Parallel SPH on Cray T3E and NEC SX-4 using DTS*

T. Bubeck[1], M. Hipp[1], S. Hüttemann[1], S. Kunze[2], M. Ritt[1], W. Rosenstiel[1], H. Ruder[2], and R. Speith[2]

[1] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
[2] Institut für Astronomie und Astrophysik, Universität Tübingen

Abstract. In this paper we report on the results of a joint effort of astrophysicists and computer scientists to develop and implement a parallel program that enables us to solve large systems of hydrodynamic equations and covers a wide range of applications in astrophysics. We introduce the *Distributed Threads System* (DTS) as an environment for the development of portable parallel applications. The numerical method *Smoothed Particle Hydrodynamics* (SPH) is used to simulate the viscous spreading of an accretion disk around a massive compact object as an astrophysical test problem. The SPH code was parallelized using DTS and successfully ported to systems of different architecture. The use of a parallel SPH code on supercomputers enables us to treat astrophysical systems that were not accessible before. The achieved speedup proves the efficiency of DTS as a parallel programming environment. The physical results show the consistency and accuracy of the SPH method.

1 Introduction

The numerical simulation of large physical systems is still a great challenge even for modern computers. Especially in astrophysics, where laboratory experiments are not available, computer simulations are often the only way to gain new insights, by verifying, improving, or excluding theoretical models based on observational data. In problems involving fluids, which are the major problems in astrophysics, usually large systems of coupled differential equations have to be solved. One suitable method is *Smoothed Particle Hydrodynamics* (SPH). In this study, SPH is used to simulate a gaseous disk around a compact star. Due to the nature of the problem, hardware and CPU time requirements are high and super-computing is a necessity. Usually, physicists are more involved in their physical problems than in parallel programming. Therefore, the parallel programming environment *Distributed Threads System* (DTS), developed and implemented by computer scientists, is a great help for physicists as it provides the means of easily porting their simulation codes to different parallel architectures and effectively uses all parallel features.

* This project is funded by the DFG within SFB 382: Verfahren und Algorithmen zur Simulation physikalischer Prozesse auf Höchstleistungsrechnern (Methods and algorithms to simulate physical processes on supercomputers).

In section 2 we give the motivation for using DTS and present some of its implementation details. We also describe the experiences made when porting DTS to the NEC SX-4. Section 3 reviews some fundamentals of SPH. In section 4 the parallelization of SPH is detailed. Finally, we present the results of this study in section 5 and give a conclusion in section 6.

2 Distributed Threads System

Rapidly changing parallel architectures and programming interfaces make it difficult to achieve on-the-edge performance for parallel applications. Either the code has to be adapted with much effort to the native programming model for every new architecture, or it is based upon a portable parallel programming interface, trading code reuse for efficiency. Because we are working on long term research projects, portability was one of our major goals.

Of course, there are already standardized and widespread interfaces like MPI [11] or PVM [7], which provide a portable basis for parallel programming. But the main disadvantage of the message passing systems is their low level approach: It is difficult to concentrate on core parallelization and resulting programs are often erroneous and hard to debug. Another goal in this context was the separation of tasks: Physicists should be able to concentrate on solving their problems, while computer science is to make the physicist's job as easy as possible.

Therefore, we decided to suggest DTS as a portable parallel programming environment. It includes a parallelizing compiler [14], a parallel runtime system [2,3] as well as evaluation tools [4]. The runtime system is based on a thread interface, which has been extended to distributed memory architectures. The thread interface was chosen because it is a well known programming interface and a suitable target for our parallelizing compiler. It provides enough abstraction from simple message passing to concentrate on parallel algorithms, which, in the present study, are mainly regular data-parallel or irregular control-flow-parallel (divide-and-conquer algorithms).

From the user's perspective, in most cases programming in DTS does not differ from programming with conventional threads. The current implementation provides a pure functional progamming model, which proved to be sufficient for our current needs. For algorithms with stronger data dependencies, an extension allowing for direct thread-to-thread communication via distributed shared memory methods is planned in the near future.

2.1 Implementation of DTS

The implementation of DTS comes in two flavors, a completely distributed managed version and one with a centralized manager process. Although the central manager becomes a bottleneck on large numbers of processors, it proved to be more efficient for small and medium processor numbers. Therefore, we used the centralized approach on the NEC SX-4.

Figure 1 shows the major steps in executing a thread in DTS. On one node, called the root node, a central manager administers and controls the creation and distribution of threads to other nodes in the virtual machine. Every client node runs a client manager, waiting for threads to be executed. If a new thread is created, the caller asks the central manager for an executor, and sends it the job. The results are directly passed back to the caller. The choice of a suitable machine for executing the thread (load balancing), sending the parameters to the executor, and collecting them after execution, as well as error recovery is done transparently by the different managers.

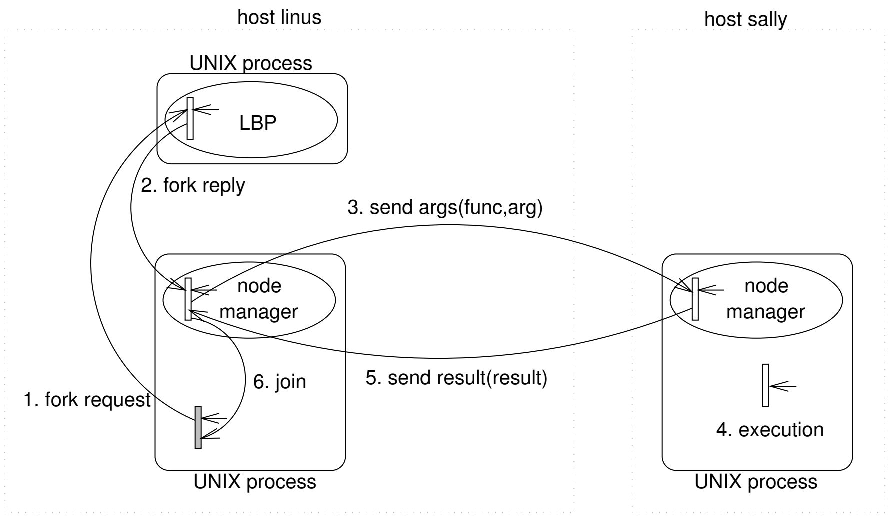

Fig. 1. Remote thread execution in DTS.

2.2 Porting DTS to the NEC SX-4

The DTS system is based upon threads and the message passing library PVM. Porting usually involves translating the machine independent thread interface layer of DTS into native thread calls only.

As the NEC SX-4 is a shared memory system, there was no need to use any kind of message passing. Therefore, we decided to eliminate the message passing layer completely. The PVM library was replaced by a version containing only empty function stubs, and the runtime system was modified to pass thread calls of DTS directly to the underlying thread layer. Actually, DTS has been reduced to a pure threads package. Measurements have shown, that the extra overhead of the DTS system using only threads is about 2%.

To realize the thread functionality, we chose to use the native implementation of POSIX threads. Since we already did a port of the DTS threads interface to another POSIX threads implementation, only minor changes concerning some missing functions which are optional in the POSIX standard, were required. Most problems we had to cope with had their origin in the POSIX-threads implementation of the NEC SX-4 itself[1] and were fixed by NEC.

Currently we are working towards a port for DTS to the Cray T3E. Additionally, the SPH code is parallelized using the native Cray SHMEM interface to serve as a base for quantifying the efficiency of DTS (see section 4).

3 Smoothed Particle Hydrodynamics

3.1 Basics

Smoothed Particle Hydrodynamics is a grid-free Lagrangian particle method for solving a system of hydrodynamic equations for compressible and viscous fluids. It was first introduced independently by Gingold & Monaghan [8] and Lucy [10]. SPH is especially suited for problems with high density contrasts and free boundaries. Rather than being solved on a grid, the equations are solved at the positions of the so-called particles, each of them representing a continuum with a certain mass, density, temperature, etc. and moving with the flow according to the equations of motion.

In contrast to most other flavors of SPH, here the viscous stress tensor is implemented to describe the physical viscosity correctly according to the Navier-Stokes equations. Usually only artificial viscosity is used, which is needed for the treatment of shocks, but vanishes in the continuum limit.

3.2 The Hydrodynamic Equations

The motion of the fluid is described by the Navier-Stokes equation, here in Lagrangian formulation:

$$\frac{dv_\alpha}{dt} = -\frac{1}{\varrho} p_{,\alpha} + \frac{1}{\varrho} t^{\mathrm{visc}}_{\alpha\beta,\beta} + f_\alpha \quad , \tag{1}$$

with f being the gravitational force of the central object and the viscous stress tensor

$$t^{\mathrm{visc}}_{\alpha\beta} = \eta \left(v_{\alpha,\beta} + v_{\beta,\alpha} - \tfrac{2}{3} \delta_{\alpha\beta}\, v_{\gamma,\gamma} \right) + \zeta\, \delta_{\alpha\beta}\, v_{\gamma,\gamma} \quad . \tag{2}$$

[1] First experiences have shown that the implementation was not POSIX conform, because a `pthread_detach` was required *after* joining the thread to free up internal thread administration structures. Without doing so the total number of executable fork/join-pairs was reduced to 512.

(Here all spatial derivatives $\partial/\partial x_\alpha$ are represented by $_{,\alpha}$ and Einstein's summing convention holds). The coefficients of shear and bulk viscosity, η and ζ, are positive scalars and independent of the velocity [9]. The term in parentheses is the shear, denoted by $\sigma_{\alpha\beta}$. Although the treatment of bulk viscosity is no principal problem we assume $\zeta = 0$. Furthermore, the kinematic viscosity coefficient $\nu = \eta/\varrho$ is assumed to be constant.

3.3 SPH Approximations

The purpose of the SPH method is to transform a system of coupled partial differential equations into a system of coupled ordinary differential equations, which can be solved by a standard integration algorithm.

The transformation is achieved by two steps of approximations. First, any variables are replaced according to the convolution

$$f(\boldsymbol{r}) \longrightarrow \int f(\boldsymbol{r}\,')\; W(|\boldsymbol{r} - \boldsymbol{r}\,'|, h)\; dV' \tag{3}$$

with an appropriate kernel W.

Secondly, the convolution integral is evaluated only at the positions of the particles, hence transformed to a sum:

$$f(\boldsymbol{r}_i) \approx f_i = \sum_j \frac{f(\boldsymbol{r}_j)}{n_j} W(|\boldsymbol{r}_i - \boldsymbol{r}_j|, h) \tag{4}$$

(n_j: Particle density at the point $\boldsymbol{r}_j$).

The spatial derivatives can now be transferred onto the kernel by partial integration:

$$\nabla f(\boldsymbol{r}_i) \approx \sum_j \frac{f(\boldsymbol{r}_j) + \tilde{f}(\boldsymbol{r}_i)}{n_j} \nabla W(|\boldsymbol{r}_i - \boldsymbol{r}_j|, h) \quad . \tag{5}$$

Since the derivatives of the kernel are known analytically, we now have the desired system of ordinary differential equations that can be integrated numerically.

3.4 SPH Formulation of the Equations

As an example, the application of the smoothing discretization scheme to the viscous part of the Navier-Stokes equation (1) yields

$$\begin{aligned} \left(\frac{dv_\alpha}{dt}\right)_i^{\text{visc}} &= \left(\frac{1}{\varrho} t_{\alpha\beta,\beta}\right)_i \qquad (6) \\ &= \sum_j m_j \left(\frac{\nu_j}{\varrho_i} \left(\sigma_{\alpha\beta}\right)_j + \frac{\nu_i}{\varrho_j} \left(\sigma_{\alpha\beta}\right)_i\right) (W_{,\beta})_{ij} \end{aligned}$$

with the particle form of the shear $\sigma_{\alpha\beta}$

$$(\sigma_{\alpha\beta})_i = (V_{\alpha\beta})_i + (V_{\beta\alpha})_i - \tfrac{2}{3}\delta_{\alpha\beta}(V_{\gamma\gamma})_i \quad , \tag{7}$$

where $(V_{\alpha\beta})_i$ is the particle representation of the velocity gradient $v_{\alpha,\beta}$

$$(V_{\alpha\beta})_i = (v_{\alpha,\beta})_i = \sum_j \frac{m_j}{\varrho_j} \left((v_\alpha)_j - (v_\alpha)_i\right)(W_{,\beta})_{ij} \quad . \tag{8}$$

All other equations (continuity equation, energy equation, state equation) can be formulated in SPH in a similar way.

4 Parallelizing SPH with DTS

First, we will describe our SPH implementation on NEC SX-4 using DTS. Second, we will discuss SPH codes for machines with distributed memory.

There are several papers on parallel SPH for machines such as Cray T3D, CM-5 or Intel Paragon [13,6]. The method introduced in [5] is compared with our parallel SPH code for machines with distributed memory in section 4.3.

4.1 Shared Memory Machines

The basic idea is to find data structures to allow independent parallel threads and to avoid critical sections as much as possible. There are two main problems involved with a SPH algorithm. First, the nearest neighbor search and second, the evaluation of the list of neighbors to calculate the physical quantities for each particle.

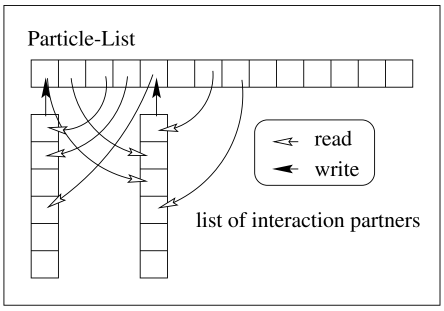

Fig. 2. Data structure of parallel SPH for shared memory machines (the list of particles and the corresponding list of interaction partners).

Concerning the nearest neighbor search, using the linked list algorithm of [1] we were able to develop a parallel algorithm with a parallel efficiency of at least 90% on NEC SX-4 using 20 CPUs (see figure 3). The search for nearest neighbors is divided into a sequential and a parallel part. In the sequential part the particles are projected on a grid which is used to build the linked list. This is an algorithm of order $O(N)$, where N is the number of particles. Since the linked list is only read in the process of finding the interaction partner for each particle, the nearest neighbor search can be done in parallel.

For the evaluation of physical quantities every particle has a complete list of all its neighbors (figure 2). To compute a physical quantity for a certain particle, the information stored in the neighbor list is read only. Thus, a given physical quantity can be evaluated in parallel for each particle. By storing a complete list of all neighbors per particle, the physical symmetry relations between the particles cannot be exploited by the algorithm. This makes the parallel code at most two times slower than an optimized sequential code (see section 5.6).

4.2 SPH on the NEC SX-4 using DTS

The SPH code was implemented on the NEC SX-4 using DTS. The usage of DTS makes it possible to run the same SPH code both on the NEC SX-4 and on other machines[2], just by recompiling with `dtscc`. The parallel SPH code was used for benchmarking, using different numbers of CPUs. The results prove the high quality of the parallelization features of the NEC SX-4.

Figure 3 shows the speedup and the parallel efficiency of the parallel SPH code on the NEC SX-4. For 10 000 particles the parallel efficiency decreases

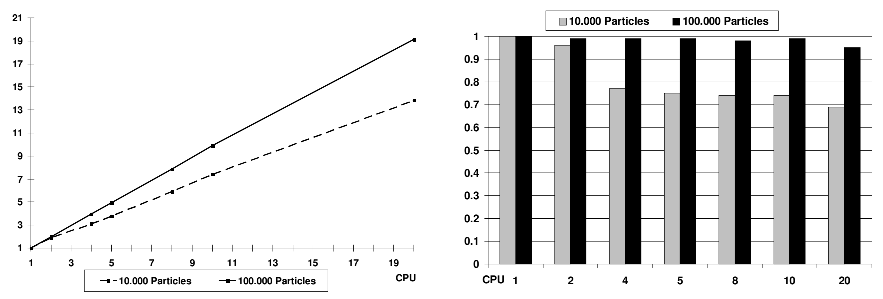

Fig. 3. *Left:* speedup of parallel SPH. *Right:* parallel efficiency on NEC SX-4.

from 90% on two CPUs to 60% on 20 CPUs. For 100 000 particles the parallel efficency is more than 90% for all 20 CPUs.

[2] e.g. Ross Hyper Sparc, Sun MP, SGI Onyx2

4.3 Parallel SPH Codes on Distributed Memory Architectures

Besides the DTS SPH code for shared memory machines, there are a few implementations for SPH on parallel machines. Three of them developed especially for machines with distributed memory are the PTreeSPH [5], a port for the MEMSY architecture developed in Erlangen [12], and our own implementation for the Cray T3E. All of them handle the communication in a different way.

The PTreeSPH code is based on MPI and, therefore, is portable to nearly every platform. A general disadvantage of MPI is a loss in performance due to the complex communication protocol. Thus, good domain decomposition and, even more important, smart communication are crucial. The solution was to build a so-called *Locally Essential Particle List* (LEPL). In a synchronous preprocessing step, all nodes check which of their particles interact with particles from a neighbor domain and put them into the LEPL. Further communication is necessary only for those particles. Domains are separated in equally sized parts with an *orthogonal recursive bisection tree.*

On the MEMSY architecture, communication is possible only between neighbor nodes in the squared processor plane. The N particles are combined with every other particle to $N \times N$ interaction pairs. This set of pairs is divided by the number of nodes and distributed over the plane. Figure 4 shows an example. Every node computes a partial sum over its particle pairs. The total sums are computed by sending the partial sums of each node line step by step to the diagonal node of the plane. From there the total sum is sent back to all nodes in the same node line and column. We tested a simple port

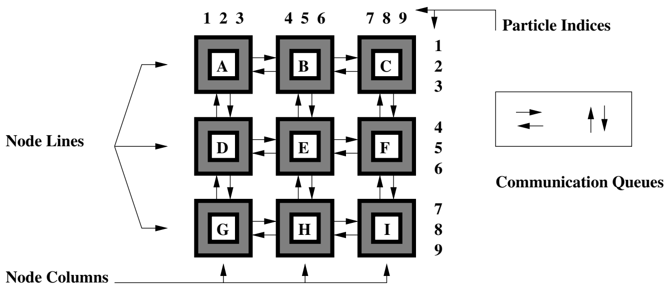

Fig. 4. Example for the distribution of 9 particles 1-9 over 9 nodes A-I. For example node B computes interaction between particles 1-3 and 4-6.

of this implementation on the T3E by simulating the communication queues between two neighbor nodes. It shows that the main disadvantage is that the algorithm to check whether two particles interact scales with $O(N^2)$, where N is the particle number.

4.4 Parallel SPH on Cray T3E

Since DTS has not yet been ported to the Cray T3E, we decided to implement SPH using the Cray T3E SHMEM library, which provides functions oriented at the Cray T3E hardware capabilities. The main reason was to get an optimal implementation by using all possible features the Cray T3E offers, which can serve as a reference implementation for quantifying the efficiency of a later DTS port. Another reason was that we already had a shared memory implementation. A message passing based version would have required a major redesign. The SHMEM library allowed us to reuse as much as possible of the original code.

An essential idea was to use two different domain decompositions depending on the type of computation:

1. All computations without neighbor interaction are done on an equally sized subset on every node. The subset is selected by splitting the particle field into n parts for n nodes. A node also operates as a *relay* node for its subset. Information about a specific particle can always be found on its *relay* node.
2. For computations with neighbor interaction all particles are sorted according to their positions into a grid with equally sized cells. These cells are assigned to nodes in a way that every node holds the same number of particles.

The load balancing is good in both cases, because the computation takes about the same time for every particle except the neighbor search.

The neighbor search uses the grid from the second domain decomposition. The problem here is, that it is not known *a priori* how many comparisons are necessary to find the nearest neighbors of every particle. Hence we added the possibility that one node helps another to find its neighbors. In this step the ability of the Cray T3E to read and write remote memory asynchronously is used. The neighbor indices are stored in a list for later reuse.

Both the neighbor list and two lists similar to the LEPL described in the PTreeSPH section are built simultaneously. Because there are two domain decompositions we also need two lists, a *get* list and a *put* list. The *get* list holds the indices of particles which are not assigned to a node by one of the two domain decompositions but are neighbors of a particle in the second domain decomposition. The *put* list holds the list of particles which are assigned to a node by the second domain decomposition, but not by the first.

It is not possible to decide which node needs which particle positions because of the neighbor search. So we decided to distribute the positions of all particles to all nodes using a broadcast.

To compute a physical quantity which is independent of neighbor interaction, no communication is needed. Every node does this work on its *relay* particles.

The first evaluated quantity with neighbor interaction is the kernel. We can compute it with no further communication, because we still have distributed the particle position. To distribute the computed kernel every node first uses the *put* list to send the kernel back to the *relay* node and then uses the *get* list to get the kernel of all other necessary particles. The same is done for the mass density and all other physical quantities.

The final steps are the computation of the gravitational force and the integration timestep itself. This again requires no neighbor interaction and every node does it on its *relay* particles.

For most physical quantities an array is allocated with enough space to theoretically hold every particle on a single node. On the Cray T3E with 128MB local memory, this currently limits the maximum number of particles to about 750 000 for the 2D case when using double precision floating point. For single precision the maximum number of particles doubles.

5 Simulations and Results

5.1 The Test Problem

The test problem is a thin accretion disk around a compact central mass M. 'Thin' means the size of the disk perpendicular to the disk plane can be neglected. The following approximations are made: $v_z \approx 0$, $z^2 \ll x^2, y^2$.

All variables are integrated over the height of the disk, e.g. the surface density is given by $\Sigma = \int \varrho dz$. The motion of the gas in the disk is described by the Navier-Stokes equation (1), as pointed out in section 3. With the approximations made above, in 2 D form the equation reads

$$\Sigma \frac{dv_\alpha}{dt} = -p_{,\alpha} + t^{\text{visc}}_{\alpha\beta,\beta} \qquad (\alpha, \beta = 1, 2)\,. \tag{9}$$

In order to obtain an analytic solution, we transform to polar coordinates (r, φ) and assume rotational symmetry. Furthermore, pressure forces are neglected, and $v_r \ll v_\varphi$ is assumed. With the constant viscosity coefficient ν and the initial density profile

$$\Sigma_0 = \frac{m}{2\pi r_0} \delta(r - r_0) \tag{10}$$

we get the following analytic solution for the surface density:

$$\Sigma(r, t) = \frac{m}{\pi r_0^2 \tau} \left(\frac{r}{r_0}\right)^{-\frac{1}{4}} e^{-\frac{r_0^2 + r^2}{r_0^2 \tau}} I_{\frac{1}{4}} \left(\frac{2r}{r_0 \tau}\right), \tag{11}$$

Here m denotes the mass of the disk, $I_{\frac{1}{4}}$ is the modified Bessel function to the base $\frac{1}{4}$, and $\tau = 12\nu t / r_0^2$ is the dimensionless 'viscous time'.

Now we have the analytic radial density profile of the disk, which can be compared with the results of the simulations.

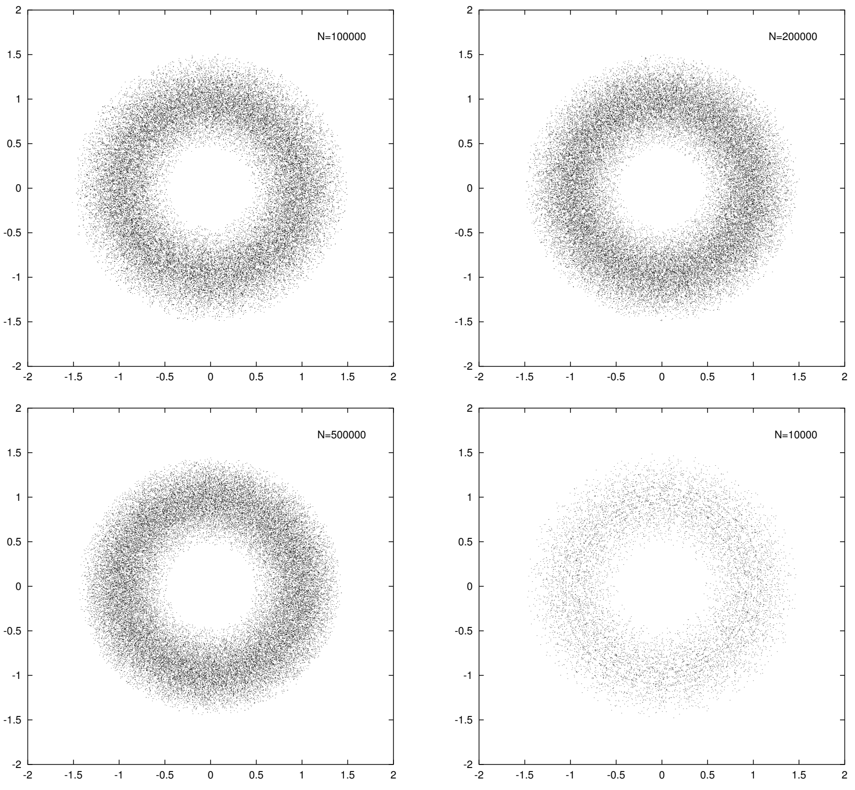

Fig. 5. Positions of particles at $\tau = 0.09$. Upper left: 100 000 particles, upper right: 200 000 particles, lower left: 500 000 particles, lower right: for comparison: simulation with 10 000 particles with a serial code on a workstation. Note the lack of spiral-like structures in the 200 000 and 500 000 particle simulations.

5.2 Simulations of the Test Problem

Numerical simulations of the test problem were performed both on workstations and on the NEC SX-4 at the HLRS. The SPH forms of the 2 D cartesian hydrodynamic equations were used, neglecting pressure forces. The physical parameters for all simulations were ($\odot$ denotes solar units): mass of central object $M = 1M_\odot$, total mass of the disk $m = 10^{-10}M_\odot$, initial radial distance of the disk from the center $r_0 = 1R_\odot$, and the coefficient of viscosity: $\nu = 3 \cdot 10^{-8}R_\odot^2/s$.

Initial particle distribution were produced, according to the density profile at viscous time $\tau = 0.018$, but with the individual particles being placed stochastically on Kepler orbits.

Simulations with 100 000, 200 000, and 500 000 particles were made, using the parallel queues NP2GB8CPU and NP4GB16CPU. Since most people

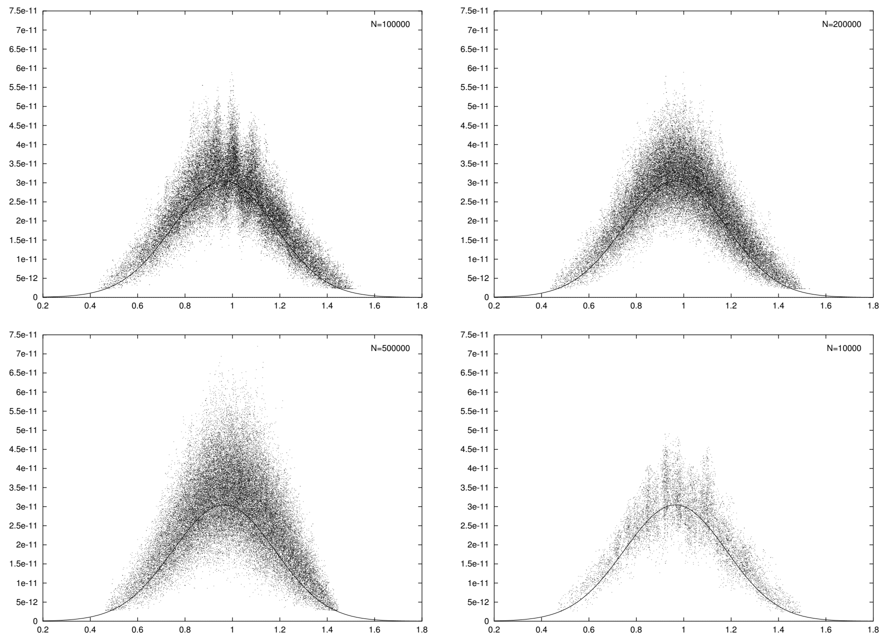

Fig. 6. Surface density of particles at $\tau = 0.09$, only radial coordinate plotted. Arrangement as in figure 5. Underlying is the analytic solution. Note the mean deviation from the analytic value does not decrease with particle number, because the mean number of interaction partners for each particle is the same in all simulations.

seem to use the NEC SX-4 in single processor mode, the jobs started quickly, once submitted to the parallel queues. Output files were produced at viscous times 0.036, 0.054, 0.072, 0.090, and 0.126.

In figures 5 and 6 we show the positions of the particles and the radially plotted surface density $\Sigma(r)$ at viscous time 0.09. Furthermore, the analytic solution of the surface density is shown in figure 6. Also shown are the results of a simulation with 10 000 particles that was carried out on a SGI workstation with a 100MHz R4000 processor (figures 5 and 6 lower right corner). For clarity, from the HLRS simulations only 50 000 randomly chosen particles are plotted.

5.3 Physical and Numerical Accuracy

We know from earlier simulations that the most important numerical parameter is the number of interaction partners per particle. In this implementation of SPH the smoothing length, which corresponds to the interaction radius of a particle, is kept constant throughout a simulation. In order to give each particle the same average number of interaction partners in all simulations,

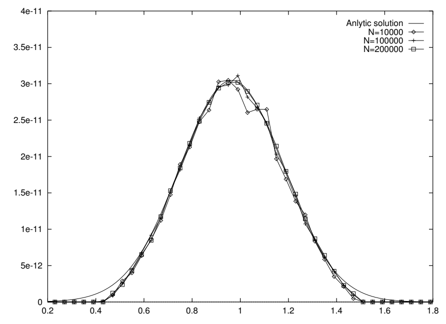

Fig. 7. Averaged radial surface density distribution of the particles. The simulation with 10 000 particles shows some deviation from the analytic value, whereas the simulations with more particles are practically identical.

the smoothing length was scaled with the inverse square root of the total particle number.

With this setup we expect the accuracy to be very similar for all simulations. This can be seen in the distribution of the surface densities in figure 6. The deviation from the mean value is very similar in the simulations with 10 000, 100 000, and 200 000 particles. All these simulations have been carried out with the same error tolerance of the time integrator. In the simulation with 500 000 particles a higher error was tolerated, obviously too high, since the deviation in the surface density distribution is clearly too large. This is not to be considered as a failure, because we also wanted to determine the numerical parameter 'accuracy of integrator'.

Although there is a large scatter about the analytical value of the surface density, the *mean* density is conserved to very high accuracy in the simulations. This can be seen in figure 7, where the surface density of the particles is averaged in 50 bins between radial coordinates 0.05 and 2.05. (Because of the insufficient integrator accuracy the simulation with 500 000 particles is excluded.) The results from the NEC simulations are practically identical, whereas the simulation with only 10 000 particles shows some deviation from the analytical value. This is mainly due to the spiral structures, which are of mainly numerical origin. This can be seen from the fact that they are strongest for smallest particle number and not present at all in the simulations with 200 000 and 500 000 particles.

5.4 Speedup of SPH on Cray T3E

We measured the speedup of SPH simulations of the test problem (section 5.1) with 10 000 and 100 000 particles. The speedup is quite satisfactory, see figure 8. One can also see that there is a problem in the SHMEM communication when the number of nodes is not a power of two (here with 96 and 384 nodes).

The 100 000 particle simulation has a higher efficiency. This is because in the nearest neighbor search, the computation cost rise faster than the communication cost with increasing particle number. Since the computation has a higher parallel efficiency, the total efficiency is better for more particles.

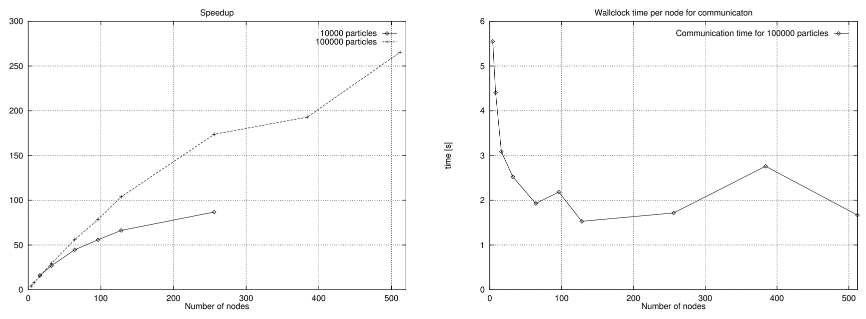

Fig. 8. Speedup for the T3E implementation with 10 000 and 100 000 and time for communication for 100 000 particles. No values were measured for 10 000 particles on 384 and 512 nodes. The communication time is the time for internode data transfers and synchronization including data preprocessing necessary for communication.

5.5 Comparison of Parallel SPH Methods for the Cray T3E

The PTreeSPH implementation reaches a maximum speedup of 27 at 64 nodes for 256 000 particles while our SPH implementation gains a speedup of 260 for 512 nodes with only 100 000 particles. We expect even better results for higher particle numbers. The ported version of the MEMSY code cannot keep up with this result. On top of higher total computation time, test runs showed that the queued communication is quite expensive. In a test simulation with 20 000 particles the contribution of the communication to the total computation time rised from 17% on 4 nodes to 76% on 256 nodes.

5.6 Vectorization and F77 solution on NEC SX-4

Apart from good speedup and scalability, it is of course also desirable to achieve a considerable runtime improvement compared to an optimized sequential code. Further runtime improvement can be reached by vectorization of the code. In figure 9 we present a comparison of the runtimes of SPH simulations of the same test problem with different codes. From this, and from figure 3 one can see that already using as less as 2 CPUs on the NEC SX-4 a runtime improvement compared to the optimal sequential SPH code can be achieved. As the speedup is excellent, very good runtime improvements can be obtained using more CPUs.

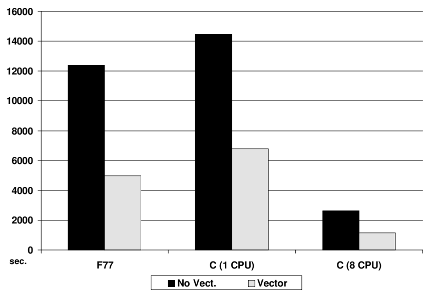

Fig. 9. Vectorization and F77 solution of a SPH simulation with 10 000 particles (runtime in seconds). *Left:* optimized F77 sequential code. *Middle:* parallel SPH on 1 CPU. *Right:* parallel SPH on 8 CPUs.

6 Conclusion

From the physical point of view the simulations on the NEC SX-4 were successful in different aspects. First, there is still no mathematical proof of the consistency of the SPH method. Nevertheless, it is widely, and often uncritically, used throughout the astrophysical community. Therefore, the confirmation of the consistency and accuracy of the method as shown by the results of the test simulations is a good justification for the usage of the method and lets us sleep with a good conscience. Furthermore, the possibility to treat large systems with high spatial resolution opens the door to new fields of research. In future, we want to use our code to simulate accretion disks in binary systems, where a gas stream from a secondary star hits the disk. We want to look in greater detail on the stream-disk impact region and on gas streaming over the disk. Now we have the means to tackle these problems, and will do so, computing time provided.

The major goals in providing our own parallel programming environment, namely the portability of code and a simplified parallel programming interface, have been reached. Besides running the same code without modification on all supported platforms, we gained much from a port of DTS to networks of workstations: We were able to develop and debug all of the codes in our local environment, saving time and computing cost. Only the final tests and modifications had to be done directly on the production platforms.

The efficiency and parallel speedup of applications based on DTS proved to be satisfying. The gain of portability outweights the loss of efficiency due to system overheads. The system will be ported to the Cray T3E in the near future.

We were able to show that parallel SPH codes with an efficiency of 90% and more on the NEC SX-4 are possible. We could also show that the Cray T3E's fast communication hardware together with an optimized communication protocol and domain decomposition allows a parallel SPH version with high efficiency on machines with distributed memory. Even for a small SPH problem (10 000 particles) we gain significant speedup reducing the computation time by a factor of 160.

All this was possible due to the fruitful collaboration of astrophysicists and computer scientists within the SFB 382 "Verfahren und Algorithmen zur Simulation physikalischer Prozesse auf Höchstleistungsrechnern" (Methods and algorithms to simulate physical processes on supercomputers).

References

1. Allen, M. P., Tildesdley, D. J.: *Computer Simulation of Liquids.* Oxford University Press (1992)
2. Bubeck T.: *Eine Systemumgebung zum verteilten funktionalen Rechnen.* Eberhard-Karls-Universität Tübingen, Technical Report WSI-93-8 (1993)
3. Bubeck, T.: *Distributed Threads System DTS User's Guide.* Universität Tübingen, SFB 382/C6 (1995)
4. Bubeck, T., Schreiner, J., Rosenstiel, W.: *Timing multi-threaded Message-Passing Programs.* Proceedings SIPAR Workshop 96 (1996) 15–18
5. Davé, R., Dubinski, J., Hernquist, L.: *Parallel TreeSPH.* New Astronomy volume **2** number **3** (1997) 277–297
6. Dubinski, J.: *A Parallel Tree Code.* Board of Studies in Astronomy and Astrophysics, University of California, Santa Cruz (1994)
7. Geist, A., Beguelin, A., Dongarra, J., Jian W.: *PVM 3 User's Guide and Reference Manual.* Oak Ridge National Laboratory, Tennessee (1994)
8. Gingold, R. A., Monaghan, J. J.: *Smoothed particle hydrodynamics: theory and application to non-spherical stars.* Mon. Not. R. astr. Soc. volume **181** (1977) 375–389
9. Landau, L. D., Lifshitz, E. M.: *Fluid Dynamics, 2nd ed.* Pergamon Press, Oxford (1987)
10. Lucy, Leon B.: *A Numerical Approach to Testing the Fission Hypothesis.* Astron. J volume **82** (1977) 1013–1924
11. Message Passing Interface Forum: *MPI: A Message-Passing Interface Standard.* Computer Science Department, University of Tennessee, Knoxville, TN Techn. Rep. CS-94-230 (1994)
12. Schneider, A.: *Meßstudie paralleler Simulationen von Akkretionsscheiben.* Universität Erlangen (1997)
13. Warren, S. Micheal, Salmon, K. John: *A portable parallel particle program.* Comp. Phys. Comm. volume **87** (1995) 266–290
14. Wedeck, J.: *Automatische Parallelisierung von sequentiellen Programmen unter besonderer Berücksichtigung von Hardware-Beschleunigern.* Universität Tübingen, Technische Informatik (1996)