Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

Jack Dongarra 的图灵奖与并行数值线性代数库之路

一点从业人员的吐槽。

1. Dongarra 的图灵奖，和它的反面

前几天，Jack Dongarra 教授荣获 2021 年图灵奖。当天，笔者的微信朋友圈里充满了各种和 Dongarra 教授的合影，笔者身边几乎所有从事（过）与高性能计算相关工作的人都为这个消息感到高兴。当然，也有反面的声音，比如 这个知乎回答。在这个知乎问题下面，这个回答算是比较难得地提到了一些值得思考的问题，其他的回答除了一些同行的肯定，大部分都没什么价值（师老党立也来凑热闹了，他自称是做计算的，但是他的论文在哪里呢，笔者暂且蒙在古里）。这个唱反调的回答主要强调了两点：

> 1. 现在分布式稠密线性代数的并行算法的主要组成部分是 Communication-avoiding 算法，主要贡献人是James Demmel 教授等。Jack Dongarra 现在最新的 SLATE 这种分布式稠密线性代数库的算法就是 CA 那套。
> 2. Top500 的 LINPACK 更加是一个没什么太大用处的排名。实际问题大多都是以稀疏矩阵为基础的。

这两个论点都很有意义。不幸的是，具体的回答内容显示出答主似乎对自己的两个论点没有很好的理解。下面我们深入探讨一下这两个论点。

2. 新一代并行数值线性代数库 SLATE

经过了几十年的发展，稠密数值线性代数库在共享内存机器上有了两套标准接口 BLAS 和 LAPACK，并且有 netlib 的未调优参考实现。而在分布式内存机器上，PBLAS (Parallel BLAS) 和 ScaLAPACK (Scalable LAPACK) 虽然不是名义上的标准接口，但是事实上的标准接口。由于很少有人单独使用 PBLAS, 而 ScaLAPACK 自某个版本以后就自带了 PBLAS，故我们一般不加区分地只称 ScaLAPACK。ScaLAPACK 的参考实现是用 Fortran 写的，已经有超过 20 年的开发历史了。一方面，ScaLAPACK 代码的可读性已经大不如前；另一方面，ScaLAPACK 设计之初仅仅面向 MPI 并行，无法很好驾驭现代的多核和 GPU 计算节点。

Rainmaker's Notebook
『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

自从 Exascale Computing 被摆上桌面，Dongarra 领导的团队就开始为之准备和开发新一代的并行数值线性代数库 SLATE, Software for Linear Algebra Targeting Exascale. SLATE 采用了现代的 C++ 和 OpenMP 来实现各种线性代数算法（因此代码量比起 ScaLAPACK 要小了一个量级），并且广泛吸收了近 20 年来数值线性代数算法的新成果。反面回答认为 SLATE 现在采用的算法都是 Demmel 的那一套 CA 算法，因此 Dongarra 在并行数值代数算法上贡献并没有那么大。我们姑且不讨论前半句这种文无第一的问题，前半句本身也不太对。让我们来看看 SLATE 自己怎么说。

数值线性代数库主要包含几类算法：矩阵乘法、LU 与 Cholesky 分解、QR 分解、特征值与奇异值分解。我们逐个讨论。

首先是矩阵乘法。笔者在 前一篇文章 中分析和介绍了主要的几种矩阵乘法算法，其中包括了 Demmel 的 Communication-Optimal 2.5D 算法以及摸到了通信量理论下界 CARMA 算法。然而 SLATE 采用的是 SUMMA 算法 [1]，和 ScaLAPACK 一样。对此，SLATE 的解释是：

> The current emphasis in SLATE is on simplicity and robustness. More advanced algorithms are up for consideration in the future.

然后是 LU 和 Cholesky 分解。这两个分解的计算模式高度相似，所以我们只讨论 LU 分解。为了偷懒，我从 [2] 中截取了一段内容来说明 LU 分解在做什么：

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

2.3 LU 分解算法

我们基于 HPL 软件包的实现，使用 LU 分解方法求解线性方程组。LU 分解存在三种方法，right-looking，left-looking 和 crout-looking，此处采用的是更容易被并行化的 right-looking 的分解方法[23]。式 2.2 给出 right-looking LU 分解的数学描述。

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} = \begin{bmatrix} L_{1,1} & 0 \\ L_{2,1} & I_{2,2} \end{bmatrix} \begin{bmatrix} U_{1,1} & U_{1,2} \\ 0 & U_{2,2} \end{bmatrix} \tag{2.2}$$

以下是具体分解顺序：

1) $A_{1,1} \leftarrow L_{1,1}U_{1,1}, A_{2,1} \leftarrow L_{2,1}U_{1,1}$
2) $A_{1,2} \leftarrow L_{1,1}U_{1,2}$
3) $A_{2,2} \leftarrow A_{2,2} - L_{2,1}U_{1,2}$
4) $A_{2,2} \leftarrow L_{2,2}U_{2,2}$

图 2.1 给出了对输入矩阵进行分解的示意图。分解过程中，每次循环迭代分解一个 $M \times NB$ 的矩阵，这个被分解的 NB 列矩阵称之为 panel，NB 是算法的输入参数，在输入文件 HPL.dat 中指定。第一步是对 NB 列、M 行的矩阵进行分解，得到 $L_{1,1}$、$U_{1,1}$ 和 $L_{2,1}$，这一步称之为 panel 分解（panel factorization），在分解过程中，同时会执行 panel 内的选主元操作，并将结果进行保存；第二步进行三角矩阵求解（trsm），得到 $U_{1,2}$；第三步是对 $(M-NB)\times(M-NB)$ 的剩余子矩阵（trailing submatrix）进行更新（gemm），得到待分解的 $L_{2,2}U_{2,2}$。

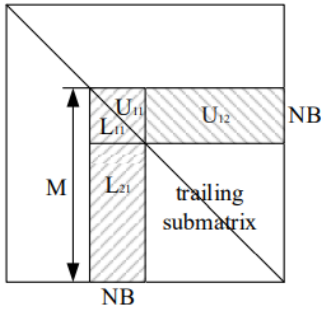

图 2.1 对输入矩阵 A 进行 LU 分解的示意图

SLATE 在做 LU 分解的时候，对 panel factorization 采用了 Communication-Avoding LU 算法 [3]。在 panel factorization 完成以后，SLATE 调用自己的矩阵乘法算法来计算 rank-k update 以更新 trailing matrix。实际上，这个算法并不是通信最小化的算法。通信最小化的算法仍然是 Demmel 提出

Rainmaker's Notebook
『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

的 2.5D LU [4]。与 2D LU 相比，2.5D LU 其实非常好理解：用 2.5D 矩阵乘法来计算 trailing matrix rank-k update。

接着是 QR 分解。关于 QR 分解，Demmel 有两个 CA 算法：TSQR 和 CAQR [5], SLATE 采用了 CAQR [6]。TSQR 针对 tall-and-skinny matrix, 假设这样的矩阵只进行行划分，每个进程持有若干行，通过二叉树归并的方式计算出矩阵的 Householder QR 分解。CAQR 是为通用矩阵设计的，其主体框架是分块化的 Householder QR。在算 panel factorization 的时候，CAQR 采用 TSQR 来算出当前 panel 的 householder vectors，随后对 trailing matrix 的更新和 ScaLAPACK 中的做法完全一致。[5] 中的 Table 16 也指出，Parallel CAQR 和 ScaLAPACK 中的 Householder QR 有着同阶的通信复杂度，前者比后者好的地方是复杂度中的某些部分少了一些常系数。

最后是特征值和奇异值分解。这两种分解的算法比较复杂，当前并没有什么 communication-avoiding 的算法，SLATE 的实现也是使用现有的 1-stage 和 2-stage 方法 [7]。

总结一下，SLATE 采用了很多 CA 算法，但并没有全部采用 CA 算法；同时，其采用的 CA 算法不全部是最小化通信的 CA 算法。在渐进通信复杂度上，SLATE 和 ScaLAPACK 并没有区别。总体而言，SLATE 依靠常系数优化和对 MPI + OpenMP 优化来提升了并行数值线性代数操作的可扩展性。

3. 『历史的终结及最后之人』

上一节对 SLATE 里面矩阵乘法和 LU 分解算法的分析很容易导向一个问题：为什么不能采用达到理论通信复杂度下界的 3D 矩阵乘法算法？这样的话起码矩阵乘法和 LU 分解都可以达到最优的渐进通信复杂度了。

这个问题的答案其实笔者在 前一篇文章 中已经指出来了：CARMA 和 COSMA 这些达到理论通信复杂度下界的矩阵乘法算法，都对矩阵分布有特殊的要求。笔者认为，ScaLAPACK 设计中的一个要点是它采用了 2D block-cyclic (2DBC) 的矩阵分布 [8]，并且所有的算法都是基于这个矩阵分布的。笔者非常讨厌这个反直觉的矩阵分布方式，但是在经历过相关的工作以后，笔者认为这是最不坏的方式。之所以这么说，是因为线性代数的主要算法是各种矩阵分解，随着这些分解算法的进行，trailing matrix 都在变小，大部分是行和列都在变小，有一些是行或者列变小。因此，所有静态的、连续的 1D 或者 2D 矩阵划分，都将随着算法的运行出现有些进程不再持有任何 trailing matrix 的情况。此时如果不重新分布 trailing matrix，则会负载不均浪费计算资源；如果重新分布 trailing matrix，则通信模式变化巨大，且通信量也巨大。2DBC 的使用保证了在矩阵分解的任意阶段，各个进程持有的 trailing matrix 大小几乎相同，差值是一个较小的常数，因此计算和通信的负载大体上是均衡的。如果矩阵乘法使用 COSMA，则必然要在矩阵乘法和其他计算之间来回转换矩阵分布，这个代价同样是不可接受的。

Rainmaker's Notebook
『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

由此看来，并行稠密线性代数算法已经摸到了它的天花板。矩阵乘法、LU 分解、QR 分解都已经有达到理论通信复杂度下界的算法，但这些算法又难以统一在相同的矩阵分布和并行框架下面。SLATE 已经充分利用了多层次的硬件并行和数据访问模式来提高性能，10年以后如果人类的超级计算机还是像现在一样是十年前的放大版，那么接替 SLATE 的新软件还能如何提高性能？

让我们回到反对答案的第二点，即稠密矩阵线性代数已经不实用了，HPL 也只是个用处不大的排名。诚然，现在大规模的科学计算与仿真，其算法主体基本都不再使用稠密线性代数了，唯一的例外是深度神经网络。但是有些基于稀疏矩阵的算法依然需要稠密线性代数运算 [9]。因此，稠密矩阵线性代数并行算法的开发仍然有其必要性。至于 HPL 的价值，笔者非常同意上海交通大学廖秋承研究员的观点，大意为 HPL 仍然是最好的超级计算机稳定性测试工具，其性能波动、能耗波动仍然具有许多值得研究的地方。至于说 Top500 只是给了各个超算中心虚名去做广告要经费，那么哪个 benchmark 不是这样呢？HPL 过度强调浮点性能，HPCG 过度强调内存带宽（甚至可以被模型很好地预测出性能，[10]），Graph500 则毫无反应浮点计算能力。只要怪物亮血条不锁血，总有刮死它的办法。只要有 benchmark 允许改代码，总有针对性优化来刷榜的方法。求全责备，大可不必。

致谢

笔者衷心感谢 SYSU-SCC 水群群友的全天候高强度水群行为，群友的正论激发了本文的写作。

参考文献

1. Parallel BLAS Performance Report, link
2. 面向国产异构处理器的 HPL-AI 基准实现及优化, link
3. Communication Avoiding LU with Tournament Pivoting in SLATE, link
4. Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms, link
5. Communication-Optimal Parallel and Sequential QR and LU Factorizations, link
6. Least Squares Performance Report, link
7. Implementing Singular Value and Symmetric/Hermitian Eigenvalue Solvers, link
8. The Two-dimensional Block-Cyclic Distribution link
9. Numerical Methods for Large Eigenvalue Problems, link
10. Performance Modeling of the HPCG Benchmark, link

发布于 2022-04-03

tags: { LinearAlgebra } { MathLibrary } { MPI }