耗散粒子动力学平衡并行算法的实现

赵 英 佟 林
(北京化工大学 信息科学与技术学院 信息中心, 北京 100029)

摘 要: 提出了一种耗散粒子动力学(DPD)的平衡并行(BPDPD)算法,解决了传统 DPD 程序计算时间过长的问题。通过动态的划分模拟空间实现了节点间的负载均衡,并且运用子节点数据本地重建机制有效减少了节点通信的数据量。对锚定蛋白质的聚集做了模拟,实验结果显示,该算法有效的减少 DPD 模拟的时间,并在 10 节点的集群上得到了 6.3 的加速比。

关键词: 耗散粒子动力学; MPI 程序设计; 并行计算; 分子模拟; 锚定蛋白质
中图分类号: TP399

引 言

耗散粒子动力学(DPD)模拟是基于介观体系的分子模拟方法。介观体系介于原子级的微观尺度和人们肉眼可以看到的宏观尺度之间,是连接微观世界和宏观世界的桥梁。DPD 模拟研究的对象往往是大分子和聚合物,如 Pan 等[1]用 DPD 模拟来发现角落诱捕对 DNA 的分离不产生影响;Pivkin 等[2]用 DPD 来模拟血小板的聚合过程。但传统的串行 DPD 程序一般要执行很长的时间才能给出结果,计算时间少则几天,多则几周,甚至更长。解决这一问题最有效的办法就是将原有的串行 DPD 程序并行化,使其在多台计算机组成的集群上进行运算,以达到缩短计算时间的目的。如 Sims 等[3]用并行化的方法在 16 个节点组成的 linux 集群上得到了 4.3 倍的加速比。然而,并行化程序难以解决的问题就是节点间的负载均衡问题以及节点通信数据量过大问题。

为了解决负载均衡问题,本文提出了一种平衡的并行 DPD 模拟(BPDPD)算法,并采用 MPI 进行并行化编程。BPDPD 算法的执行效率要明显高于串行 DPD 程序以及固定划分计算区域的并行 DPD 程序,达到了提高执行速度,平衡节点负载的目的。

收稿日期: 2012-03-20
基金项目: 国家"973"计划(2011CB706900)
第一作者: 男,1966 年生,教授
E-mail: zhaoy@mail.buct.edu.cn

1 耗散粒子动力学

1.1 DPD 的基本理论

在 DPD 模拟中,每个基本粒子都受到 3 种基本力的影响,保守力(F_C)、耗散力(F_D)和随机力(F_R),所以每个 DPD 基本粒子所受的力(f_i)是上述 3 种力的合力

$$f_i = \Sigma_{j\neq i} F_{ij}^C + F_{ij}^D + F_{ij}^R \tag{1}$$

DPD 模拟中 3 种基本力的作用范围受截断半径 r_c 的影响。当两个粒子的距离超过 r_c 时,认为这两个粒子之间不存在基本力的作用。DPD 中所有的 3 种力由式(2)~(4)给出

$$F_{ij}^C = \begin{cases} a_{ij}\left(1-\dfrac{r_{ij}}{r_c}\right)\hat{r}_{ij} & (r_{ij}<r_c) \\ 0 & (r_{ij}>r_c) \end{cases} \tag{2}$$

$$F_{ij}^D = -\gamma_{ij}\omega^D(r_{ij})(r_{ij}\cdot v_{ij})r_{ij} \tag{3}$$

$$F_{ij}^R = \sigma_{ij}\omega^R(r_{ij})\theta_{ij}/\sqrt{\Delta t}\cdot r_{ij} \tag{4}$$

式(2)~(4)中,a_{ij}是相互作用参数;$r_{ij}=r_j-r_i$,是粒子 i 与粒子 j 之间的距离;$v_{ij}=v_j-v_i$,是粒子 i 与粒子 j 的相对速度;r_c是截断半径;γ_{ij}是摩擦系数;σ_{ij}是噪声幅度;ω^D和 ω^R是与 r_{ij}有关的权重函数。

通过式(1)~(4),就能得到每个粒子的加速度,由此能进一步算出下一步迭代计算所需要的各个粒子的速度和位置。DPD 模拟程序就是对上述过程不断迭代的一个过程。

1.2 锚定蛋白质的聚集

本实验是用来模拟锚定蛋白质和磷脂细胞膜之间的相互作用。锚定蛋白质的粒子和磷脂分子不仅受 DPD 传统的 3 个力影响,也受到键力(F_S)和键角

力(F_φ)的作用。上述两种力是不存在截断半径的,这就是说,无论相距多远的两个粒子,只要它们之间存在键的作用,它们彼此就对对方有键力和键角力的作用。

模拟实验的初始状态是在一个 50 × 50 × 30 大小的盒子里,有一个双层磷脂分子组成的生物膜,并且生物膜被水分子包围着。在保持总体密度不变的前提下,一些锚定蛋白质会被垂直的插入生物膜中,整齐的排列在生物膜表面。

本实验所预期的结果是,在许多步迭代计算之后,锚定蛋白质分子在与磷脂生物膜的相互作用下倾斜,并与邻近的锚定蛋白质分子聚集,形成若干较大的蛋白质区域。

2 耗散粒子动力学的平衡并行算法

2.1 并行算法的基本设计

在整个 DPD 模拟程序中,对于力的计算是耗时最久的部分[4],所以,对这一部分进行并行化编程可以有效的减少程序的计算时间。

DPD 模拟并行化的基本方法是基于空间划分的办法。粒子在限定的空间内四处运动,因此,并行程序可以把长方体的模拟空间沿 x 方向分成几部分,如图 1 所示。

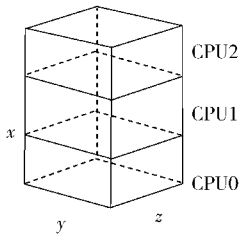


图 1 并行 DPD 的区域划分

Fig. 1 Partition of the parallel DPD

图 1 中长方体的模拟区域沿 x 轴被等分成了 3 部分,每个部分的计算任务被分配给了一个单独的 CPU。

这种区域划分的方法存在一个问题,就是处在每个区域边界附近的粒子所受的力不能被正确计算,因为位于各个子区域边界附近的粒子不仅受到本区域内粒子的作用,还受到与它相邻的区域中靠近边界处的粒子的作用

为了解决这一问题,划分子区域的时候,在等分的同时,在每个子区域的两端各加入一个假想层。假想层是各个子区域边界附近一个特定区域的副本。为了满足 DPD 模拟中要求的周期性边界条件[5],第一个分区的第一个假想层是从最后一个分区拷贝得来的,最后一个分区的最后一个假想层是从第一个分区拷贝得来的。图 2 中给出了二维情况下的示例。

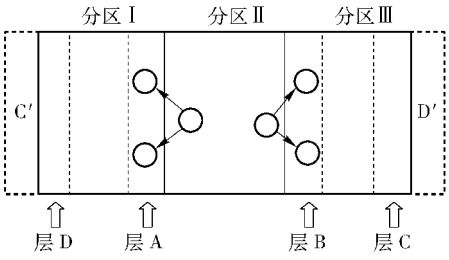


图 2 假想层的划分

Fig. 2 Division of the ghost cell

由图 2 可以看到,在分区 I 中的层 A 和分区 III 中的层 B 是分区 II 的假想层,而分区 I 的假想层 C' 是从分区 III 的层 C 复制得到的,同样的,分区 III 的假想层 D' 是从分区 I 的层 D 复制得到的。因为 DPD 3 种基本力的截断半径是 1[5],所以在本例中假想层沿 x 轴的长度就是 1。

假想层的引入,使得每个子区域范围内的粒子所受的 3 种基本力都能被正确的计算了。但是如 1.2 节所提到的,键力和键角力的作用是不存在截断半径的,这使得两个属于不同分区的粒子间也会存在相互作用。在这种情况下,单纯的基于空间划分的方法又无法正确的计算各个粒子的受力情况了。为了解决这一问题,在本文中,当一个粒子被分配给一个计算节点以后,可能与它存在键力及键角力作用的所有粒子的信息都会被传递给该节点。这样做可以解决无法正确计算键力和键角力的问题,但会产生一些多余的数据通信量,因为在传递数据时,传递的是可能与节点所属区域内粒子存在键力及键角力的粒子,也就是说有些与该区域内的粒子没有相互作用的粒子信息也被传递给了该节点,这就造成了通信的冗余。为了确定数据通信过程中的冗余率,现在以一个由 540000 个粒子组成的蛋白质磷脂分子相互作用的实验为例,在 9 个节点上进行模拟实验,各个节点接收的粒子数及冗余率如表 1 所示。

在表 1 中,编号是计算节点的编号,p(i)是对应节点实际参与计算的粒子数,p(a)是实际传递给该节点的粒子数,p(e)是对应节点通信时产生的冗余粒子数,η 是该节点的传输冗余率。由表 1 可知,每个节点的数据冗余率是很低的,也就是说上述解决

表 1　节点传递粒子数冗余律

Table 1　Redundancy rate of data transmission

编号	p(i)	p(a)	p(e)	η/%
0	81566	82951	1385	1.67
1	81339	82654	1315	1.59
2	80817	82089	1272	1.55
3	80680	81883	1203	1.47
4	80632	81731	1099	1.34
5	80913	82197	1284	1.56
6	80998	82292	1294	1.57
7	81138	82476	1338	1.62
8	54150	55503	1353	2.43

键力和键角力计算问题的方法并不会产生过多的通信负担,因此,这种解决键力和键角力计算问题的方法是可行的。

2.2　模拟区域的平衡划分方法

2.1 节中所介绍的并行 DPD 模拟的基本方法已经可以正确的解决 DPD 模拟并行化问题,但还没有解决计算节点负载不均的问题,因为模拟区域中的粒子并不是绝对均匀分布的,所以只是简单的在空间上平均划分各个节点的计算区域,是无法实现负载均衡的。为了解决这个问题,论文首先将模拟区域沿 x 轴分成许多更小的小区间,每个小区间称为一个原子区域,在本文中,原子区域沿 x 轴的长度设定为 0.5。在划分原子区域的同时用一个向量 NumBex 来记录每个原子区域中的粒子数,这样就大致获得了粒子沿 x 轴的分布情况。获得向量 NumBex 的具体方法为

```
if( Xx(J). lt. (floor( Xx(J)) +0. 5)) then
      tmpxx = floor( Xx(J)) * 2 - 1
else
      tmpxx = floor( Xx(J)) * 2
endif
NumBex( tmpxx) = NumBex( tmpxx) + 1
```

其中 Xx 是粒子的 x 坐标。

在划分计算区域之前还需要对平均每个节点所处理的粒子数做一个预测,得到预测值 E。E 可通过式(5)得到

$$E = \alpha(n/k + (2n)/x) \tag{5}$$

式(5)中, n 是总粒子数,k 是计算节点的个数,x 是模拟区域沿 x 轴的长度,α 是一个常量。

当程序读取数据的时候,向量 NumBex 同时就被初始化了。在迭代计算开始之前,计算节点的计算区域将会首先根据向量 NumBex 被划分,基本的划分方法是初始时,节点 i 的计算区域沿 x 轴的区间下限是节点 $i-1$ 的计算区域沿 x 轴的区间上限。节点 i 的计算区域的大小是一个原子区域。节点 i 所处理的粒子数就是 NumBex 向量中对应位置原子区域的粒子数。然后依次将与其相邻的原子区域不断地合并进节点 i 的计算区域,并将对应的粒子数累加进节点 i 所处理的粒子数中,直到节点 i 所处理的粒子数大于等于预测值 E,或节点 i 的计算区域沿 x 轴的上限超过了整个模拟区域沿 x 轴的上限。

该程序具体实现为

```
XxL(1) = 0
j = 1
Do i = 1, size
  XxR(i) = XxL(i)
  Do while(. true. )
    if(j/2. 0. gt. Boxlx) then
      XxR(i) = Boxlx
      Exit
    endif
    N(i) = N(i) + NumBex(j)
    If(N(i). gt. E) then
      XxR(i) = j/2. 0
      exit
    endif
    j = j + 1
  enddo
  XxL(i + 1) = XxR(i)
Enddo
```

其中 N(i)是分配给节点 i 的粒子数,XxR(i)是节点 i 的计算区域沿 x 轴的上限。由于本文所做的空间划分是沿 x 轴进行的,所以在划分时只考虑了粒子沿 x 轴方向上的分布状况。

尽管通过这种方法划分出的计算区域不是平均划分的,但这样的划分让计算节点的负载更加均衡了。多次的实验结果显示,当式(5)中的 α 为 0.933 的时候,节点间的负载最为平衡。下面以一个在 $20\times20\times20$ 区域中做的实验为例,每一步迭代各个节点粒子数的方差如图 3 所示。

图 3 中横坐标是迭代计算的步数,纵坐标显示的是在 5 个计算节点上运行并行模拟程序得到的粒子数方差。标三角的线给出的是用平均划分的方法

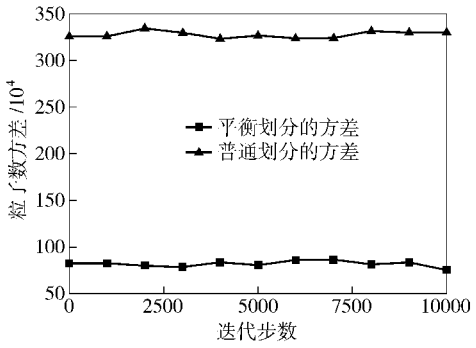


图 3 不同划分方法粒子数方差的比较

Fig. 3 Variance of the particle numbers with different division method

得到的结果,标方块的线给出的是用平衡划分的方法得到的结果。从图 3 中不难看出,平衡划分的方法能够有效的提高各个节点的负载均衡性。

2.3 数据的重建机制

当并行程序在集群上运行的时候,节点间传递的数据量是十分可观的。以一个 540000 个粒子组成的蛋白质聚集实验为例,其要传递的属性信息如表 2 所示。

表 2 粒子需传递的属性

Table 2 Attributes of one particle

属性	类型	是否变化
coordinate(x)	REAL(8)	是
coordinate(y)	REAL(8)	是
coordinate(z)	REAL(8)	是
velocity(x)	REAL(8)	是
velocity(y)	REAL(8)	是
velocity(z)	REAL(8)	是
force(x)	REAL(8)	是
force(y)	REAL(8)	是
force(z)	REAL(8)	是
type	INTEGER	否
particle	INTEGER	否
kmol	INTEGER	否
aspring	INTEGER	否
smolecule	INTEGER	否
tmolecule	INTEGER	否

在 Fortran 语言中,一个 REAL(8)类型的变量占用 8 个字节,一个 INTEGER 类型的变量占用 4 个字节,一个粒子的所有属性就占用了 96 个字节,这样一来 540000 个粒子,总共需要传递的数据量是 49.42 MB。但在这些属性中,并不是所有的属性随着模拟的进行都会发生变化,在表 2 中,"是否变化"一项为"否"的属性都不随着模拟的进行而变化,是粒子的固有属性。既然固有属性是不变的,那就不需要每次都把这些信息都传递一遍。

数据重建机制就是在计算节点需要通信时,用粒子的编号来代替粒子的固有属性进行通信。当程序从配置文件中读取实验初始化数据时,每个节点通过对配置文件的一次性读取,都会在本节点中建立一个粒子的固有属性表,并以粒子编号为索引。当一个节点接收到从主节点传来的粒子编号时,就可以通过这一编号直接在本地重建粒子的固有属性信息,这样就减少了节点间的通信量。粒子编号的类型是 INTEGER 型,只占用 4 字节,540000 个例子总共需要传递的信息就减少为 39.13 MB,比之前提到的 49.42 MB 减少了 20.84%。

2.4 平衡并行算法的进一步讨论

如今应用广泛的开源分子模拟工具在划分模拟空间时所用的方法各异,主要的方法包括第八壳(eighth shell)的方法以及中点法(midpoint)。第八壳的方法通过对空间进行三维的划分实现并行,其区域调整策略是对每一个维度进行独立的调整。由于最初该算法是根据 MD 的并行化提出的,在空间的尺度上较小,实现这一过程比较简单。但 DPD 的空间尺度较大,采用这一方法就需要大量的通信和计算工作,DPD 的区域划分需要更加粗粒化的划分策略。中点发需要每个节点都具有发送和接收两个独立的网络连接,对于硬件条件的依赖较大。本文提出的平衡的并行 DPD 模拟在划分区域的粒度上是可以调节的,只要调整原子区域的大小即可,原子区域越小区域划分的粒度就越细,相应的区域划分的开销也就越大。本算法对硬件的依赖很小,在普通的计算机组成的集群上也可以运行,易于实现。

3 实验结果分析

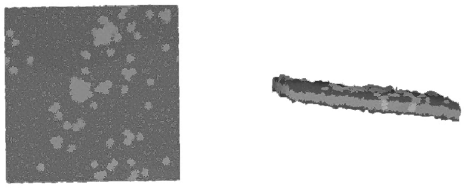
使用锚定蛋白质的聚集实验来测试程序的性能,模拟的具体问题是在 50 × 50 × 30 的空间里,540000 个粒子的运动情况。

3.1 测试环境

测试环境是由多台计算机组成的集群,集群中的每一个节点都有一个 Intel x5650 2.66 GHz 的 6 核 CPU,以及 4 GB 的内存。计算节点的操作系统都为 Linux,使用的编程语言是 Fortran,使用的并行编程工具是 MPICH2。

3.2　串行 DPD 程序的运行结果

串行程序在实验集群上的一个节点运行了 100000 步迭代以后，程序得出了最终的模拟结果，用时 67 h 5 min 28.28 s，如图 4 所示。



(a) 正面图　　(b) 侧面图

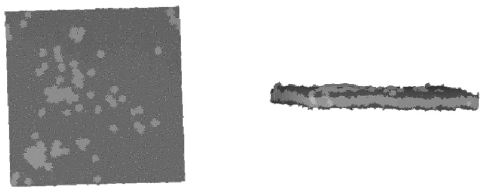
图 4　串行程序模拟结果

Fig. 4　Results of the serial simulation program

图 4 中，浅色区域代表锚定蛋白质，可以看到锚定蛋白质聚集到了一起，并形成了若干大片的蛋白质区域。

3.3　普通并行 DPD 程序的实验结果

普通并行程序的实验结果如图 5 所示。



(a) 正面图　　(b) 侧面图

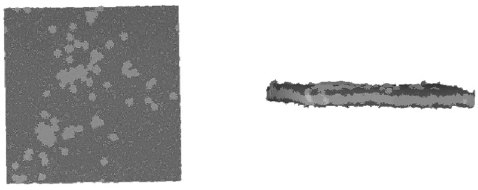
图 5　普通并行程序的模拟结果

Fig. 5　Results of the normal parallel simulation program

在 10 个节点上，平均划分计算区域的并行 DPD 程序模拟相同问题所得到的结果，如图 5 所示。普通并行程序迭代 100000 次，总共运行了 12 h 2 min 32.36 s。

3.4　平衡的并行 DPD 程序的运行结果

平衡的并行在同样的测试环境下，模拟同样的问题，在 10 节点并行的情况下，迭代 100000 次，运行了 10 h 38 min 55.43 s 后，程序得出了最终的结果，如图 6 所示。



(a) 正面图　　(b) 侧面图

图 6　BPDPD 的运行结果

Fig. 6　Results of the balanced parallel simulation program

从图 6 中可以看到，与串行的 DPD 程序类似，锚定蛋白质同样聚集成了若干较大的蛋白质区域，可见平衡并行程序的实验结果是正确的。

综合前面 3 次实验得到的结果，3 种不同的模拟程序同样执行 100000 步迭代模拟，串行程序耗时 67 h 5 min 28.28 s，折合 241528.28 s，普通并行程序耗时 12 h 2 min 32.36 s，折合 43352.36 s，平衡并行程序耗时 10 h 38 min 55.43 s，折合 38335.43 s。由此可得普通并行程序在 10 节点上的加速比是

$$S_C = 241528.28/43352.36 = 5.6$$

平衡的并行程序在 10 节点上的加速比是

$$S_W = 241528.28/38335.43 = 6.3$$

由此可见，BPDPD 程序可以有效的减少计算时间，并在普通并行程序的基础上将计算时间缩短了 11.57%。

3.5　平衡并行程序的扩展性分析

将同样的模拟问题在测试集群上用平衡并行程序进行模拟，分别使用 2 节点、4 节点、6 节点和 10 节点参与计算，得到加速比与计算节点数之间的关系如图 7 所示。

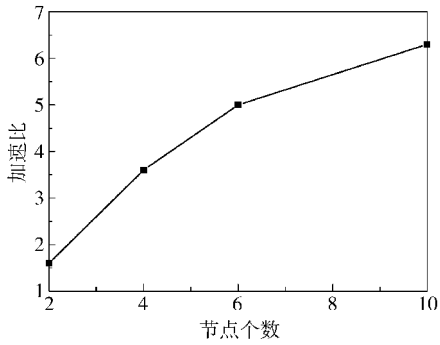


图 7　加速比与节点数的关系

Fig. 7　Relationship between the speed-up ratio and the CPU number

从图 7 中可以看出平衡并行程序的加速比随着节点数目的增加而提高，但增长曲线逐渐趋于平滑，主要是由于以下几点原因。首先随着节点数目的增加，划分区域的工作量会加大，整合计算结果的工作量也会增加。其次由于假想层的加入，无论子区域被分的多小，都要加入总长度为 2 的假想层，对于这部分的计算不会随着节点数的增加而减少。所以无限的增加节点个数并不利于提高计算效率，因为对于假想层内的粒子受力情况的计算是不正确的，是无效的计算，提高节点个数也就相当于提高了无效计算在总计算量中的比重，造成计算资源的浪费。

4 结束语

本文提出了耗散粒子动力学的平衡并行算法,并加以了实现。与传统的并行 DPD 程序相比,BPDPD 在节点负载上更加平衡,数据重构机制的引入使得节点间通信的数据量得到了减少。与串行的 DPD 程序相比,BPDPD 程序在 10 个节点上的加速比是 6.3,显著的提高了程序的计算速度。

参考文献:

[1] Pan H, Ng T Y, Li H, et al. Dissipative particle dynamics simulation of entropic trapping for DNA separation [J]. Sensors and Actuators A: Physical, 2010, 157(2): 328-335.

[2] Pivkin I V, Richardson P D, Karniadakis G E. Effect of red blood cells on platelet aggregation[J]. IEEE Engineering in Medicine and Biology Magazine, 2009, 28(2): 32-37.

[3] Sims J S, Martys N. Simulation of sheared suspensions with a parallel implementation of QDPD[J]. Journal of Research of the National Institute of Standards and Technology, 2004, 109(2): 267-277.

[4] Yue Y, Zhao Y, You G H, et al. Parallelization of dissipative particle dynamics simulation[C]// 2009 International Joint Conference on Computational Sciences and Optimization, IEEE Computer Society, 2009: 18-22.

[5] Groot R D, Warren P B. Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation[J]. J Chem Phys, 1997, 107(11): 4423-4435.

Balance parallelization of dissipative particle dynamics simulation

ZHAO Ying TONG Lin

(Center for Information Technology, College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract: In this paper, a balance parallelization method for dissipative particle dynamics (BPDPD) simulation is presented. By dynamically calculating the range of each computed node, the load of the cluster becomes balanced, and by means of data rebuilding, the amount of communication data is also reduced. An anchored protein gathering problem was simulated in BPDPD and showed that the BPDPD can effectively reduce the computing time and afford a speed-up ratio of 6.3 on a cluster with ten nodes.

Key words: dissipative particle dynamics; MPI; parallel computing; molecular simulation; anchored protein