU设备的计算能力,对任务进行合理划分,交由多块GPU设备并行执行。对于计算能力相同的设备,对数据平均分配,交由各块GPU设备是最有效的方式。使用一个或多个CPU线程控制多块GPU设备,将划分好的数据传给多GPU。假定GPU数目为 N ,数据规模为 M ,则每块GPU设备上分到 M/N 。在传输时,通过设置主机端指针和设备端指针,主机端指针根据数据划分规模指向不同的起始位置,并通过`cudaMemcpy()`函数从主机端线程的位置拷贝 M/N 的数据到每块目的GPU设备上。数据计算是在多块GPU设备上同时运行的过程,在运行过程中各块GPU设备可能是互不相干的,这种称为松耦合模式,各自独立计算,最后将结果回传给主

机端。当然也有可能运行过程中需要别的GPU设备上的数据,这种称为紧耦合模式,在计算过程中需要通信,计算完成后对数据进行合并。数据合并是指所有运算在各自独立的GPU设备上运行完之后,结果回传给主机端并将所有数据合并,得到最终的结果。

4.2 基于OpenMP和Pthread的多GPU并行计算

在CUDA 4.0以前,多个GPU设备需要多个主机端线程来进行管理。在每个CPU线程中,必须在执行任何CUDA运行时API函数之前调用一次`cudaSetDevice()`函数,以此与一个GPU设备关联,并且以后也不能与其他GPU设备关联。CPU线程的数量可以比设备数量多,但一个时刻一块GPU设备上只有一个CPU线程的上下文。一般认为,为达到较好的性能,最好使CPU线程数量等于设备数量,以使每个线程与设备一一对应。

将通常用于多核CPU编程的OpenMP^[11](Open Multi-Processing)和Pthread^[12](POSIX threads)等多线程技术引入到GPU编程中,实现多GPU的并行计算。OpenMP是基于共享内存编程模型的应用程序接口,采用fork/join并行模式,在源代码中加入指导性语句(Compile Directive)pragma来指明程序的并发性。当主线程执行到并行区时,会派生出其他线程,这些线程与主线程共同完成并行区的任务,必要时可加入通信和同步函数等。当忽略这些指导性语句时,程序自动转成串行程序。OpenMP移植性好、可扩展性高,已成为共享存储系统中的并行编程标准,适合控制多GPU进行运算。

Pthread是POSIX线程的简称,它与OpenMP类似,但更加注重底层实现。Pthread是在Linux下开发的,定义了各种平台下均可使用的C语言函数、类型和常量,因而易于移植。Pthread根据系统库创建活动线程,并发执行各自线程内的任务。对每个创建的线程适当控制,如通过加锁操作,可以保证各个线程之间不会交叉,每个线程只运行自己的那一部分。根据Pthread的特性,其同样适合控制多GPU进行运算。

4.3 基于流的多GPU并行计算

基于流的方式也可实现多GPU并行计算。流(Stream)是对CUDA并发进行管理的一种方式^[13],默认情况下,CUDA在一个GPU内部创建一个执行流,若不指定编号,默认为0。所有的数据传输与Kernel调用都在这个流中排队等候,并按队列顺序依次执行。通过显式地创建与使用多个执行流,能够在单位时间内进行更多的工作,使应用程序执行得

更快。

(1) 隐式同步使用多 GPU

在单个 CUDA 程序中使用多个 GPU 最简单的方式是为每个设备的每个上下文隐式地创建一个流,通过 `cudaSetDevice()` 更改当前设备来使用多个 GPU。CUDA 在创建上下文时通过 `cudaMalloc()` 引起上下文状态的改变,从而引入上下文环境的创建。GPU 设备驱动通过在设备驱动程序中为应用程序提供多个上下文环境,可以使单个 CUDA 应用程序使用多个设备。在每个设备上对任务进行排队,进而通过增加系统中 GPU 数量来提升应用程序性能。

(2) 显式同步使用多 GPU

在 CUDA 4.0 及以后版本中,可以使用单线程方案轮番作用于多个设备。通过显示同步使用流,使用单 CPU 线程就能进行控制。这种方式减少了 CPU 的开销,但在分配工作时需要额外处理不同 GPU 间的切换,这会消耗一定的时间,而且流之间相对无序,无法保证行为的正确性。通过在上下文环境间切换,设备驱动可以如同一个小的操作系统一样,使多个 GPU 计算程序多任务化。执行完所有 Kernel 函数后,使用 `cudaStreamSynchronize()` 函数在指定流上执行同步,等待指定流执行完成之后再继续执行后面的代码。

5 实验结果与性能分析

5.1 实验环境

实验所用服务器配备 Intel(R) Xeon(R) CPU E5410(2.33 GHz)处理器,32 GB 内存和 1TB 硬盘,并包含 4 块 Tesla C2070 GPU 设备。在虚拟机中使用 Ubuntu 12.04 的操作系统,物理机中使用 CentOS 6.5 操作系统,虚拟机平台为 VMware 10.0, CUDA 版本为 5.5。

5.2 数据传输优化

为了比较不同的数据传输模式,在本文的实验平台上,使用 SDK 中 `bandwidthTest` 测试不同数据大小的传输速度。平台中的主板都兼容 PCIe2.0 × 16 标准,理想带宽可以接近 8.0 GB/s。实验结果如图 2 所示。其中, H2D 表示从主机端到设备端; D2H 表示从设备端到主机端。可以看出,使用页锁定内存和可分页内存有显著区别,页锁定内存写入速度快 2 倍,读取速度快 2.48 倍。随着数据规模的增加,页锁定内存的传输带宽也在增长,最终达到 6.3 GB/s。页锁定内存中主机端到设备端稳定带宽达到 3.3 GB/s,反向的设备端到主机端稳定带宽仅为

2.7 GB/s。在主机端可以发现一个特殊的现象,PCI-E 提供两个方向(来自设备以及传输到设备的)速度相同的全双工通信,但是设备端到主机端的带宽比主机端到设备端的带宽高 10%。这种差异是依赖于硬件的。要想获得接近峰值的带宽,传输数据的大小需要是 2 MB 左右。事实上,只有传输数据大小为 16 MB 或以上时才能获得绝对峰值带宽。所以,在传输数据时,可以考虑将多个传输组合起来以增加总线的整体使用带宽。

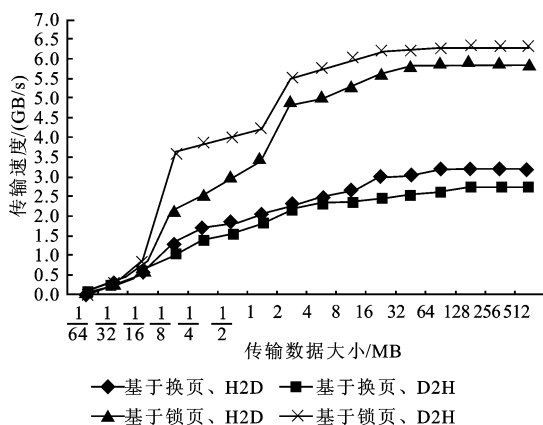


图2 CPU-GPU 传输带宽测试

5.3 蒙特卡罗算法的多 GPU 并行计算

蒙特卡罗^[14](MC)方法使用概率的方式求解近似解,目前被广泛应用于期权定价方案中。由于采样次数多而导致计算开销大,故而可以采用并行的方法来加速蒙特卡罗方法计算,提高采样次数来减少误差。通过随机采样次数(为 N)得到 N 个可能预期价格,再对这 N 个预期价格的平均值作为某个时刻期权价格的预估值,得出最终结果。可以看出,整个过程中每次采样以及处理过程都是相互独立的,适合大规模的并行化处理,故而可使用 GPU 高性能计算对其进行加速。同时,由于其独立性,可以使用多 GPU 并行计算的方法进一步提升性能,本小节主要验证在虚拟化环境下多 GPU 并行计算的加速效果。

将期权数量分割到多 GPU 上,每个 GPU 设备上随机数生成,采样计算以及采样结果处理这三个步骤。随机数的生成采用 CUDA 中的 CURAND 库,生成伪随机序列。在多 GPU 并行计算过程中,每个 GPU 设备的处理过程相同,不同之处在于生成的随机数不同。将计算划分到 M 个 GPU 设备上后,需要保证划分后的随机数要与原来未被划分的 N 次采样有相同的随机特性,即随机数质量和随机数分布类型一致,而对内容和顺序没有要求。

下面验证蒙特卡罗方法在多 GPU 并行计算的

效率. 随机采样次数越多, 误差就越少. 在本实验中, 路径数量采用 524 288 个, 期权数目为 1 024 个. 分

别使用一块 GPU 设备和 4 块 GPU 设备进行运算, 实验结果分别如图 3 和图 4 所示.

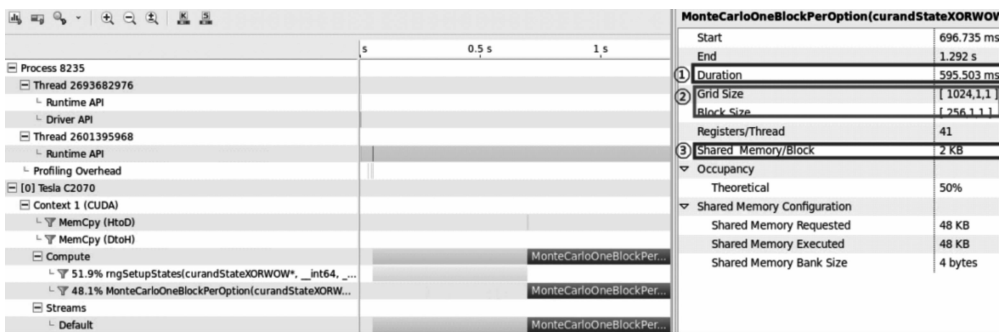


图3 使用单线程单 GPU 设备对蒙特卡罗函数的计算

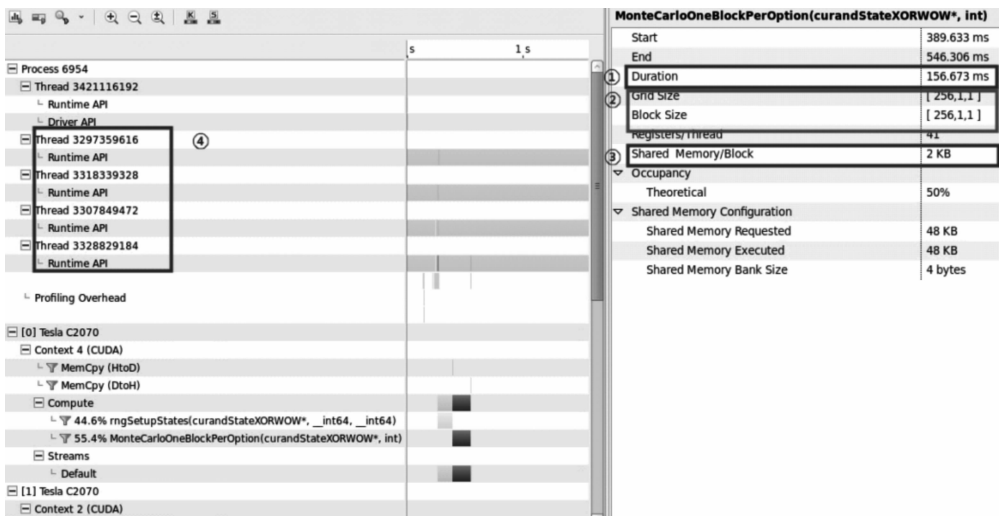


图4 使用多线程方式 4 块 GPU 设备对蒙特卡罗函数的计算

图 3 为使用单 GPU 进行运算得到的效果图, 不考虑 GPU 生成伪随机数的时间, 采用一个期权数量对应一个 block 的方法, 则共计使用 1 024 个 block. 为了使 GPU 资源利用率达到最大化, 通过使用 NVIDIA 提供的 CUDA_Occupancy_calculator.

xls 设置不同的线程数目对 GPU 资源利用率进行计算, 经过计算得到当 GPU 端的线程数目为 256 时 GPU 利用率最大(图 3 中②), 蒙特卡罗方法运算时间为 595.503 ms(图 4 中①). 当使用 Pthread 多线程的方式进行计算时, 即一个线程控制一块 GPU 设备, 在主机端开设 4 个线程, 如图 4 中④所示(由于受篇幅限制未能全部显示), 建立了 4 个 GPU 上下文, 每个上下文可以同时进行运算, 互不干扰. 采用与单 GPU 同样数目的期权和路径数目, 将 1 024 个期权数量平均分到 4 块 GPU 设备上, 每块 GPU 设备将有 256 个期权数目, 因而每块 GPU 设备的规模降低 1/

4. 0 号 GPU 设备的蒙特卡罗方法计算时间为 156.673 ms(图 4 中①), 其余几块设备的计算时间都与 0 号设备接近, 且各设备同时运行. 故加速比达到 3.801, 接近于 GPU 的数目. 最后将 GPU 设备同步, 并将计算结果传回给主机端. 由于需要计算每个期权的路径值最后的总和, 因此使用了共享存储器, 数组的大小与每个 block 中的线程数相同, 即 threadsPerBlock, 这样线程块中的每个线程都能将它计算的临时结果保存到数组中, 然后启动并行计算的最后规约, 如图 4 中③所示.

同样, 使用流的方式使得一个 CPU 线程控制多块 GPU 设备同时运算(图 6 中④), 使用一块 GPU 设备和 4 块 GPU 设备的运行结果分别如图 5 和图 6 所示. 当使用单块 GPU 设备时, 蒙特卡罗的计算使用了 595.474 ms(图 5 中①). 使用 4 块 GPU 设备时, 通过流的方式建立了 4 个 GPU 上下文, 并且在

每块 GPU 设备上分配一个乱序的流(图 6 中⑤),在 0 号设备上蒙特卡罗的计算函数使用了 156.953 ms (图 6 中①),其余设备的运行时间与 0 号设备接近.故加速比为 3.794,接近 GPU 数目.在使用流的方式时,最后需调用 `cudaStreamSynchronize()` 函数

执行同步,等待所有流执行完成之后再继续执行后面的代码.使用流的方式流之间相对无序.由于流之间复杂的并发性,使用流来开发算法可能会很难以调试,CUDA 提供了一些同步的函数来便于调试和计时.

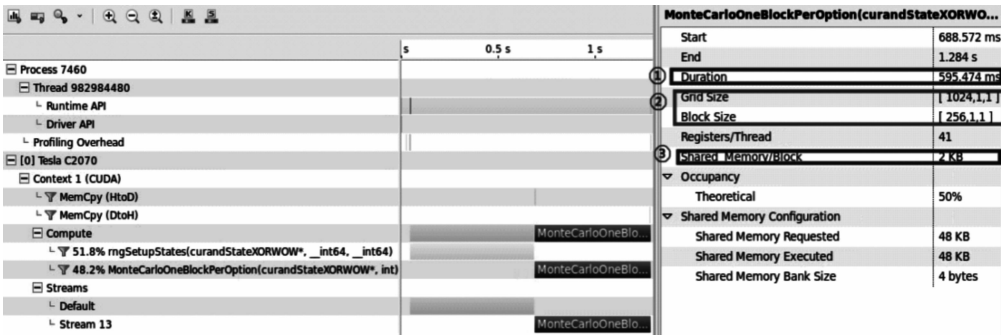


图 5 使用流的方式单 GPU 设备对蒙特卡罗函数的计算

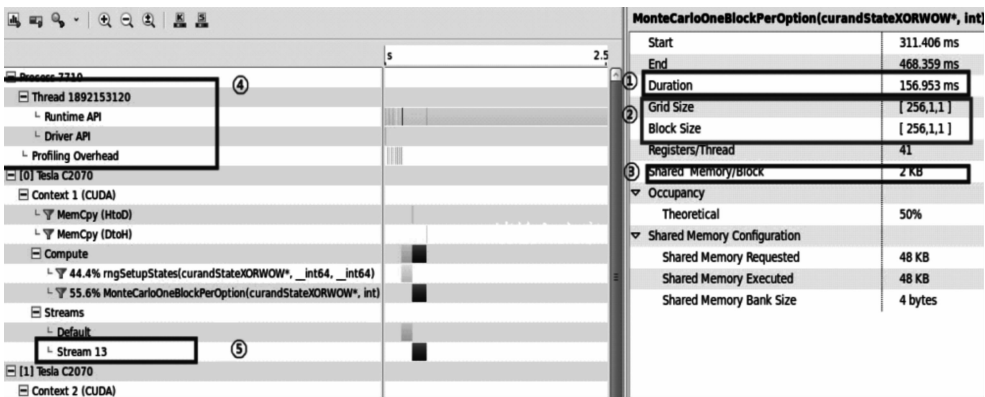


图 6 使用流的方式 4 块 GPU 设备对蒙特卡罗函数的计算

6 结束语

提出在虚拟化环境下使用多 GPU 并行计算加速大规模程序的方案,使用多线程的方式用一个 CPU 线程控制一块 GPU 设备,或者使用流的方式用单个 CPU 线程控制多块 GPU 设备.为提升程序的性能,探讨了 GPU 多层次存储技术与优化技术.给出虚拟化环境下多 GPU 并行计算的实现方案,并用实验验证了该实现方案的有效性和可行性.

参考文献:

[1] 崔泽永,赵会群. 基于 KVM 的虚拟化研究及应用[J]. 计算机技术与发展, 2011, 20(6): 108-111.

[2] Ravi V T, Becchi M, Agrawal G, et al. Supporting GPU sharing in cloud environments with a transparent runtime consolidation framework [C] // Proc of international symposium on High performance distributed computing. New York, 2011: 217-228.

[3] Chih-Yuan Y, Chung-Yao K, Wei-Shu H, et al. GPU virtualization support in cloud system [C] // Grid and Pervasive Computing. Korea, Seoul, 2013: 423-432.

[4] Shi L, Chen H, Sun J. vCUDA: GPU accelerated high performance computing in virtual machines [C] // Proc of International Parallel & Distributed Processing Symposium. Rome, 2009: 1-11.

[5] Gupta V, Gavrilovska A, Schwan K, et al. GVIM: GPU-accelerated virtual machines [C] // Proc of ACM Workshop on System-level Virtualization for High Performance Computing. New York, 2009: 17-24.

[6] Giunta G, Montella R, Agrillo G, et al. A GPGPU transparent virtualization component for high performance computing clouds [C] // Proc of EuroPar conference on Parallel Processing. Berlin, 2010: 379-391.

[7] Duato J, Pena A, Silla F, et al. rCUDA: reducing the number of GPU-based accelerators in high performance clusters [C] // Proc of International Conference on High Performance Computing and Simulation. Caen, 2010:

- 224-231.
- [8] 王海峰, 陈庆奎. 图形处理器通用计算关键技术研究综述[J]. 计算机学报, 2013, 36(4): 757-772.
- [9] 张舒, 褚艳利, 赵开勇, 等. GPU 高性能运算之 CUDA [M]. 北京: 中国水利水电出版社, 2010.
- [10] 肖灵芝. GPU 存储管理系统的设计与实现[D]. 西安: 西安邮电大学, 2013.
- [11] 汪超, 黄运保, 孙宇航. 激光间接驱动对称性分析的 GPU 并行求解[J], 计算机仿真, 2014, 30(8): 10-14.
- [12] 吴恩华, 柳有权. 基于图形处理器 (GPU) 的通用基数按[J]. 计算机辅助设计与图形学学报, 2004, 16(5): 601-602.
- [13] 董小社, 刘超, 王恩东, 等. 面向 GPU 异构并行系统

的多任务流编程模型[J]. 计算机学报, 2014, 37(7): 1638-1645.

- [14] 游晓鹏, 鲍金凤, 邱奉美, 等. RSSI 辅助的蒙特卡罗盒定位算法[J], 计算机技术与发展, 2013, 22(12): 11-15.

作者简介:

闵 芳 女, (1980-), 讲师. 研究方向为虚拟化.

E-mail: fangm@nuaa.edu.cn.

张志先 男, (1980-), 硕士. 研究方向为虚拟化.

张玉洁 女, (1991-), 硕士. 研究方向为虚拟化、高性能计算.

(上接第 68 页)

- [3] Patroklos Argyroudis G, Raja V, Hitesh T, et al. Performance analysis of cryptographic protocols on handheld devices [C] // Third IEEE International Symposium on Network Computing and Applications. Los Alamitos, 2004: 169-174.
- [4] Vipul G, Sumit G, Sheueling C, et al. Performance analysis of elliptic curve cryptography for SSL [M]. ACM Workshop on Wireless Security. USA, New York, 2002: 87-94.
- [5] 周永彬, 张振峰, 冯登国. 一种认证密钥协商协议的安全分析及改进[J]. 软件学报, 2006, 17(4): 868-875.
- [6] 谢志强, 郭军, 杨静. 新型 S/KEY 认证方案的分析与设计[J]. 计算机工程, 2009, 35(5): 175-176.
- [7] Kumar V K N. Mangipudi. New authentication and key agreement protocols for wireless applications [D]. USA,

State of North Dakota, North Dakota State University, 2005.

- [8] 徐泽水, 岳振军. 模糊综合评价的两种新算法[J]. 解放军理工大学学报: 自然科学版, 2001, 1(4): 5-8.
- [9] Yang M, Runtong Z, Qin W, New authentication scheme for M-commerce based on two dimension bar code [C] // IEEE International Conference on Service Operations and Logistics, and Informatics. China, Beijing, 2008: 1029-1034.

作者简介:

李丹丹 女, (1982-), 博士, 讲师. 研究方向为认知网络的智能 QoS 技术. E-mail: li_dandanhdj@163.com.

华 蕊 女, (1986-), 硕士. 研究方向为移动电子商务的身份认证.