基于 GPU 实现的 AES 加密

彭海洋[1],杨红雨[1],杨 光[2]
(1. 四川大学 计算机学院 视觉合成图形图像技术重点学科实验室,四川 成都 610064;
2. 西北空管局,陕西 西安 710082)

摘 要:高级加密标准(Advanced Encryption Standard,AES),在密码学中又称 Rijndael 加密法,是美国联邦政府采用的一种区块加密标准。该算法已经被多方分析论证并广为全世界所使用。传统的 AES 加密运算是在 CPU 上实现的,现在为了提高加密速度以处理大规模的加密运算,文中提出了一种在图像处理器(Graphics Processing Unit)上实现 AES 加密算法的方法。该方法的实现有两种,一种是基于传统 OpenGL 的 AES 实现,另一种是基于最新技术 CUDA 的 AES 实现。文中阐述的是前者。经过测试,该方法比传统 CPU 的实现提高了 15 到 40 倍左右的速度。

关键词:图像处理器;美国高级加密标准算法;并行计算;图形流水线;快速实现
中图分类号:TP31 **文献标识码**:A **文章编号**:1673-629X(2013)02-0241-04
doi:10.3969/j.issn.1673-629X.2013.02.062

Implementation of AES Encryption Based on GPU

PENG Hai-yang[1], YANG Hong-yu[1], YANG Guang[2]
(1. National Key Laboratory of Fundamental Science on Synthetic Vision, College of Computer, Sichuan University, Chengdu 610064, China;
2. Air Traffic Management Bureau of Northwestern Region of CAAC, Xi'an 710082, China)

Abstract: AES is advanced encryption standard of American, also called Rijndael encryption method, which will replace DES in the future, and will be applied in many fields. Traditional AES encryption algorithm is based on CPU. It proposes a method based on GPU (Graphics Processing Unit). There are two implementation of this method. One is realized by AES based on traditional OpenGL, another one is based on AES of the newest CUDA. It introduces the first one. According to experiment's result, this method runs up to 15 to 40 times faster than traditional style approaches based on CPU.

Key words: GPU; AES; parallel computing; graphic pipeline; fast implementation

0 引 言

图形处理器(Graphics Processing Unit,GPU),亦称图形处理单元。是一种专用图像渲染设备,分担了中央处理器(CPU)的二维或三维图形处理任务。2001 年到 2006 年,各种硬件加速的出现使显卡性能突飞猛进。正是这个时候,第一款可编程图形流水线(programmable graphics pipeline)的 GPU 产品诞生,即 GeForce3[1]。从此,可编程的着色功能加入了硬件。

GPU 拥有了更大的可扩展性和适应性。GPU 高度并行化的架构和可编程的着色器使人们渐渐开始用它计算通用任务,实现了对通用数据的处理。GPGPU 技术也是在这个时期开始发展起来的。GPGPU 代表 General Purpose Computing on Graphics Processing Unit,就是"图形处理器通用计算技术"。这种新兴的加速技术试图把个人计算机上的显卡当作 CPU 这样的通用处理器来用,使显卡的强劲动力不仅发挥在图形处理上。用着色语言实现的 GPGPU 技术是第一代的 GPGPU 技术,也称为经典 GPGPU、传统 GPGPU[2]。但是传统的 GPU 几乎专门用来进行浮点操作,因为整数操作只能使用浮点数尾数来完成,所以就不能完成那些要求按位逻辑操作的处理。然而随着 GeForce 8 系列 GPU 的出现,多种新的扩展和功能被引入。新的整数处理功能不仅包括算术操作,还包括按位的逻辑操作,以及移位操作。然后,数组变量代替纹理提供了一种灵活的方式存储常量表[3]。目前,如何方便、快速地实现加密算法是密码学领域遇到的一个新的挑战。而 GPU 其特有的并行计算特性如果应用到 AES 加密算法,将显著提高该算法的加密速度。文中正是基于

收稿日期:2012-05-04;**修回日期**:2012-08-10
基金项目:国家重点基础研究发展计划(2009CB320803)
作者简介:彭海洋(1988-),男,硕士,研究方向为计算机图形学与 GPU 通用计算;杨红雨,教授,博士生导师,研究方向为计算机图像处理和图形学。

上述原因,提出了一种基于传统 OpenGL 可编程模型的一种 AES 加密算法。

文中首先将介绍传统 GPU 编程模型及相关概念,然后概述 AES 加密算法,接着介绍 AES 加密算法在 GPU 上的实现,最后将与在传统 CPU 上实现的 AES 加密算法进行性能上的比较。

1 图形化处理单元

1.1 图形流水线

图形流水线是 GPU 工作的通用模型。它以某种形式表示的三维场景为输入,输出二维的光栅图像到显示器,也就是位图。GPU 中的图形流水线的总体框架如图 1 所示[4]。上半部分为 OpenGL 传统图形流水线,下半部分为 OpenGL 可编程图形流水线。

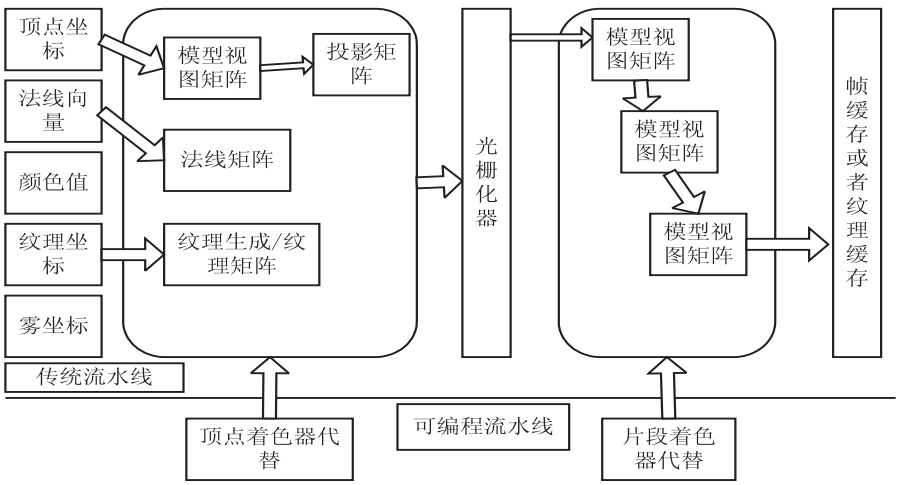

图 1 图形流水线

传统图形流水线,首先经过顶点级的光照计算和坐标变换求出每个顶点的光照颜色值,同时还将顶点坐标从物体坐标系转换到裁剪空间。然后,对每个三角形进行光栅化处理并对三角形顶点的颜色进行双线性插值得到了三角形中每一个像素的颜色值。接着进行纹理映射,即根据每一个像素的纹理坐值将纹理图颜色分配到每个像素上。最后进行颜色混合计算和雾化效果计算得到的结果将会放进帧缓存并显示到屏幕上[5]。

可编程着色管线意味着可以用小型程序控制顶点和片段的处理,即顶点着色器和片段着色器。这种小型程序是由 OpenGL 着色语言(GLSL)编写。着色器(shader),又叫着色单元,实际上就是 GPU 的处理器。一般情况下,一个 GPU 会有多个处理器(几百上千个),它们同时工作,体现了 GPU 大规模并行处理的能力。进行几何计算的处理器叫顶点着色器,它负责对顶点进行坐标变化、投影变换等;进行片段的颜色处理的叫做片段着色器。应用程序输入 GPU 的是三维的点云数据。从流水线输入端直到顶点着色器,流水线计算的对象都是三维几何模型;从光栅化开始,所有的操作都是针对二维的像素了。

1.2 经典 GPGPU 技术

在了解 GPGPU 技术之前,有必要了解一下 GPU 的并行计算模型 SIMD,与数据并行模型实际上是完全相同的。GPU 的并行概念实际就是数据并行(data parallelism)。它是指多个不同的数据同时被相同的指令、指令集或者算法处理。回顾图形流水线,每个顶点都经过了相同的流水线,实际上流水线只是个抽象的概念,它指的就是一段算法。因此,GPU 计算任务的本质与程序中的循环语句是相同的,但是需要串行处理的数据量减少到了 *N*parallel,如公式(1)所示。

$$N\text{parallel} = N\text{data} / N\text{threads} \quad (1)$$

式中,*N*data 是数据的总个数,*N*threads 是线程的个数。当每个线程处理一个数据所需的时间相等时,数据并行处理的速度是单线程循环语句的速度的 *N* threads 倍[6]。

经典 GPGPU 借助的是 GPU 图形流水线的大规模并行计算能力。成千上万的顶点都流经同样的流水线,成为屏幕上的像素。由于每个顶点和每个像素都经过了相同算法的处理,因此 SIMD 模型就是 GPU 计算的基本模型。

目前为止,还有一些很重要的问题亟待解决。比如,流水线的终点是帧缓存或者显示器,而科学计算一般需要写入存储器,这是如何做到? 图形流水线处理的是坐标信息和像素信息,如何才能处理通用数据? 在 OpenGL 提供有限数量的图形处理函数的情况下,如何制定科学计算所需要的算法? 对于第一个问题其实就是纹理缓存。第二个问题,其实用户只要选择合适的数据类型来声明数组,再将数组作为数据的容器,最后把这样的数组看作一幅纹理并传输至纹理缓存就可以啦。第三个问题其实就是通过着色语言和着色器来实现的[6]。

2 AES 算法概述

AES 加密算法是由 Daemen 和 Rijmen 早期所设计的 Square 改良而来。使用的是置换-组合架构,在软件及硬件上都能快速地加解密,相对来说较易于实际操作,且只需要很少的内存[7]。作为一个新的加密标

准,目前正被部署应用到更广大的范围。

该算法具有如下几个特点:

(1)对称密钥密码;

(2)块密码;

(3)支持 128 位块大小,当前该算法的联邦信息处理标准规范只支持固定大小的 128 位块;

(4)支持 128 位、192 位、256 位密钥长度[8]。不同的密钥长度加密的轮数不一样,文中使用 128 位的密钥长度。

AES 加密过程是在一个 4×4 的字节矩阵上运作,这个矩阵又称为"体(state)",其初值就是一个明文区块(矩阵中一个元素大小就是明文区块中的一个 Byte)[9]。

加密时,各轮 AES 加密循环(除最后一轮外)均包含如下 4 个步骤(加密过程中使用的密钥是由 Rijndael 密钥生成方案产生):

(1)AddRoundKey:矩阵中的每个字节都与该次回合金钥做 XOR(异或)运算。

(2)SubBytes:矩阵中的各字节透过一个 8 位元的 S-box 进行转换。这个步骤提供了加密法非线性的变换能力。S-box 是一个具有 256 个元素的已知数组。

(3)ShiftRows:针对矩阵的每一个横列操作的步骤。在此步骤中,每一行都向左循环位移某个偏移量。在 AES 中(区块大小 128 位元),第一行维持不变,第二行里的每个字节都向左循环移动一格。同理,第三行及第四行向左循环位移的偏移量就分别是 2 和 3。经过 ShiftRows 之后,矩阵中每一竖列,都是由输入矩阵中的每个不同列中的元素组成[10]。

(4)MixColumns:每一直行的四个字节透过线性变换互相结合。每一直行的四个元素分别当作 1,x^2,x^3,x^4 的系数,合并即为 GF(2^8)中的一个多项式,接着将此多项式和一个固定的多项式 $C(x) = 3x^3 + x^2 + x + 2$ 在 modulo x^4+1 下相乘。此步骤亦可视为 Rijndael 有限域之下的矩阵乘法。MixColumns 函数接受 4 个字节的输入,输出 4 个字节,每一个输入的字节都会对输出的 4 个字节造成影响。因此 ShiftRows ows 和 MixColumns 两步骤为这个密码系统提供了扩散性。其流程图如图 2 所示。

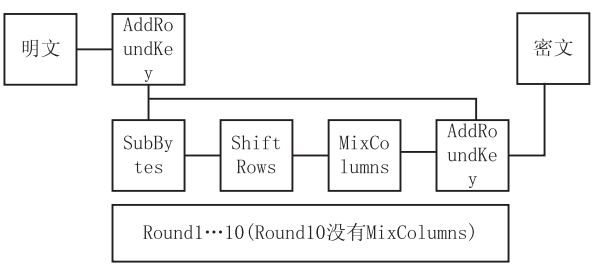

图 2　AES 加密流程

3　AES 在 GPU 上的实现

了解了可编程 GPU 流水线及 AES 算背景后,就可以开始着手设计该算法。如图 2,是 AES 加密算法的流程。图中大方框内的内容就是 GPU 代替 CPU 实现的步骤。首先考虑输入问题。明文是应用程序的输入,这可以通过二维纹理的形式存入 GPU 的纹理缓存,以供 GPU 使用。其实每一个 Round 就是一次渲染,每次渲染是由四个 Pass 组成,这四个 Pass 分别是 SubBytes, ShiftRows, MixColumns, AddRoundKey。优化后将 SubBytes 和 ShiftRows 合并为一个 Pass[3]。

3.1　初始化阶段

在初始化阶段,需要用 CPU 来扩展密钥,生成十组新的密钥,用于每一次 Round 中的 AddRoundKey。然后需要将输入数据(明文)InPutData[Width][Height][4],绑定到一张二维纹理。输入数据的大小是 Width * Height * 4 个字节。由于明文存储在二维纹理中,所以需要快速准确地用纹理坐标中找到对应的数据。通过如下的 Draw 函数能实现该功能,纹理坐标 = (顶点坐标+1.0)/2。由于需要将每个 Pass 的结果输出到纹理缓存,所以需要分别创建四张二维纹理及四个帧缓冲区,并一一绑定[11]。

```
void draw(int w, int h){
float w_bias=0.5/w;float h_bias=0.5/h;
float h_inverse=1.0/h;float w_inverse=1.0/w;
glBegin(GL_POINTS);
for(int i = 0;i < w;i++){
for(int j = 0;j < h;j++){
glVertex3f(i * 2 * w_inverse+2 * w_bias-1, j * 2 * h_inverse+
2 * h_bias-1, 0.0);}
}
glEnd();
}
```

3.2　SubBytes 操作

SubBytes 操作使用一个非线性的称为 S-box 的替换表,独立地代替字节,如图 3 所示。S-box 是提前已计算好的一个统一的表,所以可以直接预先存储在 GPU 内存中,在对应的 Shader(SubBytes)中声明。

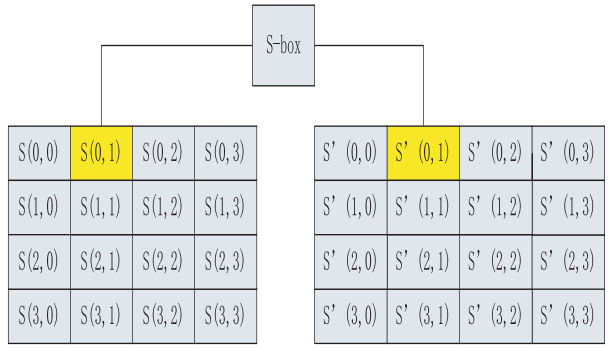

图 3　SubBytes 操作

3.3 ShiftRows 操作

ShiftRows 操作循环地移动体的最后三行，从而有效地让行数据变得不规则，如图 4 所示。其实此时可以做一个简单的优化，就是把 SubBytes 操作和 ShiftRows 操作融合到一个 Pass 里[12]。

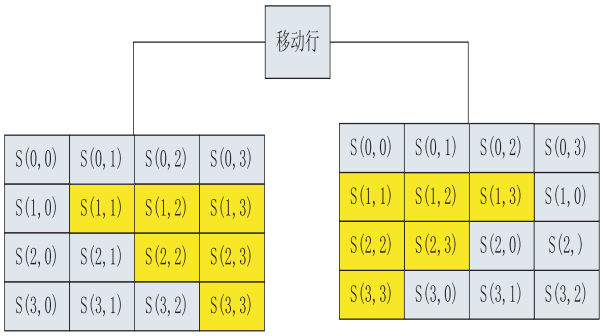

图 4 ShiftRows 操作

3.4 MixColumns 操作

MixColumns 操作的目的是使每一列的数据变得不规则，如图 5 所示。该操作通过对每一个列向量执行矩阵乘操作来完成，如公式(2)所示。因为乘法是在 AES 的一个有限域上操作，所以不能使用一般的乘操作。

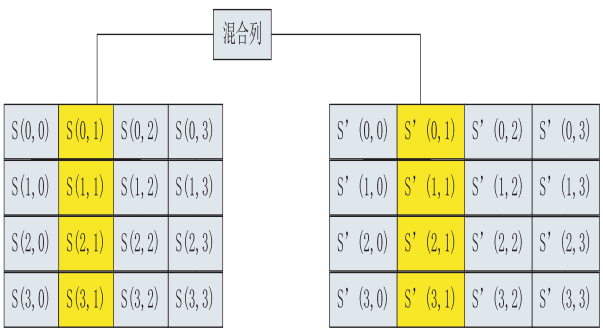

图 5 MixColumns

$$\begin{pmatrix} S'(0,1) \\ S'(1,1) \\ S'(2,1) \\ S'(3,1) \end{pmatrix} = \begin{pmatrix} 2\ 3\ 1\ 1 \\ 1\ 2\ 3\ 1 \\ 1\ 1\ 2\ 3 \\ 3\ 1\ 1\ 2 \end{pmatrix} \begin{pmatrix} S(0,1) \\ S(1,1) \\ S(2,1) \\ S(3,1) \end{pmatrix} \quad (2)$$

这个有限域的闭包在一个字节的范围内。其程序清单如下：

```
void MixColumns(vec2 pos)//pos 为纹理坐标
{// InputDataMix 为 sampler2D 的 uniform 变量
data=ivec4(texture2D(InputDataMix,pos));
int a[4];int b[4];int h;
for(int c = 0;c < 4;c++) {
a[c] = data[c];
if(data[c] > 127){h=0x000000FF;}
else{h=0x00000000;}
b[c]=(data[c]<<1)&0x000000FF;
b[c] ^= (0x0000001B& h);
}
data[0] = b[0] ^ a[3] ^ a[2] ^ b[1] ^ a[1];
/* 2 * a0 + a3 + a2 + 3 * a1 */
data[1] = b[1] ^ a[0] ^ a[3] ^ b[2] ^ a[2];
/* 2 * a1 + a0 + a3 + 3 * a2 */
data[2] = b[2] ^ a[1] ^ a[0] ^ b[3] ^ a[3];
/* 2 * a2 + a1 + a0 + 3 * a3 */
data[3] = b[3] ^ a[2] ^ a[1] ^ b[0] ^ a[0];
/* 2 * a3 + a2 + a1 + 3 * a0 */
OutputData = vec4(data);
//OutputData 是 varying 变量}
```

3.5 AddRoundKey 操作

这个操作是很简单的。就是用当前轮的密钥与体对应的每个字节进行异或操作。

4 性能对比

有了一个在基于 GPU 的 AES 实现，现在就可以测量一下该方法的性能。这里忽略了解密，因为在 AES 算法中，加密和解密的性能是相同的。性能对比如表 1 所示。批块大小是指，每次被送往 GPU 中的数据量。测试的机器配置如下：

(1) CPU：Intel(R) XeOn(R) W3520 2.67GHZ
(2) 内存：4GB
(3) 显卡：GTX 470 显存：1GB
(4) 系统：Windows7 x64 旗舰版

表 1 对比结果

批块大小	CPU 耗时(MS)	GPU 耗时(MS)
4KB	0.3	0.3
64KB	2.01	0.4
256KB	7.1	0.51
1MB	31.1	0.89
4MB	126.3	3.2
8MB	261.7	827
16MB	1062	1638

5 结束语

从表 1 可知，当批块小于 4MB 时，基于 GPU 的 AES 加密方法显著地提高了加密速度。当批块大小为 4MB 时，加密速度大约是基于 CPU 加密的 40 倍。当批块过大时，由于纹理大小的限制加密速度显著降低。此时可以把明文分割为 4MB 的批块，同时使用多张二维纹理进行加密。

参考文献：

[1] 徐少平，文 喜，肖 建，等. 一种基于 Cg 语言在图形处理器 GPU 上实现加密的方法[J]. 计算机应用与软件，2008，

(下转第 249 页)

个互联网制定出一套实用的副本管理机制是一项重要任务。其中,数据的副本数量、副本的分布、副本的定位和副本的数据一致性对网络的总体性能影响较大,因而是副本问题研究的重点。

参考文献:

[1] 葛建清. 异质结构化对等网络动态副本访问负载均衡策略研究[D]. 上海:华东师范大学,2010.

[2] 冯国富,张金城,陆桑璐,等. 无结构覆盖网络中面向搜索范围最小化的副本分布[J]. 计算机学报,2011,34(4):628-635.

[3] Lv Q, Cao P, Cohen E, et al. Search and Replication in Unstructured Peer-to-Peer Networks[C]//16th International Conference on Supercomputing. New York: ACM Press, 2002.

[4] Bassam A A, Chen W, Zhou B B, et al. Effects of Replica Placement Algorithms on Performance of Structured Overlay Networks[C]//Parallel and Distributed Processing Symposium. Nice: IPDPS, 2007.

[5] Gopalakrishnan V, Silaghi B, Bhattachar B, et al. Adaptive replication in peer-to-peer systems[C]//Proceedings of the 24th International Conference on Distributed Computing Systems. Japan: IEEE Press, 2004.

[6] 蒋砚军,马华东,张　海. 网格中热点服务的多副本部署策略[J]. 北京邮电大学学报(自然科学版),2007,30(2):89-92.

[7] 蒋砚军,马华东,张　海. 网格中多副本数据的后向预测调度算法[J]. 华中科技大学学报,2006,34(ZI):67-70.

[8] Rao A, Lakshminarayanan K, Surana S, et al. Load Balancing in Structured P2P Systems[C]//Proc of International Workshop on Peer-to-Peer Systems. Berlin: IPTPS, 2003.

[9] 陈建英,刘心松. 基于多维度 QoS 约束的大规模企业信息副本定位方法[J]. 计算机集成制造系统,2011,17(1):171-176.

[10] 李美安,刘心松,王　征. 多副本访问控制的分层结构分布式互斥算法[J]. 计算机工程,2006,32(9):112-114.

[11] Lamport L. Paxos made simple[J]. ACM SIGACT News, 2001,32(4):18-25.

[12] Ychellboy. 分布式一致性 Paxos 算法回顾[EB/OL]. 2010-04-05. http://kb.cnblogs.com/a/1704883/.

[13] 刘　鹏. 云计算[M]. 第 2 版. 北京:电子工业出版社,2011:25-51.

[14] 李振宇,谢高岗,李忠诚. PATCOM:基于分割树的无结构 P2P 系统一致性维护方法[J]. 计算机学报,2007,30(9):1500-1510.

[15] Ratnasamy S, Francis P, Handley M, et al. A salable content addressable network[C]//Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications. San Diego: SIGCOMM, 2001:161-172.

[16] Stoica I, Morris R, Karger D, et al. Chord: A scalable peer to peer lookup service for internet applications[C]//ACM SIGCOMM 2001. San Deigo: SIGCOMM, 2001:149-160.

[17] Dabek F, Kaashoek M F, Karger D, et al. Wide area cooperative storage with CFS[C]//Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP). Banff: SOSP, 2001:202-215.

[18] Datta A, Hauswirth M, Aberer K. Updates in highly unreliable, replicated peer to peer systems[C]//Proceedings of the 23rd International Conference on Distributed Computing Systems. Washington: ICDCS, 2003:76-85.

[19] 陈　赓,余宏亮,张　堃. 对等网络中基于位置信息和文件流行度的自适应复本管理机制算法[J]. 计算机学报,2009,32(10):1927-1937.

(上接第 244 页)

25(4):260-262.

[2] NVIDIA. NVIDIA CUDA Programming Guide ver. 2.1.2008[EB/OL]. 2008. http://developer.download.nvidia.com/compute/cuda.

[3] Nguyen H. GPU Gems 3[M]. 杨柏林,陈根浪,王　聪译. 北京:清华大学出版社,2010.

[4] Shreiner D. The Khronos OpenGL ARB Working Group. OpenGL Programming Guide[M]. 李　军,徐　波译. 7th ed. 北京:机械工业出版社,2010:424-427.

[5] Cook D L, Ioannidis J, Keromytis A D, et al. CryptoGraphics: Secret key cryptography using graphics cards[C]//LNCS 3376. [s.l.]:[s.n.],2005:334-350.

[6] 仇德元. GPGPU 编程技术[M]. 北京:机械工业出版社,2011:19-20.

[7] 维基百科. 高级加密标准[S/OL]. 2012-01-16[2012-04]. http://zh.wikipedia.org/wiki.

[8] NIST. Advanced Encryption Standard (AES)[S]. USA: FIPS, 2001.

[9] 吴亚联,段　斌. AES 密码计算构建的设计及应用[J]. 计算机工程,2005,31(21):181-186.

[10] 曹华平,罗守山,温巧燕,等. AES 算法轮密钥与种子密钥之间的关系研究[J]. 北京邮电大学学报,2002,25(4):47-50.

[11] Wang F, Qiu J, Yang J, et al. Hadoop high availability through metadata replication[C]//Proceeding of the First International Workshop on Cloud Data Management. Hong Kong, China: [s.n.],2009.

[12] Rost R J. OpengGL Shading Language[M]. 北京:人民邮电出版社,2006:43-62.