第 1 章

随机分批 Ewald 算法结合 LAMMPS 在鲲鹏处理器上的高性能实现

周颀[1]，江震[1]，张倩茹[2]，王一超[3]，徐振礼[1]
1. 上海交通大学数学科学学院和科学与工程计算教育部重点实验室
2. 杭州华为企业通信技术有限公司
3. 上海交通大学网络信息中心

1.1 应用简介：LAMMPS-RBE

本节介绍在鲲鹏处理器上自研开发的分子模拟高效算法——随机分批 Ewald（Random Batch Ewald，RBE）算法，该算法搭载于分子动力学模拟软件 LAMMPS（Large-scale Atomic/Molecular Massively Parallel Simulator），利用鲲鹏处理器搭载的消息传递接口（Message Passing Interface，MPI）工具及 ARM NEON 向量化指令集命令，实现了可进行高精度、高效率及高可扩展性模拟计算的优势。现有实验结果表明，在鲲鹏超算平台上，RBE 方法的单节点（128 个 MPI 计算核心）计算效率与主流粒子-粒子粒子-网格（Particle-Particle Particle-Mesh，PPPM）方法持平。而随着节点数增加，RBE 方法关于远场的计算效率优势逐渐显现，并在 16 节点时相较于 PPPM 方法具有一个数量级以上的提升。

分子动力学（Molecular Dynamics）[1~4]广泛应用于解决化学、生物医药、能源、材料和环境等领域的科学问题，是面向微纳尺度体系的计算机模拟最主流的工具之一，也是大规模科学计算中最为重要的方法之一。分子动力学通过在微观尺度计算粒子之间的相互作用力并应用牛顿运动方程确定所有粒子的速度和轨迹，统计系统平均获得相关物理量，如密度分布、结合能、扩散系数、迁移率等宏观物质的热力学和动力学

性质。因此，如何快速高效地运行分子动力学进而模拟宏观真实的体系，成为目前分子动力学领域最迫切的核心科学问题。经典的分子动力学模拟计算粒子的相互作用力，包括短程的键能相互作用、范德瓦尔斯相互作用和长程的静电相互作用，短程的键能相互作用由于其势函数快速收敛的性质，只需要计算一定距离之内的有限粒子的相互作用。然而静电相互作用由于库仑势的长程性，在大规模模拟过程中需要考虑全部模拟粒子的相互作用，以及周期边界条件下无穷镜像粒子对模拟粒子的作用。因此长程的静电相互作用是分子动力学模拟的瓶颈，往往会导致进行大规模快速计算变得十分困难。正因如此，经过近半个世纪的发展，尽管分子动力学取得了巨大进展，高性能的大规模并行计算（尤其是异构加速）仍具有极大的挑战性。

在当前主流的分子动力学模拟软件中，计算静电相互作用的算法通常可以分为如下两类。

（1）基于快速傅里叶变换（Fast Fourier Transform，FFT）的网格插值算法。此类算法的核心思想是基于 Ewald 方法，将库仑势分解为短程和长程相互作用两部分，其中长程部分在 Fourier 空间是指数衰减的，可通过选择合适的频率截断进行求和[5]。其整体计算复杂度为 $O(N\log N)$，在主流分子动力学模拟软件中的具体应用包括如下 3 种：

- LAMMPS 采用的 PPPM 方法[6]；
- GROMACS（GROningen MAchine for Chemical Simulations）采用的粒子−网格 Ewald 方法（Particle-Mesh Ewald，PME）[7]；
- AMBER（Assisted Model Building with Energy Refinement）采用的光滑粒子−网格 Ewald 方法（Smooth Particle-Mesh Ewald，SPME）[8]。

（2）基于多级展开的快速多极子算法（Fast Multipole Method，FMM）。这类算法通过多级展开处理粒子的相互作用，是被广泛认可的另一类主流算法，其提供了数学上渐近最优的 $O(N)$复杂度[9]。

然而随着模拟体系的增大，分子动力学需依托多 CPU 核或 GPU 核并行运算，其中，基于 FFT 的静电算法需要将粒子的位置映射到网格点处，再根据网格点的离散傅里叶变换值逆映射回原粒子位置，这需要核之间不停交换粒子分布信息，严重降低了模拟的并行效率。在某些情况下，模拟速度不仅不会随着并行核数增加而提升，反而可能下降，进而导致可扩展性进一步降低。同时，研究表明，FMM 和 PME 算法在时间开销和并行效率上表现相近，均面临可扩展性的瓶颈，这在一定程度上限制了更大规模的分子动力学模拟[10]。

为了突破主流静电算法在大规模分子动力学模拟中可扩展性上的瓶颈，并行优化一直是近些年引人瞩目的发展方向。20 世纪 90 年代至 21 世纪初，科研工作者一直致

力于在高性能计算机上基于多 CPU 核并行开发分子动力学模拟算法。现如今随着硬件性能的提升，算力更强的 GPU 逐渐被应用在大规模分子动力学模拟的计算中，基于 GPU 开发的分子动力学模拟算法也得到了广泛的关注。目前，大部分主流的分子动力学模拟软件都已开发了 CPU 或 GPU 并行版本，包括 AMBER[11]、LAMMPS[6]、NAMD（NAnoscale Molecular Dynamics）[12]和 GROMACS[13]等。

随着现代科学的进一步发展，对分子动力学模拟体系规模的要求将进一步加大。无论是新算法的发展还是具体的代码实现，都需要同计算机的并行架构进行更深层次的结合才能更好地进行复杂系统的模拟。同时，近年来由于数据科学的发展，机器学习的方法和随机算法取得了长足进展，利用随机算法开发新算法以降低复杂度是现今分子动力学模拟的重要方向。如基于蒙特卡罗方法或随机梯度下降法的小批量思想的设计随机算法[14]使计算复杂度大大降低，同时结果的正确性也在平均、统计意义下得以保证。

最近，研究人员开展了一种新型的随机分批 Ewald 算法加速全原子体系的分子动力学模拟，并且证明了该新方法可实现高扩展的并行效果。RBE 很好地解决了通信受限问题，提供了 $O(N)$复杂度，并充分利用当前并行计算的多核架构，实现了良好的可扩展性。同时，RBE 能够准确再现一系列基准算例的结构和动力学信息，包括纯水系统、微相分离电解质和蛋白质溶液，证明了算法的高精度。基于这些优点，RBE 算法有望成为下一代分子动力学引擎的核心算法[15~16]。

本章着重描述 RBE 算法如何移植至鲲鹏架构超算的 LAMMPS 上，形成可供高性能分子动力学模拟的 LAMMPS-RBE。基于算法设计中傅里叶空间计算的特性（详见 1.3 节），可采用负载均衡的 MPI 多核并行计算以保持其良好的强/弱可扩展性（详见 1.5 节），并通过 NEON 向量化指令操作进行傅里叶空间力计算的加速（详见 1.6 节），同时对于实空间力的计算采取掩码查表的方式进一步加速（详见 1.7 节），取得的结果和结论将在 1.8 节中分析讨论。

1.2 研发团队简介：上海交通大学快速算法与高性能计算实验室

上海交通大学快速算法与高性能计算实验室致力于计算物理、计算化学和大数据科学等领域的快速算法研究。团队研究课题包括分子动力学和蒙特卡罗模拟（Monte Carlo Simulation）、高性能计算和偏微分方程数值算法等，受到国家自然科学基金、上海市科委、中国科学院先导专项和其他专项的经费支持。

面对大规模分子模拟的长程计算瓶颈、可扩展性瓶颈及缺少跨尺度研究的现状，团队开发并维护了适用于 CPU 的 LAMMPS-RBE 和 CPU-GPU 异构加速的基于 SaaS（Softuare as a Service，软件即服务）技术的自研高性能分子动力学软件云计算平台——随机分批动力学模拟（Random Batch Dynamics Simulation，RBMD）软件。LAMMPS-RBE 现已开发了 Intel X86 版本及鲲鹏架构版本，极大程度上解决了长程计算瓶颈和可扩展性瓶颈，并在大规模全机测试中表现出良好的计算效率。RBMD 软件则有效提升了大规模粒子模拟中的软硬件协同加速，充分发挥了超级计算机的强劲算力。同时，RBMD 对比主流分子动力学模拟软件 LAMMPS，在相同 GPU 硬件环境下，长程力计算峰值加速比接近 100 倍。LAMMPS-RBE 与 RBMD 赋能新材料、新能源、新型储能、智能制造等关键领域，促进数字经济与实体经济深度融合，助力科研与工业创新。

1.3　分子模拟理论与算法设计

分子动力学[6]是许多领域（如化学物理、材料科学和生物物理）用于模拟纳米/微米尺度系统最流行的工具之一。分子动力学通过求解牛顿第二定律的方程来确定所有原子的轨迹，并通过统计集合平均获得相关的物理量。每个原子所受的力是通过对势能函数在力场下取负梯度来获得的，力场由化学键、键角、扭转、范德瓦耳斯力（van der Waals force）和静电力等组成。尽管模型相对简单，分子动力学在计算成本上却极为昂贵，因为一个典型系统可能包含 10^5 到 10^7 个粒子，而模拟需要进行超过 10^9 步的牛顿方程积分。因此，分子动力学成为超级计算机的主要应用之一，随着计算机辅助药物设计和创新的能量存储设备纳米技术应用的扩展，其重要性日益凸显。

图 1-1 所示为确定构型下经典库仑粒子系统的势能表达式，它以及作用在各原子上的力在分子演化中至关重要。

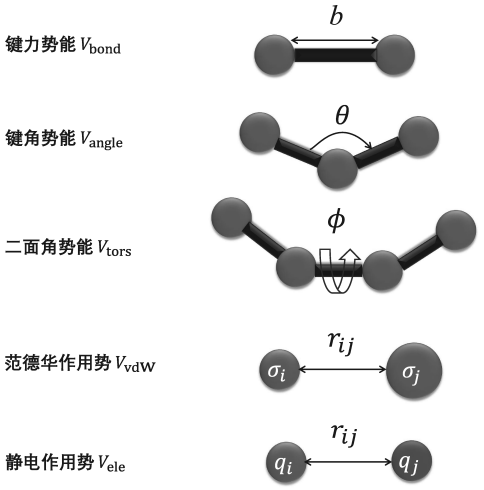

图 1-1　经典库仑粒子系统的势能表达式

在图 1-1 中，各贡献分别来源于键

力、键角、二面角、静电力和范德瓦耳斯力，所有参数均可由更精确的量子力学数据进行参数化或与实验观察结果进行拟合。

前两项使用调和函数建模双原子分子状态偏离平衡位置 b_0 及三原子分子偏离平衡角度 θ_0 所产生的能量，k_b、k_θ 为系数。

第三项使用带截断的傅里叶余弦级数建模二面角相对平衡状态 ϕ_0 的扭转能量贡献，其中 k_ϕ 为系数。

第四项来源于范德瓦耳斯力，一般用伦纳德-琼斯（Lennard-Jones，LJ）势（又称 12-6 势）进行建模，其中 ϵ_{ij} 为作用强度，σ_{ij} 给出两原子间的吸引距离和排斥距离。

最后一项为库仑静电势能，与点电荷间的距离 $r_{ij}=\left|\boldsymbol{r}_i-\boldsymbol{r}_j\right|$ 及电荷强度 q_i 和 q_j 相关，其中 ϵ_0 为介电常数，计算时通常使用约化单位，即将 $1/\left(4\pi\epsilon_0\right)$ 约化为 1。库仑作用部分同时具有奇异性和长程性，因而成为计算瓶颈的主要来源。

边界条件的建模同样对体系性质具有重要影响，通常的设置是三维周期边界条件，即在模拟区域的 3 个方向均存在无穷的周期镜像复制，从而减小有限体积效应带来的非物理效应。除去库仑相互作用外，其余贡献均具有快速衰减性质或有限作用范围，因而可以在实空间内通过有限截断进行高效计算，而库仑作用将通过 Ewald 方法分解为两部分：

$$\frac{1}{r}=\frac{\operatorname{erfc}(\sqrt{\alpha}r)}{r}+\frac{\operatorname{erf}(\sqrt{\alpha}r)}{r} \tag{1-1}$$

这里 $\operatorname{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x \mathrm{e}^{-t^2}\mathrm{d}t$ 为误差函数，$\operatorname{erfc}(x)=1-\operatorname{erf}(x)$ 为误差补函数，α 为分解参数，该参数在对应精度下控制了长短程计算消耗。该式将库仑作用分解为快速衰减的短程部分和光滑的长程部分，前者与其他贡献一同在实空间内高效求和计算，后者则利用傅里叶变换插值到傅里叶网格上，在傅里叶空间中使用高效求和方法计算。具体而言，根据上述 Ewald 方法分解可得库仑静电势能表达式为

$$V_{\mathrm{ele}}=\frac{1}{2}\sum_{\boldsymbol{n}}{}'\sum_{i,j}q_iq_j\frac{\operatorname{erfc}(\sqrt{\alpha}\left|r_{ij}+nL\right|)}{\left|r_{ij}+nL\right|}+\frac{2\pi}{V}\sum_{k\neq 0}\frac{1}{k^2}\left|\rho(\boldsymbol{k})\right|^2\mathrm{e}^{-k^2/4\alpha}-\sqrt{\frac{\alpha}{\pi}}\sum_{i=1}^{N}q_i^2 \tag{1-2}$$

其中 $n\in\mathbb{Z}^3$ 表示周期镜像，求和上的“$'$”记号表示不包括粒子自身 $i=j,\boldsymbol{n}=0$ 的贡献，$V=L^3$ 表示模拟长方体盒子体积，L 为边长，$k=2\pi m/L$ 代表全体傅里叶空间网格，$m\in\mathbb{Z}^3$，$\rho(\boldsymbol{k}):=\sum_{j=1}^{N}q_j\mathrm{e}^{\mathrm{i}\boldsymbol{k}\cdot\boldsymbol{r}_j}$ 称作结构因子。上述 3 项对应实空间求和、傅里叶空间求和及消除自能项贡献。由此，可得各粒子所受静电力表达式为静电势能的负梯度，即

$$\boldsymbol{F}_i^{\text{ele}} = -q_i \sum_{j,n}{}' q_j G(|r_{ij}+nL|)\frac{r_{ij}+nL}{r_{ij}+nL} - \sum_{k\neq 0}\frac{4\pi q_i \boldsymbol{k}}{Vk^2}\mathrm{e}^{-k^2/(4\alpha)}\,\mathrm{Im}(\mathrm{e}^{-ik\cdot r_i}\rho(\boldsymbol{k})) \tag{1-3}$$
$$:= \boldsymbol{F}_i^{\text{ele,short}} + \boldsymbol{F}_i^{\text{ele,long}}$$

其中

$$G(r) := \frac{\mathrm{erfc}(\sqrt{\alpha}r)}{r^2} + \frac{2\sqrt{\alpha}\mathrm{e}^{-\alpha r^2}}{\sqrt{\pi}r} \tag{1-4}$$

有关分子模拟势函数建模、库仑静电表达式及更多数学和物理上的解释可在分子模拟理论的图书和文章中获得，如参考文献[17]。

图 1-2 所示为在 LAMMPS 中进行分子动力学模拟的流程，用户可通过输入脚本（以下简称为“.in 文件”）设置本次模拟的时间步长、计算方式、系综、输出物理量等指令，具体可参考 LAMMPS 官网用户指南的“5. Commands”部分进行设置。

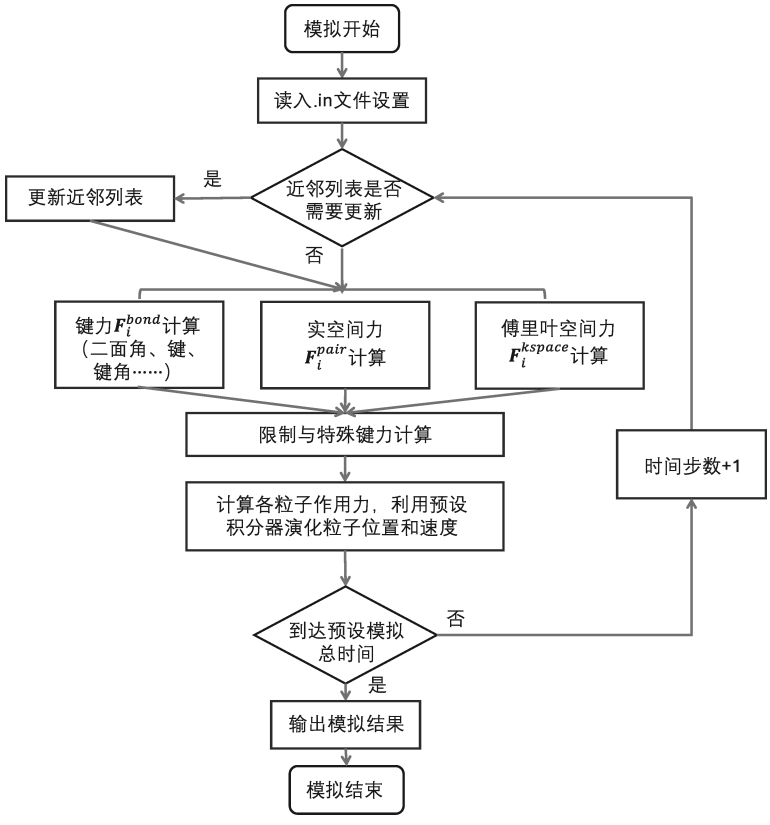

图 1-2　在 LAMMPS 中进行分子动力学模拟的流程

相比于 PPPM 等基于网格的快速求和方法，RBE 算法最显著的特点是利用小批量随机近似傅里叶空间力来同时提升计算效率和可扩展性。对于静电力的傅里叶空间求和部分 $F_i^{\mathrm{ele,long}}$，RBE 算法使用 P 个独立同分布高斯样本计算：

$$\tilde{F}_i^{\mathrm{ele,long}} \coloneqq -\sum_{l=1}^{P}\frac{S}{P}\frac{4\pi q_i \boldsymbol{k}_l}{Vk_l^2}\mathrm{Im}(\mathrm{e}^{-i\boldsymbol{k}_l\cdot r_i}\rho(\boldsymbol{k}_l)),\quad \boldsymbol{k}_1,\ldots,\boldsymbol{k}_P \sim \mathrm{e}^{-k^2/4\alpha} \tag{1-5}$$

其中 S 为归一化常数。关于离散高斯样本 $\boldsymbol{k}_l$ 的生成，RBE 算法应用 Metropolis 算法进行采样，有关该算法的理论和更多细节见参考文献[18]。注意到高斯分布的三维可分性，因而可以在各维度上独立地以 $\rho(m)$ 进行采样，其中概率分布表示为

$$\rho(m) \sim H^{-1}\mathrm{e}^{-(2\pi m/L)^2/4\alpha},\quad m\in\mathbb{Z} \tag{1-6}$$

这里 H 为单维度的归一化因子，对应可得傅里叶格点为 $\boldsymbol{k}=(k_1,k_2,k_3)=2\pi m/L$，并且全零向量将被丢弃。理论结果表明，在体系密度固定的前提下，RBE 算法给出方差可控的无偏近似力，并且可以实现最优的 $O(PN)$ 线性复杂度。更为重要的一点是，在大规模并行计算中，相比于 PPPM 方法，RBE 算法避免了网格间密集的六轮通信（向前 FFT 及向后 FFT 过程各三轮），只需对各 MPI 上计算的结构因子做一次全局归约即可完成通信，并且 MPI 多核设计同样可以加速上述采样过程，具体将在 1.5 节中讨论。

1.4 软件编译技巧

LAMMPS 适用于各环境下的大规模分子动力学模拟，其中包括各部分计算、并行、通信、积分器、特殊作用（限制）等大量模块，需要利用 CMake 工具生成 Makefile，再利用 make 工具批处理上述大量模块的编译工作，其编译对环境要求也相对较高。相比于 Intel X86 超算平台，鲲鹏处理器具有不同的原生编译器，包括数学库、MPI 库等，因而在进行开发的时候需要使用不同的编译技巧。以下将具体阐述 RBE 算法在鲲鹏处理器上的编译过程，并提供相应指令以方便读者直观理解、操作和做进一步的开发。

（1）申请节点，然后开始编译。注意：不要在登录节点编译。

```
srun -p arm128c256g --exclusive --pty /bin/bash
#注意，与其他超算平台节点不同，鲲鹏处理器上使用的节点为 ARM 架构下的 arm128c256g
```

（2）定位至 LAMMPS，建立一个 build 文件夹，用以存放 CMake 生成的 Makefile 以及 make 后的二进制文件。

```
rm -rf build && mkdir build
#删除 build 文件夹，并新建一个 build 文件夹
```

（3）加载编译所需的 3 个模块。

```
module load bisheng
```

```
#加载鲲鹏处理器毕昇编译器模块

module load hypermpi
#加载鲲鹏处理器的 MPI 模块，注意 Intel X86 中编译 LAMMPS 一般加载 oneapi

module load cmake
#加载 CMake 模块
```

（4）在使用 CMake 前，确保 cmake 文件夹中的 CMakeLists.txt 文件链接了鲲鹏数学库，以确保.cpp 文件中的三角函数等运算能被正确编译。具体需要确保以下两段语句在 CMakeLists.txt 文件中。

语句 1：

```
#链接 kml 库
include_directories("/usr/local/km l/include")
link_directories("/usr/loa cl/km l/lib")
```

语句 2：

```
#指定链接 kml 的依赖项 libksvml.so
target_link_libraries(lammps PRIVATE/usr/local/km l/lib/libksvml.so)
```

（5）定位到 build 文件夹下，使用 CMake 工具生成 Makefile。

```
cmake -D PKG OMP=no -D PKG MOLECULE=yes -D PKG MANYBODY=yes -D PKG KSPACE=yes
-D PKG_RIGID=yes ../cmake
```

（6）使用 make 工具批处理编译 LAMMPS，并生成二进制文件。注意：对算法开发者而言，每次进行计算模块、其他.cpp 或头文件的修改，均须进行一次 make 使得修改被成功编译进二进制文件中，再进行算例测试。

```
make -j
```

（7）成功编译后，build 文件夹中应生成名为 lmp 的文件，代表成功完成了编译，此时可退出向平台申请的节点。

```
exit
```

有关 LAMMPS 编译的更多信息，具体可参考 LAMMPS 官网“Programmer Guide”部分的“3. Build LAMMPS”。接下来将具体展示 RBE 算法在鲲鹏处理器上的进程级并行方法、数据级并行方法以及额外的实空间优化方法，以方便读者理解鲲鹏处理器上的 LAMMPS 编译过程与 RBE 力计算的修改方法，读者可参考以下过程开展进一步的自研算法设计与搭载。

1.5　进程级并行：MPI 并行编程

分子动力学模拟体系往往较大，因而多核并行计算经常被考虑使用在分子动力学计算过程中以满足计算速度和长时模拟的需求，同时可扩展性瓶颈也需要在算法设计和算法应用中被仔细考虑。在多 MPI 框架下，无网格的 RBE 算法相较于通信密集的 PPPM 等网格算法，能更好地降低通信消耗，从而期望能在多核并行计算中拥有更为

卓越的计算效率和可扩展性。鲲鹏架构中的 LAMMPS 同样支持 MPI 多核并行计算，并且关于实空间力和傅里叶空间力的计算资源分配均有其独有特点。总体而言，LAMMPS 基于粒子所在物理空间进行对应划分，并将所在子空间的粒子信息分配到不同的 MPI 进程中以实现并行计算。针对均相体系，由于体系密度的近似均一性，均匀划分网格（.in 文件中指令为“comm_style brick”或默认）即可实现相当出色的负载均衡分配，而针对非均相体系，则需要针对体系团簇特点做额外的空间划分及负载均衡操作，具体可参考 LAMMPS 官网“Programmer Guide”部分的“4.4 Parallel algorithms”进行设置。这里仅以均相体系为例说明 RBE 算法如何实现 MPI 多核并行计算。

针对实空间力的计算，由于其近邻列表与 MPI 物理空间划分具有高度一致性，因而在 LAMMPS 中已隐式自动地为短程部分计算提供了多核并行计算实现。LAMMPS-RBE 同样为实空间力的计算设计了掩码查表的快速算法，在 1.7 节中有详细讨论。而针对傅里叶空间力的计算，对傅里叶格点的求和具有全局性，即需要全体粒子的信息，因而该部分需要更为细致地在算法文件中设置 MPI 通信交互指令。在 MPI 并行框架下，RBE 中设计了多核采样及无干扰广播的过程[19]，如图 1-3 所示。

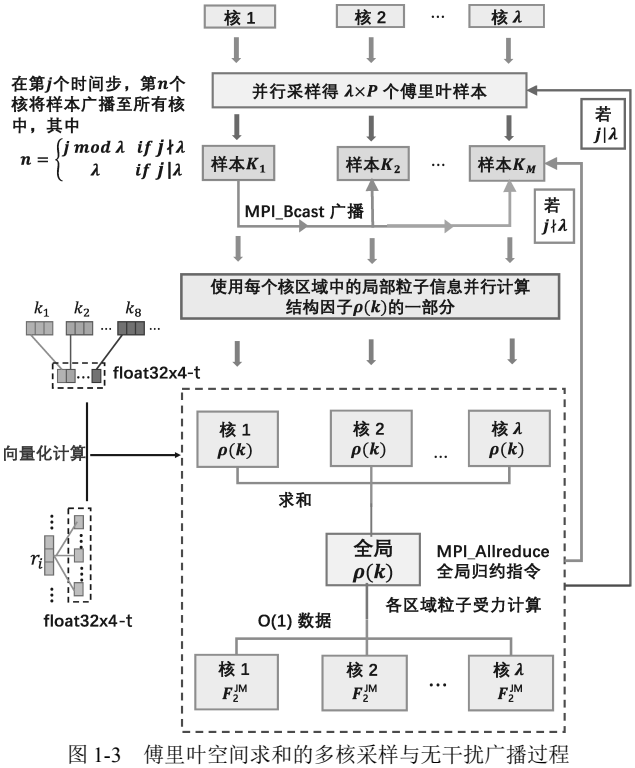

图 1-3 傅里叶空间求和的多核采样与无干扰广播过程

在申请 λ 个核参与并行计算的前提下，可先在全体核上独立并行采样所需样本（供接下来 λ 个时间步使用）；然后第一个核将其上所有样本广播至每个 MPI，执行结构因子和力的计算并演化，与此同时第二个核开始广播样本；在第二个时间步则无须采样，直接利用上一步第二个核广播的样本执行计算，同时第三个核开始广播样本。以此类推，直至 λ 个时间步演化完成，开始下一个 λ 时间步过程。

该过程容易实现，并能极大提高可扩展性与并行效率。而针对结构因子的计算，各 MPI 独立并行计算其上粒子的结构因子信息，再通过一次全局归约即可得到正确的结构因子，此后各 MPI 均可使用该信息并行进行对应粒子长程作用力的计算，相比于 PPPM 等算法的多轮通信消耗具有显著优势。以下将提供相应代码和说明，以方便读者直观理解、操作和做进一步的开发。如需阅读完整代码，可参考平台 LAMMPS 中的“/src/kspace/rbe.cpp”。

（1）链入 MPI 指令库。

```
#include <mpi.h>
```

（2）在傅里叶空间计算函数内，借助 Metropolis-Hasting 或其他采样算法，多核并行采样及无干扰广播。

```
/*该部分在各 MPI 上同时采样，并依设计执行广播过程*/
//该函数用于计算长程力，输入参数分别用于控制能量计算和维里计算
void Rbe::compute(int eflag, int vflag)
{
    ...
    MPI_Barrier(MPI_COMM_WORLD); //保持所有 MPI 可同时进行计算
    int this_rank = (Step % RankID); //待广播样本的 MPI 编号
    if (((Step % RankID)==0)&&(me< RankID))
    {
        ...//仅在对应时间步上进行全体并行采样操作
    }

    if (me == this_rank)//提取广播样本的 MPI 采样信息
    {
        for (int i = 0; i < P; i++)
        {
            K[i][0] = K_Sample[i][0];
            K[i][1] = K_Sample[i][1];
            K[i][2] = K_Sample[i][2];
        }
    }
    MPI_Bcast((float*)K, 3 * P, MPI_FLOAT, this_rank, MPI_COMM_WORLD);
    //广播
    Step++;//计算步数更新

    ...
}
```

（3）在各 MPI 上，独立计算结构因子 $\rho(\boldsymbol{k}_t)$ 的对应部分。

（4）全局归约各 MPI 核心的结构因子。

```
MPI_Allreduce ((float*)Rho , (float*)Rho_All , 2* P, MPI_FLOAT, MPI_SUM ,
MPI_COMM_WORLD);
```

（5）在各 MPI 调用核心上，并行计算各粒子所受的长程力。尽管各 MPI 执行速度不一定相同，但可以在对应调用核心计算完后立即累加，得出各粒子所受的长程力。

在使用 LAMMPS 进行模拟时，.in 文件中关于 RBE 长程作用的计算可做如下参考设置：

```
kspace_style rbe 1e-4 256
# 傅里叶空间计算使用 RBE 算法
# 对应于实空间截断设置，利用误差估计计算出 Ewald 分解参数 alpha，使误差小于 1e-4
# RBE 样本数为 256 个
# 并行计算的核数量保持与 MPI 调用核数一致
```

对应作业提交脚本（slurm 文件）的运行设置如下，更多平台的作业提交说明可参见上海交通大学 ARM 节点上的使用文档。

```
mpirun -np 128 ./lmp -i ./in.rbe
#采用 128 个 MPI 并行计算
#./lmp 为 LAMMPS 中进行 cmake/make 后的二进制文件，这里展示时位于当前目录下，编写脚本时需替换为 build 文件夹下的正确路径
#./in.rbe 为.in 文件，同样在编写时需替换为正确路径
```

使用 RBE 方法同样可结合 OpenMP 做线程级别的进一步并行化加速，关于 LAMMPS 中 MPI 和 OpenMP 并行计算的更多信息可参考 LAMMPS 官网“Programmer Guide”部分的“4.4 Parallel algorithms”。

1.6 数据级并行：NEON 向量化

在 RBE 算法中，傅里叶空间求和将利用无网格的随机分批近似代替，以求获得更高的计算效率和并行效率。通过 $F_i^{\mathrm{ele,long}}$ 的数学表达式不难发现，各样本 $\boldsymbol{k}_t$ 对结构因子等力计算的组成部分具有完全一致的形式，因而向量化和单指令多数据流（Single-Instruction Multiple-Data Stream，SIMD）成为加速傅里叶空间计算的重要工具。鲲鹏处理器已支持 ARM NEON 向量化计算，同时鲲鹏数学库支持对应数据格式的单指令多数据运算，这里主要是指支持结构因子的复杂三角函数计算。以下将具体阐述如何通过 RBE 算法利用 NEON 向量化操作及鲲鹏数学库实现傅里叶空间计算加速，并展示相应代码，完整代码参考平台 LAMMPS 中的“/src/kspace/rbe.cpp”。

（1）链入 NEON 向量化指令集和鲲鹏数学库。

```
//链入 NEON 向量化指令集
#include <arm_neon.h>

//链入鲲鹏数学库
```

```
#include "/usr/local/kml/include/kevml.h"
#include "/usr/local/kml/include/km.h"
```

（2）在傅里叶空间计算函数内，利用 Metropolis-Hasting 或其他采样算法获得样本并存储。如果不启用 MPI 并行计算，则每演化时间步均需以.in 文件设置抽取对应样本数。若启用 MPI 并行计算，则可利用多核实现高性能抽样方法，详见 1.5 节步骤（2）。

（3）基于 NEON 向量化对应长度和格式，对输入粒子坐标数组、电荷量数组及待输出力数组进行适宜分段，以保证每个向量读取到正确数据。

```
/*鲲鹏处理器上的 NEON 向量化指令针对单精度计算，长度最多为 4（即 128 位向量），因此需要将数组
按 4 个元素为一组进行划分*/
/*第 2 步中抽取的样本和其他可向量化操作的数组也以同样的方式进行切割*/
//该函数用于计算长程力，输入参数分别用于控制能量计算和维里计算
void Rbe::compute(int eflag, int vflag)
{
    ...
    float X[ int( ceil( ( nlocal + 0.0 ) / 4.0 ) ) * 4 ][3];
    float F[ int( ceil( ( nlocal + 0.0 ) / 4.0 ) ) * 4 ][3];
    float Q[ int( ceil( ( nlocal + 0.0 ) / 4.0 ) ) * 4 ];
    ...
}
```

（4）在 1.5 节的步骤（3）中，利用 NEON 向量化指令快速计算结构因子 $\rho(\boldsymbol{k}_t)$。

```
/*该部分利用 NEON 向量化指令和鲲鹏数学库中的三角函数计算结构因子*/
//该函数用于计算长程力，输入参数分别用于控制能量计算和维里计算
void Rbe::compute(int eflag, int vflag)
{
    ...
    //向量长度为 4，每 4 个数据进行一次下列计算，共需计算 P 个样本对应的结构因子
    for (int i = 0; i < P; i += 4)
    {
        float32x4_t Real, Imag, X0, X1, X2, qq, Cos, Sin, Moment, KXKX1, KXKX2,
mid0, mid1;
        /*float32x4_t 表示 4 个 32 位单精度浮点数形成的向量，可类比 Intel AVX128 中的数据
类型__mm128*/

        Real = Imag = vdupq_n_f32(0.0f);
        /*vdupq_n_f32()指令将 32 位单精度浮点数复制给向量 float32x4_t 中的每一个分量，这
个操作等价于给向量赋初值 0.0f*/
        KXKX0 = vld1q_f32(&KxKx0[i]);
        KXKX1 = vld1q_f32(&KxKx1[i]);
        KXKX2 = vld1q_f32(&KxKx2[i]);
        /*vld1q_f32()指令从对应地址中取出相邻的 4 个单精度浮点数，并存放到对应的 float32x4_t
中*/

        for (int j = 0; j < nlocal; j++) //nlocal 表示该 MPI 上分配的粒子数
        {
            ...//读取当前粒子位置与电荷信息

            mid0 = vmulq_f32(KXKX0, X0);
            mid1 = vfmaq_f32(mid0, KXKX1, X1);
```

```
            Moment = vfmaq_f32(mid1, KXKX2, X2);
            /*vmulq_f32()及 vfmaq_f32()指令分别代表向量间的乘法和混合乘加，与 Intel AVX
向量化不同，NEON 向量化尚不支持重载后的四则运算符进行运算，因而需要此种内部指令进行四则运算
*/
            /*以上 3 句表示 Moment=KXKX0*X0+KXKX1*X1+KXKX2*X2 ，也即完成三角函数内向量
点积的计算*/

            svml128_sincos_f32(Moment, &Sin, &Cos);
            /*svml128_sincos_f32(a,&b,&c)是鲲鹏数学库中支持 float32x4_t 的三角函数,该
句代码将 sin(a)存入 b 地址处，将 cos(a)存入 c 地址处*/

            Real = vfmaq_f32(Real, qq, Cos);
            Imag = vfmaq_f32(Imag, qq, Sin);
            //结构因子实虚部计算
        }
        vst1q_f32(&Rho_Cos[i], Real);
        vst1q_f32(&Rho_Sin[i], Imag);
        //vst1q_f32(&a,b)将 float32x4_t 的向量 b 存入以地址 a 开始的相邻 4 个位置中
    }
    ...
}
```

（5）在 1.5 节的步骤（5）中，利用全局归约后的结构因子，使用 NEON 向量化指令快速计算各粒子所受的长程力。

```
/*该部分利用 NEON 向量化指令和鲲鹏数学库中的三角函数计算长程力*/
//该函数用于计算长程力，输入参数分别用于控制能量计算和维里计算
void Rbe::compute(int eflag, int vflag)
{
    ...
    //力表达式中与结构因子无关的系数因子
    double MIDTERM = -4 * pi * (S / (P + 0.00)) * qqrd2e / (V);

    for (int i = 0; i < nlocal; i+=4)
    {
        float X1[4], X2[4], X3[4];
        for (int j = 0; j < 4; j++)//读取该次循环待计算的 4 个粒子的位置信息
        {
            X1[j] = X[i + j][0];
            X2[j] = X[i + j][1];
            X3[j] = X[i + j][2];
        }
        ...//获取样本和电荷信息
        for (int j = 0; j < P; j++)
        {
            Kx = vdupq_n_f32(KxKx0[j]);
            Ky = vdupq_n_f32(KxKx1[j]);
            Kz = vdupq_n_f32(KxKx2[j]);

            mmid0 = vmulq_f32(Kx, X_0);
            mmid1 = vfmaq_f32(mmid0, Ky, X_1);
            moment = vfmaq_f32(mmid1, Kz, X_2);
            moment = vnegq_f32(moment);
            /*vnegq_f32()返回 float32x4_t 各分量的相反数，计算力表达式 Im()中结构因子前
的复指数*/
```

```
            svml128_sincos_f32(moment, &Sin, &Cos);

            midterm_128 = vdupq_n_f32(MIDTERM);
            Rho_All_0 = vdupq_n_f32(Rho_All[j][0]);
            Rho_All_1 = vdupq_n_f32(Rho_All[j][1]);
            //4 个粒子拥有相同的系数因子，并在循环内读取各样本对应的结构因子

            float32x4_t imag_cos, imag_temp;
            imag_cos = vmulq_f32(Cos, Rho_All_1);
            imag_temp = vfmaq_f32(imag_cos, Sin, Rho_All_0);
            Imag = vmulq_f32(imag_temp, midterm_128);

            midterm_0 = vdupq_n_f32(midterm[j][0]);
            midterm_1 = vdupq_n_f32(midterm[j][1]);
            midterm_2 = vdupq_n_f32(midterm[j][2]);
            //midterm[P][3]为采样结束后计算的 k/|kl^2 的 3 个分量，共 P 个样本，参见力表达式

            ...//计算各 MPI 上力的贡献
        }
        ...//将力累加到全局变量上
    }
    ...
}
```

在使用 LAMMPS 进行分子动力学模拟时，.in 文件中关于 RBE 长程作用计算的参考设置与 1.5 节展示的一致。有关 ARM NEON 架构的介绍可访问 NEON 的相关网站进行查阅，更多 NEON 向量运算指令或鲲鹏数学库函数可访问 NEON 内部指令集和鲲鹏数学库开发指南进行查阅。

1.7　实空间优化方法

实空间力包括 LJ 力及通过 Ewald 方法分解后的短程部分力，在给定参数下由粒子间的距离 r_{ij} 决定。对直接截断内的粒子对进行计算尽管只需要线性复杂度消耗，但表达式中涉及多项求和及复杂函数运算，这将在短程 $O(N)$复杂度前引入过大的前因子而降低计算效率。鲲鹏处理器上的 LAMMPS 支持使用掩码查表方法加速实空间短程部分求和计算[19]，方法示意如图 1-4 所示。

图 1-4 表明，掩码查表方法将截断内区域分为两个部分——核区域（Core part）和壳区域（Shell part），并分别采用不同的计算方式加速计算以达到设定精度。

在核区域，使用极少项（一般不超过 5 项即可满足误差小于 1e-4 的精度要求）泰勒展开计算短程部分力，其中泰勒系数可被事先解析计算出，因而只涉及极少量的多项式运算，同时避免了计算误差补函数等复杂函数和多项运算。

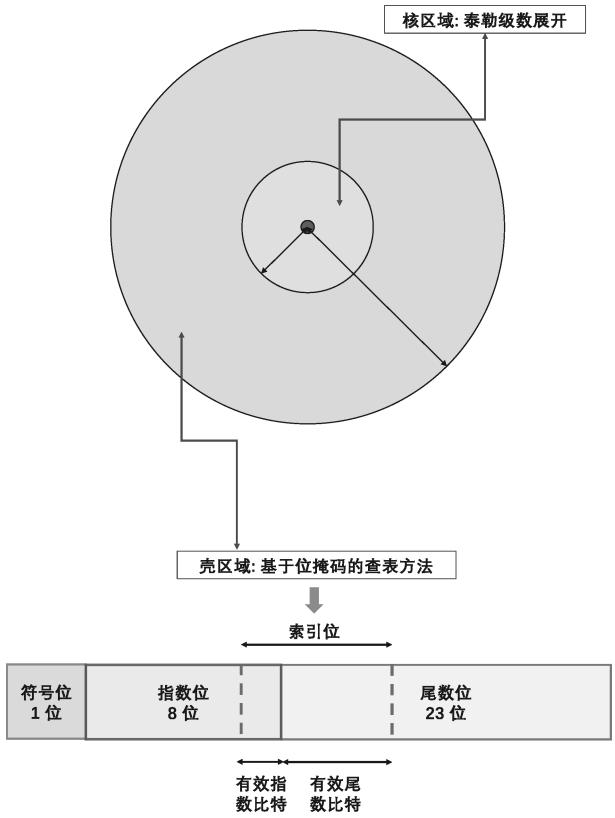

图 1-4 实空间短程部分求和的掩码查表方法示意

在壳区域，则采用基于位掩码的表格技术进行处理，尽可能以 CPU 计算开销低的二进制移位操作代替四则运算。需要注意的是，单精度浮点数的二进制表示通常包含 1 个符号位、8 个指数位和 23 位尾数（双精度浮点数则对应为 1 个符号位、11 个指数位和 52 位尾数）。从指数中取出若干低位（以 Bexp 表示），并从尾数中取出一些高位（以 Bman 表示），总的表格可记录 2Bexp +Bman 个数据，这些数据将在迭代步运行前计算好并存入表格。在表格中，使用线性插值法来近似获取相邻点之间的数据，即可完成壳区域的短程力计算，该方法在保证精度的同时避免了计算复杂函数和多项运算。该掩码查表过程对应的部分.cpp 代码如下所示，详细代码可在平台上的 LAMMPS 软件“/src”或“/src/kspace”文件夹中找到。

```
/*该部分代码位于/src/kspace/pair_lj_cut_coul_long.cpp 中，用于计算库仑系统+LJ 势函数的短程相互作用*/
/*掩码制表过程位于/src/pair.cpp 中，基于 Ewald 分解和高斯与 SOG 分解（见参考文献[19]）的制表已完成，如使用其他作用核函数或长程核分裂形式，可在 pair.cpp 中搜索"rbosgflag"，在对应位置进行制表核函数的类似修改，其余逻辑一致*/
```

```
...
//该函数用于计算短程力，输入参数分别用于控制能量计算和维里计算
void PairLJCutCoulLong::compute(int eflag, int vflag)
{

    ...//用于定义各变量及赋值
    for (ii = 0; ii < inum; ii++) //中心粒子
    {
        ...//用于定义与粒子 ii 有关的各变量及赋值
        for (jj = 0; jj < jnum; jj++) //当前粒子
        {
            ...//用于定义与粒子 jj 有关的各变量及赋值，计算两粒子间的距离，rsq 变量表示距离
的平方
            if (rsq < cutsq[itype][jtype])
            {
                r2inv = 1.0/rsq;
                if (rsq < cut_coulsq)
                {
                    //判断当前粒子是否位于中心粒子核区域内
                    if (!ncoutablebits || rsq <= tabinnersq)
                    {
                        ...//5 项泰勒展开快速计算核区域相互作用
                    }
                    else //当前粒子位于中心粒子壳区域内
                    {
                        //掩码查表法计算壳区域相互作用
                        union_int_float_t rsq_lookup;
                        rsq_lookup.f = rsq;
                        //使用 Bitmask，仅保留用于查表的有效二进制位 B_exp+B_man
                        itable = rsq_lookup.i & ncoutmask;
                        //右移操作使 itable 可解读为对应表格的十进制编号
                        itable >>= ncoutshiftbits;
                        //线性插值因子计算
                        fraction = (rsq_lookup.f - rtable[itable]) * drtable[itable];
                        table = ftable[itable] + fraction*dftable[itable];
                        forcecoul = qtmp*q[jj] * table;//相互作用计算
                        ...
                    }
                }
                else forcecoul = 0.0;//当前粒子位于截断范围外，设定力为 0.0
                ...//依据 flag 设定值计算物理量（同样可以用掩码查表法）
            }
        }
    }
    ...
}
```

在使用 LAMMPS 进行分子动力学模拟时，一般将库仑相互作用和 LJ 作用的截断半径设为一致（通常为 8 至 12），.in 文件中关于短程作用的掩码查表计算可做如下参考设置：

```
pair_style lj/cut/coul/long 10.0 10.0
#实空间计算使用 pair_lj_cut_coul_long.cpp，LJ 半径与库仑半径均为 10.0 埃

pair_modify tabinner 2 table 12
#采用掩码查表计算，核内区域半径为根号 2，表格长度为 12（即 2^12=1024 个查表点）
```

该掩码查表技术在使用或开发时有两点需要说明。首先，表格长度的选择需要权衡准确性和计算速度要求，较大的表格尺寸能够提供更精确的力计算，但会占用更多内存，从而可能降低模拟的速度。其次，当核函数趋于奇异时，掩码查表部分的误差会增加，但通过设置核区域可以有效解决这一问题。有关掩码查表方法更深入的知识，读者可阅读参考文献[19]和[20]。

1.8 计算结果与计算效率

本节展示鲲鹏处理器 LAMMPS-RBE 上的计算结果与性能结果，用于对比的基准算法是 PPPM 方法，该方法目前是分子动力学模拟最主流的算法之一。用于分子动力学的测试模型为一个包含 24327 个水分子的 SPC/E 纯水系统，该系统具有立方体三维周期几何，边长为 90 埃。我们首先执行一个持续 50 ps 的平衡过程，随后是持续 1 ns 的数据采集过程，平均每 200 步（0.1 ns）记录一次数据。系统的平衡温度被设置为 298K，积分步长Δt 为 0.5 fs。粒子的初始速度满足麦克斯韦（Maxwell）分布。在平衡过程中，库仑力的短程部分和 LJ 相互作用都将使用 10 埃的截断半径直接计算，PPPM 的精度预设为 1e-4，并以此确定 PPPM 方法与 RBE 算法共用的 Ewald 分解参数 α。所展示的均为 NVT 系综下的模拟结果，并使用 SHAKE 算法约束氢键。LAMMPS-RBE 部署于“交我算”的鲲鹏超算上，超算平台搭载鲲鹏处理器（64 核），单节点双路配备 128 核（2.6 GHz）、256 GB（16 通道 DDR4-2933）内存、Infiniband 高速互联。测试规模从单节点（128 核）扩展至 16 节点（2048 核）。

首先要说明的是 RBE 算法的模拟准确性。通过模拟计算系统的几个重要物理性质，包括径向分布函数（Radial Distribution Function，RDF）、均方位移（Mean Square Displacement，MSD）和速度自相关函数（Velocity Autocorrelation Function，VACF），可以验证正则系综（NVT）下使用 RBE 的准确性结果。其中 RDF 刻画了全原子水体系的平衡结构，MSD 和 VACF 度量水体系的动力学性质。图 1-5 所示为使用 RBE 和 PPPM 模拟全原子水系统的精度比较。

图中展示了不同物理量的计算结果：（a）氧-氧，（b）氢-氢和（c）氧-氢原子对的 RDF；（d）每个水分子的平均能量；（e）原子的 MSD；（f）氧原子的 VACF。在计算中，我们分别使用了样本数 P 为 100、200 和 500 的 RBE 算法和 PPPM 算法，比较了不同算法得到的模拟结果。在小提琴图（d）中，每个小提琴内白色的点和黑色棒的两个端点分别表示均值和两个四分位数。图 1-5（d）的子图中展示了每个水分子平均能量的均值误差随样本数 P 变化的函数关系。

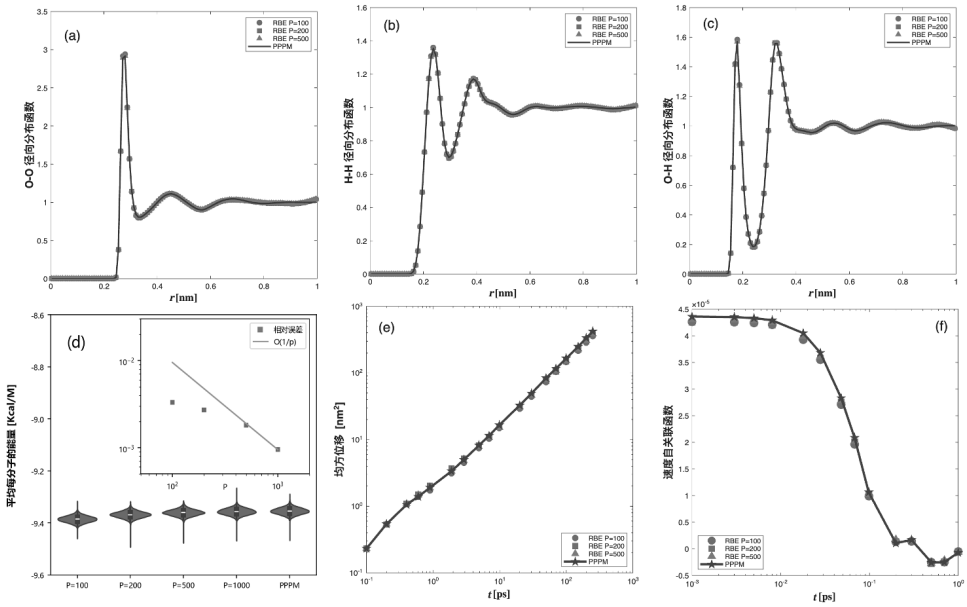

图 1-5　使用 RBE 和 PPPM 模拟全原子水系统的精度比较

图 1-5 分别展示了氧-氧（O-O）、氢-氢（H-H）和氧-氢（O-H）原子对的 RDF，每个水分子的平均能量，氧原子的 MSD 和氧原子的 VACF。RBE 和 PPPM 计算出的 3 个 RDF 在统计意义上是足够精准的。图 1-5（d）显示出正确的玻尔兹曼分布，进一步证明了搭载于鲲鹏超算的 RBE 算法的正确性。在图 1-5（d）中，总能量的收敛速度呈现出与 RBE 理论近似的 $O(1/P)$。图（e）和（f）分别展示了 MSD 和 VACF 的比较。RBE 和 PPPM 计算的结果在多个时间尺度上都是一致的，证明了当 $P \geqslant 200$ 时使用 RBE 算法能够较好地复现系统的动力学性质。RBE 同样支持其他常用系综（NPT、NVE……）的模拟[21~22]。

接下来展示 RBE 算法的计算效率和可扩展性。这里选用与精度计算相同的模拟体系和系统参数测试 RBE 算法在鲲鹏超算上的性能效率，通过执行 1000 步模拟来计算每步更新所需的平均 CPU 时间。CPU 性能的度量标准选用壁钟（Wall-Clock）时间。鲲鹏超算上 ARM 单节点配备 128 核（2.6 GHz）、256 GB（16 通道 DDR4-2933）内存、240 GB 本地硬盘，节点间采用 IB 高速互联，本测试包含单节点到最高 16 节点的并行效率结果（即核数从最少 128 核到最多 2048 核）。在展示测试结果前，给出强/弱可扩展性的定义与对应数学表达。强可扩展性为在保持系统粒子数不变的前提下改变所使用的 CPU 核数，进而分析时间的变化趋势。令 λ 是所使用的 CPU 核数，$T(\lambda)$是相应的运行时间，强可扩展性的数学定义为

$$\eta(\lambda)=\frac{\lambda\min}{\lambda}\cdot\frac{T\min}{T(\lambda)}$$

其中 $T_{\min}=T(\lambda_{\min})$，$\lambda_{\min}$ 是计算中所使用的最小核数。弱可扩展性为固定每个 CPU 核处理的平均原子数后，随 CPU 核数的增加（体系的总粒子数也相应地发生改变），平均每个 CPU 核计算时间的变化趋势。图 1-6 和图 1-7 所示分别为 RBE 和 PPPM 两个算法的 CPU 性能和弱/强可扩展性的比较结果。

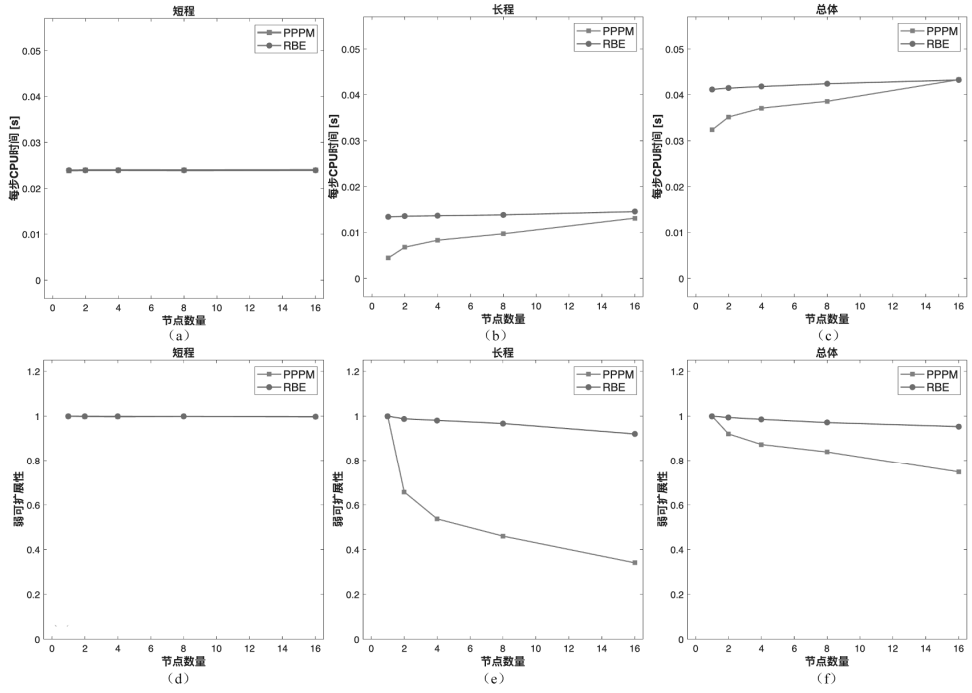

图 1-6 RBE 和 PPPM 在弱可扩展性指标上的比较

图 1-6 中近场（用 Short 标注）、远场（用 Long 标注）和总时间（用 Total 标注）各自的平均 CPU 时间如图 1-6（a）～（c）所示，对应的弱可扩展性如图 1-6（d）～（f）所示，我们固定平均每核粒子数为 2703，样本数为 200。

由图 1-6 可得到以下弱可扩展性结果：当节点数为 1 时，RBE 和 PPPM 大致具有相同的性能。当节点数增加时，RBE 算法的弱可扩展性会显著超过 PPPM。当节点数为 2 时，这一优势已经体现得十分明显。当使用 16 个节点时，对于远场部分，RBE 依然维持了 90%的弱可扩展性，而 PPPM 的这一数据已经跌到 30%。该测试结果证明了 RBE 在近场、远场和总时间 3 项测试中都能表现出近乎完美的弱可扩展性。

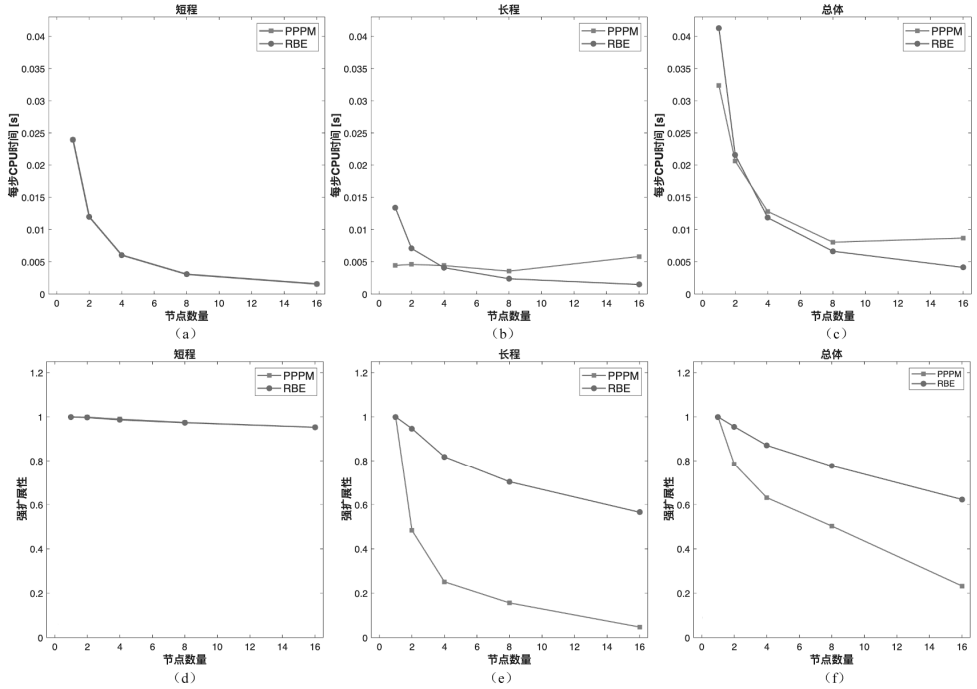

图 1-7　RBE 和 PPPM 在 CPU 时间和强可扩展性指标上的比较

使用的节点数最多为 16，体系中包含 927129 个原子，样本数为 200。图 1-7 中，近场（用 Short 标注）、远场（用 Long 标注）和总时间（用 Total 标注）3 个部分各自的平均 CPU 时间如图 1-6（a）～（c）所示，对应的强可扩展性如图 1-7（d）～（f）所示。

由图 1-7 可得到以下强可扩展性结果：当节点数少时，RBE 和 PPPM 大致具有相同的性能。当节点数增加时，RBE 算法的性能会逐渐超过 PPPM。当节点数达到 4 时，这一优势开始体现得更为明显。当使用 16 个节点时，RBE 的远场计算速度比 PPPM 要快一个数量级以上。此时对于远场部分，RBE 依然维持了 60%的强可扩展性，而 PPPM 的这一数据已经跌到 10%以下。这些结果充分地证明了 RBE 在并行可扩展性方面具有非常强大的性能。

1.9　总结

基于鲲鹏处理器自研开发的分子模拟高效算法 RBE 充分利用了单节点多核心的优势，对结构因子的归约及采样样本的通信进行 MPI 操作，并使用 ARM NEON 向量

化指令集命令对采样、结构因子及力的计算进行向量化操作，以及使用掩码查表等 LAMMPS 的特性功能，实现了可进行高精度、高效率及高可扩展性模拟计算的优势。

在鲲鹏超算平台上，LAMMPS-RBE 可以做到单节点计算效率与 PPPM 方法持平，而随节点数增加，在 16 节点时相较于 PPPM 方法具有一个数量级以上的提升。

参考文献

[1] ALLEN M P, TILDESLEY D J. Computer Simulation of Liquids [M]. Oxford: Oxford University Press, 2017.

[2] FRENCH R H, PARSEGIAN V A, PODGORNIK R, et al. Long Range Interactions in Nanoscale Science [J]. Reviews of Modern Physics, 2010, 82(2): 1887.

[3] HOLLINGSWORTH S A, DROR R O. Molecular Dynamics Simulation for All [J]. Neuron, 2018, 99(6): 1129-1143.

[4] KARPLUS M, PETSKO G A. Molecular Dynamics Simulations in Biology [J]. Nature, 1990, 347(6294): 631-639.

[5] HOCKNEY R W, EASTWOOD J W. Computer Simulation Using Particles [M]. Boca Raton: CRC Press, 1988.

[6] THOMPSON A P, AKTULGA H M, BERGER R, et al. LAMMPS-a Flexible Simulation Tool for Particle-Based Materials Modeling at the Atomic, Meso, and Continuum Scales [J]. Computer Physics Communications, 2022, 271: 108171.

[7] DARDEN T, YORK D, PEDERSEN L. Particle mesh Ewald: An N log(N) Method for Ewald Sums in Large Systems [J]. Journal of Chemical Physics, 1993, 98(12): 10089-10092.

[8] ESSMANN U, PERERA L, BERKOWITZ M L, et al. A Smooth Particle Mesh Ewald Method [J]. The Journal of Chemical Physics, 1995, 103(19): 8577-8593.

[9] GREENGARD L, ROKHLIN V. A Fast Algorithm for Particle Simulations [J]. Journal of Computational Physics, 1987, 73(2): 325-348.

[10] KOHNKE B, KUTZNER C, GRUBMÜLLER H. A GPU-Accelerated Fast Multipole Method for GROMACS: Performance and Accuracy [J]. Journal of Chemical Theory and Computation, 2020, 16(11): 6938-6949.

[11] CASE D A, AKTULGA H M, BELFON K, et al. AmberTools [J]. Journal of Chemical Information and Modeling, 2023, 63(20): 6183-6191.

[12] PHILLIPS J C, HARDY D J, MAIA J D C, et al. Scalable Molecular Dynamics on CPU and GPU Architectures with NAMD [J]. The Journal of Chemical Physics, 2020, 153(4): 044130.

[13] PÁLL S, ZHMUROV A, BAUER P, et al. Heterogeneous Parallelization and Acceleration of Molecular Dynamics Simulations in GROMACS [J]. The Journal of Chemical Physics, 2020, 153(13): 134110.

[14] JIN SH, LI L, LIU J G. Random Batch Methods (RBM) for Interacting Particle Systems [J]. Journal of Computational Physics, 2020, 400: 108877.

[15] JIN SH, LI L, XU ZH L, et al. A Random Batch Ewald Method for Particle Systems with Coulomb Interactions [J]. SIAM Journal on Scientific Computing, 2021, 43(4): B937-B960.

[16] LIANG J Y, TAN P, ZHAO Y, et al. Superscalability of the Random Batch Ewald Method [J]. The Journal of Chemical Physics, 2022, 156(1): 014114.

[17] FRENKEL D, SMIT B. Understanding Molecular Simulation: From Algorithms to Applications(Second Edition) [M]. Amsterdam: Academic PressElsevier, 2001.

[18] METROPOLIS N, ROSENBLUTH A W, ROSENBLUTH M N, et al. Equation of State Calculations by Fast Computing Machines [J]. The Journal of Chemical Physics, 1953, 21(6): 1087-1092.

[19] LIANG J, XU ZH L, ZHOU Q. Random Batch Sum-of-Gaussians Method for Molecular Dynamics Simulations of Particle Systems [J]. SIAM Journal on Scientific Computing, 2023, 45(5): B591-B617.

[20] WOLFF D, RUDD W G. Tabulated Potentials in Molecular Dynamics Simulations [J]. Computer Physics Communications, 1999, 120(1): 20-32.

[21] LIANG J Y, TAN P, HONG L, et al. A Random Batch Ewald Method for Charged Particles in the Isothermal-Isobaric Ensemble[J]. The Journal of Chemical Physics, 2022, 157(14)：140901.

[22] LIANG J Y, XU ZH L, ZHAO Y. Energy Stable Scheme for Random Batch Molecular Dynamics [J]. The Journal of Chemical Physics, 2024, 160(3): 034101.