Euroconference on Computer Simulation in Condensed Matter Physics Chemistry School 1995 Lecture Notes

Parallelization of Computational Physics Problems

Dieter W. Heermann
Institut für theoretische Physik
Universität Heidelberg
Philosophenweg 19
D-69120 Heidelberg
Germany
and
Interdisziplinäres Zentrum
für Wissenschaftliches Rechnen
der Universität Heidelberg

voice: +49 6221 549448 (431)
fax: +49 6221 549331
e-mail: heermann@surface.tphys.uni-heidelberg.de
www: http://philosoph.tphys.uni-heidelberg.de

1 Introduction

The motivation to parallelize the algorithms for the treatment of physics problems [1] by numerical simulations [2, 3, 4, 5, 6] is manifold. It ranges from being able to gather better statistics, to speeding up the application or to allow for real time visualization. There is strong motivation for parallelization to be able to simulate rather large systems, i.e. many particles, molecules, spins, basis for wave functions etc. Even though theories exist to exploit the finiteness of the system for some problems, it is often necessary to exclude finite size effects as much as possible.

I want to focus in these lectures, on the concepts to exploit parallelism in physics problems and on parallelism inherent in the numerical method's one applies to physics problems.

While a physics problem itself, in general, will be inherently parallel, we must understand parallelism or its absence resulting from the formulation of the problem or the method of solution. Two formulations of the same problem may result in a vastly different complexity regarding the amount of computations necessary and may be in their parallelization. Take the Ising model

$$\mathcal{H} = J \sum_{<i,j>} s_i s_j + h \sum_i s_i \tag{1}$$

$$Z = \sum_s e^{-\beta \mathcal{H}} \quad . \tag{2}$$

In this formulation we only have a local interaction among the spins. If we apply a conventional Monte Carlo method to study the Ising model in this formulation, we can exploit that the lattice naturally can be partitioned into independent sublattices concerning the updating.

The Ising model can also be studied using the formulation of a random cluster model [7] with the partition function

$$\mathcal{Z} = \sum_C B(\beta, C) 2^{n(C)} \quad . \tag{3}$$

In this formulation the clusters are independent but stretch over the entire lattice and we have lost the locality. This implies that the lattice does not partition into independent sublattices! The point I am trying to make is that the formulation of the model can play a central role whether one can parallelize a problem to a good degree. This does not necessarily imply that the better parallelized problem is more efficient. Efficiency here can mean for example how effectively one has reduced the correlation's among successively generated states.

The issue of parallelization of physics problems is even broader. We need to consider also the methods we are using to simulate a system to obtain observable quantities as well. Again let us use the Ising model to exemplify the point.

Suppose Poisson events arrive at a spin. Every spin carries an individual time. The Poisson event changes the spin to a new orientation. The spin can only change to a new orientation if its local time is less than the minimum of the time of the neighbour spins with which the spin is interacting. If this is not the case it must wait till it is in the past. This idea was beautifully worked out by Lubachevsky [8].

In this formulation the spins are independent and can be operated upon concurrently. We changed from the paradigm of time-driven simulations to event-driven simulations. The point that this is not necessarily a statement of efficiency must be made again. Also the efficiency in terms of wall clock time may not be better.

A further example for an algorithm [9] for the simulation of the Ising model will be given latter.

2 Concepts to Exploit Parallelism

In general we can say that there are several types of parallelism inherent in physics problems. These are:

- independence

- time correlated
- space correlated

As worked out in the introduction, the parallelism does not always come natural with the problem. Rather, one needs to consider the model, the method and the implementation. There are, however, concepts for parallelization that work independent of the actual problem. These are general concepts applicable to all kinds of science problems.

One can distinguish between the following general concepts

- poor mans parallelization
- data parallelization
- algorithmic parallelization
- domain decomposition

These make use of the independence relations that are naturally associated with any problem. The poor mans parallelization exploits that one often is interested in gathering a sample of independent runs. The data parallelization exploits the volume of the data and partitions the data workload among the processors. The algorithmic parallelization exploits the independence of the execution of parts of the algorithm and finally the domain decomposition partitions the space into independent subspaces.

3 Poor Mans Parallelization

The most often used parallelization today is presumably the so called poor mans parallelization. In the poor mans parallelization a given code is replicated as many times as there are processors (c.f. Figure 1). Each program on a processor executes independently from the other copies. No communication beside the input/output of results is necessary, that is, the programs or processes on different processors do not communicate with each other. Since there is no communication, there are no idle times for the processors (except for possible data communication to the outside world) and therefore the efficiency with which the processors are used is 100 percent. Of course, in terms of the computational cost, i.e., money per run, this concept is not so cost effective since we are not using the parallel computer with all its infrastructure (communication network, ...). Nevertheless, for the gathering of statistics in Monte Carlo or Molecular Dynamics problems, this concept is very well suited. Clearly this concept is applicable to almost any physics problem, provided the problem size is sufficiently small and the processor sufficiently powerful, to sustain the execution of the code and finish in reasonable time.

This type of parallelism is exploitable without almost any change to the codes. Existing code can be used.

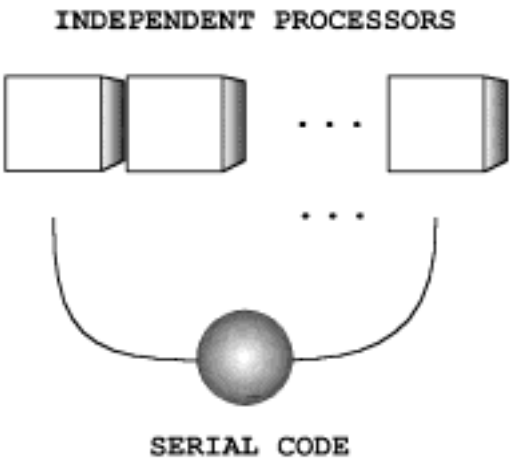

Figure 1: *In the poor mans parallelization a production code for a single processor machine is distributed in identical copies to the number of available or desired processors. All execute concurrently without communication among the copies.*

This concept is not new! Already during pre vector and parallel computer time one used the parallelism that natural come with the word length of the computer. For example, on an n bit machine, n spins of independent Ising systems can be stored and operated upon. Also during the vector period it was common practice to store many independent entities in one computer word.

3.1 Data Parallel Algorithms

Data parallel algorithms assign a fixed workload to a processor. The amount is in general evenly spread among the processors. An obvious application for this idea is a situation where the system cannot be split into spatial domains since the system changes in its spatial uniformity. In Figure 2 is shown a randomly triangulated surface [10]. If we were to split space into domains, then an uneven workload would result because the surface can change its state from collapsed to extended. Today there is no general algorithm available to handle this imbalance of the workload. Such a scheduler would need to shift "particles" or nodes in the present case from processor to processor. Such a situation is more efficiently handled by a user written code.

In general the application of this concept is good for situations where one cannot partition the system into small units with only nearest neighbour interaction. The typical situation is a system with long-range forces. Also from the implementation side this concept is advantageous since the coding work is less demanding and less complex than for the domain decomposition. Also it is less error prone and allows rapid code development.

Let us take the simulation of a simple system with atomic particles by molecular dynamics. The application of the concept of data parallelization implies that we divide the number of particles N evenly among the available p processors. This is done irrespective of the spatial position of the particles. Once a particle is assigned to a processor, it will continue to reside their (c.f. Figure 3). While there is no problem with

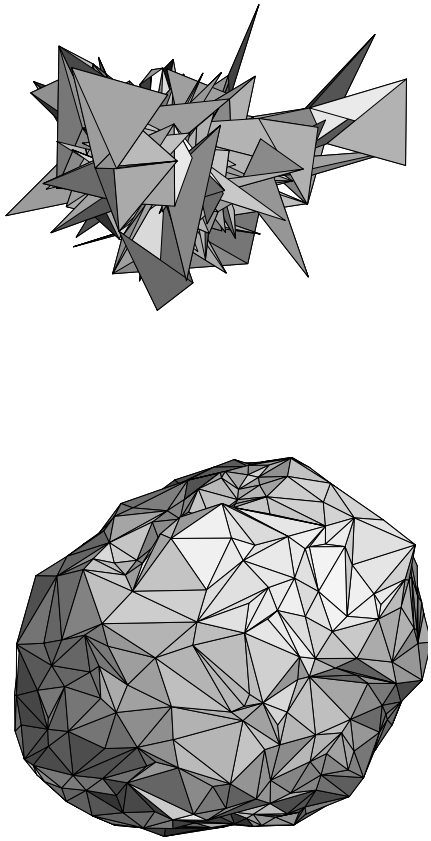

Figure 2: *Shown is a randomly triangulated surface at two different bending rigidities. These two figures show that if one uses a simple domain decomposition, problems with the load balancing can occur. In this situation data parallelism may be an alternative.*

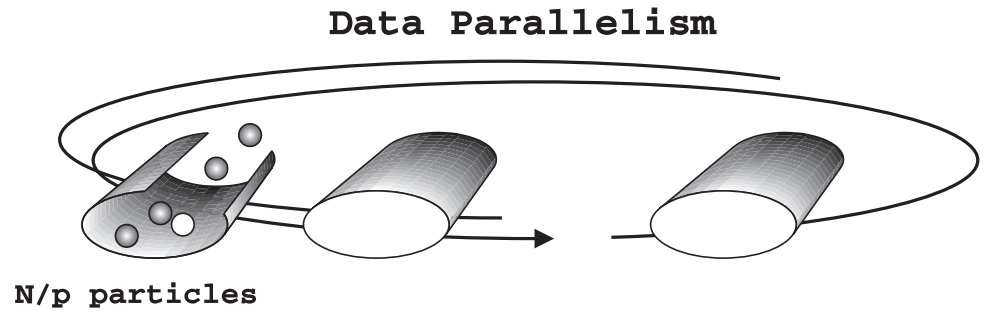

Figure 3: *Example of a data parallelization for the particles in a molecular dynamics simulation. The particles are assigned to processors irrespective of their spatial position. The particle are circled to allow the computation of the forces.*

the workload, there is now a much larger demand for communication. Every processor needs the information from all the other to gather the particle positions to calculate the forces acting on the particles residing or assigned to that processor. The particle positions need to be circled, implying that a ring communication network may used.

While this concept can be implemented in a fairly straight-forward way, it has some draw backs. Strong emphasis is put on the communication infrastructure. Also, it is difficult to implement further schemes to reduce the computational overhead, for example tables and the like.

3.2 Algorithmic Parallelism

In some sense, already the general computer is able to perform an algorithmic parallelization. Already in the early days of computer and processor design, one looked for parallelism in the computer codes. If you look at the design of RISC processors for example, you find that several instructions can be computed in parallel. The algorithmic parallelism also wants to exploit the parallelism that is inherent in the "code". Take for example the stages that one needs to go through in a molecular dynamics simulation. You can easily convince yourself that parts of the algorithm can be done simultaneously if you had components in the computer that can sustain such a parallelism. Most prominently this type of parallelism found application in special purpose computer.

Let us look at the molecular dynamics example. We do not want to go into the details of possible algorithms but pursue here the algorithm in a top-down approach.

```
------ Basic Molecular Algorithm ------
...
FOR n = 1 TO NumberOfMDSteps DO
  Compute the forces on the particles
  Integration for the positions
  Integration for the velocities
END FOR
```

On this level it is hard to see where we have parallelism so let's be more explicit.

```
------ Basic Molecular Algorithm ------
...
FOR n = 0 TO NumberOfMDSteps - 1 DO          // Loop 1
  Compute the forces on the particles
  FOR i = 0 TO N - 1                         // Loop 2
     Integration for the ith particle position
  END FOR
  FOR i = 0 TO N - 1                         // Loop 3
     Integration for the ith particle velocity
  END FOR
END FOR
```

Now we see that we can unroll the loops 2 and 3 onto the available number of processor's p. Every processor receives N/p loop cycles. This is very similar to the data parallelism that we discussed in the previous section. Automatic parallelizers try to pursue such a strategy that works best when the underlying hardware has a shared memory (see later).

Similarly, the N^2 loop implicit in the computation of the forces, if we do not provide measures to reduce the complexity, can be unrolled to the available processors reducing the computational complexity.

Along the same line, we can study the basic Monte Carlo algorithm. Here we want to use a lattice model.

```
------ Baisic Monte Carlo Algorithm ------
...
FOR n = 0 TO NumberOfMCSteps - 1 DO            // Loop 1
  FOR i = 0 TO N - 1                           // Loop 2
    FOR j = 0 TO N - 1                         // Loop 3
       Choose a lattice site
       Calculate the energy change
       Generate a random number r
       IF accept change THEN
         change site value
    END FOR
  END FOR
END FOR
```

Here unrolling the loop is dangerous. We must retain detailed balance. On this level, the generation of the random number can be done concurrently with the fetching of the site value and the computation of the change of energy. If we had, as possible for the Ising or similar models with only nearest neighbour interaction, split the lattice into two sublattices, we could unroll the loops and assign them to processors. Every site on a sublattice is independent and for example the innermost loop for one sublattice can be unrolled.

If you recall the Ising formulation from the introduction, we now also see that the reformulation of the basic method in terms of an event-driven simulation removes some of the obstacles to the parallelizabilitly.

On the hardware level machines were build to support concurrency. Examples are the Delft machine[11] (Delft Ising System Processor: DISP) that reflects in a direct way the structure of the Monte Carlo algorithm or the machine build by the Santa-Barbara group [12]. The Santa Barbara architecture allows to exploit the inherent parallelism of Monte Carlo Ising simulations that result from the data structure and the condition of detailed balance. Instead of using just one processor, one can include many more so that one can update spins in parallel.

The processor as such reflects, similar to the Delft computer, the structure of the Monte Carlo algorithm. The Santa Barbara machine optimizes the performance exploiting the data structure and the algorithmic structure.

Also for Molecular Dynamics the algorithmic parallelism is exploitable on the hardware level. Examples here include the IBM Almaden Research machine [13] or the implementation using of the algorithm on transputers put forward by the CCP5 group.

3.3 Domain Decomposition

In a wide sense, domain decomposition has been applied earlier than parallel computers became available. The multi-spin coding [14] method serves as an example. Different independent sublattices are coded into a single computer word and can be operated upon simultaneously. What we here want to follow and develop, is the concept of geometrically breaking up a system such that the parts can be operated upon, at least for some time, before there is a possible exchange of information.

The most straight forward application of this concept is to partition a lattice or a computational box for the molecular dynamics methods into subboxes. Every subbox holds, in general, a different number of particles in the case of a molecular dynamics computational box. This brings up immediately the question of load-balancing. If the number of particles is not equal on all processors, then some finish earlier than others and the overall utilization of the machine will drop. Here we do not want to go into discussions on this issue but it should be kept in mind.

For the application to lattice systems this issue does not play a role. I will later on also discuss very briefly the application of the idea of geometric or domain decomposition to lattice systems (see section 6).

Even though there is no real need to partition the computational box into equal subboxes we will for the sake of simplicity assume so. Graphically this is shown in Figure 4. Some applications with perhaps non-homogenous particle distribution may be better treated with a different kind of partitioning or perhaps adaptive partitioning.

We shall also assume that the communication between the processors is synchronous, that is, the sender cannot send the message and carry on with the execution of the code but has to wait till the receiver actually accepts the data. We shall also assume that the processor and language do not allow for a concurrent execution of sending and receiving.

In the section on the algorithmic parallelization I exposed the basic steps for a molecular dynamics simulation and I want to use this example here too. Let us focus

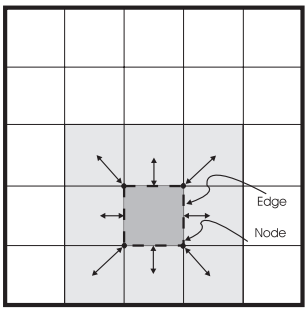

Figure 4: *Example of a domain decomposition for the square.*

on the force calculation, because the integration part can be done on each processor independently, once the forces acting on the particles in one subcell have been accumulated.

```
obtain node id myid
obtain neigbour node ids
...
do force calculation on particles in own box
exchange particles over edges
exchange particles over nodes
compile particle list
force calculation for interaction with neighbouring cells
update positions and velocities
```

With the notation introduced in the Figure 4 we can write down a pseudo-code for the communication part.

```
//      ---- COMMUNICATION VIA THE EDGES ----
// Odds send, even receives over the edges
FOR # edges DO
  IF myid is odd THEN
     SEND particles TO PID( edge )
  ELSE
     RECEIVE particles FROM PID( edge )
  END IF
END FOR

// Odds receives, even sends over the edges
FOR # edges DO
  IF myid is even THEN
     RECEIVE particles FROM PID( edge )
  ELSE
     SEND particles TO PID( edge )
  END IF
END FOR
```

```
//      ---- COMMUNICATION VIA THE NODES ----
// Odds send, even receives over the nodes
FOR # nodes DO
  IF myid is odd THEN
     SEND particles TO PID( node )
  ELSE
     RECEIVE particles FROM PID( node )
  END IF
END FOR

// Odds receives, even sends over the nodes
FOR # node DO
  IF myid is even THEN
     RECEIVE particles FROM PID( node )
  ELSE
     SEND particles TO PID( node )
  END IF
END FOR
```

Care with the periodic boundary condition must be taken. Clearly the example in Figure 4 does not work if we apply periodic boundary conditions. At the opposite end we must have opposite colored boxes.

4 Machine Considerations

Today one has three basics choices for the machine architecture:

- Shared-Memory Machine
- Distributed-Memory Machine
- Hybrid between shared and distributed

Shared-memory machine's can be often found among the workstations. Up to four or sixteen processors share a global memory as shown schematically in Figure 5. Shared-memory machines are difficult to handle from a technological point of view.

The memory is shared between the processors. Communication is done via the shared memory. Data is put into designated regions of the memory by a processor who wants to communicate a result to another processor. The receiving processor looks at the communication region and reads the data from there. This is fine as long as there is only one producer and one consumer. If there are many producers and many consumers, a synchronization mechanism must be established to avoid conflicts.

Today technology allows to handle a fair number of processors to be attached to a shared memory and the users need not to worry about the synchronization. It must be stressed, however, even though message passing is "easy", there is a constraint in that

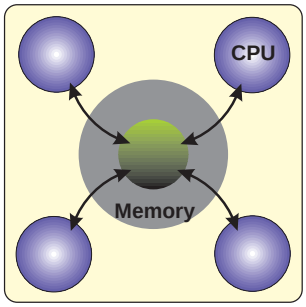

Figure 5: *Shared memory machine designs*

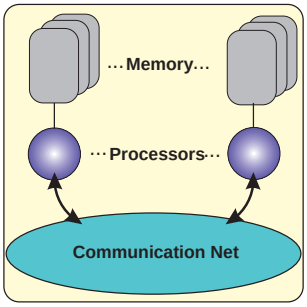

Figure 6: *Distributed memory machine design*

the number of processors that can be attached to one memory unit is much less than for machines with a distributed memory.

Distributed memory machines can be build with a huge number of processors. The only limit here is the communication bandwidth between the processors. Each processor has its own memory in this design (c.f. Figure 6) making it necessary to communicate date via an interconnection network to another processor. Here the issue is the kind of network and the bandwidth between to processor nodes.

In Figure 7 are shown some possibilities to connect the processors in a distributed memory design.

5 Language Considerations

Automatic parallelization can only be done on the most trivial level. Certain kind of "do" or "for" loops can be split and distributed among the processors. This is a parallelization not on the operations level as for the vectorization, but on the level of constructs. However, it must be stressed, that so far, and perhaps for some time to come, automatic parallelization will be not as advanced as the automatic vectorization. In any case, this can serve to parallelize algorithms designed for single processor machines. For multi-processor machines new algorithmic ideas need to be developed.

The most common languages that support in some sense the parallelization are FORTRAN 90 and C. There are various "dialects", for example

- C* [17]

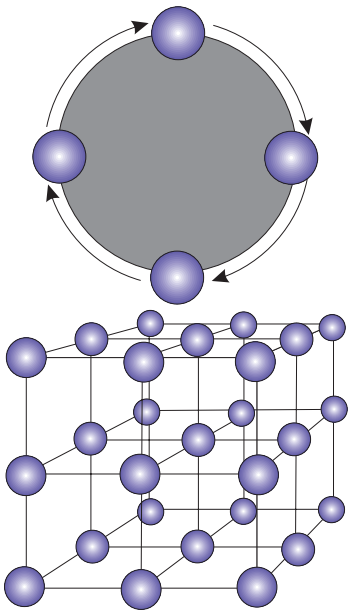

Figure 7: *Various topologies for the interprocessor connectivity*

- CC++ [18]
- Cp++ [19]
- FORTRAN M [20]

that specifically support parallel constructs. More common is the augmentation of a language with a set of routines to facilitate the data exchange between processes. There is quite a number of such libraries on the market. Lately, however, an initiative was successful to standardize the message passing. This has produced the MPI (Message Passing Interface) standard [21]. The standard has been adopted by all major computer and software vendors and there are public domain implementations available. Thus for reasons of software portability and software reengineering this is the software of choice for the years to come.

MPI incorporates two models of message passing

- Point-to-Point communication by direct message passing and
- global communication with primitive objects on distributed data

In passing we note that the popular PVM does not support both models of message passing!

6 A new algorithm for the simulation of large lattice systems

In the following, I describe a new algorithm [9] for simulating two–dimensional Ising lattices with local updates and discuss the parallelization. Unlike conventional algo-

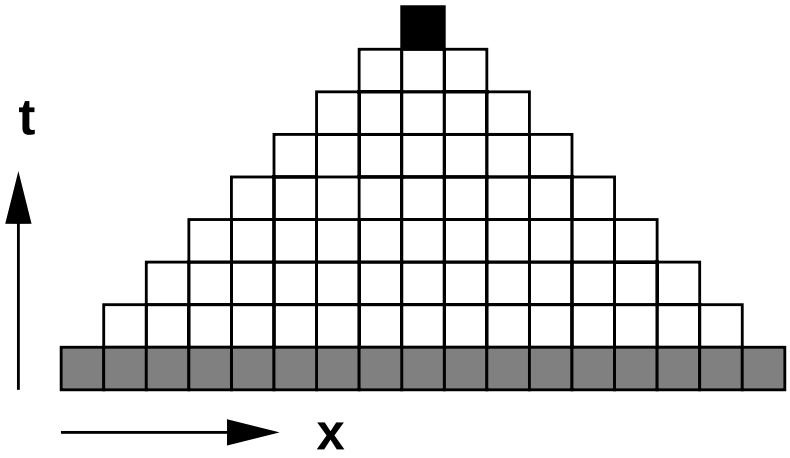

Figure 8: *The spin value of the last MC step (black) depends on a triangle of spins at earlier time steps. The spins at $t = 0$ (grey) can be drawn randomly according to a certain distribution (warm start) or can be fixed (cold start).*

rithms, memory requirements scale linear with only *one* system dimension and with the number of Monte Carlo time steps. With the algorithm it is was possible to simulate a 10^6x10^6 lattice on systems with only 128 MB main memory [22].

Because of the locality of the interactions in an Ising type lattice system, it is in principle possible to calculate the value of each spin at an arbitrary MC step *ab initio*, i.e. from initial conditions (see Figure 8).

The general Metropolis[15] update rule for calculating the new value of a spin s_{new} from its old value s_{old} and the values of its neighbour spins $\{s_{\mathrm{nn}}\}$ can be written as

$$s_{\mathrm{new}} = f(s_{\mathrm{old}}, \{s_{\mathrm{nn}}\}, r) \tag{4}$$

where r is a (pseudo) random number

$$f(s, \{s_{\mathrm{nn}}\}, r) = \begin{cases} -s & \text{for } r < \exp\left(-2\beta s \sum\{s_{\mathrm{nn}}\}\right) \\ s & \text{else} \end{cases}$$

The four next neighbours of spin $s(x, y)$ in a rectangular lattice are, of course,

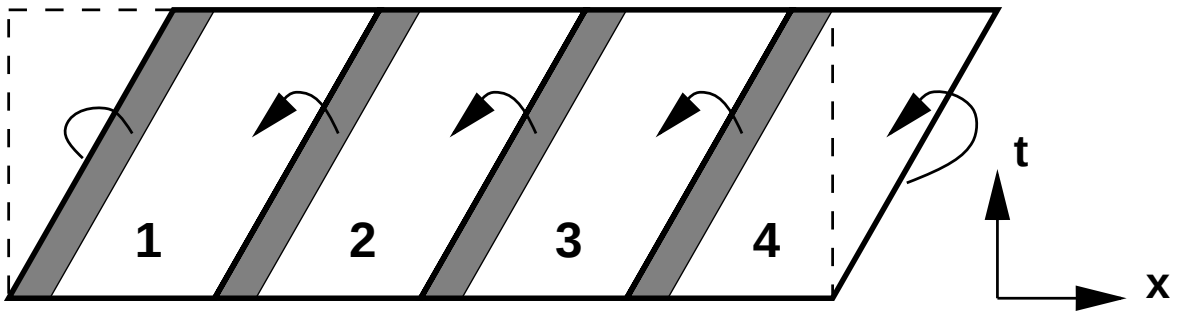

Figure 9: *Parallelization of the lattice using domain decomposition. The shaded borders are sent to the left neighbour processor in a round-robin fashion after the step 3 of the initial setup.*

$$\{s_{\mathrm{nn}}(x,y)\} = \{s(x+1,y), s(x-1,y), s(x,y-1), s(x,y+1)\}$$

If one is only interested in local observables like magnetization and energy, it is therefore possible to calculate spins at different MC steps simultaneously and avoid storing the whole lattice. Instead of iterating the lattice through the MC update steps, one can iterate spin values at different MC steps through the lattice. For a fixed number of MC steps that is smaller than the linear dimension of the system (as in a study of relaxation into equilibrium, where only the first update steps are of interest) this may lead to less memory requirement than storing the full lattice at one time step.

Although the memory constraints for simulating very large lattices have been released by the algorithm described above, computing time of course still scales linear with the system size and the number of Monte Carlo steps. A natural approach to decrease computation time is to divide the computational work on to n processors, for instance, on a cluster of workstations or a parallel computer. This can be done by dividing the lattice into sub-lattices of equal size (domain decomposition). It turns out that the problem here is suited very well for parallelization. However, unlike in conventional domain-decomposition schemes, where the rectangular lattice is divided along one or more dimensions into rectangular sublattices (cf. [16]), in this case a division of the (x,y,t)-lattice into parallelogram-like sublattices along the x-axis turns out to be most efficient. (see Figure 9)

Because the lattice is calculated for all time steps simultaneously, communication of the (y,t) lattice border among neighbour processors needs to take place only *once* after the setting up of the roof structure. Additional communication is needed only for the global sums of the observables making the parallelization highly efficient even for moderate problem sizes.

7 A Case Study

It seems quite obvious that one can develop a parallel algorithm for the simulation of polymer systems. What I want to present in these notes is one solution for such a simulation algorithm.

For polymer simulations it is necessary to have good computational algorithms [23]. Such simulations tend to be very time consuming. This is first off all due to the long relaxation times. Before a single polymer has reached a new conformation that is statistically different, one has to generate in the worst case of the order of $O(N^3)$ move. Here N is the length of the polymer that we regard as a string with N beads.

The model I want to use, is derived from a corarse-graining approach. In the coarse-grained approach, the detailed chemistry only enters in the derivation of the potential between new interacting units. These are substitutes for the original detailed chemistry. The system is considered on a larger length and longer time-scale. Loosely speaking, the fast degrees of freedom, like the bond vibrations have been eliminated and the short spatial length scales upscaled.

In this model the basic building block is an ellipsoid (c.f. Figure 10). Using this building block chains one can construct, may they be linear chains, chains with side chains or branched. Because the ellipsoid can degenerate into a sphere, the model is able to accommodate and model even complex monomer units. An example is shown in Figure 11 for the modeling of the HIP-PC with additional spheres attached to the ellipsoids to model the side group [24].

7.1 The Hybrid Monte Carlo Method

In conventional Monte Carlo (MC) calculations of condensed matter systems, such as an N-particle system with a Hamiltonian $\mathcal{H} = \mathcal{U}$, only local moves (displacement of a single particle) are made [2, 3, 5]. Updating more than one particle typically results in a prohibitively low average acceptance probability $\langle P_A \rangle$. This implies large relaxation times and high autocorrelations especially for macromolecular systems. In a Molecular Dynamics (MD) simulation, with $\mathcal{H} = \mathcal{T} + \mathcal{U}$, on the other hand, global moves are made. The MD scheme, however, is prone to errors and instabilities due to the finite step size in time. In order to introduce temperature in the microcanonical context, isokinetic MD schemes are often used [5]. However, they do not yield the canonical probability distribution, unlike Monte Carlo calculations.

The Hybrid Monte Carlo (HMC) method [25, 26, 27] combines the advantages of Molecular Dynamics and Monte Carlo methods: it allows for global moves (which

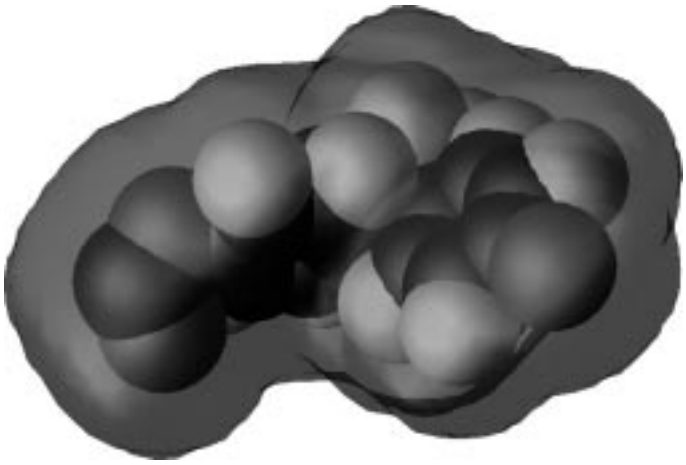

Figure 10: *The general ellipsoidal model for polymer systems*

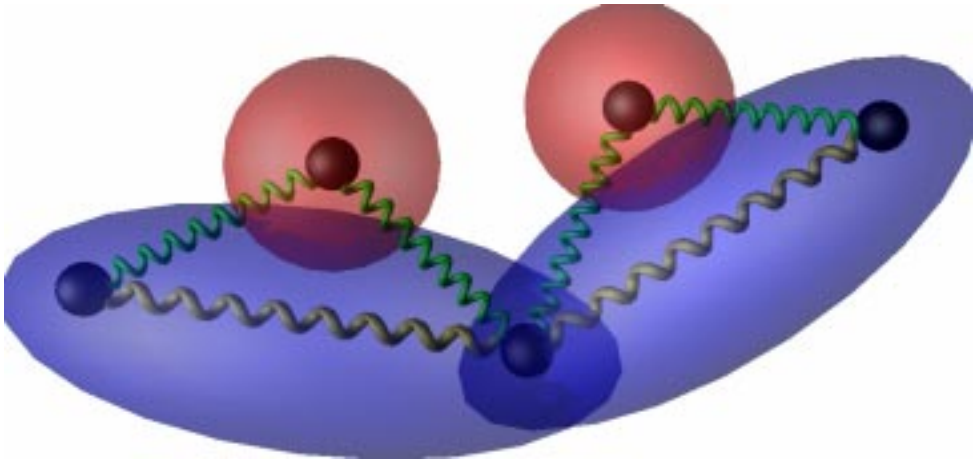

Figure 11: *Schematic representation of the modelling of HIP-PC within the ellipsoidal model.*

essentially consist in integrating the system through *phase* space); HMC is an exact method, i.e., the ensemble averages do not depend on the step size chosen; algorithms derived from the method do not suffer from numerical instabilities due to finite step size as MD algorithms do; and temperature is incorporated in the correct statistical mechanical sense.

The application of the Hybrid Monte Carlo method has been proposed [26] for condensed-matter systems and investigated for atomic fluids. In the present contribution the method will be described briefly and applied to macromolecular systems.

In a HMC scheme global moves can be made while keeping the average acceptance probability $\langle P_A \rangle$ for a move high. This can be achieved as follows. One global move in *configuration* space consists in integrating the system through *phase* space for a fixed time t using some discretization scheme $g^{\delta t}$ (δt denotes the step size)

$$\begin{aligned} g^{\delta t}: \quad \mathbb{R}^{6N} &\longrightarrow \mathbb{R}^{6N} \\ (x,p) &\longrightarrow g^{\delta t}(x,p) =: (x',p') \end{aligned}$$

of Hamilton's equations

$$\begin{aligned} \frac{dx}{dt} &= \partial_p \mathcal{H} \\ \frac{dp}{dt} &= -\partial_x \mathcal{H}. \end{aligned} \tag{5}$$

At the beginning of each global Monte Carlo step the initial momenta are drawn from a Gaussian distribution at inverse temperature β:

$$p_{Gaussian}(p) \propto e^{-\beta \mathcal{T}}, \tag{6}$$

and it can be shown [26] that the acceptance probability is then

$$P_A((x,p) \to g^{\delta t}(x,p)) = \min\{1, e^{-\beta \delta \mathcal{H}}\}, \tag{7}$$

where

$$\delta \mathcal{H} = \mathcal{H}(g^{\delta t}(x,p)) - \mathcal{H}(x,p)$$

is the discretization error associated with the discretization scheme $g^{\delta t}$. Provided the discretization scheme is *time-reversible* and *area-preserving* detailed balance is satisfied [26]. Thus the HMC algorithm generates a Markov chain with a Boltzmann distribution as the stationary probability distribution. The probability distribution is entirely determined by the detailed balance condition, therefore neither the distribution nor any ensemble averages depend on the step size δt chosen.

However, the average acceptance probability $\langle P_A \rangle$, because of (7), depends on the average discretization error $\langle \delta \mathcal{H} \rangle$ and hence does depend on δt. Increasing the step size will result in a lower average acceptance probability $\langle P_A \rangle$. Varying δt, the average acceptance probability $\langle P_A \rangle$ can thus be adjusted to minimize autocorrelations of observables.

We have proven a more general result which can be exploited in the simulations. The Metropolis transition probability is really composed of two probabilities [25]:

$$p_M((x,p) \to (x',p')) = P_C \, P_A$$

i.e.,. a propositional probability for a configuration and one for the acceptance. As long as we accept the configuration with the Hamiltonian $\mathcal{H}$ we can use any other Hamiltonian $\mathcal{H}'$ to derive equations of motion for the proposition. Specifically, we can use $\mathcal{H}' = \alpha \mathcal{H}$. This choice is motivated by the observation [28] that the effect of the discretization of the original Hamiltonian on the equations of motion, is to renormalize the original Hamiltonian. Thus, instead of the original Hamiltonian a new scaled one is solved *exactly*. This observation can be used to scale out to a certain degree the effect of the discretization error on the acceptance rate and accelerate the algorithm.

With respect to the parallelization the algorithm has the definite disadvantage that a global sum is need to decide whether the computed configuration be accepted or rejected. However, since in the applications that are envisaged for this method (see below) this is only a tiny fraction of the actual computing cost and be neglected.

7.2 Parallelization of the Polymer System

The Molecular Dynamics part for the above outlined method is fairly general and can be considered on general grounds, except that we are dealing we long chain molecules. If we were to consider only atoms, then we could apply the domain decomposition concept. For polymer systems, if we are to keep a single chain to be the "unit" then the computational box can not be partitioned into many subvolumes. The chain may stretch over many such subvolumes. From this point of view it is not advantageous to use a data structure and integration algorithms that work on the basis of the connectivity knowledge. Rather we should have as the basic data structure one for the atom. Then we could apply the ideas developed for the domain decomposition also to the polymer system. The price here to paid is a higher administrational overhead. The algorithm can not be based on the connectivity and every "atomic data structure" must carry information on the chain connectivity. What is passed around is this data structure with the accumulated results for the force on that particular "atom". This also implies that there must be some instance that ensures that after a certain number of steps all information was gathered to guarantee a position update in terms of an integration step.

This approach also facilitates the application of the program to many different types of polymer systems. Since there is no reference to the chain connectivity with the program (all information of this nature is hidden to the program and build into the

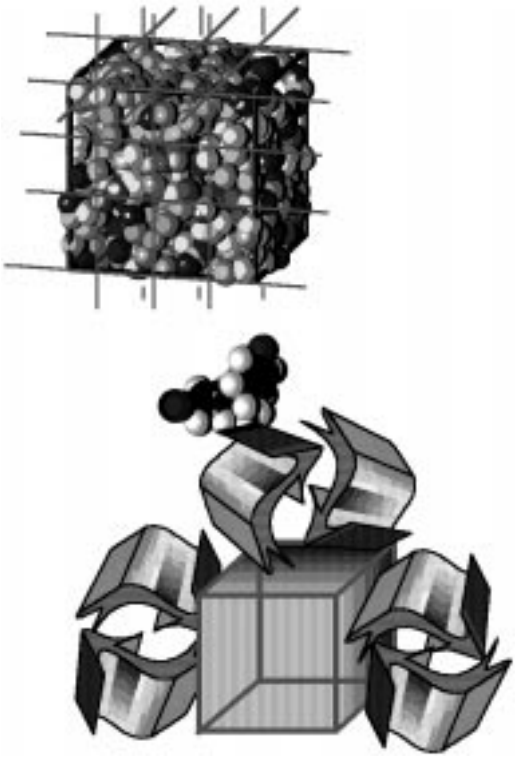

Figure 12: *Shown is a schematic picture for the parallelization of the polymer system. In this implementation not polymers are the units that are passed but atoms.*

data structure) one is able to simulate systems ranging from simple linear chains to branched or even networks.

Acknowledgement This work was partially funded by a grant from the BMFT and EU. I would like to thank A. Linke, K. Zimmer, P. Altevogt and A.N. Burkitt for the stimulating discussion and the work they have putted into our joint efforts.

References

[1] D.W. Heermann and A.N. Burkitt, **Parallel Algorithms of Computational Science Problems** Springer Verlag, Heidelberg 1990

[2] K. Binder and D.W. Heermann, **The Monte Carlo Method in Statistical Physics: An Introduction** Springer Verlag, Heidelberg 1988

[3] D.W. Heermann, **Computer Simulation Methods in Theoretical Physics**, Springer Verlag, Heidelberg 1986

[4] Hoover, **Molecular Dynamics**, Springer Verlag, Heidelberg 1985

[5] M.P. Allen and D.J. Tildesley, **Computer Simulation of Liquids**, Clarendon, Oxford, 1987

[6] M.H. Kalos, **Monte Carlo Methods** Wiley and Sons, New York, 1986

[7] C.M. Fortuin and P.W. Kastelyn *Physica* **50**, 297 (1972)

[8] B.D. Lubachevsky, *Journal of Comput. Phys* **75**, 103 (1988)

[9] A. Linke, D.W. Heermann and P. Altevogt, *Commput. Phys. Commun.* in press

[10] Ch. Münkel and D.W. Heermann, *Phys. Rev. Lett. 1995*

[11] A.Hoogland, J. Spaa, B. Selman, and A. Compagner, *J. Comput. Phys.* **51**, 250 (1983)

[12] R.Pearson, J.L. Richardson, and D. Toussaint, *J. Comput. Phys.* **51**, 241 (1983)

[13]
- D.J. Auerbach, A.F. Bakker, T.C. Chen, A.A. Munshi, W.J. Paul, *Mat. Res. Soc. Symp. Proc.* **63**, 219 (1985)
- D.J. Auerbach, W. Paul, A.F. Bakker, C. Lutz, W.E. Rudge, and F.F. Abraham, *J. Phys. Chem.* **91**, 4881-4890 (1987)

[14] L. Jacobs,C. Rebbi, *J. Comp. Phys.* **41** 203 (1981)

[15] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, *J. Chem. Phys.*, **22** 881 (1954)

[16] P. Altevogt, A. Linke, *Parall. Comp.* **19** 1041 (1993)

[17] C* *C* Thinking Machines Co. Manual*

[18] CC++ *http://www.compbio.caltech.edu/ccpp/hpcpp.html*

[19] Cp++ http://www.cs.uoregon.edu/ bhelm

[20] Fortran M http://www.mcs.anl.gov/fortran-m

[21] The report on the standard can be optain via anonymous ftp from the following sites

- netlib2.cs.utk.edu/mpi/mpi report.ps
- aurora.cs.msstate.edu/pub/mpi/mpi report.ps.Z
- info.mcs.anl.gov/pub/mpi/mpi report.ps.Z
- bag.osc.edu/pub/lam/mpi report.ps.Z

[22] A. Linke, D.W. Heermann and P. Altevogt, *Physica A* submitted

[23] K. Kremer and K. Binder, Phys. Rep. xxx

[24] K. Zimmer, A. Linke and D.W. Heermann, *in preparation*

[25] S. Duane, A. D. Kennedy, B. J. Pendleton and D. Roweth, Phys. Lett. B **195**, 216 (1987)

[26] B. Mehlig, D. W. Heermann and B. M. Forrest, Phys. Rev. B, **45**, 679 (1992)

[27] B. Mehlig, D. W. Heermann and B. M. Forrest, Mol. Phys. **76**, 1347 (1992)

[28] G.G. Batrouni, G.R. Katz, A.S. Kornfeld, G.P. Lepage, B. Svetitsky, K.G. Wilson, Phys. Rev. D **32**, 2735 (1985)

[29] M. Creutz and A. Goksch, Phys. Rev. Lett. **63**, 9 (1989)

[30] D. Rigby and R.J. Roe, J. Chem. Phys. **87**, 7285 (1987)

[31] A. Greiner, W. Strittmatter, and J. Honerkamp, J. Statis. Phys. **51**, 85 (1988)

[32] J. Baschnagel, K. Qiun, K.Binder, and W. Paul, Macromolecules, **25**, 3117 (1992)

[33] R. Gupta, G.W. Kilcup, and S.R. Sharpe, Phys. Rev. D, **38**, 1278 (1988)

[34] G.S. Grest and K. Kremer, Phys. Rev. A, **33**, 3628 (1986)

[35] W. Paul, K. Binder, D.W. Heermann, and K. Kremer, J. Chem. Phys. **95**, 7726 (1991)

[36] A. L. Rodríguez, H.–P. Wittmann, and K. Binder, Macromolecules, **23**, 4327 (1990)