Journal of Computer Applications
计算机应用,2012,32(1):163 - 166
ISSN 1001-9081
CODEN JYIIDU
2012-01-01
http://www.joca.cn

文章编号:1001 - 9081(2012)01 - 0163 - 04
doi:10.3724/SP.J.1087.2012.00163

基于 OpenMP 的分子动力学并行算法的性能分析与优化

白明泽[1,2],程　丽[2],豆育升[2,3*],孙世新[1]

(1.电子科技大学 计算机科学与工程学院,成都 610054;　2.重庆邮电大学 高性能计算与应用研究所,重庆 400065;
3. Department of Physical Sciences, Nicholls State University, Thibodaux, Louisiana 70310, USA)

(*通信作者电子邮箱 douys@cqupt.edu.cn)

摘　要:为提高分子动力学模拟在共享内存式服务器上的计算速度,对基于 OpenMP 的分子动力学并行算法(Critical 方法)进行了性能分析与优化。通过在多核服务器上的测试,以及加速比和并行效率的计算分析了 Critical 方法的并行性能,进而提出优化的三角形方法。所提方法中每个线程所计算的粒子数固定,且粒子数目呈阶梯状上升,使得各线程能够错时到达临界区。从而使程序在临界区的闲置时间比 Critical 方法减半,加速比明显提高。

关键词:分子动力学;并行计算;多核中央处理器;OpenMP;临界区

中图分类号:TP399;O641　　**文献标志码**:A

Performance analysis and improvement of parallel molecular dynamics algorithm based on OpenMP

BAI Ming-ze[1,2], CHENG Li[2], DOU Yu-sheng[2,3*], SUN Shi-xin[1]

(1. *School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu Sichuan* 610054, *China*;
2. *Institute of High Performance Computing and Application, Chongqing University of Posts and Telecommunications, Chongqing* 400065, *China*;
3. *Department of Physical Sciences, Nicholls State University, Thibodaux, Louisiana* 70310, *USA*)

Abstract: To enhance the computing speed of the molecular dynamics simulations on the shared memory servers, the performance of parallel molecular dynamics program based on Open Multi-Processing (OpenMP) approach with the critical section method was analyzed and improved. After testing performance on a multi-core server, as well as the calculations of speedup and parallel efficiency, an optimized triangle method was developed. In this method, stationary atom sets were assigned to threads respectively, and the number of atoms increased stepwise, which made the threads arrive at critical sections at different time. The triangle method can efficiently halve the idle time in critical sections and therefore can significantly enhance the parallel performance.

Key words: molecular dynamics; parallel computing; multi-core Central Processing Unit (CPU); Open Multi-Processing (OpenMP); critical section

0　引言

分子动力学(Molecular Dynamics, MD)是一种应用广泛并可在粒子级别模拟固态、液态物质的主要计算方法之一。它将粒子视为经典粒子,通过求解各粒子的运动方程得到不同时刻粒子的空间位置信息,进而根据统计物理方法计算出物质微观结构、热力学性质及平衡输运性质等在实验条件下较难得到的宏观属性。MD 模拟已被广泛应用于物理科学、材料科学、生命科学等领域[1]2-3,[2]7-8,成为一种重要的科学研究方法。MD 模拟对计算资源有着巨大的需求,是高性能计算的典型应用之一。在空间尺度上,为了获得更精确的宏观性质,MD 模拟体系的粒子个数可能多达几百万甚至更多;在时间尺度上,MD 模拟的步长时间通常取几个飞秒(fs)[2]60,数百万步的计算也仅模拟了现实过程中约几纳秒的时间。为了能快速完成 MD 计算任务,人们通常让多台计算机同时完成一个计算任务,这需要首先对串行 MD 程序进行并行化。常用的 MD 软件如 LAMMPS[3]、NAMD[4]、CHARMM[5]、AMBER[6]等都有并行版本,而且都是基于消息传递模型如 MPI(Message Passing Interface)[7]。

随着计算机硬件的快速发展,节点采用对称多处理(Symmetric Multi-Processing, SMP)机的集群系统逐渐成为高性能计算机的主流体系结构。2011 年 6 月的 TOP 500 名单中有 80% 以上的高性能计算机采用了这种结构[8]。这种集群存在两级速度差异较大的通信路径:一是节点内部的共享存储通信,它可通过对共享内存上的数据读写来实现数据通信,速度较快;另一种是在介于节点间的网络上进行的消息传递通信,速度较慢。运行于 SMP 集群上的消息通信模型通常忽略了这一区别,将 SMP 主机节点内的多路 CPU 也视为不同的节点进行消息通信。这样的并行编程模型没有完全利用 SMP 集群的特点,在性能上还有一定的改进空间[9]。

一种 MPI + OpenMP 的混合模型于近年来被采用以改进基于纯消息通信机制的并行编程模型[10]。MPI + OpenMP 混合模型在节点间采用基于 MPI 的消息通信机制,在节点内则采用 OpenMP[11]实现共享内存的并行计算。这种混合并行编程模型具有两个明显优点:一是在节点内部采用共享内存数据读写后,通信更方便速度更快;二是将整个任务划分的粒度变大,可以有效减少子任务之间也就是节点之间的通信量[9]。针对 MD 的混合并行算法也被提出并实现[12],研究表

收稿日期:2011-07-15;**修回日期**:2011-09-11。　**基金项目**:国家自然科学基金资助项目(21073242)。

作者简介:白明泽(1982 -),男,重庆酉阳人,讲师,博士研究生,主要研究方向:并行算法设计、分子动力学模拟;　程丽(1983 -),女,山西大同人,硕士研究生,主要研究方向:并行算法设计;　豆育升(1953 -),男,陕西礼泉人,教授,主要研究方向:计算化学、高性能计算;　孙世新(1940 -),男,湖北孝感人,教授,博士生导师,主要研究方向:并行与分布式计算。

明混合并行编程模型可以有效加快 MD 并行,但这些方法都采用比较简单的共享内存并行方式,没有深入地对线程级并行进行性能分析与优化。

随着多核(Chip MultiProcessors, CMP)技术的发展快速,在 SMP 节点上的计算核心数量越来越多,导致在混合模型中的线程数量也随之增多。线程级并行算法的性能将对整个程序并行性能产生很大的影响。对基于 OpenMP 的 MD 并行算法运行性能分析和优化,可以帮助提高 MD 算法的混合并行算法的效率。本文分析了一种常见的基于 OpenMP 的 MD 并行方法的并行加速比和效率,得出在普通 X86 SMP 服务器上该方法的并行加速比和效率会受到线程数限制的结论。进而提出了一种优化方法,理论分析和测试结果表明优化方法的并行加速比有明显提高。

1 基于 OpenMP 的分子动力学并行算法

1.1 OpenMP 并行编程模型

OpenMP 目前已成为并行程序设计的主流模型之一,它具有易用性好、支持增量并行等优点[13]。OpenMP 利用编译制导,运行时库函数(Run-Time routines)和环境变量等工具快速实现串行代码的线程级并行工作。OpenMP 目前只支持 C/C ++ 和 Fortran 语言,它通过添加编译制导语句到串行代码里,并由支持 OpenMP 的编译器根据这些编译制导语句和运行时库函数产生相应的可执行并行代码。整个并行化过程比较简单易行,不需要对串行代码做较大的改动。

OpenMP 是基于线程的并行编程模型[13],遵循 Fork-Join 执行模式。OpenMP 并行程序的运行过程如图 1 所示。进程中的主线程负责执行串行代码,直到进入并行区(parallel region)开始多线程并行执行并行区代码;主线程在进入并行区时创建或唤醒一个从线程队列(fork),从线程与主线程一起并行执行并行区代码;当并行区代码执行完之后,从线程被退出或挂起,只有主线程继续执行串行区代码(join)。运行在不同处理器上的线程之间可通过共享变量来实现数据的交换。

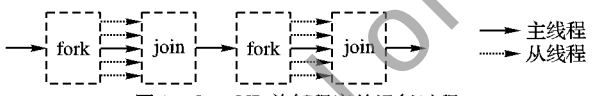

图 1 OpenMP 并行程序的运行过程

1.2 MD 串行算法

为了减少计算量,串行 MD 程序会采用一些方法来降低计算量,尤其是对于只计算短距离作用的 MD 程序。截断半径 R_C 在很多方法中被采用以减少短距离作用力的计算:当两个粒子的间距 $R > R_C$ 时,它们之间的作用力变得非常小,往往可以忽略。基于截断半径人们发展出了两种不同的算法来加速模拟。一种是近邻表法(Verlet NeighborList)[1]146-149,[2]54-58,它通过建立列表保存每个粒子周围截断半径 R_C 区域内的相邻粒子,然后根据列表计算作用在这个粒子上的力;使用近邻表法可以在计算一个粒子所受力时避免搜索整个模拟空间。另外由于粒子是运动的,在计算时每隔一定时间步就需要更新列表。另外一种是元胞法(Link-cell)[1]149-152,该法将模拟体系分割成许多元胞(cells),元胞边长 d 等于或略大于截断半径 R_C,这样进行作用力计算时只需在周围 26 个元胞和当前元胞中查找相邻粒子。这两种方法也常常被结合使用,即采用近邻表法计算作用力,但是在构建近邻表时采用元胞法搜寻可能的近邻粒子,从而降低近邻表构建任务的时间复杂度[14]。

基于近邻表的串行分子动力学模拟的步骤[1]6-7,[2]22-23如图 2 所示:首先给定系统初始位形,然后开始 N 步循环;每隔一定时间步数更新一次粒子的近邻表;每一个时间步都根据近邻表计算粒子之间作用力;最后积分牛顿方程获得粒子新坐标。在整个模拟过程中,粒子的受力计算是最费时的部分。

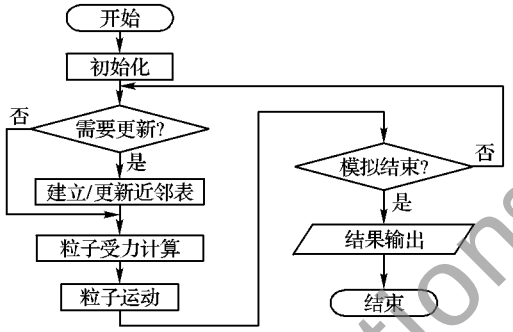

图 2 串行分子动力学计算步骤

1.3 基于 OpenMP 的 MD 并行算法

使用共享内存模型编程时,数据竞争是个不可回避的问题。数据竞争是指使用共享内存模式编程时,在多个线程可并发读写一个变量的情况下会导致计算结果随读写次序不同而不同[15]。举一个简单的例子:如果两个线程 T_1, T_2 都需要给变量 X 增 1(假设 X 初值为 0),即两个线程并发执行 $X = X + 1$。如果 T_1 首先读取到的 X 为 0,相加后得到 $X = 1$ 并准备写回 X;但是如果 T_2 在 T_1 写回前就进行了读取 X 的操作,则读取到的值也为 0,它进行运算后得 $X = 1$,并将这个最终值 1 写回 X。这与设想中的并行执行结果($X = 2$)不一致,即存在数据竞争。

基于共享存储并行模型的 MD 算法在引入牛顿第三定律时会发生数据竞争。一般的 MD 串行程序为减少计算量,在计算粒子 i 受到粒子 j 的作用力 f_{ij} 时会根据牛顿第三定律将它同时分别加/减到两个粒子的力元素 $f(i)$ 和 $f(j)$ 上,从而避免重复计算。所以在使用 OpenMP 的对这类 MD 程序进行并行化时,会发生这样的情况:线程 1 计算 $f(i)$ 时扫描 i 的近邻表获得粒子 j 坐标,从而计算粒子 i 与粒子 j 之间的作用力 f_{ij};按照牛顿第三定律它需要将 f_{ij} 会同时更新到 $f(i)$ 与 $f(j)$ 上,而恰好此时另一线程所计算的粒子也是 j 的邻居,它也在更新 $f(j)$;这样就产生了数据竞争,从而导致程序计算结果出错。白树仁等采用不使用牛顿第三定律的策略来避免数据竞争[16]。但是这样做效率较差,因为计算力是相当耗时的部分,同一个力需要计算两遍则会浪费较多计算资源。

Tarmyshov 等通过在代码设置 critical section(临界区)的方法,实现了无数据竞争的牛顿第三定律策略(记为 Critical 方法)[17]。该方法在串行版本已有的力数组 $f()$ 基础上为每个线程增加了一个私有副本 $f\_local()$ 数组。在多线程程序执行过程中,$f()$ 是全局数组,被所有的线程共享;该线程计算出来的作用力 f_{ij} 在被写入 $f(i)$ 的同时也被线写入该线程所私有的 $f\_local(j)$。这些私有的 $f\_local()$ 数组元素需要最终汇总到 $f()$ 数组里,为了避免数据竞争,Critical 方法在临界区里进行这个工作。图 3(a) 演示了 Critical 方法中“粒子受力计算”部分的时间线,图中大小箭头线分别代表该方法中各线程进行作用力计算和副本更新所经历的时间线。图中左边部分(即大箭头线区)代表 omp do 执行区,程序在此负责计算各个粒子(假设编号为 i)与它们的相邻粒子(假设编号为 j)之间的作用力,并将力更新到当前粒子的 $f(i)$ 和 $f\_local(j)$;右边部分(即小箭头线区)代表临界区,在此程序采用类似于串行

的方法将各个线程私有的 *f_local*( ) 矩阵加到 *f*( ) 中。从图 3(a) 可以看出,在线程较多时临界区的执行时间与线程数成正比,同时 omp do 执行区会相应缩短,从而整个程序的性能会受到较大的影响,本文将在第 2 章对该算法性能进行分析。

2　基于 OpenMP 分子动力学并行算法性能分析

2.1　Critical 方法的时间线

临界区的存在导致并行程序受到较大影响。每个线程在临界区内做的工作都是相同的:将 *f_local*( ) 数组加到 *f*( ) 数组里。模拟体系的大小决定了这两个数组的大小,即使线程数 N_{th} 发生变化,临界区内工作量保持一致;因此每个线程在临界区内的时间也应相同,在图 3(a) 中反映出来就是小箭头线长度相同。并行程序在临界区内的墙上时间是每个线程在临界区内的时间 T_c 和线程数 N_{th} 的乘积,因此临界区墙上时间与 N_{th} 成正比。同时随着 N_{th} 增大,每个线程能分到的工作量将减少,导致 omp do 执行区的墙上时间 T_{omp} 与线程数 N_{th} 成反比。图 3(a) 演示的线程数 N_{th} 为 8,可以分析出此时并行程序的闲置时间。从图中看出每个线程都要在临界区闲置 7 个 T_c,则总的闲置时间是 $56T_c$。推广到一般情况下,可得总闲置时间为 $N_{th}(N_{th}-1)T_c$,其中在临界区前的闲置时间为 $N_{th}(N_{th}-1)T_c/2$。为了简便起见,本文中只提线程数 N_{th},默认线程数等同于计算核心数。

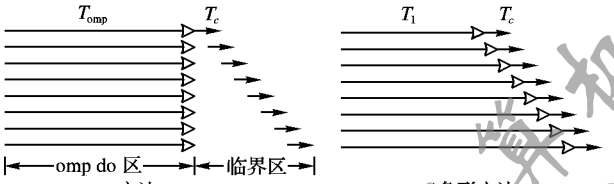

图 3　两种并行方法中粒子受力计算部分的时间线演示

2.2　Critical 方法性能分析

Critical 方法粒子受力计算部分的理论加速比可以根据上述分析计算出来(在本文中如非特别指明,讨论的均只是粒子受力计算部分)。在本文第 4 章中描述的实验条件下,体系大小为 171 500 时测得 T_c 与 T_s 的比值 $k=0.0094$(根据实验测试,k 值受体系大小影响较小),假设 OpenMP 的开销(包括线程建立/唤醒等工作)忽略不计,则 $T_{omp}=T_s/N_{th}$,$T_c=kT_s$。基于上述假设,可得 Critical 方法的加速比 S_c 如下:

$$S_c=\frac{T_s}{T_{omp}+N_{th}*T_c}=\frac{1}{\frac{1}{N_{th}}+N_{th}*k} \tag{1}$$

可见 Critical 方法的加速比与线程数成反比。进一步计算 Critical 方法的并行效率:

$$E_c=\frac{S_c}{N_{th}}=\frac{1}{1+k*N_{th}^2} \tag{2}$$

从式(2) 看出,Critical 方法的并行效率与线程数 N_{th} 的平方成反比,可知该方法的可扩展性较差。就本文中的实验平台而言,因为 k 的值约为 0.01,当 $N_{th}=10$ 时,效率约为 50%,当 $N_{th}>30$ 时,效率将低于 10%,体现出较差的可扩展性。

3　优化方法

3.1　优化方法的时间线

为了尽量减少在临界区所闲置的时间,本文提出一种优化方法。Critical 方法的重要特点是临界区内只有一个线程在进行计算,其他进入过和未进入过临界区的线程都在闲置状态。如果能让未进入过临界区的线程不必在临界区等待而继续执行力计算的任务,则可以避免临界区前的等待。根据前面对 Critical 方法的分析,在临界区前的闲置时间为 $N_{th}(N_{th}-1)T_c/2$。也就是说,新方法可以让闲置时间减少为原来的一半。

图 3(b) 演示了改进方法的时间线(图中箭头线含义与图 3(a) 相同)。在第一个线程开始进入临界区后的任意时刻内只有一个线程在临界区内,同时其他未进入过临界区的线程则还在继续计算力,从而避免等待。可以看出原来呈矩形的临界区时间线被压缩成三角形状,因此本文将新方法称为三角形法。由图 3(b) 可以看出该方法可以减少线程在临界区的闲置时间,以线程数 $N_{th}=8$ 为例:线程 1 要闲置 $7T_c$,线程 2 要闲置 $6T_c$,线程 3 要闲置 $5T_c$,……,总的闲置的时间由原来的 $56T_c$ 减少到 $28T_c$,时间节省了一半,与前面分析一致。

3.2　优化方法的性能分析

三角形方法需要给每个线程分配固定的工作量,本工作中采用为每个线程分配固定粒子数的策略。与传统 Critical 方法在计算过程中动态地为每个线程分配粒子不同,三角形方法在进入受力计算并行区前就给每个线程分配固定数目的粒子集,粒子数目呈阶梯状上升从而实现使每个线程在不同时间依次进入临界区的目的。粒子数目的确定需先计算各个线程所应进行的力计算的时间长度。这里采用与 Critical 性能分析时同样的假设:$T_c=kT_s$,设 $k'=1/k$,以及 OpenMP 的并行开销忽略不计。图 3(b) 中的 T_1 表示第一个线程的力计算时间线,其他线程的力计算时间长度均在前一个线程的基础上加上 T_c。基于以上假设,可得式(3):

$$(N_{th}*T_1)+(1+2+3+\cdots+N_{th}-1)*T_c=T_s \tag{3}$$

解得:

$$T_1=4T_c\left(\frac{k'}{N_{th}}-\frac{N_{th}-1}{2}\right) \tag{4}$$

再根据 T_1 的值求得线程 i 的粒子数:

$$N_i=\frac{T_1+(i-1)T_c}{T_{af}}=\left(\frac{k'}{N_{th}}-\frac{N_{th}-1}{2}+i-1\right)*\frac{T_c}{T_{af}} \tag{5}$$

其中:T_{af} 为每个粒子进行力计算所用的时间,T_{af} 不必求出就可得到 N_i 之间的比例,再根据比例计算出各自的值即可。T_c 的数值可由 Critical 方法计算:所有线程在临界区的持续时间的平均值即为 T_c。通过实验测试线程数不同时 T_c 值稳定,这与前面的分析一致。

值得注意的是,在粒子数的计算中有一个隐含的条件,那就是 $T_1>0$,由式(4) 解得 N_{th} 的取值范围为:

$$0<N_{th}<0.5(1+\sqrt{1+8k'}) \tag{6}$$

从式(6) 可以看出线程的最大数目受到 k' 的限制,并不能随意取得。

从图 3(b) 可已得到三角形方法的力计算部分的墙上时间为 $T_1+(N_{th}-1)T_c$,可得该方法的加速比 S_a 为:

$$S_a=\frac{T_s}{T_1+(N_{th}-1)T_c}=\frac{1}{\frac{1}{N_{th}}+\frac{N_{th}+3}{2}k} \tag{7}$$

对比式(7) 和式(1) 可以发现三角形法的加速比同样与 N_{th} 成反比,但是相比 Critical 方法来讲加速比有较大提高。

4　结果与讨论

为验证优化后算法的正确性和有效性,本文在一个具有 16 个计算核心的 SMP 服务器上进行了测试。服务器型号为浪潮英信 NF560D2,具体配置如下:4 颗 Intel Xeon 四核处理

器 E7420(主频 2.13 GHz),内存 32 GB;操作系统为 CentOS 5.4(Linux 核心 2.6.18 -194 x86_64);编译器为 Intel Fortran Compiler 11.1 版。测试体系的粒子数分别为 171 500(晶格数 35×35×35)和 665 500(晶格数 55×55×55),势能函数选用 LJ 函数。

图 4 演示了体系大小为 171 500 时两种方法的力计算时间部分加速比。由于本平台上测试的 k' 值为 107,根据式(6)计算得到 N_{th} 的最大值为 15。所以在测试时,没有测试三角形方法(Triangle)在线程数大于 15 的数据。从图 4 看出,当线程数大于 7 时,三角形方法的力计算部分加速比明显高于 Critical 方法,与前面的分析相符。在线程数为 11 时,两种方法加速比分别为 5.17 和 5.61,三角形方法比 Critical 方法提高了约 9 个百分点。

从图 4 还可看到,两种方法分别在 N_{th} 为 10 和 11 的时候加速比达到各自的最大值,之后加速比就随着线程数增加反而降低,符合式(1)和式(7)的推论。这表明在进行 MD 并行计算时,更多的线程并不能一直提高计算速度,只有选择合适的线程数才能获得最快的计算速度。

前面只是针对两种方法的力计算部分的加速比进行分析,为了观察两种方法对整个 MD 程序加速比的影响,本文分别测试了两种体系下的 MD 程序整体加速比(坐标计算和宏观信息统计部分通过对循环体添加 omp do 的方式进行并行化),如图 5 所示,35 和 55 代表两种不同大小的体系。可以看出随着线程数增大,相同体系下三角形方法始终要比 Critical 方法更好,当然较大体系的加速比会比较小体系的略低。

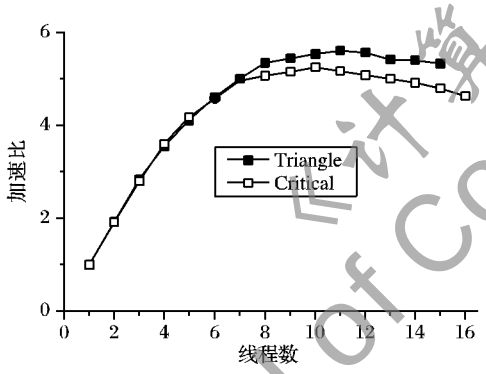

图 4 两种方法中力计算部分的并行加速比

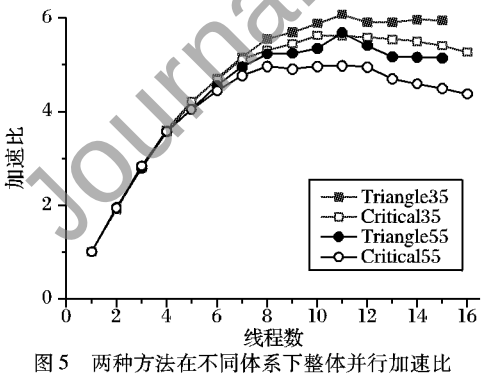

图 5 两种方法在不同体系下整体并行加速比

5 结语

采用牛顿第三定律的分子动力学模拟串行程序在用 OpenMP 并行化时会遇到数据竞争,本文对这一概念进行了阐述和分析。Critical 方法是一种应用广泛的用来解决数据竞争的方法,本文在 X86 平台的 SMP 服务器上对它的并行性能进行了详细分析。理论分析和测试结果表明该方法并行效率和可扩展性不好,在线程超过一定数目时加速比反而下降。本文在分析其并行性能的基础上给出了改进的三角形方法,实验结果表明该方法可以明显提高程序的并行性能。本文所提出的三角形方法可以帮助科研人员在应用 OpenMP 并行化 MD 程序时获得较高的加速比,同时还能指导他们根据具体计算平台选取合适的线程数,避免因线程较多而速度更慢所导致的计算资源浪费。

参考文献:

[1] ALLEN M P, TILDESLEY D J. Computer simulation of liquids [M]. New York: Oxford Clarendon Press, 1991: 2-3.

[2] RAPAPORT D C. The art of molecular dynamics simulation [M]. Cambridge: Cambridge University Press, 2004: 7.

[3] PLIMPTON S, HENDRICKSON B. A new parallel method for molecular dynamics simulation of macromolecular systems [J]. Journal of Computational Chemistry, 1996, 17(3): 326-337.

[4] KALE L, SKEEL R, BHANDARKAR M, *et al*. NAMD2: Greater scalability for parallel molecular dynamics [J]. Journal of Computational Physics, 1999, 151(1): 283-312.

[5] BROOKS B R, BROOKS C L, III, MACKERELL A D, *et al*. CHARMM: The biomolecular simulation program [J]. Journal of Computational Chemistry, 2009, 30(10): 1545-1615.

[6] CASE D A, CHEATHAM T E III, DARDEN T, *et al*. The Amber biomolecular simulation programs [J]. Journal of Computational Chemistry, 2005, 26(16): 1668-1688.

[7] Message Passing Interface Forum. MPI: A message-passing interface standard, version 2.2 [M]. Stuttgart: High Performance Computing Center Stuttgart, 2009.

[8] Top 500 supercomputer sites. Architecture share for 11/2010 [EB/OL]. [2010-11-11]. http://www.top500.org/stats/list/36/archtype.

[9] CHORLEY M J, WALKER D W. Performance analysis of a hybrid MPI/OpenMP application on multi-core clusters [J]. Journal of Computational Science, 2010, 1(3): 168-174.

[10] 李鸿健,白明泽,唐红,等.混合并行技术在激光化学反应模拟中的应用[J].计算机应用,2010,30(6):1687-1689.

[11] AKHTER S, ROBERTS J. Multi-core programming: Increasing performance through software multi-threading [M]. Santa Clara: Intel Corporation, 2004: 139-175.

[12] CHORLEY M J, WALKER D W, GUEST M F. Hybrid message-passing and shared-memory programming in a molecular dynamics application on multicore clusters [J]. International Journal of High Performance Computing Applications, 2009, 23(3): 196-211.

[13] CHAPMAN B, JOST G, van der PAS R. Using OpenMP: Portable shared memory parallel programming [M]. Cambridge: MIT Press, 2007: 23-34.

[14] GUPTA S. Computing aspects of molecular dynamics simulation [J]. Computer Physics Communications, 1992, 70(2): 243-270.

[15] NEITZER R H, MILLER B, P. What are race conditions? some issues and formalizations [J]. ACM Letters on Programming Languages and Systems, 1992, 1(1): 74-88.

[16] BAI SHU-REN, RAN LI-PING, LU KUI-LIN. Parallelization and performance tuning of molecular dynamics code with OpenMP [J]. Journal of Central South University of Technology: English Edition, 2006, 13(3): 260-264.

[17] TARMYSHOV K B, MÜLLER-PLATHE F. Parallelizing a molecular dynamics algorithm on a multiprocessor workstation using OpenMP [J]. Journal of Chemical Information and Modeling, 2005, 45(6): 1943-1952.