MATHEMATICA MONTISNIGRI
Vol XXXIX (2017)

ANALYSIS OF PARALLEL MOLECULAR DYNAMICS FOR MPI, CUDA AND CUDA-MPI IMPLEMENTATION

A. V. UTKIN

Khristianovich Institute of Theoretical and Applied Mechanics of the Siberian Branch of the Russian Academy of Sciences
Institutskaya 4/1, 630090 Novosibirsk, Russia
e-mail: utkin@itam.nsc.ru

Summary. In the framework of current study, three implementations of parallel MD algorithms were compared. The first approach was based on design of parallel program for a computer cluster with distributed memory using Message Passing Interface (MPI). The second type of parallel algorithms was implemented on CUDA based General Purpose GPUs by NVIDIA. It should be noted, that modern high performance computing systems are a combination of MPI clusters equipped with GPGPUs, what turns them into so-called heterogeneous computing clusters. In this case MPI technology is used for internode communications, while all computations are carried out by GPUs. Thus, the third approach to the parallelization discussed in this study was based on design of a CUDA-MPI algorithm. The detailed studies and comparison of all three approaches (MPI, CUDA and CUDA-MPI) were performed in order to define optimal parameters and conditions of applicability of each algorithm.

1 INTRODUCTION

The method of molecular dynamics (MD) offers a good possibility of detailed investigations of specific features of the nanostructure formation mechanism at the submolecular level and provides thermal and mechanical characteristics of nanostructures [1, 2].

One of the most complicated aspects in MD modeling is a long time required for computation even for comparatively small systems of atoms. The main method of solving this problem is implementation of highly efficient parallel programs [3]. Using high-performance parallel clusters and stations, one can study systems consisting of tens of millions of atoms, i.e., having a qualitatively different scale as compared to systems whose dynamics can be calculated on a personal computer.

Traditional and well-developed area is the use of parallel codes based on MPI (Message Passing Interface) technology for parallel computing. Another promising and fairly efficient way is to create algorithms and codes that will perform calculations on CUDA capable GPUs [4–10]. Design of hybrid computational clusters gave rise to the development of new algorithms, which use the CUDA technology for resource-consuming computations, whereas the MPI technology ensures communications between different GPUs that physically belong to different nodes of the cluster.

2010 Mathematics Subject Classification: 65Y05, 65Y10, 65Y20, 65Z05.
Key words and phrases: molecular dynamics, CUDA, MPI, parallel computations.

At the present moment LAMMPS package is the most widely used software for molecular dynamics simulation and it also can exploit GPU. The main idea of GPU utilization approach there is dynamic load balancing of force calculation between CPU and GPU. Fraction of particles assigned for GPU calculation could be fixed or could be based on CPU and GPU timings [11–13].

But despite the GPUs are efficient computing units and heterogeneous supercomputers are presented among TOP50 supercomputers, the number of GPU-enabled applications remains relatively low (not only MD-related applications). It should be noted that majority of existing GPU-enbaled applications have poor cluster-wide scalability; either single GPU per node is exploited or the OpenMP technology is used to establish inter-GPUs communications and hence scalability is limited to the size of a node. We believe that the main reason is induced by complexity of transformation of existing and well-tested MPI applications to CUDA technology. The aim of this work was to show that existing baseline MPI code can be successfully translated into the CUDA-enabled one.

The present paper describes such a hybrid algorithm, which allows the CUDA and MPI technologies to be combined in one code written for microscopic-level simulations of various phenomena within the framework of the MD method. The efficiency of this code is compared both with a code using only the MPI and with a CUDA code designed for operation with only one GPU.

All CUDA versions of Molecular Dynamics algorithms discussed below are implemented using PGI Fortran as the programming language [14].

2 PHYSICAL MODEL AND COMPUTATION ALGORITHM

The numerical algorithms and programs were first tested on a sufficiently simple physical model, which was a rectangular reservoir filled with a gas (argon). The reservoir size was $2500 \times 2500 \times 3000$ Å (503000 atoms) or $2500 \times 2500 \times 12550$ Å (2516100 atoms). Interaction of atoms with the reservoir wall was described by the reflection model, and interatomic interaction was simulated by the Lennard-Jones potential [1, 2].

After that more complicated physical systems were studied. They described an impact of two copper clusters (figure 1) and uniaxial tension of copper nanorods (figure 2). Interaction of copper atoms was described by the embedded atom model (EAM) [15].

The equations of motion were integrated with the use of the second-order Verlet scheme in terms of the time step [1, 2]. The time step in these numerical simulations was 5×10^{-16} s for the gas and 10^{-16} s for copper.

All functions from EAM potential (electron density, pair potential and embedding function) were tabulated and B-spline interpolation was used for force and energy calculation.

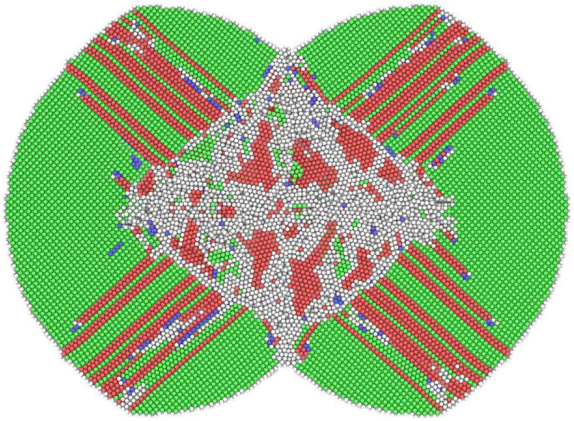

Figure 1: Longitudinal section of the collided clusters. Structure types: green—fcc, red—hcp, blue—bcc, grey—unknown coordination structure [16–18].

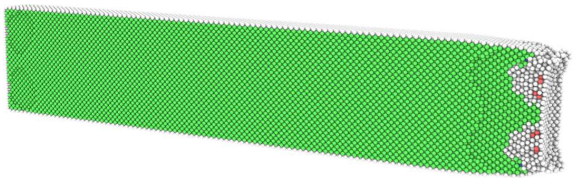

Figure 2: Uniaxial tension of copper nanostructure. Structure types: green—fcc, red—hcp, blue—bcc, grey—unknown coordination structure [16–18].

The calculation of interatomic interaction forces is the most computationally expensive task in MD simulations because the contributions of all neighboring atoms should be taken into account in calculating the forces acting on the i-th atom.

There are several commonly used methods for optimizing force computations, which ensure significant reduction of the computational time. As it will be demonstrated below that the choice of a particular technique can play a key role, it seems reasonable to consider the most popular methods of optimization [3].

2.1 Verlet list method

If a large system of atoms is studied and the cutoff radius r_c is smaller than the computational domain, some of the atoms do not contribute to the force acting on the i-th atom. In this method, an additional cutoff radius r_v (greater than r_c) is introduced. Before the computations, a list of neighboring atoms within a sphere with the radius r_v is formed for each i-th atom. When the forces acting on this atom are calculated, however, only the neighboring atoms from this sphere are taken into account. The interval between updating of the Verlet list of neighbors is usually 10–20 time steps or is determined automatically [19].

2.2 Cell-linked list method

An alternative method for effective determination of atom neighbors in sufficiently large systems is the cell-linked list method (CLLM). A three-dimensional computational domain is divided into rectangular cells (in the classical version) whose size is slightly greater than or equal to the cutoff radius r_c of the atomic interaction potential. An array containing the list of numbers of neighbors of each cell is generated at the beginning of computations. Each atom interacts only with atoms from its own cell or with atoms from the neighboring cells (26 cells).

One of the additional advantages provided by the cell-linked list method is the possibility of using the third Newton's law in force calculations: $F_{ij} = -F_{ji}$. As a result, it is possible to avoid double computations of the force in the pair of the i-th and j-th atoms and to reduce the number of the neighboring cells from 26 to 13, which substantially decreases the computational time [1].

2.3 Various hybrid combinations of the CLLM and Verlet list (hybrid Verlet list)

The most obvious hybrid combination is the method where the Verlet list for the i-th atom is generated on the basis of the analysis of the distances to the atoms in the cell to which the i-th atom belongs and the distances to the atoms in 26 neighboring cells rather than on the basis of considering all atoms of the system. Thus, one of the major problems in constructing the Verlet list is eliminated [1].

3 OPTIONS OF THE COMPUTATIONAL STRATEGY

3.1 Code implementation with the use of the MPI technology based on one-dimensional parallelization

An important aspect of programming for cluster systems with distributed memory is correct organization of data exchange between the nodes and its synchronization.

In the present study, we use the method of splitting of the computational domain into subdomains in one of the geometric directions (one-dimensional parallelization). This approach offers the following advantages: particles "assigned" to one CPU interact with each other, and data transfer between different CPUs is performed only because it is necessary to exchange information between geometrically neighboring subdomains. However, successful application of this algorithm is only possible in sufficiently homogeneous media, because uniform loading of CPUs should be provided [20]. In studying physical properties of spatially nonhomogeneous systems characterized by significant gradients of atomic concentration, it is necessary to use an additional algorithm of dynamic balancing. This algorithm corrects the size of the computational subdomains by means of shifting the boundaries and ensures more or less uniform loading of CPUs.

3.2 Code implementation of the parallel algorithm for graphics processing units (GPUs) based on the CUDA NVIDIA technology

A specific feature of program implementation on GPUs is performing an identical set of instructions by each computational processor (one atom–one thread or atom decomposition scheme) [9, 10]. It should be noted that the bottleneck of using GPUs is a low speed of data transfer from the main memory of the computer to the GPU memory. Thus, the main challenge is to minimize data exchange between the computer RAM and GPU memory. Two versions of the GPU code were developed. The first version is based on the CLLM, whereas the second

version of the computational code employs a hybrid combination of the CLLM and Verlet list.

3.3 Code implementation with the use of the MPI and CUDA NVIDIA technologies

The computational code was developed on the basis of a previously tested MPI-based program. This code is based on dividing the system of atoms in space (one-dimensional parallelization with an additional algorithm of dynamic load balancing). As the computation of force interactions is the most computationally expensive part of the process, it was this part that was transferred from the CPU to GPU. As the operation of data copying from the CPU to GPU takes a long time, it was decided to apply the CLLM directly on the GPU rather than to copy already available arrays from the computer memory. Correspondingly, the GPU computations were performed with the use of the CLLM.

An obvious drawback of this scheme is the necessity of copying a large amount of information from the CPU to GPU and vice versa at each step of computations in order to organize data exchange between GPUs concerning the motion of atoms inside the computational domain. As the number of atoms N_{CPU} on each CPU core is different at each computational step, it is not necessary for simple potentials to use the Verlet lists by virtue of their definition given above.

4 COMPARATIVE ANALYSIS OF ALGORITHMS AND CODES

The computations were performed on the following computational systems:

System No. 1 Hybrid parallel cluster NKS 30-T in Siberian Supercomputing Center SB RAS (40 nodes, each node has two sixcore Xeon X5670 (2.93 GHz) CPUs and 3 NVIDIA Tesla M 2090 GPUs.

System No. 2 PC with a Tesla C1060 GPU and i7-920 CPU.

4.1 Physical system based on the Lennard-Jones potential

Comparative testing of the above-mentioned programs was performed. The total computational time was 0.05 ns, i.e. 100000 steps of 5×10^{-16} s. It follows from the time dependences presented in table 1 that the program based only on the CUDA technology and hybrid Verlet lists employs the smallest amount of computational resources (the program was run on one GPU).

If only the CLLM is used, the computational time is significantly longer (almost by a factor of 5) because it is necessary to process additional information from the neighboring cells. The code based on the MPI+CUDA hybrid scheme provides a shorter computational time than the MPI code (the program was run on 9 CPUs and 9 GPUs). The force computation speedup was almost by a factor of 4, but computational time was supplemented with the time of data exchange between the CPU and GPU, which is almost equal to the time spent on the force computation on the GPU. The code based only on the CUDA technology and hybrid Verlet lists ensures faster computations than the MPI+CUDA code (almost by a factor of 8). It should be noted that the MPI+CUDA algorithm inherits all drawbacks inherent in the MPI algorithm,

	MPI (9 CPUs)[1]	MPI+CUDA (9 CPUs + 9 GPUs)[1]	CUDA (1 GPU)[1]	CUDA (1 GPU)[1]	CUDA (1 GPU)[2]
Total computational time	11600	6840	4379*/4853**	839*/5096**	1602*/11477**
Total time of force computation	2100	548	3996*/3996**	137*/138**	199*/202**
Data exchange between the CPU and GPU		507			
Sorting method	CLLM	CLLM (26 neighboring cells)	CLLM	CLLM + Verlet list	

[1] System No. 1.
[2] System No. 2.
* Updating of the list of neighbors every 20 steps.
** Updating of the list of neighbors at each time step.

Table 1: Task computation time (seconds).

Number of atoms	MPI (9 CPUs)[1]	MPI+CUDA (9 CPUs + 9GPUs)[1]	CUDA (1 GPU)[1]	CUDA (1 GPU)[2]
243644	14792	4234	5269*	10238*
709214	32081	8218	16970*	30429*
Sorting method	CLLM	CLLM (26 neighboring cells)	CLLM+Verlet list	

[1] System No. 1.
[2] System No. 2.
* Updating of the list of neighbors every 20 steps.

Table 2: Task computation time (seconds).

which are primarily associated with the necessity of data exchange between CPU nodes and with their nonuniform loading. As was mentioned above, the Verlet lists should not be used because of the permanently changing number of atoms in each node, which further aggravates the situation.

However, despite all these drawbacks, the resultant code will ensure effective modeling of physical systems consisting of a large number of atoms. The reason is that an increase in the number of atoms will inevitably lead to higher requirements to the memory for information storage.

A situation may arise where the code based only on the CUDA technology and hybrid Verlet lists will require more computational resources than the GPU can provide (such a situation occurred in computations with a large system consisting of 2516100 atoms). Fortunately, such a situation is little probable in the case of using the MPI+CUDA hybrid code because the amount

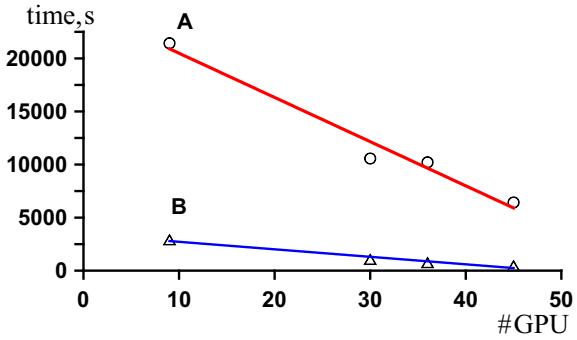

Figure 3: Total computational time (seconds) of the MPI+CUDA program (A) and total time of force computation versus the number of GPUs used (B).

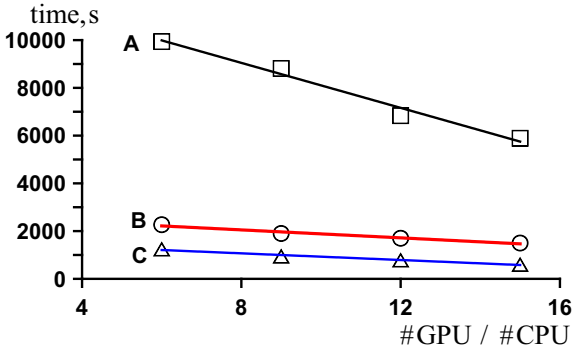

Figure 4: Total computational time (seconds) versus the number of GPU/CPU used. A—MPI-code. B—MPI+CUDA code, C—force computation in MPI+CUDA code. 168941 atoms.

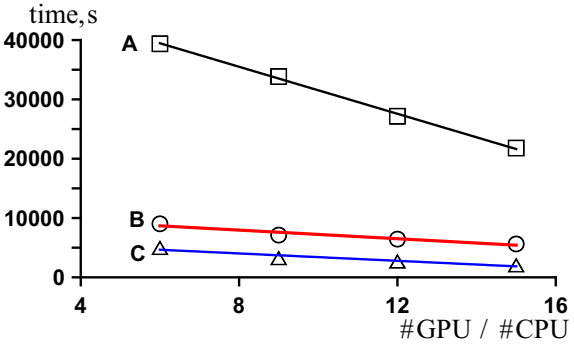

Figure 5: Total computational time (seconds) versus the number of GPU/CPU used. A—MPI-code. B—MPI+CUDA code, C—force computation in MPI+CUDA code. 659381 atoms.

of the required resources is inversely proportional to the number of GPUs and CPUs used. Figure 3 shows the time dependences for the system consisting of 2516100 atoms.

4.2 Physical system based on the embedded atoms model (EAM)

The next stage was testing of the code that ensures a microscopic-level description of the collision of copper clusters and uniaxial tension of copper nanostructure. The total computational time was 0.005 ns—50000 steps of 10^{-16} s. It follows from the results of table 2 for cluster collision that the MPI+CUDA hybrid code, similar to the code based on the Lennard-Jones potential, ensures faster computations than the MPI-based code (speedup almost by a factor of 2). The code based only on the CUDA technology and hybrid Verlet lists provides slightly slower computations than the MPI+CUDA code for the system consisting of 243644 atoms, but the difference in the computation speed becomes more pronounced for the system consisting of 709214 atoms. The same dependences were observed for copper nanorods uniaxial tension. Figures 4–7 show results for algorithm scalability. Strong scaling results are presented for sys-

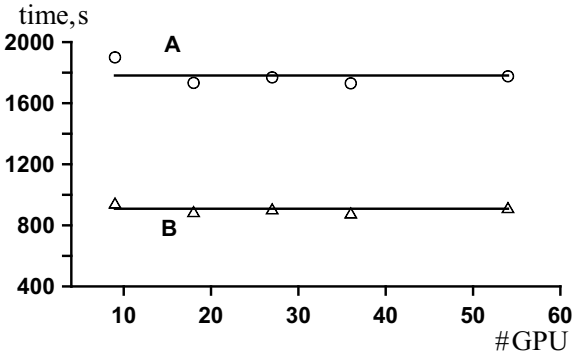

Figure 6: Weak scaling results with 18700–18750 atoms per node. A—MPI+CUDA code, B—force computation in MPI+CUDA code.

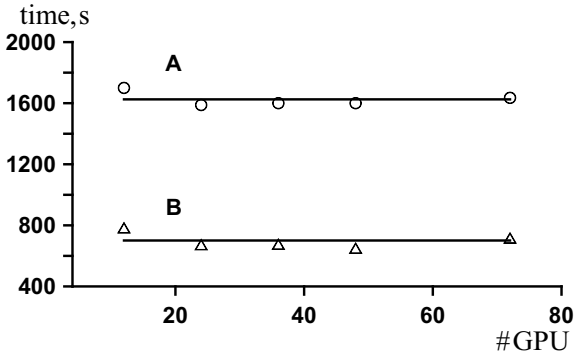

Figure 7: Weak scaling results with 14020–14080 atoms per node. A—MPI+CUDA code, B—force computation in MPI+CUDA code.

tem of 168941 atoms ($100 \times 20 \times 20$ lattice cells along X, Y and Z coordinate axes, respectively figure 4) and for system of 659381 atoms ($100 \times 40 \times 40$ lattice cells, figure 5). One can see that MPI+CUDA hybrid code is much faster that MPI-based code. Weak scaling results (figures 6 and 7) are obtained for the set of systems consist of 168941, 337041, 505141, 673241 and 1009441 atoms.

5 CONCLUSIONS

A parallel hybrid algorithm was developed and tested. This method ensures computations on the GPU and CPU (CUDA + MPI). Comparisons with similar algorithms implemented only with the use of the MPI or CUDA technology demonstrated fast performance of the hybrid code. The developed algorithms and codes were used for molecular dynamics modeling of microscopic-level phenomena both in solids subjected to intense external loading and in gas media.

Acknowledgments: The paper is based on the proceedings of the XXXII International Conference on Interaction of Intense Energy Fluxes with Matter, which was held in Elbrus settlement, in the Kabardino-Balkar Republic of the Russian Federation, during March 1–6, 2017.

REFERENCES

[1] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids* (Oxford Science Publications, New York, NY, USA, 2000).

[2] D. Frenkel and B. Smit, *Understanding Molecular Simulation, Second Edition: From Algorithms to Applications (Computational Science)* (Academic Press, 2001).

[3] A. Y. Kuksin, A. V. Lankin, I. V. Morozov, G. E. Norman, N. D. Orekhov, V. V. Pisarev, G. S. Smirnov, S. V. Starikov, V. V. Stegailov, and A. V. Timofeev, *Program Systems: Theory and Application* **5**, 191–244 (2014).

[4] D. J. Auerbach, A. F. Bakker, C. Lutz, W. J. Paul, W. E. Rudge, and F. Abraham, *J. Phys. Chem.* **91**, 4881 (1987).

[5] J. E. Stone, D. J. Hardy, I. S. Ufimtsev, and K. Schulten, *J. Mol. Graphics Modell.* **29**(2), 116–125 (2010).
[6] J. A. Anderson, C. D. Lorenz, and A. Travesset, *J. Comput. Phys.* **227**(10), 5342–5359 (2008).
[7] M. S. Friedrichs, P. Eastman, V. Vaidyanathan, M. Houston, S. Legrand, A. L. Beberg, D. L. Ensign, C. M. Bruns, and V. S. Pande, *J. Comput. Chem.* **30**(6), 864–872 (2009).
[8] I. V. Morozov, A. M. Kazennov, R. G. Bystryi, G. E. Norman, V. V. Pisarev, and V. V. Stegailov, *Comput. Phys. Commun.* **182**, 1974–1978 (2011).
[9] M. S. Ozhgibesov, A. V. Utkin, V. M. Fomin, T. S. Leu, and C. H. Cheng, *Int. J. Comput. Mater. Sci. Eng.* **1**(1), 1250007 (2012).
[10] M. S. Ozhgibesov, A. V. Utkin, V. M. Fomin, T. S. Leu, and C. H. Cheng, *Comput. Continuum Mech.* **5**(3), 265 (2012).
[11] W. M. Brown, P. Wang, S. J. Plimpton, and A. N. Tharrington, *Comput. Phys. Commun.* **182**, 898–911 (2011).
[12] W. M. Brown, A. Kohlmeyer, S. J. Plimpton, and A. N. Tharrington, *Comput. Phys. Commun.* **183**, 449–459 (2012).
[13] W. M. Brown and M. Yamada, *Comput. Phys. Commun.* **184**, 2785–2793 (2013).
[14] T. P. Group, PGI CUDA Fortran Compiler, URL: http://www.pgroup.com/resources/cudafortran.htm.
[15] A. F. Voter, The embedded atom method, in: *Intermetallic Compounds: Principles & Practice*edited by J. H. Westbrook and R. L. Fleischer, (John Wiley and Sons, London, 1994), 77.
[16] J. D. Honeycutt and H. C. Andersen, *J. Phys. Chem.* **91**(19), 4950–4963 (1987).
[17] A. Stukowski, *Modell. Simul. Mater. Sci. Eng.* **20**(4), 45021 (2012).
[18] A. Stukowski, *Modell. Simul. Mater. Sci. Eng.* **18**(1), 015012 (2009).
[19] L. Verlet, *Phys. Rev.* **159**(1), 98–103 (1967).
[20] Y. Deng, R. F. Peierls, and C. Rivera, *J. Comput. Phys.* **161**(1), 250–263 (2000).

Received June 13, 2017