SIMD指令在基因单模式串匹配里的应用

王力

2014年12月

中图分类号：TQ028.1

UDC分类号：540

SIMD指令在生物序列串匹配里的应用

作 者 姓 名 王力

学 院 名 称 计算机学院

指 导 教 师 戴林教授

答辩委员会主席 \*\*教授

申 请 学 位 工学硕士

学 科 专 业 计算机科学与技术

学位授予单位 北京理工大学

论文答辩日期 2015年1月

**Synthesis and Application on textile of the Shape Memory Polyurethane**

Candidate Name： Wang Li

School or Department: \*\*\*\*

Faculty Mentor: Prof. \*\*

Chair, Thesis Committee： Prof. \*\*

Degree Applied: \*\*\*\*

Major： \*\*\*\*

Degree by: Beijing Institute of Technology

The Date of Defence： \*，\*\*\*\*

SIMD指令在生物信息串匹配里的应用 北京理工大学

**研究成果声明**

本人郑重声明：所提交的学位论文是我本人在指导教师的指导下进行的研究工作获得的研究成果。尽我所知，文中除特别标注和致谢的地方外，学位论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京理工大学或其它教育机构的学位或证书所使用过的材料。与我一同工作的合作者对此研究工作所做的任何贡献均已在学位论文中作了明确的说明并表示了谢意。

特此申明。

签 名： 日期：

**关于学位论文使用权的说明**

本人完全了解北京理工大学有关保管、使用学位论文的规定，其中包括：①学校有权保管、并向有关部门送交学位论文的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存学位论文；③学校可允许学位论文被查阅或借阅；④学校可以学术交流为目的,复制赠送和交换学位论文；⑤学校可以公布学位论文的全部或部分内容（保密学位论文在解密后遵守此规定）。

签 名： 日期：

导师签名： 日期：

**摘要**

串匹配算法是一个在计算机科学里被广泛研究的问题，它在生物信息学、自然语言处理、信息检索里有着广泛的应用等。例如，串匹配算法常用来在基因序列中检索相似串或定位某一基因片段。截止目前，已经有一些专门针对生物序列的串匹配算法被提出，如tvsbs[1]，graspm[2]等。在高通量序列检测技术的快速发展下，获取生物基因序列变得越来越容易和便宜。同时也带来了海量基因序列匹配查询的巨大挑战。因此，设计更有效的生物序列匹配算法去应对这种挑战是及其重要的。

有很多精确串匹配算法在过去十几年里被提出。其中，有一个叫SSECP[3]的算法开始使用块字符技术。块字符就是将多个字符组合成一个字符块，该字符块能单独进行运算而不用逐字符匹配。

在本文中我们针对生物序列提出了一个使用块字符的精确字符串匹配算法。该算法使用一个优化的块字符大小和一个哈希值去减少全匹配的调用。其中，IEPSM通过一些单指令多数据流指令去计算哈希值。而对于优化的块字符大小，我们是通过一组实验得出来的。

本文后面的多个算法在不同生物序列的比较实验中显示出IEPSM获取了比其他算法更好的效率。因此在检索海量生物数据里，IEPSM将是一个广泛运用的工具。

**关键词：**串匹配；生物信息学；Intel SSE指令

**Abstract**

String matching is an important problem that has been thoroughly studied in computer science, with broad applications in bioinformatics as well as natural language processing, information retrieval, etc. For example, it is used to find similar sequence or locate a segment in a long sequence. Currently, several string matching algorithms are used on biological sequences, such as tvsbs, graspm etc. With the rapid development of high-throughput sequencing technologies, it has become easier and cheaper to obtain vast quantities of biological sequences, accordingly posing great challenges in searching for a specific pattern within a large volume of biological sequences. Therefore, it is of fundamental importance to design more effective string matching algorithms to address this challenge.

There are several algorithms that have developed for exact string matching in the past years. Among them, an algorithm called SSECP begin to use exact packed string matching technique, in which multiple characters are packed into one block-character, so that the characters can be compared in bulk rather than indi-vidually.

Here we present IEPSM, an exact packed string matching algorithm that is dedicated for biological sequences. IEPSM features optimized word-size packed strings and adopts a big hash value to decrease byte-by-byte comparisons. And IEPSM computes fingerprint values by a hash function using Single Instruction Multiple Data instructions, which sup-ports parallel execution of some operations via a set of special instructions. And we take some experiments to obtain the optimal shift distance.

Comparative results on multiple empirical datasets show that IEPSM achieves better efficiency by comparison with existing algorithms. Thus, IEPSM is of broad utility for searching a specific pattern in the era of big biological data.

**Key Words**: string matching; bioinformatics; Intel SSE instruction

目录

[第1章 绪论 1](#_Toc407328517)

[1.1 课题研究背景及意义 1](#_Toc407328518)

[1.1.1 课题的研究背景 1](#_Toc407328519)

[1.1.2 课题的研究意义 1](#_Toc407328520)

[1.2 针对生物学的字符串匹配算法的国内外研究现状及发展趋势 2](#_Toc407328521)

[1.2.1 相关研究工作 2](#_Toc407328522)

[1.2.2 存在的问题 5](#_Toc407328523)

[1.2.3 发展趋势 6](#_Toc407328524)

[1.3 本文的目标与组织结构 6](#_Toc407328525)

[第2章 精确字符串匹配算法综述 7](#_Toc407328526)

[2.1 基于前缀搜索的算法 8](#_Toc407328527)

[2.1.1 Knuth-Morris-Pratt算法 8](#_Toc407328528)

[2.1.2 Shift-And/Shift-Or算法 10](#_Toc407328529)

[2.2 基于后缀搜索的算法 11](#_Toc407328530)

[2.2.1 BM算法 11](#_Toc407328531)

[2.2.2 QS算法 14](#_Toc407328532)

[2.3 基于子串搜索的算法 14](#_Toc407328533)

[2.3.1 BOM算法 14](#_Toc407328534)

[2.3.2 SBDM算法和SBOM算法 15](#_Toc407328535)

[第3章 生物信息学基本概念 17](#_Toc407328536)

[3.1 核酸 17](#_Toc407328537)

[3.2 蛋白质 17](#_Toc407328538)

[3.3 高通量测序结果文件 18](#_Toc407328539)

[3.3.1 FASTA格式 18](#_Toc407328540)

[3.3.2 FASTQ格式 19](#_Toc407328541)

[3.3.3 数据样例 20](#_Toc407328542)

[第4章 SIMD指令集在串匹配算法中的应用 21](#_Toc407328543)

[4.1 Intel SSE指令集介绍 21](#_Toc407328544)

[4.2 已有的应用 24](#_Toc407328545)

[4.2.1 位并行 24](#_Toc407328546)

[4.2.2 块字符 24](#_Toc407328547)

[4.2.3 Four-Russians 25](#_Toc407328548)

[第5章 改进算法IEPSM详述 26](#_Toc407328549)

[5.1 综述 26](#_Toc407328550)

[5.1.1 wscmp(a,b)字长比较指令 26](#_Toc407328551)

[5.1.2 wsmatch(a,b)字长匹配指令 27](#_Toc407328552)

[5.1.3 wsblend(a,b)字长翻转指令 28](#_Toc407328553)

[5.1.4 wscrc(a)字长CRC码计算指令 29](#_Toc407328554)

[5.2 模式串串长m大于等于16 29](#_Toc407328555)

[5.3 模式串串长m大于等于4且小于16 31](#_Toc407328556)

[第6章 实验结果与分析 33](#_Toc407328557)

[6.1 测试算法选取 33](#_Toc407328558)

[6.2 测试集选取 33](#_Toc407328559)

[6.3 参数选取实验 33](#_Toc407328560)

[6.4 算法对比实验 35](#_Toc407328561)

[6.4.1 算法效率 35](#_Toc407328562)

[6.4.2 算法灵活性 35](#_Toc407328563)

[结论 37](#_Toc407328564)

[参考文献 38](#_Toc407328565)

[攻读学位期间发表论文与研究成果清单 44](#_Toc407328566)

[致谢 45](#_Toc407328567)

**第1章 绪论**

**1.1 课题研究背景及意义**

**1.1.1 课题的研究背景**

进入二十一世纪以来，生命科学和生物技术伴随着人类基因组计划的成功实施得到了迅猛发展，基因组学相关数据也呈现指数式增长。尤其随着新一代测序技术的改进，测序成本不断降低，测序速度进一步提高，测序平台单日通量达到数亿序列。相比于2000年，2010年基因组数据的测序规模增长了6个数量级[1]，整个基因组数据产量呈现爆炸式增长。2012年2月份，纳米级测序仪GridION 和 MinION的研制成功[2]为生物信息学测序领域带来了一次技术革新，这两台测序仪能够完成超长序列（100kb以上）的测序，进一步提高了测序通量、降低了测序成本。当前，生物信息领域的测序数据产出速度远远超过了经典的摩尔定律[3]，尤其新世纪人类基因组项目（Human Genome Project，HGP）顺利开展以来，海量基因序列数据添加到生物学数据库中，生物信息序列相关数据的迅速扩张给串匹配算法在数据敏感度和计算速度方面提出了更高的要求。大型的生物学数据库包含着丰富的信息，如何充分利用这些数据来挖掘有价值的信息，不但是生物学领域更是数学领域和计算机领域的研究者所共同面临的挑战性问题[4]。

串匹配算法是生物信息学中的一个基础算法，是生物学数据分析计算的基石。生物序列数据库是生物学数据分析的对象。截止今天，世界已有的生物序列库包括了生物学研究的所有方向，如文献数据库、蛋白质片段、蛋白质序列、蛋白质三维结构、核酸序列等数据库。而现在世界上最重要的几个的蛋白质、核苷酸数据库为：欧洲生物信息学研究所（EMBL-EuropeanBioinformatics Institute）的核苷酸序列数据库EMBL（<http://www.ebi.ac.uk/embl>），美国国立生物技术信息中（National Center of Biotechnology Information）的GenBank数据库（<http://www.ncbi.nlm.nih.gov>）和日本信息生物学中心（Center for Information Biology）的DNA数据库DDBJ（<http://www.ddbj.nig.ac.uk/embl>）。现有库总共至少510个，而和分子生物学有联系的库就有337个。每个库都存储着巨大的生物信息，如存储在GenBank系统中的基因数据总量已经达到32549400条序列，超过37893844733亿碱基对[6]。

因此，在测序技术的不断升级的情况下，获取生物基因序列变得越来越容易和便宜，因此出现了巨大的基因数据。在这种情况下，海量基因序列匹配查询的巨大挑战给字符串匹配算法的性能提出了更高的要求。

**1.1.2 课题的研究意义**

字符串匹配算法是基因序列和蛋白质序列分析研究的基础算法。所以找出一个适用于基因序列且更有效率的算法显得十分必要。

**1.2 针对生物学的字符串匹配算法的国内外研究现状及发展趋势**

**1.2.1 相关研究工作**

从1970年到现在，如何高效地匹配字符串就一直是计算机科学研究里的热点问题，大量学者在这一领域取得了很多新的理论和实验性的成果，其中很经典的串匹配算法主要有Knuth-Morris-Pratt算法、Boyer-Moore算法、Sunday算法和KR算法，早期的很多算法均是改进自以上这些算法。

* KMP的改进算法

文献[7]在串匹配中引入并行处理的思想，提高了主串匹配速度。

因为有种情况下，T串和P串由于存在重复字符，所以KMP算法需进行多次重复比较的缺陷，文献[8]进行了创新，当P遇到不匹配时，算法会依据已有的匹配词去运算window的移动长度，把比较位置移到P串开始位置后进行匹配，而非马上将P串的比较位置往后移，提高了算法效率。

在P串的初始化步骤里，文献[9]基于传统算法，提出在P串中对两边分别求值，根据P串当前字符的值，提高移动长度。

* BM的改进算法

在处理随机串的过程中，即字符串是由字母表随机生成的，文献[10]基于BM算法做了一点创新。该算法通过使用BMH和BMHS算法的方法去判定P串的移动距离，并利用了P串首尾的特性，很好地降低了匹配过程中的字符比较次数。

文献[11]使用字符串中坏字符、尾字符和尾字符的下一个字符的关系去提高P串的移动长度。即从后向前扫描去查找最长后缀串的出现，然后依据该后缀的对应的值来向后移动。

通过将好后缀移动和坏字符移动合并进行处理，文献[12]采用P串子串去判定移动距离。只有匹配后才对P串的剩余部分调用全匹配；若这时存在不匹配的情况，就把window向前移动。也就是说当发生上述情况时，移动子串长度；其他情况则移动大于子串长度。

文献[13]对BM的改进类似于Sunday算法，该算法采用最长前缀的思想，在后缀匹配时使用一个转移数组shift。window的移动长度由T子串最后一个字符的后继字符判定。若此字符不属于P串，则将window移动至其后继字符处。这是为了减少因window内存在不匹配而导致接下来的匹配的重复匹配问题。而该算法仅当匹配成功时才对P串的其他字符进行匹配，若匹配不成功，将P串移动移动一个字段。同时算法使用转移数组去计算P串的移动值。

文献[14]的改进点在于没有采用常见的由前往后的窗口匹配顺序，而是由两端至中间。其匹配方法为：从模式串最末位置开始扫描，若失败则将匹配窗口向后移动，由shift函数得到下一步的位置；若匹配上了就从开始位置匹配，如果开始位置匹配失败，则继续比较P串最后一个字符的后继字符，若该字符不在字符表内，则P串向后移m+l；若开始位置对上了，则继续扫描最后一个字符的上一字符并通过一定的规则向中间移动，反复执行该过程直到匹配完成。

~~文献[15]KMP模式匹配算法在最差情况下的执行时间是线性的，并且和字母表大小以及模式串长度无关，BM算法具有接近最优的平均效率和最好效率，并且实际运行速度也非常快；设计一个包含了KMP和BM主体思想(受到Sunday算法的启发)，结合两者优点的混合算法。~~

* Sunday的改进算法

文献[16]在每次匹配失败时通过特殊位置上字符的启发来获得向后移动的距离，首先检查window中子串最后一个字符在P串字符表中是否存在，若否，则直接将window右移m+l位，降低多余的匹配次数。当T串的字符表较大时，而P串的字符表较小时，T串中不存在P串的可能性很高。因此该算法通过一些规则尽可能地跳过不必要比较的字符，减少了匹配次数，提升算法效率。而在应用中T串出现P串不存在的字符的可能性比较高，因此该算法在实际应用中表现不错。

文献[17]在P串中得到长度最大的真首子串p1(p1<P)，并将它做为新的模式串和T串进行比较，当两串对不上时，该算法能移动相当远的长度；而当能匹配后，调用全比较函数去比较剩余的模式串。当剩余的模式串匹配不成功时，指针移动到未成功匹配的位置，进行下一轮比较；而匹配成功后，则该轮比较完成。

文献[18]提出的算基于CPU并行处理特性和QS算法的特性，即CPU单位时间内处理机器字长的数值和处理单字符的时长相等以及QS算法对接下来匹配的顺序和失败位置不敏感的特性。该算法把单字符比较通过强类型转换变成数值间比较，比如ulong就能容纳8个字符，导致匹配次数降低8倍，提高算法性能。

文献[19]主要考虑字符在P串中出现的可能性差不多，那么QS算法的比较顺序选择就没有太大关系，因为这种情况下都会存在无效匹配，降低性能。该算法改变了window的匹配方向，即改为由两边至中间，按左右顺序往中间聚合，反复执行改过直至结束。并在这一期间统计P串字符出现的次数，在每一次的匹配过程中从按字符出现概率排序顺序去执行匹配，当全部字符匹配完成后结束，提高了算法性能。

文献[20]提出了新的移动判定方法，即利用2个字符去判定P串匹配与否。在每一次匹配时这2个字符采用window内P串最后一个字符对应T子串中的某一字符和该字符的后继字符，一旦发生不匹配，window跳转的长度采用这两者跳转距离最大的。

文献[21]发现，当不匹配的情况发生时，window的跳转长度和window内的最后一个字符的后继字符有关。如果后继字符在P串内，那么把P串中最右边的同一字符和后继字符排好；如果后继字符不在P串内，那么越过这一字符。反复执行该过程，直到找到这一字符，再继续跳转window。

文献[22]主要通过改变字符表的大小去提高串匹配效率，即将原字符表通过某种方式改为较小的新字符表，并把这种想法和采用Four-Russian的BM算法相结合。实际执行后发现，通过减小字符表的大小能提高算法的效率，特别是当字符表大小降低很多时，新算法在匹配英文文本时效率很高。

文献[23]针对Boyer-Moore-Horspool算法提出了一个改进算法，该算法没有按照一个个地去比较P串和T串的字符，而是直接将其组合成CPU字长来进行比较。这样可以显著改善算法性能，特别是P串较长的情况改进更为明显。

但专门针对生物序列的串匹配算法并不多，大部分算法还是通用的算法，最新的算法在向并行串匹配研究方向发展。

如文献[24]采用先分组再并行处理的串匹配算法。该算法首先证明最优分组定理，并使用两种最优分组方法，即最短路径和动态规划。该算法在P串长短不一时能工作很好，对于改进多模式串匹配算法是一个很好的思路。

文献[25]采用Band-Row的思想去压缩SNORT中的AC算法，使得算法性能得到大幅度改善。这种表示法能够得到的状态结点转换速度，只用多执行两次边界检查，但是当一行中的非空元素个数大于3时，算法压缩的性能不显著。同时，尽管多执行的检查次数有限，但还是对效率有一定影响。

文献[26]采用位图压缩（Bitmap Compression）和路径压缩（Path Compression）两种策略来压缩存储大小，让其适应于硬件环境。该算法通过一系列的实验，我们可以发现在ASIC等硬件上实现AC算法，算法性能有很大改善，但软件版的该算法并不易实现。

从硬件并行和体系结构的角度，去开发串匹配算法的特定硬件也是最近研究的一个方向。文献[27]开发出使用TCAM去实现并行化串匹配算法的硬件，算法性能有飞跃式的提高。文献[28]从硬件并行的角度改进了串匹配算法，它通过将AC自动机分组，然后并行地去进行处理，最后合并这些匹配结果。上述改进都是在利用可编程逻辑处理器的特性。但是，由于可编程逻辑器件在存储和功能上的限制，传统算法里的并行方法不易应用到这些器件上。

串匹配算法的另一发展方向是利用GPU的并行计算框架来处理。因为CPU的发展已接近极限，但GPU却在最近十年获得巨大的发展，其发展速度自1999年以来是CPU的三倍。它的浮点运算和并行计算对计算密集型的科学计算有很大的帮助，并逐渐发展为科学计算的的一个新的环境 [29]。直至现在已有许多算法使用在GPU上实现了一遍，诸如矩阵相乘等基础算法。如CMATH算法是在GPU上使用CUDA包去实现的单模式串匹配算法，其运行效率有很大提高。

**1.2.2 存在的问题**

大部分算法还是沿用常用的字符串处理思路，逐个读入字符，未考虑到DNA字符集大小问题和DNA字符出现特征。因为基因数据的字典大小仅为4，所以如果按QS之类的算法每次只读下一个字符，就不能实现很好的跳转。另外许多算法没有考虑串长对算法的影响，多数情况都是一个算法应对任意长度模式串。最后一点，尽管有些算法已经开始考虑利用CPU和GPU的硬件指令去加速算法，但这些改进很多只是将原来的经典算法换成硬件并行版本，并且有些改进算法也忽视了编程实现上的一点，大部分使用跳转表，通过读表得到不同值再去跳转，实验表明，常值跳转在程序实现上能达到极好的性能，以一个较小的常值跳转在多数情况下会比较大的变值跳转快。比如作者最早开始的算法改进是生成一个跳转数组，该数组能根据读入的串选择跳转距离，在多数情况下跳转距离都能达到模式串串长，极少几种情况会很短。但后来经实验发现，这种跳转距离的优化还没有一个安全的常值距离跳转来得快。

**1.2.3 发展趋势**

以后的串匹配算法会更多地利用SIMD指令或者GPU指令，使用硬件指令实现多字符一次性处理，并根据程序优化实践去设计算法。

## 1.3 本文的目标与组织结构

本文第二章根据分类描述了现有的一些单模式精确串匹配算法。在第三章和第四章分别介绍了核酸、蛋白质和SSE指令的相关知识。第五章详述新算法iepsm。第六章是新算法与其他算法的比较实验并给出了结论。

作者的主要工作是设计出一款针对生物基因序列特性的单模式串匹配算法，使其的运行效率在比较算法中最好。

**第2章 精确字符串匹配算法综述**

一条字符串是定义在有限字符表Σ上的字符序列[30]。例如，GTACTGGTAATG是字符表Σ={G,A,T,C}上的一个字符序列。单模式精确串匹配问题即在长字符序列T中检索出给定字符序列P的全部出位置。其中，T称为文本串，P称为模式串，T和P都由相同的字符表Σ产生。假如存在字符串由a，b和c组成，则我们定义字符a是ab的一个前缀，c是bc的一个后缀，a是abc的一个因子。

字符串匹配算法可以通过字符串扫描策略的不同而分为三种基本的方法。

一是从文本中一个个地按顺序扫描字符，每扫描一个字符就改变初始化阶段的值，并且判断串匹配出现与否。其中的经典算法就是Knuth-Morris-Pratt算法，而更高效的算法是Shift-Or算法，该算法改进后还能适用于长P串匹配[30]。

二是基于滑动窗口。滑动窗口即为一搜索窗口，其长度与模式串长度相等。搜索窗口沿着文本从左向右滑动，搜索模式串的过程在窗口内进行。Window首先在文本上对齐，再在T根据算法判定方向移动，这样可以通过改变window的移动距离来检查所有可能的配对情况，然后在window内由后往前检索其中文本T和P串的公共后缀。Boyer-Moore算法就使用了这种方法。但BM算法只是理论较优，由于好坏规则计算的复杂，导致BM的简化算法Horspool算法比它要快[30]。

三是子串搜索方法，该方法是最后一个被提出。在实际应用中若P串较长足够长，则采用这种方法性能提高很大。该方法也使用了滑动窗口，window中的搜索顺序是由后往前。区别在于，该方法只关注window内T子串的最长后缀串，同时这一子串也正好P串的一个因子。BDM算法是应用该方法的算法之一，但该算法的灵活性不足，于是产生了两个改进算法，短串处理中，BNDM算法简单高效，长串处理中，BOM则更有效率[30]。

因此由上所述，我们可以将单模式串匹配算法分为三种类型来描述并简要介绍对于类型的代表算法。

前缀搜索在window内从前向后（沿着文本的顺序方向）一个个扫描文本字符，搜索window中文本T和P串的最长公共前缀。

后缀搜索在window内从后向前（沿着文本的反方向）一个个扫描文本字符，搜索window中文本T和P串的最长公共后缀。使用该方法的算法可以通过比较后几个字符来越过一些多余的串比较，并获得亚线性的平均时间复杂度。

子串搜索在window内从后向前一个个扫描文本字符，若满足如下条件的最长字符串u：u既是窗口中文本T的后缀，也是P串的子串。和后缀搜索方法类似，采用这种搜索方法的算法也具有亚线性的平均时间复杂度，在某些情况下还可以获得最优的时间复杂度。

## 2.1 基于前缀搜索的算法

即在window内从前向后(沿着文本的顺序方向)一个个扫描文本字符，搜索window中文本T和P串的最长公共前缀。采取该方法可以越过一些字符，从而得到亚线性时间复杂度。

### 2.1.1 Knuth-Morris-Pratt算法

Knuth-Morris-Pratt算法（简称KMP算法）是由D.E.Knuth、J.H.Morris和V.R.Pratt共同设计出的极为高效的精确单模式串匹配算法，该算法采用前缀匹配策略，和一般的暴力搜索BF算法相比，其优势在于它能根据读入字符去改变下次指针的跳转长度，这样一般能获取大于1的跳转长度，提高了算法效率。图2.1通过一个实例来演示暴力搜索算法和KMP算法的不同，由上文可知这二者唯一的区别在于P串跳转长度[31]。



图2.1 KMP算法P串跳转样例图

上图中，当BF算法扫描到不匹配的字符时，就向右跳转一个字符，而KMP算法通过读入的字符再去运算已有的前缀，计算出下一次的跳转长度为2个字符，也就是说每次读入T串的一个字符，KMP算法就能计算出下一次应该将指针移动多远的距离，如图2.1所示，当读入T串的第i+1个字符a时，KMP算法通过计算得知需将指针移动到k位置。因为，这时T串的子串是T[i-k, i]=aba，而P串的子串是P[0,k]=aba，后缀和前缀相等。故将P串跳跃到k，然后算法接着去匹配T串的后继字符和P串前缀的后继字符。

同时我们可以发现，P串和T串分别在j和i上对上了，即T[i-k, i] = P[j-k, j] = aba。另外由上文T[i-k, i] = P[0, k] = aba，可推知P[0, k] = P[j-k, j] = aba。并且易知，该算法即是在P串中寻找一最长公共前缀，可以让[j-k, j] = [0,k]。

所以我们能定义一个跳转表，当jump[j] = k，即表示P[0, k] == P[j-k, j] 的最大k值。也就是说，若P串在j+1位置对不上T串在i+1位置，就移动至P串k+1位置再去比较。假设存在上述的jump表，易知KMP算法如表2.1所示。

表2.1 KMP算法

|  |  |
| --- | --- |
| **KMP算法** | |
| 01  02  03  04  05  06  07  08 | J ← 0  For i ← 1 to n do  While (j > 0) and (P[j+1]!=T[i])  Do j ← jump[j]  If P[j+1] = T[i]  J ← j+1  If j == m  Report pattern occur, i-m |

KMP算法中跳转表我们使用数学归纳法来生成，第一步令jump[1] = 0，假定jump[j]=k，即P[0, k] == P[j-k, k]，若P[j+1] == P[k+1]，则可知[0,k+1] = P[j-k, j+1]，从而更加定义得出jump[j+1] = k+1；

如果P[j+1] != P[k+1]，那就接着比较P[j+1] 和P[k1+1]是否相等，其中（jump[k] = k1），根据（jump[k] = k1）的定义，P[0,k1] == P[k-k1, k]，根据（jump[j] = k）的定义，P[0, k] == P[j-k, k]，根据这两个等式，推出P[0, k1] == P[j-k1, j]，如果此时P[j+1] == P[k1+1]，则得出：jump[j+1] = K1 +1 = jump[k] +1。

如果P[j+1] != P[K1+1]，继续递归比较P[j+1]和P[jump[jump[k]]+1]至P[1]；

如果依次比较都不相等，那么jump[j+1] = 0。其初始化算法如表2.2所示。

表2.2 Jump数组初始化

|  |  |
| --- | --- |
| **Jump数组初始化** | |
| 01  02  03  04  05  06  07 | Jump[1] ← 0; j ← 0;  For i ← 2 to m do  While (j > 0) and (P[j+1]!=T[i])  Do j ← jump[j]  If P[j+1] = T[i]  J ← j+1  Jump[i] ← j |

若遇到最坏的情况，即P串不属于T串，那么BF算法的最坏时间复杂度为O(nm)，最好时间复杂度为O(n),这里n为T串的大小，m为P串的大小。KMP算法最坏时间复杂度为O(n)，最好时间复杂度为O(n/m)。

### 2.1.2 Shift-And/Shift-Or算法

BF算法实际上是枚举出所有可能，所有性能不高，导致所有T串中的字符都在算法流程中做了m次匹配。

在1990年，出现了一种使用位并行（Bit-Parallelism）的方法去处理串匹配问题的做法，并且在这个思想的指导下迅速诞生了shift-or算法，该算法的改善版本为shift-and算法，shift-and算法又称为BAP（Bit-ParallelAutomaton）算法。

最早的位并行算法主要还是用位并行思想去重新实现已有的串匹配算法。在匹配过程中，采用组合字符并行处理的手段，能提高原算法的效率。该类算法的最佳适用场景是P串小于等于机器字长的时候。

位并行即是考虑到CPU处理一个机器字长的数值和一个字符所耗时间相等得事实，所以将多个字符组合到长度为ω的机器字内，如ulong就能组8个字符，组完后通过一次CPU运算即可得到相应结果。采用位并行，原算法的时间复杂度能降低ω（机器字长的位数）倍，获取巨大的性能提升。

考虑到现在的CPU基本都是64Bit的，并且其中每一位都能保存相应的匹配结果，所以我们能在单词比较中完成对64个字符的比较，BF算法采取这种改进方式能提高64倍的运行速度。

Shift-And算法思想：设模式字符串为P，文本串为T。其改进点就是加入了一个字符串集合D，该集合通过读入的字符不断更新最长前缀串，并且保证集合中的元素同时是window内T串的后缀和P串的前缀。当window每扫描一个字符时，集合D就会使用公式更新一次[32]。

我们可以这么具体理解。

* 设P长度为m，则集合D可表示为D = dm…d1 而用D[j]代表dj；
* 定义D[j]=1，当且仅当p1…pj 是 t1…ti 的某个后缀；
* 当D[m]=1时，就认为P能在T中匹配；
* 每扫描后继字符 ，套用公式更新D′。

当且仅当D[j]=1并且等于时D'[j+1]=1。这是因为D[j]=1时有 …是 … 的一个后缀，而当等于可推出…是… 的一个后缀。这个集合可通过位运算来更新。

该算法对集合D的更新规则如下，即先定义一个数组A，其中A的大小是文本串字符集的大小（比如英文文本的数组A大小是26），若P的第j位等于c(P[j]=c)，则将A[c]中第j位置为1，否则为0。

而D首先初始化为，逐个扫描T串中的字符，然后用下列公式更新集合。

D ((D << 1) | 0m-11) & A[ti+1]

易知，通过向左移位我们可以把D′的第i+1位上的值挪到D的第i位上。同时由于空字符串也是T的后缀，故左移操作后将结果与上0m-11。并且为了对上满足=的情况，故最后将结果与上A[]。

该算法在P串小于等于机器字长工作最好，上述的步骤均能在O(1)内结束，所以shift-and算法的时间复杂度为O(n)。

## 2.2 基于后缀搜索的算法

基于后缀搜索的难点在于不漏扫任何可能匹配的情况下，寻找合适的窗口跳转方法。

### 2.2.1 BM算法

Boyer-Moore为了做到这点定义了两个规则：坏字符和好字符规则[31]。

* 坏字符规则

1. 若是坏字符不在P串中，那么把P串指针指向坏字符的后继字符位置。如图2.2所示，由于P串中没有出现字符c，那么把指针指向c的后继字符处。



图2.2 坏字符未出现在模式字符中

1. 若是坏字符在P串中，那么把P串指针指向最接近好后缀的坏字符和T串的坏字符的位置。如图2.3所示。

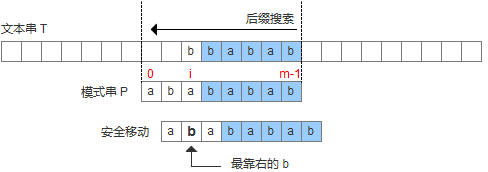


图2.3 坏字符出现在模式串中

* 好字符规则

第一，P串若存在某一子串，其含有好后缀，则跳转P串，将指针移至这一子串和后缀位置，若存在多个满足条件的子串，那么把最右的子串与之对齐。如图2.4所示。

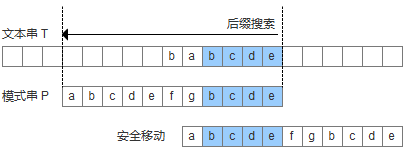


图2.4 存在子串匹配好后缀

第二，P串中不存在含有好后缀的子串，那么应找到该P串的最长前缀，使其与好后缀的后缀相同，发现这一子串后，将其与好后缀对齐匹配。如图2.5所示。

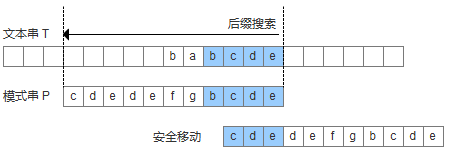


图2.5 没有子串匹配好后缀

第三，若P串都不满足上述条件，就可以越过模式串串长，将指针指向好后缀的后继字符。如图2.6所示。

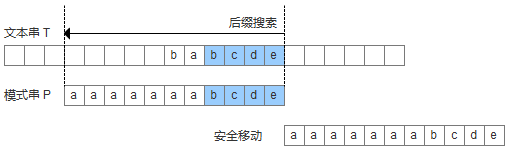


图2.6 没有子串匹配好后缀且没有最长后缀

BM算法的算法描述可以查询相应文档，这里不再给出。

BM算法是早期应用比较广泛的高效串匹配算法之一。该算法使用由后向前的字符读取顺序，同时使用好字符表和坏字符表去计算最大跳转长度。BM算法的空间复杂度为，初始化部分的时间复杂度为，在某些环境下BM算法的最佳时间复杂度可达

### 2.2.2 QS算法

QS算法是由BM算法改进而来，它通过简化Horspool算法，获得了很好的效率提升，因此又称为BMHS算法。

QS算法与BM算法采用一样的处理方式，但是QS的算法效率更胜一筹。它没有采用常见的窗口内字符读取方式，而是创造性地去读取T串的后继字符，即当发现不匹配时，window的跳转长度由window内子串的最后一个字符的后继字符判定。如果该字符在P串内，那么找到P串中最右端的同一字符并与之对齐；如果该字符不在P串内，那么越过这一字符，把window挪至该字符的后继字符处。由于当window中发生匹配失败时，当前指针需要越过失败，这样可以减少大量不必要的匹配。这个算法的关键点：

* 为P串初始化查找表，来加快字符定位的效率。
* 窗口内按由后向前的顺序比较，减少匹配次数。

QS算法在最坏时间复杂度为Ο(mn)，但考虑到该算法在每一轮比较前会先检查window内的最后一个字符，所以适用于由大字符表组成的字符串中。

## 2.3 基于子串搜索的算法

若P串很长，那么采用基于子串的搜索算法是相当有效的。类似基于后缀的搜索算法，子串搜索同样采用移动窗口，并在窗口中由后往前读取字符。差别在于，该子串同样是P串的一个因子。最早采用这一思想的是 BDM 算法，当 P串长非常小时，该算法的改进版本BNDM可以很好处理这种情况。而当串长很大时，改进算法BOM 也能获得不错的效率。另外在BDM和 BOM这两者之上改进的多模式串匹配算法为 SBOM算法 [33]。

### 2.3.1 BOM算法

BOM 算法采用确定性的有向无环自动机去匹配子串，该算法可以搜索出window内与P串子串相等的最大后缀串，且非反向搜索P串。若P串和window不匹配，那么通过最大后缀串去跳转window，进行下一轮比较 [34]。

而Factor Oracle结构在这通过数组实现，即定义一个（m+1）\*B的数组，在这里B是P串的字符表大小。该方法在短串的情况下可以在的时间复杂度里运行，完成状态转移，提高算法效率。但在长串情况下，就要使用的时间复杂度了。

BOM算法的最差时间复杂度为，平均时间复杂度为。

### 2.3.2 SBDM算法和SBOM算法

SBDM 算法在大小是 min的window内采用后缀自动机由后往前去搜索P串的子串，再根据子串去判定window跳转长度。同时该后缀自动机在全部长度等于 min的P串前缀反转串上，初始化阶段的时间复杂度是Ο(r× min)。匹配过程在长度是 min的window中展开，同时指针向T串右移。在每一轮的匹配window里，算法由后往前搜索T串的最长后缀串，也就是P中大小为 min的前缀的子串。下面探讨匹配过程中的如下情况[33]。

* 不能搜索出子串，也就是说自动机不能搜索出文本字符σ。那么说明不存在一P串前缀使得其可以包含window中的字符。所以，我们将指针跳转至字符σ后。
* 而当同时满足指针已指向window的开始处且当前自动机状态为x时，那么说明算法已搜索到F(x)中的一个前缀字符串L(x)，所以把F(x)里的任一P串和window进行匹配看是否能匹配上。最后，我们把指针往右移一个字符，进行下一轮的处理。

SBDM的最差时间复杂度是Ο(np)。但是，在字符表大小适中的情况下，该算法的时间复杂度是亚线性的。在编程实现里，SBDM初始化后缀自动机的时间复杂度较高。所以若P串集合规模较大时，其初始化所占时间和匹配时间比就不能忽视掉。同时当P串集合变大时，后缀自动机也需申请较大的内存空间。考虑到这一缺陷，SBOM算法通过简化实现后缀自动机，在不改变匹配算法的情况下做出了改进。实验结果表明，SBOM比SBDM在长短串的情况都更有效率。

SBOM算法也采用Factor Oracle结构。其实现的Factor Oracle自动机匹配的字符串集合是P串集合的超集。其匹配方法和SBDM差不多，通过Factor Oracle自动机，在大小是min的window中由后往前进行字符匹配，以此完成指针跳转。若不能匹配字符σ，则把window跳转至σ后；若window中的字符全匹配后，且已指向window的开始字符处，则把P的自己和T做匹配判定。

SBOM算法的最差时间复杂度是，平均时间复杂度是亚线性的。改算法中自动机的初始化阶段效率很高，同时对存储空间的需求很小，所以对文本大小具有很高的灵活性。

**第3章 生物信息学基本概念**

生物信息学的主要任务是运用应用数学、信息学、统计学和计算机科学的方法来分析、处理和研究基因序列和蛋白质序列数据中所包含的各种生物学信息，主要研究发掘生物数据的内在信息。下面简要介绍这两种序列和高通量测序结果文件格式。

## 3.1 核酸

核酸是一种主要位于细胞核内的生物大分子，通过多种核苷酸聚合成的生物大分子化合物，对生物的生长、遗传、变异等现象起着重要的作用。核酸有许多功效，比如与蛋白质合成核蛋白。而通过不同的化学分子和不同的碱基组合方式，可以形成各种不同的核酸。根据化学组成不同，核酸可分为核糖核酸（简称RNA）和脱氧核糖核酸（简称DNA）。DNA是储存、复制和传递遗传信息的主要物质基础，引导生物发育与生命机能运作。RNA主要用于合成蛋白质。其中转运核糖核酸，简称tRNA，起着携带和转移活化氨基酸的作用；信使核糖核酸，简称mRNA，是合成蛋白质的模板；核糖体的核糖核酸，简称rRNA，是细胞合成蛋白质的主要场所[35]。

核酸是一种一维高分子链，链中包含4种单体，每个单体叫做核苷酸。核酸中携带着遗传信息，遗传信息主要表现在核苷酸的排列次序上。根据核苷酸类型的不同，核酸分为脱氧核糖核酸（DNA）和核糖核酸（RNA）。核苷酸由磷酸、脱氧核糖或核糖和碱基组成。构成核苷酸的碱基分为嘌呤和嘧啶两大类。前者主要指腺嘌呤（adenine，A）和鸟嘌呤（guanine，G），DNA和RNA中均含有这两种碱基。后者主要指胞嘧啶（cytosine，C）、胸腺嘧啶（thymine，T）和尿嘧啶（uracil，U），胞嘧啶存在于DNA和RNA中，胸腺嘧啶只存在于DNA中，尿嘧啶则只存在于RNA中。其中，DNA是存储、复制和传递遗传信息的主要物质基础，RNA在蛋白质合成过程中起着重要作用。

## 3.2 蛋白质

蛋白质是一种由氨基酸分子组成的有机化合物，是构成生物体的直接元素，是氨基酸脱水缩合形成的肽链。它是生命活动的主要承担者，参与细胞生命活动的每个过程，在生物物质结构和功能上发挥作用，约占体重的20%内，是生物的重要组成部分。而其中的氨基酸是蛋白质的基本组成单位[36]。它是含有一个碱性氨基和一个酸性羧基的有机化合物，由于结构的差别形成了各种不同的氨基酸，而这些氨基酸又根据一定规则组合形成了不同功效的蛋白质。生物体内含有多种蛋白质，并具有各种作用。而这些蛋白质均通过二十种氨基酸由肽键连接而成，并在体内不断进行代谢与更新。这二十种氨基酸是蛋白质的基本组成单位，造就了蛋白质性态各异的性质和作用。表3.1列出了这二十氨基酸的名称、字母表示和符号。

表3.1 蛋白质的氨基酸名称及字母符号

|  |  |  |
| --- | --- | --- |
| **名称** | **三字符号** | **单字符号** |
| 丙氨酸 | Ala | A |
| 精氨酸 | Arg | R |
| 天冬氨酸 | Asp | D |
| 半胱氨酸 | Cys | C |
| 谷氨酰胺 | Gln | Q |
| 谷氨酸 | Glu | E |
| 组氨酸 | His | H |
| 异亮氨酸 | Ile | I |
| 甘氨酸 | Gly | G |
| 天冬酰胺 | Asn | N |
| 亮氨酸 | Leu | L |
| 赖氨酸 | Lys | K |
| 甲硫氨酸 | Met | M |
| 苯丙氨酸 | Phe | F |
| 脯氨酸 | Pro | P |
| 丝氨酸 | Ser | S |
| 苏氨酸 | Thr | T |
| 色氨酸 | Trp | W |
| 酪氨酸 | Tyr | Y |
| 缬氨酸 | Val | V |

## 3.3 高通量测序结果文件

### 3.3.1 FASTA格式

在生物信息学中，FASTA格式（即Pearson格式），是一种基于文本用于表示核苷酸序列或氨基酸序列的格式。该格式采取单字符表示碱基和氨基酸，同时在每一行前写上注释等信息[37]。

在这类格式中，首行用大于号">"或分号";"来指明序列的开始位置，并在这行记录一些相关信息。接着便是序列信息，其中序列的组成元素是由上面介绍的核苷酸和氨基酸的字母，一般只有氨基酸必须用大写，而核苷酸不做要求，所以读取这些数据时，首先需确定大小写规定。单行的字符个数应小于80个。

在核酸序列中，刨去最常见的几种字符A、C、G、T、U外，还有如下几种：R代表G或A（嘌呤）；Y代表T或C（嘧啶）；K代表G或T（带酮基）；M代表A或C（带氨基）；S代表G 或C（强）；W代表A或T（弱）；B代表G、T或C；D代表G、A或T；H代表A、C或T；V代表G、C或A；N代表A、G、C、T中任意一种。

采用FASTA的数据库一般存储采集的原始数据，并根据实际添加相应的注释。业界一般开发多种软件去检索这些信息[38]。因此尽管这种格式计算机处理起来不是最高效的，但由于其存储的是原始信息，我们可以很方便在这之上进行信息筛选加工，并且也是人可以阅读的，所以这种格式有着极大的通用性。

### 3.3.2 FASTQ格式

FASTQ格式是另一种经常使用的序列文本格式，它记录了生物信息（一般是核酸序列）和对这些信息进行质量评价以及相应的质量评价。

该格式采用ASCII编码，是如今高通量测序的首选。NCBI Short Read Archive基于该格式做了扩展，加上了其他的信息。

通常FASTQ格式的序列由四行组成，类似于FASTA格式，首行开头是"@"，然后是其相关的补充信息。接着是序列信息，第三行开头是"+"，这一行也能加上部分补充描述。最后一行是对序列测序的质量评价，且行长和序列长度一样[39]。

### 3.3.3 数据样例

一个FASTQ数据的典型格式如下。

@SEQ\_ID

GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT

+

!''\*((((\*\*\*+))%%%++)(%%%%).1\*\*\*-+\*''))\*\*55CCF>>>>>>CCCCCCC65

这里首行的起始字符@记录数据的描述信息，接着是原始序列信息，第三行只有一个+号，其后能加上相关补充信息，最后是对测序质量的评价。

而一个典型的FASTA数据格式如下。

>SEQ\_ID

GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT

可以看出该格式的描述信息是以>开头，不含数据质量信息。

**第4章 SIMD指令集在串匹配算法中的应用**

## 4.1 Intel SSE指令集介绍

单指令流多数据流（英语：Single Instruction Multiple Data，缩写：SIMD）是一种采用一个控制器来控制多个处理器，并对一组数据（又称“数据矢量”）中的每一个分别执行相同的操作从而实现空间上的并行性的技术[40]。

在CPU中，单指令流多数据流技术实现为一个控制器控制多个平行的处理单元，常见的实现有Intel的MMX或SSE，和AMD的3D Now!指令集。

在GPU也开发了类似的处理单元，利用其可编程流水线在处理单指令流多数据流时，有很高的计算效率。开源的开发语言有OpenCL和CUDA。

本文将应用Intel SSE指令集。

SSE(Streaming SIMD Extensions)指令集是MMX指令集的后继版本，发布于AMD的3D Now！指令集后，最初是在Pentium III的CPU上实现的。该指令集包括 70 条新指令[41]，而AMD则在Athlon XP里才引入该指令集。

该指令集的最初作用是为改善流媒体的运算效率而引入的，其后续版本有SSE、SSE2、SSE3、SSE4，这些指令集使用了一系列处理器优化技术，把需要多步执行的指令组合成一条硬件指令，在较短的CPU执行周期内完成，这样大大改善了计算性能。SSE系列指令集引入128位寄存器，可以将多个数据组成128位长度的数据进行并行计算。一般来说，SSE指令的运算性能是以前指令的四倍。从该指令集推出到现在，其已应用在3D游戏、加密运算、科学计算、图像计算、流数据处理等多个领域[42]。考虑到SSE指令集在并行计算上的巨大优势，本文中我们使用SSE指令去解决串匹配的问题，同时利用该指令集中的128位寄存器去优化算法。

该指令集的扩展版本简介如下。

SSE2指令集是Intel最早在Pentium 4型号的CPU上实现的，而AMD则是在Opteron 和Athlon 64型号CPU里才引入该指令集。由于该指令集引入64位双精度浮点数运算和整型数据运算，并加入CPUcache相关指令，改进了cache延迟。AMD对其的实现是加入多个XMM寄存器，但要改到64位下才能使用该指令集。而Intel直至到推出Intel 64时才引入x86-64的相关指令。

SSE3指令集是Intel于2004年在Prescott的奔腾4型号的CPU上实现的[43]。该指令集同时也是AMD在Athlon 64型号CPU上实现的第五个版本，另外在Venice中同样也引入了该指令集。SSE3指令数不多，才13条，有寄存器的高低位间加减计算，浮点运算，通过对水平式寄存器整型的指令，能一次计算多个数值的加减法，极大地提高了CPU的DSP和3D计算性能。同时，该指令集优化了多线程程序指令执行效率，改进了CPU中的超线程运行性能。

SSSE3指令集是Intel在2006年在酷睿型号的CPU上实现的，该指令集新加入32条指令，是SSE3指令集的补充，它使得处理器在图像视频和网页上的计算能力得到更大提高。而Intel最早是想将该指令集其加入至SSE4指令集中，而由于摩尔定律对处理器性能的极大提高，所以提前发布了出来，并加入在当时的酷睿处理器中。

SSE4指令集是Intel在2007年在酷睿双核型号CPU中实现的，SSE4指令集共有4.1和4.2两个版本，其中4.1版本含47条指令，4.2版本含7条，共54条指令，另外4.2版本是在Ndahem型号CPU中实现的。该指令集不仅进一步完善了Intel 64指令集架构，还加入有关图形、视频编码及协同处理、向量运算、三维成像及游戏应用等指令，改进了多媒体处理和数据压缩等诸多领域的性能，是近几年来处理器领域最大的改进[43]。该指令集为Penryn型号CPU引入了2个32位矢量整数乘法运算模块，同时改进了8位无符号极值计算和32位有符号运算[43]。采用SSE4指令集的应用，能极大地提高程序编译性能和单精度及向量运算效率。另外，SSE4为提高向量计算性能，改善了cache存取、离散、跨步负载等指令。SSE4还加入了多条浮点运算指令，支持单精度、双精度浮点运算及浮点产生操作，能马上改变运算路径模式，极大改进延迟的影响，该类指令对改进3D性质计算性能有着很大的作用。另外该指令集还加入了传流式负载指令，可以改善帧缓冲区的读取数据频宽，即可以将多个数值组成一个128位的数值进行统一运算，从而获得完整的快取缓存行，提高最高八倍的读取频宽效能。另外该指令集还改进了视频编码性能，即可一次性对8个4字节的SAD（Sums of Absolute Differences）运算进行处理，这种改进对于新一代如VC.1及H.264等规格的高清影像编码中，可以让视频编码速度获取更大提高。同时AMD也在Phenom与Opteron等K10型号CPU里实现了相应的SSE4多媒体指令集，但该类实现和Intel的SSE4指令集不兼容。

SSE5指令集是AMD独自创立的新指令集，其与Intel的不兼容，该指令集加入超过100条新指令，里面比较重要的指令有三运算对象指令（3-Operand Instructions）及熔合乘法累积（Fused Multiply Accumulate）。其中，三运算对象指令能使CPU利用数学或逻辑运算，并应用到运算对象或输入数据，同时将多条x86指令组成一条SSE5指令，这样增加了运算对象的数量，提高了指令效率，是少数复杂指令集架构的水平。熔合乘法累积可以根据需要新增新的指令，提高了各类复杂运算性能，并与其他指令组合在一起去计算原本需要多次重复计算的运算。同时精简编码，让处理器能快速进行图像处理、音频声效及其他复杂向量运算等计算密集的运算。但由于不能与Intel兼容，所以AMD并未将SSE5指令集加入Bulldozer核心中，而改用内置Intel授权SSE4系列指令集，但该指令集的最大意义是为下一代的AVX指令集提供了很好的思路。

AVX(Advanced Vector Extensions) 是Intel与AMD在X86指令集的SSE扩展补充，即将原128位的寄存器扩展至256位，这样又能一次性处理更多数据，是原性能的两倍。该指令集采用了AMD SSE5的实现想法，并对其进行了改进，构建出了新一代的完整SIMD指令集规范。该指令集支持了三运算指令，即减少以前操作中要先copy再计算的操作。同时在微码部分采取LES LDS这两个指令作为延伸指令前缀。AVX指令集是Sandy Bridge架构中最大的改善[44]。

FMA是Intel的AVX指令集的另一个实现，所以该指令集采用和AVX指令一样的编码设计规范。类似于熔合乘法累积（Fused Multiply Accumulate），它可以根据需要新增新的指令，提高了各类复杂运算性能，并与其他指令组合在一起去计算原本需要多次重复计算的运算，同时精简编码，让处理器能快速进行图像处理、音频声效及其他复杂向量运算等计算密集的运算。FMA改进了浮点运算能力。在新推出的Haswell型号处理器里存在两个新的FMA单元，每个FMA单元支持8个单精度或4个双精度浮点数。

## 4.2 已有的应用

近二十年间，探索使用计算机字并行ram模型来加速串匹配算法的尝试有很多。在这个模型中，计算机操作长度为ω的字，使得能一次性读入块字符。这意味着许多算法中的一些对字的操作步骤能在一个CPU单位时间内完成。

许多使用字并行ram模型的算法都基于以下两种技术，位并行技术和块字符技术。

### 4.2.1 位并行

位并行技术即是考虑到CPU处理一个机器字长的数值和一个字符所耗时间相等得事实，所以将多个字符组合到长度为ω的机器字内，如ulong就能组8个字符，组完后通过一次CPU运算即可得到相应结果。采用位并行，原算法的时间复杂度能降低ω（机器字长的位数）倍，获取巨大的性能提升。考虑到现在的CPU基本都是64Bit的，并且其中每一位都能保存相应的匹配结果，所以我们能在单词比较中完成对64个字符的比较，BF算法采取这种改进方式能提高64倍的运行速度。Shift-Or（SO）算法就是第一个使用位并行的算法，该算法高效地模拟了KMP自动机的非确定性实现并且算法时间复杂度在。并且在小字母表的短串模式匹配中，SO算法仍然是一个相当快速的实用算法。不久后快速BDM改进算法（BNDM）产生，该算法使用位并行去实现非确定性前缀自动机。其中一些BNDM算法的变体也是目前最快的串匹配算法。

然而，位并行技术最大的限制在于它只能处理长度小于计算机位长的串，当串长m小于该长度时，位并行算法非常高效。一旦超过该长度，对超过该长度的比较就只能使用BF算法。虽然有一些改进技巧能让位并行在长串模式匹配时仍能保持优势，但实际效果没有强太多，因为内在的位长限制是不可改变。

### 4.2.2 块字符

在块字符技术中，多个字符被打包成一个较大的字，使得这些字符能同时被计算而不用逐个去比较。在这种情形下，若模式串的字符是由字符表大小为σ组成，那么能组成一个单字的字符个数为。在这里，我们定义块字符因子α=。

第一个使用块字符的方法是由Fredriksson提出 [45]。他提出了一个方案能给许多串匹配算法加速。而第一个使用块字符获得很好实验结果的算法是由Ben-kiki等人于2011年提出的ssecp算法 [3]。该算法只基于两条特殊的SSE指令，pcmpestrm和pcmpestri指令，但算法时间复杂度只有(n/α)。该作者通过实验在论文中得出结论在短串模式匹配中ssecp算法是最快的。

### 4.2.3 Four-Russians

所谓Four-Russians技术，是指将某个问题划分成若干小问题，如果某个小问题在求解大问题的过程中反复出现，则可以将这个小问题事先计算出来，则求解大问题过程中如果碰到这个小问题只需要查表或者通过一些简单计算就可以得到这个小问题的结构，从而加速整个问题的求解。

当模式串的长度增长时，ssecp算法的优势逐渐减弱。这时，Kulekci提出了流式SIMD扩展过滤算法ssef[46]，该算法利用Four-Russians的思想使用一种过滤方法去检查块字符而不是逐个去比较。尽管该算法在理论上存在最差的时间复杂度，但在实际应用中，它是长串模式匹配中最快的。

**第5章 改进算法IEPSM详述**

## 5.1 综述

尽管可供使用的SSE指令有很多，但大多数指令在实际执行需要的CPU时间周期数仍然较大。因此在本文中，我们将挑选四条执行周期数少的指令来实现我们的精确块字符匹配算法（iepsm）。

但在设计算法之前，我通过编程实验发现计算机处理串匹配问题有如下特性，如果不将这些特性运用进去，只根据理论预测的算法可能无法达到预期的效果。

这些特性如下：

* 查询表的大小（即数组定义）最好设置在2048左右。因为大部分串匹配算法都引入了查询表，而从计算机体系结构来看，过大的数组会导致cache不命中，从而带来的硬件中断，降低效率。Iepsm将大小设置为2048，这样既能保证查询表足够大又能降低cache不命中率。
* 模式串的指针跳转长度定义成常量。经实验，指针的常量跳转速度比读变量跳转有倍数的性能提升。

由于基因串的特征和字符表大小相同的随机串表现类似，且由于基因模式串的长度一般较长，所以我们重点考虑长串匹配的情况。考虑到大部分SSE指令的输入参数为128位的数据，所以iepsm算法根据串长的不同以16个字符为界分为两种不同的优化方案，并在长串匹配中将上述的两点特性运用进去。

首先，我们需要通过组合SSE指令来为我们的算法设计合适的伪指令。

### 5.1.1 wscmp(a,b)字长比较指令

Wscmp指令被设计来比较两个ω位的字，并返回一组α个γ位的值。如图5.1所示。

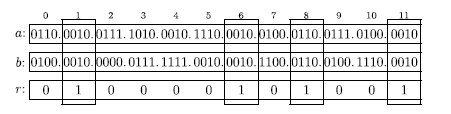


图5.1 wscmp指令样例，假定ω=48，α=12，γ=4

由图可以得知，在这条指令中，两个48位的字a和b被分成12组，每组是4位的数值。然后这些数据按序比较得到数值r。假定，b，那么wscmp指令将返回一个12位的值，即，其中当且仅当=时，=0，否则=1。

该伪指令能在一个较短的常数时间内模拟，它由如下两条SSE指令组成。

其中，\_\_mm\_cmpeq\_epi8的完整函数定义为\_\_m128i \_\_mm\_cmpeq\_epi8(\_\_m128i \_A, \_\_m128i \_B)。它分别比较寄存器\_A和寄存器\_B对应位置8位整数是否相等，若相等，则该位置返回0xff，否则返回0x0。

而\_\_mm\_movemask\_epi8指令的传参是一个128位的数值，该指令返回一个16位的数值为其签名。

### 5.1.2 wsmatch(a,b)字长匹配指令

Wsmatch指令被设计用来返回短串b在a中的出现次数[47]。如图5.2所示。

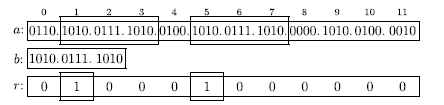


图5.2 wsmatch指令样例，假定ω=48，α=12，γ=4，k=3

假定，b，那么该指令将返回一个12位的数值，即，其中当且仅当对于，有，则=0，否则=1。也就是说只有当短串b出现在字符串a的起始位置i的子串中，那么=0。

该伪指令能在一个较短的常数时间内模拟，它由三条SSE指令组成，后两条即为wscmp指令。

其中，z是一个128位的数值，初始设为。

第一条指令又称为单指令32字节查分绝对值求和指令（MPSADBW），其完整函数定义为\_\_m128i \_\_mm\_mpsadbw\_epu8(\_\_m128i a,\_\_m128i b, const int mask)。该指令功能很强大，能同时执行8个4字节的SAD（Sums of Absolute Differences）运算。并且该指令很灵活，能通过选取不同的mask值来改变源操作数和目的操作数的位置。该掩码值虽然是32位数值，但指令只用后后三位。其中后两位用来选取源操作数的连续4个字节的起始位置。掩码的第三位用来选取目的操作时连续11个字节的起始位置。

该指令的运算过程类似于如下代码，如图5.3所示。

i = mask2 \* 4

j = mask0-1 \* 4

for (k = 0; k < 8; k = k + 1) {

t0 = abs(a[i + k + 0] - b[j + 0])

t1 = abs(a[i + k + 1] - b[j + 1])

t2 = abs(a[i + k + 2] - b[j + 2])

t3 = abs(a[i + k + 3] - b[j + 3])

r[k] = t0 + t1 + t2 + t3

}

图5.3 MPSADBW指令运算过程图

后两条指令已经在上一节介绍，这里不赘述。

### 5.1.3 wsblend(a,b)字长翻转指令

Wsblend指令用于混合两个ω位的参数。如图5.4所示。

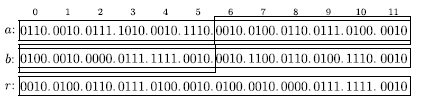


图5.4 wsblend指令样例，假定ω=48，α=12，γ=4

假定，b，那么wscmp指令将返回一个128位的值，即，由图易知r即为。

该伪指令能在一个较短的常数时间内模拟，它由如下三条SSE指令组成。

这里c设为。

第一条指令\_\_mm\_blend\_epi16的作用是根据参数c将a和b这两个128位的数值混合在一起，即。

第二条指令\_\_MM\_SHUFFLE是一条宏指令，用来直接在寄存器里调整数值里分量的顺序。

第三条指令\_\_mm\_shuffle\_epi32的作用是根据上一条指令定义的SHUFFLE值跳帧128位的数值h。在这里h被划分成4个int值。例如假如，则得到的。

### 5.1.4 wscrc(a)字长CRC码计算指令

Wscrc指令通过传入一个8bytes长度的字符串来生成一个32bit的CRC值。该指令通过该SSE指令\_\_mm\_crc32\_u64(α)来模拟，由于该指令只有一个时间周期，因此具有很快的速度和很好的鲁棒性。

但因为该指令生成的是64bit的数字，而我们的数组大小只有2048，所以还需将结果和掩码0x11111111111（2047）进行与运算来得一小于数组大小的值。

## 5.2 模式串串长m大于等于16

表5.1 IEPSM1-算法

|  |  |
| --- | --- |
| **IEPSM1算法** | |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14 | Prepare(P,m,T,n)  For i ← 0 to m-α do  h = wscrc(P+i)  shift[h].pos = i  shift[h].fingerprint = P+i  Search(P,m,T,n)  For i ←0 to n-α do  h = wscrc(T+i)  f = T+i  p = shift[h]  while(p!=null)  if f == p.fingerprint  memcmp(P,T)  p = p.next |

该算法分为两个阶段。

首先我们定义了查询表，它记录采样串在模式串的出现位置，其中采样串的长度是个试验参数B，取值在4bytes和8bytes之间。程序里由一指针数组Node\* shift[2048]来实现。其中结点Node，数据结构如表5.2所示。

表5.2 Node数据结构

|  |  |
| --- | --- |
| **Node数据结构** | |
| 01  02  03  04  05  06 | typedef struct node  {  struct node \*next;  int pos;  unsigned long long val;  }NODE; |

它存储某一采样串p1在模式串P中的位置，其中p1通过计算hash值h去定位到它的结点shift[h]。

那么在预处理阶段，对于模式串P，从位置i起取8bytes强转成64位ulong类型做为该子串的指纹值，然后将该子串与上掩码并调用wscrc(a)计算得到串hash值h，那么对于采样串p1，算得hash值h后，将p1在p中出现的位置i存储在shift[h]中，对于hash冲突造成的重值，存储在shift[h]对应的下一链表结点中。

在搜索阶段，首先指针指向匹配串T的m-8位置，并取8bytes子串，算得hash值h，若shift[h]有值i，说明该子串可能是串P在i位置的一部分，然后判断该子串是否和shift[h]的指纹值相等，若相等，再根据i值将串P的i位置和该子串对齐，并调用memcmp函数进行比较，若shift[h]有多个链表结点，则需根据i值进行多次对齐和比较。完成后指针再继续前进m-B的长度，并重复刚才的搜索步骤。

在这里，我们采用了m-B的常值跳转，而根据鸽笼原理，m-B是iepsm不漏扫的最大距离。所以容易看出，如果采样串长度B较小，即能获取较大的跳跃距离m-B，但与此同时数组结点对应的链表会很长，这样每次比较都会去遍历这个链表。另外B越小，采样串的匹配概率越大，导致mencmp全匹配函数的调用次数加大，抵消改进；而根据经验，shift数组越大，程序会因CPU缓存原因查询越慢，所以合适的大小是2048。但对于基因串来说，4的8次方为65536，会造成较大的hash冲突，同样降低性能。

因此我们可以通过实验去找到合适的采样串长度，使得它既能获取较大的跳跃距离，同时又不造成很高的hash冲突。

另外解释下指纹值的作用。不论B选多大，我们实际上都是读取8bytes的采样串p0，但通过掩码来获取变长的采样串p1， 所以我们通过在调用memcmp前比较p0对应的64bit数值与对应模式串位置的64bit数值，这样就能大大减少全比较函数调用次数。具体实现是在Node定义中加一字段存储原采样串的64bit数值，同时在预处理阶段根据模式串P将其初始化。

## 5.3 模式串串长m大于等于4且小于16

表5.3 IEPSM2-算法

|  |  |
| --- | --- |
| **IEPSM2算法** | |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | Prepare(P,m,T,n)  m1 ← min{m,α/2}  p1 ← p[0..m1-1]  Search(P,m,T,n)  For i ←0 to n/α -1 do  R ← wsmatch(,p1)  If r != Then  If m = m1 then  report occurrences at iα+{r}  else check positions iα+{r}  S ← wsblend(,)  R ← wsmatch(S,p1)  If r != then  If m = m1 then  Report occurrences at iα+α/2+{r}  else check positions iα+α/2+{r} |

该算法只处理串长大于等于4字节且小于16字节的情况。因为如上文所述，由于基因的字符表很小，所以基因序列的查询一般不会涉及4字节以下，所以串长低于4字节的iepsm将不考虑。

**第6章 实验结果与分析**

本实验的测试机为CPU型号为Intel(R) Xeon E3-1230 V2，工作频率为3.30GHz，4G内存，操作系统为Linux Mint 13。实验中所有的测试算法均由C语言实现，并由GCC编译。

## 6.1 测试算法选取

我们选取了最新的基于前缀搜索的算法ufndmq、基于后缀搜索的算法hashQ、基于子串搜索的算法fsbndmq、针对生物学的串匹配算法tvsbs和使用SSE指令处理块字符的ssecp这五种算法和我们的新算法进行比较。

## 6.2 测试集选取

我们选取了三种类型生物的基因序列和氨基酸序列做为测试集，这三个物种具有很好的代表性。测试集详细在表6.1中列出。

表6.1 测试数据集选择

|  |  |  |
| --- | --- | --- |
| **种类** | **基因序列** | **蛋白质序列** |
| 大肠杆菌 | ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria/Escherichia\_coli\_K\_12\_substr\_\_MG1655\_uid57779/NC\_000913.fna | ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria/Escherichia\_coli\_K\_12\_substr\_\_MG1655\_uid57779/NC\_000913.faa |
| 水稻 | ftp://ftp.ensemblgenomes.org/pub/plants/release-24/fasta/oryza\_sativa/dna/Oryza\_sativa.IRGSP-1.0.24.dna.genome.fa.gz | ftp://ftp.ensemblgenomes.org/pub/plants/release-24/fasta/oryza\_sativa/pep/Oryza\_sativa.IRGSP-1.0.24.pep.all.fa.gz |
| 人类 | ftp://ftp.ensembl.org/pub/current\_fasta/homo\_sapiens/dna/Homo\_sapiens.GRCh38.dna.primary\_assembly.fa.gz | ftp://ftp.ensembl.org/pub/current\_fasta/homo\_sapiens/pep/Homo\_sapiens.GRCh38.pep.all.fa.gz |

## 6.3 参数选取实验

在该实验中，我们通过选取五种不同的参数来获取不同种类序列数据下的最优参数。表6.2-7中最优数据用斜体加粗标出。

表6.2 大肠杆菌基因测试数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| iepsm4 | ***3.27*** | 3.08 | 2.09 | 2.21 | 2.12 | 2.02 | 1.31 | 1.81 | 1.36 | 1.76 | 1.31 | 1.22 | 1.46 | 1.22 | 1.7 |
| iepsm5 | 3.84 | 2.77 | 2.2 | 1.37 | 1.74 | 1.96 | 1.71 | 0.97 | 1.37 | 1.05 | 1.25 | 1.13 | 1.11 | 0.71 | 0.71 |
| iepsm6 | 3.54 | ***2.31*** | ***2.09*** | ***1.14*** | ***1.18*** | ***1.03*** | ***0.83*** | 1.13 | 1.31 | 0.5 | 0.65 | 0.34 | 0.6 | 1.05 | 0.3 |
| iepsm7 | 3.95 | 2.42 | 2.31 | 1.27 | 1.84 | 1.61 | 1.57 | 1.34 | 1.28 | 1.25 | 1.21 | 0.9 | 0.84 | 0.62 | 0.51 |
| iepsm8 | 4.85 | 3.6 | 2.46 | 1.44 | 1.36 | 1.51 | 1.51 | ***0.69*** | ***0.57*** | ***0.52*** | ***0.81*** | ***0.79*** | ***0.17*** | ***0.4*** | ***0.43*** |

表6.3 水稻基因测试数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| iepsm4 | ***269.64*** | 192.34 | 154.25 | 134.13 | 130.48 | 116.81 | 108.32 | 103.69 | 100.46 | 86.73 | 74.67 | 72.00 | 71.69 | 68.10 | 60.38 |
| iepsm5 | 271.99 | 190.55 | 145.98 | 120.71 | 108.45 | 96.35 | 88.26 | 79.59 | 77.07 | 60.81 | 53.69 | 48.03 | 48.64 | 46.16 | 37.02 |
| iepsm6 | 296.24 | ***188.66*** | ***137.71*** | ***113.11*** | ***94.04*** | ***81.05*** | 73.69 | 64.99 | 64.88 | 49.67 | 42.68 | 35.22 | 27.21 | 23.70 | 19.37 |
| iepsm7 | 358.05 | 207.44 | 144.98 | 118.34 | 95.81 | 82.95 | 74.96 | 64.81 | 60.37 | 47.32 | 40.90 | 33.56 | 23.38 | 17.28 | 13.45 |
| iepsm8 | 440.13 | 226.16 | 153.86 | 118.23 | 96.98 | 84.19 | ***72.78*** | ***64.14*** | ***59.39*** | ***45.95*** | ***40.89*** | ***30.60*** | ***20.59*** | ***15.19*** | ***10.92*** |

表6.4 人类基因测试数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| iepsm4 | ***1833.65*** | 1416.69 | 1265.22 | 1089.91 | 954.36 | 937.78 | 918.14 | 848.27 | 728.48 | 719.65 | 591.28 | 621.64 | 497.39 | 536.42 | 406.07 |
| iepsm5 | 1976.54 | 1350.44 | 1045.03 | 865.67 | 763.62 | 685.66 | 663.47 | 612.89 | 567.02 | 480.07 | 386.43 | 374.30 | 346.34 | 364.98 | 273.49 |
| iepsm6 | 2134.28 | ***1345.82*** | ***992.15*** | ***807.76*** | ***683.10*** | ***582.12*** | ***517.52*** | 500.12 | 459.53 | 386.18 | 322.16 | 282.56 | 213.18 | 191.98 | 176.22 |
| iepsm7 | 2516.66 | 1455.80 | 1025.39 | 817.82 | 697.70 | 595.98 | 544.09 | 485.27 | 449.21 | 360.28 | 308.06 | 252.78 | 193.30 | 144.38 | 122.68 |
| iepsm8 | 3077.91 | 1549.02 | 1078.94 | 824.01 | 699.93 | 594.25 | 532.21 | ***466.46*** | ***445.42*** | ***351.45*** | ***303.27*** | ***236.61*** | ***151.72*** | ***116.49*** | ***92.90*** |

表6.5 大肠杆菌蛋白质测试数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| iepsm4 | ***1.58*** | ***1.2*** | ***0.79*** | ***0.63*** | ***0.36*** | ***0.46*** | 0.49 | ***0.33*** | ***0.22*** | 0.44 | 0.66 | 0.69 | 0.99 | 0.87 | 1.11 |
| iepsm5 | 1.61 | 1.54 | 1.08 | 0.98 | 0.86 | 1.02 | 1.19 | 0.44 | 0.78 | 0.52 | 0.96 | 0.84 | 0.82 | 0.54 | 0.48 |
| iepsm6 | 1.72 | 1.68 | 0.85 | 1.03 | 0.89 | 1.16 | ***0.31*** | 0.67 | 0.84 | ***0.35*** | ***0.42*** | ***0.16*** | 0.48 | 0.93 | 0.18 |
| iepsm7 | 1.89 | 1.66 | 1.01 | 0.85 | 0.79 | 0.65 | 1.23 | 0.39 | 0.36 | 0.43 | 0.46 | 0.73 | 0.31 | 0.34 | 0.39 |
| iepsm8 | 1.97 | 1.66 | 1.05 | 0.94 | 0.42 | 0.46 | 0.87 | 1.08 | 0.93 | 0.9 | 0.98 | 0.38 | ***0.28*** | ***0.19*** | ***0.15*** |

表6.6 水稻蛋白质测试数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| iepsm4 | ***9.7*** | ***7.25*** | ***5.56*** | ***4.92*** | ***4.3*** | ***3.91*** | ***3.61*** | 3.57 | 2.59 | 3.12 | 2.31 | 1.63 | 1.58 | 1.22 | 1.09 |
| iepsm5 | 11.55 | 7.56 | 6.2 | 5.25 | 4.68 | 4.31 | 4.36 | 3.02 | 3.01 | 2.4 | 2.55 | 1.9 | 1.23 | 0.83 | 0.76 |
| iepsm6 | 11.89 | 8.57 | 6.38 | 5.55 | 4.89 | 4.63 | 3.88 | 3.42 | 3.13 | ***2.27*** | ***2.01*** | ***1.39*** | 1.36 | 1.34 | 1.21 |
| iepsm7 | 14.13 | 8.31 | 6.48 | 5.45 | 4.31 | 4.43 | 4.58 | ***2.74*** | ***2.8*** | 2.31 | 2.05 | 1.73 | 0.74 | 0.85 | 0.64 |
| iepsm8 | 17.74 | 9.31 | 6.58 | 6.08 | 4.42 | 4.04 | 4.22 | 3.67 | 3.34 | 2.84 | 2.39 | 1.72 | ***0.69*** | ***0.69*** | ***0.19*** |

表6.7 人类蛋白质测试数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| iepsm4 | ***21.97*** | ***15.36*** | ***11.56*** | ***9.23*** | ***9.90*** | ***6.99*** | ***6.34*** | ***5.80*** | ***5.29*** | ***4.51*** | 4.44 | ***2.83*** | 2.22 | 1.39 | 1.36 |
| iepsm5 | 26.37 | 17.30 | 13.17 | 10.78 | 10.73 | 7.65 | 6.84 | 6.10 | 5.48 | 4.75 | 4.47 | 3.08 | 2.30 | 1.60 | 1.35 |
| iepsm6 | 30.75 | 18.80 | 13.85 | 11.14 | 10.88 | 7.95 | 7.12 | 6.50 | 5.57 | 4.58 | ***4.13*** | 3.06 | 2.40 | 1.49 | 1.38 |
| iepsm7 | 36.49 | 20.66 | 14.94 | 11.65 | 11.34 | 8.27 | 7.27 | 6.52 | 5.59 | 4.93 | 4.40 | 3.23 | 2.36 | 1.52 | 1.44 |
| iepsm8 | 44.74 | 22.52 | 15.73 | 11.99 | 11.50 | 8.25 | 7.33 | 6.49 | 5.49 | 4.68 | 4.21 | 2.87 | ***2.11*** | ***1.35*** | ***1.24*** |

通过表6.2-7，我们发现，对大部分数据而言，存在表6.8和表6.9。

表6.8 基因数据参数选择

|  |  |
| --- | --- |
| **模式串串长m** | **基因序列** |
| 12≤m<16 | 4 |
| 16≤m<40 | 6 |
| 40≤m | 8 |

表6.9 蛋白质数据参数选择

|  |  |
| --- | --- |
| **模式串串长m** | **蛋白质序列** |
| 12≤m<256 | 4 |
| 256≤m | 8 |

## 6.4 算法对比实验

在该实验中，iepsm在长串处理中选取了6.3节实验中得到的最优参数。

### 6.4.1 算法效率

从在附录中的对比实验的数据图A1-6中易知，iepsm算法在与其他五种算法比较中是最高效的。随着模式串串长的增加，iepsm在大体上保持了稳定性。而其他基于启发式的算法这方面的轻微的增长。虽然基于位并行和可变优化参数的算法如hashQ，在长串查询中显示出不错的性能，但iepsm还是更胜一筹。从数据中我们还可以发现尽管ssecp使用了块字符的思想和SSE指令，但iepsm在效率上依然胜出，并且这一差距随着串长的增加表现得更加明显。

### 6.4.2 算法灵活性

在串匹配算法研究领域，算法的灵活性是指对于不同的输入数据，算法都能保持一个稳定的性能。也就是说不管输入数据是长串还是短串，是大字符表还是小字符表，算法都能保持很好的性能。

从实验中可以看出，我们通过给iepsm在不同数据环境下选取合适的块字符大小，使算法获得了很好的灵活性。其他算法如hashQ在字符表大小为20的氨基酸序列中表现很差，tvsbs则在字符表大小为4的基因序列中表现不佳。但这五种算法对串长的变化还是保持了稳定性。

结论

串匹配算法的研究在计算机研究领域里有相当长的历史，并且该类算法也广泛地应用在各类软件里。而近十几年来，由于生物信息学的飞速发展，尤其是基因检测技术的成本的降低，生成了海量的生物信息数据，提升串匹配算法的效率显得日益重要。

目前存在的算法改进一般分为三个方向。一是在经典串匹配算法的思想上进行优化，这一方向由于仍采取单字符读入处理的流程，所以对基因序列处理帮助不大；二是使用快速发展的GPU技术将原有的串匹配算法改成对应的版本，这一方向有很大前景，但由于GPU的指令版本较多，多数机器上未必能配上相应的环境，导致应用场景有局限；三是使用Intel SSE指令改进原有算法，但很多这类算法只是简单利用SSE指令去替代原经典串匹配算法里的指令组，虽然能获取不错的性能提升，但算法思想没有太大进步。

本文针对生物信息学中的单模式串匹配问题，使用Intel SSE指令并结合块字符思想提出了一个新的算法。由于基因串检索一般较长，所以本文重点考虑了模式串串长大于等于16的情形，另外对小串也给出了优化算法，只是没有过多考虑生物信息特点。

对于长串情形，该算法针对生物信息的特点，采用优化的块字符大小来生成hash表。除此之外，新算法采用一个64bit的大整数来表示字符串的指纹，该指纹通过8个字符生成。算法在进行逐字节的比较前通过该指纹过滤掉很多逐字匹配的调用。

该算法较其他使用SSE指令算法的创新点在于使用了最优编程实践，在长串处理中将跳转长度设为常值，仅当采样串满足一定条件才进行全匹配，而这种情况的发生概率在基因串中较低，所以该算法获得了极大的性能提升。

考虑到基因的变异性，在以后的研究中，会将SSE指令应用在多模式的近似串匹配中。

**参考文献**

[1] Thathoo R, Virmani A, Lakshmi S, et al. TVSBS: A Fast Exact Pattern Matching Algorithm for Biological Sequences[J]. Indian Acad. Sci, 2003, vol.91:47—53.

[2] Deusdado S, Carvalho P. An efficient algorithm for exact pattern-matching in genomic sequences[J].Bioinformatics Res. Appl, 2009, vol.5:385-401.

[3] Ben-Kiki O, Bille P, Breslauer D, et al. Optimal packed string matching[C]. IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, 2011, vol.13:423-432

[4] 王樱.基于模式匹配的DNA多序列比对及相似性分析[D].长沙:湖南大学,2011.

[5] 王樱.多序列对比算法综述[J].中国新通信,2014,5:92-93.

[6] 曹顺利,张忠平,李荣等.BioDW一个生物信息学数据集成系统[J].微计算机应用,2005,1:59-62.

[7] 孟晓笑.KMP算法的并行研究[J].湖北第二师范学院学报,2011,28(2):20-21．

[8] 鲁宏伟,魏凯,孔华峰.一种改进的KMP高效模式匹配算法[J].华中科技大学学报,2006,34(10):41-43.

[9] 俞松,郑骏,胡心文.一种改进的KMP算法[J].华东师范大学学报,2009,2009(4):92-97.

[10] 王文鹏,黄俊.对BM模式匹配算法的一种改进[J].计算机工程与应用,2011,47(32):108-111.

[11] 张红梅,范明钰.模式匹配BM算法改进[J].计算机应用研究,2009,26(9):3249-3252.

[12] 贺龙涛,方滨兴,胡铭曾.对BM串匹配算法的一个改进[J].计算机应用,2003,23(3):6-8.

[13] 渠瑜,王亚弟,韩继红等.对BM模式匹配算法的一个改进[J].计算机工程,2006,12(32):78-81.

[14] 丁国强,赵国增,李传锋.改进BM算法策略的网络入侵检测系统设计[J].计算机测量与控制,2011,19(11):2661-2664.

[15] 徐成,孙伟,戴争辉等.一种面向入侵检测的BM模式匹配改进算法[J].计算机应用研究,2006(11):89-91.

[16] 张国平,徐汶东.字符串模式匹配算法的改进[J].计算机工程与设计,2007,28(20):4881-4884.

[17] 王成,刘金刚.一种改进的字符串匹配算法[J].计算机工程,2006,32(2):62-64.

[18] 李雪梅,代六玲.对QS串匹配算法的一种改进[J].计算机工程,2006,23(3):108-109.

[19] 单懿慧,蒋玉明,田诗源.面向入侵检测的改进BMHS模式匹配算法[J].计算机工程,2009,35(24):28-31.

[20] 杨微微,廖翔.一种改进的BM模式匹配算法[J].计算机应用,2006,26(2):318-319.

[21] 刘沛骞,冯晶晶.一种改进的BM模式匹配算法[J].计算机工程,2011,37(17):248-249.

[22] Leena Salmela,Jorma Tarhio.Approximate string matching with reduced alphabet[J].Algorithms and Applications.Springer-Verlag,Berlin,Heidelberg,2010.

[23] Ateeq Sharfuddin, Xiaofan Feng. Improving Boyer-Moore—Horspool using machine-words for comparison[C].Proceedings of the 48th Annual Southeast Regional Conference,April 15-17,2010．

[24] Rajkumar Buyya,郑玮民译.高性能集群计算：结构与系统(第一卷),编程与应用(第二卷).电子工业出版社,2001,12.

[25] Marc Norton. Optimizing Pattern Matching for Intrusion Detection([www.idsreseach.org,2006](http://www.idsreseach.org,2006)).

[26] Calder B, Tuck N, Sherwood T, et al. Deterministic memory-efficient string matching algorithms for intrusion detection[J]. IEEE INFOCOM,2004.

[27] Katz R, Yu F,Laskhman T. Gigabit rate packet pattern matching with tcam[J]. Technical Report,2004.

[28] Tan L, Sherwood T. A high throughput string matching architecture for intrusion detection and prevention[J]. Proc. 31nd Annual International Symposium on Computer Architecture(LISA),2005.

[29] 张庆丹,戴正华,冯圣中等.基于GPU的串匹配算法研究[J].计算机应用,2006,7:1735-1737.

[30] Gonzalo N, Mathieu R. 柔性字符串匹配.北京,电子工业出版社.2007,17-19.

[31] ITEYE.字符串模式匹配算法. <http://dsqiu.iteye.com/blog/1700312>.

[32] 志文工作室.位并行算法与shift-and,shift-or算法.2011,11. <http://lzw.me/a/1383.html>.

[33] 赵学锋. 基于后缀数组的字符串模式查找的算法[D].兰州:西北师范大学,2011.

[34] 汪荣贵. 快速精确字符串匹配算法研究[D].合肥:合肥工业大学,2010.

[35] 百度文库.核酸. <http://wenku.baidu.com/view/67de5e7027d3240c8447efc4.html>.

[36] 百度文库.蛋白质. <http://wenku.baidu.com/view/4a9c953002768e9951e738d0.html>.

[37] 新浪博客.FASTA格式. <http://blog.sina.com.cn/s/blog_620b35790100kjsk.html>.

[38] 博客园.GENBANK格式. <http://www.cnblogs.com/frostbelt/archive/2010/07/26/1785508.html>.

[39] 百度文库.FASTQ格式. <http://wenku.baidu.com/view/f39875a7770bf78a6429541d.html>.

[40] 维基百科.单指令流多数据流. <http://zh.wikipedia.org/wiki/%E5%8D%95%E6%8C%87%E4%BB%A4%E6%B5%81%E5%A4%9A%E6%95%B0%E6%8D%AE%E6%B5%81>.

[41] 维基百科.SSE. <http://zh.wikipedia.org/wiki/SSE>.

[42] 邵研, 刘燕兵, 刘萍等.基于SSE指令集的串匹配算法优化.第三届中国计算机网络与信息安全学术会议[C].保定,2010.

[43] 范亚琼.基于SSE4指令集的H.264编码标准的运动估计优化[D].武汉:武汉理工大学,2010.

[44] AVX指令集技术与应用解析. <http://www.expreview.com/tag/AVX.html>.

[45] Fredriksson K. Faster string matching with super-alphabets[J]. String Processing and Information Retrieval,2002,207-214.

[46] Külekci M. Filter based fast matching of long patterns by using SIMD instructions[C].Proceedings of the Prague Stringology Conference,2009,118-128.

[47] Faro S, Kulekci M. Fast Packed String Matching for Short Patterns[C].Meeting on Algorithm Engineering and Experiments, ALENEX 2013.

**附录**

**附录A**：

图A1

图A2

图A3

图A4

图A5

图A6

**攻读学位期间发表论文与研究成果清单**

[1] Lin Dai,Li Wang & Jingru Wang. Evaluation of compression methods for genomic sequence [C]. Asia-Pacific Computer Science and Application Conference, 2014.

**致谢**

不知不觉，来到北京已经快三年了，而在北京理工大学的两年半硕士研究生生活也接近尾声，回首往昔，深深地觉得自己对这个城市、这个学校有着割舍不了的情感，离别的思绪开始蔓延。值此毕业论文完成之际，向所有帮助过我、关心过我的老师及同学表示诚挚的感谢和衷心的祝愿。

首先要感谢我的导师戴林副教授，他那严谨的治学态度，渊博的学识，宽广的胸襟，敏捷的思维，忘我的工作精神，以及平易近人的为人，为我树立了人生的榜样。平易近人的戴老师亦是我的良师益友，他严格把关、循循善诱，从本论文的选题，系统的设计一直到论文撰写都倾注了大量的心血，使得本论文得以顺利完成，他在软件开发，数据库设计领域的许多看法也让我受益匪浅。

感谢朱玉文、刘万春、黄河燕、鉴萍、李侃等老师，以及刘峡壁、马波、刘庆辉等老师，为我们传授知识；感谢王景如、宋健龙等同学，让我在实验室的生活中很舒适。

特别要感谢王景如同学，为我的论文和毕业设计提供了很多有价值的参考信息。

最后要感谢我的父母，他们25年来对我的关怀和照顾无微不至，养育之恩，无以为报。同时也是他们对我的鼓励和支持才使得我能走到今天，让我有勇气不断进取，不断攀登。

最后衷心所有关心、支持和帮助过我的人们！。