第 39 卷 第 12 期
Vol.39 No.12

计 算 机 工 程
Computer Engineering

2013 年 12 月
December 2013

• 人工智能及识别技术 •　文章编号：1000—3428(2013)12—0157—05　文献标识码：A　中图分类号：TP18

基于非对齐双字节读机制的单模式串匹配算法

张 建[1,2]，范洪博[1,2]，黄青松[1,2]，刘利军[1]

(1. 昆明理工大学信息工程与自动化学院，昆明 650000；2. 云南省计算机技术应用重点实验室，昆明 650000)

摘 要：在线精确单模式匹配问题在几乎所有涉及文本和符号处理的领域中均有广泛应用。SBNDMq 是目前该领域性能最高的算法之一。通过向其引入非对齐双字节读机制，对 SBNDMq 算法进行改进，从而提出 SBNDMq_Shortb 系列算法。该系列算法拥有与 SBNDMq 算法一致的跳跃能力，但核心循环的内存访问次数降低为原来的 50%，算法性能更高。实验结果表明，在大多匹配条件下，SBNDMq_Shortb 系列算法性能优于其他已知算法。

关键词：串匹配；精确单模式；算法设计；位并行；非对齐读；SBNDMq_Shortb 算法

Single Pattern String Matching Algorithm Based on Unaligned 2-byte Reading Mechanism

ZHANG Jian[1,2], FAN Hong-bo[1,2], HUANG Qing-song[1,2], LIU Li-jun[1]

(1. Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650000, China;
2. Yunnan Key Laboratory of Computer Technology Applications, Kunming 650000, China)

【Abstract】Online exact single pattern string matching is widely used in almost all of fields involving text and symbol processing. SBNDMq is one of the fastest algorithms in this field. By introducing the unaligned 2-byte reading, SBNDMq is improved and a serial of algorithm named SBNDMq_Shortb is presented. The jump capability of this algorithm is same with SBNDMq, but the memory access reduces 50%. Thus, this algorithm is faster than SBNDMq. Experimental results show that SBNDMq_Shortb is faster than other well-known algorithms in many matching conditions on the platform.

【Key words】string matching; exact single pattern; design of algorithms; bit-parallel; unaligned reading; SBNDMq_Shortb algorithm
DOI: 10.3969/j.issn.1000-3428.2013.12.034

1 概述

字符串匹配问题是计算机科学的基本问题之一，在网络安全、数据检索和计算生物学等众多重要领域中广泛应用。特别是在集中研究的大规模信息检索和海量医疗信息处理中，串匹配性能的低下对该研究造成了严重的困扰，急需高性能算法出现。

对给定字符集 Σ ($|\Sigma|=\sigma$)，Σ^* 为字符集 Σ 的 Kleene 闭包，给定文本串 $T=t_0t_1\cdots t_{n-1}\in\Sigma^*$ 和模式串 $P=p_0p_1\cdots p_{m-1}\in\Sigma^*$，在线精确单模式串匹配问题是求解集合：$O=\{pos|P[j]=T[pos+j],\ \forall\ 0\leqslant j<m,\ 0\leqslant pos\leqslant n-m\}$ 的问题，同时，T 为在线给出，即在求解该集合过程中，没有针对 T 构建如索引等用于快速定位模式位置的结构。在线精确单模式串匹配问题是其他各类串匹配问题的基础，其他串匹配领域算法大多数由在线精确单模式串匹配算法扩展而来。目前，在线精确单模式串匹配领域得到了广泛而深入的研究，现已经提出数百种算法。在线精确单模式串匹配已成为几乎所有涉及文本和符号处理领域应用的一种基本组件。同时该领域是目前计算机科学各领域中，程序优化程度最高的领域之一，优秀的精确单模式串匹配算法可以作为其他领域中程序优化的范本[1-2]。因此，对在线精确单模式匹配领域的研究有特别重要的意义。

文献[1-2]给出了目前该领域最全面的性能对比，共比较了 85 个算法，涵盖了截至 2010 年 5 月绝大多数著名已知算法。文献指出，一种采用 q-grams 的 BNDM 类位并行算法 SBNDMq[3]，是目前已知算法在性能处理中等长度模式时最高的算法之一。

本文通过向 SBNDMq 算法中引入非对齐双字节读机制，提出 SBNDMq_Shortb 的系列改进算法。SBNDMq_Shortb 的跳跃能力和 SBNDMq 一致，非对齐读机制可以在一次内存访问中同时处理 2 个字符，SBNDMq_Shortb 核心循环的操作比 SBNDMq 更简洁，并且有更少的内存访问

基金项目：云南省科技厅应用基础研究基金资助面上项目(2012FB131)；昆明理工大学人陪基金资助项目(KKSY201203091)；云南省社会发展科技计划基金资助项目(2010CA016)；科技部科技型中小企业技术创新基金资助项目(10C26215305130)

作者简介：张 建(1987－)，男，硕士研究生，主研方向：智能信息系统；范洪博，讲师、博士；黄青松，教授；刘利军，讲师、硕士

收稿日期：2012-10-22　**修回日期：**2012-12-27　**E-mail：**kmustailab@hotmail.com

次数。

2 SBNDMq 算法

现有串匹配算法大多采用滑动窗口机制进行处理，滑动窗口与模式等长并延文本自前向后单向滑动，如果滑动窗口内文本内容与模式每个对应字符均匹配，则报告在当前滑动窗口中出现模式匹配。若滑动窗口移出文本边界，则匹配过程结束。对任意串 $W,X,Y,Z\in\Sigma^*$，若 $W=XYZ$，则称 X 为 W 的前缀，Z 为 W 的后缀，X,Y,Z 均为 W 的子串。根据滑动窗口内计算跳跃距离的方法不同，现有算法可分为按前缀跳跃、按后缀跳跃和按子串跳跃 3 种。SBNDMq 算法属于子串匹配类算法。在子串匹配算法中存在某种结构能够识别模式的所有子串，匹配时窗口中字符自后向前逐个输入算法对应识别结构中。记已输入的窗口的长为 q、后缀为 U，若 U 不是 P 的子串，则窗口可跳过已读入后缀，跳跃距离为 $m-q+1$ 字符。若窗口中所有字符组成的串是模式的子串，因与模式等长的子串唯一且等于模式本身，此时应报告当前窗口中出现模式匹配，之后窗口移动 1 字符。子串匹配类跳跃算法示意图如图 1 所示。

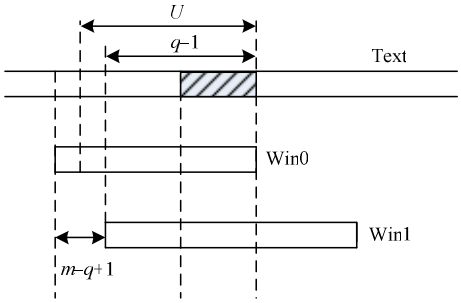

图 1 子串跳跃机制

SBNDMq 使用后向非确定 DAWG 自动机(BNDM[4]自动机)识别模式所有的子串。所有使用此自动机作为判定结构的算法均称为 BNDM 系列算法。

根据 P =“baabbba”构建的 BNDM 自动机如图 2 所示。

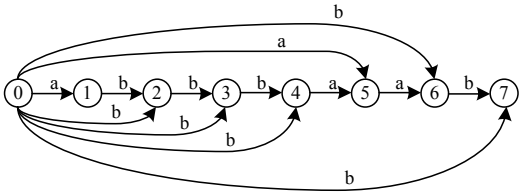

图 2 BNDM 自动机

该自动机所有状态均为终结状态，记 P 的反转为 P^{rv}，该自动机接受的语言等于 P^{rv} 所有子串的集合。因此，若从窗口自后向前输入字符且输入串 U^{rv} 不被自动机接受时，U^{rv} 不是 P^{rv} 的子串，进而可知窗口的长为 $|U|$ 的后缀不是 P 的子串，从而可根据子串跳跃机制进行跳跃。

因传统方法处理 NFA 时要维护活动状态的集合，这对字符串匹配来说开销很大。若模式长度小于机器字长，基于位并行的算法巧妙地利用计算机机内机器字位运算的内在并行性，将自动机的状态与机器字内对应位对应起来，从而实现对 NFA 的模拟，并以 $O(1)$ 的时间复杂度同时更新多个活动状态。称使用位并行机制对 BNDM 自动机进行模拟的算法称为 BNDM 类算法，该位并行方法及其正确性证明见文献[5]。

SBNDMq 算法在上述位并行机制的基础上引入了 q-grams 技术。q-grams 是一种在窗口尾部连续读入 q 个字符后才进行一次分支检查的优化技术，该技术是串匹配领域中一种通用加速技术，目前几乎所有高速算法中均使用了该技术。q-grams 把连续的 q 个字符处理为 Σ^q 上的一个字符，如 3-grams 把串“hello”处理为“hel-ell-llo”。该技术能提高算法的平均跳跃距离和分支预测正确率，提高了算法性能。此外，采用 q-grams 技术后，单个字符对跳跃距离影响降低，因此 q-grams 还能提高算法柔性(flexible)。但 q-grams 会将可跳跃算法的最大跳跃距离降低至 $m-q+1$。通常，较长的模式或较小的字符集需要较大的 q。q-grams 机制可明显提高 BNDM 类算法的性能。

完整的 SBNDMq 算法如代码 1 所示。

代码 1 SBNDMq 算法

F(pos, q)函数为：

$(B[t_{pos}]<<(q-1))\&(B[t_{pos-1}]<<(q-2))\&\cdots\&B[t_{pos-q+1}]$。

```
SBNDMq(p, t, m, n)
//proprecessing
for c ∈ ∑ do B[c]←0; set←1; pos←m–1; for j←0 to m–1 do t_{n+i} ←p_i
for j←m–1 downto 0 do {B[p_j]←B[p_j]|set; set←set <<1;}
//searching
while pos<n do
{while (D←F((pos, q))=0 do pos←pos+m–q+1;
j←pos; while (D←(D<<1) & B[t_{pos–q}])≠0 do pos←pos–1;
pos←pos+m–q; if pos=j then{Report find a matching; pos←pos+1;}}
```

在 SBNDMq 中使用了该领域常用的越界保护技术。该技术在文本尾部接入模式，可保证滑动窗口移出文本边界时退出算法核心循环，因此在算法核心循环中无需进行边界检查，从而该技术可减少分支操作总数，提高算法性能。

同时，SBNDMq 还存在下面的变种：算法最内层循环使用 2-grams 进行跳跃，如果 2-grams 无法实现跳跃，则使用 4-grams 进行跳跃，该变种被记为 SBNDM2_2。

3 SBNDMq_Shortb 系列算法

在 SBNDMq 的核心循环中，有 q 次查表操作。因查表操作可能访问到慢速内存，其平均开销高于位操作等简单

操作。若能降低查表操作次数，将可进一步提高算法性能。

文献[3]提出了一种双字节读加速机制，并将其应用与 SBNDMq 上，从而提出了 SBNDMqb 算法。该机制可一次处理 2 个字节后才进行一次查表操作，降低总操作数。该机制采用空间换时间的方法，在预处理过程中，用一个容量为 $|\Sigma|\times|\Sigma|$ 的一维表 $\boldsymbol{B}2$ 来存储读入 2 个字符后的位向量，即对 $\forall a,b\in\Sigma$，$\boldsymbol{B}2[(a<<8)+b]\leftarrow(\boldsymbol{B}[a]<<1)\,\&\,\boldsymbol{B}[b]$。利用 $\boldsymbol{B}2$ 表，可以通过函数 $Fb(pos,2)=\boldsymbol{B}2[(t_{pos}<<8)+t_{pos-1}]$ 来得到 $q=2$ 时的位向量 $\boldsymbol{D}$。相对于 $F(pos,2)$，$Fb(pos,2)$ 虽然增大了缓存压力，但可以减少操作，在中小字符集上，$Fb(pos,2)$ 的性能高于 $F(pos,2)$，但在大字符集上，$F(pos,2)$ 性能更高。对 $q>2$ 时，若 q 为偶数，仍可以通过上述双字节读机制模拟 $F(pos,q)$。模拟方法的递归定义为：$Fb(pos,q)=(Fb(pos,q-2)<<2)\&Fb(pos-q+2,2)=F(pos,q)$。

文献[6]提出了 SBNDMq 的一种改进算法 FSBNDMqb，在其中引入了一种双字节读的改进机制，进一步降低了内存访问次数，并提高了算法性能。因在 X86 等 big-endian 处理器上，存在性质：

$*(u\text{int}16*)(t+pos-1)=(t_{pos}<<8)+(t_{pos-1})$

因此函数：

$Fshortb(pos,2)=\boldsymbol{B}2[*(u\text{int}16*)(t+pos-1)]=$

$\boldsymbol{B}2[(t_{pos}<<8)+t_{pos-1}]=Fb(pos,2)=F(pos,2)$

该机制因存在不对齐整数边界的读入，被称为非对齐双字节读机制。虽然$*(u\text{int}16*)(t+pos-1)$在读入的 short 型数据跨越整数边界时，读取性能将比对齐读有所下降，但相对于 $Fb(pos,2)$，$Fshortb(pos,2)$可在一次内存访问中同时处理 2 个字节，将核心循环中的内存访问次数降低了一半，并节省了将 2 个 Σ 中字符拼接为 Σ^2 字符的拼接开销。在大多数情况下，非对齐双字节读拥有更高的性能。该文献将这种非对齐双字节读机制应用于 SBNDMq 算法的一种改进算法，在 FSBNDMq 中，从而提出 FSBNDMqb 算法，获得优异的实际性能。实验表明，FSBNDMqb 在许多匹配条件下是目前性能最高算法。

FSBNDMq 在 SBNDMq 基础上引入前向读入机制，如果前向读入字符数为 0，则 FSBNDMq 将退化为 SBNDMq。虽然 FSBNDMq 是 SBNDMq 的改进算法，但 SBNDMq 相对于前向读入字符数为 0 的 FSBNDMq 有更简单的控制逻辑，因此，SBNDMq 性能高于前向读入字符个数为 0 的 FSBNDMq。若向 SBNDMq 算法中引入非对齐双字节读机制，则在某些匹配条件下，将可能得到更高的匹配性能。

现将非对齐双字节读引入 SBNDMq 系列算法，从而得到 SBNDMq 的一种改进算法，称得到的新算法为 SBNDMq_Shortb 系列算法。其代码如代码 2 所示，其中函数 $Fshortb(pos,q)$的递归定义如定义 1 所示。

代码 2 SBNDMq_Shortb 算法

```
SBNDMq_Shortb(p, t, m, n)   //q 为偶数
    //proprecessing
    for c ∈ ∑ do B[c]←0; set←1; pos←m–1; for j←0 to m–1 do t_{n+i←}p_i ;
    for j←m–1 downto 0 do {B[p_j]←B[p_j] | set; set←set<<1;}
    for a, b ∈ ∑ do B2[(a<<8)+b]=(B[a]<<1) & B[b];
    //searching
    while pos<n do
    {while (D←Fshortb(pos, q))=0 do pos←pos+m–q+1;
    j←pos; while (D←(D<<1) & B[t_{pos-q}])≠0 do pos←pos–1;
    pos←pos+m–q; if pos=j then {Report find a matching; pos←pos+1;}}
```

定义 $Fshortb(pos,2)=\boldsymbol{B}2[*(u\text{int}16*)(t+pos-1)]$;

对 $\forall\ q=2k$，$k=2,3,\cdots$，$Fshortb(pos,q)=(Fshortb(pos,q-2)<<2)\ \&\ Fshortb(pos-q+2,2)$。

与 SBNDMq 算法一致，SBNDMq_Shortb 系列算法同样可以进行 2_2 形式的扩展，即核心循环分为内层循环和外层循环，在核心循环的内层循环中读入 2 个字符，在这 2 个字符不足以引起一次跳跃时，进入外层循环，自后向前的继续读入 2 个字符。称进行 2_2 形式扩展的 SBNDMq_Shortb 系列算法为 SBNDM2_2_Shortb，其匹配过程如代码 3 所示，该算法预处理过程和代码 2 预处理过程一致，且函数 Fshortb 和代码 2 中该函数一致。

代码 3 SBNDM2_2_Shortb 算法匹配过程

```
//SBNDM2_2_Shortb searching
while pos<n do
  {D←Fshortb(pos, 2);
while (D←(D<<2) & Fshortb(pos–2, 2))=0 do
{pos←pos+m–3; while (D←Fshortb(pos, 2))=0 do pos←pos+m–1;}
j←pos; while (D←(D<<1) & B[t_{pos-4}])≠0 do pos←pos–1;
pos←pos+m–4; if pos=j then {Report find a matching; pos←pos+1;}}
```

4 实验与数据分析

本文进行了如下对比实验：

实验平台为 Intel® Core2® E3400 @ 3.45 GHz (266×13)/Intel P43/DDR3 1333 RAM (2 GB)/UBUNTU 10.04 LTS 64 位桌面版/g++ 4.4.5/-O3 优化。对每种模式长度，本文实验选择从文本中随机抽取的 100 个互不重叠的文本段作为模式的样本集合，最终结果为去掉匹配速度中最高和最低的 20%后取平均。为避免磁盘影响实验结果，实验前所有文本已读入内存，并在每次匹配之前，顺序读取一个较大的表清空高速缓存以避免各次匹配之间相互影响。实验使用 RDTSC 指令对匹配时间进行计时，精度 ±30 CPU 时钟。测试时 CPU 主频已锁定，并关闭网络和无关后台服务，确保不匹配时处理器利用率维持在 3%以下。

该实验环境类似文献[1-2]对应实验环境，只是依照该领域研究习惯，采用-O3 优化，并开启分支预测(文献[1-2]为-O2 优化，并关闭了自动分支预测，实验测试方法更符合该领域研究习惯)。

文献[1-2]公开了该文献对应实验平台 SMART 11.09[7]，其中给出 80 余种算法的对比实验，包含截至 2010 年 5 月绝大部分著名已知算法。本文实验延续了 SMART 的工作，补充了 SMART 中未收录的 FSO[8]、BXS[9]、QF[9]、Q-Hash_4096[9]、SQ-Hash[10]、SBNDMqb[3]、BSDMq[11]，FSBNDM-wk[12]、SBNDM-wk[12]、FS-wk[12]、TVSBS-wk[12]、FSBNDMq[6]、FSBNDMqb[6]、GSB2[6]、GSB2b[6]、S2BNDM[5]等算法(补充算法过多，无法全部列出)。在本文实验中，所有位并行算法均测试了 32 位版本(记为_i32)和 64 位版本(记为_a64)。所有位并行算法进行了长模式扩展，扩展的方法为用位并行算法搜索模式的前 w 个字符(w 为字长，i32 版本文算法中 $w=32$，a64 版本算法中 $w=64$)，在找到模式前 w 个字符后，用 BF 算法匹配模式其他字符。同时，实验补充了 SMART 中收录算法所未列出的参数(如 SMART 中对 Q-Hash[13]算法只给出了 q=3、5、8 共 3 种情况，实验补充了 q 值为 4、6、7 等情况)。若将参数不同的算法记为不同算法(如将 SBNDM2[3]、SBNDM2b[3]、SBNDM4[3]记为 3 种算法)，实验将本文所提算法和超过 300 种算法进行性能对比，基本涵盖了目前已知算法。

实验测试文本为长度为 20 MB，字符集分别为 2、16、64 的随机文本，随机函数为 G++库函数 rand()以及 DNA 序列 E.coli、英文文本 Bible.txt、自然语言样本 world192.txt，这 3 个文本均来自 SMART。因实验数据量较大，这里只列出每种匹配条件下最快的 4 类算法(如果多个算法只有参数不一致，如 q-grams 中的 q 值，并且这些算法由一篇文献提出，称这些算法为一类算法)的匹配速度(单位为 MB/s)和使其达到最高性能的参数。实验数据见表 1~表 6 所示，其中本文所述算法已加粗显示。

表 1 字符集容量为 2 的随机文本实验数据 (MB·s^{-1})

R2	m=4	m=8	m=16	m=32	m=64	m=128
1	FSO_59_a64 987.1	UFNDM8b_a64[3] 1 379.4	FSBNDMqb_q8f2a64 2 741.3	**SBNDM10_sb_i32** **4 951.9**	SHash8 5 898.6	**SBNDM12_sb_a64** **5 828.5**
2	Shift_Or_i32[14] 790.9	FSO_53_a64 1 141.2	**SBNDM8_sb_a64** **2 637.1**	FSBNDMqb_q8f0i32 4 419.2	Hash8_4096 5 897.4	Hash6 5 420.3
3	Shift_And_i32[14] 615.9	FSBNDMqb_q6f1i32 1 099.8	UFNDM8b_i32 2 336.3	SHash7 3 881.1	Hash8 5 880.7	SHash6 5 395.2
4	DFA 501.6	**SBNDM6_sb_a64** **1 084.6**	SHash6 1 872.9	Hash7 3 786.2	QF_12_1_i32 5 849.7	Hash6_4096 5 389.1

表 2 字符集容量为 16 的随机文本上实验数据 (MB·s^{-1})

R16	m=4	m=8	m=16	m=32	m=64	m=128
1	**SBNDM2_2_sb_i32** **3 347.8**	**SBNDM2_2_sb_i32** **5 340.6**	**SBNDM4_sb_i32** **6 229.3**	**SBNDM4_sb_i32** **6 737.5**	FSBNDMq_q4f1a64 6 915.9	QF_3_6_i32 7 155.7
2	GSB2b_i32 2 736.5	GSB2b_i32 4 894.0	GSB2b_i32 5 747.6	FSBNDMq_q3f0i32 6 425.8	BXS3_a64 6 716.4	BXS3_a64 7 049.4
3	EBOM[15] 2 561.6	EBOM 4 439.5	FSBNDMqb_q4f0i32 5 711.8	UFNDM4_a64 6 261.4	BNDM4_a64 6 654.8	FSBNDMq_q4f1a64 6 912.1
4	SBNDM2b_i32 2 472.0	SBNDM2b_i32 4 422.4	SBNDM2_2_i32 5 551.0	BNDM4_a64 6 164.0	**SBNDM4_sb_i32** **6 654.4**	**SBNDM6_sb_a64** **6 748.9**

表 3 字符集容量为 64 的随机文本实验数据 (MB·s^{-1})

R64	m=4	m=8	m=16	m=32	m=64	m=128
1	TVSBS_w4 2 726.9	**SBNDM2_2_sb_i32** **4 984.2**	**SBNDM2_2_sb_i32** **6 223.5**	SBNDM2_2_a64 6 893.2	QF_2_6_i32 7 063.5	BXS2_a64 10 320.4
2	**SBNDM2_2_sb_i32** **2 690.5**	GSB2b_i32 4 915.9	FSBNDM_w2_a64 6 220.8	FSBNDMqb_q8f0i32 6 883.7	BXS3_a64 6 994.6	QF_2_6_i32 8 905.2
3	GSB2b_i32 2 686.3	SBNDM2_2_i32 4 734.1	BNDM2_a64 6 095.4	BNDM2_i32 6 839.2	FSBNDMq_q4f0a64 6 911.6	EBOM 8 889.4
4	FSBNDMqb_q2f1i32 2 447.5	BNDM2_i32 4 709.0	UFNDM2_i32 6 085.7	GSB2_i32 6 812.6	SBNDM2_2_i32 6 890.2	FSBNDMq_q4f2a64 6 906.6

表 4 DNA 序列(E.coli)的实验数据 (MB·s^{-1})

E.coli	m=4	m=8	m=16	m=32	m=64	m=128
1	UFNDM4b_a64 1 187.9	**SBNDM4_sb_i32** **2 956.5**	**SBNDM4_sb_i32** **5 118.5**	**SBNDM6_sb_i32** **6 136.1**	**SBNDM6_sb_a64** **6 691.8**	**SBNDM6_sb_a64** **6 657.7**
2	FSO_56_a64 1 152.0	FSBNDMqb_q4f1i32 2 947.1	FSBNDMqb_q6f2i32 4 926.5	SBNDM6b_i32 5 973.6	FSBNDMq_q5f1a64 6 392.6	FSBNDMq_q5f0a64 6 400.8
3	FSBNDMqb_q4f1a64 1 121.2	SBNDM4b_a64 2 205.1	SBNDM4b_a64 4 218.9	FSBNDMqb_q6f0i32 5 968.8	QF_6_2_i32 6 138.6	BNDM6_a64 6 159.9
4	**SBNDM2_2_sb_i32** **904.5**	UFNDM4_i32 2 131.7	UFNDM6b_i32 3 690.4	UFNDM6_a64 5 898.0	UFNDM8_a64 6 124.1	UFNDM8_a64 6 120.5

表 5　英文文本(Bible.txt)的实验数据　　(MB·s^{-1})

Bible	*m*=4	*m*=8	*m*=16	*m*=32	*m*=64	*m*=128
1	**SBNDM2_2_sb_i32** **2 883.0**	**SBNDM2_2_sb_i32** **4 162.7**	**SBNDM4_sb_i32** **5 597.7**	**SBNDM4_sb_i32** **6 195.1**	FSBNDMq_q5f1a64 6 400.2	FSBNDMq_q5f1a64 6 389.1
2	GSB2b_i32 2 704.3	GSB2b_i32 3 789.0	SBNDM4b_i32 4 712.2	FSBNDMq_q5f0i32 5 946.3	**SBNDM6_sb_a64** **6 328.0**	**SBNDM6_sb_a64** **6 299.6**
3	EBOM 2 235.0	SBNDM2_2b_i32 3 424.5	FSBNDMqb_q4f0i32 4 628.1	UFNDM6_a64 5 923.9	BNDM4_a64 6 248.5	BNDM4_a64 6 227.2
4	SBNDM2_2b_i32 2 158.0	FSBNDMqb_q4f2i32 3 374.3	UFNDM4_i32 4 361.0	BNDM4_a64 5 774.6	BXS5_a64 6 154.6	UFNDM8_a64 6 071.6

表 6　自然语言样本(world192.txt)上的实验数据　　(MB·s^{-1})

world192	*m*=4	*m*=8	*m*=16	*m*=32	*m*=64	*m*=128
1	**SBNDM2_2_sb_i32** **3 007.8**	**SBNDM2_2_sb_i32** **4 610.4**	**SBNDM4_sb_i32** **5 556.2**	**SBNDM4_sb_i32** **6 171.5**	FSBNDMq_q4f1a64 6 444.8	FSBNDMq_q4f0a64 6 411.9
2	GSB2b_i32 2 896.3	GSB2b_i32 4 360.1	SBNDM2_2_i32 5 204.5	FSBNDMq_q3f0i32 5 945.7	BNDM4_a64 6 270.4	QF_3_4_i32 6 308.6
3	EBOM 2 336.4	SBNDM2_2b_i32 3 841.4	GSB2b_i32 5 104.4	BNDM4_i32 5 832.7	QF_3_4_i32 6 201.4	BNDM4_a64 6 263.7
4	SBNDM2b_i32 2 263.8	EBOM 3 747.7	BNDM2_i32 4 867.2	SBNDM2_2_i32 5 797.7	BXS5_a64 6 161.3	BXS3_a64 6 215.2

由实验数据可见，在本文实验平台上，对上述列出的共 36 种匹配条件中，在 21 种匹配条件下(占所有匹配条件的 58%)，SBNDMq_Shortb 系列算法性能高于其他已知算法。因本文实验已基本涵盖了所有已知算法，可认为本文提出算法，性能高于其他已知算法。

5　结束语

高性能精确单模式串匹配算法设计是计算机科学中的重要问题。本文通过引入非对齐双字节读机制改进了现有在线精确单模式匹配中性能较高的 SBNDMq 系列算法，并得到了 SBNDMq_Shortb 系列算法。实验结果表明，在多种匹配条件下，本文算法性能高于其他已知算法，改进效果明显。由本文研究可见，将串匹配算法中的处理单位由字节提高到更大的单位如 short 型整数、int 型整数甚至更大的单位可以有效地改进算法的性能。因此，下一步将研究如何提高现有算法的最小处理单位，以期获得更多的高性能算法。同时，现有多模式匹配算法中性能较高的 BG[16]算法是一种由 BNDM 算法扩展而来的多模式匹配算法，而本文所述算法也是一种 BNDM 类算法，其性能比 BNDM 算法更高，因此，可以考虑将本文所述算法向多模式匹配领域进行扩展，以期获得更高性能的多模式匹配算法。

参考文献

[1] Faro S, Lecroq T. The Exact String Matching Problem: A Comprehensive Experimental Evaluation[EB/OL]. (2012-10-06). http://arxiv.org/pdf/1012.2547.

[2] Faro S, Lecroq T. The Exact Online String Matching Problem: A Review of the Most Recent Results[EB/OL]. (2012-10-06). http://www-igm.univ-mlv.fr/~lecroq/articles/acmsurv2013. pdf.

[3] Durian B, Holub J, Peltola H, et al. Tuning BNDM with q-grams[C]//Proc. of the 11th Workshop on Algorithm Engineering and Experiments. New York, USA: [s. n.], 2009.

[4] Navarro G, Raffinot M. A Bit-parallel Approach to Suffix Automata: Fast Extended String Matching[R]. Santiago, Chile: Department of Computer Science, University of Chile, Technical Report: DC-98-1, 1998.

[5] 范洪博，姚念民. 一种高速精确单模式串匹配算法[J]. 计算机研究与发展, 2009, 46(8): 1341-1348.

[6] Peltola H. Variations of Forward-SBNDM[C]//Proc. of Prague Stringology Conference. Prague, Czech: [s. n.], 2011.

[7] Faro S, Lecroq T. String Matching Research Tool[EB/OL]. (2012-10-06). http://www.dmi.unict.it/~faro/smart/.

[8] Fredriksson K, Grabowski S. Practical and Optimal String Matching[C]//Proc. of SPIRE'05. 2005: 376-387.

[9] Ďurian B, Peltola H, Salmela L. Bit-parallel Search Algorithms for Long Patterns[C]//Proc. of SEA'10. 2010: 129.

[10] Fan Hongbo, Yao Nianmin, Ma Haifeng. A Practical and Average Optimal String Matching Algorithm Based on Lecroq[C]//Proc. of Internet Computing for Science and Engineering Conference. Harbin, China; [s. n.], 2011.

[11] Faro S. A Fast Suffix Automata Based Algorithm for Exact Online String Matching[C]//Proc. of CIAA'12. 2012: 149-158.

[12] Faro S, Lecroq T. A Multiple Sliding Windows Approach to Speed Up String Matching Algorithms[C]//Proc. of TIBS'12. 2012: 172-183.

[13] Lecroq T. Fast Exact String Matching Algorithms[J]. Information Processing Letter, 2007, 102(6): 229-235.

[14] Baeza-Yates R A. A New Approach to Text Searching[J]. Communications of the ACM, 1992, 35(10): 74- 82.

[15] Faro S, Lecroq T. Efficient Variants of the Backward-Oracle-Matching Algorithm[C]//Proc. of Prague Stringology Conference. Prague, Czech: [s. n.], 2008.

[16] Salmela L, Tarhio J, Kytöjoki J. Multi-pattern String Matching with q-grams[J]. ACM Journal of Experimental Algorithmics, 2006, 11(1): 1-19.

编辑　索书志