

本科生毕业设计

|  |
| --- |
| 稀疏矩阵乘优化机制研究 |

|  |  |
| --- | --- |
| 院 系 | 计算机科学与技术 |
| 专业班级 | CS1907 |
| 姓 名 | 程潇鹏 |
| 学 号 | U201915139 |
| 指导教师 | 张宇 |

2023年06月01日

**学位论文原创性声明**

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名： 2023年06月01日

**学位论文版权使用授权书**

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保 密□，在 年解密后适用本授权书

2、不保密☑。

（请在以上相应方框内打“√”）

作者签名： 2023年06月01日

导师签名： 2023年06月01日

摘 要

理想

**关键词**：对象存储；磁盘阵列；存储策略；数据放置

Abstract

Fail

**Keywords:**

目 录

[摘 要 I](#_Toc10712)

[Abstract II](#_Toc26022)

[1 绪 论 1](#_Toc9723)

[1.1 课题背景 1](#_Toc11825)

[1.2 国内外研究现状 2](#_Toc3233)

[1.3 研究目的和主要内容 7](#_Toc21517)

[1.4 论文结构 8](#_Toc15811)

[2 相关技术基础 9](#_Toc18462)

[2.1 背景知识概述 9](#_Toc31810)

[2.2 方案可行性分析 11](#_Toc10540)

[2.3 开发环境以及开发工具 11](#_Toc15777)

[2.4 关键技术分析 12](#_Toc2325)

[2.5 本章小结 12](#_Toc15596)

[3 基于CSR5和DIA的SpMV算法 13](#_Toc3264)

[3.1 SpMV算法 13](#_Toc4460)

[3.2 基于CSR5的SpMV算法 13](#_Toc17503)

[3.3 功能模块设计 14](#_Toc20562)

[3.4 本章小结 14](#_Toc18177)

[4 XXX系统实现 15](#_Toc14479)

[4.1 过滤器实现 15](#_Toc13193)

[4.2 属性管理模块实现 15](#_Toc8798)

[4.3 数据迁移模块实现 15](#_Toc2690)

[4.4 本章小结 16](#_Toc30814)

[5 性能测试与分析 17](#_Toc12135)

[5.1 测试环境 17](#_Toc28195)

[5.2 功能测试 17](#_Toc5464)

[5.3 系统界面 17](#_Toc2218)

[5.4 性能测试 17](#_Toc31319)

[5.5 本章小结 17](#_Toc28436)

[6 总结与展望 18](#_Toc28030)

[致 谢 20](#_Toc28306)

[参考文献 21](#_Toc32714)

# 绪 论

本章首先介绍了当前稀疏矩阵乘法研究的现状，然后分析了稀疏矩阵乘法优化在存储格式方向的技术产生以及发展现状，介绍了国内外稀疏矩阵存储优化相关的研究工作，并对本文研究内容的工作意义做出了说明。

## 课题背景

### 研究背景和趋势

稀疏矩阵向量乘(SpMV)是指将一个稀疏矩阵与一个向量相乘。稀疏矩阵是指矩阵中零元素个数远大于非零元素个数。稀疏矩阵向量乘法被广泛应用于高性能计算等领域，因此，寻求稀疏矩阵向量乘的优化很有必要。

稀疏矩阵运算效率受稀疏矩阵存储格式，稀疏矩阵向量乘算法等影响。2019年Torsten Hoefler发表的COSMA算法[1]，在绝大多数条件下可以接近并行矩阵乘法的通信复杂度的理论下界。因此对稀疏矩阵存储格式优化的研究逐渐受到重视。其中，单一存储格式在近些年有大量研究，混合存储格式的研究也开始出现。近几年的研究重点集中在混合存储格式上。

### 稀疏矩阵存储优化的途径

稀疏矩阵存储格式的优化思路主要是压缩存储空间，除了忽略为零的元素外，还包括用指针和索引代替非零元素的位置等。基于这种思想产生的压缩存储格式包括COO、CSR、ELL、DIA、CSR5等单一存储格式。

### 面临的问题和挑战

单一存储格式的研究逐渐遇到瓶颈，对于不同的单一存储格式，在处理具有不同特征的稀疏矩阵时，存储效率各不相同。如DIA存储格式在处理对角矩阵时存储效率很高，但是在处理分布均匀的稀疏矩阵时效率大幅下降[2]，如图 1‑1和图 1‑2所示：

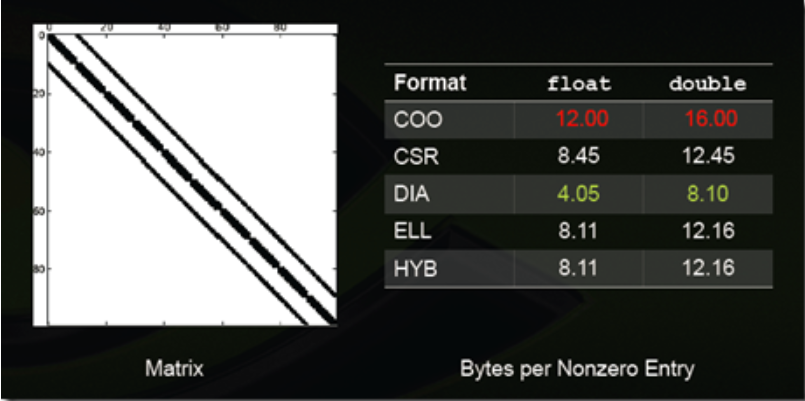


图 1‑1几种存储格式处理对角矩阵时的压缩效率

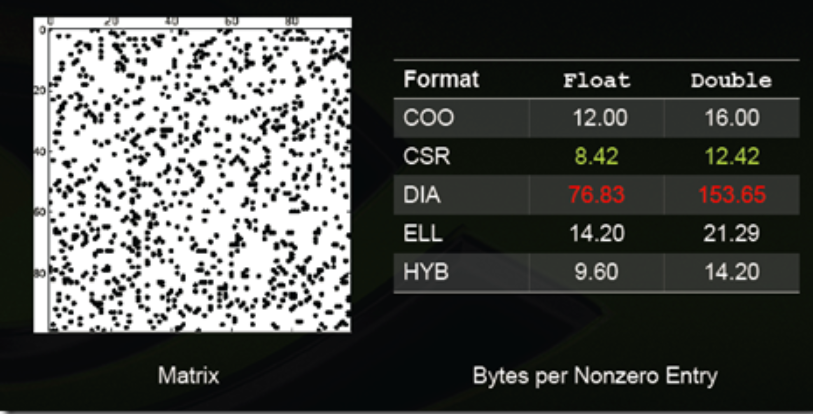


图 1‑2 几种存储格式处理随机矩阵时的压缩效率

压缩效率指每个非零元素平均占用的字节数，数值越小表示压缩效率越高。可见DIA存储格式在处理非对角矩阵时效率极低，而COO、CSR等格式非常稳定。

因此，如果采用单一存储格式，就不能在所有情况下都取得最优的压缩效率。因此，研究混合存储格式具有必要性。

## 国内外研究现状

### 单一存储格式

稀疏矩阵中非零元素个数远远小于矩阵元素总个数，因此可以采用压缩二维数组的格式来提高存储效率。

COO[4]：这是一种非常直观的存储格式，它的基本思想就是压缩存储。COO存储格式包含非零元素值和该元素值的行索引和列索引，并且非常容易由二维数组转换得到，然而这种格式非常不利于计算。COO存储格式常用于从文件中读取稀疏矩阵，martix market[3]采用的就是COO存储格式。如图 1‑3所示，其中COL表示列索引，ROW表示行索引，DATA存储非零元素。

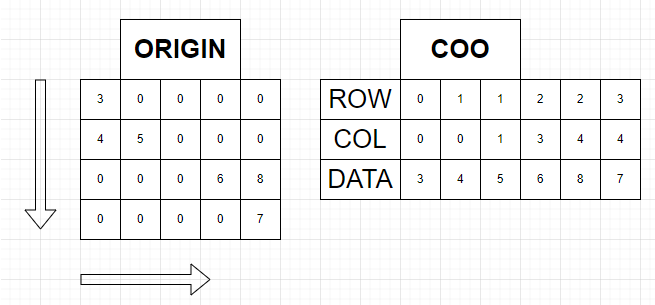


图 1‑3 COO存储格式

CSR：CSR存储格式是广泛流行的稀疏矩阵存储结构。这种格式要求元素按行存储，但是每一行元素可以乱序存储。CSR通过三个数组来存储稀疏矩阵，数组col\_ptr[i]存储data[i]的列索引，数组data存储非零元素，数组index\_ptr用于存储第i行的非零元素在数组data中的起始位置以及数组data中的结束位置，其中左边取闭区间，右边取开区间，长度为非零元素个数加一。CSR格式实际上是COO格式的压缩，它节省了存储空间，并且引入了row\_ptr以便于查找，方便计算。缺点是不直观，不易转换到稀疏结构。CSC存储格式是CSR存储格式的一种变种，它们的原理相同。

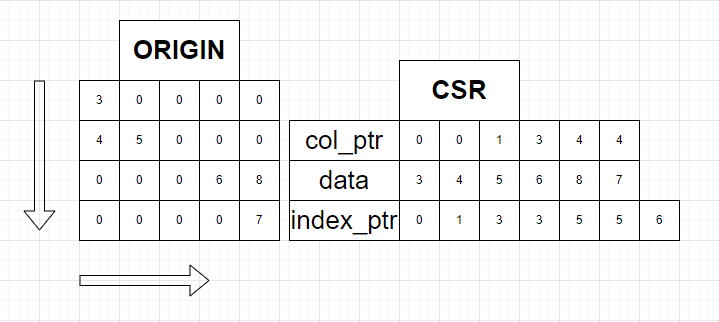


图 1‑4 CSR存储格式

ELL[5]：它需要用两个与原始矩阵行数相等的矩阵来存储，第一个矩阵存放列号，第二个矩阵存放非零元素，行号就是数据所在行。这种格式的缺点在于如果有一行过长，那么存储效率会大幅降低。因此ELL存储格式的压缩效率不稳定。

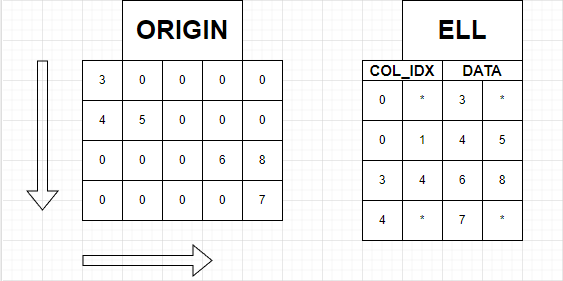


图 1‑5 ELL存储格式

DIA：即对角线存储法，设置一个新矩阵，用行表示行，用列表示对角线，忽略全为零的对角线，用\*补齐没有元素的位置。DIA存储格式在处理具有对角特性的矩阵时，具有很高的效率，但是处理随机矩阵时效率极低。因此DIA存储格式的压缩效率不稳定。

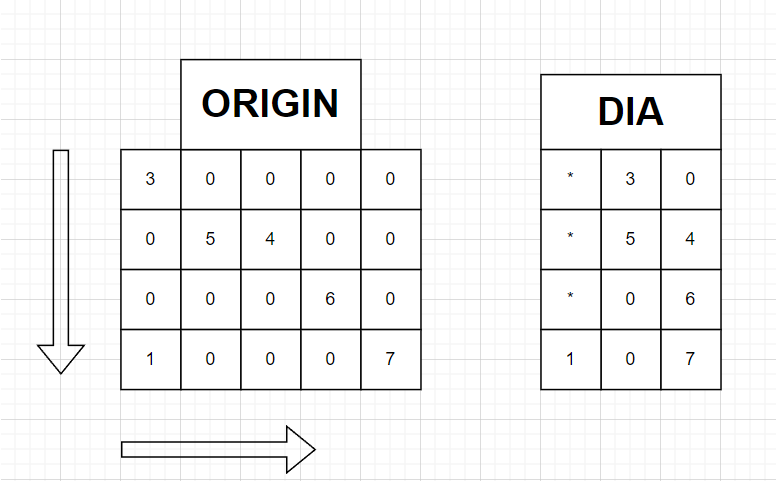


图 1-6 DIA存储格式

SELL[6]：SELL是为了解决ELL存储格式存在的问题的另一种存储格式，将ELL切割成每块有C行的矩阵快，再分别用ELL格式进行存储，从而减少零元素的扩充。下图 1-7是C取2时的SELL存储格式，前两行一组用ELL格式存储，长度为2，后两行长度为3.

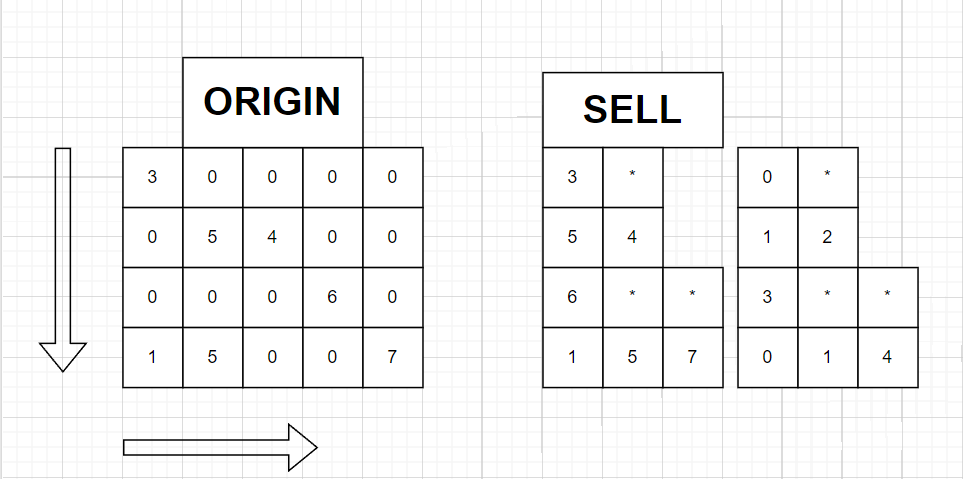


图 1-7 SELL存储格式

SELL-C-σ[7]：SELL-C-是SELL的变体，它在 个连续行内排序，通常选择为C的倍数。排序策略是按行长度由大到小排序，通过排序可以将长度相近的行放在一起，这样能进一步减少零元素填充占据的空间。SELL-C-σ存储格式在宽SIMD下拥有较好的性能，但是它同样有ELL类存储格式的通病：对于某一行非常长的特殊矩阵，它的存储效率并不好。

CSRL[10]：在CSR基础上，对稀疏矩阵中列下标连续的非零元素，存储首个非零元素的列下表和段长度。CSR存储格式在SpMV算法中表现不佳，是由于数据的空间局部性和时间局部性不好，造成大量访存导致的。然而，实际应用中的稀疏矩阵在一定程度上具有局部性特征，即下标相邻。CSRL存储格式可以有效利用这些局部性特征来提高其在SpMV算法中的表现。

CSR5[8]：将所有非零元素都划分成相同大小的2D块(除了最后一块)，记行数为，列数为。CSR5引入了块指针和块描述符作为辅助，并对CSR存储格式中的col\_idx数组和val数组都进行转置，当每个处理器处理一个2D块时，处理器的每个SIMD(单指令多数据流)进行合并内存访问。

块指针存储每个2D块中第一个元素在原始矩阵中的行索引。

CSR5的块描述符包含以下四个部分：

1. bit\_flag：该矩阵与col\_idx矩阵大小一致，指示对应位置的元素是否是一行的第一个非零元素。
2. y\_offset：长度与col\_idx矩阵列数一致，记录每一列的行索引偏移。第一列的行偏移固定为0，而后续偏移可以通过计算bit\_flag矩阵中该列之前所有列的T数量来获得。
3. seg\_offset：作为生成分段和的辅助阵列，可以在对应bit\_flag矩阵中向右搜索，计算全部为F的连续列的数量来获得。
4. empty\_offset：考虑这种情况，矩阵中有一行空行，那么在计算y\_offset时，行偏移就会发生错误。因此引入了empty\_offset，记录正确的偏移量。

CSR5是目前该领域效率最高的存储结构之一，并且对于任何稀疏矩阵结构都有很好的效果。

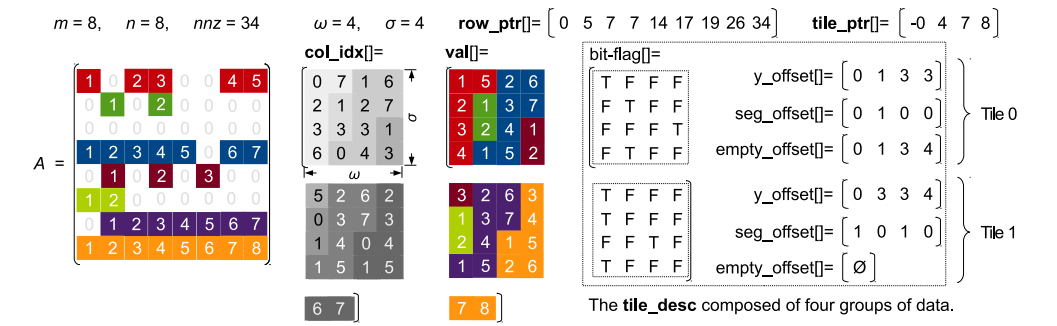


图 1-8 CSR5存储结构

### 混合存储格式

HYB[5]：为了解决ELL存储格式在某一行过长造成的臃肿问题，引入了COO+ ELL存储格式，即HYB存储格式，当ELL某一行过长时，用COO存储格式单独存储，提高存储效率。

HYB5[11]：HYB5是基于SELL-C-σ 格式和 CSR5 格式相结合设计出的一种混合稀疏矩阵存储格式。HYB5 按列将矩阵切割为两个部分，较规则的一部分由SELL-C-σ 来存储，相对不规则的一部分存储为 CSR5 格式。切割矩阵的位置由性能参数K决定。然而，论文作者并未给出K的确定方法，无法预先确定最优切割点的K值。

## 研究目的和主要内容

### 课题研究的意义

围绕稀疏矩阵存储格式开展的性能优化工作有助于提高SpMV(稀疏矩阵乘向量)的运算效率，在高性能计算领域有着重要的理论意义，在工业领域、生物信息技术领域、计算化学领域以及游戏领域有实用价值。

### 课题研究的内容

1. 对常用单一存储结构以及混合存储结构的综述；
2. 从CSR5存储结构出发，结合DIA等存储格式，设计一种新的混合存储结构；
3. 在CSR5源码基础上进行修改，编写混合存储结构代码；
4. 将混合存储格式与CSR5存储格式在不同测试集下的表现进行比较。

### 课题研究的目标

在本次课题的研究中，首先我们调研了多种存储格式，随后从CSR5存储结构出发，设计一种新的混合存储结构，编写代码并进行测试，最终目的是实现课题要求，预期新的存储结构在面对多种矩阵时综合存储效率高于CSR5。

## 论文结构

本文的主要内容如下：

第一章我们首先介绍了稀疏矩阵存储结构的研究背景和趋势，然后分析了稀疏矩阵存储结构当前存在的问题，介绍了国内外在稀疏矩阵存储机构优化方向的相关研究工作，并对本文的主要研究内容及工作意义作了具体说明。

第二章着重介绍了混合存储格式HYB和HYB5，分析了它们如何通过混合结构取得性能提升、以及它们的优缺点。并分析如何改进这些缺点，给我们在混合存储结构的设计上能够带来什么启发。并提出了新混合存储结构的设计思路以及可能存在的问题，并对问题的解决提供了初步方法。

第三章着重研究了SpMV算法以及它在CSR5和DIA存储格式上的具体实现，并给出了伪代码。

第四章给出了

第六章总结了所做的工作，并计划了下一步的工作以及展望了基于对象存储的前景。

# 相关技术基础

在绪论中，稀疏矩阵混合存储结构的重要性已经得到了论证，然而，什么样的混合存储结构能够提升存储效率，乃至提升SpMV性能，这是需要讨论的问题。在这一章节，混合存储格式HYB和HYB5将得到详细的分析。同时，方案的可行性将得到论证。改进的混合存储结构将会面临两个问题，分别是采用何种存储结构的判断问题和效率能否提升的问题，对于前者，已经提出了解决方案，对于后者需要在实验中验证。

## 背景知识概述

HYB存储格式的引入是为了解决ELL存储格式在某一行过长造成的臃肿问题，它的具体思路是当ELL某一行过长时，用COO存储格式单独存储，从而提高存储效率。如下图图 2-1 HYB存储格式，原始矩阵8个元素占用了20个单位的存储空间，然而，采用了ELL存储格式后，占用了32个单位的存储空间，存储效率反而下降，这是因为第二行过长。而采用HYB存储格式后，只占用了15个单位的存储空间，压缩效率得到有效提升。

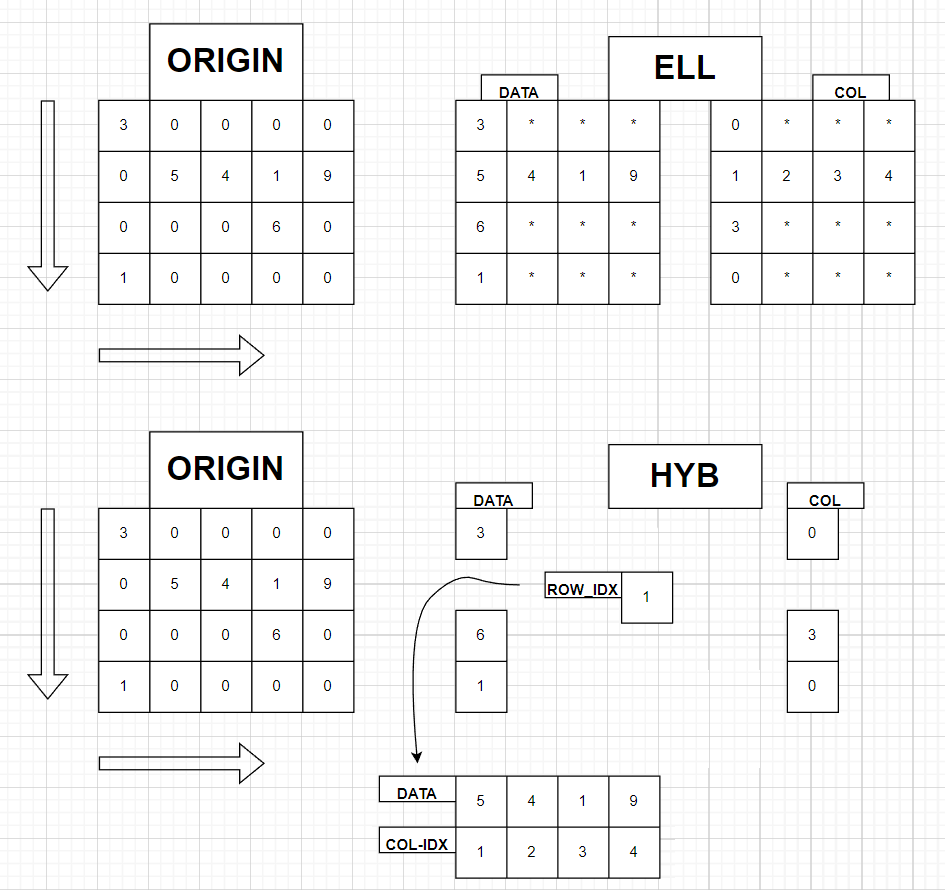


图 2-1 HYB存储格式

HYB5存储格式将CSR5和SELL-C-σ存储格式结合起来[11]，它将矩阵分为两个部分，较规则的一部分采用SELL-C-σ存储格式，其他部分用CSR5存储格式。HYB5引入了参数K，将矩阵划分为左右两部分，左侧采用SELL-C-σ，右侧采用CSR5存储结构。一般来说，SELL-C-σ处理的部分非零元更少，在这种情况下它的处理效率高于CSR5存储格式。而CSR5存储格式处理空行较多的稀疏矩阵时，可以采取压缩空行的方法提升存储效率。

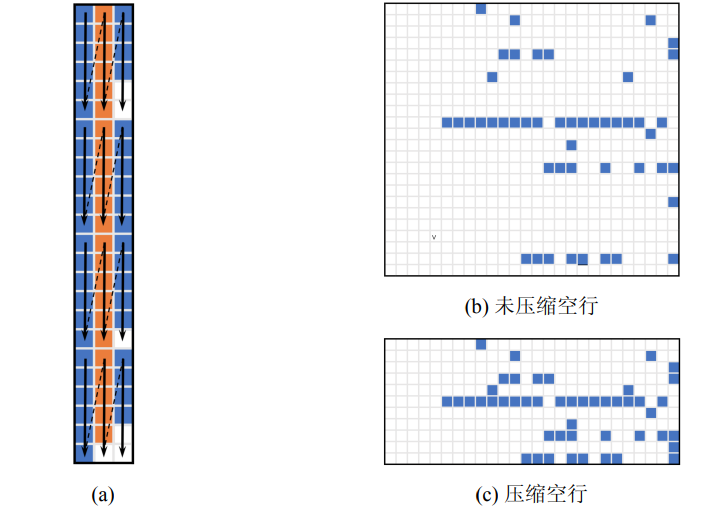


图 2-2 HYB5存储格式

## 方案可行性分析

没有一种存储格式可以在面对所有稀疏矩阵时都取得最优存储效率[15]，因此，采用混合存储格式是提升稀疏矩阵存储效率的突破点。在前人研究中，陈世钊等人提出的HYB5[11]存储结构能在选取最优的 K 值时，取得比CSR5存储结构更优的存储效率。这证明混合存储结构技术上是可行的。

Putnam模型估算成本如下：

CK值取3000，源码行数L预估500行，开发时间td取0.25年，按以下公式估算工作量：

(1)

计算得出的工作量约0.296年/人。

本文属于高性能计算领域，不存在伦理、社会等问题，实验所用工具均为开源或正版工具，未侵犯他人知识产权，不存在学术诚信问题。

## 开发环境以及开发工具

开发环境：GCC，CMake

开发工具：vs code

开发平台：ubuntu16.0.4 四核八线程处理器

## 关键技术分析

改进的混合存储结构主要采用如下思想：由于DIA存储格式在处理具有类对角性质的稀疏矩阵时能取得比CSR5存储格式更高的存储效率，而CSR5存储格式在面对不同稀疏矩阵时表现十分稳定，因此，可以设计一种判断机制，当稀疏矩阵具有类对角性质时，采用DIA存储格式，否则采用CSR5存储格式。

混合存储结构的设计主要要考虑以下两个问题：

1. 什么时候采用CSR5存储结构，什么时候采用DIA存储结构。
2. 优化节省的开销能够抵消分支判断增加的开销。

对于第一个问题，我们采用如下解决方案：引入一个参数p，对于稀疏矩阵，当所有非零元集中在以主对角线为中心的个对角线上时，认为该稀疏矩阵具有类对角性质。当一个稀疏矩阵满足类对角性质时，采用DIA存储格式。

对于第二个问题，我们将在后续实验中验证。

## 本章小结

本章主要分析了HYB和HYB5两种混合存储结构的原理，并由此得出了新的混合存储结构的设计方案。论证了设计方案的可行性，并分析了如何解决新混合存储结构方案潜在的两个问题。

# 基于CSR5和DIA的SpMV算法

SpMV算法，即稀疏矩阵的向量乘法。研究高效的稀疏矩阵存储格式目的之一，就是提升SpMV算法的效率。本章介绍了SpMV算法，通过基于CSR的SpMV算法对其进行了分析，并给出分别了基于CSR5存储格式和DIA存储格式的SpMV算法，并给出了在混合存储结构中使用SpMV算法的策略。

## SpMV算法

SpMV算法指稀疏矩阵与一个向量相乘，对于不同的稀疏矩阵，由于稀疏特性差异较大，SpMV算法在实现细节上也各有不同。SpMV算法可以代替稀疏矩阵和稠密矩阵的运算，从而节省开销和内存。

由于本文用于实验的稀疏矩阵格式为Matrix Market，存储格式为COO，在进行SpMV算法算法前，需要先由COO存储格式转换为CSR5存储格式或DIA存储格式，然后再参与SpMV算法，因此，格式转换步骤的开销也应该计入SpMV算法开销。

通过使用广泛的CSR存储格式来简单分析SpMV算法。

下面是基于CSR格式的SpMV算法伪代码：

|  |
| --- |
| Algorithm 1 基于 CSR 的 SpMV算法 |
| **INPUT** mtx格式矩阵A，向量  **OUTPUT** 向量y  1: **procedure** SpMV(A,x,y)  2: COO格式←读取mtx文件  3: CSR格式 A←COO格式  4: data\_m m ← A.m;  5: data\_r \*row\_idx ← A.row\_idx;  6: data\_c \*col\_idx ← A.col\_idx;  7: data\_v \*value ← A.value;  8: **for** i←1,2,...m **do**  9: data\_s sum ← 0;  10: **for** p←row\_idx[i]...row\_idx[i+1] **do**  11: data\_k k ← col\_idx[p];  12: sum += value[p]\*x[k];  13: y[i] ← sum;  14: return y;  15:**end procedure** |

在上述代码中，矩阵的数据结构可以是类或者结构体，它有五个成员，分别是行数m、列数n、存储数据的数组value，存储非零元素对应列数的数组col\_idx，以及可以指明一行在value数组中起始和终止位置的数组row\_idx。在执行SpMV算法时，本质上是执行，因此只需要知道行数即可，列数隐含在数组col\_idx的长度中。在8-13行的双层循环中，执行的就是SpMV算法的核心部分，索引i表示行数，矩阵A的每一行通过数组row\_idx寻找自己在数组value中的起始和终止位置，向量x通过数组col\_idx确定要与数组value中非零元相乘的元素，随后计算乘积之和。算法结果存储在向量y中。

上述算法还涉及了从COO存储格式到CSR存储格式的转换。转换过程的伪代码如下：

|  |
| --- |
| Algorithm 2 COO到CSR格式转换 |
| **INPUT** mtx(COO)格式矩阵A  **OUTPUT** CSR格式矩阵B  1:**procedure** TransCOOToCSR(A)  2: **for** i←1,2...A.value.length **do**  3: B.value[i] ← A.value[i];  4: **for** i←1,2...A.col\_ptr.length **do**  5: B.col\_idx[i] ← A.col\_idx[i];  6: **for** i←1,2...A.m+1 **do**  7: B.row\_idx[A.row\_idx[i]]++;  8: **return** B;  9:**end** **procedure** |

COO存储结构的数据结构较为简单，包含行数、列数、非零元数组value、非零元对应的行号数组row\_idx和列号数组row\_idx。从COO存储格式转换到CSR存储格式，value数组和col\_idx数组保持不变，只将COO存储格式中的行在矩阵中的索引转换为在value数组中的索引即可。

基于CSR存储结构的SpMV算法存在以下问题：对数组value和数组row\_idx显然是连续访问的，但是这两个数组地址不一定对齐，因此无法进行合并访问提高访存效率，同时对数组x的访问是随机的，因此cache命中率不会太高。在不进行其它优化的情况下，上述算法时间复杂度为。

## 基于CSR5的SpMV算法

在讨论基于CSR5的SpMV算法之前，需要先进行CSR存储格式到CSR5存储格式的转换，这一转换过程比COO到CSR的转换更为复杂，首先要确定两个参数和，然后再进行转换。

在进行转换时，CSR5存储格式会将稀疏矩阵中的非零元划分为长为，宽为的矩阵块，而这两个参数的确定取决于硬件。通常取决于处理器的SIMD执行单元大小，例如对于double类型的数据(64位)，在拥有256位SIMD单元的CPU上，取,即取4。CSR5希望让一个SIMD处理一个分块，通过并行处理提高SpMV算法效率。而参数的选取就较为复杂，可以总结为下列公式：

(1)

上述公式中nnz意为一行中非零元数量，row即为一行长度，r、s、t和u

是与硬件相关的参数，它们是一些边界值，用于防止过大或者过小[8]。CSR5的数据结构第一章已经介绍过，这里不再赘述。

CSR转换为CSR5存储格式的算法主要分为三部分，分别是生成每个块的首行指针tile\_ptr，转置排列成块的col\_idx和value以及生成每个分块的信息，包括bit\_flag、y\_offset、seg\_offset和empty\_offset。

生成tile\_ptr的思路比较简单，因为此时和的大小已经确定，第i个矩阵块的首行索引在数组row\_idx中的索引即为(i从0开始计数)，需要注意的一点是如果一个矩阵块中含有空行，需要将这个行索引取负值以便于后续计算。数组tile\_ptr元素类型应使用无符号数，防止第一个分块存在空行时无法取负值。

生成bit\_flag只需遍历一遍value矩阵块，判断元素是否是一行的第一个非零元即可，是则为TRUE，否则为FALSE。y\_offset记录value矩阵块每一列中相对该矩阵块首元素所在行的相对行数，只需要遍历前一列对应的bit\_flag中TRUE的数量即可。生成seg\_offset只需要计算该列右侧全FALSE的连续列的数量即可。empty\_offset只在矩阵块索引为负值时生成，它纠正错误的y\_offset，纠正方法是计算出bit\_flag中为TRUE的对于元素在原始矩阵中的索引，计算公式为：

(2)

通过该索引找出在数组row\_idx它在value数组中的索引即可。

完成格式转换后，即可开始基于CSR5的SpMV算法。

## 功能模块设计

磁

## 本章小结

本

# XXX系统实现

我

## 过滤器实现

### 过滤器注册数据结构

数

### 过滤器工作流程

加

## 属性管理模块实现

磁

### 属性管理数据结构

*t*

### 属性管理模块工作原理

磁

## 数据迁移模块实现

### 数据迁移相关数据结构

在

### 数据迁移请求处理

数

### 数据迁移速度调整

降

## 本章小结

本

# 性能测试与分析

## 测试环境

我们

## 功能测试

## 系统界面

不

## 性能测试

其

## 本章小结

本

# 总结与展望

致 谢

….

参考文献

1. Grzegorz Kwasniewski et al. Red-blue pebbling revisited: near optimal parallel matrix-matrix multiplication.[J]. CoRR, 2019, abs/1908.09606
2. Sparse Matrix Representations & Iterative Solvers, Lesson 1 by Nathan Bell. <http://www.bu.edu/pasi/files/2011/01/NathanBell1-10-1000.pdf>
3. Timothy A. Davis and Yifan Hu. 2011. The university of Florida sparse matrix collection. ACM Trans. Math. Softw. 38, 1, Article 1 (November 2011), 25 pages. <https://doi.org/10.1145/2049662.2049663>
4. Bell N, Garland M. Implementing sparse matrix-vector multiplication on throughput-oriented processors [C]. In SC. 2009.
5. Kincaid D, et al. ITPACKV 2D user’s guide [R]. 1989.
6. Monakov A, Lokhmotov A, Avetisyan A. Automatically Tuning Sparse MatrixVector Multiplication for GPU Architectures [C/OL]. In High Performance Embedded Architectures and Compilers, 5th International Conference, HiPEAC 2010,Pisa, Italy, January 25-27, 2010. Proceedings. 2010: 111–125. <https://doi.org/10.1007/978-3-642-11515-8_10>.
7. Kreutzer M, et al. A Unified Sparse Matrix Data Format for Efficient General Sparse Matrix-Vector Multiplication on Modern Processors with Wide SIMD Units [J]. SIAM J. Scientific Computing. 2014
8. Weifeng Liu 0002;Brian Vinter CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication. [J] IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences,2015
9. DavidR.Kincaid and ThomasC.Oppe. Recent vectorization and parallelization of ITPACKV[J]. Lecture Notes in Mathematics, 1990, 1457(1) : 58-78.
10. 刘芳芳,杨超.一种提高SpMV向量化性能的新型稀疏矩阵存储格式[J].数值计算与计算机应用,2014,35(04):269-276.
11. 陈世钊. 新型众核并行体系结构高效稀疏矩阵向量乘研究[D].国防科技大学,2018.DOI:10.27052/d.cnki.gzjgu.2018.000507.
12. 谢佩珍. 适合向量化的稀疏矩阵存储格式研究[D].国防科学技术大学,2016.
13. 杨世伟,蒋国平,宋玉蓉等.基于GPU的稀疏矩阵存储格式优化研究[J].计算机工程,2019,45(09):23-31+39.DOI:10.19678/j.issn.1000-3428.0053513.
14. 王志奇. GPU上稀疏矩阵向量乘积优化及最优存储格式预测方法[D].东北师范大学,2020.DOI:10.27011/d.cnki.gdbsu.2020.000347.
15. Bhuyan L N, Chong F, Sarkar V. Proceedings of the 29th ACM on International Conference on Supercomputing, ICS’15, Newport Beach/Irvine, CA, USA, June 08 - 11, 2015 [C/OL]. ACM, 2015
16. Arash Ashari, Naser Sedaghati, John Eisenlohr, Srinivasan Parthasarathy, and P. Sadayappan. 2014. Fast sparse matrix-vector multiplication on GPUs for graph applications. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14). IEEE Press, 781–792. https://doi.org/10.1109/SC.2014.69