DOI:10.11918/202208043

RKDG 有限元 GPU 算法及其重排加速技术

高缓钦[1,2],陈红全[1,2],张加乐[1,2],贾雪松[1,2]

(1.南京航空航天大学 航空学院,南京 210016;
2.非定常空气动力学与流动控制工信部重点实验室(南京航空航天大学),南京 210016)

摘 要:为提升并行化求解 Navier Stokes 方程的效率,构建了高阶有限元单元及单元边界映射线程结构和对应的各类 GPU 核函数,成功地把 RKDG 方法移植到 GPU 架构,发展出 RKDG 有限元 GPU 并行算法。算法数据访存能兼容 GPU 快慢不一的存储器,尤其在结构网格上,算法涉及的数据依赖区结构有序,能较好满足 GPU 对齐合并访问的要求。但在非结构网格上,非结构化的数据依赖区,影响到访存效率。基于此提出一种适合高阶有限元算法框架的单元分层重排加速技术,致力于网格的层化结构,提升 GPU 访存效率。具体基于初始网格拓扑,创建单元或单元边界对应的分层结构,逐层重排,汇总形成适合 GPU 对齐合并访问的数据存储结构。文中结合排序实例,给出了这一重排加速技术的具体实施过程。算例表明,发展的算法逼近的阶数符合预期,计算结果能与现有文献或实验结果接近,且最大 GPU 加速比可达 67.47。此外,非结构网格算例证实,算法可处理较为复杂的几何边界,且所提重排技术可进一步赢得重排加速。

关键词:RKDG 方法;GPU;分层排序;非结构网格;Navier Stokes 方程
中图分类号:V211.3 **文献标志码**:A **文章编号**:0367-6234(2023)08-0032-11

A RKDG GPU parallel algorithm and its acceleration with reordering

GAO Huanqin[1,2], CHEN Hongquan[1,2], ZHANG Jiale[1,2], JIA Xuesong[1,2]

(1. College of Aerospace Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China; 2. Key Laboratory of Unsteady Aerodynamics and Flow Control (Nanjing University of Aeronautics and Astronautics), Ministry of Industry and Information Technology, Nanjing 210016, China)

Abstract: To enhance the parallel efficiency of solving Navier Stokes equations, a graphic processing unit (GPU) parallel algorithm, ported from Runge-Kutta discontinuous Galerkin (RKDG) method, is presented through constructing element-based or edge-based thread hierarchy and corresponding GPU kernels. The data storage and access of the algorithm are designed to be compatible for the various types of memories with different latencies. In comparison with the structured mesh counterpart, in which the structured domain of data dependence is already quite good for the requirement of coalesced memory access, the irregularity of unstructured mesh shows a negative effect on the performance of memory access. To remedy the negative effect, a multi-layered element reordering approach suitable for high-order finite element method is proposed to achieve further acceleration. Starting with the initial mesh, layer structures of elements or edges are constructed with reordering in a layer-by-layer manner to form the data structures suitable for coalesced memory access. An example of mesh reordering is provided with the implementation process detailed. Numerical results of typical flow simulations reveal that the expected order of accuracy of the proposed algorithm is realized, and the calculated results agree well with experiment data or other computed resules in the existing literature, with the maximum GPU speedups achieved up to 67.47. Moreover, the algorithm exhibits the potential to cope with more complex geometries, and the proposed technique can further achieve reordering acceleration.

Keywords: RKDG method; GPU; multi-layered reordering; unstructured mesh; Navier Stokes equations

间断 Galerkin(Discontinuous Galerkin method, DG)有限元方法源于 20 世纪 70 年代求解中子输运方程[1]。后期流行的 Runge-Kutta DG(RKDG)有限元方法[2-3],是结合求解非线性守恒律方程(组)提出的。该方法的主要特点是能够构造任意阶精度的格式(基于非结构网格等),但相较于传统的低阶精度格式,同一网格上控制方程的求解,因高阶近似而涉及的未知量更多,消耗的计算量也就更多,影响到

收稿日期:2022-08-11;**录用日期**:2022-10-31;**网络首发日期**:2023-03-10
网络首发地址:https://kns.cnki.net/kcms/detail/23.1235.T.20230310.1041.002.html
基金项目:国家自然科学基金(11972189,12102188)
作者简介:高缓钦(1990—),男,博士研究生;陈红全(1962—),男,教授,博士生导师
通信作者:陈红全,hqchenam@nuaa.edu.cn

实际复杂工程问题的推广应用[4-5]。因此，很有必要提高算法的计算效率。实现算法的并行化是可考虑的可行途径。

现代图形处理器（GPU）通常拥有数以百计或千计的计算核，浮点运算能力强劲，适合并行地处理数据密集且规模大的运算[6-8]。DG 有限元方法具有很好的局部紧致性，构造的高阶格式不需要非常宽的模板（stencil），因此，十分适合算法的 GPU 并行化[9]。早在 2009 年，Klöckner 等[9]就把 Nodal DG 有限元方法 GPU 化，求解了三维 Maxwell 方程。之后，Siebenborn 等[10]又发展用于求解三维 Euler 方程。Karakus 等[11]则拓展用于求解二维不可压 Navier Stokes 方程。与上述基于 Nodal DG 有限元方法不同，Xia 等[12]基于 Modal DG 有限元方法，实现了三维 Euler 方程 GPU 并行加速。Fuhry 等[13]则结合求解二维 Euler 方程，依据单元或单元边界构建所需的线程结构，开展了算法 GPU 并行化的深入研究，其并行化效率能与 Nodal DG GPU 算法求解线性问题[9]相当。Gao 等[14]进一步结合 Navier Stokes 方程，发展用于求解二维层流流动。不难看出，上述工作大多关注的是数据存储、线程结构和核函数构建等具体的 GPU 化问题，尚未涉及网格拓扑存在的单元不规则分布（非结构等）对并行效率的影响问题。就低阶精度格式的 GPU 并行化而言，这种影响不仅存在且是负面的，可采用单元[15]或点重排[16-17]等技术加以改善。因此有理由结合高阶精度 DG 格式，对此类问题加以深入研究。

本文结合求解 Navier Stokes 方程，选用 Modal DG 有限元方法，先构建出对应的 RKDG 算法，然后移植到 GPU 架构，发展出 GPU 加速的 RKDG 并行算法。该算法在结构网格上涉及的数据依赖区结构有序，但在非结构网格上，数据依赖区杂乱无序，影响到数据访存效率。为此，本文针对性地提出一种适合高阶 DG 有限元算法架构（任意阶）的单元分层排序方法，致力于改善并行效率。具体基于初始网格拓扑，创建单元或单元边界对应的分层结构，逐层重排，汇总形成适合 GPU 对齐合并访问（coalesced access）的数据存储结构。结合 RKDG GPU 算法的算例验证，展示出算法本身的 GPU 加速效果。同时，分析了网格拓扑对并行效率的影响，算例证实所提排序方法有助于进一步提升 GPU 并行化效率。

1　RKDG 有限元并行算法

1.1　流动控制方程

二维守恒形式可压缩 Navier-Stokes 方程为

$$\frac{\partial \boldsymbol{U}}{\partial t}+\nabla\cdot(\boldsymbol{F}_{\mathrm{i}}-\boldsymbol{F}_{\mathrm{v}})=0 \tag{1}$$

其中守恒变量 $\boldsymbol{U}$、无黏通量 $\boldsymbol{F}_{\mathrm{i}}=(\boldsymbol{f}_{\mathrm{i}},\boldsymbol{g}_{\mathrm{i}})$ 和黏性通量 $\boldsymbol{F}_{\mathrm{v}}=(\boldsymbol{f}_{\mathrm{v}},\boldsymbol{g}_{\mathrm{v}})$，可分别写为：

$$\boldsymbol{U}=\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ \rho E\end{pmatrix},\boldsymbol{f}_{\mathrm{i}}=\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ u(\rho E+p)\end{pmatrix},\boldsymbol{g}_{\mathrm{i}}=\begin{pmatrix}\rho v\\ \rho vu\\ \rho v^2+p\\ v(\rho E+p)\end{pmatrix},$$

$$\boldsymbol{f}_{\mathrm{v}}=\begin{pmatrix}0\\ \tau_{xx}\\ \tau_{xy}\\ u\tau_{xx}+v\tau_{xy}-q_x\end{pmatrix},\boldsymbol{g}_{\mathrm{v}}=\begin{pmatrix}0\\ \tau_{yx}\\ \tau_{yy}\\ u\tau_{yx}+v\tau_{yy}-q_y\end{pmatrix}$$

式中：密度 ρ、单位质量总能 E、压强 p 与沿坐标 x 和 y 方向的速度分量 u 和 v，满足如下状态方程：

$$E=\frac{1}{\gamma-1}\frac{p}{\rho}+\frac{1}{2}(u^2+v^2) \tag{2}$$

式中 γ 为流体的比热比，对于空气而言，$\gamma=1.4$。黏性通量项中，即：

$$\tau_{ij}=\mu\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)-\frac{2}{3}\mu\frac{\partial u_k}{\partial x_k}\delta_{ij},q_j=-\frac{\mu c_P}{Pr}\frac{\partial T}{\partial x_j} \tag{3}$$

式中：i、j 分别为 x 或 y 方向，δ_{ij} 为 Kronecker 德尔塔函数，T 为温度，c_P 为质量定压热容，Pr 为普朗特数，μ 为黏性系数，可由 Sutherland 公式计算确定。

1.2　间断 Galerkin 有限元离散

采用间断 Galerkin 有限元方法，将计算域 Ω 划分为 N_{c} 个互不重叠单元 Ω_i 的集合。定义有限元空间：

$$\boldsymbol{V}_{\mathrm{h}}^{p}=\{v_{\mathrm{h}}\in[L_2(\Omega)]^m:\boldsymbol{v}_{\mathrm{h}}|_{\Omega_i}\in[\boldsymbol{V}_p^m]\ \forall\Omega_i\in\Omega\} \tag{4}$$

式中：$\boldsymbol{V}_p^m$ 为单元 Ω_i 上最高次数不超过 p 阶的多项式 m 维函数的集合，m 为守恒变量的个数。对式（1）两端乘以测试函数 $\boldsymbol{v}_k$，并在任一单元 Ω_i 上进行分部积分，得到如下空间离散形式：

$$\int_{\Omega_i}\boldsymbol{v}_k\frac{\partial U_h}{\partial t}\mathrm{d}\Omega=-\int_{\partial\Omega_i}\boldsymbol{v}_k(\widetilde{\boldsymbol{F}}_{\mathrm{i}}-\widetilde{\boldsymbol{F}}_{\mathrm{v}})\cdot\boldsymbol{n}\mathrm{d}\sigma+\int_{\Omega_i}\nabla v_k\cdot(\boldsymbol{F}_{\mathrm{i}}-\boldsymbol{F}_{\mathrm{v}})\mathrm{d}\Omega \tag{5}$$

这里间断有限元边界上不唯一的通量已用唯一的数值通量 $\widetilde{\boldsymbol{F}}_{\mathrm{i}}$ 和 $\widetilde{\boldsymbol{F}}_{\mathrm{v}}$ 替换。由于黏性项存在二阶偏导数项，离散形式（5）还须进一步结合著名的 BR2 格式[18]，进一步改写为

$$\int_{\Omega_i}\boldsymbol{v}_k\frac{\partial \boldsymbol{U}_h}{\partial t}\mathrm{d}\Omega=-\int_{\partial\Omega_i}\boldsymbol{v}_k(\widetilde{\boldsymbol{F}}_{\mathrm{i}}(\boldsymbol{U}_h^+,\boldsymbol{U}_h^-)-\widetilde{\boldsymbol{F}}_{\mathrm{v}})\cdot\boldsymbol{n}\mathrm{d}\sigma+\int_{\Omega_i}\nabla\boldsymbol{v}_k\cdot(\boldsymbol{F}_{\mathrm{i}}(\boldsymbol{U}_h)-\boldsymbol{F}_{\mathrm{v}}(\boldsymbol{U}_h,\nabla\boldsymbol{U}_h+\boldsymbol{R}))\mathrm{d}\Omega \tag{6}$$

式中：$(\cdot)^+$、$(\cdot)^-$ 分别对应边界左、右单元，$\boldsymbol{n}$ 为边界外法向，$\boldsymbol{R}=\sum\boldsymbol{r}_e$，$\boldsymbol{r}_e$ 为局部提升算子。无黏数

值通量 $\widehat{\boldsymbol{F}}_{\mathrm{i}}$ 可由常规的 LLF 格式[19]确定，而黏性数值通量 $\widehat{\boldsymbol{F}}_{\mathrm{v}}$ 则沿用文献[18]的做法，结合稳定因子 η，可表示为

$$\widehat{\boldsymbol{F}}_{\mathrm{v}}=\frac{1}{2}(\boldsymbol{F}_{\mathrm{v}}(\boldsymbol{U}_h^{+},\nabla\boldsymbol{U}_h^{+}+\eta\boldsymbol{r}_e^{+})+\boldsymbol{F}_{\mathrm{v}}(\boldsymbol{U}_h^{-},\nabla\boldsymbol{U}_h^{-}+\eta\boldsymbol{r}_e^{-})) \tag{7}$$

有关 $\boldsymbol{r}_e$ 和 η 的具体求解过程详见文献[18]。单元上的近似解 $\boldsymbol{U}_h$ 可表示为

$$\boldsymbol{U}_h=\sum_{k=1}^{N_p}\widetilde{\boldsymbol{U}}_{i,k}(t)\boldsymbol{v}_k(\boldsymbol{x}) \tag{8}$$

那么，式(6)可转化为与待求的单元上解系数 $\widetilde{\boldsymbol{U}}_{i,k}(t)$ 对应的 N_p 个方程组，注意 $N_p=(p+1)(p+2)/2$。为统一起见，本文沿用文献[3]的做法，相关高斯积分均在参考单元上进行。对计算域所有单元，运用式(6)并进行汇总，得到的总体方程组可简写为

$$\boldsymbol{M}\frac{\partial\widetilde{\boldsymbol{U}}}{\partial t}=\mathrm{RHS} \tag{9}$$

式中：$\boldsymbol{M}$ 为块对角矩阵，每一块对应一个单元，$\widetilde{\boldsymbol{U}}$ 为汇总的解系数未知量，RHS 为对应的残值。采用强稳定性保持的三步龙格 - 库塔格式[20]，式(9)可进一步改写为：

$$\begin{aligned}\widetilde{\boldsymbol{U}}^{(1)}&=\widetilde{\boldsymbol{U}}^{n}+\Delta t\boldsymbol{M}^{-1}\mathrm{RHS}(\widetilde{\boldsymbol{U}}^{n})\\ \widetilde{\boldsymbol{U}}^{(2)}&=\frac{3}{4}\widetilde{\boldsymbol{U}}^{n}+\frac{1}{4}[\widetilde{\boldsymbol{U}}^{(1)}+\Delta t\boldsymbol{M}^{-1}\mathrm{RHS}(\widetilde{\boldsymbol{U}}^{(1)})]\\ \widetilde{\boldsymbol{U}}^{n+1}&=\frac{1}{3}\widetilde{\boldsymbol{U}}^{n}+\frac{2}{3}[\widetilde{\boldsymbol{U}}^{(2)}+\Delta t\boldsymbol{M}^{-1}\mathrm{RHS}(\widetilde{\boldsymbol{U}}^{(2)})]\end{aligned} \tag{10}$$

可以看出，上述构建的 RKDG 算法，相较于传统的低阶方法，相同网格上待求的未知量与 N_p 成正比，耗费的计算量往往难以承受[21]。因此有必要对算法进行并行化加速。

1.3 基于 GPU 的 RKDG 算法

采用 NVIDIA 公司发布的 CUDA C 编程语言，对 RKDG 算法进行 GPU 并行化。这一过程也可理解为算法的 CPU 程序移植到 GPU 上。GPU 内嵌平台大多拥有数以百计乃至千计的 CUDA 核(core)，适合并行地处理数据密集且规模大的运算，但在涉及逻辑判断和分支结构等的计算时表现不佳[6]。为此，可结合算法任务归类，与 CPU 协同运行。如图 1 所示，本文把前、后处理等保留在 CPU，而把最耗时的部分(迭代运算等)移植到 GPU。一旦 GPU 上指派的任务计算完成，就再把结果传回 CPU。可以看出，每一步迭代运算由计算时间步、积分点插值、解系数更新等子任务构成。每一个子任务对应一个由用户开发的核函数(kernel)。要实现算法的 GPU 化，就必须对各类子任务创建核函数。可以看到，就 RKDG 方法而言，待创建的核函数主要与高斯积分相关。

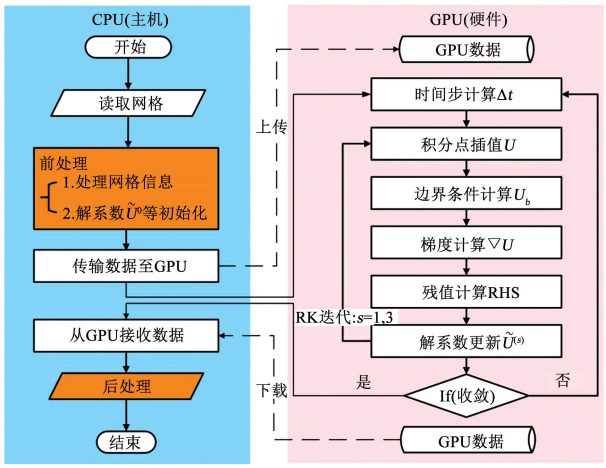


图 1 算法的 GPU 并行化架构

Fig. 1 General framework of GPU computing

1.3.1 核函数的构建说明

在 GPU 上运行一个并行任务(调用一个核函数)，是基于 Grid-Block-Thread 线程结构进行的。所有的线程(thread)被组织成若干个线程块(block)，再由 Block 汇聚成对应的线程网格(grid)。在线程结构中，块中线程号(threadIdx. x)、线程块号(blockIdx. x)是内置给定的，而线程块维度(blockDim. x)可由用户调节确定[6]。此外，内嵌的各类存储器，访存速度不一，其合理利用问题也是核函数创建时所必须考虑的。按访存速度的快慢排列，GPU 上的存储器主要有寄存器、常数存储器、共享存储器和全局存储器。全局存储器容量最大，适合存放算法产生的主要数据。共享存储器适合存储线程块中线程之间共享的数据，但容量相对不大。具有缓存加速等特点的常数存储器容量十分有限，适合存放规模不大且使用频繁的常量。寄存器在线程块中的数目有限，但访存速度最快，是需要考虑最大限度利用的对象。

如前述，DG 方法主要涉及单元和边界等积分计算。本文根据 DG 方法局部紧致的特点，沿用 Fuhry 等[13]的做法，对每一计算子任务，依据单元或单元边界构建所需的线程结构，使得任一单元积分(或边界积分)都有对应的一个线程负责其积分运算。依据这一原则，本文已对算法涉及的相关积分运算构建了对应的核函数。以残值计算为例(见式(6)右端项)，本文构建了单元积分核函数 k_surf 和单元边界积分核函数 k_line。k_surf 仅依赖于单元内部信息，积分运算可简单直接。比较而言，k_line涉及关联单元，相对复杂。这里结合 k_line 的

伪代码，就核函数构建给出实例说明。

同一单元所有边界积分一体计算，逐单元更新残值，可按单元循环进行。这种方式计算量依单元密集，适合 GPU 合并访问。但由于边界左、右单元数值通量（$\widehat{\boldsymbol{F}}_{\mathrm{i}} \cdot \boldsymbol{n}$ 和 $\widehat{\boldsymbol{F}}_{\mathrm{v}} \cdot \boldsymbol{n}$），在边界积分点处的取值是大小相等和符号相反的，计算中难免存在重复运算。为此，本文沿用文献[13]的做法，按边循环计算边界积分，构建 k_line，其伪代码及简要的注释如图 2 所示。利用块中线程号、线程块号和线程块维度，计算确定调用所需的线程总体编号 ID（第 1 行）；界定 ID 范围（第 2 行）；第 3 ~ 16 行是 k_line 的核心部分。该部分遍历边界积分点，分两步循环进行边界积分运算。第 1 步准备所需数据到快速寄存器（第 7 ~9 行）。内循环完成边界积分点的残值贡献量计算（11 ~13 行第 2 步）。在上述计算完成后，将数据输出存放到容量允许的全局存储器。注意，输出存放数据时，必须避免同一位置被不同线程同时写入的访问冲突。因此，在第 14 和 16 行中利用了 CUDA 提供的原子函数（atomicAdd）[6] 可避免此类问题。

对于算法涉及的诸如龙格 - 库塔时间推进等其他计算子任务，创建对应的核函数相对简单，但为了节省 GPU 内存，可从算法层面压缩物理量的存储，个别物理量可考虑用到时直接计算不再存储，本文不再一一赘述。可以看出，与线程结构相关的核函数，其创建还必须注意数据结构与存储问题，这些因素与 GPU 访存效率密切相关。

核函数 k_line()
　输入：常数存储器上的基函数 v^s 及全局存储器上的相关物理量
　输出：更新后的残值 RHS^s　！遍历所有边，ID∈[0, N_1-1], N_1 为边数
1 ID=threadIdx.x+blockIdx.x×blockDim.x ！线程编号
2 if ID < N_1 then
3　$\mathrm{RHS}_{\mathrm{L}}^s[j,k]\underset{(j,k)}{\longleftarrow}0$；$\mathrm{RHS}_{\mathrm{R}}^s[j,k]\underset{(j,k)}{\longleftarrow}0$ ！(j,k)循环残值初始化，j∈[0,3],k∈[0,N_p-1]
4　$cl^s=L^s[\mathrm{ID}]$；$cr^s=R^s[\mathrm{ID}]$　　！读取左右单元编号
5　for 边界积分点 i do　　！遍历所以积分点
6　　**Step1:读取数据，供(j,k)循环重复调用(从慢速全局存储器→到快速寄存器)**
7　　$\mathrm{ID}^s\leftarrow pL^s[i,\mathrm{ID}]$；$icr^s=pR^s[i,\mathrm{ID}]$　！读取积分点在左、右单元中的编号
8　　$\mathrm{flux}^s[j]\underset{(j)}{\longleftarrow}\mathrm{flux}^s[i,j,\mathrm{ID}]$；　！读取积分点上的数值通量
9　　$sJ^s\leftarrow sJ^s[i,\mathrm{ID}]$；　！读取雅克比值
10　　**Step2:根据当前积分点的贡献量更新左、右单元残值**($\mathrm{RHS}_{\mathrm{L}}^s$和$\mathrm{RHS}_{\mathrm{R}}^s$)
11　　$\mathrm{RHS}_{\mathrm{L}}^s[j,k]\underset{(j,k)}{\longleftarrow}\mathrm{RHS}_{\mathrm{L}}^s[j,k]+\mathrm{flux}^s[j]v^s[\mathrm{ID}^s,k]sJ^s$　！左单元更新
12　　if cr^s > 0 then
13　　　$\mathrm{RHS}_{\mathrm{R}}^s[j,k]\underset{(j,k)}{\longleftarrow}\mathrm{RHS}_{\mathrm{R}}^s[j,k]+\mathrm{flux}^s[j]v^s[\mathrm{ID}^s,k]sJ^s$　！右单元更新
14　$\mathrm{RHS}^s[j,k,cl^s]\underset{(j,k)}{\longleftarrow}\mathrm{RHS}^s[cl^s,j,k],-\mathrm{RHS}_{\mathrm{L}}^s[j][k]$　　！**数据输出存放到全局存储器**
15　if cr^s > 0 then
16　　$\mathrm{RHS}^s[j,k,cr^s]\underset{(j,k)}{\longleftarrow}\mathrm{atomicAdd}(\mathrm{RHS}^s[cr^s,j,k],-\mathrm{RHS}_{\mathrm{L}}^s[j][k])$　！

图 2　k_line 核函数伪代码

Fig. 2　Pseudocode of the k_line kernel

1.3.2　数据结构与存储问题

如前述，本文算法涉及的线程结构是依据单元或单元边界构建的，线程运行相对独立，因此上述开发的核函数仅涉及全局存储器、常数存储器和寄存器。具体来说，常数存储器用于存放参考单元基函数相关的物理量，寄存器用于处理循环重复利用（访存频率高）的数据（如 k_line 中的残值项处理），而全局存储器则用于存放计算产生的大容量数据。必须指出，减少算法对最慢的全局存储器的访存次数，有助于提升整体计算效率。通常可通过合并访问优化大容量数据的访存效率[22]。要做到这一点，就要结合算法构建合适的数据结构。为尽可能地满足合并访问的优化要求，本文把通常的多维数组映射到连续的一维数组，要求相同物理特征的变量依单元或单元边界编号连续存放。不失一般性，如果算法涉及的物理量为如下二维数组：

$$\boldsymbol{a}(i,j)=\begin{bmatrix} a_1^{(1)} & a_1^{(2)} & \cdots & a_1^{(N)} \\ a_2^{(1)} & a_2^{(2)} & \cdots & a_2^{(N)} \\ \vdots & \vdots & \ddots & \vdots \\ a_m^{(1)} & a_m^{(2)} & \cdots & a_m^{(N)} \end{bmatrix},i=1,\cdots,m,j=1,\cdots,N \tag{11}$$

式中：m 为变量维度（如守恒变量等），N 为单元数或单元边界数。把二维数组 $\boldsymbol{a}$ 映射到一维数组 $\boldsymbol{A}$，可直接表示为

$$\boldsymbol{A}=(a_1^{(1)},a_1^{(2)},\cdots,a_1^{(N)},a_2^{(1)},a_2^{(2)},\cdots,a_2^{(N)},\cdots,a_m^{(1)},a_m^{(2)},\cdots,a_m^{(N)}) \tag{12}$$

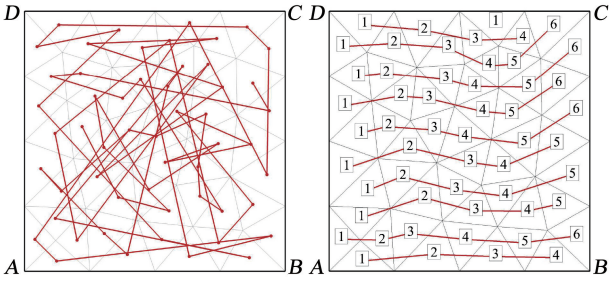
对于高阶 DG 算法特有的总体对角块质量矩阵（见式(9)），每一对角块相当于一个单元上的二维数组（$N_p \times N_p$），可按上述原则，先把所有对角块映射成一维数组，再按单元顺序排列形成最终的整体一维数组。可以看出，这样的数据结构是依赖于网格拓扑中单元或边排列顺序的，还不能确保算法依赖区信息的连续提取，进而影响到所谓的 GPU 合并访问。

2　重排加速技术

如果并发的 32 个线程（线程束[22]）访存的数据处在同一数据段，则只需要一次数据传输即可完成访存，否则需要多次传输。研究表明，访存效率一般与传输次数成反比[17]。将线程束需访问的数据集中置于尽可能少的数据段，可有效减少数据传输次数。可以看出，上述算法 GPU 化构建的核函数，单元相关的积分运算，是依赖于自身及其关联单元（相邻共边单元）的。对于依据单元编号构建的数据结构，提高访存效率的关键在于同一线程束访存单元的编号应尽可能接近或集中。这一点在复杂计算域非结构网格生成时，通常是难以顾及的。可以预见，在结构网格上，因单元排序的结构化，数据依赖单元相对集中，有助于线程束所需数据合并访问，访存效率会相对较高。

本文面向实用性强的非结构网格，借鉴无网格点云分层排序思想[17]，针对性地提出一种单元分层排序方法，力求获得层化结构，提升计算效率。具体基于任意初始网格单元拓扑，创建单元或单元边界对应的分层结构，逐层重排，汇总形成适合 GPU 对齐合并访问的单元排序。方法的具体实施过程可从图 3、4 排序实例看出。

图 3(a)为任意初始非结构网格，其单元初始顺序已用中心点连线示意。运用所提排序方法，先依据初始层创建对应的分层结构。初始层依附于计算域边界(如$\overrightarrow{AB}$等)，由该边界紧邻关联单元构成。然后逐层扫描新的关联单元，形成层化结构。初始层排序取为计算域边界自然的离散顺序。依照上一层单元的连接顺序逐层重排(图 3(b))，最后，依据层内序号，逐层汇总，得到整体排序(图 3(c))。图 4 给出了排序前后的单元编号顺序。

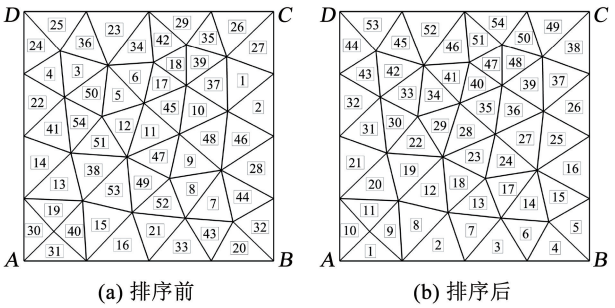


(a) 初始顺序(连线示意) (b) 层化结构与层内重排

(c) 汇总后的最终排序

图 3 排序方法实施实例

Fig. 3 An example for implementation of reordering



(a) 排序前 (b) 排序后

图 4 排序前、后单元编号顺序

Fig. 4 Element orders before and after reordering

经过单元分层排序后，邻近单元编号虽已相近，但基于边界积分的线程束，其依赖的关联单元是横跨不同层的，不利于合并访问。为此，在单元重排的基础上，还需要进行边排序。与单元排序类似，利用计算域边界$\overrightarrow{AB}$的自然离散顺序，创建边的初始层，再基于上述层化结构，遵循“边距相近，则边也相近”的原则，逐层提取并排序，汇总形成单元边整体排序。需要强调的是，在计算域边界上，应按边界条件的类型集中排序存放，这样做可减少线程束内分支操作，提高 GPU 并行效率[22]。

上述重排加速技术是基于任意非结构网格提出的，可作为一般网格生成后处理应用。本文将结合 Delaunay 网格生成，给出排序算法加速效果算例分析。

3 算例与分析

本文用上述 RKDG GPU 算法，先对具有解析解的库埃特(Couette)流动问题进行了数值模拟，验证了所提算法所能达到的精度，并就网格拓扑对 GPU 并行效率影响问题，给出了结构与非结构网格比较分析；接着，给出了 NACA0012 翼型及圆柱绕流的算例，展示出基于结构或非结构网格排序对算法并行效率的影响；最后，给出了多段翼型绕流算例，展示算法处理复杂气动外形的能力。为定量分析，算法的 T(每步迭代计算耗时)在 CPU 和 GPU 上分别记为 T_{CPU} 和 T_{GPU}，单元重排后记为 T^{+}_{GPU}，那么 GPU 加速比和重排加速比可分别定义为 $T_{\mathrm{CPU}}/T_{\mathrm{GPU}}$ 和 $T_{\mathrm{GPU}}/T^{+}_{\mathrm{GPU}}$。本文仿照传统做法[17]，$T$ 统一取为 1 000 次 Runge-Kutta 迭代的平均值(单一时间步计算耗时)。所有算例均在 Windows 工作平台上采用双精度计算。该平台配备了英特尔 i5-3450 CPU 和英伟达 GTX TITAN GPU，其相关参数已在表 1 中列出。

表 1 i5-3450 CPU 和 GTX TITAN GPU 的参数

Tab. 1 Specifications of i5-3450 CPU and GTX TITAN GPU

硬件	理论性能		内存(大小或个数)			
	双精度浮点运算/10^9 (Flop·s^{-1})	内存带宽/(GB·s^{-1})	全局存储器/GB	常数存储器/KB	寄存器/(个·block^{-1})	共享存储器/(KB·block^{-1})
i5-3450 CPU	99.2	25.6	16	–	–	–
GTX TITAN GPU	1 500.0	288.0	6	64	49 152	48

3.1 库埃特流动

选用经典库埃特模型化流动[23]对发展的算法进行考核计算。该流动是两块无限长平行平板间的

层流流动，下面的平板水平放置且固定不动，上面的平板则以速度 U 水平向右运动。假定黏性系数 μ 为常数，板间距为 2，则该流动有如下形式的解析解：

$$u=\frac{y}{2}U, v=0, p=p_\infty,$$

$$T=T_0+\frac{y}{2}(T_1-T_0)+\frac{PrU^2}{2c_p}\frac{y}{2}\left(1-\frac{y}{2}\right), \rho=\frac{p}{RT} \tag{13}$$

沿用文献[23]的参数设置，$U=1$，下壁面温度 $T_0=0.80$，上壁面温度 $T_1=0.85$ 和马赫数 $Ma_1=0.1$，黏性系数 $\mu=0.001$ 时的雷诺数 $Re=100$。用结构或非结构网格离散计算域（$0\leqslant x\leqslant 4, 0\leqslant y\leqslant 2$），并用精确的解析解施加所需的边界条件。为了验证算法计算精度所能达到的阶数，两种计算网格均按比例进行了剖分加密（如图 5 所示），并用密度的绝对误差（数值解与解析解的差值）的 L2 范数指示计算误差，进行了不同阶数算法的测试。测试结果已在表 2 中列出。可从表 2 中看出，数值解逼近的阶数在密网格上更符合预期。比较而言，结构网格优于非结构网格，高阶格式如预期比低阶格式更精确（表现为 L2 误差更小）。

接着，基于结构与非结构网格，构建了非结构区域占比约为 0%、33%、67% 和 100% 的一组网格（对应图 6 中网格 G 到 J），用于测试网格不规则程度对 GPU 加速的影响。注意，为排除网格单元类型的影响，结构和非结构区域均统一用三角形单元离散。表 3 列出了这一测试结果。从表 3 中可看出，发展的不同阶精度算法都能实现期望的 GPU 加速，加速比最高可达 58.46；加速比大小是与非结构区域占比相关的，占比越高则加速比越小，体现出网格不规则程度对 GPU 加速的不利影响。

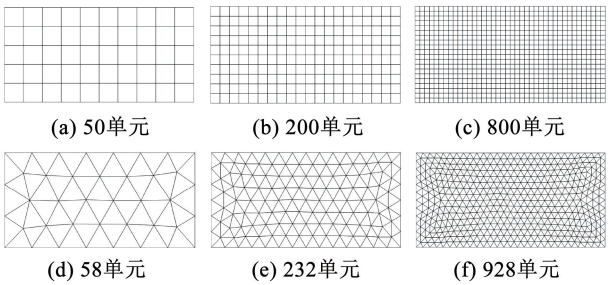


图 5　渐次加密的结构（(a)～(c)）与非结构（(d)～(f)）网格

Fig. 5　Successively refined meshes (structured: (a) to (c); unstructured: (d) to (f))

表 2　RKDG GPU 算法不同阶精度测试结果

Tab. 2　Test results of RKDG GPU algorithms with different orders

算法	结构网格（四边形）			非结构网格（三角形）		
	网格（大小）	L2 误差	阶数	网格（大小）	L2 误差	阶数
2 阶（$p=1$）	50	3.45×10^{-5}	–	58	5.59×10^{-5}	–
	200	7.90×10^{-6}	2.13	232	1.55×10^{-5}	1.85
	800	1.93×10^{-6}	2.03	928	4.06×10^{-6}	1.93
3 阶（$p=2$）	50	2.16×10^{-7}	–	58	4.30×10^{-7}	–
	200	2.66×10^{-8}	3.02	232	6.73×10^{-8}	2.68
	800	3.26×10^{-9}	3.03	928	8.70×10^{-9}	2.95
4 阶（$p=3$）	50	6.22×10^{-10}	–	58	1.85×10^{-9}	–
	200	3.86×10^{-11}	4.01	232	1.57×10^{-10}	3.56
	800	2.42×10^{-12}	4.00	928	1.09×10^{-11}	3.85

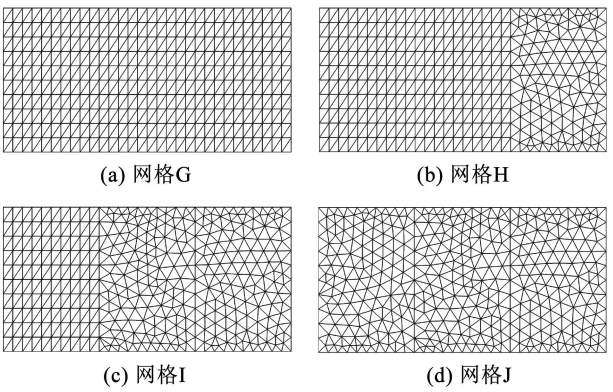


图 6　网格构建示意

Fig. 6　Schematic diagram of mesh construction

表 3　算法不同阶精度加速特性测试结果

Tab. 3　GPU speedups of algorithms with different orders

算法	网格（非结构占比/%）	计算耗时/s		加速比
		T_{CPU}	T_{GPU}	T_{CPU}/T_{GPU}
2 阶（$p=1$）	G(0)	$5.452\,1\times10^{-1}$	$9.326\,0\times10^{-3}$	58.46
	H(33)	$5.694\,3\times10^{-1}$	$1.084\,8\times10^{-2}$	52.49
	I(67)	$6.050\,5\times10^{-1}$	$1.275\,6\times10^{-2}$	47.43
	J(100)	$6.407\,2\times10^{-1}$	$1.398\,7\times10^{-2}$	45.81
3 阶（$p=2$）	G(0)	$1.140\,3\times10^{0}$	$2.140\,7\times10^{-2}$	53.27
	H(33)	$1.171\,5\times10^{0}$	$2.489\,7\times10^{-2}$	47.05
	I(67)	$1.255\,6\times10^{0}$	$2.873\,4\times10^{-2}$	43.70
	J(100)	$1.277\,9\times10^{0}$	$3.219\,6\times10^{-2}$	39.69
4 阶（$p=3$）	G(0)	$2.476\,7\times10^{0}$	$4.940\,5\times10^{-2}$	50.13
	H(33)	$2.515\,5\times10^{0}$	$5.630\,3\times10^{-2}$	44.68
	I(67)	$2.599\,2\times10^{0}$	$6.562\,7\times10^{-2}$	39.61
	J(100)	$2.677\,7\times10^{0}$	$7.398\,7\times10^{-2}$	36.19

3.2 NACA0012 翼型亚声速黏性绕流

NACA0012 翼型亚声速黏性绕流[24]经常被用来考核发展的高阶算法。本文先用 2 阶、3 阶和4 阶算法对该绕流进行了数值模拟。计算网格由 2 240 个四边形单元构成[24]，翼型表面仅有 41 个点(如图 7所示)。计算来流马赫数为 0.5，攻角为 1°，雷诺数为 5 000。随同文献计算结果[25]，表 4 给出了计算得到的阻力系数。可以注意到，随着算法阶数的提高，得到的阻力系数能与文献 5 阶结果接近，对应的马赫数等值线也变得清晰光滑(如图 8 所示)。

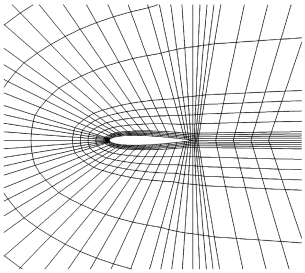


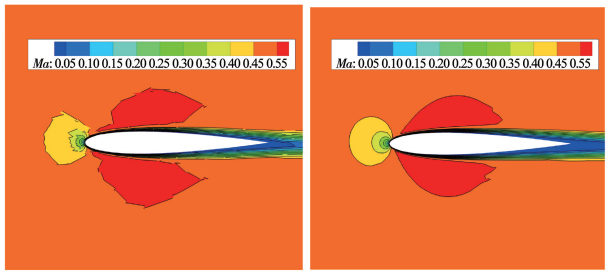
图 7 NACA0012 翼型计算网格

Fig. 7 Mesh for NACA0012 airfoil

表 4 阻力系数计算结果

Tab. 4 Computed drag coefficients

算法	C_D
2 阶($p=1$)	$6.341\,7\times10^{-2}$
3 阶($p=2$)	$5.714\,7\times10^{-2}$
4 阶($p=3$)	$5.555\,5\times10^{-2}$
5 阶(文献[25])	$5.531\,7\times10^{-2}$



(a) 2阶($p=1$) (b) 4阶($p=3$)

图 8 马赫数云图(2 阶与 4 阶)

Fig. 8 Mach contours (second order and fourth order)

接着，本文依据如图 9 所示的初始网格，构建了一套渐次加密的结构网格网格规模依次为 170 × 30、340 × 60、680 × 120 和 1 360 × 240，进行了算法加速性能的测试。大体上，在同一阶数下，加速比随着网格规模增大而提高(表 5 第 6 列)，最大加速比可达 67.47。表 5 同时列出了重排加速比，如预期，接近于 1 且几乎不随网格变化，表明网格的结构化已很适合 GPU 对齐合并访问，无需进一步重排加速。

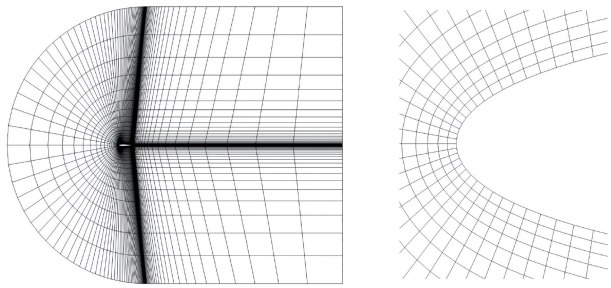


图 9 初始结构网格(170 × 30)

Fig. 9 Initial structured mesh (170 × 30)

表 5 不同阶算法 NACA0012 翼型绕流加速特性测试结果

Tab. 5 Speedups of algorithms with different orders for flow over NACA0012 airfoil

算法	网格规模	计算耗时/s			加速比	
		T_{CPU}	T_{GPU}	T^{+}_{GPU}	T_{CPU}/T_{GPU}	T_{GPU}/T^{+}_{GPU}
2 阶($p=1$)	170 × 30	$6.275\,1\times10^{-2}$	$2.831\,1\times10^{-3}$	$2.809\,2\times10^{-3}$	22.16	1.01
	340 × 60	$2.614\,2\times10^{-1}$	$5.435\,8\times10^{-3}$	$5.294\,9\times10^{-3}$	48.09	1.03
	680 × 120	$1.052\,7\times10^{0}$	$1.644\,6\times10^{-2}$	$1.635\,3\times10^{-2}$	64.01	1.01
	1 360 × 240	$4.151\,7\times10^{0}$	$6.153\,8\times10^{-2}$	$6.090\,5\times10^{-2}$	67.47	1.01
3 阶($p=2$)	170 × 30	$1.497\,4\times10^{-1}$	$4.673\,2\times10^{-3}$	$4.609\,8\times10^{-3}$	32.04	1.01
	340 × 60	$5.958\,5\times10^{-1}$	$1.271\,6\times10^{-2}$	$1.254\,9\times10^{-2}$	46.86	1.01
	680 × 120	$2.378\,9\times10^{0}$	$4.325\,7\times10^{-2}$	$4.294\,0\times10^{-2}$	54.99	1.01
	1 360 × 240	$9.945\,1\times10^{0}$	$1.693\,0\times10^{-1}$	$1.665\,7\times10^{-1}$	58.74	1.02
4 阶($p=3$)	170 × 30	$3.058\,3\times10^{-1}$	$7.961\,4\times10^{-3}$	$7.942\,0\times10^{-3}$	38.41	1.00
	340 × 60	$1.236\,5\times10^{0}$	$2.419\,3\times10^{-2}$	$2.390\,0\times10^{-2}$	51.11	1.01
	680 × 120	$5.208\,1\times10^{0}$	$8.953\,2\times10^{-2}$	$8.730\,4\times10^{-2}$	58.17	1.03
	1 360 × 240	$2.075\,0\times10^{1}$	$3.510\,4\times10^{-1}$	$3.496\,2\times10^{-1}$	59.11	1.00

3.3　圆柱绕流

圆柱绕流卡门涡街[26]常用来考核高阶算法的低耗散性。计算来流马赫数为 0.2，雷诺数为 100。计算网格由整体非结构网格（20 376 个三角形单元）和局部结构网格（圆柱附近，1 560 个四边形单元）混合构成（如图 10 所示）。本文先用不同阶数的算法对该绕流进行了数值模拟。总体上，阶数不同的算法都能捕捉到卡门涡街，阶数越高则涡结构越清晰，表现为涡量等值线越光滑（如图 11 所示）。图 12 给出了对应的升、阻力系数周期解，表 6 则列出了对应的斯特哈尔数（Strouhal）。可从中看出，对应 3 阶和 4 阶的计算斯特哈尔数能与文献[26]计算和实验[27]结果接近。

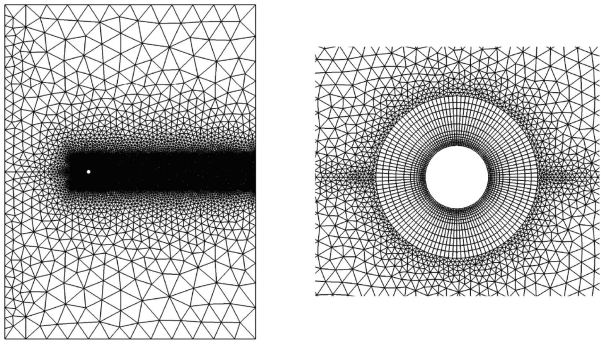
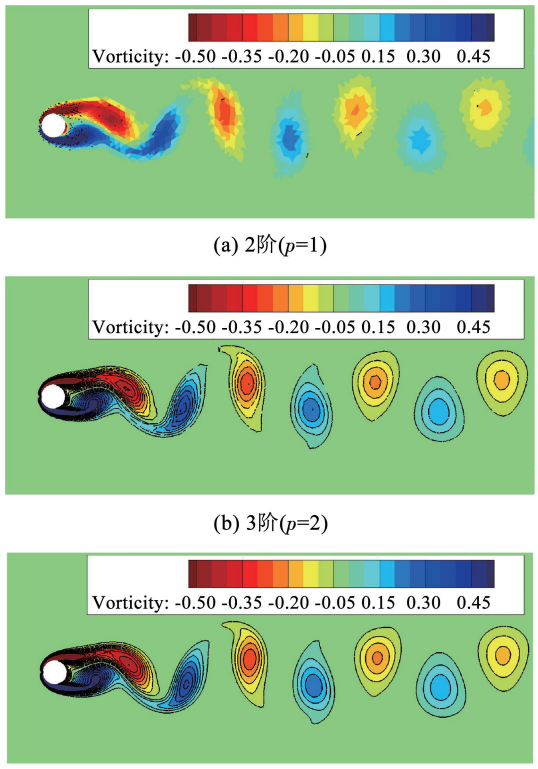


图 10　圆柱绕流计算网格

Fig. 10　Hybrid mesh around a circle cylinder



(a) 2阶(p=1)

(b) 3阶(p=2)

(c) 4阶(p=3)

图 11　圆柱绕流涡量等值线

Fig. 11　Vorticity contours for flow over a circle cylinder

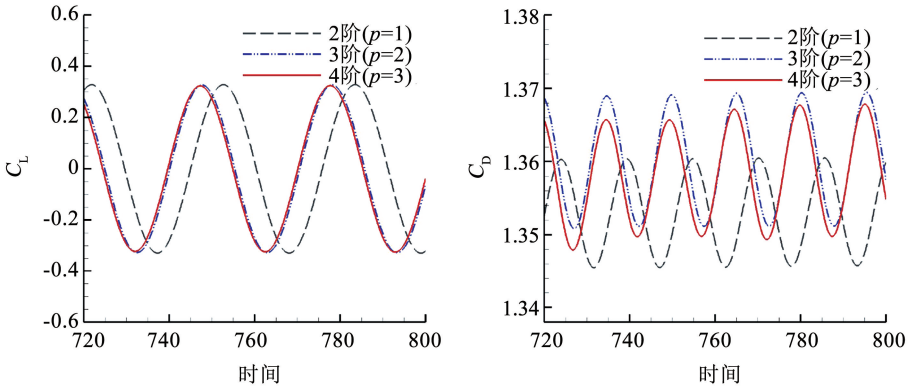


图 12　计算得到的升、阻力系数周期解

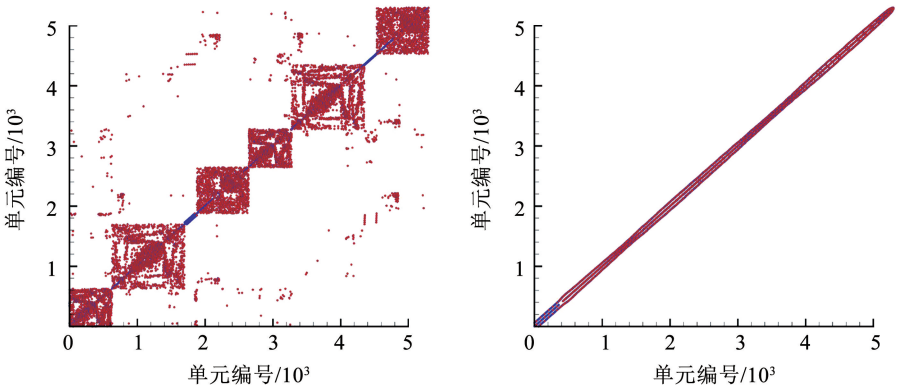
Fig. 12　Periodic solutions of calculated lift and drag coefficients

表 6　斯特哈尔数

Tab. 6　Strouhal values

数据来源	算法	Strouhal
本文	2 阶（p = 1）	0.162 3
	3 阶（p = 2）	0.163 9
	4 阶（p = 3）	0.164 5
文献	实验[27]	0.165 0
	计算[26]	0.166 0

接着，对图 10 所示的网格进行粗化和细化操作，由此形成一套渐次加密的结构与非结构混合网格（单元数分别为 5 300、21 936 和 79 428），进行算法加速特性测试。图 13 给出了本算例单元关联点阵图，图 13 中纵、横向坐标都为单元编号，点代表两单元关联。从图 13 中可以看出，本例初始网格对应点阵无序散布，重排后，关联单元编号已十分接近，表现为点阵向主对角集中，呈清晰的条带状，展示出所提重排算法对网格排序的影响。大体上，在同一阶数下，所有加速比随着网格规模增大而提高，GPU 加速比最大可达 45.68（表 7 第 6 列），最大重排加速比达 1.28（表 7 第 7 列），实现了近 30% 的进一步 GPU 加速。表明整体非结构化的混合网格，是与上述结构网格算例不同的，有必要进一步结合重排加速。



(a) 排序前 (b) 排序后

图 13 结构与非结构混合网格单元关联点阵图

Fig. 13 Elemental correlation matrix for hybrid meshes

表 7 圆柱绕流不同阶数算法 GPU 加速特性测试结果

Tab. 7 GPU speedups of algorithms with different orders for flow over a cycle cylinder

算法	网格规模	计算耗时/s			加速比	
		T_{CPU}	T_{GPU}	T_{GPU}^{+}	T_{CPU}/T_{GPU}	T_{GPU}/T_{GPU}^{+}
2 阶($p=1$)	5 300	$5.676\,3\times10^{-2}$	$2.928\,2\times10^{-3}$	$2.826\,0\times10^{-3}$	19.38	1.04
	21 936	$2.461\,2\times10^{-1}$	$6.140\,3\times10^{-3}$	$5.124\,1\times10^{-3}$	40.08	1.20
	79 428	$9.197\,8\times10^{-1}$	$2.013\,5\times10^{-2}$	$1.590\,8\times10^{-2}$	45.68	1.27
3 阶($p=2$)	5 300	$1.196\,2\times10^{-1}$	$5.373\,0\times10^{-3}$	$5.213\,2\times10^{-3}$	22.26	1.03
	21 936	$5.034\,3\times10^{-1}$	$1.343\,3\times10^{-2}$	$1.134\,6\times10^{-2}$	37.48	1.18
	79 428	$1.849\,9\times10^{0}$	$4.390\,9\times10^{-2}$	$3.465\,3\times10^{-2}$	42.13	1.27
4 阶($p=3$)	5 300	$2.564\,7\times10^{-1}$	$9.152\,7\times10^{-3}$	$9.385\,9\times10^{-3}$	28.02	1.09
	21 936	$1.073\,5\times10^{0}$	$2.615\,7\times10^{-2}$	$2.205\,1\times10^{-2}$	41.04	1.19
	79 428	$3.887\,2\times10^{0}$	$9.070\,4\times10^{-2}$	$7.068\,6\times10^{-2}$	42.86	1.28

3.4 NLR7301 多段翼型亚声速绕流

为了展示发展的算法处理复杂气动外形绕流的能力,这里给出了 NLR7301 多段翼型[28]亚声速无黏绕流算例,来流条件为马赫数 0.187 和攻角 6°。计算网格为一套渐次加密的非结构网格,单元数依次分别为 5 030、19 850、78 370 和 314 704。两段翼型和远场边界在初始网格(如图 14 所示)中,表面网格点数相同,均取为 65 个点。本文先基于初始网格,用发展的 3 阶 RKDG GPU 算法对该绕流问题进行了数值模拟。计算得到的压力云图和表面压强系数分布已在图 15 中给出,图中同时给出了文献[28]计算结果和实验结果[29]供比较。接着,以主翼、襟翼和远场边界作为重排算法的起始边界,进行了不同分层方向对重排加速比的影响测试,大体上影响不大(见表 8)。

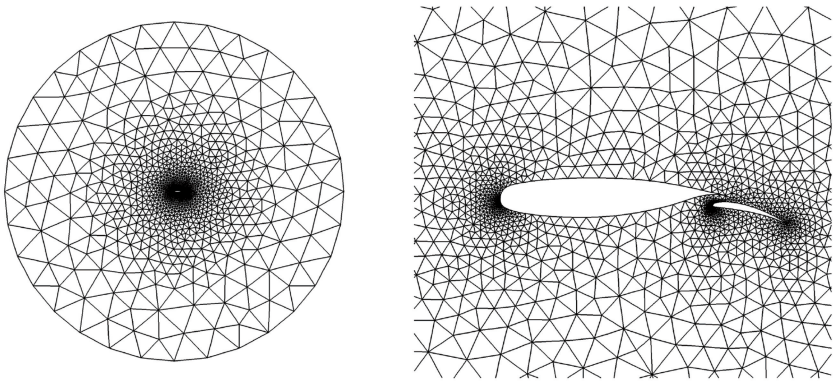
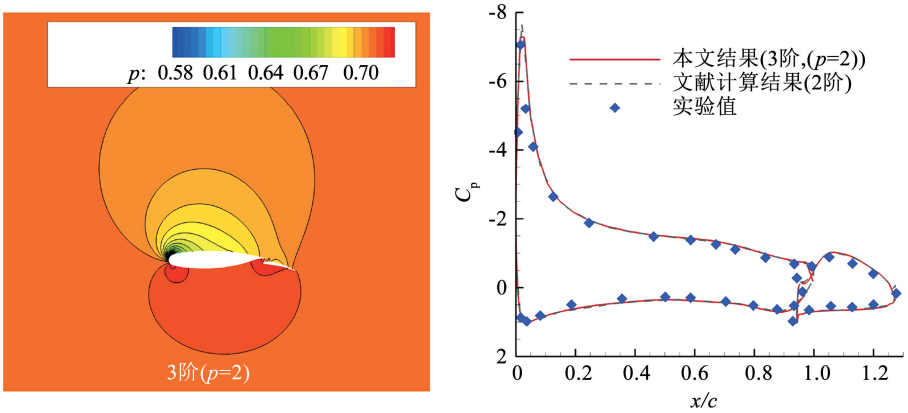


图 14 NLR7301 翼型绕流的计算网格

Fig. 14 Computational mesh for flow over a NLR7301 airfoil



(a) 压力云图　　(b) 表面压强系数分布

图 15　NLR7301 翼型绕流的计算结果

Fig. 15　Results for flow over a NLR7301 airfoil

表 8　不同起始边界对重排加速比的影响

Tab. 8　Effects of reordering with different starting boundaries

网格规模	起始边界		
	主翼	襟翼	远场
19 850	1.19	1.20	1.21
78 370	1.26	1.26	1.27
314 704	1.36	1.36	1.36

不失一般性，表 9 列出了以主翼作为起始边界的测试结果，表中结果再次证实，对于非结构网格，GPU 加速比和重排加速比都随着网格规模增大而提高，这在一定程度上表明，对于涉及大规模非结构网格的数值模拟问题，算法的 GPU 化和重排加速是很有必要的。

表 9　计算耗时和加速比（NLR7301 多段翼型，3 阶格式（$p=2$））

Tab. 9　Consuming time and speedups (NLR7301 multi-airfoil, third order ($p=2$))

算法	网格规模	计算耗时/s			加速比	
		T_{CPU}	T_{GPU}	T^{+}_{GPU}	T_{CPU}/T_{GPU}	T_{GPU}/T^{+}_{GPU}
3 阶($p=2$)	5 030	$3.761\,1\times10^{-2}$	$2.628\,4\times10^{-3}$	$2.521\,2\times10^{-3}$	14.31	1.04
	19 850	$1.597\,5\times10^{-1}$	$6.277\,3\times10^{-3}$	$5.256\,8\times10^{-3}$	25.45	1.19
	78 370	$6.470\,7\times10^{-1}$	$2.030\,5\times10^{-2}$	$1.610\,9\times10^{-2}$	31.87	1.26
	314 704	$2.776\,3\times10^{0}$	$7.509\,1\times10^{-2}$	$5.526\,7\times10^{-2}$	39.67	1.36

4　结　论

1）算法涉及的线程结构是依据单元和单元边界构建的，使得阶数不同的算法可采用统一的程序结构，线程资源能得以有效利用。

2）算法结构网格上结构有序的数据依赖区，已能较好满足 GPU 对齐合并访问的要求，无需重排加速。

3）算法在非结构等混合网格上，数据依赖区非结构，重排加速是很有必要的。重排后的网格，层化结构明显，有助于 GPU 对齐合并访问，使得算法赢得 GPU 加速基础上的进一步重排加速。

4）网格规模大的数值模拟问题更能赢得大的 GPU 加速。

5）算法是依据任意类型有限元单元设计的，适合涉及非结构等各类混合网格的复杂气动外形数值模拟问题。

参考文献

[1] REED W H, HILL T R. Triangle mesh methods for the neutron transport equation: LA-UR-73-479[R]. Los Alamos: Los Alamos Scientific Laboratory, 1973

[2] COCKBURN B, HOU Chiwang, SHU C W. The Runge-Kutta local projection discontinuous Galerkin finiteelement method for conservation Laws. Ⅳ: The multidimensional case[J]. Mathematics of Computation, 1990, 54(190): 545. DOI: 10.2307/2008501

[3] COCKBURN B, SHU Chiwang. The Runge-Kutta discontinuous galerkin method for conservation laws V: Multidimensional systems [J]. Journal of Computational Physics, 1998, 141(2): 199. DOI: 10.1006/jcph.1998.5892

[4] 蔚喜军，周铁. 流体力学方程的间断有限元方法[J]. 计算物理，2005，22(2)：108
YU Xijun, ZHOU Tie. Discontinuous finite element methods for solving hydrodynamic equations[J]. Chinese Journal of Computational Physics, 2005, 22(2): 108. DOI: 10.19596/j.cnki.1001-246x.2005.02.001

[5] 龚小权，贾洪印，陈江涛，等. 基于雅可比矩阵精确计算的 GMRES 隐式方法在间断 Galerkin 有限元中的应用[J]. 空气动

力学学报, 2019, 37(1): 121
GONG Xiaoquan, JIA Hongyin, CHEN Jiangtao et al. Applications of GMRES based on exact calculations of Jacobian matrix in discontinuous Galerkin methods[J]. Acta Aerodynamic a Sinica, 2019, 37(1): 121. DOI: 10.7638/kqdlxxb-2018.0189

[6] ZHANG Jiale, CHEN Hongquan, CAO Cheng. A graphics processing unit-accelerated meshless method for two-dimensional compressible flows[J]. Engineering Applications of Computational Fluid Mechanics, 2017, 11(1): 526. DOI: 10.1080/19942060.2017.1317027

[7] ZIMMERMAN B J, WANG Z J. The efficient implementation of correction procedure via reconstruction with GPU computing[C]// Proceedings of the 21st AIAA Computational Fluid Dynamics Conference. Reston, Virginia: AIAA, 2013: AIAA2013-2692. DOI: 10.2514/6.2013-2692

[8] SRIDHAR A, TISSAOUI Y, MARRAS S, et al. Large-eddy simulations with ClimateMachine v0.2.0: a new open-source code for atmospheric simulations on GPUs and CPUs[J]. Geoscientific Model Development, 2022, 15(15): 6259. DOI: 10.5194/gmd-15-6259-2022

[9] KLÖCKNER A, WARBURTON T, BRIDGE J, et al. Nodal discontinuous Galerkin methods on graphics processors[J]. Journal of Computational Physics, 2009, 228(21): 7863. DOI: 10.1016/j.jcp.2009.06.041

[10] SIEBENBORN M, SCHULZ V, SCHMIDT S. A curved-element unstructured discontinuous Galerkin method on GPUs for the Euler equations[J]. Computing and Visualization in Science, 2012, 15(2): 61. DOI: 10.1007/s00791-013-0197-0

[11] KARAKUS A, CHALMERS N, ŚWIRYDOWICZ K, et al. A GPU accelerated discontinuous Galerkin incompressible flow solver[J]. Journal of Computational Physics, 2019, 390: 380. DOI: 10.1016/j.jcp.2019.04.010

[12] XIA Yidong, LUO Lixiang, LUO Hong, et al. OpenACC-based GPU acceleration of a 3-D unstructured discontinuous Galerkin method[C]//Proceedings of the 52nd Aerospace Sciences Meeting. Reston, Virginia: AIAA, 2014: AIAA2014-1129. DOI: 10.2514/6.2014-1129

[13] FUHRY M, GIULIANI A, KRIVODONOVA L. Discontinuous Galerkin methods on graphics processing units for nonlinear hyperbolic conservation laws[J]. International Journal for Numerical Methods in Fluids, 2014, 76(12): 982. DOI: 10.1002/fld.3963

[14] GAO Huanqin, CHEN Hongquan, ZHANG Jiale, et al. A GPU-accelerated discontinuous Galerkin method for solving two-dimensional laminar flows[J]. Transactions of Nanjing University of Aeronautics & Astronautics, 2022, 39(4): 450. DOI: 10.16356/j.1005-1120.2022.04.007

[15] LACASTA A, MORALES-HERNÁNDEZ M, MURILLO J, et al. An optimized GPU implementation of a 2D free surface simulation model on unstructured meshes[J]. Advances in Engineering Software, 2014, 78: 1. DOI: 10.1016/j.advengsoft.2014.08.007

[16] MA Z H, WANG H, PU S H. GPU computing of compressible flow problems by a meshless method with space-filling curves[J]. Journal of Computational Physics, 2014, 263: 113. DOI: 10.1016/j.jcp.2014.01.023

[17] CAO Cheng, CHEN Hongquan, ZHANG Jiale, et al. A multi-layered point reordering study of GPU-based meshless method for compressible flow simulations[J]. Journal of Computational Science, 2019, 33: 45. DOI: 10.1016/j.jocs.2019.04.001

[18] BASSI F, CRIVELLINI A, REBAY S, et al. Discontinuous Galerkin solution of the Reynolds-averaged Navier-Stokes and k-ω turbulence model equations[J]. Computers & Fluids, 2005, 34(4/5): 507. DOI: 10.1016/j.compfluid.2003.08.004

[19] ZHANG Fan, CHENG Jian. A bound-preserving and positivity-preserving finite volume WENO scheme for solving five-equation model of two-medium flows[J]. Communications in Nonlinear Science and Numerical Simulation, 2022, 114: 106649. DOI: 10.1016/j.cnsns.2022.106649

[20] 韩德超, 刘卫华, 司文朋. 三角网格间断有限元法弹性波模拟精度分析[J]. 石油地球物理勘探, 2021, 56(4): 758
HAN Dechao, LIU Weihua, SI Wenpeng. Analysis of elastic wave simulation accuracy with discontinuous Galerkin finite element method based on triangular meshes[J]. Oil Geophysical Prospecting, 2021, 56(4): 758. DOI: 10.13810/j.cnki.issn.1000-7210.2021.04.009

[21] 夏轶栋, 伍贻兆, 吕宏强, 等. 高阶间断有限元法的并行计算研究[J]. 空气动力学学报, 2011, 29(5): 537
XIA Yidong, WU Yizhao, LV Hongqiang, et al. Parallel computation of a high-order discontinuous Galerkin method on unstructured grids[J]. Acta Aerodynamic a Sinica, 2011, 29(5): 537. DOI: 10.3969/j.issn.0258-1825.2011.05.001

[22] NVIDIA. CUDA C++ programming guide v11.7[Z/OL]. 2022-10-01. https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[23] CHENG Jian, LIU Tiegang, LUO Hong. A hybrid reconstructed discontinuous Galerkin method for compressible flows on arbitrary grids[J]. Computers & Fluids, 2016, 139: 68. DOI: 10.1016/j.compfluid.2016.04.001

[24] NASA. International Workshop on High-Order CFD Methods[Z/OL]. 2021-04-01. https://www1.grc.nasa.gov/research-and-engineering/hiocfd/

[25] BALAN A, WOOPEN M, MAY G. Hp-adaptivity on anisotropic meshes for hybridized discontinuous Galerkin scheme[C]// Proceedings of the 22nd AIAA Computational Fluid Dynamics Conference. Reston, Virginia: AIAA, 2015: AIAA2015-2606. DOI: 10.2514/6.2015-2606

[26] HENNINK A, TIBERGA M, LATHOUWERS D. A pressure-based solver for low-Mach number flow using a discontinuous Galerkin method[J]. Journal of Computational Physics, 2021, 425: 109877. DOI: 10.1016/j.jcp.2020.109877

[27] WANG Anbang, TRÁVNÍČEK Z, CHIA K C. On the relationship of effective Reynolds number and Strouhal number for the laminar vortex shedding of a heated circular cylinder[J]. Physics of Fluids, 2000, 12(6): 1401. DOI: 10.1063/1.870391

[28] 张加乐. 面向求解三维复杂流动问题的 GPU 并行算法及其应用研究[D]. 南京: 南京航空航天大学, 2018
ZHANG Jiale. Research on GPU-accelerated numerical methods and their applications for three-dimensional complex flows[D]. Nanjing: Nanjing University of Aeronautics and Astronautics, 2018. DOI: 10.27239/d.cnki.gnhhu.2018.000193

[29] VAN DEN BERG B, OSKAM B. Boundary layer measurements on a two-dimensional wing with flap and a comparison with calculations: NLR MP 79034 U[R]. Netherlands: National Laboratorium Luchten Ruimtevoart, 1979

(编辑 张 红)