一种适合中文的多模式匹配算法

侯整风 杨 波 朱晓玲
(合肥工业大学计算机与信息学院 合肥 230009)

摘 要 中文字符的相互独立性导致AC算法的时空性能急剧下降。针对此问题,对AC算法的存储结构进行了改进,提出了一种适合中文的多模式匹配算法——AC_SC算法。该算法以邻接链表存储有限状态自动机,尝试解决存储空间快速膨胀问题,并将状态"0"的长链表转化为散列链表,以提高算法的匹配效率。实验结果表明,AC_SC算法具有良好的时空性能。

关键词 多模式匹配,AC算法,邻接链表,有限状态自动机

中图法分类号 TP393.08 **文献标识码** A

Multiple Pattern Algorithm for Chinese

HOU Zheng-feng YANG Bo ZHU Xiao-ling
(School of Computer and Information, Hefei University of Technology, Hefei 230009, China)

Abstract Because of the independent of the Chinese characters, the space and time performances of AC algorithm decline sharply. For this problem, the storage structure of AC algorithm was improved and a multi-pattern algorithm for Chinese named AC_SC was proposed. The algorithm uses adjacency-list to store the finite state automata to solve the problem of the storage space rapid expansion. Besides, the long linked list of the state '0' is changed into a Hash linked table to improve the matching efficient. Experimental results show that AC_SC has better time and space performances.

Keywords Multi-pattern matching, AC algorithm, Adjacency-list, Finite state automata

1 引言

模式匹配广泛应用于入侵检测[1]、内容过滤及DNA序列匹配[2]等。1977年,Knuth、Morris和Pratt提出了能够消除回溯的单模式匹配算法——KMP算法[3],该算法利用"部分匹配"的结果,将模式串向右移动若干位置后继续与文本串当前字符进行匹配。同年,Boyer和Moore提出了一种更高效的模式匹配算法——BM算法[4],该算法运用坏字符规则(Bad Char)和好后缀规则(Good Suffix)来计算模式串右移距离,实现了模式串的跳跃式匹配,进一步提高了匹配效率。随后人们对模式匹配算法进行了广泛深入的研究,相继提出了BMH[5]及QS[6]等单模式匹配算法。1975年,Alfred V. Aho和Margaret J. Corasick提出了一种多模式匹配算法——AC算法[7],该算法基于有限状态自动机,利用goto函数、failure函数和output函数,扫描一遍文本串可匹配所有模式串。Commentz-Walter对AC算法进行改进,提出了AC_BM算法[8],该算法在AC算法的基础上,结合BM算法的思想,实现了模式树的跳跃式匹配,因而具有更高的匹配效率。Wu Sun和Manber提出了WM算法[9],其结合BM算法字符跳转的思想和Hash散列的方法,利用前缀表获得了较高的匹配效率。

AC算法基于有限状态自动机,由于中文字符的相互独立性,随着中文模式串数目的增加,有限状态自动机所需存储空间快速膨胀,导致goto函数计算量大,Cache命中率下降,难以处理大数量级的模式串。王永成等[10]提出用完全Hash表存储组合状态自动机,存储空间快速膨胀问题有所改善,但对于大规模中文模式串匹配,存储空间仍非常大。Norton等[11]提出了状态矩阵的两种稀疏存储方式:带状行方式和稀疏行方式,这两种存储方式在一定程度上减少了有限状态自动机的存储空间,然而对中文模式串的压缩率均不够高,且会增加goto函数的计算时间。Tuck[12]和巫喜红[13]尝试用位图压缩来减少有限状态自动机的存储空间,但对中文模式串的压缩率约为25%。张元竞等[14]提出的AC-bitmap算法,将访问频率较低的状态对应的转移表压缩存储,但该算法同样存在对中文模式串压缩率不高的问题。王培凤等[15]提出了CA-AC算法,该算法尝试通过减少自动机状态数目来减少存储空间,但由于中文模式串含相同前缀的中文词组较少,状态数目难以减少,空间开销仍很大。

本文对有限状态自动机存储结构进行研究,提出了一种适合中文的多模式匹配算法——AC_SC(AC Suit Chinese)。该算法采用邻接链表存储有限状态自动机,较好地解决了有限状态自动机存储空间快速膨胀问题。此外,将状态0的链

到稿日期:2013-01-22 返修日期:2013-05-24 本文受安徽省自然科学基金(090412051),广东省教育部产学研结合项目(2008B0905002400)资助。

侯整风(1958—),教授,主要研究方向为计算机网络、信息安全,E-mail:houzf@hfut.edu.cn;杨 波(1986—),硕士生,主要研究方向为计算机网络、信息安全;朱晓玲(1974—),讲师,主要研究方向为信息安全。

表转化为散列链表，以提高算法的时间性能。

2 AC 算法

AC 算法包括两个阶段：预处理阶段和匹配阶段。

2.1 预处理

预处理需构造有限状态自动机并计算 3 个函数：goto(转移函数)、failure(失效函数)和 output(输出函数)。

(1)构造有限状态自动机

设模式串集合 $K=\{y_1,y_2,\cdots,y_i,\cdots,y_n\}$。$K$ 中的每个模式串 y_i 的初始状态为 0。从状态 0 出发，逐个取出 y_i 中的每一个字符，如果从当前状态出发能够找到标注该字符的矢线，则将矢线所指的状态作为当前状态；否则，添加一个新状态，新状态为已有的最大状态加 1，并将新加入的状态作为当前状态，继续对 y_i 中的下一个字符进行处理。当模式串集合 K 中的所有模式串都处理完毕，再添加一条从状态 0 到状态 0 的自反线，表示保持状态 0 的字符集。

例如，模式串集合 $K=\{he,she,his,hers\}$，有限状态自动机如图 1 所示。

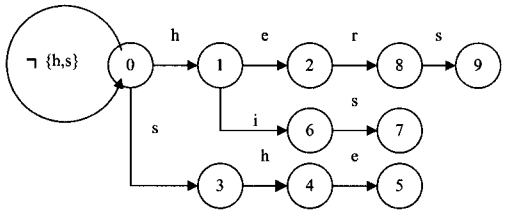

图 1 有限状态自动机

(2)goto 函数

设状态集合 $U=\{0,1,2,\cdots\}$，文本串当前字符为“a”，则 goto 函数为一映射：$goto(U,a)->U$。计算过程：如果从当前状态出发能够找到标注字符“a”的矢线，则 goto 函数返回矢线所指的状态，否则返回跳转失败状态 fail。

如图 1 所示的有限状态自动机，$goto(1,e)=2$，$goto(1,i)=6$，$goto(1,b)=fail$。

(3)failure 函数

失效函数 failure 用来计算某个字符失配时，应转移的状态。计算过程：

①若某状态的父状态为 0，该状态的失效函数为 0。

②对于其它状态 m，设其父状态为 r，标注的字符为“a”，则该状态的失效函数 $failure(m)=goto(failure(r),a)$。

如图 1 所示的有限状态自动机，$failure(1)=0$，$failure(6)=goto(0,i)=0$，$failure(4)=goto(0,h)=1$。

(4)output 函数

输出函数 output 用来输出已匹配的模式串，计算过程分为两步：

①构造有限状态自动机的过程中，每处理完一个模式串，就将该模式串加入到当前状态 s 的输出函数中。

②计算失效函数过程中，若 $failure(s)=s'$，则 $output(s)=output(s)\cup output(s')$。

如图 1 所示的有限状态自动机，$output(7)=\{his\}$，$output(9)=\{she,he\}$。

2.2 匹配过程

①初始时，当前状态 s 为 0，文本串指针指向文本串的首字符。

②若文本串的当前指针不为空，则取出所指的字符“a”；否则匹配过程结束。

③计算 $s'=goto(s,a)$。若 $s'=fail$，则转④；否则 $s=s'$，向右移动文本串指针 1 位；如果 $output(s)=NULL$，转②；如果 $output(s)\neq NULL$，则输出 $output(s)$ 的值，表示有模式串匹配成功，转②。

④调用 failure 函数，即当前状态 $s=failure(a)$，转③。

3 有限状态自动机的常用存储方式及其缺陷

3.1 完全 Hash 表方式

完全 Hash 表存储方式，为每一个状态 s 建立一个完全 Hash 表 h_s，h_s 的值表示跳转的状态。

图 1 所示的有限状态自动机，其完全 Hash 表存储方式如图 2 所示。

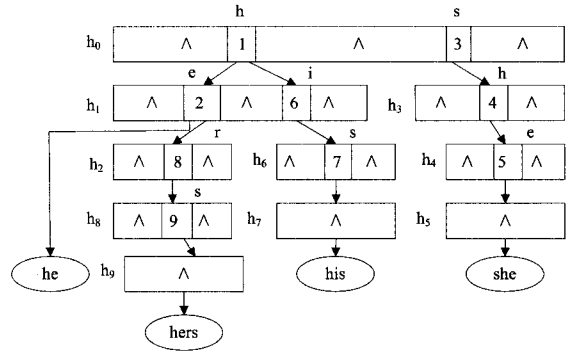

图 2 完全 Hash 表存储方式

该存储方式所需的存储空间为 $statenum * maxcharset$，其中 $statenum$ 为有限状态自动机的状态数目，$maxcharset$ 为字符集的最大字符数。对于英文模式串，$maxcharset$ 为 256，而对于中文模式串，$maxcharset$ 为 $256*256(65536)$，随着模式串数目的增大，状态数增大，存储空间快速膨胀，巨大的存储空间也会导致算法时间效率相应下降。

针对这一问题，文献[10]提出用完全 Hash 表存储组合状态自动机。组合状态自动机将状态分为两类：一类为虚状态，负责匹配中文字符的高字节部分；另一类为实状态，负责匹配中文字符的低字节部分。这种基于组合状态机的完全 Hash 表存储方式所需存储空间为 $statenum*256*2$，在一定程度上缓解了存储空间快速膨胀问题。由于中文模式串汉字之间的相互独立性，$statenum$ 随着模式串的增长而近似线性增长；对于大规模中文模式匹配，组合状态机的空间开销依然巨大。

3.2 状态矩阵方式

状态矩阵方式为每一个状态 s 建立一个数组 A_s，A_s 值表示跳转的下一状态。所有状态的数组组成二维矩阵 $M[0\cdots s][0\cdots j]$，其中 $0\cdots s$ 表示状态，$0\cdots j$ 表示模式串中不重复的字符。

图 1 所示的有限状态自动机的状态矩阵存储方式如表 1 所列。

该存储方式所需的存储空间为 $statenum * charnum$，其中 $charnum$ 为模式串集中不重复字符的个数。对于英文模式串，$charnum$ 最大为 256，所需的存储空间同完全 Hash 表方式；一般情况下，$charnum$ 远小于 256(共 94 个，52 个大小写字母和 42 个特殊字符)，存储空间开销不大。对于中文模式

串，随着模式串的增大，由于汉字之间的相互独立性，*statenum* 近似呈线性增大，*charnum* 也相应线性增大，导致所需存储空间迅速增加。

表 1　状态矩阵

状态 \ 字符	e	h	i	r	s
0	0	1	0	0	3
1	2	0	0	0	0
2	0	0	0	8	0
3	0	4	0	0	0
4	5	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	7
7	0	0	0	0	0
8	0	0	0	0	9
9	0	0	0	0	0

文献[11]提出了两种稀疏存储方式：带状行方式和稀疏行方式，其中带状行方式存储矩阵中第一个非 0 值到最后一个非 0 值之间的全部元素，而稀疏行方式则只存储矩阵中的非 0 值元素。对于中文模式串，一个汉字需拆分成两个字节存储，状态矩阵不再是稀疏矩阵，压缩率不高，且会增加 goto 函数的计算量。因此，文献[11]提出的存储方式不适合中文模式匹配。而文献[12,13]提出的位图存储方式，同样存在压缩率不高的问题。

4　AC_SC 算法

针对完全 Hash 表和状态矩阵方式的存储空间快速膨胀问题，本文提出以邻接链表方式存储有限状态自动机，并将扇出系数最大的状态 0 的单链表转化为散列链表，提高了 goto 函数的计算速度。在此基础上，设计了一种适合中文的多模式匹配算法——AC_SC 算法。

4.1　邻接链表存储方式

为每一状态 s 建立一个结点，结点由字符域、状态域和链域组成，其中字符域存储从状态 s 出发跳转到其它状态时所经过的字符，状态域存储从状态 s 出发跳转的下一状态，链域存储另一结点的地址。此外，设置一个顶点表，记录各个单链表的表头地址。

图 1 所示的有限状态自动机的邻接链表存储方式如图 3 所示。

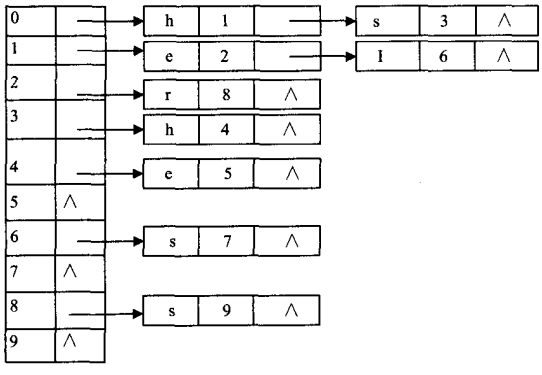

图 3　邻接链表存储方式

对于当前状态 s，计算 goto 函数需搜索对应的单链表 Vs。由于中文模式串字符之间的独立性较高，导致状态 0 的扇出系数较大，而其它状态的扇出系数较小。极端情况下，每个模式串的首字符都不相同，则状态 0 的扇出系数为模式串的数目。当模式串的数目较大时，计算 goto 函数的时间开销较大。

因此，对于中文模式集，本文将状态 0 的单链表 V_0 转化为散列链表 H_0，即将状态 0 的长链表转化为多个短链表。设立一个散列表 H，大小为 256。为状态 0 单链表中每一结点的字符域中的字符计算散列值 n，并将该结点链接到 $H[n]$。计算 goto 函数时，若当前状态为 0，则计算输入字符的散列值，根据该散列值搜索散列链表中相应的链表；否则直接搜索对应状态的单链表。该方法增加了大小为 256 的存储空间，但可将长链表转化成多个短链表，增大了 goto 函数的计算速度，提高了匹配效率。

例如，模式串集＝{升职，时尚白领，中国，外企，生存}，适合中文的邻接链表存储方式如图 4 所示。

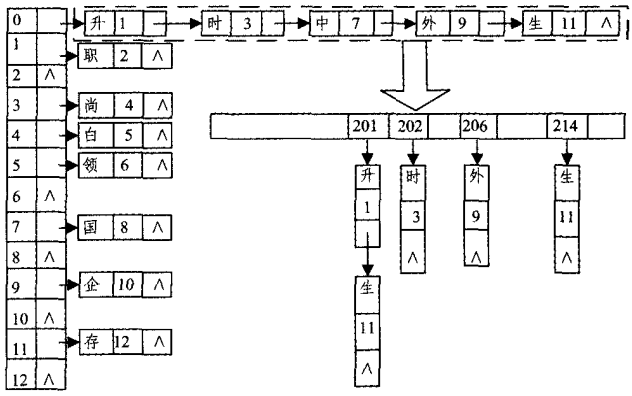

图 4　适合中文的邻接链表存储方式

4.2　AC_SC 算法描述

(1)预处理阶段

①建立邻接链表。建立顶点表 vertices_table，用于记录单链表的表头地址。从状态 0 出发，广度优先搜索建立的有限状态自动机，为能够跳转的下一状态建立一个结点，并将其链接到 vertices_table[0]。继续对下一状态进行处理，直到所有状态都处理完毕。

②将状态 0 的单链表转换为散列链表。定义一个散列表，以储存字符域中字符的首字节的散列值。为状态 0 链表中每一结点字符的首字节计算散列值，并将该结点链接到散列表中。

③计算 failure 函数。父状态为 0 的状态的 failure 函数为 0。对于其它状态 s，若其父状态为 r，标注字符为“a”，则 failure(s)＝goto(failure(r)，a)。计算 goto 函数时，如果当前状态为 0，则根据标注字符首字节的散列值，搜索散列链表中对应的链表；否则直接搜索 vertices_table[s]对应的单链表。

④计算 output 函数。构造有限状态自动机的过程中，每处理完一个模式串，将该模式串加入到当前状态 s 的输出函数中。若 failure(s)＝s'，则 output(s)＝output(s)∪output(s')。

(2)匹配阶段

①将当前状态 s 初始化为 0，文本串指针指向文本串的首字符。

②若文本串指针不为空，则取出所指的字符“a”；否则匹配过程结束。

③调用 goto 函数，计算 s'＝goto(s，a)。若 s＝0，则计算字符的散列值，根据散列值搜索散列链表中对应的单链表；否则直接搜索状态 s 对应的单链表。

④如果 s'=fail，调用 failure 函数，即当前状态 s=failure(s)。如果 $s\neq 0$，转步骤③；

⑤若 $s'\neq$fail，则 $s=s'$。若 output(s)=NULL，转步骤②；如果 output(s)≠NULL，输出 output(s)的值，表示有模式串匹配成功，转步骤②。

例如，模式串集={升职，时尚白领，中国，外企，生存}，文本串为“杜拉拉升职记体现了都市时尚白领在外企的生存法则”，AC_SC 算法的匹配过程如图 5 所示。

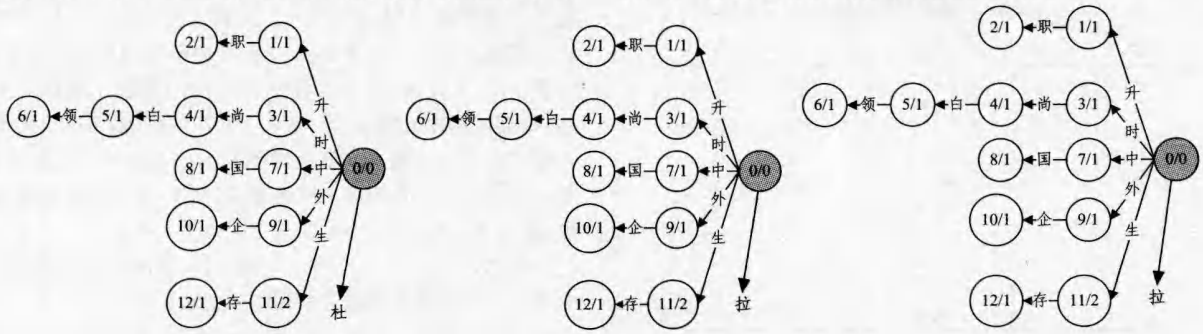

图 5　AC_SC 算法的匹配过程

图 5 中圆代表结点，“/”前面的数字代表状态，后面的代表跳转到该状态搜索链表的次数。当前状态以灰色背景标出。限于篇幅，只列出了前 3 次的匹配过程。从图中可得出，每次仅搜索链表一次即可得到下一状态。

4.3　AC_SC 算法分析

(1)空间复杂度

完全 Hash 表方式，对于中文模式串，每一状态所需的存储空间为 256 * 256(=65536)，算法的空间复杂度为 O(n * 65536)，其中 n 为状态数。

状态矩阵存储方式，算法空间复杂度为 O($n * m$)，其中 m 为模式串中不重复的字符个数。

邻接链表方式，需建立 n 个结点(除顶点表)，每一结点所需存储空间为 3，顶点表所需的存储空间为 n，散列表所需的存储空间为 256，算法的空间复杂度为 O($n+n*3+256$)(=O($4n+256$))，故邻接链表方式的空间性能优于 Hash 表和状态矩阵方式。

(2)时间复杂度分析

算法主要的时间开销为 goto 函数的计算，即根据当前状态和输入字符在有限状态自动机上查找下一状态。

完全 Hash 表和二维数组均具有直接存取的特性，因此，计算 goto 函数的时间复杂度为 O(1)。

邻接链表方式时间复杂度分两种情况讨论。

设模式串集合 $K=\{a_0,a_1,a_2,\cdots,a_n\}$，单链表长度集合 $P=\{p_0,p_1,\cdots,p_n\}$，散列表中单链表的长度集合 $P'=\{m_1,m_2,\cdots,m_{256}\}$，装填因子 $\alpha=\frac{\sum_{i=1}^{256} m_i}{256}$。

①最好的情况，即 $a_0\in a_1\cdots\in a_n$，所有状态的单链表长度都为 1，时间复杂度为 O(1)；

②一般的情况，时间复杂度为 $O(\alpha+\frac{\sum_{i=1}^{n} p_i}{n-1})$。

由于汉字之间的相互独立性，散列链表的装填因子 α 为常数，故时间复杂度略低于完全 Hash 表和状态矩阵。

5　测试结果与分析

测试环境：CPU 为 Intel(R) Pentium(R) 4，2.40GHz，1G 内存，操作系统为 Microsoft Windows XP Professional Service Pack 2，VC++6.0 语言实现。

模式串集为百度过滤关键词集，其中二字词约占 36.8%，三字词约占 9.7%，四字词约占 45.6%，多字词约占 7.9%。测试文本为 5 个不同的纯中文文本(大小约为 12M)。每个文本测试 10 次，取其平均值。

(1)空间性能

如图 6 所示，完全 Hash 表存储方式所需的存储空间较大，且随着模式串数量的增加，所需存储空间近似呈几何增加。当模式串数目大于 1500 时，完全 Hash 表存储方式所需内存超出了系统的所能分配的最大内存，因此没有实验数据(下同)。随着模式串数目的增加，邻接链表方式的存储空间性能明显优于其他 3 种存储方式，适合大规模中文模式匹配。

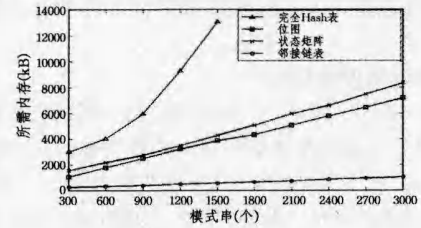

图 6　4 种存储方式的空间性能

(2)时间性能

如图 7 所示，完全 Hash 表存储方式的时间性能最佳，邻接链表方式优于状态矩阵和位图方式，位图方式的时间性能最差。

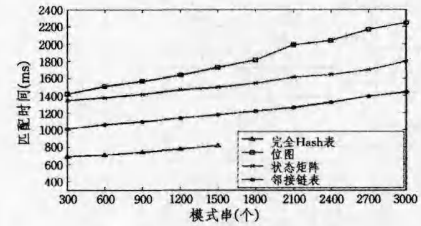

图 7　4 种存储方式的时间性能

当模式串数量超过 1500 时，由于内存消耗过大，完全 Hash 表方式无法运行。尽管前面的理论分析(4.3 节)表明，状态矩阵方式的时间复杂度略低于邻接链表方式，但由于状态矩阵所需存储空间较大，Cache 命中率下降，频繁进行缺页中断操作，导致时间性能下降。因此，实际测试中，状态矩阵

方式的时间性能低于邻接链表方式。

(3)不同算法的时间性能

如图 8 所示，AC_SC 算法的时间性能最优，匹配所需时间大约是 AC-bitmap 算法的 62%，CA-AC 算法的 74%。

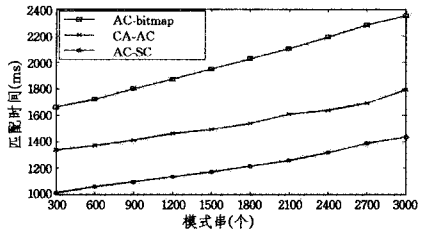

图 8　不同算法的时间性能

结束语　本文提出了一种适合中文的多模式匹配算法。该算法采用邻接链表方式存储有限状态自动机，减少了空间开销，同时将扇出系数较大的状态“0”的单链表转化为散列链表，以便快速查找下一状态，提高了算法的时间性能。最后对算法空间和时间性能进行了测试，测试结果表明，本文提出的邻接链表方式，其时间性能高于状态矩阵存储方式，略低于完全 Hash 表存储方式，而其空间性能远优于完全 Hash 表、状态矩阵和位图方式。AC_SC 算法的时间性能优于 AC-bitmap和 CA-AC 算法，适合大规模中文模式匹配。

参 考 文 献

[1] 杜大军，费敏锐，宋扬，等. 网络控制系统的简要回顾及展望[J]. 仪器仪表学报，2011，3(32)：713-720

[2] Wheeler D L，Barrett T，Benson D A，et al. Database resources of the National Center for Biotechnology Information [J]. Nucleic Acids Research，Database issue，2007，35：5-12

[3] Knuth D E，Morris J H，Pratt V R. Fast Patter Matching in Strings[J]. SIAM Journal of Computer，1977，6(2)：323-350

[4] Boyer R S，Moore J S. A Fast String Searching Algorithm[J]. Communications of the ACM，1977，20(10)：762-772

[5] Horspool R N. Practical fast searching in strings [J]. Software-Practice & Experience，1980，10(6)：501-506

[6] Sunday D M. A very fast substring searching algorithm [J]. Communications of the ACM，1990，33(8)：132-142

[7] Aho A V，Margaret J. Corasick Efficient String Matching[J]. An Aid to Biographic Search Communications of the ACM，1975，18(6)：333-340

[8] Commentz-Walter B. A String Matching Algorithm Fast on the Average [J]. Proceedings of 6th ICALP，1979，71：118-132

[9] Wu Sun，Manber U. Fast algorithm for multi-pattern searching [R]. Tucson：Department of computer science university of Arizona，1994

[10] 王永成，沈州，许一震. 改进的多模式匹配算法[J]. 计算机研究与发展，2002，39(1)：55-60

[11] Norton M. Optimizing pattern matching for intrusion detection [EB/OL]. http://does. idsreseareh. Org/OptimizingPatterm-Matching-ForIDS. pdf，2006-05-11

[12] Tuck N，Sherwood T，Calder B，et al. Deterministic Memory Efficient String Matching Algorithms for Intrusion Detection[C]// IEEE Iafocom. 2004：333-340

[13] 巫喜红，曾锋. AC 多模式匹配算法研究[J]. 计算机工程，2012，38(6)：279-281

[14] 张元竞，张伟哲. 一种基于位图的多模式匹配算法[J]. 哈尔滨工业大学学报，2010，42(2)：277-280

[15] 王培凤，李莉. 一种改进的多模式匹配算法在 Snort 中的应用[J]. 计算机科学，2012，39(2)：72-74

(上接第 88 页)

本文对 MGTP-AS 系统的串行代码比例 f 和并行处理引入的开销比例 f_r 进行测量之后得到了多核优化加速比 S_r。图 7 说明了在系统进行多核优化时，如果能够得到准确的 f 和 f_r，通过计算得到的 S_r 就可以对实际优化效果和理想效果之间的差距进行评估，从而给出系统优化后性能上升的趋势。

结束语　随着网络新型业务的不断出现、原有业务的更新升级，都要求网络流量监测系统软件能具有良好的性能，同时，多核处理器平台为上层应用软件提供了强大计算能力的支撑，本文将网络流量监测与多核处理器结合之后，对网络流量监测系统在多核处理器平台上的性能优化进行了深入研究。总之，多核处理器平台强大的计算能力不仅可以支持网络流量监控软件，还可以使优化后的系统性能得到提高，有效处理网络中的大量数据处理，在保证网络服务质量、发掘网络资源价值、分析网络关键数据、保障网络稳定运行等方面具有良好的应用前景。

参 考 文 献

[1] 刘热. OpenMP 多核技术研究及其在遗传算法中的应用[J]. 沈阳大学学报，2010(05)：21-35

[2] 吴俊杰，潘晓辉，杨学军. 面向非一致 Cache 的智能多跳提升技术[J]. 计算机学报，2009(10)：31-42

[3] 肖俊华，冯子军，章隆兵. 片上多处理器中延迟和容量权衡的 cache 结构[J]. 计算机研究与发展，2009(01)：19-21

[4] 黄国睿，张平，魏广博. 多核处理器的关键技术及其发展趋势[J]. 计算机工程与设计，2009(10)：43-49

[5] 黄国睿，张平，魏广博. 多核处理器的关键技术及其发展趋势[J]. 计算机工程与设计，2009(10)：67-72

[6] Gummaraju J，Rosenblum M. Stream Programming on General-Purpose Processors[M]. 2005：113-127

[7] McCool M D. Scalable Programming Models for Massively Multicore Processors[M]. 2008：89-101

[8] Bell S，Edwards B，Amann J，et al. Tile64processor：A64-core soc with mesh interconnect[C]// Proc of Int Solid-State Conference. 2008：145-152

[9] Ros A，Acacio M E，Garcia J M. Scalable directory organization for tiled cmp architectures[C]// Proc of Int Conf on Computer Design. 2008：94-99

[10] Guz Z，Keidar I，Kolodny A，et al. Nahalal：Cache organization for chip multiprocessors[C]// IEEE Computer Architecture Letters. 2007：54-68