基于 CPU-GPU 混合架构的大规模并行张量网络态算法

Andor Menczer[1,2] 和 ¨Ors Legeza[1,3]
[1]*Strongly Correlated Systems "Lendület" Research Group, Wigner Research Centre for Physics, H-1525, Budapest, Hungary*
[2]*Eötvös Loránd University, Budapest, Hungary*
[3]*Institute for Advanced Study, Technical University of Munich, Lichtenbergstrasse 2a, 85748 Garching, Germany*(日期：2023年5月10日)

量子模拟与经典模拟之间的相互作用及其微妙界限，是面向高性能计算（HPC）的海量并行张量网络态（TNS）算法关注的重点。本文介绍了全新的算法解决方案及其实现细节，旨在基于最先进的硬件和软件技术，拓展高性能计算基础设施上的张量网络态算法现有能力。本文还展示了针对选定强关联分子体系开展的大规模密度矩阵重整化群（DMRG）模拟所获得的基准测试结果，涉及希尔伯特空间维度高达 2.88×10^{36} 的问题。

arXiv:2305.05581v1 [quant-ph] 9 May 2023

I. 引言

过去几十年来，数值（经典）模拟已成为基础研究和应用研究的重要组成部分。这一发展得益于高性能计算（HPC） [1], 的巨大进步，以及数值算法的发展；这些算法可用于模拟物理、化学、生物、经济和生态等众多系统。然而，将经典数值方法应用于相互作用的量子系统时，会遇到一个根本限制：所谓的维数灾难，即对于由多粒子薛定谔方程描述的系统，计算开销会随着希尔伯特空间的维度呈指数级增长。遗憾的是，目前还没有已知的通用"解决方案"可以应对这一问题。 [2–11]。

由于高效量子计算机的设计和大规模制造仍是研究热点 [12–14], ，量子多体问题的数值模拟仍依赖于经典计算。量子模拟与经典模拟之间的相互作用，以及二者之间微妙的分界， [15]正是面向高性能计算基础设施设计的大规模并行张量网络态（TNS）算法 [16–23]所关注的重点。

我们基于张量网络态（TNS）的方法着重开发大规模并行算法，这些算法不仅具有高度可扩展性，非常适合在高性能计算环境中使用，而且依托量子多体物理和应用数学的基础，将所需的算术计算量减少了多个数量级。因此，模拟的指数级时间成本已降至多项式复杂度。

在这项工作中，我们重点关注张量网络态算法的一个子类，即密度矩阵重整化群（DMRG）方法 [24], ，但所提出的算法解决方案同样适用于一般的张量网络态拓扑结构。在此类情况下，大规模张量运算可以替换为数百万次向量和矩阵运算，其中许多运算可以独立执行。通过利用这些依赖关系与非依赖关系，可以重新排列算术运算，并将其划分为多个层级的组，分别对应不同的软件和硬件层，从底层基于中央处理器和图形处理器的单指令多数据执行，到高层的高性能计算调度。对于每个层级，我们都可以独立执行同一组中的所有运算，而不受组外其他算术运算的影响，因此可以在每个组层级分别实现大规模并行性。由此形成的并行化结合了各层级自身的大规模并行能力；因此，借助合适的硬件基础设施，百亿亿次级计算 [22]正成为基于密度矩阵重整化群的量子模拟的现实。

本文结构安排如下。第二节介绍根据各种并行化策略开发的方法，以及用于实现高效混合 CPU-多 GPU 内核的创新算法解决方案，以便在高性能计算基础设施上进行模拟。第三节通过密度矩阵重整化群方法的框架，阐述实现细节；该方法适用于包括长程相互作用在内的一般模型哈密顿量。第四节给出针对所选化学体系的数值基准测试和扩展性分析，并探讨未来的可能性和能耗。第五节总结全文，并逐点给出结论。

II. 方法

为了追求极致性能，我们对更轻量、更快速、更灵活且更易于部署的并行化构造的需求日益增长，且从未止息。本节将介绍我们的一些发现，以及助力我们不断改进软件的新方法和工具集。

A. Maze-Runner

在传统的生产者-消费者模型中，线程被划分为互不相交的集合，分别标记为 生产者 和消费者。前者负责将当前正在执行的程序的主要任务拆分成更小、可独立解决的子任务。随后，这些任务片段会以某种形式存入缓冲区，并持续传递给消费者，由消费者负责解决这些子任务。

理想情况下，生产者线程和消费者线程可以并行运行，使这一模型尽管简单，却成为构建多线程工作负载的高效途径。

1. *Batched Type Task Processing*

理论上，在理想的生产者-消费者模型中，缓冲区在使用期间应始终既非空也未满。若出现这两种情况中的任何一种，生产者或消费者就会停滞。因此，必须在生产者和消费者之间近乎完美地分配线程。

遗憾的是，线程的最优分配往往难以实现，尤其是在创建任务与解决任务的难度关系变化不定时。在这种情况下，任务流并不均匀，这意味着任务生成时间可能迅速波动，且不会随任务规模线性变化。除非在生产者和消费者之间持续重新分配线程，否则性能就会受到影响。

我们没有采用依赖任务特定优化的高复杂度动态调度系统，而是提出了一种更通用的方法：不去解决问题，而是直接避开整个问题。

通过创建一个通用线程池，并让线程池先收集所有当前可用的任务，再允许各个线程领取任务，我们可以用另一组问题来替代调度问题。主要的问题是，连续的数据流被打断了，而且我们预计会失去基于流水线的并行化所带来的好处。

让我们通过在模型中加入迭代来改进这种方法。以小批次的方式生产和消费任务，可以重新引入数据流，尽管流水线元素之间传递的数据包会更大、频率也更低。这还会因可用任务池的碎片化而引发性能和利用率问题。

然而，如果由于与并行化主题相关的算法原因，已经存在隐式屏障[1] ，那么这些问题都不会构成问题，因为

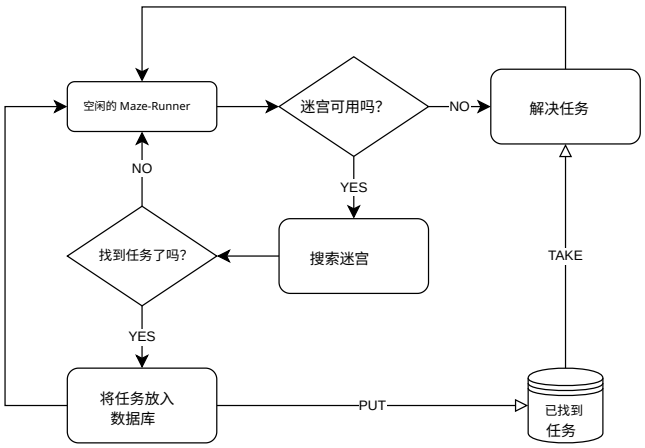

图 1. Maze-Runner 线程的生命周期

并行化模型的迭代可以与执行算法的逻辑同步。简单来说，对于迭代算法，这种对生产者-消费者模型的修改几乎不会带来任何开销，因为用于实现算法自身迭代的框架可以重复用作并行化模型的主循环。从某种意义上说，我们利用基础算法本身无法并行执行连续迭代这一特性，来掩盖系统因分多个批次处理任务而引入的同一瓶颈。

2. *Replacing Producers and Consumers with Maze-Runners*

在生成的任务是复杂系统之间交互的产物时，为消费者生成任务的过程开始呈现出一种迷宫般的结构，其中隐藏着各种各样的项目。当一个线程进入其中时，结果具有不确定性，因为很难预测该线程会找到什么类型的任务，以及搜索需要多长时间。如果没有高级调度和预处理，这些线程就会深入未知领域。引入新的术语后，我们不妨将这些未探索的区域简单称为迷宫。

线程将不断重新进入迷宫，直到其中的所有奥秘都揭晓为止。当一些线程仍在迷宫内，试图找到最后几个任务时，外部线程将不再重新进入迷宫，而是开始处理已收集的任务。由此，任务生产向任务消费的转换开始发生。由于生产和消费任务使用的是完全相同的线程，直观地说，我们可以将这些线程称为生产者-消费者线程 *Maze-Runners* 。

作为一种推广，迷宫运行器都是基础算法本身生成的高阶任务的消费者。消费者通过生成另一个任务来解决这类高阶任务——

[1] 在并行编程中，*barrier* 是一种结构，会阻塞某些线程，使其无法越过执行过程中的某个点——也就是说，直到所有相关线程都调用屏障为止。[2] 我们要并行化的基本算法。

然后，该任务会作为迷宫中的一个发现重新引入线程池。最终得到的是一个递归、自我供给的执行器[3]。

我们的模型之所以能够提升性能，是因为线程并不与当前递归层级绑定，任务才是。相反，线程可以接收来自任意递归层级的任务。这使线程利用率得以大幅提升，而经典的生产者-消费者流水线难以做到这一点，因为这类方法会为线程分配特定角色，从而限制可处理任务的范围。尽管这些经典模型的扩展版本可以在必要时改变线程角色，但引入复杂的调度系统会增加开销和代码复杂度。我们的方法无需复杂的调度，同时允许所有线程不受限制地访问线程池中的所有可用任务。

3. Notes on Implementation and Code Complexity

一般而言，内层循环包含难以直接收集的子任务的顺序循环，是经典生产者-消费者模型中 Maze-Runner 变体的良好应用场景。

在大多数情况下，实现起来十分简单：只需将并行化模型封装在算法自身迭代的主体外围，并说明哪些内容应视为任务，从而在内层循环的内部与外部之间创建一个中间层。这样一来，并行化模型与算法本身仅松散耦合，因此这类架构相对容易实现和开发。

4. Notes on non-CPU based Task Consumption

所提出的模型将任务生成速度置于首位，因为所有线程最初都分配用于创建任务。使用迷宫运行器为图形处理器或 FPGA 等非中央处理器设备提供任务时，设备因任务不足而闲置的风险会降至最低。

B. 树遍历优化的虚拟内存寻址

随着硬件计算能力不断提升，软件也日益增强可扩展性，分配、释放和复制等输入输出操作越来越难以实现，[25]而又不使算法的计算部分成为瓶颈。本节将提出

[3] *Executors* 是扩展线程池，具备调度功能和用于接收任务的接口。

算法 1 TTcache - 外部数据库上的深度优先搜索

过程 LOAD($data, cache, offset$)
  $memcpy(cache + offset,\ data,\ size(data))$
  $offset \leftarrow offset + size(data)$ ▷ 可从外部访问
结束过程
过程 UNLOAD($data, cache, offset$)
  $offset \leftarrow offset - size(data)$ ▷ 可从外部访问
结束过程
过程 VISIT($v, database, cache, offset$)
  $data \leftarrow database.get(v)$
  $load(data,\ cache,\ offset)$
  $execute(v,\ cache)$ ▷ 在 v 本地解决任务
  对于 $child:\ v.children$ **执行**
    $visit(child, database, cache, offset)$
  结束循环
  $unload(data,\ cache,\ offset)$
结束过程
过程 TTCACHE($database, root$)
  $cache \leftarrow malloc()$ ▷ 分配所有可用内存
  $offset \leftarrow 0$
  $visit(root, database, cache, offset)$
结束过程

一种能够创建虚拟内存寻址系统的方法。该系统不仅能让我们在 $\mathcal{O}$O(1) 时间内分配和释放内存，还能凭借其缓存并重用连续使用的数据组之间交集的天然能力，大幅减少所需的复制操作数量。

1. Data Dependency Trees

内存管理的一种朴素解决方案，是始终将所有所需数据存储在内存中。当内存充足时，没有必要改进这一模型，因为我们已经可以保证每个数据集只需一份副本；从性能角度来看，这是理想的做法。

然而，当数据集的总大小超过可分配内存的大小时，尽量减少每个数据集的复制调用次数可能会变得极其困难。在简单情况下，使用缓冲区并以数据块的形式加载数据即可解决这一问题。不幸的是，在复杂情况下——即不存在算法在语义上等价的重构方式，且连续批次的数据之间的重叠程度足够低，无法将 I/O 隐藏在并行运行的计算之后——则需要更复杂的方法。

通过重新排列与特定数据集相关的计算，执行期间通常可以将相互重叠的数据集放在一起。然而，这本身并不一定能提升性能，因为数据集的重叠子集可能会递归地与其他重叠子集相互重叠。换言之，只有当相似性这一概念能够由单一属性来定义时，将相似数据放在一起才有效。

为了有效缓存数据的重叠部分，需要一种用于管理计算与数据之间多级依赖关系的工具集。

为了表示数据集与计算之间关联所依据的相似属性，我们将构建一棵树，并将其命名为 数据依赖树。由于每个节点都包含一个可执行的子程序，遍历这棵树即可执行程序本身。与将程序组织成一系列代码块不同，这种基于树的执行路径视图使我们能够存储额外信息，否则处理这些信息会十分繁琐。

具体而言，要求子节点包含与父节点相同的属性后，父节点所定义的整个子树便可执行，而无需进行任何与父节点内属性相关的输入输出操作。显然，父节点属性所定义的所有数据也都存在于子树的每个节点中，因此，对于每棵子树，根节点的数据都与子树中的所有节点重叠。通过在图的较高层级选择频繁出现的属性，在子树执行期间存储祖先节点数据，便能实现一种合理的缓存方式。

需要明确的是，为给定问题生成数据依赖树是一项存在歧义的任务。这尤其是因为我们可以自由缓存当前计算并不需要的数据。这意味着，一个节点可能拥有一些祖先节点，而这些祖先节点的依赖并未被该节点完全体现，但仍会被纳入其中，因为这样一来，该节点便能与更广泛的族群共享共同特征[4]。

2. *Fragmentation Free, Constant Time Pseudo-Allocations*

数据依赖树可以预先计算。此外，在许多情况下，数据生成时就已经具有类似树的结构，因此创建依赖树也就变得轻而易举。

用于缓存时，深度优先搜索[5] 遍历会将祖先节点串联起来，从而保证内存中的写入连续且无间隙。沿树向上移动时，只需缩短内存中索引部分，使其仅容纳当前节点的依赖项。

兄弟节点彼此并不知道对方的存在，因此会毫不犹豫地从父节点数据末尾开始覆盖先前的缓冲区信息。这种行为是我们模型中值得欢迎的特性，因为它使传统的内存分配和内存释放都变得不必.....

要。

[4] 例如，我们可以让非电动发动机依赖于电动发动机，反之亦然，尽管这并非必需。这样做的好处是，我们可以同时处理电动汽车和非电动汽车，而无需因发动机类型而持续进行输入输出操作。[5] 深度优先搜索是一种图遍历算法。在树中，深度优先搜索总会优先选择子节点，而不是兄弟节点。

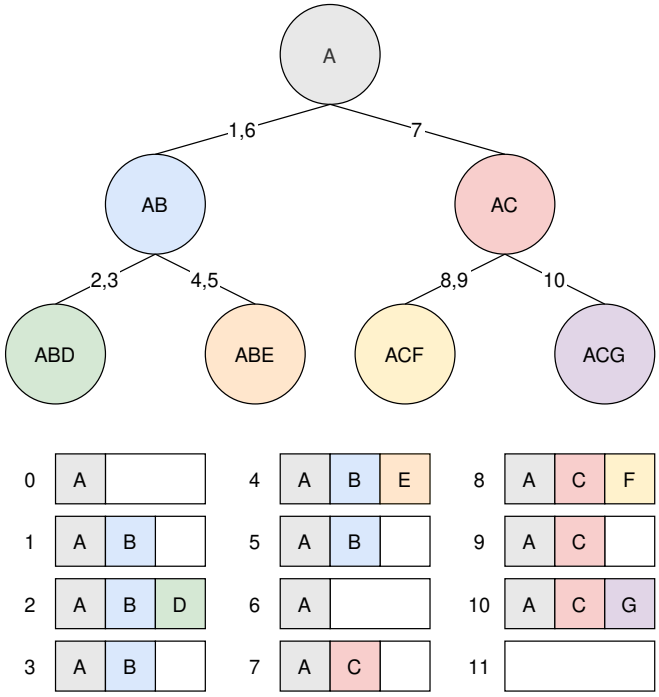

图2. 遍历数据依赖树时的缓冲。数字表示访问顶点的顺序。数组展示了每一步中缓冲区的内容。

可以安全地对数据进行索引，因为祖先节点数据始终位于缓冲区的前端，不会被当前节点触及；这些数据也都是当前依赖项，并由该节点使用。从祖先依赖项末端开始覆写内存，不会影响任何其他节点，因为只有后代节点会使用先前节点的数据。使用深度优先搜索，可以确保在这些数据被舍弃时，所有这些子节点都已访问过。

所述内存模型提供无间隙的顺序写入和读取操作，而且无需进行传统意义上的内存分配和内存释放。内存分配只需返回当前尚未编入索引的内存区域的起始位置。该系统中不存在内存释放和内存泄漏，因为索引取决于节点，所有非后代节点都会忽略该索引。

我们将上述内存模型称为树遍历优化虚拟内存寻址，简称$TTcache$ 。图2展示了 TT缓存的一个简单示例。我们的流程概要见算法 1。作为程序的基础，我们使用深度优先搜索遍历数据依赖树。由于整个操作所需的外部数据量过于庞大，无法装入本地内存，因此需要采用某种缓冲方式。然而，将数据分成多个数据块加载，可能会导致输入输出操作重叠。采用 TT缓存可以消除这类冗余，从而减轻外部与内部内存空间之间数据传输的负担。

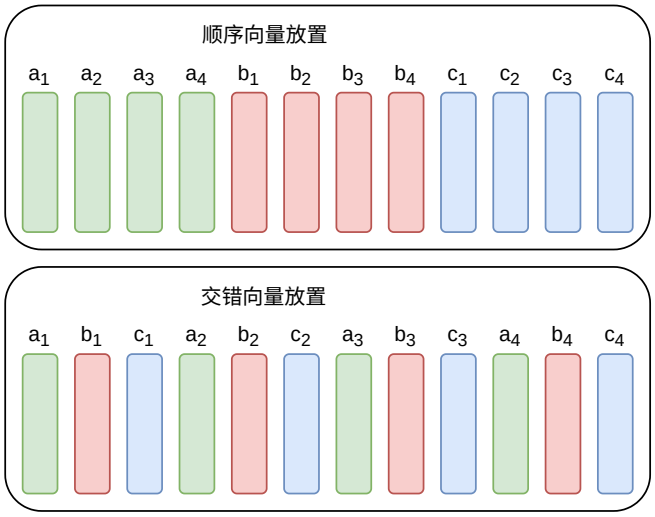

图 3. 步长批量通用矩阵乘法的向量化输出。通常，属于同一矩阵的输出向量按顺序排列（上图）。然而，通过调整输出矩阵的主维度和步长值，可以交错排列属于不同矩阵的向量（下图）。

C. 用于求和的跨步批量矩阵乘法

当 SIMD 工作负载中充斥着大量小型任务时，往往会表现不佳[26]。这并不难理解，因为数据级并行无法减少线程需要执行的指令数量。此外，较小的任务也没有太多内容可供分配给线程。对于无法实现指令级并行的硬件，将较小的任务合并为较大的任务，似乎是提升性能的正确方向。通过采用数量更少、规模更大的 SIMD 并行化块，我们不仅能减少顺序执行的指令数量，也能让调度器更容易高效利用线程；在指令数量不变的情况下处理更多数据，意味着更高的并行性。

对于矩阵乘法的聚合，Intel 和 NVIDIA 均已实现相应的解决方案。批量通用矩阵乘法[6] 函数的工作方式与其非批量版本类似。它们不再处理两个输入矩阵和一个输出矩阵，而是处理两个输入矩阵数组和一个输出矩阵数组。通常还会提供跨步变体，该变体无需数组。相反，这些函数使用偏移量来确定批次内的内存地址。跨步批量变体有望消除数组带来的开销。

遗憾的是，由于每次乘法都是单独计算的，结果也以独立矩阵的形式呈现。当目标是计算多个矩阵乘法的总和时，这可能会带来问题。尽管求解归约[7] 问题相当容易，但这些都是不必要的操作；如果我们按顺序执行矩阵乘法，并将结果累加到一个由各次 GEMM 调用共享的输出矩阵中，就不会有这些操作。

解决该问题的第一步，也是实现批量类型化矩阵乘法而不产生求和归约开销的关键，是了解矩阵的存储方式。按列优先顺序存储时，矩阵是由表示矩阵各列的向量组成的数组；按行优先顺序存储时，这些向量则表示矩阵的各行。对我们来说，重要的是，无论采用哪种方式，矩阵都是由向量组成的数组。

我们方法的第二个支柱是顺序存储的矩阵之间的关系。根据排列顺序，相邻矩阵可以视为水平或垂直拼接而成的较大矩阵。我们无法对这两个聚合矩阵进行运算，原因在于两批矩阵采用了相同的拼接方式。要进行有效的乘法，左侧批次必须水平拼接，而右侧批次则需要垂直拼接。这样，这些大矩阵多出的长度和高度会在乘法过程中消解，最终得到尺寸正常的矩阵。由于我们使用了矩阵乘法本身内置的求和归约，因此无需进行其他计算。

最后，唯一缺少的工具是翻转矩阵批次拼接顺序的方法。此时我们再次采用跨步批量矩阵乘法。尽管这些操作无法直接产生单个结果，但我们可以创建拼接顺序已翻转的中间矩阵批次。为此，只需设置输出矩阵偏移量，使输出矩阵向量交错排列。如图 3 所示，来自不同矩阵但索引相同的向量会彼此相邻排列。在顺序读取内存时，这些向量将构成聚合矩阵中的长向量。

通过依次对聚合后的大型矩阵应用跨步批量矩阵乘法和传统矩阵乘法，我们可以执行批量链式矩阵乘法，而无需在末尾进行求和归约。我们将此方法称为用于求和的跨步批量矩阵乘法，简称$SBMM4S$。

III. 实现细节

前几节中讨论的算法开发成果将针对密度矩阵

[6] 通用矩阵乘法，通常提供单精度、双精度、实数和复数版本。

[7] 在向量化工作负载中，由于单指令多数据的特性，算法会输出多个结果，从而产生归约问题，而并非因为实际需要多个结果。在这种情况下，需要将结果向量归约为单个值。求和归约是一种归约问题，通过求和来完成归约。

重整化群（DMRG）方法 [24]该方法是张量网络态（TNS）算法的一种特殊变体。 [3, 6–9, 11] 我们重点研究一种非常通用的哈密顿量算符形式，该形式已在我们的代码 [27],中实现，可以处理与双粒子散射过程相关的任意形式的非局域相互作用。相应的哈密顿量可写成以下形式

$$\mathcal{H} = \sum_{ij\alpha\beta} T_{ij}^{\alpha\beta} c_{i\alpha}^{\dagger} c_{j\beta} + \sum_{ijkl\alpha\beta\gamma\delta} V_{ijkl}^{\alpha\beta\gamma\delta} c_{i\alpha}^{\dagger} c_{j\beta}^{\dagger} c_{k\gamma} c_{l\delta}, \quad (1)$$

其中，指标 α，β，γ，δ 表示内部自由度，如自旋或同位旋。算符 $c_{i\alpha}^{\dagger}$ 或 $c_{i\alpha}$ 通常表示自旋升降算符或费米子产生湮灭算符。指标 i，j，k，l 表示模，例如实空间表示中的晶格点 [24]、动量空间表示中的能带和动量指标 [28], 、量子化学应用中的分子轨道 [29], 、相对论问题中的旋量 [30]、核结构理论中的质子和中子轨道 [31, 32], 、谐振子势阱中受限粒子的模 [33, 34], ，或科恩–沈轨道 [35] ，等等。为提升该方法的性能，人们采用了多种基于量子信息理论概念的算法解决方案 [36–40]。

在本研究中，我们没有提供上述系统的基准测试结果，而是详细分析了两种化合物的各种扩展特性。此外，我们通过对所谓的有效量子多体哈密顿量进行对角化，并执行所谓的重整化过程，利用了大规模并行、多图形处理器加速的架构。这两个步骤通常占总执行时间的 95%。算法详情可参见多篇综述文章 [2, 3, 6–9, 11]。

从计算机科学的角度来看，张量网络态/密度矩阵重整化群算法的关键特点在于：精确对角化所受指数级扩展的限制可以降低为多项式形式；而底层的张量与矩阵代数运算，即张量积因式分解，可以组织成数百万个相互独立的操作（任务）。因此，稠密矩阵运算会依据所谓的量子数分解表示（扇区）并行执行。完整矩阵的规模以 DMRG 键维数表示， D 决定了计算的精度，同时也决定了所需的计算复杂度。DMRG 的整体扩展关系为 D^3N^4 ，其中 N 表示模的数量，即系统规模。内存需求与 D^2N^2 成正比。

有效哈密顿量的对角化通过 L´anczos 或 Davidson 算法迭代完成，通常占总执行时间的 85%。这涉及一系列矩阵乘法和求和。耗时第二长的过程是算符变换，也称为重整化或分块，该过程可分解为一系列张量积运算、矩阵乘法和求和。重整化步骤占总执行时间的 10%。

A. DMRG 集成

这里我们介绍第 II 节所讨论的各种算法方案的基本实现细节。我们遵循传统的 DMRG 图景，而非基于 MPS 的描述，并强调两者是等价的。此外，我们仅总结与并行化问题相关的技术方面。

1. Quantum number dependent tensor library

在密度矩阵重整化群算法中，网络的模被划分为子系统（块），并基于这些子系统上表示的算符的张量积 [2, 3, 7–9]进行代数运算。此外，算符的矩阵和张量表示会根据量子数分解为不同扇区，即按照行列量子数扇区对及其对应的稠密矩阵[8, 18, 19]存储完整矩阵。这类似于矩阵的稀疏表示，其中非零标量元素按照对应的行列索引对进行存储。底层密度矩阵重整化群线性代数的所有运算均通过这种扇区表示实现。在我们的张量库 [41], 中，同一算符类型的依赖扇区的稠密矩阵还具有第三个索引，用于标记其在给定子系统中的位置。所有张量运算均按照四级层次结构实现：

1. 仅根据扇区描述表（扇区行列对列表）确定所有可能的输出扇区对（扇区表）。2. 根据扇区描述表确定所有可能的输出扇区对，同时生成一个表，用于存储输入算符的所有扇区组合以及相应运算所得结果的输出扇区（任务表）。3. 除了执行第一步和第二步外，还执行实际运算。4.（可选）除了执行第一步至第三步外，还以完整形式表示执行相同运算，并进行自检。

2. Independent operational tasks

对于长程相互作用，哈密顿量式（1）中出现的算符被分配到网络的各个子系统中，这也称为算符分解或矩阵乘积算符（MPO）表示 [7]。此外，还会进行部分求和或预收缩，以降低整体复-

杂度，将哈密顿量的 MPO 表示从 N^4 降至 N^2（详情请参见例如参考文献 [8, 28]）。

在双站点密度矩阵重整化群拓扑中，各模被划分为四个子系统：所谓的左块和右块（由多个模组成）之间由两个代表单个模的中间子系统隔开。 [42]。因此，通过 Davidson 或 L´anczos 算法进行迭代对角化时，矩阵向量乘法可表示为沿四维张量的四个不同维度进行的四次矩阵乘法之和。对于本文研究的模型体系，当两个中间模的局部希尔伯特空间按量子数划分为一维扇区时，也就是说，对于这两个中间模的算符，仅根据行列扇区索引对存储标量时，这些标量的贡献可以预先计算，从而使任务表中存储的每个运算元素最终只需进行标量乘法。这样便将问题简化为一系列如下形式的矩阵乘法

$$B \coloneqq B + \alpha \sum_{i=1}^{p} L_i A R_i^T \,, \tag{2}$$

其中， p 表示独立任务的数量， α是预先计算的常数， L 和 R 分别表示左块和右块算符，而 A 和 B 是量子多体波函数在按量子数扇区分解后的形式下的矩阵表示。更多详情请参阅文献 [7, 43]以及下一节讨论的伪代码算法 2。

在我们的实现中，每种此类算符组合都存储在一个表（算符表）中，并且每种算符组合都配有相应的任务表。因此，需要执行三个主要循环：遍历算符表的各行；对每一行，遍历相应任务表的各行；最后，遍历子系统块内的位置索引。这些循环的乘积给出了执行给定代数运算 [18, 19]所需的独立操作总数。基于上述四级层次结构，在执行给定操作序列之前，首先执行一级和二级过程，以确定相应的表。这意味着，只有完成此类初始化过程后，才会执行表中列出的独立任务，因此，我们的实现非常适合通过动态调度协议在高性能计算基础设施上运行。

3. Constructing the Data Dependency Tree

如第二节所述，将底层的密度矩阵重整化群代数运算组织成相互独立的任务有多种方法，各方法在渐近空间复杂度、I/O 时间复杂度和计算时间复杂度方面各不相同。因此，我们开发了多种算法解决方案，用于根据给定问题所采用的并行化策略，生成第二节中介绍的数据依赖树[18, 19, 23]。

由于 GPU 设备是在较低的抽象层级上处理的，所有 DMRG 算法的数学方面都在主机（中央处理器）端进行。GPU 设备仅用于批量执行基本代数运算。这些高度向量化的运算由所谓的任务表定义，而任务表本身是主机端预处理的直接输出。尽管这些初始步骤实现了复杂的数学模型，但由于没有发生实际的数据变换，其计算开销很小。这种架构使我们能够使用 MATLAB 这一面向复杂系统建模的高级解释型语言，同时，由于整个计算部分均以原生 C++ 实现，数值运算仍能贴近硬件。借助 MATLAB 的 MEX 接口，我们得以在这两个环境之间进行开销很低的调用。

对于具有长程相互作用的模型哈密顿量，DMRG计算需要将大量数据存储在随机存取存储器中，其容量远超当前 GPU内存的大小。我们发现，按照哈密顿量的算符分解（算符表的行）来构建各个任务的并行化方案，比按照波函数的量子数输出扇区对所有任务进行分组的模型更高效[18, 19]。通过应用第 II 节介绍的新方法，我们大幅减少了所需的输入输出操作。

计算期间，每个 GPU设备都会分配一个线程，其主要目标是将计算和输入输出操作都提交到相应设备的 CUDA 流中。结果由同一线程收集，因此设备到主机[8] 数据的检索也可并行进行，从而实现单张卡无法达到的随机存取存储器和 PCI-E通道饱和度。最后，收集到的结果会在主机端合并到波函数的量子数扇区分量中。通过细粒度互斥确保同时执行写入操作时的线程安全。由于每个锁对应的内存段远小于给定扇区的跨度，因此能够实现高吞吐量，同时最大限度地减少线程间的冲突。

随着分配给 GPU 设备的线程数量增加，主机留给本地计算的线程就会减少。如今繁重的计算都由加速器负责，主机剩下的只有些零碎任务。将这些微小操作卸载到其他设备并不值得。所幸，这也意味着原位执行几乎瞬间即可完成，因此我们的 CPU-GPU混合模型对较低的 CPU 核心数并不十分敏感。

[8] 设备到主机：将内存从显存复制到随机存取存储器

B. 竞争算法

并行编程有两种思路[44]。程序由数据和指令组成，因此考虑数据级并行模型和指令级并行模型是顺理成章的。虽然这两个概念都是涵盖多种方法的宽泛术语，但通常最好将二者结合起来[45] ，使当前使用的硬件在处理特定类型的工作负载时能够发挥到极限。

在以下各节中，我们将比较四种算法。其中两种参考设计使用业界标准的高性能线性代数库*IntelMKL*[46]构建。在这两种算法的基础上，我们实现了自己的方法，并根据需要补充或替换相应的 Intel MKL 功能。

为确保测试公平，四种算法均基于同一基础模型，也就是说，计算算法的代码完全相同。所有版本的优化程度也相同，包括对矩阵链乘法[47], 进行最优括号化、缓冲区复用、动态选择原位计算或基于缓冲区的计算、尽可能使用指针运算替代 IO，以及其他诸多优化。

1. *Multithreaded SIMD Matrix Operations*

在第一个参考设计中，我们的代码不使用显式并行化结构，而是将软件与库*Intel Threading*链接。该库包含 Intel MKL API 的函数定义，并具有内置的并行化能力。每次调用向量或矩阵运算时，API 都会调用内置的预编译函数，通过内部 OpenMP 多线程执行。

通过逐一执行向量和矩阵运算，并将代数结构分配给不同线程，我们实际上获得了 SIMD[9] 类型的并行化。不同线程执行相同的指令，但处理的是数据的不同部分。

2. *Task Parallelization with Dynamic Scheduling*

另一种方法是将向量和矩阵运算保留为整体，而改为向每个线程提供不同的运算。这样一来，我们就创建了一种更加抽象、处于指令级别的实现方式。遗憾的是，随着抽象程度的提高，我们现在已经达到了弗林分类法[48], 的顶峰，也就是说，这个版本的程序将严格遵循多指令多数据[10] ，无法移植到单指令多数据硬件上，例如图形处理器。

我们在迷宫的外层循环上使用 OpenMP 并行编译指示，本质上是为每个线程分配迷宫中的一个独立区域。这里不存在单独的任务创建和消费过程。线程一旦进入迷宫，就会持续工作，直到分配的区域被清空。该模型类似于基于线程池的并行 for 循环和传统执行器。

为了使实现更加灵活且更适合迷宫求解，我们使用英特尔对 OpenMP 动态调度[49]的实现。它不是采用静态分配，而是动态地将任务流式分配给线程，使线程池能够弥补任务难度和线程执行速度之间的差异。

3. *Implementing Maze-Runner using Intel OpenMP*

Maze-Runner 版本通过将并行循环转换为任务创建，并允许稍后进行任务消费，扩展了先前多指令多数据版本的功能。这带来了可调节的任务粒度，同时还支持乱序[11] 任务调度。

通过使用 Maze-Runner 线程，任务创建类似于经典的生产者-消费者模型，但无需多角色线程，也避免了多程序多数据[12]模型更高复杂度所带来的调度复杂性。

4. *Implementing SBMM4S and TTCache using CUDA*

对于最终算法，我们再次扩展先前的实现，通过引入对多GPU系统的完整支持，实现了数据级并行化与指令级并行化的结合。

此外，通过允许嵌套的多级 OpenMP 线程处理，Maze-Runner 的中央处理器线程本身如今也能够执行混合 MIMD 和 SIMD。这包括显式的线程池创建、处理和任务分配，以及以多线程向量代数形式实现的隐式第二级并行化构造。

---

[9]单指令多数据，向量化任务通常属于此类别

[10]多指令多数据不要与指令级并行化混淆，后者也可能表示多指令单数据（MISD，即 Multiple-Instruction Single-Data）

[11] *Out of order execution* 通常指对指令进行硬件级动态调度；然而在此上下文中，通过 *out of order*我们仅表示任务消费的顺序不必与任务收集的顺序一致。[12] 多程序多数据中，传统的生产者-消费者模型和流水线会在不同线程上运行不同的子程序。这类架构被归类为多程序多数据。与此同时，我们的 Maze-Runner 模型享有单程序多数据架构的简洁性，同时仍提供可定制的任务处理。

算法 2使用张量算子的 SBMM4S（CUDA）

要求： $\alpha \in \mathbb{R}$
$A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{q\times r}$
$L \in \mathbb{R}^{q\times m\times p}$, $R \in \mathbb{R}^{r\times n\times p}$
确保： $B := B + \alpha(\sum_{i=1}^{p} L_i * A * R_i^{\top})$

```
// 步骤 1：交错输出的批量通用矩阵乘法
cublasDgemmStridedBatched(
    handle,                 // 处理
    CUBLAS_OP_N,  // 转置操作数 A
    CUBLAS_OP_T,  // 转置操作数 B
    m,  r,  n,              // m、n、k
    1.0                     // 阿尔法系数
    A, m, 0,                // A，A 的前导维度，strideA
    R, r, r * n,            // B，B 的前导维度，strideB
    0.0                     // 贝塔系数
    temp, m * p, m,         // C, C 的前导维度, strideC
    pos                     // 批次数
);
// 步骤 2：对拼接后的向量执行 GEMM
cublasDgemm(
    handle,             // 句柄
    CUBLAS_OP_N,  // 转置操作数 A
    CUBLAS_OP_N,  // 转置操作数 B
    q, r, m * p,        // m, n, k
    α                   // 阿尔法系数
    L, q,               // A，A 的前导维度
    temp, m * p         // B，B 的前导维度
    1.0                 // 贝塔系数
    B, q                // C，C 的前
                        导维度
);
```

For detailed descriptions of the abbreviations used above, we guide the readers to the NVIDIA cuBLAS documentation.

这种新的混合中央处理器线程处理机制与手动处理的 C++11 线程相结合，专门用于异步向属于多个 GPU 卡的 CUDA 流数组供给数据。设备端缓冲由我们自主开发的 TTcaching 方法负责处理，该方法展示于算法 1 中。

计算由 SBMM4S 加速，同时仍与 CUDA 库完全兼容。如算法 2 所示，无需使用自定义编写的内核即可实现这一点，因为只需调整 cuBLAS 内核的某些偏移值，就能按合适的顺序生成结果向量。此外，我们的实现还支持部分 SBMM4S，并提供传统执行路径作为回退方案，以应对未满足所有要求的情况。

最终，所得算法成为一种多范式超级混合算法，在软件和硬件的多个层面实现了大规模并行化。

IV. 结果

A. 模型体系

在本节中，我们将介绍两个选定量子化学模型体系的基准测试结果。首先，我们将展示 F_2 分子在 CAS(18,18) 轨道空间 [50] 中的结果；对于高性能计算超级计算机而言，该问题已达到采用精确对角化获得全组态相互作用能的当前极限，即描述 18 个电子在 18 个轨道上的关联，形成维度为 9.075×10^9的希尔伯特空间。接下来将展示 FeMoco 的结果；由于其在固氮中的重要作用 [55]，这一金属中心多参考问题 [19, 51–54] 是现代量子化学的研究重点。在这里，我们将考察文献 [51] 和 [52], 中分别提出的更大模型空间，即 CAS(54/54) 和 CAS(113/76)，其完整希尔伯特空间维度分别为 2.485×10^{31} 和 2.88×10^{36}。后者也是研究并行化问题的良好基准体系，因为文献中还提供了通过不同方法获得的各种参考数据[19, 53]，例如使用多达 10^4 个 CPU 核心。这些体系的精确电子结构计算需要进行大规模密度矩阵重整化群模拟，其键维度 D非常大；因此，必须采用大规模并行化，才能将计算时间控制在合理范围内 [18–20]。

下面我们将展示基准测试结果，这些结果以 DMRG 键维数 D 和 GPU 设备数量为变量。对于 F_2 分子，我们进行了三次 DMRG 扫掠，使基态能量达到 10^{-10}的相对精度；对于 FeMoco CAS(54,54)，我们进行了七次扫描，以获得收敛能量，其误差范围为 10^{-3}，符合现有参考数据 [52–54]。这里提醒一下，一次 DMRG 扫掠包含 N 个独立的 DMRG 迭代周期，每个周期又分解为一系列按顺序执行的算法子程序。还需强调，我们提供的是通过扫描过程测得的总执行时间数据；而此前许多研究通常测量的是最佳 DMRG 配置下的执行时间，也就是说，左块和右块的规模各为系统规模的一半。

由于完整的密度矩阵重整化群迭代周期的计算时间主要由有效哈密顿量的对角化和重整化步骤决定，[18, 19]我们将重点介绍这两种算法过程。通常，基于密度矩阵重整化群导出的矩阵乘积态（MPS）波函数计算算符的期望值和其他可测量量，属于 DMRG 后处理过程。

这里我们指出，在许多高性能计算基础设施中，计算时间受到较短时段的限制（通常设有一天的时限），因此，如果总计算时间超过这一限制，算法必须包含可供重新启动的检查点。为此，可以将矩阵乘积态数据保存到硬盘存储空间中。

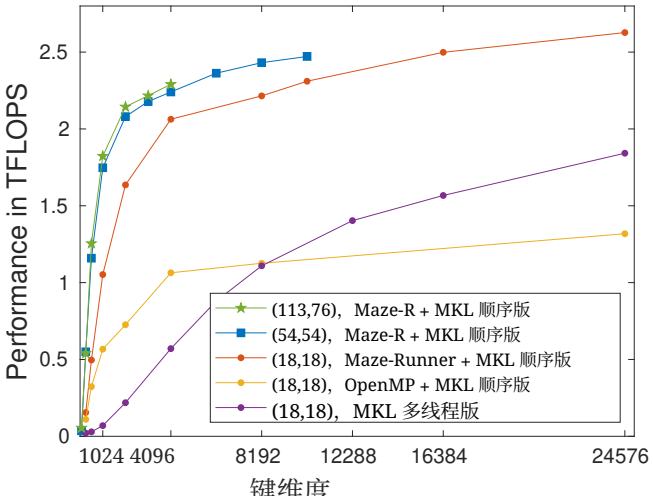

图 4. 在双路 Intel(R) Xeon(R) Gold 5318Y CPU 系统上，以 2.10 GHz 运行、配备 2×24 个物理核心时，F_2 和 FeMoco 化学体系分别在 CAS(18,18) 和 CAS(54,54) 轨道空间中的性能（以太浮点运算每秒计），随 DMRG 键维数的变化。

可在从上一次迭代步骤重新启动密度矩阵重整化群计算时重新加载。

B. 多线程对角化

本节介绍仅依靠 CPU 核心绑定[13] Maze-Runner 线程进行并行化的有效哈密顿量对角化结果，相关线程已在第 II 和 III 节讨论。目前，我们专注于单节点计算，因此，我们目前介绍的 CPU 和 CPU-GPU 混合方案与基于消息传递接口（MPI）的实现相结合的方案，将在后续工作中展示。

图 4 展示了多种基于 CPU 的并行化方案的性能，这些方案源自有效量子多体哈密顿量的对角化，性能随 DMRG 键维数的变化而变化。测量是在一台双路 Intel(R) Xeon(R) Gold 5318Y CPU 系统上进行的，该系统配备 2×24 个物理核心，运行频率为 2.10 GHz。

通过将原本顺序执行的算法与 Intel 的多线程 MKL 库链接，生成的程序便能够同时利用多个核心。这之所以可行，是因为该库包含严格意义上的高级矩阵运算作为可调用的子程序，并在函数定义中构建了并行性。

正如预期，依次执行单个矩阵运算并不适合处理小规模矩阵乘法。可供处理的工作量不多，或者说，可供并行化的工作不多，因此，单个矩阵运算采用多线程所带来的加速比不足以显著提升吞吐量。遗憾的是，性能受到了数百万个依次运行的小规模运算的拖累。尽管该算法似乎能从不断增大的矩阵维度中持续获益——甚至足以与其某个多指令多数据表亲一较高下——但对于极大的 D 值，基于 MKL 的并行化要缓慢得多，距离约 2 太浮点运算每秒的估算值仍相去甚远。

另一种方法是使用单线程 MKL 子程序并配合 OpenMP 指令，形成包含 MKL 调用的并行循环。对于较小和中等的 D 值，这种方法的性能有了显著提升。然而，由于循环体作为独立执行的任务时表现欠佳，性能会在显著较低的阈值处趋于饱和。

相比之下，Maze-Runner 线程不会受到此类问题的影响。它全面支持类似生产者-消费者模型的自定义任务调度，任务创建与底层数学模型的控制结构[14] 彼此解耦。这使负载均衡更加精细，最终也能更快达到性能饱和，尤其是在 CAS 空间较大时。此外，该算法能迅速达到一个性能水平，似乎是其竞争对手难以企及的，即使键维数高得多也是如此。

C. 多 GPU 加速对角化

图5展示了我们在第II节和第III节讨论的中央处理器-多图形处理器混合模型的基准测试结果。性能测量是通过对有效量子多体哈密顿量进行对角化得出的，并考察了不同 D 值下 GPU 设备数量的影响。

与仅使用中央处理器的流程相比，即使只使用一个GPU设备，也能使性能提升两到四倍。在多图形处理器配置下，性能与图形处理器数量的扩展关系呈现出近似线性的趋势。随着键维度增大，曲线的斜率也会随之增加。这意味着，由于矩阵和张量规模更大，GPU 硬件的计算能力能够得到更高效的利用。对于 $D = 24576$，单 GPU 配置下达到了 11 万亿次浮点运算每秒的性能，而使用八张卡时，同一运行达到了 70 万亿次浮点运算每秒，这非常接近八块 NVIDIA A100-PCIE-40GB GPU 在双精度浮点下的上限——

[13] 将线程固定到 CPU 核心，指的是线程（软件）与核心（硬件）之间一对一的对应关系。线程数量不能超过核心数量；理想情况下，调度器不会在不同核心之间迁移线程，因此称为“固定”。

[14] 控制结构是程序的基本构建块。常见的控制结构包括： *sequence*, *loop* 和 *condi-tional*

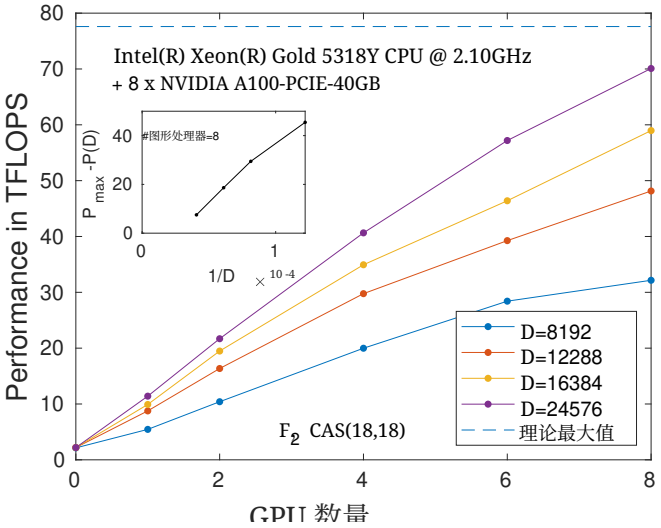

图 5. 对 F_2molecule 的性能进行测量，单位为万亿次浮点运算；该分子对应于 CAS(18,18)，性能随 GPU 设备数量变化，并针对不同的固定 DMRG 键维数取值进行测量。计算在一台双路 Intel(R) Xeon(R) Gold 5318Y CPU 系统上完成，该系统配备 2×24 个物理核心，运行频率为 2.1 GHz，并搭载八个 NVIDIA A100-PCIE-40GB GPU 单元。插图展示了使用八个 GPU 设备时，性能相对于估算理论最大值的扩展情况，$P_{\max}$ 即 DMRG 键维数倒数的函数。

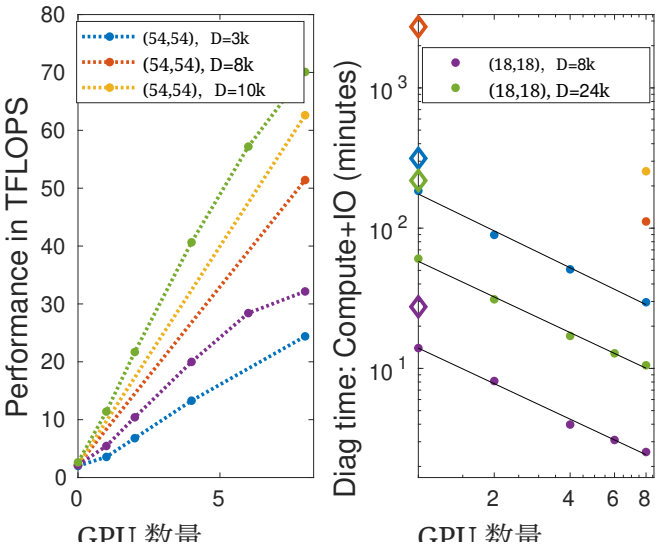

图 6。（左图）对于两种不同 CAS 空间，在不同固定 DMRG 键维数下，性能（以太浮点运算每秒计）随 GPU 设备数量的变化。CAS(18,18) 和 CAS(54,54) 分别对应 F_2 和 FeMoco 分子系统。（右图）对于这两种分子系统，在固定键维度下，用于对角化有效哈密顿量的总时间（秒）随 GPU 设备数量的变化。空心菱形表示仅使用 CPU 完成的计算，即未使用任何 GPU 设备。虚线为视觉参考线。对于 FeMoco，$D = 8192$ 和 $D = 10240$ 的结果仅针对 8 个 GPU 设备确定。实线表示一阶多项式拟合结果，对应的指数分别为 -0.8438、-0.8416 和 -0.8729。

设备（$P_{\max} = 8 \times 9.7$ 万亿次浮点运算） [56]。插图显示了性能相对于理论估计最大值的扩展关系，该关系是 DMRG 键维数的倒数。

图 6 总结了 F_2 和 FeMoco 分子体系中，有效 Hamiltonian 对角化过程所花费的总执行时间随 GPU 设备数量的变化情况；该总执行时间包括与矩阵-向量乘法相关的计算时间以及相应的 IO 开销，并针对不同的固定键维度取值进行了测量。空心菱形表示仅使用 CPU 核心进行的计算，即未使用 GPU 设备的情况。对于计算量要求极高的模拟（FeMoco），我们仅给出了使用八个 GPU 设备时的结果；这些结果由双路 AMD EPYC 7702 CPU 提供 2×64 个核心，并配备八个 NVIDIA A100-SXM4-40GB 设备。虚线用于引导视线，实线则表示一阶多项式拟合的结果。

对于少数选定的数据集，相对于仅 CPU 上限测得的相应加速比如图 7 所示。显然，对于中等和较大的 D 值，使用单个 GPU设备已经可以实现两到三倍的加速比。然而，对于较小的键维数，开销可能会占据主导地位，因此正如预期的那样，与仅 CPU 实现相比，计算需要更多时间。CPU 与图形处理器加速计算以虚线表示。

除了单个图形处理器带来的加速比外，测得的总执行时间与 GPU 设备数量之间的关系，在双对数尺度上呈线性，且不受 D 值和系统规模的影响（见图 6 右图）。事实上，一阶多项式拟合得到的指数分别为 -0.8438、-0.8416 和 -0.8729。这意味着，GPU 设备数量翻倍时，总时间几乎减半；这种近乎完美的幂律缩放一直保持到使用八个 GPU 设备为止。所得指数接近负一，表明输入输出通信开销已被有效地隐藏在代数运算之后，而缩放主要由底层矩阵和张量乘法的计算复杂度决定。

对于更大的系统规模，即更大的 CAS 空间，可以获得更高的万亿次浮点运算值；与仅使用中央处理器的方案所能达到的上限相比，加速比也更加显著。值得注意的是，对于较小的 CAS(18,18)计算，当键维度非常大时，性能提升已达到阈值。这是因为，从 $D = 16384$ 增加到 $D = 24576$，两种方案的性能都仅提升了边际性的 ≈ 10 %：

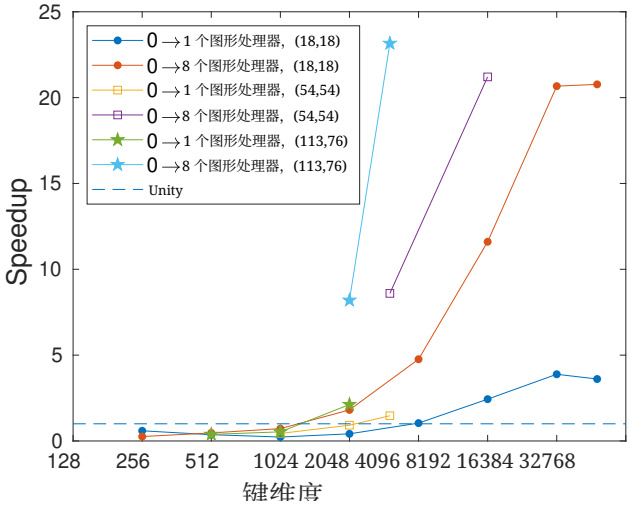

图7. 所选数据集的加速比随键维度的变化。位于 Unity 处的虚线可用于确定一个最小键维度，在该维度下，依赖于系统的 GPU加速求解比仅使用 CPU 的极限更快。

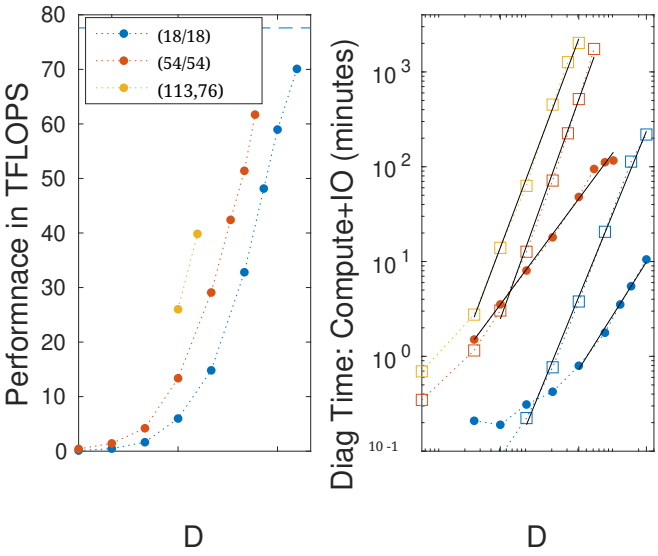

图 8. 对 F_2 CAS(18,18)、FeMoco CAS(54,54) 和 CAS(113,76)，八 GPU 加速对角化过程的性能和相应总耗时，分别以太浮点运算每秒和分钟为单位表示，并展示其随 DMRG 键维数的变化。右图还包含仅使用 CPU 时的数据（方形符号）。点线仅作视觉参考，虚线表示理论最大值。实线为一阶多项式拟合结果。

仅使用中央处理器的方案（见图 4）和多图形处理器加速方案（图 5）。这与以下事实有关：起初，随着 D 值增大，扇区数量和扇区规模都会增加；然而，当键维数非常大时，所有扇区都已被填满，只有扇区规模继续增加。对于更大的 CAS 空间，预计只有在 D 值大得多时才会出现这种饱和。

图8展示了使用八个 GPU 设备进行对角化所得的性能和相关执行时间，它们随 F_2和 FeMoco 分子系统的键维度变化。右图中还用方形符号标出了仅使用 CPU 时对应的数据。显然，在图的右侧面板中，对于很大范围的 D 值，数据点在双对数尺度上落在一条直线上。这里强调，不利用基于量子数的对称性时，单线程计算的计算时间扩展规律为D^3。而在采用基于量子数的分区稀疏表示，并结合我们的 Maze-Runner 求解方案（见第II节和第III节）时，即使仅使用 CPU，所得指数也显著更小，分别为 2.24、2.53 和 2.43。因此，正如预期，利用对称性并结合基于 CPU 的并行化，显著降低了扩展指数，而 GPU加速求解进一步提升了这一效果。对于 8 GPU 加速求解，一阶多项式拟合得到的指数分别为 1.45 和 1.23，对应 CAS(18,18) 和 CAS(54,54)。最终指数接近 1，表明几乎已达到线性扩展。

为了更好地理解相关时间尺度，我们从表 I 中收集了图 6 右侧面板的数据。矩阵-向量乘法时间

对于我们的混合中央处理器-多图形处理器解决方案，包括 IO 开销，使用 F_2 CAS(18,18)、$D = 8129$ 和三次密度矩阵重整化群扫描需要 1.2 分钟。使用相同的设置，但将 D =设为 24576，则需要 10.3 分钟。对于 FeMoco CAS(54,54)，使用 $D = 3072$ 并进行七次扫描需要半小时，而将 $D = 8192$ 设得更高则需要 1.9 小时。最后，使用 $D = 10240$ 时，执行时间增加到 3.9 小时。此外，对于非常大的 CAS(113,76)，即使进行七次扫描且 D 达到几千，执行时间仍在几小时范围内。这清楚地表明，基于图形处理器的技术能够彻底革新张量网络态方法在非常大型、复杂且强关联（多参考）问题中的应用。

对于其他各种量子多体系统，例如核壳模型 [57]和二维量子晶格问题[58, 59], ，我们也得到了类似结果；然而，由于篇幅限制，这些结果将在后续工作中呈现。

D. 重整化

在本节中，我们展示通过重整化过程（也称为分块步骤）获得的结果。与对角化步骤不同，这里会使用一系列张量积运算和矩阵乘法构造多个算符。此外， the

CAS	D	扫描次数	对角线：计算+输入输出
(18/18)	8192	3	1.7 分钟
(18/18)	24756	3	10.3 分钟
(54/54)	3072	7	34 分钟
(54/54)	8192	7	1.9 小时
(54/54)	10240	7	3.9 小时
(113/76)	2048	7	1.6 小时
(113/76)	3072	7	2.2 小时

表 I. 对于 F_2 和 FeMoco 分子系统，在不同密度矩阵重整化群参数下，八个经图形处理器加速的对角化步骤的总计算时间及 IO 开销。

底层代数运算的内存需求也会随块大小的增加而上升，因为必须生成块内所有位置相关的算符。因此，与对角化步骤相比，IO 开销要大得多。这还需要对调度算法进行少量修改，但仍采用了第II节和第III节中概述的相同通用框架。

主要瓶颈可确定为设备到主机 CUDA 内核，它们负责从设备中取回计算所得的数据，以便将其迁移到最终的主机端数据结构中。在每个设备上，收集之前，最终矩阵是由任务表定义的数百万次运算的结果。这样的工作流程与对角化过程并无太大不同。然而，在重整化过程中，会省略求和归约步骤，即不会对单个量子数相关扇区中矩阵分量的块内位置索引求和。原本只会作为加数、无需离开设备内存的内容，如今却成为必须传回主机的最终结果。这大幅增加了每次重整化调用结束时的设备到主机 IO 内核负担，使该过程更加受 IO 限制。

对于 F_2 CAS(18,18) 分子，测得的性能随 GPU 设备数量的变化汇总于图 9。测试了两个不同的 D 值，每种情况下均进行了 3 次 DMRG 扫掠。

这里，将对角化和重整化的总执行时间分为两部分。在这两种情况下，分别展示了仅使用中央处理器的部分和经图形处理器加速的部分——后者还包括相应的 IO 开销。显然，重整化过程的执行时间也大幅缩短。即使只使用一个设备，获得的加速比也十分显著。然而，由于 IO 需求量大，8 张卡的性能仅为 ≈ 12 太浮点运算每秒，因此重整化的扩展性不如对角化步骤。

事实上，对于 $D = 8129$，将设备数量从 4 个增加到 8 个会导致重归一化步骤的执行时间略有增加。这是因为计算时间本身已经非常短，进一步并行化只能带来微乎其微的改进。从 4 个图形处理器开始，进一步增加设备数量将导致图形处理器开销相对更快地增长，最终超过通过增加设备所获得的计算性能提升速度。因此，这不仅抵消了额外显卡带来的额外性能，还会整体上使总执行时间变差。

对于较大的 D 值，情况则不再如此，因为计算复杂度越高，执行所需的计算时间也越长。增加 GPU 数量可以缩短这部分时间，而且需要缩减的时间越多，运行时间方面的收益也越大。因此，D 值越大，计算相较于 IO 和 GPU 开销就越占优势。

尽管使用加速器的代价可能会随 D 值略微增长，但即使在最坏情况下，额外开销也仅包括与底层矩阵和张量代数[15]相关的输入输出操作。这意味着，所有非计算子程序相对于矩阵前导维度的时间复杂度均为 $\mathcal{O}(n^2)$，而计算本身——受益于图形处理器数量的持续增加——则为 $\Omega(n^3)$，因为其中包含大规模矩阵乘法。

FeMoco CAS(54,54)也呈现出类似的情况。从较小和中等的 $D \simeq 3072$ 值开始，扩展性相当差，但对于较大的 D 值，性能会迅速提升。

改进重整化的一个自然步骤是，将不同算符所需的代数运算分配到不同节点上执行。因此，单个算符重整化中耗时最长的部分将决定总体执行时间。根据我们的测量结果，我们估计，将混合式中央处理器-多图形处理器解决方案与我们的消息传递接口实现相结合，将立即缩短重整化过程的执行时间。我们估计性能将提升 10 倍。

此外，通过部分求和[8] 来准备辅助算符的预处理和后处理步骤也可以卸载到加速器上，因此，黄色区域所示的大部分耗时都可以减少，并转移到紫色区域所示的图形处理器耗时中。经过这些改进后，图 9 中的黄色区域将几乎不可见。

同样，目前与 L´anczos 和 Davison 对角化过程相关的所有代数运算，除矩阵向量乘法外，均在中央处理器上执行，但其中相当一部分也可以迁移到加速器上。因此，蓝色所示的耗时将低于红色所示的耗时。

[15] 如果它未与底层代数运算绑定，那么开销就与 D 值无关，也就是说，从计算的角度来看，其时间复杂度为 $\mathcal{O}(1)$。

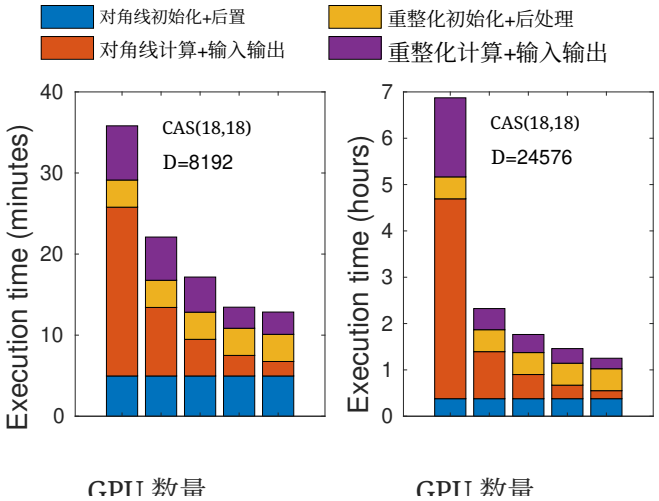

图 9. 使用三次 DMRG 扫掠，对 F_2 分子在 CAS(18,18) 轨道空间中的总执行时间按对角化和重整化步骤进行分解，$D = 8192$ （左图）以及 $D = 24576$ （右图）。

E. 非阿贝尔对称性和更一般张量拓扑的利用

从技术角度来看，利用更多对称性会增加基于量子数分解的矩阵和张量的扇区数量，最终降低内存需求；更重要的是，这会显著增加底层代数运算所需执行的独立任务数量。对于 $U(1)$ 对称性，这一过程很直接；然而，对于非阿贝尔对称性，则必须采用更精细的数学框架 [60–65]。在我们的张量库中，与非阿贝尔对称性相关的层与 MPS 层分离 [66]，，因此混合 CPU-多 GPU 内核只需进行少量修改，即每项操作只需额外乘以一个预先计算的标量，并在 GPU 设备上额外复制一次数据。因此，预计会有约 20% 的性能下降，但在精度相同的情况下，键维度通常可缩小 2 至 3 倍，从而轻松实现一个数量级的加速比。我们将在后续工作中介绍扩展后的 $SU(2)$ 混合 CPU-多 GPU 内核的细节和结果。

我们的结果已针对一维密度矩阵重整化群拓扑进行了验证，但混合式 CPU-多图形处理器内核可以轻松扩展到更一般的张量网络拓扑 [7–9]。对于更一般的网络，随着子系统（块）数量增加，底层代数运算中的独立任务数量也会大幅增加。这使得数据和任务的分配方式更加灵活，并可分布到数量显著更多的 GPU 设备上，从而能够充分利用 GPU 硬件所提供的计算能力。

效率更高。

从图5和图8还可以清楚地看出，对于较大的系统规模和键维数，最优的 GPU 设备数量在使用八张卡时尚未达到。因此，基于消息传递接口的多节点多GPU版本有望进一步提升性能，使密度矩阵重整化群迈入千万亿次级计算领域 [23]。我们将在后续工作中对此进行分析。

F. 大规模并行 DMRG-RAS-X 方法

最近，DMRG算法与限制活性空间（RAS）方法的概念相结合 [67]，严谨的数学分析由此提出了一种高效且稳定的外推流程，只需使用有限的 CAS 空间 [54]，即可在化学精度范围内获得全组态相互作用基态能量。将这一新方法 DMRG-RAS-X 与混合 CPU-多 GPU 内核结合起来（参见文献 [54]中 FeMoco 的结果），有望使 DMRG 成为一种常规方法，日常用于求解需要大型轨道空间的复杂强关联（多参考）问题。此外，通过 GPU加速求解即时采样 RAS空间，并结合进一步的数学发展，将轻松提升 DMRG 高效处理数百个模的系统中静态和动态相关性的能力。

G. 绿色 DMRG

最后，我们来分析张量网络态计算的功耗；如今，由于能耗需求和成本高昂，这已成为最重要的问题之一。$2 \times$ Intel(R) Xeon Gold 5318Y CPU 的热设计功耗（TDP）为2×165 瓦特 [68]，，因此，我们基准测试中 2.5 万亿次浮点运算每秒的结果估算出的能耗为 ≈ 7.5 每瓦十亿次浮点运算。

相比之下，对于 NVIDIA A100-PCIE-40GB 设备，其 TDP 为 250 瓦 [56]。因此，我们的 8 卡加速混合算法以 70 TFLOPS 的性能实现了 $70000/(330 + 8 \times 250) = 30.04$每瓦十亿次浮点运算。这意味着能效可以提高四倍。换言之，对于给定的计算任务，处理器产生的能耗成本可以降低至原始消耗的四分之一。在这里，我们仅取了相关量的上限；而在实际计算过程中，GPU 设备的能耗会显著波动，因此甚至可以获得更高的比率。

V. 结论

在本研究中，我们提出了新颖的算法解决方案及其实现细节，以突破当前高性能计算基础设施上张量网络态算法的限制；该基础设施建立在先进的硬件和软件技术之上。主要贡献包括以下几项（相关性能指标见图 10）：

- **Maze-Runner:** 一种轻量级、低复杂度的并行构造，能够实现类似生产者-消费者的任务处理，但无需为线程强制分配角色。由于采用同质线程化和任务创建优先级排序，调度过程得以简化至几乎不言而喻的程度。这使其在面对波动的任务创建时间和执行时间时，能够保持较低的开销，并避免调度缺陷。

- **TTCache：** 虚拟内存寻址，旨在大幅减少冗余输入输出操作，并消除内存碎片和分配开销。TTCache 通过将数据因子分解为属性，然后将这些属性分层映射到执行块来运行。执行过程通过遍历树状结构完成，其中彼此靠近的节点依赖于大致相同的一组属性。

- **SBMM4S：** 带有固有零成本求和归约的批处理类型矩阵乘法。通过将整个矩阵批次与交错矩阵的串联向量数组相乘，生成单个结果。通过使用具有特定偏移值的跨步批处理类型矩阵乘法来获得链式矩阵乘法的中间结果，从而实现交错。

本文展示了 F_2 分子的基准测试结果，该分子代表了超级计算机上精确对角化的极限；同时也展示了 FeMoco 团簇的基准测试结果，该团簇是现代量子化学研究的重点，也是方法开发基准测试的对象。我们证明，采用大规模并行混合中央处理器与多图形处理器的架构，能够充分利用现代中央处理器和图形处理器硬件的全部能力，高效地对有效量子多体哈密顿量进行对角化。

使用 GPU 设备时，即使仅采用单图形处理器配置，对于较大的 D 值，加速比也达到了 2 至 4 倍。此外，根据测量结果，我们发现性能与所用加速器的数量呈线性关系。如此高效的扩展能力，使八块 NVIDIA A100-PCIE-40GB GPU 的理论上限——77.6 太浮点运算每秒——

已经几乎达到设备数量的上限。我们的努力显著缩短了计算时间：总耗时随 GPU 设备数量的增加而线性下降，

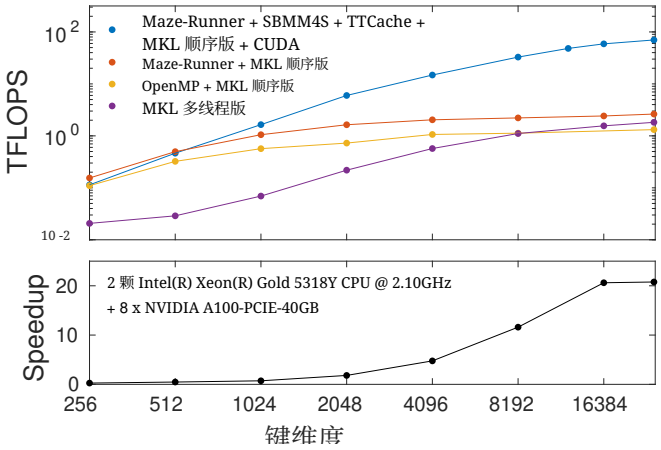

图 10. 汇总图 4 和图 7 中重新收集的数据，以总结通过对角化过程在 CAS(18,18) 轨道空间中对 F_2 分子实现的最终性能。加速比根据 8×GPU 加速求解（蓝色）与仅使用 CPU 的版本（橙色）之间的差异进行测量。

在双对数坐标下尤为明显。因此，当D 值足够大时，使用八个 GPU 设备可将性能提升到 20 倍以上。

作为当前工作的一部分，我们正在进一步优化适用于中等键维度值的 $SU(2)$ 自旋适配图形处理器方案，开发并实现新的调度系统和优化协议，将密度矩阵重整化群方法的其他部分转换为高效的图形处理器加速代码，并将我们的消息传递接口协议迁移到新开发的混合中央处理器-多图形处理器方案中，从而能够利用多个节点。这些升级结合起来，足以轻松将密度矩阵重整化群方法的性能提升至千万亿次级计算水平，以解决具有长程相互作用的相互作用量子多体问题。

致谢

作者感谢 Tam´as Kozsik 和 Mikl´os Werner 进行的有益讨论。本研究由匈牙利国家研究、开发与创新办公室（NKFIH）通过拨款编号 K134983 和 TKP2021-NVA-04，以及匈牙利量子信息国家实验室提供支持；同时还获得了由慕尼黑工业大学高等研究院资助的 Hans Fischer 高级研究员计划的支持。O.L. 还获得了可扩展与预测性激发及关联现象方法中心（SPEC）的支持。该中心作为计算化学科学计划的一部分，由美国能源部（DOE）科学办公室基础能源科学办公室化学科学、地球科学与生物科学部在太平洋西北国家实验室资助。我们感谢 Wigner 科学计算实验室提供计算支持

（WSCLAB）、E¨otv¨os Lor´and 大学，以及政府信息技术发展局提供支持，使我们得以使用 Komondor 超级计算机。模拟还在国家斯图加特高性能计算中心（HLRS）的 HPE Apollo Hawk 超级计算机，资助编号为 MPTNS/44246。

[1] H. Anzt、A. Bienz、P. Luszczek 和 M. Baboulin。*High Performance Computing. ISC High Performance 2022 International Workshops: Hamburg Germany May 29 – June 2 2022 Revised Selected Papers*、、、、。《计算机科学讲义》。施普林格国际出版，2023年。[2] Ulrich Schollw¨ock。《密度矩阵重整化群》。*Rev. Mod. Phys.*，77：259–315，2005年4月。[3] Reinhard M. Noack、Salvatore R. Manmana。用于相互作用量子系统的基于对角化和数值重整化群的方法。载于*Conference Proceedings* 2005 。AIP，。[4] F. Verstraete、V. Murg 和 J.I. Cirac。量子自旋系统的矩阵乘积态、投影纠缠对态和变分重整化群方法。*Advances in Physics*，57(2)：143–224，2008年。[5] ¨O. Legeza、R.M. Noack、J. S´olyom 和 L. Tincani。量子信息在密度矩阵重整化群中的应用。载于 H. Fehske、R. Schneider 和 A. Weiße 编，*Computational Many-Particle Physics*，《*Lecture Notes in Physics*》第739卷，第653–664页。施普林格，柏林，海德堡，2008年。[6] Garnet Kin-Lic Chan、Jonathan J. Dorando、Debashree Ghosh、Johannes Hachmann、Eric Neuscamman、Haitao Wang 和 Takeshi Yanai。量子化学中的密度矩阵重整化群拟设简介。载于 Stephen Wilson、Peter J. Grout、Jean Maru*Frontiers in Quantum Systems in Chemistry and*ani、Gerardo Delgado-Barrio 和 Piotr Piecuch 编，*Physics* 18 *Progress in Theoretical Chemistry*，第 卷，。*and Physics* 2008施普林格，荷兰，。[7] Ulrich Schollw¨ock。矩阵乘积态时代的密度矩阵重整化群。*Annals of Physics* 326 1 96 192, 2011 2011 ，(): –。一月特别期。[8] Szil´ard Szalay、Max Pfeffer、Valentin Murg、Gergely Bar-¨cza、Frank Verstraete、Reinhold Schneider 和 Ors Leg-eza。从头算量子化学中的张量积方法与纠缠优化。*Int. J. Quan-* ，(): –。*tum Chem.* 115 19 1342 1391, 2015[9] Rom´an Or´us。张量网络实用简介：矩阵乘积态和投影纠缠对态。*Annals of Physics*，349(0)：117–158，2014年。[10] Venera Khoromskaiaab 和 Boris N. Khoromskijb。量子化学中的张量数值方法：从 Hartree–Fock方法到激发能。*Phys. Chem. Chem.* ，，。*Phys.* 17 2015[11] Alberto Baiardi 和 Markus Reiher。化学与分子物理学中的密度矩阵重整化群：近期进展与新挑战。*The Journal of Chemical Physics* 152 4 040903, 2020 ，(): 。[12] Davide Ferrari、Angela Sara Cacciapuoti、Michele Amoretti 和 Marcello Caleffi。分布式量子计算的编译器设计。*IEEE Transactions on Quantum Engineering* 2 1 20, 2021 ，：–。[13]Chu Guo、Yong Liu、Min Xiong、Shichuan Xue、Xi- ang Fu、Anqi Huang、Xiaogang Qiang、Ping Xu、Jun- hua Liu、Shenggen Zheng、He-Liang Huang、Mingtang Deng、Dario Poletti、Wan-Su Bao 和 Junjie Wu。基于投影纠缠对态的通用量子电路模拟器与量子霸权前沿。*Phys. Rev. Lett.*，123:190501，2019年11月。[14]Francesco Tacchino、Alessandro Chiesa、Stefano Car- retta 和 Dario Gerace。量子计算机作为通用量子模拟器：现状与展望。*Advanced Quantum Technologies* 3 3 1900052, , (): 。2020[15] Xiaosi Xu、Simon Benjamin、Jinzhao Sun、Xiao Yuan 和 Pan Zhang。一项艰巨任务：量子计算机的经典模拟，2023年。[16] G. Hager、E. Jeckelmann、H. Fehske 和 G. Wellein。共享内存系统上密度矩阵重整化群算法的并行化策略。*Journal of Computational Physics* 194 2 795 808, 2004 ，(): –mar 。[17] E. M. Stoudenmire 和 Steven R. White。实空间并行密度矩阵重整化群。*Physical Review B* 87 15 2013 ，(), apr 。.[18] Csaba Nemes、Gergely Barcza、Zolt´an Nagy、Ors Legeza 和 P´eter Szolgay。千处理器架构上的密度矩阵重整化群算法：实现与权衡。*Computer Physics Communications* 185 6 1570 1581, 2014 ，(): –。[19]Jiri Brabec 、 Jan Brandejs 、 Karol Kowalski 、 Sotiris Xantheas、 ¨Ors Legeza 和 Libor Veis。大规模并行量子化学密度矩阵重整化群方法。*Journal of Computational Chemistry*，42 8 534 544，2021 (): –。[20] Huanchen Zhai 和 Garnet Kin-Lic Chan。低通信、高性能的从头算密度矩阵重整化群算法。*Journal of Chemical Physics* 154 22 0021 9606, 2021 ，(): –。[21] Johnnie Gray 和 Stefanos Kourtis。超优化张量网络收缩。*Quantum*，5，2021年。[22] Dmitry Lyakh、Thien Nguyen、Daniel Claudino 和 Eu- gene Dumitrescu。Exatn：可扩展的 GPU 加速高性能通用张量网络处理，面向百亿亿次级计算。*Frontiers in Applied Mathematics and Statistics*，8 2022，。[23] Martin Ganahl、Jackson Beall、Markus Hauru、Adam G.M. Lewis、Tomasz Wojno、Jae Hyeon Yoo、Yijian Zou 和 Guifre Vidal。使用张量处理单元实现密度矩阵重整化群。*Quantum* 4 010317, 2023 ，：。[24] Steven R. White。量子重整化群的密度矩阵表述。*Phys. Rev. Lett.*，69:2863–2866，1992年11月。[25] Isaac Gelado 和 Michael Garland。面向吞吐量的GPU内存分配。*Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming* 2019 ，。

[26] Jack Dongarra、Sven Hammarling、Nicholas J. Higham、Samuel D. Relton、Pedro Valero-Lara 和 Mawussi Zounon。现代高性能计算系统上批处理 BLAS 的设计与性能。*Procedia Computer Science* 108 495 504, 2017 , : –。

国际计算科学会议，ICCS 2017，2017年6月12日至14日，瑞士苏黎世。[27] ¨Ors Legeza、Libor Veis 和 Tam´as Mosoni。DMRG- budapest：用于凝聚态物质、量子化学和原子核壳层 DMRG 计算的程序，2022年。[28] T. Xiang。动量空间中的密度矩阵重整化群方法。 *Phys. Rev. B*，53:R10445–R10448，1996年4月。[29] Steven R. White 和 Richard L. Martin。使用密度矩阵重整化群进行从头算量子化学计算。

The Journal of Chemical Physics，110(9):4127–4130，1999年。[30] Stefan Knecht、 ¨Ors Legeza 和 Markus Reiher。通信：四分量密度矩阵重整化群。

The Journal of Chemical Physics，140 4 041101，2014 ( ): 。[31] Jorge Dukelsky 和 Stuart Pittel。有限费米系统的密度矩阵重整化群。 *Reports on Progress in Physics* 67 4 513, 2004 , ( ): 。

[32] ¨O. Legeza、L. Veis、A. Poves 和 J. Dukelsky。用于核结构计算的高级密度矩阵重整化群方法。 *Phys. Rev. C*，92:051303，2015年11月。[33] ¨Ors Legeza 和 Christian Schilling。配对势在广义泡利约束饱和中的作用。*Phys. Rev. A*，97:052105，2018年5月。[34] I. Shapir、A. Hamo、S. Pecker、C. P. Moca、 ¨O. Legeza、G. Zarand 和 S. Ilani。一维电子维格纳晶体的成像。 *Science* 364 6443 870 875, , ( ): –。2019

[35]Gergely Barcza、Viktor Iv´ady、Tibor Szilv´asi、M´arton V¨or¨os、Libor Veis、 ´Ad´am Gali 和 ¨Ors Legeza。基于平面波科恩-沈轨道的 DMRG：缺陷氮化硼案例研究。

J. Chem. Theory Comput.，17 2 1143 1154，2021 ( ): –。[36] ¨O. Legeza、J. R¨oder 和 B. A. Hess。控制密度矩阵重整化群方法的精度：动态块态选择方法。*Phys. Rev. B*，67:125114，2003年3月。[37] ¨O. Legeza 和 J. S´olyom。利用量子信息熵优化密度矩阵重整化群方法。 *Phys. Rev. B*，68:195116，2003年11月。[38] G. Barcza、 ¨O. Legeza、K. H. Marti 和 M. Reiher。不同分子结构电子态的量子信息分析。 *Phys. Rev. A*，83:012508，2011年1月。[39]E. Fertitta、B. Paulus、G. Barcza 和 ¨O. Legeza。通过从头算密度矩阵重整化群方法研究类金属-绝缘体转变。*Phys. Rev. B*，90:245129，2014年12月。[40] C. Krumnow、L. Veis、 ¨O. Legeza 和 J. Eisert。张量网络态中的费米子轨道优化。 *Phys. Rev.* , : 11 *Lett.* 117 210402, 2016 月。[41] ¨Ors Legeza。密度矩阵重整化群方法的矩阵与张量库，1995—2023年。[42] S. R. White 和 R. M. Noack。实空间量子重整化群。 *Phys. Rev. Lett.*，68:3487–3490，1992年6月。

[43] Szil´ard Szalay. 多体纠缠度量。*Phys. Rev. A*，92:042329，2015年10月。[44] R. Espasa 和 M. Valero。充分利用指令级并行和数据级并行。 *IEEE Micro*，17(5):20–27，1997年。[45] Jaspal Subhlok、James M. Stichnoth、David R. O'Hallaron 和 Thomas Gross。在多计算机上充分利用任务并行和数据并行。

SIGPLAN Not.，28 7 13 22, 1993 ( ): – July 。[46] Endong Wang、Qing Zhang、Bo Shen、Guangyong Zhang、Xiaowei Lu、Qing Wu、Yajuan Wang、Endong Wang、Qing Zhang、Bo Shen 等。英特尔数学核心函数库。 *High-Performance Computing on the Intel® Xeon Phi*TM*: How to Fully Exploit MIC Architectures* 167 188, , 页码 – 。2014

[47]Igor Chikalov、Shahid Hussain 和 Mikhail Moshkov。针对不同代价函数的矩阵链乘法顺序优化。见 *SOFSEM*，第157–165页。施普林格，2011年。[48] Neha K Shinde 和 Sunil R Gupta。弗林分类法中各类架构——方法比较。*IJISET-International Journal of Innovative Science Engineering and Technology* 2 9 135 140, 2015 , , ( ): – 。[49] Alejandro Duran 、 Julita Corbal´an 和 Eduard Ayguad´e。OpenMP任务调度策略的评估。见 *OpenMP in a New Era of Parallelism: 4th International Workshop, IWOMP 2008 West Lafayette, IN, USA May 12-14, 2008 Proceedings 4*，页码 – 。施普林格，2008年。 100 110 ,

[50] Ansgar Schafer、Reinhart Horn、Hans 和 Ahlrichs。锂至氪原子的全优化收缩高斯基组。 *The Journal of Chemical Physics*，97(4):2571–2577，1992年。[51] Markus Reiher、Nathan Wiebe、Krysta M. Svore、Dave Wecker 和 Matthias Troyer。在量子计算机上阐明反应机制。 *Proceedings of the National Academy of Sciences* 114 29 7555 7560, 2017 , ( ): – 。

[52] Zhendong Li、Junhao Li、Nikesh S. Dattani、C. J. Umri-gar 和 Garnet Kin-Lic Chan。与量子模拟相关的固氮酶铁钼辅因子基态电子复杂性。 *The Journal of Chemical Physics* 150 2 024302, 2019 , ( ): 。

[53] Kai Guther、Robert J. Anderson、Nick S. Blunt、Niko-lay A. Bogdanov、Deidre Cleland、Nike Dattani、Werner Dobrautz、Khaldoon Ghanem、Peter Jeszenszki、Niklas Liebermann、Giovanni Li Manni、Alexander Y. Lozovoi、Hongjun Luo、Dongxia Ma、Florian Merz、Catherine Overy、Markus Rampp、Pradipta Kumar Samanta、Lauretta R. Schwarz、James J. Shepherd、Simon D. Smart、Eugenio Vitale、Oskar Weser、George H. Booth 和 Ali Alavi。NECI：N电子组态相互作用，重点介绍先进的随机方法。 *The Journal of Chemical Physics* 153 3 034107, 2020 , ( ): 。

[54] Gero Friesecke、Gergely Barcza 和 Legeza ¨Ors。通过限制性活性空间密度矩阵重整化群计算，以化学精度预测大型系统的全组态相互作用能。 *arXiv:2209.14190*，2022年。[55] Brian M. Hoffman、Dmitriy Lukoyanov、Zhi-Yong Yang、Dennis R. Dean 和 Lance C. Seefeldt。固氮酶固氮机制：下一阶段。*Chemical Reviews* 114 8 4041 4062, 2014 24467365 , ( ): – 。

PMID: 。[56] NVIDIA。英伟达A100张量核心GPU。 *https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datashee*

t-us-nvidia-1758950-r4-web.pdf，2021年。[57] A. Tichai、S. Knecht、A. T. Kruppa、 ¨O. Legeza、C. P. Moca、A. Schwenk、M. A. Werner 和 G. Zarand。将介质内相似性重整化群与密度矩阵重整化群相结合：壳层结构与信息熵，2022年。[58] Bo-Xiao Zheng、Chia-Min Chung、Philippe Corboz、Georg Ehlers、Ming-Pu Qin、Reinhard M. Noack、Hao Shi、Steven R. White、Shiwei Zhang 和 Garnet Kin-Lic Chan。二维Hubbard模型欠掺杂区域中的条纹序。 *Science*, 358(6367):1155–1160，2017年12月。[59] C. Krumnow、L. Veis、J. Eisert 和 ¨O. Legeza。通过模变换有效降维：利用矩阵乘积态模拟二维费米子凝聚态系统。 *Phys. Rev. B*，104 075137，2021年8月。[60] Ian P McCulloch 和 Mikl´os Gul´acsi。非阿贝尔密度矩阵重整化群算法。 *Europhysics Letters* 57 6 852, 2002 ，（）：。

[61] AI T´oth、CP Moca、 ¨O Legeza 和 G Zar´and。适用于非阿贝尔对称性的密度矩阵数值重整化群。 *Physical Review B*, 78(24):245109，2008年。[62] Sandeep Sharma 和 Garnet Kin-Lic Chan。用于量子化学的自旋适配密度矩阵重整化群算法。 *The Journal of Chemical Physics* 136 12 124121, 2012 ，（）：。

[63] Andreas Weichselbaum。张量网络中的非阿贝尔对称性：量子对称空间方法。 *Annals of Physics* 327 12 2972 3047, 2012 ，（）：–。

[64] Sebastian Keller 和 Markus Reiher。自旋适配的矩阵乘积态和算符。 *The Journal of Chemical Physics* 144 13 134101, 2016 ，（）：apr。[65] Klaas Gunst、Frank Verstraete 和 Dimitri Van Neck。具有 SU(2) 和分子点群对称性的三腿树张量网络。 *Journal of Chemical Theory and Computation* 15 5 2996 3007, 2019 ，（）：–apr。[66] Mikl´os Antal Werner、C˘at˘alin Pa¸scu Moca、Ors Legeza 和 Gergely Zar´and。SU(3) 哈伯德模型中的量子猝灭与电荷振荡：采用一般非阿贝尔对称性的时间演化块解耦方法检验。 *Physical Review B*，102(15):155108，2020年。[67] Gergely Barcza、Mikl´o s Antal Werner、Gergely Zar´and、¨Anton Pershin、Zsolt Benedek、Ors Legeza 和 Tibor Szilv´asi。受施密特分解启发的面向大规模限制性活性空间计算的方法。 *The Journal of Physical Chemistry A*，126(51):9709–9718，2022年12月。[68] 英特尔。Intel Xeon Gold 5318Y 2.10 GHz。 *https://ark.intel.com/content/www/us/en/ark/products/215271/intel-xeon-gold-5318y-processor-36m-cache-2-10-ghz.html* 2022 ，。