doi:10.3772/j.issn.1002-0470.2024.08.003

高性能稀疏矩阵向量乘的程序设计综述①

杜 臻②*** 谭光明* 孙凝晖*

(*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)

摘 要 稀疏矩阵向量乘(SpMV)广泛应用于科学计算、图计算、数据分析等领域,是自现代计算机诞生以来经久不衰且挑战依旧的研究热点。本文系统回顾了20世纪70年代以来稀疏矩阵向量乘程序设计的发展脉络和各阶段的代表性工作;分析比较了这一领域4条技术路线,即人工程序设计、自动调优器、稀疏编译器和自动程序设计器,在当今的流行方法;并在此基础上对高性能稀疏矩阵向量乘程序设计的研究趋势做出预测,力图给学习者和研究者带来有益的知识与启示。

关键词 稀疏矩阵向量乘(SpMV);稀疏矩阵格式;自动调优;稀疏编译器;高性能计算;并行算法

稀疏矩阵是科学计算领域最常见的数据类型之一,围绕着稀疏矩阵的计算问题是高性能计算重要的研究热点。被广泛接受的佛罗里达稀疏矩阵集[1]已经包含了2 893个稀疏矩阵。它们涵盖了91个领域,包括能源网络、最小二乘、材料、结构和半导体器件等问题。加州大学伯克利分校的研究报告[2]将稀疏矩阵计算列为第2重要的并行计算模式。

稀疏矩阵向量乘(sparse matrix-vector multiplication,SpMV)是最具研究价值的稀疏矩阵计算之一。它的计算模式常见于诸多高性能计算负载中。为了提升SpMV的性能,相关研究者已经做出了大量高性能SpMV程序设计的工作,但迄今SpMV程序的性能依旧低于人们的预期。不断增进对这一问题的了解和认识,是提升相关应用性能不可避免的主题。

本文关注基于中央处理器(central processing unit,CPU)和图形处理器(graphics processing unit,GPU)的单机SpMV程序设计方法。第1节介绍SpMV的计算原理及面临的挑战;第2节梳理相关研究的历史脉络;第3~6节分别介绍和分析SpMV程序设计的4条路线:人工程序设计、自动调优器、稀疏编译器、自动程序设计器;第7节在GPU上对几条路线的代表性工作进行测试;第8节对4条路线进行总结,并对高性能SpMV程序设计的研究趋势进行预测。

1 SpMV计算原理及挑战

本节将介绍SpMV程序的计算流程、分析SpMV程序的特点,从而揭示SpMV研究的挑战性。

1.1 SpMV的计算过程

图1展现了SpMV的计算过程。稀疏矩阵中的每一个非零元根据其列号与输入向量对应位置的元素相乘,相乘的结果根据非零元的行号累加到输出向量的对应位置。

稀疏矩阵由大量零元和少量非零元构成,其中零元对计算结果没有贡献,因此仅将稀疏矩阵的非零元压缩存储在稀疏矩阵格式中可以节省存储开销和提升计算性能。所以,如何设计出良好的稀疏矩

① 国家杰出青年科学基金(T2125013)资助项目。

② 男,1996年生,博士生;研究方向:高性能计算,稀疏计算;联系人,E-mail:duzhen18z@ict.ac.cn。

(收稿日期:2023-09-26)

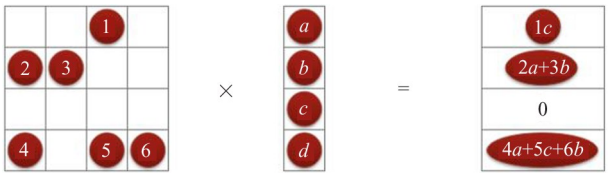

图 1　SpMV 的计算过程

阵格式来编码稀疏矩阵的非零元、设计合适的算法来加速 SpMV 的计算流程并在二者之前达成良好的权衡和配合，是设计出高性能 SpMV 程序的关键。

1.2　高性能 SpMV 程序设计的挑战

设计高性能的 SpMV 程序面临的挑战可以总结为以下 3 点。

挑战 1　SpMV 的程序设计和稀疏矩阵稀疏模式的匹配程度决定了性能，但是矩阵的稀疏模式存在巨大的多样性[3-4]。稀疏矩阵的稀疏模式又称为稀疏矩阵的非零元分布，SpMV 程序的性能与稀疏模式高度相关，并不存在一种适合所有矩阵的程序设计。为了获得极致的性能，人们需要为大量来自不同应用背景的矩阵定制程序设计。然而，考虑到矩阵的稀疏性，将少量的非零元分布到矩阵中存在巨量的可能。佛罗里达矩阵集中已经包含了 2 893 个矩阵。考虑到其他数据集以及在未来不断产生的新的稀疏矩阵，让 SpMV 的研究者穷尽所有的稀疏模式并为它们分别设计 SpMV 程序是不可能的。

挑战 2　稀疏矩阵的稀疏模式与程序性能之间具有复杂的关联[5-6]。为所有稀疏模式分别设计程序是困难的，而为某一种稀疏模式设计程序也并不容易。SpMV 程序的性能与诸多因素相关，比如负载均衡、存储压缩率、并行度、局部性、数据缓存方式、预处理时间等。每个单一因素都不唯一决定 SpMV 的程序性能，而是根据稀疏模式以不同权重共同决定了性能。但是，程序设计内在的复杂权衡与稀疏模式对性能的影响至今也没有被量化，需要相关研究者依靠模糊、定性的经验进行繁琐的尝试。

挑战 3　稠密的、规则的多核及向量计算器件与稀疏的、不规则的数据之间存在天然的疏离，如何充分利用硬件的特性和性能需要精妙的程序设计[7-8]。自单核处理器性能的发展进入瓶颈以来，多核与并行已经成为了维持摩尔定律的最大动力。扩大处理器规模和提升算力均依赖于对相同处理器核心的复用，这一趋势同时出现在 CPU 和 GPU 的设计中。此外，为了提升单指令的数据吞吐量，向量计算（CPU 的“单指令多数据”、SIMD（single instruction multiple data）、GPU 的“单指令多线程”和 SIMT（single instruction multiple threads））器件也在发挥重要的作用。但是，高度密集的、规则的向量和多核处理器的硬件设计与稀疏矩阵中稀疏的、不规则的数据分布是不协调的，这导致 SpMV 的计算难以在多个核心之间做到完全的负载均衡，也难以完全利用向量计算器件。

2　SpMV 程序设计发展脉络

由于 SpMV 研究的价值和挑战，自现代计算机诞生以来，人们对其进行了大量的研究。图2系统梳

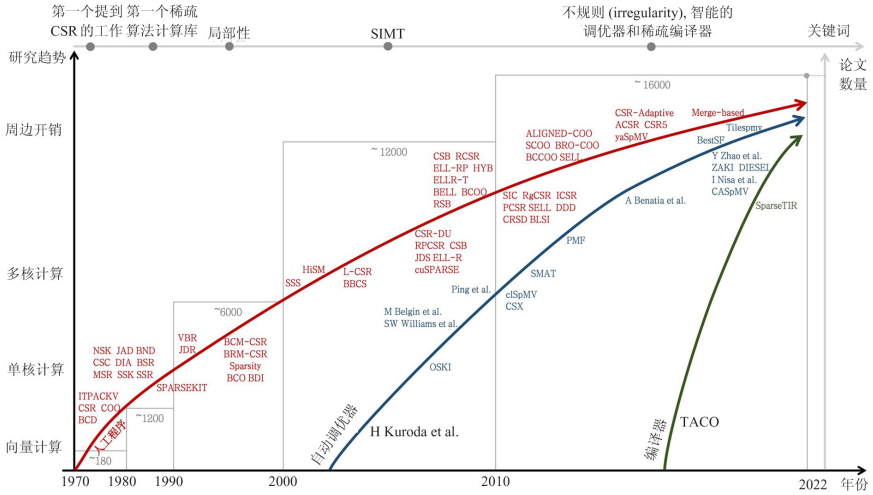

图 2　50 年来 SpMV 程序设计的研究路径

理了 SpMV 程序研究在不同时间段的主流工作、主要路线、研究趋势和相关论文数量。

2.1　1970 – 1980 年:基础的稀疏矩阵格式

SpMV 最早可考证的一批公开发表的研究[9-11]出现在 1970 年。彼时的 SpMV 程序多为早期的向量计算机设计。这类程序需要在稀疏矩阵格式设计上适应向量计算的特点,从而提升计算效率。但是,将稀疏矩阵格式调整为适应向量计算的形式会引入额外的存储和计算开销,所以这类程序的设计还必须考虑向量计算效率和额外开销的权衡。ITPACKV(iterative techniques package version V)软件包[12](经过少量改进,现在被称为 ELL 或 ELLPACK(elliptic package))是在当时的实际应用中流行的 SpMV 程序。图 3 底部展现了 ELL 格式存储图左侧稀疏矩阵的方式,data 数组存储了稀疏矩阵每一个非零元的值,indices 数组存储了每一个非零元的列号。虽然忽略零元是稀疏矩阵格式设计的基本目标,但是 ELL 依旧保留了一些零元以保证每一行的零元与非零元的数量之和相等。在按行并行的基础上,这保证了每一行的计算逻辑完全相同,从而适应向量计算机的架构。

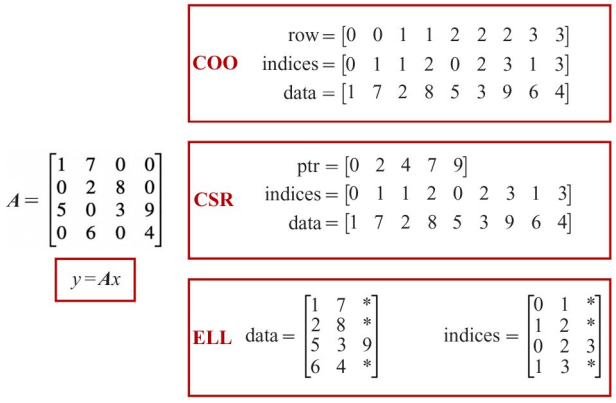

图 3　基础的稀疏矩阵格式

此外,因为通用处理器的流行,另外 2 种更适合通用处理器的基础稀疏矩阵格式 CSR(compressed sparse row)和 COO(coordinate)也被提出。图 3 也展现了图左侧稀疏矩阵在 COO 和 CSR 两种格式中的存储方式。COO 用 3 个数组存储每一个非零元的行号(row 数组)、列号(indices 数组)和值(data 数组)。CSR 将非零元行号压缩为每一行非零元数量的前缀和 ptr 数组,即每一行的第 1 个非零元在所有非零元中的位置。

2.2　1980 – 1990 年:关注结构化矩阵

20 世纪 80 年代,稀疏模式对 SpMV 程序性能的敏感性逐渐获得了人们的关注。SpMV 的研究者开始为一些特定的、常见的稀疏特征(即结构化矩阵)设计专用的 SpMV 程序。其中,DIA(diagonal storage format)适用于非零元聚集在对角线上的矩阵;BSR(block sparse row format)适用于局部稠密的矩阵;BND(banded storage format)适用于 LINPACK(linear algebra package)的条带矩阵[13];SSK(symmetric skyline storage format)与 SSR(symmetric sparse row storage format)适用于对称矩阵。相比 ELL、COO 和 CSR 所对应的 SpMV 程序,针对结构化矩阵的程序设计通过牺牲它们在大多数矩阵上的性能来换取它们在少量稀疏模式上的高性能。因此,这种针对结构化矩阵的程序也揭示了一种重要的权衡,即一个特定的 SpMV 程序不能在所有稀疏矩阵上同时获得较大的平均性能和最大性能。这些程序都被收集在开源算法库 SPARSEKIT[14]中。

2.3　1990 – 2000 年:通过提升数据局部性来提升性能

随着缓存结构逐渐在处理器中普及,20 世纪 90 年代 SpMV 程序设计工作主要关注数据局部性[15-17]。数据局部性的提升源于数据的重排。彼时,数据重排的方式主要是对矩阵进行分块,并且将同一个块的非零元存在一起,从而让在计算顺序上相近的非零元具有相近的行号和列号。这种方式保证了对 SpMV 中输入向量和输出向量访问的局部性。

2.4　2000 – 2010 年:SIMT 与自动调优

2000 年后,除了延续以往的研究热点,GPU 在通用计算中的应用也为 SpMV 研究提供了更多的空间和机会。GPU 相比 CPU 有 2 个优点:其一,GPU 更快的访存速度缓解了 SpMV 的访存瓶颈;其二,GPU 更高的并行度使得其拥有比 CPU 大得多的峰值性能,增加了 SpMV 的性能上限。为了适应 GPU 单指令多线程(SIMT)的计算模型和一个线程束(warp)内合并访存的条件,这个时期的工作在已有工作基础上进行了一些针对性的改进[18-19]。

在这个时期,自动调优也成为了提升 SpMV 性

能的关键方法。这类工作根据矩阵的稀疏模式给出最合适的 SpMV 程序设计。2005 年提出的 OSKI (optimized sparse kernel interface)[20] 是早期 SpMV 自动调优的代表性工作。它支持程序参数的调优,可以根据矩阵调整 BCSR(blocked compressed sparse row format)和 VBR(variable block row format)[21] 的分块大小,以保证对 SpMV 中向量访问的较高局部性。

2.5 2010 年 – 至今:关注高度不均衡的矩阵、智能化和编译技术

在最近十几年,除了进一步发展 SpMV 程序设计和自动调优技术,也有研究者提出了另外 2 条路线:稀疏编译器和自动程序设计器。最终在这一领域形成了图 2 所示的 4 条路线的布局。

SpMV 程序设计在这一时期的工作面对新的时代背景、关注新的问题。新的时代背景是高性能处理器的发展。Intel 至强融核(Xeon Phi,SIMD)协处理器和 GPU 发展使得这个时期的 SpMV 程序几乎都为 SIMT 和 SIMD 设计。其中,因为 Xeon Phi 在 2019 年停产,GPU 成为了在当今学界研究热度更高的设备。

就 NVIDIA GPU 而言,其性能的提升和架构的革新为 SpMV 程序性能的提升提供了新的机会。更先进的架构扩展了 SIMT 模型的能力,比如更宽松的合并访存条件(只要访存位置都在一个高速缓存块之内就能触发合并访存,而不需要遵循特定的排列)、一个线程束内不同线程更方便灵活的数据交换(一个线程束内的线程共享同一个寄存器的命名空间,一个线程可以方便地读任何其他线程寄存器的数据)。

这一时期自动调优工作采用了更高水平的人工智能。因为 SpMV 程序设计与矩阵稀疏模式的关联没有办法通过观察直接明晰地得到,所以自动调优器更倾向于机器学习的方法,将二者的关联拟合到神经网络所构成的黑盒中。

在这个时间,稀疏编译器也被提出,代表性工作是 TACO[22](tensor algebra compiler,另一个稀疏编译器 SparseTIR[23] 采用了与其类似的设计)。它包含了一系列针对稀疏张量计算(包括 SpMV)的原语和优化。TACO 基于多面体模型,关注程序的循环优化和自动并行。为了支持稀疏张量计算,TACO 通过新的“索引压缩”的原语,将张量每一个维度遍历改造为对压缩索引的遍历,从而支持稀疏张量的计算。

2022 年,国内的相关研究者提出了 AlphaSparse[24],开启了一条立足于自动程序设计的研究路线。相比 SpMV 自动调优器和稀疏编译器建模一整个现成的程序,AlphaSparse 模仿并建模了人工设计 SpMV 程序的流程,并通过机器自动生成不同的设计流程来达到用机器从头设计程序的效果。

接下来的 4 节将依次介绍人工设计 SpMV 程序的方法、SpMV 自动调优器、稀疏编译器和自动程序设计器。

3 人工程序设计

人工程序设计是贯穿整个 SpMV 程序研究历史的路线,是 SpMV 性能提升的源头,也是其他路线的基础。本节总结了人工程序设计,尤其是格式设计的基本方法(这些方法被总结在图 4 中),并列举了不同类型的优化方法的典型案例,分析其蕴含的权衡。

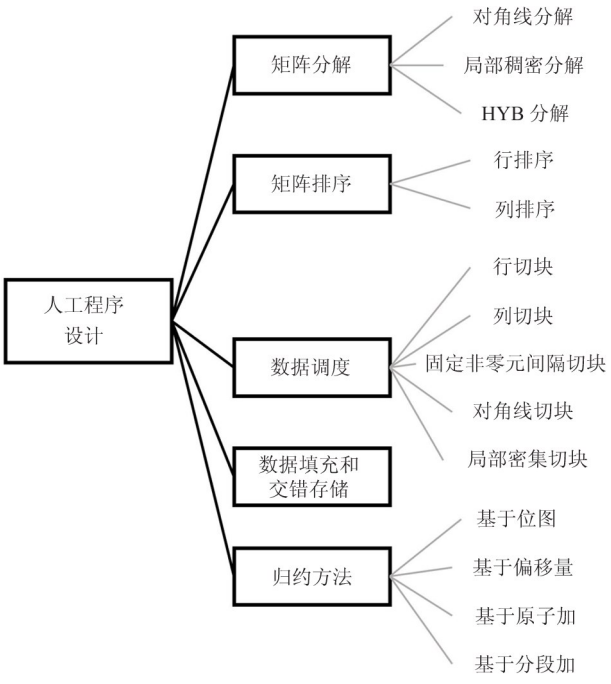

图 4 现今主流人工程序设计方法的总结

3.1　矩阵分解

矩阵分解将一个矩阵分解成多个相同形状的矩阵,并且针对每一个分解出的矩阵分别使用不同的SpMV 程序。主流的矩阵分解手段包括对角线分解[25-26]、局部稠密分解[27]和 HYB(hybrid format)分解[28]。

图 5 展现了 3 种主流分解方式的案例。其中灰色部分为非零元所在的位置,白色部分为零元所在的位置。图最上方为对角线分解,其将矩阵分为对角线条带部分和其他部分。图中间为局部稠密分解,其将稠密块从原始的矩阵中分离开来,并放到另一个相同形状的矩阵中。图最下面的是 HYB 分解,其将矩阵每一行某个位置及其之前的非零元(在此例中这个位置是行第 3 个非零元)分到一个矩阵中,剩下的非零元分到另一个矩阵中。

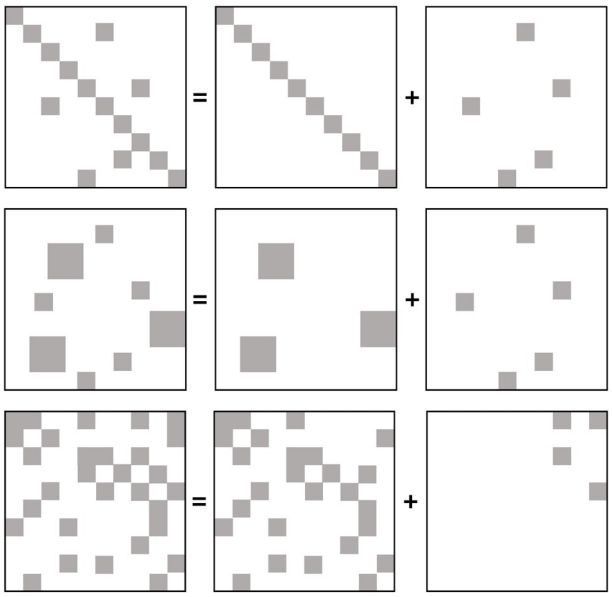

图 5　对角线分解、局部稠密分解和 HYB 分解的例子

由矩阵分解产生的其中一个子矩阵往往拥有某种典型的特征(比如对角线明显、局部稠密块多、行非零元均衡),可以在特定的程序设计中获得较高的性能表现。然而,经由矩阵分解产生的另一个矩阵往往又拥有很多棘手的、不利于得到高性能的特征,比如会限制并行度的高稀疏性以及需要特殊处理的大量空行。所以分解方法和分解参数的选择需要根据矩阵的稀疏模式进行权衡。

3.2　矩阵的排序

矩阵的排序会调整矩阵非零元在稀疏矩阵格式中的位置,进而改变非零元被计算的顺序。这里介绍 2 种常见的排序方法:列排序、行排序。

列排序的工作倾向于将列号相近的非零元排在一起。因为矩阵的非零元会按照其列号与输入向量对应位置做乘法计算,所以当非零元列号相近时,它们对输入向量的访问位置也更接近;此外,无论是串行还是并行器件,在稀疏矩阵格式中排列在一起的非零元在多数时候也会在趋近的时间点进行计算。综合这两点因素,这种优化同时增加了对输入向量访问的空间与时间局部性。

图 6 展现了矩阵列排序的典型案例。这个例子中,矩阵从纵向被切为 3 个列条带。每个列条带中的非零元按行存储,并且一个列条带的最后一个非零元和下一个条带的第一个非零元接在一起。这一操作还有另一种等价的、基于列号的非零元粗粒度排序的视角,即在非零元列号与特定系数(在此例中这个系数为列条带的宽度)相除的结果增序的方向对非零元进行重排。

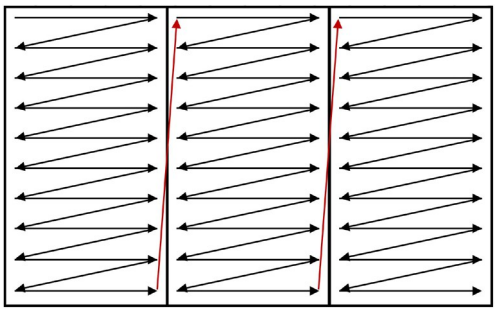

图 6　一个典型的列排序的例子

与列排序不同,行排序的工作倾向于将非零元数量相似的行放在一起[29-30]。这一优化并不会给SpMV 程序的性能带来直接的提升。相反,行排序会产生性能的劣化,原因有三。其一是对输出向量的访存局部性的降低。非零元的行号决定了当前非零元与向量元素的计算结果被归约的位置,即写输出向量的位置。在原始的行顺序下,对输出向量的访问就有最佳的局部性,调整行的顺序会增加稀疏矩阵格式中来自相邻行的非零元的平均距离,破坏访问输出向量的局部性。其二是访存量的增加。改

变行的顺序需要加入新的索引来记录新矩阵的行在老矩阵中的位置,访问这一索引需要引入新的访存。其三是排序非常耗时,会引入大量的预处理开销。

但是,行排序增加了行非零元数量分布的规律性,为后续的程序设计方法提供了性能提升的机会,从整体上带来了新的性能提升潜力。图 7 的例子展现了行排序在 SELL 这类典型的基于行填充的格式中的作用。图中,矩阵中的零元都被事先去掉,所有的非零元都被压在矩阵一侧。白色的部分代表之后被填充的显式零元,剩余的部分代表矩阵的非零元。SELL 将矩阵按照固定的间隔切分为一个个行条带,并且为每一行填充尽可能少的、显式的零元,使得每一个行条带中每一行的非零元和显式零元的数量之和相等(后文将会说明这类程序设计方法的作用)。将非零元数量近似的行放在一起,有利于减少每一个行条带中被填充的显式零数量。考虑到显式零会带来额外的计算开销,却对计算结果没有贡献,更少的显示零可以提升性能。

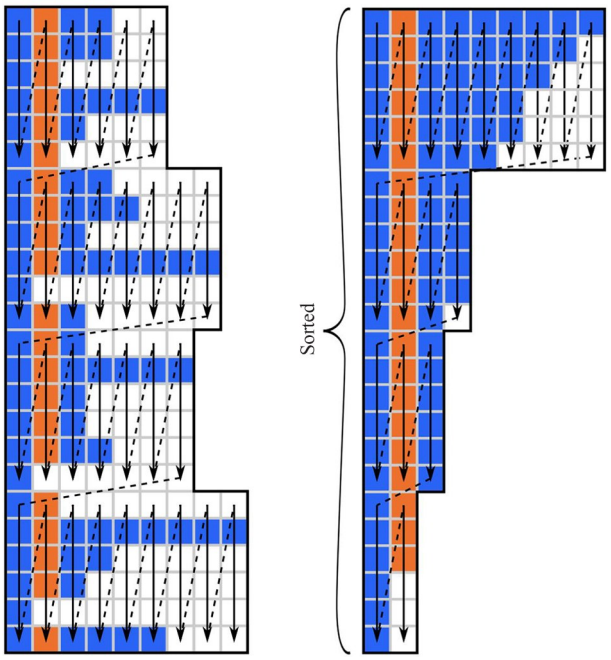

图 7　行排序在 SELL 格式中的作用

3.3　数据调度

为了运行在并行计算器件上,稀疏矩阵的非零元被切块并映射到不同并行级别的计算单元中。切块方法主要分为 5 种:行切块[30-31]、列切块[32-33]、固定非零元间隔切块[8]、对角线切块和局部密集切块。此外,根据计算器件的不同,并行级别的选择也不同。NVIDIA GPU 有线程块、线程束和线程 3 个并行级别;CPU 有向量和线程 2 个并行级别。而数据调度是切块方法和映射方法的组合。

图 7 展现了行切块的一个案例,SELL 按照向量单元或线程束的宽度对矩阵进行行切块、将一个行条带映射给一个向量单元或者线程束、并进一步将行条带的每一行映射给一个线程。图 8(与图 7 一样,矩阵中零元被去除,非零元被压到矩阵左侧,表现为灰色的方块)左侧展现了切块宽度为 2 的列切块的例子。这类切分会将行切为多段,并且每一段的非零元数量不多于 2。图 8 右侧展现了固定非零元间隔切块。这种切块会将相邻的 3 个非零元放到一个数据块中。

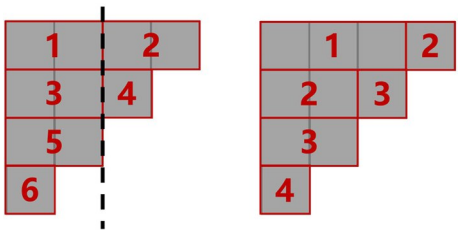

图 8　列切块和按照非零元间隔的切块

三种切块方法各有其利弊。行切块保证了行的完整性,避免了块之间的归约;但是,行切块使得较长的行和较短的行被同等地映射到硬件中,不利于负载均衡。列切块增加了计算一行的并行度,有利于处理长行,但会产生相对低效的多个数据块之间的归约(因为数据块会被映射到不同的计算单元上,这意味着更多计算单元之间的通信)。固定非零元间隔切块可以保证相对更好的负载均衡;但是,这种方式会导致数据块的边界与换行位置不对齐,从而导致较难的归约算法的设计。

对角线切块和局部密集切块根据矩阵的对角线和局部稠密块对矩阵进行切块。这 2 种切块方法主要应对带有对角线和局部稠密特征的结构化矩阵(如图 6 所示)。

3.4　数据填充与交错存储

向数据块中填充显式零元是稀疏矩阵格式常见的设计手段。除了在行分块之后进行填充的图 7,图 9 展现了 row-grouped CSR[34] 在列分块之后进行

填充的案例。首先对矩阵进行以 2 为宽度的列切块,然后显式零元(在图中表达为“ * ”)被填充到数据块中,使得每个数据块非零元和显式零的数量之和相同。

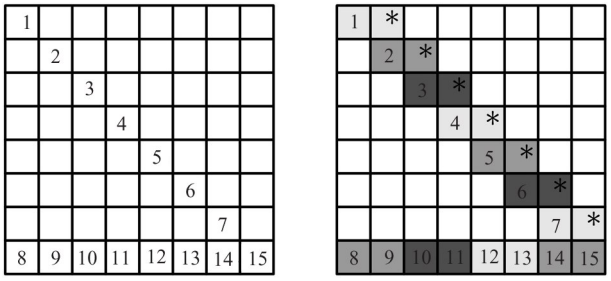

图 9　一个在列切块之后执行显式零元填充的案例

数据填充会造成存储的浪费和冗余的计算。但是,它也能带来两个收益。其一是更规则的数据排布降低了稀疏矩阵格式的存储开销。比如,在 SELL 中,因为每一个行条带中每一行的元素数量相同,所以只需要记录每一个行条带的行元素(非零元和显式零元)数量而不需要存储每一行的非零元数量。体现在 row-grouped CSR 中,每个数据块的大小可以直接用一个常量来存储,从而不需要在稀疏矩阵格式中记录所有数据块大小相关的信息。其二是数据填充可以带来交错存储的机会。

交错存储是广泛存在于 SpMV 矩阵格式中的设计方法,其将映射到同一个线程束或向量单元的不同线程的数据块中的元素交错放置在稀疏矩阵格式中,从而提升 SIMD 和 SIMT 的计算效率。以图 7 中 SELL 格式为例,黑色虚线箭头代表了映射到线程束的行条带的元素在经过了交错存储之后的存储顺序。因为在一个线程束或向量单元对应的行条带中,每一行的相同位置的元素被同时计算,将每一行相同位置的元素放置到一起可以增加每个线程束或者向量单元的访存局部性,从而提升性能。与其获得的优势相对的是,交错存储需要将数据完全重新排布,会带来一定的预处理开销。

3.5　归约方法的设计

前文所涉及的程序设计方法主要关注稀疏矩阵的非零元在内存中的排布以及并行计算单元与矩阵非零元的映射关系,对于稀疏矩阵格式的影响较大。而归约方法决定了矩阵非零元与向量元素计算以及不完全归约所产生的中间结果通过何种方式按行累加到一起。它们对 SpMV 算法的影响较大。先进人工程序的归约方法有图 10 所示的 4 类:基于位图的归约、基于行偏移量的归约、基于原子加的归约和基于分段加(segmented sum)的归约。图中,填入白色数字的黑色矩形代表矩阵非零元和向量元素相乘的结果或者它们按行归约产生的结果;填入黑色数字的白色矩形是需要被归约的中间结果对应的行号;圆角矩形代表这个归约方法在稀疏数据格式中配套的信息。这些归约方法虽然在不同的硬件和编程模型上有不同的代码实现,但是它们遵从相同的内在逻辑。

基于位图的归约[8,35]如图 10(a)所示。这类归约策略使用一个位图来记录中间结果的换行位置。位图中的一个比特对应一个中间结果。每一行的第一个中间结果被标记为 1,其他中间结果被标记为 0。这个归约算法的串行版本会在一个循环中同时遍历输入的中间计算结果和位图的对应比特,在比特位为 0 时累加当前的结果,在比特位为 1 时初始化遍历的状态(初始化用以累加的寄存器,当累加的寄存器有前一行的归约结果时记录这一结果),并开始对新一行的中间结果进行累加。在这个归约方法的并行版本中,每一个计算单元常常对应一个被归约的中间结果,对应位图中比特位为 0 的计算单元需要将自己的中间结果分享映射到同一行的、对应比特位为 1 的计算单元中。根据硬件特定和存储结果结构的不同,计算单元之间归约中间结果分享的方式也不同。在有向量计算特性的硬件中,分享的方式是使用向量计算的 shuffle 类操作[36];在有高层次、可编程共享内存的硬件中,分享的方式是将中间结果存到共享内存中。后文所述的基于偏移量和基于分段相加的归约方式都可以在其并行版本中使用这 2 种方式来进行计算单元之间的数据交换。

基于偏移量的归约[37-38]如图 10(b)所示。这类归约记录了需要被归约到同一行的中间结果的范围,并依此叠加同一行的中间结果。图中,0、2、6、8 是每一行的第 1 个中间结果的偏移量。即第 0 个、第 2 个、第 6 个是每一行第 1 个中间结果的位置,而 8 是所有中间结果的数量(它们的计算的方法与

CSR 的压缩非零元行号的计算方法类似)。这个归约算法的串行版本会在一个循环中遍历矩阵的所有行,并且根据中间结果的偏移量累加每一行的结果。在这个方法的并行版本中,不同的计算单元会分别处理不同行的中间结果的累加。

基于原子加的归约[28]方法直接根据中间结果的行号将中间结果原子加到内存的输出向量的对应位置,如图 10(c)所示。原子操作保证了对向量特定位置的累加的线程安全,是只针对并行器件的归约方法。基于原子操作的归约的性能取决于计算设备是否支持硬件级别的原子操作。

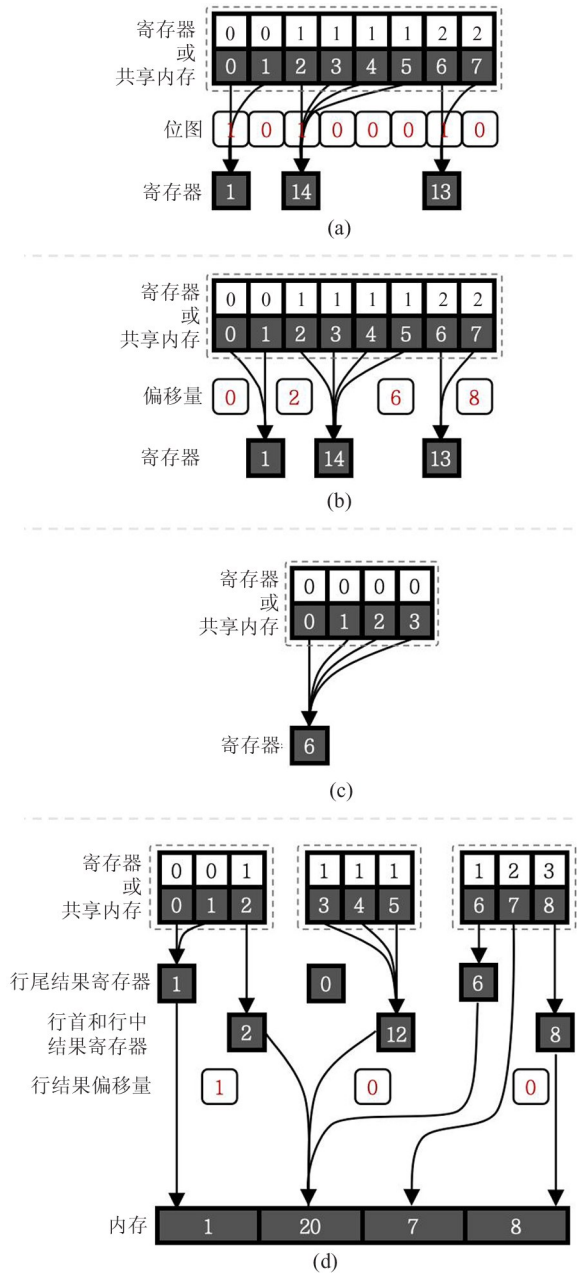

图 10　4 类常见的归约方法

基于分段相加的归约方式[35]可以应对切块位置和换行的位置不对齐的情况(比如按照固定非零元间隔的切块)。在这种情况下,不同行需要被归约的中间结果与计算单元存在复杂的对应关系:来自同一行的中间结果被包含在同一个计算单元中;来自同一行的中间结果横跨在多个相邻的计算单元中;一个计算单元中只包含一行的中间结果;一个计算单元包含了多行的中间结果。基于分段相加的归约方式可以用一套逻辑处理全部的这些对应关系。如图 10(d)所示,3 个相邻的中间结果被分配到同一个计算单元中,其中第 1 行的结果横跨了 3 个计算单元。针对前面所述的第 1 种情况,第 2 行的中间结果不参与计算单元之间的计算结果归约,直接写回内存。而针对其他情况,中间结果存储在“行首和行中结果寄存器”和“行尾结果寄存器”中。前者存储了每一行头部的计算结果(一个数据块最后一行的计算结果)或者每一行中间的计算结果(对应一个数据块中的所有中间结果属于同一行的情况),后者存储了每一行尾部的计算结果(一个数据块第 1 行的计算结果)。“行结果偏移量”存储了一行的头部和中间的计算结果横跨的计算单元数量。最终,利用“行结果偏移量”的信息和计算单元之间的数据交换累加“行首和行中结果寄存器”的值,再将累加的结果与“行尾结果寄存器”的值相加,可以归约每一行的所有中间结果。

4　自动调优器

SpMV 程序的性能与矩阵的稀疏模式存在难以量化的关联。自动调优器可以利用这种关联来根据矩阵的稀疏模式直接给出合适的程序。其中,对于矩阵稀疏模式的建模和对于调优空间的搜索是 SpMV 自动调优器要解决的 2 个关键问题。矩阵的稀疏模式的建模方式有 2 种,一种为人工特征统计,一种为基于机器学习的特征提取。调优的方式也有 2 种,一种是基于分类器的调优,另一种是基于迭代的调优。它们被总结在图 11 中。

随着自动调优的发展,一些工作与这些基础特征关注相同的问题,并且提出新的特征从更全面的视角来看待这些问题。

描述行非零元数量不均衡程度的新特征有:(1)行非零元数量的基尼系数[39-41];(2)行非零元数量的 P 值[41];(3)行非零元数量的频率直方图[42](矩阵的行非零元数量在不同区间的频率);(4)行非零元数量的中位数和众数[43];(5)对于行非零元数量分布与幂律分布近似的矩阵,其分布的幂指数[3];(6)独特的行非零元数量的个数和均值[4]。

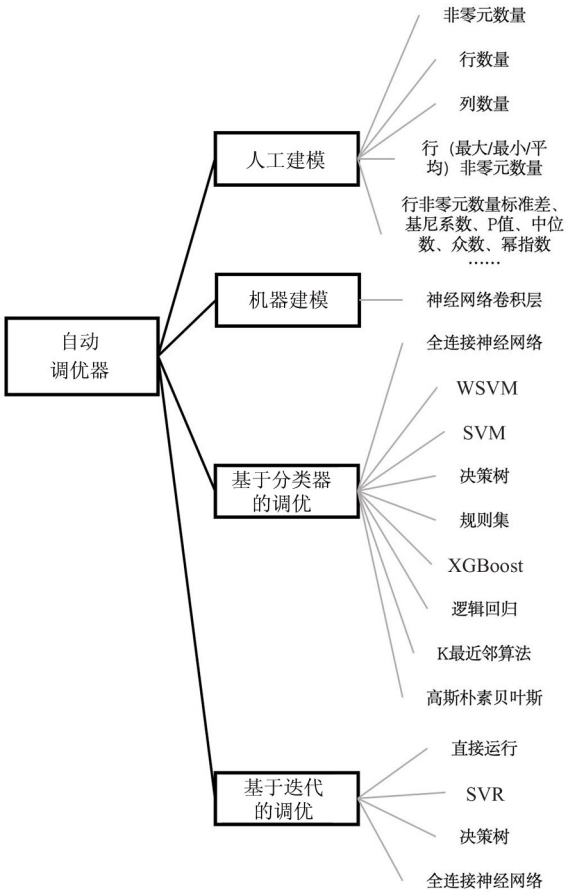

图 11　现今主流自动调优方法的总结

4.1　基于人工统计的矩阵稀疏模式的建模

因为 SpMV 先按位进行乘法计算,而后按行叠加乘法计算的结果。所以,非零元在矩阵中不同行的分布影响程序的选择。

描述输入向量访问局部性的新特征有:(1)每一行(列)由连续的非零元构成的数据块的数量[44]、大小[5-6]和距离[45]的均值和方差;(2)一行内第 1 个和最后 1 个非零元列号差值的均值和方差[5];(3)相邻两行的非零元分布模式差异(非零元的列号的差值)的均值[4]。这些新特征源于对非零元在列方向上相对距离的计算。

因为一些 SpMV 的程序的性能对输入更敏感,所以包含了这些 SpMV 程序的自动调优器会额外关注与它们有关的矩阵特征。在已有的自动调优工作中,被关注的 SpMV 程序有 ELL、DIA 和它们的变体,以及对于空行有特别处理的程序。针对 ELL 及其变体的特征有填充率[3],即在按照 ELL 的方式进行填充后,矩阵的非零元数量占所有元素数量的比例。针对 DIA 及其变体的特征有矩阵对角线的数量和稠密程度[3]。此外,因为一些程序需要对矩阵的空行进行额外的处理,改变了性能变化的规律,所以在包含了这些程序的自动调优器中,空行的数量需要被额外考虑[4,41]。

矩阵的建模也可以和硬件的信息结合起来,代表性的工作是 WISE[41],其按照 L1、L2 cache 的大小将矩阵切成行条带、列条带和数据块,并且按照前文所述的统计方法来计算三者的特征。比如行条带、列条带和数据块的非零元数量的均值和方差、空行的数量等。并且,WISE 将同一行距离在一个高速缓存块之内的非零元视为连续的非零元,用以代替以往严格的连续。这种方式更符合内存实际的访问模式。

新的特征可以从旧的特征计算而来。比如矩阵非零元的密度[46-47](矩阵的非零元数量和矩阵行列数量乘积的比值)、行最大非零元数量和最小非零元数量的差值[48]、最大和最小行非零元数量的差值与列数量的比值[39]、行最大非零元数量和平均非零元数量的差值[45]。这些从旧特征中被进一步计算出的特征可以视为对原始特征的归一化。

4.2　基于机器学习的矩阵稀疏模式的建模

除了人工的特征统计,另有一派自动调优方法利用神经网络的卷积层[49]来提取特征。

在卷积层中,一个滤波器会罩在稀疏矩阵上并滑动,矩阵中被覆盖的元素会和滤波器对应位置的元素相乘。相乘的结果会被累加到一起,并写到输出矩阵的对应位置。滤波器的参数是通过机器学习的训练得到的。虽然相比人工特征,这些通过卷积

得到的特征的可解释性较弱,但是它们可以包含矩阵中没有被人们注意到的隐含的信息。

因为主流的卷积操作将输入矩阵视同为稠密矩阵(传统的卷积方法一般面向计算机视觉,图像本就是稠密的),在整个矩阵上执行卷积操作会带来计算和存储 2 个层面的巨大开销。所以大多数基于机器学习的矩阵建模都会先将矩阵采样为一个更小的特征矩阵。采样的策略都是先将矩阵切块、用人工统计的方式统计出每一块的特征、将这些特征按照块的位置组合成特征矩阵(大小为 64 × 64、128 × 128、256 × 256 等)。针对每一个块的特征统计方式有:(1)块是否为空块[3];(2)块的非零元密度[3,50];(3)块中元素与对角线的距离[3];(4)块中行非零元数量平均值[50];(5)块中行非零元数量最大值[50];(6)块的非零元数量[51];(7)块的行数量[51];(8)块的列数量[51]。因为针对每一个块的人工特征提取方法和之后基于卷积的进一步的特征提取是解耦的,所以从原理上,前文所述的所有人工特征统计方法都能运用在特征矩阵的抽样中。基于不同抽样方法产生的特征矩阵会作为卷积神经网络不同通道的输入。

WACO(workload-aware co-optimization)[52]进一步去除了人工的矩阵抽样,从原始稀疏矩阵开始做卷积。为了解决将稀疏矩阵视为稠密矩阵导致的存储开销过大的问题,WACO 采用了稀疏卷积[53]的方法。

4.3 基于分类器的调优

SpMV 自动调优的调优空间包含了不同程序整体设计(比如 COO、CSR、ELL)、程序参数(比如 SELL 行条带被切分的间距、CUDA 内核占用的线程块和线程数量、循环拆开与平铺的次数)。基于分类器的调优本质上是矩阵的分类问题。其首先找到每一个矩阵对应最优的程序,然后将对应的程序作为这个矩阵的一个类别。而分类器可以通过矩阵的特征直接给出这个矩阵的类别所对应的程序。主流的分类器都基于机器学习(也有基于人工规则的分类器[39],但效果不如基于机器学习的分类器,它们只适合在比较小的矩阵中使用[54])。这类工作为大量的矩阵找到合适的程序整体设计和(或)程序参数。然后,将矩阵的特征作为模型的输入,将适合矩阵的程序整体设计和程序参数作为对应特征的标签,拟合二者之间的联系。常用的模型有:支持向量机(support vector machine,SVM)[55]、带权重的支持向量机(weighted support vector machine,WSVM)[44,48]、全连接神经网络[44,46]、决策树[46,56]、规则集[3,57]、XGBoost[44-45]、高斯朴素贝叶斯[46]、K 最近邻算法[46]、逻辑回归[46]。基于分类器的自动调优器的优点是较快的调优速度。自动调优器可以直接根据矩阵的特征直接给出合适的程序整体设计和程序参数。缺点是对矩阵类别的数量的限制。在分类器中,每个类别都是独立的,模型中的知识难以在相似的矩阵和相似程序参数选择之间泛化。这使得针对每一个类别的训练集都需要有一定的数据量。但是,找出每一个矩阵合适的程序需要通过实际运行 SpMV 程序来尝试各种程序整体设计和程序参数的组合。这个过程的开销很大,限制了训练集中矩阵的数量(已有工作的训练集一般包含了 3 000 ~ 10 000个矩阵),进而限制了矩阵类别的数量(已有的工作一般包含 3 ~ 10 个类别)。较低的类别数量增大了自动调优器在调优空间中的搜索步长,丢失了性能增长的机会。

4.4 基于迭代的调优

基于迭代的调优会在特定矩阵上尝试大量程序整体设计和程序参数的组合,并且从这些组合中选择性能最高的那一种。有 3 种方式在迭代过程中获得枚举出的程序的性能。第 1 种是直接运行的方式[25,58],即将程序直接编译、运行并得到其运行性能。这种方式的优点可以直接给出绝对性能(程序运行时间或每秒平均浮点计算次数),缺点是 SpMV 的实际运行开销比较大。第 2 种是使用采样的方式,即将矩阵进行采样,在采样矩阵上运行程序,用采样矩阵的性能估算完整矩阵的性能[59]。这种方式的优点在于在更小的矩阵下执行 SpMV 的开销更低,缺点在于这样的性能估计方式会有精度损失。第 3 种是基于性能模型的方式。这种方式不实际运行 SpMV 程序,而是根据程序和矩阵特征直接估计程序的性能。基于性能模型的方式又可以分为 2 种方式。第 1 种是基于人工性能模型的方式[60]。这

种方式根据研究者对 SpMV 执行流程和影响性能的因素的理解,根据硬件的性能(通过微基准测试得到)计算出每一个环节所需要的时间,并最终得到整体的性能或性能的概率分布。第 2 种是基于机器学习的方式。这种方式以矩阵与程序的特征作为输入,输出对应矩阵在程序上执行的性能。常用的模型有支持向量机回归(support vector regression, SVR)、全连接神经网络[52]和决策树[39](输出是程序性能的档位)。基于性能模型的优点是其完全不需要运行 SpMV 程序。缺点在于因为矩阵稀疏模式与程序性能的关联的复杂性,性能模型难以满足较高的精度,比如 WISE[41]和 WACO[52]。前者预测矩阵在特定程序上相对于 CSR 程序的性能档位;后者预测在特定矩阵上的 2 个程序(其模型的输入包含 2 个程序的特征)性能的大小关系。

整体上看,基于迭代的自动调优器的优势与劣势与基于分类器的自动调优器相反。通过迭代,各种程序整体设计与程序参数的组合都可以被一一尝试,这降低了在调优空间中搜索的步长,增加了找到更高性能程序的机会。它的缺点是不能直接给出合适的程序、需要较长的搜索时间。

5　稀疏编译器:TACO

TACO[22]是近些年被提出的一种针对稀疏张量计算的编译器。与常见的矩阵计算编译器一致,TACO 基于多面体模型,关注循环优化和自动并行。为了适应稀疏张量的计算,它在此基础上进行了微调。它将矩阵(张量)每个维度的索引进行压缩,这些被压缩的索引共同构成稀疏格式。TACO 提供了 6 种压缩索引(称为层次格式(level format),分别是 Dense、Compressed、Singleton、Range、Offset 和 Hashed。图 12 展现了由 TACO 的压缩索引生成的稀疏矩阵格式。

TACO 的代码生成过程从爱因斯坦表达式开始。SpMV 其对应的爱因斯坦表达式为 $\boldsymbol{y}(i) = \boldsymbol{A}(i,j) \times \boldsymbol{x}(j)$。它会生成 2 层循环分别遍历矩阵 i 和 j2 个维度,并在最内层迭代中将矩阵 $\boldsymbol{A}$ 的每一个元素按照其在 j 维度的位置与向量 $\boldsymbol{x}$ 的对应位相乘,并最终按照其在 i 维度的位置将数据归约到 $\boldsymbol{y}$

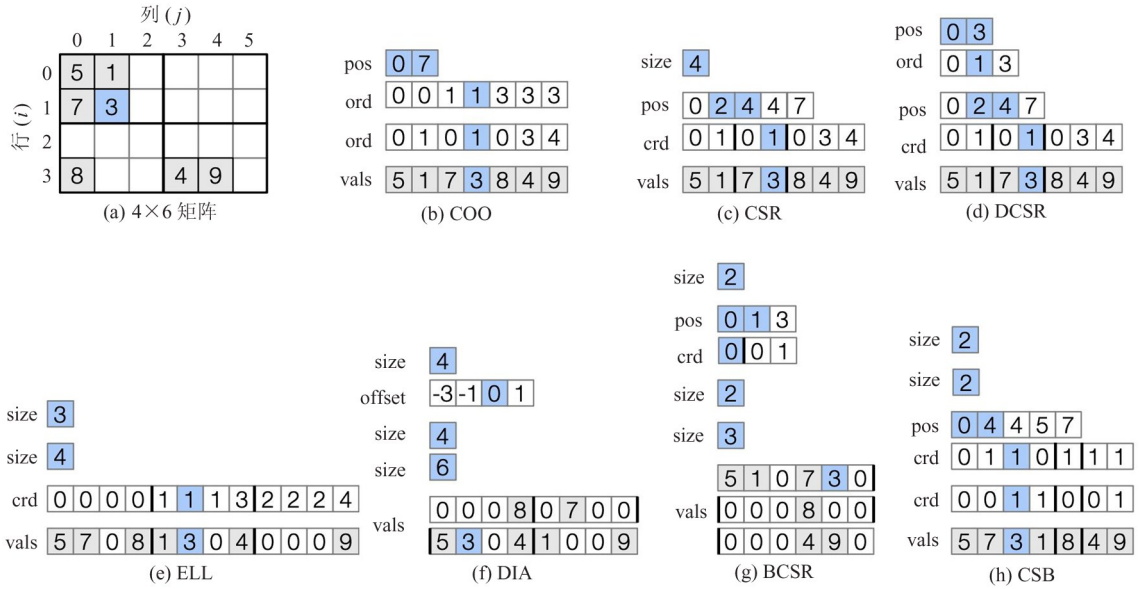

图 12　在 TACO 中构成的基本稀疏格式

的对应位置。通过将每一个维度的索引分别用一种方式进行压缩,并且将每一个维度的迭代转换为对应压缩索引的遍历,就可以生成矩阵(或张量)的稀疏矩阵格式和其对应的迭代方法,并且在迭代中完成张量计算。以图 12 中的 CSR 为例,其由 i 维度(行)的 Dense 和 j 维度(列)的 Compressed 索引构成。Dense 代表了 CSR 中行的数量(size),Compressed 代表了在非零元列号(crd)中每一行的起始

位置(pos)。在对应的 CSR 的卷积核中,这 2 个索引分别代表 2 层嵌套的循环,外层为 Dense,遍历所有的行(在此例中即 for (i = 0; i < 4; i ++));内层遍历根据每一行的偏移量遍历每一行的非零元(在此例中即 for(j = pos[i]; j < pos[i + 1]; j + +)),二者合起来遍历了整个矩阵。

6　自动程序设计器:AlphaSparse

AlphaSparse[24]是近些年提出的在 GPU 上的 SpMV 的自动程序设计器,它可以根据稀疏矩阵的特征由机器自动生成定制的稀疏矩阵格式和对应的 SpMV 算法。其生成的程序超过了已有的人工程序的范畴。AlphaSparse 认为现有的 SpMV 程序设计受限于人工的局限:矩阵的特征难以被观察;矩阵特征与适合的程序之间的关联难以被量化;为巨量的矩阵稀疏模式分别设计程序几乎不可能。所以,AlphaSparse 提出了绕过人工、由机器设计 SpMV 程序的新思路。

为了达成尽可能大的代码生成自由度,AlphaSparse 构建了 SpMV 程序的设计过程,并通过让机器产生设计过程来达到其设计程序的效果。

为此,AlphaSparse 抽象了稀疏矩阵格式生成的过程,并且将这个过程建模为一系列的步骤。图 13 是几个包含在 AlphaSparse 中的步骤组成的基础稀疏矩阵格式。COMPRESS 表示原始矩阵的零元被去除,非零元被压缩到矩阵一侧。ROW _ BLOCK 将被压缩之后的矩阵按行切块,从而生成 CSR 格式。在 CSR 的基础上,PAD 将每一个数据块的元素数量填充到一致,生成 ELL(ELLPACK)。在 CSR 的基础上,COL _ BLOCK 对矩阵进行列切块,生成 COO。

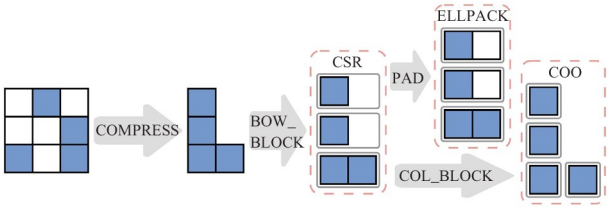

图 13　AlphaSparse 生成基础格式的过程

最终,一系列的 SpMV 的程序设计步骤(称作 Operator)被抽象到 AlphaSparse 中,这些步骤会同时修改矩阵描述和代码模版,在执行完所有设计步骤之后给出完整的稀疏矩阵格式和 SpMV 算法。此外,AlphaSparse 中的搜索组件会找出输入矩阵对应的最优的程序设计。因为 AlphaSparse 的细粒度建模产生了巨大的程序设计空间,所以为了获得高性能的程序,其需要数小时的搜索时间来生成适合某个矩阵的高性能的 SpMV 程序。

与自动调优器相比,AlphaSparse 产生的程序远多于预制的人工程序,从而可以增加获得更优程序的机会。与稀疏编译器相比,AlphaSparse 专为 SpMV 设计,它的程序表达空间不仅仅包含了一般稀疏计算中常见的迭代空间和索引压缩,也包含了数据排布、细粒度并行调度和数据归约等诸多方面。

7　实　验

此节在 GPU 上对几条路径的代表性工作进行测试,比较它们的性能并分析它们获得如此性能的原因。

7.1　实验配置

测试平台　实验采用的硬件是 NVIDIA A100。A100 基于安培架构,包含了 6 912 个 CUDA 核心、40 GB HBM2 的显存(吞吐量达到 1.5 TB · s^{-1})、单精度峰值性能可达 19.5 TFLOPS。

测试数据集　实验采用的测试数据集是来自佛罗里达稀疏矩阵集[1]的 843 个中大型矩阵。考虑到过大的矩阵会带来过长的预处理和测试时间、过小的矩阵无法充分利用 GPU 的性能。

人工程序　人工程序的测试对象包括 5 种先进的 SpMV 程序:CSR-Adaptive[37-38]、ACSR[7]、CSR5[35]、Merge-based CSR[61]和 HYB。

自动调优器　为了尽可能测出当今自动调优的性能上限,本文采用了自己实现的自动调优器作为测试对象。这个调优器包含了先进的程序:CSR-Adaptive、ACSR、CSR5、Merge-based CSR、HYB,以及在 cuSPARSE 算法库中经典的程序:COO、CSR 和 ELL。它采用基于迭代的调优方式,依次执行所有的候选程序,并从中选出最优的。已有的自动调优

器没有被采用的原因是:(1)它们大多年久失修,难以在最新的硬件上运行;(2)它们大多只包含经典的程序,这些程序的性能不极致。

稀疏编译器　实验采用 TACO 作为稀疏编译器的代表。生成的 SpMV 程序采用了默认的层次格式和代码优化组合。

自动程序设计器　对于程序设计器 AlphaSparse,本文设定了 8 h 的最大搜索时间。

7.2　实验结果与分析

图 14 记录了参与实验的人工 SpMV 程序、自动调优器、稀疏编译器和自动程序设计器的性能。在人工 SpMV 程序中,ACSR 提供了 262.1 GFLOPS 的最高性能和 56.4 GFLOPS 的平均性能;CSR-Adaptive 提供了 87.5 GFLOPS 的最高性能和 21.3 GFLOPS 的平均性能;CSR5 提供了 280.2 GFLOPS 的最高性能和 67.1 GFLOPS 的平均性能;Merge-basedCSR 提供了 270.8 GFLOPS 的最高性能和 67.3 GFLOPS 的平均性能;HYB 提供了 254.4 GFLOPS 的最高性能和 40.6 GFLOPS 的平均性能;自动调优器提供了 280.2 GFLOPS 的最高性能和 82.4 GFLOPS 的平均性能;TACO 提供了 279.7 GFLOPS 的最高性能和 73.6 GFLOPS 的平均性能;AlphaSparse 提供了 396.7 GFLOPS 的最高性能和 115.1 GFLOPS 的平均性能。

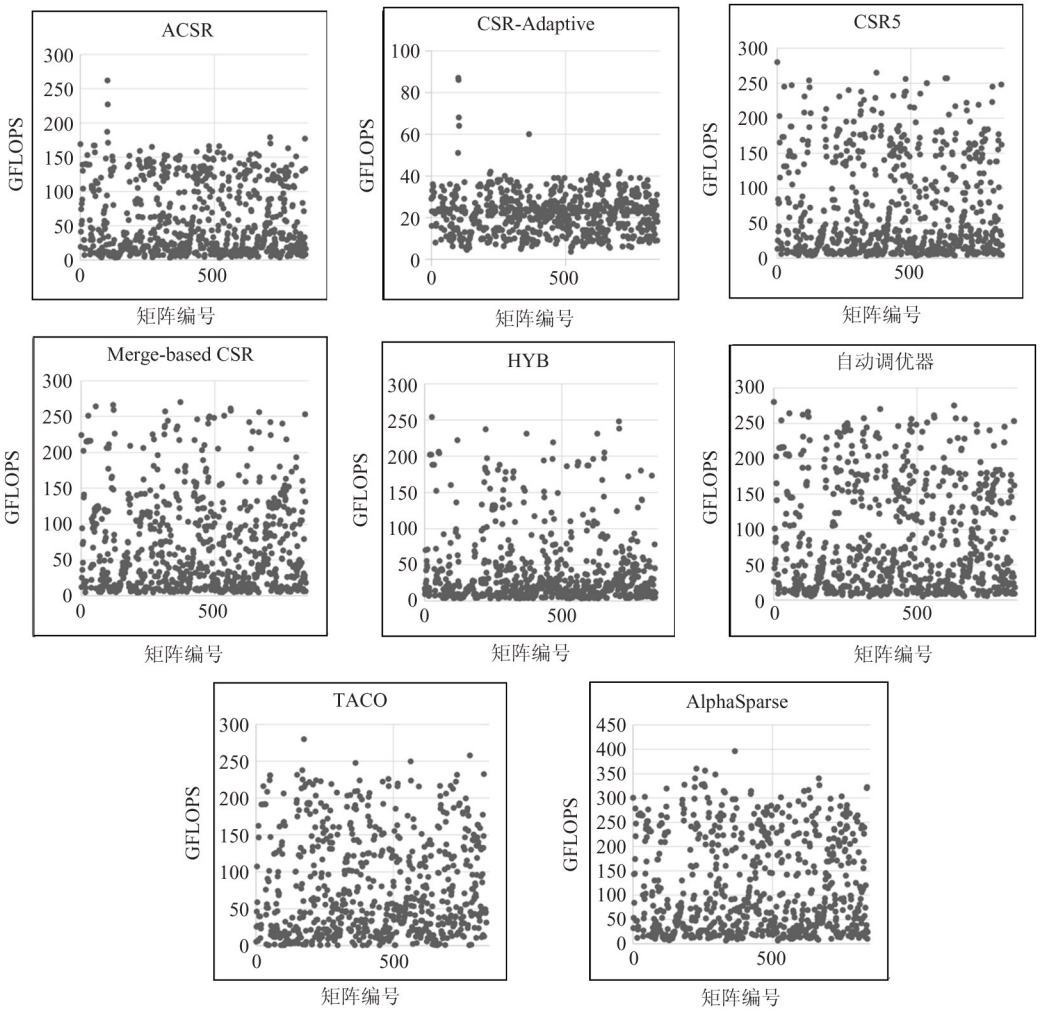

图 14　先进人工 SpMV 程序、自动调优器、TACO 和 AlphaSparse 的性能

在 4 条路线整体性能的横向比较中,自动程序设计器 AlphaSparse 由于构造了最大的 SpMV 程序设计空间,性能最高;稀疏编译器 TACO 为一般的张量计算设计,对 SpMV 程序缺乏针对性,故性能最低;自动调优器因为包含了多个主流的 SpMV 人工程序,所以性能不低于单个人工程序。

8 结 论

表 1 总结并比较了 SpMV 的 4 条路线,它体现了 4 条路线在 SpMV 性能、预处理时间、工程量和专用程度上的权衡。它们的性能从高到低分别是自动程序设计器、自动调优器、人工程序和稀疏编译器,其缘由已在第 7 节详细地讨论。在预处理时间上,人工程序只需要在运行前将输入稀疏矩阵格式转换为 SpMV 程序所需要的格式(如果二者一致,则完全没有预处理开销),所以预处理时间最低;自动程序设计器需要在一个很大的程序设计空间中作数个小时的程序搜索,预处理时间最高;自动调优器和稀疏编译器除了格式转换时间之外,还分别包括少量程序选择的时间和程序编译的时间,故它们的预处理时间在人工程序和自动程序设计器之间。在工程量上,稀疏编译器和自动程序设计器都包括相对复杂的程序建模和程序生成过程,故工程量最大;人工程序聚焦于数百行 SpMV 算法和格式转换代码,故工程量最小;自动调优器在人工程序的基础上需要少量的自动调优逻辑。在专用程度上,只有稀疏编译器针对一般稀疏张量计算,其他的 3 条路线都只能给出高性能 SpMV 程序。

表 1 高性能 SpMV 4 条路线的总结与比较

	SpMV 性能	预处理时间	工程量	专用程度
人工程序	中	低	低	高
自动调优器	较高	中	较低	高
稀疏编译器	低	中	高	低
自动程序设计器	高	高	高	高

回看 SpMV 程序设计的发展脉络,比较 SpMV 研究的 4 条技术路线优劣和权衡,可以得出以下 3 点结论。

(1)SpMV 研究经历了一条从人工到自动再到智能的发展路径。在 21 世纪之前,人工程序是提升 SpMV 程序性能的主要方法。但随着需要应对的矩阵稀疏模式越来越复杂、涉及的人工程序越来越多,SpMV 自动调优器开始被提出。早期的自动调优器,如 clSpMV,自动地为不同的矩阵尝试不同的程序,并选择出针对当前矩阵最优的程序,但是这种调优方式无法利用历史的搜索经验。之后,更新的自动调优器,如 WACO,使用机器学习来拟合历史的搜索经验以加速调优过程。

(2)SpMV 研究的技术路径都存在一个从萌芽到蓬勃发展再到成熟的发展周期。以 SpMV 的人工程序设计为例,经过数十年发展,如今的提升已经非常有限。经过测算,近 10 年来,人工程序的年均性能提升不到 5%。人工程序的设计和开发已经存在巨大的边际效应。相比之下,自动调优器、稀疏编译器和自动程序设计器正处在蓬勃发展的上升期。

(3)性能的提升往往也意味着预处理开销的增加。人工程序设计一般仅有格式转换的开销,自动调优器在格式转换开销的基础上引入了程序搜索的开销,AlphaSparse 因为引入了更细的程序设计空间,所以相比自动调优器又需要大得多的调优开销。

综上所述,SpMV 程序设计的相关工作存在如下几个趋势。

(1) 更加关注专家知识的抽象。在当今的人工程序设计中已经积累了大量的 SpMV 程序设计的专家知识,提出高质量的新知识的难度大于对于已有知识的充分利用。为了充分利用已有的知识,自动调优器、稀疏编译器和自动程序设计器作为不同类型的专家知识系统需要被进一步地发展。

(2) 更加关注细粒度的程序抽象。当前自动调优的研究充分关注了稀疏矩阵建模的不同方法,准确把握稀疏矩阵的细粒度特征可以增加调优的精度。但是,为了追求矩阵稀疏模式和程序设计的极致匹配,程序设计空间也需要被细粒度地构造,以扩大机器创造程序的空间。这种越发细粒度的程序建模在自动调优器、稀疏编译器和自动程序设计器的设计原则中已有所体现。自动调优器以一整个程序设计作为调优的粒度;稀疏编译器 TACO 关注到程序内部的迭代空间;自动程序设计器不仅仅关注程序内部的迭代空间,也关注程序的归约方法。在将来,为了给机器赋予更高的创造力、创造出更高性能的程序,更细粒度的建模是个趋势。

(3) 更加关注智能化方法的使用。自动调优和自动程序设计所带来的预处理时间需要用更加智能化的方法来降低。在更细、更大的程序设计空间中找出最适合某个矩阵的程序设计,需要精度更高的矩阵建模和更复杂的性能模型。以自动调优器的发展为例,传统的、粗放的对矩阵的人工特征统计或采样逐渐转向基于更精细的、更能捕获矩阵隐式特征的机器学习方法。稀疏编译器和自动程序设计器面临的情况也类似,而且更为复杂和紧迫。

(4) 更加关注 SpMV 程序设计底层机理的研究。自动调优器、稀疏编译器和自动程序设计器的产生都源于相关研究者对 SpMV 程序根本认识的进步。自动调优器的产生源于矩阵稀疏模式与程序的关联;稀疏编译器的产生源于程序迭代空间对性能的影响;AlphaSparse 的产生源于稀疏矩阵格式与算法设计的内在统一。即便在人工程序设计中,如果能提出更接近本质的、独特的 SpMV 程序设计方法,而不拘泥于裁剪和缝合现成的 SpMV 程序,也依旧能为这条成熟的路线带来进步。

参考文献

[1] DAVIS T A, HU Y. The University of Florida sparse matrix collection[J]. ACM Transactions on Mathematical Software (TOMS), 2011,38(1):1-25.

[2] ASANOVIC K, BODIK R, CATANZARO B C, et al. The landscape of parallel computing research: a view from Berkeley[R]. Berkeley: University of Berkeley, 2006.

[3] LI J, TAN G, CHEN M, et al. SMAT: an input adaptive auto-tuner for sparse matrix-vector multiplication[J]. ACM SIGPLAN Notices: A Monthly Publication of the Special Interest Group on Programming Languages, 2013, 48(6):117-126.

[4] YILMAZ B, AKTEMUR B, GARZARAN M J, et al. Autotuning runtime specialization for sparse matrix-vector multiplication[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2016,13(1):1-26.

[5] ELAFROU A, GOUMAS G, KOZIRIS N. Performance analysis and optimization of sparse matrix-vector multiplication on modern multi-and many-core processors[C]// 2017 46th International Conference on Parallel Processing. Bristol, UK: IEEE, 2017:292-301.

[6] ELAFROU A, GOUMAS G, KOZIRIS N. A lightweight optimization selection method for sparse matrix-vector multiplication[EB/OL]. (2015-11-08)[2023-09-23]. https://arxiv.org/pdf/1511.02494.

[7] ASHARI A, SEDAGHATI N, EISENLOHR J, et al. Fast sparse matrix-vector multiplication on GPUs for graph applications[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New Orleans, USA: IEEE, 2014:781-792.

[8] YAN S, LI C, ZHANG Y, et al. yaSpMV: yet another SpMV framework on GPUs[J]. ACM SIGPLAN Notices, 2014,49(8):107-118.

[9] SCHUBERT L K. Modification of a quasi-Newton method for nonlinear equations with a sparse Jacobian[J]. Mathematics of Computation, 1970,24(109):27-30.

[10] HAYAMI K, SAUTER S A. Application of the panel clustering method to the three-dimensional elastostatic problem[J]. WIT Transactions on Modelling and Simulation, 1970,18:625-634.

[11] DUFF I S. A survey of sparse matrix research[J]. Proceedings of the IEEE, 1977,65(4):500-535.

[12] KINCAID D R, GRIMES R G, YOUNG D M, et al. ITPACK-adaptive iterative algorithms using symetric sparse storage[C]//SPE Reservoir Simulation Symposium. Denver,USA: SPE, 1979:76-87.

[13] ZLATEV Z, VU P, WASNIEWSKI J, et al. Computations with symmetric, positive definite and band matrices on a parallel vector processor[J]. Parallel Computing, 1988,8(1-3):301-312.

[14] SAAD Y. SPARSKIT: a basic tool kit for sparse matrix computations[R]. Urbana-Champaign: University of Illinois, 1990.

[15] WHITE J B, SADAYAPPAN P. On improving the performance of sparse matrix-vector multiplication[C]//Proceedings the 4th International Conference on High-Performance Computing. Bangalore, India : IEEE, 1997: 66-71.

[16] TOLEDO S. Improving the memory-system performance of sparse-matrix vector multiplication[J]. IBM Journal of research and development, 1997,41(6):711-725.

[17] 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.

[18] KOURTIS K, GOUMAS G, KOZIRIS N. Exploiting compression opportunities to improve SpMxV performance on shared memory systems[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2010,7(3):1-31.

[19] 白洪涛, 欧阳丹彤, 李熙铭, 等. 基于 GPU 的稀疏矩阵向量乘优化[J]. 计算机科学, 2010,37(8):168-181.

[20] VUDUC R, DEMMEL J W, YELICK K A. OSKI: a library of automatically tuned sparse matrix kernels[C]// Journal of Physics: Conference Series. IOP Publishing, 2005,16(1):521.

[21] REMINGTON K, POZO R. NIST sparse BLAS: user's guide[R]. Gaithersburg: NIST, 1996.

[22] KJOLSTAD F, KAMIL S, CHOU S, et al. The tensor algebra compiler[J]. Proceedings of the ACM on Programming Languages, 2017,1:1-29.

[23] YE Z, LAI R, SHAO J, et al. SparseTIR: composable

abstractions for sparse compilation in deep learning[C]// Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM, 2023: 660-678.

[24] DU Z, LI J, WANG Y, et al. AlphaSparse: generating high performance SpMV codes directly from sparse matrices[C]// Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. Dallas, USA: SC, 2022:1-15.

[25] SU B Y, KEUTZER K. clSpMV: a cross-platform OpenCL SpMV framework on GPUs[C]// Proceedings of the 26th ACM International Conference on Supercomputing. Venice, Italy: ACM, 2012:353-364.

[26] MAGGIONI M, BERGER-WOLF T, LIANG J. GPU-based steady-state solution of the chemical master equation[C]//2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum. Cambridge, USA :IEEE, 2013:579-588.

[27] MONAKOV A, AVETISYAN A. Implementing blocked sparse matrix-vector multiplication on NVIDIA GPUs[C]// Proceedings of the 9th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation. Samos, Greece: Springer Berlin Heidelberg, 2009:289-297.

[28] BELL N, GARLAND M. Efficient sparse matrix-vector multiplication on CUDA[R]. NVIDIA Technical Report NVR-2008-004, Nvidia Corporation, 2008.

[29] CAO W, YAO L, LI Z, et al. Implementing sparse matrix-vector multiplication using CUDA based on a hybrid sparse matrix format[C]// The 2010 International Conference on Computer Application and System Modeling. Taiyuan, China: IEEE, 2010:161-165.

[30] 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):401-423.

[31] ANZT H, TOMOV S, DONGARRA J. Implementing a sparse matrix vector product for the sell-C/sell-C-σ formats on NVIDIA GPUs[R]. Knoxville: University of Tennessee, 2014.

[32] ASHARI A, SEDAGHATI N, EISENLOHR J, et al. An efficient two-dimensional blocking strategy for sparse matrix-vector multiplication on GPUs[C]// Proceedings of the 28th ACM International Conference on Supercomputing. Munich, Germany: ACM, 2014:273-282.

[33] ZHENG C, GU S, GU T X, et al. BiELL: a bisection ELLPACK-based storage format for optimizing SpMV on GPUs[J]. Journal of Parallel and Distributed Computing, 2014,74(7):2639-2647.

[34] OBERHUBER T, SUZUKI A, VACATA J. New row-grouped CSR format for storing the sparse matrices on GPU with implementation in CUDA[EB/OL]. (2010-12-10)[2023-09-23]. https://arxiv.org/pdf/1012.2270.

[35] LIU W, VINTER B. CSR5: an efficient storage format for cross-platform sparse matrix-vector multiplication[C]// Proceedings of the 29th ACM on International Conference on Supercomputing. Newport Beach, USA: ACM, 2015: 339-350.

[36] 顾越, 赵银亮. 基于 RISC-V 向量指令的稀疏矩阵向量乘法实现与优化[J]. 计算机工程与科学, 2022,44(1):1-8.

[37] DAGA M, GREATHOUSE J L. Structural agnostic SpMV: adapting CSR-adaptive for irregular matrices[C]//2015 IEEE 22nd International Conference on High Performance Computing. Bengaluru, India: IEEE, 2015:64-74.

[38] GREATHOUSE J L, DAGA M. Efficient sparse matrix-vector multiplication on GPUs using the CSR storage format[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New Orleans, USA: IEEE, 2014:769-780.

[39] LI M, AO Y, YANG C. Adaptive SpMV/SpMSpV on GPUs for input vectors of varied sparsity[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 32(7):1842-1853.

[40] DORFMAN R. A formula for the Gini coefficient[J]. The Review of Economics and Statistics, 1979:146-149.

[41] YESIL S, HEIDARSHENAS A, MORRISON A, et al. WISE: predicting the performance of sparse matrix vector multiplication with machine learning[C]// Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming. New York, USA: ACM, 2023:329-341.

[42] CHEN S, FANG J, XU C, et al. Adaptive hybrid storage format for sparse matrix-vector multiplication on multi-core SIMD CPUs[J]. Applied Sciences, 2022, 12(19): 9812.

[43] ASHOURY M, LONI M, KHUNJUSH F, et al. AutoSpMV: automated optimizing SpMV kernels on GPU[EB/OL]. (2023-02-11)[2023-09-23]. https://arxiv.org/pdf/2302.05662.

[44] NISA I, SIEGEL C, RAJAM A S, et al. Effective machine learning based format selection and performance modeling for SpMV on GPUs[C]//2018 IEEE International Parallel and Distributed Processing Symposium. Vancouver, Canada: IEEE, 2018:1056-1065.

[45] ZHAO Y, ZHOU W, SHEN X, et al. Overhead-conscious format selection for SpMV-based applications[C]//2018 IEEE International Parallel and Distributed Processing Symposium. Vancouver, Canada: IEEE, 2018: 950-959.

[46] CHEN S, FANG J, CHEN D, et al. Adaptive optimization of sparse matrix-vector multiplication on emerging many-core architectures[C]//2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems. Exeter, UK: IEEE, 2018:649-658.

[47] PICHEL J C, PATEIRO-LÓPEZ B. A new approach for

sparse matrix classification based on deep learning techniques[C]//2018 IEEE International Conference on Cluster Computing. Belfast, UK: IEEE, 2018:46-54.
[48] BENATIA A, JI W, WANG Y, et al. BestSF: a sparse meta-format for optimizing SpMV on GPU[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2018,15(3):1-27.
[49] 周飞燕, 金林鹏, 董军. 卷积神经网络研究综述[J]. 计算机学报, 2017,40(6):1229-1251.
[50] PICHEL J C, PATEIRO-LOPEZ B. Sparse matrix classification on imbalanced datasets using convolutional neural networks[J]. IEEE Access, 2019,7:82377-82389.
[51] CUI H, HIRASAWA S, KOBAYASHI H, et al. A machine learning-based approach for selecting SpMV kernels and matrix storage formats[J]. IEICE Transactions on Information and Systems, 2018,101(9):2307-2314.
[52] WON J, MENDIS C, EMER J S, et al. WACO: learning workload-aware co-optimization of the format and schedule of a sparse tensor program[C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. Vancouver, Canada: ACM, 2023:920-934.
[53] LIU B, WANG M, FOROOSH H, et al. Sparse convolutional neural networks[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Boston, USA: IEEE, 2015:806-814.
[54] NIU Y, LU Z, DONG M, et al. TileSpMV: a tiled algorithm for sparse matrix-vector multiplication on GPUs[C]//2021 IEEE International Parallel and Distributed Processing Symposium. Portland, USA: IEEE, 2021:68-78.
[55] BENATIA A, JI W, WANG Y, et al. Sparse matrix format selection with multiclass SVM for SpMV on GPU[C]//2016 45th International Conference on Parallel Processing. Philadelphia, USA: IEEE, 2016:496-505.
[56] SEDAGHATI N, MU T, POUCHET L N, et al. Automatic selection of sparse matrix representation on GPUs[C]//Proceedings of the 29th ACM on International Conference on Supercomputing. Newport Beach, USA: ACM, 2015:99-108.
[57] HOU K, FENG W, CHE S. Auto-tuning strategies for parallelizing sparse matrix-vector (SpMV) multiplication on multi-and many-core processors[C]//2017 IEEE International Parallel and Distributed Processing Symposium Workshops. Lake Buena Vista, USA: IEEE, 2017:713-722.
[58] GUO P, WANG L. Auto-tuning CUDA parameters for sparse matrix-vector multiplication on GPUs[C]//2010 International Conference on Computational and Information Sciences. Chengdu, China: IEEE, 2010: 1154-1157.
[59] ZHU G, AGRAWAL G. Sampling-based sparse format selection on GPUs[C]//2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing. Belo Horizonte, Brazil: IEEE, 2021: 198-208.
[60] 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.
[61] MERRILL D, GARLAND M. Merge-based parallel sparse matrix-vector multiplication[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Salt Lake City, USA: IEEE, 2016:678-689.

A survey of high-performance sparse matrix-vector multiplication programming

DU Zhen[*,**], TAN Guangming[*], SUN Ninghui[*]
([*]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)
([**]University of Chinese Academy of Sciences, Beijing 100049)

Abstract

Sparse matrix-vector multiplication (SpMV) are fundamental operations in scientific computing, graph computation, and data analysis. They have been an enduring and challenging research topic since the birth of modern computing. This paper systematically reviews the development of SpMV from 1970s and the representative work at each stage. It analyzes and compares four technical routes in this field: manual programming, automatic tuners, sparse compilers, and automatic programmers. These are the popular approaches today. On this basis, the paper makes predictions on the future trends of research on SpMV programs. It aims to provide useful insights to learners and researchers.

Key words: sparse matrix-vector multiplication (SpMV), sparse matrix format, auto-tuning, sparse compiler, high performance computing, parallel computing