中国科学院计算技术研究所

硕士学位论文

面向体系结构的串匹配算法优化研究

姓名：戴正华

申请学位级别：硕士

专业：计算机系统结构

指导教师：冯圣中

20060601

摘 要

串匹配算法广泛应用于生物信息学、信息检索等领域。随着基因数据和网络数据的爆炸式增长，对串匹配算法的性能也提出更高的要求。

本文从计算机体系结构的角度出发，立足于开发串匹配算法的数据并行性和提高算法的 cache 性能。本文的主要工作包括：

(1) 基于 SIMD 指令集的动态规划算法数据并行性开发与实现研究。

动态规划是模糊串匹配和序列联配中的基本算法。一类动态规划算法以 Smith-Waterman 算法为代表，计算复杂度为 $O(n^2)$；一类是以 Zuker 算法为代表，计算复杂性为 $O(n^3)$。本文分析了两类动态规划算法的数据依赖关系及其数据并行性。基于前驱计算思想，优化设计了 Smith-Waterman 算法，充分挖掘了算法的数据并行性，并基于 SIMD 指令集实现，数倍提高了该算法性能。通过设计辅助矩阵，有效组织数据，优化设计了 Zuker 算法，对于较大的数据规模，获得了接近理论值的加速比。

(2) 基于组合字符集的高性能 shift-or 算法研究

Shift-or 算法广泛应用于内容过滤领域。基本的 shift-or 算法以 char 类型的字符集合作为字符集，因而状态转换单位为 8bits。本文通过组合 NFA 的状态，建立 Σ_{short} 上的 NFA，状态转换单位提高为 short 型整数，显著降低了自动机状态转换次数，提高了计算效率。

(3) 面向位并行的低内存开销高性能关键词表达式匹配算法研究

关键词表达式匹配算法是一类重要的串匹配算法。本文面向位并行，利用 four-russian 和分治等优化方法设计了新的关键词表达式匹配算法，可提高计算效率、降低内存开销。

关键词：串匹配算法；SIMD；算法优化；位并行

ABSTRACT

Optimization of string matching algorithm based on computer architecture

Dai Zhenghua (Computer Architecture)

Directed By Feng Shengzhong

String matching algorithm is a fundamental problem in Bioinformatics and Information retrieval. Despite of a big amount of efforts spent by researchers on designing efficient string matching algorithm, improving the string matching efficiency remains of primary importance, which is due to the drastic growth of gene data and web content and consequently the more and more critical demand on the performance of string matching.

This thesis aims at addressing the problem of how to optimize string matching algorithm with respect to the characteristics of computer architecture, whose primary aim is to explore ILP and improve the performance of data accessing through cache. The main work is listed as below:

(1) Research on exploiting and implementing data parallelization of dynamic programming and based on SIMD instruction set.

Dynamic programming is the basis of approximate string matching and sequence alignment. Two most popular kinds of such algorithms are presented in this thesis: the one is represented by Smith-Waterman algorithm whose computing complexity is O(n2); the other is represented by Zuker algorithm whose computing complexity is O(n3). We analyze the data dependency and data Parallelism of these two kinds of dynamic programming. Then we exploit data parallelization of Smith-Waterman algorithm based on the thinking of Prefix Computation, and implement it with SIMD instruction set. With the help of the auxiliary array, we organize the way of accessing the data of the Zuker algorithm, which can gain the speedup ratio near the theoretical value when the input size is big.

(2) Research on high performance shift-or algorithm based on combining character set.

Shift-or algorithm is widely used in information retrieving and filtering. The basic shift-or algorithm treats the set of ASCII code as its character set, so it transfers its state when reading an 8-bit data. With respect to this point, we combine the states of NFA, set up the NFA on the basic element, which is read to transfer the state of NFA, and improve its size to that of short int. So the frequency of state transfer is reduced and the computing efficiency is enhanced.

(3) Research on high performance and low memory cost key-word expression matching algorithm from bit parallelization.

Key-word expression matching algorithm is a significant sort of string matching

algorithm. In ground of bit-parallel, we design this key-word expression matching algorithm with four-Russian technology which can reduce the cost of memory.

Keywords: sequence alignment algorithm, string matching algorithm, SIMD

图目录

表目录

声　　明

我声明本论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，本论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

作者签名：戴正华　　日期：2006年4月10日

论文版权使用授权书

本人授权中国科学院计算技术研究所可以保留并向国家有关部门或机构送交本论文的复印件和电子文档，允许本论文被查阅和借阅，可以将本论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编本论文。

（保密论文在解密后适用本授权书。）

作者签名：戴正华　导师签名：邱振岳 代签　日期：2006年4月10日

第一章 引言

作为一类基本算法，串匹配算法广泛应用在生物信息学、入侵检测、信息过滤、数据库、中文分词等众多领域。高性能串匹配算法研究具有重要的理论意义和应用价值。

1.1 背景

近年来，随着众多生物基因组测序分析项目的完成，基因和蛋白质数据爆炸式增加，基因组研究迅速发展，对数据的搜集、管理、处理、分析、释读能力的要求迅速提升。基因数据的总量每 14 个月翻一番（如图 1.1.1），而按 Moore 定律，计算性能每 18－24 个月增长一倍。因此，基因数据的增长，对于高性能算法设计也提出了挑战。生物信息学中的问题大部分可以转化成串的问题。

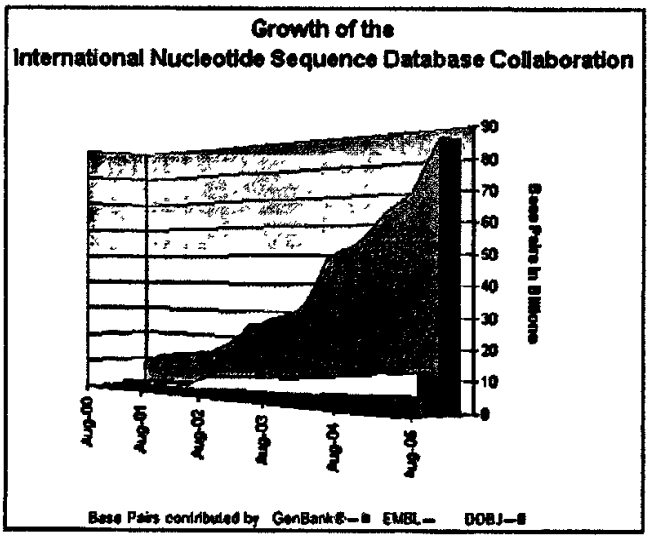


图 1.1.1 2000 年到 2005 年基因数据增长趋势图

同时在网络内容分析中，截止到 2005 年 6 月 30 日，我国国际出口带宽的总容量为 82617M，与半年前相比增加了 8188M，增长率为 11.0%，和上年同期相比增加 53.2%（如图 1.1.2 所示）【网络信息 2005】。

计算机硬件的性能的提升遵循摩尔定律，即每 18 个月翻一番，已经无法满足高速增长的网络数据和基因数据对计算机性能的需求。 这就对软件的性能提出了更高的需求。无论是生物信息还是网络信息的处理，核心都是串的匹配问题。

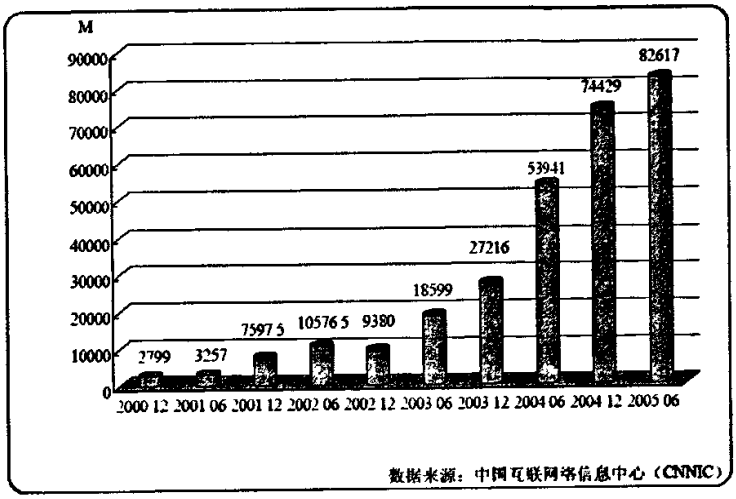


图 1.1.2 2000 年到 2005 年我国国际出口带宽增长趋势图

1.2 串匹配算法

1.2.1 串匹配算法基本概念

约定Σ为给定字符集。例如在生物信息学中，Σ＝{A, G, C, T}。

定义 1.1 给定字符串 s∈Σ*，如果 s=uvw，u,v,w∈Σ*，则称 v 为 s 的子串。

定义 1.2 给定字符串 s∈Σ*，如果 s=uv，u,v∈Σ*，则称 v 为 s 的后缀。

定义 1.3 给定字符串 s∈Σ*，如果 s=vu，u,v∈Σ*，则称 v 为 s 的前缀。

定义 2. 给定两条序列 $S= s_1...s_n$ 和 $T=t_1...t_m$。那么我们用|S|来表示 S 的长度，$S[i]$表示序列 S 的第 i 个字符。如果序列 S 和 T 相同，则必须满足：

（1）、 $|S|=|T|$；

（2）、 $S[i]=T[i]$，$(0<i\leqslant|S|)$；

定义 3. 串匹配算法是指：给定模式串 P∈Σ*和文本串 T∈Σ*，其中|P|=m，|T|=n，m,n>0 和 m<n，从 T 中找出所有子串 v 发生的位置使得 v=P。

定义 4. 任意给定两个串 X,Y∈Σ*,串 X 和 Y 之间的编辑距离 edit(X,Y)为三种编辑操作将串 X 转化成串 Y 所需的操作的次数：从 X（或 Y）中删除一个字符；向串 X（或 Y）插入一个字符；用另一个字符替换串 X（或 Y）中指定的某个字符。

定义 5. 所谓允许 k-差别的模糊串匹配问题是指，给定模式串 P∈Σ*,P 的长度|P|=m,和文本串 T∈Σ*，T 的长度|T|=n,满足条件 m,n>0 和 m<n，并任意给定一个正整数 k，0<=k<m，从 T 中找出所有子串 v 发生的位置使得 v 满足条件 edit(v,P)<k。

定义 6 序列的联配（alignment）两个或多个符号序列按字母比较，尽可能确切地反映它们之间的相似或相异，称为序列的联配。

定义 7：如果 S 和 T 是两个序列，那么 S 和 T 的全局联配（alignment）A 可以用序列 S' 和 T' 来表示，其中：

（1）、$|S'|=|T'|$；

（2）、将 S' 和 T' 中的空字符除去后所得到的序列分别为 S 和 T，（例 S="$acbcdb$"，T="$cadbd$"，那么 S'="$ac--bcdb$"，T'="$-cadb-d-$"）；

联配就是把序列 S' 和 T' 上下罗列起来，相应的位置进行一一的比较。联配 A 的分值 *Score* 可以用如下的公式来表示：

$$Score=\sum_{i=1}^{l}\sigma(S'[i],T'[i])$$

其中 $l=|S'|=|T'|$；

定义 8. 对于两个序列 S 和 T，它们的全局最优联配 A 是指在 S 和 T 的所有相似性比较中最高分值 *Score* 所对应的排列。

定义 9. 所谓局部联配是指：给定两条序列 S 和 T，其中 $|S|=n$，$|T|=m$，设 α 为 S 的一个连续的子序列，β 为 T 的一个连续的子序列，找出这样的子序列 α 和 β，使得 α 和 β 的相似度是 S 和 T 中所有的这样的子序列的相似度的最大值。即，α 和 β 的相似度不小于 S 中其它的子序列 A 和 T 中其它的子序列 B 的相似度。

定义 10. 如果 x 和 y 是两个任意的字符，那么 $\sigma(x,y)$ 表示字符 x 和 y 在进行比较时所得的分值，称为一个记分函数。记分函数包括了当 x 为空字符或 y 为空字符的情况，在序列中一个所谓的空字符表示序列在此位置可能缺失了一个字符，我们用“—”来表示这种缺失。在不同的算法当中，记分函数可以有不同的记分方法。例如可以这样定义记分函数：$\sigma(x,x)=+2$，$\sigma(x,y)=\sigma(x,-)=\sigma(-,y)=-1$。

定义 11. 空位处罚　空位是指在某一排列的单个序列中，任意连续的尽可能长的空格。空位的引入是为了补偿那些插入或缺失，使序列的排列能更紧密地符合某种所期望的模型。空位的长度是指在空位中 indel 的个数。在序列中空位的数目用# gaps 来表示，所有空位中空格的数目（indel 操作的数目）用# space 表示。每引入一个空位，联配的分值都会有所扣除。

对于这些空位有很多罚分的规则，常见的罚分规则主要有两种：

恒定空位权值模型

这是最简单的一种规则，每个空位中的空格的分值设为零，即对任意的 x，s(—, x) =s(x,—)=0。这样联配的相似度就为：

$$\sum_{i=1}^{l}\sigma(S'[i],T'[i])+W_g\times(\#gaps)$$

其中 S'和 T'分别为 S 和 T 加入空位后的序列，|S'| = |T'| = l，Wg 为开放一个空位的罚分。

仿射空位处罚模型（Affine Gap Model）

这是最常用的一种罚分规则。空位处罚函数依赖于空位中空格的数量：用一个附加

的罚分比例去乘空位的长度，其中有两个参数，Wg 表示空位开放处罚，Ws 表示空位延伸处罚。仿射处罚函数可表示为：Wg+q×Ws，q 表示某一个空位的长度。这样联配的相似度就为：

$$\sum_{i=1}^{l}\sigma(S'[i],T'[i])+W_g\times(\#gaps)+W_s\times(\#space)$$

1.2.2 串匹配算法

串匹配算法可以分为两大类，精确串匹配算法和模糊串匹配算法。精确串匹配算法又可分为多模式串匹配算法和单模式串匹配算法。

单模式串匹配算法大致可分为四类：滑动窗口算法，自动机算法，计数算法和 hash 算法。但这四类划分不是严格的，例如在滑动窗口算法中经常用到后缀自动机来确定窗口跳跃的距离。

滑动窗口算法的基本思想是根据当前窗口的匹配情况确定下一窗口的位置，这类算法有 BM 算法【BOYER 1977】以及 BM 算法的各种改进算法【HORSPOOL 1980】【SUNDAY 1990】【SMITH 1991】，利用自动机确定跳跃距离的算法 BOM 算法、BAWG 算法等。此类最差时间复杂度为 O(mn),最好情况下时间复杂度 O(n/m)，平均时间复杂度为 O(n/m)【 Mike Liddell 1997】。

他们共同的特点是采用自右向左、跳跃滑动的策略。问题的关键在于如何用最短的时间计算出最长的跳跃距离。如果设 tj 为计算跳跃距离所需的时间，sj 为计算出的跳跃距离，则问题在于如何使得 sj/tj 的值最大。依此标准可大致判断出该类算法的优劣。

自动机类算法首先根据模式串构造自动机，然后扫描文本串，驱动自动机运行，到达终态表明发现匹配字符串。此类算法主要有：基于确定性自动机的算法【CROCHEMORE 1997】、KMP 算法【KNUTH 1977】；基于非确定性自动机的算法 shift-or 算法【WU AND MANBER 1992】。此类算法时间复杂度为 O(n)，空间复杂度为 O(m|Σ|)。自动机的存储是困扰该类算法的一个重要问题。

计数算法的时间复杂度与模式串字符的重复率成正比，当模式串中的所有字符均不同时，时间复杂度为 O(n)。当模式串字符重复率较高时，用 SIMD 指令集可以加速该算法。此类算法应用面较窄，但是能方便地应用到 k-误配串匹配中，这是该算法的一个显著优点。

Hash 算法的时间复杂度理论上为 O(n)，其关键在于构造好的 hash 函数，既简单又能显著降低冲突，从而提高其性能。

多模式串匹配算法可分为三类：跳跃类算法，自动机类算法，hash 类算法。在思想上，各类算法与单模式串匹配算法类似，时间复杂度也类似。

跳跃类算法【WM1994】(WM 算法)、SBOM 算法首先截取各个模式串使得得到的模式串等长，然后根据模式串集合建立跳跃数组。扫描时根据跳跃数组移动窗口。

自动机类算法【AC1975】首先根据模式串集合建立自动机，随后的过程类似于单模式串匹配算法。

Hash 类算法一般要经过两次 hash 才能确定是否有候选匹配项发生。

模糊串匹配算法可分为三类：动态规划算法，自动机算法，过滤算法。

简单的动态规划算法时间复杂度为 O(mn)，是模糊串匹配的一类核心算法。动态规划矩阵各元素之间的数据依赖关系限制了该算法的并行性。

模糊串匹配算法中的自动机与精确串匹配中的自动机有很大不同。模糊串匹配算法中确定性自动机的状态数目多到严重影响到算法的性能，因此，模糊串匹配算法多采用非确定自动机。基本的非确定自动机类算法复杂度为 O(kmn)。

过滤类算法首先将模式串分解成数个子串，然后按多模式精确匹配算法查找分解得到的子串，获得候选位置，然后对候选位置执行动态规划算法计算相似度。

1.3 体系结构对算法的影响及算法优化

程序的性能与计算机体系结构密切相关。本文只考虑单机体系结构对程序性能的影响。直观地看，要想提高程序的性能，需要在两个方面下功夫：尽可能多地执行指令、尽可能早地执行指令。这就是面向现代体系结构算法优化地两个方面：ILP 优化和存储优化，这主要涉及计算机系统地两大部分：CPU 和存储系统。

ILP 优化

ILP 优化的出发点是增大程序中可并行执行的操作的数目，尽可能将程序的资源需求与机器提供的资源弥合，其原则是减少和消除数据与控制相关。具体的方法有：循环展开、过程嵌入、软件流水、superblock、Hyperblock、trace scheduling、值预测等。在对算法进行优化之前，首先要了解计算机系统的体系结构特征。

处理器作为计算机的核心部件一直被给予最大的关注，为了达到单机的高性能，很多体系结构上的新技术被芯片的设计者广泛采用。其中，最常用的有超标量技术、超长指令字技术、超流水线技术、乱序执行、同时多线程技术、向量 IRAM 技术、单芯片多处理器技术、超传输技术等等。此外还有一些最新推出的技术，如 Intel 在 Xeon 和 P4 中采用的超线程技术、IBM 的双核芯片技术等等。提高应用程序的性能很重要的一点是如何最大限度的发挥处理器的利用率，充分利用处理器中的各项技术。处理器中不同的技术对程序性能的影响的幅度和方式有所区别，有些是对程序员透明的，不要程序员做任何工作；有些虽然透明，但可以通过调节程序以充分利用这些技术；有些则是针对程序的特点来改变软硬件环境的设置，从而提高性能。

使用 SIMD 技术【Bouknight 1972】【Lee1995】是现代体系结构的一个重要特征。SIMD 技术通过增加指令中完成的操作数、减少完成一个任务或程序所需的指令条数来提高处理器的性能。普通指令同时只能执行一次计算，所以虽然处理器是由许多个 ALU 构成，但这却并不能加快它的计算速度，因为只有一个 ALU 处于工作状态，无法充分

利用处理器中的计算资源。为了改进这种情况，各个处理器生产公司都推出了单指令多数据（Single Instruction Multiple Data，SIMD）扩展架构的 CPU。如 Intel 公司引入了 MMX（Multimedia Extension）、SSE（Streaming SIMD Extensions）和 SSE2 指令集【Intel v1】【Intel v2】【Intel v3】，AMD 的 3DNow!（3D no waiting）指令集【AMD1】【AMD2】【AMD3】，IBM 的 AltiVec【AltiVec】，Sun 的图像指令集（Visual Instruction Set，VIS）等等。龙芯也有自己的多媒体指令集【Godson】。

【CZN2004】的测试表明，使用 SSE2 指令集的 linpack 测试要比未使用 SSE2 指令集的 linpack 测试性能提高近 90%。

SWMMX 算法利用 MMX 指令集加速 Smith-Waterman 算法，一般情况下能获得 6 倍的加速比。

用位并行的方法模拟非确定性自动机，可以获得 w 倍的加速比，w 为机器字长。

存储优化

CPU 主频的提升带动了系统性能的改善，但系统性能的提高不仅仅取决于 CPU，还与系统架构、指令结构、消息在各个部件之间的传送速度及存储部件的存取速度等因素有关。特别是与 CPU 与内存之间的存取速度有关。与飞速发展的 CPU 主时钟频率相比，作为主存储器主要器件的 DRAM 已成为高速计算机的主要瓶颈，而且 CPU 和主储器之间的速度差距仍在不断扩大【Wulf 1995】。因此，充分利用存储层次结构的特点【Curt Schimmel】，提高数据局部性，是提高应用程序的性能的重要途径。

存储优化的软件方法分为三大类：容忍 cache 不命中延迟；优化存储带宽利用；减少访存或减少 cache 不命中数。

容忍 cache 延迟的方法有指令调度、软件预取【Aneesh 2001】等。

优化存储带宽利用的存储优化方法的思想是：只将程序使用频繁的数据调入 cache。其方法有编译时的 clustering 方法和着色方法，部分 cache 局部性方法【李明 1997】

减少访存的优化方法有：寄存器优化、数组收缩【Geo2002】、标量替换。

降低 cache 缺失率的方法有：优化程序的空间局部性、优化程序的时间局部性和减少 cache 冲突。按技术手段可分为循环变换、数组变换和数组 Padding【车永刚 2004】。

【Nishimura 2001】通过重新排列 AC 自动机状态在内存中的排列顺序，提高了 cache 命中率。可以获得一倍左右的加速比。

【Stephen】通过寄存器优化来优化 BF 算法，可以获得一倍以上的加速比。

【Nathan2004】通过压缩存储 AC 自动机的查询表，降低内存消耗，同时大幅降低了 cache 缺失率，最终可以获得 3 倍的加速比。

1.4 面向体系结构串匹配算法优化的主要挑战

优化串匹配算法受串匹配算法本身 ILP 的限制。

动态规划算法中，动态规划矩阵每个单元都依赖于它左、上、左上三个元素的值，

从而严重限制了该算法的数据并行性，消除或减弱或规避这种数据依赖关系是提高该算ILP 的关键。

自动机类算法，当前状态直接依赖于上一个活动状态，导致扫描文本时只能依次读取一个字符并进行状态转换然后读取下一个字符。算法中数据的依赖关系较强，导致无法直接并行化。

1.5 本文的贡献

本文的主要贡献如下：

(1) 基于 SIMD 指令集的动态规划算法数据并行性开发与实现研究。

本文分析了两类动态规划算法，Smith-Waterman 算法和 Zuker 算法的数据依赖性及其数据并行性。基于前驱计算思想，优化设计了 Smith-Waterman 算法，充分挖掘了算法的数据并行性，并基于 SIMD 指令集实现，数倍提高了该算法性能。通过设计辅助矩阵，有效组织数据，优化设计了 Zuker 算法，对于较大的数据规模，获得了接近理论值的加速比。

(2) 基于组合字符集的高性能 shift-or 算法研究

本文通过组合 NFA 的状态，建立 Σ_{short} 上的 NFA，状态转换单位提高为 short 型整数，显著降低了自动机状态转换次数，提高了计算效率。

(3) 面向位并行的低内存开销高性能关键词表达式匹配算法研究

本文面向位并行，利用 four-russian 和分治等优化方法设计了新的关键词表达式匹配算法，可提高计算效率、降低内存开销。

1.6 论文的组织

第一章中论述了串匹配算法的应用背景及串匹配算法和算法优化的基础知识，明确了本文的工作方向。

第二章介绍了基于 SIMD 指令集的动态规划算法优化，实现了基于 SSE2 指令集的 Smith-Waterman 算法和 Zuker 算法。

第三章研究介绍了单模式精确串匹配算法 shift-or 的优化。

第四章分析了关键词表达式匹配算法的研究现状，并提出了基于 four-Russian 技术的算法。

第五章总结本文的研究成果，并提出了进一步工作的展望和设想。

第二章 基于 SIMD 指令集的串匹配算法优化研究

本章主要讨论对串匹配算法中的动态规划算法的优化。动态规划对模糊串匹配算法有深远的影响，形成串匹配算法的一个基础类别。Smith-Waterman 算法和 RNA 二级结构预测的 Zuker 算法都是动态规划算法，有着不同的特征，同时又都是生物信息学的基础算法。本章讨论这两类动态规划算法的数据并行性。

2.1 Smith-Waterman 算法

最早在序列的联配问题中引入动态规划思想的是 Needleman 和 Wunsch，并且利用这种技术，提出了 Needleman-Wunsch 算法【Needleman andWunsch 1970b】，解决了序列全局联配的问题；在此基础上，Smith 和 Waterman 提出用于局部联配的 Smith-Waterman 算法【Smith-Waterman 1981】，Gotoh【1982】和 Myers【1988】又在这些算法的基础上进行了时间和空间的改进。

序列联配算法算法主要有两部分组成【Martin 2000】【 Ron 2001】：(1)计算所给两个序列相似分值，得到一个相似度矩阵（similarity matrix），也称作动态规划矩阵或得分矩阵；(2)根据相似度矩阵，回溯寻找最优的联配。我们主要考虑第一步。

给定两个序列 S 和 T，其中|S| = n，|T| = m，定义 $V(i,j)$为子序列 S[1] . . . S[i]和 T[1]...T[j]最优联配的分值。V(i,j)的计算公式如下：

前提条件：

$$\begin{aligned} &V(0,0)=0 \\ &V(i,0)=V(i-1,0)+\sigma(S[i],-) \\ &V(0,j)=V(0,j-1)+\sigma(-,T[i]) \end{aligned} \qquad (2\text{-}1)$$

$V(i,0)$表明序列 S 中的每一个字符都和 T 中的空格相匹配，$V(0, j)$与 $V(i, 0)$类似。

递归关系：

$$V(i,j)=\max\begin{cases} V(i-1,j-1)+\sigma(S[i],T[j]) \\ V(i-1,j)+\sigma(S[i],-) \\ V(i,j-1)+\sigma(-,T[j]) \end{cases}, \quad 1\leqslant i\leqslant n,\ 1\leqslant j\leqslant m \qquad (2\text{-}2)$$

在相似度矩阵中，任意一个元素的分值 $V(i,j)$依赖于其它的三个元素的分值：①、左上角的元素 $V(i-1,j-1)$；②、左边相邻的元素 $V(i,j-1)$；③上一行相邻的元素 $V(i-1,j)$。

对全局联配动态规划的基本策略稍作修改可以应用到局部最优联配的算法之中。这种联配的路径不需要到达搜索图的尽头，只需要在内部开始和终结。如果某种联配的分值不会因为增加或减少联配的数量而增加时，这种联配就是最佳的。这个过程依赖于记分系统的性质：因为某种路径的记分会在不匹配的序列段减少。当分值降为零时，路径

的延展将会终止，一个新的路径就会产生。这样就会得到许多独立的路径，它们以不匹配的序列段为界限。在这些路径中拥有分值最高的一个就是最佳的局部联配。

前提条件：

$$V(i,0)=0;$$

$$V(0,j)=0;$$

递归关系：

$$V(i,j)=\max\begin{cases}0\\ V(i-1,j-1)+\sigma(S[i],T[j])\\ V(i-1,j)+\sigma(S[i],-)\\ V(i,j-1)+\sigma(-,T[j])\end{cases}\quad 1\leqslant i\leqslant n,\ 1\leqslant j\leqslant m \qquad (2\text{-}3)$$

找出 i*和 j*，使得：

$$V(i^*,j^*)=\max_{1\le i\le n,1\le j\le m}V(i,j)$$

公式(2-3)与前面介绍的全局联配中的递归公式(2-2)的唯一不同之处是：在 max 函数中添加了一项“0”，该公式所表示的四种可能情况是：（假设最优联配 A=(a, ß)，其中 a、ß 分别为序列 S[1] ...S[i]和 T[1] ...T[j]的后缀。）

a =?、ß =?，则此时联配 A 的值为 0；

a≠λ、β=λ，且联配 A 最后的匹配为(S[i], T[j])，则此时联配 A 的值为 V(i-1,j-1)+s(S[i], T[j]);

a≠λ，联配 A 最后的匹配为(S[i], —)，则此时联配 A 的值为 V(i-1,j)+s(S[i], —);

ß≠λ，联配 A 最后的匹配为(—, T[j])，则此时联配 A 的值为 V(i,j-1)+s(—, T[j]);

在 max 函数中加入“0”选项是为了确保在计算中丢弃分值为负的前缀。计算完相似度矩阵，并且找到最大值(i* , j*)后，最优联配 a 和 ß 可利用回溯算法从(i* , j*)开始直到选项(i' , j'), 其中(i' , j') = 0。这样局部最优联配 a 和 ß 分别为 S[i'] ...S[i*]和 T[j'] ...T[j*]。

2.2 基于指令集的 Smith-Waterman 串匹配算法优化现状

基于指令集的优化主要应用在模糊串匹配算法中。主要集中在利用 SIMD 指令集开发算法的数据并行性。

Alpern ET al.【1995】提出了一种在 Intel paragon i860 处理器种利用 64 位指令并行化 Smith-Waterman 的方法，他将 64-bit 的 Z-buffer 寄存器分成四个不同的部分。利用这种方法，可以将一条查询序列与四条库序列同时进行比较，他们的试验结果表明同传统方法相比可以获得 5 倍的加速比。

Wozniak【1997】利用 Visual Instruction Set(VIS)技术（Sun UltraSPARC 微处理器）实现了 Smith-Waterman 算法，同使用普通指令集相比可以获得 2 倍的加速比。

Rognes【2000】利用 Intel 的 MMX 指令集设计实现的 Smith-Waterman 算法，同普通指令相比获得了 6 倍的加速比。下面我们介绍 SWMMX 的实现方法，对我们设计并

行算法会有启发意义。

SWMMX 算法利用了 Smith-Waterman 算法的一个特点，这个特点被 Green 首先利用来优化 Smith-Waterman 算法，并在 SWAT【Green 1993】程序中实现，即得分矩阵中大部分单元的 e 和 f 值为零,如果 h 的值小于阈值 q+r，那么同一行或同一列中下一单元 e 和 f 的值仍然是 0。

SWMMX 算法的数据排列方式如下图所示，将得分矩阵按列划分，同一列中 8 个单元组成一个向量，利用 MMX 指令集同时处理这 8 个单元。

Pseudocode	Comments
`FUNCTION SWMMX(MM, DSEQ, q, r, m, n)`	MM is a query-specific score matrix DSEQ is the database sequence q and r are gap open and extension penalties m and n are query and database sequence lengths

`CHAR c`
`INTEGER i,j`
`CONST INTEGER y = m/8`
	One database sequence symbol (c)
	Loop indices (i,j)
	Number of vectors along query sequence (y)

`VECTORS H,X,E,F,T1,T2,SCORE,HH[y],EE[y]`	Vectors (H,X,E,F,T1,T2) and arrays (HH,EE)

`CONST VECTORS BIAS = [4, 4, 4, 4, 4, 4, 4, 4]`
`CONST VECTORS QQ = [q, q, q, q, q, q, q, q]`
`CONST VECTORS RR = [r, r, r, r, r, r, r, r]`
	Score bias vector (constant)
	Gap open penalty vector (constant)
	Gap extension penalty vector (constant)

`FOR i=0 TO y-1 DO`
`{`
`  HH[i] = [0, 0, 0, 0, 0, 0, 0, 0]`
`  EE[i] = [0, 0, 0, 0, 0, 0, 0, 0]`
`}`
`SCORE = [0, 0, 0, 0, 0, 0, 0, 0]`
	Initialise HH-array of H-values from previous column
	Initialise EE-array of E-values from previous column
	
	Initialise score vector

`FOR j = 0 TO n-1 DO`
`{`
`  X = [0, 0, 0, 0, 0, 0, 0, 0]`
`  F = [0, 0, 0, 0, 0, 0, 0, 0]`
`  c = DSEQ[j]`
 	
`  FOR i = 0 TO y-1 DO`
`  {`
`  H = HH[i]`
`  E = EE[i]`
 	
`  T1 = (H RSHIFT 7)`
`  H = (H LSHIFT 1) OR X`
`  X = T1`
 	
`  H = ( H + MM[c][i] ) - BIAS`
`  H = MAX(H , E)`
 	
`  F = (H LSHIFT 1) OR (F RSHIFT 7)`
`  F = F - QQ - RR`
 	
`  IF (any element of F > 0)`
`  {`
`  T2 = F`
`  WHILE (any element of T2 > 0)`
`  {`
`  T2 = (T2 LSHIFT 1) - RR`
`  F = MAX(F , T2)`
`  }`
 	
`  H = MAX(H , F)`
`  F = MAX(H , F + QQ)`
`  }`
`  ELSE`
`  {`
`  F = H`
`  }`
 	
`  HH[i] = H`
`  EE[i] = MAX(H - QQ, E) - RR`
 	
`  SCORE = MAX(SCORE, H)`
`  }`
`}`
	For each symbol in the database sequence...
	
	Initialise X-vector for 1 round
	Initialise F-vector for 1 round
	Get one database symbol
	
	For each vector of 8 matrix cells along query sequence..
	
	Load previous H-vector from HH-array
	Load previous E-vector from EE-array
	
	Save previous H[7] for use below
	Shift H-vector and OR with H[7] from previous round
	Save old H[7] in X for next round
	
	Add score profile vector to H and subtract bias
	Check if score with database gap is better
	
	Calculate initial F-vector by shifting H and previous F
	Subtract single gap penalty
	
	Check if vertical gaps are possible
	Compute correct F-vector if necessary
	T2 is initial F-vector
	Repeat while any element of T2 is nonzero..
	
	Shift and subtract gap extension penalty
	Update F if new score is higher
	
	Update H if vertical gap is better
	Update F for use in next round
	
	Update F for use in next round
	
	Store H-vector in HH-array
	Store E-vector in EE-array
	
	Update Score with new H-vector if it is better

`RETURN MAX(SCORE[0], SCORE[1], .... SCORE[7])`	Return largest element in score vector

图 2.2.1 SWMMX 算法伪码

将 SWAT 优化应用到 SWMMX 算法中，如果一个向量中 8 个单元 h 值都小于阈值，

则这 8 个单元的 f 值都无需再计算而直接跳过这个向量，从而可以节约大量时间。否则，就要进行十分耗时的计算。算法伪代码如图 3.2.1：

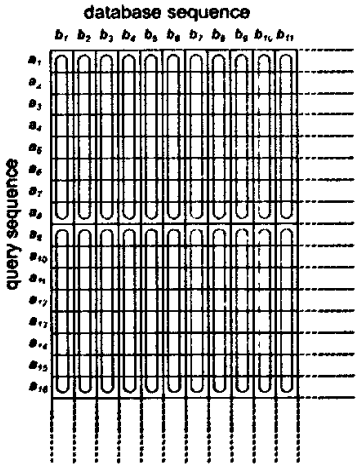


图 2.2.2 SWMMX 算法数据排列方式

SWMMX 算法的优点：同 ssearch 算法相比，加速比可达 6 倍。

SWMMX 算法的缺点：一，以一个字节作为一个数据单元，限制得分值必须小于 255，只能比较较短的序列。二，利用 SWAT 优化加速程序，使得该算法普遍性受到严重限制，即该算法只适合 e、f 值大部分为零的应用，当 e、f 值大部分不为零时该算法性能大大降低，不适合向其它动态规划算法推广。

2.3 基于 SSE2 指令集的 Smith-Waterman 算法研究

Smith-Waterman 算法是一种计算密集型算法，对它的并行的研究由来已久。我们首先利用前驱计算的思想减弱得分矩阵元素之间的数据依赖关系，分离出算法的并行操作，然后用 SSE2 指令集实现这些并行操作。

2.3.1 前驱计算

Srinivas Aluru 等【Aluru 1998】提出了一种利用前趋计算（Prefix Computation）的并行序列比较算法。其基本思路是：

给定两个生物序列 $A=a_1,a_2,\ldots,a_m$，$B=b_1,b_2,\ldots,b_n$，$m\leqslant n$。对于任意的字符 c_1，c_2，设定序列的得分函数 $f(c_1,c_2)$ 为：$c_1=c_2$ 时其值为 1；$c_1\neq c_2$ 时其值为 0；c_1 为'-'或 c_2 为'-'时其值为-1。我们按照如下公式构造动态规划矩阵 T，令 $T[0,0]=0$;

$$T[0,j]=\sum_{k=1}^{j} f('-',b_k) \tag{1}$$

$$T[i,0]=\sum_{k=1}^{i} f(a_k,'-') \tag{2}$$

$$=T[i-1,0]+f(a_i,'-') \tag{3}$$

$$T[i,j]=\max\begin{cases}T[i-1,j]+f(a_i,'-')\\T[i,j-1]+f('-',b_j)\\T[i-1,j-1]+f(a_i,b_j)\end{cases} \quad (4)$$

利用公式(4)在计算第 i 行的元素 $T[i,j]$时要用到第 i-1 行的值 $T[i-1,j]$和 $T[i-1,j-1]$。而此时第 i-1 行的这两个值已经提前计算完毕。对公式(4)进行变形，把提前计算出来的项提取出来：

令 $$w[j]=\max\begin{cases}T[i-1,j-1]+f(a_i,'-')\\T[i-1,j-1]+f(a_i,b_j)\end{cases} \quad (5)$$

则 $$T[i,j]=\max\begin{cases}w[j],\\T[i,j-1]+f('-',b_j)\end{cases} \quad (6)$$

再令 $$x[j]=T[i,j]-\sum_{k=1}^{j}f('-',b_k)$$

$$=\max\begin{cases}w[j]-\sum_{k=1}^{j}f('-',b_k)\\T[i,j-1]-\sum_{k=1}^{j-1}f('-',b_k)\end{cases}$$

$$=\max\begin{cases}w[j]-\sum_{k=1}^{j}f('-',b_k)\\x[j-1]\end{cases}$$

$$y[j]=\sum_{k=1}^{j}f('-',b_k) \quad (7)$$

$$z[j]=w[j]-y[j] \quad (8)$$

由于公式(7)可以提前计算得到，则 $z[j]$也就可以提前计算得到。那么

$$x[j]=\max\begin{cases}z[j]\\x[j-1]\end{cases} \quad (9)$$

从公式(9)可知，$x[j]$的值可以通过并行前趋计算一个 max 二元函数来得到，那么动态规划矩阵 T 的计算公式可变为：

$$T[i,j]=x[j]+\sum_{k=1}^{j}f('-',b_k)$$

$$=x[j]+y[j] \quad (10)$$

这样利用公式(5)、(7)、(8)、(9)和(10)就可以通过上一行的值来计算当前行中的某一项分值。如果考虑到空位处罚的情况：对任意的字符 c，令 $f(c,'-')=f('-',c)=-g$。则，

$$x[j]=\max\begin{cases}w[j]+gj\\x[j-1]\end{cases} \quad (11)$$

$$T[i,j]=x[j]-gj \quad (12)$$

利用公式（5）（11）（12）即可按行计算出整个得分矩阵。

前驱计算的最大贡献在于将公式中串行部分和并行部分分离出来，为动态规划算法

的并行化指出了一条道路。

2.3.2 SWSSE2 算法设计

给定两条序列，库序列 T[0…n]和查询序列 S[0…m]：

初始条件：

H(i, 0) = E(i, 0)=F(i,0) = 0;(0 <= i <= m)

H(0, j) = =E(0,j)=F(0, j) = 0;(0 <= j <= n)

递归过程：

E(i, j) = max {E(i, j–1) – r, H(i, j–1) – q– r， 0}; .....(1)

F(i, j) =max {F(i–1, j) – r, H(i–1, j) – q r， 0}.....(2)

H(i,j)= max{0,H(i–1, j–1) + σ (S[i],T[j]),E(i,j),F(i, j)}·····(3)

其中,H、E、F 均为(m+1)*(n+1)的矩阵，$\sigma(S[i], T[j])$为得分函数，q表示空位开放处罚，r表示空位延伸处罚。

观察 SW 算法得分矩阵 H 中各单元之间的依赖关系，可以看出在同一条反对角线上的各单元之间不存在依赖关系，它们可完全并行计算出来，这就是波前式并行算法的基本思想。由于输入输出数据排列不规则，采用这种算法不能充分利用 cache 因而造成严重的时间浪费，从而导致这种算法的失败。我们根据前驱计算的思想，通过改进 F 的计算，设计并实现了 SWSSE2 算法。其数据排列方式如图 2.3.2.1 所示。

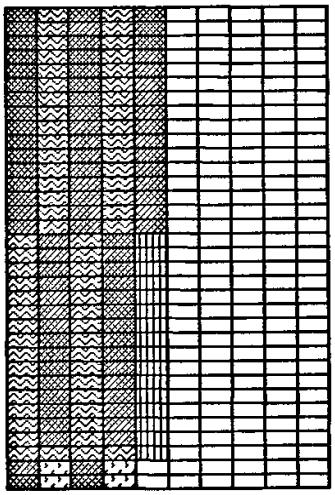


图 2.3.2.1 SWSSE2 算法的数据排列方式

设当前要计算的向量为 E(t…t+15,j), H(t…t+15,j),F(t…t+15,j)，从 0 到 j 列各单元以及第 j 列的 0 到 t–1 个单元均已计算完毕。如图所示花纹填充的空格表示已经计算完毕，白色的空格表示还没有计算到的单元，竖线填充的空格表示正在计算的一组单元。

首先根据式(1)计算得 E(t…t+15,j)。参考式(3)，对 H 的计算分解两步进行，第一步，只考虑左侧一列单元的贡献，不考虑上方单元的贡献；第二步，考虑上方单元贡献：

$$H'(i,j)=\max\{0, H(i-1,j-1)+s(S[i],T[j]), E(i,j)\};\quad (t<=i<=t+15) \qquad \cdots\cdots(4)$$

$$H(i-1,j)=\max\{H'(i-1,j), F(i-1,j)\} \qquad \cdots\cdots(5)$$

这样，H’ 可以并行地计算出来。

根据式(2)可得

$$F(i,j)=\max\begin{cases} E(i-1,j)-r \\ \max\{H'(i-1,j), F(i-1,j)\}-q-r \end{cases}$$

即 F(i,j)=max{F(i-1,j)-r,H’(i-1,j)-q-r,F(i-1,j)-q-r}

因为 F(i-1,j)-r 恒大于 F(i-1,j)-q-r,所以

$$F(i,j)=\max\begin{cases} F(i-1,j)-r \\ H'(i-1,j)-q-r \end{cases}$$

令 $p[i]=F[i,j]+i*r$，$w[i]=i*r$，$z[i]=H'[i-1,j]-q-r+w[i]$ 则：

$$p[i]=\max\begin{cases} p[i-1] \\ H'[i-1,j]-q-r+w[i] \end{cases}$$，即：

$$p[i]=\max\begin{cases} p[i-1] \\ z[i] \end{cases} \qquad \cdots\cdots(6)$$

$$F[i,j]=p[i]-w[i] \qquad \cdots\cdots(7)$$

上述过程中，z[i], 即 H’ [i–1,j]–q–r+w[i]可以事先计算出来。

根据式（3）即可计算出最终需要的分值矩阵 H。在上述算法中，并没有使用 SWAT 假定，即无论 E、F 是否多数情况下为零，上述算法性能不受影响。

虽然在计算 F 的过程中没有完全消除依赖关系，但计算 F 矩阵中 16 个单元的过程中，只需两次算术运算和少量的逻辑运算即可完成。

2.3.3 SWSSE2 算法实现

首先，根据查询序列建立相似度表，设查询序列长度为 m，字符集大小为 δ，则相似度表为 δ *m,对于 DNA 序列，δ 为 4；对于蛋白质序列 δ 为 20。该表可在读取查询序列的同时建立。因为所有的操作均为无符号数运算，所以操作数中不能出现负数。对负操作数要做特殊处理，例如–1 可用 3–4 来实现(4 称为基数)。这样相似度表中均为正数，使用时再减去一个基数即可得到真正的相似度值。算法采用按列计算方式。

核心算法可描述如下：

```
for(j=0;j<n;j++){   //计算第 j 列的得分值
    for(i=0;i<m/16;i++)   { //计算一列中一组(由 16 个单元组成),E、H、z、p、F、
max16 均为包含 16 个元素的向量
      计算只对左侧单元有依赖关系的向量 E;
      计算只对左侧和左上侧单元有依赖关系的向量 H’;
   利用前驱计算思想计算只对上方单元有依赖关系的向量 F;
   计算向量 H;
      更新最大得分值向量 max16;
}
}
```

其中计算 F 的算法如图 4.3.1 所示

```
{
        F=F-QQ-RR +JR ;.          /*为 F 的前驱计算做准备*/
/*前驱计算开始*/
        Fi=F Lshift 1             /*F 左移一个单元*/
        while(存在（Fi 中的一个值>F 中的对应单元的值)）{
        {
          F=max (F, Fi);          /*F[i]取 F[i]与 F[i-1]中较大者*/
          Fi=F  Lshift 1          /*F 左移一个单元*/   ;
        }
/*前驱计算结束*/
        F=F-JR;                   /*得到真正的 F 各单元的值*/
/*F 计算完毕*/
}
```

图 2.3.3.1 SWSSE2 算法计算 F 向量得伪码

其中各向量长度均为 16，JR 为 W[i](0<=i<=15)，即 0, r, 2*r,…,15*r 构成的向量。

2.3.4SWSSE2 试验结果

库序列长度		300	400	500	600
时间 (单位 ms)	ssearch	2.3	2.8	3.7	4.3
	SWMMX	0.71	0.94	1.2	1.4
	SWSSE2	0.40	0.53	0.71	0.81

表 2.3.4.1 SWSSE2 算法试验结果

可以看出 SWSSE2 算法的速度与 ssearch 算法、SWMMX 算法的速度都基本正比与序列 的长度。在序列相似性不显著的情况下，即得分矩阵中 F 的值大部分大于零的情况下，SWSSE2 算法性能基本上是 SWMMX 算法的 1.5 倍。

2.4 基于 SIMD 指令集的二级结构预测算法优化

Zuker 的核心算法是一种时间复杂度为 $O(n^3)$的动态规划算法，一般称之为简单动态规划算法，可以用如下公式表示：

$$m[i,j]=\begin{cases}0 & j=i,0<i\le n\\ \min_{i\le k<j}\{m[i,j],m[i,k]+m[k+1,j]\} & 1\le i<j\le n\end{cases}$$ 公式 4.2.1

矩阵中每个元素的值依赖于同一行左侧和同一列下方的元素。下图表示了这种计算关系。

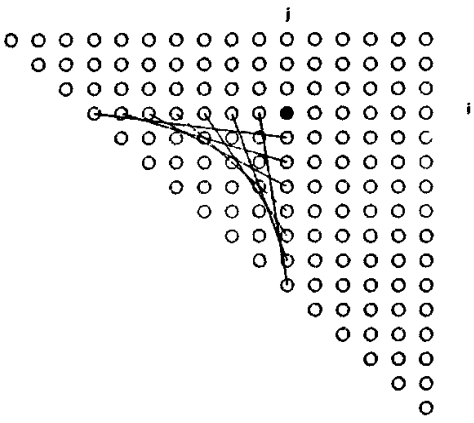


图 2.2.1 简单动态规划算法得数据依赖关系

对于矩阵中的某个元素 m[i,j],其值的计算类似于两个向量内积的计算，只是把内积运算中的乘法换成加法，加法换成取最小值运算。设有 n 维向量 $X=\{x_1,x_2,\ldots,x_n\}$和 $Y=\{y_1,y_2,\ldots,y_n\}$,定义其内积运算为 $dy(X,Y)=(x_1+y_1)\bullet(x_2+y_2)\bullet\cdots\bullet(x_n+y_n)$,此处+表示普通加法，$\bullet$ 表示取最小值运算。则用于加速内积运算的技术可以用来加速此类动态规划算法。更进一步，加速矩阵乘的技术经过适当改造也可以用来加速此类动态规划算法【Cherng2005】【刘方爱 2001】。在矩阵乘算法中，SIMD 指令集有成功的应用【ATLAS】。我们将之应用到本算法中。

2.4.1 二级结构预测算法的数据并行性挖掘

用 SIMD 指令集优化 Zuker，有三种不同的方式。

SIMD 指令集对存储有比较严格的要求，它要求每次访问的 16 字节数据必须连续存储。回顾公式 4.5.1,m[i, j]的计算需要本行和本列的向量。由于二维数组是按行存放，故同一列的元素在内存中不连续。为了使之适应 SIMD 指令集的存储方式，我们首先将数组 m 转置，得 mr 数组。这样原来得公式变为：

$$m[i,j]=\begin{cases}0 & j=i,0<i\le n\\ \min_{i\le k<j}\{m[i,j],m[i,k]+mr[j,k+1]\} & 1\le i<j\le n\end{cases}$$

$$mr[j,i]=m[i,j];$$

这种方式比较直观易懂，算法有很好的局部性。但是要增加 $O(n^2)$的辅助空间，同时每次计算 m[i,j]的值，都要同时更新 mr[j,i]的值，增加了额外的操作。

再来看另一种方式。

我们采用按行自低向上的计算方式，假设 m 矩阵的每个元素都是 int 类型整数。则一个 __m128i 类型的数可以同时处理连续四个元素。假设当前处理的四个元素是 m[i,j],m[i,j+1],m[i,j+2],m[i,j+3],那么，从第 i+1 行到第 n 行的元素都已计算完毕，第 i 行中 m[i,i..j-1]都已计算完毕。用 v[i,j]表示向量（m[i,j],m[i,j+1],m[i,j+2],m[i,j+3]），用 v4[i,j]表示向量(m[i,j], m[i,j], m[i,j], m[i,j])。m 对 m[i,j..j+3]的计算分为两个部分，第一部分为

向量计算部分，第二部分为标量计算。如图2.4.1所示：

首先是向量计算部分

For t=i to j

v[i,j]=min{v[i,j], add(v4[i,t],v[t,j])}

End for

向量部分计算完毕。余下的蓝色部分不能直接放入SSE2寄存器，采用标量计算。

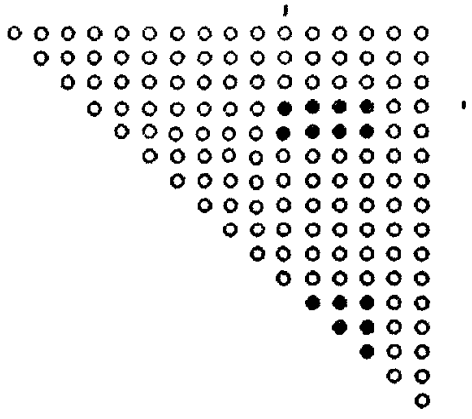


图2.4.1 第一种计算方法得数据排列方式

采用这种方法，可以不需要辅助存储空间。缺点是每次计算都有标量部分，不能充分使用SSE2指令。

我们由第二种方法扩展到第三种方法。

首先看标准的按行计算程序。

```
for  i=n-1  to  1
   for  j=i+1  to  n
      for  k=i  to  j-1
      D[i,j]=min(D[i,j],D[i,k]+D[k+1,j])
```

我们将它 改造为具有向量形式的程序。

```
for  i=n-1  to  1
   for  k=i  to  n-1
      for  j=k+1  to  n
      D[i,j]=min(D[i,j],D[i,k]+D[k+1,j])
```

这样内层循环具有向量计算的形式，可以方便地利用SIMD指令集计算。

2.4.2 试验结果与性能分析

我们采用第一种方法实现基于SSE2的Zuker算法。测试平台为Intel和AMD两种。平台配置如下表所示：

	CPU	L1cache	L2cache	OS

Intel	Xeon 2.4G	8KB/64B/4	512KB/64B/8	Red Hat Linux
AMD	Opteron 1.6G	64KB/64B/2	1024KB/64B/2	United linux1.0

表 2.4.2.1 测试环境参数表

标准程序用 c 语言编写，在 Intel 平台上用 ICC 编译器编译，优化选项为 O3，在 AMD 平台上用 gcc3.2,优化选项为 O3。基于 SSE2 指令集的程序采用嵌入式汇编，在 Intel 和 AMD 平台上均采用 gcc 编译器，优化选项为 O1。下表为测试数据。S 表示标准实现。

时间(s)	Xeon			Opteron		
矩阵规模	S	SSE2	S/SSE2	S	SSE2	S/SSE2
1000	2.05	1.17	1.75	1.85	1.19	1.55
2000	19.04	9.11	2.09	22.82	9.91	2.3
3000	102.94	29.45	3.4	93.83	31.24	3.0
4000	339.72	69.18	4.9	236.96	75.72	3.1

表 2.4.2.2 标准指令集与 SSE2 指令集实现的简单动态规划算法执行时间

对其 cache 性能进行分析，测试平台为 Opteron 系统。

标准程序的 cache 性能。

矩阵规模	cache access	cache miss	cache miss ratio
1000	1045433	97748	0.0935
2000	8736314	1934596	0.221443
3000	30066641	8637760	0.287287
4000	71722712	23020200	0.320961

表 2.3.2.3 标准 c 实现的简单动态规划算法在 Opteron 系统上的 cache 性能

基于 SSE2 的程序 cache 性能

矩阵规模	cache access	cache miss	cache miss ratio
1000	2052348	22569	0.010997
2000	16378260	180340	0.011011
3000	55138675	604166	0.010957
4000	127361191	1414260	0.011104

表 2.3.2.3 基于 SSE2 指令集的简单动态规划算法在 Opteron 系统上的 cache 性能

从试验结果可以看出，利用 SSE2 指令集，虽然其访存次数增加了 1/2，但 cache 缺失率大大降低，远远低于原始算法。利用 SSE2 指令集，理论上加速比应为 4，但因为访存次数的增大，抵消了部分性能提升，因而实际加速比基本都小于 4，只有当矩阵规模超过一定规模时，加速比才开始超过 4。利用第二和第三种方法，可以进一步降低访存次数，因而能够带来更大的性能提升。

2.5 讨论与小结

动态规划算法解决模糊串匹配算法和序列联配的一个有力工具。

动态规划算法具有较强的数据依赖关系，因而严重限制了该类算法本身的数据并行性。同时动态规划算法属于计算密集型算法，对计算资源有潜在的巨大需求。弱化或者规避动态规划矩阵元素之间的数据依赖关系，将能够大大提升算法的数据并行性。通过合理组织动态规划矩阵各元素的计算次序，可以大大降低 cache 缺失率或者 cache 缺失次数。

利用算法本身的性质，在某些情况下可以避免不必要的计算，例如 SWMMX 算法利用 Smith-Waterman 算法的 swat 特征。虽然此种方法能提高算法的速度，但它没有变动算法的数据依赖关系，只是利用算法本身的性质，因而不具有推广性。

我们将动态规划算法分为两大类来讨论：算法时间复杂度为 $O(n^2)$的算法和算法时间复杂度在 $O(n^3)$或以上的算法。

以 Smith-Waterman 算法作为第一类算法的代表来讨论。此类算法有很强的数据依赖关系，同时较差的数据并行性。分解、减弱或规避数据依赖关系是解决问题的关键。我们用前驱计算的思想把并行操作部分同串行超作分离开来，分解之后，大部分工作可以在并行操作种完成，少量运算留在串行操作中完成。这种方法具有较好的推广性。

以 Zuker 算法作为第二类算法的代表。此类算法虽然也有很强的数据依赖关系，但每个单元的计算都可看作两个向量的内积，因而本身具有较好的数据并行性。解决此类问题的关键为如何组织数据，使得 SIMD 指令集能高效地将两个向量加载到寄存器中。我们采用一个辅助矩阵来解决数据存放的问题，并分析了此中方法下的性能和 cache 使用情况，当算法规模增大时，实际加速比逐渐接近理论加速比。通过重新组织算法计算次序，也可以使得算法对数据的访问方式适合 SIMD 指令集的访问方式。

综上所述，通过分解动态规划算法的数据依赖关系，可以有效提高第一类算法的数据并行性；通过组织数据、增加辅助矩阵的方式，可以使得该算法适合 SIMD 指令集实现并有效提高数据命中率，最终提高计算性能。

第三章 基于组合字符集的高性能 shift-or 算法研究

Shift-or 算法是用位并行的方法模拟非确定自动机。基本的 shift-or 算法每次读入一个字符，我们通过组合 NFA 的状态，使得扫描文本时可以一次读入两个字符，而只进行一次状态转换。

3.1 位并行算法研究现状

3.1.1 基于非确定性自动机的算法

非确定性自动机 NFA 具有十分简洁的形式，下图是根据模式串 P＝“GCAGAGAG”构造的非确定性自动机，

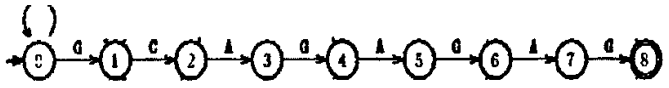


图 3.1.1.1 由“GCAGAGAG”构造的 NFA

NFA 具有多个活动状态，扫描文本串 T 时，每读入一个字符，需更新 NFA 中的所有状态，耗时 O(m),而文本串中共 n 个字符，所以总的时间复杂度为 O(mn)。建立 NDFA 需要 O(m)的空间。可见，直接利用 NFA 时间复杂度与 BF 算法最坏情况下时间复杂度相同，并没有改善算法的性能。

在非确定自动机中，在同一时刻，每个结点只有两种状态，活动和非活动，可以用 1 表示该结点活动，0 表示该结点为非活动状态，由此产生了用位并行来模拟非确定自动机的算法。

3.1.2 shift-or 算法

下面首先以一个例子介绍 shift-or 算法【WU AND MANBER 1992】的运算过程。

设 P＝ “GCAG”,T=“AGCAGT”, Σ={A,G,C,T}

首先建立掩码表（cancel mask table)。表格的列数等于模式串的长度，行数等于字符集中字符的个数。列的编号从右往左递增。对于某个字符 a，如果 a 在 P 中出现，在 a 这一行，a 在 P 中出现的位置填 0，其它位置填 1。如 A 在模式串 P 第三个位置出现，则 A 行 3 列填 0，其它位置填 1。

Cancel mask table				
	4	3	2	1
A	1	0	1	1
G	0	1	1	0
C	1	1	0	1
T	1	1	1	1

表 3.1.2.1 shift-or 算法中根据模式串"GCAG"构造得掩码表

mask 表计算公式表示如下

$$mask[c][i] = \begin{cases} 0, if\ (c = Pi) \\ 1, otherwise \end{cases}$$

下面用到 matching register ，大小为|P|即 mr 有 m 个 bit，各个 bit 从右向左编号。初始化各位置 1，代表各个状态为 false。

匹配时从 T 中读入字符 a，根据 cancel mask table 找到 mask[a]，将 mr 左移一位并与 mask[a]相或，结果放回 mr 中，如果 mr 最高位为 0 则报告在 T 中发现了 P。

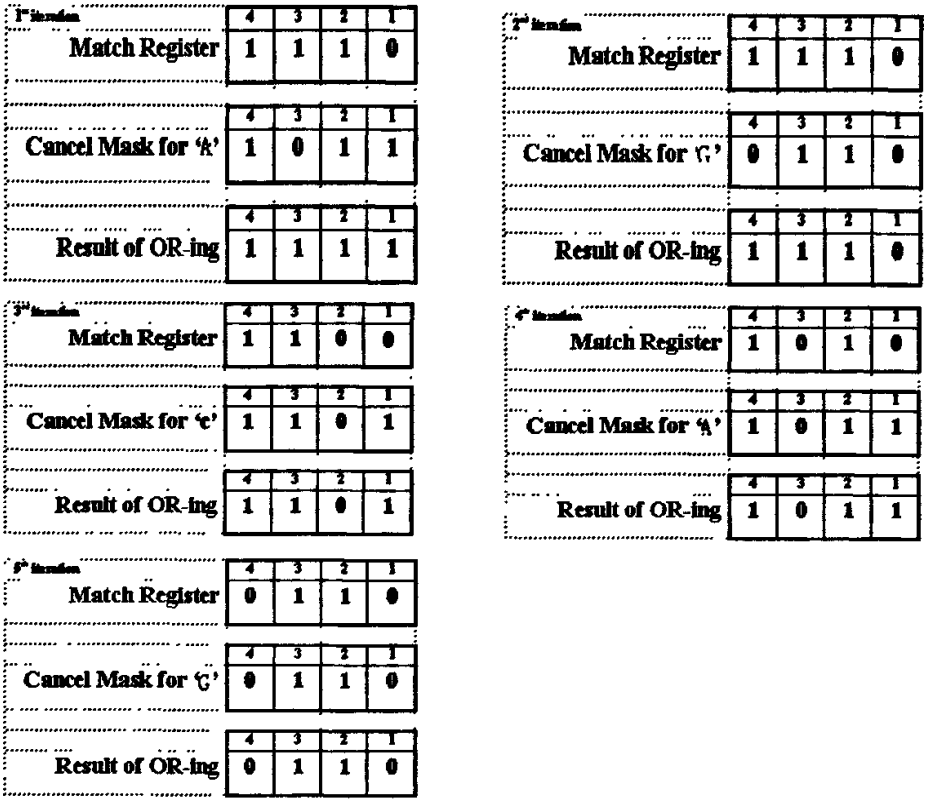


图 3.1.2.1 shift-or 算法匹配过程

```
        1 2 3 4 5 6
T:      A G C A G T
P:        G C A G
        匹配结果
```

下面我们进行更进一步的分析。设 R 为长度为 m 的位串。R_j 代表处理过 T[j]后 R 的值。它记录了所有与 j 点文本串匹配的 P 的前缀。

$$R_j[i] = \begin{cases} 0 & \text{if } x[0,i] = y[j-i,j], \\ 1 & \text{otherwise.} \end{cases}$$

如下图所示：

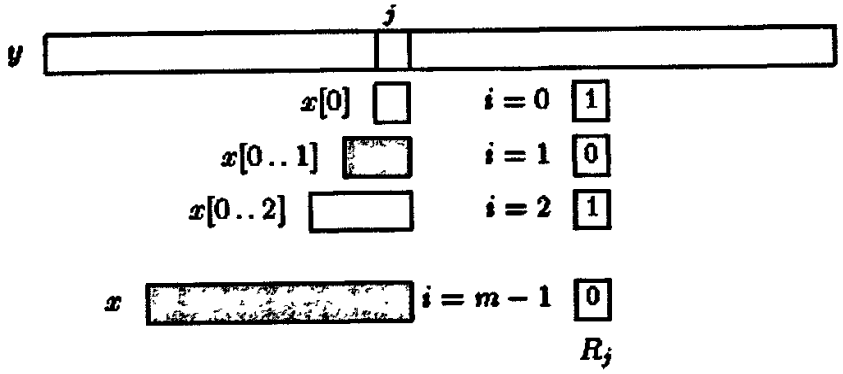


图 3.1.2.2 matching register 各位得含义

R_{j+1} 可由 R_j 递推得到：

R_{j+1}[I+1]=0 if R_j[i]=0 and T[i+1]=P[j+1]

否则，R_{j+1}[i+1]=1。.....shift-or(10)

如果 R_{j+1}[m-1]=0 则报告匹配成功。

由公式 shift-or(10)可以得到 ： R_{j+1}=*SHIFT*(R_j) *OR mask*[*T*[*j*+1]]

shift-or 算法简单易行，所以在多模式串和近似串匹配也得到广泛得应用。算法预处理时间和空间均为 $O(m+\sigma)$。匹配过程需时间 O(n)。

3.2 改进的 shift-or 算法: D(Double)shift-or 算法

对于基于自动机的算法，算法复杂度为 O(n)，算法所需时间只与文本串 T 的长度 n 有关，而与模式串 P 的长度和字符集Σ无关。这里取 n=strlen(T)×8/(sizeof(Σ)×8)，一般取字符集为 ASCII 码，即Σ=char。算法的空间复杂度 O(m|Σ|)，是模式串长度 m 和字符集Σ大小|Σ|的函数。要改进此类算法的性能，只能减少文本串 T 的长度，或者减少查询自动机状态的次数。很明显，T 的长度的是固定的，但是如果我们将两个字节即 short int 类型作为一个基本元素，逻辑上 T 的长度变为 n/2。基于这种思路，我们可以将Σ_{char}

上的自动机改变为Σ_{short}上的自动机。下面约定n_{char}表示字符集 char 下的字符串 T 长度，n_{short}表示字符集 short int 下的字符串 T 长度。

例如对于模式串 P= “GCAGAGAG”，对应的Σ_{char}上的自动机如图所示：

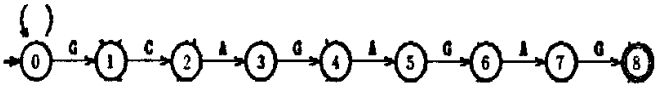


图 3.2.1 由 “GCAGAGAG”构造的 NFA

它对应的Σ_{short}上的 NFA 为

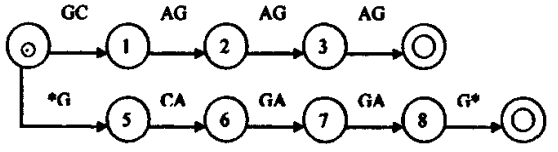


图 3.2.2 由 “GCAGAGAG”构造的Σ_{short} 上得 NFA

当字符串长度为奇数时，例如对于 P=”GCAGAGA”,构造的Σ_{short}上的 NFA 为

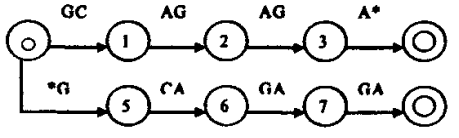


图 3.2.2 由 “GCAGAGA”构造的Σ_{short} 上得 NFA

比较这两种字符集上的 NFA 可以看出，Σ_{short}上的自动机比Σ_{char}上的自动机状态数，多 1，但是根据前面的分析，匹配阶段的时间复杂度仅与 n 有关，由于$n_{short}=n_{char}/2$,故利用Σ_{short}上的 NFA 所需的基本操作是利用Σ_{char}上的 NFA 所需的基本操作的一半。

算法可分为两个阶段，第一阶段为预处理，构造字符集Σ_{short}上的自动机，并根据此自动机按照多模式串 shift-or 算法建立掩码表。第二阶段根据建立的掩码表利用多模式串 shift-or 算法模拟 NFA 运行。

第一阶段，分两种情况处理，一种情况是 m_{char} 为偶数，所构造的自动机如图 5.1.2 所示；另一种情况是 m_{char} 为奇数，构造的自动机如图 5.1.3 所示。

以 m_{char} 为奇数时为例。把构造的自动机视为两个模式串 P*和*P 的组合，*为通配。依次从 P*和*P 中取出一个字符(字符集Σ_{short}上的字符)组成一个新的字符串 P’，例如 P=GCAGAGA 时构造的 P’= ‘*G’ ’GC’ ‘CA’ ’AG’ ‘GA’ ‘AG’ ‘GA’ ‘A*’。m_{char} 为偶数时类似。然后根据 P’ 构造字符集Σ_{short}掩码表。例如 MASK[‘AG’]=11101011。算法伪码如下：

```
m=strlen(P);
int MASK[65536],i;
memset(MASK,255,sizeof(int)*65536);
int pre(int *MASK, char *P)
{
if(m&1)
{//如果模式串的长度为奇数
        int final;
        char c=P[0],e=P[m-1];
        for(i=0;i<255;i++)
        {
                pos=i<<8+c;
                MASK[pos]&=~1;
                pos=e<<8+i;
                MASK[pos]&=~(1<<m);
        }
        unsigned short *p=P+1;
        for(i=0;i<m-1;i++)
        {
                p=P+i;
                MASK[*p]&=~(1<<(1+i));
        }
        final=-1;
        final&=((~(1<<m))&(~(1<<m-1)));
        return final;
}
else
{//
        略
}//end if(m&1)
}//end function
```

第二阶段，扫描文本串 T，每次读出两个字节，即一个 short 类型的整数。

```
short *ps=T;
for (state = ~0, j = 0;   *((char*)ps+1)!=0;j+=2) {
        state = (state<<2) | MASK[*ps];
        if (state < final)
           output(j - m + 1);
        ps++;
 }
```

3.3 试验结果

所有测试在 AMD Opteron 平台上进行，平台参数为：CPU 1.6GHZ；主存 3GB；L1 cache 2 路组相联 64KB；L2 cache 2 路组相联 1024KB。编译器采用 gcc 3.2.2，优化选项为 O3，操作系统为 United LINUX 1.0。测试数据集选自英文小说《War and Peace》和《Wuthering Heights》，数据集一文本长度 100KB，数据集二文本长度 640KB，数据集三

文本长度 10MB。

时间(ms)	Shift-or	Dshift-or
数据集 1	0.0275	0.0208
数据集 2	0.178	0.118
数据集 3	2.8	1.8

表 3.3.1 shift-or 算法与 Dshift-or 算法执行时间

Cache 分析

	Shift-or		Dshift-or	
	Cache access	Cache miss	Cache access	Cache miss
数据集 1	436	3	326	4
数据集 2	2768	23	2068	28
数据集 3	41245	336	33062	453

表 3.3.2 shift-or 算法与 Dshift-or 算法 cache 性能比较

3.4 算法分析

算法时间复杂度为 O(n)，但比较次数是原始算法的 1/2，内存开销为 64K。理论上算法消耗时间是原始算法的 1/2，但因为 cache 缺失率高于原始算法，故性能提高 1/3 左右。

3.5 小结

本节通过合并 NFA 的状态，减少了扫描过程中查询状态的次数，虽然增加了 cache 缺失率，但同时降低了 cache 缺失的绝对次数，减少了访存次数和比较次数。因而加速了算法。

第四章 基于 four-Russian 技术优化关键词表达式匹配算法研究

本章讨论的关键词表达式匹配算法主要应用于网络内容分析中，首先来看一下网络内容分析的算法的特点。

4.1 面向网络内容分析的算法复杂度分析

网络内容分析具有很强的实时性要求。网络内容分析系统对查询的执行分为三个阶段【谭建龙 2003】:

第一阶段. 初始化：针对未来全部网络连接（可能是无穷的网络连接），执行全部查询的预处理；

第二阶段. 数据流初始化：针对单个网络连接，执行全部查询的附加预处理；

第三阶段. 扫描数据流：当网络连接的每一个数据块到来时，执行全部的登记查询。

第一阶段的预处理工作所消耗的时间在算法的全部运行时间中所占的比例比较少，同时由于第一阶段只执行一次，所以我们通常不需要考虑它的算法复杂度。

如果网络连接的规模不是很大，对于单个网络连接而言，第二阶段的处理时间比较小。这情况下，通常不需要考虑第二阶段的算法复杂度。但是在网络内容分析中，每个网络连接的数据虽然比较少，却可能同时存在百万级的并发网络连接，则第二阶段的算法的性能也会对整个体统造成巨大影响。

第三阶段是需要多次重复执行的操作，针对每个网络连接的每个数据块都需要执行一次，所以它是网络内容分析系统中最关键的部分，是影响整个系统性能的主要因素。

该算法的内存开销可以划分为两部分。一部分是对每个网络连接都必须分配的，这个空间复杂程度会随着处理网络连接数目的增加而线性地增长；另一部分在处理多个网络连接时可以共享，则这种空间不会网络连接数目的增加而增加。

4.2 关键词表达式匹配算法研究现状

关键词表达式（query expression, keywords expression, predicate algorithm for rule）是搜索引擎（web retrieve）和数据库等使用的查询语言，在安全操作系统、入侵检测、网络内容分析中和 publish/subscribe 系统中也有广泛应用。

4.2.1 关键词表达式匹配定义

关键词表达式（Query Expression，Query Terms，Keywords Expression）定义如下【谭建龙 2003】:

<query_expression>::=<rule> or <query_expression>

<query_expression> ::= <rule>

<rule>::=<KeyEvent>and <rule>

<rule> ::= < KeyEvent >

直观地说,一个字符串如果出现在文本中,则表示这个字符串的事件(KeyEvent)发生了。一个规则(rule)是多个事件的"与"，也就是只有这些事件全部发生，才认为这个规则被满足了。我们定义的关键词表达式匹配对事件出现的次序没有限制。一个关键词表达式匹配是多个规则的"或"，也就是只要有一个规则被满足，我们就说这个表达式被满足了。

关键词表达式匹配（Expression Matching , Predicate Matching，event matching）定义如下：

对于一个表达式即一个规则集合，给定一个事件序列流，关键词表达式匹配算法的目标是找出所有满足的规则和规则的出现位置。

4.2.2 研究现状

关键词表达式匹配算法基本可以分为三类【E. Hanson and J. Widom1993，1999】【E. Hanson 1990】【Walid Rjaibi 2002】：基于索引的算法，基于树的算法和基于自动机的算法。基于索引的方法有计数法【J. Pereira 2000】【Kumar 1994】, The Hanson et al. algorithm【E. Hanson 1990】及其改进算法【G. Banavar 1999】。同时有许多利用计算机存储特性如 cache、主存特点设计的算法如 cache conscious algorithm【Hanson1999】, main memory algorithms【Marcos K 1999】。基于树的方法主要有【Aguilera 1999】【Gough 1995】。基于自动机的算法有【谭建龙 2003】。

4.2.2.1 计数算法

第一步：初始化

初始化即为建立索引的过程，需要建立两张映射表（索引）：一张是规则表（m_pExpr），它保存一个规则中包含哪些事件；一张事件表（m_pKey），它保存事件包含在哪些规则中。

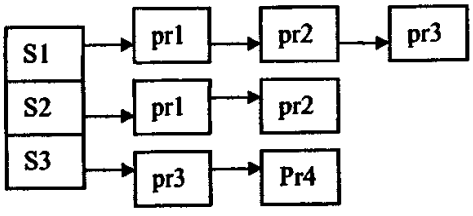
例如对于有三个规则 S1,S2,S3 和 4 个事件 pr1,pr2,pr3,pr4:

S1=pr1 and pr2 and pr3

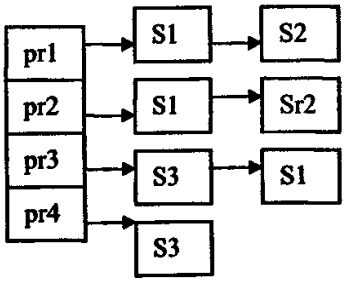
S2=pr1 and pr2

S3=pr3 and pr4

规则表如下：



事件表如下：



表m_pExpr和表m_pKey是所有网络连接都可以共享。

第二步，数据流初始化

```
typedef struct EXPRSTREAMITEM{
    int pKeyIsMatch[m_KeySize];// 事件是否是第一次出现
    int pRuleHaveMatch[m_ExprSize];//规则已经出现的事件个数
}TStreamItem;//一个数据流的临时数据结构
```

对于每个网络连接都必须存在一个 TStreamItem 的数据实例。同时必须在网络连接开始前初始化 TStreamItem 实例，使 m_KeySize 大小表的 pKeyIsMatch 初始化为0，表示现在还没有任何对应事件出现。同样 m_ExprSize 大小表的 pRuleHaveMatch 也必须初始化为 0，表示对应规则中还没有发现任何事件。

第三步，扫描数据流

当网络连接中出现一个事件 e 时，需要计算是否有任何规则被满足。如果该事件 e 首次发生，对于每个包含 e 的规则 s，pRuleHaveMatch[s]增一，如果 pRuleHaveMatch[s]等于 s 中的规则个数则表明该规则满足条件。

时间复杂度和空间复杂度

对面向网络内容分析系统的算法时间复杂度主要考虑第二、第三两个阶段的时

间复杂度。在第二阶段我们主要需要初始化临时空间 TstreamItem 的实例，所以它的时间复杂度是 O(m_KeySize + m_ExprSize)。第三阶段是处理一个数据项（事件发生）的时间复杂度。我们可以知道 counting 算法需要遍历这个事件出现的所有规则数，在最坏情况，时间复杂度是 O(m_ExprSize)，平均复杂度是（m_ExprSize/m_KeySize)。

类似的，第二阶段的空间复杂度为 O(m_KeySize + m_ExprSize)，在 C 阶段的空间复杂度为 O（1)。

计数算法存在的问题

首先，对每个网络连接，它需要创立一个 O(m_KeySize + m_ExprSize)的临时存储空间。随着网络连接数目的增加，临时存储空间也会快速地增长，对于 O（10^7）个并发的网络连接，需要的总空间是 O(m_KeySize + m_ExprSize)× O（10^7），也是比较多的。

其次，对每个网络连接，它的预处理时间即 B 阶段时间是 O(m_KeySize + m_ExprSize)。在网络内容分析的时候，每秒需要建立的网络连接个数可能接近 O（10^4）,所以每秒花在 B 阶段的总时间是 O(m_KeySize + m_ExprSize)×O（10^4），也是比较多的。

最后，计数算法在 C 阶段执行的时间是（m_ExprSize/m_KeySize)。

4.2.2.2 自动机算法

ExprAuto 算法是只支持滑动窗口 w 的关键词表达式匹配算法。一个规则在窗口中被满足，就是窗口 w 中出现了这个规则的全部事件。对于文本的关键词表达式匹配来说，就是在一段文字中出现了 w 个关键词，并且这些关键词包含了规则所要求的全部关键词。

一般来说，在网络内容分析系统中，一条规则中只包含两个关键词，窗口的大小为四。和第一章介绍的三个阶段对应，下面分为 A、B、C 三个部分来介绍 ExprAuto 算法。

第一部分 初始化

根据规则和事件构造自动机。构造共享自动机分为三个步骤：首先构造一个多叉完全树，其次增加转移指针，最后对各个状态增加这个状态满足的规则。下图是一个构造好的自动机。

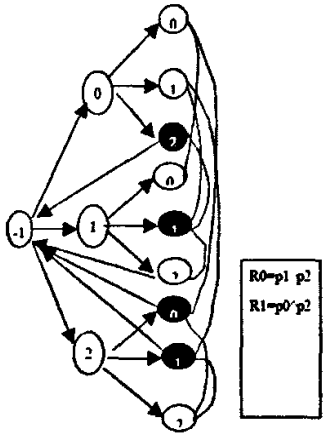


图　ExprAuto 自动机

第二部分 对每个数据流的初始化

定义下面的数据结构，保存每个数据流在匹配过程中使用的临时数据。

typedef struct TEXPRAUTUITEM{ TExprNode * m_pWork;//保存当前状态指针 }TItemAuto;//一个数据流的临时数据结构

每一个新的数据流到来的时候，我们新建一个 TitemAuto 类型的 pItem，并使 pItem->m_pWork= m_pRoot(根节点)。

第三部分扫描数据流

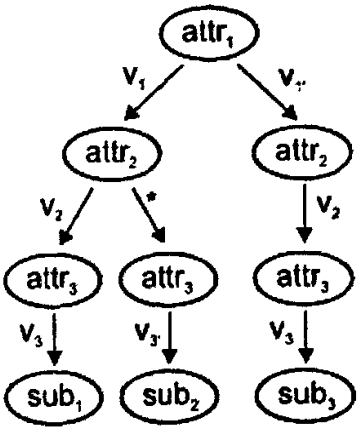
建立全部网络连接所共享的自动机和每个网络连接所使用的临时数据后，匹配算法就只需要根据当前的事件编号移动 pItem->m_pWork 指针使它等于对应的儿子指针就可以了。如果当前状态为满足状态，则报告匹配输入事件的规则。

算法的复杂度：

对比于经典的计数算法，ExprAuto 的主要缺点在于它的所有网络连接所共享的自动机的空间复杂度和时间复杂度都是 O(m_KeySize^{w})，但是在 B、C 两个阶段它的时间和空间复杂度都为 O(1)。但是在面对千万级别的网络连接和千级别的规则的应用环境中，对 比计数算法在 B 阶段的时间和空间复杂度都是 O（m_KeySize+m_ExprSize）和在 C 阶段的时间复杂度是 O（m_ExprSize/m_KeySize），ExprAuto 算法具有一定的优势。同时 ExprAuto 的 B 阶段的空间复杂性低也导致它更适合海量网络连接的应用场合。

4.2.2.3 基于树的算法

第一步根据规则和事件建立 matching tree。内部节点及从内部节点引出的边表示事件，叶子节点表示规则。下图是 matching tree 的一个例子



第二步，对每个事件流，初始化一个指针指向 matching tree 根节点。

第三步，从根节点出发，根据事件访问孩子节点，如果达到页节点表明有符合要求的规则出现。否则，表明没有符合要求的规则。

算法的时间和空间复杂度

建立 matching tree 需要 O(MN)的时间和空间(M 为事件个数，N 为规则个数)。第三步匹配过程所需时间在 M 到 MN 之间。

4.2.3

自动机算法虽然具有 O（1）的时间复杂度，但其空间复杂度为 M^w（M 为事件个数，w 为窗口大小），计数算法虽然对单个数据流比较有效，单对大量并发网络连接会消耗大量的内存。基于树的算法空间复杂度低于自动机算法，但时间复杂度与规则数目成正比，大大低于自动机算法。

4.3 利用位运算改进关键词表达式匹配算法

计算机的位运算非常适合与或逻辑运算，并且具有并行的特点，我们利用位并行理论设计了位并行的关键词表达式匹配算法，并利用 four-Russian 技术降低其时间复杂度，用分治的思想降低其空间复杂度。

4.3.1 Four-Russians

所谓 Four-Russians 技术【Arlazarov et.al 1970】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。

【Ukkonen 1985b】中根据这种思想提出一种算法，他把动态规划矩阵的一列作为一个状态，把所有可能出现的状态预先计算出来，并建立一个自动机。这样每从 T 中读入一个字符，自动机就从一个状态转移到另一个状态。如图所示。

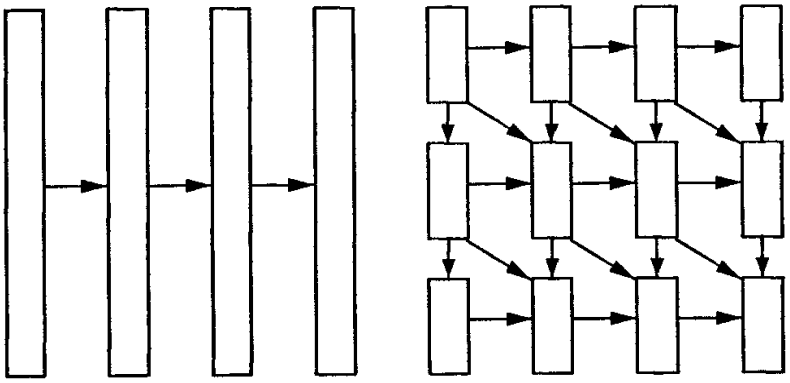


图 4.3.1.1 根据动态规划矩阵构造自动机的两种不同方式

由于动态规划矩阵的每个单元有 k+1 中可能取值，故这样建立的自动机共有$(k+1)^m$个状态。即使采用差值动态规划矩阵，状态数目仍然有3^m个。

【Myer1992】根据这种思想，采用另一种组织自动机状态的方式。把一行分成数个相等的块，每个块作为一个状态，如果一个块的大小为 r，则表示这个块共需 2r 位，所得的状态总数要远小于按列组织状态的方式。

【Masek 1980】把动态规划矩阵中大小为 t×t 的方块作为一个 t-块。相邻的 t-块重叠一行或一列。以每个 t-块左面一列、上面一行，已经 S 和 T 中对应的子串为输入，以右面一列，下面一行为输出，事先构造出所有的 t-块。计算过程中只需检索遇到的每个 t-块的输出，作为后面 t-块的输入。若选择 t=logn，则算法的时间复杂度为 $O(n^2/logn)$。

4.3.2 预处理

首先约定 M 表示事件个数，N 表示规则个数，以 WORD 表示一个机器字，L 表示 WORD 的字长。为了叙述方便，首先假设 M 和 N 都不大于 32（x86 机器字长）。

第一步，将事件从 0 到 M-1 编号，则 WORD 可以看作是一个长度为 L 的位串。WORD[i]=1 表示第 i 个事件出现，WORD[i]=0 表示第 i 个事件没出现。对每一个规则用一个字 RULEWD 来编码, RULEWD[i]=0 表示该规则不包含第 i 个事件，RULEWD[i]=1 表示该规则包含第 i 个事件。例如：有 8 个事件 exp0、exp1、exp2、exp3、exp4、exp5、exp6、exp7。规则一 RULE1=exp0 & exp3 & exp4, 则规则一的编码 RULEWD1 为 00011001。

第二步建立函数 whichrule。第一种方案可将 whichrule 作为一个数组，WORD RULES=whichrule[exp_WORD]。对于事件序列 exp_WORD 通过查 whichrule 表可得到该事件序列满足的规则序列。RULES[i]=1 表示第 i 个规则符合。此种方法虽然理论上可行，但通常 WORD 字长为 32，故 whichrule 数组的大小为 16G。故需进一步优化该数组使内存消耗量降低到可以接受的程度。

首先将 WORD 分成左右两部分，每部分各有 L/2 位。建立两个数组 whichrule_left[1<<L/2]、whichrule_right[1<<L/2].数组 whichrule_left 各元素的值由规则编

码 RULEWD 的左半部分确定，whichrule_right 各元素的值由规则编码 RULEWD 右半部分确定。Whichrule(i)=whichrule_left[i] & whichrule_right[i];

数组 whichrule_left 和 whichrule_right 的建立如下：

Setup_whichrule(RULEWD *RULE, WORD * *whichrule_left, WORD ** whichrule_right){

/* 输入：RULE 表示规则数组，数组每个元素为一个 RULEWD 字，其意义如前所述。

输出：数组 whichrule_left 和 whichrule_right。

算法用到的宏和函数：

宏：L 表示 WORD 字长。

函数：left(WORD) 取出 WORD 的左半部分；

Right(WORD) 取出 WORD 的右半部分。

*/

```
*whichrule_left=(WORD*)malloc(sizeof(WORD)*(1<<L/2));
*whichrule_right=(WORD*)malloc(sizeof(WORD)*(1<<L/2));
int i;
for(i=0;i<1<<L/2;i++)
{
    WORD leftwd＝0，rightwd＝0;
    for(int j=0;j< rule_size;j++)
    {
    if ((left(i) & left(RULE[j])) = = left(RULE[j]))
        {
            leftwd |= (1<< j);
        }
      if ((right(i) & right(RULE[j])) = = right(RULE[j]))
       {
            rightwd |= (1<< j);
        }
    }
    (*whichrule_left)[i]=leftwd;
    (*whichrule_right)[i]=rightwd;
}

}
```

函数 whichrule 如下：

```
WORD whichrule( exp_WORD)
{
/* 输入：exp_WORD 是用一个 WORD 表示的事件序列。
输出：RULES 表示满足输入事件序列的规则
*/
WORD RULES= whichrule_left[exp_WORD] & whichrule_right[exp_WORD];
Return RULES;
}
```

4.3.3 对每个数据流的初始化

对每一个数据流分配一个 WORD expswd=0，用来保存事件序列。

4.3.4 匹配过程

扫描数据流的过程中，每当有事件发生，用 insertexp(i)来判断是否有符合要求的规则出现。

```
WORD insertexp( int expi)
{
/* 输入：当前发生的事件的序号。
    输出：－1：该事件已经发生。
            0：  未找到符合要求的规则。
          >0:   由发生匹配的规则号编码得到的字。定义在 4.3.2 节
    if((expswd & (1<< expi))!=0) return -1;
    expswd |=(1<<expi);
    Rules=whichrule(expswd);
Return Rules;
}
```

4.3.5 算法分析

4.3.5.1 算法的时间复杂度

预处理阶段，算法主要建立两个表 which_right 和 which_left。每个表长度为 4×64kB，每个元素都要与每个规则进行比较，故共需 2×64×1024×m_RULESIZE 次比较。通常预处理阶段所需时间较短，可以忽略不计。

在匹配阶段，对每个发生的时间只需查两次表就可得到结果，故时间复杂度为 O(1).

算法时间消耗主要集中在匹配阶段，故可以认为算法总的时间负责度为 O(1)。

算法的空间复杂度

算法所需空间可分为两部分：所有数据流共享的数据，各个数据流的私有数据。

共享数据是算法在预处理阶段建立的两个表：which_right 和 which_left。每个表的大小为（1<<L/2）*4B。在本例中，WORD 字长 32bits，每个表的大小为 256kB，共需内存 512kB。实际上，其内存消耗量可根据需要动态调整，例如将 WORD 分为 3 个部分，共享数据需内存 4kB，但同时匹配过程中要以多一次查表操作为代价。

当事件的数目小于 L 时，各个数据流的私有数据为 WORD 型数据 expswd。

4.3.5.2 算法的可扩展性

算法受机器字长的限制，其扩展性受到一定限制。当事件的数目大于 L 时，不能直接利用该算法，有两种方法扩展该算法。

第一种方法，利用 SIMD 指令集，例如 SSE2 指令集，其数据宽度为 128bit，可在一定程度上扩展该算法。

第二种，将规则和事件作为图的顶点，如果规则 RULEi 中包含事件 EXPj，则在 RULEi 和 EXPj 之间建立一条边。此图是一个二分图。将此图按连通分量划分为几个部分，每个部分独立编号并建立各自的 which_right 和 which_left 表。

4.4 小结

本节总结了关键词表达式匹配算法的发展，并利用位并行、four-russian 和分治等优化方法设计了新的关键词表达式匹配算法，其时间复杂度为 O(1)，对每个数据流而言其空间复杂度也是 O(1)。在本例中，WORD 分为左右两部分，其共享数据需内存 512kB。

第五章 结束语

5.1 工作总结

串匹配算法作为计算机科学的一类基础算法，广泛应用在众多领域。尤其近年来，生物信息学的迅速发展、基因数据和网络数据的飞速增长，串匹配算法的重要作用日益突出。

本文从计算机体系结构的角度出发，分析、优化几类应用较为广泛的串匹配算法。

(1) 对于 Smith-Waterman 算法，通过分解，降低数据依赖性，充分开发了其数据并行性，并基于 SSE2 指令集实现，和目前该算法的最优实现相比，提高到 1.5 倍；对于 Zuker 算法，设计辅助矩阵，有效组织数据，对于较大的数据规模，获得了接近理论值的加速比。

(2) 对于 shift-or 算法，通过组合 NFA 的状态，建立Σ_{short}上的 NFA，状态转换单位由 char 型提高为 short 型整数，显著降低了自动机状态转换次数，性能提高 30%以上。

(3) 对于关键词表达式匹配算法，利用 four-russian 和分治等优化方法设计了新的关键词表达式匹配算法，并基于位并行实现，可提高计算效率、降低内存开销。

5.2 研究展望

不同的串匹配算法适应于不同的领域，本文仅对几个主要算法进行了优化，基于相关领域的发展趋势，未来的工作可以将在以下几方面展开：

（1） 基于体系结构的算法由于和体系结构的紧密结合，天然削弱了其代码的可移植性。近年来，自适应优化方法获得了巨大成功，如 ATLAS、FFTW 等。采用自适应优化思想，优化设计串匹配高性能算法，将是一个很有意义的挑战。

（2） 非数值计算类算法种类繁多，单单是串匹配算法，就多达几十种，因此，在非数值计算领域，至今没有类似数值计算领域的 BLAS、LAPACK 等基本数学库。是否可以抽象出类似的基本运算库？如何抽象？这也是一个悬而未决的问题。

（3） 非数值计算类算法和数值计算类算法的计算行为特征（包括访存行为、I/O 行为等）有什么共同点？有什么不同点？计算机体系结构应该相应的有什么样的优化？这些问题，同样值得深入研究。

参考文献

【AC 1975】Aho, A. V., and M. J. Corasick, "Efficient string matching: an aid to bibliographic search,"Communications of the ACM 18 (June 1975), pp. 333-340.

【Aguilera 1999】M. Aguilera, R. Strom, D. Sturman, M. Astley and T. Chandra. Matching events in a content-based subscription system. In Eighteen ACM Symposiumon Principles of Distributed Computing (PODC'99), 1999

【Allauzen 1999】Cyril Allauzen, Mathieu Raffinot Oracle des facteurs d' un ensemble de mots Technical report 99-11, Institut Gaspard Monge Universite Marne la Valee, 1999

【Allauzen 2001】Cyril Allauzen, Maxime Crochemore, Mathieu Raffinot Efficient Experimental String Matching by Weak Factor Recognition in Proceedings of 12th conference on Combinatorial Pattern Matching, 2001

【Alpern 1995】Alpern,B., Carter,L. and Gatlin,K.S. (1995) Microparallelism and high performance protein matching. In Proceedings of the 1995 ACM/IEEE Supercomputing Conference: San Diego, California, Dec 3–8, 1995.

【AltiVec】http://www.SIMDtech.org/home

【Aluru 1998】S. Aluru, N. Futamura and K. Mehrotra. Parallel Biological Sequence Comparison using Prefix Computations. Proceedings of the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, 1998

【AMD1】3DNow!™Technology Manual

【AMD2】AMD Extensions to the 3DNow!™ and MMX™ Instruction Sets Manual

【AMD2】Software Optimization Guide for AMD Athlon™ 64 and AMD Opteron™ Processors

【Aneesh 2001】Aneesh Aggarwal Abdel-Hameed A.Badawy,Donald Yeung,Chau-Wen Tseng. Evaluating the impact of memory system performance on software prefetching and locality optimizations. In Proceedings of the 2001 International Conference on Supercomputing. Sorrento,Napoli,Italy,2001:486-500.

【Arlazarov et.al 1970】V.L.Arlazarov,E.A.Dinic,M.A.Kronrod,and I.A.Faradzev. Oneconomic construction of the transitive closure of a directed graph.Dokl.Acad.Nauk SSSR,194:487-88,1970

【Assayag 2004】G. Assayag, S. Dubnov Using Factor Oracles for Machine Improvisation Soft Computing, 2004

【BAEZA 1989】BAEZA-YATES, R. 1989. Efficient Text Searching. Ph.D. thesis, Dept. of Computer Science, University of Waterloo. Also as Res. Rep. CS-89-17.

【Baeza 1989】BAEZA-YATES, R. 1989. Efficient Text Searching. Ph.D. thesis, Dept. of Computer Science, University of Waterloo. Also as Res. Rep. CS-89-17.

【Baeza-Yates and Navarro 1999】BAEZA-YATES, R. AND NAVARRO, G. 1999. Faster approximate string matching. Algorithmica 23, 2, 127–158. Preliminary versions in Proceedings of CPM '96 (LNCS, vol. 1075, 1996) and in Proceedings of WSP'96, Carleton Univ. Press, 1996.

【Bouknight 1972】Bouknight, W.J., et al., *The ILLIAC-IV System.* Proc. IEEE, April 1972, pp. 369-388. (reprinted in CSPE)

【BOYER 1977】BOYER R.S., MOORE J.S., 1977, A fast string searching algorithm. Communications of the ACM. 20:762-772

【Cherng 2005】Cary Cherng and Richard E. Ladner, cache efficient simple dynamic programming, DMTCS proc. AD, 2005, 49–58

【Cleophas 2003】Loek Cleophas, Gerard Zwaan, Bruce Watson Constructing Factor Oracles in Proceedings of the Prague Stringology Conference 2003, 2003

【CROCHEMORE 1997】CROCHEMORE, M., HANCART, C., 1997. Automata for Matching Patterns, in Handbook of Formal Languages, Volume 2, Linear Modeling: Background and Application, G. Rozenberg and A. Salomaa ed., Chapter 9, pp 399-462, Springer-Verlag, Berlin

【Curt Schimmel】 Curt Schimmel 著，张辉译，《现代体系结构上的 UNIX 系统－内核程序员的 SMP 和 Caching 技术》，人民邮电出版社，北京.2003.

【F. Fabret 2001】 F. Fabret, A. Jacobsen, F. Llirbat, J. Pereira, K. Ross and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In SIGMOD' 2001, 2001.

【Fabret 2001】F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In ACM SIGMOD 2001, Santa Barbara, CA, May 2001

【Fabret 2001】Françoise Fabret, H. Arno Jacobsen, François Llirbat, Joao Pereira, Kenneth A. Ross, Dennis Shasha filtering algorithms and implementation for very fast publish/subscribe systems ,ACM SIGMOD Record , Proceedings of the 2001 ACM SIGMOD international conference on Management of data SIGMOD '01, Volume 30 Issue 2 May 2001

【G. Banavar 1999】G. Banavar, T. Chandra and B. Mukherjee. An efficient multicast protocol for content-based publishsubscribe systems. In Proc. of the 19th International Conference on Distributed Computing Systems, 1999.

【Geo 2002】Geo Pike and Paul N.Hilnger.Better Tiling and Array Contraction for Compiling Scientic Programs. In Proceedings of the IEEE/ACM conference on Supercomputing. Baltimore,Maryland,USA,2002:1-12

【Godson】李祖松,齐子初. Godson-2 处理器运算功能部件指令集. 龙芯 CPU 研制组技术报告，2003 年 10 月

【Gotoh 1982】O. Gotoh. An improved algorithm for matching biological sequences. Journal of Molecular Biology, 1982, pp.705-708

【Gough 1995】K. Gough and G. Smith. Efficient recognition of events in distributed systems. In Proc. of the ACSC 18, 1995.

【Green 1993】Green,P. (1993) SWAT. http://www.genome.washington.edu/uwgc/analysistools/swat.htm

【Hanson 1990】E. Hanson, M. Chaabouni, C. Kim and Y. Wang. A predicate matching algorithm for database rule systems.In SIGMOD'90, 1990.

【Hanson 1999】E. Hanson and J. Widom. An overview of production rules in database systems. The Knowledge Engineering Review, vol. 8 no. 2, June 1993 1999.

【Hanson 1999】E.Hanson, C.Carnes, L.Huang, M.Konyala, L.noronha, S.parasarathy, J.Park,

and A.vernon.Scalable trigger processing.In Proceedings of the Int.Conf. on Data Engineering,1999.

【HORSPOOL 1980】 HORSPOOL R.N., 1980, Practical fast searching in strings, *Software - Practice & Experience*, 10(6):501-506

【Intel c】 Intel® C++ Compiler User's Guide

【Intel v1】 Intel: IA-32 Intel Architechture Software Developer's Manual, Volume 1: Basic Architecture

【Intel v2】 Intel: IA-32 Intel Architechture Software Developer's Manual, Volume 2: Instruction Set Reference

【Intel v3】 Intel: IA-32 Intel Architechture Software Developer's Manual, Volume 3: System Programming Guide

【J. Pereira 2000】 J. Pereira, F. Fabret, F. Llirbat and D. Shasha. Efficient matching for web-based publish/subscribe systems.In Proc. of the Fifth IFCIS International Conference on Cooperative Information Systems (CoopIS'2000), Eilat, Israel, September 2000.

【KNUTH 1977】 KNUTH D.E., MORRIS (Jr) J.H., PRATT V.R., 1977, Fast pattern matching in strings, *SIAM Journal on Computing* 6(1):323-350

【Kumar 1994】 Kumar s,Spafford E H .An Application of Pattern Matching in Intrusion Detection,Technical Report CSD-7R-94-013,Department of Computer Science ,Purdue University ,1994

【Lee 1995】 R. Lee. Accelerating Multimedia with Enhanced Microprocessors. IEEE Micro,1995,15(2):22-32.

【Levebvre 2000】 Arnaud Levebvre, Thierry Lecroq Computing repeated factors with a factor oracle in Proceedings of 11th Australian Workshop on Combinatorial Algorithms, 2000

【Marcos K 1999】 Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, Tushar D. Chandra Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing May 1999

【Martin 2000】 Martin Tompa, Biological Sequence Analysis, Technical Report, Department of computer Science and Engineering University of Washington, 2000

【Masek 1980】 W.J.Masek and M.S.Paterson. A faster algorithm for computing string edit distances. J.Comp. Sys.Sci,20:18-31,1980.

【Myers 1988】 E.Myers and W. Miller. Optimal Alignments in linear space. Computer Applications in the Biosciences, 4:11-17,1988

【Myers 1992】 Gene Myers A Four Russians algorithm for regular expression pattern matching Journal of the ACM (JACM) Volume 39 , Issue 2 (April 1992) Pages: 432 - 448 Year of Publication: 1992

【Myers 1999】 MYERS, G. 1999. A fast bit-vector algorithm for approximate string matching based on dynamic progamming. J. ACM 46, 3, 395–415. Earlier version in Proceedings ofCPM'98 (LNCS, vol. 1448).

【Nathan 2004】 N. Tuck, T. Sherwood, B. Calder, and G. Varghese. Deterministic Memory-Efficient String Matching Algorithms for Intrusion Detection. In Proceedings of IEEE Infocom, Hong Kong, March 2004.

【Navarro 2001】 Gonzalo Navarro,A Guided Tour to Approximate String Matching, ACM

Computing Surveys, Vol. 33, No. 1, March 2001.

【Needleman andWunsch 1970a】NEEDLEMAN, S. AND WUNSCH, C. 1970. A general method applicable to the search for similarities in the amino acid sequences of two proteins. J. Mol. Biol. 48, 444–453.

【Needleman andWunsch 1970b】S.B. Needleman and C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 48:443-453,1970

【Nishimura 2001】T. Nishimura, Shuichi Fukamachi, Takeshi Shinohara: Speed-up of Aho-Corasick Pattern Matching Machines by Rearranging States. SPIRE 2001: 175-185

【Rognes 2000】Torbjørn Rognes and Erling Seeberg，Six-fold speed-up of Smith–Waterman sequence database searches using parallel processing on common microprocessors. Vol. 16 no. 8 2000 Pages 699–706

【Ron 2001】Ron Shamir, Algorithms for Molecular Biology, Technical Report, Tel Aviv University Department of Computer Science, 2001

【Sellers 1980】SELLERS, P. 1980. The theory and computation of evolutionary distances: pattern recognition. J. Algor. 1, 359–373.

【SMITH 1991】SMITH P.D., 1991, Experiments with a very fast substring search algorithm, Software - Practice & Experience 21(10):1065-1074

【Smith-Waterman 1981】T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. Journal of Molecular Biology, 147(1):195-197.1981

【Stephen】Stephen R. van den Berg，GNU C Library ftp://ftp.gnu.org/pug/glibc.

【SUNDAY 1990】SUNDAY D.M., 1990, A very fast substring search algorithm, Communications of the ACM . 33(8):132-142

【UKKONEN 1985】UKKONEN, E. Finding approximate patternsin strings. J. Algor. 6, 132–137. 1985.

【Vintsyuk 1968】VINTSYUK, T. 1968. Speech discrimination by dynamic programming. Cybernetics 4, 52–58.

【Waldvogel 1997】Waldvogel et. al. Scalable High Speed IP Routing Lookups . In Computer Communication Review, Vol 27, #4, October 1997.

【Walid Rjaibi 2002】Walid Rjaibi, Klaus R. Dittrich, Dieter Jaepel September 2002 Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research

【WM 1994】 Sun Wu，“A FAST ALGORITHM FOR MULTI-PATTERN SEARCHING”,Technical Report Department of Computer Science Chung-Cheng University Chia-Yi, Taiwan

【Wozniak 1997】Wozniak,A. (1997) Using video-oriented instructions to speed up sequence comparison. Comput. Appl. Biosci., 13, 145–150.

【WU and MANBER 1992】WU, S., MANBER, U., 1992, Fast text searching allowing errors, *Commun. ACM.* 35(10):83-91.

【Wulf 1995】W.Wulf and S. McKee, Hitting the memory wall:Implications of the obvious. Computer Architecture News,1995,23(1)：20－14

【车永刚 2004】车永刚，Performance Analysis and Optimization Techniques for Scientific Programs.博士论文，国防科学技术大学，长沙，2004.

【李明 1997】李明，唐志敏，一种新的 cache 优化方法－部分 cache 局部性. 计算机学报，1997，27(1):1-7.
【刘方爱 2001】刘方爱，高效并行计算系统中的计算模型与通信网络，博士论文，中科院计算所 ，2001
【谭建龙 2003】谭建龙，串匹配算法及其在网络内容分析中的应用 博士论文，中科院计算所，2003.
【网络信息 2005】中国互联网络发展状况统计报告，中国互联网络信息中心，2005.

附录

利用 SSE2 指令集编程

Intel 为 SSE2 指令集提供了两种编程接口【Intel v1,2,3】【Intel c】：汇编指令集和 c 语言指令集。利用嵌入式汇编可以直接使用 SSE2 汇编指令集操作 SSE2 寄存器。如果要使用 C 语言指令集必须用 Intel 的 c++编译器进行编译。

下面用一个简单的例子介绍如何利用 SSE2 指令集编程。

算法：实现两个向量对应元素相加，结果放入第三个向量中。假设向量的长度是 4 的倍数。

标准 c 实现如下：

```
void add(int *s1,int *s2,int *s3,int len)
{
    int i=0;
    for(i=0;i<len;i++)
        s3[i]=s1[i]+s2[i];
}
```

使用 Intel 平台下的嵌入式汇编如下所示

```
void add(int *s1,int *s2,int *s3,len)
{
    __asm(
        next:
          movdqa (%%ess),%%xmm1;
          movdqa (%%ebx),%%xmm2;
          paddd    %%xmms1,%%xmm2;
          movdqa   %%xmm2,(%%eds);
          addl      $16,%%esi;
          addl      $16,%%edi;
          addl      $16,%%ebx;
          subl      $16,%%ecx;
          cmp         %%ecx,$16;
          jge      next;
    ":
    :"S"(s1),"b"(s2),"D"(s3),"c"(len)
        );
}
```

在 Opteron 平台上，所有的寄存器都是 64 位寄存器，各寄存器的名字为 rax,rbx,rcx,rdx,rsi,rdi 等等。只需将上述代码中的寄存器名字(Intel 平台)换作 r 开头的名字即可。SSE2 指令集寄存器的名字保持不变。例如：movdqa (%%rss), %%xmm1;

使用 c 语言接口

```
void add(int *s1,int *s2,int *s3,len)
    {
        int len4=len/4;
        __m128i *ps1=(__m128i*)s1;
        __m128i *ps2=(__m128i*)s2;
        __m128i *ps3=(__m128i*)s3;
        for(i=0;i<len4;i++)
        {
            *ps3=_mm_add_epi32(*ps1,*ps2);
            ps1++;
            ps2++;
            ps3++;
        }
    }
```

__128i 表示 128 位的整数。paddd 是汇编指令，_mm_add_epi32 都表示两个__128i 类型的数相加。

致 谢

三年时间倏然而逝，回首三年，计算所的各位老师和同学对我的指导和帮助让我终生难忘，感谢三年来在学习上指导帮助我的老师们，感谢三年来与我朝夕相处的同学们。

首先要感谢我的导师冯圣中老师。感谢冯老师三年来学习上对我的悉心教导，生活中对我的谆谆教诲。三年来我前进中的每一步都凝结着冯老师的汗水。感谢冯老师为我们提供一个自由、宽松、团结、奋进的学习和生活环境。冯老师渊博的学识、严谨的态度将对我影响至深。我永远忘不了冯老师一字一字帮我修改论文的情景。冯老师忘我的工作作风将永远激励我为理想拼搏。三年来冯老师为我们付出的心血我当铭记在心。

感谢我的师兄谭光明、李玉岗，师姐徐琳。他们执著的科研精神、乐观上进的生活态度，深深感染着我。感谢他们在学习上对我不厌其烦的帮助。和他们的讨论和合作大大拓宽了我的思维，为论文的顺利完成打下了基础，让我受益匪浅。

感谢邱振戈老师、石锦彩老师、张法博士，他们如兄长般对我的帮助和支持让我倍受感动。他们渊博的学识永远促我奋进。

感谢张庆丹、彭柳、丁玉垒。他们热情诚恳的帮助让我时时感受到这个集体里浓浓的学术气氛，感受到相互帮助、共同进步的温暖。

感谢徐铸、王仁重、尹华祥，他们给我的工作和生活提供了诸多的帮助。

感谢研究生部的宋老师、张老师在学习和生活中给予我的无微不至的关怀和帮助。

感谢室友谷晓铭对我的理解和支持。

最后，再一次真诚地感谢所有帮助过我的人们。真心地祝福他们！

作者简历

姓名：戴正华　　性别：男　　出生日期：1980.12.25　　籍贯：山东兖州

2003.10 --2006.7　　中科院计算所计算机系统结构专业硕士研究生
1999.9 --2003.7　　山东师范大学计算机科学与技术专业本科生

【攻读硕士学位期间发表的论文】

戴正华，张庆丹，徐琳，谭光明，冯圣中，基于 SSE2 的 Smith-Waterman 算法，《计算机工程与应用》，2006 年 6 月

【攻读硕士学位期间参加的科研项目】

[1] 2004.4-2004.7 参与 4000A linpack 测试

[2] 2004.7-2004.12 国家自然科学基金项目 基于 SIMD 指令集的高性能序列联配算法研究。

[3]2005.1-2005.6 参与曙光 4000H 研究。

面向体系结构的串匹配算法优化研究

作者： 戴正华
学位授予单位： 中国科学院计算技术研究所