计算机科学
COMPUTER SCIENCE

第一性原理极化率计算中的众核优化方法研究

罗海文, 吴扬俊, 商红慧

引用本文

罗海文, 吴扬俊, 商红慧.第一性原理极化率计算中的众核优化方法研究[J]. 计算机科学, 2023, 50(6): 1-9.

LUO Haiwen, WU Yangjun, SHANG Honghui. Many-core Optimization Method for the Calculation of Ab initio Polarizability [J]. Computer Science, 2023, 50(6): 1-9.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220700162

第一性原理极化率计算中的众核优化方法研究

罗海文　吴扬俊　商红慧
中国科学院计算技术研究所处理器芯片全国重点实验室　北京 100190
(luohaiwen20g@ict.ac.cn)

摘　要　基于量子力学的密度泛函微扰理论(DFPT)可以用来计算分子和材料的多种物理化学性质,目前被广泛应用于新材料等领域的研究中;同时,异构众核处理器架构逐渐成为超算的主流。因此,针对异构众核处理器重新设计和优化DFPT程序以提升其计算效率,对物理化学性质的计算及其科学应用具有重要意义。文中对DFPT中一阶响应密度和一阶响应哈密顿矩阵的计算针对众核处理器体系结构进行了优化,并在新一代神威处理器上进行了验证。优化技术包括循环分块、离散访存处理和协同规约。其中,循环分块对任务进行划分从而由众核并行地执行;离散访存处理将离散访存转换为更高效的连续访存;协同规约解决了写冲突问题。实验结果表明,在一个核组上,优化后的程序性能较优化前提高了8.2～74.4倍,并且具有良好的强可扩展性和弱可扩展性。

关键词:密度函数微扰理论;第一性原理计算;高性能计算;新一代神威异构众核处理器
中图法分类号　TP391

Many-core Optimization Method for the Calculation of Ab initio Polarizability

LUO Haiwen,WU Yangjun and SHANG Honghui
State Key Laboratory of Processors,Institute of Computing Technology,Chinese Acadamy of Science,Beijing 100190,China

Abstract　Density-functional perturbation theory(DFPT) based on quantum mechanics can be used to calculate a variety of physicochemical properties of molecules and materials and is now widely used in the research of new materials. Meanwhile,heterogeneous many-core processor architectures are becoming the mainstream of supercomputing. Therefore,redesigning and optimizing DFPT programs for heterogeneous many-core processors to improve their computational efficiency is of great importance for the computation of physicochemical properties and their scientific applications. In this work,the computation of first-order response density and first-order response Hamiltonian matrix in DFPT is optimized for many-core processor architecture and verified on the new generation Sunway processors. Optimization techniques include loop tiling,discrete memory access processing and collaborative reduction. Among them,loop tiling divides tasks so that they can be executed by many cores in parallel;discrete memory access processing converts discrete accesses into more efficient continuous memory accesses;collaborative reduction solves the write conflict problem. Experimental results show that the performance of the optimized program improves by 8.2 to 74.4 times over the pre-optimization program on one core group,and has good strong scalability and weak scalability.

Keywords　Density-functional perturbation theory,First-principle calculation,High-performance computing,New generation Sunway heterogeneous many-core processor

1　引言

基于量子力学的密度泛函微扰理论(Density-Functional Perturbation Theory,DFPT)可以用来计算多种物理响应性质[1-2],例如,极化率是分子在外电场下的响应性质;拉曼光谱可以用来得到分子的振动信息,从而进一步解析得到分子的结构信息[3]。同时,异构众核处理器架构逐渐成为超算的主流。因此,针对异构众核处理器重新设计和优化DFPT程序以提升其计算效率,对物理化学性质的计算及其科学应用具有重要意义。

对于全电子全势框架下的DFPT模块[4],本文在神威平台优化DFPT模块中一阶响应密度和一阶响应哈密顿矩阵的计算。具体的软件开发和优化在分子材料模拟软件包FHI-aims[5-6]中完成。

本文的主要贡献如下:

(1)对DFPT模块中极化率的计算中一阶响应密度和一阶

到稿日期:2022-07-18　返修日期:2022-11-26
基金项目:国家重点研发计划(2020YFB1709500);国家自然科学基金(22003073)
This work was supported by the National Key Research and Development Program of China(2020YFB1709500) and National Natural Science Foundation of China(22003073).
通信作者:商红慧(shanghonghui@ict.ac.cn)

响应哈密顿矩阵的部分进行了热点分析，并实现了一系列众核优化技术来优化热点函数，包括循环分块、离散访存处理和主从核协同规约。

(2)通过不同的测试用例测试了优化后程序的整体加速比以及热点函数的带宽利用率；通过聚乙烯分子体系测试了优化后程序的强可扩展性和弱可扩展性。测试强可扩展性的聚乙烯分子的原子数为 770，进程数从 10 增加到 160。当进程数为 160 时，计算一阶响应密度部分和计算一阶响应哈密顿矩阵部分的并行效率分别为 90.0%和 88.0%。测试弱可扩展性的聚乙烯分子的原子数从 98 增加到 3074，进程数从 1 增加到 360，当原子数为 3074、进程数为 360 时，计算一阶响应密度部分和计算一阶响应哈密顿矩阵部分的并行效率分别为85.0%和 83.9%。

2 背景

2.1 密度泛函微扰理论

DFPT 的流程如图 1 所示。在执行 DFPT 之前，需要先求解密度泛函理论(Density Functional Theory，DFT)[7]方程获得 0 阶密度。DFT 的计算完成之后，迭代执行 DFPT 的 4 个部分直到收敛，收敛的条件为相邻两次迭代计算得到的一阶响应密度矩阵中对应元素的差值的绝对值之和小于用户定义的阈值。

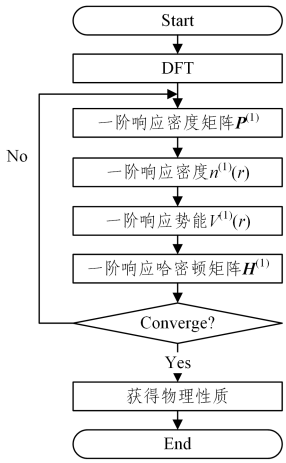


图 1 密度泛函微扰理论(DFPT)流程图

Fig. 1 Flowchart of density-functional perturbation theory(DFPT)

一阶响应密度矩阵可以由式(1)得到：

$$\boldsymbol{P}_{u,v}^{(1)}=\sum_i f(\varepsilon_i)(\boldsymbol{C}_{u,i}^{(1)}\boldsymbol{C}_{v,i}^{(0)}+\boldsymbol{C}_{u,i}^{(0)}\boldsymbol{C}_{v,i}^{(1)}) \tag{1}$$

其中，$f(\varepsilon_i)$表示费米-狄拉克分布，$\boldsymbol{C}_{u,v}$表示波函数的膨胀系数。

一阶响应密度可以由式(2)得到：

$$\boldsymbol{n}^{(1)}(r)=\sum_{u,v}\boldsymbol{P}_{u,v}^{(1)}\boldsymbol{\chi}_u^{(0)}(r)\boldsymbol{\chi}_v^{(0)}(r) \tag{2}$$

其中，$\boldsymbol{P}^{(1)}$为上一步计算得到的一阶响应密度矩阵，$\boldsymbol{\chi}_v(r)$为基组。

一阶响应势能$\boldsymbol{V}^{(1)}(r)$通过求解式(3)所示的泊松方程得到。

$$\nabla^2\boldsymbol{V}^{(1)}(r)=\boldsymbol{n}^{(1)}(r) \tag{3}$$

最后，通过对一阶响应势能和基组积分得到一阶响应哈密顿矩阵，如式(4)所示：

$$\boldsymbol{H}_{uv}^{(1)}=\langle\boldsymbol{\chi}_u|\boldsymbol{V}^{(1)}(r)|\boldsymbol{\chi}_v\rangle \tag{4}$$

2.2 新一代神威异构众核处理器

本文使用新一代神威超级计算机进行加速，该系统采用了新一代国产高性能异构众核处理器，其结构如图 2 所示。处理器包括 6 个核组，每个核组有 65 个核，其中包括一个通用的计算主核(MPE)和一个由 64 个精简的计算从核(CPE)组成的从核组。主核主要用于管理从核和通信。从核组组织为 8×8 的网格，每 4 个从核共享一个从核簇管理部件。主核和从核分别支持 256 字节和 512 字节的向量指令。

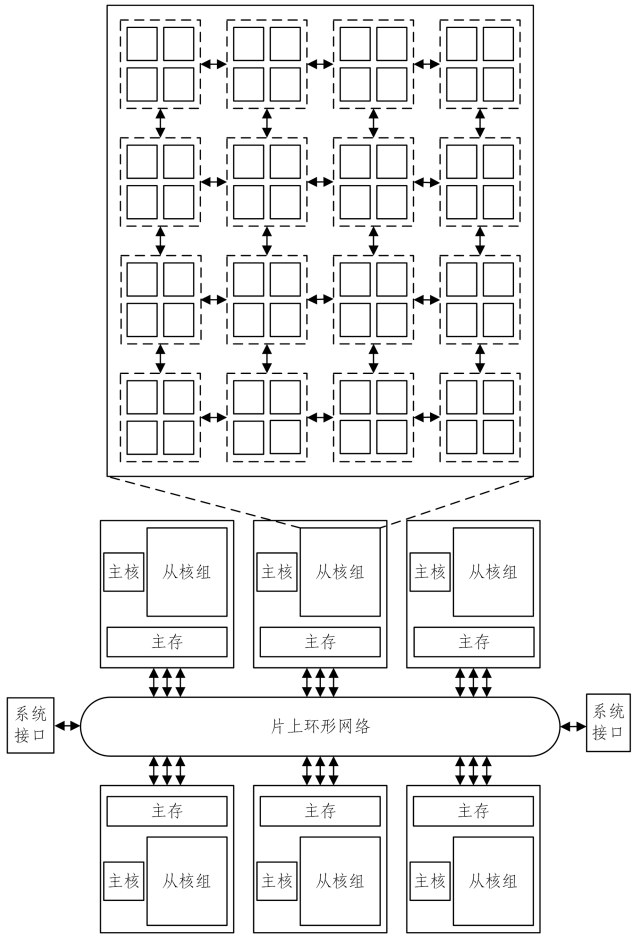


图 2 新一代神威众核处理器体系结构

Fig. 2 Architecture of the new generation Sunway many-core processor

每个核组有一个内存控制器，连接大小为 16 GB 的主存，理论带宽为 52 GB/s。主存被主核和从核共享。主核有 32 kB 一级数据缓存、32 kB 一级指令缓存，以及 512 kB 二级缓存。从核有 32 kB 一级指令缓存和大小为 256 kB 的高速暂存存储器，在读取高速暂存存储器中的数据时具有低访问延迟。高速暂存存储器可以全部被配置为本地设备内存(LDM)，由用户完全控制，也可以把部分空间作为本地数据缓存(LDcache)，由硬件控制。

内存和 LDM 之间的数据传输可以由直接内存访问(DMA)和全局读写(gst/gld)指令来实现。DMA 传输一个或多个数据块，gst/gld 指令只能传输一个数据。DMA 分为

连续 DMA 和跨步 DMA，连续 DMA 读写内存的一个数据块，跨步 DMA 读写若干数据块，数据块与数据块之间有固定的跨步。通过 DMA 在内存与 LDM 之间传输数据能充分利用带宽，当数据块大小大于 1 024 个字节时，DMA 的带宽利用率可达到 80%。LDM 和 LDM 之间的数据传输由远程内存访问(RMA)来实现。

2.3 相关工作

文献[8]同样描述了在新一代神威异构处理器上加速一阶响应密度和一阶响应哈密顿矩阵的计算的工作，该工作将计算一阶响应密度和一阶响应哈密顿矩阵这两部分作为两个 kernel，通过从核组加速，在计算一阶响应哈密顿矩阵部分，通过神威 OpenACC(见 3.3.3 节)更新稀疏形式的一阶响应哈密顿矩阵的值。

3 设计与实现

3.1 并行性分析

FHI-aims 在初始化阶段会构建以原子核的几何坐标为中心的非均匀径向球形格点[9]，并通过实空间区域分解算法[10]将所有格点划分为若干个 batch，每个 batch 包含若干个格点。因此进程级的并行策略是每个进程负责计算若干个 batch，进程间通过 MPI 进行通信。单个进程在计算一阶响应密度和一阶响应哈密顿矩阵的过程中，一层循环的循环次数是该进程分配到的 batch 数。在一层循环内部有两类循环：一类是循环次数为格点数的循环，一类是循环次数与该进程中基函数的个数有关的循环。在众核处理器上的线程级并行策略有两种：一种是以每个 batch 的计算作为一个任务单位，并行计算多个 batch，这是文献[8]中采用的并行策略；另一种是在计算每个 batch 时以每个格点或每个基函数的计算作为一个任务单位，并行计算多个格点或多个基函数，即将一层循环内部的循环进行划分。

3.2 设计

从 2.2 节可以得知，从核是新一代神威异构众核处理器性能的主要来源，因此，我们利用从核来加速计算一阶响应密度和一阶响应哈密顿矩阵的部分热点函数，而主核用来启动从核以及执行非热点函数。一阶响应密度和一阶响应哈密顿矩阵具体的计算流程如图 3 所示，其中函数 *Get density matrix*(*kernel*3)为计算一阶响应密度部分特有，目的是获取一阶响应密度矩阵，并将一阶响应密度矩阵转换成稠密矩阵形式。函数 *Tab atom centered coords*(*kernel*1)和函数 *Calculte waves*(*kernel*2)计算得到 ***waves*** 矩阵。函数 *Calculate first order density/first order H* 通过矩阵乘法计算得到一阶响应密度或一阶响应哈密顿矩阵，这一部分我们通过调用神威平台的高性能数学库 *xMath* 来加速，具体内容将在 3.3.2 节中介绍。函数 *Convert dense matrix to sparse matrix*(*kernel*4)为计算一阶响应哈密顿矩阵部分特有，目的是将计算得到稠密矩阵形式的一阶响应哈密顿矩阵转换成稀疏矩阵形式。在这个过程中多个从核在更新稀疏矩阵时会发生写冲突，为了避免写冲突并且提高主存的带宽利用率，我们提出了主从核协同规约的方法，其中主核通过调用函数 *Update first order H sparse matrix* 来更新稀疏矩阵形式的一阶响应哈密顿矩阵。主从核协同规约的具体内容将在 3.3.3 节介绍。

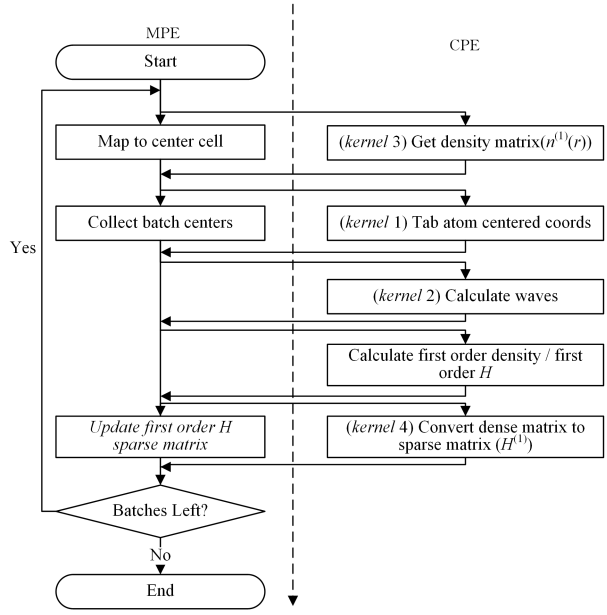


图 3 一阶响应响应密度和一阶响应哈密顿矩阵的计算流程

Fig. 3 Calculation flowchart of first-order response density and first-order response Hamiltonian matrix

3.3 优化方法总结

3.3.1 循环分块

循环分块通过把循环划分成若干个循环块来挖掘循环中的局部性，是程序优化中的常用技术。文献[11-13]提出了基于 LDM 体系结构的循环分块模型。同时，在神威平台，循环块也可以作为从核的基本任务单位被从核组并行地执行。因此，从核优化的第一步是将 kernel 中的循环划分成若干个循环块，然后将这些循环块分配给 64 个从核执行。从核在执行循环块的过程中，为了提高访存效率，将访存模式连续的输入数组的分块通过 DMA 传输到 LDM 中，从核直接访问 LDM 中的数据并进行计算，将计算结果暂存 LDM 中，循环块执行完毕后再将暂存的计算结果通过 DMA 传输到主存中。

*kernel*1 是通过一个二重循环计算一个二维数组，如算法 1 所示。将该二重循环分成 64 个循环块，每一个循环块计算结果数组中的一个数据块，同时将访存模式连续的两个输入数组都划分为 8 个数据块通过 *DMA* 传输，划分方式如图 4 所示。每个从核计算一个数据块，其中第 i 行第 j 列的从核需要 *centers_basis_integrals* 数组的第 i 个数据块和 *coord_current* 数组的第 j 个数据块。

算法 1 Tab atom centered coords

```
输入：(coord_current，coords_center，centers_basis_integrals)
输出：dir_tab
1. for i←1 to n_points do
2.   for j←1 ton_centers_integrals do
3.     dir_tab[1:3][j][i]=coord_current[1:3][i]-coords_center[1:
       3][centers_basis_integrals[j]]
5.   enddo
6. enddo
```

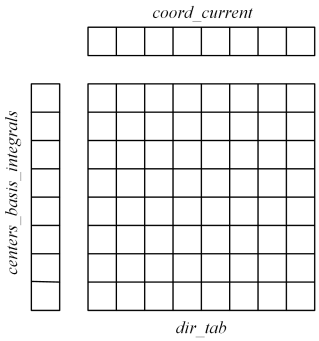


图 4 Tab atom centered coords 的循环分块方式

Fig. 4 Loop blocking methods of Tab atom centered coords

*kernel*2 的循环结构为一个一层循环内部嵌套多个二层循环计算 ***waves*** 矩阵，由于多个二层循环之间有数据依赖，所以只将一层循环划分成 64 块，每一块计算 ***waves*** 矩阵中的若干列，如图 5 所示。

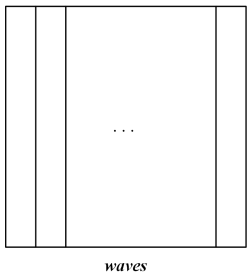


图 5 Calculate ***waves*** 的循环分块方式

Fig. 5 Loop blocking methods of Calculate ***waves***

*kernel*3 为一个三重循环，将稀疏矩阵转换成稠密矩阵，如算法 2 所示。*kernel*3 的分块方式是使用 round-robin 调度算法将一层循环划分成 64 块，如图 6 所示。为了方便表示，把同一个从核负责计算的不连续的若干列表示在一起。因为内层循环的执行时间会随着 i 的增大而增加，所以这种分块方式能够保证从核组的负载均衡。访存模式为连续的数组 *i_basis_index* 在循环块开始执行之前通过 DMA 传输到 LDM 中。对于数组 *column_index_hamiltonian*，每执行一次最内层循环会访问该数组从索引 *start* 到索引 *end* 的数据，因此在每次开始执行最内层循环之前会通过 DMA 将这部分数据传输到 LDM 中。在写回结果时，分块中连续的一列数据通过连续 DMA 传输，分块中一行中的数据通过跨步 DMA 传输。

算法 2 Get density matrix

输入：(i_basis_index, index_hamiltonian, column_index_hamiltonian, first_order_density_matrix_sparse)

输出：first_order_density_matrix_con

```
1. for i←1 to n_compute do
2.     i_basis=i_basis_index[i]
3.     start=index_hamiltonian[1][1][i_basis]
4.     end=index_hamiltonian[2][1][i_basis]
5.     for j←1 to i do
6.       j_basis=i_basis_index[j]
7.       for k←start to end do
8.         if column_index_hamiltonian[k]=j_basis
9.           first_order_density_matrix_con[i][j]=first_order_density_matrix_sparse[k]
10.          first_order_density_matrix_con[j][i]=first_order_density_matrix_sparse[k]
11.          i_index_real=k
12.          break
13.        else if column_index_hamiltonian[k]>j_basis
14.          i_index_real=k
15.          break
16.        endif
17.      enddo
18.      start=i_index_real
19.    enddo
20. enddo
```

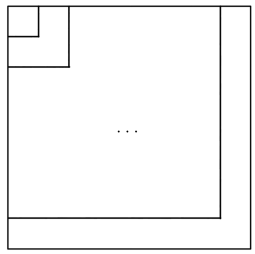


图 6 Get first order density matrix 的循环分块方式

Fig. 6 Loop blocking methods of Get first order density matrix

*kernel*4 通过一个二重循环将稠密矩阵转换成稀疏矩阵，其与 *kernel*3 的区别在于 *kernel*4 在算法 2 第 10 行中执行相反的操作，因此分块方式与 *kernel*3 一致。*kernel*4 在写回结果时的代码可以简化成 $arr[i]+=val[j]$，其中 i 是规约索引，$val[j]$ 为规约值。由于 64 个从核在更新 *arr* 数组时会出现多个从核同时更新同一个规约索引处的值从而发生写入冲突，因此，在这一步优化中我们使用神威平台提供的 OpenACC 的分布式规约子句来将结果写回主存，具体内容见 3.3.3 节。

3.3.2 离散访存处理

从核在访问访存模式不连续的数组时，如果不经过处理，就只能使用 gld/gst 指令访问，导致访存效率低下。为了提高访存效率，需要将离散的访存模式转换成连续的访存模式，然后使用 DMA 来传输数据。

访存模式不连续的数组可以分成 3 类：第一类是访存模式固定的数组，当每次对数组 A 的访问或者每个从核对数组 A 的访问模式都为 $A[B[i]]$，并且数组 B 不会发生改变，则 A 的访存模式是固定的；第二类是访存模式不固定的数组，对该类数组的访问取决于其他会随着程序执行而改变的数组或者程序中的判断条件是否成立；第三类为对称矩阵，由于对称性，只需要连续地传输一半的数量。

对于第一类数组，虽然访存模式不是连续的，但是每次访问时需要的数据不会改变，可以对这类数组做预处理[14]，把需要的数据保存在数组 C 中，$C[i]=A[B[i]]$，从而从核对数组 A 的离散访存可以转换成对数组 C 的连续访存，进而使用 DMA 提高带宽。以 *kernel*1 中对数组 *coords_center* 的访问为例，在预处理之前，对数组 *coords_center* 的访问取决于

数组 *centers_basis_integrals* 的值,因此只能通过 *gld* 指令每次从主存中读取数组 *coords_center* 的一个数据,但是由于数组 *centers_basis_integrals* 的值始终不会改变,因此每次执行这个 *kernel* 时,从主存中读取数组 *coords_center* 的数据也不会改变,可以在计算一阶响应密度之前将这些值保存在大小为 *n_centers_integrals* 的数组 *pre_coords_center* 中,之后在执行 *kernel*1 时只需要连续的访问 *pre_coords_center* 即可。经过处理之后,该部分的数据访问量降低了一半,同时连续的数组可以使用 *DMA* 传输,提高了数据传输速率。

对于第二类数组,我们使用类似于 cache 的机制来优化访存效率[15]。由于这类数组的访存模式不固定,因此不能预测下一次访问数组时会访问该数组中的哪个值。我们可以在 LDM 中创建一个缓冲区来维持该数组的一个数据块副本,在第一次访问该数组时,不再使用 gld 指令读取一个数据,而是通过 DMA 将这个数据周围的数据都读入到 LDM 的缓冲区中,下一次再访问该数组时先检查缓冲区中是否有需要的数据的副本,若有则直接读取,若无则更新缓冲区。通过 DMA 读入一块新的包含所需数据的数据块副本,然后再读取缓冲区的值。

第三类数组对应的是 *kernel*3 中数组 *first_order_density_matrix_con*,该数组是函数 *Calculate first order density* 的输入矩阵之一,并且是对称矩阵,因此在写回结果时只需写回上三角部分的数据,即分块中每一列的数据,即可消除通过跨步 DMA 写回行数据的操作。而在 *Calculate first order density* 的矩阵乘法部分,我们可以调用神威平台的高性能数学库 xMath 的对称矩阵乘法。

3.3.3 神威 OpenACC 分布式规约子句

神威 OpenACC 是以新一代神威众核处理器为目标定制的并行编译工具,支持 OpenACC 语法。同时,为了充分利用新一代神威众核处理器,神威 OpenACC 扩展了标准 OpenACC,比如分布式规约子句[16]。分布式规约子句由从核代码调用,在编译初期神威 OpenACC 编译器会把分布式规约子句转换成源代码,这段代码会执行如下的操作:

(1)把 *arr* 数组等分成 64 个部分,计算每个部分在 *arr* 数组中的开始索引和结束索引,每个从核负责更新数组中一个部分的值。

(2)每个从核在 LDM 上创建 64 个发送缓存单元和 64 个接收缓存单元,分别用于发送和接收其他从核通过 RMA 传输的规约值和规约索引,同时在 LDM 上创建一个缓冲区 buf 用来缓冲主存中 *arr* 数组的一个数据块。

(3)在获得一对规约索引和规约值之后,不再直接进行规约操作,而是根据当前的规约索引,确定此次规约操作由哪个从核执行,并把规约索引和规约值缓存到相应的发送缓存单元,检查缓存单元是否已满。如果没满,则继续获取规约索引和规约值;如果已满,则把该发送缓存单元的数据通过 RMA 发送给对应的从核,并清空该发送缓存单元。

(4)轮询 64 个接收缓存单元,若发现有接收缓存单元存在有效数据,则根据该接收缓存单元的规约索引和规约值进行规约。

(5)规约过程中遍历接收缓存单元,对于遍历得到的每一对规约索引和规约值,如果发现该规约索引对应的值已经缓冲到 buf 中,则直接更新 buf 中的该值,否则先将该 buf 中的数据通过 DMA 的方式写回内存,再通过 DMA 将内存中对应的一个包含当前规约索引的数据块写到 buf 中,然后更新 buf。

3.3.4 主从核协同规约

在 3.3.3 节中,我们使用神威 OpenACC 的分布式规约子句来避免从核组在更新一阶响应哈密顿矩阵时产生的写入冲突问题,本节将介绍主从核协同规约。主从核协同规约的思路是从核只负责获取规约索引和规约值,而将规约值写入数组中,规约索引位置的操作由主核来完成,因此主核和从核之间需要协同。算法 3 描述了主从核协同规约方法中主核和从核分别需要执行的操作。每个从核需要在 LDM 中开辟一块空间作为保存规约索引和规约值的发送缓冲区,同样,主核需要在主存中开辟一块空间作为接收缓冲区,接收 64 个从核发送过来的规约值和规约索引,大小为发送缓冲区的 64 倍。由于从核获取规约索引和规约值的速率和主核进行规约操作的速率可能不一致,因此需要在主存中设置一个大小为 64 的 *flag* 数组来保证主核和从核组之间的同步性。以 0 号从核为例,当 *flag*[0]为 1 时,0 号从核可以向接收缓冲区发送数据,而主核在轮询到 *flag*[0]之后不读取接收缓冲区中 0 号从核对应的数据进行规约操作;当 *flag*[0]的值为 0 时,0 号从核暂停向接收缓冲区中发送数据,直到 *flag*[0]的值变成 1,而主核在轮询到 *flag*[0]之后读取接收缓冲区中 0 号从核对应的数据进行规约操作。初始时 *flag* 的值为 1,每个从核在向接收缓冲区发送完数据之后会将从核对应的 *flag* 的值置为 0,主核在处理完接收缓冲区中一个从核的数据后会将该从核对应的 *flag* 值置为 1。

算法 3 主从核协同规约

输入:get_index:获取一个规约索引的函数;get_value 为获取一个规约值的函数;test_done 为测试从核是否运行结束的函数,从核运行结束返回 1,否则返回 0;update_matrix 为主核更新一阶响应哈密顿矩阵的函数,当参数为 1 时更新 1 号从核对应的接收缓冲区中的数据;n 为单个从核可以获取到的规约索引和规约值的个数;flag 为大小为 64 的数组,存放在主存中,初始化为 1;LDM_index_buffer 为 LDM 上用来保存规约索引的缓冲区;LDM_value_buffer 为 LDM 上用来保存规约值的缓冲区;Main_memory_index_buffer_x 为第 x 号从核在主存的规约索引接收缓冲区;Main_memory_index_buffer_x 为第 x 号从核在主存的规约值接收缓冲区;buffer_size 为缓冲区的大小;MYID 为从核号

输出:first_order_H_sparse:稀疏矩阵形式的一阶响应哈密顿矩阵

```
Procedure MPE_reduction
1. while(!test_done())
2.   for i←0 to 63 do
3.     if flag[i]=0
```

```
4.        update_matrix(i)
5.        flag[i]=1
5.      endif
6.    enddo
7. endwhile

Procedure CPE_get_index_and_value
1. count=0
2. for i←1 to n do
3.    LDM_index_buffer[count]=get_index(i)
4.    LDM_value_buffer[count]=get_value(i)
5.    count++
6.    if count=buffer_size
7.       while(flag[MYID]);//等待 flag 值为 1
8.       DMA_put(LDM_index_buffer,Main_memory_index_buffer_
         MYID)
9.       DMA_put(LDM_value_buffer,Main_memory_value_buffer_
         MYID)
10.      flag[MYID]=0
11.      count=0
12.   endif
13. enddo
```

主从核协同规约的流程如图 7 所示，在从核部分，1)从核 $X(0\leqslant X\leqslant 63)$获取规约索引和规约值并暂存在发送缓冲区中，直到发送缓冲区满；2)发送缓冲区满后，通过 gld 指令读取 $flag[X]$的值直到 $flag[X]$的值为 1；3)当读取到 $flag[X]$的值为 1 后，通过 DMA 将发送缓冲区的内容传输到主存，然后通过 gst 指令将 $flag[X]$值置 0。重复上述操作直到获取所有的规约索引和规约值。

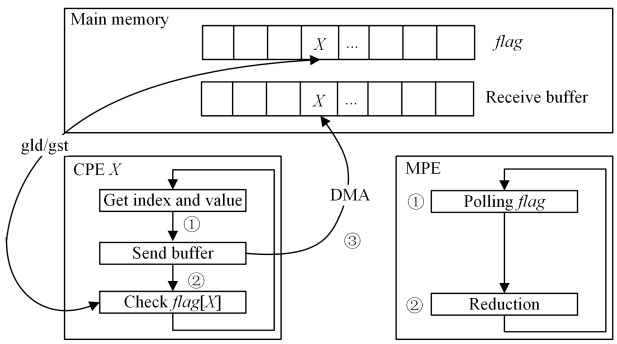


图 7 主从核协同规约

Fig. 7 MPE-CPE collaborative reduction

在主核部分，1)轮询 $flag$ 数组；2)当轮询到 $flag[X]$的值为 0 时，读取 X 号从核对应的接收缓冲区的数据进行规约操作。重复上述操作直到从核执行完毕。

4 性能分析

4.1 测试用例

本文使用的测试用例如表 1 所列，测试用例 1 到测试用例 10 为硅扩展体系，其中测试用例 1 到测试用例 5 的原子数为 2，测试用例 6 到测试用例 10 的原子数为 16。测试用例 11 是新冠病毒(SARS-CoV-2)的刺突糖蛋白质上的受体结合区(RBD)，原子数为 3006。

表 1 用于性能分析的测试用例

Table 1 Test case for performance analysis

	Test case	grids	basis	points per batch	MPI task
Si 2 atoms	#1	35836	18	100	6
	#2	35836	36	300	6
	#3	56860	50	100	6
	#4	35836	72	100	6
	#5	35836	114	100	6
Si 16 atoms	#6	286688	144	100	6
	#7	286688	288	300	6
	#8	454880	400	100	6
	#9	286688	576	100	6
	#10	286688	912	100	6
RBD	#11	16 182 074	28 949	100	64

4.2 正确性验证

我们通过配置 X86 架构处理器的超算平台来验证优化后程序的正确性。X86 架构处理器为 AMD EPYC 7452 32-Core Processor，频率为 2.4GHz。在 X86 超算平台和神威平台使用相同的 MPI 进程分别对 11 个测试用例进行模拟。其中 X86 超算平台在矩阵乘法部分调用的数学库为 Math Kernel Library(2018.2.199)，X86 超算平台执行 DFPT 程序得到极化率 1。在神威平台执行串行程序和优化后的并行程序，分别得到极化率 2 和极化率 3。比较两个平台执行 DFPT 程序得到的 3 个极化率，误差为 10^{-10}。

我们还对两个平台模拟 11 个测试用例所用的时间进行了统计，如图 8 所示，其中 X86 为 X86 超算平台上的运行时间，Sunway 为神威平台执行优化后的并行程序的运行时间。图 8(a)给出了计算一阶响应密度部分与 X86 平台的对比，加速比为 1×到 6×；图 8(b)给出了计算一阶响应哈密顿矩阵部分与 X86 平台的对比，加速比为 0.7×到 5×。

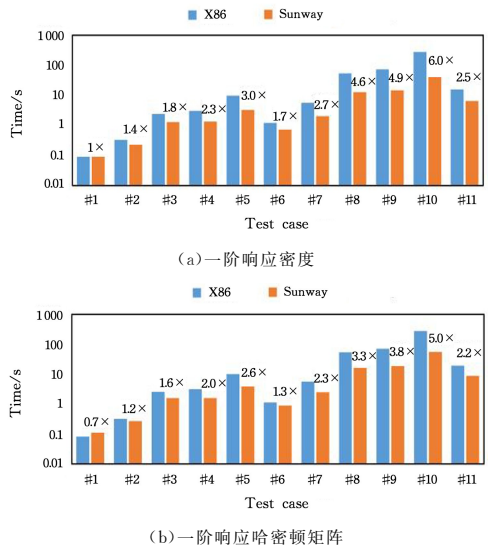


图 8 与 X86 平台的性能对比

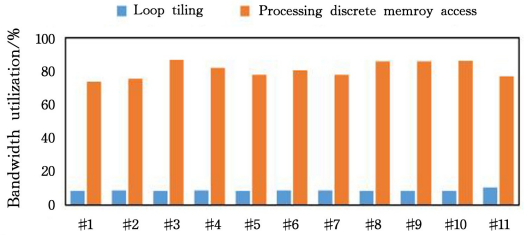
Fig. 8 Performance comparison with X86 platform

当基组数增大时，*kernel*3 的执行效率提升(*kernel*3 的执行效率随基组数增大而提升的原因已在 4.3 节讨论)，加速比增大。当基组数较小时，*kernel*3 的执行效率降低，同时程序的整体运行时间减少，启动从核的时间无法忽略，导致加速比降低。

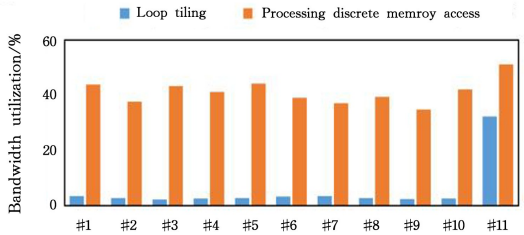
4.3 带宽利用率

kernel 的带宽利用率可以通过执行 kernel 所需要的时间和执行 kernel 期间从核组与主存之间的数据传输量来计算，其中数据传输量包括通过 DMA 和 gld/gst 指令从主存中读取和写入的数据量的总和。

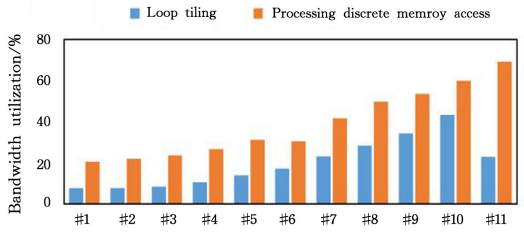
图 9 给出了使用两种优化方法之后 *kernel*1，*kernel*2 和 *kernel*3 的带宽利用率，其中 Loop tiling 为循环分块优化，Processing discrete memory access 为离散访存优化。*kernel*4 由于在使用主从核协同规约之后主核与主存之间的数据传输量无法由测试得到，因此主从核协同规约的优化效果在 4.4 节讨论。



(a)*kernel*1



(b)*kernel*2



(c)*kernel*3

图 9 kernel 在不同测试用例下的带宽利用率

Fig. 9 Bandwidth utilization of kernel in different test cases

图 9(a)给出了 *kernel*1 的带宽利用率。当只有循环分块优化时，无法直接通过 DMA 从主存中读取数组 *coords_center* 的数据，带宽利用率为 8%。在预处理数组 *coords_center* 之后，*kernel*1 所有的数据传输操作都通过 DMA 来完成，并且单次 DMA 操作的数据量都大于 1024 字节，因此带宽利用率达到 80%左右。

图 9(b)给出了 *kernel*2 的带宽利用率。对于硅扩展体系，由于原子数少，当只有循环分块优化时通过 DMA 传输的数据量占比小，因此只有循环分块优化时，硅体系 *kernel*2 的带宽利用率为 2.5%，而 RBD 的带宽利用率为 32%。使用离散访存优化之后，*kernel*2 中的数据传输操作均由 DMA 完成，但是由于数据传输部分和计算部分无法重叠，带宽利用率无法到达 80%，其中硅扩展体系的带宽利用率在 40%左右，RBD 数据传输部分占比更高，带宽利用率为 51%。

图 9(c)给出了 *kernel*3 的带宽利用率。*kernel*3 中每次通过 DMA 读取数组 *column_index_hamiltonian* 的数据量为算法 2 中(end-start)的值，当基组数增大时，(end-start)的值增大，同时小于 1024 字节的比例降低，因此 *kernel*3 的带宽利用率会随着基组数的增大而提高。在离散访存优化之后，RBD 的带宽利用率达到 69%。

4.4 主从核协同规约

主从核协同规约可以解决从核组并行规约时的写冲突问题，图 10 给出了主从核协同规约与 OpenACC 规约子句的性能对比，横坐标为硅扩展体系的 10 个测试用例，纵坐标为执行一次 *kernel*4 所用的时钟周期数，主从核协同规约的效率是 OpenACC 规约子句的 2 倍左右。

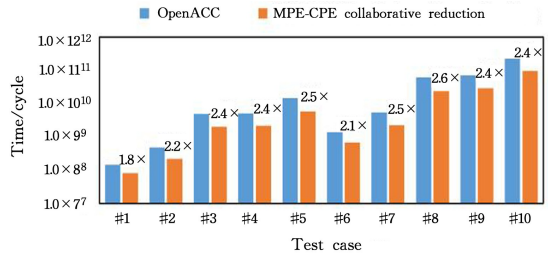
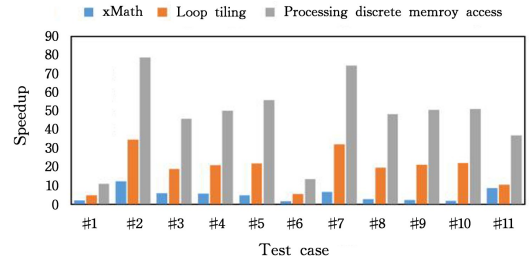


图 10 主从核协同规约与 OpenACC 规约子句的性能对比

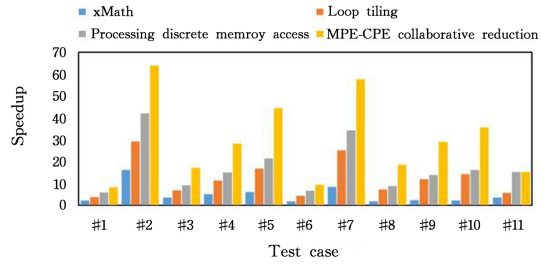
Fig. 10 Performance comparison between MPE-CPE collaborative reduction and reduction clause of OpenACC

4.5 加速比

本小节给出了计算一阶响应密度和一阶响应哈密顿矩阵部分的整体的加速比，结果如图 11 所示。



(a)一阶响应密度



(b)一阶响应哈密顿矩阵

图 11 优化之后的加速比

Fig. 11 Speedups after optimization

图 11(a)为计算一阶响应密度部分优化后的加速比，基准是不通过从核加速的主核程序的运行时间。图 11(a)使用的优化技术中，xMath 为调用 xMath 数学库加速矩阵乘法，Loop tiling 为循环分块优化，Processing discrete memory ac-

cess 为离散访存优化。

使用 xMath 优化后，11 个测试用例的加速比为 1.7×至 12.4×，其中 RBD 的加速比为 8.8×。对于硅扩展体系，原子数为 2 的 5 个测试用例中加速比最高的是测试用例 2，加速比为 12.4×，原子数为 16 的 5 个测试用例中加速比最高的是测试用例 7，加速比为 6.7×。测试用例 1 和测试用例 6 的加速比最低，分别为 2.1×和 1.7×，这两个测试用例运行时间短，从核启动开销无法忽略。原子数为 2 的 5 个测试用例的加速比总体上大于原子数为 16 的 5 个测试用例的加速比，因为随着原子数和基组数的增加，*kernel*3 的运行时间占比变大，同时矩阵乘法部分的运行时间占比降低。

使用循环分块优化后，11 个测试用例的加速比为 4.8×至 34.8×，其中 RBD 的加速比为 10.5×。对于硅扩展体系，原子数为 2 的 5 个测试用例中最高的是测试用例 2，为 34.8×；加速比最低的是测试用例 1，为 4.8×；测试用例 3—测试用例 5 的加速比为 21×左右。原子数为 16 的 5 个测试用例中，加速比最高的是测试用例 7，为 32.2×；加速比最低的是测试用例 6，为 5.6×；测试用例 8、测试用例 9 和测试用例 10 的加速比为 21×左右。

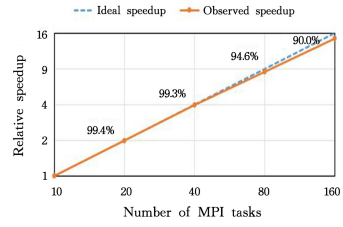
使用离散访存优化后，11 个测试用例的加速比为 11.1×至 78.6×，其中 RBD 的加速比为 37.1×。对于硅扩展体系，原子数为 2 的 5 个测试用例中加速比最高的是测试用例 2，为 78.6×；加速比最低的是测试用例 1，为 11.1×；测试用例 3、测试用例 4 和测试用例 5 的加速比分别为 45.9×，50.3×，55.9×。原子数为 16 的 5 个测试用例中加速比最高的是测试用例 7，为 74.4×；加速比最低的是测试用例 6，为 13.6×；测试用例 8、测试用例 9 和测试用例 10 的加速比为 50×左右。

图 11(b)给出了计算一阶响应密度部分优化后的加速比。其中 MPE-CPE collaborative reduction 为主从核协同规约优化。

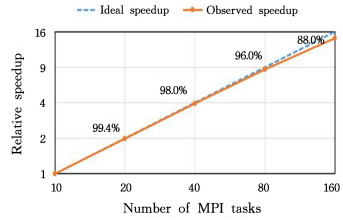
使用主从核协同规约优化后，11 个测试用例的加速比为 8.2×至 63.9×，其中 RBD 不存在规约时写冲突问题，加速比和离散访存优化之后的加速比相同，为 15.2×。对于硅扩展体系，原子数为 2 的 5 个测试用例中加速比最高的是测试用例 2，为 63.9×；加速比最低的是测试用例 1，为 8.2×；测试用例 3、测试用例 4 和测试用例 5 的加速比分别为 17.3×，28.4×，44.5×。原子数为 16 的 5 个测试用例中加速比最高的是测试用例 7，为 57.63×；加速比最低的是测试用例 6，为 9.5×；测试用例 8、测试用例 9 和测试用例 10 的加速比分别为 18.6×，29.1×，35.7×。

4.6 扩展性测试

图 12 给出了测试 $H(C2H4)_nH$ 分子时的强可扩展性，其中 $n=128$，一共 770 个原子。MPI 进程数从 10 增加到 160，括号中的标注为并行效率。图 12(a)给出了计算一阶响应密度部分的强可扩展性，当 MPI 进程数为 20 时，并行效率为 99.4%；当 MPI 进程数为 160 时，并行效率为 90.0%。图 12(b)给出了计算一阶响应哈密顿矩阵部分的强可扩展性，当 MPI 进程数为 20 时，并行效率为 99.4%；当 MPI 进程数为 160 时，并行效率为 88.0%。



(a)一阶响应密度

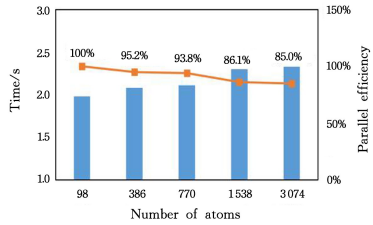


(b)一阶响应哈密顿矩阵

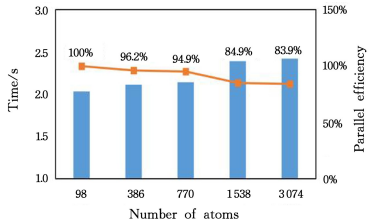
图 12 $H(C2H4)_nH$ 分子计算时间的强可扩展性

Fig. 12 Strong scalability of computation time of $H(C2H4)_nH$ molecule

图 13 给出了测试 $H(C2H4)_nH$ 分子时的弱可扩展性，原子数从 98 增加到 3074，对应的 MPI 进程数为 1，8，28，100，360。其中图 13(a)给出了计算一阶响应密度部分的弱可扩展性，图 13(b)给出了计算一阶响应哈密顿矩阵的弱可扩展性。当原子数达到 3 074、MPI 进程数达到 360 时，两个部分的并行效率分别达到 85.0%和 83.9%。



(a)一阶响应密度



(b)一阶响应哈密顿矩阵

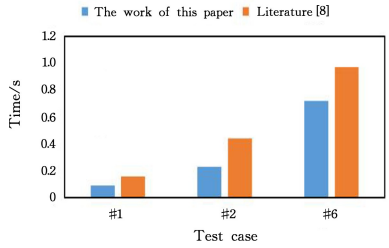
图 13 $H(C2H4)_nH$ 分子计算时间的弱可扩展性

Fig. 13 Weak scalability of computation time of $H(C2H4)_nH$ molecules

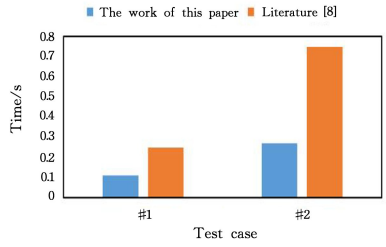
4.7 与相关工作中加速情况的对比

图 14 给出了本文方法与文献[8]中加速情况的对比，因为在模拟硅扩展体系时 MPI 进程数都是 6，所以本文选择直接对比执行时间。其中图 14(a)给出了计算一阶响应密度部分的时间对比，图 14(b)给出了计算一阶响应哈密顿矩阵部分的时间对比。可以看到本文所优化的程序的运行时间比文献[8]中所优化的程序的运行时间短，并且文献[8]在 DFPT 第一次迭代时将 *kernel*3 和 *kernel*4 中稀疏矩阵和稠密矩阵相互转换过程中的索引保存在内存中。在接下来的迭代中 *kernel*3 和

*kernel*4 可以直接读取内存中保存的索引从而减少访存量，因此表 1 中大部分例子会因为内存不足而无法得到结果。



(a)一阶响应密度



(b)一阶响应哈密顿矩阵

图 14 本文工作与文献[8]中所做工作的性能对比

Fig. 14 Performance comparison between the work of this paper and the work of literature[8]

结束语 本文对密度泛函微扰理论中一阶响应密度和一阶响应哈密顿矩阵的计算针对众核处理器体系结构进行了优化。在神威平台，优化后的程序较优化前性能有了较大的提升，同时拥有良好的强可扩展性和弱可扩展性。但是当基组数较小时，从核的启动开销以及较低的主存带宽利用率导致不能充分利用新一代神威异构众核处理器，接下来，我们将解决这两个问题。

参 考 文 献

[1] GONZE X. First-principles responses of solids to atomic displacements and homogeneous electric fields: Implementation of a conjugate-gradient algorithm [J]. Physical Review B, 1997, 55(16): 10337-10354.

[2] GONZE X, LEE C. Dynamical matrices, Born effective charges, dielectric permittivity tensors, and interatomic force constants from density-functional perturbation theory [J]. Physical Review B, 1997, 55(16): 10355-10368.

[3] VEITHEN M, GONZE X, GHOSEZ P. Nonlinear optical susceptibilities, Raman efficiencies, and electro-optic tensors from first-principles density functional perturbation theory [J]. Physical Review B, 2005, 71(12): 125107.

[4] SHANG H, CARBOGNO C, RINKE P, et al. Lattice dynamics calculations based on density-functional perturbation theory in real space [J]. Computer Physics Communications, 2017, 215: 26-46.

[5] BLUM V, GEHRKE R, HANKE F, et al. Ab initio molecular simulations with numeric atom-centered orbitals [J]. Computer Physics Communications, 2009, 180(11): 2175-2196.

[6] REN X G, RINKE P, BLUM V, et al. Resolution-of-identity approach to Hartree-Fock, hybrid density functionals, RPA, MP2 and GW with numeric atom-centered orbital basis functions [J]. New Journal of Physics, 2012, 14(5): 053020.

[7] HOHENBERG P, KOHN W. Inhomogeneous Electron Gas [J]. Physical Review B, 1964, 136(3B): B864-B871.

[8] SHANG H, LI F, ZHANG Y Q, et al. Accelerating all-electron ab initio simulation of raman spectra for biological systems [C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New York, NY, USA: ACM, 2021: 1-15.

[9] LEBEDEV V I. Quadratures on a sphere [J]. USSR Computational Mathematics and Mathematical Physics, 1976, 16(2): 10-24.

[10] HAVU V, BLUM V, HAVU P, et al. Efficient O(N) integration for all-electron electronic structure calculation using numeric basis functions [J]. Journal of Computational Physics, 2009, 228(22): 8367-8379.

[11] WOLF M, LAM M S. A data locality optimizing algorithm[C]// Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation. New York, NY, USA: ACM, 1991: 30-44.

[12] COLEMAN S, MCKINLEY K S. Tile size selection using cache organization and data layout[C]// Proceedings of the ACM SIGPLAN 1995 Conference on Programming Language Design and Implementation. New York, NY, USA: ACM, 1995: 279-290.

[13] MEHTA S, GARG R, TRIVEDI N, et al. Leveraging prefetching to boost performance of tiled codes[C]// Proceedings of the 2016 International Conference on Supercomputing. New York, NY, USA: ACM, 2016: 1-12.

[14] WANG X L, LIU W F, XUE W, et al. swSpTRSV: a fast sparse triangular solve with sparse level tile layout on sunway architectures[C]// Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York, NY, USA: ACM, 2018: 338-353.

[15] DUAN X H, GAO P, ZHANG T J, et al. Redesigning LAMMPS for peta-scale and hundred-billion-atom simulation on Sunway TaihuLight[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2018: 148-159.

[16] SHANG H, LI F, ZHANG Y Q, et al. Extreme-scale ab initio quantum raman spectra simulations on the leadership HPC system in China[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New York, NY, USA: ACM, 2021: 1-13.

LUO Haiwen, born in 1998, postgraduate, is a member of China Computer Federation. His main research interests include high performance computing and parallel software.

SHANG Honghui, born in 1984, Ph. D, associate professor. Her main research interests include the development of the first-priciples methods and their applications on the high-performance computer systems.

（责任编辑：何杨）