第37卷 第14期 计 算 机 工 程 2011年7月
Vol.37 No.14 Computer Engineering July 2011

·开发研究与设计技术· 文章编号：1000—3428(2011)14—0265—03 文献标识码：A 中图分类号：N945

基于GPU的位并行多模式串匹配研究

赵光南，吴承荣

(复旦大学计算机科学技术学院，上海 200433)

摘 要：图形处理器(GPU)具有较强的单一运算能力及高度并行的体系结构。根据上述特点，选择基于位并行技术的多模式串匹配算法M-BNDM，将其移植到GPU上加以实现和优化。通过对需要处理的数据进行预处理，将串匹配的过程简化为更适合CUDA计算数据的位操作。对基于CUDA架构的并行串匹配算法的性能影响因子进行分析。实验结果表明，与同等CPU算法相比，该算法能够获得约十几倍的加速比。

关键词：图形处理器；多模式字符串匹配；位并行；M-BNDM算法；加速

Research on Multiple Pattern String Matching with Bit-parallelism Based on GPU

ZHAO Guang-nan, WU Cheng-rong

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

【Abstract】Considering the strong computing ability and a high degree of parallel architecture of Graphic Processing Unit(GPU), the paper chooses one of multiple string match algorithms based on bit-parallelism, called M-BNDM algorithm, which is to be implemented on GPU and optimized. The process for string matching is simplified to bit operation that is more suitable for data computing of Compute Unified Device Architecture (CUDA) through data preprocessing. Experimental result shows the solution is about 10 times faster than equivalent CPU algorithm. Furthermore, some factors that will infect string matching performance are analyzed.

【Key words】Graphic Processing Unit(GPU); multiple pattern string matching; bit-parallelism; M-BNDM algorithm; speed up

DOI: 10.3969/j.issn.1000-3428.2011.14.090

1 概述

虽然串匹配是一个非常经典的问题，而且至今已被广泛研究，并衍生出众多不同的匹配算法[1]，但当前却面临着较大的挑战：处理数据的规模越来越大，匹配效率要求越来越高，尤其是在信息检索、信息过滤和计算生物学等领域。同时，基于图形处理器(Graphic Processing Unit, GPU)的通用计算已经成为了计算领域发展的新趋势。目前，高端单芯片GPU的单精度浮点处理能力达到1 Tflop/s，存储器带宽达到141 GB/s，均已超过主流CPU的10倍以上。而且与硬件方法相比，基于GPU的软件在开发成本费用和开发周期方面则有较大的优势。

本文研究Tesla GPU体系架构和CUDA编程体系，根据CUDA架构的特点选取基于位并行技术的多模式串匹配算法M-BNDM在GPU上进行实现，并与现有的基于GPU的有串匹配算法进行比较。

2 相关工作

2.1 模式匹配算法分析

串匹配算法是一个基础算法，它的解决以及在这个过程中产生的方法对计算的很多问题都产生了巨大的影响。串匹配算法分为单模式匹配和多模式匹配。前者每次匹配一个模式串，后者一次匹配多模式串。其中多模式匹配算法备受重视，许多学者对多模式匹配在不同环境下的快速算法进行深入研究[2-4]。

2.2 基于GPU的已有串匹配算法

文献[3]提出一种在网络入侵检测系统中基于GPU的多模式匹配算法，该方法在进行GPU计算前使用哈希的方法对规则做预处理，这样减少了不必要的匹配，其实验结果表明该方法的性能是Snort中改进的Wu Manber算法性能的2倍。

文献[4]设计的基于CUDA编程的网络入侵检测系统Gnort实现了A-C算法在GPU上的移植，它有效地利用了GPU计算密集型、高带宽的特点，性能比Snort系统高2倍。

3 统一计算设备架构(CUDA)介绍

3.1 CUDA概述

CUDA是由硬件和软件2个部分组成[5]，在硬件方面，指的是Tesla及其衍生的硬件架构，而在软件方面，包括了CUDA C编译器以及围绕Tesla的各种开发工具。透过CUDA的扩展工具，程序员可以直接为针对通用用途的大规模并行处理器编写程序，而不是透过图形API给图形处理器编写程序，如图1所示[5]。

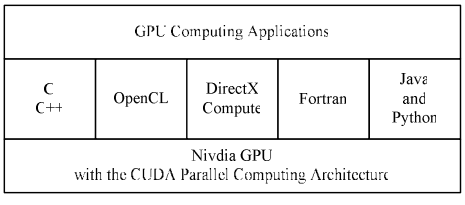

图1 CUDA被设计为支持多种语言或编程接口

3.2 Tesla GPU体系架构

Tesla体系架构的设计重点是能够在处理数据并行问题时获得很高的数据吞吐量。为了提高芯片整体的性能，单个

作者简介：赵光南(1986－)，男，硕士研究生，主研方向：网络安全；吴承荣，副教授

收稿日期：2011-03-16 **E-mail：**leozgn@gmail.com

线程的性能和执行时间做出了牺牲。这与通用处理器形成了反差：通用处理器的设计重点是提高单个线程的性能，减少执行和通信延迟，而不在计算吞吐量上。

3.3 CUDA 编程模型

CUDA 是一种使用 C 扩展语言的细粒度并行计算平台，它通过线程组层次结构、共享存储器、屏蔽同步提供了细粒度的数据并行化和线程并行化，嵌套于粗粒度的数据并行化和任务并行化之中，从而将问题分解为更小的片段，以便通过协作的方法并行解决[5]。CUDA 编程模型的重点是让 GPU 与 CPU 协同工作，在计算高度数据并行任务时发挥作用。NVIDIA 将使用 GPU 通用计算的模型进行了抽象，将图形硬件中运算单元组织层次结构抽象为线程组层次结构，图 2 显示了这种抽象的大致对应关系。

(a)GPU 中运算单元的组织层次

(b)相应的 CUDA 中线程组织层次

图 2 运算单元的组织层次及对应的线程组织层次

4 基于 CUDA 的 M-BNDM 算法

4.1 位并行技术的串匹配

位并行技术利用了计算机机器字位运算的内在并行性，这样就可以利用一次操作或者运算来实现原本需要若干次操作或运算才能完成的功能。其中，位并行的串匹配典型代表是 Shift-And 算法和 BNDM(Backward Nondeterministic Dawg Matching)算法[1]。

BNDM 算法使用位并行来识别子串，具有更好的引用局部性。在当前搜索窗口内，设已读入的字符串为 u，BNDM 算法维护一个集合，记录 u 在 p 中的所有出现位置。该集合可以用一个位向量 $\boldsymbol{D}$ 来表示。如果子串 $p_jp_{j+1}\cdots p_{j+|u|-1}$ 等于 u，那么 $\boldsymbol{D}$ 的第 m-j+1 位是 1，表示 p 的位置 j 是一个活动状态。图 3 显示了这种关系。该集合 $\boldsymbol{D}$ 可以用一个机器字 $\boldsymbol{D}=d_md_{m-1}\cdots d_1$ 来表示，记录了 u 在 p 中的所有出现的开始位置。

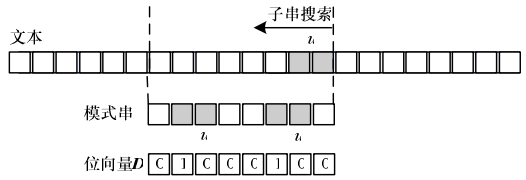

图 3 子串搜索的位并行方法

当读入一个新的文本字符 σ 时，要从 $\boldsymbol{D}$ 更新到 $\boldsymbol{D}$'。$\boldsymbol{D}$'的一个活动 j 对应于 σ_u 在模式串中的一个起始位置，也就是说：u 出现在模式串的位置 j+1，即 $\boldsymbol{D}$ 的第 j+1 位是活动的且 σ 在模式串的位置 j 处出现。BNDM 算法预先计算一张表 B，表 B 用一个位掩码记录了字符在 p 中的出现位置，可利用公式 $\boldsymbol{D}'\leftarrow(\boldsymbol{D}<<1)\&B[\sigma]$更新 $\boldsymbol{D}$ 到 $\boldsymbol{D}$'中。

M-BNDM 算法是 BNDM 算法扩展到多模式匹配的情况[1]。用位并行方法搜索一组模式串 $P=\{p^1,p^2,\cdots,p^r\}$是非常高效的。与 BNDM 的区别是：每次移位后，都需要将一些比特位清零，防止在搜索 p^i 时对 p^{i+1} 形成干扰。变量 *last* 同样被使用只不过现在它表示的位置是其中某一个模式串的前缀开始的地方。

4.2 M-BNDM 算法的 CUDA 实现

4.2.1 算法框架和结构

串匹配中处理的数据主要是输入待匹配的文本和一个具有多个关键词的集合。把输入的文本划分成很多页 page，用不同的流处理器(SP)处理不同的 page 来实现并行；而关键词集合可以划分成多个更小的子集，然后由不同的流多处理器(SM)来负责一个或几个这些子集对输入文本的匹配，实现并行化。对于 SM，不同的 SM 利用不同的每个关键词子集去处理相同的 page；对于 SP，在同一个 SM 上的不同的 SP，它们处理同一个关键词子集，但是处理不同的文本 page。程序框架如图 4 所示。

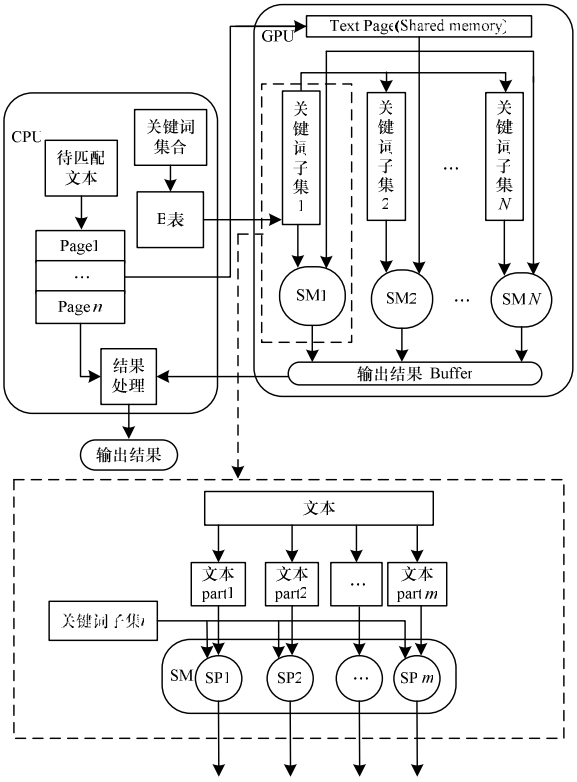

图 4 程序框架图

4.2.2 CPU 上的数据预处理

M-BNDM 算法分为两个部分：预处理和模式匹配。预处理的主要任务是构造 B 表和文本划分。这是一个串行过程，难以在 GPU 上移植。

B 表：BNDM 算法中定义的位串数组。它需维护一个集合，记录关键词中每个字符在关键词中出现的位置。关键词为“google”的 B 表如图 5 所示。

0	0000 0100 0000 0000 0000 0000 0000 0000	‘e’
1	1001 0000 0000 0000 0000 0000 0000 0000	‘g’
2	0000 1000 0000 0000 0000 0000 0000 0000	‘l’
3	0110 0000 0000 0000 0000 0000 0000 0000	‘o’

图 5 B 表

输入文本预处理：将输入的待匹配的文本按规定大小 P_SIZE 进行分割，分割成 tSize/P_SIZE 个页面(tSize 为整个文本大小)。对于最后一页不足 P_SIZE 大小的文本将填充至 P_SIZE。为了不漏检和不重复检测信息，采取连续页间信息“重叠”的方案，前一页尾部 maxPLength 的内容和后一页的头部 maxPLength 的内容是一样的，maxPLength 为最长关键

词的长度。

4.2.3　GPU 上的算法匹配实现

M-BNDM 算法简单易懂并且易于实现，特别是使用内存较少，这对于目前的显卡有限的显存非常有实际意义。

GPU 主要实现 M-BNDM 算法中的匹配过程，执行分为多个 block，若干个 block 组成一个 group，group 内的不同 block 处理不同的关键词子集，每个 block 里的每个 thread 处理一个关键词，每一个 group 处理所有的关键词，不同 group 处理相同的关键词去匹配不同的 page。每个 thread 都相互独立地利用 M-BNDM 算法去匹配 page。程序流程如图 6 所示。

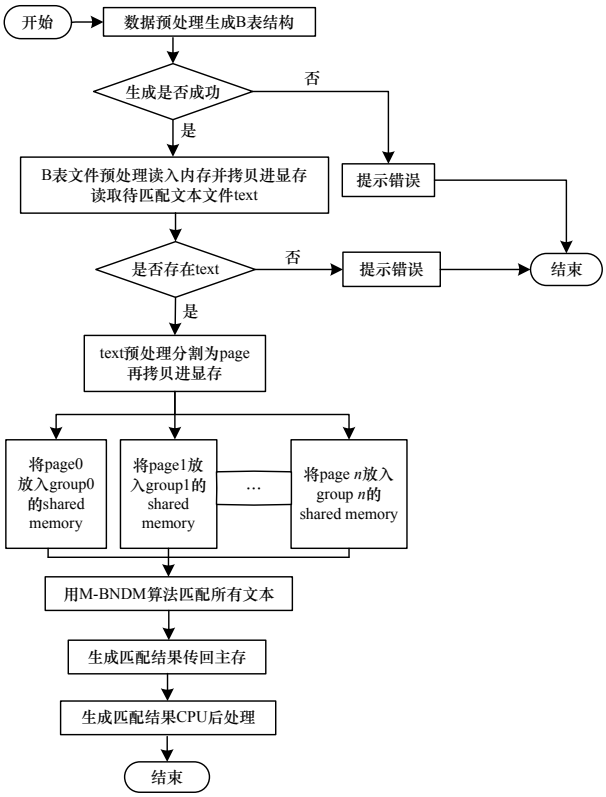

图 6　程序流程

4.2.4　GPU 程序的优化

在 CUDA 的使用中，数据结构和对内存的访问对 GPU 性能的发挥有极大的影响[5]。因此，GPU 程序的优化主要解决存储器访问和线程配置问题。

(1)存储器访问优化

流处理器能够存取的存储器有 6 种存储器，但由于 Global Memory 的读写数据非常耗时，而 Local Memory 和 Registers 的空间太小[5]。因此，表 1 中采用余下 3 种存储组合方式得到的测试结果，选取其中最优的尽可能以最优方式组织存储器访问。测试采用 1 000 个 pattern 匹配 20 MB 文本数据，可见存储方式 B 的计算时间较少。

表 1　不同 Memory 测试结果

存储方式	Texture Memory	Constant Memory	Shared Memory	匹配时间/总时间
A	—	pattern	文本	1.52/2.98
B	Pattern	—	文本	1.39/2.79
C	文本	pattern	—	1.46/2.95

(2)线程配置优化

线程配置是指在 kernel 函数运行时的有关函数配置的部分参数，包括：每个 block 中 threads 的数目，每个 block 使用 shared memory 的大小等。通常的线程配置能够决定程序执行的颗粒度的大小，有时会严重影响程序的性能，因此需多次测试找到当前实验环境最适合程序运行的线程配置。

不同线程配置测试结果由于限于篇幅，选取部分数据见表 2。分析表 2 可知，每个 Block 中的线程数越多，所用时间越少；随着页面大小的减少，执行匹配所用的时间都在减少。

表 2　不同线程配置测试结果

页面大小/KB	每个 SM 的 Block 数目	每个 Block 中的线程数	GPU 的执行时间/s
2	128	256	4.07
2	128	512	2.49
2	64	512	2.47
1	64	512	1.39

因此，对于本实验测试环境，选取的最佳的页面大小为 1 KB，每个流多理器的 Block 数目为 64，每个 Block 的 thread 数目为 512。

5　实验结果及分析

实验平台如下：

CPU 是 Intel(R) Core(TM) Dual CPU E7500 2.93 GHz，内存为 2 GB。GPU 是 GeForce GTX260，显存频率为 625 MHz，显存大小为 896 MB，总线接口为 PCI-E 2.0，拥有 192 个 SP 分别在 24 个 SM 上。操作系统是 Windows XP2，软件平台为 CUDA 2.3，实验集成环境 Visual Studio 2005。

实验数据如下：

实验选取大小为 20 MB 的文本作为待处理的比较文本，选取 4 种不同个数但长度均为 4 个字符的不同个数关键词集合：100 个关键词，500 个关键词，1 000 个关键词，2 000 个关键词

CPU 下采用 C 实现 M-BNDM 算法的程序与之比较，实验结果如图 7 所示。

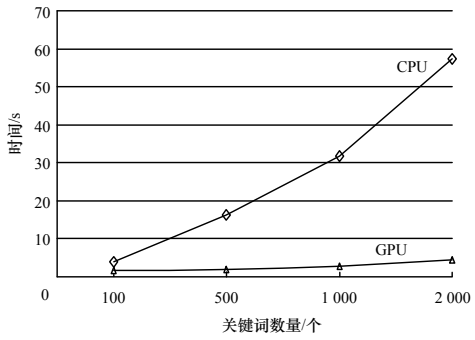

图 7　CPU 和 GPU 比较的实验结果

从上述实验结果分析可以得到：

(1)随着关键词的增多，GPU 和 CPU 的加速比逐渐提高，加速比趋向于约 13 倍，最初加速比比较小是因为操作匹配运算量较小，然后当运算量变大时，GPU 体现出相当大的优势；而且当关键词为 1 000 时 CPU 使用率在 50%左右，因此可以推断即使 CPU 的程序采用了多线程的技术也无法大幅度提升速度，因为 CPU 使用率已经比较高。

(2)随着模式集合的增大，花费在 GPU 和 CPU 间传输数据及预处理等操作所占的时间比例逐渐变小，比较适用于数据量较大的环境，但是同时也要考虑显存有限的局限性。

(3)由于当模式为小集合时，花费在 GPU 和 CPU 间传输数据及预处理等操作所占的时间占总时间较大，算法不适用应用于小模式集合的匹配，若应用在实时的入侵检测系统时实时响应性性能较弱。

(下转第 273 页)