计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20131332
52(4):823-832, 2015

千万亿次可扩展可容错自由网格数值模拟系统

黎雷生[1,2] 王朝尉[1] 马志涛[1] 霍志刚[1] 田 荣[1]
[1](中国科学院计算技术研究所高性能计算机研究中心 北京 100190)
[2](中国科学院大学 北京 100049)
(lileisheng@ncic.ac.cn)

petaPar: A Scalable and Fault Tolerant Petascale Free Mesh Simulation System

Li Leisheng[1,2], Wang Chaowei[1], Ma Zhitao[1], Huo Zhigang[1], and Tian Rong[1]
[1](*High Performance Computer Research Center*, *Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract With the emergence of petaflops (10^{15} FLOPS) systems, numerical simulation has entered a new era—a times opening a possibility of using 10^4 to 10^6 processor cores in one single run of parallel computing. In order to take full advantages of the powerfulness of the petaflops and post-petaflops supercomputing infrastructures, two aspects of grand challenges including the scalability and the fault tolerance must be addressed in a domain application. petaPar is a highly scalable and fault tolerant meshfree/particle simulation code dedicated to petascale computing. Two popular particle methods, smoothed particle hydrodynamics (SPH) and material point method (MPM), are implemented in a unified object-oriented framework. The parallelization of both SPH and MPM consistently starts from the domain decomposition of a regular background grid. The scalability of the code is assured by fully overlapping the inter-MPI process communication with computation and a dynamic load balance strategy. petaPar supports both flat MPI and MPI+Pthreads hierarchial parallelization. Application-specific lightweight checkpointing is used in petaPar to deal with the issue of fault tolerance. petaPar is designed to be able to automatically self-restart from any number of MPI processes, allow a dynamic change of computing resources arisen in a scenario of, for example, nodal failure and connection timeout etc. Experiments are performed on the Titan petaflops supercomputer. It is shown that petaPar linearly scales up to 2.6×10^5 CPU cores with the excellent parallel efficiency of 100% and 96% for the multithreaded SPH and the multithreaded MPM, respectively, and the performance of the multithreaded SPH is improved by up to 30% compared with the flat MPI implementation.

Key words petascale computing; meshless/particle simulation; high scalable; fault tolerance; MPI+Pthreads; dynamic load balancing

摘 要 在千万亿次计算能力的驱动下,数值软件的发展进入了一个以海量并行为基本特征的历史转折期,可扩展和可容错成为大规模数值模拟的两大关键技术.petaPar模拟程序是以对传统数值技术形成优势互补的无网格类方法为切入点,面向千万亿次级计算而开发的下一代新兴通用数值模拟程序.petaPar在统一架构下实现了光滑粒子动力学(smoothed particle hydrodynamics, SPH)和物质点法

收稿日期:2013-10-12;**修回日期**:2014-06-12
基金项目:国家自然科学基金项目(11072241,11111140020,91130026);橡树岭国家实验室/美国国家计算科学中心主任基金项目(MAT028)
通信作者:田 荣(rongtian@ncic.ac.cn)

(material point method, MPM)两种最为成熟和有效的无网格/粒子算法,支持多种强度、失效模型和状态方程;其中 MPM 支持改进的接触算法,可以处理上百万离散物体的非连续变形和相互作用计算. 系统具有以下特点:1)高可扩展. 实现单核单 Patch 极端情形下计算和通信的完全重叠,支持动态负载均衡;2)可容错. 支持无人值守变进程重启动,在系统硬件出现局部热故障时可以不中止计算;3)适应硬件体系结构异构架构的变化趋势,同时支持 flat MPI 和 MPI+Pthreads 并行模型. 程序在 Titan 千万亿次超级计算机上进行了全系统规模的可扩展性测试,结果表明该代码可线性扩展到 26 万个 CPU 核,SPH 和 MPM 的并行效率分别为 100%和 96%.

关键词 千万亿次计算;无网格/粒子模拟;高可扩展;高可容错;多线程;动态负载平衡

中图法分类号 TP301.6; TP338.6

我国千万亿次机(天河、曙光星云以及神威)的发展取得了世界瞩目的成就. 以此为标志,我国高性能计算正式迈入千万亿次时代.

在千万亿次计算能力的驱动下,数值软件的发展进入了一个以海量并行为基本特征的历史转折期. 过去计算机处理器速度的发展一直遵循着摩尔定律:程序会随着处理器频率的提高而自然地变快(即所谓的"免费的午餐"). 然而,这一发展趋势由于受到功耗等物理限制而难以为继,转而通过多/众核技术和片内并行性来提高性能. 而市场上现有的国外商用数值软件,由于大多诞生于 20 世纪 70 年代到 80 年代,一个以 PC 为主的串行时代,原始软件架构设计中没有、也不可能考虑大规模并行的特点和需求. 这导致目前的数值软件,特别是商用数值软件,与海量并行计算的架构性矛盾将日益突出,甚至面临重新设计的可能.

为了发挥千万亿次以及"后"千万亿次超级计算机强大的计算能力,应用程序必须突破 2 大关键技术:可扩展性和可容错性. 1)为了有效发挥千万亿次超级计算机的计算能力,应用代码需要支持数万至上百万处理器核的高可扩展计算①. 2)随着超级计算机规模的扩大,系统出现故障,特别是互联等"热"故障的概率不断增加. 对于目前的千万亿次超级计算机,系统平均无故障运行时间以天计. 随着系统规模的进一步扩大,这一时间将继续缩短,甚至以小时计. 对于一个千万亿次级及更大规模的应用程序,必须能够容忍硬件经常性的故障才能最终完成生产性计算任务. 这是一个过去并行程序开发者完全无需考虑的、但却成为未来大规模科学与工程计算非常关键的技术.

20 世纪 90 年代初出现的无网格方法,因其新颖的学术思想,一经诞生就吸引了人们的广泛关注. 目前,这类新一代数值技术正处于实用化研究阶段,商用软件正逐渐、有选择地采纳无网格计算技术. 大浪淘沙,如今最能体现无网格技术生命力的特点当属能够有效模拟固体材料大变形和在拉格朗日格式下统一求解固体、流体力学、固体变形与流体流动耦合问题. 目前成功解决高速碰撞等大变形固体力学问题的代表性方法有光滑粒子动力学方法(smoothed particle hydrodynamics, SPH)和物质点法(material point method, MPM). 相对于传统基于网格的方法,这些无网格粒子方法在极限力学问题以及流固耦合方面具有先天优势.

本文采用"自由网格"这一术语,以区别于传统意义上的无网格思想. 随着对无网格技术研究的深入,人们也越来越深刻地认识到,在计算机辅助工程(computer aided engineering, CAE)世界中的诸多方面,如自由界面描述、空间拓扑关系确定、动解析中的质量计算、弱形式的数值积分计算、大规模问题的可视化后处理等等,网格的作用仍然是必要的、甚至是无法取代的. 实际上,经过了对无网格技术的初始热捧阶段之后,学术界也开始重新认识网格与无网格的本质区别. 虽然表面上看拉格朗日型有限元的绝大部分困难似乎毫无例外地与"网格"相关,如网格生成、网格扭曲、网格重新划分等;然而,从本质上看,"网格"并非真正症结所在,真正的症结应该在于有限元固定不变的"节点拓扑连接关系". 同理,无网格方法的精髓也并不在于一定要抛弃网格,而应在于"动态可变的节点间拓扑连接关系". 相反,对于有限元方法,无网格插值放松了对网格几何连续协调性的要求会使得网格生成和使用变得更加灵活方便,例如负雅克比单元、任意网格组合、重叠网格等有限元无法直接使用的"坏"网格都可以在无网格计算中正常使用,所以关键是如何实现"网格离散"和

① 目前的大多数商业数值软件的可扩展性一般在数十至数千处理器核规模.

"无网格插值"的灵活结合;"自由网格"同时强调了网格在前后处理方面的固有重要性和无网格插值的灵活性.

petaPar 模拟程序是伴随国产曙光千万亿次超级计算机的研发以对传统数值技术形成优势互补的无网格类方法为切入点面向千万亿次级计算而开发的下一代新兴通用数值模拟程序. petaPar 在统一架构下实现了光滑粒子动力学 SPH 和物质点法 MPM 两种最为成熟和有效的无网格/粒子方法,支持多种强度、失效模型和状态方程,其中 MPM 还支持改进的接触算法,可以处理上百万离散系统的非连续变形和相互作用计算. 该系统具有以下 3 个特点:

1) 高可扩展. 实现单核单个块单元(Patch)极端情形下计算和通信的完全重叠,支持动态负载均衡;

2) 可容错. 支持无人值守变进程重启,在系统硬件出现局部故障(如个别计算节点失效或由于网络互连等原因不可用等)时可以不中止计算;

3) 适应硬件体系结构异构架构的变化趋势,同时支持 flat MPI 和 MPI+Pthreads 并行模型.

1 SPH 和 MPM

SPH 方法由 Lucy,Gingold 和 Monaghan[1-2] 提出,用于模拟天体物理学问题. Libersky 和 Petschek 等人[3]将 SPH 方法扩展到固体力学领域,引入了材料强度模型,模拟了动态断裂和破片运动等材料的动态响应. SPH 方法广泛用于天体物理学、流体动力学、固体力学、水下爆炸和流固耦合等问题. 国内学者刘谋斌[4]研究员做了大量工作.

MPM 是一种混合拉格朗日-欧拉方法(mixed Lagrangian-Eulerian method),由 Sulsky 等人[5-6]提出. MPM 中的接触处理算法适合处理接触问题,Ma 等人对接触算法进行改进[7]. MPM 方法在碰撞冲击、侵彻、爆炸、动态断裂、流固耦合、颗粒材料流动和岩土失效等方面得到广泛应用.

国内张雄教授领导的研究组对 MPM 算法作了大量研究,包括 MPM 的接触算法、自适应算法以及与 FEM 的耦合等,利用 MPM 研究高速碰撞、金属切削和冲击爆炸等问题,并且研究 MPM 粒子建模方法. 其研究组开发的 MPM3D 代码实现了 MPM 算法的各个方面,利用 OpenMP(open multi-processing)和消息传递接口(message passing interface, MPI)实现并行[8]. 张洪武等人[9]提出多相 MPM 模拟饱和多孔介质动力学问题.

SPH 和 MPM 与其他各种自由网格/粒子方法在并行实现上有很多类似之处. 葛蔚等人[10]利用分子动力学研究多相流体模拟、SPH 模拟流体等,并利用 GPU 进行加速. 曹小林、莫则尧等人[11]开发的 JASMIN 框架实现了分子动力学、PIC 等粒子方法,并行规模达到 36 000 核.

本文重点介绍 SPH 和 MPM 并行算法与 petaPar 系统的可扩展性和可容错特点,基本理论和算法请参考文献[4]和文献[8].

2 SPH 和 MPM 的并行计算和通信

SPH 和 MPM 算法的并行计算的 2 个主要数据结构是格子单元(Cell)和块单元(Patch).

2.1 Cell

无网格类方法与基于网格的方法的最大、最本质的区别,如引言所述,是"动态的"拓扑邻接关系. 这种"动态"具体表现在计算时每一时间步都需要重新建立粒子间的邻接关系. 为了建立和管理这种邻接关系、实现相邻关系的快速搜索,引入 Cell 结构.

Cell 是求解域的外接矩形(2D)或长方体(3D)上的一个结构化网格单元. Cell 形成对粒子或物质点的一个空间划分,以实现对粒子的管理和快速查找.

图 1 说明 SPH 中 Cell 的结构. SPH 算法利用 Cell 加快相邻粒子的搜索,搜索算法复杂度从 $O(N^2)$ 降低到 $O(NM)$,其中 N 是计算域中粒子的总数,M 是 Cell 中的平均粒子数.

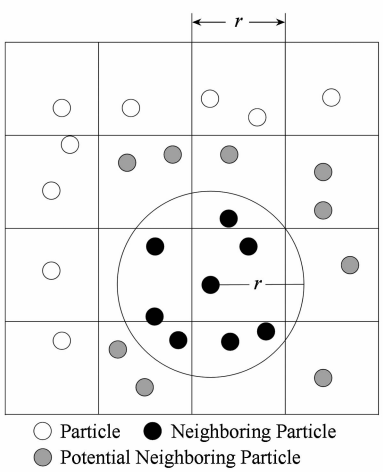


Fig. 1 Cell of SPH.

图 1 SPH 中 Cell 的结构

图 2 说明 MPM 算法的 Cell 和节点以及它们之间的相互映射关系. 与 SPH 不同的是,粒子的属性

值映射到节点,在节点上求解动量方程,然后节点数据映射回粒子进行材料状态和构型的更新.

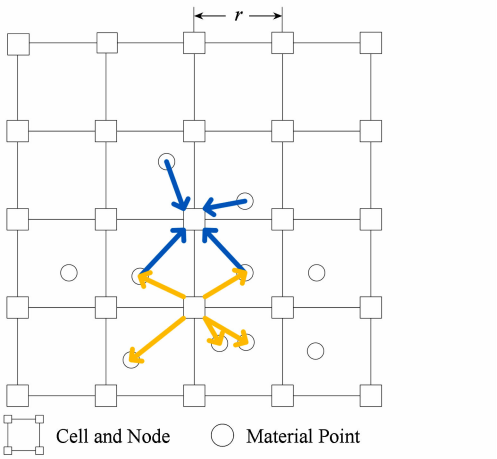


Fig. 2 Cell and node of MPM.

图 2 MPM 中的 Cell 和节点

Cell 大小或边长是一个关键参数. 为了便于描述,对 SPH 和 MPM 统一引入"粒子影响半径"的概念,记为 r. r 为与"相邻粒子平均间距"相关的一个输入参数. 无论是 SPH 还是 MPM,Cell 边长取为"粒子影响半径"的长度,即 r.

需要指出,无论 SPH 还是 MPM,其 Cell 边长的选取都应保证材料域内的 Cell 分别包含足够数量的粒子或物质点,以降低离散误差并分别保证插值精度或数值积分精度. 这一点对于这类粒子方法的计算稳定性非常关键. 粒子太少会导致计算结果失真(计算可以进行但结果无意义)和发散.

2.2 Patch 划分及邻接关系

三维问题中,Patch 定义为由 $n_x \times n_y \times n_z$ 个紧邻 Cell 组成的规则区域. 根据粒子或物质点的坐标,可确定其所属的 Cell 和 Patch. Patch 是数据并行的基本单位.

求解域被划分成若干 Patch,Patch 的邻接关系依据 Patch 的空间位置来建立. Patch 具有结构化特征,Patch 的邻接关系可以显式计算. 三维中把 Patch 按照 x,y,z 方向依次编号,使用 (i,j,k) 对 Patch 进行索引定位,使用 I,J,K 分别表示 x,y,z 方向上 Patch 的数量,由此计算出每个 Patch 的唯一标识 id. 计算方法:$id=i+j\times I+k\times I\times J$;反过来 id 可以计算 Patch 的索引:$k=id/(I\times J)$,$j=(id-k\times I\times J)/I$,$i=id-k\times I\times J-j\times I$. 每个 Patch 最多有 26 个相邻 Patch:面相邻最多 6 个,边相邻最多 12 个,顶点相邻最多 8 个,这些相邻 Patch 可以通过 Patch 的 (i,j,k) 索引快速定位.

2.3 Patch 间数据交换

每个 Patch 的数据具有独立性,只有 Patch 边界区域与邻接 Patch 存在数据依赖关系. 并行计算时,相邻 Patch 进行数据交换获取完整的计算数据. 为了描述 Patch 间数据交换,引入内部 Cell 和边界 Cell 的概念:

1) 内部 Cell. 与其他 Patch 中的 Cell 没有邻接关系,Cell 内粒子计算不依赖当前时间步的其他 Patch 的数据.

2) 边界 Cell. 与其他 Patch 的 Cell 相邻,Cell 内粒子计算依赖于当前时间步的其他 Patch 的数据.

类似地,把 MPM 中的节点分为内部节点和边界节点:

1) 内部节点. 内部 Cell 的节点,节点处的计算与其他 Patch 的物质点无关.

2) 边界节点. 边界 Cell 节点中被多个 Patch 共享的部分. 边界节点处的计算与相邻 Patch 边界 Cell 上的物质点相关.

可以看出,Patch 为了获得完整的数据,SPH 算法中相邻 Patch 需要交换 Patch 边界 Cell 中的粒子数据,MPM 算法中相邻 Patch 需要交换边界节点上的数据. 由于 petaPar 中的 MPM 实现了接触算法,在可能接触的边界节点上,与之相接的边界 Cell 中的粒子位置和体积属性也需要进行交换.

为了交换边界数据,在 Patch 周围增加 1 层 Cell 作为相邻 Patch 的边界 Cell 内粒子数据的复制,称之为 Ghost Cell. 数据交换的特点是,每个 Patch 用自己边界 Cell 的数据更新邻接 Patch 的 Ghost Cell. Ghost Cell 中的粒子只用于辅助计算,不作自身状态更新,在内存操作上为只读.

MPM 算法中边界节点被多个相邻 Patch 共享每个 Patch 的边界节点数据都是不完整的. 数据交换的特点是聚合操作,内存数据操作为读和写.

2.4 Patch 间粒子迁移

无网格粒子法的一个主要特点是粒子在计算过程中不断移动而 Cell 结构是欧拉的. 当粒子跨越 Patch 边界移动时,需要将粒子迁移到目的 Patch. Patch 间粒子迁移是一种特殊的 Patch 间数据交互.

在显格式计算中,每一步计算符合小变形假设. 如果 Cell 大小与相邻粒子平均间距相当的话,一个时间步内粒子不会发生跨越 Cell 的大距离移动,粒子迁移的基本模式是离开或进入边界 Cell. 需要迁移的粒子,会首先进入 Patch 的 Ghost Cell 区,然后在下一时间步计算开始前把进入 Ghost Cell 的粒子

数据通信到目的 Patch,完成粒子数据迁移. SPH 和 MPM 粒子迁移方式相同.

3 计算通信重叠

计算通信重叠是并行程序优化的关键手段. CPU 负责发起通信和处理接收到的数据,网络接口设备(network interface controller, NIC)负责数据的发送和接收等底层工作. NIC 处理通信任务的时候,CPU 可以同时执行与通信无关的计算.

SPH 和 MPM 每个时间步的总时间 T 表示为

$$T = T_c + T_t,$$

其中,T_c 是计算时间,T_t 是通信时间.

为了讨论计算通信重叠,T_c 表示为

$$T_c = T_i + T_d,$$

其中,T_i 是不依赖通信可以独立计算的计算时间,T_d 是依赖通信的计算时间.

计算通信重叠的目标是尽量保证 T_i 和 T_t 同时分别占用 CPU 和 NIC,使得计算与通信并发进行,减少总时间 T,提升并行性能和效率. 实现计算通信重叠的关键是找出 T_i 对应的计算部分.

为了便于描述 SPH 的计算通信重叠的算法,定义内部 Cell 的最外层 Cell 为第 2 边界 Cell,定义除去第 2 边界 Cell 的内部 Cell 为独立 Cell. 粒子迁移和 Ghost Cell 更新操作涉及到边界 Cell 层. 在粒子迁移之前,边界 Cell 数据不完整. 由于第 2 边界 Cell 中的粒子计算时需要边界 Cell 粒子的数据,所以只有独立 Cell 上的计算与粒子迁移以及 Ghost Cell 更新完全无关. SPH 采用的重叠策略是用部分独立 Cell 上的计算来重叠粒子迁移通信,另一部分独立 Cell 和第 2 边界 Cell 重叠 Ghost Cell 更新通信的开销.

对于 MPM 并行计算,每个时间步内 Patch 上发生 4 次数据交换:边界节点聚合、接触数据交换(如果有必要)、边界节点动量聚合和粒子迁移. 同样地,Patch 分为内部 Cell 和边界 Cell. Patch 边界节点的数据只和边界 Cell 中的物质点相关.

4 次数据交换的重叠策略如下:

1) 边界节点聚合. 通过内部节点的计算进行重叠. 具体地,首先计算边界节点,接着发起非阻塞通信,数据传输的同时执行内部节点计算.

2) 接触数据交换. 首先发起非阻塞通信,数据传输的同时进行内部节点的接触计算,通信完成后执行边界节点的接触计算.

3) 边界节点动量聚合. 首先计算边界节点动量,接着发起非阻塞通信,数据传输的同时进行内部节点的动量计算.

4) 粒子迁移. 首先更新边界 Cell 中的物质点的状态,接着发起非阻塞通信,数据传输的同时更新内部 Cell 物质点状态.

MPI 提供的非阻塞通信接口 MPI_Irecv,MPI_Isend,MPI_Test,MPI_Wait 等可以用于实现计算通信重叠. 研究表明,应用基于不同 MPI 实现(如 OpenMPI,MPICH 等)计算通信重叠的效果不同[12],原因在于它们处理点到点数据传输的方式不同. MPI 点到点消息传输有 2 种方式:Eager 协议适合传输"小"消息,发送端直接发送,接收端预留缓冲区存储接收的消息;Rendezvous 协议适合传输"大"消息,发送端和接收端协商数据的传输,发送端首先发送 RTS(request to send)消息给接收端,接收端发送 CTS(clear to send)消息给发送端通知其开始传输数据."小"消息一般是小于数十 KB 的消息,其他为"大"消息. CPU 执行计算密集型的科学计算时,有些 MPI 实现不能及时处理 RTS 或 CTS 消息,导致数据传输延迟,不能实现通信重叠计算的效果. petaPar 使用在执行计算过程中调用 MPI_Test 的策略解决这一问题,及时处理 RTS 和 CTS 消息,保证数据传输与计算同时进行.

4 动态负载平衡

负载平衡是并行系统充分发挥效能的关键[13]. SPH 和 MPM 算法的计算负载与粒子数近似成正比. 负载平衡等价于每个进程上的粒子数相同. 计算过程中粒子在 Patch 间迁移,每个进程的负载动态变化,保证进程间负载平衡是无网格粒子可扩展并行计算的一个难点.

petaPar 使用图分解算法实现负载平衡. 每个 Patch 作为图的一个顶点,Patch 中的粒子数量为顶点的权重,空间中所有 Patch 形成一个加权图. 利用图分解算法把 Patch 分配到各个计算进程. petaPar 图分解基于 parMetis 库实现[14].

为了实现动态负载平衡,定义进程最大粒子数与进程平均粒子数之比为进程平衡因子,1 是理想平衡状态. 平衡因子越大,负载越不平衡. 计算过程中当平衡因子偏离到一定程度时,重新执行图分解,依据新的图分解结果对进程数据进行增量式的调整.

5 MPI+Pthreads 层次并行模式

为了适应超级计算机的多核以及异构架构,在量级上减少 MPI 进程的数量和管理开销[15],petaPar 实现了 MPI+X 的混合并行.在本文,X 为 Pthreads,支持多核 CPU 和 GPU 上的多线程计算.为了实现多线程模式下的通信重叠,使用层次化并行方案解决计算与通信任务协调的问题.首先,全局负载均衡把 Patch 划分给 MPI 进程;然后 MPI 进程内执行负载平衡,把 Patch 划分给每个线程,每个线程负责计算一部分 Patch.由于进程内 Patch 数量和线程数量相对较少,可以使用最优算法进行负载平衡.从 MPI 层次来看,每个 MPI 进程与多个 MPI 进程存在点对点通信关系,把与单个 MPI 进程的通信作为一个任务,通信任务平均分配到每个线程,每个线程负责若干个通信任务.线程负责发起 MPI_Isend 和 MPI_Irecv,在计算过程中调用 MPI_Test,计算完成后调用 MPI_Wait,这种方式能及时处理 RTS 或 CTS 消息.通过这种方式,多线程情况下也能达到计算完全重叠通信的效果.

通信数据的封装和解封装实现了多线程并行.数据封装以目标进程为单位,一个目标进程的数据封装由一个线程负责,线程从 Patch 读取数据,写入目标进程的存储空间;解封装以 Patch 为单位,每个 Patch 的解封装操作由一个线程执行.封装和解封装都避免了多个线程的写共享冲突.

6 可容错性

随着超级计算机规模不断扩大,系统出现故障的概率不断增加.一个千万亿次级以及更大规模的应用程序,必须能够容忍硬件这种经常性的局部性故障才能最终完成计算.petaPar 实现了用户级检查点功能和自动变进程重启动功能,系统具有良好的容错性和弹性.

6.1 用户级检查点

检查点技术是科学与工程计算的主要容错方式.传统的系统级进程检查点是通过保存所有相关进程的完整状态(称为进程映像)以使得程序能够从中断处恢复执行.系统级检查点由于把应用程序视作一个黑盒子,要保存的检查点数据量很大.目前普遍认为在后千万亿次特别是百亿亿次计算时代,系统级检查点由于数据 I/O 时间可能会大于系统平均无故障时间而不再可行.

petaPar 采用用户级检查点策略,其主要目的在于有效降低检查点的数据量.petaPar 利用粒子模拟算法的特点,把检查点数据限制在最小依赖集,保证计算完全恢复的同时做到数据量最小,降低写入检查点数据和恢复检查点的开销.

以 MPM 算法为例,每个粒子有 185 B 的检查点数据(不同材料稍有差异),262 144 个粒子的检查点数据量仅为 46 MB,与相应的系统级检查点(进程映像)的数据量 200 MB 相比,数据量大大降低.

每个 Patch 的检查点数据与并行计算逻辑无关.任一进程都可以读取任一 Patch 的检查点数据,重启动的进程和保存检查点数据的进程之间无需任何依赖关系,所以在计算资源和进程数发生改变的情况下仍然可以重启动.

除容错功能之外,petaPar 的检查点功能还可以用于程序的调试和数值模拟参数的调整(如多时间尺度模拟的手工实现等).

6.2 自动变进程重启动

系统通过支持无人值守变进程重启动功能实现了高可容错和弹性.在计算机系统硬件出现局部故障,如个别计算节点失效或由于网络互连等原因不可用等情况下,可以不中断计算.

为了实现自动变进程重启动,首先需要对计算资源和应用进行监控,发生异常后根据设置的策略进行重启动.自动变进程重启动模块包括全局控制模块 gctrl、gctrl 命令工具 gctrl_cmd、节点监控模块 dcrd 和面向应用的接口库 libdcr.系统中只有 1 个 gctrl 进程,功能包括启动节点上的 dcrd、接受 dcrd 的注册和各种请求、实时监测 dcrd 的运行、发起重启动命令等.gctrl_cmd 负责设置和清除重启动策略、主动发起重启动等.每个计算节点有 1 个 dcrd,负责接受本地应用进程的注册、监测进程的状态、应答 gctrl 的查询等.libdcr 库向应用提供初始化和终止接口.

应用启动时调用 libdcr 的初始化函数向 dcrd 注册,正常退出时调用终止函数通知 dcrd 应用正常结束.应用异常退出时,由于没有调用终止函数,dcrd 可以检测到异常情况,通知 gctrl 重启动应用.自动重启动功能模块独立,除用于 petaPar 系统外,也可用于基本的计算资源管理.

7 实验与结果

程序的正确性已经进行了大量的验证.限于篇幅,本文侧重报告 petaPar 程序在千万亿次超级计算

机上进行大规模数值模拟时的计算通信重叠、动态负载均衡、可容错性和可扩展性等关键技术上的突破.

7.1 计算通信重叠效果测试

使用纯 MPI 版本验证计算通信重叠的效果,验证模型是两个物体高速碰撞.使用 8 台 8 核服务器集群进行测试,测试中限制每核 1 个 Patch(每个 Cell 中 8 个粒子),这一情形为通信计算比最大的极端情形,如果这种情况下能达到计算完全重叠通信时间,那么其他情况下也一定能够达到重叠.

MPM 算法测试结果如图 3 所示,可以看出:Patch 尺寸越大重叠效果越好,Patch 尺寸大于等于 16×16×16 时计算时间能够完全重叠通信时间.

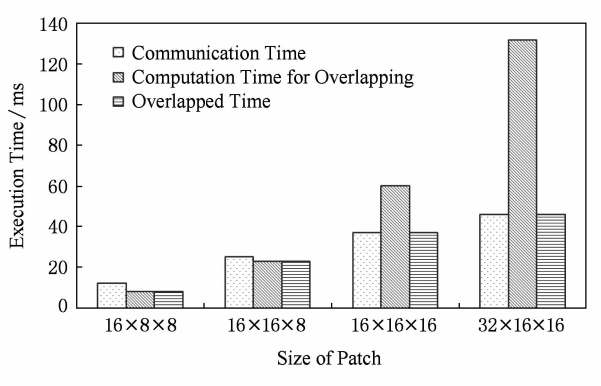


Fig. 3 Results of MPM overlapping.

图 3 MPM 计算重叠通信效果

SPH 算法测试结果如图 4 所示,SPH 相对于 MPM 通信量较大,其通信计算比大于 MPM. Patch 尺寸在 16×16×16 时 SPH 的计算时间不能完全重叠通信时间;随着 Patch 的尺寸增大计算时间和通信时间都增长,但通信时间增加较慢;当 Patch 增大到32×32×16 时,SPH 算法的计算时间可以完全重叠通信时间.

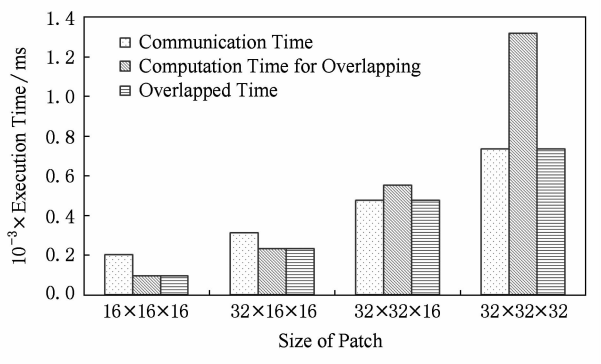


Fig. 4 Results of SPH overlapping.

图 4 SPH 计算重叠通信效果

7.2 动态负载均衡测试

以弹体打靶为例,模型共有 32×32×32 个 Patch,通过 Patch 把粒子分配到 128 个进程,分解后的进程平衡因子为 1.05.模型 Patch 间两两邻接关系数是 797 816,而进程间两两通信关系数仅为 4 688,进程通信连接数只有 Patch 邻接关系数的 0.6%,说明数据的局部性非常好.负载平衡算法执行时间约 300 ms,由于平衡算法几百个时间步或更多时间步执行一次,其占用的时间可以忽略不计.

7.3 可容错性实验

petaPar 系统的基本特点是可以做到在程序出现异常终止或出现硬件局部故障时无需人工干预,即可恢复正常运行.下面在 4 节点集群上进行测试.这里的硬件故障通过 MPI 进程的人为异常终止进行模拟.测试过程如下:

1) 启动控制模块 gctrl. gctrl 负责启动计算节点上的 dcrd,每个节点的 dcrd 都与 gctrl 连接.

gctrl-f ~/restart/host

屏幕提示输出为:

dcrd connect:host=blade01,cores=12

dcrd connect:host=blade02,cores=12

dcrd connect:host=blade04,cores=12

dcrd connect:host=blade03,cores=12

2) 使用配置工具 gctrl_cmd 向 gctrl 注册应用程序的重启命令.

gctrl_cmd-f./restart

gctrl 接收命令并提示:

restart command:

/home/lils/petaPar: ~/mpich/bin/mpiexec ::/home/lils/petaPar/petaPar mpm./input.ict./input

3) 启动 petaPar. petaPar 通过调用 libdcr 库函数向 dcrd 注册应用程序,dcrd 把 petaPar 的信息通知 gctrl. 首先使用 2 个节点运行.

启动 petaPar(MPM 算法):

nohup mpiexec-f host-np 16./petaPar mpm./input.ict&

gctrl 提示:

app connect:petaPar

app host:nth=0,host=blade01

app host:nth=1,host=blade02

4) 手工中止 petaPar,模拟应用程序的意外终止.此时 gctrl 监测到 petaPar 的终止,然后根据重启动策略立即使用所有可用计算资源自动执行重启动,说明容错系统可以应对应用异常终止的情况.

中止 petaPar:

killall petaPar

gctrl 自动执行重启动:

broadcast to kill app processes

restart app now

restart: cd/home/lils/petaPar && nohup ~/mpich/bin/mpiexec-f/tmp/. dcrd3. hosts-np 48/home/lils/petaPar/petaPar mpm input. ict input 2>&1>>/dev/null&

app connect:petaPar

app host:nth=1,host=blade02

app host:nth=3,host=blade03

app host:nth=0,host=blade01

app host:nth=2,host=blade04

5) 中止一个节点的 dcrd,模拟计算节点的失效. gctrl 监测到节点 dcrd 失去连接,使用剩余的节点重启动 petaPar. 此时,gctrl 使用 3 个节点 36 核重启动.

中止 dcrd:

killall dcrd

gctrl 提示使用 36 核进行了自动重启动:

blade01 loss connection

broadcast to kill app process

restart app now

restart: cd/home/lils/petaPar && nohup ~/mpich/bin/mpiexec-f/tmp/. dcrd3. hosts-np 36/home/lils/petaPar/petaPar mpm input. ict input 2>&1>>/dev/null&

app connect:petaPar

app host:nth=1,host=blade03

app host:nth=3,host=blade04

app host:nth=2,host=blade02

6) 重新启动节点的 dcrd,模拟有新的计算节点加入. 此时 dcrd 会通知 gctrl 有新的计算资源加入,然后重新启动 petaPar,以加速计算.

重新启动 dcrd,并发送重启动 petaPar 命令:

dcrd-g blade01

gctrl_cmd-r petaPar

restart command for petaPar

gctrl 提示使用了 4 个节点、共 48 核进行了重启动:

dcrd connect:host=blade01,cores=12

broadcast to kill app processes

restart app now

restart: cd/home/lils/petaPar && nohup ~/mpich/bin/mpiexec-f/tmp/. dcrd3. hosts-np 48/home/lils/petaPar/petaPar mpm input. ict input 2>&1>>/dev/null&

app connect:petaPar

app host:nth=1,host=blade02

app host:nth=3,host=blade01

app host:nth=2,host=blade03

app host:nth=0,host=blade04

整个过程中,petaPar 运行速度会受到可用计算资源变化的影响,但计算可以在无人值守和无人工干预的情况下不中断执行,从而实现可容错计算.

7.4 可扩展性实验

实验在 Titan 上完成,在该测试工作进行时,该系统为世界最快、可用处理器核数最多的超级计算机. 进行测试时,Titan 全部系统拥有 8 972 个 CPU 节点和 9 716 个 CPU+GPU 节点. 每个节点为 16 核 CPU,以非一致内存访问(non-uniform memory access architecture, NUMA)架构组织,每个 NUMA 有 8 个核心,每个节点占 32 GB 内存,节点间使用 Cray 公司定制的 3D Torus 拓扑高速网络连接. 由于 Titan 正在建设中,petaPar 最多使用 16 384 个节点,即 262 144 个 CPU 核心[16].

计算模型为 2 个物体高速对撞,物体接触面粒子均有接触作用,模型包含 85 亿粒子. Cell 和 Patch 配置如表 1. 其中,262 144 核规模时,每个处理器核仅负责计算 1 个 Patch. 这样做的目的是测试程序在模型数据最小的情况下是否具有很好的强可扩展性. 并行程序如果在最小模型数据具有好的强可扩展性的条件下,则在增加模型数据时其加速效果会更好.

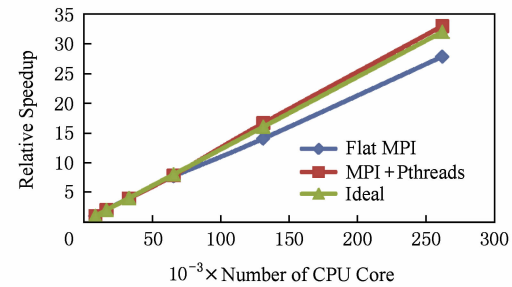
Table 1 Configuration of Model

表 1 模型配置

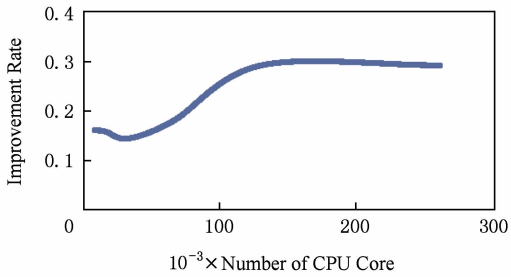
Items	Configurations
Particle	8589934592
Cell	1024×1024×1024
Particle per Cell	2×2×2
Patch	64×64×64
Cell per Patch	16×16×16

可扩展性实验分别针对 8 192, 16 384, 32 768, 65 536, 131 072 和 262 144 核. 强可扩展性曲线如图 5 和图 6 所示. SPH 和 MPM 的纯 MPI 版本,262 144 核规模相比 8 192 核并行效率分别达到 87% 和 89%. MPI+Pthreads 多线程版本 SPH 和 MPM 的并行效率分别达到 100% 和 96%. 其中,SPH 的

多线程版本比纯 MPI 性能提升高达 30%;MPM 多线程版本在核数较少时性能比纯 MPI 版本稍低,随着核数增多多线程 MPM 性能超过纯 MPI 版本.



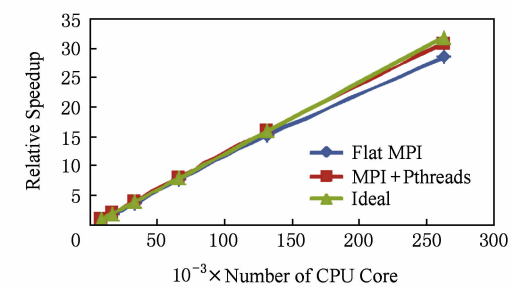
(a) Strong scalability



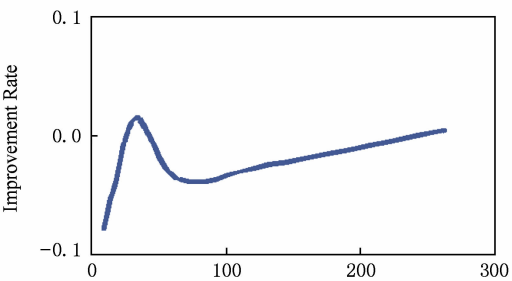
(b) MPI+Pthreads performance improvement

Fig. 5 The strong scalability and the performance improvement of SPH.

图 5 SPH 强可扩展和性能提升



(a) Strong scalability



(b) MPI+Pthreads performance improvement

Fig. 6 The strong scalability and the performance improvement of MPM.

图 6 MPM 强可扩展和性能提升

SPH 的 MPI+Pthreads 版本并行效率达到 100%和性能提升的原因在于减少了 MPI 进程间的通信边界,降低了处理边界数据的开销.可扩展性验证时,MPI+Pthreads 方式每个 MPI 进程使用 4 个线程,262 144 核时每个 MPI 进程包含 4 个 x,y 方向上空间连续的 Patch,Patch 的 2 个面边界数据不需要进程间的通信.而纯 MPI 每个进程包含 1 个 Patch,其 6 个面都需要进行进程间通信.对于 SPH 来说,MPI+Pthreads 方式进程间通信量相比纯 MPI 情况降低了 1/3,整体性能提升约 30%,MPI 进程数增加时通信减少带来的性能提升越有优势,所以可以达到超线性的加速.

MPM 算法中每个 16×16×16 尺寸的 Patch 通信量较小,16×16×16 尺寸的 Patch 在纯 MPI 情况下计算时间可以完全重叠通信时间,性能已经比较优化,相比之下多线程方式对其性能提升不明显.核数规模较小时其性能反而比纯 MPI 版本低一些,原因在于多线程降低通信量带来的性能提升不足以抵消多线程增加的开销,如线程切换、线程同步等,但随着核数的增多可以显现出多线程的优势,说明多线程适合更大规模的扩展.

8 结 论

伴随千万亿次超级计算机的出现,可扩展和可容错成为并行计算的 2 大关键技术.本文以无网格/粒子类方法中最为成熟和流行的算法为基础,面向千万亿次超级计算机的有效使用,开发了程序 petaPar.该系统具有高可容错和高可扩展的特点,并且可以在计算资源动态变化的环境下实现无人值守自动变进程重启,从而实现应用级容错.测试表明,程序可以实现 Titan 千万亿次超级计算机全系统规模的可扩展计算,并行效率达 96%以上.

petaPar 在并行度、解题规模和计算能力上大大突破了目前现有商业软件的限制,可望应用于爆炸以及高速碰撞等极限力学问题、离散系统的非连续变形、流-固-材料破坏的耦合等问题的全三维、高分辨率计算.目前正在进行的工作包括:1)全三维、高分辨率计算和应用;2)开发 GPU CUDA 版本和 Intel MIC 架构的版本.

参 考 文 献

[1] Lucy L B. A numerical approach to the testing of fission hypothesis [J]. The Astronomical Journal, 1977, 82(12): 1013-1024

[2] Gingold R A, Monaghan J J. Smoothed particle hydrodynamics: Theory and application to non-spherical stars [J]. Monthly Notices of the Royal Astronomical Society, 1977, 181(11): 375-389

[3] Libersky L D, Petschek A G, Carney T C, et al. High strain lagrangian hydrodynamics: A three dimensional SPH code for dynamic material response [J]. Journal of Computational Physics, 1993, 109(1): 67-75

[4] Liu Moubin, Liu Guirong. Smoothed particle hydrodynamics (SPH): An overview and recent developments [J]. Archives of Computational Methods in Engineering, 2010, 17(1): 25-76

[5] Sulsky D, Chen Z, Schreyer H. A particle method for history dependent materials [J]. Computer Methods in Applied Mechanics and Engineering, 1994, 118(1): 179-196

[6] Sulsky D, Zhou S J, Schreyer H L. Application of a particle-in-cell method to solid mechanics [J]. Computer Physics Communications, 1995, 87(2): 236-252

[7] Ma Zhitao, Zhang Xiong, Huang Peng. An object-oriented MPM framework for simulation of large deformation and contact of numerous grains [J]. Computer Modeling in Engineering & Sciences, 2010, 55(1): 61-87

[8] Lian Yanping, Zhang Fan, Liu Yan, et al. Material point method and its applications [J]. Advances in Mechanics, 2013, 43(2): 237-264 (in Chinese)
(廉艳平, 张帆, 刘岩, 等. 物质点法的理论和应用[J]. 力学进展, 2013, 43(2): 237-264)

[9] Zhang Hongwu, Wang Kunpeng, Chen Zhen. Material point method for dynamic analysis of saturated porous media (I): Coupling material point method [J]. Chinese Journal of Geotechnical Engineering, 2009, 31 (10): 1505-1011 (in Chinese)
(张洪武, 王鲲鹏, 陈震. 基于物质点方法饱和多孔介质动力学模拟(I): 耦合物质点方法[J]. 岩土工程学报, 2009, 31 (10): 1505-1011)

[10] Zhou Guangzheng, Chen Zhihai, Ge Wei, et al. SPH simulation of oil displacement in cavity-fracture structures [J]. Chemical Engineering Science, 2010, 65(11): 3363-3371

[11] Mo Zeyao, Zhang Aiqing, Cao Xiaolin, et al. JASMIN: A parallel software infrastructure for scientific computing [J]. Frontiers of Computer Science in China, 2010, 4(4): 480-488

[12] Rashti M J, Afsahi A. Assessing the ability of computation/communication overlap and communication progress in modern interconnects [C] //Proc of the 15th Annual IEEE Symp on High-Performance Interconnects. Los Alamitos, CA: IEEE Computer Society, 2007: 117-124

[13] Sun Ninghui, Li Guojie. Load balance of efficient splitting strategy [J]. Journal of Computer Research and Development, 1992, 29(12): 5-12 (in Chinese)
(孙凝晖, 李国杰. 采用有效切分的负载平衡[J]. 计算机研究与发展, 1992, 29(12): 5-12)

[14] Schloegel K, Karypis G, Kumar V. Parallel static and dynamic multi-constraint graph partitioning [J]. Concurrency and Computation: Practice and Experience, 2002, 14(3): 219-240

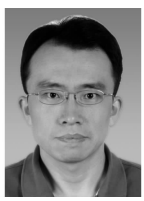
[15] Balaji P, Buntinas D T, Goodell D J, et al. MPI on millions of cores [J]. Parallel Processing Letters, 2011, 21(1): 45-60

[16] Oak Ridge National Laboratory. Titan user guide [EB/OL]. [2013-05-01]. http://www.olcf.ornl.gov/support/system-user-guides/titan-user-guide

Li Leisheng, born in 1981. Master. Student member of China Computer Federation. His main research interests include particle simulation and high performance computing.

Wang Chaowei, born in 1984. Master. His main research interests include numerical analysis and high performance computing (wangchaowei@ncic.ac.cn).

Ma Zhitao, born in 1979. PhD, associate professor. His main research interests include particle simulation and high performance computing (mazhitao@ncic.ac.cn).



Huo Zhigang, born in 1978. PhD and associate professor. His main research interests include opeating system and runtime, parallel computing middleware and fault tolerance(zghuo@ncic.ac.cn).

Tian Rong, born in 1973. PhD, professor. Member of China Computer Federation. His main research interests include generalized FEM, meshfree/particle simulation and high performance computing.