

Simulated Annealing : Comparison of Vector and Parallel Implementations

Final report on the NCF project : 'Crystallization on a Sphere'

CRG.91.08

Technical report: CS-06-93

Authors : P.M.A. Sloot, J.M. Voogd, D. de Kanter and L.O. Hertzberger

University of Amsterdam

Parallel Scientific Computing and Simulation group

Faculty of Mathematics and Computer Science

Kruislaan 403

1098 SJ Amsterdam

Contents

| | page | |
|---|---|----|
| 1 | Introduction | 3 |
| | 1.1 Aim of the research | 3 |
| | 1.2 Background | 3 |
| 2 | Functional Aspects of the S.A. procedure | 5 |
| | 2.1 The sequential version | 5 |
| | 2.2 The vector version | 6 |
| | 2.3 The parallel version | 6 |
| 3 | Implementation Aspects | 8 |
| | 3.1 The vector version | 8 |
| | 3.1.1 Short description of vector processing | 8 |
| | 3.1.2 Description of the implementation | 9 |
| | 3.2 The parallel version | 12 |
| 4 | Time complexities | 14 |
| | 4.1 The sequential version | 14 |
| | 4.2 The vector version | 15 |
| | 4.2.1 Time complexity of the perturbation step | 16 |
| | 4.2.2 Time complexity of the potential energy update | 16 |
| | 4.2.3 Time complexity of the radius update | 17 |
| | 4.2.4 Total timecomplexity of the vector implementation | 17 |
| | 4.3 The parallel version | 18 |
| | 4.3.1 Functional decomposition | 18 |
| | 4.3.2 Systolic and hybrid S.A. | 19 |
| 5 | Results | 20 |
| | 5.1 The vector version | 20 |
| | 5.2 The parallel version | 24 |
| | 5.2.1 The tree decomposition | 24 |
| | 5.2.2 The systolic simulated annealing algorithm | 26 |
| | 5.3 Comparison of the vector and parallel implementations | 29 |
| 6 | Results of the crystallization experiments | 31 |
| 7 | Conclusions | 34 |
| | 7.1 Comparison of the vector and parallel implementations | 34 |
| | 7.2 Crystallization experiments | 34 |
| 8 | Future research | 35 |
| | Acknowledgements | 36 |
| | Reference list | 36 |

1 Introduction

Computer simulations are considered as an important research tool in many areas of science. With this widespread use comes an ever increasing demand for computing power. Often a single simulation can take many hours of CPU time on modern computers, indicating the need for fast and efficient algorithms.

The application that we are working on is a simulation of crystallization with spherical boundary conditions. This is implemented with a simulated annealing (S.A.) algorithm. Since this is a problem that requires an enormous amount of computing power even for modest problem sizes, we started looking for methods to speed up the simulations.

In this report we compare a vector version of S.A. on a supercomputer with a parallel implementation on a transputer platform. After this comparison we give some of the results of the crystallization experiments that we have obtained with the implementations discussed in this work.

1.1 Aim of the research

The goal of the research presented here is to make a comparison of the performance of the Simulated Annealing algorithm on a parallel computer and on a vector computer. We have investigated the scalability of the parallel implementation with the number of particles (N) and the number of processors (P) and compared it with the scalability of the vector implementation with the number of particles. The parallel version is implemented on a Parsytec CGel with 512 nodes and the vector version on one node of the CRAY Y-MP 4/464.

1.2 Background

Particle dynamics simulations with spherical boundary conditions for large numbers (10^3 - 10^6) of particles (e.g. molecules) with Lennard-Jones or similar interactions at high density, provide an important testing ground for the study of closed 2D systems. Spherical boundary conditions have been used as an alternative to periodic boundary conditions to approximate bulk systems. Although the spherical topology of the boundary conditions has only a limited effect on the properties of bulk systems, they do affect the properties of the crystalline state in an essential way (for example by inducing global symmetry). Particularly the study of hierarchical clustering and the ordering of defects in a spherical matrix is a challenging 'close packing' problem.

The hypothesis has been put forward [1] that in spherical bilayer vesicles (spontaneously formed from fragmented biomembrane material), hierarchically ordered arrangements of the lipid matrix may be induced in the most densely packed ('backbone') shell of the vesicles. Such patterns result from highly non-linear co-operative effects. Like in the case of viruses these patterns may be very specific, reflecting the optimization in enthalpy and entropy in the self-assembly process. The study of spherical crystallization can test the hypothesis on

hierarchical regular polyhedral arrangements. In particular, it may help to understand the quite remarkable observation of 'quantum' jumps in the size of biomembrane vesicles [2].

To study these types of phenomena we have to address the long standing problem of equilibrium arrangements of particles on a sphere under the influence of two-body forces. In the case of Coulomb interactions, the problem resembles the mathematical exercise of spreading a given number of points equally spaced over a spherical surface. This equivalence relates the physics of the problem to a geometrical principle. It may be understood by considering the geodesic connecting lines between N points, giving a spherical polyhedral net with N vertices. Below $N = 20$ one has perfectly regular polyhedral nets corresponding to the Platonic solids with 4, 6, 8, 12 and 20 vertices. Beyond $N = 20$ no perfectly regular polyhedral net have been observed.

Many problems originating from physics, chemistry and mathematics, like the experiments that we are working on can be formulated as optimization problems. A vast majority of these problems involve the determination of the absolute minimum of a underlying multidimensional function. Usually optimization of these complex systems is far from trivial since the solution must be attained from a very large irregular candidate space, containing many local extrema. As a consequence the computational effort required for an exact solution grows more rapidly than a polynomial function of the problem size, the problem is said to be NP (non-polynomial time) complete. Because it is impossible to examine all solution candidates, even in principle, approximation methods are required.

A well established computational scheme is the Simulated Annealing (S.A.) method, a stochastic optimization procedure that mimics the essentials of physical annealing. In physical annealing a material is heated to a high temperature and is then allowed to cool slowly. At high temperature the molecules move freely with respect to one another. If the liquid is cooled slowly, thermal mobility is lost. The molecules search for the lowest energy consistent with the physical constraints. Normally the S.A. method is applied to combinatorial optimization with discrete steps. However, in our research we clearly need a continuous algorithm. The perturbation steps in the S.A. algorithm are adapted for our application and are discussed in the section on the functional aspects of the sequential version.

Although S.A. guarantees the finding of the global minimum, the time required for the algorithm to converge increases rapidly with increasing number of particles and/or local minima. In the biophysical problem we are dealing with, the number of particles to investigate is typically larger than 10^5 and the number of local minima, all stable particle configurations, is also very large although there is no expression for the amount of minima. Especially with the Lennard-Jones potential there may be a large number of local minima corresponding to crystal patterns with a non optimal radius, see the results of the crystallization experiments in section 6. Therefore conventional annealing implementations do not apply, and more efficient methods need to be investigated. A standard method to lower the computational time is to use vector super computers. With the new breed of parallel machines and programming paradigms, other very fast implementations come within reach.

2 Functional Aspects of the S.A. procedure

2.1 The sequential version

The S.A. algorithm for solving combinatorial optimization problems was formulated in 1983 by Kirkpatrick et al.[3]. It is based on a method developed by Metropolis et al.[4] to study the equilibrium properties of very large systems of interacting particles at finite temperature. In terms of the crystallization problem at hand the procedure works as follows. First N particles are randomly placed on a virtual supporting sphere. The annealing begins by creating a Markov chain, of given length, at a certain temperature. The Markov chain grows by randomly displacing particles and calculating the corresponding change in energy of the system and deciding on acceptance of the displacement.

Calculation of potential energies in particle systems is the most time consuming part. There are several methods of speeding up these kind of calculations. But in our application we have to bear in mind that our target systems consist of several thousand particles. This makes some of these methods very demanding on memory. For example in order to find the energy difference after a particle move we have to calculate the energy before and after the move. If a matrix would be used with the potential energies of all interactions, we would not need to calculate the potential energy before the move since that energy can be derived by summing the potential energies in the array for that particle. When the new situation is calculated the array must be updated. For the system sizes that we are thinking of the memory demand is much too high. Another method of speeding up is the usage of a interaction list where all the interactions in the system are listed. The interaction list method can only be used for short range interactions whereas we want to leave the door open to all kinds of interactions, for example a Coulomb potential. In the section on future research an other method is described, namely the hierarchical tree algorithms and the fast multipole method.

The perturbations in our application involve continuous steps. For our application we have developed the following method to use the S.A. algorithm on continuous spaces. The displacement of a particle is constructed by generating a random distance and a random direction. The distance is drawn from a Cauchy distribution of a certain width. The Cauchy distribution is used because of the relatively strong tail. This tail makes de-trapping from local minima possible by allowing large deviations to be generated. The width of the Cauchy distribution is adaptively changed so that about half of the particle moves are accepted. The direction in which the particle is moved is a random angle between 0 and 2π . Naturally all displacements have to leave the particle on the sphere.

Moves result in an energy change ΔE , where ΔE is the energy of the new situation minus the energy of the current situation. The moves are accepted with probability $P(\Delta E, T)$ at temperature T according to the following scheme :

$$\begin{aligned} P(\Delta E, T) &= \exp(-\Delta E / T) & \text{if } \Delta E > 0 \\ P(\Delta E, T) &= 1 & \text{if } \Delta E < 0 \end{aligned} \tag{1}$$

Unaccepted moves are undone. This choice of $P(\Delta E, T)$ guarantees that the system evolves into the Boltzmann distribution [4].

After a certain number of steps the radius is perturbed. This is done by calculating the energy of the system at a randomly generated new radius and subtracting the current energy. Acceptance is also decided according to the probability scheme given above.

After a chain has ended the temperature is lowered by multiplying the temperature with the cool-rate, which is a number slightly less than 1 (typically 0.9) after which a new chain is started. This process continues until a stop criterion is met. The stop criterion in our implementation is met when the standard deviation in the final energies of the last ten chains falls below a certain value (typically 10^{-6}). The energy of the system is defined as the energy per particle. This removes the dependency of the energy on the number of particles such that the stop criterion can be fixed.

2.2 The vector version

In addition to the faster clock speed of the target machine, the CRAY Y-MP 4/464, the capability of vector processing is used. However since we aim at a comparison of the vector implementation versus the parallel implementation we have not exploited the parallelization provided by the 4 processors. The algorithm for the vector version is in large respect the same as the algorithm for the sequential version described above. Some changes in the sequential code are made to improve the processing. These changes are described in the section about the implementation aspects. The vectorization is carried out by the highly optimized CRAY Y-MP C-compiler.

2.3 The parallel version

The parallel version actually consists of two kinds of decompositions, the first is a decomposition of the Markov chains (systolic), the second is a functional decomposition of the energy calculation in each step in the Markov chain.

A synchronous algorithm that does not mimic sequential annealing is systolic S.A. [5]. In systolic S.A. a Markov chain is assigned to each of the available processors. All chains have equal length. The chains are executed in parallel and during execution information is transferred from a given chain to its successor. Each Markov chain is divided into a number of subchains equal to the number of available processors. The execution of chain $k+1$ is started as soon as the first subchain of chain k is generated. Equilibrium is not yet established by then. Quasi-equilibrium of the system is preserved by adopting intermediate results of previous Markov chains.

Let P be the number of processors, L the length of the Markov chains, $SL = (L/P)$ the length of the subchains, T_k the temperature in Markov chain M_k , and $X_{k,m,i}$ the i -th configuration vector of subchain $M_{k,m}$. Figure 1 shows a diagram of a 3 processor implementation.

The first configuration of a new subchain, $X_{k,m,1}$, $m > 1$, is either the last configuration of the previous subchain, $X_{k,m-1,SL}$, or the final configuration of the last generated subchain of M_{k-1} , $X_{k-1,m,SL}$. The original algorithm based its choice on the probability of occurrence of these configurations according to the Boltzmann distribution. Since this distribution is often unknown, Kim & Kim [6] suggested $X_{k-1,m,SL}$ to be a newly generated configuration and

accept it with acceptance probability 1 if $\Delta C \leq 0$ and $\exp(-\Delta C/T)$ for $\Delta C > 0$ (with ΔC equal to $C_{k-1,m,SL} - C_{k,m-1,SL}$).

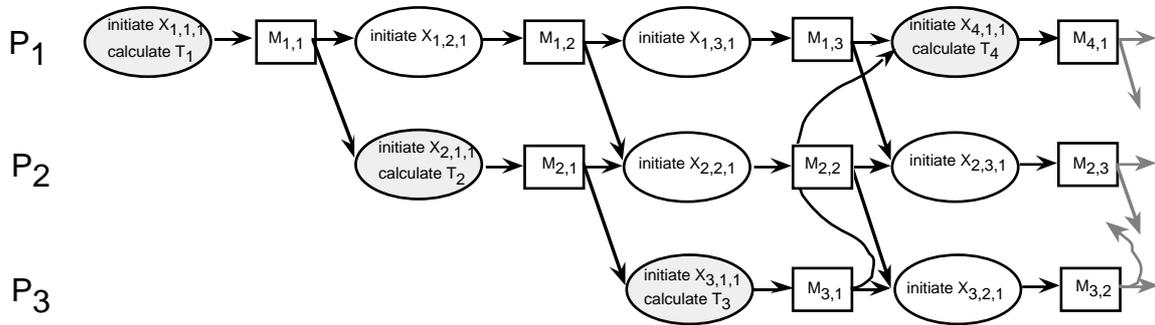


Figure 1) Diagram of a 3 processor implementation of the overlap algorithm.

The overlap enhancement consists of transferring the final configuration of M_{k-P} to M_k . M_k chooses probabilistically between $X_{k-P,P,SL}$ and $X_{k-1,1,SL}$. The enhancement is explicit in Figure°1. We have used a fixed cooling schedule based on the cool-rate. This parallelization method introduces a functional difference to the originally sequential scheme and will have consequences for the accuracy of the iterative process (see section 5.2.2).

In addition to this parallel algorithm we have exploited the parallelism that can be obtained in an ordinary Monte Carlo algorithm. Here the most time consuming part of the program is the calculation of the energy difference resulting from the perturbations. Since these calculations are independent, we parallelize this part of the program by functional decomposition [7].

3 Implementation Aspects

All calculations of all implementations are performed with double precision variables (64°bits).

All distances are scaled with a factor $2^{1/6}\sigma$, where σ is the distance parameter of the Lennard-Jones potential. In digits this means that our cut-off radius is 2.245σ . The Lennard-Jones potential goes through zero at $r = \sigma$, where r is the distance between two particles.

3.1 The vector version

3.1.1 Short description of vector processing

The CRAY Y-MP is a vector processor. This means that it possesses special hardware to process series of numbers. The concept of 'pipelining' is used to make this possible. In pipelining the instruction set given to a functional unit is split into components such that instructions are 'piped' one after each other through the functional unit. An example of a component of an arithmetic instruction, for example a multiplication, is the comparison of the binary exponents of a pair of numbers. Now pipelining means that if one instruction component of the multiplication is finished for a pair of numbers the result is passed to the instruction component which has to be done next. After this the instruction component that has received the results continues the processing while at the same time the first instruction component processes the next pair of numbers. This means that if the component pipeline is filled, one floating point operation can be executed each clock cycle.

Another important concept in vector processing is the concept of 'vector instruction' which means that one instruction is issued to a functional unit, for example a floating-point add, for a series of operands. The functional unit will generate the results for each operand at the rate of typically one result per clock cycle.

The performance of a program on a vector computer depends on many elements of the hardware design and how well these are used by the compiler. Among the hardware elements are: The size and number of vector registers, number of concurrent paths to memory, instruction issue rate and number of duplicate functional units (multiple 'vector pipelines'). The execution time of any vector instruction consists of the set-up time and a processing time that is proportional to the vector length (number of elements of the vector). The time involved to route the operands to the functional units is the set-up time (τ_{setup}).

The memory of the CRAY Y-MP is organised in 128 memory banks. The locations of subsequent elements are in a different memory bank. Reading and writing from the memory banks costs the system 5 clock cycles. However if the elements of a vector are read from memory one after another, it means that no extra time is involved in addressing the memory. For more details see [8].

3.1.2 Description of the implementation

The parts of the algorithm best suited for vector processing are the calculations of the energy of one particle and the calculation of the energy of the total system. It turns out that the part in which the calculation of the energy of one particle is done is the most time consuming part of the sequential algorithm. Therefore vector processing of these parts promise a significant speedup of the total program with respect to the sequential version. In this section the implementation of the parts that are vector processed is described in more detail. The pseudo code of the implementation of the one particle energy calculation with a Lennard-Jones interaction with a cut-off distance is shown in Diagram 1. This loop shall be referred to in the sequel as the 'one particle energy loop'. We looked for possibilities to improve the performance of the loop. One arithmetic floating point divide operation is saved by rewriting the function of the Lennard-Jones potential as follows:

$$V(r) = \frac{1}{r^{12}} - \frac{2}{r^6} = 2 \frac{0.5 - r^6}{r^{12}} \quad (2)$$

where r is the distance between two particles. The multiplication by 2 is performed outside the loop.

```
Loop over i from 1 to N-1
  distance_x [i] = x_coord [i] - perturbed_x_coord [i]
  distance_y [i] = y_coord [i] - perturbed_y_coord [i]
  distance_z [i] = z_coord [i] - perturbed_z_coord [i]
  distance [i] = distance_x[i] * distance_x[i] + distance_y [i] * distance_y[i] +
  distance_z[i] * distance_z[i]
  Condition : distance[i] < cut-off
    distance [i] = distance [i] * distance [i] * distance [i]
    energy = energy + (0.5 - distance[i]) / (distance[i] * distance°[i])
  End Condition
End loop
energy = energy * 2
```

Diagram 1) Pseudo Code for the one particle energy calculation.

Vector registers are used to pipeline the vector operations. With the use of these registers chaining of pipelines actions can be done. This means that the result of an arithmetic action is not written to memory but stored in one of the vector registers, in order to be used by a different arithmetic unit which works in parallel with the first one. The CRAY Y-MP 4/464 has eight 24 bits address registers, eight 64 bits scalar registers and eight 64 bits vector registers. This number of available registers sets a limit to the amount of chaining of pipelines that can be obtained. Three vector registers in the chaining are used for memory access. This means that five registers are left for the chaining of arithmetic and logic operations. The degree of chaining is defined as the number of operations that can be done in parallel, including memory load and memory store operations.

Diagram 2 gives the important parts of the actions that are done by the functional units and the corresponding vector registers that are used, the actual machine code is too complicated to be discussed here. We have analysed the vector processing in terms of Compound Vector Functions (CVF 's) [8]. A CVF is a symbolic notation for a series of arithmetic actions on vectors which can be chained. An action in the multiply unit is denoted by a '*', and likewise an action in the addition/subtraction unit is denoted by a '+' and '-' respectively. The divide operation is a special case. The CRAY doesn't have a floating point operational divide unit. A divide operation is carried out by first calculating the reciprocal in a reciprocal approximation unit. Then two Newtonian approximation steps are done to get the required precision of 64 bits.

$V_i(I)$ are vector registers in the processor. $A(I)$, $B(I)$ and $C(I)$ represent vector registers in memory. Scalars are stored in a scalar register S .

A vector mask instruction is used to handle the IF condition. The technique is to pick out the indices that satisfy the condition, and gather these in a compressed vector of much shorter length. For this purpose the scalar vector mask register is used (64 bits). Bit n of the mask register corresponds to bit n of the vector register. The elements of the mask register are set to one whenever the condition is true. The count population functional unit is used to set the length of the compressed vector. After that a gather operation is needed to gather the elements corresponding to these indices. Vector registers with a compressed length are denoted with $V_i(J)$.

| Vector operations | Corresponding code and comments |
|---|---|
| $V1(I) = S - A(I)$ | distance_x[i] = perturbed_x_coord. - x_coord[i] |
| $V2(I) = S - B(I)$ | distance_y[i] = perturbed_y_coord. - y_coord[i] |
| $V3(I) = S - C(I)$ | distance_z[i] = perturbed_z_coord. - z_coord[i] |
| $V4(I) = V3(I) * V3(I)$ $V5(I) = V2(I) * V2(I) + V4(I)$ $V6(I) = V1(I) * V1(I) + V5(I)$ | distance[i] = distance_x[i] * distance_x[i] + distance_y[i] * distance_y[i] + distance_z[i] * distance_z[i] |
| Vector Mask: The Vector mask register is set according to the entries in V6(I). Then the indices of the corresponding elements of V6(I) that satisfy the condition are stored in W(J). (Vector of reduced length.) | if (distance[i] < cut-off) |
| Vector Gather: The entries in V6(I) corresponding to the indices in W(J) are fetched and stored in V'1(J). | |
| $V'2(J) = V'1(J) * V'1(J)$ $V'3(J) = V'2(J) * V'1(J)$ | distance[i] = distance[i] * distance[i] * * distance[i] |
| $V'4(J) = 0.5 - V'2(J) * V'1(J)$ | energy = energy + (°0.5 - distance[i]) / (°distance [i] * distance [i]) |
| $V'5(J) = V'3(J) * V'3(J)$ | |
| $V'6(J) = V'4(J) * (1 / V'5(J))$ | divide operation |
| Energy = Energy + $\sum V'6(J)$ | Calculate the energy in partial sums |

Diagram 2) Code diagram for Pseudo Vector Code implementation

In the algorithm the total energy of the system is also calculated. In the sequel this is referred to as the 'total energy calculation'. The pseudo code is given in Diagram 3. The inner loops are vector processed in exactly the same way as the one particle energy loop.

```

Loop over i from 1 to N-1
  Loop over j from i+1 to N
    distance_x [j] = x_coord [j] - x_coord [i]
    distance_y [j] = y_coord [j] - y_coord [i]
    distance_z [j] = z_coord [j] - z_coord [i]
    distance [j] = distance_x[j] * distance_x[j] + distance_y [j] * distance_y[j] +
    distance_z[j] * distance_z[j]
    Condition : distance[j] < cut-off
    distance [j] = distance [j] * distance [j] * distance [j]
    energy = energy + (0.5 - distance[j]) / (distance[j] * distance°[j])
    End Condition
  End loop
End loop

```

Diagram 3) Pseudo Code for the total energy calculation.

3.2 The parallel version

The systolic algorithm has a simple communication pattern that can be efficiently implemented as a ring. The communication overhead is small since each processor contains a complete independent database for the optimization problem. To interchange information about the intermediate state it only has to send and receive at the end of each subchain. This SIMD scheme can be implemented efficiently on a MIMD architecture.

A perturbation in our implementation consists of the displacement of one particle. To calculate ΔE the energy of the particle before the move and after the move is required. Both energy calculations take $N-1$ two-body potential energy calculations. Since the $N-1$ calculations are independent we can perform them in parallel. If we connect a processor farm to a master processor that generates Markov chains it can assign ΔE calculation jobs to processors in the farm. If we use a hybrid implementation, systolic S.A. with energy calculations in a farm, we need a farm attached to every processor in the systolic decomposition, see Figure 2.

In order to cut the communication costs of the farm for the energy calculation, we use a tree configuration and let every processor hold a copy of the complete configuration. The set of N particles is divided into P_{tree} , seven in our implementation, equally sized subsets. Each of the P_{tree} processors is assigned to one of the subsets. The master processor, the root of the tree, generates random moves and sends them down into the tree and takes care of one subset. It collects the returned ΔE values and decides upon acceptance. If a move is accepted an update message is sent to the slave processors together with the next move that has to be calculated. If the radius of the sphere is also optimized then each accepted change of the radius means that the co-ordinates in the slave processors have to be updated. This can be done analogous as described above for the perturbation of a particle.

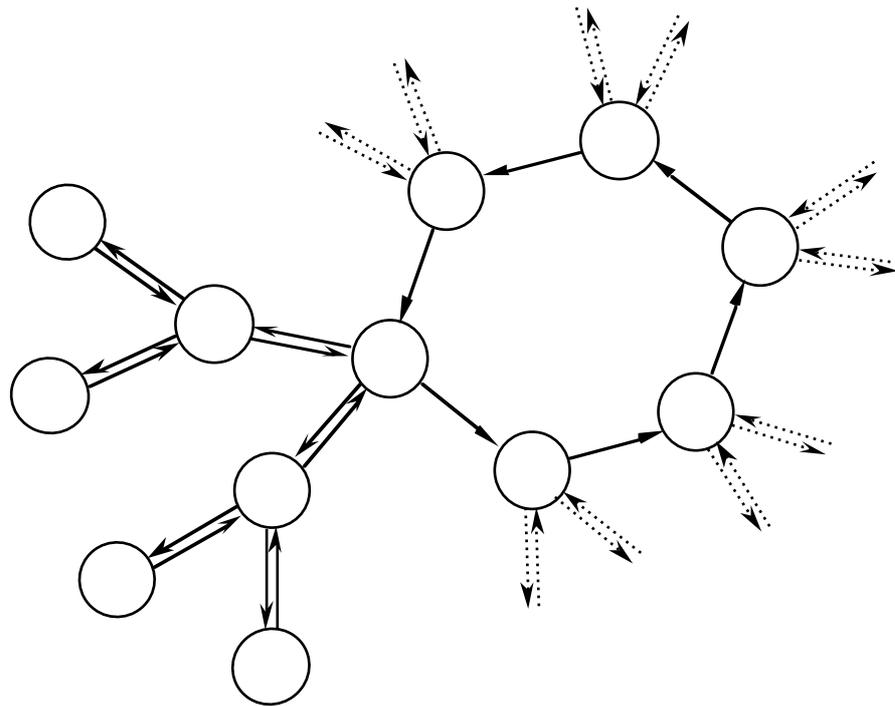


Figure 2) Diagram of hybrid implementation. Ring processors perform the annealing process, tree-slave processors perform energy function calculations, 7 in our implementation.

4 Time complexities

4.1 The sequential version

Each step in a Markov chain consists of the particle move, and the calculation of the energy difference with an update if the move is accepted. The radius is only perturbed in intervals, not every step. In the following we will multiply the time taken for a radius perturbation with a number α , between zero and one, to account for this.

The time needed for the particle move is independent of the number of particles. However the energy calculations for the perturbations of a particle and the radius are dependent on the number of particles. The time complexity for one step in the Markov chain can be written as:

$$T_1^{\text{seq}} = T_m + T_e(N) + \alpha T_r(N) \quad (3)$$

where

T_1^{seq} is the time needed for one step in the Markov chain with a radius update once every $1/\alpha$ steps,

T_m is the time needed to displace one particle,

$T_e(N)$ is the time needed to calculate the energy difference of a displacement $O(N)$,

$T_r(N)$ is the time needed to perturb the radius and calculate the energy difference $O(N^2)$.

If the factor α is chosen as $1/N$, so the radius is updated every N steps, the calculation of a radius perturbation is also of order N . Equation (3) can then be written in the more general form:

$$T_1^{\text{seq}} = c_1 + c_2 * N \quad (4)$$

The constant c_1 is small compared to $c_2 * N$ for larger N , however c_1 can not be neglected for small N . The update step described in the time complexity analyses above has to be multiplied by the length of the chain to get the time taken by producing one chain. We then have to multiply the time for one chain with the number of chains generated for reaching a stable minimum to get the time complexity of the complete annealing algorithm. This can be expressed as follows:

$$T^{\text{seq}} = T_1^{\text{seq}} * L * M \quad (5)$$

where

L is the length of the Markov chain,

M is the number of chains that are generated during the annealing.

L is dependent on the number of particles because the convergence slows down if the number of particles increases. In this work we will use $L=50*N$ although we have determined that the real dependence of L on N is far less optimistic, namely $L \sim N^3$ (data not shown).

In our simulations M has an almost constant value. This can be explained as follows. We adjust the radius of the sphere such that the average distance between neighbouring particles is constant with varying N . Since moving one particle mainly affects particles in the neighbourhood, because of the short range of the Lennard-Jones interaction, the average ΔE per move is independent of N . Consequently, we can keep the initial temperature fixed.

4.2 The vector version

The theoretical maximum performance of the CRAY Y-MP is reached when the 2 functional units, dedicated to addition/subtraction calculations and multiplication calculations respectively, are constantly active. In this case during a clock cycle the results of both an addition/subtraction and a multiplication can be completed. The clock cycle of the CRAY Y-MP is 6 ns, which corresponds to a peak rate of 166.7 MFlop/s, so the theoretical peak performance would be 333.3 MFlop/s.

In this section we shall derive an expression of the time complexity on the basis of an analysis of the implementation of the program. The total time complexity of the vector version consists of a part that is sequentially processed plus a part that is vector processed.

We can derive an expression for the time a vector instruction will take that is executed in a functional unit. This time complexity contains the following contributions:

- + n times τ_{cycle} , the time to process the n elements in the vector pipe,
- + l times τ_{cycle} , the time required to 'fill' the unit, where l is the length of the pipe (equal to the number of components of the operation),
- + 2 times τ_{cycle} , for the following operations: the transfer of each operand to the top of the functional unit and the transfer from the bottom of the pipeline to the result vector register. This contribution is called the latency time.

The expression for the time complexity of a vector instruction on n elements therefore is:

$$t^{\text{vec instr.}} = (n + l + 2)\tau_{\text{cycle}} \quad (6)$$

The length of the pipe for the functional units of the CRAY Y-MP are given in Table 1.

| functional unit | length of the pipeline (l) |
|---|--------------------------------|
| floating point addition | 6 |
| floating point multiplication | 7 |
| floating point reciprocal approximation | 14 |

Table 1) Pipeline lengths per operation

When a number of vector operations is chained the total operation acts like a single pipeline. The following expression can be derived for the time complexity of k chained vector instructions:

$$t^{\text{k chained vec. instructr.}} = \left(n + \sum_{i=1}^k (l_i + 2) \right) \tau_{\text{cycle}} \quad (7)$$

where the index i corresponds to the i -th vector instruction.

With the use of expression (7) it is possible to give an expression of the total time complexity by summing up the contributions from the different parts of the algorithm that can be performed as chained vector operations.

The time complexity for the vector processing on a vector of n elements is often written in the following form [9]:

$$T^{\text{vec}}(n) = R_{\infty}^{-1} q (n + n_{1/2}) \quad (8)$$

where

- R_{∞} is the maximum performance of the loop in MFlop/s. This value is reached in the theoretical case when the loop size goes to infinity,
- q is the number of arithmetic operations,
- $n_{1/2}$ is the loop size for which the performance is $\frac{1}{2} R_{\infty}$.

However the value of R_{∞} doesn't reach the theoretical maximum performance in practice. The reason for this is that it is impossible to chain the functional units in such a way that they are constantly active. The value of $n_{1/2}$ is different for each type of vector operation.

In the next sections we will describe the time complexity of the different parts of the vector version. We will use the parameters introduced in this section. We have to analyse which parts of the algorithm can be vector processed, and which parts can only be sequentially processed. Our aim is to give a complete description of the time complexity of the vectorised version of the S.A. algorithm.

4.2.1 Time complexity of the perturbation step

In this part of the algorithm there are no dependencies on N , therefore there are no parts that can be vector processed. Therefore the time complexity of this step is determined by the sequential processing of the algorithm and is not dependent on N .

$$T_m^{\text{seq}} = \text{constant} \quad (9)$$

4.2.2 Time complexity of the energy update

This step contains two 'one particle energy loops', which are vector processed. These loops take by far the most time. A calculation of the difference in the results of these loops is also done. The time of this sequential calculation can be neglected with respect to the time of the energy calculation loops. For a derivation of the time complexity of the one particle energy loop the discussion given in the section about the implementation aspects is used. It turns out that the cut-off condition reduces the vector length substantially in the part of the algorithm which

has the most intensive arithmetic actions, namely at the point of the calculation of the potential (Lennard-Jones) function. The full vector length of the loop is equal to $N-1$. However the vector of reduced length (set by the vector mask operation) is equal to the number of particles that fall within the cut-off distance. This number is a constant for high values of N , and is about 15.

A simple model of the time complexity is based on the addition of the times of all chained vector functions. The total time complexity of the one particle energy loop with the potential cut-off can be written as follows:

$$T_{\text{energy calc.}}^{\text{vec}}(N) = \left(\sum_{j=1}^m \left[\sum_{i=1}^{k_j} [l_i + 2] + N - 1 \right] + \sum_{j=1}^{m^*} \left[\sum_{i=1}^{k_j^*} [l_i^* + 2] + 15 \right] \right) \tau_{\text{cycle}} + \tau_{\text{load}} \quad (10)$$

where l_i is the length of the i -th pipeline and where the latency time is equal to 2 clock cycles. In both terms j is the index for all separate parts that could not be chained, there are m of those parts. Within such a part we have calculations that are chained, there are k_j of those calculations in part j and they have index i . τ_{load} comes from the time required to load the vector registers from memory.

The second term comes from the part of the algorithm where a reduced vector length is used, due to the cut-off condition. The asterisk (*) is used for the vector operations with a reduced vector length.

The time $T_{\text{energy calc.}}^{\text{vec}}$ is the time for the calculation of the energy of one particle. The energy difference that needs to be calculated in the Markov chain consists of two of those loops and is denoted by : T_e^{vec}

4.2.3 Time complexity of the radius update

The calculation of the total energy is the only part of the algorithm in this step that substantially adds to the time complexity in this step, because this calculation is of order N^2 . In good approximation we can therefore neglect all other contributions. From the discussion of section 3.1 it follows that this calculation consists of $N-1$ one particle loops with an average length of $\frac{1}{2}N$. The time complexity of this step is therefore given by:

$$\begin{aligned} T_r^{\text{vec}} &= \sum_{j=2}^N T_{\text{energy calc.}}^{\text{vec}}(j) \\ &\cong \frac{1}{2}(N-1)T_{\text{energy calc.}}^{\text{vec}}(N) \end{aligned} \quad (11)$$

4.2.4 Total time complexity of the vector implementation

The three steps given above together give the expression for the time complexity of one step in a Markov chain. This cumulative expression is as follows, where α will be chosen as $\frac{1}{N}$ (see section 4.1) :

$$\begin{aligned}
T_1^{\text{vec}}(N) &= T_e^{\text{vec}}(N) + \alpha T_r^{\text{vec}}(N) + T_m^{\text{seq}} \\
&\approx 2T_{\text{energy calc.}}^{\text{vec}}(N) + \alpha \frac{1}{2}(N-1)T_{\text{energy calc.}}^{\text{vec}} + T_m^{\text{seq}}
\end{aligned} \tag{12}$$

Multiplication with the chain length, L , and the total number of Markov chains, M , gives an expression for the total time complexity of the vector version:

$$T^{\text{vec}}(N) = L * M * T_1^{\text{vec}}(N) \tag{13}$$

4.3 The parallel version

Our parallel implementation is a combination of systolic S.A. and functional decomposition of the energy calculation. First the functional decomposition is discussed.

4.3.1 Functional decomposition

The times for the calculation of the energy difference during a perturbation of a particle and the calculation of a perturbation of the radius in the tree decomposition are described by :

$$T_e^{\text{fun}} = c_3 * \frac{N}{P_{\text{tree}}} + T_{c1} \tag{14}$$

$$T_r^{\text{fun}} = c_4(N) * \frac{N}{P_{\text{tree}}} + T_{c1} \tag{15}$$

where

T_{c1} is the communication time associated with the parallel implementation of the tree,
 c_3 is the time constant for the calculation of the energy difference of a particle move,
 $c_4(N)$ is the time constant for a radius perturbation.

The constant c_4 is dependent on N because the calculation of the total energy of the system is of order N^2 , if we choose $\alpha = 1/N$ this dependency is removed.

The time complexity of one step in the Markov chains is analogous to the sequential time complexity, equation (3):

$$\begin{aligned}
T_1^{\text{fun}} &= T_m + c_3 \frac{N}{P_{\text{tree}}} + T_{c1} + \alpha \left(c_4(N) \frac{N}{P_{\text{tree}}} + T_{c1} \right) \\
&= T_m + (c_3 + \tilde{c}_4) \frac{N}{P_{\text{tree}}} + T_{c1} \left(1 + \frac{1}{N} \right)
\end{aligned} \tag{16}$$

where

$$\tilde{c}_4 = \frac{c_4(N)}{N}$$

The speedup gained by functional decomposition, $\frac{T_1}{T_1^{\text{fun}}}$, can be close to P_{tree} for large numbers of particles.

4.3.2 Systolic and hybrid S.A.

In systolic S.A. P chains are generated at the same time. This means that in the time that in the sequential version one chain is generated, P chains are generated in the systolic version. Again a communication time is associated with the implementation.

The number of chains that are generated in systolic S.A. is somewhat more than in the sequential version since the algorithm has to start up the processors in the ring one by one (see Figure 1). This means that the total number of chains is equal to the number of chains in the sequential version plus is the number of processors in the ring minus one. Thus the time complexity of the hybrid version is given by:

$$T_p^{\text{hyb}} = (T_1^{\text{fun}} * L / P + T_c) * (M + P - 1) \quad (17)$$

where

P is the number of processors in the systolic ring,

T_c is the communication time for transfer of information between subchains.

T_c can be neglected for large N because the communication contribution to the total time is much lower than the contribution of the calculation of the subchains.

We can now calculate the speedup of the hybrid version with respect to the sequential version :

$$S_p^{\text{hyb}} = \frac{T^{\text{seq}}}{T_p^{\text{hyb}}} = \frac{T_1 L M}{(T_1^{\text{fun}} * L / P + T_c) * (M + P - 1)} \quad (18)$$

$$= \frac{1}{\frac{T_1^{\text{fun}}}{T_1} \left(\frac{1}{P} + \frac{1}{M} - \frac{1}{MP} \right) + \frac{T_c (M + P - 1)}{T_1 L M}} \approx \frac{T_1}{T_1^{\text{fun}}} P \quad (19)$$

For large P the speedup will not be given by this function because at large values of P the subchains are too short. This results in a higher number of chains that have to be calculated in order to achieve a stable minimum. Therefore we have the dependency $M=M(P)$. If the length of the subchains is still large enough, the number of generated chains is equal to the number of chains generated with the sequential version. Instead of using $M=M(P)$ we can also use $L=L(N,P)$. If we use a longer chain for higher numbers of processors we can make sure that the found solution and the number of chains become equal to the sequential case.

5 Results

5.1 The vector version

This section presents the expressions for the time complexity of the vector version of the S.A. algorithm.

We assume that for high vector lengths (high N) we can write the time complexity of the one particle energy loop with a term that is constant and a term that is linearly dependent on N . We base this assumption on the fact that for high N a constant number of operations is done per clock period in which case the processor reaches its optimal performance. The time complexity of the energy update can now be written as:

$$T_{\text{energy calc.}}^{\text{vec}}(N) = c_1 + c_2(N) \quad (20)$$

It was not possible to give a reliable theoretical expression of the time complexity based on equation (10) because it was not possible to analyse the machine code in terms of chained vector functions.

We have done timings of the potential energy loop as a function of N in vector mode under full optimization. The loop is timed with the potential cut-off restriction and without this restriction for N ranging up to 4000. The results are summarised in Table 2.

| N | $T_{\text{energy calc.}}^{\text{vec}}(N)$ (no cut-off) (μs) | $T_{\text{energy calc.}}^{\text{vec}}(N)$ (with cut-off) (μs) |
|------|--|--|
| 50 | 7.00±0.07 | 8.20±0.08 |
| 100 | 11.00±0.10 | 13.50±0.13 |
| 200 | 18.8±0.18 | 23.6±0.24 |
| 400 | 33.3±0.3 | 36.3±0.3 |
| 650 | 52.3±0.5 | 51.4±0.5 |
| 1000 | 77.6±0.7 | 66.0±0.7 |
| 2000 | 152±1.5 | 125±1.3 |
| 4000 | 300±3 | 225±2 |

Table 2) Timings of the one particle energy calculation on the CRAY Y-MP

This data are shown graphically in Figure 3.

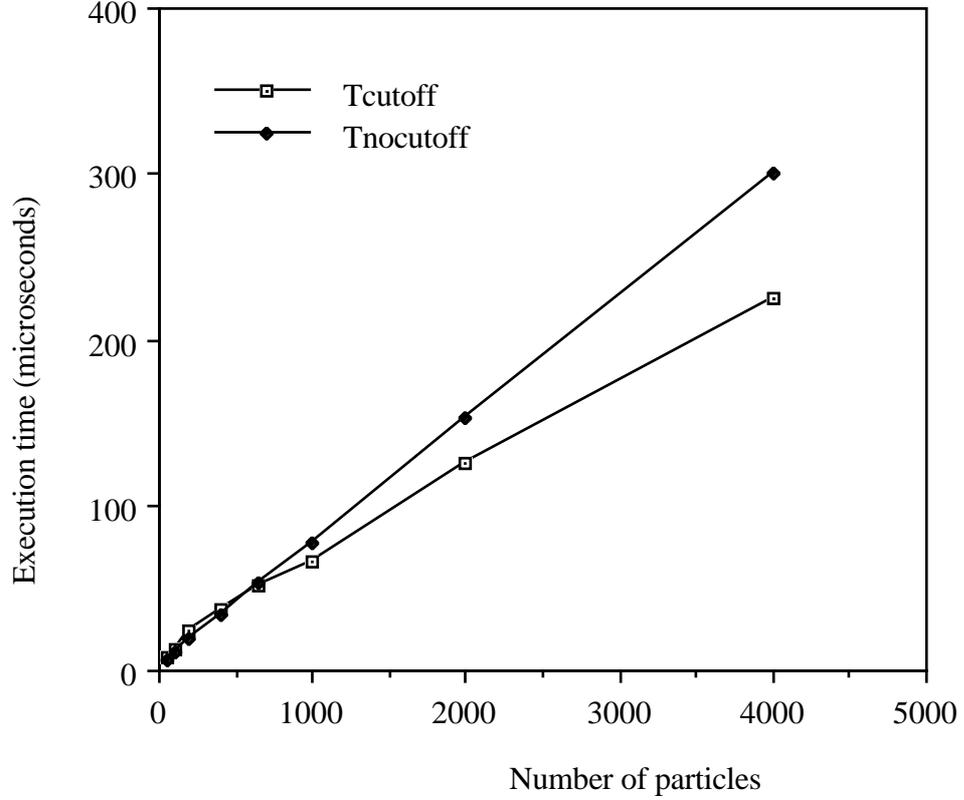


Fig. 3) The execution times for the vector implementation on the CRAY for the one particle potential energy loop both with potential cut-off and without potential cut-off.

We see that for N less than about 650 the calculation with potential cut-off is not faster than the calculation without potential cut-off. This is due to the overhead induced by the cut-off condition. Above this point however there is much less arithmetic that has to be done and therefore the version with the potential cut-off is faster. In both cases above a certain value of N the time complexity grows linearly with N . In this case the time complexity consists of a constant term plus a term that is linearly dependent on N .

A fit on this data gives an experimentally derived expression for the time complexity of the one potential energy loop both with and without the cut-off condition implied. The time complexity for high N is given by the following expressions for the energy update step for the cases with and without potential cut-off. Based on the last 4 data points we find (the correlation coefficient in both cases was equal to 1.00) :

$$T_{\text{energy, cut-off}}^{\text{vec}}(N) = (8.9 \pm 0.65) + (0.053 \pm 9.8 * 10^{-5})N \mu s \quad (21)$$

$$T_{\text{energy, no cut-off}}^{\text{vec}}(N) = (3.65 \pm 0.65) + (0.074 \pm 1.7 * 10^{-4})N \mu s \quad (22)$$

These expressions have the same form as equation (20). We therefore deduce the following values for c_1 and $c_2(N)$ for the cut-off case for high N :

$$c_1 = 8.9 \mu s$$

and

$$c_2(N) = 0.053N\mu s$$

The calculation of the energy update step consists of two loops which we have timed and which have the time complexity function given by equation (21) and (22). The total complexity of the energy update step with cut-off condition, assuming that the sequential part can be neglected, is for high values of N equal to:

$$T_{e, \text{cut-off}}^{\text{vec}}(N) = 17.8 + 0.106N \quad (23)$$

The timing, given above, of the one energy loop is used to give an expression for the time complexity of the radius update step which contains a calculation of the total energy. If we fill in equation (23) in the expression for $T_r^{\text{vec}}(N)$, equation (11), we get:

$$T_r^{\text{vec}}(N) = (N - 1)(4.44 + 0.0265N) \mu s \quad (24)$$

We have done a timing of the perturbation step in the experimentally derived time description, which is nearly completely sequential. The result is: $T_m^{\text{seq}} = 125.0\mu s$.

If we choose $\alpha = \frac{1}{N}$ and if we fill in the values and expressions for T_m , $T_e^{\text{vec}}(N)$, and $T_r^{\text{vec}}(N)$ in equation (12) we get the following experimentally derived expression for the time complexity of one step in the Markov chain for the vector version for high number of particles:

$$\begin{aligned} T_1^{\text{vec}}(N) &= 125.0 + 17.8 + 0.106N + 4.44 + 0.0265N \mu s \\ &= 147.2 + 0.133N \mu s \end{aligned} \quad (25)$$

Our conclusion is that from $N = 780$ and up the vectorised part of the code takes more time than sequential part. The time complexity of the complete vector version with cut-off restriction for high number of particles is given by:

$$T^{\text{vec}}(N) = L * M * T_1^{\text{vec}}(N) = L * M * (147.2 + 0.133 * N) \mu s \quad (26)$$

We have made a comparison of the experimental timings of the complete program running on the CRAY Y-MP and the theoretical values following from equation (26). The results are given in the following table:

| N | L | M | $T_{\text{timed}}^{\text{vec}}(N)$ (sec) | $T_{\text{theoretical}}^{\text{vec}}(N)$ (sec) |
|-----|-------|-----|--|--|
| 50 | 3000 | 393 | 156 | 181 |
| 100 | 11000 | 407 | 652 | 717 |

Table 3) A comparison of the theoretical and timed execution times on the CRAY Y-MP of the vectorised version of the S.A.-program with applied cut-off condition.

We see that the predicted results are in reasonable correspondence with the timed values.

5.2 The parallel version

If we look at the time complexities, given in section 4.3, we see that they contain constants which are unknown. We can find these constants if we have the implementation and a table of the time that is consumed by the various operations such as multiplication, addition etc. The operations and their time constants were experimentally derived for the T805 under PARIX and are listed in the Table 4 below :

| operation | consumed time (μs) |
|-----------------------|---------------------------|
| empty loop (per step) | 1.32 |
| assignment | 1.11 |
| multiplication | 0.602 |
| division | 1.14 |
| addition | 0.341 |
| subtraction | 0.376 |
| if (double<double) | 1.55 |
| if (int < int) | 0.976 |
| if (int == int) | 0.671 |
| comm. start-up | 65 |
| comm. per byte | 0.87 |

Table 4) Consumed time by some relevant operations on the T805 transputer

5.2.1 The tree decomposition

The comparison between the method of counting operations, as described above, and direct timing turns out to give the correct order of magnitude. We timed the calculation of the energy difference on one of the processors in the tree decomposition without the communication operations and found for $N=140$:

$$T_{e, \text{calc}}^{\text{tree}} = 550 \mu s \quad T_{e, \text{timed}}^{\text{tree}} = 650 \mu s$$

Where $T_{e, \text{calc}}^{\text{tree}}$ is the calculated time from the instruction count. These results show that it is indeed possible to estimate the time a specific part of the code will consume by looking at the instruction level. It is also clear that there is some discrepancy between the timed code and the time deduced from the operations. This discrepancy is due to some overhead that is not well accounted for by just counting the number of times an operation occurs.

Now that we know that it is possible to estimate the time by looking at the code we switch to timing the basic blocks and not the operations that are processed within the blocks.

First we timed the costs of a send/receive operation. The communication time consists of two parts, the start-up time and a part that is dependent on the length of the message. The timing of these communication routines gave the following results [10] :

$$T_{\text{startup}}^{\text{com}} = 65\mu\text{s} \quad T_{\text{size}}^{\text{com}} = 0.87\mu\text{s} / \text{byte}$$

These times are for synchronous communication. The mentioned time for the size dependency of the communication can be higher if the message needs to be routed through other processors.

Now that we have timed the calculation blocks and the communication time we can fill in the total time needed for the tree decomposition. To do this one has to realise that some of the communication in the tree can be done in parallel. In fact if we use a tree of 7 processors we have, at the start of the calculation, 4 communications that can not be done in parallel (each 16 bytes long, 3 doubles and 2 integers). The communication needed for the return of the results, has a shorter length than the messages for the start of the calculations since it only contains the calculated energy difference (one double value). We also have the fact that some processors are finished in a shorter time than others. This has two reasons, first, some processors received the start data earlier and, secondly, because some have a set of data that contain particles that are not in the cut-off radius of the perturbed particle (so not all potentials have to be calculated).

If we ignore this last (helpful) effect we can say that the total time is 8 communication times plus one calculation time. Filling in the measured time constants for the communication and for the calculation of the energy difference we find that for $N = 140$:

$$T_{\text{calc}}^{\text{tree}} = 1250\mu\text{s} \quad T_{\text{timed}}^{\text{tree}} = 1550\mu\text{s}$$

This discrepancy in measured and calculated time of the tree decomposition is due to the time spend waiting for the processor that happens to have to do the most calculations to finish and to send its results.

If these measurements are repeated for other numbers of particles we can write down the function that gives the time spent by the tree as a function of N . We will give a similar formula for the sequential version of the energy calculation as a comparison :

$$T^{\text{seq}} = 31 * N \mu\text{s} \quad T^{\text{tree}} = 4.4 * N + 960 \mu\text{s}$$

If we use several thousands of particles we can expect a speedup close to 7 which is the number of processors calculating the interactions. This decomposition is faster than the sequential program if we use more than 36 particles.

The radius update has the same functional description as the calculation of the energy difference. Again only a small message has to be send back and forth so the time for communication is the same. The only difference is that a calculation of the energy difference for a radius perturbation needs more calculations. If we perturb the radius once every N particle perturbations we get :

$$\frac{T_r}{N} = 2.0 * N + \frac{960}{N} \mu\text{s} \quad (27)$$

5.2.2 The systolic simulated annealing algorithm

The complete time that one step in the Markov chain without the tree decomposition takes is (see sections 4.3.2 and 5.2.1) :

$$T_1^{\text{sys}} = T_m + T_e(N) + \alpha T_r(N) = 1400 + 31 * N + \alpha * 28 * \frac{1}{2} * N * (N - 1) \quad \mu s \quad (28)$$

If we choose $\alpha = 1/N$ then we get :

$$T_1^{\text{sys}} = 1400 + 31 * N + 14 * (N - 1) = 1386 + 45 * N \quad \mu s \quad (29)$$

There is also a communication time associated with the systolic algorithm. The time complexity already indicates that if the sub chain length (L/P) is long, the communication time is small compared with the calculation time of the Markov chain. The communication times are much longer than the time estimated from the length of the message and the start-up time :

$$T_{\text{calc}}^{\text{com}} = 12 * 10^3 \mu s \quad T_{\text{timed}}^{\text{com}} = 75 * 10^3 \mu s$$

This is due to the fact that processors have to wait for their predecessor to obtain the information needed to continue. This information that has to be sent to the next processor in the ring is done by using asynchronous communication. The deviation of the timed communication time is very large, indicating that some processors have to wait a long time before they can receive their information from the previous processor. The minimum time found is $22 * 10^3 \mu s$.

The value for the number of chains, M, and the necessary chainlength are the same for the sequential and the vector versions, this is also true for a parallel implementation where only the calculation of the potential energy is parallelized. But the systolic algorithm is functionally different. We notice that if we increase the number of processors in the ring that the number of chains that have to be generated increases. This is because the smaller the subchains, the more chains have to be generated before a stable minimum is found. This leads to a decrease in efficiency. Together with the increase in the number of chains the found solution gets worse. This means that we actually have the dependency $L(N,P)$ if we want to keep the same quality of the solutions and the number of chains given by $M(P)=M+P-1$ where M is the sequential number of chains.

To show what the influence of the number of processors is on the systolic S.A. algorithm we have listed the results of the experiments with $N=50$ in Table 5. In this table we give the average number of chains, the average minimum energy found and the average time that the simulations took. These simulations were done without optimization of the radius. Instead we kept the density equal to the optimal density on a flat surface.

| number of processors | number of chains | energy | time (s) |
|----------------------|------------------|--------------------|--------------------------|
| 1 (sequential) | 157 ± 1 | -2.761 ± 0.002 | $1.31 \cdot 10^3 \pm 20$ |
| 4 | 161 ± 2 | -2.754 ± 0.007 | 359 ± 4 |
| 8 | 175 ± 10 | -2.74 ± 0.02 | 196 ± 4 |
| 16 | 230 ± 30 | -2.70 ± 0.03 | 140 ± 20 |
| 32 | 360 ± 50 | -2.68 ± 0.06 | 120 ± 15 |

Table 5) the results of the systolic S.A. algorithm for N=50 (based on 5 experiments)

If the model description of the systolic S.A. is filled in for N=50 and N=100 using the number of chains found (given in Table 5 for N=50) we find a good agreement with the experimentally found times (see Figure 4). If we fill in $(M+P-1)$ for the number of chains the calculated times have a different form than the experimentally found times. The difference between the calculated and timed execution times indicate that the parts of the real implementation that are not accounted for in the model have a dependency on N. The extra time can be expressed by : $T_{\text{extra}} \approx 1.1 * N$ seconds.

The S.A. algorithm is an inherently sequential scheme. Parallelizing it by the systolic S.A. method we introduce a functional difference to the sequential version. This parallelization has consequences for the accuracy of the iterative processes so that more chains have to be generated. We can counter balance this by using a larger chain length than in the sequential case.

If we determine what the chain length $L(N,P)$ should be to keep the quality of the solutions the same as in the sequential case we find for N=100 (see Table 6):

| P | L |
|----------|----------|
| 4 | 90 |
| 8 | 140 |
| 16 | 275 |
| 32 | 700 |

Table 6) Chainlength at some values for the number of processors

The execution times for these new chain lengths are such that a large part of the speedup is lost. See Figure 5 in section 5.3.

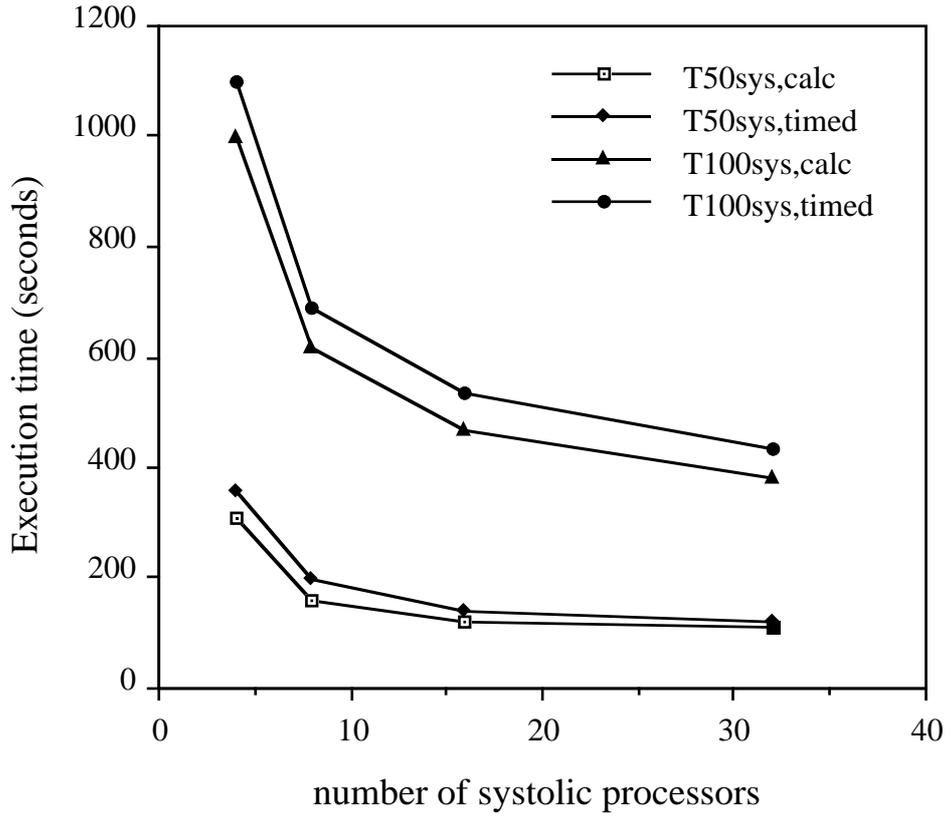


Fig 4) The calculated and timed execution times for N=50 and N=100 of systolic S.A.

The complete time complexities of the systolic and the hybrid (systolic and tree decomposition) implementations without the extra time defined in this section are :

$$T^{\text{sys}}(N) = \left(\left(1.4 * 10^3 + 45 * N \right) * \frac{L(N,P)}{P} + 75 * 10^3 \right) * (M + P - 1) \quad \mu s \quad (30)$$

$$T^{\text{hyb}}(N) = \left(\left(2.4 * 10^3 + \frac{9.6 * 10^2}{N} + 6.4 * N \right) * \frac{L(N,P)}{P} + 75 * 10^3 \right) * (M + P - 1) \quad \mu s \quad (31)$$

5.3 Comparison of the vector and parallel implementations

Vector computing does not have the ability to be adapted to a specific problem but its power lies in the fact that you can easily adapt the sequential code. With a minimum of trouble, you can achieve high speedups.

A massively parallel machine has a large degree of freedom in connectivity so that the processor configuration can be adapted to the parallelism in the algorithm. Therefore, at the cost of much time, all kinds of implementations can be tested to find the one which performs best. The systolic implementation is different from the sequential code. It turns out that the systolic implementation in the present form is not capable of using large numbers of processors. This problem is not cured by using large numbers of particles although this is what you would expect since the subchains get longer. The reason for this is that not many configurations from a chain get accepted in the next chain of lower temperature. So the first configuration of a chain (which is the configuration of the first sub-chain of the previous Markov chain) is often the only configuration that is used for the calculation of the whole chain. One expects that the cool-rate is of influence on the rate of acceptance of configurations of a previous chain since the difference in temperature between the chains is smaller. It turns out that a better cool-rate (closer to 1) has as effect that more chains need to be generated because the temperature drops slower but the chains are allowed to be shorter. The ratio of the number of accepted configurations of a previous chain divided by the total number of tried configurations from a previous chain is constant and very low (≈ 0.04).

Next we compare the time complexities of the vector version with the best performing parallel version, formula (26) and (31). The results are given in Figure 5.

Our calculations show that the systolic implementation does not have good scaling properties and therefore it is not possible to outrun the CRAY computer.

The computing power of the CRAY is 333 MFlop/s while that of a T805 transputer is about 1 MFlop/s. If we look at the data of Figure 5 at 16 systolic processors we have 112 processors for the hybrid implementation. This is equivalent with 112 MFlop/s which is three times lower than the CRAY. The execution time, however, is 6 times longer for the parallel version. This means that a factor 2 is spent on communication and on the loss of precision in the iterative scheme of the parallel version.

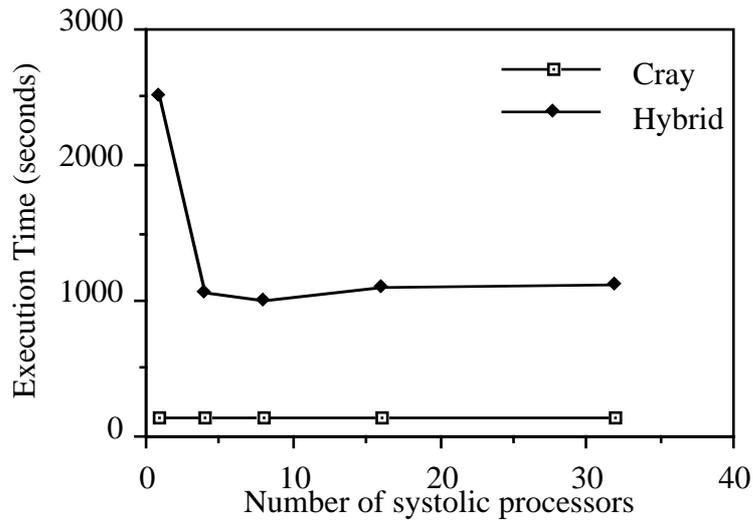


Fig 5) The execution times (N=100) for the vector implementation on the CRAY is compared with the execution time of the hybrid implementation as a function of the number of processors. The total number of processors is 7 times larger for the hybrid implementation than for the systolic since each processor is now part of a tree with 7 processors.

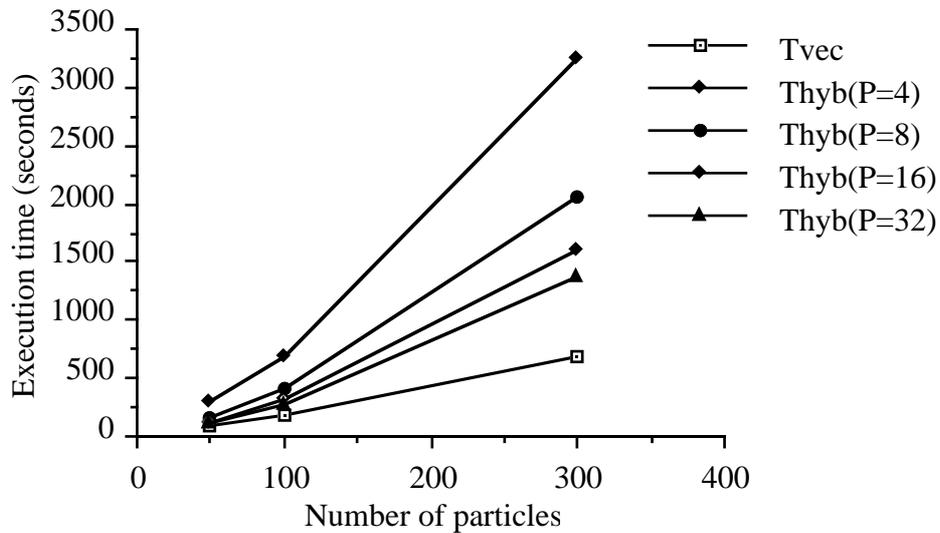


Fig 6) The execution times for the vector implementation on the CRAY is compared with the execution time of the hybrid implementation as a function of the numbers of particles. P is the number of systolic processors, the total number of processors is $7 \cdot P$ for the hybrid version.

6 Results of the crystallization experiments

In this section we will discuss some of the results of crystallization experiments on a spherical surface, obtained with the S.A. implementations described above (See for more details [11]).

Some of the results that we obtain with the S.A. implementations can be checked for correctness. For example if twelve particles are annealed, $N=12$, we know that the result is one of the Platonic solids. This is reproduced by our annealing program. Some other configurations can be checked with literature, especially comparable experiments with a Coulomb potential [12]. The solutions found there can be reproduced with our implementation.

If we look at the coordination number of the particles i.e. the number of direct neighbours identified by the Voronoi construction, we see that 12 particles with a pentagonal surrounding are needed to account for the curvature. For $N=20$ we do not find the corresponding Platonic configuration since this configuration does not have the optimal number of pentagonal particles. For $N=32$ we find a configuration consisting of a dodecahedron and icosahedron merged. The configuration thus obtained has 12 pentagonal and 20 hexagonal particles.

With the Lennard-Jones potential there is a radius where the distances are optimal. From a series of simulations with different radii we find that the underlying inter particle potential is reflected in the energies of the total system, as is given in Figure 7.

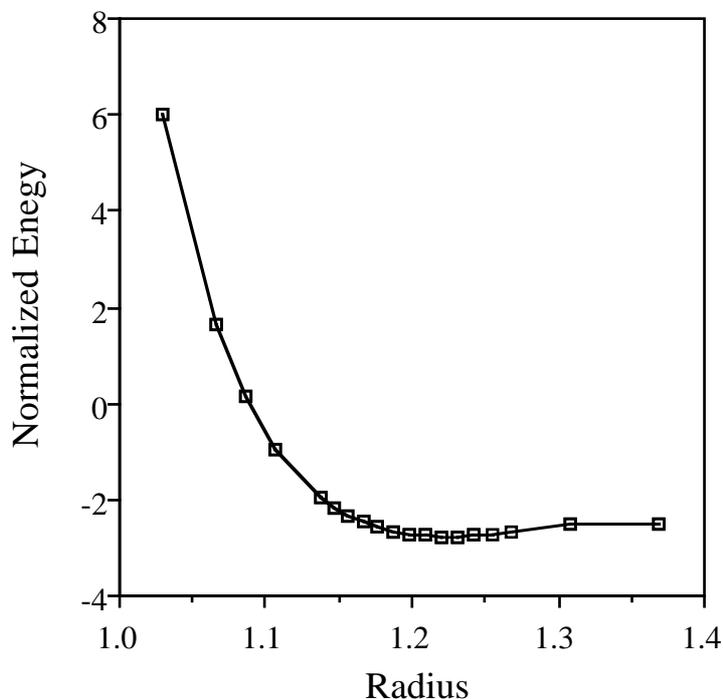


Figure 7) The energy of a system of 20 particles as a function of the radius of the supporting sphere

If we examine the configurations of the particles we notice that they are not the same for all radii. If the radius is larger than some specific value the configuration will not span the whole sphere because the Lennard-Jones potential will hold the particles together. With the Coulomb potential this effect is not observed. If the radius is larger than the optimum radius but small enough to span the sphere, it is found that with $N=20$ the configuration is not the same as that at the optimal radius, see Figure 8. This phenomenon is known as symmetry breaking.

Figure 8) $N=20$ the configuration with $R=1.7$ (left) and $R=R_{\min}=1.223$ (right). The lattice defects are indicated by two white square planes.

We can plot the potential energy versus the number of particles. This is shown in Figure 9. It is clear that at some values of N there are configurations of lower potential energy than at the neighbouring values of N . This induces a sort of discretization in the size of spherical particle systems.

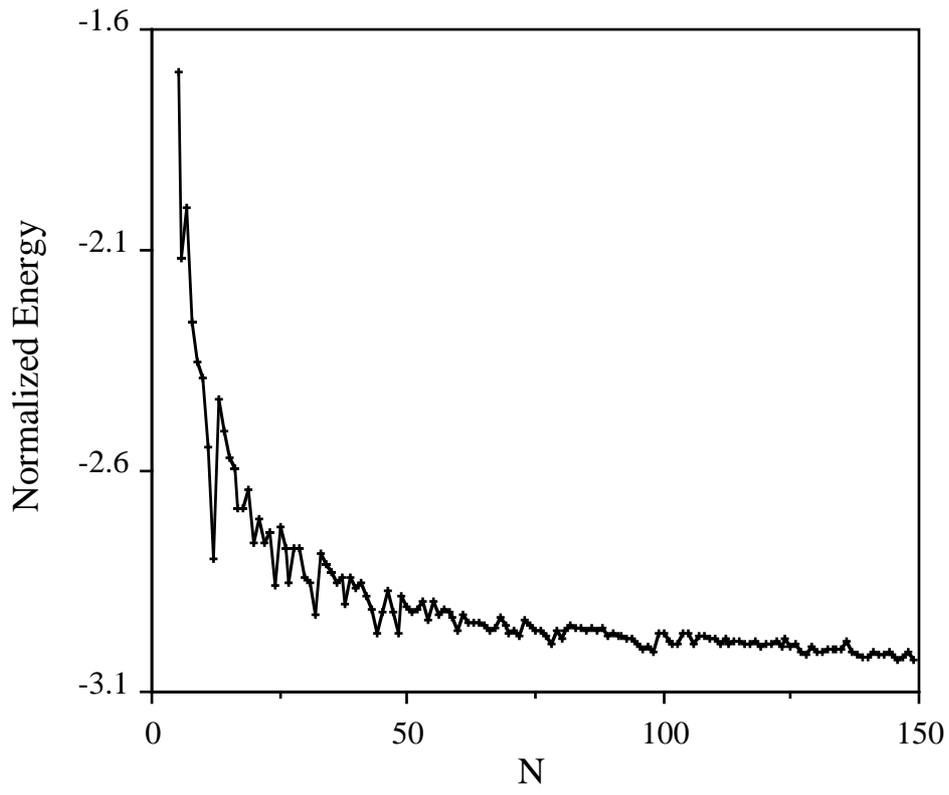


Figure 9) The energy of the system as a function of N.

7 Conclusions

7.1 Comparison of the vector and parallel implementations

The description of the time complexities have been compared to the execution times found by experiments. The agreement was good, the important parts which contain the parameters for scaling, P and N , are well modelled. This shows that the derived time complexity of the computational model is a good approximation for the actual computations so we can use this model for predictions.

The S.A. algorithm is an inherently sequential scheme. Parallelizing it by the systolic S.A. method we introduced a functional difference with respect to the sequential version. This parallelization has consequences for the accuracy of the iterative processes. In order to counterbalance this parallelization effect we need to find $L(N,P)$. It turns out that the systolic implementation in the present form is not capable of exploiting large numbers of processors efficiently.

With small number of processors the systolic S.A. is not faster than the vector version. This is due partly to the fact that the clock speed of the CRAY is much higher than on the T805 transputer. In a short while a T9000 based transputer platform will be installed at this institute. The clock speed of those transputers is higher than the T805 of the Parsytec GCel. Also the communication will be completely handled by hardware. Preliminary results [13] show that the T9000 will be a factor 5 to 8 faster. With those processors we will be able to come close to the performance of the CRAY. If we look at the cost/performance rate for the CRAY and the Parsytec transputer system then we see that in a typical case (see section 5.3) the CRAY has an execution time that is 6 times faster with 3 times as much computing power. But the cost of one processor of the CRAY is about 3 million dollars while 112 nodes on the transputer system cost about half a million dollars. The execution time is six times faster on the CRAY but the costs are also six times higher. So the cost/performance figures for the transputer system and the CRAY are the same.

7.2 Crystallization experiments

From our experimental results, and by following the annealing process, we can infer that the programs do find the optimal configurations.

The discretization that is visible in Figure 9 is the most distinct for small numbers of particles. This does not mean that higher numbers of particles do not have this effect. It is expected that clusters will form in some systems with many particles. These clusters can then be arranged in patterns according to the optimal configurations with a low potential energy. This hypothesis of hierarchically ordered clusters will be subject to future study.

8 Future research

Future research will be conducted in various directions. The physics of the optimal distributions of particles will be examined closely. For example properties like the orientation correlation functions and distance distributions will be determined. The optimal distributions will also be examined by a tool that tries to construct the simulated configuration starting from basic configurations like Plato's solids and triangles and pentagons. This will be used to test the hypothesis on hierarchically clustered particles.

Research is planned for further examination of the implementations discussed in this work. Attention will be given to a implementation with a dynamically adapted temperature. In those implementations the temperature of the following chain is determined by the deviations of the cost function, the potential energy in our case. We will keep on looking for other implementations that could beat our current implementations.

A new research project will be devoted to the implementation of a method for lowering the complexity of the potential energy interactions. As mentioned in the section 2.1 the interactions are the most time consuming parts of particle simulation programs. N-body methods possess an algorithmic complexity of $O(N^2)$ if all pair-wise interactions are calculated (the direct algorithm). For our simulations the number of interacting particles has to be very large. The $O(N^2)$ complexity of the direct algorithm is a severe restriction for these large scale many-body simulations. Even on the most powerful (massively parallel/vector) super computers the execution times of realistic many-body simulations will soon rise above acceptable (or affordable) values. The conclusion is that the algorithmic complexity of the direct method must be reduced. Some interaction potentials (e.g. Lennard-Jones) allow the use of cut-off techniques, which can reduce the complexity to $O(N)$. However, for long range interaction potentials, such as Coulomb or gravitational potentials, such cut-off techniques cannot be applied.

A very important class of 'clever' many-body algorithms, which reduce the complexity to $O(N \log N)$ or even to $O(N)$, are the so-called hierarchical tree methods [14,15]. In these methods the interaction is not calculated for each particle pair directly, but the particles are grouped together in a hierarchical way, and the interaction between single particles and this hierarchy of particle groups is calculated. Appel [16] introduced the first hierarchical tree method, which relies on the use of a mono-pole (centre-of-mass) approximation for computing forces over large distances and sophisticated data structures to keep track of which particles are sufficiently clustered to make the approximation valid. This method achieves dramatic speedups compared to the direct algorithm, but is less efficient when the distribution of particles is relatively uniform and the required precision is high.

The next step, which was set by Greengard [17], is the use of multipole expansions to compute potentials or forces. This approach is known as the Fast Multipole Method (FMM), and requires an amount of work proportional to N to evaluate all pairwise interactions to any degree of accuracy. Up till now FMM algorithms have only been developed for $1/r$ potentials in two and three dimensions. Hierarchical tree methods have proven to be very efficient and accurate, and well suited to be used in realistic many body simulations. However, efficient implementation on High Performance Computing platforms, specifically massively parallel distributed memory computing systems, is far from trivial.

Acknowledgements

The authors wish to thank Dr. R. van Dantzig (NIKHEF-K) and Prof. D. Frenkel (AMOLF) for fruitful discussions on the physics of the experiments.

The research was co-funded by the 'Stichting Nationale Computer Faciliteiten' (NCF CRG-91-08) and the FOM 'Fundamenteel Onderzoek der Materie' under number FI-A-a-3640.

Reference List

- [1] R. van Dantzig, P.M.A. Sloot and W.S. Bont, *Stability and size of spherical bilayer vesicles*. submitted
- [2] W.S. Bont, *The Diameters of Membrane Vesicles Fit in Geometric Series*, J. of Theor. Biology 74 (1978), pp. 361-375
- [3] S. Kirkpatrick, C.D. Gelatt, Jr., M.P. Vecchi, *Optimization by Simulated Annealing*, Science 220, number 4598 (May 1983), pp. 671-680
- [4] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller, *Equation of State Calculations by Fast Computing Machines*, J. of chem. physics, Volume 21, number 6 (1953), pp. 1087-1092.
- [5] E.H.L. Aarts, F.M.J. de Bont, E.H.A. Habers and P.J.M van Laarhoven, *Parallel implementations of the Statistical Cooling algorithm*, North Holland Integration, the VLSI journal 4 (2986), pp. 209-238
- [6] Youngtak Kim, Myunghwan Kim, *A step-wise-overlapped parallel annealing algorithm on a message passing multiprocessor system*, Concurrency: practice and experience, vol. 2(2) (June 1990), pp. 123-148
- [7] A. ter Laak, L.O. Hertzberger, P.M.A. Sloot, *NonConvex Continuous Optimization Experiments on a Transputer System*, Transputer Systems - Ongoing Research, ed. A.R. Allen (IOS Press, Amsterdam, 1992) p.251
- [8] K. Hwang, *Advanced Computer Architecture, Parallelism, Scalability and Programability*, (Mc Graw-Hill Series in Computer Engineering, 1993, ISBN 0-07-031622-8)
- [9] R.W. Hockney, C.R. Jesshope, *Parallel Computers 2, Architecture, Programming and Algorithms*, (Adam Hilger, Bristol and Philadelphia, second edition, 1985)
- [10] P.M.A. Sloot, A.G. Hoekstra, J. Vesseur, F v.d. Linden, M. van Muiswinkel, *PVM, Express, Parix, Occam : Performance comparison on a transputer platform*, submitted
- [11] P.M.A. Sloot, A. ter Laak, P. Pandolfi, R. van Dantzig and D. Frenkel (1993). *Crystallization on a Sphere*, in Physics Computing '92, World Scientific Singapore, 1993 pp 471-472
- [12] T. Erber, G.M. Hockney (1991). *Equilibrium Configurations of N Equal Charges on a Sphere*, Fermilab-pub-91/222-T
- [13] G. Bader, B. Przywara (1993, May 5), *T9000 - A Preliminary Evaluation of Arithmetic Performance*, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen, Universität Heidelberg, Im Neuenheimer Feld 368, D-6900 Heidelberg, t9@iwrl.iwr.Uni-Heidelberg.de
- [14] Leslie F. Greengard, *Rapid Evaluation of Potential Fields in Particle Systems*, (The MIT Press, 1988).
- [15] John K. Salmon, *Parallel Hierarchical N-Body Methods*. Thesis California Institute of Technology, Pasadena, USA, 1991.
- [16] Andrew W. Appel, Siam J. Sci. Stat. Comput., **6**, 85 (1985).
- [17] Feng Zhao and S. Lennart Johnsson, Siam J. Sci. Stat. Comput., **6**, 1420 (1991).