计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
62(6): 1347−1362, 2025

面向天河新一代超算系统的大规模精确对角化方法

李 彪[1,3] 刘 杰[1,2,3] 王庆林[1,2,3]

[1](高端装备数字化软件湖南省重点实验室(国防科技大学) 长沙 410073)

[2](并行与分布计算全国重点实验室(国防科技大学) 长沙 410073)

[3](国防科技大学计算机学院 长沙 410073)

(libiao1719@126.com)

Large-Scale Exact Diagonalization Methods for the New Generation of Tianhe Supercomputing Systems

Li Biao[1,3], Liu Jie[1,2,3], and Wang Qinglin[1,2,3]

[1] (*Laboratory of Digitizing Software for Frontier Equipment* (*National University of Defense Technology*), *Changsha* 410073)

[2] (*National Key Laboratory of Parallel and Distributed Computing* (*National University of Defense Technology*), *Changsha* 410073)

[3] (*College of Computer Science and Technology*, *National University of Defense Technology*, *Changsha* 410073)

Abstract The exact diagonalization (ED) method, a key numerical technique in quantum and condensed matter physics, directly computes the ground state of quantum systems. This study leverages Hamiltonian matrix symmetry, matrix-free methods, hierarchical communication models, and a data-level parallel algorithm optimized for the MT-3000 architecture. We propose a heterogeneous parallel algorithm for large-scale sparse Hamiltonian matrix-vector multiplication on the new generation Tianhe supercomputing system, enabling large-scale exact diagonalization for the one-dimensional Hubbard model. Tests on the Tianhe new generation system show a strong scaling efficiency of 55.27% when scaling from 256 to 8 192 processes for a 140-billion-dimension matrix. Weak scaling efficiency remains above 51.25% when scaling from 64 to 13 740 processes for a 730-billion-dimension matrix.

Key words exact diagonalization; Hubbard model; heterogeneous parallel computing; MT-3000 processor; Quantum many-body system

摘 要 精确对角化(exact diagonalization)方法是一种在量子物理、凝聚态物理等领域广泛应用的数值计算方法,是最直接求得量子系统基态的数值方法.仅从哈密顿矩阵的对称性出发,利用无矩阵(matrix-free)方法、分层通信模型以及适配于MT-3000的数据级并行算法,提出了面向天河新一代超算系统上的超大稀疏哈密顿矩阵向量乘异构并行算法,可以实现基于一维Hubbard模型的大规模精确对角化.提出的并行算法在天河新一代超算系统上进行了测试,其中在1 400亿维度矩阵规模上,8 192进程相比256进程强扩展效率为55.27%,而弱扩展到7 300亿维度矩阵规模上,13 740个进程相比64进程的弱扩展效率保持在51.25%以上.

关键词 精确对角化;Hubbard模型;异构并行计算;MT-3000处理器;量子多体系统

中图法分类号 TP311.13; TP309

DOI: 10.7544/issn1000-1239.202550150 **CSTR:** 32373.14.issn1000-1239.202550150

收稿日期: 2025-02-28; **修回日期:** 2025-04-14

基金项目: 国家重点研发计划项目(2023YFA1011704, 2021YFBO300101)

This work was supported by the National Key Research and Development Program of China (2023YFA1011704, 2021YFBO300101).

通信作者: 王庆林(wangqinglin_thu@163.com)

量子力学中最有趣的研究课题之一是研究量子多体系统的行为. 在这样的量子系统中，物理学家最有兴趣的是找出这个量子系统的基态，也就是能量本征值最小的态.量子多体问题涉及系统内多个粒子之间的相互作用，计算过程极其复杂，通常难以通过解析方法精确描述这些微观系统，物理学家们都转而利用数值方法求解量子多体系统. 量子多体理论的数值方法发展到现在，相关的工作非常多，例如量子蒙特卡洛方法[1-4]、密度矩阵重整化群方法[5-8]、精确对角化(exact diagonalization)方法[9-11]等. 相比前面2种数值方法，精确对角化方法是一种最直接求得基态的数值方法，它的核心思想是将量子多体系统的哈密顿算符表示成哈密顿矩阵，紧接着利用计算机计算实现哈密顿矩阵的对角化. 该方法通常用于确定多体系统模型在有限格点上的少数几个最低本征值和本征向量. 通过这些本征值和本征向量，可以轻松计算各种基态期望值和关联函数. 尽管这种方法对于计算小格点的多体系统非常精确，但在早期的研究中，受限于计算资源，无法研究具有较多格点数的系统，研究对象的系统规模通常小于16个格点. 进入21世纪后，随着芯片技术的飞速发展，量子多体系统的数值模拟规模也相应的上升，可研究具有特定性质的更多格点的量子多体模型，例如，36个格点的一维自旋−1/2的环、40个格点的二维自旋−1/2的平方晶格、32个格点的二维Hubbard模型的平方晶格等.

虽然，目前可求解36格点甚至40格点的具有特殊性质的量子多体模型，但是对一般性质的32格点甚至36格点的量子多体模型并没有一个很好的解决方案. 本文将沿着精确对角化方法这条路，针对具有一般性质的Hubbard模型，提出一种面向天河新一代超算系统的量子多体系统的大规模并行算法. 本文的主要贡献有3点：

1) 仅从哈密顿矩阵的对称性出发，利用无矩阵(Matrix-free)方法和分层通信模型，在天河新一代超算系统上实现了一种新的基于Hubbard模型的超大稀疏哈密顿矩阵向量乘异构并行算法.

2) 提出了一种快速查找哈密顿矩阵非零元的算法，该算法可在时间复杂度为$O(1)$的计算下快速确定哈密顿矩阵非零元的索引号.

3) 提出了一种面向天河新一代超算系统MT-3000体系架构的局部矩阵向量乘的向量化算法以提高计算效率.

1 相关工作

精确对角化方法被广泛应用于量子多体系统的求解，其核心思路是通过哈密顿矩阵的对角化来确定量子哈密顿算符的本征态和能量本征值. 精确对角化方法通常利用Lanczos算法[12]和Davidson算法[13]来实现基态求解，该方法已经成为研究量子多体系统物理性质的重要工具，其研究范围包括凝聚态物理中强相关的晶格模型的化学性质[14]、强相关材料的谐振X射线光谱[15]、核结构[16-17]和量子场论[18]等. 精确对角化方法广泛应用于量子多体模型的研究，相关模型包括Hubbard模型、t-J模型、Heisenberg模型以及SYK模型等. 已有研究利用精确对角化方法对上述模型在一维[19-23]和二维[24-27]体系中的物理性质展开了深入探讨，揭示了不同维度下这些模型的量子行为及相关现象。

精确对角化方法具有收敛速度快、结果精确的优势，因而广泛用于求解量子多体系统的基态问题. 但该方法的缺点也非常明显，即哈密顿矩阵的维数随系统规模呈指数增长.在早期的研究中，受限于当时的计算机内存，精确对角化方法主要是应用于格点数较少的量子多体系统，得到的数值计算结果为现代凝聚态物理中的许多问题提供了有价值的见解，例如，高温超导性[28]、磁力性质[29]以及量子霍尔效应[30]等. 许多的文献对精确对角化算法进行了优化与改进，文献[31]利用对称性来减少哈密顿矩阵的维数，并且提出动态计算Lanczos算法中的矩阵非零元的策略避免了存储矩阵，只需存储少量几个向量，极大减少了存储. 但是随着格点数增加，Lanczos算法中的向量也超出了单核计算机的存储，从而需要引入分布式并行计算. 进入21世纪后，随着计算机芯片飞速发展，超级计算机的引入使得研究者能够通过采取分布式存储的方式实现更大尺寸的量子多体模型的精确对角化. 2005年，Yamada等人[32]为了研究光学晶格上的性质，在地球模拟器上实现了1 590亿维度的Fermion-Hubbard模型的精确对角化. 该工作分别研究了Lanczos算法和预条件共轭梯度(PCG)算法求解哈密顿矩阵的最小特征值，在大规模情况下PCG算法的性能是传统Lanczos算法的1.5倍，这归因于PCG算法可以更有效地隐藏通信开销. 文献[32]同时得出，相比PCG算法实现需要保存多组向量(包括残量向量、搜索方向向量以及预条件向量)，Lanczos

算法在只需保存当前和前一时刻迭代向量，内存占用更低，因而能够进一步扩展到极高维度的求解规模. 2006 年，Yamada 等人[33]在地球模拟器上通过 MPI 实现了并行的 Lanczos 算法，可以成功计算 25 格点的量子多体系统的哈密顿矩阵的最小特征值，为精确对角化算法的大规模求解做出了进一步贡献. 2018 年，Wietek 等人[34]基于子晶格编码算法提出了一种大规模分布式并行 Lanczos 算法实现了可达 50 个自旋−1/2 粒子的量子多体系统的精确对角化.

近些年，随着协处理器的快速发展，研究者也将精确对角化算法移植到 GPU 和 MIC 等协处理器上，并取得不错的效果. 2012 年 Siro 等人[35]将精确对角化方法移植到 GPU 上，该算法采用压缩稀疏行（CSR）存储格式节省了存储哈密顿矩阵所需要的内存，采用了 CUDA 编程接口实现了 Lanczos 算法中的矩阵向量乘法；在 Tesla M2070 GPU 和 Intel Xeon X5650 两个平台上比较了算法的性能，在一维 Hubbard 模型测试中，相比使用单核心 CPU，GPU 的单精度和双精度性能分别实现了最高 100 和 60 的加速比；在二维 Hubbard 模型测试中，相比使用单核心 CPU，GPU 的单精度和双精度性能分别实现了最高 100 和 70 的加速比. 2016 年，Siro 等人[36]将精确对角化方法移植到 Intel Xeon Phi 协处理器上，并同时与 Intel Xeon CPU 处理器和 NVIDIA GPU 进行了比较. 在 16 个格点的 Hubbard 平方晶格模型（8 个上旋电子，8 个下旋电子）测试中，当电子数增加到 16 个时，GPU 性能是最高，相比 CPU 的加速比为 7.6，Intel Xeon Phi 协处理器次之，相比 CPU 的加速比为 2.5.

虽然已经有很多作者在不同处理器架构上做了很多工作，但是随着计算能力更强、体系架构更新的超算系统出现，重新设计适合新型架构的大规模精确对角化并行算法，实现适用于更大规模量子多体系统的通用精确对角化求解具有重要意义. 本文将面相天河新一代超算系统，针对 Hubbard 模型，实现基于 Lanczos 算法的大规模精确对角化.

2 精确对角化方法

2.1 Hubbard 模型

Hubbard 模型是最经典的一个量子多体系统，其最初是由物理学家 John Hubbard 在 1963 年提出，用于描述电子在晶体中的相互作用[37−39]. 这个模型提供了最简单的方法来理解电子之间的相互作用，即如何在固体中产生绝缘、磁性甚至是新的超导效应. Hubbard 模型的哈密顿量可以表示为：

$$\boldsymbol{H}=\boldsymbol{H}_{\mathrm{hop}}+\boldsymbol{H}_{\mathrm{int}}=-t\sum_{i,j,\sigma}\left(c_{j,\sigma}^{+}c_{i,\sigma}+h.c\right)+U\sum_{i}n_{i,\uparrow}n_{i,\downarrow}, \tag{1}$$

$\langle i,j\rangle$ 表示相邻格点的集合，$c_{i,\sigma}^{+}$和 $c_{i,\sigma}$ 代表在格点 i 的电子产生和湮灭，σ 代表电子旋转的方向（上旋或下旋），$n=c_{i,\sigma}^{+}c_{i,\sigma}$. 在一个格点中存在 4 种可能的电子占据方式：无电子占据、上旋电子独占、下旋电子独占、双电子占据. 哈密顿量 $\boldsymbol{H}_{\mathrm{hop}}$ 代表动能，描述电子在晶格中的运动，通常表示为电子的跃迁（跳跃）操作，表示电子从一个晶格点移动到另一个晶格点. $\boldsymbol{H}_{\mathrm{int}}$代表格点上的相互作用能，描述电子之间的相互排斥作用，通常是对同一格点上 2 个电子的相互作用进行建模.

2.2 哈密顿矩阵

对 Hubbard 模型进行精确对角化求解，也就是对该模型的哈密顿量$\boldsymbol{H}$进行精确对角化. 因此，这就需要找到一组正交基，求得哈密顿量$\boldsymbol{H}$在此正交基下的矩阵，通过求此矩阵的对角化就可实现 Hubbard 模型的精确对角化. 在给定的一组正交基下，哈密顿量 $\boldsymbol{H}$ 与其在该组正交基下的矩阵是等价的，为了阐述方便，后续将对哈密顿量与其所对应的矩阵不做区分，统一用 $\boldsymbol{H}$ 表示. 考虑N个格点、M个电子的 Hubbard 模型，通常用该M个电子在此N个格点上的所有可能分布状态表示一组正交基. 为了描述简便，采用二进制数来表示每个状态，式（2）描述了 8 个格点、2 个自旋向上电子以及 3 个自旋向下的电子量子系统其中的一个分布状态.

$$\underbrace{|00010010\rangle}_{\text{up}}\underbrace{|00101001\rangle}_{\text{down}}. \tag{2}$$

该分布（也称为基态）包含 2 个部分，左边表示自旋向上（up）的电子在格点的分布，右边表示自旋向下（down）的电子在格点的分布. 如果某个格点被电子占用，就给此格点位置赋值为 1，如果此格点没被电子占有，就给它赋值为 0. 式（2）中基态，上旋电子驻留在 2 和 5 号格点，而下旋电子驻留在 1、4 和 6 号格点. 给定具有N个格点的 Hubbard 系统，该系统包含$N_{\uparrow}$个自旋向上的电子和$N_{\downarrow}$个自旋向下的电子，由泡利排除原理，禁止 2 个或多个相同自旋的电子占据同一格点，因此，哈密顿矩阵的维数可由下式表示：

$$\dim\boldsymbol{H}=\binom{N}{N_{\uparrow}}\binom{N}{N_{\downarrow}}. \tag{3}$$

由式（3）可以得出，哈密顿矩阵的维数呈指数级增长. 例如，在半填充的情况下，即$N_{\uparrow}=N_{\downarrow}=N/2$，对

于 10 个格点情形, $\dim \boldsymbol{H} = 63\,504$; 对于 16 个格点情形, $\dim \boldsymbol{H} \approx 1.7\times 10^8$; 对于 24 个格点情形, $\dim \boldsymbol{H} \approx 7.3\times 10^{12}$. 虽然这些矩阵维数巨大, 但每行中的非零元个数却极为有限, 因此哈密顿矩阵为超大规模稀疏矩阵.

由线性代数知识可知, 想要唯一确定哈密顿量 $\boldsymbol{H}$ 在一组正交基上的线性映射矩阵, 需对这组基进行排序, 即对该量子多体系系统的所有基态进行排序. 一个简单的方法是根据代表状态的二进制数的大小来对基态进行排序[31], 考虑更一般的基态表示:

$$|n_\uparrow(N)n_\uparrow(2)\cdots n_\uparrow(1)\rangle|n_\downarrow(N)n_\downarrow(2)\cdots n_\downarrow(1)\rangle, \tag{4}$$

其中, $n_\uparrow(i)$和$n_\downarrow(i)$分别表示格点i处, 自旋向上和向下的电子个数, 取值只能为 0 或 1. 为了对基态进行排序, 把式(4)中的形式转化为二进制数值, 对应方式如下:

$$|n_\uparrow(N)n_\uparrow(2)\cdots n_\uparrow(1)\rangle \leftrightarrow \sum_{i=1}^{N} n_\uparrow(i)2^{i-1} = I_\uparrow, \tag{5}$$

$$|n_\downarrow(N)n_\downarrow(2)\cdots n_\downarrow(1)\rangle \leftrightarrow \sum_{i=1}^{N} n_\downarrow(i)2^{i-1} = I_\downarrow. \tag{6}$$

由于$I_\uparrow$和$I_\downarrow$都是整数, 有大小关系, 因此可分别针对电子自旋向上或向下的所有分布进行排序. 记$J_\uparrow$和$J_\downarrow$分别表示$I_\uparrow$和$I_\downarrow$各自排序中的序号, 则式(4)中基态对应的索引号J可由如下式(7)表示.

$$J = J_\uparrow \binom{N}{N_\downarrow} + J_\downarrow. \tag{7}$$

基于式(7)可构造 $N=3$、$N_\uparrow=2$和$N_\downarrow=2$模型的正交基所有排序的对应关系, 具体如表 1 所示. 有了上述准备工作, 下面阐述如何形成哈密顿矩阵, 也就是说, 如何确定哈密顿矩阵的所有非零元素. 式(1)中的$\boldsymbol{H}_{\text{int}}$所形成的矩阵仅在对角线上存在非零元, 后续涉及到的计算均在本地, 不会跨进程通信, 因此在后续并行算法的讨论中可以忽略它. 本文重点关注如何在式(1)中找到矩阵$\boldsymbol{H}_{\text{hop}}$的非零元. 对于给定的行索引$J$, 从上面的讨论中, 存在唯一的基态$(I_\uparrow, I_\downarrow)$与它对应, 如果存在$J'$, 使得$(\boldsymbol{H}_{\text{hop}})_{JJ'}$不等于 0, 则索引$J'$对应的基态应满足等于$(I_{\uparrow k}, I_\downarrow)$或$(I_\uparrow, I_{\downarrow k})$, 其中$I_{\uparrow k}(I_{\downarrow k})$表示如果$I_\uparrow(I_\downarrow)$的第$k$和$(k+1)$位等于“10”或“01”, 则$I_{\uparrow k}(I_{\downarrow k})$等于$I_\uparrow(I_\downarrow)$将此 2 个位置的值更改为“01”或“10”. 若格点数为N, 则k可以从 1 遍历到N, 对于特殊情况$k=N$时, $k+1$对应的格点位置为 1, 于是可知矩阵$\boldsymbol{H}_{\text{hop}}$的每一行的非零元个数不超过 $2N$. 这表明当规模较大时, 哈密顿矩阵其实是一个超大规模的稀疏矩阵.

Table 1 Basis States Order Relation for N=3, $N_\uparrow$=2, $N_\downarrow$=2

表 1 N=3, $N_\uparrow$=2, $N_\downarrow$=2 基态序关系

$I_\uparrow$	$I_\downarrow$	$J_\uparrow$	$J_\downarrow$	J
011	011	0	0	0
011	101	0	1	1
011	110	0	2	2
101	011	1	0	3
101	101	1	1	4
101	110	1	2	5
110	011	2	0	6
110	101	2	1	7
110	110	2	2	8

2.3 Lanczos 算法

由于哈密顿矩阵的维数随着格点和电子数目的增加呈指数级增长, 因此对于大规模情形, 完全对角化哈密顿是不可能做到. 幸运的是物理学家们通常只对基态和低级激发态感兴趣, 因此求解的核心在于逼近哈密顿矩阵最小特征值及其关联态, 而 Lanczos[40-41] 算法是一个众所周知的能快速且准确逼近稀疏矩阵最小特征值的方法. Lanczos 算法的核心思想是将哈密顿矩阵投影到由 Krylov 子空间生成的一组正交基上, 定义为:

$$\mathcal{K}_m(\boldsymbol{b};\boldsymbol{H}) = span\left\{\boldsymbol{b}, \boldsymbol{H}\boldsymbol{b}, \boldsymbol{H}^2\boldsymbol{b}, \cdots, \boldsymbol{H}^{m-1}\boldsymbol{b}\right\}, \tag{8}$$

其中$\boldsymbol{b}$是一个随机起始向量, m是 Krylov 子空间的维数. Lanczos 算法的处理过程将产生出一个$m\times m$的三对角矩阵. 随着m的增加, 该矩阵的最小(最大)特征值越来越精确地接近$\boldsymbol{H}$的最小(最大)特征值. Lanczos 算法收敛速度很快, 即使当哈密顿矩阵非常大, $m\approx 100$时也能保证足够快的收敛性[42]. Lanczos 算法的具体执行过程如算法 1 所示.

算法 1. Lanczos 算法.

① $\boldsymbol{q}(1:n) \leftarrow 0$, $\beta_0 \leftarrow 1$, $j \leftarrow 0$;
② while $\beta_j \neq 0$ do
③ if $j \neq 0$
④ for $i = 0$ to n do
⑤ $t \leftarrow b_i$, $b_i \leftarrow q_i/\beta_j$, $q_i \leftarrow -t\beta_j$;
⑥ end for
⑦ end if
⑧ $\boldsymbol{q} \leftarrow \boldsymbol{q} + mult(\boldsymbol{H}, \boldsymbol{b})$;
⑨ $j \leftarrow j+1$;
⑩ $\alpha_j \leftarrow \boldsymbol{b}^{\mathrm{T}}\boldsymbol{q}$, $\boldsymbol{q} \leftarrow \boldsymbol{q} - \alpha_j\boldsymbol{b}$, $\beta_j \leftarrow \sqrt{\boldsymbol{q}^{\mathrm{T}}\boldsymbol{q}}$;
⑪ end while

假设矩阵的维度为n, 向量$\boldsymbol{b}$被随机初始化为单

位向量，通过正交 Krylov 子空间的基$\mathcal{K}_m(\boldsymbol{b};\boldsymbol{H})$，生成一组所谓的 Lanczos 正交基$(q_1,q_2,\cdots,q_m)$，$\boldsymbol{H}$在这组正交基中的投影矩阵为：

$$\boldsymbol{T}=\begin{pmatrix}\alpha_1 & \beta_1 & 0 & \cdots & 0\\ \beta_1 & \alpha_2 & \beta_2 & \ddots & \vdots\\ 0 & \cdots & \cdots & \cdots & 0\\ \vdots & \ddots & \beta_{m-2} & \alpha_{m-1} & \beta_{m-1}\\ 0 & \cdots & 0 & \beta_{m-1} & \alpha_m\end{pmatrix}.$$

$\boldsymbol{T}$是一个三对角矩阵，此三对角矩阵由生成的常数α_j和β_j唯一确定. 由于$\boldsymbol{T}$是三对角矩阵，而且维数远小于$\boldsymbol{H}$的维数，它的特征值很容易用标准方法计算出来，从而通过求解小规模三对角矩阵的最小特征值来逼近哈密顿矩阵的最小特征值，从而可实现哈密顿矩阵的精确对角化.

3 大规模异构并行 Lanczos 算法

3.1 MT-3000 体系结构

天河新一代超算系统由 MT-3000 处理器组成，该处理器是由国防科技大学自主研制的异构众核处理器[43]. 如图 1 所示，MT-3000 包含 16 个通用 CPU、96 个加速域控制核心和 1 536 个面向计算的加速核心，在 1.2 GHz 下运行时可实现 11.6 Tflops 的双精度性能. MT-3000 采用多区域微架构，16 个 CPU 被分组到通用区，96 个控制核心和 1 536 个加速核心被分组到加速区，加速区被平均分为 4 个自主加速域. 每个加速域包含 24 个控制核心、384 个加速核心，以及片上共享全局存储器（GSM）、高带宽共享存储器（HBSM）和 DDR 存储器空间，并且每个加速域彼此独立执行.

在每个加速域中，24 个控制核心和 384 个加速核心被组织成 24 个加速阵列（acceleration array）. 每一个加速阵列包括 1 个控制核心和 16 个加速核心，其中 16 个加速核心由一个单一指令流驱动并同步工作，可支持 1 024 位的向量操作. MT-3000 引入了阵列存储器（AM），它是一个大小为 768 KB 的矢量存储器，支持加速阵列的向量化操作. 为了进一步提高整体效率，每个加速器核心都支持超长指令字（very long instruction word, VLIW），包括 3 个浮点乘加运算部件 MAC（multiply and accumulate units）、2 个向量 Load/Store 单元和 1 个整数执行部件（integer execution unit, IEU），通过控制核心可以使得同一个加速阵列中 16 个加速核心在 1 个时钟周期内同时发射 6 条一样的指令，利用 VLIW 的特性可以实现指令级并行，使加速核成为强大的计算引擎. MT-3000 的多级存储之间支持 DMA（direct memory access）传输数据，DMA 可以实现各级存储空间（AM、GSM、HBSM、外部存储器 DDR）的快速访问.

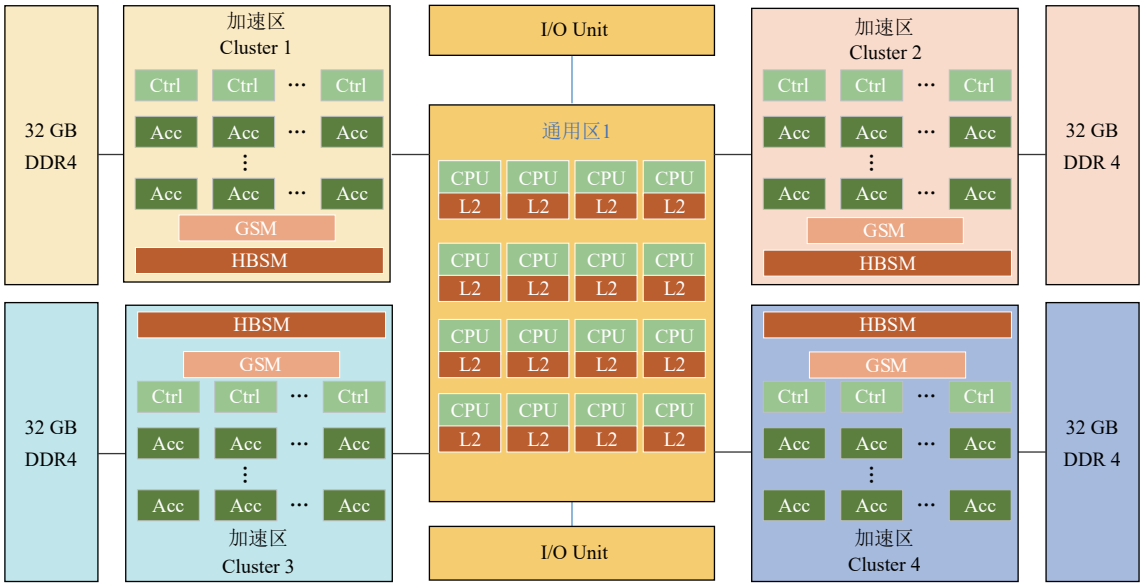


Fig. 1 Overall architecture of MT-3000

图 1 MT-3000 的整体架构

3.2 基于 matrix-free 方法的矩阵向量乘并行化

Lanczos 算法最耗时的部分是哈密顿矩阵向量乘法运算. 当系统规模较小时，哈密顿矩阵可以完整地存储到内存中，采用通用的矩阵向量乘法进行计算. 然而，如 2.2 节所述，当格点数量超过 16 时，哈密顿矩阵的维数成指数级增长，只能采取分布式的方式

存储矩阵. 但是, 随着格点数继续增大, 达到 32 甚至 36 时分布式方法同样也会面临哈密顿矩阵存储爆炸问题. 因此, 为了避免矩阵存储爆炸问题, 本文采取 Matrix-free 方法来解决在超大规模情形下哈密顿矩阵无法存储的问题. 该方法的核心思想是通过增加计算代价换取存储空间, 在有限的内存容量内, 能够计算更大的规模. 针对算法 1 中 Lanczos 算法, 本文仅分布式存储向量$\boldsymbol{q}$和$\boldsymbol{b}$, 而哈密顿矩阵$\boldsymbol{H}$的非零元可通过在本地进程中计算得到, 从而可以在有限资源下实现更大规模的量子多体模型的对角化求解.

在并行 Lanczos 算法设计中, 矩阵非零元的分布特点对通信的复杂度有重要影响, 因此不能简单按照矩阵$\boldsymbol{H}$和向量$\boldsymbol{b},\boldsymbol{q}$的行均匀分配存储. 本文针对哈密顿矩阵非零元的生成特点提出了一种新的矩阵向量并行划分方式, 如图 2 所示, 将一个固定的自旋向上分布与所有自旋向下分布组成一个行索引块, 以这种行索引块为基本单位, 均匀划分哈密顿和向量. 假设总进程数为P, 进程号分别为$0,1,\cdots,P-1$. 图 2 的并行划分方式可简化为:

$$\boldsymbol{H}=\begin{pmatrix}H^{(0)}\\H^{(1)}\\\vdots\\H^{(P-1)}\end{pmatrix},\boldsymbol{b}=\begin{pmatrix}b^{(0)}\\b^{(1)}\\\vdots\\b^{(P-1)}\end{pmatrix},\boldsymbol{q}=\begin{pmatrix}q^{(0)}\\q^{(1)}\\\vdots\\q^{(P-1)}\end{pmatrix},\tag{9}$$

其中$H^{(i)}$, $b^{(i)}$和 $q^{(i)}(i=0,1,\cdots,P-1)$ 分别表示矩阵和向量在不同进程的分块. 特别需要注意, 把矩阵并行划分到相应进程只是形式上的, 事实上并不存储矩阵, 只是为了方便指出矩阵块的非零元素的计算是在局部进程中执行的.

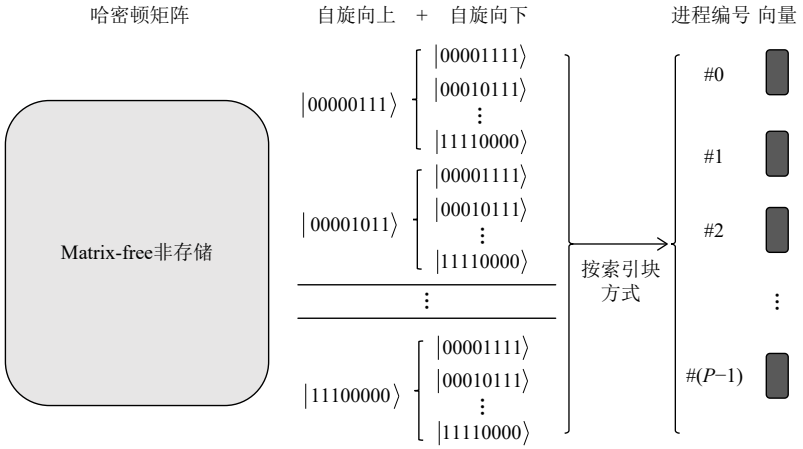


Fig. 2 Parallelization of matrix-vector multiplication

图 2 矩阵向量乘并行化

下面提出简化版并行 Lanczos 算法的伪代码, 具体过程如算法 2 所示.

算法 2. 基于 Hubbard 模型的简化版并行 Lanczos 算法.

/*N表示格点数, p表示当前本地进程的标识号*/

① for 遍历$\boldsymbol{H}^{(p)}$行i do /*对角线元素计算*/

② $q_i^{(p)}\leftarrow q_i^{(p)}+H_{ii}^{(p)}b_i$;

③ end for

④ for $k=1$ to N do /*非对角线元素计算*/

⑤ for 遍历$\boldsymbol{H}^{(p)}$行 i do

/* 电子自旋向上的方向*/

⑥ if $k,k+1$处满足“10”或“01”then

⑦ 计算j_k使得$H_{ij_k}\neq 0$;

⑧ $q_i^{(p)}\leftarrow q_i^{(p)}+H_{ij_k}b_{j_k}$;

⑨ end if

⑩ end for

⑪ for 遍历$\boldsymbol{H}^{(p)}$行 i do

/*电子自旋向下的方向*/

⑫ if $k,k+1$处满足“10”或“01”then

⑬ 计算j_k使得$H_{ij_k}\neq 0$;

⑭ $q_i^{(p)}\leftarrow q_i^{(p)}+H_{ij_k}b_{j_k}$;

⑮ end if

⑯ end for

⑰ end for

算法 2 描述了本地进行矩阵向量乘的计算过程, 该过程可分为 2 个部分, 即矩阵对角线元素计算和非对角线元素计算, 由于对角线元素的计算不涉及进程间通信以及矩阵非零元索引计算, 后续不做详细讨论. 对于非对角线上的元素计算, 首先, 外部循环对格点位置从 1 到N进行遍历; 然后, 对于每个固定格点位置k, 内部循环遍历本地矩阵的所有行, 并对每一行i计算出满足$H_{ij_k}\neq 0$的列 j_k; 接着计算矩阵

向量乘. 由图 2 的并行划分可知，对于自旋向下的电子计算情形，计算 j_k 时仅需修改 i 的二进制表示 $(I_\uparrow, I_\downarrow)$ 的 $I_\downarrow$ 部分，这表明 j_k 与 i 处在同一个行索引数据块，意味着 j_k 与 i 处在同一个进程，因此自旋向下的电子计算并不涉及到通信. 对于自旋向上的电子计算情形，计算 j_k 时需修改 i 的二进制表示 $(I_\uparrow, I_\downarrow)$ 的 $I_\uparrow$ 部分，这意味着 j_k 与 i 处在不同的行索引数据块，因此可能存在通信. 综合来看基于图 2 的并行划分方式，仅在自旋向下的电子计算存在通信，其他的对角线计算和自旋向下的电子计算均不涉及通信.

下面分析算法 2 的计算和通信负载均衡问题. 首先考虑计算，由于并行算法是按照行索引块均匀划分的，因此每个进程对应的矩阵行数量相同，进而可知对角线上的计算负载是均衡的. 在计算自旋向下的情形时，哈密顿矩阵非零元的计算仅与自旋向下的分布相关，由于行索引块的定义是包含所有自旋向下的分布，因此，对于任何固定格点位置 k，任何一个索引块中的所有行的非零元计算完全相同，这表明任何一个索引块中对应的非零元个数相同，因此自旋向下情形计算负载也是均衡的. 在计算自旋向上情形时，对于任何固定格点位置 k，仅当 k 和 $k+1$ 格点位置处满足“10”或“01”时，该索引块中的行才可以计算出矩阵的非零元，因此可以计算得出存在非零元的索引块数量与所有的索引块数量的比值为：

$$\frac{2\times\binom{N-2}{N_\uparrow-1}}{\binom{N}{N_\uparrow}}=\frac{2N_\uparrow(N-N_\uparrow)}{N(N-1)}. \tag{10}$$

假设 $N=32$、$N_\uparrow=5$，则式(10)的比值约等于 0.27，也就说明不到 30% 的索引块存在非零元，这表明自旋向上计算负载并不均衡.

至于通信负载均衡问题，由于对角线和自旋向下情形计算不涉及通信，因此仅考虑自旋向上计算情形. 由于通信仅在存在矩阵非零元的情形时才会发生，因此通信负载与自旋向上计算情形的分析是一致的，通信负载量与式(10)高度相关联，唯一的区别就是，2 个具备非零元的相邻的索引块有可能是存储在一个进程中. 综合来看通信负载也存在不均衡.

本文的目的是提供一种求解 32, 36 格点以上超大规模的量子多体模型精确对角化方法，而内存爆炸问题是制约规模扩大的最关键因素. 因此，为了解决此问题，算法 2 是仅在存储必要的向量 $\boldsymbol{b},\boldsymbol{q}$ 的情形下，通过遍历格点位置的方式实现了“计算换存储”的并行 Lanczos 算法. 否则，为了实现计算负载均衡，在自旋向上情形下，每一行均需要提前存储非零元所在的列，而非通过遍历计算获得. 在这种情形下，一行的非零元个数不超过 $2N_\uparrow$，如果考虑整型数据存储的话，非零元列索引所需要的内存接近 $N_\uparrow$ 个向量 $\boldsymbol{b},\boldsymbol{q}$ 的内存，这对于大规模求解显然是不合理的，也不可能做到.

随着系统规模的增大，通过增加计算时间来换取存储空间的方法虽然节省了矩阵的存储，但也导致每次内部迭代都需要重复计算哈密顿矩阵的非零元，这将导致计算时间的显著增加，此时算法 2 面临 3 个需要克服的问题：

1)如何快速计算矩阵非零元对应的索引 j_k.

2)b_{j_k} 可能不存储在本地进程中，需要实现不同进程之间的通信，如何高效合理地设计通信模型.

3)如何将矩阵向量乘计算移植到 MT-3000 加速簇上，实现高效的数据并行计算.

本节的后续部分将重点讨论这 3 个问题.

3.3 非零元索引快速求解算法

为了快速确定矩阵 $\boldsymbol{H}$ 的非零元，关键在于如何高效计算矩阵行或列索引与其二进制表示之间的一一对应关系：

$$J\leftrightarrow(I_\uparrow,I_\downarrow). \tag{11}$$

由式(7)可知 J 可以由 $J_\uparrow$ 和 $J_\downarrow$ 唯一确定，因此只需要考虑如何分别快速确认 $I_\uparrow$ 与 $J_\uparrow$ 以及 $I_\downarrow$ 与 $J_\downarrow$ 之间的一一对应关系. 此外，由式(5)(6)可以看出 $I_\uparrow$ 与基态二进制数表示的对应规则和 $I_\downarrow$ 与基态二进制数表示的对应规则完全相同. 为了阐述简便，后续只讨论 $I_\uparrow$ 和 $J_\uparrow$ 之间的一一对应关系. 当格点数目很小的时候，可以用表 1 列出 $I_\uparrow$ 和 $J_\uparrow$ 的一一对应关系，但是如果格点数目增加，列表方法就不再适用，此时需要用算法来刻画它们之间的对应关系. 由于基态和其二进制数表示法是一一对应的，所以可以用二进制数表示法来代替基态.

考虑格点数为 N、自旋向上电子数为 k 的情形. 对 k 个电子在 N 个格点上的所有分布进行排序，并用 $J_\uparrow$ 来表示 $I_\uparrow$ 在序列中的序号. 由式(5)可知，最小的配置就是把 k 个电子全部放在二进制表示的低位，如下表示：

$$00\cdots0\underbrace{1\cdots11}_{k}. \tag{12}$$

式(12)中处在右侧的 k 个位置都被“1”占据，剩余的左侧格点被“0”占据，其二进制数值 $I_\uparrow$ 是所有二进制表示中数值最小的，把它对应的 $J_\uparrow$ 取值为 0. 将

式(12)中的“1”向高位移动,则可增大 $I_{\uparrow}$ 的值,考虑最左侧中第k处的“1”,它只能从格点k处移动到N处,移动最左侧的“1”可以得到以下$N-k+1$个二进制形式:

$$\begin{array}{l} 1*****\cdots* \\ 01*****\cdots* \\ \vdots \quad \ddots \quad \cdot \\ 0\cdots 0\underbrace{1*\cdots**}_{k} \end{array} \tag{13}$$

余下的$k-1$个电子任意分布在标记为“*”的位置,所以“*”的值为0或1.不管“*”的值是0还是1,式(13)中的 $N-k+1$组二进制表示的值从上到下依次递减.接下来,考虑式(11)中第 $k-1$ 处的“1”,由于它向高位移动不能跨越第k处的“1”,所以它只能从格点$k-1$处移动到$N-1$处,而且所有可能的二进制表示只能出现在式(13)中的$N-k+1$组形式中.由于式(13)中二进制表示的大小是依次递减,所以组间的二进制表示的排序关系是确定的,因此只需考虑组内二进制数的排序关系.对于式(13)中任意一个组二进制表示,假设式(12)中第 k 处的“1”出现在该组二进制表示的格点i_1处 $(k\leqslant i_1\leqslant N)$,则$k-1$的“1”只能出现在格点$i_2$处,$i_2$的取值只能从$k-1$到$i_1-1$,于是这个固定的组又可以被细分为 i_1-k+1 个子组,排列方式与式(13)相同,子组之间的二进制数大小确定,可以被排序.采用递归的方法处理式(11)中余下的“1”,最终可得到k个电子分布在N个格点的所有二进制表示的排序.将$J_{\uparrow}$到$I_{\uparrow}$的对应关系存储到数组$hash_{k_{\uparrow}}$中,同理也可以得到$J_{\downarrow}$到$I_{\downarrow}$的对应数组$hash_{k_{\downarrow}}$,特殊情况下当自旋向上和自旋向下电子数相同时,这2个对应关系完全相同,仅需1个数组就可以存储此对应关系.

因此,对于任意给定的J,通过数组$hash_{k_{\uparrow}}$和$hash_{k_{\downarrow}}$可以快速确定$(I_{\uparrow},I_{\downarrow})$.如果存在$J'$使得 $H_{JJ'}\neq 0$,由2.2节的讨论中得知J'对应的二进制表示应该等于$(I_{\uparrow k},I_{\downarrow})$或$(I_{\uparrow},I_{\downarrow k})$.为了便于描述,下面给出了从$J$到$J'$的计算步骤流程:

$$J\rightarrow(I_{\uparrow},I_{\downarrow})\rightarrow\frac{(I_{\uparrow k},I_{\downarrow})}{(I_{\uparrow},I_{\downarrow k})}\rightarrow J'. \tag{14}$$

如果$I_{\uparrow}(I_{\downarrow})$的第$k$和第$k+1$位等于“10”或“01”,则把第$k$和第$k+1$位的二进制位互换得到 $I_{\uparrow k}(I_{\downarrow k})$.该过程的计算需要引入掩码 $mask=2^k+2^{(k+1)\%N}$,其第k和第 $(k+1)\%N$位等于1,其他的二进制位都为0.为了找到$I_{\uparrow k}$,首先使$I_{\uparrow}$和 $mask$ 执行位与操作运算,并将结果记为K,然后用$I_{\uparrow}$和K执行位异或操作,并将结果记为L,最后得到:

$$I_{\uparrow k}=I_{\uparrow}-K+L. \tag{15}$$

利用此方法可由$I_{\downarrow}$计算得到$I_{\downarrow k}$.接下来是如何从$(I_{\uparrow k},I_{\downarrow})$或$(I_{\uparrow},I_{\downarrow k})$得到 J',也就是如何从$I_{\uparrow}(I_{\downarrow})$快速计算$J_{\uparrow}(J_{\downarrow})$.利用数组$hash_{k_{\uparrow}}(hash_{k_{\downarrow}})$以及二分查找从$I_{\uparrow(\downarrow)}$快速获得$J_{\uparrow(\downarrow)}$,并将将$I_{\uparrow(\downarrow)}$到$J_{\uparrow(\downarrow)}$的映射关系保存到另一个数组$hash_{r_{\uparrow}}(hash_{r_{\downarrow}})$中,然后利用式(7)快速获得$J'$的值.$hash_{k_{\uparrow}}$, $hash_{k_{\downarrow}}$, $hash_{r_{\uparrow}}$以及 $hash_{r_{\downarrow}}$这4个数组可在迭代进行前计算获得,后续的Lanczos迭代过程中仅需访问数组,而不需要重复计算,再加上式(7)(15)的计算步骤是常量,因此计算矩阵非零元(J,J')的时间复杂度是固定的常量,也就是复杂度为$O(1)$.

下面针对大规模测试中,4个 $hash$ 数组所占内存大小进行量化分析.分析图2的并行划分,进程间是按照索引块均匀划分,假设总进程数为P,则平均到1个进程的索引块个数为$\binom{N}{N_{\uparrow}}/P$,该值也等于一个进程内的自旋向上的分布个数,由于任意一个行索引块均包含所有的自旋向下分布,因此一个进程内的自旋向下分布数等于$\binom{N}{N_{\downarrow}}$.假设考虑$N=32(N_{\uparrow}=5,N_{\downarrow}=5)$,显然进程总数 P会起到决定性作用,当P超过100时,在一个进程中自旋向下的分布总数是自旋向上分布总数的2个数量级以上,因此$hash_{k_{\uparrow}}$和$hash_{r_{\uparrow}}$分配到一个进程的存储远少于$hash_{k_{\downarrow}}$和$hash_{r_{\downarrow}}$,几乎可以忽略不计,下面讨论$hash_{k_{\downarrow}}$和$hash_{r_{\downarrow}}$所占内存.还是以$N=32(N_{\uparrow}=5,N_{\downarrow}=5)$为例,$\binom{N}{N_{\uparrow}}=\binom{32}{5}=201\,376$,将进程总数$P$=10 000,此时可知一个进程分到的索引块个数等于201 376/10 000,约等于20,这说明向量$\boldsymbol{b}$和$\boldsymbol{q}$在一个进程的长度等于20个所有自旋向下分布的长度,$hash_{k_{\downarrow}}$和$hash_{r_{\downarrow}}$加起来刚好等于2个所有自旋向下分布的长度,比较发现,$hash_{k_{\downarrow}}$和$hash_{r_{\downarrow}}$数组存储大小刚好是向量$\boldsymbol{b}$和$\boldsymbol{q}$的1/10,如果P<10 000,$hash_{k_{\downarrow}}$和$hash_{r_{\downarrow}}$所占内存与向量 $\boldsymbol{b}$ 或 $\boldsymbol{q}$ 的内存相比将超过1个数量级.因此在大规模计算时,每个进程都存储这4个数组是可行的.

3.4 分层通信模型

从3.2节的讨论可知,算法2仅在行⑩的矩阵向量乘,也就是电子自旋向上的情形下才可能涉及进程间的通信.考察算法2中的行⑩,若对于固定的行i计算得到$H_{ij_k}\neq 0$,当行i和j_k不在同一进程时,此时行⑩的计算涉及进程通信,行j_k所在的进程需将b_{j_k}的值发送给行i所在进程.然而,索引j_k是由行i所在进

程计算得出，b_{j_k}所在进程并没有j_k的信息，因此，当行i所在进程需要发起通信时，而存储b_{j_k}的进程并不知道该向哪个进程发送数据. 因此，数据接收方与发送方之间难以直接建立对等通信. 为了解决此难点，本文基于哈密顿矩阵的对称性，提出一种分层的通信模型，能够实现进程间的对等通信.

如图 3 所示，假设第 i 行处在进程p，存在 j_k 使得$H_{ij_k}\neq 0$，由哈密顿矩阵的对称性可知$H_{j_ki}\neq 0$. 假设b_{j_k}存储在进程$s(s\neq p)$中，则存在如下 2 个计算：

$$\begin{cases} q_i^{(p)} = q_i^{(p)} + H_{ij_k}^{(p)} b_{j_k} \\ q_{j_k}^{(s)} = q_{j_k}^{(s)} + H_{j_ki}^{(s)} b_i \end{cases}. \tag{16}$$

结合式(16)，从通信的角度来看，本地进程p需要从进程s接收数据b_{j_k}，而进程s需要从进程p接收数据b_i，因此只要$H_{ij_k}=H_{j_ki}\neq 0$时，数据交换成对发生. 基于此对称性性质，可以实现对等通信，下面详细讨论如何实现这个通信过程.

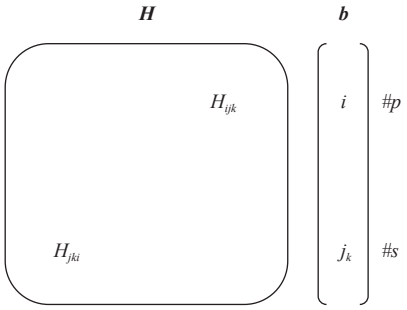


Fig. 3 Symmetry of the Hamiltonian matrix

图 3 哈密顿矩阵的对称性

由 3.2 节可知，并行算法以行索引块为基本单位均匀划分，假设本地分配到m个行索引块，当前遍历格点位置k处，这m个行索引块只有$s(s\leqslant m)$个索引块满足k，$k+1$处为“10”或“01”，不妨按照顺序依次记为$J_{\uparrow}^1, J_{\uparrow}^2, \cdots, J_{\uparrow}^s$. 由 3.3 节中的$hash_{\uparrow}$数组可以快速计算出该$s$个索引块对应的矩阵非零元的列索引块$J_{\uparrow k}^1, J_{\uparrow k}^2, \cdots, J_{\uparrow k}^s$. 由于并行划分是依据索引块的标号按照顺序均匀划分，因此可以由索引块$J_{\uparrow k}$值算出该索引块所在的进程号. 为了阐述方便，将$J_{\uparrow k}^1, J_{\uparrow k}^2, \cdots, J_{\uparrow k}^s$记为$J_{\uparrow k}^{(p_1)}$, $J_{\uparrow k}^{(p_2)}$, $\cdots$, $J_{\uparrow k}^{(p_s)}$，这里的上标p_1, $p_2, \cdots$, p_s分别表示其对应的索引块所在的进程标号. 由上述分析的通信对称性可知，当前进程需要的通信操作是：分别向p_1, $p_2, \cdots$, p_s进程发送$J_{\uparrow}^1$, $J_{\uparrow}^2$, $\cdots$, $J_{\uparrow}^s$索引块对应的向量$\boldsymbol{b}$的分块数据. 同时从p_1, $p_2, \cdots$, p_s进程接收$J_{\uparrow k}^{(p_1)}$, $J_{\uparrow k}^{(p_2)}$, $\cdots$, $J_{\uparrow k}^{(p_s)}$索引块对应的向量$\boldsymbol{b}$的分块数据. 整个通信过程实现了本地进程与进程p_1, $p_2, \cdots$, p_s的对等通信，并且该过程中不管是通信操作还是计算过程仅在本地操作，并不需要与其他进程进行任何交互.

前面只考虑了 1 个进程的通信收发操作，从全局来考虑的话，每个进程都可能与多个进程进行收发通信操作，这时通信复杂性要高很多，为了简化通信的复杂度，本文提出了一种分层通信模型. 假设进程总数为P，则通信被划分为P层，并按顺序编号为$0,1,\cdots,P-1$. 整个通信序列从第 0 层开始，到$P-1$层通信结束，具体过程如算法 3 所示.

算法 3. 分层通信模型.

/*N表示格点数，P, p 分别为总进程数和当前本地进程的标识号*/

① for $k=1$ to N do

② 初始化数组$rand_id[P]$为−1，用作通信流程控；

③ 计算索引块$J_{\uparrow}^1, J_{\uparrow}^2, \cdots, J_{\uparrow}^s$和$J_{\uparrow k}^{(p_1)}, J_{\uparrow k}^{(p_2)}, \cdots, J_{\uparrow k}^{(p_s)}$；

④ 若不存在上述索引块，令$s=0$；

⑤ for $i=1$ to s do

⑥ $rand_id[p]=J_{\uparrow}^i$；

⑦ end for

⑧ end for

⑨ for $h=0$ to $P-1$ do /*遍历通信层*/

/* 进程号不等于h的其他进程将数据块发送给进程h*/

⑩ if $p\neq h$ and $rand_id[h]\neq -1$ then

⑪ MPI_Send 向量$\boldsymbol{b}$在块$rand_id[h]$的分量到进程h，置通信标志$tag=J_{\uparrow k}^{(h)}$；

⑫ end if

/*进程h接收其他进程发送过来的数据*/

⑬ if $s\neq 0$ and $p=h$ then

⑭ 动态分配s缓存，临时存储从其他进程接收到的数据，令$count=0$；

⑮ while $count<s$ do

⑯ 利用非阻塞探针 MPI_Iprobe 获取通信的信息；

⑰ if $flag=true$探针获取到的进程号为 src，通信标志为 tag then

⑱ 利用 MPI_Irecv 从进程 src 接收通信标志为 tag 的消息，并将接收到的消息存储到第$count$块缓存；

⑲ $count=count+1$；

⑳ end if

㉑ end while

㉒ 等待非阻塞数据，该消息的tag作为数据的索引，接着采用异步方式将数据加

载到 MT-3000 加速簇上进行计算, 计算完成后释放该缓存;

㉓ end if

㉔ end for

在开始通信迭代之前, 每个进程都会根据需要通信的数据块计算出索引, 并将其记录到相应的控制数组中. 当通信遍历到层 $p(p=0,1,\cdots,P-1)$时, 先进行判断该层是否涉及通信, 若没有通信则跳过该层通信, 否则, 进程p仅执行接收消息的操作, 至于其他不等于p的进程, 如果通过计算得出需要与进程p进行了数据交换则将索引块对应的向量$\boldsymbol{b}$分量发送给进程p.

分层通信模型利用哈密顿矩阵的对称性降低了通信复杂度, 各进程无需与其他进程进行“握手”式交互, 仅需本地计算相关索引值, 即可实现高效批量通信. 通信的数据以一个索引块$J_{\uparrow}$为基本单位, 索引块的电子自旋向下分布完全相同, 这表明数据的发送方和接收方的行索引的顺序是完全一致的. 于是, 发送方和接收方都不需要对数据做重排序处理, 基于此数据对齐优点, 发送方不需要开辟缓存来存储通信数据, 接收方可以直接将接收到的数据加载到 MT-3000 的加速簇上进行计算, 极大地提高了计算效率. 同时, 分层设计将每层的通信分开处理, 减少了多进程之间交叉依赖的风险. 每个进程仅在自己的层级内等待数据, 避免了复杂的同步和死锁情况.

为了求解更大的规模, 分层通信最关键的目标是在尽可能节省内存的基础上实现高效的并行通信, 这就要求除了存储必要的向量$\boldsymbol{b}$和进程p, 其他的存储要求尽可能少. 由式(10)可计算出需要通信的索引块数量与所有的索引块数量的比值, 假设取格点$N=32$, 将$N_{\uparrow}$依次取值 5, 7, 10, 由式(10)可计算出这 3 个系统规模的通信数据总占比依次为 0.27, 0.35, 0.44. 这表明随着系统规模增大, 通信数据占比越高并且通信数据所占内存大小接近长向量$\boldsymbol{b}$存储的一半, 如果不采用分层通信, 各个进程同时收发数据, 这就要求每个进程都需要开辟缓存, 用来存储收发数据. 总的来看需要额外增加接近长向量$\boldsymbol{b}$一半的存储, 由于系统资源是有限的, 因此额外存储的增加, 会缩减向量$\boldsymbol{b}$和$\boldsymbol{q}$的规模, 不利于问题求解的扩展. 并且, 还存在一个关键的问题, 需要通信的数据块并不是分散在各个进程中, 而是存在聚集效应. 由图 2 的并行划分可知, 索引块是连续分配的, 因此在格点k, $k+1$处满足“10”的索引块也就是存在通信的数据是连续存储在相邻的进程中, 这会导致在程序的实际执行中这些相邻的进程可能分配到同一个计算节点, 因此在该计算节点中的通信比值远超式(10)的计算, 极端情形比值可能接近 1. 本文设计的分层通信的每一层通信仅有 1 个进程接收数据, 能够极大地节省缓存, 从而实现大规模求解的扩展.

3.5 基于 MT-3000 的异构并行计算

由上述讨论可知, 向量$\boldsymbol{b}$和$\boldsymbol{q}$都可以分成若干个索引块, 每一索引块对应的向量$\boldsymbol{b}$和$\boldsymbol{q}$分量都是连续存储的, 这种存储方式适合 MT-3000 架构的 DMA 数据读写, 可以极大提高计算性能. 本文的异构并行策略是在一个 MT-3000 节点里面启动 4 个进程, 每一个进程控制一个加速簇. 在一个进程中向量$\boldsymbol{b}$和$\boldsymbol{q}$以索引块分量作为基本存储单元, 本进程中的若干个存储单元被均匀分配给加速簇上的 24 个加速阵列, 实现加速阵列级并行.

为了充分发挥加速阵列的性能, 需要开发加速阵列的 SIMD 数据并行. 由 3.3 节的矩阵非零元查找算法可知对于固定的矩阵行索引 i, 该行满足非零元的列索引仅与 i 的二进制表示以及当前遍历到的格点位置 k 决定, 与其他行索引值无关. 这表明算法 2 中的矩阵向量乘计算各行之间没有依赖关系, 可以完全并行执行, 因此支持 MT-3000 的向量化操作, 具体的异构并行流程如算法 4 所示.

算法 4. 基于 MT-3000 的异构并行算法.

/*N表示格点数, p为当前本地进程的标识号*/

/*对角线元素计算*/

① 将矩阵$\boldsymbol{H}^{(p)}$的行索引均匀划分为 24 个区间 $[i_1^k, i_2^k]$, $k=0,1,\cdots,23$;

② for $s=0$ to 23 do

③ 利用 DMA 将向量$\boldsymbol{b}$和$\boldsymbol{q}$在区间$[i_1^k, i_2^k]$的分量发送到加速阵列的向量(AM)空间;

/*每 16 个数据作为一个向量进行 SIMD 计算 */

④ for $(i_1,i_2,\cdots,i_{16})$ 遍历 $[i_1^k, i_2^k]$ do

⑤ 向量计算 $(H_{i_1}^{(p)},\cdots,H_{i_{16}}^{(p)})$;

⑥ $(q_{i_1}^{(p)},\cdots,q_{i_{16}}^{(p)})+=(H_{i_1}^{(p)}b_{i_1},\cdots,H_{i_{16}}^{(p)}b_{i_{16}})$;

⑦ end for

⑧ end for

/*非对角元素计算, 自旋向上 */

⑨ 首先将数组$hash_{k_{\uparrow}}$和$hash_{r_{\uparrow}}$加载到加速簇的高速缓存 GSM 上以提高访存效率;

⑩ for $k=1$ to N do

⑪ 将所有涉及到计算的索引块均匀分配给 24

个加速阵列；

⑫ for $s=0$ to 23 do

⑬ 利用 DMA 将索引块对应向量$\boldsymbol{b}$和$\boldsymbol{q}$的分量发送到加速阵列 AM 空间，同样将$hash_{k_\uparrow}$和$hash_{r_\uparrow}$加载到 AM 空间；

⑭ for 遍历该加速阵列里的索引块$j_\uparrow^s$ do

⑮ 对于索引块$j_\uparrow^s$，$hash_{k_\uparrow}$和$hash_{r_\uparrow}$每 16 行组成一个向量单元，并与标量$hash_{k_\uparrow}[j_\uparrow^s]$和$hash_{r_\uparrow}[j_\uparrow^s]$计算出矩阵非零元值；

⑯ $(q_{i_1}^{(p)},\cdots,q_{i_{16}}^{(p)})+=(H_{i_1}^{(p)}b_{i_1},\cdots,H_{i_{16}}^{(p)}b_{i_{16}})$;

⑰ end for

⑱ end for

⑲ end for

/*非对角元素计算，自旋向下 */

⑳ 首先将数组$hash_{k_\downarrow}$和$hash_{r_\downarrow}$加载到加速簇的高速缓存 GSM 上，提高访存效率；

㉑ for $s=0$ to 23 do

㉒ 将所有涉及到计算的索引块均匀分配给 24 个加速阵列. 利用 DMA 将索引块对应向量$\boldsymbol{b}$和$\boldsymbol{q}$的分量发送到加速阵列 AM 空间，同样将$hash_{k_\downarrow}$和$hash_{r_\downarrow}$加载到 AM 空间；

㉓ for $k=1$ to N do

㉔ for 遍历该加速阵列里的索引块$j_\uparrow^s$ do

㉕ 对于索引块$j_\uparrow^s$，$hash_{k_\downarrow}$和$hash_{r_\downarrow}$每 16 行组成一个向量单元，并与标量$hash_{k_\uparrow}[j_\uparrow^s]$和$hash_{r_\uparrow}[j_\uparrow^s]$计算出矩阵非零元值；

㉖ 在 AM 空间上重排$\boldsymbol{b}$的数据，使得索引顺序跟$\boldsymbol{q}$一致；

㉗ $(q_{i_1}^{(p)},\cdots,q_{i_{16}}^{(p)})+=(H_{i_1}^{(p)}b_{i_1},\cdots,H_{i_{16}}^{(p)}b_{i_{16}})$;

㉘ end for

㉙ end for

㉚ end for

算法 4 分 3 个部分描述哈密顿矩阵乘的 MT-3000 异构并行计算. 首先，考虑对角线上的元素计算，将矩阵的行索引均匀分配给 24 个加速阵列，在加速阵列中每 16 个行作为一个向量单元进行向量计算. 接着，考虑非对角线自旋向上的情形，涉及到矩阵非零元在加速阵列上如何构造. 3.3 节中矩阵非零元的构造中涉及 4 个数组$hash_{k_\uparrow}$，$hash_{k_\downarrow}$，$hash_{r_\uparrow}$和$hash_{r_\downarrow}$. 对于同一个索引块，行索引在自旋向上的分量保持不变，可以考虑将$hash_{k_\uparrow}$和$hash_{r_\uparrow}$加载到 MT-3000 加速簇的缓存 GSM 中，后续的向量计算中可以共享此标量，因此$hash_{k_\uparrow}$和$hash_{r_\uparrow}$不必作为向量加载到 AM 空间中. 另外，同一个索引块中，行索引自旋向下的分量变化剧烈，所以需要将$hash_{k_\downarrow}$和$hash_{r_\downarrow}$与向量$\boldsymbol{b}$和$\boldsymbol{q}$一样按照向量存储方式连续加载到 AM 空间中. 有了这些准备工作，就可以类似对角线上的向量计算一样，完成自旋向上情形的哈密顿矩阵向量乘的向量化计算. 最后，与自旋向上的情形相比，考虑自旋向下的情形有 2 点变化：第 1 点变化是此种情形下遍历格点位置从 1 到 N 时，同一个索引块中的行对应的矩阵非零元的列索引还是在原本的索引块中. 因此为了减少 CPU 内存数据反复在主存与加速簇 AM 空间移动，将格点遍历循环放到加速阵列循环的里面. 第 2 点变化是由于索引块的自旋向下分量变化剧烈，导致向量$\boldsymbol{q}$与$\boldsymbol{b}$的索引顺序不一致，因此需要在加速阵列的 AM 存储空间上对$\boldsymbol{b}$作重排序，保持向量$\boldsymbol{q}$与$\boldsymbol{b}$的顺序的一致性，便于向量化操作.

4 实验结果与分析

4.1 实验规模设置

为了验证本文提出的大规模并行算法的性能，在天河新一代超算系统平台上进行了验证测试，系统配置请参考 3.1 节. 下面主要从并行算法的强可扩展和弱可扩展 2 个方面进行性能分析，为了便于衡量性能，统计时间以 Lanczos 迭代到 192 步的时间作为参考. 为了验证大规模情形，选取格点数为 24, 32 和 36 的 5 组不同电子配置的情形，表 2 给出了这 5 组规模的具体细节. 从表 2 可以看出矩阵维度大小的数量级从 10^9 增加到 10^{11}，相对应的程序内存大小从 0.07 TB 变化到 16.11 TB. 程序的总内存起主导地位的是向量$\boldsymbol{q}$和$\boldsymbol{b}$，以及接收其他进程的通信数据的缓冲区大小，这个缓冲区的大小可能接近向量$\boldsymbol{b}$. 另外，当规模较大时，数组$hash_k$和$hash_r$所占内存要比向量$\boldsymbol{b}$小 1 个数量级，因此可以忽略不计. 所以模型的内存总需求可以按照 3 个向量$\boldsymbol{b}$的大小来估计.

Table 2 Parameter Details for Five Groups of Test Scales Based on the One-Dimensional Hubbard Model

表 2 基于一维 Hubbard 模型的 5 组测试规模的参数详情

格点数	自旋向上	自旋向下	矩阵维数	存储/TB
24	6	6	1.8×10^{10}	0.39
32	5	5	4.1×10^{10}	0.88
36	4	4	3.4×10^{9}	0.07
36	5	5	1.4×10^{11}	3.10
36	6	5	7.3×10^{11}	16.11

4.2 性能分析

分析算法 2 中的执行流程, 整个流程主要分为通信和求解, 而求解的过程中占据支配地位是求解矩阵满足$H_{ij}\neq 0$时的索引对, 这是由于采用“计算换存储的方法”导致每次迭代都需要重复计算非零元索引对. 对于格点数为N的系统, 如果在迭代中计算非零元索引对, 由于需要遍历所有格点位置, 该计算的复杂度最小不会低于$O(N)$, 而本文在 3.3 节提出的非零元快速查找算法的时间复杂度仅为$O(1)$. 相比较而言, 当格点数N越大, 本文提出的快速求解非零元的算法提升越高, 当N超过 24 以后, 性能甚至可以提升 2 个数量级以上.

下面分析 MT-3000 优化前后的性能对比, 为了阐述方便, 以$N=32(N_{\uparrow}=5,N_{\downarrow}=5)$规模为例, 比较 128 进程规模纯 CPU 与 128 进程带加速簇(1 个进程启动 1 个加速簇)的对比测试, 测试结果如图 4 所示. 在进行 MT-3000 异构加速前, 程序的主要 3 个耗时部分分别为自旋向上计算、自旋向下计算、通信, 这 3 个部分中前二者由于规模相同, 时间均为 2 912.24 s, 通信时间为 660.72 s, 通信占比约为 10.18%. 经过 MT-3000 的向量加速优化后, 这二者通信时间不变, 电子自旋向上和自旋向下的计算时间分别变为 156.18 s 和 384.46 s, 二者的加速比分别为 18.64 和 7.57. 由算法 4 的异构计算流程可知自旋向上计算相比自旋向下计算缺少了对向量$\boldsymbol{b}$的索引块的重排序, 因此自旋向上的加速比更高. 从图 4 可以得出, 优化后程序的通信占比从 10.18% 提高到 55% 左右, 使得通信成为影响并行效率的关键因素, 后面的扩展性测试将着重分析通信对程序扩展性的影响.

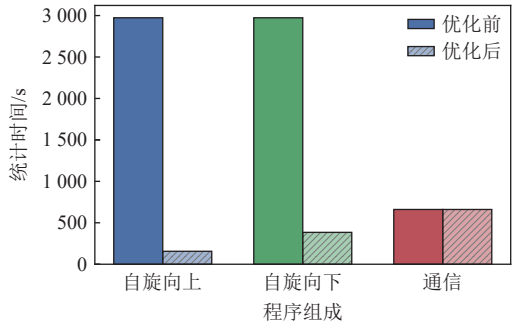


Fig. 4 Comparison of the optimization in the solution process

图 4 求解过程的优化对比

4.3 强扩展测试

本节针对表 2 中的前 4 组规模作了强扩展测试, 根据表 2 的内存需求数据, 针对前 3 个规模$N=24(N_{\uparrow}=6,N_{\downarrow}=6)$、$N=32(N_{\uparrow}=5,N_{\downarrow}=5)$和$N=36(N_{\uparrow}=4,N_{\downarrow}=4)$统一测试了 128 进程到 8 192 进程的结果. 在第 3 节的并行算法设计讨论中, 我们将一个进程的计算任务分配给一个 MT-3000 加速簇, 因此 1 个计算节点启动 4 个进程, 32 个计算节点可支持 128 个进程, 32 个节点的内存需求足以覆盖这 3 组规模. 规模$N=36(N_{\uparrow}=5,N_{\downarrow}=5)$的内存需求为 3.10 TB, 64 个节点的内存容量可以支持该规模计算, 因此该规模的强扩展测试从 256 进程到 8 192 进程. 表 3 给出了 4 组规模在不同进程下完成 Lanczos 迭代到 192 步的运行时间.

Table 3 Strong Scaling Test Results for Four Groups of Scales

表 3 4 组规模的强扩展测试结果 s

进程数	规模			
	24.6+6	32.5+5	36.4+4	36.5+5
128	445.08	1 201.44	121.41	*
256	240.30	686.07	68.65	2 236.69
512	127.38	388.46	37.97	1 338.37
1 024	70.21	216.93	21.57	723.19
2 048	39.53	122.64	11.47	414.26
4 096	21.54	64.42	6.20	231.91
8 192	*	*	*	126.46

注: “*”表示无效.

为了更好地评估程序的性能, 需要对表 3 的结果进行二次加工以提炼出并行效率指标, 这个指标能够更清晰地看出程序的扩展性能. 图 5 给出了前 4 组规模的并行效率曲线图, 其中图 5(a)(b)(c)图像的数据表明$N=24(N_{\uparrow}=6,N_{\downarrow}=6)$, $N=32(N_{\uparrow}=5,N_{\downarrow}=5)$和$N=36(N_{\uparrow}=4,N_{\downarrow}=4)$这 3 组规模强扩展以 128 进程为基准, 4 096 进程的并行效率均可以保持在 58% 以上. 图 5(d)可以得到, 系统规模为 $N=36(N_{\uparrow}=5,N_{\downarrow}=5)$时, 以 256 进程为基准, 8 192 进程的强扩展效率为 55.27%.

强扩展测试结果显示, 并行效率随着进程规模增加呈线性下降趋势, 这主要是大规模并行下通信和同步的开销不断增大所致. 由式(10)及 3.2 节的分析, 可以得出非零元的索引块具备聚集效应, 会聚集在部分进程而导致通信不均衡, 这是影响并行效率的关键因素. 另外还有一个关键原因, 当进程数较少时, 每一个进程分配的索引块较多, 此时存在部分索引块$j_{\uparrow}$与$j_{\uparrow k}$可能处在同一进程, 因此存在一些非零元的索引块不涉及进程通信. 但是随着进程数的增加, 每个进程分配到的索引块数目减少, 导致这些对

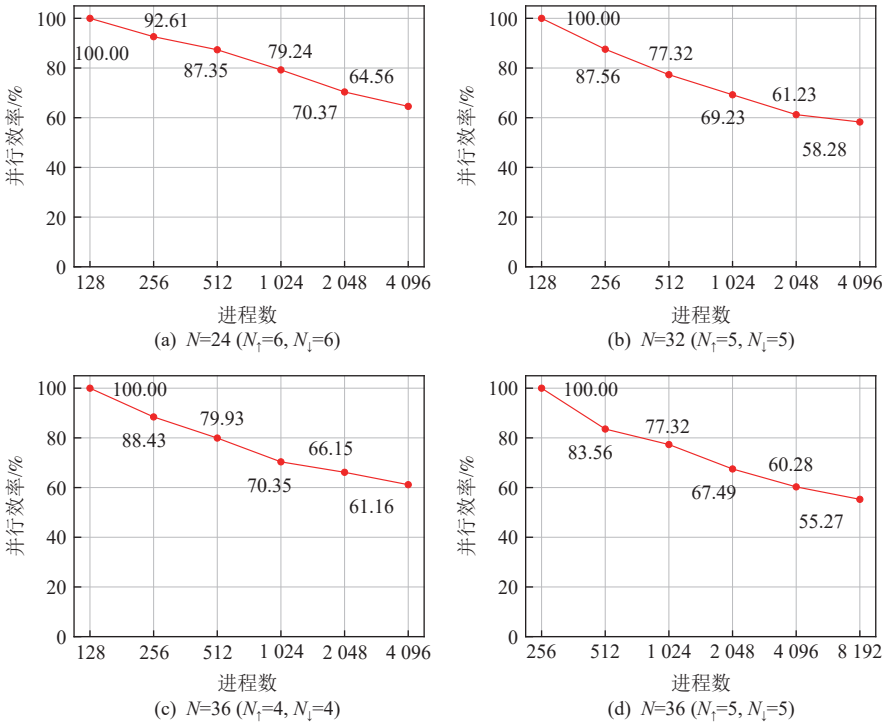


Fig. 5 Parallel efficiency of strong scaling tests

图 5 强扩展测试的并行效率

应的 $j_{\uparrow}$ 与 $j_{\uparrow k}$ 分处到不同进程，从而对于进程数越多的情形，通信量反而增加了，导致并行效率成线性下降，虽然效率有所下降，但是对于强扩展测试，4 096 进程相比 128 进程的并行效率保持在 60%，8 192 进程相比 256 进程可以保持 55% 以上，表明程序具备良好的强可扩展性.

4.4 弱扩展测试

4.3 节分析了强可扩展性能，考虑表 2 中 5 组规模的弱扩展测试. 弱扩展是指问题规模的大小和处理器数量保持相同的增速，从而保证每个处理器的工作量不变. Lanczos 算法的核心过程是矩阵向量乘法，由于非零元的多少与格点数 N 密切相关，所以问题规模的大小由矩阵维数和格点数的乘积决定.

表 4 给出 5 组规模的弱扩展测试结果. 表中第 2 列指出了问题规模与哈密顿维数和格点数有关系，第 3 列是问题规模的增长比例，以 $N=36(N_{\uparrow}=4,N_{\downarrow}=4)$ 作为参考标准，进程数的增长比例与问题规模增长比列一致.

Table 4 Weak Scaling Test Results for Five Groups of Scales

表 4 5 组规模的弱扩展测试结果

系统配置	问题规模	问题规模增长比例	进程数	时间/s
36.4+4	$36\times(3.4\times10^{9})$	1.0	64	221.41
24.6+6	$24\times(1.8\times10^{10})$	3.5	224	240.37
32.5+5	$32\times(4.1\times10^{10})$	10.7	689	288.26
36.5+5	$36\times(1.4\times10^{11})$	41.2	2 637	323.68
36.6+5	$36\times(7.3\times10^{11})$	214.7	13 740	432.02

图 6 给出了弱扩展的并行效率趋势图，可以得出前 4 组规模的弱扩展并行效率均在 68% 以上，仅在规模 $N=36(N_{\uparrow}=6,N_{\downarrow}=5)$ 时效率才急剧下降，这主要是由于该规模相比 $N=36(N_{\uparrow}=4,N_{\downarrow}=4)$ 增长了 214.7 倍，进程数也达到了 13 740，规模越大涉及的通信占比越高，导致效率下降. 但是，规模 $N=36(N_{\uparrow}=6,N_{\downarrow}=5)$ 的并行效率仍然可以保持在 51.25% 以上，尤其 13 740 进程比 2 637 进程规模时，并行效率能保

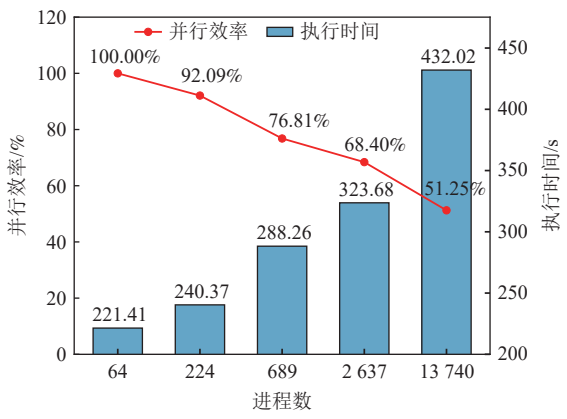


Fig. 6 Execution time and parallel efficiency of weak scaling tests

图 6 弱扩展测试的执行时间与并行效率

持 74.92% 以上, 表明弱扩展性能尚未出现“性能雪崩”式的失效. 另外, 虽然弱扩展按照矩阵的规模与进程规模成比例增长, 但实际上仅凭矩阵规模难以全面表征问题的计算规模. 例如考虑弱扩展的最后 2 组测试 36.6+5($N=36, N_{\uparrow}=6, N_{\downarrow}=5$)和 36.5+5($N=36, N_{\uparrow}=5, N_{\downarrow}=5$), 前者矩阵规模是后者的 5.2 倍. 由式(10), 可计算出两者的非零元索引块比值分别为 0.285 7 和 0.246 0, 再将此比例乘以各自的总索引块数 $\binom{N}{N_{\uparrow}}$, 得到非零索引块分别约为 556 484 和 92 740, 两者比值等于 6.0. 该结果表明在进程数增加 5.2 倍左右规模下, 对应的问题规模却增长了 6.0 倍, 因此实质上弱扩展并行效率分别超过 51.25% 和 74.92%. 综合来看本文提出的并行模型可在更大规模下依旧发挥有效作用, 通信复杂度虽上升, 但不会让效率骤降到完全无法扩展的地步. 只要在硬件资源允许的前提下进行适当的参数调优和通信优化, 方法仍具有进一步扩展到更大规模的潜力.

4.5 收敛性和存储分析

为了验证精确对角化并行算法的正确性, 下面讨论并行算法的收敛性. 图 7 给出了图 5 中 4 组测试规模各自对应的收敛性结果, 图 7(a)(b)(d)的曲线图表明迭代步数接近 200 时误差精度基本上可达到 10^{-10}, 只有图 7(c)中的曲线在迭代步数接近 200 时误差精度达到 10^{-6}. 总体上来看 4 组规模的测试都能保证收敛, 并且可以快速达到合理的误差精度范围, 验证了并行算法的正确性.

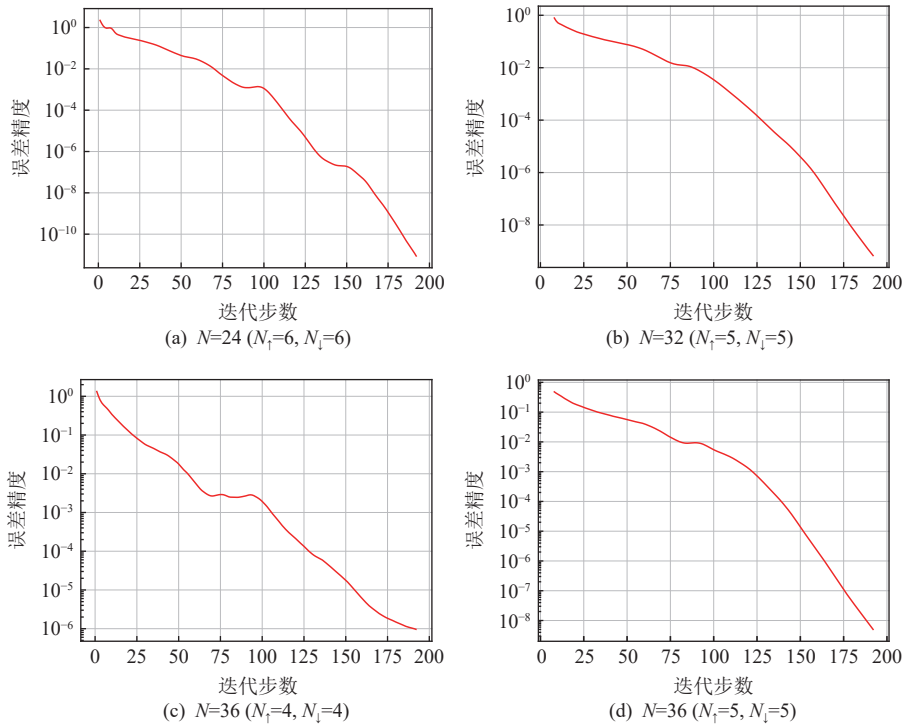


Fig. 7 Convergence for different problem scales

图 7 不同问题规模的收敛性

除了并行通信算法的设计外, 本文算法另一个优势就是能够最低限度地节省内存, 内存需要得越小就能在受限资源下计算更大的问题规模. 表 5 给出本文算法的内存使用情况与文献 [33−34] 作对比的结果. 文献 [33] 在地球模拟器超算系统上提出了针对 Hubbard 模型的 Lanczos 并行算法求解. 文献 [34] 从波函数的对称自适应性角度出发, 提出了基于子晶格编码算法的大规模分布式并行 Lanczos 算法, 实现了 50 个自旋−1/2 粒子的海森堡模型的精确对角化. 表 5 中在矩阵维度为 10^{10} 时, 本文提出的大规模并行 Lanczos 算法所需的内存小到接近 1 个数量级. 而在矩阵维度为 10^{11} 时, 显然, 与文献 [33] 提出的方

Table 5 Comparison of Memory Usage

表 5 内存使用比较

模型	哈密顿矩阵维数	总内存/TB
Triangular 48[34]	2.8×10^{10}	2.50
24 格点(6,6)	1.8×10^{10}	0.39
22 格点(8,8)[33]	1.0×10^{11}	5.40
Square 50[34]	3.2×10^{11}	15.50
36 格点(5,5)	1.4×10^{11}	3.10

法相比，本文算法在实现更大规模模拟的同时，内存需求更低.与文献 [34] 相比，若将本文算法的矩阵规模线性扩展到与文献 [34] 相同（扩大 2.28 倍），则本文算法的内存需求约为 7.07 TB，远低于文献 [34] 的 15.50 TB，体现出本文算法在存储效率方面的显著优势.

5 结　　语

本文针对量子多体系统精确对角化问题，基于哈密顿矩阵的对称性，提出了一种适用于 Hubbard 模型的超大规模稀疏矩阵向量乘并行算法. 在矩阵和向量的存储方面，设计了一种时间换空间的哈密顿矩阵表示方法，每次执行向量乘法之前，动态计算矩阵非零元，节省了矩阵的存储，向量则按行存储均匀分配给各个进程. 在哈密顿矩阵非零元的计算方面，设计了一种快速计算矩阵非零元算法，可实现在时间复杂度为 $O(1)$ 的计算下快速确定哈密顿矩阵非零元的索引号. 在分布式哈密顿矩阵向量乘的数据通信方面，设计了一种分层的通信模式，使得矩阵向量乘过程中各进程间可以高效传输数据. 在天河新一代超算系统上，强扩展测试表明，以 128 进程为基准，扩展到 4096 进程时并行效率维持在 58% 以上；以 256 进程为基准，扩展到 8192 进程时并行效率达到 55.27%，显示出良好的扩展能力. 在弱扩展性测试方面，13 740 进程相比 64 进程并行效率保持在 51.25% 以上.

作者贡献声明：李彪负责实验测试、论文撰写；刘杰负责论文总体架构设计；王庆林提出指导意见和修改论文.

参　考　文　献

[1] Bois F Y, Maszle D R. MCSim: A Monte Carlo simulation program[J]. Journal of Statistical Software, 1997, 2(9): 1−60

[2] Foulkes W M C, Mitas L, Needs R J, et al. Quantum Monte Carlo simulations of solids[J]. Review of Modern Physics, 2001, 73(1): 33−83

[3] Li Z X, Yao H. Sign-problem-free fermionic quantum Monte Carlo: Developments and applications[J]. Annual Review of Condensed Matter Physics, 2019, 10(1): 337−356

[4] Kim A J, Prokof'ev N V, Svistunov B V, et al. Homotopic action: A pathway to convergent diagrammatic theories[J]. Physical Review Letters, 2021, 126(25): 257001

[5] White S R. Spin gaps in a frustrated Heisenberg Model for CaV4O9[J]. Physical Review Letters, 1996, 77(17): 3633−3636

[6] Penc K, Fouet J B, Miyahara S, et al. Ising phases of Heisenberg ladders in a magnetic field[J]. Physical Review Letters, 2007, 99(11): 117201−117201

[7] Jiang H C, Weng Z Y, Sheng D N. Density matrix renormalization group numerical study of the kagome antiferromagnet[J]. Physical Review Letters, 2008, 101(11): 117203-117203

[8] Pradhan S, Patel N D, Trivedi N. Two-magnon bound states in the Kitaev model in a [111] field[J]. Physical Review B, 2020, 101(18): 180401

[9] Bernu B, Lecheminant P, Lhuillier C, et al. Exact spectra, spin susceptibilities, and order parameter of the quantum Heisenberg antiferromagnet on the triangular lattice[J]. Physical Review B, 1994, 50(14): 10048

[10] Lecheminant P, Bernu B, Lhuillier C, et al. Order versus disorder in the quantum Heisenberg antiferromagnet on the kagomé lattice using exact spectra analysis[J]. Physical Review B, 1997, 56(5): 2521

[11] Morita K, Tohyama T. Finite-temperature properties of the Kitaev-Heisenberg models on kagome and triangular lattices studied by improved finite-temperature Lanczos methods[J]. arXiv preprint, arXiv: 1911.09266, 2019

[12] Cullum J K, Willoughby R A. Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Vol. I: Theory[M]. Society for Industrial and Applied Mathematics, 2002

[13] Davidson E R. The iterative caluculation of a few of the lowest Eigenvalues and corresponding Eigenvectors of large real-symmetric matrices[J]. Journal of Computational Physics, 1975, 17(1): 87−94

[14] Yanai T, Fann G I, Gan Z, et al. Multiresolution quantum chemistry in multiwavelet bases: Analytic derivatives for Hartree–Fock and density functional theory[J]. The Journal of Chemical Physics, 2004, 121(7): 2866−2876

[15] Wang Y L, Fabbris G, Dean M P M, et al. EDRIXS: An open source toolkit for simulating spectra of resonant inelastic X-ray scattering[J]. Computer Physics Communications, 2019, 243: 151−165

[16] Vary J P, Maris P, Ng E, et al. Ab initio nuclear structure–the large sparse matrix eigenvalue problem[C]//Journal of Physics: Conf Series. UK: IOP Publishing, 2009, 180(1): 012083

[17] Maris P, Sosonkina M, Vary J P, et al. Scaling of ab-initio nuclear physics calculations on multicore computer architectures[J]. Procedia Computer Science, 2010, 1(1): 97−106

[18] Elias-Miro J, Rychkov S, Vitale L G. NLO renormalization in the Hamiltonian truncation[J]. Physical Review D, 2017, 96(6): 065024

[19] Parkinson J B, Bonner J C. Spin chains in a field: Crossover from quantum to classical behavior[J]. Physical Review B, 1985, 32(7): 4703−4724

[20] Bonner J C, Fisher M E. Linear magnetic chains with anisotropic coupling[J]. Physical Review, 1964, 135(3A): A640−A658

[21] Flocke N. Exact diagonalization study on spin-1/2 ladders as a function of two coupling parameters using the symmetric group approach[J]. Physical Review B, 1997, 56(21): 13673−13676

[22] Nemati S, Batebi S, Mahdavifar S. Concurrence and entanglement entropy in a dimerised spin-1/2 two-leg ladder[J]. The European

Physical Journal B, 2011, 83: 329−335

[23] Usami M, Suga S. Critical exponents of S = 1/2 Heisenberg ladders with ferromagnetic interchain interaction in magnetic fields[J]. Physics Letters A, 1999, 259(1): 53−56

[24] Huse D A. Ground-state staggered magnetization of two-dimensional quantum Heisenberg antiferromagnets[J]. Physical Review B, 1988, 37(4): 2380

[25] Lüscher A, Läuchli A M. Exact diagonalization study of the antiferromagnetic spin-1/2 Heisenberg model on the square lattice in a magnetic field[J]. Physical Review B—Condensed Matter and Materials Physics, 2009, 79(19): 195102

[26] Oitmaa J, Betts D D. The ground state of two quantum models of magnetism[J]. Canadian Journal of Physics, 1978, 56(7): 897−901

[27] Richter J, Schulenburg J, Honecker A. Quantum magnetism in two dimensions: From semi-classical Néel order to magnetic disorder[J]. Quantum Magnetism, 2004: 85−153

[28] Poilblanc D. Phase diagram of the two-dimensional t-J model at low doping[J]. Physical Review B, 1995, 52(13): 9201

[29] Wietek A, Läuchli A M. Chiral spin liquid and quantum criticality in extended S = 1/2 Heisenberg models on the triangular lattice[J]. Physical Review B, 2017, 95(3): 035141

[30] He S, Simon S H, Halperin B I. Response function of the fractional quantized Hall state on a sphere. II. Exact diagonalization[J]. Physical Review B, 1994, 50(3): 1823

[31] Lin H Q, Gubernatis J E, Gould H, et al. Exact diagonalization methods for quantum systems[J]. Computers in Physics, 1993, 7(4): 400

[32] Yamada S, Imamura T, Machida M. 16.447 tflops and 159-billion-dimensional exact-diagonalization for trapped fermion-hubbard model on the earth simulator[C]//Proc of the 2005 ACM/IEEE Conf on Supercomputing. Piscataway, NJ: IEEE, 2005: 44−44

[33] Yamada S, Imamura T, Kano T, et al. High-performance computing for exact numerical approaches to quantum many-body problems on the earth simulator[C]//Proc of the 2006 ACM/IEEE Conf on Supercomputing. Piscataway, NJ: 2006: 47-es

[34] Wietek A, Läuchli A M. Sublattice coding algorithm and distributed memory parallelization for large-scale exact diagonalizations of quantum many-body systems[J]. Physical Review E, 2018, 98(3): 033309

[35] Siro T, Harju A. Exact diagonalization of the Hubbard model on graphics processing units[J]. Computer Physics Communications, 2012, 183(9): 1884−1889

[36] Siro T, Harju A. Exact diagonalization of quantum lattice models on coprocessors[J]. Computer Physics Communications, 2016, 207: 274−281

[37] Kanamori J. Electron correlation and ferromagnetism of transition metals[J]. Progress of Theoretical Physics, 1963, 30 (3): 275–289

[38] Gutzwiller M C. Effect of correlation on the ferromagnetism of transition metals[J]. Physical Review Letters, 1963, 10 (5): 159

[39] Hubbard J. Electron correlations in narrow energy bands[J]. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences. 1963, 276 (1365): 238–257

[40] Saad Y. Iterative Methods for Sparse Linear Systems[M]. Philadelphia: Society for Industrial and Applied Mathematics, 2003

[41] Golub G H, Van Loan C F. Matrix Computations[M]. Baltimore: JHU press, 2013

[42] Pissanetzky S. Sparse Matrix Technology-Electronic Edition [M]. New York: Academic Press, 1984

[43] Lu K, Wang Y, Guo Y, et al. MT−3000: A heterogeneous multi-zone processor for HPC[J]. CCF Transactions on High Performance Computing. 2022, 4 (2): 150–164



Li Biao, born in 1987. PhD candidate. His main research interests include parallel algorithm and high performance computing.

李　彪, 1987 年生. 博士研究生. 主要研究方向为并行算法、高性能计算.



Liu Jie, born in 1969. PhD, professor, PhD supervisor. His main research interest includes supercomputer, CFD, artificial intelligence, parallel algorithms, parallel applications, and high-performance computing.

刘　杰, 1969 年生. 博士, 研究员, 博士生导师. 主要研究方向为超级计算、计算流体力学(CFD)、人工智能、并行算法、并行应用、高性能计算.



Wang Qinglin, born in 1987. PhD, associate professor. Member of CCF. His main research interests include parallel algorithm, reconfigurable computing, and machine learning.

王庆林, 1987 年生. 博士, 副研究员. CCF 会员. 主要研究方向为并行算法、可重构计算、机器学习.