doi:10.3969/j.issn.1000-436x.2013.z2.003

# 基于 GPU 的 LCS 算法加速机制研究与实现

张常志，牟澄，黄小红，马严
（北京邮电大学 网络技术研究院信息网络中心，北京 100876）

**摘 要**：协议特征识别技术中用到了一种重要的 LCS 算法，它是一种字符串比对算法，提取出字符串中的最长连续公共子串。然而，通过理论分析和实验表明：这个查找过程是一个时间复杂度较高的运算过程，如果输入的数据分组比较大，那么运行的时间将会非常长，为此不得不控制输入数据分组的大小和数量，这严重限制了所采用样本集的大小。提出了基于 GPU 对 LCS 运算实现加速的方法。在此基础上搭建和配置了 CUDA 平台，在此平台下研究并实现了 LCS 算法的并行性。通过对 LCS 算法在 CUDA 下并行性的研究，有效地加快了 LCS 算法的运行速度。实验结果表明，GPU 下 LCS 算法的运行效率比 CPU 有了显著的提高。

**关键词**：协议特征识别；LCS 算法；CUDA 平台；GPU 加速
**中图分类号**：TP393.0 **文献标识码**：A **文章编号**：1000-436X(2013)Z2-0009-05

# Research and implementation of the GPU-based LCS algorithm acceleration mechanism

ZHANG Chang-zhi, MU Cheng, HUANG Xiao-hong, MA Yan
(Network Information Center, Institute of Networking Technology,
Beijing University of Posts and Telecommunications, Beijing 100876, China)

**Abstract:** The LCS algorithm used in protocol feature recognition is a string matching algorithm to extract the longest string of continuous public substring. However, through theoretical analysis and some experimental situation, it can be seen that this process is a time complexity of higher computing process. If the input data packet is relatively large, the running time will be very long. To this end, the size and number of input packets have to be controled, which severely limits the size of the sample set. A GPU based method for accelerating the LCS algorithm was proposed. The CUDA platform was built and dedoyed and the parallel of LCS algorithm was researched on this platform. By the parallel study of LCS algorithm on the CUDA, the operation speed of the LCS is effectively enhanced. Highly competitive experimental results show that the LCS algorithm in the GPU is more effective and efficient than that in the CPU.
**Key words:** protocol feature recognition; LCS algorithm; CUDA platform; GPU acceleration

## 1 引言

基于端口的识别是传统的互联网应用程序识别技术，但这是一种不成熟而且不切实际的互联网应用识别技术。由于当今互联网通信的多样性，所以这种识别方式不能保证精确性。Web 存储和 P2P 文件共享应用程序已经迅速地成长并普及，其通信占据了互联网总流量的很大一部分。尤其是新一代的 P2P 应用纳入了各种绕过防火墙策略，比如临时端口分配和中继节点，以避免被发现和过滤。虽然，由于流量识别中的高通信量和有限的计算资源，使得基于端口的方法在互联网骨干网中仍然是一种

**收稿日期**：2013-09-06
**基金项目**：国家自然科学基金资助项目(61003282)；国家 CNGI 专项基金资助项目：可演进的下一代高智能网络架构研究和实验基金资助项目
**Foundation Items**：The National Natural Science Foundation of China(61003282); China Next Generation Internet (CNGI) Project: Research and Trial on Evolving Next Generation Network Intelligence Capability Enhancement

流行的解决方案，但是这种方法产生非常差并且不准确的应用鉴定结果。

在这种情况下，基于签名的识别技术被提出，这种识别方式消除了以前鉴定方法的任何不确定性。签名是有效负载数据的一部分，这种数据是静态的，它可以被描述为一个字符串或十六进制值序列。这种协议特征识别技术需要事先对协议的语义进行分析。而且，这种特征提取技术需要人为的手动提取。在这个过程中，由人来决定会导致产生一个缓慢的响应时间来处理新应用。而且由于专家的水平不同，签名的质量也有所不同。因此，迫切需要一个系统化的方法来定义和提取互联网应用识别签名。这是一个面向流量识别的系统具有自我更新引擎重要的一步。

LCS 算法[1]是当今信息生物学中最传统也是使用最多和最先被其他领域所借鉴的序列比对算法。但是这一算法的时间复杂度较高，如果输入分组的大小是 10k 左右，那么其运行时间在几秒到几十秒钟不等，在数据分组大小超过 1M 时，运行时间在一天左右。为此不得不控制输入的数据分组的数量，这严重限制了所采用样本集的大小。所以通过 GPU 实现运算加速具有重要的意义。

## 2 CUDA 简介

CUDA[2]（compute unified device architecture），显卡厂商 NVIDIA 推出的运算平台，该架构使 GPU 能够解决复杂的计算问题。本实验正是基于这一平台架构的 GPU 加速算法的实现。

### 2.1 编程模型

在 CUDA 架构下，一个程序分为 2 个部分，host 端和 device 端。Host 端是指在 CPU 上执行的部分，而 device 端则是在 GPU 上执行的部分。Device 端的程序又称“kernel”。Device 端通常作为 CPU(host)的协处理器，它拥有独立的存储设备，能同时启动大量的线程。而且 GPU 的线程非常轻量，线程间切换约 1cycle，而 CPU 需要大约 1 000cycle[3]。

通常 host 端会将数据准备好后，复制到显卡的内存中，再由显示芯片执行 device 端程序，完成后再由 host 端程序将结果从显卡的内存中取回，如图 1 所示。

### 2.2 编程框架

编程框架程序如下[4]。

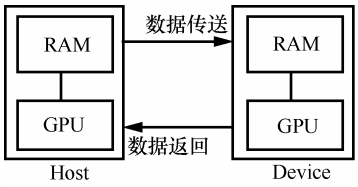



图 1 CPU 和 GPU 数据传送结构

```
//全局变量声明
__host__ , … , __device__ … __global__, __constant__, __texture__
__global__ void kernelOne(…)                    // 内核函数
float handyFunction(…)                          //普通函数
main(){
cudaMalloc(&d_GlblVarPtr, bytes )
cudaMemCpy(d_GlblVarPtr, h_Gl…)
kernelOne<<<execution configuration>>>( args… ); //调用内核函数
cudaMemCpy(h_GlblVarPtr, …);
}
__global__ void kernelOne(type args, …){
__local__, __shared__
}
float handyFunction(int inVar…){ }
```

## 3 算法研究和实现

LCS 算法的通常解法是递归回溯法。这种方法主要是通过矩阵来实现的，把它称之为“矩阵”算法。但是它有很大的局限性，基于此，提出了类似 BF 算法的一种双序列比对算法，称之为“BF”算法。

### 3.1 2 种算法的探索研究及比较

矩阵算法是用一个二维的矩阵来记录 2 个字符串中所有位置的 2 个字符之间的匹配情况，若匹配则为 1，否则为 0。然后求出连续的对角线最长的 1 序列，其对应位置就是最长匹配的子串的位置[5]。

表 1 是字符串 s1=“hang0an”和字符串 s2=“09ngzhan”的匹配矩阵，前者为 $X$ 方向的，后者为 $Y$ 方向的。若对应位置匹配则为 1，否则为 0。

通过查找位置得到最长的匹配子串为 han。但是在 0 和 1 的矩阵中找最长的 1 对角线序列又要花去一定的时间。通过改进矩阵的生成方式和设置标记变量，可以省去这部分时间。下面探索这种算法的实现方式。

**表 1　　2 个字符串按位比较所得矩阵**

| | h | a | n | g | 0 | a | n |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| n | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| g | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| z | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| h | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| n | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

可以采取如下措施：当字符匹配的时候，并不是简单地给相应元素赋上 1，而是赋上其左上角元素的值加一。用 2 个标记变量来标记矩阵中值最大的元素位置，在矩阵生成的过程中来判断当前生成的元素值是不是最大的，据此来改变标记变量的值，那么到矩阵完成的时候，最长匹配子串的位置和长度就已经出来了。规定用变量 max 标记矩阵中的最大元素值，用 maxj 标记最大值所对应的元素在字符串中的位置，这里本文选择 $X$ 方向的字符串。结果如表 2 所示。

**表 2　　经过改进后所得匹配矩阵**

| | | h | a | n | g | 0 | a | n | h |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| n | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| g | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 |
| z | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| h | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 0 |
| n | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 2 | 0 |

通过对比可以看出，这种方法在 2 个序列比对的过程中就可以判断最长子串的位置，而不用再花费时间去寻找最大子串，降低了时间复杂度。

但是，由表 2 可以发现，这种算法存在一个很大的局限。当 2 个字符串比较小时，比如都为 1 k，那么矩阵大小为 1 M，这个可以接受；但是当 2 个矩阵为 100 k 时，矩阵大小就为 10 G，内存根本不可能分配这么大的空间，所以这种算法严重的限制了需要比对的字符串的大小。

上面讨论了“矩阵”算法，了解了它的优缺点，优点是时间复杂度小，缺点是空间复杂度太大，导致不能接受。

BF 算法的基本思想：BF 算法运用在文本搜索领域，具有简单、直接、无需对文本进行预处理等操作，因此被广泛地运用到多种文本检索系统中，但是 BF 算法实际上是一种保利匹配的算法，算法的时间复杂度开销很大。

对于给定的原始字符串，当需要匹配待匹配字符串时，从原始的第一个位置开始匹配，观察是否匹配成功，如果失败，则将位置向后移动一位，继续匹配，以此类推，直到原始字符串最后位置前倒数查询字符串长度的位置截止。

原始的字符串是通过从文件汇总读入标准的字符串序列，而待查询的字符串是通过用户手动输入的一个字符串，实际匹配时通过观察待查询的子字符串是否在文件字符串序列中出现。

而本文提取 2 个字符串的最长连续公共子串的想法就是基于这一思想，称之为“BF-like”算法。

其基本思想为:设有 2 个字符串，str1、str2，长度分别为 $m$ 和 $n$，保持 str2 的长度不变，给 str1 字符串前后分别添加 $n$ 个元素（如‘\0’），而且保证添加的元素不会出现在字符串中。在保持改变后的 str1 不变，str2 和 str1 开始按趟对比，每比对完一趟，str1 座椅一个元素，这样从 str1 第一个元素比较到第 $m+n$ 个元素结束，共比较了 $m+n$ 趟，每一趟都比较 $n$ 次，记录下最长的子串。如图 2 所示。

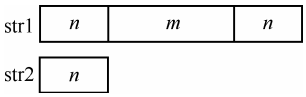



图 2　BF 算法中的字符串比较示意

虽然时间复杂度增大了很多，但是空间复杂度却大大地减少了。比如 2 个字符串都是 100 k，那么只需要 400 k 的空间就可以实现，所以，最终采取了这种算法来编程实现。

### 3.2　CPU 和 GPU 下的算法流程对比

CPU 下程序正常运行，主体函数为自定义的 lcs 函数，执行后返回最大子串长度 max 和位置 maxj；用 clock 函数来计算主体函数运行的时间。

GPU 下算法是对 CPU 下算法的改动，因为 GPU 上的代码运算时必须使用 GPU 上的内存，所以程序的编写过程中必须注意数据在 CPU 的内存和 GPU 的内存上的拷贝，而且由于 GPU 的多线程

并行执行，每个 block 中 thread 限制在 512 个，而 block 之间的线程不能通信，所以还要额外设定 2 个数组用来存储每个线程一趟比较中得到的 max 和 maxj，这 2 个数组的大小和比较的趟数（也是线程数）是一样的，为 $m+n$。

在 GPU 上执行完主体函数 lcs 后，会得到 2 个大小为 $m+n$ 的数组，分别存储着每一趟比较的结果；编写自定义函数 max_substr，判断最长子串的个数及位置，得到结果如图 3 所示。

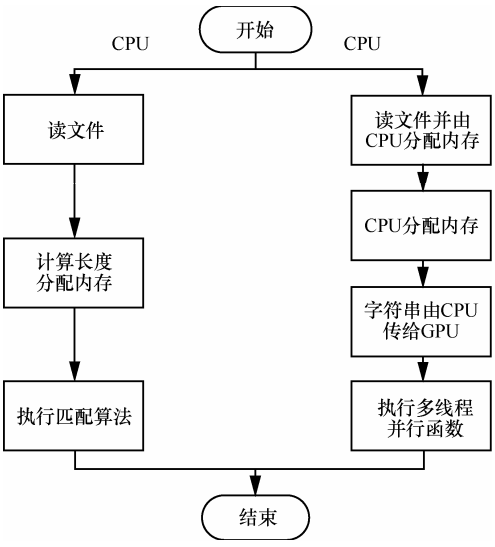



图 3 CPU 与 GPU 算法流程

### 3.3 关键算法（核函数）分析

图 4 是每个 thread 中程序执行的流程图。线程 id 的计算方式为 tid=threadIdx.x+blockIdx.x*blockDim.x; 使用下面提到的线程个数设置可以避免执行过多的无用线程，节约了时间[6]。

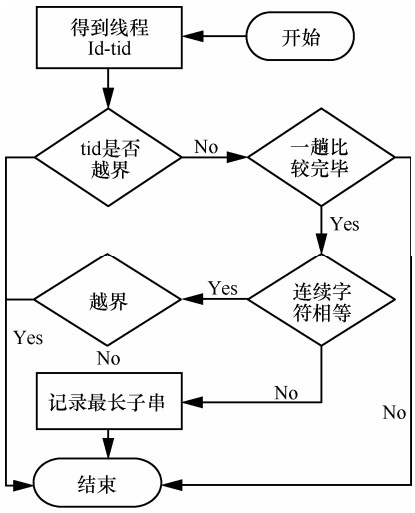



图 4 核函数运行流程

但是从流程图中也可以看出，程序中需要控制的地方非常多，但是控制处理是 GPU 的弱项，这使得每一个 thread 执行的时间随着数据的增大而增大，这些都说明程序的并行性还有待于进一步挖掘。

## 4 实验结果及分析

要测试的是提取 2 个字符串的最大连续公共子串，这 2 个字符串分别写在 2 个 txt 文件内，规定将 str1 写在 a.txt 内，str2 写在 b.txt 内。并且提前设定好它们的最大公共子串为 zhangchangzhi001，给定不同大小的 2 个 txt 样本，分别测试。

本文比较了 5 组数据，从这些数据可以看出，当比对的字符串较小时，在 CPU 下运行的时间和在 GPU 下运行的时间都很短暂，基本为 0，加速比也不是很大，所以这种加速没有什么必要。但是当数据比较大时，可以明显的看出，GPU 的运行时间明显缩短，如表 3 所示。

GPU 和 CPU 的内存都有一定的容量限制，在不超出这种限制的情况下，通过实验测试的几组数据可以预测，加速比（CPU 运行时间/GPU 运行时间）大概固定在 4.5 左右。在 2 个字符串的长度固定的前提下，GPU 下程序所运行的时间大概是 CPU 下程序所运行时间的 4 倍半。

在利用 LCS 和 smith waterman 进行特征提取的时候，如果输入分组的大小是 10 k 左右，那么程序运行时间是在几秒到几十秒钟不等。在数据分组大小超过 1 M 时，程序运行时间在一天左右。现在将时间缩短到小于 1/4，很大程度上节约了时间。

**表 3 经过改进后所得匹配矩阵**

| 项目 | 数值 | | | | |
|---|---|---|---|---|---|
| Str1 大小/k | 3.31 | 15.1 | 56.6 | 103 | 150 |
| Str2 大小/k | 3.50 | 15.6 | 56.6 | 104 | 150 |
| CPU 运行时间/s | 0.00 | 3.00 | 47.0 | 165 | 342 |
| GPU 运行时间/s | 0.00 | 0.00 | 11.0 | 37 | 78 |
| 加速比 | 0.00 | 3.00 | 4.30 | 4.5 | 4.4 |

通过程序的运行结果可以看出，其实在 GPU 上运行的核函数，一致性并不是很好。这种一致性是指，比如说矩阵的相加，每个线程里面的控制语句部分其实都是一样的，只有运行的数据有所不同，也就是说每个线程运行的时间基本上应该是一样的，这样的程序并行性非常好，执行起来效率也非常高。但是本程序中，每个线程里面程序的控制

语句都不相同，执行的时间复杂度也不相同，每个线程执行的时间要依据所检测的字符串的大小来确定，所以这个程序中就需要占用 GPU 更多的程序控制单元。但是就如先前所讲，GPU 目前更多的是数据计算单元，如果线程里面的控制语句和分支语句等需要控制单元的语句很多，那可能并行性就不会很理想。

对于这个程序来说，加速比比较理想，若想要再提高加速比，就要简化 GPU 中线程执行的过程中控制语句的使用。

## 5　结束语

从传统来说，因特网的应用程序能够通过预先定义好的、众所周知的端口来进行识别，但是又对这种端口的精确性有所怀疑。如应用层签名所描绘的一种可供选择的方法，此方法包括对可靠签名的详细的探索，而且伴随着更加有前途的精确性。由于有着优先的协议知识，签名的生成过程能确保高精准性。当越来越多的应用开始使用私有的协议时，获得精准签名的同时避免时间消耗和人工签名的过程就变得越来越困难了。

这就涉及到了流量的自动识别，而这种自动识别的方法用到的一个主要算法就是 LCS，但是当前的算法所需的时间消耗太大，这就迫切需要对算法运行的时间进行压缩。但是，很明显，CPU 的本身特性就是顺序执行，在这上面不太可能有什么大的突破，除非找到一种更加有效的算法。

近年来 GPU 的发展速度很快，它对大数据具有很好的并行处理能力，如果某种算法能够并行执行，那么使用 GPU 绝对可以大幅度地减少程序对时间的消耗。

## 参考文献：


[1] 王映龙, 宋泽锋, 陈卓等.基因序列相似程度的 LCS 算法研究[J]. 2007, 43(31):45-47.
WANG Y L, SONG Z F, CHEN Z, *et al*. LCS algorithm research for gene sequence similar degree[J].Computer Engineering and Applications. 2007, 43(31):45-47.

[2] 吴长茂, 张聪品, 张慧云等. CUDA 平台下多核 GPU 高性能并行编程研究[J]. 河南机电高等专科学校学报, 2011, 19(1): 19-21.
WU C M, ZHANG C P, ZHANG H Y, *et al*. Research on parallel program of high performance computing based on CUDA platform for multicore GPU system[J]. Journal of Henan Mechanical and Electrical Engineering College, 2011, 19(1): 19-21.

[3] 张舒, 褚艳丽. GPU 高性能运算之 CUDA[M]. 北京: 中国水利水电出版社, 2009.
ZHANG S, CHU Y L. GPU Performance Computing of CUDA [M]. Beijing: China Waterpower Press, 2009.

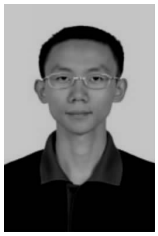
[4] SANDERS J. GPU 高性能编程 CUDA 实战[M]. 北京: 机械工业出版社, 2011.
SANDERS J, NIE X J (Translator). CUDA by Example:an Introduction to General-Purpose GPU Programming[M]. Beijing: China Machine Press. 2011.

[5] 彭江锋. 基于 CPU+GPU 异构平台的字符串匹配算法研究与实现[D]. 华南理工大学, 2011.
PENG J F. CPU + GPU-based Heterogeneous Platforms String Matching Algorithm and Implementation[D]. South China University of Technology, 2011.

[6] 孙兴文. 并行算法设计及编程基本方法[J]. 零陵学院学报(教育科学), 2004, 2(4): 182-184.
SUN X W. Parallel algorithm design and programming basic method[J]. Journal of Lingling University, 2004, 2(4): 182-184.


**作者简介：**


**张常志**（1992-），男，内蒙古赤峰人，北京邮电大学硕士生，主要研究方向为网络监控及流量分析。



**牟澄**（1984-），男，贵州六盘水人，北京邮电大学博士生，主要研究方向为网络监控及流量分析。

**黄小红**（1978-），女，广东河源人，博士，北京邮电大学网络技术研究院信息网络中心主任，主要研究方向为下一代互联网关键技术。

**马严**（1955-），男，北京人，北京邮电大学教授、博士生导师，主要研究方向为基于 TCP/IP 网络的网络管理技术、网络安全技术、移动 IP 技术、IPv6 技术及其应用。