Weekly Research Progress Report

Student: 从 兴 **Date:** 2023/11/8 - 2023/11/22

List of accomplishments this week

为了完成相关课程报告的书写，熟悉 LaTeX 的使用

学习 MPI 规范，编写 Spmv 核的相关并行化 Demo

阅读最新的 PanguLU 论文

Paper summary

Name：PanguLU:A Scalable Regular Two-Dimensional Block-Cyclic Sparse Direct Solver on Distributed Heterogeneous Systems

Motivation：对于线性方程 $Ax = b$ 来说，在使用直接求解法进行完 LU 分解后，现在的大多是求解器（例如：SuperLU）使用 multifrontal 或 super-nodal 来聚合密集列，以使用密集 BLAS，从而为具有许多类似列结构的矩阵提供良好的可扩展性和性能。但是，在分布式内存系统中，它可能会对不规则矩阵造成性能问题。要么是聚集的相似列太少从而导致扩展性不足，要么是使用了太多的零填充来聚集更多的相似列从而导致计算效率降低。

Solution：提出一种新的直接求解器用于分布式异构系统的 $Ax = b$ 的求解——PanguLU，它使用规则的二维区块进行布局，并以块内稀疏子矩阵为基本单元，构建了一种新的分布式稀疏 LU 分解算法。由于存储的矩阵块是稀疏的，因此利用稀疏 BLAS 进行计算，进而避免不必要的填充，并优化稀疏属性，使得计算更加高效。

Related to us：为直接求解方法提供了新的思路，不在使用 dense 核进行计算，而是通过辨别稀疏模式，设计相应的稀疏算法，通过决策树选出对应稀疏算法再进行计算。

Work summary

稀疏矩阵 LU 分解三阶段：

稀疏矩阵的 LU 分解需要分为三个阶段来适应稀疏性：reordering、symbolic factorisation、numeric factorisation。

- reordering：减少填充非零点，保持数值稳定性。

- symbolic factorisation：确定矩阵 L 和 U 的结构。
- numeric factorisation：进行 LU 分解（**数值因式分解阶段通常包含大量浮点运算**，因此近年来许多 **直接求解器将数值因式分解阶段的并行性**作为单线程处理器、多线程处理器、异构处理器和分布式内存系统的主要优化方向）。

在分布式异构超级计算机上使用直接求解器的挑战与应对方案：

对于在具有异构处理器的大规模超级计算机上，使用直接求解器具有以下几个挑战：

- 如何平衡各个分布式处理器之间的**工作负载**
  - 设计一种静态块映射方案来**平衡负载**
  - 思路：计算每个任务的相应权重，权重高的任务会及时迁移到负载较轻的进程，从而平衡每个进程的工作量。
- 如何基于稀疏矩阵结构设计适当的**并行算法**，从而利用异构加速器的优势
  - 引入专用的稀疏 BLAS 设计，**使用不同的并行方法开发稀疏内核**
  - 设计**决策树**方法，根据稀疏矩阵的结构选择更快的稀疏内核
- 如何降低具有不规则稀疏结构依赖性的进程之间**同步成本**
  - 设计无同步调度策略，该策略使用无同步阵列，**允许分布式系统中的每个进程计算尽可能多的可执行稀疏内核**，并确保稀疏 LU 因子分解的正确性。
  - 减少同步开销来提高性能

块因式分解算法

为了提高性能，可以利用数据的空间位置使用**块 LU 因式分解算法**。该算法采用固定大小的分块法，对角线块从矩阵的左上角到右下角依次执行。相同颜色的区块可同时处理，以充分利用并行性。

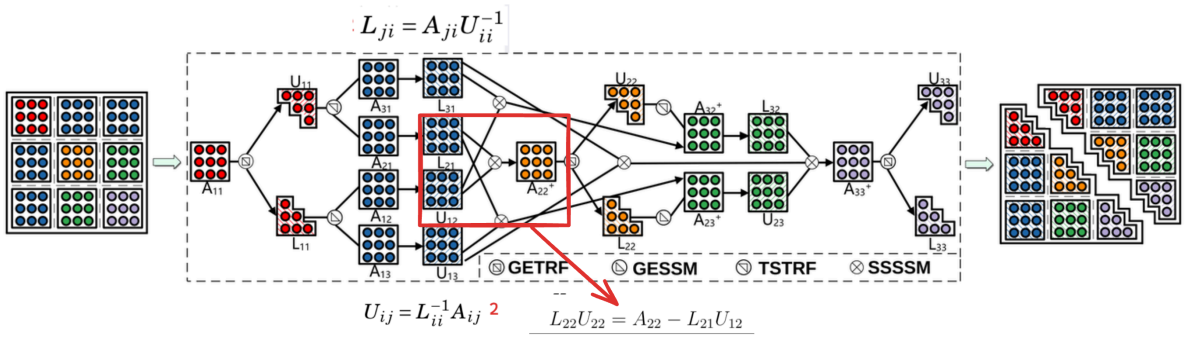

图 1 块 LU 因式分解算法示意图

上面这张图实际上就是对 SuperLU 文章中对特定矩阵生成 etree 分析后，各个矩阵

块的依赖关系的另一种表现形式。同样的，也是对角线区块从矩阵的左上角到右下角依次执行。相同颜色的图块可以同时处理，从而可以充分利用并行性。例如，当块 A_{11} 完成 LU 因式分解后，三角块 U_{11} 和 L_{11} 可以同时对 A_{21}、A_{31}、A_{12} 和 A_{13} 进行三角求解。然后，四个区块可以同时进行舒尔补码，更新 A_{22}、A_{23}、A_{32} 和 A_{33}。接下来以类似的方式计算对角线块，直到处理完最后一个对角线块，标志着块 LU 因式分解的完成。

LU 分解三阶段示例

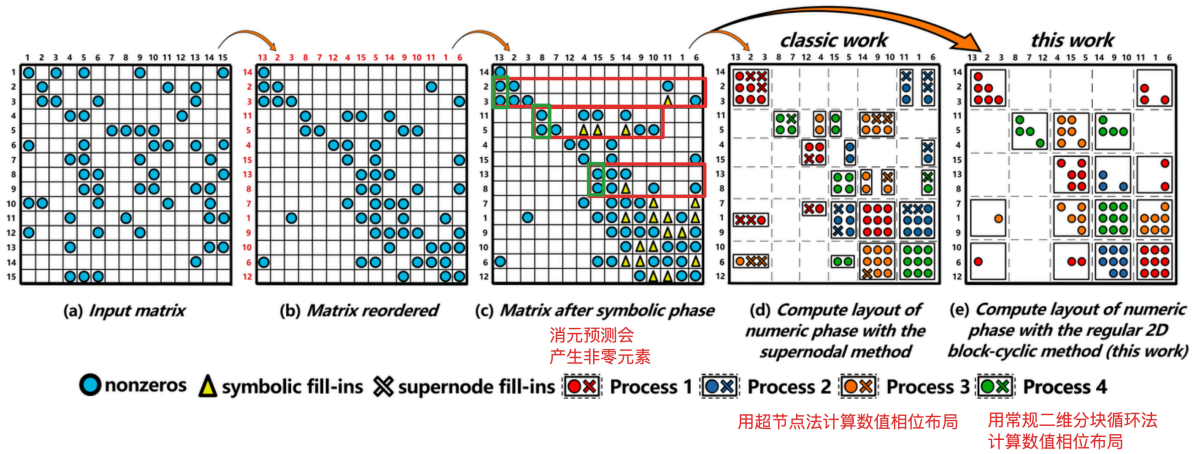

图 2 稀疏 LU 分解示例

- **重新排序**：某些行和列重新排序方法通常用于在符号因子填充阶段之前**最小化非零填充**。
- **符号分解**：在图 (c) 所示的符号相位中，引入了额外的非零填充来**确定三角形矩阵 L 和 U 的稀疏模式**，并提前为数值相位分配空间。这个阶段涉及创建因子分解所需的数据结构，但不执行任何数值计算。它包括**在不计算因子 L 和 U 的数值的情况下确定它们的结构**。这个步骤是通过假设要对主元所在列下一行元素进行消元时，由于主元所在行的非零元所对应列的下一行是零元素，而导致消元时产生 fill-in 现象，从而需要确定新的存储数据的结构。
- **数值计算**：在数值阶段，执行必要的浮点运算以确定 L 和 U 中的值。由于浮点运算的复杂性和广泛性，数字分解通常需要很长时间。此外，由于矩阵的稀疏性和分布不均匀，会导致数据访问模式的不均匀，从而增加了计算时间。

注：对于 Multi-frontal 方法和 Super-nodal 方法还得进一步阅读文献去理解。

传统方法的局限性

超节点法将具有相同非零模式的列集中在一起，作为密集块进行计算，但是这样可能会存在超级块大小不一致的问题。

可以看出，由超级节点生成的矩阵块可能非常不规则，这会影响计算和存储效率，

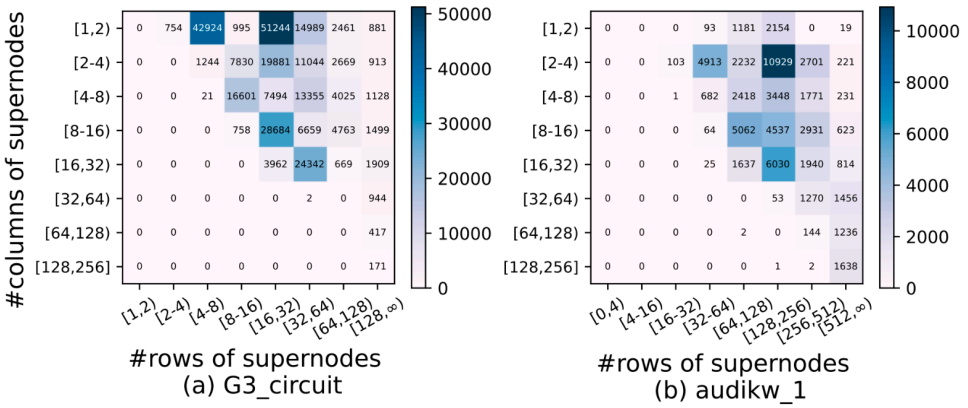

图 3 两个矩阵超级块大小的显著差异

并且不规则的结构使得难以在内核级别优化性能。

Multifrontal 算法和 Supernodal 算法**将矩阵划分为不均匀的密集块**，使用第 3 级 BLAS 例程进行计算，这可能会导致两个问题：

- 在形成密集块时可能会出现多余的零填充，从而增加额外的浮点运算。
- 在对密集块进行 GEMM 计算时，无法利用矩阵的局部稀疏性，从而可能导致性能下降。

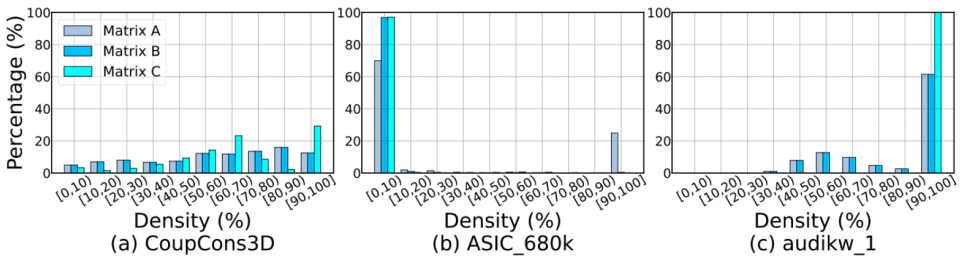

图 4 矩阵块密度的巨大差异

块矩阵密度是指：上面的算法将矩阵分为不均匀的密集块，说的是这个块中非零元素与所有元素的比值。

Density 是指：块矩阵密度在一定范围内的数量和这个稀疏矩阵可以被分为密集块的总数量的比值。

SuperLU_DIST 使用 level-set method 生成消除树，并将**树节点作为最小调度单元。树节点由多个密集的 BLAS 操作组成，每个层级之间都有依赖关系，需要在完成时同步**，因此会产生额外的同步开销。

下图 5 展示了 SuperLU_DIST（在 64 个英伟达 A100 GPU 上，从 1 到 64 个进程）在不同应用的六个矩阵上的同步成本比。如图所示，**同步成本随着进程数的增加而逐渐增加，在 64 个进程时，同步成本可占总计算时间的 60%**。因此，对于一些同步时间比例较高的矩阵，可以**考虑降低同步成本来探索优化空间**。

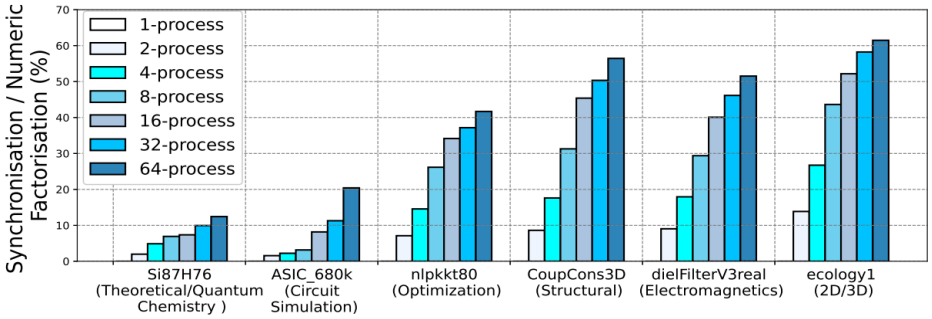

图 5 不同进程数量下的同步通信时间

PanguLU 的五个阶段

PanGuLU–新的用于分布式异构系统的直接求解器，包括五个主要的步骤:

1. **reordering**
   - 使用 **MC64 算法**来保持数值稳定性
   - 使用 **METIS 算法**来减少符号分解过程中的非零填充。
2. **symbolic factorisation**
   - 相比非对称剪枝法，PanguLU 的符号因式分解法采用了**对称剪枝法**，以降低计算复杂度并提高性能。
3. **preprocessing**
   - 预处理将矩阵划分为子矩阵块，并将它们发送给每个进程。
   - 块的大小根据矩阵阶数和符号因式分解后的矩阵密度计算得出，以平衡计算和通信。
   - 每个进程构建自己的双层稀疏结构。
4. **numeric factorisation**
   - 数值因式分解包含大量浮点运算，以确定 L 和 U 的数值。
5. **triangular solve**
   - 使用三角解法求解 $Ly = b$ 和 $Ux = y$ 的最终解 x，其中 x、y 和 b 为向量，b 为已知值。

文章的三个主要创新点

为了使 PanGuLU 能够更好地利用异构分布式系统的计算能力，提出了一种新的稀疏 LU 数值分解算法。它由三个主要的组件组成，以消除现有工作的缺点:

1. **规则的二维稀疏块结构**
   - PanGuLU 将原始矩阵分割成几个大小相等的块，并使用 (CSC) 格式存储它们。

- 为了平衡进程之间的计算负载，开发了一种映射方法来重新分配计算负载

2. **自适应稀疏 BLAS 内核**

- 在 PanGuLU 的数值分解中，由于矩阵块是稀疏的，我们开发了四个专用的稀疏核: **一般三角分解 (GETRF)、稀疏下三角解 (GESSM)、稀疏上三角解 (TSTRF)** 和**稀疏-稀疏矩阵乘法的 Schur 补 (SSSSM)**。对于具有不同结构和不同方法的稀疏块，稀疏 BLAS 的性能差异很大。因此，我们设计自适应稀疏 BLAS，根据输入稀疏矩阵的结构选择合适的算法，加快计算速度。

3. **无同步调度策略**

- 为了更好地减少分布式系统上的同步开销，PanGuLU 提出了一种**无同步调度控制策略**。该策略使用稀疏核作为最小调度单元。无同步阵列用于执行调度和维护正确性。
- 每个进程首先执行最高优先级的稀疏内核。

规则的二维稀疏块结构

PanGuLU 使用规则的二维分块，分块大小相等，按照**二维进程网格**分配给每个进程。每个进程使用**双层稀疏结构**来存储矩阵。

- 在块层，我们使用**基于块的 CSC 格式来压缩非零块**。
- 在每个块内，**子矩阵块也采用 CSC 格式存储**。

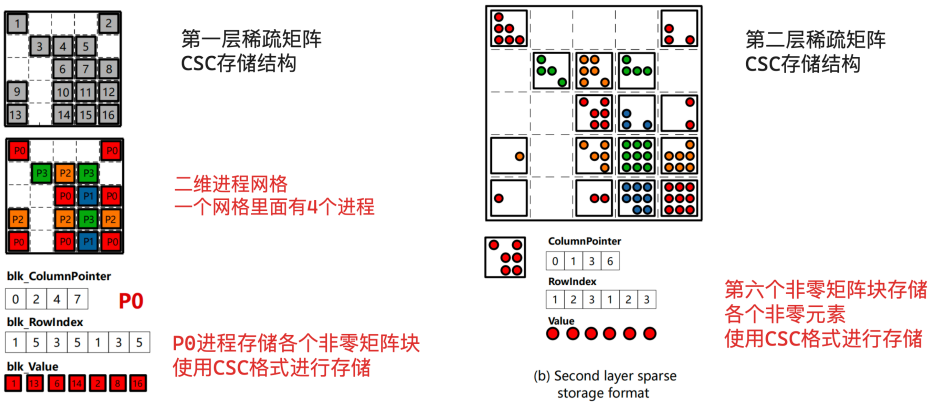

图 6 双层稀疏结构

子矩阵块由常规的二维分块法产生，连续编号的灰色块表示包含非零的子矩阵块。空块用空白表示。非空块分配给二维进程网格中的四个进程，每个进程用一种独特的颜色标识。每个进程只存储用于计算的重要子矩阵块，并利用子矩阵块在原始矩阵中的位置信息来促进通信。

子矩阵块内的非零点用 CSC 格式进行压缩存储，图 6 的 b 图展示的是第六个子矩

阵块的存储结果。从上节可以看出，子矩阵块的计算使用了四个主要内核。GETRF 将输入矩阵 A 因式分解为下三角矩阵 L 和上三角矩阵 U ，GESSM 和 TSTRF 执行下三角或上三角求解；SSSSM 执行舒尔补码运算。**相应任务的权重可通过计算内核的 FLOPs 得出**。

Next

- 将 PanguLU 未读完部分读完
- 继续学习 MPI 接口规范