计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(3):60−69 [doi: 10.15888/j.cnki.csa.007842]

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

适用于异构集群的混合并行流线生成系统①

刘　俊[1,2], 高　阳[1], 单桂华[1], 迟学斌[1,2]

[1](中国科学院 计算机网络信息中心, 北京 100190)
[2](中国科学院大学, 北京 100049)
通讯作者: 单桂华, E-mail: sgh@sccas.cn

摘　要: 流线是流场可视化的主要方法之一, 而针对大规模流场的流线生成由于计算量大往往需要采用高性能计算机这样的并行计算环境结合并行化算法以实现计算加速. 在当前异构计算系统越来越普遍的情况下, 为了充分利用并行异构计算环境的计算能力, 实现更高效的并行流线生成, 本文采用了基于数据并行原语结合分布式消息通讯的技术架构, 设计了一套适用于异构集群的混合并行流线生成系统, 并在此基础上针对数据分块、数据冗余化及进程通讯策略等方面进行设计, 提出并实现了一套并行粒子追踪算法. 该系统被部署于国产超算平台上, 并针对大规模CFD 流场模拟结果数据可视化应用开展了实验. 本文给出了相关实验结果, 分析了核心并行算法的速度性能、可扩展性以及负载均衡等方面情况, 说明了系统及算法的有效性和可扩展性.

关键词: 流场可视化; 并行流线生成; 粒子追踪; 数据并行原语; 异构并行算法

引用格式: 刘俊,高阳,单桂华,迟学斌.适用于异构集群的混合并行流线生成系统.计算机系统应用,2021,30(3):60–69. http://www.c-s-a.org.cn/1003-3254/7842.html

Hybrid Parallel Streamline Generation System Suitable for Heterogeneous Clusters

LIU Jun[1,2], GAO Yang[1], SHAN Gui-Hua[1], CHI Xue-Bin[1,2]

[1](Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Streamline is one of the main methods of flow visualization. In light of a large amount of computation, the streamline generation from large flow fields usually requires parallel computing environments, such as high-performance computers and parallel algorithms, to accelerate computation. As wider application of heterogeneous computing systems, we design a hybrid parallel streamline generation system suitable for heterogeneous clusters in terms of data decomposition, overlapping and communication strategy with technologies such as data-parallel primitives and message passing interface to maximize the computing power of the heterogeneous parallel computing environment and achieve more efficient parallel streamline generation. A set of algorithms related to parallel particle advection are proposed and implemented. The system is deployed on a domestic supercomputer, and experiments are conducted to visualize the results of a large-scale CFD flow field simulation. This study provides relevant experimental results and analyzes the performance, scalability, and load balance of the core parallel algorithm, verifying the effectiveness and scalability of the system and algorithm.

Key words: flow visualization; parallel streamline generation; particle advection; data parallel primitives; heterogeneous parallel algorithm

① 基金项目: 国家数值风洞工程基础研究 (NNW2019ZT6-B19); 国家重点研发计划 (2019YFB1704201)
Foundation item: Basic Research on National Numerical Wind Tunnel Engineering (NNW2019ZT6-B19); National Key Research and Development Program of China (2019YFB1704201)
收稿时间: 2020-07-26; 修改时间: 2020-08-25; 采用时间: 2020-09-01; csa 在线出版时间: 2021-03-03

针对 CFD 流场模拟结果的可视化处理是流体动力学分析的重要一环. 常用的流场可视化处理方法包括流线、轨迹线、FTLE 分析等方法. 而上述方法的核心算法均为粒子追踪算法, 因此, 考虑到粒子追踪算法之于流场可视化的重要性, 为了提高流场可视化分析效率, 领域研究人员针对该算法的优化和并行化开展了一系列研究工作. 现有的并行化算法往往考虑的是比较单一的计算架构, 有的是基于共享存储的计算环境, 有的是 HPC 集群计算环境, 有的是结合 GPU 加速器的异构计算环境. 为了兼容各类型计算架构, 充分发挥异构并行计算环境的计算资源以实现流线快速生成, 我们基于自主研发的 GPVis 平台[1], 采用了数据并行原语结合分布式消息通讯的技术架构, 设计了一套流线生成系统, 通过数据并行原语实现设备无关的并行粒子追踪算法, 并结合数据空间域分解技术实现整个流场区域的流场生成任务的分而治之策略. 本文基于这一系列方法在国产超算平台上完成了该系统的实现和部署, 并与已有的流线生成系统在不同参数条件下进行了测试比较. 在对测试结果进行分析后, 我们发现新系统可有效改善传统流线生成方法的负载不均衡问题, 以较高的并行效率实现分布式计算环境下的可扩展计算.

1 研究背景

1.1 并行粒子追踪

粒子追踪计算是流场可视化的重要算法之一. 而针对粒子追踪算法并行化的基本方法主要可分为两类: 种子点并行法 Parallelize-Over-Seeds (POS) 以及数据块并行法 Parallelize-Over-Data (POD)[2]. 其中, POS 算法的基本思想是将初始种子点均匀分布在各处理单元上, 并且让各处理单元进行独立计算. 而 POD 算法则是通过域分解将整体数据分为众多数据块, 并将各数据块分配到各处理单元, 然后每个处理单元负责完成进入该数据块的粒子的追踪计算, 直至粒子达到终止条件或进入其它数据块空间.

POS 方法往往需要较大的计算设备内部存储空间来支撑, 或者需要通过频繁的 I/O 加载操作来消减一部分的内部存储空间需求. 这是由于 POS 方法并未针对数据分块方面制定域分解策略, 它要求每个节点在执行期间可以随机访问全部数据. 这就造成各处理单元要么在启动时读入所有数据, 要么将整个数据按空间分成小块再按需读取. 两种方法中前者的性能容易受到设备存储空间 (系统内存以及 GPU 显存) 的瓶颈限制, 后者则会带来较高的 I/O 相关的成本 (如大量的 I/O 启动延迟所带来的时间成本, 或者为防止各计算单元的 I/O 操作之间的冲突而提供的高带宽并行 I/O 能力支撑而带来的硬件成本).

POD 方法适用于高度可扩展的 HPC 环境, 可通过数据分块将单节点的内部存储空间需求降低到可接受的水平, 但同时由于任务的不可迁移也往往导致严重的负载不均衡问题.

鉴于两类基本方法存在的弊端, 研究人员在两类方法基础上, 发展出任务窃取法[3]、任务请求法[4] 等混合并行方法. 混合并行方法一般会动态地且冗余地将数据块分配给各处理单元, 以实现更优的负载均衡, 同时也通过先验的规划或实时的策略来降低冗余数据读取所带来的附加的 I/O 操作对算法整体可扩展性的影响.

1.2 数据并行原语

数据并行原语 (Data Parallel Primitives, DPP) 的设计思想最早来源于一种用于数据并行计算的扫描矢量模型[5]. 在数据结构、计算几何、图分析以及数值分析等众多算法领域, 扫描矢量模型被应用于重新设计各种算法以实现算法的并行化. 而数据并行原语 DPP 作为扫描矢量模型的升级版本, 可采用包括 Map、Scan、Sort 和 Reduce 在内的一些基础操作来实现各类并行算法. 从目前的研究来看, 除了 Delaunay、FFT 等少量算法, 在 DPP 模型基础上, 大部分的可视化算法都可以有相应的实现方法[6]. 虽然并不是所有可视化算法都可以通过 DPP 来实现, 但并不妨碍 DPP 模型在海量线程计算设备中的应用.

VTK-m[7] 是较典型的且功能相对全面的一套基于 DPP 的可视化算法库. 为了发展适用于 GPU、MIC 等海量线程计算环境下的可视化算法, Kenneth 等综合借鉴了 PISTON[8]、Dax[9] 和 EAVL[10] 等关注于海量线程计算的并行可视化架构的设计思想, 并在设备及算法通用性方面进行了优化, 于 2016 年提出了 VTK-m 并行可视化计算架构. VTK-m 采用数据并行原语 DPP 作为算法并行化基本单元, 提供了一套与 DPP 兼容的计算设备抽象模型, 并针对底层硬件体系架构进行了兼容适配, 以提高 VTK-m 的兼容性和可移植性. 由于

用户无需考虑底层硬件实现细节就可以设计实现各类高效率的算法, 因此 VTK-m 有助于降低海量线程计算环境下的并行算法的实现难度. 目前, VTK-m 的计算设备抽象模型可兼容包括 GPU、MIC 在内的多种海量线程计算设备.

VTK-m 将各类通用算法以基本任务单元的形式进行继承封装, 而基本任务单元的运行调度是通过各类符合 DPP 模型的设备适配器来驱动的 (如图 1 所示). 这样, 基于 VTK-m 设计的可视化应用则可以通过底层设备适配器的切换而运行于各类海量线程环境. 通过 VTK-m 来编写可视化算法除了可保障算法的设备通用性之外, 在稳定性和安全性的保障方面也很便利. 例如, 如果通过 Map 映射操作来跟踪数据的拓扑连接时, VTK-m 会在计算设备端的内部自动完成拓扑结构索引的处理而避免了不安全的操作, 这就使得编写基本任务单元比直接并行操作整体数据结构要更容易且更安全.

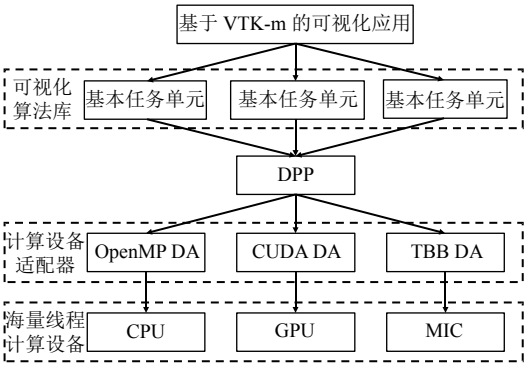


图 1 VTK-m 可视化应用运行机制

1.3 基本任务单元

基本任务单元是指组成大规模数据并行处理程序的针对小规模数据的无状态的细粒度串行程序片段[11], 而采用 DPP 设计并行算法的核心任务之一就是基本任务单元的定义. 我们采用 VTK-m 来设计海量线程加速器环境下的并行粒子追踪算法. 由于 VTK-m 是面向共享内存的海量线程运行环境而设计的可视化算法库, 没有分布式存储及高成本通讯等问题, 而且计算资源受到单机能力顶值的限制, 不需要过多考虑算法的大规模扩展问题, 因此, 较适合采用 POS 的并行策略来设计并行粒子追踪算法. 但在实际算法设计中, VTK-m 与传统的 POS 方法也有一定的区别. 比如在定义基本任务单元的层面, VTK-m 虽然采用了与传统 POS 方法类似的做法, 将并行粒子追踪的基本任务单元定义为单个粒子的追踪任务, 但在细节上仍然与传统方法有所不同.

很多并行粒子追踪算法的基本计算单元将固定步数的单个或一组粒子追踪作为任务单元, 如文献 [12] 中采用的办法, 这样可以将计算粒度做到足够小, 从而有利于整体的负载均衡的实现. 但由于过小的粒度在异构计算环境下容易产生额外的调度开销, 因此, VTK-m 并没有按这个方法来定义任务单元, 而是采用单个粒子的整条的追踪计算任务看作基本任务单元. 这样做既有利于降低任务调度开销, 也有利于降低任务单元之间的依赖性, 从而使得采用 Map 操作原语进行算法映射更为高效.

在确定了基本任务单元后, 只要将基本任务单元映射到一组执行线程上, 就可以通过海量线程计算设备的大规模并行能力来完成总体计算任务. 相应的, 在 VTK-m 的并行粒子追踪模块中, 采用设备无关的通用方式来描述单个粒子追踪基本操作的流程, 形成 Advect 函子, 然后通过 Map 操作来实现以函子形式封装的基本任务单元向底层并行计算设备的映射. Map 操作是 VTK-m 中被高度优化的 DPP 之一, 针对底层硬件的特点采用了定制的任务映射方法以执行海量的基础任务单元. 特别是当运行环境是 GPU 这类加速器设备时, Advect 函子所涉及的输入输出数据都将自动传输到设备内存中, 或者从设备内存中导出. 通过这种方式, 基本任务单元的执行位置实际上对开发人员是不可见的, 这也是为什么在编写基本任务单元时只能通过数组句柄来访问输入输出数据的主要原因.

2 系统设计

受到单机计算存储能力顶值的限制, 大规模流场数据无法直接加载到单台机器的主线内存或加速器设备内存中. 因此, 无法仅靠基于 GPU、MIC 等海量线程计算硬件的并行粒子追踪系统来满足大规模流场数据可视化需求. 为了实现大规模流场的并行可视化, 我们需要在其基础上构建适用于异构集群环境的并行粒子追踪系统.

流场可视化方法常用于处理由 CFD 数值模拟产生的结果数据或中间数据. 而 CFD 模拟往往会因为计

算过程的特点在各类不同的计算环境下运行, 因此, 流场可视化软件也需要适应如 GPU 加速器、SMP 计算机、HPC 计算机集群等各类不同的计算环境. 基于 VTK-m 的并行粒子追踪算法可以通过其架构中的计算设备适配器产生可运行于不同的计算硬件上的可执行程序, 具有很强的可移植性. 这是适应于上述的科学可视化运行环境特点的设计, 但美中不足的是, VTK-m 并没有解决在大规模高性能计算机环境下开展可视化计算的问题, 这是因为在类似于 SMP、GPU 和 MIC 这种采用共享存储结合高速总线的体系架构中, 数据访问速度非常高, 无需过多考虑由于数据迁移所产生的时间消耗. 而 HPC 环境下却需要重点考虑如 I/O、通讯等方面的负载对计算效率的影响. 这也正是本文中系统设计方面工作所面临的主要问题.

由于 VTK-m 可以在普通计算节点以及大规模线程环境下以粒子为处理单元并行地运行流线生成算法, 因此, 为了实现异构集群计算环境下的大规模并行粒子追踪, 我们只需要基于 MPI 消息通讯接口, 实现相关算法在 HPC 条件下的动态扩展即可. 为了达到上述目标, 我们设计了的适用于流场可视化的数据处理系统框架. 如图 2 所示, 系统框架结构包含 5 大部分, 分别为: 函子计算层, 数据分块层, 任务调度层, 进程通讯层以及渲染融合层.

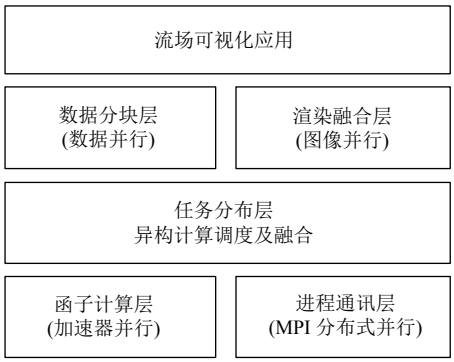


图 2 并行粒子追踪系统结构图

(1) 函子计算层. 函子计算层负责调度单节点中的多线程或大规模线程资源, 以实现分配到单个节点上的可视化任务可以得到节点上足够全面的计算能力支撑. 在本系统中, 函子计算层的主要任务是基于 VTK-m 的多线程环境并行处理分配到节点本地的粒子追踪基本任务单元, 因此, 函子计算层主要是由 VTK-m 库以及之上的算法适配库组成. 通过继承和调用按 VTK-m 设计思想封装的基本任务单元模块 WorkletMapField, 系统实现了兼容 GPU、MIC 等海量线程计算设备的并行粒子追踪模块.

(2) 进程通讯层. 进程通讯层主要负责节点进程间的基于 MPI 的消息接收和发送, 以及通讯相关的一系列并行算法, 主要模块包括消息通讯模块、数据交换模块以及进程协调模块. 其中消息通讯模块是对 MPI 接口的封装, 可用于体量小且实时性要求较高的消息的发送和接收. 数据交换模块在 MPI 协议基础上将节点间交换的大块数据封装成 MPI 数据包, 并逐包跟踪发送及接收进度, 以保证数据的顺序性和完整性. 进程协调模块包含了一系列分布式并行算法的 MPI 实现, 如扫描、搜索、合并、排序等. 在并行流线生成系统中, 粒子位置信息的交换以及积分结果曲线的汇总过程中需要用到较多的通讯资源, 此时需调用进程通讯层相关模块完成相应任务.

(3) 任务分布层. 任务分布层主要负责各节点及进程中计算任务的全生命周期管理. 从初期的任务创建, 到中期的任务执行以及任务迁移, 再到最后的任务回收, 任务分布层各功能模块以任务为单元跟踪其即时状态, 并根据任务的起始条件、终止或暂停条件调整任务的执行状态, 根据任务的数据需求以及节点的负载情况, 按总体任务调度策略发送或接收任务, 完成任务负载在节点间的流动. 由于任务执行过程采用的是基于 VTK-m 的多线程架构, 因此任务分布层还需要在线程级实现基本任务单元的调度运行, 从而实现线程级和进程级任务的有机融合.

(4) 数据分块层. 数据分块层主要负责实现可视化算法并行化中的域分解策略 (如循环分块、八叉树分块、二分树分块等), 并在数据分布、存储及 I/O 等方面提供可调用的接口. 其中数据分布策略在并行算法中的 I/O 消耗最小化、负载均衡化等方面起到非常关键的作用, 尤其是对于粒子追踪这类负载不可预测的情况. 数据分布过程中单元数据块大小、重叠区域的设置、数据区块的节点分配都对可视化算法的整体运行效率以及负载均衡有很大影响. 在本文中, 我们针对并行粒子追踪算法中区域划分策略及冗余扩充策略进行了讨论, 并设计了新算法, 具体方法请见第 3 节.

(5) 渲染融合层. 渲染融合层主要负责图像相关的计算任务生成、执行以及合并操作, 主要功能包括图像渲染、图层融合、图元拼合等功能. 该层算法主要针对于图像并行的并行可视化算法, 而该类算法在体绘制和光线追踪法渲染等可视化应用中更为常见. 在本文所涉及的任务范围中, 渲染融合层只负责流线在并行生成并汇总后的单机渲染.

3 算法实现及改进

正如 1.1 节中分析的那样, 包括 POS 和 POD 在内的基础并行算法均存在较严重的弊端, 而现有的混合并行算法有些并不适用于异构集群环境, 有些则存在各种其它问题, 如基础任务单元不兼容、包含整体预先分析处理步骤等.

为了避免上述问题, 我们设计了一套混合并行粒子追踪算法, 在总体的并行策略上采用了 POD 并行策略, 而在每个节点上由于采用了多线程的共享存储架构, 因此在节点层面上为了充分利用本地的存储计算资源采用了基于 VTK-m 实现的 POS 并行粒子追踪策略.

以下针对本文提出的混合并行算法中的具体方法进行说明如下.

3.1 粒子追踪算法

设 f 为流场, y_0 为预设种子点, 则由该流场生成的流线是指如下常微分方程的解:

$$\frac{\mathrm{d}y(s)}{\mathrm{d}s} = y'(s) = f(y(s)); \quad y(s_0) = y_0$$

其中, $y(s)$ 是一个关于沿流线方向的参数化步长距离 s 的三维位置函数. 求解流线的过程一般采用沿流线方向逐步积分的办法, 这个过程我们称之为粒子追踪过程. 粒子追踪过程其实质是粒子在流场作用下不断进行积分推进的过程, 常用的粒子追踪算法包括 2 阶、3 阶、4 阶以及 4.5 阶的龙格库塔积分方法. 本系统采用的是 4 阶龙格库塔积分求解方法.

采用 DPP+POD 策略的并行粒子追踪算法让不同的进程分管不同的数据块区域, 其算法如算法 1 所示. 因为积分过程会将粒子推向一系列难以预测的空间位置而形成流线, 而组成流线的这些空间位置有可能属于分布于不同进程的不同的数据块区域, 所以各个处理进程在计算本地的粒子积分的同时, 还需要接收可以落入本地数据区域的粒子, 以及将由于积分推进跨界进入非本地所属数据区域的粒子准确发送到数据块所属进程. 由上述流程可看出, 各进程无法仅根据自身情况判断计算过程是否可以终止, 而是需要将一个全局的统计信息作为终止条件来进行判断, 这就为算法的大规模并行带来了难度, 是实现算法高可扩展性所必须解决的问题. 另外, 与传统的 POD 策略并行算法不同的是, 由于基本任务单元的差别, 积分计算前将按份抽取一定量的粒子进行积分, 每份粒子的数量要求与海量线程计算设备的线程个数尽量接近.

算法 1. 基于 DPP+POD 策略的并行粒子追踪算法

1) 加载本进程负责的数据块和种子点粒子, 已终止粒子累积计数器置零;
2) 判断本地粒子数是否为 0, 如为 0 则到步骤 9);
3) 从全部本地粒子中取出一份粒子;
4) 调用 *Map* 操作原语在多线程设备中对取出的粒子执行粒子追踪, 得到一部分已终止粒子和另一部分跨界粒子;
5) 将本轮已终止粒子数广播到全部进程;
6) 将本轮已终止粒子数累计入已终止粒子累积计数器;
7) 接收由其它进程广播发出的已终止粒子数, 并全部累计入已终止粒子累积计数器;
8) 对跨界粒子排序, 按粒子当前位置所属进程号将全部跨界粒子发送至所属进程;
9) 接收由其它进程发出的跨界粒子, 存入本地粒子库;
10) 判断当前已终止粒子累积计数器值是否与粒子总数相等, 如相等则退出, 否则转到步骤 2).

3.2 数据分块

实际上, 由于流场本身的不确定性, 粒子追踪的负荷在空间域的分布是随机的, 因此, 在没有先验信息的前提下, 采用数据无关的分块策略肯定是无法达到最佳的负载均衡的. 这也正是循环分块法建议在每个节点增加数据块数, 以逼近更均衡的负载分布的原因.

粒子追踪中较常用的数据分块策略包括循环分块法[13]、k-D 树分块法[14] 等. 循环分块法注重于计算进程的负载均衡, 因此该算法在每个进程分配散布于空间不同位置的数据块, 以缓解由粒子聚集所导致的负载失衡. 为实现进程中各个数据块的空间均匀散布, 循环分块法会要求进程个数不能被任意维度的网格数整除. 但是, 循环分块法选择忽略由于粒子流动本身的空间连续性而造成的算法邻域空间强相关性, 从而导致较高的节点间通讯量并极大地影响了整体并行效率. 而 k-D 树分块法想尽量避免粒子的跨节点漂移以减少节点间通讯从而提高并行效率, 该方法针对粒子分布

来进行数据块的分配, 适用于面向时变流场可视化的动态分块策略, 但这种策略不适用于将粒子整条追踪过程设置为基本任务单元的算法, 因为该方法要求在追踪过程中检查粒子位置. 另外, 循环分块法采用固定的块大小也不利于算法的灵活性和可扩展性.

为了改善循环分块法的以上问题, 我们采用了递归二分法进行空间分块. 假设采用 N 个计算进程开展并行计算, 每个进程要求分配 nb 个数据块, 那么, 首先对整个数据空间进行多次切分, 切分原则为在 3 个维度中选择最长轴进行切分, 在多次切分后, 分块数总能超过 $N\times nb$ 块, 再将部分多余的分块与相邻分块融合后, 分块数可达到 $N\times nb$ 块, 从而满足数据分布化需求. 同时, 为减少维度分块数与进程数之间可能存在的整除关系所带来的数据块空间聚集的可能性, 各数据块将在随机洗牌后再进行循环分配.

3.3 数据冗余化

并行算法的可扩展性与执行过程中的通讯量以及负载均衡程度紧密相关. 针对并行粒子追踪算法, 数据通讯主要发生在粒子跨越数据块边界进入另一个数据块的情况, 因此, 当粒子所属数据块足够大时, 越界情况发生的机率将会减少. 另外, 并行粒子追踪过程中负载失衡的主要原因在于某些区域存在相对低速区域、涡旋、临界点等需要长时间积分才能通过甚至无法通过的区域 (又称为陷阱区), 尽管可以采用如文献 [13] 中所述办法, 在经过判断后发现这类粒子时立刻中止该粒子积分过程, 但这种办法实际阻断了后续结果的计算, 有可能对流场分析过程产生干扰, 而且基于 VTK-m 的任务单元定义方法也决定了不可能在计算中途阻断粒子积分, 而这种情况也为整体系统的并行效率提升增加了难度.

由前面的分析可看出, 不管是在降低数据通讯量或是提升负载均衡程度的解决办法中, 在无重叠的数据分块的基础上, 将数据区域进行冗余化扩充都是有效的办法之一, 我们称之为数据冗余化. 在并行算法中, 区域重叠法通过建立重叠区来扩展数据区域是最常用的数据冗余化策略. 重叠区又称为鬼区, 在多类并行算法中均有采用, 当然也包括一些并行粒子追踪算法.

3.4 进程间通讯

并行粒子追踪算法中主要的进程间通讯分为两类: 一类是由于粒子越界所造成的负载流动, 这类通讯是点对点的通讯, 但具有随机不可控的特性, 无法预测通讯发生的位置和数量; 另一类是粒子终止状态汇报所产生的进程间通讯, 这类通讯是点对全体的通讯, 需要将单个进程的计算结果汇报到所有进程, 让每个进程了解当前已完成积分的粒子数, 在全部粒子完成积分时终止等待负载的工作循环, 进入流线生成的收尾阶段.

粒子在流场推动下会在不同的数据块区域之间移动, 由于不同的数据块预先指定由不同的进程负责, 因此积分计算负载将不可控地且被动地在进程间流动, 从而造成可能的负载聚集. 通过前面讲到的循环数据分块方法, 我们可以缓解这种负载聚集带来的通讯量聚集. 另外, 由于频繁的发送越界粒子有可能造成通讯数量过, 多且接收粒子的进程有可能在收到不足份的粒子前就开始积分计算从而造成计算时间的浪费, 因此, 改进的算法会在一个时间段里累积一部分越界粒子, 当粒子数量达到足份量时才进行发送, 除非在该时间段里无法累积到足量的粒子, 此时只能将全部越界粒子发出.

对于粒子终止状态汇报机制, 原有的广播形式的汇报算法实际上是非常昂贵的, 尤其是在进程数足够大时, 广播通讯的规模以几何规模增长, 对整体的扩展性能产生了非常大的影响. 为了改善这一情况, 我们设计了一套树型汇报机制, 如图 3 所示.

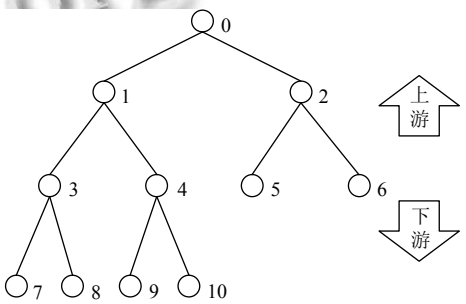


图 3 树型状态汇报机制示意图

如图 3 中, 所有进程被组织成二叉树的节点, 每个进程最多有一个上游进程和两个下游进程, 当下游进程汇报粒子终止情况下, 上游进程进行统计汇总再上报上游进程, 当最上游根进程发现全部粒子积分均终止时, 将向下游进程发送退出循环的信号并退出循环, 其它进程在收到退出循环信号后也进行同样操作. 整

个通讯进程是异步且点对点的, 因此对整个计算过程的影响被降到最小. 经改进的并行算法如算法 2 所示.

算法 2. 改进的基于 DPP+POD 策略的并行粒子追踪算法

1) 加载本进程负责的数据块和种子点粒子, 已终止粒子累积计数器置零, 打开循环开关;
2) 判断本地粒子数是否为零, 如为零则转到步骤 12);
3) 从全部本地粒子中取出一份粒子;
4) 调用 *Map* 操作原语在多线程设备中对取出的粒子执行粒子追踪, 得到一部分已终止粒子和另一部分跨界粒子;
5) 对跨界粒子排序, 按粒子当前位置所属进程号将全部跨界粒子发送至所属进程;
6) 接收下游进程发来的经汇总的已终止粒子数消息, 并与本轮已终止粒子数相加得到待发送的经汇总的已终止粒子数;
7) 向上游进程发送经汇总的已终止粒子数;
8) 将经汇总的已终止粒子数累计入已终止粒子累积计数器;
9) 判断当前已终止粒子累积计数器值是否与粒子总数相等, 如相等则转到步骤 11), 否则转到步骤 12);
10) 接收上游发送的退出循环指令, 如未收到则转到步骤 12);
11) 关闭循环开关, 并向下游进程发送退出循环指令;
12) 接收由其它进程发出的跨界粒子, 存入本地粒子库;
13) 判断循环开关是否打开, 如关闭则退出, 否则转到步骤 2).

4 应用测试

4.1 测试环境

我们在国产高性能计算集群上开展了系统和算法的相关测试. 该集群每个节点包含 32 个海光 C86 7185 CPU (32 核, 主频 2.0 GHz), 128 GB 内存. 编译器采用 Intel 编译器, 启用 OpenMP 与 VTK-m 相结合实现多线程并行, 多节点分布式并行 MPI 环境采用的是 OpenMPI, 我们根据实验需要使用了 1 至 256 个节点开展测试.

4.2 测试数据及方法

我们采用了由 CFD 模拟程序实际产生的数据来开展算法测试. 该测试数据是一套燃烧模拟结果数据[15,16], 其网格分辨率为 670×459×459, 包含亿级网格规模. 相关的 CFD 模拟实验模拟了超音速喷射的可燃气体在空气中混合燃烧产生涡流的过程, 以研究燃烧反应过程对涡结构的影响. 模拟采用的是 OpenCFD-Comb 模拟程序. 本次测试只采用了一个时间步, 以实现基于可视化的稳态流场分析. 图 4 显示了该数据的流场可视化效果.

我们采用了 16 至 1024 个进程开展并行粒子追踪计算, 单个粒子的追踪步数最大值设定为 10 000 步. 并行化方法均采用经过随机排序的递归二分法进行数据分块, 并采用了区域重叠最大化的数据冗余化策略, 单进程限定的内存使用规模为 64 MB. 为了测试系统性能, 我们在相同的运行环境下编译运行了 ALPINE 系统[17] 中的粒子追踪模块, 并基于相同的数据以及计算目标进行了对比测试. ALPINE 系统中的粒子追踪模块是同样采用了 VTK-m 作为核心计算库的异构并行可视化软件.

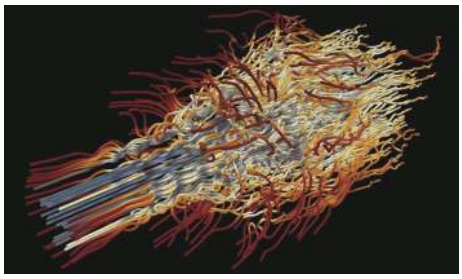


图 4 测试数据可视化效果图

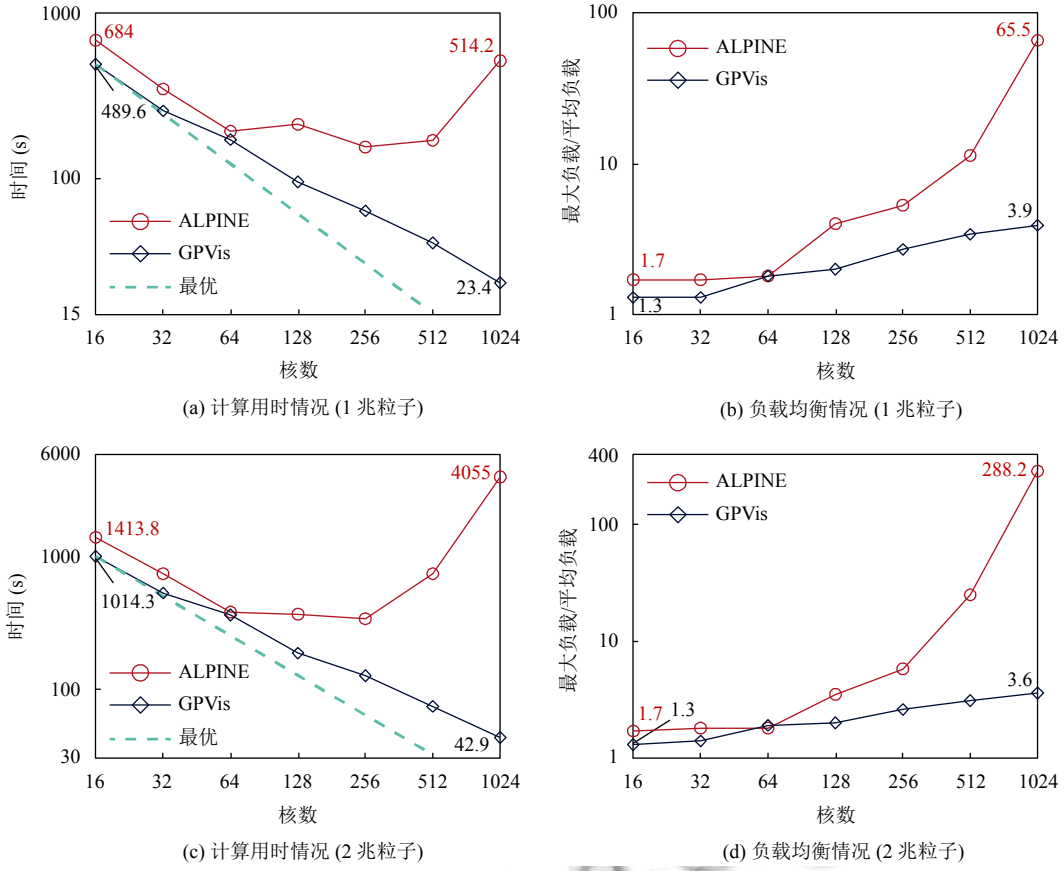
4.3 性能对比分析

我们设计了一套强扩展测试方案, 即通过总量固定的任务量来测试并行算法在不同的节点进程数的情况下的加速效率. 对于粒子追踪算法来说, 在设置固定的粒子总数的前提下, 将积分任务均匀分布到各节点或进程上开展强扩展测试, 我们可以测量并观察总的执行时间是如何随进程数的变化而发生变化的. 理想状态下, 加速比将线性增加. 而实际上对于进程间存在通讯的并行算法来说, 在进程数达到一定规模的情况下会出现加速比无法提升甚至下降的情况.

本次测试按两种粒子规模进行了粒子追踪测试, 其粒子规模 n 分别为 1 兆以及 2 兆. 测试结果如图 5 所示. 其中图 5(a) 为跟踪 1 兆粒子时的计算时长与核数关系图, 图 5(b) 为对应的负载均衡与核数关系图, 图 5(c) 为跟踪 2 兆粒子时的计算时长与核数关系图, 图 5(d) 为对应的负载均衡与核数关系图. 从测试结果可看出, ALPINE 系统在核数达到 512 核之后迅速表现出背离加速方向的发展趋势, 且在数据规模相对较大时, 这种趋势更为明显且出现的时间也更早, 如图 5(c) 所示, 针对 2 兆粒子进行粒子追踪时, 核数达到 128 核之后, 该系统就无法进一步实现计算时间的缩减. 相比较而言, GPVis 平台的粒子追踪系统在核数达到 1024 核的规模时, 仍然表现出加速潜力, 且整体加速趋势趋近于线性加速.

4.4 负载均衡分析

图 5(b) 和图 5(d) 通过负载失衡度这一指标说明了计算过程的负载均衡情况. 负载失衡度是各进程中进程最大负载与各进程平均负载的比值. 从图可看出, 两组测试表现出相似的模式, 即 ALPINE 在核数达到 256 核之后, 负载失衡的情况迅速恶化, 而 GPVis 在核数增加时, 负载失衡度略有提升, 但仍在可控范围. 从两组测试结果中也可看出不同之处, 在数据规模相对较大时, ALPINE 的失衡度升高较快, 而 GPVis 的负载失衡度受到数据规模的影响几乎可以忽略.



(a) 计算用时情况 (1 兆粒子)

(b) 负载均衡情况 (1 兆粒子)

(c) 计算用时情况 (2 兆粒子)

(d) 负载均衡情况 (2 兆粒子)

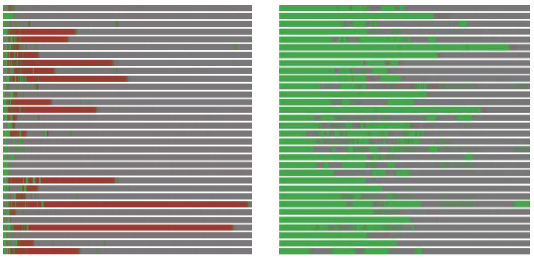
图 5 系统强扩展性能测试对比图

在对运行过程中的具体计算日志进行详细记录后, 我们发现了两者的主要区别. 我们记录两组测试中 512 核的运行日志, 并在 512 个进程中选取了 32 个进程进行可视化, 图 6 即为两组运行测试日志的可视化结果, 图 6(a) 为 ALPINE 运行日志的抽样, 图 6(b) 为 GPVis 运行日志的抽样. 图中每个横条代表一个进程的工作状态发展过程, 向右方向为时间方向, 两组横条的长度被归一化为相同的长度, 以表现整个计算过程. 横条中绿色代表计算时间, 红色代表通讯时间. 从图中可看出, ALPINE 系统由于采用了全局广播通讯的终止粒子汇报机制, 当核数增加时, 相应的通讯时长逐渐占据主导而拖慢了整体性能, 而 GPVis 粒子追踪系统采用的优化的终止粒子汇报机制, 相应的通讯压力则非常小, 从整个计算周期来看, 占据非常小的比例. 这也是图 5 中两者负载失衡度变化趋势发生区别的主要原因.

4.5 大规模测试分析

当采用 FTLE 进行流场分析处理时, 需要开展大规模的流线生成计算, 计算规模为每个网格点设立一个被追踪的粒子. 那么对于本案例针对数据来说, 粒子数取与网格数相当的规模后大小约为 140 兆. 针对这一规模, 我们开展了相应的测试. 测试结果如图 7 所示, 从中可看出, 目标系统在较大的计算负载下且当核数达到万核级别时依然表现出可接受的加速趋势, 但并

行效率则在 1024 核的 50% 之后出现快速下降, 其中在采用 2048 核、4096 核和 8192 核时的并行效率分别为 29%、21.3% 和 11.1%.



(a) ALPINE 运行日志抽样 (b) GPVis 运行日志抽样

图 6 系统运行甘特图对比

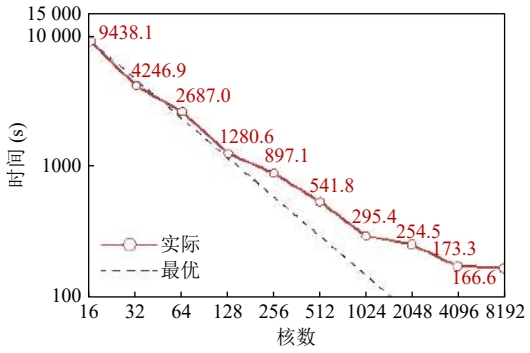


图 7 大规模粒子追踪任务测试结果

4.6 测试小结

从测试结果可看出, 本文提出的系统及算法可适用于异构集群计算环境, 且在计算速度、可扩展性等方面表现优于 ALPINE 系统粒子追踪模块. 经过细致的运行日志分析, 我们发现了文中提出进程通讯算法优化方法在系统整体的可扩展性提升及负载均衡化方面起到了重要作用. 经过大规模测试分析, 我们认为, 本系统可用于大规模集群环境下的并行流线生成及流场分析场景.

5 结束语

本文针对异构集群环境下的并行流线生成系统提出一种改进的基于 POD+DPP 的粒子追踪算法设计实现方案, 并针对流线生成过程中的粒子追踪算法的并行化的实现细节进行了分析和介绍. 在针对流线生成系统的并行测试后, 我们发现该系统可实现大规模 CFD 流场的流线并行生成, 达到了理想的可扩展能力, 并且在并行规模达到千核级别时仍表现出较高的并行效率, 在核数提高至万核级别时保持计算时长持续减少.

由于这次测试针对的数据属于稳态流场数据, 因此并未涉及非稳态流场的分析问题. 稳态流场与非稳态流场的流场或轨迹线生成方法有很大的差别, 前者更注重计算的分配, 后者更注重数据的动态加载过程, 后续我们将对本系统进行进一步升级, 并应用于时变流场的可视化处理. 另外, 从性能分析结果中可看出, 本文提出的优化方法虽然较传统方法提高了计算时间占用率, 但在进程数较多时并行效率仍然处于较低的水平, 存在进一步优化的空间.

参考文献

1 单桂华, 刘俊, 李观, 等. 面向大规模数据的科学可视化系统 GPVis. 数据与计算发展前沿, 2019, 1(1): 46–62.

2 Pugmire D, Childs H, Garth C, *et al*. Scalable computation of streamlines on very large datasets. Proceedings of Conference on High Performance Computing Networking, Storage and Analysis. Portland, OH, USA. 2009. 1–12.

3 Dinan J, Larkins DB, Sadayappan P, *et al*. Scalable work stealing. Proceedings of Conference on High Performance Computing Networking, Storage and Analysis. Portland, OH, USA. 2009. 1–11.

4 Müller C, Camp D, Hentschel B, *et al*. Distributed parallel particle advection using work requesting. Proceedings of 2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV). Atlanta, GA, USA. 2013. 1–6.

5 Blelloch E. Vector Models for Data-parallel Computing. Cambridge: MIT Press, 1990.

6 Moreland K, Geveci R, Ma KL, *et al*. A classification of scientific visualization algorithms for massive threading. Proceedings of the 8th International Workshop on Ultrascale Visualization. Denver, CO, USA. 2013. 1–10.

7 Moreland K, Sewell C, Usher W, *et al*. VTK-m: Accelerating the visualization toolkit for massively threaded architectures. IEEE Computer Graphics and Applications, 2016, 36(3): 48–58. [doi: 10.1109/MCG.2016.48]

8 Lo LT, Sewell C, Ahrens J. PISTON: A portable cross-platform framework for data-parallel visualization operators. The Eurographics Association. 2012. 11–20. [doi: 10.2312/EGPGV/EGPGV12/011-020]

9 Moreland K, King B, Maynard R, *et al*. Flexible analysis software for emerging architectures. Proceedings of 2012 SC

Companion: High Performance Computing, Networking Storage and Analysis. Salt Lake City, UT, USA. 2012. 821–826.

10 Meredith JS, Ahern S, Pugmire D, *et al*. EAVL: The extreme-scale analysis and visualization library. The Eurographics Association. 2012. 21–30. [doi: 10.2312/EGPGV/EGPGV12/021-030]

11 Moreland K, Ayachit U, Geveci B, *et al*. Dax toolkit: A proposed framework for data analysis and visualization at extreme scale. Proceedings of 2011 IEEE Symposium on Large Data Analysis and Visualization. Providence, RI, USA. 2011. 97–104.

12 Sisneros R, Pugmire D. Tuned to terrible: A study of parallel particle advection state of the practice. Pugmire 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). Chicago, IL, USA. 2016. 1058–1067.

13 Peterka T, Ross R, Nouanesengsy B, *et al*. A study of parallel particle tracing for steady-state and time-varying flow fields. Proceedings of 2011 IEEE International Parallel & Distributed Processing Symposium. Anchorage, LA, USA. 2011. 580–591.

14 Zhang J, Guo HQ, Hong F, *et al*. Dynamic load balancing based on constrained K-D tree decomposition for parallel particle tracing. IEEE Transactions on Visualization and Computer Graphics, 2018, 24(1): 954–963. [doi: 10.1109/TVCG.2017.2744059]

15 Fu YW, Yu CP, Yan Z, *et al*. The effects of combustion on turbulent statistics in a supersonic turbulent Jet. Advances in Applied Mathematics and Mechanics, 2019, 11(3): 664–674. [doi: 10.4208/aamm.2018.s10]

16 Fu YW, Yu CP, Yan Z, *et al*. DNS analysis of the effects of combustion on turbulence in a supersonic H2/Air jet flow. Aerospace Science and Technology, 2019, 93: 105362. [doi: 10.1016/j.ast.2019.105362]

17 Larsen M, Ahrens J, Ayachit U, *et al*. The ALPINE in situ infrastructure: Ascending from the ashes of strawman. Proceedings of the in Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization. Denver, CO, USA. 2017. 42–46.