基于 CUDA 的 Wu-Manber 多模式匹配算法①

马 计，王国平，杨 明

(复旦大学 计算机科学技术学院，上海 200433)

摘 要：多模式匹配是计算机科学中最基本的问题，其应用在许多领域，在一些情形下也是比较耗时的。GPU 拥有比 CPU 更强的并行计算能力，随着 CUDA 架构的推出，GPU 用于通用计算领域的并行编程工作变得更加轻松。实现了基于 CUDA 架构的 Wu-Manber 多模式匹配算法，实验结果表明，相比传统串行算法而言，本文的实现获得了 10 倍以上的加速。

关键词：多模式匹配；GPU；CUDA；Wu-Manber

Wu-Manber Multi-pattern Matching Algorithm Based on CUDA

MA Ji, WANG Guo-Ping, YANG Ming

(School of Computer Science, Fudan University, Shanghai 200433, China)

Abstract: Multi-pattern matching is a basic problem in computer science and used in many fields, in some cases, also the most time-consuming. GPU has more parallel computing capabilities than the CPU. With the introduction of CUDA, GPU computing for general purpose parallel programming becomes easier. This paper proposes Wu-Manber multi-pattern matching algorithm based on the CUDA, and evaluating the implementations we have achieved speedups up to 10 faster than the sequential implementations.

Key words: multi-pattern matching; GPU; CUDA; Wu-Manber

1 引言

多模式字符串匹配问题是计算机信息处理的最基本问题。它在信息检索，模式识别，病毒检测等领域具有广泛应用。在当今信息资源丰富，信息流动频繁的时代，传统的基于 CPU 的串行模式匹配算法存在速度慢，效率低等缺点。因此，研究并设计更加快速的模式匹配并行算法具有重要的理论价值和实际意义。

近年来，GPU 得到了迅速的发展，GPU 已经不再局限于图形处理领域，在通用计算方面也逐渐崭露头角，其已经应用在图形动画、科学计算、生物、物理模拟等领域。事实也证明在浮点运算、并行计算等部分计算方面，GPU 可以提供数十倍乃至于上百倍于 CPU 的性能。而且，随着 CUDA 通用并行计算架构推出，GPU 编程也正变得和 CPU 编程一样简单。自 CUDA 平台推出后，许多学者提出了基于 GPU 的模式匹配算法，获得了较好的加速效果。文献[1]在 GPU 上移植了几种经典的单模式匹配算法。文献[2]提出了基于 GPU 的 AC 多模式的 NIDS 算法，比传统的 NIDS 快了两倍。文献[3]实现了基于 GPU 的 BNDM 多模式网页匹配的程序，并取得了 13 倍以上的加速。本文实现了基于 CUDA 的 Wu-Manber[4]算法并予以优化。

本文余下的部分组织如下：第 2 节介绍了 CUDA 的编程模型以及 Wu-Manber 算法，第 3 节是关于 Wu-Manber 算法在 CUDA 下的实现与优化，第 4 节是实验结果与分析，最后是结束语。

2 相关技术介绍

2.1 CUDA 编程模型

CUDA(Compute Unified Device Architecture)[5]是 NVIDIA 公司推出的一种全新的软件和硬件架构，它将 GPU 视为一个并行数据计算的设备，对所进行的任务进行分配和计算。CUDA 编程模型[5,6]将执行的程序分为两部分：Host 端与 Device 端。Host 端是指在 CPU 上执行的部分，Device 端则是在 GPU 上执行的部分，

① 收稿时间:2011-06-20;收到修改稿时间:2011-07-23

在 Device 上运行的函数称为内核函数(Kernel)，内核函数不是一个完整的程序，只是 CUDA 程序中可以被并行处理的步骤。

在运行时，CUDA 会产生许多并行的线程，线程是 CUDA 中基本的执行单元，每个线程都有唯一的标识，都会执行 Kernel 函数的程序段，根据其标识的不同，获取不同的数据进行计算。CUDA 的程序执行流程[5]如图 1 所示，由图可知在 Device 上执行的线程(Thread)以网格(Grid)的形式组织。每一个网格由若干个块(Block)组成，每一个块又包含多个线程。网格和块都能够以多维的形式组织。

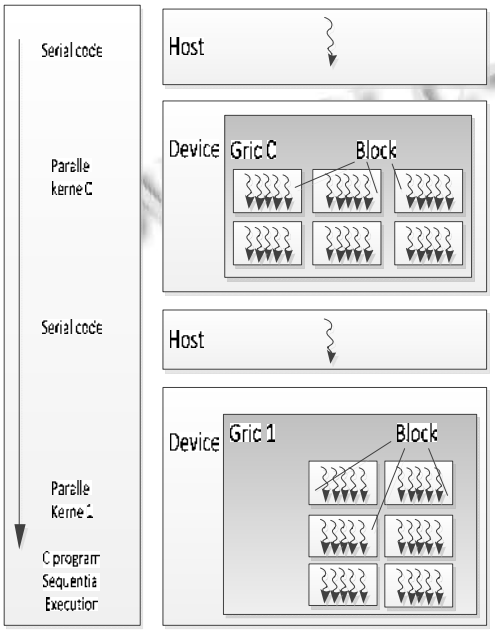

图 1　CUDA 程序流程

CUDA 中包含多种存储空间，合理使用这些存储空间能够提高数据访问速度，节省带宽，提升程序的性能。

2.2 Wu-Manber 算法介绍

Wu-Manber 算法是一种将过滤思想和单模式 BM(Boyer-Moore)[7]算法结合起来的多模式匹配算法。Wu-Manber 采用了 BM 的坏字符规则，与 BM 算法利用单个字符计算 SHIFT 表不同的是 Wu-Manber 使用块字符计算 SHIFT 表，块大小记为，一般为 2 或 3。在匹配的时候使用散列表过滤一些不必要的匹配，因此它在多模式匹配算法中有着较高的效率。

Wu-Manber 算法分为预处理和模式匹配两个阶段。在预处理阶段根据模式串集合得到所有模式串中的最短模式长度 m，并构造三个表：SHIFT 表，HASH 表和 PREFIX 表。SHIFT 表中存储字符集中所有块字符在待查文本中出现时的转移距离。HASH 表用来存储匹配窗口内尾块字符散列值相同的模式串。PREFIX 表用来记录匹配窗口内首块字符散列值相同的模式串。其中 SHIFT 表按如下方法计算：

(1) 用 $m-B+1$ 初始化 SHIFT 表，这里假定待匹配文本的块字符不在所有模式串窗口内出现时，匹配窗口跳转的最大安全距离。

(2) 考察所有模式串的前 m 个字符内的块字符，即匹配窗口的大小为 m，h 是块字符对应的哈希值，假设一个匹配窗口内的某个块字符在某个模式串 p_j 的第 q 位置出现，且在其它模式串中前 m 个字符大小匹配窗口内，该块字符的位置都不大于 q 或根本不存在，那么就更新 SHIFT[h]的值为 $m-q$，而不再是，以防忽略可能的正确匹配。

算法的匹配阶段，就是利用上面建立的三个表完成对待查文本的扫描和寻找匹配的过程，匹配窗口 T 的大小为 m，$B=2$，具体可分为如下几步[4]：

(1) 考察当前 T 内的块字符，计算其尾块字符 $T_{m-B+1}...T_m$ 的哈希值为。

(2) 检查 SHIFT[h]的值，若 SHIFT[h]>0, 将 T 向右滑 SHIFT[h]大小，转到(1)，否则转到(3)。

(3) 计算该窗口 T 的前缀即 $T_1...T_B$ 的哈希值，记为 text_prefix。

(4) 对于符合 HASH[h] $p \le <$HASH[$h+1$]所有 p，若 PREFIX[p]text_prefix，则把文本和模式串进行完全匹配，若有完全匹配则报告结果。$T = T + 1$，转(1)。

3　Wu-Manber算法在CUDA中的实现

3.1 GPU 上的普通实现

由 2.2 我们知道，Wu-Manber 算法可分为两个部分：预处理和模式匹配。预处理部分主要的任务就是构造三个表：SHIFT，HASH，PREFIX。这三个表可以一次构造，反复使用，用于大量数据的匹配。通常模式匹配部分是整个程序中最耗时的部分，所以我们把计算任务繁重的部分交由 GPU 并行计算，预处理则让 CPU 执行。通过这种协同处理计算充分利用了 GPU 的高性能特点提升了程序的性能，基于 GPU 的算法流

程如图 2 所示。

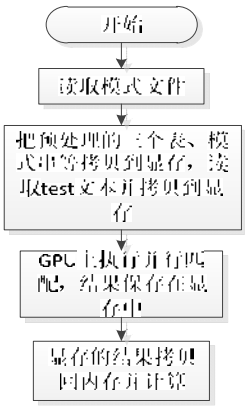

图 2 基于 GPU 的 Wu-Manber 算法字符串匹配流程图

因为 Wu-Manber 算法是一次扫描文本就能完成对所有模式的匹配，所以不宜采用为每个线程分配部分模式。所以我们将文本数据进行划分，为每个线程分配 PAGESIZE bytes 大小的数据段去匹配所有模式，以达到并行处理的目的。

然而这样把数据分段的方法有可能遗漏数据段交界处的匹配。为了避免遗漏，必须使两线程交界处有一定的重叠区域，重叠区域的长度为模式串集中的最大模式长度，记为 MaxPatternLen，如图 3 所示。为了

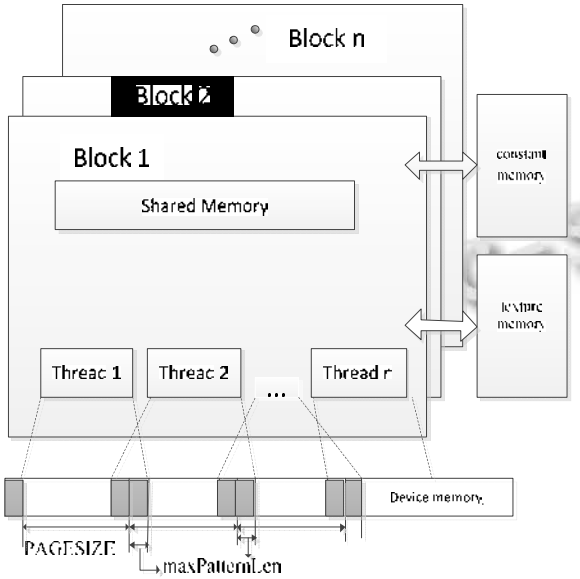

图 3 线程分配数据示意图

消除重叠区域的重复匹配问题，程序采取两个 Kernel 函数先后在 GPU 上执行，它们分别为记 Kernel1，Kernel2。其中 Kernel1，为每个线程分配 PAGESIZE + 2*MaxPatternlen bytes 大小的数据。Kernel2 用于处理 Kernel1 函数的线程交界处的区域即图 3 阴影部分。由前面可知，每段的阴影区域数据的大小为 2* MaxPatternlen bytes。

预处理后，在 Host 端除了要把三个表拷贝到显存中，还要把待匹配的所有模式串都拷贝到显存中，并且这些模式串是经过预处理后排序好的。为了简化运算，我们在 Host 端定义结构体，结构体要大于 MaxPatternLen，以便结构体数组能保存所有模式串。把经过排序好的所有模式串放在结构体数组中，然后传送到显存，这样在 GPU 上进行完全匹配时根据偏移大小就很容易找到具体模式串的位置。例如对于只有 match、pattern、better 、string 的模式串，它们最小的模式串长度为 5，记 m =5，最大长度为 7，记 MaxPatternLen=7，对每个模式串的前位的后两位即 $m-2$，$m-1$位，计算其哈希值，按从小到大排序。结构体大小至少 8，图 4 为排序后保存在结构体数组的情形。

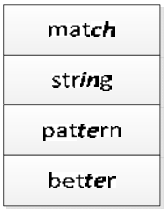

图 4 模式串按截取部分哈希值排序示意图

3.2 GPU 上的优化实现

在 3.1 的实现中，预处理产生的三个表，整个模式串集以及待查文本都被传送到 Device 端的全局内存(Global Memory)中。在 CUDA 的存储器结构中，全局内存是最慢的，其延迟多达几百个时钟周期。在 CUDA 中还有比全局内存访问延迟小的存储空间可以使用，包括常量内存(Constant Memory)，纹理内存(Texture Memory)，共享内存(Shared Memory)等。NVIDIA 的硬件提供了 64KB 的常量内存，并且在芯片上有 Cache，并且其里的内容是只读的。与从全局内存中读取数据相比，从常量内存中读取相同的数据可以节约内存带宽。在某些情形下，使用常量内存可以提升程序的性能，特别是所有线程都访问相同的只读数据时，可明显地提升性能。

与常量内存一样，纹理内存的内容也是只读的，也有缓存在芯片上。在某种情形下，它能够减少显存

的请求，并提供更高效的带宽。

综上所述，提出了一种上节的优化实现。对于多模式匹配来说，往往模式串的数量很大，我们假定模式串数量为 2000，其中模式串中最大长度为 17B，为了能够保存所有模式串，我们把保存模式串的结构体大小设为 20B，那么所需结构体数组的大小约为 40KB，由 Wu-Manber 算法特点可知 PREFIX 表为 4KB，HASH 表 128KB, SHIFT 表为 64KB，且这些内容都是只读的。为了提升性能，我们把所有模式串与 PREFIX 表存储在常量内存中，把 HASH 和 SHIFT 表存储在纹理内存中。对于查找的纯文本，通常几十兆，上百兆甚至更大，且各个线程各自匹配，不需要相互通信，因此我们采用纹理存储空间而不采用共享内存空间。

4 实验结果与分析

4.1 实验环境

操作系统：Windows XP2；

CPU：Intel core2 E7200；

GPU：GeForce 9800GT；

其它：Visual Studio 2008；CUDA3.2；2GB RAM；

4.2 实验结果

本文实现了基于 CUDA 的多模式匹配算法，为了验证其实际执行效果，也实现了的传统的串行 Wu-Manber 算法。

实验的待匹配文本大小为 32MB，模式串的数量分别为 100、200、500、1000、1500 和 2000，它们的长度在 6 到 17 之间。为了测试较好的 Grid 和 Block 的组织形式，在未优化情况下，用模式串数量为 1000，来获得不同线程组织形式的加速比，实验结果如表 1 所示。由表 1 可知，当 Grid 为 512，Block 为 128 时，加速比最好。利用表 1 中的最好的线程组织形式，来测试不同模式数下的优化前后的加速比，实验结果如图 5 所示。

表 1　不同 Grid，Block 组织下的加速比

Grid	Block	PageSize(KB)	Speedup
512	128	0.5	9.42
128	512	0.5	错误
256	128	1	7.43
128	256	1	7.26
128	128	2	7.31
128	64	4	6.12

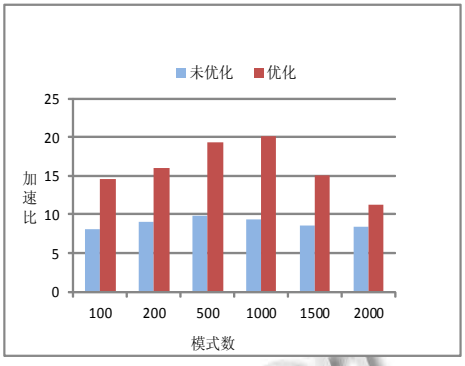

图 5　不同模式数优化前后的加速比

实验表明，当模式数在 1000 以内时，随着模式数的增加，加速比也在增大，模式数为 1000 时，加速比达到了最大。但是当模式数为 1500 与 2000 时，加速比反而递减，但优化后的加速比依然达到 10 倍以上。加速比下降的原因可能是 Wu-Manber 算法本身的特点，比如需要反复查找三个表，当模式串数量很大时，各个字符块在模式串截取的尾端的几率也变大，模式串移动的距离也就减小。另外的原因可能是受实验所用硬件的条件限制，本实验的 GPU 只有 112 个流处理器，所以采用最新的 GPU 会取得更好的加速效果。

5 结语

本文设计了一种基于 CUDA 的多模式匹配的方案并对其进行了优化，结果表明，将具有高性能计算能力的 GPU 用于多模式匹配领域，能够取得较好的加速效果。从实验中可以看出，为了更好的发挥 GPU 的并行处理能力，需要对设备端访存进行优化。近年来，系统中包含多个 GPU 也日趋常见，下一步的工作将研究模式匹配算法在多 GPU 上的实现与优化，以及多 GPU 计算在其它领域的应用。

参考文献

1 Kouzinopoulos CS, Margaritis KG. String Matching on a Multicore GPU Using CUDA. Proc. of the 13th Panhellenic Conference on Informatics, 2009:14–18.

2 Vasiliadis G, Antonatos S, Polychronakis M, Markatos EP, Ioannidis S. Gnort:High performance network intrusion detection using graphics processors. Proc. of the 11th International Symposium on Recent Advances in Intrusion

（下转第 175 页）

获得的结果可知笔记本{3}拥有最高综合满意度，而笔记本{1，2，3，5，6}在属性上是让步满意的，因此推荐顺序应为{3，6，4，1，2，5}，第 3 件商品最符合用户的需求。

值 属性 / 笔记本	价格(元)	保修年限(年)	重量(kg)
1	5000	2	1.6
2	6000	2	1.4
3	4500	2	1.4
4	3000	1	1.8
5	7000	2	1.2
6	3500	1	1.6

图 5 笔记本属性

笔记本的价格，保修年限，重量三个属性的满意度如图 6 所示：

满意度 属性 / 笔记本	价格	保修年限	重量
1	0.6	1	0.6
2	0.4	1	0.8
3	0.7	1	0.8
4	1	0.5	0.4
5	0.2	1	1
6	0.9	0.5	0.6

图 6 笔记本属性满意度

3 结语

随着电子商务网站提供的商品不断增多，商品推荐难度越来越大，本文给出了一种商品推荐方法，以模糊逻辑算法为基础，建立商品满意度矩阵，同时结合商品类型推荐因素矩阵，得到综合满意度值，旨在提高推荐的效率和准确性。

本文提到的基于模糊逻辑的商品信息处理方法可以作为用户在电子商务网站上的购买支持，为了提高推荐的准确性，参与运算的属性权重值较为重要，商品类型推荐因素权重表的建立是之后工作的重点，将在本系统的基于遗传算法的知识学习模块中进行处理。

参考文献

1 周惠宏等.推荐技术在电子商务中的运用综述.计算机应用研究,2004,(1):8–9.

2 刘玮.电子商务系统中的信息推荐方法研究.情报科学,2006,(2):10–11.

3 黄晓斌.网络信息挖掘.北京:电子工业出版社,2005.5–6.

4 黎星星.电子商务推荐系统研究.计算机工程与科学,2004,26(5):7–10.

5 Budan IA, Graeme H. Evaluating WordNet-Based Measures of Semantic Distance. Comutational Linguistics, 2006,32(1): 13–47.

6 扎德,陈国权.模糊集合、语言变量及模糊逻辑.北京:科学出版社,1982.15–16.

7 Shipley MF, de Korvin A, Omer K. A fuzzy logic-based decision model to satisfy goals for successful product/service introduction. European Journal Operational Research, 2001(133): 209–219.

8 Dubois D, Rade HP. Addition of interactive fuzzy numbers. IEEE Trans, Automat. Contr. AC, 1981(26):926–936.

9 Ryu YU. A hierarchical constraint satisfaction approach to product selection for the electronic shopping support. IEEE Trans, Syst Man Cybern, Part A, Syst Humans, 1999(29):525–532.

10 李擎,郑德玲,唐勇,陈占英.一种新的模糊遗传算法.北京科技大学学报,2001.

（上接第 54 页）

Detection。 2008:116–134.

3 Peng JF, Chen H. CUgrep: A GPU–based high performance multi–string matching system. International Conference on Future Computer and Communication, 2010:77–81.

4 Wu S, Manber U. A fast algorithm for multi-pattern searching. University of Arizona, Technical Report TR– 94–17, 1994.

5 NVIDIA Corporation.NVIDIA CUDA Programming Guide 3.0..http//www.nvidia.com/.

6 张舒,褚艳利,赵开勇,张钰勃.GPU 高性能运算之 CUDA.北京:中国水利水电出版社,2009.

7 Boyer RS, Moore JS. A fast string searching algorithm. Communications of the ACM, 1977,20(10):762–772.