文章编号:1002-1175(2011)06-0776-10

基于 GPGPU 和 CUDA 的高速 AES 算法的实现和优化*

顾 青[1],高 能[2†],包珍珍[3],向 继[2]

(1 国家 863 计划信息安全基础设施研究中心,上海 200336;
2 中国科学院研究生院信息安全国家重点实验室,北京 100049;3 中国科学技术大学,合肥 230026)

(2010 年 7 月 23 日收稿;2010 年 11 月 1 日收修改稿)

Gu Q, Gao N, Bao Z Z, et al. Implementation and optimization of high speed AES algorithm based on GPGPU and CUDA[J]. Journal of Graduate University of Chinese Academy of Sciences, 2011, 28(6):776-785.

摘 要 随着高性能计算需求的不断增长,人们开始将目光投向具有强大计算能力及高存储带宽的 GPU 设备. 与擅长处理复杂性逻辑事务的 CPU 相比,GPGPU(general purpose graphic processing unit,通用图形处理器)更适合于大规模数据并行处理. CUDA(compute unified device architecture,统一计算架构)的出现更加速了 GPGPU 应用面的扩张. 基于 GPGPU 和 CUDA 技术对 AES 算法的实现进行加速,得到整体吞吐量 6 ~ 7Gbit/s 的速度. 如果不考虑数据加载时间,对于 1MB 以上的输入规模,吞吐量可以达到 20Gbit/s.

关键词 通用图像处理器,统一计算架构,AES 算法,并行计算

中图分类号 TP393

信息成为能源,被人们广泛挖掘出来并传播之后,空间中以指数增长的已存信息数据量和人们以指数增长的信息需求量,使得人们必须加快对数据的处理速度. 单个处理器有时力不从心,它希望有种分身术,来应付堆放在存储器中大量的数据处理作业. 有的处理器克隆出多个自己,有的则增加几双助手,作它的协处理器;面对大量的作业,它们分工,同时进行处理,于是,“并行计算”成为“大量数据和加速处理”问题的一种解,在这个求解过程中,需要“数据并行计算”硬件和软件模型的研究和实现.

对于密码应用领域,加解密数据量随着信息空间的扩张而增加,信息文件大小的数据量级也在不断增长,从而增加了密码学应用的加速与可计算领域的关系. 作为有加速需求应用中的一员,这一“密码学应用的需求”可以尝试借助硬件的发展来满足. 于是,探试新的并行计算机和并行处理器也是一种可行的努力方向.

在充分挖掘计算机 CPU 的计算能力时,人们看到了其上具有强大计算能力及高存储带宽的另一个设备:图形处理器即 GPU,并开始了 CPU 与 GPU 的异构并行结构的实现与应用. 传统的 GPU 因其 2 种可编程着色器是分离的,使得应用程序很难同时发挥它们的性能;并且由于缺乏片内的存储器,计算单元之间不能进行通信,新出现的 GPGPU(general purpose graphic processing unit, 通用 GPU)采用严格的 SIMD(single instruction multiple data,单指令多数据)模式解决了这一问题.

传统 GPU 没有得到广泛应用,不仅在于这些硬件方面的限制,同时因其上的编程接口是基于图形学 API 编程,加重了开发人员的负担. 在人们的多年努力下,基于 GPU 的软件开发语言和开发环境也在

* 中国科学院知识创新工程(YYYJ-1013)和国家科技支撑课题(2008BAH32B04)资助

† 通讯联系人,E-mail:gaoneng@ lois. cn

逐渐进步, 引进 CUDA(compute unified device architecture,统一计算架构)的背景是硬件 GPU 采用了统一的处理构架,可以充分利用各种计算资源.另外引入片内共享存储器,支持了线程间的通信.最近,它可以支持整型运算和位运算,这进一步拓宽了 GPU 的应用面.于是,对于应用密码学领域,GPU 高计算能力的充分利用不再遥不可及.

为了使 GPU 在密码学领域充分发挥它强大的计算能力,本文对 GPU 的硬件结构和特点进行了分析,以 CUDA 编程技术和基于 CUDA 的 GPU 应用程序优化技术为基础,尝试基于新一代 GPU 实现 AES 算法;并对实现加以优化,通过性能测试和对测试结果的比较分析不断改善实现,以达到预期的加速效果.最后,实现了对于 8MB 规模的数据输入,达到 6 ~7Gbit/s 的加速效果(包括数据加载的时间).并希望今后可以进一步突破纪录,实现 AES 算法,以及其他密码学算法和 GPU 并行处理器的友好结合,为密码应用创造便捷.

本文第 1 节给出相关的研究成果;第 2 节介绍了 GPGPU 和 CUDA 的基本概念和编程模型;第 3 节介绍了 AES 算法和分析了用 GPGPU 来实现 AES 算法的可行性;第 4 节从并行模式、算法模式、数据分配和任务分配等角度介绍了基于 GPGPU 的 AES 算法实现;第 5 节给出实验结果,并对性能瓶颈进行分析;第 6 节从内核函数和数据加载 2 个方面对算法实现进行优化;第 7 节总结.

1　相关研究

最早想到使用 GPU 图形处理器解决密码学问题[1]的作者,使用的是一种称为 Pixel Flow[2]的图形引擎,这种引擎由大量 8bit 运行于 100MHz 上的处理器组成.文献[2]不仅成功地在 2 ~3 天破解 UNIX 系统口令/密钥,而且说明了本来是为解决图形处理问题而设计的硬件,同样适用于更通用的基于流的密码学问题.

最早尝试基于 GPU 的 AES 算法实现的是 D. Cook[3-5],这些研究工作是在 2003 年 GPU 引领浮点计算长跑的 2 年后开始的.由于当时硬件的限制和软件环境的不完善,只能通过图形 API:OpenGL 用标准的固定图形处理流水线与 AES 密码算法的执行路线进行映射.这些研究工作使用图形流水的图形子集来实现查找函数,图形子集是流水线中的固定功能部件,允许建立颜色的映射.文献[5]用颜色的映射来模拟 XOR 指令,然而系统吞吐量只达到 1.53 Mbit/s,而在 CPU 上实现需要 64 Mbit/s 之高.所以,当时的结论是传统的 GPU 并不适合密码学的应用.其实当时图形硬件的性能并不理想,很多功能部件虽有很高的计算能力,但缺少可编程性,必须依赖于性能低的子集加以实现.况且当时使用的最高级图形处理器是 Geforce3 Ti200,比现在使用的要落后 5 代.

2007 年,Svetlin A. Manavski[6],第 1 次证明 GPU 可以用作快速加密器.作者利用 NVIDIA GeForce 8800 GTX,基于 CUDA 编程平台,实现了使用 128 位密钥的 AES 算法,加密 8MB 数据.单纯的计算吞吐量达到 8.28Gbit/s(不考虑 CPU-GPU 数据传输的时间),同基于 Intel CPU 上的实现相比,总体加密速度达到 5.4 倍加速比(包括 CPU 和 GPU 之间数据传输的时间).

文献[6]总结了早期 GPU 不能很好支持加密算法的原因:1)密码算法一般都是整数运算,而传统的 GPU 恰恰缺少整数运算和位运算的表达方式;2)由于图形 API 接口的局限性,没有像 scatter 散列这样的基本操作;3)必须局限于图形 API 提供的存储器模型;4)要有管理固定的硬件流水线的开销.

但是,文献[6]作者没有尝试通过减小查找表的个数,以减少缓存占用率可能带来的性能提升,也没有实验用纹理缓存实现查找表的效果.同时,由于当时 CUDA 版本的限制,没有考虑主机端到设备端数据传输等更高层次的优化,以及寄存器管理指令调度等低层次的优化措施.对于线程任务分配量的分析和实验过程没有在该研究中提到.

在另一方向做出的努力是:使用 OpenGL 图形编程 API 的尝试. Harrison O 等[7]做了大量的工作,这些工作成为后期研究应用更强大的 GPU 设备实现 AES 算法的先驱.文献[7]使用 128 位密钥,采用 ECB 模式;基于 OpenGL 图形编程 API,在 Linux Fedora Core 4,2GHz AMD CPU 上,分别用 GeForce 6600GT AGP8x 和 GeForce 7900GT PCIe 图形卡,基于可编程的 Pixel pipeline,实现加密算法中的各轮操

作. 对异或操作 XOR 设计了 3 种实现方式,同时对纹理的使用方式也进行了测试. 实验结果是获得 0. 8708Gbit/s 吞吐量.

文献[7]作者最初选择 OpenGL 编程 API 的原因是:接近于硬件底层的语言可能更有利于对设备的直接管理,从而更有利于性能的优化. 总体上,这种实现并没有比文献[6]得到更高的性能,或许这是硬件和测试方法的不同造成的. 另外,使用 OpenGL 实现 AES 的还有文献[8-9]等,但是这些文献没有列出实验结果. Andrea[10] 提出基于 CUDA 的 AES_CTR 高效实现,比基于 OpenGL 的 CPU 实现[7] 快 14 倍. 文献[11]使用了更高配置的 GPU 设备进行研究,基于 GTX8800 和 GTX295 的实现分别达到 14. 6Gbit/s 和 59. 6Gbit/s 的吞吐量(不包括数据加载部分的时间). 文献[11]把查找表放在共享存储器中,证明这种方式很有效.

2 GPGPU 和 CUDA

2.1 GPGPU

并行处理器出现一支趋向于"蚁群"方式(众多功能较弱的处理器核组成一个并行处理器),而非"象阵"方式(数量有限的超级处理器核组成一个计算系统)的发展方向;或者称前者为"众核"方式,后者为"多核"方式. 一方面,数据并行计算模式下的应用需求所具有的特点是待处理数据具有规模性,以及数据元素具有同构性,这同时影响着并行处理器向"众核"的发展;另一方面,摩尔定律决定着高性能硬件发展趋势为"多核"集成. 例如传统 CPU 作为主流计算机处理能力的主要来源,比如 Intel core 2 Duo 处理器核,指令能耗为 10nJ;但是计算机上另一个具有计算能力的处理器 GPU 图形处理器,比如 8-core 32-thread Intel Graphics Media Accelerator X3000 指令能耗只有 0. 3nJ.

于是,在充分挖掘计算机的计算能力时,人们看到了其上具有强大计算能力及高存储带宽的另一个设备:图形处理器 GPU,并开始了 CPU 与 GPU 的异构并行结构的实现与应用. GPU 上的计算单元是分层次组织的,而不是数个强大内核的平行排列. 这种组织方式的层次性可以通过流处理器阵列即 SPA 上计算资源的层次性控制方式和存储资源的层次性和多样性体现出来. 整个 GPU 硬件资源是由分层组织的计算资源和多样的存储资源 2 部分组成.

流处理阵列(SPA)最高层次的计算资源管理部件是全局线程块调度器(global block scheduler),它控制着下级的多个流多处理器(SM)控制器,而每个 SM 控制器控制着少量(2 ~ 3 个)流多处理器(SM),SM 内部由 warp 指令调度单元控制着计算单元流处理器(SP)上线程组的执行情况. 所以,SPA 所呈现的组织形式是:最上层是由全局线程块调度器控制的数个(8 ~ 10 个)线程处理器群(TPC);中间层次是 TPC 内部由一个 SM 控制器负责管理的 2 或 3 个流多处理器(SM);最底层是每个 SM 拥有的 8 个流处理器(SP)和其他的一些计算部件. 全局线程块调度器顺序地将用户程序员组织的作业任务分派到可用 TPC 上. 随机分配有利于实现各 SM 的负载均衡. 这种计算资源的层次性组织方式,对应于 GPU 的设计目标——大规模数据并行处理模式的应用. GPU 硬件的设计针对于数据并行处理模式 SIMD.

存储资源的层次性可通过存储器的多样性体现出来:最高层存储资源由全局存储器构成,中间层次存储资源为支持 2 级缓存的纹理存储系统和常量存储器(常量存储器也是放在全局存储器上,只是支持缓存机制),次内层由片上的各种缓存和高效共享存储器组成,最内层是可以和算术逻辑单元(ALU)直接通信的寄存器.

计算单元的层次性和存储系统的层次性是相辅相成的. 比如:整个 SPA 共用一个全局存储器,即在 SPA 内的所有线程都可以访问这一层次的存储器. 每个 TPC 共用一块一级缓存和同一块共享存储器,但是这种共享存储器又是动态地分配给每个 SM 中的每个线程块的. 即共享存储器对应的是 SM 级的计算单元层次,每个 SM 中的寄存器文件是平均分配给其上的线程分别独自使用的,而线程的执行是在执行单元上的,所以寄存器这一层次的存储器间接地对应着计算单元层次中的单个执行单元. 计算单元的组织形式考虑到存储器的组织形式,而存储器的组织形式,又兼顾计算单元的结构设计. 各级存储器控制器以及各级计算单元控制器的设计相互影响,但是因为各自的特点和全局的考虑又不需要完全对应.

图 1 显示了 GT200 的一个 TPC 内部的计算资源和存储资源的配置情况.

2.2 CUDA

CUDA 是 NVIDIA(英伟达)公司针对其基于 GPU 的产品而引进的一个平台,拥有一套适用于 GPU 计算的开发环境;CUDA 技术,便于利用 GPU 设备进行通用并行计算,使 GPU 不仅限于图形处理,而且使程序员可以不需要通过繁琐的图形处理 API,推进 GPGPU 的开发.

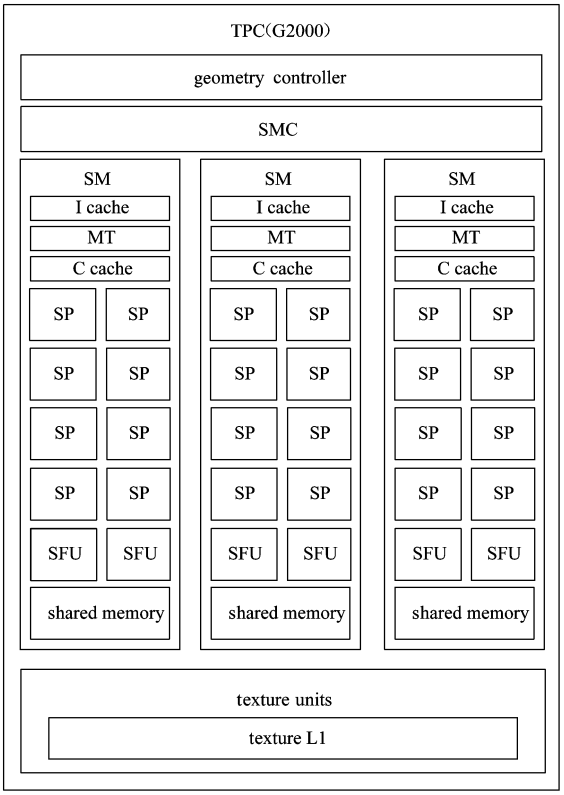

图 1　GT200 的一个 TPC 内部结构

CUDA 将 GPU 视为一个并行数据计算的设备,对所进行的计算进行分配和管理;CUDA 的 GPU 编程语言基于标准的 C 语言,扩展出一些关键字和组件;CUDA 执行模型是将 CPU 作为终端,而 GPU 作为服务器或协处理器,或者设备,从而让 GPU 来运行一些能够被高度线程化的程序. CUDA 的基本思想是尽量开发线程级并行,这些线程能够在硬件中被动态地调度和执行.

在 CUDA 编程中,程序员不能直接操纵处理器,所以不能指定哪些数据分配到哪个处理器,但是,他可以通过 CUDA 提供的线程模型,指定哪个线程处理哪些数据、哪些线程组成一个线程块,用户程序员处于编程模型之中,当编译器将程序编译完成后,便对应到并行线程执行模式中了. 之前的任务分派在执行模型中是哪些数据对应于哪些协作线程阵列(cooperative thread array, CTA),而线程阵列 CTA 又是通过计算分发单元动态地分发到处理器上的,于是程序员在驱动的辅助之下就实现了计算和数据的对准,完成任务分配问题.

3 AES 算法

AES,即 advanced encryption standard,高级加密算法. 它是美国新的数据加密标准,替代了早期的 DES 标准. 在 AES 的评估中,按重要性排序的候选算法特征准则中,除了安全性之外第 2 位就是成本:NIST 期望 AES 必须具有很高的计算效率,以便其能适于各种高速应用. 评选过程中 NIST 主要考虑算法软/硬件实现对存储空间的要求. 存储空间要求包括用硬件实现时所需的门数以及用软件实现时代码的长度和执行时所需 RAM 大小. 而位于第 3 位的是:算法的执行特征,要求算法具有很高的灵活性,适合于多种软、硬件实现方式,鼓励算法可以在固件上得以实现.

最终评选出来的候选算法 Rijndael 非常符合征集过程中所要求的这些特点. 这可以从各种方面体现出来:对于软件实现,Rijndael 非常利于在包含 8 位、32 位、64 位以及数字信号处理器(digital signal processing,DSP)在内的各种平台上执行加密和解密. 在密钥安装速度快的同时,其固有的分布执行机制能够充分有效地利用处理器资源,另外对 RAM 的要求也在合理的范围内. 同时它具有很好的密钥灵活性,支持加密中的快速子密钥计算,而且 Rijndael 对于单个分组加密有很好的并行执行能力.

所有这些特点也是本文采用它作为利用 GPU 加速分组密码算法的研究入点的原因之一. 由于 AES 对 RAM 的大小要求并不高,正如前面所认识的,GPU 把大量的晶体管用于计算单元,用小的缓存和控制器维护执行单元的高速运行,没有复杂分支预测、没有多个功能部件间的超级流水线和指令缓存,这些并不妨碍 AES 算法在 GPU 上的各线程间并行执行.

4 基于 GPGPU 和 CUDA 的 AES 实现

NVIDIA 的 GPU 设备经过 CUDA 封装,以一种支持各种字长的“众核标量处理器”模型呈现在程序员面前. 可以借鉴前期已有的基于 32 位平台的 AES 算法实现的相关研究,进行基于这种新设备的高速算法实现的研究.

由于 AES 算法密钥安装速度快,且没有理由在设备端进行冗余的扩展密钥的计算,所以可以由主机端预先计算子密钥,在启动内核函数之前与待处理数据一起加载于设备显存中.

4.1 并行模式选择

AES 算法将输入数据划分为结构相同的数据分组,对每个分组施行具有“数据依赖”关系的轮变换. 这一特点使得 AES 算法的 CUDA 并行化实现方法适合于按输入进行并行步的划分,而非按算法内部的轮变换(按功能)划分. 同时,基于 GPU 的应用程序采用 SPMD 编程模式,通常 FPGA 和 CPU 用来将性能发挥到极致的多功能部件的流水化和利用大的指令缓存等方法并不是 GPU 所擅长的. 结合 GPU 更适合于作为数据并行处理设备的特点,本文采用按输入划分的数据并行处理模型进行基于 GPU 的 AES 算法实现.

4.2 算法模式选择

出于安全性的考虑,人们在对长文件进行加解密时,要使用合适的链接模式,这样在增加全文混淆程度以确保安全性的同时,一些链接模式限制了 AES 算法并行计算的能力. 本文使用保留双向并行计算能力的链接模式 ECB(electronic code book,电子密码本)和 CTR(counter,计数器)模式. 其中 CTR 模式因为对每个数据分组使用的计数器不同而且可以同时计算,所以对于长文件的处理在保持着安全性的同时仍然适用于大规模数据并行计算,适合大范围推广.

4.3 数据分布和任务分配方案

一般数据并行处理模式下的程序设计首先需要考虑数据集合到处理器集合之间的映射. 实现这一映射所依据的原则是:提高程序执行的并行性(目的是增加执行单元的利用率)、促进数据相关的局部性(目的是减少通信开销),从而提高程序的执行效率. 对于基于 CUDA 的并行程序设计,程序员的工作是为线程分配数据. 在这一过程中,还要依据的另一个原则是:尽量实现最简的线程与数据对准方式.

综上分析,在基于 CUDA 的 AES 算法并行程序设计时需要依据的原则:1)提高算法的并行性,细化并行粒度,以增加执行单元的利用率;2)尽量减少线程间处理数据的相关性,最小化线程间通信开销;3)合理组织线程、合理组织数据存储结构,以便线程直接通过线程 ID 定位其所对应的数据.

针对这些原则,本文对 3 种实现方案(依据并行粒度划分)进行了分析(见图 2).

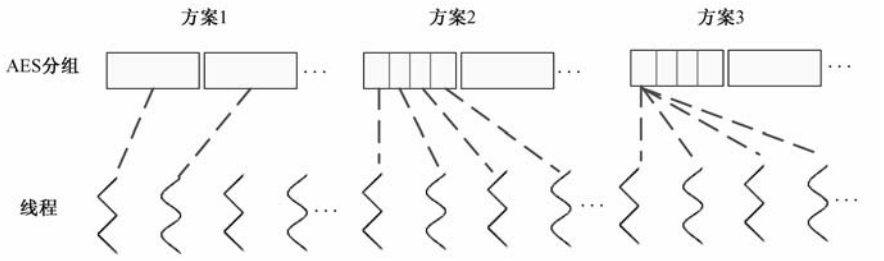

图 2 AES 的 3 种实现方案

方案 1:面向整块 AES 分组

第 1 种方案是让 GPU 上的线程模拟 CPU 上的 1 个线程的行为执行 AES 加密. 于是,首要任务是完成 1 个线程与整个 AES 数据分组间的映射. 把重点放在如何实现合并访问,以及避免共享存储器访问冲突等问题上.

此方案对第 2 个原则有很好的适应性:当用 1 个线程处理 1 个 AES 分组时,每个线程之间都可以是独立的,并不需要线程间的通信.但对于第 3 个原则:考虑到合并访问以最大化全局存储器的传输带宽的利用,需要 16 个属于同一个 half - warp(CUDA 中存储操作的基本单位)中的线程访问位于同一个 128Bytes 端中的连续数据.如果每个线程独自地下载自己负责的数据,就不能符合合并访问的要求.所以,为了更好地利用带宽,需要线程之间相互合作地将数据下载到共享存储器中,线程块同步之后各线程分别执行自己的计算任务,最后再次需要在线程块内进行同步,合作地将计算结果写回到全局存储器中.

对于第 1 个原则,这种数据分配方案,在每个线程开始计算之前都有 4 次访问全局存储器的操作,这里存在着连续的大段时间不能利用计算单元,这连续的 4 次访存操作至少带来 200 * 4 个时钟周期的延迟,需要大量活跃线程才能掩盖这些延迟.但是一个 SM 上最多可有 32 个活跃线程,所以很难掩盖如此长时间的延迟.这一方案的实验结果显示,它可以提供一定性能的提升,但是最终被后期的第 2 种方案取代.

方案 2:单个线程面向 32 位字

第 2 种方案由 4 个线程处理 1 个 AES 分组,每个线程负责 1 个字的状态转换.在增加并行性的同时,减少了第 1 种方案中的线程块内的同步开销.这是因为:虽然,每个线程负责 1 个 32 位字的下载,1 个 32 位字的状态更新需要其他 3 个字的上一轮状态,每个线程需要和相邻的其他 3 个线程通信和同步,因此可能带来通信要求.但是由于这种方案,可以利用 warp 内线程同步具有零开销的特点:合作计算一个 AES 分组的相邻 4 个线程因处于 1 个 half-warp 中,而具有的完全同步特征,每一轮的计算结束后,都同时将自己上一轮的计算结果存在共享存储器中,于是,每个字同时得到状态更新,然后同时被 4 个线程读取.这样,便在没有任何显式同步的情况下达到很好的线程间合作.同时,在这一方案中,由于 1 个线程在进入计算状态之前只需下载 1 个 32 位字,所以只有 1 次全局访问的读操作.

但是,需要注意的是,这里在线程组织层次上,增加了一层以 4 个线程为一组的小单元,对这一小单元的地址定位,以及每个线程在这一小单元中的位置计算是这种方案所带来的额外开销.这种开销实际是因为线程间的合作而带来的一种通信开销,因为,如果计算过程没有合作,如第 1 种方案,就并不需要线程记住自己在一种合作小组中的位置.而这种位置的要求是 AES 算法所必须的,这是由行移位和列混淆所带来的"组内所有字应当具有相关性"的要求决定的.虽然有这种开销,第 2 种方案却是实验结果中最好的一种,成为进一步优化的对象.

方案 3:线程小组面向 32 位字

第 3 种数据分布和任务分配方案同样存在着额外的通信开销,而且因为并行的粒度更细,通信开销相对更多.在第 3 种方案中,令 4 个线程负责 1 个 32 位字的状态转换.这时,每个线程在一轮中只需进行一次查找表,3 次 XOR(其中为了减少执行分支,有 2 个线程的计算是冗余的);这种方案实现了更细粒度的并行,但在增大并行性的同时,却增加了通信的开销,每一个线程的一次简单 XOR 操作时需要将自己的计算结果写回到共享存储器中,然后在共享存储器中读取所需数据.另外,如果一组线程在处理完 1 个 AES 分组后就退出,那么 1 个 warp 指令同样需要 2 次全局存储器读操作,但是却只需读 16 字节,即 1 次内存事务只读 16 字节,此时,带宽利用率很低.如果令一组线程在处理多个 AES 后才退出(比如 4 个或 8 个),那么可以在最初的数据下载时进行 64 字节或者 128 字节的合并下载.在执行过程中,合作完成 1 个 AES 分组后,继续进行另 1 个 AES 分组的计算.这种方案的实验测试结果不如第 2 种.但是仍需要进一步的研究.初步认为是因为通信开销的增多,减少了可以连续进行的计算;同时由于影响性能的瓶颈并不在于单个 AES 分组加密需要的指令数,减少由单个线程执行程序中一个 AES 分组加密的指令数,并不等于减少这个 AES 分组加密的总体时间,所以这也可能是对更细粒度的实现版本并不成功的一个解释.

经过上面 3 种方案的比较,本文最终选择第 2 种方案用于进一步优化,并进行性能测试.实验步骤是:首先确定线程组织方式,以最大化计算单元的利用率,确定对存储器的合理使用方案;随后针对影响

性能的各种瓶颈提出解决方案,并对各种方案进行了实验测试.我们没有立即排除那些对于一种瓶颈的解决没有效果的方案,因为考虑到基于 GPU 的 CUDA 程序的优化往往由多种因素共同影响,不同优化措施之间的不同结合可能产生各种差异很大的结果.在各种方案中找到最优的结合也是优化过程中花费精力而又重要的一部分.

5 实验和性能瓶颈分析

下面列出了实验所使用的软硬件环境.

• GPU 设备:采用 NVIDIA GTX285 设备,核心数量 30 个;内存接口宽:512bit,存储器容量 1024MB;存储器带宽:159GB/s;GPU 时钟频率:648MHz;存储器频率:1242MHz;总线接口:PCI-E x16@ x16.

• CPU:Intel Pentium 4 530;核心速度:2992.6MHz;一级数据:16KB 8 路组相联 64 字节管线大小;二级缓存:1024KB 8 路组相联 64 字节管线大小;核心数 1.

• 主机内存:DDR2,双通道数,1024MB;DRAM 频率:199.5MHz.

• 编程环境:使用 CUDA2.3 版本的编程环境,Windows XP,VS2005.

本文对整体性能做了测试,并与前人的工作[6]进行了比较,具体见表 1。

表 1 与前人工作进行比较

输入文件长度	NVIDIA G80/(Gbit/s)	NVIDIA GT200/(Gbit/s)
2KB	0.10	0.50
512KB	6.51	7.88
1MB	7.51	6.84
4MB	8.22	6.16
8MB	8.28	6.00

对照文献[6]的性能测试,做了相应的比较结果发现,虽然对于 2KB 的数据规模,本文的实现较其有了近 5 倍的性能提升;对于 512KB 也有相应的性能提升,但是,更大规模的数据输入时,无论对于加密内核吞吐量还是对于整体吞吐量都没有前人的工作效果好.

本文使用 NVIDIA 提供的软件 cudaprof 进行了更进一步的性能测试,并希望从中找到性能的瓶颈加以优化,使用 cudaprof 时,针对范围为 2KB ~8MB,长度倍增的输入数据规模进行测试.

由图 3 中可见,吞吐量与加密内核所占时间的关系非常密切,对于整体吞吐量,这种关系在数据量非常大时并不明显.可知,对于小规模数据,计算单元的利用率是提高它们性能的瓶颈:此时计算所占的 GPU 时间最长(通过 cudaprof 可以看到加密内核所占的 GPU 时间百分比).如果不能从内核函数本身加以优化,整体性能很难得到明显的提升.

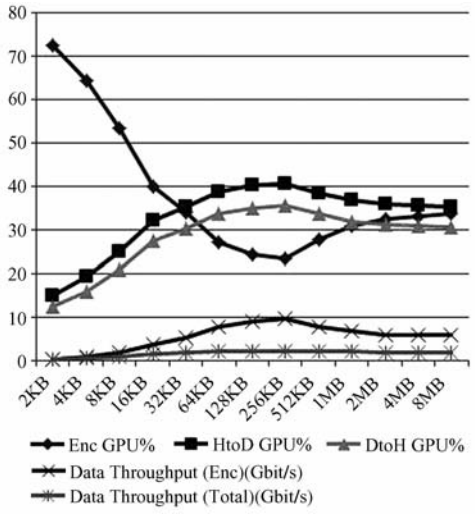

图 3 数据吞吐量同数据加载、数据计算、结果写回各阶段所占比例之间的关系

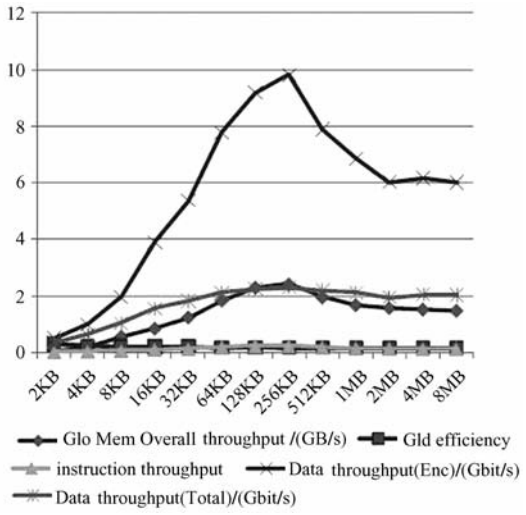

图 4 数据吞吐量与指令吞吐量、全局存储器总的吞吐量之间的关系

而对于大规模数据输入,数据加载和写回时间占用了总时间的大部分,从数据加载速度方面进行优化是提升大规模数据输入情形的关键. 通过图 4 数据吞吐量与全局存储器总的吞吐量之间的关系比较可见,当数据规模增大时,全局吞吐量趋于饱和,此时,性能的提升变化很少. 由于数据规模很大时,加密时间占用的时间只有 25% ~30% ,如果能够提高全局存储器总的吞吐量,才可以得到期望中的更高吞吐量.

针对不同数据规模,实现进一步加速需要在 2 个方面都能突破瓶颈:加密内核的执行效率和数据加载的速度. 针对于 2K ~32K 数据量,因为计算资源利用率为主要矛盾,针对计算资源的优化会带来相应的性能提高;而针对于 32K ~128K 数据量,虽然数据传输开始成为整体中的主要部分,但计算资源的优化仍然会起到一定的作用. 而针对于 32K ~128K 以及 128K 之后的所有数据量,更重要的是数据加载方面所需做出的优化.

6　性能优化

6.1　内核函数内部优化

对于加密内核的执行效率,本文提出的措施是:针对计算中占主要部分的查找表部分做进一步优化.

针对查找表的存放位置,我们之前考虑把查找表放在 constant(常量存储器)中,同时扩展密钥也放在 constant 中;但是,纹理存储器功能部件一直没有被充分利用,它同样拥有 2 层缓存,适合随机读取. 虽然它与数据加载流水线共用了一些片上资源,但同时有一部分独立部件,有助于避开同时使用同一个功能部件(在这里是数据加载流水线部分)时的竞争. 于是,本文尝试着把查找表放在纹理存储器中,并进行了性能的比较. 实验结果表明,较常量存储器的使用,能够带来一定的性能提升(见表 2):

表 2　使用纹理与使用常量加密 kernel 的数据吞吐量比较
(不包括数据加载和结果交付)

输入大小	优化前吞吐率/(Gbit/s)	优化后吞吐率/(Gbit/s)	加速比
2kB	0.502409	0.865683	1.72
4kB	0.99813	1.730662	1.73
8kB	1.962591	3.470947	1.77
16kB	3.89305	4.700313	1.21
32kB	5.363889	5.676988	1.06
64kB	7.772146	7.028225	0.90
128kB	9.174054	7.865578	0.86
256kB	9.800646	8.223545	0.84
512kB	7.88483	8.527729	1.08
1MB	6.847843	8.835499	1.29
2MB	6.019034	8.448253	1.40
4MB	6.163743	9.014605	1.46
8MB	6.000971	9.036186	1.51

使用常量缓存,在 256kB 数据的规模下可以获得最好的性能,当数据规模增大时,使用常量存储器的性能开始下降. 使用纹理存储器并没有这种性能上的峰值,随着输入数据规模的增加,吞吐量变化比较均匀. 进一步分析使用常量存储器时出现这种峰值以及后续吞吐量并不理想的原因是因为常量缓存比纹理缓存相对要少,如果需要处理大量数据,则需要大量线程,线程的指令此时会占据大量的常量一级缓存,从而导致缓存的不足.

虽然使用纹理比使用常量存储器有了一定的性能提升,但是这种提高的幅度是有限的. 纹理缓存需要纹理流水线上 Texture AGU 等额外的透明操作. 另外,纹理流水线和加载流水线共享了一部分硬件,所以不能同时使用. 之前所设想的“功能部件负载均衡”的理想情况,更合理地说是“功能部件压力的部分缓解”.

6.2　数据加载性能优化

前期注意的是设备端内核函数内部的优化,包括线程的组织和查找表的设计,内核函数内部注意的是“量子”级的操作. 但是,如果想要实现更高的性能,必须同时在另一瓶颈——数据加载效率上施以合理措施.

无论前期使用 cudaprof 时所分析到的,还是通过同前期人们的工作比较中所发现的,在数据加载这一方面的性能并不理想. 基于数据加载过程的认识,以及在这一阶段可以使用的优化措施,我们在数据加载方面使用特殊主机端内存加以改进.

采用特殊的主机端内存,指定开辟的主机端内存为 Mapped | Write Combinded 类型,以使得设备端的线程可以直接访问主机端内存,减少数据加载和结果写回时所用的时间,实现上载和下载的全双工. 实际上,直接使得主机端内存,原来的主机端与设备端的异步并行细化为线程块中的 warp 级线程组的异步并行. 此时,"数据加载"和"利用执行单元计算"的并行对象细化为各个 warp. 这非常类似于计算机网络中 IP 包的传输方式中分组传输比报文传输要高效很多的原理.

表 3 和图 5 分别是使用 Mapped_WC 与使用普通页锁定内存(不使用流机制)比较的情况:

表 3 使用 Mapped 和 Write Combined 与一般页锁定内存(不使用流机制)性能比较

输入大小	优化前吞吐率 /(Gbit/s)	优化后吞吐率 /(Gbit/s)	加速比
2KB	0.52	0.82	1.58
4KB	0.89	1.48	1.66
8KB	1.37	2.49	1.82
16KB	1.66	3.51	2.11
32KB	1.88	4.44	2.36
64KB	2.05	5	2.44
128KB	2.16	5.29	2.45
256KB	2.2	5.56	2.53
512KB	2.2	5.96	2.71
1MB	2.28	6.35	2.79
2MB	2.15	6.21	2.89
4MB	2.29	6.55	2.86
8MB	2.29	6.64	2.90

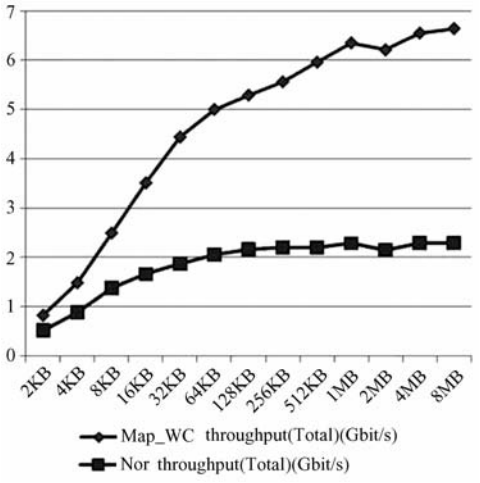

图 5 使用 Mapped 和 Write Combined 与一般页锁定内存性能比较

此时,性能提升近 3 倍,与预期相同. 可见,硬件的特殊支持,使得用 20% 的努力可以获得 80% 的性能提升. 这是硬件上的优化做出的贡献.

7 总结

基于 NVIDIA 的 GT200 硬件平台和 CUDA 软件编程环境,本文对 AES 算法加以实现并进行优化. 通过不断的实验和比较,最后得到了近 7G 的数据吞吐量. 实验证明,AES 算法可以很好地适应新一代的并行计算设备 GPU,同时,新一代的面向数据并行处理的 GPU 可以很好地帮助加速 AES 算法的执行.

参考文献

[1] Kedem G, Ishihara Y. Brute force attack on UNIX passwords with SIMD computer[C]// Proceedings of the 8th USENIX Security Symposium. 1999.

[2] Olano M, Lastra A. A shading language on graphics hardware: the PixelFlow shading system[J]. Journal of Computer Graphics, 1998: 159-168.

[3] Cook D L, Keroymytis A D. Cryptographics: exploiting graphics cards for security[C]// Advancements in Information Security Series. Springer, 2006.

[4] Cook D L, Ioannidis J, Keromytis A D, et al. CryptoGraphics: secret key cryptography using graphics cards[C]// RSA Conference, Cryptographer's Track (CT-RSA). 2005.

[5] Cook D L, Baratto R, Keromytis A. Remotely keyed cryptographics secure remote display access using (mostly) untrusted hardware[C]// ICICS05 Conference Proceedings. December 2005.

[6] Manavski S A. CUDA compatible GPU as an efficient hardware accelerator for AES cryptography[C]//2007 IEEE International Conference on Signal Processing and Communications (ICSPC 2007). Dubai, United Arab Emirates, November 2007.

[7] Harrison O, Waldron J. AES encryption implementation and analysis on commodity graphics processing units[C]// Ches 2007. LNCS, 2007,4727:209-226.

[8] Fiorese C, Budak C. AES on GPU: a CUDA implementation[C]// Proc of CHES. 2007.

[9] Yamanouchi T. GPU Gems 3[M/OL]. chapter 36: AES encryption and decryption on the GPU[2010-07-01]. SEGA Corporation. http://http. developer. nvidia. com/GPUGems3/gpugems3_ch36. html.

[10] Biagio A D, Barenghi A, Agosta G, et al. Design of a parallel aes for graphics hardware using the cuda framework[C]// IEEE International Symposium on Parallel& Distributed Processing. May 2009.

[11] Joppe W Bos1, Dag Arne Osvik, Deian Stefan. Fast implementations of AES on various platforms[C]// EPFL IC IIF LACAL, Station 14, CH-1015 Lausanne, Switzerland Fjoppe Bos, Dept of Electrical Engineering. The Cooper Union, NY 10003, New York, USA,2009.

Implementation and optimization of high speed AES algorithm based on GPGPU and CUDA

GU Qing[1], GAO Neng[2], BAO Zhen-Zhen[3], XIANG Ji[2]

(1 *National* 863 *Program Research Center for Information Security Infrastructure*, *Shanghai* 200336, *China*;
2 *State Key Laboratory of Information Security*, *Graduate University*, *Chinese Academy of Sciences*, *Beijing* 100049, *China*;
3 *University of Science and Technology of China*, *Hefei* 230026, *China*)

Abstract Compared with the CPU which is good at handling logic complexity service, GPGPU (general purpose graphic processing unit) is suitable for large-scale parallel processing computing. The emergence of CUDA (compute unified device architecture) accelerates the expansion of application of GPGPU. We accelerate the implementation of AES algorithm based on GPGPU and CUDA and achieve a total throughput of 6 ~ 7Gbit/s. Regardless of the time of data loading and storing, a throughput of 20Gbit/s towards an input size over 1MB can be achieved.

Key words GPGPU (general purpose graphic processing unit), CUDA (compute unified device architecture), AES algorithm, parallel computing