计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202330204
61(12): 2969−2984, 2024

HVMS: 基于混合向量化的 SpMV 优化机制

颜志远　解壁伟　包云岗
（处理器芯片全国重点实验室（中国科学院计算技术研究所）　北京　100190）
（中国科学院大学　北京　100049）
（yanzhiyuan@ict.ac.cn）

HVMS: A Hybrid Vectorization-Optimized Mechanism of SpMV

Yan Zhiyuan, Xie Biwei, and Bao Yungang
（*State Key Lab of Processors* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190）
（*University of Chinese Academy of Sciences*, *Beijing* 100049）

Abstract　Sparse matrix-vector multiplication (SpMV) plays an important role in a wide variety of scientific and engineering applications.Executing SpMV efficiently on vector processors is challenging because of the irregular memory access of sparse matrices. We conduct a detailed analysis of the main factors that have impact on the efficiency of SpMV vectorization.In addition to the irregular distribution of non-zero elements within sparse matrices, different sparse matrices also exhibit huge variations in the distribution characteristics of non-zero elements. Therefore, it is difficult to apply a universal vector optimization method for matrices with diverse characteristics. Furthermore, there is a big difference in vector computing and vector instructions for various vector processors. The primary challenge of SpMV vectorization lies in mapping the irregular sparse matrices onto the regular vector processor. In this paper, we propose a hybrid vectorization-optimized mechanism of SpMV(HVMS). HVMS models the characteristics of vector processors and designs corresponding rules based on the abstracted basic operations to guide the regularization conversion of sparse matrices. HVMS divides the matrix into different parts according to the different characteristics. For each part, the non-zero distribution can be less irregular and then HVMS introduces corresponding optimization mechanisms to boost the vectorization efficiency of SpMV. We implement and evaluate HVMS on an Intel Xeon processor and compare it with three state-of-the-art approaches using 30 sparse matrices. Experimental results show that HVMS can achieve an average speedup of 1.60x, 1.72x, and 1.93x over CVR, SELL-C-σ, and Intel MKL, respectively.

Key words　SpMV; vectorization; sparse matrices; SIMD; multi-core system

摘　要　在科学计算和系统工程等领域，稀疏矩阵向量乘（sparse matrix-vector multiplication, SpMV）占据着极其重要的位置. 受限于矩阵稀疏性所导致的访存不规则性，向量优化一直是 SpMV 的难点. 针对此问题，进行深入分析并且总结影响 SpMV 向量化效率的主要因素. 除却稀疏矩阵内非零元分布的不规则，不同稀疏矩阵之间的非零元分布特征亦有明显不同，导致单一的向量优化策略难以适用于多种不同特征的稀疏矩阵. 另一方面，多样化向量硬件在向量特性和指令上的差异，影响了 SpMV 向量优化方法的通用性. 把不规则的稀疏矩阵映射到规则的向量硬件上进行计算，是 SpMV 向量化面临的最主要挑战. 基于此，提出一种基于混合向量化方法的 SpMV 优化机制（hybrid vectorization-optimized mechanism of SpMV, HVMS）.

收稿日期：2023-03-31；**修回日期**：2023-08-14
基金项目：国家重点研发计划项目(2022YFB4500403)；国家自然科学基金项目 (62090022)；中国科学院战略性先导科技专项 (XDA0320300)
This work was supported by the National Key Research and Development Program of China (2022YFB4500403), the National Natural Science Foundation of China (62090022), and the Strategic Priority Research Program of the Chinese Academy of Sciences (XDA0320300).
通信作者：包云岗(baoyg@ict.ac.cn)

HVMS 首先对向量硬件的特性进行抽象建模,并基于抽象出的基本操作,设计相应的规则指导稀疏矩阵进行规则化转换.按照不同的矩阵特征,HVMS 将稀疏矩阵划分为不同的部分,弱化稀疏矩阵的不规则程度,并引入不同的优化策略最大化 SpMV 的向量化效率,从而提升性能.基于 Intel Xeon 平台,在 30 个常用稀疏矩阵上对 HVMS 进行实验分析.结果表明,相比现有代表性工作如 CVR, SELL-C-σ, Intel MKL, HVMS 分别获得 1.60 倍、1.72 倍和 1.93 倍的平均加速比.

关键词 稀疏矩阵向量乘;向量优化;稀疏矩阵;SIMD;多核系统

中图法分类号 TP391

稀疏矩阵向量乘(sparse matrix-vector multiplication, SpMV)作为数值计算中的基础算子, 被广泛应用在科学计算及各类工程应用中,包括但不限于图计算、机器学习、流体力学和金融数学等. SpMV 将一个矩阵 $\boldsymbol{A}$ 和向量 $\boldsymbol{x}$ 相乘得到目标向量 $\boldsymbol{y}$($\boldsymbol{y}=\boldsymbol{Ax}$). 其中矩阵 $\boldsymbol{A}$ 是稀疏矩阵, $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 是稠密向量. 系统中的很多问题都会被抽象成 SpMV 问题,譬如迭代法求解稀疏线性方程组[1] 等. 随着矩阵维度的不断增大, SpMV 的计算成本随之增高,成为很多系统中的性能瓶颈. SpMV 的计算效率亟需提高. 另一方面,硬件架构在持续演进,多核/众核架构[2-3] 和向量处理能力(如 Intel AVX512[4] 和 ARM NEON[5])等在提升应用性能方面展示了巨大的潜力. 然而,稀疏矩阵的不规则性使 SpMV 无法有效利用这些先进的硬件特性.

SpMV 的向量优化面临诸多问题,例如:大量的短向量(单行内非零元较少)导致向量计算不饱和;向量寄存器把值写回数组 $\boldsymbol{y}$ 时,会因为数据不整齐造成写冲突问题等. 现有 SpMV 向量优化方法主要分为 3 种:逐行向量化(如 Intel MKL[6] 和 CSR5[7] 等)、多行向量化(如 CVR[8] 等)和零元填充(如 SELL-C-σ[9] 等). 实验分析表明这 3 种向量优化方法,无法在特征各异的稀疏矩阵上皆取得最优性能. 逐行向量化方法,适合处理长向量(单行内非零元较多)且访存跨度不大的稀疏矩阵;多行向量化方法,同时从多行加载元素,适合处理相邻行的非零元数量和分布模式较为接近的稀疏矩阵;零元填充作为前 2 种向量化方法的补充,难点在于填充比例和向量优化效果中间的权衡. 综合来看,这 3 种方法各有优势,可用于应对具备特定非零元分布特征的稀疏矩阵,但尚无法较好处理非零元分布较为复杂和多样化的稀疏矩阵.

另一方面,向量处理架构已经成为当前提升数据处理性能的主要方向之一. 因设计目标和服务场景各异,不同向量硬件间虽有较大的向量指令重合,但也通常会引入一些特定的向量特性和指令. 这为研究通用的 SpMV 向量优化方法带来了巨大的挑战.

故而,把极不规则的稀疏矩阵映射到极度规则的向量硬件上进行计算,需首先解决稀疏矩阵的规则化问题,以及向量硬件计算能力的通用化表示问题. 同时,为达成最优的 SpMV 性能,需在提升向量化性能的同时,不影响多线程并行执行性能.

基于此,本文提出一种混合向量化的 SpMV 优化机制(hybrid vectorization-optimized mechanism of SpMV, HVMS),以应对复杂和非零元分布较为多样化的稀疏矩阵. HVMS 首先分析向量硬件,抽象其基本操作,例如常用的规约求和操作、前向求和操作、零元填充操作和掩码运算操作,以及由基础操作组合而成的较高级向量操作,如前缀求和等. 其后, HVMS 设计多条规则,指导稀疏矩阵进行规则化转换,弱化其不规则的非零元分布特征为“规则(regular)”或“弱不规则(less irregular)”. 此处需要指出的是, HVMS 的每种规则对应一种指定特征的非零元分布模式,并可由对应的向量操作完成计算. 由此, SpMV 的向量优化问题转化为一个较规则的稀疏矩阵到一系列基础向量操作的映射问题.

最后,本文基于 Intel Xeon, 在 30 个稀疏矩阵数据集上测试 HVMS 的性能和有效性. 实验结果表明,相比 3 种代表性方案 CVR, SELL-C-σ, Intel MKL, HVMS 可分别获得 1.60 倍、1.72 倍和 1.93 倍的平均加速比. 同时,本文分析 HVMS 的可扩展性,随着线程数量从 1 增加至 64, HVMS 相比单线程 Intel MKL 的平均加速比可达 20.35 倍,分别高于其他 3 种方案 10.93 倍、11.91 倍和 12.76 倍.

1 研究背景和研究动机

1.1 基于 CSR 格式的 SpMV 算法

基于 CSR (compressed sparse row)格式的 SpMV 算法是基于 CSR 格式的 SpMV 代码实现的. 如算法 1 所示. 图 1 是稀疏矩阵 $\boldsymbol{A}$ 的 CSR 格式存储的示例描

述，开辟 3 个数组来存储稀疏矩阵：***csrVal***, ***csrColIdx***, ***csrRowPtr***. 其中数组 ***csrVal*** 存储非零元素值；数组 ***csrColIdx*** 存储非零元素的列索引；数组 ***csrRowPtr*** 存储每行第 1 个非零元素在数组 ***csrVal*** 中的偏移量. 本文使用 CSR 格式作为基本存储格式.

算法 1. 基于 CSR 格式的串行 SpMV 算法.

输入：稀疏矩阵 $\boldsymbol{A}(m\times n)$，稠密向量 $\boldsymbol{x}$；

输出：稠密向量 $\boldsymbol{y}$.

① for $i = 0 : m$ do

② $sum = 0$;

③ for $j = csrRowPtr[i] : csrRowPtr[i+1]$ do

④ $sum \mathrel{+}= csrVal[j]\times x[csrColIdx[j]]$;

⑤ end for

⑥ $y[i] = sum$;

⑦ end for

算法 1 逐行遍历稀疏矩阵 ***A***，每次计算矩阵的 1 行与数组 ***x*** 的点积（行③~⑤），并将结果写回数组 ***y***.

1.2 向量优化技术

SIMD（single instruction multiple data）[10] 即单指令多数据流，即一条指令处理多个数据. 自问世以来，SIMD 技术得到了蓬勃的发展，Intel 的 SSE 系列[11]、AVX 系列[11]、ARM 的 NEON[5] 系列以及 AMD 的 3D Now![12] 指令集是其中的代表. 从 AVX2（256 b）到 AVX512（512 b），通过扩大向量指令的宽度，Intel 处理器在数据级并行（data level parallelism, DLP）方面取得了不错的性能，AVX512 指令可同时处理 8 个双精度浮点类型或者 16 个单精度浮点类型的数据，表 1 列举了部分常用的向量指令操作.

因设计目标和服务场景各异，不同的向量硬件如 Intel AVX, AMD 3D Now!, ARM NEON, GPGPU[13] 等的向量特性和指令虽有重合，但也有较大区别. 对向量硬件进行抽象，支持通用 SpMV 向量优化方法，是很重要的.

为便于表述，本文约定向量寄存器宽度为 ω，且 ω 是指向量寄存器可存储的非零元数量，亦即如果寄存器宽度为 512 b. 若稀疏矩阵中元素为整型（int）或单精度浮点型（single float），则 ω= (512 b/32 b) =16；若稀疏矩阵中元素为双精度浮点型（double float），则 ω= (512 b/64 b) = 8. 鉴于 ω 的值对表述向量优化机制并无影响，本文为便于表述和图表展示，设定 ω=4.

1.3 研究动机

为研究通用的 SpMV 向量优化方法，本文首先分析现有 SpMV 向量优化方法的技术路线和性能表现. 图 2 描绘了 3 种常用的 SpMV 向量化策略及其代表方案：逐行向量化、多行向量化和零元填充.

1）基于 CSR 格式的逐行向量化. 代表性工作有 Intel MKL 库中的 *csrSpMV* 算子实现. 该算子对 CSR 进行自动向量优化，每次对单行内的 ω 个非零元进行自动向量化；不足 ω 个则执行标量计算.

2）同时计算多行的向量化实现. 代表性工作有 CVR 等. CVR 按序同时读入 ω 行，从每行各取一个元素加载入 ω 个 SIMD 通道中.

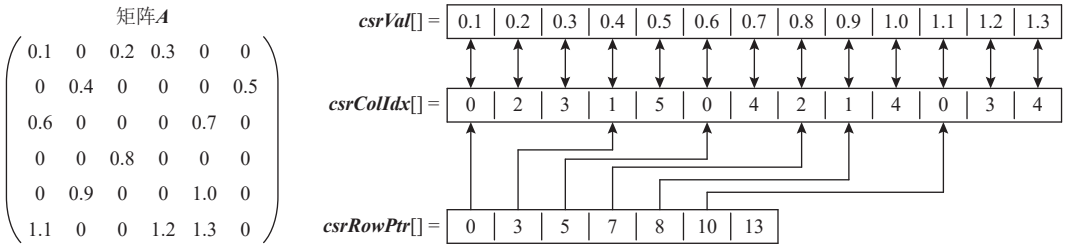


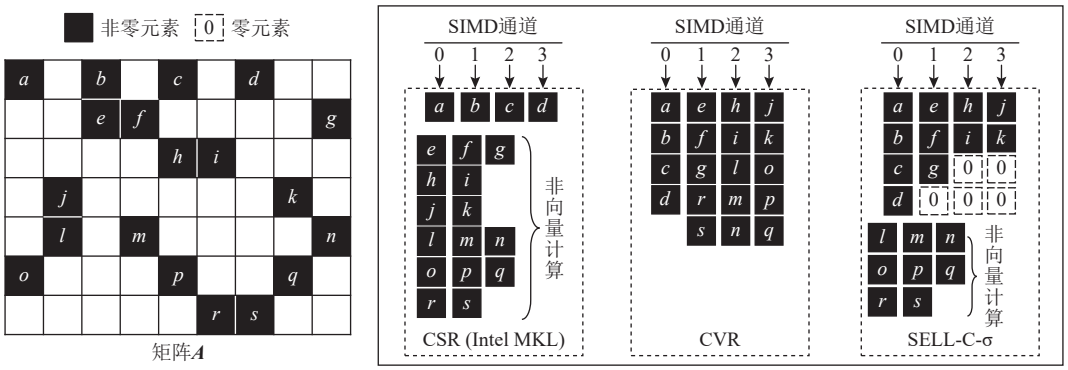
Fig. 1 Illustration of CSR format

图 1 CSR 格式示意图

Table 1 Some Common Vector Instructions

表 1 部分常用的向量指令

指令操作	指令行为说明
load(&addr)	将数据（例如双精度浮点类型的数据）从内存地址 addr 处加载到寄存器.
fmadd_pd(Reg_1, Reg_2, Reg_3)	执行向量乘加操作，即目标寄存器的值=$Reg_1\times Reg_2 + Reg_3$.
gather(*idx*, ***&array***)	使用 *idx* 寄存器提供的索引值从数组 ***array*** 中 gather 多个数据（例如 8 个 double 数据），并将其加载入寄存器中.
scatter(***&array***, *idx*, *Reg*)	使用 *idx* 寄存器提供的索引值从 *Reg* 中 scatter 多个数据（例如 8 个 double 数据），并将其存储到内存数组 ***array***.
reduction_add(*Reg*)	返回将寄存器 *Reg* 内所有值相加后的结果.
mask_reduction_add(*Mask*, *Reg*)	利用给定的掩码值（*Mask*）将寄存器 *Reg* 内相应位置的值相加并返回.



(a) SpMV向量化策略 (b) 3种代表方案

Fig. 2 A example of sparse matrix and the corresponding data layouts of Intel MKL, CVR and SELL-C-σ in the calculation

图 2 稀疏矩阵示例以及 Intel MKL，CVR，SELL-C-σ 在计算时的对应数据排布

3)基于零元填充的向量化方法. SELL-C-σ 将矩阵按照行内非零元数量进行排序，并采取零元填充的方式来达到数据对齐的效果，目的是确保同时计算的 ω 行也可同时写回.

如上 3 种 SpMV 优化方法各具特点且分别代表了一类 SpMV 优化思路. 本文选择 4 个稀疏矩阵数据集，评测以上 3 种方案的性能，如图 3 所示. Intel MKL，CVR，SELL-C-σ 这 3 个方案分别在不同的稀疏矩阵数据集上取得了最优性能，亦即没有一种方案可以在全部 4 个稀疏矩阵上都取得最好性能.

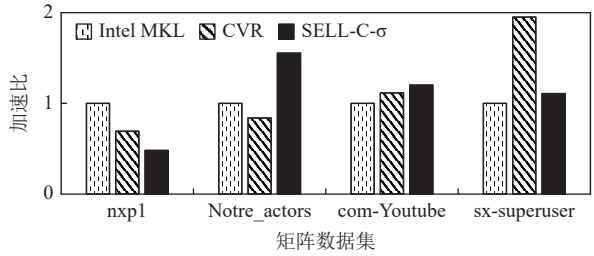


Fig. 3 Performance results of Intel MKL, CVR and SELL-C-σ on four datasets

图 3 Intel MKL，CVR，SELL-C-σ 在 4 个数据集上的性能结果

为分析其中原因，本文描绘了 4 个矩阵的非零元分布密度图，如图 4 所示，不同颜色代表不同密度，黄色为密度最高部分. 4 个稀疏矩阵的非零元分布差异极大. 因稀疏矩阵非零元分布的不规则性，以及不同稀疏矩阵非零元分布的差异性，使得 SpMV 很难借助单一优化手段在差异较大的稀疏矩阵上都取得较好性能. 一般来讲，单一优化方法如 CVR 和 SELL-C-σ 等仅对稀疏矩阵符合特定规则或矩阵中的一些行符合特定规则时较为有效.

基于如上分析，本文将研究一种 SpMV 通用向量优化机制以应对多样化的稀疏矩阵非零元分布模式和差异化的向量硬件特性等，把 SpMV 映射到规则的向量硬件上进行计算.

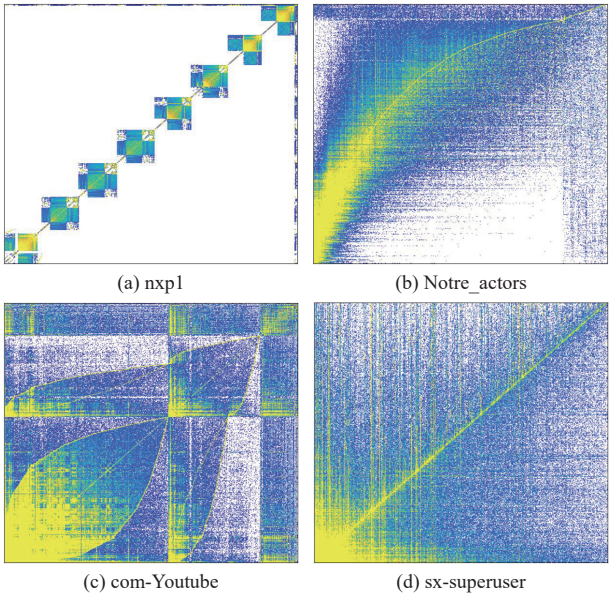


Fig. 4 Density distribution of four typical matrix datasets

图 4 4 个典型矩阵数据集的密度分布

2 HVMS 方案设计

为使用极度规则的向量化计算方法来提升极不规则的稀疏矩阵运算问题(如 SpMV)的性能，本文提出一种混合向量化的 SpMV 优化机制 HVMS，图 5 刻画了 HVMS 的设计思想，主要分为 3 部分：

1)对硬件的向量计算能力进行抽象建模，转化为规约求和操作(reduction-based operation)、前向求和操作(forward-sum operation)、零元填充操作(zero-padding operation)和掩码运算操作(mask-based operation)等基本操作. 鉴于不同的向量化硬件，如 AVX512，ARM NEON，GPGPU 等支持的向量化操作有较大重合但

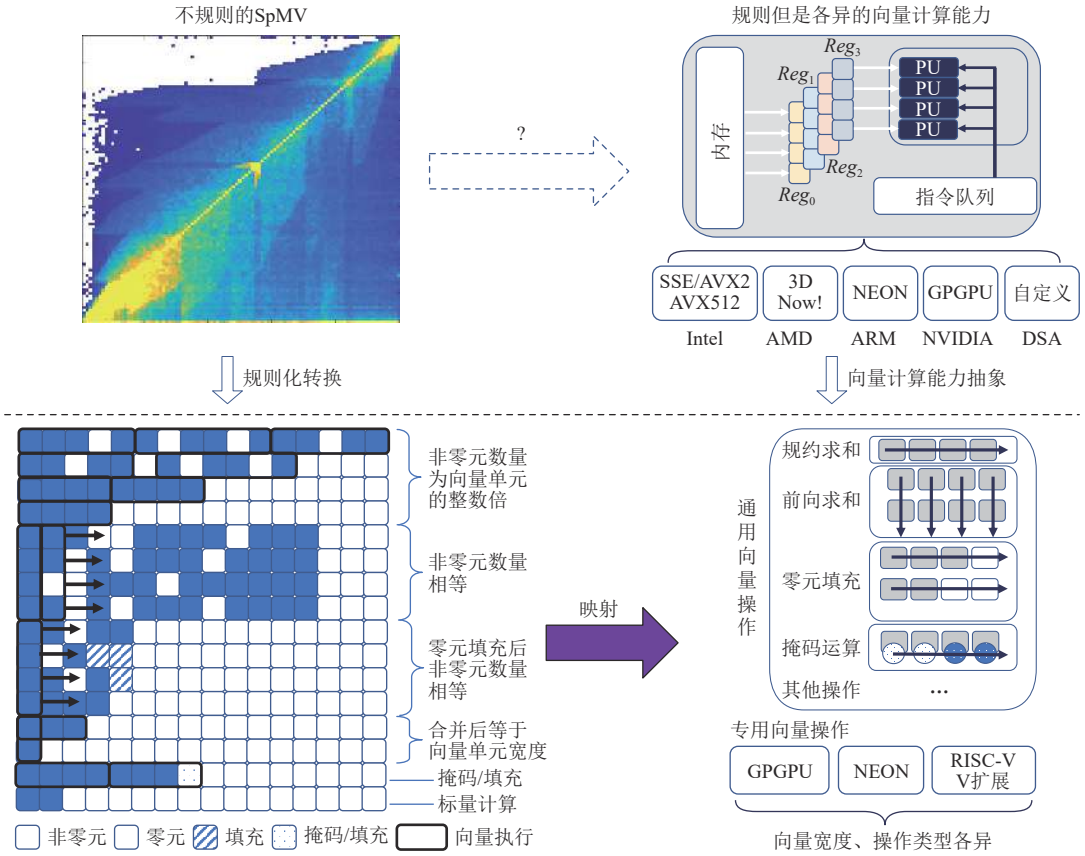


Fig. 5 The design philosophy of HVMS

图 5 HVMS 的设计思想

也各有差异，HVMS 针对不同硬件建模后的向量化操作集合是不同的. 本文主要以 AVX512 为例进行建模，但本文方法可以扩展到其他向量硬件.

2)鉴于 SpMV 计算性能受稀疏矩阵 $\boldsymbol{A}$ 的非零元分布模式的影响较大，HVMS 需着重处理稀疏矩阵 $\boldsymbol{A}$ 的不规则问题，把极度不规则的稀疏矩阵，通过统计、切块和分类等方式，进行规则化转换(regularization transformation). 亦即，结合向量硬件的特性，把不规则的稀疏矩阵分解为若干相对规则的部分.

3)把稀疏矩阵的计算(如 SpMV)，按照不同的方式映射到向量计算单元上，由此达到最优的向量化性能，最大程度利用当前先进体系架构 AVX512 等强大的向量计算能力.

区别于现有工作，HVMS 并不关注单独针对特定稀疏矩阵的非零元分布模式进行优化，而是通过抽象建模的方式，把稀疏矩阵运算按照指定规则映射到指定的向量计算特性上. 故而，其他稀疏矩阵运算的向量化方法，以及新增的向量特性或自定义指令，都可以定义为规则集成到 HVMS 中.

2.1 向量硬件的计算能力抽象

很多不同硬件的向量处理能力，例如 Intel AVX2, AVX512, ARM NEON, GPGPU 等，都可通过 SIMD 的方式提供向量处理能力从而提升性能，但在具体的特性实现上存在较大差异. 体现在硬件特性上，这些差异包括但不限于向量处理位宽，以及对 load, store, gather, scatter, reduction, mask 等指令及其变种的支持. 例如：mask_reduction 指令可以对向量单元里的指定元素进行规约操作；gather 指令可以从内存中加载非连续数据到向量寄存器；permute 和 shuffle 指令可以按照指定要求重新排布向量单元内的元素位置等. 这些指令特性可以增加设计稀疏矩阵向量优化方法的灵活性. 对向量计算单元的能力进行抽象和建模，首先需要分析指定硬件的向量指令或特性，将其中有助于实施稀疏矩阵运算的特性抽象为基本操作. 因不同的向量计算单元支持的特性各异，故而抽象后的模型也会有较大区别，例如 AVX2 仅支持 gather，AVX512 则同时支持 gather 和 scatter.

HVMS 将以 Intel AVX512 为示例硬件，对稀疏矩阵运算的过程进行建模. 该模型中将覆盖大部分通用向量计算特性，如 fmadd, gather 等及部分 Intel AVX512 的专有特性，如 mask_reduction 等. 该模型具体分为 4 个部分：

1)规约求和. 如图 6(a)所示, reduction 指令对向量寄存器内的所有元素求和, 是稀疏矩阵运算中最常用的向量操作之一. 规约的元素数量取决于向量寄存器的宽度 ω. 当稀疏矩阵以 CSR 格式进行运算时, 逐行计算并规约后写回到数组 $\boldsymbol{y}$, 是常见的向量优化方法. 规约求和一般用于处理矩阵中同一行的非零元. 当行内非零元数量是 ω 的整数倍时, 可调用 reduction 指令, 把运算结果写回数组 $\boldsymbol{y}$ 指定位置.

2)前向求和. 如图 6(b)所示, 向量寄存器读入 ω 个非零元计算后, 顺序读入另外 ω 个非零元继续计算, 每次计算的结果自动求和. 该过程通过使用 fmadd 指令完成. 前向求和在 CSR 格式的稀疏矩阵运算中通常会同时读入多行的非零元, 计算后把多个结果同时写回数组 $\boldsymbol{y}$.

3)零元填充和掩码运算. 如图 6 (c)所示, 为使得数据较为整齐, 方便进行向量计算, 通常会填充部分零元. 同样, 图 6 (d)的掩码运算在应对分支语句和数据不规则等影响向量化性能的问题上也有较大帮助. 通常, 在稀疏矩阵运算中, 每行的非零元数量并不是 ω 的整数倍, 故而零元填充和掩码运算便成为了向量优化方法的重要辅助手段.

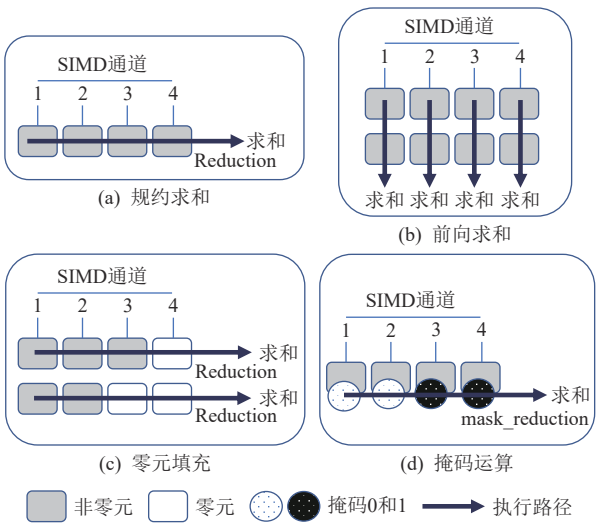


Fig. 6 Four basic operations of vector computing

图 6 4 种向量计算基本操作

HVMS 是把不规则的稀疏矩阵运算映射到规则的向量计算单元的机制和方法, 可以覆盖到各类向量计算单元, 例如 Intel SSE, AVX2/AVX512, AMD3D Now!, ARM NEON, GPGPU 以及自定义指令的向量加速器等. 本文为简化方案描述, 仅阐述部分通用基础操作的建模过程和方法. 但除此之外, HVMS 可以覆盖更为复杂且特性各异的向量化操作, 以及诸如前缀求和、CSR5 和 CVR 等高级向量化方法.

2.2 稀疏矩阵的规则化转换

稀疏矩阵极不规则的非零元分布, 使得稀疏矩阵运算的访存过程和计算过程也都极不规则. 其中, 不规则的访存过程, 亦即非连续的向量存取操作, 可以借助向量单元中的 gather 和 scatter 等操作完成. 但是, 不规则的计算过程为稀疏矩阵运算的向量优化带来了额外的阻碍. HVMS 对稀疏矩阵进行规则化转换、重新排布稀疏矩阵的非零元、弱化甚至消除稀疏矩阵运算的不规则性, 使其更加适合利用规则向量化硬件的计算能力.

如图 7 所示, 对照向量计算硬件建模后的操作类型, 以规约求和、掩码运算、前向求和以及零元填充为例, HVMS 对稀疏矩阵中满足指定要求的行进行统计和分类. 具体规则为:

1)规则 1. 对应规约求和操作. 统计非零元数量为向量位宽 ω 整数倍的行, 亦即满足规则 $nnz_{row}[i]\%\omega$

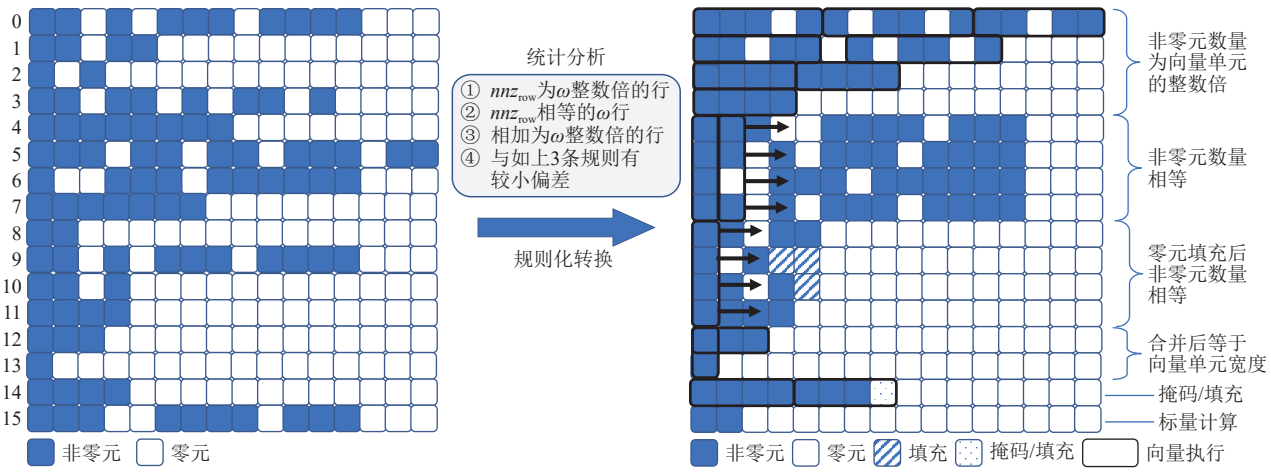


Fig. 7 Regularization transformation of sparse matrix

图 7 稀疏矩阵的规则化转换

== 0 的行，标记为使用规约求和操作计算.

2）规则 2. 对应掩码运算操作. 统计 2 行可以同时进行运算的情况，亦即满足规则 $nnz_{row}[i]+nnz_{row}[j]==\omega$ 的 2 行，标记为合并后使用掩码运算.

3）规则 3. 对应前向求和操作. 统计非零元数量相等的 ω 行，亦即满足规则 $nnz_{row}[i]==nnz_{row}[j]==nnz_{row}[k]==\cdots==nnz_{row}[l]$ 的 ω 行，标记为使用前向求和操作计算.

4）规则 4. 对应零元填充. 对于适合规约求和操作和前向求和操作但是略带偏差的行. 例如，填充比例在阈值以下时（阈值可由用户指定），可判定为零元填充，同时归入规则 1、规则 2 或规则 3.

HVMS 对稀疏矩阵进行规则化转换所遵循的规则，取决于向量化硬件建模后的操作类型，亦即由向量化硬件可以提供哪些类型的向量化方法来决定. 故而，随着 HVMS 覆盖更多向量特性和硬件类型，稀疏矩阵的转换规则也会随之丰富. 值得指出的是，HVMS 通常会设定好规则间的优先级，以解决指定行满足不止 1 条规则时的冲突问题. 同时，本文也将在实验部分分析，当规则间优先级不同可能会带来的性能差异.

2.3 稀疏矩阵运算到向量计算单元的映射

稀疏矩阵 $\boldsymbol{A}$ 被规则化并分解为多个块（block），且块内各行的非零元分布规则一致. 符合指定规则的块会被分配，由对应的向量操作来执行. 具体如图 8 所示，0~3 行属于单行非零元数量为 ω 倍数的行，交由规约求和操作计算；4~7 行每行的非零元数量相等，故可同时读入多行非零元，交由前向求和操作计算；8~11 行经零元填充后交由前向求和操作计算；12 行经零元填充后交由规约求和操作计算；13 和 14 行合并后元素数量为 ω，可使用掩码进行计算；15 行只有 2 个非零元素，可执行标量计算，无需映射到向量计算单元.

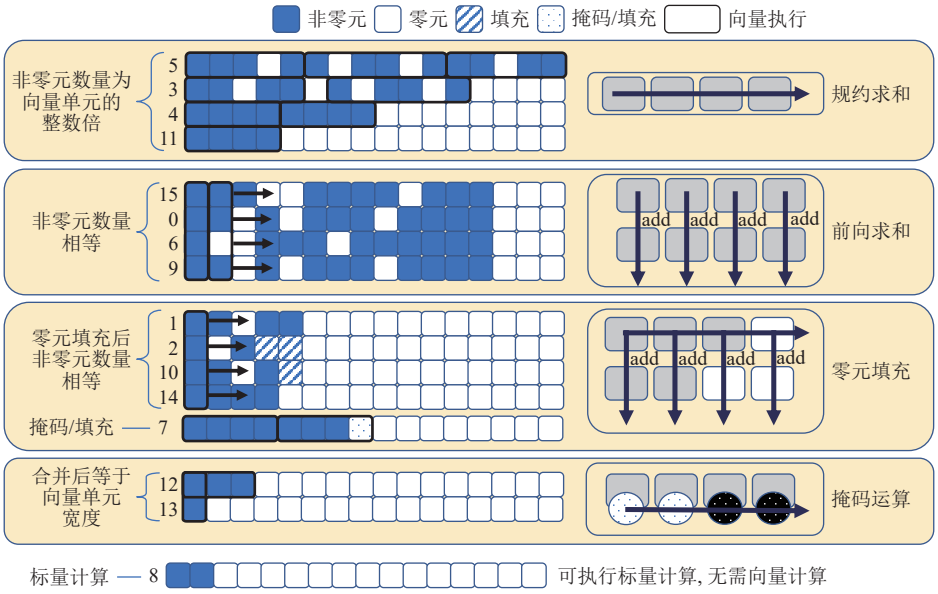


Fig. 8 Converted sparse matrix maps to the basic operations of vector computing

图 8 转换后的稀疏矩阵到基本向量计算操作的映射

该映射关系并不是固定的，例如 8~11 行在进行零元填充后，同时满足规约求和及前向求和 2 个操作的要求. 此时，这 4 行的非零元分布以及执行指令的延迟等因素都可能会影响性能. 通常在此种情况下，HVMS 采用默认优先级顺序执行，即规约求和→掩码运算→前向求和→CSR 计算.

3 HVMS 的实现及其 SpMV 计算方法

HVMS 把非规则的 SpMV 运算映射到规则的向量硬件，其方法可覆盖到非零元分布各异的稀疏矩阵，以及不同特性的向量硬件. 本节描述 HVMS 的实现细节，首先介绍稀疏矩阵的规则化数据转换过程，然后阐述基于 HVMS 进行 SpMV 计算的过程.

3.1 稀疏矩阵的数据转换

HVMS 的输入矩阵可以是任意格式. 本文为便于表述，选择以 CSR 格式为例描述稀疏矩阵到 HVMS 格式的转换. 如图 9 所示，经 HVMS 转换后的稀疏矩阵存储在 4 个辅助数组中：数组 ***value*** 存储非零元的值；数组 ***col_idx*** 存储非零元的列索引；数组 ***row_idx*** 存储对应非零元的行索引；数组 ***offset*** 存储偏移量，亦即每执行一次向量计算基本操作，从内存加载的

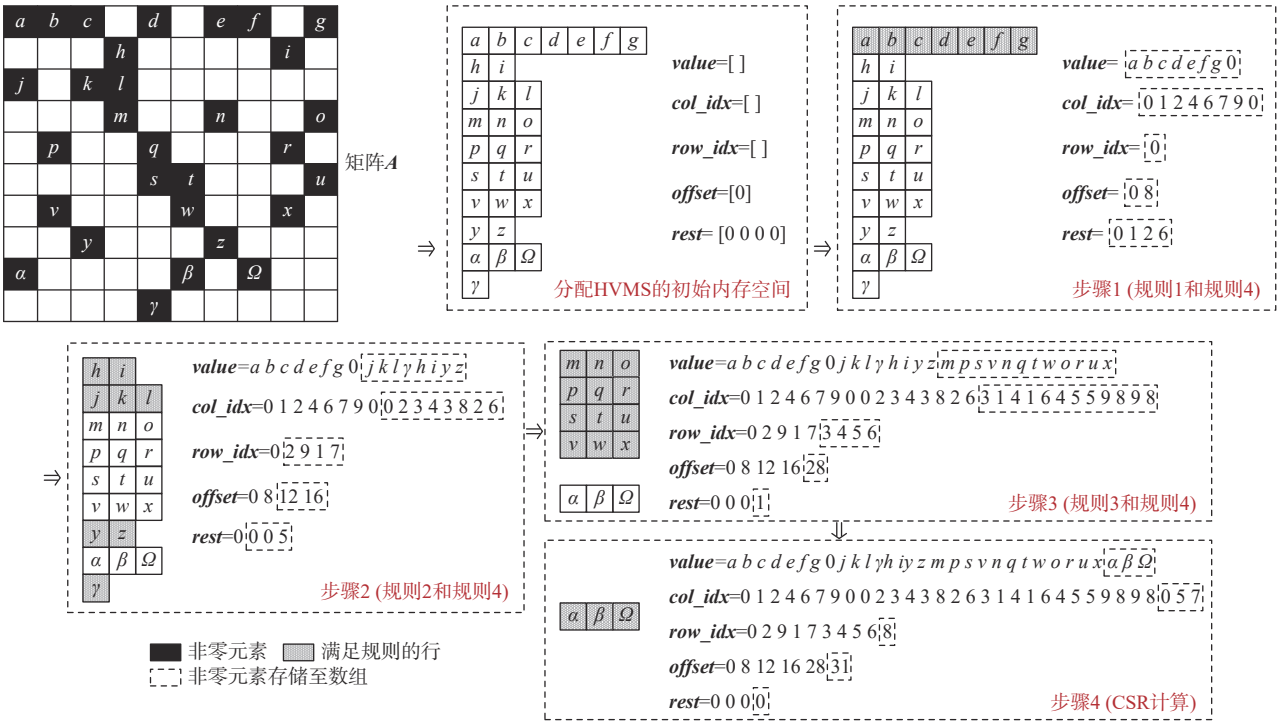


Fig. 9 Data conversion process of HVMS

图 9 HVMS 的数据转换过程

元素值在 ***value*** 中的偏移量. 除此之外, HVMS 引入数组 ***rest***, 用于统计非零元数量少于 ω 的行的数量. 数组 ***rest*** 中共有 ω 个元素, 且 $rest[i]=k$ 表示非零元数量为 i 的行共有 k 个.

整体来看, HVMS 的数据转换过程可分为 4 个步骤, 在此之前需要初始化上述 4 个数组. 算法 2 是数据转化过程的伪代码实现, 接下来本文以图 9 为例, 分别对每一步骤做一个简单的介绍.

1)步骤 1. 遍历稀疏矩阵 ***A***, 找出所有满足规则 1 的行. 本文定义 $nnz_{\text{row}}[i] \geqslant \omega$ 和 $\omega - nnz_{\text{row}}[i]\%\omega \leqslant \delta$ 的所有行执行规约求和操作, 其中 δ 的值可由用户指定. 本文默认 $\delta=\omega$.

图 9 中, 稀疏矩阵 ***A*** 的第 1 行满足规则 1. HVMS 将该行的非零元值、行索引以及非零元值的列索引依次存储至数组 ***value***, ***row_idx***, ***col_idx***. 数组 ***offset*** 存储该行首个非零元在数组 ***value*** 中的偏移量. 同时, 在遍历稀疏矩阵 ***A*** 的过程中, ***rest*** 的值也可同步统计出来.

2)步骤 2. 利用辅助数组 ***rest***, 找出所有满足规则 2 的行. 本文定义 $nnz_{\text{row}}[i]+nnz_{\text{row}}[j] \geqslant \theta$ 和 $nnz_{\text{row}}[i]+nnz_{\text{row}}[j] \leqslant \omega$ 的所有行执行掩码运算操作, 其中 θ 的值可由用户指定. 本文默认 $\theta=\omega$. 该步骤需要借助数组 ***rest*** 完成.

图 9 中, 矩阵 ***A*** 的第 2 行和第 9 行的非零元数量分别为 3 和 1, 相加之后满足上述定义, 因此将上述 2 行的数据信息顺序存储至 4 个数组中. 此处需要注意的是, 数组 ***offset*** 存储的是 2 行非零元数量相加之后的偏移量, 并非单行非零元数量的偏移量. 最后, 数组 ***rest*** 也被更新, 例如在本例中, 非零元数量为 3 的行已被标记掩码运算操作, 因此 $rest[3]=rest[3]-1$. 同理, 第 1 行和第 7 行亦符合规则 2, 因此操作同上.

3)步骤 3. 遍历矩阵 ***A*** 的剩余行, 找出所有满足规则 3 的行. 本文定义 2 点来选择满足要求的行:

1)如果 $nnz_{\text{row}}[i] == nnz_{\text{row}}[j] == \cdots == nnz_{\text{row}}[l]$, 并且行数相加等于 ω, 定义 ω 行执行前向求和操作;

2)如果行数相加不足 ω, 那么继续遍历. 当第 k 行满足

$$\frac{|nnz_{\text{row}}[i]-nnz_{\text{row}}[k]|}{nnz_{\text{row}}[i]} \leqslant \epsilon,$$

行数加 1, 直至行数相加等于 ω. 最后标记 ω 行执行前向求和操作. 其中 ϵ 的值可由用户指定, 本文默认 ϵ 值等于 0.

算法 2. 数据转换过程的伪代码实现.

输入: 稀疏矩阵 $A(m\times n)$;

输出: ***value***[], ***col_idx***[], ***row_idx***[], ***offset***[].

① for($i=0$; $i<m$; i++)

② if($nnz_{\text{row}}[i] \geqslant \omega$ &($\omega - nnz_{\text{row}}[i]\%\omega$)$<\delta$)

③ *update*(***value***, ***col_idx***, ***row_idx***, ***offset***);

④ end if
⑤ end for
⑥ for($i = 0$; $i < m$; i++)
⑦ if (($nnz_{row}[i]+nnz_{row}[j]$) $\geqslant\theta$ &
⑧ ($nnz_{row}[i]+nnz_{row}[j]$) $\leqslant\omega$)
⑨ *update*(***value***, ***col_idx***, ***row_idx***, ***offset***);
⑩ end if
⑪ end for
⑫ for($i = 0$; $i < m$; i++)
⑬ if($nnz_{row}[i] == nnz_{row}[k]$/
$|nnz_{row}[i] - nnz_{row}[k]| / nnz_{row}[i] \leqslant \epsilon$)
⑭ *row_num*++;
⑮ end if
⑯ if (*row_num* == ω)
⑰ *update*(***value***, ***col_idx***, ***row_idx***, ***offset***);
⑱ *row_num* = 0;
⑲ end if
⑳ end for

图 9 中，矩阵 ***A*** 的第 3, 4, 5, 6 行满足要求. 因此将 4 行的数据信息分别存储至 4 个数组，图 9 中的示例比较特殊，4 行是连续的，不过本文对 ω 行是否连续没有硬性要求. 最后，*rest*[3] 的值也要减 4.

对于稀疏矩阵来说，严格满足规则 1~3 的行在矩阵中的比例并不算大. 本文提出参数化的方式(δ, θ, ϵ)扩充该比例. 通过追加规则 4，即零元填充的方式，使略有偏差的行能够满足要求.

4)步骤 4. 剩余的所有行将执行 CSR 计算. 经过步骤 1~3 的筛选后，矩阵 ***A*** 的非零元数量已经极少，本文在 30 个数据集上的结果表明数量平均占比只有 0.06%，因此对该部分的计算不再进行任何优化.

3.2 HVMS 的 SpMV 实现

本节以图 10 为例，介绍 HVMS 的向量计算流程. 此时，输入的稀疏矩阵已经完成规则化转换. 本文设定 ω=4，亦即图 10 中的向量单元可同时处理 4 个双精度浮点数据. 从整体上看，基于 HVMS 的 SpMV 计算流程分为 3 个步骤：初始化、向量计算以及数组写回. 这 3 个步骤结合图 10 分别进行介绍.

1)初始化阶段. 向量寄存器 Reg_1, Reg_2, Reg_3, Reg_4 顺序执行规约求和、掩码运算以及前向求和操作. 其中规约求和以及前向求和包含累加计算，因此 Reg_1 和 Reg_4 在初始化阶段需被初始化为 0.

2)向量计算. HVMS 从数组 ***col_idx*** 中取得索引值，并使用 gather 指令从数组 ***x*** 中读取多个非连续的值；使用 load 指令将矩阵 ***A*** 中非零元的值加载入向量寄存器；使用 fmadd 指令实现乘加操作；使用 reduction_add 指令将向量寄存器内的数据进行累加，其结果即可写回数组 ***y*** 的对应位置；使用 mask_reduction_add 指令对向量寄存器中对应数据求和，并将结果写回数组 ***y*** 的对应位置. 其中，前 4 条指令分别对应规约求和、前向求和以及掩码运算操作，第 5 条指令对应掩码运算操作.

如图 10 所示，HVMS 首先将步骤 1 中转换出的数据(对应规则 1 和规则 4)载入 Reg_1，使用 fmadd 指令跟数组 ***x*** 中的对应数据做乘加运算，并借助 reduction_add 指令将结果值相加. 此后，HVMS 将步骤 2 中转换出的数据(对应规则 2)载入 Reg_2 和 Reg_3，与数组 ***x*** 中的对应数据做乘法运算后，借助 mask_reduction_add 指令将对应的结果值相加. 最后，HVMS 把步骤 3 中

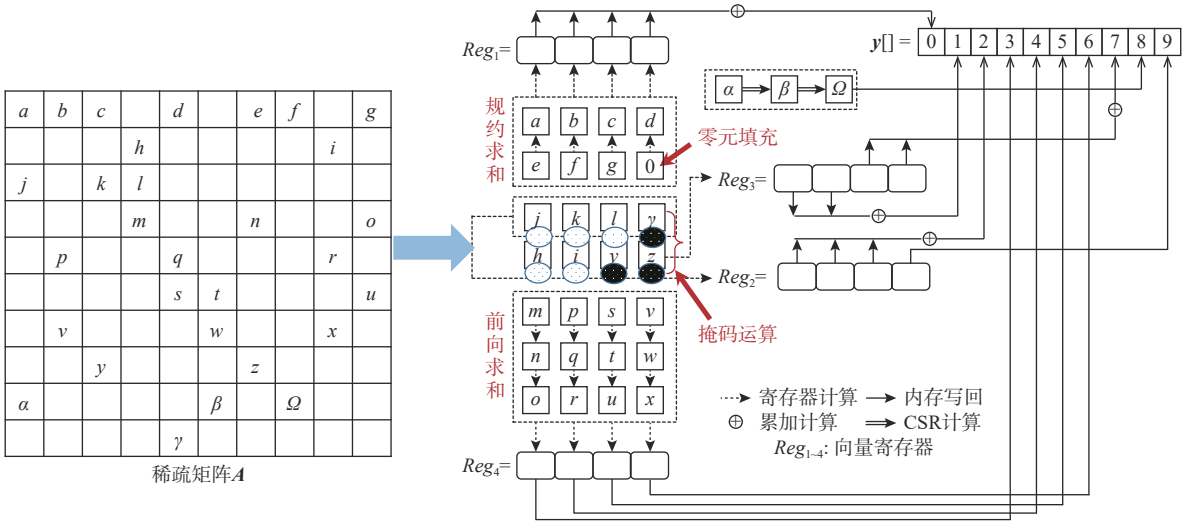


Fig. 10 SpMV implementation of HVMS
图 10 HVMS 的 SpMV 实现

转换出的数据(对应规则 3)载入 Reg_4, 并使用 fmadd 指令做乘加运算.

3)数组写回. 前向求和操作可同时处理 ω 行数据, 从而同时写回 ω 个 $\boldsymbol{y}$ 值. 在此种情况下, 因 ω 行未必连续, 故 HVMS 会调用 scatter 指令完成数据的间接内存写回(以 ***row_idx*** 为其索引值). 使用其他操作计算的数据, 如 Reg_1, Reg_2, Reg_3, 则在相加后使用 store 或 vstore 指令分别写回数组 $\boldsymbol{y}$ 的对应位置.

4 实 验

4.1 实验配置

1)实验平台. 基于 Intel Xeon(Gold 6130, Skylake)和 Intel Core(i9-10940X, Cascade Lake)开展实验对比. 其中 Intel Xeon 平台包含 16 个处理器核, 最多可开启 64 个线程, 支持 AVX512 向量指令集, 内存为 126 GB; Intel Core 平台包含 14 个处理器核, 最多可开启 28 个线程, 支持 AVX512 向量指令集, 内存为 62 GB. 表 2 列出本文实验平台的详细参数.

Table 2 Configuration Information of Experimental Platforms

表 2 实验平台的配置信息

参数	Intel Xeon	Intel Core
处理器	Gold 6130 (Skylake)	i9-10940X (Cascade Lake)
处理器个数	2	1
核心数	16	14
每核线程数	2	2
L1 cache 大小/KB	32(I)+32 (D)	32 (I)+32 (D)
L2 cache 大小/MB	1	1
L3 cache 大小/MB	22	19
内存大小/GB	126	62
操作系统	Ubuntu18.04.5	Ubuntu18.04.3

注: I 表示指令缓存, D 表示数据缓存.

2)数据集. 本文从稀疏矩阵集合[14]中选取 30 个有代表性的稀疏矩阵, 该矩阵覆盖电路仿真、社交网络图、高性能计算等多个领域, 涵盖不同的矩阵规模和非零元分布模式等. 详细信息如表 3 所示.

3)实验对比. 本文选取 3 个现有的 SpMV 工作进行实验对比: CVR, SELL-C-σ, Intel MKL(部分现有工作存在编译错误或运行时间过长等问题, 本文不予对比. 例如: SpV8[15]运行时间过长; CSR5[7]无法在 AVX512 上成功运行等). 选取的 SpMV 具体情况为:

① Intel MKL[6]是由 Intel 开发的基于 x86 平台的基础数学库, 提供了绝大部分的线性/非线性算子函数, 在众多领域有较广泛的应用.

② CVR[8]是较具代表性的多行同时计算(前向求和)向量优化方法, 经常用于 SpMV 实验对比.

③ SELL-C-σ[9]采用零元填充进行向量优化. 本文按照其方案建议, 选择 C=8(C 是向量寄存器的位宽), 同时调整 σ 的值, 使用最优结果进行对比.

本文所有实验使用双精度浮点类型, HVMS 及对比工作均使用“icc -O3 -xCORE-AVX512 -fopenmp”进行编译, 程序在运行过程中开启 64(28)个线程. 同时, 为准确获取各方案的 SpMV 执行时间, 本文迭代运行 10 000 次并使用平均时间.

Table 3 Datasets Used for Evaluation

表 3 评测数据集

数据集	维度	非零元个数	平均每行非零元个数
cage15	$5.1\times10^6\times5.1\times10^6$	99×10^6	19
caidaRouterLevel	$192\times10^3\times192\times10^3$	1.2×10^6	6
c-big	$345\times10^3\times345\times10^3$	2.3×10^6	6
circuit5M	$5.5\times10^6\times5.5\times10^6$	59×10^6	10
citationCiteseer	$268\times10^3\times268\times10^3$	2.3×10^6	8
coAuthorsDBLP	$299\times10^3\times299\times10^3$	1.9×10^6	6
com-DBLP	$317\times10^3\times317\times10^3$	2.0×10^6	6
com-Orkut	$3\times10^6\times3\times10^6$	234×10^6	76
com-Youtube	$1.1\times10^6\times1.1\times10^6$	5.9×10^6	5
dblp-2010	$326\times10^3\times326\times10^3$	1.6×10^6	4
flickr	$820\times10^3\times820\times10^3$	9.8×10^6	11
wiki-topcats	$1\times10^6\times1\times10^6$	28×10^6	15
higgs-twitter	$456\times10^3\times456\times10^3$	14×10^6	32
in-2004	$1\times10^6\times1\times10^6$	16×10^6	12
language	$399\times10^3\times399\times10^3$	1.2×10^6	3
loc-Gowalla	$196\times10^3\times196\times10^3$	1.9×10^6	9
mip1	$66\times10^3\times66\times10^3$	10×10^6	155
mouse_gene	$45\times10^3\times45\times10^3$	28×10^6	642
NotreDame_actors	$392\times10^3\times127\times10^3$	1.5×10^6	3
nxp1	$414\times10^3\times414\times10^3$	2.6×10^6	6
preferential	$100\times10^3\times100\times10^3$	1×10^6	9
rail4284	$4\times10^3\times1\times10^6$	11×10^6	2 633
road_central	$14\times10^6\times14\times10^6$	33×10^6	2
pwtk	$217\times10^3\times217\times10^3$	11×10^6	52
soc-Pokec	$1.6\times10^6\times1.6\times10^6$	30×10^6	18
soc-sign-epinions	$131\times10^3\times131\times10^3$	0.8×10^6	6
sx-superuser	$194\times10^3\times194\times10^3$	0.9×10^6	4
webbase-1M	$1\times10^6\times1\times10^6$	3.1×10^6	3
ldoor	$952\times10^3\times952\times10^3$	42×10^6	44
CurlCurl_2	$806\times10^3\times806\times10^3$	8×10^6	11

4.2 SpMV 性能结果分析

首先，本文对比 Intel MKL, CVR, SELL-C-σ, HVMS 在 2 个平台上的 SpMV 运行时间. 图 11 描绘了 30 个数据集的实验对比结果. 其中，Intel MKL 的时间被标准化为 1. 纵坐标表示 CVR, SELL-C-σ, HVMS 相对于 Intel MKL 的加速比. 从图 11 可知，在 Intel Xeon 平台上，HVMS 相比 Intel MKL 的加速比最高可达 5.16 倍，在 Intel Core 平台上则为 3.16 倍. 同时，在 30 个数据集上，HVMS 在 Intel Xeon 平台上分别获得了 1.60 倍（CVR）、1.72 倍（SELL-C-σ）和 1.93 倍（Intel MKL）的平均加速比；在 Intel Core 平台上分别获得了 1.02 倍（CVR）、1.42 倍（SELL-C-σ）和 1.39 倍（Intel MKL）的平均加速比.

由图 11 可知，HVMS 在 Intel Xeon 平台上的性能结果优于 Intel Core 平台，该结果与 2 个服务器核的 CPU 架构、内存系统配置、算法本身特性等多方面因素相关. 整体来看，HVMS 在高性能计算场景下具备更好的适用性，该优势跟 SpMV 的应用领域重合度较高，例如深度学习、数值模拟等. 并且在通用场景下，HVMS 可取得不低于其他方案的性能.

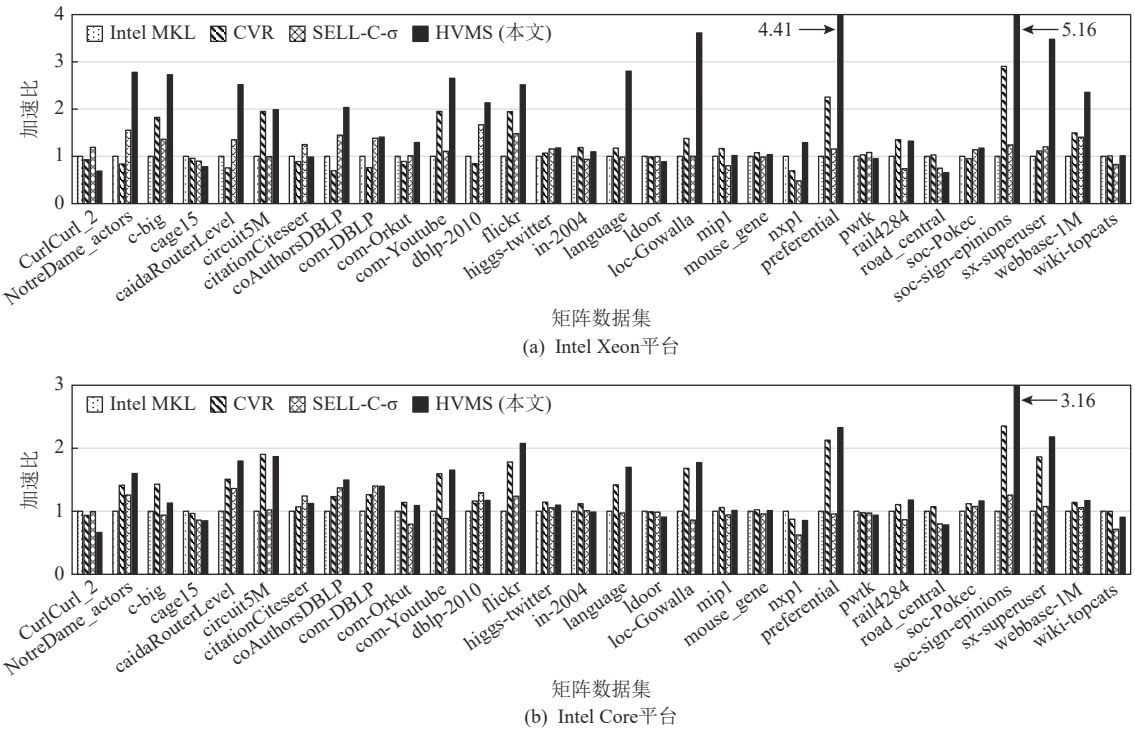


Fig. 11 Experimental performance comparison of four schemes on two platforms

图 11 4 种方案在 2 个平台上的实验性能对比

具体分析来看，HVMS 在大部分数据集上的性能都是最优的，且在 soc-sign-epinions 和 preferential 上表现较为突出. 这 2 个数据集的特点是：行与行之间的非零元数量相差很大，非零元分布极不规则. HVMS 机制首先对稀疏矩阵进行规则化转换，并使用不同的向量操作处理不同特征的行，从而缓解非规则性对向量优化的影响. 而在 com-DBLP, in-2004, ldoor, mip1 等矩阵上，HVMS 的性能表现较为普通. 分析这些矩阵可知，它们的非零元分布相对比较规则，使用单一规则即可获得较好性能.

另一方面，HVMS 在对稀疏矩阵进行规则化转换的过程中，也会影响一部分行的计算顺序，从而影响稀疏矩阵 $\boldsymbol{A}$ 和数组 $\boldsymbol{x}$ 内元素的访问顺序，并一定程度上影响访存局部性. 虽然在绝大部分稀疏矩阵上，HVMS 的规则化转换是有益的. 但是对于部分稀疏矩阵，例如 CurlCurl_2 和 road_central 上，也会因影响访存局部性而使性能有轻微下降.

4.3 预处理开销分析

为提升 SpMV 的性能，不少研究工作提出比 CSR 更为复杂的数据存储格式，故而格式转换过程（也被称之为预处理过程）亦备受关注，它对程序的整体性能会产生一定的影响. 本节利用指标 A_{pre} 来分析预处理开销所带来的性能损失，其值表示平摊预处理开销所需的 SpMV 迭代次数. 计算公式为：

$$A_{\text{pre}} = \frac{T_{\text{pre}}}{T_{\text{SpMV}}^{\text{MKL}} - T_{\text{SpMV}}^{\text{new}}}, \tag{1}$$

其中T_{pre}表示预处理时间，T_{SpMV}^{MKL}表示 Intel MKL 的 SpMV 算子执行时间，T_{SpMV}^{new}表示对比方案的 SpMV 执行时间(本文中的对比方案为 SELL-C-σ, CVR, HVMS). 本文指定 Intel MKL 的 SpMV 执行时间作为衡量其他方案的基准数据. 故而，若 Intel MKL 优于对比方案，此时A_{pre}的值被置为∞，表示对比方案的执行时间大于 Intel MKL 的执行时间. A_{pre}的值越小，表示平摊预处理开销所需的 SpMV 迭代次数越少. 因篇幅受限，本文仅列举 Intel Xeon 平台上，CVR, SELL-C-σ, HVMS 在 30 个数据集上的A_{pre}值，如表 4 所示.

Table 4 SpMV Iterations that Need to Amortize the Format Conversion Overhead on Three Schemes

表 4 3 种方案平摊格式转换开销所需的 SpMV 迭代次数

数据集	CVR	SELL-C-σ	HVMS (本文)
cage15	∞	∞	∞
caidaRouterLevel	∞	161.47	**4.16**
c-big	**1.73**	105.80	3.29
circuit5M	**2.66**	∞	3.56
citationCiteseer	∞	172.75	**16.08**
coAuthorsDBLP	∞	88.11	**4.01**
com-DBLP	∞	119.02	**6.60**
com-Orkut	∞	474.72	**7.35**
com-Youtube	**2.06**	96.67	4.09
dblp-2010	∞	133.69	**7.50**
flickr	**2.53**	21.63	4.09
wiki-topcats	369.29	∞	**78.67**
higgs-twitter	51.17	71.15	**46.99**
in-2004	**19.28**	∞	56.19
language	8.14	∞	**3.73**
loc-Gowalla	3.50	11 015.26	**1.98**
mip1	**32.85**	∞	115.37
mouse_gene	**60.95**	∞	133.37
NotreDame_actors	∞	89.13	**4.62**
nxp1	∞	∞	**28.12**
preferential	**1.22**	215.17	1.93
rail4284	**12.25**	∞	24.19
road_central	**28.39**	∞	∞
pwtk	**153.19**	186.86	∞
soc-Pokec	∞	84.20	19.49
soc-sign-epinions	**1.66**	137.77	2.38
sx-superuser	**11.00**	193.27	2.95
webbase-1M	**4.61**	98.82	6.49
ldoor	∞	∞	∞
CurlCurl_2	∞	118.43	∞

注：加粗数字表示 3 种方案中的最少迭代次数，亦即最优方案.

由表 4 可知，SELL-C-σ 的预处理开销较大，需更多的迭代次数平摊开销，如在 2 个平台上的最大迭代次数分别为 11 016 和 315，平均迭代次数分别为 715 和 102. 而 HVMS 在所有数据集上的平摊迭代次数均在可接受范围之内，如在 2 个平台上的最大迭代次数分别为 134 和 221，平均迭代次数分别为 24 和 28，CVR 亦能获得不错的效果，如在 2 个平台上的最大迭代次数分别为 370 和 135，平均迭代次数分别为 43 和 17.

4.4 整体性能分析

真实应用场景下，SpMV 过程通常会被迭代计算多次，譬如迭代法求解稀疏线性方程组，该过程将会被计算成千上万次以达到结果值收敛的目的. 而预处理过程仅需执行 1 次，故而为做更准确的性能评测，本文考虑迭代场景下的程序执行，包含 1 次预处理过程以及多次 SpMV 计算过程. 本文将 Intel MKL 的 SpMV 算子作为基准测试程序，与其相比，本文其他方案的性能加速比计算公式为：

$$Speedup=\frac{nT_{SpMV}^{MKL}}{T_{pre}+nT_{SpMV}^{new}},\tag{2}$$

其中 n 是迭代计算的次数，nT_{SpMV}^{MKL}表示 n 次迭代后 Intel MKL 的 SpMV 总执行时间，T_{pre}表示预处理时间，nT_{SpMV}^{new}表示 n 次迭代后对比方案的 SpMV 总执行时间.

图 12 展示了随着迭代次数的增加，相比于 Intel MKL，其他 3 种方案在 30 个数据集上的平均加速比. 可以看出，在 2 个平台上 HVMS 均能达到最优加速比，故而其在真实应用场景下具有较好的适用性.

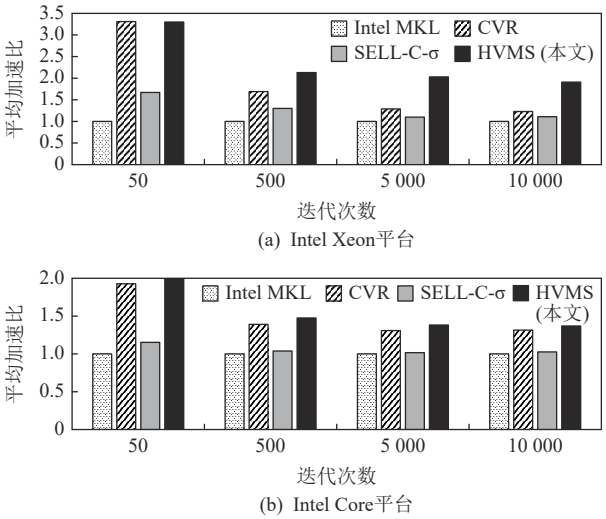


Fig. 12 Average speedups of four shemes compared with Intel MKL after the iteration numbers change

图 12 更改迭代次数后 4 种方案相比于 Intel MKL 的平均加速比

4.5 可扩展性分析

本文对 4 种 SpMV 方案在 30 个稀疏矩阵上的扩展性进行对比分析. 由实验结果可知, 随着线程数目的增加, HVMS 获得了较好的性能加速比, 具备较好的可扩展性. 在 64 个线程上, 较之其他 3 种方案的平均加速比(Intel MKL 为 10.93 倍, CVR 为 11.91 倍, SELL-C-σ 为 12.76 倍), HVMS 可获得平均 20.35 倍的性能提升. 在 28 个线程上, 较之其他 3 种方案的平均加速比(Intel MKL 为 5.17 倍, CVR 为 6.92 倍, SELL-C-σ 为 5.35 倍), HVMS 可获得平均 7.43 倍的性能提升. 此外, 因篇幅受限, 本文选取 4 个有代表性的稀疏矩阵, 图 13 刻画了随着线程数目的增加, 4 个矩阵在不同优化方案下的性能变化趋势. 由图可知, HVMS 的可扩展性优于其他方案, 最高可分别获得 41 倍(Intel Xeon 平台)和 18 倍(Intel Core 平台)的性能提升.

4.6 规则优先级对 HVMS 性能的影响

HVMS 中默认的规则优先级为: 规约求和→掩码运算→前向求和→零元填充/掩码运算→标量计算. 本文更改规则的优先级, 并分析其对性能的影响.

通过对前 3 种基本操作排列组合, 可得到 6 种不同的优先级组合. 图 14 展示了在 2 种平台上、不同的优先级组合下, HVMS 相比 Intel MKL 的性能提升. “增量加速比”指 6 种组合中最优结果相比默认优先级的性能提升. 其中, 几乎所有稀疏矩阵都在不同的规则优先级下获得了不同程度的性能增长. 但大多数矩阵调整规则优先级后的性能增长较小, 仅部分矩阵, 例如在 Intel Xeon 平台上, c-big 和 preferential 获得了较大幅度增长, 分别为 0.83 和 1.19. 原因在于这 2 个稀疏矩阵的行内非零元数量偏少, 且具有相同非零元个数的行数较多, 故而更适合基于前向求和的操作进行计算. HVMS 倾向使用默认的规则优先级来获得较好性能, 而非自动调优, 以免引入额外的成本影响性能优化效果.

5 相关工作

针对 SpMV 的优化, 已有数十年的研究历史. 研究人员致力于在各个平台、各个领域, 通过改进现有的存储格式、挖掘硬件特性、设计自动调优框架等

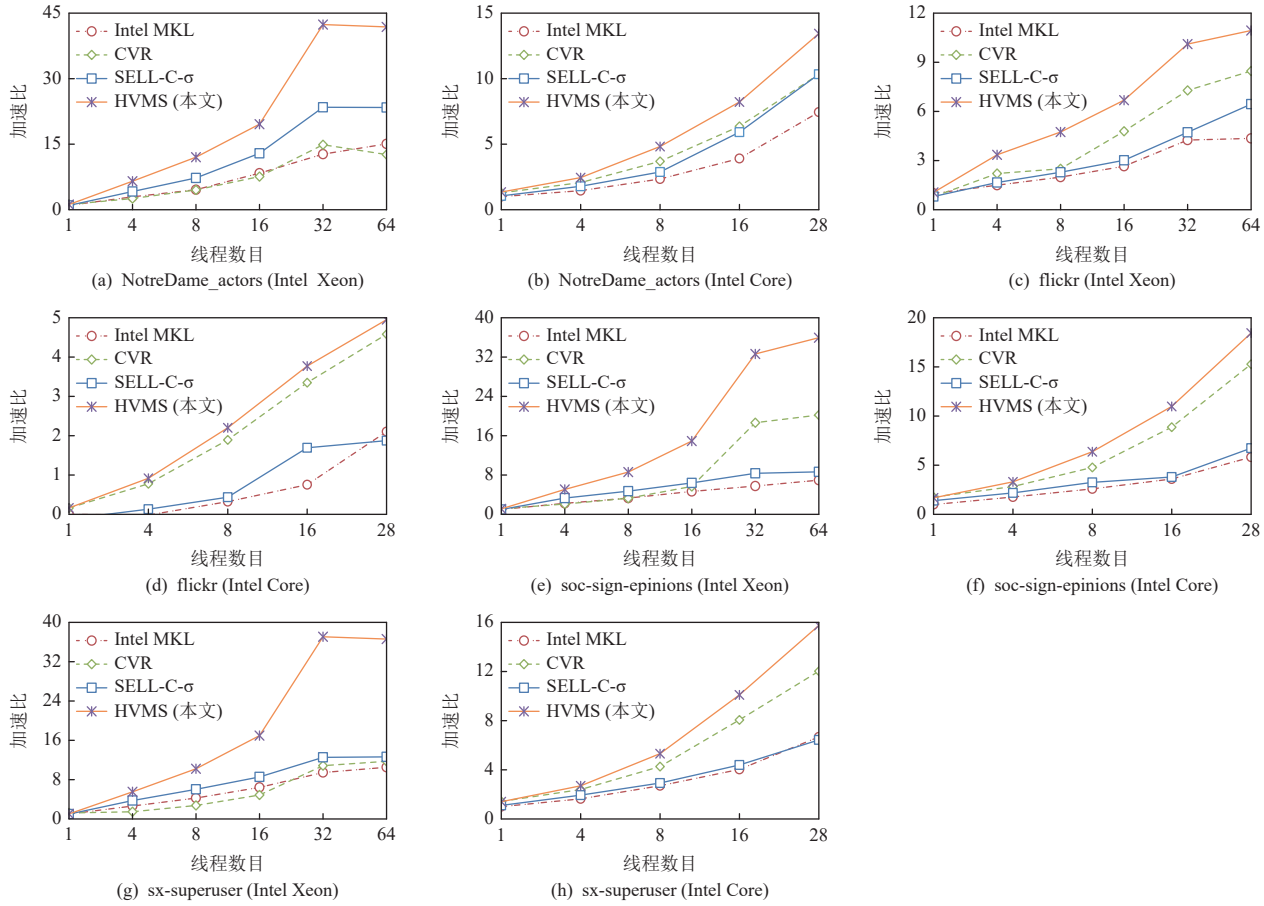
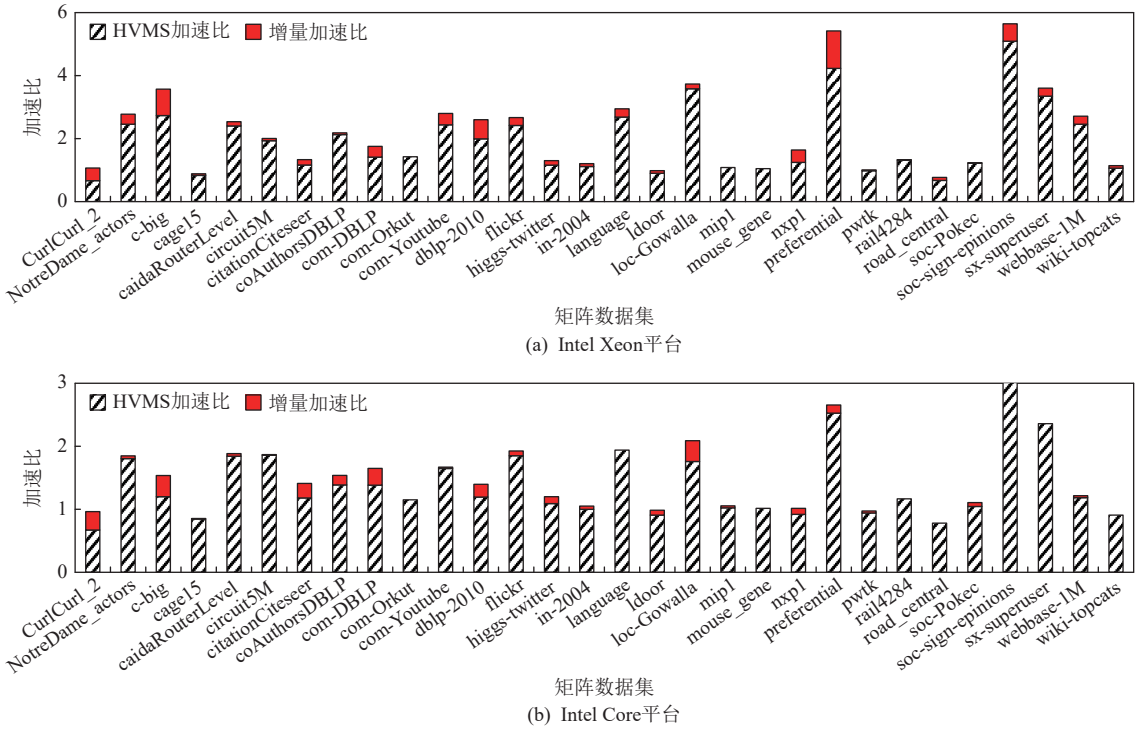


Fig. 13 Scaling comparison of four schemes on four sparse matrices

图 13 4 种方案在 4 种稀疏矩阵上的可扩展性对比



(a) Intel Xeon平台

(b) Intel Core平台

Fig. 14 Effect of changing priorities of rules in HVMS on performace

图 14 更改 HVMS 中规则的优先级对性能的影响

等技术手段, 从多个维度提升 SpMV 的性能.

ALBUS[16] 设计了一种简单的逐行向量计算机制, 将单行内的非零元依次加载入向量寄存器计算; CSR2[17] 利用零元填充和切块的方式进一步提升向量计算的效率; VHCC[18] 通过拼接多行来提高向量计算的效率, 同时对矩阵做 2 维划分来提升数据局部性; CSR5[7] 通过拼接多行以及分段求和的方法设计向量优化机制, 并且还提供了在多个平台上的具体优化实现; CVR[8] 将非零行依次加载入 SIMD 通道内计算, 同时处理 SIMD 通道宽度的行, 可大大提升向量计算的效率; BCSR[19] 通过切分稠密子块, 从而将稀疏矩阵计算转换成整齐的数据计算模式; SELL-C-σ[9] 可看做是对 ELL 格式的改进, 矩阵重排和行划分能够大量减少零元填充的数量; BCCOO[20] 采用分块策略, 利用位压缩行索引存储来缓解内存压力; Merge-SpMV[21] 设计了一种更优的负载均衡机制, 并且无需数据预处理即可获得不错的性能加速; LAV[22] 针对符合幂律分布的矩阵, 设计了行/列划分的机制来提升数据访问局部性; CFS[23] 利用图着色算法缓解对称矩阵的写冲突问题, 可大大减轻多线程情况下的内存访问压力; Regu2D[24] 利用矩阵划分、规则重排以及索引压缩的方式提升向量计算的效率; SpV8[15] 聚合规则子矩阵, 目的是让更多的数据进行向量计算.

SpMV 的优化平台也非常的广泛, 文献 [25] 是在神威平台上的优化工作, 文献 [26] 是在 POWER8 系统上的优化工作, 文献 [27] 是在 FPGA 平台上的优化工作, 文献 [28] 是在众核平台上的优化工作, 文献 [29–31] 是在 GPU 平台上的优化工作.

近几年, 由于海量数据的出现, 数据类型以及数据分布多种多样, 因此自动调优和性能建模的相关工作成为热门的研究内容, 代表性工作有文献 [32–40].

6 总 结

本文深入分析了向量优化 SpMV 计算的难点以及所带来的挑战. 向量化一直以来都是加速程序性能的关键技术手段之一, 然而由于稀疏矩阵数据的不规则性, 规则化的向量计算难以为 SpMV 带来成正比的性能提升. 尤其随着越来越丰富的矩阵数据类型的产生, 单个矩阵内的非零元分布以及矩阵之间的非零元分布特征千差万别, 并且向量化硬件的多样性也给通用 SpMV 的向量优化带来了不小的挑战. 基于此, 本文抽象了 4 种向量计算的基本操作: 规约求和操作、掩码运算操作、前向求和操作以及零元填充, 并且提出了基于混合向量化的 SpMV 优化机制——HVMS. 该机制根据抽象的向量基本操作, 设计相应的规则指导稀疏矩阵的规则化数据转换, 目的是为了将不规则的稀疏矩阵运算映射到规则的

向量运算上进行计算，最终利用规则的向量指令加速 SpMV 运算，充分发挥向量计算的效率. 本文以 AVX512 为例，完成基于 HVMS 的 SpMV 优化实现，在 30 个常用矩阵数据集上进行实验设计与分析，并且相比 CVR, SELL-C-σ, Intel MKL, HVMS 获得了不错的性能加速比以及更优的线程可扩展性.

作者贡献声明：颜志远提出方案、完成实验并撰写论文；解壁伟提出实验方案并指导论文写作；包云岗提供研究思路、给予指导意见.

参 考 文 献

[1] Zilli G. Iterative methods for solving sparse linear systems with a parallel preconditioner[J]. International Journal of Computer Mathematics, 1992, 44(1/2/3/4): 111−119

[2] Diaz J, Munoz-Caro C, Nino A. A survey of parallel programming models and tools in the multi and many-core era[J]. IEEE Transactions on Parallel and Distributed Systems, 2012, 23(8): 1369−1386

[3] Chrysos G. Intel® Xeon Phi™ coprocessor-the architecture[EB/OL]. 2014[2023-07-31]. http://gec.di.uminho.pt/minf/cpd1314/SCD/Intel_Xeon-PhiArch.pdf

[4] Anderson C S, Zhang Jingwei, Cornea M. Enhanced vector math support on the Intel® avx-512 architecture[C]//Proc of the 25th Symp on Computer Arithmetic (ARITH). Piscataway, NJ: IEEE, 2018: 120−124

[5] Reddy V G. NEON technology introduction[J]. ARM Corporation, 2008, 4(1): 1−33

[6] Wang Endong, Zhang Qing, Shen Bo, et al. Intel math kernel library[EB/OL]. 2014[2023-06-13].https://link.springer.com/chapter/10.1007/978-3-319-06486-4_7

[7] Liu Weifeng, Vinter B. CSR5: An efficient storage format for cross-platform sparse matrix-vector multiplication[C]//Proc of the 29th Int Conf on Supercomputing. New York: ACM, 2015: 339−350

[8] Xie Biwei, Zhan Jianfeng, Liu Xu, et al. CVR: Efficient vectorization of SpMV on x86 processors[C]//Proc of the 16th Annual IEEE/ACM Int Symp on Code Generation and Optimization. New York: ACM, 2018: 149−162

[9] Kreutzer M, Hager G, Wellein G, et al. A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units[J]. SIAM Journal on Scientific Computing, 2014, 36(5): C401−C423

[10] Flynn M J. Some computer organizations and their effectiveness[J]. IEEE Transactions on Computers, 1972, 21(9): 948−960

[11] Lomont C. Introduction to Intel advanced vector extensions[EB/OL]. 2011[2023-06-13].https://hpc.llnl.gov/sites/default/files/intelAVXintro.pdf

[12] Oberman S, Favor G, Weber F. AMD 3DNow! Technology: Architecture and implementations[J]. IEEE Micro, 1999, 19(2): 37−48

[13] Luebke D, Harris M, Govindaraju N, et al. GPGPU: General-purpose computation on graphics hardware[C/OL]//Proc of the 2006 ACM/IEEE Conf on Supercomputing. New York: ACM, 2006[2023-08-04].https://dl.acm.org/doi/10.1145/1188455.1188672

[14] Davis T A, Hu Yifan. The University of Florida sparse matrix collection[J]. ACM Transactions on Mathematical Software, 2011, 38(1): 1−25

[15] Li Chenyang, Xia Tian, Zhao Wenzhe, et al. SpV8: Pursuing optimal vectorization and regular computation pattern in SpMV[C]//Proc of the 58th ACM/IEEE Design Automation Conf (DAC). Piscataway, NJ: IEEE, 2021: 661−666

[16] Bian Haodong, Huang Jianqiang, Liu Lingbin, et al. ALBUS: A method for efficiently processing SpMV using SIMD and load balancing[J]. Future Generation Computer Systems, 2021, 116: 371−392

[17] Bian Haodong, Huang Jianqiang, Dong Runting, et al. CSR2: A new format for SIMD-accelerated SpMV[C]//Proc of the 20th IEEE/ACM Int Symp on Cluster, Cloud and Internet Computing (CCGRID). Piscataway, NJ: IEEE, 2020: 350−359

[18] Tang Waiteng, Zhao Ruizhe, Lu Mian, et al. Optimizing and auto-tuning scale-free sparse matrix-vector multiplication on Intel Xeon Phi[C]//Proc of the 13th Annual IEEE/ACM Int Symp on Code Generation and Optimization. Piscataway, NJ: IEEE, 2015: 136−145

[19] Eberhardt R, Hoemmen M. Optimization of block sparse matrix-vector multiplication on shared-memory parallel architectures[C]//Proc of the 2016 IEEE Int Parallel and Distributed Processing Symp Workshops (IPDPSW). Piscataway, NJ: IEEE, 2016: 663−672

[20] Yan Shengen, Li Chao, Zhang Yunquan, et al. yaSpMV: Yet another SpMV framework on GPUS[J]. ACM SIGPLAN Notices, 2014, 49(8): 107−118

[21] Merrill D, Garland M. Merge-based parallel sparse matrix-vector multiplication[C]//Proc of the 2016 Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2016: 678−689

[22] Yesil S, Heidarshenas A, Morrison A, et al. Speeding up SpMV for power-law graph analytics by enhancing locality & vectorization[C/OL]//Proc of the 2020 Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2020[2023-07-31].https://ieeexplore.ieee.org/document/9355205

[23] Elafrou A, Goumas G, Koziris N. Conflict-free symmetric sparse matrix-vector multiplication on multicore architectures[C/OL]//Proc of the 2019 Int Conf for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2019[2023-07-31].https://dl.acm.org/doi/10.1145/3295500.3356148

[24] Fei Xiang, Zhang Youhui. Regu2D: Accelerating vectorization of SpMV on Intel processors through 2D-partitioning and regular arrangement[C/OL]//Proc of the 50th Int Conf on Parallel Processing. New York: ACM, 2021[2023-07-31].https://dl.acm.org/doi/10.1145/3472456.3472479

[25] Liu Changxi, Xie Biwei, Liu Xin, et al. Towards efficient SpMV on sunway manycore architectures[C]//Proc of the 2018 Int Conf on Supercomputing. New York: ACM, 2018: 363−373

[26] Buono D, Petrini F, Checconi F, et al. Optimizing sparse matrix-vector multiplication for large-scale data analytics[C/OL]//Proc of the 2016 Int Conf on Supercomputing. New York: ACM, 2016[2023-07-31].https://dl.acm.org/doi/10.1145/2925426.2926278

[27] Umuroglu Y, Jahre M. An energy efficient column-major backend for FPGA SpMV accelerators[C]//Proc of the 32nd Int Conf on Computer Design (ICCD). Piscataway, NJ: IEEE, 2014: 432−439

[28] Liu Xing, Smelyanskiy M, Chow E, et al. Efficient sparse matrix-vector multiplication on x86-based many-core processors[C]//Proc of the 27th Int ACM Conf on Int Conf on Supercomputing. New York: ACM, 2013: 273−282

[29] Li Kenli, Yang Wangdong, Li Keqin. Performance analysis and optimization for SpMV on GPU using probabilistic modeling[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 26(1): 196−205

[30] Niu Yuyao, Lu Zhengyang, Dong Meichen, et al. TileSpMV: A tiled algorithm for sparse matrix-vector multiplication on GPUS[C]//Proc of the 2021 IEEE Int Parallel and Distributed Processing Symp (IPDPS). Piscataway, NJ: IEEE, 2021: 68−78

[31] Greathouse J L, Daga M. Efficient sparse matrix-vector multiplication on GPUs using the CSR storage format[C]//Proc of the 2014 Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2014: 769−780

[32] Li Jiajia, Tan Guangming, Chen Mingyu, et al. SMAT: An input adaptive auto-tuner for sparse matrix-vector multiplication[C]//Proc of the 34th ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2013: 117−126

[33] Zhao Yue, Li Jiajia, Liao Chunhua, et al. Bridging the gap between deep learning and sparse matrix format selection[C]//Proc of the 23rd ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2018: 94−108

[34] Feng Siying, Sun Jiawen, Pal S, et al. Cosparse: A software and hardware reconfigurable SpMV framework for graph analytics[C]// Proc of the 58th ACM/IEEE Design Automation Conf (DAC). Piscataway, NJ: IEEE, 2021: 949−954

[35] Tang Waiteng, Zhao Ruizhe, Lu Mian, et al. Optimizing and auto-tuning scale-free sparse matrix-vector multiplication on Intel Xeon Phi[C]//Proc of the 13th Annual IEEE/ACM Int Symp on Code Generation and Optimization. New York: ACM, 2015: 136−145

[36] Zhao Yue, Zhou Weijie, Shen Xipeng, et al. Overhead-conscious format selection for SpMV-based applications[C]//Proc of the 2018 IEEE Int Parallel and Distributed Processing Symp (IPDPS). Piscataway, NJ: IEEE, 2018: 950−959

[37] Benatia A, Ji Weixing, Wang Yizhuo, et al. Sparse matrix format selection with multiclass svm for SpMV on GPU[C]//Proc of the 45th Int Conf on Parallel Processing (ICPP). Piscataway, NJ: IEEE, 2016: 496−505

[38] Xie Zhen, Tan Guangming, Sun Ninghui. Research on optimal performance of sparse matrix-vector multiplication and convoulution using the probability-process-RAM model[J]. Journal of Computer Research and Development, 2021, 58(3): 445−457(in Chinese)
(谢震, 谭光明, 孙凝晖. 基于 PPR 模型的稀疏矩阵向量乘及卷积性能优化研究[J]. 计算机研究与发展, 2021, 58(3): 445−457)

[39] Choi J W, Singh A, Vuduc R W. Model-driven autotuning of sparse matrix-vector multiply on GPUS[J]. ACM SIGPLAN Notices, 2010, 45(5): 115−126

[40] Vuduc R, Demmel J W, Yelick K A. OSKI: A library of automatically tuned sparse matrix kernels[J]. Journal of Physics: Conference Series. 2005, 16(1): 521−530

Yan Zhiyuan, born in 1992. PhD candidate. Her main research interests include high performance computing and computer architecture.
颜志远, 1992 年生. 博士研究生. 主要研究方向为高性能计算、计算机体系结构.

Xie Biwei, born in 1987. PhD, assistant professor. His main research interests include open-source EDA, open-source chip, high performance computing, and computer architecture.
解壁伟, 1987 年生. 博士, 助理研究员. 主要研究方向为开源 EDA、开源芯片、高性能计算、计算机体系结构.

Bao Yungang, born in 1980. PhD, professor. His main research interests include data-center architecture, agile design methodology of processor chips, and ecosystem of open-source processor chips.
包云岗, 1980 年生. 博士, 研究员. 主要研究方向为数据中心体系结构、处理器芯片敏捷设计方法论、开源处理器芯片生态.