Contents

1

1

Efficient and Object-Oriented Libraries for Particle Simulations

S. Ganzenmüller[1], M. Hipp[1], S. Kunze[2], S. Pinkenburg[1], M. Ritt[1], W. Rosenstiel[1], H. Ruder[2], and C. Schäfer[2]

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

Summary. We present two libraries for the parallel computation of particle simulations. One is the object-oriented library sph2000 written in C++, the other is ParaSPH, a library written in C, that supports hybrid architectures (clustered SMPs). They are portable and performant on a variety of parallel architectures with shared and distributed memory. We give details of the object-oriented design of sph2000, the parallelization of ParaSPH for hybrid architectures using MPI and OpenMP and discuss the speedups of the codes. Further, we give three examples of applications based on these libraries, which simulate protoplanetary discs, colliding rubber rings and the injection of diesel into a combustion chamber.

Keywords: Computational physics, Object-orientation, Parallelization, Simulation

1.1 Introduction

In the Sonderforschungsbereich 382 there are several particle codes for the simulation of astrophysical problems. In the last years there was a strong effort to develop fast parallel particle libraries to support these applications. However, there is still a need for larger simulations needing more memory and computing power. To meet this requirements we are continuously investigating to reduce the parallel overhead and the serial parts of our codes, that limit the overall speedup. For the increasing number of hybrid architectures, a parallelization combining threads and message passing is a promising way to reduce the parallel overhead.
Our main interest is to develop efficient libraries for particle simulations, which are portable to all important parallel platforms. Therefore access to several different parallel architectures is crucial for us to verify our concepts. Here we

* 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).

can profit from the large number of parallel architectures the HLRS provides. For the development of the libraries, the parallel machines are not needed for large production runs, but the applications on top of the libraries have a great demand for parallel computing power.
We are mainly working with three parallel machines. The Hitachi SR8000 and the Cray T3E, both installed at the HLRS in Stuttgart and the Kepler Cluster installed in Tübingen. The Hitachi is a 8-way SMP machine coupled with a fast communication network. The Cray is a conventional parallel machine with one 450 MHz Alpha processor per node and a fast 3D–torus communication network. Kepler is built of commodity hardware and has dual SMP Pentium III nodes with 650 MHz processor speed and a switched full–bisection-bandwidth Myrinet network.

1.1.1 The SPH Method

One main focus in our Sonderforschungsbereich 382 are particle simulations. An important particle simulation method is Smoothed Particle Hydrodynamics (SPH). SPH is a grid-free numerical Lagrangian method for solving the system of hydrodynamic equations for compressible and viscous fluids. It was introduced in 1977 by [3] and [6] and has become a widely used numerical method for astrophysical problems. Due to the mesh-less nature of the method SPH is especially suited for free-boundary problems. Another advantage of SPH is the ability to handle large density gradients well. Rather than being solved on a grid, the equations are solved at the positions of the so-called particles, each of which represents a mass element with a certain volume, density, temperature, etc. while moving with the flow according to the equations of motion.

1.2 The Libraries

1.2.1 A Hybrid Parallel Library for Particle Simulations

Why Hybrid?

The dominant part of the TOP500 super computers are so called hybrid parallel architectures, a combination of shared memory nodes with message passing communication between the nodes. Two large hybrid machines at the HLRS in Stuttgart are the Hitachi SR8000 and the NEC SX5.
Hybrid parallelization is the combination of two programming models optimized for hybrid parallel architectures. It uses a thread based programming model for parallelization on the nodes together with message passing parallelization between the nodes.
Obviously, the share of common data structures on one SMP node can reduce the amount of memory used. But more important is the reduction of

the communication. Every parallel implementation has a maximum speedup limited by its serial part. In our non trivial parallel codes the dominant serial part is the communication. A hybrid implementation reduces the amount of communication, because some data is shared on the SMP node and thus the communication is implicit. But also the communication itself is faster. A small test shows this for the MPI *Allgather* call with a 200 MB data array on the Hitachi SR8000. K is the number of nodes. The numbers are the time for 50 MPI *Allgather* calls in seconds.

	K=1	K=2	K=4	K=8
MPI intra/inter	15.8	49.2	90.2	116.8
MPI inter	9.3	14.6	19.5	19.7

One can see, that the MPI *Allgather* with pure inter-node communication (comparable to the hybrid programming model) is much faster than the MPI call where MPI is also used for intra-node communication.

OpenMP

To keep the implementation portable and support the majority of hybrid architectures, we chose OpenMP, a wide-spread standard for shared memory parallelization. Compilers for OpenMP are available on all important hybrid plattforms including Hitachi SR8000, NEC SX5 and Linux (Intel C++ or Portland Group compilers). OpenMP has other advantages over explicit thread programming, for example POSIX threads. It annotates sequential code with compiler directives (pragmas). This makes an incremental parallelization possible and keeps the code portable, since non-OpenMP compilers ignore the directives. POSIX threads require to implement parallel sections in separate functions and to write wrappers (sometimes called jackets) for functions with more than one argument, since the POSIX threads API supports only one argument. OpenMP, on the other hand, allows to join and fork threads at arbitrary positions in the source code.

Hybrid Development

The hybrid implementation is based on the ParaSPH library for particle simulations. ParaSPH is written in C and parallelized with MPI. The library separates the parallelization from the physics and numerics code. The interface between the library and the application is optimized for particle simulations. The library provides an iterator concept to step through all particles and their neighbors and later communicates the results of this time step. In parallel mode, the library transparently distributes the work amongst all processors. Every local iterator processes only a subset of all particles. The code performs well on machines with a fast message passing network. We tested the code on Cray T3E, Hitachi SR8000, IBM SP and the Kepler cluster.

In the hybrid implementation, OpenMP is used for the inner node parallelization. MPI is still used for inter-node communication. It showed, that it is necessary to optimize the load balancing. Therefore, we introduced a two stage balancing. The "old" load balancer for distributed memory is only used for a coarse load balancing between the nodes. For the fine balancing on the node we provide two new load balancers. The user can choose between a fixed load balancing and a dynamic master-worker load balancing. The master-worker algorithm promises the best load-balancing, especially for inhomogeneous problems but has disadvantages in cache utilization. That is why the static load balancing is usually faster.
We used three different OpenMP features (omp parallel, omp barrier, omp threadprivate) together with some functions of the OpenMP library. All over, we used 9 *parallel* pragmas, one *barrier* pragma and one *threadprivate* pragma to parallelize about 95 percent of the code. We also needed two additional locks to protect internal structures.

Experiences on Hitachi SR8000 and Kepler

The first OpenMP version of ParaSPH on Hitachi SR8000 frequently called sections with a *critical* region and showed a bad performance. Explicit locks instead of critical regions improved the performance only a little. Enhancing the fixed load balancer with a lock free implementation fixed the problem.
On the Linux platform, we used Intels C++ compiler. To port the code we had to replace the *threadprivate* directives by explicit memory allocation for every thread, because the *threadprivate* directive triggers a bug in the compiler.

Performance results

On both architectures, we measured the speedup (figure 1.1) of a SPH simulation with 300 000 particles and about 80 interaction partners per particle. This medium sized simulation requires about 900 megabytes of memory on one node. One time step needs about 23 seconds on one Hitachi node (8 CPUs) Typical production runs need 1 000 or even more time steps resulting in several hours of computation time on eight CPUs and about two days on one CPU.
We compared speedups of the pure MPI and the hybrid parallelization for different numbers of processors. On Kepler with only two CPUs per SMP node, we observe almost no difference between the pure MPI and the hybrid parallelization. The Hitachi with 8 CPUs per node shows a big difference between the pure MPI and the Hybrid version especially for large node numbers.

There are three main reasons for the performance impact of the hybrid version. First, the code is not fully annotated with OpenMP instructions. There is a

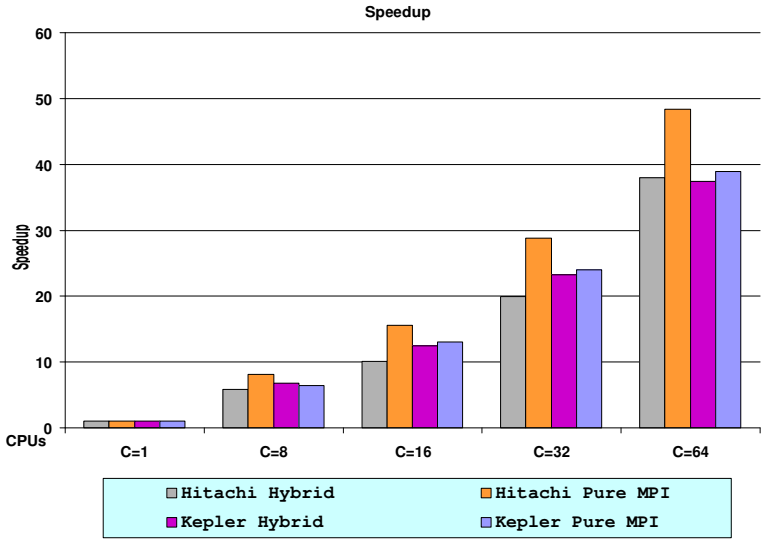

Fig. 1.1. Speedups using different parallelization strategies on Hitachi SR8000 and Kepler.

small serial part (about 5%), which is not present in the pure MPI version. This shows a performance decrease especially on the Hitachi SR8000 for large numbers of OpenMP threads. The second reason may be a lesser efficient cache utilization and the third reason is the OpenMP overhead itself (thread creation, locking of critical sections).

1.2.2 Object-Oriented SPH

An Object-Oriented Library for Particle Simulations

The SPH simulation program sph2000 is implemented on top of an object-oriented particle simulation library written in C++. We wanted to prove the feasibility of the object-oriented approach in the performance critical domain of particle simulations. First results are shown in this section. Our principle subject is the development of an object-oriented library for particle simulations. The object-orientation brings a well structured design of a library with classes modeling the elements of the problem domain. Design patterns [2] help to organize the classes to get a clear and efficient design. One main concern in the design was the strict decoupling of the parallelization and physics. These aspects increase the extendibility, maintainability and reusability of the code. Another goal was to simplify the configuration of simulations, which mainly means to configure a simulation run after the compilation at runtime by reading a parameter file. To avoid conditional compilation with preprocessor directives, as it is often seen in C libraries, the strategy pattern, which is based

on the object-oriented concept of polymorphism is used. With this pattern the program instantiates objects at runtime due to the configuration parameters. One such strategy is the communication shown in figure 1.2.

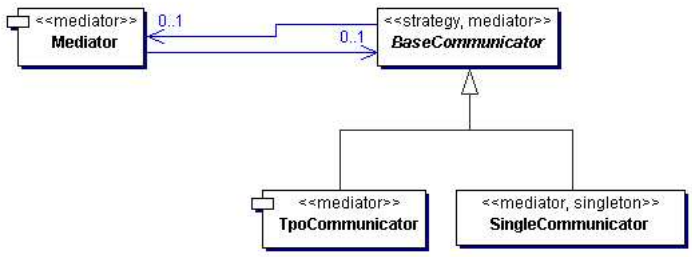

Fig. 1.2. Class diagram of the sph2000 communication classes.

An abstract base class defines the interface for the communication classes, concrete subclasses define the communication code for different machine architectures. sph2000 runs on architectures with shared and distributed memory. On distributed memory machines, a domain decomposition splits the simulation space in rectangular subregions, which are distributed one per node. Each subregion calculates all its particles geometrically. To interact with the particles in the subregion of the neighbors, proxy particles are copied and sent between the neighbor subregions. Both communication codes, the SingleCommunicator and the TpoCommunicator classes are compiled but due to a configuration parameter only one will be loaded and instantiated. The same strategy pattern is used for the kernel function, the integrator, the physical quantities and the handling of the subregions of the simulation domain.
The mediator pattern is used twice to decouple parallelization, the numerics and the physics. First, the communicator class mediates between the nodes. It handles the whole communication. If neighboring nodes need proxies, the communicator requests them from the subregion mediator.
Inside a node, a subregion mediator class controls the execution of the calculation classes as shown in figure 1.3. It is responsible for a correct order of calculation and communication and decouples the calculation classes by keeping the objects from referring to each other explicitly. This concept separates the object interactions and class dependencies from the physical classes. The result of object-oriented techniques with design patterns is a library, in which classes have clear and strictly separated responsibilities. Different methods can be interchanged without influencing other code. Extensions, for example a new physical effect, are delimited to the corresponding strategy.

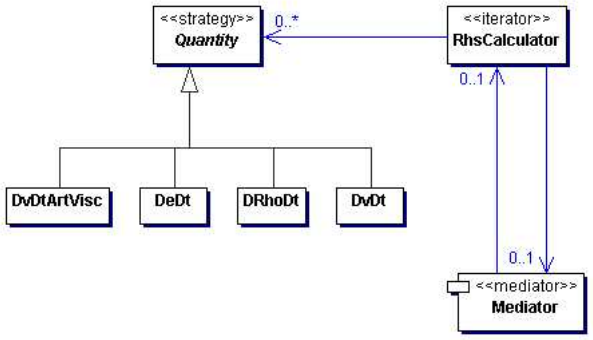

Fig. 1.3. Simplified class diagram of the sph2000 calculation classes.

Performance Results

We run simulations with the object-oriented SPH code sph2000. The speedups can be found in figure 1.4.

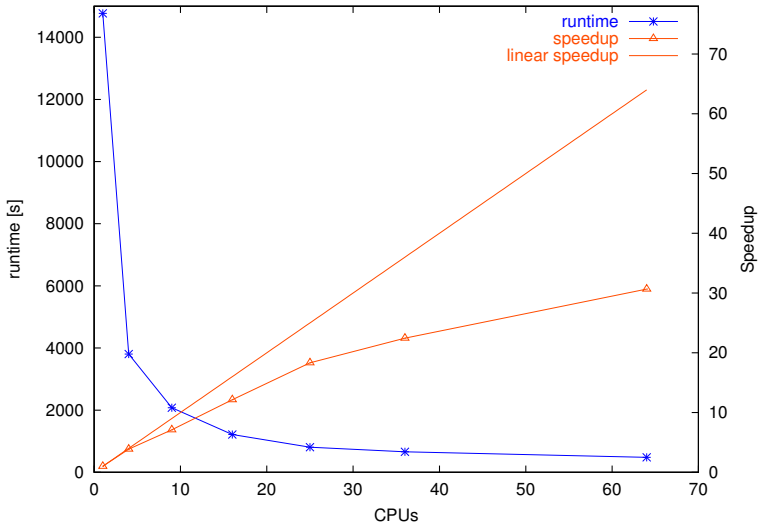

Fig. 1.4. Time and speedup on Kepler for 100 000 particles with sph2000.

The measuring took place on the Kepler cluster, simulating 100 000 particles on 1, 4, 9, 16, 25, 36 and 64 nodes. We plan to port the C++ program to the Cray T3E to compare the performance of the object-based communication with the Kepler results.

1.3 Applications

Although its main application still lies in the astrophysical area, the SPH approach is used nowadays also in fields of the material sciences, for modeling multiphase flows (Diesel injection problem, see [9] for details), and the simulation of brittle solids (e.g. [1]). To give an short overview and to show the advantages of the method, we present two examples of calculations, which were done using ParaSPH and one example done using sph2000. The first example is an astrophysical application, the simulation of a protoplanet embedded in a protoplanetary accretion disc, the second is a collision of two rubber rings and the last example is the injection of diesel into an air filles chamber.

Simulations of protoplanetary discs

Since the discovery of the first extrasolar planet in 1995 [7], the interest in planet formation theory has grown intensely. Most of the over one hundred new found planets feature attributes that differ completely from those of the planets of the solar system. These planets have masses in the range of 0.4 to 11 Jupitermasses, and large eccentricities up to 0.67, while the most massive planet in our solar system is Jupiter and the highest eccentricity is 0.25. Nowadays planet formation theories need not only to explain the formation of the solar system, but the formation of planetary systems with these different features.
It is generally believed that planets form in circumstellar discs, which have formed out of a gravitational collapse of a dense molecular cloud. The typical lifetime of such a disc is between 10^6-10^7 years. During this time material migrates constantly radially inwards in the disc and finally accretes onto central protostar. The dust particles in the disc grow through a hit-and-stick mechanism and finally combine into larger bodies, which are eventually able to build planetesimals. Planetary sized objects form out of these planetesimals by collision processes. These objects grow further by accretion of gas and dust, and build the giant planets. A more detailed description of planet formation can be found e.g. in [10].
The high masses of the new found planets give rise to more questions. The tidal interaction between a massive protoplanet and the protoplanetary disc can lead to a so-called gap formation at the orbital region of the planet. This gap formation has severe impact on the accretion rate on the planet, and can probably prevent accretion through this gap completely depending on the mass of the protoplanet. In numerical grid-based simulations it was shown by [5] that the gap formation process does not inhibit accretion in the case of a jovian protoplanet and material can still reach the planet.
By the use of supercomputers, we were now able to study the protoplanet-disc system with SPH. The evolution of the surface density during the first tens orbits of the protoplanet is shown in figure 1.5 for a Jupiter-sized planet at

an orbital radius of 5.2 AU and a central star with one solar mass. Already after ten orbits, the decrease of density at the orbital region of the planet is visible. After forty orbits the gap is completely formed, and the accretion onto protoplanet diminishes. In order to achieve a high enough spatial resolution of the accretion on the protoplanet, more than 360 000 particles have been used for this simulation.

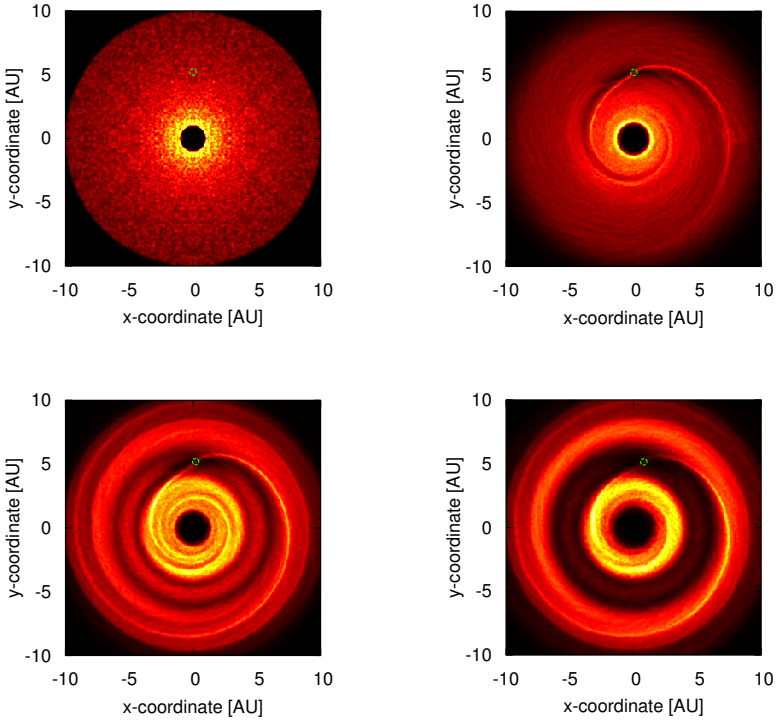

Fig. 1.5. Color-coded plots of the surface density in the case of a jovian protoplanet at 5.2 AU away from the protostar. The protostar is located in the center of the disc and has one solar mass. The surface density is shown for four different times: at the beginning of the simulation, after one, ten and forty-two orbits of the protoplanet. The circle represents the approximation for the Roche lobe of the planet.

Simulation of colliding rubber rings

Although the SPH method has its roots in the astrophysical area, recent enhancements lead to the application of SPH in solid body physics. By intro-

ducing an additional elastic stress tensor, the SPH method can be successfully extended to model elastic solid bodies (see [4] for details).
As an example we present the simulation of the collision of two rubber rings, which is a typical test problem for elastic solid body codes: Two rubber rings with the initial density ϱ_0 collide with 6% of the sound speed c_0. The shear modulus μ, which characterizes the elastic behavior of the material, is set to 0.22 $\varrho_0 c_0^2$ in this problem. The results of the simulation are presented in figure 1.6. The rings bounce back off each other as expected and continue to oscillate.

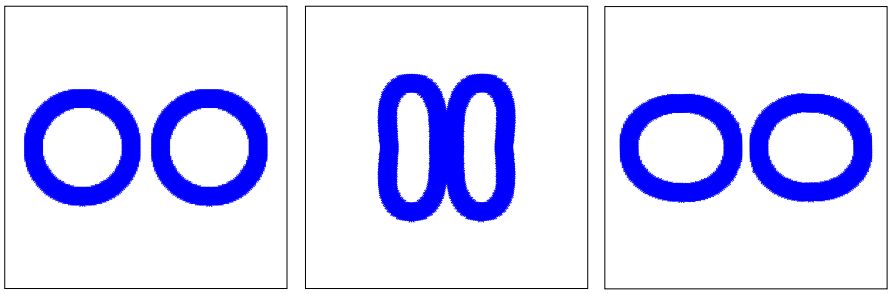

Fig. 1.6. Collision of two rubber rings. The pictures show the rubber rings at the start of the simulation, in the moment of the highest compression, and after the collision when they continue to oscillate.

Simulation of diesel injection

Diesel engine manufactures are interested in an optimal injection of the diesel into the combustion chamber. A perfect mixing of diesel and air means means an efficient use of the fuel and therefore reduces emissions. To achieve this, the breakup of the diesel jet must be examined and understood. When injected into the cylinder of an engine, the diesel jet undergoes two stages of breakup. In the *primary breakup* large drops and filaments split off the compact jet. These turn into a spray of droplets during the *secondary breakup*. This secondary process is well known and can be modeled as a spray, but the understanding of the primary breakup is only in the initial stages. The physical effects that might influence the primary breakup are the pressure forces in the interface region of diesel and air, instabilities of the jet induced by cavitation inside the injection nozzle, surface tension and turbulences.

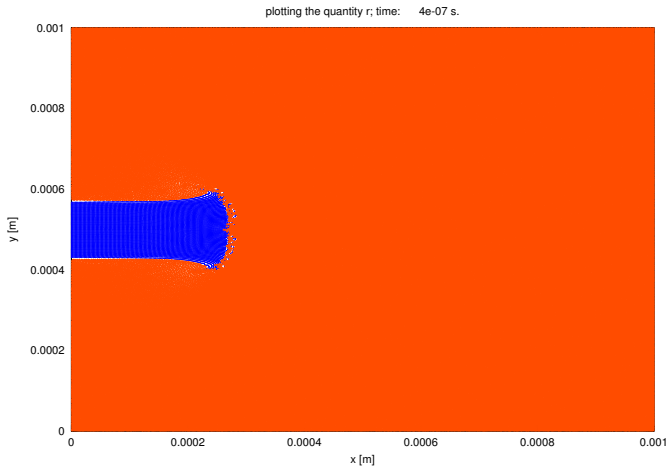

Fig. 1.7. Injection of diesel in an air filled chamber, simulation with 200.000 particles. The picture shows the injected diesel after $7.2e-07$ s. First drops are already splitted off the jet.

1.4 Future work

1.4.1 Parallel I/O

The computation of our massive parallel application showed a lack in performance due to sequential I/O. For example, writing the data of $10 \exp 6$ particles in sph2000 in each integration step limits the maximum speedup of the parallelization to 25. Moreover, visualizations of physical simulations typically require hundreds of gigabyte. As a result, the I/O bottleneck limits the scalability of the parallelization significantly. The constantly growing gap between processor and hard disk performance during the last decades makes this problem even worse and requires the use of parallel I/O systems. Distributed hard disks in special I/O nodes, which use the underlying communication network for data exchange are the most common way in massive parallel systems to overcome the problem. Thus, the application can profit from the aggregated performance of all hard disks available in the system.
However, special parallel I/O interfaces have to be implemented and used by the application to benefit from the whole hard disk performance. The most often used interface for parallel I/O is MPI-IO [8]. One implementation of this part of the widely accepted and used standard MPI 2 is ROMIO [11], which is setup on top of the file system using an abstract device interface for I/O (ADIO) to achieve high portability (e.g. NFS, PIOFS, HFS, XFS).
In implementing parallel I/O two major problems rise: First, the application has to be ported on the parallel I/O interface of MPI 2 to enable the use of higher hard disk throughput. Second, besides a parallel implementation

sequential I/O also has to be implemented for keeping the portability of the interface. Both leads to a more complex and larger implementation. Moreover, MPI 2 only supports procedural interfaces for parallel I/O, which makes it hard to be used in object-oriented applications. Therefore, concepts for object-oriented parallel I/O have to be developed, which is a very difficult task due to irregular distributed object structures in main memory and dynamic memory sizes of the objects.
To close the gap between object-oriented software development and procedural communication we developed an communication library for parallelizing object-oriented applications called TPO++. Its concepts for mapping objects on simpler communication structures can be reused for the implementation of object-oriented parallel I/O. Additional, wrapper classes for file management are already implemented and only have to be extended by interfaces for data transfer and function mapping on MPI 2 calls. The interface design is very important to keep the high portability of TPO++ and the compatibility to already existing applications.
Finally, the interface will be integrated in our object-oriented applications to increase their performance and functionality. Verifications will then help to optimize and improve the object-oriented parallel I/O interface.

1.4.2 Applications

We have extended the two dimensional object-oriented code to three dimensions. Due to the increased number of particles in three dimensions, we are investigating solutions to decrease the amount of calculated interactions. First, we try to filter the shock wave of the injected diesel out of the outer air regions. The problem with the shock wave is its movement with sound speed towards the system boundaries together with the interference of its reflections with the diesel jet. So, the cylinder size must be chosen big enough to prevent this artificial interference. The second step will be the online generation of air particles according to the motion of the jet. The idea is to calculate only air regions, which are affected by the diesel jet. Another field of work is the development of new physical models of the surface tension and the cavitation. To implement these quantities efficiently, we need new concepts for the parallelization, for example to adjust the neighborhood of particles.

1.5 Summary

The application of object-oriented development methods has improved the quality of our simulation codes. While having a performance, which is comparable to procedural approaches, an object-oriented implementation is easier to maintain and extend, f.ex. to add new physics.
For massive parallel applications, scalability is always an issue. Besides improving the parallel algorithms, we currently follow two other approaches to

reduce the serial parts: A hybrid parallelization promises to reduce the communication cost, because intra-node data can be shared, and parallel I/O can reduce the time spent for persistency. However, the study of ParaSPH on Hitachi SR8000 shows that improving the performance with an hybrid parallelization is not trivial. The main obstacle for the use of parallel I/O is the lack of an object-oriented interface in current standards.

References

1. W. Benz and E. Asphaug. Catastrophic Disruptions Revisited. *Icarus*, 142:5–20, November 1999.
2. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, 1995. 22nd Printing July 2001.
3. R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics: Theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389, 1977.
4. J. P. Gray, J. J. Monaghan, and R. P. Swift. SPH elastic dynamics. *Computer Methods in Applied Mechanics and Engineering*, 190(49-50):6641–6662, 2001.
5. W. Kley. Mass flow and accretion through gaps in accretion discs. *Monthly Notices of the Royal Astronomical Society*, 303(4):696–710, March 1999.
6. L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82(12):1013–1024, 1977.
7. M. Mayor and D. Queloz. A Jupiter-mass companion to a solar-type star. *Nature*, 378:355–359, 1995.
8. MPI-IO Committee. *A Parallel File I/O Interface for MPI*. Online. URL: http://lovelace.nas.nasa.gov/MPI-IO, 1996.
9. F. Ott and E. Schnetter. A modified SPH approach for fluids with large density differences. *ArXiv Physics e-prints*, pages 3112–+, March 2003.
10. M. A. C. Perryman. Extra-solar planets. *Reports on Progress in Physics*, 63(8):1209–1272, 2000.
11. R. Thakur, E. Lusk, and W. Gropp. Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation. In *Technical Memorandum ANL/MCS-TM-234*, Mathematics and Computer Science Division, Argonne National Laboratory, 1998.