Institut für Numerische Simulation
Rheinische Friedrich-Wilhelms-Universität Bonn
Endenicher Allee 19b • 53115 Bonn • Germany
phone +49 228 73-69828 • fax +49 228 73-69847
www.ins.uni-bonn.de

R. Lago, M. Obersteiner, T. Pollinger, J. Rentrop,
H.-J. Bungartz, T. Dannert, M. Griebel,
F. Jenko, D. Pflüger

EXAHD – A Massively Parallel Fault Tolerant Sparse Grid Approach for High-Dimensional Turbulent Plasma Simulations

INS Preprint No. 2001

March 2020

EXAHD – A Massively Parallel Fault Tolerant Sparse Grid Approach for High-Dimensional Turbulent Plasma Simulations

Rafael Lago, Michael Obersteiner, Theresa Pollinger, Johannes Rentrop, Hans-Joachim Bungartz, Tilman Dannert, Michael Griebel, Frank Jenko, Dirk Pflüger

Abstract Plasma fusion is one of the promising candidates for an emission-free energy source and is heavily investigated with high-resolution numerical simulations. Unfortunately, these simulations suffer from the curse of dimensionality due to the five-plus-one-dimensional nature of the equations. Hence, we propose a sparse grid approach based on the sparse grid combination technique which splits the simulation grid into multiple smaller grids of varying resolution. This enables us to increase the maximum resolution as well as the parallel efficiency of the current solvers. At the same time we introduce fault tolerance within the algorithmic design and increase the resilience of the application code. We base our implementation on a manager-worker approach which computes multiple solver runs in parallel by distributing tasks to

Rafael Lago
Max Planck Computing & Data Facility, Garching, e-mail: rafael.lago@mpcdf.mpg.de

Michael Obersteiner
Technical University of Munich, Garching, e-mail: michael.obersteiner@mytum.de

Theresa Pollinger
University of Stuttgart, Stuttgart, e-mail: theresa.pollinger@ipvs.uni-stuttgart.de

Johannes Rentrop
University of Bonn, Bonn, e-mail: rentrop@ins.uni-bonn.de

Hans-Joachim Bungartz
Technical University of Munich, Garching, e-mail: bungartz@in.tum.de

Tilman Dannert
Max Planck Computing & Data Facility, Garching, e-mail: tilman.dannert@rzg.mpg.de

Michael Griebel
University of Bonn, Bonn & Fraunhofer Institute for Algorithms and Scientific Computing SCAI, Sankt Augustin, e-mail: griebel@ins.uni-bonn.de

Frank Jenko
Max Planck Institute for Plasma Physics, Garching, e-mail: Frank.Jenko@ipp.mpg.de

Dirk Pflüger
University of Stuttgart, Stuttgart, e-mail: Dirk.Pflueger@ipvs.uni-stuttgart.de

different process groups. Our results demonstrate good convergence for linear fusion runs and show high parallel efficiency up to 180k cores. In addition, our framework achieves accurate results with low overhead in faulty environments. Moreover, for nonlinear fusion runs, we show the effectiveness of the combination technique and discuss existing shortcomings that are still under investigation.

1 Introduction

Scientists widely agree that the human-made climate change due to CO_2 emission will pose severe challenges for the future. Hence, to reduce the overall emission of greenhouse gases, carbon free energy sources will be required. Nuclear fusion is one candidate which offers abundant fuel with a large energy output. However, there are still many difficulties to overcome in the task to build an energy-positive fusion reactor. A major challenge are micro-instabilities caused by turbulent plasma flow. These can be studied by numerical simulations to further improve the design of fusion reactors. One of the codes dedicated to tackle this problem is GENE, which solves the gyrokinetic Vlasov-Maxwell equations.

Unfortunately, the computational costs of these simulations are enormous. Since they involve calculations on a five-dimensional spatial grid, they suffer from the curse of dimensonality, i.e. from the exponential dependence of the grid size on dimension. In our project, we overcome this issue with the help of the sparse grid combination technique. This method splits a simulation into several independent runs on coarser anisotropic grids and then aggregates the results on a sparse grid. As a consequence, the curse of dimensionality is alleviated, which makes larger simulations feasible. In addition, the arising subproblems can be computed in parallel which allows for scaling to larger processor numbers. This enables the use of future exascale computers. Finally, the method opens up an algorithmic approach to fault tolerance, which will be a key requirement for exascale computing.

In this paper, we report on the main contributions of the two funding periods of our project EXAHD within the priority program Software for Exascale Computing of the DFG, with an emphasis on the second funding period. Partners in this joint project were the Institute for Parallel and Distributed Systems at the University of Stuttgart (PI Pflüger), the Institute for Numerical Simulations at the University of Bonn (PI Griebel), the Institute for Informatics at the Technical University of Munich (PI Bungartz), the Max-Planck Institute for Plasma Physics (PI Jenko), the Supercomputing Center of the Max-Planck Society Garching (PI Dannert, second period), and the Center for Mathematics and Its Applications of the Australian National University (Hegland, external partner).

Our main contributions include significant progress for the solution of higher-dimensional plasma flow problems. In particular, we addressed several exascale challenges: We have realized the first-ever massively parallel computations with the sparse grid combination technique, enabling scalability beyond the petascale for mesh-based discretizations by numerically decoupling the underlying systems and

by introducing a novel level of parallelism. We tackled load-balancing on massively parallel systems, learning from gathered runtime data. And we have developed new and innovative approaches for fault tolerance on multiple levels, in particular algorithm-based fault tolerance, which can be a crucial aspect in settings where checkpoint-restart is infeasible due to the massive amount of data that would have to be handled.

In the first funding period we have focused on linear simulations of hot fusion plasma which govern the exponential growth phase of turbulent modes. We have developed core algorithms for fault tolerance and scalability, including algorithm-based fault tolerance with the combination technique, suitable for parallel settings, and we considered optimal communication schemes.

In the second funding period, we have extended the simulation setting to fully nonlinear simulations, which pose several hurdles as they do not match the classical setting for the sparse grid combination technique any more. We have further developed a flexible framework for massively parallel simulations with the combination technique, including software interfaces to GENE and the general PDE framework DUNE. We have realized fault tolerance within both GENE and the combination framework, which even allows to detect and mitigate soft faults, for example due to silent data corruption. Finally, we have extended load balancing to a fully data-driven approach based on machine learning.

In the following, we first give an introduction to the underlying theory and the mathematical model. To this end, we describe the sparse grid combination technique and the gyrokinetic approach to plasma physics underlying GENE. Then, we outline the principles behind our approach to fault tolerance. The third section focuses on the implementation while numerical results are presented in section four.

2 Theory and Mathematical Model

2.1 The Sparse Grid Combination Technique

The sparse grid combination technique [10] is a method for approximating high-dimensional problems based on sparse grids [2]. It yields an alternative representation of a sparse grid solution, not reliant on hierarchical surpluses but using diffferent coarse full grid solutions to build a combination solution. The underlying idea of the sparse grid approximation was first introduced by Smolyak for the case of quadrature [24] and was since applied to a broad field of applications [7, 21].

Let's assume we want to approximate a given function u. Furthermore let $\Omega_{\mathbf{n}}$ be the regular Cartesian grid on $[0,1]^d$ with mesh size $\mathbf{h_n} = 2^{-\mathbf{n}} := (2^{-n_1}, \ldots, 2^{-n_d})$ resulting in $2^{n_i} \pm 1$ points along the i-th direction, depending on whether there are points on the left and/or right boundary. Arbitrary rectangular domains can be treated by scaling them onto the unit hypercube.

Now we can define a piecewise linear approximation of u on $\Omega_{\mathbf{n}}$,

$$u_{\mathbf{n}} = \sum_{\mathbf{j}} u_{\mathbf{n},\mathbf{j}}\, \phi_{\mathbf{n},\mathbf{j}} \tag{1}$$

where $u_{\mathbf{n},\mathbf{j}}$ are the function values at the grid points and $\phi_{\mathbf{n},\mathbf{j}}$ are piecewise d-linear hat functions with support of volume $\prod_{i=1}^{d}(2\,h_{n_i})$ anchored at the grid points, which we call *nodal basis*. There are various other types of basis functions one could choose in order to increase the order of the approximation. For various examples we refer to [2]. In this presentation, we stick to the piecewise linear case for reasons of simplicity.

The function Eq. (1) can also be represented in the so called *hierarchical basis*,

$$u_{\mathbf{n}} = \sum_{\boldsymbol{\ell}\le\mathbf{n}} \sum_{\mathbf{j}} \alpha_{\boldsymbol{\ell},\mathbf{j}}\, \hat{\phi}_{\boldsymbol{\ell},\mathbf{j}} \coloneqq \sum_{\boldsymbol{\ell}\le\mathbf{n}} \hat{u}_{\boldsymbol{\ell}}\,, \tag{2}$$

where the hierarchical basis functions $\hat{\phi}_{\boldsymbol{\ell},\mathbf{j}}$ are those hat functions of level $\boldsymbol{\ell}$ with odd indices only (for $\ell_i > 0$). Here, the expression $\boldsymbol{\ell} \le \mathbf{n}$ is to be understood componentwise. We call the linear transformation between the two representations *hierarchization* and *dehierarchization*, respectively.

Now, if u possesses a certain smoothness property, namely (for our case of piecewise d-linear hierarchical basis functions) that its second mixed derivative is bounded, then the size of the coefficients $\alpha_{\boldsymbol{\ell},\mathbf{j}}$ decays as $\sim 2^{-2|\boldsymbol{\ell}|_1}$. Hence, they are called *hierarchical surplusses*. This leads to the idea of approximating $u_{\mathbf{n}}$ by truncating the sum in Eq. (2). Compared to the full Cartesian grid, merely a subset of points carries a basis function, which is called a *sparse grid*. This can be formalized as follows: For any multi-index set $\mathcal{I}$ that is downward-closed (for any $\boldsymbol{\ell} \in \mathcal{I}$ all $\boldsymbol{\ell}' \le \boldsymbol{\ell}$ are in the set as well) we set

$$u_{\mathbf{n}} = \sum_{\boldsymbol{\ell}\in\mathcal{I}} \hat{u}_{\boldsymbol{\ell}} \tag{3}$$

with $n_i = \max\{\ell_i \,:\, \boldsymbol{\ell} \in \mathcal{I}\}$ defining the *target level*. The classical sparse grid found in the literature for an isotropic grid with $\mathbf{n} = n{\cdot}\mathbf{1}$ is given by $\mathcal{I} = \{\boldsymbol{\ell} \in \mathbb{N}_0^d : |\boldsymbol{\ell}|_1 \le n\}$, i. e. only a standard simplex of levels instead of the full hyperrectangle is taken into account. In this case, as shown in [2], the number of points is drastically reduced, from N^d to $O(N\,(\log N)^{d-1})$, where $N = 2^n \pm 1$. Furthermore, for functions from Sobolev spaces with dominating mixed derivatives H^2_{mix} the asymptotic approximation error with regard to the exact function u is only slighty worse, $O(N^{-2}(\log N)^{d-1})$ compared to the usual $O(N^{-2})$. Generalized sparse grids [4] are defined by means of general downward-closed index sets $\mathcal{I}$ and its associated truncation (3).

Due to the fact that the hierarchical increments $\hat{u}_{\boldsymbol{\ell}}$ can be expressed as differences of full grid functions $u_{\boldsymbol{\ell}}$, Eq. (3) yields a telescopic sum that evaluates to

$$u^{(c)}_{\mathbf{n}} = \sum_{\boldsymbol{\ell}\in\mathcal{I}} c_{\boldsymbol{\ell}}\, u_{\boldsymbol{\ell}} \qquad , \qquad c_{\boldsymbol{\ell}} = \sum_{\mathbf{z}\le\mathbf{1}} (-1)^{|\mathbf{z}|_1}\, \chi_{\mathcal{I}}(\boldsymbol{\ell}+\mathbf{z})\,, \tag{4}$$

with $\chi_{\mathcal{I}}$ being the characteristic function of $\mathcal{I}$. We call this equivalent representation the (generalized) *combination technique*. Note that most of the *combination*

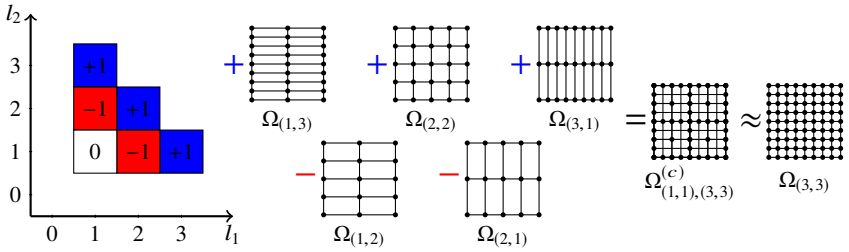

Fig. 1: Grids resulting from a 2D combination technique with $\boldsymbol{\ell}_{\min} = (1, 1)$ and $\boldsymbol{\ell}_{\max} = (3, 3)$. The result of the combination is the sparse grid $\Omega^{(c)}_{(1,1),(3,3)}$.

coefficients $c_{\boldsymbol{\ell}}$ vanish, except the ones whose upper neighbors are not all included in $\mathcal{I}$ (i. e. the ones near the upper boundary of $\mathcal{I}$). The combination technique is advantageous in practical applications, since code that produces full grid solutions can be readily used as a black box and different *component solutions* $u_{\boldsymbol{\ell}}$ can be computed independently from each other, thus introducing a coarse grain layer of parallelism. The point set produced by the union of all component grids is a sparse grid as before. In practice, if one wants to evaluate the combination solution at a point that is not shared by all component grids, the respective component functions have to be interpolated according to the basis used.

The specific combination scheme used throughout this work is the *truncated planar combination technique*, introducing a minimal level $\boldsymbol{\ell}_{\min}$ and a maximal level $\boldsymbol{\ell}_{\max} \equiv \mathbf{n}$. The formula for the index set reads

$$\mathcal{I}_{\boldsymbol{\ell}_{\min},\boldsymbol{\ell}_{\max}} = \left\{ \boldsymbol{\ell} \in \mathbb{N}_0^d : \boldsymbol{\ell} \in \operatorname{conv}\left(\boldsymbol{\ell}_{\min}, \boldsymbol{\ell}_{\min} + \hat{\boldsymbol{\ell}}^{(1)}, \ldots, \boldsymbol{\ell}_{\min} + \hat{\boldsymbol{\ell}}^{(d)} \right) \right\} , \qquad (5)$$

where $\hat{\boldsymbol{\ell}}^{(i)} = (0, \ldots, \ell_{i,\max} - \ell_{i,\min}, \ldots, 0)$, i. e. it includes all level indices that lie inside the simplex spanned by $\boldsymbol{\ell}_{\min}$ and its adjacent corners of the hyperrectangle defined by $\boldsymbol{\ell}_{\min}$ and $\boldsymbol{\ell}_{\max}$. For example, the 2D sparse grid constructed with $\boldsymbol{\ell}_{\min} = (1, 1)$ and $\boldsymbol{\ell}_{\max} = (3, 3)$ is shown in Fig. 1, where the simplex is just a shifted triangle.

As a remark, the approximation rate shown above does not, in general, hold in the case of solving PDEs with the combination technique, which we are interested in. However, it can be shown [3] that the combination technique produces the same rate, as long as a certain ANOVA-like error expansion exists. In fact, the combination technique is applicable to any quantity (not just functions) that is the solution to a problem depending on some discretization parameters, provided there is such an error expansion. Alternatively, there is a second route to prove that the sparse grid Galerkin approximation and the combination method for a PDE possess errors of the same order. To this end, in [8] optimal convergence rates of the combination technique for elliptic operators acting on arbitrary Gelfant triples were shown.

Nevertheless, since it may be too hard to proof for a given case that the necessary error expansion exists, or that the respective Gelfand triple indeed leads to

equal approximation rates between the sparse grid Galerkin approach and the combination method, this motivates the use of numerical experiments to determine the applicability of the combination technique.

The type of PDE we will be concerned with is a time-dependent initial value problem. We will, however, not include the time direction as a dimension of the sparse grid approximation, but rather treat it as a parameter. The reason is that the solver we will mainly deal with employs an explicit time stepping method, which means that anisotropic grids with large time steps may be forbidden by a CFL condition. The solver also dynamically adapts the time step size, so that it may be disadvantageous to interfere with its capabilities.

Another difficulty is that the global truncation error introduced by a time stepping scheme usually grows exponentially in time and, for a nonlinear PDE, even the exact solution may be highly sensitive to perturbations in the initial condition (which are always present due to different spatial discretizations). In consequence, we cannot expect the component solutions to stay similar for long simulation times and so the combination technique would break down.

Hence, we will look for the solution after some interval ΔT, starting at time t, then repeat the process beginning at time $t + \Delta T$ and so forth. Using the combination technique for the remaining (spatial) dimensions, the solution is constructed from the time evolution of the component solutions:

$$u_{\mathbf{n}}^{(c)}(t + \Delta T) \coloneqq \sum_{\boldsymbol{\ell} \in \mathcal{I}} c_{\boldsymbol{\ell}} \, \mathcal{T}_{t \to t+\Delta T} \left[u_{\boldsymbol{\ell}}(t) \right] \tag{6}$$

with the time evolution operator $\mathcal{T}$. The initial values are set by interpolating the combination solution from the previous time step onto the different component grids. Once in the beginning they are set by discretizing a known initial function u_0:

$$u_{\boldsymbol{\ell}}(t) \coloneqq I_{\boldsymbol{\ell}} \left[u_{\mathbf{n}}^{(c)}(t) \right] \qquad , \qquad u_{\boldsymbol{\ell}}(0) \coloneqq I_{\boldsymbol{\ell}} \left[u_0 \right] , \tag{7}$$

where $I_{\boldsymbol{\ell}}$ denotes the interpolation operator onto grid $\Omega_{\boldsymbol{\ell}}$. The interval ΔT has to be chosen sufficiently small, so that the error introduced by the time evolution is small compared to the spatial discretization error (which we want to optimize by using the combination technique). Sensible values for ΔT depend on the scenario at hand and will be investigated in our experiments.

2.2 Plasma Physics with GENE

The most common way to study turbulence on a microscopic scale are the gyrokinetic equations. They are a system of partial integro-differential equations for the particle distribution function f in a 5D phase space plus time for every particle species s in the plasma (typically hydrogen ions and electrons). The model consists of a Vlasov equation (when neglecting collisions) with Lorentz force,

$$\frac{\partial f_s}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f_s + \frac{Z_s e}{m}\left(\mathbf{E} + \frac{\mathbf{v}}{c} \times \mathbf{B}\right) \cdot \nabla_{\mathbf{v}} f_s = 0\,, \tag{8}$$

coupled with Maxwell's equations for the electromagnetic fields as well as the self-consistent computation of charge and current density from the respective moments of the distribution function [6].

The gyrokinetic approach uses several approximations tailored to the setting of a suspended plasma in a strong toroidal magnetic field, which is the concept behind tokamaks and stellarators, the most popular types of fusion reactors. Since charged particles move on helical trajectories along magnetic field lines, the motion is split into the direction parallel to field line, $v_\parallel$, and the plane perpendicular to it. The phase information of the rapid circular motion in the perpendicular direction is then averaged out and its magnitude encoded in the magnetic moment $\mu \propto |v_\perp|^2$. Also, the coordinate system is commonly aligned to the magnetic field lines and, lastly, a δf-splitting is employed, dividing the distribution function into a static equilibrium distribution and an unknown turbulent part g.

Eventually one arrives at a $5+1$ dimensional nonlinear partial differential eqation for $g(x, y, z, v_\parallel, \mu; t)$ of the general form

$$\frac{\partial g}{\partial t} = \mathcal{L}\, g + \mathcal{N}(g) \tag{9}$$

where $\mathcal{L}$ and $\mathcal{N}$ are the linear resp. nonlinear parts of the differential operator.

One of the most widely used codes to solve the gyrokinetic equations is GENE [17]. It is a Eulerian (fixed grid) solver that uses mostly finite differences to discretize the spatial domain and an explicit Runge-Kutta scheme of fourth order for the time evolution. Even though it is highly optimized and employs domain decomposition to scale well on large computing systems, it still requires huge amounts of computation time and for some scenarios the desired resolution is still out of scope. This is why we deem GENE a good candidate to profit from the combination technique. Not only are the component solutions cheaper in terms of memory, hence always feasible to compute, but additionally can afford a larger time step size (due to the CFL condition), reducing the computational cost further. GENE operates in different modes described here for later reference:

linear/nonlinear – In the linear mode, $\mathcal{N}(g)$ in Eq. (9) is neglected. In this case one is interested in the growth rate of the turbulence, i. e. the eigenvalue of $\mathcal{L}$ with the largest real part and its corresponding eigenvector. To this end, GENE can either be used as a direct eigenvalue or as an initial value solver. The nonlinear mode treats the full equation and is used to study the quasi-stationary state after the initial growth phase.

local/nonlocal – The local case uses the so called *flux tube approximation*, simulating only a small cross section of the full torus. Periodic boundary conditions are employed in the x- and y-direction, enabling Fourier transformation. In nonlocal runs the whole domain along the x-direction is treated, losing periodicity in x.

adiabatic electrons – Setting the number of species to one (only ions), the electrons are not treated with the full kinetic model, which reduces the problem size. This approximation was mostly used for our results.

In order to use GENE within our combination technique framework, several adjustments to GENE had to be implemented during the project. They are minor enough to not undermine the black box functionality. First of all, the grid setup of GENE had to be adjusted so that grids using 2^{ℓ_i} grid points in each respective dimension become nested. This is important because, on one hand, the combination technique requires nested grids and, on the other hand, powers of two are necessary for an efficient parallelization in GENE. As a result, only the left boundary may carry a grid point (cf. Section 2.1), which was originally not the case for every direction. Additionally, the μ-direction uses a Gauss-Laguerre grid by default for efficient quadrature. Here we had to switch to an equidistant grid, losing accuracy, but a potential solution to this problem using Clenshaw-Curtis points is discussed in [18]. Furthermore, GENE requires an unusual quasi-periodic boundary condition in the z-direction, which we had to incorporate into the interface of our framework.

Finally some performance issues had to be overcome. At every restart, GENE performs an initialization routine that produces a large overhead when the simulation time itself is short. In particular for nonlocal runs, a large *gyromatrix* has to be assembled which is costly. To this end we had to integrate the functionality to store and reload this matrix from memory. The same had to be done for the checkpoints which could previously only be written to and read from disk [13].

2.3 Fault Tolerance

2.3.1 Fault Tolerant Combination Technique

The Fault Tolerant Combination Technique (FTCT) is an algorithm-based fault tolerant version of the Sparse Grid Combination Technique, which was proposed in the project proposal of phase one of EXAHD and first published in [12]. The method addresses the problem of both hard faults and soft faults[1]. Since these faults cause a corruption or loss of data on certain processors, it is not guaranteed that all combination grids can safely contribute to the combination solution. After identifying such a fault we therefore need to construct a new combination scheme that excludes faulty parts from the original scheme. Ultimately, this modified scheme should only consist of existing component grids, thus eliminating the necessity to recompute results. In addition, switching to less and possibly coarser component grids should only slightly decrease the accuracy compared to the original scheme. In case of a time-dependent problem with frequent recombination, the FTCT restores the original combination

[1] Hard faults are detected by the operating system and usually cause the affected system part to fail while soft faults remain undetected by the system and usally appear in the form of bitflips during computation or processing of data.

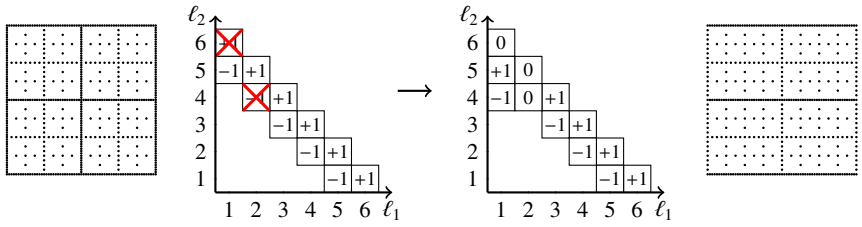

Fig. 2: Recovering the combination scheme after two failed component grids (middle left). A new scheme (middle right) is constructed which excludes the failed resources while utilizing a component grid from the next lower subdiagonal. The resulting sparse grid (right) should be only slightly less accurate than the origianl sparse grid (left).

scheme for all future combinations[2] and continues computation. We will briefly summarize the main steps of the algorithm: fault detection and reconstruction of a fault-free combination scheme.

Fault detection heavily depends on the class of faults that we experience. For hard faults or process failures, typically the operating system in combination with the MPI runtime can be used to identify all processes which are affected by a fault. These faulty ranks are then removed and computation can be continued. In case of soft or silent faults it is more challenging to detect a faulty component as the failure is undetected by the operating system. Therefore, dedicated routines are used to detect such faults, e.g. running a program multiple times or with different algorithms or by checking application dependent properties. An example of the latter case can be found in Section 3.3 where we describe our implementation of the silent fault detection.

Once all faulty component grids are detected, the FTCT constructs a new fault-free combination scheme (see Fig. 2). The problem of finding such a new scheme is known as the General Coefficient Problem (GCP). For further information about the problem and on how to solve it most efficiently we refer to [11, 12]. In order to increase convergence speed and the probability of finding such a solution we already compute the component solutions of two additional lower diagonals[3] of the level set from the start, alongside the required ones, even though they originally have a coefficient of zero. Since the degrees of freedom of component grids decrease exponentially with the magnitude of the level index, this adds only minor overhead to the overall computation time. Section 3.3 outlines our implementation of the FTCT in more detail.

[2] In case of process failures either failed resources are replaced by spare ranks or the individual tasks are distributed to the remaining fault-free processes.

[3] For dimensions $d > 2$ it is in fact a $d-1$ dimensional slice and it might be tilted in general.

2.3.2 Fault Recovery Algorithms

In addition to the FTCT, a fault tolerant version of GENE was developed (FT-GENE). The purpose was to allow FT-GENE to tackle small faults, whereas larger faults would be handled by the FTCT. To that end, an object oriented Fortran 2008 library called libSpina was written. The purpose of this library is to provide functionalities that aid the implementation of a fault tolerant application. In contrast to existing tools such as ULFM [1], libSpina does not replace any existing MPI implementation and does not attempt to extend the MPI standard.

The library is responsible for managing and separating "spare nodes" from the application, the management of a channel for broadcasting errors, detection of faulty resources and sanitization of the MPI environment. Instead of forcing a complete rewriting of the code in order to fit a fault tolerant framework, libSpina provides several preprocessor macros for encapsulating tasks such as exception handling and timeout-based error detection. Some checkpointing and message logging capabilities are available, but mostly for the initialization steps of FT-GENE. The use of the library causes a negligible overhead (circa 3%).

Since libSpina is just a tool designed to assist in programming, it does not provide any specific data recovery mechanism. Whenever a fault occurs, the lost data must be recovered by checkpoint/rollback strategies or an algorithmic/approximate recovery, which is not addressed by the library.

FT-GENE inherits the exception handling mechanism of libSpina and is fully written using non-blocking MPI calls thanks to libSpina's macros. Additionally, FT-GENE is able to withstand faults and provides two recovery methods: checkpoint/rollback and blank recovery (purely for testing purposes) which are discussed later.

3 Implementation

3.1 Parallel Implementation of the Combination Technique

The main parallelization strategy of our combination framework is the *manager-worker approach*. In this strategy, a dedicated process – the manager – distributes work packages to workers which then process their tasks. Within the combination technique these tasks consist of solving the respective PDE on a specific component grid. Since these grids will usually be still too large to be solved on one MPI rank, we assign the tasks to a group of workers which we will call a *process group*. To avoid a communication bottleneck at the manager, only one dedicated rank in each process group – the master – communicates with the manager. This master process then broadcasts all information from the manager to the remaining processes in the process group. In Fig. 3 one can see an example of the task distribution to individual process groups.

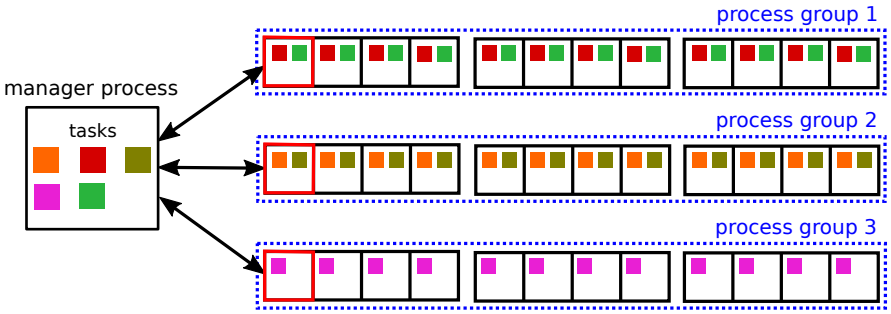

Fig. 3: The manager-worker implementation in our combination framework [13]. The manager assigns tasks to process groups which perform a domain decomposition to work on the tasks in parallel.

The main concern of our framework is achieving high parallel efficiency for potential exascale computing. Due to the independence of the component grids, we can solve all tasks independently. Aditionally, all process groups compute each task in parallel by applying domain decomposition. This results in two levels of parallelism: parallelism between process groups and within process groups.

Unfortunately, in a chaotic time-dependent setting, we cannot completely decouple all tasks, as the individual solutions would eventually drift too far apart from the exact solution resp. from each other, which would deteriorate the result of the final combination. To avoid this effect, frequent recombination is applied (see Figure 4). Here, we construct the sparse grid solution after a short simulation interval by gathering all component grids, and then redistribute the aggregated information to proceed with the computation. This process introduces a global synchronization point as well as global communication. To ensure high parallel efficiency, we therefore require efficient load balancing (see Section 3.2) as well as an efficient recombination mechanism.

The recombination (Fig. 5) is split into three phases: hierarchization, global reduction of the sparse grid and dehierarchization. In the hierarchization step, each

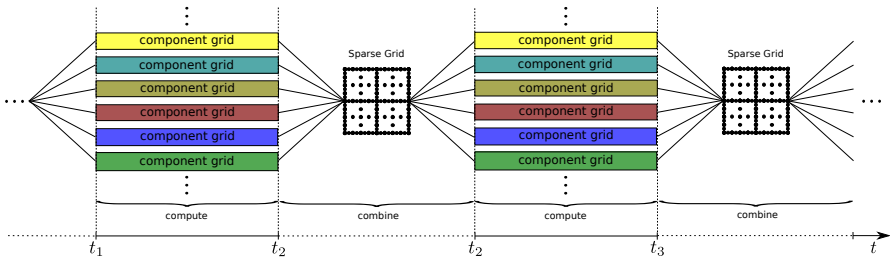

Fig. 4: Solving time-dependent PDEs with the combination technique [13]. Frequent recombination is applied in between computation phases to avoid that the individual solutions drift too far apart from each other.

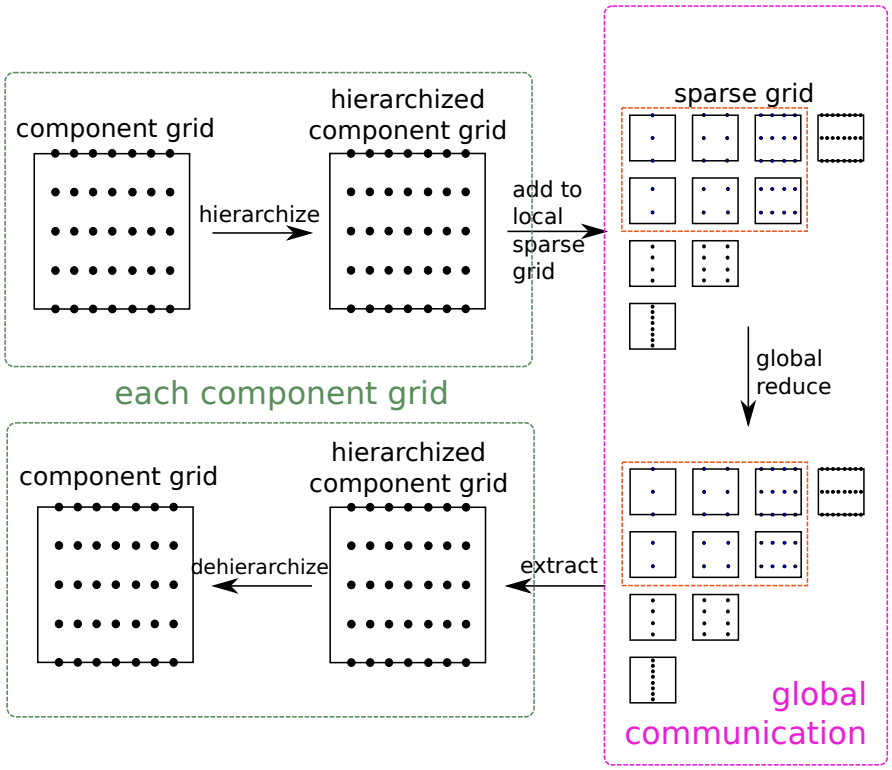

Fig. 5: The recombination step in detail [13]. After the hierarchization phase (top left), the sparse grid is globally reduced between process groups (right) and then dehierarchized within each process group (bottom left).

process group transforms the local solutions from its assigned tasks into the hierarchical basis. These hierarchized solutions are then added locally to the respective subspaces of the corresponding sparse grid. To this end, each process holds the part of the sparse grid that corresponds to their geometric subdomain in a buffer.

Thereafter, the global communication phase begins in which the partial sparse grid solutions from all process groups are added together with an *MPI_Allreduce*. In order to achieve low communication overhead, all process groups are constructed from the same number of ranks and apply the same domain decomposition on the sparse grid. A rank therefore directly communicates only with the ranks of other process groups that are assigned to the same subdomain. This procedure heavily reduces the number of communications and ensures a low communication overhead. Once the global communication has finished, the individual process groups extract the hierarchical information for their component grids from the sparse grid. The procedure is concluded by the dehierarchization step which transforms the data back to the nodal basis. It should be noted that only the second phase requires

global communication while the hierarchization and dehierarchization only require communication within each process group.

3.2 Load Balancing

Load balancing for HPC simulations is routinely implemented via different standard techniques, such as domain decomposition or resource assignment. With the massively parallel distributed combination technique framework, there is an algorithmic way of performing load balancing. Due to the manager-worker scheme with process groups, cf. Fig. 3, the manager is free to assign multiple component grids to different process groups (provided there is enough free memory). In particular, this assignment can be chosen to balance the runtimes such that process groups do not have to wait for each other longer than is necessary. An illustration is given in Fig. 6.

Of course, this assignment works best if accurate estimates of the individual grid runtimes can be obtained beforehand; then, we can even statically assign the grids. Reaching further, this can be improved using the information obtained in the first solver run: by collecting the runtimes of the (presumably) longest-running grids, the process groups can be "filled" with work. The initial dynamic "work-stealing" approach was presented by Heene, Kowitz, and Pflüger [14]. The next step, dynamic reassignment according to load imbalances has not been implemented so far, since the reassignment of a grid to another process group could become very costly. In addition to the field grid data, extensive simulation data is required for `GENE`, such as the gyromatrix; it would have to be transferred or explicitly recomputed on the new process group. Now, using the estimates obtained by the model, a decreasingly ordered list can be created, and the grids can be assigned to the process group. This means that the relative ordering between tasks is more important than the absolute accuracy of the model.

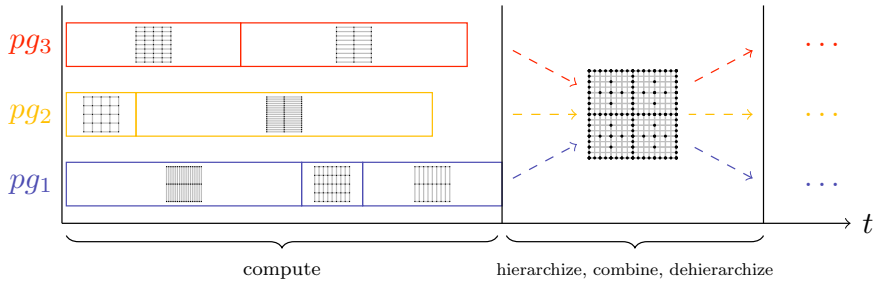

Fig. 6: Possible parallel computation scheme for different 2D grids, cf. [22]. In principle, the hierarchization step for a task could happen immediately after its completion, further improving load balancing. This is not included in our implementation to date, but this does not prove to be a significant bottleneck.

The simplest approach would be to just take the number of grid points into account [9], assuming that all grids of the same level sum $|\boldsymbol{\ell}|_1$ take the same time to run. In larger scenarios, this will lead to variable results, as the assignment will be ambiguous. A more expert knowledge based model can be designed by adding a dependence on grid anisotropy [14]. In this model the runtime contribution of the number of grid points N are estimated with a function $r(N)$,

$$r(N) := mN^k + b\,, \tag{10}$$

where the coefficients m, k and b are fitted to runtime data of isotropic grids. Moreover, the influence of anisotropy is added through an additional term $h(\mathbf{s}_{\boldsymbol{\ell}})$, where $s_{\boldsymbol{\ell},i} = \frac{\ell_i}{|\boldsymbol{\ell}|_1}$. The total runtime estimate $t(N, \mathbf{s}_{\boldsymbol{\ell}})$ then reads

$$t(N, \mathbf{s}_{\boldsymbol{\ell}}) = r(N) \cdot h(\mathbf{s}_{\boldsymbol{\ell}}) \qquad h(\mathbf{s}_{\boldsymbol{\ell}}) := c + \sum_{i=1}^{d-1} c_i s_{\boldsymbol{\ell},i} + \sum_{i=1}^{d-1} \sum_{j \le i} c_{ij} s_{\boldsymbol{\ell},i} s_{\boldsymbol{\ell},j} + \dots\,. \tag{11}$$

Again, the coefficients c in the polynomial ansatz for $h(\mathbf{s}_{\boldsymbol{\ell}})$ are fitted to runtime data, this time including anisotropic grids.

For nonlinear, global simulations, the load modeling was recently extended to fully data-driven techniques, such as support vector regression and neural networks [22].

3.3 Fault Tolerant Combination Technique

Fault tolerance is becoming more and more important in exascale frameworks as the increasing process number will most certainly also increase the probability for some components to fail. In our case we assume that such an error will most likely occur during the computation phase, which takes the majority of the overall runtime. The computation step will therefore not complete for all affected process groups. In the next section we briefly summarize the progress made in [19, 15, 20, 16].

Such faulty groups are detected by the manager at the beginning of the global communication phase with our fault simulation layer. This simulation layer imitates the behavior of ULFM and returns an error signal if a process has failed that should interact in a certain communication. It is also possible to simulate the process failure of specific ranks at specific simulation points. Once the affected groups are known,

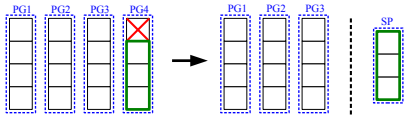

Fig. 7: If a process failure occurs in one of the process groups the remaining ranks are declared as spare processes [19].

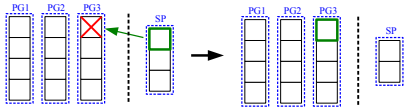

Fig. 8: In case all failed ranks can be replaced by spare processes, the process group is restored [19].

the manager process calculates a fault-free combination scheme by solving the GCP (cf. Section 2.3.1). We use GLPK [5] to solve this optimization problem which returns the new combination scheme.

Unfortunately we cannot proceed directly with the communication at this point since failed ranks are contained in the MPI communicators. The naive implementation would just remove all failed process groups and continue computation on the remaining groups. This procedure, however, might waste valuable resources if only few of the ranks in a process groups have failed. We therefore save these non-faulty ranks and declare them as spare ranks (Fig. 7). For future faults we can now substitute failed ranks by spare ranks to restore a process group (Fig. 8). This enables us to simulate high failure rates without quickly losing all process groups.

Once the MPI environment is restored, our implementation proceeds with the recombination step by applying the global communication to the fault-free combination scheme. For future computations we restore the original combination scheme by redistributing failed tasks to the remaining process groups. These tasks are then initialized by extraction from the sparse grid that results from the temporary fault-free combination scheme.

4 Numerical Results

4.1 Convergence

A very important task in order to confirm the applicability of the sparse grid combination technique to GENE is to show that we can produce meaningful results while reducing the computational cost. This has to be verified in different ways and for different quantities, depending on the setting of the simulation. Therefore, this section is split into two parts.

The first one is concerned with convergence results for local linear GENE runs, where the accuracy of computations of the growth rate $\lambda_{\max}$ and its corresponding eigenvector $\mathbf{g}^{(\lambda_{\max})}$ is studied. The second part presents results for nonlinear GENE simulations, in particular the most recent investigations of nonlocal runs. Here, one is interested in time averages of certain quantities of interest during the quasi-stationary phase, mainly the mean heat flux Q_{es}.

4.1.1 Linear Runs

Linear simulations only use the linear part of the gyrokinetic equations,

$$\frac{\partial \mathbf{g}}{\partial t} = \mathcal{L}\,\mathbf{g}\,, \tag{12}$$

according to Eq. (9) (the bold face $\mathbf{g}$ signifies the discretized version). The dynamics of this equation can be understood by considering an eigenvalue decomposition (assuming every eigenvalue has multiplicity one)

$$\mathbf{g}(t) = \sum_{\lambda \in \sigma(\mathcal{L})} \alpha^{(\lambda)}(t)\,\mathbf{g}^{(\lambda)} \qquad , \qquad \mathcal{L}\,\mathbf{g}^{(\lambda)} = \lambda\,\mathbf{g}^{(\lambda)}\,. \tag{13}$$

Note that $\mathbf{g}$ is complex valued because a Fourier basis is always used in the y-direction and, hence, eigenvalues will be complex in general.

Plugging this decomposition into Eq. (12) yields the solution

$$\mathbf{g}(t) = \sum_{\lambda \in \sigma(\mathcal{L})} \alpha^{(\lambda)}(0)\,e^{\lambda t}\,\mathbf{g}^{(\lambda)}\,. \tag{14}$$

The exponential growth (or decay) will, for large times, be dominated by the eigenvector corresponding to the eigenvalue $\lambda_{\max}$ which has the largest real part. This growth rate plus eigenmode are the quantities one is interested in. To this end, GENE can be run either as a direct eigenvalue solver or as an initial value solver. For the latter case, one initializes a random initial state and simulates long enough, until the shape of $\mathbf{g}(t)$ stays constant and its ratio at subsequent times settles on $e^{\lambda_{\max}\Delta t}$.

First results for the linear case were obtained during the first funding period of this project. The combination technique was successfully applied to both the eigenvalue solver and initial value runs. In each case, combining the eigenvalues as well as the eigenvectors produced by the component solutions yielded satisfactory results. We refer to [18] and omit the details here.

Instead we want to focus on results obtained for local linear initial value runs with the massively parallel framework described in Section 3.1, using the possibility of frequent recombination after short time intervals. The first studies with this framework were conducted during the first half of the second funding period and are recorded in [13]. They examined the convergence of the eigenvector in a GENE test case simulating ion temperature gradient (ITG) driven instabilities.

The following combination schemes were set up for this experiment, always using the z-, $v_\parallel$- and μ-directions for combination (the k_x- and k_y-direction were fixed at level 3 and 1, respectively, as not much resolution is needed in these dimensions in local linear settings): The terms "combi 4 grids" and "combi 10 grids" describe schemes where $\boldsymbol{\ell}_{\min}$ and $\boldsymbol{\ell}_{\max}$ are $(0,0,1,1,1)$ resp. $(0,0,2,2,2)$ apart. This results in the mentioned number of component solutions. Finally, "combi lmin" means that $\boldsymbol{\ell}_{\min} = (3,1,4,4,4)$ is kept fix regardless of maximal level. In order to calculate the error of the obtained eigenvectors, a reference solution was computed on a high

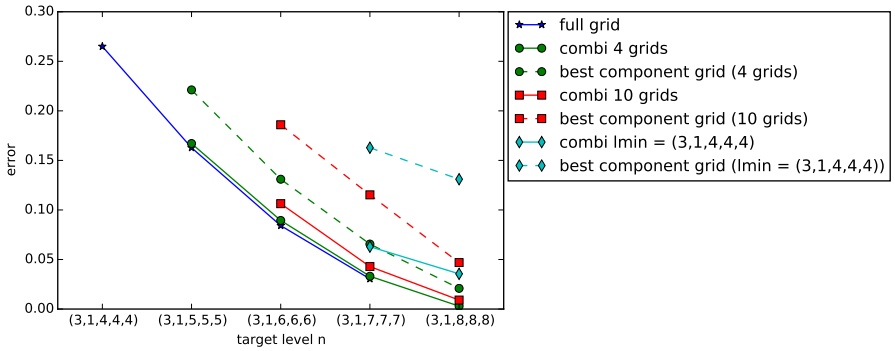

Fig. 9: Error of $\mathbf{g}^{(\mathcal{I}_{\max})}$ compared to the reference solution with $\boldsymbol{\ell} = (3, 1, 8, 8, 8)$. The combined solutions were obtained by combining after each time step with a total of 6000 time steps.

resolution full grid with $\boldsymbol{\ell} = (3, 1, 8, 8, 8)$. All other solutions were interpolated to this reference grid and normalized to unity (because only the shape matters), then the L_2-norm of the difference in absolute value was taken as the error.

Fig. 9 shows results for all combination schemes as well as single full grid solutions for comparison. Here, a recombination was applied after each GENE time step, which turned out to yield the best overall results. One can see that the error decreases with higher target levels and that the combined solutions compare well to the full grid convergence. The higher the number of component grids, the worse the error as interpreted at the target level, but keep in mind that the number of degrees of freedom is more and more reduced. All combination schemes also beat the best error achieved with one of their component grids, which is a crucial test, since otherwise one could be content with that one component solution.

We also have to show that the computational cost is reduced. To this end, different combination intervals (in number of time steps) have been investigated as shown in Fig. 10. It turns out that the overhead for recombination after each time step was

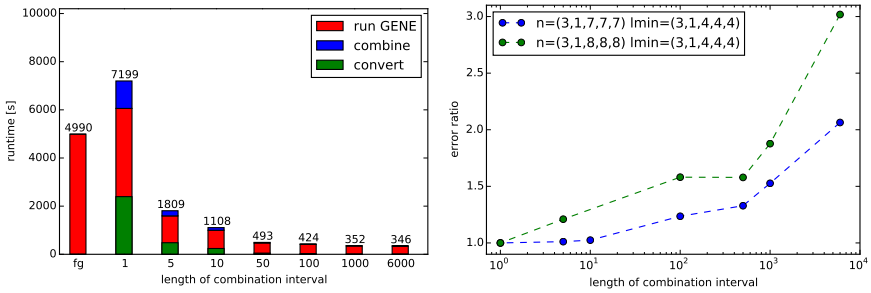

Fig. 10: **Left**: Computation times for "combi lmin" with $\mathbf{n} = (3, 1, 8, 8, 8)$ at different combination intervals and for the reference full grid solution. Four process groups with 16 processes each were used. **Right**: Relative increase of the error for $\mathbf{g}^{(\mathcal{I}_{\max})}$ for different combination intervals.

too large. There are two main reasons. Firstly, this test case was comparatively small so that the relative overhead is more significant than for much larger problem sizes. Secondly, the restart behavior of GENE (as discussed in Section 2.2) is still not optimal so that the pure GENE runtime is much larger for frequent recombination. However, the figure also shows that there is a middle ground. For example, the run time of "combi lmin" with $\mathbf{n} = (3, 1, 8, 8, 8)$ at a combination interval of 10 is already lower by a factor 5 than that of the reference solution. At the same time, the error is only increased by about 25% compared to the optimum.

In conclusion, the applicability of the combination technique with the massively parallel framework has been demonstrated for a small test case. The benefit is expected to be even bigger for larger problem sizes.

4.1.2 Nonlinear Runs

In nonlinear simulations the initial exponential growth phase is halted after some time by the increasingly dominant influence of the nonlinear part of the PDE. Henceforth, the distribution function is subjected to chaotic dynamics. This means that slight disturbances will completely alter the trajectory in the long run (butterfly effect) and, thus, the distribution function itself is not a suitable observable anymore. Still, distinctive patterns in a typical trajectory can be observed that result in a quasi-stationary state. Common observables (often averages or moments of the distribution function) will statistically fluctuate around relatively robust mean values that only depend on the problem parameters. Hence, the goal of the combination technique should be to accurately reproduce said quantities of interest with a similar statistical uncertainty. Throughout this chapter, an ITG test case with adiabatic electrons is studied. We switched to nonlocal simulations to increase the problem size in line with the conclusion of the previous section.

In order to familiarize ourselves with the behavior of important quantities of interest, we initially tested directly combining them instead of g. That is, we choose a combination scheme and, once all component runs are finished, we calculate the time averages and linearly combine them with the appropriate combination coefficients. The method we used to determine when the quasi-stationary phase begins and to estimate the statistical uncertainty follows [25].

We focused on the mean heat flux Q_{es} because it is one of the most statistically robust quantities. Results for various combination schemes in different dimensions are

$\boldsymbol{\ell}_{\max}, \boldsymbol{\ell}_{\min}$	Reference	Direct combination	Best component
(**10**,5,**6**,5,4), (**7**,5,**3**,5,4)	21.7 ± 0.4	23.1 ± 0.9 (6% 0.66)	23.7 ± 0.4 (9% 0.13)
(**9**,5,**5**,**6**,**5**), (**6**,5,**3**,**4**,**3**)	20.2 ± 1.0	28.2 ± 8.1 (40% 0.33)	29.9 ± 0.5 (48% 0.04)
(**11**,5,**5**,**7**,**4**), (**7**,5,**4**,**4**,**3**)	21.8 ± 0.5	21.6 ± 1.4 (1% 0.37)	22.8 ± 0.4 (5% 0.04)

Table 1: Results for direct combination of Q_{es}. The two values in parentheses denote the relative error and the fraction of work load (in core-hours) in relation to the reference solution.

summarized in Table 1. The reference values in each case were computed with a run at target resolution. Unfortunately, adding and subtracting statistically independent quantities causes the variance of the resulting quantity to be the sum of variances of the summands scaled by the square of their combination coefficient. Since we take the standard deviation as a measure of uncertainty, we get $\sigma^{(c)} = (\sum_i c_i^2 \sigma_i^2)^{1/2}$. This is why the uncertainty for the second scheme became very high. Furthermore, the combined values are often not significantly better than the best component solution.

The problem is that finding an efficient level set is a balancing act. On one hand, the minimal level is restricted by GENE simulations becoming erroneous for too low resolutions when physically relevant scales are no longer resolved. On the other hand, the maximal level is bounded by running into the statistical uncertainty so that an increase in accuracy is wasted. This could be addressed by including the simulation length as another dimension in the combination technique but decreasing the uncertainty is always expensive since it scales only with the inverse square root of the simulation length. On top of all this, there is a large discrepancy in the influence of different dimensions, in particular, the error is mostly dominated by the resolution in x-direction. The third scheme was chosen in a fashion to address these issues and it shows the best agreement with the reference. We currently also work on optimizing index sets with a *dimensionally adaptive* algorithm [4] to further mitigate said issues.

Recently, we obtained first results with the massively parallel framework applied to the nonlinear test case. In this context we turned back to frequently recombining g even in the nonlinear regime. Much care has to be taken since the trajectories on different component grids have the tendency to drift apart fairly quickly. Thus, the combination interval should be chosen sufficiently small. However, in contrast to the linear case, we found that recombining after each or just a few time steps leads to the simulation becoming unstable and the distribution function growing uncontrollably. We had to increase the combination interval until stable trajectories were achieved. Also, depending on which and how many dimensions were included in the combination scheme, simulations could become unstable. The causes of these effects are not yet understood and currently under investigation. We suspect that the perturbation introduced by distributing the combined checkpoint to the component grids is large enough so that each trajectory needs a certain relaxation time to reach the quasi-stationary state again.

Nevertheless, results for three 2D combination schemes with a recombination interval of two units of simulation time are presented in Table 2. The trajectories of

$\boldsymbol{\ell}_{\max}, \boldsymbol{\ell}_{\min}$	Reference	Recombination of g	Direct combination
(**10**,5,**5**,4,3), (**8**,5,**3**,4,3)	27.3 ± 0.6	28.4 ± 0.5 (4%)	27.4 ± 1.7 (0.3%)
(**11**,5,**5**,4,3), (**9**,5,**3**,4,3)	28.1 ± 1.1	24.9 ± 0.5 (11%)	27.9 ± 2.2 (0.7%)
(**10**,5,3,**6**,3), (**8**,5,3,**4**,3)	27.3 ± 1.6	27.9 ± 0.9 (2%)	28.4 ± 1.4 (4.0%)

Table 2: Results for frequent recombination of g. The values shown are the mean heat flux Q_{es} with relative error to the reference in parentheses. The alleged outlier is actually closer to the high resolution reference from Table 1.

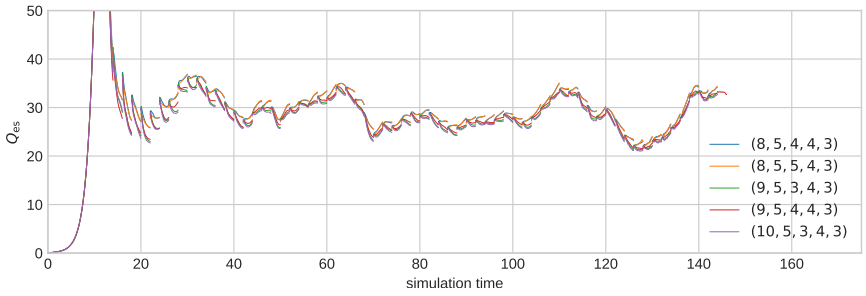

Fig. 11: Time traces of the mean heat flux Q_{es} for the five component grids of the first scheme in Table 2. One sees the effect of the recombination of g after every two units of time.

Q_{es} for the different component grids are shown in Fig. 11. They look promising since one can observe that after a combination step the values "snap" to a similar value. The remaining discrepancy is solely caused by the influence of different resolutions during the calculation of Q_{es}. There is no functionality yet to directly compute this quantity on the underlying sparse grid because its implementation would have been too time-consuming. Instead, the values at the combination times were gathered and again linearly combined according to the combination scheme. With this procedure we achieved results that are in accordance to their reference solution. Still, the same considerations as for the one-time combination apply here and we look forward to expanding the tests to more optimized index sets in the future.

4.2 Scaling Analysis

Scalability is a crucial aspect for an exascale-ready framework. For our framework this boils down to two aspects: achieving a good load balance to avoid idling process groups and keeping the overhead of recombination low as the main computation happens in the black box solver. We expect that the black box solver itself already provides good scalability up to a certain number of cores. By parallelizing over many tasks we can then boost this scalability to reach process counts beyond the capabilities of the solver. This is based on the fact that the number of cores can be scaled in two ways: increasing the number of cores in the process groups – scaling the solver – or increasing the number of process groups while keeping their core numbers equal.

We will first look at the scalability of the recombination steps [13] for a linear test case with $\boldsymbol{\ell}_{\max} = (14, 6, 6, 8, 7)$ and $\boldsymbol{\ell}_{\min} = (9, 4, 4, 6, 4)$ on Hazel Hen (HLRS). In Figure 12 we can see that the hierarchization scales almost perfectly with the number of cores as it does not involve computation between process groups. Moreover, it mainly involves local computation with little communication in the group. Dehierarchization performs almost identically as it is the inverse function of the hi-

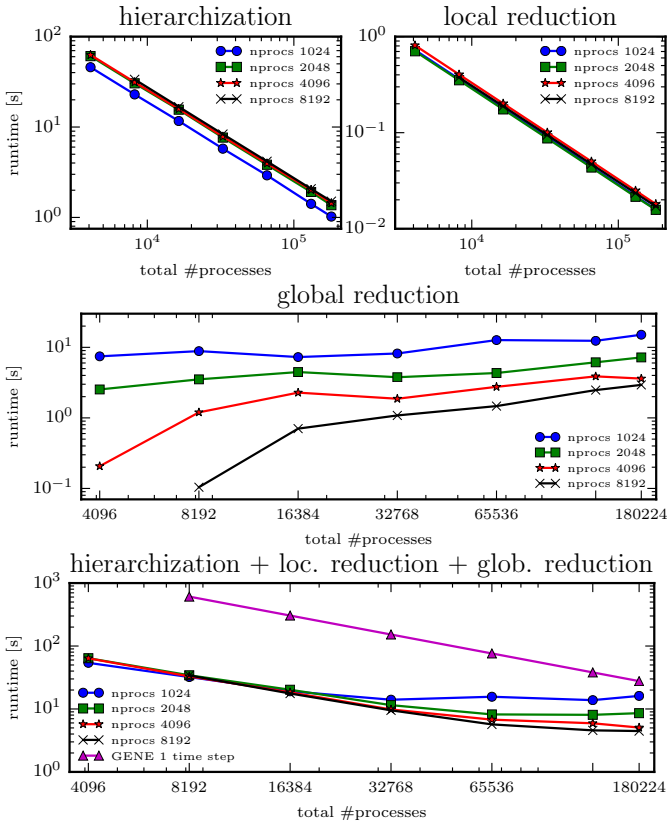

Fig. 12: Timings for the individual steps of the distributed recombination step for different sizes of the process groups. The bottom plot shows the total time of hierarchization, local reduction and global reduction. We also included a rough estimate of the computation time for one time step of GENE with process groups of size 4096. Graphs from [13]

.

erarchization. The local reduction step only adds the hierarchical coefficients of the local component grids to the sparse grid. As we enforce the same domain decomposition of the sparse grids and the component grids, this introduces no computation. Consequently, this operation scales perfectly. The global reduction of the sparse grid, however, involves the communication of the decomposed sparse grid parts over the process groups. Therefore, it shows no scaling behavior but also a comparably low runtime.

If we now sum up all components of our recombination step (bottom graph in Fig. 12), we can see that we can scale up to the whole size of the Hazel Hen supercomputer if the number of process groups is adjusted well. As a consequence, the framework introduces only a low overhead compared to the computation time of GENE itself.

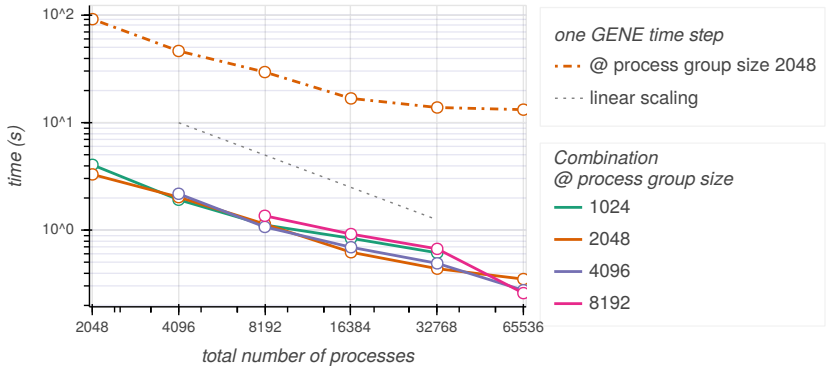

Fig. 13: Timings for a combination scheme computed with the nonlinear GENE functionality: ITG with adiabatic electrons, for $\boldsymbol{\ell}_{\min} = (7, 1, 5, 6, 4)$ to $\boldsymbol{\ell}_{\max} = (10, 1, 7, 8, 6)$, averaged over ten combination time steps on Hazel Hen.

This observation is confirmed by the measurements taken for nonlinear runs on Hazel Hen, shown in Fig. 13. This scenario consisted of 16 grids at relatively high resolutions. The optimal linear scaling is reached for moderate numbers of processes, yet the total scaling capabilities are limited by the solver's scaling capabilities. We see again that the sparse grid combination technique can play out its advantages only if the scenario is chosen to contain many grids, i.e., to have significant spans between $\boldsymbol{\ell}_{\min}$ and $\boldsymbol{\ell}_{\max}$. Even then, for our use cases, the overhead imposed by the combination routines is negligible compared to the total runtime.

4.2.1 Load Balancing

We will see here that, using the algorithmic opportunity for load balancing by grid assignment based on a good load model, cf. Section 3.2, we can achieve good parallel efficiency for large simulations – applied to both linear and nonlinear GENE runs.

The linear and anisotropy-based models were tested on 32 cores [14]. For this linear experiment, the scenario was constructed from $\boldsymbol{\ell}_{\min} = (3, 1, 3, 3, 3)$ to $\boldsymbol{\ell}_{\max} = (11, 1, 11, 11, 11)$, containing 425 grids. Fig. 14 shows that the initial dynamic approach ("work stealing") is preferable to the static assignment, because high parallel efficiencies can be sustained as the number of process groups grows. We can also conclude that taking the anisotropy into account leads to, on average, substantial improvements with respect to the parallel efficiencies.

In order to cope with the larger-scale nonlinear GENE simulations, a similar methodology was used to evaluate data-driven techniques. The methods compared were, again, the anisotropy-based model, nearest neighbor estimates, support vector regression (SVR) and neural networks [22]. These models were trained on 2048 randomly sampled tasks. As a reference, the best attainable balance that could be produced by the dynamic filling heuristic (cf. Section 3.2) – through an estimate

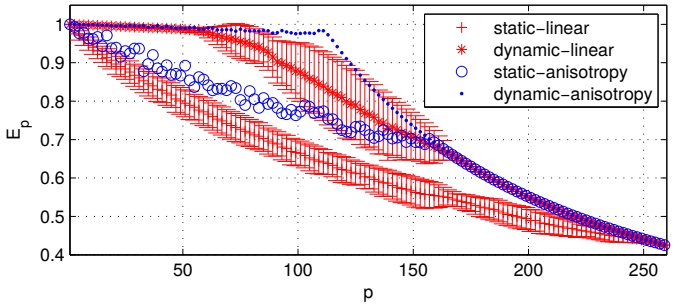

Fig. 14: Parallel efficiency for the linear ITG scenario: linear vs. anisotropy-based, static vs. linear.

that is the same as the true runtime – is included in the comparison. This time, the scenario contained grids from $\boldsymbol{\ell}_{\min} = (7, 4, 3, 4, 3)$ to $\boldsymbol{\ell}_{\max} = (12, 8, 6, 8, 6)$ to account for the anisotropy in the requirements for different solutions, comprising 237 grids. Now, the process group size was not fixed to 32 any more, but could reach up to 2^{15} or 32768 processes.

It is to be noted that Fig. 15 shows regions of high parallel efficiency compared to Fig. 14. Still, the differences between the different modeling methods are significant: while the nearest neighbor and SVR models lead to a relatively quick degradation of parallel efficiency, the expert knowledge-based anisotropy model performs a lot better, and the neural network can even achieve near-optimal scaling (the improvements

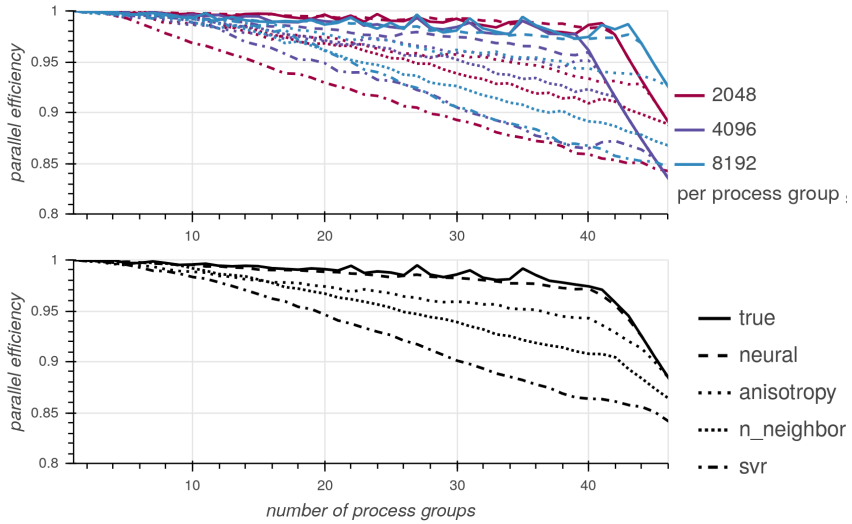

Fig. 15: Parallel efficiencies for the nonlinear, larger ITG scenario: by method and process group size (upper), and averaged over the different process group sizes for clarity (lower). The legend of the lower plot applies to the upper one in the same way.

over the optimal estimate are of course lucky guesses). This advantage is enabled by having enough data samples available. Since GENE is routinely run on various HPC machines, strong load models could be generated even across machines from the runtime data through neural networks.

4.3 Fault Tolerance

4.3.1 Fault Tolerant Combination Technique

This chapter briefly summarizes the most recent results with hard faults within the FTCT from [19]. For further results on hard and soft faults see [15, 20, 16].

To simulate statistical effects on the accuracy with faults in the FTCT, we conduct a random sampling for generating process failures during the simulation. For this sampling we choose the commonly used Weibull distribution [23]:

$$f(t;\lambda,k) = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1} e^{-(t/\lambda)^k} \tag{15}$$

where k and λ are the shape and scale parameter. λ is used to directly control the failure rate where smaller λ values cause larger error rates. This distribution is used to draw a failure time after which an MPI rank will fail. E.g., a failure time of 10s means that after a wall clock time of 10s a process will fail.

We show results for one linear test case with a 3D combination in z,μ and $v_\parallel$ with $\boldsymbol{\ell}_{\max} = (8,8,8)$ and $\boldsymbol{\ell}_{\min} = (5,5,5)$. The remaining dimensions x an y are fixed at 9 and 1 grid points, respectively. Statistical results for 100 individual runs with 512 MPI ranks are shown in Fig. 16 (left). It can be seen that for lower failure rates the error is almost identical to the base line error without any process failure (dotted line). In case of larger failure rates the accuracy decreases but the overall error

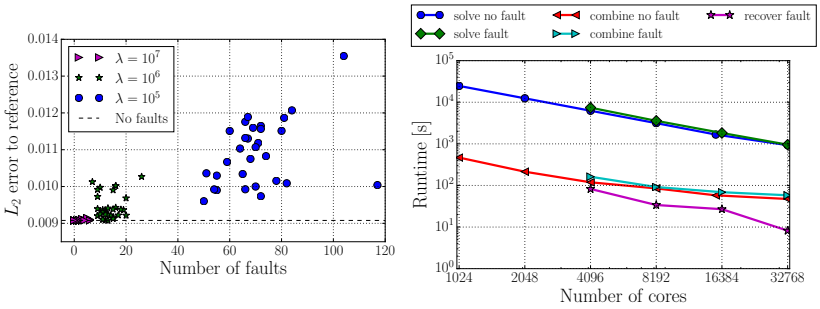

Fig. 16: **Left**: Statistical results for the L_2 error of the FTCT with different numbers of faults [19]. The errors stay relatively small, even in the presence of around 100 faults the error increases by less than a factor of two. **Right**: Average runtimes for our scaling run with the FTCT [19]. One can see that the overhead of the fault recovery is negligible compared to the runtime of the solver.

increase is still tolerable compared to the number of failing ranks. On average, the error increases by 0.09%, 4% and 20%, respectively, for lambda values of 10^7, 10^6 and 10^5.

Apart from error analysis, we are interested in the scaling behavior of the FTCT in the presence of faults. For this analysis we again choose a 3D combination in z,μ and $v_\parallel$ with $\boldsymbol{\ell}_{\max} = (14, 14, 14)$ and $\boldsymbol{\ell}_{\min} = (4, 4, 4)$. In this case we simulate only 300 time steps and apply 3 recombinations, i.e. every 100 steps, and inject a single fault in one of the process groups. The process group size stays fixed at 1024 cores and only the number of process groups is varied in the scaling experiment (Fig. 16, right).

In general we observed a very good scaling behavior even in the presence of faults. Of course, the runtimes are slightly increased due to the fault of one process group, which reduces the overall process count after the fault occurs. However, the scaling properties seem to be unaffected by the fault recovery. This can also be seen in the low overhead of the recovery step which is about 2 orders of magnitude lower than the time for solving the PDE. Also the time for the recombination scales well even though it involves global communication.

All in all, our results with linear GENE simulations have shown that we can achieve fault tolerance with small computational overhead while keeping a similar accuracy. For further details about the experiments and results we refer to [19].

4.3.2 libSpina

We answer here two questions concerning FT-GENE: how its performance compares with the performance of standard GENE and what kind of faults the framework is able to recognize and tackle. The clusters Hazel Hen, Cobra and Draco (MPCDF) were used in the tests which follow.

Robustness

These tests used realistic parameters (based on the benchmark ITGTEM) where a random subset of the MPI ranks is selected to fail. Several combinations of faults were tested (full node failure, partial node failure, multiple failures in different time steps, etc.) as well as different causes for failure (e.g. “stop” intrinsic, or externally calling the “pkill” command).

The library libSpina was able to detect the faults, notify FT-GENE, restore the MPI environment and allow FT-GENE to read the last checkpoint and rollback the computation. Not all fault types could be handled by libSpina, most notably faults caused by interrupting the interconnect hardware. Otherwise, the results were consistent and reproducible.

Nodes	μ	n_proc_μ	GENE	FT-GENE	Loss
2	2	1	379.2s	386.1s	1.83%
4	4	2	398.9s	400.7s	0.46%
8	8	4	419.6s	422.4s	0.67%
16	16	8	414.5s	416.2s	0.40%
32	32	16	424.4s	432.4s	1.89%
64	64	32	438.0s	451.4s	3.06%
150	5	5	568.4s	565.3s	-0.55%
300	10	10	590.4s	585.0s	-0.91%
600	20	20	714.6s	700.4s	-1.99%

Table 3: Cobra performance comparison. Topmost: (512,64,40,*,48) points, (8,1,5,*,1) parallelization, nonlocal, kinetic electrons (2 species). Lowermost: (512,512,20,*,60) grid size, (1,2,20,*,15) parallelization, local, kinetic electrons. Both are nonlinear runs with 100 time steps and 40 ranks per node.

Performance and Overhead

The aim of these tests was to determine the performance impact of libSpina on an application, by running both FT-GENE and GENE "back-to-back" on the same hardware with the same parameter files.

In the following, the grid dimensions are displayed as $(x, y, z, v_{||}, \mu)$. The same notation is used for the MPI partitioning of the domain (that is, n_proc_x, n_proc_y, and so on). The runtime is shown in seconds and disregards the initialization of GENE and libSpina. The reason is that the initialization of GENE involves several performance optimization tests (often lasting several minutes) which would mask the true difference in runtime.

The first part of the results in Table 3 have been repeated many times for many different configurations: nonlocal and local, linear and nonlinear, between 1 and 64 nodes, in Cobra and Draco with several different types of parallelization. On all occasions the result was consistent and the overhead of libSpina lay between -3% and $+3\%$ which is within statistical fluctuations in this case. For the sake of brevity, only the most recent of these results are shown in Table 3.

There was no deterioration of the performance of FT-GENE for large number of nodes, as it can be seen in the lowermost part of Table 3.

Nodes	x	n_proc_x	GENE	FT-GENE	Loss
1024	576	24	517s	446s	-13.7%
1024	576	24	469s	477s	1.7%
2048	1152	48	1156s	805s	-30.4%
2048	1152	48	836s	914s	9.3%

Table 4: Hazel Hen performance comparison. Grid size: (*,128,32,64,16) points, parallelization: (*,1,8,8,16), nonlocal, adiabatic electrons (1 species), nonlinear run, 1000 time steps, 24 ranks per node.

Similar experiments were performed on Hazel Hen but provide largely inconclusive results. Small tests (with 4 nodes) were analogous to the results shown in Table 3, but tests involving a large number of resources were inconclusive (see Table 4). The overhead fluctuated between -30% and $+15\%$. One of the possible explanations is that the job was distributed amongst several islands and MPI communications had to be passed through switches. Since switches are shared amongst all running jobs, the communication patterns of GENE and FT-GENE were disturbed by external factors, causing statistically invalid measurements of performance.

5 Conclusion

During the EXAHD project a massively parallel software framework for the sparse grid combination technique was developed and successfully applied to the gyrokinetic solver GENE, to linear as well as nonlinear, nonlocal settings. While its functionality has been demonstrated for medium sized test cases we expect an even greater benefit when applying it to larger simulations in future experiments.

Our implementation shows excellent scaling behavior up to 180k cores, and allows to scale the application code beyond its specific capabilities, thus making it ready for exascale computing. At the same time the overhead incurred to the total runtime is negligible.

The load balancing that can be realized with our approach allows to maintain parallel efficiencies close to the optimum. This is achieved by incorporating effects of anisotropy in our cost model and applying dynamic task scheduling. By modeling the solver loads with neural networks, this scheduling can be further enhanced. As an outlook, the additional level of parallelism offered by the combination technique may allow for scaling beyond a single system: The computation may be decoupled across compute centers. This would require more advanced distributed communication, as the combined solutions would have to be updated on both systems. First tests have been performed for this setup, but the overall performance and gains of such an approach still need to be evaluated.

We have also shown that the fault tolerant combination technique can be used to construct a resilient framework, which can tolerate hard and even soft faults. This algorithm-based fault tolerance introduces only minor effects on the overall accuracy and runtime while preserving the scaling properties of the framework. Furthermore, FT-GENE demonstrated a reasonable ability to tolerate faults, even under the restrictions of relying purely on standard MPI, causing no performance overhead.

Finally, by implementing specialized interfaces, the software framework can readily be extended to other codes dealing with high-dimensional functions and we look forward to seeing it used in many more applications.

References

[1] W. Bland, A. Bouteiller, T. Herault, G. Bosilca, and J. Dongarra. "Post-failure recovery of MPI communication capability: Design and rationale". In: *The International Journal of High Performance Computing Applications* 27.3 (2013), pp. 244–254.

[2] H.-J. Bungartz and M. Griebel. "Sparse Grids". In: *Acta Numerica* 13 (2004), pp. 147–269.

[3] H.-J. Bungartz, M. Griebel, D. Röschke, and C. Zenger. "Pointwise convergence of the combination technique for the Laplace equation". In: *East-West J. Numer. Math.* 2 (1994), pp. 21–45.

[4] T. Gerstner and M. Griebel. "Dimension–Adaptive Tensor–Product Quadrature". In: *Computing* 71.1 (2003), pp. 65–87.

[5] *GLPK (GNU Linear Programming Kit)*. URL: `https://www.gnu.org/software/glpk/`.

[6] T. Görler. "Multiscale effects in plasma microturbulence". PhD thesis. Universität Ulm, 2009.

[7] M. Griebel and J. Hamaekers. "Sparse Grids for the Schrödinger equation". In: *Mathematical Modelling and Numerical Analysis* 41.2 (2007), pp. 215–247.

[8] M. Griebel and H. Harbrecht. "On the convergence of the combination technique". In: *Sparse grids and Applications*. Vol. 97. Lecture Notes in Computational Science and Engineering. Springer, 2014, pp. 55–74.

[9] M. Griebel, W. Huber, U. Rüde, and T. Störtkuhl. "The combination technique for parallel sparse-grid-preconditioning or -solution of PDEs on workstation networks". In: *Parallel Processing: CONPAR 92 VAPP V*. Ed. by L. Bougé, M. Cosnard, Y. Robert, and D. Trystram. Vol. 634. LNCS. 1992.

[10] M. Griebel, M. Schneider, and C. Zenger. "A combination technique for the solution of sparse grid problems". In: *Iterative Methods in Linear Algebra*. Ed. by P. de Groen and R. Beauwens. IMACS, Elsevier, North Holland, 1992, pp. 263–281.

[11] B. Harding. "Fault Tolerant Computation of Hyperbolic Partial Differential Equations with the Sparse Grid CombinationTechnique". PhD thesis. Australian National University, 2016.

[12] B. Harding et al. "Fault Tolerant Computation with the Sparse Grid Combination Technique". In: *SIAM Journal on Scient. Comp.* 37.3 (2015), pp. C331–C353.

[13] M. Heene. "A Massively Parallel Combination Technique for the Solution of High-Dimensional PDEs". PhD thesis. Institut für Parallele und Verteilte Systeme der Universität Stuttgart, 2018.

[14] M. Heene, C. Kowitz, and D. Pflüger. "Load Balancing for Massively Parallel Computations with the Sparse Grid Combination Technique." In: *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*. Vol. 25. Advances in Parallel Computing. 2014, pp. 574–583.

[15] M. Heene, A. Parra Hinojosa, H.-J. Bungartz, and D. Pflüger. “A Massively-Parallel, Fault-Tolerant Solver for High-Dimensional PDEs”. In: Euro-Par. 2016.

[16] M. Heene, A. Parra Hinojosa, M. Obersteiner, H.-J. Bungartz, and D. Pflüger. “EXAHD: An Exa-Scalable Two-Level Sparse Grid Approach for Higher-Dimensional Problems in Plasma Physics and Beyond”. In: *High Performance Computing in Science and Engineering ’ 17*. Ed. by W. Nagel, D. Kröner, and M. Resch. Springer-Verlag, 2018.

[17] F. Jenko, W. Dorland, M. Kotschenreuther, and B. Rogers. “Electron temperature gradient driven turbulence”. In: *Physics of Plasmas* 7.5 (2000), pp. 1904–1910.

[18] C. Kowitz. “Applying the Sparse Grid Combination Technique in Linear Gyrokinetics”. Dissertation. München: Technische Universität München, 2016.

[19] M. Obersteiner, A. Parra Hinojosa, M. Heene, H.-J. Bungartz, and D. Pflüger. “A Highly Scalable, Algorithm-Based Fault-Tolerant Solver for Gyrokinetic Plasma Simulations”. In: *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*. 2017.

[20] A. Parra Hinojosa, B. Harding, M. Hegland, and H.-J. Bungartz. “Handling silent data corruption with the sparse grid combination technique”. In: *Software for Exascale Computing-SPPEXA 2013-2015*. Springer, 2016, pp. 187–208.

[21] B. Peherstorfer, C. Kowitz, D. Pflüger, and H.-J. Bungartz. “Selected Recent Applications of Sparse Grids”. In: *Numerical Mathematics: Theory, Methods and Applications* 8.1 (2015), pp. 47–77.

[22] T. Pollinger and D. Pflüger. “Learning-Based Load Balancing for Massively Parallel Simulations of Hot Fusion Plasmas”. In: Advances in Parallel Computing. Submitted.

[23] B. Schroeder and G. Gibson. “A Large-Scale Study of Failures in High-Performance Computing Systems”. In: *IEEE Transactions on Dependable and Secure Computing* 7.4 (2010), pp. 337–350.

[24] S. Smolyak. “Quadrature and interpolation formulas for tensor products of certain class of functions”. In: *Soviet Mathematics Doklady* 4 (1963), pp. 240–243.

[25] D. Told, J. Cookmeyer, F. Muller, P. Astfalk, and F. Jenko. “Comparative study of gyrokinetic, hybrid-kinetic and fully kinetic wave physics for space plasmas”. In: *New Journal of Physics* 18.6 (2016), p. 065011.