CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 44 卷第 1 期 2022 年 1 月
Vol. 44, No. 1, Jan. 2022

文章编号:1007-130X(2022)01-0001-08

基于 RISC-V 向量指令的稀疏矩阵向量乘法实现与优化*

顾　越,赵银亮

(西安交通大学计算机学院,陕西 西安 710049)

摘　要:开源指令集架构 RISC-V 具有高性能、模块化、简易性和易拓展等优势,在物联网、云计算等领域的应用日渐广泛,其向量拓展部分 V 模块更是很好地支持了矩阵数值计算。稀疏矩阵向量乘法 SpMV 作为矩阵数值计算的一个重要组成部分,具有深刻的研究意义与价值。利用 RISC-V 指令集的向量可配置性和寻址特性,分别对基于 CSR、ELLPACK 和 HYB 压缩格式存储的稀疏矩阵向量乘法进行向量化。同时,考虑稀疏矩阵极度稀疏和每行非零元素数量波动较大的情况,通过压缩非零元素密度低的行向量的存储、调整 HYB 分割阈值等手段,改进了 HYB 存储格式,显著改善了计算效率和存储效率。

关键词:RISC-V;向量拓展;稀疏矩阵;SpMV

中图分类号:TP311.11　　**文献标志码**:A

doi:10.3969/j.issn.1007-130X.2022.01.001

Implementation and optimization of sparse matrix vector multiplication based on RISC-V vector instruction

GU Yue, ZHAO Yin-liang

(School of Computer, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract: Open source instruction set architecture RISC-V has the advantages of high performance, modularization, simplicity, easy extension, etc., and is widely used in the Internet of Things, cloud computing and other fields. The V module of its vector expansion part supports matrix numerical calculation well. As an important part of matrix numerical calculation, sparse matrix vector multiplication (SpMV) has profound research significance and value. Using the vector configurability and addressing characteristics of RISC-V instruction set, vector multiplication of sparse matrix based on CSR, ELLPACK and HYB compressed format is vectorized respectively. Meanwhile, considering that the sparse matrix is extremely sparse and the number of non-zero elements in each row fluctuates greatly, the HYB storage format is improved by compressing the storage of row vectors with low density of non-zero elements and adjusting the HYB segmentation threshold, which significantly improves the computational efficiency and storage efficiency.

Key words: RISC-V; vector expansion; sparse matrix; SpMV

1　引言

随着云计算与物联网浪潮的兴起,人们对可定制、低功耗和高性价比 CPU 的需求日渐增加。最初由加州大学伯克利分校研究人员开发的开源指令集架构 RISC-V,凭借其可伸缩、可扩展和模块化的特性[1-3],在相关领域的影响力逐渐扩大。该指令集由整数基本指令和多个拓展指令集组成[4],拓展指令集中包括了用于向量处理的 V 指令

* 收稿日期:2020-03-23;修回日期:2021-08-24
基金项目:广东省重点领域研发计划(2019B090916003)
通信作者:赵银亮(zhaoy@xjtu.edu.cn)
通信地址:710049 陕西省西安市西安交通大学计算机学院
Address: School of Computer, Xi'an Jiaotong University, Xi'an 710049, Shaanxi, P. R. China

集——RV32V。RV32V 目前仍在 GitHub 上开发[5],最新稳定版本为 v0.9。与传统 SIMD 向量架构不同的地方在于,RISC-V 向量指令集具有向量寄存器可配置的特性,解决了向量计算中向量长度不可知的问题[6,7]。

向量计算中的稀疏矩阵与向量乘法 SpMV(Sparse Matrix-Vector multiplication),常用于科学计算和工程应用问题中的大规模稀疏线性系统求解[8],其实现与优化一直是高性能计算领域的研究重点。因稀疏矩阵具有非零元素占比少的特性,研究人员常采用压缩存储格式来提高存储效率,又因非零元素分布和硬件设备的多样性,其存储格式也具有多样性[9]。如 COO(COOrdinate list format)[10]格式以行坐标、列坐标和值组成的三元组对稀疏矩阵中非零元素进行存储,简单直观灵活,但不利于计算和向量化。另一种常用的存储格式 CSR(Compress Sparse Row format)格式,将所有非零元素的值和列索引按行顺序存储在 2 个单独的数组中,第 3 个数组保留指向这些数组中每一行的起始位置的指针。它的主要优点是对任何类型的矩阵元素分布都有很好的压缩比,其主要缺点为数据局部性差,存在写冲突、负载不均衡等问题[11]。ELLPACK(format used by ELLPACK system)格式可在 GPU 和矢量体系结构中高效运算,但需要付出额外的存储代价[12]。它以 $m*k$ 的矩阵存储数据,其中 k 的大小取决于每行非零元素个数中的最大值。面对不规则矩阵时,其存储和计算效率都会退化。HYB(HYBrid format)格式[13]结合了 ELLPACK 格式和 COO 格式,对每行中超过阈值的非零元素使用 COO 格式存储,以减少 ELLPACK 格式中零元素的填充,兼顾计算效率的同时节约了存储空间。

SELLPACK(Sliced ELLPACK format)[14]格式是基于 ELLPACK 格式的改进。通过将矩阵行向量按照其非零元素数量排序,以固定行数对矩阵分片,对得到的矩阵块分别使用 ELLPACK 格式存储,提高了存储不规则矩阵的效率和吞吐量。更进一步地,SELL-C-σ(Sliced ELLPACK format with parameter C,σ)格式[15]通过限制排序窗口避免了重新排列整个矩阵的高昂代价,同时充分利用了矩阵相邻行与被乘向量 x 相乘时,对向量 x 访问的局部性。针对 Intel 公司设计的异构众核处理器 Xeon Phi,Liu 等人[16]提出了 ESB(ELLPACK Sparse Block)格式,该格式引入了位数组编码和列分块技术,以降低带宽需求,进一步提高了向量访问的局部性。而 Liu 等人[17]提出的 CSR5 存储格式,易于从传统的 CSR 格式转换得到,对输入稀疏矩阵的非零元素分布不敏感。通过采用一种改进的分段求和算法实现了基于 CSR5 的 SpMV 算法,在跨平台测试中得到了较高的吞吐量。

RISC-V 向量指令集作为向量指令家族中的新晋成员,方兴未艾。相关研究尚处于起步阶段,基于 RISC-V 向量指令实现和优化 SpMV 算法的工作还未有报道,开展此项工作具有重要意义。

本文基于 RISC-V 向量指令集 RV32V,利用其向量寄存器可配置及寻址特性,实现了基于 CSR、ELLPACK 和 HYB 压缩格式的 SpMV 算法,比较了在稀疏矩阵非零元素不同分布情况下各个算法的计算与存储效率的变化。同时,针对极度稀疏矩阵每行非零元素数量波动较大的情况,通过压缩非零元素密度低的行向量的存储、调整 HYB 分割阈值等手段,改进了 HYB 存储格式并实现了对应的 SpMV 算法,显著改善了计算效率和存储效率。

2 稀疏矩阵存储格式

2.1 CSR 格式

CSR 格式是一种比较标准的稀疏矩阵存储格式,使用 3 类数据来表示:数值 val、列号 col 和行偏移 row。其中,数值 val 为非零元素的值,列号 col 为非零元素的列号,行偏移 row 表示某行的第 1 个非零元素在数值 val 中的起始偏移位置。对于如式(1)所示的矩阵 $\boldsymbol{A}$,采用 CSR 格式存储的结果如式(2)所示:

$$\boldsymbol{A}=\begin{pmatrix}1&7&0&0\\0&2&8&0\\5&0&0&9\\0&6&0&0\end{pmatrix} \tag{1}$$

$$\begin{aligned}row&=(0\quad 2\quad 4\quad 6\quad 7),\\col&=(0\quad 1\quad 1\quad 2\quad 0\quad 3\quad 1),\\val&=(1\quad 7\quad 2\quad 8\quad 5\quad 9\quad 6)\end{aligned} \tag{2}$$

CSR 格式的主要优点是对规则与不规则稀疏矩阵的存储都具有稳定良好的压缩比,主要缺点是基于此实现的 SpMV 算法存在数据访问局部性差、写冲突和负载不均衡等问题。

2.2 ELLPACK 格式

ELLPACK 格式是一种易于向量化的稀疏矩阵存储格式,使用 2 个和原始矩阵行数相同的矩阵

来表示:第 1 个矩阵 *val* 存的是数值,第 2 个矩阵 *col* 存的是列号,行号用自身所在的行来表示。对于行中存在的零元素,可以在 *val* 矩阵中填充零元素、在 *col* 矩阵中填充 11 来表示。对于如式(1)所示的矩阵 $\boldsymbol{A}$,采用 ELLPACK 格式存储的结果如式(3)所示:

$$\boldsymbol{val}=\begin{pmatrix}1 & 7\\ 2 & 8\\ 5 & 9\\ 6 & 0\end{pmatrix},\quad \boldsymbol{col}=\begin{pmatrix}0 & 1\\ 1 & 2\\ 0 & 3\\ 1 & -1\end{pmatrix} \tag{3}$$

当矩阵中某行的非零元素个数很多时,ELLPACK 格式的存储效率将急剧下降。

2.3　HYB 格式

HYB 格式对 ELLPACK 格式的缺点进行改进,采用 COO 格式与 ELLPACK 格式混合的方式对稀疏矩阵压缩存储。对每行元素个数小于阈值的部分采用 ELLPACK 格式,超过阈值的部分采用 COO 格式。其中,ELLPACK 格式如上所述,需要数值矩阵 *val* 和列号矩阵 *col* 存储,而 COO 格式使用 coo_r、coo_c 和 coo_v 分别存储元素的行号、列号和值。对于如式(4)所示的矩阵 $\boldsymbol{B}$,采用 HYB 格式存储的结果如式(5)所示:

$$\boldsymbol{B}=\begin{pmatrix}1 & 7 & 9 & 0\\ 0 & 2 & 8 & 0\\ 5 & 0 & 0 & 9\\ 0 & 6 & 0 & 0\end{pmatrix} \tag{4}$$

$$\boldsymbol{val}=\begin{pmatrix}1 & 7\\ 2 & 8\\ 5 & 9\\ 6 & 0\end{pmatrix},\quad \boldsymbol{col}=\begin{pmatrix}0 & 1\\ 1 & 2\\ 0 & 3\\ 1 & -1\end{pmatrix},$$
$$coo\_r=(0),\quad coo\_c=(2),\quad coo\_v=(9) \tag{5}$$

矩阵 $\boldsymbol{B}$ 中第 1 行第 3 列的元素值为 9,行号为 0,列号为 2,分别存储于 coo_v、coo_r 和 coo_c 中。因此,当每行非零元素数量相差较大时,采用 HYB 格式可节约存储空间,提高存储效率。

3　SpMV 描述、实现与优化

3.1　SpMV 描述

SpMV 可用式(6)描述:

$$\boldsymbol{y}=\boldsymbol{A}\ *\ \boldsymbol{x} \tag{6}$$

其中,$\boldsymbol{A}$ 为 $m\ *\ n$ 维的稀疏矩阵,$\boldsymbol{x}$ 为 $n\ *\ 1$ 维的乘数向量,$\boldsymbol{y}$ 为 $m\ *\ 1$ 维的结果向量。

为了更好地描述稀疏矩阵,假设矩阵中每行元素数量呈正态分布,正态分布标准差为 σ,均值为 μ。

为了表征稀疏矩阵中非零元素的分布特征,使用空行率、稠密度和波动度 3 个指标,其定义如式(7)~式(9)所示:

空行率=矩阵全零行数/矩阵总行数　(7)

稠密度=非零元素数/矩阵元素总数　(8)

波动度=标准差 σ/均值 μ　(9)

为了表征基于某特定压缩存储格式的 SpMV 算法性能,使用压缩率和加速比 2 个指标,其定义如式(10)和式(11)所示:

压缩率=压缩后数据大小/原始矩阵大小(10)

加速比=基于该压缩格式的 SpMV 执行时间/非压缩的 SpMV 执行时间　(11)

压缩率与压缩效率成反比,压缩率越小,说明该压缩存储格式的压缩效率越高。

其中,非压缩的 SpMV 伪代码如下所示:

```
for (int i = 0;i < m;i++) {
  for (int j=0;j < n;j++) {
    y[i]+= A[i*m+j] * x[j];
  }
}
```

加速比与计算效率成正比,加速比越大,说明所执行的算法计算效率越高。

3.2　RISC-V 向量指令集 RV32V

RISC-V 指令集采用模块化的方式组织,使用一个英文字母表示一个模块。其中向量指令部分使用“V”表示,32 位的 RISC-V 向量模块简记为 RV32V。对于 32 位的 RISC-V 向量架构,在执行向量乘操作的 for 循环时,可以用设置向量寄存器组的操作作为循环判断条件,每次循环后将当前剩余向量长度减去设置结果,发挥向量寄存器可配置的特性,避免了传统 SIMD 指令面对计算向量长度非架构所支持的向量数的整数倍时,需要对剩余尾向量进行额外的循环操作的缺点。例如,对于传统的拥有固定 128 位向量寄存器的处理器而言,在处

理包含 41 个 32 位整型元素的向量时,每次循环能够处理的元素个数为 4。10 次循环之后,需要额外循环处理剩余的 1 个元素。而在 RV32V 体系结构中,向量寄存器长度可通过类似 vsetvl_e32m8 的接口灵活配置,开始待处理向量较长时,使用大的向量寄存器位数,最后向量长度不足时,将向量寄存器长度设置为合适值,充分简化了代码编写,提高了程序效率。

在实现矩阵运算时,常用的向量操作包括间隔向量寻址、索引向量寻址、向量乘加和向量标量乘等,如图 1 所示。

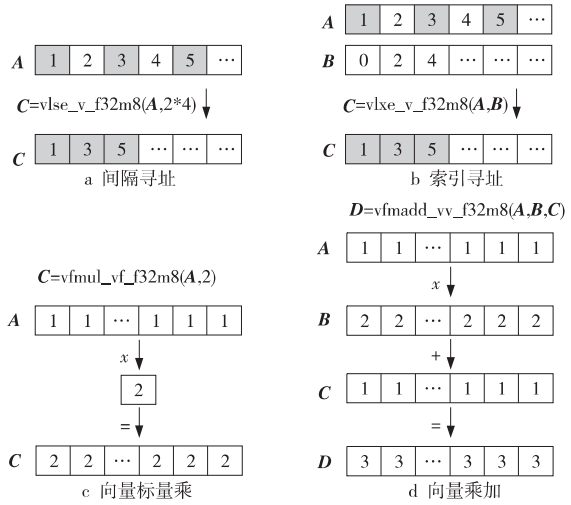

Figure 1 Vector operations of RV32V

图 1 RV32V 向量操作示意图

3.3 基于 RV32V 与传统压缩格式的 SpMV 实现

基于 RV32V,充分利用 RISC-V 向量寄存器可配置和寻址的特点,本文分别实现 CSR、ELLPACK 和 HYB 存储格式的 SpMV 算法。向量化 SpMV 算法实现总体可分为索引计算、乘数加载、向量相乘、结果回写和指针偏移这几个步骤,下面结合伪代码分别描述各压缩格式的 RISC-V 向量化 SpMV 实现。

基于 CSR 格式的 RISC-V 向量化 SpMV 伪代码如下所示:

输入 行偏移指针 row_p;row_p 数组长度 m;坐标指针 col_idx;值指针 val;向量指针 x。

输出 结果向量指针 y。

```
1:vfloat32m8_t vec_a,vec_b,vec_c;
2:vuint32m8_t index;
3:float *a=val,*b=x,*c=y;
4:while vl=vsetvl_e32m8(col) do
5:    vec_a=*(vfloat32m8_t *) a;
6:    index=*(vuint32m8_t *) col_idx;
7:    index=vfmul_vf_f32m8(index,4);
8:    vec_b=vlxe_v_f32m8(b,index);
9:    vec_c=*(vfloat32m8_t *) c;
10:   vec_c=vfmadd_vv_f32m8(vec_a,vec_b,vec_c)
11:   *(vfloat32m8_t *) c=vec_c;
12:   c+= vl;a+= vl;b+= vl;
13:   col_idx+= vl;col-= vl;
14:end while
15:return y.
```

首先对列号 col_idx 数组进行顺序寻址,计算出索引,如第 6、7 行所示;然后根据计算出的索引,在乘数向量 x 中进行索引寻址,加载第 1 个向量乘数,并对数值 val 数组顺序寻址,加载第 2 个向量乘数,如第 8、9 行所示;最后 2 个向量乘数中各元素依次相乘,结果回写暂存,如第 10、11 行所示。进行指针偏移准备下一轮计算,如第 12、13 行所示。依次循环往复直到计算完成。

基于 ELLPACK 格式的 RISC-V 向量化 SpMV 伪代码如下所示:

输入 坐标指针 col_idx;值指针 val;值矩阵长度 m;值矩阵宽度 col;向量指针 x。

输出 结果向量指针 y。

```
1:vfloat32m8_t vec_a,vec_b,vec_c;
2:vuint32m8_t index;
3:float *a=val,*b=x,*c=y;
4:for i=0 to col do
5:    num=m;a=val+i * col;
6:    b=x;c=y+i * col;
7:    col_idx=col_idx+i * col;
8:    while vl=vsetvl_e32m8(num) do
9:        vec_a=*(vfloat32m8_t *) a;
10:       index=vlse_v_f32m8(col_idx,col*4);
11:       index=vfmul_vf_f32m8(index,4);
12:       vec_b=vlxe_v_f32m8(b,index);
13:       vec_c=*(vfloat32m8_t *) c;
14:       vec_c=vfmadd_vv_f32m8(vec_a,vec_b,vec_
          c);
15:       *(vfloat32m8_t *) c=vec_c;
16:       c+= vl;a+= vl;b+= vl;
17:       col_idx+= vl;num-= vl;
18:   end while
19:end for
20:return y.
```

向量化计算过程中,首先对列号 col 数组间隔寻址,间隔为每行字节数,按列顺序获取向量,如第 10 行。对得到的向量值做向量标量乘法,转化为字节数,计算出索引,如第 11 行;然后根据计算出的索引,在乘数向量 x 中进行索引寻址,加载第 1 个向量乘数,如第 12 行;接着对数值 val 数组顺序

寻址,加载第 2 个向量乘数,如第 13 行;最后向量元素依次相乘,如第 14 行,结果回写暂存,如第 15 行。指针偏移准备下一轮计算,如第 16、17 行,依次循环往复直到计算完成。

对基于 HYB 格式的向量化 SpMV 可以分为 ELLPACK 和 COO 2 部分,其中 ELLPACK 部分与上述基于 ELLPACK 格式的 RISC-V 向量化过程相同,COO 部分的向量化过程与 CSR 格式的类似。

基于 HYB 格式的 RISC-V 向量化 SpMV 伪代码(COO 部分)如下所示:

输入　行坐标指针 *col_r*;列坐标指针 *coo_c*;值指针 *val*;长度 *num*;向量指针 *x*。

输出　结果向量指针 *y*。

```
1:while vl=vsetvl_e32m8(num) do
2:    vec_a=*(vfloat32m8_t *) a;
3:    index=*(vuint32m8_t *) coo_c;
4:    index=vfmul_vf_f32m8(index,4);
5:    vec_b=vlxe_v_f32m8(b,index);
6:    vec_c=*(vfloat32m8_t *) c;
7:    vec_c=vfmadd_vv_f32m8(vec_a,vec_b,vec_c);
8:    *(vfloat32m8_t *) c=vec_c;
9:    c+= vl;a+= vl;b+= vl;
10:   col_index+= vl;num-= vl;
11:end while
12:return y.
```

首先对列号 *col* 数组顺序寻址,如第 3 行,得到的向量值做向量标量乘法,如第 4 行,计算出索引;然后在乘数向量 *x* 中进行索引寻址,加载第 1 个向量乘数,如第 5 行;接着对数值 *val* 数组顺序寻址,加载第 2 个向量乘数,如第 6 行;最后向量元素依次相乘,如第 7 行,结果回写暂存,如第 8 行。指针偏移准备下一轮计算,如第 9、10 行。依次循环往复直到计算完成。

3.4　基于 HYB 格式的改进

HYB 格式使用 ELLPACK 格式和 COO 格式混合的方式,以更紧凑的方式存储拥有非零元素密度大的行的稀疏矩阵,提升了存储效率。图 2 给出了矩阵 ***C*** 的 HYB 格式存储结果。将第 1 行和第 2 行的元素 5 存储为 COO 格式,使所需的存储空间从 60(6*5*2)个整型大小降低到 54(6*4*2+6)个整型大小。

但是,行之间非零元素相差很大的情况不仅表现在拥有非零元素密度大的行,同时也体现在拥有非零元素密度小的行。基于这一点,本文提出改进的 HYB 格式 IHYB(Improved HYB format)。

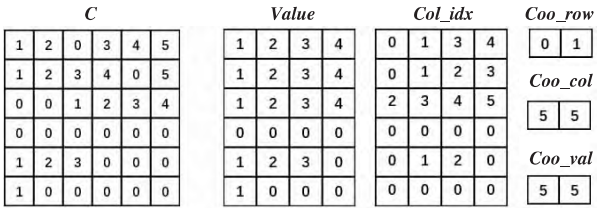

Figure 2　Storage diagram based on HYB format

图 2　基于 HYB 格式的存储示意图

IHYB 格式对非零元素密度低于 IHYB 分割阈值的行也使用 COO 格式存储。此时 ELLPACK 格式所存储元素的行号不能用自身所在的行来表示,于是引入 ***bit array*** 数组记录 ELLPACK 格式中每行元素在原始矩阵中的行号。对于 HYB 格式,在假设完全填充的 ELLPACK 格式计算速度比 COO 格式快 3 倍的情况下,通过经验法则选择数值 K 作为 ELLPACK 和 COO 之间的分割值,使得矩阵中具有 K 个以上非零元素数的行数不少于 $M/3$,其中 M 是矩阵总行数[13]。而对于 IHYB 格式,参照类似规则,阈值的确定使用迭代算法 1。

算法 1　确定 IHYB 分割阈值的算法

输入:稀疏矩阵每行非零元素数。

输出:IHYB 分割阈值。

Step 1　对稀疏矩阵每行元素数中的非零值从小到大排序,找到 2/3 分位点作为 HYB 阈值。

Step 2　将 HYB 阈值/4 作为 IHYB 分割阈值。

Step 3　去除非零元素数低于 IHYB 分割阈值的行,按照 Step 1 更新 HYB 阈值。

Step 4　将更新后的 HYB 阈值/4 作为最终的 IHYB 分割阈值。

如 HYB 使用的经验参数,IHYB 阈值中使用的参数 2/3,1/4 等,也是通过经验法则确定。阈值的两步迭代求解操作开销包括对稀疏矩阵的 2 次遍历(统计每行非零元素数时)和每行非零元素数大小的排序。矩阵遍历可以在压缩存储格式的转换过程中完成,并不增加额外开销。假设稀疏矩阵有 m 行 k 列,排序的复杂度为 $m*\log m$,一般而言不会超过矩阵元素数 $m*k$,可以忽略。因此,通常来说,阈值求解过程的开销并不会影响算法性能。

图 3 给出了图 2 中矩阵 ***C*** 的 IHYB 格式存储结果。第 6 行的元素 1 也存储为 COO 格式,同时 ***bit array*** 数组指示了矩阵 ***value*** 中每行元素的行号,使所需的存储空间从 HYB 的 54(6*4*2+6)个整型大小进一步降低到 45(4*4*2+4+9)个整型大小。

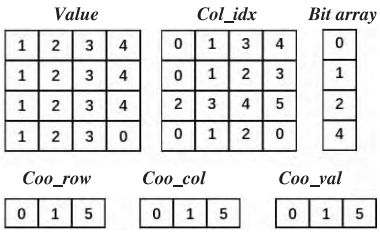

Figure 3 Storage diagram based on IHYB format

图 3 基于 IHYB 格式的存储示意图

4 算法测试与分析

本文实验使用 Spike 模拟器[18] 环境，其中 DCache 设置大小为 32 KB，ICache 大小为 32 KB，L2 Cache 大小为 128 KB，CPU 主频 1 GHz，内存使用4 GB。限定矩阵规模为 4 KB，矩阵每行非零元素个数满足正态分布，以空行率、稠密度和波动度控制稀疏矩阵非零元素分布特性，以压缩率和加速比衡量算法的空间效率和时间效率。

4.1 稠密度的影响

固定空行率为 0.3，波动度为 0，改变稠密度观察稀疏矩阵稠密度对 SpMV 算法性能的影响。随着稠密度增加，非零元素数量也增加，各算法所需存储空间增加，计算时间增多，因此，加速比减小，压缩率升高。整体上由加速比衡量的计算效率由高到低为 CSR、IHYB、HYB 和 ELLPACK，如图 4 所示。由压缩率衡量的存储效率由高到低为 IHYB、CSR、HYB 和 ELLPACK，如图 5 所示。由图 4 可知，当稠密度超过 0.08 时，CSR 的计算效率降低，IHYB 的加速比反超 CSR。原因在于 CSR 的段累加求和部分难以被向量化，当非零元素数量较多时，计算效率受到影响开始下降。得益于较高的压缩效率和较好的向量化程度，IHYB 的加速比表现优异。而 ELLPACK 中存储了大量零元素，导致整体计算效率较低。

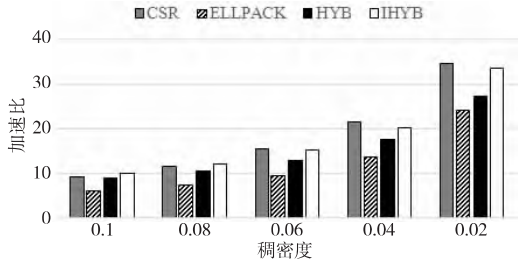

Figure 4 Effect of density on speedup

图 4 稠密度对加速比的影响

4.2 波动度的影响

固定空行率为 0.3，稠密度为 0.08，改变波动度观察稀疏矩阵波动度对 SpMV 算法性能的影响。图 6 和图 7 别显示了波动度对加速比和压缩率的影响。随着波动度增大，IHYB、HYB 和 ELLPACK 的压缩率不同程度增大，加速比不同程度减小，而 CSR 的压缩率和加速比基本保持稳定。由图 6 可知，随着波动度增大，ELLPACK 的加速比迅速降低。原因在于波动度的增大导致 ELLPACK 格式压缩率迅速升高，存储效率降低，这点由图 7 也可以看出。存储的无效零元素大量增加，进一步导致 ELLPACK 格式的计算效率下降，加速比走低。与 ELLPACK 格式不同的是，其他 3 种格式压缩率较为稳定。由图 7 可知，CSR 的压缩率最为稳定，基本不受波动度影响，HYB 和 IHYB 的压缩率随着波动度增大缓慢增加，并且 IHYB 在压缩效果上好于 HYB 格式。波动度低于 0.2 时，IHYB 的存储效率表现甚至好于 CSR。由图 6 可知，很大程度得益于此，在波动度低于0.2 时，IHYB 的计算效率也高于 CSR。

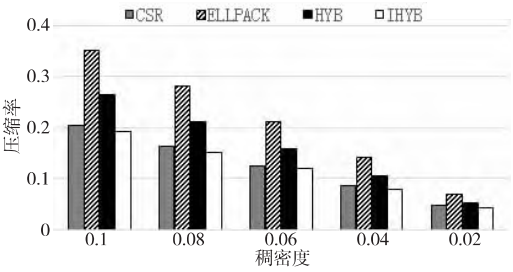

Figure 5 Effect of density on compressibility

图 5 稠密度对压缩率的影响

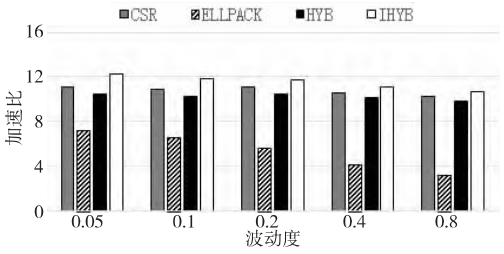

Figure 6 Effect of volatility on speedup

图 6 波动度对加速比的影响

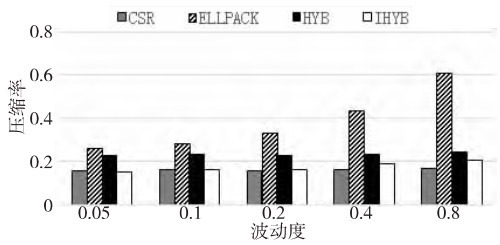

Figure 7 Effect of volatility on compressibility

图 7 波动度对压缩率的影响

图 8 和图 9 分别进行了 HYB 格式和基于 HYB 格式改进的 IHYB 在不同波动度下，加速比

和压缩率的对比。平均而言,相比 HYB 格式,IHYB 格式在加速比上提升了 13%,压缩率上降低了 6%。压缩率的降低趋势和加速比的提升趋势一致,从而体现了压缩效率的提高导致计算效率随之提高的规律。

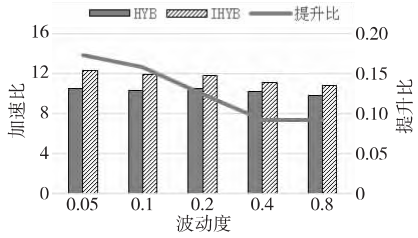

Figure 8　Comparison of speedup between HYB and IHYB

图 8　HYB 和 IHYB 的加速比对比

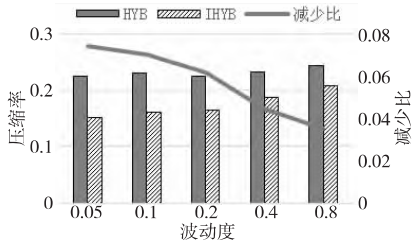

Figure 9　Comparison of compressibility between HYB and IHYB

图 9　HYB 和 IHYB 的压缩率对比

5　结束语

SpMV 算法在科学计算中占有重要地位,实现并优化基于 RISC-V 向量指令的 SpMV 算法也具有重要价值。本文首次利用 RISC-V 向量寄存器可配置性和向量寻址模式,结合 CSR、ELLPACK 和 HYB 存储格式的特点,实现了对应的 SpMV 算法。在仿真环境下比较了各 SpMV 算法面对不同非零元素分布的稀疏矩阵,表现出的计算效率和存储效率差异。同时,针对 HYB 格式,提出了更为紧凑、稳定的改进格式 IHYB。实验显示,在非零元素分布变化时,基于 IHYB 格式实现的 SpMV 表现出更为优异的存储效率和计算效率。但是,目前实现的算法都是基于单个 RISC-V 核心的。面对更为复杂的 RISC-V 多核处理器或异构计算环境,如何充分发掘算法中的并行性,提高并行度,将是一个更为复杂深刻的挑战和命题。

参考文献:

[1] Chen C, Xiang X, Liu C, et al. Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance RISC-V processor with vector extension: Industrial product[C]// Proc of 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture, 2020: 52-64.

[2] Leidel J D, Wang X, Chen Y. GoblinCore-64: A RISC-V based architecture for data intensive computing[C]// Proc of 2018 IEEE High Performance Extreme Computing Conference, 2018: 1-8.

[3] Johns M, Kazmierski T J. A minimal RISC-V vector processor for embedded systems[C]// Proc of 2020 Forum for Specification and Design Languages, 2020: 1-4.

[4] RISC-V instruction set manual volume I: Unprivileged ISA [EB/OL]. [2019-06-08]. https://riscv.org/specifications.

[5] Working draft of the proposed RISC-V V vector extension [EB/OL]. [2020-05-16]. https://github.com/riscv/riscv-v-spec.

[6] Nested-parallelism PageRank on RISC-V vector multi processor [EB/OL]. [2019-04-19]. https://digitalassets.lib.berkeley.edu/techreports/ucb/text/EECS-2019-6.pdf.

[7] Pohl A, Greese M, Cosenza B, et al. A performance analysis of vector length Agnostic code[C]// Proc of 2019 International Conference on High Performance Computing & Simulation, 2019: 159-164.

[8] Liu Fang-fang, Yang Chao, Yuan Xin-hui, et al. General SpMV implementation in many-core domestic Sunway 26010 processor[J]. Journal of Software, 2018, 29(12): 3921-3932. (in Chinese)

[9] Yang Wang-dong, Li Ken-li. Implementation and optimization of HYB based SpMV on CPU + GPU heterogeneous computing systems[J]. Computer Engineering & Science, 2016, 38(2): 202-209. (in Chinese)

[10] Li Jia-jia, Zhang Xiu-xia, Tan Guang-ming, et al. Study of choosing the optimal storage format of sparse matrix vector multiplication[J]. Journal of Computer Research and Development, 2014, 51(4): 882-894. (in Chinese)

[11] Li Yi-yuan, Xue Wei, Chen De-xun, et al. Performance optimization for sparse matrix-vector multiplication on Sunway architecture [J]. Chinese Journal of Computers, 2020, 43(6): 1010-1024. (in Chinese)

[12] Gómez C, Casas M, Mantovani F, et al. Optimizing sparse matrix-vector multiplication in NEC SX-Aurora vector engine: UPC/51.310E[R]. Barcelona: Barcelona Supercomputing Center, 2020.

[13] Bell N, Garland M. Implementing sparse matrix-vector multiplication on throughput-oriented processors[C]// Proc of Conference on High Performance Computing Networking, 2009: 1-11.

[14] Monakov A, Lokhmotov A, Avetisyan A. Automatically tuning sparse matrix-vector multiplication for GPU architectures[C]// Proc of International Conference on High-Performance Embedded Architectures and Compilers, 2010: 111-125.

[15] Kreutzer M, Hager G, Wellein G, et al. A unified sparse matrix data format for efficient general sparse matrix-vector multiply on modern processors with wide SIMD units[J].

SIAM Journal on Scientific Computing,2014,36(5):C401-C423.

[16] Liu X, Smelyanskiy M,Chow E,et al. Efficient sparse matrix-vector multiplication on x86-based many-core processors[C]// Proc of the 27th International ACM Conference on International Conference on Supercomputing,2013:273-282.

[17] Liu W F, Brian V. CSR5:An efficient storage format for cross-platform sparse matrix-vector multiplication[C]// Proc of the 29th ACM International Conference on Supercomputing,2015:339-350.

[18] Spike,a RISC-V ISA simulator[EB/OL]. [2019-04-01]. https://github.com/riscv/riscv-isa-sim.

附中文参考文献:

[8] 刘芳芳,杨超,袁欣辉,等. 面向国产申威26010众核处理器的SpMV实现与优化[J]. 软件学报,2018,29(12):3921-3932.

[9] 阳王东,李肯立. 基于HYB格式稀疏矩阵与向量乘在CPU+GPU异构系统中的实现与优化[J]. 计算机工程与科学,2016,38(2):202-209.

[10] 李佳佳,张秀霞,谭光明,等. 选择稀疏矩阵乘法最优存储格式的研究[J]. 计算机研究与发展,2014,51(4):882-894.

[11] 李亿渊,薛巍,陈德训,等. 稀疏矩阵向量乘法在申威众核架构上的性能优化[J]. 计算机学报,2020,43(6):1010-1024.

作者简介:

顾越(1996-),男,江西吉安人,硕士生,研究方向为高性能计算。**E-mail**:guyue01@foxmail.com

GU Yue, born in 1996, MS candidate, his research interest includes high performance computing.

赵银亮(1960-),男,陕西岐山人,博士,教授,研究方向为并行计算与机器学习、程序设计语言与编译优化。**E-mail**:zhaoy@xjtu.edu.cn

ZHAO Yin-liang, born in 1960, PhD, professor, his research interests include parallel computing & machine learning, programming language & compiler optimization.