计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20131548
52(5):995-1004,2015

基于指令级并行的倒排索引压缩算法

闫宏飞[1] 张旭东[1] 单栋栋[2] 毛先领[3] 赵 鑫[1]
[1](北京大学网络与信息系统研究所 北京 100871)
[2](淘宝(中国)软件有限公司 杭州 312000)
[3](北京理工大学 北京 100081)
(yhf@net.pku.edu.cn)

SIMD-Based Inverted Index Compression Algorithms

Yan Hongfei[1], Zhang Xudong[1], Shan Dongdong[2], Mao Xianling[3], and Zhao Xin[1]
[1](*Institute of Network Computing and Information Systems, Peking University, Beijing* 100871)
[2](*Taobao (China) Software Co., Ltd, Hangzhou* 312000)
[3](*Beijing Institute of Technology, Beijing* 100081)

Abstract The rapid growth of text information has brought about new challenges to traditional information retrieval. In large search engines, indexing is required to help users acquire important data they need, and techniques of inverted index have great influence on the efficiency of query processing in such systems. The data in inverted index is stored in the form of arrays of integers, and techniques of compression are required to reduce the cost of storing such data in disks and memory, as well as to boost the hit rate of CPU cache and speed up transferring data. Therefore, it is necessary to choose a highly efficient compression algorithm to process query effectively. In this paper, we propose two instruction-level-parallelized algorithms, i.e. SIMD-PB and SIMD-PFD, which improve two competitive compression algorithms respectively, i.e. PackedBinary and PForDelta, and exploit SIMD instructions to accelerate the Pack and Unpack procedure in the algorithms. Experiments based on public datasets of GOV2 and ClueWeb09B show that our novel algorithms have good performance on encoding and decoding speed without impairing the compression ratio, and outperform the former fastest inverted list compression algorithms by at most 17%, with respect to decompression speed. Furthermore, experiments indicate that our novel algorithms have better performance on longer posting list and larger block size w.r.t. decoding speed.

Key words single instruction multiple data(SIMD); inverted index; compression; integer encoding; information retrieval

摘 要 文本信息数量的快速增长给传统的信息检索技术带来了新的挑战.搜索引擎通常使用倒排索引来高效地处理查询.为了减少存储开销和加快访问速度,倒排索引通常被压缩存储.因此,如何选择一个高性能的压缩算法对高效查询处理是非常有必要的.在已有倒排链压缩算法 PackedBinary 和 PForDelta 的基础上,利用 CPU 的超标量特性和 SIMD 向量指令集,将其压缩和解压缩中的关键步骤并行化,提出了 2 种指令级并行压缩算法 SIMD-PB 和 SIMD-PFD.基于 GOV2 和 ClueWeb09B 两个公开数据集的实验表明,SIMD-PB 和 SIMD-PFD 算法在压缩率不变的情况下,压缩和解压缩速度比现有的压缩算法均有非常明显的提升.其中解压缩速度比起目前最好的倒排链压缩算法,最高能提升 17%.

收稿日期:2013-11-29;**修回日期**:2014-04-23
基金项目:国家"九七三"重点基础研究发展计划基金项目(2014CB340400);国家自然科学基金项目(61272340);江苏未来网络创新研究院项目-云服务数字资源搜索(BY2013095-4-02)

此外,实验表明算法在较长的倒排链、较大的压缩块单位上有更好的解压缩性能.

关键词 单指令多数据流;倒排索引;压缩;整数编码;信息检索

中图法分类号 TP301.6

在网络时代,文本信息数量的快速增长给传统的信息检索技术带来了新的挑战.搜索引擎作为网络时代的信息检索工具,如今已成为用户获取网络信息的主要途径之一,其核心的数据结构是倒排索引.倒排索引由词典和倒排表2部分组成.倒排表包含若干个倒排链,每个倒排链包含一组倒排记录.对于文档 m 和词项 t,其对应的倒排记录一般包含以下信息:$\langle DocID, TF_t, [pos_1, pos_2, \cdots, pos_{TF}]\rangle$.其中,$DocID$ 是文档 m 的编号;TF_t 是词 t 在文档 m 中出现的词频;pos_i 指词 t 出现在文档 m 中的位置,$i=1,2,\cdots,TF_t$.这些信息都是由一系列非负整数组成.通常,倒排链中的各个倒排记录是按照文档号($DocID$)递增排序的.在存储倒排索引之前,通常会对倒排链中原始整数序列做转换,使得新序列的整体平均值变小,从而提升潜在的压缩率.d-gap操作[1]是常用序列转换操作之一,它适用于处理递增序列:对于一个严格递增的正整数序列 $d_1, d_2, \cdots, d_n$,d-gap操作对每个整数使用 $d_i = d_i - d_{i-1} - 1$ ($i>0, d_0=0$)进行替换.

倒排表是倒排索引存储空间的主要部分.随着索引文档量的增加,索引中倒排表占用的存储空间也快速增加,这将影响到查询时倒排链的读取速度,进而影响查询性能.因此,人们通常利用倒排索引压缩的技术来提升查询处理速度,研究表明:倒排索引压缩技术除了能够减少倒排索引占用的磁盘空间外,还能减少内存占用,提高CPU缓存的命中率,进而提升数据查询处理速度和查询吞吐率.倒排索引压缩问题是指:给定 n 个以固定位数(32 b或64 b)表示的非负整数,设计一种算法将它们转变成一个二进制编码序列,使得转换后所有整数的编码总长度比输入时的二进制编码总长度短.建立索引时,通常会使用压缩技术将倒排链压缩,而在处理查询时,首先会将对应倒排链解压缩,再进行打分排序操作.

目前的倒排链压缩/解压技术主要是使用串行模式.为了充分利用机器的并行潜力,提高倒排链的访问和解压速度,越来越多的研究开始采用并行技术.并行技术按照不同粒度可分为:机器级并行、进程级并行、线程级并行、指令级并行等,前3种并行技术在查询处理时较为常见,如多机分布式部署索引查询时每个进程/线程处理一个查询请求或一个倒排链等,但它们并未充分利用硬件的并行潜力.我们注意到,CPU的SSE向量指令集使用了更大的128 b寄存器进行工作,有一次解压多个32 b整数的能力.因此,本文主要研究更细粒度的指令集并行的压缩算法.PackedBinary[2]和PForDelta[3-4]是2种使用固定比特长度对整数进行压缩的串行算法,它们在压缩速度、解压缩速度和压缩率上都有不错的性能.PackedBinary和PForDelta有一个共同的核心部分:压缩时的Pack过程和解压缩时的Unpack过程.根据其整数存储布局的特性,我们采用单指令多数据流(single instruction multiple data,SIMD)向量指令改进PackedBinary算法和PForDelta算法的Pack和Unpack过程,提出了2种新倒排链压缩算法:SIMD-PB和SIMD-PFD.相比于其串行版本,SIMD-PB算法和SIMD-PFD算法在压缩率没有任何损失的情况下,显著提高了压缩和解压速度,在解压缩速度上相比于最好的算法性能可以提高至多17%.本文的主要贡献如下:

1) 根据SIMD指令的特性,重新设计了PackedBinary算法和PForDelta算法存储数据的布局,提出基于指令级并行的压缩算法——SIMD-PB和SIMD-PFD.

2) 在公开的数据集GOV2和Clueweb09B上,评测已有压缩算法和我们提出的算法的性能.实验表明我们新提出的算法在压缩和解压缩速度要优于当前最快的算法.

1 相关工作

目前倒排索引的压缩算法主要可以分成以下4类:1)位对齐压缩算法.该类算法对每个压缩的整数用若干位表示,代表算法有Elias Gamma[5],Golomb[4],Rice[6]等.2)字节对齐压缩算法.该类算法对每个压缩整数用若干个字节表示,代表算法有Variable Byte[7],Group Variable Byte[8]等.3)字对齐压缩算法.该类算法在一个字中压缩多个整数,代表算法有Simple[9-10]系列.4)对固定位长度的数字进

行压缩的算法,如 PackedBinary,PForDelta[3-4,11-12],AFOR[13],VSEncoding[14]等.下面介绍与我们的算法最相关的 PackedBinary 和 PForDelta 算法,以及当前解压缩速度最快的 SIMD-G8IU 算法[15].

PackedBinary 是一种固定二进制位数的编码方式.该算法每次编码一组整数,选出组内最大整数的有效位宽(表示一个整数所需位个数,下同)w,再将所有整数都用 w 个位来编码,该过程称为 Pack.相应地,使用 w 个位编码后的一组整数序列解码成 32 b 整数序列的过程称为 Unpack.

PForDelta:为解决 PackedBinary 算法中出现异常数时压缩率不佳的情况,Zukowski 等人[4]提出了一种固定二进制位数的编码方式 PForDelta(PFD).该算法先选出一个位宽值 w,使得所有整数中的绝大部分都可以用 w 个位表示.这些可用 w 个位表示的整数称为"正常"整数,暂时存储在一个正常数数组中.对于那些不能用 w 个位表示的大整数,将它们储存在一个异常数组中,同时在正常数组中记下这个异常数距离上一个异常数的偏移值.如果偏移值也不能用 w 个位表示了,算法会在 2 个异常数之间将一个正常值强制作为一个异常值插入异常数组中.该算法将尝试对 w 的每种取值进行压缩,选择压缩率最好时的 w 值.算法对正常数组用 PackedBinary 算法的 Pack 过程进行编码;相应地,使用 Unpack 过程进行解码.Zukowski 等人[4]没有对异常数进行压缩,而 Zhang 等人[11]根据最大异常数的值使用 8 b,16 b,32 b 的空间来存储异常数,进一步提升了 PFD 算法的压缩率.我们提出的算法 SIMD-PFD 采用了此思想.

分组变长字节编码[8](group variable byte,GVB)对每个整数采用变长字节的编码方式,但将若干控制位集中放在一个字节(以下称为控制字节)中,一次性地对多个整数进行编码/解码.而控制字节可用一进制编码或者二进制编码表示,因此该算法有 3 个变种:GVB-Binary(GB),G8IU 和 G8CU.Stepanov 等人[15]使用 SIMD 指令对 GVB 算法作了改进,提出了并行算法 SIMD-GVB(其对应的 3 个变种称作 SIMD-GB,SIMD-G8IU,SIMD-G8CU).它们用到了 Intel 的 SSSE3 向量指令集中的 PSHUFB 指令,这是一种并行字节置换指令,通过控制字节的取值,每次可以并行解压出一组整数.实验表明 SIMD-GVB 算法的解压速度可以达到 GVB 算法的 2 倍以上,其中解压缩速度最快的是 SIMD-G8IU 算法.

目前,国内对于倒排索引压缩的研究较为有限.文献[16]在 Simple-9 算法[9-10]的基础上提出了适应于 64 b 体系结构的 SimpleX64 系列算法,该算法在 64 b 系统上整体性能相比 Simple-9/Simple-16 算法有一定提升.文献[17]对 Golomb,Elias Gamma,Variable Byte 等 5 种倒排索引压缩技术进行分析和研究,对比了各个算法的压缩比、压缩与解压缩的时间以及对文件读取和查询所花费的时间.文献[18]将倒排索引压缩技术应用于 RDBMS 全文检索中.

2 并行压缩算法:SIMD-PB 和 SIMD-PFD

本节中,首先将介绍在我们提出的指令级并行的倒排链解压算法 SIMD-PB 和 SIMD-PFD 中用到的 SIMD 向量指令集;然后阐述 SIMD-PB 算法和 SIMD-PFD 算法压缩和解压的关键过程——Pack 和 Unpack 的串行和并行实现方法;最后,我们介绍并行 Pack 和 Unpack 过程的改进方案,并讨论这 2 个过程在其他压缩算法上的通用性.

2.1 SIMD 向量指令集

我们提出的压缩算法利用了 Intel CPU 的 SSE[19](包含 SSE2,SSE3,SSSE3,SSE4 等)向量指令集的特点(最新的 AMD CPU 也将完全支持上述指令集).SSE 指令集一般会使用一种 128 b 寄存器(称为 XMM 寄存器)来存放操作数,因此每次 SIMD 操作可以并行处理 4 个 32 b 整数.近几年较新的 64 b CPU 中都具有 16 个 XMM 寄存器.

我们在算法中用到的向量指令类型介绍如下:

1) 逻辑操作指令.按位与/或操作(PAND/POR)、移位操作(PSRLD/PSLLD)等.

2) 数据移动操作.数据存取(MOVDQA/MOVDQU)等.

3) 数据交换.按字节置换(PSHUFB)、按 32 b 字交换(PUNPCKLDQ/PUNPCKHDQ).

2.2 串行 Pack 和 Unpack 过程

PackedBinary 和 PForDelta 算法压缩和解压过程中有个共同的步骤:Pack 和 Unpack.Pack 步骤一次对一组整数进行压缩,而 Unpack 步骤一次对一组整数进行解压缩.我们将一组整数称为一个块,块的大小指压缩之前块中整数的个数.通常,块大小是 32 的倍数.

针对不同的固定位宽值 w 块的压缩和解压缩,需要实现不同的 Pack 和 Unpack 过程,其流程是相似的:一次执行时压缩/解压一个块,在块内分成多个循环进行,每次循环使用移位操作和按位或/按位

与操作,Pack 时将 32 个 32 b 整数编码成 w 个 32 b 字,Unpack 时从 w 个 32 b 字中顺序地解压出 32 个整数.

2.3 并行化 Unpack 过程

并行化 Pack 和并行化 Unpack 的步骤也互为一组逆过程,但是在实现中,并行化 Pack 过程比起 Unpack 过程有更多的步骤.这里我们先介绍 Unpack 过程,然后再介绍 Pack 过程.下面以 Unpack5 为例介绍 Unpack 过程.

将 Unpack 并行化的难点或者挑战性在于,绝大部分 Unpack 过程中,压缩后整数的位宽都不是 8 的倍数(如 Unpack3,Unpack5,Unpack7 等),即不是字节对齐的.因此,需要使用一些逻辑移位指令(如 PSRLD,PSLLD)来使每个整数都处于字节对齐的位置.并行化 Unpack 函数在每次循环中可以解压出 16 个整数.图 1 说明了 Unpack5 过程中使用 SIMD 指令一次解压出 16 个整数 $a_0,a_1,a_2,a_3,b_0,b_1,b_2,b_3,c_0,c_1,c_2,c_3,d_0,d_1,d_2,d_3$ 的基本原理,该过程总共分为 8 个步骤:

① 从内存读入数据(Load From Memory).从内存中读入含有 $a_0,\cdots,a_3,b_0,\cdots,b_3,c_0,\cdots,c_3,d_0,\cdots,d_3$ 的 10B(80 b)的数据至 128 b XMM 寄存器 X_1 中(未在图 1 中画出).

② 交换-1(Shuffle-1).对 X_1 进行字节置换操作,将包含 $a_0,\cdots,a_3$ 的字节移至 X_2 最低处,包含 $c_0,\cdots,c_3$ 的字节移至 X_2 的低 64 b 位置.

③ 交换-2(Shuffle-2).对 X_1 进行字节置换操作,将 $b_0,\cdots,b_3,d_0,\cdots,d_3$ 分别移至 X_3 的低 36 b、低 100 b 位置处.此时 X_3 中的 b_0 和 d_0 不是 32 b 字对齐的,需进一步处理.

④ 右移和按位或(Shift Right and OR).将 X_3 中每个 32 b 字右移 4 b(右移指向低位移动,和图 1 方向相反),再和 X_2 进行按位或操作,将结果放入 X_2 中.此时 X_2 中含有全部的 16 个整数,且第 i 个整数和第 $i+4$ 个整数(如整数 a_0 和 b_0)最低位之间距离均为 32 b,其中 $i=1,\cdots,12$.

⑤ 复制和右移(Copy and Shift Right).使用复制和 32 b 字右移操作,将 X_2 中第 $i,i+4,i+8,i+$

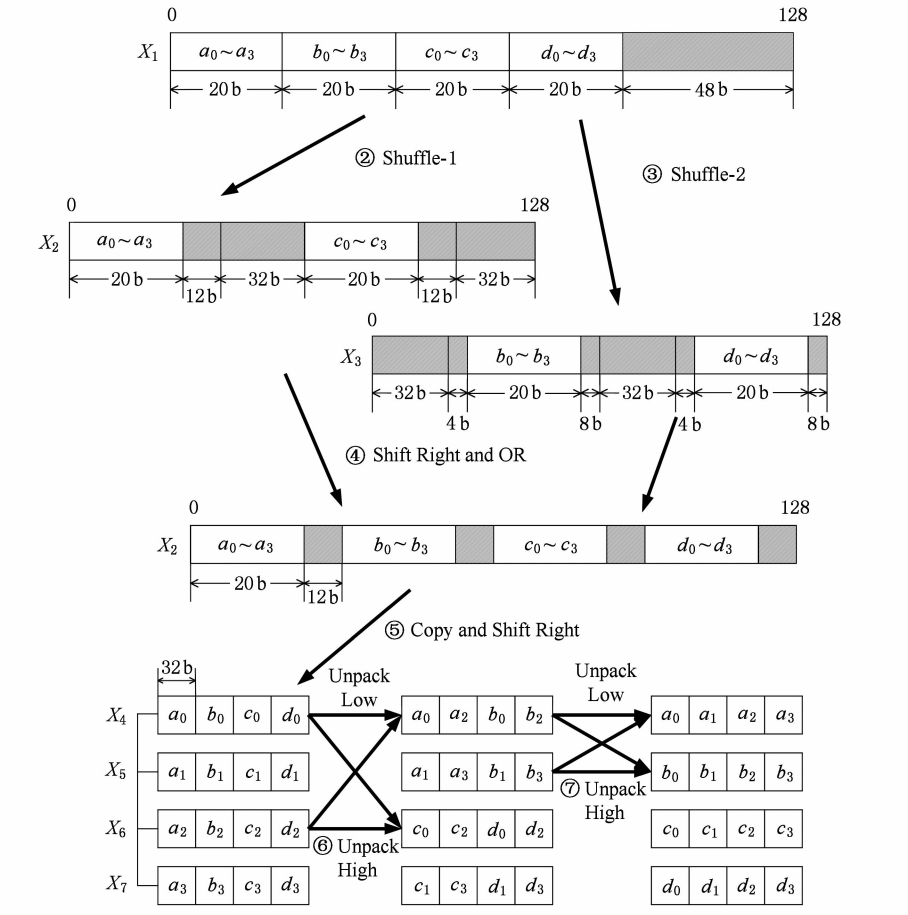


Fig. 1 Example of parallelized Unpack5.

图 1 并行 Unpack5 示例

12 个数字放入一个 XMM 寄存器中,执行 4 次,$i=1,2,3,4$. 得到新的含有数据的寄存器 $X_4,\cdots,X_7$.

⑥ 交换高低双四字-1(Unpack Low and High Double Quad-1). 通过对 $X_4,\cdots,X_7$ 进行交织放置低、高位 64 b 数据的操作,将 a_0,a_2,c_0,c_3 放在 X_4,a_1,a_3,c_1,c_3 放在 X_5,b_0,b_2,d_0,d_2 放在 X_6,b_1,b_2,d_1,d_3 放在 X_7 中.

⑦ 交换高低双四字-2(Unpack Low and High Double Quad-2). 通过对 $X_4,\cdots,X_7$ 继续进行交织放置低、高位 64 b 数据的操作,得到含有 4 个数字 $i,i+1,i+2,i+3$ 的寄存器,其中 $i=1,5,9,13$.

⑧ 写回内存(Write to Memory). 将 4 个寄存器中的 16 个数字 $a_0,\cdots,a_3,b_0,\cdots,b_3,c_0,\cdots,c_3,d_0,\cdots,d_3$ 按序写回内存(未在图 1 中画出).

Unpack6 中的步骤和 Unpack5 类似,只是在步骤③的交换操作中,数字 $b_0,\cdots,b_3$ 是字节对齐的,$d_0,\cdots,d_3$ 也是字节对齐的,因此可以将步骤②③的交换操作合并,通过一次交换操作直接得到步骤④中的 XMM 寄存器结果.

其他 Unpack 并行函数在实现上和 Unpack5,Unpack6 只有细微的差别. 例如在 Unpack1 至 Unpack4 并行函数的每个循环中,32 个压缩后的整数最多只占 $4\times32=128$ b,因此只需 1 次 MOVDQU 操作就能从内存把数据读入 XMM 寄存器中,而 Unpack5 至 Unpack8 需要 2 次 MOVDQU 操作.

在 Unpack9 至 Unpack16 中,由于每个整数的有效位宽大于 8,无法像 Unpack1 至 Unpack8 那样,一次性将 16 个整数载入一个 XMM 寄存器中. 因此,需要通过 2 次内存读取,将 8 个数字 $a_0,\cdots,a_3,b_0,\cdots,b_3$ 和 8 个数字 $c_0,\cdots,c_3,d_0,\cdots,d_3$ 分别载入 2 个 XMM 寄存器中,再使用交换、复制和右移等步骤进行操作.

从 Unpack 并行实现的算法可以看到:对于 32 个整数,在串行实现中每次移位和按位与操作只将 1 个整数写入内存;而在并行化实现中,每次可以将位置相邻的 4 个整数写入内存. 这样做每次可以节省 3 次内存寻址操作,增加 CPU 指令缓存命中率,大大提升了解压速度.

2.4 并行化 Pack 过程

并行化 Pack 过程基本上是并行化 Unpack 的逆过程,我们以 Pack5 为例介绍 Pack 过程. Pack5 每次也是处理 16 个整数,如图 2 所示,分为 8 个步骤:

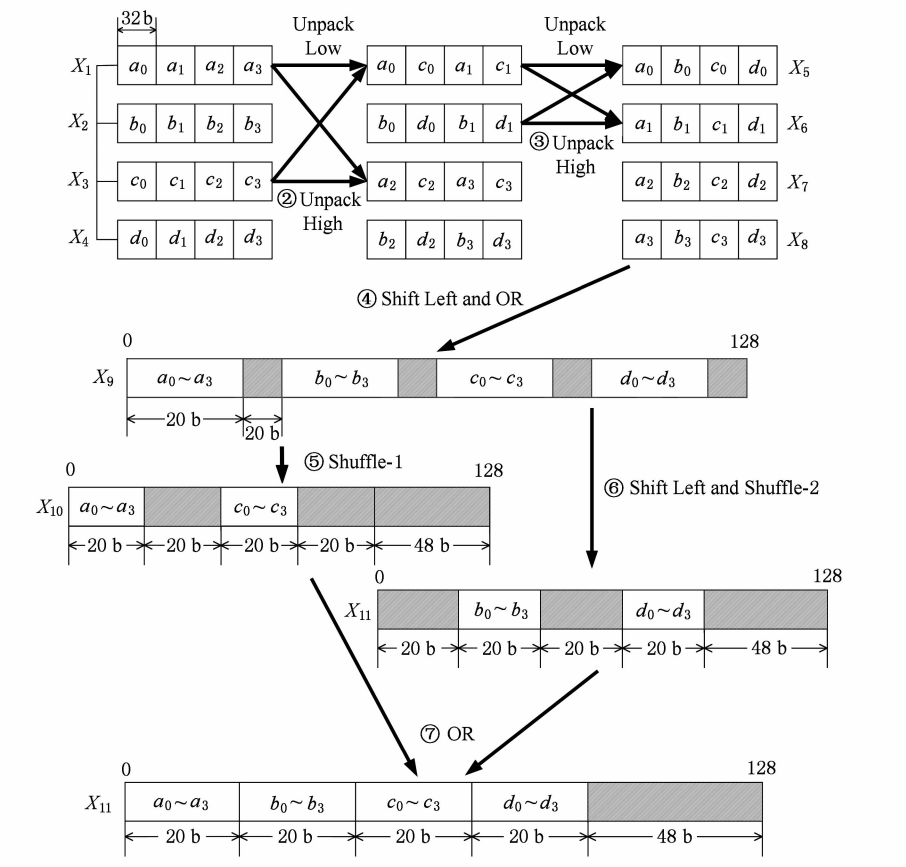


Fig. 2 Example of parallelized Pack5.

图 2 并行 Pack5 示例

① 从内存读入(Load From Memory). 从内存中读取 16 个整数 $a_0,\cdots,a_3,b_0,\cdots,b_3,c_0,\cdots,c_3,d_0,\cdots,d_3$ 至 4 个 XMM 寄存器 $X_1,\cdots,X_4$.

② 交换高低双四字-1(Unpack Low and High Double Quad-1). 同 Unpack 的步骤⑥,得到含有 4 个数字 i, $i+4$, $i+8$, $i+12$ 的寄存器 $X_5,\cdots,X_8$, 其中 $i=1,2,3,4$.

③ 交换高低双四字-2(Unpack Low and High Double Quad-2). 同 Unpack 的步骤⑦,得到含有 4 个数字 $i,i+4$, $i+8$, $i+12$ 的寄存器 $X_5,\cdots,X_8$,其中 $i=1,2,3,4$.

④ 左移和按位或(Shift Left and OR). 对 $X_5,\cdots,X_8$ 分别进行 32 b 字左移操作,然后使用按位或操作存入同一个寄存器 X_9 中,使得第 $i,i+1,i+2,i+3$ 数字连续存放在 X_9 的第 i 个字节开始的位置. $i=1,5,9,13$.

⑤ 交换-1(Shuffle-1). 将 X_9 进行字节置换操作,将 8 个数字 $a_0,\cdots,a_3,c_0,\cdots,c_3$ 放入正确位置,其他部分清 0,得到寄存器 X_{10}.

⑥ 左移和交换-2(Shift Left and Shuffle-2). 将 X_9 中每个 32 b 字分别左移 4 b,再进行字节置换操作,将 8 个数字 $b_0,\cdots,b_3,d_0,\cdots,d_3$ 放入正确位置,其他部分清 0,得到寄存器 X_{11}.

⑦ 按位或(OR). 将 X_{10} 和 X_{11} 进行按位或操作,得到 16 个数字 $a_0,\cdots,a_3,b_0,\cdots,b_3,c_0,\cdots,c_3,d_0,\cdots,d_3$ 放到正确位置的寄存器 X_{11}.

⑧ 写回内存(Write to Memory). 将 X_{11} 写回内存.

2.5 交织存储布局的优化

从 2.3 节和 2.4 节中可知,并行 Unpack 过程的步骤⑥⑦将 $a_0,b_0,c_0,d_0,a_1,b_1,c_1,d_1,a_2,b_2,c_2,d_2,a_3,b_3,c_3,d_3$ 这 16 个整数的位置进行置换,变成 $a_0,\cdots,a_3,b_0,\cdots,b_3,c_0,\cdots,c_3,d_0,\cdots,d_3$ 的存放顺序;而并行 Pack 过程的步骤②③是将 16 个整数从 $a_0,\cdots,a_3,b_0,\cdots,b_3,c_0,\cdots,c_3,d_0,\cdots,d_3$ 的顺序置换成 $a_0,b_0,c_0,d_0,a_1,b_1,c_1,d_1,a_2,b_2,c_2,d_2,a_3,b_3,c_3,d_3$ 的顺序. 显然对应的 2 个过程是互为逆过程的. 我们可以在并行 Pack 过程中直接将 16 个整数以交织存放的顺序存储在内存中,而在并行 Unpack 过程中直接取出该方式存放的 16 个数进行后续操作. 图 3 表示了位宽为 5 b 的 16 个整数 $Int_1,Int_2,\cdots,Int_{16}$ 压缩之后在内存中的交织存储布局. 可以看到,16 个整数的交织存储,不会影响到整数占用存储空间大小,却可以将 Pack 过程的步骤②③和 Unpack 过程的步骤⑥⑦省去,进一步提升压缩和解压速度. 在实验中我们采用了此存储布局.

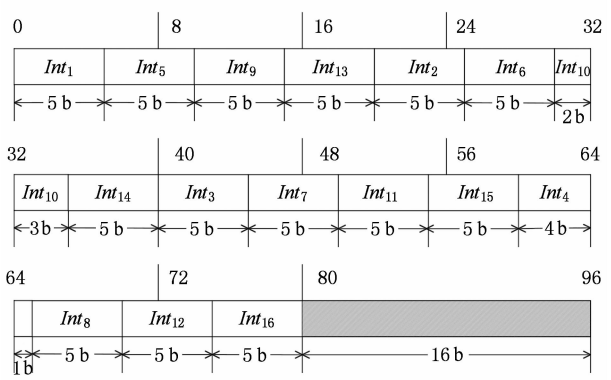


Fig. 3 Vertical storage layout of 16 integers of 5 bits each.

图 3 16 个位宽为 5 b 的整数在压缩后的交织存储布局

2.6 讨 论

从 2.3 节和 2.4 节可以看出,我们的并行化算法在时间上和串行算法一样,依然是线性的;在空间上,没有用到任何多余的内存空间. 此外,并行 Pack 和 Unpack 过程还可以用在 SIMD-PFD 算法中的异常数组的压缩/解压上(Pack/Unpack8 和 Pack/Unpack16).

事实上,在所有使用固定位宽的压缩算法中,均会有部分的 Pack 和 Unpack 过程. 因此,并行 Pack 和并行 Unpack 过程是可以应用到其他使用固定位宽的压缩算法中的. 只要 Pack 和 Unpack 时每组整数的块大小是 16 的倍数,便可以用我们的并行 Pack 和并行 Unpack 过程算法进行并行化,同时不改变算法的其他部分.

例如在 VSEncoding 算法[14]中,块大小有 1,2,4,8,12,16,32,64 共 8 种;在 AFOR[13]中,块大小有 8,16,32 共 3 种. 我们可以对这 2 种算法的大小中整数个数在 16 以上的块使用并行 Pack 和 Unpack 算法;或者将算法中所有的块值均扩大到原来的 2~4 倍(例如 AFOR 的块大小增大为 16,32,64),可以更充分地应用并行 Pack 和 Unpack 算法来提高压缩性能. 而对于 Rice 编码[6],我们可以先将所有整数的具有固定位宽的余数部分存储在一起后,再进行并行的 Pack 和 Unpack 操作.

3 实验结果与分析

本节首先介绍实验环境和实验数据集;然后把我们提出的 SIMD-PB 算法和 SIMD-PFD 算法与其他常用的倒排链压缩算法在压缩速度、解压速度和压缩率上进行比较;接下来重点关注解压速度,分别

考察倒排链长度、块大小和异常数比率3个因素对SIMD-PB算法、SIMD-PFD算法解压速度的影响.

3.1 实验环境和数据集

实验机器是1台具有48 GB内存、CPU频率2.4 GHz(Intel® Xeon® E5645)的64 b Linux服务器.各个压缩算法均采用C++实现,并使用"-O3"的编译优化级别,在G++-4.6上编译通过.

实验基于文本检索会议(Text REtrieval Conference, TREC)提供的2个公开数据集GOV2和ClueWeb09B,对其中的标题和正文2个域建立倒排索引.GOV2是美国国家标准与技术研究院(National Institute of Standards and Technology, NIST)于2004年上半年从.gov域名下的网站抓取的2 500万网页集合.ClueWeb09B是TREC09评测用的数据集,包含5 000万网页.我们从数据集对应的查询集中随机找出1 000个查询串,提取出所有不重复的词,并过滤掉其中的停用词后将剩下的词作为查询词集合.我们对查询词集合中每个词对应的倒排链中的d-gap序列和TF序列,分别采用不同的算法进行压缩和解压操作,再将各个倒排链的解压速度和压缩率进行平均,得到最终的比较结果.

3.2 实验结果

实验中将SIMD-PB算法和SIMD-PFD算法与其他常用的倒排链压缩算法进行了比较,其中带有SIMD-前缀的算法表明它是原先的顺序算法采用SIMD指令的并行版本,如未指出,则整数块大小取1 024,PFD算法和SIMD-PFD算法的异常数比率取4%.

表1列出了各个算法在GOV2和ClueWeb09B数据集下对倒排链中d-gap序列和TF序列的解压缩速度和压缩速度,单位为MIPS(每秒百万整数).从表1可以发现,基于指令级并行的压缩算法都比串行化的压缩算法快.我们提出的SIMD-PB算法相比于其串行版本PackedBinary算法,效果提升了大约25%;SIMD-PFD算法相比于其串行版本PForDelta算法,解压速度提升了50%~100%.此外,SIMD-PB算法和SIMD-PFD算法在2个数据集的d-gap和TF上均普遍优于以往研究中最快的字节对齐倒排链解压算法SIMD-G8IU[12].例如在GOV2索引的d-gap解压速度中,我们提出的SIMD-PFD算法比SIMD-G8IU算法的解压缩速度快20%左右,压缩率提升近1倍;而SIMD-PB算法由于没有异常数解压的部分,解压速度是最快的.从表1中各个算法的压缩速度上可以看到,我们提出的SIMD-PB算法在压缩速度上有着最好的性能,且相比于串行的PackedBinary算法提升了150%左右;而SIMD-PFD算法的主要开销在于寻找最合适的位宽w以及异常数的挑选上,这些串行操作需要的时间远远大于Pack过程,使得并行化Pack过程之后速度提升并不明显,只有2%左右.

Table 1 Compression and Decompression Speed of Compression Algorithms

表1 各个算法在不同数据集下对d-gap和TF的解压和压缩速度比较 MIPS

Algorithm	Decoding Speed				Encoding Speed			
	GOV2		ClueWeb09B		GOV2		ClueWeb09B	
	d-gap	TF	d-gap	TF	d-gap	TF	d-gap	TF
SIMD-GB	837	834	819	827	270	262	266	270
SIMD-G8IU	1 647	1 690	1 541	1 544	134	128	154	142
SIMD-G8CU	1 306	1 331	1 271	1 287	129	126	144	139
SIMD-PB	**1 934**	**1 926**	**1 580**	**1 749**	**691**	**637**	**667**	**660**
SIMD-PFD	1 910	1 903	1 507	1 719	42	25	41	29
GVB_Binary	513	514	511	511	270	262	267	271
G8IU	531	547	515	548	133	128	149	142
G8CU	496	409	481	514	128	125	143	138
PackedBinary	1 484	1 444	1 345	1 442	256	255	258	259
PForDelta	1 117	993	1 013	955	41	24	39	29
VarByte	551	670	514	685	614	546	602	605
Simple9	530	641	525	745	141	78	161	104
Simple16	490	639	515	739	76	45	97	59
Rice	65	83	79	85	83	70	115	72

表 2 列出了各个算法的压缩率,值越小代表压缩效果越好. 显然,SIMD 版本的算法与其串行版本的压缩率是相同的. 我们发现,由于数据集中 90% 的整数需要位宽值小于 8,因此字节对齐的压缩算法(VB,GB,G8IU,G8CU)的压缩率较差;相比之下 SIMD-PB 算法和 SIMD-PFD 算法就要好得多,SIMD-PFD 算法由于有异常数的处理,在压缩率上又好于 SIMD-PB 算法;压缩率最好的是比特对齐的 Rice 算法,但其解压速度只有 SIMD-PFD 算法的 3.4%.

Table 2 Compression Ratio of Compression Algorithms

表 2 各算法在不同数据集下对 d-gap 和 TF 的压缩率比较

%

Algorithm	GOV2		ClueWeb09B	
	d-gap	TF	d-gap	TF
SIMD-GB	31.39	31.49	31.40	31.32
SIMD-G8IU	28.30	28.44	28.32	28.21
SIMD-G8CU	28.28	28.40	28.29	28.20
SIMD-PB	14.18	27.21	18.70	23.37
SIMD-PFD	13.37	20.95	15.06	19.05
GB	31.39	31.49	31.40	31.32
G8IU	28.30	28.44	28.32	28.21
G8CU	28.28	28.40	28.29	28.20
PackedBinary	14.18	27.21	18.70	23.37
PForDelta	13.37	20.95	15.06	19.05
VarByte	25.29	25.64	25.30	25.26
Simple9	11.69	18.92	9.36	15.18
Simple16	10.77	17.42	8.63	14.00
Rice	**8.66**	**15.24**	**8.05**	**12.96**

3.3 各因素对 SIMD-PB 算法和 SIMD-PFD 算法解压速度的影响

本节中我们考虑不同因素对 SIMD-PB 算法和 SIMD-PFD 算法解压速度影响. 如无特殊说明,本节使用的数据集均为 GOV2 索引上的 d-gap 序列.

1) 倒排链长度. 我们在不同的文档频率区间中分别随机选取了至多 100 个词项,测试它们对应的倒排链解压速度. 其中,文档频率区间有 5 段,分别为 $[10^3, 10^4)$, $[10^4, 10^5)$, $[10^5, 10^6)$, $[10^6, 10^7)$, $[10^7, 10^8)$. 从 3.2 节的实验中,我们看到以往研究中解压速度最快的算法是 SIMD-G8IU. 因此,我们将以往研究中解压速度最快的 SIMD-G8IU 算法和我们的 SIMD-PB 算法和 SIMD-PFD 算法进行了比较,结果如图 4 所示. 从实验结果可以看到,在 2 个数据集下,在倒排记录数目大于等于 10^4 的倒排链中,SIMD-PB 算法和 SIMD-PFD 算法解压速度要优于 SIMD-G8IU 算法.

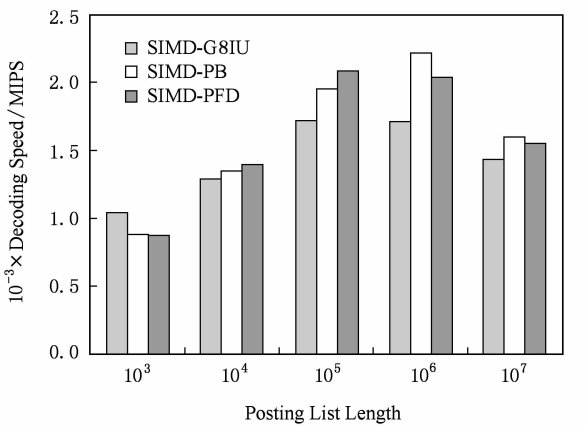


Fig. 4 Decompression speed on different posting list length.

图 4 倒排链长度对解压速度影响

2) 块大小. 我们考虑了块大小对 SIMD-PB 算法和 SIMD-PFD 算法解压速度的影响. 我们将块的大小设置为 2 的整数次幂,最小值为 32,最大值为 8 192. 实验结果如图 5 所示. 从结果可以发现,在不同的数据集和不同类型的整数序列下,SIMD-PB 算法和 SIMD-PFD 算法的解压速度均随着块大小的增大而上升. 在块大小小于 1 024 时,解压速度快速上升;在块大小达到 1 024 个整数后,解压速度趋于稳定. Unpack 过程在每次解压一个块前,需要读取块的位宽,以及对应的字节置换数组和掩码数组等,这些预处理操作将导致 CPU 的缓存命中失效,需要从内存重新加载数据,影响数据解压速度,而块大小的增加可以减少块的个数,从而减少缓存失效的总时间. 然而,随着块大小的增加,SIMD-PB 算法和

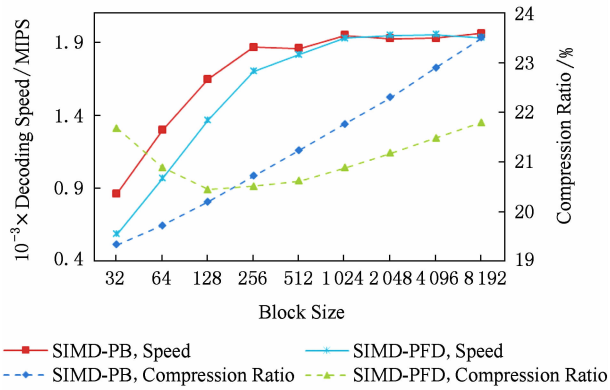


Fig. 5 Decompression speed of SIMD-PB and SIMD-PFD on different block size.

图 5 块大小对解压速度影响

SIMD-PFD 算法的压缩率会变差. 因此,在权衡解压速度和压缩率后,将块大小均取 1 024 比较合适.

3) 异常数比率. 为改进 PForDelta 算法的解压速度,我们对 Unpack 部分进行了并行化. 但是,算法异常数解压部分仍然是串行的:异常数比率越高,每个块中的异常数数目就越多,异常数组越长,将会使 SIMD-PFD 算法解压速度变慢. 但是异常数比率对压缩率也有影响,所以需要选择合适的异常数比率. 从图 6 可以看到,随着异常数比率的增加,SIMD-PFD 算法的解压速度逐渐下降;同时,算法的压缩率当异常数比率大于 0.04 时,效果下降并不明显. 因此,将异常数比率设为 4%.

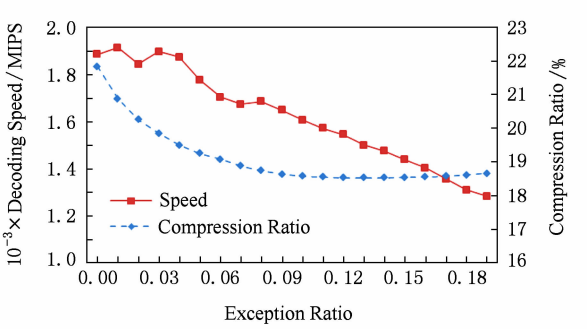


Fig. 6 Decompression speed of SIMD-PFD on different exception ratio.

图 6 异常数比率对 SIMD-PFD 解压速度影响

4 结束语

本文主要研究倒排索引的压缩问题,通过研究目前解压速度较快的串行倒排链压缩算法 PackedBinary 和 PForDelta,我们使用 SIMD 向量指令对这 2 种压缩算法的关键步骤 Pack 过程和 Unpack 过程进行了改进,提出了 2 种基于指令级并行的压缩算法 SIMD-PB 和 SIMD-PFD. 在基于 GOV2 和 ClueWeb09B 公开数据的倒排索引上的大量实验表明,SIMD-PB 算法和 SIMD-PFD 算法有着良好的压缩和解压缩性能,在压缩速度和解压速度方面均优于以往的倒排链压缩算法.

未来,我们将继续改进 SIMD-PB 算法和 SIMD-PFD 算法的 SIMD 并行压缩和解压缩部分,同时对 SIMD-PFD 算法的异常数压缩/解压部分进行优化;尝试将 SIMD-PB 算法,SIMD-PFD 算法的压缩/解压过程和其他粗粒度的并行技术进行结合;同时将探究其他倒排链压缩算法的指令级并行的可能性.

参 考 文 献

[1] Witten I H, Moffat A, Bell T C. Managing Gigabytes: Compressing and Indexing Documents and Images [M]. 2nd ed. San Francisco: Morgan Kaufmann, 1999

[2] Anh V N, Moffat A. Index compression using 64-bit words [J]. Software: Practice and Experience, 2010, 40(2): 131-147

[3] Heman S. Super-scalar database compression between RAM and CPU-cache [D]. Amsterdam, Holland: University of Amsterdam, 2005

[4] Zukowski M, Heman S, Nes N, et al. Super-scalar RAM-CPU cache compression [C] //Proc of the 22nd Int Conf on Data Engineering. Los Alamitos, CA: IEEE Computer Society, 2006: 59-71

[5] Elias P. Universal codeword sets and representations of the integers [J]. IEEE Trans on Information Theory, 1975, 21(2): 194-203

[6] Rice R, Plaunt J. Adaptive variable-length coding for efficient compression of spacecraft television data [J]. IEEE Trans on Communication Technology, 1971, 19(6): 889-897

[7] Scholer F, Williams H, Yiannis J, et al. Compression of inverted indexes for fast query evaluation [C] //Proc of the 25th Annual Int ACM SIGIR Conf on Research and Development in Information Retrieval. New York: ACM, 2002: 222-229

[8] Dean J. Challenges in building large-scale information retrieval systems: Invited talk [C] //Proc of the 2nd ACM Int Conf on Web Search and Data Mining. New York: ACM, 2009

[9] Anh V N, Moffat A. Inverted index compression using word-aligned binary codes [J]. Information Retrieval, 2005, 8(1): 151-166

[10] Anh V N, Moffat A. Improved word-aligned binary compression for text indexing [J]. Knowledge and Data Engineering, 2006, 18(6): 857-861

[11] Zhang J, Long X, Suel T. Performance of compressed inverted list caching in search engines [C] //Proc of the 17th Int Conf on World Wide Web. New York: ACM, 2008: 387-396

[12] Yan H, Ding S, Suel T. Inverted index compression and query processing with optimized document ordering [C] //Proc of the 18th Int Conf on World Wide Web. New York: ACM, 2009: 401-410

[13] Delbru R, Campinas S, Tummarello G. Searching Web data: An entity retrieval and high-performance indexing model [J]. Web Semantics, 2012, 10: 33-58

[14] Silvestri F, Venturini R. VSEncoding: Efficient coding and fast decoding of integer lists via dynamic programming [C] // Proc of the 19th ACM Int Conf on Information and Knowledge Management. New York: ACM, 2010: 1219-1228

[15] Stepanov A A, Gangolli A R, Rose D E, et al. SIMD-Based decoding of posting lists [C] //Proc of the 20th ACM Int Conf on Information and Knowledge Management. New York: ACM, 2011: 317-326

[16] Zhang Xudong, Sun Zhiming, Liu Yaning, et al. Inverted index compression algorithms based on 64-bit architecture [J]. Computer Engineering, 2014, 40(2): 71-76 (in Chinese)
(张旭东, 孙志明, 刘亚宁, 等. 基于64位体系结构的倒排索引压缩算法[J]. 计算机工程, 2014, 40(2): 71-76)

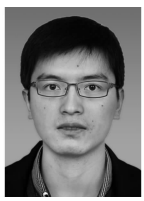
[17] Wang Hu, Wang Qianping. Research and analysis on five inverted files compression techniques [J]. Computer Engineering and Applications, 2006, 42(7): 169-173 (in Chinese)
(王虎, 王潜平. 对几种倒排文件压缩技术的研究与分析[J]. 计算机工程与应用, 2006, 42(7): 169-173)

[18] Zhu Hong, Wu Lin. Compression of inverted and implementation in full-text information retrieval system RDBMS [J]. Journal of Huazhong University of Science and Technology: Natural Science Edition, 2005, 33(4): 7-9 (in Chinese)
(朱虹, 吴林. 倒排索引压缩及在RDBMS全文检索中的实现[J]. 华中科技大学学报: 自然科学版, 2005, 33(4): 7-9)

[19] Intel Corporation. Intel 64 and IA-32 Architectures Software Developers Manual (Version 37) [M]. Santa Clara, CA: Intel Corporation, 2010

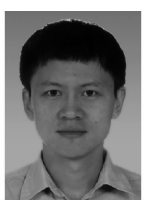
Yan Hongfei, born in 1973. PhD. Associate professor at Peking University. Member of China Computer Federation. His current research interests include information retrieval and distributed system.

Zhang Xudong, born in 1989. Master candidate at Peking University. His research interests include information retrieval and search engine(zhang. xd927@gmail. com).



Shan Dongdong, born in 1985. PhD. Software engineer of Searching Department in Taobao (China) Software Co., Ltd. His research interests include information retrieval and search engine(shandd2008@gmail. com).

Mao Xianling, born in 1983. PhD. Lecturer at Beijing Institute of Technology. His research interests include information retrieval and Web data mining(mxl. cs. pku@gmail. com).



Zhao Xin, born in 1985. PhD candidate at Peking University. His research interests include information retrieval and Web data mining(batmanfly@gmail. com).