Weekly Research Progress Report

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

List of accomplishments this week

完成上周剩余部分的 PanguLU 论文的阅读工作，主要涉及负载均衡、加速计算以及无同步通信的实现，还有相关测试结果。

（在并行计算课程中，阅读两篇关于同时多线程的论文）

（在机器学习作业中，学习训练了小型神经网络 AlexNet）

（在数量统计作业中，使用多元线性拟合回归撰写论文一篇）

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，不再使用 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。还提供了一种减少进程间进行消息通信量的新机制，使用在预处理阶段构造的非同步数组进行控制进程间的同步。

Name：Simultaneous Multithreading：Maximizing On-Chip Parallelism

Motivation：在当时，随着技术的进步，单个处理器核心的性能提升逐渐放缓，处

理器设计面临一个核心挑战，即如何充分利用芯片上日益增加的晶体管数量。因此，研究者开始寻找新的方法来提高现代处理器的性能和效率。

Solution：该论文的解决方式是引入**同时多线程技术**。SMT 允许一个处理器核心同时执行多个线程，从而更有效地利用处理器资源。通过这种方式，处理器可以在同一时刻处理更多的任务，提高了处理器的吞吐量和效率。

Related to us：课程结业需要，但也对底层的了解提供了一定的帮助。

Name: Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor

Motivation：上一篇论文的发出，导致 SMT 架构的兴起，这种架构允许多个线程在同一处理器上并行执行。SMT 被认为是提高多任务工作负载性能的潜在方式，因此研究人员致力于寻找一种方法来提高处理器的性能和资源利用率，途径之一就是通过更好地管理指令的获取和发射，从而提高处理器的整体性能。

Solution：论文中创新性的提出了 Choice 概念，它是一种指导指令获取和发射的方法。Choice 允许根据不同线程的情况来选择要获取和发射的指令，而不是简单地按顺序执行。并探讨了如何在实际硬件中实现这些想法，以提高多线程处理器的性能和资源利用率。

Related to us：课程结业需要，但也对底层的了解提供了一定的帮助。

Work summary

PanguLU 剩余部分

（1）论文中提到的四种矩阵乘法

- GETRF：一般三角分解，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：稀疏下三角解，专门用于计算 $U_{ij} = L_{ii}^{-1} A_{ij}$
- TSTRF：稀疏上三角解，专门用于计算 $L_{ji} = A_{ji} U_{ii}^{-1}$
- SSSSM：稀疏矩阵-稀疏矩阵乘法的 Schur 补更新，执行舒尔补码运算，专门用于计算 $A_{jk} = A_{jk} - L_{ji} U_{jk}$。

（2）论文实现负载均衡的方法

基于双层稀疏结构，在数值因式分解前进行预处理时，通过提前计算每个时间片上不同进程的权重（也就是每个时间片中，各个进程上面的任务量）来平衡负载。其中，相应任务的**权重是通过计算内核的 FLOPs 得出的**，也就是文章所提到的 4 种矩阵乘法

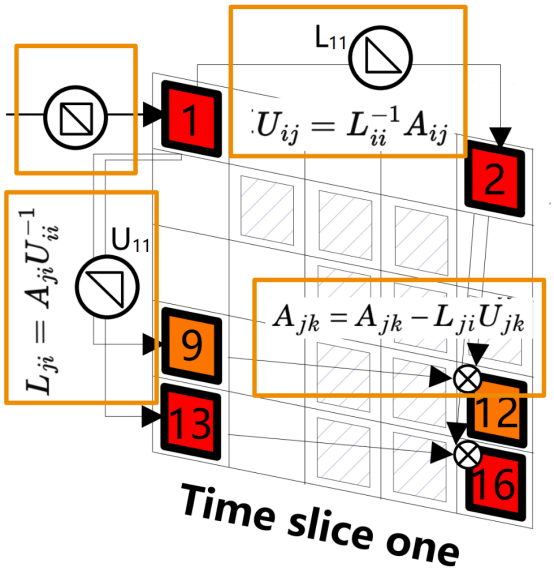

图 1 四种类型的矩阵计算

算法所对应的 17 种计算内核。

大致的负载均衡的思路如下，通过预处理，可以得到矩阵中每一个子块所在的计算阶段以及到后面所需要进行操作。同时，对于各个处理进程的分布，在默认状态时，仍然采用和 SuperLU 进程分布相同的方法，也就是划分二维进程网格，然后循环进行分布。

图 2 Super 中的二维进程网格

这样，就可以计算得到，每个时间片进程执行相应的任务所对应的权重，而且此时得到的权重有两类，一类是累积权重，就是该进程中开始到目前所累积的任务权重，第二类是当前任务权重，也就是在该时间片中，该进程所需要执行任务所对应的任务权重。

文章中所提到的负载均衡策略就是，现将每个进程在各个时间点所对应的权重算出来，然后根据 **每个进程的累积权重**以及 **每个进程在该时间片上的权重**进行任务的二次分配，将累积权重较多进程的当前时间片的任务与累积权重较少进程的当前时间片的任务进行 **交换**，从而达到负载平衡的目的。

比如下面的例子：

在第二个时间片中，若按照原来方式进行分配，总权重最高的进程 0 和 2，权重数

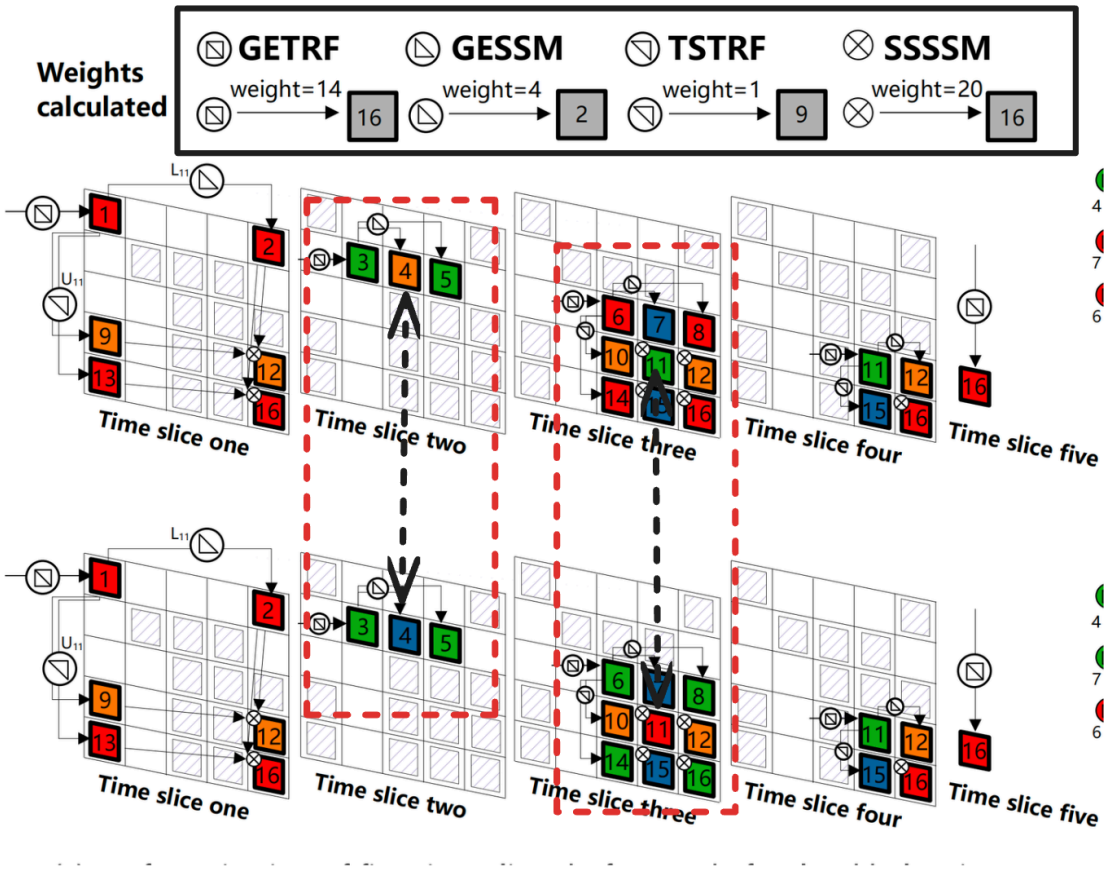

图 3 负载均衡策略示例

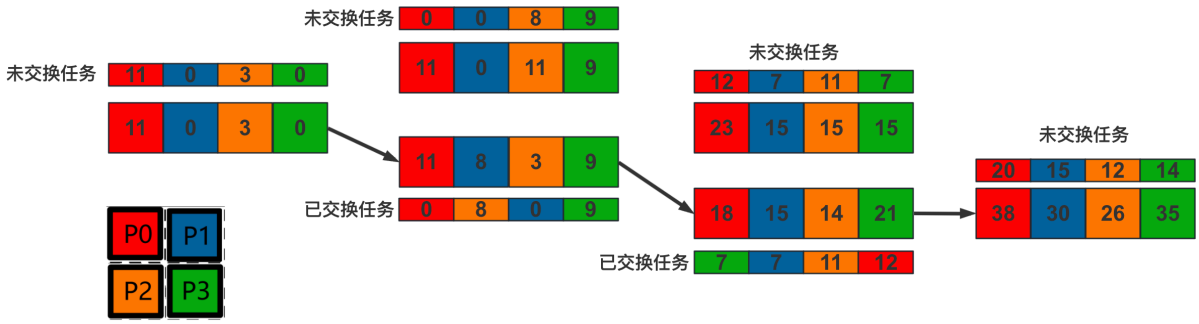

图 4 权重计算交换任务流程

最少的进程为 1，因此，可以选择在 1，2 进程之间进行平衡负载。为此，可以交换任务，从而使子矩阵块 4 的 GESSM 从进程 2 转移到进程 1，由于子矩阵块 4 的 GESSM 权重，这种负载平衡使进程 1 的权重增加了 4，而进程 2 的权重则减少了相同的数量。实现了负载均衡，其他的也是类似的过程。

（3）论文中的 17 种内核以及选取策略

稀疏内核的性能对于数值因式分解至关重要，并受到矩阵密度、结构和大小等多种因素的影响。所以，优化稀疏内核并为每种情况选择更好的算法成为实现整体性能提升的一项重要任务。PanguLU 中实现了 17 个稀疏内核（3 个 GETRF、5 个 GESSM、5 个 TSTRF 和 4 个 SSSSM），然后根据大量性能数据构建了稀疏内核算法选择策略，从而选

择出性能更好的内核。

Kernel	Version	Addressing Method	Parallelising Method	Dense Mapping
GETRF	C_V1	Direct	Row	✓
	G_V1	Bin-search	Un-sync SFLU	-
	G_V2	Direct	Un-sync SFLU	✓
TSTRF/ GESSM	C_V1	Merge	Column	-
	C_V2	Direct	Column	✓
	G_V1	Bin-search	Warp-level column	-
	G_V2	Bin-search	Un-sync warp-level row	-
	G_V3	Direct	Warp-level column	✓
SSSSM	C_V1	Direct	Approximate equal load column block	✓
	C_V2	Bin-search	Adaptive split-bin type	-
	G_V1	Bin-search	Adaptive multi-level	-
	G_V2	Direct	Warp-level column	✓

图 5 论文中的 17 个计算内核

通过利用各个矩阵对每个计算内核的性能测试，可以发现，**这些内核的性能各不相同，没有一个能始终保持最佳性能**，但如果根据矩阵特性以适当的方式组合这些内核，整体性能就会得到极大提高。

所以论文，开发出 **四种决策树来指导我们的算法选择过程**。对于属于面板因式分解的 GETRF、GESSM 和 TSTRF，必须主要根据矩阵中存在的非零（nnz）数量来选择最合适的算法。对于 SSSSM，则主要基于计算中涉及的 FLOPs。

(4) 无同步调度策略

使用稀疏内核作为最小调度单元，通过在进程间传输无同步数组的值来实现细粒度进程调度，从而尽可能使更多的进程处于工作状态。

(a) 无同步数组的构造

一个子矩阵块可以执行多个内核，包括 GETRF、GESSM、TSTRF 和 SSSSM，其中 SSSSM 可以多次执行，而且负责一个子矩阵块的进程的工作状态会受到其他相关子矩阵块状态的影响。因此构建了一个无同步数组来 **记录每个子矩阵块的剩余工作量**，其值等于该子矩阵块仍需执行 GESSM、TSTRF 或 SSSSM 的次数，每次执行时减去 1。该数组中的不同值会触发不同的动作。

对角矩阵需要进行一般的 LU 分解，因此，当它的对角线矩阵块对应的数组数值为 0 时，需要执行 GETRF 操作，在完成后将该值减去 1，当对角线上的值变为-1 时，则对角线所对应的行子矩阵块的 TSTRF 依赖关系和相应列的 GESSM 依赖关系将被打破，

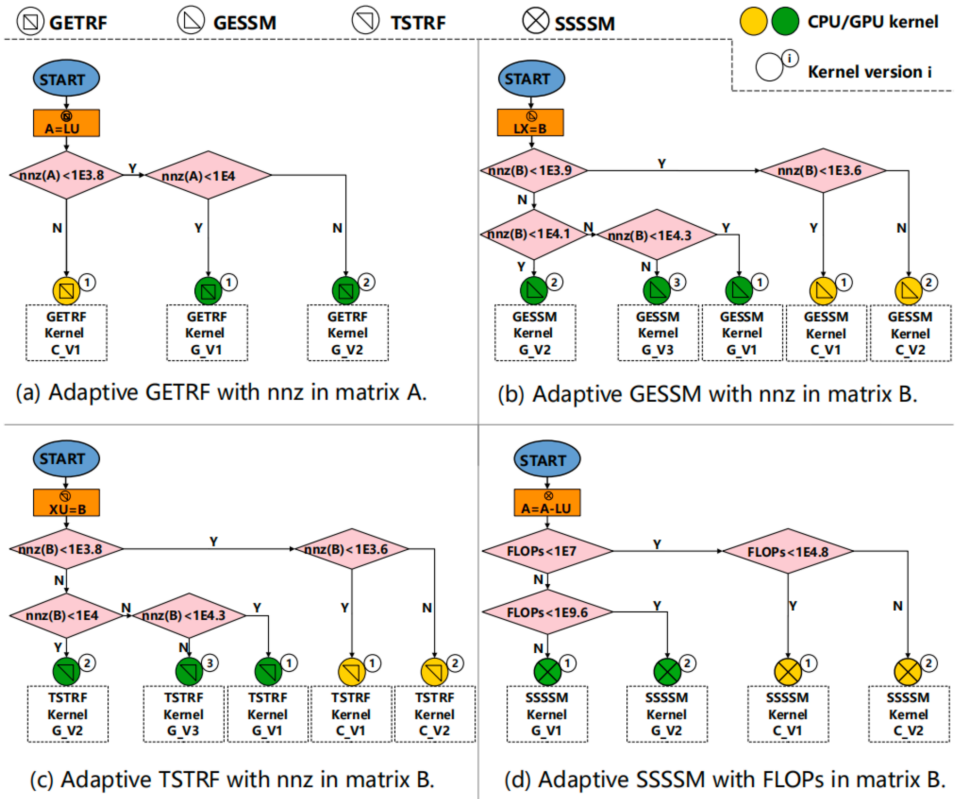

图 6 PanguLU 中的决策树

然后会将这些打破依赖关系的进程会 **将相对应的计算内核**添加到该进程的计算队列中。而当非对角线子矩阵块的数组值为 0，则其对应行或列子矩阵块的 SSSSM 依赖关系将会被打破，并将相应的计算内核添加到进程队列中。

在预处理阶段进行无同步数组的初始化。初始化的基本过程如下图 7 所示：

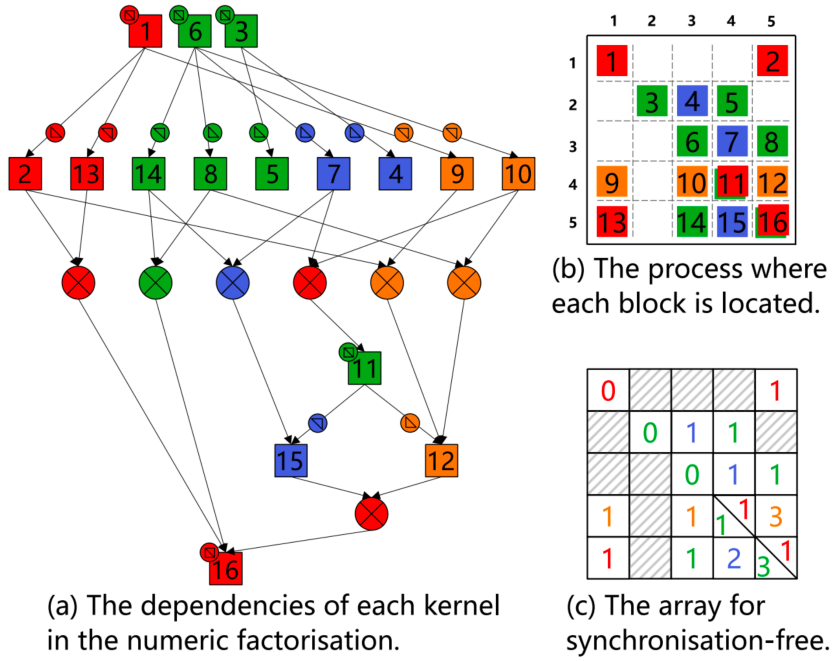

图 7 无同步数组的初始化

箭头方向表示子矩阵块之间的依赖关系，数字表示子矩阵块的剩余工作量，数字的颜色表示其所属进程。

每个块前面的箭头数量就代表的是该块还剩余的工作量。然后处理完之后就会相应的减一，随后，根据不同的状态，执行不同的操作。

(b) 更新无同步数组，管理进程的计算和通信

基本过程如下：

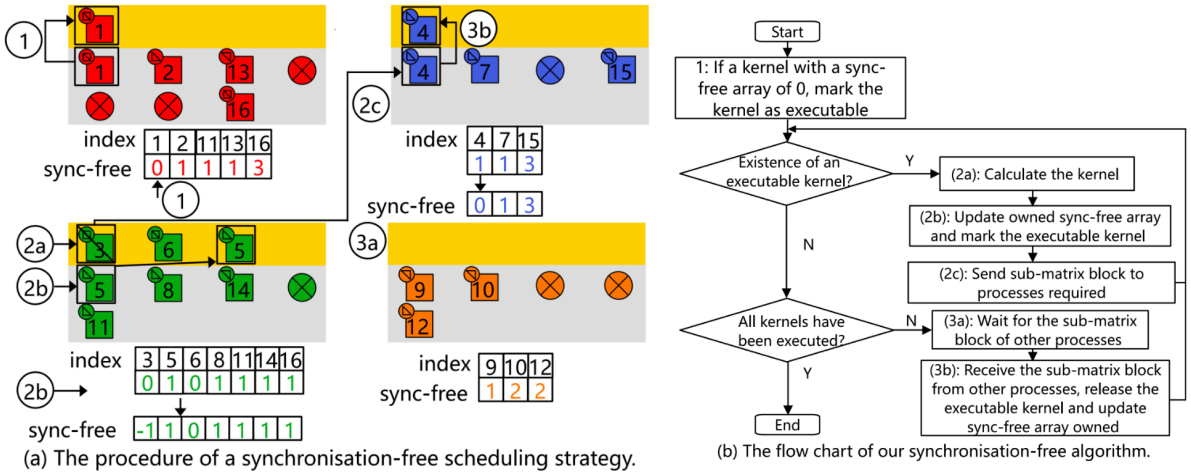

图 8 更新与通信过程

- (1) 每个进程开始时都会将无同步数组中变量为 0 的内核标记为可执行，并将内核添加到任务队列中。然后，进程会检查当前是否有可执行内核。如果是，它将开始计算稀疏内核。否则，它会等待其他进程发送所需的子矩阵块。
- (2) 进程开始计算可执行内核。如果有多个可执行内核，则按优先级顺序选择。计算结束后，进程更新自己的无同步数组，并将子矩阵块发送给其他需要的进程。在图 10 中，(2a) 表示内核的计算，(2b) 表示该进程无同步数组的更新和新内核的添加，(2c) 表示向其他进程发送子矩阵块。
- (3) 进程将等待来自其他进程的子矩阵块。当进程收到来自其他进程的子矩阵块时，就可以添加新的可执行内核，然后更新无同步数组。(3a) 表示进程正在等待来自其他进程的子矩阵块，(3b) 表示进程正在接收来自其他进程的子矩阵块，释放可执行内核并更新无同步数组。

PanguLU 提出的无同步策略可以在一定程度上重新降低同步成本。**在计算过程中，每个进程总是选择要计算的最关键的任务进行计算，使关键路径上的任务的计算速度尽可能快，从而使更多的进程处于计算状态。**

(5) 文章的测试环境和结果

128 个节点，每个节点 4 个 A100 芯片!!

实验结果效果都很不错。

(6) 本篇文章的一点总结

- 使用两层稀疏结构，更加简单的规则二维分块方法来利用矩阵的稀疏性质，通过将计算任务映射（做交换）到不太繁忙的进程中来 **实现负载均衡**。
- 设计了 17 个 4 种稀疏内核的 **计算方式**和 1 个根据矩阵的特点来选择稀疏内核的 **决策树方法**，从而提高异构架构的并行计算效率。
- 关注减少分布式系统中 LU 因式分解的同步开销，使用 **无同步策略**，从而使得每个进程尽可能的计算，减少同步开销。

Next

- 实操曙光跑相关代码
- 看一下即将参加讲座的论文