一种基于魂芯 DSP 的单模式位并行串匹配算法

陈 瑞 顾乃杰* 叶 鸿
(中国科学技术大学计算机科学与技术学院 安徽 合肥 230027)
(中国科学技术大学安徽省计算与通信软件重点实验室 安徽 合肥 230027)

摘 要 在多媒体技术飞速发展的今天,DSP 处理器以其低功耗和高性能等特点在信号处理和图像检索领域有着重要的应用。串匹配作为信号处理和图像检索应用中的基本算法,其性能和效率也因此受到越来越多的关注。通过结合 DSP 处理器的分簇结构和零开销循环技术,并利用字符串分段的方法提出一种基于 DSP 的位并行串匹配算法 EPSO。该算法可有效减少条件分支语句的时钟开销和分簇执行过程中的漏配次数,加速了串匹配过程。在国产魂芯 DSP 的仿真结果表明:EPSO 算法的匹配速度是经典 Shift-Or 算法的 7.8 倍左右,串匹配效率得到有效提升;以 KMP 算法为基准,英文语料下该算法的平均匹配速度是 KMP 算法的 6.3 倍左右,DNA 序列下是 KMP 算法的 10.5 倍左右,相比 NEW、S2BNDM 算法均具有显著的性能提升。

关键词 串匹配 移位或算法 魂芯 DSP 分簇 位并行

中图分类号 TP3 **文献标志码** A DOI:10.3969/j.issn.1000-386x.2020.07.041

A SINGLE MODE BIT-PARALLEL STRING MATCHING ALGORITHM BASED ON HXDSP

Chen Rui Gu Naijie* Ye Hong
(*School of Computer Science and Technology*, *University of Science and Technology of China*, *Hefei 230027*, *Anhui*, *China*)
(*Anhui Key Laboratory of Computing and Communication Software*, *University of Science and Technology of China*, *Hefei 230027*, *Anhui*, *China*)

Abstract As the rapid development of multimedia technology, DSP processors have important applications in signal processing and image retrieval with their low power consumption and high performance. String matching, the basic algorithm in signal processing and image retrieval applications, is gaining more and more attention to its performance and efficiency. Based on DSP, we propose a bit parallel string matching algorithm called EPSO by combining the clustering structure of DSP and zero-overhead loop technology and using string segmentation. It can effectively reduce the clock overhead of conditional branch statements and the missing allocation problem in the clustering process, and accelerate the string matching process. The experimental results of the algorithm in HXDSP emulator show that the matching speed of EPSO algorithm is about 7.8 times that of the classical Shift-Or algorithm, and the string matching efficiency is significantly improved. Based on KMP algorithm, the average matching speed of the algorithm in English text is about 6.3 times that of KMP algorithm. The DNA sequence is about 10.5 times that of KMP algorithm, which has significant performance improvement compared to NEW and S2BNDM algorithms.

Keywords String matching Shift-or algorithm HXDSP Clustering Bit-parallel

0 引 言

串匹配是在已知字符序列(文本串)中按照一定的匹配条件搜索给定字符序列(模式串)集合中元素出现位置的搜索问题,是生物信息学、信号处理和信息检索等方向的核心问题,在涉及字符处理相关领域中具有广泛的应用[1]。随着信息时代数据的高速增长,如何快速高效地处理大规模数据的串匹配问题是当前研究的重点。串匹配问题包括多模式匹配和单模式匹配。其中,多模式匹配的研究大多根据自动机原理,其经典算法包括 AM 算法[2]、AC 算法[3]、FS 算法[4]等。

收稿日期:2019-05-17。**陈瑞**,硕士生,主研领域:算法优化,并行计算。**顾乃杰**,教授。**叶鸿**,博士生。

单模式匹配是研究串匹配问题的基础,且其他模式匹配算法均由单模式匹配扩展而来,应用领域也最为广泛,已经成为涉及文本和符号处理领域应用的基本组件。因此,研究精确单模式串匹配问题有重要意义。

另一方面,随着多媒体技术的飞速发展,DSP 处理器以其低功耗和高性能等特点在信号处理和图像检索等领域有着重要的应用。高性能 DSP 处理器不仅拥有强大的计算能力,还具有多种先进技术,单指令流多数据流(Single Instruction Multiple Data,SIMD)、超长指令字(Very Long Instruction Word,VLIW)等特殊的硬件结构使得字符串匹配算法在 DSP 上得以并行处理。本文提出一种基于魂芯 DSP 的精确单模式位并行串匹配算法,并在平台上进行仿真实验。

单模式串匹配算法众多,代表性的有按照前缀匹配规则的 KMP 算法[5]、Shift-Or/And(SO/SA)算法[6]和按照后缀匹配规则的 Horspool 算法[7]和 SBNDM2 算法[8]。KMP 算法第一个在线性时间解决了串匹配问题,该算法利用已经匹配的已知信息,实现无回溯的字符比较,具有线性时间复杂度。Shift-Or/And 算法是一种最基础的位并行算法,实现线性的串匹配过程,串匹配速度远快于 KMP,该算法维护一个字符串的字符集合,然后利用位操作的内部并行性,实现非确定性有穷自动机(Non-deterministic Finite Automaton,NFA)的全部状态转移。按后缀规则匹配的 Horspool 算法是简化了的 BM(Boyer-Moore)算法[9],是一种以空间换时间的"跳跃式"匹配算法,在实际应用中比 BM 算法具有更好的性能。SBNDM2 算法在英文语料或者大字符集下搜索具有较高的性能。1985 年 Galil[10]研究了一种基于软件的并行串匹配算法,可以在 $O(\log_2 m)$ 时间复杂度下完成串匹配过程,m 是模式长度。该算法的缺点是只适用于对固定字符集进行串匹配。同年,Vishkin[11]改进了 Galil 提出的并行算法,使其支持字符集可变的情况。戴正华[12]从计算机体系结构的角度出发,基于 Shift-Or 算法通过组合 NFA 状态,降低自动机状态转换的次数,提高匹配效率,在不考虑 Cache 失效的情况下,算法比较次数可减少一半。对于并行串匹配算法,随着近些年 DSP、GPU 和 FPGA 等硬件的高速发展,把经典算法移植到硬件平台上以提高匹配效率成为研究的热点。侯淼[13]基于 GPU 高性能并行架构,把经典的 AC 算法移植到 GPU 平台进行加速,设计实现了数据并行的 PAC 算法,并取得了显著的加速效果。Irwin 等[14]实现了基于 FPGA 的并行串匹配算法,相比软件实现方式,算法性能得到显著提升。但该方法由于硬件限制,不易操作,难以广泛应用。李璋等[15]将串匹配算法在 CPU、GPU、FPGA 上进行了性能的测试与对比,并在 GPU 上充分利用 SIMD 结构对算法并行化,在 FPGA 上利用大量内部逻辑运算单元,实现了高效的串匹配算法。

本文首先通过实验对比了 KMP、Horspool、SO 三种经典串匹配算法在魂芯 DSP 处理器上的执行效率;其次结合处理器的分支预测和零开销循环等技术对算法的条件分支语句进行优化,以减少分支语句带来的额外时钟开销;最后结合 DSP 处理器分簇结构和无漏配的字符串分段方法提出一种基于魂芯 DSP 的精确单模式位并行串匹配算法 EPSO。该算法通过将文本串划分给处理器若干执行簇,减少分簇执行过程中的漏配次数,充分发挥硬件特性,从而有效提升串匹配算法性能。

1　研究背景

1.1　平台介绍

魂芯 DSP 是国内某研究所自主研发的一款高性能数字信号处理器[16],可广泛应用于各种高性能计算领域。作为魂芯 DSP 系列的第二代产品,HXDSP 1042 是一款 32 位浮点 DSP,采用哈佛结构、超流水线、超长指令字等并行技术。HXDSP 1042 处理器体系结构如图 1 所示。

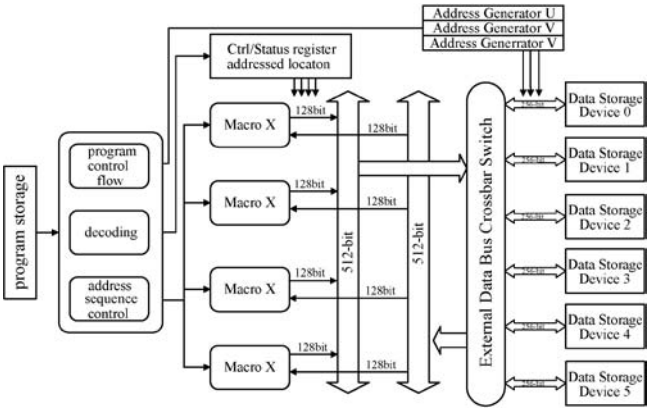

图 1　HXDSP 1042 体系结构

(1) 内核结构称为 eC104 +,核内包含 4 个结构和功能完全相同的执行单元(Element Operation Unit),分别用 X、Y、Z、T 表示。每个单元包含 128 个通用寄存器、8 个乘法器(Multiplier,MUL)、8 个算数逻辑运算单元(Arithmetic Logic Unit,ALU)、4 个移位器(Shifter,SHF)、1 个超算器(Super Unit,SPU)。

(2) 处理器含有 3 个相互独立的地址产生器,分别用 U、V 和 W 表示。内部数据总线采用非对称全双工总线,内部数据读总线位宽为 512 位,内部数据写总线位宽为 256 位。对于浮点复数而言,支持"两读一写"或"两写一读",前者是指一个时钟周期每个执行

宏读取 2 个 32 位浮点复数且写回 1 个 32 位浮点复数。后者是指一个时钟周期每个执行宏写回 2 个 32 位浮点复数且读入 1 个 32 位浮点复数。

(3) 处理器采用 VLIW 技术,单周期内可并行执行 16 条共计长达 512 位宽的指令。条件跳转等分支指令必须排布在指令槽的首部,且每个指令槽至多有一条此类指令。

1.2 相关算法

蛮力算法(Brute Force,BF)是一种最基本的串匹配算法[12]。该算法首先比较模式串的第一个字符和文本串的第一个字符,如果相等,继续比较二者的后续字符;否则,比较模式串的第一个字符和文本串的第二个字符是否相同,以此类推,直到得出匹配结果。该算法本质上是基于模式串前缀规则进行匹配,常见的前缀串匹配算法还有 KMP 和基于位并行的 Shift-Or 等,基于后缀进行搜索的算法有 BM 和 Horspool 等。下面以 KMP、Shift-Or、BM、Horspool 算法为例描述各个算法的基本原理。

(1) KMP 算法。KMP 算法从左至右逐字符地扫描文本串的当前窗口,如果在模式串的位置 i 处发生失配,则文本串向前移动若干距离,即移动 $i-f_p(i)$ 个字符。其中,$f_p(i)$ 定义[13]如下式所示,i 为 0 时取 -1,否则取最大可偏移的值。

$$f_p(i)=\begin{cases}-1\\ \max(0\leqslant k<i \mid p[0\cdots k-1]=p[i-k\cdots i-1]\cap p[k]\neq p[i],-1)\end{cases} \tag{1}$$

如果找到一个匹配,则令 $i=s$,窗口向前移动距离为 $f_p(s)$,下一个窗口对齐文本串 $s+i-f_p(i)$ 的位置。简单地说,KMP 算法记录的模式串前缀,同时也是已经扫描的文本串的最长后缀字符串。图 2 描述了文本串为“ababcab”,模式串为“abcac”的 KMP 算法匹配过程。

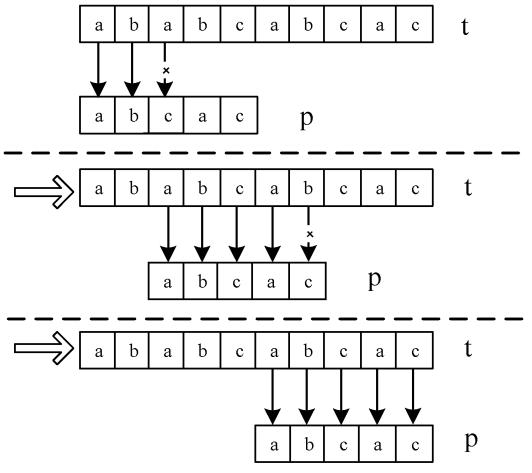

图 2 KMP 算法匹配示意图

(2) Shift-Or(SO)算法。SO 算法和 KMP 算法类似,也是基于前缀规则进行字符串匹配,但其利用计算机机器字位运算的内在并行性,将多个值装入同一个机器字内,从而加速了串匹配过程。算法 1 描述了 SO 算法的基本执行过程,该算法主要维护一个集合 D,该集合中保存了所有既是模式串前缀同时又是文本串前缀的字符串。每次读入新字符后,算法利用位并行的方式更新集合。算法中变量 B 是一个辅助表,也可以称之为字典,其关键字是模式串中的每个字符,值是一个 n 位的无符号数,记录该字符在模式串 p 中出现的位置。本文串和模式串分别用 t 和 p 表示。假设处理到模式串字符 $p[i]$,则对 D 进行更新。由于集合 D 标识了字符 $p[0\cdots j]$ 是否为文本串 $t[0\cdots i]$ 的后缀,所以 $D[j]=0$ 当且仅当更新前的 $D[j-1]=0$ 且 $t[i]=p[j]$。显然,当 $D[m-1]$ 值为 0 时,字符 $p[0\cdots m-1]$ 是 $t[0\cdots i]$ 的后缀,找到一个串 p 的完全匹配。

算法 1 移位或(Shift-Or,SO)算法

输入:$p=p_1p_2\cdots p_m$,$t=t_1t_2\cdots t_n$

输出:$[idx_1\cdots idx_s]$

```
1.  //Preprocessing
2.  for c ∈ ∑ do B[c] = 0^m
3.  for j∈1⋯m do B[pj] = B[pj]&1^m
4.  //Searching
5.    D = 1^m
6.  for pos∈1⋯n do
7.        D = (D<<1) | B[t_pos]
8.        if D | 01^(m-1) ≠ 1^m then
9.           output "pos - m + 1"
```

(3) Boyer-Moore(BM)算法。BM 算法的基本思想是对模式串从右向左进行逆向匹配,在预处理过程中先构造坏字符表和好后缀表。在匹配时利用坏字符和好后缀规则,跳过尽量多的字符。算法包括两个规则。一个是坏字符规则,假设文本串为 t,则从右向左扫描模式串 p,若发现某个字符发生失配,则按照两种情况分别讨论。如果当前文本串的字符 $t[i]$ 在模式串 p 中没有出现,则直接将模式串向右跳过模式串长度的距离,否则以最右出现该字符的位置进行对齐。另一个是好后缀规则,好后缀表示已经成功匹配的后缀串,共有以下三种情况。

情况一:模式串中存在最左子串和好后缀相匹配,将模式串向右移动,使得子串与之对齐即可。

情况二:模式串中不存在最左子串和好后缀匹配,寻址模式串的一个最长前缀,并将该前缀串等于好后缀的后缀串,然后使两者对齐。

情况三:模式串没有子串和好后缀匹配,并且模式

串中找不到最长前缀,移动模式到好后缀的下一个字符即可。

(4) Horspool 算法。BM 算法简化后得到 Horspool 算法,该算法基于文本串从后向前匹配时,将主串匹配窗口的最后一个字符和模式串的最后一个字符比较,若匹配成功,则从后往前继续比较;若匹配失败,则根据主串匹配窗口的最后一个字符 β 在模式串中下一个出现的位置将窗口向右移动,如图 3 所示。该算法对 BM 的坏字符规则进行了改进,BM 算法的对齐字符是当前发生失配时文本串当前失配字符,而 Horspool 算法对齐的是文本串匹配窗口的最后一个字符,即字符 β,该字符与模式串中最近出现的 β 对齐。

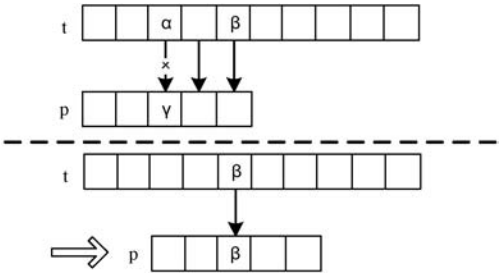

图 3　Horspool 算法匹配示意图

2　基于 DSP 的位并行串匹配算法

本节首先通过具体实验对比分析 KMP、Horspool 和 Shift-Or 算法在 DSP 上的执行效率,结合硬件特性讨论各串匹配算法的性能差异;其次利用零开销循环技术对算法的条件分支语句进行优化;最后结合处理器分簇结构和零开销循环技术,提出一种基于 HXDSP 的高效位并行串匹配算法。

2.1　算法分析

选取 KMP、Horspool 和 Shift-Or 三种串匹配算法在 HXDSP 处理器上进行实验,测试数据为经典字符集 Holy-bible. txt[17](176 813 个字符),模式串用随机抽取的方式抽取 9 个不同长度的串作为测试样本。图 4 显示了三种算法在 HXDSP 处理器上的运行结果。

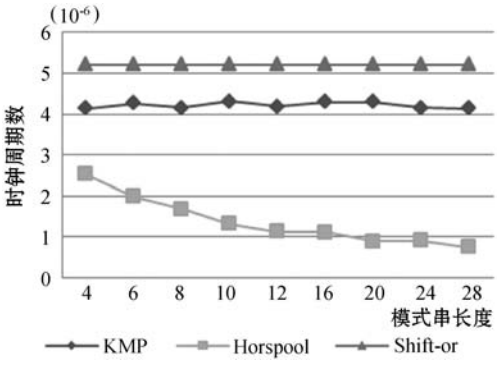

图 4　基于 HXDSP 的串行串匹配算法

测试结果表明,Shift-Or 算法和 KMP 算法性能基本不受模式串长度的影响,针对特定的测试串,其性能基本可以保持稳定;基于后缀规则匹配的 Horspool 算法不仅性能相对较好,并且随着模式串长度的增加,时钟周期数显著减少。该算法在字符集较大时性能会更好,主要原因在于当模式串长度越大,匹配窗口向前滑动的距离越远。KMP 算法作为前缀算法,模式串长度的变化会对算法性能产生一定的影响,主要是因为文本串向前移动距离的不同导致时钟开销出现一定差异。基于位并行的前缀匹配 SO 算法,由于匹配过程包含大量位运算,并且每一次通过采用位运算的方式进行比较,完成的任务量和模式串长度不相关,因此该算法的时钟周期基本保持稳定。

串匹配算法的性能和处理器硬件结构密切相关。从硬件层面出发,为了提高算法性能,需要尽可能多地或者尽可能早地执行指令。SIMD 是高性能处理器的一个重要特性,该技术通过增加指令中完成的操作数,减少完成任务或程序所需要的指令来提高处理器性能。这使得在高性能处理器上加速串匹配算法成为可能,对于上述三种串匹配算法,由于 KMP 和 Horspool 算法在匹配过程中是根据一定的规则跳过若干字符距离来提高串匹配效率,若采用数据并行策略,将划分好的文本串交给不同的函数单元进行同时匹配,算法将不能利用 SIMD 技术的优势,并且会导致无法同时做同样的匹配操作。对于位并行的 Shift-Or 算法,其核心思想是通过更新掩码集合 D 来记录模式串的前缀匹配情况,这种匹配过程能够很好地应用在 SIMD 结构中,采用分治的思想通过将划分好的文本串同时在 DSP 处理器的四个簇中并行执行,能够更好地利用 HXDSP 的硬件资源,充分发挥处理器的计算能力,从而加速串匹配过程。

2.2　算法实现

上述串匹配算法中,基于位并行的 Shift-Or 算法在高性能 DSP 处理器中能够通过分簇结构更好地将算法并行化。下面首先从 HXDSP 指令级优化层面对串行代码进行优化,再采用分治思想提出合适的字符串分段方法对算法实施并行化,最后提出高效的基于位并行的 EPSO 算法。

分析算法 1 的循环部分,其中针对辅助表 B 的预处理部分耗时可忽略不计(文本串长度达到一定规模后),时钟开销主要集中在更新集合 D 过程中,即算法 1 的第 6 行至第 9 行的循环体部分。从指令层面对算法 1 进行优化,包括计算开销和条件分支开销。针对

计算,除了尽可能提高指令行的并行处理能力之外,需考虑流水线的数据相关。算法 1 中的 D 进行更新过程中的移位操作、或操作串行性强,无法利用循环展开的方法来减少运算过程中的流水线停顿,即循环体内涉及串行运算的时钟开销已经没有更多的优化空间。反观条件跳转部分,由于每一次循环都要产生额外的 10 个时钟周期,算法 1 的第 8 行判断是否找到一个成功匹配同样是条件判断操作,同样需要产生一定的时钟开销,相比运算部分,这两个条件判断所占用的时钟周期是运算部分耗费时间的两倍,采用分支预测可以有效提高条件判断和条件分支语句的执行效率。

算法 2 优化条件分支语句。

HXDSP 1042 处理器提供如下两条指令:

指令 1:if {x,y,z,t}Rm[bit] ==0/1 b <pro>

指令 2:if lc0/nlco b <pro>

上述指令在串匹配算法的条件分支语句中具有重要作用。指令 1 表示寄存器 Rm 的第 bit 位为 0 或 1 时跳转至 pro 处,该指令简化了算法 1 第 8 行成功找到匹配点的过程,省去掩码计算过程和掩码与集合 D 的或非操作,由于该过程在算法 1 的串行版本下耗费 4 至 6 个时钟周期,循环体内计算过程以及条件分支判断总的耗费时钟周期为 40 左右,因此该指令可以减少算法 10% ~15% 的时钟开销。指令 2 是基于 LC0 和 LC1 寄存器的分支指令,预测机制较为特殊,该指令在取指时读取 LC0 和 LC1 寄存器的值,并以此作为是否分支的判断依据。该指令同样能在每个循环体内减少 8 个时钟开销左右,带来 20% 左右的性能提升。

为了发挥 HXDSP 四个簇的计算能力,针对算法 2 考虑四路并行化,提出一种字符串分段方法以提高计算资源和带宽使用率,进一步提升串匹配效率。并行化的基本思想是将长度为 n 文本串平均分成 4 组,对于不能整除的用空字符填充,记为 $Ti(i=1,2,3,4)$。将四组长度相等的文本串分别和模式串 P 匹配,实现文本匹配过程的并行处理。分段方法的关键在于子文本串在相邻处有可能出现遗漏现象,由于在各段子串的连接处有可能存在和模式串相等的串,因此在将各子串分给不同的执行簇时有一定的概率出现漏配问题。为了避免漏配,有以下 3 种解决方案。

(1) 由于四个簇并行执行,那么分配给四个簇的文本串长度要保证一致。为了解决相邻子文本串的漏匹配问题,考虑为每个簇分配长度为 $n/4+m-1$ 的串。

$$T\{x,y,z,t\}=Ti+m-1 \quad i=1,2,3,4 \qquad (2)$$

式中:$T\{x,y,z,t\}$ 表示分配给每个簇的字符串,m 是模式串的长度,字符 $Tx\{n/4\cdots n/4+m-1 \quad x,y,z,t\}$ 表示簇 X 文本串的后 m 个序列,该序列由 $T1$ 的最后一个字符和 $T2$ 的前 $m-1$ 个字符组成。Ty 则由 $T2$ 和 $T3$ 的前 $m-1$ 个字符组成。Tz 同理,Tt 则不同,除了包括 $T4$ 之外,需要额外补充长度为 $m-1$ 的字符,如图 5 中长为 $m-1$ 的区域,以保证四个簇的文本串长度一致。

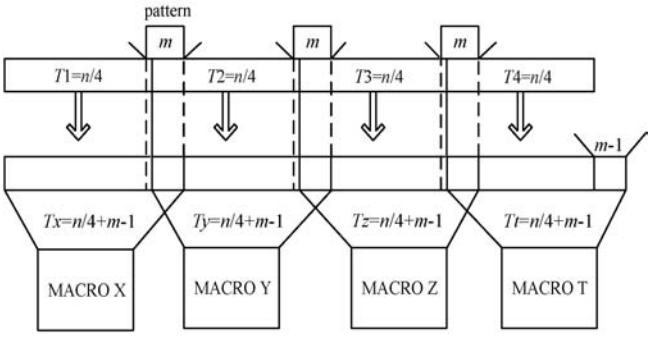

图 5 基于 HXDSP 的并行串匹配过程

(2) 针对 Tt 文本串改进,将 Tt 的起始位置向前回溯 $m-1$ 个字符,如图 6 所示,尽管这样 Tt 的匹配和 Tz 的匹配存在 $m-1$ 次的重复匹配,但硬件资源充足情况下,省去补充字符的过程,简化了 Tt 的匹配过程。

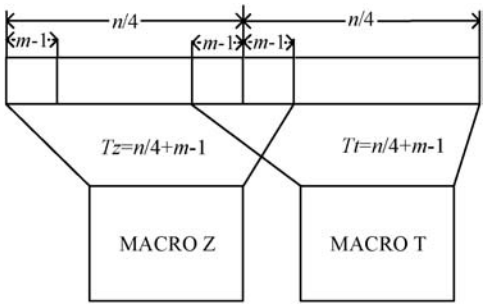

图 6 簇 T 上串划分的改进方案

(3) 由于 Tt 额外重复匹配了 $m-1$ 次,考虑继续对方案二进行优化,把该 $m-1$ 次重复匹配均匀地分配给其余三个执行单元,即每个簇需要匹配的字符减少了 $(m-1)/4$ 个,如下式所示:

$$T\{x,y,z,t\}=Ti+3(m-1)/4 \quad i=1,2,3,4 \qquad (3)$$

由于基于位并行的 SO 算法受字长限制,长度 m 往往小于处理器字长(32 或 64),文本串在一定规模时,这种优化带来的提升几乎可以忽略,并且该方案对 m 的取值敏感,当 $m-1$ 不能整除 4 时,依然需要补充额外字符,这种优化策略导致的额外时钟开销远大于其带来的性能提升。

综上所述,结合处理器的分簇结构和上述分段方法,可有效提高串匹配算法效率。在算法 2 的基础上按照上述方案(2)的划分方法,提出一种有效的并行移位或串匹配算法,称之为 EPSO,伪代码如算法 3 所示。该算法对文本串的拆分存在 4 无法被 n 整除的现象,会导致 1 ~3 个串出现漏匹配的问题。为了解决该问题,采用方案一,在文本串 text 末尾补充 e 个空字

符,其中 $e=4-n\%4$,然后按照上述算法进行并行化,匹配完成时为了消除补充字符对串匹配结果的影响,记簇 T 中成功匹配点为 p,忽略 $p>|n/4|-n\%4$ 的设置辅助表 B 和匹配过程中用于更新的集合 D,用本节介绍的字符串分段方法结合处理器的分簇结构把经过预处理的串传输给不同的执行簇,分别按照位并行的方式对 D 进行移位或计算,最后采用算法 2 中的条件判断指令分别输出四个簇的成功匹配点。

算法 3　高效位并行算法(EPSO)

输入:$p=p_1p_2\cdots p_m, t=t_1t_2\cdots t_n$

输出:$[idx_1\cdots idx_s]$

1. //Preprocessing
2. for $c\in\sum$ do $\{x,y,z,t\}B[c]=0^m$
3. for $j\in 1\cdots m$ do $\{x,y,z,t\}B[p_j]=B[p_j]\&1^m$
4. //Searching
5. $\{x,y,z,t\}D=1^m \parallel i_1=1 \parallel i_2=n/4+1$
6. $i_3=n/2+1 \parallel i_4=3n/4-m+2$
7. for $k\in 1\cdots(n/4+m-1)$ do
8. $\{x,y,z,t\}D=(\{x,y,z,t\}D<<1)\mid B[t\{i_1,i_2,i_3,i_4\}]$
9. if $\{x\}D[bit]==0$ then output $"k-m+1"$
10. if $\{y\}D[bit]==0$ then output $"n/4+k-m+1"$
11. if $\{z\}D[bit]==0$ then output $"n/2+k-m+1"$
12. if $\{t\}D[bit]==0$ then output $"3n/4+k-m+1"$
13. $i_1++ \parallel i_2++ \parallel i_3++ \parallel i_4++$

3　实验分析

本文实验基于 HXDSP 1042 仿真平台[18],待匹配文本为测试集 Holy-bible. txt(703 KB)和 E. coli 的 DNA 序列(coli_str_k_12_substr_mg1655,15,2 143 bp),在文本中随机抽取 100 个串作为测试样本。为了精确统计串匹配算法消耗时间,实验采用全局寄存器 CC0 进行计数。为避免磁盘影响实验结果,所有字符串均已读入内存,只针对匹配过程进行计时,测试三种位并行算法不同匹配条件下的匹配效率。

实验选取 KMP 算法和 Horspool 算法作为 SO、ESO、EPSO 算法的对比算法,实验结果如表 1 和表 2 所示。数据表明,703 KB 大小的英文文本串下,ESO 算法的平均匹配时间达到 2.855 ms,EPSO 算法的平均匹配时间达到 0.716 ms,和经典的 SO 算法相比,加速比分别达到 1.96 和 7.81。当文本串选取拥有 152 143 个碱基对的大肠杆菌 DNA 序列时,ESO 算法的平均匹配时间达到 3.317 ms,EPSO 算法的平均匹配时间达到 0.834 ms,相比 SO 算法,加速比分别可达 1.94 和 7.72 左右。HXDSP 的实验结果表明,基于位并行的 SO 算法通过在该处理器上的并行实施和优化,在英文语料和 DNA 序列中均有良好的性能提升。

表 1　DNA 字符集下串匹配时钟开销

E. coli	KMP	Horspool	SO	ESO	EPSO
4	5 874 478	3 131 412	4 425 875	2 283 637	572 908
6	5 620 382	2 914 470	4 414 039	2 283 671	572 856
8	6 189 696	2 877 650	4 413 791	2 283 705	572 914
10	6 026 068	2 480 126	4 413 686	2 283 739	572 782
12	5 955 113	1 919 918	4 413 865	2 283 773	572 663
16	5 847 294	2 159 421	4 413 842	2 283 841	572 941
20	6 482 064	1 358 869	4 414 073	2 283 920	572 893
24	5 838 851	1 570 736	4 414 050	2 283 988	573 039
28	6 420 792	1 222 587	4 414 281	2 284 056	572 928

表 2　英文语料下串匹配时钟开销

Bible	KMP	Horspool	SO	ESO	EPSO
4	4 139 303	2 527 033	5 213 089	2 653 709	667 347
6	4 268 366	1 974 643	5 213 156	2 653 743	667 395
8	4 153 324	1 686 443	5 213 193	2 653 766	667 302
10	4 310 534	1 312 659	5 213 245	2 653 800	667 324
12	4 190 881	1 121 259	5 213 297	2 653 834	667 381
16	4 301 442	1 113 678	5 213 401	2 653 902	667 596
20	4 298 394	892 895	5 213 505	2 653 970	667 318
24	4 156 368	899 358	5 213 609	2 654 038	667 443
28	4 142 126	749 868	5 213 713	2 654 095	667 556

为了更好地对比实验结果,将本文算法与范洪博等[20]在 Intel E2160 处理器上的串匹配工作进行比较。一般来说,魂芯 DSP 处理器相比现阶段主流 Inter 处理器拥有更低的主频和功耗。Inter E2160 处理器主频达到 2.7 GHz,功耗为 65 W,远高于魂芯 DSP 功耗,两者具有相同的核心数和相当的制作工艺。

本文工作基于单核四路 SIMD 的魂芯 DSP 进行,范洪博等基于位并行原理提出 S2BNDM 算法,并在 Intel E2160 处理器上进行测试,该算法相比 NEW、Horspool、KMP 等的性能有明显提升。其中 NEW 算法在小字符集和长模式下具有显著的优势,在英文语料情况下,算法 S2BNDM 和 S2BNDM' 分别在短模式和长模式下具有较高的匹配效率。结果的对比以 KMP 算法为基准,选取单模式匹配 NEW[19]、S2BNDM[20]、S2BNDM'[20] 作为对比算法,分别比较上述几种高效算法和 EPSO 算法相比各平台 KMP 算法的性能提升情况,对比结果如图 7、图 8 所示。

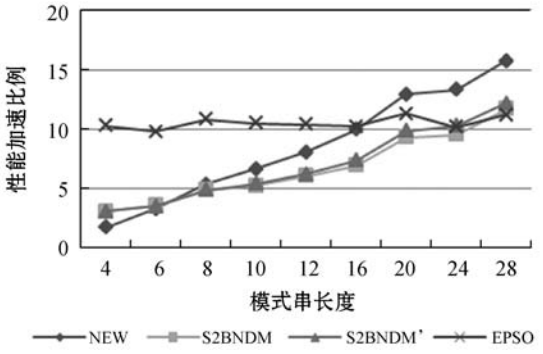

图 7 DNA 序列串匹配算法增速对比

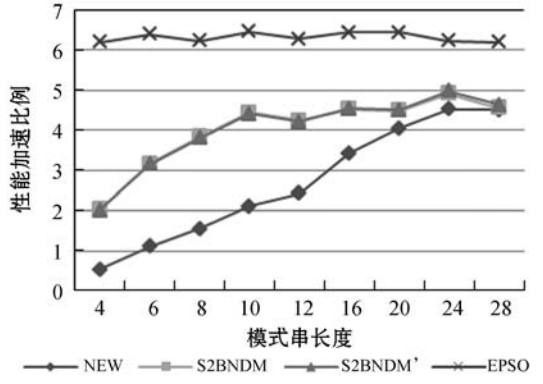

图 8 英文语料串匹配算法增速对比

结果表明,在 DNA 字符集下,当模式串长度小于 16 时,相比 KMP,EPSO 算法匹配速度快于其他三种算法。当模式串长度大于 16 时,EPSO 算法可带来的性能加速比例略小于 NEW 算法,且和 S2BNDM 算法和 S2BNDM'算法相当,这主要是因为 EPSO 算法属于非模式敏感型算法,而其他三种算法在模式长度不同时,匹配过程中跳跃的距离不同,性能也不同。在英文语料下,以 KMP 算法为基准,EPSO 算法带来的性能提升明显高于其他三种算法,EPSO 算法的匹配速度是 KMP 算法的 6.5 倍左右。根据前文对两种处理器的分析,在各项指标不超过 Inter 处理器的情况下,基于魂芯 DSP 的单模式位并行算法 EPSO 比 NEW、S2BNDM 等算法表现出更好的性能提升效果。

4 结 语

本文通过分析串匹配算法在 DSP 上的执行效果,结合 HXDSP 处理器平台的分簇体系结构和零开销循环等技术,提出一种基于位并行的无漏配的高效串匹配算法,能充分发挥硬件资源优势,有效提升单模式串匹配算法的效率。该算法在电子对抗、入侵检测等方面有应用价值,对推动国产信息化处理起着重要作用。

接下来的工作将在本文基础上,继续挖掘新一代多核魂芯 DSP 结构下位并行串匹配算法性能优化的潜力,将优化工作扩展到多模式匹配和近似匹配等更多的方面。

参 考 文 献

[1] 范洪博. 高性能精确单模式串匹配算法研究[D]. 哈尔滨:哈尔滨工程大学,2009.

[2] Aho A V, Hopcroft J E. The design and analysis of computer algorithms[M]. Pearson Education India, 1974.

[3] Aho A V, Corasick M J. Efficient string matching: an aid to bibliographic search[J]. Communications of the ACM,1975, 18(6):333 - 340.

[4] Cantone D, Faro S. Fast-search: a new efficient variant of the Boyer-Moore string matching algorithm[C]//Proceedings of the 2nd international conference on Experimental and efficient algorithms, 2003:47 - 58.

[5] Knuth D E, Morris J H, Pratt V R. Fast pattern matching in strings[J]. SIAM Journal on Computing,1977,6(2):323 - 350.

[6] Baeza-Yates R A, Gonnet G H. A new approach to text searching[J]. ACM SIGIR Forum,1989,23(SI):168 - 175.

[7] Horspool R N. Practical fast searching in strings[J]. Software: Practice and Experience, 1980, 10(6):6.

[8] Holub J, Durian B. Fast variants of bit parallel approach to suffix automata [C]//Stringology Research Workshop' 2005, 2005.

[9] Boyer R S, Moore S J. A fast string searching algorithm[J]. Communications of the Acm, 1977, 20(10):762 - 772.

[10] Galil Z. Optimal parallel algorithms for string matching[J]. Information & Control, 1985, 67(1):144 - 157.

[11] Vishkin U. Optimal parallel pattern matching in strings[J]. Information & Control, 1985, 67(1):91 - 113.

[12] 戴正华. 面向体系结构的串匹配算法优化研究[D]. 北京:中国科学院研究生院(计算技术研究所),2006.

[13] 侯淼. 并行串匹配算法研究[D]. 哈尔滨:哈尔滨工业大学.

[14] Irwin S G, Venkat A A, Winberg S L, et al. FPGA-based string matching[C]//International Conference on Energy. IEEE, 2012.

[15] 李璋,杜慧敏,王涌钢. 字符串匹配算法的实现:CPU vs. GPU vs. FPGA[J]. 电子科技,2014,27(12):5 - 8.

[16] 洪一,方体莲,赵斌,等. "魂芯一号"数字信号处理器及其应用[J]. 中国科学:信息科学,2015,45(4):574 - 586.

[17] Charras C, Lecroq T. Handbook of exact string matching algorithms[M]. King's College Publications, 2004.

[18] 中国电子科技集团第 38 研究所. BWDSP104x 软件用户手册[M]. 合肥:中国电子科技集团第 38 研究所,2015.

[19] Lecroq T. Fast exact string matching algorithms[J]. Information Processing Letters, 2007, 102(6):229 - 235.

[20] 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.