

# Weekly Research Progress Report

Student: 从 兴 Date: 2024/2/26 - 2024/3/12

---

## List of accomplishments this week

- (1) 利用曙光超算和 vscode 部署 DCU 开发环境，完成 read\_mtx、coo\_to\_csr、spmv 等函数的编写和调试，在该过程中发现了一些问题。
- (2) 阅读 Optimizing High-Performance Linpack for Exascale Accelerated Architectures 论文
- (3) 阅读 TileSpGEMM: A Tiled Algorithm for Parallel Sparse General Matrix-Matrix Multiplication on GPUs 论文

## Paper summary

**Name:** Optimizing High-Performance Linpack for Exascale Accelerated Architectures

**Motivation:** 随着 GPU 计算速度的增加，导致 GPU 的运算时间开始小于 CPU 和 GPU 之间的数据传输时间，所以，目前的 GPU 加速库的性能瓶颈在于数据传输。因此，需要对 HPL 进行优化，以适应 GPU 的高速计算。

**Solution:** 本篇论文主要通过三个方面来对上述涉及到的问题进行了优化，分别是：在 CPU 上使用多线程方法计算 panel 分解阶段，在节点上对 CPU 核心进行进程时间共享，以及隐藏 MPI 通信这三个方面。

**Related to us:** 对单个节点内，多个 GPU 之间的通信进行优化，对于之后的工作，可以利用该篇论文所提出的思想—分成多块进行计算，使得上一块的计算时间来隐藏掉下一块各个 DCU 之间的 MPI 通信时间以及 CPU 与 DCU 之间的通信时间。可能会对之后的工作具有一些帮助。

**Name:** TileSpGEMM: A Tiled Algorithm for Parallel Sparse General Matrix-Matrix Multiplication on GPUs

**Motivation:** 现在的稀疏矩阵存储主要使用 CSR 存储格式来进行存储，这虽然可以带来较大的并行性，但是由于行-行之间非零元分布的不规则性，通常会导致负载不均衡、数据局部性不佳等问题，同时，由于是一维的结构，在进行矩阵和矩阵乘法时，会导致产生大量的中间结果，而如何尽可能的减少这些中间结果的存储和计算也是一个

值得优化的地方。

**Solution:** 本篇论文主要提出三个措施来对上述涉及到的问题进行了优化，分别是：包含掩码信息的平铺稀疏格式和 CSR 风格的自适应非零结构（本质是构建出新的 mask 矩阵，减少存储，加快计算）、优化各种输入稀疏块结构的自适应稀疏累加器（本质是分支判断条件）、用于计算块布局、符号和数值 SpGEMM 的三阶段优化框架（本篇文章的核心：提出两层 csr 存储结构）。

**Related to us:** 本篇论文的核心思想是提出了两层 csr-tile 存储结构（规则分块存储结构），这个结构可以减少中间结果的存储和计算，同时，由于是两层的结构，可以使数据的局部性得到提高，从而可以减少数据的传输时间。之后可以对这个结构进行更进一步的优化，优化思路为：利用结构体提高每个 tile 的数据局部性，加快 GPU 计算速度；利用平衡二叉树，来实现每个计算的负载均衡。

## Work summary

### 1. 开发部分

(1) 使用 vscode 连接曙光超算，部署开发环境

博客地址：[曙光超算的使用](#)

(2) 无法直接使用结构体进行整体的迁移

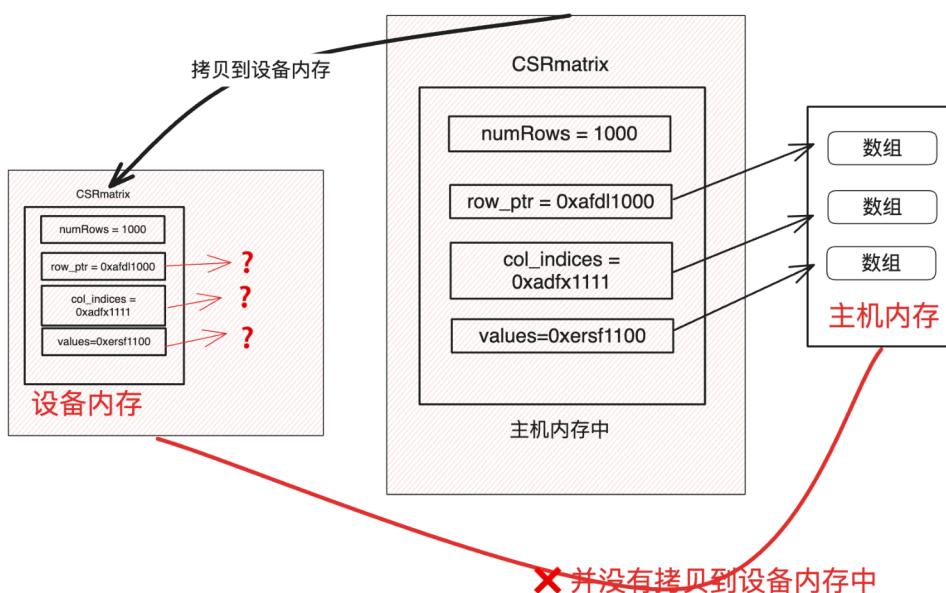


图 1 在使用结构体进行从 CPU 到 GPU 的迁移时，出现无效地址错误原因分析

**问题分析：**在 HIP 编程中，遇到了无效地址错误，检查错误的原因，是因为试图访问未正确分配或传递到设备（GPU）的内存。问题出现在如何处理 CSRmatrix 结构及其成员的内存分配和传递上。

CSRmatrix 结构体的定义中包含四个主要成员： numRows (矩阵的行数) , row\_ptr (行指针数组), values (非零值数组), 和 col\_indices (列索引数组)。

当我将 CSRmatrix 的一个实例 csr\_mtx 从主机内存复制到了设备内存 (d\_csr) 中时, CSRmatrix 内部的指针 (如 row\_ptr, values 和 col\_indices) 指向的数据实际上还在主机内存中 (也就是说, 在拷贝时, 只会拷贝结构体中的具体数据, 对结构体中指针指向的数据并不会进行拷贝)。这意味着, 当 GPU 试图通过这些指针访问数据时, 会遇到无效地址错误, 因为这些指针对设备来说没有意义。

**解决方案：**通过 CPU 运算得出三个数组所需要的内存空间, 然后在 GPU 上分配一整块内存, 然后, 对该一整块内存进行分割, 将分割后的数据内存地址拷贝到 GPU 对应的结构体内部定义指针上。

博客地址：在主机中定义复杂结构体指针无法通过内存拷贝直接到设备内存中

## 2. 阅读部分 (论文 1: Optimizing High-Performance Linpack for Exascale Accelerated Architectures)

博客地址：HPC 加速体系结构中 Linpack 优化

### (1) 区别：该论文 LU 方法与 SuperLU 中方法的区别

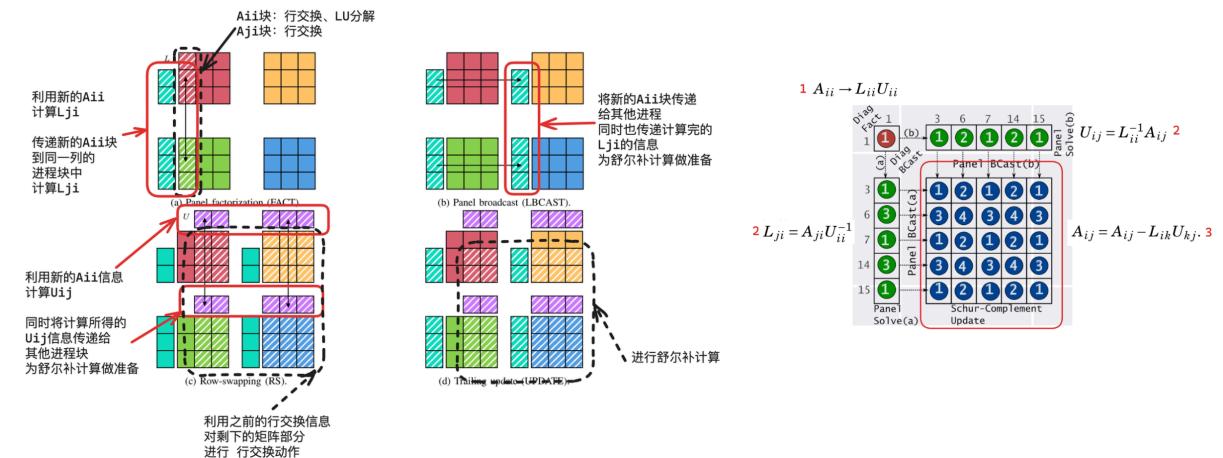


图 2 本论文 LU 过程与 SuperLU 过程对比

该论文中的 LU 方法在利用 pivot 方法时, 是分阶段进行处理的, 仅仅是对一小块对角线元素进行 partial pivot, 但是仍然需要和该对角线块下的各个列块中的行元素进行比较, 选出最好的行, 交换到需要分解的对角线块中。因此每次都需要进行各个进程块之间的通信, 来同步这些交换。

在 SuperLU 中, pivot 方法是对整个矩阵进行预先处理的, 即对整个矩阵进行 partial pivot, 然后选出最好的行, 交换到需要分解的对角线块中。不需要进行各个进程块之间的通信, 同步这些交换, 因为在开始进行 SuperLU 操作时, 矩阵已经准备完全。

## (2) 基本 HPL 主要存在问题

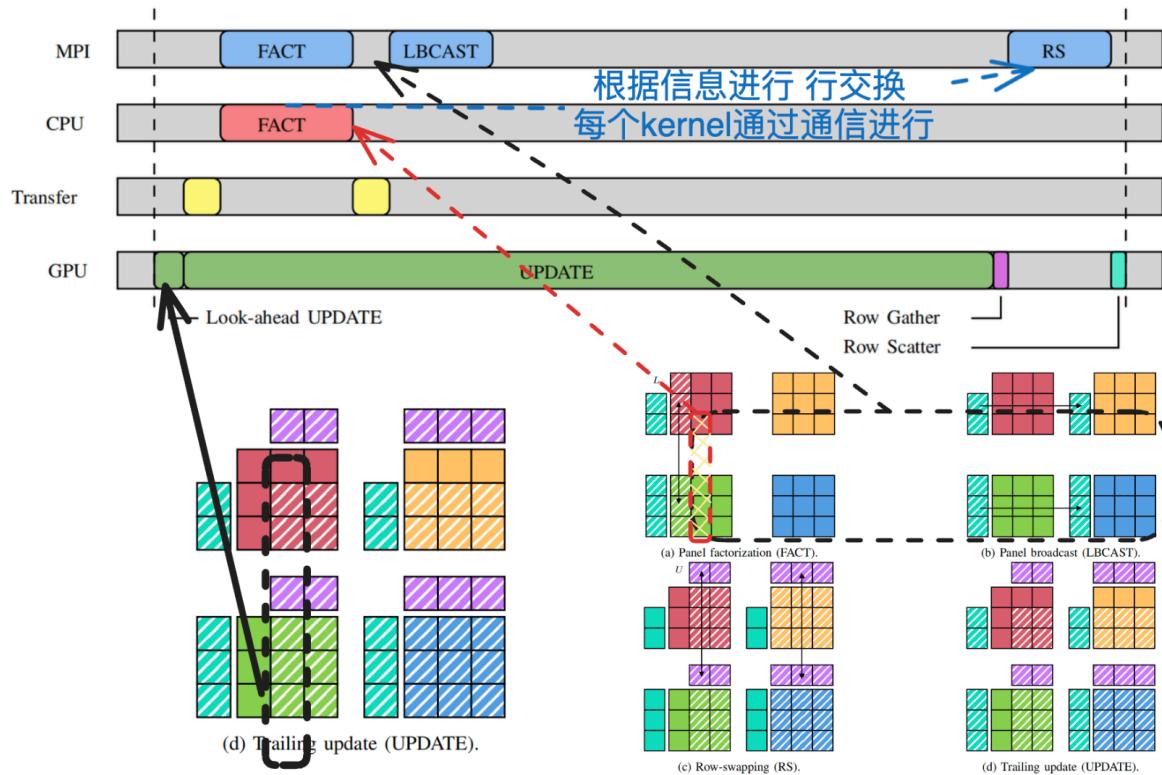


图 3 基础 HPL 的图解

在进行最后更新的阶段中，前部有一个阶段称为：look-ahead 阶段，通过这种措施可以提前计算出下一个计算中的对角块以及下面块列的结果，从而可以立马将这一个对角块和列传递给 CPU 进行 FACT、LBCAST 等操作，同时，GPU 可以对上述提到的最后更新阶段的后面数值继续进行更新。通过上述过程，就实现了一定的 overlap。

存在的问题是：虽然可以先进行 FACT、LBCAST 阶段，但是由于 pivot 的存在，仍然需要对其他的进程块进行通信，而这个 row-pivot 必须得等到所有数据经历过 Update 阶段后才可以进行，因此，在基本 HPL 中，这个阶段是无法 overlap 的。

## (3) 主要改进方法

基本思路是：考虑将更新矩阵块按列拆分为两部分，其中包含  $n_1$  和  $n_2$  列，称之为局部 A 矩阵的“左”和“右”部分。选择使  $n_1$  成为 NB 的倍数。用 UPDATE1 和 UPDATE2 表示在左侧和右侧部分上应用 UPDATE 阶段，并且类似地进行 RS 阶段处理。拆分更新的想法是利用一个区域上的 UPDATE 计算来隐藏另一个区域 RS 阶段 MPI 通信所需时间。也就是说，在另一个区域开始 UPDATE 之前必须先收集该区域 RS 阶段所需行数。与未拆分更新相同，在 look-ahead 上执行 UPDATE 阶段并将结果复制回主机以供 FACT 阶段和随后的 LBCAST 使用。在传输、FACT 和 LBCAST 执行期间，在加速器上计算

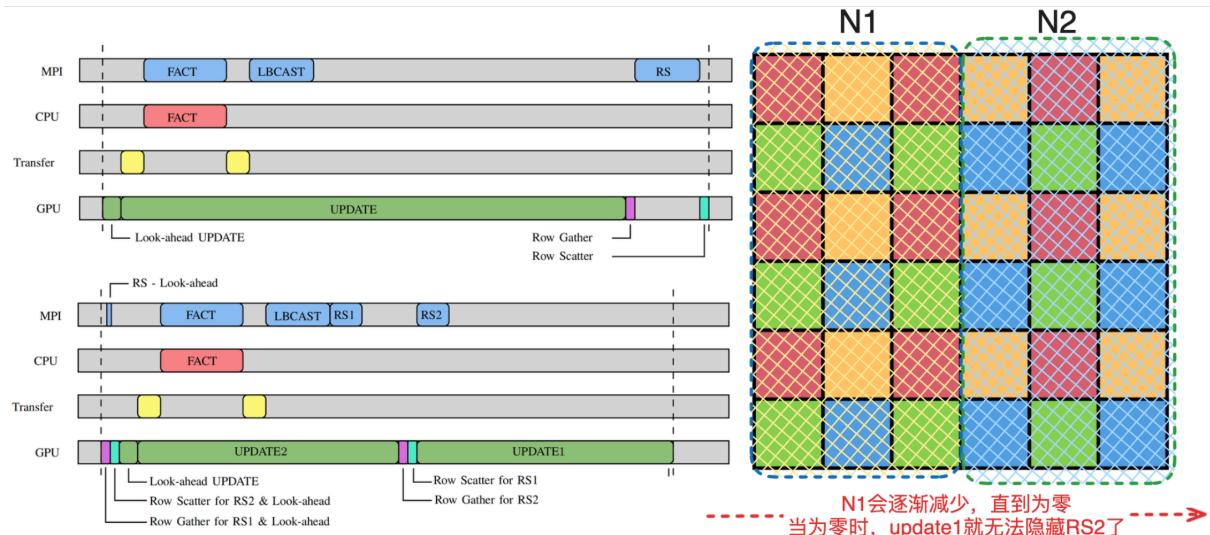


图 4 对 HPL 的改进措施

UPDATE2 阶段。但是，由于 A 左侧部分的行已经准备好用于通信，因此此时还可以执行 RS1 通信并且可以被 UPDATE2 隐藏起来。在 UPDATE2 之后，在准备下一个迭代中要用于 RS2 通信的 A 右侧部分中收集这些行。然后可以排队执行 UPDATE1 阶段，首先将已经通讯过得列返回到 A 中，并且 RS2 交流可以被这个本地计算隐藏起来。

3. 阅读部分 (论文 2: TileSpGEMM: A Tiled Algorithm for Parallel Sparse General Matrix-Matrix Multiplication on GPUs)

博客地址: Tile-GEMM

(1) 提出两层的 CSR-Tile 结构

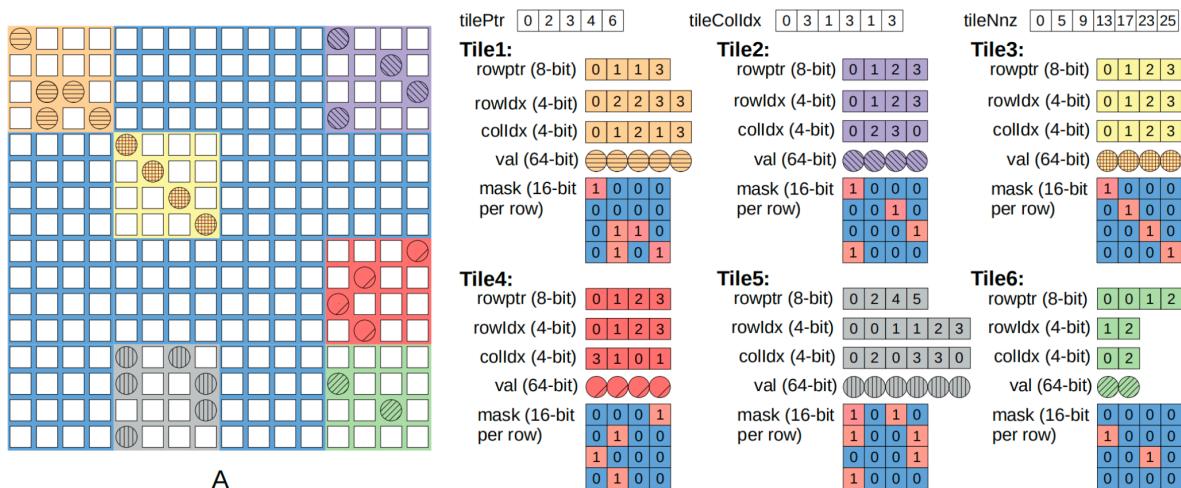


图 5 csr-tile 两成结构

瓦片结构包括三个数组 `tilePtr`、`tileColIdx` 和 `tileNnz`，代表瓦片的内存偏移量、瓦片列索引和稀疏瓦片中非零数目的偏移量。同时，每个稀疏平铺由五个数组组成：`rowPtr`、`rowIdx`、`colIdx`、`val` 和掩码。

## (2)SpGEMM 计算的三个阶段

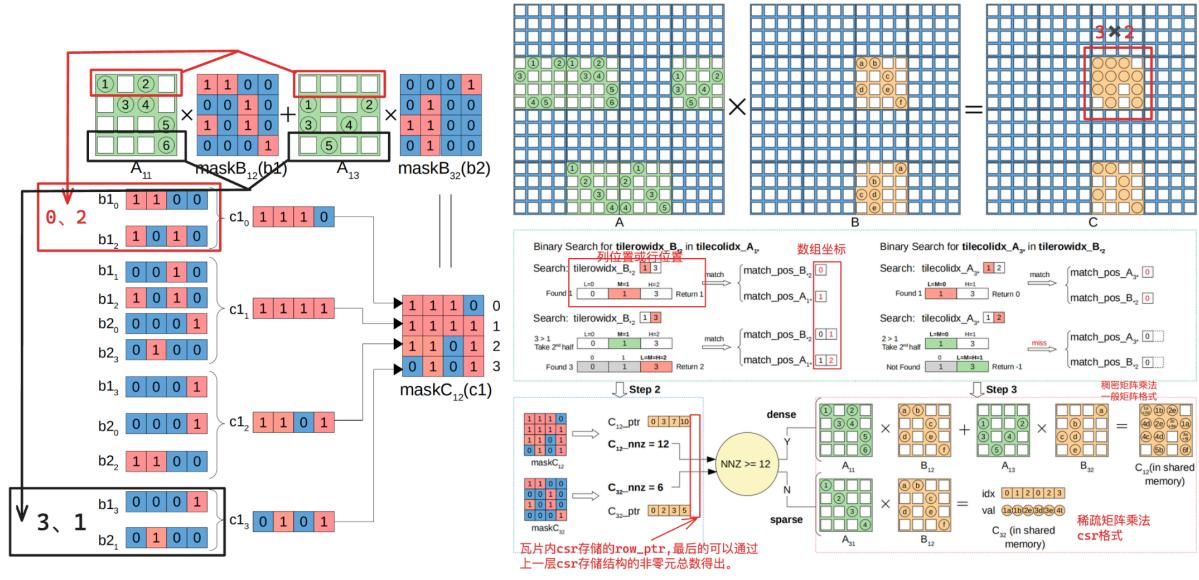


图 6 SpGEMM 计算的阶段图

- 得到整体瓦片结构数组（得出大矩阵每个可能具有非零元素的瓦片位置信息）
- 得到每个瓦片中非零元素的索引和个数（得出每个瓦片中非零元素的位置信息）
- 得到每个瓦片中非零元素的具体值（得出每个瓦片中非零元素的值）

## Next

- 阅读相关 tile 的论文
- 根据结构体的想法，在基本 kernel 上进行测试，观察性能效果