# 一种并行中英文混合多模式匹配算法

王 震，李仁发，李彦彪，田 峥

（湖南大学 嵌入式与网络计算湖南省重点实验室，湖南 长沙 410082）

**摘 要：**相比于一般模式匹配来说，中英文混合环境下的多模式匹配具有其自身独特的难点。文章基于经典的线索化完全哈希 Trie 树算法，提出了一种并行化的中英混合多模式匹配算法。该算法通过拆分文本来降低多模式匹配算法的串行度，进而在拆分出的小文本上进行并行匹配。并通过并行化预处理过程，设计新存储结构等方法进行优化。实验结果表明，此算法的效率高于与经典算法，当数据规模达到 $2^{26}$ 时，甚至可获得 8 倍以上的加速比。

**关键词：**多模式匹配；中英文混合；图形处理单元；并行计算；统一计算设备架构

**中图法分类号：**TP393.08　　**文献标识码：**A

## A Parallel Multiple Pattern Matching on Chinese/English Mixed Texts Algorithm

WANG Zhen, LI Renfa, LI Yanbiao, TIAN Zheng

(*Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan University, Changsha Hunan 410082, China*)

**Abstract:** Compared to the general pattern matching processing, multiple pattern matching on Chinese/English mixed environment owned its specific difficulties. Concerning the classical Threaded Hash Trie algorithm, a parallel multiple pattern matching on Chinese/English mixed texts algorithm is proposed. The program splits the text into a number of small texts, and runs THT algorithm to match them. Further accelerates by parallelization of pretreatment process and new storage structure. Experimental results indicate that the method is more effective than classical algorithm, and can get more than 8 times speedup ratio when the data scale reaches $2^{26}$.

**Key words:** multiple pattern matching; Chinese/English mixed; graphics processing unit (GPU); parallel computing; computer unified device architecture (CUDA)

## 1 引言

多模式匹配在网络审计、信息安全、生物计算等诸多领域内均有着广泛而重要的应用。在我国现有的网络环境下，多模式匹配将会面临中英文混合处理这一特殊难题。通常情况下，英文字符皆为单字节字符，但中文字符的字节长度却根据编码方式的不同而不同，可能为双字节、三字节或者四字节。若将传统的模式匹配算法直接用于中英文混合环境进行处理，则会产生空间膨胀、误匹配或漏匹配等问题。

由于中英文混合这一特殊环境仅出现于中文体系中，在单字节编码的英文体系中并不存在，故而国外的相关研究较少。针对这些问题，一些国内学者做出了较为有益的研究和探索。但大多数文献提出的算法在中英文混合环境下都存在匹配准确性以及匹配效率方面的问题。某些文献采用了组合状态解决空间膨胀问题，并尝试使用 QS 匹配的算法思想进行加速。算法很好地解决了空间膨胀问题，但仅适用于中文环境，而在中英混合的环境下会存在字节错位问题。另一些文献则通过添加“标记”以解决字节错位问题，但需要对待匹配文本串进行预扫描，加大了额外的开销。而文献[1]提出的基于线索化完全哈希 Trie 结构的多模式匹配算法（THT），是一个能够在中英文混合环境下进行正确高效的匹配，有着较低的空间与时间复杂度的经典算法。

图形处理单元（Graphics Processing Unit，GPU），原本仅专用于图形图像处理。随着技术的不断发展与进步，GPU 已经可以很好地被用于通用计算之中，并且成为主流计算机系统的一部分[2]。GPU 在通用计算方面的长足发展，极大地促进了各学科领域的技术进步。

早期的 GPU 并行化处理，在代码实现上非常复杂，只能使用给定的图形图像 API 进行开发，并需要严格地考虑内存资源及编程限制。NVIDA 公司为了减少 GPU 并行化过程中的限制，降低开发

**基金项目：**国家自然科学基金：以汽车为例的 CPS 若干问题研究（No.61173036）

**作者简介：**王震，1988 年生，男，硕士，主要研究方向：网络安全，并行计算，GPU 通用计算；李仁发，教授，博导；李彦彪，博士；田峥，博士.

E-mail: Lance.cs.wz@gmail.com

门槛，以吸引更多的开发人员。推出了一种全新的 GPU 计算架构——统一计算设备架构（Compute Unified Device Architecture，CUDA）[3]。CUDA 架构统一了计算机上的 GPU 和 CPU 设备，并使得用户编写的代码在性能上提升了多个数量级。在使用上，开发人员仅需将其视为 C 语言的一种很小的扩展，即可很好地进行开发工作。

文献[4]已经使用 CUDA 架构对英文多模式匹配做出了有益的研究，但若将其算法直接用于中英文混合的多模式匹配的并行化加速上，依然会存在字节错位等问题。

本文将在经典的 THT 算法上，使用 CUDA 架构，提出一种并行化加速的中英混合多模式匹配算法。并根据算法本身的一些特点，而进一步给出相应的优化。

## 2 中英混合多模式匹配算法

基于线索化的完全哈希 Trie 树算法是一种高效的多模式匹配算法。扩展常规的 Trie 结构，将 Trie 树的所有结点均设为 256 大小完全哈希表，以模式串字符的内码为键值构造完全哈希 Trie 树。并在此基础上，进行线索化过程（图 1）。即在每一次匹配失败后，使用已获得的部分匹配结果来进行下一步匹配，从而来避免匹配指针的回溯。线索化结构建立之后，仅需从头至尾将待匹配的文本依次输入，在 Trie 树结构上进行匹配与跳转。每达到 Trie 树上的一个终结节点即为找到一个匹配，记录所有到达过的终结节点，即可得到最终的匹配结果。

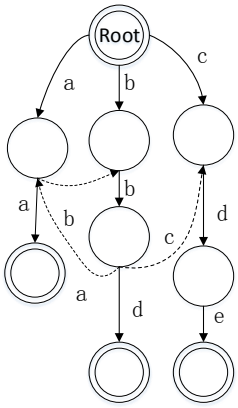


图 1 线索化

## 3 并行中英文混合多模式匹配算法

由于模式串通常是固定不变的，构造线索化完全哈希 Trie 匹配机的过程仅需进行一次。而文本匹配过程是需要多次进行的，且数据量较大。故而，本文将针对文本匹配过程进行并行化加速。

由于单个文本匹配受前后文内容影响，匹配的并行度很低。故可考虑多个文本同时进行匹配，或者将一个大文本分拆成若干小文本段，分别进行匹配，再将匹配间隔处进行匹配。

### 3.1 多本文并行算法

一般而言，当文本数量很多，而单个文本长度较小的时候，可以将所有文本同时并行处理。仅需将所有待处理的数据并发地读入，然后使用每个 GPU 线程各处理一个文本即可。但此方法在文本数量很少，但单个文本数据规模很大的情形下没有适用性。

### 3.2 基于文本拆分的并行算法设计

对于文本数量少，规模大的情形，则可将待匹配的文本拆分为多个规模较小的子文本，每个线程完成其中一个子文本的匹配。通常情况下，待匹配的网络文本规模都能使得 GPU 达到满载，可充分发挥出 GPU 高度并行化的优势。但同时需要保证拆分后的每个子文本规模相同或相当，且特殊处理文本拆分处的匹配以避免误匹配。

#### 3.2.1 文本拆分

设整个待处理的文本总长度为 $Len_{Text}$，线程数为 $N_T$ 时 GPU 处理效率达到最优，此时线程块总体数量为 $N_B$。将文本拆分为 $N_T \times N_B$ 段，则每段子文本的平均长度 $Len_{TextP} = Len_{Text} / (N_T \times N_B)$（图 2）。以固定长度拆分文本时，可能将一个汉字拆分到两个不同的子文本中，在两个连续子文本的末尾和开始均形成一个汉字片段。在此情形下进行匹配，则会产生一连串的错匹配/误匹配等。故需要将子文本末尾处的汉字片段补全，且将起始处的汉字片段移去，以消除每段子文本上的汉字片段。

例如文本“GPU 并行技术”，其对应的汉字编码为“47|50|55|B2 A2|D0 D0|BC BC|CA F5”，若从第四、八字节处拆分，则会形成错误的子文本 $P_0$=“47|50|55|B2”、$P_1$=“A2|D0 D0|BC”以及 $P_2$=“BC|CA F5”。需要在子文本 $P_0$ 后补上汉字片段“A2”，在 $P_1$ 末尾处补上“BC”，同时将子文本 $P_1$ 上的字节“A2”及 $P_2$ 上的“BC”移除。即拆分为 $P_0$=“47|50|55|B2 A2”、$P_1$=“D0 D0|BC BC”与 $P_2$=“CA F5”（图 3）。由此而形成的任意两子文本的大小相差不会超过两字节，且实现时只需记录每个子文本的在整个文本中的起始与结束位置即可，无须开辟新的内存以及内存拷贝。

#### 3.2.2 并行匹配拆分后的子文本

对于拆分后的 $N_T \times N_B$ 块小文本，运用 CUDA 技术进行并行加速。为拆分后的子文本分配 $N_T \times N_B$

个 CUDA 线程进行并行化处理，一个 CUDA 线程仅用于匹配单独的一个子文本。

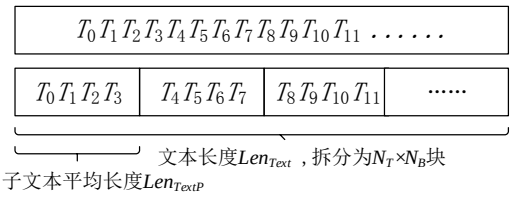


图 2 文本拆分

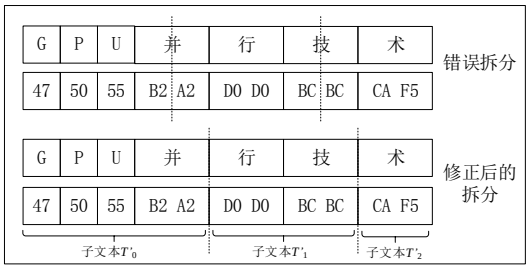


图 3 边界的拆分

### 3.2.3 文本拆分处的处理

对求得分割部分时，先求得最长模式串的长度 $Len_{MaxP}$，对相邻两子文本 $T'_i$，$T'_{i+1}$，提取出子文本 $T'_i$ 的最后($Len_{MaxP}$-1)个字节，提取出子文本 $T'_{i+1}$ 的起始（$Len_{MaxP}$-1）个字节，拼合而成新的文本 $PT_i$。注意，若新文本 $PT_i$ 起始或末尾字节为一汉字片段，则均需要移除。假设此且假设模式串最长有 $Len_{MaxP}$ 字符，则最多仅需在每个分割处多处理($Len_{MaxP}$-1)×2 个字符。在全为双字节汉字的情况下最多 ($Len_{MaxP}$-1)×2×2×($N_T \times N_B$) 字节数据。其规模较小，在参数固定的情况下为一常数，不随待匹配文本规模的增长而增大。从拆分处所形成的小文本规模较小（不足 2×$Len_{MaxP}$），无法充分利用 GPU 计算资源反会使其效率变低，故选择此步骤在 CPU 上完成。

但由于无法判断一个值大于 128 的字节是一个汉字的开始或末尾节点，故算法在实现上无法仅仅只读取边界字节来进行。须要从头扫描一遍文本，判断出每段子文本的边界是否为汉字片段，并同时求出相邻子文本拆分处所形成的字符串。

## 4 并行算法的优化

### 4.1 预处理的并行设计

由 2.2.3 节的分析可知，对于含有双字节 BGK 码汉子的文本，无法在不做预处理的情况下进行拆分。且预处理过程具有较大的数据相关性，不能直接进行并行优化。THT 算法是一种无需预扫描的高效文本匹配算法，若无法对额外预处理阶段进行优化，势必会影响程序的效率。

假设整文本存储于字节数组 $T$ 中，某段子文本为 $T[n_0n_1...n_m]$。若已知 $T[n_0]$是否为一汉字末尾字节，那么可由前向后依次判断，最终推出 $T[n_m]$是否为一汉字末尾字节。反之若 $T[n_0]$不为一汉字的末尾字节，亦可由前向后推出 $T[n_m]$是否为末尾字节。

对拆分后的每段文本，分别假设其首字节为汉字末尾字节以及非汉字末尾字节，计算出此段文本最后字节的特性并存储。此步可使用 GPU 并行加速。

对计算后的结果，由前至后依次处理，求得所有子文本末尾字节的特性。如图 4，第 $i$ 段文本 $T'_i$ 的末尾字节被确定，那么下段文本 $T'_{(i+1)}$的起始字节的特性也随之确定。由此便可求得文本段 $T'_{(i+1)}$ 末尾字节的特性。

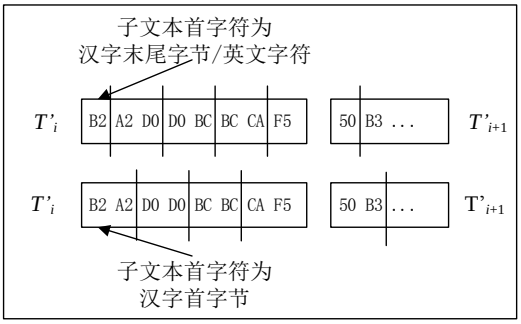


图 4 相邻子文本间首位字节的判断

本节讨论了如何并行加速预处理过程，这一方法对需要预处理的中英混合多模式匹配算法有着很好的借鉴意义。

### 4.2 数据结构的存储优化

由于需要将 Trie 树整体结构由 CPU 内存拷贝至 GPU 显存中，直接地数据拷贝势必造成大量指针的指向发生错误。故在这里设计出一种新的 Trie 树结构存储方法：预分配足够的内存数组，数组的每一个元素可存储一个节点，使用数组下标替代指针，顺序地从前往后将数组内容指定给每一个新建立的节点。即使用数组来模拟系统的内存分配，由于在整体拷贝后数组下标及节点的存储位置都不发生变化，从而保证指针的指向的正确性。这样一来，即使只是简单的内容拷贝函数 cudaMemcpy()，也可以将 Trie 树的结构拷贝至 GPU 显存中且不发生错误。

若不使用此存储结构，亦可直接在 GPU 设备上建立起 Trie 树结构。但由于 Trie 树的构建过程中节点间的相关性很大且无法提前预测，以致算法串行度过大而无法并行处理，故而会降低资源的利用率，使得其过程效率很低。

## 5 实验与结果

文本所采用的硬件平台为 Dell Power Edge T620 工作站，CPU 型号为 Intel Xeon(R) E5-2630 2.3GHz，内存大小为 4GB。GPU 型号为 NVIDIA Tesla C2075，6GB 显存，计算能力为 2.0。

软件平台为 Windows Server 2008 操作系统，Microsoft Visual C++ 2010 Express 环境下进行实现。

实验使用随机程序生成中英文字符，所有字符皆在可见的中英字符中选取。由于实际情况中，单个模式串的长度很小，通常仅为 3-7 字符，故在实验中，假设最长模式串长度为 8 字符。

使用程序随机生成 10 万个模式串，模式串长度不超过 8（中/英）字符，且不低于 1 字符，其长度也是随机决定。为了便于比较分析，控制其中 20%为纯英文模式串，20%为纯中文模式串，剩余 60%为中英混合模式串。同时对生成的中英混合模式串进行限制，设定 20%为英文字符，80%为中文字符。随机生成 9 组待匹配的文本数据，同样地，设定每组文本数据中 20%为英文字符，80%为中文字符。随机生成的 9 组文本大小从 1M 到 256M，相邻两组文本以 2 的倍数递增。实验结果如表 1 所示。

表 1 实验结果

| 数据规模（*字符*） | 经典算法时耗（*ms*） | 并行算法时耗（*ms*） | 加速比 |
|---|---|---|---|
| $2^{20}$ | 102.282 | 42.623 | 2.400 |
| $2^{21}$ | 191.280 | 58.254 | 3.284 |
| $2^{22}$ | 382.423 | 91.090 | 4.198 |
| $2^{23}$ | 765.032 | 154.712 | 4.945 |
| $2^{24}$ | 1497.20 | 280.762 | 5.333 |
| $2^{25}$ | 2985.36 | 492.534 | 6.061 |
| $2^{26}$ | 6604.53 | 755.316 | 8.744 |
| $2^{27}$ | 11926.6 | 1451.190 | 8.218 |
| $2^{28}$ | 23466.6 | 2888.190 | 8.125 |

如图 5 中所示，无论 GPU 上运行还是在 CPU 上运行，算法的时间消耗都随着文本规而模线性地增长。由图 6 可知，在文本规模较小时，算法效率随着文本规模的增大而增大。当文本规模达到一定程度后，加速比趋于稳定，且有下降的势头。这是由于规模较小时，GPU 资源利用不充分，没显示出其在大规模数据上的优势。而当数据足够大的时候，效率充分发挥，故而加速效果趋于稳定，且因实际数据的差异而有小幅波动。

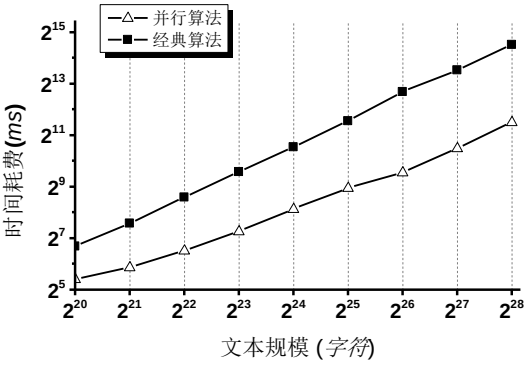


图 5 时耗对比图

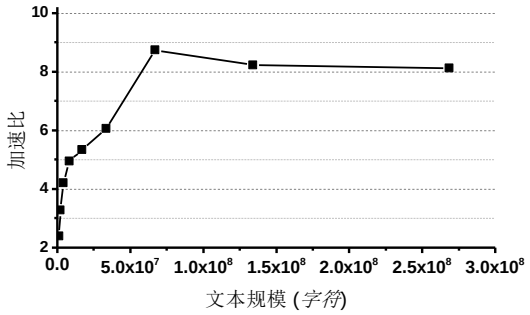


图 6 加速比

## 6 总结

本文在 CUDA 架构下，提出了一种基于经典 THT 算法的中英文混合多模式匹配算法。并进一步给出了算法的优化方案。最后，通过实验对此算法的性能进行评估并与经典算法进行对比，结果表明在理想情况下可获得 8 倍的加速比。

**参考文献：**

[1] 孙钦东，黄新波，王倩. 面向中英文混合环境的多模式匹配算法[J]. 软件学报，2008, 19(3): 674-686.

[2] JOHN D OWENS, MIKE HOUSTON, DAVID LUEBKE, et al.GPU Computing: Graphics Processing Units -powerful, programmable, and highly parallel-are increasingly targeting general-purpose computing applications [J]. Proceedings of the IEEE, 2008, 96(5): 879-899.

[3] NVIDIA Corp. CUDA C Programming Guide [Z]. [S.l.]: NVIDIA Corp: 2012.

[4] 张光斌，谢维盛，吴鸿伟. 基于CUDA的多模式匹配技术[J]. 信息网络安全, 2011, 9: 126-128.