回顾红蓝取石子游戏:接近最优的并行矩阵乘法

(Red-Blue Pebbling Revisited:
Near Optimal Parallel Matrix-Matrix Multiplication)

作者:Grzegorz Kwasniewski, Marko Kabic, Maciej Besta, Joost VandeVondele, Raffaele Solca, Torsten Hoefler

译者:王煜辉

综述

我们提出了 COSMA，一种并行的矩阵乘法算法。该算法在各种的矩阵维度，处理器数目，内存容量等组合下都达到了接近最优的通信。COSMA 背后的核心思想在于生成一个最优(对于 10MB 的高速缓存，至多达到 0.03%(?))的顺序调度序列然后使它并行，保持输入输出的最优性。

为此，我们使用红蓝取石子游戏来精准地建模矩阵乘法中的依赖关系并且得出了一个建设性的紧密的顺序和并行的输入输出下界证明。想较于预先固定了处理器分解任务然后映射到矩阵维度的二维或三维的算法，我们的算法将通信量降低了至多$\sqrt{3}$倍。COSMA 在所有情况下下都优于现有的 ScaLAPACK, CARMA 和 CTF 算法。将性能提高了至多 12.8 倍(平均 2.2 倍)，最高达到了代恩特峰（译注:代恩特峰是坐落于瑞士的超级计算机，在 2019 年世界超级计算机排名中位列第六名）巅峰性能的 88%。我们的工作不需要任何手动调试，并且以开源的形式维护。

1. 简介

矩阵乘法是科学计算中最根本的基石之一，被应用于线性代数[13，15，42], (Cholesky 和 LU 分解[42]，特征值分解[13]，三角求解器[15])，机器学习[6]，图形处理[4，8，18，36，44，52]，计算化学[21]等。因此，加速矩阵乘法的过程在许多领域都有重大意义。在这份工作中，我们专注于减少矩阵乘法中的数据传输量，包括在多级内存间传输(纵向传输)和多处理器间传输(横向传输，也被广泛地成为“通信”) (注 1：我们只专注于进行了$\mathbf{n}^3$次乘法和加法运算的“经典”矩阵乘法。我们没有分析类似于 Strassen 的方法，因为实际应用中他们通常会更慢一些。)

人们已经在通往矩阵乘法传输最优性的道路行进了至少 50 年。第一个并行的矩阵乘法由 Cannon 提出[10]，它用于方阵矩阵和方形处理器分解。随后的工作[24, 25]将矩阵乘法算法推广到矩形矩阵，不同的处理器分解以及通信模式。PUMMA 包[17]将前面的工作推广到了转置矩阵和不同的数据分布。通过优化通信，引入流水线和通信计算重叠等方式，SUMMA 算法[56]进一步拓展了它。现在它被最先进的所谓二维算法(它在二维网格上分解处理器)使用，例如 ScaLAPACK 库[14]。

Agarwal 等人[1]发现在提供额外内存的情况下，我们可以做得更好，并且引入了三维的处理器分解。基于可用的内存，由 Solomonik 和 Demmel 提出的 2.5 维算法[53]在两个结果中有效地取了插值。但是 Demmel 等人发现虽然该算法在方阵上取得最优，在各维度差异巨大的矩形运行却显得贫弱。这样的矩形在许多相关领域中比较常见，比如在机器学习[60, 61]或计算化学[45, 49]。他们引入了 CARMA[22]，一种在所有维度和内存容量配置下都达到渐近下界的迭代算法。图 2 中象征性地描述了这一步步的演变。

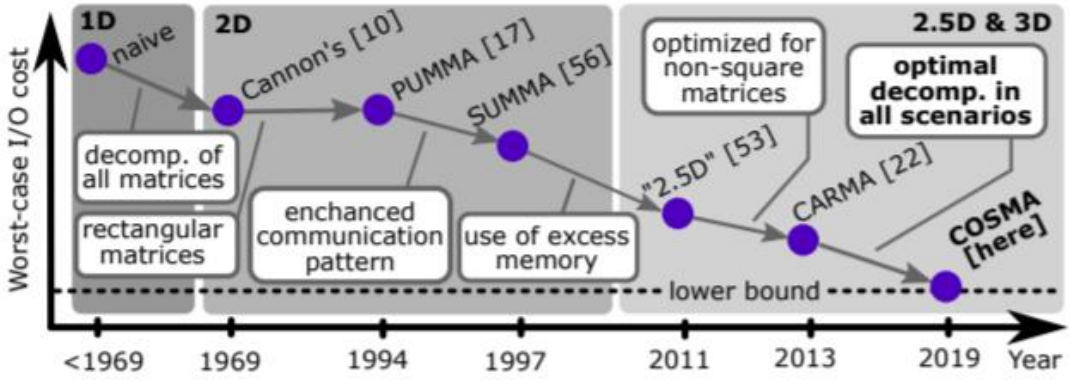

图 2：图解矩阵乘法算法向输入输出下界的演变

……
……
……

2. 背景

我们首先在介绍我们的机器模型(§2.1)和计算模型(§2.2)，然后定义我们的优化目标：输入输出代价(§2.3)

2.1 机器模型

我们的并行机器有 p 个处理器，每个处理器有 S 个字的本地内存。一个处理器每次可以发送或接收至多 S 个字(???)。进行任何运算时，所需操作数必须位于处理器的本地内存。如果存在共享的内存，那么假定它拥有无限的容量。一个字的数据从共享内存输送到本地内存的代价于在两个本地内存间传输的代价相同。

2.2 计算模型

现在我们简单的定义一个通用的计算模型；我们使用这个模型推导对于顺序和并行两种设置，其理论上的输入输出代价。我们用可计算的有向无环图(CDAG) $G=(V, E)$[11, 28, 47]建模算法的执行过程。结点 $v \in V$ 表示给定计算中的一个元操作。边$(u, v) \in E$ 表示操作 v 依赖 u 的结果。结点的直接前驱结点的集合是它的父亲(或孩子)。两个选中的子集 I, O(?)包含于 V 是输入和输出，也就是没有父亲(或对应地来说，没有孩子)的点集。

红蓝取石子游戏 Hong 和 Kung 的红蓝取石子游戏[34]展示了算法在含有一个小而快的内存和一个大而慢的内存的两级内存结构执行的模型。一颗放置在 CDAG 的结点上的红(或蓝)石子表示对应的元操作结果存放于高速(或低速)内存。在初始(或终止)情况下，只有 CDAG 的输入(或输出)结点有蓝色石子。最多只能同时存在 S 颗红色石子。一个完整的 CDAG 的计算是一串从初始态到终结态的操作序列。合法的操作有：在有蓝色石子的任意结点上放置一枚红色石子(读取)，在有红色石子的任意结点上防止一枚蓝色石子(存储)，在父亲结点皆放置有红色石子的结点上放置一枚红色石子(计算)，从任意结点上移除一颗石子，无论红蓝(释放内存)。一个输入输出最优的 CDAG 计算对应了一系列最小化读取和存储的操作(被称为图上的取石子游戏)。在矩阵乘法的语义中，它表示了 n^3 次乘法运算执行的顺序。

2.3 优化目标

在整篇文章中我们关注算法的输入/输出代价。输入输出代价 Q 是算法执行过程中传输的总字数。在一台顺序或共享内存的机器上配备有小而快的内存和大而慢的内存，数据传输

是指在较慢的内存上执行读取或存储运算(也被称为垂直输入输出)。对于一台每个节点有着有限内存的分布式机器，数据传输指的是结点间的通信操作(也被称为水平输入输出)。对于一个给定的 CDAG，输入输出最优的方案应当在所有可行的操作方案中输入输出代价最小。我们也建模分析了延迟代价 *L*，它代表了一个处理器所能发送的最大信息数。

2.4 目前最先进的矩阵乘法算法

在这里我们简要地描述现有的矩阵方法算法的策略。在整篇论文中，我们考虑矩阵乘法 $C = AB$，其中 $A \in R^{m\times k}$， $B \in R^{k\times n}$， $C \in R^{m\times n}$，其中 *m, n* 和 *k* 是矩阵维度。进一步的，我们假设矩阵的每个元素都是一个字，并且 $S < \min\{mn,mk,nk\}$，也就是说，所有矩阵都不能被装入一个处理器的内存里。

我们把自己的算法与二维，二点五维和迭代分解(我们针对二点五维挑选参数使其包含了三维的情况)。我们假定一个方形处理器网格 $[\sqrt{p},\sqrt{p},1]$ 作为二维的变种，近似于 Cannon 的算法[10]，而二点五维的变种是一个正方体网格 $[\sqrt{p/c},\sqrt{p/c},c]$，其中 *c* 是“额外”内存的数目 $c = pS/(mk + nk)$。对于迭代分解，我们假定每一级迭代我们都将 *m*，*n*，*k* 中最大的维度分解为一半。上述分解的详细的复杂度分析可见与表 3。我们注意到 ScaLAPACK 和 CTF 可以处理非方阵分解，但是像 § 1 中所讲的，它们也带来了新的问题。另外，在 § 9 我们将他们的性能表现与 COSMA 并且在所有情境下都测得了巨大的提升。

3. COSMA：高层级的描述

***

***

4. 任意的 CDAG：下界

现在我们将展示推导一般的 CDAG 输入输出下界的数学机制。我们拓展了 Hong 和 Kung[34]的主要引理。对于一个给定的 CDAG，该引理给出了计算输入输出下界的方法。但是这个引理并没有给出一个紧密的边界，因为它高估了重复利用的集合的大小(参见引理 3). 这里我们的关键性结果，引理 4，允许我们对于矩阵乘法的 CDAG 的顺序执行过程推导出一个针对紧的输入输出下界的建设性的证明。

Hong 和 Kung 的推导思路与我们的方法都表明 CDAG 取石子游戏的最优解(该问题是 PSPACE 完全问题[40])的一些性质对应了是 CDAG 图上割的性质(CDAG 子集 V_i 的集合；这些子集形成了子运算，详见 § 2.2)我们可以用割的性质来逼近取石子游戏对应的输入输出操作。Hong 和 Kung 用了该割的一个具体的变种，命名为 S 割[34]。

我们首先介绍我们对 S 割的一般化推广。基于我们的分析，将其成为 X 割。我们在表 2 中介绍我们的分析中所用到的符号。

(表 2)(?)

***

矩阵乘法

m, *n*, *k*: 矩阵维度

A, *B*: 输入矩阵 $A \in R^{m\times k}$ 和 $B \in R^{k\times n}$

$C = AB$: 输出矩阵 $C \in R^{m\times n}$

p: 处理器数目

图

G: 有向无环图 *G*=(*V*, *E*)

Pred(*v*): 结点 *v* 的直接前驱结点，*Pred*(*v*)={*u*: (*u*, *v*)∈*E*}

Succ(v): 结点 *v* 的直接后继结点，*Succ*(*v*)={*u*: (*v*, *u*)∈*E*}

输入输出复杂性

S: 红色石子的数目(高速内存的容量)

V_i: *S* 割的第 *i* 个子计算

Dom(V_i), *Min*(V_i): 子计算 V_i 的支配集和最小集

$V_{R,i}$: 重用集：包含标记有红色石子结点的集合(刚好在 V_i 开始之前)并且被 V_i 使用

H(*S*): 有效的 *S* 割的最小基

R(*S*): 重用集的最大元素数目

Q: 一个方案的输入输出代价(输入输出操作的数目)

p_i: V_i 的计算密集度

p=max$_i${p_i}: 最大计算密集度

方案

S ...

P ...

D ...

a, b ...

***

X 割 在我们定义 X 割之前，我们首先要定义两个集合，支配集和最小集。给定一个子集 $V_i \in V$,定义支配集 *Dom*(V_i)涵盖 *V* 中的部分结点，使得从 CDAG 输入到 V_i 的每条路径都必须经过至少一个 *Dom*(V_i)中的结点。同样定义最小集 *Min*(V_i)涵盖 V_i 中所有子结点不在 $V_{i\ 中}$的结点。

现在，给定一个 CDAG *G*=(*V*, *E*)，定义 $V_1, V_2, \ldots V_h \in V$ 为一系列子运算，且满足(1)两两互斥($\forall_{i,j,i\neq j} V_i \cap V_j = \emptyset$)，(2)覆盖了整个 CDAG，(3)没有循环依赖，(4)它们的支配集和最小集大小不超过 X。这些子运算 V_i 对应了 CDAG 某种执行顺序(方案)，使得在每一步 *i*，只有 V_i 中的结点被操作。这也就是 CDAG 的 X 割或方案，我们将方案标记为 S(X)={$V_1, V_2, \ldots V_h$}。

4.1 现有的通用输入输出下界

在这里我们简单地回顾 Hong 和 Kung 提出的最初的引理，以及其证明的思路。我们将使用类似的方法证明我们的引理 3

思路 现有下界的关键点是对于给定的容量 S 的高速内存使用 X=2S 割。对于子计算 V_i，如果 $|Dom(V_i)| = 2S$，那么在 V_i 计算开始前，其中最多 S 个结点可以拥有红色石子。因此我们需要从内存额外读取至少 S 颗石子。对于 $Min(V_i)$ 的也有类似的讨论。因此，已知 V_i 集

合数目的下界是一个有效的$2S$割，以及由观察得出的每个V_i至少进行S次输入输出操作，我们将 Hong 和 Kung 的引理重新表述如下：

引理 1([34]) 对于任意输入输出计算的 CDAG 的合法执行，输入输出操作的最小数目Q被界定如下

$$Q \geq S \cdot (H(2S)-1) \qquad (1)$$

证明 假定我们已经知道了 CDAG 的最优完全计算，其中计算指的是红蓝取石子游戏[34]中合法的操作序列。将完整的计算分为h组连续的子计算$V_1,V_2,...,V_h$，使得在V_i执行时，$i<h$，有恰好S次输入输出操作，并且V_h中有至多S次操作。现在，对于每个V_i，我们定义V的两个子集：$V_{R,i}$和$V_{B,i}$。$V_{R,i}$是刚好在V_i开始前放有红色石子的结点的集合。$V_{B,i}$是V_i开始前放有蓝色石子，且在V_i执行过程中被放置了红色石子的结点的集合。基于上述定义，我们得出以下的性质：

①$V_{R,i} \cup V_{B,i} = Dom(V_i)$

②$|V_{R,i}| \leq S$

③$|V_{B,i}| \leq S$

④$|V_{R,i} \cup V_{B,i}| \leq |V_{R,i}| + |V_{B,i}| \leq 2S$

类似地，我们定义最小集$Min(V_i)$的两个子集$W_{B,i}$和$W_{R,i}$。$W_{B,i}$是在V_i运算进行过程中属于V_i且标记有蓝色石子的结点的集合。$W_{R,i}$是在V_i运算结束时属于V_i且标记有红色石子的结点的集合。根据对V_i的定义，$|W_{B,i}| \leq S$。由于对红色石子数目的限制，我们可以得出$|W_{R,i}| \leq S$。依据对最小集的定义可以得出$Min(V_i) \subset W_{R,i} \cup W_{R,i}$。最后，根据对$S$割的定义，可以发现$V_1,V_2,...,V_h$形成了对于 CDAG 合法的$2S$割

4.2 通用的输入输出下界

4.2.1 数据重用 对集合$V_{R,i}$，$V_{B,i}$，$W_{R,i}$和$W_{B,i}$进行跟仔细的分析可以让我们改进 CDAG 的输入输出次数的边界。根据定义，$V_{B,i}$是我们使用读取规则放置红色石子的点的集合；我们称$V_{B,i}$为V_i的读取集。然后，我们定义了$W_{B,i}$为在V_i运算进行过程中属于V_i且标记有蓝色石子的结点的集合；我们称$W_{B,i}$为V_i的存储集。可是，这里我们要对$V_{R,i}$和$W_{R,i}$的定义作

出更多限制：$V_{R,i}$是刚好在V_i开始前放有红色石子的结点的集合，并且——对于每一个结点$v \in V_{R,i}$——v的至少一个子结点在V_i操作过程中发生依据红蓝取石子游戏的计算规则而发生改变(?)。我们称$V_{R,i}$为V_i的重用集。类似地，$W_{R,i}$是在V_i运算结束时属于V_i且标记有红色石子的结点的集合，并且——对于每一个结点$v \in W_{R,i}$——v的至少一个子结点在V_{i+1}操作过程中发生依据红蓝取石子游戏的计算规则而发生改变(?)。我们称$W_{R,i}$为V_i的缓存集。因此，如果在子运算V_i的过程中发生了Q_i次输入输出运算，则有$Q_i \geq |V_{B,i}| + |W_{B,i}|$。

首先我们观察到，给定一个最优的完全计算过程，该计算过程可以被分为多个子计算V_i使得每个子计算进行任意Y次输入输出操作。我们也知道$|V_{R,i}| \leq S$，$|V_{B,i}| \leq S$，$0 \leq |W_{B,i}|$(依据红蓝取石子游戏规则的定义)，另外，可以发现，因为在每次子计算中我们进行恰好Y次输入输出操作，并且根据定义，$V_{B,i}$中的所有结点都必须被读取。所以$|V_{B,i}| \leq Y$。同样地，$0 \leq |W_{B,i}| \leq Y$。

将$|V_{R,i}|$和$|W_{R,i}|$的上界记为$R(S)$($\forall_i \max\{|V_{R,i}|,|W_{R,i}|\} \leq R(S) \leq S$)。进一步将$|V_{B,i}|$和$|W_{B,i}|$的上界记为$T(S)$($\forall_i 0 \leq T(S) \leq \min\{|V_{B,i}|,|W_{B,i}|\}$)。我们可以用$R(S)$和$T(S)$来缩紧$Q$的边界。我们将$R(S)$称为 CDAG 的最大重用，$T(S)$称为 CDAG 的最小输入输出。

4.2.2 基于重用的引理 现在我们使用上面的定义和性质**对 Hong 和 Kung[34]的结果进行拓展**。

引理 2 进行了q次输入输出操作的 CDAG $G=(V,E)$的最优完全计算过程与G的 X 割存在如下关系

$$q \geq (X - R(S) + T(S)) \cdot (h-1)$$

对于任意$X \geq S$成立。其中h是 X 割中子计算的数目，$R(S)$是最大重用集的大小，$T(S)$是给定 X 割的最小输入输出。

证明 我们采用与原引理相近的推导方法。最优石子移动中h组连续的子计算$V_1, V_2, ..., V_h$与每组子计算V_i进行的$Y = X - R(S) + T(S)$次输入输出操作有存在联系。在Y次运算内，我们分别考虑$q_{i,s}$次存储操作和$q_{i,l}$次读取操作。对于每个V_i我们有

$q_{i,s}+q_{i,l}=Y$， $q_{i,s}\geq T(S)$ 以及 $q_{i,l}\leq Y-T(S)=X-R(S)$。

$$\forall_i :|V_{B,i}|\leq q_{i,l}\leq Y-T(S)$$

$$\forall_i :|V_{R,i}|\leq q_{s,l}\leq R(S)\leq S$$

因为 $V_{R,i}\cup V_{B,i}=Dom(V_i)$，所以：

$$Dom(V_i)\leq|V_{R,i}|+|V_{B,i}|$$

$$Dom(V_i)\leq R(S)+Y-T(R)=X$$

对于存储运算，通过相近的构造方法可以得出 $|Min(V_i)|\leq X$ 。为了证明 $S(X)=\{V_1...V_h\}$ 满足 X 割，我们的推导与最初的方法相同[34]。

因此，一个进行了 $q\geq(X-R(S)+T(S))\cdot(h-1)$ 次输入输出运算的完全计算过程有一个对应的 $S(X)$，且满足 $|S(X)|=h$ (如果 $q=(X-R(S)+T(S))\cdot(h-1)$，则 $|S(X)|=h-1$)。从上面的引理中，我们得到了一个更紧地输入输出下界

引理 3 将一个 CDAG $G=(V,E)$ 中任意合法的 X 割的最小子计算数目标记为 $H(X)$，且 $X\geq S$。该 CDAG $G=(V,E)$ 任意合法执行过程所对应的最小输入输出操作数 Q 满足如下边界

$$Q\geq(X-R(S)+T(S))\cdot(H(X)-1) \qquad (2)$$

其中 $R(S)$ 是最大重用集的基， $T(S)$ 是最小输入输出集的基。另外，我们还得出

$$H(X)\geq\frac{|V|}{|V_{\max}|} \quad (3)$$

其中 $V_{\max}=\arg\max_{V_i\in S(X)}|V_i|$ 是 CDAG 方案 $S(X)=\{V_1...V_h\}$ 中点的最大子集。

证明 根据定义， $H(X)=\min_{S(X)}|S(X)|\leq h$ ，所以我们可以通过引理 2 立刻得出 $Q\geq(X-R(S)+T(S))\cdot(H(X)-1)$ 。为了证明公式(3)，我们根据定义首先得出 $V_{\max}$ 是最优 X-割的最大子集。因为这些子集是互斥的，所以任一其他子集包含的需要被操作的结点数都要少于 $V_{\max}$。又因为子集之间没有循环依赖，我们可以给出它们的拓扑序 $V_1,V_2,...,V_{H(X)}$。为了确保下表正确，我们定义 $V_0\equiv\varnothing$。现在，定义 W_i 为没有包含于编号

从 1 到 i 所有子集中的点的集合。然后，我们得出

$$\forall_i |V_i| \le |V_{\max}|$$

$$|W_i| = |W_{i-1}| - |V_i| \ge |W_{i-1}| - |V_{\max}| \ge |V| - i|V_{\max}|$$

$$|W_{H(X)}| = 0 \ge |V| - H(X)\cdot|V_{\max}|$$

也就是说，在 $H(X)$ 步之后，我们得到 $H(X)\cdot|V_{\max}| \ge |V|$

从该引理中，我们得到了下面这个引理。我们可以用它来证明矩阵乘法的一个紧的输入输出下界(定理 1)：

引理 4 对于子计算 V_i，定义 V_i 对于单个读取数据所进行的运算数目为计算强度 $\rho_i = \frac{|V_i|}{X - |V_{R,i}| + |W_{B,i}|}$。将最大计算强度记为 $\rho = \max_i(\rho_i) \le \frac{|V_{\max}|}{X - |R(S)| + |T(S)|}$。那么，输入输出操作数 Q 被界定为 $Q \ge |V|/\rho$。

证明 注意到公式 2 中出现 $H(X)-1$ 这一项是因为最后的子计算进行的输入输出操作可能少于 $Y - R(S) + T(S)$，这是因为 $|V_{H(X)}| \le |V_{\max}|$。但是，由于 ρ 被定义为最大计算强度，所以进行 $|V_{H(S)}|$ 次运算需要至少 $Q_{H(S)} \ge |V_{H(S)}|/\rho$。因此总共的输入输出操作数是：

$$Q = \sum_{i=1}^{H(X)} Q_i \ge \sum_{i=1}^{H(X)} \frac{|V_i|}{\rho} = \frac{|V|}{\rho}$$

5. 矩阵乘法的紧的输入输出下界

在这一节会展示我们主要的理论贡献：对于传统矩阵乘法的一个紧的输入输出下界的建设性的证明。在 § 6，我们将他拓展到并行的设定下(理论 2)。我们的结果是紧的(基于递减因子 $\sqrt{S}/\sqrt{S+1}-1$)，因此可能是长久以来不断提升边界的过程的最后一步。Hong 和 Kung[34]针对顺序情况得到渐近了边界 $\Omega(n^3/\sqrt{S})$。Irony 等人[33]将这个边界拓展到了有 p 个处理器的并行机器上，每个处理器有容量为 S 的快速私有内存，证明了每个过程中通信量的下界为 $\frac{n^3}{4\sqrt{2}p\sqrt{S}} - S$。最近，Smith 和 van de Gein[48]针对顺序情况证明了一个紧的下界(基于一个可加项) $2mnk/\sqrt{S} - 2S$。我们的证明改进了这个可加项并将其拓展到了并行情景下。

定理 1 (顺序矩阵乘法输入输出下界)一个通过 mnk 次乘法使尺寸为 $m\times k$ 和 $k\times n$ 大小

的矩阵相乘的矩阵乘法的的 CDAG 的石子移动过程需要至少 $\frac{2mnk}{\sqrt{S}}+mn$ 次输入输出操作。

对定理 1 的证明需要使用引理 5 和引理 6，继而需要一些定义。

5.1 定义

(译注:对本文最重要的理论贡献定理 1 和定理 2 的阐述和完整证明还需要 12 页左右的篇幅。精力所限，后文将不再翻译。感兴趣的读者可查阅原文)

***

12 部分引用文献

[1] R. C. Agarwal et al. 1995. A three-dimensional approach to parallel matrix multiplication. IBM J. Res. Dev. (1995).

[4] Ariful Azad et al. 2015. Parallel triangle counting and enumeration using matrix algebra. In IPDPSW.

[6] Tal Ben-Nun and Torsten Hoe er. 2018. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. CoRR abs/1802.09941 (2018).

[8] Maciej Besta et al. 2017. SlimSell: A Vectorizable Graph Representation for Breadth-First Search. In IPDPS.

[10] Lynn Elliot Cannon. 1969. A Cellular Computer to Implement the Kalman Filter Algorithm. Ph.D. Dissertation

[13] Franàoise Chatelin. 2012. Eigenvalues of Matrices: Revised Edition. Siam.

[14] Jaeyoung Choi et al. 1992. ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In FRONTIERS.

[15] Jaeyoung Choi et al. 1996. Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines. Sci. Program. (1996).

[17] Jaeyoung Choi, David W Walker, and Jack J Dongarra. 1994. PUMMA: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers. Concurrency: Practice and Experience 6, 7 (1994), 543 – 570.

[18] omas H Cormen, Charles E Leiserson, Ronald L Rivest, and Cli ord Stein. 2009. Introduction to algorithms. MIT press.

[21] Mauro Del Ben et al. 2015. Enabling simulation at the h rung of DFT: Large scale RPA calculations with excellent time to solution. Comp. Phys. Comm.(2015).

[22] J. Demmel et al. 2013. Communication-Optimal Parallel Recursive Rectangular Matrix Multiplication. In IPDPS.

[24] Geo rey C Fox. 1988. Solving problems on concurrent processors. (1988).

[25] Geo rey C Fox, Steve W O o, and Anthony JG Hey. 1987. Matrix algorithms on a hypercube I: Matrix multiplication. Parallel computing 4, 1 (1987), 17 – 31.

[34] Hong Jia-Wei and Hsiang-Tsung Kung. 1981. I/O complexity: The red-blue pebble game. In STOC.

[36] Jeremy Kepner et al. 2016. Mathematical foundations of the GraphBLAS. arXiv:1606.05790 (2016).

[42] Carl D Meyer. 2000. Matrix analysis and applied linear algebra. SIAM.

[44] Andrew Y Ng et al. 2002. On spectral clustering: Analysis and an algorithm. In

NIPS.
[45] Donald W. Rogers. 2003. Computational Chemistry Using the PC. John Wiley & Sons, Inc.
[49] Ra aele Solca, Anton Kozhevnikov, Azzam Haidar, Stanimire Tomov, Jack Don-` garra, and omas C. Schulthess. 2015. E cient Implementation of antum Materials Simulations on Distributed CPU-GPU Systems. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC' 15). ACM, New York, NY, USA, Article 10, 12 pages. h ps://doi.org/10.1145/2807591.2807654
[53] Edgar Solomonik and James Demmel. 2011. Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms. In EuroPar
[52] E. Solomonik et al. 2017. Scaling Betweenness Centrality using CommunicationE cient Sparse Matrix Multiplication. In SC.
[56] Robert A Van De Geijn and Jerrell Wa s. 1997. SUMMA: Scalable universal matrix multiplication algorithm. Concurrency: Practice and Experience 9, 4 (1997), 255 – 274.
[60] Nan Xiong. 2018. Optimizing Tall-and-Skinny Matrix-Matrix Multiplication on GPUs. Ph.D. Dissertation. UC Riverside.
[61] Qinqing Zheng and John D. La erty. 2016. Convergence Analysis for Rectangular Matrix Comple