

太原理工大学

硕士学位论文

基于连续 r 位匹配规则的并行串匹配研究

姓名：王志国

申请学位级别：硕士

专业：计算机应用技术

指导教师：谢红薇

20070501

基于连续 r 位匹配规则的并行串匹配研究

摘 要

随着互联网的发展和普及,海量信息的处理和新的应用需求,对于串匹配这个计算机领域中一个基本的而又是重要的问题,提出了新的挑战。串匹配问题在 *Internet* 网络信息搜索、信息过滤、生物信息学、网络入侵检测、网络远程教育、电子商务等领域具有广泛的应用。

关于串匹配的问题很早就有相关的研究,提出了许多单模式匹配算法和多模式匹配算法。人们逐渐发现在实际应用中根据实际需要进行串匹配的研究具有重要的应用价值。因此,研究高效、快速的字符串匹配算法具有重要的理论价值和实际意义。

应该说,本文所提出的连续 r 位匹配规则是从精确串匹配规则中演变和发展而来的。这种演变思想的来源是,作者通过查阅医学领域以及计算机领域中关于免疫学、人工免疫系统和免疫算法的相关资料,从中了解到免疫系统已经应用到计算机领域来解决许多难题,并得到了满意的效果。但在这其中将免疫系统中的连续 r 位匹配规则应用到字符串匹配这个问题上的先例并不多。然而,作者却发现该免疫匹配规则有很多良好特性,如果能够将其应用到我们所讨论的串匹配问题上,那么将会得到比较好的效果。例如,其分布性、并行性的特点,如果应用到串匹配的问题上,那么当数据量剧增的时候,运行时间急剧下降的问题将会得到解决。在人类的免疫系统中,抗体的种类大概有 10^6 个,然而到目前为止,人们所发现的病毒性抗原的种类大概有 10^{16} 个,那么我们的肌体为什么能够在如此庞大的

病毒性抗原面前还能安然无恙呢？这里面，抗原与抗体的不完全匹配起着非常重要的作用。连续 r 位匹配规则正是从免疫系统中抗原与抗体不完全匹配性引出的，所以应用其特点来解决串匹配问题也是值得研究的。

作者正是基于以上的一些想法，开始了本论文的撰写工作。

首先，本文简单的介绍了一下免疫学的一些基本知识以及免疫匹配规则，给出了本文所应用的生物学基础。

其次，通过对 *KMP* 算法的分析，向其中引入了连续 r 位匹配规则因子，这样就可以通过预先控制匹配阈值 r 的大小来满足我们所需要的模式串与文本串相匹配的程度，从而得到我们满意的答案。

最后，作者通过实际搭建机群的经验，给出了搭建 *Cluster* 机群环境的详细步骤，并且在该系统下运行了并行串匹配程序。通过实验结果分析，达到了预先设计的效果。

在本文的总结部分作者提出了一些有待解决的问题，例如，如何再进一步的改善并行机群的通讯时间问题，以及通过与医学领域专家合作，设计出类似于免疫系统能够同一时间应对来自外界不同种的病毒性抗原的侵扰，而每个问题又可以并行来执行。这些都值得我们在今后的工作中进行研究。

关键词：免疫匹配规则，串匹配，并行

RESEARCH ON PARALLEL STRING MATCHING BASED ON R- CONTIGUOUS BITS RULE

ABSTRACT

With the rapidly growing of information and the appearance of new application requirements in engineering, the classic string matching algorithms face great challenges. The string matching algorithms have been widely used in the fields of Internet information search, informational filtration, biology informatics, intrusion detection, distance education, electronic commerce and so on.

Researches focus on exact string matching in early time, and many single and multiple string matching algorithms have been presented. However, the number of applications for string matching grows every day. It has very important theoretical value and practical meaning to research, so it is necessary to design very fast string matching algorithms.

The r-continuous bits matching is a kind of evolvement of the exact string matching. The reasons of the evolvement source are as follows. The author consults lots of Immunology, AIS and Immune Algorithms about medicine field and computer field. Lots of difficult problems in computer domain have been solved by AIS very well. However, few problems have been solved by the

r-continuous bits matching. Furthermore, the author discovers some excellent characteristics about the r-continuous bits matching rule. If the characteristics can be used in the string matching, we will gain satisfying result. For example, the characteristic of the immune system distribution and parallelism can be used in the string matching, and the problem of running time increasing rapidly will be solved when the data increases rapidly. Furthermore, the r-continuous bits matching rule discussed emphatically in the paper originates from the string matching of antigen and antibody. In the human immune system, the kinds of antibody are probably 10^6 , however, people have discovered that the kinds of antigen are probably 10^{16} . Why can our human body that be enclosed in giant virus be in a whole skin? The answer is the string matching of antigen and antibody.

Based on these ideas above, the author will compose the paper.

First of all, the paper simply introduces some basic knowledge about immune matching rule and presents the biological basis of the paper.

Secondly, the author analyses the KMP algorithm and introduces the r-continuous bits matching rule into the string matching. Thus, we can change the value r , which can control the string matching degree, and gain the satisfying answer.

Finally, the author introduces the detailed constructed procedure of the Cluster from experience. We run the parallel string matching program. The experimental results accord for the advance design.

In the seventh chapter, the author puts forward some unresolved problems. For example, how to improve the communication time. Cooperating with the medical experts, we can design computer immune system that is characteristic of the biological immune system. The biological immune system can defense kinds of antigen at the same time by the parallel method. In the future, these problems will be worth studying.

KEY WORDS: immune matching rule, string matching, parallel

声 明

本人郑重声明：所呈交的学位论文，是本人在指导教师的指导下，独立进行研究所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写过的科研成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律责任由本人承担。

论文作者签名： 王志国 日期： 2007.5.19

关于学位论文使用权的说明

本人完全了解太原理工大学有关保管、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： 王志国 日期： 2007.5.19

导师签名： 谢红薇 日期： 2007.5.19

第一章 绪 论

1.1 引 言

近年来,随着科技革命的进行,各个学科之间相互交叉、渗透,人们不断把一个领域的研究成果应用于另一个领域中。地球上除了人类还生活着数千万种生物,各种层次、各种类型的生物体系和生物环境本身有着无数未解之谜,人类很早以前就会利用生物技术解决工程问题,从而使仿生学这一专门学科诞生和发展起来。随着计算机技术和网络技术的飞速发展,以计算智能或软计算为代表的计算智能技术也迅速发展,其中有人工神经网络、模糊系统、进化算法,还有近几年刚刚发展起来的 DNA 计算和人工免疫系统等等,这些与模式识别和智能系统的发展相得益彰。

生物是计算问题的灵感源泉。生物免疫系统是一个复杂的自适应系统,可保护人体不受外部病原体侵害,它不依靠任何中心控制,具有分布式任务处理能力,具有在局部采取行动的智能,它通过相互作用的化学信息构成网络,进而形成全局概念。从人体免疫系统发展出来的计算方法已经引起许多不同领域研究人员的广泛兴趣。

人工免疫系统(*Artificial Immune System: AIS*)是继人工神经网络、进化计算之后又一新的软计算研究方向。人工免疫系统研究旨在通过深入探索生物免疫系统所蕴含的信息处理机制,建立相应的工程模型和算法,开拓新型智能信息处理系统,来解决国民经济和社会发展中面临的众多科技问题^[1]。近年来,人工免疫系统已经迅速成为研究热点。

随着互联网的普及和发展以及信息的爆炸式增长,早期的文本检索又被赋予新的任务,或者更确切的说是当前的需求对文本检索提出了更高的要求。要想在拥有海量信息同时又可能包含错误(或者不完全一致的表示形式)的互联网上搜寻、过滤和处理特定的信息,就必须借助于优良的匹配方法。许多搜索工具都或多或少的采用了模拟生物系统中匹配规则的理论或与之相关的技术^[2]。

作为生物界进化最成功的种群,人类的免疫系统在防止人体遭受抗原伤害上做出了巨大的贡献,也是自然界成功的免疫机制之一。如果能成功地模拟免疫系统中非常重要的一个过程即抗体与抗原的匹配机制,无疑为计算机在信息搜索以及串匹配问题上提供了一个非常良好的改进思路。

1.2 课题研究动态

文献[3]提出了一种基于分组的串匹配算法。文中证明了最优分组定理,并提出了基于最短路径和动态规划的两种最优分组策略。分组算法适合模式串长度变化幅度非常大的情形,是解决大规模串匹配的一种有效方法。

文献[4]提出了一种时间复杂度最优的精确串匹配算法。该算法将文本分成 $\lfloor \frac{n}{m} \rfloor$ 个长度为 $2m-1$ 的相互重叠的窗口。在每个窗口内,综合使用后缀自动机 *DAWG* 和 *AC* 自动机进行扫描,保证了 $O(n)$ 的最坏时间复杂度和 $O(\frac{n}{m} \log_{\sum |rm|})$ 的平均时间复杂度。

文献[5]使用 *Band-Row* 的方法来压缩 *SNORT* 中使用的 *AC* 算法,速度比原算法提高了 17%。这种表示法能够获得 $O(1)$ 的状态结点转换速度,只需要两次额外的边界检查。但是当一行中的非空元素个数大于 3 时,空间压缩的效果不明显。此外,两次边界检查,对算法性能影响较大,使得算法速度的提高有限。

文献[6]使用位图压缩(*Bitmap Compression*)和路径压缩(*Path Compression*)两种方法来节省存储空间,以利于在硬件上实现。该文的实验结果表明,用该方法在 *ASIC* 等器件实现 *AC* 算法,速度有较大幅度的提高,但该方法并不适合于软件实现。

从硬件并行和体系结构方面来设计串匹配算法的专用芯片也是近年来的一个研究热点。文献[7]中提到使用 *TCAM* 来进行串匹配算法的硬件并行化,获得 *GB* 级的匹配速度。文献[8]从体系结构方面对串匹配算法进行了研究,它将 *AC* 自动机进行分割,分配到并行的匹配引擎进行扫描,然后将匹配结果进行集成。这些方法都是利用硬件的高速和并行特性,以获取 *GB* 级的匹配速度。但是,专用硬件的存储空间有限,功能单一,软件上有效的并行算法很难应用到硬件上。

1.3 本文主要研究的内容

本文主要对 *KMP* 串匹配技术以及如何借鉴生物免疫系统中连续 r 位匹配规则的特点进行研究。在大规模串匹配方面,我们致力于解决如何利用并行计算的方法来解决串匹配的问题。本文的重点并不在如文献[7, 8]所研究的如何从硬件上或体系结构上改善匹配速度,而是应用已有的硬件并行技术,从算法上,从软件方面来进行研究,从而达到比较理想的匹配效果。

1.4 文章的组织结构

本文的主要章节安排如下：

第二章中主要介绍免疫学的一些基本概念，免疫系统的构成及特点，并且重点介绍了免疫系统中的匹配规则。

第三章主要介绍了串匹配的概念，一般的串匹配算法及缺点。重点讨论了 *KMP* 串匹配，简单的介绍了其串行算法。

第四章主要介绍了分布式环境与并行算法，这一章是本文的基点。讨论了免疫系统中淋巴系统如何并行运作的原理，以及借鉴其特点如何应用到并行串匹配问题当中去，并且据此给出了基于连续 r 位匹配规则的并行串匹配算法基本思想。本文作者正是注意到了免疫系统中的分布性特点，从而进行研究如何在分布式计算机系统下，利用免疫系统中的连续 r 位匹配规则，并行实现串匹配这个问题。

第五章主要介绍了并行计算的基本概念，以及并行计算机分类。并且根据自己实际成功搭建并行 *Cluster* 机群的经验，给出了搭建 *Cluster* 机群的具体方法。

第六章首先从理论方面分析了连续 r 位匹配概率。并且通过实验结果的比较，说明了在数据规模不断增加的情况下，本章给出的并行串匹配的算法具有比较好的效果。

第七章主要总结了本文的研究内容，同时指出了本文研究还存在的一些没有解决的问题，讨论了未来在该领域所研究的大概内容。

第二章 免疫学及免疫匹配规则

2.1 免疫学的基本理论

生物免疫系统是一个极其复杂的自适应系统,是人工免疫算法的生物学基础。因此,以下将简单介绍一下生物免疫系统的基本原理,以便于读者更好的理解人工免疫算法的原理和机制。

2.1.1 免疫学的一些基本理论^[9, 10, 11]

- ① 免疫:“免疫(*Immunity*)”一词源于拉丁文 *immunitas*,原意是免除税赋和差役,引入医学领域则指免除瘟疫之意。现代免疫学将“免疫”定义为:机体接触抗原性异物(如各种微生物)后,能产生一种特异性的生理反应,其作用可排除这些异物保护机体,因此长期以来免疫性仅指机体抗感染的防御能力。但近代免疫的概念是指机体对“自我”或“非我”的识别并排除非我的功能,籍以维持机体的稳定性。具体说,免疫是机体的一种生理反应,当抗原性异物进入机体后,机体能识别“自我”与“非我”,并发生特异性的免疫应答,排除抗原性的非我物质,这一过程又称为正免疫应答,或被诱导而处于对这种抗原性物质呈非活化状态,称为免疫耐受或负免疫应答。
- ② 抗原:“抗原(*Antigen*)”是一类能刺激机体的免疫系统使之产生特异性应答,并能与相应的免疫应答产物在体内或体外产生特异性结合的物质。它通常是由外部入侵的感染微组织或者有毒物质等组成。抗原有两种特征:其一为免疫原性,即抗原能刺激特定的免疫细胞,使免疫系统活化、增殖和分化,最终产生免疫效应物质(抗体和致敏淋巴细胞)的特性;其二为免疫反应性,即抗原与相应的免疫效应物质在体内或体外相遇时,可发生特异性结合而产生免疫反应的特性。
- ③ 抗体:“抗体(*Antibody*)”是 *B* 细胞识别抗原后克隆扩增分化为浆细胞所产生的一种蛋白质分子,也称为免疫球蛋白分子。抗体结合抗原,然后依靠自己或借助免疫系统其他元素(*T* 细胞等)帮助破坏这些抗原。抗体由两种截然不同的功能区的 *DNA*

分子片段组成：一部分是保持相对静态的稳定区，而另一部分是负责与不同的多种感染抗原结合的分子可变区。可变区基本只关心抗原结合，稳定区与所在宿主组织上的受体相互作用。稳定区变化有限，而可变区的变化是免疫系统在自适应处理中具有一定的能力和速度。对免疫应答期间产生的抗体多样性的研究已经证明了正是可变区为免疫系统提供了大部分的鲁棒性和自适应能力。

- ④ *T* 细胞：即 *T* 淋巴细胞，它在胸腺中成熟，其功能包括调节其它细胞的活动以及直接袭击宿主感染细胞。*T* 细胞可分为毒性 *T* 细胞和调节 *T* 细胞两类。而调节 *T* 细胞又可分为辅助性 *T* 细胞和抑制性 *T* 细胞。辅助性 *T* 细胞的主要作用是激活 *B* 细胞，与抗原结合时分泌作用于 *B* 细胞并帮助刺激 *B* 细胞的分子。毒性 *T* 细胞能够清除微生物入侵者、病毒或者癌细胞。
- ⑤ *B* 细胞：*B* 细胞是体内唯一能产生抗体（免疫球蛋白分子）的细胞。体内含有识别抗原特异性不同的抗体分子，其多样性是来自千百万种不同 *B* 细胞克隆。每个 *B* 细胞克隆的特性是由其遗传性决定的，可产生一种能与相应抗原特异结合的免疫球蛋白分子。*B* 细胞约占淋巴细胞总数的 10%~15%。

2.1.2 免疫系统的构成

随着现代免疫学的发展，已证明在高等动物和人体内存在一组复杂的免疫系统。它的生理功能主要是识别区分“自己”与“非己”成分，并能破坏和排斥“非己”成分，而对“自己”成分则能形成免疫耐受，不发生排斥反应，以维持机体的自身免疫稳定。

免疫系统是由免疫器官、免疫细胞和免疫分子组成^[12, 13]。

免疫器官根据它们的作用，可分为中枢免疫器官和周围免疫器官。禽类的法氏囊（腔上囊）、哺乳类动物和人的胸腺和骨髓属于中枢免疫器官。骨髓是干细胞和 *B* 细胞发育分化的场所，法氏囊是禽类 *B* 细胞发育分化的器官。胸腺是 *T* 细胞发育分化的器官。脾和全身淋巴结是周围免疫器官，它们是成熟 *T* 细胞和 *B* 细胞定居的部位，也是发生免疫应答的场所。此外粘膜免疫系统和皮肤免疫系统也是重要的局部免疫组织。

免疫细胞的广义概念可包括造血干细胞、淋巴细胞系、单核吞噬细胞系、粒细胞系、红细胞以及肥大细胞和血小板等。

免疫分子可包括免疫细胞膜分子，如抗原识别受体分子、分化抗原分子、主要组织相容性分子以及一些其它受体分子等；也包括由免疫细胞和非免疫细胞合面和分泌的分

子，如免疫球蛋白分子、补给分子以及细胞因子等。

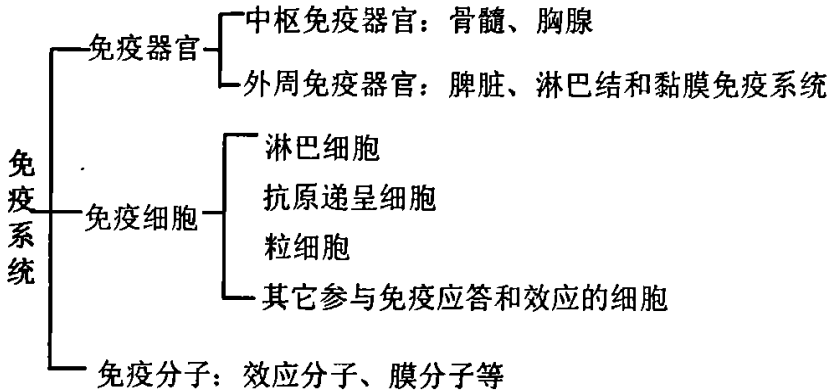


图 2-1 免疫系统构成

Fig 2-1 immune system composing

2.1.3 生物免疫系统的一些基本特点

- ① 分布性：在生物免疫系统中，分布于全身各处的淋巴细胞能够独立检测外来病菌和病毒的入侵，免疫应答是无集中控制的，是通过分布在全身众多的局部免疫部件之间的相互作用，来实现对整个机体的保护。这就意味着某一淋巴细胞检测的失败不会导致整个免疫系统的失败。*Mark Crosbie and Gene Spafford* 提出了一种在计算机安全领域的分布式、移动代理结构。而结合应用免疫学原理可以建立一个高度分布性和增强鲁棒性的分布式结构。本文正是从免疫系统具有分布性这一特性为基点，详细的分析了其特点，并且由此设计了本文的并行串匹配算法。关于分布式环境与并行计算的具体内容在第四章有更加详细的介绍。
- ② 多样性：在生物免疫系统中，每个免疫细胞都具有自己的特点而和其它免疫细胞不完全相同，每个免疫细胞所能识别和删除的病原体的能力都不一样，这就形成了免疫细胞的多样性。首先，有机体内免疫细胞的多样性保证：当每一种抗原侵入机体时，都能在机体内选择出可识别和消灭相应抗原的免疫细胞，并使之激活、分化和增殖，进行免疫响应，最终清除抗原。其次，每个个体都有一个独一无二的免疫系统，存在着群体免疫系统的多样性。这种多样性保证：当某个体对某病菌呈现脆弱性时，不会出现所有个体都对同一病菌呈现脆弱性的情况。因此，免疫系统的多样性可大大增强个体和群体的鲁棒性。
- ③ 自治性：是指在免疫系统中并不需要外部的控制，免疫系统是机体的一个完整的组

成部分,因此,监控和保护机体其它部分的机制也同样监控和保护着免疫系统。此外,免疫系统的分布式和非中心控制的本质也对自治性做出了贡献。免疫系统不仅没有外部的控制,而且没有办法施加外部控制,甚至连内部控制和中心控制都无法施加,这种自治能力如果能够在并行计算中得到应用,那么系统的稳定性就将会得到很大程度的提升,某个机器的死机不会影响整个系统的性能。

- ④ 不完全匹配性:前面已经做过介绍,免疫响应是特异性的,每种抗体或受体只能识别和结合特定的抗原。同时人们又发现,体内的约 10^6 种不同的抗体能识别出约 10^{16} 种不同的外来抗原模式。因此,抗原和抗体的匹配是不完全的,即不是一对一的。一种抗体可以对几种不同的、结构相近的抗原做出反应。借鉴这一特性,解决计算机的字符串不完全匹配方面的问题,将会十分有效。例如在字符串匹配过程中,只要我们给出需要匹配的阈值即匹配的程度就可以按照要求搜索出符合条件的答案,否则,如果都是完全匹配才是答案的话,那么匹配成功率将是非常的低。
- ⑤ 不完美检测:免疫系统凭借这一特性可以灵活的分配其资源。举例来说:在免疫系统中,检测特定病原体的淋巴细胞越少则免疫系统检测病原体的范围就越大,但是这将会降低检测特定病原体的效率。

当然,免疫系统还有一些其他的特点,如:多层次性、个体的非关键性、自学习功能,等等。在这些特点当中,我们着重强调不完全匹配性,在下面的篇幅中我们会重点讨论一下免疫系统中的不完全匹配性的特点和应用。

2.2 免疫细胞的激活门限

免疫系统是通过淋巴细胞表面的受体(或抗体)与外部抗原形成化学键(即结合),来完成对抗原的检测或识别。由于特异性,受体只能结合结构相似的抗原,受体结构与抗原结构互补性越好,二者之间就越容易形成化学键,化学键的强度称为亲合力。一个淋巴细胞表面大约有 10^5 的受体,这些受体都可以结合抗原。我们可以通过受体所形成的化学键的数目,来估计亲合力的大小,淋巴细胞周围结合的抗原越多,亲合力就越大。当结合的抗原数目超过了某个门限(称为激活门限)时,淋巴细胞被激活,即淋巴细胞表面的受体与相应抗原的亲合力足够高时,该淋巴细胞被激活,它才具有免疫功能。此处的激活门限与描述两个字符串匹配的程度所规定的匹配阈值非常相似,只要待匹配的字符串与模式串匹配的阈值大于等于规定的阈值,我们就认为他们是匹配的。

2.3 免疫匹配规则

匹配规则是免疫系统正常运作的一个关键点：在某种抗原入侵机体时，抗体首先识别这种抗原的种类，应用匹配规则来识别是否是已识别过的抗原，并且判断这种抗原是否有异常变化发生。在这过程中，匹配分为完全匹配和部分匹配。如果两个等长字符串的每个对应位上的符号都相同，那么这样的匹配称为完全匹配。然而，在生物免疫系统中，抗体和抗原的结合，更多地表现出不完全匹配特性，完全匹配只是其中的一个特例，因此人们更关心部分匹配规则。

免疫系统中通常有许多部分匹配规则，如海明规则、连续 r 位匹配规则等。在连续 r 位的匹配规则中，它需要根据连续匹配的位数来确定两个字符串是否匹配。当连续匹配的位数大于等于 r 值时，两串匹配，否则不匹配。这些匹配都是部分的匹配，规则中的阈值或 r 值，类似于生物免疫系统中免疫细胞的激活门限。每一个淋巴细胞表面有许多相同的受体，当有足够多的受体被抗原结合，即细胞表面结合的抗原数量超过某个值时，该免疫细胞被激活，这个值就是免疫细胞的激活门限。在连续 r 位匹配规则中， r 表示了文本串与模式串至少有 r 个对应位取值相同时，才能检测出匹配的字符串。连续 r 位的部分匹配规则，更接近于生物免疫系统的匹配过程和特性。

2.3.1 匹配规则

根据生物免疫系统的原理，我们知道淋巴细胞表面的受体是检测和清除病原体的关键，受体是否“合格”对于免疫系统是否能正常工作是非常重要的。同样道理在人工免疫系统中，匹配算法是否有效也是非常重要的，它直接关系到匹配系统是否能正常工作。所以我们在人工免疫系统中，需要用准确的匹配规则判断已有的文本串与模式串是否能够按照预先设置好的阈值来进行匹配，以确定匹配算法是否“合格”。

在免疫系统中有两种常用的匹配规则：海明匹配规则(*Hamming Match Rule*) 和连续 r 位匹配规则(*r-Contiguous bits Rule*)。

下面将简单的介绍一下海明匹配规则，并且详细的讨论连续 r 位匹配规则。

2.3.2 海明匹配规则 (Hamming Match Rule)

海明匹配规则(Hamming Match Rule) 是基于海明距离的匹配规则。

其定义为: 对于两个长度为 L 的字符串 a 和 b , 如果两个字符串 a 和 b 相应位上至少有 r 位相同, 则这两个字符串在海明匹配规则下匹配, $Match_h(a, b)$ 为匹配函数, 并有:

$$Match_h(a, b) = \begin{cases} 1 & \text{字符串 } a \text{ 与 } b \text{ 在海明规则下匹配} \\ 0 & \text{否则} \end{cases} \quad (2.1)$$

例如: 串 $a = 0 \boxed{1} 0 \boxed{0 \ 1 \ 1} 0 \ 0 \ 1 \ 1 \ 1 \ 1 \boxed{0} \ 1 \ 0 \ 1$
 串 $b = 1 \boxed{1} \ 1 \boxed{0 \ 1 \ 1} \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \boxed{0} \ 0 \ 1 \ 0$
 串 $c = 0 \ 1 \boxed{0} \ 1 \boxed{1 \ 1 \ 0} \ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0$
 串 $d = 1 \ 0 \boxed{0} \ 0 \boxed{1 \ 1 \ 0} \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1$

本例中, 设二进制字符串长度 $L = 16$, 阈值 $r = 5$ 。在海明匹配规则下, 字符串 a 和 b 匹配, $Match_h(a, b) = 1$; 而字符串 c 和 d 不匹配, $Match_h(c, d) = 0$ 。

在进行字符串匹配的过程中待匹配的字符串与已有的字符串进行匹配, 只要被检测的字符串与已有的字符串在海明匹配规则下匹配的位数之和 $\geq r$, 系统就认为该字符串匹配成功。

两个长度为 L 的随机的二进制字符串 a 和 b , 在海明匹配规则下的匹配概率为:

$$P(Match_h(a, b) = 1) = P_{ham} = 2^{-1} \sum_{i=r}^L \binom{L}{i} \quad (2.2)$$

生物体中常发生基因复制与删除, 许多情况下直接运用海明距离来衡量两个序列(串)的相似程度是不合理的^[14]。在计算机免疫系统中也是如此。所以海明匹配规则的使用范围也受到限制。

2.3.3 连续 r 位匹配规则 (r -Contiguous bits Rule)

在生物免疫系统中，淋巴细胞对病原体即抗原的检测是通过依附于其表面的受体即抗体去检测的。每一淋巴细胞表面有很多种不同的抗体，当病毒性抗原入侵机体后，淋巴细胞表面的抗体和病毒性抗原开始发生识别反应。当抗体编码基因与抗原编码基因之间匹配成功的数量超过一定值时，该抗体所依附的淋巴细胞就被激活并开始发生免疫行为，这个值被称为免疫细胞的激活门限值。而我们所研究的重点就是抗体编码基因与抗原编码基因之间的识别过程，以及抗体与抗原之间匹配成功时的条件。

依据这个思想，可以把免疫系统中抗体与抗原之间匹配过程所用到的术语与计算机领域中字符串之间匹配过程所用到的术语有如下的映射关系：

免疫学领域	计算机领域
抗体	←→ 文本串
抗原	←→ 模式串
抗体编码基因	←→ 文本串中的字符
抗原编码基因	←→ 模式串中的字符
激活门限值	←→ 匹配阈值

表 2-1: 免疫学领域与计算机领域映射关系
Table 2-1 immune field and computer field relationship

连续 r 位匹配规则定义：如果有两个字符串 a 和 b ，它们至少有连续 r 个对应位上的符号相同，则称字符串 a 和字符串 b 相互匹配。也就是说，通过比较两个字符串 a 和 b 在对应的符号位上连续匹配的数量与预先设定的阈值 r 相比较来判断字符串 a 和 b 是否匹配，如果至少有连续 r 个对应位上的符号相同则匹配，即 $Match(a, b)$ ，否则为不匹配，即 $noMatch(a, b)$ 。

例如当 $r = 4$ 时，字符串 $a=100100$ 与字符串 $b=100111$ 相互匹配，而字符串 $c=100001$ 与字符串 $d=100110$ 不匹配。但当 $r = 3$ 时，后者也相互匹配。即：

$$\begin{array}{ll} a = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} 0 & 0 & c = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} 0 & 0 & 1 \\ b = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} 1 & 1 & d = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} 1 & 1 & 0 \end{array}$$

任意两个字符串 a , b 的连续 r 位匹配的概率为^[15]:

$$P(\text{Match}_{\text{conlg}}(a, b)) = P_{\text{conlg}} \approx 2^{-r} \left(\frac{L-r}{2} + 1 \right) \quad (2.3)$$

由以上公式可知当 r 减少 1 时, 连续 r 位匹配概率 P 的增加量 ΔP 如下:

$$\Delta P_{r-\text{continue}} \approx 2^{-r} \left(\frac{L-r}{2} + 2 \right) \quad (2.4)$$

即不论取值为多少, 当 r 减少 1 时, 任意两个模式的连续 r 位匹配的概率大概增加一倍。

串的匹配我们采用 KMP 算法^[16]。这个算法不需要回溯指针, 而是利用已经得到的“部分匹配”的结果将子串向右“滑动”尽可能远的一段距后, 继续进行比较, 该算法可以简单的描述如下, 在后面的章节中将会有更加详细的论述。

一般的 KMP 算法的算法如下:

```

m = length(T) //m 为主串的长度
n = length(P) //n 为模式串的长度
i = 1; j = 1;
while(i < m, j < n)
do if(j == 0 || T[i] == P[j])
    ++i;
    ++j; //继续比较后面字符
else j = next[j];
if(j > n) return i - T[0]; //匹配成功
否则 return 0; //没有匹配成功
    
```

KMP 算法的时间复杂度是 $O(n+m)$ ^[17], 如何进一步提高匹配算法的效率, 也是目前人工免疫系统的研究热点。在下文里, 将要介绍利用并行计算的方法来加快其运算速度,

在使用连续 r 位匹配规则的情况下，并行来实现串匹配算法。通过实验结果分析，我们可以看到采用该算法，在提高效率上，起到了一定的效果。

我们下面简单地讨论一下，在使用连续 r 位匹配规则的情况下，把字符串匹配扩展到概念向量匹配的情景，这也是其可以应用的背景之一。

假设任意两个随机概念向量的匹配概率为 $P_M^{[18]}$ 并且：

M ：表示概念所含的所有属性个数；

L ：表示向量所含属性的数目，即向量的长度；

R ：表示匹配中所要求的匹配位数，即匹配长度；

对于向量的每一个位置上的属性值，从所有属性中选取与之匹配（即相同）的概率是 $1/M$ 而与之不匹配（即互补）的概率是 $(1-1/M) = (M-1)/M$ 。如果两个向量匹配，并且这种匹配是：从 L 长度向量的最左端开始，有连续 R 个对应位取值相同，则匹配的概率为：

$$(1/M) * (1/M) \dots (1/M) = M^{-R} \quad (2.5)$$

如果两个向量匹配，并且匹配的起始位置是从 L 长度向量左边的第二位到第 $(L-R+1)$ 位，那么在每次匹配成功的起始位置之前，总有不匹配发生时，其每次匹配成功的概率为： $(M-1)/M \times M^{-R}$ ，于是，两个字符串匹配的总概率 P_M 为：

$$\begin{aligned} P_M &= M^{-R} + M^{-R} \cdot \frac{M-1}{M} \cdot (L-R) \\ &= M^{-R} \cdot \left[1 + \frac{(M-1)(L-R)}{M} \right] \end{aligned} \quad (2.6)$$

下表列出了随 M 、 R 和 L 的变化，不同的匹配概率取值 P_M ：

M	R	L	P_M
6	1	2	0.296
6	1	3	0.438
6	1	4	0.576
6	1	5	0.721
6	1	6	0.861
6	2	3	0.051
6	2	4	0.074
6	2	5	0.096
6	2	6	0.120

表 2-2: 概率公式 (2.6) P_M 值
 Table2-2: Probability Formula 2.6 P_M Value

从公式可知, 对于给定的连续 r 位匹配规则, 一旦指定了参数 M 、 R 和 L , 其匹配概率 P_m 也随之确定, 所以估计对某一类型向量匹配的成功几率时, 所需的欲查询的向量规模的大小, 直接影响最终的匹配成功概率。

这里我们也可以把匹配的对象扩展到本体中的概念向量, 这也是本篇论文所能够应用的实际背景之一, 但是本文主要讨论的是如何能够快速的实现连续 r 位匹配, 所以关于本体以及本体的概念向量化的内容不做过多的介绍。

第三章 串匹配及 KMP 串匹配

搜索问题是计算机科学中最基本的问题,而串匹配就是在一个符号序列中查找另一个(或一些)符号序列的搜索问题。在现实生活中,串匹配技术的应用十分广泛,其主要应用领域包括:信息过滤、信息检索、计算生物学、入侵检测、病毒检测等等。串匹配技术的研究与发展是与实际应用息息相关的,近年来,新的应用需求对串匹配技术提出了新的要求和挑战。而且,串匹配是这些应用中最耗时的核心问题,好的串匹配算法能显著地提高应用的效率。因此,研究并设计快速的串匹配算法具有重要的理论价值和实际意义。

3.1 串匹配的定义

所谓串匹配^[19](*String Matching, or Pattern Matching*),就是给定一组特定的字符串集合 P ,对于任意的一个字符串 T ,找出 P 中的字符串在 T 中的所有出现位置。我们称 P 为模式串集合,称 P 中的元素为模式串(或关键词),称 T 为文本串。字符串中的字符都取自一个有限的符号集合 E ,简称字母表或字符集。

按照文献[20]的分类方式,串匹配可以分为四类:精确串匹配(*Exact String Matching*)、扩展串匹配(*Extended String Matching*)、近似串匹配(*Approximate String Matching*)和正则表达式匹配(*Regular Expression Matching*)。

精确串匹配要求文本中被匹配的字符串与模式串严格相等。

近似串匹配允许被匹配的字符串和模式串之间有一定的误差(*error*)。两个字符串之间的误差可以用距离来度量,常用的距离有编辑距离(*Edit Distance*)、海明距离(*Hamming Distance*)等。

正则表达式匹配中,待匹配的不是简单的字符串,而是正则表达式,这使得正则表达式匹配比精确串匹配要困难的多。

扩展串匹配介于精确串匹配和正则表达式匹配之间,它的模式串通常是一些特殊的正则表达式,因而常常可以通过特殊的手段解决。

3.2 一般的串匹配算法

一般的串匹配算法是从文本串的第一个字符和模式串的第一个字符进行比较,若相等则进一步比较二者的后续字符,否则从文本串的第二个字符起再重新和模式串的第一个字符进行比较。依次类推,直至模式串和文本串中的每一个子串相等,此时称为匹配成功,否则称为匹配失败。

一般的串匹配算法思想如下:首先将文本串 T 中的第一个字符记为 T_i 与模式串 P 中的第一个字符记为 P_1 进行比较,若不同,就将 T_2 与 P_1 进行比较,……,直到 T 的某一个字符 T_i 和 P_1 相同,再将它们之后的字符进行比较,若也相同,则如此继续往下比较,当 T 的某一个字符 T_i 与 P 的字符 P_j 不同时,则 T 返回到本趟开始字符的下一个字符,即 T_{i-j+2} , P 返回到 P_1 ,继续开始下一趟的比较,重复上述过程。若 T 中的字符全部比较完,则说明本趟匹配成功,本趟的起始位置是 $i-j+1$ 或 $i-P[0]$,否则,匹配失败。

设文本串 $T = 'ABABCACBACBAB'$, 模式串 $P = 'ABCA'$, 匹配过程如下所示:

第一趟匹配:

$T: \quad A B A B C A B C A C B A B$
 $\quad \quad = = \neq$
 $P: \quad A B C A C$

第二趟匹配:

$T: \quad A B A B C A B C A C B A B$
 $\quad \quad \neq$
 $P: \quad A B C A C$

第三趟匹配:

$T: \quad A B A B C A B C A C B A B$
 $\quad \quad = = = = \neq$
 $P: \quad A B C A C$

第四趟匹配:

T: A B A B C A B C A C B A B
 \neq
P: A B C A C

第五趟匹配:

T: A B A B C A B C A C B A B
 \neq
P: A B C A C

第六趟匹配:

T: A B A B C A B C A C B A B
 $= = = =$
P: A B C A C

一般的字符串匹配问题形式化定义如下:

在字符集 Σ 上, 给定一个长度为 N 的文本字符串 $T[1...N]$, 以及一个长度为 M 的模式字符串 $P[1...M]$ 。如果对于存在一个 S , 即 $\exists S(1 \leq S \leq N)$, 存在 $T[S+1...S+M] = P[1...M]$, 则模式串 P 在文本串 T 中匹配的位置为 S 处, 即模式串与文本串匹配成功。字符串的模式匹配问题就是要寻找模式串 P 在文本串 T 中是否出现, 以及出现的位置。

这是最简单的也是最经典的字符串的匹配问题。该算法的匹配过程易于理解, 在文本编辑等场合效率较高, 算法的时间复杂性为 $O(n+m)$ (注: n 和 m 分别为文本串和模式串的长度)。然而, 在有些情况下, 该算法的效率却很低, 时间复杂变为 $O(m * n)$ 。下面介绍一个比较好的模式匹配算法即 KMP 算法。

3.3 KMP 串匹配算法

简单的模式匹配算法匹配失败重新比较时只能向前移一个字符, 若文本串中存在和模式串只有部分匹配的多个子串, 匹配指针将多次回溯, 而回溯次数越多算法的效率越低。 KMP 匹配算法正是针对上述的不足做了实质性的改进。

KMP 算法由图灵奖获得者、对中国计算机科学事业的发展给予大力关注和支持的 *Knuth* 教授和另外两位计算机科学家 *Moms* 教授与 *Pratt* 教授发明，所以该算法被命名为 *KMP* 算法。此算法的理论基础是有限自动机理论，它通过构造失配链接函数对模式串进行预处理，在匹配检查阶段根据失配链接函数的值来判断，避免重复比较模式串中存在的相同子串，从而加快匹配速度。*KMP* 算法的设计思想十分巧妙，堪称典范。其所需的时间为 $O(n+m)$ ，是线性时间复杂度的串匹配算法。

KMP 串匹配算法的基本思想是^[21, 22, 23]：对于给定的文本串 $T[1, \dots, n]$ 与模式串 $P[1, \dots, m]$ ，假设在串匹配的进程中，执行 $T[i]$ 和 $P[j]$ 的匹配检查。若 $T[i]=P[j]$ ，则继续检查 $T[i+1]$ 和 $P[j+1]$ 是否匹配。若 $T[i] \neq P[j]$ ，则分成两种情况：

- ① 若 $j=1$ ，则模式串右移一位，检查 $T[i+1]$ 和 $P[1]$ 是否匹配；
- ② 若 $1 < j \leq m$ ，则模式串右移 $j - \text{next}(j)$ 位，检查 $T[i]$ 和 $P[\text{next}(j)]$ 是否匹配（其中 next 是根据模式串 $P[1, m]$ 的本身局部匹配的信息构造而成的，在下文中对 next 函数将会有详细描述）。重复此过程直到 $j=m$ 或 $i=n$ 结束。

还是针对上面的字符串匹配的例子来说，用 *KMP* 算法的匹配过程如下：

第一趟匹配：

```

T:  A B A B C A B C A C B A B
    = = ≠
P:  A B C A C
    
```

第二趟匹配：

```

T:  A B A B C A B C A C B A B
    = = = = ≠
P:      A B C A C
    
```

第三趟匹配：

```

T:  A B A B C A B C A C B A B
    = = = = =
P:      A B C A C
    
```

一般情况下，假设文本串为：‘ $T_1 T_2 T_3 \dots T_n$ ’，模式串为：‘ $P_1 P_2 P_3 \dots P_m$ ’，分析

可知，当匹配过程中“失配”时，模式串向右滑动可行的距离，其具体的过程如下：

假设此时文本串与模式串中第 k ($k < j$) 个字符进行比较，则模式串中前 $k-1$ 个字符的子串必须满足下面的关系式：

$$'P_1 P_2 P_3 \dots P_{k-1}' = 'S_{i-k+1} S_{i-k+2} \dots S_{i-1}' \quad (3.1)$$

而已经得到的部分匹配的结果如下：

$$'P_{j-k+1} P_{j-k+2} \dots P_{j-1}' = 'S_{i-k+1} S_{i-k+2} \dots S_{i-1}' \quad (3.2)$$

则由(3.1)，(3.2)得：

$'P_1 P_2 P_3 \dots P_{k-1}' = 'P_{j-k+1} P_{j-k+2} \dots P_{j-1}'$ 则当匹配过程中，文本串中第 i 个字符与模式串中第 j 个字符比较不相等时，只需要将模式串向右移动到模式串中第 k 个字符和文本串中第 i 个字符对齐。

令 $next[j]=k$ ，则 $next[j]$ 表明当模式串中第 j 个字符与文本串中相应的字符失配时，在模式串中需重新和文本串中该字符比较时字符的位置 $[j]$ 。则 $next$ 的函数定义如下：

$$next[j] = \begin{cases} 0 & j=1 \\ \max\{k \mid 1 < k < j \text{ 且 } 'P_1 P_2 \dots P_{k-1}' = 'P_{j-k+1} P_{j-k+2} \dots P_{j-1}'\} & \\ 1 & \text{其它情况} \end{cases} \quad (3.3)$$

因此，基于以上定义的 $next$ 函数给出 KMP 串匹配算法。

文本串： $T: T_1 T_2 \dots T_n$

模式串： $P: P_1 P_2 \dots P_m$

func index-KMP(S,P:stdlp):Integer;

*/*利用模式串的 next 函数求模式串 P 在文本串 S 位置的 KMP 算法*/*

i ← 1; j ← 1; / 指针初始化*/*

While (i ≤ T, curlen) and (j ≤ P, curlen) do

If (j=0) or (T.ch[i]=P.ch[j])

*then {i ← i+1; j ← j+1} /*继续下一对属性值的比对*/*

```

Else  $j \leftarrow \text{next}[j]$  /*子串向右滑动*/

if  $j > P.\text{curlen}$  then

return( $i - l.\text{curlen}$ ) /*匹配*成功/

else return(0)

endif;

Proc get-next ( $P:\text{string}; \text{var next}: \text{array}[1,P.\text{en}]$  or integer):

/*求模式串  $P$  的 next 函数值并存入数组  $\text{next}$ */

 $j \leftarrow 1, k \leftarrow 0;$ 

 $\text{next}[1] \leftarrow 0;$  /*初始化*/

while  $j < P.\text{curlen}$  do

if ( $k=0$ ) or ( $P.\text{ch}[j]=P.\text{ch}[k]$ )

then { $l=j+1; k=k+1; \text{next}[j] \leftarrow k$ }

else  $k \leftarrow \text{next}[k]$ 

endp; /*get-next*/

```

第四章 基于人工免疫的串匹配并行算法

4.1 分布式环境与并行算法

在分布式环境下,基于人工免疫的算法具有并行性、健壮性。本节将要给出下文所设计的串匹配并行算法的生物基础。

免疫系统由许多局部相互作用的基本单元组成来提供全局的保护,没有集中控制。免疫系统具有分布性的一个原因是它必须应答的抗原是散步在整个免疫体内,效应细胞检测抗原的过程就是分布式检测。

免疫系统由分布在机体各个部分的细胞、组织和器官等组成。免疫系统的分布式特性首先取决与抗原的分布式特征,即抗原是分散在机体内部的;其次免疫系统的分布式特性有利于加强系统的健壮性,从而使得免疫系统不会因为局部组织损伤而使整个功能受到很大影响。分散于机体各部分的淋巴细胞采用学习的方式实现对特定抗原的识别。同时,通过分化效应细胞和记忆细胞分别实现对抗原的有效清除和记忆信息保留。这个过程实际上是一个适应性的应答过程。由于免疫应答机制是通过局部细胞的交互作用来完成的,不存在集中控制,所以系统的分布式进一步强化了其自适应特性^[1,24]。

由于工作载荷分布在不同的多个工作单元上,系统的工作效率得到有效提高;同时,其分布式特性还可以减少由局部工作单元失效所引起的对系统整体的不利影响。

免疫系统的分布性由于其潜在的效率和错误耐受而受到计算机科学家的欢迎;但是,除了一些特殊的环境,如计算机网络安全系统之外,这种性质在具体的串匹配算法中得到应用的还不是很多。本文作者这是注意到了这个情况,所以力争在这方面有所突破。

4.1.1 分布式环境

从广义上看,任何一种计算机网络都是一种分布式计算机平台。机群并行计算系统是一种基于网络的并行计算系统。它将若干独立的计算机系统通过高速通信网络互联起来以支持并行计算,具有如下特点:

① 易于实现。只需将现有的计算机通过高速通信网络互联起来即可实现

- ② 可伸缩性强。在现有网络上增加新的计算机即可提高机群并行系统的处理能力。
- ③ 平台无关性。可以将各种不同体系结构的计算机互联起来构成一个异构并行计算环境。
- ④ 可重用性。因为机群系统中的结点都是通用计算机，所以可以充分利用原有的成熟程序代码进行并行程序设计。
- ⑤ 输入输出高度并行。可以利用数据分布技术，充分发挥机群系统中输入输出的并行操作性能。
- ⑥ 性能/价格比高。

机群并行计算系统的不足之处是，连接入系统中的结点达到一定规模时，系统可能需要花费大量的时间进行通信，从而降低了并行效率。

4.1.2 并行算法基础

算法是指求解问题的步骤，它包含一组有穷的规则，这些规则规定了解决某一特定类型问题的系列运算。

所谓并行算法^[25]是指适合于在各种并行计算机上求解问题的算法。它是一些可同时执行的诸进程的集合，这些进程互相作用和协调工作，从而达到对给定问题的求解。我们可以从不同的角度将并行算法分为数值并行算法和非数值并行算法；同步并行算法、异步并行算法和分布式并行算法；SIMD 并行算法、MIMD 并行算法和 VLSI 并行算法等。

4.1.3 程序并行性的条件

在实际应用中，要使其中的若干个程序段能够并行执行，必须使得每段程序与其他各段程序无关。相关性主要是指数据相关性、控制相关性和资源相关性。1966 年，Bernstein 教授提出了一种条件集合的概念。在这种条件集合的基础上，可以并行执行两个进程，

其定义为：进行 P_i 的输入集合 I_i ， I_i 为执行进程 P_i 所需的全部输入变量的集合，而 O_i 是由进程 P_i 输出的集合，它由程序运行后所产生的全部输出变量组成。对于两个进程 P_1 和 P_2 而言，如果此两个进程互不相关，并且不会产生混沌的输出，那么可以将这些条件形式化表示成如下等式(称为 Bernstein 条件)

$$I_1 \cap O_2 = \Phi$$

$$I_2 \cap O_1 = \Phi$$

$$O_1 \cap O_2 = \Phi$$

对于多个进程, 如果 *Bernstein* 条件能两两成对成立, 那么进程集合 $P_1 P_2 P_3, \dots, P_k$ 就能并行执行。

4.2 并行算法的基本设计技术

并行算法的设计有一些基本技术可供参考, 而且并行算法的部分设计技术直接借用了顺序算法的设计思想。以下是一些常用的、重要的并行算法设计技术^[26]。

- ① 分治策略: 首先将原问题分解成若干个规模相当的子问题, 然后各处理器并行递归的求解这些子问题以获得子解, 最后并行的归并各子问题的解形成原问题的解。这种方法的关键是如何有效地完成并行归并子解的工作, 以确保归并工作所花费的代价不大于并行求解子问题所需的工作量。
- ② 平衡树方法: 将输入元素作为叶节点构筑平衡二叉树, 然后自叶节点向树根节点逐层处理, 同一层上的各节点并行处理, 直到树根节点为止, 这样即可获得问题的解。
- ③ 倍增技术: 反复地将计算问题分解成规模相同的两个子问题, 使得算法所需处理的数据之间的距离(下标值)逐步加倍、子问题的个数逐步倍增, 这样经过迭代之后, 即可完成一定距离(规模)的所有数据处理并获得问题的解。
- ④ 划分设计技术: 就是将原始问题分成若干个部分, 然后各部分由相应的处理器同时执行。用划分设计法求解问题可分为两步: 一是将给定的问题分成 P 个相互独立的几乎等尺寸的子问题; 二是用 P 台处理器并行求诸子问题。划分时的关键在于如何将问题进行分组, 使得子问题较容易并行求解, 或者子问题的解比较容易被组合成原问题的解。本文就是利用划分设计技术的思想设计并行算法的。
- ⑤ 流水线技术: 其基本思想是将一个计算任务 T 分成 k 个子任务 $T_1, T_2, T_3, \dots, T_k$, 使得一旦完成 T_i 子任务的求解, 则后续的其他子任务就可以立即开始并以同样速率的流水线方式进行计算。

4.3 基于连续 r 位匹配规则的并行串匹配思想

在分布式环境下,充分利用网络中的主机来共同并行完成对字符串的匹配,会有效地提高匹配的性能与效率。

本节主要介绍将免疫系统中的连续 r 位匹配规则算法嵌入到并行 KMP 串匹配算法当中去,以及相应的算法描述。

将长为 n 的文本串 T 根据处理器的个数 m 相应的分成 m 段,分布于处理器 0 到 $m-1$ 中,使得其相邻的文本段分布在相邻的处理器上。文本串 T 每一段划分的规则是,首先考虑是否能够均匀划分成互不重叠的 m 段,如果可以则按此划分,每个处理器得到文本串的长度为 n/m 。如果无法均匀划分,那么除最后一个处理器以外的其他处理器均匀划分,最后一个处理器的文本串使用末尾补上特殊字符的方法使其长度与其它字符串长度相同。

如果匹配过程中某个局部段(最后一段除外)需要跨段才能找到匹配位置,那么这种情况如何解决呢?假设某匹配过程每个处理器分到的文本串的长度为 n ,模式串 P 长度为 m ,匹配阈值为 r ,那么可以使用如下的方法来解决上面提出的问题。把每个处理器(最后一个处理器除外)的本局部段的段尾 $r-1$ 个字符传送给下一处理器,下一处理器接收到前一个处理器传来的字符串后,再接合本段的段首 $r-1$ 个字符构成一个长为 $2(r-1)$ 的段间字符串,对此字符串做匹配,就能找到所有段间匹配位置。

在匹配过程中,由主节点机播送各种必要参数其中包括匹配阈值 r 。在运算过程中每一个节点机上运行的部分串匹配都要符合连续 r 位匹配规则。

子节点机利用 *Linux* 系统的网络文件系统(*Network File System NFS*)通过“*mount -a*”命令将主节点机上的文件挂载到各自的子节点机上,这样子节点机就能够很方便的共享主节点机上的连续 r 位匹配规则代码段。

运算结束后,各自的处理器把运算结果返回给主节点机,这时主节点机有一个检测匹配结果的过程。主节点机根据子节点机传回的匹配位置,在文本串中找到相应的位置,应用连续 r 位匹配规则检测匹配结果是否符合要求,如果正确则打印结果,如果错误,则把相应文本段返回给处理器,重新运算。

4.4 并行实现串匹配算法

4.4.1 MPI 简介

并行程序日益增强的简易性和灵活性增加了它对于科学家或工程师的吸引力。在众多的并行程序库中,应用较广泛的有消息传递接口(MPI)标准和并行虚拟机(PVM)环境,PVM是Oak Ridge国家实验室和Tennessee大学在1989年开发出来的;MPI即是在数年后出现,由几所大学和国家实验室出于创建消息传递库标准的目的共同开发成功。MPI是一种并行程序的消息传递编程模型,是一个库而不是一种语言。MPI(Message Passing Interface)既提供高效的点对点的通讯,还提供高效的群组通讯。这对于基于并行机群的软件开发来说是十分有利的。在消息传递库方法的并行编程中,一组进程所执行的程序是用标准串行语言书写的代码加上用于消息接收和发送的库函数来调用的,因此,它实际上是一个消息传递函数库的标准说明,它吸取了众多消息传递系统的优点,是目前国际上最流行的并行编程环境之一,尤其是分布式存储的可缩放并行计算机和 workstation 网络以及机群的一种编程范例。因而为并行软件产业的增长提供了必要的条件^[27]。

MPI 的特点概括起来有以下五个方面^[28]:

- ① 通用性: MPI 是可移植的标准平台,其通讯单元包含上下文和组的信息,以保证消息传递的安全性。
- ② 点对点通讯: MPI 能有效地管理消息、缓存区,具有结构化缓存,扩充数据类型及异构性, MPI 异步执行时能保护用户的其他软件不受影响,能实现完全的异步通信。立即发送与接受可完全与计算同步进行。
- ③ 数据汇集方式: 具有内定和用户自定义的数据汇集操作方式,可对大量数据进行整体传输,可直接或依据拓扑结构定义组。
- ④ MPI 的实现方式多样化: 同一编程界面可有多种开发工具,具有面向应用的信息传递拓扑结构: 内定义支持网络和图拓扑结构。
- ⑤ 良好的操作环境: 具有差错控制功能。MPI2 在 I/O、主动消息、进程启动、动态进程控制等方面有进一步改善, MPICH 版本可以在 Linux 的 X 窗口使用,用户通过 Upshot 或 Nupshot 直观地考察 MPI 运行过程中各处理器之间的同步、计算、消息发送、接收等情况,从而为程序的修改提供依据。

MPI 的基本通信机制是在一对进程(一方发送、另一方接收)之间传递数据,称之为

点对点通信(*Point-to-Point Communication*),它是 *MPI* 中比较复杂的一部分,几乎所有的 *MPI* 结构都是围绕着点对点来设计的。其数据传送有阻塞和非阻塞两组机制:对于阻塞方式,它必须等到消息从本地送出之后才可以执行后续的语句,保证了缓冲区等资源的可再用性;对于非阻塞方式,它不需等到消息从本地送出就可以执行后续的语句,从而允许通信和计算的重叠,但非阻塞调用的返回并不保证资源的可再用性。

MPI 共有 4 种通信模式^[27]。标准通信模式(*standard communication mode*),缓存通信模式(*buffered communication mode*),同步通信模式(*synchronous communication mode*)和就绪通信模式(*ready communication mode*)。

- ① 标准通信模式(*standard communication mode*)发送:*MPI_SEND*,接收:*MPI_RECV* 在 *MPI* 采用标准通信模式时,是否对发送的数据进行缓存是由 *MPI* 自身决定的,而不是程序员来控制的。如果 *MPI* 缓存要发送数据,发送操作不管接收操作是否执行都可以进行执行,且正确返回,而不要求接收操作收到数据,但缓存数据会延长数据通信时间,且不一定得到缓冲区;如果 *MPI* 不缓存数据,则要求相应的接收操作执行,且发送数据完全到达接收缓冲区,发送操作才完成。

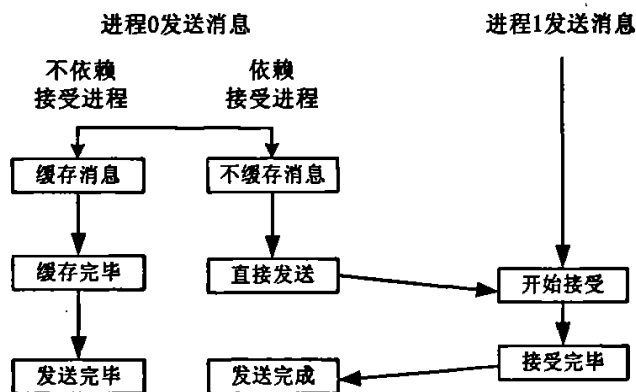


图 4-1 标准通信模式

Fig 4.1 standard communication mode

- ② 缓存通信模式(*buffered communication mode*)发送:*MPI_BSEND* 在这种模式下,用户可直接对通信缓冲区进行申请、使用和释放。它不管接收操作是否启动,发送操作都可以执行,但是在发送消息之前必须有缓冲区可用。

对于非阻塞发送，正确退出并不意味着缓冲区可被其它操作使用，但阻塞发送返回后其缓冲区是可以使用的。

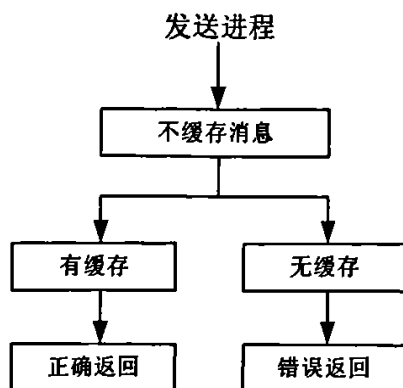


图 4-2 缓存通信模式

Fig4-2 buffered communication mode

- ③ 同步通信模式 (*synchronous communication mode*) 发送: `MPI_SSEND`
- 同步通信模式的开始不依赖于接收进程相应的接收操作是否已经启动，但是同步发送必须等到相应的接收进程开始后才可正确返回。因此，同步发送返回后，意味着发送缓冲区中的数据已经全部被系统缓冲区缓存，且已经开始发送。这样同步发送返回后，发送缓冲区可以被释放或重新使用。

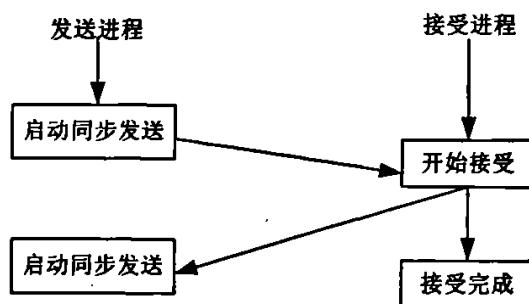


图 4-3 同步通信模式

Fig 4-3 synchronous communication mode

- ④ 就绪通信模式 (*ready communication mode*) 发送: `MPI_RSEND`
- 就绪通信模式中，只有当接收进程的接收操作已经启动时，才可以在发送进程启动发送操作，否则，当发送操作启动而相应的接收还没有启动时，发送操作将出错。对于非阻塞发送操作的正确返回，并不意味着发送已完成，但对于阻

塞发送的正确返回，则发送缓冲区可以重复使用。

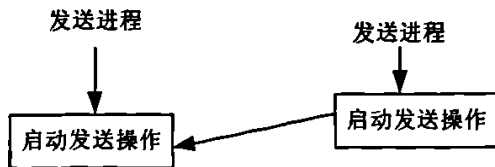


图 4-4 就绪通信模式

Fig 4-4 ready communication mode

以上是对 MPI 4 种通讯模式的简单介绍，下面我们了解一下对于某一并行 MPI 程序来说其执行的步骤一般为：

- ① 编译源程序得到 MPI 可执行程序；
- ② 将可执行程序广播到各个节点机上；
- ③ 通过 *mpirun* 命令并行执行该 MPI 程序；

图 4-5 描述了 MPI 程序的执行过程。

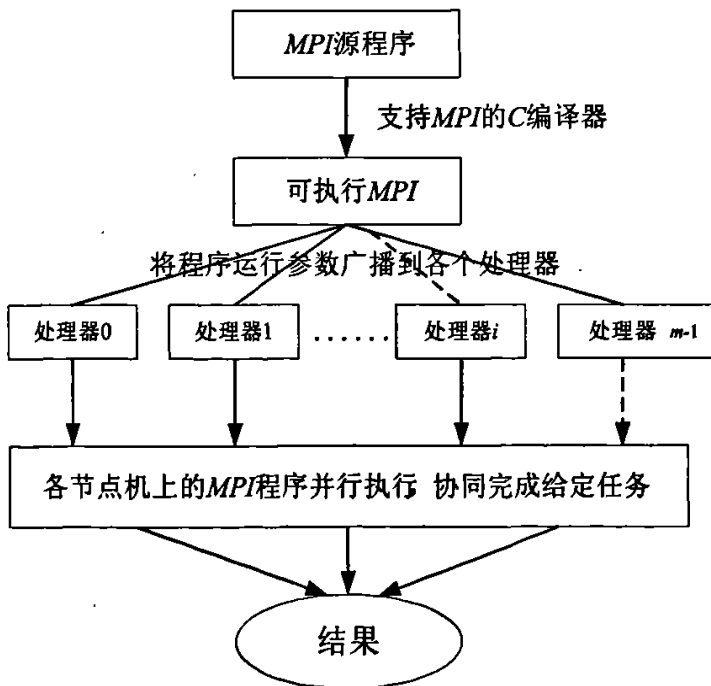


图 4-5 MPI 程序的执行过程

Fig 4-5 MPI execution procedure

4.4.2 算法的实现

首先，我们通过下图 4-6 了解一下整体算法流程图：

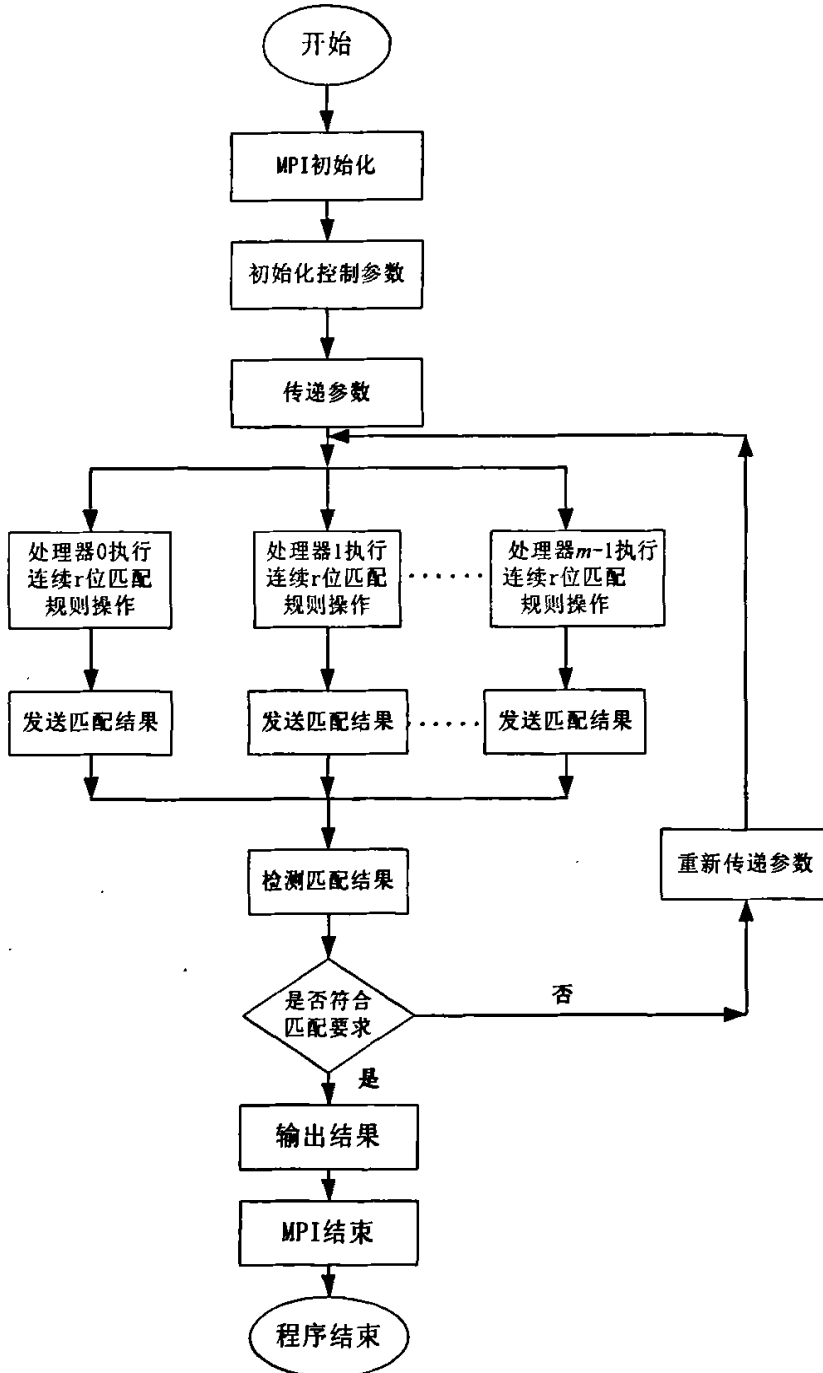


图 4-6 算法流程图

Fig 4-6 algorithm flow chart

在前述的算法思想以及对其并行实现设计分析的基础上，我们给出并行串匹配算法。

设： a, b 是长度为 L 的二进制字符串； r 是匹配阈值； $match(a, b)$ 为匹配函数，并有： $match(a, b) = Match_{contig}$ ； \oplus 是异或操作(XOR)。

算法 1：连续 r 位匹配规则算法

输入：匹配阈值 r ，字符串 a, b

输出： $Match_{contig}$

procedure r -CONTIGUOUS-MATCH(r, a, b)

BEGIN

$c = 0$;

$x = 0$; /* x 是长度为 L 的二进制字符串 */

$match(a, b) = 0$;

对于任意长度为 L 的二进制字符串 a 和 b

if $r = 0$ **then**

$match(a, b) = 1$;

if $L \geq r$ **then**

{

$L \leftarrow \text{length}[a]$;

$x \leftarrow a \oplus b$;

for $i = 1$ **to** L

{

if $x[j] = 1$ **then**

$c = 0$;

$match(a, b) = 0$;

else

{

$c = c + 1$;

if $c = r$ **then**

$match(a, b) = 1$;

break;

```

    }
    }
}

```

END

算法 2 KMP 串行算法

输入：文本串 T 和模式串 P

输出：所有的匹配位置

```

procedure KMP( $T, P, \text{textlen}, \text{pattlen}$ )
  Begin
     $i=1$ 
     $j=\text{matched\_num}+1$ 
    while  $i \leq \text{textlen}$  do
      while  $j \neq 0$  and  $P[j] \neq T[i]$  do
        {
           $j=\text{next}[j]$ 
           $a=T;$ 
           $b=P;$ 
           $r\text{-CONTIGUOUS-MATCH}(r,a,b)$ 
        }
      end while
      if  $j=m$ 
         $\text{match}[i-(m-1)]=1$ 
         $j=\text{next}[m+1]$ 
         $i=i+1$ 
      else
         $j=j+1$ 
         $i=i+1$ 
      end if
    end while

```

$maxprefixlen=j-1$

End

算法 3 $next$ 函数的计算算法

输入: 模式串 $P[1, m]$

输出: $next[1, m]$

procedure NEXT

Begin

$next[1] = 0$

$j=2$

while $j \leq m+1$ do

$i=next[j-1]$

 while($i \neq 0$ and $W[i] \neq W[j-1]$) do

$i=next[i]$

 end while

$next[j]=i+1$

 if $j \neq m+1$

 if $W[j] \neq W[i+1]$

$next[j]=i+1$

 else

$next[j] = next[i+1]$

 end if

 end if

$j=j+1$

end while

End

算法 4 并行 KMP 串匹配算法:

输入: 分布存储的文本串 T 和模式串 P

输出: 匹配结果

Begin

PE_0 broadcast $r, period_len, period_num, period_suffixlen$ to other processors /* 播送匹配阈值 r 、模式串 P 长度 */

PE_0 broadcast $P[1, period_len]$

If $period_num=1$

then

broadcast $newnext[1, m]$

else

broadcast $newnext[1, 2*period_len]$

end if

/*播送模式串 P 的部分 $next$ 函数, */

PE_0 call procedure NEXT /*处理器 PE_0 通过调用算法 3, 求模式串 P 的 $next$ 函数*/

for $i=0$ to $p-2$ *par-do* /*处理器 PE_i 把 $maxprefixlen$ 发送给处理器 PE_{i+1} */

PE_i send $maxprefixlen$ from PE_{i+1}

end for

for $i=1$ to $p-1$ *par-do* /*处理器 PE_i 接收 PE_{i-1} 发送来的 $maxprefixlen$, 调用 KMP 作段间匹配*/

PE_i receive $maxprefixlen$ from PE_{i-1}

PE_i call procedure KMP($T, P, textlen, pattlen$)

end for

for $i=1$ to $p-1$ *par-do* /*由传送来的模式串 P 的周期和部分 $next$ 函数重构整个模式串*/

call procedure REBUILD

end for

```

for i=1 to p-1 par-do
    KMP(T, P, textlen, pattlen)/*各处理器调用算法 2，做局部段匹配， */
    r-CONTIGUOUS-MATCH(r,a,b)/*处理器 0 调用算法 1，做全局匹配检测， */
    if Matchcong = 0
        重新传送参数，重新运行匹配过程。
    else
        end for
END

```


第五章 并行计算及 *Cluster* 机群

通过对免疫匹配规则的分析,将免疫系统中的连续 r 位匹配规则应用并行计算来实现,是免疫系统与并行计算领域相结合的一种尝试和应用。并行计算的方法很自然的应用到了免疫系统具有天然的并行性这个特征,从而更好的发挥免疫系统在字符串匹配中的应用。

5.1 并行计算概述

并行计算机的发展基于人们在两方面的认识^[29]:第一,单机性能不可能满足大规模科学与工程问题的计算需求,而并行计算机是实现高性能计算、解决挑战性计算问题的唯一途径;第二,同时性和并行性是物质世界的一种普遍属性,具有实际物理背景的计算问题在许多情况下都可以划分成能够并行计算的多个子任务。针对某一具体应用问题,我们可以利用它们内部的并行性,设计并行算法,将其分解成为互相独立但彼此又有一定联系的若干个子问题,分别交给各个处理机来运算,而所有的处理机按并行算法完成初始应用问题的求解。根据几十个常用应用软件的统计,60%-80%的标量计算可以被向量化,而90%左右的串行计算可以并行化。

实现或提高计算机系统的并行性,可以通过时间重叠、资源重复和资源共享等技术途径来实现^[30]。

时间重叠(*Time Superposition*)是在并行性概念中引入时间因素,让多个处理过程在时间上相互错开,轮流重叠的使用同一套硬件设备的各个部分,以加快硬件周转而赢得速度。

资源重复(*Resource Replication*)是在并行概念引入空间因素,通过重复设置硬件资源来提高可靠性或并行性。在结构上,采用多操作部件和多存储部件。早期受限于硬件价格,资源重复是以提高可靠性为主,随着硬件价格的减低,资源重复利用被大量用于提高系统的速度性能,成为提高并行性的重要方面。

资源共享(*Resource Sharing*)是按时间顺序轮流地使用同一套资源,包括CPU、主存、

外设硬件资源和软件、信息资源，以提高其利用率，从而提高整个系统的性能。

5.2 并行计算机分类

并行计算机即并行计算环境，是并行算法赖以生存的物质基础，它们的发展直接影响着并行算法的设计和实现^[31]。

传统的 *Von Neumann* 结构计算机的中心是一台以串行方式操作的主处理器。由于现代工艺的发展使得元器件的速度接近极限，而社会发展对超大规模计算的需求却与日俱增，因而它们之间的矛盾导致了以后并行机的蓬勃发展。

M.J.Flynn 提出了著名的 *Flynn*^[26]分类法，根据指令流与数据流方式的不同将计算机系统分类。指令流是指机器执行的指令序列；数据流是指指令调用的数据序列，包括输入数据和中间结果；据此，可以把计算机系统分成以下四类：

- ① 单指令流单数据流 *SISD* (*Single Instruction Stream Single Data Stream*);
- ② 单指令流多数据流 *SIMD* (*Single Instruction Stream Multiple Data Stream*);
- ③ 多指令流单数据流 *MISD* (*Multiple Instruction Stream Single Data Stream*);
- ④ 多指令流多数据流 *MIMD* (*Multiple Instruction Stream Multiple Data Stream*);

SISD 计算机就是传统的单处理器计算机，按指令顺序执行，一次只执行一条指令并且只对一个操作部分分配数据。直到目前，还很少见到 *MISD* 类型的计算机。*SIMD* 型和 *MIMD* 型并行计算机是研究和开发的主流。

5.2.1 向量处理 *SIMD* 型计算机

SIMD(*Single Instruction and Multiple Data*)机器是指单指令流多数据流并行机，也即指系统中各功能部件或处理机对多组数据执行相同的指令流或操作。*SIMD* 机器在任何时刻只有一条指令在执行。所以该类计算机的主要特征是，同步的、确定的。它适合于指令/操作级并行。

向量处理机、阵列处理机等均属于此类并行机，包括 *DAP*, *CM-2*, *MasPar MP-1*、*MasPar MP-2*, *Cray-1*, *Convex C1* 和国产 *YH-1* 等。

在并行机发展的初期，*SIMD* 类型并行机对其发展起到了重要的推动作用，但随着微处理芯片技术的发展，90 年代以后，单处理机性能得到了极大地提高，也因此导致并行机朝 *MIMD* 方向发展，*SIMD* 并行机基本退出了历史舞台。

5.2.2 共享存储 MIMD 并行多处理机

MIMD(Multiple Instruction and Multiple Data)并行机是指多指令流多数据流并行机,也即指系统中的各处理机在各自唯一的数据流上执行各自的指令流,与其它处理机无关。MIMD 的一个特例是单程序、多数据(SPMD, Single Program and Multiple Data)计算^[32],即所有处理机执行同一程序,而由进程指标加以参数化,从而完成对不同数据的操作(SPMD 也是目前并行算法实现中普遍采用的编程模式)。

MIMD 并行机按内存分布和访问方式的不同又可分为三种:共享存储 MIMD 并行机、分布存储 MIMD 并行多处理机、分布共享存储 MIMD 并行机。其中的共享存储 MIMD 并行处理机是指多台处理机通过互连网络共享一个统一的内存空间或多个存储器模块,各处理机可直接访问所有的数据,通过共享内存实现处理机间的通信和协调。

例如:下图给出了共享多个存储模块的 MIMD 并行机结构。

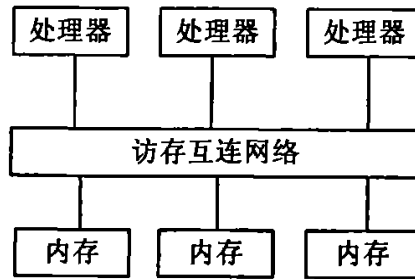


图 5-1 MIMD 并行机结构

Fig.5-1 MIMD Structure

图中的处理机可以是标量处理器(如 SGI Power Challenge)也可以是向量处理器(如: Cray X-MP, Cray Y-MP, Fujitsu VPP500, YH-2 等)。若各处理机完全等价,则又称之为对称多处理机(SMP, Symmetric Multiple Processors)。对称多处理机是最常见的多处理计算机系统,是目前网络服务器广泛采用的系统结构。

共享存储 MIMD 并行机中的处理机可通过总线互连、交叉开关、多级互连网络等方式连接存储模块。由于存储系统共享,各处理机间的协调通过共享内存实现,因而它的并行程序设计较简单(基本上相当于传统的多线程编程)。但是,这类计算机有一个无法突破的瓶颈—即处理机与存储器系统间存在严重的访存冲突。因此,共享存储 MIMD 并行机的可扩展性较差,比较适合中小规模应用问题的计算和事务处理。目前国际上流行的 SMP 一般都是 4-8 个 CPU,有少数是 16 个 CPU 的。

5.2.3 分布存储 MIMD 并行多处理机

为了突破共享存储并行机系统中处理器与内存系统间的瓶颈,分布存储 MIMD 并行多处理机器系统应运而生。该系统中每台处理机有自己的局部存储器,构成一个单独的节点,节点之间通过互连网相互连接。每台处理机只能直接访问局部内存,不能访问其它处理机的存储器,它们之间的协调以消息传递的方式进行。

与共享存储器并行机比较,分布式存储并行机具有很好的可扩展性,可以最大限度地增加处理机的数量,是目前实现超大规模科学与工程计算的唯一途径。但由于它的每个节点机需要依赖消息传递来相互通信,而消息传递对编程者是不透明的,所以,分布存储并行多处理机系统的编程较共享存储复杂。

分布存储 MIMD 并行多处理机其主要有 MPP 系统和 Cluster 系统两种形式^[33]。

MPP (Massively Parallel Processors) 系统是常见的并行系统,由成百上千的功能相同的处理机通过互连网络连接而成,且往往都采用专用结构。MPP 的优点是拥有良好的伸缩性,只要为系统增加更多的结点,便能增加系统的计算峰值,如 Inter Paragon XP/S, Cray T3D, 国产 YH-3 和曙光 1000 都是 MPP 并行系统。但是, MPP 的缺点是较难为其开发并程序,程序员需要根据系统的特点来平衡并程序的粒度和结点间通信量。

Cluster 计算机集群也叫计算机机群。是指把两个以上高性能的工作站或高档次 PC 机用互连网连接在一起,并配以相应的支撑软件,构成一个分布式并行计算机系统,结点间的消息传递和程序执行均是并行化的。Cluster 机群具有并行计算性能而且可以加速作业的执行,这是和一般的网络系统不同的。Cluster 的互连大多采用通用局部网,如 Ethernet, FDDI, Myrinet, ATM 等。集群系统包括工作站群 NOW (Network of Workstations), PC 机群、全对称 SMP 机群等。

5.3 Cluster 与 MPP 的区别

基于以上的描述,可以看出 Cluster 系统与 MPP 系统相比,它们除具有分布式存储系统的共同特点外,主要不同点表现在以下几个方面^[34]。

- Cluster 体系中的每一个节点都是一个完整的计算机。
- Cluster 体系中的节点机往往依赖于现有的、成熟的技术,如常见的工作站或 PC 机等,运行 UNIX, Windows NT 等成熟的工业标准操作系统,有一大批现有

的应用(单节点运行, 要并行化按消息传递的方式重新编写或做一定修改)。

- 典型的 *Cluster* 体系常常依赖于通用的网络体系(如以太网等), 这样节点机之间的通信效率要低于专用的 *MPP* 体系, 因此 *Cluster* 体系比较适用于中、粗粒度的并行。

此外, *Cluster* 与 *MPP* 相比, 具有以下优点:

- 系统开发周期短: 由于 *Cluster* 体系的节点机、操作系统、网络通信全采用成熟的技术, 因而节省了大量研制时间。
- 用户投资风险小: 机群系统的每个节点都是一台独立的工作站或高档 *PC* 机, 相比于 *MPP* 系统而言, *Cluster* 系统性能的发挥可得到保证, 避免资金的浪费。
- 系统性价比高: 传统巨型机或 *MPP* 的价格都比较昂贵, 工作站或高档 *PC* 机由于是批量生产出来的, 因而售价较低, 且由近十台或几十台工作站组成的机群系统可以满足多数应用的需求, 所以 *Cluster* 系统的性价比较 *MPP* 系统高。
- 节约系统资源: 可以充分利用现有设备, 将不同体系结构、不同性能的工作站连在一起, 包括原有的一些性能较低或型号较旧的机器在群集系统中仍可发挥作用。
- 可扩展性好: 从规模上说, 群集系统大多使用通用网络系统扩展容易; 从性能上说, 对大多数中、粗粒度的并行应用都有较高的效率。
- 系统容错性好: *Cluster* 的一个重要发展方向是作为应用服务器、数据库服务器等等。因此, *Cluster* 体系在软件的功能上下了很多工夫, 一个重要的贡献就是失效切换技术, 它是指当系统中的一个节点出错时, 这个节点上的任务可转移到其它节点上继续运行, 用户本身感觉不到这种变化。这也因此导致这种机型的发展前途非常之好, 特别是可以用作超级服务器或服务器集群。
- 应用面较广: 由于 *Cluster* 是建立在成熟的技术基础之上的, 因此, 与 *MPP* 系统相比, *Cluster* 往往能获得更广泛的应用。

显然, 机群系统在性能价格比 (*Cost/Performance*)、可扩展性(*Scalability*)、可用性(*Availability*)等方面都显示出了很强的竞争力, 尤其是它对现有单机上的软、硬件产品

继承和对商用软、硬件最新研究成果的快速运用,从两方面表现出传统 *MPP* 无法比拟的优势。

目前,机群系统已在许多领域获得应用^[35]。可以预见,随着 *SMP* 产品的大量使用和高性能网络产品的完善以及各种软硬件支持的增多和系统软件、应用软件的丰富,新一代高性能集群系统必将成为未来高性能计算领域、商务计算和网络信息服务的主流平台之一,未来的超级服务器将大量采用群集技术。早在 1997 年 5 月《*Scientific American*》就已经指出世界上最快的计算机将是 *Internet* 或是由网络连接起来的计算机机群。基于它的巨大市场份额,世界上众多的商用系统供应商,包括 *IBM*, *DEC*, *Microsoft* 等均已致力于群集系统的研究和开发工作。

综上所述,在本论文的实验中,将采用 *Cluster* 机群技术,并行实现串匹配的过程。下面详细的描述一下如何在已有的实验设备情况下搭建机群的过程。

5.4 *Cluster* 机群环境构建

5.4.1 硬件部署

由于本系统是一个小型机群系统,因此将用户节点、控制节点、管理节点、存储节点和安装节点都设定为同一台计算机,这台计算机称为主节点,同时,主节点也参与运算。

由于在并行计算的过程中,大量的计算任务是由主节点(*Master Node*)分配至各子节点(*Slave Node*)。主节点在分配的过程中,是按节点数来进行分配的,即给每个节点分配基本相同的计算工作量。各节点的计算结果又将返回到主节点。

如果计算节点有的运算速度快,有的运算速度慢,那么速度快的节点运算完后,节点出现资源闲置,整个系统会等待速度慢的节点计算完毕才能得到最终结果。系统的运算速度接近于慢节点的速度,降低了系统效率。因此,在计算节点的选用上应尽量采用相同配置的 *PC* 机。而主节点除了要作网络服务器外,还作为计算节点参与运算。因此,主节点机应选用配置高于计算节点的 *PC* 机。

本机群系统主节点配置为: *CPU Intel P4 1.8GHz*, 内存 *768M*, 硬盘 *80G*

子节点配置为: *CPU Intel P3 1.0GHz* (双 *CPU*), 内存 *512M*, 硬盘 *80G* (三台)

主、从节点硬件连接如图所示:

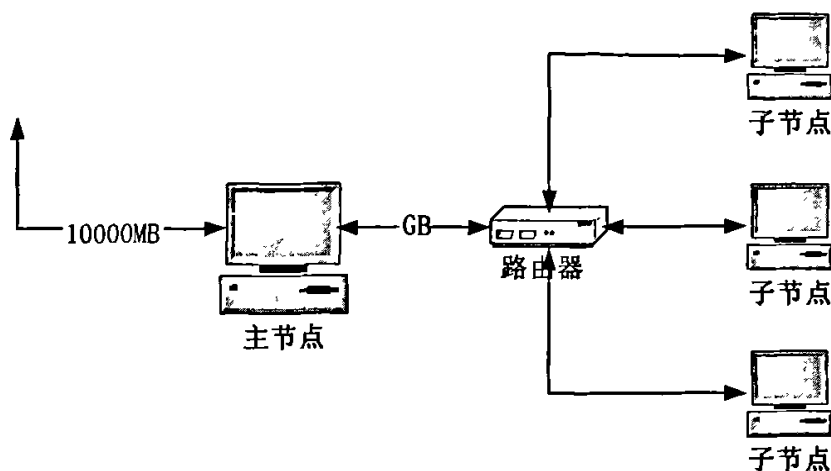


图 5-2 Cluster 机群结构
Fig.5-2 Cluster Structure

5.4.2 软件部署

所有节点的操作系统均采用 *Linux REDHAT 9.0* 内核 2.4.20。各节点为了支持日常的独立使用，可同时安装 *Windows* 操作系统。*Linux* 的安装在此不再赘述，下面分别介绍主节点和子节点的配置方法。

主节点配置：

- (1) 建立一个共享目录：在 */home* 目录下建立一个共享目录 *cluster*。
 - (2) 打开“新建终端”，敲入 *setup* 命令，然后选择“*Authentication Configuration*”，选择 *NIS*，填入 *NIS* 域名：*PCCLUSTER*；在服务器名一栏上敲入本机 IP 地址。
 - (3) 初始化数据库：运行命令：*/usr/lib/yp/ypinit -m* 在出现的画面上直接敲入：
Ctrl+D
 - (4) 在 */etc/exports* 目录下填入：
- | | | |
|----------------------|-----------------------|-------------|
| <i>/home/cluster</i> | <i>219.226.86.105</i> | <i>(rw)</i> |
| <i>/home/cluster</i> | <i>219.226.86.106</i> | <i>(rw)</i> |
| <i>/home/cluster</i> | <i>219.226.86.107</i> | <i>(rw)</i> |
| <i>/home/cluster</i> | <i>219.226.86.108</i> | <i>(rw)</i> |

说明：主节点主机名为：*tyut0* IP 地址为：*219.226.86.105*

节点机 IP 地址 主机名

219.226.86.105 *tyut0*

节点机 IP 地址 主机名

219.226.86.106 tyut1

219.226.86.107 tyut2

219.226.86.108 tyut3

(5) 在/etc/hosts 目录下填入:

219.226.86.105 tyut0

219.226.86.106 tyut1

219.226.86.107 tyut2

219.226.86.108 tyut3

(6) 在/etc/hosts.equiv 目录下填入: (注意: /etc 下面如没有此目录, 需要自己建立)

tyut0

tyut1

tyut2

tyut3

(7) 在/etc/xinetd.d 目录下 找到 rexec rlogin rsh 这三个文件依次打开, 将三个文件中的 disable=yes 改为: disable=no

(8) 打开 “服务”管理器: 确保以下服务已打开:

ana apmd atd aut can cron cup echo gpm iptables isdn keytable kudzu
netfs network nfs nfslock ntalk pcm portmap random raw rexec rhn rlogin
rsh rsync sendmail sgi spa sshd syslog talk tel time time-upd xinetd
ypbind yppasswdd ypserv

(9) 正确安装 MPICH

① MPICH 是一个开放源码的软件, 所以可以从网上免费获取它的源代码。用户可以直接从 MPICH 的主页下载最新的软件包 mpich.tar.gz, 然后将它置于 /home/Cluster 目录下

② 使用如下命令解压缩源代码:

[root@tyut0 cluster] # tar -xzf mpich.tar.gz

解压缩后会生成一个名字为 mpich-1.2.4 的目录

③ 进入该目录，并执行位于该目录下 *configure* 脚本，为下一步编译源代码进行准备。

该配置脚本可以接受很多的参数 (*Options*)，通过运行如下命令：

```
[root@tyut0 mpich-1.2.4] # ./configure --help
```

可以获取更详细的参数信息。这里只列举几个最常用的参数：

--prefix: 指定 *mpich* 的安装目录。

--with-device: 指明所使用的通信系统类型。一般情况下我们使用 *ch_p4*，它表示通常的 *TCP/IP* 通信系统。

--with-arch: 指明所使用的操作系统的类型。

运行如下命令完成前期配置：

```
[root@tyut0 mpich-1.2.4] # ./configure --prefix=/Cluster/MPICH \
```

```
--with-device=ch_p4 \
```

```
--with-arch=LINUX
```

④ 最后，再运行如下命令完成 *MPICH* 的编译和安装：

```
[root@tyut0 mpich-1.2.4] # make
```

```
[root@tyut0 mpich-1.2.4] # make install
```

至此就完成了机群系统中主节点 *MPICH* 的安装

(10) 安装好 *MPICH* 后，到安装好 *MPI* 目录下的 *Share/machine.linux* 填入：

```
tyut0
```

```
tyut1
```

```
tyut2
```

```
tyut3
```

(11) 在文件 */etc/passwd* 中最后一行加入： +

(12) 在“新建终端”内，敲入“*exportfs -a*”

(13) 重启服务器

子节点机配置：(在所有的子节点机上进行如下同样的操作)

(1) 建立一个共享目录 */home/cluster*

(2) 在 */etc/hosts* 目录下填入：

219.226.86.105 tyut0

219.226.86.106 tyut1

219.226.86.107 tyut2

219.226.86.108 tyut3

(3) 在/etc/hosts.equiv 文件内填入: (如 hosts.equiv 不存在, 需要自己建立)

tyut0

tyut1

tyut2

tyut3

(4) 在/etc/fstab 文件内加入:

tyut0: /home/cluster /home/cluster nfs defaults 1 1

(5) 在/etc/xinetd.d 目录下 找到 rexec rlogin rsh 这三个文件 依次打开, 将三个文件中的 disable=yes 改为: disable=no

(6) 打开“新建终端”, 敲入 setup 命令, 然后选择 “Authentication Configuration”, 选择 NIS, 填入 NIS 域名: PCCLUSTER; 在服务器名一栏上敲入主节点 IP 地址, 即: 219.226.86.105

(7) 打开“服务”管理器: 查看以下服务, 确保已打开:

ana apm atd aut can cro cups echo gpm ipt isdn keytable kud netfs
network nfs nfslock ntalk pcm portmap random raw rex rhn rlo rsh
sendmail sgi_fam spa ssh sys tel time time-udp xinetd ypbind yppasswdd
ypserv

(8) 在文件/etc/passwd 中最后一行加入: +

(9) 在“新建终端”, 敲入 “mount -a”, 然后检查/home/cluster 文件内是否已挂接上主节点上的/home/cluster 内容, 如果看到, 则挂接成功。

(10) 重启节点机.

MPICH 的运行:

① MPICH 中最常用的两个命令就是 mpicc 和 mpirun。mpicc 是一个 MPI 编译器, 它负责将源程序编译为可执行文件, 它最常用的参数是 -o 用来指明输出文件。

```
[root@tyut0] # cd /Cluster/MPICH/examples
```

```
[root@tyut0] # ../bin/mpicc cpi.c -o cpinew
```

② *mpirun* 则是用来执行一个编译好的 *MPI* 程序。下面是它最常用的一些参数：

-np <np>: 用来指明所要生成的进程数。

-machinefile <machinefile name>: 缺省时使用的 *machines* 文件是前面介绍过的位于 *share* 目录下的 *machines.LINUX*；但通过这个参数可以指定一个临时的 *machines* 文件，从而使用不同的进程指派方式。

至此，整个机群系统搭建完成了，下面我简单的介绍一下并行编译环境的配置过程，这是进行并行编程的必要工作。

为了编程人员能够方便地进行编辑工作，我们可以在 *Windows* 系统下进行编辑工作。因为 *C* 或 *C++* 能够在 *Windows* 和 *Linux* 系统下都可以进行运行，所以我们选用 *C* 和 *C++* 语言进行编写工作。我们选用的编辑工具是 *Visual Studio 6.0*。具体方法如下：

① 修改运行库 左半部 *Settings For* 选择 “*Win32 Debug*”；右半部选择 *C/C++* 选项，*Category* 的选项切换到 “*Code Generation*”；*Use run-time library* 切换成 “*Debug Multithreaded*”。这时，在 “*Project Options*” 中应该有 “*MTd*”，并在正确的位置上。

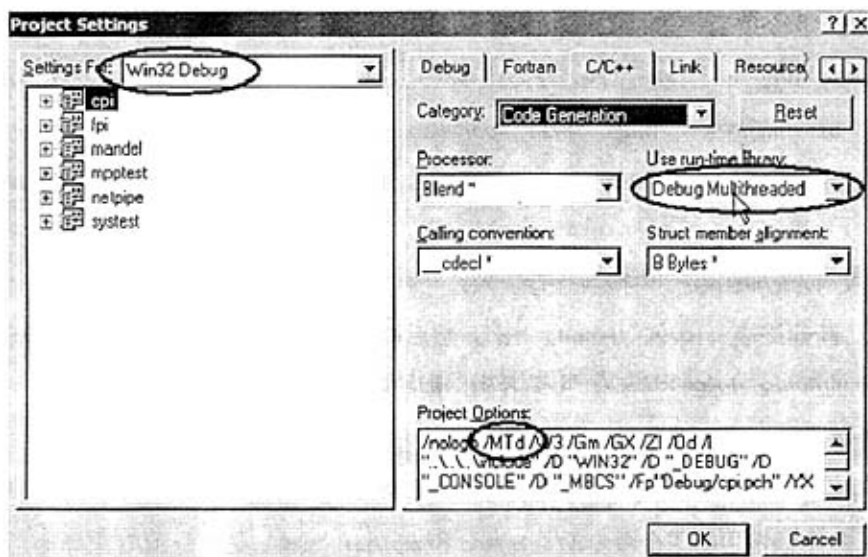


图 5-3 VC++ 修改运行库

Fig 5-3 VC++ Modify Library

② 置为多线程运行 再在左半部分的 *Settings For* 中选择“Win32 Release”; *Use run-time library* 选择“Multithreaded”; 确定“*Project Options*”中有 MT, 并在正确位置。

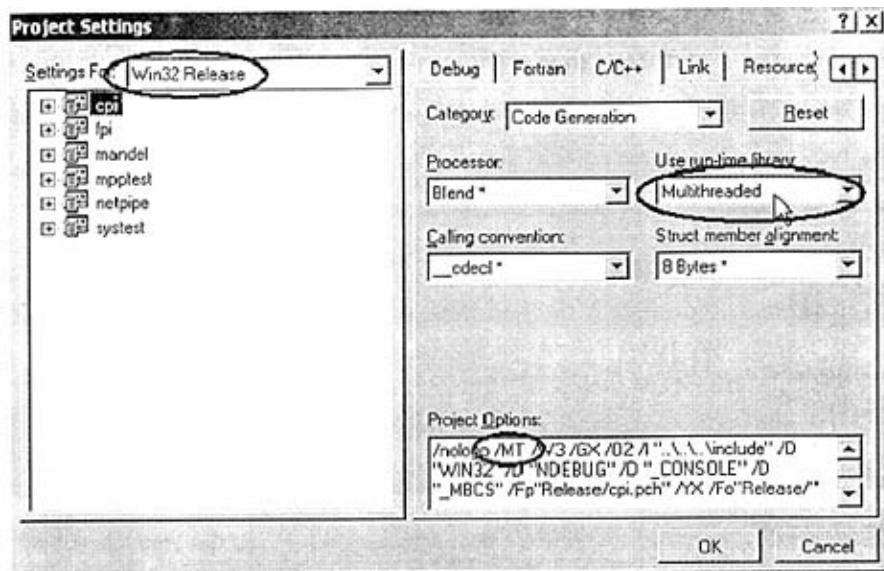


图 5-4 VC++置为多线程运行

Fig.5-4 VC++ Modify Multithreading

③ 添加包的路径 接着把左半部分的 *Settings For* 切换到 “*All Configurations*”，右半部的 *Category* 选择 “*Preprocessor*”。如下图 5-5 把 *Additional include directories* 的路径改成安装 *MPICH* 的 *include* 的路径。通常是 `\\...\\MPICH\\SDK\\include`

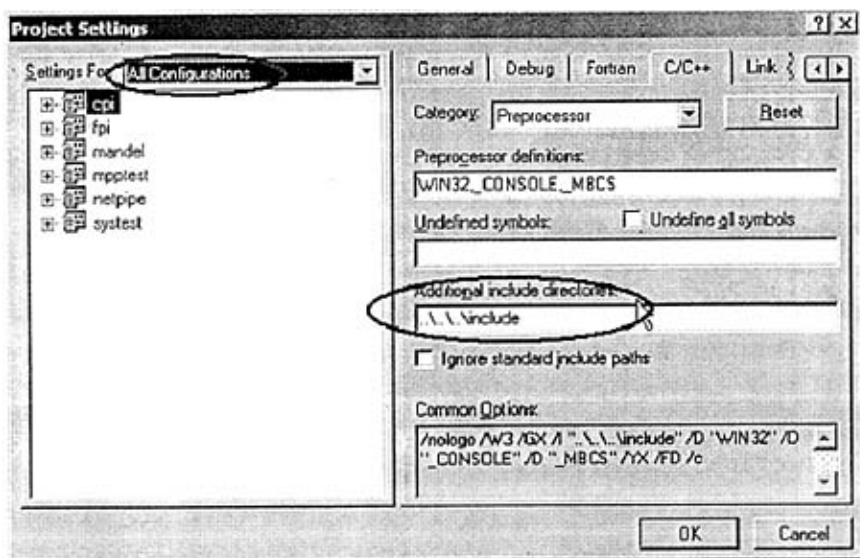


图 5-5 VC++添加包

Fig.5-5 VC++ Append Package

④ 添加附加库的路径 现在把左半部分 *Settings For* 切换到 “*All Configurations*”, 右半部分部分的页面切换到 *Link* 选项; 在右边部分的 *Category* 中选择 “*Input*”, 设置 *Additional library path* 的路径, 改成安装 *MPICH* 的 *Lib* 的路径。一般为 `\\.\MPICH\SDK\lib`。如图所示:

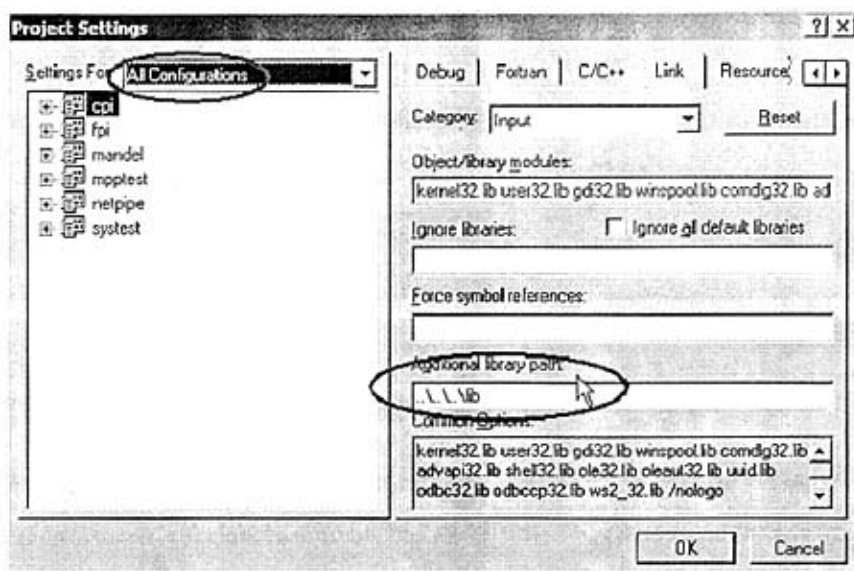


图 5-6 VC++添加附加库

Fig.5-6 VC++ Append Add-ons Library

⑤ 添加编译时用到的模块或库 现在把左部分的 *Settings For* 切换到 “*Win32 Debug*”, 右半部分在 *Link* 选项下, *Category* 中选择 “*General*”, 然后在对象和库模块 *Object/library modules* 中加上 “*ws2_32.lib*” 和 “*mpichd.lib*”。

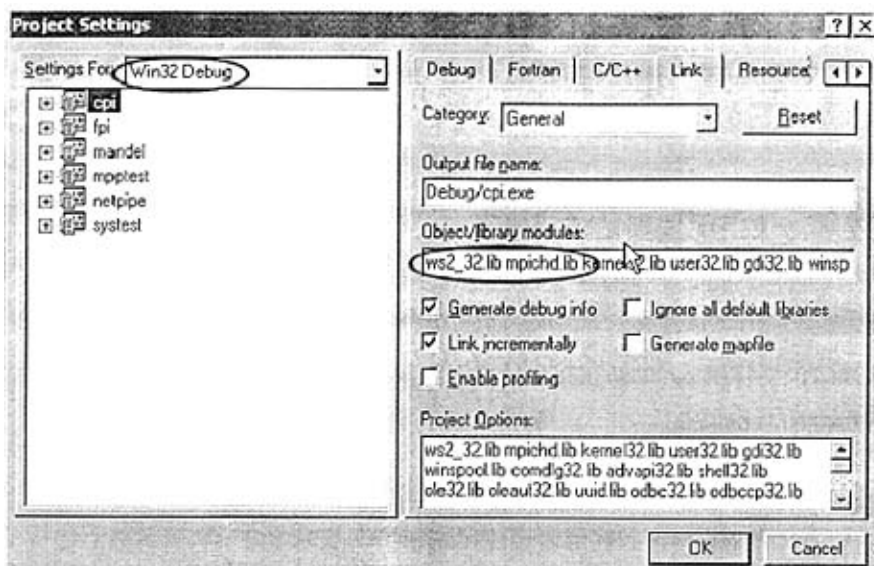


图 5-7 VC++添加模块

Fig.5-7 VC++ Append Module

⑥添加编译时用到的模块或库 把左部分的 *Settings For* 切换到“Win32 Release”，右半部分在 *Link* 选项下，*Category* 中选择“General”，然后在对象和库模块 *Object/library modules* 中加上“ws2_32.lib”和“mpich.lib”。如下图所示：

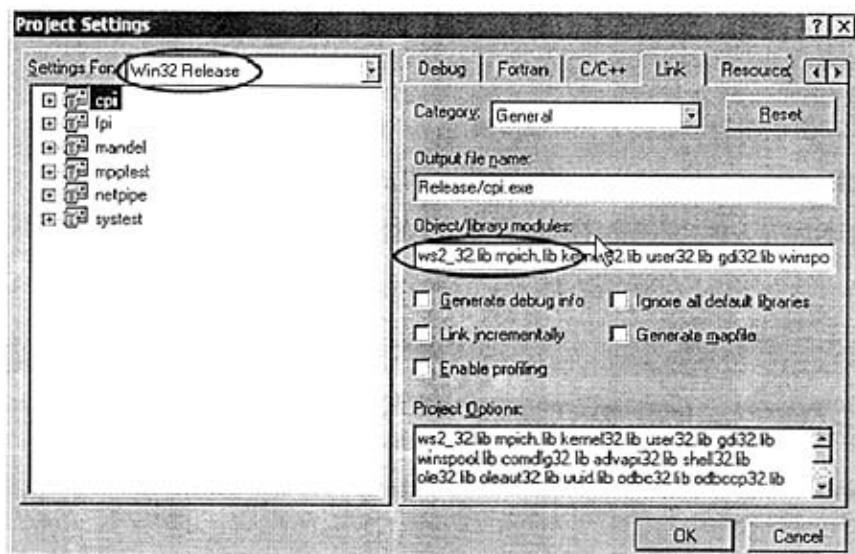


图 5-8 VC++添加模块

Fig.5-8 VC++ Append Module

至此，我们就可以进行并行编辑工作了

第六章 串匹配概率与实验结果分析

6.1 一般的连续 r 位匹配规则的匹配概率

连续 r 位匹配规则(r -contiguous bits rule)定义为^[36]: 对于任意两个长度为 L 的字符串 a 和 b , 如果两个字符串 a 和 b 在相应位置上至少连续 r 位相同, 则这两个字符串是连续 r 位匹配的, $Match_{contg}(a, b)$ 为匹配函数, 并有:

$$Match_{contg}(a, b) = \begin{cases} 1 & \text{字符串 } a \text{ 与 } b \text{ 在连续 } r \text{ 位匹配规则下匹配成功} \\ 0 & \text{否则} \end{cases} \quad (6.1)$$

例如: 串 $a = \boxed{001100}1010001101$
 串 $b = \boxed{001100}0101110010$
 串 $c = \boxed{0111}001011111001$
 串 $d = \boxed{0111}110100000110$

本例中, 设二进制字符串长度 $L=16$, 反应阈值 $r=6$ 。在连续 r 位匹配规则下, 字符串 a 和 b 匹配, $Match_{contg}(a, b)=1$; 而字符串 c 和 d 不匹配, $Match_{contg}(c, d)=0$ 。

两个随机的长度为 L 的二进制字符串 a 和 b , 在连续 r 位匹配规则下的匹配概率计算公式为^[15]:

$$P(Match_{contg}(a, b)=1) = P_{contg} \approx 2^{-r} \left(\frac{L-r}{2} + 1 \right) \quad (6.2)$$

6.2 改进的连续 r 位匹配规则的匹配概率

首先,我们先对一般的连续 r 位匹配规则下的匹配概率公式分析一下,当二进制字符串长 $L=49$, 阈值 $r=4$ 时,有:

$$P_{contg} \approx 2^{-r} \left(\frac{L-r}{2} + 1 \right) = 2^{-4} \left(\frac{49-4}{2} + 1 \right) = \frac{1}{16} * \frac{47}{2} = 1.46875 \quad (6.3)$$

显然不符合实际情况。也就是说,当 $L \geq \frac{r+L}{2} + r - 2$ 时,公式(6.2)不能正确地描述在连续 r 位匹配规则下的匹配概率。

准确地概率描述是匹配问题的重要数学依据,下面我们讨论这个问题。

假设已有字符串和欲匹配字符串都是长度为 L 的二进制字符串,匹配阈值为 r ,要对 $U \in \{0, 1\}^L$ 空间的所有 2^L 字符串进行检测,对于每个检测字符串要进行 $L-r+1$ 次比较,才可以完成串的匹配检测。设 P_i 是第 i 次比较时两个字符串不重复匹配的概率,则在连续 r 位匹配规则下的匹配概率计算公式为:

$$P(\text{Match}_{contg}(a, b) = 1) = P_{contg} \quad (6.4)$$

$$P_{contg} = \begin{cases} 0 & L < r \\ 1 & r = 0 \\ \sum_{i=r}^L P_i & L \geq r \neq 0 \end{cases} \quad (6.5)$$

以上公式中的参数 L, r 的意义与公式(6.2)相同。对于二进制字符串公式(6.5)中的 P_i 可以写成:

$$P_i = \frac{1}{2^i} A_y \quad j = r, r+1, \dots, L \quad (6.6)$$

$$\text{这里 } A_y = \begin{bmatrix} a_{r,r} & 0 & 0 & \dots & 0 \\ a_{r+1,r} & a_{r+1,r+1} & 0 & \dots & 0 \\ a_{r+2,r} & a_{r+2,r+1} & a_{r+2,r+2} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ a_{L,r} & a_{L,r+1} & \dots & \dots & 0 \end{bmatrix} \quad (6.7)$$

$$\text{其中: } a_{r,r}, a_{r+1,r}, a_{r+2,r}, \dots, a_{L,r} = 2^0, 2^1, 2^2, \dots, 2^{L-r} \quad (6.8)$$

其它非 0 项为重复串系数。

改进后的连续 r 位匹配规则下的匹配概率公式为:

$$P_{\text{contig}} = \begin{cases} 0 & 1 < r \\ 1 & r = 0 \\ \sum_{i=r}^L \sum_{j=r}^L \frac{1}{2^i} A_{ij} & 1 \geq r \neq 0 \end{cases} \quad (6.9)$$

下表是利用公式(6.9)计算出来的概率表:

$L \backslash r$	0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	0.5000	0	0	0	0	0	0	0	0	0	0	0
2	1	0.7500	0.2500	0	0	0	0	0	0	0	0	0	0
3	1	0.8750	0.3750	0.1250	0	0	0	0	0	0	0	0	0
4	1	0.9375	0.5000	0.1875	0.0625	0	0	0	0	0	0	0	0
5	1	0.9688	0.5938	0.2500	0.0938	0.0313	0	0	0	0	0	0	0
6	1	0.9844	0.6719	0.3125	0.1250	0.0469	0.0156	0	0	0	0	0	0
7	1	0.9922	0.7344	0.3672	0.1563	0.0625	0.0234	0.0078	0	0	0	0	0
8	1	0.9961	0.7852	0.4180	0.1875	0.0781	0.0313	0.0117	0.0039	0	0	0	0
9	1	0.9980	0.8262	0.4648	0.2168	0.0938	0.0391	0.0156	0.0059	0.0020	0	0	0
10	1	0.9990	0.8594	0.5078	0.2451	0.1094	0.0469	0.0195	0.0078	0.0029	0.0010	0	0
11	1	0.9995	0.8862	0.5474	0.2725	0.1245	0.0547	0.0234	0.0098	0.0039	0.0015	0.0005	0
12	1	0.9998	0.9080	0.5837	0.2988	0.1394	0.0625	0.0273	0.0117	0.0049	0.0020	0.0007	0.0002

Table 6-1 Probability Table

根据公式 (6.2) 以及表 6-1, 我们可以画出如下的对比曲线图, 从中可以看出一般的连续 r 位匹配概率公式 (6.2) 与改进后的概率公式 (6.9) 的概率曲线。改进后的公式更加符合实际情况。

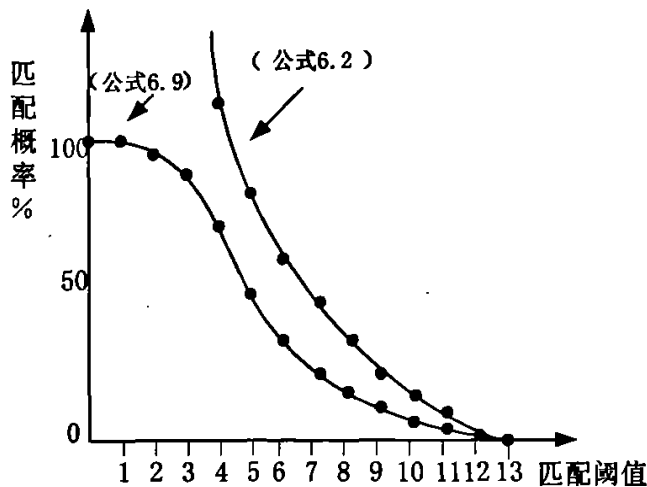


图 6-1 公式 (6.2) 与公式 (6.9) 比较
Fig.6-1 Formula 6.2 Compare with Formula 6.9

我们分析了连续 r 位匹配规则的数学基础和算法的实现, 在实际情况中, 每次匹配的过程都要多次调用匹配算法, 所以我们更加考虑的是其执行的快慢。根据我们以往的编程经验来看, 当随着问题的规模不断扩大, 数据量成指数上升时, 即使再好的匹配规则和算法都可能被海量的数据所吞噬, 发挥不出其优势, 所以我们还是要通过实际的实验结果来说明问题。

6.3 实验结果分析

在上文我们已经给出了具体的实验算法, 以及实验的平台搭建, 下面我们将对实验结果进行演示与分析。

我们在已有的并行实验室条件下实现了连续 r 位串匹配并行算法。我们分别测试了文本串长度为: 100k、500k、1M、2M、3M、4M、5M、6M、7M。我们选取模式串长度为固定值 5, 匹配阈值 r 取值为: 1、2、3、4、5。节点机个数为: 1, 2, 3, 4。其中文本串和模式串都是根据指定的长度随机生成的字符串。

6.3.1 运行结果

该实验是在 *Linux* 环境下进行的，实验步骤如下：

- ① 在 */home* 目录下建立一个共享目录 *cluster*。
- ② 将软件包 *mpich.tar.gz*，置于 */home/cluster* 目录下。
- ③ 然后在终端命令行中敲入 “*cd /home/cluster*” 命令，进入 */home/cluster* 目录下。
- ④ 然后使用如下命令解压缩源代码：*[root@ tyut0 cluster]# tar -zxvf mpich.tar.gz*
- ⑤ 解压缩后在 */home/cluster* 目录下会生成一个名字为 *mpich-1.2.4* 的目录。
- ⑥ 然后敲入 “*cd mpich-1.2.4*” 命令进入这个目录下。
- ⑦ 使用如下命令完成前期配置、安装工作：

```
[root@ tyut0 mpich-1.2.4]# ./configure --prefix=/Cluster/MPICH
```

```
[root@ tyut0 mpich-1.2.4]# make
```

```
[root@ tyut0 mpich-1.2.4]# make install
```

- ⑧ 将实验的 *C* 语言源代码拷入 *MPICH* 安装目录下面的 *BIN* 文件夹下面。
- ⑨ 敲入 “*cd MPICH/bin*” 命令进入 *bin* 这个目录下
- ⑩ 然后编译、运行、显示实验结果。

下面是对实验结果的演示：

```

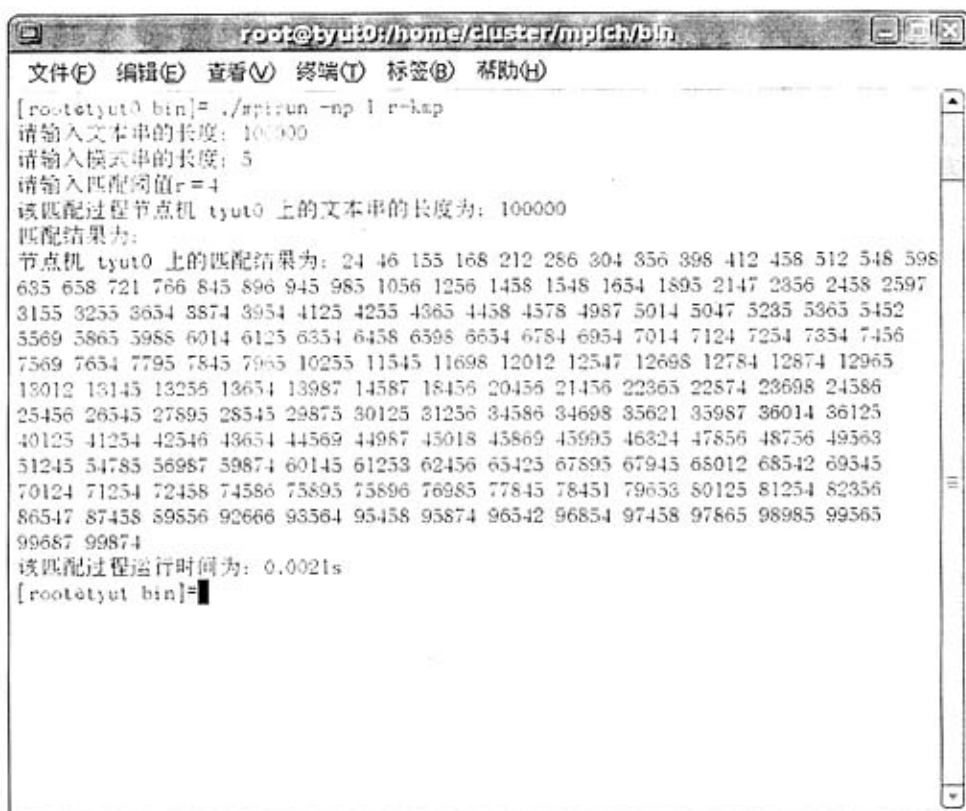
root@tyut0:/home/cluster/mpich/bin
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
[root@tyut0 bin]# ./mpirun -np 1 r-kmp
请输入文本串的长度: 100000
请输入模式串的长度: 5
请输入匹配阈值r=5
该匹配过程节点机 tyut0 上的文本串的长度为: 100000
匹配结果为:
节点机 tyut0 上的匹配结果为: 36 125 246 396 486 598 635 746 896 985 1056 1548 1896
2598 3236 4586 5698 5866 6985 7895 9568 10223 11254 12568 13569 15689 21549 25689
26987 27895 29875 30265 31565 32565 34512 35689 36875 39875 41256 42568 45698 48695
51235 55632 56898 61235 64523 65423 69875 71235 75698 78954 81236 85698 86512 88569
90132 92365 94565 96547 97854 98654
该匹配过程运行时间为: 0.0025s
[root@tyut bin]#

```

图 6-2 实验结果截图

Fig 6-2 Experiment Result Figure

图 6-2 说明：敲入可执行命令“`/mpirun -np 1 r-kmp`”，说明本次运算分成 1 个线程来进行运算，一个处理器参与运算。该实验文本串的长度为 100k，模式串长度为 5，匹配阈值为：5，节点机个数为 1 个，实验结果为：36、125 ...，即在节点机 *tyut0* 上的文本串符合匹配要求的字符串的位置在第 36 位、第 125 位 ...（注：第 1 个字母是第 0 位），程序运行时间为：0.0025 秒。



```

root@tyut0:/home/cluster/mpi/ch/bin
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
[root@tyut0 bin]# ./mpirun -np 1 r-kap
请输入文本串的长度: 100000
请输入模式串的长度: 5
请输入匹配阈值r=4
该匹配过程节点机 tyut0 上的文本串的长度为: 100000
匹配结果为:
节点机 tyut0 上的匹配结果为: 24 46 153 168 212 286 304 356 398 412 458 512 548 598
635 658 721 766 845 896 945 985 1056 1256 1458 1548 1654 1895 2147 2356 2458 2597
3155 3255 3654 3874 3954 4125 4255 4365 4458 4578 4987 5014 5047 5235 5365 5452
5569 5865 5988 6014 6125 6354 6458 6598 6654 6784 6954 7014 7124 7254 7354 7456
7569 7654 7795 7845 7965 10255 11545 11698 12012 12547 12698 12784 12874 12965
13012 13145 13256 13654 13987 14587 18456 20456 21456 22365 22874 23698 24386
25456 26545 27895 28545 28875 30125 31256 34586 34698 35621 35987 36014 36125
40125 41254 42546 43654 44569 44987 45018 45869 45995 46324 47856 48756 49563
51245 54785 56987 59874 60145 61253 62456 63425 67895 67945 68012 68542 69345
70124 71254 72458 74586 75895 75896 76985 77845 78451 79653 80125 81254 82356
86547 87458 89656 92666 93564 95458 95874 96542 96854 97458 97865 98985 99565
99687 99874
该匹配过程运行时间为: 0.0021s
[root@tyut0 bin]#

```

图 6-3 实验结果截图

Fig.6-3 Experiment Result Figure

图 6-3 说明: 该实验文本串的长度为 100k, 模式串长度为 5, 匹配阈值为: 4, 参与运算的处理器个数为 1 个, 实验结果为: 24、46 ..., 即在节点机 tyut0 上的文本串符合匹配要求的字符串的位置在第 24 位、第 46 位, ...。程序运行时间为: 0.0021 秒。本图与上图对比可以看出, 当匹配阈值 r 减少时, 匹配结果会有相应的增加。

```

root@tyut0:/home/cluster/mpich/bin
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
[root@tyut0 bin]# ./mpirun -np 4 r-kmp
请输入文本串的长度: 100000
请输入模式串的长度: 5
请输入匹配阈值r=5
该匹配过程节点机 tyut0 上的文本串的长度为: 25000
该匹配过程节点机 tyut1 上的文本串的长度为: 25000
该匹配过程节点机 tyut2 上的文本串的长度为: 25000
该匹配过程节点机 tyut3 上的文本串的长度为: 25000
匹配结果为:
节点机 tyut0 上的匹配结果为: 25 65 123 256 365 458 578 698 714 745 789 879 987
1025 1245 1298 1368 1568 1895 2014 2365 2456 2789 2987 3012 3589 3987 4789 5784
6259 7845 8956 9837 10235 11256 12569 14568 15478 16985 17845 19875 20145 24756
节点机 tyut1 上的匹配结果为: 14 98 187 278 369 487 589 698 784 956 1054 1478
1698 2368 3658 4875 5874 6987 7845 8542 9652 11478 12478 13659 14785 15896 16897
17854 18546 19568 21589 24156
节点机 tyut2 上的匹配结果为: 56 145 698 874 956 1256 2458 5874 6985 12546 18956
19857 21356 24569
节点机 tyut3 上的匹配结果为: 125 475 687 837 956 1254 1785 1857 1987 2145 2365
2458 2568 2687 2748 2854 2984 3125 3254 3354 3478 3568 3687 3784 3954 4321 4712
5478 6854 7845 9568 10455 10784 11458 14578 15784 18545 23456 24157
该匹配过程运行时间为: 0.0049s
[root@tyut0 bin]#

```

图 6-4 实验结果截图

Fig.6-4 Experiment Result Figure

图 6-4 说明: 该实验文本串的长度为 100k, 模式串长度为 5, 匹配阈值为: 5, 参与运算的处理器个数为 4 个, 实验结果为: 在节点机 *tyut0* 上的文本串符合匹配要求的字符串的位置在第 25 位、第 65 位, ...; 在节点机 *tyut1* 上的文本串符合匹配要求的字符串的位置在第 14 位、第 98 位, ...; 在节点机 *tyut2* 上的文本串符合匹配要求的字符串的位置在第 56 位、第 145 位, ...; 在节点机 *tyut3* 上的文本串符合匹配要求的字符串的位置在第 125 位、第 475 位, ...。程序运行时间为: 0.0049 秒。

```

root@tyut0:/home/cluster/mpich/bin
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
[root@tyut0 bin]# ./spirun -np 4 r-kep
请输入文本串的长度: 100000
请输入模式串的长度: 5
请输入匹配阈值r=4
该匹配过程节点机 tyut0 上的文本串的长度为: 25000
该匹配过程节点机 tyut1 上的文本串的长度为: 25000
该匹配过程节点机 tyut2 上的文本串的长度为: 25000
该匹配过程节点机 tyut3 上的文本串的长度为: 25000
匹配结果为:
节点机 tyut0 上的匹配结果为: 29 95 145 230 345 489 547 658 740 789 879 956 994
1104 1214 1345 1468 1587 1847 2214 2347 2498 2689 2974 3312 3554 3677 4189 5647
5981 6595 7845 8956 9857 10235 11256 12589 14568 15478 16985 17845 19875 24756
节点机 tyut1 上的匹配结果为: 74 99 147 247 278 325 389 469 477 567 687 774 945
966 1123 1465 1494 1678 1691 2255 2488 3144 3976 4813 4962 5558 6558 6869 7956
8624 9778 11478 12147 13012 14147 15987 16254 17684 18658 19987 21354 24478
节点机 tyut2 上的匹配结果为: 86 175 236 541 624 844 962 1157 2348 3186 4235 5654
6525 7615 8546 9465 10584 11548 12565 13265 14487 15548 16254 17698 18254 19235
22546 23469 24516
节点机 tyut3 上的匹配结果为: 245 375 987 1057 1256 1854 1586 1658 1758 1985 1997
2245 2389 2412 2541 2602 2732 2847 2936 3103 3289 3358 3460 3564 3623 3714 3864
4536 4602 5376 6124 6604 7125 7655 8562 9368 11465 11982 12498 13478 14584 15553
19856 21456 23654 24158
该匹配过程运行时间为: 0.0055s
[root@tyut0 bin]#
    
```

图 6-5 实验结果截图

Fig.6-5 Experiment Result Figure

图 6-5 说明: 该实验文本串的长度为 100k, 模式串长度为 5, 匹配阈值为: 4, 参与运算的节点机个数为 4 个, 实验结果为: 在节点机 tyut0 上的文本串符合匹配要求的字符串的位置在第 29 位、第 95 位, ...; 在节点机 tyut1 上的文本串符合匹配要求的字符串的位置在第 74 位、第 99 位, ...; 在节点机 tyut2 上的文本串符合匹配要求的字符串的位置在第 86 位、第 175 位, ...; 在节点机 tyut3 上的文本串符合匹配要求的字符串的位置在第 245 位、第 575 位, ...。程序运行时间为: 0.0055 秒。

总结一下上面的实验结果, 当文本串长度为 100k 时, 模式串为 5 时, 改变匹配阈值 r , 改变参与运算的节点机个数, 最终得到的答案是不一样的, 所花费的时间也是不一样的。关于这些数据的对比与分析, 在下文有详细的描述, 在这里我们只是演示一下运行结果。

对于文本串长度、匹配阈值为其它值时, 运行结果的形式与上面的类似, 下表列出

了各种情况时所用的时间：（单位：秒）

模式 串长度 为 5	文本串 长度 匹配 阈值	100k	500k	1M	2M	3M	4M	5M	6M	7M
1 个 节 点	1	0.0011	0.0031	0.1045	0.2353	0.3987	0.5152	0.6886	0.7882	0.8914
	2	0.0019	0.0045	0.1125	0.2467	0.4357	0.5869	0.6991	0.7981	0.9045
	3	0.0012	0.0035	0.1158	0.2457	0.4895	0.5997	0.7882	0.8765	0.9254
	4	0.0021	0.0043	0.1186	0.2564	0.4125	0.6214	0.7958	0.8964	0.9578
	5	0.0025	0.0098	0.1198	0.2987	0.4586	0.6199	0.7954	0.9399	0.9659
2 个 节 点	1	0.0015	0.0027	0.1382	0.2456	0.3761	0.5196	0.6556	0.7662	0.8870
	2	0.0018	0.0028	0.1425	0.2413	0.3987	0.5001	0.6442	0.7012	0.8542
	3	0.0029	0.0045	0.1478	0.2985	0.4156	0.5123	0.6125	0.7125	0.8866
	4	0.0025	0.0039	0.1502	0.2997	0.4012	0.5248	0.6033	0.7158	0.9154
	5	0.0045	0.0045	0.1687	0.2566	0.4256	0.5124	0.6899	0.8016	0.9450
3 个 节 点	1	0.0029	0.0045	0.1885	0.2894	0.3705	0.5086	0.6882	0.7551	0.8709
	2	0.0045	0.0066	0.1968	0.2687	0.3875	0.5014	0.6756	0.7458	0.8354
	3	0.0039	0.0055	0.1914	0.2874	0.3458	0.5012	0.6788	0.7235	0.8311
	4	0.0041	0.0044	0.1785	0.2645	0.3998	0.5124	0.6425	0.7211	0.9025
	5	0.0048	0.0066	0.158	0.225	0.3256	0.5001	0.6254	0.7012	0.9095
4 个 节 点	1	0.0056	0.0089	0.1990	0.3090	0.3800	0.4880	0.6550	0.7660	0.8320
	2	0.0068	0.0093	0.2014	0.3124	0.3844	0.4758	0.6254	0.7412	0.8147
	3	0.0049	0.0087	0.1989	0.3145	0.3745	0.4795	0.6874	0.7215	0.8014
	4	0.0055	0.0047	0.2145	0.3044	0.3624	0.3555	0.6758	0.7158	0.8958
	5	0.0049	0.0051	0.2005	0.3014	0.3258	0.3789	0.6347	0.7128	0.9020

表：6-2 运行时间表

Table 6-2: running time table

6.3.2 运算时间的分析

对于计算时间的分析我们选择了在匹配阈值都等于1、模式串的长度为5的条件下，1~4个节点机进行比较。如图6-6所示：

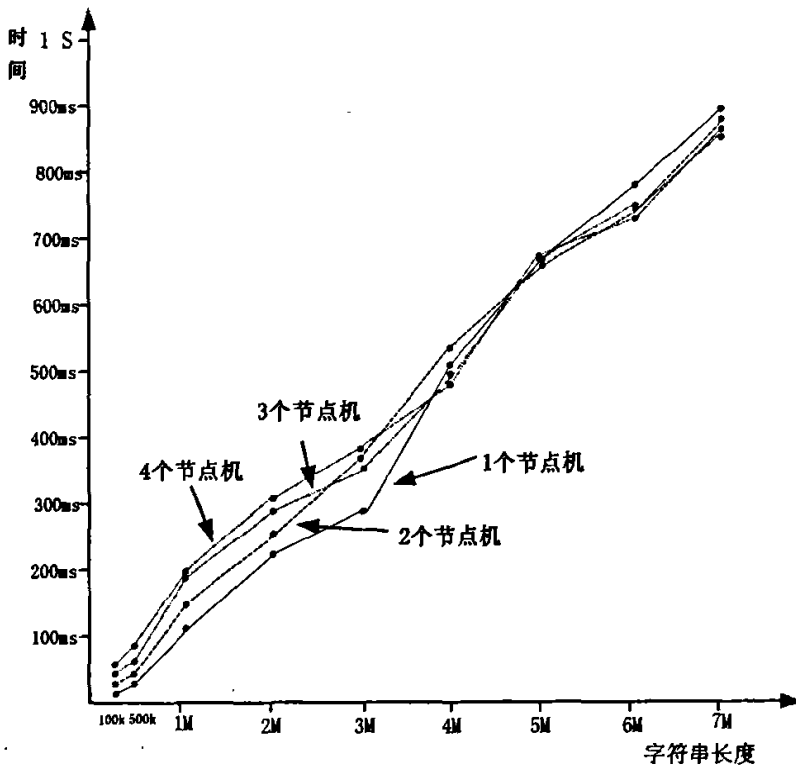


图 6-6 运算时间的分析

Fig 6-6 running time analyzing

从图6-6可以看出，当字符串长度比较短的时候，1个节点机完成匹配的时间要少于2个，3个和4个节点机所用的时间，随着字符串长度越来越大的时候，1个节点机所用的运算时间要比2个，3个或4个节点机所用的运算时间从图上的曲线变化趋势来看增加的速度要快。而且从图中可以看出，文本串的长度是影响运行时间的主要因素。

6.3.3 匹配阈值 r 的分析

下图 6-7 中我们采用了文本串长度为固定值 $7M$, 模式串的长度为 5 情况下进行的比较:

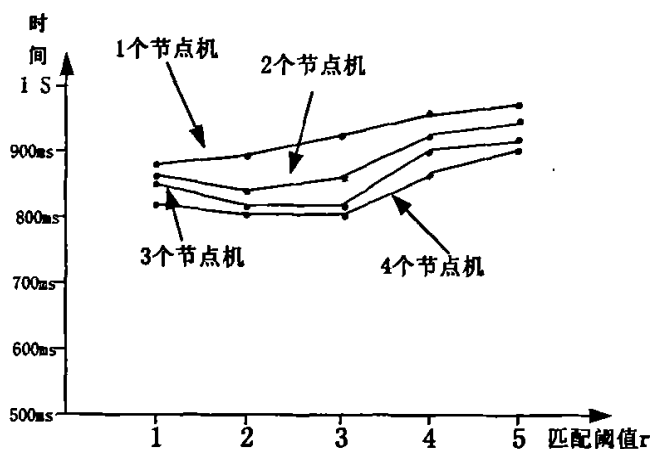


图: 6-7 匹配阈值 r 的分析

Fig 6-7: matching threshold r analyzing

从上图可以看出。当文本串长度为 $7M$ 时, 匹配阈值 r 不断增加时, 1~4 个节点机匹配运行的时间都是有所变化的, 但是 4 个节点机并行运行的时间要比其它情况时间要少一些。

下图 6-8 中我们采用了文本串长度为固定值 $100k$, 模式串的长度为 5 情况下的进行的比较

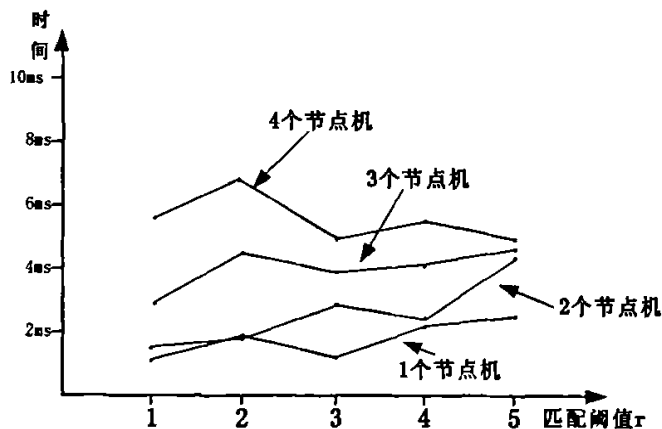


图: 6-8 匹配阈值 r 的分析

Fig 6-8: matching threshold r analyzing

从上图可以看出。当文本串长度为 $100k$ ，匹配阈值 r 不断增加时，1~4 个节点机匹配运行的时间都是有所变化的，但是 1 个节点机并行运行的时间要比其它情况时间要少一些。

从图 6-7 与图 6-8 对比来看，可以发现，当文本串长度比较短的时候，匹配阈值 r 的变化，对整体运算时间的影响比较小，但是，当文本串长度比较长的时候，匹配阈值 r 的变化，对整体运算时间的影响就比较大了。

综述以上的实验结果我们可以了解到，文本串规模的大小、匹配阈值的变化以及节点机的个数是影响运算时间的主要原因。当文本串规模比较小的时候，单个节点机进行匹配运算的时间要比多个节点机运算时间要少，但在本论文所用的并行机群环境条件下，当文本串长度大于 $6M$ 左右的长度时，多个节点机运行的时间就要比单个节点机运行的时间要少了，而且，此时单个节点机在字符串长度增加时，运算时间上的增加率要快与多个节点机运算时间上的增加率。而且，当匹配阈值变化时，匹配运算的时间也要有相应的变化，但是他们总体的运算变化趋势并没有改变。

6.3.4 可扩放性分析

Kumar 在文献[37]中提出了等效率函数(*iso-efficiency function*)，这种度量并行算法可扩放性的标准指出，对于给定的算法，如果计算量保持固定，则效率随着节点机的个数的增加而减小；为了维持效率在某一规定值的范围内变化，即等效率(*iso-efficiency*)，则在节点机个数增多的同时，计算量亦必须按比例增大才可以。

等效率函数定义为，维持效率不变时计算量随着节点机个数增长的模式。如果此匹配过程为线性的，则算法是线性可扩放的；如果匹配过程为亚线性的，则算法是可扩放的；如果匹配过程为指数的，则算法是不可扩放的^[38]。根据以上演示的试验结果，我们可以看出该匹配过程是非线性的，算法是可扩放的，但不是线性可扩放的。

第七章 总 结

串匹配是计算机研究领域的一个经典问题，是众多计算机搜索问题的关键技术之一。随着互联网的普及和发展，海量信息的处理和新的应用需求对串匹配技术提出了新的挑战。

本文主要对不完全串匹配技术进行了研究。在串匹配方面，本文分别从运算时间、以及影响运算时间的因素这两方面出发，提出了将并行计算引入串匹配求解当中，这样当问题规模不断扩大的情况下，运算时间和串匹配效果上都得到了比较好的改善。

对于不完全串匹配问题，作者引入了免疫系统中抗体与抗原的部分匹配原理。这么选择主要考虑到了两方面的原因：

- ① 抗体与抗原的匹配过程是一种典型的不完全匹配的例子，在人体免疫系统中，抗体的种数大概只有 10^6 个左右，而到现在人们已经知道的病毒性抗原的种数大概就有 10^{16} 个，在这种情况下，人类仍然能够利用强大的免疫系统抵御外来的绝大多数病毒性抗原，其原因就要归功于免疫系统中的抗体与抗原相结合的时候是按照不完全匹配的机制来完成的，即抗体与抗原的匹配是按照一对多的规则进行的。
- ② 免疫系统具有天然的并行性和健壮性。这一点也正是符合我们所考虑的当问题规模逐渐扩大时如何解决求解时间呈指数增加这一问题。免疫系统由分布在机体各个部分的细胞、组织和器官等组成。抗原也是分散在机体内部的，其工作特性就是同时性、并行性。其次免疫系统的分布式性尤其利于加强系统的健壮性，从而使得免疫系统不会因为局部组织损伤而使整个功能受到很大影响。

基于以上两个原因，我们最终选择了串匹配问题与免疫系统相结合的研究方向。虽然，免疫系统的分布性由于其潜在的效率和错误耐受而受到计算机科学家的欢迎；但是，除了一些特殊的环境，如计算机网络安全系统之外，这种性质很少在具体的串匹配算法中得到应用。本文作者正是注意到了这个情况，所以力争在这方面有所研究。通过算法思想的介绍，算法的具体描述，以及最终的结果分析，得到了比较满意的效果。但是本文还有一些未解决的问题。例如，从最后实验结果分析可以看出，当文本串的规模增大时，运算时间增加的也非常快，其原因是多方面的，我们如何解决这个问题，是值得进一步研究的。而且，对于本文的实验环境来说当文本串长度大于 $6M$ 左右的时候，多个

处理器逐渐显现出比单个处理器运算时间增加缓慢的现象,这个文本串长度的临界值是具体机群系统有关系的,其关系是什么,也是值得研究的。另外,本文是跨学科、跨领域的一次研究,虽然借鉴了免疫系统的许多特性,但是这些只是免疫系统中很少的一部分,如何能够通过对生物免疫系统的深入理解和研究来发挥其更大的优势,则需要与医学领域专家合作共同来完成。例如,本文所设计的算法以及最后的运行只是对一个问题处理,即使使用了并行计算,也只是对同一个问题来研究如何并行,然而在人类的生物免疫系统中,大多数情况是免疫系统要同一时间应对来自外界不同种的病毒性抗原的侵扰,对于不同的问题同时进行处理,而每个问题又可以并行来执行。这些问题都值得我们在今后的工作、学习中进一步研究。

参考文献

- [1]李涛。计算机免疫学。电子工业出版社, 2004 年 7 月
- [2]K. Fredriksson, G Navarro. *Average-Optimal Single and Multiple Approximate String Matching*. In *ACM Journal of Experimental Algorithmics (JEA)* 9(1.4): 1-47, 2004.
- [3]Tan Jian-long Liu Ping and Liu Yan-bing. *A partition-based efficient algorithm for large scale multiple-string matching*. *SPIRE 1005*, 2005.
- [4]贺龙涛, 方滨兴, 余翔湛。一种时间复杂度最优的精确串匹配算法。软件学报。16(5):676-683, 2005.
- [5]Marc Norton. *Optimizing Pattern Matching for Intrusion Detection*
(<http://www.idsresearch.org>, 2006)
- [6]B.Calder N. Tuck, T Sherwood and G Varghese. *Deterministic memory-efficient string matching algorithms for intrusion detection*. *IEEE INFOCOM*, 2004.
- [7]R. H. Katz F. Yu and T. V Lashman. *Gigabit rate packet pattern matching with tcam*. *Technical Report*, 2004.
- [8]L. Tan and T. Sherwood. *A high throughput string matching architecture for intrusion detection and prevention*. *Proc. 31nd Annual International Symposium on Computer Architecture (LISA)*, 2005.
- [9]莫宏伟。人工免疫系统原理与应用。哈尔滨工业大学出版社。2003. 1~390
- [10]龚非力。医学免疫学。北京: 科学出版社。2003.1. 1~27
- [11] Ge Hong, Mao Zong -Yuan. *Immune Algorithm, Proceedings of the 4th World Congress on Intelligent Control and Automation*. 2002.6. 1784~1787
- [12]于善谦, 王洪海, 朱乃硕等, 免疫学导论[M]。北京: 高等教育出版社, 1999.
- [13] Leandro Nunes de Castro, Fernando Jose Von Zuben. *Artificial Immune Systems: PartII -A Survey of Applications*[D]. 2000.1 28~59
- [14] Sun Xiao. *Bio Informatics* [EB /OL]. Southeast University,
(<http://www.lmbe.seu.edu.cn/chenyuan/xsun/bioinformatics/Web/CharpterThree>)
- [15]Percus, J. K., Percus, O. E., & Perelson, A. S. *Predicting the size of*

- the antibody-combining region from consideration of efficient self/unsself discrimination. In Proceedings of the National Academy of Science 90. 1993.(pp. 1691 — 1695).*
- [16]赵念强, 鞠时光. 入侵检测系统中模式匹配算法的研究[J]. 微计算机信息, 2005, 14: 22—24.
- [17]卢开澄. 算法与复杂性. 高等教育出版社, 1995: 1~324
- [18] S. Forrest. *Using Genetic Algorithms to Explore Pattern Recognition in the Immune System. Evolutionary Computation, 1993, (3): 191-211*
- [19]T. Lecroq C. Charcas. *Handbook of Exact String Matching Algorithms. Feb.2004.*
- [20] G.Navarm and M. RaflSnot. *Flexible Pattern Matching in Strings: Practical on-line search algorithms for texts and biological sequences. 2002.*
- [21] A. V. Aho, M. J. Corasick. *Efficient String Matching: An Aid to Bibliographic Search. Communications of ACM, 1975, 18(6): 333-340*
- [22] D. E. Knuth, J. H. Morris, V. R. Pratt. *Fast pattern matching in strings. SIAM Journal on Computing, 1977,6(2):323-350*
- [23]严蔚敏. 数据结构(C语言版)[M]. 北京: 清华大学出版社, 2001
- [24] Michael. W. *The Research and Development of Parallel Computation, Parallel Computing Theory and Practice, 2000, 129 -135*
- [25]David E.Culler 等编, 并行计算机体系结构, 机械工业出版社, 1999
- [26]M. J. Flynn. *Some Computer Organizations and Their Effectiveness. IEEE Trans. Computers, 21(9): 948 — 960,1992.*
- [27]都志辉, 高性能计算并行编程技术—MPI 并行程序设计, 北京: 清华大学出版社, 2001 年
- [28]陈国良, 吴俊敏 并行计算机体系结构, 北京: 高等教育出版社, 2002
- [29]莫宏伟. 人工免疫系统原理与应用. 哈尔滨: 哈尔滨工业大学出版社, 2002
- [30] Barry Wilkinson, Michael Allen, *Parallel Programming, China Machine Press,2005*
- [31]陈国良, 并行计算的设计与分析(修订本), 北京: 高等教育出版社, 2002.
- [32]A. H. Karp. *Programming for Parallelism. IEEE Computer, 20(5):43-57, 1987*
- [33]Rajkumar Buyya 编, 郑纬民译, 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

[34]黄凯, 徐志伟.可扩展并行计算机技术。机械工业出版社, 2000

[35]陆鑫达。并行和分布计算技术现状及发展策略。机械工业出版社, 2003

[36]*S A Hofmeyr. An Immunological Model of Distributed Detection and Its Application to Computer Security[D]. PhD Dissertation. Albuquerque, NM: The University of New Mexico, 1999*

[37] *Kumar V, Rao V N. Parallel depth-first search, Part two: analysis. International Journal of Parallel Programming, 1997,16 (6) : 501 ~ 519*

[38]漆安慎, 杜婵英。免疫的非线性模型[M]。上海: 上海科技教育出版社, 1998

致 谢

三年的研究生生活很快就要过去了,在三年的学习生活中我得到了许多老师和同学们的无私关怀和帮助,使我终身难忘。我衷心地感谢导师谢红薇,三年来她给予了我无微不至的关怀和孜孜不倦的教诲!谢老师严谨的治学态度、渊博的专业知识、为人宽厚的处世态度、一丝不苟的工作作风给我留下了极为深刻的印象,使我获得了一份宝贵思想财富。

课题研究过程中,许多老师和同学给了我很多的启示和帮助,从而使我能顺利进行并完成我的毕业设计。在此我十分感谢这些给过我帮助的老师 and 同学。感谢那些关心我的亲人和朋友,在我多年的求学路上他们给了我无微不至的关怀,他们的勉励和支持使我得以顺利地完成学业。

特别感谢我的家人默默付出的无价支持、鼓励和关怀!

攻读硕士学位期间发表的论文

谢红薇, 王志国, 余雪丽。基于免疫匹配规则的本体语义匹配。微计算机信息
2007 年第 7 期

作者: [王志国](#)
学位授予单位: [太原理工大学](#)

本文读者也读过(10条)

1. [席亮](#) [免疫入侵检测自体集优化和检测器生成算法研究](#)[学位论文]2009
2. [马明](#) [串匹配算法的并行实现策略](#)[学位论文]2010
3. [姜恩龙](#) [基于否定选择的检测器生成算法研究](#)[学位论文]2007
4. [张鑫](#) [快速模式串匹配技术的研究及一个邮件内容过滤系统的实现](#)[学位论文]2003
5. [武永超](#) [基于网络处理器的多模式串匹配算法研究](#)[学位论文]2008
6. [贾春朴](#) [基于并行处理的熟料强度预测的研究与实现](#)[学位论文]2007
7. [谢建全](#), [Xie Jianquan](#) [入侵检测中一种快速串匹配算法](#)[期刊论文]-[信息安全与通信保密](#)2006(10)
8. [刘伟](#), [诸昌铃](#) [一种新的硬判决并行干扰消除算法](#)[期刊论文]-[西南交通大学学报](#)2002, 37(3)
9. [邹大毕](#), [林东岱](#), [Zou Dabi](#), [Lin Dongdai](#) [抗边信道攻击的快速并行标量乘法](#)[期刊论文]-[计算机工程与应用](#)2006, 42(9)
10. [朱永利](#), [宋少群](#), [冯建衡](#), [Zhu Yongli](#), [Song Shaoqun](#), [Feng Jianheng](#) [互联电网节点阻抗阵实时修改与边界等值化简的并行计算方法](#)[期刊论文]-[电工技术学报](#)2007, 22(9)

本文链接: http://d.wanfangdata.com.cn/Thesis_Y1202424.aspx