计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2018.20160877
55(2):418-425,2018

针对天河2号的一种嵌套剖分负载平衡算法

刘 旭[1,2] 杨 章[1,2] 杨 扬[2]
[1](计算物理重点实验室(北京应用物理与计算数学研究所) 北京 100094)
[2](北京应用物理与计算数学研究所高性能计算中心 北京 100094)
(liu_xu@iapcm.ac.cn)

A Nested Partitioning Load Balancing Algorithm for Tianhe-2

Liu Xu[1,2], Yang Zhang[1,2], and Yang Yang[2]
[1](*Laboratory of Computational Physics* (*Institute of Applied Physics and Computational Mathematics*), *Beijing* 100094)
[2](*High Performance Computing Center*, *Institute of Applied Physics and Computational Mathematics*, *Beijing* 100094)

Abstract As energy consumption becomes a major design concern of supercomputers, three design trends emerge in supercomputer architectures: massive parallelism, deep memory and network hierarchy, and heterogeneous computing. Large scale computing on such supercomputers as Tianhe-2 requires the load balancing algorithms with three properties: fast, minimal data movement cost, and load balance among heterogeneous devices such as CPU cores and accelerators. On the other hand, multi-physics and multi-scale applications are becoming ubiquitous for many challenge scientific simulations, which results in non-uniform load distribution and demands powerful load balancing algorithms. In this paper, we propose a load balancing algorithm with the above properties by combining a nested partitioning scheme, a greedy partitioning algorithm and an inner-outer subdomain partitioning algorithm. Model experiment shows our algorithm can guarantee good load balance efficiency. Furthermore, experiment on Tianhe-2 with 32 nodes shows our algorithm is able to achieve low communication cost. Finally, experiments of 5 real applications on Tianhe-2 with 936 thousand CPU and MIC cores show that, our algorithm can support large scale simulations efficiently.

Key words parallel computing; load balance; heterogeneous computing; Tianhe-2; MIC

摘 要 天河2号等亿亿次计算机上的大规模异构协同计算对负载平衡算法提出了3方面要求:低算法复杂度、适应多级嵌套的数据传输系统和支撑异构协同计算.通过组合3级嵌套负载平衡算法框架、贪婪剖分算法和内外子区域剖分算法,设计了一种能够同时满足这3方面要求的负载平衡算法.模型测试表明,算法可以达到90%以上的负载平衡效率.天河2号上32个节点的测试表明,算法能够保证通信开销较小.5个典型应用在天河2号上最大93.6万核的测试表明,算法能够支撑应用高效扩展,并行效率最高可达80%.

关键词 并行计算;负载平衡;异构协同计算;天河2号;至强融核协处理器

中图法分类号 TP301

收稿日期:2016-11-21;**修回日期**:2017-02-20

基金项目:国家自然科学基金重大研究计划重点项目(91430218)

This work was supported by the Major Research Plan of the National Natural Science Foundation of China (91430218).

当前,超级计算机,特别是亿亿次计算机,呈现出海量并行、多级嵌套的数据传输系统和异构加速的体系结构特征:

1) 海量并行. 现在的亿亿次计算机都具有数十万核,例如,神威太湖之光具有超过 1 000 万核;若包括至强融核协处理器(many integrated core, MIC),天河 2 号具有超过 3 百万核.

2) 多级嵌套的数据传输系统. 超级计算机的数据传输系统具有很深的层次结构,以天河 2 号为例,可以分为"主干交换机-分支交换机-叶子交换机-远端内存-本地内存-多级缓存系统". 在这些级之间,数据传输的平均带宽具有逐级递增的趋势,延迟具有逐级递减的趋势,每级之间一般具有一倍至几倍的差异. 在多级嵌套的数据传输系统中,数据传输的开销不但与数据传输量和数据传输次数相关,还与数据传输的通信链路相关.

3) 异构加速不可忽视. 为了在较低的功耗下获得较好的计算性能,很多超级计算机采用了异构加速的技术. 据文献[1]统计,在全球最快的 10 台超级计算机中,神威太湖之光使用了主从核结构,天河 2 号使用了 MIC 加速,Titan 和 Piz Daint 使用了 GPU 加速. 特别地,我国排名前 5 的超级计算机中,神威太湖之光采用了主从核结构,2 台采用了 CPU/MIC 加速的结构,2 台采用了 CPU/GPU 加速的结构. 异构超级计算机节点内的体系结构有 2 点显著的变化:① 节点内 CPU 核和加速器之间(或主从核)的计算速度出现了差异;②CPU 核和加速器之间(或主从核)的访存特征出现了差异,且节点内的通信出现了与节点间通信相似的特性(见图 1).

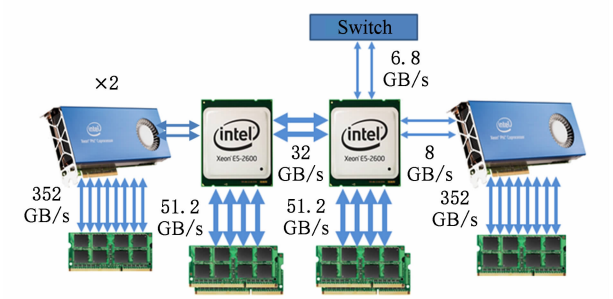

Fig. 1　Intra-node bandwidth distribution of Tianhe-2

图 1　天河 2 号节点内带宽分布

亿亿次超级计算机的上述 3 个特征,给负载平衡算法提出了 3 个要求:

1) 低算法复杂度. 一方面,超级计算机具有海量的并行度;另一方面,应用模拟问题也具有海量的未知量规模. 因此,负载平衡算法面对的问题规模也非常巨大. 如果将高算法复杂度的负载平衡算法用于解决动态负载平衡问题,其负载平衡算法执行开销就会接近甚至超过数值计算时间,从而变得不可接受.

2) 适应多级嵌套的数据传输系统. 在多级嵌套的数据传输系统中,数据传输速度与通信链路相关. 传统的负载平衡算法只考虑了通信量,没有考虑通信链路,因此,即使能够最小化数据传输量,也不能最小化数据传输开销.

3) 支撑异构协同计算. 对于复杂应用来说,要想充分利用计算资源,就需要做好异构协同计算. 但是,在异构协同计算中,由于 CPU 与异构加速器之间计算速度等性能特征存在很大差异,因此会造成即使在同构计算中自然平衡的应用,此时也需要处理复杂的负载平衡问题. 另外,对于天河 2 号而言,其 MIC 加速器与远程 CPU 或其他 MIC 加速器的通信链路性能很差,因此负载平衡算法必须规避 MIC 加速器的远程通信.

面向亿亿次超级计算机,针对结构网格并行应用的异构协同计算,我们设计了一种 3 级嵌套负载平衡算法,并且针对天河 2 号进行了特殊处理,使得该算法能够满足上述 3 个要求.

1　相关研究与问题分析

1.1　通过并行化降低负载平衡算法复杂度

面对数百上千万核的并行度,即便是线性复杂度的串行负载平衡算法都难以满足实际应用的需求,而并行化是最主要的降低负载平衡算法复杂度的方法.

明尼苏达大学的 Karypis 等人[2-3],Sandia 国家实验室的 Boman 等人[4],波尔多国立高等电子、信息与无线电通讯学校的 Pellegrini 等人[5]设计并实现了并行多层次图剖分方法,提升了负载剖分的速度. 这些工作分别集成到了软件 ParMetis[6],Zoltan[7],pt-scotch[8]中. Menon 等人[9]以 Charm++为平台,研究了基于统计学的并行扩散式负载平衡算法,在保持算法高可扩展性的同时改进了负载调整的质量. Lawrence Livermore 国家实验室的 Gunney 等人[10]提出了一种并行扩散式负载平衡算法(cascade partitioning algorithm),算法集成到 SAMRAI 框架,能够支撑自适应计算的示例程序在 Sequoia 上强扩展到 150 万核.

总体而言,由于扩散式负载平衡算法主要使用

局部操作,因此其并行可扩展性较好,可以扩展到上百万核.图剖分类和空间填充曲线类的负载平衡算法由于具有一些非局部操作,因此并行可扩展性较差.但是,上述算法未考虑异构体系结构,还不能适应异构协同计算.

1.2 匹配多级嵌套的数据传输系统负载平衡算法

Zheng[11]设计了一种多级的负载平衡算法,该算法使用一棵树管理所有的核,树的每个叶子节点对应1个核,算法将计算任务沿着树的边从过载核调整到轻载核,从而达到负载平衡.特别地,该树的结构与计算机的体系结构相匹配.利用这种匹配性,负载平衡算法可以适应多级嵌套的数据传输系统,减小数据传输开销.此外,算法自根向树叶执行,不同子树的负载平衡可以并行执行,因此,算法具有较好的并行可扩展性.

南里奥格兰德联邦大学的Pilla等人[12]和法国国家信息与自动化研究所的Jeannot等人[13]对上述算法进行了改造,重点减小了数据传输的开销.但是,这2种算法时间复杂度过高,不适用于大规模问题.

1.3 支撑异构协同计算的负载平衡算法

薛巍、杨超和卢宇彤等人[14-15]对CPU/MIC异构协同计算的负载平衡算法进行了深入的研究.特别地,针对MIC加速器与远程CPU或其他MIC加速器的通信链路性能差的问题,提出了inner-outer subdomain partition方法.如图2所示.该方法使用2级剖分,第1级以MIC加速器及其本地CPU为单位剖分计算区域,第2级进一步将每个子区域剖分为内部区域和边界区域,MIC加速器分配内部区域,CPU分配边界区域.该方法避免了MIC加速器与远程CPU或其他MIC加速器的通信,提升了通信性能.

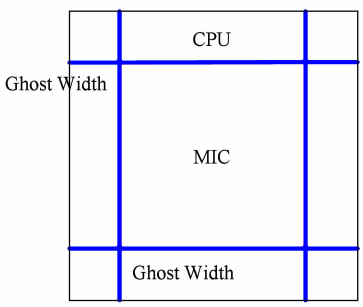

Fig. 2 Illustration of the inner-outer subdomain partitioning method: one MIC and one CPU

图2 Inner-outer subdomain剖分方法

但是,目前该方法只能适用于负载均匀分布的应用,对于负载非均匀分布的应用(如粒子类模拟),还不能适应.

1.4 已有负载平衡算法的问题

这些改进的算法针对特定的优化目标取得了明显的进展.但是在实际亿亿次计算机上,实际应用的负载平衡问题往往是复杂的多约束多目标优化问题,不但要满足低算法复杂度、匹配数据传输系统和支撑异构协同计算,还要能适应非均匀的负载分布、复杂的计算区域以及满足应用的特殊约束[16-17].例如,在使用PIC方法进行的惯性约束核聚变激光等离子体相互作用模拟中,粒子在空间中非均匀分布(如图3,粒子分布于图中圆椎体部分),造成计算负载在空间中的非均匀分布,负载平衡算法必须能够处理负载非均匀分布的问题.然而,现有的针对亿亿次协同计算设计的负载平衡算法还不能解决上述问题.

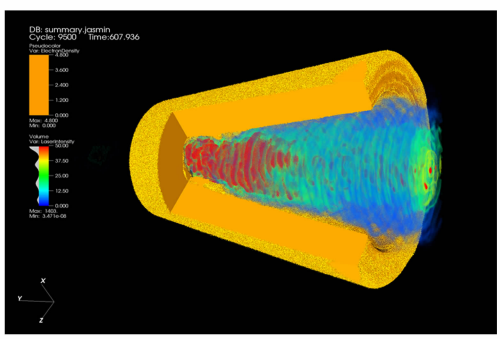

Fig. 3 The non-uniform distribution of particles in a laser plasma interaction simulation

图3 激光等离子体相互作用模拟中的非均匀粒子分布

2 支撑异构协同计算的3级嵌套负载平衡算法

2.1 设计目标

为了支撑异构协同计算,本文设计的负载平衡算法需要满足2个目标:

1) 适应亿亿次计算机,做到低算法复杂度、适应多级嵌套的数据传输系统和支撑异构协同计算;

2) 适应实际应用,能够处理非均匀负载分布.

2.2 3级嵌套的负载平衡算法框架

当计算机不包含加速器时,3级嵌套的负载平衡算法框架执行流程如下(见图4所示):

步骤1. 计算节点层负载平衡(串行执行).

步骤2. CPU层负载平衡(并行执行).

步骤 3. CPU 核层负载平衡(并行执行).

该算法框架与文献[11]比较接近,利用步骤2～3 的并行执行,降低整体的算法复杂度. 同时,该框架在步骤 1 中采用极小化数据通信的算法、步骤 2 中采用极小化远程访存的算法,从而适应多级嵌套数据传输系统. 与文献[11]不同,该算法框架直接与最典型的 3 级嵌套体系结构相匹配,并且在计算节点内采用共享存储线程并行,从而进一步提升执行性能. 基于该算法框架,剩下的核心问题就是处理异构协同计算.

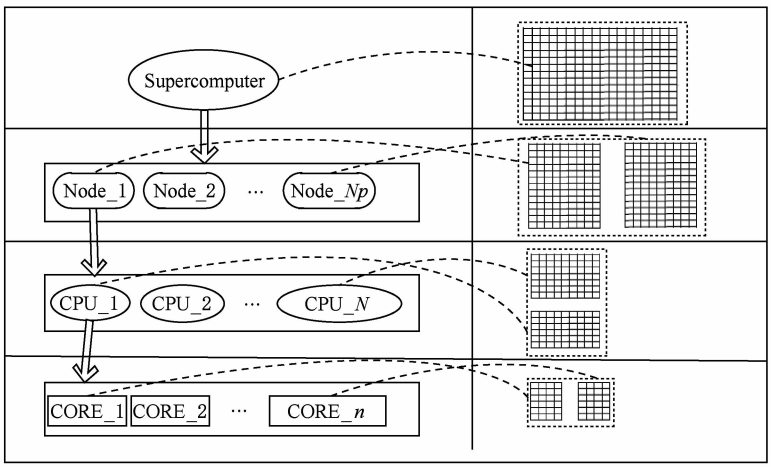

Fig. 4　Illustration of the nested partitioning load balancing algorithm

图 4　3 级嵌套的负载平衡算法框架流程示意图

2.3　适应计算能力差异

异构协同计算的首要要求是适应计算能力的差异. 基于 2.2 节的 3 级嵌套负载平衡算法框架,在流程的步骤 1～3 分别嵌入适应计算能力差异的剖分算法,则该流程就能够适应异构协同计算. 此时,每层剖分算法需要求解的问题为:给定加速器与 CPU 的计算速度比,给定计算区域及其上的负载分布,求计算区域的剖分及其到 CPU、加速器的映射. 我们基于贪婪策略设计了算法 1.

算法 1. 异构协同计算的贪婪子区域剖分算法.

输入:计算区域集合 B、计算资源集合 P(包括计算速度的比值);

输出:子区域集合 B'、B'到 P 的映射 M.

步骤 1. 计算每个计算资源应分到的计算量,初始化剩余的计算资源集合 $L=P$;

步骤 2. 初始化未分配的计算区域集合 $Q=B$;

步骤 3. 根据贪婪原则分配计算区域;

While $Q\neq\varnothing$

① 从 Q 中取出最大的子区域 b,从 L 中取出剩余计算能力最多的元素 c;

② If $b>c$ then

将 b 剖分成若干子区域 $b_0\sim b_n$,其中 b_0 是计算量不小于 c 的最小子区域,b_0 分配给 c,更新 B'和 M,将 $b_1\sim b_n$ 放回 Q;

Else

将 b 分配给 c,更新 B'和 M;

End If

③ 更新 c,加入 L;

End While

步骤 4. Return B'和 M.

MIC 计算能力较强,且适合处理单块计算区域. 如果要保证 1 个 MIC 仅分配 1 个子区域,则在步骤 3③增加:如果 c 是加速器,则不再加入 L 即可. 从步骤 3 可以看出,算法优先将较大的整块区域分配给计算能力强的 MIC,满足了 MIC 的特点. 从步骤 3②可以看出,算法能够定量地根据 MIC 与 CPU 的计算速度比值,分配不同大小的计算区域.

图 5 表现了算法 1 在天河 2 号的 1 个节点上执行的效果,其中不同颜色表示 MIC 或者不同 CPU

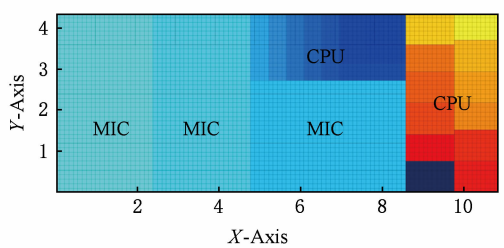

Fig. 5　Domain partitioning result produced by algorithm 1 for the case of three MICs and two CPUs

图 5　3 块 MIC 和 2 块 CPU 时算法 1 得到的子区域剖分

核上分配的子区域.从图 5 中可以看出,算法 1 能够根据计算速度的不同分配不同计算量的子区域.

算法 1 具有一个额外的功能:能够处理计算节点具有不同计算能力的问题.(该功能主要为了解决由于部件老化造成的计算节点间存在计算能力不同,从而引发的负载不平衡问题.)图 6 表现了算法 1 在 1 个具有 32 个节点,节点分别具有 0～3 个 MIC 的计算环境上执行的效果,其中不同颜色表示不同节点分配的子区域.从中可以看出,算法 1 能够根据不同节点具有 MIC 数量的不同分配不同计算量的子区域.

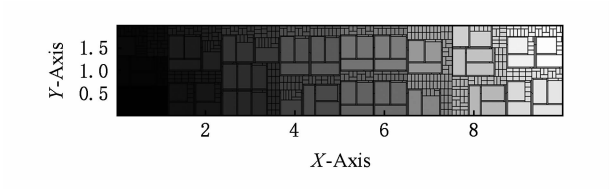

Fig. 6 Gray-scale illustration of partition assignment over 32 nodes

图 6 32 节点情况下进程的子计算区域分布灰度图

2.4 规避 MIC 的远程通信

对于天河 2 号来说,异构协同计算还需要规避 MIC 的远程通信瓶颈.基于 2.2 节的 3 级嵌套负载平衡算法框架,在步骤 3 嵌入规避 MIC 远程通信的剖分算法,则该流程就能够适应异构协同计算.为此,推广了文献[14-15]中的 inner-outer subdomain partitioning 算法,设计了算法 2,使之能够适应负载非均匀分布的应用.

算法 2. 推广的内外区域剖分算法.

输入:计算区域集合 B(包括影像区宽度)、计算资源集合 P(包括计算速度的比值);

输出:子区域集合 B'、B'到 P 的映射 M.

步骤 1. 计算每个计算资源应分到的计算量,初始化 MIC 计算资源集合 L;

步骤 2. 初始化未分配的计算区域集合 $Q=B$;

步骤 3. 根据 inner-outer subdomain 算法为 MIC 分配计算区域;

While $L\neq\varnothing$

① 从 L 中取出计算能力最强的元素 c,计算 Q 的内部区域,从中取出适当的子区域分给 c,更新 B'和 M;

End While

步骤 4. 使用算法 1 分配剩余计算区域到 CPU,更新 B'和 M;

步骤 5. Return B'和 M.

从步骤 3①可以看出,算法能够保证 MIC 仅分配当前的内部计算区域,因此 MIC 可以避免远程通信.图 7 表现了算法 2 在天河 2 号的 1 个节点上执行的效果,其中不同颜色表示 MIC 或者不同 CPU 核上分配的子区域.在该算例中,影像区宽度为 1.从中可以看出,算法 2 不但能够根据计算速度的不同分配不同计算量的子区域,而且可以保证 MIC 避免远程通信.

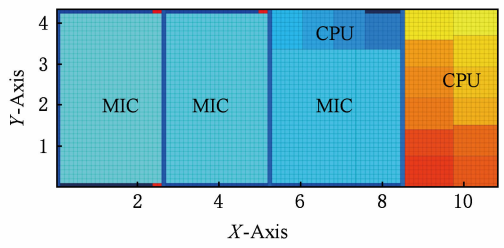

Fig. 7 Domain partitioning result produced by algorithm 2 for the case of three MICs and two CPUs

图 7 3 块 MIC 和 2 块 CPU 时算法 2 得到的子区域剖分

3 应用效果

我们设计了 3 组测试验证算法的效果.第 1 组测试使用模型问题和模型计算环境,统计负载平衡算法的模型负载平衡效率,检测理想环境下负载平衡算法的负载平衡效果.第 2 组测试使用 JEMS-TD 应用程序,在天河 2 号上执行,比较应用程序使用算法 1 和算法 2 的通信时间,测试算法 2 规避 MIC 远程通信的效果.第 3 组测试使用 JEMS-TD,LAP3D,LARED-S,LARED-P,MOASP 5 个应用程序,在天河 2 号上执行,统计程序执行的并行效率,考察负载平衡算法的整体效果.

3.1 异构计算环境下的负载平衡效率计算公式

为了评价负载平衡算法的效果,需要一个能够从平衡性方面评价负载平衡算法执行结果的指标.在同构计算环境下,评价平衡性的指标主要有负载平衡效率、负载不平衡因子等[18].这些指标可以互相转换,没有本质区别.

在同构计算环境下,负载平衡效率指平均计算时间与最大计算时间的比值,记为 LBE.负载平衡效率满足 2 个性质:1)属于(0,1]区间的实数;2)在仅考虑负载平衡因素,不考虑通信开销等因素的情况下,负载平衡效率越高,计算时间越短.对于负载平衡算法而言,其执行效果受限于输入负载模型的精度.因此,根据负载模型计算出的模型负载平衡

效率可以更好地反映负载平衡算法的效果.模型负载平衡效率

$$LBE_{\text{model}}=\frac{ave(\{l_i,i=1,2,\cdots,n\})}{\max(\{l_i,i=1,2,\cdots,n\})},\qquad(1)$$

其中 l_i 表示核 i 分到的负载.

对于异构计算环境,如果直接使用上述计算公式,则无法保证性质成立.例如,假设在某个异构计算环境中,有 10 个 CPU 核,CPU 核计算速度为 1,有 1 个加速器,加速器计算速度为 100,有 100 个计算量为 1 的计算任务需要分配.分配方式 1:100 个计算任务全部分配给加速器,$LBE_{\text{model}}=\frac{1}{11}$,计算时间为 1.分配方式 2:100 个计算任务全部分配给 CPU,$LBE_{\text{model}}=\frac{10}{11}$,计算时间为 10.比较 2 种分配方式可知,在异构计算环境中,直接使用上述计算公式,无法保证负载平衡效率越高,计算时间越短.

为了在异构计算环境中,保证 2 个性质继续成立,将式(1)推广为

$$LBE_{\text{model}}=\frac{sum(\{l_i,i=1,2,\cdots,n\})}{sum(\{s_i,i=1,2,\cdots,n\})}\Big/\max(\{l_i/s_i,i=1,2,\cdots,n\}),\qquad(2)$$

其中 s_i 表示核 i 的计算速度.显然 LBE_{model} 是属于(0,1]区间的实数,性质 1 成立.给定计算任务和计算环境,则 $\frac{sum(\{l_i,i=1,2,\cdots,n\})}{sum(\{s_i,i=1,2,\cdots,n\})}$ 是固定值,$\max(\{l_i/s_i,i=1,2,\cdots,n\})$ 就是计算时间,因此 LBE_{model} 越大,计算时间越短,性质 2 成立.式(2)对于在模型层面比较算法的效果已经足够.但是,式(2)要求给定计算速度的比值,这在数值模拟测试中会很困难.因此,本文仅比较模型负载平衡效率.

3.2　模型负载平衡效率测试

测试计算区域是一个 1600×320 的 2 维区域.模型计算环境有 3 种:1)纯 CPU;2)每个节点 3 个 MIC;3)每个节点具有 0～3 个 MIC(数量随机生成).其中每个节点有 2 个 CPU,CPU 具有 12 核,MIC 卡与 CPU 核的计算速度之比为 12:1.测试使用算法 2.

图 8 给出了算法将计算区域分到 32 个节点的结果,实线代表剖分.从图 8 中可以看出 MIC 和 CPU 分到的计算区域大小不同.此外,图 8 中灰度表示进程号,可以看出,相邻区域进程号接近(灰度接近的进程号也接近),避免了远距离通信.

图 9 显示了算法在 1～32 个节点上执行得到的负载平衡效率.首先,3 条负载平衡效率曲线都高于

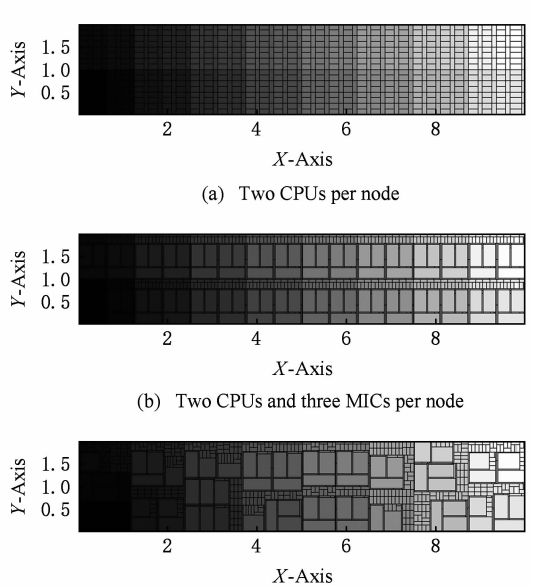

Fig. 8　Partition assignment over 32 nodes

图 8　32 节点的子计算区域剖分

0.9,表明算法可以保证较好的负载平衡.其次,点线(3MIC 环境)与实线(全 CPU 环境)基本相当,同时虚线略低于点线和实线,表明算法对异构协同计算的支持效果接近同构计算,而且能够适应计算机的节点之间具有差异的情况.

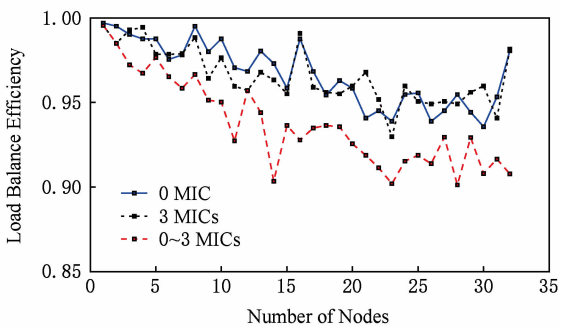

Fig. 9　Model load balance efficiency for different number of nodes

图 9　模型负载平衡效率

3.3　规避 MIC 远程通信测试

JEMS-TD 是一个 3 维时域全波电磁模拟程序,使用时域有限差分(FDTD)方法求解 Maxwell 方程组,对包含多种材料的复杂模型的辐射和散射问题进行全波电磁模拟,获得时域近场和远场电磁数据.测试计算区域是一个 600×400×400 的 3 维区域.测试在天河 2 号上进行,每个节点具有 3 个 MIC,2 个 CPU,每个 CPU 具有 12 核.测试中每个计算节点用满了 195 个 CPU 核和 MIC 核.对于该程序,MIC 卡与 CPU 核的计算速度之比约为 12:1.

由于时间所限,我们只进行了中等规模的测试.图 10 显示了应用程序的通信时间,其中实线表示使用算法 1 的通信时间,虚线表示使用算法 2 的通信时间).从图 10 中可以看出,使用算法 2 普遍可以比算法 1 降低通信时间.32 个节点时,使用算法 2 比算法 1 降低约 50%的通信时间.

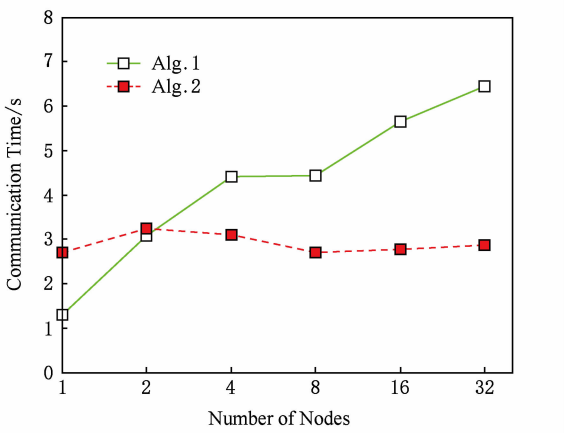

Fig. 10 Communication time for different number of nodes

图 10 通信时间随计算节点数的变化

3.4 并行效率验证

选取了 5 个典型应用进行了并行效率的测试.除了 3.3 节介绍的 JEMS-TD 外,LAP3D 是一个 3 维激光等离子体相互作用成丝不稳定性模拟程序,LARED-S 是一个 3 维辐射流体力学界面不稳定性模拟程序,LARED-P 是一个 3 维激光等离子体相互作用粒子模拟程序,MOASP 是一个经典分子动力学模拟程序.测试在天河 2 号上进行,每个计算节点用满了 195 个 CPU 核和 MIC 核.测试使用算法 2,采用弱可扩展性测试,由于篇幅限制,这里不列出测试模型的具体信息.

表 1 给出了 5 个程序的并行效率.其中,由于时间所限,LARED-S 和 MOASP 没有测到 93.6 万核的数据.从测试结果看,本文的负载平衡算法成功支撑了 5 个典型应用高效扩展到数十万到近百万核,并行效率最高可达 80%.

Table 1 Parallel Efficiency of Five Applications Using CPU/MIC Hybrid Computing

表 1 5 个应用程序异构协同计算的并行效率

Number of Cores	Parallel Efficiency/%				
	JEMS-TD	LAP3D	LARED-S	LARED-P	MOASP
195	100	100	100	100	100
780	92	92	97	90	84
1 560	94	91	95	89	92
6 240	92	86	93	87	65
12 480	92	81	92	85	60
49 920	89	69	89	84	48
199 680	90	79	91	82	41
399 360	84	62	89	79	43
798 720	80	54	84	79	
936 000	80	53		78	

4 结束语

面向亿亿次超级计算机,针对结构网格并行应用的异构协同计算,通过组合 3 级嵌套负载平衡算法框架、贪婪剖分算法和内外子区域剖分算法,我们设计了一种负载平衡算法,能够同时满足低算法复杂度、适应多级嵌套的数据传输系统和支撑异构协同计算 3 方面要求.模型测试表明,算法可以达到 90%以上的负载平衡效率.天河 2 号上 32 个节点的测试表明,算法能够保证通信开销较小.5 个典型应用在天河 2 号上最大 93.6 万核的测试表明,算法能够支撑应用高效扩展,并行效率最高可达 80%.

本文的算法需要输入 CPU 与加速器的计算速度比值,但是对于实际应用,如何得到该比值也是一个需要研究的问题,在下一步的工作中,我们将在这个方面开展研究.此外,我们准备移植和实现更多异构协同负载平衡算法,进行更加充分的对比和分析.

参 考 文 献

[1] Dongarra J. TOP500 [EB/OL]. Knoxville: University of Tennessee, 2016-11 [2017-01-13]. https://www.top500.org/lists/2016/11/

[2] Karypis G, Kumar V. Parallel multilevel k-way partitioning scheme for irregular graphs [C] //Proc of the 1996 ACM/IEEE Conf on Supercomputing (Supercomputing'96). Los Alamitos, CA: IEEE Computer Society, 1996: No. 35

[3] Lasalle D, Karypis G. Multi-threaded graph partitioning [C] //Proc of the 27th IEEE Int Symp on Parallel and Distributed Processing (IPDPS'13). Los Alamitos, CA: IEEE Computer Society, 2013: 225-236

[4] Devine K D, Boman E G, Heaphy R T, et al. Parallel hypergraph partitioning for scientific computing [C] //Proc of the 20th Int Conf on Parallel and Distributed Processing (IPDPS'06). Los Alamitos, CA: IEEE Computer Society, 2006: 124-124

[5] Chevalier C, Pellegrini F. PT-Scotch: A tool for efficient parallel graph ordering [J]. Parallel Computing, 2008, 34(6): 318-331

[6] Karypis G. ParMETIS [CP/OL]. Twin Cities: University of Minnesota, (2013-03-30) [2017-01-13]. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview

[7] Boman E. Zoltan [CP/OL]. Albuquerque: Sandia National Laboratories, 2016-01 [2017-01-13]. http://www.cs.sandia.gov/zoltan/

[8] Pellegrini F. PT-SCOTCH [CP/OL]. Bordeaux: INRIA, (2012-12-02) [2017-01-13]. http://www.labri.fr/perso/pelegrin/scotch

[9] Menon H, Kalé L. A distributed dynamic load balancer for iterative applications [C] //Proc of the Int Conf on High Performance Computing, Networking, Storage and Analysis (SC'13). New York: ACM, 2013: No. 15

[10] Gunney B T N, Anderson R W. Advances in patch-based adaptive mesh refinement scalability [J]. Journal of Parallel & Distributed Computing, 2016, 89(C): 65-84

[11] Zheng G, Bhatelé A, Meneses E, et al. Periodic hierarchical load balancing for large supercomputers [J]. Int Journal of High Performance Computing Applications, 2011, 25(4): 371-385

[12] Pilla L L, Ribeiro C P, Coucheney P, et al. A topology-aware load balancing algorithm for clustered hierarchical multi-core machines [J]. Future Generation Computer Systems, 2014, 30(1): 191-201

[13] Jeannot E, Mercier G, Tessier F. Topology and affinity aware hierarchical and distributed load-balancing in Charm++ [C] //Proc of the 1st Workshop on Optimization of Communication in HPC (COM-HPC'16). Piscataway, NJ: IEEE, 2016: 63-72

[14] Xue Wei, Yang Chao, Fu Haohuan, et al. Enabling and scaling a global shallow-water atmospheric model on Tianhe-2 [C] //Proc of the 28th IEEE Int Symp on Parallel and Distributed Processing (IPDPS'14). Los Alamitos, CA: IEEE Computer Society, 2014: 745-754

[15] Xue Wei, Yang Chao, Fu Haohuan, et al. Ultra-scalable CPU-MIC acceleration of mesoscale atmospheric modeling on Tianhe-2 [J]. IEEE Trans on Computers, 2015, 64(8): 2382-2393

[16] Li Leisheng, Wang Chaowei, Ma Zhitao, et al. petaPar: A scalable and fault tolerant petascale free mesh simulation system [J]. Journal of Computer Research & Development, 2015, 52(4): 823-832 (in Chinese)
(黎雷生, 王朝尉, 马志涛, 等. 千万亿次可扩展可容错自由网格数值模拟系统[J]. 计算机研究与发展, 2015, 52(4): 823-832)

[17] Cao Xiaolin, Zhang Aiqing, Mo Zeyao. Parallel computation for particle simulations based on object-oriented design [J]. Journal of Computer Research & Development, 2007, 44(10): 1647-1651 (in Chinese)
(曹小林, 张爱清, 莫则尧. 基于面向对象的粒子类模拟并行计算研究[J]. 计算机研究与发展, 2007, 44(10): 1647-1651)

[18] Willebeeklemair M H, Reeves A P. Strategies for dynamic load balancing on highly parallel computers [J]. IEEE Trans on Parallel & Distributed Systems, 1993, 4(9): 979-993

Liu Xu, born in 1981, PhD and associate professor. His main research interests include parallel computing, load balancing and computational geometry.

Yang Zhang, born in 1984. PhD, associate professor. His main research interest is high performance computing middle-ware.

Yang Yang, born in 1983. PhD. His main research interest is domain specific language.