A massively space-time parallel N-body solver

R. Speck*, D. Ruprecht*¶, R. Krause*, M. Emmett†, M. Minion‡, M. Winkel§, P. Gibbon§
*Institute of Computational Science, Università della Svizzera italiana, Lugano, Switzerland.
Email: {robert.speck, daniel.ruprecht, rolf.krause}@usi.ch
†Lawrence Berkeley National Laboratory, Berkeley, USA.
Email: mwemmett@lbl.gov
‡Institute for Computational and Mathematical Engineering, Stanford University, Stanford, USA.
Email: mlminion@gmail.com
§Jülich Supercomputing Centre, Jülich, Germany. Email: {m.winkel, p.gibbon}@fz-juelich.de
¶Mathematisches Institut, Heinrich-Heine-Universität, 40225 Düsseldorf, Germany.

***Abstract*—We present a novel space-time parallel version of the Barnes-Hut tree code PEPC using PFASST, the *Parallel Full Approximation Scheme in Space and Time*. The naive use of increasingly more processors for a fixed-size N-body problem is prone to saturate as soon as the number of unknowns per core becomes too small. To overcome this intrinsic strong-scaling limit, we introduce temporal parallelism on top of PEPC's existing hybrid MPI/PThreads spatial decomposition. Here, we use PFASST which is based on a combination of the iterations of the parallel-in-time algorithm parareal with the sweeps of spectral deferred correction (SDC) schemes. By combining these sweeps with multiple space-time discretization levels, PFASST relaxes the theoretical bound on parallel efficiency in parareal. We present results from runs on up to 262,144 cores on the IBM Blue Gene/P installation JUGENE, demonstrating that the space-time parallel code provides speedup beyond the saturation of the purely space-parallel approach.**

I. Introduction

Time-dependent partial differential equations are ubiquitous in scientific and industrial applications and usually must be solved numerically. Because of the tremendous number of degrees-of-freedom in contemporary large-scale problems, using massively parallel computer systems is essential to absorb overwhelming memory requirements and to obtain results in reasonable times. This requires the employed solution algorithms to expose a sufficient degree of parallelism in order to allow for a decomposition of the discrete problem into sub-problems, which can be processed by individual cores.

The typical approach is to decompose the (spatial) computational domain into sub-domains and let each core process the degrees-of-freedom of one sub-domain. We refer to this approach as *spatial parallelization*. While this strategy works very well and can provide near perfect speedup to rather large numbers of cores, its strong scaling inevitably saturates as the number of degrees-of-freedom per core becomes too small and communication time begins to dominate. If, for a problem of fixed size, more cores are to be used to further speed up simulations, additional directions of parallelism must be considered. As the number of cores in state-of-the-art high-performance computing systems continues to increase, new inherently parallel algorithms for the solution of PDEs are required to fully exploit the possibilities of upcoming systems. This will be particularly important when systems capable of exascale computing, anticipated to feature processing unit counts of the order of 10^8 [1], begin to become available. One possible and promising strategy is to employ parallelization along the time direction in conjunction with spatial parallelization.

First approaches to parallelizing solution algorithms for differential equations in time go back as early as [2]. However, the topic gained considerable momentum with the introduction of the parareal parallel-in-time algorithm in [3], which can be used for large numbers of processors. Parareal relies on the iterative application of two integration schemes: an accurate and computationally expensive ("fine") scheme running in parallel on multiple time slices and a less accurate and cheaper ("coarse") scheme used to serially propagate corrections through time. The method iterates over both schemes and converges to a solution of the same accuracy as would be provided by the fine scheme run in serial. Over the last decade, a growing body of literature has emerged, dealing with different mathematical and algorithmic aspects of parareal, see for example [4], [5]. The algorithm's performance has been investigated for very different applications, for example quantum control [6], simulation of oceanic flow [7], or simulation of turbulent plasma [8]. An event-based implementation of parareal is investigated in [9]. Studies of the related *parallel implicit time integration* (PITA) algorithm for fluid-structure interactions and structural dynamics can be found in [10], [11], [12]. Most authors focus on the algorithmic aspects of parareal while studies on the performance of time-parallel schemes in combination with spatial domain decomposition for large core counts are scarce. An evaluation of parareal including spatial parallelization for the three-dimensional Navier-Stokes equations on up to 2,048 cores was conducted in [13].

However, all prior investigations for PDEs use grid-based methods, and the coupling of parallel-in-time algorithms and particle-based spatial solvers has not been considered so far. Like in space-parallel grid-based methods, the efficiency of classical parallelization strategies for particle methods – for direct summation as well as for sophisticated multipole-based summation techniques such as the Barnes-Hut tree code [14] or the Fast Multipole Methods [15] – saturates as the number of unknowns per core becomes too small. Beyond this point,

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

using additional cores for spatial decomposition may even increase runtime due to data management and communication overhead.

A pivotal problem of parareal is its rather strict inherent limit on parallel efficiency, which is bounded by the inverse of the number of performed iterations, see e. g. [16]. Very recently, the *Parallel Full Approximation Scheme in Space and Time* (PFASST), a time-parallel scheme based on intertwining the iterations of parareal with the sweeps of spectral deferred correction (SDC, see [17]) integration schemes was introduced in [18], [19]. By using a fine integrator of iteration-dependent, increasing accuracy, PFASST overcomes the strict limit on parallel efficiency of parareal and features a much less severe efficiency bound.

PFASST and parareal both profit from decreasing the runtime of the coarse scheme by not only coarsening in time but also using a coarser spatial discretization, either by using fewer degrees-of-freedom in the spatial discretization of the coarse scheme [20] and/or by using lower order stencils [21]. PFASST essentially improves this strategy by using a full approximation scheme (FAS) technique to recycle coarse grid information efficiently and to increase the accuracy of the coarse grid SDC sweeps. While coarsening in time can be easily obtained in PFASST by using less intermediate collocation points, coarsening in space has to date only been studied using a hierarchical multigrid approach. This mechanism can naturally be applied in the case of grid-based methods, but it is not clear how such a strategy can be employed for purely particle-based methods.

In the present paper, we demonstrate the coupling of the parallel-in-time scheme PFASST with the parallel Barnes-Hut tree code PEPC, the *Pretty Efficient Parallel Coulomb Solver*. We introduce a concept of spatial coarsening for the Barnes-Hut approach that allows time-and-space coarsening for grid-free algorithms and thereby improves the scaling of PFASST significantly in this case. The performance of the resulting space-time-parallel code is explored on the IBM Blue Gene/P machine JUGENE with runs using up to 262,144 cores. It is demonstrated that time-parallelism can provide considerable additional speedup beyond the saturation of the spatial parallelization. Besides proving the success of combining PFASST with a particle scheme, this also constitutes by far the largest test run reported to date using a combination of a time-parallel integration scheme with spatial parallelization. The results give a conclusive illustration of the potential of time-parallel methods, even for extreme-scale applications. Summarized, the major contributions of this work are:

- first large-scale combination of parallel-in-time integration methods with particle discretization techniques,
- effective particle-based coarsening by means of multipole approximation,
- unique combination of the hybrid MPI/Pthreads Barnes-Hut tree code PEPC with an MPI-based time-decomposition method,
- significant speedup of the full space-time parallel algorithm using PFASST beyond the saturation of the purely space-parallel version,
- largest space-time parallel simulations performed up to now (to the best of our knowledge).

Section II describes the model problem, which is based on the 3D vortex particle method. Here, the discretization leads to (a) an N-body problem that can be handled using multipole-based fast summation algorithms, and (b) evolution equations for particle position and vorticity that require integration in time. A concise overview of the combination of PFASST and PEPC with respect to the implementation is given in Section III, followed by a more detailed description of PEPC and its space-parallel capabilities in Section III-A and of SDC and PFASST in Section III-B. The accuracy of SDC and PFASST for a direct N-body solver and a small problem size is analyzed in Section IV-A, in order to find a setup where PFASST and serial SDC provide comparable accuracy. This should make sure that the subsequent speedup studies are fair and compare runtimes of solutions of similar quality. Speedups for large-scale problems for the combination of PFASST with PEPC are reported in Section IV-B. Finally, results are summarized in Section V.

II. The vortex particle method

To analyze the combination of particle-based algorithms and parallelized time integration, we focus on the vortex particle method for incompressible, inviscid Newtonian fluid flows in three spatial dimensions, see [22]. Based on the *vorticity-velocity formulation* of the Navier-Stokes equations

$$\frac{\partial\omega}{\partial t} + (u\cdot\nabla)\omega = (\omega\cdot\nabla)u \tag{1}$$

for a solenoidal velocity field u and the corresponding vorticity field $\omega = \nabla\times u$, the velocity field can be determined by the Green's function $G(x) = \frac{1}{4\pi|x|}$ for three-dimensional unbounded domains (omitting boundary and irrotational effects) with

$$u(x,t) = \int K(x-y)\times\omega(y,t)\,\mathrm{d}y, \quad K = \nabla G. \tag{2}$$

This explicit formula exemplifies one big advantage of vorticity-based formulations of the Navier-Stokes equations in contrast to many other approaches in CFD.

The integral and therefore the vorticity field ω in (2) is discretized using N regularized vortex particles x_p, so that

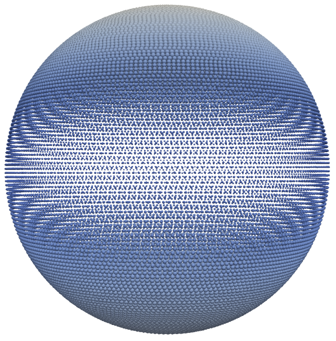
$$u(x,t) \approx \tilde{u}_\sigma(x,t) = \sum_{p=1}^{N} K_\sigma(x-x_p)\times\omega(x_p,t)\mathrm{vol}_p, \tag{3}$$

$$K_\sigma = \nabla G * \zeta_\sigma. \tag{4}$$

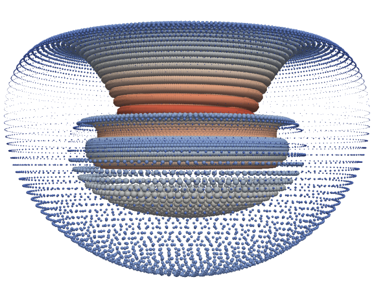
Here, each quadrature point x_p is connected to a volume vol_p in the 3D simulation domain. Its vorticity vector $\omega(x_p)$ is 'smeared' onto the support of a radial symmetric smoothing function ζ_σ with small core size σ. The evolution equations for the particles are then given by

$$\frac{\mathrm{d}x_q(t)}{\mathrm{d}t} = \tilde{u}_\sigma(x_q(t),t), \tag{5}$$

$$\frac{\mathrm{d}\omega(x_q(t),t)}{\mathrm{d}t} = \left(\omega(x_q(t),t)\cdot\nabla^T\right)\tilde{u}_\sigma(x_p(t),t). \tag{6}$$



(a) time 1



(b) time 25

Fig. 1: Time evolution of the spherical vortex sheet. Downwards movement is filtered out in the pictures, size and color of the particles correspond to the magnitude of the velocity: small/blue = low velocity, large/red = high velocity. $N = 20{,}000$ particles, second-order Runge-Kutta with $\Delta t = 1$. Sixth-order algebraic kernel with $\sigma \approx 18.53h$.

As our 3D model problem, we choose the time evolution of a spherical vortex sheet. A given number of N particles is initially placed on a sphere with radius $R = 1$, centered at the origin, and attached with vorticity ω, volume h and core radius σ using

$$\omega(x) = \omega(\rho, \theta, \varphi) = \frac{3}{8\pi} \sin(\theta) \cdot e_\varphi, \tag{7}$$

$$h = \sqrt{\frac{4\pi}{N}}, \quad \sigma \approx 18.53h, \tag{8}$$

where e_φ is the φ-unit-vector in spherical coordinates. For our simulations, we use a sixth-order algebraic kernel, see [23] for details. Fig. 1 visualizes the evolution of this setup for $N = 20{,}000$ particles and a classical, second-order Runge-Kutta time-stepping scheme with $\Delta t = 1$. As noted in [24], the initial conditions are the solution to the problem of flow past a sphere with unit free-stream velocity along the z-axis. While moving downwards in the z-direction, the sphere collapses from the top and wraps into its own interior, forming a large, traveling vortex ring in the inside, see [23], [24], [25], [26] for comparison.

The smoothing function ζ_σ is commonly chosen with non-compact support, so it can be interpreted as a long-range potential. As we can see from Eq. (3), this formulation yields an N-body problem: to evaluate the right-hand side of the evolution equations (5) and (6) for a particle x_q, $q = 1, \dots, N$, we need to compute the N summands of $\tilde{u}_\sigma(x_q(t), t)$ and $(\omega(x_q(t), t) \cdot \nabla^T)\, \tilde{u}_\sigma(x_p(t), t)$, thus leading to a total of $\mathcal{O}(N^2)$ operations for all particles. For generalized algebraic smoothing kernels, multipole expansion techniques such as the Barnes-Hut tree code can be applied to speed up these evaluations, see [23] for details. In Section III-A, we describe one implementation of this approach. Moreover, Eqs. (5) and (6) are the basis for the time integration scheme. Classically, time-serial third- or fourth-order Runge-Kutta schemes are used to update particle positions and vorticity strengths, see e. g. [27], and spatial parallelization is used to speed up evaluations of the right hand side. Here, a new direction of parallelization is added by using PFASST for integration in time, which we describe in detail in Section III-B.

III. Spatial and temporal parallelization: PEPC and PFASST

The PFASST algorithm [19] is a parallel-in-time solver for initial value problems

$$\frac{\partial u}{\partial t} = f(t, u), \quad u(0) = u_0. \tag{9}$$

It is based on intertwining the iterations of parareal [3] with the iterations of Spectral Deferred Correction (SDC) integration schemes [17]. As in parareal, it relies on a coarse propagator $\mathcal{G}$ to generate approximations of the solution at later points in time. These are then corrected by a fine propagator $\mathcal{F}$ which is run concurrently on multiple time-slices. A Full Approximation Scheme (FAS) approach allows for an effective transfer of information from the fine to the coarse scheme so that $\mathcal{G}$ can be made faster by using a coarsened spatial discretization as well. The close coupling of SDC with parareal iterations leads to a significant improvement of the parallel performance compared to the original parareal approach [19]. For evaluating the right-hand side f in (9) in PFASST – corresponding to the right-hand sides of Eqs. (5) and (6) in the setup considered here – we choose a space-parallel approach based on the Barnes-Hut tree code PEPC [28]. By means of the multipole acceptance criterion explained below, PEPC computes approximations of f with different accuracy for the $\mathcal{G}$ and $\mathcal{F}$ SDC sweep. PEPC also provides the decomposition of the particle systems using MPI communicators in the spatial direction. PFASST, on the other hand, uses separate instance of PEPC (each with a given number P_S of compute nodes for the spatial parallelism) for each time slice (up to P_T instances simultaneously) and connects them using MPI communicators in the temporal direction, see Fig. 2. Hence, a compute node is always a member of two communicators: it is responsible for a subset of the particles within PEPC, at a specific time slice within PFASST.

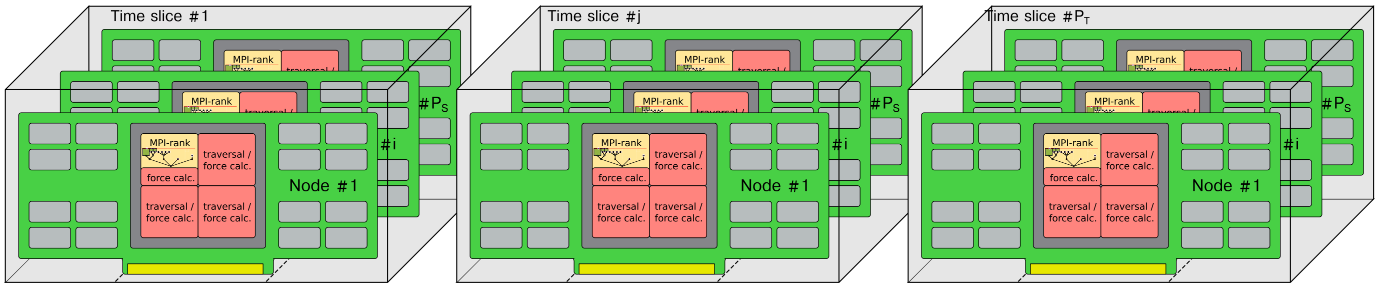


Fig. 2: The combination of PEPC with PFASST uses $P_T \times P_S$ nodes. Spatial decomposition for each of the P_T time slices is performed by PEPC using P_S nodes within each PEPC-communicator (depicted as one box in the figure). Within PEPC, one MPI-rank per node is used to act as data and communication management thread, while the other cores perform the traversal of the tree using Pthreads, see Section III-A. For PFASST, this structure is duplicated P_T times to create independently running instances of PEPC. PFASST connects the ith node of each box to one new MPI communicator, which results in P_S separated PFASST-communicators for the temporal decomposition.

In the following two sections, we describe the PEPC and PFASST codes in more detail, focusing on the parallelization strategies in space and time.

A. *The Barnes-Hut tree code* PEPC

In order to avoid the infeasible quadratic complexity for evaluating f for particle-based, long-range dominated systems, the idea of multipole-based summation techniques is to replace interactions with distant particles by interactions with particle clusters via multipole expansions that approximate the contribution of the corresponding distant particles. By hierarchically integrating particles into these clusters using a tree data structure, the number of interactions for all N particles and therefore the overall complexity of this approach can be estimated as $\mathcal{O}(N \log N)$ in the case of the Barnes-Hut tree code [14].

In the Barnes-Hut algorithm, all particles are inserted into an oct-tree data structure to efficiently group nearby particles into clusters and to identify interaction partners. First, space is recursively subdivided: the simulation domain is halved in each spatial direction on each level of recursion until every particle resides inside its own sub-box. All boxes that contain particles – including the coarser boxes of previous subdivisions – are inserted into the tree. The different levels of subdivision correspond to different tree levels: While the root node represents the single box covering the whole simulation domain, the finest boxes containing the individual particles are leaf-nodes in the tree. Fig. 3 shows a symbolic sketch of the tree construction process for a two-dimensional example that results in a quad-tree structure. Obviously, nearby particles are also in close relationship to each other in the tree since they share their ancestor nodes. This information is used for clustering the particles: Every intermediate-level box corresponds to a group of particles (or even clusters) representing them through their multipole moments. These moments can be efficiently calculated from the leaf nodes towards the root node that finally covers all particles.

For each particle, the tree is traversed to identify interaction partners (clusters or particles) using a multipole acceptance criterion (*MAC*): In the classical Barnes-Hut approach [14], [29] as depicted in Fig. 4, interaction with a cluster is allowed if the ratio between the size s of the corresponding box and its distance d from the current particle is smaller than a parameter $\vartheta \geq 0$, that is $\frac{s}{d} \leq \vartheta$. Otherwise, the respective node has to be resolved into its children. We refer to [30] for an overview and in-depth discussion on multipole acceptance criteria for Barnes-Hut tree codes. Large values of ϑ lead to interactions with larger clusters, which is a faster, more inexact representation of the right-hand sides of Eqs. (5) and (6), while smaller values of ϑ improve yet slow down the summation. In combination with PFASST, we exploit this mechanism to provide a concept of spatial coarsening to accelerate the coarse propagator $\mathcal{G}$ by using a significantly larger value of ϑ for the coarse than for the fine scheme.

Parallelization of the original Barnes-Hut algorithm – thus, parallelization in the spatial direction – is commonly performed via the *Hashed-Oct-Tree* scheme by Warren and Salmon [31], [32]. Here, data parallelism is obtained by distributing distinct tree sections onto the MPI-ranks that reside on the distributed memory nodes of a compute cluster. To this end, before starting the tree construction, each particle is assigned a unique key that represents its position on a space-filling curve. This curve is then partitioned and redistributed across the MPI-ranks. Afterwards, all MPI-ranks construct their local trees. Their respective root nodes are inserted into a common global tree that is shared among the ranks and forms the starting point for the tree traversals. In Fig. 3, the distribution of a particle set among 3 MPI-ranks is shown as an example. The local root nodes (or *branch nodes*) are shown as colored boxes. These nodes must be exchanged between all MPI-ranks for building the globally shared tree above them. As shown in Fig. 5, this significantly contributes to the overall runtime of the parallel Barnes-Hut tree code.

In [28], we have presented a novel hybrid MPI/Pthreads implementation of the *Pretty Efficient Parallel Coulomb Solver* PEPC, which is developed at Jülich Supercomputing Centre. The code has undergone a transition from a pure gravitation/Coulomb solver to a multi-purpose N-body suite that

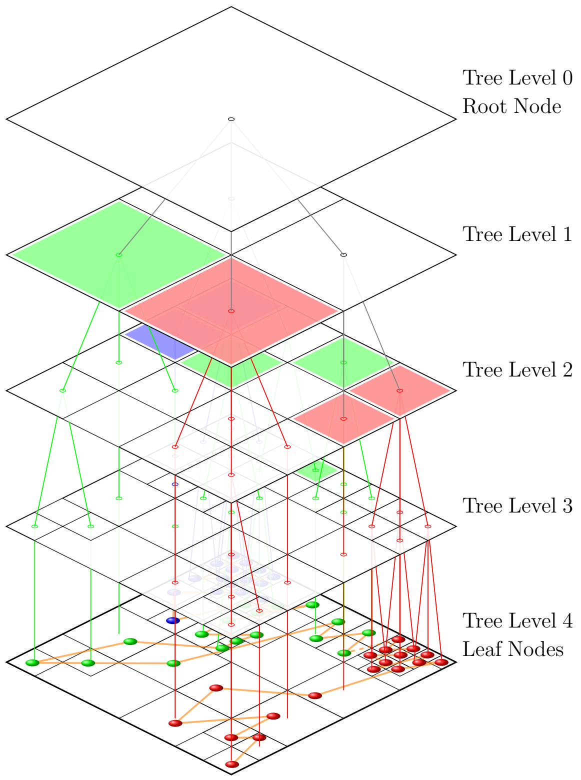


Fig. 3: Tree construction for a two-dimensional example (quad-tree, shown bottom-up here). The root node covers the whole simulation domain; leaf nodes represent individual particle boxes. The orange line shows the space-filling curve used to repartition the set of particles across the MPI-ranks. The particles' color indicate their rank assignment; box colors represent the MPI-ranks' local root nodes, colored lines the local trees. The gray tree is shared globally.

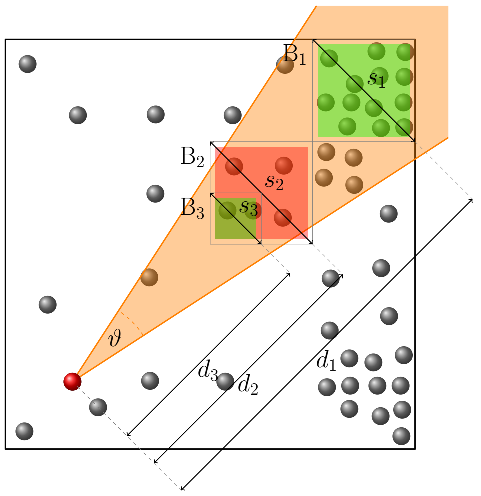


Fig. 4: The classical Barnes-Hut MAC can be interpreted as a cone with opening angle ϑ, exemplarily shown for a selected particle in the two-dimensional setup from Fig. 3. Boxes being larger than the cone diameter at their position, i. e. $\frac{s_i}{d_i} > \vartheta$, have to be further resolved into their child boxes. Consequently, the green boxes B_1 and B_3 are accepted for interaction while the red box B_2 has to be resolved further.

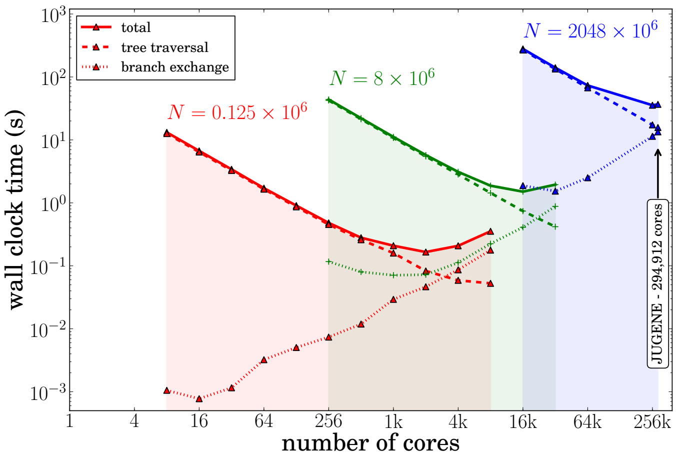


Fig. 5: Scaling of the parallel Barnes-Hut tree code PEPC across all 300k cores of the Blue Gene/P system JUGENE for a homogeneous neutral Coulomb system and different particle numbers N, from [28].

is utilized for numerous applications due to a convenient exchangeability of interaction kernels. Today, it is applied to modelling laser-plasma and plasma-wall interaction, stellar disc dynamics using Smooth Particle Hydrodynamics (SPH), and vortex particle methods for fluid simulations as depicted in Section II. We refer to [28], [33] and references therein for further details. One key aspect of this code is, that, by using node-local Pthreads parallelization for the force computation, an additional layer of concurrency is achieved. Communication and traversal tasks are distributed among several sub-threads on the different cores of shared-memory nodes. Consequently, force evaluation and communication, i. e. exchange of tree nodes (particle cluster properties) with remote MPI-ranks, now run completely asynchronously and overlap natively. This approach leads to a highly efficient use of resources on HPC systems with many-core compute nodes.

As shown in Fig. 5, PEPC exhibits excellent scalability with up to 2 billion particles across all 300k cores of the IBM Blue Gene/P installation JUGENE at Jülich Supercomputing Centre, provided the number of particles per compute core is sufficiently large. Naturally, when approaching small numbers of particles per core, communication, in particular the branch node exchange, dominates the actual computational work. This is critical for all algorithms designed to efficiently reduce the computational work: Every spatial domain decomposition approach, not only particle-based solvers for long-range interacting N-body systems, is naturally restricted in its strong-scaling capabilities and additional concepts are required to further reduce time-to-solution.

B. *SDC and* PFASST

The PFASST algorithm is a novel synthesis of parareal, SDC, and FAS. SDC methods are variants of traditional deferred or defect correction methods for ODEs, and were originally introduced in [17]. High-order accurate solutions within one time step are constructed by iteratively approximating a series of correction equations at intermediate nodes using low-order methods. The FAS corrections provide an elegant way of interpolating the results of coarse SDC sweeps in both space and time as well as initializing the coarse problem itself using the results of SDC on the fine level [19]. Thus, PFASST's coarse and fine propagators are connected in the same manner as FAS for non-linear multigrid methods.

1) Spectral Deferred Correction Methods: To describe SDC, we consider the initial value problem (9). It is convenient to work with the equivalent Picard integral form

$$u(t) = u_0 + \int_0^t f\big(\tau, u(\tau)\big)\,\mathrm{d}\tau. \tag{10}$$

As with traditional deferred correction methods, a single time step $[t_n, t_{n+1}]$ is divided into a set of intermediate sub-steps by defining $M+1$ intermediate points $t_m \in [t_n, t_{n+1}]$ such that $t_n = t_0 < t_1 < \cdots < t_M = t_{n+1}$. Then, the integrals of $f\big(\tau, u(\tau)\big)$ over the intervals $[t_n, t_m]$ are approximated by

$$\int_{t_n}^{t_m} f\big(\tau, u(\tau)\big)\,\mathrm{d}\tau = \Delta t \sum_{j=0}^{M} q_{m,j} f(t_j, U_j), \tag{11}$$

where $U_j \approx u(t_j)$, $\Delta t = t_{n+1} - t_n$, and $q_{m,j}$ are quadrature weights. The quadrature weights $q_{m,j}$ that give the highest order of accuracy for given intermediate points t_m are obtained by computing exact integrals of Lagrange interpolating polynomials.

To simplify notation, we define the *integration matrix* $\boldsymbol{Q}$ to be the $M \times (M+1)$ matrix consisting of entries $q_{m,j}$. We also define the vectors $\boldsymbol{U} = [U_1, \cdots, U_M]$ and $\boldsymbol{F}(\boldsymbol{U}) = [f(t_1, U_1), \cdots, f(t_M, U_M)]$. Finally, it is convenient to decompose $\boldsymbol{Q}$ as $\boldsymbol{Q} = [\boldsymbol{q}|\boldsymbol{S}]$, where $\boldsymbol{q}$ is the first column of $\boldsymbol{Q}$ and $\boldsymbol{S}$ is a square $M \times M$ matrix.

With these definitions, the SDC method employed here is an iterative method for solving

$$\boldsymbol{U} - \Delta t \boldsymbol{S}\boldsymbol{F}(\boldsymbol{U}) = U_0 + \Delta t \boldsymbol{q} f(t_0, U_0) \tag{12}$$

for $\boldsymbol{U}$. The method begins by computing a provisional solution $\boldsymbol{U}_0 = [U_1^0, \cdots, U_M^0]$ at each of the intermediate nodes. Subsequent iterations (denoted by k superscripts) proceed by applying the quadrature matrix $\boldsymbol{S}$ to $\boldsymbol{F}^k$ and correcting the result using a low-order time-stepper. For fully explicit equations (such as those in PEPC), a first-order time-stepping method similar to forward Euler for computing $\boldsymbol{U}^{k+1}$ is given by

$$U_{m+1}^{k+1} = U_m^{k+1} + \Delta t_m \big[f(t_m, U_m^{k+1}) - f(t_m, U_m^k)\big] + \boldsymbol{S}_m \boldsymbol{F}^k, \tag{13}$$

where $\Delta t_m = t_{m+1} - t_m$ and $\boldsymbol{S}_m \boldsymbol{F}^k$ approximates $\int_{t_m}^{t_m+1} f(\tau, \boldsymbol{U}^k)\,\mathrm{d}\tau$. Implicit-explicit (IMEX) schemes can be built in a similar fashion using forward/backward Euler [17].

The process of solving (13) at each node t_m is referred to as an *SDC sweep*. The accuracy of the solution generated after k SDC sweeps done with a first-order method is formally $\mathcal{O}(\Delta t^k)$ as long as the spectral integration rule is at least of order k.

To construct a parallel time-stepping scheme from SDC we note that: (a) the SDC and parareal iterations can be combined into a single hybrid iteration, and (b) the intermediate nodes of SDC schemes can be coarsened and refined to form a multigrid hierarchy in time with each level corresponding to a different time propagator. The use of an iterative time-stepper is of fundamental importance to the efficiency of PFASST: the costs of performing multiple SDC sweeps is amortized over the parareal iterations. Furthermore, the intermediate nodes of the SDC scheme lead to a multigrid structure in time within which FAS corrections can be computed directly. The black-box nature of the time-stepping schemes $\mathcal{G}$ and $\mathcal{F}$ in parareal does not allow this to be done so easily.

2) Full Approximation Scheme: The FAS correction τ_F^C for SDC iterations from the fine level F to the coarse level C is determined by considering SDC as an iterative method for solving (12). For a non-linear equation of the form $A(\boldsymbol{x}) = \boldsymbol{b}$ the corresponding residual equation is

$$A(\tilde{\boldsymbol{x}} + \boldsymbol{e}) = A(\tilde{\boldsymbol{x}}) + \boldsymbol{r}, \tag{14}$$

where $\boldsymbol{e}$ is the error and $\boldsymbol{r} = \boldsymbol{b} - A(\tilde{\boldsymbol{x}})$ is the residual. In the multigrid approach, the residual equation is re-written by replacing the coarse residual $\boldsymbol{r}^C$ by the restriction $T_F^C \boldsymbol{r}^F$ of the fine residual $\boldsymbol{r}^F$, where the restriction operator is denoted by T_F^C. With $\boldsymbol{y}^C = \tilde{\boldsymbol{x}}^C + \boldsymbol{e}^C$, the coarse FAS residual equation becomes $A^C(\boldsymbol{y}^C) = \boldsymbol{b}^C + \boldsymbol{\tau}_F^C$, where the FAS correction term is given by

$$\boldsymbol{\tau}_F^C = A^C(\tilde{\boldsymbol{x}}^C) - T_F^C A^F(\tilde{\boldsymbol{x}}^F). \tag{15}$$

Returning to SDC methods, combining (12) with (15), the FAS correction becomes

$$\boldsymbol{\tau}_F^C = \Delta t\big(T_F^C \boldsymbol{S}^F \boldsymbol{F}^F - \boldsymbol{S}^C \boldsymbol{F}^C\big). \tag{16}$$

This allows the coarse SDC iterations to achieve the accuracy of the fine SDC iterations at the resolution of the coarse grid, and ultimately allows the PFASST algorithm to achieve similar accuracy as a serial computation performed on the fine level [19].

Note that for a three level scheme the fine and coarse grid equations are

$$\boldsymbol{U}^0 - \Delta t \boldsymbol{S}^0 \boldsymbol{F}^0(\boldsymbol{U}^0) = \boldsymbol{B}^0 \tag{17a}$$
$$\boldsymbol{U}^1 - \Delta t \boldsymbol{S}^1 \boldsymbol{F}^1(\boldsymbol{U}^1) = \boldsymbol{B}^1 = T_0^1 \boldsymbol{B}^0 + \boldsymbol{\tau}_0^1 \tag{17b}$$
$$\boldsymbol{U}^2 - \Delta t \boldsymbol{S}^2 \boldsymbol{F}^2(\boldsymbol{U}^2) = \boldsymbol{B}^2 = T_1^2 \boldsymbol{B}^1 + \boldsymbol{\tau}_1^2 \tag{17c}$$

where the restriction operator T_F^C applied to integral terms corresponds to summing the fine values between coarse nodes (that is, integration in time) and subsequently restricting in space, and

$$\boldsymbol{B}^0 = \boldsymbol{U}_0 + \Delta t \boldsymbol{q} f(t_0, \boldsymbol{U}_0). \tag{18}$$

3) PFASST *algorithm:* For a PFASST run with L levels (with level 0 being the finest), the time interval of interest $[0, T]$ is divided into N uniform intervals $[t_n, t_{n+1}]$ which are assigned to processors[1] $\boldsymbol{P}_n$, $n = 0 \ldots N-1$. Each interval

[1]Note that in the case of combined space and time parallelization, $\boldsymbol{P}_n$ is not a single processor but one communicator collecting all processors handling the distributed solution at the corresponding point in time, see Fig. 2.

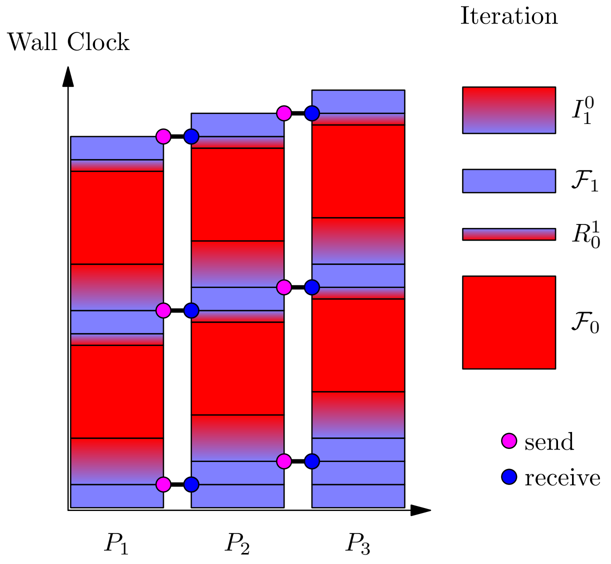


Fig. 6: Graphical depiction of the PFASST algorithm for the initialization stage and two PFASST iterations of a two level run with three processors (horizontal axis). Solid blocks denote SDC sweeps on the fine ($\mathcal{F}_0$) and coarse ($\mathcal{F}_1$) levels. Gradient blocks represent restriction (R_0^1) and interpolation (I_1^0) between levels. Solid lines between processors denote communication.

is subdivided on each level ℓ by defining $M_\ell + 1$ SDC nodes $\boldsymbol{t}_\ell = [t_{\ell,0} \cdots t_{\ell,M_\ell}]$ such that $t_n = t_{\ell,0} < \cdots < t_{\ell,M_\ell} = t_{n+1}$. The SDC nodes $\boldsymbol{t}_{\ell+1}$ on level $\ell + 1$ are chosen to be a subset of the SDC nodes $\boldsymbol{t}_\ell$ on level ℓ to facilitate interpolation and restriction between coarse and fine levels. The solution at the mth node on level ℓ during iteration k is denoted $U(\ell, k, m)$. For brevity, let

$$\boldsymbol{U}(\ell, k) = [U(\ell, k, 0), \cdots, U(\ell, k, M_\ell)] \tag{19}$$

and

$$\begin{aligned}\boldsymbol{F}(\ell, k) &= [F(\ell, k, 0), \cdots, F(\ell, k, M_\ell)] \\ &= [f(t_{\ell,0}, U(\ell, k, 0)), \cdots, f(t_{\ell,M_\ell}, U(\ell, k, M_\ell))].\end{aligned} \tag{20}$$

Finally, the FAS corrections on level ℓ for iteration k are denoted by $\boldsymbol{\tau}(\ell, k)$.

To begin we employ an initialization scheme wherein each processor $\boldsymbol{P}_n$ performs n coarse SDC sweeps (lowest $\mathcal{F}_1$ sweeps in Fig. 6, where $\mathcal{F}_1$ replaces the former coarse propagator $\mathcal{G}$ of parareal). This procedure is similar to the classical parareal initialization stage and propagates starting values for the subsequent parallel iterations. It has the same total computational cost of doing one SDC sweep per processor in serial, but the additional SDC sweeps compared to parareal can improve the accuracy of the solution significantly, as was demonstrated in [19].

After the initialization stage, the provisional solution is interpolated to the fine levels and the PFASST iterations on each processor are started immediately. In Algorithm 1, we describe

```
*Go down the V-cycle*
**for** $\ell = 0 \ldots L-2$ **do**
    *Sweep and send*
    $\boldsymbol{U}(\ell, k+1)$, $\boldsymbol{F}(\ell, k+1)$ =
    SDCSweep$(\boldsymbol{U}(\ell, k), \boldsymbol{F}(\ell, k), \boldsymbol{\tau}(\ell, k))$;
    **if** $n < N-1$ **then**
        Send $U(\ell, k+1, M_\ell)$ to $\boldsymbol{P}_{n+1}$;
    *Restrict and compute FAS correction*
    $\boldsymbol{U}(\ell+1, k+1)$ = Restrict$(\boldsymbol{U}(\ell, k+1))$;
    $\boldsymbol{F}(\ell+1, k+1)$ = FEval$(\boldsymbol{U}(\ell+1, k+1))$;
    $\boldsymbol{\tau}(\ell+1, k+1)$ =
    FAS$(\boldsymbol{F}(\ell, k), \boldsymbol{F}(\ell+1, k+1), \boldsymbol{\tau}(\ell, k+1))$;
*Coarsest level: get new initial value, sweep, and send*
**if** $n > 0$ **then**
    Receive $U(L-1, k, 0) = U(L-1, k, M_{L-1})$ from
    $\boldsymbol{P}_{n-1}$;
$\boldsymbol{U}(L-1, k+1)$, $\boldsymbol{F}(L-1, k+1)$ =
SDCSweep$(\boldsymbol{U}(L-1, k), \boldsymbol{F}(L-1, k), \boldsymbol{\tau}(L-1, k))$;
**if** $n < N-1$ **then**
    Send $U(L-1, k+1, M_{L-1})$ to $\boldsymbol{P}_{n+1}$;
*Return up the V-cycle*
**for** $\ell = L-2 \ldots 1$ **do**
    *Interpolate (time and space)*
    $\boldsymbol{U}(\ell, k+1)$ = Interpolate$(\boldsymbol{U}(\ell+1, k+1))$;
    $\boldsymbol{F}(\ell, k+1)$ = FEval$(\boldsymbol{U}(\ell, k+1))$;
    *Get new initial value, interpolate correction, sweep*
    **if** $n > 1$ **then**
        Receive $U(\ell, k+1, 0) = U(\ell, k+1, M_\ell)$ from
        $\boldsymbol{P}_{n-1}$;
        $U(\ell, k+1, 0)$ =
        Interpolate$(U(\ell+1, k+1, 0))$;
    $\boldsymbol{U}(\ell, k+1)$, $\boldsymbol{F}(\ell, k+1)$ =
    SDCSweep$(\boldsymbol{U}(\ell, k+1), \boldsymbol{F}(\ell, k+1), \boldsymbol{\tau}(\ell, k+1))$;
*Interpolate at finest level (time and space)*
$\boldsymbol{U}(0, k+1)$ = Interpolate$(\boldsymbol{U}(1, k+1))$;
$\boldsymbol{F}(0, k+1)$ = FEval$(\boldsymbol{U}(0, k+1))$;
**if** $n > 0$ **then**
    Receive $U(0, k+1, 0) = U(0, k+1, M_0)$ from $\boldsymbol{P}_{n-1}$;
    $U(0, k+1, 0)$ = Interpolate$(U(1, k+1, 0))$;
```

Algorithm 1: PFASST iteration k on processor $\boldsymbol{P}_n$

the multigrid-like V-cycle in more detail, cf. also Fig. 6. Here, we use the following functions:

- FEval(x): evaluate the right-hand side function $f(t, x)$ for the initial value problem (9) given the solution x (e. g. using PEPC).
- SDCSweep(x, y, τ): perform one SDC sweep (iteration) given the current solution x, y = FEval(x) and FAS correction τ to generate a new solution z. This also

returns `FEval`(z) so that the FAS correction can be computed.

- `FAS`(F^C, F^F, τ^F): Compute the cumulative FAS correction using (16) and (17) given the coarse and fine function evaluations F^C and F^F, and the fine FAS correction τ^F.
- `Interpolate`(x) and `Restrict`(x): Compute the interpolation/restriction of the coarse/fine solution to the fine/coarse grid.

4) Parallel speedup and efficiency: The parallel speedup S of PFASST can be estimated by comparing the cost of a PFASST run with P_T processors to a serial SDC method. The parallel efficiency then is S/P_T. In all comparisons, we assume that the serial SDC and PFASST method compute the solution to approximately the same accuracy, and hence we denote by K_s and K_p respectively the number of serial and parallel iterations needed to achieve the desired accuracy.

Let τ_ℓ denote the cost of the method used for each of M_ℓ sub-steps of the SDC sweep on level ℓ. Hence $\Upsilon_0 = M_0\tau_0$ is the cost of one SDC sweep at the finest level, and the total cost for P_T steps of the serial SDC method is

$$C_s = P_T K_s \Upsilon_0. \tag{21}$$

In the PFASST algorithm, let n_ℓ denote the total number of SDC sweeps performed at level ℓ per PFASST iteration and Γ_ℓ the additional cost of the operations performed for the FAS procedure (restriction, interpolation, and additional function evaluations). If the PFASST iterations converge to the required accuracy in K_p iterations, the total cost on P_T processors assigned to the temporal parallelization is

$$C_p = P_T n_L \Upsilon_L + K_p \sum_{\ell=0}^{L} (n_\ell \Upsilon_\ell + n_\ell \Gamma_\ell). \tag{22}$$

Using these definitions, the parallel speedup is

$$S(P_T) = \frac{C_s}{C_p} = \frac{P_T K_s \Upsilon_0}{P_T n_L \Upsilon_L + K_p \sum_{\ell=0}^{L}(n_\ell \Upsilon_\ell + n_\ell \Gamma_\ell)}. \tag{23}$$

For a two level PFASST run ($L = 1$) the speedup becomes

$$S(P_T;\alpha) = \frac{P_T K_s}{P_T n_L \alpha + K_p(1 + n_L\alpha + \beta)}, \tag{24}$$

where $\alpha = \Upsilon_1/\Upsilon_0$ is the ratio between a sweep at the coarse level ($\ell = 1$) and a sweep at the fine level ($\ell = 0$). Reducing the runtime of coarse sweeps by also coarsening in space reduces Υ_1 and α and hence increases the speedup $S(P_T;\alpha)$ for fixed P_T. The parameter β is the total overhead per iteration relative to Υ_0. Note that the maximum speedup in the two level case is bounded by

$$S(P_T;\alpha) \leq \frac{K_s}{K_p} P_T \tag{25}$$

independently of α. This allows for a maximum parallel efficiency of K_s/K_p in contrast to the much stricter bound of $1/K_p$ in parareal, cf. [19]. Nevertheless, as with parareal, PFASST cannot provide optimal efficiency and is hence considered as an additional direction for parallelization on top of a saturated spatial parallelization.

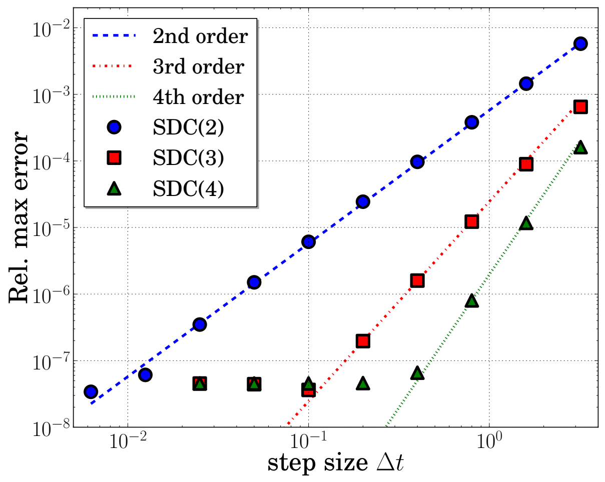
IV. Numerical Results

First we analyze the accuracy and order of integration of SDC and PFASST using our model problem as introduced in Section II. Since no analytical solution is available for this problem (an issue that is typical for 3D fluid flow studies, at least when using open boundaries), we perform a reference run for $N = 10{,}000$ particles with eighth-order SDC and very fine time step sizes, i. e. $\Delta t = 0.01$, from $t_0 = 0$ to $T = 16$. In addition, to eliminate spatial errors, the evaluations of the right-hand sides of Eqs. (5) and (6) are performed using a direct solver with theoretical complexity $\mathcal{O}(N^2)$. Clearly, this procedure is unfeasible for larger number of particles where fast summation methods must be used, but it identifies a set of parameters for which SDC and PFASST yield solutions of comparable accuracy. Thus, we distinguish two different scenarios in the following: accuracy checks are performed using direct summation on small ensemble sizes (see Section IV-A) and performance test are performed with the Barnes-Hut tree code PEPC (see Section IV-B). For the latter, we monitor the residuals of PFASST to ensure that it properly converges towards the SDC solution.

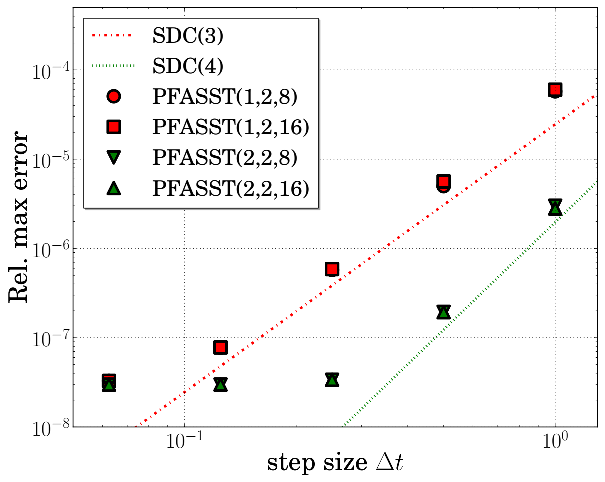
A. Accuracy analysis for a direct particle code

Fig. 7a shows the results of the accuracy tests using time-serial SDC only. Here, the relative maximum errors of $N = 10{,}000$ particle positions are depicted at time $T = 16$ for different step sizes and numbers of SDC sweeps. In each run, three Gauss-Lobatto nodes are used as intermediate points on the fine level, see [34] for a detailed discussion on the choice of quadrature nodes. In addition, we indicate theoretical curves for second-, third- and fourth-order convergence. Clearly, SDC with two, three and four iterations matches these curves down to the limit induced by the number of intermediate collocation nodes used, thus verifying the expected corresponding integration orders.

To obtain third- and fourth-order accuracy, as commonly applied in recent vortex method implementations, see e. g. [27], three and four sweeps of SDC are required in the serial case as expected. In Fig. 7b, these time-serial SDC runs are depicted as dashed/dotted lines. In addition, we show parallel PFASST runs with one and two iterations on three fine and two coarse Gauss-Lobatto nodes. To obtain a good approximation to third-order SDC, one PFASST iteration is sufficient, while for a fourth-order scheme, two iterations are necessary. However, to achieve a comparable level of accuracy as SDC, PFASST requires two coarse sweeps and one fine sweep per PFASST iteration in our case. In Fig. 7b, PFASST(X, Y, P_T) runs with $X = 1, 2$ iterations, $Y = 2$ coarse sweeps and $P_T = 8, 16$ time slices are shown and compared to serial SDC runs, both now indicating similar accuracy orders and levels. For fourth-order integration, fourth-order SDC (SDC(4) in short) with step size $\Delta t = 0.5$ and PFASST with two iterations, two coarse sweeps and step size $\Delta t = 0.5$ yield an accuracy of approximately 10^{-7} in terms of the relative maximum error for the particle positions with respect to our high-order SDC reference run and can thus be compared in terms of parallel performance.



(a) SDC relative maximum error in particle positions w.r.t. high-order SDC reference run



(b) PFASST relative maximum error in particle positions w.r.t. high-order SDC reference run

Fig. 7: Relative max. error of $N = 10{,}000$ particle positions at time $T = 16$ vs. time step size Δt for SDC(X) with $X = 2, 3, 4$ sweeps on three fine nodes (left) and PFASST(X, Y, P_T) using $X = 1, 2$ iterations, each with $Y = 2$ coarse sweeps and $P_T = 8, 16$ time ranks, three fine and two coarse nodes (right). Direct summation of spherical vortex sheet with sixth-order algebraic kernel, $\sigma \approx 18.53h$, $h \approx 0.035$.

B. Speedup results for PFASST *and* PEPC

The space-parallel version of PEPC provides excellent scalability with up to 2 billion particles on $P_S = 300$k cores of the IBM Blue Gene/P installation JUGENE at Jülich Supercomputing Centre, provided the number of particles per compute core is sufficiently large (see Section III-A). In particular, PEPC scales well down to 2,000 particles/core up to $P_S = 8{,}192$ cores and down to 300 particles per core up to $P_S = 512$ cores.

To analyze the additional speedup gained by applying PFASST, we use the example of the spherical vortex sheet as described in Section IV-A for $N_{\text{small}} = 125{,}000$ and $N_{\text{large}} = 4{,}000{,}000$ particles on JUGENE. We have seen that in these cases PEPC with purely spatial parallelism scales well up to approximately $P_S = 512$ and $P_S = 2{,}048$ nodes, respectively. Therefore, we take these scenarios (N_{small} on $P_S = 512$ nodes, N_{large} on $P_S = 2{,}048$ nodes) as the basis for the time-serial SDC(4) runs with $\Delta t = 0.5$ according to our results of Section IV-A.

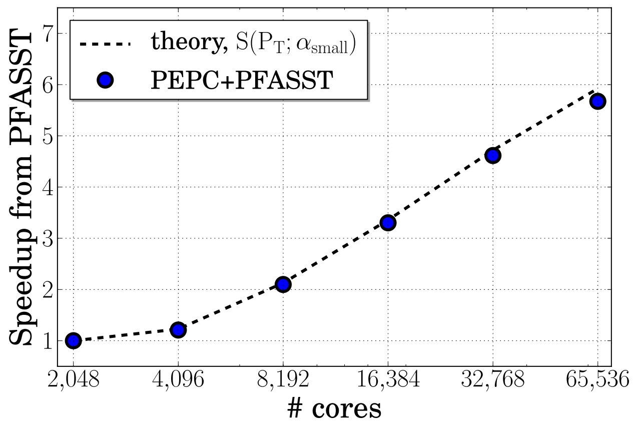
To obtain a fast coarse and an accurate fine propagator for PFASST, we need to define the process of spatial coarsening in the context of particle methods. In the context of the Barnes-Hut approach, this can be done through the multipole acceptance criterion as explained in Section III-A. For ϑ approaching zero, the results (and the runtimes) of the tree code converge to the result (and the runtimes) of a direct summation. Thus, smaller ϑ yield better and slower approximations for the fine propagator, as e. g. higher-order Finite Differences schemes do in the case of mesh-based approaches [21], while larger ϑ yield a faster but less accurate coarse propagator. For the fine and coarse propagators of PFASST$(2, 2, P_T)$, we define $\vartheta = 0.3$ and $\vartheta = 0.6$, respectively, both with $\Delta t = 0.5$. We then note:

- For N_{small} on $P_S = 512$ nodes, the ratio between PEPC runs with $\vartheta = 0.3$ and with $\vartheta = 0.6$ is approx. 2.65, for N_{large} on $P_S = 2{,}048$ nodes we find a factor 3.23. This corresponds to

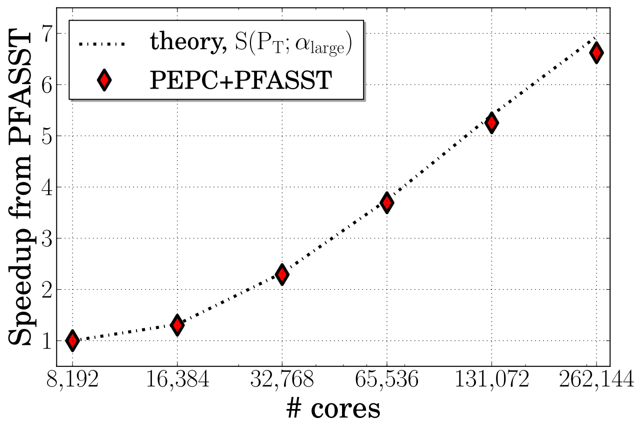
$$\alpha_{\text{small}} = \frac{2}{2.65 \cdot 3} \quad \text{and} \quad \alpha_{\text{large}} = \frac{2}{3.23 \cdot 3}, \tag{26}$$

  in (24), since PFASST uses two collocation nodes on the coarse and three collocation nodes on the fine level.
- To check for convergence of PFASST, we define the residual as the difference between the solution of iteration n and iteration $n + 1$ of PFASST. PFASST$(2, 2, P_T)$ runs with $P_T = 2$ time slices and $\vartheta = 0.3$ on both levels yield residuals of $1.93 \cdot 10^{-5}$ and $1.90 \cdot 10^{-5}$ on slice 1 and 2 after the iterations of the last time step, while PFASST$(2, 2)$ runs with $\vartheta = 0.6$ on the coarse level give $1.93 \cdot 10^{-5}$ and $5.22 \cdot 10^{-5}$ on slice 1 and 2. For the largest run with $P_T = 32$ time slices we find $6.64 \cdot 10^{-7}$ on the first and $0.11 \cdot 10^{-5}$ on the last time slice with $\vartheta = 0.6$ on the coarse level.

Therefore, modifying the multipole acceptance criterion of Barnes-Hut tree codes is a valid and convenient possibility for particle methods to implement the coarsening required in PFASST. The approach is not inhibiting the convergence of PFASST in the examples studied here and provides an acceleration of at least 2.65 for function evaluations of the coarse propagator. As for mesh-based methods, this approach controls the approximation quality of the right-hand side of the evolution equations (5) and (6).



(a) small setup with 125k particles



(b) large setup with 4M particles

Fig. 8: Speedup of PEPC and PFASST$(2, 2, P_T)$ compared to SDC(4), $\Delta t = 0.5$, with different number P_T of parallel-in-time instances on JUGENE, spherical vortex sheet setup with sixth-order algebraic kernel, $\sigma \approx 18.53h$, $h \approx 0.035$. The small example used $P_S = 512$ nodes (2,048 cores) for the spatial parallelization (which corresponds to one time instance in the plot), the larger one $P_S = 2{,}048$ nodes (8,192 cores). Using $P_T = 32$ time slices we end up with 16,384 nodes (65,536 cores) for the small example and 65,536 nodes (262,144 cores) for the larger one.

With these facts at hand, we present the results of the speedup measurements in Fig. 8. The dashed lines $S(P_T; \alpha_{\text{small}})$ and $S(P_T; \alpha_{\text{large}})$ represent the theoretical speedup based on equation (24). Starting from serial SDC(4) runs on $P_S = 512$ and $P_S = 2{,}048$ nodes (corresponding to 2,048 and 8,192 cores on JUGENE) for N_{small} and N_{large}, respectively, PFASST$(2, 2)$ resembles the theoretically predicted scaling very well, even up to extreme scales with 65,536 and 262,144 cores. We stress that speedup of PFASST is measured against the runtime of the time-serial solution with already saturated spatial parallelization. While the parallel efficiency of PFASST is limited according to Eq. (24), time-parallelism allows additional speedup by a factor of seven for the large and a factor of five for the small example. We note that it is not advisable and may even not be possible to use that many cores for purely spatial decomposition of these problem sizes: in our cases, only 16 particles per core for the large and barely 2 particles per core for the small setup would remain after distributing the particles across all available cores in a purely space-parallel approach.

V. Conclusion and Outlook

In this work, we describe and analyze a unique combination of N-body solvers with parallel time integration schemes. We verify integration order and accuracy for SDC and identify matching PFASST variants for a direct summation algorithm with vortex particles. To efficiently use a space-time parallel N-body code on large scales, the algorithmic complexity of the space-parallel part must be reduced and coarse/fine propagators for the time-parallel part are required. We show that both goals can be achieved by means of the multipole-based Barnes-Hut approach. Combining the space-parallel Barnes-Hut tree code PEPC with PFASST, we are able to simulate four million particles on 262,144 cores on JUGENE, which is – to the best of our knowledge – the largest space-time parallel run to date.

In the PFASST approach, the parallel efficiency achieved depends to a large extent on the reduction in computational cost of the coarse problem. For grid-based problems, spatial multigrid techniques can be used to efficiently create a hierarchy of coarse problems, but this approach is not directly applicable to particle systems. Our approach of using the multipole acceptance criterion in Barnes-Hut tree codes is shown to be effective, but more elaborate strategies could further increase the overall efficiency. One possibility is to use a splitting of the force summation by spatial proximity. Then coarse problems could update the contribution from well separated particle clusters less frequently than nearby clusters. The spatial decomposition implicit in the tree structure provides a natural hierarchy of spatial scales, and such a splitting could be combined with the acceptance criterion model used here.

The novel combination of PEPC and PFASST presented in this work appears to be scalable to machines with even more cores than used here: the addition of time-parallelism considerably extends the intrinsic strong scaling limit of classical space-parallel codes and the peak performance for the coupling of both concepts has seemingly not been reached yet.

Acknowledgments

This research is partly funded by the Swiss “High Performance and High Productivity Computing” initiative HP2C, the US Department of Energy and the ExtreMe Matter Institute (EMMI) in the framework of the German Helmholtz Alliance HA216. Computing resources were provided by Jülich Supercomputing Centre under project JZAM04.

References

[1] J. Dongarra, P. Beckman, and al., "The international exascale software roadmap," *Int. Journal of High Performance Computer Applications*, vol. 25, no. 1, 2011.

[2] J. Nievergelt, "Parallel methods for integrating ordinary differential equations," *Commun. ACM*, vol. 7, no. 12, pp. 731–733, 1964.

[3] J.-L. Lions, Y. Maday, and G. Turinici, "A "parareal" in time discretization of PDE's," *C. R. Acad. Sci. – Ser. I – Math.*, vol. 332, pp. 661–668, 2001.

[4] G. A. Staff and E. M. Rønquist, "Stability of the parareal algorithm," in *Domain Decomposition Methods in Science and Engineering*, ser. LNCSE, R. Kornhuber and al., Eds., vol. 40. Berlin: Springer, 2005, pp. 449–456.

[5] M. J. Gander and S. Vandewalle, "Analysis of the parareal time-parallel time-integration method," *SIAM J. Sci. Comp.*, vol. 29, no. 2, pp. 556–578, 2007.

[6] Y. Maday and G. Turinici, "Parallel in time algorithms for quantum control: Parareal time discretization scheme," *Int. J. Quant. Chem.*, vol. 93, no. 3, pp. 223–228, 2003.

[7] Y. Liu and J. Hu, "Modified propagators of parareal in time algorithm and application to Princeton Ocean model," *Int. J. for Num. Meth. in Fluids*, vol. 57, pp. 1793–1804, 2008.

[8] D. Samaddar, D. E. Newman, and R. Sánchez, "Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm." *J. Comp. Physics*, vol. 229, no. 18, pp. 6558–6573, 2010.

[9] L. Berry, W. Elwasif, J. Reynolds-Barredo, D. Samaddar, R. Sanchez, and D. Newman, "Event-based parareal: A data-flow based implementation of parareal," *J. Comp. Physics*, vol. 231, no. 17, pp. 5945 – 5954, 2012.

[10] C. Farhat and M. Chandesris, "Time-decomposed parallel time-integrators: Theory and feasibility studies for fluid, structure, and fluid-structure applications," *Int. J. Numer. Methods Engng.*, vol. 58, pp. 1397–1434, 2005.

[11] C. Farhat, J. Cortial, C. Dastillung, and H. Bavestrello, "Time-parallel implicit integrators for the near-real-time prediction of linear structural dynamic responses," *Int. J. Numer. Methods Engng.*, vol. 67, pp. 697–724, 2006.

[12] J. Cortial and C. Farhat, "A time-parallel implicit method for accelerating the solution of non-linear structural dynamics," *Int. J. Numer. Methods Engng.*, vol. 77, pp. 451–470, 2008.

[13] R. Croce, D. Ruprecht, and R. Krause, "Parallel-in-space-and-time simulation of the three-dimensional, unsteady Navier-Stokes equations for incompressible flow," in *Proceedings of 5th Int. Conf. on High Performance Scientific Computing*, 2012, (Submitted).

[14] J. E. Barnes and P. Hut, "A hierarchical $\mathcal{O}(N \log N)$ force-calculation algorithm," *Nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[15] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comp. Phys.*, vol. 73, no. 2, pp. 325–348, 1987.

[16] E. Aubanel, "Scheduling of tasks in the parareal algorithm," *Parallel Computing*, vol. 37, no. 3, pp. 172 – 182, 2011.

[17] A. Dutt, L. Greengard, and V. Rokhlin, "Spectral deferred correction methods for ordinary differential equations," *BIT Numerical Mathematics*, vol. 40, no. 2, pp. 241–266, 2000.

[18] M. L. Minion, "A hybrid parareal spectral deferred corrections method," *Comm. App. Math. and Comp. Sci.*, vol. 5, no. 2, pp. 265–301, 2010.

[19] M. Emmett and M. Minion, "Toward an efficient parallel in time method for partial differential equations," *Comm. App. Math. and Comp. Sci.*, vol. 7, no. 1, pp. 105–132, 2012.

[20] P. F. Fischer, F. Hecht, and Y. Maday, "A parareal in time semi-implicit approximation of the Navier-Stokes equations," in *Domain Decomposition Methods in Science and Engineering*, ser. LNCSE, R. Kornhuber and al., Eds., vol. 40. Berlin: Springer, 2005, pp. 433–440.

[21] D. Ruprecht and R. Krause, "Explicit parallel-in-time integration of a linear acoustic-advection system," *Computers & Fluids*, vol. 59, pp. 72–83, 2012.

[22] G.-H. Cottet and P. Koumoutsakos, *Vortex Methods: Theory and Applications*, 2nd ed. Cambridge University Press, 2000.

[23] R. Speck, "Generalized algebraic kernels and multipole expansions for massively parallel vortex particle methods," Ph.D. dissertation, Universität Wuppertal, 2011.

[24] G. Winckelmans, J. K. Salmon, M. S. Warren, A. Leonard, and B. Jodoin, "Application of fast parallel and sequential tree codes to computing three-dimensional flows with the vortex element and boundary element methods," *ESAIM: Proceedings*, vol. 1, pp. 225–240, 1996.

[25] R. Speck, R. Krause, and P. Gibbon, "Parallel remeshing in tree codes for vortex particle methods," in *Applications, Tools and Techniques on the Road to Exascale Computing*, ser. Advances in Parallel Computing, K. D. Bosschere, E. H. D'Hollander, G. R. Joubert, D. Padua, F. Peters, and M. Sawyer, Eds., vol. 22, 2012.

[26] J. K. Salmon, M. S. Warren, and G. Winckelmans, "Fast parallel tree codes for gravitational and fluid dynamical N-body problems," *Int. J. Supercomp. App.*, vol. 8, pp. 129–142, 1994.

[27] W. M. van Rees, A. Leonard, D. I. Pullin, and P. Koumoutsakos, "A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high Reynolds numbers," *J. Comp. Phys.*, vol. 230, pp. 2794–2805, 2011.

[28] M. Winkel, R. Speck, H. Hübner, L. Arnold, R. Krause, and P. Gibbon, "A massively parallel, multi-disciplinary Barnes-Hut tree code for extreme-scale N-body simulations," *Comp. Phys. Comm.*, vol. 183, no. 4, pp. 880–889, 2012.

[29] J. E. Barnes and P. Hut, "Error analysis of a tree code," *Astrophys. J. Supp. Ser.*, vol. 70, pp. 389–417, 1989.

[30] J. K. Salmon and M. S. Warren, "Skeletons from the treecode closet," *J. Comp. Phys.*, vol. 111, no. 1, pp. 136–155, 1994.

[31] M. S. Warren and J. K. Salmon, "A parallel hashed oct-tree N-body algorithm," in *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, Portland, Oregon, USA, 1993, pp. 12–21.

[32] ——, "A portable parallel particle program," *Comp. Phys. Comm.*, vol. 87, pp. 266–290, 1995.

[33] P. Gibbon, M. Winkel, L. Arnold, and R. Speck. (2012, August) PEPC website. [Online]. Available: http://www.fz-juelich.de/ias/jsc/pepc

[34] A. T. Layton and M. L. Minion, "Implications of the choice of quadrature nodes for picard integral deferred corrections methods for ordinary differential equations," *BIT Numerical Mathematics*, vol. 45, no. 2, pp. 341–373, 2005.