Parallel Programming Library for Molecular Dynamics Simulations

R. TROBEC,[1] M. ŠTERK,[1] M. PRAPROTNIK,[2] D. JANEŽIČ[2]

[1]*Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia*

[2]*National Institute of Chemistry, Hajdrihova 19, 1000 Ljubljana, Slovenia*

Received 16 September 2002; accepted 28 May 2003

DOI 10.1002/qua.10742

ABSTRACT: A parallel programming library for molecular dynamics (MD) simulations is described and applied to the recently proposed split integration symplectic method (SISM) for MD simulation. The results show that for a system of 1024 linear chain molecules with an integration step of 4.5 fs parallel execution of SISM with the particle–particle interactions (PPIs) library on 32 computers gives efficiency of 95.6%. The results also show the parallel simulation of n particles is scalable with the number of processors p and the time requirement is proportional to n^2/p for n/p large enough, which guarantees optimal speed-up. © 2003 Wiley Periodicals, Inc. Int J Quantum Chem 96: 530–536, 2004

Key words: parallel simulation; molecular dynamics; object-oriented library

1. Introduction

The advent of computer technology initiated new methods for the study of molecular systems. They provide us with data that are of intermediate character between theory and experiment; therefore, they are known as "computer experiments." From the theoretician's point of view computer experiments are important because essentially they provide exact data on well-defined models. The molecular dynamics (MD) method has been long used with considerable success. In MD the computer is used to solve numerically the classic equations of motion for an assembly of interacting particles. The equations to be solved are coupled second-order differential equations whose number is equal to the number of degrees of freedom in the system.

The interparticle interactions, which determine the microscopic behavior of the system, are determined by the corresponding force field. There are several methods for calculating these interactions. The simplest, particle–particle methods, iterate through all pairs and directly compute the interactions, leading to the time complexity of $O(n^2)$ for an n-particle system. There are several simple and efficient parallel implementations of particle–particle methods [1–3]. Indirect methods, such as particle–mesh or the fast multipole methods, have asymptotically lower complexities but are harder to parallelize. For small to midsized systems (up to tens

Correspondence to: D. Janežic; e-mail: dusa@cmm.ki.si

of thousands of atoms) the particle–particle methods are faster or at least competitive with indirect methods [4, 5].

After computing all interparticle interactions, the simulation evolves the system with the computed interactions. The evolution of the system with time complexity of $O(n)$ is performed independently for each particle, which makes it trivial to parallelize.

Several MD algorithms for solving Hamiltonian systems have been proposed. The simplest and most commonly used is the leapfrog Verlet (LFV) algorithm [6]. Although simple to use it tends to quickly become unstable as the integration time-step is increased [7]. However, for studies of dynamics of large molecular systems larger time-steps in the MD integration are needed [8]. The problem of how to increase the time-step in the MD integration procedure can be overcome by use of symplectic integration methods for Hamiltonian systems [9].

We recently introduced an efficient split integration symplectic method (SISM) [10–12] for MD integration. The method allows the use of a time-step up to an order of magnitude larger than the commonly used LFV algorithm, which is of the same order and computational complexity as SISM.

The structure of this article is as follows: In Section 2 the basic concepts of the programming library for particle–particle interactions (PPIs) are presented. The SISM algorithm is described in Section 3 and programming SISM with the PPI library is given in Section 4. Next, the measured parallel performance of the SISM is given on clusters with different numbers of computing nodes. The work concludes with some comments on results and directions for future work.

2. PPI Library

Particle simulations are often programmed either for many similar systems or even for the same system, but with different numerical methods. Typically, a new program is obtained by changing a similar sequential program. Parallelizing a sequential program and debugging the parallel program is not always an easy task and the effort is multiplied in the case of several different programs.

When a parallel algorithm is selected, the principle of parallelization is the same for all applications. To avoid repeating the same steps to parallelize every program, we developed the universal programming library PPI [13]. The library is object

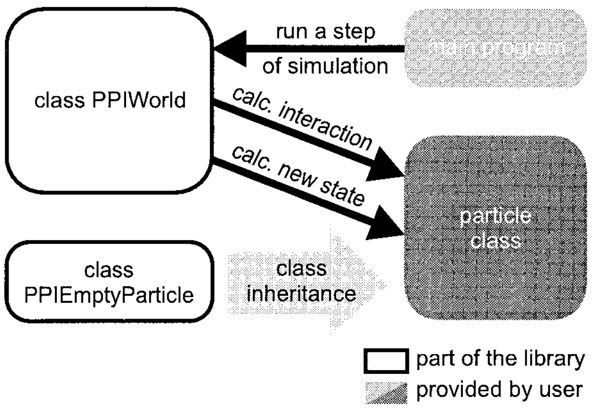

FIGURE 1. Structure of a parallel simulation program using the `PPI` library.

oriented, written in C++, and uses the message passing interface (MPI) [14] for communication. To use the PPI library the user has to describe the problem through writing a class representing a particle. The particle class must contain all attributes of a particle and knowhow for the calculation of interactions and new particle states and for reading or writing particle attributes from/to a file. Such a particle class together with the library class template `PPIWorld` forms the required new simulation class `PPIWorld`⟨particle class⟩. Figure 1 depicts the schematic representation of a user-defined simulation by PPI library.

Parallel algorithms for direct calculation of the interactions can distribute either particles among processors (particle decomposition), parts of the $N \times N$ force matrix (force decomposition), or geometric subdomains of the simulated system (domain decomposition) [15, 16]. The particle and force decomposition methods always divide computations optimally, reducing the time complexity to $O(n^2/p)$ on a p-processor system. Both can be implemented on different network topologies. The chosen method as well as topology affects the parallel performance.

The proposed PPI library implements the systolic loop particle decomposition [17]. In most applications the interaction between two particles is symmetrical and no particle can interact with itself. The sequential algorithm for n particles for each time step is thus:

```
for (i = 0; i < n; i++)
    for (j = i+1; j < n; j++)
      calculateInteraction(i, j);
```

There are $n(n-1)/2$ calculations of particle interactions.

On a parallel system with p processors numbered as $0, 1, \ldots, p-1$ and connected in a ring, the particles are divided uniformly among processors. All local particles on a processor are called self-particles. The copies of self-particle attributes, being transmitted to the neighboring ring processors, are called guest-particles. Several variations of the algorithm were tested. We describe here the fastest variation only.

Each processor performs, within a particular time-step, $p/2$ calculation passes using the following rules:

1. Calculate the interactions between self-particles.
2. Send copies of self-particle attributes to the right neighbor and receive guest-particle (copies of self-particle) attributes from the left neighbor. The interpretation of left and right directions is arbitrary.
3. In the next pass calculate the interactions between all pairs: self-particle–guest-particle, and accumulate the calculated interactions with the self-particle and guest-particle attributes.
4. Send guest-particles to the right neighbor and receive new guest-particles from the left.
5. Repeat steps 3 and 4 until the particles are moved halfway around the ring.
6. If the number of processors is odd, the same pairs of particles are now on processors i and $i + p/2$. Each processor therefore calculates only half of the interactions in the last pass.
7. After all the interactions are calculated, but not yet added to all particles, it is necessary to return guest-particles halfway around the ring to their original processors, which add all the remaining interactions. On topologies with more than two connections per processor, this step can be implemented directly with a single point-to-point message for each processor.

The systolic loop particle decomposition algorithm requires more communication than some versions of the force decomposition algorithms. This is a potential disadvantage on massively parallel computers, but not as much on smaller computer systems. However, the force decomposition algorithm must store all n particles on each processor, running out of cache memory sooner than particle decomposition algorithms, which only store $2n/p$ particles on each processor [15].

3. SISM for MD Integration

To perform the MD simulation of a system with a finite number of degrees of freedom the Hamilton equations of motion

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i}, \frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}, \quad i = 1 \ldots d \tag{1}$$

are to be solved, where H is the Hamiltonian, q_i and p_i are the coordinate and momentum, respectively, and d is the number of degrees of freedom.

In terms of free Lie algebra, the Hamiltonian Eq. (1) can be written in the form

$$\frac{d\mathbf{x}}{dt} = \{\mathbf{x}, H\} = \hat{L}_H \mathbf{x}, \tag{2}$$

where $\{\mathbf{x}, H\}$ denotes the Poisson bracket, $\hat{L}_H$ is the Poisson bracket operator, and $\mathbf{x} = (\mathbf{q}, \mathbf{p})$ is a vector of coordinates and momenta of all particles.

The formula

$$\mathbf{x}|_{t_i+\Delta t} = \exp(\Delta t \hat{L}_H)\mathbf{x}|_{t_i} \tag{3}$$

is the formal solution of the Hamilton equations and represents the exact time–evolution of trajectory in the phase space from t_i to $t_i + \Delta t$, where Δt is the integration step [9].

The construction of an efficient algorithm rests on the ability to separate the Hamiltonian into parts that are themselves integrable and also efficiently computable [18]. To derive the SISM [11] we have split the MD Hamiltonian into

$$H = H_0 + H_r, \tag{4}$$

where H_0 is the pure harmonic part and H_r is the remaining part of the potential.

The following approximation for $\mathbf{x}|_{t_i+\Delta t}$ was then used:

$$\mathbf{x}|_{t_i+\Delta t} \approx \exp\left(\frac{\Delta t}{2}\hat{L}_{H_0}\right)\exp(\Delta t \hat{L}_{H_r})\exp\left(\frac{\Delta t}{2}\hat{L}_{H_0}\right)\mathbf{x}|_{t_i}, \tag{5}$$

which prescribes how to propagate from one point in phase space to another. First, the system is propagated for a half integration step by H_0, then for a whole step by H_r, and finally for another half step by H_0. This integration scheme defines the SISM, a second-order symplectic integration algorithm for MD integration. Knowing H_0, the high-frequency terms are treated analytically, i.e., independently of the size of the integration step. This permits the SISM to employ up to an order of magnitude larger integration step size than can be used by other methods of the same order and complexity. The whole integration step thus combines the analytic evolution of H_0 with a correction arising from the H_r performed by numerical integration. The motion governed by H_0 is resolved by diagonalizing the Hessian, which is the root-mass-weighted second derivative matrix of the bond stretching and angle bending part of the potential function, to obtain the vibrational frequencies and normal mode vectors of H_0 [19]. The Hessian depends only on constant parameters of the simulation. Therefore, normal modes are calculated only once, at the outset of the calculation.

Schematically the SISM reads as follows [10]:

1. At the outset of calculation vibrational frequencies and normal modes, represented by normal coordinates P, Q, of H_0 are determined.
2. Rotate the normal coordinates, P_j^0, Q_j^0, in the phase space by the corresponding vibrational frequency ω_j for $\Delta t/2$:

$$\begin{bmatrix} P_j' \\ Q_j' \end{bmatrix} = \mathbf{R} \begin{bmatrix} P_j^0 \\ Q_j^0 \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} \cos\left(\omega_j \dfrac{\Delta t}{2}\right) & -\omega_j \sin\left(\omega_j \dfrac{\Delta t}{2}\right) \\ (1/\omega_j)\sin\left(\omega_j \dfrac{\Delta t}{2}\right) & \cos\left(\omega_j \dfrac{\Delta t}{2}\right) \end{bmatrix}.$$

   For $\omega_j \neq 0$ vibrations are taken into account and for $\omega_j = 0$ translations and rotations. Equations

$$P_j' = P_j^0$$
$$Q_j' = P_j^0 \frac{\Delta t}{2} + Q_j^0$$

   describe translations and rotations in normal coordinates.
3. Coordinate transformation from normal coordinates P_j', Q_j' to Cartesian displacement coordinates p_i', q_i'.
4. Numerical integration of momenta (one force calculation per integration step)

$$p_i'' = p_i' - \Delta t \left(\frac{\partial H_r}{\partial q_i} \right)_{q_i = q_i'}.$$

5. Back transformation from Cartesian displacement coordinates p_i'', q_i'' to normal coordinates P_j'', Q_j''.
6. Again, the rotation of normal coordinates in the phase space by the corresponding vibrational frequency ω_j for $\Delta t/2$:

$$\begin{bmatrix} P_j \\ Q_j \end{bmatrix} = \mathbf{R} \begin{bmatrix} P_j'' \\ Q_j'' \end{bmatrix},$$

   which concludes one SISM integration step.
7. Go to 2 until the desired number of integration steps is reached.

In the scheme the high-frequency terms are treated analytically. This enables the SISM to use much larger integration time-step Δt than the standard methods of the same order and complexity. Hence, the SISM performs superiorly for systems in which high-frequency motions can be treated analytically, e.g., systems with stiff internal degrees of freedom.

To determine the parallel efficiency of the SISM the system of linear molecules was used as a model system. For this model system the MD Hamiltonian [10] is

$$H = \sum_i \frac{\mathbf{p}_i^2}{2m_i} + \sum_{\text{bonds}} k_b(b - b_0)^2 + \sum_{\text{angles}} k_\vartheta(\vartheta - \vartheta_0)^2 + \sum_{i>j} \frac{e_i e_j}{r_{ij}} + \sum_{i>j} 4\varepsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^6\right], \quad (6)$$

where i and j run over all atoms, m_i is the mass of the i-atom, b_0 and ϑ_0 are reference values for bond lengths and angles, respectively, k_b and k_ϑ are corresponding force constants, e_i denotes the charge on the i-atom, r_{ij} is the distance between atoms i and j, and ε_{ij} and σ_{ij} are the corresponding constants of the Lennard–Jones potential.

Into the split part H_0, which describes the vibrational motion of the system as well as translation and rotation of molecules, kinetic energy, and harmonic part of the bond stretching and angle bending potential energy of the system are included, while H_r is the remaining part.

4. Programming the SISM with PPI Library

The PPI library requires one to provide a particle class following some guidelines [13], which includes methods that are called from `PPIWorld`:

`clearInteraction()`—resets the attributes that represent interactions, e.g., force vectors.

`interact()`—calculates the interaction between two particles and writes the result into both particles' attributes.

`preIteration()/postIteration()`—do whatever is necessary before and after calculating the interactions, e.g., propagate harmonic part and convert to Cartesian coordinates.

`createMPIType();`—creates a description of an MPI type that is used in communication. The user would have to study the MPI documents [14] at this step only. This method is necessary provided that a heterogeneous parallel system composed of different computer types is used. However, on homogeneous computer clusters leaving out this method will only result in minimal degradation of the communication performance.

Once the particle class is written, the parallelization is done automatically. In our case, we already had the `ButadiyneLFV` class that implements LFV simulation of a butadiyne system. The object-oriented design of PPI library enabled us to derive the `ButadiyneSISM` class from `ButadiyneLFV`, overriding only the calculation of new atom coordinates and momenta, which now includes the harmonic part of the Hamiltonian. A pseudocode example for `ButadiyneLFV`, `ButadiyneSISM`, and the main program is given in Figure 2.

5. Parallel Performance of the SISM

The SISM was evaluated on the model system composed of a different number of linear butadiyne molecules, a six-atom molecule of the form H—(—C≡C—)$_2$—H, at T = 300 K and ρ = 0.1 g/cm^3. Potential parameters were the same as in Ref. [12], and the periodic boundary conditions were imposed to overcome the problem of surface effects.

```
class ButadiyneLFV: public PPIEmptyParticle {
public:
    Vector coordinate[6], momentum[6], force[6];
    static double mass[6];

    void preIteration() {
        for (int i = 0; i < 6; i++)
            coordinate[i].add(momentum[i]/mass[i]*timeStep/2);
    }
    void clearInteraction() {
        for (int i=0; i<6; i++)
            force[i] = intramol. forces on atom i
    }
    void addPartialForce(ButadiyneLFV &other) {
        calculate Lennard-Jones force between
            atoms in *this and in other
        add it to *this and subtract from other
    }
    void postIteration() {
        for (int i = 0; i < 6; i++) {
            coordinate[i].add(momentum[i]/mass[i]*timeStep/2);
            momentum[i].add(force[i]*timeStep);
        }
    }
};

class ButadiyneSISM: public ButadiyneLFV {
public:
    Vector coordSys[3], normCoord[6], normMomentum[6];
        //vectors defining normal coord. system
        //and normal coordinates

    void preIteration() {
        harmonical part of dynamics
            (vibration, rotation, translation)
        transform normal coordinates to Cartesian
    }
    //clearInteraction inherited from ButadiyneLFV
    //addPartialForce inherited from ButadiyneLFV
    void postIteration() {
        transform Cartesian coordinates to normal
        harmonical part of dynamics
            (vibration, rotation, translation)
    }
};

int main(int argc, char **argv) {
    PPIWorld<ButadiyneSISM> world(argc, argv, &cout);

    analyse harmonical part
    ifstream inFile(argv[1], "r");
    world.readParticles(inFile);
        //read initial states from file
    world.mainLoop(atoi(argv[2]), settings);
        //run argv[2] steps
}
```

FIGURE 2. Pseudocodes of particle classes `ButadiyneLFV` and `ButadiyneSISM`, and program `main`.

The simulation programs were tested on a cluster of 16 dual-processor Athlon MP 1600+ workstations running Linux, connected by a fast Ethernet network through a switch. Network latencies were 38 μs between two processors in the same computer and 80 μs between different computers. When executing the systolic loop communication scheme, each of 32 processors achieved simultaneous inbound and outbound bandwidths of about 30 Mbps.

Figures 3 and 4 show the speed-ups of the SISM for a various number of processors and molecules as a function of p and n/p, respectively. The speed-up is substantial already at 8 molecules per processor (n/p = 8) and almost linear if $n/p \geq 32$, which is reflected by almost horizontal lines in

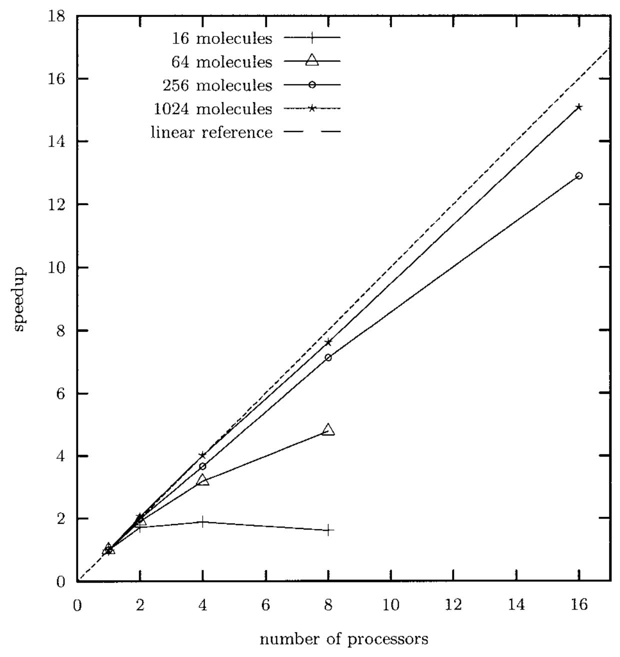

FIGURE 3. Speed-up of the parallel PPI-based SISM as a function of *p* for various system sizes.

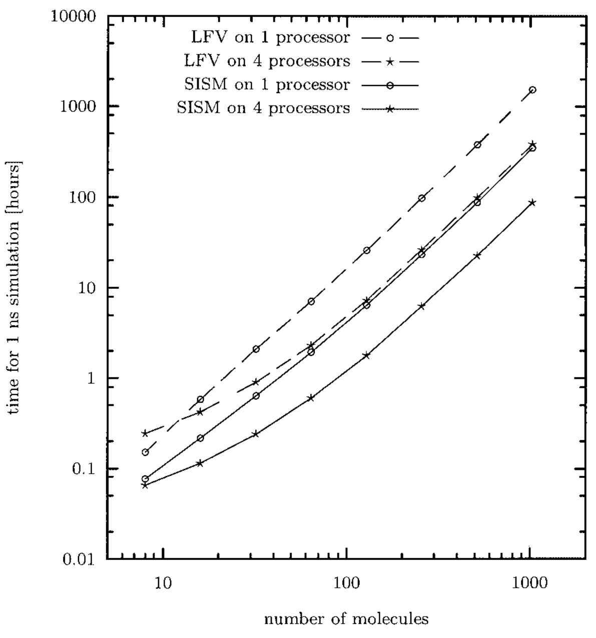

FIGURE 5. Comparison of the sequential and parallel LFV and SISM for different system sizes on four processors.

Figure 4. In particular examples, it is even slightly superlinear, which can be explained by cache effects, e.g., 1024 molecules do not fit into the cache on a single processor but increasing the number of processors increases the total amount of cache and therefore the hit rate. The amount of total communication is on the order of $O(n)$, independent of p. The computation time is on the order of $O(n^2/p)$. With larger p, the computation time decreases but the communication time does not. Further, more short messages increase the communication time. These explain the poor speed-up for $n/p \leq 4$.

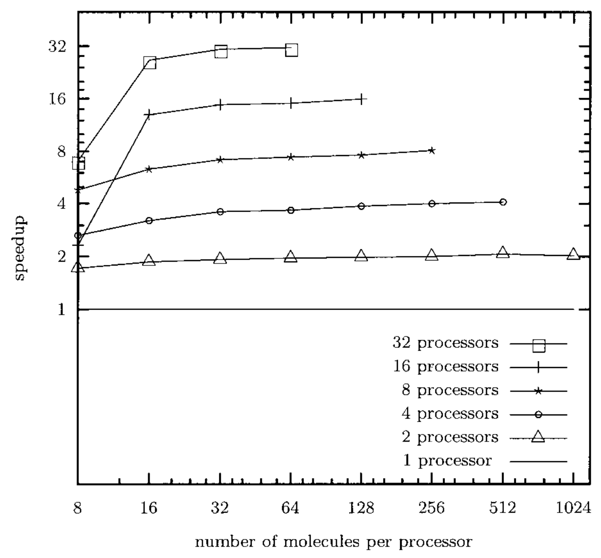

FIGURE 4. Speed-up of the parallel PPI-based SISM as a function of *n/p* for various number of processors.

The results of comparison of the sequential and parallel LFV and SISM are presented in Figure 5. The integration steps of 1 fs for LFV and 4.5 fs for SISM were used to compare computational performances for the same level of accuracy for both methods [10]. Total execution times are shown for a simulation length of 1 ns. The sequential LFV scales as $O(n^2)$, which is revealed by a straight line on the logarithmic plot. The parallel LFV is 4 times faster than the sequential LFV for values of $n > 100$ on 4 processors. For smaller systems, the parallelization is less efficient. The calculation of the intermolecular interactions, the most time-consuming part of the simulation, is the same for both methods [20]. The calculation complexity of transformations between different types of coordinates in the case of SISM is linear, as opposed to quadratic complexity of the intermolecular interactions calculation. For smaller systems ($n \leq 10$), the sequential SISM is only twice as fast as the sequential LFV. For larger systems ($n \geq 100$), the quadratic complexity of the intermolecular interactions calculation prevails.

Therefore, the SISM is 4.5 times faster than the LFV due to the larger integration step that can be used by SISM. In the parallel example, the SISM achieves better speed-ups for $n/p < 32$ because the additional work is optimally distributed among processors and requires no extra communication.

6. Conclusion

The present work examined the parallel library `PPI` for particle interactions for developing various MD simulation programs. The PPI library was applied to the recently proposed SISM for MD integration. For comparison, the PPI library was also applied to the standard LFV method.

The performances of parallel and sequential LFV and SISM programs are given and analyzed in detail. It is shown that the SISM achieves approximately linear speed-ups for systems larger than 32 molecules per processor. For smaller systems, the additional work required by the SISM contributes significantly to the total execution times. However, efficient parallelization compensates for this.

Further work lies primarily in the testing of the PPI library on different parallel computer systems [21] and also for more complex and realistic particle systems [22, 23].

ACKNOWLEDGMENT

This research was funded under Grants P1-513-106 and P1-503-104 by the Ministry of Education, Science, and Sports of Slovenia.

References

1. Hockney, R. W.; Eastwood, J. W. Computer Simulation Using Particles; Institute of Physics Publishing: Bristol, PA, 1988.
2. Murty, R.; Okunbor, D. Parallel Comput 1999, 25, 217–230.
3. Gibbon, P.; Sutmann, G. In: Grotendorst, J.; Marx, D.; Muramatsu, A., eds. Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms; John von Neumann Institute for Computing: Jülich, Germany, 2002; p. 467–506.
4. Spurzem, R. J Comp Appl Math 1999, 109, 407–432.
5. Heffelfinger, G. S. Comput Phys Commun 2000, 84, 219–237.
6. Allen, M. P.; Tildesley, D. J. Computer Simulation of Liquids; Clarendon Press: Oxford, UK, 1987.
7. Pastor, R. W.; Brooks, B. R.; Szabo, A. Mol Phys 1988, 65, 1409–1419.
8. Schlick, T.; Barth, E.; Mandziuk, M. Annu Rev Biophy Biomol Struct 1997, 26, 181–222.
9. Sanz-Serna, J. M.; Calvo, M. P. Numerical Hamiltonian Problems; Chapman & Hall: London, 1994.
10. Janežič, D.; Praprotnik, M. Int J Quantum Chem 2001, 84, 2–12.
11. Janežič, D.; Merzel, F. J Chem Info Comput Sci 1995, 35, 321–326.
12. Janežič, D.; Merzel, F. J Chem Info Comput Sci 1997, 37, 1048–1054.
13. Trobec, R.; Šterk, M.; Praprotnik, M.; Janežič, D. Int J Quantum Chem 2001, 84, 23–31.
14. Snir, M.; Otto, S.; Huss-Lederman, S.; Walker, D.; Dongarra, J. MPI: The Complete Reference; MIT Press: Cambridge, MA, 1996.
15. Sutmann, G. In: Grotendorst, J.; Marx, D.; Muramatsu, A., eds. Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms; John von Neumann Institute for Computing: Jülich, Germany, 2002; p. 211–254.
16. Plimpton, S. J Comput Phys 1995, 117, 1–19.
17. Trobec, R.; Jerebic, I.; Janežič, D. Parallel Comput 1993, 19, 1029–1039.
18. Wisdom, J.; Holman, M. Astron J 1992, 104, 2022–2029.
19. Brooks, B. R.; Janežič, D.; Karplus, M. J Comput Chem 1995, 16, 1522–1542.
20. Trobec, R.; Merzel, F.; Janežič, D. J Chem Info Comput Sci 1997, 37, 1055–1062.
21. Trobec, R. Parallel Comput 2000, 26, 1945–1953.
22. Janežič, D.; Brooks, B. R. J Comput Chem 1995, 16, 1543–1553.
23. Janežič, D.; Venable, R. M.; Brooks, B. R. J Comput Chem 1995, 16, 1554–1566.