John von Neumann Institute for Computing

Communication and Load Balancing of Force-Decomposition Algorithms for Parallel Molecular Dynamics

Godehard Sutmann, Florian Janoschek

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 45-52, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

© 2007 by John von Neumann Institute for Computing
Permission to make digital or hard copies of portions of this work for personal or classroom use is granted provided that the copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise requires prior specific permission by the publisher mentioned above.

http://www.fz-juelich.de/nic-series/volume38

Communication and Load Balancing of Force-Decomposition Algorithms for Parallel Molecular Dynamics

Godehard Sutmann[1] **and Florian Janoschek**[2]

[1] Jülich Supercomputing Centre (JSC)
Research Centre Jülich (FZJ)
D-52425 Jülich, Germany
E-mail: g.sutmann@fz-juelich.de

[2] Stuttgart University
Institute for Computational Physics, Pfaffenwaldring 27
D - 70569 Stuttgart, Germany
E-mail: FJanoschek@gmx.net

A new implementation of a force decomposition method for parallel molecular dynamics simulations is presented. It is based on a geometrical decomposition of the influence matrix where sections are dynamically reorganized during the simulation in order to maintain a good load balance. Furthermore space filling curves are used to sort particles in space, which makes memory access more efficient and furthermore reduces communication between processors due to better locality. Benchmark runs are presented which shows in improvement in scalability due to well balanced work on the processors.

1 Introduction

Classical molecular dynamics simulations are often considered as the *method par excellence* to be ported to parallel computers, promising a good scaling behaviour. On the one hand parallel algorithms exist which enable good scaling. On the other hand the complexity of the problem at hand, often scales like $\mathcal{O}(N)$, enabling a linear increase of problem size with memory. However, this point of view applies only to a limited class of problems which can be tackled by molecular dynamics. E.g. in the case of homogeneous periodic systems, where particles interact via short range interactions, the most efficient algorithm is a domain decomposition scheme, guaranteeing local communication between processors and therefore allowing good parallel scaling. In combination with linked-cell lists, the problem scales like $\mathcal{O}(N)$ both in computational complexity and memory demand, so that an ideal behaviour in both strong and weak scaling might be expected.

On the other hand, this ideal behaviour breaks down if different problem classes are considered, e.g. the case of long range interactions, where not only local communications between processors are required. Another class of counter examples is the case of inhomogeneous systems, which occur e.g. in open systems, where the particle density is considerably larger in the centre of the system than in the diffuse halo or e.g. in systems consisting of different thermodynamic phases as is the case for the coexistence of liquid/-gas or solid/gas phases. In this case, domain decomposition algorithms often fail. Due to a more or less regular geometric decomposition of space, processors are responsible for different numbers of particles, often introducing a strong load-imbalance, which leads to inefficient CPU usage on some processors and therefore to a bad parallel scaling.

This fact requires some sort of load-balancing strategy, which is able to rearrange domains dynamically in order to achieve equal work load through a simulation, which might pass a non-homogenous dynamical particle flow across processors.

In literature, there exist several different approaches to solve the problem of load-imbalance. A general diffusion scheme, which uses near neighbour information was introduced in Ref.[1]. For two-dimensional MD simulations a one-dimensional Cellular Automaton Diffusion scheme was proposed in Refs.[2,3], which was further generalized to three-dimensional MD in Ref.[4] via the concept of *permanent cell* to minimize interprocessor communication. For one- and two-dimensional MD, based on a parallel link-cell domain decomposition method, the Multilevel Averaging Weight was introduced in Ref.[5]. For fully three dimensional MD simulations, based on a linked-cell domain decomposition strategy a load-balancer was proposed in Ref.[6]. For a force-decomposition method, based on Ref.[7], a method based on the generalized dimension exchange was presented in Ref.[8].

In the following a new decomposition scheme, based on an a distributed data model with MPI communication protocol, is presented. It combines elements from force- and domain-decomposition approaches. First, the basic concept is introduced. This is extended to reduce communication overhead and combined with a load balancing strategy. Finally, benchmark results are presented for different kinds of model systems.

2 Force-Domain-Decomposition Scheme

2.1 The Connectivity Matrix

The connectivity matrix $\mathbf{C} \in \mathbb{N}^{N\times N}$ contains information of the system, which particle pairs (i,j) interact with each other, i.e. $C_{ij} = \{1 : \|\mathbf{r}_i - \mathbf{r}_j\| \in \Omega_I\}$, where $\mathbf{r}_k$ is the geometric coordinate of particle k and Ω_I is the range of interaction. For short range interactions, this corresponds to a cutoff radius R_c, beyond of which interactions are switched off. If $\mathbf{C}$ is dense, this corresponds to long range interactions. On the other hand, short range interactions will result in a sparse structure. Since mutual interactions are symmetric, it is clear that also $\mathbf{C}$ is symmetric. Furthermore it is traceless, because there are no self interactions of particles considered. Therefore, taking $\mathbf{C}$ as the basic building block for a parallel algorithm, it is sufficient to consider the upper triangular matrix $\mathbf{C}^u$. Considering on average the system to be homogeneously distributed, a parallel strategy consists in distributing domains of equal area of $\mathbf{C}^u$ to the processors. Various methods were discussed in Ref.[9]. The one, which forms the basis for the following implementation is the stripped-row method. Subsequent rows are combined to blocks, which have nearly similar size, A. The recursive relation

$$X_k = \frac{1}{2}\left(Q_k - \sqrt{Q_k^2 - \frac{4N(N-1)}{P}}\right) \tag{1}$$

with

$$Q_k = 2N - 1 - 2\sum_{j=0}^{k-1} X_j \tag{2}$$

defines a fractional number of rows in $\mathbf{C}$. Taking the integer part $L_k = \lfloor X_k \rfloor$, the area, which is assigned to each processor is calculated as

$$A(L_k) = \left(N - \sum_{j=0}^{k-1} L_j - \frac{L_k + 1}{2} \right) L_k \tag{3}$$

Here, $k \in [0, P-1]$ and consistency has to give $\sum_{k=0}^{P-1} L_k = N$. This scheme defines a load balanced distribution, if the matrix is dense or if the probability of $P(j > i|i)$ for finding a particle $j > i$ for a given particle i in Ω_I is constant. In addition, this scheme will also only work satisfactorily, if memory access is uniform all over the system.

2.2 Communication

Considering a system, where particles are free to move in the simulation, it is not clear *a priori*, which particle pairs $(i, j) \in \Omega_I$. Therefore, traditional schemes of force decomposition algorithms require a replication of the whole system[9] or replication of full subsets of coordinates[7]. In both cases positions are usually distributed by an `mpi_allgatherv` command while forces are summed up by an `mpi_allreduce` procedure. This is certainly the most simple way to proceed. Since most MPI implementations internally make use of a tree-wise communication protocol, the global communication will scale like $\mathcal{O}(\log_2(P))$, if P is the number of processors.

In general there are two steps where communication between processors is required: (i) Exchange of coordinates. Due to the partitioning of the connectivity matrix in the present algorithm, coordinates are transferred only to processors with lower rank than the local one, i.e. the processor with rank $P-1$ only calculates local forces, but it sends coordinates to remote processors. (ii) Exchange of forces. Due to the skew-symmetric nature of forces between particles i, j, forces are sent analogously, but in the back direction of the coordinate flow.

However, for big systems and large number of processors, there will be a lot of redundant data transferred between processors. I.e. global communication operations do not take into account whether transferred data are really needed on remote processors. Therefore an alternative method is considered here.

The basic principle is to combine a domain decomposition with a force decomposition scheme. Domain decomposition is achieved by sorting the particles according to their positions along a space filling Hilbert curve[10]. This ensures that most particles which are local in space are also local in memory. For short range interactions this implies that most interaction partners are stored already on the same processor. According to this organisation, the force matrix becomes dominant around the diagonal and off-diagonal areas are sparse. The next step is to calculate interaction lists, which are distributed onto all processors. In the present implementation, interactions are calculated via Verlet lists, which store indices of particle pairs $(i, j) \in \Omega_{I+R}$, where Ω_{I+R} is the actual interaction range plus a reservoir of potentially interacting particles j. Therefore, during construction of these lists, exchange lists, $\mathcal{L}_X(p\{I\} \rightarrow p'\{J\})$, can be created, which store information, which index set $\{I\}$ from the local processor p has to be transferred to the remote processor p' to calculate interactions with particles from index set $\{J\}$. On the other hand this immediately gives a list $\mathcal{L}_F(p'\{J\} \rightarrow p\{I\})$ which contains information, which forces have to be sent

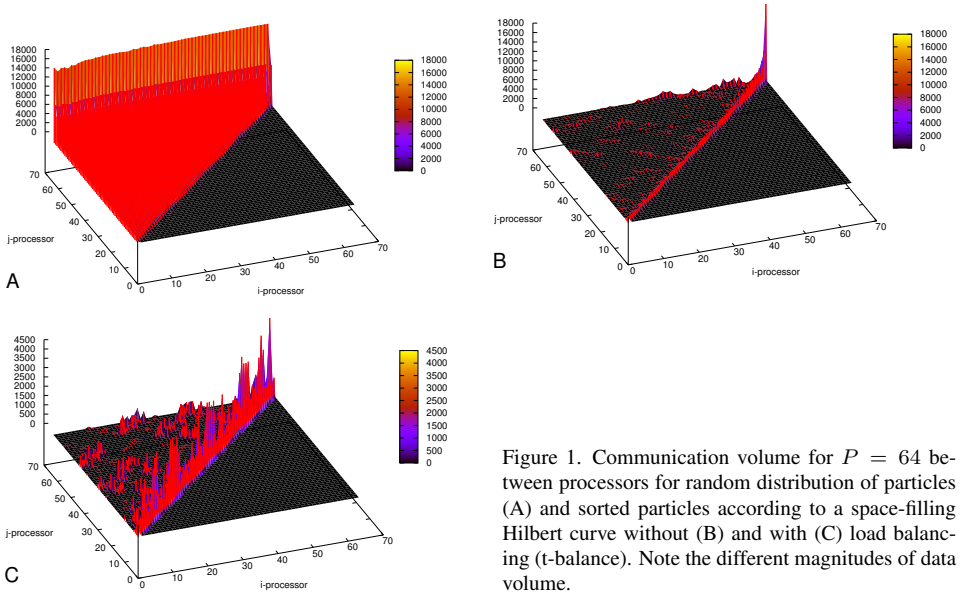

Figure 1. Communication volume for $P = 64$ between processors for random distribution of particles (A) and sorted particles according to a space-filling Hilbert curve without (B) and with (C) load balancing (t-balance). Note the different magnitudes of data volume.

from processor p to p'. It is the latter list, which is communicated to processors with lower rank than the local one. Verlet lists only have to be created from time to time (this may be controlled with an automatic update criterion[11]). Therefore, neighbour lists, as well as exchange lists only have to be created every one of these time intervals (usually neighbour lists are updated after ≈ 20 timesteps, depending on list parameters[11]).

Since the amount of data is very small with respect to global communications of positions and forces, the communication overhead of this method is strongly reduced, enabling a very much better parallel scaling. Since randomisation of particle positions occur on a diffusive time scale, the space filling curve has to be updated only one or two orders of magnitude less than the interaction lists, thus introducing only a small overhead. Therefore, it is not a main bottleneck that at the moment, sorting is done with a sequential algorithm. However, parallel algorithms exist, which will be built into the algorithm in near future.

2.3 Load-Balancing

Having set up the particle distribution according to Eq. 3, does not guarantee a load balance among processors. Considering e.g. inhomogeneous systems, where particles cluster together, this may lead to different work load on processors. If particles on one processor mainly belong to a certain cluster, in which they are densely packed, they will create a large number of interactions on this processor. This might be different on another one, which stores particles belonging to a diffusive halo, and therefore generating less interactions.

Therefore, for inhomogeneous systems, this approach might become quite inefficient. Therefore, two different work measures are defined which lead to two different load balancing strategies. The first is based on the number of interactions (n-balance), which are

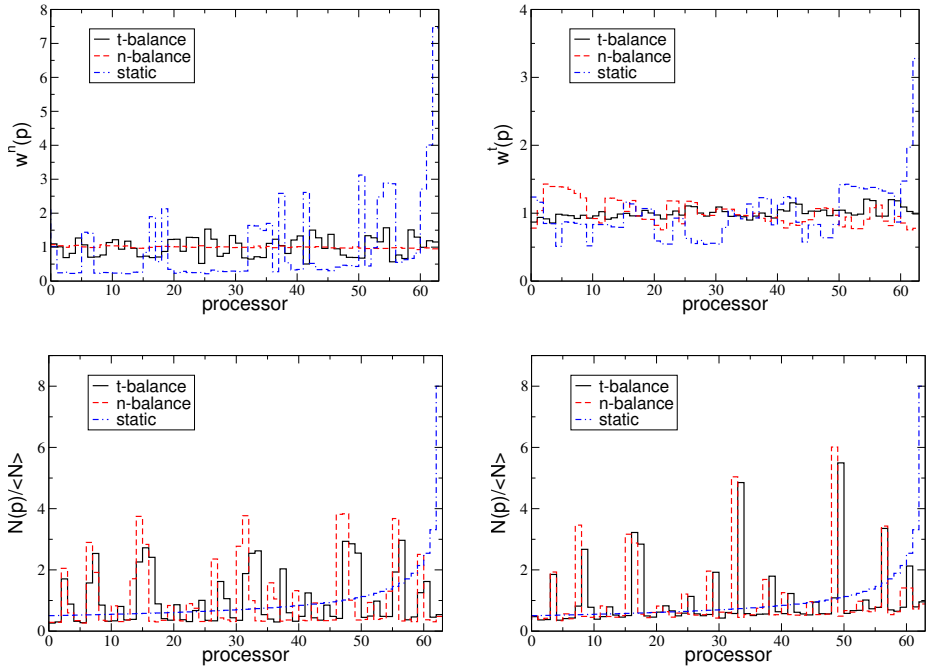

Figure 2. Work load of the static-, t-balance- and n-balance schemes. Upper left: number of interactions, $w^n(p)$; upper right: CPU time, $w^t(p)$, lower left: relative number of particles/processor. All cases for inhomogeneous system. Lower right: relative number of particles/processor for the homogenous system.

calculated on every processor, $n(p_i)$, $i \in [0, P-1]$. Here, one may define the ratio

$$w^n(p_i) = \frac{n(p_i)}{\langle n \rangle_P} \quad , \quad \langle n \rangle_P = \frac{1}{P} \sum_i n(p_i) \tag{4}$$

The second method is similarly defined, but uses the time $t(p_i)$, consumed on every processor (t-balance)

$$w^t(p_i) = \frac{t(p_i)}{\langle t \rangle_P} \quad , \quad \langle t \rangle_P = \frac{1}{P} \sum_i t(p_i) \tag{5}$$

In both cases, if $w = 1 \; \forall \; p_i$, a perfect balance for the force routine is achieved.

Practically, the load balance routine is called, before the interaction lists are being constructed. Within the time interval between list updates, each processor gathers information about the number of interactions or the consumed time [a]. In the load-balancing step, the average quantities per row is calculated. For n-balance, an `mpi_allgatherv` operation collects the number of local interactions on every processor. This vector is then analysed and it is decided, how many rows, i.e. particle information, are transferred or obtained from $p_i \rightarrow p_{i-1}$ and $p_i \leftarrow p_{i+1}$. Similarly it is proceeded in the case of t-balance. Here,

[a] At present, only the CPU time which is consumed in the force routine is considered. Since this is both the most time consuming routine and the only one in which loop structures are likely to develop an imbalance, an even distribution of this time leads to an overall balance of the processes.

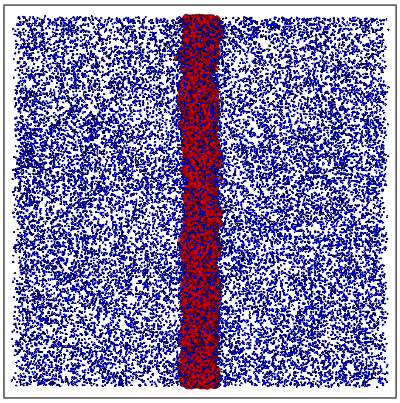

Figure 3. Snapshot of the inhomogeneous system after several thousand time steps. The heavy, immobile particles form a wire structure in a gaseous environment.

`mpi_allgatherv` operation collects the total time, spent on a processor. In a second step, this is translated to an average time t_i per particle, i.e. per row. It was found out that this is more precise and efficient than the measurement of time for each individual particle. According to the time t_i a repartitioning of rows can be performed across processor boundaries.

Since the load imbalance develops with the movement of particles across geometrical domain regions, it is the diffusive time scale which governs this process. Calling the load balance routines with a similar frequency than interaction lists, this ensures a rapid convergence of the load. Also, developing inhomogeneities may be easily compensated by this scheme. Nevertheless, it should be noted that the balancing routine always uses information from the past in order to construct the particle distribution for the next future. Therefore, load balancing is most likely to be not perfect.

3 Benchmark Results

In order to test the algorithms, two types of systems were considered: (i) a homogenous one and (ii) an inhomogeneous one. In both cases, particles interact via a modified Lennard-Jones interaction, which avoids the potential divergence at small particle separations

$$u_{ij}(r_{ij}) = \begin{cases} 48\epsilon \dfrac{1}{\pi} \cos\left(\dfrac{\pi}{2}\dfrac{r_{ij}}{\sigma_{ij}}\right) & r_{ij} < \sigma_{ij} \\ 4\epsilon_{ij}\left(\left(\dfrac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\dfrac{\sigma_{ij}}{r_{ij}}\right)^{6}\right) & r_{ij} \geq \sigma_{ij} \\ 0 & r_{ij} > R_c \end{cases} \tag{6}$$

The homogenous system consists of a binary mixture with parameters $\epsilon_{11} = 120\ K$, $\sigma_{11} = 3$Å and $\epsilon_{22} = 160\ K$, $\sigma_{22} = 3$Å. Combinations are taken into account via usual Berthelot mixing rules. The number density was $\rho = 0.02$ Å^{-3}, the temperature of the system was set to $T = 90\ K$ and masses were taken from Argon and Xenon ($m_1 = 39.9\ amu$, $m_2 = 131.8\ amu$). The cutoff-radius for particle interactions was set to $R_c = 2.5\ \sigma_{ij}$.

In the case of inhomogeneous system, one particle species was assigned an artificially large mass ($m_1 = 10000\ amu$), in order to slow down dynamics and to cluster in the

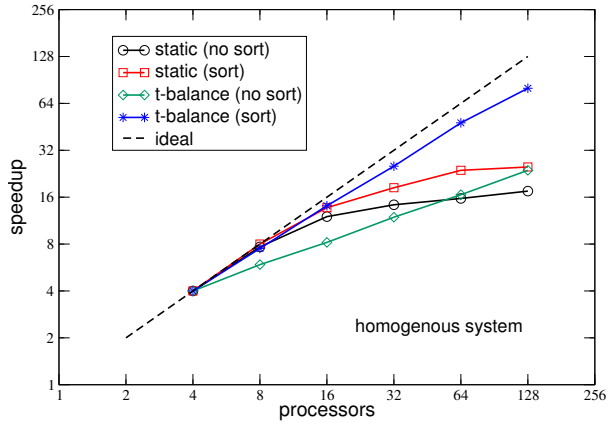

Figure 4. Parallel speedup for the force-domain decomposition method for the case of $N = 10^5$ particles and two different cutoff radii for interparticle interactions.

centre of the simulation box as a wire structure (cmp. Fig. 3). The density was reduced to $\rho = 0.002$ Å^{-3}. Other parameters were $\epsilon_{11} = 50\ K$, $\sigma_{11} = 3$Å, $\epsilon_{22} = 350\ K$, $\sigma_{22} = 3$Å, $T = 150\ K$. Fig. 1 shows results for the load balancing on 64 processors, which was obtained after about 1000 simulation steps. The initial particle distribution onto processors was always chosen according to Eq. 3. Shown is the case for the inhomogeneous system where the load of every processor is measured in terms of number of interactions, CPU time and number of particles, resident on a processor. As it is shown, the static partitioning gives in all cases the worst result. Without load balancing, the processors were out of equal load by a factor of 15. Applying the n-balance scheme, clearly equilibrate the number of interactions within a band of about 10%. However, as is seen from the workload in terms of CPU time, there is still a clear imbalance of about 40%. This is because (i) of different speed of memory access (cache effects are not accounted for in the model) and (ii) because of different waiting times in the communication procedure. If one applies t-balance, a very much better load is observed for the CPU times, while now the number of interactions per processor shows a larger variance ($\approx 50\%$). Looking at the distribution of particle numbers, resident on each processor, a rather non-trivial distribution is found. This is partly due to the distribution of heavy wire particles, which are located in the centre of the simulation box. Using $P = 64$ processors and Hilbert curve sorting of particles, means that approximately half of the processors share heavy particles, while the other half is responsible only for light atoms. For a balanced simulation, processors with only light particles will store more particles, since they exhibit less number of interactions per particle (less dense packed).

Finally a speedup curve for the interaction routine including communication routines is presented for the algorithm. The measurement was performed on the JULI cluster at ZAM/FZJ, consisting of Power 5 processors[12,13]. Shown are results for a homogenous system with $N = 10^5$ particles, where a interaction range of $R_c = 8.5$Å was applied. Cases for static distributions with and without particle sorting are presented as well as those for t-balance. Static distributions saturate already at ≈ 32 processors. Best scaling is

achieved for t-balance with particle sorting. If no sorting is applied, scaling is not linear, because the larger amount of transferred data.

It is noteworthy to state that at present a bottleneck of the method is the construction of interaction lists. At the moment, Verlet lists, from where interaction lists are derived, are constructed by collecting all particles on each processor via an `mpi_allgatherv` operation and then testing mutual distances between particles. Since this is done not frequently, it does not dominate the communication. Nevertheless it limits the overall speedup of the algorithm. To solve this problem, it is planned to use link-cell-lists, in stead of Verlet-lists, in future and to reduce coordinate exchange between processors in the load-balance step to the next smaller and larger ranks. Only in extreme cases, larger rearrangements in index space would be necessary.

References

1. M. H. Wellebeek-LeMair and A. P. Reeves, *Strategies for dynamic load balancing on highly parallel computers*, IEEE Trans. Parall. Distr. Sys., **4**, 979–993, (1993).
2. F. Bruge and S. L. Fornili, *Concurrent molecular dynamics simulation of spinodal phase transition on transputer arrays*, Comp. Phys. Comm., **60**, 31–38, (1990).
3. G. A. Koring, *Dynamic load balancing for parallelized particle simulations on MIMD computers*, Parallel Computing, **21**, 683, (1995).
4. R. Hayashi and S. Horiguchi, *Relationships between efficiency of dynamic load balancing and particle concentration for parallel molecular dynamics simulation*, in: Proc. of HPC Asia '99, pp. 976–983, (1999).
5. M. Zeyao, Z. Jinglin, and C. Qingdong, *Dynamic load balancing for short-range molecular dynamics simulations*, Intern. J. Comp. Math., **79**, 165–177, (2002).
6. N. Sato and J. M. Jezequel, *Implementing and evaluating an efficient dynamic load-balancer for distributed molecular dynamics simulation*, in: International workshop on Parallel Processing, pp. 277–283, (2000).
7. S. Plimpton and B. Hendrickson, *A new parallel method for molecular dynamics simulation of macromolecules*, J. Comp. Chem., **17**, 326, (1996).
8. A. Di Serio and M.B. Ibanez, *Distributed load balancing for molecular dynamics simulations*, in: 16th Annual International Symposium on High Performance Computing Systems and Applications, pp. 284–289, (2002).
9. R. Murty and D. Okunbor, *Efficient parallel algorithms for molecular dynamics simulations*, Parallel Comp., **25**, 217–230 (1999).
10. M. Griebel, S. Knapek, G. Zumbusch, and A. Caglar, *Numerische Simulationen in der Moleküldynamik*, (Springer, Berlin, 2004).
11. G. Sutmann and V. Stegailov, *Optimization of neighbor list techniques in liquid matter simulations*, J. Mol. Liq., **125**, 197–203, (2006).
12. *JULI Project – Final Report*, Eds. U. Detert, A. Thomasch, N. Eicker, J. Broughton, FZJ-ZAM-IB-2007-05, (2007).
13. `http://www.fz-juelich.de/zam/JULI`