Available online at www.sciencedirect.com
SCIENCE DIRECT
Journal of Computational Physics 215 (2006) 566–588
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

PPM – A highly efficient parallel particle–mesh library for the simulation of continuum systems

I.F. Sbalzarini, J.H. Walther, M. Bergdorf, S.E. Hieber, E.M. Kotsalis, P. Koumoutsakos *

Computational Science and Engineering Laboratory, ETH-Zurich, CH-8092, Switzerland

Received 4 May 2005; received in revised form 31 October 2005; accepted 7 November 2005
Available online 5 January 2006

Abstract

This paper presents a highly efficient parallel particle–mesh (PPM) library, based on a unifying particle formulation for the simulation of continuous systems. In this formulation, the grid-free character of particle methods is relaxed by the introduction of a mesh for the reinitialization of the particles, the computation of the field equations, and the discretization of differential operators. The present utilization of the mesh does not detract from the adaptivity, the efficient handling of complex geometries, the minimal dissipation, and the good stability properties of particle methods.

The coexistence of meshes and particles, allows for the development of a consistent and adaptive numerical method, but it presents a set of challenging parallelization issues that have hindered in the past the broader use of particle methods. The present library solves the key parallelization issues involving particle–mesh interpolations and the balancing of processor particle loading, using a novel adaptive tree for mixed domain decompositions along with a coloring scheme for the particle–mesh interpolation.

The high parallel efficiency of the library is demonstrated in a series of benchmark tests on distributed memory and on a shared-memory vector architecture. The modularity of the method is shown by a range of simulations, from compressible vortex rings using a novel formulation of smooth particle hydrodynamics, to simulations of diffusion in real biological cell organelles.

The present library enables large scale simulations of diverse physical problems using adaptive particle methods and provides a computational tool that is a viable alternative to mesh-based methods.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Parallel library; Particle–mesh; Vortex methods; Smooth particle hydrodynamics

1. Introduction

A large number of physical problems can be modeled using particle-based methods. Particle descriptions can be used for the simulation of continuum systems as in the case of discrete fluid or solid elements in smooth

* Corresponding author. Tel.: +41 1 632 5258; fax: +41 1 632 1703.
E-mail addresses: sbalzarini@inf.ethz.ch (I.F. Sbalzarini), walther@inf.ethz.ch (J.H. Walther), bergdorf@inf.ethz.ch (M. Bergdorf), hiebers@inf.ethz.ch (S.E. Hieber), kotsalie@inf.ethz.ch (E.M. Kotsalis), petros@ethz.ch (P. Koumoutsakos).

0021-9991/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2005.11.017

particle hydrodynamics (SPH) and vorticity-carrying fluid elements in vortex methods (VM); or for inherently discrete systems as in gravitational particles for astrophysics, dissipative particle dynamics (DPD) for mesoscale polymer descriptions, atomistic molecular dynamics (MD) simulations, and charged particles in plasma physics [1].

While for discrete systems particle descriptions are the method of choice, this is not usually the case for the simulation of continuum systems due to numerical and implementation issues. A number of recent works (see [2] and references therein) have successfully addressed the numerical problems of particle methods and have showcased their unique advantages for the simulation of continuum physical systems. These advantages include adaptivity and multiresolution capabilities of the computational elements, good stability properties of the discretization, and, similar to discrete systems, an inherent link of the computational elements to the physics that they represent.

Nevertheless, the computational implementation of particle methods has prevented their widespread utilization. A key difficulty is associated with their efficient parallelization and the unavailability of relevant scientific libraries that would facilitate this task. In recent years, a number of efficient parallel particle-based codes have been presented, allowing for billion particle simulations in MD and astrophysical SPH simulations. These codes are, however, problem and in some cases machine specific and they do not provide the tools for readily extending, for example, an SPH code to VM. Contrary to mesh based methods, where a number of efficient libraries (e.g. Prometheus [3], Hypre [4]) has been developed, there is a shortage of libraries for particle simulations, preventing the further development and application of the method across different disciplines. The present work, to the best of our knowledge, addresses for the first time this deficiency.

The dynamics of particle methods are governed by the interactions of the N computational particles resulting in an N-body problem with a computational cost that scales nominally as $\mathcal{O}(N^2)$. For short-ranged particle interactions, as in simulations of diffusion [5], the computational cost scales linearly with the number of particles. In the case of long-range interaction potentials such as the Coulomb potential in electrostatics, the gravitational potential in astrophysics, or the Biot-Savart law in VM, fast multipole methods (FMM) [6] reduce the computational cost to $\mathcal{O}(N)$. Alternatively, long-range interactions can be described by equivalent field equations (such as the Poisson equation) that can be solved using meshes, resulting in hybrid particle–mesh (PM) algorithms [7,1]. The computational cost of hybrid methods scales as $\mathcal{O}(M)$, where M denotes the number of mesh points used for resolving the field equations. The choice between FMM and PM techniques is dictated by the boundary conditions of the problem with FMM techniques allowing more flexibility on their specification, while PM schemes are well suited for periodic systems. An important factor, distinguishing FMM and PM techniques, is the parallelization efficiency of these methods, as the mesh regularity of the PM algorithm enables implementations that are typically one or two orders of magnitude faster than corresponding FMM [8,9] implementations. FMM-based particle methods have limited scalability for shared memory systems [10], while their implementation in distributed memory systems is difficult due to the inherent global nature of the underlying tree data structure. It is important to observe, however, that even when FMM are used for the evaluation of the particle interactions, the need for hybrid PM algorithms is imperative in adaptive particle methods such as VM or SPH for the reinitialization of the distorted particle locations [2].

The parallel implementation of PM techniques is hindered by several factors:

- exploiting the symmetry of the particle interactions requires sending back of ghost contributions to the proper real particle,
- the simultaneous presence of particles and meshes prohibits a single optimal way of parallelization,
- complex-shaped computational domains and strong particle inhomogeneities require spatially adaptive domain decompositions,
- particle motion may invalidate the existing domain decomposition causing rising load imbalance, and complicates the implementation of multi-stage integration schemes,
- inter-particle relations constrain decompositions and data assignment.

State-of-the-art particle codes have successfully addressed some of aforementioned parallelization issues as for example the electromagnetic PIC code QUICKSILVER [11], demonstrating a parallel efficiency of 60% solving a scaled-size irregular case on 1024 processors, while achieving 90% efficiency in the ideal uniform load

case on 3200 processors. For purely particle-based simulations a number of application-specific parallel software libraries is also available, such as PARTI for Monte-Carlo simulations [12], or the parallel utilities library (PUL) [13]. Recent hybrid PM implementations include VORPAL [14] for plasma simulations and the particle-in-cell code PICARD [15].

While these codes provide a development platform for certain classes of particle methods, they do not allow generalizations to other classes of particle methods. For example PICARD and VORPAL have been designed for PIC simulations and they can be used for load balancing and domain decomposition. Beyond these two aspects, however, they cannot be used for developing SPH, VM, or MD codes.

The present work bridges the gap between general-purpose infrastructure libraries and application-specific simulation libraries, and provides a general-purpose parallel framework that can handle particles-only, mesh-only, as well as hybrid particle–mesh systems. The presented parallel particle–mesh (PPM) library is portable through the use of standard languages (Fortran 90 and C) and libraries (MPI) and is applicable on single processor machines as well as on distributed memory, shared memory, and vector parallel processors. Computational efficiency is achieved by dynamic load balancing, dynamic particle re-distribution, explicit message passing, and the use of simple data structures. The library core provides several adaptive domain decomposition schemes, multiple processor assignment methods, load balance monitoring, dynamic load balancing, data mapping (sending, receiving), update of overlap regions, parallel file I/O, optimized inter-processor communication, neighbor lists (cell lists and Verlet lists [16]), routines for building trees, particle-to-mesh, and mesh-to-particle interpolation. This core infrastructure is supplemented with commonly used numerical methods such as mesh-based solvers, evaluation of differential operators on particles [17], FMM, parallel FFT, and multi-stage ODE integrators. Moreover, the PPM library provides bindings for the external libraries fftw, Math-Keisan FFT (NEC Inc.), and Metis (for graph partitioning for load assignment [18]).

The library has been successfully used in the development of a number of particle codes enabling state-of-the-art calculations as discussed in Section 4 of this paper. The key concepts of PPM and the implemented functionality are described in Sections 2 and 3. In Section 4, the use of PPM in various areas of computational physics, science, and engineering is demonstrated by presenting prototype simulation codes that have been developed on the basis of PPM. Parallel timings, speedup, and efficiency are shown for each example, demonstrating the high parallel efficiency of the present library.

The library is intended as the founding step in establishing an open source program for the development and implementation of particle methods.

2. Particle–mesh algorithms for the simulation of continuum systems

Particle methods are based on the replacement of functions and differential operators by equivalent integral representations that are being discretized using the particle locations and weights in a quadrature [2]. The simulation of the physical system amounts then to tracking the dynamics of N particles that carry the physical properties of the system that is being simulated. The dynamics of the particles are governed by ordinary differential equations (ODEs) that determine the trajectories of the particles p and the evolution of their properties $\boldsymbol{\omega}$:

$$\frac{\mathrm{d}\boldsymbol{x}_p}{\mathrm{d}t} = \boldsymbol{u}(\boldsymbol{x}_p, t) = \sum_{q=1}^{N} \boldsymbol{K}(\boldsymbol{x}_p, \boldsymbol{x}_q; \boldsymbol{\omega}_p, \boldsymbol{\omega}_q), \quad p = 1, \ldots, N, \tag{1}$$

$$\frac{\mathrm{d}\boldsymbol{\omega}_p}{\mathrm{d}t} = \sum_{q=1}^{N} \boldsymbol{F}(\boldsymbol{x}_p, \boldsymbol{x}_q; \boldsymbol{\omega}_p, \boldsymbol{\omega}_q), \quad p = 1, \ldots, N, \tag{2}$$

where $\boldsymbol{x}_p$ the position of particle p, $\boldsymbol{u}_p$ its velocity, and $\boldsymbol{\omega}_p$ the vector of particle properties such as concentration, charge, vorticity, or temperature. The dynamics of the simulated physical system are represented by the functions $\boldsymbol{K}$ and $\boldsymbol{F}$ that represent solutions of scalar or vector field equations (such as Poisson equations for velocity–vorticity formulations of the Navier–Stokes equations in VM) or integral representations of differential operators (such as Laplacian operators in SPH). In PM methods, the functions $\boldsymbol{K}$ and $\boldsymbol{F}$ are evaluated on a mesh through the corresponding field equation. The hybrid method requires:

- the interpolation of the $\boldsymbol{\omega}_p$ carried by the particles from the irregular particle locations $\boldsymbol{x}_p$ onto the regular mesh points ($\boldsymbol{\omega}_m$)

$$\boldsymbol{\omega}_m = \sum_{p=1}^{N} Q(\boldsymbol{x}_m - \boldsymbol{x}_p)\boldsymbol{\omega}_p, \quad m = 1, \ldots, M, \tag{3}$$

- the interpolation of the field quantities $\boldsymbol{F}_m$ from the mesh to the particle locations ($\boldsymbol{F}_p$)

$$\boldsymbol{F}_p = \sum_{m=1}^{M} R(\boldsymbol{x}_p - \boldsymbol{x}_m)\boldsymbol{F}_m, \quad p = 1, \ldots, N. \tag{4}$$

The accuracy of the method depends on the smoothness of $\boldsymbol{K}$ and $\boldsymbol{F}$, on the interpolation function, and on the discretization scheme employed for the solution of the field equations. To achieve high accuracy, the interpolation functions Q and R must be smooth to minimize local errors, and conserve the moments of the interpolated quantity to minimize far-field errors. In addition, it is necessary that Q is at least of the same order of accuracy as R, to avoid spurious forces [1]. This can be easily achieved by selecting the same type of interpolation, W, for both operations: $Q = R = W$.

3. The PPM library

3.1. Fundamentals

The use of the PPM library requires that the simulated systems are formulated in the framework of PM algorithms as outlined in the preceding section. The field equations are solved using structured or uniform Cartesian meshes. As a result, the physical and computational domains are rectangular or cuboidal in two and three dimensions. Complex geometries are handled by immersed boundaries, through the use of source terms in the corresponding field equations, or through boundary element techniques. Adaptive meshing capabilities are possible using AMR concepts as adapted to particle methods [19].

The simultaneous presence of particles and meshes requires different concurrent domain decompositions. These decompositions divide the computational domain into a minimum number of cuboidal *sub-domains* with sufficient granularity to provide adequate load balancing. The concurrent presence of different decompositions allows to perform each step of the computational algorithm in its optimal environment with respect to load balance and the computation-to-communication ratio. For the actual computations, the individual sub-domains are treated as independent problems and extended with *ghost mesh layers* and *ghost particles* to allow for communication between them.

The PPM library supports *connections*/*relations* between particles such as particle pairs, triplets, quadruplets, etc. These relations may describe a physical interaction, such as chemical bonds in molecular systems, or a spatial coherence, such as a triangulation of an immersed boundary or an unstructured mesh.

Memory for internal lists and communication buffers is allocated by the PPM library. All other memory, such as simulation data (particles, fields) and index lists (cell lists, Verlet lists, etc.), is held by the client application. This ensures user-control over the data and allows multiple different sets of particles, connections, and fields to be used concurrently. The number of topologies, sub-domains, particle sets, fields, and meshes is only limited by the cumulative memory capacity of all processors.

3.2. Topologies

A *topology* is defined by the decomposition of space into sub-domains with the corresponding boundary conditions, and the assignment of these sub-domains onto processors. Multiple topologies may co-exist and library routines are provided to *map* particle and field data between them (cf. Section 3.3). Fields are defined on *meshes*, which in turn are associated with topologies. Every topology can hold several meshes. The only constraint is that sub-domain boundaries must align with mesh lines/planes.

As the domain decomposition may take several seconds to complete, a given topology is assumed to persist through longer periods of the simulation. For problems with free-space boundary conditions, the extent of the

computational domain is adjusted in order to enclose all particles at any time. An extra margin may be added to the computational domain to avoid repeated update of the topology. For problems in confined systems, subject to, e.g. periodic boundary conditions, the extent of the computational domain is fixed and the decomposition is performed filling the entire space, disallowing void space(s). This assures that particles cannot leave the computational domain, which would require an immediate, potentially expensive, re-decomposition.

In order to achieve good *load balance*, both the load distribution and the computational cost of the topology creation are monitored throughout the simulation. The SAR heuristic [12] is used in the PPM library to decide when problem re-decomposition is advised, i.e., when the cost of topology re-definition is amortized by the gain in load balance. Moreover, all topology definition routines can account for the true computational cost of each particle, for example defined by the actual number of its interactions. A routine is provided to compute this number based on the lengths of Verlet lists.

3.2.1. Domain decompositions

The PPM provides a number of different adaptive domain decomposition techniques for particles, meshes, and volumes, the latter defining geometric sub-domains with neither meshes nor particles present. These decompositions currently include: recursive orthogonal bisection, x-, y-, and z-pencils, xy-, xz-, and yz-slabs, cuboids, and a user-defined decomposition. Recursive orthogonal bisection is based on an adaptive binary tree (cf. Section 3.9), where subdivisions are allowed in all spatial directions. Pencil decompositions prohibit subdivisions in one direction, resulting in an adaptive decomposition where each sub-domain extends over the whole computational domain in at least one spatial dimension. Such decompositions are useful when performing fast Fourier transforms. In slabs, two directions are fixed. Cuboids are created using adaptive quad- and oct-trees in two and three dimensions, respectively, and the user-defined decomposition allows the client program to explicitly specify the sub-domains. After checking the validity of such a decomposition, the PPM library directly proceeds with assignment of the sub-domains to the processors.

In addition, a special null decomposition is provided, that does not perform any domain decomposition. It creates only one ''sub-domain'' which is the computational domain itself. This trivial ''decomposition'' is used to evenly distribute the particles among processors, irrespective of their spatial location. The resulting special topology is called the *ring topology* and the sub-domain is assigned to every processor. The ring topology supports full $\mathcal{O}(N^2)$ calculations, and also allows to distribute data of initially unknown processor affiliation (cf. Sections 3.3 and 3.4).

To assess the performance of the different domain decomposition schemes, we compare them on four test cases using 16 processors (Table 1). The quality of decomposition is quantified by the standard deviation of the number of particles across processors and by the total number of ghost particles needed to communicate the boundaries. The domain is decomposed using a non-adaptive binary tree, recursive orthogonal bisection (ROB), and an adaptive oct-tree. The subdomains are assigned onto the processors in an optimal way, minimizing the total length of communication boundaries. This assignment is performed using the external library Metis [18]. One million particles are distributed in the unit cube in four ways: uniformly, on a diagonal from

Table 1
Comparison of different domain decomposition schemes on four test problems

Particle distribution	Non-adaptive tree	ROB	Adaptive oct-tree
Standard deviation of particles per processor			
Uniform	422	268	265
Sphere	62,501	1865	2626
Spiral	73,350	2336	6011
Diagonal line	108,255	148	161
Average number of ghost particles per processor			
Uniform	33,847	33,832	33,750
Sphere	35,268	36,526	28,187
Spiral	10,584	31,102	22,297
Diagonal line	20,050	28,940	46,232

the point (0,0,0) to the point (1,1,1), on the surface of a sphere with radius 0.25, and on a spiral. The computational time needed to construct the topologies is about 30 ms per subdomain in all cases.

3.2.2. Assignment of sub-domains onto processors

Load balancing in the PPM library comprises two main components: domain decomposition and assignment of sub-domains onto processors. While the former has to ensure sufficient granularity and partitioning of the computational cost, the latter has to ensure even distribution of computational load among processors, accounting for possible differences in processor speeds. The computational cost for each sub-domain, as determined by the number of particles, the number of mesh points, or the true computational cost, is known from the domain decomposition step. The individual processor speeds are measured internally by the PPM library.

Using this information, PPM provides several methods of assigning the sub-domains to the processors. The PPM-internal method assigns contiguous blocks of sub-domains to processors until the accumulated cost of a processor is greater or equal to the theoretical average cost under uniform load distribution. The average is weighted with the relative processor speeds. In addition, four different Metis-based [18] assignments, and a user-defined assignment are available.

3.2.3. Boundary conditions

At the external boundaries of the computational domain Neumann, Dirichlet, free space, symmetric, and periodic boundary conditions are supported. These conditions complement the particular mesh-based solver that is being employed. More involved boundary conditions and complex boundary shapes are represented inside the computational domain by defining connections among the particles, or using immersed interfaces.

3.3. Mapping

PPM topologies implicitly define a data-to-processor assignment. *Mapping* routines provide the functionality of sending particles and field blocks to the proper processor, that is the one that ''owns'' the corresponding sub-domain(s) of the computational space. Three different mapping types are provided for both particles and field data:

1. a *global* mapping which involves an all-to-all communication,
2. a *local* mapping for neighborhood communication, and
3. *ghost* mappings to update the ghost layers.

In addition, a special *ring shift* mapping is provided for particle data on the ring topology, and a *connection mapping* is provided for taking into account links between particles.

The global mapping is used to perform the initial data-to-processor assignment or to switch from one topology to another, whereas the local mapping is mainly used to account for particle motion during a simulation. Communication is scheduled by solving the minimal edge coloring problem using the efficient approximation algorithm by Vizing [20–22]. Ghost mappings are provided to receive ghost particles or ghost mesh points, or to send ghost contributions back to the corresponding real element, for example after a symmetric particle–particle interaction or a particle-to-mesh interpolation. The ring shift mapping sends data-sets around all processors, while each processor keeps a local copy of its original data. After every ring shift, each processor can perform its operations using the original local data-set as well as the current traveling set. During a complete cycle, all possible pair interactions will thus be considered. Finally, connection mappings are provided to distribute connections among processors according to an existing distribution of particles, and to update connection lists when particles have moved across processor boundaries.

All mapping types are organized as stacks. A mapping operation consists of 4 steps: (1) defining the mapping, (2) pushing data onto the send stack, (3) performing the actual send and receive operations, and (4) popping the data from the receive stack. This architecture allows data stored in different arrays to be sent together to minimize network latency, and mapping definitions to be re-used by repeatedly calling the push/send/pop sequence for the same persistent mapping definition.

Mappings of field data can be masked. An optional binary mask selects which mesh points are to be mapped and which ones are not. The values of non-mapped points remain unaffected by the mapping operation.

3.4. Particle–particle interactions

The evaluation of particle–particle (PP) interactions is a key component of PM algorithms. Sub-grid scale phenomena can require local particle-based corrections, differential operators can be evaluated on irregular locations [17], or the main dynamics of the system can be governed by particle interactions.

The PPM library implements PP computations using cell lists, Verlet lists, or the full $\mathcal{O}(N^2)$ direct method. Both symmetric and non-symmetric interactions are supported, the former to reduce the amount of duplicated work. In each method, the interaction potential or kernel can be specified either by a function pointer to a user function, by passing a look-up table of kernel values, or by choosing one of the predefined PPM-internal kernels.

The direct evaluation makes use of the PPM ring topology (cf. Section 3.2.1) and the ring shift mapping (cf. Section 3.3) to compute all N^2 pair interactions. Cell lists are provided for local (short range) interactions. Hereby, particles are sorted into equisized cuboidal cells, whose size reflects the interaction cutoff. In PPM, cell lists are defined per sub-domain and *ghost cells* are used around each sub-domain. Fig. 1 illustrates the cell–cell interactions in asymmetric and symmetric evaluations. To achieve complete symmetry, a novel interaction scheme involving diagonal interactions is introduced. This scheme reduces the amounts of memory overhead and communication for symmetrically evaluated particle interactions by 33% in two dimensions and 40% in three dimensions. Given the cells are numbered in ascending x, y, (z), starting from the center cell with number 0, the cell–cell interactions in PPM are: 0–0, 0–1, 0–3, 0–4, and 1–3 in two dimensions, and 0–0, 0–1, 0–3, 0–4, 0–9, 0–10, 0–12, 0–13, 1–3, 1–9, 1–12, 3–9, 3–10, and 4–9 in three dimensions. The difference between symmetric and non-symmetric PP interactions is measured using the PSE diffusion problem (cf. Section 4.3). The computational time per time step is found to decrease by a factor of 1.72 when going from asymmetric to symmetric interactions. Due to the additional overhead caused by sending back the ghost contributions, this factor is below 2.

For spherically symmetric interactions, cell lists contain up to $27/(4\pi/3) = 81/(4\pi) \approx 6$ times more particles than actually needed. Verlet lists [16] are provided to reduce this overhead. For each particle they involve an explicit list of all other particles it has to interact with.

Besides PP interactions, PPM also supports interactions based on inter-particle connections. Neighbor lists are not required in this case since a connection is an explicit list of all its member particles. Connection interactions are supported by separate routines.

Alternatively, the client program can implement its own interaction routines. Template subroutines are provided for the use of cell lists, Verlet lists, direct interactions, and connection interactions.

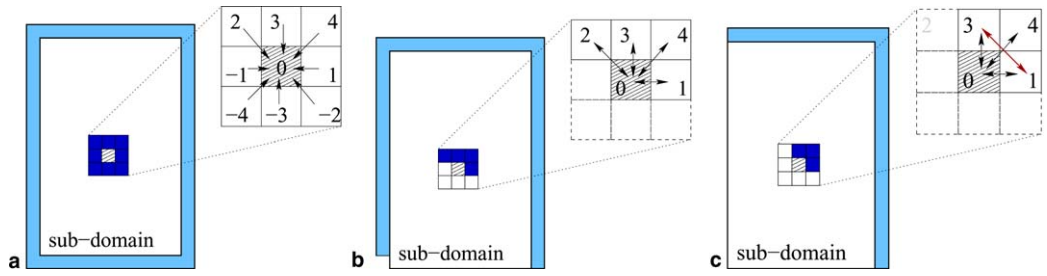

Fig. 1. Cell–cell interactions and ghost-layer arrangement. (a) For non-symmetric particle–particle interactions, the ghost layer (blue) extends all around the sub-domain. Interactions are one-sided. (b) In traditional symmetric cell list algorithms, ghost layers are required on all but one boundary of the domain. (c) In PPM, diagonal interactions are introduced (1–3). Ghost layers are now symmetric and do not overlap with any other ghost layers of neighboring sub-domains. This results in less communication, better scaling in memory and simpler algorithms (e.g. when considering connected particles). The two-dimensional case is depicted. See text for interactions in the three-dimensional case. (For interpretation of the references to colors in this figure legend, the reader is referred to the web version of this paper.)

In addition to the routines performing the actual computations, the PPM library also provides a routine to create look-up tables from either a function pointer or an internal kernel. Such tables can then be passed to any of the compute routines for the evaluation.

3.5. Particle–mesh and mesh–particle interpolations

All hybrid PM methods involve interpolation of irregularly distributed particle quantities from particle locations onto a regular mesh and interpolation of field quantities from the grid points onto particle locations.

These interpolations are utilized for two purposes, namely:

- the communication of the particle solver with the field solver;
- the reinitialization of distorted particle locations.

While the first issue is a well-established notion in PM techniques, the *reinitialization* of particle locations and weights when particle locations get distorted by the flow map is a critical, albeit often overlooked, aspect of particle methods for the simulation of continuous systems [2]. Particle overlap is needed in order to ensure convergence of the method and this is achieved by periodically reinitializing particles onto a regular mesh (''remeshing''). This involves the interpolation of particle properties onto the mesh and replacing the current set of particles by new particles created at the locations of the mesh points.

The PPM library provides routines that perform these operations. The interpolation weights $W(\boldsymbol{x}_m - \boldsymbol{x}_p)$ can be pre-computed and stored to facilitate adjustments of the interpolation or interpolate several sets of quantities. If the weights are not pre-computed, they are determined during the actual interpolation. Currently, implemented interpolants include first and second order B-Splines and the M'_4 function [23].

The interpolation of mesh values onto particle locations readily vectorizes: the interpolation is performed by looping over the particles and receiving values from mesh points that lie in the support of the interpolation kernel. Therefore, the values of individual particles can be interpolated independently.

The interpolation of particle values onto mesh locations, however, leads to data dependencies as the interpolation is still performed by looping over particles, but a mesh point may receive values from more than one particle. To circumvent this problem, the PPM library implements the following technique [24]: when new particles are created in the course of remeshing, we assign colors to the particles such that no two particles within the support of the interpolation kernel have the same color. Particle-to-mesh interpolation then visits the particles ordered by color to achieve data independence. This coloring scheme enables vectorization of particle-to-mesh interpolations as confirmed by a test on the NEC SX-5 vector computer (Table 2). Without this coloring scheme, interpolation in hybrid particle–mesh methods would be prohibitively expensive on vector architectures.

3.6. Mesh-based solvers

In PPM, meshes can be used to solve the field equations associated with long-range particle interactions [1] or to discretize the differential operators in the governing equations of the simulated physical system. These operators are often local and their computational cost scales linearly with the number of particles or mesh points.

A large class of pair potentials in particle methods can be described by the Poisson equation as it appears in MD of charged particles via electrostatics (Coulomb potential), fluid mechanics in stream-function vorticity

Table 2
Comparison of the vector performance of classical particle-to-mesh interpolation and the present coloring scheme

	CPU time (s)	Vector operation ratio (%)	Vector length (words)
Colored	2.69	99	230.6
Classical	30.1	0.36	4.1

formulation (Biot-Savart potential), and astrophysics (gravitational potential). The Poisson equation is expressed as

$$\nabla^2 \Phi = \rho(x, y, z). \tag{5}$$

The PPM library provides Poisson solvers based on FFTs and geometric MultiGrid (MG).

The FFT-based Poisson solver parallelizes a multi-dimensional FFT using a sequence of one- or two-dimensional FFTs performed on pencil and slab topologies (cf. Section 3.2.1). A single three-dimensional Fourier transform thus consists of mapping the data onto a temporary xy-slab topology, performing a two-dimensional FFT, mapping onto a temporary z-pencil topology, and performing a one-dimensional FFT. The actual serial one-dimensional or two-dimensional FFTs are performed using the external libraries fftw or Math-Keisan (on NEC SX vector architectures).

The geometric MG method is implemented in PPM as a fast iterative method for solving the Poisson equation. The advantage of parallel MG solvers consists in restricting communication to the ghost layers whereas the corresponding FFTs require several global mappings. The PPM MG supports both the V and W cycle [25]. The Laplacian is discretized using five and seven point stencils in two and three dimensions, respectively. As residual smoother we employ the red-black successive over-relaxation scheme, which includes the Gauss–Seidel smoother as a special case. Furthermore, the full-weighting scheme [25] is used for the restriction of the residual, and bilinear (in two dimensions) or trilinear (in three dimensions) interpolation for the prolongation of the function corrections [25].

3.7. ODE solvers

Simulations using particle methods entail the solution of systems of ODEs as outlined in Section 2. The characteristics of the initial value problems (IVP) represented by these ODEs explicitly reflect the physics of the system that is being simulated.

The PPM library provides a set of explicit integration schemes to solve these IVPs. The ODE solver of PPM is designed as a ''black-box'' solver. The user selects the method to be used and provides as a function pointer a routine that computes the right-hand sides of the ODEs. Both allocation of storage (for the stages of multi-step schemes) and the actual computation of the stages is performed by the library. Second order ODEs are solved by transforming them into a system of first order problems and parallelism is achieved by mapping the integrator stages along with the other particle quantities (cf. Section 3.3). Thus, at the last stage of the integrator, the previous stages are available on the processor that currently hosts the particles, and the final particle update is completed without further communication. Low-storage schemes have the additional advantage of requiring little communication. The set of available integrators includes forward Euler with and without super time stepping [26], 2-stage and 4-stage standard Runge–Kutta schemes, Williamson's low-storage third order Runge–Kutta scheme [27], and 2-stage and 3-stage TVD Runge–Kutta schemes [28].

3.8. Parallel I/O

File I/O in distributed parallel environments exist in two different modes: *distributed* and *centralized*. By distributed we denote the situation where each processor writes its part of the data to its local file system. Centralized I/O on the other hand will produce a single file on one of the nodes, where the data contributions from all processors are stored. The latter is convenient for small or aggregated data, or for writing files that will later be read on a different number of processors, e.g. to continue an interrupted simulation.

The PPM library provides a parallel I/O module which supports both binary and ASCII read and write operations in both modes, distributed and centralized. The I/O mode is transparent to the client application. Write operations in the centralized mode can concatenate or reduce (sum, replace) the data from individual processors; read operations can transparently split the data in equal chunks among processors or send an identical copy to each one. The basic assumption behind the split mode is that no processor will be able to hold all the data in memory at any time. To improve performance of the centralized mode, network communication and file I/O are overlapped in time using non-blocking MPI calls.

3.9. Adaptive trees

A novel general tree construction is provided for both internal and client use. It supports both non-adaptive and adaptive binary trees, quad-trees, and oct-trees. At any stage of the tree, the space is subdivided into M boxes $\{B_k\}$. The indices i and j are used to denote coordinate directions. Adaptivity and subdivision behavior are guided by two *cost functions* ϕ_1 and ϕ_2. Both cost functions are linear combinations of the three cost contributions: particle costs c_p (user-specified or unity per particle), mesh points (number of mesh points in the box $m_B = \prod m_{B,i}$), and geometry (volume of the box $|B| = \prod |B|_i$), with user-provided coefficients α, β, γ

$$\phi_{\{1,2\}}(B_k) = \alpha_{\{1,2\}} \sum_{p \in B_k} c_p + \beta_{\{1,2\}} m_{B_k} + \gamma_{\{1,2\}} |B_k|. \tag{6}$$

The first cost function ϕ_1 guides adaptivity of the tree since the next subdivision will be applied to the box B_K of largest ϕ_1. The second cost function ϕ_2 defines the direction(s) of subdivision and position of the subdivision plane(s). Suppose B_K is to be subdivided next. In order to create the minimum ϕ_2-cut when subdividing a box, the tensor of intertia T is computed from the particle locations $\boldsymbol{x}_p = (x_{p,i})$ and costs c_p as:

$$T_{ii} = \sum_{p \in B_K} \left(\sum_{j \neq i} x_{p,j}^2 \right) c_p, \quad T_{ij} = \sum_{p \in B_K} x_{p,i} x_{p,j} c_p. \tag{7}$$

The eigenvectors $\boldsymbol{v}_r$ of T are scaled with the corresponding eigenvalue λ_r: $\boldsymbol{v}_r = \lambda_r (\boldsymbol{v}_r / |\boldsymbol{v}_r|)$ and projected onto the unit coordinate vectors $\boldsymbol{e}_i$. The number of mesh points in this direction $m_{B,i}$ and the length of the box in this direction $|B|_i$ are normalized and added to form a score value s for each coordinate direction

$$s(\boldsymbol{e}_i) = \alpha_2 \sum_r \boldsymbol{e}_i \cdot \boldsymbol{v}_r + \beta_2 \left(\sum_j m_{B_K,j} \right)^{-1} m_{B_K,i} + \gamma_2 \left(\sum_j |B_K|_j \right)^{-1} |B_K|_i. \tag{8}$$

The subdivision directions (1, 2, or 3) are chosed in order of ascending score. The client program can however specifically disallow subdivisions in certain directions to enforce pencil-type or slab-type boxes. The actual position of a cut perpendicular to direction I is determined as the corresponding component of the center of mass of ϕ_2 within the box B_K

$$\phi_2(B_K)^{-1} \left[\alpha_2 \sum_{p \in B_K} x_{p,I} c_p + \mu_I(B_K)(\beta_2 m_{B_K} + \gamma_2 |B_K|) \right], \tag{9}$$

subject to the constraint that a client-specified minimum box size is not under-run. Here, $\boldsymbol{\mu}(B_k) = (\mu_i(B_k))$ denotes the geometric center of box B_k. To terminate the tree, multiple concurrent termination criteria can be prescribed.

4. Benchmarks and results

The parallel efficiency of the library is measured based on the following five tests:

1. the solution of the scalar Poisson equation using FFTs,
2. the solution of the scalar Poisson equation using geometric MG,
3. simulation of protein diffusion in the endoplasmic reticulum (ER) of live cells,
4. simulation of compressible viscous flow induced by a double shear layer using remeshed smooth particle hydrodynamics (rSPH), and
5. simulations of an incompressible viscous double shear layer using VM.

In addition, we illustrate the application of the library to problems in fluid dynamics with the simulation of a compressible vortex ring using rSPH and with the simulation of Crow and elliptic instabilities of trailing anti-parallel vortex tubes in an incompressible fluid using VM.

Performance of the PPM library is tested for both a fixed-size and a scaled-size problem for all cases except the diffusion simulation. In the fixed-size problems, the number of mesh points and particles is kept constant, i.e., the work load per processor decreases with increasing number of processors. In the scaled problems, mesh point and particle numbers grow proportionally to the number of processors, resulting in a constant work load per processor. Timings and parallel efficiency figures are collected on the IBM p690 computer of the Swiss National Supercomputing Centre (CSCS). The machine consists of 8 Regatta nodes with 32 1.3 GHz Power4 processors per node. Within each node, it is configured in 4 groups with 8 processors sharing 12 GB of memory. Each processor has a peak performance of 5.2 GFlop/s, and the nodes are connected by a 3-way Colony switch system.

In each test, we measure the elapsed wall-clock time t_{ij} for each time step j on each processor $i = 1,\ldots,N_{\text{proc}}$. To account for synchronous communication steps, we report the maximum of these times over all processors. This maximum is averaged over 5–10 samples to compute speedup S and efficiency e:

$$S(N_{\text{proc}}) = \frac{t(1)}{\text{mean}_j \max_i t_{ij}(N_{\text{proc}})} \cdot \frac{N(N_{\text{proc}})}{N(1)}, \tag{10}$$

$$e(N_{\text{proc}}) = \frac{S(N_{\text{proc}})}{N_{\text{proc}}}, \tag{11}$$

where $t(1)$ is the time on a single processor (linearly extrapolated if not measured), $t_{ij}(N_{\text{proc}})$ is the time on N_{proc} processors, $N(1)$ is the problem size on a single processor, and $N(N_{\text{proc}})$ is the problem size on N_{proc} processors. To account for the $\mathcal{O}(N \log N)$ scaling of the FFTs, the second factor of the speedup is accordingly adjusted in the benchmarks of the FFT-based Poisson solver.

Vectorization and parallel efficiency on vector machines are tested using the NEC SX-5 computer at CSCS. This is a shared memory machine with 16 NEC SX-5 vector processors. Each processor has a peak performance of 8 GFlop/s and 64 vector registers of a length of 256 words (2048 Bytes) each.

In addition to the benchmark tests, simulations are performed on a distributed memory cluster consisting of 16 2.2 GHz AMD Opteron 248 processors running under Linux. The nodes of this cluster are connected by a switched gigabit ethernet network.

4.1. Parallel FFT-based Poisson solver

We test the parallel performance of the FFT-based Poisson solver and compare it to the MG solver by solving the same scalar Poisson equation (5) with the same right hand side Eq. (12), subject to periodic boundary conditions. All Fourier transforms are performed using the parallel FFT routines of the PPM library as described in Section 3.6.

The parallel speedup and efficiency for the scaled problem as shown in Fig. 2 exhibit two characteristic regions. The first one ranges from 1 to 8 processors, the second one from 8 and beyond. From 1 to 8 processors

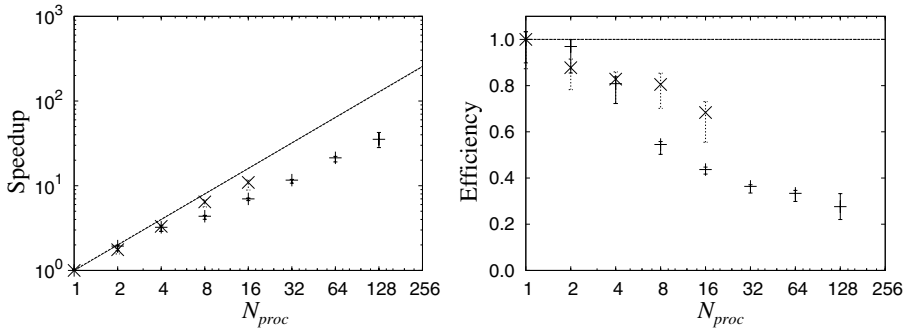

Fig. 2. Parallel speedup and efficiency of the FFT-based Poisson solver for the scaled-size problem starting with $128 \times 128 \times 128$ mesh points on one processor (+). Using only one processor per node, the bottleneck of the shared memory is removed (×). Each point is averaged from 5 samples, error bars indicate min–max span. All timings are performed on the IBM p690.

the efficiency drops significantly, due to conflicts and congestion in the shared memory architecture. This is verified in a separate benchmark (Fig. 2: ×), in which only one processor per node is used. In this case, the congestion is removed and the efficiency significantly improves to 68% on 16 processors for the scaled case. Solving the Poisson equation to machine precision on a $128 \times 128 \times 128$ mesh takes 0.6 seconds on a single processor. The corresponding scaled system on 64 processors ($512 \times 512 \times 512$) requires 2.4 s. Speedup and efficiency for the fixed-size problem are shown in Fig. 3. Again, the scaling improves beyond 8 processors, similar to the scaled case.

4.2. Parallel multigrid Poisson solver

We test the performance of the PPM MG field solver by solving the scalar Poisson equation (5) with the right hand side

$$\rho(x,y,z) = \sin(2\pi x)\sin(2\pi y)\sin(2\pi z), \quad x,y,z \in [0,1], \tag{12}$$

subject to periodic boundary conditions. The initial value of Φ is zero everywhere and we use the V(2,1) cycle with one smoothing step at the finest level.

We conduct three tests. The first involves the fixed case with $256 \times 256 \times 256$ mesh points, while the two others are scaled cases, one starting from a $128 \times 128 \times 128$ mesh, the other one starting from $256 \times 256 \times 256$. Efficiency and speedup for the scaled cases are shown in Fig. 4 and for the fixed case in Fig. 5. We observe a strong decrease in the parallel efficiency up to 8 processors due to the congestion of the shared memory. This is removed when using only one processor per node in a pure distributed memory setup, cf. Figs. 4 and 5, and the efficiency

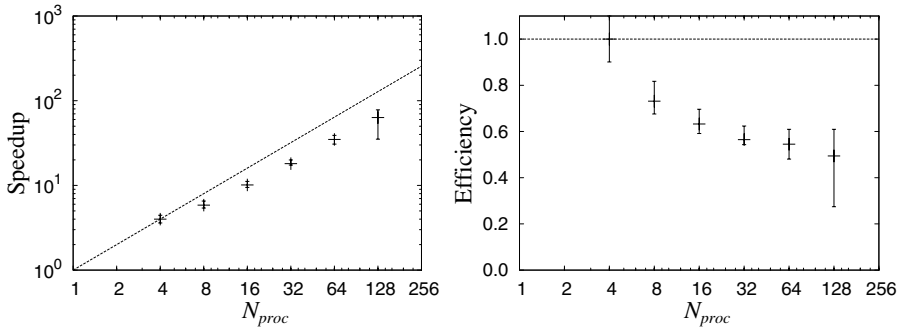

Fig. 3. Parallel speedup and efficiency of the FFT-based Poisson solver for the fixed-size problem with $512 \times 512 \times 512$ mesh points on 4–128 processors. Each point is averaged from 5 samples, error bars indicate min–max span. All timings are performed on the IBM p690.

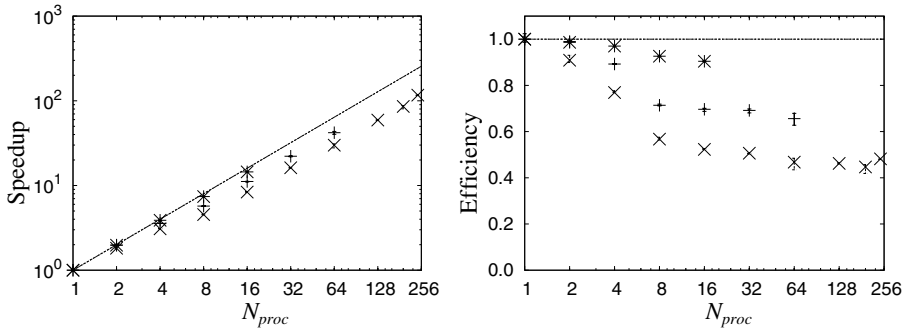

Fig. 4. Parallel speedup and efficiency of the MG Poisson solver for the scaled-size problem. The initial mesh resolution on one processor is $256 \times 256 \times 256$ (+) and $128 \times 128 \times 128$ (×), respectively. Using only one processor per node in the large case, the bottleneck of the shared memory is removed (∗). Each point is averaged from 5 samples, error bars indicate min–max span. All timings are performed on the IBM p690.

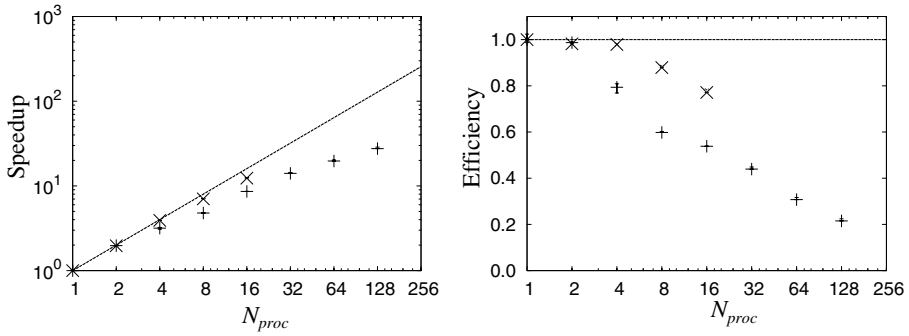

Fig. 5. Parallel speedup and efficiency of the MG Poisson solver for the fixed-size problem with $256 \times 256 \times 256$ mesh points on 2–128 processors (+). Using only one processor per node in the large case, the bottleneck of the shared memory is removed (×). Each point is averaged from 5 samples, error bars indicate min–max span. All timings are performed on the IBM p690.

improves to 90% on 16 processors for the scaled case. The effective efficiency based on the timing obtained on 8 processors is 92% for the large scaled case on 64 processors. The efficiency of the PPM MG solver for a $1024 \times 1024 \times 1024$ system is 66% for the large scaled case on 64 processors. For this system, the elapsed time is 10.5 s per V-cycle, and thus 42 s for the 4 V-cycles needed to reduce the L_2 error to 10^{-4}. A system with half a billion unknowns is solved in the small scaled case on 242 processors at 48% efficiency in 1.7 s per V-cycle. This compares well with the 41% efficiency achieved by the Prometheus multigrid library [29] on 128 IBM Power3 processors for the same problem size.

The vectorization of the PPM MG solver is tested on the NEC SX-5 using up to 8 processors. The PPM MG sustains a performance of 2.4 GFlop/s per processor (30% of peak performance) with a vector operation ratio of 95% and a parallel efficiency of 96%. On this machine, a single V cycle on a $512 \times 512 \times 512$ system takes 1.21 s on 8 processors.

4.3. Diffusion in complex geometries

We present a client application for the simulation of three-dimensional diffusion in the endoplasmic reticulum (ER) of a live cell. This test demonstrates the capability of the library in handling complex-shaped domains, and helps to assess load balance and inter-processor communication for an irregular domain decomposition.

The ER is the major biosynthetic organelle of eucaryotic cells. It consists of a complex network of sheets and reticulated tubules, enveloped by a single lipid bilayer. We simulate a fluorescence recovery after photobleaching (FRAP) [30] experiment in ER geometries reconstructed from live cells. In such an experiment, green fluorescent protein is expressed and retained in the ER lumen. After equilibration, a specific portion of the ER is bleached using laser light. Recovery of fluorescence by diffusion of non-bleached protein into this volume is monitored over time. This method is used in cell biology to measure molecular diffusion constants of protein species in different compartments of live cells.

In order to identify this diffusion constant, it is necessary to postulate a diffusion equation along with suitable boundary conditions. In the present case, we consider isotropic homogeneous diffusion with diffusion constant D

$$\frac{\partial c(\boldsymbol{x},t)}{\partial t} = D\nabla^2 c(\boldsymbol{x},t) \tag{13}$$

and homogeneous Neumann boundary conditions. In the context of the particle strength exchange (PSE) [5], the Laplacian is approximated by an integral operator which is in turn discretized using the particle locations as quadrature points. Particles carry concentration as their scalar property ω and the resulting system of ODEs is given in Section 2 with $\boldsymbol{K} \equiv 0$ and $\boldsymbol{F} = \frac{D}{\sigma^2}\zeta_\sigma(\boldsymbol{x}_q - \boldsymbol{x}_p)h^3(\omega_q - \omega_p)$, where h is the inter-particle distance. The accuracy of this approximation is related to the moment properties of the three-dimensional kernel function $\zeta_\sigma = \sigma^{-3}\zeta(\boldsymbol{x}/\sigma)$. In the present study, we use the second order accurate kernel proposed by Cottet [31]

$$\zeta(\boldsymbol{x}) = \frac{15}{\pi^2}\frac{1}{|\boldsymbol{x}|^{10}+1}. \tag{14}$$

The ODEs are integrated in time using the explicit Euler scheme.

4.3.1. Parallel speedup, timing and efficiency

The problem size is fixed at 3.4 million particles, uniformly distributed inside the ER geometry. The simulation uses Verlet lists and a cutoff of 2σ for the particle–particle interactions. Each particle thus interacts with 32 neighbors. Domain decomposition is done using adaptive recursive orthogonal bisection (cf. Section 3.9) with the z direction fixed as the organelle is very thin in the z direction. Fig. 6 shows the resulting decomposition into 9311 sub-domains. The elongated domains at the periphery are a result of the recursive orthogonal bisection domain decomposition.

Parallel performance is tested on 4–242 processors. Fig. 7 summarizes the results. The simulation sustains 20% of the peak performance on the IBM p690, thus reaching a total of 250 GFlop/s on 242 processors at 84% efficiency. The load balance is quantified by

$$\frac{\text{mean}_j \min_i t_{ij}(N_{\text{proc}})}{\text{mean}_j \max_i t_{ij}(N_{\text{proc}})}. \tag{15}$$

If we use the actual number of interactions of each particle as that particle's computational cost for the topology creation (cf. Section 3.2), we observe values in the range of 90–95%. Using an assumed unit cost for every particle, the load balance is on the order of 10–60%, depending on the actual number of processors used.

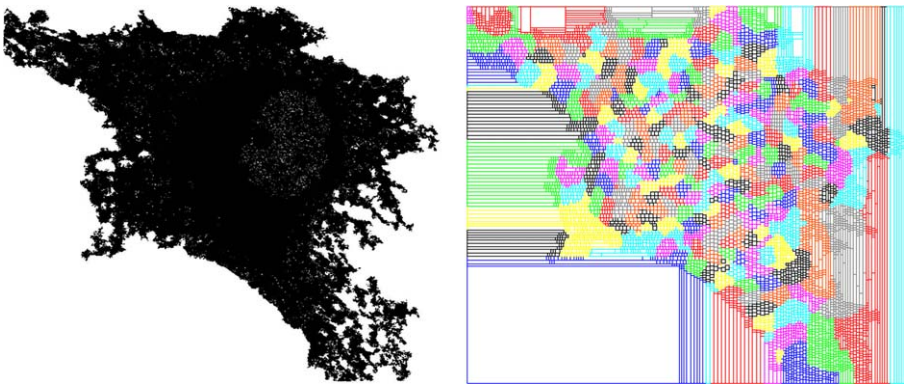

Fig. 6. Top view of the computational domain (left panel) used for this PSE test case and the resulting PPM domain decomposition (right panel) on 242 processors using recursive orthogonal bisection in x and y directions (z direction fixed). Rectangles show the 9311 sub-domains, color codes processor affiliation. The peripheral elongated domains are a result of the recursive orthogonal bisection decomposition.

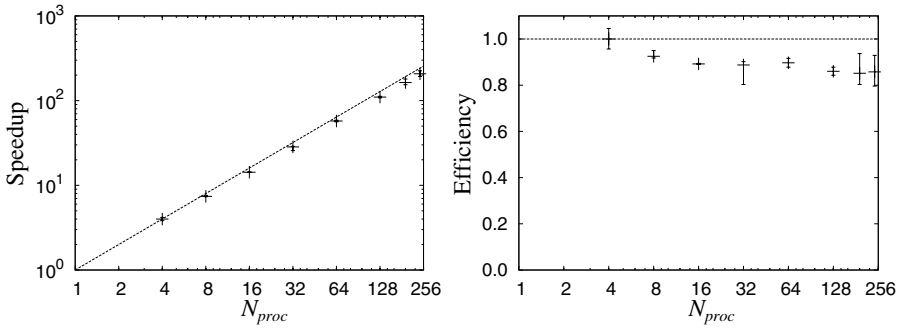

Fig. 7. Parallel speedup and efficiency of the PPM PSE client for the fixed-size problem with 3.4 million particles on 4–242 processors. Each point is averaged from 10 samples, error bars indicate min–max span. All timings are performed on the IBM p690.

A visualization of the simulation result is shown in Fig. 8. Comparing such simulations of diffusion to real FRAP experiments allows to measure molecular diffusion constants in living cells. The molecular diffusion constant of ssGFP-KDEL in the ER lumen of VERO cells is determined in 8 different ER samples as 34 ± 0.95 μm^2/s, which is in good agreement with the value of 30 μm^2/s determined by Weiss et al. [32] using fluorescence correlation spectroscopy for the closely related ssYFP-KDEL in the ER lumen of HeLa cells.

On the NEC SX-5 vector computer, more than 99% of the loops in non-initialization routines vectorize and the average vector length is >254 words. The parallel efficiency is 88% on 8 processors and 86% on 15 processors. One time step in the latter case takes 1.15 s. 2.64 GFlop/s are sustained on each of the 15 processors thus reaching 33% of the machine's peak vector performance. Again using the actual number of interactions (i.e., the length of the Verlet list) as the computational cost for each particle, the load balance exceeds 80% on up to 15 processors.

The largest simulation performed using this PPM client uses 1 billion particles. The computation is based on cell lists and a cutoff of 1σ, i.e., each particle interacts with 26 neighbors. The simulation is performed on 64 processors of the IBM p690, takes 54 s per time step, and sustaines 20% of peak performance. Extrapolating from the previous runs, 50–60 s per time step are expected for this large simulation. The measured 54 s fall within this range, showing linear scaling to large numbers of particles.

4.4. Three-dimensional remeshed smooth particle hydrodynamics

We present a client application based on a novel, computationally efficient formulation of the remeshed SPH [33]. The rSPH client is applied to the simulation of a three-dimensional compressible double shear layer [34]. In order to measure the parallel performance, we consider a computational domain fully populated with particles so that the reported performance measures are independent of the particular flow problem. We furthermore present results from the application of the present rSPH methodology to the evolution of a compressible vortex ring, demonstrating the stability and accuracy of the method.

The three-dimensional Navier–Stokes equations for viscous compressible isothermal flow in non-dimensional Lagrangian form are expressed as

$$\frac{\mathrm{D}\rho}{\mathrm{D}t} = -\rho\nabla\cdot\boldsymbol{u} \tag{16}$$

and

$$\rho\frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t} = -\frac{1}{M^2\gamma}\nabla p + \frac{1}{Re}\nabla\cdot\boldsymbol{\tau}, \tag{17}$$

where

$$p = T\rho \tag{18}$$

and the components of the stress tensor $\boldsymbol{\tau}$

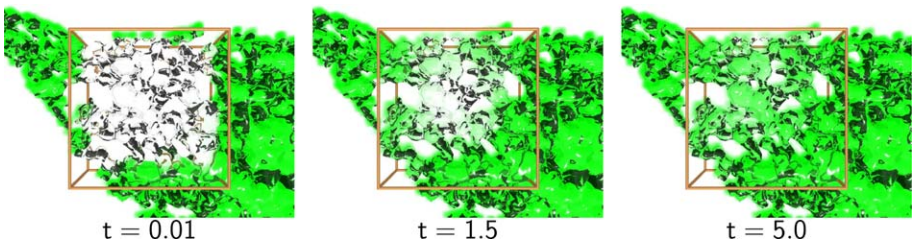

Fig. 8. Snapshots of concentration distribution from a sample PSE simulation. The ER membrane is visualized as a transparent surface and the concentration of green fluorescent protein as a volume density cloud inside it. The bleached region is represented by the outline of a cube. Only the part of the ER around the bleached volume is shown.

$$\tau_{ij} = \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}. \tag{19}$$

δ_{ij} is the Kronecker delta symbol, Re is the Reynolds number, and T the temperature, normalized by the characteristic temperature T_0 of the flow. It is set to $T = 1$ for all simulations (iso-thermal fluid). The Mach number M is defined as $M = \frac{u_0}{\sqrt{\gamma R T_0}}$, where u_0 denotes the characteristic velocity, γ the ratio of specific heats, and R the gas constant. The density ρ is normalized by the mean density ρ_0.

The governing equations (16) and (17) are discretized using the rSPH approach [33]. In this context the particles are reinitialized onto a Cartesian mesh when they cease to overlap. Field quantities $A(x)$ are approximated by a summation of the particle attributes A_b, weighted by the smoothing kernel $W(r,h)$

$$A(\boldsymbol{x}) = \sum_b A_b v_b W(||\boldsymbol{x} - \boldsymbol{x}_b||, h), \tag{20}$$

where v_b is the volume of a particle, $\boldsymbol{x}_b$ is the particle position, and h the smoothing length. Spatial derivatives of the field quantities are approximated by spatial differentiation of Eq. (20). In the present implementation, the summations of the rSPH approximation are grouped and the kernel evaluations are replaced by pre-computed look-up tables in order to obtain a formulation that includes three simple look-up tables Λ_1, Λ_2, and Λ_3, thus,

$$\rho_p \frac{\mathrm{d}\boldsymbol{u}_p}{\mathrm{d}t} = \frac{1}{M^2\gamma}\sum_q \Delta\boldsymbol{x}_{pq}\Delta p_{pq}\Lambda_1 v_q + \frac{1}{3Re}\sum_q \left[\Lambda_3 + \left(\Delta\boldsymbol{x}_{pq}\Delta\boldsymbol{x}_{pq}^{\mathrm{T}}\right)\Lambda_2\right]\Delta\boldsymbol{u}_{pq}v_q. \tag{21}$$

The look-up tables Λ_i are sampled at the distance $\|\Delta\boldsymbol{x}_{pq}\|$ between particles p and q, where $\Delta\boldsymbol{x}_{pq}$ denotes the vector of the Cartesian distance between the particles, $\Delta\boldsymbol{u}_{pq}$ the vector of the velocity difference, and Δp_{pq} the pressure difference. The look-up table values are pre-computed as:

$$\Lambda_1 = \frac{1}{||\Delta\boldsymbol{x}_{pq}||h}\frac{\mathrm{d}W(r,h)}{\mathrm{d}r}\bigg|_{r=||\Delta\boldsymbol{x}_{pq}||}, \tag{22}$$

$$\Lambda_2 = -\frac{1}{||\Delta\boldsymbol{x}_{pq}||^3 h}\frac{\mathrm{d}W(r,h)}{\mathrm{d}r}\bigg|_{r=||\Delta\boldsymbol{x}_{pq}||} - \frac{1}{||\Delta\boldsymbol{x}_{pq}||^3 h}\frac{\mathrm{d}^2W(r,h)}{\mathrm{d}r^2}\bigg|_{r=||\Delta\boldsymbol{x}_{pq}||}, \tag{23}$$

$$\Lambda_3 = 10\Lambda_1 + ||\Delta\boldsymbol{x}_{pq}||^2\Lambda_2, \tag{24}$$

where $W(r,h)$ is chosen to be the quartic spline kernel M_5 [33]. This novel formulation of rSPH has several advantages compared to the explicit calculation of the individual components. Firstly, the evaluation of the right-hand side is reduced from originally five to two summations. Secondly, the kernel evaluations in the summations are avoided by using the look-up tables. Note that the look-up tables are only created once, using the corresponding PPM routine (cf. Section 3.4). Comparing the performance of the look-up table formulation of the SPH to the classical SPH with direct kernel evaluations, we find that the computational time to evaluate the right hand side is reduced by about 20% on the AMD Opteron processor.

The particles are reinitialized (remeshed) after each time step using the M_4' kernel function [23]. Time integration is done with a second-order Runge–Kutta scheme.

For the compressible vortex ring, we use $M = 0.5$, $Re = \Gamma\rho_0/\mu = 3000$, and a computational domain of size $2\times1\times1$. The initial vortex ring is assumed to have a Gaussian distribution of vorticity $\omega = \Gamma/(\pi r_0)\exp(-r^2/r_0^2)$, where $\Gamma = 0.3$, r is the distance to the core of the tube, and $r_0 = 0.025$ the tube radius. The ring radius R_0 is perturbed around a mean value of 0.125 by a truncated Fourier series of amplitude 9×10^{-4} [35]. For the initialization of the velocity field, we assume that the flow is incompressible with an initial unit density field.

4.4.1. Parallel speedup, timing and efficiency

The speedup and parallel efficiency of the rSPH client are shown in Figs. 9 and 10 for the scaled and fixed-size problem, respectively. The largest simulation considered in this rSPH test case comprises 268 million particles and achieves a parallel efficiency of 91% on 128 processors. The efficiency on 32 processors using 67 million particles is also 91%, which compares well with the 85% efficiency of the GADGET SPH code by Springel

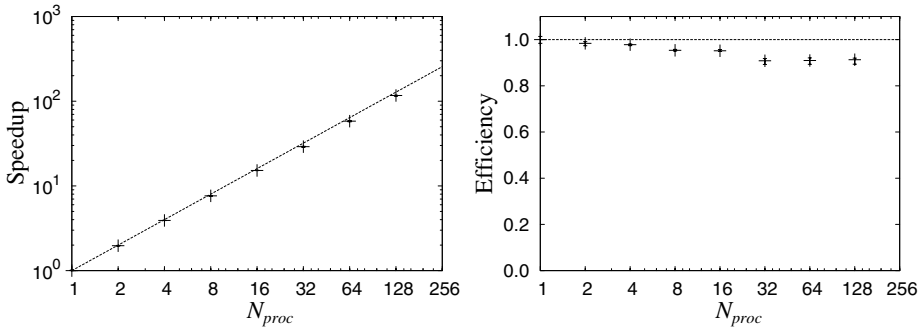

Fig. 9. Parallel speedup and efficiency of the PPM rSPH client for the scaled-size problem starting with 2 million particles on one processor. Each point is averaged from 5 samples, error bars indicate min–max span. All timings are performed on the IBM p690.

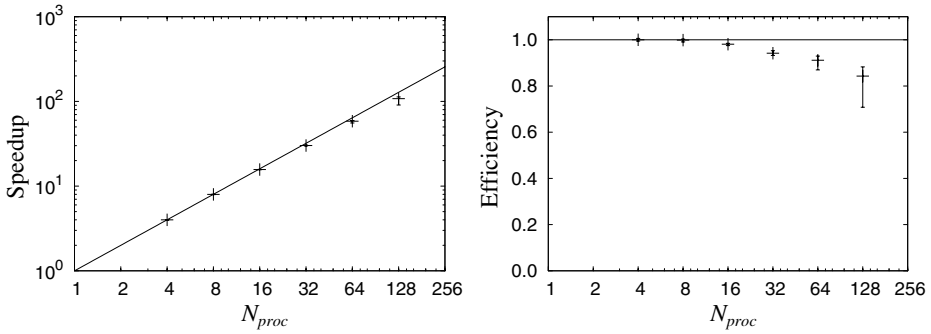

Fig. 10. Parallel speedup and efficiency of the PPM rSPH client for the fixed-size problem with 16.8 million particles on 4–128 processors. Each point is averaged from 5 samples, error bars indicate min–max span. All timings are performed on the IBM p690.

et al. [36] on 32 processors of the same computer model (IBM p690).The efficiency in the fixed-size problem ranges between 100% and 84%. One time step of the simulation using 16.8 million particles takes 196.9 s on 4 processors and 7.3 s on 128 processors.

The result of the vortex ring simulation using 33 million particles distributed onto 16 AMD Opteron processors is shown in Figs. 11 and 12. We use a constant time step of 5×10^{-4}, corresponding to a maximum CFL number of 0.5. The velocity profile of the vortex ring creates a density profile that has its minimum at the core of the ring. The density field evolves to create an accumulation of mass around the ring, resulting in pressure waves that travel through the system, re-entering through the periodic boundary downstream of the vortex ring. Interferences with the vortex ring create additional pressure waves that decay over time (Fig. 11). Iso-surfaces of vorticity at corresponding times are shown in Fig. 12.

The propagation speed of the compressible vortex ring is 0.48, which is within 4% of the analytical solution [37], including corrections for compressibility [38].

The communication overhead of the present rSPH client is assessed using 16 million particles. The fraction of time spent in communication is less than 13% of the total computational time in all cases (Table 3). Using 4 processors, only 5% of the total time is spent in communication. The communication effort increases by a factor of 2.5 when using 64 times more processors. This demonstrates the high efficiency of the mapping and communication routines in the PPM library.

4.5. Three-dimensional vortex methods

The final client application involves simulations using three-dimensional particle vortex methods. The application demonstrates a large number of the present library tools and their interplay as it involves particle convection and diffusion, particle–mesh/mesh–particle interpolation, particle reinitialization, and the solution

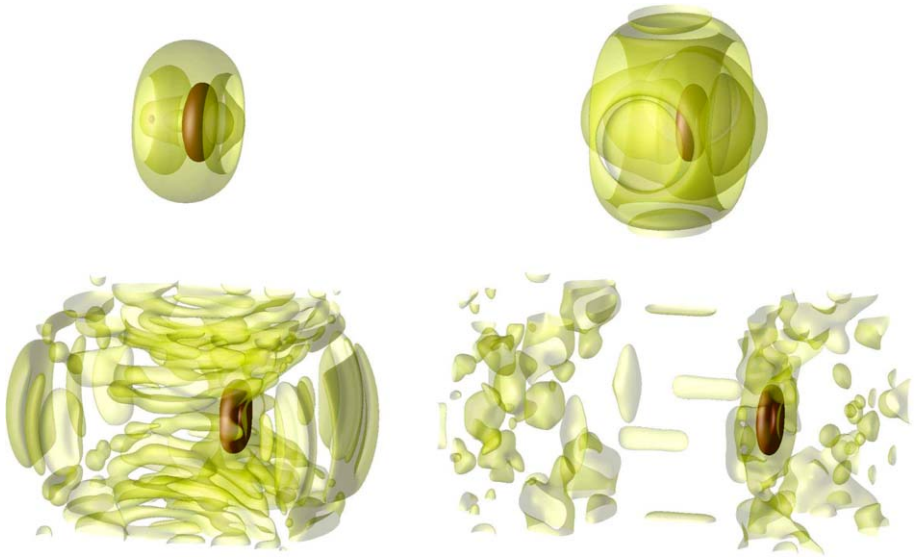

Fig. 11. Iso-surfaces of the density field for $\rho = 0.900$, 0.990, and 1.015 at times $t = 0.15$, 0.25, 0.50, and 1.00. The field is discretized using 33 million particles. Acoustic pressure waves propagating from the ring can be seen as lightly shaded surfaces. The darkest iso-surface of the density field indicates the position of the vortex ring.

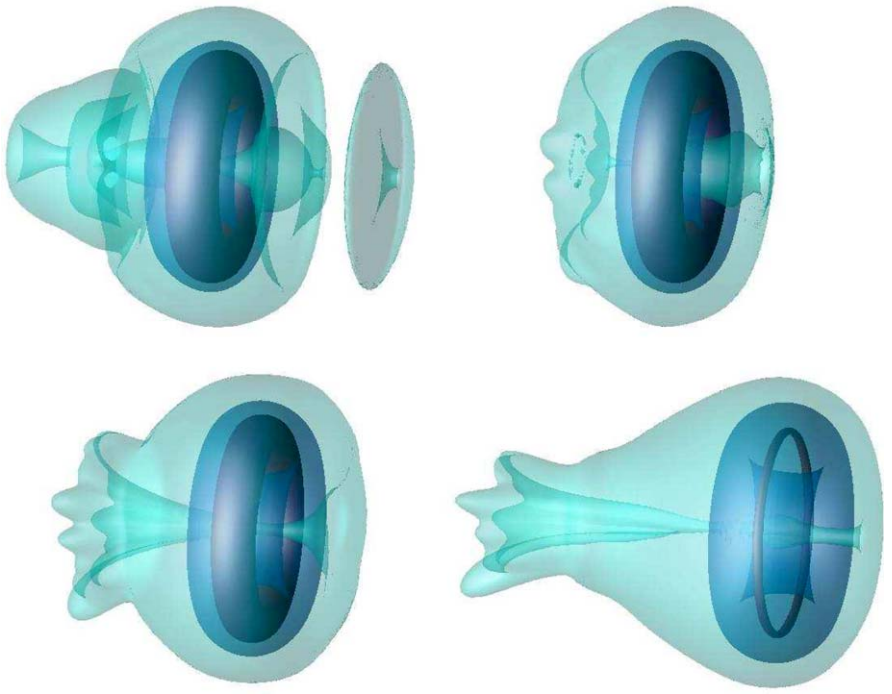

Fig. 12. Iso-surfaces of vorticity for $|\boldsymbol{\omega}| = 40$, 10, and 0.2 at t=0.15, 0.25, 0.50, and 1.00 for the same simulation as in Fig. 11.

of Poisson equations on the mesh. Hybrid vortex methods [39,24,8] solve the incompressible Navier–Stokes equations in the Lagrangian vorticity–velocity formulation

$$\frac{\mathrm{D}\boldsymbol{\omega}}{\mathrm{D}t} = (\boldsymbol{\omega}\cdot\nabla)\boldsymbol{u} + \nu\nabla^2\boldsymbol{\omega} \tag{25}$$

Table 3
Communication-to-computation ratio of the PPM using the rSPH client with 16 million particles

N_{proc}	Total time (s)	Communication (s)	In percent
4	195	10	5%
16	50	4	8%
64	14	1.2	11%
128	7	0.8	12%

and

$$\nabla^2 \boldsymbol{\Psi} = -\boldsymbol{\omega}, \tag{26}$$

where $\boldsymbol{u} = \nabla \times \boldsymbol{\Psi}$ is the velocity and ν is the viscosity of the fluid. These equations are discretized using particles that carry vorticity $\boldsymbol{\omega}$ and that are convected by the local flow velocity field $\boldsymbol{u}$. The vorticity of the particles is interpolated onto a mesh where it is used as right-hand side for the vector Poisson equation (26), which is solved for the stream function using one of the PPM Poisson solvers (cf. Section 3.6). Velocities are computed from the stream function using second order finite differences, and the vorticity diffusion and stretching are evaluated at mesh point locations also employing second order finite differences. The time step is completed by interpolating the grid functions $\frac{D\boldsymbol{\omega}}{Dt}$ and $\boldsymbol{u}$ back onto particle locations. Distortion of the particle locations leads to spurious vorticity structures and the flow ceases to be well represented by the particles. Therefore, particles are remeshed onto regular positions after each time step using the remeshing routines of the PPM library. The M'_4 function [23] is used for all interpolation steps.

4.5.1. Parallel speedup, timing and efficiency

To study the parallel performance of the vortex client, we consider the double shear layer [34] as used for the rSPH tests (Section 4.4). We use the whole computational domain as vorticity support, so that the number of particles is equal to the number of grid points. All simulations include the solution of a convection–diffusion equation for a passive scalar. The simulation starts from the initial condition proposed by Ghoniem and Knio [34] with $Re = 990$. The Reynolds number is computed as

$$Re = \frac{|\boldsymbol{u}|_\infty \sigma}{\nu},$$

where σ denotes the thickness of the shear layer. The ODEs are integrated using the PPM ODE solver with a second order midpoint Runge–Kutta method and the vector Poisson equation (26) is solved with the PPM MG solver. The results for the scaled and fixed-size cases are depicted in Figs. 13 and 14, respectively. The largest system comprises 268 million particles distributed onto 128 processors. For this system, one iteration takes 85 s on average with a parallel efficiency of 63%. Vectorization of the code is tested on the NEC SX-5

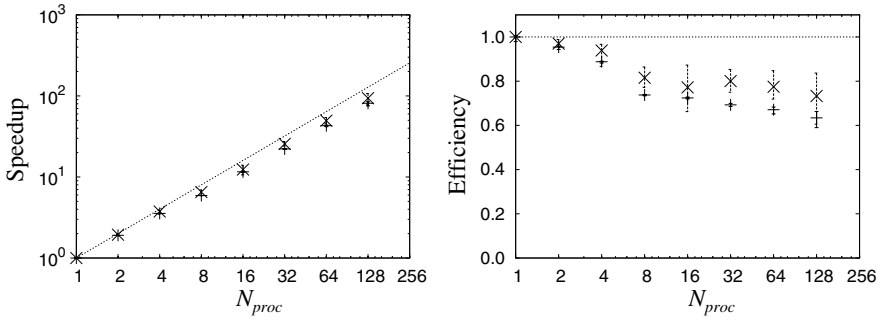

Fig. 13. Parallel speedup and efficiency of the PPM vortex client (+) and of the particle–mesh interpolation alone (×) for the scaled-size problem. The initial mesh resolution on one processor is $128 \times 128 \times 128$. Each point is averaged from 5 samples, error bars indicate min–max span. All timings are performed on the IBM p690.

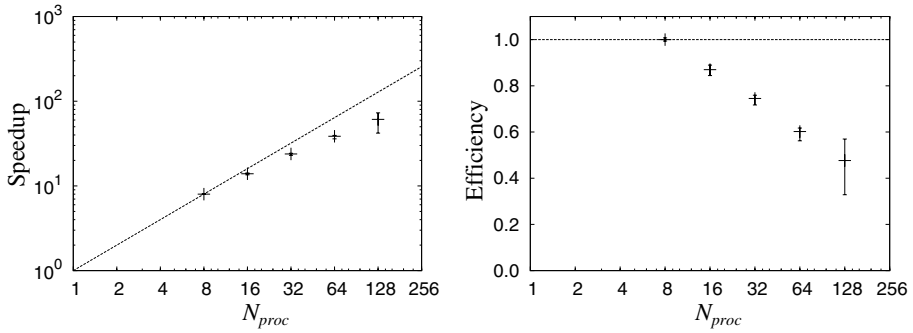

Fig. 14. Parallel speedup and efficiency of the PPM vortex client for the fixed-size problem with $256 \times 256 \times 256$ mesh points on 8–128 processors. Each point is averaged from 5 samples, error bars indicate min–max span. All timings are performed on the IBM p690.

computer using 8 processors. All major loops vectorize, including 99% of the particle-to-mesh and mesh-to-particle interpolations with an average vector length of 230 words. Interpolating 2 million particles onto a $128 \times 128 \times 128$ mesh takes 3.4 s on a single processor. Interpolating the field back onto the particles takes 1.1 s. The particle-to-mesh interpolation using color-encoded particles as described in Section 3.5 vectorizes 99% with an average vector length of 230 words.

We also perform a high-resolution simulation of the triply-periodic elliptic instability of two anti-parallel vortex tubes on 16 AMD Opteron processors. The vorticity initial condition follows Laporate and Leweke [40] with a circulation based Reynolds number of $Re = 2400$. For this test case, the ODEs are solved using a low-storage third order Runge–Kutta scheme [27] and the vector Poisson equation is solved with the FFT-based Poisson solver of the PPM library.

The results are computed on a $512 \times 256 \times 256$ mesh with a maximum of 33 million particles. One time step takes 73 s. The left panel of Fig. 15 shows the effective viscosity ν_{eff} [41] given by

$$\nu_{\text{eff}} = -\frac{\mathrm{d}k^h}{\mathrm{d}t}\left(\int_\Omega \boldsymbol{\omega}^h \cdot \boldsymbol{\omega}^h \,\mathrm{d}\boldsymbol{x}\right)^{-1}, \quad \text{as } \frac{\mathrm{d}k}{\mathrm{d}t} = -\nu \int_\Omega \boldsymbol{\omega} \cdot \boldsymbol{\omega} \,\mathrm{d}\boldsymbol{x}, \tag{27}$$

where k^h and $\boldsymbol{\omega}^h$ denote the kinetic energy and vorticity of the numerical solution. The error in the effective viscosity is below 2% throughout the simulation. The right panel of Fig. 15 illustrates the discretization of the vorticity by particles for an initial condition where the mode of the Crow-instability [40] is dominant.

At the onset of the elliptic instability, the non-dimensional wave number, given by $k^* = \frac{2\pi}{\lambda}a$, with λ the wavelength of the instability and a the core size of the tubes, is determined to be $k^* = 1.85$. This is in good agreement with the 1.87 found by Laporte and Leweke [40]. Fig. 16 displays the formation of vortex "bridges" connecting and eventually destroying the two tubes.

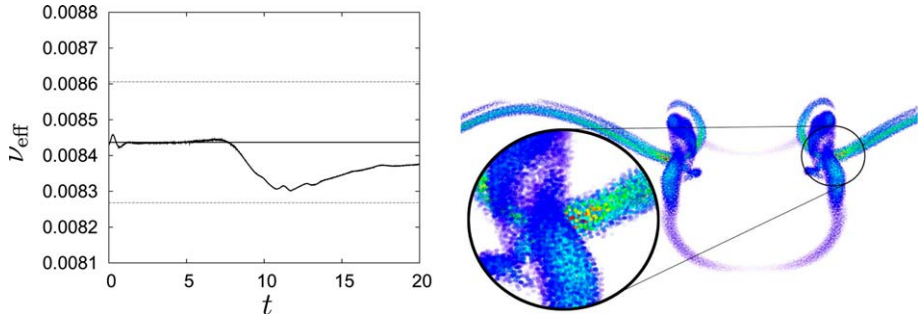

Fig. 15. Left panel: Effective viscosity as defined in Eq. (27) measured during the course of the simulation. Horizontal lines denote +2% error, target viscosity (0.008437), and −2% error, respectively. Right panel: Illustration of the discretization of the vorticity by particles. Particles are colored according to the local magnitude of vorticity. Only particles with $|\boldsymbol{\omega}| > 6$ are shown.

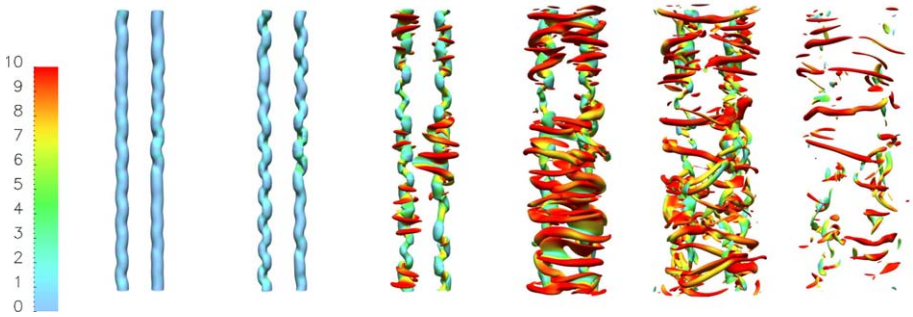

Fig. 16. Iso-surfaces of the vorticity ($|\omega| = 10$) for $t = 5.0$, 6.0, 9.0, 10.5, 12.0, and 14.2. The surfaces are colored according to the magnitude of the vorticity components perpendicular to the tube axes.

5. Summary

We have presented a new parallel particle–mesh library PPM which provides a general-purpose, physics-independent infrastructure for simulating systems using particle methods. The library integrates particle, mesh, and hybrid particle–mesh algorithms and its design goals include ease of use, flexibility, state-of-the-art parallel scaling, good vectorization, and platform independence. The library provides for the first time an enabling computational tool for large scale simulations using particle methods.

The library's ease of use was achieved by limiting the number of user-callable functions and using generic interfaces, overloaded for different variants of the same task. This feature of the library is demonstrated by the development of a number of client applications, with high parallel efficiency as presented in this paper.

Flexibility and independence from specific physics was demonstrated by having various simulation client applications. The library was successfully compiled and used on Intel/Linux, Apple G5/OS X, IBM p690/AIX, NEC SX-5/SUPER-UX, and AMD Opteron/Linux.

Parallel scaling and efficiency were assessed in the test cases presented in Section 4 of this paper. All applications showed parallel efficiencies reaching or exceeding the present state of the art, and favorable run-times on large systems. We have presented a state of the art PSE simulation using 1 billion particles, a VM simulation using 268 million particles – to our knowledge the largest VM done so far –, an SPH simulation exceeding the parallel efficiency of the current domain specific fastest code, simulations sustaining up to 33% of the machine peak performance, and a multigrid Poisson routine solving for half a billion unknowns in less than 7 s on 64 processors. Moreover, vectorization as tested on the NEC SX-5 computer demonstrated the suitability of the PPM library for vector architectures. Ongoing work involves extension of the library to multilevel and multiresolution methods, in the spirit of mesh-oriented projects such as CHOMBO [42] or SAMRAI [43] and applications of these techniques in multiscale simulations of flow–structure interactions.

The absence of suitable parallel libraries has prevented so far the widespread application of particle methods in certain domains as well as the cross fertilization among the particle methods developed in diverse disciplines for the simulation of specific complex systems. Presently, we are working on developing this library further into an open source project that would enable several state-of-the-art calculations across scientific disciplines.

Acknowledgments

The authors acknowledge discussions with D. Charypar and P. Gonnet (CSE Lab, ETHZ) as well as implementation support by I. Oppermann and B. Polasek (ETHZ). Computer resources were provided by the Swiss National Supercomputing Centre (CSCS).

References

[1] R.W. Hockney, J.W. Eastwood, Computer Simulation Using Particles, second ed., Institute of Physics Publishing, Bristol, PA, USA, 1988.

[2] P. Koumoutsakos, Multiscale flow simulations using particles, Annu. Rev. Fluid Mech. 37 (2005) 457–487.
[3] M. Adams, H.H. Bayraktar, T.M. Keaveny, P. Papadopoulos, Applications of algebraic multigrid to large-scale finite element analysis of whole bone micro-mechanics on the IBM SP, in: Proceedings of SC2003: High Performance Networking and Computing, ACM/IEEE, 2003.
[4] R.D. Falgout, J.E. Jones, U.M. Yang, The design and implementation of hypre, a library of parallel high performance preconditioners, Technical Report ucrl-jrnl-205459, Lawrence Livermore National Laboratory, 2004.
[5] P. Degond, S. Mas-Gallic, The weighted particle method for convection–diffusion equations. Part 1: The case of an isotropic viscosity, Math. Comput. 53 (188) (1989) 485–507.
[6] L. Greengard, V. Rokhlin, The rapid evaluation of potential fields in three dimensions, Lect. Notes Math. 1360 (1988) 121–141.
[7] F.H. Harlow, Particle-in-cell computing method for fluid dynamics, Meth. Comput. Phys. 3 (1964) 319–343.
[8] G.-H. Cottet, P. Poncet, Advances in direct numerical simulation of 3D wall-bounded flows by vortex-in-cell methods, J. Comput. Phys. 193 (2003) 136–158.
[9] Q. Wang, Variable order revised binary treecode, J. Comput. Phys. 200 (2004) 192–210.
[10] R. Couturier, C. Chipot, Parallel molecular dynamics using OpenMP on a shared memory machine, Comp. Phys. Commun. 124 (2000) 49–59.
[11] S.J. Plimpton, M.F. Seidel, D.B. Pasik, R.S. Coats, G.R. Montry, A load-balancing algorithm for a parallel electromagnetic particle-in-cell code, Comp. Phys. Commun. 152 (2003) 227–241.
[12] B. Moon, J. Saltz, Adaptive runtime support for direct simulation Monte Carlo methods on distributed memory architectures, in: Proceedings of the IEEE Scalable High-Performance Computing Conference, IEEE, 1994, pp. 176–183.
[13] S.R. Chapple, L.J. Clarke, The parallel utilities library, in: Proceedings of the IEEE Scalable Parallel Libraries Conference, IEEE, 1994, pp. 21–30.
[14] C. Nieter, J.R. Cary, VORPAL: a versatile plasma simulation code, J. Comput. Phys. 196 (2) (2004) 448–473.
[15] E.A. Carmona, L.J. Chandler, On parallel PIC versatility and the structure of parallel PIC approaches, Concurr.: Pract. Ex. 9 (12) (1997) 1377–1405.
[16] L. Verlet, Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules, Phys. Rev. 159 (1) (1967) 98–103.
[17] J.D. Eldredge, A. Leonard, T. Colonius, A general deterministic treatment of derivatives in particle methods, J. Comput. Phys. 180 (2) (2002) 686–709.
[18] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392.
[19] M. Bergdorf, G.-H. Cottet, P. Koumoutsakos, Multilevel adaptive particle methods for convection–diffusion equations, Multiscale Model. Simul. 4 (1) (2005) 328–357.
[20] V.G. Vizing, On an estimate of the chromatic class of a p-graph, Diskret. Anal. 3 (1964) 25–30 (in Russian).
[21] G.L. Djordjević, M.B. Tošić, A heuristic for scheduling task graphs with communication delays onto multiprocessors, Parallel Comput. 22 (1996) 1197–1214.
[22] D. Durand, R. Jain, D. Tseytlin, Parallel I/O scheduling using randomized, distributed edge coloring algorithms, J. Parallel Distrib. Comput. 63 (2003) 611–618.
[23] J.J. Monaghan, Extrapolating B splines for interpolation, J. Comput. Phys. 60 (2) (1985) 253–262.
[24] J.H. Walther, P. Koumoutsakos, Three-dimensional particle methods for particle laden flows with two-way coupling, J. Comput. Phys. 167 (2001) 39–71.
[25] U. Trottenberg, C. Oosterlee, A. Schueller, Multigrid, Academic Press, San Diego, 2001.
[26] V. Alexiades, G. Amiez, P.-A. Gremaud, Super-time-stepping acceleration of explicit schemes for parabolic problems, Commun. Numer. Meth. Eng. 12 (1) (1996) 31–42.
[27] J.H. Williamson, Low-storage Runge–Kutta schemes, J. Comput. Phys. 35 (1980) 48–56.
[28] C.-W. Shu, S. Osher, Efficient implementation of essentially nonoscillatory shock-capturing schemes, J. Comput. Phys. 77 (2) (1988) 439–471.
[29] M.F. Adams, H.H. Bayraktar, T.M. Keaveny, P. Papadopoulos, Ultrascalable implicit finite element analyses in solid mechanics with over half a billion degrees of freedom, in: Proceedings of SC2003: High Performance Networking and Computing, ACM/IEEE, 2004 (Gordon Bell Award Paper).
[30] J. Lippincott-Schwartz, E. Snapp, A. Kenworthy, Studying protein dynamics in living cells, Nat. Rev. Cell Biol. 2 (2001) 444–456.
[31] G.-H. Cottet, Private communication, 1999.
[32] M. Weiss, Personal communication, 2002.
[33] A.K. Chaniotis, D. Poulikakos, P. Koumoutsakos, Remeshed smoothed particle hydrodynamics for the simulation of viscous and heat conducting flows, J. Comput. Phys. 182 (1) (2002) 67–90.
[34] A. Ghoniem, O. Knio, The development and application of the transport element method to three dimensional reacting shear layers, in: Vortex Dynamics and Vortex Methods, American Mathematical Society, 1991, pp. 165–218.
[35] K. Shariff, R. Verzicco, P. Orlandi, A numerical study of three-dimensional vortex ring instabilities: viscous corrections and early nonlinear stage, J. Fluid Mech. 279 (1994) 351–375.
[36] V. Springel, N. Yoshida, S.D.M. White, GADGET: a code for collisionless and gasdynamical cosmological simulations, New Astron. 6 (2001) 79–117.
[37] P.G. Saffman, Vortex Dynamics, Cambridge University Press, Cambridge, 1992.

[38] D.W. Moore, The effect of compressibility on the speed of propagation of a vortex ring, Proc. R. Soc. Lond. A 397 (1812) (1985) 87–97.
[39] J.P. Christiansen, Numerical simulation of hydrodynamics by the method of point vortices, J. Comput. Phys. 13 (1973) 363–379.
[40] F. Laporte, T. Leweke, Elliptic instability of counter-rotating vortices: experiment and direct numerical simulation, AIAA J. 40 (12) (2002) 2483–2494.
[41] G.S. Winckelmans, A. Leonard, Contribution to vortex particle methods for the computation of three-dimensional incompressible unsteady flows, J. Comput. Phys. 109 (1993) 247–273.
[42] A.S. Almgren, J.B. Bell, P. Colella, L.H. Howell, M.L. Welcome, A conservative adaptive projection method for the variable density incompressible Navier–Stokes equations, J. Comput. Phys. 142 (1998) 1–46.
[43] A. Wissink, R. Hornung, S. Kohn, S. Smith, N. Elliott, Large scale parallel structured AMR calculations using the SAMRAI framework, in: Proc. SC01 Conf. High Perf. Network. & Comput. Denver, CO, 2001.