Weekly Research Progress Report

Student: 丛兴 **Date:** 10/25/2023-11/8/2023

List of accomplishments this week: (工作成果列表)

这两周的阅读文献的工作主要基于前面两个周的 3D-SuperLU 论文的阅读工作进行展开：

向前，进一步了解 SuperLU 中提到的 Static Pivoting 的具体含义以及做法还有对 partial pivoting 的认识。

向后，学习当一个稀疏矩阵进行完 SuperLU 分解以后，需要对稀疏三角形矩阵方程求解的通信优化问题。

在 Github 找到 SuperLU_Dist 的源代码库，初步学习 makefile 和 cmake 相关操作。

Paper summary: (文献总结)

Name： Making Sparse Gaussian Elimination Scalable by Static Pivoting

Motivation：Partial pivoting 在共享内存或者单个处理机上可以很好的避免高斯消元过程中元素的增长问题，从而增加在线性方程求解过程中的数值稳定性。但是，当 Partial pivoting 用于分布式处理机时，由于需要动态处理主元的原因，分布式处理机上用于同步和比较的时间将会过于昂贵，因此在分布式内存机器上，希望可以不进行动态的调整，就可以保持求解方程在数值上的稳定性。

Solution：通过各种技术来保持数值稳定性：预先将大元素转移到对角线上、迭代细化以及允许在最后进行迭代的修改等。文章还实现了基于 static pivoting 的 SuperLU_dist 算法以及三角求解器算法。

Related to us：对上篇 3D SuperLU 分解论文中所涉及到的 static pivoting 和 partial pivoting 进行概念理解和掌握。

Name： A communication-avoiding 3D sparse triangular solver

Motivation：稀疏直接求解器的一个常见用例是对一个固定矩阵 A 使用许多 b 向量，也就是说，在稀疏迭代求解时，可能会在前期对系统进行一次 LU 分解，随后在每次迭代时调用 SpTrs 和一个新的 b 向量，因此 SpTrs 的可扩展性也可能成为瓶颈。同时在之前的工作中，团队提出了一种避免通信的 LU 因式分解算法。所以，团队基于上述工作，

利用之前算法中所提到的三维稀疏 LU 数据结构，进行开发，以求获得一种避免通信的 SpTrs，从而实现渐进的减少传统 SpTrs 的延迟和通信量成本。

Solution：团队实现该求解器的思想和实现三维 LU 分解的思想基本一致，都是利用 etree 中所涉及的依赖关系，然后对每个式子中可以独立进行计算的部分进行细分，经过拷贝共同需要的数据后，实现各个部分的并行计算，从而减少通信和计算时间。即，也是通过空间换取时间的措施，来实现直接求解器的加速操作。

Related to us：承接上篇论文中提到的 3D LU 分解操作，继续深入探究 LU 分解之后所涉及到的 SpTrs 求解器的优化工作。

Work summary（工作总结）

1. Making Sparse Gaussian Elimination Scalable by Static Pivoting

（1）**数值稳定性**：在计算机中执行数学运算需要使用有限的比特位来表达实数，这会引入近似误差。近似误差可以在多步数值运算中传递、积累，从而导致理论上成功的算法失败。因此数值算法设计时要考虑将累计误差最小化，也就是说，需要考虑所谓的数值稳定性。

（2）**提高数值稳定性的方法**：尽量减少运算次数、加法运算时，避免大数加小数、避免两个相近数相减、避免小数做除数或大数做乘数。

（3）在做 LU 分解时，如果不做相应的 pivoting，则可能会出现以下情况：在进行 LU 分解之后，在主元上的元素可能会变为 0 或者一个极小值，而在方程求解时涉及到的除法使得主元元素不能是个极小值或者 0，因此需要做相应的 pivoting 来提高数值稳定性，从而避免这种情况发生。

（4）**partial pivoting**：Our shared memory GEPP algorithm relies on the fine-grained memory access and synchronization that shared memory provides to manage the data structures needed as fill-in is created dynamically, to discover which columns depend on which other columns symbolically, and to use a centralized task queue for scheduling and load balancing. 可以得出，partial pivoting 的思想是：随着矩阵方程的计算，不断将位于计算行首部元素对应列下面主元元素大的值转移到当前计算的行进行计算。所以说，partial pivoting 是一个动态更新

主元的算法，这个也就是在 3D LU 分解中提到的 partial pivoting 的缺陷，无法提前确定 etree 的结构。

（5）**static pivoting**：通过各种技术保持数值稳定性：预先将大元素引向对角线、迭代细化、在需要时使用额外精度、以及允许在最后进行修正的低秩修改。

(1) Row/column equilibration and row permutation: $A \leftarrow P_r \cdot D_r \cdot A \cdot D_c$,
where D_r and D_c are diagonal matrices and P_r is a row permutation
chosen to make the diagonal large compared to the off-diagonal
(2) Find a column permutation P_c to preserve sparsity: $A \leftarrow P_c \cdot A \cdot P_c^T$
(3) Factorize $A = L \cdot U$ with control of diagonal magnitude
if ($|a_{ii}| < \sqrt{\varepsilon} \cdot ||A||$) **then**
set a_{ii} to $\sqrt{\varepsilon} \cdot ||A||$
endif
(4) Solve $A \cdot x = b$ using the L and U factors, with the following iterative refinement
iterate:
$r = b - A \cdot x$ ... sparse matrix-vector multiply
Solve $A \cdot dx = r$ ... triangular solve
$berr = \max_i \frac{|r|_i}{(|A| \cdot |x| + |b|)_i}$ ... componentwise backward error
if ($berr > \varepsilon$ and $berr \leq \frac{1}{2} \cdot lastberr$) **then**
$x = x + dx$
$lastberr = berr$
goto iterate
endif

Figure 1: The outline of the new GESP algorithm.

图 1 GESP 算法

选择对角枢轴可以确保此类矩阵的稳定性 。因此，**如果每个对角线条目都能以某种方式相对于其行或列中的对角线条目变大，那么对角枢轴就会更加稳定**。

步骤 (1) 的目的是选择对角矩阵 Dr 和 Dc 以及置换 Pr，使每个 aii 在这个意义上更大。对于步骤(1)的一些算法，这些算法都取决于 n × n 稀疏矩阵 A 的以下图表示：它表示为一个无向加权双向图，每一行一个顶点，每一列一个顶点，每一个非零条目 aij 都有一条具有适当权重的边将行顶点 i 与列顶点 j 连接起来。

・对于寻找负责将较大的元素转移到对角线上的排序矩阵问题，可以转化为一个带权二分图匹配问题,

・对角尺度矩阵 Dr 和 Dc 可以单独选择，以使 DrADc 的每一行和每一列的最大条目在量级上等于 1。

实际应用中最好的算法似乎是（*The design and use of algorithms for permuting large entries to the diagonal of sparse matrices.*）中的算法，即同时选取 Pr、Dr 和 Dc，使 PrDrADc 的每个对角项为 ±1，每个非对角项的大小以 1 为界，并使对角项的乘积最大化。

步骤(2)：列排列矩阵 Pc 可以从任何减少填充的启发式算法中获得。比如，之前提到的嵌套剖分（ND）排序。这种排序启发式地减少了 L 和 U 矩阵中的非零填充并且揭示了稀疏 LU 因式分解和三角求解的并行性。同时还要对 A 的行应用 Pc，确保步骤(1)中得到的大对角线项保持在对角线上。

static pivoting 过程就是步骤(1)和步骤(2)两个步骤。

步骤(3)：将消除过程中遇到的任何微小主元设置为 $\sqrt{\varepsilon} + \|A\|$，其中 ε 是机器精度。这相当于对原始问题进行了一次小的扰动，并以一定的数值稳定性换取了防止主元过小的能力。（主元不能为 0）

步骤(4)：如果求解不够精确，会进行几步迭代改进，这也是对步骤 (3) 中的扰动的修正。

2. A communication-avoiding 3D sparse triangular solver

（1）线性方程稀疏矩阵的直接求解方法：

对于 Ax=b 线性方程的求解来说，除了使用迭代求解的方法来进行求解之外，还存在着直接求解的方式，比如基于高斯消元思想的直接求解器方式。还可以采用 LU 分解的方法来实现分步求解，如下所示。

$$\begin{aligned} Ax &= b \\ LUx &= b \\ \Rightarrow x &= (LU)^{-1}b \\ &= U^{-1}L^{-1}b \end{aligned}$$

转化成两个方程组：

$$\begin{aligned} Ux = L^{-1}b \quad if \quad y = L^{-1}b \quad to \\ Ux = y \\ Ly = b \end{aligned}$$

其中，L 为下三角矩阵，U 为上三角矩阵。同时，在解决相关稀疏迭代求解问题时，我们可能会在前期对系数矩阵 A 进行一次因式分解，然后在每次迭代调用时调用 SpTrs

和一个新的 b 向量，来求出对应新的 b 向量的解向量 x。因此，SpTrs 的可扩展性也可能成为瓶颈。

（2）总结稀疏矩阵线性方程直接求解器的结构

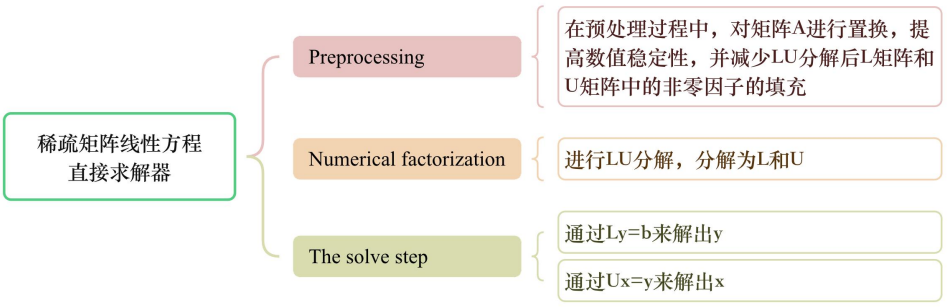


图 2 直接求解器结构以及部分作用

该篇文章主要解决的问题是第三步的三角矩阵求解器的求解步骤。

（3）3D Triangle Slover

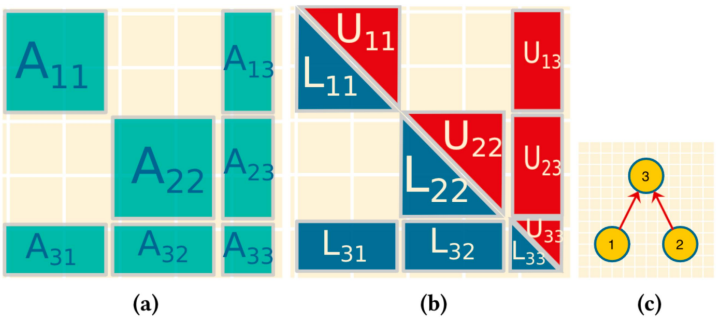


图 3 在 LU 分解中所涉及的三维数据结构

该团队之前改进数值因式分解的工作引入了一种新的三维数据结构，从而自然而然地产生了本文的新算法。

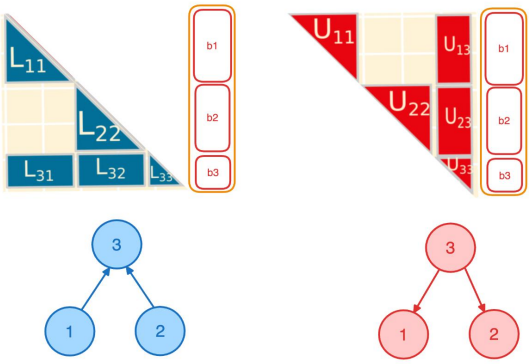


图 4 LU 分解后进行求解的依赖关系

LSlove

在 lSolve 中，网格-0 和网格-1 并行求解 $L_{11}y_1 = b_1$ 和 $L_{22}y_2 = b_2$，并更新相应的 b_3 块，在网格-0 上更新为

$$b_3^0 = b_3^0 - L_{31}y_1$$

，在网格-1 上更新为

$$b_3^1 = -L_{32}y_2$$

。更新后，网格-1 将 b_3^1 发送给网格-0，网格-0 将两个网格对 b_3 的更新累加如下：

$$b_3^0 = b_3^0 + b_3^1 = b_3^0 - L_{31}y_1 - L_{32}y_2$$

。 因此，更新后的 b_3^0 包含了来自两个处理网格的更新，然后网格-0 求解 $L_{33}y_3 = b_3$ 得到最终的 y_3。

USlove

uSolve 可以在 grid-0 计算出 $y3$ 后开始。首先，网格-0 对 $x3$ 求解 $U_{33}x_3 = y_3$，并将 x_3 发送给网格-1。

现在，利用 x_3，网格-0 和网格-1 可以分别更新 $y_1 = y_1 - U_{13}x_3$ 和 $y_2 = y_2 - U_{23}x_3$。

最后，网格-0 和网格-1 对 x_1 和 x_2 分别求解 $U_{11}x_1 = y_1$ 和 $U_{22}x_2 = y_2$。

因此，在 L 求解和 U 求解结束时，最终解 x_1 和 x_2 分别位于网格-0 和网格-1 中，而 x_3 则被复制到两个处理网格中。uSolve 的通信模式与 lSolve 相反。

当 etree 的高度为 2 时，可以得到下图所示的依赖关系以及信息传递方向：

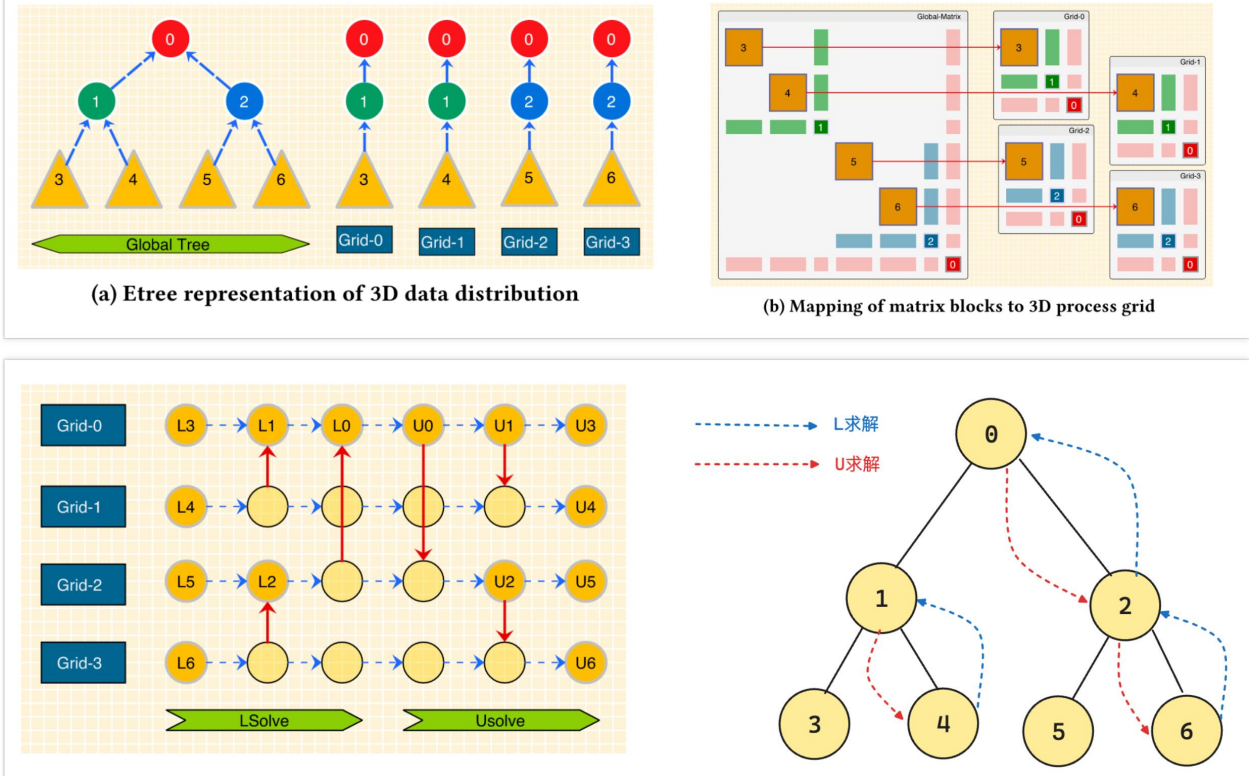


图 5 依赖关系和信息传递方向

Algorithm 2 3D Sparse Lower Triangular Solve Algorithm

Require: Factored L and U matrices, b: right hand side; Process coordinates $\{p_x, p_y, p_z\}$; E_f: grid-local etree; $p_z = 2^l$ for some integer l

LSolve: $y \leftarrow L^{-1}b$

for lvl in $l : 0$ **do**: ▷ Bottom-up traversal of E_f
  if $p_z = k2^{l-\text{lvl}},\ k \in \mathbb{Z}$ **then**:
    $\sigma \leftarrow E_f[\text{lvl}]$ ▷ σ is the index of subtree
    $y_\sigma \leftarrow \text{LSOLVE2D}(L_\sigma, b_\sigma)$
    $b_i \leftarrow b_i - \sum_{i \in Anc(\sigma)} L_i y_\sigma$ ▷ Local-update
    if lvl > 0 **then**:
      if $k \mod 2 \equiv 0$ **then**: ▷ Note $p_z = k2^{l-\text{lvl}}$
        dest $= p_z$
        src $= p_z + 2^{l-\text{lvl}}$
      else:
        src $= p_z$
        dest $= p_z - 2^{l-\text{lvl}}$
      for l_a in lvl $- 1 :\ 0$ **do**:
        for $s \in E_f[l_a]$ **do**:
          if $p_z =$ src **then**:
            Send b_s^{src} to dest
          else:
            Receive b_s^{src} from src
            $b_s^{\text{dest}} = b_s^{\text{dest}} + b_s^{\text{src}}$
  return y

图 6 下三角稀疏求解器算法

该算法总结来说，就是将二叉树的合并转化为通信，即：每个子树的右节点始终是发送方，每次都要将处理后的结果经过通信发送给上级节点，每个子树的左节点始终是接收方，接受来自下一级节点处理后的结果并进行整合。

（3）性能分析

Problem type	Communication Param	SpTrS2D	SpTrS3D
Planar (2D PDE)	Cost (W)	$O\left(\frac{n}{\sqrt{P}}+\sqrt{n}\right)$	$O\left(\frac{n}{\sqrt{P_z P}}+\sqrt{n}\right)$
	Average Volume (V^{avg})	$O\left(\frac{n}{\sqrt{P}}\right)$	$O\left(\frac{n}{\sqrt{P_z P}}\right)$
	Max Volume (V^{max})	$O\left(\frac{n}{\sqrt{P}}\right)$	$O\left(\frac{n}{\sqrt{P_z P}}+\frac{\sqrt{nP_z}}{\sqrt{P}}\right)$
Non-Planar (3D PDE)	Cost (W)	$O\left(\frac{n}{\sqrt{P}}+n^{2/3}\right)$	$O\left(\frac{n}{\sqrt{P_z P}}+n^{2/3}\right)$
	Average Volume (V^{avg})	$O\left(\frac{n}{\sqrt{P}}\right)$	$O\left(\frac{n}{\sqrt{P_z P}}\right)$
	Max Volume (V^{max})	$O\left(\frac{n}{\sqrt{P}}\right)$	$O\left(\frac{n}{\sqrt{P_z P}}+n^{2/3}\frac{\sqrt{P_z}}{\sqrt{P}}\right)$

图 7 理论性能分析

从理论性能分析可以看出，随着 Pz，也就是 etree 的层数的增大，相关的时间花费、平均通信量、最大通信量都会减小。

（4）实验结果

・不同的 etree 高度对通信时间和计算时间的影响

在 Edison 集群的 16 个节点上（每个节点每个节点都包含双插槽 12 核英特尔 Ivy Bridge 处理器），对于平面矩阵和非平面矩阵，三维稀疏三角形求解配置的速度分别比二维配置快 1.3-4.3 倍和 0.9-2.9 倍。

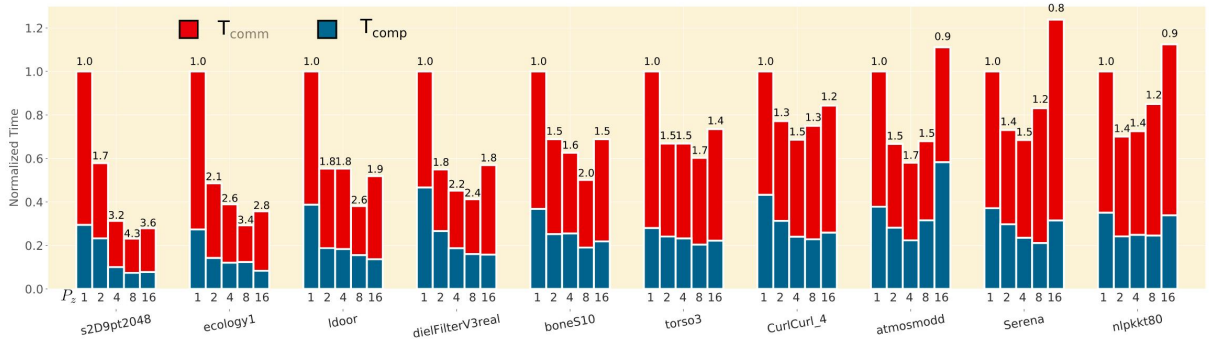


图 8 实际通信时间

Tcomp 是 L 和 U 联合求解关键路径上的局部计算时间，Tcomm 是非重叠的通信和同步时间。

・不同数量的 b 向量情况下的执行速度

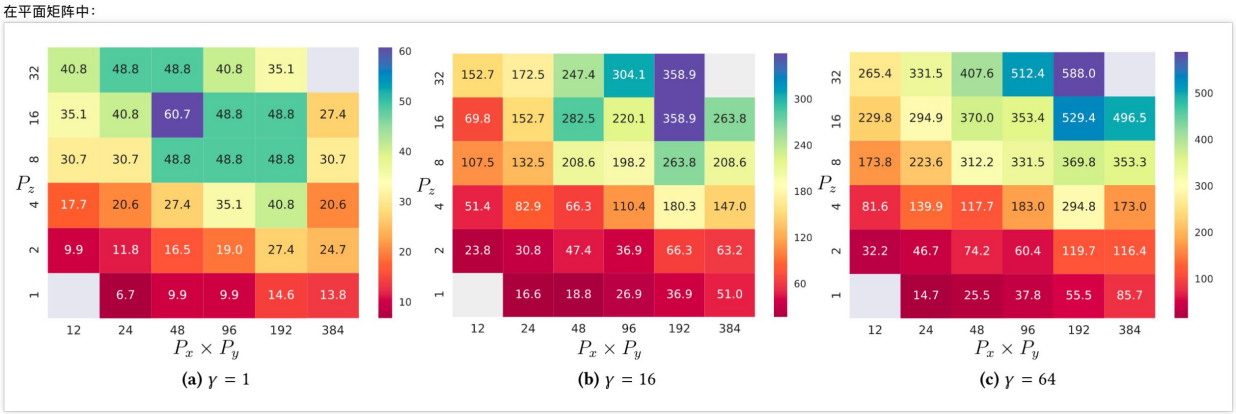
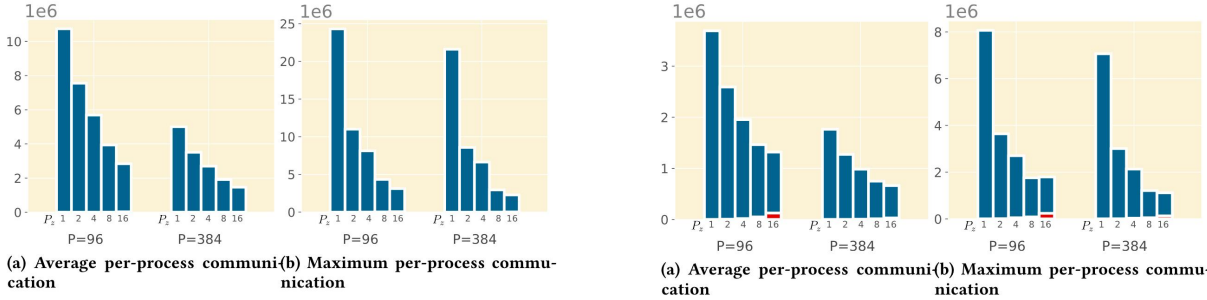


图 9 求解器的可扩展性

・平均通信量和最大通信量



(a) Average per-process communication (b) Maximum per-process communication

(a) Average per-process communication (b) Maximum per-process communication

图 10 通信量的减少

3. 学习 CMake 以及 Makefile 的基本使用

对 CMake 以及 Makefile 的使用进行基本的了解，同时正在阅读 SuperLU_Dist 的安装和运行说明。

Next（下一步）：

学习并行编程的基本知识

运行一个关于 SuperLU_Dist 的基本 Demo