International Journal of Fluid Dynamics 流体动力学, 2018, 6(2), 15-22
Published Online June 2018 in Hans. http://www.hanspub.org/journal/ijfd
https://doi.org/10.12677/ijfd.2018.62003

Hans汉斯

Accelerating the Runge-Kutta Discontinuous Galerking Method for Solving Two-Dimensional Flow on Multi GPUs

Xingyu Zhou, Tiegang Liu

LMIB and School of Mathematics and Systems Science, Beijing University of Aeronautics and Astronautics, Beijing
Email: xy_zhou@buaa.edu.cn, liutg@buaa.edu.cn

Received: May 15th, 2018; accepted: May 31st, 2018; published: Jun. 7th, 2018

Abstract

It is time-consuming to use Runge-Kutta discontinuous Galerkin method to solver flow field. In this article, we use multi GPUs to accelerate computing of two-dimension NACA0012 airfoil. The flow mesh is divided into several blocks according to the number of GPUs. We specialize kernels with a one-element-per-thread strategy, and use MPI to communicate data among computing nodes. In the process of communication, we use CUDA stream and MPI non-blocking operation to overlap computation and communication. The result shows when compared with the serial CPU program, the speedup ratio of GPU code running on one, two, four GPUs is around 33, 59 and 108.

Keywords

RKDG, Multi GPUs, MPI

龙格库塔间断有限元方法求解二维欧拉方程的多GPU加速实现

周星宇，刘铁钢

北京航空航天大学数学与系统科学学院，数学、信息与行为教育部重点实验室，北京
Email: xy_zhou@buaa.edu.cn, liutg@buaa.edu.cn

收稿日期：2018年5月15日；录用日期：2018年5月31日；发布日期：2018年6月7日

摘　要

为解决龙格库塔间断有限元方法(RKDG)求解流场耗时的问题，本文应用二维NACA0012翼型作为测试算

文章引用: 周星宇, 刘铁钢. 龙格库塔间断有限元方法求解二维欧拉方程的多 GPU 加速实现[J]. 流体动力学, 2018, 6(2): 15-22. DOI: 10.12677/ijfd.2018.62003

例，使用多GPU加速求解。将流程网格按照GPU个数进行剖分，每个GPU计算一个网格区域。各计算节点上设置核函数的线程数等于流场网格数，节点间的数据通信使用MPI (Message Passing Interface)。通信过程中采用CUDA流和MPI非阻塞操作以覆盖数据的传输和计算，减少通信代价。结果表明，与CPU串行程序相比，1个、2个、4个GPU上分别获得了33倍、59倍和108倍的加速比。

关键词

RKDG，多GPU，MPI

1. 引言

计算流体力学(CFD)在航空工业中扮演着越来越重要的角色。随着近些年的发展，CFD 能够模拟出许多现实的场景。但是对于复杂系统，需要采用高精度的算法。龙格库塔间断有限元算法是一种高精度的流场算法，具有能够捕捉间断，处理网格悬点等优点[1]。然而由于计算量大，往往需要花费很长的时间完成流场求解。

人们提出了许多方法来减少 CFD 求解时间，并行计算就是其中一种很好的方法。CPU 并行加速在工业应用上已经非常成熟，然而建立大规模 CPU 并行计算集群非常昂贵。因此 GPU 是一种更好的选择。近年来，GPU 在科学计算上表现出了重大的潜力。CUDA 是 Nvidia 公司 2007 年推出的 GPU 并行计算平台和编程模型[2]，实验表明，能够显著提高计算性能。

Klöckner A 等人在单个 GPU 上构建了高效的高阶间断有限元解算器，并在三维四面体网格上求解了麦斯威尔方程，与同一代的 CPU 相比达到了近 50 倍的加速比[3]；何晓峰等在单个 GPU 上，使用 RKDG 方法对二维欧拉方程进行了加速[4]，与 CPU 相比获得了 25 倍的加速比；Dawei Mu 等人在单个 GPU 上，使用间断有限元方法对三维弹性地震波方程进行了加速[5]，并与 CPU 并行程序进行了比较，在单精度下相比 1 个、2 个、4 个、8 个 CPU 并行程序获得的加速比分别为 12.9、6.8 和 3.6。本文在此基于多 GPU 对 RKDG 方法进行加速。论文结构安排如下，我们首先介绍了龙格库塔间断有限元方法和 CUDA 程序概述，然后介绍了单 GPU 以及多 GPU 上基于 CUDA 的间断有限元方法求解，最后分别在一个、两个和四个 GPU 上进行了求解，统计了耗时，并与 CPU 结果进行了比较。

2. 龙格库塔间断有限元求解

本文中，我们考虑二维双曲方程，格式如下：

$$\begin{cases} U_t + F(U)_x + G(U)_y = 0, \ (x,y) \in R^2 \\ U(x,y,0) = U_0(x,y). \end{cases}$$

其中

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix}, \quad F(U) = \begin{bmatrix} \rho U \\ \rho U^2 + P \\ \rho UV \\ \rho UH \end{bmatrix}, \quad G(U) = \begin{bmatrix} \rho V \\ \rho UV \\ \rho V^2 + P \\ \rho VH \end{bmatrix}$$

ρ，P，H 和 E 分别为密度，压强，单位体积的总焓和单位体积的总能。

选择解函数和试探函数空间如下：

$$U_h = V_h = V_h^N \left\{ p \in BV \cap L^1 : p\big|_K \in P^N(K) \right\}$$

K 表示三角形剖分单元，$P^N(K)$ 表示单元 K 上次数不高于 N 的多项式。在三角形单元 K 上取基函数 $\{v_0, v_1, \cdots, v_m\}$,则解函数 U_h 可表示为：

$$U_h = \sum_{p=0}^{m} U_K^{(p)}(t) \cdot v_p(x,y), \quad (x,y) \in K$$

本文处理三阶精度的间断有限元方法，选取三角形单元 K 上的一组正交基函数如下[6]：

$$v_0(x,y) = 1$$

$$v_1(x,y) = \frac{x - x_0}{\sqrt{|\Delta_0|}}$$

$$v_2(x,y) = a_{21}\frac{x - x_0}{\sqrt{|\Delta_0|}} + \frac{y - y_0}{\sqrt{|\Delta_0|}} + a_{22}$$

$$v_3(x,y) = \frac{(x - x_0)^2}{|\Delta_0|} + a_{31}\frac{x - x_0}{\sqrt{|\Delta_0|}} + a_{32}\frac{y - y_0}{\sqrt{|\Delta_0|}} + a_{33}$$

$$v_4(x,y) = a_{41}\frac{(x - x_0)^2}{|\Delta_0|} + \frac{(x - x_0)(y - y_0)}{|\Delta_0|} + a_{42}\frac{x - x_0}{\sqrt{|\Delta_0|}} + a_{43}\frac{y - y_0}{\sqrt{|\Delta_0|}} + a_{44}$$

$$v_5(x,y) = a_{51}\frac{(x - x_0)^2}{|\Delta_0|} + a_{52}\frac{(x - x_0)(y - y_0)}{|\Delta_0|} + \frac{(y - y_0)^2}{|\Delta_0|} + a_{53}\frac{x - x_0}{\sqrt{|\Delta_0|}} + a_{54}\frac{y - y_0}{\sqrt{|\Delta_0|}} + a_{55}$$

上式中正交基函数的系数 a_{ij} 由待定系数法求得。由于基函数在单元 K 上是正交的，所以单元质量矩阵 $M_{ij} = \sum_{l=1}^{6} \iint v_i v_j \mathrm{d}x\mathrm{d}y$ 为对角阵。计算过程中通量采用间断有限元方法离散，原双曲方程可以写成半离散形式如下：

$$\frac{\mathrm{d}}{\mathrm{d}t} U^h = M^{-1} L\left(U^h, t\right)$$

使用三步 Runge-Kutta 方法对上式进行迭代求解稳定解。

3. GPU 加速

3.1. Cuda 概述

CUDA 是显卡制造商 Nvidia 公司 2007 年推出的并行计算平台和编程模型，开发人员可以使用 c/c++ 语言来为 CUDA 架构编写程序。CUDA 提供 host-device 的编程模式以及非常多的接口函数和科学计算库，通过执行大量的线程来达到加速效果。CUDA 通过线程格来管理线程。线程格包含多个线程块，线程块包含多个线程。线程格和线程块可以是一维、二维或者三维结构。本文使用一维线程块结构，其线程类似于一个二维的像素数组，如图 1 所示。

线程索引号为：

```
tid = threadIdx.x + blockIdx.x * blockDim.x;
```

其中 threadIdx.x 表示线程块中的线程编号，blockIdx.x 表示线程格中线程块的编号，blockDim.x 表示每个

线程块0	线程0	线程1	线程2	线程3
线程块1	线程0	线程1	线程2	线程3
线程块2	线程0	线程1	线程2	线程3
线程块3	线程0	线程1	线程2	线程3

Figure 1. Two-dimensional organazition of blocks and thread
图 1. 线程块集合与线程集合的二维组织形式

线程块中的线程数。

3.2. 单 GPU 加速

单 GPU 加速时，由于 CUDA 支持大量的线程，因此选取 CUDA 线程数等于三角形网格个数。每个线程根据自己的全局索引号计算相应的网格单元即可。另外，还需要在 GPU 与 CPU 之间进行数据传输，所以程序的流程图如下：

```
int main(int argc, char *argc[])
{
        //变量申明
        //数据初始化
        //数据传输到 GPU
        do
            KernelFun<<<blocksize, threadsize,memsize,streamid>>>(args);
        while not convergence
        //数据传回 CPU
        //判断是否结束
        //输出结果
        return 0;
}
```

上式中 kernelFun 即为核函数，将其循环迭代到流场收敛为止。

我们通过核函数依次求解时间步长、计算守恒量、处理边界条件、计算体积分守恒残差、计算通量、计算单元各条边上的积分残差、最后执行 Runge-Kutta 时间推进。

3.3. 多 GPU 加速

在多 GPU 下，首先将网格进行区域划分，划分后一个 GPU 对应计算一块区域。划分后可以不对网格进行处理直接计算；也可以先处理网格，将本地网格进行重排序，并重新生成邻接信息等。以图 2 的网格为例进行说明，我们将它沿中间划分成两个区域。

若直接计算，需要在每个计算节点上储存所有网格单元信息，即需要储存全部 8 个单元。计算时，只处理属于分配给相应节点的网格，第一个计算节点只计算单元编号为 1、2、5、6 的流场变量值。这样没有打乱网格单元的邻接关系，操作方便。但是，需要储存所有的三角形单元信息并传输到 GPU 上，浪

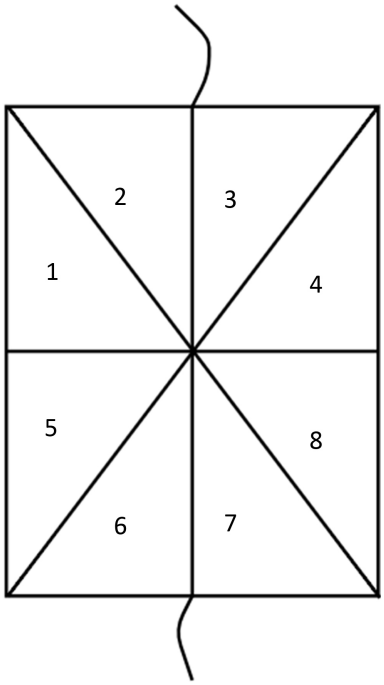

Figure 2. Triangle unstructured mesh
图 2. 三角形非结构网格示例

费内存。GPU 个数的增加并不能增大问题求解的规模，没有充分利用多个 GPU 计算的优势。同时，划分后每个区域的单元编号不再都是连续的，因此每个 GPU 上处理的单元在内存上也不是连续的。当 warp 请求访问位于不同地址的数据时，数据是非连续的，无法合并访问，大大影响访存的效率[7]。

若先将网格进行处理，在每个计算节点上，去除掉其他节点对应的网格，比如第一个计算节点上只需要储存编号为 1、2、5、6 的单元，所需的储存空间小了一倍。将分配到此节点的网格单元重新连续编号，即将 5、6 号单元重新编号为 3、4，这样每个节点上的网格单元都是连续存储的，并重新生成单元其他信息，再进行计算。缺点是处理网格会稍微麻烦，但是节约了内存，还提高了访存效率，总体上，效率会远高于前一种方式。因此本文采用这种方式。

计算过程中，需要在网格边界处进行信息交换，采用 MPI 进行通信[8]。MPI 是目前最流行的用于并行编程的消息传递接库标准。该模型底层是一组处理器，每个处理器有独立地址空间而且只能访问本地的指令和数据。各个处理器的不同进程间交互必须通过消息传递操作来实现。MPI 定义了点到点通信、集合通信、并行 I/O 等操作。

在信息交换时，我们先将数据从 GPU 拷贝到 CPU，再通过 MPI 在 CPU 之间交换，最后将数据拷贝回 GPU。由于本文处理的是定常问题，因此不需要每一个时间步步都进行交换。同时 MPI 采用非阻塞通信，上一个时间步发送和接收消息，到下一个时间步才使用 MPI_WaitAll 来同步信息的交换。由于 cuda 核函数也是异步进行的，因此这样可以较好的覆盖 CUDA 的计算和 MPI 消息传递，提高并行效率。

4. 数值实验

求解 naca0012 翼型在来流马赫数 0.4，攻角 5˚下流场收敛时的密度。网格规模为 37,360 个三角形单元，如图 3 所示。

分别在 CPU 以及不同个数 GPU 上进行了计算，CPU 使用 Intel Xeon X5690 (主频 3.47 Hz)，GPU 使用 Nvidia Tesla k20 GPGPU，含有 2496 个流处理器。GPU 个数等于网格划分后的区域个数，两个 GPU

对应的网格划分结果如图 4 所示，四个 GPU 对应划分后的四个区域如图 5 所示。计算耗时统计如表 1 所示。结果表明，GPU 个数并不影响最终的流场解，流场解的密度等值线如图 6 所示。

结果显示，单个 GPU 对比单个 CPU 获得了 33 倍的加速比。采用不同 GPU 个数相对 CPU 加速比如图 7 所示。发现随着节点个数的增加，加速比逐渐增加，能够很好的处理大规模的计算。但是由于多个

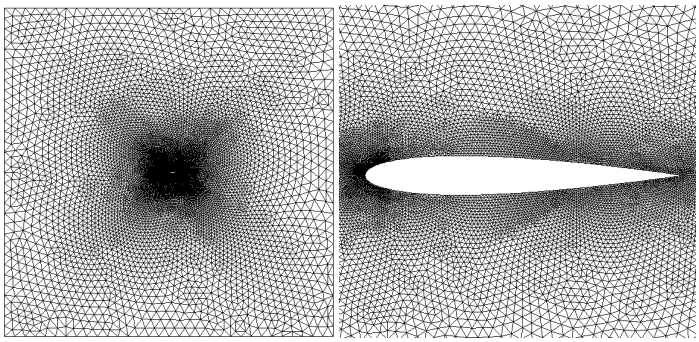

Figure 3. Triangle unstructured mesh
图 3. 三角形非结构网格

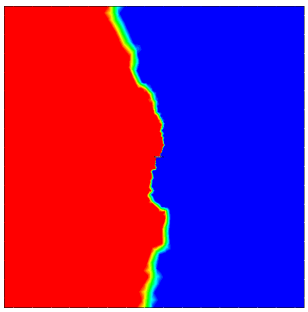

Figure 4. Mesh divided into two parts
图 4. 网格划分为两个区域

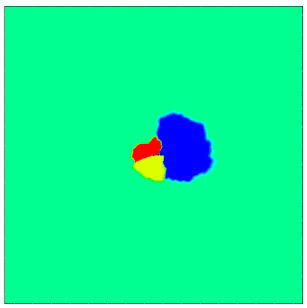

Figure 5. Mesh divided into four parts
图 5. 网格划分为四个区域

Table 1. System resulting data of standard experiment
表 1. CPU 及 GPU 上运行时间统计

处理器	Intel Xeon X5690	Nvidia Tesla k20		
		1 个	2 个	4 个
耗时(s)	85,895	2570	1465	798

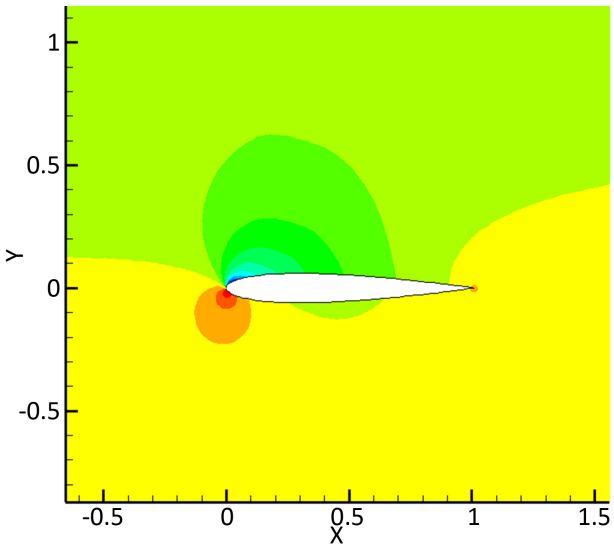

Figure 6. Density contour map of solution
图 6. 解的密度等值线图

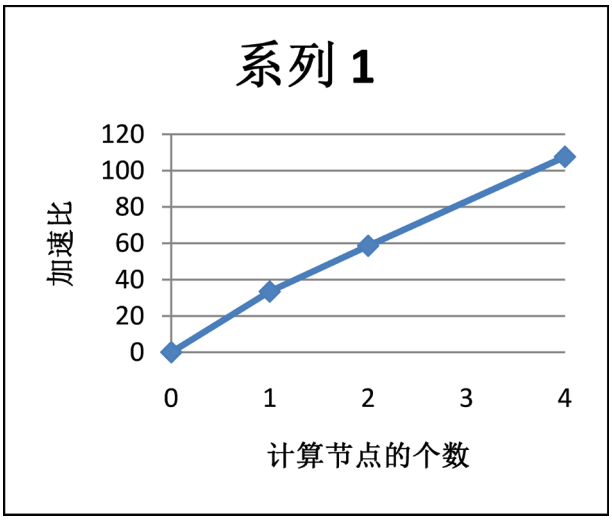

Figure 7. Speedup ratio of different numbers of GPU relative CPU
图 7. 不同 GPU 个数相对 CPU 加速比

节点计算会需要节点之间的协调和通信等耗时操作，因此加速比并没有和节点个数成正比关系。

5. 结论

本文基于 CUDA 采用多 GPU 对龙格库塔间断有限元方法进行了加速。分别测试了一个、两个和四个 GPU，并取得了较好的加速效果。充分利用了多个 GPU 的特性，可以克服规模增大后单个 GPU 显存不足的情况。同时，本文针对定常问题，采用数据交换的下一个时间步同步并处理交换的信息，减少了通信的代价，增加了并行效率。

参考文献

[1] Qiu, J.X., Khoo, B.C. and Shu, C.-W. (2006) A Numerical Study for the Performance of the Runge-Kutta Discontinuous Galerkin Method Based on Different Numerical Fluxes. *Journal of Computational Physics*, **212**, 540-565.

https://doi.org/10.1016/j.jcp.2005.07.011

[2] Sanders, J. and Kandrot, E. (2010) CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional, .

[3] Klöckner, A., Warburton, T., Bridge, J., *et al*. (2009) Nodal Discontinuous Galerkin Methods on Graphics Processors. *Journal of Computational Physics*, **228**, 7863-7882. https://doi.org/10.1016/j.jcp.2009.06.041

[4] 何晓峰, 程剑, 刘铁刚. 二维非结构网格上 RKDG 算法的 CUDA 解法器[C]//北京应用物理与计算数学研究所. 第十六届全国流体力学数值方法研讨会论文集: 2013 年卷. 北京: CNKI, 2013.

[5] Mu, D.W., Chen, P. and Wang, L.Q. (2013) Accelerating the Discontinuous Galerkin Method for Seismic Wave Propagation Simulations Using the Graphic Processing Unit (GPU)—Single-GPU Implementation. *Computers and Geosciences*, **51**, 282-292. https://doi.org/10.1016/j.cageo.2012.07.017

[6] Cockburn, B. and Shu, C.W. (1991) The Runge-Kutta Local Projection $ P^1$-Discontinuous-Galerkin Finite Element Method for Scalar Conservation Laws. *ESAIM*: *Mathematical Modelling and Numerical Analysis*, **25**, 337-361. https://doi.org/10.1051/m2an/1991250303371

[7] NVIDIA: NVIDIA CUDA C Programming Guide, NVIDIA Corporation, May, 2011.

[8] 都志辉, 李三立, 审阅, 等. 高性能计算之并行编程技术——MPI 并行程序设计[M]. 北京: 清华大学出版社, 2001.