Table of Contents

A Parallel Object Oriented Framework for Particle Methods*

M. Hipp[1], S. Hüttemann[1], M. Konold[2], M. Klingler[2], P. Leinen[3], M. Ritt[1], H. Ruder[2], R. Speith[2], and H. Yserentant[3]

[1] Wilhem-Schickard-Institut für Informatik, Universität Tübingen
[2] Institut für Astronomie und Astrophysik, Universität Tübingen
[3] Mathematisches Institut, Universität Tübingen

Abstract. A major goal of the Sonderforschungsbereich 382 is the development of different particle methods for solving physical problems. In this paper, we present two different implementations of parallel Smoothed Particle Hydrodynamics codes and a newly developed method called Finite Mass Methode. We follow up with the results of some test simulations which show the pros and cons of the different approaches.

We talk about our experiences with these codes and conclusions for parallelization in general. Our recent work concentrated on object-oriented physical codes and runtime systems. The newly established particle method called Finite Mass Method (FMM), which got developed within the SFB 382 was ported to C++, and parallelized using MPI-1.1. The experiences made with this code led to an object-oriented parallel runtime system, based on message passing and threads. Finally we describe our proposed solutions for higher-level programming abstractions, including design patterns and application-domain specific libraries. Based on this work we currently develop an object-oriented prototype of a QMC simulation.

1 Introduction

There is a need for parallelization in a considerable number of physical research fields. With more computation power, you can compute larger problems (memory-parallelization) or you gain a better resolution for your existing simulations. One major focus in our SFB are particle methods. We assisted several parallelization projects in the last years. One of the first production codes was a parallel implementation of the SPH (Smoothed Particle Hydrodynamics) method, which was done on our own procedural thread-based parallel runtime system (DTS). The system showed its limitations for large-scale distributed memory codes. An implementation of a newer SPH code based on MPI (and SHMEM for the Cray T3E), which achieves very satisfying performance is described in section 2. Included is a comparison of SHMEM and MPI performance on Cray T3E and MPI performance on the SP system.

* 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 the recent time, we noticed the need for a shift towards object-oriented programming methods for reusable parallel algorithms and design patterns in order to cut down development time and to enable the collaborative work of larger group of scientists. For gaining experience we decided to port an existing code of a new approach for a particle-based fluid simulation called Finite Mass Methode (FMM). The object-oriented code was initially parallelized using MPI. The method and some preliminary performance results of the parallel code are presented in section 4.

Based on this work, we propose an object-oriented parallel runtime system and specific algorithm libraries for the parallelization of particle methods in section 5. A new, object-oriented version of the FMM code as well as the distributed SPH code is planned for this year. We are also developing a parallel object-oriented version of a QMC code simulating a neutron star. All three codes are supposed to share a significat amount of library code.

2 SPH on SHMEM and MPI

In [11] we introduced our SPH implementation optimized for Cray SHMEM message passing. We achieved very good speedups on the Cray T3E up to 512 nodes with this code. There was a demand to continue this effort in order to get a library for a wider range of architectures and applications.

From SHMEM to MPI This code seperates the communication part rather strictly from the physical model. In order to compare the implementation for different architectures we ported the communication library of the SHMEM-based code to MPI-1.1. Due to the limitations of the MPI-1.1 standard a few one side communication parts needed a major rewrite but most of the work was simply done by exchanging the SHMEM calls by their equivalent (two-sided) MPI calls. We expect less porting efforts with the availability of a MPI-2.0 implementation on the major platforms. In the exisiting code we wrote wrapper functions around near all communication calls, so we still used only a few direct calls to SHMEM functions and therefor the exchange of the SHMEM calls was quite easy. Not a single line in the main physical application had to be changed.

Due to the dynamic of SPH, the simulation needs a good communication bandwidth and a low latency. The most expensive communication parts are the distribution of particle positions, which is a non parallelized gather/broadcast operation between all nodes. A latency-sensitive part is for example the parallel construction of a particle grid.

We plan to use this application as one of several real-world test application to compare parallel environments. The program has a good mix of different requirements, such as the need for a high communication bandwidth and low latency together with a good floating point performance for some real computations and a considerable integer performance to handle complex data structures. The application should perform well on a good balanced system.

Native SHMEM Communication vs. MPI We measured the performance of the MPI code on the Cray T3E in Stuttgart and on the SP system in Karlsruhe. On the SP we used 128 P2SC thin nodes with 120MHz. The tests showed, that the MPI implementation of the Cray T3E is worse compared to the native SHMEM library (see figure 1). Our tests show that CRAY could easily improve the MPI performance by making better wrappers around existing SHMEM calls. For some communication parts, such as gather operations of large arrays, the throughput decreased from about 300MB/s using SHMEM to 120 MB/s using MPI. On the SP we achieved the expected performance of around 50MB/s. The whole code didn't perform this good on the SP system, because the implementation is optimized for Cray T3E and depends heavily on good MPI performance in order to scale beyon 64 Processors.

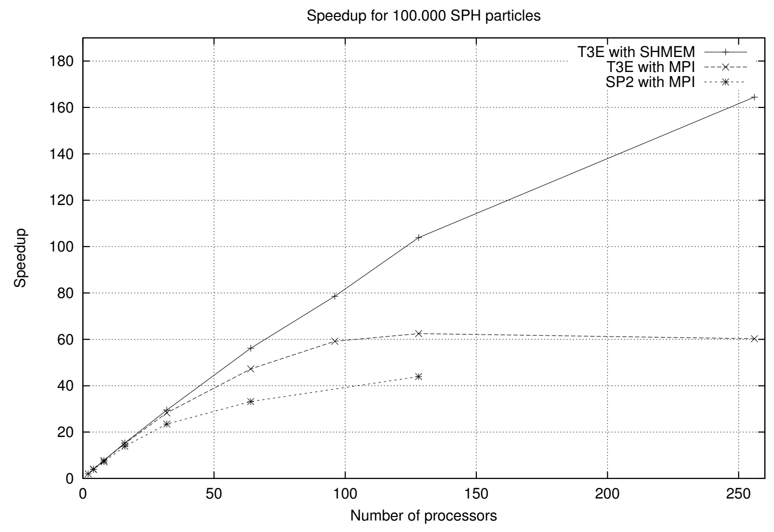

Fig. 1. Speedup of the SPH Simulation on Cray T3E with SHMEM and MPI and on SP with MPI for a mid-size problem with 100000 SPH particles. For larger node numbers the curves are dominated by the non parallelized communication parts such as gather/broadcast operations between all nodes. Please note the effect of the limitations of the MPI implementation on the CRAY T3E beyond 128 nodes.

Results The result of this performance test is, that for existing code it can be useful to exchange performance-critical communication parts in a MPI implementation by SHMEM functions to achieve better performance and better scaling to higher number of nodes. If a high level abstraction of a problem isn't possible you should at least consider to write wrapper functions around your explicit communication calls to simplify the later optimization of the applications hotspots. For new developments it proved to be beneficial to use an abstraction layer, which allows the exchange of low level parts without changing the application to obtain the best performance on a given hardware platform. We chose an object oriented programming model for this abstraction, which is introduced in the next chapter.

3 The need for Object-Oriented techniques

In providing our own runtime system we try to alleviate the error-prone and troublesome process of parallelizing physical simulation code.

Since most existing codes are procedural – written in FORTRAN or C – our first approach for a parallel runtime system was to provide the widely known interface of threads for architectures with distributed memory [2]. The standard model of communication between local threads, data exchange via shared memory, cannot be supported efficiently in a transparent and portable manner, and has been replaced by a distributed shared memory interface [9] for direct thread-thread communication.

From the work of the last years, we found reasons to redesign our simulation environment using object-oriented techniques:

1. using message objects on the lowest level to communicate between concurrent program units on distributed memory computers seems to be most natural and easy to use for our simulation methods and hides the difficult syntax of the communication libraries from the user.
2. the growing complexity of our simulation programs requires structural elements in the programming paradigm not offered by e.g. FORTRAN or C. A special requirement of scientists is ease of understanding and extending of exisiting code due to the high right of fluctuation in the scientific workplace. Also using an object-oriented approach to describe the problem is closer to the physical model used and leads to improvements of the algorithms by making use of more physical properties.
3. to exploit the usual features promised by object-oriented programming (reusability etc.) our project partners programmed in C++; which resulted in code which was reusable and modular within its own scope.

Our goal is to provide a well documented and maintained library of reusable and extensible solutions for astrophysical simulation methods in the near future. This also should give a guideline on how to use object-oriented techniques for our simulation methods (e.g. SPH).

To gain experience, we decided to port a new particle method to C++ and parallelize it using standard MPI.

4 Finite Mass Method

4.1 A new Approach to Lagrangian Fluid Dynamics

Within the SFB 382 H. Yserentant [13] developed a new approach to solving the continuums mechanics of fluid dynamics which is similiar to finite elements and finite volumes but completely Lagrangean in nature.

In contrast to finite volume and finite element methods this new method is not based on discretization of space but on discretization of mass which is from the point of view of a physicist at least as natural as the former approach.

The demand of astrophysical problems on computational power and high quality algorithms led to the development of this new method for solving fluid dynamics numerically. The method impresses with high accuracy and correctness regarding the conservation of energy, momentum and angular momentum. The later properties and the convergence of the finite mass methode are mathematically proven which is a major advantage compared to the Smoothed Particle Hydrodynamics method

These mass packets of finite size do have a finite number of interiour degrees of freedom and are under the influence of external and internal forces. The shape and size of the mass packets as well as the translation of the center of mass is controlled by internal and external forces, while obeying the laws of thermodynamics.

It proved that FMM is highly suited for problems with free boundaries, which are often encountered in astrophysics. The lagrangian approach also avoids the problem of adapting the grid during the simulation.

Arbitrary linear deformations like contraction, expansion, rotation and changes of shape combined with intersection and penetration of the packets leads to a high quality representation of the pysical function with a rather low number of packets. We already obtained very good numerical results with only 20 packets for each dimension for one of our test problems, the viscous gasball.

The higher number of physical values, compared to the Smoothed Particle Hydrodynamics method, stored in each packet and the rather low number of required simulation packets is more easy to scale efficiently on distributed memory machines like the CRAY T3E. The currently available code we used in our numerical experiments shows fourth order convergence and behaves numerically stable while using reasonable amounts of memory.

Originally the very first implementation of FMM was done in PASCAL on a Sun Workstation. This implementation lacked portability, speed and an easy way to do parallization. There is currently no standard way to use MPI safely in a portable manner with PASCAL compilers.

Instead of simply porting the available application to C we decided to use an object oriented approach, which allows the developers of the algorithms to exchanges fundamental parts of the codes without breaking other sections like the physical models. Other major reasons are efficient implementation, which forbids layering and its inheritent copying overhead and ease of parallelization. By putting a large effort in complex classes we could hide very efficent, but hard to understand implementations from the developer of the pysical model while still being able to exchange the underlying algorithms without the need to notify each other. This proved to be beneficial to heterogenious groups of developers. The object oriented design pattern was helpful in making a cache efficient implementation, which otherwise would have obscurified the code base.

A FORTRAN implementation was no option due to missing features in representing the rather complicated data structures employed in the original PASCAL version.

It showed that while the C++ version employes most complex data structures in order to gain cache efficiency and hiding of the parallization code that this code is optimal for well balanced scalar central processor units like the SPARC ULTRA and the PENTIUM PRO class.

These complex data structures lead to an integer/floating point ratio of nearly one and rather short vector lengths so that this implementation runs well on the scalar distributed memory machine CRAY T3E in contrast to the NEC SX4 parallel vector processor. Due to the complex data structures it showed that the code runs twice as fast on a single INTEL Pentium II with 450 MHz compared to a single node on the T3E-900 which suffers from branches due to a rather long pipline and missing efficient implementation of branch prediction.

The finite mass method is a purely Lagrangean approach to solve problems in continuum mechanics. It somewhat resembles much more the concepts of finite volume and finite elements methods than pure particle methods like the Boltzmann-like transport equations.

4.2 The Particle Model of Continuum Mechanics

Instead of starting from the Euler and Navier Stokes equations the prefered way to introduce FMM is to start from the basic physical principles which finally lead to these well known differential equations. In the trivial case of of an adiabatic, inviscid flow, the equations of motion are straight forwardly derived via the Lagrangian method from the potential and kinetic energies.

The necessary breakage of invariance with respect to time reversal is achieved via frictional forces which tend to zero for the limit of velocity differences of neighbouring packets.

The funktion $\Psi_i(\boldsymbol{y})$ which describes the mass distribution inside a mass packet i with finite extension in space has to fulfill the following properties.

$$\int \Psi_i(\boldsymbol{y})d\boldsymbol{y} = m_i, \int \Psi(\boldsymbol{y})\boldsymbol{y}d\boldsymbol{y} = 0 \tag{1}$$

One possible choice for the functions Ψ_i as proposed by Yserentant is

$$\Psi_i(\boldsymbol{y}) = \alpha_i \Psi\left(\frac{\boldsymbol{y}}{h_i}\right) \tag{2}$$

with the shape function Ψ.

These properties provide normalization and coincidence of the center of the packet and the center of mass. This Lagrangean approach automatically leads to the conservation of mass. The fluid is then described as the superposition of the above individual mass packets in space.

It is mathematically proven that the constants of motion energy, momentum and angular momentum are conserved within FMM.

We suppose further that with the vector components y_k and y_l of $\boldsymbol{y}$ fullfill the following condition

$$\int \Psi(\boldsymbol{y}) y_k y_l d\boldsymbol{y} = J\delta_{kl}. \tag{3}$$

In our case it showed to be beneficial to make the function Ψ up from piecewise one dimensional polynomial functions $\tilde{\Psi}(y_k)$. The above constant J is independent on the space dimension and we used J=1/12 for our calculations.

A suitable choice for building up the function $\Psi(\boldsymbol{y})$ from piecewise polynominal one dimensional functions in our computations was the normalized compact third order B-spline given by

$$\tilde{\Psi}(\xi) = \frac{4}{3} \begin{cases} 2(1+\xi)^3 & , \quad -1 \le \xi \le -1/2 \\ 1-6\xi^2(1+\xi) & , \quad -1/2 \le \xi \le 0 \\ 1-6\xi^2(1-\xi) & , \quad 0 \le \xi \le 1/2 \\ 2(1-\xi)^3 & , \quad 1/2 \le \xi \le 1 \\ 0 & , \quad |\xi| > 1 \end{cases} \tag{4}$$

which then is combined to the tensor product

$$\Psi(\boldsymbol{y}) = \prod_{k=1}^{d} \tilde{\Psi}(y_k). \tag{5}$$

The superposition of the masses of single packets results in the total mass density.

The normalized frictional force for a packet within the fluid is governed by the difference from the average sourounding velocity and leads to dampening of local velocity fluctuations, coupling of packets and production of entropy according to the second law of thermodynamics.

$$F_i^{(r)} = -\frac{1}{2}\int R\Psi_i[\boldsymbol{v}_i - \boldsymbol{v}]dx \tag{6}$$

The local relaxation time $T = 2/R$ of the system represents for constant R and $\boldsymbol{v} = 0$ the time which is needed to dampen the velocity by a factor of $1/e$. This relaxation time vanishes for smooth flows with the second order of the local packet size but is dominant for shocks.

4.3 Experimental Results

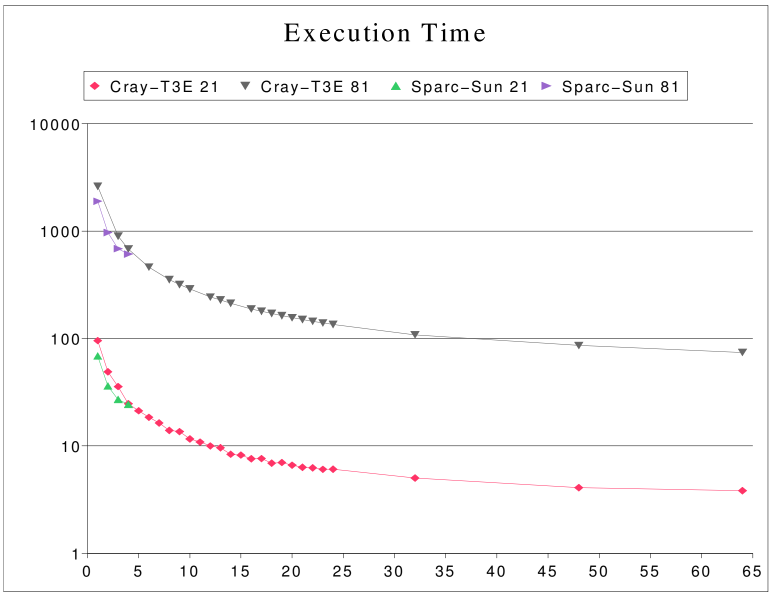

Fig. 2. Execution time over number of processors for a two-dimensional gasball with initial azimutal and radial velocity towards the center.

The current parallel FMM implementation is based on MPI-1.1 and uses a coarse grained parallel tree approach. The parallel tree helps to hide the implementation details of the parallel data structures from the physicist developing new applications and from the numerical mathematicians testing new integration algorithms. It showed to be beneficial that the integration is parallized implicitely via the distributed packets in a parallel tree structure. The users are able to develop new applications and algorithms without beeing concerned about any MPI details.

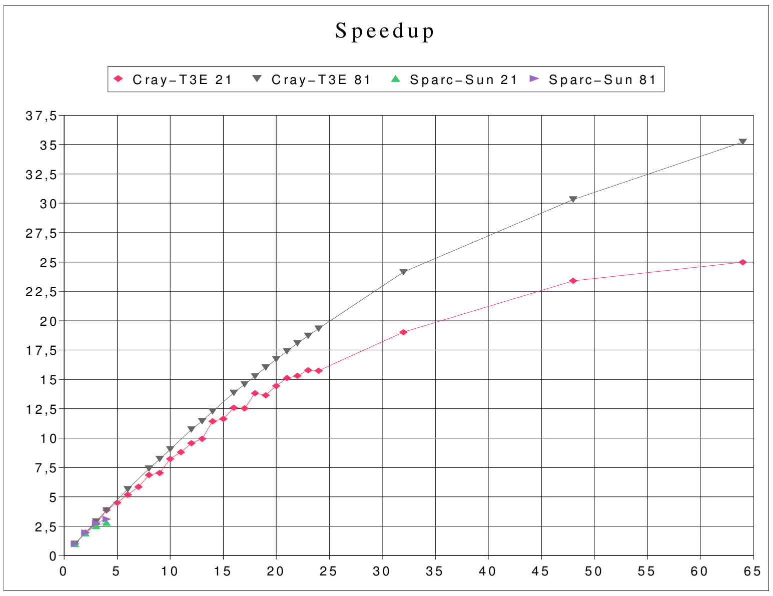

Fig. 3. Speedup over number of processors for a two-dimensional gasball with initial azimutal and radial velocity towards the center

Figure 2 shows that both the smaller two-dimensional 21×21 packet test problem and the bigger 81×81 test problem. The execution time on the CRAY T3E-900 is compared to a four processor SUN Enterprise Ultra SMP machine running at 300 MHz. It can easily be seen that the SPARC processor family is much better suited to more complex data structures than the ALPHA processor which results in about 40 percent less execution time on the single node 300 MHz SPARC ULTRA compared to the 450 MHz DEC 21164. Some preliminary tests on much cheaper Intel Pentium II/III machines showed the same performance per clock cycle like the SPARC ULTRA. Due to memory bandwidth limitations and unfortunate scheduling algorithms on the SUN ENTERPRISE noticeably performance decrease is already observed for employing only three processor nodes. Because of the excellent and well balanced network interconnect of the CRAY this machine scales very well even for relatively small problems with very high communication overhead. Figure 3 gives an estimate of the possible speedups for the current parallel FMM implementation. It is easily seen that the speedup of the SUN ENTERPRISE can not compete with the CRAY T3E. The possible speedup is very much dependent on the problem size. Three dimensional problems are between 20 and 1000 times larger than the current two dimensional test problems so that estimates show that this parallel FMM implementation should scale up to 384 processors on a T3E.

5 An object-oriented Parallel Runtime System

Based on the experiences with the implementation of FMM, we now introduce our proposal for a object-oriented parallel runtime system. The basic architecture can be seen in figure 4.

On the lowest level of our parallel programming system is an object-oriented message-passing layer with a functionality close to MPI. To keep this layer portable, it is designed to be easily implemented on different low-level communication primitives. There exists a UDP based version for local test runs in a LAN environment and a MPI based version to support nearly every parallel architecture. Due to the disapointing MPI performance on the Cray T3E, we will also implement a Cray SHMEM based version for production runs on this platform. The message passing layer is designed to be thread-safe. A portable object-oriented thread library will be integrated in the near future.

Based on local threads and object-oriented message passing, we propose a second layer with a more abstract interface to hide the message passing and data distribution details from the application programmer. This level will include synchronous and asynchronous remote method invocations as well as distributed shared objects, which are migratable and replicatable.

The second layer will be responsible for load-balancing problems. In our simulation particles move free in space, therefore different particles will interact in each time-step. Access to the particles should be transparent to the application programmer, e.g. a programmer can access a particle as if the particle is in local memory. The access time to the particle of course varies, and will be optimized by the programming system and application library. This layer is scheduled to be implemented by fall this year.

There are numerous reasons, which motivated the redesign of these layers, despite of the existence of object-oriented message passing libraries like MPI++ or MPC++ [12]. The most important are the lack of thread-safe implementations and the missing integration of modern C++ concepts like templates and the support for the standard template library.

Based on the lower-level object-oriented layers, we are developing simulation code and object-oriented parallel application libraries.

5.1 The use of Design Patterns

We cannot ignore the demand for programming in C or FORTRAN. To provide just an implementation in C++ will not be accepted by our project partners. We had to find a way to write down our solutions in a "Meta-Language". Using Design Patterns serves this purpose best. We have an easy to understand way to document our solutions, that is not bound to any programming language. Also using UML allows us to use tools to implement the documented Design Patterns in e.g. C++ (almost) automatically.

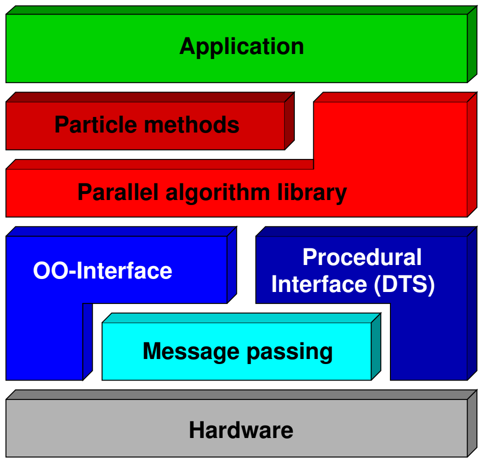

Fig. 4. Layers of the parallel programming system

Documenting the Design Patterns in a modern, easy to read way was achieved by using multi-frame HTML documents. Solutions that are easy to read and understand find generally better acceptance, even if there is no direct implementation in the favorite language of the programmer, e.g. FORTRAN. Also documenting the simplicity of our ready to use solutions motivates more physicists to take a look at a new programming paradigm (some even take a second look).

5.2 An object-oriented Application Library for Parallel SPH Simulations

As a first step towards an object-oriented SPH program, we used an easy to parallelize Monte Carlo simulation of the pulsar HER-X1. Looking at the problem as a programmer the Monte Carlo simulation and the SPH simulation are similar, because they are both particle simulation methods. We used design patterns to describe the solutions that were implemented in the class library. For algorithmic problems like coordinate transformation or different types of integrators we used a Strategy Design Pattern. The simulation data is created using Factory Design Pattern and the individual sub-domains of our simulation domain are connected with each other by being part of a Composite Design Pattern (see 5).

Prototyping a new simulation using this class library is faster than writing new C or FORTRAN code. Currently we are working on optimizing the performance of our C++-programs. We could get some positive feedback from our physics partners with respect to using these solutions for prototyping.

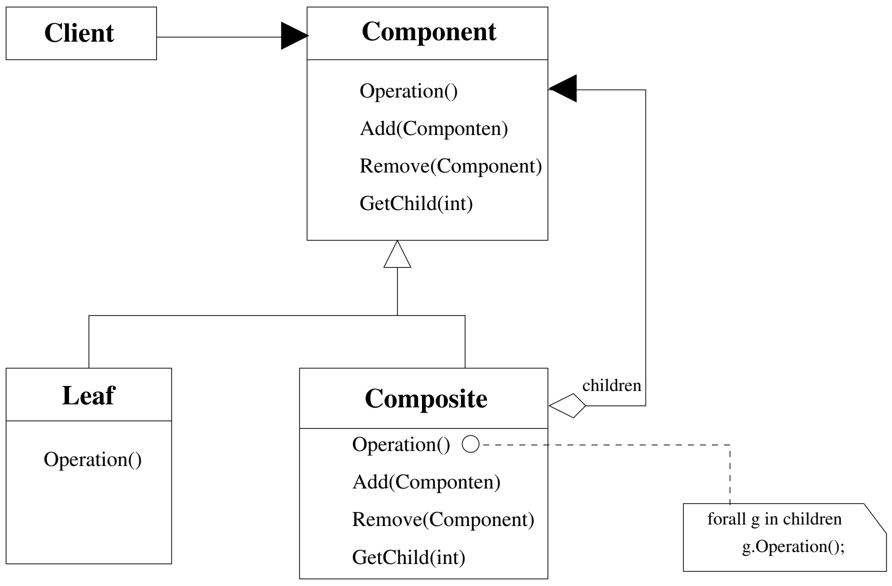

Fig. 5. UML notation of Composite Design Pattern used for Domain-Decomposition

The special SPH data structures and algorithms are also part of our current work — e.g. nearest neighbor problems.

6 Outlook

Fluid dynamics calculated with modern particle methods remains an important topic not only in the area of astrophysics but increasingly in engineering like combustion engines. A future version of our software will include online visualization on distributed memory parallel machines which requires rather good I/O and interactive use capabilities of the employed hardware.

7 Acknowledgments

We wish to thank our project partners from the SFB 382 for the close and fruitful coorporation. Especially we would like to mention the very helpful staff at the HLRS in Stuttgart which put in a lot of effort in reducing the I/O bottleneck during the recent months.

References

1. On subharmonic solutions of a hamiltonian system. *Comm. Pure Appl. Math.*, 33:609–633, 1980.
2. Tilmann Bubeck. *Distributed Threads System DTS User's Guide.* SFB 382/C6, Universität Tübingen, September 1995.
3. Tilmann Bubeck, Wolfgang Küchlin, and Wolfgang Rosenstiel. Symmetric Distributed Computing with Dynamic Load Balancing and Fault Tolerance. In Boleslaw Szymanski, editor, *Proceedings of the 3rd workshop on Languages, Compilers and Run-Time Systems for Scalable Computers*, Troy, New York, May 1995. Kluwer.
4. F. Clarke and I. Ekeland. olutions périodiques, du période donnée, des équations hamiltoniennes. *Note CRAS Paris*, 287:1013–1015, 1978.
5. F. Clarke and I. Ekeland. Nonlinear oscillations and boundary-value problems for hamiltonian systems. *Arch. Rat. Mech. Anal.*, 78:315–333, 1982.
6. R.A. Gingold and J.J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Mon. Not. R. astr. Soc.*, 181:375–389, 1977.
7. Leon B. Lucy. A Numerical Approach to Testing the Fission Hypothesis. *Astron. J*, 82(12):1013–1924, December 1977.
8. R. Michalek and G. Tarantello. Subharmonic solutions with prescribed minimal period for nonautonomous hamiltonian systems. *J. Diff. Eq.*, 72:28–55, 1988.
9. Marcus Ritt. Parallelisierung der berechnung des faltenwurfs von textilien mit partikelsystemen. Master's thesis, Universitt Tbingen, Technische Informatik, 1997.
10. G. Tarantello. Subharmonic solutions for hamiltonian systems via a $\mathbb{Z}_p$ pseudoindex theory. Annali di Matematica Pura, (to appear).
11. T.Bubeck, M.Hipp, S.Hüttemann, S.Kunze, M.Ritt, W.Rosenstiel, H.Ruder, and R.Speith. Parallel SPH on Cray T3E and NEC SX-4 using DTS. In W.Jäger E.Krause, editor, *High Performance Computing in Science and Engineering '98*, pages 396 – 410. Springer, 1999.
12. Gregory V. Wilson and Paul Lu, editors. *Parallel Programming using C++*. The MIT Press, Cambridge, 1996.
13. H. Yserentant. A particle method of compressible fluids. *Numer. Math.*, 76:111–142, 1977.