**异构平台上稀疏矩阵计算问题的计算结构分析**

**一、异构平台上稀疏矩阵计算问题简介**

稀疏矩阵与向量乘（SpMV）是迭代法求解稀疏线性方程组的核心运算，且在现在流行的大数据计算、深度学习训练模型等各个工程大规模运算领域，所用到的数据普遍都是稀疏矩阵。因此，稀疏矩阵与向量相乘的高效率计算方法的研究一直都是计算科学方面的热点问题[1]。因为矩阵的计算性能和矩阵的存储结构密切相关，为了充分利用处理器的处理性能，我们需要结合系数矩阵中的非零元素分布特性，来选择合适的存储格式，在降低空间开销的同时还能降低问题的计算规模。

另外，为了实现稀疏矩阵的高性能计算，研究者常常利用不同平台来进行SpMV的计算研究，如向量处理机、多核CPU和GPUs[2,3]，以获得更高的计算加速比。目前并行计算SpMV方面的研究还主要集中在把现有的SpMV实现分别调度到CPU和GPU上。并行编程框架一直是高性能计算的研究热点，针对多核和多CPU的OpenMP并行编程技术[4,5,6]得到了广泛的使用，而对于典型应用领域产生的稀疏矩阵存储格式的重新设计，和计算任务分配调度的研究尚少。近年来，一些不同于CPU的异构处理器作为加速器提供了强大的并行计算能力，例如IBM的Cell处理器，FPGA，GPU等[7,8]。在这些加速器当中，GPU也因其拥有高能效和经济性得到了广泛使用，并且基于GPU的编程模型也较为成熟，再加上CPU的高频率且逐步多核化的特点，SpMV能够通过CPU+GPU这样的异构计算平台提升计算性能，且在平台上也有着较高的能效比。

本文将介绍一种在混合平台上的稀疏矩阵运算方法，它提出了针对于GPU多核特性的稀疏矩阵分割策略[8]，之后根据GPU中不同计算核心Knernel的计算能力来进行合适的任务划分。

**二、稀疏矩阵计算问题的分块求解算法**

**2.1稀疏矩阵的几种存储格式**

我们统一使用矩阵A来作为例子，说明不同存储格式的存储方法。A为的稀疏矩阵[9,10]，如下所示：

**COO存储格式**：采用三元组来存储稀疏矩阵中的非零元素，其中row为元素所在的行，column为元素所在的列，value为元素的值。例A的COO存储格式如下所示：

**CSR存储格式：**是一种行压缩存储方法。对于一个长度为k的数组按行顺序存储A中的所有非零元素值，另外一个长度也为k的数组对应存储每一个非零元素在行中的列号，还有一个长度为n+1的数组，其中元素i存放A中第i行之前的非零元素的数目。 A的CSR存储格式如下：

**ELL存储格式：**利用了一个的矩阵来存储数据，k为包含非零元素最多行的非零元素数目。ELL采用一个的数据矩阵EData来存储每一行非零元素的值，和一个的列偏移指针矩阵Offset来存储对应位置元素的列号。而对于矩阵中的零元素，Offset采用-1来填充。这一格式不适用于每行非零元素个数相差太大的稀疏矩阵。A的ELL存储格式如下：

**2.2稀疏矩阵分割策略**

要实现SpMV的并行计算，需要把稀疏矩阵分割为若干块，构建多个计算子任务，然后把这些子任务分配到不同的计算核心上实现并行计算。要想实现较好的并行效率，不但需要针对并行计算体系结构特征构建与之相适应SpMV的并行算法框架，还需要进行合理的任务分割和映射，以提高计算资源利用率。

总体来看算法有以下几步：1、构建稀疏矩阵的元素分布函数。2、利用构造的分布函数对矩阵进行分块。3、对分块后的矩阵进行重排序，使得非零元素个数相近的块排列在一起。4、针对不同处理核心的计算能力，给他们分配对应权重的块进行运算。

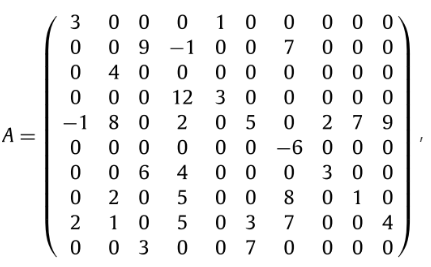
**2.2.1分布函数DF（Distribution Function）**

根据稀疏矩阵的分布模式，采用合适的存储格式，有助于提高SPMV的性能。我们可以用DF精确描述稀疏矩阵的分布模式，并从DF中得到稀疏分布的数值特征[11]。通过稀疏分布的数值特征，可以将合适的块体从稀疏矩阵中分割出来。假设我们有矩阵A，可以做如下划分：

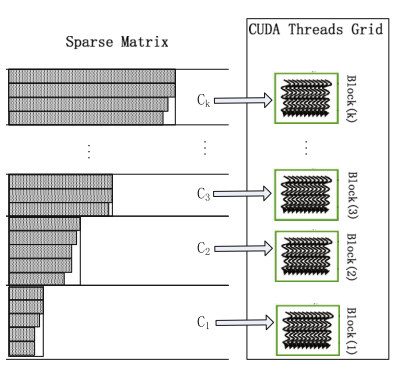
如上图所示，为N个M维行向量，他们一同组成了矩阵A。

定义为一个集合，该集合用于存储矩阵A的行向量，其中存放的向量具有m个非零元素。所以，。将这些集合称为行向量集（RVS）。我们可以基于此，于是把DF定义为：

映射方式是：，即集合中的向量个数定义为。所以显然有：。举例来看，假设我们有一个矩阵A如下：



则我们的划分可以写成：我们新添加一个矩阵B，用于存放的数目，则因为,,……我们把矩阵按分布函数DF划分后，每一行的非零元素个数一目了然，如下图所示：



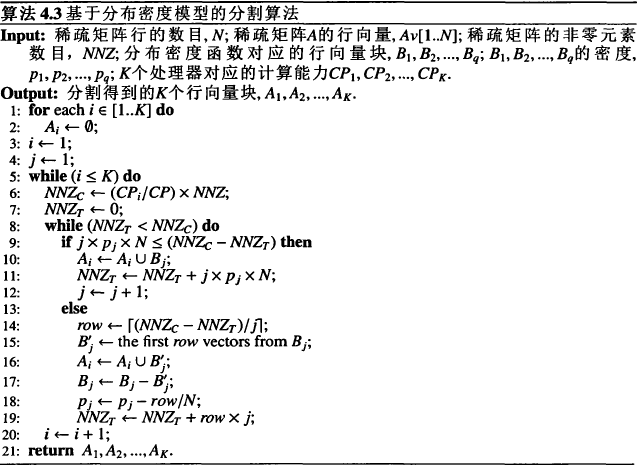
而采用ELL存储格式的存储密度与非零元素息息相关，这样的划分有助于我们对矩阵的行进行重排序，即把非零元素个数相近的行排在一起，有利于我们对矩阵的行分块进行存储，在GPU上能有效节省存储的空间。

**2.2.2行向量分割算法**

假设有K个处理器，每一个处理器的计算能力分别为。这个系统的总计算能力就等于。而如何把SpMV的计算任务分割成为K个子任务分别分配给这些运算核心上呢？主要有三种划分方法，分别是：基于行的数目划分[12]，基于非零元素个数划分和基于分布密度[11]来进行划分。我们主要介绍基于分布密度模型的分割方法。

对稀疏矩阵进行压缩，ELL压缩格式数据结构较为规则，比较适合在GPU上进行并行计算，在较多情况下性能较之CSR和COO要好[13]。虽然根据非零元素数目进行分割，但是分割出来的行向量块的非零元素数目偏差可能较大。

基于行分布密度是对基于非零元素个数划分的一个改进。设Ａ可以分割为q个行向量块,其中每一个行向量块拥有相同的非零元素数目。我们假设。A在分割成q个行向量块的时候，行的顺序就已经重排序了。这样行向量块有较高的密度，从而降低了ELL存储格式中的零元素的填充。具体算法如下：



这一种分割算法能够找到一种稀疏矩阵的分割，使得该分割得到的行向量块采用ELL格式存储需要的存储空间最少，也就是平均密度最高。对于一个大的计算任务分成几个子任务分别分配到不同的处理上进行同步计算，如果每个子任务在不同的处理器上计算时间相同，这样每个子任务将会同时完成，这种情况将会有最大的并行效率。因此一个稀疏矩阵按照这一方法分割为不同的行向量块分配到不同的处理器上进行运算SpMV，因为行﻿向量块的规模与处理器的计算能力是完全匹配的，因此每一个行向量块的计算时间将会是很一致，因此也有非常高的并行效率。

1. **稀疏矩阵计算密度分析**

**3.1 压缩格式的空间复杂度分析**

我们定义下述变量：为存储一个整数需要的存储空间，分别表示存储一个单精度和双精度所需要的存储空间。则三类存储格式的空间复杂度分析[14]如下：

**COO压缩格式的空间复杂度**：COO格式是用三个等长的数组来存储稀疏矩阵的非零元素，其中两个是整数数组，另一个是浮点数数组。数组的长度为NNZ。如果浮点数采用单精度存储，则对于稀疏矩阵采用COO格存储需要的存储空间为：

**CSR压缩格式的空间复杂度：**CSR格式也是用三个数组来存储稀疏矩阵。其中序列序号数组和值数组与COO格式一样，另外一个存储行指针的数组长度为N+1。因此CSR压缩格式的存储空间为：

**ELL压缩格式的空间复杂度**：ELL格式包含两个同等规模的二维数组构成，其中一个二维数组表示列序号为整数数组，另一个为表示值，为浮点数数组。二维数组的行数与稀疏矩阵A行数一样，列数为。因此ELL压缩格式的存储空间为：

如果稀疏矩阵存在较多的全是零的行，那么将会造成严重的存储空间浪费。我们采用先分割再存储的方式来存储，将能提高存储密度。

* 1. **行向量块分布密度与SpMV计算密度分析**

我们定义为从矩阵A分割出来的RVS，其中r为中的一个行向量。定义NNZ为r中的非零元素数目，那么中的非零元素数目为:[11,12]。定义的宽度为：。定义的行向量数目为。我们采用ELL来存储的话，需要存储的元素数目为：

所以可以定义的密度为中非零元素数目与作为稠密矩阵需要存储的元素数目之比，用表示，则可以利用以下公式进行计算：

定义零元素的填充比例为中需要填充的零元素数目与整体存储的元素数目之比，可以得到。假设稀疏矩阵A已经被分割为K个RVS，其中每一个行向量块都可以看做为的一个子矩阵。则这种分割的平均密度为：

我们以矩阵为例子，将含有q个非零元素的行在整个矩阵中出现的概率记作,那么,,,……,如果整体按照ELL格式来存储，则存储的密度为.443。如果按照上述6个RVS来存储，则平均密度为,无需进行零元素填充。而且后续根据分割的密度来分配计算任务，也将提高计算性能。

通过这一种存储格式，进行稀疏矩阵向量乘，我们可以得出它的计算密度：首先，一个稀疏矩阵中的待计算元素由两个矩阵存储，共需要读取的双精度浮点数共有个，而向量的维度为n，则共需读取n个双精度浮点数。计算结果写回以向量的格式，也需要n个双精度浮点数。则一共的访存次数为：

浮点操作数目共有次的乘法，则计算密度为：

**四、现有实现的性能概述与理想的计算结构**

**4.1 执行时间分析**

计算时间包括两个部分：数据从﻿主机传输到GPU的时间，和GPU上运算SpMV的时间。根据CUDA线程网格的并行计算模式，拥有每行等长的ELL存储格式比不等长的COO和CSR格式更适合GPU上进行并行计算[8,14]。

假设对于采用ELL格式存储的稀疏矩阵，一个行向量块传输到GPU的数据大小为。假设存储向量块的大小为，则DS的计算公式为：

而传输时间可以表示为：

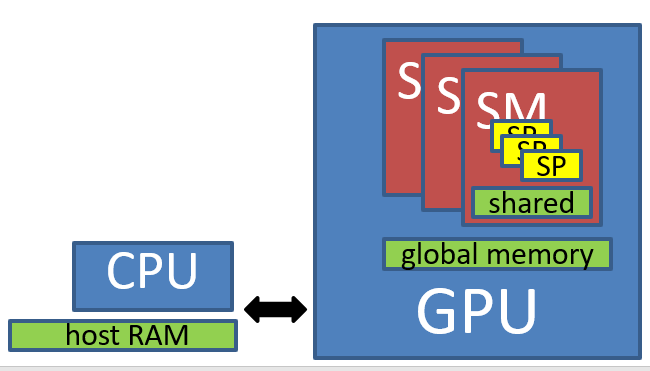
其中的为连接CPU和GPU的总线速率。针对ELL格式，采用密度分割算法得到的行向量块所占存储空间是最小的，因此传输时间也是最小的。对于一个稀疏矩阵,平均零元素填充率就是一种存储空间浪费率，存储空间浪费率的减少可以降低数据的计算规模，减少数据访问次数，从而可以提高计算效率。

**4.2 理想的计算结构与现有性能概述**

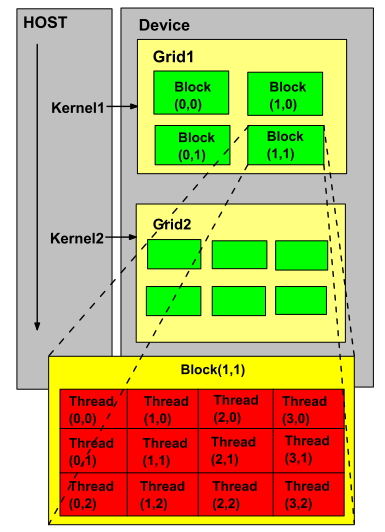
稀疏矩阵采用何种压缩格式较为合适与其稀疏模式相关，其主要是与稀疏矩阵中非零元素的分布密切相关。对于SpMV而言，其基本运算是稀疏矩阵中的一行与向量的积。对于一行与向量的乘，其计算性能与该行非零元素数目和排列相关，非零元素数目决定运算次数，非零元素排列影响数据访问。但是对于多行进行并行计算时，不同行的非零元素数目的差异会造成不同执行线程的计算任务的差异，对于每个线程可以独立调度的系统（多核CPU系统），而线程之间计算任务的不平衡一般不会因为同步而造成资源浪费，但是频繁的调度也会影响性能[16]。

而对于线程不能独立调度必须成束调度的GPU而言，同一束的线程之间计算任务的不均衡就会因为同步造成线程的等待而使计算资源闲置，从而使并行计算效率降低。另外分配给同一线程束的数据集的数据分布也会影响线程束的数据访问效率，比如数据块的对齐访问和连续访问等。因此针对GPU而言，SpMV的并行计算性能与行的非零元素分布关联较大[11,14]。

我们所需求的理想计算结构是CPU+GPU的异构平台，简单图示如下：



其中CPU起到任务分配调度的功能，而的GPU最好具有多个计算能力强大且相近的核心的GPU结构。CUDA将计算任务映射为大量的可以并行执行的线程，并且硬件动态调度和执行这些线程，GPU上的核心以线程网格（Grid）的形式组织，每个线程网格由若干个线程块组成。



因为流多处理器的计算效率依赖于分配到其上面运行的行向量块的密度，这样当我们提高稀疏矩阵的存储密度的时候，矩阵运算的计算性能就会得到更为显著的提高。

现有的实验中，在GPU和多核CPU的异构平台上，因为测试平台有两块GPU和两块CPU，不过CPU需要分配两个核心来分别控制两块GPU，另外６个核心可以参与计算。所以分割后，实验选用了10个测试矩阵，发现前两种分割算法得到的行向量块的非零元素数目与其分配的GPU和CPU的计算能力具有较好的匹配，且三类分割矩阵后的平均密度分别为11%,40%,61%[8,11,14]。在计算效率上，三类分割法单精度平均执行时间分别减少13.95%,37.94%和34.31%，双精度分别为13.10%,49.79%和50.03%。所以本文中基于分布密度的矩阵分割算法较现有的方法有较好的普适性，能够适应各种特征不同的稀疏矩阵。而且对于SpMV的计算加速效果显著，能够适应CPU、GPU以及其他加速器组成的异构计算平台。

**参考文献**

[1] Chen X, Ren L, Wang Y, et al. GPU-Accelerated Sparse LU Factorization for Circuit Simulation with Performance Modeling. IEEE Transactions on Parallel and Distributed Systems, 2015. 26(3):786-795

[2] 徐敬华,高铭宇,苟华伟,张树有,谭建荣.基于非规则分块压缩的3D打印稀疏矩阵存储与重构方法[J/OL].计算机学报,2019:1-12[2019-12-23].

[3] High Performance Computing; Reports Outline High Performance Computing Study Findings from School of Computer Science and Technology (Sparse Matrix Partitioning for Optimizing SpMV On CPU-GPU Heterogeneous Platforms)[J]. Computers, Networks & Communications,2019.

[4] 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(02):202-209.

[5] 刘芳芳,杨超,袁欣辉,吴长茂,敖玉龙.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.

[6] 杨世伟,蒋国平,宋玉蓉,涂潇.GPU上稀疏矩阵向量乘法优化策略[J/OL].计算机工程:1-16[2019-12-23].

[7] 尹孟嘉,许先斌,何水兵,胡婧,叶从欢,张涛.GPU稀疏矩阵向量乘的性能模型构造[J].计算机科学,2017,44(04):182-187+206.

[8] 阳王东. CPU+GPU异构平台上稀疏线性系统快速并行求解算法研究[D].湖南大学,2017.

[9] Shengen Yan, Chao Li, Yunquan Zhang,等. yaSpMV: yet another SpMV framework on GPUs[C]// Proceedings of the 19th ACM SIGPLAN symposium on Principles and practice of parallel programming. ACM, 2014.

[10] Jeljeli H. Accelerating Iterative SpMV for the Discrete Logarithm Problem Using GPUs[J]. 2014, 9061:25-44.

[11] K. Li, W. Yang, K. Li, Performance analysis and optimization for SpMV on GPU using probabilistic modeling, IEEE Trans. Parallel Distrib. Syst. 26 (1) (2015) 196–205.

[12] P. Guo, L. Wang, P. Chen, A performance modeling and optimization analysis tool for sparse matrix vector multiplication on GPUs, IEEE Trans. Parallel Distrib. Syst. 25 (5) (2014) 1112–1123.

[13] NVIDIA, cuSPARSE library, Tech. rep., March 2015.

[14] W. Yang, K. Li, Z. Mo, K. Li, Performance optimization using partitioned SPMV on GPUs and multicore CPUs, IEEE Trans. Comput. 64 (9) (2015) 2623–2636.

[15] A.N. Yzelman, D. Roose, High-level strategies for parallel shared-memory sparse matrix vector multiplication, IEEE Trans. Parallel Distrib. Syst. 25 (1) (2014) 116–125.

[16] Wangdong Yang, Kenli Li, Zeyao Mo,等. Performance optimization using partitioned SpMV on GPUs and multicore CPUs[J]. IEEE Transactions on Computers, 2015, 64(9):2623-2636.