信阳师范学院学报(自然科学版)
JOURNAL OF XINYANG NORMAL UNIVERSITY (Natural Science Edition)

面向多异构平台的不可压求解器的移植算法和优化

马文鹏, 李瑞莹, 袁武, 高凌云, 翟环欣

引用本文:
马文鹏, 李瑞莹, 袁武, 高凌云, 翟环欣. 面向多异构平台的不可压求解器的移植算法和优化[J]. 信阳师范学院学报自然科学版, 2023, 36(4): 632–639. doi: 10.3969/j.issn.1003–0972.2023.04.020
MA Wenpeng, LI Ruiying, YUAN Wu, GAO Lingyun, ZHAI Huanxin. Porting Algorithm and Optimization of Incompressible Solver for Multi–heterogeneous Platforms[J]. Journal of Xinyang Normal University (Natural Science Edition), 2023, 36(4): 632–639. doi: 10.3969/j.issn.1003–0972.2023.04.020

在线阅读 View online: https://doi.org/10.3969/j.issn.1003–0972.2023.04.020

DOI:10.3969/j.issn.1003-0972.2023.04.020　　文章编号:1003-0972(2023)04-0632-08

面向多异构平台的不可压求解器的移植算法和优化

马文鹏[1*],李瑞莹[1],袁　武[2,3],高凌云[2,3],翟环欣[1]

(1. 信阳师范大学 计算机与信息技术学院,河南 信阳 464000;
2. 中国科学院 计算机网络信息中心,北京 100083;3. 中国科学院大学,北京 100049)

摘　要:研究开源计算流体力学软件 OpenFOAM 中典型不可压求解器 icoFoam 在异构平台上的并行算法,针对 OpenFOAM 设计了多异构平台兼容的数据存储结构,并结合可移植异构计算接口(HIP),提出了在两种主流异构平台上移植 icoFoam 的方案;同时,提出了一种在 icoFoam 中使用 Krylov 子空间线性求解器时高效的矩阵格式转换算法,并耦合 hipSPARSE 和 hipBLAS 库实现了压力和速度方程的高效求解。实验表明,该统一的移植方案在 AMD 和 NVIDIA 异构计算平台上,均使得 icoFoam 求解器的计算效率得到较大幅度的提升,相比于 CPU,可分别获得 20.5 倍和 38.4 倍的加速效果。

关键词:异构计算平台;计算流体力学;可移植异构计算接口(HIP);icoFoam 求解器

中图分类号:TP391　　**文献标识码**:A　　**开放科学(资源服务)标识码(OSID)**:

Porting Algorithm and Optimization of Incompressible Solver for Multi-heterogeneous Platforms

MA Wenpeng[1*], LI Ruiying[1], YUAN Wu[2,3], GAO Lingyun[2,3], ZHAI Huanxin[1]

(1. College of Computer and Information Technology, Xinyang Normal University, Xinyang 464000, China;
2. Computer Network Information Center, Chinese Academy of Sciences, Beijing 100083, China;
3. Chinese Academy of Sciences University, Beijing 100049, China)

Abstract: The parallel algorithm of icoFoam, a typical incompressible solver of open-source Computational Fluid Dynamics software OpenFOAM, on heterogeneous platforms was studied. A multi-heterogeneous platforms compatible data storage structure was designed for OpenFOAM, and a scheme for porting icoFoam on two mainstream heterogeneous platforms was proposed in conjunction with the Heterogeneous-Computing Interface for Portability (HIP). Meanwhile, an efficient matrix format conversion algorithm was proposed when using the Krylov subspace linear solver in icoFoam, and the coupled hipSPARSE and hipBLAS libraries were implemented to solve the pressure and velocity equations efficiently. Experimental results showed that this unified porting scheme improved the computational efficiency of the icoFoam solver more substantially on both AMD and NVIDIA heterogeneous computing platforms, obtaining 20.5 and 38.4 times acceleration compared to the CPU, respectively.

Key words: heterogeneous computing platforms; computational fluid dynamics; Heterogeneous-Computing Interface for Portability(HIP); solver of icoFoam

0　引言

在工程计算领域,空气动力、航空航天、热能动力、天文物理等复杂流动问题大都可以采用计算流体力学(Computational Fluid Dynamics,CFD)的方法进行数值模拟,有效解决传统流体力学实验带来的巨大功耗问题。OpenFOAM(Open Source Field Operation and Manipulation,OpenFOAM)[1]是

收稿日期:2022-11-01;**修回日期**:2023-02-03;*.**通信联系人**,E-mail: mawp@xynu.edu.cn

基金项目:国家重点研发计划项目(2020YFB1709500);河南省重点研发与推广专项(科技攻关)项目(222102210162);光合基金项目(ghfund202202011058);河南省高等教育教学改革研究与实践项目(研究生教育)(2021SJGLX057Y);河南省研究生课程思政示范课程项目(YJS2023SZ23)

作者简介:马文鹏(1986—),男,河南信阳人,副教授,博士,硕士生导师,主要从事高性能计算、GPU 并行计算研究。

一个完全由C++编写的CFD软件,其前后处理功能强大、富含多种物理模型且版本更新快速,被广泛应用于各个工程领域。尤其近年来高性能计算(High Performance Computing,HPC)的迅速发展,为解决大规模工程实际问题提供了硬件支撑和有效的可编程实现手段,利用HPC强大的并行算力来提高OpenFOAM的性能已经成为主流趋势,愈来愈多的研究人员开始使用异构众核架构来改进CFD传统的数值计算方法。

VAN PHUC等[2]基于OpenFOAM对风场和自由面流场的模拟进行了多项研究,并使用日本超级计算机"京"成功生成了高达千亿单元网格的大规模瞬态CFD模拟,验证了千亿单元网格和数十万MPI(Message Passing Interface,MPI)的并行潜在能力。LIN等[3]针对OpenFOAM中的线性求解器提出了一种改进的预处理共轭梯度算法(Preconditioned Conjugate Gradient,PCG)和基于对角线的不完整Cholesky预处理器方法,在神威超算平台的单核组上分别实现了8.9倍和3.1倍的速度提升。REN等[4]提出3种减少通信操作的方案对PCG算法进行优化,基于超级计算平台天河二号验证了OpenFOAM的通信优化性能,使大规模线性方程组迭代求解的效率显著提升。ROJEK等[5]提出一种与人工智能模块集成的稳态CFD数值模拟方法,基于OpenFOAM工具箱MixIT,实现了卷积神经网络监督学习算法在GPU异构计算平台上10倍左右的加速。吴家明等[6]基于PETSc(Portable, Extensible Toolkit for Scientific Computation,PETSc)改进线性方程组求解模块,并将其集成到OpenFOAM框架中,验证了基于新PETSc框架的线性方程组在百万级大网格规模下的并行效率。孙士丽等[7]基于OpenFOAM针对三体船连接桥自由落体入水砰击模型进行数值模拟,验证了数值计算方法的正确性以及三体船连接桥入水砰击压力特性和范围,为三体船结构强度评估与结构设计提供了数值基础。HE等[8]提出一种面向对象的辅助应用框架,分别对OpenFOAM的原始求解器和湍流模型实现了高效优化辅助设计。

近年来,除了研究OpenFOAM的数值模拟计算方法外,针对其在异构架构上的性能扩展研究也日益增多,但异构架构上的移植工作多着眼于多节点MPI通信集合的性能提升。有些学者将求解器中线性解法器模块作为重点,研究其在异构架构上的大规模并行效率和优化,或是设计应用接口模型为其在异构架构上的移植扩展提供辅助分析。目前,一个完整的求解器框架在CPU+GPU加速器异构计算平台上的并行方案仍然存在一些挑战。主要表现在:一是OpenFOAM完全由C++语言开发,且为了此软件开发和维护的灵活性,不断采用C++的最新且高级的语言特性,这会带来异构平台上编译器的兼容性问题;二是CPU+GPU的异构软硬件生态成多样的发展趋势,如NVIDIA的GPU硬件平台搭配计算统一设备架构(Compute Unified Device Architecture, CUDA)[9]、AMD的GPU搭配ROCm(Radeon Open Compute, ROCm)[10],给同一求解器或算法在不同异构平台上的移植和适配带来额外的成本;三是C++中封装层次较深,底层数据不可见,给异构移植时一些面向底层数据的优化带来困难。如应用程序使用Field类表示CFD中各种物理场,对应于面向过程语言中的一维或高维数组,这些数据结构被封装在底层,无法轻易进行结构数组(Array of Structures, AoS)、数组结构(Structure of Arrays,SoA)[11]方面的访存优化。另外,软件的迭代和更新,也给某些模块的移植提供了思路。如在2020年6月发布的OpenFOAM-v2006版本[12],在求解线性系统方面对接了PETSc软件,使得PETSc中丰富的基于Krylov子空间[3]的线性解法器可以用于OpenFOAM的求解中,其中矩阵数据类型和PETSc数据类型的转化接口也为其外接异构线性求解器提供了借鉴。

对于上述问题,本文重点针对OpenFOAM的Field类和lduMatrix类设计了适应异构平台的、可灵活操作底层数据的结构,并通过数据转换接口实现两种数据的转换;采用可移植异构计算接口HIP(Heterogeneous-Compute Interface for Portability)[10]完整实现了不可压求解器icoFoam在两类主流异构平台(NVIDIA和AMD)上的移植和性能分析;针对求解过程中产生的线性方程组,在异构平台上设计了LDU(Lower Diagonal Upper)格式到CSR(Compressed Sparse Row)格式[13]的转换方案,并耦合hipSPARSE和hipBLAS[14]实现了基于Krylov子空间方法的线性求解器。

1 控制方程和PISO算法介绍

icoFoam是OpenFOAM提供的用来求解不

可压缩层流的求解器,其对应的控制方程为:

$$\nabla \cdot \boldsymbol{U}=0, \tag{1}$$

$$\frac{\partial \boldsymbol{U}}{\partial t}+\nabla \cdot (\boldsymbol{U}\boldsymbol{U})-\nabla \cdot (\nu \nabla \boldsymbol{U})=-\nabla \boldsymbol{p}, \tag{2}$$

式中:$\boldsymbol{U}$ 是速度矢量,$\boldsymbol{p}$ 是压力,ν 是黏度系数。式(1)是连续性方程,式(2)是动量方程。

icoFoam 在非结构网格上采用有限体积方法离散控制方程,使用 PISO(Pressure Implicit with Splitting Operator,PISO)算法[15]在每个时间步内实现压力和速度的多次相互修正来获得下一个时间步的压力和速度。具体的流程如图 1 所示。

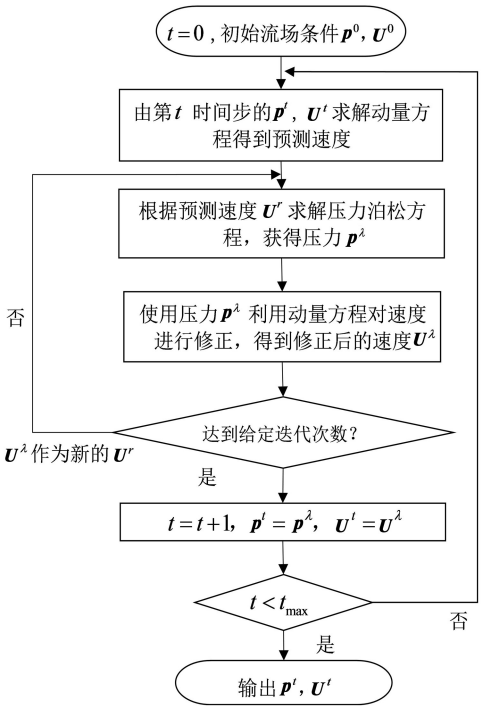

图 1　PISO 算法流程图

Fig. 1　Flow chart of PISO algorithm

在当前时间步内,将 t_0 时刻流场的压力初值 $\boldsymbol{p}^0$ 和速度初值 $\boldsymbol{U}^0$ 代入公式(2)中,PISO 循环开始。首先数值求解动量方程得到预测速度 $\boldsymbol{U}^r$,然后使用压力来表示速度,将速度代入连续性方程即公式(1)中,得到压力泊松方程,由预测速度 $\boldsymbol{U}^r$ 求解该方程,得到压力 $\boldsymbol{p}^\lambda$,继而根据新的压力 $\boldsymbol{p}^\lambda$ 对速度进行修正,得到修正后的速度 $\boldsymbol{U}^\lambda$。如果在当前时间步内未达到预期的迭代次数,则将 $\boldsymbol{U}^\lambda$ 当作新的速度值 $\boldsymbol{U}^r$ 作为下一次迭代的初始速度,当满足迭代次数时,t_0 时刻的 PISO 循环结束,跳出当前时间步,进入下一时刻继续求解压力和速度的耦合,直至最后时刻,输出最终压力 $\boldsymbol{p}^t$ 和速度 $\boldsymbol{U}^t$。

2　并行方案设计

为了兼容主流 NVIDIA 和 AMD 的异构计算平台,图 2 展示了本文利用可移植的异构并行编程模型 HIP 设计 icoFoam 异构求解器的整体方案。

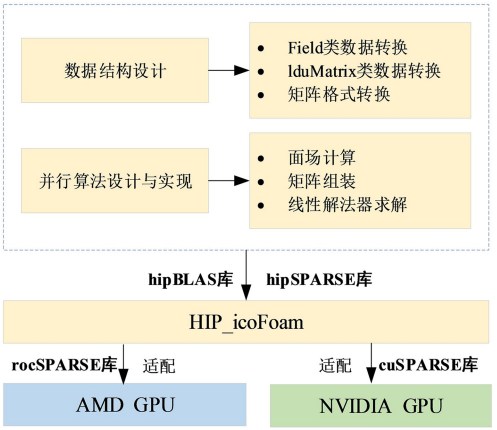

图 2　整体方案设计

Fig. 2　Overall scheme design

从数据操作的角度出发,将 icoFoam 的计算分为两大类:一类是关于物理场的操作,对应于 Field 类,如插值、通量计算、边界条件以及关于某个场数据的线性操作;另外一类是关于矩阵的操作,对应于 lduMatrix 类,如在矩阵组装成功后进行线性系统的求解,包括矩阵向量乘等。考虑到 C++类对底层操作数据的封装性,通过设计对应的包含 CPU 端和 GPU 端数据的 C 结构,用于底层数据的优化。同时,设计了数据结构的转换接口,以保证异构计算之外的其他部分,如网格和流场数据的 IO、流场数据的初始化工作、非结构网格拓扑相关计算操作均保留 OpenFOAM 的原始特性。

OpenFOAM-v2006 版本首次利用其强大的二次开发功能和动态链接库机制对接了 PETSc,使得外部丰富高效的 Krylov 子空间算法能够提供线性解法器。本文充分利用这一机制以及 hipSPARSE 支持多个后端异构平台的特性,设计了基于 hipSPARSE 的 Krylov 子空间方法的解法器,通过在数学库函数上层封装一层 hipSPARSE 的方式,使编译器 hipcc 在编译调用的数学库程序时,能够针对一套 HIP 程序在不同异构平台上链接对应的 SPARSE 库,如 cuSPARSE 或 rocSPARSE[14]。

2.1　非结构网格数据结构转换

OpenFOAM 中包含许多重要的类模板,考虑到类模板对底层操作数据的封装性,针对 icoFoam 求解器的 Field 和 lduMatrix 两种核心类,将程序的核心代码利用 HIP 编程模型重新实现,其余部分仍以 C++为基础。

关于物理场的操作,其核心是对 Field 类的操

作，如梯度计算、速度通量计算、边界条件以及关于某个场数据的线性操作，均是对 Field 类的操作。另外一类是关于矩阵的操作，对应于 LduMatrix 类，如对矩阵的组装、矩阵存储格式的转换以及矩阵组装成功后对线性系统的求解，包括矩阵向量乘等。Field 和 LduMatrix 类都继承于 List 类，相当于对数组的操作，经过多层次的封装和继承，底层数据对用户均不可见，需要提取最底层的核心数据。因此基于 HIP 编程模型设计了数组结构的存储格式和数据接口用以底层数据的转换。如图 3 所示，以体场 volFields 类为例，设计同时包含 CPU 端和 GPU 端数据的结构以及数据转换接口，用于底层数据的转换，从而提高求解器的访存优化性能。

```
//数据转换接口
SetVolVectorField(hip_volFields,volFields)
{
 //判断物理场数据类型
 if (type != VECTOR_TYPE) {...};
 //提取内场和边界场数据到 host 端
 h_idata[] = of_field()[];
 h_bdata[][] = of_field.boundaryField()[][];
 //拷贝数据至 device 端
 hipMemcpy(d_*, h_*, ..., hipMemcpyHostToDevice);
}
```

图 3　Field 类数据格式转换

Fig. 3　Field class data format conversion

2.2　速度通量并行求解算法

Field 类的核心工作是通过通量计算、插值计算和通量修正 3 个阶段实现对物理场 phiHbyA 的求解，最终返回 1 个面场，3 个函数的共性是将网格格心数据插值为网格面数据。

如图 4 所示，OpenFOAM 中通常将体场变量存储在网格格心，计算之前采用中心差分法将所有网格面对应的两个控制单元 owner(简称 O)和 neighbour(简称 N)的体场变量插值到面上。数据由体场到面场的操作中，每个网格面是独立的，由于网格的边界场数据不用进行插值操作，所以在计算过程中将内场和边界场分开处理，并组织与内面数量相等的线程数同时计算内场和边界场。

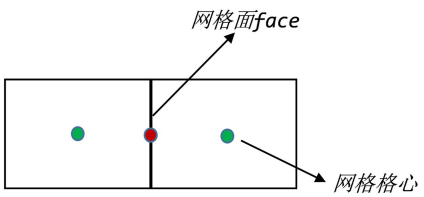

图 4　网格格心数据的插值

Fig. 4　Interpolation of grid cell center data

在线程组织方面，以网格内部面为单位来分配线程任务，假设网格内面的数量共有 nIFaces 个，那么有 nIFaces 个线程以 blockSize 的数量来分块，线程块数为⌈nIFaces/blockSize⌉个。在算法 1 中，由 Vol 表示体场，Sur 表示面场，每个网格面上都有一个线程来实现内场和边界场上的体场计算面场的操作。

算法 1：体场计算面场算法和并行实现

输入：Vol_U，Sur_Sf，插值系数 w，所有网格面的 O，N

输出：内场 Sur_phi_i、边界场 Sur_phi_b

1. tid← blockDim. x * blockIdx. x + threadIdx. x
2. if tid < InFaces then
3. $Sur\_U_i[tid] \leftarrow w * (Vol\_U_i[O[tid]] + Vol\_U_i[N[tid]])$
4. $Sur\_phi_i[tid] \leftarrow Sur\_Sf_i[tid] \cdot Sur\_U_i[tid]$
5. endif
6. if tid < nBFaces[0], [1], [2], …… then
7. $Sur\_phi_b[tid] \leftarrow Sur\_Sf_b[tid] \cdot Vol\_U_b[tid]$
8. endif
9. return Sur_phi_i, Sur_phi_b

如图 5 所示，Field 类的矢量场在 CPU 中均采用典型的 AoS 模式来存储其空间上相邻的数据分量，例如 x、y 和 z。这种顺序访存模式在 GPU 中导致访问单个类型变量时至少产生 3 倍的时间开销。因此计算过程中使用 SoA 结构将矢量场分量每一字段中所有的值存储到各自的数组中，使线程能够顺序且连续地访问数据，更高效地利用 GPU 的全局内存。

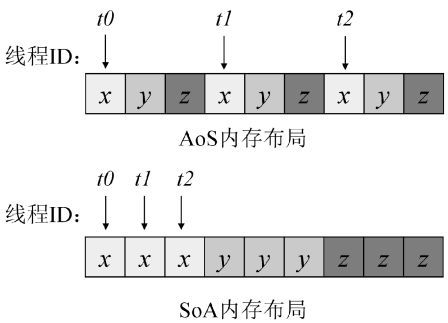

图 5　AoS 和 SoA 模式的内存布局

Fig. 5　Memory layout for AoS and SoA patterns

2.3　矩阵组装并行算法与实现

如图 6 所示，OpenFOAM 中的矩阵采用 LDU 的格式来存储线性方程组的矩阵系数：lower(简称 l)、diag(简称 d)和 upper(简称 u)分别对应下三角、对角线和上三角。其中，将网格内面对应的单元 O 和 N 代替行列索引 upperAddr 和 lowerAddr 作为矩阵上下 2 个三角元素的索引值共用的 2 个索引向量[8]。

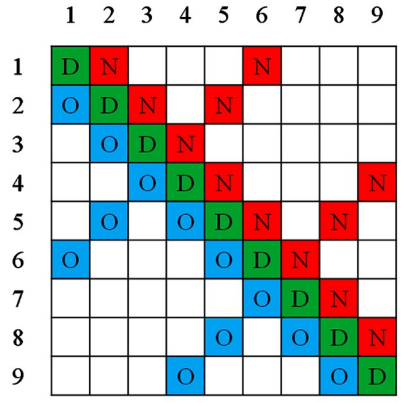

图 6　OpenFOAM 中的矩阵存储格式

Fig. 6　Matrix storage format in OpenFoam

icoFoam 求解器中，速度场和压力场对应的线性方程组信息分别存储在速度方程 UEqn 和压力方程 PEqn 中，离散后都会生成一个 LDU 格式的矩阵用于求解线性方程组。针对与速度方程 UEqn 相关的矩阵和源项组装，将其分为一阶时间导数项、对流项以及扩散项 3 个阶段。

一阶时间导数项中，基于网格单元求解对角线系数和源项；对流项和扩散项中，基于网格内面计算矩阵系数，基于网格单元计算边界贡献。因此依据网格特性来组织线程，例如以 nIFaces 个线程、⌈nIFaces/blockSize⌉个线程块同时计算矩阵系数和边界贡献。在计算对角线系数时，为了避免上、下三角系数取到同一个对角线的值而引起访问冲突，利用原子操作使读—计算—写能够连贯执行，确保只有当前线程访问计算对角线系数所需资源，不受其他线程干扰。

针对 UEqn，设计算法 2 来组装矩阵系数和源项，pEqn 中仅包含扩散项，UEqn 中扩散项的组装方法同样适用于 PEqn。

算法 2：组装矩阵系数算法和并行实现

输入：1. 网格体积 C，系数 r，w，速度 U_{t-1}，通量 phi

输出：矩阵系数 l，d，u、源项 S、边界贡献 Ic、Bc

```
1.  tid← blockDim.x * blockIdx.x + threadIdx.x
2.  if tid < nCells then
3.   d[tid] ← r * C[tid]
4.   S[tid] ← r * U_{t-1}[tid] * C[tid]
5.  endif
6.  if tid < InFaces then
7.   l[tid] ← w * phi[tid]
8.   u[tid] ← l[tid] + phi[tid]
9.   atomicAdd(&d[N[tid]],(−l[N[tid]])
10.  atomicAdd(&d[O[tid]],(−u[O[tid]])
11.  endif
12.  if tid < PatchSizes[0],…… then
13.  Ic[tid]← 0.0
14.  Bc[tid] ← phi[tid] * U_{t-1}[tid]
15.  endif
16.  return l, d, u, S, Ic, Bc
```

2.4　线性求解器并行算法与实现

OpenFOAM 中关于流场的核心运算是通过网格面来组织的，如图 6 所示，其计算流程中线性系统的系数矩阵的存储格式也与面单元的拓扑结构一一对应。LDU 格式的矩阵表示与以面为单位组织计算的特点非常契合，但是由于矩阵数据被拆分存储，也给在加速平台上矩阵按照整行(整列)为单位的操作(如稀疏矩阵向量乘操作)带来了不便，同时整行数据的分散存储阻碍了数据局部性、访存融合的优化。在加速平台上，提出了一种 LDU 到 CSR 格式的转换算法，且基于 hipSPARSE 库，针对速度方程和压力方程，耦合 BiCGStab 和 PCG 两种 Krylov 子空间方法作为线性系统的解法器。

算法 3 描述了该转换算法的过程，所有操作均在加速器上完成。

算法 3：LDU 格式到 CSR 格式的转换算法和并行实现

输入：LDU 格式的矩阵上、下三角、对角数据：

```
1.  u, l([0:nIFaces−1]), d([0:nCells−1])
2.  所有网格面的 O 和 N, ([0:nIFaces−1])
```

输出：CSR 格式的矩阵：csr_r, csr_c, csr_v

```
1.  将 LDU 格式合并成非排序的 coo 格式：
    tid← blockDim.x * blockIdx.x + threadIdx.x
    if tid < nIFaces then
    (coo_r, coo_c, coo_v)[tid] ← (N, O, l)[tid]
    endif
    if tid < nCells then
    (coo_r, coo_c, coo_v)[nIFaces+tid] ← (tid, tid, d[tid])
    endif
    if tid < nIFaces then
    nIFC ← nIFaces + nCells
    (coo_r, coo_c, coo_v)[tid + nIFC] ← (O, N ,u)[tid]
    endif
2. 将 Coo 格式按照行压缩排序，得到置换 P：
    P ← hipSparseXcooSortByRow(coo_r, coo_c)
3. 将置换应用到 coo_v 得到排序后的矩阵：
    sorted_coo_v ← hipSparseDgthr(P, coo_v)
4. 将 coo_r 按照行压缩存储得到 csr_r：
```

csr_r ← hipsparseXcoo2csr(coo_r)

5. csr_c ← coo_c; csr_v ← sorted_coo_v

6. return csr_r, csr_c, csr_v

算法 3 中，首先由 LDU 格式的存储规则，可以获得矩阵每个元素的行号和列号，发起一个 kernel 函数，将矩阵的下三角、上三角和对角元素的行号、列号及元素的值合并到一个未排序的 COO 格式存储的矩阵中；随后的任务就转化为将 COO 格式的矩阵按照行压缩进行排序，基于 hipSPARSE 库函数，此部分分为两个步骤：第一步是通过 hipsparseXcooSortByRow 获取一个从未排序到按行压缩排序的置换；第二步将置换应用到原始的 COO 格式的矩阵的元素中，得到转换过后的元素；最后，将 COO 格式表示的行信息 coo_r 压缩得到 CSR 格式表示的行信息 csr_r。该算法最耗时的部分是得到置换数组 P 的过程，属于访存密集型操作，但是该步骤在针对由速度或者压力方程得到的系数矩阵时，只需要执行一次，因为在整个时间迭代过程中，每个时间步对应的系数矩阵非零元的位置是保持不变的。在得到 CSR 格式存储的矩阵之后，基于 hipSPARSE 和 hipBLAS 库，实现了 Krylov 子空间的 PCG 和 BiCGStab 两个线性解法器，用于求解压力和速度对应的线性系统。

3 测试环境及算例

3.1 测试环境

实验是在分别包含 AMD gfx906 GPU 和 NVIDIA TeslaV100 GPU 的两个异构计算节点上进行的。其中，节点 1 采用的是 2.5 GHz 的 CPU 主频处理器，共有 128 GB 内存，并配置 4 块 16 GB gfx906 GPU 加速器，rocm 版本为 4.3(dtk 21.10)。节点 2 采用的是 2.4 GHz 的 Intel(R) Xeon(R) E5-2640 v4 CPU 主频处理器，共有 128 GB 内存，并配置 4 块 16 GB 的 TeslaV100 显卡，CUDA 版本为 10.0。

3.2 算例设置

实验使用 OpenFOAM-v2006 版本的不可压缩层流顶盖驱动方腔流(Lid-driven Cavity Flow，Cavity)[1]为测试算例，如图 7 所示。该算例通过将 z 轴方向的边界条件设置为 empty 类型来实现二维的计算。

针对 Cavity 算例在 z 轴方向进行扩展，将 front 和 back 的边界条件从原始的 empty 类型设置为 symmetryPlane 类型，movingWall 与 fixedWalls 的边界条件保持不变，其网格控制文件 blockMesh 的设置如表 1 所示。

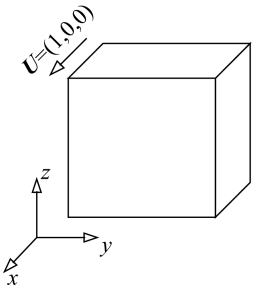

图 7 Cavity 算例模型

Fig. 7 Example model of Cavity

表 1 Cavity 中 blockMesh 设置

Tab. 1 BlockMesh setting of Cavity

参数名称	参数设置
scale	0.1
movingWall	wall
fixedWalls	wall
front	symmetryPlane
back	symmetryPlane

在使用 Cavity 算例测试时，同时将 x、y 和 z 轴方向扩展为 20 至 40，60，…，100，设置算例运行时间为 1×10^{-4} s，时间步长为 1×10^{-6} s，共迭代 100 个时间步。在测试过程中关闭 I/O 操作，避免流场信息的频繁写入、写出影响测试结果的稳定性。

4 实验结果与分析

采用 HIP 实现的 icoFoam 的同一套不可压异构求解器，分别在 ROCm 和 CUDA 两个平台上进行测试。与 2.2、2.3、2.4 小节对应，在测试过程中分别统计这三部分在加速器上的时间，并与 icoFoam 在 CPU 上的时间进行对比分析。CPU 版本的串行程序在编译过程中使用了 −O3 优化选项。为了保证对比实验的有效性，使用 CPU 最优的测试时间和 GPU 测试时间计算加速比，且为了避免测试的偶然性，统计测试时间使用连续 3 次在 GPU 上测试时间的平均值。

表 2 给出了不同网格规模下分别在 ROCm 和 CUDA 平台上面场相关操作的时间(t_{ROCm}、t_{CUDA})和对应的 CPU 上时间(t_{CPU})的对比。由表 2 的实验数据可以看出，经过 100 个时间步的迭代后，不同网格规模下的 ROCm 平台计算面场的算法加速效果在 3.7～66.0 倍。CUDA 平台加速效果在 5.4～87.2 倍，随着网格规模的增大，加速比呈现递增趋势。说明与 icoFoam 串行计算相比，GPU

的并行计算方法具有良好的加速效果,时间计算成本显著降低。

表 2　面场串行和并行计算时间对比

Tab. 2　Comparison of serial and parallel computation time of surface field

网格大小	t_{CPU}/ms	t_{ROCm}/ms	ROCm 加速比/倍	t_{CUDA}/ms	CUDA 加速比/倍
20×20×20	83.0	22.5	3.7	15.4	5.4
40×40×40	669.6	83.1	8.1	56.3	11.9
60×60×60	2 780.9	112.0	24.8	81.3	34.2
80×80×80	7 247.5	173.4	41.8	124.8	58.1
100×100×100	14 981.1	227.1	66.0	171.7	87.2

表 3 给出了不同网格规模下组装系数矩阵在各平台上的对比。根据表 3 的实验结果,发现 GPU 的矩阵组装并行方案加速效果明显,在 ROCm 计算平台上的加速比为 2.6～43.3 倍,CUDA 计算平台上的加速比为 5.4～66.4 倍。值得注意的是,算法 2 在进行 LDU 格式矩阵系数组装时,使用了双精度原子操作,尽管如此,相比于 CPU 的时间,原子操作的实现依然有较大的并行优势,这主要得益于原子操作在两个平台上的高效实现。

表 3　矩阵组装串行和并行计算时间对比

Tab. 3　Comparison of serial and parallel computation time of matrix assembly

网格大小	t_{CPU}/ms	t_{ROCm}/ms	ROCm 加速比/倍	t_{CUDA}/ms	CUDA 加速比/倍
20×20×20	42.8	16.2	2.6	7.9	5.4
40×40×40	360.3	38.3	9.4	23.1	15.6
60×60×60	1 691.4	91.6	18.5	63.6	26.6
80×80×80	4 450.6	120.0	37.1	72.0	61.8
100×100×100	9 796.5	226.4	43.3	147.5	66.4

表 4 给出了算法 3 中从 LDU 格式到 CSR 格式的转换时间,由于在时间迭代过程中,两种格式在非零元位置的对应关系上保持不变,因此,算法 3 中得到置换的操作仅需要执行一次即可。表 5 和表 6 给出了分别使用 PCG 和 BiCGStab 求解一次压力和某个速度分量相关的线性系统的时间,虽然矩阵格式转换时间(20 ms 左右)相比于求解一次线性系统的时间并不占优势,但是由于转换在整个求解流程中仅需一次,而每个时间步都需要求解一个线性系统,因此,矩阵转换时间相对于整个计算流程时间,如计算 100 个时间步,在 1%左右,并不会影响整体计算效率。

表 4　LDU 格式到 CSR 格式的转换时间统计

Tab. 4　Conversion time statistics of PCG algorithm LDU format to CSR format

网格大小	t_{ROCm}/ms	t_{CUDA}/ms
20×20×20	0.8	1.3
40×40×40	3.8	2.1
60×60×60	10.8	6.2
80×80×80	20.3	11.6
100×100×100	22.1	21.3

表 5 和表 6 的实验结果表明,基于 hipSPARSE 和 hipBLAS 实现的两个 Krylov 子空间的线性解法器能够同时支持 ROCm 和 CUDA 平台,并能获得 18 倍和 37 倍的加速效果。由于求解线性系统占整个计算的大部分时间,因此对于 icoFoam,整体在异构平台上的加速分别为 20.5 倍和 38.4 倍。

表 5　基于 PCG 算法的线性方程组串行和并行计算时间对比

Tab. 5　Comparison of serial and parallel computing time of linear equations based on PCG algorithm

网格大小	t_{CPU}/ms	t_{ROCm}/ms	ROCm 加速比/倍	t_{CUDA}/ms	CUDA 加速比/倍
20×20×20	4.2	13.3	0.3	4.7	0.9
40×40×40	38.1	14.9	2.6	6.8	5.6
60×60×60	152.1	17.8	8.6	8.8	17.2
80×80×80	343.2	23.8	14.4	12.7	27.0
100×100×100	753.6	34.8	21.7	20.1	37.5

表 6　基于 BiCGStab 算法的线性方程组串行和并行计算时间对比

Tab. 6　Comparison of serial and parallel computing time of linear equations based on BiCGStab algorithm

网格大小	t_{CPU}/ms	t_{ROCm}/ms	ROCm 加速比/倍	t_{CUDA}/ms	CUDA 加速比/倍
20×20×20	4.0	11.7	0.3	6.3	0.6
40×40×40	36.6	13.0	2.8	7.6	4.8
60×60×60	154.9	17.0	9.1	9.4	16.5
80×80×80	411.3	24.5	16.8	15.1	27.3
100×100×100	698.2	37.8	18.5	18.9	37.0

5　结论

本文面向 CPU＋GPU 众核处理器使用可移植性跨平台接口(HIP)实现了不可压缩求解器 icoFoam 在两类主流异构平台 NVIDIA 和 AMD 上的移植,基于异构众核加速器测试了不同算例规模下的求解器并行算法的计算时间。经过与

OpenFOAM 串行程序的对比分析，基于两类异构计算平台的整个求解器并行算法的设计，能够大幅度提升 OpenFOAM 的计算效率，大规模并行计算能力的提升分别高达 20.5 倍和 38.4 倍。

参考文献：

[1] ZHANG Y, LU X B, ZHANG X H. An optimized Eulerian-Lagrangian method for two-phase flow with coarse particles: Implementation in open-source field operation and manipulation, verification, and validation[J]. Physics of Fluids, 2021, 33(11): 113307.

[2] VAN PHUC P, CHIBA S, MINAMI K. Large scale transient CFD simulations for buildings using OpenFOAM on a world's top-class supercomputer[C]//The 4th Annual OpenFOAM User Conference 2016, Cologne: ESI Group, 2016: 1-61.

[3] LIN J, WEN Minhua, MENG Delong, et al. Optimizing preconditioned conjugate gradient on TaihuLight for OpenFOAM [C]//2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Washington, DC, USA: IEEE, 2018: 273-282.

[4] REN Xiaoguang, ZHOU Wenhao, CHEN Juan. Collective communication optimization for solving linear algebraic equations[J]. Advanced Materials Research, 2014, 989: 4934-4939.

[5] ROJEK K, WYRZYKOWSKI R, et al. AI-accelerated CFD Simulation Based on OpenFOAM and CPU/GPU Computing[C]//International Conference on Computational Science. Springer, Cham, 2021: 373-385.

[6] 吴家明，杨凯，黎展鹏，等. 在 OpenFOAM 中集成 PETSc 并验证大规模并行效率[C]//第十九届中国空气动力学物理气体动力学学术交流会摘要集，张掖：中国空气动力学学会物理气体动力学专业委员会，2019：99.
WU Jiaming, YANG Kai, LI Zhanpeng, et al. Integrate PETSc in OpenFOAM and verify massively parallel efficiency[C]//The 19th China Aerodynamics Physical Gas Dynamics Academic Exchange Conference, Zhangye: Physical Gas Dynamics Professional Committee of the Chinese Aerodynamics Society, 2019: 99.

[7] 孙士丽，陈雪龙. 基于 OpenFOAM 的大型三体船连接桥入水砰击载荷数值模拟分析[J]. 中国舰船研究，2022，17(S1)：43-51.
SUN Shili, CHEN Xuelong. Numerical investigation on water-entry slamming load of large trimaran connecting bridge using OpenFOAM[J]. Chinese Journal of Ship Research, 2022, 17(S1): 43-51.

[8] HE Ping, MADER C A, MARTINS J R R A, et al. An object-oriented framework for rapid discrete adjoint development using OpenFOAM[C]// Aerodynamic Design: Analysis, Methodologies, and Optimization Techniques III, San Diego: American Institute of Aeronautics and Astronautics, 2019: 1210.

[9] SIANO M , GELONI G , PAROLI B , et al. FOCUS: Fast Monte Carlo approach to coherence of undulator sources [J]. Journal of synchrotron radiation, 2023, 30(1): 217-226.

[10] WANG Penfei, JIANG Jinrong, LIN Pengfei, et al. The GPU version of LASG/IAP climate system ocean model version 3 (LICOM3) under the heterogeneous-compute interface for portability (HIP) framework and its large-scale application[J]. Geoscientific Model Development, 2021, 14(5): 2781-2799.

[11] KEHL C, NOOTEBOOM P D, KAANDORP M L A, et al. Efficiently simulating Lagrangian particles in large-scale ocean flows-data structures and their impact on geophysical applications[J]. Computers & Geosciences, 2023, 175: 105322.

[12] BOROLE A, THIAGARAJAN K B. Numerical study of effects on vortex shedding patterns due to unsteady free stream around cylinder[J]. AIP Conference Proceedings, 2020, 2311(1): 030035.

[13] BOUMZOUGH K, AZZOUZI A, BOUIDI A. The incomplete LU preconditioner using both CSR and CSC formats [J]. Advanced Mathematical Models and Applications, 2022, 7: 156-167.

[14] TSAI Y M, COJEAN T, ANZT H. Sparse linear algebra on AMD and NVIDIA GPUs: The race is on[C]//High Performance Computing, Cham: Springer international publishing, 2020: 309-327.

[15] HOP C J W, JANSEN R, BESTEN M, et al. Hydrodynamics of a rotor-stator spinning disk reactor: Investigations by large-eddy simulation[J]. Physics of Fluids, 2023, 35(3): 035105.

责任编辑：郭红建