计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190500009

基于 Python 的大规模高性能 LBM 多相流模拟

徐传福　王　曦　刘　舒　陈世钊　林　玉
国防科技大学计算机学院　长沙 410073

摘　要　Python 由于具有丰富的第三方库、开发高效等优点，已成为数据科学、智能科学等应用领域最流行的编程语言之一。Python 强调了对科学与工程计算的支持，目前已积累了丰富的科学与工程计算库和工具。例如，SciPy 和 NumPy 等数学库提供了高效的多维数组操作及丰富的数值计算功能。以往，Python 主要作为脚本语言，起到连接数值模拟前处理、求解器和后处理的“胶水”功能，以提升数值模拟的自动化处理水平。近年来，国外已有学者尝试采用 Python 代码实现求解计算功能，并在高性能计算机上开展了超大规模并行计算研究，取得了不错的效果。由于自身特点，高效大规模 Python 数值模拟的实现和性能优化与传统基于 C/C++和 Fortran 的数值模拟等具有很大的不同。文中实现了国际上首个完全基于 Python 的大规模并行三维格子玻尔兹曼多相流模拟代码 PyLBMFlow，探索了 Python 大规模高性能计算和性能优化方法。首先，利用 NumPy 多维数组和通用函数设计实现了 LBM 流场数据结构和典型计算内核，通过一系列性能优化并对 LBM 边界处理算法进行重构，大幅提升了 Python 的计算效率，相对于基准实现，优化后的串行性能提升了两个量级。在此基础上，采用三维流场区域分解方法，基于 mpi4py 和 Cython 实现了 MPI+OpenMP 混合并行；在天河二号超级计算机上成功模拟了基于 D3Q19 离散方法和 Shan-Chen BGK 碰撞模型的气液两相流，算例规模达百亿网格，并行规模达 1024 个结点，并行效率超过 90%。

关键词：格子玻尔兹曼方法；并行计算；性能优化；高性能 Python；多相流模拟
中图法分类号　TP391

Large-scale High-performance Lattice Boltzmann Multi-phase Flow Simulations Based on Python

XU Chuan-fu，WANG Xi，LIU Shu，CHEN Shi-zhao and LIN Yu
College of Computer，National University of Defense Technology，Changsha 410073，China

Abstract　Due to the plenty of third-party libraries and development productivity，Python is becoming increasingly popular as a programming language in areas such as data science and artificial intelligence. Python has been providing fundamental support for scientific and engineering computing. For example，libraries such as NumPy and SciPy provide efficient data structures for multi-dimensional arrays and rich numerical functions. Traditionally，Python was used as a script language，gluing preprocessors，solvers and postprocessors and enhancing automation in numerical simulations. Recently，some foreign researchers implement their solvers using Python and parallelize their Python codes on high performance computers，with impressive results achieved. Because of its intrinsic features，implementation and optimization of high performance large-scale numerical simulations with Python are quite different with traditional language such as C/C++ and Fortran. This paper presented a large-scale parallel open source 3D Lattice Boltzmann multi-phase flow simulation code PyLBMFlow with Python，and investigated large-scale parallel computing and performance optimization for Python numerical applications. It designed LBM flow data structures and computational kernels with NumPy multi-dimensional arrays and universal functions. Through a range of optimization including reconstruction of boundary processing，it dramatically improves the efficiency of Python computing by about 100X，compared with the baseline version，on a CPU core. Furthermore，this paper designed a 3D decomposition method and implement hybrid MPI+OpenMP parallelization using mpi4py and Cython. Tests for 3D multi-phase(liquid and gases) problem(about 10 Billion lattices) simulating drop impact with gravity effect using D3Q19 Lattice Boltzmann discretization and Shan-Chen BGK single relaxation time collision model were presented，achieving a weak parallel efficiency of above 90% in going from 64 to 1024 compute nodes.

Keywords　Lattice Boltzmann method，Parallel computing，Performance optimization，High performance Python，Multi-phase flow simulation

到稿日期：2019-05-05　返修日期：2019-11-25　本文已加入开放科学计划(OSID)，请扫描上方二维码获取补充信息。
通信作者：徐传福(xuchuanfu@nudt.edu.cn)

1 引言

与传统基于 Navier-Stokes 方程的计算流体力学(Computational Fluid Dynamics,CFD)模拟方法不同,格子玻尔兹曼方法(Lattice Boltzmann Method,LBM)从微观的角度[1-2]将流场划分为小格子(质点),并通过碰撞模型(格子的碰撞和流动)模拟流体的动态演化。结合复杂的几何外形和边界条件,LBM 正被日益广泛地应用于各种复杂流动问题的模拟中,包括湍流、多组分掺混流、自由表面流、传热流、多相流等[3-8],并取得了较好的应用效果。

大规模 LBM 模拟通常需要借助高性能计算平台。与传统的基于 Navier-Stokes 方程的 CFD 相比,LBM 的数值计算相对简单,其计算主要分为碰撞运算和流动运算,其中碰撞运算完全是局部的。因此 LBM 非常适合并行计算,各种 LBM 应用的大规模并行计算和性能优化也成为了高性能计算应用的热点之一。例如,Mountrakis 等[9]评估了开源 LBM 框架 Palabos 的 MPI 性能,其并行规模扩展到了 8 192 个核,表明该框架具有优异的强弱并行扩展性。Williams 等[10]提出的多核平台上 LBM 计算自动调优方法,可针对 Intel 和 AMD 等多核处理器自动生成高度优化的 LBM 计算内核,获得了超过 15.0 倍的加速。Pananilath 等[11]也开发了针对 LBM 模拟的代码自动优化生成工具,其支持的性能优化方法包括循环分块、负载均衡以及向量化等,在 Intel Sandy Bridge CPU 上取得了 3.0 倍的加速。GPU 和 MIC 等众核加速部件或协处理器出现以后,很多研究人员也开展了 LBM 模拟的异构并行计算和性能优化研究。例如,Feichtinger 等[12]在基于 GPU 的超级计算机 Tsubame 2.0 上实现了 LBM 代码的加速,提出了通信开销模型,并在 1 000 多块 GPU 上进行了大规模测试。笔者也在天河二号超级计算机上实现了 LBM 代码 openlbmflow 的超大规模 CPU+MIC 异构协同并行和深度性能优化[13]。已有的大规模 LBM 代码通常在 CPU 平台上采用区域分解的方法实现 MPI 并行或 MPI+OpenMP 混合并行,在 GPU 和 MIC 上则分别采用 CUDA/OpenCL 以及 Intel Offload 实现异构并行。

在 CFD 等科学计算领域,高效的数值计算内核通常采用 C,Fortran 或 C++编程实现。与 C/Fortran 等翻译型语言不同,Python 是一种动态的解释型面向对象语言,语法简洁清晰,具有开发快速、易测试、易维护、跨平台、代码紧凑易读的优点,可同时满足脚本开发和通用开发的需求,近年来已成为数据科学、智能科学和计算科学等领域最流行的编程语言之一。Python 从诞生起便强调对数值计算的支持,目前已经积累了丰富的科学与工程计算库和工具。例如,NumPy[14]提供了强大的多维数组数据结构,SciPy[15]则提供了丰富的数值算法功能。Python 同时提供了丰富的高性能并行编程支持,几乎所有的并行编程模型都有对应的 Python 接口实现,如与消息传递 MPI 并行对应的 mpi4py、与 CUDA/OpenCL 异构并行对应的 PyCUDA/PyOpenCL 等。此外,在科学与工程计算应用领域,用户通常更专注于求解器的开发,借助商业软件实现前后处理(例如网格生成、可视化等)。当前,大部分商业前后处理软件都提供了 Python 接口,用户可以利用 Python 很容易地将自主开发的求解器与商业前后处理工具以松耦合的方式集成。同时,Python 提供了大量开源前后处理库和工具,方便用户快速开发出简单的前后处理功能和控制界面。Python 由于具有动态解释型语言的特点,其数值计算效率通常远远低于 C/Fortran 等传统语言的计算效率。为了提升 Python 数值计算的效率,一方面可借助 Python 与其他语言良好的互操作性,采用 C/Fortran 等实现关键计算内核,Python 仅用于数据管理和流程控制等高层应用功能;另一方面,可以利用 Python 提供的专用性能优化库(如 Cython,numba 等),在不增加代码复杂性的情况下大幅提升性能。

近年来已有一些学者初步探索了基于 Python 的大规模高性能 CFD 计算。例如,Mortensen 等[16]在 Shaheen 超级计算机上基于 Python 实现了数千处理器核以及数十亿自由度的大规模高性能伪谱方法——湍流直接数值模拟(Direct Numerical Simulation,DNS),整个 DNS 的实现仅不到 100 行 Python 代码,采用 Cython 优化后,其性能与 C++相当且并行扩展性略好。值得一提的是,英国帝国理工学院的 Peter 等基于 Python 和通量重构方法实现了一个高精度的非结构 CFD 求解器 PyFR[17],该求解器已被应用于多个复杂流体问题的模拟中。利用跨平台模板语言引擎 Mako,PyFR 可自动生成针对 GPU 等多种异构平台的代码。在 Titan 和 Piz Daint 超级计算机上,使用 PyFR 求解非结构网格多级涡轮叶片 B5S1.760 算例,可获得 13.7 PFLOP/s 的双精度浮点运算性能,达到了峰值性能的 58%,入围了 2016 年的 ACM Gorden Bell 奖。以上表明,采用 Python 实现超大规模高性能科学与工程计算是切实可行的。

由于自身特点,高效大规模 Python 数值模拟的实现和性能优化与传统的 C/C++和 Fortran 等代码具有很大的不同,探索 Python 并行计算和性能优化对于实际应用具有重要意义。本文实现了国际上第一个完全基于 Python 的三维格子玻尔兹曼多相流模拟代码 PyLBMFlow,在天河二号上实现了大规模混合并行,并提出了一系列 Python 高性能计算优化方法。首先,基于 NumPy 多维数组和通用函数设计实现了流场数据结构和典型 LBM 计算内核,通过一系列性能优化和边界处理算法重构,大幅提升了 Python 的计算效率,其性能相对于基准实现提升了两个量级。其次,在三维流场区域分解的基础上,利用 mpi4py 和 Cython 实现了大规模 MPI+OpenMP 混合并行,在天河二号超级计算机上成功模拟了基于 D3Q19 离散方法和 Shan-Chen BGK 碰撞模型的气液两相流,算例规模达百亿网格,并行规模达1 024 个结点,并行效率超过 90%。

2 LBM 多相流方法和实现流程

PyLBMFlow 基于 LBM 中常用的 D2Q9/D3Q19 离散模型以及 Shan-Chen BGK 单松弛时间碰撞模型来模拟多相流,可以模拟在重力影响下有周期边界或反弹边界条件的 2D/3D 单相或多相流问题。

2.1 格子 BGK 模型

LBM 的粒子分布函数遵循格子玻尔兹曼方程。格子玻尔兹曼方程从微观的角度将宏观的流动现象描述为大规模微

观粒子运动的统计学平均结果。PyLBMFlow 中的粒子碰撞使用的是 BGK 模型，其中单松弛时间的格子玻尔兹曼演化方程为：

$$f_i(x+c_i\Delta t,t+\Delta t)-f_i(x,t)=-\frac{1}{\tau}[f_i(x,t)-f_i^{\mathrm{eq}}(x,t)] \tag{1}$$

其中，τ 是松弛因子，c_i 是离散速度，f_i 是粒子分布函数，f_i^{eq} 是平衡态分布函数，$i=0,1,\cdots,N$ 为离散速度方向。

1992 年，Qian 等提出了 DdQm（d 为空间维数，m 为离散速度个数）系列模型。DdQm 模型是格子玻尔兹曼方法中的基本模型，本文采用的离散模型为 D3Q19。

D3Q19 离散模型的平衡态分布函数为：

$$f_i^{\mathrm{eq}}=\omega_i\rho\left[1+\frac{c_i\cdot u}{c_s^2}+\frac{(c_i\cdot u)^2}{2c_s^4}-\frac{u^2}{2c_s^2}\right] \tag{2}$$

其中，ω_i 为权系数，c_s 为无量纲声速。对于 D3Q19 离散速度模型，有：

$$c_s^2=c^2/3 \tag{3}$$

$$\omega_i=\begin{cases}1/3, & i=0\\ 1/18, & i=1,2,\cdots,6\\ 1/36, & i=7,8,\cdots,18\end{cases} \tag{4}$$

$$c_i=\begin{cases}(0,0,0)c, & i=0\\ (\pm1,0,0)c,(0,\pm1,0)c,(0,0,\pm1)c, & i=1,2,\cdots,6\\ (\pm1,\pm1,0)c,(\pm1,0,\pm1)c,(0,\pm1,\pm1)c, & i=7,8,\cdots,18\end{cases} \tag{5}$$

其中，$c=\Delta x/\Delta t$ 为格子速度，Δx 为格子步长，Δt 为时间步长。

格子玻尔兹曼方程中流体的宏观密度和动量可由式(6)和式(7)得到：

$$\rho=\sum_i f_i \tag{6}$$

$$\rho u=\sum_i c_i f_i \tag{7}$$

流体粘性系数和松弛因子的关系为：

$$\nu=(\tau-\frac{1}{2})c_s^2\Delta t \tag{8}$$

2.2 Shan-Chen 模型

PyLBMFlow 多相流模拟中使用的是 Shan-Chen BGK 模型。Shan-Chen 模型引入了粒子间的相互作用力以及固体与液体间的相互作用力。在该模型中，微观粒子间的作用力为：

$$F(x)=-G\varphi(x,t)\sum_i w_i\varphi(x+c_i\Delta t,t)c_i \tag{9}$$

流体与固体间的作用力为：

$$F_{\mathrm{ads}}(x)=-G_{\mathrm{ads}}\varphi(x,t)\sum_i w_i\, s(x+c_i\Delta t,t)c_i \tag{10}$$

其中，w_i 是权函数。

$$w_i=\begin{cases}1/18, & i=1,2,\cdots,6\\ 1/36, & i=7,8,\cdots18\\ 0, & \text{其他}\end{cases} \tag{11}$$

式(9)和式(10)中，G 是两相液体相互作用的强度，G_{ads} 为吸附参数。当 $x+c_i\Delta t$ 所在位置为流体时，$s=0$；当 $x+c_i\Delta t$ 所在位置为固体时，$s=1$。可通过调整 G 来改变流体的表面张力，通过调整 G_{ads} 来调整流体与固体接触角的大小。$\varphi(x,t)$ 是相互作用势能，为满足等温过程，可使用如下相互作用势能：

$$\varphi(x,t)=\varphi_0\mathrm{e}^{-\rho_0/\rho} \tag{12}$$

考虑重力影响时，重力的计算公式为：

$$F_g(x)=\rho(x)g \tag{13}$$

其中，g 为重力加速度。

因此本文的多相流模拟中，体积力为：

$$F_a=F+F_{\mathrm{ads}}+F_g \tag{14}$$

原始的 Shan-Chen 模型引入力项的方法是改变平衡态分布函数中的平衡态速度，以反映粒子间的相互作用。平衡态速度为：

$$u^{\mathrm{eq}}=u+(\tau F_a)/\rho \tag{15}$$

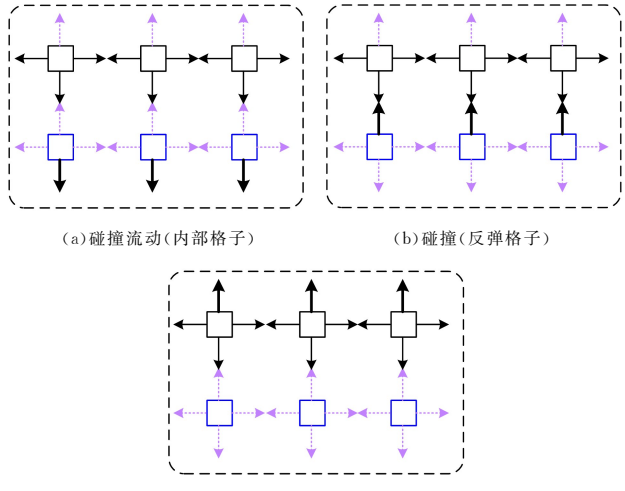
流体的真实速度为：

$$U=u+F_\rho/(2\rho) \tag{16}$$

2.3 边界条件

在边界条件的处理中，周期边界网格被视为正常的内部网格。固壁边界使用标准的反弹规则，反弹在内部格子完成碰撞和流动后进行更新。

图 1 给出了 2D 情况下的反弹机制，其中蓝色边框格子为反弹边界，黑色边框格子为反弹边界邻居，箭头表示每个格子的分布函数分量。首先，内部格子碰撞流动完后将被更新，同时反弹边界格子的向外分量（加粗黑色箭头）也被更新，如图 1(a)所示。然后，固体边界碰撞步骤，向外的分量以反弹的形式改变方向，以模仿固体表面的反弹效应，如图 1(b)所示。最后，反弹边界格子的反向分量迁移到近边界格子的相应分量中，如图 1(c)所示。



(a)碰撞流动(内部格子)　(b)碰撞(反弹格子)

(c)流动(反弹格子)

图 1　2D 固壁边界反弹示例(电子版为彩色)

Fig. 1　Example of 2D bounceback boundary

2.4 实现流程

图 2 给出了 PyLBMFlow 的实现流程，主要分为初始化、时间迭代和后处理 3 个阶段。在初始化阶段，设置流场几何结构，初始化密度和分布函数等流场变量。时间迭代阶段包含 3 个主要的计算过程：1)计算粒子间作用力 ϕ、速度 u、密度 ρ(cal_rup)；2)通过碰撞和流动计算(离散后的)粒子分布函数 fn(cal_fn，对应图 1(a))；3)更新反弹边界(boundary_update，对应图 1(b)和图 1(c))。后处理阶段根据用户指定的时间迭代步间隔输出模拟结果，模拟结束后进行可视化。对于三维 LBM 模拟，时间迭代阶段的 3 个计算内核函数采用

C/C++和 Fortran 实现时通常为多重循环结构，其中 *cal_rup* 和 *cal_fn* 是典型的模板计算(Stencil Computation)。大规模并行计算基于区域分解，每个迭代步交换一次边界影像区，因此图 2 中虚线框部分给出了实现 MPI 并行后添加的通信过程 mpi_exchange。

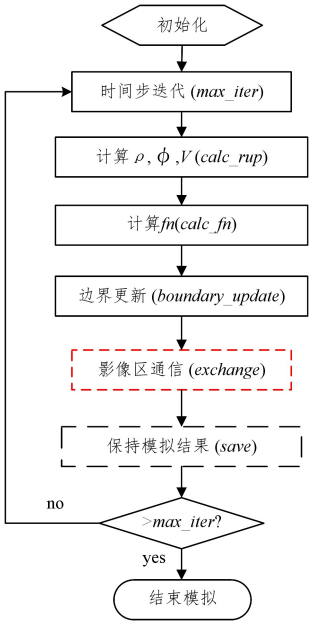


图 2 PyLBMFlow 的实现流程

Fig. 2 Flow of PyLBMFlow implementation

3 Python 实现和性能优化方法

3.1 流场数据结构和计算内核的实现

三维 LBM 应用通常采用多维数组表示流场数据结构。Python 通过 array 或 list 实现了对数组的支持，但 array 模块不支持多维数组，也没有各种数组运算函数。list 支持多维数组，但 list 中只存放了每个数组元素的指针，需要额外存储相应的对象，内存空间和间接访问开销较大。本文基于 NumPy 对流场数据结构进行了设计和优化。NumPy. ndarray 不仅提供了完善的多维数组支持，而且提供了大量内置的通用数组运算函数 ufunc。相比传统基于 C/Fortran 等多维数组和多重循环的计算内核，利用 ndarray 和 ufunc 不仅更加简洁，而且由于 NumPy 库的 ufunc 函数在 C 语言级别进行了实现和优化，计算效率也能够得到保证。

本文的密度等三维流场均采用 3 维数组；而 D3Q19 模型的粒子分布函数 *fn* 包含 19 个分量，需要采用 4 维数组。多维数组的数据布局决定了数组元素访问的连续性，直接影响数组访问开销。结构体数组(Array of Structure，AoS)更适用于面向对象的设计，便于连续访问一个对象的多个属性。而数组结构体(Structure of Array，SoA)更适用于连续访问不同对象的同一属性。为更加连续地访问数组元素，本文函数 *fn* 使用 SoA，，因此将 19 个分量设置成多维数组的高维度(即 4 维数组的第 4 维)。

3.2 性能优化

图 3 给出了优化过程的部分代码(以 *cal_rup* 为例)。

```
### 基于 list 的实现
def cal_rup():
    for x in range(nx):
        for y in range(ny):
            for z in range(nz):
                # 更新非壁面单元的密度
                if solid[x,y,z]==0:
                    rho[x,y,z]= sum_{i=0}^{18} fp[x,y,z,i] #求和
                # 更新非壁面单元的势能
                phi[x,y,z]=1.0-exp(-rho[x,y,z])
```

(a)基于 list 的实现

```
### 基于 NumPy. ndarray，ufunc 和布尔数组的实现
def cal_rup():
    #其中 bound 为流体计算域索引，用于判断是否为边界
    rho[bound]=np. sum(fp[:,bound],0)
    #更新非边界点势能
    phi[:]=1.0-np. exp(-rho)
### 基于 NumPy. where()的实现
def cal_rup():
    rho=np. where(bound,np. sum(fp,0),rho)
```

(b)基于 NumPy. ndarray 的实现

```
### 基于 Cython 的实现
@cython. boundscheck(False)
@cython. wraparound(False)
def cal_rup():
    cdef int x,y,z
    cdef double tmp_rho
    ##实现 OpenMP 线程并行
    # with nogil:
    for x in prange(bound. shape[0],schedule='guided',nogil=True):
        for y in range(bound. shape[1]):
            for z in range(bound. shape[2]):
                if bound[x,y,z]:
                    rho[x,y,z]= sum_{i=0}^{18} fp[x,y,z,i]#求和
                    phi[x,y,z]=1.0-exp(-rho[x,y,z])
```

(c)基于 Cython 的实现

```
### 基于 numba 的实现
@jit('(f8,f8,f8,int8[:,:,:],f8[:,:,:],…,f8[:,:,:],f8[:,:,:,:,:])',
  nopython=True)
def cal_rup():
    for x in range(bound. shape[0]):
        for y in range(bound. shape[1]):
            for z in range(bound. shape[2]):
                if bound[x,y,z]:
                    rho[x,y,z]= sum_{i=0}^{18} fp[x,y,z,i] #求和
                    phi[x,y,z]=1.0-math. exp(-rho[x,y,z])
```

(d)基于 numa 的实现

图 3 代码优化过程

Fig. 3 Skeleton of code optimization

图 3(a)是采用 Python 的 list 和多重循环的实现。图 3(b)第 1—5 行是采用 NumPy 多维数组和通用数组函数的实现，这里布尔数组 bound 用于判断格子单元是否为壁面边界，可以看出基于 NumPy 的实现代码非常紧凑。程序实现中边界处理部分(boundary_update)仍然需要采用循环方式。使

用下标获取 NumPy 数组中的元素时，需要将其转化为标准的 Python 标准对象，有一定的额外开销。因此，可以使用 NumPy 提供的 item()和 itemset()方法，直接以标准 Python 数值类型获得数组中的元素。图 3(b)第 6－8 行给出了采用 NumPy. where()函数代替布尔数组的优化。where()可以看成条件判断的数组版本，调用方式为 x＝where(*condition*，y，z)，其中 *condition*，y 和 z 都为数组，当 condition 中某个元素为 true 时，数组 x 中对应下标的值从数组 y 获取，否则从数组 z 获取。计算内核采用 where()实现，以减少布尔数组带来的额外开销，提升程序性能。

在边界条件处理中，周期边界网格被视为正常的内部网格。固壁边界使用标准反弹规则，反弹在内部格子完成碰撞和流动后进行更新。在边界条件实现中，本文对图 1 给出的固壁边界反弹处理方法重新进行了设计。由图 1 可以看出，白色格子(反弹边界邻居)向上分量值的获取需要经过图 1 中的 3 个过程。在程序中，将格子速度逆转后再沿原路反弹合成一步，可以使得白色格子向上分量值的获取只经过图 1(a)和图 1(b)这两状态，从而提升性能。

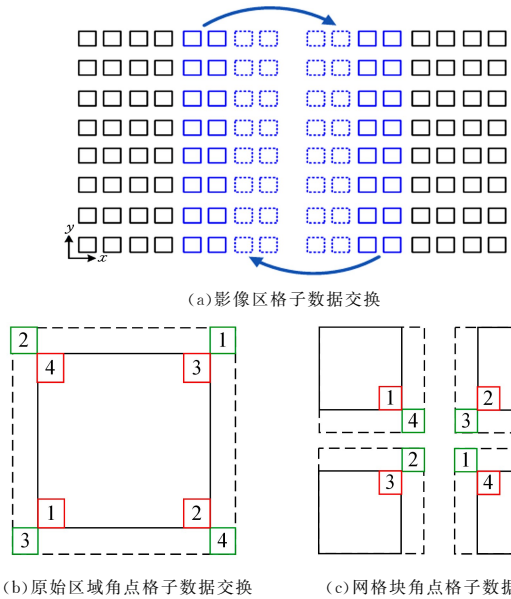
为了进一步优化计算性能，可以采用 C 语言等实现计算内核并提供 Python 接口。Python 与其他语言良好的互操作性，使得这种混合编程在保证高效开发的同时提升计算效率。Cython 的出现方便了基于 C 语言的 Python 扩展模块的开发，其语法仅在 Python 的基础上增加了直接定义和调用 C 函数、定义变量类型等。通过 Cython 编译器可以将 Cython 源程序编译成 C 源程序，进而编译成 Python 扩展模块。同时，Python 的 GIL(Global Interpreter Lock)机制，导致通常无法直接在 Python 循环代码中实现类似于 OpenMP 的共享存储线程并行，但 Cython 支持 OpenMP 并行，因此本文的共享存储并行版本是在 Cython 的基础上实现的(需要禁用 GIL，并在 C 编译器中指定 OpenMP 编译选项)。Cython 同时提供了一些编译优化选项。例如，cdef 关键字可以加快运算速度；显式定义参数数组的元素类型和维度，并使用 boundscheck 和 wraparound 关闭负数下标处理和越界检查，从而优化数组的存取速度(见图 3(c))。

Python 代码缺省是动态解释执行的，将代码编译成机器码可以极大地加快运行速度。编译可分为 AOT(Ahead Of Time)和 JIT(Just In Time)两类。AOT 指运行前编译，如普通的静态编译、在运行前编译成可用的库。JIT 指运行时编译，即边运行边编译，运行前无需太多额外的工作，但程序可能会有冷启动问题，即首次运行代码时由于需要编译运行，速度可能很缓慢。本文采用 numba 对代码实现运行时编译优化，即在运行时使用 llvm 编译器将 Python 代码编译为本地机器指令。整个运行时编译优化的实现非常简单，只需要在函数前添加 JIT 装饰器，声明函数的返回类型、输入参数类型和维度即可，如图 3(d)所示。

4　Python 大规模并行计算

为了实现 MPI 并行，需要对原始流场区域进行区域分解。本文针对三维流场采用了三维区域分解策略：将原始区域沿着 3 个坐标方向划分为多个网格块，每个 MPI 进程负责一个块的计算。为了简化实现，每个天河二号计算结点仅创建一个 MPI 进程(结点上网格块块内的并行由 OpenMP 实现)。计算过程中，网格块需要与相邻的网格块进行数据交换，因此在网格块外扩展了两层影像区(ghost)。使用两层影像区而不是一层影像区，是因为周期性边界影像区处理需要额外的一层影像区。

每个时间步迭代计算后都需要进行 MPI 通信，更新影像区，图 4 给出了 2D 情况下的数据交换示意图。图 4(a)给出了边界影像区的数据交换。源网格块边界网格数据(用实线表示)传输到目的网格块后更新相应的影像区数据(虚线网格)。图 4(b)显示了整个原始流场计算域内角点格子的数据交换，计算域内角点格子(红色线方块)传输更新到有相同数字的影像区(绿色线方块)，例如右上角绿色线块 1 由左下角红色线块 1 更新。图 4(c)给出了网格块间角点格子的数据交换方式。



(a)影像区格子数据交换

(b)原始区域角点格子数据交换　　(c)网格块角点格子数据交换

图 4　相邻网格块的 MPI 通信数据交换(电子版为彩色)

Fig. 4　MPI communication for neighboring blocks

具体实现时，三维区域分解，使得需要通信交换的边界和影像区数据位于 3 个坐标方向的 6 个面上，因此这些数据是不连续的。MPI 通信仅能够发送/接收连续的数据，因此通常需要创建发送/接收缓冲区。在发送操作之前，将边界数据收集到发送缓冲区后进行发送。接收数据到接收缓冲区后，将数据分发到影像区。本文采用 mpi4py 中的 Creat_vector()函数将数组中的数据块组合起来构建派生数据类型，用于 x，y，z 方向的影像区数据交换，语法为：Type. Create_vector(int *count*，int *blocklength*，int *stride*)。其中，Type 是通信数据类型，*count* 表示通信的数据块数(x 方向对应于 D3Q19 模型中的 Q＝19)，每块有 *blocklength* 个元素，且每块间的间隔为 *stride* 个 Type 型元素。算法 1 给出了优化前后 x 方向的 MPI 通信，可以看出基于 Creat_vector()的实现无须创建发送/接收缓冲区，避免了针对每个方向分别进行数据收集和分发导致的内存拷贝开销，程序的运行效率得到提高，代码规模也有所减小。OpenMP 共享存储的并行是在 Cython 中通过 prange(parallel range)语句实现的(见图 3(c)的第 8 行)。

算法 1 MPI 优化

```
1. ### 原始版
2. def mpi_exchange():
3.     #x 方向数据交换
4.     #收集数据
5.     fs_lef[:]=fp[:,send_lef_off:send_lef_off+ghostx,:,:]
6.     #发送/接收数据
7.     comm.Sendrecv(fs_lef,lef_pid,myid,fv_rig,rig_pid,rig_pid)
8.     #分发接收的数据
9.     fp[:,recv_lef_off+loc_nx+ghostx:recv_lef_off+loc_nx+ghostx+
       ghostx,:,:]=fv_rig
10. ### 改进版
11. def mpi_create_new_type():
12.       # x 方向数据交换,构建发送数据的数据类型 x_type
13.     x_type_t=MPI.DOUBLE.Create_vector(Q,ghostx * end_y * end_z,
       end_x * end_y * end_z)
14.     x_type=x_type_t.Create_resized(0,MPI.DOUBLE.size)
15.     x_type.Commit()
16.     return x_type,y_type,z_type
17. def mpi_exchange(x_type,y_type,z_type):
18.     # 发送/接收数据
19.     comm.Sendrecv([fp,(1,ghostx * end_y * end_z),x_type],lef_pid,myid,
       [fp,(1,(loc_nx+ghostx) * end_y * end_z),x_type],rig_pid,rig_pid)
```

5 测试结果与分析

5.1 测试平台

本文实验是在天河二号超级计算机[18]上进行的。天河二号由国防科学技术大学开发,在全球超级计算机榜单上曾获六连冠,现排名第二,共计 16000 个节点。其中每个计算节点都有 2 个具有 64GB 共享内存的 Xeon CPU 和 3 个 Xeon Phi 协处理器(即 MIC)。节点通过高速互连网络连接,网络带宽峰值为 160 Gbps。Xeon Phi 通过 PCI-e 2.0 连接到 CPU,数据传输速率为 10Gbps。每个 Xeon CPU 有 12 个核,2 个 CPU 双精度峰值浮点性能为 422 GFLOPS。每个 Xeon Phi 协处理器有 8GB 内存,57 个核心,每个核支持 4 个硬件线程,双精度浮点峰值性能为 1 TFLOPS。

测试采用的编译器为 Python 3.6 和 Icc14.0,其中用到的第三方库包括 Mpi4py 2.0,Numpy 1.13.1,Cython 0.26.1 以及 Numba 0.35.0。

5.2 测试结果

图 5 给出了各个版本优化后与基准实现的性能加速比,各个版本的描述如表 1 所列。基准版本为采用 Python 原生数据结构 list 实现的多维数组及其循环计算(v0_1)。可以看出,通过性能优化程序的加速比提升了近两个数量级。

表 1 各版本的描述

Table 1 Descriptions of various versions

版本号	版本说明
v0	采用 Python list 实现多维数组
v1	采用 narry 代替 list
v2	使用 ufunc 进行数组运算
v3	优化数据结构(Aos 到 SoA)
v4	采用 item 获取数组元素
v5	使用 where 函数来进行边界判定
v6	边界反弹算法重构
v7	改用 Cython 编写计算内核
v8	numba 即时编译

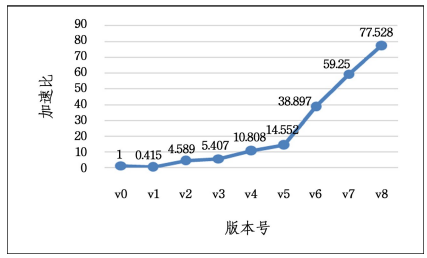


图 5 串行版本的性能优化效果

Fig. 5 Optimization performance of serial versions

图 6 给出了基于 Cython 版本(v5)实现的 OpenMP 线程并行的加速比。可以看出,OpenMP 并行达到 24 个线程时仍然具有良好的并行可扩展性,获得了最高达 10 倍左右的 OpenMP 并行加速比。

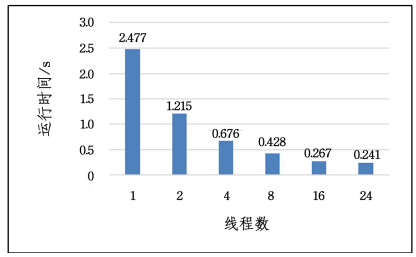


图 6 OpenMP 并行效果

Fig. 6 Effects of OpenMP parllelization

图 7 给出了在天河二号超级计算机上 PyLBMFlow 大规模 MPI+OpenMP 并行的弱并行可扩展性测试结果。实验由 64 个计算节点扩展到 1024 个计算节点,以 64 个计算节点作为性能基准,每个节点固定问题规模为 256 * 256 * 256,总的最大问题规模约为 170 亿网格单元。可以看出,扩展到 1024 个节点时,并行可扩展性仍然保持在 90%以上,充分证明了本文方法在并行实现和优化方面的有效性。

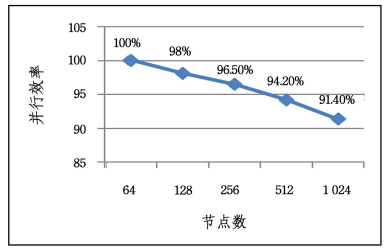
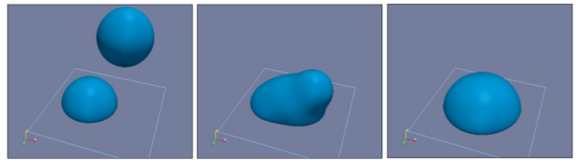


图 7 MPI+OpenMP 混合并行可扩展性

Fig. 7 Scalability of hybrid MPI+OpenMP implementation

4.3 模拟结果

本文模拟的算例为基于 D3Q19 离散方法和 Shan-Chen BGK 碰撞模型的三维气液两相流,考虑了重力影响。图 8 给出了模拟结果。



(a) (b) (c)

图 8 考虑重力效应的 3D 气液两相流模拟结果

Fig. 8 3D drop on drop impact multi-phase problem with gravity effect

整个计算的流场区域为长方体空间，不可见背景为空气，空气中包含了两个液滴，一个位于地板，另一个位于空气中（见图 8(a)）。由于重力影响，上方的液滴将坠落（见图 8(b)）并与地板上的液滴碰撞融合成为一个更大的液滴（见图 8(c)）。

结束语 本文实现了一个完全基于 Python 的三维格子玻尔兹曼多相流模拟代码 PyLBMFlow，在天河二号上实现了大规模并行和性能优化。通过优化，其串行性能相对于基准实现提升了近两个量级。在天河二号上成功模拟了基于 D3Q19 离散方法和 Shan-Chen BGK 碰撞模型的气液两相流，算例规模为百亿网格，并行规模为 1024 个节点，并行效率超过 90%。本文工作对于进一步推广 Python 在科学与工程计算领域以及其他领域的应用具有重要的实际意义。

参 考 文 献

[1] ANDERSON J D. Computational Fluid Dynamics[M]. McGraw-Hill，1995：1-30.

[2] SUCCI S，BENZI R，HIGUERA F. The lattice Boltzmann equation：A new tool for computational fluid dynamics[J]. Physica D：Nonlinear Phenomena，1991，47(1)：219-230.

[3] DAWSON S P，CHEN S，DOOLEN G D. Lattice Boltzmann computations for reaction-di_usion equations[J]. The Journal of Chemical Physics，1993，98(2)：1514-1523.

[4] SHAN X，CHEN H. Lattice Boltzmann model for simulating flows with multiple phases and components[J]. Physics Review E，1993，47(3)：1815-1819.

[5] ZHAO Y，WANG L，QIU F，et al. Melting and flowing in multiphase environment[J]. Computers and Graphics，2006，30(4)：519-528.

[6] BOSL W J，DVORKIN J，NUR A. A study of porosity and permeability using a lattice Boltzmann simulation[J]. Geophysical Research Letters，1998，25(9)：1475-1478.

[7] BAILEY P，MYRE J，WALSH S D，et al. Accelerating lattice Boltzmann fluid flow simulations using graphics processors [C]// Internation al Conference on Parallel Processing. 2009：550-557.

[8] BIFERALE L，MANTOVANI F，PIVANTI M，et al. Optimization of Multi-Phase Compressible Lattice Boltzmann Codes on Massively Parallel Multi-Core Systems[J]. Procedia Computer Science，2011，4(4)：994-1003.

[9] MOUNTRAKIS L，LORENZ E，MALASPINAS O，et al. Parallel performance of an IB-LBM suspension simulation framework [J]. Journal of Computational Science，2015，9：45-50.

[10] WILLIAMS S，CARTER J，OLIKER L，et al. Optimization of a lattice boltzmann computation on state-of-the-art multicore platforms[J]. Journal of Parallel and Distributed Computing，2009，69(9)：762-777.

[11] PANANILATH I，ACHARYA A，VASISTA V，et al. An optimizing code generator for a class of lattice-boltzmann computations[J]. ACM Transactions on Architecture and Code Optimization，2015，12(2)：1-14.

[12] FEICHTINGER C，HABICH J，KESTLER H，et al. Performance modeling and analysis of heterogeneous lattice boltzmann simulations on cpu-gpu clusters[J]. Parallel Computing，2015，46(C)：1-13.

[13] LI D，XU C，WANG Y，et al. Parallelizing and optimizing large-scale 3d multi-phase flow simulations on the tianhe-2 supercomputer[J]. Concurrency and Computation：Practice and Experience，2016，28：1678-1692.

[14] NumPy Official[OL]. http://www.numpy.org.

[15] Scipy Official[OL]. http://www.scipy.org.

[16] MORTENSEN M，LANGTANGEN H P. High performance Python for direct numerical simulations of turbulent flows[J]. Computer Physics Communications，2016，203：53-65.

[17] VINCENT P，WITHERDENY F，VERMERIRE B，et al. Towards Green Aviation with Python at Petascale[C]// IEEE International Conference for High Performance Computing and Data analytic. 2016：1-11.

[18] LIAO X，XIAO L，YANG C. Milkyway-2 supercomputer：system and application[J]. Frontier of Computer Science，2004，8(3)：345-356.

XU Chuan-fu，born in 1980，Ph.D，associate professor，is member of China Computer Federation（CCF）. His main research interests include parallel computing and applications.