文章编号:1001 - 9081(2005)09 - 2192 - 04

图形硬件通用计算技术的应用研究

张 杨,诸昌钤,何太军
(西南交通大学 计算机与通信工程学院,四川 成都 610031)
(zhya2000@163.com)

摘 要:在通用计算的图形硬件加速研究中,综合了在 OPENGL 体系下的计算模型。通过实验,测试了该计算结构的性能并分析了提高计算性能的一些方法。在此基础上,介绍一种基于 GPU 的并行计算二维离散余弦变换方法。该方法可在 GPU 上通过一遍绘制,对一幅图像 1 至 4 个颜色通道,同时进行 8×8 大小像素块的离散余弦变换。实验表明在该实验硬件基础上,采用 GPU 加速的并行离散余弦变换,可比相同算法的 CPU 实现提高数百倍。

关键词:图形处理器(GPU);离散余弦变换(DCT);可编程图形管线;并行计算
中图分类号:TP317.4 **文献标识码**:A

Application research on general purpose computation on GPU

ZHANG Yang, ZHU Chang-qian, HE Tai-jun
(*School of Computer & Communication Engineering, Southwest Jiaotong University, Chengdu Sichuan* 610031, *China*)

Abstract: In the research on general purpose computation accelerated by graphics hardware, a computation model based on OPENGL was synthesized, the performance of the computation structure was tested and several ways to enhance computing performance were analyzed. Then a method of parallel 2-d DCT on GPU was presented. This method computes DCT on 8×8 pixel blocks by one pass rendering. Up to 4 color channels of a picture can be simultaneously performed during the computation. Experiment results indicate that performance of hardware accelerated DCT is hundreds of times faster than that of CPU implementation in our hardware conditions.

Key words: graphics processing unit (GPU); discrete cosine transform (DCT); programmable pipeline; parallel computation

0 引言

在最近的几年里,GPU 的可编程性能得到很大的提高,成为计算机中新的计算资源[1],用它对软件进行硬件加速。有人把该应用研究称作 GPGPU(General Purpose Computation on GPU),即,基于 GPU 的通用计算。

采用 GPU 做软件的加速,利用了 GPU 的某些特性。GPU 可以看作一个并行的数据流处理器[2]。GPU 是向量操作结构。另外,GPU 从显存中存取数据有较大的带宽。目前,大量 GPU 应用研究集中在数值计算和各种场合的模拟仿真计算中,文献[3]中有详细介绍。在信息领域,文献[4]介绍了图形处理器上实现小波变换。文献[5]介绍了在 GPU 上实现的 FFT 方法。文献[6]介绍了基于 GPU 的快速 level Set 图像分割。本文属于 GPU 在图形图像领域的应用。

1 基于 OPENGL 体系构建通用计算模型

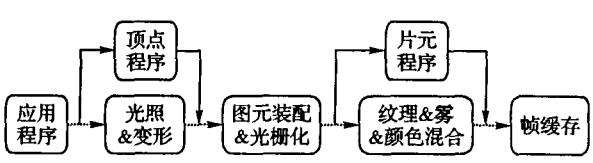


图 1 可编程图形管线

如图 1 所示,在 OPENGL 可编程图形管线中,顶点程序(ARB_vertex_program)取代了 OPENGL 固定的光照计算和几何变形计算;片元程序(ARB_fragment_program)取代了纹理应用、颜色混和以及雾化应用。GPU 中片元处理器相对于顶点处理器功能更强。因此选择片元处理器和片元程序作为通用计算应用的核心。

如图 2 所示的通用计算模型在 OpenGL 1.5 下实现,采用 32 位浮点精度。该模型可以在 nVIDIA NV30 及其以上的 GPU 上实现。

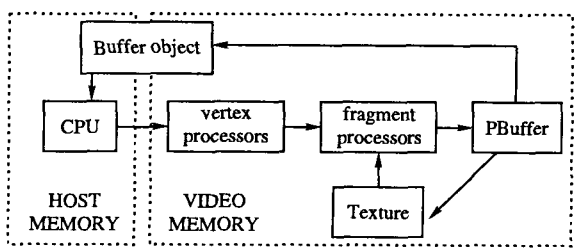


图 2 OpenGL 下的通用计算模型

在初始化数据阶段,OPENGL 应用程序将输入数据写入 2 维纹理。每个纹理有 4 个颜色通道(RGBA),那么将获得 n×m×4的三维数据空间。在并行计算阶段,实现一次并行计算的过程即是在 OPENGL 下绘制一个矩形块(quad)的过程。当片元程序有效时,片元处理器对每个片元执行片元程序。为了使纹理单元和片元一一对应,需要设置正投影以及和纹理大小相同的视区。使得每个片元(fragment)和每个纹理单元(texel)一一对应,如图 3 所示。经过计算后的数据,在

收稿日期:2005 - 03 - 29;**修订日期**:2005 - 05 - 25

作者简介:张杨(1980 -),男,主要研究方向:多媒体与虚拟现实、信息处理的硬件加速; 诸昌钤(1938 -),男,教授,博士生导师,主要研究方向:计算机在交通信息工程及控制中的应用; 何太军(1969 -),男,博士研究生,主要研究方向:图形图像、多媒体、虚拟现实.

图形应用中一般直接把它们输出到帧缓存里显示出来。

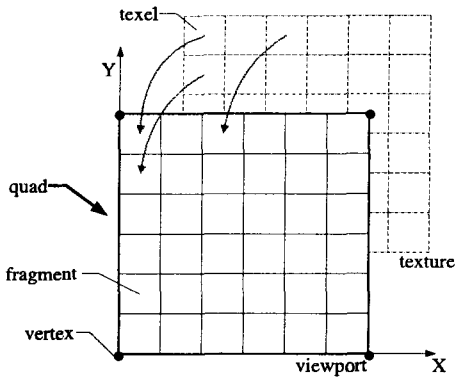


图3　片元纹理2维数据映射关系

帧缓存中的数据精度和当前显示模式有关,每个颜色通道最多8位。加上帧缓存的数据范围也被限定在0~1之间,所以不适宜做通用的数值计算。为了获得大范围、高精度的数据输出缓存,使用 WGL_ARB_pbuffer 扩展提供的 pbuffer[7] 作为输出的缓存。Pbuffer 在显存中开辟一个不可见的数据缓存区,当 pbuffer 有效时,它将代替帧缓存接收片元处理器的计算结果。如果这个结果只是中间数据,需要片元程序对其做进一步的处理,就需要把 pbuffer 中的数据传递到纹理中供下一遍绘制的片元程序取用。数据在此间传递速度越快,多遍绘制的性能就越高。最终计算的结果数据需要传回主内存和应用程序做交互,但是把 pbuffer 中传回主内存是个耗时的过程。采用 EXT_pixel_buffer_object 扩展配合 Buffer Object 在显存和主存间采用 DMA 传输,可以提高传输效率。

2　计算步骤及性能测试

我们假设一个多遍绘制的通用计算应用需要如下一些步骤:

1) 初始化数据:在主存中初始化数据空间,给3维数组赋初值。

2) 初始化第1遍绘制的纹理:把主存中数据传到显存中,用第1步初始化的数组给二维纹理赋初值。

3) 使 pbuffer 有效,绑定当前纹理。

4) 绑定片元程序。

5) 并行程序执行:绘制矩形,设纹理坐标,GPU 图形管线执行片断程序。

6) 获取中间数据:把 pbuffer 中的数据传递到纹理中,为下一遍绘制作准备。该步骤可选方法有:6.1)拷贝到纹理,选用 glcopyTexSubimage 函数;6.2)渲染到纹理,使用扩展 WGL_ARB_render_texture。

7) 重复5~7步,多遍计算。

8) 返回数据到主存:把存在显存 pbuffer 中的结果返回到主存中。该步骤可选方法有:8.1)选用 buffer object, 会启用 DMA 传输;8.2)直接使用 glReadPixels。

9) 数据后期的处理。

我们测试环境为 nVIDIA GeForce 5 700(GPU)128MB 显存 128bits 带宽,AMD 650MHz(CPU)192MB 内存,AGP 4X,驱动程序版本为66.72,在 Windows XP 操作系统下采用同一个简单的线性插值片元程序,分别对 128×128、256×256、512×512 这3组数据计算进行测试。得到的结果见表1。

虽然在 Windows XP 这种多任务操作系统下无法的测得函数的精确耗时,但是我们采用以下的方法保证这些数据在定性分析上述计算步骤的性能时是可靠的。我们使用高精度的定时函数对应用函数计时,分析筛选数值稳定性较好的数据组。筛选后的数据中仍会出现个别较大数字。在每组数据删除两个最大数后求平均值,得到上面的结果。从这些数据中可以看出,使用6.1和8.1将会有利于提高计算性能。需要说明的是,由于 DMA 传输在传输一旦建立后函数就立即返回,所以和要传输的数据量大小关系不大,加之我们测试不同大小的数据不是在同一个程序中完成,所以8.1行数据出现第2列比第3、4列的耗时大一点,是可解释的。

表1

步骤	128×128 耗时/ms	256×256 耗时/ms	512×512 耗时/ms
1	2.753	11.270	50.194
2	1.764	7.522	34.276
3	0.631	0.559	0.573
4	0.016	0.021	0.018
5	2.534	9.277	36.490
6.1	0.062	0.065	0.077
6.2	0.375	0.352	0.396
8.1	0.295	0.204	0.212
8.2	4.001	14.285	56.226
9	0.447	0.543	1.231

3　基于 GPU 的二维离散余弦变换

DCT(离散余弦变换),广泛用于音频、视频的压缩编码。目前二维 DCT 变换已经成为许多图像、视频标准的重要组成部分。本文介绍的 DCT 变换,是基于 GPU 硬件加速的二维 DCT 变换,在不对图像进行预先分割的情况下实现8×8大小的像素块的分块和并行计算。

3.1　二维 DCT 算法描述

对于二维平面上的数字信号,设二维平面上的离散点 $(m,n) \in S, S = \{(m,n) \mid m = 0,1,\cdots,M-1; n = 0,1,\cdots,N-1\}$,各点信号用 $x(m,n)$ 表示,信号序列 $\{x(m,n)\}$ 的二维 DCT 序列为 $\{Z(k,l)\}$,则有

$$Z(k,l) = \frac{2c(k)c(l)}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x(m,n) * \cos\frac{(2m+1)k\pi}{2M} \cos\frac{(2n+1)l\pi}{2N} \tag{1}$$

其中:$k = 0,1,\cdots,M-1$

$l = 0,1,\cdots,N-1$

$$c(k) = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0 \\ 1, & k = 1,2,\cdots,M-1 \end{cases}$$

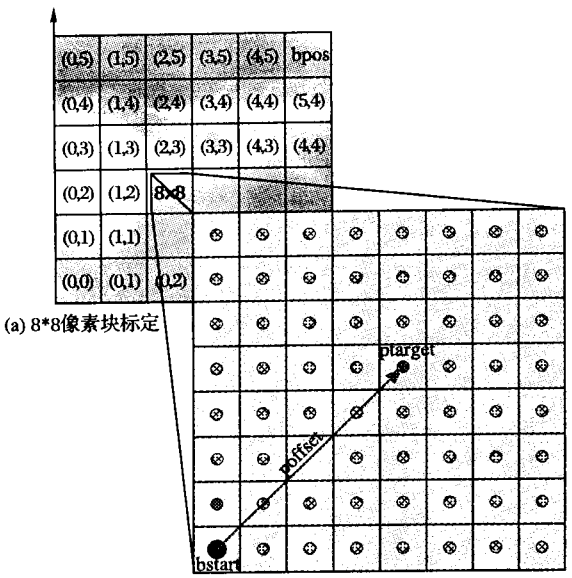
$$c(l) = \begin{cases} \frac{1}{\sqrt{2}}, & l = 0 \\ 1, & l = 1,2,\cdots,N-1 \end{cases}$$

从式(1)可知,对于 $\{Z(k,l)\}$ 中每一点的 DCT 计算需要遍历它所有的信号序列 $\{x(m,n)\}$,考虑到 GPU 的指令程序长度限制,我们实现的是8×8大小的 DCT 变换。在8×8的二维 DCT 变换中 $M = N = 8$。

3.2　像素标定和像素块划分

以一幅 RGB3 通道的二维数字图像为例,按图3所示方向对每个像素设定二维平面的整数坐标,用二维向量 pxy 表示该坐标,$pxy = (x,y)$。图4(a)所示如何在二维平面上划分像素块。其中每个网格代表一个8×8的像素块,网格上的二维向量标记该像素块在整个二维平面中唯一的坐标,用二维向

量 *bpos* 表示这个坐标。图 4(b) 所示为放大的图 4(a) 中的某个像素块,带有黑点的网格表示每个像素点。



(a) 8*8像素块标定

(b) 单个像素块坐标示意

图 4

对于二维平面内任意一个像素点,该像素点位置的 DCT 值只与它所属的那个 8 × 8 像素块内的像素的颜色值有关。而任意像素点的坐标 *pxy* 和该像素所属的像素块的位置 *bpos* 有如下关系:

$$bpos = floor(pxy/8) \tag{2}$$

其中 $floor(x)$ 函数为对二维向量 x 的每个分量取整数部分;

设每个像素块左下角起点位置的像素点坐标为 *bstart*,它为遍历该像素块起到标定的作用。那么有

$$bstart = bpos * 8 \tag{3}$$

其中 $bpos * 8$ 表示向量 *bpos* 的每个分量乘以 8;

设像素点与它所属的像素块的 *bstart* 位置的坐标的偏移量为 *poffset*,那么有如下关系式:

$$poffset = pxy \bmod 8 \tag{4}$$

$pxy \bmod 8$ 表示 *pxy* 的每个分量对 8 取余;

在任意个 8 × 8 像素块内,式(1) 中的 (k,l) 等于 *poffset*,式(1) 中的 $x(0,0)$ 值,为 *bstart* 位置对应的像素值。有如下关系:

$$x_{rgb}(m,n) = sample(bstart + (m,n)) \tag{5}$$

其中 $sample(x)$ 函数表示从二维向量 x 的位置采样图像,RGB 分量组成的向量,用 $x_{rgb}(m,n)$ 表示。

因此只要知道任意一个像素的坐标 *pxy*,就可以知道它所属的像素块,以及该像素块的起始坐标和相对该起始坐标的偏移。从像素块的起始坐标 *bstart* 开始,遍历块内 8 × 8 个像素,按照式(1)求出该点的 DCT。该算法在没有使用条件判断逻辑情况下对所有像素执行同一程序,易于在 GPU 实现。

3.3 算法实现及其性能优化讨论

我们用 cg 语言写描述上述算法,公式(2)(3)(4)(5)的表示方法都是类似 cg 语言的向量表示法。通过 NVIDIA 的 cg1.3 编译器编译成的 OPENGL 片元程序,绑定到 OPENGL 应用程序中实现该 DCT 计算,图 5 为一幅原图和一幅 8 × 8 DCT 计算结果图。在与第 2 部分相同测试环境下,测试 GPU 计算对 CPU 计算的加速效果,对于 128 × 128 大小的图像,同为单精度浮点数下,直接在电脑上编程计算平均需要 2058ms 的时间,而采用本算法在 GPU 硬件加速仅需 7ms。

片元程序的性能,和许多因素有关。片元程序使用的中间变量寄存器越少,性能越好,上述程序编译成片元程序需要用到 4 个中间变量寄存器。在使用 NVIDIA GPU 时,编译成 NV_fragment_program 比 ARB_fragment_program 会获得优化的性能,在上述实验中我们的程序在 NV_fragment_program 描述下需要 7ms,而 ARB_fragment_program 描述下需要 11ms。

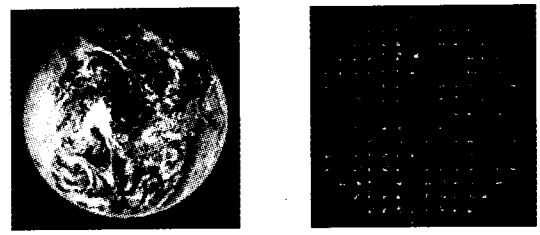


图 5 单幅 RGB 图像及其 8 * 8DCT 数据的可视化结果

3.4 传统快速算法的实现及性能分析

多数 DCT 的快速算法,是以减少计算中相对耗时的乘法运算为目的[8]。以行列分解法为例子,该快速算法也是利用二维 DCT 的性质减少乘法运算量。二维 DCT 的行列分解性质如下所示:

$$Z(k,l) = \sqrt{\frac{2}{M}}c(k)\sum_{m=0}^{M-1}\left[\sqrt{\frac{2}{N}}c(l)\sum_{m=0}^{N-1}x(m,n)\cos\frac{(2n+1)l\pi}{2N}\right]\times\cos\frac{(2m+1)k\pi}{2M} \tag{6}$$

其中 $(k=0,1,\cdots,M-1;l=0,1,\cdots,N-1)$,除去系数相乘的影响,相对于式(1) 的乘法运算量 $N \times M$,该算法的乘法计算量为 $N+M$。3.2 节描述的方法同样适用于行列分解法,在程序循环计算部分稍做改动就可实现行列分解法。通过 cg 编译成 NV_fragment_program 描述后,相对于非快速算法,汇编指令数由原来的 464 行减少为 248 行,乘法指令由 134 条减少到 30 条,但是使用的寄存器从 4 个增加到 6 个。在使用 nvshaderperf 对片元程序性能软件仿真显示,快速算法的速度要更快。对 128 × 128 图像实际测量结果表明,同为 NV_fragment_program 描述,行列法快速算法平均耗时 11ms,比先前用定义法计算的 7ms 还要慢 3ms,和仿真器提供结果不符。图 6 为在 NV_fragment_program 描述下的两种算法的片元程序分别在 128 × 128,256 × 256,512 × 512 大小图像上的实测结果。

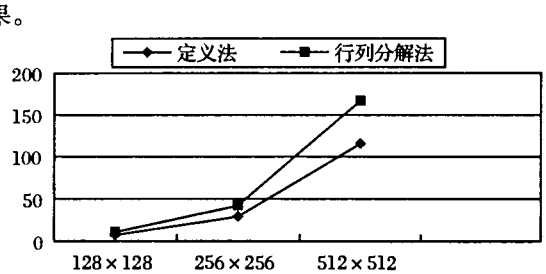


图 6 定义法和行列分解法 GPU 算法效率比较

4 结语

在多遍绘制计算下,中间数据的存取速度仍旧不能满足应用的需要,因而给应用带来不便。在将传统 CPU 计算移植到 GPU 上并行计算的过程中,传统经验认为计算更快、性能更优的方法在图形硬件新环境下也需要重新认识和验证。该 DCT 并行算法在应用中简化了对一副完整图像的划分过程,在具体的应用中综合使用该方法会使 GPU 的硬件加速性能得到更好的发挥。

参考文献:

[1] MACEDONIA M . The GPU enters computing's mainstream [J] . Computer, 2003, 36(10) : 106 - 108.

[2] COMBA JLD, DIETRICH CA, PAGOT CA. Computation on GPUs: From a Programmable Pipeline to an Efficient Stream Processor[J]. Revista Informática Teóricae Aplicada, 2003, X(2): 41 - 70.
[3] 吴恩华,柳有权. 基于图形处理器(GPU)的通用计算[J]. 计算机辅助设计与图形学学报,2004,16(5):601 - 612.
[4] HOPF M, ERTL T. Hardware Accelerated Wavelet Transformations [A]. Proceedings EG/IEEE TCVG Symposium on Visualization VisSym '00[C], 2000. 93 - 103.
[5] 吴仲乐,王遵亮,罗立民. 基于 GPU 的快速 Level Set 图像分割[J]. 中国图象图形学报,2004,9(6):679 - 683.
[6] KRüGER J, WESTERMANN R. Linear Algebra Operators for GPU implementation of Numerical Algorithms[J]. ACM Transactions on Graphics, 2003, 22(3): 908 - 916.
[7] NVIDIA Corporation . NVIDIA OpenGL Extension Specifications [EB/OL] http://developer. nvidia. com /object/nvidia _ opengl _ specs. html, 2004-8-10/2004-11-10.
[8] (日)谷萩隆嗣. 快速算法与并行信号处理[M]. 薛培鼎,徐国弼,译. 北京:科学出版社,2003. 24 - 25.
[9] LAN B, PAT H. Data parallel Computation on Graphics hardware [EB/OL]. http://graphics. stanford. edu /projects/brookgpu/, 2004-8-10/2004-11-10.

(上接第 2176 页)

此种方法可以有效的提高探测的精度,缺点是增加了特征报文的数量。一种有效的改进途径是结合先验知识和实验结果,动态地删除对 $\sum_{\mu(i,j)}(j=1,2,3,\cdots,n)$ 影响小的测试方法和增加影响比较大的测试方法,并且根据大量的统计数据结果提高或者减少 $\mu_{(i,j)}$ 的取值。同传统的严格匹配方法,此方法有了更强的抗干扰能力。

2.2.2 决策层融合方法

决策层融合的数据为主机上开放服务服务与端口的对应关系结论、服务程序版本结论和操作系统类型结论。网络服务与端口对应关系的寻找方法有两种:第一种是查找默认端口/服务对照表,将端口直接对应某个网络服务,网络中服务不开放在默认端口的情况很多,这种直接对应的方法可信度较差;第二种是通过应用层特征匹配的方法找出对应关系,例如在基于监听的方法中,数据包应用层数据的前 3 个字节为"GET"时,可以标识出服务器的端口上开放着 HTTP 服务。显然这两种方法得到的结论有着不同的可信度。同样,不同扫描方法、利用不同特征进行匹配得到的操作系统类型、网络服务类型、服务程序版本都具有不同的可信度。

探测节点首先完成方法可信度的判断,决策层融合中心然后再根据节点的可信度完成最终的可信度判断,找出最佳的结论。可信度用一个 0 ~ 1 的小数表示。设某探测节点利用可信度为 mt_i 的方法得到某个结论 R_i,该探测节点得到该类结论的可信度为 nt_i,则决策层融合中认为该探测节点得到的结论 R_i 可信度为 $mt_i * nt_i$。决策中心计算针对同一问题所有结论的可信度,将可信度最大的作为最终结论。

系统中的每个探测节点上维护一张检测方法与得到结论之间可信度对照表 MT,决策层融合中心中维护一张结论类型与探测节点之间的可信度对照表 NT。其表结构如表 3、表 4 所示。

表 3　方法与结论可信度的对应关系表

结论可信度	方法
M_1	mt_1
M_2	mt_2
…	…
M_n	mt_n

表 4　结论类型与探测节点可信度对应关系

结论类型	探测节点	可信度
R_1	1	nt_1
R_1	2	nt_2
…	…	…
R_i	j	nt_m

2.3 系统知识库

参照 RFC 和已有的各种网络系统信息,本文提出并建立了一个较详细的系统知识库,其内容包括:

1) 默认网络端口与服务类型对照表后门程序开放端口等。

2) 特征关键字表,存放应用协议自身的特征关键字和服务程序特征关键字。

3) 系统协议栈指纹特征表,对应于表 1 的一个协议栈特征库。

4) 探测节点中每种方法与结论可信度的对照表。

5) 结论类型与探测节点可信度的对照表。

3 结语

计算机网络信息发现是一个具有广泛应用背景的课题。随着网络规模的扩大和网络应用技术的发展,针对网络安全、网络管理方面研究的需求,本文提出了基于信息融合的计算机网络信息发现方法。这种方法可以有效的解决单个或单种探测工具在空间和时间上的限制,充分发挥了各种探测工具的优势,具有较强的实用性。

参考文献:

[1] ISS Website[EB/OL]. http://www. iss. com/, 2003.
[2] NMAP Website[EB/OL]. http://www. insecure. org/nmap/index. html, 2003.
[3] ARKIN O. ICMP Usage in Scanning Version 3.0[EB/OL]. http://www. sys-security. com/ archive/papers/ICMP_Scanning_v3. 0. pdf, 2002.
[4] libpcap-0. 7. 2. tar. gz. http://www. tcpdump. org/release/, 2002.
[5] Passive Vulnerability Scanning[EB/OL]. http://www. tenablesecurity. com/nevo. html, 2004.
[6] 何友, 王国宏. 多传感器信息融合及应用[M]. 北京:电子工业出版社, 2000.
[7] YE N, GIORDANO J, FELDMAN J, *et al.* Information fusion techniques for network intrusion detection[A]. Information Technology Conference[C], 1998. 117 - 120.
[8] VALDES A, SKINNER K. An Approach to Sensor Correlation[EB/OL]. http://www. raid-symposium. org/raid2000/Materials/Abstracts/41/corr_approach. pdf, 2000.
[9] SHEYNER O. Scenario Graphs and Attack Graphs[D]. Carnegie Mellon University, 2004.
[10] STEVENS WR. TCP/IP Illustrated, Vol. 1[M]. Addison-Wesley, 1994.
[11] VIVO M, CARRASCO E. A review of port scanning techniques[J]. ACM SIGCOMM Computer Communication Review, 1999, 29(2): 42 - 48.