N-body 算法及其并行化

王小伟，郭力，杨章远
（中国科学院过程工程研究所多相反应开放实验室，北京，100080）

摘要：N-body 问题涉及了科学和工程中的许多领域，它的主要特点就是 $O(N^2)$ 的计算量，采用并行计算方法是解决 N-body 问题巨大计算量的终极选择。针对该类问题的具体特点以及不同的并行计算机体系结构，目前有多种算法有效地减少了计算量，加快了求解速度。本文介绍了 N-body 问题的几种常见算法和它们的并行化方法。

关键词：N－body；算法；并行
中图分类号：TP 338.6
文献标识码：A
文章编号：1001-4160(2003)02-195-200

N-body algorithms and parallelization of them

WANG Xiao-Wei, GUO Li, YANG Zhang-Yuan
(Multi－Phose Reaction Laboratory, Institute of Process Engineer, Chinese Academy of Science, Beijing, 100080, China)

Abstract: The N-body problem covers many fields in science and technology, which is computationally intensive for $O(N^2)$ complexity, thus a candidate for parallel computation. Many algorithms have been developed according to the character of this problem and the architecture of the parallel computers, which effectively reduced the computational work and fastened the speed. Several algorithms for N-body problem and the parallelization of them are introduced in this paper.

Keywords: N-body; algorithm; parallelization

1 引言

N-body 问题可以准确地描述为如下：在一定的物理空间中，分布有一定数量的粒子，每对粒子间都存在相互作用（如万有引力，库仑力等）。它们初始的速度和位置是确定的，每隔一定的时间步，由于粒子间的作用，需要更新粒子的速度和位置。更新给定粒子的速度和位移的计算很简单，就是把另外 N－1 个粒子对它作用的结果叠加起来，但这导致了每一时间步 $O(N^2)$ 的计算量。

N-body 模拟问题覆盖了自然科学的很多领域，从宇观的天体物理到宏观的流体动力学，直至微观的分子动力学。例如通过研究围绕着银河系的暗物质晕轮的形状和动力学特性来探索银河系的形成过程，需要模拟数百万的星体和暗物质间的作用。现代生物物理学和化学中的许多研究，如细菌或植物体的光合作用膜处发生的光能向化学能的转化，染色体中 DNA 和蛋白质分子的描述，都需要模拟上千万的原子和分子的作用。

可见，N-body 问题的两个重要的特征是：① 计算规模大，因为无论是宇观的天体尺度还是微观的分子尺度，都包含了大量的粒子，粒子的规模大到数百万、千万。由于在系统中任意的两个粒子间都存在着作用，因此直接计算粒子间的相互作用的量级就是 $O(N^2)$。②系统是动态变化的。为了反映系统的具体变化，尤其是在微观分子结构中，要求时间步足够的小。这两个特征决定了计算机模拟时巨大的计算量，这对于任何高性能的单台计算机来说都是一个很难突破的瓶颈，因此采用并行计算是解决 N-body 计算问题的必然选择。

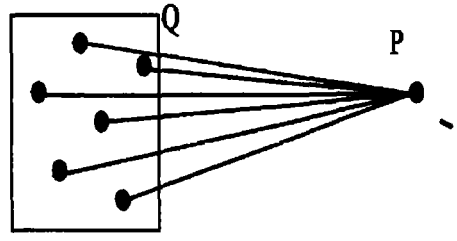
并行计算的发展是基于两方面的认识：第一，单机性能不可能满足大规模科学与工程问题的计算需求，而并行计算是实现高性能计算、解决挑战性计算问题的唯一途径；第二，同时性和并行性是客观物质世界存在的普遍属性，具有实际的物理背景的计算问题在许多情况下都可划分为能够并行计算的多个子任务。针对某一具体应用问题，我们可以利用它内部的并行性，将其分解为相互独立、但彼此又有一定联系的若干个子问题，分别交给各处理器，由各处理器采用并行算法完成初始问题的求解。因此，基于并行算法，我们可以利用高性能的并行机来求解大规模应用问题，满足实际的需求。

收稿日期：2002-11-23；**修回日期**：2002-12-20
基金资助：国家自然科学基金资助项目（20221603）
作者简介：王小伟（1976—），男，山东人，博士生.

为了解决直接计算粒子受力的 $O(N^2)$ 的计算量,研究人员对 N-body 模拟的算法进行了广泛而深入的研究,提出了许多种解决方法,主要可以归结为以下两种途径:① 在考虑远程一组粒子对单个粒子的作用时,不是简单的计算组中每一个粒子对它的作用,而是将组中所有粒子作为一个整体来考虑,基于此产生了树形分级算法。② 将长程的作用截断到某一局部区域,从而可以减少相互作用的粒子数,通过采用邻居列表(Neighbor List)或关联格子(Link Cell)等方法来确定邻近的作用粒子,从而可以有效的减少计算量。

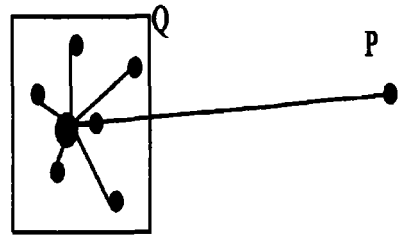
2 树形分级算法

分级算法是基于以下理论:如果粒子相互间作用力的大小随着距离的增大下降很快时,当一组粒子距离被作用的粒子足够远时,它们的作用可以近似地用一个等价粒子来代替,如图 1 所示。



(a) Q 处 N 个粒子对 P 的作用

(a) the effect of N particles at Q on P



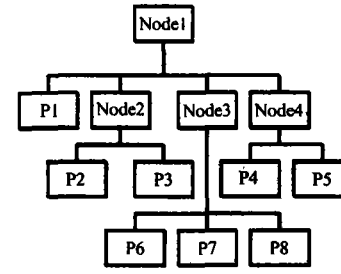
(b) Q 处等价粒子对 P 的作用

(b) the effect of equivalence particle at Q on P

图 1 远程一组粒子对单个粒子的作用

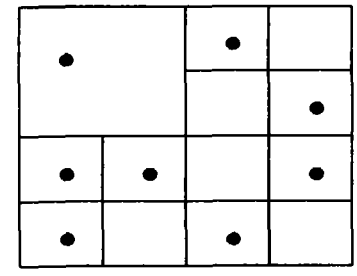
Fig.1 The effect of a group of particles faraway on single particle

分级算法用树形数据结构来代表对整个物理空间划分形成的粒子集,在给定的精度范围内来近似粒子和粒子集间的相互作用。在这种算法里,整个物理空间被递归的分成不同层次的单元格,在三维空间中就是八元树,二维空间中是四元树。在最细级别的单元(分级树的叶子)包含了粒子,较高层次的单元(节点)代表了以它们为根的子树中粒子的集合。分级树的创建过程是这样的:首先从根节点开始,根结点代表了整个物理空间,包含了所有的粒子,将物理空间递归的分成 8 个(三维)子节点或 4 个(二维)子节点来形成树的每一层次,直到最细的单元上有 m 个粒子为止,如图 2 所示。



(a)二维空间的分级树结构(Node:节点,P: 粒子)

(a) the hierarchical tree in two-dimension space



(b) 粒子在物理空间中的分布

(b) the particle distribution in physical space

图 2 分级树的结构

Fig.2 The structure of hierarchical tree

Barnes-Hut 分极树算法[1](Barnes-Hut Hierarchical Tree-code)和快速多极算法 FMA[2](Fast Multipole Algorithm)是分级算法的典型代表,在 N-body 问题中得到了广泛的应用。另外,还有 Anderson 方法[3]和 Duke 大学的 PMTA[4](Parallel Multipole Tree Algorithm)等。这里主要介绍一下 Barnes-Hut 分极树算法和快速多极算法 FMA。

2.1 Barnes-Hut 分级树算法

Barnes-Hut 分级树算法首先是由 John Barnes 和 Piet Hut 在 1986 年提出的[1]。在 Barnes-Hut 分级树算法中,远程一组粒子的作用通常用它们的总质量和质量中心来近似,并且在分级树的最细层次上每个单元格最多有一个粒子。在计算粒子的受力时,粒子从分级树的根节点开始遍历分级树,如果节点和粒子间的距离符合足够远的判据,则利用节点的总质量来计算对粒子的作用,否则就需要细分到更

细的层次来进一步比较。这样,可以将粒子间作用的计算时间减少到 $O(N \log N)$量级。判据可以表示为如下形式:

$$d > \theta \cdot l_m \quad (1)$$

式中,d 为粒子与节点中心的距离,θ 为常数,l_m 为节点的空间尺寸,$l_m = l/2^m$,l 为物理空间的尺寸,m 为节点所在树的级数。

计算粒子间受力的精确度可以通过采用较小的 θ 值来提高,更有效的方法是对粒子组采用高阶的近似[5],而不是仅一阶的质量中心来代替。

串行的 Barnes-Hut 分级树算法可以分为以下几步:

① 初始化粒子的质量、位移和速度。

② 根据粒子位置,在整个计算空间创建分级树。

③ 从下至上遍历分级树,计算树中每一节点的总质量和质量中心(或更高阶近似)。

④ 每一个粒子从上至下遍历分级树来计算它的受力,若粒子距离某一节点足够远,则利用总质量和质量中心(或更高阶近似)进行计算,否则需要遍历它的子节点。

⑤ 更新粒子的速度和位移,转到②继续迭代,直到满足要求为止。

2.2 快速多极算法 FMA(Fast Multipole Algorithm)

Matthew Pincus 和 Herold Scheraga 在 1977 年最初提出了关于 FMA 的基本观点[6]。他们认为远程一组粒子的属性可以方便地用该组粒子的多极展开式来代表,无穷的但是快速收敛的多极展开式可以方便地取一定阶数(通常取 3 - 8 阶),阶数越多则近似程度的准确率越高,这样就可以根据保证的精度来选择多极展开式的项数。

在 FMA 算法中,分级树最细的单元格上可以有 $S > 1$ 个粒子,而不像 Barnes-Hut 那样最多有一个粒子。在计算远程粒子集对单个粒子的作用时,不是象 Barnes - Hut 中直接计算粒子和粒子集间的作用,而是通过包含该粒子的粒子集和远程的粒子集间作用实现的。因此,FMA 算法中受力计算的基本单元不是粒子,而是粒子集。这是因为远程粒子集的作用在 FMA 算法中是通过多极展开式来计算的。Greengard 和 Rokhlin 在快速多极算法中引入了局部展开的概念[2],如果知道了在某一点 P 的多极展开式的系数,通过一系列的数学变换,我们就可以在 P 的临近区域构造势的泰勒展开式,即所谓的局部展开式。这个展开式可以用来估计 P 点附近的粒子的受力。这样进一步地将受力计算的复杂度降低到 $O(N)$ 量级,大大减少了计算量。

串行的 FMA 算法的主要步骤如下:

① 初始化粒子信息、分级树的层次 l、多极展开式的项数 p。

② 把粒子分配到分级树的各级单元中,并且构建每一单元的作用列表。

③ 从下至上遍历分级树,计算出各单元的多极展开式。

④ 从上至下遍历分级树,由②得到的多极展开式转化为在单元中的局部展开式。

⑤ 根据局部展开式和粒子的位置,计算出远程粒子集的作用。

⑥ 直接计算邻近的粒子间的作用,并和远程的作用累加,得到粒子的受力。

⑦ 更新粒子速度和位移,转到②继续迭代。

2.3 并行化策略

在树形分级算法中,虽然可以将计算受力的复杂度降低到 $O(N \log N)$ 甚至 $O(N)$ 量级,但由于 N-body 问题本身的特点,计算量还是相当大。同时由于算法中的许多部分可以并行执行,如分级树的创建、单个粒子的受力和位移的更新等,因此研究人员根据分级树算法的特点,对分级树算法进行了并行化研究。

负载平衡性和数据局部性是对并行算法的性能影响比较大的两个因素。在并行计算时,应使计算负载在各个处理器上尽量均布,从而可以减少处理器间由于同步而相互等待的时间。合理地分配数据到各个处理器上,使处理器间的通信尽量在局部的范围内,可以减少通信的开销,从而提高程序的并行效率。在分级树算法中,尽管粒子或单元格的空间分布在计算结束时可能与开始时有很大的不同,但它随时间的变化是缓慢的,在两个相继的时间步之间变化很小。因此,在一个时间步里计算作用力的时候,记录下在该时间步中每个粒子所做的工作(即它和其他粒子或单元格相互作用的次数),然后用它作为下一时间步和该粒子相关工作的一种度量来实现负载的平均分配。正交递归二分法 ORB(Orthogonal Recursive Bisection)和代价区法(CostZone),是树形分级并行算法中主要使用的分解策略。

正交递归二分法(ORB),通过直接划分物理空间,保持空间物理上的局部性。用上面提到的负载平衡测度,该空间被递归地分为两个带有相等工作量的矩形子空间,直到每个处理器有一个子空间(见

图3)。开始时所有进程都和整个空间相关,每次分割一个空间的时候,和它相关的一半处理器分给所得到的子空间。划分所取的笛卡尔方向通常随相继的分割交替改变,一个并行的中值计算方法用来确定在哪儿分割当前的子空间。ORB可以保证有较好的计算负载平衡和规则有效的通信结构。利用ORB对Barnes-Hut算法进行并行化时,各处理器在自己的子空间内创建分级树,并计算出各节点的质量和质量中心。此时各处理器的子空间是整个物理空间ORB树的节点。本地分级树和从其它处理器分级树获得的粒子的信息,构成了所谓的本地必要树LE-tree (Local Essential tree),利用LE-tree就可以在本地处理器上进行粒子受力计算。具体可参见[7-9]。因为在FMA算法中基本单位是单元格,不像在Barnes-Hut方法中基本单位是粒子,因此单元格有可能位于对分线上,因此需要作些处理,Singh对ORB作了改进,并利用它对FMA进行了并行化[10]。

代价区(CostZone)方法是基于分级树算法的树形数据结构,因为数据结构里已经有了一个空间分布的表示,因此可以对这个数据结构本身做划分,从而达到划分空间的目的(见图4)。每个内部单元所存放的是和它所包含的所有粒子相关的总代价。系统中总的工作量或者代价在处理器之间分担,每个处理器有一个连续的、相等区域的工作(例如,1000个单位的总工作量在10个处理器之间分配,1-100单元区分给第一个处理器,101-200区分给第二个处理器,等等)。树中的一个粒子或单元格属于哪个区,可通过按一定顺序遍历到粒子(单元格)时,它前面粒子或单元格的总代价确定。处理器并行地遍历这个树,摘取属于它们代价区的粒子或单元格。Singh[10]和Almojel[11]利用CostZone分别对FMA和Barnes-Hut算法进行了并行化。代价区方法在共享存储计算机上有较好的总体性能,这主要是由于花在划分阶段的时间本身(即计算这个划分)要小些。

3 近程作用N-body的并行算法

近程作用的N-body算法在经典的分子动力学中有广泛的应用,例如由于电子屏蔽作用,限制了粒子间作用在一局部范围,或是为了减少计算负载在某些情况下的简化处理。通常粒子间的近程作用由截断半径 r_c 来规定,距离粒子 r_c 以外的粒子的作用可以忽略,这样,计算粒子间受力的计算量就和粒子数 N 成线性关系了。尽管如此,由于粒子数众多,大量的计算时间还是消耗在计算粒子受力上。为了有效的计算粒子受力,就需要在每一时间步知道距离粒子 r_c 内的粒子,其中的关键就是把每次计算受力时需要检查的邻近粒子的数目最小化,因为检查 r_c 以外的粒子就意味着计算的浪费。通常有两种方法来解决此问题:邻居列表和关联格技术。邻居列表方法对每一个粒子建立一个邻居列表,其中存储了 $r_s = r_c + \delta$ 内的粒子(δ 是一个相对 r_c 较小的值,它的取值由温度、密度、扩散率等参数决定)。邻居列表可以用于在几个时间步内计算粒子的受力,在粒子运动到 $r > r_s$ 或 $r < r_c$ 前,需要重新建立邻居列表。通过在邻居列表中检查可能作用的粒子,将所要检查的粒子限定在一个较小的范围,减少了计算量。关联格技术就是在每一时间步,把计算空间分成小的单元格,单元格的尺寸 $d = r_c$ 或稍大一些。这样,寻找与给定粒子可能作用的粒子就限定在其周围的27个单元格子:粒子自身所在的格子和三维空间中附近的26个格子。

实际上,许多近程作用的算法既采用了邻居列表又用到了关联格技术。在这种方法中,每隔几个时间步把计算空间分成小的单元格($d >= r_s$),单元格被用来建立邻居列表,这样减少了建立邻居列表的计算量,因为可以在粒子临近的27个格子中寻找邻居,而不必检查所有粒子。在其内的每一时间步,邻居列表单独用来寻找距离粒子 r_c 内可能作用的粒子。由于以 r_s 为半径的球内的粒子数要少于以 $3r_c$ 为边长的立方体内的粒子数,因而比单独的关联格方法又减少了部分计算量。

尽管有些算法出于对具体问题和并行计算机的类型的不同作了一些改进,但近程作用的N-body并行算法总体上来说可以分为3种:粒子分解法、力分解法和区域分解法。下面分别对这三种方法进行简单的介绍。

3.1 粒子分解法 Atom Decomposition(AD)

粒子分割法是将所有粒子平均分配给各处理器,每台处理器在计算前所分配的粒子是随机的,在空间中没有任何关系。各处理器只负责计算属于它的 N/P(N 为总的粒子数,P 为处理机数)个粒子的受力,并由此更新它们的位移和速度信息。在分子动力学的模拟中,比较典型的计算就是分子间作用力的 $N \times N$ 的矩阵 F,粒子分割法可视为每台处理器计算该矩阵的 N/P 行(如图5所示)。但进行粒子受力计算时,粒子需要知道分别属于其它处理器的另

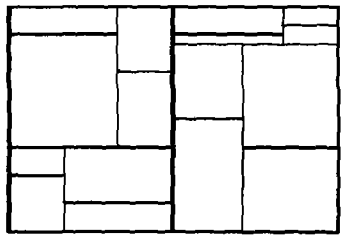


图 3 ORB 中的空间分区

Fig.3 The space partitioning partition in ORB scheme

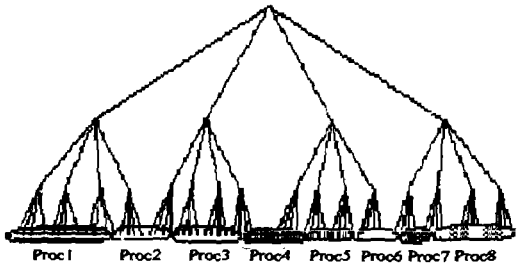


图 4 CostZone 中分级树的划分

Fig.4 Tree partition in CostZone scheme

外 N-1 个粒子的信息,因此需要所有处理器进行粒子位置的全交换操作。这就意味着每一时间步,各处理器都要从所有其他处理器处获得更新的粒子信息,这称为全局通讯。如果考虑牛顿第三定律的话,则位于不同处理器的粒子间相互作用可以只计算一次,将受力计算减少了一半,但处理器间的通信负荷要增加,因为需要将计算的力传递给相应的处理器。粒子分解算法需要处理器间的全局通信,但总的通讯量和粒子数 N 相关而独立于处理器台数 P,因此限制了为获得高的计算效率而使用的处理器数量。该算法易于编程,适合粒子个数较少的问题,但由于需要全局的通信,因此该算法不具有可扩展性。

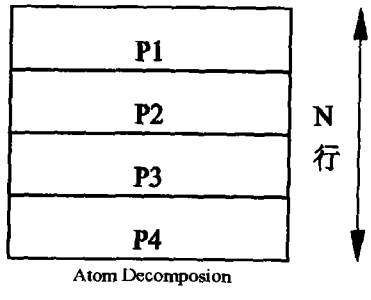


图 5 每处理器计算力矩阵的 N/P 行

Fig.5 Each processor calculates the N/P rows of force matrix

3.2 力分解法 Force Decomposition(FD)

矩阵的块分解法在线性代数的并行算法中是很常见的,Steve Plimpton 首先将这种方法引入到近程作用的分子动力学模拟中[12],即力分解法。简单的说,力分解法是把力矩阵 F 按块分割,而不是象粒子分解法那样,简单的按行来分割。每台处理器分配的是 F 矩阵的一个 $N/\sqrt{P} \times N/\sqrt{P}$ 的子块(如图 6 所示),这就意味着,为了进行受力计算,每台处理器仅需知道 $N/\sqrt{P}$ 粒子的信息,因此通讯和内存的开销相对于粒子分割法要减少 $\sqrt{P}$ 倍。计算粒子受力时,处理器只需要同所在行和列的处理器进行通信,如图 4 中,处理器 6 只需要和 5、7、8、2、10、14 进行通信,即可完成受力计算。力分解法同粒子分割法一样,为了优化计算性能,对物理问题的几何形状要求并不严格,且易于程序实现,但也较适合粒子数较少的情况。

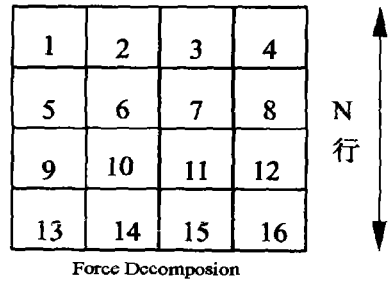


图 6 处理器计算 F 的 $N/P^{1/2} \times N/P^{1/2}$ 子块

Fig.6 Each processor calculates the $N/P^{1/2} \times N/P^{1/2}$ blocks of force matrix

3.3 区域分解法 Spatial Decoposition(SD)

区域分解法是将物理区域分成和处理器数目相同的子区域。每个时间步,各处理器计算自己区域内的所有粒子的受力、速度和位移,当粒子运动到新的子区域时,就把它分配给新的处理器。为了计算其内的粒子的受力,处理器只需知道临近子区域中粒子的信息即可。因此,区域分解法和 AD、FD 比起来,在本质上是局部的。子区域的大小跟粒子数、处理器数和物理区域的形状有关。区域分解法对物理区域形状的要求比 AD、FD 要严格,因为对不规则形状很难划分成大小相等的区域,并且相邻处理器间的通信比较复杂。另外,SD 编程的工作相对于 AD、FD 要复杂一些,但在粒子数较多时,采用 SD 算法可以获得较好的计算性能,比较适合基于消息传递的可扩展并行计算。

Plimpton 在多种并行计算环境下,粒子间作用力以 Lennard-Jones 势为基准,对这三种算法进行了对比[12],研究了它们的适用情况。当通讯的开销和

计算的相比可忽略时,宜选用 AD 算法,因为算法上的简单可以抵消通信效率带来的额外开销。典型的情况如处理器的数目较少,或者是作用力的计算比较复杂,因为此时的计算时间占了支配地位。在其它情况下,FD 算法比 AD 计算速度要快,因为当处理器的数目固定时,AD 和 FD 的计算规模都跟粒子数 N 成线性关系,这意味着在给定的处理器下,AD 和 MD 的并行效率都和粒子数无关。但是,AD 算法的通信时间随处理器数目的改变不变化,而 FD 算法则减少了处理器数的平方根倍,所以计算时间要少。SD 算法的计算规模和粒子数之间不是呈线形关系,这主要是由于处理器的计算和通信只是限定在一个局部区域。当粒子数较少时,通信和其它的开销比较明显,计算效率低,但粒子数较多时,计算效率则渐进的优化。另外,有一点需要指出的是,以上情况都是在假设负载平衡的条件下获得的。区域分解法相对于其他算法更容易引起负载的不平衡,而此时的计算效率就会大打折扣。

4 结论与展望

N-body 模拟在科学研究和工程应用中有着广泛的应用前景,根据其特点研究新的并行计算方法或是对现有算法做一些改进,以提高并行效率,减少计算时间来满足实际需求,无疑具有重要的意义。针对 N-body 问题的特点,要求并行算法能够合理地规划系统资源,满足以下要求。

4.1 良好的通信结构

并行计算的各处理器之间不可避免地要交换信息,因此通信结构的改善,会大大减少由于通信所产生的开销。这就要求处理器间数据和相关参数的传递要少,尽量实现数据的局部化,并且选择适当的通信模式。另外,通过实现通信和计算时间的重叠,也意味着通信开销的降低。

4.2 计算负载在各处理器间的平衡

N-body 模拟中的负载不平衡首先和问题本身有关,如粒子的非均匀分布以及边界条件的影响,以及由于模拟的动态性引起的相邻时间步处理器的负载变化可能会很大。另外也和处理器系统有关,如数据访问的延迟,操作系统的影响等。因此,根据处理器的计算能力和问题的具体特点,需要采取相应的措施来实现计算负载在各个处理器上的平衡,以减少由于同步要求而引起的各个处理器之间相互等待。

4.3 可扩展算法

算法可扩展,即计算性能随着问题规模的大小和有效资源的变化基本上不变。随着系统中的粒子数目的增加或减小,或是由于处理器个数的改变,算法都应能保持高效的计算性能。

References

1 Barnes J, Hut P. A hierarchical O(N log N) force calculation algorithm. Nature, 1986, 324:446 - 449.

2 Greengard L, Rokhlin V. A fast algorithm for particle simulations. Journal of Computational Physics, 1987, 73:325 - 348.

3 Anderson C. An implementation of the fast multipole method without multipoles. SIAM, J Sci Stat Comput, 1992, 13(4):923 - 947.

4 William Theodore Rankin. Efficient parallel implementation of multipole based *n*-body algorithms, PhD thisis, Duke University, 1999.

5 Guy Blelloch, Girija Narlikar. A practical comparison of N-body algorithms. In Dimacs Implementation Challenge Workshop, 1994, October.

6 Board J, Schulten K. The fast multipole algorithm. Computing in Science & Engineering, 2000, January/February:56 - 59.

7 Warren M S, Salmon J K. Astrophysical N-body simulations using hierarchical tree data structures. In supercomputing'92-Proceedings Minneapolis, 1992, Nov:570 - 576,16 - 20.

8 *Dubinski* J. A parallel tree code. New Astronomy, 1996, 1:133 - 147.

9 Franklin M, Govindan V. The N-body problem: distributed system load balancing & performance evaluation. Proc 6th Inter Conf on Parallel & Distributed Processing, 1993, Oct:252 - 262.

10 Singh J P, Holt C, Hennessy J L, Gupta A. A parallel adaptive fast multipole method. In Proceedings of Supercomputing 93, 1993, November:54 - 65.

11 Almojel A I. The implementation and performance evaluation of N-body gravitational simulation algorithm on high-performance computers. Computers and Electrical Engineering, 2000, 26:297 - 316.

12 Plimpton S. Fast parallel algorithms for short-range molecular dynamics. Journal of Computational Physics, 1995, 117:1 - 19.