文章编号:1001 - 9081(2008)09 - 2371 - 04

并行模式下分子散射模型的求解

黄忍冬[1],彭 舰[1],冯 灏[2]
(1. 四川大学 计算机学院,成都 610065; 2. 四川大学 物理科学与技术学院,成都 610065)
(lab330@ gmail. com)

摘 要:串行模式下,求解复杂电子分子碰撞的散射问题存在单机处理时间长、内存耗用大等缺陷,而并行处理的思想可显著降低其计算时间,是解决该问题的有效途径。在主-从结构的并行模型基础之上设计并实现了电子分子碰撞散射的并行算法,提出了有效的优化步骤。实验结果得到了满意的加速比和并行效率,验证了该方案的可行性与正确性。

关键词:电子分子散射;主从结构;加速比;并行效率
中图分类号: TP301.6 **文献标志码**:A

Solution of molecule scattering model based on parallel mode

HUANG Ren-dong[1], PENG Jian[1], FENG Hao[2]
(1. *School of Computer Science, Sichuan University, Chengdu Sichuan* 610065, *China*;
2. *School of Physics Science and Technology, Sichuan University, Chengdu Sichuan* 610065, *China*)

Abstract: The solution of Electron-molecule scattering model based on serial mode meets two challenges: long running time and large memory requirement. Parallel computing which would reduce the workload of one computer is a useful method to meet the challenge. Based on master-slave parallel model structure, several optimized approaches were introduced and the parallel algorithm for Electron-molecule scattering was realized. Results of the experiments obtain satisfied speedup and parallel efficiency, and prove the feasibility and validity of this solution.

Key words: electron-molecule scattering; master-slave; speedup; parallel efficiency

0 引言

电子与分子的碰撞是原子分子散射领域的一个重要部分。N2 分子是自然界广泛存在的双原子分子,它与低能电子之间的散射过程在气体放电物理、激光控制以及天体物理中都具有重要的应用价值,长期以来一直受到物理学家的重视[1]。在计算电子和 N2 分子碰撞散射所产生的相互作用势能和散射截面的过程中,伴随着海量数据的处理,单机的计算能力有限,远远无法满足研究的需要。在单机性能瓶颈出现的情况下,利用集群(Cluster)进行并行计算的需求尤为迫切。目前,大规模并行计算技术已经在复杂分子计算领域得到广泛的应用,并取得显著成功。

集群是并行或分布计算机系统的一种实现,是由一组完整的计算机互联而成的,能作为一个单独的统一计算资源来使用的系统。相对于 SMP(对称多处理) 和 MPP(大规模分布式处理) 类型的大型主机而言,使用相同等级的集群计算机,价格仅为大型主机的 1/ 6[2]。消息传递接口(Message Passing Interface,MPI)[3]是目前最流行的并行编程标准,具有移植性好、功能强大、效率高等优点,而且支持不同的软硬件平台。利用消息传递机制,除了支持多核并行,SMPD 并行[4]之外,还能便利实现多个节点的并行处理。

本文在给出分子散射模型基础之上,针对分子散射模型的具体特征,采用主从模式和分治策略[4]设计实现了电子分子碰撞散射的并行算法,通过优化碰撞散射并行计算的关键步骤,得到一种高效、可扩展的并行处理模式。同时,也为大规模科学计算提供了一条新思路。

1 碰撞散射物理串行模型

低能电子与分子散射的量子效应十分显著,必须通过求解整个体系的非相对论性量子力学方程:积分方程形式的 Lippmann-Schwinger 方程和微分方程形式的 Schrödinger 方程,才能正确解释低能散射现象[5]。

Lippmann-Schwinger 方程:

$$u(x) = u0(x) - (k^{\Lambda}2) \cdot \int (Gk(x-y)u(y)q(y)) \quad (1)$$

Schrödinger 方程:

$$[\hat{T}_e + \hat{T}_m^{(n)} + \varepsilon_{a0} + V_{\text{int}} - E] \times \Phi_{a0}(\vec{r}, \vec{R}_n) = 0 \quad (2)$$

在方程(2)中,散射波函数 $\Phi_{a0}(\vec{r}, \vec{R}_n)$ 含有核的坐标 $\vec{R}_n$ 和入射电子的坐标 $\vec{r}$,因此它同时描述了电子的运动与分子中核的所有振动-转动结构[6]。采用固定核取向近似,不考虑核的转动结构,并把入射电子的角度部分以球谐函数 $\{Y_l^{\Lambda}(\vec{r})\}$ 描述,对所有核坐标 R 和角度部分积分,最后得到电子径向运动的振动耦合散射方程:

$$[\frac{d^2}{dr^2} - \frac{l(l+1)}{r^2} - 2V_{vl,vl}^{\Lambda}(r) + k_v^2] u_{vl,v_0l_0}^{\Lambda}(r) =$$

收稿日期:2008 - 03 - 31;**修回日期**:2008 - 06 - 04。
基金项目:国家自然科学基金资助项目(10474068);四川省应用基础研究(2008JY0027);四川省科技攻关项目(07GG006-040)。
作者简介:黄忍冬(1983 -),男,四川成都人,硕士研究生,主要研究方向:并行计算与分布式处理、中间件; 彭舰(1970 -),男,四川成都人,副教授,博士,主要研究方向:并行计算与分布式处理、中间件; 冯灏(1975 -),男,重庆人,博士,主要研究方向:分子结构和分子光谱、低能电子与分子的激发散射。

$$2\sum_{v',l'\neq v,l}[V^{\Lambda}_{vl,v'l'}(r)u^{\Lambda}_{v'l',v_0l_0}(r)] \tag{3}$$

其中耦合势能项,也就是我们的目标如下:

$$V^{\Lambda}_{vl,v'l'}(r) = \langle \phi_v(R)Y^{\Lambda}_l\hat{r} \mid V_{st}+V_{pol}+V_{ex} \mid \varphi_{v'}(R)Y^{\Lambda}_{l'}\hat{r}\rangle \tag{4}$$

电子和 N2 分子的相互作用势能是非球对称的势能[7],求解径向方程时的散射耦合势能主要包含三种相互作用的贡献:静电势、交换势和电子关联极化势[1]。静电势源自于入射电子与未变形的分子电荷分布之间的库仑作用,包括入射电子与分子中的电子、入射电子与核的库仑作用;交换势能属于非局域的相互作用势,表示入射电子与分子中束缚电子的交换作用;在低能电子散射中极化势则描述了十分重要的电子相关效应[8]。

由式(4)可以看出,在传统的串行模型中,为了求解入射电子的径向散射方程,还必须把以上求出的所有静电、交换和极化作用势用振动波函数和球谐函数做耦合运算。在做耦合时,静电、交换和极化势能不仅相互依赖,故只能串行计算相互作用势能。由于还需处理大量矩阵运算,故常常耗费大量时间。因此,引入并行计算的需要显得尤为迫切。

2 碰撞散射并行算法设计与分析

2.1 算法分析

并行算法是指将一个过程分配到不同子进程在不同处理器上同时处理的计算方法。目前,普遍使用的并行算法模式有:主从式、SPMD、数据流水线、分治策略等。并行算法设计通常分为四个步骤:首先将给定问题划分成若干子任务,划分方法可以使用任务分解法或数据分解法;其次分析任务之间的通信需求,然后使用组合方法,在尽可能保持灵活性的同时,减少通信和开发成本;最终以最小化执行时间为目标将诸任务分配给各处理器。

在前文所述的串行模型中,各项势能计算的相互依赖导致并行计算难以实施。为了实现物理并行模型,必须通过化简降低它们之间的耦合度,具体步骤如下:

将式(4)角度部分单独积分去掉以简化公式,首先将总的相互作用势展开:

$$V_{\text{int}} = V_{st}+V_{pol}+V_{ex} = \sum_{\lambda} v_{\lambda}P_{\lambda}(\cos\theta) \tag{5}$$

$$v_{\lambda} = v_{\lambda}^{(st)}+v_{\lambda}^{(pol)}+v_{\lambda}^{(ex)} \tag{6}$$

将式(5)代入式(4),并利用球谐函数的正交归一性[8],可以将耦合势能项变为:

$$V^{\Lambda}_{vl,v'l'}(r) = \sum_{\lambda}^{\lambda_{\max}} g_{\lambda}(ll';\Lambda)\omega^{\lambda}_{v,v'}(r) \tag{7}$$

其中,角度部分的耦合系数为:

$$g_{\lambda}(ll';\Lambda) = \left(\frac{2l'+1}{2l+1}\right)^{1/2}C(l'\lambda l;\Lambda 0)C(l'\lambda l;00) \tag{8}$$

振动耦合势能项:

$$\omega^{\lambda}_{v,v'}(r) \equiv \langle \phi_v \mid v_{\lambda} \mid \phi_{v'}\rangle = \int_0^{\infty}\phi_v^*(R)v_{\lambda}(r,R)\phi_{v'}(R)\mathrm{d}R \tag{9}$$

至此,通过将耦合势能项分解成角度部分耦合系数与振动耦合势能项的乘积形式,各项势能之间的耦合度大大降低。由于角度部分耦合系数的计算较简单,故只需要完成对振动耦合势能项的并行运算就能顺利得到耦合势能项。

2.2 区域分配与主从计算

根据本物理模型的数据局域性和积分运算的特点,可方便利用主从模式和分治策略实施并行处理。本文采用一维区域分割的方式将 R 的积分区间划分成连续的若干子区间,为保证每次传递的数据量最小,在一维区域分割时采用平均分割,使分割后的子区域大小尽可能相同,以保证各节点机的负载平衡,基本思想为。

1) 由于 R 的积分区间$[0,\infty)$是一个开区间,无法平均分配,故考虑将此区间先分割成连续的两个区间$[0,M]$和$[M,\infty)$,其中 $M>0$。

2) 通过划分获得的$[0,M]$是一个闭区间,故可以完成对此区间的平均分配。将$[0,M]$区间划分成 N 个等间距的子区间,每个子区间的宽度为 M/N,将对 R 在$[0,\infty)$开区间中的积分运算转变为对 N 个子区间的积分运算。

3) 将 N 个子区间的积分求和,可得到一个关于变量 M 的函数 $F(M)$,通过对函数 $F(M)$ 求极限($M\to\infty$),便可得到振动耦合势能项的结果。

这种区域分割方法具备很好的可并行性,并行算法如下:

1) 初始化 MPICH2 和硬件环境,启动 Master 进程和指定的 Slave 进程数;

2) Master 进程按照上述区域分割的策略将划分目标区域划分成 N 个宽度为 M/N 的子区间;

3) Master 进程将各个子区间中关于 R 的积分运算所需数据发送给 Slave 进程;

4) Master 和 Slave 同时在各自的内存空间中完成宽度为 M/N 的子区间积分,Slave 在运算完成时利用 MPI_Send 方法将中间结果返回给 Master 临时保存;

5) Master 继续向空闲的 Slave 派发运算任务;

6) 当所有计算节点运算完成后,由 Master 进程利用 MPI_Reduce 方法将接收到中间结果进行归纳求和,得到 $[0,M]$ 区间的积分结果 $F(M)$;

7) *Master* 进程对 $F(M)$ 求极限($M\to\infty$),求得振动耦合势能并保存最终结果。

2.3 算法设计与实现

主体算法描述如图 1 所示。

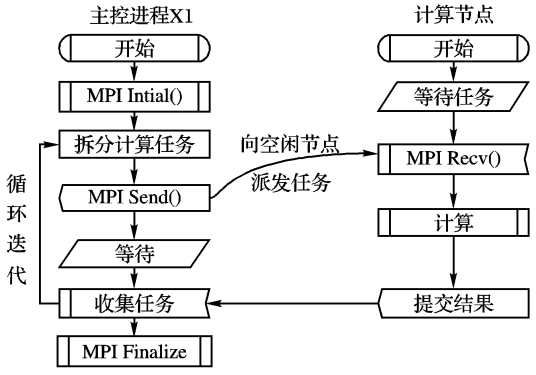

图 1 主体算法流程

程序部分代码如下:

```
Master:
MPI_Init();                                          //MPI 初始化
MPI_Comm_size( MPI_COMM_WORLD, &numprocs);
                                                     //返回所有的进程数
MPI_Comm_rank( MPI_COMM_WORLD, &id);
                                                     //返回本进程在指定通讯器中的 ID 号
MPI_Get_processor_name( processor_name, &name);
                                                     //返回运行本进程的处理器的名称
fprintf( stderr, "process %d on %s\n", id,  processor_name);
if ( rank = =0)
```

```
{startwtime = MPI_Wtime();}                    //设定起始时间
MPI_Bcast(&x[0], &f[0], &n);
                          //将需要参与并行计算的数据广播出去
Calculating (f);                          //各节点分别计算部分 f
MPI_Reduce(&f, &sum);          //将所有计算出的 f 值规约求和
if (rank = =0)
{endwtime = MPI_Wtime();                    //设定完成时间
 printf("f is %.4f\n", sum);                      //输出结果
 printf("wall clock time = %f\n", endwtime-startwtime);
                                          //输出计算时间
 MPI_Finalize();                          //MPI 并行计算结束
}
```

Slave 程序与 Master 大体类似,只需要将程序初始化的部分和数据广播的部分去掉即可。该算法通过数据并行策略可以使更多的处理机保持忙碌状态,并带来了更灵活的可扩展性。该算法并行时间复杂度为 $O(n)$,比串行程序的数量级要小,并且消息的传递只涉及 R 以及各个处理机计算得到的部分和,通信开销比较小,适合并行化。其中关键之处在于对 R 积分区间的划分,本文采取细粒度[9]的划分方式,根据每个计算节点的进程号,将连续的数据块分配给某个进程集中处理,使所有计算节点均在高效率下保持满负荷运转。

2.4　矩阵求解

在实际的 N2 分子碰撞散射计算当中,存在大量的多重循环和矩阵运算,而其中的数据相关性并不大,这也为利用并行策略提高效率带来了便利。目前矩阵相乘并行算法有多种,它们主要取决于矩阵的划分,有行列划分、行行划分、列列划分、列行划分、对角线划分以及 Cannon 算法[10]。

本文采用对角线划分的方法来完成矩阵乘法。为了方便描述,设矩阵乘法 $\boldsymbol{A}\times\boldsymbol{B}=\boldsymbol{C}$,其中 $\boldsymbol{A}$ 是 $m\times j$ 阶矩阵,$\boldsymbol{B}$ 是 $j\times n$ 阶矩阵,$\boldsymbol{C}$ 是 $m\times n$ 阶矩阵,计算节点数量为 P,第 k 台处理机记为 $Pk(k=0,1,\cdots,p-1)$,并将矩阵 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$ 分别映射到一个 $P\times P$(设 $P=4$)的二维网格上,矩阵 $\boldsymbol{A}$、$\boldsymbol{B}$ 的二维网格映射如下:

$$\boldsymbol{A}=\begin{bmatrix}A_{00} & A_{01} & A_{02} & A_{03}\\ A_{10} & A_{11} & A_{12} & A_{13}\\ A_{20} & A_{21} & A_{22} & A_{23}\\ A_{30} & A_{31} & A_{32} & A_{33}\end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix}B_{00} & B_{01} & B_{02} & B_{03}\\ B_{10} & B_{11} & B_{12} & B_{13}\\ B_{20} & B_{21} & B_{22} & B_{23}\\ B_{30} & B_{31} & B_{32} & B_{33}\end{bmatrix}$$

将矩阵 $\boldsymbol{A}$、$\boldsymbol{B}$ 在二维网格的主对角线上的各子矩阵分配给处理机 $P0$,然后矩阵 $\boldsymbol{A}$ 采取基于对角线向左下方平移的子矩阵分配策略,而矩阵 $\boldsymbol{B}$ 采取基于对角线向右上方平移的子矩阵分配策略。矩阵 $\boldsymbol{C}$ 采取同样映射方式并在每个处理机中都存有一个备份,矩阵 $\boldsymbol{A}$、$\boldsymbol{B}$ 的各子矩阵依次存放在各处理机中的矩阵 $\boldsymbol{A}1$ 和 $\boldsymbol{B}1$ 中。表 1 给出了矩阵 $\boldsymbol{A}$、$\boldsymbol{B}$ 的所有子矩阵在各处理机中的分布存储情况。

表 1　矩阵 $\boldsymbol{A}$、$\boldsymbol{B}$ 的各子矩阵的分布存储情况

处理机	$\boldsymbol{A}1$				处理机	$\boldsymbol{B}1$			
$P0$	A_{00}	A_{11}	A_{22}	A_{33}	$P0$	B_{00}	B_{11}	B_{22}	B_{33}
$P1$	A_{01}	A_{12}	A_{23}	A_{30}	$P1$	B_{10}	B_{21}	B_{32}	B_{03}
$P2$	A_{02}	A_{13}	A_{20}	A_{31}	$P2$	B_{20}	B_{31}	B_{02}	B_{13}
$P3$	A_{03}	A_{10}	A_{21}	A_{32}	$P3$	B_{30}	B_{01}	B_{12}	B_{23}

在矩阵划分完毕并分配给各处理机后,以处理机 $P1$ 为例具体描述该方法的执行过程(各子矩阵的位置并没有发生真正的循环移动,只是通过设置指针的方式,使它的变化能够代表矩阵的循环移动)。

1)初始时处理机 $P1$ 中的矩阵 $\boldsymbol{A}1$、$\boldsymbol{B}1$ 为:A_{01}　A_{12}　A_{23}　A_{30},B_{10}　B_{21}　B_{32}　B_{03},将矩阵 $\boldsymbol{A}1$、$\boldsymbol{B}1$ 中的各对应子矩阵对应位置作乘,即 $A_{01}\times B_{10}$,$A_{12}\times B_{21}$,$A_{23}\times B_{32}$,$A_{30}\times B_{03}$,结果存放在 $\boldsymbol{C}_{00}$,C_{11},C_{22},C_{33} 中,然后将矩阵 $\boldsymbol{B}1$ 在各处理机间循环传递一次。

2)循环传递一次后处理机 $P1$ 中的矩阵 $\boldsymbol{B}1$ 为 B_{20}　B_{31}　B_{02}　B_{13},为保证各子矩阵可以相乘,需要将矩阵 $A1$ 中的各子矩阵向左循环移动一位,移位后变为:A_{12}　A_{23}　A_{30}　A_{01},然后将矩阵 $\boldsymbol{A}1$、$\boldsymbol{B}1$ 中的各对应子矩阵相乘,结果存放在 C_{10}　C_{21}　C_{32}　C_{03} 中,然后将矩阵 $\boldsymbol{B}1$ 继续在各处理机间循环传递一次。

3)此时 $P1$ 中的矩阵 $\boldsymbol{B}1$ 为 B_{30}　B_{01}　B_{12}　B_{23},需要再次将矩阵 $\boldsymbol{A}1$ 中的各子矩阵向左循环移动得到 A_{23}　A_{30}　A_{01}　A_{12},将矩阵 $\boldsymbol{A}1$、$\boldsymbol{B}1$ 中各对应子矩阵做乘,结果存放在 C_{20}　C_{31}　C_{02}　C_{13} 中,然后继续将矩阵 $B1$ 在各处理机间循环传递一次。

4)最后 $P1$ 中的矩阵 $\boldsymbol{B}1$ 为 B_{00}　B_{11}　B_{22}　B_{33} 再将矩阵 $A1$ 中的各子矩阵向左循环移动一次得到 A_{30}　A_{01}　A_{12}　A_{23},矩阵 $\boldsymbol{A}1$、$\boldsymbol{B}1$ 中的各对应子矩阵相乘,结果存放在 C_{30}　C_{01}　C_{12}　C_{23} 中。

经过以上 4 步,只需要再将所有处理机中的部分结果矩阵 $\boldsymbol{C}$ 累加起来,就将得到最终结果 C。传统串行计算矩阵乘法的时间复杂度为 $O(n\times n)$,通过并行复杂度降低为 $O(n)$,大大提高了运算效率,节省了大量的时间。该算法的外循环内部还存有一个 P 次内循环,如果将这个内循环进一步并行展开,那么该算法的并行度将会进一步提高。

3　实验及性能评估

3.1　实验环境

本文并行计算实验系统包括以下几个部分。

1)硬件环境:我们的集群系统是由实验室里的 8 台(CPU Intel Core2 2.0 GHz、MEM 1 GB)运行 Ubuntu 操作系统的主机构成,使用 100 Mbps 的交换机构建了一个 100 Mbps 带宽的采用 TCP/IP 协议的局域网。

2)软件环境:每台主机上都配置了 MPICH2[3] 的并行计算编程环境和相应的工具包,能够进行串行作业和并行作业的计算。

3)集群中间件:系统的主要部分,包含有作业提交模块、资源管理模块和作业调度模块。作业提交模块接受用户提交并行计算作业,并返回结果。资源管理模块负责对系统资源进行收集,管理和分配。作业调度模块则根据系统资源的状况对作业队列进行调度。

3.2　结果及性能评估

加速比和并行效率是最传统的并行算法评价指标。加速比定义为:

$$S_p=T_{opt}/T_p \tag{10}$$

其中:T_{opt} 为最优串行算法在单处理机上的运行时间;T_p 为并行算法在并行机上使用 P 台处理机所需的时间。相应的并行效率定义为:

$$E_p=S_p/P \tag{11}$$

其中 P 为处理机台数。并行效率反映加速比随节点数目的增加而增加的比例,因而,并行效率往往随着计算节点的增加而降低。以电子碰撞 N2 分子发生散射为例进行计算,求解

相互作用势能，由式(10)、(11) 可以计算出群并行计算的加速比和并行效率。为了测试并行算法的性能，设置了不同的计算节点数目，分别统计了各次的计算总耗时，如表 2 所示。

表 2 不同数量的处理机下的实验结果

节点数目	计算耗时/s
1	77314.13
2	46020.23
4	31816.46
6	28956.55
8	24312.57

表 1 对比了单机求解和多机并行求解的效率。可以看出使用区域分解法[11]在 MPI 环境下并行求解电子分子碰撞所产生的耦合势能问题有较高的加速比和较好的并行效率。说明使用区域分解法在基于 MPI 的并行环境下，对求解一些复杂的原子分子物理问题有着很好的效果和前景。图 2 和图 3 给出了并行计算的加速比和并行效率。

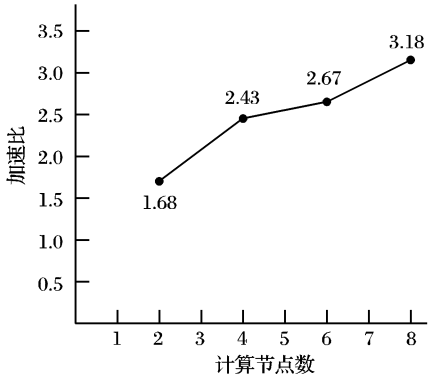

图 2 并行加速比

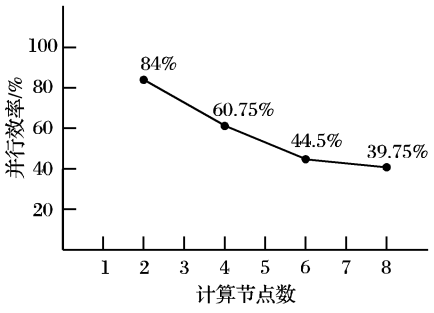

图 3 并行效率

通过图 2、3 可以看出，程序的加速比较理想，并行效率较高，随着节点数目的增加，计算耗时在减少，在计算节点从 1 个增加到 4 个时，表现得很明显，之后效果并不显著。同样，随着计算节点的增加，加速比一直在增加，但计算节点超过 4 个之后效果不显著。本例中，计算节点为 4 个以内的情况下并行效率是相当可观的，但是计算节点为超出 4 个时，并行效率并不乐观。由于在互相作用势能计算中存在一定的依赖关系，集群规模变大时，计算后期集群节点的闲置率将变高，这是本例节点数超过 4 个后加速比上升缓慢的一个原因。可以预见，在其他条件一定的情况下，积分区间的划分将存在一个优化的规模。而将各个独立区间进一步切分，以再次减小并行粒度，是克服空间并行的不足、提高并行效率的手段。为加快算法的执行效率，还可采用主频较高的处理器减少节点的工作负载或采用高速网络，减少消息延时等。

4 结语

依靠现有设备搭建集群平台，在原有模型中引入集群并行计算方法之后，作用势能和散射截面的计算速度得到了大幅度的提高。需要指出，一定的问题规模对应着一个最优的集群规模，在这一最优集群规模下存在着较理想的模型运算速度和较高的并行效率。如果集群规模与最优规模之间出现重大偏离就会严重影响运算速度和并行效率，如何确定最优规模是下一步研究的重点。

参考文献：

[1] LIU X, SHEMANKSY D E, CIOCCA M, *et al*. Analysis of the physical properties of the N_2 $c'\ ^1\Sigma_{+u}(0) - X^1\Sigma_{+g}(0)$ transition [J]. The Astrophysical Journal, 2005, 623: 579 - 584.

[2] 陈国良. 并行计算——结构、算法、编程: 修订版[M]. 北京: 高等教育出版社, 2003.

[3] MPICH2 Home Page [EB/ OL]. [2007 - 12 - 15]. http://www.mcs.anl.gov/mpi/mpich2.

[4] 都志辉. 高性能计算并行编程技术—— MPI 并行程序设计[M]. 北京: 清华大学出版社, 2001: 13 - 15.

[5] TAYLOR J R. Scattering theory [M]. New York: Wiley, 1972: 21 - 37.

[6] 申立, 戴伟, 刘秀英, 等. 低能电子与 N_2 分子碰撞的振动共振激发散射截面的密耦合研究[J]. 原子与分子物理学报, 2006, 23(2): 237 - 240.

[7] 刘国跃, 孙卫国, 冯灏, 等. 用 ECM 方法研究 N2 分子部分激发态的势能函数[J]. 原子与分子物理学报, 2004, 21(2): 255 - 259.

[8] 冯灏. 双原子分子势能函数和振动激发散射的理论研究[D]. 成都: 四川大学, 2001.

[9] IAN F. Designing and building parallel programs [M]. 北京: 邮电出版社, 2002.

[10] 陈国良, 安虹, 陈峻, 等. 并行算法实践[M]. 北京: 高等教育出版社, 2004: 458 - 466.

[11] 王小伟, 郭力. 近程作用分子动力学模拟的两级并行[J]. 计算机与应用化学, 2003, 20(5): 639 - 642.

(上接第 2370 页)

Keerthi 改进的 SMO 算法的优点，又提高了处理非平衡数据集的能力，如实验结果所示，在处理非平衡数据集 Abalone(17) 和 Yeast(5)时，都表现出较高的分类能力。

参考文献：

[1] PLATT J C. Sequential minimal optimization: A fast algorithm for training support vector machines[R]. Technical Reports MSR-TR-98-14, 1998.

[2] KEERTHI S S, SHEVADE S K, BHATTACHARYYA C, *et al*.. Improvements to platt' s SMO algorithm for SVM classifier design [J]. Neural Computation, 2001, 13(3): 637 - 649.

[3] VEROPOULOS K, CAMBELL C, CRISTIANINI N. Controlling the sensitivity of support vector machines[C]// Proceedings of the International Joint Conference on AI, 1999: 55 - 60.

[4] VAPNIK V N. Statistical learning theory[M]. 许建华, 张学工, 译. 北京: 电子工业出版社, 2004.

[5] LEE L, LIN Y, WAHBA G. Multicategory supportvector machines [R]. Wisconsin: Technical Report 1040, Department of Statistics, University of Madison, 2001.

[6] DAVID V, SANCHEZ A. Aadvanced support vector machines and kernel methods [J]. Neurocomputing, 2003(1/2)55: 5 - 20.