Evaluation of MPI's One-Sided Communication Mechanism for Short-Range Molecular Dynamics on the Origin2000

T. Matthey[1] and J. P. Hansen[2]

[1] Department of Informatics, University of Bergen
Høyteknologisenteret i Bergen, N-5020 Bergen, Norway
Thierry.Matthey@ii.uib.no
[2] Institute of Physics, University of Bergen
Allégaten 55, N-5007 Bergen, Norway
JanPetter.Hansen@fi.uib.no

Abstract. In this paper we evaluate the possibilities of one-sided communication, a new feature of the MPI-2 standard, on the Origin2000 for relatively short-range molecular dynamics (MD) simulations. Our algorithm is based on an asynchronous message-passing multi-cell approach using MPI as message-passing layer and the Leap-Frog/Verlet algorithm for the time integration. We compare one-sided with two different two-sided communication approaches for typical production runs ($10^5 - 10^9$ atoms) where we discuss the communication vs. computation time for increasing number of processes. We also show how the partitioning of the problem affects the different communication approaches. Using one-sided communication we achieved 10-70% better performance over two-sided.

1 Introduction

Molecular dynamics (MD) [1,2] has been known for several decades and successfully used in atomistic simulation models with a few thousands interacting particles. With the introduction of large scale parallel computers it became possible to study more realistically sized systems[3,4] and nowadays MD is an appreciated tool for a whole range of scientists to study dynamic properties of micro- or macro phenomena [5].
For computer scientists MD simulation is interesting in order to evaluate new parallelization and communication approaches, since it includes several challenging parallelization problems. MD simulations may also be used in benchmarking, to study the efficiency of different machines.
In this paper we evaluate the one-sided communication mechanism of MPI-2[6] for relatively short-range MD simulations. For our evaluation we use a shared memory machine, the Origin2000 with 128 CPU's. Our system contains approximately $10^5 - 10^7$ particles representing a general metal atom with a dense, regular distribution. The implementation is well optimized[7] and in C++. For testing and evaluation purposes we implemented the two-dimensional case, but

the design and the algorithms extend naturally to higher dimensions. MPI[1] was used as communication layer.

The paper is organized as follows: In Section 2 we review the basic physical principles for MD simulations, in Section 3 we explain the parallelization, distribution and load balancing of the work. In Section 4 we describe our three different communication approaches based on MPI and in Section 5 we give a short overview of the one-sided communication mechanism of MPI-2. In Section 6 we discuss the timing results and in Section 7 we close with a conclusion and prospects for future work.

2   Short-Range Molecular Dynamics

The MD method is based on the solution of Newton's equation of motion for N interacting particles. This general N-body problem first involves the calculation of $N(N-1)/2$ pairs of interactions to compute all forces. On a given particle at position $\boldsymbol{r_j}$,

$$m_j \ddot{\boldsymbol{r_j}} = \boldsymbol{F_j} = \sum_{i=1, i\neq j}^{N} -\nabla u(r_{ij}). \tag{1}$$

Here m_j is the mass of particle j and r_{ij} is the distance between particle i and j. The complexity of the force calculation is governed by the potential function $u(r_{ij})$. To simplify the potential we define a finite range of interaction, which is a reasonable approximation of atomistic interactions. In our evaluation we use the general Lennard-Jones 6-12 (LJ) potential,

$$u(r_{ij}) = \begin{cases} 4\epsilon \left[\left(\frac{\alpha}{r_{ij}}\right)^{12} - \left(\frac{\beta}{r_{ij}}\right)^{6} \right] : 0 < r_{ij} \leq r_{\text{cutoff}} \\ 0 \qquad\qquad\qquad\qquad\quad : r_{\text{cutoff}} < r_{ij} \end{cases}. \tag{2}$$

Where ϵ, α and β are LJ parameters with $\alpha \approx \beta$, often is $\alpha = \beta$, it is called σ. The potential $u(r_{ij})$ is cut-off at r_{cutoff}, i.e. no interactions are evaluated beyond this distance.

The number of interacting neighbors for each particle is determined by the interaction range r_{cutoff}, and the particle density ρ. Considering that the repulsive core of $u(r_{ij})$ limits the local density (the maximal number of particles inside the range), we can conclude that the number of interactions has an upper constant bound, i.e. does not depending on the number of particles, N. Once the forces for each particle are computed, the positions and velocities have to be updated. For our purpose we use the Leap-Frog/Verlet algorithm, which despite its low order of accuracy has excellent energy conservation properties[2].

3   The Multi-cell Algorithm and Distribution

The multi-cell algorithm[3] provides a means of organizing the spatial information for each particle into such a form that the particle's neighbors can be quickly

---

[1] MPT 1.4 from SGI.

located for the force calculation. Thus the number of interactions evaluated is minimized. We briefly outline the main features of the computation for the two-dimensional case, which extends directly to three dimensions.

The region is divided into rectangular, equal cells (Fig. 1) with dimensions at least the r_{cutoff} distance. Particles are assigned to the cells geometrically according their positions. The computation of the force on a single particle involves only the particles of the same cell and the neighboring cells. The evaluation of forces for all particles in a cell consists of two steps. First all interactions between particles in the original cell (5) are computed. Then the forces between neighboring cells (6,3,2,1) and the original cell are computed, by following an interaction path that describes how the interactions with neighboring cells should occur. Following the path the interactions for the original and the visited cells are
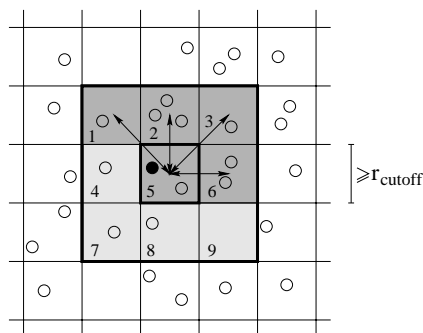

Fig. 1. Partition of cells, where cell nr. 5 is the origin and 1-4,6-9 the neighbor cells. Cell nr. 6,3,2 and 1 describe the interaction path.

accumulated, e.g. using Newton's third law. The original cell (5) will accumulate the interactions from its lower neighbors when they calculate their interactions. To calculate all forces, this procedure has to be carried out for each cell. However, the defined path and the assumption that the dimensions of cells are at least r_{cutoff}, guarantee that all interactions with a contribution will be taken into acount and the total amount of computed interactions is of order $O(N)$.

With the cell structure now in place, the particles can be distributed. Each cell with its particles is assigned to a particular process by a given partitioning algorithm. To study how partitioning affects the amount of communication and communication conflicts (i.e. two or more processes request/need the same cell) we introduced three different approaches to distribute the cells. The first partitioning (Fig. 2, *stripe*) uses a one-dimensional array partitioning simply assigning an approximately equal stripe to each process. Fig. 3 illustrates a partitioning (*metis*) minimizing the interface between processes for large systems. Our implementation uses METIS[8] – a family of programs for partitioning unstructured graphs and hypergraphs. The last partitioning (Fig. 4, *rect*) is a two-dimensional block partitioning trying to minimize the interface. Additionally, for the purpose
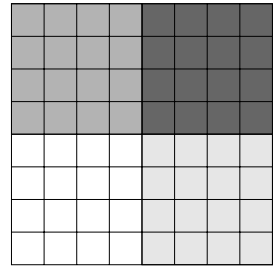
of testing the robustness of the communication approaches we used for some smaller systems a partitioning where we scattered the cells, trying to assign neighboring cells to as many as possible different processes.
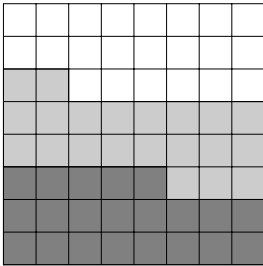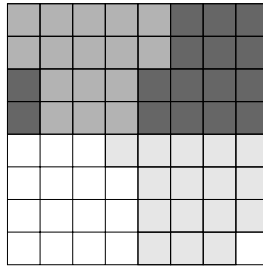
Fig. 2. Stripe.	**Fig. 3.** METIS.	**Fig. 4.** Rectangle.

4   The Communication Approaches

Each process calculates the interactions in parallel, proceeding through all assigned (locally) – the owned – cells sequentially. As long as the neighboring cells are owned, the cell is independent from other processes. Once a process needs a neighboring cell owned by another process – a *shared* cell, we have to make sure that this cell with its particles is made available. Note that the passing of a cell may happen by either an explicit or implicit request. We have studied three communication approaches where one is based on MPI's one-sided communication mechanism and the other two on two-sided communication.

First we discuss the two-sided communication approaches which require a matching send for each posted receive. This criterion synchronize the involved processes and can in the worst case lead to a dead-lock. The first approach (*simple*) does an implicit passing depending strongly on the distribution algorithm, i.e. it works only for the *stripe* partitioning. Each process starts with the cells, which will be used (later) by another process. Once these cells are not needed any more for further computations they are sent asynchronously to the appropriate process. As soon as the receiver needs these cells it will receive them and continue to compute. The cells are sent back when the process has completed its computations. To simplify we assume that each process will request cells from only one process. *stripe* is the only approach which requests cells from at most one process.

The second approach (*dist*) passes cells by explicit requests. The owner of the cell will decide if it can give away the requested cell, i.e. the cell is not needed by the owner itself. The requesting process will hold the cell as long as the owner does not request it back, and the process has not finished all its computations. Thus the cell may be used for further interactions and passed back only if it is

really needed. Further requests may occur for a cell, which actually is held by a third process. The owner has then to request back the cell and pass it to the requesting process. All this leads to a more complicated communication policy, but the method will handle all kind of partitionings including scattered distribution. Additionally we can show that the cells are coherent at any time.

The last approach (*one*) is based on MPI's one-sided communication mechanism. Before any cell can be passed we have to define the (memory-)*window*'s for the remote access (see next section) for *shared* cells, which will be needed by other processes. We assume that the allocated storage in each cell is large enough for the whole run, otherwise we have to redefine the *window* later. Before computing the forces each process will copy and make public the *shared* cells in its own *window*. Whenever a process needs a cell it will grab the cell from the according process *window*. Once the processes have computed all interactions they write their contribution of the *shared* cells for each process in a separate *window*. Then the owner of the *shared* cells will collect and accumulate all contributions scattered among all (neighboring) processes. The approach has a simple policy without any synchronization when passing or requesting cells. We elaborate on this communication mechanism next.

5   MPI's One-Sided Communication Mechanism

One-sided communication[6] or Remote Memory Access (RMA) extends the communication mechanisms of MPI by allowing one process to specify all communication parameters, both for the sending side and for the receiving side. This mode of communication suits applications with dynamically changing data access patterns, where the data distribution is fixed or slowly changing. For such applications, each process can access or update data at other processes where the remote process may not know which data in their memory need to be accessed or updated. RMA communication mechanisms also avoid the need for global communication or explicit polling.

Where regular send/receive communication requires matching operations by sender and receiver, RMA requires once the definition of the *window*[2] for the remote memory access or update. The definition of the *window* – a collective operation – specifies the memory that is made accessible to remote processes. Once the *window* is not any more needed or obsolete, it should be freed[3] and if necessary redefined.

The access[4] is similar to the execution of a send by the target process, and a matching receive by the origin process with the obvious difference that the receiver does not depend on the sender. The update[5] works similarly, except that the direction of data transfer is reversed. For a more sophisticated update than

---

[2] `MPI_WIN_CREATE().`

[3] `MPI_WIN_FREE().`

[4] `MPI_GET().`

[5] `MPI_PUT().`

copying data from the origin to the target process, a general accumulation function[6] is provided. RMA synchronization *fence*[7] – a collective operation – entails the completion of RMA's and the synchronization of RMA calls on the given *window*. Beside *fence*, there are several functions to implement finer and more subtle synchronization of RMA calls.

6   Results

To monitor the different communications approaches and distributions we performed a variety of 190 test runs with different lattice sizes (Table 1) and number of CPU's. All test runs were performed for 200 timesteps representing 1.01805×10^{-4} [ns]. The lattices are two-dimensional and consist of 89.9% particles representing an idealized, optimal packed metallic alloy with 10% of an additive atomic species and 0.1% holes. Table 1 shows the number of particles

Table 1. The three different test cases. L2-bound is the lower bound of CPU's required so that all particles fit into the L2-cache on each process.

Cases	Particles	Cells	L2-bound	Sequential [s]
small	91,800	82×82	1.93	193.4
medium	925,600	260×259	19.4	2300.0
large	9,218,300	820×819	193.4	24692.5

for each test case and the number of cells. L2-bound is the lower bound of CPU's required so that all particles fit into the L2-cache (4MB) on each process, assuming an equal distribution. Note that each process will required some additional memory for the communication buffers (*window*'s) containing the *shared* cells. The last column is the time of a sequential run. Periodic boundaries were applied in all dimensions and $r_{\text{cutoff}} \approx 3\sigma$, where each particle has approximately 43 interacting neighbors and approximately 14 particles per cell. The test runs for one to 32 CPU's were performed on a normally loaded machine, where for 64 or more CPU's the machine was empty.

The following tables and figures is a selection of the runs of interest. Tables 2-4 are a summary of the achieved speedups and the gain of *one* compared to *dist*. The bold numbers represent the best achieved speedup for each approach. Fig. 13, 5 and 6 are the corresponding graphs for *small*, *medium* and *large*. Figs. 7, 9 and 11 show the time spent in [s] for the communication, which includes the redistribution of particles moving from one process to another and idle cycles when a process have to wait to receive data. Figs. 8, 10 and 12 present the time spent for the computation of forces and computation of new positions and velocities (upper curves) relatively to the total run time. Additionally the update timings, which is the amount spent for the update of the cell structure, are shown.

---

[6] `MPI_ACCUMULATE().`
[7] `MPI_WIN_FENCE().`

Table 2. Speedups for the test case *small*, 91,800 particles.

CPU's	*dist*			*one*				*simple*	Cells/
	stripe	*metis*	*rect*	*stripe*	*metis*	*rect*	Gain	*stripe*	CPU
1	1.00	–	–	–	–	–	–	–	6,724
2	1.45	1.49	**1.62**	1.63	**2.00**	1.80	23%	1.85	3,137
4	3.15	3.31	**3.38**	**4.21**	3.79	3.65	25%	3.67	1,569
8	4.21	**5.64**	5.60	5.80	**7.71**	6.86	37%	6.47	841
16	8.15	**8.92**	8.28	10.93	11.88	**12.49**	40%	11.70	421
32	13.83	**14.58**	14.44	17.19	**21.09**	20.16	45%	16.35	211
64	18.43	22.98	**24.10**	30.87	**41.68**	38.59	73%	–	106
121	21.30	29.30	**32.41**	30.50	33.25	**33.52**	3%	–	56

Table 3. Speedups for the test case *medium*, 925,600 particles.

CPU's	*dist*			*one*				*simple*	Cells/
	stripe	*metis*	*rect*	*stripe*	*metis*	*rect*	Gain	*stripe*	CPU
1	1.00	–	–	–	–	–	–	–	67'346
2	**1.77**	1.67	1.66	1.69	**1.83**	1.75	3%	1.61	33,673
4	3.08	**3.31**	2.83	**3.43**	3.14	3.15	4%	3.34	16,837
8	5.74	5.71	**5.85**	6.24	6.29	**7.17**	23%	6.92	8,419
12	7.23	8.01	**8.09**	9.06	**9.27**	7.87	15%	–	5,613
16	**10.39**	8.50	9.66	10.79	11.74	**12.56**	21%	12.39	4,210
20	**12.05**	10.84	10.99	14.71	**15.07**	14.22	25%	–	3,368
24	13.26	12.55	**13.70**	13.70	**15.23**	15.08	11%	–	2,807
28	12.37	13.80	**14.07**	19.23	17.86	**23.66**	68%	–	2,406
32	15.71	18.17	**21.76**	21.72	24.54	**24.90**	14%	23.11	2,105
64	–	**42.42**	–	–	**62.15**	43.47	47%	44.50	1,053
121	–	–	**54.43**	–	–	**102.73**	89%	62.97	557

Table 4. Speedups for the test case *large*, 9,218,300 particles.

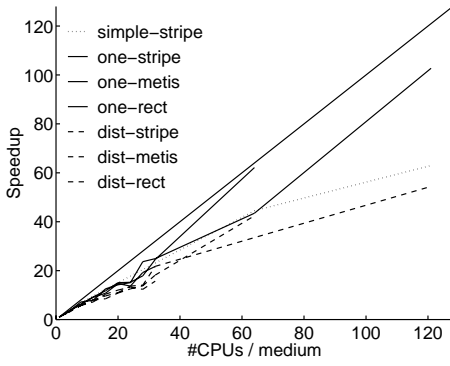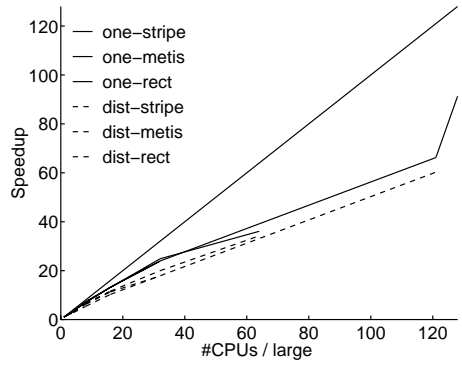| CPU's	*dist*			*one*				Cells/
	stripe	*metis*	*rect*	*stripe*	*metis*	*rect*	Gain	CPU
1	1.00	–	–	–	–	–	–	671,580
2	1.73	1.70	**1.76**	**1.93**	1.84	1.89	10%	335,790
4	3.27	**3.33**	2.80	3.41	3.30	**3.65**	10%	167,895
8	6.17	**6.69**	5.40	7.05	**7.37**	6.95	10%	83,948
16	11.31	**11.62**	10.28	**13.22**	13.12	13.20	14%	41,974
32	17.82	**19.96**	–	23.83	**24.84**	23.93	24%	20,987
64	–	**34.24**	–	–	**36.09**	–	5%	10,494
121	–	–	**60.25**	–	–	**66.22**	10%	5,551
128	–	–	–	–	**91.31**	–	–	5,247

Fig. 5. Speedups, *medium.*
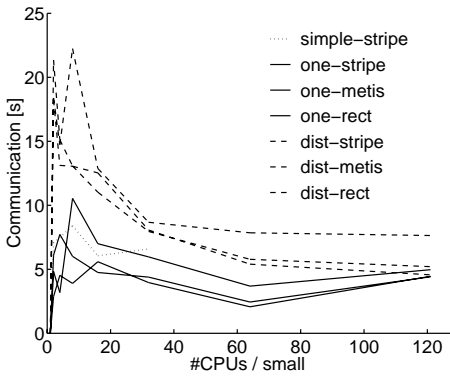

Fig. 6. Speedups, *large.*

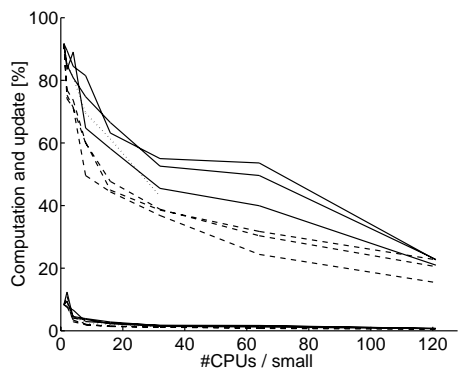**Fig. 7.** Communication [s], *small.*

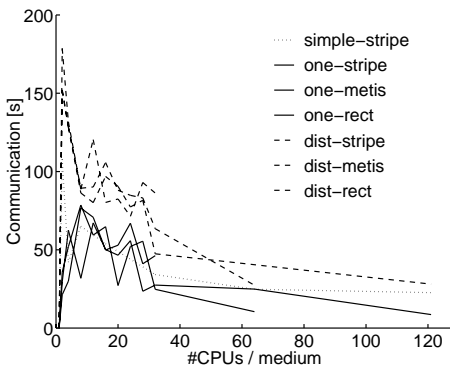**Fig. 8.** Computation & update, *small.*

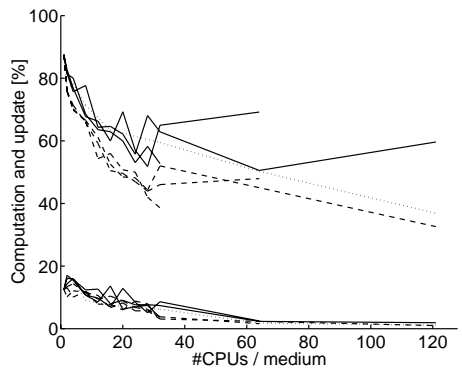**Fig. 9.** Communication [s], *medium.*

Fig. 10. Computation & update, *medium.*
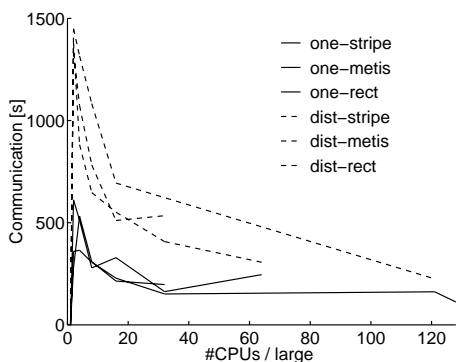
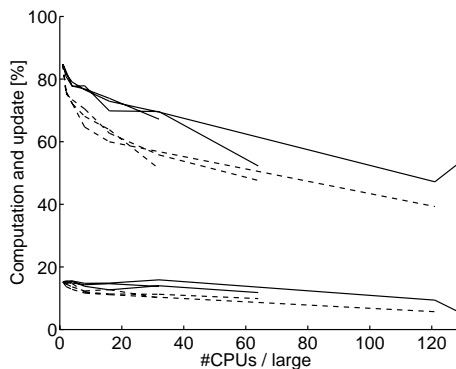**Fig. 11.** Communication [s], *large.*

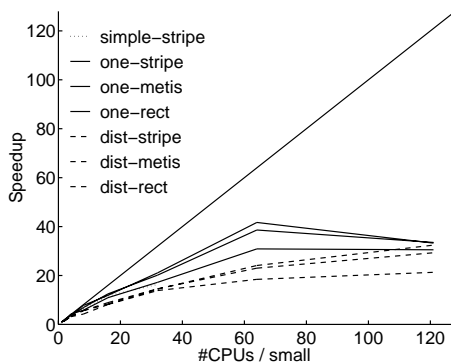**Fig. 12.** Computation & update, *large.*

Fig. 13. Speedups, *small.*

7   Discussion and Outlook

From the figures it can be concluded that the use of MPI's one-sided communication mechanism on the Origin2000 is not only feasible but it improves performance as well. The gain of the one-sided communication approach – not to have to wait for a process sending data – is about 10% (Table 4). As soon as the domains get smaller and requesting conflicts occur more often, we get 20% up to 80% better performance. From Figs. 7, 9 and 11 we see clearly that the time spent for communication depends linearly on the number of particles regardless of the distribution approach: *one* compared with *dist* is more or less unaffected by the amount of communicated data since it uses non-blocking communication and does not need to wait when receiving data. Better performance, however, requires more memory; *one* will request more then twice the memory if the domains have less than 320 cells. Nevertheless we obtain a better performance, even around the L2-bound when only *dist* completely benefits from the L2-cache.

For the partitioning we did not find a clear "best choice": In some cases *rect*

is slightly better than *metis* for large number of CPU's. METIS may produce a partitioning with non-connected domains and domains with irregular shape, which introduces more communication. Furthermore we noticed that *metis*'s irregular shape of domains reduces the conflicts (e.g. several processes requesting the same cell at the same time), especially for *dist. rect* may not generate an optimal load balance for small number of cells or for large numbers of CPU's with few, unequal factors (e.g. prime numbers). *stripe* is a competitive partitioning for small numbers of CPU's and scales better for huge number of cells.

With the cell algorithm we get an excellent local memory access pattern since the particles are stored in a dense array in each cell. This can be observed in Fig. 8 where the time spent for the update of the cells drops down from 10% for one CPU to 2% for four CPU's; we even get a super linear speedup. For *medium* (Fig. 10) we see that already for 16 CPU's parts of L2-cache can be reused for the next timestep, but it fits completely into L2-cache after 32 CPU's. *large* (Fig. 12) does not benefit from the L2-cache at all until 128 CPU's as expected (L2-bound in Table 1).

In future we will extend our systems to three dimensions and approve the timing number for this case. We here expect an even better improvement since the three-dimensional case involves more communication, which has been shown by [3]. The memory usage for *one* should be improved especially for the three-dimensional case in order to avoid running out of memory. Furthermore, we will consider dynamic load balancing to improve the performance for non-regular systems or systems where the local density is changing heavily.

References

1. M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Oxford Clarendon Press (1987).
2. D.C. Rapaport , *The Art of Molecular Dynamics Simulation*, Cambridge University Press, (1995).
3. D.M. Beazley, P.S. Lomdahl, *Message-Passing Multi-Cell Molecular Dynamics on the Connection Machine 5*, Theoretical Division and Advanced Computing Laboratory Los Alamos National Laboratory, Las Alamos, New Mexico 87545, (1993).
4. P.S. Lomdahl, P. Tamayo, N. Grøbech-Jensen, and D.M. Beazley, *50 GFLops Molecular Dynamics on the Connection Machine 5*, Theoretical Division and Advanced Computing Laboratory Los Alamos National Laboratory, Las Alamos, New Mexico 87545, (1993).
5. RS. J. Zhou, D. L. Preston, P. S. Lomdahl and D. M. Beazley, SCIENCE, **279**, 1525 (1998)
6. *MPI-2: Extensions to the Message-Passing Interface*,
   `http://www.mpi-forum.org/docs/docs.html`.
7. Igor Zacharov, *Origin Optimisation and Parallelisation Training*, notes, SGI, Bergen, Norway, (2000).
8. *METIS – Family of Multilevel Partitioning Algorithms*,
   `http://www-users.cs.umn.edu/~karypis/metis/metis.html`.