Comparison of Vector and Parallel Implementations of the Simulated Annealing Algorithm

J.M. Voogd*, P.M.A. Sloot*, R. v. Dantzig**

***Parallel Scientific Computing and Simulation group**
University of Amsterdam, Kruislaan 403
1098 SJ Amsterdam, The Netherlands
****NIKHEF, PO Box 41882**
1009 DB Amsterdam, The Netherlands

1 Introduction

The Parallel Scientific Computing and Simulation group at the University of Amsterdam is pursuing research in the field of parallel natural solvers. Natural solvers are algorithms that are inspired by processes from nature. In parallel computing especially the class of natural solvers provides a very promising approach, since the characteristics of the original physical phenomenon remain visible in the solving method and the implicit and explicit parallelism of the problem remains conserved. One of the natural solvers is an optimisation algorithm called Simulated Annealing (SA) which is the topic of this paper. Because of the inherent sequential nature of the algorithm, this particular method however, turns out to be hard to parallellize.

The SA algorithm is applied to a case study where simulation of crystallisation with spherical boundary conditions is studied. 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 implementation of SA on a super computer (CRAY Y-MP 4/464) with parallel implementations on a transputer platform (Parsytec GCel with 512 nodes). We have investigated the scalability of the parallel implementations with the number of particles (N) and the number of processors (P) and compared it with the scalability of the vector implementation.

In section 2 we explain the background of our study, in section 3 we discuss the algorithms and time complexities for the sequential, vector and parallel implementations. Section 4 contains the conclusions.

2 Background

Particle dynamics simulations with spherical boundary conditions for large numbers ($\geq 10^3$) of particles with Lennard-Jones or similar interactions at high density, provide an important testing ground for the study of closed 2D systems. Crystallisation with this type of constraints is poorly understood. As a model for such (bio)physically relevant systems, we started with particles, e.g. molecules, confined to a spherical surface. Examples of actual systems are buckyballs, viruses and membrane vesicles. Particle simulations on a spherical surface are also an alternative to simulations with periodic boundary conditions to approximate bulk systems in

a non-solid state. Although the spherical topology has only a limited effect on the properties of such bulk systems, they do affect the properties of the crystalline state in an essential way. Particularly the short range and long range order as a function of curvature and aggregation number and the tendency for hierarchical clustering of defects in a spherical matrix allow challenging ‘close packing’ studies [1].

Many problems in physics, chemistry and mathematics, like our case study, can be formulated as a global multidimensional optimisation problem. A vast majority of these problems involve the determination of the absolute minimum of an underlying cost-function. Usually optimisation of these complex systems is far from trivial since the solution must be attained from a very large and 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, approximation methods are required.

The Simulated Annealing method is a stochastic optimisation procedure that mimics the essentials of physical (thermodynamic) annealing and which is therefore closely related to the crystallisation phenomena we are studying.

In physical annealing a material is heated to a high temperature and then allowed to cool slowly. At high temperature the molecules move freely with respect to one another. As the liquid is cooled slowly, the molecules search for the lowest energy, consistent with the physical constraints. In this way, for example, large single crystals are formed in nature and in industry.

In simulated annealing the system is initialised in a highly disordered state at a high temperature and then allowed to equilibrate. The temperature is lowered in small steps while maintaining equilibrium. At high temperature the system can explore most of the phase space. During cooling the system will spend more time in minima from which it can still escape; molecules move relatively freely with respect to each other. When thermal mobility is lost, all molecules tend to search for the lowest energy, consistent with the global physical and topological constraints.

Although SA can guarantee, in principle, to find the global minimum, the time required for the algorithm to converge increases rapidly with the number of particles and/or local minima. In the crystallisation problem we are dealing with a large number of particles and the number of local minima (quasi-stable particle configurations) can also be large. Therefore conventional sequential annealing implementations are too slow, and more efficient methods need to be investigated. A standard method to reduce the computational time is to use vector super computers. But also with the new breed of parallel machines and programming paradigms, fast implementations come within reach, which however require tests and specific development [2].

3 Functional aspects and time complexities of simulated annealing

3.1 The sequential version

The SA algorithm for solving combinatorial optimisation 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 large systems of interacting particles at temperature T. For the crystallisation problem at hand the procedure works as follows. First, N particles are randomly placed on a supporting sphere. The annealing begins by creating a Markov chain of given length at a high temperature. The Markov chain grows by randomly displacing particles and calculating the corresponding change in energy of the system, while deciding on acceptance of the displacement. The length of the chain is made sufficient for the system to equilibrate.

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

$$P(\Delta E, T) = \exp(-\Delta E / T) \qquad \text{if } \Delta E > 0 \qquad (1)$$
$$P(\Delta E, T) = 1 \qquad \text{if } \Delta E \leq 0$$

This choice of $P(\Delta E,T)$ guarantees that the system asymptotically reaches a Boltzmann distribution [4].

After a certain number of steps the radius is optimised in a similar way as the particles. This is done by calculating the energy of the system at a new radius, randomly generated near the current value of the radius, and subtracting the current energy. Acceptance is also decided according to the probability scheme given above.

After a chain has ended, and the system is in equilibrium, the temperature is lowered. For the sequential and vector version we use a fixed cooling schedule. For the parallel implementation we use a fixed and a dynamic cooling schedule. In the fixed cooling case the temperature is multiplied by the cool-rate, which is a number slightly less than unity (for example 0.9), to obtain the new temperature of the system. The dynamic temperature decrement is only used in the parallel implementation (see section 3.3). After the temperature decrease a new chain is started. This process continues until a stop criterion is met, i.e. when the standard deviation in the final energies of the last ten chains falls below a certain value (for example 10^{-8}). The energy of the system is defined as the average energy per particle. This removes the main dependency of the energy on the number of particles such that the stop criterion can be fixed. In the SA procedure the calculations of potential energies for the particle and radius steps are the most time consuming parts.

Usually the SA method is applied to combinatorial optimisation with discrete perturbation steps. However, in our research we are dealing with particles on a continuous surface and thus we need a continuous algorithm. We have developed the following method to apply the SA in our studies. 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 variable width. The Cauchy distribution makes large perturbations (and thus escapes from local minima) probable. The width of the Cauchy distribution is dynamically changed such that about half of the particle moves are accepted.

Time complexity of the sequential algorithm

Each step in a Markov chain consists of a particle move and the calculation of the energy difference, with an update if the move is accepted. The execution time needed for a particle move and update is constant, that of the energy calculation is proportional to N. The radius is

only perturbed and updated every N steps. Since the calculation of the energy for the radius perturbation is of order N^2, while it is performed once every N steps, the contribution to the time complexity is of order N. One step in the Markov chain has complexity

$$T_1^{seq} = c_1 + c_2 * N \qquad (2)$$

where c_1 and c_2 are constants.

We have to multiply the time for one step with the number of steps in one chain and the number of chains generated to reach a stable minimum to get the time complexity of the complete annealing algorithm :

$$T^{seq} = T_1^{seq} * L * M \qquad (3)$$

where L=L(N) is the length of the Markov chain, and M the number of chains generated during the annealing process.

3.2 The vector version

The target machine, the CRAY Y-MP 4/464, is used for vector processing. We have not exploited the parallelism provided by the 4 processors. The algorithm for the vector version is largely the same as for the sequential version above. Some changes are made to improve the vector processing. 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, which are the most time consuming parts in the sequential version.

The time complexity of the vector version

It is hard to find the time complexity of the vector version by studying the code of the program. Many important contributions, like cache performance and memory bottlenecks, can not be modelled easily although they do have a large effect on the execution time (see [5]). The constants that appear in the time complexity of the sequential version are measured for the vector implementation. The time complexity of the complete vector version is given by (in μs units) :

$$T^{vec}(N) = T_1^{vec}(N) * L * M = (147.2 + 0.133 * N) * L * M \qquad (4)$$

We have made a comparison of the experimental timings of the complete program running on the CRAY Y-MP and the values following from equation (4). The results are given in the following table:

N	L	M	$T^{vec}_{timed}(N)$ (sec)	$T^{vec}_{theoretical}(N)$ (sec)
50	3000	393	156	181
100	11000	407	652	717

Table 1: A comparison of the theoretical and measured execution times on the CRAY Y-MP of the vectorised SA-program.

The data indicate that the predicted results are in reasonable correspondence with the timed values.

3.3 The parallel version

The parallel version involves two kinds of decompositions. The first is a systolic [6] decomposition of the Markov chains, the second a functional decomposition of the energy calculation in each step in the Markov chain.

In the systolic case a Markov chain is assigned to each of the available processors. All chains have equal length but correspond to different temperatures. The chains are executed in parallel and during execution information is transferred from a given chain to the successing chain in the cooling schedule. Each Markov chain is divided into sub-chains. The execution of chain k+1 is started as soon as the first sub-chain of chain k is completed. Equilibrium is not yet established at this point. By allowing a sub-chain to choose between the configuration of the previous Markov chain and its own configuration at the end of the previous sub-chain (see Fig. 1), quasi-equilibrium of the system is preserved. The choice is made according to eq 1.

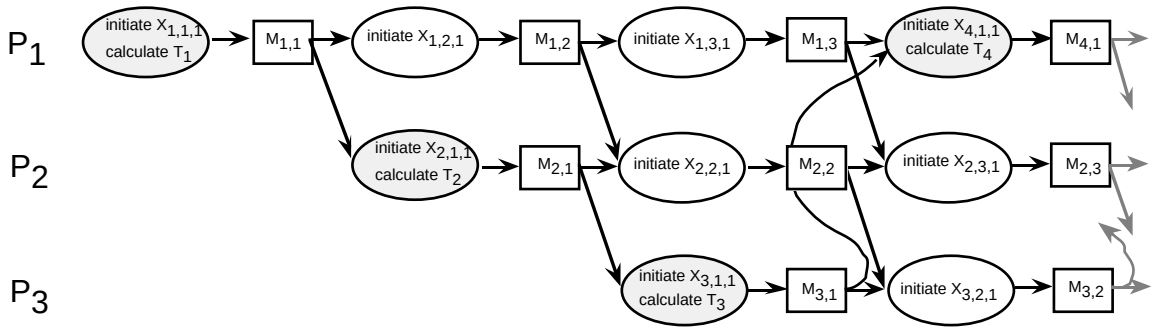

Figure 1: Diagram of a three processor implementation of systolic simulated annealing.

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 a functional decomposition [7]. A farm of processors is connected to a master processor that generates Markov chains which can assign the calculation of the energy difference to the farm.

The systolic algorithm has a simple communication pattern suited for implementation in a ring topology. The communication overhead is small since each processor contains a complete independent set of coordinates for the optimisation problem. To interchange information about the intermediate state it only has to send and receive at the end of each sub-chain. This SIMD scheme can be implemented efficiently on a MIMD architecture. If we use a hybrid

implementation, systolic SA with energy calculations in a farm, we need a farm attached to every processor in the systolic decomposition, see Fig. 2. The farm processors are connected as a tree.

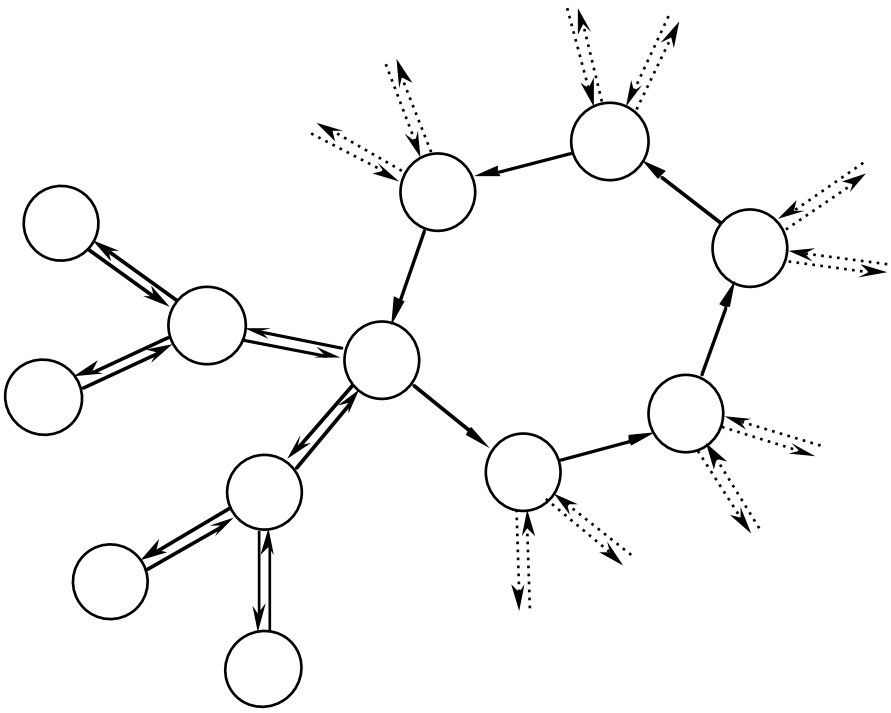

Figure 2: Diagram of hybrid implementation. Ring processors perform the annealing process, tree-slave processors (7 in our implementation) perform energy function calculations.

The time complexity of the parallel version

For the systolic implementation one step in the Markov chain has the same complexity as the sequential version. With functional decomposition the time complexity has to be adapted to account for the energy calculations in a tree of processors. The time complexity for the systolic implementation is different from the sequential time complexity because of the parallel execution of the Markov chains. Between the sub-chains there is communication that takes some time. The communication time is short compared to the generation of the Markov chains. But since the processors have to wait for their predecessors, the actual waiting time takes longer than the communication time estimated from the length of the message and the communication speed. The number of Markov chains, M, that have to be generated before a stable configuration is reached should not change because of the parallel implementation. But the processors in the systolic ring have to be started one by one, so for the time complexity the number of chains is M+P-1.

We have determined the constants appearing in our estimation of the time complexity by measuring execution times of subroutines and communication primitives. For the systolic and the hybrid (systolic and functional decomposition) implementations we find (in μs units) :

$$T^{sys}(N) = \left(\left(1.4*10^3 + 45*N \right) * \frac{L(N,P)}{P} + 75*10^3 \right) * (M + P - 1) \qquad (5)$$

$$T^{hyb}(N) = \left(\left(2.4*10^3 + \frac{9.6*10^2}{N} + 6.4*N \right) * \frac{L(N,P)}{P} + 75*10^3 \right) * (M + P - 1) \qquad (6)$$

Results for the parallel version

For N=50 and N=100, we find good agreement (see Fig 3) with the predicted execution times (eq. 6). The experiments show that the number of chains M increases with the number of processors in the systolic ring.

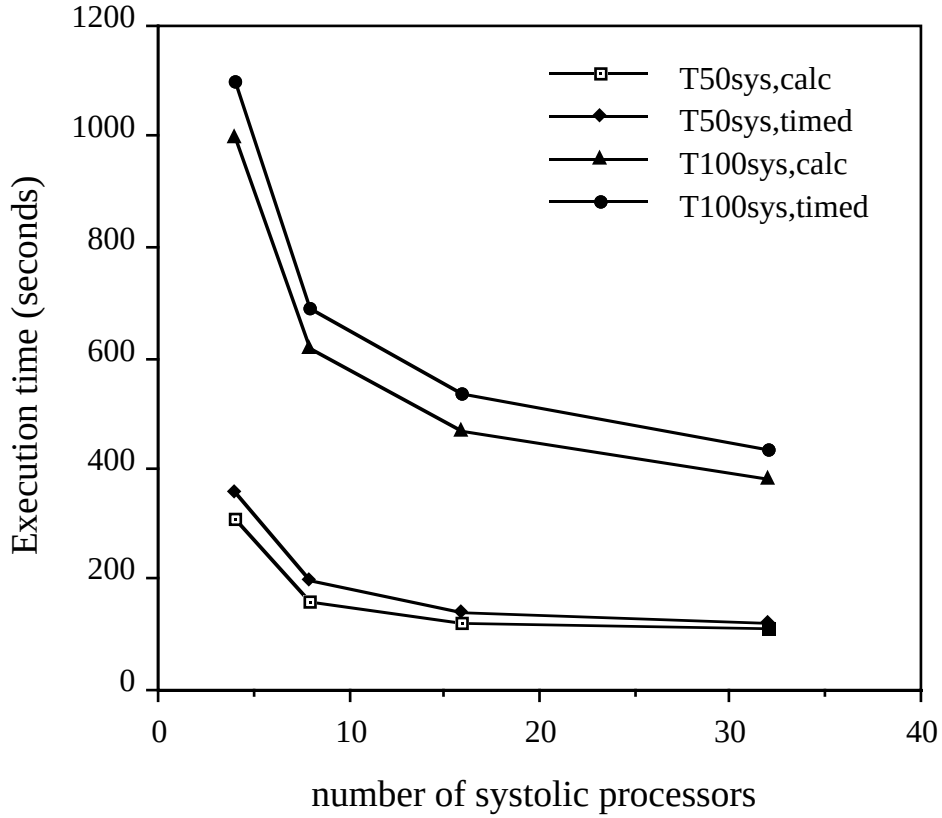

Figure 3: The calculated and measured execution times for N=50 and N=100 of systolic SA.

The SA algorithm is an inherently sequential scheme. Parallellizing it by the systolic SA method we introduce a functional difference with the sequential version. This parallelization is detrimental to the accuracy of the iterative processes so that more chains have to be generated. This counteracts the speed increase due to the parallelism. Also, the quality of the solutions decreases as the number of processors increases. It turns out that the fraction of configurations, adopted from a previous chain is very low (a few percent). Thus the configuration at the end of the first sub-chain is likely to be the only one used for the next chain. If more systolic processors are used, the sub-chains get smaller and consequently the system is further from

equilibrium if the next chain is started. We can balance the effects of increasing number of chains and decreasing quality by using a larger chain length than in the sequential case. So we have to find the relation L(N,P).

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 :

P	**L**
4	90
8	140
16	275
32	700

Table 2: Chain length as a function of the number of systolic processors for N=100.

The execution times for these new chain lengths are such that a large part of the speedup is lost. Here we have used the fixed cooling schedule (see below), the results are plotted in Fig 5.

In sequential annealing the chain length and number of chains are dependent on the cooling schedule. Until now we have used a fixed cooling schedule where the temperature of the next chain is given by the temperature of the previous chain as

$$T_k = \text{cool-rate} * T_{k-1} . \qquad (7)$$

We can also try a different cooling schedule, based on extra information of the current state of the system, and study its influence on the behaviour of the systolic implementation. The cooling schedule that we have tried is given by the formula

$$T_k = T_{k-1} * \left(1 + \frac{\ln(1+\delta) * T_{k-1}}{3 * \sigma(T_{k-1})} \right). \qquad (8)$$

This adaptive rule is based on the principle that the stationary distribution of states of two successive chains should be close to each other. This strategy uses additional information about the previous chain to determine the temperature of the new chain. This additional information comes from $\sigma(T_{k-1})$ which is the standard deviation in the energy values of the previous chain. The parameter δ controls how much the stationary distributions of the successive chains differ; it has a small value (0.1) which is kept constant for the experiments we present here. It is observed for this dynamic cooling schedule that the quality is not decreasing with increasing number of chains. This is because the configuration that is adopted from the previous chains is not in equilibrium yet. Therefore the deviation in the energy is high, which has as a consequence that the temperature is only slightly lowered. This means however that the number of Markov chains needed is increasing with increasing number of systolic processors. The speed-up gained by parallellizing the code is lost for larger numbers of systolic processors.

In order to give a fair comparison between the two cooling strategies we have looked for the chain lengths that gave the same quality of solutions. For the fixed cooling schedule this means that we have used a larger chain length for larger numbers of processors and for the dynamically adjusted temperature we could in some cases use lower chain lengths with

increasing number of processors. In Fig. 4 we give the execution times for both strategies for N=20. We can see that the execution times for both cooling schedules increase after a certain number of systolic processors. In the fixed cooling case the execution time increases because of the longer chain length that have to be used. In the dynamically adjusted temperature case the increase is due to the larger number of Markov chains that have to be generated before a stable configuration is found.

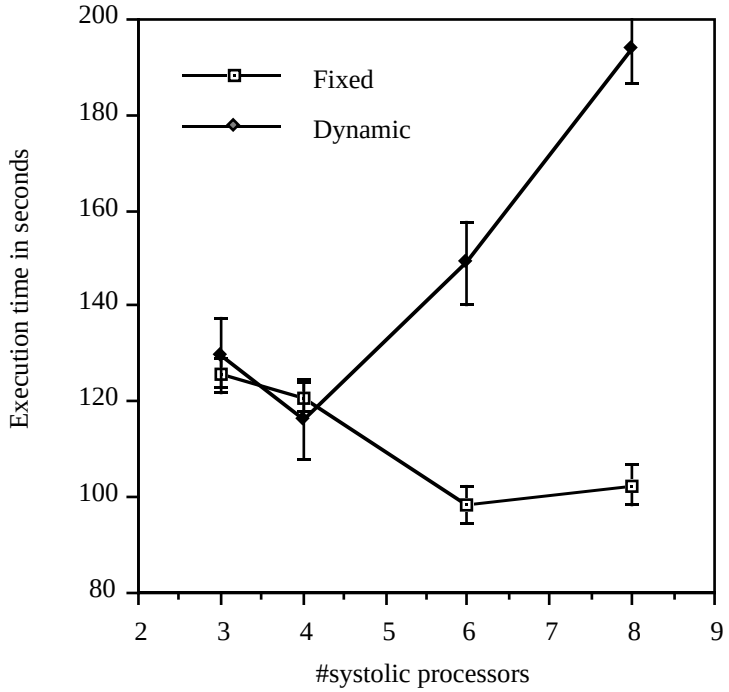

Figure 4: Execution times of the parallel implementation with different cooling schedules for N=20.

In Fig. 4 we show that, for the example N=20, the dynamic adjustment has a minimum in the execution time at 4 systolic processors, while the fixed temperature adjustment has its minimum at 6 processors. The execution time that can be achieved is lowest for the fixed temperature adjustment strategy.

4 Conclusions and discussion

Vector computing can be very efficient for sequential programs with a large number of parallel executable equivalent algorithmic steps. However, it can not be adapted to a specific problem in a flexible way.

A massively parallel machine has a large degree of freedom in connectivity so that the (virtual) processor topology 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 SA scheme is inherently sequential. Parallellizing it by the systolic SA method, dividing Markov chains in sub-chains with communication between them, introduces a functional difference with the sequential version. This parallelization has a negative effect on the accuracy of the iterative processes. We observe that the quality of the solutions tends to be influenced by

the number of processors used in the systolic algorithm. For the two cooling rules which were used, we had to optimise chain lengths in order to ensure that the quality of the solutions is independent of the number of processors.

The descriptions of the time complexities have been compared to the execution times found in 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 our test computations so that we can use this model for predictions.

Comparing the obtained results for the two cooling schedules, we find that the fixed cooling rule has its minimum in the execution time at a larger number of processors than the dynamic temperature adjustment. Therefore the fixed rule implementation gives shorter execution times.

Next we compare the time complexities of the vector version with the best performing parallel version. The results are given in Fig. 5 for N=100.

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 Fig. 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 hybrid version. This means that a factor 2 is lost on communication and on the decrease of precision in the iterative scheme of the parallel version.

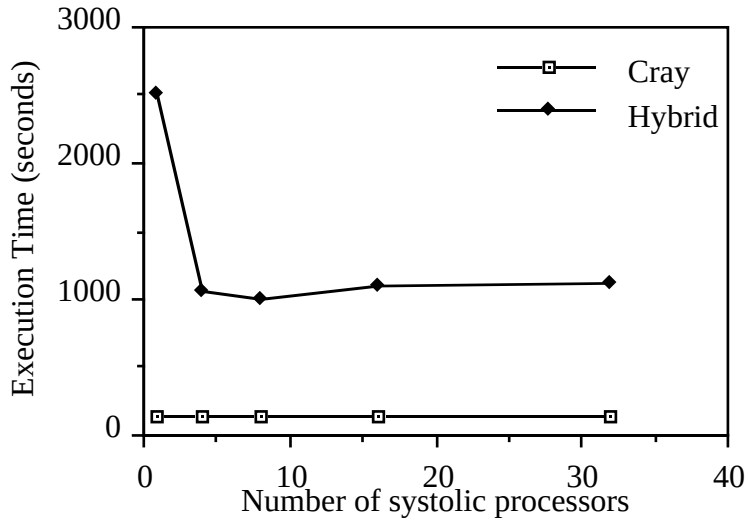

Figure 5: The execution times of a N=100 simulation for the vector implementation on the CRAY (horizontal line) is compared with the execution time of the hybrid implementation as a function of the number of processors. The total number of processors is 7 times the number of systolic processors.

Acknowledgements

The authors wish to thank Prof. D. Frenkel (AMOLF) for fruitful discussions on the physics of the experiments.

The research was supported by the 'Stichting Fundamenteel Onderzoek der Materie' (FOM) under number FI-A-a-3640.

Reference List

[1] J.M. Voogd, P.M.A. Sloot, R. v. Dantzig, *Two-dimensional crystallisation on spherical surfaces*, Proceedings of the 6th joint EPS-APS international conference on physics computing, eds R. Gruber, M. Tomassini, Lugano Switzerland (Aug. 1994), pp. 463-466

[2] P.M.A. Sloot, J.M. Voogd, D. de Kanter, L.O. Hertzberger, *Simulated annealing: Comparison of vector and parallel implementations*, Technical report CS-93-06, Dept. of Computer systems, University of Amsterdam (Oct. 1993)

[3] S. Kirkpatrick, C.D. Gelatt, Jr., M.P. Vechi, *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] W. Schönauer and H. Häfner, *Explaining the gap between theoretical peak performance and real performance for supercomputer architectures*, Scientific Programming, **3**, pp 157-168, 1994

[6] 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

[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