COLLOQUIA: CSFI 2008

A simple asynchronous replica-exchange implementation

GIOVANNI BUSSI(*)

Dipartimento di Fisica, Università di Modena e Reggio Emilia
CNR-INFM Center on nanoStructures and bioSystems at Surfaces (S3)
via Campi 213, 41100 Modena, Italy

Summary. — We discuss the possibility of implementing asynchronous replica-exchange (or parallel tempering) molecular dynamics. In our scheme, the exchange attempts are driven by asynchronous messages sent by one of the computing nodes, so that different replicas are allowed to perform a different number of time steps between subsequent attempts. The implementation is simple and based on the message-passing interface (MPI). We illustrate the advantages of our scheme with respect to the standard synchronous algorithm and we benchmark it for a model Lennard-Jones liquid on an IBM-LS21 blade center cluster.

PACS 89.20.Ff – Computer science and technology.
PACS 05.10.-a – Computational methods in statistical physics and nonlinear dynamics.
PACS 02.70.-c – Computational techniques; simulations.

Parallel tempering is a popular method used to enhance sampling in Monte Carlo or molecular dynamics simulations [1-3], with applications ranging from molecular biology to statistical physics. In parallel tempering simulations, many replicas of the system are simulated at different temperatures and their temperatures are exchanged in a Monte Carlo fashion. The algorithm is very flexible, and can be extended to simulation parameters other than the temperature (see, *e.g.*, refs. [4-6]). In this generalized form the method is usually referred to as *replica exchange*. Furthermore, parallel tempering (or replica exchange) can be easily combined with other enhanced-sampling methods [7,8].

A very important feature of replica-exchange simulations is that they allow exploiting large parallel machines: the standard implementation is done assigning a different computing node to each replica, so that inter-process communications are only needed when attempting an exchange move, which is typically done with a prefixed stride. Moreover, parallel tempering can be coded in such a way that temperatures are exchanged instead of coordinates. In this case, the amount of data that are transmitted between computing nodes is negligible, resulting in a theoretically perfect scalability.

(*) E-mail: gbussi@unimore.it

However, practical problems arise when the number of replicas grows. In fact, for hundreds of computing nodes, even the overhead of synchronization can be large, especially when using short strides between subsequent attempts as suggested, *e.g.*, in ref. [9]. Furthermore, there is no warranty that the different nodes will take the same time to perform the same number of steps. As an example, the frequency of updates of the neighbor list depends on the diffusion coefficient [10], which in turns depends on the temperature, so that high-temperature simulations are expected to be slower. The situation is even worse when heterogeneous machines are used.

A possible solution for these problems is the adoption of serial methods [11,12]. In this paper we will discuss a different approach, namely the implementation of replica exchange in an asynchronous manner. The present implementation is simple and based on the MPI library [13]. It is designed as a master-slave scheme, where a node is asynchronously driving the exchanges for the other replicas. To the best of our knowledge, the only other attempt to implement asynchronous parallel tempering is the SALSA framework introduced in refs. [14,15].

1. – Implementation

In our implementation temperatures are exchanged rather than coordinates. This allows to minimize the amount of data that has to be communicated, at the price of a slightly more complex analysis of the resulting trajectories. Our asynchronous algorithm consists of the iteration of the following steps:

– Each node sends (asynchronously) its temperature to the master node, then starts to integrate molecular dynamics.

– When N_x steps are elapsed since the last exchange attempt, the master node collects the temperatures of all the other nodes, sorts them, and establishes the exchange pattern (*i.e.* which pairs of nodes should attempt the exchange). Then, it sends (asynchronously) a message to each node (including itself) containing the rank of the partner node.

– At every step, each node checks if a message arrived from the master. If the answer is negative, computation is continued. Otherwise, a message is sent (asynchronously) to the partner node, containing the actual temperature, potential energy and a uniformly distributed random number. Then, the node waits for the corresponding message from the parter node. Once both temperatures and energies are available, the random number generated by one of the two nodes (arbitrarily fixed to be that with the lowest rank) is used to establish if the exchange is successful using the standard Metropolis rule [3]. In case of positive answer, the temperature and velocities are properly scaled.

As a further improvement, the rank of the master node is changed at each exchange, so that the master overhead (sorting replicas and sending/receiving many messages) is distributed over the full set of nodes.

Only two MPI calls need to be done in a blocking way: a) the collection of temperatures done by the master node, since all the temperatures should be known to prepare the exchange pattern; b) the reception of the triplet of temperature, energy and random number, which is needed to determine whether the temperature has to be changed or not. All the other calls to the MPI library are asynchronous and allows for overlap of

communication and computation. The scheme can be trivially extended to support non-nearest neighbor exchanges. When compared with the SALSA framework [14, 15], our scheme is simpler and has the advantage that it is based on a standard MPI library and do not require multi-thread capabilities.

As a reference, we also implement a synchronous version, where the collection of the temperature is done at every node with the prefixed stride using a MPI_ALLGATHER call. Here all the nodes are able to decide the exchange pattern and perform the exchange after the same prefixed number of steps.

2. – Example

We test our implementation using a Lennard-Jones liquid. This system does not exhibit significant barriers, and parallel tempering is in principles not needed to properly sample the phase space. However, the diffusion of the replicas in temperature space is similar to what is usually observed for simulation of solvated molecules, so that it can be used as a meaningful test case. We simulate 4000 particles at density $\rho = 0.844$, with cutoff in the interparticle interaction at 2.5. Throughout this section we use Lennard-Jones units for distances, energies and time. A neighbor-list containing pairs up to a distance 3.0 is used, and it is updated when the displacement of at least one particle is larger than half the skin width [10]. The integration time step is 0.005, and temperature is controlled using stochastic velocity rescaling [16]. N_r replicas are simulated at exponentially distributed temperatures in the range from $T_{min} = 0.7$ to $T_{\max} = T_{\min} \times 1.0275^{N_r-1}$, which gives an exchange acceptance of approximately 30%, irrespectively of N_r. In the asynchronous implementation, exchanges between replicas are attempted when the master node has done N_x steps since the last attempt. Since the number of steps performed by the other nodes is not prefixed, the actual average number of steps between attempts $\langle N_x \rangle$ is not equal to N_x. On the other hand, when the synchronous implementation is used the same number of steps is performed by all the nodes between subsequent attempts, so that $\langle N_x \rangle = N_x$. All the calculations are performed on the BCX cluster at CINECA.

2˙1. *Diffusion in temperature space*. – A very important parameter in replica-exchange simulations is the speed at which each replica diffuses in the temperature space. This speed is inversely related to the average time required for a replica to cross the full temperature range from the lowest to the highest temperature. We here calculate the diffusion coefficient of the logarithm of the temperature in the case $N_r = 48$ for different choices of N_x in both the synchronous and asynchronous implementation. The results are plotted in fig. 1(a) as a function of the observed exchange stride $\langle N_x \rangle$. As is seen, diffusion in both cases is equivalent and only depends on the effective exchange stride. Moreover, as has been observed in ref. [9], the optimal performance is obtained by using an exchange stride which is as small as possible.

2˙2. *Scalability*. – We then consider the overhead due to the exchanges as the number of replicas N_r grows. To this aim we fix $N_x = 5$ for the synchronous implementation and $N_x = 1$ for the asynchronous implementation. For the latter, this choice gives a practical waiting time of $\langle N_x \rangle \approx 5$, so that the two algorithms are comparable. We then define the parallel efficiency λ as the ratio between the total number of steps performed by all the replicas in a prefixed wall-clock time and the same number as obtained during a simulation without exchanges. In practice, this number should be equal to 1 if there
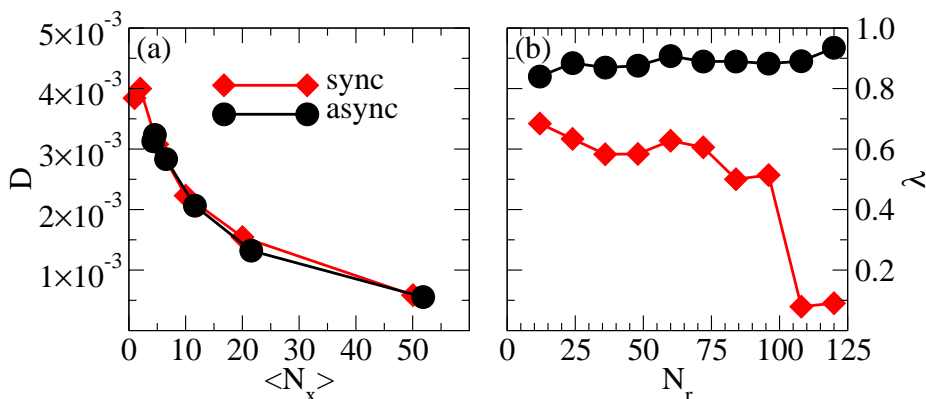
Fig. 1. – (a) Diffusion coefficient D in the log-temperature space, as a function of the average number of waiting steps $\langle N_x \rangle$. (b) Parallel efficiency λ, as a function of the number of replicas N_r. Both quantities are plotted for the synchronous (diamonds) and asynchronous (circles) implementations. See text for details.

was no computational overhead, and decreases towards zero as the overhead increases. In fig. 1(b) we plot the efficiency λ as a function of N_r. It is clearly seen that the synchronous implementation is much less efficient than the asynchronous one, and that it gets slower and slower as N_r increases. On the other hand, the efficiency of the asynchronous implementation is very close to 1 even when more than a hundred replicas are used.

3. – Conclusions

We have shown an asynchronous implementation of replica-exchange molecular dynamics, where the exchange attempts are driven by external commands sent by one of the replicas. This implementation is based on the MPI library. Its performance is excellent and allows for a significant saving of computational effort in simulations with hundreds of replicas. The code is available on request.

<p align="center">* * *</p>

REFERENCES

[1] Swendsen R. and Wang J., *Phys. Rev. Lett.*, **57** (1986) 2607.
[2] Hansmann U. H. E., *Chem. Phys. Lett.*, **281** (1997) 140.
[3] Sugita Y. and Okamoto Y., *Chem. Phys. Lett.*, **314** (1999) 141.
[4] Fukunishi H., Watanabe O. and Takada S., *J. Chem. Phys.*, **116** (2002) 9058.
[5] Jang S., Shin S. and Pak Y., *Phys. Rev. Lett.*, **91** (2003) 58305.
[6] Liu P., Kim B., Friesner R. A. and Berne B. J., *Proc. Natl. Acad. Sci. U.S.A.*, **102** (2005) 13749.
[7] Sugita Y., Kitao A. and Okamoto Y., *J. Chem. Phys.*, **113** (2000) 6042.

[8] BUSSI G., GERVASIO F. L., LAIO A. and PARRINELLO M., *J. Am. Chem. Soc.*, **128** (2006)
     13435.
[9] SINDHIKARA D., MENG Y. and ROITBERG A., *J. Chem. Phys.*, **128** (2008) 024103.
[10] ALLEN M. P. and TILDESLEY D. J., *Computer Simulation of Liquids* (Oxford University
     Press, Oxford) 1987.
[11] MARINARI E. and PARISI G., *Europhys. Lett.*, **19** (1992) 451.
[12] HAGEN M., KIM B., LIU P., FRIESNER R. and BERNE B., *J. Phys. Chem. B.*, **111** (2007)
     1416.
[13] `http://www.mpi-forum.org/`.
[14] ZHANG L., PARASHAR M., GALLICCHIO E. and LEVY R., *Salsa: Scalable Asynchronous
     Replica Exchange for Parallel Molecular Dynamics Applications*, in *Proceedings of the 2006
     International Conference on Parallel Processing* (IEEE Computer Society Washington,
     DC, USA) 2006, pp. 127-134.
[15] GALLICCHIO E., LEVY R. and PARASHAR M., *J. Comput. Chem.*, **29** (2008) 788.
[16] BUSSI G., DONADIO D. and PARRINELLO M., *J. Chem. Phys.*, **126** (2007) 014101.