计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
[doi: 10.15888/j.cnki.csa.009817] [CSTR: 32024.14.csa.009817]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Kokkos 框架的 CFD 求解器性能可移植性分析①

王　辰, 陈　龙

(南京航空航天大学 航空学院, 南京 210016)
通信作者: 陈　龙, E-mail: lchen@nuaa.edu.cn

摘　要: 为提高计算流体力学的求解速度, 一般采用并行执行的方法, 然而由于计算硬件架构和编程语言的多样性, 对程序的可移植性带来了挑战. 为此本文使用 Kokkos 框架实现了计算流体力学求解器的异构并行计算, 并且使用规约法、原子操作和染色法来处理并行计算过程中的数据冲突问题, 提出了基于此框架下的异构并行计算的数据冲突的算法实现方案. 针对图形处理器的架构特点, 分析了不同硬件上单精度和双精度计算的加速比, 得出了不同计算硬件上的最优并行策略. 研究证明使用原子操作单精度计算对于使用图形处理器加速流体力学计算而言, 能够极大地提升求解效率.

关键词: 异构计算; 计算流体力学; Kokkos; 非结构网格; 有限体积法

引用格式: 王辰,陈龙.基于 Kokkos 框架的 CFD 求解器性能可移植性分析.计算机系统应用. http://www.c-s-a.org.cn/1003-3254/9817.html

Performance Portability Analysis of CFD Solver Based on Kokkos

WANG Chen, CHEN Long

(College of Aerospace Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract: To accelerate the solution of computational fluid dynamics (CFD), parallel execution is commonly used. However, the diversity of computing hardware architectures and programming languages poses challenges to program portability. In this study, the Kokkos framework is used to implement heterogeneous parallel CFD computing. Moreover, the reduction method, atomic operations, and the coloring approach are employed to address data conflicts in the process of parallel computing. A specific algorithmic solution for data conflict in heterogeneous parallel computing under this framework is proposed. Given the architectural characteristics of the graphics processing unit (GPU), the speedup ratios of single-precision and double-precision calculations on different hardware are analyzed, and optimal parallel strategies on different computing hardware are obtained. The study demonstrates that using atomic operations for single-precision computations on GPUs significantly enhances CFD solving efficiency.

Key words: heterogeneous computing; computational fluid dynamics (CFD); Kokkos; unstructured grid; finite volume method

计算流体力学 (computational fluid dynamics, CFD) 是一门广泛应用于航空航天、海洋环境等领域的学科[1]. 非结构网格有限体积法是 CFD 中广泛使用的数值离散方法[2], 因其能够简易地生成不同形状的网格, 适用于复杂几何体的流体流动分析. 然而, 为了提高计算精度, 往往需要对网格进行加密, 这导致了计算量的显著增加. 图形处理器 (graphics processing unit, GPU) 的快速发展为解决这一类数据密集型计算问题

① 基金项目: 江苏省高校优势学科建设工程; 国家自然科学基金委员会联合基金 (U20A2070)
收稿时间: 2024-10-07; 修改时间: 2024-10-21; 采用时间: 2024-11-12; csa 在线出版时间: 2025-02-28

提供了新的解决方案. 与传统中央处理器 (central processing unit, CPU) 比, GPU 拥有更多的轻量级计算核心和更高的带宽, 使其在大规模数据并行计算中具有明显优势[3].

由于 Dennard 缩放定律的终结[4], 晶体管尺寸的缩小以及处理器频率的增长已遇到瓶颈, 传统处理器架构无法再实现线性的性能提升. 在这种背景下, 使用 GPU 等作为加速器或协处理器的异构计算系统成为提高计算能力的最佳解决方案. 根据 2024 年 6 月发布的超级计算机榜单, 目前前十名的超级计算机中有 90% 使用了协处理器或加速器[5]. 这一趋势突显了异构计算在高性能计算中的重要地位. 然而, 不同的 GPU 厂商如 Nvidia 和 AMD, 其硬件架构技术路线不尽相同, 导致了开发中出现了多种编程范式, 从而产生了重复劳动[6]. 例如由 Nvidia 开发的 CUDA, 是目前针对 Nvidia GPU 性能优化最好的编程模型, 但其跨平台性差, 无法兼容其他硬件架构.

主流异构性能可移植方案有 OpenMP4.0、OpenACC 和 OpenCL 等. 前两者语法类似, 同样采用#pragma 标记并行代码块. 这样在改写已有代码时能够极大地提供便利, 但是在内存管理和并行粒度控制上就显得不够灵活. OpenCL 具有极强的跨平台性, 且能够在不同平台上进行细粒度的并行优化, 然而其编程接口复杂, 开发者需要手动管理设备、内存、线程等, 导致开发时间和调试成本较高.

为解决这一问题, 美国能源部组织了 E 级计算计划, 旨在通过性能可移植性来简化开发过程[7−9], 从而诞生了 Kokkos 和 RAJA 等项目[10,11]. Kokkos 和 RAJA 均采用了模板元编程技术, 通过抽象模板将并行计算的具体细节封装在库内, 开发者只需遵循抽象规范即可进行编程, 极大地简化了开发流程. 同时, 这两者都具有与 OpenCL 相当的跨平台特性, 支持 CPU、GPU (CUDA、HIP)、FPGA. 针对 CFD 代码在这两个框架下的性能测试, Law 等 [12]对二维非结构网格可压缩 CFD 程序 Bookleaf 进行了测试, 结果发现 Kokkos 在 Tesla P100 GPU 上比 CUDA 快 0.5%–1.0%, 而 RAJA 则比 CUDA 慢约 2%. 此外, Kirk 等[13]使用结构网格热传导的隐式求解器 Tealeaf, 在 P100 GPU 上的测试显示, 手工编码的 CUDA 表现最佳, Kokkos 和 RAJA 则落后 7%–15%. Martineau 等[14]使用 Tealeaf 对 Kokkos 和 RAJA 进行测试，发现 Kokkos 在 Nvidia GPU 中的性能优于 RAJA. 通过 Pennycook 等[15]提出的性能可移植性评估矩阵, Kokkos 和 RAJA 都能取得良好的分数. 对矩阵点积函数测试结果表明, Kokkos 综合性能略强于 RAJA[16].

针对非结构网格几何信息的储存特点, 使用有限体积法求解常利用按面循环的思路, 此时对网格并行遍历时就会存在数据竞争的问题. 高效的并行策略成为解决问题的关键. 而目前对于并行策略的性能可移植性的评估研究较少. 为兼顾跨平台性与异构计算效率, 本文使用 Kokkos 异构并行框架开发了一套非结构网格有限体积法的 CFD 求解器, 实现了 3 种并行策略, 及在此框架下并行策略的详细实现方案, 并在 Nvidia 和 AMD 的主流 GPU 上探索了其性能可移植性的潜力, 分析了不同策略在不同硬件平台上的运行效率.

1 数值计算方法

采用忽略源项的 Navier-stokes 方程, 其形式如下:

$$\frac{\partial}{\partial t}\int_{\Omega} W\mathrm{d}\Omega + \oint_{\partial\Omega}(F-G)\mathrm{d}S = 0 \tag{1}$$

其中, W, F, G 分别为守恒性变量、对流通量和粘性通量. 为了达到良好的激波分辨率同时保证方程的守恒性, 对流通量采用基于通量差分分裂 (flux difference splitting, FDS) 格式的 Roe 近似黎曼求解方法[17].

本求解器采用格心格式储存有限体积法离散控制体的流场变量, 形式如下:

$$\Omega_I\frac{\mathrm{d}W_I}{\mathrm{d}t} = -R_I = -\sum_{m=1}^{N_F}(F-G)_m\Delta S_m \tag{2}$$

网格面上通量采用分段线性重构 (piecewise linear reconstruction) 达到二阶精度[18], 如图 1 所示.

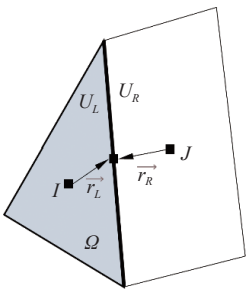


图 1 基于格心格式的线性重构

使用收敛性良好的 Venkatakrishnan 限制器[19,20]以防止大梯度区域求解时产生震荡. 湍流模型采用航空

领域应用广泛的 Spalart-Allmaras 一方程模型[21]. 时间推进格式采用四步 Runge-Kutta 显式格式.

2 并行方法

2.1 Kokkos 异构模型

基于 C++为底层编写的 Kokkos, 为数据存储方式和程序执行空间提供了抽象, 以满足性能可移植的要求. Kokkos 作为中间层, 连接了软件层和硬件层, 使代码编写时无需关心硬件实现并行计算的细节, 只需专注于任务的并行逻辑. 图 2 概述了 Kokkos 运行体系. 在计算硬件解决方案的数量及程序复杂度不断增加的趋势下, 大大减少了软件开发周期, 提高了软件通用性.

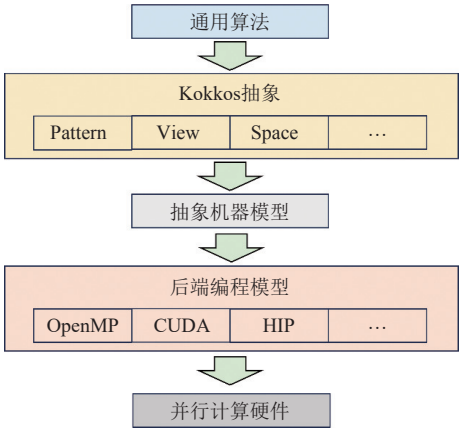


图 2 Kokkos 体系概览

Kokkos 编程模型的特点体现在 6 个核心抽象, 即将数据结构抽象为内存空间、内存布局和内存特性, 以及将并行执行抽象为执行空间、执行模式和执行策略. 其中, View 容器允许通过模板参数来指定数据的存储空间, 并根据硬件的索引特点选择适当的布局方式, 同时支持设置数据的访问属性, 例如原子访问 (Atomic)、随机访问 (RandomAccess) 等. 在并行执行方面, Kokkos 提供了丰富的模式 (Pattern), 包括 parallel_for、parallel_scan 和 parallel_reduce 等, 并允许通过指定执行空间来确定并行硬件. 执行策略则更加灵活, 涵盖了范围循环、线程团队以及调度循环等方式, 从而能够适应不同的逻辑算法需求. Kokkos 通过精心设计的接口, 将内存对齐、索引映射、异构设备与主机内存之间的通信与同步等复杂细节进行了封装. 用户只需通过统一的接口, 根据需求选择硬件设备, 而无需关心底层的执行细节, 通过适当调整模板参数, 便可简洁、高效地实现硬件跨平台并行计算.

2.2 面循环并行策略

基于非结构网格的有限体积法在求解流场变量时, 一般可通过对网格面循环, 将面上变量经由法向矢量二阶插值获得控制体的物理量, 包括通量、梯度、限制器等. 以下算法皆以无粘通量求解为例, 面通量结果在相邻单元中累计, 在右侧形成残差项. 该串行算法表示为算法 1.

算法 1. 无粘通量的串行算法

输入: 网格面左右单元索引 *nl*, *nr*, 流场变量 *cellVals*, 法向矢量 *fn* 和无粘通量计算函数 *inviscid_eval*.
输出: 右手残差项 *rhs*.

1. **for** $i = 0$ **to** $n_{\text{faces}} - 1$ **do**
2. $l = nl[i]$
3. $r = nr[i]$
4. $fluxes = inviscid\_eval(cellVals[l], cellVals[r], fn[i])$
5. **for** $ic = 0$ **to** 4 **do**
6. $rhs[l][ic] \mathrel{+}= fluxes[ic]$
7. $rhs[r][ic] \mathrel{-}= fluxes[ic]$
8. **end for**
9. **end for**

而在无粘通量过程的欧拉方程并行解过程中, 不同的面可能访问同一单元, 允许多个线程同时写入相同的内存地址, 从而导致数据竞争并导致计算错误. 为此, 且考虑性能可移植的目的, 本文采用了 3 种适用性良好且不依赖硬件架构的并行策略: 规约法、原子操作和染色法.

2.2.1 规约法

利用规约法求解通量时, 不直接将面通量累计至控制体, 而是通过在每个控制体上创建一个维数与该控制体网格面数相同的数组储存面通量, 并在面循环结束后, 启用新循环将面通量规约到控制体上. 其 Kokkos 实现方式如算法 2 所示. 这一方法优势在于无需对网格进行额外处理, 也无需进行加锁操作, 即可避免数据竞争. 由于对 *fluxes* 数组为赋值操作, 无需在每次 Runge-Kutta 迭代前进行初始化, 可以节省体循环的时间. 缺点在于需要占用额外存储空间, 且需要进行一次面循环和一次体循环才能计算出右端残差项.

算法 2. 规约法的 Kokkos 并行算法代码片段

```
1. using namespace Kokkos;
2. typedef View<double **[5] > flux_type;
3. typedef View<double *[5] > rhs_type;
4. flux_type fluxes("flux", ncells, nfpc);
5. rhs_type rhs("residual", ncells);
```

```
6. parallel_for(nfaces, KOKKOS_LAMBDA(int i) {
7.    int l = nl[i];
8.    int r = nr[i];
9.    auto flux = inviscid_eval(cellVals[l], cellVals[r], fn);
10.   for (int ic = 0; ic < 5; ic++) {
11.      fluxes(l,f_idl[i], ic) = flux[ic];
12.      fluxes(r,f_idr[i], ic) = –flux[ic];
13.   }
13. });
14. parallel_for(ncells, KOKKOS_LAMBDA(int i) {
15.   for (int iface = 0; iface < nfpc; iface++) {
16.      for (int ic = 0; ic < 5; ic++) {
17.         rhs(i, ic) += fluxes(i, iface, ic);
18.      }
19.   }
20. });
```

2.2.2 原子操作

在 Kokkos 中可以直接使用 Kokkos::atomic_[*op*] 语句, 对数据进行原子操作. 根据目标硬件和操作数大小, Kokkos 能够自动在直接原子指令、CAS 循环和分片锁表之间进行选择[22]. 串行代码仅需修改写入操作, 即可避免数据写入的竞争问题, 如算法 3 所示. 然而, 当多个线程同时访问同一单元时, 会导致计算效率随着竞争的增加而降低.

算法 3. 基于 Kokkos 原子操作的并行算法代码片段

```
1. using namespace Kokkos;
2. typedef View <double *[6]> rhs_type;
3. rhs_type rhs("residual", ncells);
4. parallel_for(nfaces, KOKKOS_LAMBDA(int i) {
5.    int l = nl[i];
6.    int r = nr[i];
7.    auto flux = inviscid_eval(cellVals[l], cellVals[r], fn);
8.    for (int ic = 0; ic < 5; ic++) {
9.       atomic_add(&rhs(l, ic), flux[ic]);
10.      atomic_add(&rhs(l, ic), –flux[ic]);
11.   }
12. });
```

2.2.3 染色法

利用染色法为网格面分配不同的颜色, 以防止相同颜色网格面通量向左右相邻单元写入时产生的数据冲突, 使其在多个硬件平台[23]上具有高度的灵活性. 这种方法需要对网格面进行预处理分组, 其中同一组中的所有网格面都有不同的左右单元, 以消除组内并行计算过程中的数据竞争. 理论上, 在无竞争的情况下不会引入额外开销, 所以在同一颜色小组之内, 其计算效率大于原子操作. 然而, 组间的串行计算可能会降低整体的计算效率. 测试表明, 扩大每组面数的规模能够显著提高计算效率.

此外, 由 RCM 排序[24]产生的紧凑的非结构网格数据结构降低了网格面分组过程中的时间复杂度. 图 3 展示了 RCM 排序预处理过后的网格面染色分组. 图 4 展示了 RCM 排序的效果. 算法 4 展示了染色法分组后的面循环方法.

算法 4. 染色法的 Kokkos 并行算法代码片段

```
1. using namespace Kokkos;
2. typedef View<int *> vec_type;
3. typedef View<double *[5]> rhs_type;
4. rhs_type rhs("residual", ncells);
5. vec_type seq("arr", nfaces);
6. vec_type offset("offset", nseqs);
7. for (int i = 0; i < nseqs; i++) {
8.    int b = offset(i);
9.    int e = offset(i+1);
10.   parallel_for(RangePolicy<>(b, e), KOKK_LAMBDA(int j) {
11.      int l = nl[seq(j)];
12.      int r = nr[seq(j)];
13.      auto flux = inviscid_eval(cellVals[l], cellVals[r], fn);
14.      for (int ic = 0; ic < 5; ic++) {
15.         rhs(l, ic) += flux[ic];
16.         rhs(r, ic) –= flux[ic];
17.      }
18.   });
19. }
```

3 实验分析

3.1 算例验证

为验证求解器计算正确性, 采用 ONERA M6 机翼三维非结构网格算例. 网格由全四面体组成, 其中节点数 3.74×10^5, 网格面数 4.23×10^6, 网格单元数 2.09×10^6. 初始流场设置为来流马赫数 0.839 5, 迎角 3.06°, 无侧滑角, 雷诺数 11.72×10^6. 图 5 展示了表面网格及声速无量纲化的压力系数分布云图. 翼面呈现出两道激波相互干扰而形成的 λ 型激波. 图 6 给出了机翼展向 5 个截面处使用单精度 (FP32) 和双精度 (FP64) 计算的压力系数分布与 Schmitt[25]实验数据对比.

3.2 跨平台性能分析

本节采用第 3.1 节中的 ONERA M6 机翼三维非结构网格算例对求解器的跨平台性能展开测试. 计算硬件分别为 Intel I7 14700K、Intel Xeon E5-2670V3、Nvidia Tesla K40c、Nvidia GeForce RTX 4090、Nvidia Tesla P100、Nvidia Tesla A100 以及 AMD Radeon RX7900 XTX. 其中 Intel I7 14700K 和 Intel Xeon E5-

2670V3 测试串行计算性能以及基于 OpenMP 的并行性能. 在使用 OpenMP 时设置环境变量 OMP_PLACES=threads 和 OMP_PROC_BIND=spread. 前者将线程绑定到单独的硬件线程上, 能够提高单核多线程硬件的并行度; 后者将线程平均分布在各个核心上, 提高缓存和内存带宽的利用率. 图 7 给出了 3 种并行策略在不同硬件平台上使用 Kokkos 并行框架编译执行, 并且分别以单、双精度对 Runge-Kutta 四步格式迭代 100 步的总执行时间. 表 1 是不同 GPU 的加速比, 对照基准为 Intel Xeon E5-2670V3 的串行执行时间.

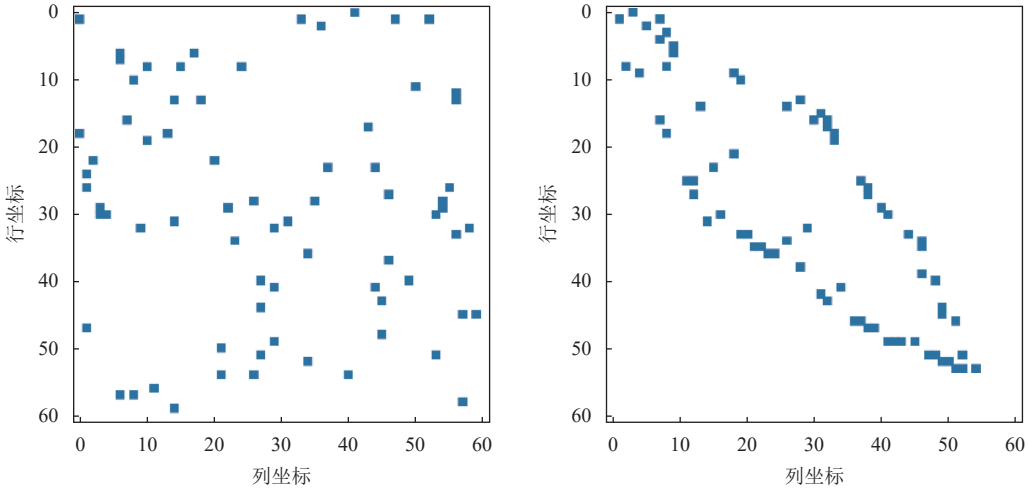


图 3 RCM 排序效果

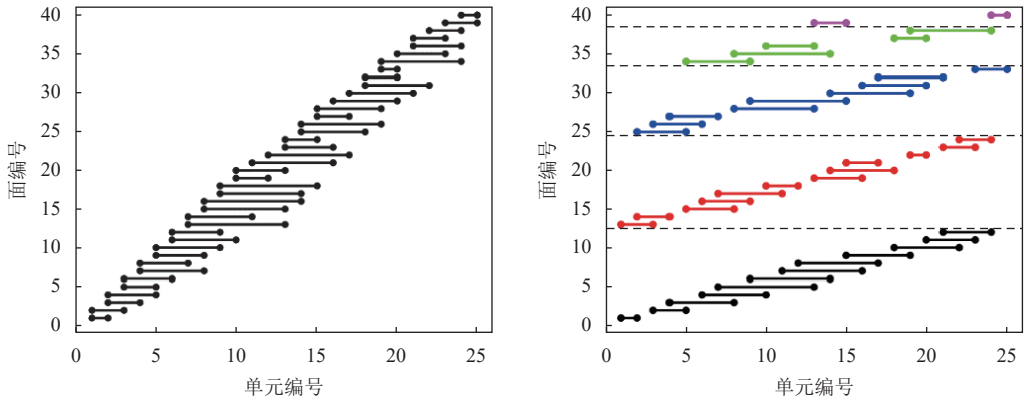


图 4 RCM 预处理网格的染色法分组

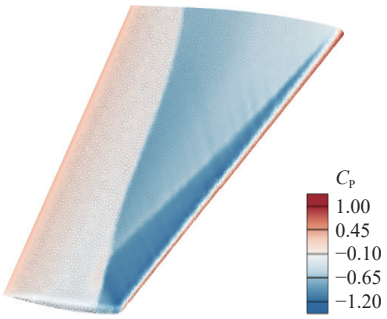
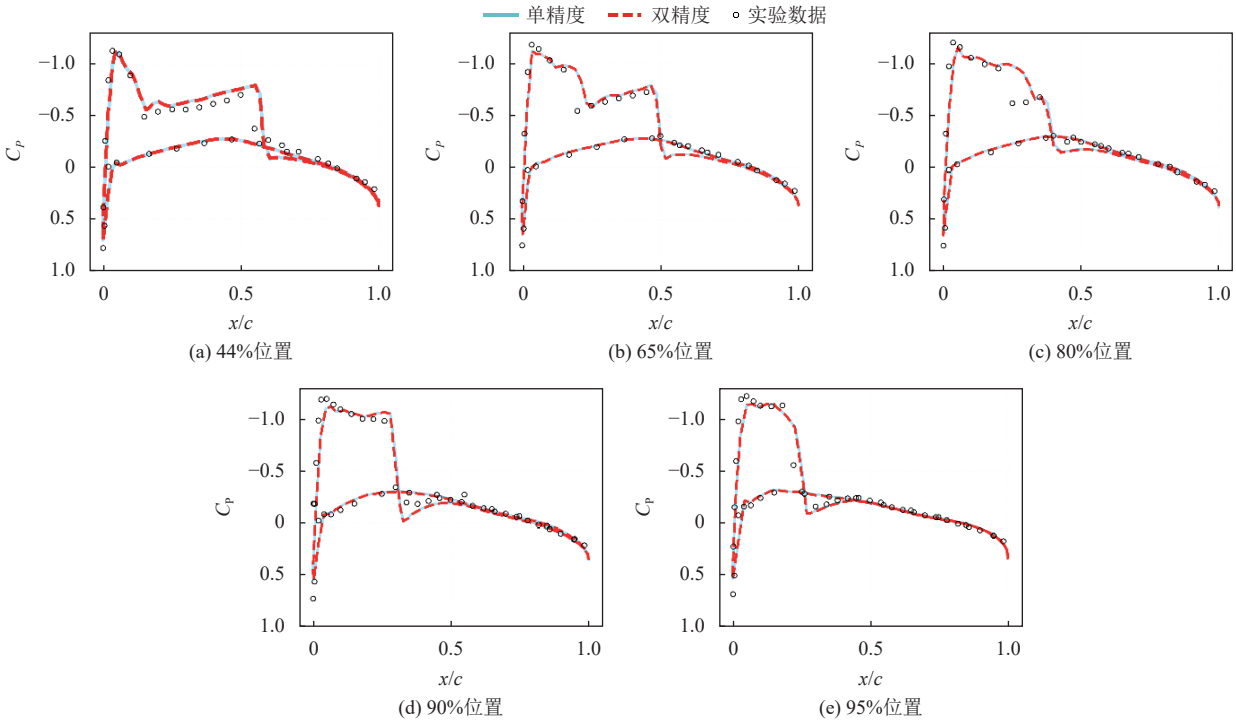


图 5 ONERA M6 机翼表面网格及压力系数分布

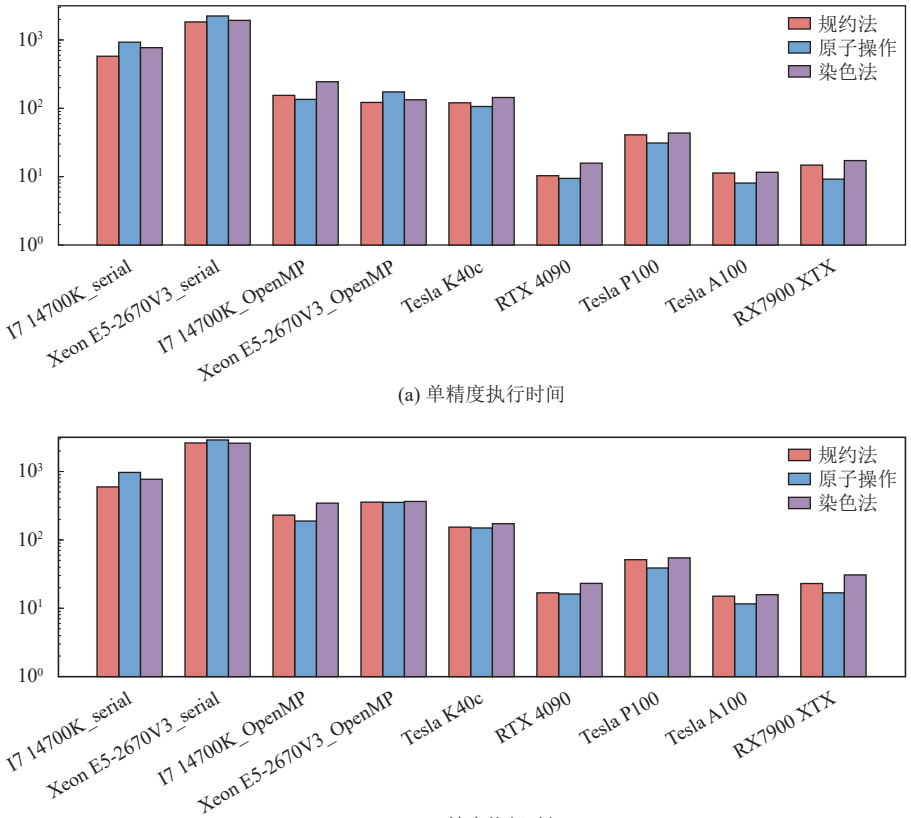
本研究对比了多种硬件平台执行单精度和双精度计算时的性能, 涵盖了 3 种不同的并行化策略, 包括规约法、原子操作和染色法. 从图 5 可以明显看出相较于 CPU 并行执行求解器时, GPU 展现出了显著的性能优势. 由于 Intel I7 14700K 的最高主频为 5.6 GHz, 而 Intel Xeon E5-2670V3 的最高主频仅为 3.1 GHz, 前者展现出更强大的串行性能. 表 1 显示了 Xeon E5-2670V3 在不同并行策略下, 加速比均高于 I7 14700K. 这归因于其具有 4 条内存通道, 高于 I7 14700K 的 2 条内存通道. 然而因为更新的架构设计、更小的制程工艺以及更大的带宽, I7 14700K 的并行执行时间依然小于 Xeon E5-2670V3.



(a) 44%位置 (b) 65%位置 (c) 80%位置

(d) 90%位置 (e) 95%位置

图 6 机翼展向压力系数分布



(a) 单精度执行时间

(b) 双精度执行时间

图 7 求解器在各计算硬件上的执行时间

在求解器执行时, 引入单精度浮点数计算来取代双精度浮点数可以获得更好的性能. 这种改进与现代计算硬件的能力有关, 能够显著缩短计算时间和降低计算能耗. 表 2 总结了测试所用 GPU 的计算性能, 即带宽以及理论峰值的每秒 10 亿浮点数运算次数 (tera floating-point operations per second, TFLOPS). 查表可知, 在早期的 Nvidia GPU 的 Kepler 架构中, Tesla K40c 的单双精度计算能力比值为 3, 而在后期推出的基于 Pascal 架构的 Tesla P100 和基于 Ampere 架构的 Tesla A100 这一比值均为 2. 以上为专业计算 GPU 的运算性能, 在消费级游戏 GPU 中, 基于 Nvidia 最新 Ada Lovelace 架构的 RTX 4090 和基于 AMD 最新 RDNA 3 架构的 RX 7900 XTX 这一比值分别为 64 和 32. 由于单精度浮点数的占用的内存空间仅为双精度的一半, 于是在相同的带宽下能够处理更多的数据量. 因此当对计算精度要求不高时, 使用单精度计算方法是极具时间与成本性价比的.

表 3 展示了以 Xeon E5-2670V3 串行时间为基准, 在不同 GPU 平台并行计算的加速比. 可以看出加速比与 GPU 的 TFLOPS 和带宽呈正相关. 同为专业计算 GPU 的 Tesla K40c、Tesla P100 和 Tesla A100 展现出的单双精度加速比比较接近, 这是由于 Xeon E5-2670V3 的单双精度性能是不同的. 通过观察图 5 的执行时间, 单精度执行速度明显快于双精度执行时间. RTX 4090 和 RX 7900 XTX 凭借更强大的单精度浮点数运算能力, 在单双精度执行总时间之差和单双精度加速比之差都展现出单精度计算的优势. Tesla A100 与两块消费级游戏 GPU 由于在带宽上有着 2 倍的优势, 这使得其无论在单精度还是在双精度计算都获得了最大的加速比.

表 1 基于 OpenMP 的 CPU 并行计算加速比

并行策略	I7 14700K		Xeon E5-2670V3	
	FP32	FP64	FP32	FP64
规约法	3.84	2.58	14.90	7.35
原子操作	5.37	5.09	12.88	8.23
染色法	3.15	3.28	14.62	7.12

表 2 GPU 理论峰值性能

型号	带宽 (GB/s)	TFLOPS		单双精度比
		FP32	FP64	
Tesla K40c	288	4.29	1.43	3
Tesla P100	732.2	9.526	4.763	2
Tesla A100	1940	19.49	9.746	2
RTX 4090	1010	82.58	1.29	64
RX 7900 XTX	960	61.39	1.918	32

表 3 GPU 并行计算加速比

并行策略	Tesla K40c		Tesla P100		Tesla A100		RTX 4090		RX 7900 XTX	
	FP32	FP64	FP32	FP64	FP32	FP64	FP32	FP64	FP32	FP64
规约法	15.17	17.38	45.14	51.91	162.84	177.67	178.84	158.38	125.67	117.11
原子操作	21.34	19.81	71.81	76.01	279.24	257.46	239.36	184.57	244.15	175.16
染色法	13.51	15.06	45.12	48.38	170.99	168.65	124.49	114.23	113.39	86.49

通过比较表 2 与表 3 可以发现, 在 GPU 中虽然单精度计算性能与双精度相比有着成倍地增加, 但是实际求解器运算中却无法达到相应的加速提升. 这是由于 CFD 计算往往是内存带宽受限型的, 性能瓶颈往往来自内存的读写速度, 尽管单精度浮点数操作消耗较小内存, 但总体运行速度仍然受内存带宽的限制. 同时, GPU 线程块之间的通信同步也会降低整体的运算速度, 使得求解器无法达到预计的计算性能.

通过比较图 5 中 3 种不同并行策略执行时间, 很容易发现在 CPU 和 GPU 上执行最快的策略并不相同. 在不同 GPU 平台上, 原子操作都是效率最高的并行策略, 染色法略次于规约法. 这是由于染色法中组间串行对 GPU 计算性能产生影响, 而规约法则是因为数据膨胀占用了更大的带宽导致性能下降. CPU 平台上, 大多数情况下规约法展现出略优或是相似于染色法的性能, 而原子操作往往效率最低. 唯一例外的是在 I7 14700K 上使用 OpenMP 并行执行时, 原子操作效率最高, 染色法次于规约法, 这一趋势与 GPU 相同.

4 结论与展望

使用 Kokkos 并行框架开发了一个基于有限体积法的 CFD 非结构网格异构跨平台求解器, 提出并详细论述了 3 种解决数据冲突的并行策略实现方案, 并且支持单精度与双精度求解. 测试了在不同 CPU 和 GPU 平台上不同并行策略、不同浮点数精度的性能差异, 对于使用 GPU 作为加速计算硬件时使用原子操作是最优方法, 在 Tesla A100 上可展现出 279.46 倍单精度加速比和 257.46 倍双精度加速比. 同时发现, 除浮点数运算性能外, 内存带宽瓶颈和数据同步也是制约 CFD 求解器计算速度的重要因素. 由于 Kokkos 框架强大的

异构跨平台特性, 未来在其基础上开发基于高精度格式的多卡并行 CFD 求解器是一个极具工程研究意义的研究方向.

参考文献

1 张曦, 孙旭, 郭晓虎, 等. 面向 GPU 的非结构网格有限体积计算流体力学的图染色方法优化. 国防科技大学学报, 2022, 44(5): 24–34. [doi: 10.11887/j.cn.202205003]

2 张帆. 非结构网格有限体积法的空间离散算法研究 [博士学位论文]. 大连: 大连理工大学, 2017.

3 陈龙, 徐添豪, 田书玲. 基于非结构网格隐式算法的 GPU 加速研究. 计算机系统应用, 2018, 27(5): 238–243. [doi: 10.15888/j.cnki.csa.006371]

4 Haensch W. Scaling is over—What now? Proceedings of the 75th Annual Device Research Conference (DRC). South Bend: IEEE, 2017. 1–2. [doi: 10.1109/DRC40984.2017]

5 TOP500. 超级计算机榜单. https://www.top500.org/lists/top500/2024/06/. [2024-10-01].

6 郑亮, 黎坤运, 周兴彬, 等. 基于 Kokkos 模板元编程的性能可移植求解器开发. 数据与计算发展前沿, 2024, 6(1): 12–20. [doi: 10.11871/jfdc.issn.2096-742X.2024.01.002]

7 Messina P. The exascale computing project. Computing in Science & Engineering, 2017, 19(3): 63–67. [doi: 10.1109/MCSE.2017.57]

8 Dubey A, McInnes LC, Thakur R, *et al*. Performance portability in the exascale computing project: Exploration through a panel series. Computing in Science & Engineering, 2021, 23(5): 46–54. [doi: 10.1109/MCSE.2021.3098231]

9 Deakin T, McIntosh-Smith S, Price J, *et al*. Performance portability across diverse computer architectures. Proceedings of the 2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC. Denver: IEEE, 2019. 1–13. [doi: 10.1109/P3HPC49587.2019.00006]

10 Edwards HC, Trott CR. Kokkos: Enabling performance portability across manycore architectures. Proceedings of the 2013 Extreme Scaling Workshop. Boulder: IEEE, 2013. 18–24. [doi: 10.1109/XSW.2013.7]

11 Beckingsale DA, Burmark J, Hornung R, *et al*. RAJA: Portable performance for large-scale scientific applications. Proceedings of the 2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC. Denver: IEEE, 2019. 71–81. [doi: 10.1109/P3HPC49587.2019.00012]

12 Law TR, Kevis R, Powell S, *et al*. Performance portability of an unstructured hydrodynamics mini-application. Proceedings of the 2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC. Dallas: 2018. 0–12.

13 Kirk RO, Mudalige GR, Reguly IZ, *et al*. Achieving performance portability for a heat conduction solver mini-application on modern multi-core systems. Proceedings of the 2017 IEEE International Conference on Cluster Computing. Honolulu: IEEE, 2017. 834–841. [doi: 10.1109/CLUSTER.2017.122]

14 Martineau M, McIntosh-Smith S, Gaudin W. Assessing the performance portability of modern parallel programming models using TeaLeaf. Concurrency and Computation: Practice and Experience, 2017, 29(15): e4117. [doi: 10.1002/cpe.4117]

15 Pennycook SJ, Sewall JD, Lee VW. Implications of a metric for performance portability. Future Generation Computer Systems, 2019, 92: 947–958. [doi: 10.1016/j.future.2017.08.007]

16 Davis JH, Sivaraman P, Minn I, *et al*. Evaluating performance portability of GPU programming models. Comput. Sci. Eng, 2018, 17: 247–262

17 Roe PL. Approximate Riemann solvers, parameter vectors, and difference schemes. Journal of Computational Physics, 1981, 43(2): 357–372. [doi: 10.1016/0021-9991(81)90128-5]

18 Barth T, Jespersen D. The design and application of upwind schemes on unstructured meshes. Proceedings of the 27th Aerospace Sciences Meeting. Reno: AIAA, 1989. 366. [doi: 10.2514/6.1989-366]

19 Venkatakrishnan V. On the accuracy of limiters and convergence to steady state solutions. Proceedings of the 31st Aerospace Sciences Meeting. Reno: AIAA, 1993. 880. [doi: 10.2514/6.1993-880]

20 Venkatakrishnan V. Convergence to steady state solutions of the Euler equations on unstructured grids with limiters. Journal of Computational Physics, 1995, 118(1): 120–130. [doi: 10.1006/jcph.1995.1084]

21 Spalart P, Allmaras S. A one-equation turbulence model for aerodynamic flows. Proceedings of the 30th Aerospace Sciences Meeting and Exhibit. Reno: AIAA, 1992. 439. [doi: 10.2514/6.1992-439]

22 Trott CR, Lebrun-Grandié D, Arndt D, *et al*. Kokkos 3: Programming model extensions for the exascale era. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(4): 805–817. [doi: 10.1109/TPDS.2021.3097283]

23 Balogh GD, Reguly IZ, Mudalige GR. Comparison of parallelisation approaches, languages, and compilers for unstructured mesh algorithms on GPUs. Proceedings of the 8th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems. Denver: Springer, 2017. 22–43. [doi: 10.1007/978-3-319-72971-8_2]

24 Cuthill E, McKee J. Reducing the bandwidth of sparse symmetric matrices. Proceedings of the 24th National Conference. New York: ACM, 1969. 157–172. [doi: 10.1145/800195.805928]

25 Schmitt V, Charpin F. Pressure distributions on the ONERA-M6-wing at transonic Mach numbers. Experimental Data Base for Computer Program Assessment. Report of the Fluid Dynamics Panel Working Group 04. AGARD AR-138, 1979.

(校对责编: 王欣欣)