计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201200129

基于 DGX-2 的湍流燃烧问题优化研究

文敏华[1]　汪申鹏[1]　韦建文[1]　李林颖[2]　张　斌[2]　林新华[1]
1 上海交通大学高性能计算中心　上海 200240
2 上海交通大学航空航天学院　上海 200240
(james@sjtu.edu.cn)

摘　要　湍流燃烧问题的数值模拟是航空发动机设计的关键工具。由于需要使用高精度计算模型求解 NS 方程,湍流燃烧的数值模拟需要庞大的计算量,而物理化学模型的引入则导致流场极为复杂,使得计算域内的负载平衡问题成为大规模并行计算的瓶颈。为此文中将湍流燃烧的数值模拟方法在单台具有强大计算能力的服务器——DGX-2 上进行移植和优化,设计了通量计算的线程分配方式,并以 Roofline 模型为工具分析指导了实际的优化方向。此外,还设计了高效的数据通信方式,并结合 DGX-2 的高速互联实现了湍流燃烧数值模拟方法的多 GPU 并行版本。实验结果表明,相较于双路 Intel Xeon 6248 CPU 40 核心的并行版本,迭代过程的计算部分在单块 V100 上获得了 8.1 倍的性能提升,在 DGX-2 共 16 块 V100 上达到了 66.1 倍的加速,优于 CPU 并行版本所能达到的最高性能。

关键词:湍流燃烧;NS 方程;DGX-2;CUDA
中图法分类号　TP311.1

DGX-2 Based Optimization of Application for Turbulent Combustion

WEN Min-hua[1],WANG Shen-peng[1],WEI Jian-wen[1],LI Lin-ying[2],ZHANG Bin[2] and LIN Xin-hua[1]
1 Center for High Performance Computing,Shanghai Jiao Tong University,Shanghai 200240,China
2 School of Aeronautics and Astronautics,Shanghai Jiao Tong University,Shanghai 200240,China

Abstract　Numerical simulation of turbulent combustion is a key tool for aeroengine design. Due to the need of high-precision model to Navier-Stokes equation,numerical simulation of turbulent combustion requires huge amount of calculations,and the physicochemical models causes the flow field to be extremely complicated,making the load balancing a bottleneck for large-scale parallelization. We port and optimize the numerical simulation method of turbulent combustion on a powerful computing server,DGX-2. We design the threading method of flux calculation and use Roofline model to guide the optimization. In addition,we design an efficient communication method and propose a multi-GPU parallel method for turbulent combustion based on high-speed interconnection of DGX-2. The results show that the performance of a single V100 GPU is 8.1x higher than that on dual-socket Intel Xeon 6248 CPU node with 40 cores. And the multi-GPU version on DGX-2 with 16 V100 GPUs achieves 66.1x speedup,which is higher than the best performance on CPU cluster.

Keywords　Turbulent combustion,Navier-Stokes equation,DGX-2,CUDA

1　引言

湍流燃烧的数值模拟是航空发动机燃烧室内部燃烧流动过程的重要研究手段,在燃烧室性能预估和设计定型中发挥了关键作用[1]。由于湍流燃烧过程涉及复杂的流体运动和动力学过程,相比于传统的实验方法,数值计算能够以更短的时间与更低的经济成本模拟更复杂的流场和条件,提供更丰富的流场数据。但是,湍流燃烧问题需要使用高精度模型,如直接数值模拟(Direct Numerical Simulation,DNS)[2]、大涡模拟(Large-eddy Simulation,LES)[3]对纳维-斯托克斯方程(Navier-Stokes Equations,NS 方程)数值求解以进行湍流模拟,带来了庞大的计算量。而复杂的物理化学模型的引入,导致流场极为复杂,使计算负载在整个计算域分布非常不平衡,给大规模并行计算带来了严重挑战,尤其对于传统的 CPU 集群,通常涉及上万核心的进程级并行计算,很难实现针对不同算例的高效率并行扩展。

近年来,大量研究人员利用 GPU 的浮点计算优势进行 NS 方程的求解,早在 2005 年 Krüger 等[4]就使用 GPU 实现了线性方程组求解,并扩展到二维不可压 NS 方程求解;Goodnight 等[5]在 GPU 上实现了基于多重网格的边值求解,获得了 2 倍左右的加速。这些早期工作都是基于图形学 API 映射,对编程要求很高,难以推广。直至 2007 年,随着统一计

到稿日期:2020-12-14　返修日期:2021-04-09　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划(2016YFB0201800)
This work was supported by the National Key Research and Development Program of China(2016YFB0201800).
通信作者:林新华(james@sjtu.edu.cn)

算架构(Compute Unified Device Architecture,CUDA)发布,GPU 计算的编程难度大幅降低,GPU 在 NS 方程求解的应用也迎来快速发展。Aissa 等[6]对基于结构网格的隐式求解和显式求解进行 GPU 优化,其中隐式求解性能获得 6 倍加速,而显式求解性能提升了 136 倍。Phillips 等[7]在结构网格上采用 GPU 求解二维 Euler 方程,在 8 节点 16 块 GPU 上相比 CPU 串行程序获得了 496 倍性能提升。Jacobsen 等[8]在 64 个节点共 128 块 GPU 上相比 8 核 CPU 获得了 130 倍加速。Bolz 等[9]对稀疏矩阵求解的多网格方法进行了 GPU 实现。Corrigan 等[10]对基于非结构网格的三维不可压流动进行了 GPU 移植,相比串行程序获得了 33 倍加速。Nguyen 等[11]对多网格的 RANS 求解进行 GPU 优化,在 1 块 GPU 上对比 CPU 16 核心,获得了 2.4 倍加速。Oyarzun 等[12]对 LES 求解高雷诺数问题进行 GPU 大规模并行优化,在 512 个 GPU 节点下其计算性能比 6144 CPU 核心提升 4.2 倍。

随着求解问题规模的增大,对计算资源的需求增加,GPU 集群的并行扩展也成为难题。DGX-2[1)]作为单机双精度浮点性能超过 120TFLOPS 的强大计算系统,可以有效解决湍流燃烧模拟的大规模并行问题。DGX-2 中采用了 NVSwitch[13]技术,拥有 18 个 51.5GB/s 的 NVLink 端口,整合 16 块完全互联的 Tesla V100 GPU,整个系统拥有共计 40960 个双精度 CUDA 核心和 512GB 的 HBM2 高带宽显存。

为了将 DGX-2 的计算能力以及高速的通信速率应用于湍流燃烧问题,本文进行了如下优化研究:1)使用 CUDA 对采用两阶半隐式龙格-库塔方法的 NS 方程求解器进行了 GPU 的移植和优化,在 GPU 上实现了求解的主要步骤,如刚性/非刚性通量的计算、燃烧化学反应的计算等;2)使用 Roofline 作为工具分析并指导 GPU 优化方向;3)采用 MPI_Alltoall,MPISend/Recv,CUDA-Aware MPI 3 种方式实现了多 GPU 之间的并行通信并进行了性能评估。总而言之,本文有以下 3 点贡献:

(1)使用 CUDA 框架首次在 GPU 上实现了湍流燃烧问题的求解过程;

(2)利用 Roofline[14]指导 GPU 的优化方向,最终在单块 V100 上,比 Intel Xeon Gold 6248 40 核心节点性能提高了 8.1 倍;

(3)在 DGX-2 平台上实现了对湍流燃烧 NS 方程的多 GPU 并行求解,利用 DGX-2 的高速互联技术,得到了 66.1 倍的加速效果,优于 CPU 所能达到的最高性能。

本文第 2 节介绍了相关工作;第 3 节介绍了背景,主要包含纳维-斯托克斯方程及其求解方法;第 4 节主要介绍了整个程序在 GPU 上的移植优化及并行加速的实现;第 5 节结合实验结果分析了本文工作的优点和不足;最后总结全文并展望未来。

2 相关工作

在求解 NS 方程时,龙格-库塔方法[15]由于精度高而被广泛使用。然而,对于刚性项的计算,龙格-库塔显式求解不足以满足精度和速度的要求。基于此,Zhong[16]提出了隐式求解刚性项的方法,这种方法可以将刚性项的计算由非线性变为线性,在提高计算效率的同时加强了稳定性,而对于非刚性项的计算,其采取显式处理。Zhong 提出的半隐式龙格-库塔方法,可以精确地描述流体力学问题中刚性项和非刚性项的计算,并且有着高阶精度和稳定性强的特点。

对于湍流燃烧问题,Wu[1]对比了多种湍流燃烧模型在 CFD 软件上的计算结果。他计算了燃烧室的内部流场、压力场、温度场和浓度场,将采用不同燃烧模型计算得到的出口温度径向和周向分布系数与已知的实验数据进行比较,并验证了计算结果的合理性。其中湍涡耗散模型(Eddy Dissipation Model,EDM)的出口温度和分布系数均在规定值的范围之内,且最为接近实测值。

Thibault 等[17]实现了 NS 方程求解的 CPU 和 GPU 并行计算。他们对计算区域进行了划分,将其平均分配给各个进程(显卡)进行并行求解。他们在 GTX280 及 Intel/AMDCPU 上分别实现了该求解方法,结果表明,相比 CPU 的串行版本,该方法最终取得了 4 卡 100 倍的加速效果,计算峰值达到了 1TFLOPs,其 GPU 版本 4 卡的并行效率约为 75%。

与已有的工作相比,本文首次尝试使用半隐式龙格-库塔方法在类似 DGX-2 的单机密集 GPU 的计算系统上对湍流燃烧问题进行优化研究,针对 NVSwitch 的优势进行了通信上的优化,并以 Roofline 模型指导该方法在 GPU 上进一步的性能提升。

3 背景介绍

3.1 纳维-斯托克斯方程

纳维-斯托克斯方程是在宏观连续性介质的假设下,对质量、动量、能量守恒的数学描述。其一般的形式为:

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t}=\boldsymbol{f}(\boldsymbol{u})+\boldsymbol{g}(\boldsymbol{u}) \tag{1}$$

其中,$\boldsymbol{f}(\boldsymbol{u})$和$\boldsymbol{g}(\boldsymbol{u})$分别是非刚性项和刚性项,结合边界条件可对该方程进行求解。

3.2 基于半隐式龙格-库塔方法的 NS 方程求解

龙格-库塔(Runge-Kutta)方法是处理 NS 方程中间阶段以实现高阶精度的一种方法。一般的 r 阶龙格-库塔方法通过同时显式地处理 $\boldsymbol{f}$ 和 $\boldsymbol{g}$ 得到积分方程:

$$\boldsymbol{u}^{n+1}=\boldsymbol{u}^n+\sum_{j=1}^{r}\omega_j\boldsymbol{k}_j \tag{2}$$

$$\boldsymbol{k}_i=h\{\boldsymbol{f}(\boldsymbol{u}^n+\sum_{j=1}^{i-1}b_{ij}\boldsymbol{k}_j)+\boldsymbol{g}(\boldsymbol{u}^n+\sum_{j=1}^{i-1}c_{ij}\boldsymbol{k}_j+a_i\boldsymbol{k}_i)\} \tag{3}$$

其中,$\boldsymbol{u}^n$和$\boldsymbol{u}^{n+1}$是位于t_n和t_{n+1}时刻的系统状态。a_i,b_{ij},c_{ij},ω_j均为与精度和稳定性有关的参数。可以通过网格划分和限制时间步的方法对 NS 方程进行迭代求解。其中每一步都会得到当前时间步新的近似解,通过不停地迭代可以达到最终的稳态,从而得到最终的稳态近似解。

这种求解方法对刚性项 $\boldsymbol{g}$ 的计算是非线性的,其实际的计算效率较差。因此,对刚性项 $\boldsymbol{g}$ 进行隐式地处理,即可得到半隐式的龙格-库塔方法:

$$[\boldsymbol{I}-ha_i\boldsymbol{J}(\boldsymbol{u}^n+\sum_{j=1}^{i-1}d_{ij}\boldsymbol{k}_j)]\boldsymbol{k}_i =h\{\boldsymbol{f}(\boldsymbol{u}^n+\sum_{j=1}^{i-1}b_{ij}\boldsymbol{k}_j)+\boldsymbol{g}(\boldsymbol{u}^n+\sum_{j=1}^{i-1}c_{ij}\boldsymbol{k}_j)\} \tag{4}$$

1) https://www.nvidia.com/en-us/data-center/dgx-2/

其中，$\boldsymbol{J}$ 是刚性项的雅可比矩阵。相对于式(3)，式(4)对于 $\boldsymbol{g}$ 的计算有着线性求解的能力，其实际的计算效率较高。

为了进一步简化计算，在式(4)中，令 $d_{ij}=c_{ij}$，固定 a_i 和 b_{ij}，令阶数为 2。这种简化会有一定的精度损失，但是可以通过一定程度的加密网格进行弥补。因此，本文采取的最终迭代形式为：

$$[\boldsymbol{I}-ha_1\boldsymbol{J}(\boldsymbol{u}^n)]\boldsymbol{k}_1=h\{\boldsymbol{f}(\boldsymbol{u}^n)+\boldsymbol{g}(\boldsymbol{u}^n)\} \tag{5}$$

$$[\boldsymbol{I}-ha_2\boldsymbol{J}(\boldsymbol{u}^n)]\boldsymbol{k}_2=h\{\boldsymbol{f}(\boldsymbol{u}^n+b_{21}\boldsymbol{k}_1)+\boldsymbol{g}(\boldsymbol{u}^n+c_{21}\boldsymbol{k}_1)\} \tag{6}$$

$$\boldsymbol{u}^{n+1}=\boldsymbol{u}^n+\omega_1\boldsymbol{k}_1+\omega_2\boldsymbol{k}_2 \tag{7}$$

3.3 湍流燃烧

湍流是流体的一种流动状态。当流体流速较小时流体相互间会分层，且互不干扰，这种情况称为层流状态；而当流速增加，由于层间的相互作用力，流线会开始出现波浪状的摆动，这种情况称为过渡流；当流速继续增加到很大时，流线难以分辨，流场中会出现一个个不稳定的小漩涡，这种情况称为湍流。流体中可能存在化学反应，会对流体的状态造成复杂的影响。其中物质之间存在的化学反应平衡为：

$$\sum_{k=1}^{K}v'_{ki}x_k\Leftrightarrow\sum_{k=1}^{K}v''_{ki}x_k\ (i=1,\cdots,I) \tag{8}$$

其中，I 是反应的种类，K 是粒子的种类。v'_{ki} 和 v''_{ki} 是第 i 个反应中，物质 x_k 正逆向反应的系数。从这些化学反应的反应平衡以及平衡常数可以得到其反应速率为：

$$\omega_k=\sum_{i=1}^{I}v_{ki}q_i\ (k=1,\cdots,K) \tag{9}$$

其中，$v_{ki}=v''_{ki}-v'_{ki}$，q_i 是通过反应平衡常数以及物质实时浓度计算得到的参数。

同时流体的温度、压力等也会影响其反应速率，进而影响能量的变化和流体的状态。并且包含燃烧的流体会产生剧烈的能量波动，其实际的状态必须用湍流来描述。因此，包含湍流燃烧的流体力学问题有更多更复杂的计算量[10]。

4 DGX-2 集群的移植优化

本文使用二阶半隐式龙格-库塔方法对 NS 方程进行数值模拟，并在计算中加入了湍流燃烧的计算模块。

4.1 NS 方程的求解过程在 GPU 上的实现

NS 方程在 GPU 上的求解流程如图 1 所示。流程的关键步骤如下。

(1)读入网格文件、边界条件以及参数文件，获取所有网格内的初始状态，包括位置、速度等；

(2)根据给定的并行数对计算域进行划分，给每个进程分配相应的计算域并把数据传输给对应进程；

(3)每个进程对各自计算域内的网格初始化，计算网格内部的温度、压强等信息；

(4)进行 CPU 内存和 GPU 显存之间的信息传输，主要数据为每个网格的状态数组(速度、温度、压强等)；

(5)计算刚性通量、非刚性通量以及化学反应数据，并根据式(5)计算 $\boldsymbol{k}_1$；

(6)刷新网格状态信息，相邻计算域间需要通信；

(7)重新计算刚性通量、非刚性通量以及化学反应数据，并根据式(6)计算 $\boldsymbol{k}_2$；

(8)根据式(7)更新网格状态，判断其是否满足稳定条件，若满足稳定条件则输出结果，若不满足则重复步骤(5)—(7)。

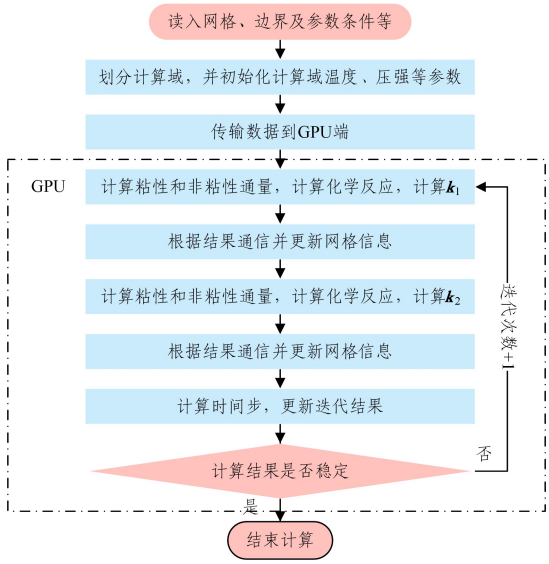


图 1 加法半隐式龙格-库塔方法求解 NS 方程的流程

Fig. 1 Process of additive semi-implicit Runge-Kutta method solving Navier-Stokes eqution

由于整个迭代计算过程都在 GPU 上实现，因此可以将计算数据从 CPU 内存一次性传入到 GPU 显存，之后的计算均在 GPU 内实现，这样可以减少 CPU 和 GPU 间的数据传输，大幅提高计算效率。

4.1.1 线程分配策略

在 GPU 上容易实现的迭代部分计算函数都不存在网格单元间的依赖关系，因此可以直接分配给线程，只需要让每个线程对应一个网格单元，再根据上一步迭代的结果完成相应单元内部的计算，进而完成整个迭代过程的计算。

对于刚性通量和非刚性通量的计算，其计算域是网格对应的面，存在相邻网格共用同一个面的情况，网格单元机间存在一定的数据依赖。龙格-库塔求解器使用的网格是六面体结构网格，面的通量的计算是根据其相邻的两个网格的信息来进行的。因此，本文提出了一种策略来应对这种计算模式(详见 4.2 节)。

4.1.2 数据存储结构

本文代码采用的是 CUDA Fortran，使用 PGI 编译器进行编译。由于本文采取线程与网格单元一一对应的方法，且计算中需要对不同的分子种类进行统计，因此代码中大部分的数组都包含网格单元和分子种类两个维度，以二维或者三维的格式呈现。GPU 函数以网格单元进行 CUDA 线程分配，因此计算过程中相邻线程访问的是数组中不同网格的相同部分的数据。为了有效利用 GPU 的合并访存特性，对数据采取如图 2 所示的存储格式。

P1	P2	P3	P4
P5	P6	P7	P8
…			
T1	T2	T3	T4
…			

图 2 合并访问的数据格式

Fig. 2 Data structure for coalesced access

在计算时，相邻线程会同时访问网格单元的 P 数据，而在这种存储格式下，该数据的存储位置相邻，即可实现高效的合并访问。

4.2 通量计算的线程分配策略

通量的计算是以面为单位的。然而，其他函数的线程分配方式都是以网格单元为单位进行的。重新让每个面有自己的独立存储格式会使计算产生不连贯性。因此，要以网格单元为基础单位，使用合理的策略来优化面通量的计算。

4.2.1 通量单元的存储格式

为了与其余的计算统一，面通量的数据存储格式和其余数据保持一致，其 3 个维度分别为：网格单元序号、分子种类序号、面序号(本文使用的是结构六面体网格，因此面序号均为 1—6)。由于相邻网格对应的面是同一个面，其通量结果也应保持一致(这里可以由面到网格的索引维持)。

4.2.2 通量单元的计算策略

本文依旧采用一个线程对应一个网格单元的线程分配模式，且每个线程需要计算其 6 个面的通量。然而，与相邻网格对应的线程可能会计算同一个面的通量，这会导致数据的竞写，造成计算错误。

为了避免这种情况的发生，本文采取了预处理的方法，明确给出了每个面应该被计算的单元序号，确保每个面只被计算一次。即维持一个新的数据结构，其包含的信息是某个网格单元需不需要计算其某个面的通量。

如图 3 所示，我们需要维持一个用于判断本网格单元 n_i 是否计算其第 f_i 个面的通量的数据结构 $is\_face\_calculate(n_i, f_i)$。如果需要计算则该线程需要计算该通量，并在计算之后判断是否需要更新信息到相邻网格的相同面(可能存在边界节点，其面并不与本计算域内网格相邻)，此信息由 $ncell(n_i, f_i)$ 和 $nface(n_i, f_i)$ 维持。这样，可以保证计算域内每个面只被计算一次。

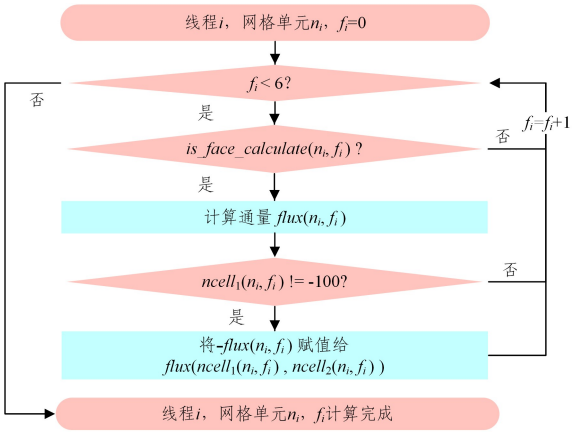


图 3 网格单元计算面通量的流程图

Fig. 3 Flow chart for calculating face flux with a cell

4.2.3 策略分析

本节提出的分配策略可以很好地进行面的通量计算。然而，根据 GPU warp SIMD 的特性，有些线程可能需要等待，但是本文采取的是随机存储网格单元的方式，相邻线程一般存储的是不相邻的网格，因此这种线程资源的浪费较少。

4.3 Roofline 分析及优化

Roofline 是一个测试性能指标的工具，对于高性能计算的平台，一般有算力和带宽两个指标。Roofline 基此于提出了更加直观的指标——计算强度峰值，其为算力和带宽的商，表示每单位内存交换最多可以进行多少次计算。

对具体的应用程序，Roofline 指标包括计算量、访存量和计算强度。计算量指该程序进行计算的浮点计算数，访存量指该程序进行访存的次数，计算强度为计算量与访存量的商。而程序的计算量和实际运行时间的商就是该程序的实际计算速度。有了计算强度和实际的计算速度，我们就可以测算目前程序所处的位置是带宽瓶颈还是计算瓶颈，从而指导程序的优化方向。

4.3.1 V100 GPU 上 Roofline 的分析结果

我们构建了移植后的 Roofline 模型，如图 4 所示。

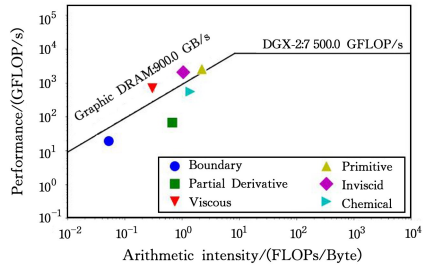


图 4 GPU 上热点函数的 Roofline 模型

Fig. 4 Roofline model for hotspot skernels on GPU

如图 4 所示，迭代过程中的主要计算函数的计算强度较低，因此目前程序的瓶颈在于数据的存取，且未能有效利用 V100 的带宽。因此我们认为，程序的主要问题在于：1)合并访问做的不够，存在计算单元等待数据传输的问题；2)没有利用好 GPU 的高效存储单元，如 Shared Memory，Register 等。

4.3.2 Roofline 指导下的优化策略

一方面，对于 GPU 显存的带宽问题，我们对不符合合并访存的计算进行了顺序调整，使其尽量符合合并访存的特性。

另一方面，针对较难进行合并访存的计算函数(如通量计算)，我们利用 Shared Memory 和 Register，对其算顺序进行了一定的调整。此外，Shared Memory 和 Register 的使用也会对 GPU block 的并行度产生影响。因此，本文在充分利用 GPU Shared Memory 及 Register 资源的基础上进行了优化。

4.4 通信优化

对于通信部分，本文使用代码的 CPU 版本的通信策略，其将需要通信的数据全都整合起来，以 MPI_Alltoall 的形式一次性发送，即进程需要与其余所有的进程进行通信。这种通信模式会引入更多开销，比如某些进程所负责的计算域并不相连时，特别地对于大规模扩展，其调用的开销更大。因此，选取合适的通信模式也有利于程序整体性能的提升。

4.4.1 MPI_Alltoall—>Send/Recv 的通信优化

首先，本文将 Alltoall 通信替换成没有额外开销的 Isend/Irecv。

迭代过程中不同进程负责的计算域是固定的，也就是说其相邻计算域以及通信量也是固定的。因此，我们对各进程进行筛选，除去通信为 0 的进程，维持一个只有通信不为 0 的相邻进程序号的数组以及对应的通信量。

在传输数据时，我们使用异步 Isend/Irecv 的形式对相邻

进程一一传输数据（为了避免死锁）。最后使用同步语句 MPI_Waitall 确认数据传输完成。

4.4.2 CPU Bus－＞GPU Bus 的通信优化

我们发现，在通信 API 优化之后，通信处仍有很高的延迟。这是因为维持网格单元间数据通信需要在 CPU 和 GPU 间传输数据，即 GPU－＞CPU－＞MPI－＞CPU－＞GPU 的通信模式。在 GPU 计算单元的高效率下，这种传输效率十分低下。

因此，本文利用 DGX-2 的 CUDA-Aware MPI 来简化这种通信传输。其可以允许 GPU 显存之间直接传输数据，而不需要通过 CPU 总线。因此，通信可以简化成 GPU－＞CUDA-Aware MPI－＞GPU 的通信模式，使性能得到进一步的提升。除此之外，CUDA-Aware MPI 本身的传输带宽高于 CPU，因此可以得到更好的通信性能。

5 实验结果及分析

5.1 实验环境

本文实验所采用硬件配置如下：CPU 为 Intel Xeon Gold 6248，架构为 Cascade Lake，主频为 2.5 GHz，核心数为 20，双精度浮点性能为 1.6 TFLOPS，节点内存大小为 192 GB；GPU 版本测试使用的是 NVIDIA 的 DGX-2 平台，该平台采用的 GPU 为 V100，5 120 个单精度 CUDA 核心，2 560 个双精度 CUDA 核心，双精度浮点性能达到 7.5 TFLOPS，显存带宽可达 900 GB/s；DGX-2 内部通过 NVSwtich 桥接器支持 16 个全互联的 V100 GPU 卡。

本文用于测试的算例为二维同轴燃料-空气射流模拟，网格大小为 825 000，由 30％氢气和 70％氮气组成的燃料从中央喷出，燃料来流速度为 1 000 m/s，温度为 390 K，压力为 1 atm，环境温度为 1 400 K，气压为 1 atm，来流速度为 2 000 m/s，涉及 10 种组分及 19 种化学反应。

5.2 优化后的 Roofline 性能分析

经过 4.3.2 节的优化之后，我们得到了优化后的 Roofline 模型。

从图 5 可以看到，优化后的计算函数的实际性能均向带宽增加的方向移动，甚至有些已经超过了显存的带宽限制。这是因为我们利用了 Shared Memory 和 Register 等计算资源，而这些资源的带宽高于显存。

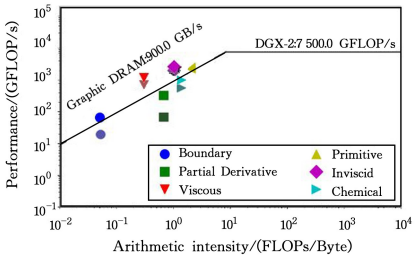


图 5 优化前后的 GPU 热点函数 Roofline 模型

Fig. 5 Roofline model for optimized kernels on GPU

计算函数的性能在优化之后一般都获得了 2～3 倍的提升。

5.3 MPI 通信优化结果

根据 4.4 节的优化策略，我们对 DGX-2（16 块 V100）的 GPU 并行版本的通信进行了优化，并给出了如图 6 所示的实验结果。

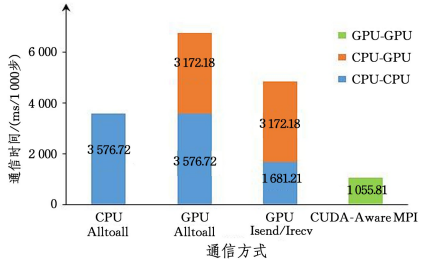


图 6 通信时间

Fig. 6 Communication time

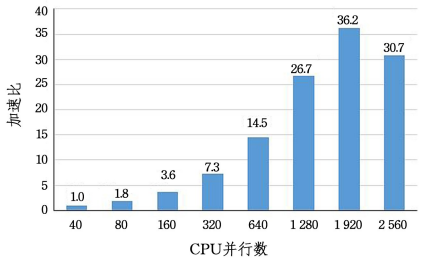
如图 6 所示，我们一方面将 MPI_Alltoall 通信转化为 MPI Isend/Irecv 通信，减少了通信开销；另一方面，使用 CUDA-Aware MPI 不仅提高了通信的带宽，也减少了 CPU-GPU 之间的数据传输。

结合图 6 中的数据可以得到，迭代过程的通信时间减少了约 86％，通信成本大幅降低。

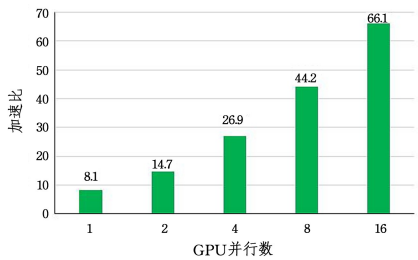
5.4 性能测试及对比

本文对比的 CPU 性能以一个双路 Intel Xeon 6248 节点（40 核心）为基准，另外，由于代码本身位于迭代循环之外的部分耗时较少，因此本文仅对比迭代过程中的计算耗时，这里选取的是每步迭代的计算时间，不包括初始化以及对数据 I/O 的耗时。

本节对比了 CPU 和 GPU 各自并行版的扩展性。加速比均以 Intel Xeon 6248 40 核心版本的计算时间为基准，具体情况如图 7 所示。



(a)



(b)

图 7 CPU 和 GPU 上的并行性能对比

Fig. 7 Parallel efficiency on CPU and GPU

从图 7(a)可以看到，CPU 上的扩展性在 1 920 核心处达到最大值，在 2560 核心处开始下降。这是因为随着并行数的增加，计算占比下降，通信占比上升，于 2 560 核心处开始逆转。CPU 版本在 1 920 核心处达到最高性能为 36.2 倍。

图 7(b)描述了 GPU 并行版本的扩展性。在做了通信优化之后，DGX-2 16 核心版本的加速比达到 66.1 倍，优于

CPU 所能达到的最好性能，这也说明在 GPU 上的并行计算可以超过 CPU 的极限。

然而 GPU 并行版本在 16 核心处的扩展性仅为 51%，一方面是因为 GPU 本身带来的巨大性能提升让计算时间大幅减少，而通信带来的额外开销会有上升；另一方面，实验表明该算例并未完全跑满 16 核心的全部计算资源，因此也有一定的性能损失，这部分潜能在面对更大的算例时会体现出来。

结束语 本文实现了二阶半隐式龙格-库塔方法对湍流燃烧问题在 GPU 上的数值模拟，并在 DGX-2 上进行了扩展性优化。本文提出了通量计算的线程分配策略，并使用 Roofline 对程序进行了 kernel 级别的分析和指导。在单块 V100 上，网格大小为 825000 的算例，相较于 Intel Xeon 6248 40 核心，其得到了 8.1 倍的性能提升。

同时，本文利用了 DGX-2 的 GPU 高速互联的特性，采用消减通信以及 CUDA-Aware MPI 的方法对 GPU 的并行版本的通信进行了优化，最终在 DGX-2 上实现了 66.1 倍的性能提升，优于 CPU 并行所能达到的极限性能。

我们利用 DGX-2 的计算能力给出了便捷的优化手段。很多传统的 CPU 大规模并行问题都面临着高并行度时扩展性下降、负载难以均衡等额外难题。而对于 DGX-2，我们仅需要控制 16 块 GPU，对 16 个进程的负载均衡问题进行处理。DGX-2 提供的强大算力和高速互联为我们解决大规模并行问题提供了新的解决方案。

目前本文仅对计算函数进行了初步的优化，从 Roofline 结果来看，其达到了 GPU 显存带宽的瓶颈，但距 Shared Memory 和 Register 的带宽仍有差距。后续我们将继续 Shared Memory 和 Register 的优化，以获得更好的性能。目前本文使用 Roofline 模型针对 kernel 进行分析，未来会构建更加精细的类似于循环级别的 Roofline 模型。另外，通量计算的并行策略仍存在一定的资源浪费，未来会将网格单元级的线程分配策略展开为循环级的分配策略，进一步提高 GPU 利用率。同时，我们将使用更多的计算模型以及不同网格的算例对目前程序的性能和瓶颈进行分析。

参 考 文 献

[1] WU C. Study on applicability of turbulent combustion model in the numerical calculation of combustor[D]. Shenyang: Shenyang Institute of Aeronautical Engineering, 2009.

[2] MOIN P, MAHESH K. Direct numerical simulation: a tool in turbulence research[J]. Annual Review of Fluid Mechanics, 1998, 30(1): 539-578.

[3] PITSCH H. Large-eddy simulation of turbulent combustion[J]. Annu. Rev. Fluid Mech., 2006, 38: 453-482.

[4] KRÜGER J, WESTERMANN R. Linear algebra operators for GPU implementation of numerical algorithms[M]// ACM SIGGRAPH 2005 Courses. 2005: 234-242.

[5] GOODNIGHT N, WOOLLEY C, LEWIN G, et al. A multigrid solver for boundary value problems using programmable graphics hardware[M]// ACM SIGGRAPH 2005 Courses. 2005: 193-203.

[6] AISSA M, VERSTRAETE T, VUIK C. Toward a GPU-aware comparison of explicit and implicit CFD simulations on structured meshes[J]. Computers & Mathematics with Applications, 2017, 74(1): 201-217.

[7] PHILLIPS E, ZHANG Y, DAVIS R, et al. Rapid aerodynamic performance prediction on a cluster of graphics processing units[C]// 47th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition. 2009: 565.

[8] JACOBSEN D, THIBAULT J, SENOCAK I. An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters[C]// 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition. 2010: 522.

[9] BOLZ J, FARMER I, GRINSPUN E, et al. Sparse matrix solvers on the GPU: conjugate gradients and multigrid[J]. ACM Transactions on Graphics (TOG), 2003, 22(3): 917-924.

[10] CORRIGAN A, CAMELLI F F, LÖHNER R, et al. Running unstructured grid-based CFD solvers on modern graphics hardware[J]. International Journal for Numerical Methods in Fluids, 2011, 66(2): 221-229.

[11] NGUYEN M T, CASTONGUAY P, LAURENDEAU E. GPU parallelization of multigrid RANS solver for three-dimensional aerodynamic simulations on multiblock grids[J]. The Journal of Supercomputing, 2019, 75(5): 2562-2583.

[12] OYARZUN G, CHALMOUKIS I A, LEFTHERIOTIS G A, et al. A GPU-based algorithm for efficient LES of high Reynolds number flows in heterogeneous CPU/GPU supercomputers[J]. Applied Mathematical Modelling, 2020, 85: 141-156.

[13] LI A, SONG S L, CHEN J, et al. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(1): 94-110.

[14] WILLIAMS S, WATERMAN A, PATTERSON D. Roofline: an insightful visual performance model for multicore architectures[J]. Communications of the ACM, 2009, 52(4): 65-76.

[15] BUTCHER J C. On the implementation of implicit Runge-Kutta methods[J]. BIT Numerical Mathematics, 1976, 16(3): 237-240.

[16] ZHONG X. Additive semi-implicit Runge-Kutta methods for computing high-speed nonequilibrium reactive flows[J]. Journal of Computational Physics, 1996, 128(1): 19-31.

[17] THIBAULT J, SENOCAK I. CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompressible flows[C]// 47th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Sxposition. 2009: 758.

WEN Min-hua, born in 1988, associate engineer, is a member of China Computer Federation. His main research interests include engineering computing and so on.

LIN Xin-hua, born in 1979, senior engineer, is a member of China Computer Federation. His main research interests include performance modeling and optimization.