计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210700060

OpenFoam 中多面体网格生成的 MPI+OpenMP 混合并行方法

刘 江[1] 刘文博[1,2] 张 矩[1]
1 中国科学院重庆绿色智能技术研究院 重庆 400714
2 中国科学院大学 北京 100049
(liujiang@cigit.ac.cn)

摘 要 网格生成是计算流体力学中非常重要的一环,大规模数值模拟过程中对网格精度要求的提高会导致网格生成所耗的时间增加。文中基于 OpenFoam 开源软件中的网格生成算法,主要研究多面体网格的并行生成,并提出 OpenMP 和 MPI 混合并行的多面体网格生成方法。通过理论分析得到,使用混合并行方法生成相同质量的网格时,混合并行方法生成网格的时间消耗随着线程数量和网格单元数量的增加而减少。3 组使用不同求解器的数值模拟实验结果表明,该混合并行方法不但可以保证生成网格的质量——可以正常进行数值计算模拟且模拟结果与原方法相比几乎没有差别,而且生成同样质量与数量网格的耗时最多可以缩短至未使用 OpenMP 并行方法之耗时的 1/4 以内。

关键词:计算流体力学;OpenFoam;多面体网格生成;MPI+OpenMP 混合并行;并行效率
中图法分类号 V211.3

Hybrid MPI+OpenMP Parallel Method on Polyhedral Grid Generation in OpenFoam

LIU Jiang[1], LIU Wen-bo[1,2] and ZHANG Ju[1]
1 Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 400714, China
2 University of Chinese Academy Science, Beijing 100049, China

Abstract Grid generation is an important step of computational fluid dynamics. In the process of large-scale numerical simulation, the time consumption of grid generation increases with the number of grids which often increases with the simulation accuracy. Based on the grid generation algorithm in an open-source software called OpenFoam, this paper proposes a hybrid parallel method of OpenMP and MPI for polyhedral grid generation. By theoretical analysis, we show that when the hybrid parallel method is used to generate the same quality grids, increasing the number of threads and grid cells will reduce the time consumption of grid generation. Three numerical simulations using different solvers show that the grids generated by the hybrid parallel method and the original method have close qualifications, and the simulation results are almost indistinguishable from those of the original method. Furthermore, the time consumption of this method to generate the same quality and quantity grids can be reduced to less than a quarter of the time consumption without using OpenMP parallel method.

Keywords Computational fluid dynamics, OpenFoam, Polyhedral grid generation, MPI+OpenMP hybrid parallelization, Parallel efficiency

1 引言

计算流体力学(Computational Fluid Dynamics,CFD)技术已被证明是预测空气动力学性能的可靠工具[1],其具有许多突出的优势,如效率高、应用广泛和模拟数据详细等[2]。在 CFD 的计算仿真过程中,有预处理、求解计算和后处理 3 个模块[3]。预处理部分[4]包括建模、生成网格、设定物理参数等;求解计算部分[5-6]主要是使用求解器在上一步所生成的网格上对物理模型进行求解;后处理部分[7]对求解生成的数值结果进行处理及可视化。在第一步的预处理过程中,优质的网格生成方法可以在保证所生成网格的质量足够好的同时,还能使网格生成的时间消耗比较少。OpenFoam(Open Source Field Operation and Manipulation)是目前被广泛使用的一款计算流体力学开源软件[8-9],与 FLUENT,CFX 等商业软件相比,其优势在于源代码、程序结构及软件架构设计完全开放,用户可以看到其中的核心求解器和工具,并且可以修改

到稿日期:2021-07-06 返修日期:2021-12-06
基金项目:国家重点研发计划(2018YFC0116704);四川省科技计划项目(2020YFQ0056)
This work was supported by the National Key R & D Program of China(2018YFC0116704) and Sichuan Science and Technology Program (2020YFQ0056).
通信作者:刘文博(liuwenbo@cigit.ac.cn)

或自定义各种有效的求解器、辅助工具和库文件，达到个性化应用的目的。在 OpenFoam 中使用 gmsh，snappyHexMesh 等工具就可以完成简单的网格生成预处理工作，但是网格质量和网格生成速度都还有较大的提升空间[10-11]。

在 OpenFoam 中，最常用的多面体网格生成方法是 snappyHexMesh，其具有加密灵活、使用方便等特点。snappyHexMesh 工具可以自动地从 STL，OBJ 文件生成多面体网格。网格依靠迭代将一个初始网格细化，并将细化后的网格变形，以依附于物理模型表面，生成最终用于计算的多面体非正交网格[12]。

近几十年来，随着工程及科学计算所使用的数值计算方法的快速发展，大规模和超大规模并行机等硬件计算能力的增长，解决计算问题的能力也随之提高，大规模并行计算成为了计算流体力学中最重要的研究领域之一[13-14]。而近年来，计算流体力学领域中求解计算部分的并行化研究和推广，使得数值模拟风洞实验实用化。然而，网格生成的低效率会导致在进行大型数值模拟实验时的时间消耗过多。因此从上世纪九十年代起[15]，使用并行手段来有效提高网格生成速度成为研究热点。到现在为止，仍有不少研究人员在对非结构网格[16]和 Delaunay[17]等四面体网格[18]的并行生成方法进行研究。

OpenFoam 使用基于 MPI 库的 OpenMPI 来支持并行计算，这种并行是基于网格的分区来实现的，也有科研人员对这种并行效率进行了研究[19]和改进[20]。在整个 CFD 的数值模拟实验中，前处理、求解计算和后处理这 3 个模块都可以进行并行计算。现阶段对这些部分的并行化处理方法有使用 GPU 生成网格[21]和开发新的求解器[22]，或基于 CPU 并行提出新的网格生成方法[23]和求解器算法[24-25]进行并行加速等。

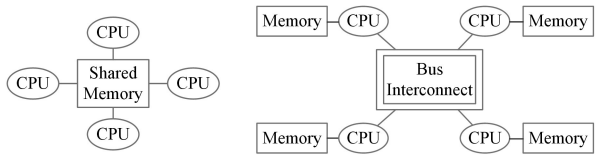
本文在 OpenFoam 基于网格分区并行计算的基础上，使用 OpenMP＋MPI 混合并行方法生成非结构多面体网格，并分析网格生成的效率，再通过数值模拟实验验证这种并行方法所生成的网格对数值模拟的可支持性。使用服务器 CPU：2 * Intel(R) Xeon(R) Gold 5218 CPU @ 2.30 GHz，OpenFoam 版本：v2012，OpenMP 版本：5.0，OpenMPI 版本：4.0.3。

2 基础方法

2.1 并行方法

OpenMP 是一种基于共享内存的多线程并行方案，大部分编译器，如 Sun Compiler，GNU Compiler 和 Intel Compiler 等都支持 OpenMP。但是 OpenMP 的缺点是不能在非共享内存的系统上使用，这种情况下一般会选择使用 MPI 并行。在开源软件 OpenFoam 中，网格生成和求解计算部分均支持 MPI 标准的 OpenMPI 库。使用时，用户可以根据任务需求自由选择串行或并行功能，以及并行时的网格分区配置。OpenFoam 中的并行基于网格分区实现，在并行计算时，进程之间采用 MPI 消息传递机制进行数据传递，每个进程可以处理一个网格分区，在分区边界处需要与相邻网格分区交换数据时进行 MPI 通信。而在每个网格分区内部则与串行时的计算流程一致，没有达到完全并行，因此可以考虑使用 OpenMP 来实现分区内部的完全并行。

OpenMP 基于共享内存实现并行，如图 1(a)所示，多线程在共享内存上进行计算，完成并行任务。如图 2 所示，OpenMP 程序在运行时由一个主线程完成非并行区域的任务，在到达并行区域时，会创建线程池，并在线程池中自动创建多个子线程，完成计算任务退出并行区域时，子线程会处于 Sleeping 状态，主线程保持 Running 状态继续运行，控制整个程序。



(a)共享内存访问 (b)非共享内存访问

图 1 内存访问模型

Fig. 1 Memory access model

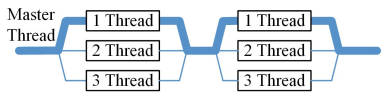


图 2 Fork-Join 模型

Fig. 2 Fork-Join model

MPI 基于 CPU 间的通信实现并行，如图 1(b)所示，因此其适用于分布式系统。OpenMP 和 MPI 策略之间不会产生冲突，可以同时使用，而且在实现 MPI＋OpenMP 的混合并行后，可以达到更快的运行速度。已有研究人员使用混合并行方法，并在超大规模计算任务中达到了 55.3% 的并行效率[26]。

2.2 网格生成方法

snappyHexMesh 是 OpenFoam 软件中被最广泛使用的多面体网格生成器，使用者只需要编写脚本控制文件并设置好所需网格的参数后运行即可生成网格。它的网格生成过程(图片参考文献[12])为：首先以六面体结构网格和给定的物理模型为基础(如图 3(a)、图 3(b)所示)，将物理模型表面附近的网格单元分裂细化，并将 50%以上的体积在物理模型内部的单元删去(如图 3(c)、图 3(d)所示)，其中图 3(d)是图 3(c)中的切面，然后根据物理模型的表面将部分网格单元细化，并将部分网格顶点向几何表面对齐并贴合(见图 3(e))，最后根据需要添加边界层(见图 3(f))。

上述网格生成流程中，将网格单元的顶点向几何表面对齐并贴合的操作比较耗费时间，并且网格单元的数量越多，这部分所耗费的时间就越多。它的步骤如下[27]：

(1)把网格边界(见图 3(d))的顶点移动到模型表面。

(2)根据新的网格点位置，使用松弛算法进行迭代处理[28]，得到重新排布后的网格点。

(3)检查网格质量，并找出使网格质量变差的顶点。

(4)对于步骤 3 中选中的顶点，减少它们在步骤 1 中的位移，并从步骤 2 重新开始计算，直到得到满意的网格质量。

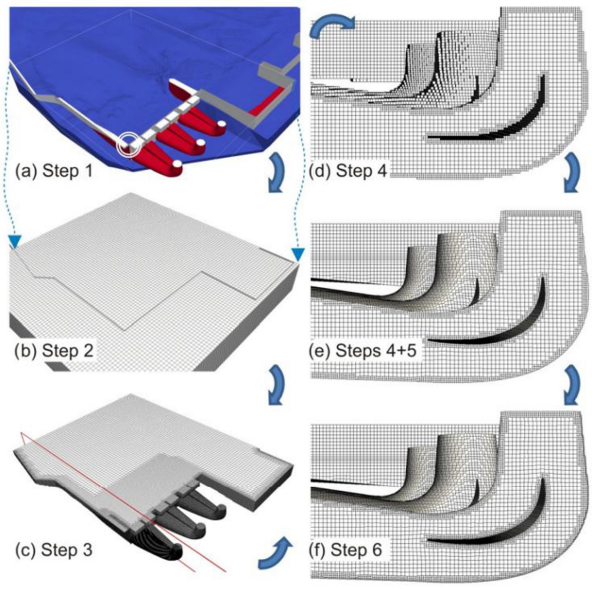


图 3　网格生成步骤

Fig. 3　Grid generation steps

本文基于上述网格生成方法，新编写网格点与物面对齐操作的代码，实现了多面体网格的混合并行生成。

3　MPI＋OpenMP 混合并行方法

在 OpenFoam 中，利用 OpenMP 多线程和共享内存的优势，对网格生成时网格点的循环处理并行化，对单元顶点向物理模型边界对齐并贴合时的位移进行计算，通过有效的并行策略可以减少网格生成的时间消耗。

在网格生成的混合并行方法中，MPI 策略针对网格分区间的并行，OpenMP 策略针对分区内网格点循环处理时的并行，通过混合并行减少时间消耗。本节中，我们首先分析纯 OpenMP 并行的网格生成方法，然后对混合并行方法进行讨论。

3.1　纯 OpenMP 并行

单独使用 OpenMP 并行策略，到达对网格点进行处理的循环部分时，进入并行区域，创建线程池并开辟多线程（如图 2 所示）。在并行区域中对每一个网格单元顶点处的位移计算进行并行（如图 3(e)所示），并采用松弛算法迭代处理[28]，将每一个顶点向几何边界移动并对齐。通过这种方法实现 OpenMP 并行可提高 CPU 的利用率，减少时间消耗。

假设有 h 个线程，n 个点，新建线程池耗时 x_1 秒，每次从线程池取出线程并为其分配计算任务耗时 x_2 秒，计算时的总迭代次数为 I 次。使用原串行方法时，每个网格点处的计算时间为 t_1 秒，使用并行方法时，每个网格点处的计算时间为 t_2 秒。由于在并行区域内线程之间可能会出现对数据的互斥访问，或等待其他线程等情况，因此一般情况下 $t_2 \geqslant t_1$，其余时间消耗忽略不计。原网格生成的时间消耗 T_{old} 和 OMP 并行化后时间消耗 $T_{omppast}$ 如下：

$$T_{old}=I\times(n\times t_1) \tag{1}$$

$$T_{omppast}=I\times\left(\frac{n\times t_2}{h}+x_1+n\times x_2\right) \tag{2}$$

$$T_{old}>T_{omppast}\Rightarrow x_1+nx_2<\frac{n}{h}(ht_1-t_2) \tag{3}$$

定理 1　在上述假设条件下，忽略并行任务的管理和同步、运行时调用函数以及内存和 CPU 被其他进程占用等因素对计算速度的影响，当 $x_1+nx_2<\frac{n}{h}(ht_1-t_2)$ 时，即可满足使用 OpenMP 达到加速效果的目的。

但是这里使用的是 OpenMP 默认的线程分配方案，将循环中的每一步，也就是每一个网格点的位移计算分配给一个线程，这个线程的任务完成后再给它分配下一个循环的计算任务。在整个过程中需要分配 n 次计算任务，将大量的时间消耗在分配任务而不是计算上，加速效果仍有提升的空间。

我们对线程分配方案进行改进，尽量减少并行区域内分配任务的次数，将 CPU 资源留给计算部分。因此，可以将整个循环分成 h 个部分，将每一部分分配给各个线程，即将整个循环中的第$\left[0,\frac{n}{h}\right),\left[\frac{n}{h},\frac{n}{h}\times 2\right),\cdots,\left[\frac{n}{h}\times(h-1),\frac{n}{h}\times h\right)$步分别分配给 h 个线程。优化后的时间消耗估算为：

$$T_{omp}=I\times\left(\frac{n\times t_2}{h}+x_1+h\times x_2\right) \tag{4}$$

$$T_{old}>T_{omp}\Rightarrow x_1+hx_2<\frac{n}{h}(ht_1-t_2) \tag{5}$$

定理 2　在与定理 1 相同的条件下，当 $x_1+hx_2<\frac{n}{h}(ht_1-t_2)$ 时，即可满足使用改进的 OpenMP 方案达到加速效果的目的。

因为一般情况下，网格点数远多于线程数，即 $n\gg h$，可以得到 $x_1+nx_2<x_1+hx_2$，所以当 $n\gg h$ 时，其余参数相同的条件下，定理 2 的条件比定理 1 的条件更容易满足。也就是说，改进后的 OpenMP 并行方案比默认的并行方案更容易获得加速的效果。

由式(4)对 h 求导得：

$$(T_{omp})'=-nt_2h^{-2}I+I\cdot x_2=0 \tag{6}$$

$$\Rightarrow h^2=\frac{nt_2}{x_2} \tag{7}$$

因此在其他参数不变的情况下，当 $h<\sqrt{\frac{nt_2}{x_2}}$ 时，随着 h 的增大，$\frac{T_{omp}}{T_{old}}$ 降低；$h>\sqrt{\frac{nt_2}{x_2}}$ 时，随着 h 的增大，$\frac{T_{omp}}{T_{old}}$ 升高。

因此，随着网格点的数量增多，或线程数量在一定范围内增加，并行生成网格所耗时间减少，加速效果更明显。

由式(2)与式(4)得：

定理 3　$T_{omppast}-T_{omp}=I\times(n-h)\times x_2$ (8)

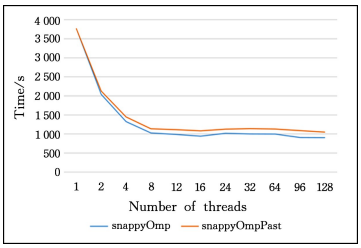
随着 h 的减少或 n 的增加，$T_{omppast}-T_{omp}$ 会增大。不过实际上网格点的数量远远多于线程的数量，因此线程数量的变化对 $T_{omppast}$ 和 T_{omp} 之差影响不大；网格点数量越多，默认并行方案与改进后的并行方案之间的时间消耗相差越大。

算例 1 为螺旋桨在封闭圆柱体空间上部旋转的流场模拟，分别使用 snappyHexMesh 和 snappyOmp 工具生成算例 1 的网格，对它们所消耗的时间进行比较。

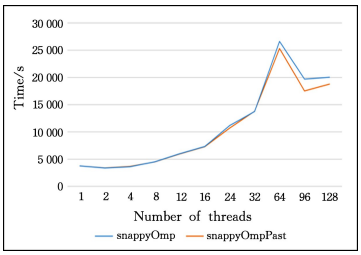
算例 1　OpenFoam 目录：tutorials/incompressible/pimpleFoam/RAS/propeller.

设置基础网格均为 36 * 60 * 36 分辨率的笛卡尔网格，最终生成的多面体网格共有 11 404 617 个单元。只考虑将网格

点对齐至物理模型表面的操作，时间消耗统计如图 4 所示。



(a)系统实际时间消耗



(b)CPU 时间片消耗

图 4 纯 OpenMP 并行时间消耗

Fig. 4 OpenMP parallel time consumption

由于在 OpenFoam 内部所使用的计算时间的函数是通过 CPU 的时间片轮转计算的，在多线程环境下，每个线程都会占有时间片，统计出的不是实际时间消耗，而是多线程共使用的时间，因此需要使用 OpenMP 内置函数 omp_get_wtime() 来计算系统时间。图 4 给出了实际时间和占用 CPU 时间片的统计结果。可以得到，分别使用工具 snappyHexMesh，snappyOmpPast 以及 snappyOmp，可以在使用不同数量的线程时，生成相同网格的时间消耗。

由图 4 可以发现：

(1)在线程数量相同时，改进的 snappyOmp 的实际时间消耗比改进前少，且均比 snappyHexMesh 时间短，符合定理 3。

(2)分别观察改进前和改进后的时间消耗，可以发现，使用同一工具时，随着线程数量的增加，时间先快速减少，到 8 线程后，无法进一步获得更快的加速效果。符合式(7)的推论，且与文献[23]中的结论一致。

(3)在线程数小于 64 时，CPU 时间片的消耗会随着线程数量的增加而成倍增加，而在线程数量达到 64 后，其不会继续增加，但是实际时间消耗也没有减少，这是因为使用的计算机 CPU 具有 64 个逻辑处理核心，在开启超线程后，多余线程只能处于 Sleeping 状态，不参与计算，也不计算时间消耗。

(4)可以观察到随着线程数量的增多，两种网格生成工具的所耗时间之差没有明显变化，符合定理 3。

以下分析了在网格数量不同的情况下，与 snappyHexMesh 相比，snappyOmp 的加速比变化趋势。定义加速比 S 为使用 snappyHexMesh 所耗时间与使用 OMP 方法所耗时间之比：$S=\frac{T_{old}}{T_{omp}}=\frac{nht_1}{nt_2+hx_1+h^2x_2}$，其余参数不变，对 n 求导得：

$$\frac{\partial S}{\partial n}=\frac{h^2t_1(x_1+hx_2)}{(nt_2+hx_1+h^2x_2)^2}>0 \tag{9}$$

$$\frac{\partial^2 S}{\partial n^2}=-\frac{2h^2t_1t_2(x_1+hx_2)}{(nt_2+hx_1+h^2x_2)^3}<0 \tag{10}$$

也就是说，随着网格单元数量的增多，加速比会一直提高，但是提高的速率会降低。同样以算例 1 为例，观察加速比随着网格单元数量增多的变化趋势。如图 5 所示，图中的散点为实验结果，实线为使用最小二乘法对加速比公式进行拟合得到的曲线，可以看出，加速比随网格单元数量变化的趋势。

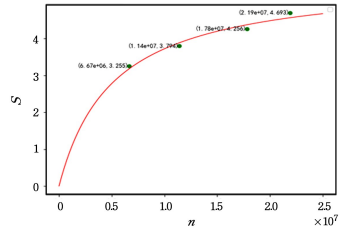


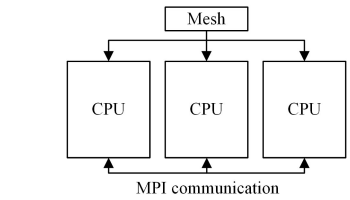
图 5 不同网格单元数量时的加速比

Fig. 5 Speedup with different number of grid cells

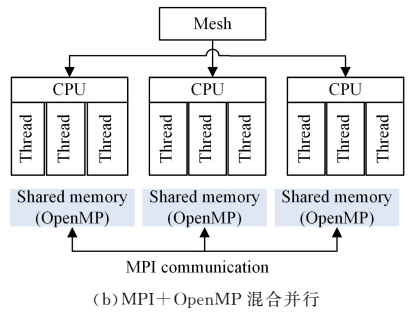
图中的散点无法与曲线完全吻合，这是因为在实际计算过程中计算机的运算速度受各种实际因素的影响。前面的公式推导中也带有很多假设条件，没有考虑并行任务的管理和同步、运行时调用函数等因素占用的时间，但是加速比随单元数量变化的整体趋势与理论分析一致。

3.2 混合并行

OpenFoam 中的网格并行生成是基于网格分区构建的，如图 6(a)所示，将整个基础网格分为多个区域，每个区域再进行网格细分操作。利用 3.1 节中的 OpenMP 并行方法，进行 MPI+OpenMP 混合并行改造，并行生成网格，如图 6(b)所示，依然基于分区进行 MPI 并行，在每个分区内部使用 OpenMP 并行，同一分区内部的多线程并行计算，可以有效提高网格生成的速度。



(a)纯 MPI 并行



(b)MPI+OpenMP 混合并行

图 6 CPU 并行结构

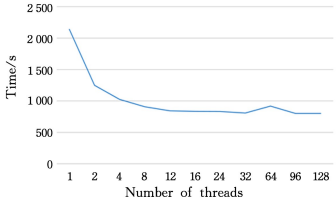
Fig. 6 CPU parallel architecture

由于 OpenMPI 在默认情况下为绑定 CPU，即每个进程固定在某个 CPU 核心上，使用多线程时，会使每个进程的多个子线程全部在同一个 CPU 核心上运行。实验发现，在网格分区数量和 MPI 进程数量不变且均为 n 时，开启 2 线程相比单线程可提速近 1 倍，但是开启 4，8，16 等线程数量时，由于默认开启了线程绑定，使实际运行时所耗时间与开启 2 线程

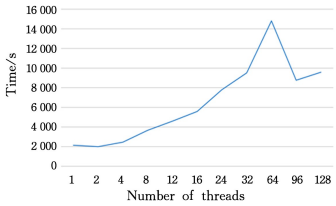
时几乎一样，而且随着线程数量的增多，每个子线程的 CPU 占用率越来越低，都在互相争抢 n 个 CPU 核心的使用，同时也只有 n 个 CPU 核心的占用率为 100%，其余核心几乎为 0。

因此考虑将 OpenMPI 解除进程与 CPU 间的绑定，使得新创建的子线程可以在其他物理核心上运行。实验发现，上述现象消失，可以达到预期结果。随着开启线程数量增多，运行速度提高，CPU 总占用率也提高。

以算例 1 为例，在分区为 2 块情况下，使用不同线程数量进行 MPI＋OpenMP 混合并行生成网格，实验结果如图 7 所示。其中标示的线程数是每个 MPI 进程内的线程数。



(a)系统实际时间消耗



(b)CPU 时间片消耗

图 7　MPI＋OpenMP 混合并行时间消耗

Fig. 7　MPI＋OpenMP hybrid parallel time consumption

由图 7 可以发现：

(1)保持分区数不变的情况下，随着线程数量增多，计算时间先快速减少，到 8 线程为止无法得到更快的速度，与 3.1 节中的结论一致。

(2)小于 64 线程时，CPU 时间片消耗会随着线程数量的增多而增加；到 64 线程后，线程数量到达 CPU 逻辑处理器数量上限，运行时，有部分线程处于 Sleeping 状态，因此时间片消耗不会提升，与 3.1 节中的结论一致。

4　数值实验

本节将进行 3 组对比实验，在 OpenFoam 中使用不同的求解器，分别在 snappyHexMesh 和 snappyOmp 工具生成的网格上进行数值实验模拟。

4.1　螺旋桨旋转模拟

以算例 1 为例进行数值实验模拟，基础笛卡尔网格设置为 24 * 40 * 24。流场及求解器部分参数设置如表 1 所列。

表 1　流场及求解器部分参数

Table 1　Some parameters of flow field and solver

网格数量	流体粘性系数	螺旋桨角速度	求解器	时间步长	库朗数
约 352 万	1×10^{-6}	158 rad/s	pimple-Foam	2×10^{-6}	0.5

计算至 0.1 s 时刻网格生成所耗时间，网格质量检查、求解器计算耗时如表 2 所列。

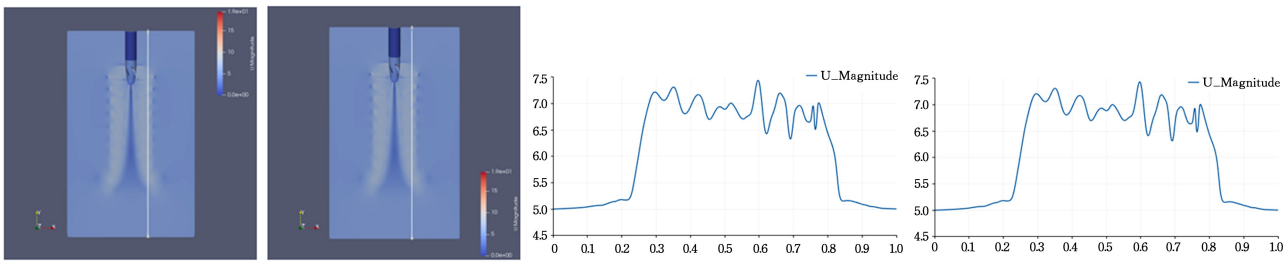
表 2　网格质量与计算时间对比

Table 2　Comparison of grid quality and runtime

比较	网格对齐耗时/s	高偏斜度面数量	求解器计算耗时/s
snappyHexMesh	1 143.35	4	47 753
snappyOmp	338.233 1	4	47 054

图 8 为 0.1 s 时刻两个实验的速度示意图。其中，在图 8(a)、图 8(b)中选取螺旋桨右侧 y 轴方向上的一条线，图 8(c)、图 8(d)为这条线上的速度分布。

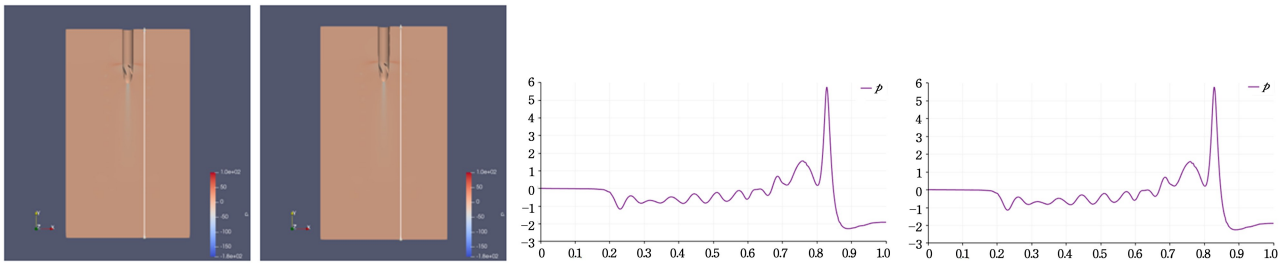
图 9 为 0.1 s 时刻两个实验的压力示意图。同样在图 9(a)、图 9(b)中选取螺旋桨右侧 y 轴方向上的一条线，图 9(c)、图 9(d)为这条线上的压力分布。



(a)snappyOmp 速度示意图　(b)snappyHexMesh 速度示意图　(c)图(a)线上的速度分布　(d)图(b)线上的速度分布

图 8　0.1 s 时刻速度分布比较

Fig. 8　Comparison of velocity distribution at 0.1 s



(a)snappyOmp 压力示意图　(b)snappyHexMesh 压力示意图　(c)图(a)线上的压力分布　(d)图(b)线上的压力分布

图 9　0.1 s 时刻压力分布比较

Fig. 9　Comparison of pressure distribution at 0.1 s

对生成的两种网格进行质量检查发现，在生成约 352 万个单元的网格时，仅有 4 个高偏斜度的面，说明使用新的并行方案所生成的网格与之前的网格相比，网格质量相差不大；分别对两种网格使用 pimpleFoam 求解器进行计算时，数值实验模拟结果和求解器耗费时间也相差不大。

4.2 摩托车迎风模拟

算例 2 OpenFoam 目录：tutorials/incompressible/adjointOptimisationFoam/sensitivityMaps/motorBike.

对算例 2 进行数值实验模拟，实验具体内容为静止不动的摩托车，模拟迎面来风时的流场。基础笛卡尔网格设置为 60 * 24 * 24，流场和求解器部分参数设置如表 3 所列。

表 3 流场及求解器部分参数

Table 3 Some parameters of flow field and solver

网格数量	流体粘性系数	流场速度	流场压力	求解器	时间步长
约 1313 万	1.5×10^{-5}	(20,0,0)	0	adjointOptimisationFoam	0.1

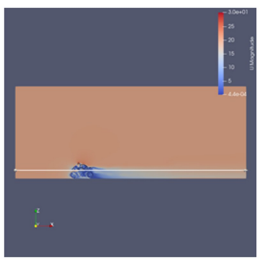
计算至收敛结束(收敛条件为压力速度等参数的计算残差小于 1×10^{-5})，网格生成所耗时间、网格质量检查、求解器计算耗时如表 4 所列。

表 4 网格质量与计算时间对比

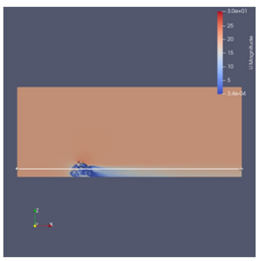
Table 4 Comparison of grid quality and runtime

比较	网格对齐耗时/s	高偏斜度面数量	求解器计算耗时/s
snappyHexMesh	9888.34	70	106457
snappyOmp	3152.54	46	99827

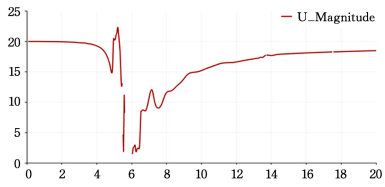
图 10 为计算收敛时流场的速度示意图。其中，在图 10(a)、图 10(b)中选取 x 轴方向上穿过摩托车的一条线，图 10(c)、图 10(d)为这条线上的速度分布。



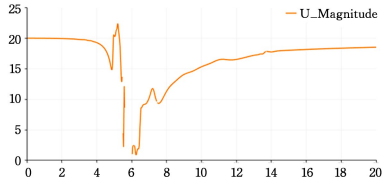
(a) snappyOmp 速度示意图



(b) snappyHexMesh 速度示意图



(c)图(a)线上的速度分布

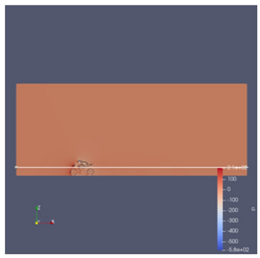


(d)图(b)线上的速度分布

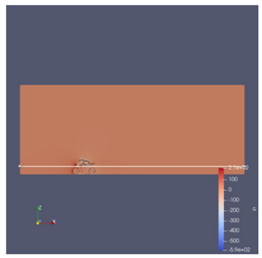
图 10 计算收敛时速度分布比较

Fig. 10 Comparison of velocity distribution in convergence

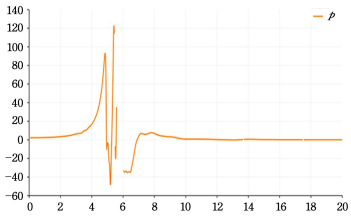
图 11 为计算收敛时流场的压力示意图。同样在图 11(a)、图 11(b)中选取 x 轴方向上穿过摩托车的一条线，图 11(c)、图 11(d)为这条线上的压力分布。



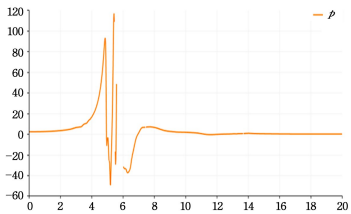
(a) snappyOmp 压力示意图



(b) snappyHexMesh 压力示意图



(c)图(a)线上的压力分布



(d)图(b)线上的压力分布

图 11 计算收敛时压力分布比较

Fig. 11 Comparison of pressure distribution in convergence

对两种网格进行质量检查，在生成约 1313 万个单元的网格时，分别仅有 70 和 46 个高偏斜度面，说明使用新的并行方法对网格质量的影响不大；分别在两种网格上使用 adjointOptimisationFoam 求解器求解计算时，数值实验模拟结果和求解器计算耗费时间也相差不大。

4.3 建筑群风场模拟

算例 3 OpenFoam 目录：tutorials/incompressible/simpleFoam/windAroundBuildings.

对算例 3 进行数值实验模拟，实验具体内容为城市建筑群，模拟城市中风的流场。流场和求解器部分参数设置如表 5 所列。

表 5 流场及求解器部分参数

Table 5 Some parameters of flow field and solver

网格数量	流体粘性系数	流场速度	流场压力	求解器	时间步长
约 327 万	1.5×10^{-5}	(10,0,0)	0	simpleFoam	0.5

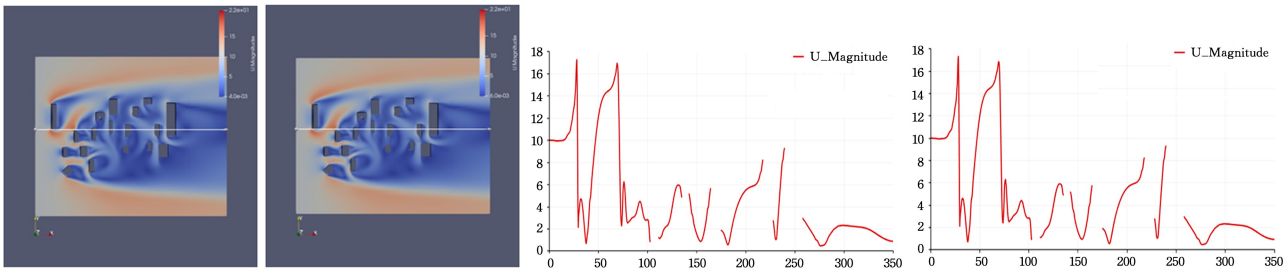
计算至 1000s 时刻网格生成所耗时间，网格质量检查、求解器计算耗时如表 6 所列。

表 6 网格质量与计算时间对比

Table 6 Comparison of grid quality and runtime

比较	网格对齐耗时/s	高偏斜度面数量	求解器计算耗时/s
snappyHexMesh	866.73	0	33750
snappyOmp	398.9818	0	33391

图 12 为 1 000 s 时刻两个实验的速度示意图。其中，在图 12(a)、图 12(b)中选取 x 轴方向上穿过建筑群的一条线，图 12(c)、图 12(d)为这条线上的速度分布。

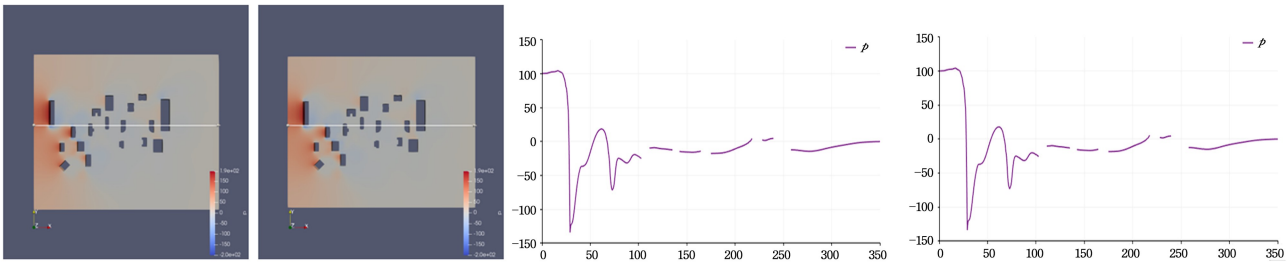


(a)snappyOmp 速度示意图　(b)snappyHexMesh 速度示意图　(c)图(a)线上的速度分布　(d)图(b)线上的速度分布

图 12　1 000 s 时刻速度分布比较

Fig. 12　Comparison of velocity distribution at 1 000 s

图 13 为 1 000 s 时刻两个实验的压力示意图，同样在图 13(a)、13(b)中选取 x 轴方向上穿过建筑群的一条线，图 13(c)、13(d)为这条线上的压力分布。



(a)snappyOmp 压力示意图　(b)snappyHexMesh 压力示意图　(c)图(a)线上的压力分布　(d)图(b)线上的压力分布

图 13　1 000 s 时刻压力分布比较

Fig. 13　Comparison of pressure distribution at 1 000 s

对两种网格进行质量检查，在生成约 327 万个单元的网格时，高偏斜度面的数量均为 0，说明网格质量差别不大；分别使用两种网格、采用 simpleFoam 求解器计算时，数值实验模拟结果和求解器计算耗费时间也相差不大。

结束语　本文首先在 OpenFoam 中实现了多面体网格的纯 OpenMP 并行生成和 OpenMP＋MPI 混合并行生成，然后对生成网格时的 OpenMP 并行效率和混合并行效率进行了理论分析与验证，最后使用不同的求解器在不同网格上进行数值模拟实验测试。主要结论如下：

(1)在 OpenMP 并行区域分配线程时，将并行区域内循环的各个部分直接指定给各个线程，而不是在计算过程的每一步循环时进行分配，可以有效减少并行时的时间消耗。

(2)使用纯 OpenMP 并行和混合并行方法生成网格，少于 8 线程时，随着线程数量的增多，所耗时间减少；多于 8 线程时，其无法随着线程数量的增多而获得更好的加速效果。

(3)线程数量少于计算机的逻辑处理器个数时，CPU 时间片消耗会随着线程数量的增多而增加；线程数量大于等于计算机逻辑处理器个数时，CPU 消耗不会继续增加。

(4)随着网格单元数量的增加，加速比不断提高，但加速比的提高速度会有所降低。

(5)经过 3 组求解器数值模拟实验证明，混合并行方法生成的网格可以用于实际使用，与原网格的差别不大。

在本文实验中总结发现，开启一定数量的线程可以达到并行区域内的速度上限，如本文所用计算机进行多面体网格生成，开启约 8 线程时，速度最快，再继续增加线程数量时，也无法进一步减少时间消耗。在未来的工作中，可以考虑如何对计算速度的上限进行处理优化，突破 8 线程的限制，以提高计算速度。

参 考 文 献

[1] LUO L，GAO Z X，JIANG C W. CFD Development and Application in Aviation[J]. Aeronautical Manufacturing Technology，2016，515(20)：77-81.

[2] WINTER M，HECKMEIER F M，BREITSAMTER C. CFD-based aeroelastic reduced-order modeling robust to structural parameter variations[J]. Aerospace Science and Technology，2017，67：13-30.

[3] JAMESON A，MARTINELLI L，VASSBERG J. Using computational fluid dynamics for aerodynamics--a critical assessment[C]// Proceedings of the Proceedings of ICAS. 2002.

[4] ALI Z，TYACKE J，WATSON R，et al. Efficient preprocessing of complex geometries for CFD simulations[J]. International Journal of Computational Fluid Dynamics，2019，33(3)：98-114.

[5] NGUYEN V T，VU D T，PARK W G，et al. Navier--Stokes solver for water entry bodies with moving Chimera grid method in 6DOF motions[J]. Computers & Fluids，2016，140：19-38.

[6] ALFONSI G. Reynolds-averaged Navier--Stokes equations for turbulence modeling[J]. Applied Mechanics Reviews，2009，62(4)：933-944.

[7] AHRENS J，GEVECI B，LAW C. Paraview：An end-user tool for large data visualization[J]. The Visualization Handbook，2005：717-31.

[8] WANG M. Design and Implementation of Parallel Graph-partitioning Methods for OpenFoam [D]. Changsha：National Uni-

versity of Defense Technology, 2012.
[9] JASAK H, JEMCOV A, TUKOVIC Z, et al. OpenFoam: A C++ library for complex physics simulations[C]// Proceedings of the International Workshop on Coupled Methods in Numerical Dynamics. 2007.
[10] FABRITIUS B O, TABOR G. Improving the quality of finite volume meshes through genetic optimisation[J]. Engineering with Computers, 2016, 32(3): 425-440.
[11] KORTELAINEN J. Meshing Tools for Open Source CFD: A Practical Point of View [J/OL]. VTT. https://cris.vtt.fi/en/publications/meshing-tools-for-open-source-cfd-a-practical-point-of-view.
[12] GISEN D. Generation of a 3D mesh using snappyHexMesh featuring anisotropic refinement and near-wall layers [C]// Proceedings of the ICHE 2014 Proceedings of the 11th International Conference on Hydroscience & Engineering. 2014.
[13] XU C F, CHE Y G, LI D L, et al. Research progresses of large-scale parallel computing for high-order CFD on the Tianhe supercomputer [J]. Computer Engineering & Science, 2020, 42(10): 1815-1826.
[14] SUN H Q. Redearch of Parallel Computing in Computational Fluid Dynamics [D]. Dalian: Dalian University of Technology, 2005.
[15] LÖHNER R, CAMBEROS J, MERRIAM M. Parallel unstructured grid generation[J]. Computer Methods in Applied Mechanics and Engineering, 1992, 95(3): 343-357.
[16] LIU S. Research on High Quality Unstructured Mesh Generation Algorithms with Multilevel Parallelism [D]. Changsha: National University of Defense Technology, 2019.
[17] QI L, XIAO S M, LIU Y C, et al. Research on Parallel Unstructured Mesh Generation Technology Based on GPU[J]. Machinery Design & Manufacture, 2013, 2: 184-186.
[18] XU Q, LIU T T, LENG J L, et al. Multilevel parallel tetrahedral mesh generation for complex geometric models[J]. Journal of National University of Defense Technology, 2021, 43(2): 33-39.
[19] LEE S B. Numerical discrepancy between serial and MPI parallel computations[J]. International Journal of Naval Architecture and Ocean Engineering, 2016, 8(5): 434-441.
[20] TOWARA M, SCHANEN M, NAUMANN U. MPI-parallel discrete adjoint OpenFoam[J]. Procedia Computer Science, 2015, 51: 19-28.
[21] CAI Y L, XIAO S M, QI L. Locking paralleled GPU - based method research for unstructured mesh generation[J]. Computer Engineering and Applications, 2013, 50(6): 56-60.
[22] WEI F, JIN L, LIU J, et al. GPU acceleration of a 2D compressible Euler solver on CUDA-based block-structured Cartesian meshes[J]. Journal of the Brazilian Society of Mechanical Sciences and Engineering, 2020, 42(5): 1-12.
[23] WANG N H, CHANG X H, ZHAO Z, et al. Implementation of hybrid MPI+OpenMP parallelization on unstructured CFD solver and its applications in massive unsteady simulations[J]. Acta Aeronautica et Astronautica Sinica, 2020, 41(10): 190-204.
[24] HU X, LU Z, ZHANG J, et al. A parallel algorithm for chimera grid with implicit hole cutting method[J]. The International Journal of High Performance Computing Applications, 2020, 34(2): 169-177.
[25] CHANG X H, MA R, WANG N H, et al. A parallel implicit hole-cutting method based on background mesh for unstructured Chimera grid[J]. Computers & Fluids, 2020, 198: 104403.
[27] OpenFOAM User Guide [M/OL]. https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.4-mesh-generation-with-the-snappyhexmesh-utility, 2021.
[28] JARPNER C. Projection of a mesh on a. stl surface[J/OL]. Chalmers University of Technology. http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2011/ChristofferJarpner/ReportInOpenFOAMbyChristofferJarpner.pdf.

LIU Jiang, born in 1979, Ph.D, associate professor, is a member of China Computer Federation. His main research interests include computability theory, formal methods and computer algorithms.

LIU Wen-bo, born in 1996, postgraduate. His main research interests include computational fluid dynamics and high-performance computing.

(责任编辑:李亚辉)