Efficient Parallel Implementation of Molecular Dynamics with Embedded Atom Method on Multi-core Platforms

Changjun Hu
School of Information Engineering
University of Science and Technology Beijing
Beijing, P.R.China
huchangjun@ies.ustb.edu.cn

Yali Liu
[1] School of Information Engineering
[1] University of Science and Technology Beijing
[2] School of Science and Technology
[2] Beijing City University
Beijing, P.R.China
liuyali@bcu.edu.cn

Jianjiang Li
School of Information Engineering
University of Science and Technology Beijing
Beijing, P.R.China
jianjiangli@gmail.com

Abstract*—We present a scalable spatial decomposition coloring approach to implement molecular dynamics simulations with embedded atom method (EAM) on multi-core architectures. It effectively solves parallelization of reduction operations on irregular arrays in molecular dynamics simulations. In OpenMP program model, our methodology avoids that the same memory location is simultaneously modified by more than one thread when the short-range forces is calculated, meanwhile our method reduces memory requirements. The methodology comes from the idea of Red-Black Coloring, popular in linear algebra. We developed the spatial decomposition coloring algorithm, and our work applied this algorithm to implement the embedded atom method formalism for molecular dynamic. In this paper we also describe other optimizing methods applied in our serial and parallel implementations. Results show that our method is scalable and can achieve nearly linear speedup. Additionally we also compared it with other methods which can parallelize reduction operations on irregular array, and we discussed them in detail.

Keywords-Parallel Computing; Irregular Reductions; Spatial Decomposition; Molecular Dynamics; Embedded Atom Method

I. INTRODUCTION

In Molecular Dynamics (MD) simulations, the Embedded-Atom Method (EAM) [1] is usually used to calculate inter-atomic forces in particle systems for metals and alloys. However compare with pair-wise potential method which is commonly used for MD simulations, it has more intensive computation and more memory requirements.

We know that the intensive computation of MD appears in force calculations procedure, but the force calculations of EAM potential have more computations. It involves three computational phases, however pair-wise potential only involves one computational phase (compute forces directly). The three computational phases of EAM potential are evaluating electron densities, evaluating embedding energies and computing forces last.

The equation for the electron density ρ_i on embed atom i is calculated by

$$\rho_i = \sum_{j}^{N} \varphi(r_{ij}) \tag{1}$$

The equation for the force $\vec{F_i}$ on atom i is given by

$$\vec{F_i} = -\sum_{j \neq i}^{N} (V'(r_{ij}) + F'(\rho_i)\rho'_{ij} + F'(\rho_j)\rho'_{ji})\vec{r_{ij}} \tag{2}$$

From (1) and (2) we can see that the computation workload required by the embedded atom method is nearly more than twice the workload of the pair-wise potential for the same number of particles, and that EAM method requires extra memory space to store electron densities and its derivative of all atoms.

As we know that EAM is often used on metals or alloys simulation, and that majority metals have very high density, therefore metal atoms usually have more neighboring atoms than other type atoms. In most cases the neighboring atoms that lie within the cut-off range r_c from each atom are stored in neighbor list [2]. So MD simulations with EAM need more memory space to store the neighboring atoms. With the increasing of number of atoms, not only does computation increase, but memory requirement also becomes an increasingly heavy burden even to modern computers.

1530-2016/09 $26.00 © 2009 IEEE
DOI 10.1109/ICPPW.2009.24

The computing-intensive characteristic and high memory requirements together present a greater challenge to deal with parallelization of forces calculations of MD simulations on shared memory multi-core machines.

We extracted sketch code from sequential program of electron density and force calculations (shown in Fig. 1 and Fig. 2 respectively). Both piece of code have two nested loops, in which the outer loop deals with all atoms that constitute the system, and the inner loop deals with neighbors of atom i. In both of inner loops there are reduction operations on irregular arrays rho and force. How to efficiently parallelize reduction operation on irregular array has important impact on the overall parallel performance of MD simulations.

There are several popular molecular simulations frameworks in use today, including NAMD, GROMACS and LAMMPS etc. NAMD [3] is a parallel molecular dynamics code based on the Charm++ runtime system. Charm++ supports MPI and uses it as communication library. GROMACS [4] is another versatile package to perform molecular dynamics, using standard MPI communication. LAMMPS [5] is a classical molecular dynamics code, which can runs in parallel using message-passing techniques.

```
for ( i = 0;  i < N;  i++)
{
        neighstart = neighindex[i];
        neighend = neighstart + neighlen[i];
        for ( k = neighstart ;  k < neighend;  k++)
        {
                j = neighlist[k];
                …
                rho[i] += dfn_t1 ;
                rho[j] += dfn_t2 ;
        }
}
```

Figure 1. A piece of code of electron densities.

```
for ( i = 0;  i < N;  i++)
{
        neighstart = neighindex[i];
        neighend = neighstart + neighlen[i];
        for ( k = neighstart ;  k < neighend;  k++)
        {
                j = neighlist[k];
                …
                forc = …
                force[i][X]  += forc*xd ;
                force[i][Y]  += forc*yd ;
                force[i][Z]  += forc*zd ;
                force[j][X]  -= forc*xd ;
                force[j][Y]  -= forc*yd ;
                force[j][Z]  -= forc*zd ;
        }
}
```

Figure 2. A piece of code of forces calculations.

Most frameworks are based on distributed memory models, and not designed for multi-core platforms. So a lot of research work [6-8] has been done on whether these frameworks using MPI can get utilize multi-core resources effectively. Their experiments show there are some factors limit these frameworks efficiency on multi-core platforms: memory resource contention, and MPI intra-node communication.

And some work has been done to implement MD simulations using shared memory model on multi-core platform [9-13]. To the best of my knowledge, little work had been done to design efficient parallel implementation of short-range force calculations based on shared memory model in general multi-core processors. There are a number of parallel programming models based on shared memory models which have been proposed in the last years, OpenMP [14] standard usually appears to be more appropriate to parallelize computationally intensive applications on shared memory architectures. It gives programmers a convenient way to obtain parallelism. However OpenMP doesn't support the array reduction in C/C++.

Based on shared memory model, some types of solutions have been proposed in the literature to solve parallelization of reduction operations on irregular arrays. Most of these solutions can be classified into five categories. Different from the categories proposed by E. Gutiérrez etc. [15], we think transactional memory and redundant computations as solutions to deal with reduction irregular arrays.

The first one includes the simplest solution that enclosed the reference to the reduction array in a critical section, which ensures that multiple threads do not attempt to update the same reduction array simultaneously. Its disadvantage is the high synchronization cost when using critical region, atomic or lock in loop. Even in the worst case, multiple threads must work in serial execution [16].

The second class privates the reduction array to minimizing synchronization [17], but with the disadvantages of high memory overhead. Not only does it limit the number of particles allowed in simulations, but it also competes for cache space and can reduce the program speed. And it is not scalable, as memory overhead grows linearly with the number of threads [18].

The third class partitions computations and distributes it among threads in order to avoid write conflicts. In the literature different methods like LOCALWRITE [19, 20] and SYNCHWRITE [21] have been proposed. It has low synchronization cost and low memory requirements, but it needs an inspector at runtime, therefore it will bring the cost of reorder reduction array and computations.

The fourth class uses transactional memory [22] method which has been proposed to avoid the need for blocking, but one question that affects portability is the number of operations that can be contained in a single transaction.

If the background of application is observed, there is the last class which uses redundant computations strategy and can avoid reduction operations on irregular array appeared in MD simulations. Its advantage is the high parallelizability since data dependence has been removed between the loop iterations, but its disadvantage is there are double computations and that neighbor list requires more memory space.

In this paper, we proposed a scalable Spatial Decomposition Coloring (SDC) approach for solve the parallelization for irregular array reductions in MD simulations. On the base of knowledge about MD simulations, the loop iterations can be partitioned and be distributed among executing threads, and then multiple threads writing to the same address simultaneously can be avoided. Our method uses the implicit barrier at the completion point of the loop parallelized, and it has low synchronization cost.

The rest of the paper is organized as follows. In Section II, we described our spatial decomposition coloring approach, its OpenMP implementations and other optimizing methods applied in our MD simulations with EAM. The results from our experiments were presented and analyzed in Sections III and IV. And we concluded our work in Section V.

II. METHOD

A. Spatial Decomposition Method

For the MD simulations with short-range interactions, Spatial Decomposition (SD) [23] method is commonly used on distributed memory multi-processors involving several hundreds of processors. Spatial decomposition is a much more complex approach to implement [24] on distributed memory, since the programmer must change all array declarations and all loop bounds, and explicitly codes the periodic transfer of the boundary data between processors. This is a large and difficult step.

However it is simple to carry out parallel domain decomposition in OpenMP programming model rather than in MPI. Message passing is replaced by shared data that can be read and written by any thread. Programmer divides the spatial domain of simulation system into several subdomains which will be assigned among executing threads to executing its computations. In other words, the iterations of outer loop in Fig. 1 (and in Fig. 2) are partitioned among threads according to the coordinates of atom i.

But from Figs. 1 and 2 we can see that when the thread works on its subdomains to compute electron densities and forces of its atoms, it will also update electron densities and forces of its neighboring atoms which are located in neighbor regions (See Fig. 3, both neighbor atoms and neighbor regions of subdomain 4 were marked with strip), however the neighbor regions belong to other subdomains. Therefore when computations on subdomains are running in parallel, synchronization will be required to ensure that multiple threads do not attempt to update the same atom simultaneously. In this paper we proposed Spatial Decomposition Coloring (SDC) method to deal with this problem. We described SDC method in detail below.

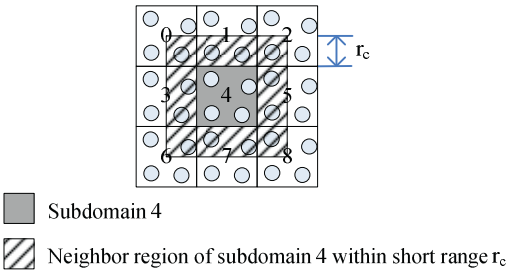

Figure 3. Neighbor regions of the atoms in the subdomain 4.

B. Spatial Decomposition Coloring (SDC) Approach

Our SDC method consists of the following steps:

1) SDC method firstly split the spatial domain of simulations into several subdomains. But in order to make computations as supposed, we require that the length of subdomains in each of the spatial decomposed dimensions should be longer than $2r_c$, and we require that the number of subdomains in each of the spatial decomposed dimensions should be even.

2) Then subdomains are colored with a set of different colors in such a way that each subdomain is surrounded only by those subdomains with different colors. And the number of subdomains with each color is equal.

3) After steps 1 and 2 have been done, computations can be done in the following way. For a given color, each OpenMP executing thread is assigned some subdomains with this color, and it can run parallelization calculations of forces on the subdomains with this color. But calculations on subdomains with different colors must run in a serial fashion. If the number of subdomains with one color is adequate for threads provided by multi-core platforms, then our method can not only effectively exploit the inherent parallelism of MD simulations, but also effectively exploit multi-core architectures. This requirement is easily achieved, for example in our experiments, there are 340 subdomains with each color in medium test case, and there are nearly 5000 subdomains with each color in large test case.

The steps 1 and 2 will be done when the neighbor list is created or updated. Since the neighbor list usually doesn't be updated in every time-step and the cost of spatial decomposition and coloring is very low, therefore the times of steps 1 and 2 can be omitted. SDC method has the same disadvantage of Spatial Decomposition method, which is overload imbalance. However, under condition of simulation system has uniformity of density, the overload balance can be achieved by the subdomains with same color have roughly equal volume. We do not require that subdomains with different colors should have same volume, which can have different volume from subdomains with other colors.

Based on the number of dimensions divided, SDC method can be classified into three categories: one-dimensional spatial decomposition coloring method, two-dimensional spatial decomposition coloring method and three-dimensional spatial decomposition coloring method. We described them in detail below.

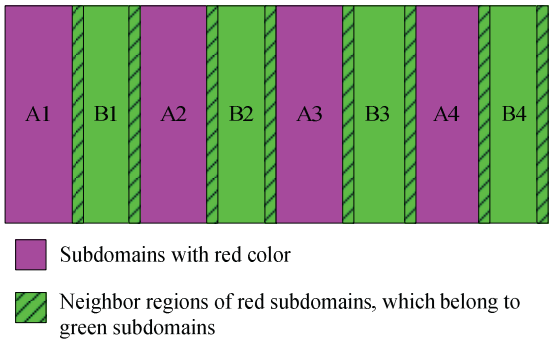

Figure 4. The one-dimensional SDC method.

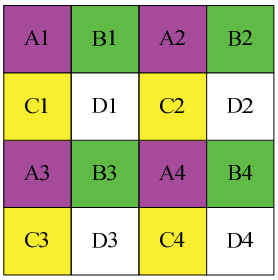

Figure 5. The two-dimensional SDC method.

In the one-dimensional spatial decomposition, algorithm decomposes the spatial horizontal-level into a set of subdomains, and colors subdomains red or green. The length of horizontal-level edges of each subdomain is longer than $2r_c$. Each subdomain has two neighbor regions on both the north and south which belongs to subdomains with another color. Fig. 4 is a chart of the one-dimensional spatial decomposition coloring method. In this chart spatial domains are divided into eight subdomains which were coloring with red or green alternately, and the neighbor regions of red subdomains were marked with strip.

From Fig. 4 we can see that if only computations are run for those subdomains with red color in parallel, values of atoms in their neighboring regions then will be updated simultaneously. Because the data spaces updated by threads do not overlap, we don't need synchronization for updating the shared array. However before green subdomains are computed, a barrier should be needed for waiting all threads to complete computation on red subdomains. Compare to using critical region, atomic or lock, barriers at the end of parallel loop has low synchronization cost.

Two-dimensional spatial decomposition coloring with four colors is shown in Fig. 5. Spatial domain is decomposed along both horizontal and vertical dimensions. Both the length of horizontal-level edges and vertical-level edges must larger than $2r_c$. As same as one-dimensional decomposition, computations of the electron densities and forces on subdomains with a given color can be executed in parallel, and a barrier will be needed for waiting those threads to complete computation.

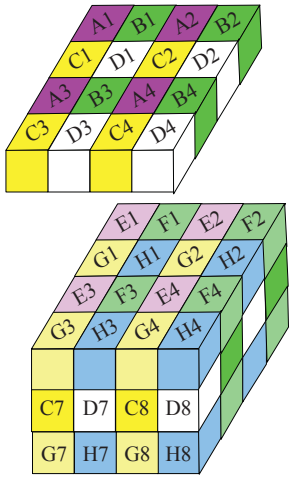

Figure 6. The three-dimensional SDC method.

Three-dimensional spatial decomposition coloring with eight colors is shown in Fig. 6. It is very similar to one-dimensional and two-dimensional spatial decomposition. It is obvious that the higher-dimensional decomposition method creates more subdomains. So it is impossible that some cores are idle while others are busy. SDC method is scalable and suitable on multi-core and many-core architectures.

Although the implementation of spatial decomposition coloring approach is a bit more complex than the implementation of share array privation approach, it has higher speedup than the latter, and has low memory requirements. We gave their speedup and analysis of efficiency in Section III and IV.

In Section II.C, we discussed the parallel implementation of forces calculations of EAM using SDC method

C. Parallel Implementation of Forces Calculations of EAM using SDC method

Force calculations procedure is the most time consuming part in MD simulations, especially in EAM potential. We know that forces calculations of EAM involve three computational phases: the evaluating electron densities (according (1)), the evaluating embedding energies and the computing forces (according (2)) last. The most time consuming parts are the calculations of the electron densities and forces. Their main code has been list in Fig. 1 and Fig. 2. Both of the codes (in Fig. 1 and Fig. 2) show the loops have cross-iteration dependences. If we use a simple directive (the #pragma omp parallel for directive) to specify that the iterations of the outer loops could be distributed among the executing threads, the programs can't get results as supposed.

Here we used spatial decomposition coloring approach to parallel those two parts. Then the parallel procedure of forces calculations of EAM involve the following phases:

1) For a given color, run electron density computations on subdomains with the color in parallel. But the computations on subdomains with different color must proceed serially. The mainly parallel code of electron densities calculations procedure is indicated in Fig. 7.

2) For all atoms in simulation system, calculate their embedding function value and their derivative, and accumulate embedding energies in parallel. Since this loop does not contain data dependences, so we can use a single directive (the #pragma omp parallel for directive) to parallelize this part. The parallel code is not shown in this paper.

3) For a given color, run force calculations on subdomains with the color in parallel. Like the electron density computations, the computations on subdomains with different color must proceed serially. Fig. 8 shows the mainly parallel code of forces calculations procedure.

```
#pragma omp parallel private(cpart)
for (cpart = 0; cpart < colors; cpart++)
{
...
#pragma omp for private(spart,i,j,k,…)
   for (spart = cpart; spart < subdomains; spart += colors)
   for ( ipart = pstart[spart];  ipart < pstart[spart+1];  ipart++)
   {
            i = partindex[ipart];
            neighstart = neighindex[i];
            neighend = neighstart + neighlen[i];
            for ( k = neighstart ;  k < neighend;  k++)
            {
                     j = neighlist[k];
                     ...
                     rho[i] += dfn_t1 ;
                     rho[j] += dfn_t2 ;
            }
   }
}
```

Figure 7. A piece of parallel code of electron density calculations.

```
#pragma omp parallel private(cpart)
for (cpart = 0; cpart < colors; cpart++)
{
 ...
#pragma omp for private(spart,i,j,k,…)
   for (spart = cpart; spart < subdomains; spart += colors)
   for ( ipart = pstart[spart];  ipart < pstart[spart+1];  ipart++)
   {
            i = partindex[ipart];
            neighstart = neighindex[i];
            neighend = neighstart + neighlen[i];
            for ( k = neighstart ;  k < neighend;  k++)
            {
                     j = neighlist[k];
                     …
                     forc = …
                     force[i][X]  += forc*xd ;
                     force[i][Y]  += forc*yd ;
                     force[i][Z]  += forc*zd ;
                     force[j][X]  -= forc*xd ;
                     force[j][Y]  -= forc*yd ;
                     force[j][Z]  -= forc*zd ;
            }
   }
}
```

Figure 8. A piece of parallel code of force calculations.

The mainly parallel code of electron densities calculations procedure is indicated in Fig. 7. And Fig. 8 shows the mainly parallel code of forces calculations procedure.

Both of them use the #pragma omp parallel directive to specify that the outer loops should be executed in parallel on the executing threads, and both to use the #pragma omp for directive to specify that the iterations of the inner loops (its iteration variable is spart) should be distributed among the executing threads. We didn't use a single directive (the #pragma omp parallel for) to specify that the iterations of the inner loops should be distributed among the executing threads, since it will increase the overheads introduced by forking and joining threads.

D. Other optimizing methods

Based on knowledge of MD simulations with EAM, we used two methods to accelerate sequential and parallel simulations in this paper. Both of them can reduce redundant computations in MD simulations.

1) The first method is that when we calculate the electron density ρ_i of atom i, we not only add the density contribution ρ_{ji} from neighboring atom j to atom i to ρ_i, but also calculate and add the density contribution ρ_{ij} from atoms i to neighboring atom j to ρ_j.

2) The second method is that we take advantage of Newton's third law in the force routine. When we calculate the force $\vec{F}_i$ on atom i, we add the interactions $\overrightarrow{f_{ij}}$ to $\vec{F}_i$ and we also add $-\overrightarrow{f_{ij}}$ to $\overrightarrow{F_j}$.

In our experiments, we updated serial code and parallel code to improve their data locality in our sequential and parallel implementations. We can see that irregular computations frequently appear in the loop codes of Figs. 1, 2, 7 and 8. It has poor temporal and spatial locality because they do not repeatedly access data in memory with small constant strides, which will be more evident with increasing number of atoms [25]. In our experiments, data reordering technique is used to increase locality of reference.

1) Achieve sequence accessing on irregular array by the data reordering.

Both in serial code and parallel code, the irregular arrays neighlist[] and rho[] were accessed in each iteration of inner loops, and their cache hits have important impact upon performance of simulations. We took advantage of the information of spatial positions of atoms to create neighbor list of atoms, and reordered index of neighboring atoms in neighlist[]. This adjustment also has impact on the cache hits of irregular array rho[].

2) Use data reordering technique to transform irregular arrays into regular arrays.

From Figs. 7 and 8, we can see there are other two irregular arrays neighindex[] and neighlen[]. We use data reordering to transform both two irregular arrays (neighindex[] and neighlen[]) into regular arrays. Then the accesses to the two arrays have good data locality.

After using data reordering technique, the simulation efficiency increased was 12% in serial simulations and was 39% in parallel simulations in our experiments on our large test case. Efficiency increased is measured as:

$$(Time_{\text{unoptimized}} - Time_{optimzied}) * 100 / Time_{\text{unoptimized}} \quad (3)$$

Since Section II.D is not the subject of study in this paper, so we didn't discuss them in detail. But in all of our experiments listed in Section III, we used these optimizing techniques.

III. Experiments and Results

A. Experimental environment

We make experiments on a machine with four Intel Xeon(R) Quad-core E7320 (L2 Cache 4MB) processors, 16 GB memory. The Operating-System is Fedora release 9 with kernel 2.6.25. The compiler is gcc 4.3.0.

Our experimental programs of MD application with EAM are written in C. The serial programs came from XMD code (which is a free MD program written by Jon Rifkin at the University of Connecticut [26]). We modified the serial programs using the optimizing methods described in Section 2.4. Then we programmed its parallel programs with OpenMP programming model using our Spatial Decomposition Coloring (SDC) strategy. And we also programmed XMD parallel programs using some parallel strategy discussed in Section I. These methods are critical construct strategy (which enclosed the reference to the reduction array in critical region, and belongs to the first class strategy), shared array privatization strategy (the second class strategy), redundant computations strategy (the last class strategy) respectively. We use sched_setaffinity() to control binding of a thread to cores at startup, which is supported in OpenMP.

All of execution times of our experiments are the running times of the calculations of the electron densities and forces, since these two parts are the most time-consuming components. So we merely observed the running times of these two parts and gave its speedups. We use system call gettimeofday() for measuring execution time. All of our experiments are run for 1000 simulation time-steps.

Then we gave our test cases on serial program and parallel programs.

B. Experimental cases

Our four test cases were designed to observe micro-deformation behaviors of the pure Fe metals material. The system was simulated under periodic boundary conditions and the time-step is 10^{-17} seconds. The initial state uses the same body-centered cubic (bcc) lattice arrangement. We considered the following test cases of our simulation systems for experiments.

Small -scale case (1) : 54,000 atoms
Medium-scale case (2): 265,302 atoms
Large-scale case (3) : 1,062,882 atoms
Large-scale case (4) : 3,456,000 atoms

Those test cases for our experiments in initial state are very similar. The differences are the number of atoms and initial energy of the particular atoms, because the number of atoms should increase with the increase of initial energy of the particular atoms.

C. Experimental results

We first gave speedup results of four test cases with one-dimensional, two-dimensional and three-dimensional spatial decomposition coloring method in Table 1. The speedup equals runtimes of serial programs on one core divided by runtimes of parallel programs on multiple cores.

In Table 1, there are blanks for the speedup of one-dimensional SDC method on small test case and medium test case with 12 threads and 16 threads. Since one-dimensional SDC algorithm decomposes the spatial horizontal-level into a set of subdomains, and the length of horizontal-level edges of each subdomain must be longer than $2r_c$. Therefore the number of subdomains split by one-dimensional SDC method is less than 24 in our small test case. So we didn't use one-dimensional SDC method on our small test case with 12 threads and more threads and on medium test case with 16 threads.

TABLE I. The Speedups of Spatial Decomposition Coloring (SDC) Methods (one-dimensional SDC, two-dimensional SDC and three-dimensional SDC).

Speedup	Small case (1) on 2~16 cores						Medium case (2) on 2~16 cores					
	2	*3*	*4*	*8*	*12*	*16*	*2*	*3*	*4*	*8*	*12*	*16*
SDC (one-dimensional)	1.71	2.46	3.07	4.17			1.84	2.64	3.37	6.24	6.33	
SDC (two- dimensional)	1.70	2.46	3.07	4.74	5.90	6.43	1.84	2.65	3.39	6.20	8.89	10.90
SDC (three- dimensional)	1.66	2.40	2.99	4.61	5.74	6.30	1.82	2.65	3.36	6.16	8.76	10.78
	Large case (3) on 2~16 cores						**Large case (4) on 2~16 cores**					
	2	*3*	*4*	*8*	*12*	*16*	*2*	*3*	*4*	*8*	*12*	*16*
SDC (one- dimensional)	1.86	2.76	3.67	6.82	9.76	9.59	1.88	2.79	3.66	6.30	9.97	9.82
SDC (two- dimensional)	1.87	2.78	3.64	6.74	9.73	12.31	1.87	2.80	3.65	6.77	9.84	12.42
SDC (three- dimensional)	1.86	2.75	3.64	6.64	9.65	12.29	1.87	2.80	3.67	6.74	9.82	12.34

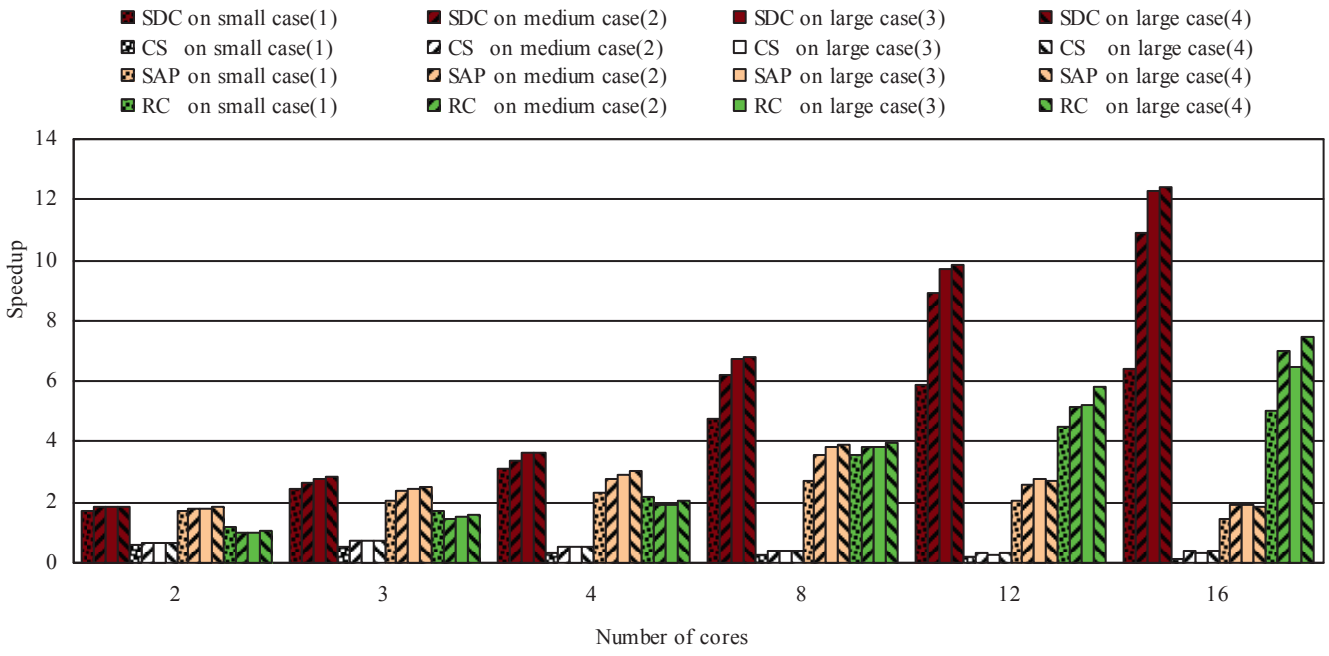

Figure 9. The speedup curves of two-dimensional Spatial Decomposition Coloring (SDC) method, Critical Section (CS) method, Share Array Privatization (SAP) method and Redundant Computations (RC) method on test case.

In order to compare the performance of SDC method with other methods, we had used these methods on all test cases. We gave the speedups of SDC method and other methods in Fig. 9. Here we selected two-dimensional decomposition SDC method to represent SDC method.

IV. DISCUSSION

Firstly we observe the scalability of our SDC method. Table 1 show that the performance of multi-dimensional (two-dimensional and three-dimensional) SDC method has been improved with the increase in the number of cores and the increase in the number of atoms. Therefore our multi-dimensional SDC method is a scalable method for parallel MD simulations (especially with EAM) on multi-core machines. And it is expected that our method will continue to be scalable on future many-core architectures.

One-dimensional SDC method has one restriction which limits the scalability of one-dimensional SDC method. As we know, parallelism is exploited by decomposing the spatial domain into subdomains in SDC method. Therefore when small-scale case is simulated using one-dimensional SDC method, maybe the degree of parallelism is less than the number of cores of machine. It's the disadvantage of one-dimensional SDC method. But it is not the restriction of multi-dimensional SDC method. Since multi-dimensional SDC method can divide spatial domain into more subdomains with same color, in other words, multi-dimensional SDC method can fully exploit the processing capability of the multi-core machine.

Secondly we compare the performance of one-dimensional, two-dimensional and three-dimensional SDC methods on test cases. Table 1 lists speedup results of SDC methods on our four test cases. We can see that two-dimensional SDC method achieves highest efficiency. Compare with one-dimensional SDC method, two-dimensional SDC method has more overhead of fork-join threads and more scheduling overheads. In detail, the one-dimensional SDC method only needs two different colors. It does fork-join threads two times to calculate of electron densities and forces in one time-step. And the two-dimensional decomposition does fork-join threads four times and three-dimensional decomposition does fork-join threads eight times. However two-dimensional SDC method does not degrade performance. Since our two-dimensional decomposition algorithm strives to make subdomains with small surface area and large volume, which results in better cache locality compared to the one-dimensional decomposition strategy. From Table 1, we also can see that three-dimensional SDC method degrades the performance but only slightly due to the more overhead of fork-join threads and scheduling.

Lastly we compared the parallel efficiency of our strategy with that of other parallel method on our test cases in Fig. 9. It is obvious that our two-dimensional SDC method not only achieves a linear speedup, but also has highest speedup than other methods on all of test cases. The reason of linear speedup is that the low synchronization cost of implicit barriers in our method can be amortized over a large amount of computation.

Critical Section (CS) method achieves lowest efficiency. CS method encloses reduction operations on irregular array

in critical section. Although it is simplest method, it is not feasible on multi-core architectures.

When the number of executing cores is less than 8, Share Array Privatization (SAP) method achieves better performance than CS method and Redundant Computations (RC) method. However its performance will degrade with the increase of the number of executing cores. It is partly because memory overhead grows linearly with the number of threads [7], which competes with cache space during the computations. But it is mainly due to synchronization overhead, since SAP method also needs synchronization to update share array according the value of private share array. In detail, after the thread-private copies had been updated, the shared array should be updated using the thread-private copies. Updating shared array must be done in a critical section, and which brings on high overhead when the number of executing cores is more than 8. So it is not a scalable method.

We also show speedup result of Redundant Computations (RC) in Fig. 9. RC method achieves a nearly linear speedup, because its double computation cost can be amortized over many cores. And it gets better performance when the number of executing cores is more than 8. But Because there is two-fold computation work for the force calculations in RC method than in SDC method, the efficiency of RC method is low than that of SDC method. Our experimental results demonstrated that SDC method can gain about 1.7-fold increase in performance as compared to RC method on medium and large test cases.

V. Conclusions and Future Work

In this paper, we presented a scalable spatial decomposition coloring (SDC) method to solve a class of short-range force calculations problems on shared memory multi-core platforms. Although in this paper SDC method was applied in the MD implementation with EAM potential, but it is obvious that our method can be applied in MD simulations with other potentials. Our experiments show that SDC method can effectively solve the parallelization of force calculations which involve the reduction operations on irregular array. And our experiments show that SDC method is scalable not only to large simulation system but also to many-core architectures.

There are two directions for future research. Firstly, a detailed study of SDC method on NUMA memory architecture is needed. How to achieve better performance under multi-core and multi-socket shared memory system is of particular interest. Lastly, it will be promising to implement SDC method using mixed programming models such as MPI+OpenMP in multi-core cluster.

Acknowledgment

This work reported in this paper is supported by National High-tech R&D Program of China under Grant No. 2006AA1Z105, and by the Key Project of Chinese Ministry of Education under Grant No.106019 and No.108008.

References

[1] M.S. Daw and M.I. Baskes, "Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals," J. Phys. Review B, vol. 29, 1984, pp. 6443–6453.

[2] L. Verlet, "Computer experiments on classical fluids i. thermodynamical properties of Lennard-Jones molecular", Phys. Review, vo.159, 1967, pp. 98-103.

[3] NAMD Scalable Molecular Dynamics, http://www.ks.uiuc.edu/Research/namd/.

[4] GROMACS: Fast, Free and Flexible MD, http://www.gromacs.org/.

[5] LAMMPS Molecular Dynamics Simulator, http://lammps.sandia.gov/.

[6] S.R. Alam, and P.K. Agarwal, "On the path to enable multi-scale biomolecular simulations on PetaFLOPS supercomputer with multi-core processors", IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007) , IEEE Press, Mar. 2007, doi:10.1109/IPDPS.2007.370443.

[7] S.R. Alam, P.K. Agarwal, S.S. Hampton, Hong Ong, and J.S. Vetter, "Impact of multicores on large-scale molecular dynamics simulations," IEEE International Symposium on Parallel and Distributed Processing Symposium(IPDPS 2008), IEEE Press, Apr. 2008, doi:10.1109/IPDPS.2008.4536181.

[8] S.R. Alam, P.K. Agarwal, S.S. Hampton, and Hong Ong, "Experimental evaluation of molecular dynamics simulations on multi-core systems," High Performance Computing(HiPC 2008), Springer Verlag Press, Dec. 2008, pp. 131-141, doi:10.1007/978-3-540-89894-8_15.

[9] C.I. Rodrigues, D.J. Hardy, J.E. Stone, K. Schulten, and W.W. Hwu, "GPU acceleration of cutoff pair potentials for molecular modeling applications," 2008 Conference on Computing frontiers(CF'08), ACM Press, May 2008, pp. 273-282, doi:10.1145/1366230.1366277.

[10] D.J. Hardy, J.E. Stone, K.Schulten, "Multilevel summation of electrostatic potentials using graphics processing units," Parallel Computing, vol. 35, Mar. 2009, pp.164-177, doi: 10.1016/j.parco.2008.12.005.

[11] G. Shi, and V. Kindratenko, "Implementation of NAMD molecular dynamics non-bonded force-field on the Cell Broadband Engine processor," IEEE International Parallel and Distributed Processing Symposium(IPDPS 2008), IEEE Press, Apr. 2008, doi: 10.1109/IPDPS.2008.4536470.

[12] S. Goedecker, "Optimization and parallelization of a force field for silicon using OpenMP," Computer Physics Communications, vol. 148, Oct. 2002, pp. 124-135, doi: 10.1016/S0010-4655(02)00466-6.

[13] R. Couturier, and C. Chipot, "Parallel molecular dynamics using OpenMP on a shared memory machine," Computer Physics Communications, vol. 124, Jan. 2000, pp. 49-59, doi: 10.1016/S0010-4655(99)00432-4.

[14] OpenMP website. http://www.openmp.org

[15] E. Gutiérrez , O. Plata, and E.L. Zapata, "An analytical model of locality-based parallel irregular reductions," Parallel Computing, vol. 34, Mar. 2008, pp. 133–157, doi: 10.1016/j.parco.2008.01.003.

[16] B. Chapman, G. Jost, R. Van Der Pas, and D.J. Kuck, Using OpenMP: portable shared memory parallel programming, The MIT Press, 2007.

[17] M.W. Hall, J.M. Anderson, S.P. Amarasinghe, B.R. Murphy, and S.W. Liao, et al. , "Maximizing multiprocessor performance with the SUIF compiler", Computer, vol. 29, Dec. 1996, pp. 84–89, doi: 10.1109/2.546613.

[18] E. Dedu, S. Vialle, C. Timsit, "Comparison of OpenMP and classical multi-threading parallelization for regular and irregular algorithms," International Conference on Software Engineering Applied to Networking and Parallel/Distributed Computing(SNPD'00), May. 2000, pp. 53-60.

[19] H. Han, C.W. Tseng, "Efficient compiler and run-time support for parallel irregular reductions", Parallel Computing, vol. 26, Dec. 2000, pp. 1861–1887, doi: 10.1016/S0167-8191(00)00062-4.

[20] H. Han, C.W. Tseng, "A comparison of parallelization techniques for irregular reductions," IEEE International Parallel and Distributed Processing Symposium(IPDPS 2001), IEEE C.S. Press, Apr. 2001,doi: 10.1109/IPDPS.2001.924963.

[21] E. Gutiérrez, O. Plata, and E.L. Zapata, "A compiler method for the parallel execution of irregular reductions in scalable shared memory multiprocessors," Conference Proceedings of the 2000 International Conference on Supercomputing, ACM Press, May 2000, pp. 78-87, doi: 10.1145/335231.335239.

[22] M. Herlihy, and J.E.B. Moss, "Transactional Memory: architectural support for lock-free data structures," Proceedings of the 20th Annual International Symposium on Computer Architecture (ISCA'93), IEEE Press, May 1993, pp. 289–300.

[23] A. Nakano, R. K. Kalia, and P. Vashishta, "Multiresolution molecular dynamics algorithm for realistic materials modeling on parallel computers," Computer Physics Communications, vol. 83, Dec. 1994, pp.197-214, doi: 10.1016/0010-4655(94)90048-5.

[24] R. A. Kendall, E. Aprã, D.E. Bernholdt, E.J. Bylaska, M. Dupuis, et al. "High performance computational chemistry: an overview of NWChem a distributed parallel application," Computer Physics Communications, vol. 128, Jun. 2000, pp. 260-283, doi: 10.1016/S0010-4655(00)00065-5.

[25] H. Han, C.W. Tseng, "A comparison of locality transformations for irregular codes," Proc. Fifth Workshop on Languages, Compilers and Run-Time Systems for Scalable Computers, Springer-Verlag Press, May 2000, pp. 70-84, doi: 10.1007/3-540-40889-4_6.

[26] J. Rifkin, XMD-molecular dymamics for metals and ceramics, http://xmd. sourceforge.Net.